CSL: Coordinated and Scalable Logic Synthesis Techniques for Effective NBTI Reduction

Chen-Hsuan Lin*, Subhendu Roy†, Chun-Yao Wang‡, David Z. Pan† and Deming Chen*
*Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, IL, USA
{clin54, dchen}@illinois.edu
†Department of Electrical and Computer Engineering
University of Texas at Austin, TX, USA
subhendu@utexas.edu, dpan@ece.utexas.edu
‡Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan, R.O.C
wcyao@cs.nthu.edu.tw

***Abstract*—Negative Bias Temperature Instability (NBTI) has become a major reliability concern in nanoscale designs. Although several previous studies have been proposed to address the NBTI effect during logic synthesis, their performance is limited because of focusing on a certain logic synthesis stage. Additionally, their complicated algorithms are not scalable to large designs. To tackle this, we propose a coordinated and scalable logic synthesis approach, which integrates techniques at different logic synthesis stages, ranging from subject graph to technology mapping and mapped netlist, to achieve an effective NBTI reduction. To our best knowledge, this is the first work that considers and mitigates NBTI impact in subject graphs, the earlier stage of logic synthesis. Experimental results on industry-strength benchmarks show that our approach can achieve 6.5% NBTI delay reduction with merely 2.5% area overhead on average, while a previous work barely gets NBTI delay reduction when the circuits are optimized beforehand, the circuit sizes are large, and standard cell libraries are richer.**

I. Introduction

With technology downscaling to nanometer range, circuit reliability has become a critical challenge for robust system designs [6]. Reliability degradation results from factors such as soft errors, manufacturing variability, temperature effects, and aging. As the trend moves to nanoscale devices, aging, which causes significant loss on circuit performance and lifetime, is becoming relatively dominant in reliability concerns. *Hot Carriers Injection* (HCI) [18] and *Negative Bias Temperature Instability* (NBTI) [17][19] are two major aging phenomena, which can lead to permanent degradation of transistors, thus hurting the reliability of nanoscale circuits. Among these aging phenomena, NBTI has become particularly prominent and has received considerable attention.

NBTI is an aging phenomenon that increases the threshold voltage (V_{th}) of PMOS transistors over a long period of time, thus slowing down the speed of logic gates and preventing circuits from meeting the timing requirements. NBTI occurs when PMOS transistors are under negative gate-to-source bias (stress phase: $V_{gs} = -V_{dd}$). During the stress phase, interface traps along the silicon-oxide interface take place due to the dissociation of $Si - H$ bonds. For instance, over a period of ten years, these traps can increase the V_{th} of PMOS in 65nm technology by up to $50mV$ [21], resulting in the delay degradation of circuits. Although some of interface traps can be annealed by relaxing the stress condition ($V_{gs} = 0$), this recovering process is incomplete. Therefore, the NBTI-induced delay degradation crucially depends on the amount of time during which PMOS transistors are under the stress phase. The signal probability SP (the probability of signal to be logic 1) is an effective metric to estimate the NBTI-induced aging degradation of PMOS, which causes the delay of logic gates increase over a period of time [9].

From the logic synthesis perspective, previous works mitigate NBTI effect by taking account of signal probability during synthesis, and these works can be classified into two major groups: considering NBTI either during or after technology mapping (TechMap). During-TechMap: [9] matched the standard cells with the most suitable gate size based on signal probability for reducing NBTI effects. [5] proposed a commercial tool flow to balance the circuit timing with respect to specific NBTI-aware guardbands for improving lifetime. After-TechMap: [22][23] used logic restructuring and pin reordering techniques with considering signal probabilities to mitigate NBTI-induced delay degradation. [14][20][24] applied gate sizing techniques with variable V_{th} to decrease NBTI impact and achieve timing closure. However, some of them might have scalability issues because of applying complicated algorithms designed only for a certain synthesis stage. For example, the complexity of restructuring algorithms propsoed in [22] might be up to $O(n^3)$, where n is the number of gates in a circuit. Furthermore, their performance is constrained by the results of corresponding technology mapping.

Technology mapping [7][8] based on tree- or directed acyclic graph-covering has a known issue of suffering from structural bias. In other words, the structure of the resultant mapped netlist depends heavily on the given subject graph, which is a multi-level network of simple gates for representing the Boolean function of the circuit. Although some researches [3][11][16] targeted at mitigating structural bias heuristically, they cannot avoid this issue completely. Therefore, based on this fact, our proposed work is inspired by two ideas: "*Can the mapped netlist have better NBTI tolerance if the given subject graph is NBTI-friendly?*" and further *"How to generate an NBTI-friendly subject graph?"*

Unlike previous works, which attacked the NBTI effect at certain later stages of logic synthesis and were limited by complicated algorithms, we propose a **C**oordinated and **S**calable **L**ogic synthesis approach (**CSL**), which integrates techniques at different stages to achieve an effective NBTI reduction. Furthermore, the proposed techniques are designed to deal with large-scale benchmarks. We observe that considering the NBTI effect in the early stages of the design flow can have a better chance to boost the results with less overhead. At the first stage, *subject graph*, we propose an algorithm to restructure the subject graph into NBTI-friendly one iteratively. At the second stage, *technology mapping*, we search for the best matching gates that result in better NBTI tolerance with minimum area overhead from standard cell libraries. This stage also prevents the performance gain at the previous stage from being eliminated. At the last stage, *mapped netlist*, we propose a scalable pin reordering techniques, *smart pin*, to tweak the structure of transistor connections for further reducing NBTI effect with negligible runtime overhead. In sum, NBTI-aware logic restructuring, NBTI-aware technology mapping, and NBTI-aware pin reordering work together to construct a comprehensive and robust NBTI-aware logic synthesis approach.

The contributions of this work are three-fold. To our best knowledge, this is the first NBTI study that (1) considers the NBTI effect at the subject graph stage, (2) deals with large-scale benchmarks even with around a million of gates, and (3) coordinates techniques across several stages to build a comprehensive logic synthesis approach for NBTI reduction. Experimental results show that on average CSL can achieve 6.5% NBTI delay reduction with 2.5% area overhead among the industry-strength benchmarks from ISPD'12 contest [12] without worrying about the size of circuits and the composition of standard cell libraries.

II. Preliminaries

This section introduces the background of this work, including subject graph, NBTI modeling, and transistor stacking effect in the PMOS network.

A. Subject graph

The subject graph, the input to the technology mapping stage, used in this work is in *And-Inverter Graph* (AIG) format, which has been shown an efficient data structure for manipulating large Boolean networks in logic synthesis and formal verification [1][3][10]. An AIG is a multi-level Boolean network composed of two-input ANDs and INVs. The data structure of AIG is a *directed acyclic graph*, in which nodes with no incoming edge are primary inputs (PIs) while with two incoming edges are two-input AND gates. The edges in AIGs represent wires. Inverters are represented by bubbles on the edges. All primitive gates have their corresponding forms in AIGs as shown in Fig. 1; therefore, arbitrary Boolean networks can be represented by AIGs.

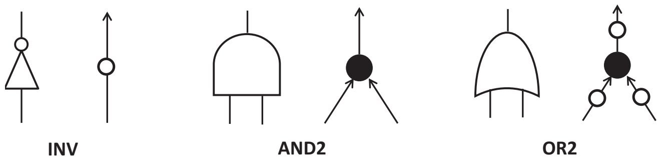

Fig. 1. The corresponding AIGs of primitive gates: INV, AND2, and OR2.

B. NBTI modeling

This section briefly introduces the NBTI modeling [2][9][14] used in this work. The model is used to estimate the delay degradation of each gate in the standard cell library, as a function of the signal probabilities of gate inputs and intrinsic gate delay. An *NBTI-stress factor* of a gate input i, denoted by $\gamma_{nbti}(i)$, represents the probability of the PMOS transistor at input i being stressed. $\gamma_{nbti}(i)$ impacts the *rise delay* of the timing arcs from the gate input i to the gate output. The idea of NBTI modeling is to estimate the corresponding increase in V_{th} for different NBTI-stress factors at the end of a time period. Then, the final V_{th} is plugged into HSpice simulation to get the NBTI-affected rise delay. Finally, a piecewise linear model, similar to that in [2][14], is developed (within 1% difference to the HSpice simulation data) for adjusting the output rise delay caused by an input i based on $\gamma_{nbti}(i)$ during timing analysis. The details of the modeling can be referred to in [9][14]. Fig. 2 shows the rise delay characterization of an inverter with input i based on $\gamma_{nbti}(i)$. We can see that the maximum increase of rise delay of the inverter is about 25%.

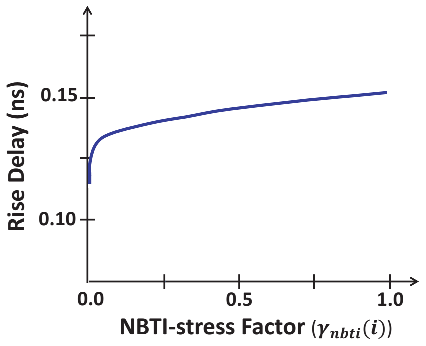

Fig. 2. Rise delay vs. NBTI-stress factor in an INV [14].

Next, let us discuss the relationship between signal probability and $\gamma_{nbti}(i)$ of a gate input i. The $\gamma_{nbti}(i)$ can be derived from the signal probability of input i and that of other inputs. Take an example of And-Or-Inverter (AOI12) gate in Fig. 3(a), whose output function is $o = \overline{a + bc}$. Let SP_a, SP_b, and SP_c denote the signal probabilities of inputs a, b, and c, respectively. The γ_{nbti} of b and c, $\gamma_{nbti}(b)$ and $\gamma_{nbti}(c)$, are simply their probabilities of being logic 0, while $\gamma_{nbti}(a)$ is

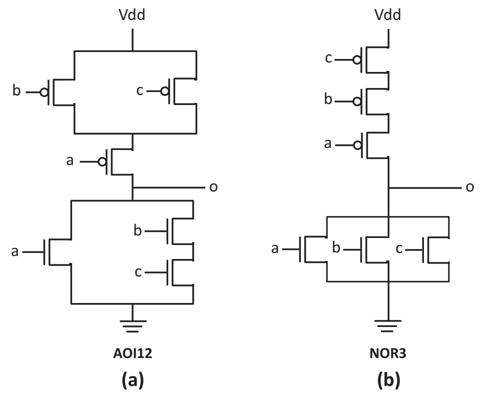

Fig. 3. The transistor schematics of gates AOI12 and NOR3.

the probability that a is equal to logic 0 as well as at least one of b and c is equal to 0, where the PMOS transistor at a is stressed. The γ_{nbti} for inputs, a, b, and c, of the AOI12 gate are shown in Eq. 1. Therefore, the NBTI-aware timing model can be built by characterizing all gates in the standard cell library accordingly.

$$\begin{aligned}\gamma_{nbti}(b) &= (1 - SP_b)\\ \gamma_{nbti}(c) &= (1 - SP_c)\\ \gamma_{nbti}(a) &= (1 - SP_a)(1 - SP_b \cdot SP_c)\end{aligned} \tag{1}$$

C. PMOS transistor stacking effect

Two transistors connected in series are called stacking. For a CMOS logic gate, if its pull-up (PMOS) network has stacking transistors, we said that these PMOS transistors have the *stacking effect* [9][22]. The stacking effect causes the NBTI effect of lower PMOS transistors, which are closer to the output signal, is to be **milder** than that of upper PMOS transistors, which are closer to the power supply (V_{dd}). This is because the lower PMOS transistors are under stress (connected to V_{dd}) only when its upper PMOS transistors are "on" simultaneously. In other words, a lower PMOS transistor can be protected from NBTI-induced aging by its upper ones. Take a three-input NOR gate in Fig. 3(b) as an example. The PMOS transistor of input a can be protected by the PMOS transistors of inputs b and c. By leveraging the stacking effect of pull-up transistors, the previous work [22] proposed a pin reordering method to reduce the NBTI effect. However, it only dealt with NOR gates. In this work, we also propose a new pin reordering method, which considers all kinds of gates that leverage the stacking effect in the library. The details are presented in Sec. III-C.

III. NBTI-AWARE LOGIC SYNTHESIS

This section introduces CSL, a coordinated and scalable NBTI-aware logic synthesis approach, which consists of three techniques: *NBTI-aware subject graph restructuring*, *NBTI-aware technology mapping*, and *NBTI-aware smart pin reordering* for the three stages mentioned, respectively. To make the approach scalable, we restrict both time and space complexity of the proposed techniques.

To begin with, we perform parallel simulations to accelerate the calculation of the signal probabilities of gates in the subject graph and mapped netlist. Please note that CSL is a general approach that can accept both purely random simulation and directed simulation; therefore, if a workload is given, designers can apply the same directed simulation patterns through the whole process to optimize the NBTI behavior of designs specifically.

Additionally, the major overhead of this NBTI effect reduction approach is area, which is a common trade-off in logic synthesis: the shorter delay of a circuit, the bigger area of the circuit. Therefore, our objective is not only to reduce the longest NBTI-affected delay of circuits, but also to control the area overhead in an acceptable range. The techniques in the approach for mitigating NBTI effect at different stages are discussed separately in the following subsections.

A. Subject graph restructuring

Our purpose at this stage is to provide an NBTI-friendly subject graph to CSL's technology mapping stage as well as to control the area overhead of mapped netlist in advance. CSL is the first work that considers and reduces NBTI effect in the subject graph.

1) NBTI-aware static timing analysis for AIG: An AIG, the format of subject graph used in this work, is a multi-level Boolean network of simple nodes to represent the functionality of a circuit. The delay of an AIG is usually measured by performing *static timing analysis (STA)* with a *unit-delay model*, in which both the rise and fall delays of gates are set to one. However, to generate NBTI-friendly AIGs, *NBTI-aware STA for AIG* is needed. Therefore, we propose an *NBTI-aware delay model* based on the NBTI modeling in Sec. II-B to estimate the NBTI-degraded delay of AIG nodes. Although the NBTI modeling in Sec. II-B is designed for standard cells, we found that its concepts can be still applied to AIG nodes, so that the NBTI delay paths in AIGs can be identified.

The usage of the NBTI modeling for AIG nodes is similar to that for standard cells. That is, the **rise delays** of simple nodes (AND gates) of AIGs should reflect NBTI degradation by considering inputs' NBTI-stress factors. As aforementioned, the structure of the mapped netlist depends strongly on the subject graph. Therefore, the intuition behind the prediction is that long NBTI delay paths in an AIG might have higher probabilities to be mapped as the long NBTI ones in the final mapped netlist. Although this predictive model might not be completely accurate, it still provides useful guidance to generate NBTI-friendly subject graphs for technology mappers, which can be observed in our experimental results.

2) NBTI balance: With the NBTI-aware STA for AIGs, we propose an AIG restructuring procedure, named *NBTI balance*, to reduce the NBTI effect. The proposed idea is illustrated in Fig. 4, and NBTI balance consists of three main steps as follows:

Step. 1: Identify the NBTI-critical POs based on the parameter $threshold$.

Step. 2: Extract and remove the fanin cones of these NBTI-critical POs.

Step. 3: Add the optimized fanin cones (with better NBTI delays) back for these NBTI-critical POs and minimize the area overhead.

In Step. 1, we identify *NBTI-critical primary outputs (POs)* in an AIG as follows. After finding the maximal NBTI delay of the circuit, d_{max}, the POs whose NBTI delays are larger than or equal to $d_{max} \times threshold$ are considered NBTI-critical POs as shown in triangles with red bold line in Fig. 4(a). The parameter $threshold$ is a user-defined parameter within an interval $[0, 1]$, which is used to determine how many POs to be re-synthesized in the next step. Optimizing a group of POs together not only saves runtime but also helps reduce area overhead down the road. The parameter setting is shown in the experimental section.

In Step. 2, as shown in Fig. 4(b), *NBTI-critical cones*, the fanin cones that belong to the NBTI-critical POs, are extracted and removed from the AIG to form a subcircuit. While extracting NBTI-critical cones, the functionalities of non-NBTI-critical POs are preserved by duplicating the sharing logic between critical and non-critical cones if needed. Please note that the most part of duplicate logic can be shared again when the optimized NBTI-critial cones are added back in the next step.

In Step. 3, the subcircuit of NBTI-critical cones is re-synthesized using a *resyn2* script in ABC [1]. This script can optimize both timing and area of the subcircuit. The physical meaning behind this operation is to destroy long NBTI delay paths of critical POs in the AIG. In the end, the optimized cones with respect to the NBTI-critical POs are added back to the AIG to maintain the original functionality as shown in Fig. 4(c). Additionally, the *structural hashing* mechanism in ABC, which can increase the number of sharing nodes by merging functionally equivalent ones, is also adopted to control the area overhead during the adding back operation as shown in Fig. 4(d).

Next, let us discuss the efficiency of this procedure. The time complexities of Step. 1 and Step. 2 are linear to the number of nodes in the AIG. For Step. 1, the delay is calculated from the PIs to the POs in the breadth-first search (BFS) manner. For Step. 2, the logic cones of NBTI-critical POs are also extracted in BFS manner from the POs to the PIs. In Step. 3, the *resyn2* script can optimize a circuit of a million of gates in few seconds, and *structural hashing* mechanism is also efficient by using hash tables. Therefore, the proposed NBTI balance procedure is efficient and thus scalable to large-scale benchmarks. This can be seen in the experimental results.

3) Complete NBTI-aware subject graph restructuring: Although NBTI balance generates an NBTI-friendly AIG for the next stage, this procedure cannot be complete without appropriate termination conditions. In other words, we have to determine the number of iterations of NBTI balance for maximizing the reduction of NBTI effect with **acceptable area overhead**. Therefore, we add two termination conditions that complete our restructuring technique. The pseudo code of the proposed ***NBTI-aware subject graph restructuring*** is shown in Algorithm 1, where the two termination conditions, "$imprvIter < minImprv$" and "$imprvTotal > targetImprv$", are involved. The physical meanings of these two conditions are explained as follows.

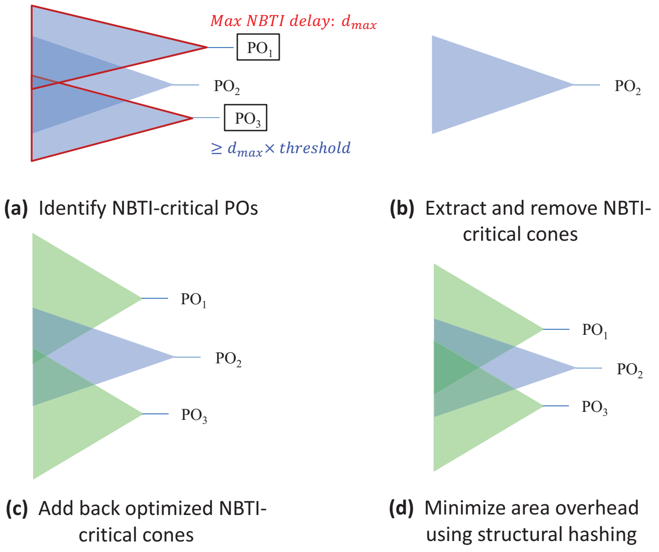
Max NBTI delay: d_{max}
PO1
PO2
PO3
$\geq d_{max} \times threshold$
PO2
(a) Identify NBTI-critical POs
(b) Extract and remove NBTI-critical cones
PO1
PO2
PO3
PO1
PO2
PO3
(c) Add back optimized NBTI-critical cones
(d) Minimize area overhead using structural hashing

Fig. 4. The illustration of NBTI Balance procedure.

- $(imprvIter < minImprv)$: Terminate the iteration when the improvement of NBTI delay of **one iteration** is **less than *minImprv***, which is set to 0.1% in this work. This condition indicates that the NBTI balance has reached its limit and no more significant improvement of NBTI delay would be expected.
- $(imprvTotal > targetImprv)$: Terminate the iteration when the **accumulated** improvement of NBTI delay (compared to the original NBTI delay) is **larger than *targetImprv***. According to our experiments, the average delay degradation under NBTI effect among the benchmark set we used is around 9%; therefore, the $targetImprv$ is set to 10% in this work to avoid over-optimization of NBTI delay. The value of $targetImprv$ should be set according to the observation of NBTI degradation of benchmark set.

In sum, the NBTI balance procedure will be iterated until the improvement is tiny or the target improvement has been achieved.

Area is traded for NBTI delay improvement in this work. However, large area overhead is not acceptable. Therefore, for maintaining the area overhead within a range, the total improvement of NBTI delay is "restricted" to $targetImprv$ as mentioned. As a result, if the total improvement of NBTI delay is larger than $targetImprv$, we undo the last iteration and explore the solution space of last iteration by **increasing threshold parameter**, such that **fewer** NBTI-critical POs are selected for optimization in this iteration. The objective is to achieve an NBTI delay improvement less than but close to $targetImprv$. This idea about solution space exploration with various threshold parameters is detailed in the pseudo code of Algorithm 2.

Algorithm 1 NBTI-aware subject graph restructuring.

```
1: function NBTIBALANCEAIG($oriAig$, $thld$)
2:     // initialization
3:     $oriDelay \leftarrow$ NBTISTA($oriAig$)
4:     $currAig \leftarrow nbtiBalAig \leftarrow oriAig$
5:     repeat
6:         // save the results of previous iteration
7:         $currAig \leftarrow nbtiBalAig$
8:         $currDelay \leftarrow$ NBTISTA($currAig$)
9:         // detect NBTI critical POs and optimize their delays
10:        $nbtiBalAig \leftarrow \boxed{\text{NBTIBALANCE(currAig, thld)}}$
11:        $newDelay \leftarrow$ NBTISTA($nbtiBalAig$)
12:        // examine termination conditions
13:        $imprvIter \leftarrow$ COMPIMPRV($currDelay$, $newDelay$)
14:        $imprvTotal \leftarrow$ COMPIMPRV($oriDelay$, $newDelay$)
15:    until $\boxed{imprvIter < 0.1\% \textbf{ or } imprvTotal > 10\%}$
16:    // control $imprvTotal$ not exceed 10% by exploring the
           solution space of this iteration with different $thld$'s
17:    if $imprvTotal > 10\%$ then
18:        $nbtiBalAig \leftarrow$ EXPLDIFFTHLD($oriDelay$, $currAig$, $thld$)
19:    return $nbtiBalAig$
```

Algorithm 2 Explore the solution space of a specific iteration with various threshold parameters.

```
1: function EXPLDIFFTHLD($oriDelay$, $currAig$, $startThld$)
2:     $thld \leftarrow startThld$
3:     $nbtiBalAig \leftarrow currAig$
4:     repeat
5:         // use a bigger $thld$ to balance the tradeoff
6:         $thld \leftarrow thld + 0.01$
7:         // have explored all possible $thld$'s, discard
               the results of this iteration
8:         if $thld > 1$ then
9:             return $currAig$
10:        $nbtiBalAig \leftarrow$ NBTIBALANCE($currAig$, $thld$)
11:        $newDelay \leftarrow$ NBTISTA($nbtiBalAig$)
12:        $imprvTotal \leftarrow$ COMPIMPRV($oriDelay$, $newDelay$)
13:    until $imprvTotal \leq 10\%$
14:    return $nbtiBalAig$
```

B. Technology mapping

Given an NBTI-friendly subject graph (AIG), we propose an NBTI-aware technology mapping technique, which not only preserves the NBTI reduction gains from the previous stage, but also alleviates NBTI impact from the aspect of technology mapping. Additionally, the mapping selection step in it also considers the succeeding stage, smart pin reordering, ahead for maximally reducing the NBTI effect.

The technology mapper adopted in the work is based on a cut-based boolean matching method [1][3]. The mapper consists of five major steps: **(1)** *Compute k-feasible cuts.* A feasible cut of a node n in the AIG is a set of nodes C_n in the fanin cone of n such that any path from a PI to n passes through C_n. A k-feasible cut means the size of the cut must be less than or equal to k. A k-feasible cut is *redundant* if there exits a node in the cut whose value can be completely determined by the other nodes in it. For example, in Fig. 5, the set $\{a, b, c\}$ is a 3-feasible cut of node n, while the set $\{a, b, c, e\}$ is a redundant 4-feasible cut of node n because the value of node e can be determined by nodes b and c in the same cut. The redundant k-feasible cuts will not be considered during cut enumeration. The parameter k is heuristically set to 5 for the tradeoff of efficiency and effectiveness in the work. **(2)** *Compute the truth tables of cuts.* The local function of a node in terms of its cut is computed symbolically. With considering 5-feasible cut, the truth table (function) of each cut can be stored in a 32-bit integer, thus accelerating the symbolic function computation as well as the matching process in the next step. **(3)** *Perform Boolean matching.* Each node in the AIG might have more than one 5-feasible cut. For each cut, a matching gate, if existing, is selected from the library. **(4)** *Compute the best arrival time of each node.* The best arrival time of each node is computed and selected from all its matchings of cuts in a topological order. **(5)** *Select the best cover.* In a reverse topological order, which is from the POs to the PIs, the best matching gates are chosen using a delay-oriented method with the area constraint until all nodes in the AIG are covered.

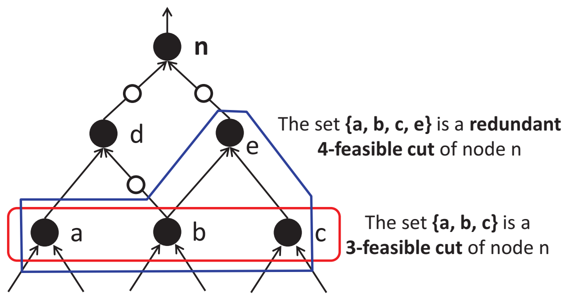
n
d
e
a
b
c
The set {a, b, c, e} is a redundant 4-feasible cut of node n
The set {a, b, c} is a 3-feasible cut of node n

Fig. 5. The redundant and irredundant feasible cuts of node n.

To reduce NBTI effect at this stage, the arrival time computation of matches should reflect NBTI-induced degradation accordingly. Given a matched gate and a cut, γ_{nbti} for the inputs are computed from their signal probabilities based on the NBTI modeling mentioned in Sec. II-B. Since only the rise delay would be affected by NBTI, two out of four timing arcs, **input rise to output rise** and **input fall to output rise**, are adjusted with considering γ_{nbti}.

Furthermore, to break a tie during the best cover selection, NOR or Or-And-Inverter (OAI) gates will be chosen with **high priorities** to benefit our next stage. The beneficial effects will be discussed in the next subsection. Our technology mapping stage produces an NBTI-tolerant mapped netlist to the next stage.

C. Smart pin reordering

Given an NBTI-tolerant mapped netlist, we propose a scalable pin reordering technique, named *smart pin*, to tweak the structure of the netlists for more NBTI effect reduction. Based on the discussion of stacking effect in Sec. II-C, it is intuitive to assign inputs to the PMOS transistor stack of a gate in descending order of input signal probabilities from the top to the bottom of pull-up network. Although this assigning order can result in the smallest NBTI degradation for the gate, it might not lead to the smallest signal arrival time at the gate's output. This is because the inputs with higher signal probabilities (smaller probabilities of being logic 0) might have larger signal arrival time. Therefore, to minimize the NBTI-degraded delay of the overall circuit, both signal probabilities and signal arrival time of inputs should be considered simultaneously for pin reordering technique. The previous work [22] considered the stacking effect in NOR gates only and only considered input arrival time to search a pin ordering that leads to the best timing **exhaustively**. Therefore, it might not be applicable for richer standard cell libraries, which have other gates with PMOS transistor stacking or gates with many pins.

The two major characteristics of the proposed smart pin reordering technique are applicability and scalability. For applicability, in addition to leveraging stacking effect of NOR gates, we also explored and leveraged the stacking effect in OAI gates, as marked in rectangles in Fig. 6, for NBTI reduction. For scalability, the smart pin reordering technique heuristically determines pin assignment by slack, which is defined as the difference between the required time and arrival time of gate's output signal for the timing path, rather than exhaustively searches like the previous work.

The scalable heuristic is described as follows. First, given the required time of the POs, the slack of each gate in the mapped netlist is computed using an NBTI-aware STA with considering signal probabilities of the gate inputs. Next, pins of PMOS stack(s) in NOR and OAI gates are reassigned based on the slack information in a topological order from the PIs to the POs. Specifically, the input with the smallest slack is assigned to the lowest PMOS transistor, and the other inputs are dealt with in the same way. The physical meaning behind this strategy is that a timing path through an input with a small slack is tight and more **fragile** to NBTI-induced delay degradation; therefore, assigning this input to the lower position of PMOS stack can protect the tight timing path against NBTI effect in the future.

In addition to the applicability and scalability of smart pin technique, in this coordinated approach, CSL's technology mapping stage chooses NOR and OAI gates with higher priorities when a selection ends in a tie, thus inducing more flexibility and increasing the effectiveness of this smart pin reordering technique.

IV. Experimental results

The CSL for NBTI reduction was implemented in C/C++ in ABC [1], which is a state-of-the-art logic synthesis and verification platform. The benchmarks are industry-strength

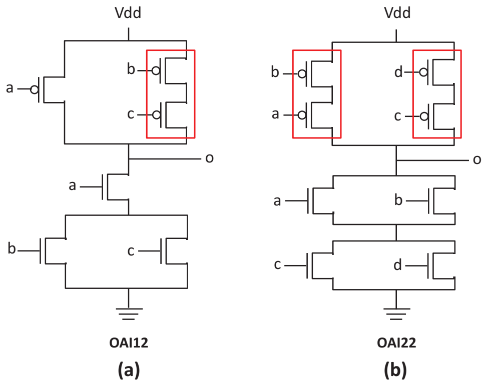

Fig. 6. The PMOS transistor stacking effect exists in the schematics of gates OAI12 and OAI22.

TABLE I
The Statistics of Benchmarks from ISPD'12 Contest [12]

Circuit	PI #	PO #	Comb. cell #	Seq. cell #	Total cell #
pci_bridge32	160	201	29844	3359	33203
DMA	683	276	23109	2192	25301
des_perf	234	140	102427	8802	111229
vga_lcd	85	99	147812	17079	164891
b19	22	25	212674	6594	219268
leon3mp	254	79	540352	108839	649191
netcard	1836	10	860949	97831	958780

designs from ISPD'12 contest [12] and benchmark statistics is listed in Table I. The standard cell library used in the experiments is a subset of library *mcnc.genlib* [15], which contains INV, NAND2, NAND3, NAND4, NOR2, NOR3, NOR4, AOI12, AOI22, OAI12, and OAI22. The technology process used is 32nm Predictive Technology Model (PTM) [13], and NBTI effect is considered at the end of a 5-year period. All experiments were run on a Linux machine with AMD Opteron 6276 16-Core 2.3GHz CPU and 128GB RAM.

The previous work [22], which is the most related to CSL, combines logic restructuring and pin reordering based on functional symmetry detection and transistor stacking effect to mitigate NBTI-induced delay degradation. Given a mapped netlist, it identifies functional symmetries using the concept of *supergates (SG)* [4], where a supergate is a group of connected gates that logically equals a big AND/OR gate. Having these SGs detected, [22] can swap wires inside supergates to improve NBTI delay without altering the functionality of netlist. The NBTI delay of the netlist will be improved iteratively until no further improvement is obtained. To extract SGs in a netlist, [22] first treats all POs as SG roots and assigns non-controlling values to them. Then backward implication is applied to each gate in a reverse topological order to determine the values of all inputs until **(1)** no more implication can be made or **(2)** the current gate is not fanout-free. The gates where backward implication stops are treated as new SG roots. The same backward implication is applied to those new SG roots with non-controlling values recursively until no more SGs can be detected. To compare [22] and CSL, we reimplemented [22] on the same platform ABC, and the results

TABLE II
THE COMPARISON OF [22] AND CSL ON NBTI REDUCTION OVER INDUSTRY-STRENGTH BENCHMARKS.

Circuit	Non-NBTI Optimization				[22]		CSL			[22] Performance	CSL Performance	
	Area	Nominal Delay	NBTI Delay	Aging (%)	NBTI Delay	Time (s)	Area	NBTI Delay	Time (s)	NBTI Delay Imprv. (%)	Area Overhead (%)	NBTI Delay Imprv. (%)
pci_bridge32	44000	20.5	22.26	8.59%	22.23	16.46	44195	19.61	3.35	0.13%	0.44%	11.90%
DMA	45490	18.2	19.87	9.18%	19.87	20.54	48528	18.73	4.13	0.00%	6.68%	5.74%
des_perf	169322	15.9	17.45	9.75%	17.39	219.51	176750	17.01	21.61	0.40%	4.39%	2.52%
vga_lcd	266398	15.8	17.23	9.05%	–	OOT	284537	15.64	29.51	–	6.81%	9.23%
b19	441778	61.3	66.60	8.65%	66.57	352.14	442631	63.38	103.14	0.05%	0.19%	4.83%
leon3mp	1250676	39.7	43.57	9.75%	–	OOT	1249915	40.17	2796.92	–	−0.06%	7.80%
netcard	1532384	29.6	32.44	9.59%	–	OOT	1517186	31.24	1503.62	–	−0.99%	3.70%
Ave.				9.22%						0.15%	2.49%	**6.53%**

[1] Area size is normalized by INV's size and reported by ABC [1].
[2] OOT denotes "Out Of Time" and the time limit is 5 hours.

TABLE III
THE STATISTICS OF SUPERGATES DETECTED BY [22] IN BENCHMARKS

Circuit	Supergate #	Max Size of SG	Wire # in Max-size SG
pci_bridge32	2021	11	37
DMA	1269	12	41
des_perf	6915	7	13
vga_lcd	1167	219	730
b19	15313	21	50
leon3mp	12545	225	735
netcard	59196	877	2669

[1] Supergate size is measured by the number of primitive gates in it.
[2] The number of wires in a SG includes inputs to SG and its internal wires.

were verified by the equivalent checking commands of ABC to guarantee the functional correctness of logic restructuring and pin reordering.

Table II shows the comparison of [22] and CSL on NBTI reduction over industry-strength benchmarks. Since CSL includes stages that optimize and remap the original benchmarks, to fairly compare the performance of [22] and CSL, the original benchmarks were first optimized and remapped by ABC to eliminate redundant logic. These new optimized benchmarks are our baseline results. The $threshold$ parameter of CSL is set to 0.97 empirically. Because of lack of real workloads, the signal probabilities are calculated using purely random simulation for both [22] and CSL. The NBTI delay is obtained by an NBTI-aware STA using the NBTI modeling in Sec. II-B. The aging degradation is the percentage of difference between the nominal delay and NBTI delay. Columns 1-5 list the basic information and NBTI-induced degradation of the baseline benchmarks. Columns 6-7 and Columns 8-10 list the results of [22] and CSL, respectively. The runtime is in seconds. Compared to the baseline, the NBTI delay improvement of both methods are listed in Columns 11 and 13, and the area overhead of CSL is listed in Column 12. There is no area overhead in [22], since it only swaps wires and does not introduce additional gates.

A. The Performance of [22]

As shown in Table II, we observed the performance of [22] on large-scale benchmarks is limited based on our reimplementation. After investigating the algorithm and benchmarks, the possible explanations for this phenomenon are in the following paragraphs.

First, as we aforementioned, the performance of after-TechMap works might be constrained by the results of technology mapping. Therefore, after benchmarks are optimized and remapped (e.g., by ABC in this work) for delay, there might be no much space left for further delay or NBTI delay improvement, thus affecting the performance of these works. However, this effect is more dramatic in [22] because it only performs wire swapping without inducing any additional gates.

Second, the size of supergates does matter to the performance due to the complexity of algorithm. The statistical information of supergates detected by [22] for each benchmark is listed in Table III. As shown in Table II, [22] cannot finish `vga_lcd`, `leon3mp`, and `netcard` within the specified time limit because these benchmarks have big SGs whose sizes are 219, 225, and 877, respectively. In Table III, we can know the number of wires in a SG is essentially proportional to its size. However, the number of valid combinations of wire swappings is indeed exponential to the number of wires. Therefore, when the size of a SG is enormous, [22] spends much time in exploring many possible combinations of wire swappings to find the one that reduces the NBTI delay of the SG most. Although [22] has proposed some heuristics to prune the exploring space, the number of wire swappings to try is still intractable in big SGs. Furthermore, large-scale benchmarks usually have more big SGs than small-scale ones; therefore, we could infer that large-scale benchmarks are not friendly to [22].

Third, the NBTI delay reduction is limited because not many SGs are located on the NBTI longest path of circuits. Please note that the standard cell library used in the paper is richer than the one used in [22]. Our library has two additional primitive gate types, OAI and AOI, which do not have non-controlling values and cannot perform backward implication. Additionally, large-scale benchmarks usually have many non-fanout-free gates because of logic sharing. Therefore, for `pci_bridge32`, `DMA`, `des_perf` and `b19`, [22] cannot find enough SGs on the longest path to improve NBTI delay due to keeping restarting SG expansion on new SG roots, while encountering OAI, AOI, and non-fanout-free gates. This scattered SG structure compromises the performance of [22]

TABLE IV
THE EFFECTIVENESS OF NBTI-AWARE SUBJECT GRAPH RESTRUCTURING OF CSL.

Circuit	CSL (AIG restr.)		CSL (AIG restr.) Performance	
	Area	NBTI Delay	Area Overhead (%)	NBTI Delay Imprv. (%)
pci_bridge32	44166	19.86	0.38%	10.78%
DMA	48494	19.16	6.60%	3.57%
des_perf	180731	17.04	6.74%	2.35%
vga_lcd	284397	15.64	6.76%	9.23%
b19	442878	63.81	0.25%	4.19%
leon3mp	1254236	40.89	0.28%	6.15%
netcard	1617043	31.85	5.52%	1.82%
Ave.			3.79%	**5.44%**

* Area size is normalized by INV's size and reported by ABC [1].

drastically. Take DMA as an example, we found that there is only one SG of size 2 on the NBTI longest path. What's worse, this SG, consisting of only a NAND2 and an INV, has no valid wire swappings that can improve the NBTI delay. Therefore, no NBTI improvement can be made in DMA.

B. The Performance of CSL

According to Table II, CSL can achieve 6.53% NBTI delay improvement with merely 2.49% area overhead on average. Thanks to NBTI-aware subject graph restructuring technique, NBTI-friendly graphs can be generated by considering NBTI effect and controlling area overhead as early as possible. Given the NBTI-friendly graphs, our technology mapping technique can have more flexibility in choosing the matching gates that can reduce NBTI delay most as well as not sacrificing area too much. For example, CSL can mitigate the NBTI effect of b19 and pci_bridge32 with insignificant area overhead (< 1 %). This contribution is even more significant for large benchmarks like netcard and leon3mp, for which CSL can improve NBTI delay without any area overhead or even a little area reduction. Interestingly, for pci_bridge32 and vga_lcd, the improved NBTI delay is slightly better than the original nominal delay, possibly due to the restructuring at the subject graph stage. Additionally, the runtimes of benchmarks also demonstrate the scalability of CSL. All the benchmarks can finish in several seconds to an hour. Please note that the major part of runtime is spent on technology mapping, which is also a necessary effort for non-NBTI-aware approaches. We could find that CSL has a great ability to handle large-scale benchmarks as well as has no constraints on the libraries used.

To demonstrate the benefit of considering NBTI effect at an earlier stage, we conducted another experiment that only applies NBTI-aware subject graph restructuring technique in CSL without using NBTI-aware technology mapping and smart pin reordering techniques. Results in Table IV demonstrate that NBTI-aware subject graph restructuring alone can achieve 5.44% NBTI delay improvement on average, which are the major contributions of the complete CSL. These results support the idea of early consideration of NBTI effect. Although the techniques of the rest stages, NBTI-aware technology mapping and NBTI-aware smart pin reordering, seemingly provide 1.09% ($6.53\% - 5.44\% = 1.09\%$) improvement in NBTI delay, they help reduce area overhead from 3.79% to 2.49%. Therefore, the coordination among the techniques at different stages can result in the best performance of CSL.

V. CONCLUSION

This paper proposes a coordinated and scalable logic synthesis approach, CSL, to address NBTI effect, which is a major cause of aging and reliability issues in nanometer IC designs. It consists of NBTI-aware subject graph restructuring, technology mapping, and smart pin reordering techniques at different stages to form a coordinated NBTI-aware logic synthesis approach. Experimental results demonstrated the capability and scalability of CSL to mitigate the NBTI effect with acceptable area overhead and runtime.

ACKNOWLEDGMENT

This work was supported by NSF CCF 12-55846 and SRC 2013-TJ-2420.

REFERENCES

[1] Berkeley Logic Synthesis and Verification Group, *ABC: A system for sequential synthesis and verification*. http://www.eecs.berkeley.edu/∼alanmi/abc/
[2] A. Chakraborty and D. Z. Pan, "Skew management of NBTI impacted gated clock trees," in *Proc. of ISPD*, pp. 127-133, 2010.
[3] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," in *Proc. of ICCAD*, pp. 519-526, 2005.
[4] C.-W. Chang, C.-K. Cheng, P. Suaris, M. Marek-Sadowska, "Fast Post-Placement Rewiring Using Easily Detectable Functional Symmetries," in *Proc. of DAC*, pp. 286-289, 2000.
[5] M. Ebrahimi, F. Oboril, S. Kiamehr, and M. B. Tahoori, "Aging-aware logic synthesis," in *Proc. of ICCAD*, pp. 61-68, 2013.
[6] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: lessons learnt and future trends," in *Proc. of DAC*, pp. 1-10, 2013.
[7] K. Keutzer, "DAGON: Technology binding and local optimizations by DAG matching," in *Proc. of DAC*, pp. 617-623, 1987.
[8] Y. Kukimoto, R. K. Brayton, and P. Sawkar, "Delay-optimal technology mapping by DAG covering," in *Proc. of DAC*, pp. 348-351, 1998.
[9] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "NBTI-aware synthesis of digital circuits," in *Proc. of DAC*, pp. 370-375, 2007.
[10] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. CAD*, Vol. 21(12), pp. 1377-1394, 2002.
[11] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Trans. CAD*, vol. 16, no. 8, pp. 813-833, 1997.
[12] M. M. Ozdal et al., "The ISPD-2012 discrete cell sizing contest and benchmark suite," in *Proc. of ISPD*, pp. 161-164, 2012.
[13] "Predictive technology model," Device Group at Arizona State University, Available at http://www.eas.asu.edu/∼ptm
[14] S. Roy and D. Z. Pan, "Reliability aware gate sizing combating NBTI and oxide breakdown," in *Proc. of VLSID*, pp. 38-43, 2014.
[15] E. M. Sentovich et al., "SIS: A system for sequential circuit synthesis," *Technical Report*, UCB/ERI, M92/41, ERL, Dept. of EECS, UC Berkeley, 1992.
[16] L. Stok, M. A. Iyer, and A. J. Sullivan, "Wavefront technology mapping," in *Proc. of DATE*, pp. 531-53, 1999.
[17] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, Vol. 94, No. 1, Jul. 2003.
[18] E. Takeda and N. Suzuki, "An empirical model for device degradation due to hot-carrier injection," *IEEE Electron Device Lett.*, vol. EDL-4, pp. 111-113, 1983.
[19] V. Reddya, A. T. Krishnana, A. Marshalla, J. Rodrigueza, S. Natarajanb, T. Rosta, and S. Krishnan, "Impact of negative bias temperature instability on digital circuit reliability," in *Proc. Int. Reliability Phys. Symp.*, pp. 248V54, 2002.
[20] R. Vattikonda, W. Wang, and Y. Cao, "Modeling and minimization of PMOS NBTI effect for robust nanometer design," in *Proc. of DAC*, pp. 1047-1052, 2006.
[21] W. Wang et al., "The impact of NBTI on the performance of combinational and sequential circuits," in *Proc. of DAC*, pp. 364V369, 2007.
[22] K.-C. Wu and D. Marculescu, "Joint logic restructuring and pin reordering against NBTI-induced performance degradation," in *Proc. of DATE*, pp. 75-80, 2009.
[23] K.-C. Wu and D. Marculescu, "Aging-aware timing analysis and optimization considering path sensitization," in *Proc. of DATE*, pp. 1-6, 2011.
[24] X. Yang and K. Saluja, "Combating NBTI degradation via gate sizing," in *Proc. of ISQED*, pp. 47-52, 2007.