Area Optimization of Resilient Designs Guided by a Mixed Integer Geometric Program

Hsin-Ho Huang[1] hsinhohu@usc.edu
Huimei Cheng[1] huimeich@usc.edu
Chris Chu[2] cnchu@iastate.edu
Peter A. Beerel[1] pabeerel@usc.edu

[1]Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA

[2]Department of Electrical and Computer Engineering
Iowa State University
Ames, IA

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 a gate-sizing based mixed integer geometric programming framework to analytically model and optimize paths during resynthesis. We evaluate our approach on a set of ISCAS89 benchmarks and compare the overall area improvement after resynthesis guided by our mathematical model versus a previously published naive brute-force approach. Our experimental results demonstrate that our approach achieves up to 11% larger average area improvement.

1. 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 [5]. 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., [10, 15]). 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 [3, 4, 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 higher average-case performance than traditional worst-case designs. This additional performance can then be traded off for lower power via further voltage scaling. Some EDA techniques have been proposed to minimize the probability of timing errors: [9, 11, 14, 17]. Liu et al. [11] proposed to reshape the delay distribution of near-critical paths to reduce timing errors. Dynatune [14] optimizes throughput by selectively choosing low V_{th} gates to speed up near-critical paths. Ye et al. [17] and Kahng et al. [9] both use clock skew scheduling to reduce timing errors.

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. Previous work proposed minimizing the EDL via *re-timing [12]*, relocating sequential gates to reduce the amount of EDL without changing the combinational logic. Others proposed *resynthesis [8, 9]*, 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 logic area. The basic challenge in resynthesis is that many paths share logic and it is not obvious which combination of paths should be sped up to achieve the best gains. As part of the Blade [8] resilient flow, a naive brute-force resynthesize method was explored. Near-critical paths were sped up one end-point at a time to find a suitable single candidate end-point to speed up 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 design with minimum cost considering the overhead of the error-detecting logic. 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.

During resynthesis many types of optimizations may be explored including gate sizing, logic restructuring, and repeater insertion. But we hypothesize that it is sufficient to predict the impact of logic synthesis by considering gate sizing alone. To evaluate this hypothesis, this paper proposes a gate sizing based mixed integer geometric programming (MIGP) framework which models both the normal combi-

DAC '16, June 05 - 09, 2016, Austin, TX, USA

ISBN 978-1-4503-4236-0/16/06...$15.00

DOI: http://dx.doi.org/10.1145/2897937.2897990

national/sequential logic area as well as the required EDL area overhead. Moreover, it understands shared logic across paths and thus can more accurately guide which particular near-critical-paths to resynthesize compared to the naive brute force method. In our MIGP, we adopt an Elmore-delay continuous sizing model to predict the logic synthesis impact and introduce binary variables that determine if the path must be terminated by an error detecting sequential gate (i.e., a latch or flip-flop) or not. However, integer variables in mathematical optimization are computationally expensive; therefore, we also propose a heuristic relaxation-based algorithm to iteratively solve the MIGP.

The benefits of our approach were evaluated using the ISCAS89 benchmark circuits. The results show that our MIGP approach achieves up to 11% larger average area improvement over the naive brute-force approach. Moreover, our relaxation-based iterative algorithm achieves close to the optimal integer results at a fraction of the run-time and memory usage and thereby allows completion of much larger benchmarks than otherwise possible. Moreover, as a side benefit of minimizing the error detecting logic, we also reduce the error-rate by, in our experiments, an average of 6.8% over that achieved by the brute-force approach and thus improve average-case performance.

The remainder of this paper is organized as follows. Section II introduces the notation and terminology in our proposed model. Section III formulates our gate sizing mixed integer geometric program and describes our relaxation-based iterative method for efficiently solving this program. Section IV shows our experimental results and comparisons on ISCAS89 benchmark circuits. Section V concludes our work and discusses potential future improvements.

2. PRELIMINARIES

This section explains the notation and terminology used in our resynthesis approach. We are given a gate-level VLSI circuits with combinational gates (C) and sequential gates (S), 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 fanin gate (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:

$$D_{i_k} = \mu * \frac{R_{i_k}}{z_i} * \sum_{j_l \in FO(i)} Cin_{j_l} * z_j \tag{1}$$

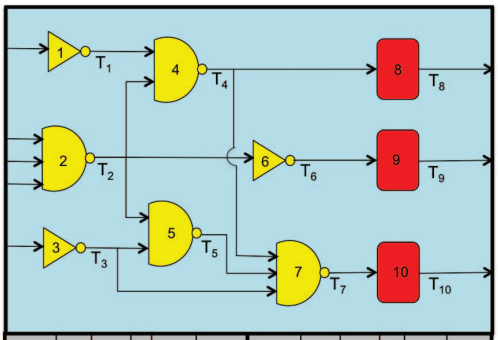
in which $z_i = 1$ represents the nominal size of the gate and μ is a constant scaling factor, typically set to 0.69.

Every gate i has an arrival time (T_i) at its output. For simplicity, we assume an arrival time (T_i) of 0 for primary inputs and that delay paths start at sequential gates. We then calculate the arrival time of each combinational gate as follows:

$$T_i \geq max_{\forall k \in I(i)}(D_{i_k} + (T_{FI(i_k)})) \tag{2}$$

As an example, Fig.1 shows an illustration of a simple circuit and the arrival times of all of its gates. Here, every pin of a gate has the same Cin, R and D.

The size of the speculative window (W) in resilient designs bounds the maximum increase in performance they can achieve when there is no timing error. If the path endpoint has an arrival time within the speculative window, it must be terminated with an error detecting latch/flop. In



Gate	R	C_{in}	A	D	T	Gate	R	C_{in}	A	D	T
1	1.5	2	2	1.04	1.04	6	3	0.5	3	3.11	5.87
2	1	1	3	2.76	2.76	7	3.2	2	1	3.31	7.87
3	0.4	1.5	1	1.24	1.24	8	0.5	1.5	2	0	0
4	0.7	1	2	1.69	4.45	9	0.5	1.5	2	0	0
5	1.3	2.5	4	1.79	4.55	10	0.5	1.5	2	0	0

Figure 1: An example circuit with arrival times: Combinational gates are in yellow and sequential gates are in red. T represents the minimum arrival time of gate outputs.

Figure 1, assuming the synthesis clock cycle (P) is 8 and we aim for a 30% resiliency window, the timing window W is 2.4 time units wide spanning the times 5.6 to 8. Then, the sequential gates G_9 and G_{10} must be error detecting because the arrival times of their inputs (T_6, T_7) are greater than 5.6.

Our approach assumes the relative additional cost associated with each latch or flip-flop that must be error detecting is X. In particular, the objective of our area minimization problem is:

$$min\ (\sum_{i \in C,S} (A_i * z_i) + X * \sum_{j \in S} e_j) \tag{3}$$

For each gate i, the logic area is proportional to its size z_i. The first part of area 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 area model is the additional area associated with the error detecting logic (EDL). The variable e_j is a binary variable whose value is determined by whether the end-point of path ending in 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 is prior to the speculation window, the paths that end at this sequential element are not close to critical and the sequential element need not be error-detecting, i.e., $e_j = 0$. Otherwise, the arrival time must be within the speculation window $[(P - W), P]$ and the sequential element must be error-detecting, i.e., $e_j = 1$. More mathematically, for all $j \in S$, we have

$$e_j = \begin{cases} 1, & \text{if } T_i \geq P - W, \ \ \forall i \in FI(j) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

We propose to solve this problem using geometric programming with some modifications discussed in Section 3.

It is also important to emphasize that the specific structure of the error detecting latches/flops varies among resilient designs and consequently has different associated overheads. Bowman [1] analyzed three types in Fig. 2. a) The

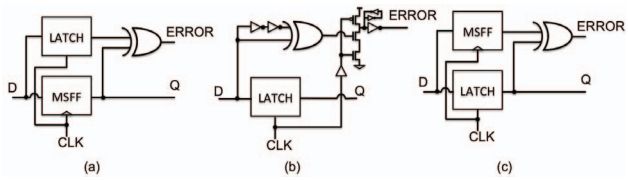


Figure 2: Three kinds of resilient designs in [1]:(a) The simplified Razor FF (b) Error detector and TDTB latch (c) TB latch with error detector

simplified razor flip-flop: a conventional master-slave flip-flop with an additional shadow latch and XOR that detects the difference between the flop and latch [6]. b) A transition detector and time borrowing (TDTB) latch: a conventional time-borrowing latch, an XOR to detect errors, and a C-element to hold the value of errors [13]. c) A time-borrowing latch with a shadow master-slave flop and a XOR. Each of these structures yields an error signal. The error signal of multiple error detecting latches/flops within a pipeline stage must be combined with some type of OR-gate to produce an error signal for an entire pipeline stage. Moreover, some type of synchronizer is often needed in the control path to address metastability. For example, in the asynchronous resilient scheme Blade proposed by Hand et al, Q-Flops were used to sample the error signal in a metastability safe manner and these also contributed to the EDL overhead [8]. Moreover, techniques to make the TDTB more sensitive to glitches come at an additional area cost [13]. This suggests there may be a tradeoff between EDL overhead and robustness. For these reasons, our experiments are conducted with a range of different values of X representing low, medium, and high values of the EDL area overhead. Namely, we choose X to be 0.5, 1, and 2 times the area of a minimum-sized sequential gate.

3. PROPOSED APPROACH

In this section, we solve the problem of minimizing logic and EDL area subject to a performance constraint using geometric programming. Geometric programming enables large-scale non-linear mathematical problems to be solved but required both the objective function and the inequality constraints to be posynomials [2,16]. Section 3.1 shows how we formulate the resynthesis problem described above as a mixed integer geometric program. Section 3.2 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 optimal integer solutions.

3.1 Mixed Integer Geometric Program

In Section 2 we described a mathematical model that uses binary variables e_j to determine if sequential element j need be error-detecting governed by the equation:

$$T_i - W * e_j \leq P - W, \tag{5}$$

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 [2].

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

$$e_j = 2 - ne_j,\ 1 \leq ne_j \leq 2,\ \forall j \in S \tag{6}$$

The delay constraint now becomes

$$T_j + W * ne_i \leq (P + W), \tag{7}$$

which is posynomial.

Substituting ne_j into the EDL portion of the objective function described in Equation 3 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 as follows:

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

Subject to:

$$D_{i_k} = \mu * \frac{R_{i_k}}{z_i} * \sum_{j_l \in FO(i)} Cin_{j_l} * z_j,\ \forall k \in I(i)\ \forall i \in C, S \tag{8}$$

$$\begin{cases} T_i \geq max_{\forall k \in I(i)}\{(D_{i_k} + T_{FI(i_k)})\},\ \forall i \in C \\ T_i = D_i,\ \forall i \in S \end{cases} \tag{9}$$

$$T_j + W * ne_i \leq (P + W),\ \forall i \in S,\ j \in FI(i) \tag{10}$$

Bounds:

$$\begin{cases} LB_i \leq z_i \leq UB_i,\ \forall i \in C \\ z_i = 1,\ \forall i \in S \end{cases} \tag{11}$$

$$1 \leq ne_i \leq 2,\ \forall i \in S, ne_i \in \mathbb{Z} \tag{12}$$

$$0 \leq T_i \leq P,\ \forall i \in C, S \tag{13}$$

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.

Note that the constraints in Equations 8 - 9 implement the same Elmore delay model and arrival time described in Section 2. Moreover, Equations 11 - 12 show the bounds of all variables that can be set to avoid unrealistic changes in size. Lastly, T_i needs to be less or equal to P for all combinational elements i to maintain the desired clock period.

3.2 Geometric Program Iterative Algorithm

The integral constraint on ne_i generally adds significant computational complexity to the mathematical program because they are handled using computationally expensive branch-and-bound techniques [16]. 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 binary values for future iterations. After setting some variables to be integral, we squeeze the high and low thresholds closer together and repeat. We keep running the geometric program iteratively

```
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 3: Relaxation-based iterative algorithm

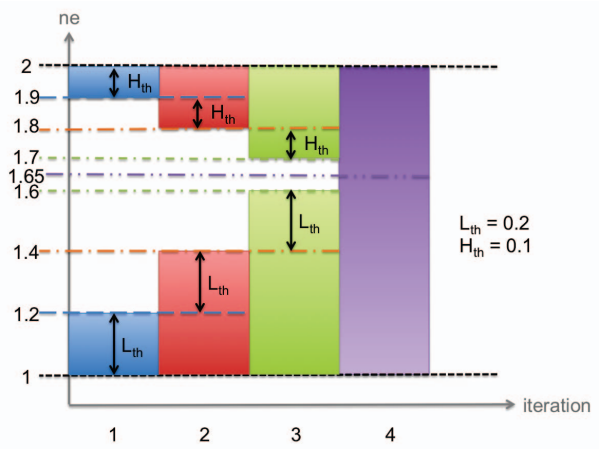


Figure 4: Example of how thresholds vary across iterations

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 3.

Figure 4 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 the value of the *ne* variables from the 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). However, forcing *ne* variables to be 2 might lead to an infeasible solution. If this happens, we call the function *AllowHighCostNegativeSlack*() to add a high cost negative slack variable to the formulation to determine which *ne*'s cannot be set to 2 and reset them back to 1.

In this threshold setting example, there is a maximum of 4 iterations in all runs at which time the high and low threshold cross. When high and low threshold cross, we find the mid-point by averaging high threshold and low threshold of the previous iteration. Then, if variable *ne* is greater than mid-point, *ne* is set to 2. Otherwise, it is set to 1.

The threshold steps play an important role on the quality of the results. Iterations with small threshold steps might have more similar critical paths as the optimal 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 finish faster but have higher chance to force *ne* variables to non-optimal values.

Table 1: Circuit information after initial synthesis

Circuit	Circuit Size			Area	# of EDL	Error-rate
	# of C	# of S	Total			
s1196	331	18	349	343	3	0.25%
s1238	415	18	433	504	6	2.38%
s1423	454	74	528	490	34	0%
s1488	318	6	324	248	6	4.35%
s5378	809	164	973	1028	47	0.79%
s9234	530	125	655	789	64	0.52%
s13207	1562	460	2022	2543	68	27.05%
s15850	1935	448	2383	2686	77	3.31%
s35932	5443	1728	7171	9582	390	14.98%
s38417	5752	1490	7242	8719	451	5.59%
s38584	6684	1248	7932	7974	201	69.78%

3.3 Calculating Resistance and Capacitance

Note that in the geometric program, we use the same Elmore delay model as discussed in Section 2. 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. Eq. 9 is the same as we described in Section 2 for both combinational and sequential gates.

4. EXPERIMENTAL RESULT

We implemented our algorithm using Perl and TCL scripts that interface the Synopsys Design Compiler framework to the YALMIP MIGP solver [16] for MATLAB and evaluated it on ISCAS89 benchmark circuits using an FDSOI 28nm cell library. Table 1 shows the size of chosen benchmark circuits and how many EDLs are needed after normal logic synthesis, prior to starting our resynthesis algorithm.

As mentioned earlier, our mixed integer geometric program and iterative algorithm determine which near-critical paths with EDLs should be sped up to minimize area. We then add *set_max_delay* timing constraints on those paths to constrain them to be indeed non-near-critical after resynthesis with Design Compiler. However, since the synthesis tool does not understand EDL area overhead, it might 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 which static worse-case delay is less than $P - W$ to remain non-critical using additional *set_max_delay* constraints. We set $W = 0.3P$, modeling a resiliency window that is 30% of the original clock period. We calculate a gate's upper and lower sizing bounds by comparing 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 area (min_A_i) and maximum area (max_A_i).

Table 2: Area improvement: (IA: Iterative algorithm, MIGP: Mixed integer geometric program, BF: Brute-force)

Circuit	Area Improvement %								
	High Overhead			Medium Overhead			Low Overhead		
	IA	MIGP	BF	IA	MIGP	BF	IA	MIGP	BF
s1196	11.43	11.43	8.05	9.15	8.44	6.07	8.16	8.16	5.03
s1238	14.27	14.27	1.27	12.65	12.65	-0.18	11.78	11.78	-0.96
s1423	23.03	25.11	16.8	10.89	10.62	8.96	2.91	3.56	4.67
s1488	0.39	2.1	4.58	-2.52	-4.65	3.51	-4.17	-4.17	2.9
s5378	25.13	25.13	7.95	13.56	13.56	4.58	6.32	6.32	2.47
s9234	25.08	30.65	13.79	11.23	TO	8.36	4.82	TO	4.52
s13207	15.98	17.35	9.67	9.56	9.56	3.32	5.35	5.27	1.8
s15850	17.14	17.14	3.65	8.88	TO	2.03	4.09	TO	1.08
s35932	24.39	24.39	6.73	13.81	13.81	3.97	7.32	TO	2.28
s38417	18.29	TO	3.34	9.22	TO	2.2	3.99	TO	1.49
s35854	16.27	TO	-4.49	8.76	TO	-2.4	4.35	TO	-1.2

Table 3: Run-time Comparison

Circuit	Run-time					
	IA		MIGP		BF	
	CLK	CPU	CLK	CPU	CLK	CPU
s1196	1.3m	11.6m	1m	11.5m	1m	2.5m
s1238	2.1m	19m	4m	55m	1.25m	5m
s1423	1.7m	14m	19m	4hr	3.5m	28m
s1488	1.7m	14m	1.5m	18m	1.5m	5m
s5378	7m	1.3hr	1hr	13hr	5m	39m
s9234	2.2m	17.3m	TO	TO	6m	53m
s13207	11.5m	1.77hr	2.8hr	32.5hr	7m	56m
s15850	18.5m	2.78hr	TO	TO	8m	64m
s35932	2.7hr	22.19hr	TO	TO	45m	5hr
s38417	4.8hr	32hr	TO	TO	1hr	6hr
s35854	12.3hr	57.7hr	TO	TO	32m	2.7hr

Then, LB_i and UB_i will be calculated in Eq.14.

$$LB_i = \frac{min\_A_i}{A_i};\ \ UB_i = \frac{max\_A_i}{A_i}; \tag{14}$$

Moreover, for the iterative algorithm, we set $Hth = 0.05$ and $Lth = 0.2$ because it achieves reasonable runtime and similar results to the optimal integer program.

In Table 2, we show the achieved area improvements from our iterative algorithm and mixed integer geometric program as well as the previously-proposed naive brute force method [8].[1] The logic synthesis tool reports logic area and how many paths are near-critical and thus need to be terminated with EDLs. We then calculate the resulting area by summing up logic and EDL area and report the area improvement by comparing calculated areas before and after resynthesis. For the naive brute-force approach, we speed up near-critical paths one at a time and we only show the best area improvement among all of them, as described in [8].

Except for circuit s1488, our algorithm achieves lower area than the brute-force approach with all different EDL overheads. Careful analysis of the s1488 re-synthesized circuits indicate that the synthesis tool in this case unexpectedly reduced area using non-gate-sizing techniques including restructuring which is not modeled in our approach.

On average, our iterative algorithm achieves 10.8% larger area improvement than brute-force with high EDL area overhead, 5.9% larger improvement with medium EDL area overhead, and 2.75% larger improvement with low EDL area overhead. Moreover, the area improvement of the iterative algorithm is similar to that of the mixed integer geometric program. but faster by an average of 5 times. In some circuits, the area improvement of the iterative program is better than the optimal integer program. This may be because of differences in logic synthesis optimizations other than gate sizing, such as restructuring and repeater insertion. For several of the largest circuits the MIGP timed out (TO) after 24 hours of wall clock time. This suggests our iterative algorithm is an effective approach to solve the MIGP. Note that both the geometric programming and brute-force approaches use multi-threaded computation so Table 3 reports both worst clock and CPU run-time among the three overheads. Although our iterative algorithm is slower than brute-force, the run-times are still reasonable.

To further analyze the impact of different threshold settings, we ran three different threshold settings for the high EDL overhead case: A ($Hth = 0.1$, $Lth = 0.2$), B ($Hth = 0.05$, $Lth = 0.2$), and C ($Hth = 0.01$, $Lth = 0.1$) and got average area improvements of (16.32%, 17.31%, 17.12%). As mentioned above, the threshold setting B was chosen in Table 2 because it has reasonable run-times and yields results similar to the optimal integer program.

Speeding-up near critical paths may not only improve area but also improve performance by reducing the error-rate. Although our geometric program only targets minimizing area, we still see an average error-rate improvement of 5.2% over original synthesis and 6.8% over the brute-force approach in Table 4. Even though we reduce the error-rate to 0 in some circuits, for other circuits, such as s38584, the new error-rate remains high. This is interesting as 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 performance considering voltage scaling.

5. CONCLUSION AND FUTURE WORK

The advent of resilient designs offers a path to achieve average-case silicon. Numerous resilient circuit templates and styles have been advocated and explored but few CAD works address this open field. In fact, to the best of our knowledge, this paper is the first to propose a mathematically formal model to optimize the hardware of resilient designs. The initial goal of this research is to optimally tradeoff the area of resilient designs for a worst-case performance target at a fixed voltage supply, but future work should consider modeling power for a given average-case performance considering voltage scaling.

The approach studied is to use analysis of the netlist obtained from standard logic synthesis to guide the resynthesis of the design using additional timing constraints to hopefully reduce the final area of the design (including the area associated with the error-detecting logic overhead). In particular, during resynthesis many types of optimizations may be explored including gate sizing, logic restructuring, and repeater insertion. But we hypothesize that it is sufficient to predict the impact of resynthesis by considering gate sizing

[1]It would also be interesting to compare our approach to that of [9], but unfortunately their tool is not publicly available.

Table 4: Error-rate (%)

Circuit	Old Error-rate	Overhead	Error-rate					
			IA		MIGP		BF	
			Δ	New	Δ	New	Δ	New
s1196	0.25	H/M/L	-0.25	0	-0.25	0	-0.08	0.17
s1238	2.38	H/M/L	-0.08	2.3	-0.08	2.3	-0.41	2.79
s1423	0	H/M/L	0	0	0	0	0	0
s1488	4.35	H	2.98	7.33	-4.35	0	0.56	4.91
		M	2.98	7.33	-2.56	1.79	0.56	4.91
		L	2.98	7.33	2.98	7.33	0.56	4.91
s5378	0.79	H/M/L	-0.79	0	-0.79	0	-0.29	0.59
s9234	0.52	H/M/L	-0.52	0	-0.52	0	-0.37	0.15
s13207	27.05	H	-27.05	0	-27.05	0	-26.09	0.96
		M	-27.02	0.03	-27.05	0	-26.09	0.96
		L	-27.02	0.03	-27.02	0.03	-11.88	15.17
s15850	3.31	H	-1.03	2.28	-1.15	2.16	0.4	3.71
		M/L	-1.67	1.64	TO	TO	0.4	3.71
s35932	14.98	H/M/L	-14.98	0	-14.98	0	0	14.98
s38417	5.59	H	-3.89	1.7	TO	TO	26.08	31.67
		M	-3.64	1.95	TO	TO	45.69	51.28
		L	-3.81	1.78	TO	TO	45.69	51.28
s35854	69.78	H	-10.88	58.9	TO	TO	1.11	70.89
		M	-14.67	55.11	TO	TO	1.11	70.89
		L	-7.63	62.15	TO	TO	1.11	70.89

alone. To test this hypothesis, this paper presents a gate-sizing based mixed integer geometric programming framework and a relaxed iterative algorithm to guide resynthesis. Our experimental results show that our iterative algorithm significantly improves the area compared to a previously published brute-force resynthesis approach while for most examples simultaneously reducing error-rates. Moreover, the iterative algorithm dramatically reduces the computational complexity associated with the mixed integer geometric program and still achieves close to optimal (integer) results. These results suggest that a gate-sizing based model can indeed be used to effectively guide resynthesis.

We tested our approach after traditional logic synthesis but applying it after place-and-route is also possible and would enable us to more accurately model interconnect. Finally, we believe this framework is particularly interesting because the mathematical objective function can be enhanced to capture not only area but also overall error-rate/performance and ultimately power. This is part of our future work.

Acknowledgements

This work is supported in part by NSF award CCF-1219100.

6. 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*, 44(1):49–63, Jan 2009.

[2] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. *Optimization and engineering*, 8(1):67–127, 2007.

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

[4] 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*, 44(1):32–48, Jan 2009.

[5] 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*, 98(2):253–266, Feb 2010.

[6] 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*, pages 7–18, Dec 2003.

[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*, 48(1):66–81, Jan 2013.

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

[9] A. B. Kahng, S. Kang, J. Li, and J. Pineda De Gyvez. An improved methodology for resilient design implementation. *TODAES*, 20(4):66, 2015.

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

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

[12] Y. Liu, F. Yuan, and Q. Xu. Re-synthesis for cost-efficient circuit-level timing speculation. In *DAC*, pages 158–163. ACM, 2011.

[13] M. T. Moreira, D. Hand, N. L. V. Calazans, and P. A. Beerel. TDTB error detecting latches: Timing violation sensitivity analysis and optimization. In *ISQED*, 2015.

[14] 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*, 40(1):28–35, 2005.

[15] 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*, pages 112–113, June 2009.

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

[17] R. Ye, F. Yuan, H. Zhou, and Q. Xu. Clock skew scheduling for timing speculation. In *DATE*, pages 929–934. IEEE, 2012.