Timing Variation-Aware High-Level Synthesis

Jongyoon Jung
School of Electrical Engineering and Computer Science
Seoul National University, Korea
bellrich@ssl.snu.ac.kr

Taewhan Kim
School of Electrical Engineering and Computer Science
Seoul National University, Korea
tkim@ssl.snu.ac.kr

Abstract*-The timing closure problem is one of the most important problems in the design automation. However, the rapid increase of the impact of the process variation on circuit timing makes the problem much more complicated and unpredictable to tackle in synthesis. This work addresses a new problem of high-level synthesis (HLS) that effectively takes into account the timing variation. Specifically, the work addresses the following four problems: (1) *how can the statistical static timing analysis (SSTA) used in logic synthesis be modified and applied to the delay and yield computation in HLS?* (2) *how does the resource binding affect yield?* (3) *how does the scheduling affect yield?* (4) *how can scheduling and resource binding tasks be combined together to efficiently solve the problem with the objective of minimizing latency under yield constraint?

I. Introduction

The process variations cause a serious design problem today. Designing architecture/logic under the consideration of the worst case process margin is not a viable solution any more, because the degree of the variations in the new process technologies is very high. For example, variation on transistor parameters causes to 20X chip leakage variation and 30% chip performance variation across 1000 samples of a design manufactured in an 180nm technology [1]. To deal with the impact of these process variations on design, several analysis and/or optimization methods are developed, using statistical information. The works in [2], [3], [4], [5] attempted to estimate the yield of design considering the impact of variability on timing, whereas the works in [6], [7] estimated the yield considering power. In addition, the works in [8], [9] and in [10], [11] proposed timing yield and power yield optimization approaches, respectively.

This work belongs to the performance (timing) yield optimization in high-level synthesis (HLS). While there have been proposed several effective approaches [2], [3], [4], [5] on logic level optimization using statistical static timing analysis (SSTA), the problem of HLS with the consideration of timing variation has not been well established and solved so far. Very recently, the HLS work in [12] took into account the timing variation. However, its critical limitation is that the performance yield constraint is not directly combined with the scheduling and resource binding, in that once an iteration of scheduling and binding of simulated annealing (SA) is completed, SSTA is applied to the scheduled and bound DFG (dataflow graph) to obtain delay distribution, which definitely causes the low chance of finding a better solution of scheduling and binding in SA.

II. Timing Variation-Aware High-Level Synthesis

The two major tasks in HLS are operation scheduling and resource binding. The conventional scheduling algorithms usually use the worst case delay of each functional unit to schedule operations to clock steps. However, too much is sacrificed. That is, as the delay variation of the fabricated functional unit continuously increases, the use of the worst case delay is not possible because it allows too much slack time to be wasted in clock steps even if it always guarantees 100% performance yield. A better method will be to find a much tighter schedule at the expense of a slight loss of performance yield. Consequently, statistical delay information is necessary in the scheduling and resource binding. The key concern is how the statistical information is efficiently manipulated and effectively used in the process of scheduling and resource binding.

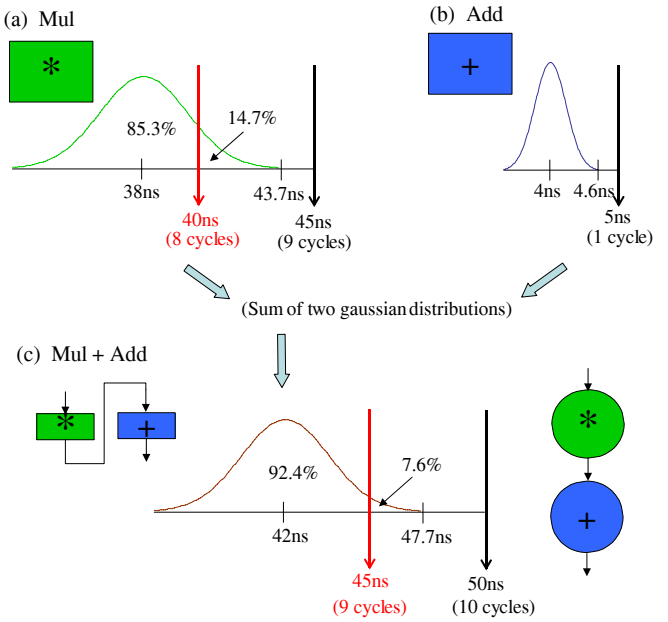


Fig. 1. An example showing the trade-off between schedule length (in cycles) and performance yield in the presence of timing variation of resources (1 cycle delay = 5ns).

First, let us show how trading performance yield with latency looks like, using small examples. Fig. 1(a) and (b) show the Gaussian delay distributions of a multiplier and an adder, respectively. In addition, Fig. 1(c) shows the delay distribution of a chain of multiplier in (a) followed by an adder

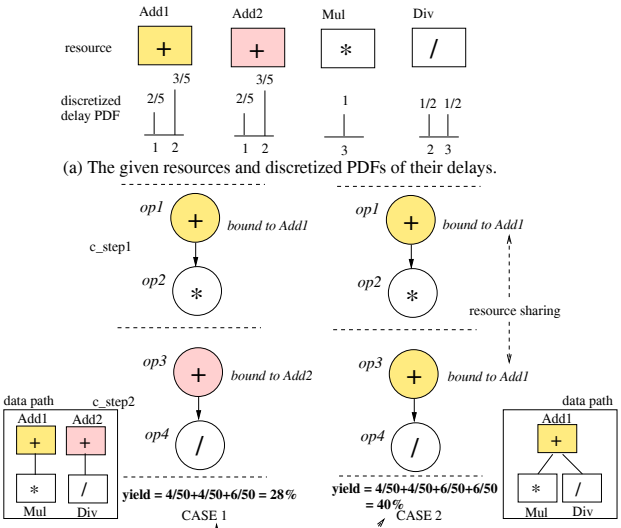
in (b). A conventional scheduling method schedules a multiplication operation in Fig. 1(a) in 9 cycles because the worst case delay of a multiplier is 9 cycles. However, if we sacrifice 14.7% yield, we can schedule the operation in 8 cycles, saving 1 cycle. Furthermore, if multiple operations are involved in scheduling, much shorter latency of schedule can be obtained even with a better performance yield. For example, for the two consecutive operations in Fig. 1(c), the conventional method schedules them in 10 cycles (= 1 + 9). However, if we examine the delay distribution of the corresponding hardware modules, as shown in Fig. 1(c), we can schedule the operations in 9 cycles at the expense of only 7.6% yield loss.

A. Statistical Static Timing Analysis in HLS

To check the timing of operations in the process of scheduling and resource binding, a statistical static timing analysis (SSTA) of the resources is required. There have been many SSTA works on logic circuits. To accurately model the circuit delay variations, most of the works use the continuous delay distributions [4], [9], [13] of logic gates. However, in HLS such a continuous delay distribution can cause a serious run time problem because we need to recompute the delay distribution for each attempt of scheduling and resource binding. Consequently, a discrete delay distribution would be a better choice for timing analysis in HLS. One noticeable work on the discrete delay model is that in [14] where it generates a discretized probability density function (PDF) based on the continuous PDFs of logic cells. For example, Fig. 2(a) shows discretized delay PDFs of resources Add1, Add2, Mul, and Div where Add1 and Add2 take delays of 1 and 2 units of time with probabilities of 2/5 and 3/5, respectively, Mul takes delay of 3 with probability of 1, and Div takes delays of 2 and 3 with probabilities of 1/2 and 1/2, respectively.

B. Variation-Aware Resource Binding

Fig. 2 shows how the resource binding affects the performance yield. Let us consider two binding solutions, CASE 1 and CASE 2 in Fig. 2(b) for a scheduled DFG using the available four resources in Fig. 2(a). Note that in the binding of CASE 2, addition operations *op1* and *op3* scheduled in different clock steps are both bound to Add1, but in CASE 1, *op1* is bound to Add1 and *op3* is bound to Add2. From the results of resource binding, we can compute their delays. The first four columns in the table of Fig. 2(c) show all the delay combinations of resources with their probabilities in the fifth column. For example, 1, 1, 3, 2 in the row marked by '→' in Fig. 2(c) represent the delays of Add1, Add2, Mul, Div with probabilities of 2/5, 2/5, 1, 1/2, respectively. The product of those probabilities equals to 4/50 (=(2/5)*(2/5)*1*(1/2)) in the fifth column. From each combination of delays, we can compute the execution time at each clock step. The longest execution time (i.e., final delay) will be the delay of the corresponding binding solution. The last two columns in Fig.



delay of Add1	delay of Add2	delay of Mul	delay of Div	prob.	final delay in (b) CASE 1	final delay in (b) CASE 2
1	1	3	2	4/50	4	4
1	1	3	3	4/50	4	4
1	2	3	2	6/50	4	4
1	2	3	3	6/50	5	4
2	1	3	2	6/50	5	5
2	1	3	3	6/50	5	5
2	2	3	2	9/50	5	5
2	2	3	3	9/50	5	5

(c) Statistical delay calculations for resource bindings in (b) using the PDFs in (a)

Fig. 2. Examples showing the effect of resource binding on performance yield.

2(c) show the final delays of bindings CASE 1 and CASE 2. For example, the circled number '4' in the last column is the maximum delay of the delays in clock steps 1 and 2 whose computation is shown in the right side of Fig. 2(c). When we assume that clock period is 4 time units, the combinations of delays which lead to no clock period violation for CASE 1 are the first three (rows) in the table while the first four (rows) in CASE 2. Consequently, the performance yield computations for CASE 1 and CASE 2 are the sums of the probabilities of the corresponding three and four combinations in the table, respectively, as indicated by the dotted circles in Fig. 2(c). As we can see the yield numbers in Fig. 2(b), resource "sharing" gives a positive effect on the performance yield.

Observation 1: Let *G1* be a scheduled and resource bound DFG where operations *op1* and *op2*, scheduled at clock step *c1* and *c2*, are bound to the same resource *r1*, and G_2 be the same as G_1 except *op1* and *op2* are bound to *r1* and *r2*, respectively. In addition, let Y_G denote the performance yield of *G*. Then, $Y_{G1} \geq Y_{G2}$. (We omit the proof due to space limitations.)

According to Observation 1, it is highly desirable to perform as many resource sharings as possible. The problem is how to maximize the resource sharing systematically, taking into account the binding conflicts among the operations. This is the main concern of our proposed timing variation-aware HLS algorithm, HLS-tv, which will be presented in Section II-D.

Observation 2: Note that in terms of performance yield calculation the two subgraphs in Fig. 3(a) are identical. We can easily check that the delay PDFs of the two subgraphs in Fig. 3(b) are the same. This leads to defining a concept called *yield-equivalent*. Combined with Observation 1, finding such a set of similar subgraphs in DFG and trying to bind resources among the operations in the subgraphs can improve the yield.

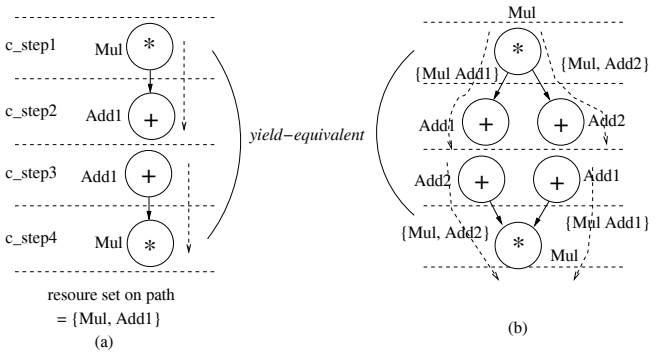


Fig. 3. Examples of *yield-equivalent* subgraphs.

Definition 1: Two connected operation subgraphs $u1$ and $u2$ of DFG are called in *yield-equivalent* relation if they satisfy:
(1) The numbers of paths from top to bottom in $u1$ and $u2$ are the same;
(2) For every path in $u1$ ($u2$), there is a path in $u2$ ($u1$) such that the sets of the operations on the two paths are the same;
(3) For every path in $u1$ ($u2$), there is a path in $u2$ ($u1$) such that the sets of the resources bound to the operations on the two paths are the same;
(4) $u1$ and $u2$ are scheduled to the same number of clock steps.

In addition, $u1$ and $u2$ are called *potentially yield-equivalent* if they satisfy conditions (1) and (2) only. The condition (3) defines *yield-equivalent binding* and the condition (4) defines *yield-equivalent scheduling*. So, if two potentially yield-equivalent subgrahps $u1$ and $u2$ are yield-equivalently bound and scheduled, they become yield-equivalent. Since two yield-equivalent subgraphs $u1$ and $u2$ have the same delay distribution, either $u1$ or $u2$ should be considered in calculating performance yield. Hence, a set of yield-equivalent subgraphs can be represented by one element in it. This element is called *yield-equivalent pattern*.

Yield-equivalent pattern μ is characterized by two values, t and $DP(t)$, that is, $\mu(t,DP(t))$ where t is the number of clock steps the pattern takes for execution and $DP(t)$ represents the discretized PDF of the pattern's delay. Then, the performance yield constrained HLS problem we want to solve can be formulated as:
Problem 1: Under a given resource set R and a yield constraint κ (≤ 1) find the optimal set $L = \{\mu1,\mu2,\ldots\}$ of yield-equivalent patterns such that (i) L disjointly covers all the operations in DFG, (ii) (the performance yield of L) $\geq \kappa$, where the performance yield is computed in the tabular method using $DP(t)$s of μ_is, and (iii) the total latency is minimal.

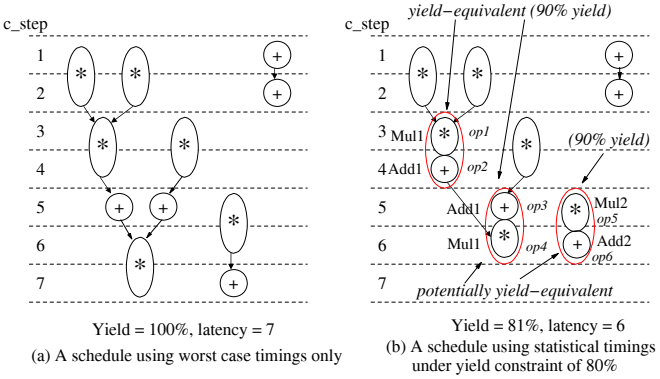


Fig. 4. Conventional scheduling and timing variation-aware scheduling under yield constraint using yield-equivalent patterns.

C. Variation-Aware Scheduling and Binding

The objective of variation-aware scheduling is to find a schedule of operations that minimizes the latency under resource and performance yield constraints. If only the worst case delays of resources are used in scheduling, the performance yield will be always satisfied, but the latency is too long. For example, Fig. 4(a) shows such a schedule, confirming 100% yield. On the other hand, Fig. 4(b) shows the case where the two subgraphs *op1→op2* and *op3→op4* are yield-equivalently bound to resources and scheduled in two clock steps rather than three clock steps, and another subgraph *op5→op6* is scheduled in two clock steps, by which the overall yield is down. But it is still above the yield constraint of 80%, resulting in the latency reduced by one clock. Here, the key point in scheduling (together with resource binding) is to find as many subgraphs that are potentially yield-equivalent each other as possible, because only one of the PDFs of the yield-equivalent subgraphs contributes to the resultant yield.

In the example of Fig. 4(b), suppose the discretized probability (*DP*) that *op1→op2* with *op1* bound to *Mul1* and *op2* bound to *Add1* to be executed in two clock steps is 0.9, and *DP* that *op5→op6* with *op5* bound to *Mul2* and *op6* bound to *Add2* to be executed in two clock steps is 0.9. Then, since *op1→op2* and *op3→op4* are yield-equivalent and the operations other than *op1* through *op4* are scheduled and bound with their *DP*s all 1.0, the final yield becomes 0.9 (for *op1→op2* and *op3→op4*) *0.9 (for *op5→op6*) = 81%.

D. Integrated Algorithm for Variation-Aware Scheduling and Resource Binding

As shown in the Fig. 4, to effectively find yield-equivalent patterns, it is necessary to perform scheduling and resource binding simultaneously. The input to our algorithm HLS-tv is a DFG with resource set R and yield constraint κ to satisfy.

Essentially, HLS-tv is based on the branch-and-bound strategy in an attempt to search all (potentially) yield-equivalent subgraphs. The algorithm builds an optimal set of yield-equivalent patterns iteratively. To speed up the search process, our algorithm adopts a concept of *window-based search* in DFG.

Definition 2: *Window W* is defined to be the maximum consecutive clock steps beyond which no subgraphs satisfy the resource constraint. *Frontier* operations in a window are the operations scheduled in the beginning clock step of the window.

HLS-tv recursively performs the two steps: (Step 1) *selecting the frontier operations and setting a window*; (Step 2) *extracting yield-equivalent subgraphs from the window and finding their combination that minimizes the latency under the yield constraint*. We explain the procedure of HLS-tv with the example in Fig. 5.

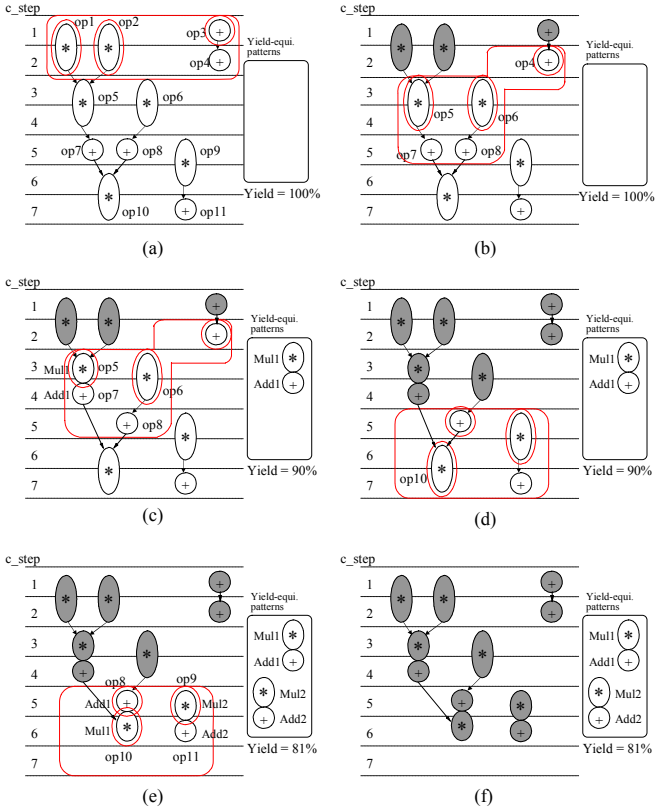


Fig. 5. Steps showing the generation of yield-equivalent patterns by HLS-tv.

Suppose the available resources are two multipliers (*Mul1, Mul2*) and two adders (*Add1, Add2*) for the DFG in Fig. 5(a). Moreover, we assume the performance yield κ=0.8. For a multiplication $DP(t = 1cycle) = 0.7$ and $DP(t = 2cycles) = 1.0$, for an addition, $DP(t = 1cycle) = 1.0$ and $DP(t = 2cycles) = 1.0$, for two consecutive additions, $DP(t = 1cycle) = 0.75$, and for an addition followed by a multiplication (or a multiplication followed by an addition), $DP(t = 2cycles) = 0.9$ and $DP(t = 3cycles) = 1.0$.

In Step 1, we extract *W* from DFG with frontier operations. Let's focus on clock step 1 where operations *op1*, *op2*, and *op3* become the frontier operations. We set a window *W* (i.e., Step 1) for the frontiers according to Definition 2. The outer box shows *W*. We can see that if another operation were added to *W*, some pattern extracted from *W* would violate resource constraint (e.g., if *op5* were added to *W*, pattern {*op1, op2, op5*} would require 3 multipliers). In Step 2, from *W*, we can extract yield-equivalent operation patterns; there are four patterns: (*case1*) multiplication to be scheduled in a single clock step; (*case2*) multiplication in two clock steps; (*case3*) addition in a single clock step; (*case4*) two consecutive additions in a single clock step. Here, the valid ones are *case2* and *case3* because the other cases' *DP(·)* values are less than κ (=0.8). Furthermore, since the *DP(·)* values of *cases2* and *cases3* are all 1.0, we don't need to include the patterns in the set *L* of yield-equivalent patterns. Then, the two steps are repeated. The *selection of new frontier operations* is performed by the following rules:

(a) The children (operations) of previous frontiers whose yield-equivalent patterns are not in *L*;

(b) The children (operations) of the last yield-equivalent patterns included in *L*;

(c) The unscheduled operations which can be scheduled at the same clock steps with some operations obtained by (a) and (b).

In Fig. 5(b), new frontier *op4* and *op5* are obtained by (a), *op6* by (c). An example of applying rule (b) will be found in the next iteration. Window *W* now includes *op4*,…, *op8*. Note that a window is not always rectangular. Among subgraphs in *W*, suppose we select subgraph {*op5, op7*} (Fig. 5(c)). Since its $DP(t = 2cycles)$ is greater than κ, it is added to *L*. Note that we can also select subgraph {*op6, op8*}. However, it can be easily seen that adding one more pattern to *L* doesn't reduce the latency any more at the expense of the decrease of yield from 0.9 to 0.81=0.9*0.9. Now, we proceed to the next iteration. In Fig. 5(d), new frontier *op10* is obtained by rule (b). In the new window, we choose two subgraphs: {*op8, op10*}, {*op9, op11*}. The first subgraph is potentially yield-equivalent to the element in *L*. Thus, we perform yield-equivalent binding from *L*: *op8* is bound to *Add1*, *op10* to *Mul1*. The second subgraph is also potentially yield-equivalent, but there is no yield-equivalent binding from *L* because the first one has already used *Add1* and *Mul1*. In this case, we need to find a binding that maximizes resource sharing to minimize the decrease of yield. (See Observation 1.) There is only one binding choice: *op9* to *Mul2* and *op11* to *Add2*. The bound subgraph is included to *L* and yield is updated to 0.81(=0.9*0.9), which is still above κ = 0.8, as shown in Fig. 5(e). Now, all operations are scheduled and bound with the latency of 6. We are now in the final stage of finding one of solutions, following through one branch of search tree, from which HLS-tv applies the backtracking strategy in the branch-and-bound method.

III. Experimental Results

We have implemented our proposed timing variation-aware HLS algorithm HLS-tv in C++, ran on a PC equipped with 3.06GHz XEON processor, and tested it on a set of HLS benchmarks to assess the effectiveness of HLS-tv. We use the data in [16] for the statistical delay distribution of resources, ALU (addition/subtraction) and MUL (multiplication): ALU

follows a normal Gaussian delay distribution of $\mu = 2.5ns$, $\sigma = 0.125ns$. We extract four sample points at μ-3σ, μ-σ, μ+σ, μ+3σ. MUL also follows a normal Gaussian distribution of $\mu = 25ns$, $\sigma = 1.25ns$. We extract 11 points with delay of *0.75ns* for the interval between the consecutive points. Table I and Table II show the comparisons of results produced by list scheduling based timing variation-unaware method [15] and our HLS-tv under yield constraints of 90% and 80%, respectively. In short, HLS-tv is able to reduce the latency by 18.8% and 20.2% on average with only 7.1% and 11.9% yield penalties, respectively.

IV. Conclusion

In this paper we presented a solid and effective solution to the problem of timing variation-aware high-level synthesis (HLS). Our study showed that both of resource binding and scheduling significantly affected the results of performance yield HLS. This work comprehensively investigated and addressed the following new problems in HLS: (1) *how can the statistical static time analysis (SSTA) used in logic synthesis be modified and applied to the delay and yield computation in HLS?* (2) *how does the resource binding affect yield?* (3) *how does the scheduling affect yield?* The key concept introduced to solve the problems efficiently and effectively was *yield-equivalent pattern*. This work (4) *proposed a performance yield constrained integrated HLS algorithm* based on the concept. It has been experimentally shown that our algorithm can effectively reduce the latency.

Acknowledgment

This research work has been supported by Nano IP/SoC Promotion Group of Seoul R&BD Program in 2007, and also by IT-SoC Program and ETRI project. This work was also partially supported by the Ministry of Science and Technology (MOST)/Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," In *Proc. DAC*, pp. 338-342, 2003.

[2] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," In *Proc. ICCAD*, pp. 621-625, 2003.

[3] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," In *Proc. DAC*, pp. 331-336, 2004.

[4] A. Agarwal, V. Zolotov, and D. T. Blaauw, "Statistical timing analysis using bounds and selective enumeration," *IEEE TCAD*, pp. 1243-1260, September 2003.

[5] M. Orshansky and A. Bandyopadhyay, "Fast statistical timing analysis handling arbitrary correlations," In *Proc. DAC*, pp. 337-342, 2004.

[6] R. Rao, A. Srivastava, D. Blaauw, and D. Sylvester, "Statistical analysis of subthreshold leakage current for VLSI circuits," *IEEE TVLS*, pp. 131-139, February 2004.

[7] S. Narendra, V. De, S. Borkar, D. Antoniadis, and A. Chandrakasan, "Full-chip subthreshold leakage power prediction model for sub-0.18µm CMOS," In *Proc. ISLPED*, pp. 19-23, 2002.

[8] S. Raj, S. Vrudhula, and J. Wang, "A methodology to improve timing yield in the presence of process variation," In *Proc. DAC*, pp. 448-453, 2004.

[9] S. Choi, B. C. Paul, and K. Roy, "Novel sizing algorithm for yield improvement under process variation in nanometer technology," In *Proc. DAC*, pp. 454-459, 2004.

[10] A. Srivastava, D. Sylvester, and D. Blaauw, "Statistical optimization of leakage power considering process variation using dual-Vth and sizing," In *Proc. DAC*, pp. 773-778, 2004.

[11] A. Davoodi, V. Khandelwal, and A. Srivastava, "Variability inspired implementation selection problem," In *Proc. ICCAD*, pp. 423-427, 2004.

[12] W.-L. Hung, X. Wu, and Y. Xie, "Guaranteeing performance yield in high-level synthesis," In *Proc. ICCAD*, pp. 303-309, 2006.

[13] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical analysis and optimization for VLSI: timing and power*, Springer 2005.

[14] J. -J. Liou, K. -T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," In *Proc. DAC*, pp.661-666, 2001.

[15] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuits*, New York, McGraw Hill, 1994.

[16] K. Kang, B. C. Paul, and K. Roy, "Statistical timing analysis using levelized covariance propagation," In *Proc. DATE*, pp. 764-769, 2005.

TABLE I

COMPARISONS OF RESULTS PRODUCED BY A CONVENTIONAL LS BASED ON [15] AND OUR HLS-tv UNDER PERFORMANCE YIELD CONSTRAINT OF 90%

Design (#ops)	#ALU, #MUL	p_{clk} (ns)	Latency(cycles) LS[15]	Latency(cycles) HLS-tv(Y)	Reduction	Run time(s)
DIFF (18)	3, 3	3.5	32	28 (94.5%)	12.5%	928
		4.0	29	24 (90.7%)	17.2%	930
		4.5	26	22 (93.2%)	15.4%	637
LATT (22)	3, 2	3.5	47	36 (94.3%)	23.4%	2122
		4.0	42	32 (94.3%)	23.8%	3325
		4.5	37	30 (90.2%)	18.9%	1207
AR (28)	2, 3	3.5	57	45 (93.9%)	21.1%	1241
		4.0	51	40 (93.9%)	21.6%	1534
		4.5	45	36 (90.8%)	20.0%	680
EWF (34)	2, 3	3.5	46	37 (93.6%)	19.6%	157
		4.0	42	34 (93.6%)	19.0%	367
		4.5	38	33 (91.5%)	13.2%	113
avg.				**- (92.9%)**	**18.8%**	

TABLE II

COMPARISONS OF RESULTS PRODUCED BY A CONVENTIONAL LS BASED ON [15] AND OUR HLS-tv UNDER PERFORMANCE YIELD CONSTRAINT OF 80%

Design (#ops)	#ALU, #MUL	p_{clk} (ns)	Latency(cycles) LS[15]	Latency(cycles) HLS-tv(Y)	Reduction	Run time(s)
DIFF (18)	3, 3	3.5	32	27 (80.8%)	15.6%	1267
		4.0	29	24 (90.7%)	17.2%	1210
		4.5	26	21 (80.1%)	19.2%	1282
LATT (22)	3, 2	3.5	47	36 (94.3%)	23.4%	4467
		4.0	42	32 (94.3%)	23.8%	4492
		4.5	37	29 (82.5%)	21.6%	3611
AR (28)	2, 3	3.5	57	45 (93.9%)	21.1%	2079
		4.0	51	40 (93.9%)	21.6%	2491
		4.5	45	36 (90.8%)	20.0%	1216
EWF (34)	2, 3	3.5	46	36 (80.2%)	21.7%	478
		4.0	42	33 (93.6%)	21.4%	490
		4.5	38	32 (81.9%)	15.8%	372
avg.				**- (88.1%)**	**20.2%**	