Automatic Retiming of Two-Phase Latch-Based Resilient Circuits

Huimei Cheng, *Member, IEEE,* Hsiao-Lun Wang, *Member, IEEE,* Minghe Zhang, *Member, IEEE,* Dylan Hand, *Member, IEEE,* Peter A. Beerel, *Senior Member, IEEE*

***Abstract*—Timing resilient design has shown significant promise in mitigating the excess margins associated with rare worst-case data and increased process, voltage, and temperature (PVT) variations. However, resilient circuits need error detecting sequential logic (EDL) to detect timing errors which incur area and power overhead. This paper proposes two alternatives to reduce the overhead in two-phase latch-based resilient circuits. The first is a new resiliency-aware graph-based approach to solve the retiming problem. The second uses a virtual resynthesis library to enable commercial synthesis tools to recognize the EDL overhead and optimize total area during retiming. We compare both approaches to a commercially-standard retiming approach, which ignores the resiliency overheads, on a wide variety of benchmarks. Our experimental results show that our methods are computationally efficient and reduce the total circuit area by an average of up to 10-15% when compared to traditional retiming.**

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 PVT 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 timing resilient circuits have been proposed. For example, canary FFs attached to replica critical paths predict when the design is close to a timing failure [3]. Designs can then adjust their supply voltage or clock frequency either statically or dynamically to ensure correct operation at the edge of failure [4]. Other resilient design techniques use extra logic to detect and recover from timing violations [5]–[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 often be traded off for lower power via further voltage scaling.

H. Cheng, H.-L. Wang, D. Hand, and P. A. Beerel are with the University of Southern California, Los Angeles, CA. P. A. Beerel is also Chief Scientist at Reduced Energy Microsystems. M. Zhang is currently with Georgia Institute of Technology. This work was supported in part by NSF Grant #1619415.

Some EDA techniques have been proposed to minimize the probability of timing errors [9]–[11]. Of particular importance to this paper, however, 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 EDL 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. Previously, resynthesis techniques have been proposed to reduce overheads [8], [11], [12] in which near-critical-paths are sped-up by re-running logic synthesis with a tighter max delay constraint to reduce the EDL needed at the cost of increased logic area. Other research has explored retiming flop-based resilient designs, i.e., relocating sequential gates to reduce the amount of EDL without changing the combinational logic [13].

Our work, on the other hand, is focused on retiming latch-based resilient circuits, for which both synchronous [7], [14] and asynchronous [8] styles have been proposed. In particular, we assume a flow in which flops are converted into master-slave latches and only the slave latches are retimed to avoid challenges associated with changing the circuit's initial state [15]. Latch-based resilient circuits have higher hold margins but introduce complications in managing error-detecting overhead associated with doubling the number of the sequential elements. Following the assumption in [8], [14] that the clock cycle is composed of one error-detecting stage followed by zero or more time-borrowing stages, we make all master latches to be the error-detecting stage and all slave latches to be time-borrowing stages. Slave latches can be reduced by retiming and master latches can be made non error-detecting if their input arrival times falls before the opening of the resiliency window. To minimize the total area, this paper presents two approaches, a graph based method (G-RAR) and a virtual library (VL) based method, and compares both to the traditional base-retiming resilient-unaware method.

This paper extends [16], which introduces an Integer Linear Programming (ILP) formulation to model retiming latch-based resilient circuits, efficiently solves the ILP via a network flow transformation, and offers comparisons to min-area retiming. However, [16] evaluates this technique in isolation using an academic logic synthesis tool with a simple gate-delay based model. To assess real-world practicality, this paper integrates the technique within a commercial synthesis tool that uses a path-based delay model and compares it with existing commercial-grade retiming algorithms. This paper also proposes a second approach to retiming using a virtual library that enables the logic synthesis tool to incorporate the area tradeoff between error-detecting and non-error-detecting latches in its optimization routines. A similar approach was employed by [17] to reclaim logic area through resynthesis of resilient circuits. This paper shows their approach can be modified effectively for retiming as well. In addition, we improve the virtual library approach with a post-retiming step that effectively reclaims area by swapping unnecessary error-detecting latches with their smaller non-error-detecting counterparts.

Experimental results on a wide variety of benchmark circuits show that the G-RAR approach can reduce sequential logic area by an average of up to 29.6% and total area by an average of up to 15% compared to the baseline retiming approach. Moreover, G-RAR outperforms the best VL based approach by an average of 5.1% in total logic area. Benchmarks of both algorithms completed on all ISCAS89 circuits [18] within 10 minutes of CPU time on a modest Intel Xeon-based workstation. We also ran our algorithms on a more complex open-core CPU, Plasma [19], with run-times of less than 62 minutes.

The remainder of the paper is organized as follows. Section II introduces the background of retiming and resilient circuits. Section III formulates the problem. Then, Section IV develops our network simplex approach after which Section V describes our library based approach. Section VI shows our experimental results, followed by a brief summary in Section VII.

II. Background

A. Two-Phase Latch-Based Resilient Circuits

The clock model of a latch-based circuit design with k periodic clocks and k-phases [20] is generally notated as $\Pi = \langle\phi_1, \gamma_1, \phi_2, \ \gamma_2, \ldots, \phi_k, \gamma_k\rangle$, where ϕ_i the transparent window of phase i, and γ_i is duration from the falling edge of phase i to the rising edge of phase $i+1$. For a two-phase latch-based design, $\Pi = \langle\phi_1, \gamma_1, \phi_2, \gamma_2\rangle$. As in [8], [14], we assume a symmetric clocking scheme for resilient latch-based design in which $\phi_1 = \phi_2$ and $\gamma_1 = \gamma_2$ and a timing resiliency window of ϕ_1. Fig. 1 shows the clocking diagram of a two-phase latch-based resilient circuit. Assuming ideal clock trees, the maximum delay P of all logic paths between master stages in the circuit is $\phi_1 + \gamma_1 + \phi_2 + \gamma_2 + \phi_1 = \Pi + \phi_1$. However, the period Π of such circuits is only $\phi_1 + \gamma_1 + \phi_2 + \gamma_2$. Therefore, if data transitions during the timing resiliency window of a master latch, the rising edge of the next pipeline stage's clock must be delayed to ensure the setup time requirement is satisfied for this and subsequent stages. Thus, timing constraints on the error detection logic necessitate smartly grouping error-detecting latches into manageable clusters [8] and, in the case of synchronous designs, imposing a complex local clock stalling scheme [7]. Note also that, in this formulation, time borrowing is allowed for slave stages while only master stages can be error detecting. This reduces the overhead of EDL while simultaneously reducing the complexity of the timing constraints [8], [14].

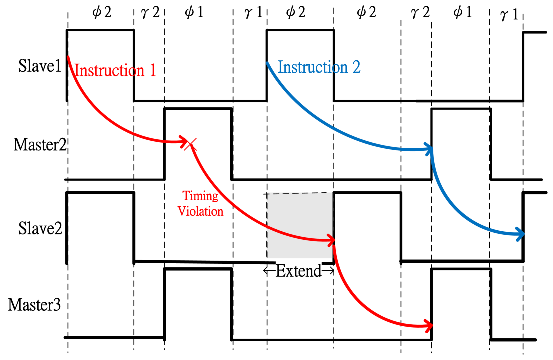


Fig. 1: Timing of a two-phase latch-based resilient circuit.

B. Error Detecting Latches

Latch-based timing resilient circuits rely on error detecting latches to detect whether the data remains constant during the timing resiliency window. Fig. 2 shows two types of error detecting latches [1]: a) a time-borrowing latch with a shadow master-slave Flip-Flop (MSFF) in which the MSFF samples the value of data at the rising edge of resiliency window and an XOR gate compares the sampled value with the data during the resiliency window and b) a transition detecting time-borrowing latch (TDTB) which consists of a conventional latch, an XOR gate to detect transitions, and an asymmetric C-element to hold the error value. In addition, the error signals of all error detecting latches within a pipeline stage must be routed and collected with some type of OR gate tree to produce a single error signal for the entire stage. There are specific forms of EDL designs that target low power [14] and/or handle metastability [8]. Different EDL designs have varying area overhead relative to normal sequential design, which in this paper is denoted c. For the purposes of this paper we consider a range of c from 0.5 to 2, similar to [12], representing the fact that the amortized area of different proposed EDL schemes can range from 50% to 2X larger than a normal latch.

C. Min-Area Retiming

There are two traditional objectives for retiming algorithms: minimizing the clock period [21], [22] and minimizing the number of sequential elements [21], [23], [24]. We now review the traditional min-area retiming algorithm because it is more closely related to our work. As in [21], a sequential circuit can be described as $G(v, e)$ where each v represents a combinational gate and each directed edge e_{uv} represents a connection between gate u and gate v. The weight of the edge $w(e_{uv})$ represents the number of FFs or latches between gate nodes. Each vertex has a fixed delay $d(v)$. A special vertex, the host vertex, is introduced into the graph. Edges are added from the host vertex to all primary inputs of the circuit and from all primary outputs to the host vertex. Two additional variables $D(u, v)$ and $W(u, v)$ are defined as follows:

$$W(u, v) = \min_{\forall p, u \rightsquigarrow v} w(p) \tag{1}$$

$$D(u, v) = \max_{\forall p, u \rightsquigarrow v, \ w(p) \ = \ W(u,v)} d(p) \tag{2}$$

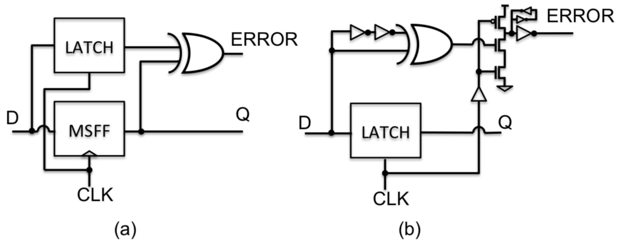


Fig. 2: Two examples of error detecting latches [1].

In the equations above, p represents a path from gate u to gate v, $w(p)$ represents the total number of latches along p, and $d(p)$ is the delay from u to v via p.

The notation $r(u)$ is introduced to represent the number of FFs or latches that are retimed from the output of gate u towards the inputs of gate u. $w_r(e_{uv}) = W(u,v) - r(u) + r(v)$ denotes the number of FFs or latches on edge $e(u,v)$ after retiming. P is the clock period and the path delay between any two sequential gates should be less than P, ignoring setup time for simplicity. Mathematically, this means for a retiming to be legal, for all $D(u,v) > P$, $w_r(e_{uv}) \geq 1$. Substituting the definition of $w_r(e_{uv})$, this constraint becomes $r(u) - r(v) \leq W(u,v) - 1$, $\forall D(u,v) > P$.

The problem of minimizing the total number of registers after retiming should also consider the trivial optimization of sharing registers across different fanouts of the same register. As an example, instead of adding k registers along each of k fanout edge of a node, we should instead add one register that fans out to each of the k fanouts. To model this fanout sharing, for a gate u with fanout k, a *breadth* $\beta(e_{uv}) = \frac{1}{k}$ is added as the cost coefficient of a flip-flop on an edge e_{uv} [21]. The objective function and constraints of the classic retiming algorithm for flip-flop based design can thus be described as follows:

$$\begin{aligned} \min \sum_{v \in V} & \left[\sum_{\forall u \in FI(v)} \beta(e_{uv}) - \sum_{\forall u \in FO(v)} \beta(e_{uv}) \right] r(v) \\ s.t. \quad & r(u) - r(v) \leq w(e_{uv}) \qquad \forall e_{uv} \in E \\ & r(u) - r(v) \leq W(u,v) - 1 \quad \forall D(u,v) > P \end{aligned} \tag{3}$$

To simplify the constraints, [24] re-writes them as follows:

$$\begin{aligned} \min \sum_{v \in V'} & \left[\sum_{\forall u \in FI(v)} \beta(e_{uv}) - \sum_{\forall u \in FO(v)} \beta(e_{vu}) \right] r(v) \\ s.t. \qquad & r(u) - r(v) \leq c_{uv} \quad \forall (u,v) \in C' \\ & L_u \leq r(u) \leq U_u \quad \forall u \in V' \end{aligned} \tag{4}$$

Here, L_u and U_u are the lower and upper bounds of the number of latches or flip-flops that can be moved backward through gate u without violating the timing constraints [24] and their values are typically calculated through ASAP and ALAP skews [22]. C' is the set $\{(i,j) \in C | U_i - L_j > c_{ij}\}$, where C is the constraints set of (3), and $V' = \{v \in V | U_v \neq L_v\}$.

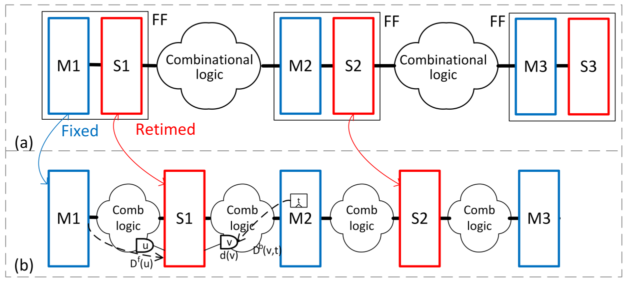


Fig. 3: (a) Original FF based design. (b) Two-phase latch-based resilient circuit obtained after retiming. Our algorithm minimizes the overall cost of sequential elements by choosing the optimal retimed position for all slave latches.

III. The Problem Statement

As mentioned earlier, in our latch-based retiming algorithm we assume the location of the master latches are fixed and retime only the slave latches as illustrated in Fig. 3. The original position of slave latches before retiming are at the output of their associated master latches. The combinational logic path delay between the master latches in stage 1 (M1) and the master latches in stage 2 (M2) is unaffected by retiming because the positions of master latches in M1 and M2 are fixed. However, the different positions of the slave latches change the arrival time of the data at the M2 master latches. This is because the slave latches become transparent at time $\phi_1 + \gamma_1$ and only then start propagating signals through the subsequent combinational logic as illustrated in Fig 1. The delay from source master latches s through a repositioned slave latch on edge $e(u,v)$ to a given destination master t can be expressed as

$$\begin{aligned} A(u,v,t) = & \max\{\phi_1 + \gamma_1 + d^{ck\_q}(l), D^f(u) + d^{d\_q}(l)\} \\ & + d(v) + D^b(v,t) \end{aligned} \tag{5}$$

Here, $d(v)$ is the delay of gate v and $D^f(u)$ is the maximum delay from any master latch in the current stage to the slave latch at output of gate u, obtained through classic static timing analysis (STA). Similarly, $D^b(v,t)$ is the maximum delay from a slave latch at output of gate v to the destination master latch t. It is the delay calculated backwardly starting from master t to slave v. $d^{ck\_q}(l)$ and $d^{d\_q}(l)$ are the *clock* to Q delay and D to Q delay of a latch, respectively. This equation differs slightly from $A(u,v,t)$ in [16] by differentiating a latch's D-to-Q delay from its clock-to-Q delay, which may vary by up to 40% in a modern standard cell library. In the case of gate u being placed close to the startpoint, which means data always arrives before the latch opens if the latch is placed between u and v, the arrival time to the endpoint t is $\phi_1 + \gamma_1 + d^{ck\_q}(l) + d(v) + D^b(v,t)$. In the other case that the max delay through combinational logic to gate u is larger than $\phi_1 + \gamma_1$, the delay till the output of the latch if it is placed after u is $D^f(u) + d^{d\_q}(l)$ and $A(u,v,t)$ should be computed as $D^f(u) + d^{d\_q}(l) + d(v) + D^b(v,t)$. This computation is more accurate and closer to the STA used in commercial-grade synthesis tools and thus will generate a better model for G-RAR. Figure 3(b) shows the delay from M1 to a certain master latch t in M2 passing through a slave latch between gates u

and v. The arrival time at the master latch t is maximum of (5) over those slave latches in its fan-in cone.

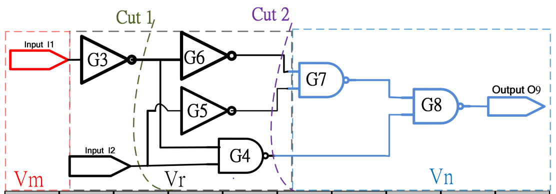
For a retiming to be valid, there must be exactly one slave latch along any path from a master s to a master t. In Fig. 3, when we want to find the optimal positions of latches in S1, the source master s is in M1, and destination master t is in M2. Similarly, when we want to find the optimal positions of latches in S2, the master s is in M2 and master t is in M3. As Fig. 1 shows, if the data changes during the resiliency window ϕ_1, the clock signal of the next stages is delayed by ϕ_1. Data launched by a master latch s has $\phi_1 + \gamma_1 + \phi_2$ to propagate to the slave latches of the next stage even if the data is still changing during the resiliency window. For this reason, we can view that a master s always propagates data at time 0 and that slave latches are transparent during time $\phi_1 + \gamma_1$ to $\phi_1 + \gamma_1 + \phi_2$. In Fig. 3, we can assume M2 always propagate data at time 0. The combinational logic and sequential elements before stage M2 do not affect the arrival time at stage M3. As a result, whether the master latches in M3 are error-detecting or not depends on the positions of the slave latches S2 and the combinational logic between M2 and M3. It does not depend on the positions of the slave latches of S1 or combinational logic between M1 and M2. Similarly, whether master latches in M2 are error-detecting only depends on the positions of S1 and combinational logic between M1 and M2. In other words, each pipeline stage can be retimed independently without any loss of optimality.

For a master latch t, the fan-in cone of t is denoted as $FIC(t)$. If $\exists\, v \in FIC(t)$, $D^f(v)+D^b(v,t) > \Pi$, then master latch t must be error detecting regardless of where the slave latches are placed. However, when $D^f(v) + D^b(v,t) \leq \Pi$ the location of the slave latches determines whether or not the master latch must be error-detecting. For example, if the slave latches v are close to the master latch s and $D^b(v,t)$ is large, then a master latch t may have an arrival time that exceeds Π forcing it to be error-detecting. If, however, we move these slave latches forward through more combinational logic, the total path delay between two master stages s and t may decrease to less than or equal to Π due to a lower $D^b(v,t)$. Thus, the error detecting latch can be replaced with a standard latch. For this reason, the position of slave latches directly impacts the total number of error-detecting latches. We refer to master latches whose error-dependent status is retiming dependent as *target* master latches.

Note, the maximum distance we can move any slave latch through the combinational logic is dictated by the constraint that the data needs to stabilize through the slave latch before it becomes opaque. This is sometimes referred to as the time borrowing constraint. Assuming the slave latch has a transparency phase of time ϕ_2, if a slave latch is placed at gate v, we have

$$D^f(v) \leq \phi_1 + \gamma_1 + \phi_2. \tag{6}$$

Similarly, the data from slave stages v should arrive at termination master latches t before they become opaque. That is, if a slave latch is placed at node v, we have following



GATE(v)	d	$D^f(v)$	$D^b(v,O7)$	Region	B(v)	X(v)
I1	0	0	9	Vm	0	0
I2	0	0	7	Vr	0	0
G3	2	2	7	Vr	0	0
G4	2	4	1	Vr	1	-1
G5	5	5	2	Vr	1	-1
G6	5	7	2	Vr	1	-1
G7	1	8	1	Vn	-1	1
G8	1	9	0	Vn	-1	1
O9	0	9	0	Vn	-1	1

Node(v)	B(v)	X(v)
host	2	-2-c
P(O9)	0	c
mG3	-1	1
mI2	-1	1

Fig. 4: An illustrative circuit.

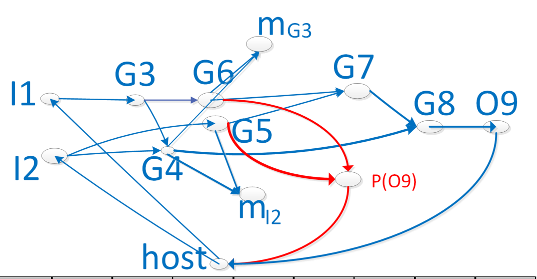
constraint for all terminating master latches t,

$$D^b(v,t) \leq \phi_2 + \gamma_2 + \phi_1 \tag{7}$$

The goal of our work is to retime the slave latches to minimize the overall sequential logic cost, including the total number of slave and master latches and the error-detecting overhead, while still satisfying the above constraints.

Fig. 4 is an example used to illustrate the problem. The circuit is cut at the flip-flops (turned master-slave latches) such that the primary inputs (PI) of the circuit include outputs of the (fixed) master latches and the primary output (PO) of the circuit can in reality be the input of a (fixed) master latch as the combinational logic cloud in Fig. 3(a). Note that slave latches before retiming are at the inputs of the circuit. We find a new cut to place the slave latches in the combinational logic cloud.

Assume $\phi_1 = \gamma_1 = \phi_2 = \gamma_2 = 2.5$ and $D_l = 0$. The delay of each gate is shown in column d, and $D^f(v), D^b(v,t)$ of each gate is shown in the third and fourth column respectively of lower left-hand table in Fig. 4. The slave latches cannot be placed at the output of $G7$ or $G8$ since the forward delay $D^f(G7), D^f(G8)$ would be larger than $\phi_1 + \gamma_1 + \phi_2 = 7.5$ which violates Constraint (6). Slave latches can neither be placed at $I1$ since the backward delay from master $O9$, $D^b(I1, O9)$, is larger than $\phi_2 + \gamma_2 + \phi_1 = 7.5$ which violates Constraint (7). If we choose Cut1 to place the slave latch, the arrival time at $O9$ is $max(D^f(G3), \phi_1 + \gamma_1) + D^b(G6, O9) + d(G6) = 12$. Since this is larger than $\Pi = 10$, $O9$ should be an error detecting master latch. If we choose Cut2 to place the slave latches, there are two paths from inputs to output $O9$, the longest delay goes through $G6$, and arrival time at $O9$ is $\max(D^f(G6), \phi_1 + \gamma_1) + D^b(G7, O9) + d(G7) = 9$ which is less than $\Pi = 10$. We can analyze the total sequential element cost and compare between the two cuts. Cut1 requires two slave latches and one error detecting master latch, while Cut2 needs three slave latches and one non-error detecting master latch. Suppose the area cost of an error-detecting latch is three units, and the cost of a slave and non-error-detecting master latch is one, i.e., the amortized overhead associated with an error-detecting latch $c = 2$. Cut1 needs 5 units cost, but Cut2 only needs 4 units cost. Traditional min-area retiming only minimizes the number of latches (Cut1) while a resilient-aware retiming minimizes the overall area of latches (Cut2).



edge(u,v)	$\beta(e_{uv})$	$w(e_{uv})$	edge(u,v)	$\beta(e_{uv})$	$w(e_{uv})$	edge(u,v)	$\beta(e_{uv})$	$w(e_{uv})$
(I1,G3)	1	0	(G6,mG3)	0.5	0	(G7,G8)	1	0
(I2,G4)	0.5	0	(G4,mG3)	0.5	0	(G8,O9)	1	0
(I2,G5)	0.5	0	(G3,G4)	0.5	0	(O9,h)	0	0
(h,I1)	1	1	(G3,G6)	0.5	0	edge(u,v)	cost	$w(e_{uv})$
(h,I2)	1	1	(G6,G7)	1	0	(G5,P(O9))	0	0
(G4,mI2)	0.5	0	(G5,G7)	1	0	(G6,P(O9))	0	0
(G5,mI2)	0.5	0	(G4,G8)	1	0	(P(O9),h)	-c	0

Fig. 5: Node graph representation of the circuit in Fig. 4.

IV. Graph-Based Retiming

In this section, we modify the retiming graph to incorporate the resiliency window and EDL overheads and introduce the notion of retiming regions to pre-divide the graph nodes into different types. After that, an Integer Linear Programming (ILP) formulation is developed and translated to an equivalent network flow problem. This yields a computationally efficient algorithm for the retiming problem without loss of exactness.

A. *Modified Retiming Graph*

Our graph-based approach to solve the retiming problem enhances the traditional latch-based retiming algorithm to incorporate timing resilient designs. To start, we pre-compute the cut set of gates $g(t)$ for each target master t for which, if the slave latches are moved forward beyond these gates, the target master t need not be error detecting. This condition is met when the retiming of latches beyond the gates in $g(t)$ satisfies (5) $<$ Π. In particular, for a master t, the corresponding $g(t)$ is the set of gates:

$$g(t) = \{v | \exists n \in FO(v), A(v,n,t) \leq \Pi \quad (8)$$
$$\wedge \exists k \in FI(v), A(k,v,t) > \Pi\} \quad (9)$$

Notice that if the longest combinational logic path delay from the previous master stage s to the master latch t in the subsequent stage is larger than Π, then $g(t)$ is the empty set and master t must be error detecting regardless of where the slave latches are re-timed. If the combinational logic delay from any previous master stage s to master t is smaller than $\phi_2+\gamma_2$, then master t is an non-error detecting latch regardless of where the slave latches are re-timed and $g(t)$ is also empty. If $g(t)$ is non-empty, every path from the previous stage to the master t traverses at least one gate in $g(t)$. If all slave latches are placed in the fan-out cone of $g(t)$ and in the fan-in cone of t, master t may be a non-error detecting latch since any slave latches placed on edge $e(u,v)$ in the fan-out cone of $g(t)$ satisfy $A(u,v,t) \leq \Pi$. However, if any slave latch is positioned between gates u, v in the fan-in cone of $g(t)$, t is forced to be an error-detecting latch since $A(u,v,t) > \Pi$. To find $g(t)$, a reverse depth first search (DFS) from master t to s is performed until the constraints of $g(t)$ are satisfied. For example, the $g(O9)$ of Fig. 4 is $\{G6, G5\}$, since $A(G6,G7,O9) = 9 < 10, A(G3,G6,O9) = 12 > 10$ and $A(G5,G7,O9) = 7 < 10, A(I2,G5,O9) = 12 > 10$. If slave latches are placed at the fan-out of $G6$ and $G5$, latch $O9$ can be non-error-detecting. Cut2 satisfies this requirement, but Cut1 does not.

A new retiming graph is created with extra edges and nodes to reflect the benefits of moving latches to the point where specific masters need not being error detecting. In particular, for each target master latch t, we add edges from all gates in $g(t)$ with cost 0 to a pseudo node $P(t)$. The pseudo node $P(t)$ connects to the host node h with another edge with negative cost $-c$, where c is the overhead of an error detecting latch as defined earlier. The set of additional edges added to traditional retiming graph is defined as $E2$. Set $E1$ contains the original edges of the retiming graph. In addition, we place the nodes of the traditional retiming graph in set $V1$, including the host node h, and the pseudo nodes $P(t)$ in set $V2$, as illustrated in Fig. 5. The nodes m_{G3} and m_{I2} in Fig. 5 are pseudo nodes used to model fanout sharing as described in [21]. $E1$, $V1$ are shown in blue in Fig. 5, while $E2$, $V2$ are shown in red. Notice that we create a pseudo node $P(O9)$ for master latch $O9$, and $G5, G6 \in g(O9)$ are connected to $P(O9)$. If a slave latch is placed on $edge(u,v) \in E1$, we have the slave latch cost $\beta(e_{uv})$ as the table in Fig. 5 indicates. If a slave latch is placed on edge $(P(O9), h) \in E2$, we know that the slave latches are moved to the fan-out of $g(O9)$ thus making $O9$ non-error-detecting, and we add cost $-c$ to the objective function to deduct the associated EDL overhead. Lastly, we list in Fig. 5 the original number of slave latches on each edge before retiming $w(e_{uv})$. Initially the slave latches are at the inputs, so $w(e_{h,I1}), w(e_{h,I2})$ are 1 and $w(e_{uv})$ of all other edges are zero.

B. *Retiming Regions*

Since we only retime slave latches between master stages and the slave latches before retiming are placed at the inputs of combinational logic, as illustrated in Fig. 3(a), the retiming value $r(v)$ can only be -1 or 0. To reduce the complexity of the retiming algorithm, we pre-divide the graph nodes into three regions.

1) V_m is the set of gates v for which there exists a terminating latch t for which $D^b(v,t) > \phi_2 + \gamma_2 + \phi_1$. The slave latches should be retimed through these gates (i.e., $r(v) = -1$) and thus after retiming there should be no slave latches in this region. Otherwise, Constraint (7) is violated. For example, for the circuit shown in Fig. 4, V_m is $\{I1\}$ as $D^b(I1, O9) = 9$ which exceeds $\phi_2 + \gamma_2 + \phi_1 = 7.5$.
2) V_n is the set of gates v whose $D^f(v)$, the delay from a master s to gate v, exceeds $\phi_1 + \gamma_1 + \phi_2$. These gates can be determined by analyzing $D^f(v)$ via static timing analysis. No slave latches should be retimed through such gates (i.e., $r(v) = 0$). Otherwise, Constraint (6) is violated. For example, for the circuit shown in Fig. 4, V_n is $\{G7, G8, O9\}$ as $D^f(v)$ of $G7, G8$, $O9$ equals 8, 9, and 9 which all exceed $\phi_1 + \gamma_1 + \phi_2 = 7.5$.

3) V_r is the remaining region of combinational gates except V_m, V_n. $\forall v \in V_r, -1 \leq r(v) \leq 0$. This is the region where slave latches can be positioned after retiming. The optimization procedure only needs to determine the value of $r(v)$ in this region. For example, for the circuit shown in Fig. 4, V_r is $\{I2, G3, G4, G5, G6\}$.

C. ILP formulation

The following Integer Linear Programming (ILP) formulation minimizes the combined area of slave latches and error detecting latches:

$$\begin{aligned}\min \sum_{v \in V1} &\left[\sum_{\forall u \in FI(v)} \beta(e_{uv}) - \sum_{\forall u \in FO(v)} \beta(e_{vu}) \right] r(v) + \\ \sum_{P(t) \in V2} &-c \times (r(h) - r(P(t))) \\ s.t. \quad & r(u) - r(v) \leq w_{u,v} \quad \forall (u,v) \in E_1 \\ & r(u) - r(v) \leq 0 \qquad \forall (u,v) \in E_2 \\ & L_v \leq r(v) \leq U_v, r(v) \in \mathbb{Z} \qquad \forall v \in V\end{aligned} \tag{10}$$

Equation (10) is similar to traditional retiming (4) except for the extra node set $V2$ in the objective function and the extra constraints associated with $E2$. As Fig. 5 shows, if a slave latch is positioned on edge $e_{uv} \in E1$ after retiming, the cost is $\beta(e_{uv})$. If a slave latch is positioned on edge $e_{P(t),h}$, the cost is $-c$ because the master t can be non-error detecting, saving c units of area. $r(h) - r(P(t))$ is the number of slave latches on edge $e_{P(t),h}$ after retiming. The $-c \times (r(h) - r(P(t)))$ in the objective function represents the reduction of EDL overhead.

The extra constraints $r(u) - r(v) \leq 0, \forall (u,v) \in E2$ guarantee that there is a slave latch on $e_{P(t),h}$ only when the slave latches are moving beyond all gates in $g(t)$. As discussed in Section IV-B, the bounds on $r(v)$ depend on the region of gate v. After the ILP solver returns the value of the $r(v)$ variables, the slave latches are placed on the edge where $w_{u,v} + r(v) - r(u) = 1$.

To construct the ILP for Fig. 4, we use the retiming graph in Fig. 5. Each $edge(u,v)$ represents constraints $r(u) - r(v) < w(e_{uv})$, $w(e_{uv})$ is represented in the table in Fig. 5. The objective function can be obtained using the listed $\beta(e_{uv})$ values and a proper choice of EDL overhead c. The upper bound and lower bound of each node depends on the region. If $v \in V_m, -1 \leq r(v) \leq -1$. If $v \in V_n, 0 \leq r(v) \leq 0$, i.e., $r(v) = 0$. If $v \in V_r, -1 \leq r(v) \leq 0$. Also, $r(h) = 0$ and nodes in $V2$ have the same bounds nodes as nodes in V_r. The ILP solver would return $r(I1) = r(I2) = r(G3) = r(G4) = r(G5) = r(G6) = r(P(O9)) = -1$ with all other $r()$ values set to 0.

D. Network Flow Formulation

To solve (10) with integer retiming values $r(u)$, we can, in principle, call an Integer Linear Programming (ILP) solver directly. However, ILPs are NP-hard and thus the worst case run-time is exponential in the size of the problem statement. Rather, we show that our modified retiming formulation (10) can be mapped to a min cost network flow algorithm, similar to traditional retiming [24], and solved with the network simplex method [25] in polynomial time.

Towards this goal, instead of solving the min in (10), we solve a max of the modified objective function in which each coefficient is multiplied by -1 in the Lagrangian expression equation (12). For simplicity and legibility, we define

$$B(v) = \sum_{\forall j \in FO(v)} \beta(e_{vj}) - \sum_{\forall j \in FI(v)} \beta(e_{jv}). \tag{11}$$

Adding the constraints in (10) with Lagrange multipliers x_{uv} for each corresponding constraints in $(u,v) \in E1, E2$ of (10) we obtain the following Lagrange function:

$$\begin{aligned}L(r,x) = &\sum_{v \in V1} r(v)[B(v)] + \sum_{P(t) \in V2} r(P(t))[-c] + r(h)c|V2| \\ &+ \sum_{(u,v) \in E1} (w_{uv} + r(v) - r(u))\, x_{uv} \\ &+ \sum_{P(t) \in V2} \sum_{g \in g(t)} (r(P(t)) - r(g))\, x_{g,P(t)} \\ &+ \sum_{P(t) \in V2} (r(h) - r(P(t))) x_{P(t),h}\end{aligned} \tag{12}$$

We next introduce the notion of the demand of node v,

$$X(v) = \sum_{(u,v) \in E} x_{uv} - \sum_{(v,u) \in E} x_{vu}, \tag{13}$$

i.e., the amount of total flow on v's inputs minus the total flows on v's outputs. We then transform (12) into a min cost network flow formulation similar to traditional min area retiming [21], [24], with the difference that we have extra pseudo nodes $P(t)$ for each master latch t:

$$\begin{aligned}&\min \sum_{(u,v) \in E1} w_{uv} x_{uv} \\ s.t. \quad & X(v) = -B(v) \qquad \forall v \in V1 - h \\ & X(P(t)) = c \qquad \forall P(t) \in V2 \\ & X(h) = -B(h) - c \times |V2| \\ & x_{u,v} \geq 0 \qquad \forall u, v \in V1 \cup V2\end{aligned} \tag{14}$$

This is the dual problem of (10) and as such the $r(v)'s$ in (12) are now the implicit Lagrange multipliers of this formulation. The physical meaning of x_{uv} is the amount of flow on edge $e(u,v)$ and this flow has a cost w_{uv} for each unit flow on $e(u,v)$. The constraints set the demand of each type of node in the retiming graph and ensure the amount of flow on each edge is non-negative. For example, the specific $B(v)$ and $X(v)$ values of each node in our sample circuit are shown in Fig. 4. Each $e(u,v)$ in Fig. 5 represents the variable x_{uv}. The constraint is that the total flow in to a node minus the total flow out of a node must equal its demand. Using node $G3$ as an example, $x_{I1,G3} - x_{G3,G4} - x_{G3,G6} = -B(G3) = 0$. Given the demand of each node and flow cost of each edge, the Network Simplex Solver returns the corresponding flow x_{uv} of each edge and the $r(v)$ of each node.

Notice that the integer constraint on $r(v)$ is relaxed because the network simplex algorithm [25] guarantees $r(v)$ will be integral when $w_{u,v}$ are integral. In addition, the upper and lower bounds of each $r(v)$ are not considered in min cost network flow problem because we use the trick proposed in [24] to add extra edges to the host node which implicitly enforce the upper and lower bounds.

V. Virtual Library Approach

An alternative approach is to create a virtual library (VL) that enables the synthesis tools to automatically select between non-error-detecting and error-detecting latches. Authors in [17] used a similar VL approach to resynthesize resilient circuits with resilient-aware area optimizations and demonstrated that, while not as powerful as dedicated algorithms for resiliency, the VL approach is promising. In this paper we apply and extend this approach to retiming resilient circuits and compare the best variant to our custom graph-based algorithm.

In the VL approach, each latch in the cell library is augmented with two new versions, forming three distinct groups of latches. The latches in the first group capture the usage of non-error-detecting latches by extending their setup times to include the resiliency window. In other words, the transition time from any startpoint to the latch must be no greater than $\phi_1 + \gamma_1$. The second group of latches are used to capture the usage of error-detecting latches with their area enlarged by the factor c, to represent the EDL overhead. Their arrival times can be between $\phi_1 + \gamma_1$ and $\phi_1 + \gamma_1 + \phi_1$. The third group consists of normal latches without setup times or area modifications from the standard-cell library. For synthesis, all latches in non-error-detecting pipeline stages are mapped to normal latches in the third group, while we constrain the synthesis tool to use only the first and second latch groups in the error-detecting pipeline stages.

Since the library enables the tool to understand the area overhead of EDL and the timing constraints of non-error-detecting latches, it can, in principle, efficiently optimize the total area of the design during retiming. In practice, however, synthesis tools are typically not designed to choose between latches with disparate trade-offs between area and setup time. The work in [17] suggests that the optimization results are impacted by multiple factors, and, in particular, the initial cell types of the latches. To explore this, we evaluated three different variants of the VL retiming approach. First, we map the latches in error-detecting pipeline stages to be initially all error-detecting latches and name this variant as E-type Virtual Library Based Resilient Aware Retiming, i.e. EVL-RAR. Secondly, we set all latches in error-detecting stages to be non-error-detecting latches and call this variant N-type Virtual Library Based RAR, i.e. NVL-RAR. Thirdly, we map the master latches to a mix of error-detecting/non-error-detecting latches based on timing and call this variant R-type Virtual Library Based RAR, i.e. RVL-RAR.

In addition, the work in [17] found that it was sometimes necessary to fix timing violation after resynthesis by manually switching some non-error-detecting latches to error-detecting latches. Similarly, we observed many error-detecting latches after retiming actually meet the increased setup time of non-error-detecting latches and thus could be replaced with non-error-detecting latches, saving area. We have thus evaluated adding a post-retiming step for just this purpose.

TABLE I: Circuit information of original flop-based designs

Circuit	P (ns)	flop #	NCE #	Run-time	Area
s1196	0.4	32	6	161	376.18
s1238	0.5	32	4	160	334.89
s1423	0.6	91	54	161	559.9
s1488	0.4	14	6	171	264.38
s5378	0.5	198	55	166	1149.42
s9234	0.5	160	61	168	893.36
s13207	0.5	502	188	179	2670.28
s15850	0.8	524	174	178	2980.52
s35932	1	1763	288	222	9681.35
s38417	1	1494	213	224	8635.73
s38584	0.7	1271	632	220	8100.11
Plasma	2.1	1652	217	208	10371.2
average	0.75	644	158	185	3834.78

Our experimental results, described in detail in Section VI, show that RVL-RAR outperforms the other two approaches. Compared to base retiming, RVL-RAR yielding an average area improvements of up to 9.6%. These improvements are somewhat larger than observed for VL-based resynthesis [17], possibly because retiming occurs earlier in the flow than resynthesis [17] and thus has more opportunities for optimization.

As mentioned earlier, we generally add a constraint to the commercial synthesis tool to ensure the master latches are not retimed to avoid issues with changing the circuit's initial state. Interestingly, the VL-based approach can trivially be extended to support the simultaneous moving of master and slave latches by removing this constraint. More precisely, releasing the "do-not-retime" constraints on the master latches enables the commercial tools to optimize the positions for both master and slave latches. This is in contrast to the proposed graph based approach which is dependent upon this constraint to properly annotate the retiming graph. This difference may be important because the added flexibility of retiming the master latches might lead to lower area overhead. For this reason, we will evaluate this extension in our experimental results, Section VI-E.

VI. Experimental Results

In this section, we compare the results from two proposed approaches with results from traditional resilient unaware retiming approach on latch-based ISCAS89 benchmark circuits and Plasma, a 3-stage OpenCore MIPS CPU using a commercial FDSOI 28nm cell library. All experiments were run on an Intel Xeon E5-2450 v2 CPU with 32GB of RAM.

A. Circuit Information

The benchmark circuits are all flip-flop based designs. For each circuit, the maximum delay through combinational logic, $P = \Pi + \phi_1$, is set so that the initial number of near-critical end-points is reasonable. Table I shows the maximum delay between stages P, the number of flops and, the number of near-critical end-points (NCE) in the original flop-based designs.

To generate two-phase latch-based designs, each flip-flop is split into master and slave latches. As described in Section III, we keep the master latches fixed and reposition only the slave latches. The resiliency window ϕ_1 is modeled as 30% of the max delay between detecting pipeline stages and thus set to $0.3P$ for each circuit. We set $\gamma_1 = 0$, $\gamma_2 = 0.05P$, and $\phi_2 = 0.35P$, hence Π is $0.7P$ and $Pi + \phi_1$ is P. As Section

TABLE II: Total area comparison between gate-based and path-based delay G-RAR

Circuit	Low Overhead			Medium Overhead			High Overhead		
	Gate	Path	Impr(%)	Gate	Path	Impr(%)	Gate	Path	Impr(%)
s1196	488.28	387.34	20.67	487.41	398.34	18.27	529.12	420.34	20.56
s1238	336.80	324.57	3.63	325.92	330.57	-1.43	412.90	334.00	19.11
s1423	561.27	562.61	-0.24	544.27	581.28	-6.80	544.27	573.08	-5.29
s1488	293.12	284.75	2.86	519.32	290.75	44.01	551.32	307.48	44.23
s5378	1137.42	1101.90	3.12	1100.78	1098.09	0.24	1112.70	1115.15	-0.22
s9234	972.73	893.26	8.17	922.24	883.36	4.22	922.24	893.03	3.17
s13207	2506.39	2353.35	6.11	2457.47	2365.18	3.76	2466.93	2367.14	4.05
s15850	3016.86	2807.43	6.94	2945.43	2897.29	1.63	2950.33	2907.41	1.45
s35932	8832.46	8795.50	0.42	8797.46	8795.50	0.02	8797.46	8795.50	0.02
s38417	8007.83	7855.14	1.91	7970.85	7925.81	0.57	7941.64	7951.10	-0.12
s38584	7751.46	7665.01	1.12	7716.26	7684.11	0.42	7725.56	7677.25	0.63
Plasma	11041.38	10599.51	4.00	10967.51	10598.86	3.36	10986.13	10598.70	3.53
average	3745.50	3635.86	4.89	3729.58	3654.09	5.69	3745.05	3661.68	7.59

TABLE III: Area comparison of virtual library approaches

Circuit	Low Overhead			Medium Overhead			High Overhead		
	NVL	EVL	RVL	NVL	EVL	RVL	NVL	EVL	RVL
s1196	509.35	408.64	492.54	501.02	415.10	474.42	519.30	424.66	469.69
s1238	392.82	355.39	389.72	397.56	360.39	391.68	396.58	370.39	388.42
s1423	570.22	604.21	560.10	570.22	651.21	568.92	570.22	751.21	561.90
s1488	391.68	320.81	323.91	400.66	326.81	329.75	382.87	338.98	341.91
s5378	1145.66	1146.24	1141.22	1144.68	1205.56	1151.83	1137.01	1315.56	1149.26
s9234	920.61	941.64	925.02	921.59	983.64	950.64	934.65	1043.64	929.26
s13207	2567.79	2521.32	2465.32	2553.26	2651.20	2485.29	2532.05	2943.69	2473.54
s15850	3095.74	3045.04	3136.21	3122.02	3169.20	3248.99	3123.81	3528.46	3243.27
s35932	9223.90	9137.98	9230.10	9223.90	9638.61	9225.21	9223.90	13922.69	9227.33
s38417	8059.80	8066.65	8066.65	8071.55	8055.06	8055.06	8069.10	8068.93	8068.93
s38584	7744.98	8545.05	7729.48	7734.37	9322.56	7740.25	7739.27	11161.45	7728.66
Plasma	10732.52	11239.00	10759.45	10732.52	11263.90	10755.53	10732.52	11323.18	10753.08
average	3779.59	3861.00	3768.31	3781.11	4003.60	3781.46	3780.11	4599.40	3777.94

II-A describes, master latches with arrival times larger than Π must be error-detecting.

There are different proposed EDL designs in the literature with different trade-offs in power, area, delay, and robustness. For this reason we report all our experimental results with a range of EDL overheads, similar to [12] that spans from 50% of the area of a latch to $2X$ the area of a latch. In particular, we tested the approaches with the EDL cost c set to low ($c = 0.5$), medium ($c = 1.0$), and high ($c = 2.0$).

B. Graph-Based Approach

For G-RAR, we read in the netlist and timing of synthesized gate-level circuits, generate the min cost network flow graph of (14) with a custom python program, and call the *Gurobi Optimization* tool to solve the network simplex problem.

To compute the forward $D^f(u)$ in Eq. 5, we queried the commercial synthesis tool for the latest arrival time of any fanout of u. Similarly, we identified the backward delays $D^b(v, t)$ by reporting the transition time of the longest combinational path from the fanout of v to the master latch endpoint t. Compared to previous work [16], which computes the path timing by adding up maximum cell delays for all gates through that path, this per-path delay model is more accurate as it applies commercial-grade pin-to-pin delay model while taking into account only valid combinations of rise and fall delays. This implementation is also simple and computationally tractable for moderately-sized circuits, as it leverages the advanced timing engines implemented in these commercial synthesis tools.

We read in the results from the network simplex solver and reposition the slave latches accordingly while also identifying which master latches should be error-detecting. Repositioning the slave latches sometimes causes minor timing violations due to the change in driving strengths and capacitive loads. To fix these issues we create a set max-delay constraints for paths ending with non-error-detecting masters and run an incremental compile step in which we allow only sizing of gates. This step resolves any introduced timing violations.

To quantify the benefit of the path-based delay model, we compared it to the gate-delay-based G-RAR [16] algorithm that we re-implemented using the commercial logic synthesis tool. The gate delay model is conservative and can negatively impact the region calculations. In particular, nodes that could be in the retiming region V_r can be placed in the V_m or V_n region, reducing the flexibility of the position of the slave latches. In addition, some non-critical endpoints can be assigned EDL overheads because of unnecessarily pessimistic arrival times. The experimental result, as reported in Table II, shows that compared to the gate-based delay model, our path-based delay model reduces total area, including both sequential and combinational logic area, by 4.9%, 5.7%, and 7.6% with low, medium, and high EDL overheads, respectively.

A quantitative comparison of our path-based G-RAR approach to both the virtual library and base retiming approaches will be reported below in Section VI-D.

C. Virtual Library Approach

To create the virtual library, latches added to represent EDL gates are defined with an extra area overhead c and those representing non-EDL gates are assigned a larger setup time to mimic the resiliency window.

During an initial synthesis run, all latches are only mapped to the original standard library cells. Then the virtual library is loaded and the initial types of latches in error-detecting stages, i.e. master latches in our paper, are set based on the three retiming variants discussed in Section V. In RVL-RAR, all near-critical end-points in error-detecting stages are mapped to error-detecting latches while other latches within the error-detecting stage are left regular. In contrast, EVL-RAR makes all master latches error-detecting and NVL-RAR makes them all non-error-detecting, regardless of criticality. For each variant, we applied retiming followed by a size-only incremental compile for all examples in the ISCAS89 benchmark suite as well as an open-source CPU, Plasma. As mentioned in V, the commercial synthesis tool is not designed to robustly choose between sequential gates with such a wide range of parameters during retiming. Thus, we evaluated including a post-retiming step to manually switch unnecessary error-detecting latches to non-error-detecting. For example, considering RVL-RAR with high overhead, the post-retiming step increases the average area improvement from -0.36% to 9.6%. We have therefore added this step into all virtual library variants and all subsequent results are reported after this step.

Table III provides a detailed area comparison among the three VL variants. Examining the average total area across the benchmarks, the results show the RVL-RAR variant outperforms EVL-RAR in all three of the overhead cases. It also either matches or outperforms NVL-RAR. Based on these results, only the RVL-RAR variant will be considered for comparing the virtual library method to the other two retiming approaches.

D. Comparison of Retiming Approaches

We evaluate the efficiency of our methods by comparing it to the traditional resiliency-unaware retiming approach, referred to as base-retiming, using a leading commercial logic synthesis tool. We load the standard cell library and use the built-in retiming command to retime the design subject to worst-case timing constraints. Master latches whose input arrival times fall in the resiliency window are then replaced with error-detecting counterparts. The other master latches remain as normal, non-error-detecting latches. This section compares the resulting total area, number of slave latches and error-detecting master latches, run-time, and error-rate with those obtained with the RVL-RAR and G-RAR approaches.

Table IV reports the sequential logic area of the three approaches. This area includes the retimed slave latches, original master latches, and the overhead of any error-detecting master latches, but not the combinational logical. The results show that G-RAR produced the lowest sequential logic area, saving $29.6\%, 23.9\%$, and 20.4% compared to the base retiming method for the high, medium, and low EDL overheads, respectively. These improvements exhibit the same trend as reported in [16] but are generally larger possibly because of the adoption of the path-based delay model, as suggested by our earlier results reported in Section VI-B.

Table V reports the total area savings of our path-based-timing G-RAR approach with both the VL and Base-retiming approaches. Compared to base retiming, for high, medium, and low EDL overheads, G-RAR achieves total area reduction of $14.7\%, 9.5\%,$ and 7.0%, respectively. RVL-RAR performs somewhat worse than G-RAR, yielding total area improvements over Base-Retiming of 9.6%, 2.9%, and -0.3%. We can thus conclude that G-RAR outperforms RVL-RAR by an average of 5.1%. This result quantifies the benefit that the G-RAR approach yields by more tightly coupling retiming decisions with the binary decision of making a latch error-detecting or not than more traditional algorithms. In particular, the VL-based approach relies on the typical commercial CAD flows that make latch-type decision in separate optimization steps that are decoupled from retiming. Our results suggest that this decoupling is particularly problematic when the choice of latch type yields a much larger tradeoff between area and setup times than is typical. That said, the G-RAR and VL approaches are not in conflict with each other and, with minor modifications, both can be applied to real circuits in any other. In particular, VL can be trivially applied after G-RAR but to apply G-RAR after VL, we must modify the upper and bounds of the retiming variables to account for the difference in original positions of the slave latches.

We can also compare these results to the non-resilient flop-based designs whose area is reported in Table I. Somewhat surpisingly, the results suggest that we can generate a latch-based resilient design with on average no area increase. This happens because the latches in our library are very efficient. In particular, the average area of our latch is 43% of the area of a flip-flop. Thus the pair of latches we use before retiming is smaller than the original flip-flops they replace. This coupled with the fact that retiming does not add many latches and produces relatively small number of error-detecting latches yields surprisingly low area. That said, this analysis does not consider the fact that our two-phase latch-based design requires the generation of two clock trees instead of one, which could introduce additional overhead during physical design.

When we compare our latch-based resilient designs to flop-based resilient designs the area comparison is even more favorable. In particular, we can estimate the area of a flop-based resilient design by adding the overhead to all near-critical-end-points of the original FF-based design. Our latch-based design are on average 12.4%, 18.2%, and 28.2% smaller for $c = 0.5$, 1, and, 2, respectively. That said, it is important to note that these results are library dependent and if the included latches were less efficient the results may be significantly different.

It may also be interesting to note that the relative increase in area is not large for the runs with larger values for EDL overhead c. This is likely because faced with a higher EDL cost, the algorithms compensate by paying a small area penalty to speed-up the combinational logic and avoid more EDLs.

Providing a bit more detail, Table VI reports the number of slave latches and error-detecting master latches after retiming achieved by the three different approaches. For circuits larger than s1196 and s1238, both of which have 32 flops originally, G-RAR typically assigns the least number of error-detecting master latches. With a higher EDL overhead, G-RAR tends to move latches further into the logic, even at the cost of adding slave latches, so that more master latches need not be error-detecting.

Table VII compares the run-time of the three different approaches. The run-time for G-RAR involves analyzing and reporting retiming paths by the logic synthesis tool, formulating and solving network flow problem with a simplex solver, and moving slave latches followed by a size-only incremental compile. The run-times for base retiming and virtual library methods includes only one call to the built-in retiming function within the logic synthesis tool. Table VII reports the CPU run-time of all these approaches using multi-threaded computation. The Base-Retiming method has the lowest average running time. Interestingly, the G-RAR approach is faster on average than RVL-RAR. This is because even though the VL approaches involve only one call to the built-in retiming function, the logic synthesis tool spends a relatively long time on the retiming and other optimization steps included in this call. In particular, latch-based retiming fails to complete on several larger circuits we tested. The G-RAR run-time, on the other hand, is dominated by the computation of backward delays, largely because the report timing function built-in to the commercial tools we are using can only compute each requested delay independently. In fact, the G-RAR network simplex optimization step takes, on average, less than 2% of the total computation time. Thus, using the simpler gate-delay based model proposed in [16], or a custom static timing analysis tool may enable optimization of significantly larger circuits.

Table VIII shows the resulting final error-rates obtained with the three methods. It shows that the error-rate is often reduced by both G-RAR and RVL-RAR, despite only targeting area.

TABLE IV: Comparison of Sequential Logic Area among Base-Retiming, RVL-RAR, and G-RAR

Circuit	Low Overhead					Medium Overhead					High Overhead				
	Base	RVL	Impr(%)	G	Impr(%)	Base	RVL	Impr(%)	G	Impr(%)	Base	RVL	Impr(%)	G	Impr(%)
s1196	192.21	193.39	-0.61	124.26	35.35	198.21	192.74	2.76	135.26	31.76	210.21	191.76	8.78	157.26	25.19
s1238	185.83	177.72	4.36	111.75	39.86	190.83	177.72	6.87	117.75	38.29	200.83	177.72	11.50	131.96	34.29
s1423	369.34	283.48	23.25	309.49	16.20	409.34	283.48	30.75	316.41	22.70	489.34	283.48	42.07	317.34	35.15
s1488	107.51	110.45	-2.73	51.70	51.92	113.51	116.45	-2.59	57.70	49.17	125.51	128.45	-2.34	72.63	42.13
s5378	741.34	668.76	9.79	672.04	9.35	792.34	676.76	14.59	688.79	13.07	894.34	684.14	23.50	689.36	22.92
s9234	597.96	525.18	12.17	523.12	12.52	636.96	525.99	17.42	530.04	16.79	714.96	525.83	26.45	536.60	24.95
s13207	1841.84	1638.71	11.03	1593.00	13.51	2012.84	1643.18	18.37	1603.69	20.33	2354.84	1643.51	30.21	1604.84	31.85
s15850	1930.31	1797.16	6.90	1684.13	12.75	2031.31	1861.62	8.35	1678.02	17.39	2233.31	1873.54	16.11	1678.02	24.86
s35932	6097.12	6173.04	-1.25	5750.03	5.69	6425.12	6173.04	3.92	5750.03	10.51	7081.12	6173.04	12.82	5750.03	18.80
s38417	5520.90	4861.73	11.94	4751.57	13.93	5834.90	4861.40	16.68	4755.00	18.51	6462.90	4861.73	24.77	4755.00	26.43
s38584	5094.61	4232.76	16.92	4176.12	18.03	5808.61	4231.29	27.15	4176.12	28.10	7236.61	4231.61	41.52	4176.12	42.29
Plasma	5862.85	5113.71	12.78	4933.05	15.86	6147.85	5113.71	16.82	4928.64	19.83	6717.85	5113.71	23.88	4928.64	26.63
average	2378.48	2148.01	8.71	2056.69	20.41	2550.15	2154.78	13.42	2061.45	23.87	2893.48	2157.38	21.61	2066.48	29.62

TABLE V: Comparison of Total Area among Base-Retiming, RVL-RAR, and G-RAR

Circuit	Low Overhead					Medium Overhead					High Overhead				
	Base	RVL	Impr(%)	G	Impr(%)	Base	RVL	Impr(%)	G	Impr(%)	Base	RVL	Impr(%)	G	Impr(%)
s1196	411.72	492.54	-19.63	387.34	5.92	417.72	474.42	-13.58	398.34	8.67	429.72	469.69	-9.30	420.34	2.18
s1238	355.39	389.72	-9.66	324.57	8.67	360.39	391.68	-8.68	330.57	8.28	370.39	388.42	-4.87	334.00	9.83
s1423	595.21	560.10	4.24	562.61	5.48	635.21	568.92	10.44	581.28	8.49	715.21	561.90	21.44	573.08	19.87
s1488	323.91	323.91	0.00	284.75	12.09	329.91	329.75	0.05	290.75	11.87	341.91	341.91	0.00	307.48	10.07
s5378	1145.91	1141.22	0.41	1101.90	3.84	1196.91	1151.83	3.77	1098.09	8.26	1298.91	1149.26	11.52	1115.15	14.15
s9234	934.64	925.02	1.03	893.26	4.43	973.64	950.64	2.36	883.36	9.27	1051.64	929.26	11.64	893.03	15.08
s13207	2574.61	2465.32	4.24	2353.35	8.59	2745.61	2485.29	9.48	2365.18	13.86	3087.61	2473.54	19.89	2367.14	23.33
s15850	3017.71	3136.21	-3.93	2807.43	6.97	3118.71	3248.99	-4.18	2897.29	7.10	3320.71	3243.27	2.33	2907.41	12.71
s35932	9092.33	9230.10	-1.52	8795.50	3.26	9420.33	9225.21	2.07	8795.50	6.63	10076.33	9227.33	8.43	8795.50	12.71
s38417	8555.93	8066.65	5.72	7855.14	8.19	8869.93	8055.06	9.19	7925.81	10.64	9497.93	8068.93	15.05	7951.10	16.29
s38584	8501.09	7729.48	9.08	7665.01	9.83	9215.09	7740.25	16.00	7684.11	16.61	10643.09	7728.66	27.38	7677.25	27.87
Plasma	11311.28	10759.45	4.88	10599.51	6.29	11596.28	10755.53	7.25	10598.86	8.60	12166.28	10753.08	11.62	10598.70	12.88
average	3901.64	3768.31	-0.29	3635.86	6.96	4073.31	3781.46	2.85	3654.09	9.52	4416.64	3777.94	9.59	3661.68	14.73

TABLE VI: Number of slave and error-detecting master latches decided by the three approaches

Circuit	Approach	Low Overhead		Medium Overhead		High Overhead	
		slave #	EDL #	slave #	EDL #	slave #	EDL #
s1196	Base	88	6	88	6	88	6
	RVL	81	7	81	6	81	7
	G	32	11	32	11	32	11
s1238	Base	84	5	84	5	84	5
	RVL	73	4	73	4	73	4
	G	31	6	31	6	32	5
s1423	Base	113	40	113	40	113	40
	RVL	79	54	79	54	79	54
	G	97	3	97	3	100	1
s1488	Base	54	6	54	6	54	6
	RVL	54	6	54	6	54	6
	G	14	6	14	6	16	6
s5378	Base	228	51	228	51	228	51
	RVL	201	55	201	55	201	54
	G	184	46	222	2	224	0
s9234	Base	185	39	185	39	185	39
	RVL	157	61	157	61	157	61
	G	156	3	156	3	161	0
s13207	Base	524	171	524	171	524	171
	RVL	492	85	492	82	492	84
	G	454	16	462	6	464	5
s15850	Base	605	101	605	101	605	101
	RVL	556	32	556	11	556	23
	G	489	11	495	0	495	0
s35932	Base	1773	328	1773	328	1773	328
	RVL	2048	219	2048	230	2048	257
	G	1760	0	1760	0	1760	0
s38417	Base	1719	314	1719	314	1719	314
	RVL	1483	213	1483	213	1483	213
	G	1409	0	1409	0	1409	0
s38584	Base	1429	714	1429	714	1429	714
	RVL	1325	392	1325	498	1325	509
	G	1288	0	1288	0	1288	0
Plasma	Base	1984	285	1984	285	1984	285
	RVL	1690	186	1690	184	1690	185
	G	1564	0	1564	0	1564	0

TABLE VII: Run-time (sec) comparison

Circuit	Low Overhead			Medium Overhead			High Overhead		
	Base	RVL	G	Base	RVL	G	Base	RVL	G
s1196	185	183	183	185	185	186	185	188	186
s1238	180	180	183	180	179	182	180	185	183
s1423	201	185	179	201	186	187	201	199	189
s1488	183	179	181	183	184	183	183	184	179
s5378	188	185	186	188	185	182	188	185	180
s9234	188	186	192	188	186	184	188	194	181
s13207	192	201	210	192	201	213	192	203	213
s15850	197	208	271	197	206	286	197	206	277
s35932	238	362	563	238	366	412	238	382	408
s38417	230	345	315	230	366	379	230	350	372
s38584	230	454	430	230	488	436	230	553	412
Plasma	347	3663	2559	347	3918	2799	347	3718	2790

TABLE VIII: Error-rate (%) comparison

Circuit	Low Overhead			Medium Overhead			High Overhead		
	Base	RVL	G	Base	RVL	G	Base	RVL	G
s1196	20.24	0	62.21	20.24	0	62.21	20.24	0	62.21
s1238	8.71	0	19.38	8.71	0	19.38	8.71	0	20.32
s1423	46.78	0	0	46.78	0	0.01	46.78	0	0.01
s1488	23.47	23.48	24.81	23.47	23.44	24.81	23.47	23.48	24.81
s5378	11.87	0	70.34	11.87	0	0.75	11.87	0	0.01
s9234	0.37	0	0.01	0.37	0	0.01	0.37	0	0
s13207	74.95	0	1.34	74.95	0.01	1.24	74.95	0	1.26
s15850	2.13	0	0	2.13	0	0.01	2.13	0	0.01
s35932	60.67	0	0	60.67	0	0	60.67	0	0
s38417	3.07	0	0.01	3.07	0	0.01	3.07	0	0.01
s38584	0	0	0	0	0	0	0	0	0
Plasma	0	0	0	0	0	0	0	0	0
average	21.02	1.96	14.84	21.02	1.95	9.04	21.02	1.96	9.05

In particular, G-RAR yields on average half the error-rate of base retiming approach. Interestingly, RVL-RAR achieves even lower error-rates. There are two different reasons this happens. In two circuits (s13207 and s5378) the data path does not trigger errors in our random-input-based simulation. In many other circuits, all near-critical end-points become non-near-critical after retiming. Interestingly, for those circuits, even though the max delay to the endpoint does not fall in the resiliency window, the synthesis tool fails to switch the latches to non-error-detecting. This indicates that commercial synthesis tool tends favor increasing combinational logic area to avoid the resiliency window but sometimes fails to actually swap the sequential cells if the resiliency window is avoided. Fortunately, our post-retiming swapping routine detects these situations and manually switches the latches. These results also suggest that with a modest area increase of, on average 5%, error-rates can be further reduced, sometimes to 0.

TABLE IX: Total area comparison between fixed-master RVL-RAR and movable-master RVL-RAR

Circuit	Low Overhead			Medium Overhead			High Overhead		
	fixed	movable	diff (%)	fixed	movable	diff (%)	fixed	movable	diff (%)
s1196	492.54	475.91	3.38	474.42	462.55	2.50	469.69	468.22	0.31
s1238	389.72	416.65	-6.91	391.68	407.84	-4.12	388.42	403.43	-3.86
s1423	560.10	565.98	-1.05	568.92	544.76	4.25	561.90	546.23	2.79
s1488	323.91	323.75	0.05	329.75	329.75	0.00	341.91	341.75	0.05
s5378	1141.22	1136.69	0.40	1151.83	1132.93	1.64	1149.26	1120.37	2.51
s9234	925.02	960.43	-3.83	950.64	968.10	-1.84	929.26	934.16	-0.53
s13207	2465.32	2489.62	-0.99	2485.29	2459.18	1.05	2473.54	2636.74	-6.60
s15850	3136.21	3069.14	2.14	3248.99	3316.22	-2.07	3243.27	3138.01	3.25
s35932	9230.10	9230.43	0.00	9225.21	9225.04	0.00	9227.33	9227.82	-0.01
s38417	8066.65	8077.91	-0.14	8055.06	8070.57	-0.19	8068.93	8079.22	-0.13
s38584	7729.48	7740.41	-0.14	7740.25	7756.57	-0.21	7728.66	7746.29	-0.23
Plasma	10759.45	10938.98	-1.67	10755.53	10855.08	-0.93	10753.08	10847.74	-0.88
average			-0.73			0.01			-0.28

E. Retiming Master and Slave Latches

As discussed in Section V, the VL-based approach can easily be extended to supporting the simultaneous retiming of both master and slave latches. As shown in Table IX, this extra level of flexibility does not always lead to smaller circuits. Whereas the flexibility can lead to up to a 4% improvement in area, on average, the results show little to no gain. This also suggests that restricting retiming to slave latches, as is done in the graph-based approaches, may not overly constrain the problem.

VII. Conclusions

This paper presents two methods to minimize the area of latch-based resilient designs through retiming. The first modifies the traditional retiming graph with extra edges and nodes to take the EDL overheads into account for two-phase latch-based resilient circuit design. We mapped the ILP problem to a min cost network flow problem and solved it with a simplex network solver. The second is a virtual library method which enables commercial synthesis tools to compensate for additional overheads in resilient designs by allowing synthesis to pick from two groups of modified lathes that reflect the tradeoff between higher area of error-resilient latches and increased setup time of normal latches in error-detecting stages. The virtual library approach is easy to incorporate into an existing CAD flow, taking only one call to the built-in retiming algorithm. Our experimental results show that our graph-based approach achieves the highest total area improvement of up to 15% compared to resiliency-unaware retiming and on average 5.1% higher total area improvement compared to the virtual library method. Moreover, our results show that both approaches are computationally tractable.

The results suggest that creating a specialized algorithm that targets retiming of resilient design can indeed lead to significant savings with little to no increase in complexity compared to existing commercial flows. Moreover, some savings can be achieved simply by augmenting existing cell libraries while achieving the optimum savings requires using a problem-specific optimization algorithm.

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," *Proc. of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb 2010.

[3] 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.

[4] B. Munger, D. Akeson, S. Arekapudi, T. Burd, H. R. Fair, J. Farrell, D. Johnson, G. Krishnan, H. McIntyre, E. McLellan *et al.*, "Carrizo: A high performance, energy efficient 28 nm APU," *IEEE JSCC*, vol. 51, no. 1, pp. 105–116, 2016.

[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. Vilar Calazans, and P. Beerel, "Blade – a timing violation resilient asynchronous template," in *ASYNC*, May 2015, pp. 21–28.

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

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

[11] 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.

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

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

[14] S. Kim and M. Seok, "Variation-tolerant, ultra-low-voltage μP with a low-overhead, within-a-cycle in-situ timing-error detection and correction technique," *IEEE JSSC*, vol. 50, no. 6, pp. 1478–1490, Jun 2015.

[15] V. Singhal, S. Malik, and R. K. Brayton, "The case for retiming with explicit reset circuitry," in *ICCAD*, 1997, pp. 618–625.

[16] H.-L. Wang, M. Zhang, and P. A. Beerel, "Retiming of two-phase latch-based resilient circuits," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 70.

[17] H.-H. Huang, H. Cheng, C. Chu, and P. A. Beerel, "Area optimization of timing resilient designs using resynthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

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

[19] "Plasma CPU," http://opencores.org/project,plasma, available: 2014.

[20] M. C. Papaefthymiou and K. H. Randall, "Tim: A timing package for two-phase, level-clocked circuitry," in *DAC*. ACM, 1993, pp. 497–502.

[21] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," Digital (Palo Alto, CA US ; Cambridge, MA US). Systems research center, Tech. Rep. D-SRC-13, 1986.

[22] S. S. Sapatnekar and R. B. Deokar, "Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits," *IEEE Trans. on CAD*, vol. 15, no. 10, pp. 1237–1248, 1996.

[23] N. Maheshwari and S. S. Sapatnekar, "Efficient minarea retiming of large level-clocked circuits," in *DATE*, Feb 1998, pp. 840–845.

[24] N. Maheshwari and S. Sapatnekar, "Efficient retiming of large circuits," *IEEE Trans. on VLSI*, vol. 6, no. 1, pp. 74–83, 1998.

[25] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice hall, 1993.