Recovery-Driven Design: A Power Minimization Methodology for Error-Tolerant Processor Modules

Andrew B. Kahng[†+], Seokhyeong Kang[†], Rakesh Kumar[‡] and John Sartori[‡]
[†]ECE and [+]CSE Departments, University of California at San Diego
[‡]Coordinated Science Laboratory, University of Illinois
abk@cs.ucsd.edu, shkang@vlsicad.ucsd.edu, rakeshk@illinois.edu, sartori2@illinois.edu

ABSTRACT

Conventional CAD methodologies optimize a processor module for correct operation, and prohibit timing violations during nominal operation. In this paper, we propose *recovery-driven design*, a design approach that optimizes a processor module for a target timing error rate instead of correct operation. We show that significant power benefits are possible from a recovery-driven design flow that deliberately allows errors caused by voltage overscaling ([10],[3]) to occur during nominal operation, while relying on an error recovery 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 error rate. We demonstrate power benefits of up to 25%, 19%, 22%, 24%, 20%, 28%, and 20% versus traditional P&R at error rates of 0.125%, 0.25%, 0.5%, 1%, 2%, 4%, and 8%, respectively. Coupling recovery-driven design with an error recovery technique enables increased efficiency and additional power savings.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS—*Design Aids*; J.6 [**Computer Applications**]: COMPUTER-AIDED ENGINEERING

General Terms

Algorithms, Design, Performance

Keywords

Recovery-Driven Design, Power Minimization

1. INTRODUCTION

Trends in the semiconductor industry point to a significant increase in device-level variation as lithographic technology lags behind physical device scaling. Such trends have promoted variability to the forefront in terms of circuit-level design constraints and exposed the ever-increasing power cost of conservative design techniques aimed at ensuring correctness under worst-case conditions. Better-than-worst-case design techniques [1] have been employed successfully to reduce power consumption by eliminating guardbands, but such techniques still assume correct design in the average case, limiting their effectiveness for power reduction.

In this paper, we show that significant power benefits are possible from a design flow that (i) deliberately allows errors caused by voltage overscaling ([10],[3]) to occur during nominal operation, while (ii) relying on an error recovery technique to tolerate these errors. Moving away from the traditional philosophy of designing for correctness and correcting variation-induced errors as they arise, we propose *recovery-driven design* – a design approach that allows errors that can be gainfully tolerated by a hardware [3] or software-based [10] error tolerance technique. In other words, errors are selectively allowed in order to maximize the power savings for a given error rate. Power benefits result from timing slack being distributed from infrequently-executed paths to frequently-executed paths, reducing the error rate at a given voltage, and hence reducing the minimum supply voltage for a target error rate.

Our work shows that optimizing power for a target error rate results in significant power savings for similar levels of performance. Errors are either detected and corrected by a hardware error tolerance mechanism [3] or allowed to propagate to an error tolerant application [2] where the errors manifest themselves as reduced performance or output quality [10]. Increasing the target error rate increases the potential for power savings, since the processor module can be operated at a lower voltage. In practice, the target error rate is chosen such that an error recovery technique can correct the resulting errors and still achieve lower power (after considering the error tolerance overhead) for an acceptable degradation in performance or quality of output.

In this paper, we present a detailed evaluation and analysis of a CAD methodology that minimizes processor power for a target error rate. Our proposed methodology for designing for a target error rate makes the following contributions.

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 significant power savings – up to 25%, 19%, 22%, 24%, 20%, 28%, and 20% versus traditional P&R at error rates of 0.125%, 0.25%, 0.5%, 1%, 2%, 4%, and 8%, respectively.

We provide an exploration of the heuristic choices and tradeoffs that are fundamental to the optimization quality achieved. We evaluate choices for path ordering 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 error rate.

To support the proposed recovery-driven design flow, we present a fast, novel, and accurate technique for post-layout activity and error rate estimation that uses functional information to redistribute slack efficiently in a circuit, significantly extending the range of allowable voltage scaling for a given target error rate.

We demonstrate that the power benefits of a design produced by a recovery-driven design flow increase when a hardware-based error recovery technique is used. We consider Razor [3] as an example and show additional power reductions of 21% with respect to the next best approach.

2. BACKGROUND

2.1 Motivation

Figure 1 motivates this work by demonstrating that efficient redistribution of timing slack from infrequently-exercised paths to frequently-exercised paths reduces the error rate at a given voltage, allowing a reduction in voltage for a given target error rate.

The goal of the design problem can be stated formally as follows. Given an initial netlist N_0, a set of cell libraries characterized for allowable operating voltages, toggle rates for the toggled paths in the netlist, and a target error rate ER_{target}, produce the optimized netlist $N_{V_{opt}}$ and operating voltage V_{opt} that minimize the total power consumption $W_{V_{opt}}$ of the circuit, such that the error rate of the optimized netlist does not exceed ER_{target}. In this paper, we present a methodology to solve this design problem.

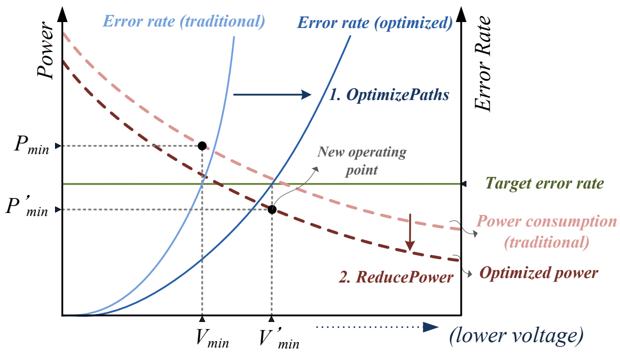

Figure 1: Our power optimization redistributes slack from infrequently-exercised paths to frequently-exercised paths and performs aggressive cell downsizing for average-case conditions. These optimizations reduce the power consumption of a circuit and extend the range of voltage scaling before a target error rate is exceeded. The combination of these effects results in operation at a much lower power point.

2.2 Related Work

Design-level Optimization. Design-level optimizations for error-tolerant designs ([6], [5], [11]) identify and optimize critical paths which are frequently exercised during operation. *BlueShift* [6] identifies the most-frequently violated timing paths during gate-level simulation, and optimizes the paths iteratively until the error rate 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. *CRISTA* [5] isolates critical paths by Shannon-expansion-based partitioning. After partitioning, *CRISTA* downsizes cells on the critical path and upsizes cells on the non-critical paths: critical paths are made slower while non-critical paths are made faster. When a critical path is excited, the corresponding operation takes two cycles. Kahng et al. [11] [12] propose *power-aware slack redistribution* to shift the timing slack of frequently-exercised, near-critical timing paths in a power-efficient manner. They identify frequently exercised paths with switching activity information and optimize the paths using a cell sizing method.

Our present work differs from *BlueShift* in both objective and approach. Our objective is to minimize power, while *BlueShift*'s objective is to improve performance. Consequently, our sensitivity functions are voltage-aware. Also, *BlueShift* requires iterative gate-level simulation, making the approach impractical for large SOC designs. *CRISTA* changes the structure of the original circuit to isolate critical paths. Our method can be used much more generally, as we do not change the original circuit's structure. *Slack optimizer* in [11] [12] uses switching activity from *SAIF* (switching activity interchange format) to extract frequently-exercised paths and calculate error rate. However, *SAIF*-based estimation is pessimistic and has limited accuracy. We use not only switching activity but also functional simulation information directly. So, we can identify the critical paths and estimate error rates precisely. *Slack optimizer* may also become stuck at a local optimum during voltage scaling-based optimization. However, by finding a minimum-power design at each voltage, our heuristic affords better global results.

Sensitivity-based Cell Sizing. Sensitivity-based downsizing approaches have been proposed in [4], [16], [17], [8], [7] and [9]. *TILOS* [4] 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*. $\Delta L(T)$ and $\Delta D(T)$ represent the change of leakage and delay for a resized transistor T. The techniques proposed in [17] use the same sensitivity function as $TILOS$.

$$Sensitivity(T) = \Delta L(T)/\Delta D(T) \tag{1}$$

For the cell sizing in [8], all cells are sorted in decreasing order of $\Delta leakage \times slack$ (Equation (2)), where $\Delta leakage$ is the improvement in leakage after a cell is replaced with its less leaky variant, and *slack* is its timing slack after the replacement has been made.

$$Sensitivity = \Delta leakage \times slack \tag{2}$$

The techniques proposed in [7] and [9] use sensitivity-based *downsizing* (i.e., begin with all nominal cell variants and replace cells on non-critical paths with long channel-length–variants) heuristics for leakage optimization. In their heuristics, they defined P_p as the sensitivity associated with cell instance p.

$$P_p = (\ell_p - \ell'_p)/(s_p - s'_p) \tag{3}$$

In Equation (3), s_p represents the slack of a given cell instance p, and s'_p represents the slack of p after downsizing. ℓ_p and ℓ'_p indicate the initial and final leakage values of cell instance p before and after downsizing, respectively. The sensitivities P_p are computed for all cell instances p. The heuristics of [7],[9] select a cell with the largest sensitivity and perform downsizing with a logically equivalent cell. If there is no timing violation in incremental STA, this move is accepted and saved.

3. HEURISTIC DESIGN

3.1 An 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-exercised circuit paths (*OptimizePaths*) , and leakage power reduction achieved by downsizing cells in non-critical 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.

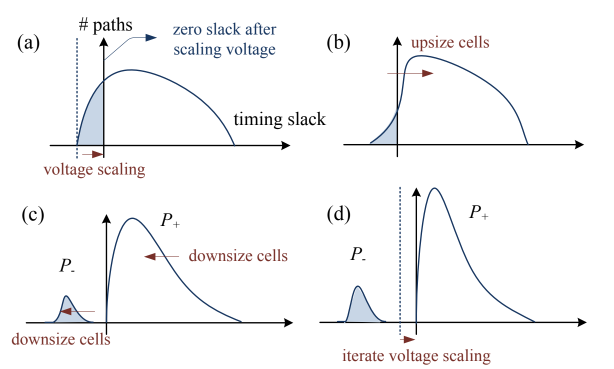

Figure 2: The power minimization heuristic reshapes the path slack distribution by redistributing slack from paths that rarely toggle to paths that toggle frequently.

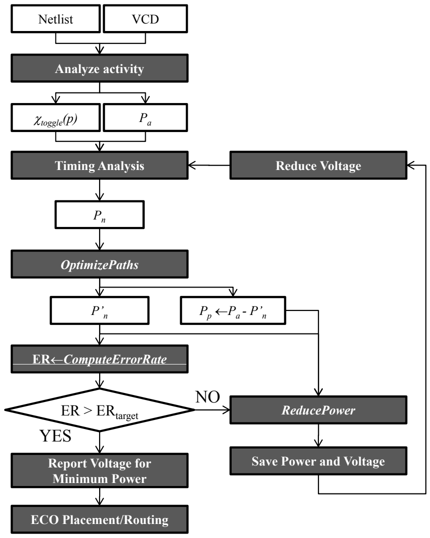

Figure 3: Algorithmic flow of a heuristic for minimizing power for a target error rate. P_a is the set of all paths toggled during simulation. P_p is the set of all non-negative-*slack* paths. P_n is the set of all negative-slack paths in P_a. $\chi_{toggle}(p)$ is the set of cycles in which path p is toggled.

Figure 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 (a). After partitioning the paths into sets containing positive and negative-slack paths, *OptimizePaths* attempts to reduce the error rate by increasing timing slack on negative-slack paths (b). Next, the heuristic allocates the error rate budget by selecting paths to be added to the set of negative-slack paths, and downsizes cells to achieve area / power reduction while respecting the partition between positive- and negative-slack paths (c). This cycle is repeated over the range of voltages to find the minimum power netlist and corresponding voltage (d). In Figure 2, P_+ is a set of paths that must have non-negative slack after power reduction, and P_- is a set of paths that are allowed to have negative slack.

Figure 3 gives 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*).

3.2 Heuristic Procedures

Path Optimization. The goal of the path optimization procedure (*OptimizePaths*) presented in Algorithm 1 is to minimize the error rate at a voltage level by transforming negative slack paths into non-negative-slack paths. This is accomplished by performing cell swaps that upsize cells in the negative slack paths and increase the path slack. Negative slack paths with maximum toggle rates are selected first during optimization, since they have the most potential to reduce the error rate if converted into positive-slack paths.

When a path is targeted for optimization, upsizing cell swaps are attempted on all cells in the path to increase slack as much as possible until non-negative path slack is achieved. Once a cell has been visited during optimization, it is marked to prevent degradation of timing slack on any paths that the cell is in. 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 paths, 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 toggle rates, and thus higher priority of optimization. If cell upsizing on a path fails to shift the path back into the set of positive slack paths, then the path is ignored during subsequent path optimization iterations.

$$Sensitivity(c) = \frac{s_c - s_{c'}}{w_c - w_{c'}} \quad , where \;\; w_c = w_{stat}(c) + w_{dyn}(c) \qquad (4)$$

Any cell swap that increases the error rate (by causing a path to switch from the set of positive 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.

Algorithm 1 Pseudocode for *OptimizePaths* and *ReducePower*.

```
Procedure OptimizePaths(P, N_Vi, V_i)
Clear 'visited' mark in all cells in the netlist N_Vi;
while P ≠ ∅ do
    Select path p from P with maximum toggle rate;
    for each cell c in path p do
        if c.visited == true then continue;
        c.visited ← true;
        for each logically equivalent cell m for the cell instance c do
            Resize cell c with logically equivalent cell m;
            Q ← c ∪ visited fanin and fanout cells of c;
            for each path q in P that contains a cell in Q do
                if Δslack(q, c, m, V_i) < 0 then restore cell change;
            end for
        end for
    end for
    P ← P − p;
end while

Procedure ReducePower(P_p, P_n, N_Vi, V_i, ER_target)
P+ ← P_p and P− ← P_n;
while P+ ≠ ∅ do
    Select path p from P+ with minimum ΔER(p);
    ER ← ComputeErrorRate(P− + p);
    if ER ≤ ER_target then
        P− ← P− + p;   P+ ← P+ − p;
    else
        break;
    end if
end while
Insert all downsizable cells into set C;
ComputeSensitivity(C, N_Vi, V_i, −1);
while C ≠ ∅ do
    Downsize cell c from C with minimum Sensitivity(c);
    Q ← c ∪ fanin and fanout cells of c;
    for each path p in P+ that contains a cell in Q do
        if slack(p, V_i) < 0 then
            Restore cell change;
            C ← C − c;
            continue while loop;
        end if
    end for
    ComputeSensitivity(Q, N_Vi, V_i, −1);
    if cell c is not downsizable then
        C ← C − c;
    end if
end while
```

Power Reduction. After path optimization, the error rate 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 error rate budget. Algorithm 1 (*ReducePower*) describes our power reduction procedure. The goal of the power reduction heuristic is to efficiently allocate the remaining error budget to dormant paths in order to maximize power reduction achieved by cell downsizing. Typically, cells on negative-slack paths can be downsized to a greater extent, 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 error rate of the circuit is matched. Paths are selected in order to minimize the additional contribution to the error rate of the circuit. After defining the partition between negative and non-negative-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 Equation 4 as the change in cell slack (Δs_c) divided by the change in cell power (Δw_c) when the cell is downsized by one size. The slack of a cell, c, is defined as the minimum slack on any timing arc containing c. This formulation of sensitivity is similar to those proposed by previous works targeting leakage power reduction [7, 9].

3.3 Path Extraction and Error Rate Estimation

Path Extraction. Our heuristic has many path-based procedures – *OptimizePaths*, *ReducePower*, and *ComputeErrorRate* – and it is impossible to consider all of the topological paths in these procedures. Therefore, we reduce the number of paths we consider by extracting paths that are 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 v6.1* [19]. Figure 4 shows an example VCD file and the path extraction method. The VCD file contains a list of toggled nets in each cycle time, as well as their new values. We can use this information to extract truly toggled paths in each cycle. Changed nets 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 Figure 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 $b-y$.

Toggle Rate and Error Rate Estimation. In order to accurately minimize power for a target error rate, we must be able to produce accurate estimates for error rate during our optimization flow. Thus, we propose a novel approach to error rate estimation that enables design for a target error rate.

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

$$TR(p) = |\chi_{toggle}(p)|/X_{tot} \quad (5)$$

where $|\chi_{toggle}(p)|$ 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 error rate precisely. The error rate (ER) of the design is calculated as:

$$ER = \frac{|\bigcup_{p \in P_n} \chi_{toggle}(p)|}{X_{tot}} \quad (6)$$

where P_n is the set of negative-slack paths in the set of all toggled paths. In Figure 4, if paths $a-x-y$ and $b-y$ both have a toggle rate 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 error rate is 0.4 for this example.

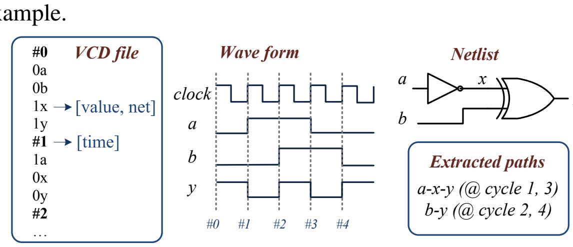

Figure 4: VCD file format and path extraction.

3.4 Heuristic Design Choices

In this section, we discuss heuristic design choices.

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. This order matters also because we protect previously optimized paths from slack degradation from other attempted upsizing moves, since 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 toggle rate, $TR(p)$. Paths with the highest toggle rates have the greatest potential to decrease error rate 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 less negative slack, since these paths can be converted into non-negative-slack paths with least upsizing.

Experiment 2: Optimization Radius. The goal of optimization is to maximize the slack of a targeted path through cell upsizing. We evaluate two alternatives for the radius of optimization. In one case, we only target cells on a given path for upsizing. 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.

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.

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

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

Experiment 5: Error Rate Budget Utilization. During power reduction, the final error rate after cell downsizing could be less than the target error rate, ER_{target}, since some paths in P_- might still have non-negative 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 non-negative slack after cell downsizing).

Experiment 6: Starting Netlist. Here, we evaluate heuristic performance for different starting netlists corresponding to loose and tight timing constraints. This can significantly affect both the final voltage reached and the amount of power savings afforded by the power minimization algorithm

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} – 0.01V and 0.05V – and compare the characteristics of the final netlist as well as the heuristic runtime.

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

4. EXPERIMENTAL RESULTS

For our experiments, we use eight modules (Table 1) of the *OpenSPARC T1* processor [18]. Module designs are implemented with a TSMC 65GP library (65nm) and the initial netlists are synthesized with *Synopsys Design Compiler vY-2006.06-SP5* [20].

4.1 Evaluation of Error Rate Estimation

Accurate error rate estimation is critical to achieving power minimization when designing for a target error rate. Inaccurate estimation can lead to over- or under-optimization. Figure 6 compares the error rate estimation approach proposed by this work against the result computed during functional simulation, and an estimator used by the slack optimization heuristic in [11] [12].

Our estimation technique compares favorably against the previously proposed estimator, and matches well with actual error rate. Root mean squared error for our technique is 0.1575 as opposed to 0.6002 for the technique used in the slack optimizer. Figure 5 compares the runtime of our estimation technique with that of actual simulation demonstrating over an order of magnitude decrease.

Table 1: Target modules for experiments.

Module	Stage	Description	Cell #	Area(um^2)
lsu_dctl	MEM	L1 Dcache Control	4537	13850
lsu_qctl1	MEM	LDST Queue Control	2485	7964
lsu_stb_ctl	MEM	ST Buffer Control	854	2453
sparc_exu_div	EX	Integer Division	4809	14189
sparc_exu_ecl	EX	Execution Unit Control	2302	7089
sparc_ifu_errdp	FD	Error Datapath	4184	12972
sparc_ifu_fcl	FD	L1 Icache and PC Control	2431	6457
spu_ctl	SPU	Stream Processing Control	3341	9853

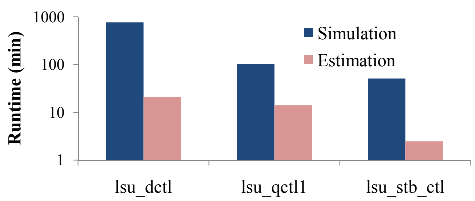

Figure 5: Runtime comparison of actual and estimated error rates.

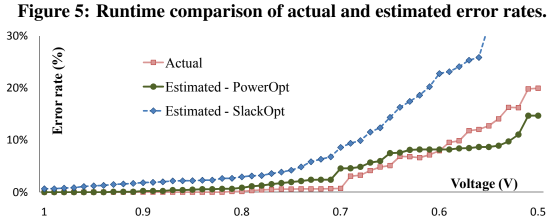

Figure 6: Actual error rate from functional simulation vs. estimated error rate for *lsu_stb_ctl*. Estimated error rate is also compared against an estimator used in [11] demonstrating much better correlation with actual error rate.

4.2 Evaluation of Heuristic Design Choices

Figure 7 shows power and runtime of the various heuristic design alternatives that we evaluated. For **path ordering during optimization**, considering the slack in the prioritization function results in higher power than the case where only toggle rate 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 toggle rate priority function for its simplicity and lower power.

The results for **optimization radius** show that swapping cells in the fanin/fanout network not only increases power at some error rates, 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 **error rate budget utilization**, we observe no difference in power. Both updating the error rate contribution continuously during path selection and ensuring full utilization of the error rate budget increase runtime significantly without providing power benefits, and these techniques are not used in the final heuristic implementation.

Choice of **starting netlist** and **voltage step size** have significant effects on power. Our heuristic for power minimization employs two main procedures – *OptimizePaths* (cell upsizing to reduce the error rate) and *ReducePower* (cell downsizing to reduce area and

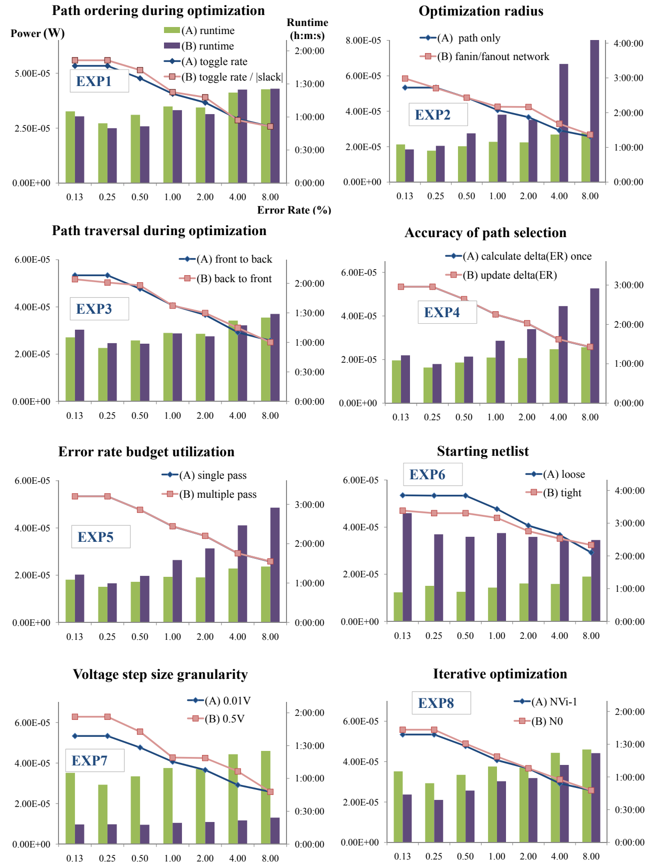

Figure 7: Evaluation of different heuristic design choices. The choices are evaluated in terms of power of the resulting design as well as runtime.

power). When starting the optimization flow from a loosely constrained design, we observe that path optimization provides the most substantial contribution to power reduction by reducing the error rate 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 to power minimization. Overall, a tightly constrained netlist provides a better starting point since it permits more voltage scaling, which has a larger effect on power reduction, since power of all cells scales rather than only the downsized cells.

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 error rates, a large step size can provide a near-optimal power result and a large reduction in runtime. Thus, error rate-aware step sizing can be beneficial.

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.

4.3 Comparison Against Alternative Flows

To demonstrate of the benefits of our power minimization design flow, we compare five alternative design flows – traditional P&R implementations with tight and loose timing constraints, slack optimization [11] [12], and our heuristic for power optimization. Figure 8 compares the power consumptions of the various design techniques at several target error rates.

After deciding how to allocate the error rate budget, the *ReducePower* stage of our power minimization heuristic performs aggressive cell downsizing to reduce circuit area and power. Table 2 compares design for a target error rate against other design flows in terms of area overhead with respect to the baseline design. Notice that design for a target error rate has higher area overhead than slack optimization but still produces a design with less power. The reason is that designing for a target error rate allows more aggressive voltage scaling before the target error rate is exceeded. At lower voltages, there are more negative slack paths to be optimized during *OptimizePaths*, resulting in more area overhead. However, the area is well spent, since the additional voltage scaling it enables contributes to a net win in terms of power savings. Comparing this result with tightly constrained P&R and *BlueShift* reveals that designing for a target error rate focuses on the best set of paths to optimize, while techniques that spend a comparable amount of area fail to identify the best set of paths and end up increasing power.

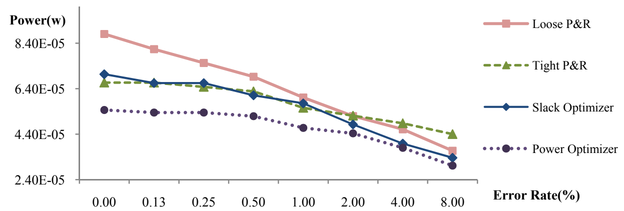

Figure 8: Power consumption of each design technique at various target error rates for *lsu_stb_ctl*.

Table 2: Average area overhead with respect to the baseline.

Tight P&R	*BlueShift*	SlackOpt	PwrOpt 0.125%	PwrOpt 0.25%
25.9%	11.8%	3.7%	7.7%	9.9%
PwrOpt 0.5%	PwrOpt 1%	PwrOpt 2%	PwrOpt 4%	PwrOpt 8%
9.8%	11.2%	13.9%	12.2%	12.8%

4.4 Evaluation of Recovery-driven Design for Hardware-based Error Tolerance

The objective of recovery-driven design is to increase the power efficiency of a BTWC design by employing functional information to optimize for errors during normal operation. These errors can be gainfully tolerated by an error tolerance mechanism in hardware or software, and since our design flow optimizes for the most power efficient way to allow a particular error rate, we observe a power benefit when we design for a target error rate.

Figure 9 compares power consumption of design for a target error rate against other design techniques when Razor [3] is used to detect and correct errors in the *lsu_stb_ctl* circuit. The figure shows that design for a target error rate that is efficient for Razor enables extended voltage scaling (0.75V vs. 0.82V) before recovery overhead becomes too high, affording an additional 21% power savings.

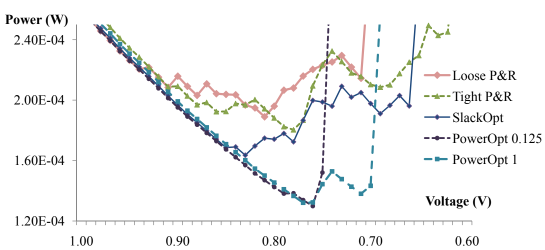

Figure 9: Power consumption for *lsu_stb_ctl* using Razor for error detection and correction. Recovery-driven design enables extended voltage scaling for additional power benefits.

4.5 Design Considerations for Recovery-driven Processors

The heuristic presented in Section 3.1 optimizes a circuit module for a target error rate. This heuristic can potentially be expanded to provide power savings for processor designs that consist of many modules. Such a processor-level heuristic would be well-suited for design of stochastic processors [15, 13, 14, 12]. However, in order to apply the power optimization heuristic to a large-scale processor design, several considerations must be addressed.

The power optimization heuristic minimizes the power of a circuit module for a target error rate. Since a processor consists of many modules, a processor-level heuristic would need to find the most efficient way to assign target error rates to modules in order to achieve the desired error rate target for the processor. Similarly, a processor-level heuristic must decide which modules should be optimized to allow errors and which modules should be optimized for correct operation. The module-level heuristic selects an optimal operating voltage for each module. Since the optimal voltage may be different for each module, a processor-level heuristic should also be able to select the common operating voltage that minimizes processor power. These considerations, along with the design of recovery-driven processors, are the subject of ongoing work.

5. SUMMARY AND CONCLUSION

In this paper, we have proposed *recovery-driven design*, a design approach that optimizes a processor module for a target timing error rate instead of correct operation. We have presented a detailed evaluation and analysis of a recovery-driven design methodology to minimize processor power for a target error rate. We demonstrate that such a design flow can result in significant power savings – up to 25%, 19%, 22%, 24%, 20%, 28%, and 20% compared to traditional P&R at error rates of 0.125%, 0.25%, 0.5%, 1%, 2%, 4%, and 8%, respectively. Additional power savings are possible for recovery-driven designs that employ error tolerance. We have demonstrated up to 21% additional power savings for a recovery-driven design flow that uses Razor to correct timing violations. To support our recovery-driven design flow, we have presented a fast, novel, and accurate technique for post-layout activity and error rate estimation that can significantly extend the range of allowable voltage scaling for a given target error rate.

6. REFERENCES

[1] T. Austin, V. Bertacco, D. Blaauw and T. Mudge, "Oppotunities and Challenges for Better Than Worst-Case Design", *Proc. ASPDAC*, 2005, pp. 2–7.

[2] 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", *Proc. DATE*, 2006, pp. 1110-1115.

[3] 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", *Proc. IEEE/ACM MICRO*, 2003, pp. 7–18.

[4] J. P. Fishburn and A. E. Dunlop, "Tilos: A Polynomial Programming Approach to Transistor Sizing", *Proc. IEEE ICCAD*, 1985, pp. 326–328.

[5] 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. on CAD* , 2007.

[6] 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", *Proc. IEEE HPCA*, 2009, pp. 213–224.

[7] P. Gupta, A. B. Kahng and P. Sharma, "A Practical Transistor-Level Dual Threshold Voltage Assignment Methodology", *Proc. ISQED*, 2005, pp. 421–426.

[8] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Selective Gate-Length Biasing for Cost-Effective Runtime Leakage Control", *Proc. ACM/IEEE DAC*, 2004, pp. 327–330.

[9] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Gate-Length Biasing for Runtime-Leakage Control", *IEEE Trans. on CAD*, 25(8), 2006, pp. 1475–1485.

[10] R. Hegde and N. R. Shanbhag, "Energy-Efficient Signal Processing via Algorithmic Noise-Tolerance", *Proc. ISLPED*, 1999, pp. 30–35.

[11] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Slack Redistribution for Graceful Degradation Under Voltage Overscaling", *Proc. ASPDAC*, 2010, pp. 825–831.

[12] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Designing a Processor From the Ground Up to Allow Voltage/Reliability Tradeoffs", *Proc. IEEE HPCA*, 2010, pp 119–129.

[13] R. Kumar, "Stochastic Processors", *NSF Workshop on Science of Power Management*, March 2009.

[14] S. Narayanan, J. Sartori, R. Kumar and D. Jones, "Scalable Stochastic Processors", *Proc. DATE*, 2010.

[15] N. Shanbhag, R. Abdallah, R. Kumar and D. Jones, "Stochastic Computation", *Proc. ACM/IEEE DAC*, 2010.

[16] 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. VLSI Systems*, 10(2), 2002, pp. 79–90.

[17] A. Sultania, D. Sylvester, and S. S. Sapatnekar, "Tradeoffs between Gate Oxide Leakage and Delay for Dual T_{ox} Circuits", *Proc. ACM/IEEE DAC*, 2004, pp. 761–766.

[18] *Sun OpenSPARC Project.*, http://www.sun.com/processors/opensparc/.

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

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

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