Lifetime Reliability Trojan based on Exploring Malicious Aging

Tien-Hung Tseng and Kai-Chiang Wu
Department of Computer Science
National Chiao Tung University, Hsinchu, Taiwan
E-mail: {eric830303.cs05g@g2.nctu.edu.tw and kcw@cs.nctu.edu.tw}

***Abstract*—Due to escalating complexity of hardware design and manufacturing, not only are integrated circuits (ICs) designed and fabricated in multiple nations, but also software tools may be supplied worldwide. It makes hardware security become more subject to various kinds of tampering in the supply chain. Hardware Trojan horses (HTHs) can be implanted to facilitate the leakage of confidential information or cause the failure of a system. Reliability Trojan is one of the main categories of HTH attacks because its behavior is progressive and is thus not trivial to be detected, or not considered malicious. In this work, we propose to insert reliability Trojan into a circuit which can finely control the circuit lifetime as specified by attackers (or even designers), based on manipulating BTI-induced aging behavior in a statistical manner. Experimental results show that, given a specified lifetime target and under the influence of process variations, the circuit is highly likely to fail within a desired lifetime interval. Instead, in a typical design considering pessimistic corners, the resulting lifetime is usually far longer than the target.**

I. INTRODUCTION

Due to escalating complexity of hardware design and manufacturing [1], integrated circuits (ICs) are usually designed and fabricated in multiple locations worldwide. Moreover, some design tools are also supplied from different units. With the utilization of third-party IP components and off-shore IC manufacturing, the overall cost and time-to-market are significantly reduced. However, hardware security becomes more subject to various kinds of tampering in the supply chain [2][3]. Typically, a hardware system does no more than its requirements. Doing more than required, hardware Trojan horses (HTHs) can be implanted to facilitate the leakage of confidential information or cause the failure of a system [4]-[5]. Outsourcing (e.g., third-party IP components, design tools and off-shore IC manufacturing) makes malicious HTH attacks possible. In order to introduce the motivation of deploying HTH attacks, here we make some scenario from various aspects, i.e., attackers. Manufacturers: Given a design house A with its competitor B, in order to interfere B's commercial development, A paid B's ICs manufacturer M such that M tampered B's layout, making B's product(s) malfunction earlier than expected. Design-tool suppliers: Given a country C with its imaginary enemy D, and we assume that D's design houses utilize at least one design tool supplied from C's software corporation S. For the purpose of obstructing D's high-tech military weapon development, C forced S to embed malicious mechanisms in its software merchandise. Therefore, no matter what kinds of military equipment D produces, they are stealthily put HTH in the inner ICs. After the insertion of HTHs, it is difficult to prove their existence since they are pervasive and inappreciable. In this sense, the proposed research provides new primitives for aforementioned hardware security threats, by exploring the feasibility of different HTH attacks and associated detection/prevention countermeasures. Reliability Trojan is one of the main categories of HTH attacks because its behavior is progressive and is thus not trivial to be detected, or not considered malicious. Time-de- pendent dielectric breakdown (TDDB), bias temperature in- stability (BTI), and electromigration (EM) are some of the critical failure mechanisms affecting lifetime reliability. With the continuous shrinking of transistor and interconnect dimensions, the rate of such progressive wear-out failures is getting higher. In addition, due to the increasing transistor density without proportional downscaling of supply voltage, the power density and thus the operating temperature will rise significantly, which further accelerates the failure mechanisms because they are all exponentially dependent on temperature. In this work, we propose to insert reliability Trojan into a circuit which can finely controls the circuit lifetime as specified by attackers (or even designers), based on manipulating BTI-induced aging behavior in a statistical manner.

II. RELATED WORK

Studies about reliability Trojan have been proposed since last few years. [6] details BTI and HCI effects which induce aging failures, and accelerates the effects by aggravating the most influential parameters of BTI and HCI. [7] proposes a few Trojan designs which accelerate EM, BTI and TDDB effects by stressing/modifying specific interconnects and gates. Some studies also try to control the lifetime of a circuit by counters or timers. In [8], authors present a Trojan which controls lifetime by analog mechanism. It siphons charge from target wire and stores to a capacitor until voltage on the capacitor rises above the threshold and sets its output flip-flop to a desired value. [9] presents an unmodified Trojan by analyzing the netlist of a circuit to identify its critical paths; then they generate patterns/instructions for stressing those paths. These patterns can be fed by external programs or embedded devices to accelerate the aging and decrease the circuit performance and lifetime. [8] [9] focus on the logic blocks which highly depend on users' operational modes.

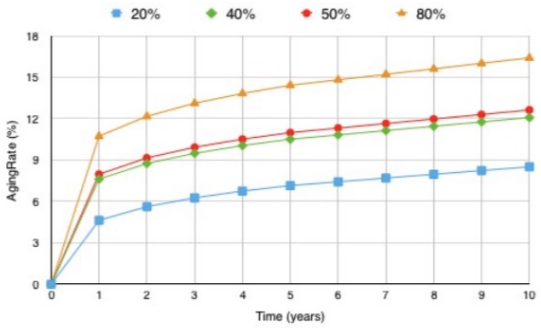


Figure 1. aging rates of buffers with different clock duty cycles

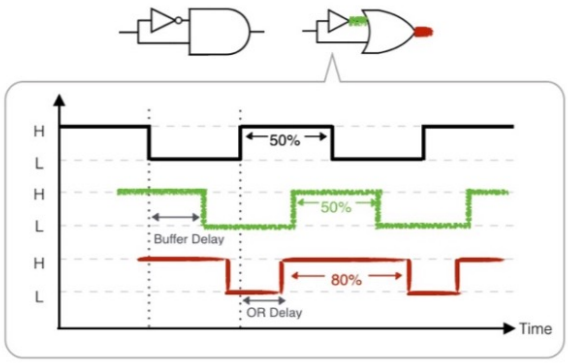


Figure 2. Construction of DCC and duty-cycle transformation

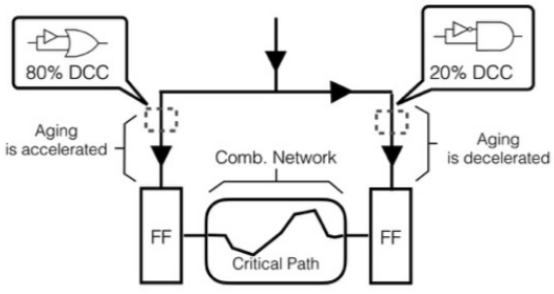


Figure 3. Example of DCC insertion

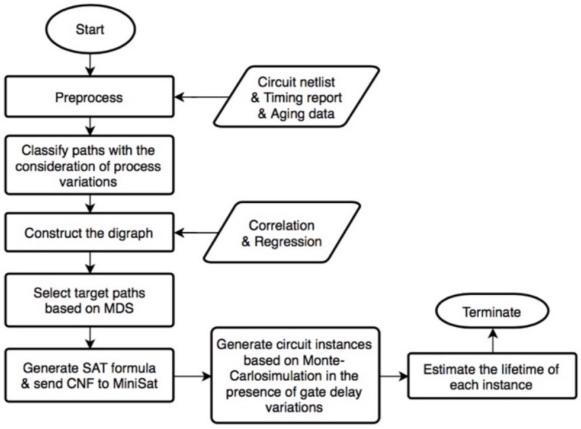


Figure 4. DCC insertion flow

However, [6] does not estimate circuit lifetime in detail and [7] has relatively high cost based on using counters to control lifetime. To predict circuit lifetime with Trojans, [10] uses mathematical modeling to estimate circuit reliability, but it only tries on tiny circuit C17 and does not consider aging. In addition, [11] proposes an idea using aging effects to induce a circuit into its redundant states (i.e., operational modes) and thereafter execute malicious function. This paper proposes a method of hardware Trojan insertion to control the lifetime of a circuit based on manipulating the rate of circuit aging. We consider (i) the aging of both clock trees and combinational logical paths, and (ii) the correlation of aging rates between critical paths. These considerations ensure the effect of our proposed Trojan to be manifested on time under all possible workloads due to various users' operational conditions. More clearly, we present a methodology that deploys duty-cycle converters (DCCs) into a clock tree to accelerate the aging of predesignated clock buffers/inverters associated with critical paths. Those paths will fail around the time we set regardless of operational conditions.

III. MOTIVATING EXAMPLE

A. Duty-Cycle Converter (DCC)

Duty cycle is the percentage of one period in which a signal is high (i.e., logic 1). We have known that 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. 1 shows the aging rates of clock buffers with different clock duty cycles. The unit we use to change the clock duty cycle is duty-cycle converter (DCC). A DCC includes an inverter/buffer and an AND/OR gate. It can convert the duty cycle of a clock signal to a smaller/larger one. 2 shows a DCC and the conversion of the clock duty cycle from 50% to 80%. It separates the source clock wave (black line) to a delayed wave (green line, may be inverted) and original wave. Then, those two waves are combined with the OR gate to obtain a new clock wave (red line) of 80% duty cycle.

B. DCCs against a Critical Path

Once we insert a DCC 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%. Consider the circuit in 3: we insert an 80% DCC on the left clock path to accelerate its aging and a 20% DCC on the right clock path to decelerate its aging. Over several years, the left clock latency will gain greater than the right one does. Therefore, setup-time violations are likely to occur on this critical path.

Figure 5. Classification of critical paths

IV. DEFINITION AND DCC INSERTION FLOW

Our proposed flow of DCC insertion is shown in Figure 4. First, we read the data (e.g. circuit netlist, timing report, aging prediction parameters) and perform preprocessing. The detail about preprocessing will be described later. Then we construct a graph according to the correlation and regression between critical paths. In the graph, a vertex represents a critical path, and an edge between two vertices (i.e., a pair of critical paths) represents the correlation/regression between them. Afterwards, we choose some paths as our targets and transform our problem of DCC insertion based on SAT (Boolean Satisfiability), which can be solved by existing SAT solvers such as miniSat. Finally, we estimate the circuit lifetime after DCC insertion to evaluate the quality of our HTH attack.

Definition

- n: Expected circuit lifetime under the proposed HTH attack based on DCC insertion. The unit is year.
- ε: Maximum tolerable error. The unit is year.
- *Desired lifetime interval*: A desired interval of path/circuit lifetime under workload variations and statistical considerations of aging behavior among critical paths. Given n and ε, the desired lifetime interval is $[n-\varepsilon, n+\varepsilon]$.

Terminology

- *Premature failure* denotes that a path leads to timing violations earlier than $n-\varepsilon$ years.
- *Postmature failure* denotes that a path leads to timing violations later than $n+\varepsilon$ years.
- *Proper failure* denotes that a path leads to timing violations within desired lifetime interval.

Classification of DCC Placements

- *Prematurely-failing DCC placement*: A kind of DCC placement which leads the critical path to premature failure.
- *Properly-failing DCC placement*: A kind of DCC placement which leads the critical path to proper failure.
- *Posmaturely-failing DCC placement*: A kind of DCC placement which leads the critical path to postmature failure.
- *Illegal DCC placement (insertion)*: At most one DCC is allowed along a clock path from the clock source to a flip-flop. Placing more than one DCC along the same clock path is called illegal DCC placement.

Classification of Critical Paths

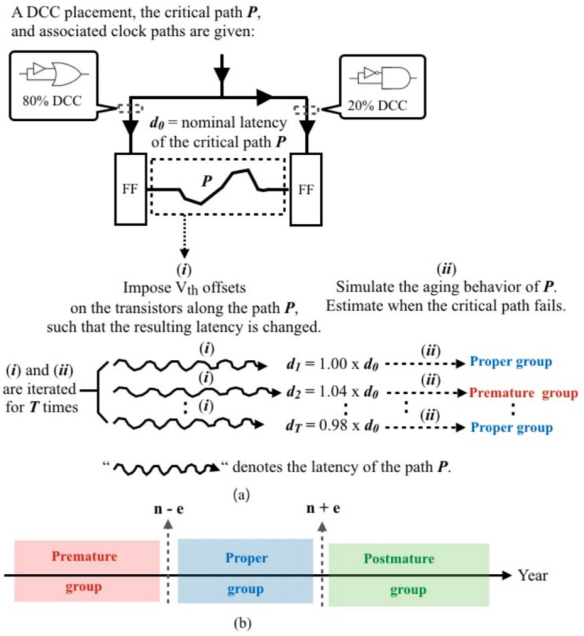


Figure 6. Classification DCC placements considering process variation

- *Candidate*: A critical path is defined as a candidate if there exists one properly-failing DCC placement.
- *Shortlist*: A shortlist is a subset of candidates which are selected as target paths to be attacked. Attacking such paths involves placing DCCs on their associated clock paths.
- *Mine*: A path is called a victim if one or more DCCs exist on its associated clock paths, as a result of attack on *shortlist*.
- *Effective victim*: A path is called an effective victim if it is a victim and the existing DCC(s) will lead the path to *proper failure*.

Figure 5 illustrates the relationship among those classification of critical paths. First, there is no intersection between candidate and mine. Second, the intersection between candidate and victim is effective victim. Third, shortlist is a subset of candidate; effective victim is a superset of shortlist, and so is victim. Moreover, victim and mine are likely to intersect because some paths in victim may belong to mine, and we have to ensure that these paths will not fail prematurely.

A. Classification of DCC Placements Considering Process Variations

Note that the classification of DCC placements introduced earlier is deterministic, i.e., unaware of process-induced timing variability when referring to timing violations. In this subsection, we include the influence of process variation (PV) when classifying DCC placements, such that the resulting attack is PV-aware. The PV-aware method is depicted in Figure 6(a). Given a DCC placement on the clock path associated with one critical path, we classify the DCC placement based on Monte-Carlo simulation with the following two procedures: (*i*)

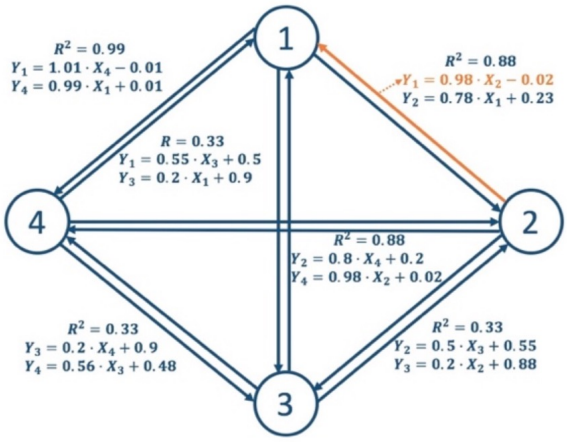


Figure 7. Example of graph used in choosing targets

Impose extra positive/negative Vth offsets on the transistors along the critical path. Note that, the Vth offsets follow a normal distribution, with the standard deviation of a specified value, which is usually set to 30mV. That is, 68% of Vth offsets of transistors reside between positive and negative 30mV. (*ii*) Then, a model, introduced in Section VI, is used to simulate the aging behavior of the critical path, such that we can estimate when the timing violation will occur. Note that, the model not only converts the Vth offset to the delay shift of the transistor, but also considers the correlation of PV and BTI. The model will be explained in Section VI. Then, as illustrated in Fig. 6(a), the steps of (*i*) and (*ii*) are iterated for T times. Each iteration results in a time point, at which the critical path fails (i.e., timing violation occurs). Then, as shown in Figure 6(b), the time points are categorized into 3 groups: premature group, proper group, and postmature group. Finally, we count the size of each group (i.e., the number of time points in the group). If the size of proper group is larger than a threshold, said H, the placement is classified as a properly-failing DCC placement. Otherwise, the sizes of the other two groups are compared. The larger one determines the classification of the DCC placement. For instance, if the size of proper group is smaller than the threshold H, the sizes of premature group and postmature group are compared. Without the loss of generality, assume the size of postmature group is larger; then, the DCC placement is finally classified as postmaturely-failing one.

B. Selection of Target Paths (Shortlist) to be Attacked

If an attacked critical path always ages as estimated (i.e., under worst-case aging), a successful attack can be obtained just by inserting DCCs on its associated clock paths. However, uncertainty of user-dependent operational modes (e.g., watching video, playing games) can influence/vary the aging behavior. Therefore, we must ensure that the attack will succeed under any operational mode. We make the following assumption, which will be used in Section [Undefined]:

"Every operational mode causes at least one candidate path to undergo worst-case aging."

An operational mode, which causes one path to undergo worst-case aging, is called ?critical operational mode? of the path. Further, the union of critical operational modes of all candidate paths is equivalent to the universe of operational modes. Based on the assumption, attacking all candidates is a nave method to guarantee successful attack. Nevertheless, it is very costly and may be impossible.

Therefore, after observing the relationship among aging of paths, we find that the aging behaviors of many paths are highly correlated. If several paths are highly correlated in terms of aging behavior, one operational mode can lead all of them to age to a similar extent. Thus, we can simply attack one out of those highly correlated paths to cover multiple operational modes. For example, given two critical paths A and B. Their critical operational modes are O_A and O_B, respectively. Assume that A and B are highly correlated in terms of aging behavior. Because A and B age similarly/closely, O_A causes A to age in the worst-case and also causes B to age severely. Consequently, even if we simply attack path A, not only O_A but also O_B can make the attack successful (premature failure of path A). This property helps reduce the number of targeted paths to be considered/formulated. To choose the attack targets (shortlist), we transform the relationship of paths to a directed graph (also known as digraph). In Figure 7, vertices represent candidates and arcs (i.e., directed edges) are correlation coefficients (R^2) and linear regression equations between each pair of vertices. Each arc has a regression equation, whose coefficients are obtained by running functional simulation. X_i denotes the worst-case aging rate of path i, whose exact value will be introduced in Section [Undefined]. Y_j denotes the aging rate of path j predicted based on the linear regression equation. Consider the orange equation in Figure 7:

$$Y_1 = 0.98 \cdot X_2 - 0.02$$

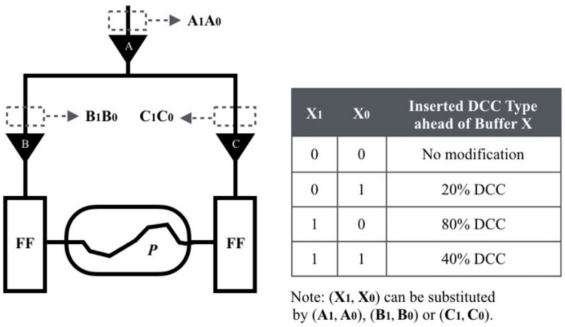
Given the worst-case aging rate of vertex/path 2, X_2, the aging rate of vertex/path 1, Y_1, can be predicted as 0.98 multiplied by X_2 minus 0.02 Before the shortlist is determined by selecting a subset of candidates in the graph, we can simplify the graph by removing some arcs which indicate the relationships of weak aging correlation between pairs of paths.

The cost of our proposed HTHs is the count of inserted DCCs. In order to minimize the cost, we must select minimum-sized targets to cover all candidate paths, that is, to dominate all candidate paths in the digraph. This problem is similar to a classical digraph problem, **Minimum Dominating Set (MDS)**:

On digraph $G = (V, E)$, find a minimum-sized set of vertices $S \subseteq V$ such that $\forall y \notin S$, $\exists x \in S$, there exists an arc from x to y. And we say that y is dominated by x.

C. Generation of SAT Formula

After the shortlist is determined, the problem is transformed into **Conjunctive Normal Form (CNF)**. That is, it is for-



X_1	X_0	Inserted DCC Type ahead of Buffer X
0	0	No modification
0	1	20% DCC
1	0	80% DCC
1	1	40% DCC

Note: (X_1, X_0) can be substituted by (A_1, A_0), (B_1, B_0) or (C_1, C_0).

Figure 8. Introduction of SAT-formulation

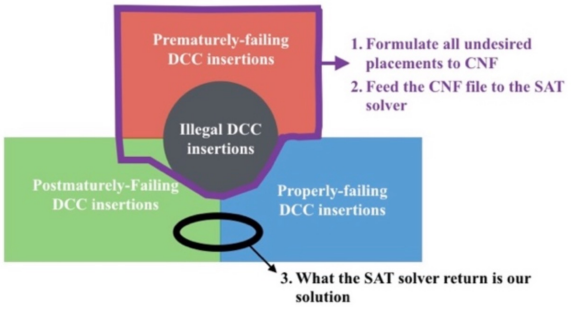


Figure 9. SAT-formulation and relationship among 3 groups of DCC insertions

mulated as a **Boolean satisfiability (SAT)** problem. First, as shown in Figure 8, each clock buffer is labeled by two Boolean variables to represent four types of DCCs (none, 20%, 40%, 80%) inserted ahead of it. Next, the proposed formulation is based on the classification of critical paths:

1) Paths in the shortlist (i.e., targets): On their associated clock paths, formulate all conditions of prematurely-failing DCC placements and postmaturely-failing DCC placements.
2) Other paths (paths not in the shortlist): On their associated clock paths, formulate all conditions of prematurely-failing DCC placements.
3) All paths: Formulate all conditions of illegal DCC placements.

Consider the example in Figure 8: assume that critical path P is in the shortlist. If the insertion of an 80% DCC ahead of buffer B will lead the path to premature failure, then the following clause will be added into CNF:

$$(A_1 \vee A_0 \vee \neg B_1 \vee B_0 \vee C_1 \vee C_0)$$

Then, if the insertion of a 20% DCC ahead of buffer C will lead the path to postmature failure, then the following clause will be added into CNF:

$$(A_1 \vee A_0 \vee B_1 \vee B_0 \vee C_1 \vee \neg C_0)$$

More explicitly, according to the above statements, we must avoid those conditions denoted by the red shape illustrated in Figure 9, and expect to find a feasible solution denoted by the green shape. After the formulation, our proposed problem based on DCC insertion is transformed into a CNF file. The CNF can be fed to a SAT solver and we can find the locations and types of inserted DCCs by decoding the output from the solver.

V. LIFETIME ESTIMATION

After we obtain a solution from foregoing steps, we have to estimate the lifetime of the entire circuit. The detailed procedures are shown in algorithms in Figure 10(a) and Figure 10(b).

Before Algorithm 1, we first construct a digraph, which is similar to the previous one used in choosing targets (based on MDS). Here, the definition of an arc remains the same; however, vertices represent not only candidates but also mines. Then, we use the same measures to prune the digraph; it denotes that, arcs, which indicate the relationships of weak aging correlation between pairs of paths, will be deleted in the resulting digraph.

Furthermore, the previous assumption introduced in Section IV-B says that the union of critical operational modes of all candidate paths is equivalent to the universe of operational modes. That is, after applying worst-case aging on each candidate path (Line 2 in Algorithm 1), all operational modes are considered during lifetime estimation in our methodology. Here, Line 2 in Algorithm 1, says that a candidate path i is assumed to age in the worst case. Then, in the inner for-loop (Lines 4-8), we iteratively estimate the lifetime of other paths based on prediction from the worst-case aging of i. For each path, say p (Line 4), the following steps are applied to estimate its lifetime: (Step 1, Line 6) $p's$ lifetime can be estimated based on a binary search, which will be depicted in Algorithm 2, and (Step 2, Line 7) the estimated lifetime is compared against the smallest one, since the smallest lifetime among all paths determines the circuit lifetime. In Algorithm 2, path $p's$ lifetime is estimated based on a binary search. The upper/lower bounds for the binary search are set in Lines 1-2, respectively. In Line 6, M is set as the median value of two bounds; and in Line 10, $p's$ aging rate R_p is predicted by the regression equation of p on i, where i is assumed to age under M-year worst-case condition and its aging rate value is obtained by the following predictive model, which in presented in [12]:

$$A \cdot \alpha^n \cdot t^n \tag{1}$$

where A and n are fitted constants, α denotes the stress duty cycle, and t denotes time (unit is second). α is usually set to 0.5. A and n are fitted as 0.0039 and 0.2, respectively, after SPICE simulation.

Then, in Line 11, R_p is used to estimate $p's$ slack S_p, which is utilized to check whether the setup-time violations will occur on p under the aging rate R_p (Line 12). If the value of S_p is negative, it denotes that, setup-time violations will occur on p after M years at an aging rate of R_p (Line 14). Afterwards, according to result of the above timing check, the upper/lower bound for next iteration will be set in Line 13 or 15. While repeating the above steps (Lines 6-15), both

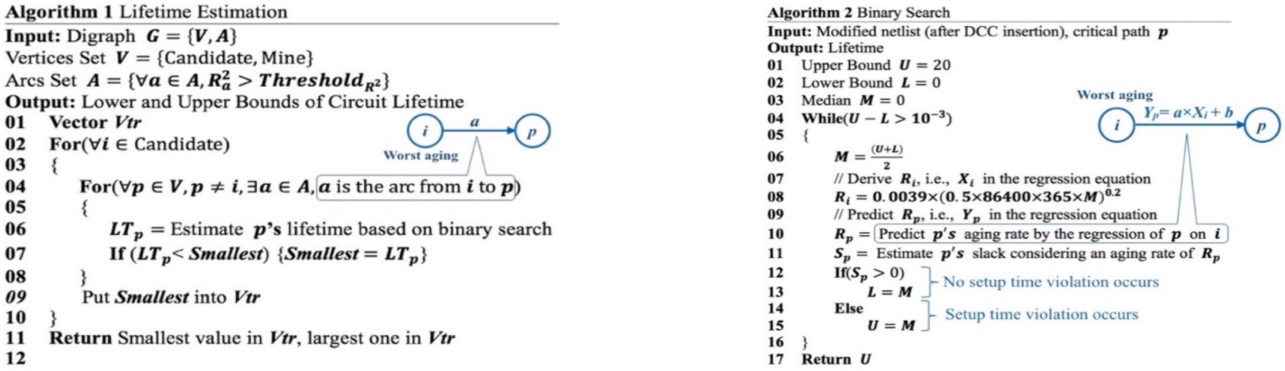


Figure 10. Algorithms used to estimate lifetime of designs

bounds gradually converge. Eventually, the converged value of the upper bound is considered as $p's$ lifetime (Line 17), which will be returned to Algorithm1 as the value of LT_p (Line 6 in Algorithm 1).

The aforementioned procedure is repeated for each candidate path. We can derive a lifetime value by considering a specific candidate path. By considering all of the candidate paths, all operational modes can be considered and we can find a group of lifetime values, i.e., V_{tr} in Algorithm 1. The smallest value and the largest one within V_{tr} are the resulting lifetime interval found based on this given attack.

REFERENCES

[1] L. Wilson, "International technology roadmap for semiconductors (itrs)," *Semiconductor Industry Association*, 2013.

[2] M. Tehranipoor *et al.*, "Trustworthy hardware: Trojan detection and design-for-trust challenges," *Computer*, vol. 44, no. 7, pp. 66–74, 2011.

[3] R. Karri *et al.*, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[4] S. Adee, "The hunt for the kill switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[5] S. Bhunia *et al.*, "Hardware trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[6] Y. Shiyanovskii *et al.*, "Process reliability based trojans through nbti and hci effects," in *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, IEEE, 2010, pp. 215–222.

[7] A. Sreedhar, S. Kundu, and I. Koren, "On reliability trojan injection and detection," *Journal of Low Power Electronics*, vol. 8, no. 5, pp. 674–683, 2012.

[8] K. Yang *et al.*, "A2: Analog malicious hardware," in *Security and Privacy (SP), 2016 IEEE Symposium on*, IEEE, 2016, pp. 18–37.

[9] N. Karimi *et al.*, "Magic: Malicious aging in circuits/cores," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 12, no. 1, p. 5, 2015.

[10] S. Burman *et al.*, "Effect of malicious hardware logic on circuit reliability," in *Progress in VLSI Design and Test*, Springer, 2012, pp. 190–197.

[11] S. Wei and M. Potkonjak, "The undetectable and unprovable hardware trojan horse," in *Proceedings of the 50th Annual Design Automation Conference*, ACM, 2013, p. 144.

[12] W. Wang *et al.*, "An efficient method to identify critical gates under circuit aging," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, IEEE, 2007, pp. 735–740.