Making Aging Useful by Recycling Aging-Induced Clock Skew

TIEN-HUNG TSENG, National Chiao Tung University
KAI-CHIANG WU, National Chiao Tung University

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 and recycling 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 effective performance degradation due to aging can be tolerated, that is, the lifespan can be maximized. On average, 25.04% aging tolerance can be achieved with insignificant design overhead. Moreover, we employ V_{th} assignment on clock buffers to further tolerate the aging-induced degradation of logic networks. When V_{th} assignment is applied on top of aforementioned aging manipulation, the average aging tolerance can be enhanced to 35.96%.

CCS Concepts: • **Hardware → Robustness**; **Hardware reliability**; **Aging of circuits and systems**;

Additional Key Words and Phrases: Clock network, Aging, Degradation, Reliability

1 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 2015 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 effect 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 [14]. Therefore, the need of a verification and optimization flow considering aging effect 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 [16] 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}) [17], which in turn lengthens the propagation delays of logic gates/paths. Experiments on MOSFET aging [4] 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 [9]. In contrast, when the stress condition is relaxed ($V_{gs} = 0$),

1084-4309/2018/0-ART0 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the aging mechanism can be recovered partially [10] and the threshold voltage bounces toward the nominal value by more than 75% if the recovery phase lasts sufficiently long [19].

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 flip-flops. 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 branches, such that aging-induced clock skews can be minimized [3, 5, 8]. 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 branch 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/recycling "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 principle 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. Also, we incorporate the idea of V_{th} assignment in our framework to explore more useful clock skews for aging tolerance. More specifically, the threshold voltage of certain clock buffers/drivers are assigned the other specific value. Consequently, clock latencies (from clock source to flip-flops) are changed, such that useful clock skew can be obtained. In this paper, the two ideas of (i) aging rate manipulation and (ii) V_{th} assignment are jointly integrated to recycle useful aging-induced clock skews in the clock network, such that aging-induced performance degradation can be tolerated effectively.

2 RELATED WORK AND PAPER CONTRIBUTION

2.1 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 [11]. 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 of [15] 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 [18, 20].

Aforementioned research work focus on mitigating the aging of logic networks only. There are some studies [3, 5, 8] addressing the aging problem of clock networks. The methodology of [5] is based on V_{th} assignment for clock buffers. The authors of [3, 8] 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

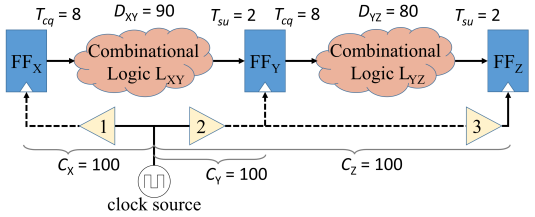
clock skews between ungated and gated clock branches. However, the studies [3, 5, 8, 12] aim to minimize aging-induced clock skews instead of making them useful.

2.2 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, by changing the duty cycle of a clock waveform delivered to each of the clock branches. In addition, we employ V_{th} assignment on clock buffers to achieve additive aging tolerance. 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/recycling "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 aging 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.

3 MOTIVATING EXAMPLE



(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
T_{su}/T_h	Setup/hold time

(b) Notations

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

This section introduces the example of motivating our idea of making aging useful. In Section 3.1, we explore and recycle the useful aging-induced clock skew by manipulating the aging rates of clock paths. In Section 3.2, high-V_{th} assignment for clock buffers is incorporated to further enhance the benefit of aging-induced clock skews for aging tolerance.

[‡‡]We do not minimize "absolute" performance degradation by directly mitigating the aging of logic networks; instead; we minimize "effective" performance degradation by manipulating the aging of clock networks to tolerate expected logic's aging based on timing borrowing, as a result of aging-induced clock skews. Of course, one can mitigate the logic's aging itself by using existing techniques [11, 15, 18, 20] before applying the proposed framework. This is however beyond the scope of this work and thus not particularly addressed in this paper.

3.1 Manipulating Aging Rates 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:

$$C_i + T_{cq} + D_{ij} + T_{su} < C_j + T_c \tag{1}$$

$$C_i + T_{cq} + d_{ij} < C_j + T_h \tag{2}$$

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 [2, 18, 19], 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}:

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

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

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.

For the objective of minimizing required T_c under aging, we insert one 20% *duty-cycle converter* (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}:

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

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

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

$L_{XY}: T_c > 108.5$

$L_{YZ}: T_c > 104$

As it can be seen, we can reduce the required T_c from 115.5 to 108.5 (dominated by L_{XY} still), by adding two DCCs in the existing synthesized clock tree to create aging-induced clock skews. The

skew for L_{XY} (between FF_X and 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.

3.2 Assigning High-V_{th} to Clock Buffers

For the circuit in Figure 1(a), the technique of high-V_{th} assignment for clock buffers is incorporated to explore more beneficial clock skews and further reduce the required T_c.

Given that a 20%DCC and an 80% DCC have been inserted at the inputs of buffers 1 and 2, respectively, we can alter buffer 2 and all its downstream buffers, by assigning them a higher threshold voltage, so as to achieve better utilization of clock skews. To include the timing information of high-V_{th} buffers in the setup-time constraints, the fresh/intrinsic delay of a high-V_{th} buffer is assumed to be 1.15X longer than that of a nominal-V_{th} buffer, and aging rate of high-V_{th} buffer, receiving a clock duty cycle of 80%, is 8%. Consider the new aging factors in the setup-time constraints on L_{XY} and L_{YZ}:

$$L_{XY}: \quad \mathbf{1.09}C_X + 1.2T_{cq} + 1.15D_{XY} + 1.2T_{su} < (\mathbf{1.15 \times 1.08})C_Y + T_c \tag{7}$$

$$L_{YZ}: \quad (\mathbf{1.15 \times 1.08})C_Y + 1.2T_{cq} + 1.15D_{YZ} + 1.2T_{su} < (\mathbf{1.15 \times 1.08})C_Z + T_c \tag{8}$$

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

$L_{XY}: T_c > 100.3$

$L_{YZ}: T_c > 104$

Apparently, the required T_c can be further reduced/optimized from 108.5 to 104 (dominated by L_{YZ} rather than L_{XY} as in Section 3.1), by inserting two DCCs and assigning a higher V_{th} to certain buffers. As it can be seen, the skew for L_{XY}, which equals $(1.15 \times 1.08)C_Y$ minus $1.09C_X$, is larger than that in Section 3.1. Therefore, the new skew for L_{XY} is more useful/beneficial and accounts for better optimization of required T_c.

One may note that *clock skew scheduling* (CSS) [6], 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 CTS [13].

4 PRELIMINARIES

4.1 Aging Prediction Model

Before discussing the proposed framework, we briefly introduce the aging (BTI degradation) model for logic gates/networks [2, 7, 18, 19] 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 consideration. First, the degradation of threshold voltage at a given time t can be predicted as:

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

where ΔV_{th_bti} is the BTI-induced V_{th} shift, K_v is a function of threshold voltage (V_{th}), 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 [19].

Next, as previously indicated in [7, 19] , the variation in V_{th} can be compensated by the BTI effect. That is, the transistor with a higher (lower) fresh V_{th} ages at a lower (higher) rate and thus, the

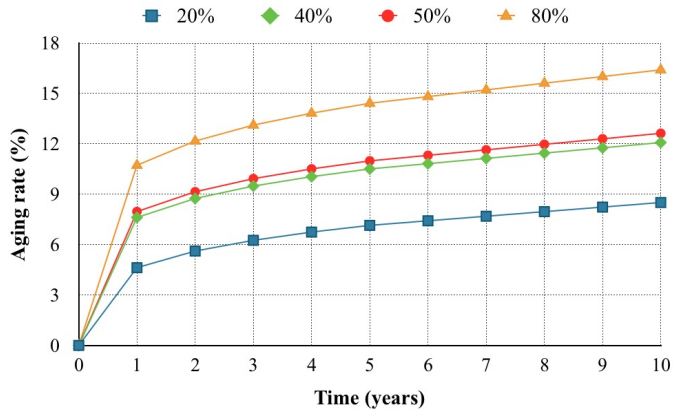


Fig. 2. Aging rates of buffers receiving different clock duty cycles

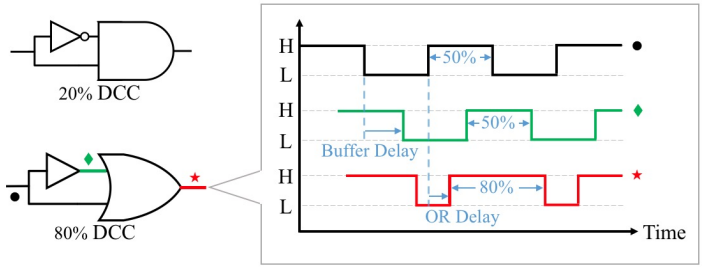


Fig. 3. Construction of DCC and duty-cycle transformation

high V_{th} and the low V_{th} tend to converge toward the nominal case as the stress of BTI continues. The phenomenon is concretely modeled as below [7]:

$$\Delta V_{th\_bti} = (1 - S_v \cdot \Delta V_{th\_fresh}) \cdot A \cdot \alpha^n \cdot t^n \tag{10}$$

where ΔV_{th_fresh} is the fresh V_{th} offset (e.g., the offset between high V_{th} and nominal V_{th} at time 0), A is $3.9 \times 10^{-3} V \cdot s^{-1/5}$ and S_v is the sensitivity factor extracted by fitting HSPICE simulation results in 45nm, Predictive Technology Model (PTM). 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:

$$\Delta\tau = a \cdot (\Delta V_{th\_bti} + \Delta V_{th\_fresh}) \tag{11}$$

where $\Delta\tau$ is the shift of propagation delay.

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 PTM. The simplified long-term model successfully predicts the MOSFET degradation, with less than 5% loss of accuracy against cycle-by-cycle simulations [2, 7, 18, 19].

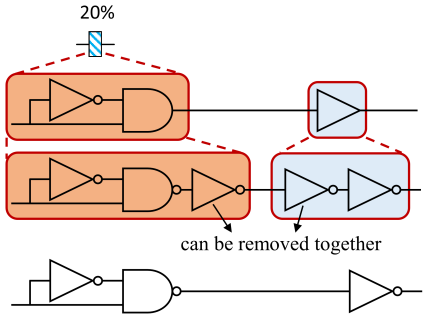


Fig. 4. Practical considerations for DCC insertion

4.2 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 [19]. 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 rates of clock buffers. Figure 2 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% $\longrightarrow$ 20% or 50% $\longrightarrow$ 80%), as shown in Figure 3. Consider the duty-cycle transformation of a clock waveform from 50% to 80% 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 somewhere 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%, explaining the acceleration/deceleration of aging rates of downstream buffers.

4.3 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.

5 PROPOSED FRAMEWORK

In the proposed framework, aging tolerance is achieved by recycling useful/beneficial skews into the designs, based on the concept of timing borrowing. The beneficial skews are progressively created due to different aging behaviors of clock paths caused by DCCs. Moreover, high-V_{th} assignment for clock buffers is also incorporated in the proposed framework to explore/recycle more beneficial skews, further improving aging tolerance of the designs. The overall flow of our framework is depicted in Figure 5, where we focus on the following three tasks:

(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.

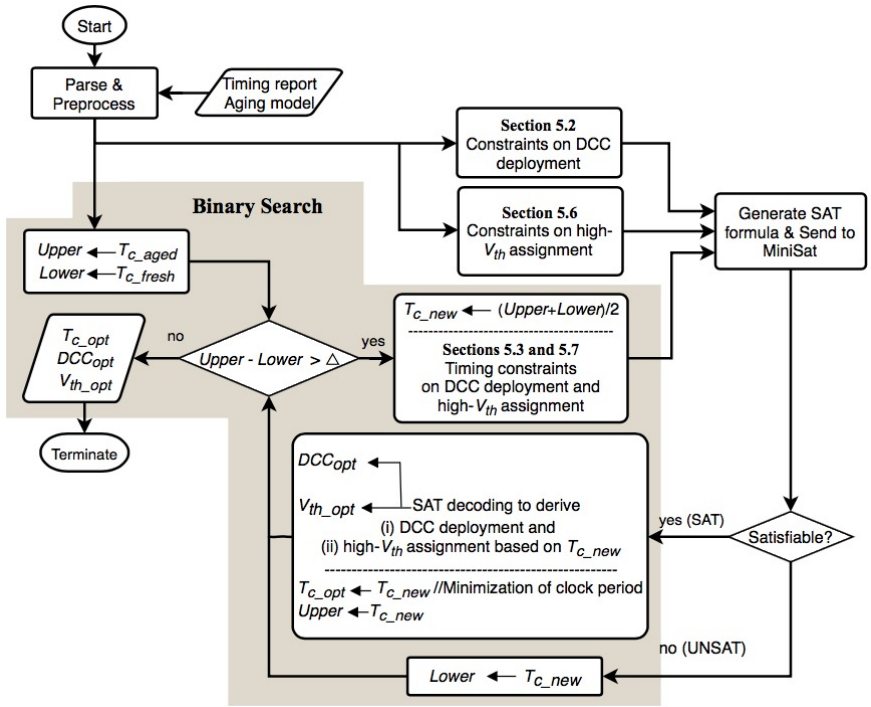


Fig. 5. The overall flow of MAUI

(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 solvers, 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) **High-V_{th} assignment for clock buffers:** The problem of high-V_{th} assignment for clock buffers is also formulated as a SAT-based problem, represented in CNF and solved by existing SAT solvers.

This section is organized as follows: Section 5.1 explains how the proposed problem of DCC deployment/insertion are encoded into Boolean variables. Section 5.2 and Section 5.3 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, Sections 5.4-5.7 introduce the additive improvements by employing high-V_{th} assignment on buffers.

5.1 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

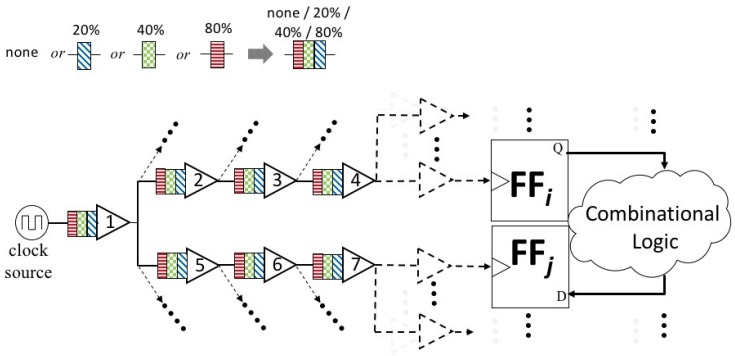


Fig. 6. Generalized DCC insertion for a target pair of flip-flops

variables, $B_{p,q}$ $(1 \le q \le Q, Q = \lceil \lg(N+1) \rceil)$, 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.

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.

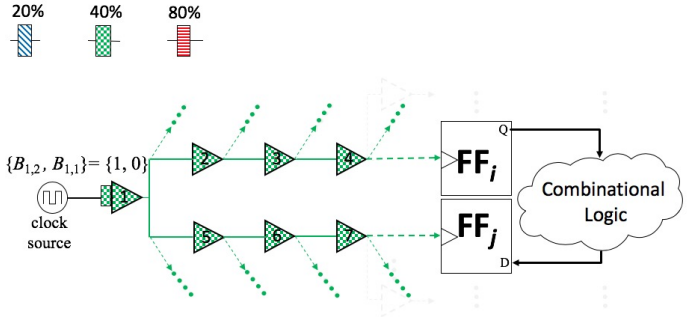
The 20% DCC will mitigate the aging of buffer 3 and its downstream buffers, while the 80% DCC will aggravate the aging of buffer 5 and its downstream buffers. It can be found that, if a DCC is deployed deep in the clock tree (i.e., close to the flip-flops), the count of affected buffers is limited and 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.

5.2 Constraints on DCC Deployment

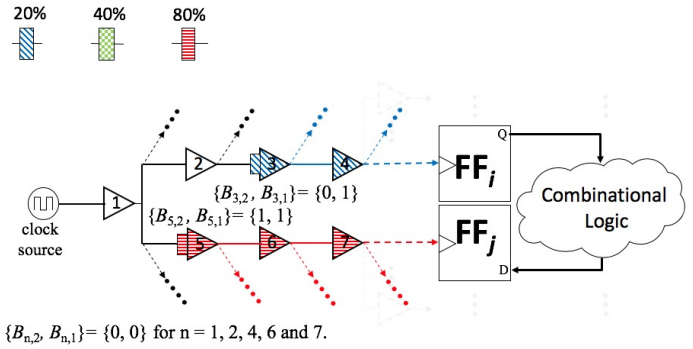
Figure 6 shows a generalized example of DCC insertion for a pair of flip-flops (FF_i and FF_j) where there exist *aging-critical paths* from FF_i to FF_j, defined as follow:

Definition 1 (***Aging-critical path***)**:** 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 insertion and according to the encoding scheme



(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

Fig. 7. Examples of DCC insertion

explained in Section 5.1, two Boolean variables are introduced for each of the seven buffers to encode four possibilities of DCC insertion. Considering all seven buffers, there are a total of 16,384 ($= 4^7$) possibilities for DCC insertion, 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:

Constraint 1 (*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:

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

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 insertion to be formulated. In the above example where the 48 clauses associated with DCC constraints are generated, the possibility count of DCC insertion 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.

5.3 Timing Constraints and Corresponding Clauses

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.

Constraint 2 (*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 deployment 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,

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

equivalent to the following CNF clause:

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

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,

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

equivalent to the following clause:

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

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 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,

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

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,

$$\begin{aligned}&\left(\{B_{1,2}, B_{1,1}\} \equiv \{0,0\}\right) \wedge \left(\{B_{2,2}, B_{2,1}\} \equiv \{0,0\}\right) \wedge \left(\{B_{3,2}, B_{3,1}\} \equiv \{0,1\}\right)\\ \wedge &\left(\{B_{4,2}, B_{4,1}\} \equiv \{0,0\}\right) \wedge \left(\{B_{5,2}, B_{5,1}\} \equiv \{1,1\}\right) \wedge \left(\{B_{6,2}, B_{6,1}\} \equiv \{0,0\}\right)\\ &\wedge \left(\{B_{7,2}, B_{7,1}\} \equiv \{0,0\}\right),\end{aligned}$$

equivalent to the following CNF clause,

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

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

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 5.2.

5.4 Additive Improvement in Aging Tolerance: High-V_{th} Assignment for Clock Buffers

Aging tolerance is achieved by exploring/recycling useful aging-induced clock skews, based on the concept of timing borrowing. So far, the useful skews can be progressively created due to different aging behaviors of clock paths caused by inserting 20%, 40% and 80% DCCs in the existing synthesized clock network. In order to enhance the benefit of useful aging-induced skews, the technique of high-V_{th} assignment for clock buffers, an additive methodology for aging tolerance, is incorporated in our framework. In this way, the exploration space of our optimization framework based on useful skews is enlarged, such that more useful skews can be recycled, enhancing our idea of making aging useful. In the following subsections, the techniques of DCC deployment and high-V_{th} assignment for clock buffers are jointly integrated.

Such a technique of high-V_{th} assignment can cause an issue of fragmentary high-V_{th} assignment for clock buffers. That is, the V_{th} values of continuous buffers along a clock path interlace between high and nominal value. It may decrease the feasibility/manufacturability of the proposed framework. In order to avoid such a phenomenon, we set a rule:

Along a clock path, if the V_{th} value of a clock buffer is assigned high value, then the V_{th} values of its downstream buffers must be also assigned high value.

Such a rule can suppress the occurrence of fragmentary high-V_{th} assignment for clock buffers. Moreover, it also transforms the problem of high-V_{th} assignment to the problem of selecting *high-V_{th} buffer leader*, which is defined as follow:

Definition 2 (*High-V_{th} buffer leader*): A clock buffer is defined as a *high-V_{th} buffer leader*, if the V_{th} values of its downstream buffers (include the leader itself) are assigned high value and the V_{th} values of its upstream buffers (exclude the leader itself) are nominal value.

Conceptually, there exists one difference between DCCs and high-V_{th} buffer leaders. DCCs are physical gates inserted at the inputs of clock buffers to manipulate the aging rates of downstream buffers. However, high-V_{th} buffer leaders, selected from the existing clock buffers, indicate where we begin assigning high V_{th} to the buffers toward flip-flops (exclusive).

5.5 Boolean Encoding for high-V_{th} Buffer Leaders and DCCs

The problem of selecting high-V_{th} 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 (as leaders) and locations of DCCs. Assume that there are three types of DCCs (20%, 40%, and 80%) and one type of high-V_{th} buffer leader. Therefore, three Boolean variables are used for encoding 8 possibilities of DCC insertion and leader selection at any buffer. Consider the clock buffer p, the 8 possibilities can be encoded as follows:

	Leader or not	DCC type	$\{B_{p,3}, B_{p,2}, B_{p,1}\}$
(1)	No	None	{0, 0, 0}
(2)	No	20%	{0, 0, 1}
(3)	No	40%	{0, 1, 0}
(4)	No	80%	{0, 1, 1}
(5)	Yes	None	{1, 0, 0}
(6)	Yes	20%	{1, 0, 1}
(7)	Yes	40%	{1, 1, 0}
(8)	Yes	80%	{1, 1, 1}

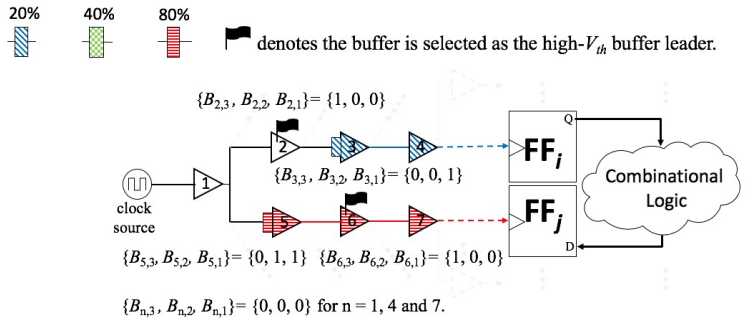


Fig. 8. Example of DCC insertion and high-V_{th} buffer leader selection

where $\{B_{p,2}, B_{p,1}\}$ are used to encode 4 possibilities of DCC insertion at the input of buffer p, and $\{B_{p,3}\}$ is used to encode 2 possibilities of leader selection for buffer p. 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 high-V_{th} 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 specified boundary.

5.6 Constraints on High-V_{th} Assignment for Clock Buffers

When two clock buffers along a single clock path are selected as leaders, the leader selection is meaningless, so that the following constraints are considered. In this way, the SAT solver will not output the corresponding leader selection in the result if the CNF is satisfiable.

Constraint 3 (*Leader constraint*): At most one buffer along a single clock path is selected as high-V_{th} buffer leader.

The constraint is similar to the aforementioned DCC constraint (see Section 5.2, Constraint 1). In order to ensure no more than one leader along any clock path, we generate some clauses to suppress the occurrence of having two leaders along a clock path. Consider buffer 2 (encoded by $B_{2,3}$) and buffer 3 (encoded by $B_{3,3}$) in Figure 6. If buffer 2 is selected as a leader (i.e., $B_{2,3} \equiv 1$), then buffer 3 must not be selected as a leader (i.e., $B_{3,3} \not\equiv 1$), and vice versa. The constraint can be formally written as:

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

Next, it can be translated into 1 CNF clause:

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

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 1 clause and a total of 12 clauses will be generated accordingly.

With leader constraint and corresponding clauses, we can drastically reduce the possibilities of leader selection to be formulated. In the above example where the 12 clauses associated with leader constraint are generated, the possibility count of leader selection drops from 128 (2^7) to 17. Subsequently, the remaining 17 possibilities are considered in the timing constraints, introduced in the next subsection.

5.7 Timing Constraints Considering DCC Deployment and High-V_{th} Assignment

The timing constraints are similar to the former timing constraints (see Section 5.3), while we here consider the influence of high-V_{th} buffer leader on clock latency. It is worth reminding 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 DCCs and 2 high-V_{th} buffer leaders in the clock network associated with aging-critical paths in Figure 8, where 20% and 80% DCC are inserted at the inputs of buffers 3 and 5, respectively, and buffers 2 and 6 are selected as high-V_{th} buffer leaders, implying that the V_{th} of buffer set {2, 3, 4} and buffer set {6, 7} are assigned high V_{th} 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,

$$\begin{aligned}&\left(\{B_{1,3}, B_{1,2}, B_{1,1}\} \equiv \{0,0,0\}\right) \wedge \left(\{B_{2,3}, B_{2,2}, B_{2,1}\} \equiv \{1,0,0\}\right) \wedge \left(\{B_{3,3}, B_{3,2}, B_{3,1}\} \equiv \{0,0,1\}\right)\\ &\wedge \left(\{B_{4,3}, B_{4,2}, B_{4,1}\} \equiv \{0,0,0\}\right) \wedge \left(\{B_{5,3}, B_{5,2}, B_{5,1}\} \equiv \{0,1,1\}\right) \wedge \left(\{B_{6,3}, B_{6,2}, B_{6,1}\} \equiv \{1,0,0\}\right)\\ &\wedge \left(\{B_{7,3}, B_{7,2}, B_{7,1}\} \equiv \{0,0,0\}\right),\end{aligned}$$

equivalent to the following CNF clause:

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

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 selection. Hence, there totally exist 17 ($9 + 7 + 1$) possibilities of leader selection, for each DCC deployment.

6 EXPERIMENTAL SETTING, RESULTS AND DISCUSSION

6.1 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 [2, 7, 18, 19] (detailed in Section 4.1).

6.2 Experimental Results

Table 1 and Table 2 reports the information of each benchmark and the experimental results. In Table 1, Columns 2 to 5 show the total count of gates, total count of flip-flops, total count of clock buffers, and maximum level of the clock tree in each benchmark respectively. In Table 2, Column 4

Table 1. Benchmark information

Benchmark	# Gates	# FFs	# Clock Buffers	Max Tree Level
des_perf	74101	8802	3416	11
leo3mp	526297	108839	26863	10
netcard	561091	97831	33651	12
vga_lcd	101496	17079	5030	10
s13207	2627	625	186	8
s15850	2967	513	100	5
s35932	6466	1728	411	6
s38417	8422	1564	376	6
s38584	6861	1275	627	10

Table 2. Results of aging tolerance

Bexchmark	Fresh	10-year Agixg									
			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	773.9	897.8	849.9	17	16.4	38.66%	835.5	25	924/3416 (27.05%)	685.3	50.28%
leo3mp	3016.7	3530.6	3458.5	11	24.9	14.03%	3406.1	14	3401/26863 (12.66%)	923.2	24.23%
xetcard	3367.0	3940.2	3897.9	1	30.7	7.38%	3860.1	1	1256/33651 (3.73%)	137x.x	13.97%
vga_lcd	864.2	1013.7	994.1	12	32.8	13.11%	981.4	19	601/5030 (11.95%)	1285.4	21.61%
s13207	776.3	891.7	873.3	1	0.7	15.94%	860.6	1	60/186 (32.26%)	11.7	26.95%
s15850	1011.1	1165.6	1089.9	3	0.2	49.00%	1071.7	3	9/100 (9.00%)	0.6	60.78%
s35932	822.1	950.8	886.6	5	1.3	49.88%	865.8	5	45/411 (10.95%)	8.5	66.05%
s38417	798.1	933.2	909.8	13	1.5	17.32%	901.8	13	112/376 (29.79%)	18.5	23.24%
s38584	731.7	842.5	820.3	2	1.4	20.04%	802	4	71/627 (11.32%)	35.9	36.55%
AVG.						25.04%			16.52%		35.96%

T_{c_fresh}: Clock period without considering aging.
T_{c_aged}: Clock period under 10-year aging
$T_{c\_aged\_opt}$: Optimum clock period under 10-year aging after applying the proposed framework

Compare Column 9 with Column 5, DCCs are redeployed (see **Section 6.3**)

Compared with Column 6, runtime in Column 11 increases (see **Section 6.4**)

to 7 show the results when DCC deployment is considered in the framework. Column 8 to 12 show the results when high-V_{th} assignment for clock buffers is applied on top of DCC deployment.

In Table 2, Column 2 demonstrates the fresh clock period that is the circuit delay without aging, denoted by T_{c_fresh}. Column 3 demonstrates the clock period of the circuit under 10-year aging, denoted by T_{c_aged}. Column 4 and 8 demonstrate 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 5 and 9 demonstrate the used DCC count. Column 6 and 11 demonstrate the runtime and the Column 7 and 12 demonstrate 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). With high-V_{th} assignment applied on top of DCC insertion, the clock period drops to 816.8ps, a smaller increment of 42.9ps against T_{c_fresh} (65.38% improvement), implying better aging tolerance. As shown in Table 2, with DCC deployment, the improvement ranges from 7.38% to 49.77% and is 24.95% on average; with high-V_{th} assignment applied on top of DCC deployment, we can achieve better improvement, ranging from 13.97% to 65.97% and being 37.61% on average. The count of inserted DCCs is between 1 (for benchmark *s13207* and *netcard*) to 35 (for benchmark *des_perf*). The inserted DCC count only has the small proportion of the total buffer count (e.g., at most 3.4%(= 13/376) for *s38417*), implying very limited degree of circuit modification and insignificant design overhead.

6.3 Discussion: DCC Redeployment due to High-V_{th} Assignment for Clock Buffers

As we can see, the DCC counts in Column 9 are different from those in Column 5. It implies that DCCs are redeployed in the clock tree while high-V_{th} assignment is incorporated in the framework. To be specific, certain buffers are not allowed to be inserted DCCs at their inputs, due to the violation of timing constraints util high-V_{th} assignment for clock buffers is employed.

For instance, given a clock path P_{clk}, a clock buffer B existing along P_{clk}, and an associated critical path $P_{critical}$ of P_{clk}. Buffer B is not allowed to be inserted any DCC at its input, because the resulting DCC deployment will violate the timing constraints of $P_{critical}$ (i.e., Equation (1) and (2) are not met). However, when one buffer along P_{clk} is selected as a high-V_{th} buffer leader, buffer B becomes allowed to be inserted certain DCCs at its input, because timing constraints of $P_{critical}$ become met. In short, some clock buffers will become allowed to be inserted at their inputs if high-V_{th} assignment is employed, accounting for the phenomenon of DCC redeployment, as observed in Column 5 and Column 9 in Table 2.

6.4 Discussion: Increase in Runtime

When high-V_{th} assignment for clock buffers is applied on top of DCC deployment, the runtime increases because the exploration space of the framework based on useful skews is enlarged. To be specific, given a pair of flip-flops and associated clock paths, we need to consider the various possibilities of leader selection, 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 in runtime.

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

6.5 Discussion: Aging Impact on DCCs

Figure 9 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.

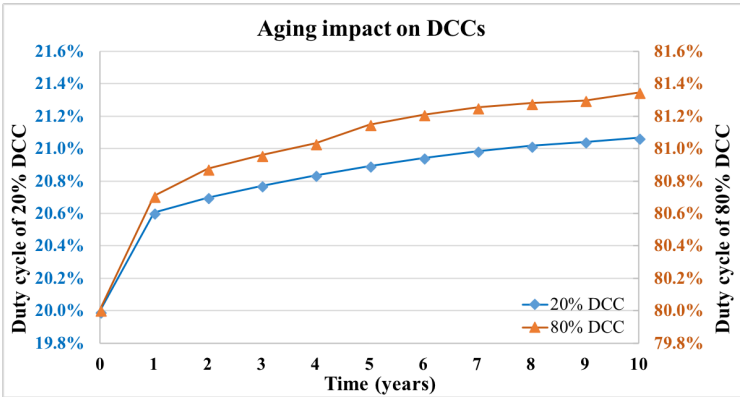


Fig. 9. Aging impact on 20%/80% DCC under BTI

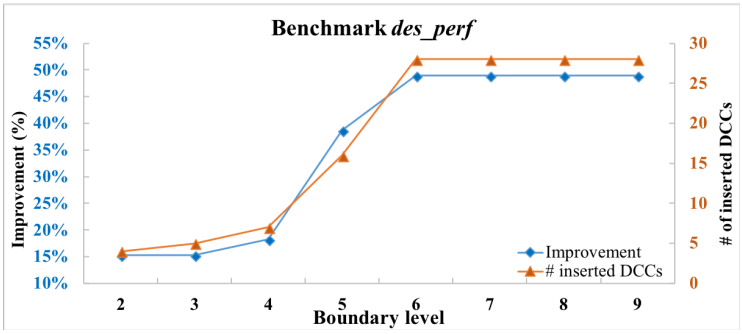


Fig. 10. Improvement/Cost versus clock tree level considered

6.6 Discussion: Depth Boundary for DCC Deployment

As mentioned in Section 5, 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 10, 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.

6.7 Discussion: Fluctuation of Upper and Lower Bounds of Clock Period during Binary Search

It is worth reminding that, the proposed framework is based on a binary search for optimized clock period T_c, with initialized upper and lower bounds of T_c (see Section 5 or Figure 5). During iterations of binary search, the two bounds of T_c are calibrated toward a convergent value, based on the satisfiability of the SAT problem, which the problem of DCC deployment and leader selection is formulated as.

Figure 11 shows the calibrations of the two bounds of T_c for benchmark *leo3mp*, during iterations of binary search. The X-axis denotes the iteration count of binary search and Y-axis denotes the upper (red line) and lower bounds (blue line) of T_c. At the first iteration of binary search, the

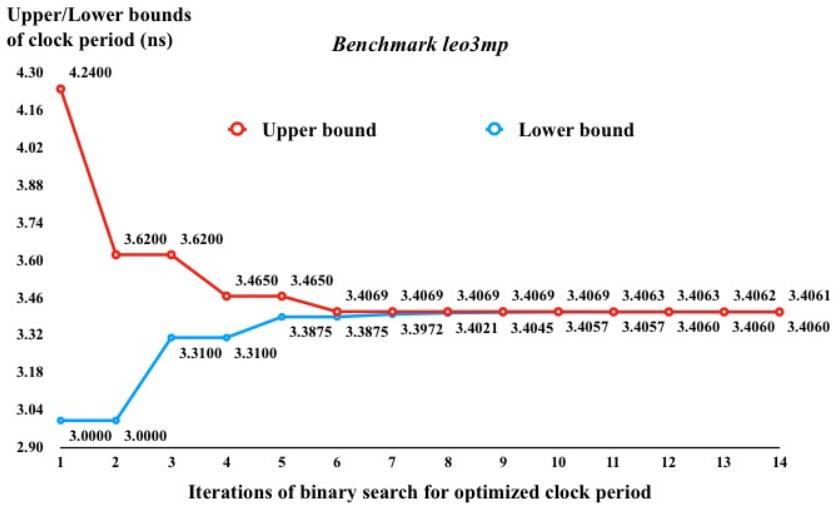


Fig. 11. Fluctuation of upper and lower bounds of T_c during iterations of binary search

upper bound and lower bound are initialized to 4.24ns and 3ns, respectively. Then, we use the mean value of two bounds (i.e., 3.62ns) to generate the associated CNF clauses based on timing constraints (i.e., inequality Equation (1) and (2)). The CNF clauses are satisfiable from the output of SAT solver, implying that we have the opportunity to explore more lower clock period (i.e., better aging tolerance) by updating the upper bound of T_c from 4.24ns to 3.62ns. Then, at the second iteration, we use the mean value of two bounds (i.e., 3.31ns) to generate associated CNF clauses, which are unsatisfiable from the output of SAT solver. It implies that the clock period 3.31ns is too low to find a SAT solution (i.e., there exists no DCC deployment and leader selection to fit the clock period), so that we update the lower bound of T_c to 3.31ns. In this manner, the upper and lower bounds of T_c will converge toward 3.4061ns. The convergent value of 3.4061ns is the optimized clock period for aging tolerance.

As observed in Figure 11, we totally use 14 iterations of binary search to reach the convergent value of T_c, with the accuracy of third digit below the decimal point. If we degrade the accuracy from the third digit to the second digit, we only need 8 iterations. In this manner, we can nearly save half of the runtime.

6.8 Discussion: Clause Count Variations during Binary Search

During iterations of binary search, we generate associated CNF clauses based on timing constraints, i.e., inequality Equation (1) and (2). As the two bounds of T_c are calibrated during binary search, the clause count changes because T_c is a variable in Equation (1) and (2).

Figure 12 plots the clause count and the runtime of SAT solver in each iteration of binary search, for benchmark *leo3mp*. The X-axis, left Y-axis and right Y-axis denote the iteration count of binary search, CNF clause count and runtime of SAT solver, respectively. The clause count depends on the mean value of two bounds of T_c. If the mean value is too low/high, more/less possibilities of DCC insertion and leader selection will violate the timing constraints, resulting in more/less associated CNF clauses. For instance, at the second iteration of binary search, the mean value of two bounds is low (3.31ns, see Figure 11), resulting in large clause count (312798 clauses, see Figure 12). Thanks

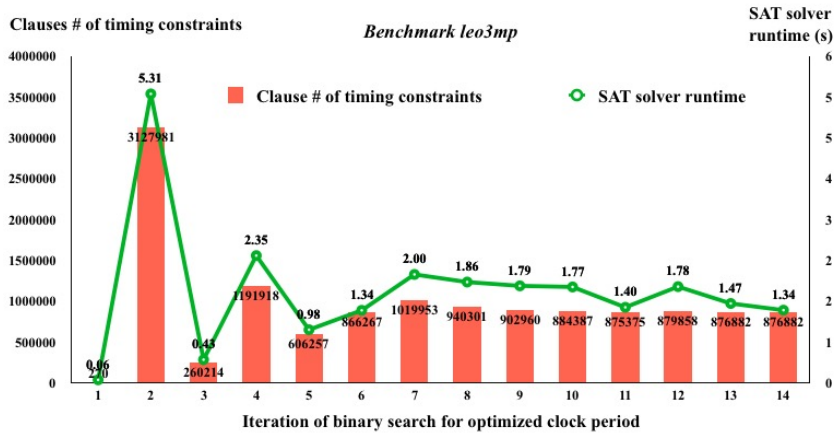


Fig. 12. Variations of CNF clause count during iterations of binary search

to the SAT solver such as MiniSat, we can efficiently solve the SAT problem, which is formulated as CNF clauses, at the cost of less than 10s in each iteration (at most 5.31s for benchmark *leo3mp*).

7 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 25.04% aging tolerance via inserting at most 16 DCCs. Furthermore, we incorporate the technique of V_{th} assignment in the framework, to further improve circuit aging tolerance, by assigning high V_{th} to a fraction of clock buffers. The problem of high-V_{th} assignment for clock buffers is also formulated as a Boolean satisfiability problem, solved by MiniSat. Experiments shows that the framework, which both considers DCC deployment and high-V_{th} assignment for clock buffers, can result in 35.96% aging tolerance on average, via inserting at most 25 DCCs and assigning high V_{th} to an average of 16.52% of buffers.

REFERENCES

[1] 2015. International Technology Roadmap for Semiconductor. (2015).
[2] Hussam Amrouch, Behnam Khaleghi, Andreas Gerstlauer, and Jörg Henkel. 2016. Reliability-aware design to suppress aging. In *Proceedings of the Design Automation Conference*. 1–6.
[3] Ashutosh Chakraborty and David Z Pan. 2013. Skew management of NBTI impacted gated clock trees. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 6 (2013), 918–927.
[4] Srini Chakravarthi, Anand Krishnan, Vijay Reddy, CF Machala, and Srikanth Krishnan. 2004. A comprehensive framework for predictive modeling of negative bias temperature instability. In *Proceedings of the International Reliability Physics Symposium*. 273–282.
[5] Jifeng Chen and Mohammad Tehranipoor. 2013. A novel flow for reducing clock skew considering nbti effect and process variations. In *Proceedings of the International Symposium on Quality Electronic Design*. 327–334.
[6] John P. Fishburn. 1990. Clock skew optimization. *IEEE Trans. Comput.* 39, 7 (1990), 945–951.
[7] Andres F Gomez and Victor Champac. 2016. Early selection of critical paths for reliable nbti aging-delay monitoring. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 7 (2016), 2438–2448.
[8] Shih-Hsu Huang, Wen-Pin Tu, Chia-Ming Chang, and Song-Bin Pan. 2013. Low-power anti-aging zero skew clock gating. *ACM Transactions on Design Automation of Electronic Systems* 18, 2 (2013), 27.

[9] Naohiko Kimizuka, Toyoji Yamamoto, Tohru Mogami, Ken Yamaguchi, Kazuhiro Imai, and T Horiuchi. 1999. The impact of bias temperature instability for direct-tunneling ultra-thin gate oxide on MOSFET scaling. In *Proceedings of the Symposium on VLSI Technology. Digest of Technical Papers*. 73–74.
[10] Sanjay V Kumar, Chris H Kim, and Sachin S Sapatnekar. 2006. An analytical model for negative bias temperature instability. In *Proceedings of the International Conference on Computer-Aided Design*. 493–496.
[11] Sanjay V Kumar, Chris H Kim, and Sachin S Sapatnekar. 2007. NBTI-aware synthesis of digital circuits. In *Proceedings of the Design Automation Conference*. 370–375.
[12] Liangzhen Lai, Vikas Chandra, Robert Aitken, and Puneet Gupta. 2014. BTI-Gater: An aging-resilient clock gating methodology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 4, 2 (2014), 180–189.
[13] Li Li, Yinghai Lu, and Hai Zhou. 2011. Optimal multi-domain clock skew scheduling. In *Proceedings of the Design Automation Conference*. 152–157.
[14] Joseph W McPherson. 2006. Reliability challenges for 45nm and beyond. In *Proceedings of the Design Automation Conference*. 176–181.
[15] Bipul C Paul, Kunhyuk Kang, Haldun Kufluoglu, Muhammad Ashraful Alam, and Kaushik Roy. 2006. Temporal performance degradation under NBTI: Estimation and design for improved reliability of nanoscale circuits. In *Proceedings of the Design, Automation and Test in Europe Conference*. 780–785.
[16] Dieter K Schroder and Jeff A Babcock. 2003. Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of Applied Physics* 94, 1 (2003), 1–18.
[17] James H Stathis and Sufi Zafar. 2006. The negative bias temperature instability in MOS devices: A review. *Microelectronics Reliability* 46, 2-4 (2006), 270–286.
[18] Wenping Wang, Zile Wei, Shengqi Yang, and Yu Cao. 2007. An efficient method to identify critical gates under circuit aging. In *Proceedings of the International Conference on Computer-Aided Design*. 735–740.
[19] Wenping Wang, Shengqi Yang, Sarvesh Bhardwaj, Sarma Vrudhula, Frank Liu, and Yu Cao. 2010. The impact of NBTI effect on combinational circuit: modeling, simulation, and analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 2 (2010), 173–183.
[20] Xiangning Yang and Kewal Saluja. 2007. Combating NBTI degradation via gate sizing. In *Proceedings of the International Symposium on Quality Electronic Design*. 47–52.

Received 0 2018