AVATAR: An Aging- and Variation-Aware Dynamic Timing Analyzer for Application-based DVAFS

Zuodong Zhang, Zizheng Guo, Yibo Lin, Runsheng Wang, Ru Huang
School of Integrated Circuits, Peking University, Beijing, China

***Abstract*—As the timing guardband continues to increase with the continuous technology scaling, better-than-worst-case (BTWC) design has gained more and more attention. BTWC design can improve energy efficiency and/or performance by relaxing the conservative static timing constraints and exploiting the dynamic timing margin. However, to avoid potential reliability hazards, the existing dynamic timing analysis (DTA) tools have to add extra aging and variation guardbands, which are estimated under the worst-case corners of aging and variation. Such guardbanding method introduces unnecessary margin in timing analysis, thus reducing the performance and efficiency gains of BTWC designs. Therefore, in this paper, we propose AVATAR, an aging- and variation-aware dynamic timing analyzer that can perform DTA with the impact of transistor aging and random process variation. We also propose an application-based dynamic-voltage-accuracy-frequency-scaling (DVAFS) design flow based on AVATAR, which can improve energy efficiency by exploiting both dynamic timing slack (DTS) and the intrinsic error tolerance of the application. The results show that a 45.8% performance improvement and 68% power savings can be achieved by exploiting the intrinsic error tolerance. Compared with the conventional flow based on the corner-based DTA, the additional performance improvement of the proposed flow can be up to 14% or the additional power-saving can be up to 20%.**

I. Introduction

With the continuous shrinking of CMOS technology nodes, design margin has become extremely tight due to the reliability and manufacturability issues like transistor aging and random process variation (local variation) [1]. To guarantee the parametric yield and circuit lifetime, designers resort to adding timing guardbands. However, these guardbands are increasing rapidly with the technology scaling, and eventually obliterating gains from device scaling [2].

To compensate for the performance loss caused by the aforementioned issues, better-than-worst-case (BTWC) design is proposed. BTWC techniques usually go beyond the conservative static timing constraints, and leverage optimization techniques such as dynamic-voltage-frequency-scaling (DVFS) [3]–[5]. Based on the observation that many emerging applications have nondeterministic specifications or are robust to noise, some BTWC techniques further explore the possibility of trading precision, accuracy, and reliability for resource usage, such as application-level error tolerance and dynamic-voltage-accuracy-frequency-scaling (DVAFS) [6]–[8]. All these techniques rely on the dynamic information obtained from dynamic timing analysis (DTA).

However, the previous DTA tools for BTWC techniques are usually performed by delay-annotated gate-level simulation with a post-processing program to calculate the dynamic timing slack (DTS) [4], [6]. This method has two shortcomings. First, the delay used in the gate-level simulation is obtained from graph-based static timing analysis (STA), which suffers from the pessimistic nature of graph-based analysis [9]. Secondly, it needs extra guardbands for path delay to cover the worst-case corners of aging and variation (i.e., each cell is in the 3σ local process corner with max transistor aging). The pessimism in both delay calculation and guardbands ultimately lead to an over-designed system.

⋆Corresponding author: yibolin@pku.edu.cn

TABLE I: Comparison among various timing analyzer.

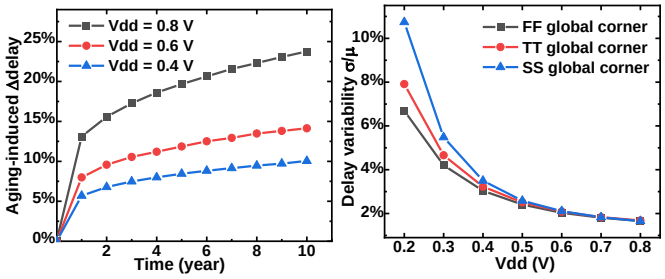
Methods	A	B	C	D
Aging-aware STA [10], [11]	✓	×	×	×
STA with POCV [12]	×	✓	×	×
Delay-annotated DTA [4], [6]	×	×	✓	×
Event-based DTA [13]	×	×	✓	✓
AVATAR (this work)	✓	✓	✓	✓

(A) Supports gate-level aging analysis.
(B) Supports gate-level random process variation analysis.
(C) Exploits dynamic timing slack to improve performance and efficiency.
(D) Avoids the pessimism of graph-based assumption.

To better support the BTWC design methodology, in this paper, we propose AVATAR, an aging- and variation-aware dynamic timing analyzer. AVATAR can accurately estimate the timing error rate under the impact of transistor aging and random variation, thus enabling a better DVAFS design flow. Table I summarizes existing timing analyzers compared with AVATAR. To the best of our knowledge, AVATAR is the first DTA engine that supports gate-level aging and random variation models. The major contributions of this work are as follows:

1) We propose AVATAR, an aging- and variation-aware dynamic timing analyzer, which can estimate the impact of transistor aging and random variation on timing analysis. Compared with the conventional corner-based DTA, AVATAR can accurately calculate the aged delay and delay variability, which can help avoid over-design.
2) We propose an application-based DVAFS flow based on AVATAR, an optimization technique that enables application-based dynamic clock/frequency adjustment. The proposed flow can determine the application-specific V_{min}/f_{max} according to the distribution of dynamic delays obtained from AVATAR and the intrinsic error tolerance.
3) We demonstrate that exploiting intrinsic error tolerance of the application to accept timing errors can boost the performance up to 45.8% or reduce power up to 68% in error-tolerant applications.
4) The results also show that, compared with the flow based on the corner-based DTA, the additional performance improvement of the proposed application-based DVAFS flow based on AVATAR can be up to 14% or the additional power-saving can be up to 20%.

The rest of this paper is organized as follows. Section II introduces the background of reliability issues in timing analysis and application-based DVFS. Section III presents details of the algorithm. Section IV demonstrated the experimental results of our algorithm. Finally, Section V concludes the paper.



(a) (b)

Fig. 1: (a) Increase in FO4 delay due to transistor aging effect under different V_{dd}. (b) FO4 variability normalized to the value at nominal V_{dd}. The data is obtained from HSPICE simulation with a commercial 16/14 nm modelcard.

II. Preliminaries

A. Reliability and Manufacturability Issues in Timing Analysis

As CMOS technology shrinks to nanoscale, transistor aging and process variation become more and more significant, making timing closure and signoff increasingly challenging [12]. Transistor aging effect increases the gate delay and transition time, which increases the possibility of timing violations over time. Random variation needs an additional design margin to ensure the yield, especially in near-threshold voltage (NTV). Fig. 1 shows the impact of transistor aging and random variation on the delay of Fan-out-of-4 (FO4) inverter, which is routinely used to optimize critical paths. It shows that the aging effect is the major reliability concern in the nominal voltage, while the random variation becomes more and more pronounced with the voltage decreasing.

To counteract the impact of transistor aging and random variation, designers typically employ a guardband when checking the timing [2]. However, the aging and variation guardbands are commonly calculated in the worst-case corner and increase rapidly due to device scaling. As a result, the impact of guardbanding on performance, power, and area (PPA) has been increasing, and eventually obliterating PPA benefits from device scaling. Therefore, aging- and variation-aware design flow is urgently needed.

B. Dynamic Timing Analysis and Application-based DVFS

Application/instruction-based DVFS is widely adopted to improve the energy efficiency or performance in embedded system [3]–[5]. It is based on the observation that an instruction executed on an embedded processor may not utilize all the functionalities. Therefore, there may exist a timing slack between the most critical path reported from STA and the longest path triggered by the instructions. Distinguish from previous static timing slack (STS), such timing slack is called DTS. The presence of DTS means that for the specific application/instruction, the V_{dd} or *clock cycle* could have been smaller. Therefore, using application/instruction-specific V_{min}/f_{max} instead of the fixed V_{dd}/f_{max} can increase the energy efficiency/performance. At the same time, many emerging applications are robust to computational errors, which means that the application-based voltage/frequency adjustment can be more aggressive for these error-tolerant applications. Hence DVAFS is proposed to trade accuracy for more performance improvements and/or power savings.

All these techniques rely on DTA to determine the application/instruction-specific V_{min}/f_{max}. Existing DTA tools either leverage delay-annotated gate-level simulation with a post-processing program to calculate DTS [4], [6] or event-based DTA algorithms considering propagation of signal transition events [13]. However, none of these tools consider the impact of transistor aging and random variation on timing. Thus the results obtained from these tools need extra aging and variation guardbands to cover the worst-case aging and variation conditions. Such pessimism in DTA hinders the further improvement of application-based DVAFS.

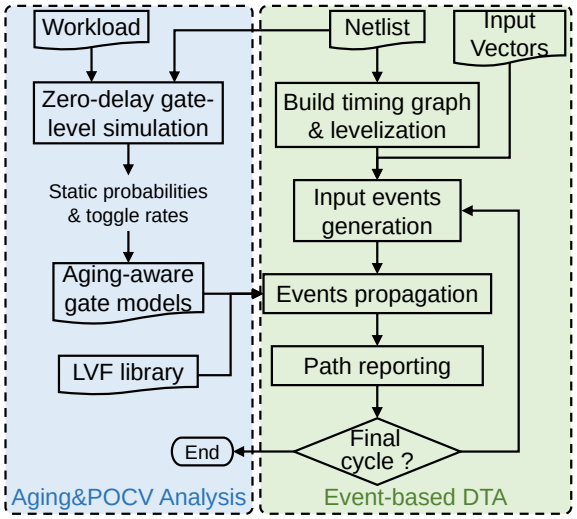


Fig. 2: The proposed aging- and variation-aware DTA flow of AVATAR.

III. Algorithm

In this section, we will explain the implementation details of AVATAR, and show how AVATAR enables better application-based DVAFS design flow. The overall task flow of AVATAR is shown in Fig. 2. It contains two main parts: gate-level aging and variation analysis and event-based DTA.

A. Gate-level Aging and Variation Model Characterization

To enable the aging and variation analysis, we first build the gate-level aging model and variation library. We utilize the method proposed in [10] to build the gate-level aging model. The basic idea is to use the first-order Taylor expansion to model the aging effect on the *delay* and *transition time (tr)*:

$$delay_{aged} = delay_{fresh} + \sum a_i \times \Delta V_{thi} \tag{1}$$

$$tr_{aged} = tr_{fresh} + \sum b_i \times \Delta V_{thi} \tag{2}$$

where the sensitivity coefficients a_i, b_i vary with different input slews and output loads, so the model uses the first-order linear model to fit the dependency.

$$a_i = a_{i0} + \alpha_{ai} \times slew + \beta_{ai} \times load \tag{3}$$

Since negative bias temperature instability (NBTI) dominates the transistor aging in digital circuits [14], the model only considers V_{th} degradation of PMOS.

In the built gate-level aging model, the inputs are: the timing arc queried, the input slew, the output capacitance, the ΔV_{th} of each transistor, and working conditions(i.e., supply voltage, and temperature); the outputs of the model are the corresponding aged cell delay and transition time.

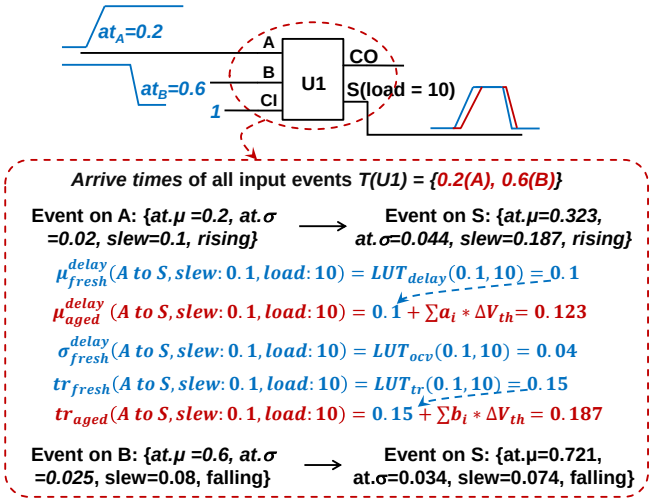


Fig. 3: An example of the event propagation for cell FA.

For random variation analysis, we adopt parametric on-chip variation (POCV) analysis [12]. POCV can be represented with the liberty variation format (LVF), a widely used industry standard. LVF models capture the delay impact of variation at multiple slew and load combinations in look-up tables. To reduce the characterization complexity, we model aging and random variation as two independent effects [15] and only consider the effect of aging on the means value of cell *delay* and *tr*. It should be noted that the aging-aware model and LVF library characterization requires lots of SPICE simulations, but it is done only once for each technology.

B. Workload Analysis

The aim of workload analysis is to simulate a realistic working scenario and provide the necessary information for aging analysis. As mentioned before, the extra inputs of the aging-aware model are the ΔV_{th} of each transistor. Therefore, the final results of workload analysis are the ΔV_{th} of each transistor in the circuit.

The workload analysis is implemented in two steps: in the first step, we use the zero-delay gate-level simulation to obtain the static probability and toggle rate of each internal net; then, we employ the cell-level analytical model [10] and device-level aging model to calculate the ΔV_{th} of each transistor. It should be noted that the zero-delay gate-level simulation can be replaced by the delay-annotated gate-level simulation or the event-based DTA, which will be more accurate, but the runtime will also increase significantly. After obtaining the ΔV_{th} of each transistor, the DTA engine can then query the aged cell *delay* and *tr* of any gate by the built aging-aware model.

C. Event-based DTA

AVATAR performs DTA based on the cycle-by-cycle event generation and propagation. In the event propagation algorithm, AVATAR uses the aforementioned gate-level aging model to calculate the aged cell delay. And to enable random variation analysis, deterministic delays are replaced with a distribution of delays, modeled with two parameters, the mean value (μ) and the stand deviation (σ). Therefore, the dynamic delay calculated by AVATAR is also a distribution rather than a fixed value.

In the event-based DTA, a digital switching of a pin is defined as an event. Each event contains at least four attributes: signal type, *slew*, and the μ and σ of *arrival time (at)*. Unlike the conventional delay-annotated DTA, in the event-based DTA, each event records its own slew, and the cell delay is calculated by the slew of the input event. Therefore, there is no need to merge the input slews of multi-input cells, and the path delay calculation is more accurate.

As shown in Fig. 2, the first step of the event-based DTA is building the timing graph and levelizing the graph to build level-by-level dependencies of the gate for events propagation.

The next step is the cycle-by-cycle timing analysis, which contains three main sub-steps in each cycle: input event generation, event propagation, and path reporting. In the first step, AVATAR reads the input vector of the current cycle and compares the value of each bit with the value of the previous cycle. A changed value indicates that an event occurs on the specific input pin.

Then, AVATAR propagates the input events to the timing endpoints (inputs of flip-flops or output ports). Propagation algorithms are divided into propagation through a net and propagation through a gate. Propagation through a net only increases the *at* of the events by net delay. While each gate will generate new events according to the input events.

Algorithm 1 presents the gate propagation process. We first build a sorted list of all arrival times of input events (lines 1-3). Then, we simulate the logic behavior of the gate and get the output value in chronological order (lines 5-6). A changed output value indicates an output event in the corresponding output pin. For each output event, we calculate the μ and σ of the corresponding cell *delay* and the output *tr* by LVF library and the aging model (lines 8-10). Then, we calculate the μ and σ of the output event *at* by cell *delay* and the corresponding input event (lines 11-14).

Fig. 3 gives an example to better describe the event propagation algorithm and the timing calculation. There are two input events for the full adder (FA) cell *U1* in the current cycle, so the list of time is $\{0.2, 0.6\}$. We first simulate the logic of FA at these two time points. It can be found that the value of the output pin *S* changes at each moment, which means that each input event triggers an output event on the output pin *S*. Then, we calculate the corresponding aged cell *delay*, *delay variation* and aged output *tr*. Finally, we build the output event and add it to the events list of pin *S*.

In the last step, AVATAR checks all the events of the endpoints, and reports the *at* (μ and σ) of the last events as the dynamic delay distribution. After reporting all endpoints, the algorithm clears all events and only reserves the state of pins as the initial state for the next cycle. This operation ensures AVATAR to have no additional memory overhead.

D. Parallel Acceleration

To accelerate multi-cycle simulation, we adopt cycle parallelism in event-based DTA. We divided the cycles to be simulated into *n* equal parts and assign them to *n* different threads for execution. It is noted that, there is a *m-cycle* overlap between different parts, and *m* is equal to the pipeline depth, because the circuit takes *m* cycles to be in the state that it is supposed to be in. Using cycle parallelism avoids complex netlist splitting algorithms, and threads are executed independently, which results in better acceleration.

E. Application-based DVAFS

We target application-based DVAFS as one of the use case of AVATAR. The DVAFS design flow is shown in Fig. 4. The key idea is to use AVATAR to determine the application-specific V_{min}/f_{max} as the guide for the dynamic voltage/frequency adjustment at runtime.

Algorithm 1: Propagate events through the gate.

Input: gate g_i, event list of input pins
Output: event list of output pins

build a list of time $T(g_i)$;
$T(g_i) \leftarrow get\_input\_event\_arrive\_time(g_i)$;
sort $T(g_i)$ and ensure that there are no two input events with the same at;
for *all time* $t_i \in T(g_i)$ **do**
  find the corresponding input event $event\_in$;
  get the states of input pins at time t_i, and calculate the output value;
  for *all pin* $p_i \in PIN_{out}$ *whose state changed* **do**
    `// calculate cell delay and tr`
    $\mu^{delay}_{aged} = LUT(event\_in.slew, load) + aging\_model(event\_in.slew, load, \Delta V_{th})$;
    $\sigma^{delay}_{fresh} = LUT(event\_in.slew, load)$;
    $tr_{aged} = LUT(event\_in.slew, load) + aging\_model(event\_in.slew, load, \Delta V_{th})$;
    `// build a new event`
    build a output event $event\_out$;
    $event\_out.at.\mu = event\_in.at.\mu + \mu^{delay}_{aged}$;
    $event\_out.at.\sigma = Sqrt((event\_in.at.\sigma)^2 + (\sigma^{delay}_{fresh})^2)$;
    $event\_out.slew = tr_{aged}$;
    add $event\_out$ to the events list of p_i;

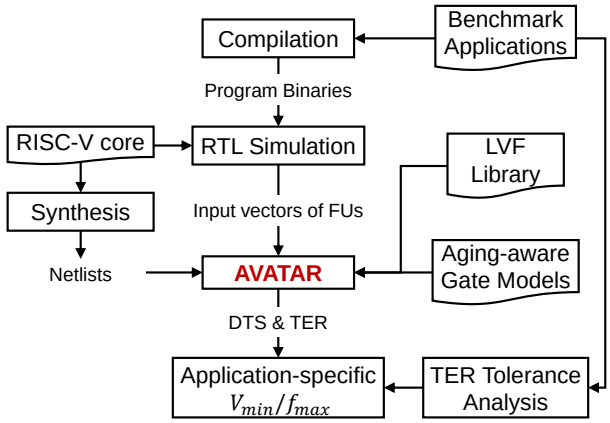


Fig. 4: The proposed application-based DVAFS design flow based on AVATAR.

Moreover, previous work has demonstrated that many computational-intensive applications usually feature an intrinsic error-resilience property [16]. Therefore, we extend the conventional DVFS to DVAFS, in which performance and energy efficiency are further improved by allowing a small percentage of computational errors.

Firstly, we compile the applications to binary files, and execute them in the RTL simulation of the target processor. This step is to obtain the input vectors of the functional units (FUs). It should be noted that AVATAR can perform the DTA for the full design, but we only consider the dynamic delay of FUs because previous work [4] shows that the maximum dynamic delays occur mainly in FUs, and the timing error occurred in FUs can be tolerant in some applications.

Then, we synthesize the RTL codes into the gate-level netlists, and use AVATAR to perform aging- and variation-aware DTA for the FUs. AVATAR will report the aged dynamic delay and delay variability

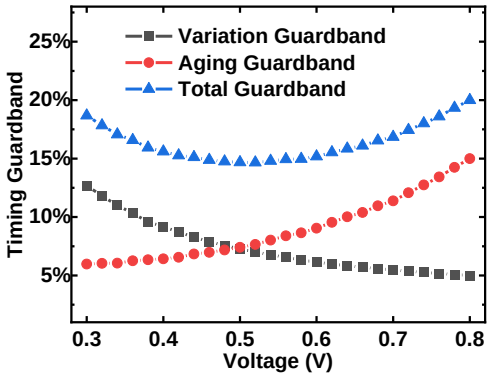


Fig. 5: The required aging guardband and variation guardband at different voltages.

per cycle for FUs running the applications. Thus the timing error rate (TER) can be estimated according to the delay distribution and toggle rate. Finally, according to the results of AVATAR and the inherent error tolerance of the application, we can determine the V_{min}/f_{max} for each application.

IV. Experimental Results

We implemented AVATAR in C++ and conducted all the experiments on a Linux machine with Intel Xeon E5-2650 at 2.20GHz and 64 GB RAM. AVATAR and the conventional event-based DTA are both executed in parallel with 32 threads. An open RISC-V core RI5CY and a set of embedded application benchmarks [17] are used to validate the proposed application-based DVAFS. The RISC-V core is synthesized with the open-source Nangate 15nm standard cell library [18], using Synopsys Design Compiler. Static timing analysis and power analysis are performed with Synopsys Primetime. Siliconsmart is used to generate the LVF libraries at each V_{dd} between 0.8V and 0.4V at 0.2V interval with a commercial 16/14nm FinFET modelcard. Gate-level aging models are built with an NBTI model from [14]. Our baseline for the results is the processor operating at nominal voltage and frequency (833MHz and 0.8V).

We use the following two methods to determine the application-specific V_{min}/f_{max}:

1) Using the corner-based DTA with extra aging guardband and variation guardband in dynamic delay calculation [3], [4]. The final dynamic delay is calculated as $delay * (1 + total\_guardband)$. We assume the aging guardband is 15% [10] and the random variation guardband is 5% [19] at nominal V_{dd}. Then, we use FO4 delay as the representative cell to characterize the trend of guardband at different V_{dd} [20]. The result is shown in Fig. 5. As the voltage decreases, the total guardband decreases first due to the aging effect, but as the voltage decreases to the near-threshold region, the rapid rise of variation guardband leads to the increase of the total guardband.
2) Using AVATAR. The results of AVATAR already include the impacts of aging and variation, so no extra guardband is needed. The final delay is calculated as $\mu(delay) + 3 * \sigma(delay)$.

To avoid the randomness of the input data from affecting the results, we simulate at least 200k cycles and up to 2M cycles per application. By comparing the performance and energy efficiency of these two methods, we will show how AVATAR enables better DVAFS at runtime.

TABLE II: Performance improvement from application-based DVAFS based on the corner-based DTA and AVATAR. The processor operates at the nominal V_{dd} of 0.8V.

Benchmark [17]	Corner-based DTA [3], [4]			AVATAR (this work)				
	Delay (ps)	Max Freq. (MHz)	Impro. (vs STA)	Delay (ps)		Max Freq. (MHz)	Impro. (vs STA)	Additional Benefits (vs corner-based DTA)
				μ	σ			
SHA	1054.95	948	13.75%	957.65	7.65	1020	22.38%	8.63%
AES_CBC	1132.18	883	5.99%	1028.41	7.76	951	14.10%	8.11%
FIR	1092.71	915	9.82%	991.29	7.56	986	18.35%	8.53%
BubbleSort	772.28	1290	55.38%	701.84	7.94	1380	65.36%	9.98%
Motion_Detection	1043.48	958	15.00%	945.27	7.56	1030	23.97%	8.97%
CNN	1151.86	868	4.18%	1045.53	7.69	936	12.30%	8.12%
Convolution	1151.78	868	4.19%	1045.70	7.69	936	12.28%	8.09%
2d_Filter	1068.33	936	12.33%	927.48	7.37	1050	26.37%	14.04%
MatrixMult	1092.05	916	9.89%	991.29	6.75	989	18.63%	8.74%
DCT	852.47	1170	40.77%	768.55	6.72	1270	52.15%	11.38%

A. Performance and Power Improvement

Table II shows the performance improvement by adopting the performance-first DVAFS strategy based on the corner-based DTA and AVATAR. The table shows that different applications can expose different maximum dynamic delays, depending on which process features are triggered. For example, the dynamic delays of *CNN, AES_CBC* are close to the critical path delay given by STA, because these applications can trigger the critical path of the multiplier. While the dynamic delays of *BubbleSort, DCT* are much less than the clock cycle, because the instructions of *BubbleSort* only trigger the comparator in ALU. Although *DCT* uses the hardware multiplier, the inputs are 8-bit images, which cannot trigger the high bits of the multiplier.

The table shows that the flow based on the corner-based DTA can improve the performance by an average of 17.13%, while the flow based on AVATAR can improve the performance by an average of 26.59%. The additional improvement of AVATAR over the corner-based DTA varies in different applications. Overall, the additional performance improvement of AVATAR is 8% to 14%. This is because that different applications trigger different hardware paths, and corner-based DTA uses the fixed guardband to all paths. In practice, however, each path has a different delay degradation and variation, depending on the circuit topology and workload.

Table III shows the power saving by adopting the power-first DVAFS strategy based on the corner-based DTA and AVATAR. On average, the flow based-on the corner-based DTA can save the power by 38%, while the flow based on AVATAR can save the power by 50.57%. Compared with the corner-based DTA, the additional power-saving of AVATAR can be up to 20%. It should be noted that the minimal application-specific V_{min} is 0.46V, which is greater than the minimum energy point (about 0.2V). Therefore, the flow based on AVATAR also improves energy efficiency.

B. Improvement in error-tolerant Applications

Fig. 6 shows the histogram of the dynamic delay for the RISC-V core running *CNN* application. The dynamic delay is calculated by AVATAR. It can be seen that although *CNN* can trigger some long paths, the trigger rate are rather small. This means that a significant amount of power and performance needs to be sacrificed to prevent small probability of error. Therefore, the minimal clock cycle can be much smaller if we relax the constraint of accuracy.

The previous work shows that LeNet-5 can tolerate error rates of 1e-4 to 1e-3 without affecting the network accuracy [21]. Although different CNN structures and databases have different intrinsic error tolerance, in this paper, we use 1e-3 as an upper bound of TER

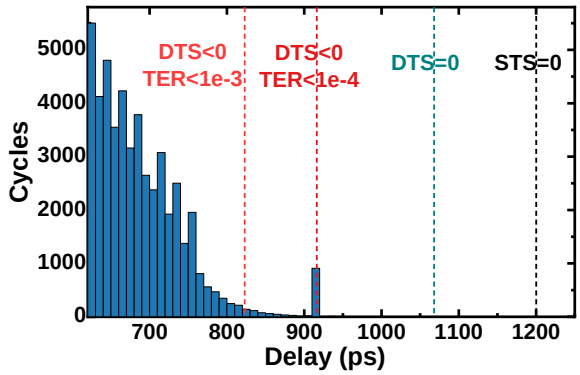
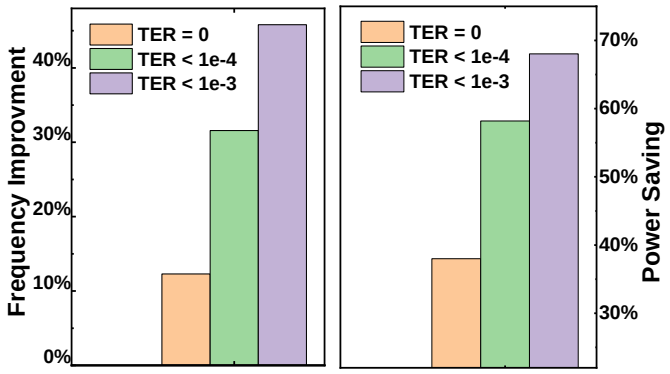


Fig. 6: Histogram of dynamic delays per cycle for the RISC-V core running CNN application.



(a) (b)

Fig. 7: (a) The performance gains and (b) the power saving by allowing small number of timing errors. The benchmark application is CNN.

to demonstrate the effectiveness of DVAFS. The results are shown in Fig. 7. Allowing timing errors to occur can leading to a 45.8% performance improvement or 68% power saving. For some CNN structures or databases with less error tolerance, using 1e-4 as the upper limit of TER can still achieve a 31.6% performance gain or 58.2% power saving. The results show that DVAFS is a promising optimization technique for all error-tolerant applications.

TABLE III: Power saving from application-based DVAFS based on the corner-based DTA and AVATAR. The processor operates at the nominal frequency of 833MHz.

Benchmark [17]	Corner-based DTA [3], [4]			AVATAR (this work)				
	Min Vdd (V)	Delay (ps)	Power Savings	Min Vdd (V)	Delay (ps)		Power Savings	Additional Benefits (vs corner-based DTA)
					μ	σ		
SHA	0.66	1177.19	38.00%	0.60	1144.04	11.44	50.73%	12.73%
AES_CBC	0.72	1191.76	23.25%	0.64	1165.70	10.45	42.41%	19.16%
FIR	0.68	1193.89	33.32%	0.62	1152.21	10.72	46.65%	13.33%
BubbleSort	0.48	1185.59	71.01%	0.46	1111.08	20.18	73.76%	2.75%
Motion_Detection	0.64	1193.09	42.41%	0.60	1130.30	11.30	50.73%	8.32%
CNN	0.74	1193.56	17.87%	0.66	1157.36	9.88	38.00%	20.13%
Convolution	0.74	1193.81	17.87%	0.66	1157.58	9.88	38.00%	20.13%
2d_Filter	0.66	1190.69	38.00%	0.60	1155.70	10.61	50.73%	12.73%
MatrixMult	0.68	1192.24	33.32%	0.62	1146.01	9.48	46.65%	13.33%
DCT	0.52	1186.73	64.92%	0.50	1103.88	14.49	68.04%	3.12%

C. Runtime Evaluation

In our application-based evaluation, when executed in parallel with 32 threads, AVATAR took 2993.45s to simulate 1M cycles for a FU containing 4400 gates, taking about twice the time of the corner-based DTA. The increased time is mainly for aging analysis, because the device-level aging model contains lots of exponential and logarithmic operations. We should note that AVATAR has been developed for CPU execution, and the runtime can be improved by up to 10 times if further utilizing graphic processing units (GPUs) [22].

V. Conclusion

In this paper, we present AVATAR, an aging- and variation-aware dynamic timing analyzer, which can calculate the dynamic circuit delay with the impact of transistor aging and random variations. We evaluate AVATAR in application-based DVAFS design flow, which leverages the application-specific V_{min}/f_{max} to obtain performance or efficiency gains. The results demonstrate that exploiting intrinsic error tolerance to accept timing errors can improve performance by up to 45.8% or increase the energy efficiency by up to 68% for error-tolerant applications. The results also demonstrate that the additional performance improvement of the application-based DVAFS flow based on AVATAR can be up to 14% or the additional power-saving can be up to 20%, compared with the conventional flow. Our future work includes accelerating the aging model characterization and the event-based DTA, and leveraging AVATAR to enable other optimization design flows.

Acknowledgements

This work was supported in part by the National Key R&D Program (2020YFB2205500), NSFC (62125401, 62141404, 62034007) and the 111 Project (B18001).

References

[1] R. Huang, X. Jiang, S. Guo, P. Ren, P. Hao, Z. Yu, Z. Zhang, Y. Wang, and R. Wang, "Variability-and reliability-aware design for 16/14nm and beyond technology," in *Proc. IEDM*, 2017, pp. 12.4.1–12.4.4.

[2] A. Rahimi, L. Benini, and R. K. Gupta, "Variability mitigation in nanometer cmos integrated systems: A survey of techniques from circuits to software," *Proceedings of the IEEE*, vol. 104, no. 7, pp. 1410–1448, 2016.

[3] H. Cherupalli, R. Kumar, and J. Sartori, "Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems," in *Proc. ISCA*, 2016, p. 671–681.

[4] J. Constantin, L. Wang, G. Karakonstantis, A. Chattopadhyay, and A. Burg, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *Proc. DATE*, 2015, pp. 381–386.

[5] T. Jia, R. Joseph, and J. Gu, "19.4 an adaptive clock management scheme exploiting instruction-based dynamic timing slack for a general-purpose graphics processor unit with deep pipeline and out-of-order execution," in *Proc. ISSCC*, 2019, pp. 318–320.

[6] I. Tsiokanos, L. Mukhanov, and G. Karakonstantis, "Low-power variation-aware cores based on dynamic data-dependent bitwidth truncation," in *Proc. DATE*, 2019, pp. 698–703.

[7] O. Assare and R. Gupta, "Accurate estimation of program error rate for timing-speculative processors," in *Proc. DAC*, 2019.

[8] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Dvafs: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling," in *Proc. DATE*, 2017, pp. 488–493.

[9] J. Bhasker and R. Chadha, *Static timing analysis for nanometer designs: A practical approach*. Springer Science & Business Media, 2009.

[10] Z. Zhang, R. Wang, X. Shen, D. Wu, J. Zhang, Z. Zhang, J. Wang, and R. Huang, "Aging-aware gate-level modeling for circuit reliability analysis," *IEEE TED*, vol. 68, no. 9, pp. 4201–4207, 2021.

[11] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Reliability-aware design to suppress aging," in *Proc. DAC*, 2016, pp. 1–6.

[12] A. B. Kahng, "New game, new goal posts: A recent history of timing closure," in *Proc. DAC*, 2015, pp. 1–6.

[13] Z. Zhang, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Eventtimer: Fast and accurate event-based dynamic timing analysis," in *Proc. DATE*, 2022.

[14] S. Guo, R. Wang, Z. Yu, P. Hao, P. Ren, Y. Wang, S. Liao, C. Huang, T. Guo, A. Chen, J. Xie, and R. Huang, "Towards reliability-aware circuit design in nanoscale finfet technology: — new-generation aging model and circuit reliability simulator," in *Proc. ICCAD*, 2017, pp. 780–785.

[15] B. Kaczer, T. Grasser, P. J. Roussel, J. Franco, R. Degraeve, L.-A. Ragnarsson, E. Simoen, G. Groeseneken, and H. Reisinger, "Origin of nbti variability in deeply scaled pfets," in *Proc. IRPS*, 2010, pp. 26–32.

[16] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. DAC*, 2013.

[17] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE TVLSI*, vol. 25, no. 10, pp. 2700–2713, 2017.

[18] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proc. ISQED*. New York, NY, USA: Association for Computing Machinery, 2015, p. 171–178.

[19] X. Jiang, X. Wang, R. Wang, B. Cheng, A. Asenov, and R. Huang, "Predictive compact modeling of random variations in finfet technology for 16/14nm node and beyond," in *Proc. IEDM*, 2015, pp. 28.3.1–28.3.4.

[20] M. Alioto, G. Scotti, and A. Trifiletti, "A novel framework to estimate the path delay variability on the back of an envelope via the fan-out-of-4 metric," *IEEE TCAS I*, vol. 64, no. 8, pp. 2073–2085, 2017.

[21] X. Jiao, M. Luo, J. H. Lin, and R. K. Gupta, "An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations," in *Proc. ICCAD*, 2017, pp. 945–950.

[22] Z. Guo, T.-W. Huang, and Y. Lin, "Heterocppr: Accelerating common path pessimism removal with heterogeneous cpu-gpu parallelism," in *Proc. ICCAD*, 2021, pp. 1–9.