A New Methodology for Reduced Cost of Resilience

Andrew B. Kahng[†‡], Seokhyeong Kang[†] and Jiajia Li[†]
[†]ECE and [‡]CSE Departments, University of California at San Diego
La Jolla, CA, 92093
abk@ucsd.edu, shkang@vlsicad.ucsd.edu, jil150@ucsd.edu

ABSTRACT

Resilient design techniques are used to (i) ensure correct operation under dynamic variations; and (ii) improve design performance (e.g., through *timing speculation*). However, significant overheads (e.g., 17% and 15% energy penalties due to throughput degradation and additional circuits) are incurred by existing resilient design techniques. For instance, resilient designs require additional circuits to detect and correct timing errors. Further, when there is an error, the additional cycles needed to restore a previous correct state degrade throughput, which diminishes the performance benefit of using resilient designs. In this work, we propose a methodology for resilient design implementation to minimize the costs of resilience in terms of power, area and throughput degradation. Our methodology uses two levers: selective-endpoint optimization (i.e., sensitivity-based margin insertion) and clock skew optimization. We integrate the two optimization techniques in an iterative optimization flow which comprehends toggle rate information and the tradeoff between cost of resilience and margin on combinational paths. Our proposed flow achieves energy reductions of up to 19% and 21% compared to a conventional design (with only margin used to attain robustness) and a brute-force implementation, respectively. These benefits increase in the context of an adaptive voltage scaling strategy.

Categories and Subject Descriptors: B.7.2 [Design Aids]: Placement and routing
Keywords: Low power, resilient design, design optimization, cost reduction

1. INTRODUCTION

IC products in advanced technology nodes are susceptible to dynamic variations that manifest via supply voltage droop, temperature fluctuation, cross-coupling, aging, and other mechanisms. To ensure correct functionality and robustness, traditional IC implementation methodologies build guardband into clock frequencies and design signoffs – notably, timing signoff at worst-case corners and for hold-time correctness. However, it is well-recognized that designing for worst-case conditions incurs considerable power and performance overheads. *Better Than Worst-Case design* [3], where an error checker and corresponding recovery mechanism enable typical-case optimization, can significantly reduce overdesign compared to traditional methodologies. A similar idea for guardband reduction has been proposed by Bowman et al. in [5], where several techniques for dynamic variation tolerance (i.e., resilient designs) are presented.

Resilient designs trade off design robustness against design quality (performance, power and area), and are used to ensure correctness against variation and improve performance [7] [9] [10] [12] [13] [18]. *Razor* [10] is a well-known technique to detect and correct timing errors due to frequency, temperature and voltage variations. Razor detects timing violations by supplementing error-tolerant flip-flops with *shadow latches*. A shadow latch strobes the output of a logic stage at a fixed delay after the main flip-flop; if a timing violation occurs, the main flip-flop and shadow latch will have different values, signaling the need for correction. Correction involves recovery using the correct value(s) stored in the shadow latch(es). In the following discussion, we define the maximum timing violation (worst negative slack) that a resilient design can tolerate as the *safety margin* of the corresponding design.

GLSVLSI'14, May 21–23, 2014, Houston, Texas, USA.

http://dx.doi.org/10.1145/2591513.2591600.

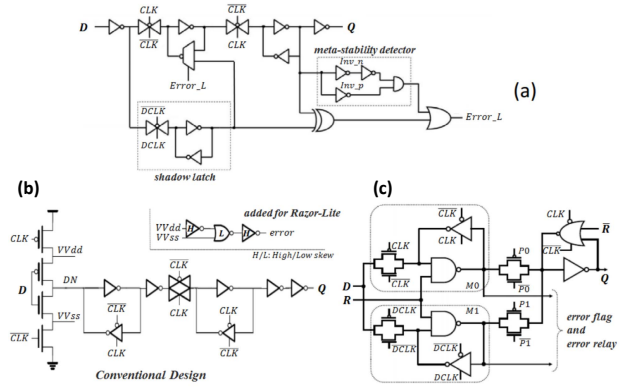

Figure 1: Structure of (a) Razor, (b) Razor-Lite and (c) TIMBER flip-flops.

By allowing timing errors, resilient designs are also used to improve performance. An example is *timing speculation* [19], which increases the clock frequency and exploits error detection and recovery mechanisms to correct the resulting errors. Timing improvement from resilient designs can further lead to power and area benefits over conventional designs. In other words, we can reduce the power and area of logic cells in a fanin cone by using the error-tolerant register at the endpoint.

However, resilient designs require additional circuits or cycles to detect and correct timing errors. Figure 1 shows the structure of Razor, Razor-Lite [17] and TIMBER [7] flip-flops. All have additional circuits, and hence power and area overheads, compared to a conventional flip-flop. For instance, Razor has its shadow latch and other error-tolerant circuits (comparator, multiplexer and OR-gate). When compared to a conventional flip-flop, the total power overhead of Razor flip-flop is 30% [9]. Although the power overhead has been significantly reduced in a recent work [17], the additional cycles needed to recover from errors can still lead to performance degradation. Moreover, error-tolerant circuits are vulnerable to hold violations. Designers must ensure that benefits (in terms of performance, and/or area and power reduction from the error resilience) outweigh the additional costs of error-tolerant circuits.

In this work, we perform in-depth studies of the tradeoff between the overhead of error-tolerant circuits and the cost of the traditional timing optimizations, with the goal of assessing 'true' benefits of resilient design techniques. We propose two effective design optimization techniques – *selective-endpoint optimization*, and *clock skew (useful skew) optimization* [1] [11] – to minimize the costs of resilience, i.e., (i) power and area overhead of resilient circuits, and (ii) throughput degradation due to additional cycles for error recovery. Since our work currently focuses on optimization at the post-placement stage, we do not yet consider the cost of hold

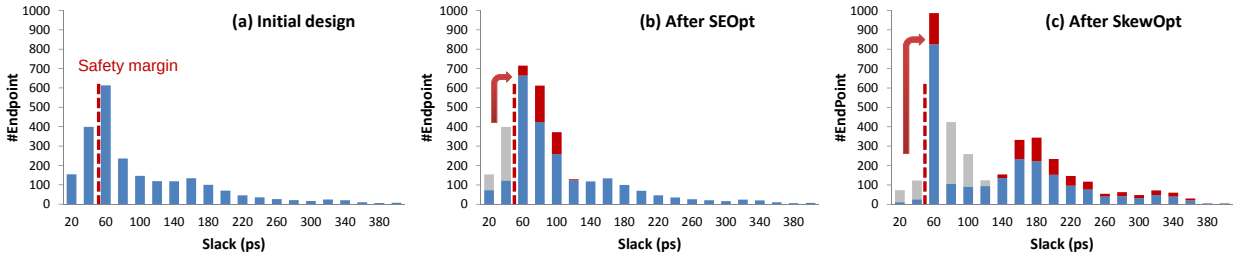

Figure 2: Slack distribution of endpoints in (a) original design; (b) design with only selective-endpoint optimization; and (c) design with combined selective-endpoint and useful skew optimization. Red dotted lines indicate required safety margin. Design: FPU (OpenSPARC T1). Technology: 28nm FDSOI.

violations due to resilient design implementation. However, our optimization flow can easily be combined with existing short-path padding optimizations (e.g., [20]). Our contributions include the following.

- We propose an optimization methodology for resilient designs to reduce the cost of resilience. Our method exploits both error-tolerant registers and clock skew scheduling.
- We study the benefits and cost of resilient design implementations, where we trade off among (i) power and area overheads of error-tolerant registers, (ii) optimization of logic cells in the fanin cone, and (iii) throughput degradation due to timing errors.
- We assess the opportunities of resilient designs across different error-tolerant registers designs as well as in the adaptive voltage scaling (AVS) context.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 formulates the problem to reduce the cost of resilience and describes our methodology for implementing low-cost resilience. Section 4 presents our experimental results and analysis, and Section 5 summarizes and concludes the paper.

2. RELATED WORK

A number of resilient design techniques have been proposed that allow timing errors with different error detection and correction mechanisms. These previous works can be roughly classified into two categories. In the first category, designs use replica circuits for error masking. These designs typically incur large power and area overheads due to its additional circuits. In the second category, designs use error-tolerant registers to detect timing errors. Although circuit power and area overheads can be smaller, rollback or instruction reply is required to recover from timing errors. The additional cycles for error recovery lead to throughput degradation.

Replica Circuits for Error Masking. A well-known technique compares output values in each cycle using redundant hardware circuits. *Paceline* [13] employs a *leader-checker* which checks timing errors due to overclocking. *CPipe* [18] enables reliable overclocking through core-replication. The outputs of the main combinational logic are compared with those of the duplicated logic in each cycle. Choudhury et al. [8] synthesize error-masking circuits and use 2-to-1 multiplexers to mask errors at the output of critical paths. Similarly, Yuan et al. [22] mask errors by adding redundant approximation logic which has higher speed than the original circuit. *TIMBER* flip-flops and latches [7] enable online timing error masking via time-borrowing from the successive pipeline stage, and hence do not require additional cycles to recover from an error. This kind of approach provides error resilience with high reliability, but also incurs significant power and area overheads due to the redundant logic circuits.

Error-Tolerant Registers with Error Recovery. Razor and related works [4] [9] [10] [17] replace registers with specialized flip-flops which detect and correct timing errors on each endpoint by capturing the correct value at shadow latches with a delayed clock. Razor [10] can correct timing errors within a specific safety margin of the error-tolerant register. *Razor II* [9] provides analysis of the Razor flip-flop – with respect to timing constraints, safety margin and clocking scheme – and reduces complexity and area of the Razor flip-flop. A more recent work – *Razor-Lite* [17] – further reduces the area and power penalties of error-tolerant registers. *STEM* [4] improves the capability of error-detection with a second shadow latch.

Resilient Design Optimization. With the above error-tolerant registers, various design-level optimization techniques [8] [14] [15] [16] [19] [22] have been proposed which identify and optimize critical paths that are frequently exercised during operation. However, these works typically fail to holistically consider the costs of the error-tolerant circuits during the optimization. For example, Choudhury et al. [8] reduce the area and power penalties of resilient designs. However, their method simply applies resilient techniques to timing-critical paths, and ignores the tradeoff between the benefits of resilience and the costs of margin insertion for data paths.

3. IMPLEMENTATION METHODOLOGY

In this section, we define a resilience cost reduction problem and describe our optimization flow for low-cost resilient design implementation. Our flow uses two optimization techniques – *selective-endpoint optimization* (SEOpt) and *clock skew optimization* (SkewOpt) – to minimize resilience overheads of energy, area and throughput degradation. Figure 2 illustrates the basic idea of our optimization approach. In the initial resilient design (a), a large number of endpoints have timing violations at the target frequency (with respect to the safety margin), and error-tolerant registers or error-masking circuits are used for those endpoints. In our selective-endpoint optimization (b), we tightly optimize a set of selected endpoints to reduce the resilience overheads. During clock skew optimization (c), we increase timing slacks of endpoints having timing violations by optimizing the clock-arrival time at individual endpoints, further reducing the resilience overheads. In our optimization flow, we iteratively perform SEOpt and SkewOpt to minimize the cost of resilient design.

3.1 Resilience Cost Reduction Problem

We solve the following **resilience cost reduction problem**. Given an RTL design along with (i) throughput requirements, (ii) power and area overheads as well as safety margin for each type of error-tolerant register, and (iii) number of cycles needed to recover from an error: implement the design to attain minimum energy, comprehending the energy penalties of additional circuits and the throughput degradation due to rollback or instruction replay.

We calculate design energy based on total power and throughput information, i.e.,

$$Energy = \frac{Power}{TP} \tag{1}$$

where TP is the throughput of the design. TP is estimated based on error rate information [19] as

$$TP = \frac{1-ER}{T} + \frac{ER}{r \cdot T} \tag{2}$$

where ER is the total error rate of the design, T is the clock period, and r is the number of cycles needed to recover from an error. Thus, for an accurate design, the throughput is $1/T$.

We further estimate the error rate based on toggle information of flip-flops (including toggles of both negative-slack and positive-

slack fanin paths) [16] as

$$ER = \alpha \cdot \frac{\sum(TG_{ff} \cdot \frac{\sum TG_{p\_neg}}{\sum TG_{p\_all}})}{\sum TG_{ff}} \quad (3)$$

where TG_{ff} is the toggle rate of a flip-flop, TG_{p_neg} and TG_{p_all} are respectively the toggle rates of negative-slack fanin paths and all fanin paths to the flip-flop, and α is a parameter to compensate pessimism due to (i) the fact that errors can occur in one cycle and (ii) the existence of false paths. We empirically use $\alpha = 0.35$ in our experiments.

3.2 Selective-Endpoint Optimization

We propose *selective-endpoint optimization* (SEOpt) to minimize the resilient design cost (primarily area, power and throughput degradation). Our SEOpt trades off between the costs of resilience and of data path optimization. In other words, we selectively increase margins at the endpoints with timing violations; this allows us to replace the error-tolerant registers with conventional ones and/or to remove replica circuits. However, these margins incur area and power cost in combinational logic cones. Therefore, key questions are (i) '*which endpoints should be optimized?*', and (ii) '*how many endpoints should be optimized?*'.

For Question (i), area and power of combinational cells in the fanin cone of an endpoint will increase when we add slack margin for the endpoint. Further, each endpoint will exhibit a different cost function due to the margin insertion. For instance, the optimization cost increases significantly for an endpoint which has a large number of timing-critical fanin cells (i.e., negative-slack cells in the fanin cone of the endpoint). Therefore, to reduce the optimization cost, we should preferentially optimize endpoints which are less sensitive to slack margin insertion. In SEOpt, we propose sensitivity functions for endpoints to estimate the potential optimization cost, based on which we select endpoints for optimization. Note that the sensitivity function of an endpoint indicates the performance vs. power and/or area tradeoff of the corresponding fanin cone. We study five sensitivity functions for a given timing endpoint p:

$$SF1(p) = |slack(p)| \quad (4)$$

$$SF2(p) = |slack(p)| \times num_{cri}(p) \quad (5)$$

$$SF3(p) = |slack(p)| \times \frac{num_{cri}(p)}{num_{total}(p)} \quad (6)$$

$$SF4(p) = |slack(p)| \times \sum_{c \in fanin(p)} Pwr(c) \quad (7)$$

$$SF5(p) = \sum_{c \in fanin(p)} (|slack(c)| \times Pwr(c)) \quad (8)$$

$slack(p)$ indicates the worst negative slack of endpoint p; $num_{cri}(p)$ and $num_{total}(p)$ respectively indicate the number of critical cells (i.e., cells with negative timing slacks) and total cell count in the fanin cone of the endpoint p; c indicates the combinational cells in the fanin cone; and $slack(c)$ and $Pwr(c)$ are respectively the worst negative slack of any path through cell c and the power of cell c.

To study the performance of each sensitivity function, we sort the endpoints in increasing order of a given sensitivity function. Then, we optimize the top k% endpoints in the sorted list, where we increase k from 0 to 100 with a step size of 5. Figure 3 shows power and area resulting from selective-endpoint optimizations based on five sensitivity functions. In this example, the safety margin is 10% of the clock period.[1] We observe that SEOpt based on SF2 and SF5 incurs smaller penalties with respect to power and area. We use SF5 in the experiments reported in Section 4.

[1] In [7], safety margins of 10%, 20% and 30% of clock period are studied.

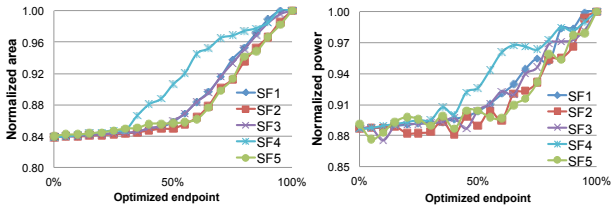

Figure 3: Cell area and total power resulting from selective-endpoint optimization with different sensitivity functions.

For Question (ii), optimizing more endpoints reduces the number of error-tolerant registers required. However, the cost of this optimization (i.e., area and power penalty on data paths) also increases. We iteratively increase the number of endpoints to be optimized and select the solution with minimum cost (e.g., a function of area and/or power).

3.3 Clock Skew Optimization

To further reduce the number of error-tolerant registers and minimize the timing errors, we use *clock skew optimization* (SkewOpt) which maximizes the timing slacks on endpoints with timing violations. In SkewOpt, we formulate the clock skew optimization problem as a maximum mean weight cycle problem [2]. This is due to the fact that the maximum achievable timing slack of a path is determined by the maximum average slack of a cycle (i.e., a loop formed by timing paths) which contains the corresponding path. We use the *parametric shortest path* algorithm [21] to determine the maximum mean weight cycle. The algorithm as we have implemented it is described in Algorithm 1. We first construct a graph G where each endpoint corresponds to a vertex and each timing path corresponds to two edges (i.e., one for the setup constraint and one for the hold constraint) (Line 1). The weights of edges indicate setup/hold slacks of timing paths in the corresponding flop-to-flop logic cones.

In SkewOpt, we optimize setup timing slacks of endpoints with error-tolerant registers (with respect to hold constraints and setup constraints on other paths). Based on the above, we classify edges in the graph into two categories – (i) *parameterized edges* and (ii) *non-parameterized edges* – where timing corresponding to parameterized edges will be optimized, while non-parameterized edges will serve as constraints during the optimization. We define parameterized edges based on setup constraints on timing paths having timing violations with respect to the safety margin, and we define non-parameterized edges based on hold/setup constraints on other paths. We formulate the constraints in SkewOpt as

$$x_q + \underbrace{(T - d_q - d^{max}_{p,q} - t^{setup}_q - t^{margin}_{p,q})}_{s_{p,q}} - \lambda \geq x_p \ (q \in R) \quad (9)$$

$$x_q + \underbrace{(T - d_q - d^{max}_{p,q} - t^{setup}_q - t^{margin}_{p,q})}_{s_{p,q}} \geq x_p \ (q \notin R) \quad (10)$$

$$x_p + \underbrace{(d_p - d^{min}_{p,q} - t^{hold}_q - d_q)}_{s_{p,q}} \geq x_q \ \ (\forall q) \quad (11)$$

where T is the clock period; x_p is the clock arrival time of endpoint p; d_p is the clock-to-Q delay of p; $d^{max}_{p,q}$ and $d^{min}_{p,q}$ are, respectively, the maximum and minimum path delay from p to q; t^{setup}_q and t^{hold}_q are the setup and hold times of q; and $t^{margin}_{p,q}$ is the required safety margin between p and q. R is the set of endpoints which use error-tolerant registers, and λ is the *parameter* which will indicate the slack change. Constraint (9) corresponds to a parameterized edge in the constructed graph with an edge weight of $(s_{p,q} - \lambda)$. Constraints (10) and (11) are respectively induced by setup and hold constraints on a given non-parameterized edge in the constructed graph with an edge weight of $s_{p,q}$.

Algorithm 1 Clock Skew Optimization (SkewOpt)

Procedure $SkewOpt(N)$
1. $G(V, E) \leftarrow$ construct graph corresponding to N
2. Initialize solution graph $G'(V, \emptyset)$
3. $V \leftarrow \{r\} \cup V$; $E \leftarrow \{e(r,p)\} \cup E$, $\forall p \neq r$; $w(r,p) \leftarrow 0$, $\forall p \neq r$
4. $E_T \leftarrow \{e(r,p)\}$, $\forall p \neq r$
5. Update $p\_w(p)$, $\forall p \in V$
   // $p\_w(p) = \sum w(p_i, p_j)$, $\forall e(p_i, p_j) \in$ shortest path from r to p
6. **while** $|E| > 1$ **do**
7. $\quad \lambda_{min} \leftarrow +\infty$
8. $\quad$ **for all** $p \in E$ but $\notin E_T$ **do**
9. $\quad\quad \lambda_{p,q} \leftarrow$ Solve $p\_w(p) + w(r,q) = p\_w(q)$
10. $\quad\quad$ **if** $\lambda_{p,q} < \lambda_{min}$ **then**
11. $\quad\quad\quad \lambda_{min} \leftarrow \lambda_{p,q}$
12. $\quad\quad\quad e_{min} \leftarrow e(p,q)$
13. $\quad\quad$ **end if**
14. $\quad$ **end for**
15. $\quad E_T \leftarrow E_T \cup \{e(p,q)\}$
16. $\quad \lambda \leftarrow \lambda_{min}$
17. $\quad$ Remove edges from E_T with the same head as e_{min}
18. $\quad$ **if** there is a cycle in E_T **then**
19. $\quad\quad slack(p,q) \leftarrow \lambda_{min}$, $\forall e(p,q) \in cycle$
20. $\quad\quad$ Add all edges on cycle to G'
21. $\quad\quad E \leftarrow E \setminus \{e(p,q) \mid e(p,q) \in cycle\}$
22. $\quad\quad$ Contract all vertices on cycle into p_{new}
23. $\quad\quad$ Update E and E_T
24. $\quad$ **end if**
25. **end while**
26. Traverse G' to calculate x_q based on $slack(p,q)$ and x_p
27. $N_{sol} \leftarrow$ apply x_p, $\forall p$ to N
28. **return** N_{sol}

Algorithm 2 Combined Optimization (CombOpt)

Procedure $CombOpt(N)$
1. Run STA to initialize slack values for the netlist N
2. $P \leftarrow \emptyset$
3. **for all** timing endpoints p in the netlist N **do**
4. $\quad$ **if** $slack(p) < 0$ **then**
5. $\quad\quad p.sensitivity \leftarrow |slack(p)| \times fanin(p)$
6. $\quad\quad P \leftarrow P \cup \{p\}$
7. $\quad$ **end if**
8. **end for**
9. $m \leftarrow |P|/k$
10. $C_{min} \leftarrow \infty$
11. **for** $i = 0$; $i < m$; $i \leftarrow i+1$ **do**
12. $\quad$ Pick the top k endpoints P_i with minimum $sensitivity$ in P;
13. $\quad N_i \leftarrow TimingOpt(N_{i-1}, P_i)$
14. $\quad N_i \leftarrow SkewOpt(N_{i-1})$
15. $\quad$ Run incremental $STA(N_i, P_i)$
16. $\quad$ **for all** endpoint p in P **do**
17. $\quad\quad$ **if** $slack(p) \geq 0$ **then**
18. $\quad\quad\quad$ Replace error-tolerant register by conventional register at p
19. $\quad\quad$ **end if**
20. $\quad$ **end for**
21. $\quad C_i \leftarrow COST(N_i)$
22. $\quad$ **if** $C_i < C_{min}$ **then**
23. $\quad\quad C_{min} \leftarrow C_i$
24. $\quad\quad N_{min} \leftarrow N_i$
25. $\quad$ **end if**
26. $\quad P \leftarrow P - P_i$
27. $\quad$ Update $sensitivity$ of all endpoints in P
28. **end for**
29. **return** N_{min}

In the graph G, we always maintain a tree to store edges corresponding to timing-critical paths. We initialize the tree by inserting a dummy vertex (i.e., root r) and dummy edges connecting r and other vertices (Lines 3-4). Then, we continuously add edges corresponding to the most timing-critical paths to the tree (Lines 7-15) and remove dummy edges that share the same head with the added edge (Line 17). When adding an edge to the tree results in a cycle[2], we coalesce the cycle (including vertices and edges on the cycle) into one vertex (Lines 18-24). The edges on the cycle are added to the solution graph and the optimized slacks are stored. We assign to the parameterized edges weights equal to the summation of weights (i.e., slacks) on the cycle divided by the number of parameterized edges on the cycle, and assign zero slack to the non-parameterized edges on the cycle. That is, timing paths with conventional registers as endpoints will have zero slack with respect to the safety margin if they are in a maximum mean weight cycle with critical paths with error-tolerant registers as endpoints. Note that assigning new weights indicates the change of clock arrival times. Therefore, we update the weights of edges incident to vertices on the cycle. Then, we optimize slacks on the updated graph. We iteratively determine and optimize the most critical maximum mean weight cycle until there is only one edge in the graph (i.e., no more cycles can be determined). Last, we traverse the solution graph and calculate the clock arrival times based on the optimized path slacks (Line 26).

Although the above algorithm improves timing slacks on paths with timing-violated endpoints, it is not aware of error rates. To enable error-rate awareness and reduce the cost of throughput degradation, we extract the toggle rate information of each timing path and replace Constraint (9) by

$$x_q + \frac{s_{p,q}}{1 + \beta \cdot TG(p,q)} - \lambda \geq x_p \ (q \in R) \tag{12}$$

where $TG(p,q)$ indicates the toggle rate of the maximum-delay path between endpoints p and q, and β is a weighting factor (we set $\beta = 2$ in our experiments).

3.4 Proposed Optimization Flow

As mentioned in Section III-B, SEOpt reduces cost of resilience via optimization on data paths. However, such an optimization incurs power and area overheads on combinational cells, and its performance is limited on timing-critical paths (i.e., upsizing and buffer insertion cannot remove timing violations on timing-critical paths that have already been optimized with these levers). On the other hand, SkewOpt does not lead to power and area penalty on data paths, but its performance is limited by the total available timing slacks in the design (i.e., it only migrates timing slacks from timing non-critical paths to timing-critical paths, but cannot generate additional slacks) and the topology of the sequential graph (i.e., the benefits of SkewOpt are limited when there are many cycles consisting of timing-critical paths).

To minimize the cost of resilience, we combine the SEOpt and SkewOpt methods and execute them iteratively. The basic idea is that we use SEOpt to create timing slacks on data paths with low power penalty; then, we use SkewOpt to migrate the timing slacks on non-critical paths and the created timing slacks from SEOpt to critical-timing paths which have high-sensitivity endpoints. In this way, we reduce the number of error-tolerant registers without incurring large power penalty on data paths.

Algorithm 2 describes our combined optimization, which we call *CombOpt*, to reduce the error-resilience overhead. The procedure takes as input a netlist N which has error-tolerant registers on endpoints with timing violations. The procedure runs static timing analysis (*STA*) and computes a sensitivity value for each endpoint p (Lines 1-8). We use the sensitivity function SF5 as described above. The procedure finds all fanin cells by tracing backward from the endpoint register using depth-first search. During the fanin-

[2]Since we always add the edge corresponding to the most timing-critical path to the tree, the resulting cycle is the most critical maximum mean weight (i.e., slack) cycle.

cone tracing, we count only the timing-critical fanin cells since non-critical fanin cells have little effect on the cost of endpoint optimization. The procedure optimizes the top k endpoints according to the sensitivity in each iteration (i.e., SEOpt) (Lines 12-13). $TimingOpt(N,P)$ (Line 13) represents a timing optimization on the set of endpoints P in netlist N. We perform SkewOpt after optimization on the fanin cones of the top k endpoints (Line 14). $ISTA(N,P)$ (Line 15) is an incremental static timing analysis (STA) after the optimization. If the timing slack of endpoint p becomes positive, the procedure replaces the register of p with a conventional register. Then, the cost of the netlist ($COST(N)$) is updated. After the iterations of endpoint optimization, the procedure finds a netlist (N_{min}) which has a heuristically minimized cost in terms of area and/or power consumption.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

We conduct experiments with four sub-modules (Table 1) from the *OpenSPARC T1* processor [26]. The modules are implemented with commercial 28nm FDSOI libraries; synthesis is performed with *Synopsys Design Compiler H-2013.03-SP3* [27], and placement and routing are performed with *Cadence EDI System 13.1* [24]. Runtime is reduced by adopting a restricted library of 90 commonly used cells (40 combinational and five sequential, with dual-V_T flavors). We use three error-tolerant flip-flops in our experiments, with overheads of power, area (estimated based on extra transistor count), and throughput as given in Table 2.

In our experiments, (i) we model power penalty by multiplying the total power of the error-tolerant flip-flops by the corresponding power overhead; (ii) we model area overhead by scaling the size of flip-flops in LEF; and (iii) we model safety margin using the SDC file (using original clock period + safety margin as the clock period for implementation, but specifying the original clock period as the maximum delay on paths with conventional flip-flops as endpoints). To obtain switching activity and accurate error rate (i.e., to determine α in Equation (3)), we perform gate-level simulation using *Cadence NC-Verilog v8.2* [23]. Figure 4 compares the actual error rates and estimated error rates at different supply voltages. Our estimated error rates roughly match the actual values. To find timing slack and power values at specific voltages, we prepare Synopsys Liberty (.lib) files for each voltage from 1.20V to 0.50V in 20mV increments, using *Synopsys SiliconSmart v2013.06-SP1* [29].

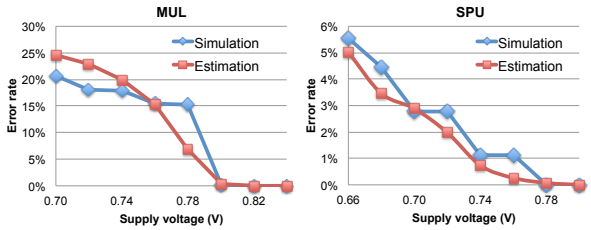

Figure 4: Actual error rates vs. estimated error rates at different voltages.

Table 1: Testcases from OpenSPARC T1.

module	description	# of cells	area (um^2)
FPU	floating point adder	12986	34633
EXU	integer execution	17614	58721
MUL	integer multiplier	13162	40693
SPU	stream processing	8066	28150

Table 2: Penalties of error-tolerant flip-flops.

design	Razor	Razor-Lite	TIMBER
power penalty	30% [9]	~0% [17]	100% [7]
area penalty	182% [17]	33% [17]	255% [6]
# of recovery cycles	5 [19]	11 [17]	0 [7]

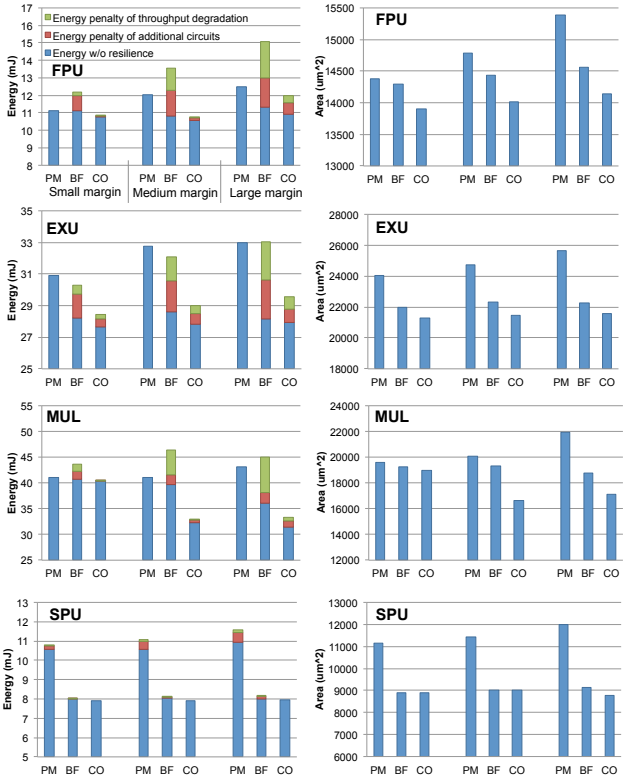

Figure 5: Energy and area results from different implementation methodologies – pure-margin (PM), brute-force (BF) and CombOpt (CO).

4.2 Methodology Comparison

In our first experiment, we compare area and energy results of CombOpt with (i) pure-margin designs[3] and (ii) a brute-force methodology where we first implement designs without considering the safety margin, then replace with error-tolerant registers any points having timing violations with respect to the required margin. We use Razor flip-flops for error resilience in this experiment. We compare our methodology at three different safety margins, where large margin, medium margin and small margins are respectively 15%, 10% and 5% of the clock period (0.8ns).

Figure 5 shows that the benefits of CombOpt increase with the required safety margin. We observe that CombOpt achieves up to 19% (6% on average) energy and 14% (4% on average) area reduction compared to the brute-force method, and up to 21% (16% on average) energy and 17% (14% on average) area reduction compared to the conventional pure-margin method.

4.3 Different Error-Tolerant Flip-Flops

In our second experiment, we study the cost of different error-tolerant flip-flops. We compare designs implemented with Razor, Razor-Lite and TIMBER types of error-tolerant flip-flops. We implement the designs with the brute-force methodology mentioned above, and CombOpt.

Table 3 shows results for the MUL testcase, where we assume a safety margin of 10% of the clock period. Using the brute-force method, we observe 17% and 15% energy penalties due to throughput degradation and additional circuit overheads in designs using Razor-Lite and TIMBER flip-flops, respectively. We also observe that CombOpt significantly reduces the number of error-tolerant flip-flops (by 60%, 56% and 62% for Razor-Lite, Razor and TIMBER, respectively). Such reductions can be enabling to the energy- and area-feasibility of resilient designs.

[3]We define a *pure-margin design* as one wherein only timing margins are inserted to ensure correct operation under dynamic variation.

Table 3: Comparison among different error-tolerant flip-flops.

design	Razor-Lite	Razor	TIMBER
method	brute-force		
total energy (mJ)	43.58	44.95	44.18
energy w/o resilience (mJ)	36.21	36.06	37.49
energy w/ additional circuits (mJ)	0.00	2.08	6.69
energy w/ throughput penalty (mJ)	7.36	6.82	0.00
# of error-tolerant flip-flops	893	927	895
total cell area (μm^2)	19063	18751	19361
method	CombOpt		
total energy (mJ)	31.68	33.25	35.35
energy w/o resilience (mJ)	31.10	31.40	32.26
energy w/ additional circuits (mJ)	0.00	1.15	3.09
energy w/ throughput penalty (mJ)	0.58	0.69	0.00
# of error-tolerant flip-flops	361	409	342
total cell area (μm^2)	17070	17138	17239

4.4 Energy Reduction from AVS

In our last experiment, we study the energy reduction of resilient design in an adaptive voltage scaling context. We compare energy of designs implemented with the brute-force method and our CombOpt at different supply voltages. In addition, we implement pure-margin designs at each voltage as references. Figure 6 shows results for our four testcases. The designs implemented with CombOpt achieve significant energy reduction with voltage scaling. This is because our optimization comprehends the toggle information and tradeoff between power consumption on combinational cells and error-tolerant registers; this results in less energy penalty from throughput degradation and additional circuits. Note that although throughput degradation increases at lower supply voltages, the total power also reduces. This leads to the observed decrease in energy cost of throughput degradation at lower supply voltages for most cases. However, further downscaling of the supply voltage is limited by the safety margin. We also observe that the benefits of resilience can be design-dependent: a design with larger error rate (e.g., FPU) derives less benefit from resilience because of large recovery overheads. From our proposed optimization (CombOpt), we achieve 11% and 18% energy reduction on average compared to the brute-force and conventional (pure-margin) methods, respectively.

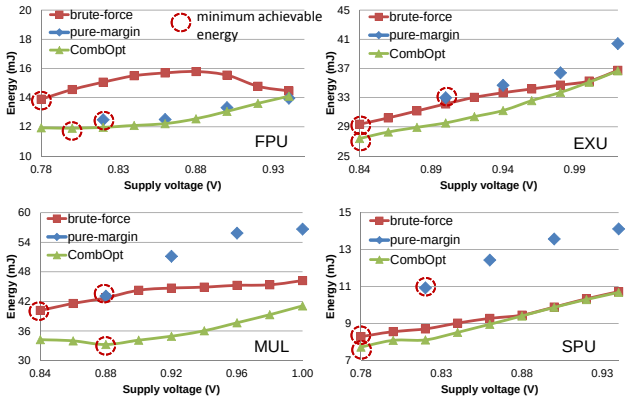

Figure 6: Energy consumption with voltage scaling, and minimum achievable energy for each method.

5. CONCLUSION

By allowing timing errors, resilient design techniques can reduce design effort and obtain power and area benefits over conventional designs which always operate correctly. However, throughput and circuit power and/or area overheads can diminish the benefits of resilient design.

In this work, we provide a new design flow for mixing of resilient and non-resilient circuits within a given implementation, so as to minimize the overhead of error resilience. We propose a *selective-endpoint optimization*, which reduces timing-critical endpoints with small cost of timing optimization. We also propose a *clock skew optimization*, specifically targeted to a resilient design methodology, which improves robustness to process, voltage and temperature variations. Our proposed optimization techniques achieve significant energy reductions – up to 19% and 21% – compared to conventional (pure-margin) design and a brute-force resilience implementation, respectively. In an adaptive voltage scaling context, our method shows further benefits of error resilience.

A number of research directions remain open. In particular, our ongoing work seeks to (1) implement an entire physical design flow for error-resilient systems including error-detection network and hold margin consideration, (2) build a unified framework for simultaneous data- and clock-path optimization, and (3) study the impact of process variation on resilient design methodologies.

6. REFERENCES

[1] C. Albrecht, B. Korte, J. Schietke and J. Vygen, "Cycle Time and Slack Optimization for VLSI-Chips", *Proc. ICCAD*, 1999, pp. 232–237.

[2] C. Albrecht, B. Korte, J. Schietke and J. Vygen, "Maximum Mean Weight Cycle in a Digraph and Minimizing Cycle Time of a Logic Chip", *Discrete Applied Mathematics* 123(1-3) (2002), pp. 103–127.

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

[4] N. D. P. Avirneni, V. Subramanian and A. K. Somani, "Low Overhead Soft Error Mitigation Techniques for High-Performance and Aggressive Systems", *Proc. DSN*, 2009, pp. 185–194.

[5] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De and S. Borkar, "Circuit Techniques for Dynamic Variation Tolerance", *Proc. DAC*, 2009, pp. 4–7.

[6] C.-H. Chen, Y. Tao and Z. Zhang, "Efficient In Situ Error Detection Enabling Diverse Path Coverage", *Proc. ISCAS*, 2013, pp. 773–776.

[7] M. Choudhury, V. Chandra, K. Mohanram and R. Aitken, "TIMBER: Time Borrowing and Error Relaying for Online Timing Error Resilience", *Proc. DATE*, 2010, pp. 1554–1559.

[8] M. R. Choudhury and K. Mohanram, "Masking Timing Errors on Speed-Paths in Logic Circuits", *Proc. DATE*, 2009, pp. 87–92.

[9] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull and D. T. Blaauw, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance", *Proc. ISSCC*, 2008, pp. 400–622.

[10] 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. MICRO*, 2003, pp. 7–18.

[11] J. P. Fishburn, "Clock Skew Optimization", *IEEE Trans. on Computers* 39(7) (1990), pp. 945–951.

[12] 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*, 26(11) (2007), pp. 1947–1956.

[13] B. Greskamp and J. Torrellas, "Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking", *Proc. PACT*, 2007, pp. 213–224.

[14] 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. HPCA*, 2009, pp. 213–224.

[15] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Recovery-driven Design: A Methodology for Power Minimization for Error Tolerant Processor Modules", *Proc. DAC*, 2010, pp. 825–830.

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

[17] S. Kim, I. Kwon, D. Fick, M. Kim, Y.-P. Chen and D. Sylvester, "Razor-Lite: A Side-Channel Error-Detection Register for Timing-Margin Recovery in 45nm SOI CMOS", *Proc. ISSCC*, 2013, pp. 264–265.

[18] V. Subramanian and A. Somani, "Conjoined Pipeline: Enhancing Hardware Reliability and Performance through Organized Pipeline Redundancy", *Proc. PRDC*, 2008, pp. 9–16.

[19] L. Wan and D. Chen, "DynaTune: Circuit-Level Optimization for Timing Speculation Considering Dynamic Path Behavior", *Proc. ICCAD*, 2009, pp. 172–179.

[20] Y.-M. Yang, I. H.-R. Jiang and S.-T. Ho, "PushPull: Short Path Padding for Timing Error Resilience Circuits", *Proc. ISPD*, 2013, pp. 50–57.

[21] N. E. Young, R. E. Tarjan and J. B. Orlin, "Faster Parametric Shortest Path and Minimum Balance Algorithms", *Networks* 21:2 (1991), pp. 205–221.

[22] F. Yuan and Q. Xu, "InTimeFix: A Low-Cost and Scalable Technique for In-Situ Timing Error Masking in Logic Circuits", *Proc. DAC*, 2013, pp. 183:1–183:6.

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

[24] *Cadence Encounter Digital Implementation System User's Manual.* http://www.cadence.com/

[25] *ILOG CPLEX.* http://www.ilog.com/products/cplex/

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

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

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

[29] *Synopsys SiliconSmart User's Manual.* http://www.synopsys.com/