Considering the Effect of Process Variations during the ISA Extension Design Flow

Mehdi Kamal[1], Ali Afzali-Kusha[1], Saeed Safari[1], and Massoud Pedram[2]
[1]*Nanoelectronic Center of Excellence, College of Engineering, University of Tehran, Tehran, Iran*
[2] *Department of EE-systems, University of Sothern California, Los Angeles, U.S.A.*

{mehdikamal, afzali, saeed}@ut.ac.ir, pedram@usc.edu

***Abstract*—In this paper, we present a technique for custom instruction (CI) extension considering process variations. The technique bridges the gap between the high level custom instruction extension and chip fabrication in nanotechnologies. In particular, instead of using the conventional static timing analysis (STA), it utilizes statistical static timing analysis (SSTA). Therefore, the approach becomes probabilistic where the delay of each CI is modeled by a Probability Density Function (PDF). Using this probabilistic approach, different subsets of the CIs extension are identified to meet predefined constraints (identification phase) and eventually selected for realization to improve a given merit function (selection phase). In the identification phase, performance yield under both random and systematic variations is added as a constraint. Also, a pruning technique is proposed to decrease the runtime of the systematic variation modeling. The results show that the technique reduces the number of the CIs which need systematic variation modeling by about 24.6% for the cases studied in this work. In the selection phase, both greedy and branch-and-bound approaches are used. In the greedy approach, the conventional merit function based on the cycle saving and area is modified to include the performance yield. The results show the proposed merit function leads to about 3.2% increasing in the speedup. In the branch-and-bound method an effective pruning technique is described to reduce the runtime. The pruning technique is able to reduce the search space about 62%.**

I. INTRODUCTION

The use of embedded processors in a wide variety of platforms such as cell phones, health monitoring devices, and automotive applications is increasing. Similar to many other digital systems, the computing speed and power consumption are two critical design parameters [1]. One of the design approaches for these processors is the application specific integrated circuit (ASIC) design technique where both high speed and low power dissipation are achieved at the expense of higher design cost and lower flexibility. Another solution which has higher flexibility and lower design cost is the general purpose processing (GPP), where the computing speed is lower while the power consumption is higher than those of the ASIC design.

The application specific instruction set processor (ASIP) methodology is the third approach which can improve the computing speed and power consumption of the GPP technique [2][3]. In the ASIP design approach, the instruction set of a GPP is extended through ASIC design based on the features of the specific application. Note that this kind of extension can be performed for graphical processors as well as VLIW architectures [4]. The augmented instructions are determined such that the desired speed, power, and cost requirements are fulfilled. The main idea behind using the ASIP design approach is to run the hotspot parts of an application using custom instructions (CIs) and the other parts of the application on the Arithmetic Logic Unit (ALU) of the processor. The CIs (which are executed using a hardware block which is called a Custom Functional Unit (CFU) in parallel with the ALU of the GPP) improve the processor speed (performance and computing speed are used interchangeably in this paper) by increasing the instruction level parallelism, and reducing the register file accesses. Similarly, by decreasing the accesses to the cache and register file, the CIs reduce the power consumption.

The ASIP design approach starts by extracting the data flow graph (DFG) of the application [1][5]. Next, all the subgraphs that meet the constraints of the parallel hardware are enumerated as CIs. Convexity, I/O port count, and propagation delays of the subgraphs are common constraints in the CI enumeration. Finally, the best CIs among the candidate CIs, will be selected based on their merit function values. For each CI, the merit function value shows the value of the parameters that should be improved by the ASIP. Normally, the main parameter of interest is the computational speedup, which shows how much of performance improvement is obtained using the CIs [1][2].

In the conventional ASIP approach, the worst-case delays of the primitives (*e.g.,* AND, ADD, SHIFT, etc.) are used as the reference to extract the latency of the CIs. In sub-100nm nanotechnologies, however, complexities in the manufacturing of the transistors with small sizes have resulted in significant variations in nominal transistor parameters (such as threshold voltage and effective channel length), which in turn has led to uncertainties and inefficiencies in the performance and power consumption optimization of the circuits [6]. There are two types of variations: (i) within die or intra-die, and (ii) die to die or inter die variations. Intra-die parameter fluctuations consist of both random and systematic components. An example of the systematic intra- die variation is the aberrations in the stepper lens while the placement of dopant atoms in the device channel region is an example of the random variation [7]. Hence, for each parameter of the transistor, we cannot consider only a fix value. Instead, the values of a parameter are shown by PDF (Probability Density Function) which shows the probability of each parameter value.

As the process variation impact increases, the gap between the high level design and fabrication may increase if proper statistical techniques are not invoked. The reason for the increase in the gap originates from the fact that the resolutions of fabrication processes/equipments remain constant while the feature size is decreased increasing the process variation impact. Hence, the values of transistor parameters which are considered in the design time may be totally different from the values of the transistor parameters in the manufactured deign. Based on the ITRS report [8], the performance variation would be 77% in 2024. Therefore, design based on the process corners to meet the latency constraint is inadequate [6]. In the worst-case design approach, to guarantee that the target delay requirement is fulfilled, the worst-case process corners should be considered. This approach has become inefficient because of variability increases due to technology scaling where the variability has exceeded an acceptable norm. For these technologies, using the worst-case approach means more and more margins should be considered to cover these increasing variations leading to overly pessimistic design and unacceptably increasing the cost. On the other hand, in the statistical design approach, where statistical variations are handled realistically, various trade-offs such as performance versus yield can be made. This will lead to the prevention of loss of performance due to guard-banding as well as new possibilities for optimization and robust design methodologies. In this approach, instead of modeling the variations of physical parameters as a fixed set of corners, they are modeled using probability density functions which represent their variability around their typical (nominal) values. Hence, the design flow of the embedded system is not an exception either and should shift from deterministic to probabilistic approaches.

Based on data reported in [9][10], due to the existence of many critical paths in the execution unit, the execution stage is more critical in comparison to other pipeline stages, and hence, it must be considered in a variation-aware design flow. In the case of ASIP processors, the designer increases the speed at which the processor runs an application by defining new instructions. Each of these instructions is created by combining basic primitives. Adding new instructions increases the number of critical paths in the execution unit. Therefore, these new instructions increase the impact of the process variation on the extended processor.

In this paper, we describe a statistical design flow for extending the instruction set architecture (ISA) by considering the manufacturing process fluctuations in the high level design. Both systematic and random variations are modeled and considered in both identification and selection phases. Additionally, to evaluate the proposed methods, a bunch of benchmarks from different benchmark suits are used (*i.e.*, PacketBench [26], SNU-RT [27], Mibench [28], MediaBench [29] benchmark suits).

The rest of this paper is organized as follow. In Section II, the ISA extension methodology and the impact of the process variation on CIs are described. Section III briefly reviews the related works while the proposed design flow is explained in Section IV. The experimental setup and the results are described in Section V. Finally, the paper is concluded in Section VI.

II. ISA Extension and the Impacts of the Process Variation

In this section, first, the overall design flow of the ISA extension is described. Also, the problem of finding the subset of CIs which maximizes the total speedup is formulated. Next, the impacts of the process variation on the selected CIs and the importance of the considering process variation on the design flow are discussed.

A. ISA Extension Flow

The overall flow of the ISA extension is shown in Figure 1. The ISA extension starts by extracting the data flow graph (DFG) of the application. The DFG shows the flow of the instructions which should be executed by the processor. Custom (extended) instructions are subgraphs of this DFG which meet some predefined constraints such as convexity, I/O ports, and propagation delay. A subgraph is architecturally feasible if its inputs are available at the time of selection which is only possible if the subgraph is convex. A subgraph S is called convex when there exists no path from a node $u \in S$ to another node $v \in S$ while the node $w \notin S$ involves in the path [1].

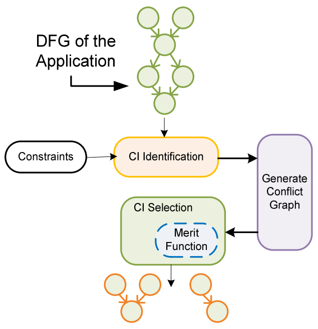


Figure 1. ISA extension flow.

Figure 2 shows examples of a DFG and two subgraphs. The subgraph *ADB* is nonconvex due to the path A→C→D while the subgraph CD is convex.

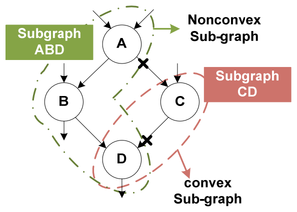


Figure 2. Examples of a DFG and two subgraphs.

Also, the number of inputs and outputs of the subgraphs should be equal or less than the read and write (R/W) ports of the base processor register file. If the I/O number is larger than the register-file R/W ports, we should use pipelining method to resolve this problem which decreases the performance of the extended processor [15]. Finally, the CI delay is defined based on the clock period and the execution unit of the processor. If the execution stage of the processor is single cycle, the subgraphs which their propagation delay is less than one clock period cycle are acceptable. If the execution stage is multi-cycle, the CI delay should be less than the number of cycle times. The enumeration of the feasible subgraphs is done in the identification phase.

If a DFG node is included in two CIs, these two CIs are considered to have overlap with each other. Because each node in the DFG should not be executed by more than one CI, all the CIs that have an overlap with a selected CI are removed from the list of the identified CIs in the selection phase. Based on the overlap between the CIs, a conflict graph is constructed. The nodes in the conflict graph are CIs and an edge between two nodes shows that these CIs have overlap.

In the last phase (selection phase), the best CI groups are selected among all identified CIs. There are two general approaches for the CI selection which are optimal and nonoptimal approaches [4]. The former are exact methods which finds the best (optimal) CIs set by searching all the search space. This technique may become intractable for real applications where the number of nodes in the DFG of the application is too large. Hence, in this approach, we should prune the search space to make this algorithm practical. nonoptimal approaches are much faster than the optimal techniques. While there is no guarantee for the optimality of their solutions, in many cases, these techniques indeed may obtain optimal solutions. In this paper, we consider the branch-and-bound and the greedy algorithms as the optimal and nonoptimal algorithms, respectively.

In the selection phase, the candidates are selected based on their merit value. Normally, cycle saving is the main objective in the ISA extension. The cycle saving may be computed as

$$CS_i = \#Iteration \times \left(\#CI_i.SW - CI_i.IO\_Penalty - ceil\left(\frac{CI_i.CriticalPathDelay}{Clock\ Period}\right)\right) \qquad (1)$$

where CS_i is the cycle saving of the i^{th} CI (CI_i), *#Iteration* is the execution frequency of the basic block to which CI_i belongs, and $\#CI_i.SW$ is the number of cycles that the CI needs to run by the base processor. Also, $CI_i.IO\_Penalty$ is the number of extra accesses to the register file for reading data to or writing data from (when the number of CI I/O ports is more than the number of register file read/write ports) and the fraction is the number of cycles for executing CI_i on the CFU (we assume that CIs can be multi-cycle or single cycle.) In the fraction, $CI_i.CriticalPathDelay$ is the propagation delay of the CI critical path and *Clock Period* is the desired clock period for the extended processor.

In addition to cycle saving, if there is a defined area budget for implementing the CFU, the merit function is specified as *Cycle saving*/*Area*[5]. In this case, the merit function of the i^{th} CI is defined as

$$Merit_i = \frac{CS_i}{Area_i} \qquad (2)$$

where $Area_i$ is the area of the i^{th} CI.

It should be note that there is another ISA extension flow which combines the identification and selection phases with each other. It means, in this approach, instead of extracting all feasible subgraphs in the identification phase, the best subgraphs which do not have conflict with each other and meet the defined constraints, are extracted. Hence, in this flow, the merit function is used in the identification phase. The combination approach is faster and also needs less memory space compared to the case when the identification and selection phases are separated. However, the later approach is better in finding the isomorphic CIs which reduce the area usage and increase the speedup. Additionally, in the latter case, a higher speed gain may also be achieved because the CI selection is performed among all the identified CIs.

Finally, we can formulate the instruction selection problem as

$$Maximize \sum_{i=1}^{|Selected\ CI|} Cycle\ Saving_{CI_i} \qquad (3\text{-a})$$

while

$$\sum_{i=1}^{|Selected\ CI\ |} Area_{CI_i} < Area_{Constarint} \qquad (3\text{-b})$$

Equation (3-b) is applicable when the area constraint ($Area_{Constarint}$) is defined as a limitation in the ISA extension.

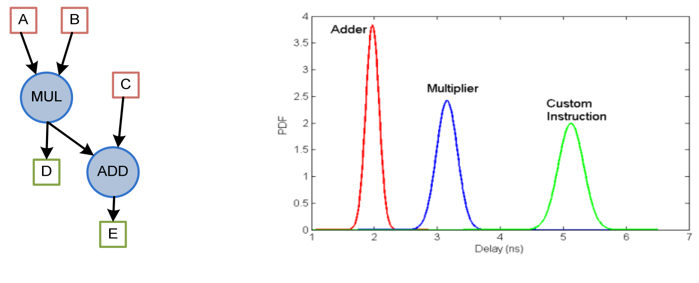
B. Process Variation Impact on ISA Extension

In conventional approaches, the delay of a CI is estimated as a constant value which is equal to the summation of the worst-case delays of the nodes (primitives) in the critical timing path which is activated during the execution of the CI [6]. Due to increase in the parametric variability in the nanoscale technologies [21], using the worst-case approach is not appropriate and statistical approaches for the delay estimation should be utilized [6]. Figure 3(a) shows an identified CI which contains two nodes whose PDFs are shown in Figure 3(b). As discussed in the introduction section, the delay of the digital circuits due to the process variation is not a constant value. Hence, the PDF show the distribution of the delay of the design.

If we wish to use the worst-case approach, the delay of the CI is equal to the delay of the multiplier plus the delay of the adder. The worst-case delay (D_{WC}) is given by [6]

$$D_{WC} = \mu + 3\sigma \qquad (4)$$

where shows the primitive worst-case delay with the mean and standard deviation of μ and σ, respectively. In our example, the worst case delay of the CI is equal to $(\mu_{Adder} + 3\sigma_{Adder} + \mu_{mul} + 3\sigma_{mul}) = 5.93ns$. Using the statistical information, the delay of the CI is modeled as a PDF which shows the delay behavior of the CI by considering the process variation. It shows that the mean value of the delay is 5.124ns while its variation is about $\pm 11\%$.



(a) (b)

Figure 3. An example of a CI a) CI schematic b) PDF of adder, multiplier, and CI.

In the conventional approaches, the CIs which do not meet the clock period constraint (which is one clock period in non-pipelined ALUs or more than one clock periods in pipelined ALUs) in the identification phase, are removed (assuming the delay is the only concern in the CI extension). In the presence of variability, we can use the worst-case delay. However, due to the variability increase, using the worst-case delay leads to a pessimistic design. In a statistical design approach, a performance yield metric is defined which describes the probability for the circuit to meet the predefined delay constraint [11]. Hence, in statistical approach, in addition to the delay constraint, we should specify this yield constraint. In the identification phase, the CIs whose performance yields are smaller than the specified performance yield are eliminated. Similarly, in the conventional approaches, in the selection phase, the best CIs are selected. The best CIs are the ones whose cycle savings are larger than the others. In the statistical approaches which consider the process variation, we should also include the performance yields of the CIs in our selection process.

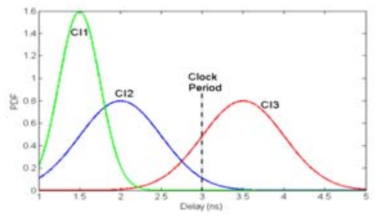


Figure 4. The PDF of three CIs.

As an example, the PDF of three CIs are depicted in Figure 4 where the clock period of the processor is assumed to be 3 ns and the performance yields of the CIs must be greater than 90% (the details will be described in the next section). The CI3 will be removed in the identification phase, because its performance yield is about 16% which is smaller than the defined constraint. The performance yield of the CI1 and CI2 are 100% and 96%, respectively. Hence, both of them are passed to the selection phase. Note that if instead of the statistical approach, we use the worst-case approach, the CI2 is also will be removed from the candidate set. This indicates that the candidate set in the statistical approach has more candidate CIs than the worse-case approach providing us with more options for increasing the speed. The ability in enhancing the speed will be seen in Section V when we present the speed gain of the statistical approach compared to the worse-case approach.

In the selection phase, the CIs with the greater merit value will be selected. Two merit functions may be defined in this phase. In one function, we can only include the speedup where the CI with the greater cycle saving will be selected. In the other function, both speedup and performance yield are included where the CI which leads to a larger speedup and performance yield will be selected. For example, in Figure 4 if we assume that the cycle saving of the CI1 and CI2 are the same, CI1 will be selected.

Therefore, the instruction selection formulation by considering the performance yield as a new constraint is modified as

$$Maximize \sum_{i=1}^{|Selected\ CI|} Clock\ Saving_{CI_i} \qquad (5\text{-a})$$

while

$$\sum_{i=1}^{|Selected\ CI\ |} Area_{CI_i} < Area_{Constarint} \qquad (5\text{-b})$$

$$\prod_{i=1}^{|Selected\ CI\ |} PY_{CI_i} > PY_{Constarint} \qquad (5\text{-c})$$

Here, the PY_{CI_i} is the performance yield of the i^{th} CI, and the $PY_{Constraint}$ is the predefined performance yield constraint. The overall yield of the extended ISA must be greater than $PY_{Constraint}$.

III. RELATED WORK

There are many published results on modeling and mitigating the process variability at the device and circuit levels of design abstraction. There are also some work in high-level synthesis (HLS) where techniques have been proposed to improve the performance and reduce the hardware cost considering process variations (see, *e.g.,* [6][7][9]-[12]). The impacts of the process variation on pipelined processors have been discussed in [9][10][13]. In these works, techniques to mitigate these impacts have also been suggested.

Additionally, there exist works in the field of ISA extension. The works deal with both the CI identification (see, *e.g.,* [1][3][14]-[17]) and selection (see, *e.g.,* [2][5][19]).

An algorithm that explores the search space of candidate subgraph set exhaustively for a single feasible CI that meets the I/O constraint is presented in [3]. The algorithm which pruned the search space runs in linear time for most DAGs (Direct Acyclic Graph) having practical numbers of nodes. In [1], the problem of identifying CIs was solved by using the genetic algorithm while in [14], the problem was formulated and solved by the Integer Linear Programming (ILP) approach. The works in [15][16][17] proposed methods to enumerating all subgraphs of the DFG (data flow graph). In these methods, only the convexities of the subgraphs were important while the I/O constraint was not defined. The technique proposed in [18] was able to identify CIs with or without the I/O constraint. In all of the proposed identification methods, increasing the cycle saving of the selected CIs has been the goal when only observing the I/O constraint and convexity of the selected CI.

In [2], for selecting the CIs (the selection phase), an algorithm based on the optimal method of the branch-and-bound and the non-optimal method of the greedy approach has been proposed. The proposed method makes the runtime of the selection process reasonable for practical cases. The optimal method may require a very large runtime in these cases. The objective of the technique was to improve the performance without considering the area budget. The technique proposed in [19] partitioned the solution into several sub-problems and presented an optimal branch-and-bound covering algorithm. This algorithm used subgraph size (number of nodes in the subgraph) as its merit function and relied on sorting the candidates in decreasing order of size. An automatic framework for designing a customized processor has been suggested in [5]. The proposed method handles the instruction set identification and selection. In the selection phase, the cycle saving and area budget of the CIs are used as the merit function. In these methods, the merit value shows the cycle saving of the CI or combination of the cycle saving and area budget.

All of the proposed approaches suggested techniques for improving the speed by finding a better ISA extension in the identification and selection phases. It should be noted that in these methods deterministic values of parameters were used in estimating the delay of the CIs. To show the importance of switching from deterministic approach to the statistical design, a yield-aware ASIP design methodology was proposed in [20]. The considered variation model in this work was simple. In this work, we propose a design flow which models both the systematic and random variations of the CIs in the identification phase. Additionally, we improve the procedures in the selection phase to select CIs with higher cycle saving while both area and performance yield constraints are not violated.

IV. Proposed Design Flow

The proposed design flow which is shown in Figure 5 has four main stages. The flow starts by enumerating the CIs in the identification phase which is performed in stages 1 and 2. Based on the random (pruning 0) and systematic (pruning 2) variations, the CI candidate set is pruned. In the proposed ISA extension flow, stage 3 generates the conflict graph, and finally, the best candidate CIs are selected in the stage of selection phase (stage 4). Here, the detailed explanation of the design flow is given by separating the identification and selection phases. It should be noted that the design flow shown in Figure 5 is applicable to the case of single-cycle CI selection. By some modifications, the flow may be used in the cases where the pipelining or resource sharing are employed to improve the speedup and the area usage of the extended ISA.

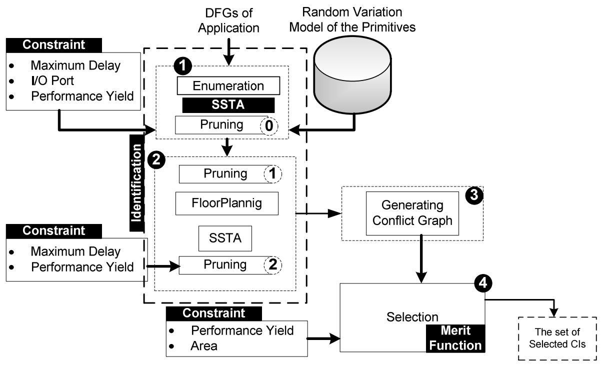


Figure 5. Overall view of the proposed ISA extension design flow.

A. Identification Phase

To model the delay variation of the CIs, we need the random variation model of each primitive. Using this variation model, a database containing the delay model of the primitives is formed. This database contains the delay variation of the CIs based on the random variation. However, to exactly model the systematic variation of the CIs, we need their placement information which is not available in this phase. Since we do not have these pieces of information, we take the following measures.

In the first step of the proposed flow, the candidate CIs are enumerated based on the predefined constraint, such as the number of I/O ports and maximum propagation delay. In conventional approaches (deterministic approaches), using STA, the propagation delay of a CI is calculated. If the propagation delay is larger than the maximum propagation delay, the enumerated subgraph is not selected as a CI candidate. In the proposed statistical approach, instead of using STA in the flow, we should use a Statistical STA (SSTA) to model the delay of CIs. Therefore, in this phase, we need an SSTA to model the PDF of the enumerated subgraphs. Additionally, using the maximum delay constraint the CDF of the subgraph is calculated. Finally, the comparison between the CDF of a subgraph and the predefined performance yield shows that whether the enumerated subgraph could be a candidate CI or not. The output of this stage is a set of the candidate CIs which satisfy the predefined constraints.

Note that, up to this stage, the CDFs of the CIs have been extracted based on the random variations. After applying the systematic variations, some of the candidates may be removed from the set. Next, we propose a method to model the systematic variation of the candidates.

To model the systematic variations, the placements of the gates in the critical paths in the layout design are needed. While this approach is accurate, the runtime of the placement algorithm to obtain this information for all the candidates will be too high (for some practical cases where the candidate CIs might be as high as one hundred thousands or even more). To reduce the runtime, one suggests using an approximate method to obtain this information. We assume that the systematic variation on a CI is estimated based on the dimension of the primitives in a CI. It is supposed the critical path in a CI is scattered in the whole area of the CI. This is an upper bound on the fluctuations due to the systematic variations. By this assumption, to model the systematic variations of the primitives in a CI, we only need the floorplanning of the primitives. Figure 6 shows a CI and the floorplan of its primitives. To find the floorplanning, the simulated annealing technique, which is a common approach for floorplanning, is used in this work.

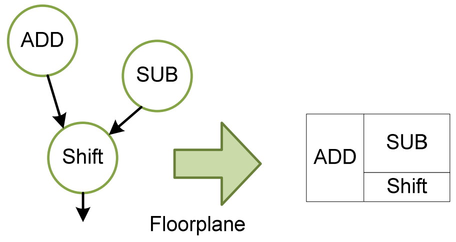


Figure 6. An example of a CI and the floorplanning of its primitives.

There are methods which have been proposed to model the spatial correlation of the systematic variation from the floorplanning (*e.g.*, [22][23]). Using these methods, we model the systematic variations of the CI delay. For example, the systematic variation model based on the multi-level quad-tree modeling, which is discussed in [22], is shown in Figure 7. In this method, the die area is divided into a multi-level quad-tree partitioning where the spatial correlation decreases as the level number decreases. Figure 7 shows the chip area partitioning in three levels. In each level, the die area is divided into 4^{LN} squares, where *LN* is the level number. As the number of level increases, the number of square also increases. Also, for each square (s) in each level (*LN*), a random number denoted by ΔL_s^{LN} is generated based on the systematic variation assigned to each level. Note that the total with-in die variation is divided among the levels. By summing the square variations (ΔL_s^{LN}) for each gate, its intra-die variation is obtained. For example, in Figure 7, assume that the *Gate A* belongs to square 13 of level 2, square *2* of level 1, and square *0* of level 0. The length (systematic) variation of this gate may be obtained from

$$\Delta L_{Gate\ A} = \Delta L_{13}^{2} + \Delta L_{2}^{1} + \Delta L_{0}^{0}$$

Using Figure 7, we can model the systematic variations of the three primitives of ADD, SUB, and Shift as

$$\Delta L_{ADD}^{SYS} = \Delta L_0^2 + \Delta L_1^2 + \Delta L_4^2 + \Delta L_5^2 + \Delta L_8^2 + \Delta L_9^2 + \Delta L_0^1 + \Delta L_2^1 + \Delta L_0^0$$

$$\Delta L_{SUB}^{SYS} = \Delta L_1^2 + \Delta L_2^2 + \Delta L_3^2 + \Delta L_5^2 + \Delta L_6^2 + \Delta L_7^2 + \Delta L_0^1 + \Delta L_1^1 + \Delta L_0^0$$

$$\Delta L_{Shift}^{SYS} = \Delta L_5^2 + \Delta L_6^2 + \Delta L_7^2 + \Delta L_9^2 + \Delta L_{10}^2 + \Delta L_{11}^2 + \Delta L_0^1 + \Delta L_1^1 + \Delta L_2^1 + \Delta L_3^1 + \Delta L_0^0$$

where the ΔL_x^y shows the variation of the x^{th} square in the y^{th} level. Based on formulation, the systematic variations may be added to the primitive delay model to make it more accurate.

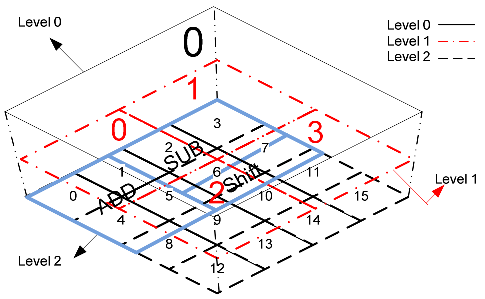


Figure 7. Modeling the spatial correlation on a CI. There are three levels.

The number of the CIs has a major impact on the runtime of the floorplanning algorithm, and hence, their number should be reduced as many as possible. Note that the performance yields of some CIs when only the random variation is considered are one. This may mean that the delays of the gates of these CIs are considerable smaller than the delay constraint. For these cases, given the small area of the CI, we may safely assume that the systematic variation may not change the performance yield and hence they may be removed from the set which should be floorplanned. This way, the runtime of floorplanning is reduced lowering the overall runtime of the design flow. To do this pruning, we extract the maximum delay impact of the systematic variation on the CIs whose performance yields are 1(pruning 1).

To model the maximum impact of the systematic variation, we use the FO4 inverter chain model of the CI (see, *e.g.,* [24]). This model means that the delay of the chain is equal to the nominal delay of the CI. The standard deviation of the systematic fluctuations for the chain is proportional to the number of its elements [7]. Also, in a highly correlated area, the fluctuation is zero. Hence, we modified the equation proposed in [7] to

$$\delta_{SYS_{Chain}} = \frac{CI\ Nominal\ Delay}{FO4\ Delay} \times \frac{\sqrt{2Inv\ Area}}{Width(HCA)} \times \delta_{SYS} \quad (6)$$

where $\frac{CI\ Nominal\ Delay}{FO4\ Delay}$ shows the number of inverter gates in the FO4 model of the CI, HCA is highly correlated area, *Width*(*HCA*) is the maximum distance between any two points in the HCA (*i.e.*, if the HCA has a square shape, its width is $\sqrt{2HCA}$), $\sqrt{2Inv\ Area}$ is the maximum distance between any two points of an inverter gate, and finally δ_{SYS} is the sigma of the systematic fluctuation. Figure 8 shows a CI and its corresponding FO4 model as well as the HCA slice of the FO4 chain.

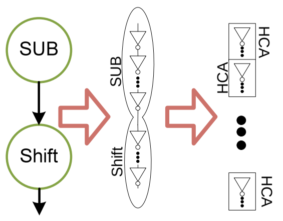


Figure 8. A CI and its corresponding FO4 model as well as the HCA slices of the FO4 chain.

After the floorplanning, the PDF delay of the CIs should be modeled again using SSTA.

B. *Selection Phase*

After Identification, the conflict graph should be generated. The last step for the ISA extension is CI selection. In this phase, among the candidate CIs, the best ones are selected as the extended ISA set. There are two general approaches for CI the selection which are the branch-and-bound and greedy approaches. In the former type, which is exact, the whole design space is searched to find the set of best CIs. These algorithms are generally time-consuming. To make them feasible, some pruning techniques, which depend on some parameters such as the extended ISA processor target and constraints, have been proposed (see, *e.g.,* [25]). On the other hand, while the greedy approach is fast, the selected CI set is not necessarily the optimal set.

In the statistical approach, to meet the predefined performance yield constraint (PY_{Const}) in an extended processor, the following equation must be satisfied:

$$PY_{Const} \leq PY_{Processor} \times \prod_{i=1}^{|Selected\ CIs|} PY_{CI_i} \qquad (7)$$

Here, $PY_{Processor}$ is the performance yield of the baseline processor (the processor without CFU), and PY_{CI_i} is the performance yield of the i^{th} selected CI. Hence, in the selection phase, the performance yield of the extended ISA must be greater than $\frac{PY_{Const}}{PY_{Processor}}$. Therefore, compared to conventional (deterministic) approach, a new constraint is added to the selection phase and the selection algorithm needs to be modified accordingly.

The proposed merit function should increase the cycle saving of the selected CIs without violating the area and performance yield constraints. Our studies showed that the higher performance yields obtained from CIs with smaller area usages. Therefore, (2) implicitly depends on the performance yield too. To make this clearer, consider the example of the G721decode benchmark. Figure 9(a) shows the joint distribution of the area and performance yield (the performance yield constraint was defined 0.90) of the CIs for this benchmark. The figure shows that, for most of the cases, CIs with smaller areas provide higher performance yields when compared to CIs with larger areas. CIs with more areas may have more gates in series leading to longer delay paths, and hence, the chance of violating the delay constraint due to the variations increases. In Figure 9(b), the distribution of the performance yield of the CIs is plotted. The distribution is mainly concentrated around 99.8% with some distribution scattered down to 91.6%. Our study showed that a similar behavior existed for other benchmarks. In the case of area, no similar concentration of the distribution is observed (see Figure 9(c)). Note that there are a considerable number of CIs with small areas. They have a higher chance of being selected if (2) is used. In most cases, however, the cycle savings of these kinds of CIs are low. Hence, to enhance the role of cycle saving in the selection process, another merit function should be invoked. We suggest using the merit function given by

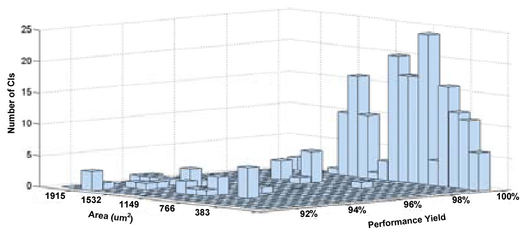
$$\text{Merit}_i = f(\text{PY}_i) \times \text{S}_i \qquad (8)$$

where $Merit_i$ shows the merit function of the i^{th} CI when the performance yield is also considered, and $f(PY_i)$ is a function obtained by a linear mapping of PY_i. Note that there is an implicit relation between the area and the performance yield as was discussed above, and hence, (8) implicitly considers the area in the selection process as well.

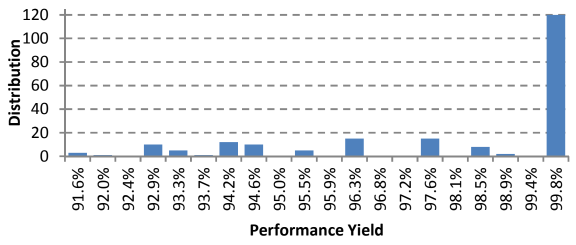
The mapping which maps the smallest PY ($= \frac{PY_{Const}}{PY_{Processor}}$) to zero and the largest PY (100%) to one, is given by

$$f(PY_i) = 1 + (\frac{PY_{Processor} - PY_{Const}}{PY_{Const}} \times \left(PY_i - \frac{PY_{Const}}{PY_{Processor}}\right)) \qquad (9)$$

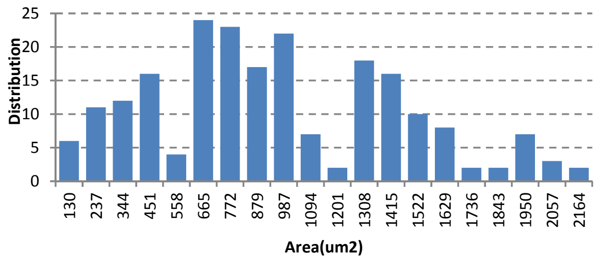
Finally, note that when the area budget (constraint) is very small, the importance of using the area parameter should be more. Hence, it is expected that (2) which has the area as an explicit parameter, give better results in comparison to (8)



(a)



(b)



(c)

Figure 9. Distributions of a) CIs versus performance yield and area b) only performance yield, and c) only area of the G721decode benchmark.

In the modification of the exact algorithm, pruning techniques play a major role. The proposed pruning technique depends on the constraints and the type of base processor (*e.g.*, VLIW and RISC). We, however, can limit the runtime of this approach by pruning the design space (search space). To do this, the candidate CIs might be sorted based on their merit values (*i.e.*, Equation (8)) or only based on their cycle savings. After sorting, a subset with better merits are chosen and sent to the selection phase. Then, in the selection phase, if the branch-and-bound technique is used, the whole design space must be searched to select the best candidates.

Figure 10 shows a branch-and-bound algorithm to select the CIs from a candidate set. In this example, the area is the only predefined constraint. Thus, first, we sort the CIs from the smallest to largest ones in the candidate set, and then, call the selection function (shown in Figure 10). The sorting helps to prune the decision tree (search space). In the selection function, if the algorithm reaches a CI whose selection violates the area budget of the CFU, the algorithm will return (will not continue that subtree) (lines 5, 6, and 7). Also, in this code, the Selected CIs Set is an empty set where, in each step, the selected CI is added to it. Additionally, before the algorithm being returned, this set is checked to determine if the set is the best one (lines 3 and 6).

```
1:BaB_Selection(int Index)
2:  If Reach the maximum number of the selected CIs
3:      Is it the best Selected CIs Set
4:      Return
5:  If CFU_Area + $CI_{Index}$.Area > Area_budget
6:      Is it the best selected CIs
7:      Return
//---------- $CI_{Index}$ will be selected----------------------------------
8:  Remove the CIs have conflict with the $CI_{Index}$ from the candidate set
9:  CFU_Area += $CI_{Index}$.Area
10: Add the $CI_{Index}$ in Select CIs Set
11: BaB_Selection(Index +1)
12: CFU_Area -= $CI_{Index}$.Area
13: Remove the $CI_{Index}$ from Select CIs set
14: Add the removed CIs due the conflict with the $CI_{Index}$ to the candidate set
//--------- $CI_{Index}$ will not be selected-----------------------------
15: BaB_Selection(Index +1)
```

Figure 10. Pseudo code of the branch-and-bound method when the area is the only predefined constraint.

In the proposed design flow, if the performance yield is the only constraint in the selection phase, the algorithm proposed in Figure 10 is applicable by using the performance yield instead of the area. On the other hand, if both the area and performance yield are the constraints, then we have to determine how the sorting should be performed. The criterion chosen for this sorting will be useful, if it can prune the design space. Hence, we should sort based on only one of the performance yield or area. This decision should be made before the selection phase. To find which one can prune the search space more than the other one, we use a heuristic approach whose pseudo-code is presented in Figure 11. The decision relies on the number of CIs which can be selected from the beginning of the sorted CIs while the constraint is not violated. The criterion which leads to the smaller number of the CIs is the one which can prune the search space more.

```
1: Make_Desicion()
2:  Sort the CIs based on the Performance Yield from highest to the lowest
3:  Maximum PY_Limit where $\prod_{i=0}^{PY\_Limit} CI_i.PY < PY_{Constraint}$
4:  Sort the CIs based on the Area from smallest to the largest
5:  Maximum Area_Limit where $\sum_{i=0}^{Area\_Limit} CI_i.Area > Area_{Constraint}$
6:  if (PY_Limit < Area_Limit)
7:      Decision: Sort based on the Performance Yield
8:  else
9:      Decision: Sort based on the Area
```

Figure 11. Pseudo code of the decision method.

The proposed branch-and-bound technique for the selection phase is given in Figure 12. When the constraint based on which the sorting has been performed is violated, the algorithm will return (one step back in the recursive calling of the *BaB_Selection()* function). If the violation is for the other constraint, the search continues neglecting the violating CI (by calling the *BaB_Selection()* function).

```
1: Make_Decision()
2: Sort the candidate set based on the decision
3: BaB_Selection(int Index)
4:   If Reach the maximum number of the selected CIs
5:        Is it the best Selected CIs Set
6:        Return
7:   If CFU_Area + $CI_{Index}$.Area > Area_budget
8:      if (the candidate is sorted based on the Area)
9:           Is it the best selected CIs
10:          Return
11:      else
12:          goto **Line 28**
13:  If CFU_PY * $CI_{Index}$.PY < predefined PY
14:     if (the candidate is sorted based on the Performance Yield)
15:          Is it the best selected CIs
16:          Return
17:     else
18:         goto **Line 28**
//---------- $CI_{Index}$ will be selected----------------------------------
19:  Remove the CIs have conflict with the $CI_{Index}$ from the candidate set
20:  CFU_Area += $CI_{Index}$.Area
21:  CFU_PY *= $CI_{Index}$.PY
22:  Add the $CI_{Index}$ in Select CIs Set
23:  BaB_Selection(Index +1)
24:  CFU_Area -= $CI_{Index}$.Area
25:  CFU_PY /= $CI_{Index}$.PY
26:  Remove the $CI_{Index}$ from Select CIs set
27:  Add the removed CIs due the conflict with the $CI_{Index}$ to the candidate set
//--------- $CI_{Index}$ will not be selected-----------------------------
28:  BaB_Selection(Index +1)
```

Figure 12. Pseudo code of the branch-and-bound method when both the area and performance yield are the predefined constraints.

V. RESULTS AND DISCUSSION

A. *Experimental Setup*

To assess the efficacy of the design flow, the custom instructions of a several benchmarks were extracted. The selected benchmarks included *IP-Sec* and *MD5* from PacketBench [26], *lms* from the SNU-RT benchmark suits [27], *adpcm* and *bitcounter* from MiBench [28], and *G271 encode/decode* form MediaBench [29]. The proposed design flow was implemented by the C# language. Using the GCC (GNU Complier Collection), the DFG and the hotspot of these applications were generated and fed to the ISA extension design flow. To extract the basic blocks and also the DFG of the applications, we used the GIMPLE. It is a part of the GCC tool chain which maps the input source code to an intermediate representation language. The language, which is simple, is similar to the assembly language. Hence, we can easily extract the DFG and also the basic blocks of the input applications (*i.e.*, benchmarks). Additionally, for each benchmark, we performed a benchmark profiling to find the number of times that each basic block of the benchmark is called during its runtime. Therefore, the dynamic behaviors of the benchmarks are extracted and used in calculating the speedup. For the profiling, we used an input pattern for the application. Note that, in the cases where the benchmark package contains large examples of input pattern, we have used one of those inputs. In the other cases, we utilized a large random input. Additionally, note that the DFG extraction and also the profiling were performed based on the machine definition of the baseline processor.

To extract the PDF of the CIs, we use the canonical delay model [22]. To obtain the canonical delay forms of the gates, we have used the HSPICE simulations for a 45nm technology. Using the SSTA methods proposed in [30], the canonical delay forms of the primitives were calculated. All the delay models of the primitives were gathered in a library which is used by the design flow to do the SSTA on the CIs. Also, to identify the candidate CIs, the exact method proposed in [1] was used. The number of the I/O ports, maximum propagation delay, and the performance yield were the constraints in this phase. To find the floorplanning of the primitives in each CI, we implemented a method based on the simulated annealing (SA). The method also used the genetic algorithm in combination with SA to minimize the area of the CIs. In the selection phase, we implemented both greedy and branch-and-bound techniques. In the greedy approach [2], the merit function proposed in (8) was used. As mentioned before, in the case of the branch-and-bound technique, the subset of the candidate CIs were pruned based on the cycle saving before the selection to reduce search space. In this case, the selection was performed based on the algorithm proposed in Figure 12.

In this paper, we assumed that there were only L_{eff} and V_{th} variations and the systematic and random variation were independent of each other. Also, without loss of generality, we assumed that $\sigma L_{sys}/L_0$= 5%, and $\sigma V_{th\_rdn}/V_{th0} = \sigma L_{rnd}/L_0$ = 10% (see, *e.g.,* [8]). Additionally, to model the systematic variation, we defined the HCA about 100 μm^2.

B. Results

1) Identification Phase

In the identification phase, to show the impacts of the variations on the candidate CIs, first, the CIs were enumerated based on the nominal delay. Then, by modeling the random and systematic variations, the candidate set was pruned. The random fluctuations change the delay of the CIs causing some CIs to violate the predefined performance yield constraint. Those which violate the constraint must be removed from the candidate set. Figure 13 depicts the impact of the random variation on the enumerated CIs by showing the percentage of the CIs which were removed from the set due to the random variation. As the results show by decreasing the performance yield the percent of the removed CIs is reduced and the numbers of the candidates are increased. Also, the results show that under different performance yields of 1, 0.95, and 0.9, on average 47.75%, 30.36%, and 28.87% of the CIs were removed, respectively.

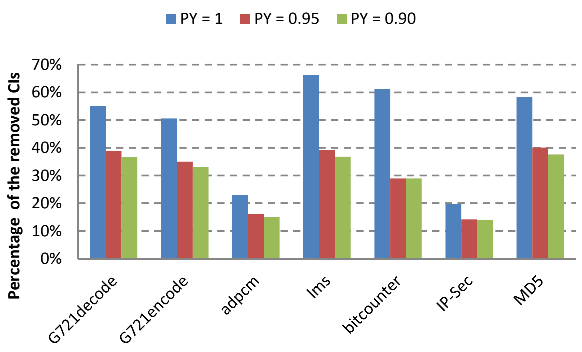


Figure 13. Percentage of the enumerated CIs which are removed from the candidate set due to the random variation.

After modeling the random variation, we need to model the systematic variation for which we need the floorplan of the primitives in a CI. As mentioned before, since this stage is time consuming, based on the method proposed in this paper, the design flow only performs floorplanning for some of the candidates. Figure 14 shows the percentage of the candidate which were selected for the floorplanning when the performance yield was 0.95. It shows that, we need to perform, on average, about 26.4% of the candidates leading to a substantial reduction in the runtime of this stage (and design flow as well).

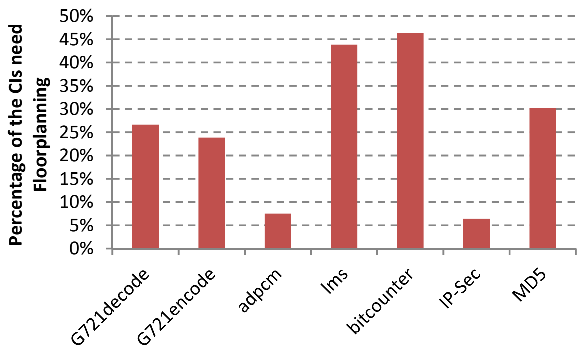


Figure 14. Percentage of the CIs which needs floorplanning.

In the last stage of the identification, based on the CI floorplan, the impact of the systematic fluctuation on the delay is modeled. Then, some CIs which violate the constraint are removed from the candidate set. Figure 15 indicates the percentage of the removed CIs in this stage. As the results show, the maximum impact of the systematic variation belongs to the *lms* benchmark where 4.17% of the candidate CIs are removed from the set. For some benchmarks, such as *adpcm* and *IP-Sec*, when the performance yield is 95%, no CIs are removed in this stage. Also, as the results show, the impact of the systematic variation is smaller for lower performance yields.

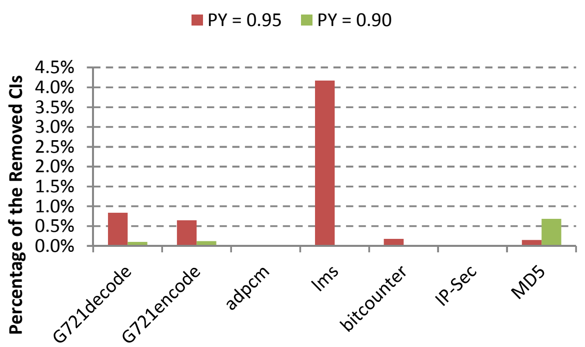


Figure 15. Percentage of the CIs which are removed from the candidate set due to the systematic fluctuation.

In Figure 16, we have plotted the distributions of the performance yield of the CIs ($\forall CI, PY_{Const} \leq PY_{CI} < 1$) before and after the systematic variation modeling. This graph shows the overall impacts of the systematic variation on the set of the candidate CIs. Hence, the comparison between the results reported in Figure 13 and Figure 15 shows that the impact of the systematic variation on the performance yield is not as high as that of the random variation. This originates from the small size of the CIs. Certainly, decreasing the highly correlated area will increase the impact of the systematic fluctuation. Figure 17 shows the percentage of the CIs which are removed from the set when the HCA is reduced to the 50 μm^2.

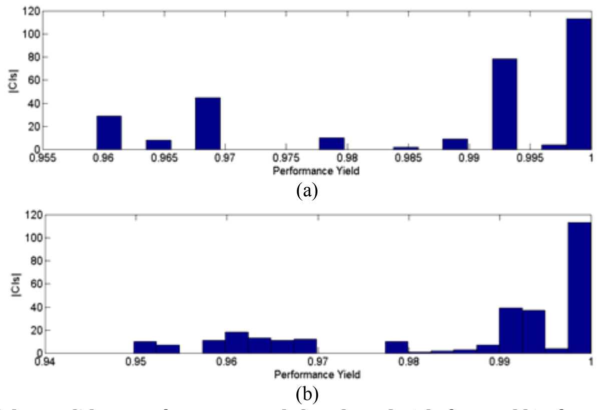


Figure 16. Performance yield of the candidate CIs for G721encode benchmark a) before and b) after modeling the systematic variation.

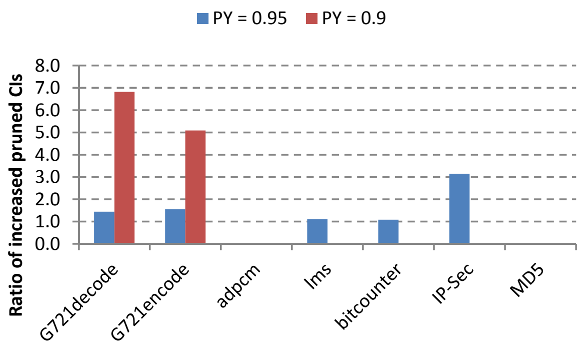


Figure 17. Impact of the HCA on the number of the pruned CIs due to the systematic variation.

2) *Selection Phase*

To study the impacts of the merit function of the greedy approach on the selection process, we studied the efficacy of the proposed merit function when both the area and performance yield constraints were considered. Table I shows the speedup of the customized processor compared to the baseline processor and also performance yield of the selected CIs (using the greedy approach) when two different merit functions were used. One merit function was based on (2) (denoted by "(2)") and the other was (8) (denoted by "(8)"). As the results show that, for most cases, the proposed merit function of (8) selected CIs, on average, about 3.2% higher speedup compared to the conventional merit function. However, as we expected, by decreasing the area constraint, the importance of the area increased giving rise to the outperformance of the conventional merit function. Specifically, in the case of small area constraint of 15% for the benchmarks *adpcm* and *MD5*, (2) led to higher speedups.

TABLE I. SELECTING THE CIS BASED ON THE PROPOSED MERIT FUNCTION AND THE CONVENTIONAL MERIT FUNCTION

Area Constraint	**PY Constraint**	**Benchmark**	**Speedup**		**Benchmark**	**Speedup**	
			(8)	**(2)**		**(8)**	**(2)**
15%	95%	**G721decode**	1.24	1.18	**adpcm**	1.23	1.24
15%	90%		1.24	1.18		1.23	1.24
15%	85%		1.24	1.18		1.23	1.24
30%	95%		1.26	1.20		1.29	1.28
30%	90%		1.26	1.20		1.30	1.29
30%	85%		1.26	1.20		1.30	1.29
50%	95%		1.28	1.21		1.29	1.28
50%	90%		1.28	1.21		1.30	1.29
50%	85%		1.28	1.22		1.30	1.29
15%	95%	**G721encode**	1.24	1.18	**MD5**	1.47	1.50
15%	90%		1.24	1.18		1.47	1.50
15%	85%		1.24	1.18		1.50	1.50
30%	95%		1.26	1.19		1.60	1.60
30%	90%		1.26	1.19		1.64	1.60
30%	85%		1.26	1.20		1.68	1.60
50%	95%		1.28	1.21		1.70	1.66
50%	90%		1.28	1.21		1.69	1.62
50%	85%		1.28	1.21		1.73	1.62

As mentioned before, the branch-and-bound technique also may be used for selecting the optimal CIs. The runtime of this technique, however, could be too high. To reduce the runtime for practical cases, one should reduce the search space using a pruning technique. In the previous section, we proposed a pruning technique based on sorting the candidate CIs. Figure 18 shows the efficacy of on the pruning technique. The results were extracted when the candidate set contained 30 of the CIs whose cycle saving were higher than the other enumerated CIs. Also, the goal was to select the five best of them under the predefined area and performance yield constraint ($Area_{Constraint}$ = 4% of the baseline processor and $PY_{Constraint}$ = 0.95%). The results show that, in the best (worst) case, the proposed technique reduced the search space about 87.5% (17.3%). On average, the search space was reduced about 62.8%.

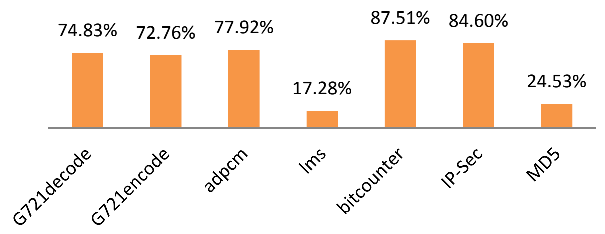


Figure 18. Percentage of the search space reduction when the pruning technique is used.

Finally, Table II shows the speedups of the extended processor when the proposed branch-and-bound ("BaB") and greedy techniques were used. Since BaB is very time consuming, we added two additional constraints. One limited the number of candidate CIs to fifty. The fifty candidates were those which had the higher cycle savings. Also, in addition, the maximum number of selected CIs was limited to fifteen. The results showed that the branch-and–bound technique yielded better CIs with higher speedups compared to the greedy approach. Also, note that due to the added constraints, in some cases, the speedups reported for the greedy approach in Table II are lower than those reported in Table I where these constraints were not considered.

TABLE II. SELECTING THE CIS BASED ON THE PROPOSED BRANCH-AND-BOUND AND GREEDY TECHNIQUES

Benchmark	Area Const.	PY Const.	Speedup BaB	Speedup Greedy
G721decode	30%	95%	1.27	1.24
	30%	90%	1.27	1.24
	30%	85%	1.27	1.26
	50%	95%	1.28	1.22
	50%	90%	1.29	1.25
	50%	85%	1.28	1.26
G721encode	30%	95%	1.27	1.23
	30%	90%	1.27	1.23
	30%	85%	1.27	1.25
	50%	95%	1.27	1.27
	50%	90%	1.30	1.27
	50%	85%	1.30	1.27
adpcm	30%	95%	1.33	1.28
	30%	90%	1.33	1.29
	30%	85%	1.33	1.30
	50%	95%	1.33	1.28
	50%	90%	1.33	1.29
	50%	85%	1.33	1.29
MD5	30%	95%	1.54	1.36
	30%	90%	1.54	1.32
	30%	85%	1.61	1.40
	50%	95%	1.54	1.46
	50%	90%	1.54	1.46
	50%	85%	1.61	1.50

VI. CONCLUSION

Hence, in this paper, we proposed a design flow technique for the ISA extension by considering the process variations. Both systematic and random variations were considered. In the proposed design flow, instead of using the conventional static timing analysis (STA), statistical static timing analysis (SSTA) had to be used. Hence, a probabilistic approach was employed to identify and select CIs in the extensible processor design flow. In the identification phase, the probability density function (PDF) of the delay of each CI due to systematic and random variations was modeled and the performance yield was added as a constraint. The effects of systematic variation were estimated based on the floorplan of CIs. To make the approach computationally more efficient, we proposed a method for reducing the modeling time of the systematic variation by reducing the number of candidate CIs for the floorplanning. The results showed that the proposed method reduced the number of the CIs for floorplanning by about 24.6%. The reduction in the number CIs was considerable revealing the efficacy of the method for reducing the modeling time. In the selection phase, both greedy and branch-and-bound approaches were used. In the greedy approach, the merit function of the conventional approach was modified to increase the speedup of the extended processor without violating the area and performance yield constraints. In most cases, the results showed that the proposed merit function reached higher performances in comparison to the conventional approach. On average, the proposed merit function led to a higher speedup of 3.2%. Finally, in the branch-and-bound approach, we proposed an effective pruning technique to reduce its execution time. The proposed technique was able to reduce the search space of the benchmarks considerably (~62%) which led to the reduction of execution time. This showed that the use of an effective pruning technique could make the branch-and-bound approach a more practical optimization technique to obtain better CIs for higher speedups. It should be noted that as the technology size shrinks, the uncertainty in the values of the physical parameters will increase, and hence, the speedup improvements of the proposed technique increases further. In addition, the increase mandates the use of statistical design approaches such as the one proposed in this paper. As a future work, one may consider the leakage power of the extended ISA as a parameter in the statistical design flow proposed in this work. In addition, including the effect of time-dependent variations (such as negative bias temperature instability) in the design of ASIPs could be another research area of importance.

REFERENCE

[1] L. Pozzi, K. Atasu, and P. Ienne, “Exact and Approximate Algorithms for the Extension of Embedded Processor Instruction Sets”, in *IEEE Transactions on Computer-Aided Design*, vol. 25, no. 7, July 2006.

[2] P. Bozini and L. Pozzi,“Recurrence-Aware Instruction Set Selection for Extensible Embedded Processors,” in *IEEE Transactions on Very Larg Scale Integration(VLSI) Systems*, vol. 16, pp. 1259- 1267, 2008.

[3] K. Atasu, L. Pozzi, and P. Ienne, “Automatic application-specific instruction-set extensions under microarchitectural constraints,” in *Proeedings of 40th Design Automation Conference*, Jun. 2003, pp. 256–261.

[4] C. Galuzzi, and K. Bertels,”The Instruction-Set Extenstion Problem: A Survay”, in *ACM Trasactions on Reconfigurable Technology and Systems*, vol. 4, no. 2, pp. 18:1-18:28, 2011.

[5] N.T. Clark, H. Zhong, and S. A. Mahlke, “Automated Custom Instruction Generation for Domain-Specific Processor Acceleration,” *in IEEE Transactions on Computers*, vol. 54, pp. 1258-1270, 2005.

[6] Y. Xie and Y. Chen,“Statistical High-Level Synthesis under Process Variability,“ in *IEEE Design and Test Computers*, Vol. 26, pp.78-87, 2009.

[7] K. A. Bowman, S.G. Duvall, and J. Meindl, “Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration,” in *IEEE Jornal of Solid-State Circuits*, , vol. 37, pp. 183-190, 2002.

[8] International Technology Roadmap for Semiconductors, 2007. [online]. Available: http://public.itrs.net/

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

[10] X. Liang and D. Brooks, “Mitigating the impact of process variations on processor register files and execution units,” in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 504-514.

[11] F. Wang, Y. Xie, and A. Takach, “Variation-Aware Resource Sharing and Binding in Behavioral Synthesis,” in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, 2009, pp. 79-84.

[12] F. Wang, G. Sun, and Y. Xie, “A Variation Aware High Level Synthesis Framework,” in *Proceedings of the conference on Design, automation and test in Europe(DATE)*, 2008, pp. 1063-1068.

[13] 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.

[14] K. Atasu et al. “An integer linear programming approach for identifying instruction-set extensions,” in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis(CODES+ISSS)*, 2005, pp. 172-177.

[15] K. Verma, P. Brisk, and P. Ienne, “Rethinking custom ISE identification: A new processor-agnostic method,” In *Proceedings of international conference on Compilers, architecture, and synthesis for embedded systems (CASES)*, Austria, 2007,pp. 125–134.

[16] K. Atasu, O. Mencer, W. Luk, C. Ozturan, and G. Dundar, “Fast custom instruction identification by convex subgraph enumeration,” In *Proceedings of the 2008 International Conference on Application-Specific Systems, Architectures and Processors*, USA, 2008, pp. 1–6.

[17] T. Li, Z. Sun, W. Jigang, and X. Lu," Fast Enumeration of Maximal Valid Subgraphs for Custom-instruction Identification," in *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems(CASES)* , 2009, pp. 29-36.

[18] M. Kamal, N. Kazemian Amiri, A. Kamran, S.A. Hoseini, M. Dehyadegari, and H. Noori, "Dual-Purpose Custom Instruction Identification Algorithm based on Particle Swarm Optimization," in *Proceeding of 21st IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP10)*, France, 2010, pp. 159-166.

[19] N. Clark, A. Hormati, S. Mahlke, and S. Yehia, "Scalable subgraph mapping for acyclic computation accelerators," *in Proceedings of international conference on Compilers, architecture and synthesis for embedded systems(CASES)* , 2006, pp. 147-157.

[20] 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.

[21] C. Kenyon et al., ''Managing Process Variation in Intel's 45nm CMOS Technology,'' in *Intel Technology Jornal*, vol. 12,no. 2, 2008.

[22] A. Agarwal, et al., "Path-based statistical timing analysis considering inter-and intra-die correlations," in *Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems*, 2002, pp. 16–21.

[23] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Mitigating parameter variation with dynamic fine-grain body biasing," in *Proceeding of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007, pp. 27-42.

[24] 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.

[25] Y.S. Lu, L. Shen, L.B. Huang, Z.Y. Wang, and N. Xiao, "Optimal subgraph covering for customisable VLIW processors," in *Computer and Digital Techniques*, vol. 3, pp. 14-23, 2009.

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

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

[28] 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 International workshop on workload characterization*, 2001, pp. 3-14.

[29] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, " MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, " in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, 1997, pp. 330-335.

[30] 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.