Retiming for Soft Error Minimization Under Error-Latching Window Constraints

Yinghai Lu
Analog Mixed Signal Group
Synopsys Inc, USA

Hai Zhou
Electrical Engineering and Computer Science
Northwestern University, USA

***Abstract*—Soft error has become a critical reliability issue in nano-scale integrated circuits, especially in sequential circuits where a latched error will be propagated for many cycles and affect many outputs at different time. Retiming is a structural operation that relocates registers in a circuit without changing its functionality. In this paper, the effect of retiming on soft error rate (SER) of a sequential circuit is investigated considering both logic masking and timing masking. A minimum observability retiming problem under error-latching window constraints is formulated to reduce the SER of the circuit. And an efficient algorithm is proposed to solve the problem optimally. Experimental results show on average a 32.7% reduction on SER from the original circuits and a 15% improvement over the existing method.**

I. Introduction

Soft error, also known as single-event upsets (SEU), caused by radiation-induced charged particles such as alpha particles, energized neutrons and protons created by cosmic ray, is becoming one of the dominant reliability concerns in logic circuitry design [1]. Technology scaling has contributed to an increasing sensitivity to naturally-occurring radiation and the consequent soft error rates of CMOS circuits. Moreover, more aggressive energy scaling makes the soft error issue persist in the further technology nodes [2]. Therefore, it is desirable to mitigate the impact of soft errors during the IC design process.

Researchers have identified three mechanisms that will mask the soft errors during SEU propagation in logic circuits [3]: *electrical masking* occurs when SEUs are attenuated before being latched because of insufficient glitch duration or amplitude; *logic masking* occurs when SEUs stop propagating due to the lack of a sensitized path to primary outputs or latches; and *timing masking* occurs when SEUs arrive at the registers during a non-latching portion of the clock. Various optimization techniques have been proposed to reduce the SER of the circuit by enhancing some or all of the above three masking mechanisms. Gate sizing [4] and voltage scaling/assignment [5], [6] are proposed to harden soft-error vulnerable components in the circuit. Such methods trade area/power cost to reduce the SER of the circuit. In [7], [8], partial duplication techniques are proposed to increase the chances of logic masking, which may also incur significant area overhead. Other approaches, such as rewiring [9], [10] and resynthesis [11], [12] can improve the SER with less overhead.

All above studies focus on combinational circuits. Recently, the soft error issue in sequential circuits has also drawn increasing attention because of the following reasons. First, registers, just as the combinational gates, are also susceptible to single event upsets. Secondly, the SEUs latched in a register is likely to propagate through the circuits for multiple clock cycles due to the existence of feedback loops in the sequential circuits. Therefore, it is even more important to address the soft error problem in sequential circuits. One intuitive approach is to harden the registers as one does for the gates in combinational circuits [13]. Other options include increasing the probability of timing masking by register sizing [14] and gate relocation [15]. In [16], the authors proposed a clock skew scheduling based approach to drastically reduce the multi-bit upsets of the circuit. In [17], the authors use retiming technique to minimize the soft-error susceptibility of the registers in the sequential circuits.

Retiming [18] is one of the most powerful sequential transformations which relocates the registers in a circuit while preserving its functionality. Previous work on retiming has been focused on minimizing the clock period or total number of registers. The pioneer work in [17] shows the power of retiming in SER optimization. However, the approach proposed in [17] suffers from two shortcomings. First, the overall SER of a sequential circuit is affected by logic masking as well as timing masking. Focusing on logic masking only, the approach in [17] may lead to larger soft-error-lathing windows and potentially degraded SER of the circuit. Secondly, the algorithms proposed in [17] are mixed-integer linear programs with $\Theta(|V|^2)$ explicit path constraints, where $|V|$ is the number of combinational gates in circuit. The computational and memory cost to solve such problem is very expensive [19], [20], especially in large designs.

In this paper, we propose an efficient retiming algorithm to optimize the SER of the sequential circuit considering both logic and timing masking. The contribution of our work is as follows. First, We formulate the retiming problem to consider its impact on both the logic observability of registers and on the change of error-latching windows of combinational gates. Next, we design an optimal algorithm to solve the proposed retiming problem. The regular forest from [20] is modified to maintain the constraints among the circuit with linear storage. The resultant algorithm is efficient in terms of both performance and memory usage, as is verified by the experimental results.

The rest of the paper is organized as follows. In Section II, preliminary background of soft-error analysis on sequential circuit is provided. In Section III, we discuss the optimization power of retiming on SER reduction and the retiming for soft-error optimization under error-latching window constraints problem is formally formulated. Section IV describes the design of optimal algorithm for the formulated retiming problem, where an important data structure called regular forest is also introduced. In Section V, we briefly discuss how to obtain a valid initial retiming for the proposed problem. Experimental results and conclusions are given in Section VI and Section VII, respectively.

II. Soft Errors in Sequential Circuits

A soft error occurs when a flip of the signal induced by a transient glitch at a gate in a logic circuit is propagated through a sensitized and observable path to the input of a register and is locked by the register during a specific interval of time around the clock switching. Due to the three masking mechanisms explained in Section I, only a small fraction of the soft errors will come into effect. Since electrical masking is related to the physical property of a gate and can be enhanced by gate hardening, we focus on logical and timing masking in this paper, which has a global impact on the sequential circuits. Later, we will show how retiming affects the two masking schemes of the circuits.

We adopt the logic simulation techniques [17] to quantify the impact of the logical masking and compute the error-latching window [15] to evaluate the impact of the timing masking on the circuit soft error rate. To trace the soft error propagation through

* This work is supported by the NSF under CCF-0811270 and CCF-1115550.

the multiple register stages in sequential circuits, we use time-frame expansion [17] during signal logic simulation.

A. Logic Masking and SER in Combinational Circuit

In [17] and [11], signal observability is used to evaluate the SER of the circuit under logic masking. The observability at the output of a gate g is expressed as

$$obs(g) = num\_ones(\mathcal{O}(g))/K,$$

where $\mathcal{O}(g)$ is the observability don't-care (ODC) mask of g, which is computed by logic simulation method introduced in [11], [21]. K is the length of signal sequences.

With the information of observability for each gate, the SER of a combinational circuit C can be expressed as

$$SER(C) = \sum_{g \in C} obs(g) \cdot err(g) \tag{1}$$

where $err(g)$ is the soft error rate of gate g.

B. Extension to Sequential Circuits

The above computation process for signal propagation breaks at the inputs of the registers or primary outputs. In a sequential circuit C_S, a soft error may propagate through the circuit for multiple cycles due to the existence of feedback loops. Besides, registers, like other gates in the combinational circuits are also susceptible to SEUs. Time-frame expansion [17] is used to simulate and compute the observability for every gate/register in a sequential circuit. In a n time-frame expansion, n time frames are allocated. During each time frame, signals are propagated through the registers into the next stage. After the n-th time frame simulation ends, we compute $\mathcal{O}(g)$ for every gate g in the circuit. Since the registers act like wires in the n-time-frame simulation, their observabilities are the same as those of the gates at their immediate inputs. Therefore, with the n-time-frame expansion technique, the SER of a sequential circuit C_S can be expressed as [17]:

$$\begin{aligned} SER(C_S, n) = &\sum_{g \in Comb(C_S)} obs(g, n) \cdot err(g) \\ &+ \sum_{r \in Reg(C_S)} obs(r, n) \cdot err(r) \end{aligned}$$

C. Timing Masking and Error-Latching Window

A transient flip of signal, even if it evades the logic masking and arrives at the input of the a register, will cause a SEU only when it is securely latched. For an edge-triggered D-type register, data is latched within the timing window $[\Phi - T_s, \Phi + T_h]$, where Φ is the clock period, T_s is the setup time and T_h is the hold time. Given a circuit C, the error-latching window [15] (ELW) of a gate $g \in C$ is defined as the time interval within which a transient glitch, if propagated to the registers, is latched and thus causes a soft-error. Note that the ELW of a gate may contain multiple disjoint intervals as is pointed out in [15]. The general form of the ELW of a gate g with l disjoint intervals is:

$$ELW_l(g) = [L_1, R_1] \cup [L_2, R_2] \cup \ldots \cup [L_l, R_l]. \tag{2}$$

The ELWs can be computed with a backward traverse starting from the inputs of the registers using the following relationship:

$$ELW(g) = \begin{cases} [\Phi - T_s, \Phi + T_h], & g \in RO \\ \bigcup\limits_{f \in fanout(g)} (ELW(f) - d(f)), & \text{else} \end{cases} \tag{3}$$

where $g \in RO$ indicates that g is either a primary output or is at the input of a register. $ELW(f) - d(f)$ is the operation that shifts the ELW of f by its delay $d(f)$. For a register, since it is treated as a wire in the time-frame expansion, its ELW can also be computed with (3).

The error-latching window models the window masking of the soft errors. Incorporating this, we arrive at the final equation for SER of a sequential circuit considering both logic and timing masking as

$$\begin{aligned} SER(C_S, n) = &\sum_{g \in Comb(C_S)} obs(g, n) \cdot err(g) \cdot \frac{|ELW(g)|}{\Phi} \\ &+ \sum_{r \in Reg(C_S)} obs(r, n) \cdot err(r) \cdot \frac{|ELW(r)|}{\Phi}, \end{aligned} \tag{4}$$

where $|ELW(g)|$ is sum of all the intervals in $ELW(g)$:

$$|ELW_l(g)| = \sum_{i=1}^{l} (R_i - L_i).$$

III. Problem Formulation

As the SER of a sequential circuit is summarized in (4), we can inspect the impact of retiming on SER by checking how it changes the terms in (4). For the sake of clarity, we drop the time-frame variable n in (4) in the later discussion.

A. Retiming Graph

First, we introduce the retiming graph. A sequential circuit can be modeled by a direct graph $G = (V, E)$ whose vertices V represent the combinational gates and edges E represent the signals between the gates [18]. A non-negative label $d : V \to \mathbb{R}^*$ is associated with each vertex to give the gate delay. And a non-negative integer weight $w : E \to \mathbb{N}$ on each edge represents the number of registers on it. A special host vertex, the edges from the host to the primary inputs and edges from the primary outputs to the host are introduced to G to represent interface with the environment.

A retiming is a relocation of the registers in the sequential circuit to achieve certain objectives such as clock period or area minimization, while preserving its functionality. To present such relocation, conventionally, Leiserson and Saxe [18] introduced a vertex label $r : V \to \mathbb{Z}$ to give the number of registers moved backward over each gate from fanouts to fanins. Given r, the number of registers on an edge (u, v) after retiming is $w_r(u, v) = w(u, v) + r(v) - r(u)$.

B. The Impact of Retiming on SER

Suppose we have retimed a sequential circuit C_S with label r. Since in a time-frame expanded SER computation, the registers are treated as wires, the observability of the combinational gates will not change after retiming, i.e. the terms $obs(g)$'s in (4) keep constant for any retiming. However, since the observability of a register equals that of its driven combinational gate, a retiming of the circuit will relocate the registers to the outputs of different combinational gates. Using the notions of retiming graph, the sum of observability of registers can be expressed as:

$$\sum_{r \in Reg(C_S)} obs(r) = \sum_{(u,v) \in E} (obs(u) \cdot w_r(u, v)). \tag{5}$$

Not only does a retiming of circuit affect the observability of the registers, it also changes the lengths of combinational paths. Timing information of the combinational gates will be updated and according to (3), the ELWs of the gates will also change. It is possible that a relocation of one register on one hand increases its observability while on the other hand increases the ELWs of the gates at its fanin cone. Figure 1 gives an example of such scenario. In Figure 1, each combinational gate is represented by a circle with its delay/observability annotated inside. A register is moved out of F to reduce its observability from 0.6 to 0.4. However, such a relocation increases the size of ELWs of A and B by 1. It can be verified that the resultant circuit has a worse SER.

In summary, a retiming of a sequential circuit will not change the observability of the combinational gates, but it can increase that of the registers. Meanwhile, the same retiming also changes the ELWs of the gates in the circuit. As is suggested by (4), it is inadequate to retime a circuit for SER reduction regardless of its impact on ELWs, as is the approach in [17].

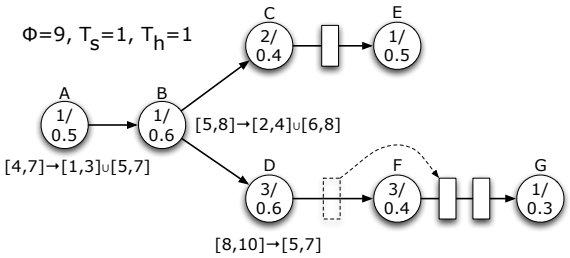

Fig. 1. An example of impact of retiming on ELW.

C. Mathematical Formulation

To balance the trade-off between timing masking and logic masking, we formulate our retiming problem for register observability minimization under ELW constraints.

First, we model the propagation of ELWs through combinational paths with simpler mathematical expression since the union operation in (3) is difficult to handle in mathematical programs. We introduce two labels for each vertex in G, $L : V \to \mathbb{R}$ and $R : V \to \mathbb{R}$. L and R are computed by the longest path and shortest path constraints respectively.

$$\begin{array}{lll} w_r(u,v) > 0 \vee u \in PO & \Rightarrow & L(u) = \Phi - T_s, \\ & & R(u) = \Phi + T_h \\ w_r(u,v) = 0 & \Rightarrow & L(u) \leq L(v) - d(v), \\ & & R(u) \geq R(v) - d(v) \end{array} \quad (6)$$

Note that the propagation of L and R is in the reverse order of the signal propagation. The following theorem shows that $R(v) - L(v)$ bounds the size of ELW of $v \in V$.

Theorem 1: With L and R defined in (6), for each $v \in V$, the following properties holds:

1) $R(v) \geq L(v)$.
2) $L(v)$ and $R(v)$ are the left and right boundaries of the ELW at the output of v, i.e. $L(v) = L_1$ and $R(v) = R_l$ for $ELW_l(v)$ defined in (2).

Next, we incorporate the ELW constraints in our problem formulation with labels L and R. Let $b : V \to \mathbb{R}$ represents the reduction of the observability of the registers if they are moved from the inputs of a gate to its outputs. For each gate $v \in V$, $b(v)$ can be pre-computed as $b(v) = K(\sum_{(u,v)\in E} obs(u) - \sum_{(v,x)\in E} obs(x))$. The scaling of K makes $b(v)$ an integer. The ELW-constrained retiming problem for sequential observability minimization is then formulated as follows:

Problem 1 (Min-Obs Retiming with ELW Constraints):

$$\max \sum_{v\in V} -b(v)r(v)$$

subject to

$$\begin{array}{ll} P0: & \forall (u,v) \in E : w_r(u,v) \geq 0 \\ P1: & \forall v \in V : L(v) > d(v) \\ P2: & \forall (u,v) \in E : w_r(u,v) > 0 \\ & \wedge \Phi + T_h - R(v) > R_{min} \\ P3: & \forall (u,v) \in E : (w_r(u,v) = 0 \wedge L(u) \leq L(v) - d(v)) \\ & \wedge ((w_r(u,v) > 0 \vee u \in PO) \wedge L(u) = \Phi - T_s) \\ P4: & \forall (u,v) \in E : (w_r(u,v) = 0 \wedge R(u) \geq R(v) - d(v)) \\ & \wedge ((w_r(u,v) > 0 \vee u \in PO) \wedge R(u) = \Phi + T_h) \end{array}$$

The objective function maximize the reduction of register observability, which is tantamount to minimizing the total register observability. $P3 \wedge P4$ is just a rearrangement of (6). Defining $P1' \triangleq P1 \wedge P3$ and $P2' \triangleq P2 \wedge P4$, we can reduce the constraints of Problem 1 to $P0 \wedge P1' \wedge P2'$.

We now explain each of the constraints. $P0$ enforces the validity of the retiming, requiring number of registers on the every edge of the retimed circuit to be non-negative. $P1'$ enforces the feasibility of the retiming with given clock period Φ. $P2'$, on the other hand, ensures that the length of the shortest path in the circuit is larger than a given number R_{min}. We will later discuss how to choose R_{min} in Section V. For a vertex v, the lengths of the longest path and shortest path started at v is $(\Phi - T_s - L(v) - d(v))$ and $(\Phi + T_h - R(v) - d(v))$. For each node $P1'$ and $P2'$ try to bound the difference between $(\Phi - T_s - L(v) - d(v))$ and $(\Phi + T_h - R(v) - d(v))$ from opposite directions, which equals to $R(v) - L(v)$. According to Theorem 1, $R(v) - L(v)$ is the bound of ELW for $v \in V$. Therefore, $P2'$ together with $P1'$ enforces the ELW constraints on Problem 1.

Compared to the min-obs retiming problem in [17], our problem considers both logic masking and timing masking and has better control on the overall SER of a sequential circuit. It is also worth noticing that we are solving an approximate problem by using the ELW bounds $R(v) - L(v)$ as constraints. However, such an approximation at retiming stage is reasonable because accurate timing can only be obtained after post-layout stage.

IV. Algorithm Description

In this section, we describe the design of an efficient optimal algorithm for Problem 1.

A. Sketch of the Algorithmic Idea

We start by examining the structure of Problem 1. If we ignore the constraints in $P2'$ in Problem 1, we actually obtain a problem equivalent to the MinObs retiming proposed in [17], which is in turn equivalent to min-area retiming [18], [20], [22] in terms of the problem structure.In traditional approaches, two $|V| \times |V|$ matrices W and D are pre-computed to capture the critical timing constraints, where for any vertex pair $u, v \in V$, $W(u,v)$ is the minimum number of registers along any path from u to v, and $D(u,v)$ is the maximum delay of these paths from u to v. Then, based on the information from W and D, the min-area or MinObs retiming is converted to the dual of the min-cost flow problem [18], [22], of which polynomial-time solvers exist. However, the bottleneck of this class of algorithms lies in the $\Theta(|V|^2)$ memory space to construct W and D and the resulting dense flow graph [19]. In [17], the MinObs retiming problem is solved by LP solver, which also involves the construction of W and D. Their algorithm is expensive in both run-time and memory.

Recently, Wang and Zhou [20] proposed an efficient incremental min-area retiming with $O(|E|)$ storage. Their iMinArea algorithm reduces the number of registers incrementally and uses a regular forest to dynamically maintain the critical timing constraints instead of explicitly constructing W and D matrices. Considering the efficiency of their algorithm and the similarity between the Problem 1 and the min-area retiming problem, we borrow the algorithmic idea of iMinArea for min-area retiming and extend the regular forest data structure to accommodate the difficulties induced by $P2'$.

We now describe the basic idea of our algorithm. Let $P(r) \triangleq P0 \wedge P1' \wedge P2'$ and $\hat{B}(r) = \sum_{v\in V} b(v)r(v)$. Our algorithm works iteratively. Starting from $r_0 = 0$, it computes a sequence of retimings $\{r_j\}_1^J$. At j-th iteration, r_j is updated incrementally from r_{j-1}, which satisfies $P(r_j)$ and $\hat{B}(r_j) > \hat{B}(r_{j-1})$. If at a certain iteration J, no improvement on $\hat{B}(r_{J-1})$ can be made, the algorithm declares that optimal solution r_{J-1} has been obtained and terminates.

At each iteration, to improve $\hat{B}(r)$, we select the vertices with $b(v) > 0$ and decrease their $r(v)$. However, such operations may violate $P0$, $P1'$ or $P2'$. Suppose for a vertex x, a decrease of $r(x)$ introduces a violation of $P(r)$. We have to decrease $r(p(x))$ of another vertex $p(x)$ to fix it. Therefore, an *active constraint* [20] $(x, p(x))$ is identified to record such dependency between the vertices, which means that $r(p(x))$ should be tied to $r(x)$ for simultaneous decrease. In Problem 1, there are three types of active constraints which correspond to violations of $P0$, $P1'$ and $P2'$ respectively. First, as is shown by Figure 2(a), if $w_r(u,v) = 0$, decrease of v should be matched with decrease of u to fix $\neg P0$. Thus, an active constraint (v,u) should be added. To explain the other two cases, we introduce two labelings $lt : V \to V$ and $rt : V \to V$. $lt(v)$ and $rt(v)$ represent the ending nodes of the longest and shortest paths on

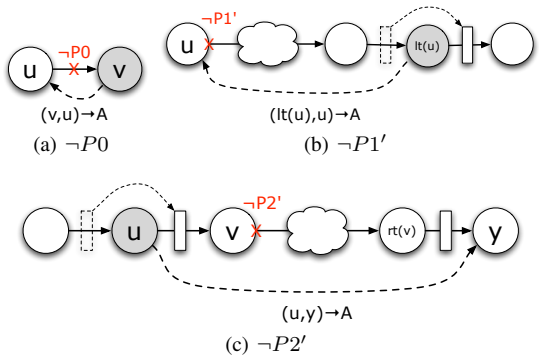

Fig. 2. Three types of active constraints.

which v lies. Figure 2(b) demonstrates the second type of active constraints associated with $P1'$ violations. A decrease of a vertex z creates a a critical longest path $u \rightsquigarrow z$. Therefore $lt(u) = z$ and $L(u)$ violates $P1'$. In order to fix $P1'$, we need to move a register out of u to cut the critical longest path. Thus, an active constraint $(lt(u), u)$ should be added. The above two cases resemble those in [20]. However, in our problem, an additional and more complicated case exists, which is shown in Figure 2(c). For a critical shortest path $u \rightarrow v \rightsquigarrow rt(u)$, a decrease of $r(u)$ may further shrink the path and introduce a violation of $P2'$. Let $z \triangleq rt(u)$. There must exist y with $(z, y) \in E \wedge w_r(z, y) > 0$, or z cannot be $rt(u)$. To fix $P2'$, for each $y : (z, y) \in E \wedge w_r(z, y) > 0$, we need to move all the registers out of (z, y) by decreasing y. A third type of active constraint (u, y) should be added. Note that it is possible that more than one registers need to be moved out in order to clear (z, y).

During the development of the algorithm, a set of active constraints A is maintained to bind the vertices with $b(v) > 0$ and those with $b(v) \leq 0$. Let a closed set of vertices I under A be the set of vertices such that $(u, v) \in A : u \in I \Rightarrow v \in I$. At each iteration, if a closed set I under A with maximum $b(I) > 0$ is found, the algorithm decreases the vertices in I to obtain a new retiming r with better $\hat{B}$. we use $r(I)$ to denote this operation. If no such I can be found, the algorithm exists with r from the last iteration as the optimal one.

B. Weighted Regular Forest

To overcome the memory bottleneck, a special data structure called regular forest is designed in [20] to manage the active constraint set A. The regular forest keeps at most $|V| - 1$ actives constraints in A and guarantees the termination of the algorithm. Since the regular forest is the key to the efficiency and correctness of the algorithm, we briefly introduce the concept of regular forest and then focus on how to employ and enhance it in our algorithm with the addition of $P2'$ constraints. The detail of the regular forest is omitted here and can be found in [20].

A regular forest F with vertices V is composed of regular trees T's. For each vertex v, there is a label $B(v) = b(T_v)$, where T_v is the subtree rooted at v. The total gain of a tree T is expressed as $b(T) = \sum_{v \in T} b(v)$. Edges between the vertices in a tree present the active constraints and a label $U : V \rightarrow \{true, false\}$ is used to maintain the direction. For a non-rooted vertex v and its parent p_v, if $U(v) = true$, then (v, p_v) is the active constraint. Otherwise, (p_v, v) is the active constraint. A regular tree T is a tree satisfying the following conditions:

1) if $b(T) > 0$, then $(U(v) \wedge B(v) > 0) \vee (\neg U(v) \wedge B(v) \leq 0)$;
2) if $b(T) = 0$, then $(U(v) \wedge B(v) > 0) \vee (\neg U(v) \wedge B(v) < 0)$;
3) if $b(T) < 0$, then $(U(v) \wedge B(v) \geq 0) \vee (\neg U(v) \wedge B(v) < 0)$,

where v is the non-rooted vertex in T. According to the sign of $b(T)$ of the regular trees they belongs to , vertices V in the regular forest F are partitioned into three sets:

1) $v \in T \wedge b(T) > 0 \Rightarrow v \in P(F)$
2) $v \in T \wedge b(T) = 0 \Rightarrow v \in Z(F)$
3) $v \in T \wedge b(T) < 0 \Rightarrow v \in N(F)$

Let $V_P(F)$ denote all the vertices in the positive trees, i.e. $V_P(F) \triangleq \{v | v \in P(F)\}$. It is shown in [20] that $I = V_P(F)$ is the closed set under A with maximum $b(I) > 0$.

In [20], the min-area retiming algorithm starts with a regular forest of $|V|$ tree and no edges. It incrementally expands $V_P(F)$ by combining the positive trees with negative or zero trees. Regularity of the forest is enforced during the update. The key operation is UpdateForest(F, x, $p(x)$), which adds a new active constraint $(x, p(x))$ to the forest. After the update, $I = V_p(F)$ is used to update the retiming and if $V_P(F) = \emptyset$, optimality is claimed.

C. Proposed Algorithm

Despite the similar structure of Problem 1, there is one fundamental difference between our algorithm and iMinArea from [20]. In iMinArea, at each iteration, the change of $r(v)$ for each $v \in V$ is at most by one while in our algorithm, we may need to relocate multiple registers for one vertex in order to fix a $P2'$ violation and the consequent $P0$ violations. This fundamental difference makes the dependencies between the vertices in A non-uniform. Therefore, instead of directly applying the regular forest and its associated UpdateForest routine in our algorithm, we make two majors changes, one in data structure and the other in algorithm itself.

First, we introduce $w(v)$ for each vertex $v \in F$ to record the decrease of $r(v)$. When the retiming r is updated by $v \in I$, $r(v) = r(v) - w(v)$. With the weight information, the gain of a tree in the regular forest is also modified to $b(T) = \sum_{v \in T} b(v)w(v)$. We call F with w a *weighted regular forest*. When the algorithm starts, it does not know how many registers it will move in. $w(v)$ is initialized to 1 for $v \in V$. The knowledge of $w(v)$ is updated during the development of algorithm. We extend the forest update routine UpdateForest(F, x, v, w) to update $w(v) = w$ in the forest. Note that the $w(v)$ is updated only when it is a tree by itself so that the regularity of F will not be violated.

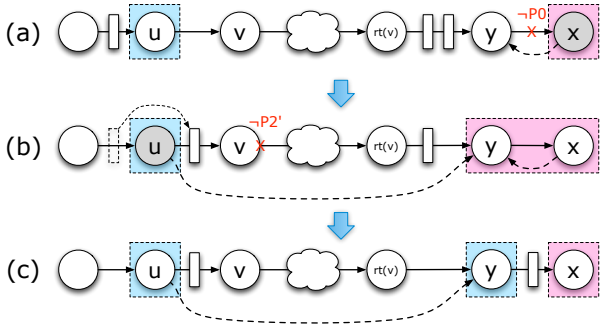

Fig. 3. An example of P-tree-and-P-tree link.

Next, the original UpdateForest routine in [20] always links a vertex in a non-positive tree into a vertex in a positive tree, i.e. condition $x \in V_P(F) \wedge \neg(v \in V_P(F))$ always holds for UpdateForest(F, x, v). The dependencies between vertices from positive trees are automatically taken care of due to the uniform update of $r(V_P(F))$. However, in our problem, due to the non-uniformity of A maintained in F and the incomplete knowledge of $w(v)$, it is possible that $x \in V_P(F) \wedge v \in V_P(F)$ for UpdateForest(F, x, v, w). Figure 3 shows an example of how such a scenario comes into existence. In Figure 3(a), each vertex constitutes a regular tree by itself. u and x are the vertices with positive gains. If x is examined first. A decrease of $r(x)$ will violate PO. To fix it, $r(y)$ is decrease by one and y is bundled with x into a positive tree. Then, in Figure 3(b), when $r(u)$ is decreased. It causes a $P2'$ violation and the algorithm should link u with y, which is already in the positive tree. Such cases indicate that previous knowledge of $w(y)$ is incorrect and needs to be updated. However, a update on $w(y)$ may disable some of active constraints in the tree it belongs to. For example, in Figure 3(c), active constraint (x, y) is no longer needed after $w(y)$ is changed to 2. We design a $O(|V|)$ BreakTree(y) routine which locates the tree $T(y)$ that y

belongs to, changes its root to y and breaks y from its sub-trees by deleting the edges from y to its children. Then, the $w(y)$ is updated along with the active constraint (u, y) into F.

Note that the root cause of positive-positive tree linking is the incomplete weight information of the nodes. It is possible that a non-positive to positive tree linking (x, v) also requires update of $w(v)$. In both cases, the BreakTree routine will resolve the problem and update $w(v)$.

D. Summary

The proposed MinObsWin algorithm is listed in Algorithm 1. It follows the same structure of the min-area retiming algorithm in [20]. Operations to handle $\neg P2'$ are added from Line 9-12. Line 19-21 breaks the tree and update the knowledge of w.

Algorithm 1 MinObsWin

Input: Φ: the clock period, R_{min}: bound on shortest path
Output: optimal retiming for Problem 1

```
 1: Initialize a feasible retiming r.
 2: Init(F)
 3: loop
 4:    p ← {}, q ← {}, w ← 0
 5:    I ← V_P(F)
 6:    if I = ∅ then
 7:       BREAK {r is optimal.}
 8:    end if
 9:    if ¬P2 then
10:       z ← ru(v)
11:       Find y : (z, y) ∈ E ∧ w_r(z, y) > 0.
12:       p ← u, q ← y, w = w_r(z, y)
13:    else if ¬P0 then
14:       p ← v, q ← u, w = −w_r(u, v)
15:    else if ¬P1 then
16:       p ← lt(v), q ← v, w = 1
17:    end if
18:    if p ≠ {} ∧ q ≠ {}  then
19:       if w(q) requires update then
20:          BreakTree(q)
21:          UpdateForest(F, p, q, w(q) + w)
22:       else
23:          UpdateForest(F, p, q, w)
24:       end if
25:    else
26:       r ← r(I)
27:    end if
28: end loop
```

With the same reasoning as that for iMinArea in [20], since we decrease r monotonically and r is bounded by $|V|$, Algorithm 1 will terminate with optimal solution.

Theorem 2: MinObsWin algorithm will terminate and when it terminates, it returns the optimal solution r of Problem 1.

In each iteration, the timing analysis and BreakTree operation takes $O(|E|)$ time, while update of r labeling takes $O(|V|)$ time. So the time complexity for each iteration is $O(|E|)$. Furthermore, the number of iterations is bound by $|V|^2$. Therefore, we obtain the complexity of MinObsWin algorithm.

Theorem 3: The space complexity of the MinObsWin algorithm is $O(|E|)$. The time complexity of the MinObsWin algorithm is $O(|V|^2|E|)$.

Actually, the MinObsWin algorithm inherits the superb linear memory requirement and the same time complexity from iMinArea.

V. Initialization

Algorithm 1 requires an initialization of a feasible retiming with given Φ and R_{min}. How to choose Φ and R_{min} is also an interesting problem. It is not reasonable to sacrifice the performance of the circuit by starting Algorithm 1 with a large Φ. In our work, we start with a sequential circuit retimed so that it has the minimal clock period Φ_{sh} under setup and hold time constraints by using the method proposed in [23]. It is possible that such a feasible retiming does not exist due to reconvergent paths [23]. In that case, we carry out the min-period retiming [24] and get the minimal clock period Φ_{min}. Such clock period constraint is very tight. So we slightly relax it by a small factor ϵ. In our work, we choose $\epsilon = 10\%$. R_{min} is then chosen as the minimal shortest path length in the retimed circuit:

$$R_{min} = \min_{(u,v)\in E \wedge w_r(u,v)>0} (\Phi_{sh} + T_h - R(v) - d(v)).$$

VI. Experimental Results

We implemented the proposed MinObsWin algorithm in C++ and tested it on a Linux machine with an Intel quad-core E5405 2.0GHz CPU and 2GB memory. The performance and solution quality are evaluated on ISCAS89 and ITC99 benchmarks obtained from authors of [20]. Φ and R_{min} are computed using the method suggested in Section V, where T_s and T_h are set as 0 and 2 as is suggested by [23]. We also implemented the SER analysis engine with time-frame expansion from [17]. A 15 time-frame expansion is used for each circuit to reach to steady operational state as is suggested in [17]. SER for single gate is extract from SPICE characterization using the method in [25]. Note that when doing the SER analysis, we compute the real size of the ELW for each gate with (3). We choose the MinObs retiming algorithm in [17] for comparison, which minimizes the observability without ELW constraints. Since the advantage of the regular forest based retiming algorithm over the W/D-matrix based ones on memory usage and CPU time is evident in [20], we do not repeat the experiments to compare memory usage and CPU time of the propose algorithm with the LP-based algorithm in [17]. Instead, by simply commenting out Line 9-12 and Line 19-21 in Algorithm 1, we can reduce the proposed algorithm into an efficient MinObs algorithm, which solve the same problem in [17] and is of the same run-time and space bounds as the algorithm in [20].

Table I lists the comparison of the performance and SER results on 21 largest circuits between MinObs retiming and the proposed MinObsWin retiming algorithm, along with the statistics of the circuits. The number of vertices ("$|V|$"), number of edges ("$|E|$"), number of registers in the original circuits ("$\#FF$"), clock period constraint ("Φ") and the SER of the original circuits ("SER") are shown under the header of "Statistics". The columns under "Efficient MinObs" are results from our efficient implementation of the method in [17] , where "$\Delta\#FF_{ref}$" is the change of number of registers after retiming, "t_{ref}" is the CPU time and "ΔSER_{ref}" is the relative improvement of SER over the original circuits. Columns with similar titles under "MinObsWin" give similar statistics of our algorithm. In addition, "#J" gives the number of iterations.

We compare the run-time of two retiming algorithms in the columns of "t_{ref}"and "t_{new}". The time unit is second. To be fair, we exclude the results of b18_1_opt, b18_opt, b19_1_opt and b19_opt when computing the average run-times because including them will make the average run-time of MinObsWin faster than MinObs, which is deceptive. The run-time behavior of MinObsWin on these special cases will be explained later. Without the noisy data, we find that the average run-time of MinObsWin is $2.5X$ slower than MinObs. This is due the extra computational effort to detect and fix $\neg P2'$. However, the algorithm we used to solve the MinObs problem is much more efficient than LP-based solvers with on average 60X speedup, as is reported in [20]. Therefore, with the inherited memory and CPU efficieny from [20], the proposed algorithm is very competittive against LP-based solver in [17].

Next, we compare the optimization ability on SER between the logic masking only retiming (MinObs) and the proposed MinObsWin retiming. "$\frac{SER_{ref}}{SER_{new}}$" compares the resulting SERs from MinObs retiming and from the proposed algorithm. In most of the cases, the proposed algorithm outperforms the MinObs algorithm in term of the resulting SER, which shows the advantage of considering both logic masking and timing masking in the optimization process. Especially, in circuit s38417, large $\frac{|V|}{\#FF}$ ratio indicates that combinational gates take a large part of the circuit than the registers. Although the MinObs

TABLE I
COMPARISON OF SER ON ISCAS89 AND ITC99 CIRCUITS

Circuit	Statistics					Efficient MinObs			MinObsWin				
	$\|V\|$	$\|E\|$	$\#FF$	Φ	SER	$\Delta\#FF_{ref}$	t_{ref}	ΔSER_{ref}	$\Delta\#FF_{new}$	t_{new}	$\#J$	ΔSER_{new}	$\frac{SER_{ref}}{SER_{new}}$
s13207	7952	10896	1508	117	7.72E-03	-43.56%	0.19	-23.14%	-24.53%	0.50	2	-47.02%	122%
s15850.1	9773	13566	1567	111	9.77E-03	-54.05%	0.26	-31.71%	-54.05%	0.27	9	-31.71%	100%
s35932	16066	28588	5814	145	2.42E-02	-45.37%	0.22	-35.45%	-34.76%	1.14	4	-66.75%	194%
s38417	22180	31127	2806	81	1.59E-02	11.51%	2.61	2.92%	13.61%	3.24	4	-8.62%	113%
s38584.1	19254	33060	7371	262	2.48E-02	-32.33%	0.89	-33.23%	-31.96%	1.82	3	-41.96%	115%
b14_1_opt	4049	9036	2382	112	9.15E-03	-64.02%	0.48	-12.89%	-64.02%	1.94	5	-32.89%	130%
b14_opt	5348	11849	2041	135	9.75E-03	-57.76%	0.51	-26.71%	-50.05%	0.35	2	-6.67%	79%
b15_1_opt	7421	16946	2798	158	1.25E-02	-36.88%	0.94	-24.58%	-33.84%	4.25	5	-37.12%	120%
b15_opt	7023	15856	2415	195	1.35E-02	-46.17%	0.59	-26.97%	-43.22%	2.26	4	-45.74%	135%
b17_1_opt	23026	52376	8791	192	3.92E-02	-27.64%	8.77	-12.64%	-37.58%	27.31	5	-36.34%	137%
b17_opt	22758	51622	7787	266	3.42E-02	-23.75%	2.05	-28.13%	-19.09%	10.09	6	-45.94%	133%
b18_1_opt	68282	151746	21027	251	9.42E-02	-30.92%	45.02	-28.51%	-0.05%	1.78	1	0.00%	71%
b18_opt	69914	155355	20907	255	9.56E-02	-30.92%	28.69	-32.92%	0.00%	0.96	1	0.00%	67%
b19_1	212729	410577	59580	317	2.45E-01	-48.35%	142.91	-30.40%	-48.35%	144.99	6	-30.40%	100%
b19	224625	433583	60801	317	2.50E-01	-49.27%	149.33	-30.72%	-49.27%	150.11	6	-30.72%	100%
b20_1_opt	10166	22456	3462	191	1.63E-02	-57.30%	1.66	-34.51%	-56.21%	1.68	4	-34.51%	100%
b20_opt	11958	26479	4761	182	2.15E-02	-65.68%	2.56	-31.48%	-65.42%	2.61	4	-31.41%	100%
b21_1_opt	9663	21246	2451	171	1.22E-02	-34.31%	0.38	-25.28%	-31.78%	1.74	4	-48.87%	146%
b21_opt	12135	26686	4186	215	1.90E-02	-66.72%	0.84	-33.35%	-66.36%	2.64	4	-40.82%	113%
b22_1_opt	14957	32663	4398	194	2.19E-02	-50.55%	3.19	-31.39%	-50.36%	3.27	4	-33.34%	103%
b22_opt	17330	37941	5556	178	2.67E-02	-50.61%	7.51	-29.56%	-51.02%	16.75	3	-35.88%	110%
AVG.						-43.04%	1.97*	-26.70%	-38.01%	4.92*	4	-32.70%	115%

retiming tries to reduce the observability of registers, its lack of control on sizes of ELWs results in an retimed circuit with degraded SER. The MinObsWin algorithm, on the other hand, limits the ELW sizes, and still manages to get improvement. Notice that for several circuits, the proposed algorithm resulted in equal or larger SER than the MinObs algorithm. After inspection, we found that this is due to the stringent constraints on Φ and R_{min}. In circuits such as s15850.1, no valid retiming under setup and hold time constraint is available. R_{min} is thus chosen to be the minimal gate delay. Therefore, $P2'$ will not be violated and our algorithm acts essentially the same as the MinObs retiming. For cases b14_1_opt, b18_1_opt and b18_opt, the paths with length small than R_{min} happened to end at the primary outputs, where no registers can be moved into host. The proposed algorithm then exited immediately. Since there are no constraints on the shortest path length in the MinObs retiming, it can optimize the observability of the registers more freely. But on average, the proposed min-obs retiming under ELW constraints achieves a 32.70% reduction of SER from the original circuits and obtains an extra 15% improvement over the MinObs retiming in [17].

Finally, as is indicated by column "$\Delta\#FF_{new}$", our algorithm results in an average of 38% reduction of number of registers. This by-product suggests a desirable reduction of area/power of the circuits. Clearly, there is an area/power penalty of the proposed MinObsWin algorithm against pure min-area retiming due to the different objectives.

VII. CONCLUSIONS

In this paper, we inspected the impact of retiming on the SER of the sequential circuit. Problem of minimum observability retiming under error-latching window constraints was proposed to reduce the SER of the circuit, with consideration of both logic and timing masking. The MinObsWin algorithm was designed to solve the problem efficiently and optimally. Experimental results show on average 32.7% reduction on SER from the original circuits and 15% improvement over the previous method in [17]. As an extension of this work, the objective function in Problem 1 can be augmented to include area/power weight. The algorithm itself remains the same.

REFERENCES

[1] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[2] L. Massengill, B. Bhuva, W. Holman, M. Alles, and T. Loveless, "Technology scaling and soft error reliability," in *IRPS*, april 2012, pp. 3C.1.1 –3C.1.7.

[3] Y. S. Dhillon, A. U. Diril, A. Chatterjee, and A. D. Singh, "Analysis and optimization of nanometer cmos circuits for soft-error tolerance," *IEEE TVLSI*, vol. 14, no. 5, 2006.

[4] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," *IEEE TCAD*, vol. 25, no. 1, pp. 155–166, 2006.

[5] M. R. Choudhury, Q. Zhou, and K. Mohanram, "Design optimization for single-event upset robustness using simultaneous dual-vdd and sizing techniques," in *ICCAD*, 2006, pp. 204–209.

[6] K.-C. Wu and D. Marculescu, "Power-aware soft error hardening via selective voltage scaling," in *ICCD*, 2008, pp. 301–306.

[7] K. Mohanram and N. A. Touba, "Partial error masking to reduce soft error failure rate in logic circuits," in *DFT*, 1997, pp. 433–440.

[8] ——, "Cost-effective approach for reducing soft error failure rate in logic circuits," in *Proc. Intl. Test Conf.*, 2003, pp. 893–901.

[9] S. Almukhaizim and Y. Makris, "Seamless integration of SER in rewiring-based design space exploration," in *Proc. Intl. Test Conf.*, 2006, pp. 1–9.

[10] M. Jose, Y. Hu, R. Majumdar, and L. He, "Rewiring for robustness," in *DAC*, 2010, pp. 469–474.

[11] S. Krishnaswamy, S. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *ICCAD*, 2007, pp. 149–154.

[12] N. Jing, J. Lee, C. Zhang, J. Tong, Z. Mao, and L. He, "Fault modeling and characteristics of sram-based fpgas," in *FPGA*, 2011, pp. 279–279.

[13] M. Zhang, S. Mitra, T. M. Mak, N. Seifert, N. J. Wang, Q. Shi, K. S. Kim, N. R. Shanbhag, and S. J. Patel, "Sequential element design with built-in soft error resilience," *IEEE TVLSI*, pp. 1368–1378, 2006.

[14] V. Joshi, R. R. Rao, D. Blaauw, and D. Sylvester, "Logic SER reduction through flipflop redesign," in *ISQED*, 2006, pp. 611–616.

[15] S. Krishnaswamy, I. L. Markov, and J. P. Hayes, "On the role of timing masking in reliable logic circuit design," in *DAC*, 2008, pp. 924–929.

[16] K.-C. Wu and D. Marculescu, "Clock skew scheduling for soft-error-tolerant sequential circuits," in *DATE*, 2010, pp. 717–722.

[17] S. Krishnaswamy, I. L. Markov, and J. P. Hayes, "Improving testability and soft-error resilience through retiming," in *DAC*, 2009, pp. 508–513.

[18] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *Journal of VLSI and Computer Systems*, vol. 1, no. 1, pp. 41–67, Spring 1983.

[19] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Application*. Prentice Hall, 1993.

[20] J. Wang and H. Zhou, "An efficient incremental algorithm for min-area retiming," in *DAC*, Anaheim, CA, Jun. 2008.

[21] S. Krishnaswamy, S. Plaza, I. L. Markov, and J. P. Hayes, "Signature-based SER analysis and design of logic circuits," *IEEE TCAD*, vol. 28, no. 1, pp. 74–86, 2009.

[22] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *ICCAD*, 1994, pp. 226–233.

[23] C. Lin and H. Zhou, "An efficient retiming algorithm under setup and hold constraints," in *DAC*, San Francisco, CA, 2006.

[24] H. Zhou, "A new efficient retiming algorithm derived by formal manipulation," *ACM TODAES*, vol. 13, no. 1, Jan. 2008.

[25] R. R. Rao, K. Chopra, D. Blaauw, and D. Sylvester, "An efficient static algorithm for computing the soft error rates of combinational circuits," in *DATE*, 2006, pp. 164–169.