An Architecture-Level Approach for Mitigating the Impact of Process Variations on Extensible Processors

Mehdi Kamal[1], Ali Afzali-Kusha[1], Saeed Safari[1], Massoud Pedram[2]
[1]School of Electrical and Computer Engineering, University of Tehran
[2]Department of Electrical Engineering-Systems, University of Southern California
{mehdikamal, afzali, saeed}@ut.ac.ir, pedram@usc.edu

***Abstract*— In this paper, we present an architecture-level approach to mitigate the impact of process variations on extended instruction set architectures (ISAs). The proposed architecture adds one extra cycle to execute custom instructions (CIs) that violate the maximum allowed propagation delay due to the process variations. Using this method, the parametric yield of manufactured chips will greatly improve. The cost is an increase in the cycle latency of some of the CIs, and hence, a slight performance degradation for the extensible processor architectures. To minimize the performance penalty of the proposed approach, we introduce a new merit function for selecting the CIs during the selection phase of the ISA extension design flow. To evaluate the efficacy of the new selection method, we compare the extended ISAs obtained by this method with those selected based on the worst-case delay. Simulation results reveal that a speedup improvement of about 18% may be obtained by the proposed selection method. Also, by using the proposed merit function, the proposed architecture can improve the speedup about 20.7%.**

I. INTRODUCTION

Embedded processors are used in many platforms such as cell phones, digital cameras, network routers, and monitoring devices. The major issues in embedded systems are computational speed, power consumption, and system flexibility. There are two approaches to implement a digital embedded system. The first is Application Specific Integrated Circuits (ASIC) where both high speed and low power dissipation may be achieved with a penalty of higher design cost and lower flexibility. The other design approach is General Purpose Processor (GPP) where the speed is lower, the power consumption is higher, and the flexibility is higher compared to the ASIC approach. The third method for realizing embedded systems is Application Specific Instruction-set Processors (ASIP). The speed, power, and flexibility of this approach is between the ASIC and GPP solutions[1].

In the ASIP approach, the GPP instruction set is extended through ASIC design based on the application specific features. The augmented instructions are determined such that the desired speed, power, and cost requirements are fulfilled. The main idea behind using ASIP is to run the hotspot parts of an application using custom instructions (CIs) and the other parts of the application on the ALU of the processor[1][2]. The extended ISA is contained in Custom Functional Unit (CFU) which executes instructions in parallel with the ALU. The CIs enhance the processor speed by reducing the number of instructions as well as the accesses to the cache and register file. In addition, they lower the power consumption.

The ISA extension flow starts by extracting the Data Flow Graph (DFG) of the hotspot parts (which are the parts to which most of the execution time belongs) of the application. Next, all the subgraphs (CIs) that meet the predefined constraints, such as, convexity, propagation delay, and I/O, are extracted from DFG of hotspot parts. The last phase of the ISA extension flow is the selection phase. In the selection phase, the best CIs are chosen from the candidate set. The CIs are selected based on the merit value which in most cases is the achieved speedup when the CIs are used. Figure 1 depicts the design flow.

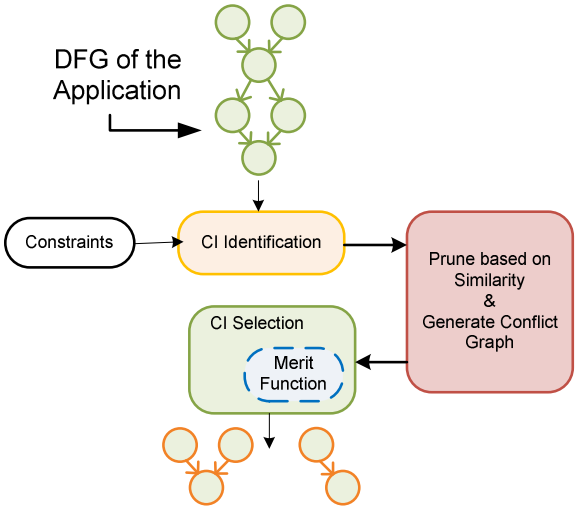


Figure 1. ISA extension design flow.

In conventional (deterministic) design flows, the nominal gate parameters are used during the design of digital systems. In sub-100nm technologies, however, complexities in the manufacturing of the transistors with very small sizes have caused significant variations in transistor parameters (such as threshold voltage and effective channel length), which in turn has led to uncertainties in the delay and power consumption of the circuits/gates [3]. This uncertainties cause the parameters in the fabricated chip to be different (less or more) from those specified in design time. In the deterministic ISA extension approach, the worst-case or nominal delay of the primitives (e.g., AND, ADD, SHIFT) are used to extract the latency of the identified CIs during the design time. Due to the die-to-die or within-die variations, the CI latency would be different from the one considered in the design time. If we design based on the worst-case delay, the process variation may not have

978-3-9810801-8-6/DATE12/©2012 EDAA

impact on the critical paths of the design. Using this approach, we however lose the speedup of the extended ISA [4]. On the other hand, when the extended ISA is generated based on the nominal delay, the performance yield of the extended ISA would be much less than one. This originates from the fact that the CI propagation delay becomes greater than the clock period, and hence, the CI could not complete its execution in one clock cycle. Hence, due to delay uncertainty, if the performance yield of the extended ISA is smaller than one, some of the manufactured extended ISA chips will not work properly.

In this paper, we propose an architecture which gives two cycles to the CIs which cannot complete their execution in one cycle due to the delay variation. In this technique, after fetching such a CI, the pipeline will be stalled by forcing the processor controller to allow some CIs to execute in two clock cycles. This is performed using the extra hardware added for the extended ISA. Due to these extra cycles, the speedup achieved through using the extended ISA is lowered. To minimize the impact of the extra cycles on the overall speedup, the selection of the CIs should be performed by considering this reduction as a selection parameter. This is performed by using a merit function which includes this parameter. The rest of the paper is organized as follows. Section II briefly reviews the related works while the proposed architecture and the design flow idea are described in Section III. The experimental setup and the results are discussed in Section IV. Finally, Section V concludes the paper.

II. Related Work

In this section, we review some of the works related to the techniques which have been proposed to lessen the impacts of the process variation on the processors as well as some prior works on ISA extension. In [5], two compile time techniques were proposed to handle non-uniform latency of different integer functional units (IFUs) in VLIW processors. In the proposed approach, the highly affected IFUs are turned off whenever the processor does not require them for running an application and turned on when required. An architectural technique (Trifecta) to mitigate the timing variations in critical pipeline stages is proposed in [6]. In the proposed technique, the inputs that make the critical path delay exceed the one-cycle delay are detected (circuit level speculation) and let the path to complete its operation in two-cycles.

In [7], a technique to tackle the performance reduction in the presence of process variation for out-of-order processors is presented. In this technique, the instructions, based on their dependency on each other, categorized in two groups. The instruction which no instruction depends on its result, are executed on a long-latency unit. On the other hand, if another instruction depends on its result, this instruction should be executed in a short-latency unit. In [8], two fine-grained post-fabrication techniques are proposed to mitigate the timing fluctuation. The voltage interpolation and variable pipeline latency are the two methods proposed in this paper. In [9], first, the process variation impacts on the propagation delay of the pipeline stages are investigated. Then, an architectural technique to decrease the impacts of the timing fluctuation on the performance of pipelined processors based on the cycle time stealing is proposed. In addition to the time borrowing techniques discussed in [7] and [8], a method which controls the clock speed in multi-issue processors is suggested in [10]. The method categorized the runtime of the program in two different phases, Low-ILP (Instruction Level Parallelism) phases, and High-ILP phases. In each phase, the impact of the process variation is lowered by changing the clock speed.

In the field of ISA extension, many techniques have been proposed. These include the techniques for increasing the quality of the selected CIs as well as decreasing the runtime of the algorithms for the CI identification (see, e.g., [1]). The proposed methods were all based on the deterministic approach. In [4], a statistical design flow for the ISA extension was proposed. The proposed design flow used the statistical approach in the both identification and selection phases where the performance yield was defined as a new constraint in the ISA extension flow.

The prior work focused on the processors multi-cycle pipeline stages and also out-of-order processors. In embedded applications, for the extensible processors, the in-order architecture is the more common. In this paper, we assume that the baseline processor is an in-order processor. The proposed architecture and the design method may be used for out-of-order processors as well. In the next section, we describe our proposed architecture for reducing the process variation impact. Also, the merit function which is used in the selection phase is discussed.

III. Proposed Architecture

In this section, we describe the proposed technique which is based on the fact that the worst-case delays of the CIs are smaller than two clock periods. Thus, by adding one extra cycle to CIs, we are able to complete the execution without worrying about the process variations. Next we describe the technique in more details.

In the presence of the process variations, the path delays are defined by PDFs (probability density functions). Due to the statistical nature of the path delays, the timing-critical circuit outputs in some manufactured chips will meet the predefined maximum propagation delay threshold while the others will violate the delay. Let us define the timing-critical outputs as the outputs that may violate the maximum delay threshold. At the design time, using the process variation modeling and also the statistical static timing analysis (SSTA), the critical outputs of each CI can be identified. Hence, to check whether or not these critical outputs indeed violate the clock period, they must be tested after the chip is manufactured. This test is performed during the chip test phase where the CI propagation delay is evaluated to determine the CIs whose worst-case path delays are greater than one clock period.

More precisely, if the process variation modeling and SSTA are used, the CIs may be selected based on their worst-case delay ($\mu+3\sigma$). In this situation where no selected CI violates the maximum clock period, and hence, the proposed architecture is not needed to be used in the extensible processor. This approach, however, may not lead to a considerable speed enhancement due to very limited number of CIs which may be selected. In the approach suggested in this work, the critical paths are specified in the design time and

only these outputs are tested in the test phase. Another use of this modeling is in the merit function that is used to minimize the speedup reduction due to the extra clock cycle added to the CIs whose worst-case path delay plus some margin for flip-flop clock to output delay and setup time can be larger than the specified clock period. Each CI may have several output bits (i.e., 32-bit). Among them, only those bits that are on the critical paths should be considered in the test phase. So, the set of critical outputs of a CI is defined as

$$\{\forall Outputs \in CI, \exists k_i > 0 \Rightarrow \mu_i + k_i\sigma_i > MPD\} \qquad (1)$$

where μ_i and σ_i are mean and sigma values of the i^{th} output, MPD denotes *Maximum Propagation Delay*, and k_i is a positive real number.

Now, we explain our proposed architecture that adds the additional clock cycles for the CIs that require them. In addition, we describe the merit function that is used to lower the speedup degradation due to addition of the extra clock cycle. Figure 2 shows the proposed architecture when it is used with a five-stage pipelined processor. The proposed architecture consists of two main parts, which are the controller and the checker. The checker is used to detect the propagation delay violation of those CI outputs that have been reported as the timing-critical outputs during the design time. These outputs are tested by injecting proper test vectors during the test time. If an output violates the maximum propagation delay, the CI corresponding to this output is added to a look-up table (LUT) in the controller part.

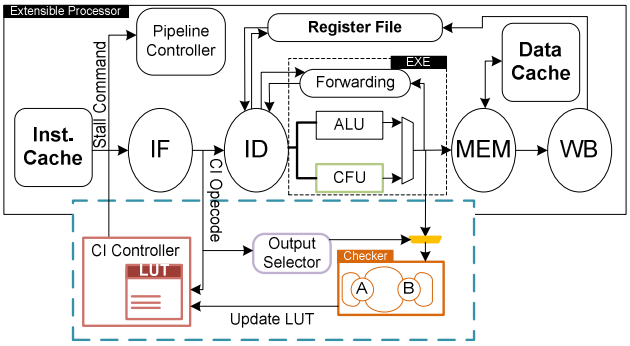


Figure 2. A five stage pipelined extensible processor enhaced with proposed architecture.

During the test time, the test vectors are injected into the CFU. The test vectors, which are extracted using the test delay method [11], are applied through the processor instruction issue unit. To test the CI outputs, the specific test vector should be written to the register file. We need *m* instructions to load the test vector where *m* is equal to the number of input ports of the CI that is under the test. After loading the test vector, by fetching a CI, the corresponding test vector is applied to the CI. Finally, the output of the CI must be checked. To check the CI output, a new instruction that compares the output of the execution stage with a known (expected) value of the specified register is defined. If the CI output is not equal to this value, the op-code of the CI is added to the LUT inside the controller. If the CI under the test contains more than one output port, each output must be checked separately. If any of these outputs fails to match the expected value, the CI's op-code will be added to the LUT.

During the test time, based on the op-codes of the CIs which need two cycles to complete their execution phase, the contents of the LUT are updated. Therefore, in the application runtime, when each instruction is fetched, the controller unit checks the instruction op-code in the LUT. For any op-code present in the LUT, the controller sends a signal to the processor controller to stall the pipeline for one clock cycle, thereby allowing the CI instruction to complete its execution phase without any timing violations. During the extra cycle, we must ensure that the inputs of the CFU are preserved. Therefore, the pipeline controller must also freeze values of the input registers of the CFU during the extra cycle.

The proposed architecture alleviates the problem of manufacturing-induced process variations (e.g., V_{th} or L variability effects). It, however, adds one extra cycle degrading the overall speedup of the extensible processor. To lower this speed deterioration, we suggest a new merit function for the selection phase. In the conventional approaches, the merit function is defined based on the speedup of the CIs. The speedup of a CI is defined as

$$CI_i.S = \#Iteration \times \left(\#CI_i.SW - IO_i.Penalty - ceil\left(\frac{CI_i.CriticalPathDelay}{Clock\ Period}\right)\right) \qquad (2)$$

where $CI_i.S$ is the *speedup* of the i^{th} CI (CI_i) which is equal to the number of clock cycles saved at the runtime of the application when the CI_i is used. Here *#Iteration* denotes the execution frequency of the basic block to which CI_i belongs, $\#CI_i.SW$ denotes the number of clocks that the CI needs to run by the base processor, and, $IO_i.Penalty$ is the number of extra accesses to the register file for reading data from or writing data in (when the number of the CI I/O ports is more than the number of the register file read/write ports). The second term in the above equation is the number of clocks needed for executing CI_i on the CFU (we assume CIs can be multi-cycle as well as single cycle instructions.) In this fraction, $CI_i.CriticalPathDelay$ denotes the propagation delay of the CI critical path and *Clock Period* is the desired clock period for the extended processor. For the sake of simplicity (but without loss of generality), we assume CIs are selected when their nominal delay is less than the *Clock Period*, and hence, the $ceil\left(\frac{CI_i.CriticalPathDelay}{Clock\ Period}\right)$ term is equal to one. Also, we assume, the CIs I/O ports are equal to the R/W ports of the register file, and thus, $IO_i.Penalty$ is equal to zero. Hence, the speedup of a CI may be calculated as

$$CI_i.S = \#Iteration \times (\#CI_i.SW - \#CI_i.Clock) \qquad (3)$$

where $\#CI_i.Clock$ is the number of clocks that the CI needs to complete its execution. The value of $\#CI_i.Clock$ is one unless, due to the process variation, $CI_i.CriticalPathDelay$ is greater than the *MPD*, in which case it will be set to two. When

#CI.SW of a CI is two and the CI executes in two cycles, the *CI.S* becomes zero indicating that there will not be any runtime speedup by using this CI. As the number of these CIs increases, the overall speedup of the extensible processor reduces. When *#CI.SW* is larger than two, there is still some speedup even if the CI requires one extra cycle. Let us define the parameter *CI.SW_Impact* as

$$CI_i.SW\_Impact = \frac{(\#CI_i.SW - 2)}{(\#CI_i.SW - 1)} \quad (4)$$

The value of this parameter changes between 0 (when *#CI.SW* is two) and 1 (when *#CI.SW* is infinity). Larger values mean smaller undesirable impacts of the extra cycle. Hence, due to the delay variation, it is better to use the CIs with the *SW_Impact* values close to one. To increase the chance of selecting these CIs, we propose a merit function defined as follows

```
If (CI_i.PY = 1)
    CI_i.Merit = CI_i.S
else
    If (CI_i.SW_Impact = 0)
        CI_i.Merit = 0;
    else
        CI_i.Merit = CI_i.S × (1+ α × CI_i.SW_Impact)
```
(5)

where $CI_i.PY$ is the performance yield of the i^{th} CI and α is the weight of $CI_i.SW\_Impact$ in the merit value. The merit value is equal to the conventional merit value when the performance yield of the CI is equal to 1. When the performance yield is less than 1, if the $CI_i.SW\_Impact$ is equal to zero, the merit value will become zero; else, the conventional merit value will be scaled by $(1 + \alpha \times CI_i.SW\_Impact)$. Note that, in this approach, we need to model the process variation in the design cycle to extract the performance yield of the CIs.

IV. Results and Discussion

A. Experimental Setup

The extracting ISA design flow was implemented by the C# language. To assess the efficacy of the design flow, the custom instructions of a bunch of benchmarks were extracted. The selected benchmarks included *IP-Sec* and *MD5* from PacketBench [12], *lms* and *adpcm* from SNU-RT benchmark suits [13], and *G271 Decode* and *bitcounter* from MiBench [14]. Using GCC (GNU Complier Collection), the DFG and the hotspot of these applications were generated and fed to the ISA extension design flow. The implemented design flow was adapted from [4]. The candidate CIs were identified based on the exact method proposed in [1] with the number of I/O ports and maximum propagation delay as the constraints considered in this phase. In the selection phase, we used the greedy approach to select the best CIs.

We used the canonical form to model the delay variation[15]. To model the canonical delay form of the gates, we used the HSPICE model of gates in a 45nm technology[16]. By using the SSTA method proposed in [15], the canonical delay form of the primitives were calculated. All the primitive delay models were gathered in a library that was used in the design flow to perform SSTA on CIs. Also, for the sake of simplicity without loss of generality, we assumed that there were only the random variations of L_{eff} and V_{th}. We assumed the variation for both V_{th} and L_{eff} were equal to $\sigma V_{th,rdn}/V_{th,0}=\sigma L_{eff,rnd}/L_{eff,0}= 10\%$, where the $V_{th,0}$ and $L_{eff,0}$ are the mean values of the threshold voltage, and the effective channel length of the transistors. The variations were applied to the HSPICE model of the primitive gate in the 45nm technology.

Finally, to evaluate the proposed architecture, we have invoked C# to model the architecture. Using the above statistical parameters, the speedups of one thousand samples of the 5-stage MIPS processor shown in Figure 2 were obtained. For each sample, based on the PDF (normal distribution) of the CIs, random delay values were assigned to the outputs of each CI. For the delays which were greater than the *MPD*, two clock cycles were used for the CI execution.

In the rest of this section, two different speedups are reported. One is the design time speedup which represents the speedup of the extended ISA when all CIs execute in one clock cycle. This means that the CI delays in none of the manufactured chips were larger than the *MPD*. The second is the speedup of the extensible processor which indicates the speedup of the CI when the impacts of the process variation on their delays are considered. This means that some CI delays in the manufactured chips were larger than the *MPD*, and hence, the corresponding CIs execute in two clock cycles.

B. Results

To study the speedup of the extensible processor using the proposed architecture, we extracted the extended ISA under three different cases. In the first case ("1"), the performance yields of the extracted CIs were assumed to be 100%. In this case, the CIs were selected based on the worst-case delay. In the second case ("0.8"), the performance yield constraint was considered and the selection algorithm was forced to select CIs while the performance yield of the CFU was greater than 80%. This meant

$$\forall\ Selected\ CIs, \prod CI_i.PY \geq 0.8 \quad (6)$$

Note that in this case the probability of a CI that violates the *MPD* is equal to 0.2, while this probability in first case is zero. In the last case ("NC") where the performance yield was not considered in the selection process, the CIs were selected based on their nominal delay. In this case, since the possibility of violating the delay constraint by the process variation is not considered in the CI selection process, the chance of violating the *MPD* by the selected CIs is larger than the other two cases leading to a smaller performance yield for this case.

Figure 3 depicts the CFU speedup reported in the design time. The results show in all cases the highest speedup belongs to the "NC" case while the worst-case design (case "1") has the lowest speedup. Also, in all the cases, except for *IP_Sec* and *lms*, the speedup is higher for lower performance yields where there are more options for selecting CIs. This is due to the fact

that the delay variation is either not considered (the "NC" case) or partially tolerated (the "0.8" case). For *IP_Sec* and *lms*, since decreasing the performance yield does not change considerably the extended ISA, the CFU speedup is remained approximately constant.

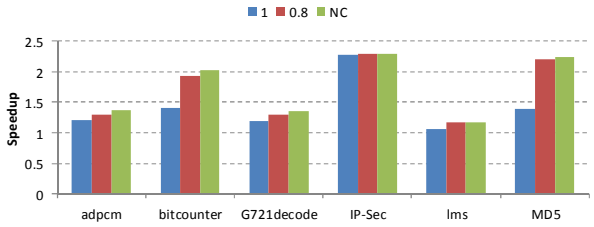
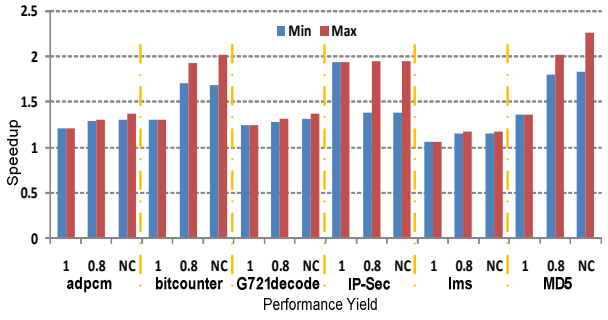


Figure 3. The speedup of the extended ISA reported in the design time.

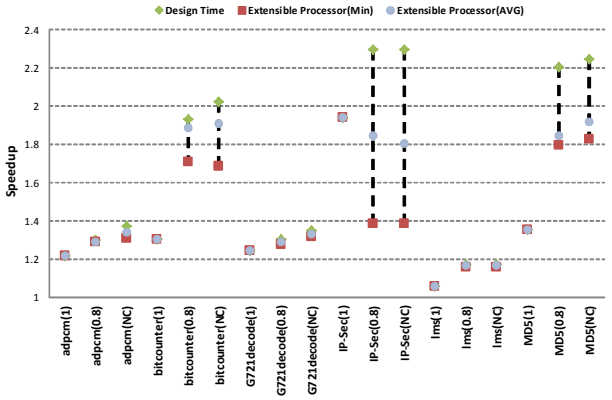
The speedups of the extensible processors under the predefined three cases are presented in Figure 4. In Figure 4(a), in each case, the minimum and maximum of the extensible processor speedups are reported. When the CIs were selected based on their worst-case delay, the delay fluctuation had no effect on the CFU, and hence, the speedup of the extensible processor was constant. In the other cases, the speedup was different from one chip (sample) to other one.

The results show that the highest maximum speedup is achieved when the CIs are selected based on the nominal delay (the "NC" case). Additionally, for all the benchmarks, except for *IP-Sec*, the highest minimum speedup belongs to the "NC" case. These results of Figure 4(a) suggest that, in most cases, the higher design time speedups of lower performance yield cases (the "NC" and "0.8" cases) are not completely cancelled out by the *MPD* violations of some selected CIs in these cases. To show this more clearly, Figure 4(b) shows the difference between the design time speedup and the extensible processor average and minimum speedups. Note that for the "1" case, the design time and extensible processor speedups are the same. In all the cases, except for the *IP-sec* benchmark, the higher reduction (indicated by the dashed line) belongs to the "NC" case. However, the average speedup of the extensible processor for this case is still higher due to the larger design time speedup. For the case of the *IP-Sec* benchmark, the design time speedups of the extended ISA were almost the same in all the two cases of "0.8" and "NC" (see Figure 3). When the design was performed for the "NC" case, the number of paths violating the *MPD* increased (from 105 in the "0.8" case to 111 in the "NC" case). The increase reduced the average speedup. A similar argument is applied when comparing the cases of "0.8" and "1". As the results of Figure 3 and Figure 4(a) show, in the case of the *lms* benchmark, the speedups for the cases of "0.8" and "NC" were similar. The reason was that the same CIs were chosen in the selection phase. Finally, note that figures predict a higher speedup of 18% (11.8%) in the case of "NC" ("0.8") compared to the design based on the worst-case delay ("1").

As mentioned before, to improve the efficacy of the proposed architecture, we have offered a new merit function. Figure 5 shows the minimum and the average speedup under two different merit functions. In the first case, the CIs are extracted based on the conventional merit function while in the second case they are extracted based on the proposed merit function (Equation (5)). Additionally, the CIs were extracted without considering the performance yield ("NC") for this study. The results show that, in all the cases, the speedups are increased when the proposed merit function is utilized. The comparison between the speedups of the conventional and proposed merit function shows that the maximum improvement is for the *MD5* benchmark which is about 4.3%, while the minimum is for the *G721decode* which is about 0.4%. On average, the speedup increase is about 2.18%. Finally, the comparison of this results with the worst-case design (Figure 4) indicates that the highest gain belongs to the *MD5* benchmark (~47.84% improvement), and the lowest one belongs to the *IP-Sec* (~5.55% reduction). Also, the average improvement was about 20.66%.



(a)



(b)

Figure 4. a) Minimum and maximum average speedups of the extended ISA when enhanced with the proposed architecture. b) Design time speedup vs. extensible processor speedup

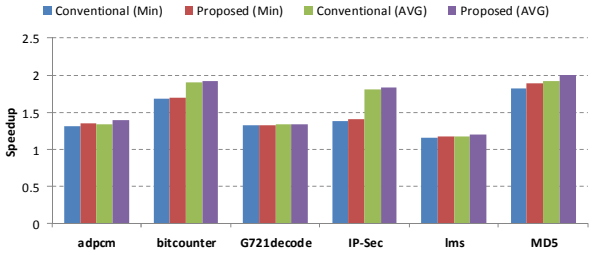


Figure 5. The ability of the proposed merit function in increasing the speedup of the extended ISA enhanced with the proposed architecture.

One solution to mitigate the process variation effect is to increase the clock period [10]. Similarly, one can take the same measure in the case of the extensible processors by increasing the clock period to the maximum worst-case delay of the selected CIs. We have compared the speedups of these two techniques in Figure 6, where ALM and CPM stands for the proposed Architecture Level Method and increasing Clock Period Method, respectively. Note that, in the ALM (CPM), we used proposed (conventional) merit function in selection phase. The results show that, in all the cases, except for *bitcounter* and *IP-Sec*, the proposed method outperforms the CPM. For the *IP-Sec* case, the CPM method is better than the proposed method in this paper. However, in *bitcounter*, the results show that in more than 95% of the extensible processors, the proposed method provides more speedup in comparison to the CPM.

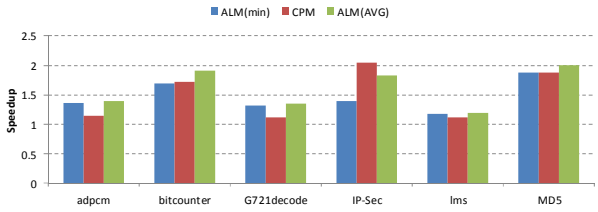


Figure 6. Speedup comparison between the ALM and CPM methods.

V. CONCULUSION

In this paper, we proposed an architecture level method to reduce the process variation impact on extensible processors. The proposed architecture concentrated on the CIs whose latencies were greater than the clock period of the extensible processor. It forced the pipeline processor to let these CIs to use one more clock cycle for their execution. In this method, by running an initialization procedure, these CIs for each chip are determined only one time before its first use. In this procedure, the op-code of the CIs are stored in a look-up table. During the runtime, the system checks the op-codes and lets some of them to be executed with one extra cycle. The results show that the speedup of this method is 18% greater than that of the design based on the worst-case delay. Additionally, to improve the ability of the proposed architecture, we modified the conventional merit function which was used in the design flow of the ISA extension. In the modified merit function, the CIs which adding one more clock cycle to their executions decrease the performance yield less were selected. The results showed that using both proposed architecture and merit function at the same time improved the speedup about 20.7% in comparison to the worst-case approach. Finally, we compared the efficacy of the proposed technique with that of the technique where the clock period is increased. The results showed that the proposed technique reached a better speedup for most of the benchmarks with an average improvement of about 9.3%.

REFERENCE

[1] C. Galluzi, and K. Bertels, "The Instruction-set Extension Proble: A Survey," in *ACM Transaction on Reconfigurable Technology and Systems*, vol. 4, no. 2, pp. 18-1:18-28, May, 2011.

[2] L. Pozzi, K. Atasu, and P. Ienne, "Exact and Approximate Algorithms for the Extension of Embedded Processor Instruction Sets," *in IEEE Transaction on CAD*, vol. 25, no. 7, pp. 1209-1229, July 2006.

[3] Y. Xie and Y. Chen, "Statistical High-Level Synthesis under Process Variability," in *IEEE Transaction Design and Test Computers*, vol. 26, pp.78-87, 2009.

[4] M. Kamal, A. Afazli-Kusha, and M. Pedram, "Timing Variation-Aware Custom Instruction Extension Technique," in *Proceedings of the Design, Automation and Test in Europe (Date)*, 2011, pp. 1517-1520.

[5] N. V. Mujadiya, "Instruction scheduling for VLIW processors under variation scenario," in *Proceedings of the 9th international conference on Systems, architectures, modeling and simulation (SAMOS)*, 2009, pp. 33-40.

[6] P. Ndai, N. Rafique, M. Thottethodi, and S. Ghosh, "Trifecta: a nonspeculative scheme to exploit common, data-dependent subcritical paths," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 53-65, 2009.

[7] T. Sato and S. Watanabe, "Uncriticality-directed scheduling for tackling variation and power challenges," in *Proceeding of 10th international Symposium on Quality Electronic Design (ISQED)*, 2009, pp. 820-825.

[8] X. Liang, G.Y. Wei, and D. Brooks, "Revival: A variation-tolerant architecture using voltage interpolation and variable latency," in *Proceeding of 35th International Symposium on Computer Architecture (ISCA-35)*, 2008, pp. 191-202.

[9] A. Tiwari, S.R. Sarangi, and J. Torrellas, "ReCycle: pipeline adaptation to tolerate process variation," in *Proceedings of the 34th annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 323-334.

[10] E. Chun, Z. chishti, and T.N. Vijaykumar, "Shapeshifter: Dynamically changing pipeline width and speed to address process variations," in *Proceedings of 41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 411-422.

[11] M. L. Bushnell and V. D. Agrawal, *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*: Springer Netherlands, 2000.

[12] R. Ramaswamy and T. Wolf, "PacketBench: A tool for workload characterization of network processing," *in Proc. of IEEE International Workshop on Workload Characterization*, October 2003, pp. 42-50.

[13] SNU-RT Real Time Benchmarks.[Online]. Available: http://archi.snu.ac. kr/realtime/benchmark/.

[14] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of Inernational. workshop on workload characterization*, 2001, pp. 3-14.

[15] R. Chen and H. Zhou, "Fast estimation of timing yield bounds for process variations," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, pp. 241-248, 2008.

[16] FreePDK, AFree OpenAccess 45nm PDK and Cell Library for university, http:// www.eda.ncsu.edu.