An Improved Methodology for Resilient Design Implementation

ANDREW B. KAHNG, University of California, San Diego
SEOKHYEONG KANG, Ulsan National Institute of Science and Technology
JIAJIA LI, University of California, San Diego
JOSE PINEDA DE GYVEZ, NXP Semiconductors

Resilient design techniques are used to (i) ensure correct operation under dynamic variations and to (ii) improve design performance (e.g., timing speculation). However, significant overheads (e.g., 16% and 14% 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 describe an improved 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 trade-off between cost of resilience and margin on combinational paths. Since the error-detection network can result in up to 9% additional wirelength cost, we also propose a matching-based algorithm for construction of the error-detection network to minimize this resilience overhead. Further, our implementations comprehend the impacts of signoff corners (in particular, hold constraints, and use of typical vs. slow libraries) and process variation, which are typically omitted in previous studies of resilience trade-offs. Our proposed flow achieves energy reductions of up to 21% and 10% compared to a conventional (with only margin used to attain robustness) design and a brute-force implementation (i.e., a typical resilient design, where resilient endpoints are (greedily) instantiated at timing-critical endpoints), respectively. We show that these benefits increase in the context of an adaptive voltage scaling strategy.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*

General Terms: Design

Additional Key Words and Phrases: Resilience, energy reduction, design optimization

ACM Reference Format:
Andrew B. Kahng, Seokhyeong Kang, Jiajia Li, and Jose Pineda de Gyvez. 2015. An Improved methodology for resilient design implementation. ACM Trans. Des. Autom. Electron. Syst. 20, 4, Article 66 (September 2015), 26 pages.
DOI: http://dx.doi.org/10.1145/2749462

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

This paper is an extended and revised version of Kahng et al. [2014].
Authors' addresses: A. B. Kahng, Department of Computer Science and Engineering, and Electrical and Computer Engineering, University of California at San Diego; S. Kang, School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology, Ulsan, South Korea; J. Li (corresponding author), Department of Electrical and Computer Engineering, University of California at San Diego; J. Pineda de Gyvez, NXP Semiconductors, Eindhoven, The Netherlands; corresponding author's email: jil150@ucsd.edu.

DOI: http://dx.doi.org/10.1145/2749462

other mechanisms. To ensure correct functionality and robustness, traditional IC implementation methodologies build guardband into clock frequencies and design signoffs—notably, timing signoff at slow 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* [Austin et al. 2005], 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. [2009b], where several techniques for dynamic variation tolerance (i.e., resilient designs) are presented.

Resilient designs, as discussed in this work, use variant register (i.e., timing endpoint) circuit designs to trade off design robustness against design quality (performance, power, and area); ideally, they can ensure correctness against variation and improve signoff performance [Choudhury et al. 2010; Das et al. 2009; Ernst et al. 2003; Ghosh and Roy 2007; Greskamp and Torrellas 2007; Subramanian and Somani 2008]. *Razor* [Ernst et al. 2003] is a well-known technique for detecting and correcting timing errors. 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) or via instruction rollback/replay. In the following discussion, we define the maximum timing violation that a resilient design can tolerate as the *safety margin* of the corresponding design.

By allowing timing errors, resilient designs are used to improve performance. An example is *timing speculation* [Wan and Chen 2009], which increases the clock frequency and exploits error detection and recovery mechanisms to correct resulting errors. Timing improvement from resilient designs can lead to further power and area benefits over conventional designs. In other words, due to relaxed timing constraints, we can reduce the power and area of logic cells in the fanin cone of an endpoint at which an error-tolerant register has been instantiated.

A practical methodology for deployment of resilient designs must overcome a number of significant overheads of resilience. Notably, resilient designs require additional circuits or cycles to detect and correct timing errors. Figure 1 shows the structure of Razor, Razor-Lite [Kim et al. 2013], and TIMBER [Choudhury et al. 2010] 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 a Razor flip-flop is 30% [Das et al. 2009]. Although the power overhead has been significantly reduced for each error-tolerant register in a recent work [Kim et al. 2013], the cost incurred by the error-detection network is still large.

We use the term *pure-resilient design* to indicate a design that uses only error-tolerant registers (instead of conventional ones). Our background studies indicate that in a pure-resilient design, the error-detection network alone can consume up to 9% of the total wirelength. Furthermore, the additional cycles needed to recover from errors can lead to performance (throughput) 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.

A crucial open question is, "What is the minimum achievable cost of (a given amount of) resilience?" As a step toward answering this question, in this work we perform in-depth studies of the trade-off between the overhead of error-tolerant circuits and the cost of the traditional timing optimizations, with the goal of assessing the 'true' benefits

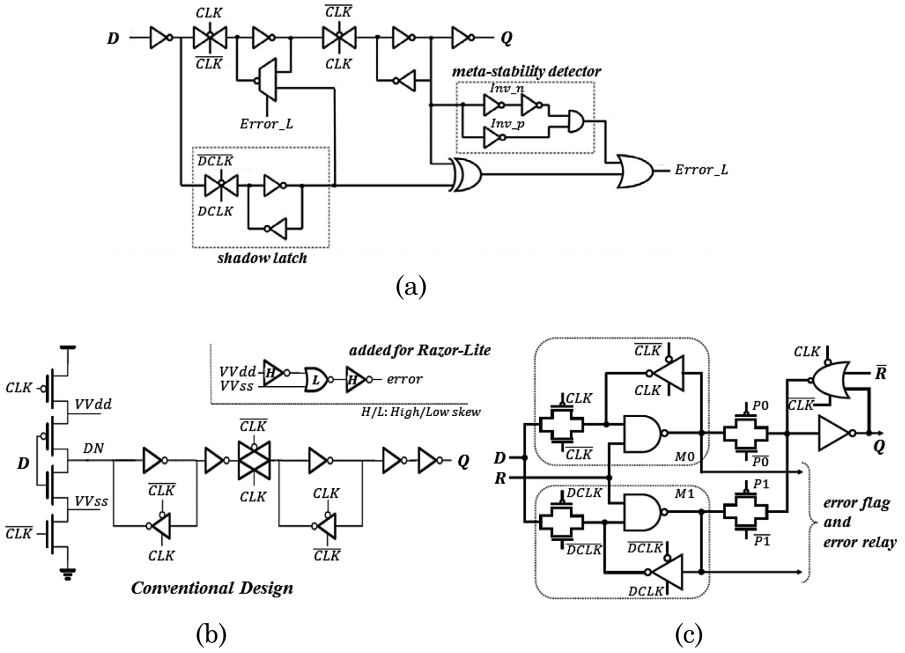

Fig. 1. Structure of (a) Razor, (b) Razor-Lite, and (c) TIMBER flip-flops.

of resilient design techniques. To our knowledge, no previous work has conducted such studies to explore the 'true' benefits of resilience comprehending all types of resilience cost and various types of resilient designs. We propose two effective design optimization techniques—*selective-endpoint optimization*, and *clock skew (useful skew) optimization*—to minimize the costs of resilience, that is, (i) power and area overheads of resilient circuits, and (ii) throughput degradation due to additional cycles for error recovery. Based on the optimization, we develop a complete implementation flow (i.e., from placement to signoff) for resilient designs. Our flow comprehends the impacts of signoff corners and process variation, ensuring timing correctness of conventional flip-flops at the slow corner while optimizing design energy at the typical corner. Since our selective-endpoint optimization reduces the number of error-tolerant flip-flops and the applied clock skew optimization is hold-timing aware, we do not specifically optimize on short-path padding, and small hold penalty is observed. However, our optimization flow can easily be combined with existing short-path padding optimizations (e.g., [Yang et al. 2013]). Our contributions include the following.

—We propose an optimization methodology to reduce the cost of resilience. Our methodology 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 and fanout cones, and (iii) throughput degradation due to timing errors.
—We propose an implementation flow to construct the error-detection network, which is required in an actual implementation. We exploit geometric placement information of the error-tolerant registers to substantially mitigate wirelength overhead of the error-detection network.

—We perform typical-corner optimization of resilient designs to maximize energy reduction. At the same time, our implementations comprehend both multiple signoff corners and process variation.
—By reducing the number of error-tolerant registers, our optimization minimizes the cost of short-path padding.
—We assess the opportunities and costs of resilient implementations across different error-tolerant register designs as well as in the adaptive voltage scaling (AVS) context.

The rest of this article is organized as follows. Section 2 presents related works. Section 3 formulates the problem of minimizing the cost of resilience and describes our methodology for implementing low-cost resilience. Section 4 presents our experimental results and analyses, and Section 5 concludes the article.

2. RELATED WORK

A number of resilient design techniques have been proposed that allow timing errors in conjunction 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, instruction rollback or replay 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. *DIVA* [Austin 1999] applies a functional checker to verify the correctness of the core processor's computation, only permitting correct results to commit. *Paceline* [Greskamp and Torrellas 2007] employs a *leader-checker* which checks timing errors due to overclocking. *CPipe* [Subramanian and Somani 2008] 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 and Mohanram [2009] synthesize error-masking circuits and use 2-to-1 multiplexers to mask errors at the output of critical paths. Similarly, Yuan and Xu [2013] mask errors by adding redundant approximation logic which has higher speed than the original circuit. *TIMBER* flip-flops and latches [Choudhury et al. 2010] enable online timing error masking via time-borrowing from the succeeding pipeline stage. 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 [Avirneni and Somani 2012; Das et al. 2009; Ernst et al. 2003; Kim et al. 2013] replace registers with specialized flip-flops which detect timing errors by capturing the correct value at shadow latches with a delayed clock. *Razor* [Ernst et al. 2003] can correct timing errors within a specific safety margin of the error-tolerant register. *Razor II* [Das et al. 2009] 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. *Bubble Razor* [Fojtik et al. 2013], which uses two-phase latch timing, stalls the pipeline based on the propagation of error clock gating control signals ("bubbles") to mitigate timing errors. A Markov chain-based analysis for a ring of Bubble Razors is provided in Zhang and Beerel [2014]. A more recent work *Razor-Lite* [Kim et al. 2013] further reduces the area and power penalties of error-tolerant registers. *STEM* [Avirneni and Somani 2012] improves the capability of error-detection with a second shadow latch. Bowman et al. [2009a] introduce two error-tolerant circuits, transition detector with time-borrowing and double sampling with time-borrowing.

Resilient Design Optimization. With the preceding error-tolerant registers, various design-level optimization techniques [Choudhury and Mohanram 2009; Greskamp et al. 2009; Kahng et al. 2010a, 2010b; Liu et al. 2011, 2012, Wan and Chen 2009; Yuan and Xu 2013] have been proposed which identify and optimize critical paths that are frequently exercised during operation. These works apply resilient techniques to timing-critical and/or frequently-exercised paths but typically fail to holistically consider the 'true' benefits and costs of the error-tolerant circuits during the optimization. In other words, the trade-off between the cost of resilience and the costs of margin insertion for data paths (which will be illustrated later) is ignored. Further, none of these works consider all types of costs in a resilient design (i.e., power and area of resilient circuits and data paths and throughput degradation) simultaneously. For instance, Choudhury and Mohanram [2009] and Yuan and Xu [2013] optimize area and power of resilient circuits but not the costs of data paths; the optimizations in other works [Kahng et al. 2010a; Liu et al. 2012; Wan and Chen 2009] consider power of data paths and throughput degradation but not the overhead of resilient registers; Liu et al. [2011] minimizes the number of error-tolerant registers and the cost of short path padding but without regard to the overhead of throughput degradation. In addition, some optimizations [Choudhury and Mohanram 2009; Liu et al. 2011, 2012, Wan and Chen 2009; Yuan and Xu 2013] occur at the synthesis stage. However, the timing-critical paths can vary after placement and routing (P&R), and this discrepancy can degrade the solution quality. Our present work is differentiated by performing optimization during P&R stage that comprehends the trade-off between the cost of resilience and the cost of margin insertion, as well as by simultaneously considering (more comprehensively) the costs in a resilient design.

Clock Skew Optimization. Of particular note are clock skew optimizations that have been proposed by previous works on enhancement of design robustness and timing speculation. An early work [Fishburn 1990] formulates a linear program to maximize the minimum timing slack in a design via clock skew scheduling, which improves the tolerance of the design to variations. Wu and Marculescu [2010] adjust clock latencies to minimize the probability of timing errors being latched by overlapping separate error-latching windows. Their optimization also prevents errors from propagating along pipeline stages. Chen et al. [2014] and Ye et al. [2011] propose online clock skew tuning methods to minimize timing errors during runtime using tunable delay buffers and clock tuning elements (i.e., circuits with multiple skew configurations), respectively. However, due to implementation complexity, fine-grained optimization is practically impossible. Further, clock tuning logic can introduce extra area and power costs. Ye et al. [2012] propose clock skew scheduling optimization to minimize the error rate in a resilient design. Based on error probability at each endpoint, they determine skew values using a gradient-descent method. The work uses Razor flip-flops for timing-critical endpoints and ignores the trade-off between cost of resilience and data path optimization. The optimization also ignores hold constraints, which are critical in a design with resilient registers, as well as the potential power implications (e.g., for data paths) of the skew scheduling. Our present work proposes clock skew optimization that maximizes both setup and hold slacks at all timing-violated paths with comprehension of toggle rate information. Further, in our work, the improved timing slacks are exploited to enable removal of error-tolerant registers and power reduction on data paths.

3. LOW-COST RESILIENT DESIGN IMPLEMENTATION

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

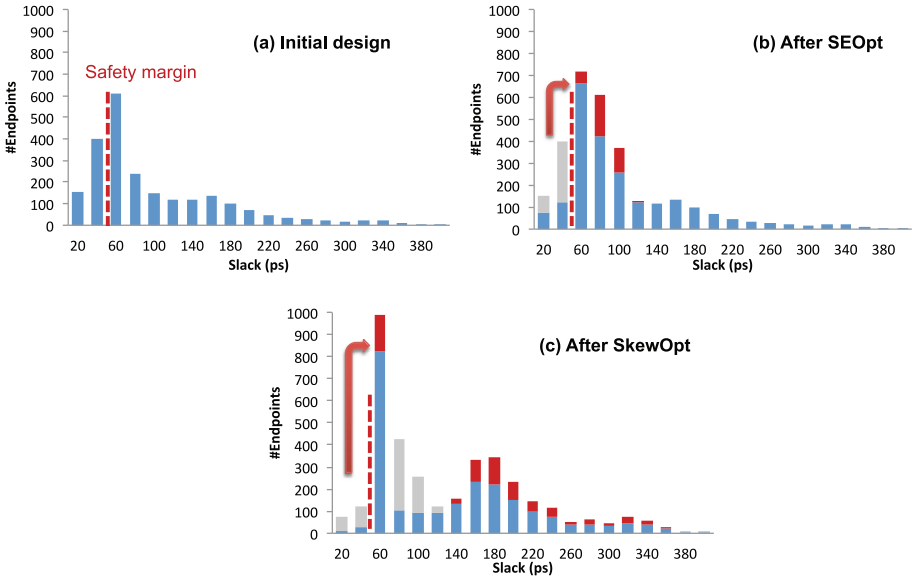

Fig. 2. Slack distribution of endpoints in (a) original design; (b) design with only selective-endpoint optimization; and (c) design with combined selective-endpoint and useful skew optimization. Red dotted lines indicate required safety margin. Design: FPU (OpenSPARC T1). Technology: 28nm FDSOI.

resilient design Figure 2(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 Figure 2(b), we tightly optimize a set of selected endpoints to reduce the resilience overheads. During clock skew optimization Figure 2(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. We will show in Section 4 that our proposed optimization achieves significant improvement in terms of area and energy as compared to previous works, that is, (i) conventional resilient design implementation and/or (ii) useful skew optimization on resilient designs.

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 instruction rollback or replay.

We calculate design energy based on total power and throughput information, that is,

$$\text{Energy} = \frac{\text{Power}}{\text{Throughput}}. \tag{1}$$

We further estimate the throughput based on error rate information as

$$\text{Throughput} = \frac{1 - \text{ErrorRate}}{T} + \frac{\text{ErrorRate}}{\theta \times T}, \tag{2}$$

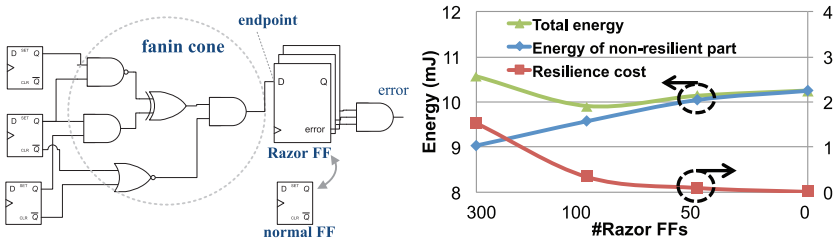

Fig. 3. (a) Illustration of the trade-off between cost of resilience and of data path optimization. (b) With reduced number of Razor flip-flops, resilience cost decreases but power of data paths increases. Design: FPU (OpenSPARC T1). Technology: 28nm FDSOI.

where T is the clock period, and θ is the number of cycles needed to recover from an error. 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) as

$$\text{ErrorRate} = \alpha \times \frac{\sum (h_{ff} \times \frac{\sum h_{p\_neg}}{\sum h_{p\_all}})}{\sum h_{ff}}, \tag{3}$$

where h_{ff} is the toggle rate of a flip-flop, h_{p_neg} and h_{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 multiple 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 now describe *selective-endpoint optimization* (SEOpt) for reduction of resilience cost (primarily area, power, and throughput degradation). SEOpt trades off between the costs of resilience and of data path optimization. We note that ours is the first work to consider such a trade-off in resilient design optimization. As illustrated in Figure 3(a), increasing the timing margin at an endpoint allows for replacement of the error-tolerant register with a conventional one and for removal of replica circuits. However, these margins incur area and power costs in combinational logic cones. Figure 3(b) shows the example of the OpenSPARC T1 FPU: as we reduce the number of Razor flip-flops from 300 to zero, resilience cost decreases while power of the non-resilient part increases. This results in an observed unimodal behavior of the design energy change. In this work, we seek the subset of endpoints for margin insertion that, when optimized (i.e., replaced with conventional registers) in this way, leads to minimum design energy. Two 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 versus margin relationship. Therefore, to reduce the optimization cost, we should preferentially optimize endpoints which are less sensitive to slack margin insertion. In SEOpt, we evaluate sensitivity functions for endpoints to estimate the potential optimization cost and guide the selection of endpoints for optimization. That is, a given sensitivity function of an endpoint reflects the available performance versus power and/or area trade-off of the corresponding fanin cone. We observe that the optimization cost increases significantly for an endpoint which (i) is timing-critical, (ii) has a large number of timing-critical fanin cells (i.e., negative-slack

cells in the fanin cone of the endpoint), and (iii) has a fanin cone with large power (e.g., due to high toggle rate). We therefore use slack at endpoint $slack(p)$, number of timing-critical fanin cells $num_{cri}(p)$, and power of timing-critical fanin cells $power(c)$ to evaluate the sensitivity of each endpoint. We study sensitivity functions for a given timing endpoint p with different combinations of these parameters, and the following five empirically show good results:

$$SF1(p) = |slack(p)|, \tag{4}$$

where we consider timing critically at the endpoint;

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

where we consider slack at the endpoint and the number of negative-slack cells in the fanin cone;

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

where we consider slack at the endpoint and portion of negative-slack cells over all cells in the fanin cone;

$$SF4(p) = |slack(p)| \times \sum_{c \in fanin(p)} power(c), \tag{7}$$

where we consider slack at the endpoint and power of timing-critical fanin cells; and

$$SF5(p) = \sum_{c \in fanin(p)} (|slack(c)| \times power(c)), \tag{8}$$

where we consider the products of slack and power of timing-critical fanin cells.

To study the performance of each sensitivity function, we sort the endpoints in increasing order of their estimated sensitivities, based on the given sensitivity function. We then optimize the top $\eta\%$ endpoints of the sorted list, where we increase η from 0 to 100 with a step size of 5. Figure 4 shows power and area resulting from selective-endpoint optimizations based on the five sensitivity functions on FPU in a foundry 28nm FDSOI technology. In this example, the safety margin is 10% of the clock period.[1] In all experiments, we assume a switching activity of 0.2 at primary inputs and propagate activities using a commercial P&R tool. We then dump out a *switching activity interchange format* (SAIF) file and use it for power analysis and error rate estimation. Note that our optimization framework can be extended to vector-based scenarios, where we generate SAIF file from the *value change dump* (VCD) file derived from gate-level simulation. We observe in Figure 4 that for a given number of endpoints to optimize, SEOpt based on SF2 and SF5 incurs smaller power and area penalties. We use SF5 in the experiments reported in Section 4.[2]

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).

[1]In Choudhury et al. [2010], safety margins of 10%, 20%, and 30% of clock period are studied.

[2]Although the best sensitivity function might vary in a different technology/library or with a different implementation tool, one can apply the same evaluation method given any specific design enablement, that is, to pick the most successful sensitivity function from among various options based on the selected parameters.

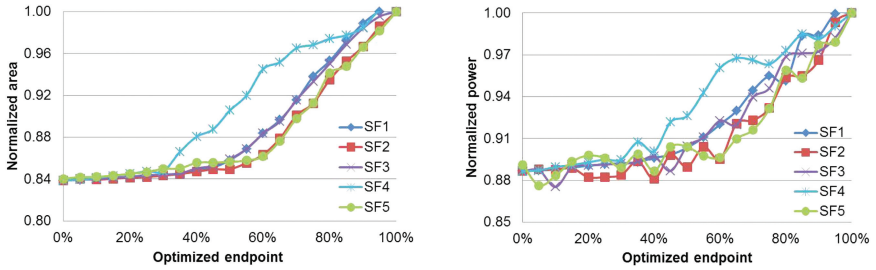

Fig. 4. Cell area and total power resulting from selective-endpoint optimization with different sensitivity functions. Design: FPU (OpenSPARC T1). Technology: 28nm FDSOI.

3.3. Clock Skew Optimization

To further reduce the number of error-tolerant registers and minimize timing errors, we apply *clock skew optimization* (SkewOpt), which maximizes slacks at timing-violated endpoints. In SkewOpt, we formulate the clock skew optimization problem as a *maximum mean weight cycle* problem [Albrecht et al. 2002]. This is because the maximum achievable timing slack of a given path is determined by the maximum average slack of a cycle (i.e., a loop formed by timing paths) which contains that path. We use the *parametric shortest path* algorithm [Young et al. 1991] 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 setup and one for hold) (Line 1). The weights of edges indicate setup/hold slacks of timing paths in the corresponding FF-to-FF logic cones.

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

$$l_q + \underbrace{(T - d_q - d_{p,q}^{max} - t_q^{setup} - t_{p,q}^{margin})}_{s_{p,q}} - \lambda \geq l_p \ (q \in R), \tag{9}$$

$$l_q + \underbrace{(T - d_q - d_{p,q}^{max} - t_q^{setup} - t_{p,q}^{margin})}_{s_{p,q}} \geq l_p \ (q \notin R), \tag{10}$$

$$l_p + \underbrace{(d_p - d_{p,q}^{min} - t_q^{hold} - d_q)}_{s_{p,q}} \geq l_q \ \ (\forall q), \tag{11}$$

where T is the clock period; l_p is the clock arrival time of endpoint p; d_p is the clock-to-Q delay of p; $d_{p,q}^{max}$ and $d_{p,q}^{min}$ are, respectively, the maximum and minimum path delay from p to q; t_q^{setup} and t_q^{hold} are the setup and hold times of q; and $t_{p,q}^{margin}$ 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

ALGORITHM 1: Clock Skew Optimization (SkewOpt)

Procedure $SkewOpt(N)$
1. $G(V,\ E) \leftarrow$ construct graph corresponding to N // N is the input netlist
2. Initialize solution graph $G'(V,\ \emptyset)$
3. $V \leftarrow \{r\} \cup V$; $E \leftarrow \{(r,\ p)\} \cup E,\ \forall p \neq r$; $w(r,\ p) \leftarrow 0,\ \forall p \neq r$
4. $E_T \leftarrow \{(r,\ p)\},\ \forall p \neq r$
5. Update $p\_w(p),\ \forall p \in V$ // $p\_w(p) = \sum w(p_i,\ p_j),\ \forall (p_i,\ p_j) \in$ shortest path (SP) from r to p
6. **while** $|E| > 1$ **do**
7. $\quad \lambda_{min} \leftarrow +\infty$
8. $\quad$ **for all** $(p, q) \in E$ for which $(p, q) \notin E_T$ **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. $\quad\quad\quad e_{min} \leftarrow (p, q)$
13. $\quad\quad$ **end if**
14. $\quad$ **end for**
15. $\quad E_T \leftarrow E_T \cup \{(p, q)\}$
16. $\quad \lambda \leftarrow \lambda_{min}$
17. $\quad$ Remove all edges from E_T that have the same head vertex as e_{min}
18. $\quad$ **if** there is a cycle in E_T **then**
19. $\quad\quad slack(p, q) \leftarrow \lambda_{min},\ \forall (p, q) \in cycle$
20. $\quad\quad$ Add all edges on cycle to G'
21. $\quad\quad E \leftarrow E \setminus \{(p, q) \mid (p, q) \in cycle\}$
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 l_q based on $slack(p, q)$ and l_p
27. $N_{sol} \leftarrow$ apply $l_p,\ \forall p$ to N
28. **return** N_{sol}

constraints on a given nonparameterized edge in the constructed graph with an edge weight of $s_{p,q}$.

In the graph $G(V,\ E)$, we always maintain a tree $(V,\ E_T)$ for storing 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). In Line 5, $p\_w(p)$ is the total weight along the shortest path (i.e., path with the minimum total weight) from r to p in G. Then, we iteratively add edges corresponding to the most timing-critical paths to the tree (Lines 7–15) while removing any dummy edges that have the same head vertex as an added edge (Line 17). When adding an edge to the tree results in a cycle[3], 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. To each parameterized edge, a weight is assigned equal to the summation of weights (i.e., slacks) on the cycle divided by the number of parameterized edges on the cycle. We assign zero slack to the nonparameterized 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 that contains critical paths with error-tolerant registers as endpoints. Note that assigning new weights indicates a change of clock arrival times. Therefore, we update the weights of edges incident to vertices on the cycle. We then optimize slacks on the

[3]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.

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 found). Last, we traverse the solution graph and calculate the clock arrival times based on the optimized path slacks (Line 26).

To further enable error-rate awareness and reduce the cost of throughput degradation, we extract toggle-rate information of each timing path and replace Constraint (9) by

$$l_q + \frac{s_{p,q}}{1 + \beta \times h(p,q)} - \lambda \geq l_p \ (q \in R), \tag{12}$$

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

3.4. Proposed Optimization Flow

As mentioned in Section 3.2, SEOpt reduces the cost of resilience via optimizations on data paths. However, such optimization incurs power and area overheads. By contrast, SkewOpt migrates timing slacks from timing non-critical paths to timing-critical paths, which does not incur power and area penalty. But, SkewOpt cannot generate additional slacks, hence its performance highly depends on the topology of the sequential graph. For example, SkewOpt might not perform well when there are many cycles consisting of timing-critical paths. In this work, 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. We then apply SkewOpt for an improved distribution of timing slacks. In this way, we reduce the number of error-tolerant registers and minimize error rates of a resilient design without incurring large power and area penalties.

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 at endpoints with timing violations. The procedure runs static timing analysis (*STA*) and computes a sensitivity value for each endpoint p (Lines 1–8). The procedure finds all fanin cells by tracing backward from the endpoint register using depth-first search. During the fanin-cone tracing, we count only the timing-critical fanin cells, since noncritical 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 (Lines 12–13). *TimingOpt*(N_{i-1}, P_i) (Line 13) represents a timing optimization on the set of endpoints P_i in netlist N_{i-1}. We perform SkewOpt after optimization on the fanin cones of the top k endpoints (Line 14). *ISTA*(N_i, P_i) in Line 15 indicates incremental static timing analysis (*STA*) after optimization. If the timing slack of endpoint p becomes positive, the procedure replaces the error-tolerant register of p with a conventional one (Line 18). Then, the cost of the netlist (*COST*(N_i)) 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.

3.5. Construction of Error-Detection Network

To detect timing errors, resilient designs typically connect all error-detection signals of error-tolerant registers via the error-detection network (e.g., an OR tree). There are two basic types of error-detection networks. In *centralized pipeline recovery*, one error-detection network connects to all error-tolerant registers. In *distributed pipeline recovery*, a separate error-detection network can be applied to each pipeline stage [Das et al. 2006]. For a design with large number of pipeline stages, centralized pipeline recovery can incur large cost in terms of wirelength, area, and power, compared to the distributed strategy. To be conservative about the resilience cost, in this work, we

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. **if** $slack(p) <$ safety margin **then**
5. Compute sensitivity value for endpoint p
6. $P \leftarrow P \cup \{p\}$
7. **end if**
8. **end for**
9. $m \leftarrow |P|/k$ // m indicates the number of iterations
10. $C_{min} \leftarrow \infty$
11. **for** $i = 0 ; i < m ; i \leftarrow i + 1$ **do**
12. Pick the top k endpoints P_i with minimum *sensitivity* in P
13. $N_i \leftarrow TimingOpt(N_{i-1}, P_i)$
14. $N_i \leftarrow SkewOpt(N_{i-1})$
15. Run $ISTA(N_i, P_i)$
16. **for all** endpoint p in P **do**
17. **if** $slack(p) \geq 0$ **then**
18. Replace error-tolerant register by a conventional one at endpoint p
19. **end if**
20. **end for**
21. $C_i \leftarrow COST(N_i)$
22. **if** $C_i < C_{min}$ **then**
23. $C_{min} \leftarrow C_i$
24. $N_{min} \leftarrow N_i$
25. **end if**
26. $P \leftarrow P - P_i$
27. Update sensitivity values for all endpoints in P
28. **end for**
29. **return** N_{min}

assume a centralized pipeline recovery scheme. We also note that in a resilient design with distributed pipeline recovery, our OR tree insertion algorithm can be applied to each pipeline stage, since all registers in the same pipeline stage are connected together with an error-detection network.

For a design in which the usage of error-tolerant registers is defined before synthesis, the construction of the error-detection network can be accomplished by commercial SP&R tools. However, in our optimization flow, the usage of error-tolerant registers is defined during the placement stage, where an algorithm is required to guide the construction of error-detection network. Further, the size of the error-detection network increases with the number of error-tolerant registers in a resilient design, and this can increase cost. Our initial studies show that in a pure-resilient design, when we construct the error-detection network based on a random clustering method, the wirelengths of the error-detection nets can contribute up to 9% of the design's total wirelength. To minimize the overhead, we develop a matching-based clustering algorithm and use a commercial router to construct the OR tree. Figure 5 depicts our implementation flow (red dotted box). Based on the locations of error-tolerant flip-flops extracted from an initial placement, we construct the OR tree bottom up. We heuristically cluster error-tolerant flip-flops and/or OR gates by iteratively applying (i) the Hungarian method[4] to achieve an assignment (in which cycles are considered to be

[4]`http://www.informatik.uni-freiburg.de/stachnis/index.html`.

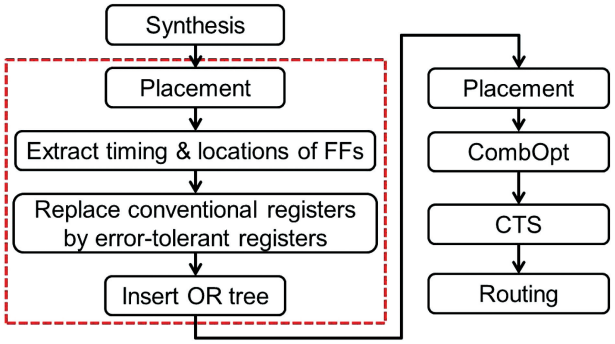

Fig. 5. Implementation flow. OR tree insertion flow is indicated by the red dotted box.

ALGORITHM 3: OR Tree Insertion

Procedure *ORTreeInsertion*(N)

1. $Q \leftarrow$ registers with timing violations w.r.t. required margin in N
2. $N_{sol} \leftarrow N$
3. **while** $|Q| > 1$ **do**
4. Compute distance matrix D where $D_{i,j} = dist(c_i,\ c_j)$ $(c_{[i,j]} \in Q)$
5. $Sol \leftarrow$ apply Hungarian method on D
6. **for all** cycle in Sol **do**
7. $Q_{local} \leftarrow$ cells on the cycle
8. **while** $|Q_{local}| > 1$ **do**
9. $(c_1, c_2) \leftarrow$ find the nearest pair of cells in Q_{local}
10. $c_{new}.x = \frac{c_1.x + c_2.x}{2}$
11. $c_{new}.y = \frac{c_1.y + c_2.y}{2}$
12. Insert OR cell c_{new} to N_{sol}
13. Connect outputs of c_1 and c_2 to inputs of c_{new}
14. $Q_{local} \leftarrow Q_{local} \cup \{c_{new}\} \setminus \{c_1, c_2\}$
15. **end while**
16. $Q \leftarrow Q \cup \{c_{new}\}$
17. **end for**
18. **end while**
19. **return** N_{sol}

clusters of flip-flops and/or OR gates) and (ii) a nearest-neighbor method to build an OR tree within a given cluster. Our OR tree insertion (i.e., clustering of error-tolerant flip-flops and insertion of OR gates) algorithm is described in Algorithm 3.

In the construction of the error-detection network, we start with a synthesized netlist which has only conventional flip-flops. Based on the timing information extracted from the initial placement, all flip-flops having negative slacks with respect to safety margin are set to be error-tolerant flip-flops (Line 1). We then calculate the Manhattan distances between each pair of error-tolerant flip-flops and construct a distance matrix accordingly. In the distance matrix, each row and each column corresponds to an error-tolerant flip-flop such that the matrix entry $D_{i,j}$ is the distance $dist(c_i,\ c_j)$ between the i^{th} and j^{th} flip-flops (Line 4). To avoid a trivial assignment, we define $dist(c_i,\ c_i) = +\infty$ for all i. Since both rows and columns correspond to the same set of error-tolerant flip-flops, the distance matrix D is a symmetric square matrix. We apply the Hungarian algorithm on the distance matrix to obtain a matching solution with minimum total distance (Line 5). The solution matrix (Sol) is a permutation matrix, that is, there

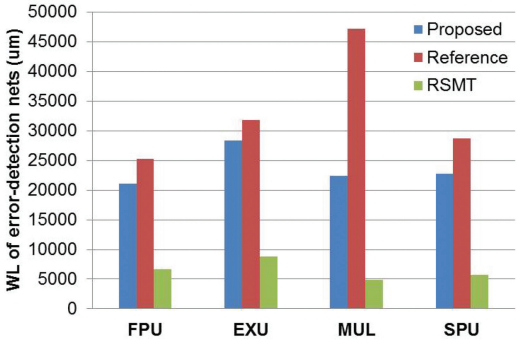

Fig. 6. Our proposed OR tree insertion flow achieves an average of 29% wirelength reduction for the error-detection network, compared to a reference flow. RSMT cost is a (loose) lower bound.

is exactly one '1' in each row and each column; this permutation can be decomposed into cycles that we consider *clusters*.[5] Within each cycle (i.e., cluster), we use a nearest neighbor-based heuristic to construct an OR tree (Lines 8–14). Then, in the next iteration, we form a new distance matrix where each row and column correspond to one cluster from the previous iteration. The location of each cluster is defined by its center coordinates (i.e., the x and y coordinates of the cluster are, respectively, the averages of the x and y coordinates of all cells in the cluster). We continue the construction of the error-detection network until all error-tolerant flip-flops are connected.

Figure 6 compares the wirelength of error-detection nets between our proposed OR tree insertion flow and a reference flow which performs ECO cell and net insertions to construct the error-detection network. The reference flow also uses the nearest-neighbor clustering method for each level of the OR tree. Both the proposed and the reference optimization methods construct an OR tree with only 2-input OR gates. A lower bound for the wirelength is given by the rectilinear Steiner minimum tree (RSMT)[6] over all error-tolerant flip-flops. However, this lower bound is far from achievable due to congestion induced by other nets and power/ground distribution. We compare wirelength values for four pure-resilient designs. The figure shows that our proposed flow achieves an average of 29% wirelength reduction compared to the reference flow.

Note that when SEOpt replaces an error-tolerant flip-flop with a conventional flip-flop, we need to modify the OR tree, as shown in Figure 7. The figure shows two cases of the flip-flop replacement. When the flip-flop (u2) is replaced with a conventional one, we remove the connected OR gate (u1) and modify the OR tree. The steps of the modification are as follows.

(1) Detach nets (n1, n2, and n3) from the OR gate output (u1/Z) and flip-flop error detection pins (u2/ED and u3/ED).
(2) Delete nets (n2 and n3) which are connected to the OR gate.
(3) Delete the OR gate instance (u1).
(4) Attach net n1 to another flip-flop (u3/ED) or OR gate (u3/Z).
(5) Replace the error-tolerant flip-flop (u2) with a conventional flip-flop.
(6) Refine placement and update timing.

[5]For example, if the solution *Sol* contains the matching edges (c_{i_0}, c_{i_1}), (c_{i_1}, c_{i_2}), $\ldots$, $(c_{i_{n-1}}, c_{i_n})$ and (c_{i_n}, c_{i_0}), this is a *cycle* in the permutation defined by *Sol*. We heuristically consider each cycle as a cluster of the i_0^{th}, i_1^{th}, $\ldots$, i_n^{th} cells.
[6]`http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/`.

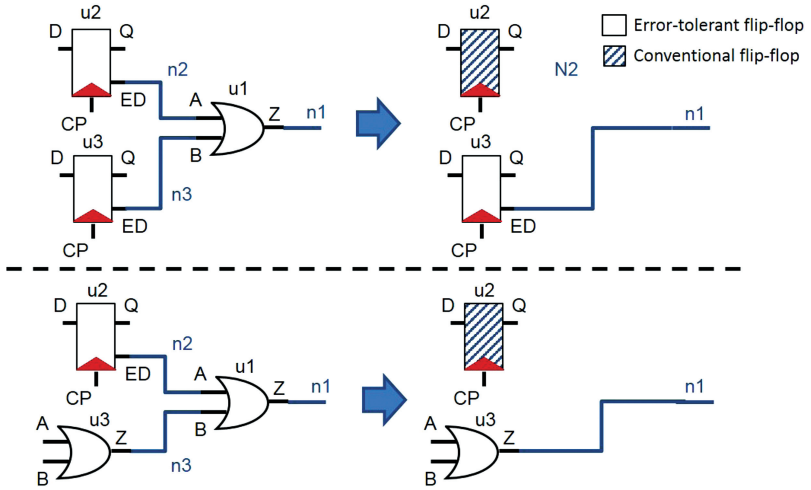

Fig. 7. Replacement of an error-tolerant flip-flop with a conventional flip-flop for u2. Note that for readability, nets connected to D, Q, and CP pins of flip-flops are not shown.

3.6. Typical-Corner Optimization

To accurately assess the benefits and overheads of resilience approaches, it is important to consider impacts of signoff corners and process variation during implementation. For paths with conventional registers as endpoints, we ensure that there is no timing violation at the slow corner. However, since error-tolerant registers can detect and correct timing errors, we can allow some amount of negative timing slack[7] for paths with error-tolerant registers as endpoints, and we should optimize these paths at the typical corner. Our process-variation-aware implementation is shown in Figure 8. In the figure, the *error-detection window* indicates the timing interval, or safety margin, during which a error-tolerant register can detect timing errors. The right-hand side of the figure indicates larger timing slacks at endpoints. When an endpoint has no timing violation at the slow corner (SS, 125°C in our experiments), we use the conventional flip-flop for that endpoint. Introduction of a guardband can enhance design robustness and enable adaptive voltage scaling. Error-tolerant registers can be used for other endpoints. Note that we evaluate error rate at the typical corner (TT, 25°C in our experiments). Therefore, when an endpoint has negative slack with respect to the typical corner, this leads to throughput degradation.

We implement resilient designs at the typical corner, but with CombOpt, there will also be conventional flip-flops in the design. To ensure timing correctness of conventional flip-flops at the slow corner, we estimate slow-corner path delays and apply maximum-delay constraints correspondingly. Specifically, we perform timing analysis at both slow and typical corners, then calculate maximum-delay constraints based on the ratio between delay values at typical and slow corners, as shown in Equation (13). We perform the timing analysis and update the maximum-delay constraints before each iteration of the CombOpt and at the pre-placement, pre-CTS, pre-routing and post-routing (i.e., before signoff) stages.

$$max\_delay = (clock\_period - guardband) \times \frac{delay\_TT}{delay\_SS}. \tag{13}$$

[7]The maximum allowable negative slack is determined by the design of the error-tolerant register.

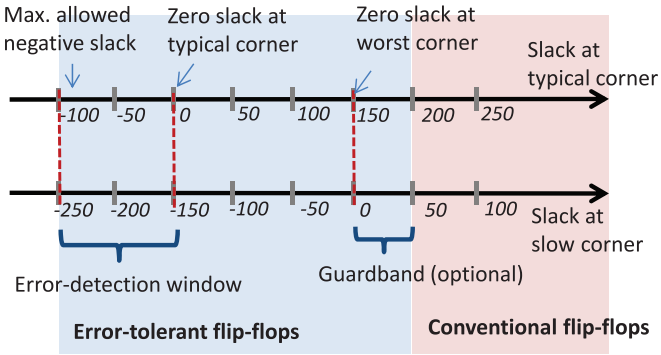

Fig. 8. Illustration of how we consider process variation in our implementations. The slack values shown here are not representative of actual values in 28nm FDSOI.

3.7. Optimization with TIMBER Flip-Flops

We now discuss optimization steps that are specific to TIMBER-based designs [Choudhury et al. 2010]. As described in Section 3.4, our optimization starts with a netlist in which all endpoints with timing violations use error-tolerant registers. To use TIMBER flip-flops, there are two additional constraints for selection of endpoints [Chandra and Choudhury 2014; Fabrie 2014]. First, since TIMBER flip-flops borrow timing from their fanout timing paths to mask timing errors, we must avoid a loop of TIMBER flip-flops within which continuous time borrowing can cause timing failure. Second, additional timing slacks are required at endpoints with conventional flip-flops to compensate the accumulated time borrowings of TIMBER flip-flops in previous stages. Thus, a larger number of chained TIMBER flip-flops will lead to tighter timing constraints on the following timing paths and corresponding increased power and area cost. Moreover, a larger number of chained TIMBER flip-flops requires more complex clock delay circuits, which also incurs area and power cost. In our study, we limit the chained TIMBER flip-flops to be less than three stages (i.e., assuming two error-detection intervals).

To address these additional constraints, we select endpoints based on sensitivity functions which indicate power changes due to the usage of TIMBER flip-flops. A higher sensitivity of an endpoint indicates that greater power reduction is expected from timing margin recovery with TIMBER. Since the maximum stage number of chained TIMBER flip-flops is two, we formulate the optimization as two *maximum-weight independent set (MWIS)* problems in sequence such that no TIMBER flip-flops are adjacent after the first-stage optimization, and no chained TIMBER flip-flops with more than two stages after the second-stage optimization.

We formulate the MWIS problem as an *Integer Linear Program* (ILP).

$$\begin{aligned} &\text{Maximize } \textstyle\sum_{p_i \in P} SF(p_i) \times B_i \\ &\text{Such that } B_i + B_j \leq 1, \quad (p_i \text{ and } p_j \text{ are adjacent in } G) \\ &\qquad\quad B_i = 0 \text{ or } 1, \end{aligned} \tag{14}$$

in which $SF(p_i)$ is the sensitivity function of endpoint p_i, P is the set of endpoints, and B_i is a binary variable indicating whether p_i uses a TIMBER flip-flop (i.e., $B_i = 1$) or not (i.e., $B_i = 0$). The constraints specify that no TIMBER flip-flops are adjacent in graph G. Note that in the first-stage optimization, G is the sequential graph extracted from the netlist. In the second-stage optimization, we update graph G such that we remove the selected vertices from the first-stage optimization. Further, for each removed vertex, we connect each of its incoming vertices to its outgoing vertices.

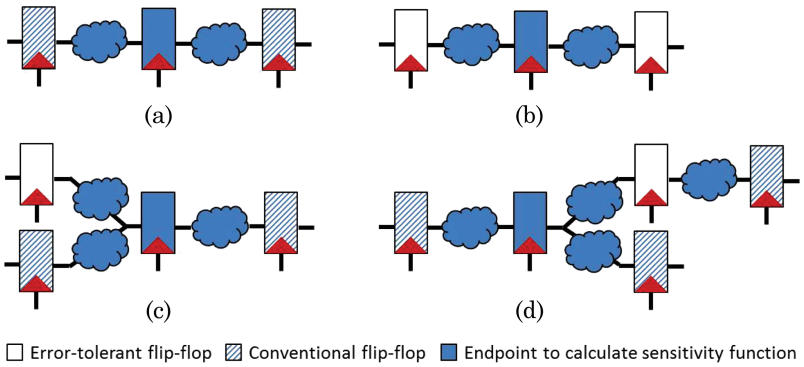

Fig. 9. Scenarios of sensitivity-function calculation for selection of TIMBER flip-flops.

To describe the calculation of sensitivity functions, we first define function $f(c, t)$ as

$$f(c,t) = \begin{cases} |slack(c) - t| \times power(c), & \text{if } slack(c) < t \\ 0, & \text{otherwise,} \end{cases} \tag{15}$$

in which c is a cell in the netlist, $slack(c)$ and $power(c)$ are, respectively, the worst timing slack and power of c, and t is a constant timing interval.

We assume that all endpoints in the initial netlist use conventional flip-flops. By replacing an endpoint with a TIMBER flip-flop, we can recover timing margins in the fanin timing paths by one error-detection window. But this also leads to additional margin insertion, which is equal to one error-detection window in the fanout paths (as shown in Figure 9(a)). To estimate the power change from such timing margin migration, we define the sensitivity of endpoint p as

$$SF_{TBF\_a}(p) = \sum_{c \in cone(c,p)} f(c, t_{ED}) - \sum_{c \in cone(p,c)} f(c, t_{ED}), \tag{16}$$

in which $cone(c, p)$ (respectively, $cone(p, c)$) indicates the combinational logic cone between conventional flip-flops and p (respectively, p and conventional flip-flops), and t_{ED} indicates the duration of the error-detection window.

In the second optimization stage, some of the endpoints have been selected to use TIMBER flip-flops. Therefore, to calculate the sensitivity function values for the remaining endpoints, there are four scenarios, as shown in Figure 9. Endpoints in Figure 9(a) are not promising; otherwise, they would have been selected in the first optimization stage. Selection of endpoints in Figure 9(b) will violate the constraint that there are no chains of more than two TIMBER flip-flops. Based on analysis similar to that shown previously, we calculate the sensitivity functions corresponding to scenarios Figures 9(c) and 9(d) as follows.

$$SF_{TBF\_c}(p) = \sum_{c \in cone(t,p)} f(c, 2 \times t_{ED}) + \sum_{c \in cone(c,p)} f(c, t_{ED}). - \sum_{c \in cone(p,c)} f(c, 2 \times t_{ED}). \tag{17}$$

$$SF_{TBF\_d}(p) = \sum_{c \in cone(c,p)} f(c, t_{ED}) - \sum_{c \in cone(p,c)} f(c, t_{ED}) - \sum_{c \in cone(t,-)} f(c, t_{ED}). \tag{18}$$

Here, $cone(t, p)$ indicates the combinational logic cone between TIMBER flip-flops and p, and $cone(t, -)$ indicates the fanout cone of TIMBER flip-flops which are endpoints of the fanout timing paths of p.

Table I. Testcases from OpenSPARC T1

module	description	# of cells	area (μm^2)
FPU	floating point adder	12,986	34,633
EXU	integer execution	17,614	58,721
MUL	integer multiplier	13,162	40,693
SPU	stream processing	8,066	28,150

Table II. Penalties of Error-Tolerant Flip-Flops

design	Razor	Razor-Lite	TIMBER
power penalty	30% [Das et al. 2009]	~0% [Kim et al. 2013]	100% [Choudhury et al. 2010]
area penalty	182% [Kim et al. 2013]	33% [Kim et al. 2013]	255% [Chen et al. 2013]
# of recovery cycles	5 [Wan and Chen 2009]	11 [Kim et al. 2013]	0 [Choudhury et al. 2010]

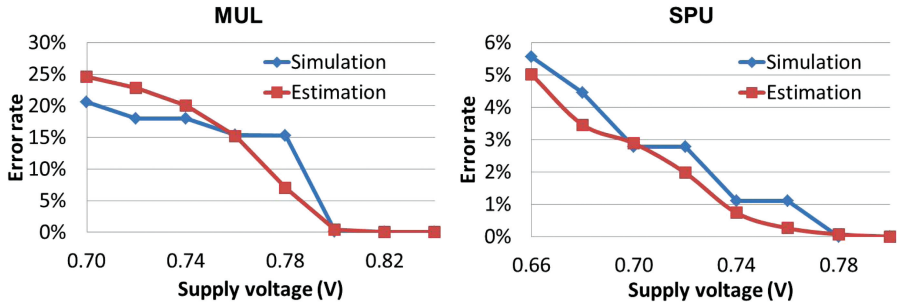

Fig. 10. Actual error rates vs. estimated error rates at different voltages.

4. EXPERIMENTAL RESULTS AND ANALYSIS

4.1. Experimental Setup

We implement experiments with four submodules (Table I) from the *OpenSPARC T1* processor[8]. The modules are implemented in a foundry 28nm FDSOI technology; synthesis is performed with the *Synopsys Design Compiler H-2013.03-SP3*[9], placement and routing are performed with the *Cadence EDI System 13.1*.[10] 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 II.

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 the safety margin and additional hold margin of error-tolerant flip-flops by adding constant shifts to setup and hold constraint values in the Liberty model. To validate our estimation of error rate (Equation (3)) and determine α, we perform gate-level simulation using *Cadence NC-Verilog v8.2*.[10] Figure 10 compares the actual error rates and estimated error rates at different supply voltages based on input vectors with random values. 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 containing 90 commonly used cells (40 combinational and five sequential, with dual-V_T flavors) for each value of supply voltage from 1.20V to 0.50V in 20mV increments, using *Synopsys SiliconSmart v2013.06-SP1*.[9]

[8]http://www.sun.com/processors/opensparc/.
[9]http://www.synopsys.com/.
[10]http://www.cadence.com/.

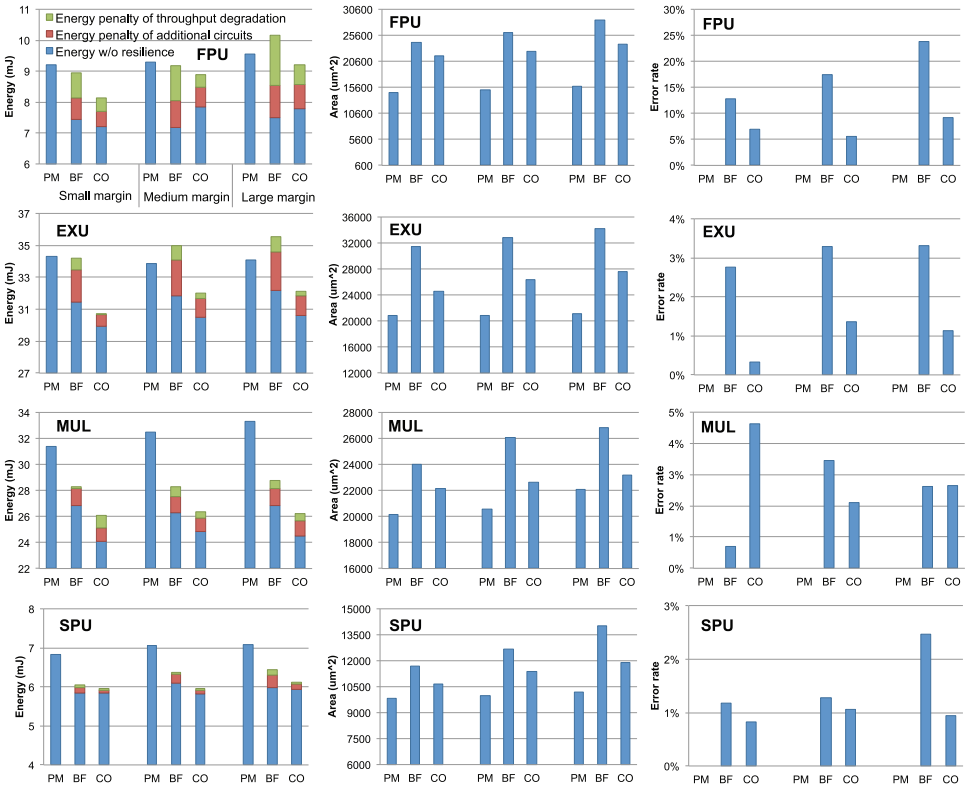

Fig. 11. Energy and area results from different implementation methodologies: pure-margin (PM), brute force (BF), and CombOpt (CO).

4.2. Methodology Comparison

In our first experiment, we compare design energy and area resulting from CombOpt to (i) pure-margin designs[11] and (ii) a brute-force methodology, that is, a typical resilient design implementation, where resilient endpoints are (greedily) instantiated at timing-critical endpoints.[12] We use Razor flip-flops for error resilience in this experiment. We compare our methodology at three different slow corners, where corresponding SS corners are at 1σ, 2σ, and 3σ. The clock period for all implementations is 0.9ns. We use the minimum voltage that satisfies timing constraints at the slow corner for pure-margin implementations; for brute-force and CombOpt implementations, we use a discretized exhaustive search to determine the signoff voltages (i.e., we search for the signoff voltage that achieves minimum energy within -80mV of the signoff voltage of pure-margin, with a step size of 20mV). The runtimes for FPU, EXU, MUL, and SPU are, respectively, 30min, 20min, 60min, and 7min per iteration of CombOpt on a 2.5GHz Intel Xeon server with four threads. We perform 10 iterations of optimization on each design in the experiments.

Figure 11 shows that the benefits of CombOpt increase with a larger process variation. In the figure, small, medium, and large margins, respectively, indicate 1σ, 2σ,

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

[12]We implement designs without considering the safety margin, then replace with error-tolerant registers any endpoints having timing violations with respect to the safety margin.

Table III. Impact of SkewOpt

design	MUL			SPU		
flow	CombOpt	w/o SkewOpt	brute force+ SkewOpt	CombOpt	w/o SkewOpt	brute force+ SkewOpt
total energy (mJ)	26.19	27.07	27.54	6.12	6.18	6.28
throughput penalty (mJ)	1.14	1.16	0.92	0.05	0.08	0.05
# of error-tolerant flip-flops	660	780	1003	225	269	465
total cell area (μm^2)	23200	24315	26352	11922	12324	13931
# of hold buffers	214	321	445	345	107	220

and 3σ for SS corner. We observe that CombOpt achieves up to 10% (8% on average) energy reduction compared to the brute-force method, and up to 21% (12% on average) energy reduction compared to the conventional pure-margin method. With larger process variation, brute-force has larger energy cost due to throughput degradation (e.g., FPU and EXU), while CombOpt is able to jointly minimize the number of error-tolerant flip-flops and error rate, thus achieving greater improvement over brute force.

The additional circuits for error detection typically cause resilient designs to have larger area than conventional designs. We observe that the brute-force method leads to an average of 45% area overhead. Using CombOpt, we reduce the area overhead to 23%. In the regime of "dark silicon" [Esmaeilzadeh et al. 2011], energy cost is "more expensive" than area cost, and our optimization thus focuses mainly on energy reduction.

Figure 11 also shows the comparison of error rates for designs resulting from CombOpt and brute force. We observe that CombOpt achieves smaller error rates on most of the test cases. However, our optimization does not explicitly minimize the error rate of a design, but rather the design energy considering the trade-off between cost of resilience and margin on combinational paths. For example, although for MUL with small margin CombOpt leads to a larger error rate than that resulting from brute force, the power of combinational paths (i.e., indicated by *energy w/o resilience*) is significantly reduced in CombOpt, which leads to smaller design energy.

4.3. Impact of Clock Skew Optimization

As discussed in Section 3, SkewOpt improves slacks of all timing-violating paths with respect to the safety margin to minimize the error rate. Moreover, the improved timing slacks are further exploited to enable removal of error-tolerant registers and power reduction on data paths. To assess the impact of clock skew optimization, we perform optimization without SkewOpt and compare the outcomes with those of CombOpt. As shown in Table III, CombOpt achieves reduced design energy (in terms of both throughput penalty and power of circuit), total cell area, and number of error-tolerant flip-flops as compared to the optimization without SkewOpt. Further, since SkewOpt comprehends hold constraints, performing optimization with SkewOpt does not increase the number of hold buffers significantly. For the MUL test case, applying SkewOpt even reduces the number of hold buffers due to the decreased number of error-tolerant flip-flops. Table III further compares our optimization solution (CombOpt) to that of a conventional resilient design implementation with application of SkewOpt at the post-placement stage. We observe from the fourth and seventh columns of Table III that although SkewOpt alone reduces the throughput penalty, ignoring the trade-off between data path optimization and cost of resilience (which is captured by SEOpt) leads to more error-tolerant flip-flops and significant area and power overheads. Specifically, for MUL, CombOpt achieves 5% and 13% more reduction in energy and area, respectively, as compared to brute force+SkewOpt.

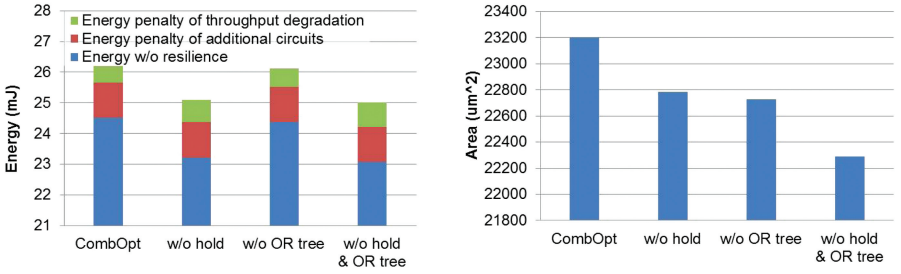

Fig. 12. Impacts of hold margin and error-detection network. Design: MUL (OpenSPARC T1). Technology: 28nm FDSOI.

4.4. Impacts of Short Path Padding and Error-Detection Network

A common observation is that resilient designs require a large hold margin, which necessitates more buffers for short-path padding. We include this overhead by assuming that the required hold margin for a resilient design is equal to its safety margin, that is, 15% of the clock period. Further, the error-detection network (i.e., OR tree) incurs additional energy and area penalties. We evaluate short-path padding and error-detection network overheads by removing the additional hold margin and/or error-detection network in the implementations, and then assessing energy and area differences.

Figure 12 shows the energy and area outcomes for MUL. All implementations are optimized with CombOpt. For the *without hold* case, we ignore the additional hold margin during the implementation; for the *without OR tree* case, we remove the error-detection network at the post-routing stage and perform an incremental optimization; and for the *without hold & OR tree* case, we ignore the additional hold margin during the implementation and remove the error-detection network at the post-routing stage.

Since CombOpt significantly reduces the number of error-tolerant flip-flops (Razor in this example) and the size of the error-detection network, the energy and area cost of short-path padding and error-detection network is small. The short-path padding leads to 4% energy and 2% area cost; and the error-detection network leads to $<$1% energy and 2% area cost. Another reason for the small energy cost of the error-detection network is its low activity. Figure 13 shows an example CombOpt implementation result in which the area overhead of hold buffers and OR gates is only 2.7%.

4.5. Typical-Corner Optimization

Because resilient designs can detect and recover from timing errors, it is not necessary to optimize them at the slow corner. Furthermore, power analysis (especially error rate estimation) of resilient designs at the slow corner (which is the case in [Kahng et al. 2014]) can be pessimistic. We assess the pessimism of slow-corner optimization by performing error rate estimation and energy analysis (of designs shown in Figure 11) at the slow corner.

As shown in Table IV, energy can be overestimated by up to 21% when we perform an energy analysis at the slow corner for resilient designs. This is mainly caused by the overestimated error rates.

4.6. Validation with Different Switching Activities

To validate our proposed optimization and study the impact of switching activity on energy consumption, we analyze the energy of an implemented design (MUL) across a range of switching activity assumptions. Figure 14 shows that CombOpt can achieve minimum energy when the activity factor is no lower than 5%. With a lower activity

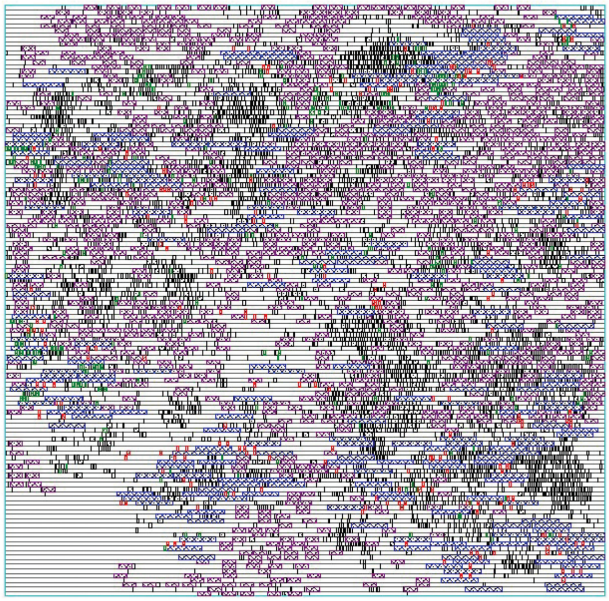

Fig. 13. Layout of CombOpt result for the SPU testcase with 3σ corner. Razor flip-flops are in blue; conventional flip-flops are in purple; OR gates are in red; and hold buffers are in green.

Table IV. Pessimism of Slow-Corner Optimization

design	FPU	EXU	MUL	SPU
typical-corner analysis (TT, 25°C)				
total energy (mJ)	9.21	32.12	26.19	6.12
throughput penalty (mJ)	0.63	0.29	0.54	0.05
slow-corner analysis (SS w/ 3σ, 125°C)				
total energy (mJ)	10.60	34.50	31.62	6.36
throughput penalty (mJ)	1.46	1.77	5.36	0.14

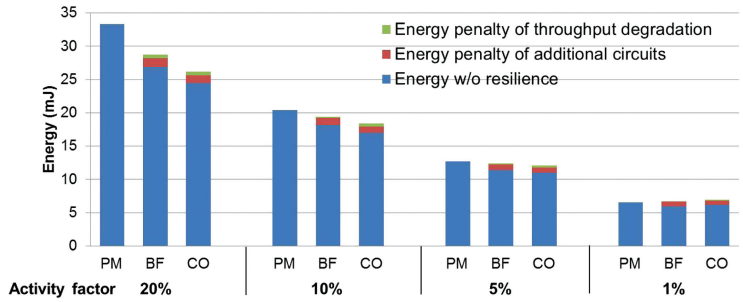

Fig. 14. Energy consumption with different switching activity factors. Design: MUL (OpenSPARC T1). Technology: 28nm FDSOI.

factor (e.g., 1%), error rate decreases and the benefits of CombOpt over the brute-force method become smaller. Compared to pure margin, CombOpt achieves reduced energy of combinational cells through SEOpt. However, with a low activity factor, clock energy dominates, and thus the benefits of CombOpt also decrease.

Table V. Comparison among Error-Tolerant Flip-Flops

design	Razor-Lite	Razor	TIMBER
method	brute-force		
total energy (mJ)	27.71	28.78	33.62
energy w/o resilience (mJ)	27.16	26.87	29.74
energy w/ additional circuits (mJ)	0.00	1.32	3.87
energy w/ throughput penalty (mJ)	0.55	0.59	0.00
# of error-tolerant flip-flops	931	924	575
total cell area (μm^2)	21064	26814	26700
method	CombOpt		
total energy (mJ)	26.14	26.19	28.20
energy w/o resilience (mJ)	25.55	24.51	27.43
energy w/ additional circuits (mJ)	0.00	1.14	0.77
energy w/ throughput penalty (mJ)	0.60	0.54	0.00
# of error-tolerant flip-flops	627	660	128
total cell area (μm^2)	19164	23200	20464

4.7. Study of Different Error-Tolerant Flip-Flops

We also 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 previously mentioned, and CombOpt.

Table V shows results for MUL, where the slow corner is at SS with 3σ and all designs are implemented using CombOpt. We observe that although Razor-Lite has negligible energy and area overheads, it leads to 11% more energy penalties due to throughput degradation compared to Razor. The small number of TIMBER flip-flops is due to additional constraints described in Section 3.7. Further, TIMBER flip-flops require additional timing margin on fanout timing paths to compensate timing errors, which leads to larger energy of combinational cells compared to Razor and Razor-Lite. Note that we also consider the area and power overheads of error relay logic[13] between two stages of TIMBER flip-flops [Fabrie 2014]. We group the TIMBER flip-flops which share the same fanin TIMBER flip-flops and insert the error relay logic for each group. We also observe that CombOpt significantly reduces the number of error-tolerant flip-flops (by 33%, 29%, and 77% for Razor-Lite, Razor and TIMBER[14], respectively). Such reductions can enable to the energy- and area-feasibility of resilient designs.

4.8. Energy Reduction from Adaptive Voltage Scaling

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. Note that to allow voltage downscaling, we build a margin of 25% of the clock period into the paths that have conventional flip-flops as endpoints. In addition, for each test case, we implement a pure-margin design that satisfies timing constraints at minimum voltage as a reference. Figure 15 shows results for our four test cases. The designs implemented with

[13]Based on the error-detection signal of a TIMBER flip-flop, an error relay logic determines the number of time intervals to mask timing errors of the TIMBER flip-flop in the successive stage.

[14]We observe similar improvement (i.e., 7% energy reduction) on an industrial processor (with 13K instances and 1,642 flip-flops) at 40nm technology compared to the brute-force method. In the example, the number of TIMBER flip-flops is reduced from 363 to 10, in which cells in the fanin cones of the selected 10 endpoints consumes 56% of total power in the corresponding conventional design.

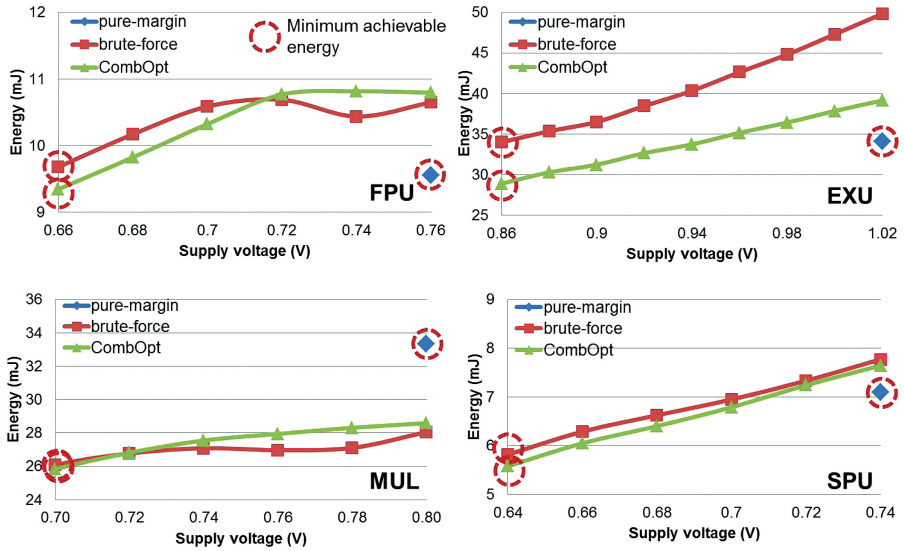

Fig. 15. Energy consumption with voltage scaling and minimum achievable energy for each method.

CombOpt achieve significant energy reduction with voltage scaling. This is because our optimization comprehends the toggle information and trade-off 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 paths with conventional flip-flops as endpoints. 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 8% and 17% energy reductions on average compared to the brute-force and conventional (pure-margin) methods, respectively.

5. CONCLUSIONS

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 nonresilient 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. In addition, we propose a matching-based algorithm to construct the error-detection network with substantially reduced wirelength cost. Our implementations further account for the impacts of sign-off corners and process variation. Our proposed optimization techniques achieve significant energy reductions—up to 21% and 10%—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) find an improved sensitivity metric to determine the minimum set of endpoints to optimize for minimum power and area, and to (2) build a unified framework for simultaneous data- and clock-path optimization.

ACKNOWLEDGMENTS

We are deeply grateful to Dr. Hamed Fatemi of NXP Semiconductors for enabling our experimental studies with an industrial platform. We also thank Dr. Vikas Chandra of ARM, Mihir Choudhury of IBM, and Sebastien Fabrie of NXP Semiconductors for useful discussions about optimization of TIMBER-based designs.

REFERENCES

C. Albrecht, B. Korte, J. Schietke, and J. Vygen. 2002. Maximum mean weight cycle in a digraph and minimizing cycle time of a logic chip. *Discrete Appl. Math.* 123, 1–3 (2002), 103–127.

T. M. Austin. 1999. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the ACM/IEEE 32nd Annual Symposium on Microarchitecture*. 196–207.

T. M. Austin, V. Bertacco, D. Blaauw, and T. Mudge. 2005. Opportunities and challenges for better than worst-case design. In *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*. 2–7.

N. D. P. Avirneni and A. K. Somani. 2012. Low overhead soft error mitigation techniques for high-performance and aggressive systems. *IEEE Trans. Comput.* 61, 4 (2012), 488–501.

K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S. L. Lu, T. Karnik, and V. K. De. 2009a. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE J. Solid-State Circuits* 44, 1 (2009), 49–63.

K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De, and S. Borkar. 2009b. Circuit techniques for dynamic variation tolerance. In *Proceedings of the ACM/IEEE Design Automation Conference*. 4–7.

V. Chandra (ARM) and M. Choudhury (IBM). 2014. Personal communication. (June 2014).

C.-H. Chen, Y. Tao, and Z. Zhang. 2013. Efficient in situ error detection enabling diverse path coverage. In *Proceedings of the IEEE International Symposium on Circuits and Systems*. 773–776.

H. Chen, S. Roy, and K. Chakraborty. 2014. DARP: Dynamically adaptable resilient pipeline design in microprocessors. In *Proceedings of the IEEE Design, Automation and Test in Europe*. 1–6.

M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken. 2010. TIMBER: Time borrowing and error relaying for online timing error resilience. In *Proceedings of the IEEE Design, Automation and Test in Europe*. 1554–1559.

M. R. Choudhury and K. Mohanram. 2009. Masking timing errors on speed-paths in logic circuits. In *Proceedings of the IEEE Design, Automation and Test in Europe*. 87–92.

J. Cong, H. Duwe, R. Kumar, and S. Li. 2014. Better-than-worst-case design: Progress and opportunities. *J. Comput. Sci. Technol.* 29, 4 (2014), 656–663.

S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. 2006. A self-tuning DVS processor using delay-error detection and correction. *IEEE J. Solid-State Circuits* 41, 4 (2006), 792–804.

S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. 2009. Razor II: In situ error detection and correction for PVT and SER tolerance. *IEEE J. Solid-State Circuits* 41, 1 (2009), 32–48.

D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 7–18.

H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. 2011. Dark silicon and the end of multicore scaling. In *Proceedings of the IEEE International Symposium on Computer Architecture*. 365–376.

S. Fabrie. 2014. NXP Semiconductors. Personal communication. (May–July 2014).

J. P. Fishburn. 1990. Clock skew optimization. *IEEE Trans. Comput.* 39, 7 (1990), 945–951.

M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. M. Harris, D. Blaauw, and D. Sylvester. 2013. Bubble razor: Eliminating timing margins in an ARM cortex-M3 processor in 45 nm CMOS using architecturally independent error detection and correction. *IEEE J. Solid-State Circuits* 48, 1 (2013), 66–81.

S. Ghosh and K. Roy. 2007. CRISTA: A new paradigm for low-power and robust circuit synthesis under parameter variations using critical path isolation. *IEEE Trans. Comput.-Aid. Desi. Integr. Circuits Syst.* 26, 11 (2007), 1947–1956.

B. Greskamp and J. Torrellas. 2007. Paceline: Improving single-thread performance in nanoscale CMPs through core overclocking. In *Proceedings of the IEEE International Conference on Parallel Architecture and Compilation Techniques*. 213–224.

B. Greskamp, L. Wan, U. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen, and C. Zilles. 2009. BlueShift: Designing processors for timing speculation from the ground up. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*. 213–224.

A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. 2010a. Recovery-driven design: A methodology for power minimization for error tolerant processor modules. In *Proceedings of the ACM/IEEE Design Automation Conference*. 825–830.

A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. 2010b. Slack redistribution for graceful degradation under voltage overscaling. In *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*. 825–831.

A. B. Kahng, S. Kang, and J. Li. 2014. A new methodology for reduced cost of resilience. In *Proceedings of the ACM Great Lakes Symposium on VLSI*. 157–162.

S. Kim, I. Kwon, D. Fick, M. Kim, Y.-P. Chen, and D. Sylvester. 2013. Razor-lite: A side-channel error-detection register for timing-margin recovery in 45nm SOI CMOS. In *Proceedings of the IEEE International Solid-State Circuits Conference*. 264–265.

Y. Liu, R. Ye, F. Yuan, R. Kumar, and Q. Xu. 2012. On logic synthesis for timing speculation. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 591–596.

Y. Liu, F. Yuan, and Q. Xu. 2011. Re-synthesis for cost-efficient circuit-level timing speculation. In *Proceedings of the ACM/IEEE Design Automation Conference*. 158–163.

V. Subramanian and A. K. Somani. 2008. Conjoined pipeline: Enhancing hardware reliability and performance through organized pipeline redundancy. In *Proceedings of the IEEE Pacific Rim International Symposium on Dependable Computing*. 9–16.

L. Wan and D. Chen. 2009. DynaTune: Circuit-level optimization for timing speculation considering dynamic path behavior. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 172–179.

K-.C. Wu and D. Marculescu. 2010. Clock skew scheduling for soft-error-tolerant sequential circuits. In *Proceedings of the IEEE Design, Automation and Test in Europe*. 717–722.

Y.-M. Yang, I. H.-R, Jiang, and S.-T. Ho. 2013. PushPull: Short path padding for timing error resilience circuits. In *Proceedings of the IEEE International Symposium on Physical Design*. 50–57.

R. Ye, F. Yuan, and Q. Xu. 2011. Online clock skew tuning for timing speculation. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 442–447.

R. Ye, F. Yuan, H. Zhou, and Q. Xu. 2012. Clock skew scheduling for timing speculation. In *Proceedings of the IEEE Design, Automation and Test in Europe*. 929–934.

N. E. Young, R. E. Tarjan, and J. B. Orlin. 1991. Faster parametric shortest path and minimum balance algorithms. *Networks* 21, 2 (1991), 205–221.

F. Yuan and Q. Xu. 2013. InTimeFix: A low-cost and scalable technique for in-situ timing error masking in logic circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*. 183:1–183:6.

G. Zhang and P. A. Beerel. 2014. Stochastic analysis of bubble razor. In *Proceedings of the IEEE Design, Automation and Test in Europe*. 109:1–109:6.

Received July 2014; revised December 2014, March 2015; accepted March 2015