Fast Estimation of NBTI-Induced Delay Degradation Based on Signal Probability

Song Bian[1] Michihiro Shintani[1] Masayuki Hiromoto[1] Takashi Sato[1]

Abstract: As technology further scales semiconductor devices, aging-induced device degradation has become one of the major threats to device reliability. Hence, taking aging-induced degradation into account during the design phase can greatly improve the reliability of the manufactured devices. However, accurately estimating the aging effect for extremely large circuits, like processors, is time-consuming. In this research, we focus on the negative bias temperature instability (NBTI) as the aging-induced degradation mechanism, and propose a fast and efficient way of estimating NBTI-induced delay degradation by utilizing static-timing analysis (STA) and simulation-based lookup table (LUT). We modeled each type of gates at different degradation levels, load capacitances and input pins. Using these gate-delay models, path delays of arbitrary circuits can be efficiently estimated. With a typical five-stage pipelined processor as the design target, by comparing the calculated delay from LUT with the reference delay calculated by a commercial simulator, we achieved 5760 times speedup within 13 % error.

1. Introduction

In a time where semiconductor devices are scaling near the limit of physical laws, they become more and more unpredictable and uncontrollable. Nano-scale transistor devices vary their characteristics greatly even within a single chip, and one of the major factors for the variations is aging-induced device degradation. The variation due to aging exerts great impact on design decisions, since that without knowing how the device age, designers could potentially waste time for optimizing paths that become non-critical in less than a year.

Among various aging mechanisms, negative bias temperature instability (NBTI) is considered to be one of the most crucial factors that shorten the lifespan of VLSI circuits. NBTI causes a gradual increase in threshold voltage, V_{th}, while a negative bias is applied to a pMOS transistor, i.e., when a pMOS is ON. This state is defined as the stress phase of NBTI. The increase in the threshold voltage (ΔV_{th}) reduces the drain current and consequently increases path delays. Meanwhile, when the pMOS is OFF, the stress is removed and its V_{th} gradually decreases to its original value. This state is defined as the recovery phase of NBTI. The V_{th} degradation of pMOS due to NBTI gradually progresses through circuit operations, while repeating the stress and recovery phases. As a result of the pMOS degradation, the delay of combinational paths becomes longer, eventually violating the design timing constraint.

The state-of-art NBTI mitigation techniques include internal node control (INC) [1], input vector control (IVC) [2] and simple design margin [3]. Obviously, these methods require path delays in the circuit to be predicted correctly, such that the NBTI-induced stresses along those paths can be reduced. In order to be NBTI-aware in design phase, tools like SPICE are adopted. Nonetheless, for large designs like processors, this method becomes extremely or even prohibitively time consuming. Especially in cases of applying heuristic or stochastic methods for design optimization, such as genetic optimization [4], a large amount of path delays have to be known in the matter of seconds while considering ΔV_{th} of each gate on the paths. Hence, a fast and sufficiently accurate method to estimate the aging-induced characteristic variations is preferred.

In this paper, we propose a lookup-table-based (LUT-based) aging-aware delay estimation technique utilizing a modified static-timing analysis approach. The technique consists of two main basic elements: signal probability propagation (SPP) and LUT interpolation. The delay of a path is a sum of the individual gates along the path, each having their own NBTI stress probability, input slew and capacitive load etc. Note that the technique is STA-like, yet not a complete STA. The details will be explained in what follows; however, for the sake of simplicity, this modified version is called STA throughout the paper. The accuracy of the result is compared to the typical SPICE simulation, and a detailed error analysis is conducted with potential space for improvements of our approach discussed.

2. Related Works

2.1 NBTI Model

To predict NBTI-induced ΔV_{th} degradation, NBTI measurements and mathematical models are studied in [5], [6]. Our degradation calculation is based on an analytical model

[1] Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo, Kyoto, 606–8501, Japan
Email: paper@easter.kuee.kyoto-u.ac.jp

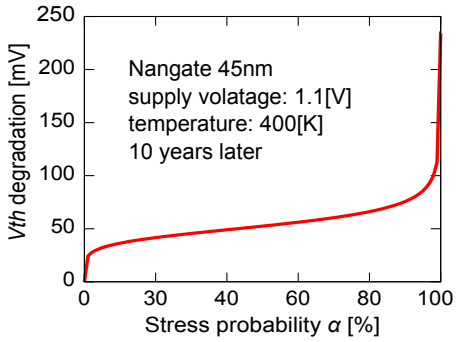

Fig. 1: ΔV_{th} degradation of pMOS with stress probability in Nangate 45 nm Open Cell Library [8], 1.1 V, 400 K, and 10 years operation.

in [5]. This model is based on the reaction-diffusion mechanism, and the long-term degradation is defined as the function of the stress probability α. The ΔV_{th} degradation at a given time t is defined as

$$|\Delta V_{\mathrm{th}}(t)| = \left(\frac{\sqrt{K_v^2 T_{\mathrm{clk}} \alpha}}{1 - \beta(t)^{1/2n}} \right)^n, \tag{1}$$

where, K_v is a function of gate-source voltage V_{gs}, threshold voltage V_{th}, and temperature. n is the time exponent which holds the value 1/6. T_{clk} is a clock period, and α expresses the stress probability of a pMOS transistor. $\beta(t)$ is a function of time, which is dependent on temperature. The stress probability α captures the effects of stress and recovery phases. In [5], [7], it is reported that $|\Delta V_{\mathrm{th}}|$ is hardly dependent on T_{clk} when $t > 1,000$ s. In this case, $|\Delta V_{\mathrm{th}}|$ at time t can be written as

$$|\Delta V_{\mathrm{th}}(t)| \approx \left(\frac{0.001 n^2 K_v^2 \alpha C t}{0.81 t_{\mathrm{ox}}^2 (1 - \alpha)} \right)^n, \tag{2}$$

where, t_{ox} is the oxide thickness, and C is a function of temperature. When $\alpha = 100\%$, $|\Delta V_{\mathrm{th}}|$ becomes infinite and the model becomes incorrect. In a similar manner to what has been shown in [7], we define an upper limit by

$$|\Delta V_{\mathrm{th}}(t)| = (K_v^2 t)^n. \tag{3}$$

Figure 1 shows the V_{th} degradation as a function of the stress probability α, which is calculated by using Eqs. (2) and (3). In this figure, we use Nangate 45 nm Open Cell Library [8] and assume 400 K and 10 years operation. The significant V_{th} degradation can be observed at near $\alpha = 100\%$. Based on this α-ΔV_{th} relation, along with the ΔV_{th}-Δd delay relation, the degradation delay Δd can be actually calculated directly from the stress probability α. Utilizing this fact, an aging-aware delay prediction method is realized in this paper.

2.2 NBTI Mitigation Techniques

As mentioned in section 1, many NBTI mitigation techniques have been proposed. Along with all the techniques introduced, a dynamic NBTI mitigation method was proposed in [9]. The method tries to mitigate the NBTI degradation by replacing gates in the design circuit with special gates that reduce the NBTI stress at their fanouts. As suggested in [9], the gates were replaced in the order of fanouts, without realizing the actual impact on the worst-path delay.

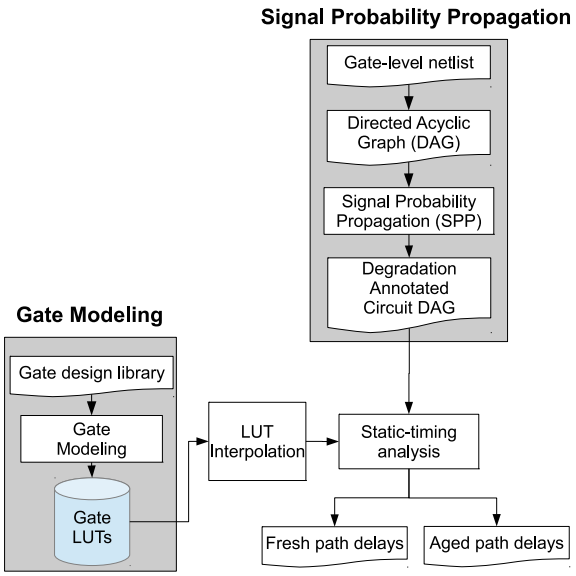

Fig. 2: Overview of the proposed LUT-based STA technique.

Hence, the paper [9] reported that a replacement, which intends to reduce the worst-path delay, in fact increases it. It would be viable to solve the problem by recalculating all path delays every time a gate is replaced in the target design. Nevertheless, fast calculation of the path delays considering the changes of V_{th} is non-trivial since it requires a large amount of delay libraries at different aging conditions of the gates. In case it were possible, it still takes excessive time, and stochastic optimization methods (e.g., genetic optimization) were unable to be applied due to this fact. This becomes the motivation for this paper to propose a much faster aging-aware path delay estimator.

3. LUT-Based Aging-Aware Static Timing Analysis

In this section, the basic elements required for performing the proposed aging-aware STA is discussed in details. As illustrated in Fig. 2, first, a directed acyclic graph (DAG) is made from the design netlist, and SPP is performed on the DAG. Meanwhile, the design library is modeled into LUT databases, and interpolated for delay calculation. In the STA calculation, each gate has its delay calculated from the LUTs, and the output path delays are the simple sums of these gate delays.

In this section, the SPP, specifically two types of SPP, will be discussed in subsection 3.1. Second, the gate modeling process is illustrated in subsection 3.2. Finally, on top of completing previous two steps, it is shown in subsection 3.3 that by doing simple interpolation and a STA-like delay calculation, path delays in large designs can be calculated in seconds.

3.1 Signal Probability Propagation

As mentioned above, a systematical approach to probability calculation is critical in predicting the aging-induced delay degradation. In this paper, intra-cell and inter-cell SPP are considered to be two separate problems, although they could potentially be solved together. Intra-cell signal SPP denotes each pMOS within a logical cell (gate) with the

NBTI-stress probability, where the NBTI-stress probability is defined as the probability of the pMOS having its gate and source voltages negatively biased. Whereas, inter-cell SPP focuses on how signal propagates through the combinational logics in a processor.

For intra-cell SPP, we manually annotated a limited number of cells for their pMOS. Each input is considered separately in terms of their probability, such that each pMOS has its input annotated relatively to the input signal(s). The NBTI-induced V_{th} degradation is then calculated based on the stress probability on its gate.

Inter-cell SPP is based on a directed acyclic graph (DAG) constructed from the gate-level netlist of the target design. To estimate the stress probability for each gate in the design, all the primary inputs applied to the circuit is fixed to follow a certain probability distribution. By constructing a topologically sorted list from the primary inputs, the probability propagates from primary inputs to each and every piece of logic in the circuit, reaching the primary outputs eventually. To make sure that probabilities propagate correctly, not only the probability mapping of a single cell has to be considered, loops in the processor circuit (e.g., forwarding logics) have as well to be handled.

To calculate the SPP along a path, an approach shown in Fig. 3 is taken. Each gate is viewed as a function (f, g) that maps all its input probabilities (α) to its output probability $(f(\alpha))$. Depending on the type of the gate, this function changes. The procedure to generate this probability mapping function is as follows. First, a table for a particular gate, AND gate in this example, is constructed as shown in Table 1. Second, from Table 1, it can be observed that the probability of outputting a logic 0 from an AND gate is the sum of the first three entries in the table. To simplify the formula, this sum can also be thought of 1 subtracting the probability of the AND gate outputting a logic 1, which becomes Eq. (4). Hence, lastly, Eq. (4) becomes the function that maps its input probabilities to the output probability.

$$P(out) = 1 - (1 - P(in1)) * (1 - P(in2)) \qquad (4)$$

By repeating the above process for each gate, a probability will start to propagate from the Q output of a D-flipflop (DFF), going through each and every gate on the path, reaching the D input of another DFF, as shown in Fig. 3. For combinational logics, this marks the end of the process. However, due to the existence of sequential logics, i.e., DFFs, combinational logics *loops* in the circuit. These logics loop in two ways: non-oscillating and oscillating. An oscillating path example is denoted in Fig. 4. It can be observed that when the DFF's output probability is 0.3, the inverter inverts it to 0.7 and feeds it back to the DFF. When the output probability of the DFF goes to 0.7, the inverter returns 0.3 to the input of the DFF, which never ends. Whereas, a non-oscillating loop converges to a stable value after looping through the path several times.

To resolve this looping of combinational logics, whether oscillating or non-oscillating, an algorithm as in Alg. 1 is used. A list of DFFs whose input/output probabilities are inconsistent is extracted from the target circuit. Note that although the circuit DAG contains DFFs, it is not cyclic, for that each node in the graph is a pin of a particular gate (or DFF), and that the D input pin (D) of the DFF is not connected to the Q output of the DFF. Hence, there is no cycle in the graph. Following the extraction, the input probabilities are applied to the output of the DFFs, and another round of probability propagation starts. This procedure is repeated until the probability converges for all non-oscillating loops. In the case of oscillating loops, the scenario that maximizes the NBTI degradation along the path is selected.

Algorithm 1 Algorithm for solving looping combinational probability propagations.

```
Require: G = Directed Acyclic Graph of target design
all_DFFs = get all DFFs from G
do
  for all DFF in all_DFFs do
    if P(DFF_out)! = P(DFF_in) and
      DFF is not marked as consistent then
      check DFF oscillation
      if DFF is oscillating then
        Choose the worst NBTI scheme
        Mark DFF as consistent
      else
        incons_dff_list.append(DFF)
  top_list = make topological list(incons_dff_list)
  for all gate in top_list do
    gate_output_probability=gate_function(
                              gate_input_probabilities)
while (incons_dff_list not empty)
```

3.2 Gate Modeling

Gates are modeled under three conditions that vary from case to case: temperature, years of aging and input vector probability distribution. Different input pins as well as rise/-fall transitions are also considered differently. In addition, since the input/output slew also propagates through combinational paths, two collections of LUTs have to be made: one for delay, another for slew.

Each collection of LUTs consists of many LUTs, where a LUT is specially designed for a particular input pin at rise, or fall, for a particular gate. For each LUT in this case, it will have three input variables: input slew, capacitive load and NBTI stress probability. The gate delay and output slew is first measured after being simulated under different conditions by varying the three variables defined above. The gate delay will then be gathered into one collection of LUTs, while the output slews are put into another. Finally, two three-dimensional LUTs of the simulated delays and output slews are constructed.

The basic circuit schematic for measuring of a gate delay of the A1 input (rising) of an AND gate is shown in Fig. 5. As mentioned, the input slew, capacitive load and NBTI stress probability are the three variables given to the circuit, and two target value are measured: the propagation delay and the output slew. From the measured values, two tables are constructed: the delay LUT and the slew LUT as shown in Figs. 6 and 7.

Table 1: AND gate under all possible input combinations.

Input Combination	Output Logic	Input 1 Probability	Input 2 Probability	Output Probability
(0, 0)	0	$P(in1)$	$P(in2)$	$P(in1)P(in2)$
(0, 1)	0	$P(in1)$	$1 - P(in2)$	$P(in1) * (1 - P(in2))$
(1, 0)	0	$1 - P(in1)$	$P(in2)$	$(1 - P(in1)) * P(in2)$
(1, 1)	1	$1 - P(in1)$	$1 - P(in2)$	$(1 - P(in1)) * (1 - P(in2))$

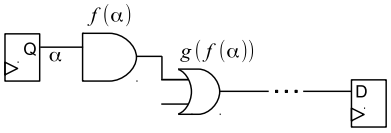

Fig. 3: DFF-to-DFF Probability propagation through combinational logics.

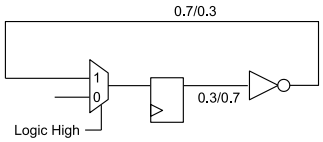

Fig. 4: Inconsistent D-flipflop with oscillating probability propagation.

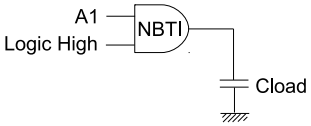

Fig. 5: Example of general simulation circuit setup for LUT construction.

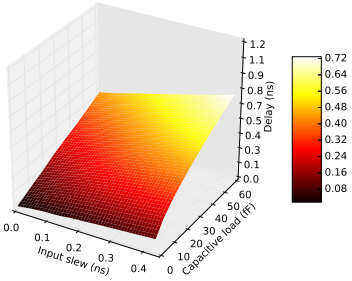

Fig. 6: Example propagation delay LUT simulated with inverter x1 falling input.

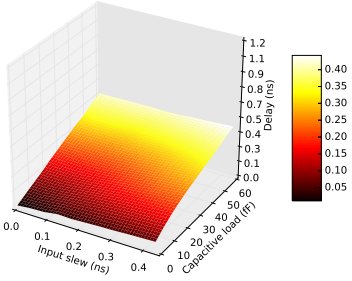

Fig. 7: Example propagation slew LUT simulated with inverter x1 falling input.

3.3 LUT Interpolation and STA

After calculating the SPP and building the delay as well as slew LUTs, interpolations are conducted on the LUTs, and path delays are calculated in an STA fashion.

Interpolation is an important part of the technique. Due to the nature of the relationship between gate delays and input and output parameters, a sophisticated interpolation method should be used to minimize the interpolation errors. This may require a complex multidimensional interpolation (above three) with consideration on the physical relationships between voltages and currents. To simplify the problem, we used a linear interpolation method, which potentially introduces interpolation errors for the gate delays estimations. Along with gate delays, as mentioned, output slew of a gate also propagates with respect to the input slew of the gate and the capacitive load of the gate. Thus, the output slew table created in the previous step is also interpolated.

After deciding the interpolation scheme for LUTs, STA is conducted in the target circuit. As mentioned earlier, this approach is not entirely STA due to the fact that paths whose delays subjected to calculation are predetermined. The reason behind this approach is that in a large VLSI design, the number of paths from one set of DFFs to another can be extremely large. Yet, only the top several tens of thousands actually have a chance of becoming the critical path. Thus, to further shorten the runtime of our delay calculation, a general STA tool is used to extract a certain amount of paths that could become critical due to the NBTI degradation. In this process, paths are extracted with capacitive load information. Following this, from the Q output pin of a DFF, slews and probabilities propagates through gates, each having their own set input/output parameters. The parameters (e.g., input slew, NBTI stress probability, etc.) are then given to the interpolated function handle, which returns the delay as well as the output slew for the specific gate requested. Finally, by summing up the gate delays along each path, the delay can be estimated efficiently along paths.

4. Experiment

4.1 Experiment Setup

Numerical experiments are conducted using a five-stage pipelined processor from a commercial IP library [10]. In the experiment, Nangate 45 nm Open Cell Library [8] is used for circuit implementation. The processor is synthesized using a logic-synthesis tool [11]. The paths with top 8% slack (a total of 25,446 paths) are extracted from the processor by a commercial STA tool [12], whose delays (aged as well as fresh) are calculated based on the proposed LUT and a SPICE simulator [13]. This is the previously mentioned predetermined “critical paths” that are vulnerable to NBTI degradation.

To generate the LUT, input slew and capacitive load are modeled in the range of [0.003ns, 470ns] and [0.01fF,

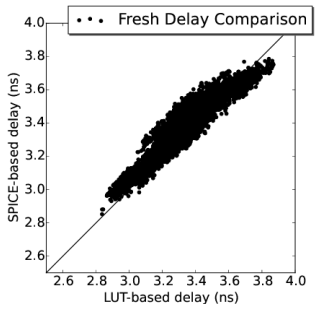

Fig. 8: Fresh delay distribution of SPICE-based simulation and LUT-based interpolation.

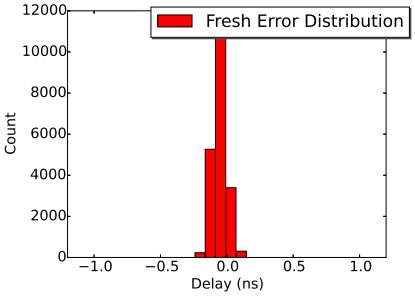

Fig. 9: Distribution error between SPICE-based path delays and LUT-based path delays for fresh processor.

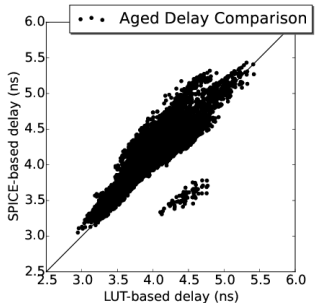

Fig. 10: Aged delay distribution of SPICE-based simulation and LUT-based interpolation.

60fF], respectively. Both input slew and capacitive load are chopped into 11 evenly distributed intervals, and the delay values at the endpoints are simulated. In terms of NBTI stress probability, 281 intervals were created in the range of [0.0, 1.0]. In ranges where probability-induced delay degradation changes significantly ([0.0, 0.1] and [0.9, 1.0]), more intervals are distributed.

As for the interpolation, a python three-dimensional interpolation library was used [14]. Due to the limitation of the library, only linear interpolation was used for 3D interpolation.

4.2 Experiment Result

Figure 6 shown previously in subsection 3.2 demonstrates the evaluation over a dense mesh after the interpolation for a X1 inverter without any NBTI stress. It can be observed that generally, as input slew and capacitive load increase, the delay increases. However, for input slew, a log-like curve is derived. This can be explained by the fact that input slew is becoming too long (> 200 ps) for an inverter gate that has a propagation delay in the range of 30-80 ps typically. Thus, the transition happens as soon as the input voltage reaches

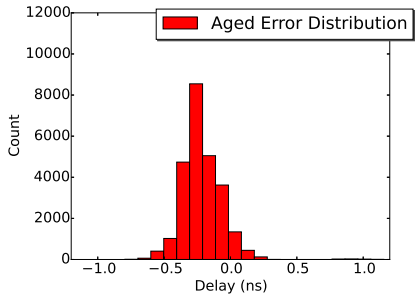

Fig. 11: Distribution error between SPICE-based path delays and LUT-based path delays for aged processor.

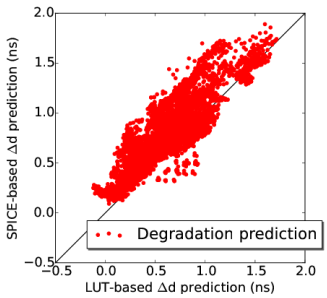

Fig. 12: Distribution of SPICE-based Δd prediction and that calculated from the LUT-based method.

Table 2: Result of proposed technique compared to HSPICE.

	Proposed	SPICE
Calc. Time	10 sec.	960 min.
Max. Error (ns)	20 %(13 % after smoothing)	-

the switching point, giving a saturating delay increase. This is also indicated in Fig. 7 that the input slew to the gate does not have strong effects on the output slew of the gate, but is mainly dominated by the capacitive load driven by the gate.

The runtime and error information is summarized in Table 2. Note that we used SPICE as a reference design, so that the SPICE is considered to be 100 % accurate in this case. Compared to SPICE simulation that takes more than 960 minutes to calculate all the path delays, our technique accomplish the same task within 10 seconds, giving an approximate speedup of 5760x. In addition, our method was implemented in the Python language without much optimization. Further speedup is expected if the programs are written in C or Fortran. The maximum error is found to be 20 %, without error smoothing which will be discussed in subsection 4.3.

Fresh and aged path delay distributions are presented in Fig. 8 and Fig. 10, respectively, with their error histogram plotted in Fig. 9 and Fig. 11. It can be observed that generally, delays are distributed over the $x = y$ line. However, especially in the aged path delay calculations, a general trend of underestimation is observed. This can be explained by the fact that log-like curve will always suffer from underestimation when the interpolation method is linear. Thus, certain compensation should be added to the interpolation algorithm. The correlation between the LUT-based path delay estimation and SPICE simulation is 0.941 and 0.873 for fresh and aged, respectively. A drop in correlation is observed for aged delay estimation due to the errors introduced in adding the NBTI-stress dimension.

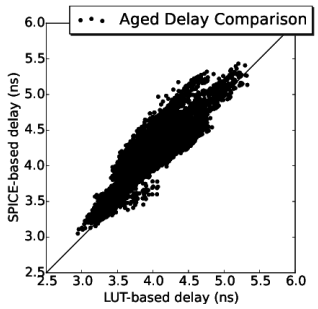

Fig. 13: Distribution of SPICE-based path delays and LUT-based path delays for aged processor with modified X1 inverter delay.

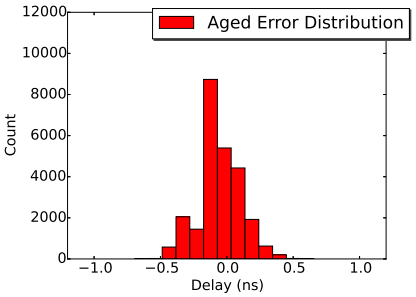

Fig. 14: Distribution error between SPICE-based path delays and LUT-based path delays for aged processor with modified X1 inverter.

The accuracy of prediction on Δd, the NBTI degradation, is also studied. In Fig. 12, it can be observed that the prediction accuracy has a trend of being underestimated in general. With some paths having extra-large overestimations, as in the path delay prediction. This distribution reveals the fact that quite a huge amount of error was introduced in the process of adding NBTI degradation into consideration.

4.3 Error Analysis

As stated above, overall, errors come from underestimations. Especially for aged delay prediction, as in Fig.11, the peak of the distribution is at -0.2 ns, rather at 0 ns. However, it is worth to mention the huge overestimation for aged processor paths in the lower-critical range in the lower-right corner in Fig. 10. After investigating the path information, it was observed that a great overestimation was given to X1 inverter in some ranges of capacitive load. In other words, for some capacitive load ranges, the X1 inverter tends to be greatly overestimated (almost doubled in some situation), while on other ranges, it is underestimated.

To show that this error is mainly due to interpolation error, a simple error-smoothing method is implemented for verification purpose. Thus, obviously, a more sophisticated error-smoothing could be implemented to further reduce the interpolation error. In this experiment, a simple error smoothing scheme of artificially reducing the delay of X1 inverter over a specific delay range (e.g., $> 80ps$) is conducted, and Fig. 13 illustrates that the overestimation is completely disappeared after this trivial smoothing.

After biasing the result from X1 inverter delay reduction to compensate for the overall underestimation from interpolation error, the maximum error becomes 13 % with the error distribution revealed in Fig. 14. It can be observed that with a good smoothing algorithm, the accuracy of our technique can be further improved.

5. Conclusion

In this paper, we proposed a fast and efficient method for predicting the NBTI-induced delay degradation in large designs like processors. The method utilizes signal probability propagation, lookup tables and static timing analysis and can calculate aged path delays for large design within a matter of seconds. This proposed technique is applied to an example five-stage RISC processor to verify its effectiveness. It is shown that using our method, the calculation of delays of 25,446 paths achieved a 5760x speedup, with a maximum error of 13 %.

Acknowledgment

This work was partially supported by MEXT/JSPS KAKENHI Grant No. 26280014, 15K15960. The authors also acknowledge support from VDEC with the collaboration with Synopsys Corporation.

References

[1] D. R. Bild, R. P. Dick, and G. E. Bok, "Static NBTI reduction using internal node control," *ACM Transactions on Design Automation of Electronic Systems*, vol. 17, no. 4, pp. 45–75, 2012.

[2] A. Abdollahi, F. Fallah, and M. Pedram, "Leakage current reduction in CMOS VLSI circuits by input vector control," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 12, no. 2, pp. 140–154, 2004.

[3] M. Ebrahimi, F. Oboril, S. Kiamehr, and M. B. Tahoori, "Aging-aware logic synthesis," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 2013, pp. 61–68.

[4] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[5] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, "Predictive modeling of the NBTI effect for reliable design," in *Proceedings of IEEE Custom Integrated Circuits Conference*, 2006, pp. 189–192.

[6] H. Awano, M. Hiromoto, and T. Sato, "BTIarray: a time-overlapping transistor array for efficient statistical characterization of bias temperature instability," *IEEE Transactions on Device and Materials Rreliability*, vol. 14, no. 3, pp. 833–843, 2014.

[7] H. Konoura, Y. Mitsuyama, M. Hashimoto, and T. Onoye, "Comparative study on delay degrading estimation due to NBTI with circuit/instance/transistor-level stress probability consideration," in *Proceedings of IEEE International Symposium on Quality Electronic Design*, 2010, pp. 646–651.

[8] Si2.org, "Nangate 45nm Open Cell Library," http://www.si2.org, [Online available].

[9] S. Bian *et al.*, "A processor-level NBTI mitigation technique of applying anti-aging gate control through instruction set architecture," *IEICE Technical report, VLD*, vol. 114, no. 476, pp. 49–54, 2015.

[10] *Processor Designer User Guide Version G-2012.09*, Synopsys, Inc., 2012.

[11] *Design Compiler User Guide Version D-2010.06*, Synopsys, Inc., 2010.

[12] *PrimeTime Fundamental User Guide Version D-2010.06*, Synopsys, Inc., 2010.

[13] *HSPICE User Guide: Basic Simulation and Analysis Version I-2013.12*, Synopsys, Inc., 2013.

[14] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–, [Online; accessed 2015-07-08]. [Online]. Available: http://www.scipy.org/