Instruction-Set Extension under Process Variation and Aging Effects

Yuko Hara-Azumi†, Farshad Firouzi‡, Saman Kiamehr‡, Mehdi Tahoori‡

†*Graduate School of Information Science*
Nara Institute of Science and Technology, Japan
yuko-ha@is.naist.jp

‡*Chair of Dependable Nano-Computing*
Karlsruhe Institute of Technology, Germany
{farshad.firouzi, kiamehr, mehdi.tahoori}@kit.edu

***Abstract*—We propose a novel custom instruction (CI) selection technique for process variation and transistor aging aware instruction-set architecture synthesis. For aggressive clocking, we select CIs based on statistical static timing analysis (SSTA), which achieves efficient speedup during target lifetime while mitigating degradation of timing yield (i.e., probability of satisfying the timing). Furthermore, we consider process variation and aging on not only CIs but also basic instructions (BIs). Even if basic functional units (BFUs), e.g., ALU, get slower due to aging, only a few BIs with critical propagation delay may violate the timing, whereas the other BIs running on the same BFU can still satisfy the timing. We then introduce "customized BFUs", which execute only such aging-critical BIs. The customized BFUs, used as spare BFUs of the aging-critical BIs, can extend lifetime of the system. Combining the two approaches enables speedup as well as lifetime extension with no or negligibly small area/power overhead. Experiments demonstrate that our work outperforms conventional worst-case work (by an average speedup of about 49%) and existing SSTA-based work (16x or more lifetime extension with comparable speedup).**

I. Introduction

Increasing diversity and complexity of applications make design of embedded systems more and more time-consuming. For improving design-productivity, embedded processors have been used with application-specific extension, which is referred as *Instruction-Set Architecture (ISA) synthesis*: only the critical computations are performed on custom functional units (CFUs) using custom instructions (CIs), while the others are executed on basic functional units (BFUs) by basic instructions (BIs) [1]. Various techniques on ISA synthesis, such as [1], [2], [3], [4], [5], have been exploring speedup (i.e., cycle reduction) with less area, power, etc. Currently, besides such metrics, due to issues arising from nanoscale technologies, improving manufacturing yield and lifetime of systems also needs to be considered during ISA synthesis, especially for mass-produced embedded systems.

Yield and lifetime of systems are particularly affected by process variation and aging of transistors, respectively. The variation of transistor parameters (e.g., channel length and gate-oxide thickness) causes large variance in circuit delay, depending on the environment, the input values, etc. Transistor aging, caused mainly by negative and positive bias temperature instability (NBTI and PBTI), increases the delay of transistors over time which can finally lead to a system failure [6]. This is further complicated as aging is highly dependent on the ambient/usage environments (e.g., temperature and operating voltage) and process variation [7]. Conventionally, a sufficient time-slack (i.e., guardband) has been given based on the worst-case delay in order to guarantee that the systems always work correctly during the expected lifetime. Obviously, this approach is too conservative, especially for processors customized through ISA synthesis, since CFUs achieving good speedup are likely to have long worst-case propagation delay and contribute to (near-)critical paths. An alternative approach is to adjust (i.e., reduce) the clock frequency at runtime as the systems age [8]. However, without considering the joint effects of process variation and aging, clock adjustment may be also pessimistic. Unlike such conventional approaches, various techniques utilizing stochastic approaches like statistical statistic timing analysis (SSTA) have been studied at architectural level (e.g., [9], [10]). For more efficient system designs [11], such stochastic design approaches need to be adopted not only at hardware level, but also at software level, e.g., during ISA synthesis.

In the field of ISA synthesis, to the best of our knowledge, SSTA-based CI selection has been studied only in [4], [5]. Considering process variation, these methods preferentially select CIs which bring high speedup for *aggressive clocking* (i.e., clock shorter than the worst-case delay) as long as the required *timing yield* (i.e., probability to satisfy the timing) is satisfied at time zero (i.e., the first time it is used). Because such CIs tend to contribute to (near-)critical paths and be susceptible to aging, they may violate the timing and lead to an early system failure. Moreover, these methods do not deal with aging of BFUs, which may cause further shorter lifetime of the system. So far, there is no technique available to address both process variation and aging.

In this paper, we propose a novel CI selection technique based on SSTA, considering the joint effects of process variation and aging on both BFUs and CFUs. Our work selects CIs for aggressive clocking so that speedup can be maximized while mitigating timing yield degradation all over the target lifetime. Furthermore, we introduce customized BFUs, which can execute only BIs with critical propagation delay, and use them as spare BFUs. This can effectively extend lifetime of the system with no or negligibly small area/power overhead. The combination of our CI selection and customized-BFU approaches provides greater speedup than conventional deterministic worst-case techniques, and longer lifetime than aging-unaware techniques (such as [4], [5]) with no or negligibly small performance overhead.

The rest of this paper is organized as follows. First, Section II reviews process variation and aging effects. Next, Section III gives a motivational example, and Section IV presents our process variation and aging-aware ISA synthesis method. Then, Section V demonstrates the effectiveness of our method compared with existing methods. Finally, Section VI concludes this paper.

II. Preliminaries

Process variation and transistor aging, which are interdependent, significantly impact yield and lifetime of systems. Although aging has been conventionally treated uniformly, it actually depends on the environment, process variation, etc. In this section, we first briefly review these two phenomena, and explain how we handle them in this work.

A. BTI Model

NBTI is a source of aging which affects PMOS transistors. PBTI is a corresponding effect on NMOS transistor which has emerged as a major reliability concern with the introduction of high-κ metal-gate technologies. NBTI (PBTI) in general has two phases which are called stress and recovery. In the stress phase, when PMOS (NMOS) transistors are negatively (positively) biased, interface traps are generated at the interface of Si-dielectric, resulting in an increase in the threshold voltage of transistors. In the recovery mode, when the negative (positive) V_{gs} is removed, some of the generated traps are recovered. Since all the traps cannot be recovered during the recovery time, the transistor threshold voltage increases gradually during its lifetime. Since NBTI and PBTI are similar effects, we use the model introduced in [12] for both phenomena.

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

where T_{clk} is the clock cycle, α is the duty cycle (the ratio between the stress time and the total time), n is a fabrication process constant and β_m is the fraction parameter of the recovery. The other parameters are described in [12].

B. Process Variation

After the circuit is fabricated, the parameters of fabricated devices and interconnects are different from die to die and within a particular die. The uncertainty in devices and interconnects, which is called process variation, causes variance in the performance of circuits. According to [13], the variation of *Physical Parameters* (PPs) such as effective gate length (ΔL) from the nominal value can be represented by the following equation:

$$\begin{aligned} \Delta PP_{total} &= \Delta PP_{d2d} + \Delta PP_{wd} \\ \Delta PP_{wd} &= \Delta PP_{cor} + \Delta PP_{rand} \end{aligned} \tag{2}$$

where ΔPP_{d2d} represents die-to-die variation, ΔPP_{wd} is within-die (intra-die) variation, ΔPP_{cor} represents the spatially-correlated variation and ΔPP_{rand} denotes the independent random variations. Since process variation affects the timing of the circuits, traditional static timing analysis has to be adopted in order to consider these effects. SSTA is used as a method for timing analysis in the presence of process variation.

C. Aging-aware SSTA

BTI-induced threshold voltage shift is affected by ambient/usage environments (duty cycle, temperature, etc) as well as process parameters (such as original threshold voltage and oxide thickness) which vary by process variation [7].

$$\overrightarrow{PP} = [L, V_{th}, ...] \tag{3}$$

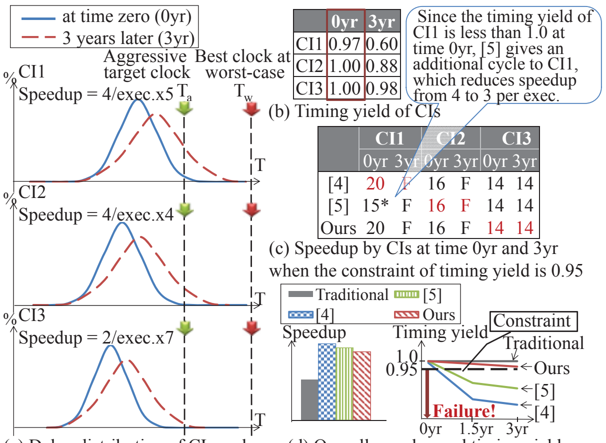
The post-aging gate delay can be expressed as follows:

$$d(\overrightarrow{PP}, t) = d_0(\overrightarrow{PP}) + \Delta d(\overrightarrow{PP}, T, \alpha, t) \tag{4}$$

where $d(\overrightarrow{PP}, t)$ is the post-aging gate delay, $d_0(\overrightarrow{PP})$ is the initial gate delay after chip fabrication, and $\Delta d(\overrightarrow{PP}, T, \alpha, t)$ is the BTI-induced delay degradation considering process variation. For the full discussion of SSTA which is used here, the reader should refer to [7].

III. Motivational Example

CI selection is an essential procedure in ISA synthesis. In conventional CI selection methods, a sufficient target clock (i.e., worst-case clock T_w) including a guardband (e.g., 20% of the clock) has been set at design time so that any instructions (both BIs and CIs) can always satisfy the timing (i.e., timing yield is 1.0) during target lifetime. Recently, SSTA-based CI selection methods [4], [5] have been proposed for an aggressive clock, T_a, which is shorter than T_w. In [4], CIs providing high speedup are selected amongst ones satisfying the designer-given constraint of timing yield, and in



	0yr	3yr
CI1	0.97	0.60
CI2	1.00	0.88
CI3	1.00	0.98

	CI1		CI2		CI3	
	0yr	3yr	0yr	3yr	0yr	3yr
[4]	20	F	16	F	14	14
[5]	15*	F	16	F	14	14
Ours	20	F	16	F	14	14

(a) Delay distribution of CIs under process variation and aging effects
(b) Timing yield of CIs
(c) Speedup by CIs at time 0yr and 3yr when the constraint of timing yield is 0.95
(d) Overall speedup and timing yield during the target lifetime (3 years)

Fig. 1. A motivational example

[5], based on the work [4], an additional cycle is given only to CIs whose timing yield is less than 1.0 in order to mitigate timing yield degradation.

Let us consider a case where for T_a, only one of three CIs (CI1, CI2, and CI3 which have speedup of 4, 4, and 2 per execution, and are accessed 5, 4, and 7 times, respectively[1]) can be selected in [4], [5]. The delay distribution and timing yield of the CIs are depicted in Figs. 1(a) and 1(b), respectively. In this example, under the yield constraint of 0.95, CI1 and CI2 are selected in [4] and [5], respectively. The speedup by these methods is summarized in Fig. 1(c). Thanks to adoption of aggressive clocking as T_a, these methods bring significant speedup compared with the conventional one (adopting T_w), as shown in the left of Fig. 1(d).

However, systems designed by aging-*unaware* methods, such as [4] and [5], may degrade quickly and end up failing within target lifetime (specified as “F” in Fig. 1(c)) because these methods do not consider that the delay distribution may transition by aging, as the dotted lines in Fig. 1(a). The more the selected CIs are critical in propagation delay (i.e., susceptible to aging), the shorter the lifetime would be. Furthermore, no consideration of process variation or aging of BFUs, even though BFUs are under stress longer and may suffer aging more than CFUs, would cause considerable overestimation of lifetime, leading to unexpectedly short lifetime. Namely, as illustrated in Fig. 1(d), aging-unaware methods will achieve large speedup over conventional methods at time zero, but may result in poor lifetime.

Considering both process variation and aging, our method maximizes speedup such that systems can be alive (i.e., the required timing yield can be satisfied) during target lifetime. In this example, our method selects CI3, which has smaller speedup but satisfies the timing yield constraint all over the target lifetime, unlike CI1 and CI2. Our method may not maximize the speedup at time zero, but provides much higher speedup than the conventional one, without unexpectedly failing as aging-unaware ones. Moreover, our method handles process variation and aging of BFUs, which can achieve further speedup and prolong lifetime with minimum area overhead, as will be explained in the next section.

[1]The total speedup of a CI is obtained by its speedup per execution multiplied by the number of times being accessed.

IV. Process Variation and Aging Aware ISA synthesis

A. Overall Approach

We propose a novel CI selection method which considers the joint effects of process variation and aging. For aggressive clocking, we select CIs to maximize speedup while mitigating degradation of timing yield during target lifetime. Unlike existing aging-unaware methods, which adopt powerful but aging-susceptible CIs, leading to high speedup at time zero but reduced lifetime, our method can achieve as high speedup as existing ones while maintaining the required timing yield all over the target lifetime.

Moreover, we present further improvement of performance and lifetime by considering process variation and aging of BFUs (e.g., ALU and multiplier), which are frequently accessed and thus suffer more aging than CFUs. Because the timing yield of BIs is different[2], we introduce an additional set of candidate CIs, "customized BFUs", which can execute only aging-critical BIs, i.e., those with particularly longer delays than the other BIs[3]. Since only a small subset of BIs is aging-critical, the area/power overhead of customized BFUs is small. By introducing customized BFUs, the scheduler can dynamically switch execution of such aging-critical BIs between the original BFUs and the customized BFUs, which can mitigate the aging and extend lifetime [14].

Combining these two approaches (i.e., process variation- and aging-aware CI selection and customized-BFUs), we aim at providing speedup and lifetime extension at the same time.

Note that in this paper, we define violation of timing yield constraints as *a system failure*. In other words, the longer the timing yield constraint is satisfied, the longer the lifetime is.

B. Cost Function of CI Selection

Here we formulate our cost function of CI selection. Table I summarizes notations and their definitions used in the formulation. Parameters of each BI/CI are obtained through our framework which will be explained in Section IV-C.

Preliminaries: The timing yield (i.e., probability to satisfy the timing) of BI_i/CI_i at time t (years later) for target clock T (ns) is given as follow, with its mean delay ($\mu_{i,t}$) and variance ($\sigma_{i,t}$) at time t:

$$y_{i,t}(T) = \frac{1}{2} + \frac{1}{2} erf(\frac{T - \mu_{i,t}}{\sigma_{i,t}\sqrt{2}}) \tag{5}$$

$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z \mathrm{e}^{-z^2} dz \tag{6}$$

where Equation (6) represents the Gauss error function.

Constraints: CI_is need to satisfy Constraint (7) during target lifetime to become selectable, whereas BI_is need the minimum number ($b_i \geq 0$) of customized BFU_i such that Constraint (8) can be satisfied during target lifetime. For example, $b_i = 0$ represents no customized BFUs are necessary. If $b_i = 1$, it is considered the aging of the full BFU_i and the added customized-BFU_i will be halved by using them interchangeably. In other words, the lifetime can be doubled. Actually, this is a conservative prediction since aging-aware scheduling can extend lifetime efficiently (2.3-3.9x) [15], i.e., the lifetime would become more than double. Note that the appropriate b_i can be automatically determined by Constraint (8).

[2]As BFUs age, only a few BIs passing through long paths may violate the timing, whereas the other BIs can still satisfy the timing.

[3]It is also applicable to CFUs, but the effect may be smaller since only one or a few similar CIs are performed on the same CFUs.

TABLE I Definition of Notations

T	Target clock period
$Const_{yield}$	The designer-given constraint on timing yield
$Const_{area}$	The designer-given constraint on area
$Const_{power}$	The designer-given constraint on power
$\mathfrak{B}$	A set of BIs
$\mathfrak{C}$	A set of CIs
$\mathfrak{S}_i$	A sequence of assembly code by BIs which is equivalent to the behavior of CI_i
$\mathfrak{D}$	A set of critical BIs whose timing yield may become lower than $Const_{yield}$ during target lifetime
$y_{i,t}$	Timing yield of BFU_i/CFU_i at time t
$p^s_{i,t}$	Static (i.e., leakage) power of BFU_i/CFU_i per unit at time t
$p^d_{i,t}$	Dynamic power of BFU_i/CFU_i per access at time t
$p_{i,t}$	Total power of BFU_i/CFU_i at time t
l_i	Latency of BI_i/CI_i
a_i	Area of BFU_i/CFU_i
b_i	The minimum number of additional customized BFU_i
c_i	1 if CI_i is selected, otherwise 0.
s_i	Speedup by CI_i (per access)
e_i	The number of time that CI_i is executed

$$y_{i,t}(T) \geq const_{yield}, CI_i \in \mathfrak{C}, \forall t \tag{7}$$

$$y_{i,t/(b_i+1)}(T) \geq const_{yield}, \forall BI_i \in \mathfrak{B}, \forall t \tag{8}$$

CIs also need to satisfy area and power constraints:

$$\sum_{CI_i} a_i \cdot c_i + \sum_{BI_j \in \mathfrak{D}} a_j \cdot b_j \leq const_{area} \tag{9}$$

$$\sum_{CI_i} p_{i,t} \cdot c_i \leq const_{power}, \forall t \tag{10}$$

where the total power overhead of CFU_i at time t ($p_{i,t}$) can be obtained as the summation of its static and dynamic power consumption, by Equation (11). Note that as shown in Constraint (10), employing customized BFUs does not introduce power overhead since the scheduler dynamically switches execution of aging-critical BIs between the original BFUs and the customized BFUs, and inactive original/customized BFUs are power-gated. To obtain the dynamic power overhead of CFU_i, the total dynamic power of $BI_j \in S_i$ should be subtracted as expressed in the second term of Equation (11).

$$p_{i,t} = p^s_{i,t} + (p^d_{i,t} - \sum_{BI_j \in \mathfrak{S}_i} p^d_{j,t}) \cdot e_{i,t} \tag{11}$$

Power consumption may be also affected by process variation and aging, which can be handled similarly as Constraint (7) with small extension. We assume that the designers do not mind the area and power overhead by the introduced $CFUs$ as long as Constraints (9) and (10) are satisfied.

Speedup: Speedup provided by CI_i is obtained as performance improvement (i.e., latency reduction against the execution by BIs only) per execution multiplied by the number of times being executed.

$$s_i = (\sum_{BI_j \in \mathfrak{S}_i} l_j - l_i) \cdot e_i \tag{12}$$

Cost function: Our objective is to maximize speedup by selecting CIs which satisfy Constraints (7)-(10):

$$Max : \sum_{CI_i} s_i \cdot c_i$$

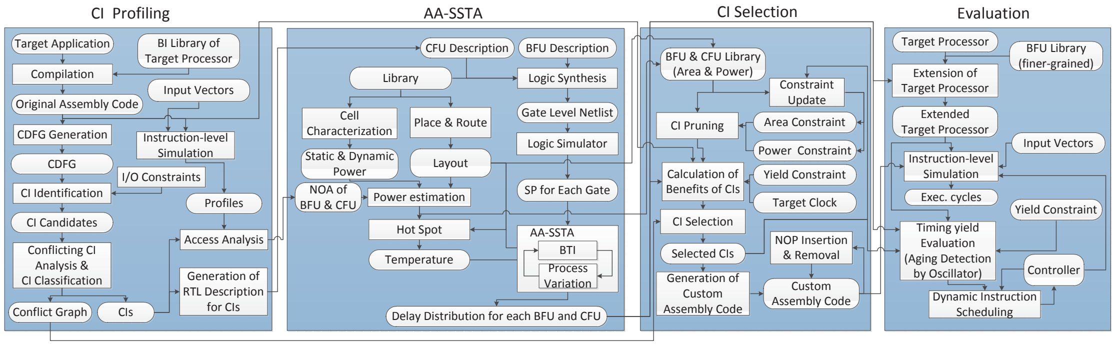


Fig. 2. The overall proposed flow for CI selection by considering the joint effect of the process variation and BTI

C. *The Overview of Our Framework*

The overall flow of our framework is depicted in Fig. 2, where rectangles and circles represent processes and inputs/outputs, respectively. The framework consists of four stages: We first obtain execution profiles of a target application and extract candidate CIs (at CI Profiling). Then, we analyze delay distribution of each CI under process variation and aging effects (at AA-SSTA). Based on the profiles and delay distribution, our CI selection is performed (at CI Selection). Finally, the speedup and timing yield of the target application customized with CIs is evaluated (at Evaluation). The following gives more detailed explanations about each stage. Note that our framework is not restricted to the tools, models, and etc. used below.

CI Profiling: First, we compile a target application to generate an original assembly code for the target processor (in our experiments, MIPS processor which has a multiplier, a divider, and an ALU as BFUs). The assembly code is simulated through SimpleScalar to obtain instruction-level profiles (e.g., the number of accesses to each BI), and used to build a control/data flow graph (CDFG), from which CI candidates are extracted under constraints on inputs/outputs (corresponding to read/write accesses to the register file) [16]. Then, conflicting CIs (i.e., CIs which are not applicable at the same time) are investigated and the similar CIs are classified into the same CI template in order to prune exploration space effectively. The potential accesses of CIs are analyzed from the profiles of the original assembly code. Finally, the RTL descriptions of CFUs which can perform their corresponding CIs (e.g., Verilog-HDL) are generated.

AA-SSTA: CFUs and BFUs are both synthesized by Synopsys Design compiler and mapped to Nangate 45nm cell library [17]. Gate-level netlists, extracted from the logic synthesis tool, are given to a logic simulator to obtain the Signal Probabilities (SPs) and switching activity of internal nodes (gates as well as transistors inside each gate). Switching activity factors are used to obtain dynamic power, and SPs of each node (the probability of being logic '1') are used to obtain effective duty cycles (impacts of the BTI) of each transistor [18]. Besides, the extracted netlists are placed using Cadence SoC Encounter to obtain layout information. To estimate the dynamic and static power consumption of the CIs, each cell of the cell library is accurately characterized by HSPICE simulation. Next, gate-level netlists, layout information (e.g., parasitic capacitance), and profiling information (e.g., the number of accesses to each BI and CI) are analyzed for assessing the CI power consumption. Afterwards, extracted floorplans of the CIs and power profiles are given to HotSpot [19] to compute the corresponding temperature profile of the CIs. To model the spatial correlations of intra-die process variation, the extracted CI layouts are partitioned into several rectangular grids where the number of grids represents the die area. For our experiments, total process variation is set such that $3\sigma/\mu = 20\%$. Considering the temperature profiles, layout information, duty cycle of each internal transistors, and assuming 15% BTI-induced delay degradation in 3 years, post-aging delay distribution of each CI is calculated by using a method presented in Section II-C.

CI Selection: Based on the parameters unique to each CI (i.e., area, power, and delay distribution), CIs which can bring high speedup while satisfying the timing yield constraint during target lifetime are selected. Selected CIs also should satisfy area and power constraints. CI selection is performed in a greedy manner for our experiments[4]. The assembly code is customized with the selected CIs stepwisely. Finally, minimum NOP instructions are inserted into the generated custom assembly code for preventing hazards if necessary.

Evaluation: The assembly code customized with the selected CIs is evaluated in terms of performance and timing yield. Only when necessary, we extend the target processor with the minimum number of customized BFUs such that the system can work for target clock and lifetime. In our experiments, we evaluate timing yield based on the delay distribution obtained from the AA-SSTA stage, considering the effects of dynamic instruction scheduling. During actual processor operation, processor's aging-monitors [20] and sensors [21] will detect the system aging, based on which the scheduler will dynamically switch execution of aging-critical BIs between the original and customized BFUs, as explained in Section IV-A. Performance is evaluated through simulation for given input vectors. Note that switching execution between the original and customized BFUs does not affect the performance since the controller will be updated during idle time.

V. Experiments

A. *Experimental Setup*

By our framework in Fig. 2, we assessed the effectiveness of our method for the following six benchmarks, which are often used in the literature on ISA synthesis: *adpcm*, *aes*, *chenidct*[5], *gsm*, and *sha* from [22], and *wavelet* from [23]. CIs are extracted under constraints of two inputs and one output. Test vectors built in each benchmark are utilized.

In Section V-B, we evaluate speedup (i.e., performance improvement over the original assembly code)[6] and timing yield by the

[4]Sophisticated algorithms, such as [1], [2], [3], can be also used.

[5]*chenidct* is a part of JPEG in CHStone [22].

[6]0% means that the performance is exactly the same as the execution of the original assembly code.

following four CI selection methods:

- A traditional deterministic worst-case method (**TRADITIONAL**), which is applicable only when clock T is long enough for every BI/CI to always complete successfully. For this, a 20% guardband was given based on the delay distribution at time zero (i.e., 0 year). The delay degradation of CIs was on average 19.5% and up to 21.8%. For each benchmark, the minimum clock such that every BI/CI can complete with timing yield of 1.0 is applied.
- An existing SSTA-based method [4] (**DATE11**), which selects CIs with high speedup as long as they satisfy the timing yield constraint. Aging of BIs/CIs or process variation of BIs are not considered.
- Another existing SSTA-based method [5] (**DATE12**), which extends DATE11 so that an additional cycle is given to CIs whose timing yield is less than 1.0 in order to mitigate timing yield degradation. Note that the additional cycles can be given only at time zero (i.e., 0 year). Like DATE11, aging of BIs/CIs or process variation of BIs are not considered.
- Our SSTA-based method (**PROPOSED**), which considers process variation and aging of BIs/CIs. CIs with high speedup amongst ones satisfying the timing yield constraint during target lifetime are selected.

For target lifetime of three years, the above four methods are evaluated varying the parameters as follows: area constraint (2x, 3x, 4x, and ∞x the total area of BFUs), power constraint (2x, 3x, 4x, and ∞x the total power of BFUs to execute original assembly code consisting of only BIs), timing yield constraint (0.95, 0.90, and 0.80), and target clock (6.5ns, 7.5ns, and 8.5ns, for all of which BIs can satisfy the timing yield constraint during target lifetime, meaning that timing yield violation is only by CIs).

Furthermore, in Section V-C, we evaluate effects of our second contribution for lifetime extension. One instance of customized BFU_i is allowed only when BIs running on BFU_i are critical in terms of timing yield.

B. Experimental Results: CI Selection

Results of speedup and timing yield are shown in Figs. 3 and 4, respectively, for target clock of 6.5ns[7], timing yield constraint of 0.95, and area/power constraints of ∞x. As shown in Fig. 3, speedup by TRADITIONAL is small for every benchmark, i.e., up to by about 26% in *chenidct*. Conventional CI selection techniques adopting deterministic worst-case approaches will face the clock wall quickly as well as TRADITIONAL. The three SSTA-based methods (i.e., DATE11, DATE12 and PROPOSED) achieve significant speedup compared with TRADITIONAL. Speedup by DATE11 is the highest for most benchmarks, i.e., up to by 65.8% and on average by about 50% (slightly higher than DATE12, which gives additional cycles to some CIs). Although PROPOSED is not the best, the overhead against DATE11 is only 1-2%, other than for *gsm*, where interestingly, PROPOSED outperforms DATE11 and DATE12. This is because CI conflict (i.e., a CI selected already (e.g., CI1) may prevent from selecting other CIs which are a subset of CI1) occurred in DATE11 and DATE12, but not in PROPOSED since CI1 was unselectable by PROPOSED due to timing yield violation. We can see from Fig. 3 that in spite of penalty by the constraint of timing yield during target lifetime (i.e., Constraint (7)),

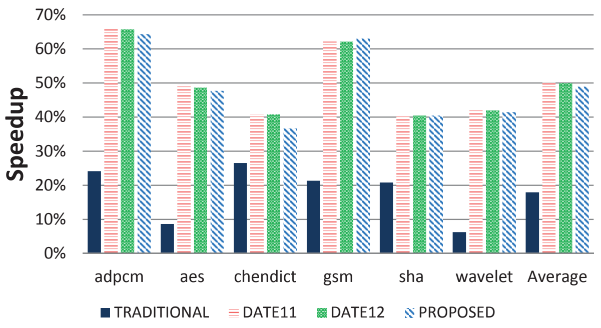


Fig. 3. Speedup (timing yield constraint 0.95 and target clock 6.5ns)

PROPOSED can provide comparable speedup as existing SSTA-based methods.

As illustrated in Fig. 4, while timing yield of DATE11 and DATE12 is considerably low, that of PROPOSED is kept above 0.95 for three years: DATE11 and DATE12 violate the constraint on average at time 0.30 year and 0.41 year, which means that PROPOSED achieves on average 10x and 7.3x lifetime extension, respectively. In *adpcm* and *aes*, DATE11 has particularly low timing yield since it looks at only how much speedup each CI can bring and selects CIs with high speedup as long as they satisfy the constraint of timing yield only at time zero, whereas timing yield degradation of DATE12 is slower thanks to the additional cycles given at time zero (although it also ends up violating the constraint on average at time 0.41 year).

Although the degree of lifetime and speedup varies depending on the parameters of constraints, similar results as shown in Figs. 3 and 4 were also observed for different sets of parameters (their results are omitted due to space limitation). As expected, speedup is smaller under stricter constraints on area, power, and timing yield for every benchmark and every method because less CIs can satisfy the constraints. Comparing effects on speedup by target clocks and those by timing yield constraints, the former was bigger (on average about 35% difference between 6.5ns and 8.5ns) than the latter (on average only 1-2% difference between 0.95 and 0.80). More aggressive clocking lets the three SSTA-based methods bring more speedup against TRADITIONAL although less CIs may be selectable. In other words, in spite of having more selectable CIs, for more relaxed target clock (such as 8.5ns), difference in speedup between TRADITIONAL and the three SSTA-based methods is smaller. On the contrary, in our experimental environment, ALU is more critical than most CFUs, and CIs running on such CFUs satisfy the timing yield constraint for the three target clocks, which caused such small differences on speedup[8].

Holistically evaluating our method (i.e., PROPOSED), it can bring comparable speedup while providing required timing yield for desired lifetime (i.e., about 4x-8x lifetime extension against aging-unaware methods, at the cost of only 1-2% performance degradation). It should be noted that our method achieves such significant benefits in spite of its straightforwardness (i.e., greediness).

C. Experimental Results: Customized BFUs

For further aggressive target clocking, even if CI selection could be done successfully in terms of timing yield, violation of timing

[7]Because TRADITIONAL cannot accept such aggressive clock, its results are for achievable minimum clock in each benchmark.

[8]Effects of constraints and clock totally depend on the parameters of the target device.

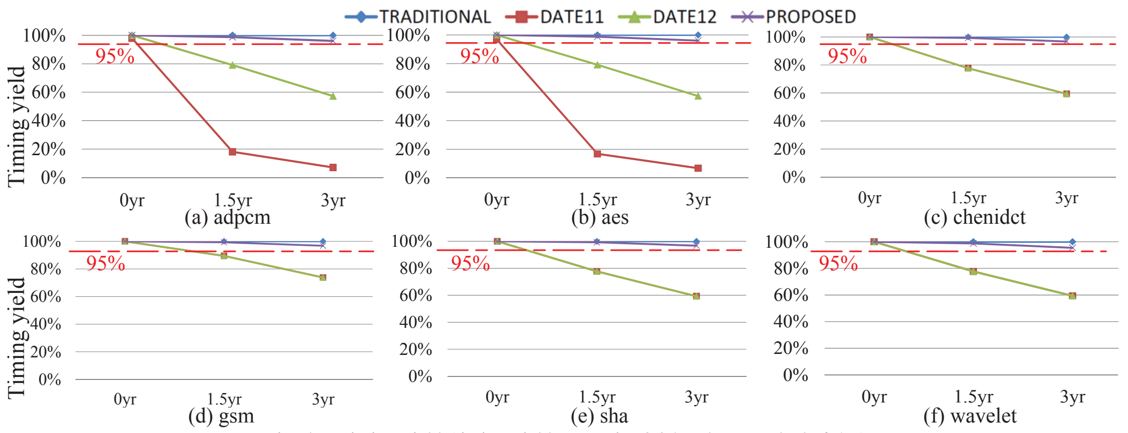


Fig. 4. Timing yield (timing yield constraint 0.95 and target clock 6.5ns)

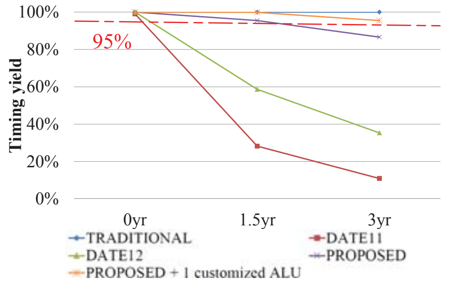


Fig. 5. Lifetime extension by adding one customized ALU (*adpcm*)

yield constraints may be unavoidable even by PROPOSED due to degradation of BFUs (e.g., ALU and multiplier). In our experiments, ALU becomes critical to satisfy the timing yield constraint. Obviously, techniques which are unaware of aging of BFUs (e.g., DATE11 and DATE12) have no solution to work for such aggressive clocking. Investigating BIs running on ALU, we then found that add was aging-critical. We thus introduced a customized ALU for *add* only (i.e., *add*-ALU), whose cost (area and power) is only about 8% of the full ALU. Using the full ALU and *add*-ALU interchangeably (i.e., dynamic instruction scheduling), the lifetime can be further extended.

Introducing one *add*-ALU could achieve 2x (more than 16x) lifetime extension[9] compared with PROPOSED (DATE11/DATE12[10]) at target clock of 5.8ns, 5.7ns, and 5.6ns, for timing yield constraint of 0.95, 0.90, and 0.80, respectively. Due to space limitation, Fig. 5 depicts lifetime comparison of the four methods in Section V-B and "PROPOSED + 1 *add*-ALU" in *adpcm* for target clock of 5.8ns, timing yield constraint of 0.95 under ∞x area/power constraints. Similar results were observed for the other sets of parameters and benchmarks. There was no performance penalty compared with PROPOSED, except for only when area/power constraints are 2x (less than 1%). These results demonstrate that by combining our two approaches, we can achieve speedup and further lifetime extension at the same time, with no or negligibly small performance penalty.

VI. Conclusion

In this paper, we proposed a novel custom instruction (CI) selection technique for process variation- and aging- aware instruction-set architecture synthesis. Our method consists of two approaches: (1) SSTA-based CI selection which maximizes speedup while satisfying the timing constraint during target lifetime, and (2) lifetime extension by extracting aging-susceptible basic instructions (BIs) and introducing their specific functional units. Integrating these two approaches, significant speedup and lifetime extension can be efficiently achieved at the same time. Experimental results demonstrated the effectiveness of our work against conventional deterministic worst-case work (more than 49% speedup, on average) and existing SSTA-based work (more than 16x lifetime extension with comparable speedup).

[9]As explained in Section IV-B, this is a conservative prediction.

[10]They violated the timing yield constraint at time 0.18 year or earlier.

References

[1] J.-E. Lee, K. Choi, and N. D. Dutt, "Instruction set synthesis with efficient instruction encoding for configurable processors," *ACM Trans. on Reconfigurable Technology and Systems*, vol. 12, no. 1, 2007.

[2] R. Koenig *et al.*, "KAHRISMA: A novel hypermorphic reconfigurable instruction-set multi-grained-array architecture," in *Proc. DATE*, 2010.

[3] C. Galuzzi and K. Bertels, "The instruction-set extension problem: A survey," *ACM Trans. on Reconfigurable Technology and Systems*, vol. 4, no. 2, 2011.

[4] M. Kamal, A. Afzali-Kusha, and M. Pedram, "Timing variation-aware custom instruction extension technique," in *Proc. DATE*, 2011.

[5] M. Kamal *et al.*, "An architecture-level approach for mitigating the impact of process variations on extensible processors," in *Proc. DATE*, 2012.

[6] K. Kang *et al.*, "Estimation of statistical variation in temporal NBTI degradation and its impact on lifetime circuit," in *Proc. ICCAD*, 2007.

[7] Y. Lu *et al.*, "Statistical reliability analysis under process variation and aging effects," in *Proc. DAC*, 2009.

[8] A. Masrur *et al.*, "Schedulability Analysis for Processors with Aging-Aware Autonomic Frequency Scaling," in *Proc. RTCSA*, 2012.

[9] J. Abella, X. Vera, and A. Gonzalez, "Penelope: The NBTI-Aware Processor," in *Proc. MICRO*, 2007.

[10] S. Jin *et al.*, "Statistical lifetime reliability optimization considering joint effect of process variation and aging," *Integration VLSI journal*, vol. 26, no. 4, 2011.

[11] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE*, vol. 100 Centennial-Issue, 2012.

[12] W. Wang *et al.*, "The impact of NBTI effect on combinational circuit: modeling, simulation, and analysis," *IEEE Trans. on VLSI Systems*, vol. 18, no. 2, 2010.

[13] J. Xiong, V. Zolotov, and L. He, "Robust extraction of spatial correlation," *IEEE Trans. on CAD*, vol. 26, no. 4, 2007.

[14] F. Oboril and M. Tahoori, "ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level," in *Proc. DSN*, 2012.

[15] F. Oboril *et al.*, "Reducing NBTI-induced Processor Wearout by Exploiting the Timing Slack of Instructions," in *Proc. CODES+ISSS*, 2012.

[16] M. Baleani *et al.*, "HW/SW partitioning and code generation of embedded control applications on a reconfigurable architecture platform," in *Proc. CODES*, 2002.

[17] "Nangate," http://www.nangate.com/.

[18] S. Kiamehr, F. Firouzi, and M. Tahoori, "Input and transistor reordering for NBTI and HCI reduction in complex CMOS gates," in *Proc. GLSVLSI*, 2012.

[19] W. Huang *et al.*, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," in *IEEE Trans. on VLSI Systems*, vol. 14, no. 5, 2006.

[20] B. Zandian *et al.*, "WearMon: Reliability monitoring using adaptive critical path testing," in *Proc. DSN*, 2010.

[21] , A. Amouri and M. Tahoori, "A Low-Cost Sensor for Aging and Late Transitions Detection in Modern FPGAs," in *Proc. FPL*, 2011.

[22] Y. Hara *et al.*, "Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis," *Journal of Information Processing*, vol. 17, no. 12, 2009.

[23] "Nangate," http://www.ics.uci.edu/ẽxpress/index.htm.