

Variation-Aware Resource Sharing and Binding in Behavioral Synthesis

Feng Wang
Qualcomm Inc.
San Diego, CA 92121
fwang@qualcomm.com

Yuan Xie
Pennsylvania State University
University Park, PA 16802
yuanxie@cse.psu.edu

Andres Takach
Mentor Graphics Corporation
Wilsonville, OR 97070
andres.takach@mentor.com

Abstract—As technology scales, the delay uncertainty caused by process variations has become increasingly pronounced in deep submicron designs. In the presence of process variations, *worst-case* timing analysis may lead to overly conservative synthesis, and may end up using excess resources to guarantee design constraints. In this paper, we propose an *efficient* variation-aware resource sharing and binding algorithm in behavioral synthesis, which takes into account the performance variations for functional units. The *performance yield*, which is defined as the probability that the synthesized hardware meets the target performance constraints, is used to evaluate the synthesis result. An efficient metric called *statistical performance improvement*, is used to guide resource sharing and binding. The proposed algorithm is evaluated within a commercial synthesis framework that generates optimized RTL netlists from behavioral specifications. The effectiveness of the proposed algorithm is demonstrated with a set of industrial benchmark designs, which consist of blocks that are commonly used in wireless and image processing applications. The experimental results show that our method achieves an average 33% area reduction over traditional methods, which are based on the worst-case delay analysis, with an average 10% run time overhead.¹

I. INTRODUCTION

Hand coded register-transfer level (RTL) is widely used in the hardware design of complex systems. However, as the complexity of the system grows, this labor intensive approach is both time consuming and error prone. High-level synthesis (HLS) automates the process of generating RTL implementations from behavioral descriptions [1] [2]. Automation also enables a wider exploration of the design space and has been reported to provide around five times speed-up of design time as compared to manual RTL methodologies [3]. Commercial HLS tools have been introduced in the market and are gaining adoption in industry, as part of an effort to move design to higher levels of abstraction in what is referred to as *Electronic System Level (ESL)* design. Process variability further complicates manual development of RTL. Developing an automated RTL creation methodology that takes into account process variability becomes of primary importance as technologies shrink below 65nm.

High-level synthesis consists of three steps: scheduling, resource sharing, and resource binding. Traditionally, all these steps are performed based on deterministic worst-case performance analysis. However, technology scaling has resulted in significant variations in transistor parameters, such as transistor channel length, gate-oxide thickness, and threshold voltage. This manufacturing variability can cause

significant performance deviations from nominal values in identical hardware designs. For example, IBM has shown that the worst case performance at the 45nm technology node is very close to the nominal performance at the 65nm technology node [4]. Although designing for worst-case process margins is the traditional approach to deal with outliers, the degree of variability encountered in the new process technologies makes this option infeasible. Deterministic worst-case performance analysis, which does not take into account the process variations, can result in pessimistic estimation in terms of performance and end up using excess resources to guarantee design constraints.

In this paper, variation-aware performance analysis is integrated into the HLS process to cope with process variations. In addition to variation-aware performance analysis, we introduce a new concept called *statistical performance improvement*, which is equivalent to the performance yield improvement, to efficiently and effectively guide the design space exploration during resource sharing and binding. Experimental results clearly indicate that the proposed variation-aware approach in resource sharing and binding can lead to significant area cost saving.

II. RELATED WORK

Related work pertaining to this paper can be divided into three different categories: traditional high-level synthesis approaches, statistical timing analysis and optimization, and recent developments in variation aware high-level synthesis.

Extensive research has been done on high-level synthesis for over two decades [1] [2]. Research in high-level synthesis has focused on the following core steps: scheduling, resource sharing, and resource binding. Scheduling is an NP-complete problem [1]. Consequently, scheduling algorithms are in general based on heuristics to guide how operations are scheduled, even though formulations based on ILP are also proposed [5], for example, many scheduling approaches are based on some variation of list-scheduling with heuristics to guide how operations are scheduled based on their *urgency* [6], their *mobility* [7], or by attempting to balance their distribution [8] [9]. A number of resource sharing and binding techniques, such as clique partitioning algorithm [10] and the *LEFT_EDGE* algorithm [11], have been explored. Recently, a number of high-level synthesis techniques have been proposed to reduce the power, the temperature and interconnect delays [12]–[14] [15]. However, all these approaches are developed based on deterministic worst-case timing analysis.

¹This research was supported in part by NSF grants of CAREER 0643902, CNS 0720659 and CCF 0702617, and a grant from SRC

Recently, research on variation aware analysis and optimization techniques has received great attention from both academia and industry. Various techniques have been proposed for statistical timing analysis, such as path-based approaches and block based approaches [16]. Based on statistical timing analysis, optimization techniques, ranging from gate sizing to buffer insertion, have been explored. However, most of these techniques fall into either gate level approaches or device level approaches.

The impact of process variability can be effectively reduced if it is considered from the very early stage of the design [17]. Although we have seen some very recent work considering the impact of process variations at the architectural and system level [18]–[20], high-level synthesis research related to process variations is still in its infancy. Recently, Hung et al. [21] proposed a *simulated-annealing* based HLS framework to take into account process variations. However, statistical timing information and yield analysis results are not used to guide the design exploration during synthesis, which may lead to suboptimal solutions. Jung et al. [22] attempted to use statistical timing information to perform high-level synthesis based on heuristics, which are only partly true. For example, one heuristic is that resource sharing always results in higher yield. These two works [21] [22] represent the first attempts to incorporate the timing variability due to process variations into high level synthesis.

Our research in this paper distinguishes itself in the following aspects: 1) we first introduce two new concepts, the statistical path delay and the criticality, to compute the statistical performance improvement, which is used to facilitate the design space exploration during resource allocation and assignment; 2) we subsequently implement the algorithm and benchmark it in the framework of a commercial HLS tool, and validate our proposed techniques with industrial design examples.

III. PRELIMINARIES AND PROBLEM FORMULATION

A. High-level Synthesis

In HLS, each operation (such as an addition or a multiplication) in the CDFG is scheduled to one or more cycles (or control steps). Each control step corresponds to a time interval that is equal to the clock period. Each operation may be performed by more than one *compatible* resource type from the resource library. For example, the addition operation can be performed by either a ripple-carry adder or a carry look-ahead adder, which have different delay and area parameters. *Resource binding* decides the type of functional units to perform the operation in the CDFG. *Resource sharing* is the assignment of the same resource (functional units or registers) to perform multiple operations or store more than one variable. Traditionally, high-level synthesis is performed under design constraints, which includes resource constraints, and performance constraints: the *resource constraints* require that the operations are performed with only a limited number of resources available; the *performance constraints* require that the operations in the CDFG finish the execution in a number of clock cycles (*latency constraints*) with a particular clock rate (*clock cycle time(CCT) constraints*).

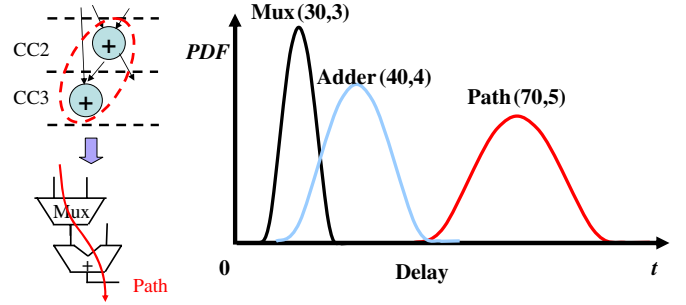


Fig. 1. An example of resource sharing for two addition operations and the comparison of worst-case execution time (WCET) based and statistical analysis based approaches. Two addition operations are mapped to an adder, and an additional multiplexer is required. The delay distributions of the adder and the multiplexer are shown as PDF (probability distribution function). The delay distribution of the critical path after resource sharing is also shown as PDF.

B. The Impact of Process Variation

Traditionally, worst-case delay parameters for each resource are used in the resource sharing and binding algorithms. However, it is becoming inappropriate as larger variability is encountered in new process technologies. A recent publication [23] reports that the delay variations range from 12% to 27%, for 11 different types of 16-bit adders with different circuit architectures and logic evaluation styles. Some adders may run faster but with large variations (for example, Kogge Stone Passgate adder has -27% to $+27\%$ of 3σ delay variation), and some adders may run slower but more resistant to variations (such as Carry Select Static).

Due to large variations in delay, the existing deterministic worst-case design methodologies in HLS may result in unexpected performance discrepancy or a pessimistic performance estimation, or may end up using excess resources to guarantee design constraints, due to overly conservative design approaches. We illustrate this with an example of *resource sharing* shown in Fig. 1. Assume that the delay of an adder (D_{add}) and a multiplexer (D_{mux}) conform to independent Gaussian distributions $N(\mu, \sigma)$, and the delay distributions of these two function units are given, $D_{add} = (40ps, 4ps)$ and $D_{mux} = (30ps, 3ps)$, and the clock cycle time is $87ps$. In conventional worst-case analysis, the worst-case execution time (WCET) is calculated as $\mu + 3\sigma$. Assuming that we have two compatible addition operations, which can share the same adder, the worst-case timing analysis for the path delay after resource sharing is $D_{path_WCET} = \mu_{D_{add}} + \mu_{D_{mux}} + 3(\sigma_{D_{add}} + \sigma_{D_{mux}})$, which is $91ps$. Consequently, the worst-case analysis prevents such resource sharing, because the path delay violates the clock cycle time constraint. However, based on the statistical information, the delay after resource sharing D_{path_ST} follows $N(\mu_{D_{add}} + \mu_{D_{mux}}, \sigma_{D_{add}}^2 + \sigma_{D_{mux}}^2)$, and the 3σ delay of D_{path_ST} is $85ps$. Thus, the statistical analysis allows the resource sharing to reduce area cost. Therefore, simply adding the WCET of two function units can result in a pessimistic estimation of the total delay, and may end up using excess resources to guarantee performance constraints.

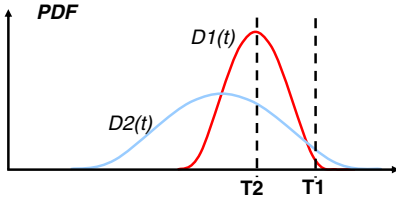


Fig. 2. An example illustrates the effectiveness of the performance yield metric. The critical path delay distributions of two synthesis resultant hardware are shown in PDF. When clock cycle time is set to T_1 , the synthesis result with $D_1(t)$ is better than that with $D_2(t)$ in terms of performance yield. However, when clock cycle time is set to T_2 , the synthesis result with $D_2(t)$ is better than that with $D_1(t)$. However, if we use worst-case delay to evaluate the results, we always choose the synthesis result with $D_1(t)$.

C. Performance Yield as an Effective Evaluation Metric

With large delay variations, it is unrealistic to guarantee 100% of the fabricated design to meet the performance constraint. A metric, called *parametric yield*, is introduced to bring the process-variation awareness to the high-level synthesis flow. The parametric yield is defined as the probability of the HLS resultant hardware meeting a specified constraint $Yield = P(Y \leq Y_{max})$, where Y represents performance. The *performance yield* is defined as the probability of the synthesis results meeting the clock cycle time constraints under the latency constraints and resource constraints.

Traditionally, the performance constraints are evaluated with the critical path delay. Under large process variations, the critical path delay is not a single value and it becomes a distribution. Thus, the yield metric is used to evaluate the results from synthesis. Fig. 2 demonstrates the effectiveness of the performance yield metric even for a simple example. Assume that we have two synthesized results with the critical path delay distribution of $D_1(t)$ and $D_2(t)$, respectively. When clock cycle time is set to T_1 , the synthesis result with $D_1(t)$ is better than that with $D_2(t)$ in terms of performance yield. However, when clock cycle time is set to T_2 , the synthesis result with $D_2(t)$ is better than that with $D_1(t)$. However, if we use worst-case delay to evaluate the results, we always choose the synthesis result with $D_1(t)$.

The detailed performance yield analysis for a synthesized DFG is described in Section IV and V.

D. Statistical Performance Improvement as an Effective Metric to Guide Optimization

In this paper, we introduce two concepts, the *statistical path delay improvement* and the *criticality* to effectively guide the optimization in resource sharing and binding. The statistical path delay represents the magnitude of the path delay based on the statistical analysis. This statistical path delay avoids the pessimistic nature of the worst-case path delay based on the worst-case analysis. The statistical path delay improvement is the difference between the statistical path delay before and after resource sharing or binding. The criticality was first introduced in gate level statistical timing analysis [16]. In this work, we extend this concept to high-level synthesis. The criticality is defined as the probability of the operation being on the critical path. This metric represents the characteristics of the path delay distribution.

These two concepts combined together are able to capture the performance yield improvement due to resource sharing

Cases	Path Delay Improvement	Criticality	Yield Improvement
Case1	small	small	small
Case2	large	small	medium
Case3	small	large	medium
Case4	large	large	large

Fig. 3. Accurately evaluating the performance improvement requires to consider both the path delay improvement and the criticality. Four cases are shown with different path delay improvements and values of the criticality. For case 1 and case 4, it is obvious that the path delay improvement and the criticality can represent the performance yield improvement. In case 2 and case 3, one metric, i.e. either the path delay improvement or the criticality, is not sufficient to represent the performance yield improvement.

or binding. We illustrate this through an example shown in Fig. 3. Four cases are shown with different statistical path delay improvements and values of the criticality. For case 1 and case 4, it is obvious that either the path delay improvement or the criticality can represent the performance yield improvement. In case 2 and case 3, one metric, i.e. either the path delay improvement or the criticality is not sufficient to represent the performance yield improvement. In case 2, the statistical path improvement is large, and there is no difference between case 2 and case 4 if we only use the statistical path improvement as a metric. Similarly, if we use the criticality alone, case 3 would be treated as the same as case 4. Thus, we define the *statistical performance improvement* as the function of the path delay improvement and the criticality to effectively represent the performance yield improvement. In Section V, the computation method of this metric will be presented.

E. Variation Aware Resource Sharing and Binding

It is obvious that the parametric yield depends on all steps of high-level synthesis: scheduling, resource sharing, and resource binding. These steps usually interact with each other during high-level synthesis, and influence the final parametric yield calculations. However, the resource sharing and binding have direct impact on the performance yield. Thus, in this work, we focus on the variation aware resource sharing and binding. The variation aware resource sharing and binding is formulated as: *Given a scheduled data flow graph, a set of design constraints, and a resource library with statistical delay characterization, determine the type of function units to perform the operations in CDFG, to reduce the area while satisfying all the design constraints, which include the performance yield constraint.*

IV. STATISTICAL ANALYSIS FOR DFG

In this section, we first briefly describe our statistical timing analysis for a synthesized DFG [14] (in which all operations have been scheduled and bound to function units selected from the resource library). We then discuss the parametric yield analysis method for a synthesized DFG.

In the statistical timing analysis for a synthesized DFG, the timing quantity is computed by using two atomic functions $sum(A, B)$ and $max(A, B)$, where A and B are random variables. The sum and max operations are computed as follows:

- 1) The sum operation is easy to perform. For example, if A and B both follow a Gaussian distribution, the distribution of $sum(A, B)$ follows a Gaussian distribution with a mean of $\mu_A + \mu_B$ and a variance

of $\sqrt{\sigma_A^2 + \sigma_B^2 - 2\rho\sigma_A\sigma_B}$, ρ is correlation coefficient between A and B .

- 2) The *max* operation is quite complex. Tightness probability [16] and moment matching [24] techniques could be used to determine the corresponding sensitivities to the process parameters. Given two random variables, A and B , tightness probability of random variable A is defined as the probability of A being larger than B . An analytical equation in [24] to compute the tightness probability is used to facilitate the calculation of the *max* operation.

The delay distribution of module instances can be obtained through gate-level statistical timing analysis tools [25] [16] or Monte Carlo analysis in HSPICE. With the atomic operations defined, the timing analysis for the synthesized DFG can be conducted using PERT-like traversal [25].

In a synthesized DFG, the operations are distributed to the clock cycles and bound to function units selected from the resource library. In this work, we compute the performance yield based on the synthesis result, which is a circuit consisting of registers and combinational logic gates, including multiplexers that enable resource sharing. As in gate-level yield analysis, the performance yield is calculated as the probability that the design meets the clock cycle time constraints where timing is computed using the statistical timing analysis described earlier in this section.

V. PROCESS VARIATION AWARE RESOURCE SHARING AND BINDING

In this section, we first present the computation method for the statistical performance improvement in resource sharing and binding. Based on this new performance improvement metric, a process variation aware resource sharing and binding algorithm is then proposed.

A. Statistical Performance Improvement Computation for Resource Reallocation and Reassignment

When resource reallocation and reassignment occur, the distribution of the path delay changes, which include the changes in both the mean and the variance of the path delay. The statistical path delay is calculated as $\mu_{path_delay} + \alpha * \sigma_{path_delay}$, where α is a weighting factor, which balances the optimization effort on reducing the mean and the spread of the performance distribution. The statistical path delay represents the magnitude of the path delay. This metric eliminates the pessimistic nature of the deterministic worst case path delay by computing the value based on the statistical performance analysis.

However, this metric alone is not sufficient to capture the distribution characteristics of the performance improvement as discussed in Section III-D. Thus, the criticality concept is introduced. The criticality is computed as the probability of the delay distribution of the function units involved in the resource reallocation and rebinding determining the critical path delay.

With the statistical delay improvement and the criticality determined, the statistical performance improvement metric is computed as

$$\Delta st_perf_impr = st_pathdelay_impr * criticality \quad (1)$$

where Δst_perf_impr represents the statistical performance improvement, and $st_pathdelay_impr$ represents the statistical path delay improvement.

B. Gain Function for Resource Reallocation and Reassignment

With the statistical performance improvement metric determined, the gain function for the resource reallocation and reassignment can be computed. The gain function for performance improvement under resource constraints and other constraints is computed as $\Delta st_perf_impr / \Delta area_cost$; the gain function for area cost reduction under performance yield constraint and other constraints is computed as $\Delta area_cost / \Delta st_perf_impr$. The computation of $\Delta area_cost$ is relatively simple and straightforward. For resource reassignment, the change of area cost is simply the area difference of the function units involved in the reassignment. For resource reallocation, the change of the area cost is the area difference of the function units plus the area cost of the additional multiplexers required.

C. Variation-aware Resource Sharing and Binding Algorithm

```

Optimization (synthesizedDFG, constraints, Library){
1. While (meet constraints){
2.   Generate_multiple_moves generates the to_move_list list;
3.   Apply_multiple_moves applies multiple moves in to_move_list;
4. } }

Apply_multiple_moves (to_move_list, constraints){
5. While (to_move_list is not empty and meet constraints){
6.   For each resource_sharing or rebinding in that list{
7.     Insert new move to to_update_list;
8.     Update Yield with new move in to_update_list;
9.   } } }

```

Fig. 4. The Pseudo Code of Variation Aware Optimization of DFG

```

Generate_multiple_moves (synthesizedDFG, Library, constraints){
1. Build the compatible graph;
2. For each edge in that compatible graph{
3.   Compute gain function for the possible resource sharing;
4. }
5. Rank the resource_sharing in that list according to gain value;
6. }

```

Fig. 5. The Pseudo Code of Generate Moves for Resource Sharing of DFG

```

Generate_multiple_moves (synthesizedDFG, Library, constraints){
1. For each operation{
2.   Compute gain function for different possible binding;
3. }
4. Rank the binding in that list according to gain value;
5. }

```

Fig. 6. The Pseudo Code of Generate Moves for Resource Binding of DFG

Our variation aware resource sharing and binding algorithm is integrated within Catapult C Synthesis [26], a commercial high-level synthesis tool that generates optimized RTL from algorithmic specifications written in C/C++. This high-level synthesis framework consists of four steps: first, the description of the algorithm in C++ to be synthesized is translated into a CDFG; next the operations of the CDFG are scheduled with the *constraints*, which include latency constraint, clock cycle time constraint, and other user specified constraints; then resource sharing and binding are performed, finally the RTL netlist is generated. In this work, we augment

the third step to be process variation aware. Thus, our method takes a scheduled *DFG*, *constraints* (latency constraint, resource constraint, and clock cycle time constraint), and a module *Library* as inputs, and outputs a synthesized *DFG* that is either performance optimized or area optimized while satisfying those constraints. Note that the scheduled *DFG* (in which all operations have been scheduled and bound to function units selected from the resource library) meets the latency constraints in terms of the number of clock cycles.

The optimization algorithm can be configured as *performance optimization* or *area optimization*: 1) For performance optimization, the resource constraints is given as an additional constraint, and *the performance yield is maximized*; 2) For area optimization, the performance yield requirement (e.g., the probability of the synthesis result can run at 500 Mhz should be at least 95%) is given as an additional constraint, and *the area is minimized*.

For the sake of simplicity, we describe the *area optimization* algorithm in Fig. 4. Our resource sharing and assignment algorithm consists of two steps: 1) resource sharing; 2) resource binding to minimize area under the performance yield constraint. At the first step, we run the *Optimization* routine by finding the possible resource sharing for the compatible operations; at the second step, we run the *Optimization* routine to map the operations to the resources to reduce the area cost. Both optimization steps use $\Delta area_cost / \Delta performance_improvement$ as the gain function. Detailed description is shown as follows:

(1) At each optimization step, we first generate the possible moves, which could be resource sharing or resource binding. We identify possible moves, and form the *to_move_list* at Line 2; We then choose the resource sharing or binding from that list to minimize the area with least performance overhead, and apply these resource sharing or binding at Line 3. The optimization will be terminated when the performance yield constraint is violated.

(2) In function *Generate_multiple_moves*, we compute the gain (i.e. $\Delta area_cost / \Delta st_perf_impr$) for each possible move. The move can be resource sharing or resource binding. For resource sharing as shown in Fig 5, we first build a compatibility graph at Line 1. Then we compute the gain of the possible resource sharing as $\Delta area_cost / \Delta st_perf_impr$ from Line 2 to Line 4. For resource binding as shown in Fig. 6, we identify the available function units for each operation and compute the gain of the rebinding as $\Delta area_cost / \Delta st_perf_impr$ at Line 2. The possible resource sharing or resource binding are ranked according to the gain and penalty ratio, and are inserted to the *to_move_list* at Line 9.

(3) In the function *Apply_multiple_moves*, we select the multiple resource sharing or binding from that list to maximize the yield (Line 7) and apply these resource sharing or binding decisions (Line 8).

VI. RESULTS ANALYSIS

In this section, we present the analysis results and show that our method can effectively reduce the impact of the process variation and minimize the area cost under the per-

formance yield constraint, as compared to the non-statistical method based on traditional worst-case performance analysis.

We implemented our variation aware resource sharing and binding algorithm within Catapult C Synthesis [26]. We conducted the experiments on twelve industrial design examples commonly used in wireless and image processing applications. The first ten designs are blocks from the IEEE 802.16 WiMAX standard: 1) Convolutional Coder (CC), 2) Deinterleaver (Deint), 3) Derandomizer (Derand) 4) Hamming windowing and 256-point FFT (RX-FFT), 5) 256-point IFFT and OFDM cyclic prefix insertion (IFFT-CPI), 6) QAM Mapper (QAMM), 7) QAM Slicer (QAMS), 8) Randomizer (Rand), 9) Reed Solomon encoder (RSEnc) and 10) Viterbi decoder (VitDec). The remaining two blocks implement an 8x8 DCT (SDCT) and an 8-tap FIR filter (SFIR). The set of benchmarks are part of the reference design toolkit that comes with Catapult C Synthesis. Since our focus was resource sharing and binding, we leveraged the tool for the remaining synthesis steps such as allocation/scheduling, register sharing, and datapath/controller generation. The sharing and binding that we use as reference for the comparisons are the ones performed by the tool. In this paper, we show the *area optimization* oriented result: i.e., the performance yield requirement is given as a constraint, and the area is minimized.

To demonstrate the effectiveness of our method, we compare our high-level synthesis results with process-variation aware (PV) resource sharing and binding against those of traditional deterministic methods using worst case (WC) delay models. Table I shows the results of our method (PV) against those of the worst case (WC) deterministic technique, under a 95% performance yield requirement. In the first column, we show the benchmarks we used in this analysis. From the second column to the third column, we show the area reduction results of the process variation aware method (PV) and the deterministic worst case technique (WC), respectively. In the fourth column, we show the relative area improvement of our method over the worst case method. As we can see from Table I, significant area cost reductions are obtained when process variation is taken into account in resource sharing and binding. The results show that the WC based technique is pessimistic with an average 33% area penalty. The worst-case analysis results in a pessimistic estimation, as explained in Section III-B, and the consequence is sharing opportunities that are not leveraged to guarantee performance constraints. Of course, if we tighten the performance yield constraint, we may obtain less area reduction. For example, Table II shows that for the same 99% performance yield constraint, the area cost is slightly increased compared to the results for 95% performance yield constraint (Table I).

Table III shows the runtime overhead of our method (PV) over traditional worst case method (WC) for different benchmarks. From the second column to the third column, we show the runtime of the process variation aware method (PV) and the deterministic worst case technique (WC), respectively. In the fourth column, we show the runtime overhead of our method over the worst case method. The maximum runtime overhead of the proposed method is 21%. The average

TABLE I

AREA REDUCTION UNDER 95% PERFORMANCE YIELD CONSTRAINT

Name	WC (μm^2)	PV (μm^2)	(PV-WC)/WC
CC	3731.675	5023.09	35%
Deint	6411.245	7514.372	17%
Derand	4636.388	5336.757	15%
RX-FFT	80390.443	101822.946	27%
IFFT-CPI	108258.755	144237.118	33%
QAMM	2856.15	7108.83	149%
QAMS	6515.371	6570.33	1%
Rand	6926.769	8102.694	17%
RSEnc	7992.56	9072.529	14%
SDCT	83482.4	95031.681	14%
SFIR	5402.935	6178.34	14%
VitDec	108880.481	172757.682	59%
Average			33%

TABLE II

AREA REDUCTION UNDER 99% PERFORMANCE YIELD CONSTRAINT

Name	WC (μm^2)	PV (μm^2)	(PV-WC)/WC
CC	3795.851	5023.09	32%
Deint	6660.507	7514.372	13%
Derand	4636.388	5336.757	15%
RX-FFT	83510.638	101822.946	22%
IFFT-CPI	124925.961	144237.118	15%
QAMM	2868.197	7108.83	148%
QAMS	6515.371	6570.33	1%
Rand	6939.646	8102.694	17%
RSEnc	8029.941	9072.529	13%
SDCT	83482.4	95031.681	14%
SFIR	5416.867	6178.34	14%
VitDec	110481.6	172757.682	56%
Average			30%

TABLE III

RUN TIME OVERHEAD OF VARIATION AWARE METHOD

Name	WC(s)	PV(s)	(PV-WC)/WC
CC	25	29	16%
Deint	11	12	9%
Derand	51	51	0%
RX-FFT	77	87	13%
IFFT-CPI	90	100	11%
QAMM	11	12	9%
Rand	41	45	10%
RSEnc	50	53	6%
SDCT	92	101	10%
SFIR	14	14	0%
VitDec	738	891	21%
Average			10%

runtime overhead is 10% compared to the the traditional deterministic method.

VII. CONCLUSIONS

In this paper, we formulate the performance yield constraint resource sharing and binding problem and propose an efficient algorithm to solve it. We introduce an efficient metric called *statistical performance improvement* to facilitate the design space exploration in the resource sharing and binding. Based on this metric, the performance yield aware resource sharing and binding algorithm is developed and evaluated within a commercial tool flow. Empirical results show that significant area cost reduction can be obtained with our variation-aware resource sharing and binding method with small amount of runtime overhead.

REFERENCES

- [1] A. Raghunathan, N. K. Jha, and S. Dey. *High-level power analysis and optimization*. Kluwer Academic Publishers, 1998.
- [2] D. Gajski, N. Dutt, and A. Wu. *High-level synthesis: Introduction to chip and system design*. Kluwer Academic Publishers, 1992.
- [3] Tohru Furuyama. Keynote speech: CHALLENGES OF DIGITAL CONSUMER AND MOBILE SoCs: MORE MOORE POSSIBLE? *DATE Conference*, April 2007.
- [4] P. Hurat, Y.-T. Wang, and N. K. Verghese. Sub-90 nanometer variability is here to stay. *EDA Tech Forum*, November 2005.
- [5] Cheng-Tsung Hwang, Jiahn-Hung Lee, and Yu-Chin Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-10(4):464–475, 1991.
- [6] E. F. Girzyc, R. J. Buhr, and J. P. Knight. Applicability of a subset of ada as an algorithmic hardware description language for graph based hardware compilation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(2):134–142, 1985.
- [7] B.M. Pangrle. *A Behavioral Compiler for Intelligent Silicon Compilation*. PhD thesis, University of Illinois at Urbana-Champaign, 1987.
- [8] P. Paulin and J. Knight. Force-directed scheduling for the behavioral synthesis of asic's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 661–679, 1989.
- [9] H. Kramer and W. Rosenstiel. System synthesis using behavioural descriptions. In *Proceedings, European Design Automaton Conference*, pages 277–282. IEEE Computer Society Press, 1990.
- [10] C. J. Tseng and D. P. Siewiorek. Automated synthesis of data paths in digital systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(3):379–395, July 1986.
- [11] F. Kurdahi and A. Parker. Real: A program for register allocation. In *Proceedings, 24th Design Automation Conference*, pages 210–215. ACM Press, 1987.
- [12] E. Kursun, A. Srivastava, S. G. Memik, and M. Sarrafzadeh. Early evaluation techniques for low power binding. In *International Symposium on Low Power Electronics and Design*, pages 160–165, 2002.
- [13] C.-G. Lyuh and T. Kim. High-level synthesis for low power based on network flow method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(3):364–375, 2003. 1063-8210.
- [14] X. Tang, H. Zhou, and P. Banerjee. Leakage power optimization with dual-vth library in high-level synthesis. In *Design automation conference*, pages 202–207, 2005.
- [15] R. Mukherjee, S. Ogrenci Memik, and G. Memik. Temperature-aware resource allocation and binding in high-level synthesis. In *Design Automation Conference*, pages 196–201, 2005.
- [16] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. *Design Automation Conference (DAC)*, pages 331–336, June 2004.
- [17] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.
- [18] N. Kim, K. Taeho, K. Bowman, V. De, and T. Mudge. Total power-optimal pipelining and parallel processing under process variations in nanometer technology. In *International Conference on Computer Aided Design*, pages 535–540, 2005.
- [19] D. Marculescu and E. Talpes. Variability and energy awareness: A microarchitecture-level perspective. In *Design Automatin Conference*, pages 11–16, 2005.
- [20] X. Liang and D. Brooks. Performance optimal micro-architecture parameters selection under the impact of process variation. In *International Conference on Computer Aided Design*, 2006.
- [21] W.-L. Hung, X. Wu, and Y. Xie. Guarantee performance yield in high-level synthesis. In *International Conference on Computer Aids Design*, 2006.
- [22] Jongyoon Jung and Taewhan Kim. Timing Variation-Aware High-Level Synthesis. *ICCAD*, November 2007.
- [23] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM J. Res. Dev.*, 50(4/5):433–449, 2006.
- [24] C. Clark. The greatest of a finite set of random variables. *Operations Research*, pages 145–162, 1961.
- [25] S. Sapatnekar. *Timing*. Kluwer Academic Publishers, 2004.
- [26] Catapult C synthesis. Technical report, Mentor Graphics Corporation, Products Overview.