A Variation Aware High Level Synthesis Framework

Feng Wang, Guangyu Sun, Yuan Xie
The Pennsylvania State University, University Park, PA, USA
{fenwang, gsun, yuanxie}@cse.psu.edu

Abstract— **The worst-case delay/power of function units has been used in traditional high level synthesis to facilitate design space exploration. As technology scales to nanometer regime, the impact of process variations increases. The degree of variability encountered in the new process technologies makes worst-case analysis undesirable, because it may result in unexpected performance/power discrepancy or a pessimistic estimation, and may end up using excess resources to guarantee design constraints. In this paper, we propose a high level synthesis framework to take into account of the performance/power variation for function units. An effective metric called *parametric yield*, which is defined as the probability of the synthesized data flow graph (DFG) meeting the performance and power constraints, is used to guide scheduling, module selection, and resource sharing. An efficient performance/power yield perturbation computation method for DFG significantly improves the effectiveness of our yield driven high level synthesis algorithm. The experimental results show that our variation-aware synthesis framework achieves significant yield improvements, and has much faster (3X) runtime speed compared against previous approach.**[1]

I. Introduction

High-level synthesis (HLS) is the process of translating a behavioral description into a hardware implementation at register-transfer level [1] [2]. The design specification is usually written as a behavioral description, in languages such as ANSI C++ or SystemC. The behavioral description is first compiled into an internal representation (such as control and/or data flow graphs (CDFGs)), which are then mapped to the functional units that are selected from the resource library to meet design goals (such as performance, area, and power). The synthesis process usually consists of scheduling, module selection, and resource sharing [1].

The worst-case latency/power for each function unit has traditionally been used by high level synthesis algorithms to perform design space exploration. As technology scales, the reliance on deep sub-micron process technologies for the fabrication of circuits gives rise to concerns about process variations, which can cause significant performance and power variations for a function unit design. 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 nonviable. Under the influence of process variation, the existing deterministic worst-case design methodologies in high level synthesis may result in unexpected performance/power discrepancy or a pessimistic estimation, and may end up using excess resources to guarantee design constraints, due to overly conservative design approaches.

As a result of larger variability when technology scales, a shift in the design paradigm, from today's worst-case deterministic design to statistical or probabilistic design [3], is critical for deep sub-micron design. Industry and academia have already realized the need for such a shift in the design paradigm, and there are a lot of research activities [3]–[7] at various design levels to deal with process variations. Recently, variation aware design techniques have been developed for the embedded system [8] [9]. However, combating the process variation effects in high level synthesis has not gained much attention until very recent attempts. In this paper, we apply statistical timing/power analysis to high level synthesis, and develop yield driven synthesis framework so that the impact of process variations is taken into account during high level synthesis.

[1]This research was supported in part by NSF grants of CAREER 0643902, CNS 0720659 and CCF 0702617.

II. Related Work

Extensive researches in high level synthesis techniques have been done. Early work focuses on high level synthesis for data flow designs in an attempt to satisfy the design specifications and minimize delay, area, and power consumption. A variety of scheduling heuristics, resource sharing, and module selection techniques have been proposed to tackle performance/power/reliability/thermal challenges [10]–[13]. Since the scheduling, module selection, and resource sharing are strongly interdependent, simulation annealing and integer linear programming formulations techniques have been used to perform these subtasks simultaneously [1] [2].

The research on statistical analysis and optimization has gained great attention due to the increasing impact of process variation on the performance and power. Recently, a variety of gate-level statistical timing analysis and optimization techniques have been proposed. Fast statistical time analysis tools have been developed to replace the Monte Carlo analysis techniques [5] [14]. Based on the gate-level statistical timing analysis, variation-aware optimization techniques [15] [7] have been proposed at gate-level logic design. Recently, the challenges caused by process variation were realized by the high level synthesis community. For example, a high-level synthesis framework considering delay variation was proposed [16]. Jung et al. [17] proposed a timing variation aware scheduling and resource binding algorithm to reduce the latency. However, power variation was ignored in these works. In another work, the leakage power variation was analyzed in the context of high level synthesis [18] to minimize the average power. However, the timing/power model and analysis in that work is overly simplified. Consequently, the timing/power analysis in that approach can lead to large errors in estimating the path delay distribution and the power distribution. Wang et al. [19] proposed a module selection algorithm that combines design-time optimization with postsilicon tuning to maximize design yield.

Our contributions in this paper distinguish itself in the following aspects: 1) develop the power and performance yield constraint high level synthesis algorithms; 2) develop an efficient timing yield perturbation computation method, which significantly reduces the run time; 3) bring performance and power variation awareness into subtasks of the high level synthesis by performing yield driven module selection, resource sharing, and scheduling simultaneously.

III. Timing and Power Yield in High level Synthesis

High level synthesis usually consists several steps: scheduling, module selection, and resource sharing. *Scheduling* assigns each operation (such as add and multiply) in a CDFG to one or more clock cycles (or control steps). Scheduling techniques in HLS are usually classified as *time-constrained* scheduling or *resource-constrained* scheduling. *Module selection* decides the type of functional units to perform the operation in CDFG. *Resource sharing* uses the same resource (functional units or registers) to perform multiple operations or store more than one variable. These steps can interact with each other and affect the final synthesis results.

Traditionally, high level synthesis is performed under design constraints, which includes resource constraints, performance constraints, and power 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

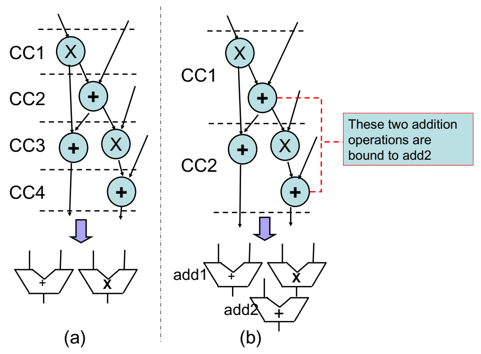

Fig. 1. A CDFG has two different schedules, with different clock cycle time and resource. Note that two addition operations chained to the multiplication operations are bound to the same adder (add2) in (b)

in the CDFG to finish the execution in a number of clock cycles (*latency constraints*) with a particular clock rate (*clock cycle time(CCT) constraints*); The *power constraints* require that the total power consumption of the synthesis result should be lower than a specified limit. Note that in this paper, we focus on *data-flow intensive* applications (represented by a DFG), in which most of the computations performed in the design are arithmetic operations (such as addition and multiplication).

In order to fully explore the design space, a diverse library of functional units is available for the high level synthesis tool. For example, to perform an addition operation, one can use a ripple-carry adder, carry-lookahead adder, or carry-select adder, each of which can be characterized by a $\{(delay, power)\}$ pair. Under the influence of process variations, the functional unit delay and power follow a probability distribution, individually. Therefore, $delay$ or $power$ for each function unit is not a fixed number, but a random variable. For example, a pass gate carry-select adder may be faster than a static carry-selected adder, but with larger delay variation.

A. Timing and Power Yield

To bring the process-variation awareness to the high level synthesis flow, we introduce a new metric called *parametric yield*. The parametric yield is defined as the probability of the synthesized hardware meeting a specified constraint $Yield = P(Y \leq Y\_max)$, where Y can be performance or power. 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. The *power yield* is defined as the probability that the total power of the synthesis result is less than the power limit under latency and resource constraints.

For each functional unit FU_i, the delay probability distribution $D_i(t)$ and power probability distribution $P_i(w)$ can be characterized using gate-level statistical analysis as described in Section II. The calculation of the parametric yield depends on scheduling, module selection, resource sharing, and clock selection. For example, in Fig. 1, the same CDFG can be either scheduled into 4 clock cycles with a much shorter clock cycle time T_S, or scheduled into 2 clock cycles with a longer clock cycle time T_L. The computation time is $4 \times T_S$ and $2 \times T_L$, respectively. In Fig. 1(a), the synthesized underlying architecture would be one multiplier and one adder, working in parallel, while in Fig. 1(b), the synthesized underlying architecture would be two adders and one multiplier, with the adders and multiplier connected in series (for illustration purpose, we ignore the possible multiplexers and registers in the synthesis result in this example). The performance yield for each of these two cases is calculated as Equation (1) and (2):

$$Y_a = (\int_0^{T_S} D_{add}(t)dt) \times (\int_0^{T_S} D_{mul}(t)dt) \tag{1}$$

$$Y_b = (\int_0^{T_L} D_{add2}(t)(\int_0^{T_L-t} D_{mul}(s)ds)dt) \times (\int_0^{T_L} D_{add1}(t)dt) \tag{2}$$

Note that equation (2) is valid with the assumption that the addition operations, which are chained to the multiplication operations in both CC1 and CC2, are bound to the same adder (*add2*) in Fig. 1(b).

The power yield for each case can be computed as the probability of the total power is less than the power limit. The total power is computed as the sum of each function unit's power (which is a random variable with power distribution $P_i(w)$).

The preceding example is only a simple illustration of the calculation of parametric yields. However, when the synthesis result is more complicated structure and delay/power distribution is complex, one may have to resort to the statistical analysis methods described later in Section IV and V.

B. Worst-case Approach vs. Yield-driven Statistical Approach

Although all the three steps can affect the performance/power yield of the synthesis results, in this section, we use module selection step as an example to compare the yield driven approach and worst-case deterministic approach. For example, the power distribution and delay distribution of two function units ($fu1$ and $fu2$) are shown in Fig. 2. The power limit (PL) and clock cycle time , T_{clk}, are given and shown in Fig. 2. The two methods are applied to this example:

- *Worst-case deterministic approach:* the faster function unit, $fu1$, will be chosen because the worst case delay of the slower function unit, $fu2$, exceeds the clock cycle time, $Tclk$. However, the power consumption of $fu1$ is much larger than that of the $fu2$ with smaller area. This module selection decision might lead to large power yield loss, as shown in the Fig. 2(b) and Fig. 2(d).
- *Yield-driven approach:* to maximize the power yield under the performance yield constraint, $fu2$ might be chosen, because the performance yield loss by selecting $fu2$ is relatively small compared to choosing $fu1$, as show in Fig. 2(a) and 2(c). However, this selection results in larger power yield, as shown in Fig. 2(b) and 2(d).

IV. Statistical Analysis in High level Synthesis

In traditional high level synthesis, performance and power analysis are based on worst-case analysis. In this section, we first present our statistical timing and power models for function units (including multiplexers and registers) in the library. Based on these models, statistical timing/power analysis of the synthesized DFG (which is the data flow graph with all operations are scheduled and bounded to function units) is described.

A. Statistical Timing Model and Analysis

Similar to the gate level statistical timing analysis [5], [14], [15], we use a first-order canonical model to model the delay for a function unit in the resource library. In this model, the delay of a gate is expressed as

$$D_m = d_0 + \sum_{i=1}^{n} d_i X_i + d_{n+1} X_m \tag{3}$$

where d_0 is the nominal delay of a gate. X_i and X_m are the independent normally distributed random variables to model the variations in process parameter variations. X_i is the correlated component of these variation parameters and X_m is the random component.

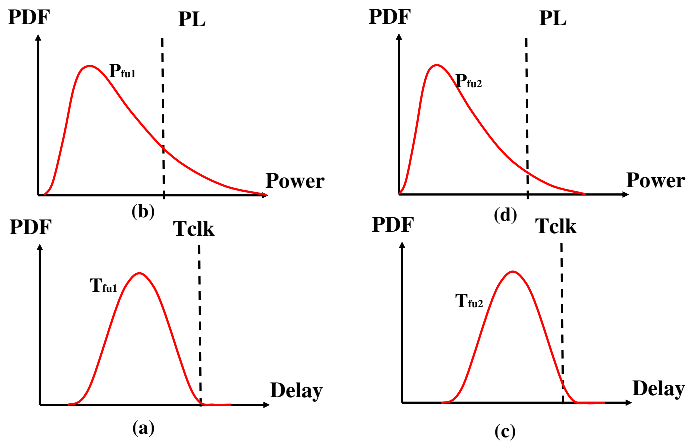

Fig. 2. Example: (a) and (b) show the delay and power distribution of the function unit 1, T_{fu1} and P_{fu1}. (c) and (d) show the delay and power distribution of the function unit 2, T_{fu2} and P_{fu2}. The performance yield is computed as $(\int_0^{T_{clk}} T_{fu}(t)dt)$ and the power yield is computed as $(\int_0^{PL} P_{fu}(w)dw)$.

In the statistical timing analysis, the max and sum operations are used to propagate the delay distribution in the synthesized DFG. The results of these two operations are maintained in the canonical form. Consequently, the delay of the circuit is also expressed in the canonical form. Thus, the delay of function units in high level synthesis can be expressed in the canonical form as in equation (3). For instance, in Fig. 1(a) , the max operation is performed over the delay values of adder and multipliers to obtain the delay distribution for each clock step; in Fig. 1(b), the sum operation is performed for the delay of adder and multiplier that are connected in series, to obtain the path delay from the input of the multiplier to the output of the adder. The results of the max and sum operations are also expressed in the linear canonical form [14]. For the sum operation, the coefficient of the result is the sum of the corresponding coefficient of the random variables. The max operation is performed using the moment matching method [20].

B. Statistical Power Model and Analysis

The statistical leakage power of a function unit can be expressed as

$$P_m = exp(a_0 + \sum_{i=1}^{n} a_i X_i + a_{n+1} X_m) \tag{4}$$

where $exp(a_0)$ is the nominal leakage power of a function unit. X_i and X_m are the independent normally distributed random variables to model the variations in process parameter variations. X_i is the correlated component of these variation parameters and X_m is the random component. The total power of the circuit is computed as summation of the power of all the components in the circuit. The result of the summation is also expressed in the same form as equation (4) [21]. The total power of the synthesized DFG is computed by iteratively adding the leakage power of the function units in the DFG. Assuming that the sum of the function unit's leakage power, $P_m = P_k + P_n$, the coefficients of P_m are determined as equation (5), (6) and (7) based on the Wilkinson's method [21].

$$a_i = log(\frac{E(P_k e^{X_i}) + E(P_n e^{X_i})}{(E(P_k) + E(P_n))E(e^{X_i})})\ \forall i \in [1, n] \tag{5}$$

$$a_0 = 0.5log(\frac{(E(P_k) + E(P_n))^4}{(E(P_k) + E(P_n))^2 + Var(P_k) + Var(P_n) + 2Cov(P_k, P_n)}) \tag{6}$$

$$a_{n+1} = [log(1 + \frac{Var(P_k) + Var(P_n) + 2Cov(P_k, P_n)}{(E(P_k) + E(P_n))^2}) - \sum_{i=1}^{n} a_i^2]^{0.5} \tag{7}$$

$E(P)$ represents the mean of the random variable P, $Var(P)$ represents the variance of the random variable P, and $Cov(P, Q)$ is the covariance of the random variables P and Q.

V. Yield Analysis in High Level Synthesis

In this section, we first introduce a parametric yield computation method for a synthesized DFG. We then present fast yield perturbation computation methods.

A. Yield Analysis for Synthesized DFG

After the resource allocation and binding, operations are bound to function units in library meeting the resource and latency requirements. In addition, the operation in each clock cycle has to meet the timing requirements in terms of clock cycle time. Thus, given the clock cycle time, $Tclk$, the timing yield of this DFG is computed as:

$$TimingYield(T_{DFG}) = Probability(T_{DFG} \leq Tclk|constraints) \tag{8}$$

The T_{DFG} is computed as:

$$T_{DFG} = max(T^1_{cycle}, T^2_{cycle}, ..., T^n_{cycle}) \tag{9}$$

where n is the number of the $POnode$s in the DFG. A $POnode$ is defined as the operation node that has no fan-out operation nodes within the same clock cycle. For instance, the two adder nodes at clock cycle $CC2$ in Fig.1 (b) are the $POnodes$ in that particular clock cycle. The T^i_{cycle} is the arrival time of the $POnode\ i$. The arrival time of the $POnode$ is computed using the statistical timing analysis method in Section IV-A. Note that the max operation is defined in Section IV-A. The *constraints* refer to the resource and latency requirements.

Given a power requirement, $Preq$, the power yield is computed as the probability that total power,P_{DFG}, is less than the requirement, as expressed in equation 10.

$$PowerYield(P_{DFG}) = Probability(P_{DFG} \leq Preq|constraints) \tag{10}$$

B. Fast Timing Yield Perturbation Computation

Based on the timing yield analysis, a brute-force approach to compute yield gain/loss due to the timing perturbation (caused by the module reselection and resource sharing during synthesis) can be used. The computation complexity of this approach is $O(n)$ for each timing perturbation, where n is the number of the $POnodes$, because the computation of the T_{DFG} in equation (9) requires $(n-1)$ *max* operations.

Since the yield computation has to be performed whenever there is a change in any synthesis steps (scheduling, module selection, or resource binding), and it resides in the inner loop of the optimization algorithm, as shown in Section VI-A and VI-C (the cost function in line 5 of Fig. 4), a fast yield perturbation computation approach is critical to the optimization algorithm.

In this work, a balanced binary tree based approach is developed to reduce the computation complexity of yield computation. In this approach, a balanced binary tree is constructed to avoid the re-computation in brute force approach. Each leaf node of this balanced binary tree is associated with a *POnode* in the DFG, and contains the arrival time of the *POnode*. Each non-leaf node contains the maximum value of the random variables of its children. Fig. 3 shows an example of a balanced binary tree for the yield perturbation computation. The DFG has eight *POnode*s and their corresponding arrival times are *f1-f8*. For each rebinding operation, the arrival time of the corresponding node is updated; meanwhile, the maximum random

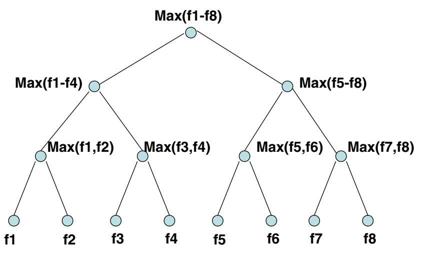

Fig. 3. Example of the binary tree based yield perturbation computation for a synthesized DFG.

variables in its parent nodes are also updated. During this updating process, the new T_{DFG} is computed. For instance, the operation 5 is rebound to a new function unit, which requires recomputing $max(f5, f6)$, and $max(f5-f8)$ and $max(f1-f8)$. This updating process requires $h-1$ operations, where h is the depth of the balanced binary tree. Since $T_{DFG} = max(f1 - f8)$, the new T_{DFG} is computed with $h-1$ max operations, and $h = log(n)$. Consequently, the computational complexity of updating T_{DFG} is $O(log(n))$, where n is the number of the $POnodes$. Resource sharing or splitting involves updating the leaf nodes, which are rebound to other function units, and their parent nodes, and the computational complexity of these perturbation is also $O(log(n))$. Thus, the overall computational complexity of the timing yield perturbation computation is reduced from $O(n)$ to $O(log(n))$.

C. Power Yield Perturbation Computation for DFG

The power yield perturbation due to the module reselection or resource sharing is computed based on the total power analysis. Assuming that N function units, fu_i^{old}, involves the change of module selection or resource sharing, and these function units are replaced with other K function units,fu_i^{new}, the total power after the module selection and binding is computed as:

$$P_{DFG}^{new} = P_{DFG}^{old} - \sum_{i=1}^{N} Pfu_i^{old} + \sum_{i=1}^{K} Pfu_i^{new} \tag{11}$$

P_{DFG}^{new} and P_{DFG}^{old} refer to the power of the DFG after the perturbation and that before perturbation. Pfu_i^{new} and Pfu_i^{old} represent the power of the function units, fu_i^{new} and fu_i^{old}, respectively. Since a relative small number of the function units are involved during a change in DFG, thus the computational complexity of the power yield perturbation is constant.

VI. Yield Driven High Level Synthesis

In this section, we first present the yield driven high level synthesis framework. We then describe the details of our simulated annealing (SA) based method, which includes two key functions: move generation function and the cost function.

A. Yield Driven Synthesis Framework

```
SA_Opt($synthesizedDFG$,$constraints$,$Library$){
1.  T<-Initial Temperature;
2.  do{
3.     While(Thermal Equilibrium){
4.        Identify_possible_move;
5.        Evaluate $\Delta Cost$;
6.        Accept($\Delta Cost$,Temperature)}
7.     Update_Temperature;
8.  }while(converge or cool down)
9. }
```

Fig. 4. The Pseudo Code of Yield Driven HLS

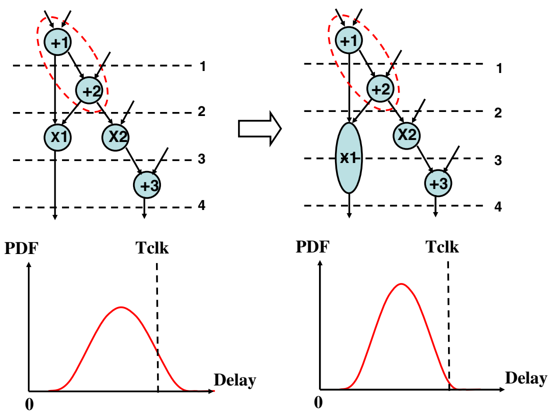

Fig. 5. Module selection example

Our yield driven high level synthesis framework, as shown in Fig. 4, takes an DFG, $constraints$ (latency constraint, resource constraint, CCT constraint, and power constraint), and a module $Library$ as inputs, and outputs a synthesized DFG that is power optimized while satisfying performance constraints. Since the subtasks of high level synthesis are strongly interdependent, simulated annealing is used to perform simultaneous scheduling, allocation, and binding. The SA technique is an effective method to explore the design space by allowing hill climbing moves [22]. Our algorithm begins with a synthesized DFG, where each operation is bound to the fastest function unit in the library and scheduled as soon as possible (ASAP). Based on the fast SA algorithm in physical synthesis [22], this SA based yield driven synthesis algorithm is characterized by the following functions:

1) Identify_possible_move function finds the possible moves (line 4), which will be described in detail in Section VI-B.
2) Accept function (line 6) accepts the move reducing the cost, or accepts the move with probability equals to $exp(-\Delta Cost/T)$ when the move increases the cost.
3) Update_Temperature updates temperature based on the current cost change (line 7).
4) The cost function (line 5) computes the cost associated with the current synthesized DFG, which will be described in detail in section VI-C.
5) The stopping criterion: (line 8).

B. Move Generation function

In order to fully explore the design space, the possible moves generated during the perturbation can be classified as four types [23]:

- **Rescheduling.** The operation is scheduled to other clock cycles. This type of move itself does not affect the timing and power yield. However, it can lead to other moves, such as module reselection and resource sharing.
- **Module reselection.** In this move, an operation is assigned to a different function unit in the library with different timing and power characteristics. For instance, a one-cycle function unit is replaced by a two-cycle one. The rescheduling might be performed to ensure that this move will not cause the violation of the latency constraints, if the latency of the function units in terms of clock cycles is changed. We illustrate this move in Fig. 5: the multiplication operation 1 is rebound to a two-cycle multiplier. The delay distribution of these function units are available in library as shown in Fig. 5. This type of move reduces the power if a faster function unit is replaced by a slower function unit. However, the timing yield of this move might be increased

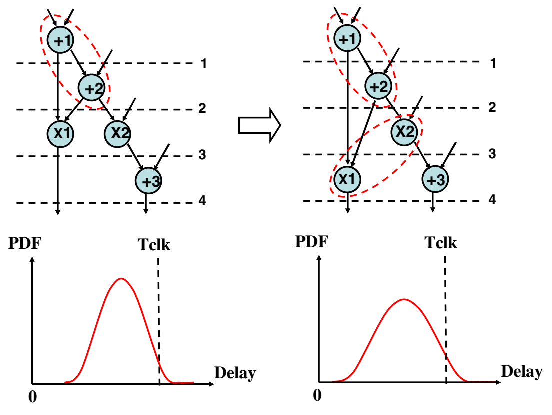

Fig. 6. Resource sharing example

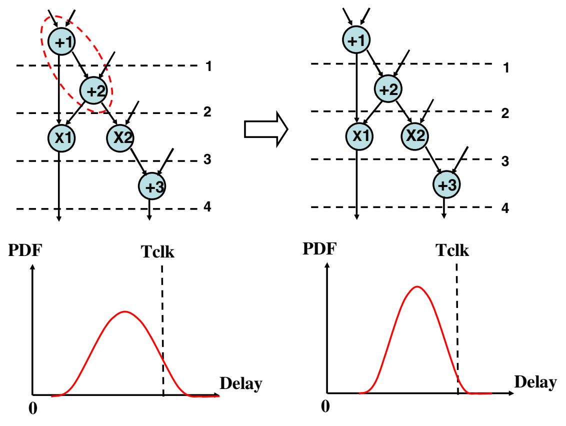

Fig. 7. Resource splitting example

or decreased depending on the delay distribution of the function units and the clock cycle time constraints. For example, in Fig. 5, the timing yield before the move is $p(Tf1 < Tclk)$, where $Tf1$ is the delay of the single cycle function unit. However, the timing yield after the move is $p(max(Tf2_{cycle1}, Tf2_{cycle2}) < Tclk)$, $Tf2_{cycle1}$ and $Tf2_{cycle2}$ represent the delay of the 2-cycle function unit in *cycle 1* and *cycle 2*, respectively. Assuming that $p(Tf2_{cycle1} < Tclk) = 0.9$, $p(Tf2_{cycle2} < Tclk) = 0.9$ and $p(Tf1 < Tclk) = 0.75$, the timing yield is improved since $p(max(Tf2_{cycle1}, Tf2_{cycle2}) < Tclk) <= p(Tf2_{cycle1} < Tclk)$*$p(Tf2_{cycle2} < Tclk) = 0.81$ due to the positive correlation between $Tf2_{cycle1}$ and $Tf2_{cycle2}$.

- **Resource Sharing.** In this move, two function units are merged into a single function unit. For example, function unit *f1* and function unit *f2* are replaced with function unit *f3*. In this work, this type of move is performed under the condition that the merged function unit *f3* can perform the operations, which are originally bound to *f1* and *f2*, and has the same latency as the original function units. In addition, these two function units can not scheduled in the same clock cycle.
We illustrate this type of move in Fig. 6. In this example *multiplier 1* and *multiplier 2* are merged into a single multiplier. This resource sharing move is preceded by the rescheduling of the *operation 1*. This type of move can effectively reduces the power. The overhead introduced by this move are additional multiplexes, which are required at the inputs of *multiplier 1* and *multiplier 2*. However, the timing yield might be increased and reduced depending on the distributions of the function units and the multiplexes. In Fig. 6, the timing yield before the move is $p(max(Tf1, Tf2) < Tclk)$; the timing yield after move is $p(Tf3 + Tmux < Tclk)$. Assuming that $Tf1$ and $Tf2$ are independent and $p(Tf1 < Tclk) = 0.9$ and $p(Tf2 < Tclk) = 0.9$, and $p(Tf3 + Tmux < Tclk) = 0.85$, the resource binding lead to timing yield improvement since $p(max(Tf1, Tf2) < Tclk) = 0.81$.
- **Resource Splitting.** In this move, a single function is split into two function units. The operation originally shared with other operations is split from the shared function unit. This type of move might lead to other moves, such as module reselection and resource sharing. For example, in Fig. 7, the operation $+1$ is split from the single cycle adder and it is possible to rebound it to a 2-cycle adder, which requires rescheduling; this move might lead to resource sharing with operation $+3$. The power yield of this move is increased and the timing yield might also be increased as shown in analysis in the resource sharing move.

C. Cost function

The cost function for our yield driven synthesis algorithm consists of two portions: the first portion is related to the power yield loss; the second portion is the penalty of the timing yield violation.

$$Cost = (1 - poweryield) + f(timingyield) \tag{12}$$

The timing yield violation penalty function, $f(timingyield)$ is 0 if *timingyield* is larger than $(1 + \alpha)\times$ $timing\_yield\_contraints$; otherwise, $f(timingyield)$ is

$$[(timingyield - timing\_yield\_contraints) * scale]^2/scale \tag{13}$$

where α, a small value, is used to guarantee the time yield when the temperature approaches zero. When the timing yield is violated, the penalty is proportional to the square of the quantity of the amount of the timing yield violation. The *scale* is a constant value. Experimental results indicate that this cost function is able to guarantee the timing yield while effectively exploring the timing yield and power yield tradeoff as shown in Section VII.

VII. Analysis Results

In this section, we present the analysis results. The results show that our method can effectively reduce the impact of the process variation and maximize the parametric yield, significantly outperforming traditional deterministic methods that use worst-case models. Note that in the deterministic synthesis methods, the slack is used to guide the design exploration and the timing qualities are computed using worst case delay models.

We implement our yield driven high level synthesis algorithm in C++ and conduct the experiments on six high level level synthesis benchmarks: a 16-point symmetric FIR filter (FF), a 16-point elliptic wave filter (EWF), an autoregressive lattice filter (ARF), an algorithm for computing Discrete Cosine Transform (DCT), a differential equation solver (DES), and an IIR filter (IIR). In this paper, we evaluate the effectiveness of our algorithm under different performance yield constraints: 99%, 90% and 85%.

To demonstrate the yield improvement of our method, we compare the results of our yield driven synthesis (YD) against traditional deterministic synthesis methods using worst case (WC) models. In Tables I, II and III show the results of our method (YD) against those of the deterministic synthesis technique (WC), under 99%, 95% and 85% performance yield requirement, respectively. In the first column, we show the benchmarks we used in this analysis. From the second column to the third column, we show the absolute power yield results of the process yield driven high level synthesis method (YD) and the deterministic worst case high level synthesis technique (WC), respectively. In the fourth column, we show the absolute value of the yield improvement of our method over worst case method. In the fifth column, we show the relative yield improvement of our method over worst case method. As can be seen from Tables I, II and III, significant yield improvement could be obtained if we take into account the variation in high level synthesis. The yield results show that WC based technique is pessimistic with average 31%, 40% and 48% power yield loss under the 99%, 95% and 85% performance constraints, respectively. It is because worst-case analysis without taking the probabilistic information into account can result in a pessimistic estimation and end up using excess resources (with excess power) to guarantee performance constraints.

Compared to the previous work [16], which only considers delay variation and each synthesis step in [16] was still deterministic, our approach considers both delay and power variation, and performs each synthesis subtask statistically. Furthermore, even though both [16] and our approach are based on simulated annealing approach, in this work, the fast yield computation method and the proper cost function greatly improve the runtime of the algorithm. The average runtime across the benchmarks is about 3 times faster than the approach in [16], which has not considered the power variability yet. Such runtime improvement lies in the effectiveness of the cost function. The cost function in [16] is hard to guarantee the timing yield, thus affecting the design space exploration. In addition, the computation complexity of the yield gain/loss computation in [16] is $O(n)$, while the computation complexity of our method is $O(log(n))$.

TABLE I
POWER YIELD UNDER 99% PERFORMANCE YIELD CONSTRAINT

Name	YD	WC	YD-WC	(YD-WC)/WC
AR	94%	67%	27%	40%
DES	94%	82%	12%	15%
EWF	77%	44%	33%	75%
FF	94%	82%	12%	15%
DCT	94%	82%	12%	15%
IIR	77%	44%	33%	75%
Average	88%	67%	21%	31%

TABLE II
POWER YIELD UNDER 95% PERFORMANCE YIELD CONSTRAINT

Name	YD	WC	YD-WC	(YD-WC)/WC
AR	97%	67%	30%	44%
DES	97%	82%	15%	18%
EWF	88%	44%	44%	100%
FF	97%	82%	15%	18%
DCT	97%	82%	15%	18%
IIR	86%	44%	42%	97%
Average	94%	67%	27%	40%

TABLE III
POWER YIELD UNDER 85% PERFORMANCE YIELD CONSTRAINT

Name	YD	WC	YD-WC	(YD-WC)/WC
AR	100%	67%	33%	48%
DES	100%	82%	18%	22%
EWF	98%	44%	54%	124%
FF	100%	82%	18%	22%
DCT	100%	82%	18%	22%
IIR	98%	44%	54%	124%
Average	99%	67%	32%	48%

VIII. CONCLUSIONS

In this paper, we formulate the power and performance yield constraint high level synthesis problem and propose an efficient algorithm to solve it. We develop the performance and power parametric yield computation methods, and a fast performance yield gradient computation scheme. Based on the fast yield perturbation computation method, the performance and power yield aware high level synthesis framework is developed. Simulation results show that significant yield gain can be obtained with our yield driven high level synthesis framework. Our future work is to incorporate the process variability into the design flow of a state-of-the-art high level synthesis tool called Catapult-C, with the support from Mentor Graphics.

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] F. N. Najm. On the need for statistical timing analysis. In *Design Automation Conference*, pages 764–765, 2005.

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

[5] S. Sapatnekar. *Timing*. Kluwer Academic Publishers, 2004.

[6] D. Marculescu and E. Talpes. Variability and energy awareness: A microarchitecture-level perspective. In *Design Automatin Conference*, pages 11–16, 2005.

[7] A. K. Singh, M. Mani, and M. Orshansky. Statistical technology mapping for parametric yield. In *International Conference on Computer Aided Design (ICCAD)*, pages 511–518, 2005.

[8] D. Marculescu and S. Garg. System-level process-driven variability analysis for single and multiple voltage-frequency island systems. In *Proc. IEEE/ACM Intl. Conference on Computer-Aided Design (ICCAD)*, Nov. 2006.

[9] Feng Wang, C. Nicopoulos, X. Wu, Yuan Xie, and N. Vijaykrishnan. System-level process variation driven throughput analysis for single and multiple voltage-frequency island designs. *Proc. of ICCAD*, Nov. 2007.

[10] 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.

[11] 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.

[12] R. Karri and A. Orailoglu. Time-constrained scheduling during high-level synthesis of fault-secure vlsi digital signal processors. *Reliability, IEEE Transactions on*, 45(3):404–412, 1996. 0018-9529.

[13] 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.

[14] 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.

[15] A. Agarwal, K. Chopra, D. Blaauw, and V. Zolotov. Circuit optimization using statistical static timing analysis. In *Design Automation Conference*, pages 321–324, 2005.

[16] W.-L. Hung, X. Wu, and Y. Xie. Guarantee performance yield in high level synthesis. In *International Conference on Computer Aids Design*, 2006.

[17] Jongyoon Jung and Taewhan Kim. Timing Variation-Aware High-Level Synthesis. *Proc. of ICCAD*, November 2007.

[18] S. P. Mohanty and E. Kougianos. Simultaneous power fluctuation and average power minimization during nano-cmos behavioral synthesis. In *Proc. of VLSID*, pages 577–582, 2007.

[19] Feng Wang, X. Wu, and Yuan Xie. Variability-driven module selection with joint design time optimization and post-silicon tuning with adaptive body biasing. *Proc. of ASPDAC*, Nov. 2008.

[20] C. Clark. The greatest of a finite set of random variables. *Operations Research*, pages 145–162, 1961.

[21] A. Srivastava, S. Shah, K. Agarwal, D. Sylvester, D.Blaauw, and S. Director. Accurate and efficient gate-level parametric yield estimation considering correlated variations in leakage power and performance. *Design Automation Conference (DAC)*, pages 535–540, 2005.

[22] Tung-Chieh Chen and Yao-Wen Chang. Modern floorplanning based on fast simulated annealing. In *Proc. of ISPD*, pages 104–112, 2005.

[23] Anand Raghunathan and Niraj K. Jha. An iterative improvement algorithm for low power data path synthesis. In *Proc. of ICCAD*, pages 597–602, 1995.