Timing Variation-Aware High-Level Synthesis Considering Accurate Yield Computation

Jongyoon Jung and Taewhan Kim
School of Electrical Engineering and Computer Science, Seoul National University, Korea
Email: {bellrich, tkim}@ssl.snu.ac.kr

Abstract—**This work proposes a new yield computation technique dedicated to HLS, which is an essential component in timing variation-aware HLS research field. The SSTAs used by the current timing variation-aware HLS techniques cannot support the following two critical factors at all: (i) *non-Gaussian delay distribution of 'module patterns'* used in scheduling and binding and (ii) *correlation of timing variation between module patterns*. However, without considering these factors, the synthesis results would be far less accurate in timing, being very likely to fail in timing closure. Even though there are advances in the logic level for SSTAs that support (i) and (ii), the manipulation and computation of (i) and (ii) in the course of scheduling and binding in HLS are unique in that there are *no concepts of module sharing and performance yield computation* in the logic level. Specifically, we propose a novel yield computation technique to handle the non-Gaussian timing variation of module patterns, where the sum and max operations are closed-form formulas and the timing correlation between modules used in computing performance yield is preserved to the first-order form. Experimental results show that our synthesis using the proposed yield computation technique reduces the latency by 24.1% and 28.8% under 95% and 90% performance yield constraints over that by the conventional HLS, respectively. Further, it is confirmed that our synthesis results are near optimal with less than 3.1% error on average.**

I. INTRODUCTION

The timing closure problem is one of the most important problem to be solved in high-level synthesis (HLS). One critical factor that makes the timing closure problem difficult to solve is the timing variations. Conventionally, to avoid the effects of timing variation on the violation of clock period constraint, the worst case delays of functional modules have been used in scheduling of HLS. As technologies scale down to deep sub-micron regime, the rapid increase of the timing variation makes the worst case design too pessimistic [1]. To accurately estimate the impact of the process variation on circuit timing, many effective statistical static timing analysis (SSTA) techniques have been developed for the gate level analysis. The SSTA techniques are classified into two types: path based SSTAs (e.g., [2], [3]) and block based SSTAs (e.g., [4]–[7]). However, there are a few SSTA techniques for the higher level design. There will be a great chance of enhancing performance yield[1] of resultant design as well as performance in the stage of HLS, because the quality of results of scheduling and binding is significantly affected by how much the timing information of hardware modules allocated and used in the design is accurate. Despite such importance of accurate SSTA information of modules in HLS, so far the timing variation-aware HLS techniques have used the SSTAs that are available in the gate level, missing the capability of supporting the unique requirements in HLS: *non-Gaussian delay distributions of module patterns*[2] and *correlation of timing variation between module patterns*. Note that two modules, for example an adder and a multiplier, allocated for execution of operations in DFG (data flow graph), have not only their own delay distributions but also their joint timing correlation, because the delay distributions of the circuit (gate) primitives in the modules are inherently correlated. The two requirements of supporting non-Gaussian delay distribution and timing correlation between modules are in fact the essential parts for the accurate computation of performance yield in the scheduling and binding (i.e., resource sharing).

Recently, the work in [9] took into account the timing variation in HLS. The work pointed out the uniqueness of the timing variation-aware HLS, introducing the concept of performance yield computation. However, the proposed method assumed Gaussian delay distributions of modules and no correlation of timing variation among the modules. In addition to that, the performance yield constraint is not fully combined with the scheduling and resource binding process. Once an iteration of scheduling and binding of simulated annealing (SA) is completed, an SSTA is applied to the scheduled and bound DFG to obtain delay distribution, which causes the low chance of finding a globally optimized solution of scheduling and binding in SA. The work in [10] refined and enhanced the performance yield computation used in [9]. However, for the computation of performance yield for an instance of binding result, it didn't consider the correlation of timing variations between modules. Furthermore, they still assumed Gaussian delay distributions for the modules, lacking timing accuracy of modules. On the other hand, the work in [11] proposed a timing variation-aware HLS technique to reduce the latency under the performance yield constraint. It used a *discretized* probability density function (PDF) to speed up the performance yield computation for every rescheduling and rebinding instance. Though the method can handle any delay distribution of module patterns including non-Gaussian, it didn't take into account the correlation of timing variations between modules.

In this work, we propose a novel yield computation technique that is able to effectively support the non-Gaussian distributions of module patterns and the timing correlation between them. By achieving this timing accuracy for the performance yield computation for every instance, consisting of module patterns, of scheduling and binding, any HLS framework that uses our proposed technique in the performance yield computation can make right (or reliable) decisions on guiding scheduling and binding in timing variation-aware HLS.

This work was supported by Nano IP/SoC Promotion Group of Seoul R&BD Program (10560), IT-SoC Program, the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (No. R01-2007-000-20891-0), and the Korea Ministry of Knowledge Economy (MKE) under the Information Technology Research Center (ITRC) support program supervised by the Institute for Information Technology Advancement (IITA) (IITA-2009-C1090-0804-0009).

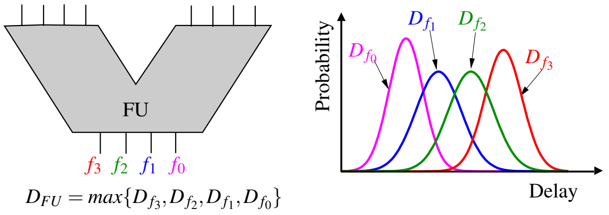
[1] The performance yield is defined as the probability of the design meeting the timing or clock period constraint [1], [8].

[2] Module patterns are the sub-graphs of operations in a DFG (data flow graph) that are scheduled to be executed in one clock step (i.e., as operation chaining) or multiple clock steps (i.e., as multi-cycling). More discussion will be given in the following sections.

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

II. STATISTICAL STATIC TIMING MODEL AND ANALYSIS

Since the functional modules used in HLS are composed of logic gates, the delay distributions of the modules can be obtained by applying a logic level SSTA to them.



(a) (b)

Fig. 1. Modeling delay distributions of a functional module: (a) FU with four output signals; (b) graphs of the delay distributions of f_3, f_2, f_1, and f_0 in (a).

For example, Fig. 1(b) shows the delay distributions of the four output signals f_3, f_2, f_1, and f_0 of a functional unit (FU) in Fig. 1(a). Thus, four random variables are needed to describe the delay distributions of the output signals (i.e., one for each output signal). As the number of FU's outputs increases, the number of random variables also increases, which makes the manipulation and computation of the compositions of delay distributions in HLS very complicated and time-consuming. Therefore, to process the delay variations efficiently, we use a single random variable for a functional module rather than use multiple random variables. The single random variable represents the distribution of the *longest delay from an input signal to an output signal* in the module. For example, for FU in Fig. 1 the random variable is $D_{FU} = max\{D_{f_3}, D_{f_2}, D_{f_1}, D_{f_0}\}$ where D_{f_i} (i=0, 1, 2, 3) represents the delay variable from an input of FU to output f_i.

To describe the delay distributions of functional modules, we adopt the first-order canonical model. (It should be noted that our proposed technique can be easily extended to the second or higher-order delay model.) By using the model, the delay variable of a functional module is expressed as

$$D = d_0 + \sum_{i=1}^{N} d_i X_i + d_{N+1} X_r \tag{1}$$

where d_0 is the nominal delay of the module, X_i and X_r are the random variables to model the process parameter variations, and d_i is the first-order sensitivity of D to X_i. X_i is the *correlated* component of the process parameter, such as channel length and threshold voltage. On the other hand, X_r is the independent random component (to represent the RDF (random dopant fluctuation)). There are two atomic operations in the block-based parameterized SSTA: sum and max where the max operation is much harder to implement. Most SSTA techniques have been assumed that all X_i's and X_r follow Gaussian distributions to simplify the computation of the sum and max operations. However, the Gaussian distribution model is limited in modeling the delay distributions of functional modules. In this work, we accept non-Gaussian delay distributions, in the form of that in *Eq.*(1), for X_i's and X_r. There have been many efficient gate level SSTA techniques developed for non-Gaussian variation sources (e.g., [6], [7]). In particular, using the method in [7] allows the result of sum or max operation for the distribution in the form of *Eq.*(1) to be also the form of *Eq.*(1). Our key contributions compared to the gate level SSTA techniques are that our work (i) *efficiently formulates the correlation of delay variables between module patterns* (in Section III) and (ii) *integrating an incremental computation of non-Gaussian delay distributions required by rescheduling/rebinding into the performance yield computation in HLS* (in Section V).

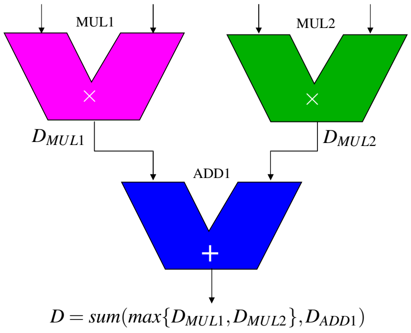


Fig. 2. Example showing the derivation of delay variable of a module pattern consisting of three modules where two ($MUL1$, $MUL2$) of them are connected to the other ($ADD1$) as inputs.

For example, Fig. 2 shows a module pattern with two multipliers being connected to an adder as inputs. The delay variable of the module pattern, denoted as D, is expressed by a composition of max and sum operators, and the delay variables, D_{MUL1}, D_{MUL2} and D_{ADD1} of the functional modules. That is, $D = sum(max\{D_{MUL1}, D_{MUL2}\}, D_{ADD1})$.

III. PERFORMANCE YIELD COMPUTATION IN HLS

This section describes how the performance yield for an instance of scheduling and binding is formulated to guide the direction of rescheduling or rebinding in the framework of HLS. Let us consider an instance of scheduling and binding in Fig. 3(a) produced by the conventional variation-unaware HLS. The modules available are shown in Fig. 3(b). The conventional HLS produces latency of 7 clock cycles by using the worst case module delays. Although the performance yield of the scheduling and binding result is 100%, the latency is too conservative.

On the other hand, the timing variation-aware HLS is able to perform more aggressive scheduling and binding. Let us consider the instance of scheduling and binding in Fig. 3(c) in which three operation sub-graphs $op1 \rightarrow op2$, $op3 \rightarrow op4$, $op5 \rightarrow op6$ are scheduled at two clock steps (i.e., (3,4), (5,6), and (5,6), respectively) instead of three clock steps. Further, operations $op2$ and $op3$ share adder $ADD1$ in Fig. 3(d) and operations $op1$ and $op4$ share multiplier $MUL1$ in Fig. 3(d). On the other hand, $op5$ and $op6$ in the sub-graph $op5 \rightarrow op6$ use $MUL2$ and $ADD2$ in Fig. 3(d) for their executions, respectively. We assume that all the remaining operations are definitely executed within the specified clock steps. Consequently, we may save 1 clock cycle at the expense of some performance yield loss. Here, the important issue is how to accurately compute the performance yield of the scheduling and binding instance in Fig. 3(c), using reliable delay distributions of module patterns in Fig. 3(d). Let D_1 (= $sum(D_{MUL1}, D_{ADD1})$) and $D2$ (= $sum(D_{MUL2}, D_{ADD2})$) be the delay variables of the two module patterns in Fig. 3(d). Then, the performance yield of the scheduling and binding instance is expressed as $P(D_1 \leq 2t_{clk}, D_2 \leq 2t_{clk})$. Given PDFs for D_1 and D_2, we need to compute $P(D_1 \leq 2t_{clk}, D_2 \leq 2t_{clk})$.

To simplify the computation of the performance yield, the previous timing variation-aware HLS techniques (e.g., [11]) compute $P(D_1 \leq 2t_{clk}, D_2 \leq 2t_{clk})$ by replacing it with $P(D_1 \leq 2t_{clk}) \cdot P(D_2 \leq 2t_{clk})$ by assuming that the delay distributions of two

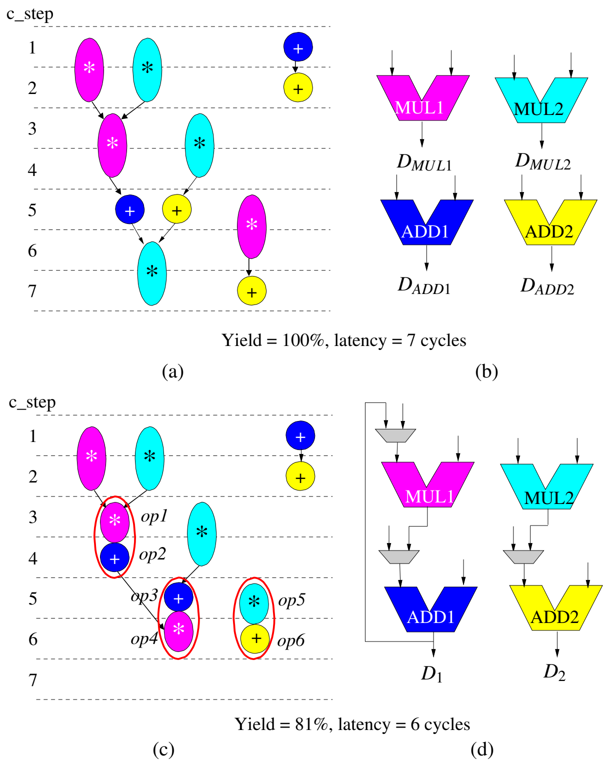


Fig. 3. (Conventional) process variation-unaware HLS vs. process variation-aware HLS: (a) scheduling and binding result using the worst case module delays by variation-unaware HLS; (b) modules bound to the DFG in (a); (c) scheduling and binding result using statistical module delays by variation-aware HLS; (d) module patterns bound to the DFG in (c).

different modules are independent.[3] In reality, however, their delay distributions are correlated because they neighbor on each other in the same die. Obviously, the assumption is not reasonable, and thus, the simplified computation is neither accurate nor useful. To precisely compute the performance yield, we should take into account the correlation between the distributions of D_1 and D_2. The correct and meaningful formulation of the performance yield computation is

$$P(D_1 \le 2t_{clk}, D_2 \le 2t_{clk}) = \int_0^{2t_{clk}} \int_0^{2t_{clk}} f(d_1, d_2)\mathrm{d}d_1\mathrm{d}d_2 \tag{2}$$

where $f(d_1, d_2)$ is the joint probability density function (JPDF) of D_1 and D_2.[4]

As the number of module patterns increases during the process of scheduling and binding, an efficient and accurate computation of their JPDF is very critical. The next section provides the details on the efficient computation of the JPDF.

IV. AN EFFICIENT JPDF COMPUTATION

It is easy, in HLS, to derive the JPDF of two "Gaussian" random variables, D_1 and D_2, by calculating the covariance between D_1 and D_2, and the JPDF still follows a Gaussian distribution (e.g., [9], [10]). However, it is difficult to obtain a closed form for the JPDF of "non-Gaussian" D_1 and D_2. Our solution is based on the use of Fourier series as follows: The JPDF of multiple random variables can be approximated by its first K orders of Fourier series. The benefit of the use of Fourier series in JPDF computation is threefold. First, we are able to efficiently and accurately compute the JPDF by manipulating parameter K. Second, any distribution including both Gaussian and non-Gaussian can be used. Further, Fourier series of the JPDF provides a closed form for the JPDF, which reduces performance yield computation to a simple checking on a lookup table.

A. JPDF Computation via Fourier series

This section presents an efficient method to compute the JPDF of D_1 and D_2 in the canonical form, as shown in *Eq.*(1). (The method can be trivially extended to compute the JPDF of three or more random variables.) Suppose D_1 and D_2 are the delay random variables of two module patterns. Since the variables model the timing quantities, we can safely assume that they are bounded in certain intervals:

$$0 \le D_1 \le l, \quad 0 \le D_2 \le h. \tag{3}$$

Their upper bounds are set as $l = \mu_1 + 4\sigma_1$ and $h = \mu_2 + 4\sigma_2$ where μ_i and σ_i represent mean and standard deviation of D_i, respectively. By applying a sequence of sum and max operations to the module patterns by using the SSTA technique in [7], we can produce the PDFs of D_1 and D_2 that are in the canonical form of *Eq.*(1):

$$D_1 = d_0 + \sum_{i=1}^{N} d_i X_i + d_{N+1} X_r, \tag{4}$$

$$D_2 = e_0 + \sum_{i=1}^{N} e_i X_i + e_{N+1} X_r. \tag{5}$$

Let $f(d_1, d_2)$ be the JPDF of D_1 and D_2. Then, we approximate $f(d_1, d_2)$ in the region of $[0, l; 0, h]$ via its first K orders of Fourier series as

$$f(d_1, d_2) \approx \sum_{p,q=-K}^{K} \alpha_{pq} \cdot e^{\zeta_p d_1 + \eta_q d_2} \tag{6}$$

where $\zeta_p = jp(2\pi/l)$, $\eta_q = jq(2\pi/h)$ with $j = \sqrt{-1}$. The Fourier coefficient α_{pq} is derived by

$$\alpha_{pq} = \frac{1}{lh} \int_0^h \int_0^l e^{-\zeta_p d_1 - \eta_q d_2} \cdot f(d_1, d_2)\mathrm{d}d_1\mathrm{d}d_2. \tag{7}$$

Since the JPDF $f(d_1, d_2)$ is zero outside the region $[0, l; 0, h]$, *Eq.*(7) can be simplified as

$$\begin{aligned} \alpha_{pq} &= \frac{1}{lh} E[e^{-\zeta_p D_1 - \eta_q D_2}] \\ &= \frac{1}{lh} E[e^{-\zeta_p (d_0 + \sum_i d_i X_i + d_{N+1} X_r) - \eta_q (e_0 + \sum_i e_i X_i + e_{N+1} X_r)}] \\ &= \frac{e^{c_{0,pq}}}{lh} E[e^{-\sum_i c_{i,pq} X_i - c_{N+1,pq} X_r}] \end{aligned} \tag{8}$$

where $c_{0,pq} = \zeta_p d_0 + \eta_q e_0$, $c_{i,pq} = \zeta_p d_i + \eta_q e_i$, and $c_{N+1,pq} = \zeta_p d_{N+1} + \eta_q e_{N+1}$. Since all X_i's and X_r are independent, α_{pq} can be restated as

$$\alpha_{pq} = \frac{e^{c_{0,pq}}}{lh} E[e^{-c_{N+1,pq} X_r}] \prod_{i=1}^{N} E[e^{-c_{i,pq} X_i}]. \tag{9}$$

---

[3] For example, let us assume that $P(D_1 \le 2t_{clk}) = P(D_2 \le 2t_{clk}) = 90\%$ in Fig. 3(c). Then, the performance is $P(D_1 \le 2t_{clk}) \cdot P(D_2 \le 2t_{clk}) = 81\%$.

[4] The performance yield can also be computed by using max operation in SSTA. With $D = max\{D_1/2, D_2/2\}$ defined, $P(D_1 \le 2t_{clk}, D_2 \le 2t_{clk}) = P(D \le t_{clk})$. However, max operation is more expensive than JPDF computation because max operation uses JPDF [7].

As it can be seen in *Eq.*(9), we have to compute $E[e^{-cX}]$ where c is a constant and X is a random variable. By definition,

$$E[e^{-cX}] = \int_{-\omega}^{\omega} e^{-cx} f(x)\mathrm{d}x \tag{10}$$

where $f(x)$ is the PDF of X whose range is bounded by $-\omega \le X_i \le \omega$. (In reality, each variation source X_i does not vary from $-\infty$ to $+\infty$ as Gaussian does.) Without loss of generality, we assume that all variation sources are centered with zero mean (i.e., $E[X_i] = 0$). So, we can approximate e^x via the first M terms of its Maclaurin series as

$$E[e^{-cX}] \approx \int_{-\omega}^{\omega} \left(\sum_{n=0}^{M} \frac{(-cx)^n}{n!} \right) \cdot f(x)\mathrm{d}x. \tag{11}$$

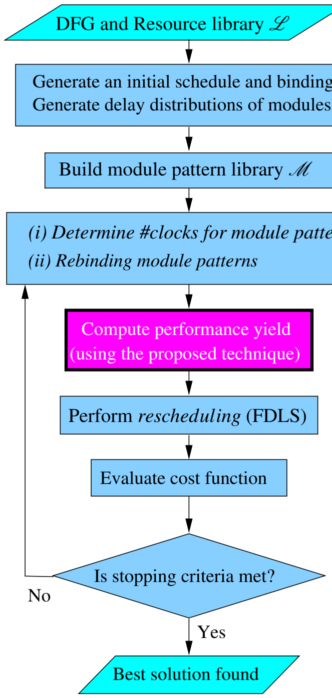
With $M = 3$, $E[e^{-cX}]$ can be approximated as

$$\begin{aligned} E[e^{-cX}] &\approx \int_{-\omega}^{\omega} \left(1 - cx + \frac{c^2x^2}{2} - \frac{c^3x^3}{6} \right) \cdot f(x)\mathrm{d}x \\ &= \int_{-\omega}^{\omega} f(x)\mathrm{d}x - c\int_{-\omega}^{\omega} xf(x)\mathrm{d}x \\ &\quad + \frac{c^2}{2}\int_{-\omega}^{\omega} x^2 f(x)\mathrm{d}x - \frac{c^3}{6}\int_{-\omega}^{\omega} x^3 f(x)\mathrm{d}x \\ &= 1 - cE[X] + \frac{c^2}{2}E[X^2] - \frac{c^3}{6}E[X^3]. \end{aligned} \tag{12}$$

Therefore, we can compute $E[e^{-cX}]$ with only several moments ($E[X^n]$) of X. For each variation source X_i, the three moments ($E[X_i]$, $E[X_i^2]$, $E[X_i^3]$) are precomputed and saved in 2D-table. In fact, it is not necessary to compute $E[X_i]$ because its value is always zero. Since the number of variation sources (N) typically ranges from 5 to 20, the table size is relatively small. Consequently, the α_{pq} computation requires only $2(N + 1)$ times of a small sized 2D-table lookup. In addition, the α_{pq} computation can be done in a constant time.

B. Complexity of JPDF computation

The complexity of JPDF computation depends on the number of the Fourier coefficients, because computing all the Fourier coefficients means that JPDF computation has been done. The number of the Fourier coefficients is determined by the order of Fourier series to approximate JPDF (K) and the number of module patterns under the consideration of performance yield computation. In practice, we obtained good approximations of JPDF with K = 4. Since we have to consider the delay distributions for module patterns, their JPDF becomes L-dimensional function with L delay distributions for L module patterns. (For example, when there are three module patterns, their JPDF becomes 3-D function.) Thus, the JPDF computation requires L-dimensional Fourier series. Moreover, the number of the Fourier coefficients is bounded by $(2K + 1)^L$. As indicated in the prior section, the Fourier coefficient computation can be done in a constant time. Thus, the complexity of JPDF computation is bounded by $\mathcal{O}((2K + 1)^L)$. Empirically, K = 4 is enough and L typically ranges from 1 to 4 depending on hardware resource and performance yield constraints. Since too many module patterns (i.e., L is large) significantly degrade the performance yield, we found that our proposed JPDF computation method was durable and efficient in almost all benchmark designs.



DFG and Resource library $\mathscr{L}$
Generate an initial schedule and binding
Generate delay distributions of modules
Build module pattern library $\mathscr{M}$
(i) Determine #clocks for module patterns
(ii) Rebinding module patterns
Compute performance yield
(using the proposed technique)
Perform rescheduling (FDLS)
Evaluate cost function
Is stopping criteria met?
No
Yes
Best solution found

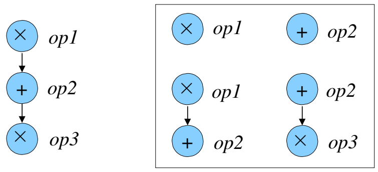
Fig. 4. The flow of the simulated annealing based iterative improvement HLS framework (HLS-FS).

V. SCHEDULING AND RESOURCE BINDING GUIDED BY PERFORMANCE YIELD

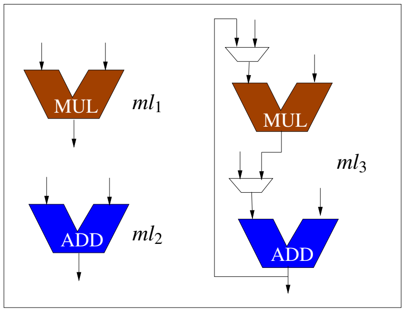
This section describes when the performance yield computation using the proposed technique takes place and where the computed performance yields are used in the framework of HLS. In short, our technique can be integrated into *any iterative* HLS algorithms. Here, we introduce one typical (simulated annealing based) iterative improvement HLS framework, called HLS-FS (HLS combined with Fourier series based yield computation technique), whose overall flow is shown in Fig. 4, to explain when the computation of yield performance occurs and where the computed results are used.

The HLS framework (HLS-FS) in Fig. 4 accepts, as input, a DFG, a library $\mathcal{L}$ of hardware modules to use, and the performance yield bound to be satisfied, and generates, as output, a scheduled and module bound DFG with the shortest latency while meeting the performance yield constraint. First, a gate level SSTA is applied to modules in $\mathcal{L}$ to produce the delay distributions of the modules in the canonical form in *Eq.*(1). The next step is then to find sub-graphs in the DFG and construct a module pattern library $\mathcal{M}$. The module patterns in $\mathcal{M}$ will be bound to the sub-graphs in the subsequent steps.

An example of DFG and its module pattern library $\mathcal{M}$ are shown in Fig. 5. Let us assume that either sub-graph $op1 \rightarrow op2$ or sub-graph $op2 \rightarrow op3$ in the DFG can be implemented with module pattern ml_3. Hence, we include the module pattern to $\mathcal{M}$ for the two sub-graphs. In this example, the number of modules on the critical path of module pattern, defined as the *length* of the module pattern, is at most 2. The module patterns with length = 1 (e.g., ml_1 and ml_2) consist of single modules. Essentially, we do not examine all possible sub-graphs in DFG, because a module pattern whose length is long causes the performance yield to get down significantly. (Refer to the analysis and experimental results in [11].) Moreover, it requires an



(a) A DFG and its sub−graphs



(b) The module pattern library for the DFG in (a)

Fig. 5. A simple DFG and the module patterns in library $\mathcal{M}$.

excessive running time to extract all sub-graphs as the size of DFG increases. In practice, we found that the longest lengths among the module patterns for the best solutions are in the range [2, 4]. Thus, it does make sense to consider only the module patterns whose lengths are less than or equal to 4, which allows a huge saving of computation time in extracting unnecessarily large sub-graphs and computing the associated JPDF.

Then the next step is to perform one of two perturbation moves in the simulated annealing process.

(i) *Determining the number of clock steps for module patterns*: According to the performance yield constraint and the delay distributions of module patterns, the numbers of clock steps for executing the module patterns should be determined. For example, suppose the delay distribution of a multiplier module provides the information that 92% of the implementations of the multiplier in wafer can complete a multiplication operation in 4 clock cycles and the remaining 8% needs 5 clock cycles to complete the operation. Then, if the performance yield constraint is given to 90%, we may set the delay of the multiplier module to 4 clock steps to improve the latency rather than 5 clock steps.

(ii) *Rebinding module patterns*: Since there are multiple module patterns in $\mathcal{M}$, we attempt to explore binding alternatives. For example, two adder modules $ADD1$ and $ADD2$ in $\mathcal{M}$ have different delay distributions, but the numbers of clock steps determined at (i) for the adders are the same. Then, the selection of one adder from the two adders to bind an addition operation in DFG does not affect the latency, but affects the performance yield. If the performance yields by $ADD1$ and $ADD2$ are both not less than the performance yield constraint, it is better to choose the one that would lead to improve the resulting performance yield. However, at this point we cannot surely tell which binding is superior to the other. Consequently, it is necessary to try all possible rebindings to find a best binding result.

By setting the number of clock steps of module patterns and/or binding module patterns, the next step is to compute the performance yield of the current scheduling and binding instance of DFG. In this step, the JPDF of the module patterns used in the instance is computed by using our proposed technique (in Section IV). Then, the performance yield (in Section III) is computed. If the performance yield does not meet the performance yield constraint, we perform the rebinding step again.

The succeeding step is to perform *rescheduling*. We use the forced directed list scheduling (FDLS) [12]. The acceptance/rejection of the new scheduling solution is determined by the probability $e^{-\Delta L/T}$ where $\Delta L = L_{new} - L_{best}$. L_{new} and L_{best} represent the latency of the new schedule and the latency of the best solution found so far, respectively. T indicates the current temperature in the simulated annealing. To speed up the overall process, we have adopted the fast simulated annealing scheme in [13]. Finally, if the stopping criteria is met, the best solution found so far is returned.

VI. EXPERIMENTAL RESULTS

Our timing variation-aware HLS system HLS-FS that combined the Fourier series based performance yield computation method was coded in C++ and the experiments were performed on a Linux machine (Quad-Core Intel Xeon Processor@1.86GHz, 4GB RAM). We have tested our system on the high-level synthesis benchmarks in [14]. The tested designs are: Auto Regression Filter (ARF), Elliptic Wave Filter (EWF), Finite Input Response Filter 1 (FIR1), jpeg Inverse Discrete Cosine Transform (JIDCT), and jpeg Forward Discrete Cosine Transform (JFDCT). Because the benchmarks do not have any timing variation information, we assume each variation source follows either a Gaussian distribution, a uniform distribution, or a triangle distribution. To assess the accuracy of our technique, we compare the synthesis results with those produced by an exhaustive HLS optimization, which tries all possible schedulings and bindings, and applies the Monte Carlo simulation for the JPDF computations.

TABLE I
COMPARISONS OF RESULTS PRODUCED BY THE EXHAUSTIVE OPTIMIZATION HLS-MC USING MONTE CARLO SIMULATION AND HLS-FS UNDER THE PERFORMANCE YIELD CONSTRAINT OF 95%.

Design (# of operations)	HLS-MC (exhaustive)		HLS-FS (ours)	
	latency reduction/yield	running time	latency reduction/yield	running time
Uniform variation sources				
ARF(28)	22.5%/95.7%	8375s	21.0%/97.3%	0.3s
EWF(34)	21.7%/96.1%	5446s	20.5%/97.5%	0.1s
FIR1(44)	23.6%/95.9%	9573s	22.6%/96.8%	0.5s
JIDCT(122)	29.2%/95.5%	59304s	24.4%/96.6%	1.2s
JFDCT(134)	32.3%/95.2%	76884s	26.1%/96.2%	2.5s
Average	25.9%/95.7%		22.9%/96.9%	
Triangle variation sources				
ARF(28)	22.7%/96.3%	8078s	21.8%/97.7%	0.1s
EWF(34)	22.1%/96.3%	6827s	21.6%/97.9%	0.1s
FIR1(44)	24.7%/95.5%	15926s	23.6%/97.9%	1.3s
JIDCT(122)	31.0%/95.5%	80512s	25.7%/97.1%	2.3s
JFDCT(134)	32.4%/95.3%	83729s	26.3%/96.8%	2.5s
Average	26.6%/95.8%		23.8%/97.5%	
Gaussian variation sources				
ARF(28)	26.8%/96.0%	8683s	24.3%/96.5%	0.3s
EWF(34)	24.5%/96.4%	7470s	24.0%/96.5%	0.4s
FIR1(44)	28.0%/95.9%	12883s	25.0%/96.4%	1.6s
JIDCT(122)	31.5%/95.8%	67202s	27.3%/96.4%	2.6s
JFDCT(134)	34.4%/95.8%	74467s	28.0%/96.1%	3.5s
Average	29.0%/96.0%		25.7%/96.4%	
Total Average	**27.2%/95.8%**		**24.1%/96.9%**	

Table I shows the comparisons of results produced by the exhaustive HLS method, HLS-MC, which searches all schedulings and bindings and uses Monte Carlo simulation of JPDF computation, and HLS-FS under the performance yield constraint of 95%. The *latency reduction* in the table represents the reduction of the latency over the latency used by a conventional HLS which uses the worst case module delay in scheduling. (We used FDLS [12].) In compared with the latency reduction by HLS-MC, HLS-FS is able to find near optimal solutions with only 3.1% error of latency on average. In addition, even under the non-Gaussian timing variations (i.e., uniform or triangle variations) as well as the Gaussian variations, HLS-FS can generate solutions with less than 1.1% error in the performance yield computation. Nevertheless, HLS-FS saves a huge running time. This clearly indicates that our HLS framework that uses the Fourier series based yield computation method is very accurate and efficient.

TABLE II
COMPARISONS OF RESULTS PRODUCED BY THE EXHAUSTIVE OPTIMIZATION HLS-MC USING MONTE CARLO SIMULATION AND HLS-FS UNDER THE PERFORMANCE YIELD CONSTRAINT OF 90%.

Design (# of operations)	HLS-MC (exhaustive)		HLS-FS (ours)	
	latency reduction/yield	running time	latency reduction/yield	running time
Uniform variation sources				
ARF(28)	27.3%/91.4%	41848s	26.1%/92.7%	1.9s
EWF(34)	25.6%/91.8%	30357s	24.8%/93.1%	2.6s
FIR1(44)	29.1%/91.5%	52479s	27.4%/92.3%	3.5s
JIDCT(122)	34.6%/91.2%	478850s	30.3%/92.5%	7.6s
JFDCT(134)	37.2%/90.5%	635293s	31.5%/91.2%	9.2s
Average	30.8%/91.3%		28.0%/92.4%	
Triangle variation sources				
ARF(28)	26.5%/91.8%	43214s	23.5%/93.1%	1.3s
EWF(34)	26.1%/91.8%	39347s	24.5%/93.6%	3.7s
FIR1(44)	29.0%/91.1%	47908s	27.9%/92.5%	3.9s
JIDCT(122)	36.2%/91.0%	420398s	31.6%/92.1%	8.1s
JFDCT(134)	37.7%/90.3%	685833s	32.5%/92.0%	8.2s
Average	31.1%/91.2%		28.0%/92.7%	
Gaussian variation sources				
ARF(28)	30.5%/91.6%	57127s	29.5%/92.7%	2.9s
EWF(34)	30.2%/91.8%	36345s	28.3%/93.0%	5.2s
FIR1(44)	31.3%/90.9%	76452s	30.8%/92.6%	6.9s
JIDCT(122)	36.8%/91.3%	475306s	31.3%/92.1%	7.1s
JFDCT(134)	38.3%/90.7%	678769s	32.0%/91.8%	9.4s
Average	33.4%/91.3%		30.4%/92.4%	
Total Average	**31.8%/91.2%**		**28.8%/92.5%**	

Table II shows other comparisons under the performance yield constraint of 90%. By relaxing the yield constraint from 95% to 90%, the search spaces of rescheduling and rebinding alternatives by HLS-MC and HLS-FS are increased. As shown in the table, HLS-MC failed in finding some solutions because of the explosive growth of the search space. However, HLS-FS is still able to find accurate solution quickly, reducing the latency by 28.8%.

Finally, Table III shows comparison results produced by the previous timing variation-aware HLS-tv in [11] that uses a discrete delay modeling and does not consider the correlation of module patterns, and HLS-FS under the performance yield constraint of 90%. As shown in Table III, HLS-FS outperforms HLS-tv on all tested benchmarks. As we expected, HLS-FS is able to reduce the latency by 7% further while achieving better performance yield than that by HLS-tv. This clearly reveals that the accurate computation of performance yield is very important in the timing variation-aware HLS and HLS-FS does the right thing in terms of accuracy and efficiency.

VII. CONCLUSION

In this paper, we presented a solution to the fundamental problems of timing variation-aware high-level synthesis (HLS) namely, supporting (i) *non-Gaussian delay distributions of functional modules and connected sets of modules* and (ii) *correlation of timing variation among the modules and connected sets of modules*. Without considering (i) and (ii) in the framework of variation-aware HLS, the synthesis results would be far less accurate in timing, failing in timing closure. It should be noted that even there are advances in the logic level for SSTAs that support (i) and (ii), the manipulation and computation of (i) and (ii) in the course of scheduling and binding in HLS are *unique* in that there are *no concepts of module sharing and performance yield computation in the logic level*.

TABLE III
COMPARISONS OF RESULTS PRODUCED BY THE PREVIOUS VARIATION-AWARE HLS-tv [11] AND HLS-FS UNDER THE PERFORMANCE YIELD CONSTRAINT OF 90%.

Design (# of operations)	HLS-tv [11]		HLS-FS	
	latency reduction/yield	running time	latency reduction/yield	running time
Uniform variation sources				
ARF(28)	18.8%/92.4%	2045s	26.1%/92.7%	1.9s
EWF(34)	18.4%/92.5%	338s	24.8%/93.1%	2.6s
FIR1(44)	19.0%/91.9%	4873s	27.4%/92.3%	3.5s
JIDCT(122)	21.3%/91.7%	6236s	30.3%/92.5%	7.6s
JFDCT(134)	24.6%/91.6%	6645s	31.5%/91.2%	9.2s
Average	20.4%/92.0%		28.0%/92.4%	
Triangle variation sources				
ARF(28)	20.5%/92.5%	3718s	23.5%/93.1%	1.3s
EWF(34)	21.0%/92.7%	349s	24.5%/93.6%	3.7s
FIR1(44)	22.4%/92.0%	4929s	27.9%/92.5%	3.9s
JIDCT(122)	23.4%/91.3%	6650s	31.6%/92.1%	8.1s
JFDCT(134)	24.8%/91.2%	7685s	32.5%/92.0%	8.2s
Average	22.4%/91.9%		28.0%/92.7%	
Gaussian variation sources				
ARF(28)	21.7%/92.6%	1574s	29.5%/92.7%	2.9s
EWF(34)	19.0%/93.0%	368s	28.3%/93.0%	5.2s
FIR1(44)	22.3%/92.1%	4683s	30.8%/92.6%	6.9s
JIDCT(122)	24.8%/92.0%	6509s	31.3%/92.1%	7.1s
JFDCT(134)	25.1%/91.0%	6535s	32.0%/91.8%	9.4s
Average	22.6%/92.1%		30.4%/92.4%	
Total Average	**21.8%/92.0%**		**28.8%/92.5%**	

REFERENCES

[1] S. Borkar, et al., "Parameter variations and impacts on circuits and microarchitecture," In *DAC*, pp. 338-342, 2003.

[2] M. Orshansky, et al., "Fast statistical timing analysis handling arbitrary delay correlations," In *DAC*, pp. 337-342, 2004.

[3] A. Ramalingam, et al., "An accurate sparse matrix based framework for statistical static timing analysis," In *ICCAD*, pp. 231-236, 2006.

[4] C. Visweswariah, et al., "First-order incremental block-based statistical timing analysis," In *DAC*, pp. 331-336, 2004.

[5] H. Chang, et al., "Parameterized block-based statistical timing analysis with non-Gaussian and nonlinear parameters," In *DAC*, pp. 71-76, 2005.

[6] S. Bhardwaj, et al., "A framework for statistical timing analysis using non-linear delay and slew models," In *ICCAD*, pp. 225-230, 2006.

[7] L. Cheng, et al., "Non-linear statistical static timing analysis for non-Gaussian variation sources," In *DAC*, pp. 250-255, 2007.

[8] M. Mani, et al., "Joint design-time and post-silicon minimization of parametric yield loss using adjustable robust optimization," In *ICCAD*, pp. 19-26, 2006.

[9] W.-L. Hung, et al., "Guaranteeing performance yield in high-level synthesis," In *ICCAD*, pp. 303-309, 2006.

[10] F. Wang, et al., "Variability-driven module selection with joint design time optimization and post-silicon tuning," In *ASP-DAC*, pp. 2-9, 2008.

[11] J. Jung, et al., "Timing variation-aware high-level synthesis," In *ICCAD*, pp. 424-428, 2007.

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

[13] T.-C. Chen, et al., "Modern floorplanning based on fast simulated annealing," In *ISPD*, pp. 104-112, 2005.

[14] ExPRESS Benchmarks at: *http://express.ece.ucsb.edu/benchmark/*