Area Optimization of Timing Resilient Designs Using Resynthesis

Hsin-Ho Huang* *Student Member, IEEE*
Huimei Cheng* *Student Member, IEEE*
Chris Chu‡ *Fellow, IEEE*
Peter A. Beerel*† *Senior Member, IEEE*

***Abstract*—Timing resilient designs can remove variation margins by adding error detecting logic (EDL) that detects timing errors when execution completes within a resiliency window. Speeding up near-critical-paths during logic synthesis can reduce the amount of EDL needed but at the cost of increasing logic area. This creates a logic optimization strategy called *resynthesis*. This paper proposes four alternatives to optimize resilient designs through resynthesis. The first is a brute force approach that explores speeding up all combinations of near-critical paths and produces good results but is computationally impractical for complex circuits. The second is a naive brute-force approach in which near-critical paths are sped up one end-point at a time. It is much faster than the brute-force approach because it does not explore the benefits of speeding up multiple end-points simultaneously and thus provides a quick-and-dirty lower bound for the benefits of resynthesis. The third is a geometric program based iterative algorithm (GPIA) that achieves area reductions that compare favorably across all four approaches. The GPIA algorithm completes within 24 hours for all examples and the average area reduction is up to 16%. Because the run-time required to solve this mathematical model can still be long, however, we propose a fourth approach that involves creating a virtual resynthesis cell library that tries to trick the synthesis tool to understand EDL overhead and optimize total area quickly and automatically. This approach obtains an average of approximately 2/3rds of the area reductions of the GPIA approach with fast run times associated with only a single synthesis run.**

I. Introduction

Traditional synchronous designs must incorporate timing margin to ensure correct operation under worst-case delays caused by process, voltage, and temperature (PVT) variations and cannot take advantage of average-case path activity [1]. This is particularly problematic in low-power low-voltage designs, as performance uncertainty due to process, voltage, and temperature variations grows from as much as 50% at nominal supply to around 2,000% in the near-threshold domain [2]. To address this problem, many design techniques for resilient circuits have been proposed.

For example, canary FFs predict when the design is close to a timing failure (see e.g., [3], [4]). Designs can then adjust their supply voltage or clock frequency either statically or dynamically to ensure correct operation at the edge of failure. Other resilient design techniques use extra logic to detect and recover from timing violations [5], [6], [7], [8]. These techniques use a variety of error-detecting latches and/or flip-flops, initiate replay-and-recovery or slow-down/stall the pipeline in the case of errors, and span both synchronous and asynchronous design styles. All resilient designs exhibit higher performance when there are no timing errors and gracefully slow down in the presence of timing errors, thus achieving a higher average-case performance than traditional worst-case designs. This additional performance can then be traded off for lower power via further voltage scaling.

Of particular importance to this paper is that the error detecting logic (EDL) that is necessary to enable resilient designs represents area and power overhead when compared with traditional worst-case designs. Thus the error detecting logic must represent a relatively small fraction of the design to not overshadow the obtained average-case performance benefits. Circuit and EDA techniques to minimize the EDL overhead will thus be important for these techniques to flourish.

Our proposed EDA technique to minimize EDL is *resynthesis [8], [9]* which involves speeding up near-critical-paths during logic synthesis with a tighter max_delay constraint to reduce the amount of EDL needed at the cost of increasing combinational logic area. The challenge in resynthesis is that many paths share logic and it is not obvious which combination of paths should be constrained to achieve the best gains. This paper presents and compares four different approaches, a brute-force, a naive brute-force, a gate-sizing-based iterative geometric program, and a low complexity library based method, that try to find the optimal combination of paths to constrain. It extends two previous publications: [8] which, as part of a complete design flow for resilient designs, introduces the naive brute force approach; and, [10], which presents a formal framework for area optimization of flop-based resilient designs guided by a gate-sizing-based geometric program. Moreover, this paper also shows how all the optimization approaches can be applied to both flop and latch-based resilient designs.

The remainder of this manuscript is organized as follows. First, Section II reviews previous and related work on optimizing resilient circuits. Section III describes a variety of different error-detecting sequential latches and flip-flops and their associated resiliency approaches. Section IV, then, formalizes the minimum area resynthesis problem for resilient designs. Section V describes the brute-force approach while, Section VI then describes the more computationally practical naive brute-force approach. Our gate-sizing-based iterative geometric program based approach is described in Section VII, and Section VIII presents our low-complexity library based approach. Section IX presents our experimental results comparing all approaches, which is followed by some conclusions presented in Section X.

II. Related Work

Some related EDA techniques focus on minimizing the probability of timing errors in resilient circuits [11], [12], [13],

This research is partially supported by NSF Grant CCF-1619415 and CCF-1219100 and a gift from Qualcomm, Inc.

*H. Huang, H. Cheng and P. A. Beerel are with the University of Southern California, Los Angeles, CA.
†Peter A. Beerel is also Chief Scientist at Reduced Energy Microsystems.
‡Chris Chu is with the Iowa State University, Iowa, IA.

[9], [14]; some focus on improving performance [15]; others focus on minimizing power consumption [16], [17].

Liu et al. [11] proposed to reorder the fanins of logic gates by their arrival time to reduce delays and lower timing errors. Dynatune [12] optimizes throughput by selectively choosing low V_{th} gates to speed up near-critical paths. Ye et al. [13], Kahng et al. [9] and [14] all use clock skew scheduling to reduce timing errors. Greskamp et al. [15] introduce two techniques to speed up most frequently access near-critical points: on-demand selective biasing which applies forward body biasing to one or more gates and path constraint tuning which reduces the delay of desired path at the expense of other paths. For power optimization, Kahng et al. [16], [17] use slack redistribution to optimize timing. In particular, they use cell swapping and re-sizing to improve timing followed by voltage scaling to trade the higher performance for lower power.

In this work we instead are interested in reducing the area of resilient designs. Other proposed EDA techniques that focus on area reduction include [18], [8], [9]. Choudhury et al. [18], rather than using error detecting logic to detect errors, propose an error-masking solution that makes the design immune to timing errors. [8], [9] and our proposed algorithms use resynthesis to minimize area that consists of constraining certain near-critical end-points to be non-near-critical and re-running logic synthesis. As part of the Blade [8] resilient flow, we explored a naive brute-force resynthesize method. Near-critical paths are constrained one end-point at a time to find the best single end-point to constrain during logic resynthesis. Kahng et al. proposed sorting near-critical endpoints by heuristic sensitivity functions and iteratively increasing the set of endpoints to speed up [9]. They then chose the synthesized resilient design with minimum overall area. Both methods achieved significant area and performance benefits and were demonstrated to be computationally practical but they lack a formal model from which we can explore any notion of optimality.

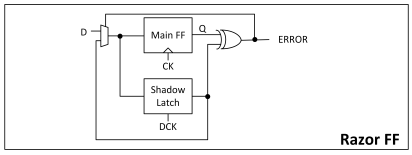
This paper first proposes a straight-forward but computationally expensive method to explore speeding up all possible combinations of near-critical end-points and pick the combination that leads to the lowest overall area. We refer to this approach as the brute-force approach. Unfortunately, as the circuit size grows, the run-time quickly becomes too high. This paper also evaluates the naive brute-force approach, first proposed in [8], that only speeds up one near-critical end-point at a time. This approach addresses the complexity issue of the brute-force approach, but often leads to far from brute-force minimum overall area results. This paper next presents a mathematical optimization framework based on a iterative geometric program. The program's constraints capture the shared logic among paths and thus more accurately guide which particular near-critical-paths to speed-up compared to the naive brute-force method. While lower complexity than the brute-force approach, the run-time required to solve this mathematical model can still be long, particularly for large circuits. We therefore also introduce a virtual resynthesis cell library approach in which modify the standard cell libraries to trick the synthesis tools to automatically select between normal sequential gates and EDL flops/latches. This method has low-complexity, consisting of only one synthesis run, and does effectively minimize total area of the circuit, including the area overhead of the error detecting logic.

III. Background

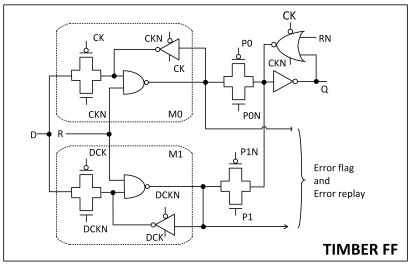
Resilient designs offer the promise to remove increasingly large margins due to process, voltage, and temperature variations and take advantage of average-case data but require extra error detecting logic to ensure correctness. In particular, in addition to the regular sequential gates, resilient designs usually require some form of shadow sequential gate for each near-critical sequential element. If the value in the regular and shadow sequential gate differ, there is a timing violation which is corrected using either some form of architectural replay or some form of clock-gating, power supply, or delay line manipulation in the downstream logic.

There are variety of different timing-resilient templates for both asynchronous [8], [19] and synchronous designs [20], [5], [21], [22], but one can categorize them into two groups: flop-based resilient templates and latch-based resilient templates. Flop-based resilient templates preserve the property of edge-sampling in the main datapath. Timing errors cannot be accounted for by simply adjusting down-stream logic and, instead, the correction involves some form of architectural replay [20], [5], [21]. Latch-based resilient designs, on the other hand, typically use the transparent phase of the latch as the timing resiliency window (TRW) [8], [22], [23], simultaneously detecting an error and allowing the new data to pass to the downstream pipeline stage. Consequently, these designs can account for the late propagation of data by proper adjustment to the timing of downstream stages.

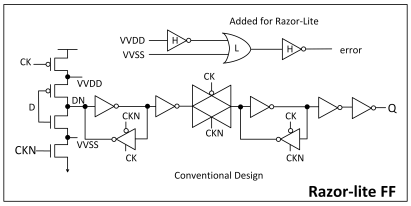
To appreciate the range of overheads of the associated error detecting logic, we review several of error detecting sequential elements in more detail. Figure 1 shows three different resilient flip-flops. The first is the Razor flip-flop [20], illustrated in Figure 1(a). In Razor, each near-critical flop is augmented with a shadow latch that is connected to a delayed clock. Because of the delayed clock, the shadow latch will always latch the correct input data. Thus, if data changes after the clock rises, the output of flip-flop will differ from that of the shadow latch and an error is asserted. When an error occurs, the controller tries to re-execute the instruction to recover from the error. The second is the TIMBER (Time Borrowing and Error Replaying) flip-flop [5], shown in Figure 1(b), which has two master latches ($M0$ and $M1$) and a slave latch. $M0$ and the slave latch combine to form a regular flip-flop and $M0$ samples the value of D when CK rises and drives the slave latch and Q. When CK falls, the transmission gate $P0$ will open and the slave latch passes the value of Q. When there is a timing violation, the value of $M0$ and $M1$ will differ and the controller will trigger a pipeline flush and architectural replay to resolve the error. The Razor-lite flip-flop [21] is illustrated in Figure 1(c). It reduces the large overhead of Razor flip-flop using a lightweight side-channel detection mechanism that avoids adding circuitry to the clock and data pins of the design at the cost of reduced noise margins. When the clock rises, one of the virtual rails ($VVDD$ or $VVSS$) will float while the other remains connected to DN. If the value of D changes while clock is high, the other virtual rail will connect to DN and will subsequently (dis)charge through the feedback inverter of the master latch, allowing a monitoring circuit to



(a) Razor flip-flop

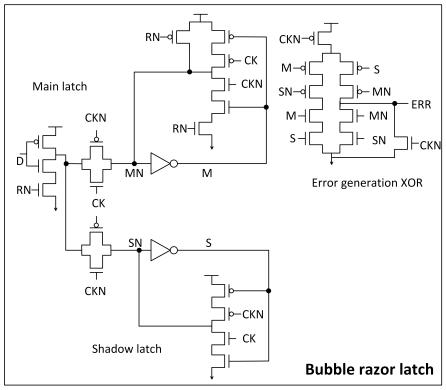


(b) TIMBER flip-flop

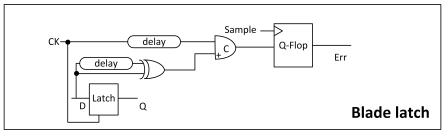


(c) Razor-lite flip-flop

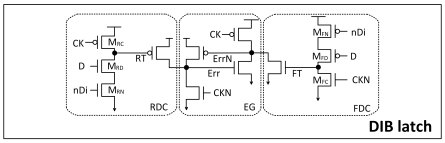
Figure 1: Flop-based resilient templates



(a) Bubble razor latch



(b) Blade latch



(c) DIB latch

Figure 2: Latch-based resilient templates

detect the transition at D and an error will be detected via the upper OR gate. Note here H stands for high transition voltage and L stands for low transition voltage.

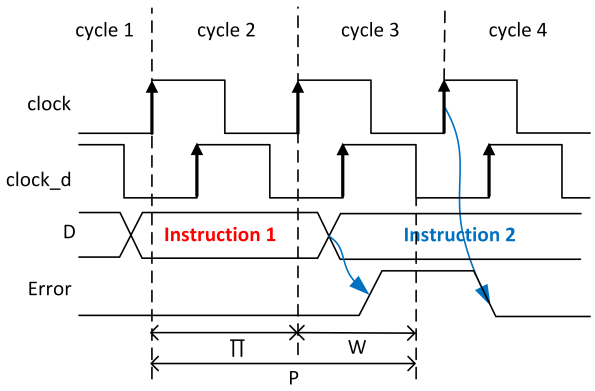
Figure 2 presents three different error detecting latches. The first is the Bubble razor latch [22], shown in Figure 2(a), which uses a shadow latch to detect timing errors. If data arrives after the latches open, the error generation XOR will flag an error. Then, control circuits act to recover from the error by pausing the clock and inserting bubbles to neighboring stages. A bubble causes a latch to skip its next transparent clock phase, giving it an additional cycle for correct data to arrive. The second is a time-borrowing error detecting latch used in Blade [8]. It uses a latch, an XOR, a C-element, and a Q-flop, where the Q-flop provides metastability-free sampling of the error condition. If data changes after the latch becomes transparent, the Err signal will be asserted. Blade resilient controllers resolve the error by adding extra delay to the handshake with downstream pipeline stages. The third is a delay-input-based (DIB) error-detecting latch [23] shown in Figure 2(c). It consists of a rising detecting circuit (RDC), a falling detecting circuit (FDC), and an error generator (EG). It also relies on signal nDi, a delayed complement of D. When there is a falling or rising transition on D during detection window, M_{FD} and M_{FN} or M_{RD} and M_{RN} will be simultaneously on for a short period of time because of the long delay between D and nDi. FT will be charged or RT will be discharged to assert the error. This design can be amortized across many data inputs, reducing the overhead significantly [23], but is more complex requiring more dynamic nodes and, thus, may be less robust to noise than its larger counterparts.

Figure 3 shows the clock timing diagrams of a resilient design that resolves errors with a single clock cycle penalty (see e.g., [24]). In Figure 3(a), Instruction 1, shown in red, launches at cycle 2 but its delay exceeds the clock period Π and only settles in the subsequent timing resiliency window of size W. The error detecting logic corrects this in the subsequent cycle delaying the processing of Instruction 2 until clock cycle 4.

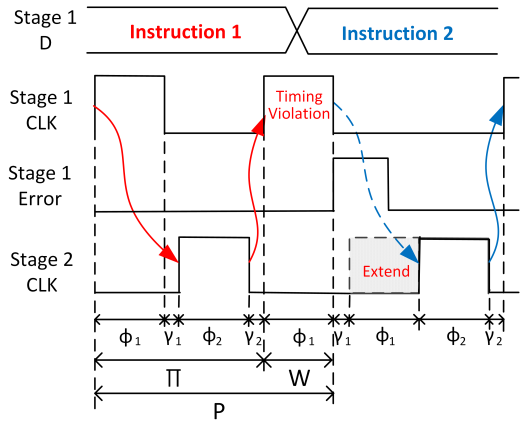
Resilient latch-based designs have more flexibility because the time-borrowing capability of latches provides some inherent resilience to timing variations. Figure 3(b) shows a latch-based resilient design with two stages in a loop, where error-detecting and time-borrowing stages alternate. ϕ represents how long the latch is transparent and γ represents the non-overlap period between neighboring latches. The figure illustrates the case of a timing violation detected while Stage 1 *CLK* is transparent. Such a violation occurs when the delay of the combinational logic that feeds Stage 1 latches (i.e., between Stage 2 and Stage 1) exceeds $\phi_2 + \gamma_2$ but is less than $\phi_2 + \gamma_2 + \phi_1$. To recover from the violation, as illustrated in the figure, the rising edge of Stage 2's *CLK* signal is delayed by ϕ_1.

IV. Problem Statement

In this section, we formally define our minimum area resiliency-aware resynthesis problem. We are given a gate-level VLSI circuit with combinational gates (C), and sequential gates (S) and a desired speculative window (W). Each gate has a list of input/output pins (I/O) and a list of fanout gate and pin pairs (FO). Each input pin of a gate has a delay (D), and a fanin gate (FI). We consider both flop-based and latch-based designs as



(a) Flop-based resilient designs



(b) Two phase latch-based resilient designs

Figure 3: CLK timing diagrams for resilient designs

illustrated in Figure 3. For the flop-based designs we assume every pipeline stage is error-detecting as in Figure 3(a) and for the latch-based designs, as proposed in [8], we assume that the clock cycle of resilient latch-based design starts with one error-detecting stage and is followed by zero or more time-borrowing stages. Even in the error-detecting stages, only those sequential gates that are near-critical need be error-detecting.

To formalize this more, every gate i is given an arrival time (T_i) at its output. We calculate the arrival time of each combinational gate $i \in C$ as follows:

$$T_i = max_{\forall k \in I(i)}(D_{i_k} + T_{FI(i_k)}) \tag{1}$$

where D_{i_k} is the pin-to-pin delay from the k fanin of i to its output.

The sequential gates can be flip-flops or latches. A flip-flop or latch i in an error detecting stage marks the beginning of a clock period and its arrival time is

$$T_i = D_i, \tag{2}$$

where, D_i represents the clock to Q delay. Based on the clock timing diagram of latch-based designs in Figure 3(b), for a latch i in time-borrowing phase $j > 1$ with data input l, the arrival time equation is:

$$T_i = D_i + max(\sum_{k=1}^{j-1}(\phi_k + \gamma_k), T_l) \tag{3}$$

Here, the summation calculates the time when latch i opens and T_l is the arrival time of the data input to latch i. T_l will be greater than summation only when time-borrowing happens.

All latches in time-borrowing stages ($j > 1$) with data input l also have a setup time constraint:

$$T_l \leq \sum_{k=1}^{j-1}(\phi_k + \gamma_k) + \phi_j. \tag{4}$$

whereas the setup time constraint for a flip-flop or latch in an error-detecting stage i with data input l is

$$T_l \leq P \tag{5}$$

where P is the max delay between error-detecting stages.

It is important to emphasize that all sequential gates in a non-error-detecting stage must not be error-detecting, but every sequential gate in an error-detecting stage may or may not be error-detecting. We, thus, introduce a binary variable e_j whose value is determined by whether the j^{th} sequential element must be error detecting or not. In particular, the determination of whether the j^{th} sequential element must be error-detecting or not is based on the arrival time of its input. If the arrival time of its data input is prior to the speculation window, the paths that end at this sequential element are not near critical and the sequential element need not be error-detecting, i.e., $e_j = 0$. Otherwise, the arrival time must be within the resiliency window $[(P - W), P]$ and the sequential element is deemed near-critical and must be error-detecting, i.e., $e_j = 1$. More mathematically, for a flop or a latch j in an error-detecting stage with data input l, we have

$$e_j = \begin{cases} 1, & \text{if } T_l \geq P - W, \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Note that in latch-based resilient designs the resiliency window W is often set to the transparency phase of the error-detecting latch.

We create a variable $\mathbf{E}$ in Equation (7) to represent the array of all e variables and we assume there is a function $Y(\mathbf{E})$ which returns the minimum logic area needed to satisfy the above timing constraints given the assignment of sequential gates to error-detecting or not.

$$\mathbf{E} = [e_1, e_2, ..., e_n] \tag{7}$$

Our approach assumes the relative additional cost associated with each latch or flip-flop that must be error detecting is a constant X, whose value can be set based on the resiliency scheme adopted and the specific choice of error-detecting sequential gate adopted. In particular, our area model is:

$$min\ (Y(\mathbf{E}) + X * \sum_{e_j \in \mathbf{E}} e_j) \tag{8}$$

The first part of the model is the sum of the total logic area of all gates assuming all sequential elements are not error detecting. The second part of the model is the area overhead associated with the subset of sequential gates that must be error detecting. As illustrated in Section III, the specific structure of the error-detecting latches/flops vary among resilient designs and, consequently, have different associated overheads. For

example, the time-borrowing transition detection latch illustrated in Figure 2(b) has an area overhead of approximately 2X that of regular latch, where as the DIB template in Figure 2(c) has an amortized area overhead of only about 0.5X that of regular latch [23]. For these reasons, our experiments are conducted with a range of different values of X representing low, medium, and high values of the area overhead. Namely, we choose X to be 0.5, 1, and 2 times the area of a minimum-sized sequential gate. These settings are referred to as low, medium, and high overheads.

We can now define the *minimum area resiliency-aware resynthesis problem* as finding the assignment of $\mathbf{E}$ that achieves the minimum total area (8) subject to the above timing constraints (6). Equations (1)-(2) and (5)-(8) apply to flip-flop-based designs and Equations (1)-(8) apply to latch-based designs.

Unfortunately, the worst-case number of assignment to $\mathbf{E}$ is $2^{\|\mathbf{S}\|}$ and finding the best assignment is non-trivial, particularly because the sharing of paths in the combinational logic favors some combinations and makes others unrealizable. Hence, in this manuscript, we describe four different approaches aimed at solving this problem.

V. Brute-force Approach

As a pre-cursor to running resynthesis, we have the logic synthesis tool generate the list of end-points that are near-critical and thereby need to be terminated with EDL. The brute-force approach consists of running many resynthesis runs, each speeding up a different combination of near-critical end-points by constraining them to have arrival times before the timing resilient window begins. In addition, we constrain all non-near-critical end-points to remain non-near-critical. When the number of near-critical end-points is small, we can explore speeding up all their possible combinations. However, when the number of near-critical end-points grow, the number of resynthesis runs needed to explore all combinations is too high and this approach looses practicality. In particular, the complexity of the brute-force algorithm in terms of the number of resynthesis run is $O(2^{\|\mathbf{NCE}\|})$ where NCE represents the number of near-critical end-points. It is, thus, useful only as a baseline approach. In particular, the result is optimal if any area gains associated with relaxing the constraints on the originally non-near-critical end-points is negligible. Figure 4 shows all combinations of speeding up near-critical end-points of the ISCAS89 benchmark circuit s1196 [25]. After synthesis, there are 3 near-critical end-points which requires 7 combinations resynthesis run; however, two combinations in the plot overlap making only 6 points visible. The best point, highlighted in red, yields a 0.88 area ratio, and 0.25% improvement in error rate. In particular, the error-rate is use-case dependent and defined as the portion of simulation cycles in which timing errors occur. Errors can be triggered by different sets of error-detecting sequential gates and the number of such sets is exponential in the number of error-detecting sequential elements. Note that the potential benefits of this resynthesis approach will heavily depend on the initial timing constraints, i.e. a design that is already tightly constrained cannot easily be constrained further to achieve area and performance benefits.

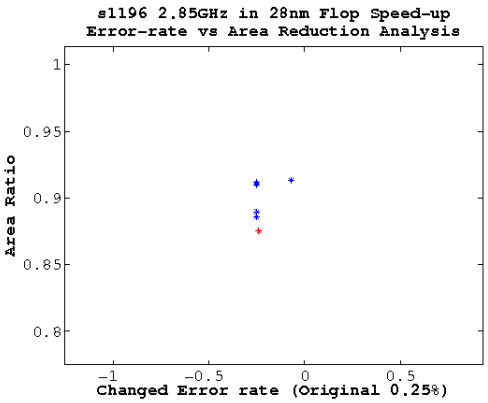


Figure 4: Area ratios and error rate changes via the brute-force method on circuit s1196

VI. The Naive Brute-force Approach

The naive brute-force approach is similar to the brute-force approach in that we first have the synthesis tool generated a list of near-critical end-points. However, the naive brute-force approach explores the impact of tightening the timing constraints of only one near-critical end-point at a time. In particular, for each near-critical end-point, we constrain it to complete before the timing resilient window begins and resynthesize the design, that is we force $T_l \leq P - W$. This approach is not optimal because it does not explore re-constraining combinations of end-points, but the number of resynthesis runs grows linearly with the number of near-critical end points, ensuring the approach remains computationally tractable.

Although the combinational area may increase due to the tighter constraint on the chosen end-point, this overhead can be offset if multiple flops/latches that were slated to become error-detecting are, after resynthesis, no longer near-critical. In particular, it is important to emphasize that the high degree of shared paths in the combinational logic makes it challenging to estimate the reduction in the number of end-points that are actually sped up after each resynthesis run, i.e., constraining one near-critical latch/flop to become non-near-critical may also speed up many other near-critical latches/flops. We also note that the reduction of EDL combined with faster combinational logic may lead to a reduced frequency of timing violations during simulation, which improves the average performance of the circuit.

Figure 5 shows the results of naive brute force re-constraining the various near-critical end-points of the ISCAS89 benchmark circuit s9234 [25]. The best point, highlighted in red in Figure 5, makes 23 near-critical end-points non-critical, yielding a 0.86 area ratio and a 0.37% improvement in error rate. This approach, first presented in [8], while not optimal, served as simple and effective way to ascertain if significant benefits can be obtained with resynthesis. We note that extensions of this approach that either blindly iterate resynthesis [8] or use heuristics to guide which additional end-points to constrain [9] are also possible but outside the scope of this paper.

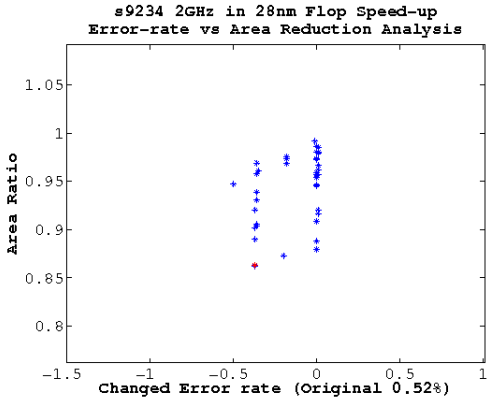


Figure 5: Area ratios and error rate changes via the naive brute-force method on circuit s9234

VII. Gate-Sizing Model-Based Area Optimization Approach

In this section, we solve the minimum area resiliency-aware resynthesis problem using a gate-sizing based model of delay and geometric programming. Geometric programming enables large-scale non-linear mathematical problems to be solved, but requires both the objective function and the inequality constraints to be posynomial [26], [27]. In particular, a posynomial is a function of the form

$$f(x_1, x_2, ..., x_n) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} ... x_n^{a_{nk}}$$

where all the coordinates x_i and coefficients c_k are positive real numbers, and the exponents a_{ik} are real numbers. Posynomials are closed under addition, multiplication, and non-negative scaling. The complexity of geometric program is polynomial of the number of variables and constraints in the program.

Section VII-A shows how we formulate the resynthesis problem described above as a mixed integer geometric program. Section VII-B, then, explains how we efficiently solve the mixed integer geometric program by relaxing the integer variables to be real and using an iterative geometric program to find, what our experiments indicate are, close to integer solutions.

A. Mixed Integer Geometric Program Formulation

In this section, we remodel the mathematical problem in Section IV into a geometric program by modeling the generic mapping function of timing constraints to area Y using a gate-sizing-only optimization model. We are given a gate-level VLSI circuits with combinational (C) and sequential (S) gates with gate sizes (z). Each gate has a nominal area (A), a list of input/output pins (I/O) and a list of fanout gate and pin pairs (FO). Each input pin of a gate has a nominal resistance (R), a nominal input capacitance (Cin), and a list of fanin gates (FI). The delay D_{i_k} of the k^{th} pin of gate i with size z_i is modeled using an Elmore delay model:

$$D_{i_k} = \mu * \frac{R_{i_k}}{z_i} * \sum_{j_l \in FO(i)} Cin_{j_l} * z_j \tag{9}$$

in which $z_i = 1$ represents the nominal size of the gate and μ is a constant scaling factor, typically set to 0.69.

Recall that in Section IV we introduced a binary variables e_j that determines if sequential element j need be error-detecting. This variable is governed by the constraint:

$$T_i - W * e_j \leq P - W, \tag{10}$$

where i is the data input of the j^{th} sequential element. That is, e_j can be 0 only if the arrival time at i is prior to the speculative window. Moreover, if e_j is 1, the arrival time i must still be before the cycle time P. Unfortunately, the subtraction on the left hand side of the constraint makes the constraint not posynomial [27].

To address this problem, we perform a change of variables, introducing a new variable ne as follows:

$$e_j = 2 - ne_j,\ 1 \leq ne_j \leq 2,\ \forall j \in S \tag{11}$$

The delay constraint now becomes

$$T_j + W * ne_i \leq (P + W), \tag{12}$$

which is posynomial.

Substituting ne_j into the EDL portion of the objective function described in Equation (8) yields:

$$X * \sum_{j \in S} (2 - ne_j),$$

which is unfortunately also non-posynomial. We thus make an approximation to the objective function, creating the complete mixed integer geometric program for flop-based resilient designs as follows:

$$Minimize\ (\sum_{i \in C,S} (A_i * z_i) + X * \sum_{j \in S} (\frac{2}{ne_j} - 1)) \tag{13}$$

Subject to:

$$D_{i_k} = \mu * \frac{R_{i_k}}{z_i} * \sum_{j_l \in FO(i)} Cin_{j_l} * z_j,\ \forall k \in I(i)\ \forall i \in C, S \tag{14}$$

$$\begin{cases} T_i \geq max_{\forall k \in I(i)} \{(D_{i_k} + T_{FI(i_k)})\},\ \forall i \in C \\ T_i = D_i,\ \forall i \in S \end{cases} \tag{15}$$

$$T_j + W * ne_i \leq (P + W),\ \forall i \in S,\ j \in FI(i) \tag{16}$$

Bounds:

$$\begin{cases} LB_i \leq z_i \leq UB_i,\ \forall i \in C \\ z_i = 1,\ \forall i \in S \end{cases} \tag{17}$$

$$1 \leq ne_i \leq 2,\ \forall i \in S, ne_i \in \mathbb{Z} \tag{18}$$

$$0 \leq T_i \leq P,\ \forall i \in C, S \tag{19}$$

More specifically, the EDL part of the modified area cost function for sequential element i is changed from the non-posynomial form $2 - ne_i$ to the posynomial form (2/ne_i - 1). This keeps the cost function the same for all possible (integer) values of ne_i. In particular, when e_i is 1, ne_i will be 1 and (2/ne_i - 1) will remain 1 as e_i. When e_i is 0, ne_i will be 2 and (2/ne_i -1) will be 0 as e_i.

```
iteration = 0; L = 1; H = 2;
while(L < H){
   if(Solution_found) {
      if all (ne_j == 1 || ne_j == 2) break;
      L = lth * iteration + 1;
      H = 2 - hth * iteration;
      foreach j in sequential gates {
         if(ne_j <= L) ne_j = 1;
         else if(ne_j >= H) ne_j = 2;
      }
      iteration++;
   } else
      AllowHighCostNegativeSlack();
}
%cross each other
L = lth * (iteration - 1) + 1;
H = 2 - hth * (iteration - 1);
Middle = (L + H) / 2;
foreach j in sequential gates{
   if(ne_j <= Middle) ne_j = 1;
   else ne_j = 2;
}
```

Figure 6: Pseudo-code of the geometric program based iterative algorithm (GPIA)

Note that the constraints in Equations (14) and (15) implement the same Elmore delay model described above and the arrival time equations described in Section VII-A. Moreover, Equations (17) and (18) show the bounds of all variables that can be set to avoid unrealistic changes in size.

To extend this formulation to latch-based resilient designs, we add the constraints of the arrival time and the setup time defined in Equations (3)-(4) to the above formulations.

B. Geometric Program Based Iterative Algorithm (GPIA)

The integral constraint on ne_i generally adds significant computational complexity to the mathematical program because they are typically handled using computationally expensive branch-and-bound techniques [26]. To address this, we propose a more efficient solution allowing these variables to be any real value between 1 and 2 within an iterative outer loop. In particular, after each iteration, we use a high threshold and low threshold to force some ne_i variables to integer values for future iterations. After setting some variables to be integral, we squeeze the high and low thresholds closer together and repeat. We iteratively run the geometric program until the high threshold and low threshold cross or all ne variables are set to integer values. The pseudo-code of this relaxation-based algorithm is shown in Figure 6.

Figure 7 shows an example of how the high and low thresholds are varied across iterations. If the high threshold step (Hth) is 0.1 and low threshold step (Lth) is 0.2, then if value of ne variables from 1^{st} run is greater than 1.9 (less than 1.2), the ne variables will be fixed to 2 (1) for the next iteration. In the second iteration, we force ne variables greater (less) than 1.8 (1.4) to be 2 (1). In this example, the maximum number of iterations is 4 because at this point the high and low thresholds cross. When high and low threshold cross, we find the mid-point by averaging the high and low thresholds of the previous iteration. Then, if the variable ne is greater than the mid-point, ne is set to 2. Otherwise, it is set to 1.

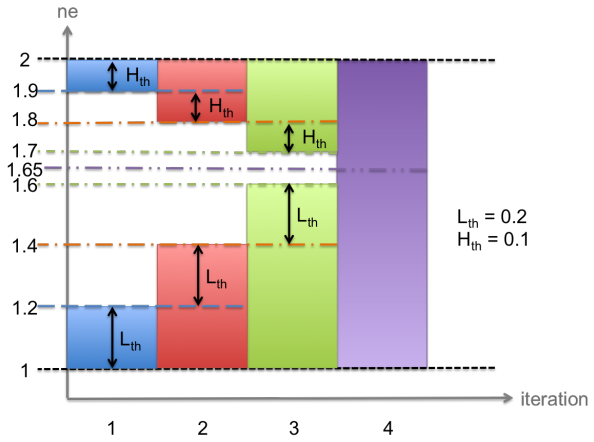


Figure 7: Example of how high and low thresholds vary across iterations

The threshold steps may play an important role on the quality of the results. Iterations with small threshold steps might have more similar critical paths as the solution with integer variable of ne. However, this comes at the cost of needing more iterations and thus higher runtimes. On the other hand, larger threshold steps can lead to reduced runtimes but have a higher chance to lead to ne variables that have larger differences from the values obtained by the integer program.

C. Calculating Gate Resistance and Capacitance

Note that, in the geometric programs, we use the Elmore delay model discussed in Section VII. For each gate in the original synthesized netlist, we obtain its nominal area (A) from the synthesis library. For each pin of each gate, we obtain its nominal input capacitance from the synthesis library and its nominal pin-to-pin delay from the initial synthesis timing report. We, then, use our Elmore delay model with $z = 1$ to back-calculate the nominal resistance (R) of each pin of each gate. Based on this pin-to-pin model, the geometric program can calculate the delay with different sizes.

VIII. Virtual Resynthesis Cell Library Approach

This section proposes an alternative optimization approach that involves creating a virtual resynthesis cell library that essentially tricks the synthesis to automatically select between normal and error-detecting sequential gates in error-detecting pipeline stages. For latch based designs, we triple all latches in the cell library. The latches in the first group, the *non-error-detecting sequential gates* in error-detecting pipeline stages, are the normal sequential gates whose setup times are adjusted to make sure the data arrives before the resiliency window W begins.

The latches in the second group, the *error-detecting sequential gates*, have their area modified to include the expected EDL overhead. For example, for the high-area overhead model ($X = 2$), the area of sequential gates in the second group is tripled. The latches in the third group, the *normal sequential gates*, are for non-error-detecting pipeline stages and are the original and unmodified sequential gates given in the standard-cell library. Similarly, for flop-based designs, we double all flops in the cell library, creating error-detecting versions with

high area and non-error detecting versions with high setup times.

In the synthesis tool, we map all sequential gates in non-error-detecting pipeline stages to the regular sequential gates and we let the tool choose either the error-detecting or non-error-detecting sequential gates for all error-detecting stages. This method has low complexity requiring only one synthesis run and will effectively minimize total circuit area, including the area overhead of the error detecting logic.

It may be useful to contrast this approach to the three approaches described above. To overcome the fact that commercial synthesis tools do not inherently understand the overhead of EDLs, the brute-force approaches are designed to directly assign which sequential gates will be error-detecting. They have to re-run synthesis to evaluate each combination, leading to either sub-optimality due to not exploring all combinations or impractically long run-times. As a compromise, our geometric program model uses a restricted model of synthesis based on gate-sizing to guide which sequential gates should be error-detecting. Because synthesis tools employ far more than gate-sizing during synthesis, this approach is also suboptimal. The virtual resynthesis cell library approach solves this problem, enabling the synthesis tool to understand the overhead of EDLs and the timing constraints associated with their use. Thus, in principle, it enables the choice of which sequential gates to be error-detecting in the context of the full suite of logic optimizations.

In practice, however, we must recognize that synthesis tools are not typically asked to pick between such widely differing sequential gates and thus may not be designed to do this well. The results of resynthesis may, thus, depend on the specific logic synthesis tool used and the proprietary algorithms surrounding sequential gate selection and optimization. Moreover, depending on the optimization algorithms of the synthesis tool, the starting point of resynthesis may have an impact on the results, a point we explore in our experimental results. In particular, we can initially map all sequential gates in error detecting pipeline stage to either error-detecting, non-error-detecting, or a mix of error-detecting/non-error-detecting gates based on timing. During re-synthesis, the synthesis tool must focus on area reduction when all the latches are initially error-detecting gates and on fixing timing violations when all the latches are initially mapped to non-error-detecting gates.

IX. Experimental Results

We implemented the four proposed approaches using Perl and TCL scripts that interface a leading commercial logic synthesis tool to the YALMIP Geometric Program solver for MATLAB (version 2013b) [26] and evaluated them on both flop and latch-based ISCAS89 benchmark circuits. All experiments were run on two Intel Xeon E5-2450 v2 CPUs with total 32 threads and 32GB of RAM. We compare the area reduction obtained through the four approaches using three different estimates of EDL overheads as described in Section IV. Section IX-A presents detailed information of our circuit examples and Sections IX-B through IX-D describe the experimental setup for all four approaches. The resulting area reductions are then described and compared in Section IX-E.

Table I: Circuit information of flop-based designs

Circuit	Circuit Size		P	Area		$\|NCE\|$	Error-rate
	$\|C\|$	$\|S\|$	(ns)	Orig.	Resilient (H/M/L)		
s1196	331	18	0.35	343	367 / 355 / 349	3	0.25%
s1238	415	18	0.5	267	283 / 275 / 271	2	0%
s1423	454	74	0.6	491	763 / 627 / 559	34	0%
s1488	318	6	0.4	248	296 / 272 / 260	6	4.35%
s5378	809	164	0.45	1029	1405 / 1217 / 1123	47	0.79%
s9234	530	132	0.5	789	1301 / 1045 / 917	64	0.52%
s13207	1562	460	0.5	2496	2712 / 2604 / 2550	27	0.18%
s15850	1935	448	0.8	2687	3303 / 2995 / 2841	77	3.31%
s35932	5443	1728	0.6	9582	12702 / 11142 / 10362	390	14.98%
s38417	5752	1490	0.7	8720	12328 / 10524 / 9622	451	0.04%
s38584	6684	1248	0.7	7975	9583 / 8779 / 8377	201	69.79%

Table II: Circuit information of latch-based designs

Circuit	Circuit Size			Area		$\|NCE\|$	Error-rate
	$\|C\|$	$\|ES\|$	$\|NES\|$	Orig.	Resilient (H/M/L)		
s1196	331	18	19	322	362 / 342 / 332	5	18.29%
s1238	415	41	18	264	448 / 356 / 310	23	19.44%
s1423	454	74	93	549	949 / 749 / 649	50	14.67%
s1488	318	6	23	235	283 / 259 / 247	6	23.64%
s5378	809	164	187	964	1468 / 1216 / 1090	63	97.79%
s9234	530	132	169	769	1297 / 1033 / 901	66	2.66%
s13207	1562	460	520	2334	2902 / 2618 / 2476	71	25.8%
s15850	1935	448	518	2646	3958 / 3302 / 2974	164	9.93%
s35932	5443	1728	689	8933	11237 / 10085 / 9509	288	87.49%
s38417	5752	1490	1662	8048	14872 / 11460 / 9754	853	93.39%
s38584	6684	1248	1318	7387	12787 / 10087 / 8737	675	99.97%

A. Circuit Information

For each benchmark circuit, we set the maximum delay through combinational logic, P, to achieve a reasonable number of initial near-critical end-points and we set W to $0.3P$, modeling a resiliency window that is 30% of the max delay between error detecting pipeline stages.

Table I shows the size of benchmark circuits that we use to evaluate flop-based resilient design techniques, the maximum delay used for synthesis, and how many end-points are near-critical after synthesis (NCE), i.e., whose static worst-case delay is larger than P - W. The table also shows the circuit area before and after adding resiliency, considering high, medium, and low EDL overheads. Lastly, it lists the error-rate of each circuit as the percentage of simulation cycles for which at least one end-point settles within W of the end of the clock period when simulating the circuit with random input patterns.

For latch-based resilient designs, we focus our experiments on two-phase latch-based designs in which every other pipeline stage is error-detecting, as proposed in [8]. The maximum delay P used for synthesis is the same as each flop-based design. Table II shows the size of the chosen benchmark circuits, including the number of combinational gates, latches in error detecting pipeline stages (ES), which are not retimed and will not time-borrow, and latches in non-error-detecting pipeline stages (NES), which are retimed and will time-borrow up to time W. It also identifies how many end-points are initially near-critical. Except for s1238, the number of ES is the same as number of flops in flop-based design. In s1238, we added an error-detecting stage of master latches to the primary outputs to fix violating timing constraints.

Table III: Near-critical end-point reductions for BF and NBF techniques. (T: # of endpoints with tightened constraints Act: # of endpoints that actually became not near-critical)

Circuit	Near-critical End-point Reduction							
	Flop-based Designs				Latch-based Designs			
	BF		NBF		BF		NBF	
	T	Act.	T	Act.	T	Act.	T	Act.
s1196	2	3	1	2	2	3	1	1
s1238	2	2	1	3	TO	TO	1	1
s1423	TO	TO	1	2	TO	TO	1	10
s1488	2	2	1	2	0	0	1	0
s5378	TO	TO	1	2	TO	TO	1	15
s9234	TO	TO	1	20	TO	TO	1	14
s13207	TO	TO	1	10	TO	TO	1	19
s15850	TO	TO	1	9	TO	TO	1	35
s35932	TO	TO	1	109	TO	TO	1	2
s38417	TO	TO	1	46	TO	TO	1	91
s38584	TO	TO	0	0	TO	TO	1	79

B. Brute Force and Naive Brute Force Approaches

In the brute-force (BF) method, we tightened each combination of near-critical end-points and report the best area reduction among all combinations; while in the naive brute-force (NBF) method, we tightened each near-critical end-point independently and report the best area reduction among all end-points. In particular, we tighten near-critical end-points by adding set_max_delay timing constraints on those paths/end-points to constrain them to be indeed non-near-critical after resynthesis with the commercial synthesis tool. However, since the synthesis tool does not understand EDL area overhead, it may actually slow down existing non-near-critical paths to optimize logic area and these paths might then require EDL and its associated overhead. Hence, we also force those non-critical paths to remain non-critical using additional set_max_delay constraints in the brute-force case. We did not force those non-critical paths to remain non-critical in naive brute-force method because we want to keep the same settings as in [8] to be faithful to the previous work.

Table III shows data for the combination identified by both the BF and NBF methods for both flop and latch-based design examples. For each identified combination, we report the # of end-points tightened (T) and the actual # of end-points made not near-critical by resynthesis (Act). The sometimes high difference between these two numbers suggests that there is a large number of shared logic paths between tightened and un-tightened end-points. The area ratios associated with these identified combinations are shown later in this section. Due to the high complexity of the brute-force approach, it completed only on the smallest three circuits. The larger circuits timed out, denoted TO, after 24 hours of wall clock time.

C. Geometric Program Based Iterative Algorithm Approach

Our geometric program based iterative algorithm (GPIA) mathematically determines which near-critical paths should be sped up to minimize area. As with the brute-force technique, we use set_max_delay timing constraints to both speed-up the identified near-critical end-points and force non-critical paths to remain non-critical.

To calculate a gate's upper and lower sizing bounds, we compare the current size to the minimum/maximum size of gates with the same functionality. For example, let the area of gate i in the gate-level netlist be A_i and assume all gates with same functionality have a minimum size with area (min_A_i) and maximum size with area (max_A_i). Then, LB_i and UB_i will be calculated in (20).

$$LB_i = \frac{min\_A_i}{A_i}; \;\; UB_i = \frac{max\_A_i}{A_i}; \tag{20}$$

To analyze the impact of different threshold settings for the GPIA, Table IV shows the area ratio from the integer program, and clocked run-time of five different threshold settings and the MIGP for the high EDL overhead case: A ($Hth = 0.1$, $Lth = 0.4$), B ($Hth = 0.05$, $Lth = 0.2$), C ($Hth = 0.4$, $Lth = 0.1$), D ($Hth = 0.2$, $Lth = 0.05$), E ($Hth = 0.2$, $Lth = 0.2$). The area ratios of the iterative algorithm are all similar to that of the mixed integer geometric program but faster by an average of 5 times. This suggests our iterative algorithm is an effective approach to solve the MIGP and is somewhat robust to the choice of thresholds. In some circuits, the area reduction of the iterative program is better than the integer program. This may be because of differences in logic synthesis optimizations other than gate sizing, such as restructuring and repeater insertion. Note that we use the $Hth = 0.05$ and $Lth = 0.2$ setting when we compare this approach to the other approaches below because it achieves reasonable runtime and similar results to the integer program.

Our GPIA technique only models gate-sizing based resynthesis; however, the synthesis tools are not constrained to use only gate-sizing, and can optimize the design using other techniques, such as restructuring and buffer insertion. Table V shows the area ratio obtained after resynthesis applying the assignment of EDLs E obtained from the GPIA algorithm, as well as the expected area ratio as measured by the change in the geometric program's cost function. It is quite interesting to note that the area ratios from GPIA closely match that predicted by its cost function. This suggests that, at least for larger circuits, our gate-sizing model is an effective predictive model for resynthesis. For smaller circuits, on the other hand, the model seems to be less accurate. This may be due to the fact that for such circuits the decision of whether or not a single end-point should be error-detecting can make a large difference in the area reduction.

D. Virtual Resynthesis Library Approach

For the virtual resynthesis cell library method, after regular synthesis, all end-points are mapped to regular sequential gates with the unmodified cell library, regardless if the end-point is near-critical or not. The first input netlist setting we explored (VLN) is where we do not change the input netlist before reading in the virtual library. Thus, all end-points will be mapped to nonEDL sequential gates before we re-compile the netlist. Because these nonEDL gates now have setup-times that mimic the TRW, timing at near-critical endpoints will be violated. Hence, during re-synthesis, the synthesis tool needs to perform timing optimization to fix such timing violations. The second input netlist setting (VLE) is where change all end-points to end with EDL. These EDL gates have higher area but do not have higher setup-times. The synthesis tool can now reduce area by smartly mapping some non-near-critical end-points back to nonEDL sequential gates while optimizing the combinational logic accordingly. The third input netlist

Table IV: Area ratio and run-time of the MIGP approach and the GPIA approach with different threshold settings for flop-based designs with high EDL overhead

Circuit	Area Ratio						Clock Run-time					
	A	B	C	D	E	MIGP	A	B	C	D	E	MIGP
s1196	0.89	0.89	0.89	0.89	0.89	0.89	29s	46s	43s	4.4m	1.2m	1m
s1238	0.92	0.92	0.92	0.92	0.92	0.92	49s	99s	49s	1.2m	1.6m	4m
s1423	0.77	0.77	0.68	0.68	0.68	0.68	41s	83s	1.4m	11.7m	6.6m	19m
s1488	0.98	1.00	1.03	0.96	0.96	0.92	52s	85s	1.1m	3.2m	10.2m	1.5m
s5378	0.77	0.77	0.75	0.75	0.75	0.75	2.6m	3.7m	3.4m	9.4m	7.4m	1hr
s9234	0.73	0.73	0.68	0.68	0.68	0.67	1.5m	3.6m	1.8m	6.6m	4.0m	14m
s13207	0.92	0.92	0.92	0.92	0.92	0.92	10m	12m	17.2m	26.4m	15.2m	2.8hr
s15850	0.83	0.83	0.85	0.85	0.85	0.83	15m	22m	24m	42.5m	46.9m	24hr
s35932	0.75	0.75	0.75	0.75	0.75	0.75	3hr	3hr	7hr	8.3hr	8.4hr	24hr
s38417	0.80	0.83	0.90	0.78	0.78	TO	3.2hr	5hr	10hr	TO	8.8hr	TO
s38584	0.84	0.84	0.84	0.84	0.84	TO	8.3hr	14.4hr	10.2hr	16.5hr	16.5hr	TO
average	0.84	0.84	0.84	0.82	0.82	0.81	1.4hr	2.1hr	2.5hr	2.6hr	3.2hr	7.4hr

Table V: Area ratio of iterative algorithm from resynthesis (GPIA) and geometric program (Exp.) for flop-based designs

Circuit	Area Ratio											
	Flop-based						Latch-based					
	High Overhead		Medium Overhead		Low Overhead		High Overhead		Medium Overhead		Low Overhead	
	GPIA	Exp.	GPIA	Exp.	GPIA	Exp.	GPIA	Exp.	GPIA	Exp.	GPIA	Exp.
s1196	0.89	0.77	0.91	0.81	0.92	0.81	0.92	0.80	0.92	0.80	0.93	0.80
s1238	0.92	0.84	0.95	0.86	0.96	0.87	0.99	0.91	1.02	0.94	1.04	0.95
s1423	0.77	0.72	0.89	0.83	0.97	0.91	0.84	0.84	0.88	0.89	0.91	0.92
s1488	1.00	0.86	1.03	0.88	1.04	0.89	1.00	1.00	1.00	1.00	1.00	1.00
s5378	0.77	0.75	0.87	0.85	0.95	0.93	0.80	0.92	0.90	0.95	0.97	0.97
s9234	0.73	0.72	0.85	0.87	0.94	0.93	0.75	0.78	0.91	0.87	1.06	0.93
s13207	0.92	0.90	0.95	0.94	0.97	0.96	0.84	0.78	0.91	0.86	0.97	0.91
s15850	0.83	0.81	0.91	0.89	0.96	0.94	0.78	0.76	0.88	0.84	0.95	0.90
s35932	0.75	0.75	0.86	0.86	0.92	0.93	0.85	0.87	0.92	0.93	0.95	0.96
s38417	0.83	0.81	0.91	0.90	0.97	0.94	0.82	0.84	0.89	0.90	0.94	0.94
s38584	0.84	0.84	0.91	0.91	0.95	0.95	0.65	0.66	0.79	0.79	0.89	0.89
average	0.84	0.80	0.91	0.87	0.96	0.91	0.84	0.83	0.91	0.89	0.97	0.92

setting (VLR) is where we map the sequential gates to either nonEDLs and EDLs based on their timing. If the end-point is near-critical, the sequential gate will be mapped to EDLs; otherwise, it stays as a nonEDL.

We applied resynthesis to the three different input netlist settings for ISCAS89 benchmark suite. However, some latch-based designs have timing violations on non-near-critical latches after resynthesis with the VLN setting, i.e., circuits s1423, s5378, s9234, s13207, and s38417. To fix these violations, we manually change the violating latches (with larger setup time) to EDL latches (with larger area) and adjust the area overhead accordingly.

Table VI and VII show the experimental results for flop and latch-based designs, respectively. Examining the average area ratios across the benchmark suite, we see that the VLN setting with both flop and latch-based design leads to better results than the other settings in both the high and medium overhead cases. The difference in the low overhead case is relatively small. This may be explained by the observation that the commercial synthesis tool seems to do a better job at timing optimization compared to area optimization. Based on these results, for comparison of the virtual resynthesis library approach to the other three resynthesis approaches, we will use the VLN setting. Lastly, we explored the benefits of running a second resynthesis run on the VLN circuits with manually fixed timing violations. The average area ratio reduced by approximately 0.01 on all overheads (detailed results not shown).

Table VI: Virtual resynthesis cell library flop-based experimental results with different input netlist settings

Circuit	Area Ratio								
	High Overhead			Medium Overhead			Low Overhead		
	VLN	VLE	VLR	VLN	VLE	VLR	VLN	VLE	VLR
s1196	0.99	0.95	0.96	0.96	0.93	0.94	0.96	0.90	0.92
s1238	0.96	0.95	0.97	0.95	0.96	0.96	0.97	0.95	0.99
s1423	0.87	0.98	0.91	0.94	0.97	0.95	0.98	0.97	0.97
s1488	1.07	0.96	0.96	1.02	0.97	0.97	0.98	0.95	0.95
s5378	0.89	1.01	0.98	0.94	1.00	0.98	0.97	0.99	0.99
s9234	0.75	0.94	0.95	0.87	0.96	0.96	0.95	0.98	0.98
s13207	0.91	1.05	0.94	0.93	0.99	0.95	0.94	0.97	0.95
s15850	0.90	1.00	0.96	0.95	1.00	0.97	0.98	0.99	0.97
s35932	1.00	0.98	0.99	1.01	1.00	1.00	1.02	1.02	1.01
s38417	0.87	1.03	0.98	0.93	1.01	0.98	0.97	0.99	0.98
s38584	0.83	0.92	0.94	0.91	0.96	0.96	0.95	0.99	0.97
average	0.91	0.98	0.96	0.95	0.98	0.97	0.97	0.97	0.97

Table VII: Virtual resynthesis cell library latch-based experimental results with different input netlist settings

Circuit	Area Ratio								
	High Overhead			Medium Overhead			Low Overhead		
	VLN	VLE	VLR	VLN	VLE	VLR	VLN	VLE	VLR
s1196	0.88	1.08	0.88	0.93	0.99	0.88	0.95	0.91	0.90
s1238	1.07	1.28	1.00	1.09	1.18	1.01	1.07	1.10	1.02
s1423	0.81	1.05	0.97	0.89	0.98	0.94	1.03	0.94	0.94
s1488	1.42	1.00	1.00	1.54	1.00	1.00	1.63	1.00	1.00
s5378	0.79	1.12	1.00	0.90	1.07	1.00	0.98	1.04	1.00
s9234	0.93	1.02	1.00	0.97	1.02	1.01	0.99	1.02	1.02
s13207	0.90	1.52	1.04	0.97	1.28	1.00	1.01	1.14	1.02
s15850	0.75	1.39	1.01	0.88	1.23	1.01	0.96	1.13	1.01
s35932	0.78	1.18	0.99	0.87	1.09	0.98	0.92	1.04	0.98
s38417	0.76	1.12	1.00	0.91	1.08	1.00	1.01	1.05	1.05
s38584	0.62	1.20	1.00	0.77	1.13	1.00	0.90	1.07	1.01
average	0.88	1.18	0.99	0.97	1.10	0.99	1.04	1.04	0.99

E. Comparison of Resynthesis Approaches

In this section, we compare the area ratios, run-time, and change of error-rate on both flop and latch-based designs across the four different proposed resynthesis approaches. For the naive brute-force method, if all runs give area ratio more than 1, we will report the area ratio as 1, which means we choose the design obtained before resynthesis. For the GPIA approach, if the near-critical end-point list for resynthesis is different than obtained before resynthesis, we will run resynthesis and report the area ratio even if it is over 1. If the expected end-point list is the same as before resynthesis, we will not do resynthesis and report the area ratio as 1. For the VL approach, resynthesis will be done first and if the end-point list after resynthesis is different than before resynthesis, we report the area ratio (even if it is over 1). In addition, if in the VL approach the expected end-point list is the same as before resynthesis, we will not do resynthesis and report the area ratio as 1.

In the following, we first report the results for the flop-based and, then, for the latch-based designs.

1) Flop-based Designs: Table VIII shows the achieved area ratios from our geometric-programming-based iterative (GPIA) method, the two brute force methods, and the virtual resynthesis cell library method. The logic synthesis tool reports the logic area and how many paths are near-critical based on value of W. We then calculate the resulting area by summing up logic and total EDL area overhead and report the area ratios. For the virtual resynthesis cell library method, since we already added EDL area overhead into cell library, the logic area from synthesis report already includes the EDL area. For the naive brute-force approach, we speed up near-critical paths one at a time and we only show the best area reduction among all of them, as described in [8]; while for the brute-force method, we speed up each combination of near-critical path at a time and report the best area reduction among all combinations. For those circuits for which it completes, the brute-force method produces the best area reductions among the four methods.

Comparing across all four methods, the GPIA achieves the largest area reduction next to the brute-force approach and completes on all examples. In fact, when we analyzed the examples where the GPIA failed to find the result identified by the BF approach, we found that in two of the three cases (s1196 and s1238) the GPIA approach identified and optimized the second best combination of end-points. For the third circuit (s1488), we know that the GPIA failed to find the same results as the brute-force because the synthesis tool actually reduced area even when we added tighter timing constraints, which is a behavior inconsistent with our GPIA model. This suggests that the GPIA's simplified model of the shared combinational logic between end-points is sufficient to find close to the best combination of end-points as the brute-force method to re-constrain.

When we compare results between the GPIA and naive brute-force approaches, the GPIA approach achieves larger area reduction in most cases. On average, our iterative algorithm achieves 9% larger area reduction than naive brute-force with high EDL area overhead, 4% larger improvement with medium EDL area overhead, and 1% larger improvement with low EDL area overhead. Finally, we compare the results between the GPIA and virtual cell library (VL) methods. The results in Table VIII, suggest that despite having access to library models that effectively model the impact of resynthesis choices, the commercial synthesis tool does not seem to perform as well as the GPIA approach. For the high and medium EDL area overhead cases, the virtual cell library method on average achieved only 56% of the area reduction of the GPIA approach. For the low EDL area overhead cases, it achieved 75% of the area reduction of the GPIA algorithm, respectively. Across all EDL area overheads, the virtual cell library method achieves approximately 2/3rds of the area reduction of the GPIA approach.

The iterative geometric program, naive brute-force and brute-force approaches use multi-threaded computation. So, Table IX reports both clock and CPU run-times. Each entry in the table is the maximum run-time observed among the three EDL overheads tested. The number of threads represents how many resynthesis runs execute in parallel during the naive brute-force and brute-force approaches. Hence, in these cases the clock run-time and the number of threads are inversely proportional. For the iterative geometric program, the mathematical solver is also implemented with multi-threading; however, the number of threads is not programmable. Although our iterative geometric program is slower than naive brute-force, the run-times are still reasonable. Because the virtual resynthesis library only requires one synthesis run, it is an average of 135 times faster than the iterative algorithm. Thus, it is a run-time efficient means of optimizing resilient designs.

It is also important to note that speeding-up near critical paths may not only improve area, but also improve performance by reducing the error-rate. Although our methods only target minimizing area, we observe, on average, a drop in error-rates, as shown in Table X. For the GPIA approach, except for s1488, the circuits' error rate are lowered and for the VL approach, 9 out of 11 circuits have lower error rates. The new error rates of many designs after GPIA resynthesis become 0. Note that there are two reasons that error rate can be 0. First, all near-critical end-points can become non-near-critical through resynthesis. This occurs in circuits s1196, s5378 and s35932. Second, near-critical end-points remain but the data path does not trigger errors in our simulations. This occurs in circuit s13207. There are three cases (s1488 in VL with low overhead, s35932 in NBF with all overheads, and s38417 in GPIA with low overhead for which the new error rates increase to more than 5%). For resilient schemes with high error penalties, this may be unacceptable. One way to address this is to adjust the TRW from 30% to 20% without changing their worst-case delay. The error rate of s1488 drops to 4% and the error rate of s35932 reduces to 20%. For s38417, reducing the TRW does not decrease error rate; however, we found all errors are caused by one particular flip-flop. Interestingly, if we manually constrain that flip-flop to be non-near-critical and resynthesize, the error rate drops to 0.04% and the area ratio changes from 0.97 to 0.96. The various changes of error rates are interesting because it motivates future work exploring adding a notion of error-rate into the cost function to simultaneously target area and performance. In fact, more generally, it would be interesting to minimize power consumption for a given

Table VIII: Area ratios comparison for flop-based designs: (GPIA: Iterative algorithm, NBF: Naive Brute-force, BF: Brute-force, VL: Virtual Library)

Circuit	Area Ratio											
	High Overhead				Medium Overhead				Low Overhead			
	GPIA	NBF	BF	VL	GPIA	NBF	BF	VL	GPIA	NBF	BF	VL
s1196	0.89	0.92	0.88	0.99	0.91	0.94	0.90	0.96	0.92	0.95	0.92	0.96
s1238	0.92	0.91	0.88	0.96	0.95	0.94	0.91	0.95	0.96	0.95	0.92	0.97
s1423	0.77	0.83	TO	0.87	0.89	0.91	TO	0.94	0.97	0.95	TO	0.98
s1488	1.00	0.95	0.92	1.07	1.03	0.96	0.97	1.02	1.04	0.97	0.97	0.98
s5378	0.77	0.92	TO	0.89	0.87	0.95	TO	0.94	0.95	0.98	TO	0.97
s9234	0.73	0.86	TO	0.75	0.85	0.92	TO	0.87	0.94	0.95	TO	0.95
s13207	0.92	0.95	TO	0.91	0.95	0.95	TO	0.93	0.97	0.95	TO	0.94
s15850	0.83	0.96	TO	0.90	0.91	0.98	TO	0.95	0.96	0.99	TO	0.98
s35932	0.75	0.93	TO	1.00	0.86	0.96	TO	1.01	0.92	0.98	TO	1.02
s38417	0.83	0.97	TO	0.87	0.91	0.98	TO	0.93	0.97	0.99	TO	0.97
s38584	0.84	1.00	TO	0.83	0.91	1.00	TO	0.91	0.95	1.00	TO	0.95
average	0.84	0.93	0.89	0.91	0.91	0.95	0.93	0.95	0.96	0.97	0.94	0.97

Table IX: Run-time comparison for flop-based designs

Circuit	Run-time						
	GPIA		NBF		BF		VL
	CLOCK	CPU	CLOCK	CPU	CLOCK	CPU	CLOCK
s1196	1.3m	11.6m	1m	2.5m	1m	4m	0.5m
s1238	2.1m	19m	0.5m	2m	0.5m	2m	0.5m
s1423	1.7m	14m	3.5m	28m	TO	TO	0.5m
s1488	1.7m	14m	1.5m	5m	2m	15m	0.5m
s5378	7m	1.3hr	5m	39m	TO	TO	0.5m
s9234	2.2m	17.3m	6m	53m	TO	TO	0.5m
s13207	11.5m	1.77hr	3m	25m	TO	TO	0.7m
s15850	18.5m	2.78hr	8m	64m	TO	TO	0.7m
s35932	2.7hr	22.19hr	45m	5hr	TO	TO	1.5m
s38417	4.8hr	32hr	1hr	6hr	TO	TO	1.6m
s38584	12.3hr	57.7hr	32m	2.7hr	TO	TO	1.4m

Table X: Error-rates (%) of flop-based designs

Circuit	Old Error-rate	Overhead	Error-rate							
			GPIA		NBF		BF		VL	
			Change	New	Change	New	Change	New	Change	New
s1196	0.25	H	-0.25	0	-0.08	0.17	-0.25	0	-0.16	0.09
		M	-0.25	0	-0.08	0.17	-0.25	0	0.08	0.33
		L	-0.25	0	-0.08	0.17	-0.25	0	-0.25	0
s1238	0	H/M/L	0	0	0	0	0	0	0	0
s1423	0	H/M/L	0	0	0	0	0	0	0	0
s1488	4.35	H	2.98	7.33	0.56	4.91	-4.28	0.07	3.15	7.5
		M	2.98	7.33	0.56	4.91	3.5	7.85	5.7	10.05
		L	2.98	7.33	0.56	4.91	3.5	7.85	7.32	11.67
s5378	0.79	H	-0.79	0	-0.29	0.5	TO	TO	-0.34	0.45
		M	-0.79	0	-0.29	0.5	TO	TO	-0.26	0.53
		L	-0.79	0	-0.29	0.5	TO	TO	-0.08	0.71
s9234	0.52	H/M/L	-0.52	0	-0.37	0.15	TO	TO	-0.38	0.14
s13207	0.18	H	-0.18	0	0	0.18	TO	TO	-0.18	0
		M	-0.18	0	0	0.18	TO	TO	-0.15	0.03
		L	-0.17	0.01	0	0.18	TO	TO	-0.15	0.03
s15850	3.31	H	-1.03	2.28	0.4	3.71	TO	TO	-1.64	1.67
		M/L	-1.67	1.64	0.4	3.71	TO	TO	-1.65	1.66
s35932	14.98	H/M/L	-14.98	0	59.65	74.63	TO	TO	-14.98	0
s38417	0.04	H	-0.04	0	0.81	0.85	TO	TO	0.07	0.11
		M	-0.03	0.01	0.2	0.24	TO	TO	0.6	0.64
		L	28.32	28.36	0.2	0.24	TO	TO	0.47	0.51
s38584	69.79	H	-10.88	58.91	0	69.79	TO	TO	-8.73	61.06
		M	-14.67	55.12	0	69.79	TO	TO	-7.2	62.59
		L	-7.63	62.16	0	69.79	TO	TO	-4.15	65.64
average	8.56		-1.53	7.03	5.48	14.04	0.22	1.75	-1.68	6.89

performance considering voltage scaling.

2) Latch-based design: Table XI shows achieved area reductions from the four proposed approaches on our latch-based resilient designs. We see similar trends as in the flop-based designs. In particular, the GPIA algorithm achieves larger improvements than both the naive brute-force and virtual library method. With high EDL overhead, GPIA has 11% larger area reduction than NBF and 4% larger than VL; with medium EDL overhead, GPIA has 6% larger area reduction than NBF and VL; with low EDL overhead, GPIA is 1% better NBF but 7% better than VL. The area reductions obtained by the GPIA approach for circuit s1196 are far away from the best result (achieved by brute-force). But, as in the flip-flop results, this may be explained by the fact that, after brute-force resynthesis the regular logic area unexpectedly became smaller than before resynthesis, even though we tightened the timing constraints to reduce the number of EDLs. In other words, the logic synthesis engine during brute force resynthesis of this example unexpectedly found a more area efficient solution that was also faster. For s1488, the new near-critical end-points list remains the same as before resynthesis through GPIA and NBF. For VL, some near-critical end-points are removed but the area ratios go up rapidly.

The run-time comparison of all approaches on latch-based designs is reported in Table XII. As in the flop-based results, the virtual resynthesis cell library method is by far the fastest because it only requires one synthesis run. For the GPIA algorithm, even though we only add a relatively few additional constraints compared to the flop-based design, the average run-time of the latch-based designs is doubled compared to the flop-based designs. If we want to run larger circuits than s35854, we may need to either choose different threshold settings or use the virtual library method.

When we optimize the area of flop-based design using the GPIA method, most of the error-rates went down to less than 10% (recall Table X). For the latch-based design using the GPIA approach, the error-rate reduction follows the same trend as the area reduction and on average it reduces the error rate by 20.79%; however, some error-rates remain over 20%, as seen in Table XIII. The error recovery penalty of flop-based resilient templates (often over 10 clock cycles [6]) is much larger than latch-based templates which are often one clock cycle or less [7], [28]. Consequently, for optimal operation, the error rate for latch-based resilient designs can be as high as 20%, as reported in [28], and the desired error rate may often be achieved by simply removing specific EDLs.

X. Conclusions

We presented four different approaches to minimize the area of timing resilient design through resynthesis. The first is a brute-force check-all-combinations approach but is so complex that can only be used on very low complex circuits. The second is a naive brute-force approach that has lower time complexity, because it only considers one near-critical end-point at a time, at the cost of losing any benefit of optimizing multiple end-points simultaneously. The third is

Table XI: Area ratio of latch-based designs: (GPIA: Iterative algorithm, NBF: Naive Brute-force, BF: Brute-force, VL: Virtual resynthesis cell library)

Circuit	Area Ratio											
	High Overhead				Medium Overhead				Low Overhead			
	GPIA	NBF	BF	VL	GPIA	NBF	BF	VL	GPIA	NBF	BF	VL
s1196	0.92	0.93	0.84	0.88	0.92	0.95	0.86	0.93	0.93	0.96	0.87	0.95
s1238	0.99	0.97	TO	1.07	1.02	0.97	TO	1.09	1.04	0.98	TO	1.07
s1423	0.84	0.91	TO	0.81	0.88	0.93	TO	0.89	0.91	0.94	TO	1.03
s1488	1.00	1.00	1.00	1.42	1.00	1.00	1.00	1.54	1.00	1.00	1.00	1.63
s5378	0.80	0.94	TO	0.79	0.90	0.97	TO	0.90	0.97	0.98	TO	0.98
s9234	0.75	0.94	TO	0.93	0.91	0.98	TO	0.97	1.06	0.99	TO	0.99
s13207	0.84	0.98	TO	0.90	0.91	0.99	TO	0.97	0.97	0.99	TO	1.01
s15850	0.78	0.93	TO	0.75	0.88	0.96	TO	0.88	0.95	0.97	TO	0.96
s35932	0.85	1.00	TO	0.78	0.92	1.00	TO	0.87	0.95	1.00	TO	0.92
s38417	0.82	0.95	TO	0.76	0.89	0.97	TO	0.91	0.94	0.98	TO	1.01
s38584	0.65	0.95	TO	0.62	0.79	0.97	TO	0.77	0.89	0.98	TO	0.90
average	0.84	0.95	0.92	0.88	0.91	0.97	0.93	0.97	0.97	0.98	0.94	1.04

Table XII: Run-time comparisons for latch-based designs

Circuit	Run-time						
	GPIA		NBF		BF		VL
	CLOCK	CPU	CLOCK	CPU	CLOCK	CPU	CLOCK
s1196	1.5m	6.6m	1m	3m	3.6m	33.6m	24.18s
s1238	1.3m	6.5m	3m	20m	TO	TO	23.73s
s1423	4.4m	25.48m	5m	35m	TO	TO	29.51s
s1488	1.13m	4.98m	1.5m	5m	6m	1hr	24.52s
s5378	10.8m	50.77m	7m	60m	TO	TO	27.89s
s9234	5.53m	32.93m	6m	53m	TO	TO	26.88s
s13207	41.5m	4.2hr	7m	75m	TO	TO	47.44s
s15850	24.7m	2.4hr	8m	2hr	TO	TO	45.47s
s35932	15hr	64hr	30m	3hr	TO	TO	3.2m
s38417	17hr	64hr	3hr	9hr	TO	TO	65.83s
s38584	24hr	120hr	2hr	7hr	TO	TO	69.78s

Table XIII: Error-rates (%) of latch-based designs

Circuit	Old Error-rate	Overhead	Error-rate							
			GPIA		NBF		BF		VL	
			Change	New	Change	New	Change	New	Change	New
s1196	18.29	H	0	18.29	-7.48	10.81	-9.99	8.3	10.22	28.51
		M/L	-1.4	16.89	-7.48	10.81	-5.21	13.08	10.22	28.51
s1238	19.44	H	-8.91	10.53	-0.02	19.42	TO	TO	-7.46	11.98
		M/L	-8.91	10.53	-0.02	19.42	TO	TO	-6.69	12.75
s1423	14.67	H	-14.67	0	-14.53	0.14	TO	TO	-14.67	0
		M/L	-14.67	0	-6.19	8.48	TO	TO	-14.67	0
s1488	23.64	H	0	23.64	0	23.64	0	23.64	-23.1	0.54
		M	0	23.64	0	23.64	0	23.64	-23.14	0.5
		L	0	23.64	0	23.64	0	23.64	-23.13	0.51
s5378	97.79	H	-6.88	90.91	-97.78	0.01	TO	TO	1.13	98.92
		M	-6.88	90.91	-97.78	0.01	TO	TO	1.16	98.95
		L	-6.88	90.91	-97.78	0.01	TO	TO	1.11	98.9
s9234	2.66	H	-0.09	1.67	-2.65	0.01	TO	TO	0.05	2.71
		M	-1.12	1.54	-2.65	0.01	TO	TO	-0.02	2.64
		L	-1.12	1.54	-2.65	0.01	TO	TO	0.05	2.71
s13207	25.8	H	-25.78	0.02	-25.79	0.01	TO	TO	-25.78	0.02
		M	-25.78	0.02	-25.79	0.01	TO	TO	-3.45	22.35
		L	-25.76	0.04	-25.79	0.01	TO	TO	-25.78	0.02
s15850	9.93	H	-5.58	4.35	2.29	12.22	TO	TO	7.24	17.17
		M	-6.42	3.51	2.29	12.22	TO	TO	-6.82	3.11
		L	2.03	11.96	2.29	12.22	TO	TO	6.95	16.88
s35932	87.49	H/M/L	-15.02	72.47	0.02	87.51	TO	TO	-11.42	76.07
s38417	93.39	H	-59.52	33.87	-3.46	89.93	TO	TO	-93.19	0.2
		M	-60.66	32.73	-3.46	89.93	TO	TO	-93.23	0.16
		L	-86.63	6.76	-3.46	89.93	TO	TO	-93.26	0.13
s38584	99.97	H	-91.25	8.72	-28.82	71.15	TO	TO	-49.48	50.49
		M	-75.83	24.14	-28.82	71.15	TO	TO	-50.58	49.39
		L	-82.46	17.51	-28.82	71.15	TO	TO	-49.53	50.44
average	44.82		-20.79	24.03	-15.70	29.13	-3.40	17.56	-18.52	26.31

a geometric program based approach that uses a gate-sizing based delay model and iterative relaxation algorithm (GPIA). This approach captures both logic and EDL area overhead and has an implicit model of all the shared logic paths. The last approach is a virtual resynthesis cell library method which has the lowest time complexity among the four approaches. It involves manipulating the synthesis cell library to duplicate sequential cells such that one group increases the setup time to match the TRW window and the other group increases area to match the expected EDL area overhead.

Our experimental results show that our GPIA approach achieves area reductions that compare favorably across these four approaches. The GPIA algorithm completes within 24 hours for all examples and the average area reduction is up to 16%. The results suggest that a gate-sizing based model can be used to effectively guide resynthesis. However, for the larger circuits, the run-time of the GPIA based approach is still high. As an alternative, the proposed virtual resynthesis cell library approach obtains an average of approximately 2/3rds of the area reductions of the mathematical program based approach with fast run times associated with only a single synthesis run.

We must emphasize that resilient designs can exhibit high average performance only when few or no timing error happens. Currently, our proposed approaches only focus on reducing total area; however, these optimizations cannot guarantee the error-rate also reduces. Towards this goal, it may be possible to model an approximate measure of error-rate and/or performance in our GPIA objective function. As we mention in Section V, modeling all sets of error-detecting sequential gates within the mathematical program is unrealistic. An interesting area of future work is to develop heuristic algorithms for error-rate reduction based on a modified GPIA that targets the reduction of specific instances of errors. Ultimately, however, we may wish to further extend our GPIA model to minimize perhaps a more critical objective function, the energy consumption of the resilient system with and without voltage scaling. Finally, because the length of individual paths can greatly vary between a logic-level netlist and the final place-and-routed design, it is important to incorporate these methods within a comprehensive place-and-route flow for resilient designs.

REFERENCES

[1] K. Bowman, J. Tschanz, N. S. Kim, J. Lee, C. Wilkerson, S. Lu, T. Karnik, and V. De, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE JSCC*, vol. 44, no. 1, pp. 49–63, Jan 2009.

[2] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb 2010.

[3] Y. Kunitake, T. Sato, H. Yasuura, and T. Hayashida, "Possibilities to miss predicting timing errors in canary flip-flops," in *MWSCAS*, Aug 2011, pp. 1–4.

[4] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De, "Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance," in *VLSI Circuits*, June 2009, pp. 112–113.

[5] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, "Timber: Time borrowing and error relaying for online timing error resilience," in *DATE*, March 2010, pp. 1554–1559.

[6] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "Razor II: In situ error detection and correction for PVT and SER tolerance," *IEEE JSCC*, vol. 44, no. 1, pp. 32–48, Jan 2009.

[7] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester, "Bubble razor: Eliminating timing margins in an ARM cortex-M3 processor in 45 nm CMOS using architecturally independent error detection and correction," *IEEE JSCC*, vol. 48, no. 1, pp. 66–81, Jan 2013.

[8] D. Hand, M. Trevisan Moreira, H.-H. Huang, D. Chen, F. Butzke, Z. Li, M. Gibiluka, M. Breuer, N. Calazans, and P. Beerel, "Blade – a timing violation resilient asynchronous template," in *ASYNC*, May 2015, pp. 21–28.

[9] A. B. Kahng, S. Kang, J. Li, and J. Pineda De Gyvez, "An improved methodology for resilient design implementation," *TODAES*, vol. 20, no. 4, p. 66, 2015.

[10] H.-H. Huang, H. Cheng, C. Chu, and P. A. Beerel, "Area optimization of resilient designs guided by a mixed integer geometric program," in *DAC*, 2016, p. 130.

[11] Y. Liu, R. Ye, F. Yuan, R. Kumar, and Q. Xu, "On logic synthesis for timing speculation," in *ICCAD*. IEEE, 2012, pp. 591–596.

[12] M. Nakai, S. Akui, K. Seno, T. Meguro, T. Seki, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumano, and M. Shimura, "Dynamic voltage and frequency management for a low-power embedded microprocessor," *IEEE JSCC*, vol. 40, no. 1, pp. 28–35, 2005.

[13] R. Ye, F. Yuan, H. Zhou, and Q. Xu, "Clock skew scheduling for timing speculation," in *DATE*. IEEE, 2012, pp. 929–934.

[14] R. Ye, F. Yuan, and Q. Xu, "Online clock skew tuning for timing speculation," in *ICCAD*, 2011, pp. 442–447.

[15] B. Greskamp, L. Wan, U. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen, and C. Zilles, "Blueshift: Designing processors for timing speculation from the ground up." in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 2009, pp. 213–224.

[16] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Recovery-driven design: a power minimization methodology for error-tolerant processor modules," in *DAC*, 2010, pp. 825–830.

[17] ——, "Slack redistribution for graceful degradation under voltage overscaling," in *ASP-DAC*, 2010, pp. 825–831.

[18] M. R. Choudhury and K. Mohanram, "Masking timing errors on speed-paths in logic circuits," in *DATE*, 2009, pp. 87–92.

[19] M. Cannizzaro, S. Beer, J. Cortadella, R. Ginosar, and L. Lavagno, "SafeRazor: Metastability-robust adaptive clocking in resilient circuits," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 62, no. 9, pp. 2238–2247, 2015.

[20] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. IEEE, 2003, pp. 7–18.

[21] 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," in *ISSCC*. IEEE, 2013, pp. 264–265.

[22] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester, "Bubble razor: An architecture-independent approach to timing-error detection and correction," in *ISSCC*, 2012, pp. 488–490.

[23] W. Hua, R. N. Tadros, and P. A. Beerel, "Low area, low power, robust, highly sensitive error detecting latch for resilient architectures," in *ISPLED*, 2016, pp. 16–21.

[24] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *MICRO-36*, Dec 2003, pp. 7–18.

[25] "ISCAS89: International symposium on circuits and systems sequential benchmark. http://www.pld.ttu.ee/˜maksim/benchmarks/iscas89/verilog/."

[26] "YALMIP: modelling language for advanced modeling and solution of convex and nonconvex optimization problems. http://users.isy.liu.se/johanl/yalmip/."

[27] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and engineering*, vol. 8, no. 1, pp. 67–127, 2007.

[28] D. Hand, H.-H. Huang, B. Cheng, Y. Zhang, M. Moreira, M. Breuer, N. Calazans, and P. Beerel, "Performance optimization and analysis of blade designs under delay variability," in *ASYNC*, May 2015, pp. 61–68.

Hsin-Ho Huang received her B.S. and M.S in Computer Science from National Tsing-Hua University in Taiwan and her Ph.D. from University of Southern California, Los Angeles, CA under the supervision of Professor Peter A. Beerel. Her main research interests are in the area of computer aided design, built-in optimization of area and power of asynchronous VLSI designs. She is a student member of the IEEE.

Huimei Cheng Huimei Cheng is a Ph.D. student in Ming Hsieh Department of Electrical Engineering at the University of Southern California. She received her B.S. degree at Nanjing University of Information & Technology (China) in 2014, and her M.S. degree from USC in 2016. Upon graduation, she worked at Synopsys in R&D Prime Time team conducting research on pessimism reduction in crosstalk. She is a student member of IEEE. Her research interests include a variety of topics in CAD and asynchronous VLSI design.

Chris Chu Chris Chu received the B.S. degree in computer science from the University of Hong Kong, Hong Kong, in 1993. He received the M.S. degree and the Ph.D. degree in computer science from the University of Texas at Austin in 1994 and 1999, respectively. Dr. Chu is a Professor in the Electrical and Computer Engineering Department at Iowa State University. His area of expertises include CAD of VLSI physical design, and design and analysis of algorithms. Dr. Chu is currently an associate editor for IEEE TCAD. He has served on the technical program committees of several major conferences including DAC, ICCAD, ISPD, ISCAS, DATE, ASP-DAC, and SLIP. Dr. Chu received the IEEE TCAD best paper award at 1999 for his work in performance-driven interconnect optimization. He received another IEEE TCAD best paper award at 2010 for his work in routing tree construction. He received the ISPD best paper award at 2004 for his work in efficient placement algorithm. He received another ISPD best paper award at 2012 for his work in floorplan block shaping algorithm. He received the ASPDAC best paper award at 2014 for his work in stencil design for electron-beam lithography. He received the Bert Kay Best Dissertation Award for 1998-1999 from the Department of Computer Sciences in the University of Texas at Austin. He is a Fellow of IEEE.

Peter A. Beerel received his B.S.E. degree in Electrical Engineering from Princeton University, Princeton, NJ, in 1989 and his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 1991 and 1994, respectively. Professor Beerel is currently an Associate Professor and the Faculty Director of Innovation and Entrepreneurship in Engineering at the University of Southern California as well as the Chief Scientist at Reduced Energy Microsystems. He co-founded TimeLess Design Automation to commercialize an asynchronous ASIC flow in 2008 and sold the company in 2010 to Fulcrum Microsystems which was bought by Intel in 2011. His interests include a variety of topics in computer-aided design, resilient design, and asynchronous VLSI and the commercialization of these technologies. He is a Senior Member of the IEEE.