Recovery-Driven Design: Exploiting Error Resilience in Design of Energy-Efficient Processors

Andrew B. Kahng, *Fellow, IEEE,* Seokhyeong Kang, *Student Member, IEEE,* Rakesh Kumar, *Member, IEEE,* and John Sartori, *Student Member, IEEE*

***Abstract*—Conventional computer-aided design (CAD) methodologies optimize a processor module for correct operation and prohibit timing violations during nominal operation. We propose *recovery-driven design*, a design approach that optimizes a processor module for a target timing error rate (ER) instead of correct operation. The target ER is chosen based on how many errors can be gainfully tolerated by a hardware or software error resilience mechanism. We show that significant power benefits are possible from a recovery-driven design approach that deliberately allows errors caused by voltage overscaling to occur during nominal operation, while relying on an error resilience technique to tolerate these errors. We present a detailed evaluation and analysis of such a CAD methodology that minimizes the power of a processor module for a target ER. We show how this design-level methodology can be extended to design *recovery-driven processors*—processors that are optimized to take advantage of hardware or software error resilience. We also discuss a gradual slack recovery-driven design approach that optimizes for a range of ERs to create *soft processors*—processors that have graceful failure characteristics and the ability to trade throughput or output quality for additional energy savings over a range of ERs. We demonstrate significant power benefits over conventional design—11.8% on average over all modules and ER targets, and up to 29.1% for individual modules. Processor-level benefits were 19.0%, on average. Benefits increase when recovery-driven design is coupled with an error resilience mechanism or when the number of available voltage domains increases.**

***Index Terms*—Cell sizing, error resilience, power minimization, recovery-driven design, slack redistribution.**

I. Introduction

CONVENTIONAL hardware is designed and optimized using techniques that aim to ensure correct operation of the hardware under different conditions. Conservative design techniques are aimed at ensuring correct hardware operation under worst-case conditions. Better-than-worst-case design techniques [1] save power by eliminating guardbands, but are still aimed at ensuring correct hardware operation under nominal conditions.

In this paper, we ask the following question: *should the availability of an error resilience mechanism change the way we approach hardware design and optimization?* That is, given that mechanisms exist to tolerate hardware errors, should hardware continue to be designed for correct operation or should it be optimized for a target error rate (ER) even during nominal operation. To address this question, we propose and evaluate a novel approach to hardware design, called recovery-driven design. Rather than optimizing for correct operation, a recovery-driven design deliberately allows timing errors [7], [15] to occur during nominal operation, while relying on an error resilience mechanism to tolerate these errors. In other words, a recovery-driven design optimizes a circuit for a nonzero target ER that can be gainfully tolerated by hardware [7] or software-based [15] error resilience. The expectation behind recovery-driven design is that the "underdesigned" hardware will have significantly lower power or higher performance than hardware optimized for correct operation. Also, because errors are now allowed, the design methodology can exploit workload-specific information (e.g., activity of timing paths, architecture-level criticality of timing errors) to further maximize the power/performance benefits of underdesign.

In this paper, we show that optimizing power for a target timing ER for voltage overscaling-induced errors indeed results in significant power savings for similar levels of performance. We show that this is true when errors are detected and corrected by a hardware error tolerance mechanism [7] or allowed to propagate to an error-tolerant application [3] where the errors manifest themselves as reduced performance or output quality [15]. Increasing the target ER for a processor module increases the potential for power savings, since the module can be operated at a lower voltage. In practice, the target ER is chosen such that an error recovery mechanism can correct the resulting errors and still reduce energy (after considering the error recovery overhead) for an acceptable degradation in performance or output quality. The power benefits of exploiting error resilience are maximized by redistributing timing slack from paths that cause very few errors to frequently exercised paths that have the potential to cause many errors. This reduces the ER at a given voltage, and hence reduces the minimum supply voltage and power for a target ER.

This paper presents a detailed evaluation and analysis of a slack redistribution-based recovery-driven design methodology

Manuscript received February 25, 2011; revised July 16, 2011; accepted September 14, 2011. Date of current version February 17, 2012. This paper was recommended by Associate Editor D. Sylvester.

A. B. Kahng is with the Department of Computer Science and Engineering and the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: abk@ucsd.edu).

S. Kang is with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: shkang@vlsicad.ucsd.edu).

R. Kumar and J. Sartori are with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: rakeshk@illinois.edu; sartori2@illinois.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2172610

that minimizes the power of a processor module for a target ER. Our cell sizing-based design-level methodology has been extended to create recovery-driven processors that are optimized for different target ERs or error-resilience mechanisms. Since some error resilience mechanisms (e.g., error-tolerant applications) require adaptation to multiple reliability targets, we have also extended our recovery-driven design approach to create gradual slack designs—designs that are optimized not for a single ER, but instead, for a range of ERs. Such gradual slack designs (or *soft processors*) have the ability to trade performance or output quality for energy savings over a range of reliability targets. We make the following contributions in this paper.

1) To the best of our knowledge, we present the first design flow for power minimization that deliberately allows errors under nominal conditions. We demonstrate that such a design flow can result in power savings of 11.8%, on average over all modules and ER targets, and up to 29.1% for individual modules.
2) We explore the heuristic choices and tradeoffs that are fundamental to the optimization quality of slack redistribution-based, recovery-driven designs. We evaluate choices for path priority and traversal during optimization, optimization radius, accuracy of path selection, error budget utilization, starting netlist, voltage step size granularity, and iterative optimization in terms of their effects on the optimization result, heuristic runtime, and sensitivity to target ER.
3) To support the proposed recovery-driven design flow, we present a fast and accurate technique for postlayout activity and ER estimation. We use collected functional information to redistribute slack efficiently in a circuit and significantly extend the range of voltage scaling for a target ER.
4) We extend our recovery-driven design methodology to create recovery-driven processors (processors that are optimized for different target ERs or error recovery mechanisms) and soft processors (processors that are optimized for efficiency over a range of target ERs). We demonstrate the power and energy benefits of such processor designs.
5) We demonstrate that the power benefits of recovery-driven processors and soft processors increase when a hardware or software-based error resilience mechanism is used. We consider Razor [7] and application-level noise tolerance [32] as examples and show additional energy reductions of 19% and 20% with respect to the best correctness-optimized processors that exploit the same error resilience mechanisms.

II. Related Work

A. Design-Level Optimization

Previous design-level optimizations for error-tolerant designs [9], [10] identify and optimize critical paths that are frequently exercised during operation. BlueShift [10] identifies the most frequently violated timing paths during gate-level simulation, and optimizes the paths iteratively until the ER is below the target. BlueShift uses two methods to add slack to the frequently exercised paths—forward body biasing of selected gates and application of tighter timing constraints to the frequently exercised paths.

Our work differs from BlueShift in objective, approach, and scope of optimization. Our objective is to minimize power, while BlueShift's objective is to improve performance. Consequently, we use sensitivity functions that are voltage-aware. Also, BlueShift requires iterative gate-level simulation and re-layout, making the approach time-consuming and impractical for large system-on-chip (SoC) designs. Furthermore, while BlueShift optimizes only the postsynthesis circuit over many layout iterations, our recovery-driven design techniques perform both activity-guided postsynthesis and postlayout optimizations in a single pass to enhance energy efficiency.

CRISTA [9] isolates critical paths with Shannon-expansion-based partitioning. After partitioning, CRISTA downsizes cells on the critical path and upsizes cells on the noncritical paths: critical paths are made slower while noncritical paths are made faster. When a critical path is excited, the corresponding operation takes two cycles. CRISTA changes the structure of the original circuit and also requires circuit-specific design to isolate critical paths. Since we do not change the original circuit structure, our techniques are more general in nature and can be applied more easily to a wider range of circuits.

Like recovery-driven design, related work on better than worst case (BTWC) logic synthesis [6] has also proposed to use activity information to reduce the ER of an overscaled design. Whereas traditional synthesis tools attempt to minimize delay for a logic block, the proposed BTWC synthesis tool uses switching probability to break a tie when two equivalent logic decompositions have the same delay. Reducing switching activity can result in fewer errors for an overscaled design.

B. Sensitivity-Based Cell Sizing

Our methodology relies on cell sizing for slack distribution. Sensitivity-based downsizing approaches have been proposed in [8], [11]–[13], [28], and [29]. TILOS [8] proposes a heuristic that sizes transistors iteratively, according to the sensitivity of the critical path delay to the transistor sizes, in order to find an optimum (with maximum delay reduction/transistor width increase). Equation (1) shows the sensitivity function of TILOS. ΔL and ΔD represent the change in leakage and delay for a resized transistor. The techniques proposed in [29] use the same sensitivity function as TILOS as follows:

$$\text{Sensitivity} = \Delta L / \Delta D. \tag{1}$$

For the cell sizing in [12], all cells are sorted in decreasing order of $\Delta L \times S$, where ΔL is the improvement in leakage after a cell is replaced with its less leaky variant, and S is its timing slack after the replacement has been made. The techniques proposed in [11] and [13] use sensitivity-based downsizing (i.e., begin with all nominal cell variants and replace cells on noncritical paths with long channel-length variants) heuristics for leakage optimization. In their heuristics, they defined the sensitivity associated with cell instance as follows:

$$\text{Sensitivity} = \Delta L / \Delta S. \tag{2}$$

In (2), ΔS represents the slack change of a given cell instance after downsizing. ΔL indicates the leakage change of cell instance after downsizing. The sensitivities are computed for all

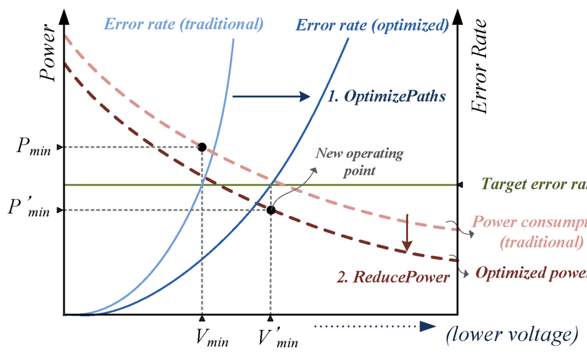


Fig. 1. Our recovery-driven design optimization redistributes slack from infrequently exercised paths to frequently exercised paths and performs cell downsizing for average-case conditions. These optimizations reduce the power consumption of a circuit and extend the range that voltage can be scaled before a target ER is exceeded. The combination of these factors produces a design with significantly reduced power consumption.

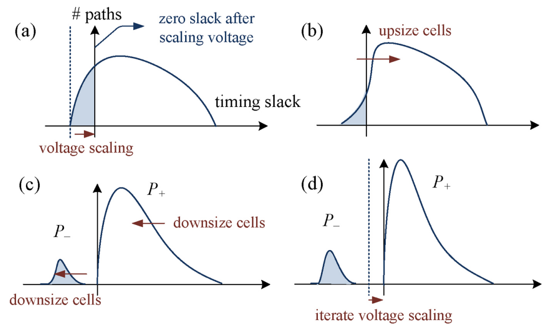


Fig. 2. Power minimization heuristic reshapes the path slack distribution by redistributing slack from paths that rarely toggle to paths that toggle frequently.

cell instances. The heuristics of [11] and [13] select a cell with the largest sensitivity and perform downsizing with a logically equivalent cell. If there is no timing violation in incremental static timing analysis, this move is accepted and saved.

Our work uses cell sizing in a novel context—as a mechanism to optimize hardware for nonzero ERs.

III. Heuristic Design

A. Motivation

The goal of recovery-driven design in context of voltage overscaling can be stated formally as follows. Given an initial netlist N_0, a set of cell libraries characterized for allowable operating voltages, toggle rates (TRs) for the toggled paths in the netlist, and a target ER $\text{ER}_{\text{target}}$, produce the optimized netlist $N_{V_{\text{opt}}}$ and operating voltage V_{opt} that minimize the total power consumption $W_{V_{\text{opt}}}$ of the circuit, such that the ER of the optimized netlist does not exceed $\text{ER}_{\text{target}}$. Fig. 1 demonstrates the goal.

In this paper, we present a cell sizing-based design methodology that relies on efficient redistribution of timing slack from infrequently exercised critical paths to frequently exercised paths to reduce the ER at a given voltage, allowing a reduction in voltage for a given target ER.

B. Abstract Heuristic for Power Minimization

Our heuristic for slack redistribution-based power minimization uses a two-pronged approach—extended voltage scaling through cell upsizing on critical and frequently

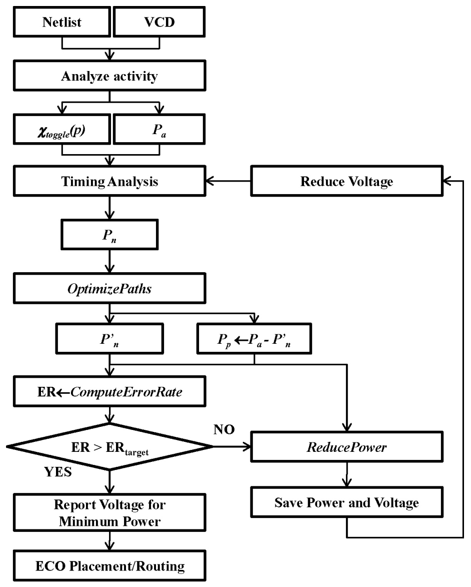


Fig. 3. Algorithmic flow of a heuristic for minimizing power for a target ER. P_a is the set of all paths toggled during simulation. P_p is the set of all nonnegative slack paths. P_n is the set of all negative slack paths in P_a. $\chi_{\text{toggle}}(p)$ is the set of cycles in which path p is toggled.

exercised circuit paths (*OptimizePaths*), and leakage power reduction achieved by downsizing cells in noncritical and infrequently exercised paths (*ReducePower*). The heuristic searches for the combination of the two techniques that results in the lowest total power consumption for the circuit, by performing path optimization and power reduction at each voltage step and then choosing the operating power at which minimum power is observed.

Fig. 2 illustrates the evolution of the circuit path slack distribution throughout the stages of the power minimization procedure. Each iteration begins as voltage is scaled down by one step [Fig. 2(a)]. After partitioning the paths into sets containing positive and negative slack paths, *OptimizePaths* attempts to reduce the ER by increasing timing slack on negative slack paths [Fig. 2(b)]. Next, the heuristic allocates the ER budget by selecting paths to be added to the set of negative slack paths, and downsizes cells to achieve area/power reduction [Fig. 2(c)]. This cycle is repeated over the range of voltages to find the minimum power netlist and corresponding voltage [Fig. 2(d)]. In Fig. 2, P_+ is a set of paths that must have nonnegative slack after power reduction, and P_- is a set of paths that are allowed to have negative slack. We ensure positive slack for P_+ paths by characterizing timing with worst case libraries.

Fig. 3 presents the algorithmic flow of our power minimization heuristic, which couples path optimization to extend the range of voltage scaling (*OptimizePaths*) with area minimization to achieve power reduction (*ReducePower*).

C. Heuristic Procedures

1) *Path Optimization:* The goal of the path optimization procedure (*OptimizePaths*) presented in Algorithm 1 is to minimize the ER at a voltage level by transforming negative slack paths into nonnegative slack paths. This is accomplished by performing cell swaps within the negative slack paths to increase path slack. Negative slack paths with maximum TRs

Algorithm 1 Pseudocode (*OptimizePaths*, *ReducePower*).

```
Procedure OptimizePaths($P$, $N_{V_i}$, $V_i$)
 1. Clear 'visited' mark in all cells in the netlist $N_{V_i}$;
 2. while $P \neq \emptyset$ do
 3.     Select path $p$ from $P$ with maximum TR;
 4.     for each cell $c$ in path $p$ do
 5.         if $c.visited$ == true then continue;
 6.         $c.visited \leftarrow$ true;
 7.         for each logically equivalent cell $m$ for the cell instance $c$ do
 8.             Resize cell $c$ with logically equivalent cell $m$;
 9.             $Q \leftarrow c \cup$ visited fanin and fanout cells of $c$;
10.             for each path $q$ in $P$ that contains a cell in $Q$ do
11.                 if $\Delta slack(q, c, m, V_i) < 0$ then restore cell change;
12.             end for
13.         end for
14.     end for
15.     $P \leftarrow P - p$;
16. end while

Procedure ReducePower($P_p$, $P_n$, $N_{V_i}$, $V_i$, $ER_{target}$)
 1. $P_+ \leftarrow P_p$ and $P_- \leftarrow P_n$;
 2. while $P_+ \neq \emptyset$ do
 3.     Select path p from $P_+$ with minimum $\Delta ER(p)$;
 4.     $ER \leftarrow ComputeErrorRate(P_- + p)$;
 5.     if $ER \leq ER_{target}$ then
 6.         $P_- \leftarrow P_- + p$;    $P_+ \leftarrow P_+ - p$;
 7.     else
 8.         break;
 9.     end if
10. end while
11. Insert all downsizable cells into set $C$;
12. $ComputeSensitivity(C, N_{V_i}, V_i, -1)$;
13. while $C \neq \emptyset$ do
14.     Downsize cell $c$ from $C$ with minimum $Sensitivity(c)$;
15.     $Q \leftarrow c \cup$ fanin and fanout cells of $c$;
16.     for each path $p$ in $P_+$ that contains a cell in $Q$ do
17.         if $slack(p, V_i) < 0$ then
18.             Restore cell change;
19.             $C \leftarrow C - c$;
20.             continue while loop;
21.         end if
22.     end for
23.     $ComputeSensitivity(Q, N_{V_i}, V_i, -1)$;
24.     if cell $c$ is not downsizable then
25.         $C \leftarrow C - c$;
26.     end if
27. end while
```

are selected first during optimization, since they have the most potential to reduce the ER if converted into nonnegative slack paths.

When a path is targeted for optimization, cell swaps are attempted on all cells in the path to increase slack as much as possible until nonnegative path slack is achieved.[1] Once a cell has been visited during optimization, it is marked to prevent degradation of timing slack on any path that the cell is on. Before accepting a cell swap, path slack is checked for all paths that the cell or any visited fanin/fanout cell is on. If the swap caused a decrease in slack for any such path, the move is rejected, and the original cell is restored. Previously optimized (visited) fanin and fanout cells are protected from slack decrease because they belong to paths that have higher TRs, and thus, higher priority of optimization. If cell swaps on a path fail to shift the path back into the set of nonnegative slack paths, then the path is ignored during subsequent iterations of path optimization.

[1]We consider only setup timing slack, since hold violations can typically be fixed by inserting hold buffers in a later step.

Any cell swap that increases the ER (by causing a path to switch from the set of nonnegative slack paths to the set of paths allowed to have negative slack) is rejected. Otherwise, we recompute the sensitivity of the swapped cell and all cells in its fanin/fanout network and select the next cell for downsizing.

2) *Power Reduction:* After path optimization, the ER of the circuit is minimized at the present voltage. From this state, we proceed to minimize the power at the present voltage by utilizing the available ER budget. Algorithm 1 (*Reduce-Power*) describes our power reduction procedure. The goal of the power reduction heuristic is to efficiently allocate the remaining error budget to infrequently exercised paths in order to maximize power reduction achieved by cell downsizing. Typically, cells on P_- paths can exploit additional downsizing, because these paths are not bound by the normal timing constraint of the circuit.

The first step in power reduction is to choose additional paths to become negative slack paths until the target ER of the circuit is matched. Paths are selected in order to minimize the additional contribution to the ER of the circuit. After defining the partition between negative and nonnegative slack paths, cell downsizing is performed for all cells in the circuit in order of minimum sensitivity. We define the sensitivity of a cell in (3) as the change in cell slack (Δs_c) divided by the change in cell power (Δw_c) when the cell c is downsized by one size. The slack of cell c is defined as the minimum slack on any timing arc containing c. The power of cell c is the sum of static power ($w_{\text{stat}}(c)$) and dynamic power ($w_{\text{dyn}}(c)$) for the cell. This formulation of sensitivity is similar to those proposed by previous works targeting leakage power reduction [11], [13] as follows:

$$\text{Sensitivity}(c) = \frac{s_c - s_{c'}}{w_c - w_{c'}} \tag{3}$$

where

$$w_c = w_{stat}(c) + w_{dyn}(c).$$

D. Path Extraction and ER Estimation

1) *Path Extraction:* Our heuristic has many path-based procedures—*OptimizePaths*, *ReducePower*, and *ComputeErrorRate*—and it is impractical to consider all of the topological paths in these procedures. Therefore, we reduce the number of paths that we consider by extracting only paths toggled during functional simulation. The value change dump (VCD) file can be used to extract toggled paths. To produce a VCD file, we perform gate-level simulation with Cadence NC-Verilog [35] at a frequency slow enough to capture all possible signal transitions. Fig. 4 shows an example VCD file and the path extraction method. The VCD file contains a list of toggled nets at each time when a transition occurs, as well as their new values. We can use this information to extract toggled paths in each cycle. Nets that glitched or toggled in each cycle are marked, and these nets are traversed to find toggled paths. We detect a toggled path when toggled nets compose a connected path of toggled cells from a primary input or flip-flop input to a primary output or flip-flop output. In Fig. 4, nets a, x, and y have toggled in the first and third cycles (#1, #3), and nets b and y have toggled in the second and fourth cycles (#2, #4). We extract two paths: $a - x - y$ and $b - y$.

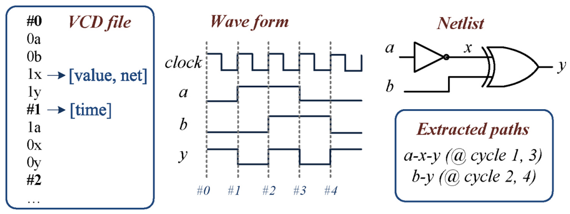


Fig. 4. VCD file format and path extraction.

2) *TR and ER Estimation:* In order to accurately minimize power for a target ER, we must be able to produce accurate estimates for ER during our optimization flow. Thus, we propose a novel approach to ER estimation that enables design for a target ER.

We calculate the TR of an extracted path using the number of cycles in which the path toggles. $\chi_{\text{toggle}}(p)$ represents the set of cycles in which path p has toggled during the simulation. TR(p) represents the TR of path p and is defined as

$$\text{TR}(p) = \frac{|\chi_{\text{toggle}}(p)|}{X_{\text{tot}}} \tag{4}$$

where $|\chi_{\text{toggle}}(p)|$ is the number of cycles in which path p has toggled, and X_{tot} is the total number of cycles in the simulation. Using the toggled cycle information of negative slack paths, we can calculate the ER precisely. The ER of the design is calculated as

$$\text{ER} = \frac{|\bigcup_{p\in P_n} \chi_{\text{toggle}}(p)|}{X_{\text{tot}}} \tag{5}$$

where P_n is the set of negative slack paths in the set of all toggled paths. In Fig. 4, if paths $a - x - y$ and $b - y$ both have a TR of 0.4 (number of toggled cycles is 2 and number of total cycles is 5), and if path $a - x - y$ has negative slack, then timing errors will occur in cycles #1 and #3. Therefore, the ER is 0.4 for this example.

Our novel technique for ER estimation has proven to be much faster than functional simulation and more accurate than previous estimation techniques. Results comparing our VCD-based technique to functional simulation and previous estimation approaches can be found in [19].

E. Heuristic Design Choices

In this section, we discuss heuristic design choices.

1) *Experiment 1: Path Ordering During Optimization:* The order in which we select paths for optimization affects the optimization result, since we prevent cells from being visited multiple times during optimization. The order also matters because we protect previously optimized paths from slack degradation due to other attempted cell swaps, as previously optimized paths have a higher optimization priority. We evaluate two prioritization functions for path selection during optimization. The first ranks paths in order of decreasing TR [TR(p)]. Paths with the highest TRs have the greatest potential to decrease ER when optimized. We compare against a function that ranks paths in order of decreasing TR(p)/|slack(p)|. In this alternative, we prefer paths with smaller negative slack, since the least effort is required to convert these paths into nonnegative slack paths.

2) *Experiment 2: Optimization Radius:* The goal of optimization is to maximize the slack of a targeted path through cell swaps. We evaluate two alternatives for the radius of optimization. In one case, we only swap cells on the target path. In the second case, we target both the cells on the path as well as cells in their fanin/fanout networks, since swaps in the fanin/fanout network can also affect cell slack.

3) *Experiment 3: Path Traversal During Optimization:* When optimizing a path, the order in which cells are visited can have an effect on the optimization result, since cell swaps affect input slew and output load. We consider two options—traversal from front to back and from back to front. We iterate over the cells in a path and make swaps until there is no further increase in the path slack.

4) *Experiment 4: Accuracy of Path Selection During Power Reduction:* During power reduction, nonnegative slack paths are selected to be added to the set of paths allowed to have negative slack, thus utilizing the available ER budget. Paths are prioritized in order of increasing incremental contribution to ER, ΔER(p). However, after moving a path from P_+ to P_-, ΔER(p) can change for paths that shared error cycles with the moved path.

To obtain precise ordering in terms of ER contribution, we can update ΔER(p) after each path selection. However, this introduces a runtime overhead, since we must continuously update ΔER(p) for all remaining P_+ paths. We compare such precise prioritization against the alternative case where ΔER(p) is calculated only once for all P_+ paths before path partitioning.

5) *Experiment 5: ER Budget Utilization:* During power reduction, the final ER after cell downsizing could be less than the target ER, $\text{ER}_{\text{target}}$, since some paths in P_- might still have nonnegative slack, even after maximum downsizing on the path cells. In this case, we might continue to reduce the power of the design by selecting more paths to add to P_- and downsizing cells again. We evaluate two cases—one where a single pass is performed for path selection and cell downsizing, and one where the *ReducePower* procedure is repeated until there is no further reduction in power (i.e., repeat *ReducePower* whenever some paths added to P_- still have nonnegative slack after cell downsizing).

6) *Experiment 6: Starting Netlist:* Here, we evaluate heuristic performance for different starting netlists corresponding to loose (clock period increased by 10%) and tight (reduced by 40%) timing constraints. This can significantly affect the final voltage reached, the dependence on engineering change order (ECO), and the amount of power savings afforded by the power minimization algorithm.

7) *Experiment 7: Voltage Step Size:* In each iteration of the power minimization heuristic, we step down the voltage by a value V_{step} and run the *OptimizePaths* and *ReducePower* procedures to produce a netlist for the present level of voltage scaling. The size of V_{step} can influence the optimization result and runtime of the heuristic. Thus, we compare two values of V_{step}, 0.01 V and 0.05 V, and compare the characteristics of the final netlist as well as the heuristic runtime.

8) *Experiment 8: Iterative Optimization:* In each iteration of the heuristic, we perform optimization of negative slack

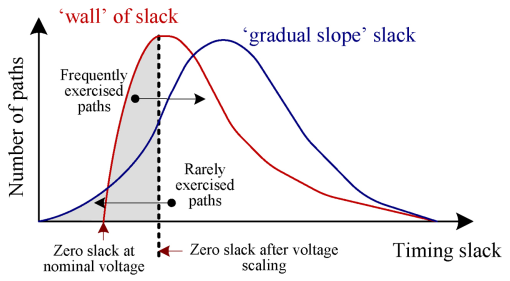


Fig. 5. Goal of the "gradual slope" slack optimization is to transform a slack distribution having a critical "wall" into one with a more gradual failure characteristic. This allows performance/power tradeoffs over a range of ERs.

paths at that voltage level. During the next iteration, we have a choice between starting from the previously optimized netlist, ($N_{V_{i-1}}$) or the original netlist (N_0). We compare the netlists produced in each case and see if they have similar power and runtime characteristics.

F. Gradual Slack Design

We extend our design methodology to implement another form of recovery-driven design called gradual slack design [17], which reshapes the slack distribution of a processor to create a gradual failure characteristic, rather than the typical critical wall. While ER-optimized, recovery-driven designs achieve better energy efficiency at a single target ER, gradual slack designs have the ability to trade reliability, throughput, or output quality for energy savings over a range of ERs. Fig. 5 describes the optimization approach for gradual slack design. To achieve a gradual slack distribution with our recovery-driven design flow, we do not optimize for a single target ER by selecting P_{-} paths. Instead, we select the maximum target ER corresponding to the desired range of scalability, and optimize only the negative slack paths in the scaling range with the highest switching activity, in order to maximize the range of voltage scalability for target range of ERs. We downsize only cells that have negligible activity so that the slack distribution for the active paths and the ER of the processor are not affected. In this way, we maintain the desired gradual sloping slack distribution rather than creating a critical wall distribution with a cluster of active paths in the permanent negative slack region.

G. Processor Power Reduction

Algorithm 2 shows a heuristic for minimizing the power of a processor core for a target ER. The first step of the above power-minimization heuristic involves characterizing the modules of the processor core in terms of their power consumption at different ER and voltage targets. These data are provided by *PowerOptimizer* and are used to select the optimal operating voltage(s) for the processor core, as well as the ER targets to assign to the processor modules.

The next step in the processor-level heuristic is to use the data from *PowerOptimizer* to solve an optimization problem. The optimization objective is to minimize the power of the processor core subject to the constraint that the processor ER must be less than the chosen target rate. Using the data from *PowerOptimizer*, we can formulate expressions for the power and ER of the processor core in terms of the module ERs

Algorithm 2 Processor-level design heuristic

```
Procedure OptimizeProcessor($\mathrm{ER}_{\mathrm{target}}$, $MODULES$, $DOMAINS$)
1.  for each module $m$ in the optimization list of MODULES do
2.      for each error rate ER $<$ $\mathrm{ER}_{\mathrm{target}}$ do
3.          PowerOptimizer($N(m)$, ER);
4.      end for
5.      Use the results from PowerOptimizer to characterize $P_m(V, \mathrm{ER})$
6.  end for
7.  for each voltage $V \in V_{\mathrm{range}}$ do
8.      Minimize $P_{\mathrm{core}}(V) = \Sigma(P_m(V, \mathrm{ER}))$ s.t.
        $\mathrm{ER}_{\mathrm{core}}(\mathrm{ER}_{\mathrm{module}_1}, \ldots, \mathrm{ER}_{\mathrm{module}_M}) \leq \mathrm{ER}_{\mathrm{target}}$
9.      Record minimum power $P^{\min}_{\mathrm{core}}(V)$ and module ER assignment $S(V) =$
        $[\mathrm{ER}_{\mathrm{module}_1}, \ldots, \mathrm{ER}_{\mathrm{module}_M}]$
10. end for
11. Select the voltage $V_{\mathrm{opt}}$ at which power $P^{\min}_{\mathrm{core}}$ is minimized
12. Let $V^*(S(V)[m])$ be the voltage that minimizes power for module $m$ at
    ER = $S(V)[m]$
13. Locate the $DOMAINS$ neighbors $\{V_1, \ldots, V_{DOMAINS}\}$ nearest to the set
    of voltages $V^*(S(V_{\mathrm{opt}}))$
14. Assign each module $m$ to the voltage domain $V_D[m] \in$
    $\{V_1, \ldots, V_{DOMAINS}\}$ that minimizes power $P_m(V_D[m], S(V_{\mathrm{opt}})[m])$
15. Layout the processor, selecting for each module $m \in$ MODULES the
    netlist $N(m, V_D[m], S(V_{\mathrm{opt}})[m])$;
```

and the operating voltage. Thus, the goal of the optimization problem for a particular voltage is to find the assignment of ER targets to modules that satisfies the optimization objective. In this paper, we use a disjunctively constrained knapsack-based [33] approach to solve the optimization problem. The knapsack solver selects the voltage and ER assignment for which the power of the processor core is minimized and uses the selected ER-optimized netlist of each module to lay out the processor.

For multiple voltage domain designs ($DOMAINS > 1$), the heuristic selects the voltage level of each domain and the partitioning of modules to voltage domains to minimize core power. This involves first selecting the ER targets for the modules based on a minimum-power global assignment, then selecting the levels for the voltage domains and module-to-level assignments such that the power of the modules is minimized. The latter step is performed using a nearest neighbor search to identify the neighbors nearest to the set of optimal module voltages corresponding to the module ER assignments in the space of voltages.

IV. Recovery-Driven Processors

The proposed design methodology enables recovery-driven processors—processors that are optimized to deliberately produce timing errors at a rate that can be gainfully tolerated by an error recovery mechanism. Below, we describe two recovery-driven processor designs—one targeting hardware-based error resilience and another targeting software-based error resilience.

A. Case Study: Circuit-Level Timing Speculation

One popular hardware-based scheme for error detection and correction is circuit-level timing speculation [7], [30]. Circuit-level timing speculation-based techniques detect errors by sampling the same computation twice—once using the regular clock and again using a delayed clock. The two outputs are compared. When the outputs do not match, an error is signaled. Correction involves treating the delayed clock output as the correct output. Razor [7] and error-detection sequential [30] provide good examples of circuit-level timing speculation.

A recovery-driven processor design targeted for Razor takes into account the frequency of errors that can be gainfully tolerated by Razor (determined by the dynamic error recovery overhead) as well as the number of latches in which an error may occur (which determines the cost of making the circuit robust to errors). For the design-level heuristic, this means that when we define the partition between paths that are allowed have errors (P_{-}) and paths that are error-free (P_{+}), we must consider the ER contribution of each path, which adds to the dynamic recovery overhead of Razor. We must also account for the cost of using a Razor flip-flop (FF) at the endpoint of any path that may potentially cause a timing error, and buffering for any short paths terminating at that endpoint. If downsizing a path during *ReducePower* requires that we must replace a regular FF with a Razor FF, then we should ensure that the energy benefit (in terms of power reduction for additional cell downsizing) outweighs the additional cost of the Razor FF and any short-path hold buffering. Since Razor assumes a maximum delay constraint on all paths [25], in addition to checking P_{+} paths for negative slack (line 16 of *ReducePower*) we must also ensure that all P_{-} paths respect the delay constraint after a downsizing move.

B. Case Study: Application Noise Tolerance

Error-tolerant applications [32] represent an opportunity to save power and increase performance by allowing errors to propagate to the application level rather than expending power to detect and correct them at the hardware level. For several such applications, data errors simply result in reduced output quality, instead of program failure.

Designing a recovery-driven processor for error-tolerant applications requires several considerations. First, the set of processor modules is partitioned into two subsets—one containing modules that produce errors that the applications can tolerate and another containing modules that should not allow errors to propagate to the application level. For the class of error-tolerant applications that we consider in this paper, errors in the arithmetic units (i.e., arithmetic logic unit and floating-point unit) can be tolerated. For this class of applications (which relies heavily on numerical computation), the arithmetic units account for approximately 35% of the dynamic power consumption of the processor.

In addition to the list of modules to optimize, the *OptimizeProcessor* procedure requires a target ER. The ER is chosen such that all applications in the class have acceptable quality for the target ER.

For the modules that produce errors that the application cannot tolerate, one of two approaches can be followed. One option is to operate those modules on the same voltage rail as the modules in which faults are allowed (single rail design). In this case, we feed these modules to the optimization heuristic targeting some hardware recovery mechanism that guarantees correctness, such as Razor. The two groups must agree on a common voltage that minimizes power consumption for the entire processor, and the optimal voltage reported by the optimization heuristic can be used as a constraint for the second

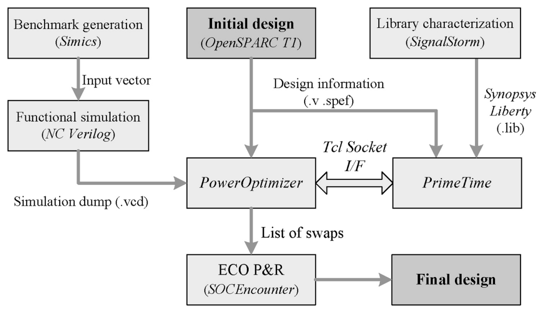

Benchmark generation (Simics)
Initial design (OpenSPARC T1)
Library characterization (SignalStorm)
Input vector
Design information (.v .spef)
Synopsys Liberty (.lib)
Functional simulation (NC Verilog)
Tcl Socket I/F
PowerOptimizer
PrimeTime
Simulation dump (.vcd)
List of swaps
ECO P&R (SOCEncounter)
Final design

Fig. 6. Computer-aided design flow incorporating the power optimization heuristic to minimize the power of a design for a given error tolerance technique.

optimization. Alternatively, the two groups can operate in separate voltage domains (dual rail design), in which case each optimization can select a different optimal voltage.

Soft processor design can also be used to adapt the reliability of the processor for reliability diverse workloads, with more power savings available as the ER target decreases. To create a soft processor design, the gradual slack module-level heuristic is used, and the optimal voltage and ER targets of the modules are chosen based on the range of ER targets that the processor should support.

V. Methodology

Our methodology for demonstrating the benefits of recovery-driven design has two parts—a design-level methodology to characterize the power and reliability of circuit modules optimized for different voltage and ER targets, and an architecture-level methodology to estimate processor power and performance when the proposed design-level techniques are applied at the processor-level.

A. Design-Level Methodology

We use the OpenSPARC T1 processor [38] to test our optimization framework. Table I describes the selected modules and provides characterization in terms of cell count and area. Module designs are implemented in TSMC 65GP technology using a standard flow of synthesis with Synopsys Design Compiler vY-2006.06-SP5 [39] and place-and-route with Cadence SoC Encounter v8.1 [37]. Runtime is reduced by adopting a restricted library of 66 commonly used cells[2] (62 combinational and 4 sequential). Conventionally constrained designs are synthesized for the target operating frequency (0.8 GHz), and tightly constrained designs are synthesized for a 40% smaller clock period to increase timing slack.

Fig. 6 illustrates our recovery-driven design flow. We perform gate-level simulation to produce a VCD file[3] using Cadence NC-Verilog v6.1 [35]. To find timing slack and power values at specific voltages, we prepare Synopsys Liberty (.lib) files for each voltage from 1.00 V to 0.50 V in 0.01 V increments, using Cadence Library Characterizer v9.1 [36].

[2]Heuristic efficiency depends on the number of available logically equivalent cells. Since we use all available cell sizes for different drive strengths, our heuristic will also be effective with a full set of library cells.

[3]Gate-level simulation is performed for one million cycles, and the size of the VCD file is about 500 MB for our test cases. To implement larger designs, a compressed VCD file could be used—e.g., Synopsys VCD Plus format.

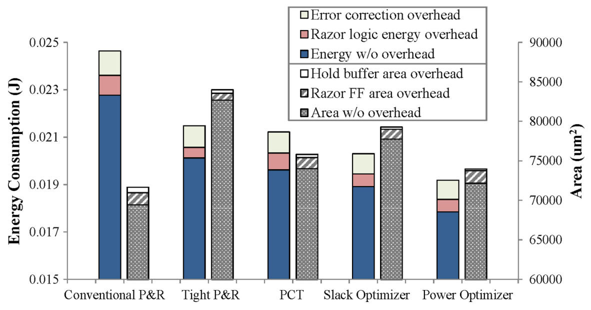


Fig. 7. Energy and area overheads for Razor-based design.

TABLE I
TARGET MODULES FOR EXPERIMENTS

Module	Stage	Description	Cell #	Area (μm^2)
lsu_dctl	MEM	L1 Dcache control	4537	13 850
lsu_qctl1	MEM	LDST queue control	2485	7964
lsu_stb_ctl	MEM	ST buffer control	854	2453
sparc_exu_ecl	EX	Execution unit control	2302	7089
sparc_ifu_dec	FD	Instruction decode	802	1737
sparc_ifu_errdp	FD	Error datapath	4184	12 972
sparc_ifu_fcl	FD	L1 Icache and PC control	2431	6457
spu_ctl	SPU	Stream processing control	3341	9853
tlu_mmu_ctl	MEM	MMU control	1701	5113

Complete characterization for 51 voltage points takes a couple of days, but this is a one-time cost.

Timing information is continually available from Synopsys PrimeTime c2009.06 [40] static timing tool through the Tcl socket interface, during the optimization process. After our optimization, all netlist changes are realized using Cadence SoC Encounter in ECO mode.

Gate-level simulation is performed using test vectors obtained from full-system register-transfer level (RTL) simulation of a benchmark suite consisting of integer and floating-point SPEC benchmarks. These benchmarks are each fast-forwarded to their early SimPoints using the OpenSPARC T1 system simulator, Simics [22] Niagara. After fast-forwarding in Simics, the architectural state is transferred to the OpenSPARC RTL using Carnegie Mellon University Transplant [5].

Our recovery-driven design techniques optimize for average activity. To ensure that the activity profiles used during optimization (training) are representative and adequate, we use mutually exclusive training and test workloads. We optimize based on the average activity of half of our benchmarks and test using the other half. Training and test sets are chosen randomly and contain half integer and half floating-point benchmarks. Table II shows the benchmarks in the training and test sets.

When characterizing Razor-based designs, we use worst case timing libraries to determine any path that might have negative slack under worst case process, voltage, temperature variations. We assign a Razor FF to the endpoint of any such path, add a maximum delay constraint of 1.5 cycles to the path, and add a minimum delay constraint of 0.5 cycle to all paths ending at that FF. We add buffers to any path that does not meet the minimum delay constraint. Razor FFs have higher power, delay, and area than normal FFs [7]. An error triggers a recovery period during which the pipeline recovers to a correct state. During this time, we assume that no progress is made,

TABLE II
BENCHMARKS

Benchmarks for Design Optimization (Training Set)	
ART	Image recognition/neural nets
BZIP2	Compression
MCF	Combinatorial optimization
MESA	3-D Graphics library
Benchmarks for Design Evaluation (Test Set)	
equake	Seismic wave propagation
gzip	Compression
twolf	Place and route simulator
sort	Sorting
Additional Benchmarks for Processor-Level Evaluation	
AMMP, APPLU, MGRID, PARSER, SWIM, CRAFTY, EON, WUPWISE	
VPR, VORTEX-2, FACEDETECT†, CG†, LSQ†	

†Error-tolerant application.

TABLE III
PROCESSOR SPECIFICATIONS

Property	Value	Property	Value
L1 cache	16 kB, 4 way, 1 cyc	RegFile	72 (int), 72 (FP)
L2 cache	2 MB, 8 way, 8 cyc	Branch predict	gshare (8K entries)
Execution	2-way OO	Mem Access	315 cycle

but we do account for the power and time consumed during recovery when reporting processor throughput and energy. We assume a counterflow pipeline-based Razor implementation [7] with a recovery penalty proportional to the depth of the pipeline (nine cycles for our nine-stage pipeline). We use the ER, in conjunction with the rates of power consumption during normal operation and error recovery, as well as the recovery time overhead of Razor to calculate the energy overhead of error recovery [7]. Fig. 7 compares the energy and area overheads of Razor for each design style that we evaluate. The fraction of Razor FFs ranges from 2.6% for a tightly constrained design to 5% for a Razor-optimized recovery-driven design.[4] Our Razor-optimized recovery-driven design heuristic directly accounts for the overheads of adding a Razor FF to ensure increased energy savings, even if additional Razor FFs are required.

B. Architecture-Level Methodology

We use SMTSIM [31] integrated with Wattch [2] to simulate processors whose single-core parameters are in Table III. The simulator reports performance and power numbers at different voltages. Our evaluations are done using benchmarks in Table II. These benchmarks were chosen to maximize diversity in terms of performance and reliability requirements. We base our out-of-order processor microarchitecture model on the MIPS R10000 [34].

To get a processor-wide ER at a given frequency and voltage, we first sum the ERs from all the sampled OpenSPARC modules and then scale up the sum based on area, such that it includes all modules that we target for optimization. The ER of a module that has not been characterized is assumed to be proportional to area. We target only logic modules with our recovery-driven design methodology. On-chip memories are assumed to operate on a separate voltage rail [16] at the lowest error-free voltage for a given operating frequency. At 45 nm and below, such "split rail" designs are common. While we provision for error-free static random-access memories

[4]In our previous work [17], [18], all flip-flops were Razor flip-flops, leading to different absolute power and area numbers.

(SRAMs), logic interfacing with SRAM structures, such as register read and writeback logic, may still produce errors. For designs that rely on error-tolerant applications, we scale the ERs of each module group separately, according to an ER characterization of sampled modules in the group. Once the processor core-wide ER is calculated, we can use performance and power numbers reported by our simulators to estimate the throughput and power impact of errors for a given error recovery overhead.

We use a similar methodology to get processor-wide power numbers. To get a dynamic power estimate, we scale the dynamic power numbers reported by Wattch for the optimizable components by the ratio of total module power for an optimization technique over total module power for the baseline design, as reported by Synopsys PrimeTime. For designs that exploit application-based error resilience, we scale the power of the module groups independently, as we did for ER. For the nonoptimizable components, the Wattch numbers are scaled based on the minimum voltage that these components can run at without producing timing errors. For static power estimation, we use the ratio of dynamic and static module power for an optimization technique, as reported by PrimeTime, to calculate static power for a dynamic power value reported using the above methodology.

When a processor designed for application-level reliability runs an application that requires correctness, we scale down the frequency of the processor so that no timing violations occur. The safe clock frequency of the design is determined by the worst case negative slack timing path in the processor plus a safety margin. All of our application simulations are executed for one billion cycles after fast-forwarding to the early SimPoints [27].

VI. Experimental Results

We now evaluate our recovery-driven design implementations, which redistribute timing slack to reduce the ER at a given voltage, allowing a reduction in voltage and energy for a given target ER and operating frequency.

A. Evaluation of Heuristic Design Choices

Fig. 8 shows power and runtime of the various heuristic design alternatives that we evaluated, as described in Section III-E. For path ordering during optimization, considering the slack in the prioritization function results in higher power than the case where only TR is used. Runtime is somewhat smaller, but since our optimization iterates over a path multiple times until no slack increase is observed, both results perform similarly. For the same reason, path traversal order has little effect on the optimization result. We choose the TR priority function for its simplicity and lower power.

The results for optimization radius show that swapping cells in the fanin/fanout networks not only increases power at some ERs, but also greatly increases runtime due to the large amount of swaps that are performed. Thus, we choose to swap cells only on the optimized path. In the experiments on accuracy of path selection and ER budget utilization, we observe no difference in power. Both updating the ER contribution continuously during path selection and ensuring

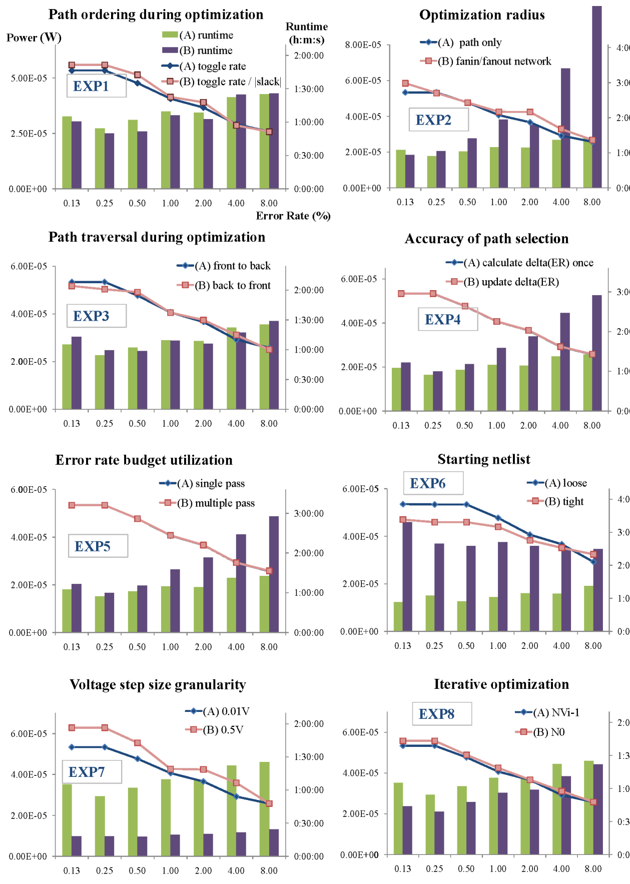


Fig. 8. Evaluation of different heuristic design choices. The choices are evaluated in terms of power of the resulting design as well as runtime.

full utilization of the ER budget increase runtime significantly without providing power benefits, and these techniques are not used in the final heuristic implementation.

The choices of starting netlist and voltage step size have a significant effect on power. Our recovery-driven design heuristic employs two main procedures: *OptimizePaths* (cell upsizing to reduce the ER) and *ReducePower* (cell downsizing to reduce area and power). When starting the optimization flow from a loosely constrained design, path optimization provides the most substantial contribution to power reduction by reducing the ER and extending voltage scaling. However, when starting from a tightly constrained design, much optimization has already been performed, and the power reduction stage of our heuristic is essential for power minimization. Although runtime increases due to evaluation of more downsizing moves, a tightly constrained netlist provides a better starting point, since it permits more voltage scaling. Voltage scaling has a stronger effect on power reduction and scales the power of all cells, while area reduction only affects the downsized cells. Also, starting from a tightly constrained design reduces the dependence on ECO, which improves the optimization efficiency. Using a coarser-granularity voltage step reduces runtime significantly, but comes at the cost of power, since the heuristic cannot hone in on the optimal voltage as easily. For higher ERs, a large step size can provide a near-optimal power result and a large reduction in runtime. Thus, ER-aware adaptive step sizing can be beneficial.

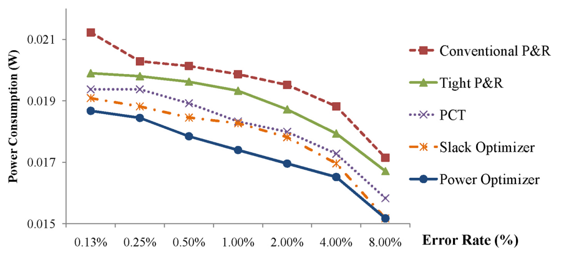


Fig. 9. Power consumption of each design technique at various target ERs for target modules in Table I. (Additional results are available at [41].)

TABLE IV
POWER SAVINGS (%) FOR ER-OPTIMIZED RECOVERY-DRIVEN DESIGNS COMPARED TO TRADITIONAL P&R

	Target ER (ER_{target})						
MODULE	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%	8.0%
lsu_dctl	29.1	16.8	16.8	16.8	16.8	16.8	21.6
lsu_qctl1	8.8	6.7	5.8	8.1	11.0	9.0	8.6
lsu_stb_ctl	17.9	17.9	18.1	15.4	9.6	19.2	2.9
sparc_exu_ecl	6.0	6.0	18.3	18.3	22.7	23.3	17.4
sparc_ifu_dec	13.7	10.1	8.6	14.3	15.9	18.5	15.1
sparc_ifu_errdp	2.2	2.8	5.7	5.7	5.7	9.3	9.3
sparc_ifu_fcl	14.5	15.4	16.5	19.2	19.2	19.2	19.2
spu_ctl	13.1	13.1	13.1	13.2	8.8	1.6	8.9
tlu_mmu_ctl	0.8	0.8	0.8	0.8	0.8	0.8	0.8

In terms of iterative optimization, we observe that our heuristic is able to achieve the same result independent of the starting netlist. Thus, we choose the option that minimizes runtime.

B. Comparison Against Alternative Flows

To demonstrate the benefits of our recovery-driven design flow, we compare five alternative design flows: traditional P&R implementations with conventional and tight timing constraints, a BlueShift-like path constraint tuning (PCT) approach, gradual slack design [17], [18], and our heuristic for ER-optimized recovery-driven design. Fig. 9 compares the power consumptions of the various design techniques at several target ERs.

Recovery-driven designs reduce power by enabling extended voltage scaling and keeping area overhead low with respect to other optimization techniques. Compared to a conventionally optimized design, a recovery-driven design operates at a much lower voltage for a given target ER, due to the functionally aware optimization approach that optimizes the paths that cause the most errors. Compared against a highly optimized design that uses tightly constrained P&R, a recovery-driven design reduces power by minimizing the amount of area spent on path optimization. Traditional tightly constrained designs are functionally agnostic and optimize all paths heavily, incurring a large area overhead. Recovery-driven designs, on the other hand, use functional information to target only the paths that cause the most errors, thereby minimizing the area cost of additional voltage scaling. In scenarios where the cost of area is high, such as for technologies with higher leakage like those forecasted in future technology generations, the cost of functionally agnostic optimizations will increase, and the benefits of recovery-driven design will increase. Table IV shows power savings for recovery-driven design for each module with respect to traditional P&R at different target ERs.

TABLE V
AVERAGE AREA OVERHEAD WITH RESPECT TO THE BASELINE

Tight P&R	PCT	SlackOpt	PwrOpt 0.125%	PwrOpt 0.25%
19.1%	5.0%	11.9%	3.9%	4.3%
PwrOpt 0.5%	PwrOpt 1%	PwrOpt 2%	PwrOpt 4%	PwrOpt 8%
4.8%	5.4%	5.8%	6.0%	5.3%

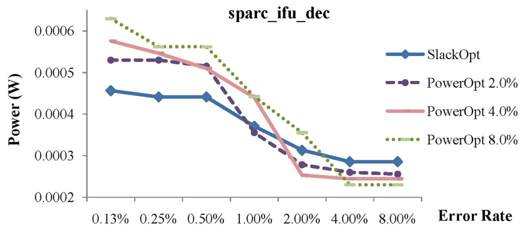


Fig. 10. Recovery-driven design for a target ER (PowerOpt) minimizes power at the target ER. Gradual slack design (SlackOpt) optimizes a design for a range of ERs to provide adaptability and smooth performance/power tradeoffs.

In our power minimization heuristic, after deciding how to allocate the ER budget, the *ReducePower* stage performs aggressive cell downsizing to reduce circuit area and power. Table V compares recovery-driven design against other design flows in terms of area overhead with respect to the baseline design. Design for a target ER has similar area overhead to PCT but still produces a design with lower power. The reason is that designing for a target ER allows more aggressive voltage scaling before the target ER is exceeded. At lower voltages, there are more negative slack paths to be optimized during *OptimizePaths*, which increases area overhead. However, aggressive downsizing keeps area overhead low, and since the paths targeted by *PowerOptimizer* are the paths that cause the most errors in the design, the area is well spent, and the additional voltage scaling contributes to a net benefit in terms of power savings. PCT, on the other hand, adds tighter timing constraints to the registers where the most errors are captured and optimizes all paths with endpoints at those registers. Since our heuristic targets paths individually, we can target the error-causing paths more efficiently, reduce overhead, and increase voltage scaling and power savings.

Compared to tightly constrained P&R and gradual slack design, design for a target ER incurs significantly less area overhead and reduces power. On one hand, tightly constrained P&R is functionally agnostic and fails to identify the set of paths that maximizes voltage overscaling per unit area overhead. Gradual slack design, on the other hand, optimizes the design to make tradeoffs between power, throughput, and reliability over a *range* of ERs. Thus, a gradual slack design is over-optimized for any single target ER.

Fig. 10 compares recovery-driven design for a target ER against gradual slack design. The results show that designing for a target ER minimizes power at the target ER. However, since a recovery-driven design can have a nonzero ER even under nominal conditions, power efficiency at ERs lower than the target may drop off steeply. Likewise, since design for a target ER creates a slack wall at the error-optimal voltage, additional benefits for ERs higher than the target are limited. A gradual slack design, on the other hand, is optimized for a *range* of ERs. Although this means that it is less efficient than an ER-optimal design for any single ER, it also means

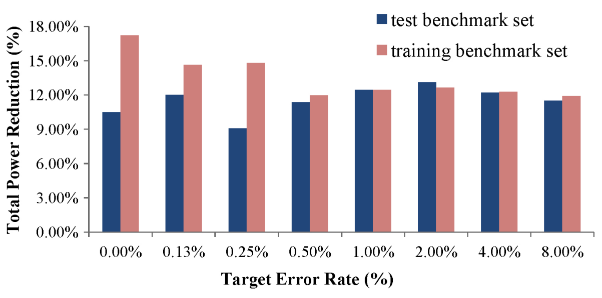


Fig. 11. Total power reduction over tightly constrained design for the training (optimization) and test benchmark sets. Power reductions for the training set are slightly higher, since the design has been optimized specifically for the activity profile of this set.

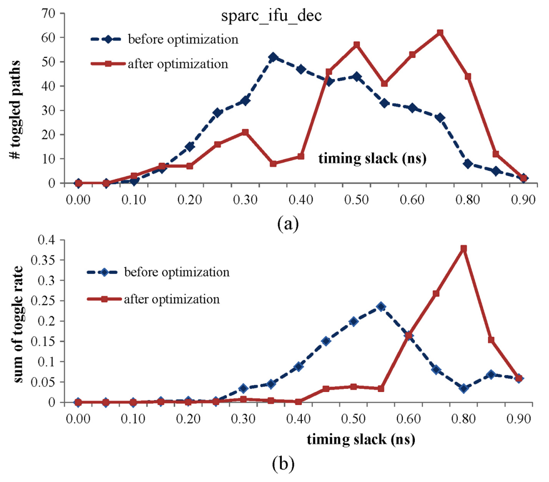


Fig. 12. (a) Recovery-driven design reshapes the slack distribution by adding slack to frequently exercised paths and removing slack from infrequently exercised paths. (b) Activity-weighted slack distribution shows the sum of TRs for all paths with a particular slack value, confirming that frequently exercised paths have more slack in the optimized netlist.

that performance or output quality can be efficiently traded for power savings over the entire range of ERs. Thus, whenever more errors can be tolerated, a gradual slack design can reduce power consumption. This may not be possible for an ER-optimal design, since it forgoes scalability to achieve additional power savings at the target ER.

Recovery-driven design optimizes for errors in the average operating behavior of a design. If the frequently exercised paths during operation are significantly different than those targeted during optimization, then too many errors may be produced, and voltage scaling may be limited for a target ER. To evaluate the robustness of recovery-driven design when the workload changes, we compared the power reduction achieved when running the training (optimization) benchmarks against power reduction for the test benchmarks. Fig. 11 shows that power reduction is slightly higher for the benchmark set that the processor was optimized for, but the difference is only about 1% on average.

C. Variation-Aware Analysis

Recovery-driven design increases energy efficiency by reshaping the slack distribution of a design, such that ER is reduced at a particular voltage. Fig. 12 shows activity-weighted slack distributions (sum of path TR versus timing slack) from before and after optimization, confirming that the optimization increases slack for frequently exercised paths, which enables

TABLE VI
VARIATION-AWARE ANALYSIS

	Target ER (ER_{target})						
	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%	8.0%
Power Consumption (W) in Baseline Design							
Min.	0.0126	0.0126	0.0122	0.0113	0.0108	0.0106	0.0095
Max.	0.0202	0.0201	0.0199	0.0196	0.0191	0.0186	0.0167
Avg.	0.0162	0.0160	0.0156	0.0153	0.0149	0.0141	0.0127
Power Consumption (W) in Recovery-Driven Design							
Min.	0.0111	0.0106	0.0105	0.0096	0.0092	0.0088	0.0080
Max.	0.0187	0.0183	0.0175	0.0172	0.0165	0.0161	0.0151
Avg.	0.0148	0.0144	0.0141	0.0134	0.0128	0.0123	0.0113
Power Reduction (%)							
Avg.	8.28	9.71	9.43	12.61	13.80	13.03	11.18

extended voltage scaling for a target ER. However, due to random variations introduced in the physical circuit by sources of static and dynamic nondeterminism, the actual slack distribution may be somewhat different than the designed distribution.

To test the benefits of recovery-driven design in the presence of variations, we have implemented a model for interdie and spatially correlated within-die variations based on the models in [4] and [14]. We use an exponential model for correlation between different die locations, in which the correlation function decays exponentially as a function of distance, with parameters supplied by the authors of [14]. We extract standard deviations (σ) of cell delay at each operating voltage from SPICE simulations, and use our variation model to assign a random delay variation to each die and each gate within the die, based on its location. We then repeat ER and power estimation with 100 different random variation maps. From the Monte Carlo simulations, we report total power consumption of the target modules at each ER in Table VI. Table VI shows that even when variations are accounted for, recovery-driven design still achieves significant power savings over a conventional design. Furthermore, the average benefits do not noticeably change when variations are accounted for. (Power reduction in Table VI is somewhat lower for ERs below 1% because the test design was optimized for a target ER of 1%.) Random variations cause perturbations within a design but do not shift the average case behavior. Since recovery-driven designs are optimized for and operate at the average case operating point, they are naturally robust to random variations.

D. Recovery-Driven Processors

In this section, we demonstrate the benefit of designing processors for specific hardware and software error resilience mechanisms, as described in Section IV.

1) *Circuit-Level Timing Speculation:* Fig. 13 compares the energy consumption of a recovery-driven processor that has been designed and optimized for Razor against the power consumption of processors designed for other objectives, such as gradual slack or PCT, and against processors that have been designed for correctness but use the traditional Razor methodology to save energy. We assume a recovery overhead of nine cycles, proportional to the pipeline depth of the processor.

Fig. 13 demonstrates that the minimum energy is indeed achieved by a processor that is designed to produce errors that can be gainfully tolerated by Razor. Designing the processor for the ER target at which Razor operates most efficiently allowed us to extend the range of voltage scaling from 0.84 V

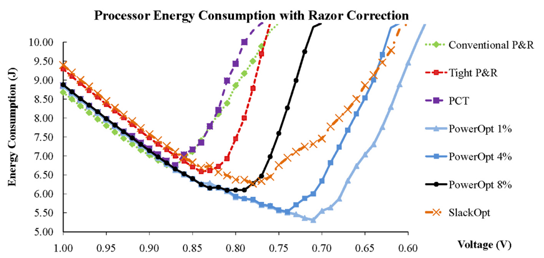


Fig. 13. Benefit of designing a processor to produce errors then correcting them with an error tolerance mechanism over designing for correctness and then relaxing the correctness guarantee can be significant. Results are shown for processors that employ Razor.

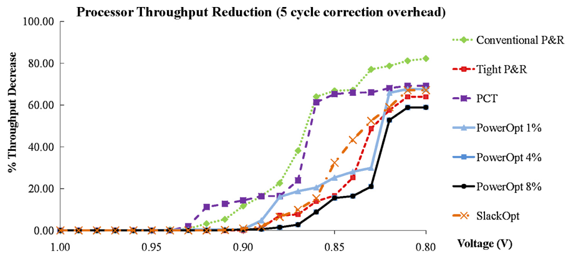


Fig. 14. Throughput reduction at different voltages for an error recovery overhead of five cycles. This recovery overhead is appropriate for a simple pipeline or lightweight recovery technique.

for the best "designed for correct operation" processor to 0.71 V for the processor designed for an ER of 1%, affording an additional 19% energy reduction.

Error recovery with a circuit-level approach like Razor imposes a throughput penalty, since error recovery requires feeding correct values back into the pipeline. Fig. 14 shows the throughput reduction caused by error recovery for a correction overhead of five cycles. As can be seen, a recovery-driven processor even minimizes the recovery overhead at the target operating voltage.

2) *Application Noise Tolerance:* To demonstrate the benefits of recovery-driven design targeted at application-level noise tolerance, we use a face detection algorithm [32] as the example application. Face detection is naturally robust to errors in several processor modules and does not require strict computational correctness. Rather than causing program failure, errors may result in reduced output quality (false positive or negative detections) [24].

Face detection, as well as the other error-tolerant applications we consider, tolerates errors in the arithmetic units of the processor. For this class of applications (which relies heavily on numerical computation), the arithmetic units account for approximately 35% of the dynamic power consumption of the processor.

Figs. 15 and 16 compare the power consumption of processors designed for application-level error tolerance of arithmetic errors using single and dual voltage rail designs, as described in Section IV. In these figures, all processors achieve the same output quality at a given ER, but processors designed to allow errors consume less power, and power is minimized for these designs at their respective ER targets. For example, at an ER of 1%, where output quality is still maximized for the face detection application, the processor designed for an ER target of 1% consumes 19% less power for dual-rail design and 15% less power for single-rail design than the baseline correctness-

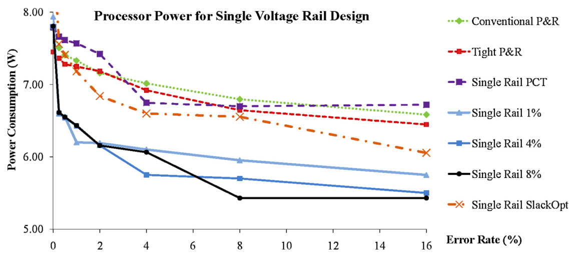


Fig. 15. Power benefit of a processor that is designed to allow errors in the arithmetic units over a processor that is designed for correctness. All modules in the processor operate at the same voltage. Razor is used to correct errors in nonarithmetic units.

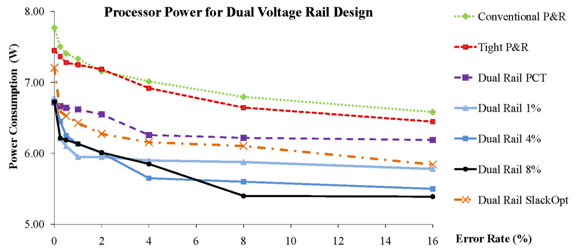


Fig. 16. This figure demonstrates the power benefit of a processor that is designed to allow errors in the arithmetic units over a processor that is designed for correctness. The processor uses a dual voltage rail design with the arithmetic units on a separate rail.

TABLE VII

Optimal Module Voltages at Different Target ERs

	Target ER (ER_{target})						
MODULE	0.0%	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%
lsu_dctl	0.75	0.72	0.71	0.75	0.74	0.73	0.72
lsu_qctl1	0.88	0.87	0.86	0.85	0.84	0.83	0.80
lsu_stb_ctl	0.77	0.76	0.75	0.75	0.70	0.68	0.66
sparc_exu_ecl	0.75	0.74	0.73	0.70	0.70	0.69	0.70
sparc_ifu_dec	0.68	0.67	0.66	0.63	0.70	0.58	0.57
sparc_ifu_errdp	0.77	0.58	0.57	0.56	0.55	0.54	0.53
sparc_ifu_fcl	0.79	0.77	0.76	0.75	0.74	0.73	0.72
spu_ctl	0.78	0.65	0.64	0.63	0.62	0.63	0.58
tlu_mmu_ctl	0.85	0.52	0.51	0.51	0.51	0.51	0.51
RANGE	0.20	0.35	0.35	0.34	0.33	0.32	0.29

optimized processor. Benefits are even higher for larger ERs if some application output degradation is permissible.

Note that we can always perform error-free computation on a core designed for application-level noise tolerance by scaling down the frequency to the point where all paths have nonnegative slack. However, this may represent a performance penalty when compared to relaxed-correctness operation.

Also note that trends in processor-level results may differ somewhat from trends in averaged module-level results. Whereas the power reduction of a recovery-driven design is limited by a module's critical paths, the power reduction of a recovery-driven processor is biased by the critical modules that begin causing errors first when voltage is scaled down. As we will show in the next section, results can be improved by utilizing multiple voltage domains.

E. Supporting Multiple Voltage Domains

Given a target ER, the module-level power minimization heuristic in [19] selects an optimal operating voltage for a processor module. However, the proposed processor core-level methodology (Algorithm 1, $DOMAINS = 1$) selects a common voltage for all modules of a processor core. Table VII shows that different modules vary (sometimes substantially) in their optimal voltage operating points due to a number of

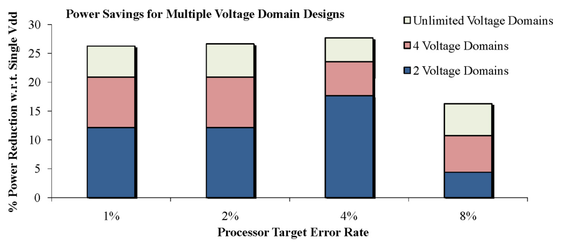


Fig. 17. Benefit of a multiple voltage domain design over a single voltage domain design can be significant when designing for an ER target. Substantial power savings can be achieved when each module is optimized for a locally optimal voltage rather than the globally optimal voltage of the module group. The stacked bars show the additional power savings afforded as the number of voltage domains increases.

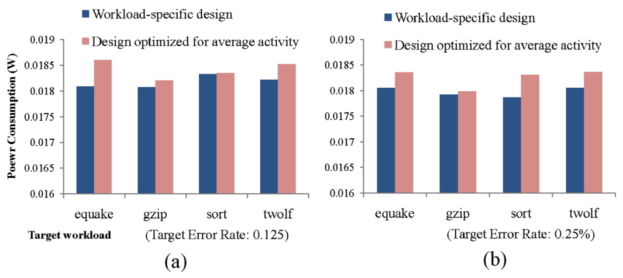


Fig. 18. Recovery-driven design is robust to application diversity. On average, processor modules that have been optimized for the average case only consume 1% more power than modules that have been customized specifically for the activity profile of the test workload. (a) Target error rate: 0.125%. (b) Target error rate: 0.25%.

factors, including module area (number of paths/cells), slack distribution (fraction of paths that are critical), and activity factor (how often paths toggle). In addition, the table shows that the range of optimal module voltages increases when designing for a nonzero ER target.

Because of the above module-level variations, there can be a substantial difference in terms of power consumption between the locally and globally optimized module implementations. Fig. 17 quantifies the difference between single and multiple voltage domain design for processor cores tolerating different ERs. We compare designs with different numbers of voltage domains, targeting different processor ERs in terms of their power consumption relative to a processor optimized for a common operating voltage. The results show that the power efficiency of recovery-driven processors will improve significantly with the number of voltage domains that are supported. In practice, the number of voltage domains should be chosen by carefully balancing the voltage overscaling benefits with the area and complexity overheads of supporting multiple power rails. The results of Fig. 17 do not consider the overhead of level shifter circuitry.

F. Robustness to Application Diversity

Different workloads exercise the timing paths of a processor core differently. Thus, the sets of frequently exercised and infrequently exercised paths may change, depending on the workload. Since recovery-driven designs are optimized according to an average case activity profile, it is important to ensure that power efficiency is not degraded significantly when the activity profile of a workload is not the same as the activity profile for which the processor was optimized.

To gauge the robustness of recovery-driven design to workload diversity, we create several recovery-driven designs, optimized for the activity profiles of each benchmark in the test set—equake, gzip, sort, and twolf. Then, we compare the power consumption of each benchmark in the test set, running on the design that was optimized for the average case, against the design that was optimized specifically for that benchmark. Fig. 18 compares the power consumption of average case design against workload-specific designs for different target ERs.

On average, the difference is small—only 1.5% difference in power at an ER of 0.125% and 0.9% difference at 0.25%—demonstrating the robustness of recovery-driven design to application diversity. The difference will decrease as the target ER increases. The reason for this robustness is that since some paths are allowed to cause errors, there is some “forgiveness” when the priority of path optimization deviates somewhat from the optimal. Our recovery-driven design heuristic bins paths into P_- paths that are allowed to cause errors and P_+ paths that should remain error free. As long as the difference in activity for a path is not so much as to make the path switch bins, the path dichotomy is preserved and power efficiency is not degraded. In the worst case, we only observe 3% degradation in power efficiency.

VII. Conclusion

In this paper, we have proposed *recovery-driven design*, a design-level approach that optimized a processor module for a target timing ER instead of correct operation. We have presented a detailed evaluation and analysis of a recovery-driven design methodology that minimized power for a target ER. We extend our recovery-driven design flow to design recovery-driven processors—processors that are designed and optimized for a target ER. We also present an extension of our recovery-driven design flow that creates a gradual slack design optimized for a range of ERs rather than a single target. The gradual slack technique can be used to design soft processors that trade throughput or output quality for energy savings over a range of reliability targets. While we have chosen to focus on improving the energy efficiency of error-resilient designs, recovery-driven design can also be used to optimize other design characteristics, such as yield. We intend to evaluate such adaptations of recovery-driven design in future work.

References

[1] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, “Oppotunities and challenges for better than worst-case design,” in *Proc. Asia South Pacific Des. Automat. Conf.*, 2005, pp. 2–7.
[2] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A framework for architectural-level power analysis and optimizations,” in *Proc. Int. Symp. Comput. Architect.*, 2000, pp. 83–94.
[3] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee, “Ultra-efficient (embedded) SoC architectures based on probabilistic CMOS (PCMOS) technology,” in *Proc. Des., Automat. Test. Eur.*, 2006, pp. 1110–1115.
[4] L. Cheng, P. Gupta, C. Spanos, K. Qian, and L. He, “Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability,” in *Proc. ACM/IEEE Des. Automat. Conf.*, Jul. 2009, pp. 104–109.
[5] E. Chung and J. Smolens. (2007). *OpenSPARC T1: Architectural Transplants* [Online]. Available: http://transplant.sunsource.net
[6] J. Cong and K. Minkovich, “Logic synthesis for better than worst-case designs,” in *Proc. Int. Symp. VLSI Des., Automat. Test*, 2009, pp. 166–169.
[7] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, “Razor: A low-power pipeline based on circuit-level timing speculation,” in *Proc. IEEE/ACM Int. Symp. Microarchitect.*, Dec. 2003, pp. 7–18.
[8] J. P. Fishburn and A. E. Dunlop, “Tilos: A polynomial programming approach to transistor sizing,” in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Des.*, Nov. 1985, pp. 326–328.
[9] S. Ghosh and K. Roy, “CRISTA: A new paradigm for low-power and robust circuit synthesis under parameter variations using critical path isolation,” *IEEE Trans. Comput.-Aided Des.*, vol. 26, no. 11, pp. 1947–1956, Nov. 2007.

[10] B. Greskamp, L. Wan, W. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen, and C. Zilles, "BlueShift: Designing processors for timing speculation from the ground up," in *Proc. Int. Symp. High-Performance Comput. Architect.*, 2009, pp. 213–224.

[11] P. Gupta, A. B. Kahng, and P. Sharma, "A practical transistor-level dual threshold voltage assignment methodology," in *Proc. Int. Symp. Qual. Electron. Des.*, 2005, pp. 421–426.

[12] P. Gupta, A. B. Kahng, P. Sharma, and D. Sylvester, "Selective gate-length biasing for cost-effective runtime leakage control," in *Proc. ACM/IEEE Des. Automat. Conf.*, Jul. 2004, pp. 327–330.

[13] P. Gupta, A. B. Kahng, P. Sharma, and D. Sylvester, "Gate-length biasing for runtime-leakage control," *IEEE Trans. Comput.-Aided Des.*, vol. 25, no. 8, pp. 1475–1485, Aug. 2006.

[14] B. Hargreaves, H. Hult, and S. Reda, "Within-die process variations: How accurately can they be statistically modeled?," in *Proc. Asia South Pacific Des. Automat. Conf.*, 2008, pp. 524–530.

[15] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proc. Int. Symp. Low Power Electron. Des.*, 1999, pp. 30–35.

[16] *Intel Atom Processor z5xx Series*, Intel Corporation, Santa Clara, CA, 2008.

[17] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *Proc. Asia South Pacific Des. Automat. Conf.*, 2010, pp. 825–831.

[18] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Designing a processor from the ground up to allow voltage/reliability tradeoffs," in *Proc. Int. Symp. High-Performance Comput. Architect.*, 2010, pp. 119–129.

[19] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Recovery-driven design: A methodology for power minimization for error tolerant processor modules," in *Proc. ACM/IEEE Des. Automat. Conf.*, Jun. 2010, pp. 825–830.

[20] R. Kumar, "Stochastic processors," in *Proc. NSF Workshop Sci. Power Manag.*, Mar. 2009.

[21] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proc. IEEE/ACM Int. Symp. Microarchitect.*, Dec. 2003, pp. 81–92.

[22] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[23] S. Narayanan, J. Sartori, R. Kumar, and D. Jones, "Scalable stochastic processors," in *Proc. Des., Automat. Test. Eur.*, 2010, pp. 335–338.

[24] J. Sartori, J. Sloan, and R. Kumar, "Fluid NMR: Performing power/reliability tradeoffs for applications with error tolerance," in *Proc. Workshop Power Aware Comput. Syst.*, 2009.

[25] J. Sartori and R. Kumar, "Overscaling-friendly timing speculation architectures," in *Proc. GLSVLSI*, 2010, pp. 209–214.

[26] N. Shanbhag, R. Abdallah, R. Kumar, and D. Jones, "Stochastic computation," in *Proc. ACM/IEEE Des. Automat. Conf.*, Jun. 2010, pp. 859–864

[27] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proc. Int. Conf. Architect. Support Program. Languages Oper. Syst.*, 2002, pp. 45–57.

[28] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda, and D. Blaauw, "Duet: An accurate leakage estimation and optimization tool for dual-Vt circuits," *IEEE Trans. Very Large-Scale Integr. Syst.*, vol. 10, no. 2, pp. 79–90, Apr. 2002.

[29] A. Sultania, D. Sylvester, and S. S. Sapatnekar, "Tradeoffs between gate oxide leakage and delay for dual T_{ox} circuits," in *Proc. ACM/IEEE Des. Automat. Conf.*, Jul. 2004, pp. 761–766.

[30] J. W. Tschanz, K. Bowman, S.-L. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. De, "A 45 nm resilient and adaptive microprocessor core for dynamic variation tolerance," in *Proc. Int. Solid-State Circuits Conf.*, 2010, pp. 282–283.

[31] D. M. Tullsen, "Simulation and modeling of a simultaneous multithreading processor," in *Proc. Annu. Comput. Measure. Group Conf.*, 2006, pp. 819–828.

[32] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vision*, vol. 52, no. 2, pp. 137–154, 2004.

[33] T. Yamada, S. Kataoka, and K. Watanabe, "Heuristic and exact algorithms for the disjunctively constrained knapsack problem," *Inform. Process. Soc. Japan J.*, vol. 43, no. 9, pp. 2864–2870, 2002.

[34] K. Yeager, "The MIPS R10000 superscalar microprocessor," in *Proc. IEEE/ACM Int. Symp. Microarchitect.*, Apr. 1996, pp. 28–40.

[35] *Cadence NC-Verilog User's Manual* [Online]. Available: http://www.cadence.com

[36] *Cadence LC User's Manual* [Online]. Available: http://www.cadence.com

[37] *Cadence SOCEncounter User's Manual* [Online]. Available: http://www.cadence.com

[38] *Sun OpenSPARC Project* [Online]. Available: http://www.sun.com/processors/opensparc

[39] *Synopsys Design Compiler User's Manual* [Online]. Available: http://www.synopsys.com

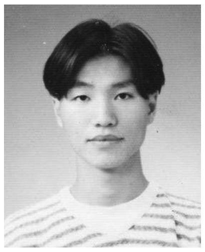
[40] *Synopsys PrimeTime User's Manual* [Online]. Available: http://www.synopsys.com

[41] *Experimental Results for All Testcases (Full Version)* [Online]. Available: http://vlsicad.ucsd.edu/RecoveryDriven/DATA.pdf

Andrew B. Kahng (F'10) received the A.B. degree in applied mathematics (physics) from Harvard College, Cambridge, MA, and the M.S. and Ph.D. degrees in computer science from the University of California at San Diego (UCSD), La Jolla.

In 1989 he joined the Department of Computer Science, University of California at Los Angeles, as an Assistant Professor, and became a Full Professor in 1998. In 2001, he joined UCSD as a Professor in the Department of Computer Science and Engineering (CSE) and the Department of Electrical and Computer Engineering (ECE); he served as Associate Chair of the CSE Department from 2003 to 2004. He has published over 350 journal and conference papers and three books, and holds 18 issued U.S. patents. Since 1997, his research in IC design for manufacturability has pioneered methods for automated phase-shift mask layout, variability-aware analysis and optimizations, chemical-mechanical polishing fill synthesis, and parametric yield-driven, cost-driven methodologies for chip implementation.

Prof. Kahng has received the NSF Research Initiation and Young Investigator Awards, and six Best Paper Awards. He was the founding General Chair of the 1997 ACM/IEEE International Symposium on Physical Design and a Co-Founder of the ACM Workshop on System-Level Interconnect Prediction. He defined the physical design roadmap as a member of the Design Tools and Test Technology Working Group (TWG) for the 1997, 1998, and 1999 renewals of the International Technology Roadmap for Semiconductors. From 2000 through 2003, he chaired the U.S. and International Design Technology Working Groups for the ITRS, and continues to serve as the Co-Chair of the Design ITWG. He has been an Executive Committee Member of the MARCO Gigascale Systems Research Center since its inception in 1998. In October 2004, he co-founded Blaze DFM, Inc., Sunnyvale, CA, and served as the Chief Technical Officer of the company until resuming his duties at UCSD in September 2006.



Seokhyeong Kang (S'11) received the B.S. and M.S. degrees in electrical engineering from the Pohang University of Science and Technology, Pohang, Korea, in 1999 and 2001, respectively.

From 2001 to 2008, he was with the System-on-Chip (SoC) Development Team, Samsung Electronics, Suwon, Korea. During his work at Samsung Electronics, he contributed to the development and commercialization of optical disk drive SoC. He joined VLSI CAD Laboratory, University of California at San Diego, San Diego, as a Ph.D. student in September 2008. His current research interests include low power design optimization and cost-driven methodology for chip implementation.

Rakesh Kumar (M'07) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, in 2001, and the Ph.D. degree in computer engineering from the University of California at San Diego (UCSD), San Diego, in Sep. 2006.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana. Prior to moving to Champaign in 2007, he was a Visiting Researcher with Microsoft Research, Redmond, WA. His current research interests include reliable and low power computing. He enjoys studying interactions between technology, policy, and society, when not involved in computing research.

Dr. Kumar has been recognized by several awards, including the Arnold O. Beckman Research Award, the FAA Creative Research Award, the UCSD CSE Best Dissertation Award, and an IBM Ph.D. Fellowship, for his research. Other recognitions include one Best Paper Award, several keynote invitations (WDSN 2011, LPonTR 2011, etc.), and invited lectures at conferences and workshops (CASES 2011, ISLPED 2010, IOLTS, 2010, etc.). He was an invited Guest Editor of the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE EMBEDDED SYSTEMS LETTERS, etc. He has served as the Chair of two workshops in the areas of robust computing and multicore computing (SELSE 2011 and dasCMP, from 2005 to 2008).

John Sartori (S'03) received the B.S. degree in electrical engineering, computer science, and mathematics from the University of North Dakota, Grand Forks, and the M.S. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign (UIUC), Urbana. He is currently pursuing the Ph.D. degree in electrical and computer engineering at UIUC. His thesis research explores design, architecture, and compiler techniques for stochastic processors.