MAUI: Making Aging Useful, Intentionally

Tien-Hung Tseng, Shou-Chun Li, and Kai-Chiang Wu, *Member, IEEE*

***Abstract*—Device aging, which causes significant loss on circuit performance and lifetime, has been a primary factor in reliability degradation of nanoscale designs. In this paper, we propose to take advantage of aging-induced clock skews (i.e., make them useful for aging tolerance) by manipulating these time-varying skews to compensate for the performance degradation of logic networks. The goal is to assign achievable/reasonable aging-induced clock skews in a circuit, such that its overall performance degradation due to aging can be minimized, that is, the lifespan can be maximized. On average, 24.95% aging tolerance can be achieved with insignificant design overhead. Moreover, we also apply V_{th} assignment to further mitigate the aging-induced degradation of logic networks. Averagely, 37.89% aging tolerance can be achieved when aging manipulation and V_{th} assignment are applied.**

I. INTRODUCTION

The design and manufacturing of semiconductor devices have recently experienced dramatic innovations, at the cost of downgrading reliability of nanoscale integrated circuits (ICs) and system-on-chips (SOCs). The 2013 ITRS [1] projects that the long-term reliability of sub-100nm technology nodes can reach a noteworthy order of 10^3 FITs (failures in 10^9 hours). Soft errors, thermal and aging effects are some of the major challenges driving reliability-aware IC/SOC design techniques. With the continuous scaling of transistor dimensions, device aging, which causes temporal performance degradation and potential wear-out failure, is becoming increasingly dominant for lifetime reliability concerns because of limited timing margins [2]. Therefore, the need of a verification and optimization flow considering aging effects emerges as a key factor in guaranteeing reliable and sustainable operation over a required lifespan. *Bias temperature instability* (BTI) is known for prevailing over other device aging phenomena, in terms of dependence on the scaling of nanometer technologies. BTI [3] is a MOSFET aging phenomenon that occurs when transistors are stressed under bias (positive or negative, i.e., $V_{gs} = \pm V_{dd}$) at elevated temperature. As a result of the dissociation of Si-H bonds along the Si-SiO_2 interface, BTI-induced MOSFET aging manifests itself as an increase in the threshold voltage (V_{th}) and decrease in the drive current (I_{ds}) [4], which in turn lengthen the propagation delays of logic gates/paths. Experiments on MOSFET aging [5] indicate that BTI effects grow exponentially with higher operating temperature and thinner gate oxide. If the thickness of gate oxide shrinks down to 4nm, the circuit performance can be degraded by as much as 15% after 10 years of stress and lifetime will be dominated by BTI [6]. In contrast, when the stress condition is relaxed (V_{gs} = 0), the aging mechanism can be recovered partially [7] and the threshold voltage decreases toward the nominal value by more than 75% if the recovery phase lasts sufficiently long [8]. In addition to the aging effect on the logic gates/paths of a circuit, the impact of aging on the clock network should not be ignored. Considering both "the aging of logic networks" and "the aging of clock networks" is essential since unbalanced aging of clock networks can greatly affect circuit performance by inducing clock skews, as a result of non-uniform increases in clock latency from the clock source to different terminals. Prior work on addressing such clock skews due to aging (i.e., aging-induced clock skews) mainly attempts to balance the aging effects on various clock sub-networks, such that aging-induced clock skews can be minimized [9]–[11]. However, suppressing aging-induced clock skews may be difficult and costly, especially for clock-gated designs where the rate of aging varies from one clock sub-network to another [12].

Then think about the following question: if one has to work hard on (and incur significant overhead for) minimizing aging-induced clock skews but it turns out that there still exist stubborn non-zero skews, why don't we try to take advantage of them? We do; we propose to intentionally make aging-induced clock skews useful for aging tolerance. More specifically, we compensate for the performance degradation (due to aging) of logic networks by exploring "useful" aging-induced clock skews, based on the concept of time borrowing. Note that the rate of aging depends on the stress time, defined as the amount of time during which a PMOS/NMOS transistor is stressed under negative/positive bias. For clock drivers (comprising pairs of inverters), their stress times are proportional to the duty cycle of the clock signal. The key idea of our framework is to manipulate the rates of aging on different clock branches, by changing the duty cycle of a clock waveform delivered to each of the clock branches. The proposed framework succeeds in mitigating effective aging-induced performance degradation with little design penalty.

II. RELATED WORK AND PAPER CONTRIBUTION

A. Previous Work on Aging-Aware Optimization

To deal with aging phenomena, traditional design methods adopt guard-banding by adding extra timing margins, which in practice imply over-design and may be expensive. To avoid overly conservatism, the mitigation of aging-induced performance degradation can be formulated as a timing-constrained area minimization problem with consideration of aging effects. Existing aging-aware techniques basically follow this formulation. A novel technology mapper considering signal probabilities for NBTI was developed in [13]. On average, 10% area recovery and 12% power saving are accomplished, as compared to the most pessimistic case assuming static NBTI on all PMOS transistors in the design. The authors

Tien-Hung Tseng, Shou-Chun Li, and Kai-Chiang Wu are with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan (e-mail: kcw@cs.nctu.edu.tw).

of [14] proposed a gate sizing algorithm based on Lagrangian relaxation. An average of 8.7% area penalty is required to ensure reliable operation for 10 years. Other methods related to gate or transistor sizing can be found in [15], [16].

Aforementioned methods focus on mitigating the aging of logic networks only. There are some works [9]–[11] addressing the aging problem of clock networks. Methodology of [9] is based on V_{th} assignment for clock buffers. The authors of [10] and [11] explore the use of alternative clock gating cells for clock-gated designs. On the other hand, two new clock gating cells were presented in [12] to balance the delay degradation of clock signal propagation, which reduces aging-induced clock skews between ungated and gated clock branches. The above works [9]–[12] aim to minimize aging-induced clock skew instead of making it useful (i.e., assign specific clock skews in the designs to mitigate the aging-induced performance degradation).

B. Paper Contribution

In this paper, we propose an optimization framework for aging tolerance. Our proposed framework manipulates the rates of aging on different clock branches, which is achieved by changing the duty cycle of a clock waveform delivered to each of the clock branches. In addition, we involve the mechanism of V_{th} assignment in the framework to further improve aging tolerance of designs. The contributions and advantages of this work are threefold:

- ***Exploration of aging-induced clock skews for aging tolerance:*** Existing work on addressing aging-induced clock skews mainly attempts to minimize the skews. This paper presents the first work on *exploring "useful" clock skews (i.e., making them useful)* for aging tolerance*.
- ***Problem formulation based on Boolean satisfiability and optimal solutions:*** The proposed formulation of making clock skew useful is transformed into a Boolean satisfiability (SAT) problem, and its optimal solution can be efficiently found by a SAT solver such as MiniSat.
- ***Low design overhead and little design modification:*** Restrained by the synthesized clock tree whose topology and structure are basically determined, our post-CTS (clock tree synthesis) framework does not involve aggressive modification and thus does not incur significant design overhead.

III. MOTIVATING EXAMPLE

This section introduces the example, motivating our idea of making clock skew useful, by manipulating the aging behaviors of clock paths, introduced in Section III-A. Moreover, high-V_{th} assignment for clock buffers are also involved in the example to enhance the benefit of useful clock skew for aging tolerance, introduced in Section III-B.

*We do not minimize "absolute" performance degradation by directly mitigating the aging of logic networks; instead; we minimize "effective" performance degradation by V_{th} assignment and manipulation of aging rates on clock networks to tolerate expected logic's aging based on timing borrowing, as a result of newly-designated clock skews in designs. Of course, one can mitigate the logic's aging itself by using existing techniques [13]–[16] before applying the proposed framework. This is however beyond the scope of this work and thus not particularly addressed in this paper.

A. Manipulation of Aging Behaviors of Clock Paths

Consider the circuit in Figure 1(a) where FF_X, FF_Y and FF_Z are three edge-triggered flip-flops, and L_{XY} and L_{YZ} are the corresponding in-between logic networks. Other notations to be used later are listed in Figure 1(b).

For each pair of flip-flops (e.g., FF_i and FF_j) between which there exists at least one logic path from FF_i to FF_j, the following setup-time (Equation (1)) and hold-time (Equation (2)) constraints need to be satisfied:

$$C_i + T_{cq} + D_{ij} + T_{su} < C_j + T_c \quad (1)$$

$$C_i + T_{cq} + d_{ij} < C_j + T_h \quad (2)$$

Assume that, at year 10, D_{ij} is degraded by 15%, and both T_{cq} and T_{su} increase by 20%. By using the predictive model presented in [8], [17], we can accurately derive the aging of C_i and C_j due to the regularity and predictability of a typical clock waveform of 50% duty cycle. In the process technology used (TSMC 45nm GP standard cell series), C_i and C_j are degraded by 13% under 10-year BTI, i.e., C_i and C_j become 1.13X larger.

To be aging-aware for the example circuit in Figure 1(a), we have to consider aforementioned aging factors in the setup-time constraints on L_{XY} and L_{YZ}:

$$L_{XY}: \quad \mathbf{1.13}C_X + 1.2T_{cq} + 1.15D_{XY} + 1.2T_{su} < \mathbf{1.13}C_Y + T_c \quad (3)$$

$$L_{YZ}: \quad \mathbf{1.13}C_Y + 1.2T_{cq} + 1.15D_{YZ} + 1.2T_{su} < \mathbf{1.13}C_Z + T_c \quad (4)$$

where $C_X = C_Y = C_Z = 100$, $T_{cq} = 8$, $T_{su} = 2$, $D_{XY} = 90$ and $D_{YZ} = 80$, as shown in Figure 1(a).

By re-arranging Equations (3) and (4):

$$L_{XY}: T_c > 115.5$$

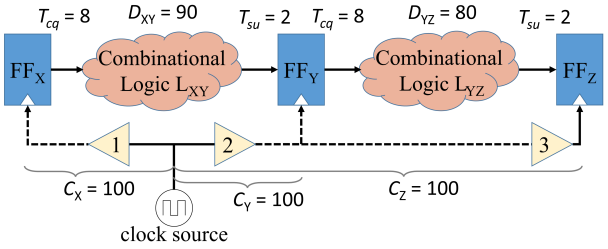
$$L_{YZ}: T_c > 104$$

Therefore, the clock period needs to be larger than 115.5 (dominated by L_{XY}) to ensure no setup-time violation over a required lifespan of 10 years. For brevity, we omit the discussion on hold-time constraints, which in our work are actually formulated to ensure no existence of racing due to short paths.

To minimize required T_c under aging, we use *duty-cycle converter* (DCC) to manipulate the aging rates of clock branches. We insert one 20% DCC at the input of buffer 1, and another 80% DCC at the input of buffer 2. The 20% (80%) DCC can decrease (increase) the stress times of downstream clock buffers by converting the clock duty cycle to 20% (80%), from a typical duty cycle of 50%. Therefore, 20% DCC can mitigate/decelerate the aging of C_X and 80% DCC can aggravate/accelerate the aging of C_Y. In the case of no DCC, C_X and C_Y are degraded by 13% under 10-year BTI in TSMC 45nm GP standard cell series, while 20% and 80% DCCs will degrade C_X and C_Y by 9% and 16%, respectively, assuming that the clock paths from the clock source to FF_X and FF_Y are disjoint.

Consider the new aging factors (due to effects of various clock duty cycles) in the setup-time constraints on L_{XY} and L_{YZ}:

$$L_{XY}: \quad \mathbf{1.09}C_X + 1.2T_{cq} + 1.15D_{XY} + 1.2T_{su} < \mathbf{1.16}C_Y + T_c \quad (5)$$



(a) Example

Symbol	Description
FF_i	Edge-triggered flip-flop, e.g., $i \in$ {X, Y, Z, ...}
L_{ij}	Corresponding in-between logic network, e.g., $i, j \in$ {X, Y, Z, ...}
T_C	Clock period
C_i	Clock arrival time to flip-flop i (FF_i)
D_{ij}/d_{ij}	Longest/shortest path delay of L_{ij}
T_{cq}	Clock-to-output delay
V_{th}^h	High threhold voltage
T_{su}/T_h	Setup/hold time

(b) Notations

Figure 1: Illustrative example and notations for the proposed framework based on DCC deployment and V_{th} assignment

$$\mathrm{L_{YZ}}: \quad \mathbf{1.16}C_Y + 1.2T_{cq} + 1.15D_{YZ} + 1.2T_{su} < \mathbf{1.16}C_Z + T_c \quad (6)$$

By re-arranging Equations (5) and (6):

$$\mathrm{L_{XY}}: T_c > 108.5$$
$$\mathrm{L_{YZ}}: T_c > 104$$

As it can be seen, we can reduce the required T_c from 115.5 to 108.5 (dominated by $\mathrm{L_{XY}}$ still), by adding two DCCs in the existing synthesized clock tree to create aging-induced clock skews. The skew for $\mathrm{L_{XY}}$ (between $\mathrm{FF_X}$ and $\mathrm{FF_Y}$), quantified as $1.16C_Y$ minus $1.09C_X$, is useful/beneficial and accounts for the reduction of required T_c. A certain level of aging tolerance is thus achieved because aging-induced performance degradation of D_{XY} (plus T_{cq} and T_{su} actually) can be tolerated, by exploring such useful aging-induced clock skews.

B. High-V_{th} Assignment for Clock Buffers

For the circuit in Figure 1(a), the approach of high-V_{th} (V_{th}^h) assignment for clock buffers is involved to explore more beneficial clock skew, i.e., further optimize/reduce required T_c.

Follow the previous DCC deployment for manipulation of aging, that is, 20%DCC and 80% DCC inserted at the inputs of buffer 1 and 2, respectively, we can further alter the threshold voltage V_{th} to V_{th}^h from buffer 2 to $\mathrm{FF_Y}$ and $\mathrm{FF_Z}$ so as to achieve a better utilization of clock skew. To include the timing information of V_{th}^h buffers in the setup-time constraint, we assume that, the fresh/intrinsic delay of V_{th}^h buffer is 1.2X longer than that of nominal buffer deducted from power-law model, and aging rates of V_{th}^h buffers, receiving clock duty cycles of 20%, 40%, 50% and 80%, are 0.5%, 4.1%, 5.4% and 8.2%*, respectively. Consider the new aging factors in the setup-time constraints on $\mathrm{L_{XY}}$ and $\mathrm{L_{YZ}}$:

$$\mathrm{L_{XY}}: \quad \mathbf{1.09}C_X + 1.2T_{cq} + 1.15D_{XY} + 1.2T_{su} < \mathbf{1.28}C_Y + T_c \quad (7)$$

$$\mathrm{L_{YZ}}: \quad \mathbf{1.28}C_Y + 1.2T_{cq} + 1.15D_{YZ} + 1.2T_{su} < \mathbf{1.28}C_Z + T_c \quad (8)$$

By re-arranging Equations (7) and (8):

$$\mathrm{L_{XY}}: T_c > 96.5$$
$$\mathrm{L_{YZ}}: T_c > 104$$

*The aging rates of V_{th}^h buffers are lower than those of nominal buffers, because the higher/lower V_{th} leads to lower/higher aging rate.

Apparently, the required T_c is further reduced/optimized from 108.5 to 104 (dominated by $\mathrm{L_{YZ}}$ rather than $\mathrm{L_{XY}}$ in the first example), by inserting two DCCs and assigning the V_{th} of certain buffers to high value. As it can be seen, the skew for $\mathrm{L_{XY}}$, which equals $1.28C_Y$ minus $1.09C_X$, is larger than that in the first example. Therefore, the new skew for $\mathrm{L_{XY}}$ is more useful/beneficial and accounts for the better optimization of required T_c.

One may note that *clock skew scheduling* (CSS) [18], which derives unequal delays for all clock branches prior to *clock tree synthesis* (CTS), can also optimize a circuit for aging tolerance. However, the optimization potential of general CSS is limited since it is difficult to precisely implement a wide range of clock delays during the early design phase. In contrast, post-CTS clock skew scheduling based on buffer insertion is another option. We will demonstrate that, if buffer insertion is employed to match our optimization results based on DCC insertion, the number of inserted buffers is usually much larger than the number of inserted DCCs. Also, as described later in Section IV-C, the overhead of a single DCC can be diminished by integrating a DCC with its downstream buffer, which further reveals the cost effectiveness of our proposed DCC-based framework.

IV. PRELIMINARIES

A. Aging Prediction Model

Before discussing the proposed framework, we briefly introduce the aging (BTI degradation) model for logic gates/networks [8], [17] used in our paper. This model enables us to analyze the long-term behavior of BTI-induced MOSFET degradation, with both aging and recovery mechanisms taken into account. First, the degradation of threshold voltage at a given time t can be predicted as:

$$\Delta V_{th} = \left(\frac{\sqrt{K_v^2 \cdot T_{clk} \cdot \alpha}}{1 - \beta_t^{1/2n}} \right)^{2n} \quad (9)$$

where K_v is a function of temperature, electrical field, and carrier concentration, α is the stress probability, and n is the time exponential constant, 0.2 for the used technology. The detailed explanation of each parameter can be found in [8].

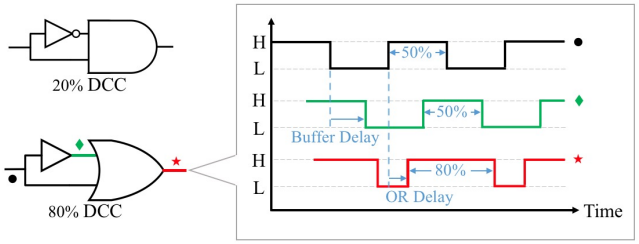


Figure 2: Construction of DCC and duty cycle transformation

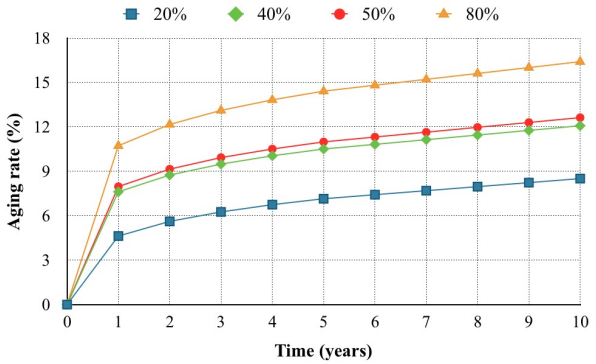


Figure 3: Aging rates of buffers receiving different clock duty cycles

Next, the authors of [17] simplify this predictive model to be:

$$\Delta V_{th} = b \cdot \alpha^n \cdot t^n = b \cdot (\alpha \cdot t)^n \tag{10}$$

where $b = 3.9 \times 10^{-3} V \cdot s^{-1/5}$.

Finally, the rising/falling propagation delay of a gate through the degraded P-type/N-type MOSFET can be derived as a first-order approximation:

$$\tau' = \tau + a \cdot (\alpha \cdot t)^n \tag{11}$$

where τ is the intrinsic delay of the gate without BTI degradation and a is a constant.

We apply Equation (11) to calculate the delay of each gate under BTI, and further estimate the performance of a logic circuit. The coefficient a in Equation (11) for each gate type and each input pin is extracted by fitting HSPICE simulation results in 45nm, Predictive Technology Model (PTM). The simplified long-term model successfully predicts the MOSFET degradation, with less than 5% loss of accuracy against cycle-by-cycle simulations [8].

B. Duty-Cycle Converter (DCC)

Duty cycle is the percentage of one period in which a signal is high (i.e., logic 1). The aging of logic gates highly depends on the stress time [8]. For a clock buffer on the clock tree, its stress time is proportional to the clock duty cycle. Therefore, by adjusting the clock duty cycle, we can manipulate the aging of clock buffers and then control the effective degradation of logic paths. Figure 3 shows the aging rates of clock buffers receiving different clock duty cycles.

The unit we use to change the clock duty cycle is duty-cycle converter (DCC), which can converts the duty cycle of a clock signal to a smaller/larger one (e.g., $50\% \rightarrow 20\%$ or $50\% \rightarrow 80\%$), as shown in Figure 2. Take 80% DCC's duty cycle transformation as an example, the delayed waveform (green waveform) is separated from the source clock waveform (black waveform). Then, the two waves are combined with the OR gate to obtain a new waveform (red waveform) of 80% duty cycle. Once a DCC is inserted into the clock tree, the downstream sub-tree of the DCC insertion point will receive a clock signal whose duty cycle is no longer 50%. This way, aging rate manipulation of downstream clock buffers can be achieved.

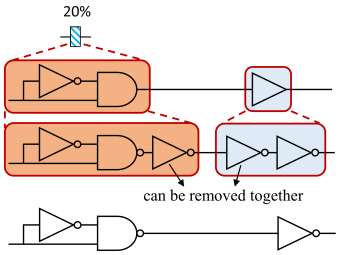


Figure 4: Practical considerations for DCC insertion

C. Practical Considerations

The diagram on the top of Figure 4 shows the primitive design of a DCC, consisting of an inverter and an AND gate. In practice, the AND gate is implemented by a NAND gate feeding an inverter. As mentioned earlier, when inserting a DCC, it is inserted at the input of a buffer, which is a pair of inverters in practice. Therefore, we can actually use the diagram on the bottom of Figure 4 to realize the insertion of a DCC. More specifically, we use it as a new cell to "replace" a buffer when a DCC is needed. By doing so, the cost of a single DCC can be significantly reduced and not much more expensive than the cost of inserting a buffer for clock skew scheduling.

V. PROPOSED FRAMEWORK

In the proposed framework, aging tolerance is achieved by specifically assigning useful/beneficial slacks into the designs, based on timing borrowing. The beneficial slacks is progressively created by different aging behaviors of the two clock paths C_i and C_j, due to the different duty cycles of clock signal caused by DCCs. Moreover, V_{th}^h assignment for clock buffers is also involved in the proposed framework to explore more beneficial slacks, further improving the aging tolerance of designs. The overall flow of the proposed framework is depicted in Figure 5, where we focus the following three issues:

1) **Minimization of clock period:** The clock period can be minimized since the performance degradation of the logic circuit is "tolerated" as a result of useful clock skews. The minimum required clock period thus implies maximum level of aging tolerance. As depicted in Figure 5, a binary search for optimal clock period is involved in the proposed framework.

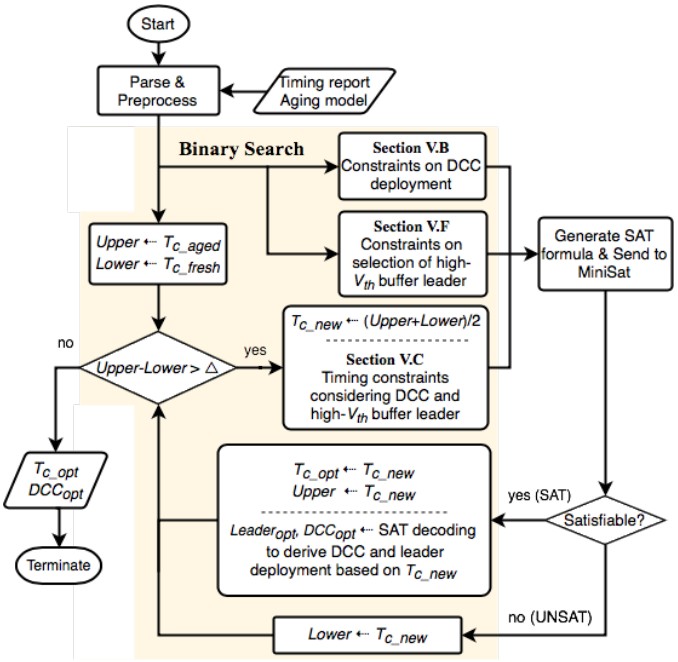


Figure 5: The overall flow of MAUI

2) **DCC deployment:** The problem of DCC deployment is formulated as a Boolean Satisfiability (SAT) problem. Therefore, the key of our framework is to represent the problem in *conjunctive normal form* (CNF). A CNF representation is a conjunction of one or more clauses, where each clause is a disjunction of one or more Boolean variables. Thanks to the efficiency of existing SAT solver, the solution can be obtained efficiently. The end result of this formulation is the locations (in the existing clock tree) of DCCs, such that the required clock period of the given circuit under n-year BTI is minimized when aging-induced clock skews are considered.
3) **Additive improvement by V_{th}^h assignments for clock buffers:** Fragmentary V_{th} assignment for clock buffers may decrease the feasibility/manufacturability of designs. Thus, we select certain clock buffers as V_{th}^h buffer leaders, indicating where we begin continuously assigning high V_{th} toward flip-flops. The problem of selecting V_{th}^h buffer leaders from existing clock buffers is also formulated as a SAT-based problem, represented in CNF and solved by existing SAT solver.

The section is organized as follows: Section V-A explains how the proposed problem of DCC deployment/insertions are encoded with Boolean variables. Section V-B and Section V-C describe three major components, DCC constraints and timing constraints, for our SAT-based formulation and how they are translated into legal SAT formula, i.e., CNF representation. Finally, Section V-D-V-G introduce the additive improvements by V_{th}^h assignment for buffers.

A. Boolean Encoding for DCC Deployment

The problem of DCC deployment needs to be encoded into Boolean representation before being transformed into a SAT-based formulation. Assume that a total of N types of DCCs can be chosen. Including the DCC-free case where no DCC is inserted, there are (N + 1) possibilities of DCC insertion for each clock buffer. We denote a clock buffer by p ($1 \leq p \leq P$) where P is the total count of clock buffers and p is buffer index. For each clock buffer, there exist two types of Boolean variables, $B_{p,q}$ ($1 \leq q \leq Q$, $Q = \lceil \lg(N+1) \rceil$), where $\{B_{p,1}, B_{p,2}, \ldots, B_{p,Q}\}$ encode the aforementioned (N + 1) possibilities of DCC insertion at the input of buffer p.

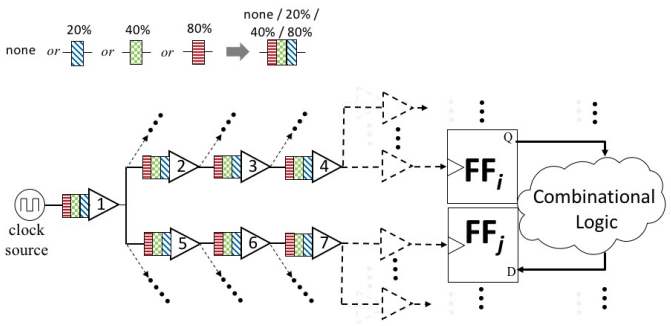


Figure 6: Generalized DCC insertion for a target pair of flip-flops

Without loss of generality, we assume N = 3. Thus, there are three types of DCCs, assumed to be 20%, 40%, and 80% DCCs, as shown in Figure 6. Therefore, three Boolean variables are used for encoding four possibilities of DCC at any buffer. The four possibilities can be encoded as follows:

	DCC type	$\{B_{p,2}, B_{p,1}\}$
(1)	None	{0, 0}
(2)	20%	{0, 1}
(3)	40%	{1, 0}
(4)	80%	{1, 1}

For example, in Figure 7(b), a 20% DCC and an 80% DCC are inserted at buffer 3 and buffer 5, respectively. Therefore, $\{B_{3,2}, B_{3,1}\}$ = {0, 1}, $\{B_{5,2}, B_{5,1}\}$ = {1, 1}, and $\{B_{p,2}, B_{p,1}\}$ = {0, 0} for p = 1, 2, 4 , 6 or 7.

The 20% DCC will mitigate the aging of buffer 3 and its downstream buffers, while the 80% DCC will aggravate the aging of buffer 5 and its downstream buffers. It can be found that, if a DCC is deployed deep in the clock tree (i.e., close to the flip-flops), the count of affected buffers is limited and thus the overall impact of aging mitigation/aggravation on C_i/C_j may be insignificant, diminishing the benefit from deploying DCCs for aging tolerance. To avoid the phenomena, we set a rule of prohibiting DCC deployment at a clock tree level larger/deeper than a specified boundary. This rule also greatly reduces the complexity of our SAT-based formulation because a significant fraction of buffers are excluded from being considered inserted DCC at their inputs. For example, in Figure 6, those dashed buffers and their downstream buffers are excluded.

B. Constraints on DCC Deployment

Figure 6 shows a generalized example of DCC insertion for a pair of flip-flops (FF_i and FF_j) where there exist aging-critical paths from FF_i to FF_j. A path is defined as an aging-critical path if, in the presence of aging, it is possible to

determine the clock period of the circuit. Every pair of flip-flops between which there exist aging-critical paths needs to be considered and here, we use this generalized example to illustrate our SAT-based formulation.

In Figure 6, buffers $1-7$ are candidate locations for DCC insertions and according to the encoding scheme explained in Section V-A, two Boolean variables are introduced for each of the seven buffers to encode four possibilities of DCC insertions. Considering all seven buffers, there are a total of 16,384 (= 4^7) possibilities for DCC insertions, just for this pair of flip-flops. This makes the SAT-based formulation intractable due to the explosion of clause count. Therefore, we set up the following constraints on DCC insertion:

DCC constraint: At most one DCC on a single clock path (from the clock source to one of the flip-flops)

In order to ensure no more than one DCC on any clock path, we can use the Boolean variables introduced for DCC insertion, i.e., $B_{p,q}\,(1 \le p \le P, 1 \le q \le Q = \lceil \lg(N+1) \rceil)$, to generate some clauses which suppress the occurrence of having two DCCs on a clock path. Consider buffer 2 (encoded by $\{B_{2,2}, B_{2,1}\}$) and buffer 3 (encoded by $\{B_{3,2}, B_{3,1}\}$) in Figure 6. If there is a DCC at buffer 2 (i.e., $\{B_{2,2}, B_{2,1}\} \not\equiv \{0,0\}$), then there must no DCC at buffer 3 (i.e., $\{B_{3,2}, B_{3,1}\} \equiv \{0,0\}$), and vice versa. The constraint can be formally written as:

$$(\{B_{2,2}, B_{2,1}\} \equiv \{0,0\}) \vee (\{B_{3,2}, B_{3,1}\} \equiv \{0,0\})$$

Next, it can be translated into 4 CNF clauses:

$$(\neg B_{2,1} \vee \neg B_{3,1}) \wedge (\neg B_{2,1} \vee \neg B_{3,2})$$
$$\wedge (\neg B_{2,2} \vee \neg B_{3,1}) \wedge (\neg B_{2,2} \vee \neg B_{3,2})$$

Any pair of buffers along a single clock path should be constrained in this way. Among buffers $1-7$ in Figure 6, there are 12 pairs: $\langle 1,2\rangle$, $\langle 1,3\rangle$, $\langle 1,4\rangle$, $\langle 1,5\rangle$, $\langle 1,6\rangle$, $\langle 1,7\rangle$, $\langle 2,3\rangle$, $\langle 2,4\rangle$, $\langle 3,4\rangle$, $\langle 5,6\rangle$, $\langle 5,7\rangle$, $\langle 6,7\rangle$. Each pair translates to 4 clauses and a total of 48 clauses will be generated accordingly.

With DCC constraints and corresponding clauses, we can drastically reduce the possibilities of DCC insertions to be formulated. In the above example where the 48 clauses associated with DCC constraints are generated, the possibility count of DCC insertions drops from 16,384 to 103. In the next subsection, we describe what the 103 possibilities are and how they are translated into the final CNF representation.

C. Timing Constraints

Given a pair of flip-flops, if there exists one logic path between them, the timing (i.e. setup-time and hold-time) constraints must be met based on the inequalities (1) and (2). Different types of DCCs reveal different influence on clock latency. Consider the lifespan specification of 10 years in Figure 7(b): the delay of each clock buffer is changed after 10 years. The clock latency of FF_i (i.e., C_i) is the sum of clock buffer delays from clock source to FF_i:

$$C_i = \tau_1 + \tau_2 + \tau_3 + \tau_4 + \cdots$$
$$C_j = \tau_1 + \tau_5 + \tau_6 + \tau_7 + \cdots$$

where τ_k is the delay of buffer k. Consider aging effects on C_i and C_j:

$$C_{i\_aged} = 1.13 \times (\tau_1 + \tau_2) + 1.09 \times (\tau_3 + \tau_4 + \cdots)$$
$$C_{j\_aged} = 1.13 \times \tau_1 + 1.16 \times (\tau_5 + \tau_6 + \tau_7 + \cdots)$$

where C_{i_aged} and C_{j_aged} denote aged C_i and C_j, respectively. Next, we apply Equations (1) and (2) to check whether timing constraints will be violated under this DCC deployment.

Timing constraint: No existence of timing violation

Given one clock period T_c derived by binary search, one aging-critical path, and its associated clock network, all possible DCC deployments can be classified into 3 classes according to the number of DCCs used. Furthermore, due to the aforementioned DCC constraints, the SAT solver will only output a DCC deployment where there does not exist more than one DCC along any clock path. Thus, in the following discussion, the deployment with more than one DCC along a single clock path can be ignored. In each class, if the DCC deployment causes a timing violation within 10 years (i.e., the lifespan specification), then the deployment will be transformed into CNF clauses, such that the solver will not output the deployment as results. Here, we explain the generation of CNF clauses by using the example in Figure 7.

Class 1: No DCC is inserted on either clock path

Consider the situation that no DCC is inserted at buffers $1-7$. If it causes a timing violation along the aging-critical path within 10 years, then the Boolean representation of the deployment,

$$(\{B_{1,2}, B_{1,1}\} \equiv \{0,0\}) \wedge (\{B_{2,2}, B_{2,1}\} \equiv \{0,0\}) \wedge \cdots$$
$$\wedge (\{B_{7,2}, B_{7,1}\} \equiv \{0,0\}),$$

equivalent to the following CNF clause:

$$(B_{1,2} \vee B_{1,1} \vee B_{2,2} \vee B_{2,1} \vee \cdots \vee B_{7,2} \vee B_{7,1}),$$

should be generated such that the solver will not output the corresponding deployment in the result if the CNF is satisfiable. In this case, a total of 1 CNF clause is generated.

Class 2: Inserting one DCC

This class can be further classified into 2 sub-classes based on the location of inserted DCCs.

Class 2-1: Inserting one DCC on the common clock path

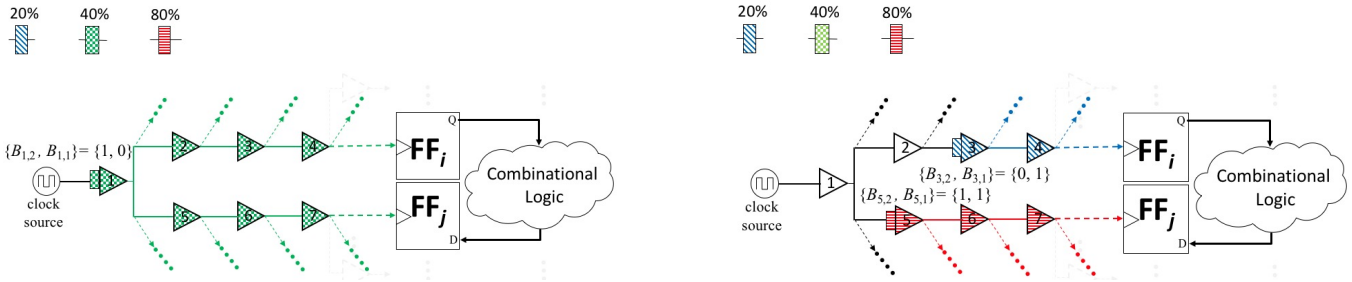
In Figure 7, buffer 1 is on the common clock path. Consider the DCC insertion shown in Figure 7(a): if the insertion of a 40% DCC at buffer 1 causes a timing violation within 10 years, then the Boolean representation of the DCC deployment,

$$(\{B_{1,2}, B_{1,1}\} \equiv \{1,0\}) \wedge (\{B_{2,2}, B_{2,1}\} \equiv \{0,0\}) \wedge \cdots$$
$$\wedge (\{B_{7,2}, B_{7,1}\} \equiv \{0,0\}),$$

equivalent to the following clause:

$$(\neg B_{1,2} \vee B_{1,1} \vee B_{2,2} \vee B_{2,1} \vee \cdots \vee B_{7,2} \vee B_{7,1}),$$

should be generated such that the solver will not output the deployment in the result if the CNF is satisfiable. Given that there are 3 choices of DCCs, a total of 3 CNF clauses will be



(a) Class 2-1: one DCC on the common clock path (e.g., at buffer 1)

(b) Class 2-2: one DCC on one of the divergent clock paths, or class 3: two DCCs, one on each of the divergent clock paths

Figure 7: Examples of DCC insertion

generated in the worst case.

Class 2-2: Inserting one DCC on one of the divergent clock paths

This class targets buffers 2, 3, 4, 5, 6, 7. If the insertion of a 20% DCC at buffer 3 causes a timing violation within 10 years, then the Boolean representation of the DCC deployment,

$$(\{B_{3,2}, B_{3,1}\} \equiv \{0,1\}) \wedge (\{B_{1,2}, B_{1,1}\} \equiv \{0,0\}) \wedge \cdots \wedge (\{B_{7,2}, B_{7,1}\} \equiv \{0,0\}),$$

equivalent to the following CNF clause:

$$(B_{3,2} \vee \neg B_{3,1} \vee B_{1,2} \vee B_{1,1} \vee B_{2,2} \vee B_{2,1} \vee B_{3,2} \cdots \vee B_{7,2} \vee B_{7,1}),$$

should be generated such that the solver will not output the deployment in the result if the CNF is satisfiable. This class includes 6 candidates: buffers 2, 3, 4, 5, 6, 7, and each has 3 choices of DCCs. Therefore, a total of 18 CNF clauses will be generated in the worst case.

Class 3: Inserting two DCCs on two clock paths respectively

Given the DCC deployment in Figure 7(b) (a 20% DCC inserted at buffer 3 and a 80% DCC inserted at buffer 5), if it causes a timing violation along the aging-critical path within 10 years, then the Boolean representation of the deployment,

$$(\{B_{3,2}, B_{3,1}\} \equiv \{0,1\}) \wedge (\{B_{5,2}, B_{5,1}\} \equiv \{1,1\}),$$

equivalent to the following CNF clause,

$$(B_{3,2} \vee \neg B_{3,1} \vee \neg B_{5,2} \vee \neg B_{5,1}),$$

should be generated such that the solver will not output the deployment in the result if the CNF is satisfiable.

Class 3 considers two buffer locations to insert DCCs, one among buffers {2, 3, 4} and the other one among buffers {5, 6, 7}; thus, there are totally 9 combinations of buffer locations. Each combination includes two buffers and thus 9 possibilities of choosing one specific DCC for each of the two buffers. Therefore, a total of 81 CNF clauses will be generated in the worst case.

Considering all of the above cases, a maximum number of 1 + 3 + 18 + 81 = 103 clauses can be derived. This is based on the existence of 48 clauses introduced in Section V-B.

D. Additive Improvement in Aging Tolerance: V_{th}^h Assignment for Clock Buffers

In this subsection, the approach of V_{th}^h assignment for clock buffers is involved in the proposed framework to further improve the aging tolerability of designs. Because the fragmentary V_{th} assignment for clock buffers may decrease the feasibility/manufacturability of the proposed framework, we make a rule/definition:

Definition: Once a clock buffer is selected as the V_{th}^h buffer leader, the V_{th} of the selected buffer and its downstream buffers will be assigned to V_{th}^h.

In this way, we avoid the cases that the V_{th} of serial buffers along a clock path interlace between high and nominal V_{th} (i.e., V_{th}^h and V_{th}^n). Such a rule can suppress the occurrence of fragmentary V_{th} assignments for clock buffers and thus can increase the feasibility of the proposed framework.

Conceptually, there exists one difference between DCCs and V_{th}^h buffer leaders. DCCs are physical gates inserted at the inputs of clock buffers to manipulate the aging behaviors of downstream buffers. However, V_{th}^h buffer leaders are selected from the existing clock buffers where their threshold voltage will be assigned to V_{th}^h toward flip-flops (exlusive).

E. Boolean Encoding for V_{th}^h Buffer Leaders and DCCs

The problem of selecting V_{th}^h buffer leaders is also formulated as a SAT-based problem, solved by existing SAT solver, as we do for DCC deployment. The end result of the formulation involves the selected clock buffers and locations of DCCs. Assume that there are three types of DCCs (20%, 40%, and 80%) and one type of V_{th}^h buffer leader. Therefore, three Boolean variables are used for encoding eight possibilities of DCC insertions and leader selections at any buffer†. Consider the clock buffer p, the eight possibilities can be encoded as follows:

	Leader type	DCC type	$\{B_{p,3}, B_{p,2}, B_{p,1}\}$
(1)	None	None	{0, 0, 0}
(2)	None	20%	{0, 0, 1}
(3)	None	40%	{0, 1, 0}
(4)	None	80%	{0, 1, 1}
(5)	High-V_{th} (V_{th}^h)	None	{1, 0, 0}
(6)	V_{th}^h	20%	{1, 0, 1}
(7)	V_{th}^h	40%	{1, 1, 0}
(8)	V_{th}^h	80%	{1, 1, 1}

†Any clock buffer can be inserted DCC at its input and can be selected as a V_{th}^h buffer leader.

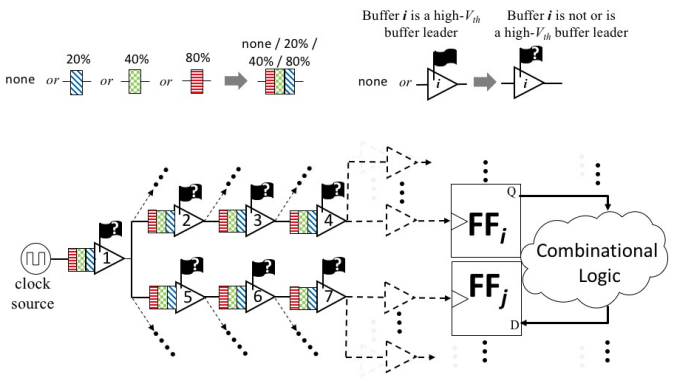


Figure 8: Generalized DCC insertion and V_{th}^h buffer leader selection, for a target pair of flip-flops

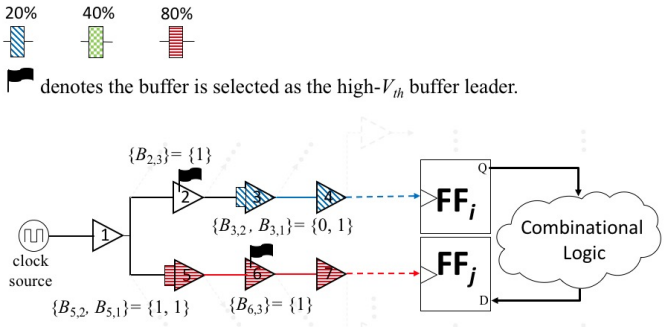


Figure 9: Example of DCC insertion and V_{th}^h buffer leader selection

Note that, if the buffer, located deep in the clock tree, is selected as the leader, then the count of affected buffers is limited, diminishing the benefit from V_{th}^h assignment for aging tolerance. Therefore, as we do for DCC deployment, we similarly set a rule of prohibiting selecting buffer leaders at a clock tree level larger/deeper than a a specified boundary.

F. Constraints on the Selection of V_{th}^h Buffer Leader

Figure 8 shows a generalized example of DCC insertion and leader selection. It is worth reminding that, because of DCC constraints, the possibility count of DCC deployments has been dropped from 16,384 to 103. However, for each DCC deployment, there are still up to 128 (2^7) possibilities of leader selections, for this pair of flip-flops. Hence, we set up the following constraints on the selections of V_{th}^h buffer leaders:

Leader constraints: At most one buffer on a single clock path is selected as V_{th}^h buffer leader.

The constraints are analogous to the aforementioned DCC constraints. The corresponding clauses can be generated in a similar manner, but are not shown in the paper for brevity. In the example, the 12 clauses associated with leader constraints will be generated, in such a way that, the possibility count of leader selections is dropped from 128 to 17. Subsequently, the remaining 17 possibilities are considered in the timing constraints, introduced in the next subsection.

G. Timing Constraints Considering DCC Deployment and Selection of V_{th}^h Buffer Leader

The timing constraints are analogous to the former timing constraints, while here we more consider the influence of V_{th}^h buffer leader on clock latency. Remind that, the former timing constraints are divided into 3 classes according to the used DCC count. Because we simultaneously consider the DCC deployment and leader selection in the existing clock network, each of the former 3 classes is further divided into 3 subclasses, in terms of the leader count (0, 1 and 2). Therefore, the timing constraints are divided into 9 (3×3) subclasses, according to the used DCC count and leader count.

For simplicity, we make an illustrative example from one of the 9 subclasses: 2 used DCCs and 2 V_{th}^h buffer leaders in the clock network associated aging-critical paths in Figure 9, where 20% and 80% DCC are inserted at the inputs of buffers 3 and 5, respectively, and buffers 2 and 6 are selected as V_{th}^h buffer leaders, implying that the V_{th} of buffer set {2, 3, 4} and buffer set {6, 7} are assigned to V_{th}^h because of the leader buffer 2, 6, respectively. In the case, if it causes a timing violation along the aging-critical path, the Boolean representation of the DCC deployment and leader selection,

$$(\{B_{1,3}, B_{1,2}, B_{1,1}\} \equiv \{0,0,0\}) \wedge (\{B_{2,3}, B_{2,2}, B_{2,1}\} \equiv \{1,0,0\})$$
$$(\{B_{3,3}, B_{3,2}, B_{3,1}\} \equiv \{0,0,1\}) \wedge (\{B_{4,3}, B_{4,2}, B_{4,1}\} \equiv \{0,0,0\})$$
$$(\{B_{5,3}, B_{5,2}, B_{5,1}\} \equiv \{0,1,1\}) \wedge (\{B_{6,3}, B_{6,2}, B_{6,1}\} \equiv \{1,0,0\})$$
$$\wedge (\{B_{7,3}, B_{7,2}, B_{7,1}\} \equiv \{0,0,0\}),$$

equivalent to the following CNF clause:

$$(B_{1,3} \vee B_{1,2} \vee B_{1,1} \vee \neg B_{2,3} \vee B_{2,2} \vee B_{2,1} \vee B_{3,3} \vee B_{3,2} \vee \neg B_{3,1}$$
$$\vee B_{4,3} \vee B_{4,2} \vee B_{4,1} \vee B_{5,3} \vee \neg B_{5,2} \vee \neg B_{5,1} \vee \neg B_{6,3} \vee B_{6,2} \vee B_{6,1}$$
$$\vee B_{7,3} \vee B_{7,2} \vee B_{7,1})$$

should be generated such that the solver will not output the corresponding DCC deployment and leader selection in the result if the CNF is satisfiable. In this case, a total of 1 CNF clause is generated.

Note that, given the DCC deployment, if the leader count is 2, there exist 9 (3×3) possibilities of leader selection (one among buffers $\{2, 3, 4\}$, the other one among buffers $\{5, 6, 7\}$). If the leader count is 1/0, there exist 7/1 possibilities of leader selections. Hence, there totally exist 17 ($9+7+1$) possibilities of leader selections, for each DCC deployment.

VI. EXPERIMENTAL SETTING, RESULTS AND DISCUSSION

A. Experimental Setting

The proposed framework for aging tolerance is implemented in C++ and the SAT-based formulation is solved by MiniSat on a 2.83GHz Intel Quad-Core CPU workstation running Linux. The benchmark circuits are chosen from the IWLS'05 and ISCAS'89 suites. The technology used is TSMC 45nm GP standard cell series.

Under 10-year BTI, the aging rates of clock buffers were obtained from HSPICE. The aging rates of clock buffers with duty cycles of 20%, 40%, 50%, and 80% are 8.51%, 12.08%,

13.51%, and 16.41% respectively and the aging rate of logic is obtained by using the predictive model presented in [8], [17] (detailed in Section IV-A).

B. Experimental Results

Table I reports the experimental results and information of each benchmark. Columns 2 to 5 show the total number of gates, total numbers of flip-flops, total numbers of clock buffers, and maximum level of the clock tree in each benchmark respectively. Column 6 demonstrates the fresh clock period that is the circuit delay without aging, denoted by T_{c_fresh}. Column seven demonstrates the clock period of the circuit under 10-year aging, denoted by T_{c_aged}. Column eight demonstrates the optimum clock period of the circuit under 10-year aging after applying our framework, denoted by $T_{c\_aged\_opt}$. A shorter clock period under aging implies better circuit performance and higher level of aging tolerance. Column 9 demonstrates the used DCC count. Column 10 demonstrates the runtime and the Column 11 demonstrates the improvement, i.e., the level of aging tolerance which is calculated as:

$$1 - (T_{c\_aged\_opt} - T_{c\_fresh})/(T_{c\_aged} - T_{c\_fresh})$$

For benchmark *des_perf*, T_{c_fresh} is 773.9ps and T_{c_aged} is 897.8ps, which means after 10-year aging the clock period of circuit will increase by 123.9ps. With DCC insertion using the proposed framework, the clock period achieved is 849.9ps, an increment of 76ps against T_{c_fresh} (38.66% improvement). As shown in Table I, the improvement ranges from 7.38% to 49.77% and is 24.95% on average. The number of inserted DCCs is between 2 (for benchmark *s38584*) to 17 (for benchmark *des_perf*), implying very limited degree of circuit modification and insignificant design overhead. Moreover, as it can be seen in Column 12 to 16, when V_{th}^h assignment for clock buffers is involved in the proposed framework, the resulting framework can give rise to lower clock periods, implying better improvement in aging tolerance.

C. Discussion: DCC Redeployment due to V_{th}^h Assignment

As we can see, the DCC counts in Column 13 are different from those in Column 9, implying that DCCs are redeployed in the clock tree while V_{th}^h assignment is involved in the framework. More specifically, when V_{th}^h assignment is considered, some clock buffers become candidate buffers to be inserted DCCs at their inputs, because timing constraints (i.e., setup-time and hold-time) are met based on the inequality Equation (1) and (2). In this way, DCCs can be redeployed to obtain lower clock periods, exploring better improvement in aging tolerance.

D. Discussion: Increase in Runtime

In addition, the run time of the framework, involving V_{th}^h assignment, increases due to the possibility explosion of DCC deployment and leader selection. To be specific, given a pair of flip-flops and associated clock paths, we need to consider the various possibilities of leader selections, for each DCC deployment. Therefore, the total count of DCC deployment and leader selection is equal to the combination of DCC deployment plus leader selection, i.e., DCC possibilities multiplied by the leader counterparts, accounting for the increase of run time.

Even though the runtime increase while V_{th} assignment is involved, the resulting framework is still practical for aging tolerance because it at most takes 1370 seconds for a comparative design (e.g., netcard).

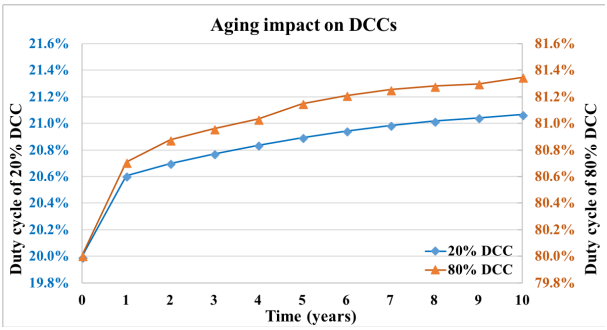


Figure 10: Aging impact on 20%/80% DCC under BTI

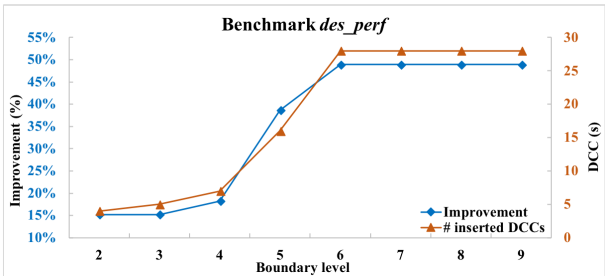


Figure 11: Improvement/Cost versus clock tree level considered

E. Discussion: Aging Impact on DCC

Figure 10 shows the change in the duty cycle of a 20%/80% DCC over 10-year aging. The y axis on the left represents the duty cycle of a 20% DCC, and the one on the right represents the duty cycle of an 80% DCC. As it can be seen, the growth in both cases are marginal: $20\% \rightarrow 21.07\%$ for a 20% DCC and $80\% \rightarrow 81.35\%$ for an 80% DCC, which in turn should not affect the benefit of our proposed framework significantly.

F. Discussion: Depth Boundary for DCC Deployment

As mentioned in Section V, inserting DCCs deep in the clock tree is less effective. For benchmark *des_perf*, we considered the deployment of DCCs at the upper half of the clock tree (i.e., level 1 to 5) and achieved 38.66% improvement in terms of aging tolerance. As demonstrated in Figure 11, if we expand the boundary of DCC deployment from level 1 in the clock tree to level 10 progressively, we can gain a considerable improvement from level 1 to 6; however, from 7 to 10, the improvement become stagnant, but more DCCs are required.

Table I: Results of aging tolerance

Benchmark	# Gates	# FFs	# Clock Buffers	Max Tree Level	No aging/ Fresh	10-year Aging									
							DCC Deployment				DCC Deployment & High-V_{th} Assignment				
					T_{c_fresh} (ps)	T_{c_aged} (ps)	$T_{c\_aged\_opt}$ (ps)	# DCCs	Runtime (s)	Improve (%)	$T_{c\_aged\_opt}$ (ps)	# DCCs	# High-V_{th} Buffers / # Total Buffers (ratio%)	Runtime (s)	Improve (%)
des_perf	7401	8802	3416	11	773.9	897.8	849.9	16	16.4	38.66%	816.8	35	56/3416 (1.64%)	685.3	65.38%
leo3mp	526297	108839	4970	10	3016.7	3530.6	3458.5	11	24.9	14.03%	3406.1	8	19/4970 (0.38%)	923.2	24.23%
netcard	561091	97831	7904	12	3367.0	3940.2	3897.9	1	30.7	7.38%	3860.1	1	7/7904 (0.09%)	1370.5	13.97%
vga_lcd	101496	17079	4346	10	864.2	1013.7	994.1	13	32.8	13.11%	981.4	19	74/4346 (1.70%)	1285.4	21.61%
s13207	2627	625	180	8	776.3	891.7	873.3	1	0.7	15.94%	857.7	1	7/180 (3.89%)	11.7	29.46%
s15850	2967	513	100	5	1011.1	1165.6	1089.9	3	0.2	49.00%	1071.7	3	4/100 (4.00%)	0.6	60.78%
s35932	6466	1728	411	6	822.1	950.5	886.6	5	1.3	49.77%	865.8	5	4/411 (5.11%)	8.5	65.97%
s38417	8422	1564	376	6	798.1	933.2	909.8	13	1.5	17.32%	901.8	13	29/376 (7.71%)	18.5	23.24%
s38584	6861	1275	557	10	731.8	842.2	820.8	2	1.4	19.38%	802	4	9/557 (5.21%)	35.9	36.41%
AVG.										24.95%			3.01%		37.89%

VII. CONCLUSION

In this paper, we propose a novel framework, successfully taking advantage of aging-induced clock skews by inserting limited count of DCCs into the clock tree, to enhance circuit aging tolerance. We transform the problem to a Boolean satisfiability formulation which is then solved by using MiniSat. Experiments demonstrate that our framework gains an average of 24.95% aging tolerance via inserting at most 15 DCCs. Furthermore, we include the mechanism of V_{th} assignment in the framework, to further improve circuit aging tolerance, by assigning small fraction of clock buffers to high V_{th}. The problem of selecting buffers to be assigned to high V_{th} is also formulated as a Boolean satisfiability problem, solved by MiniSat. Experiments shows that the framework, which both considers DCC deployment and V_{th}^h assignment for clock buffers, can result in an average of 37.89% aging tolerance, via inserting at most 35 DCCs and assigning 3.01% of buffers to high-V_{th}.

REFERENCES

[1] *International technology roadmap for semiconductor*, 2013.

[2] J. W. McPherson, "Reliability challenges for 45nm and beyond," in *Proc. of DAC*, Jul. 2006, pp. 176–181.

[3] 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, pp. 1–18, Jul. 2003.

[4] J. H. Stathis and S. Zafar, "The negative bias temperature instability in MOS devices: A review," *Microelectronics Reliability*, vol. 46, no. 2, pp. 270–286, Feb.–Apr. 2006.

[5] S. Chakravarthi *et al.*, "A comprehensive framework for predictive modeling of negative bias temperature instability," in *Proc. of Int'l Reliability Physics Symp. (IRPS)*, Apr. 2004, pp. 273–282.

[6] N. Kimizuka *et al.*, "The impact of bias temperature instability for direct-tunneling ultra-thin gate oxide on MOSFET scaling," in *Proc. of Symp. on VLSI Technology*, Jun. 1999, pp. 73–74.

[7] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "An analytical model for negative bias temperature instability," in *Proc. of ICCAD*, Nov. 2006, pp. 493–496.

[8] W. Wang *et al.*, "The impact of NBTI effect on combinational circuit: Modeling, simulation, and analysis," *IEEE TVLSI*, vol. 18, no. 2, pp. 173–183, Feb. 2010.

[9] J. Chen and M. Tehranipoor, "A novel flow for reducing clock skew considering NBTI effect and process variations," in *Proc. of ISQED*, Mar. 2013, pp. 327–334.

[10] S.-H. Huang *et al.*, "Low-power anti-aging zero skew clock gating," *ACM TODAES*, vol. 18, no. 2, p. 27, Mar. 2013.

[11] A. Chakraborty and D. Z. Pan, "Skew management of NBTI impacted gated clock trees," *IEEE TCAD*, vol. 32, no. 6, pp. 918–927, Jun. 2013.

[12] L. Lai *et al.*, "BTI-gater: An aging-resilient clock gating methodology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 2, pp. 180–189, Jun. 2014.

[13] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "NBTI-aware synthesis of digital circuits," in *Proc. of DAC*, Jun. 2007, pp. 370–375.

[14] B. C. Paul *et al.*, "Temporal performance degradation under NBTI: Estimation and design for improved reliability of nanoscale circuits," in *Proc. of DATE*, Mar. 2006, pp. 780–785.

[15] K. Kang *et al.*, "Efficient transistor-level sizing technique under temporal performance degradation due to NBTI," in *Proc. of ICCD*, Oct. 2007, pp. 216–221.

[16] X. Yang and K. Saluja, "Combating NBTI degradation via gate sizing," in *Proc. of ISQED*, Mar. 2007, pp. 47–52.

[17] W. Wang *et al.*, "An efficient method to identify critical gates under circuit aging," in *Proc. of ICCAD*, Nov. 2007, pp. 735–740.

[18] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. on Computers*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[19] L. Li, Y. Lu, and H. Zhou, "Optimal multi-domain clock skew scheduling," in *Proc. of DAC*, Jun. 2011, pp. 152–157.

[20] A. F. G. et al., "Early selection of critical paths for reliable nbti ag-ing-delay monitoring," *IEEE Transactions on Very Larger Scale Integration (VLSI) Systems*, 2016.