Guaranteeing Performance Yield in High-Level Synthesis

W.-L. Hung, Xiaoxia Wu, and Yuan Xie
Computer Science and Engineering
The Pennsylvania State University, University Park, PA 16802, USA
{whung,xwu,yuanxie}@cse.psu.edu

ABSTRACT

Meeting timing constraint is one of the most important issues for modern design automation tools. This situation is exacerbated with the existence of process variation. Current high-level synthesis tools, performing task scheduling, resource allocation and binding, may result in unexpected performance discrepancy due to the ignorance of the impact of process variation, which requires a shift in the design paradigm, from today's deterministic design to statistical or probabilistic design. In this paper, we present a variation-aware performance yield-guaranteed high-level synthesis algorithm. The proposed approach integrates high-level synthesis and statistical static timing analysis into a simulated annealing engine to simultaneously explore solution space while meeting design objectives. Our results show that the area reduction is in the average of 14% when 95% performance yield is imposed with the same total completion time constraint.

1. INTRODUCTION

Aggressive technology scaling presents challenges to fabricate small feature size transistors, and results in significant variations in transistor parameters such as channel length, gate-oxide thickness, and threshold voltage across identically designed neighboring transistors (*intra-die variation*) and across different identically designed chips (*inter-die variation*). Process variations are primarily due to uncertainty in the device and interconnect characteristics, such as gate length, the thickness of gate oxide, and doping concentrations. Although designing for worst-case process margins has been used as a traditional option when dealing with outliers, the degree of variability encountered in the new process technologies makes this a nonviable option. For example, variation on transistor parameters causes 20X variation in chip leakage and 30% variation in chip frequency [9]. It is inevitable to move the design methodologies from deterministic to probabilistic [9].

High-level synthesis (HLS) is the process of translating a behavioral description into a register level structure description [11, 12]. This process consists of resource allocation, binding, scheduling, and clock selection. Among these tasks performed during synthesis, scheduling determines the execution sequence of operations in terms of control steps. The scheduler tries to schedule as many operations as possible in the same control step in attempt to extract more parallelism. Thus, the clock cycle time plays an important role in deciding the overall system performance. The clock selection itself alone complicates HLS optimization, the complication of HLS is even exacerbated when variation is presented due to the fact that different clock cycle times result in a variety of clock latency distributions.

The resource library is provided to the HLS algorithm to optimize design goal, either area or latency, or both. The worst-case latency for each function unit has traditionally been used for HLS algorithm to perform design space exploration. However, it is becoming inapplicable as the timing variations continue to increase. For example, the delay variation for an adder may be up to 27% of its mean value [22]. Thus, under the influence of process variation, the analysis that uses worst-case deterministic delays for the functional units can overestimate the resource needed to meet a certain performance goal, due to the statistical variation in timing. To mitigate the variation induced problems in HLS, the statistical static timing analysis (SSTA) model is essentially needed to ensure timing correctness. Although SSTA was intended to model the delay distribution of a gate-level design, it is also suitable to model the latency distribution of a control data flow graph (CDFG) in high-level synthesis. In SSTA, the functional delays are random variables. The concept of "fixed time unit" [1] is adapted to discretize the delay random variables with respect to the selected clock cycle time. This process transforms the probability density function (PDF) of each functional unit delay into that of PDF in clock cycle resolution without losing correct timing information while reducing the complexity of statistical static timing analysis.

While much work has been done recently to develop statistical timing analysis methods [1, 2, 3, 4, 10] focusing on gate-level optimization, process variation-aware high-level synthesis has not been attempted. In this paper, we propose a HLS framework which tries to simultaneously satisfy performance yield and area constraints under variability. To the best of our knowledge, this is the first process variation-aware HLS framework to guarantee the performance yield requirements. The framework is based on a well-defined optimization technique, Simulated Annealing [13], and has taken the clock selection into account, which translates an already complicated HLS problem into a three-dimensional problem (area, latency, and clock). The rationale to include the clock selection is that process variation makes the delay of each functional unit become a probabilistic distribution instead of a deterministic value. As we will show in the latter section, by cleverly choosing the clock cycle time, the delay of a functional unit can display different delay distributions in terms of the selected clock cycle time. Thus, some of the units become variation free and some become less variation sensitive. Our SA-based HLS integrates such facility as one perturbation move with others in seeking a global optimal

ICCAD'06, November 5-9, 2006, San Jose, CA

solution meeting the yield constraint. To further improve the computational efficiency, a fast simulated annealing scheme is adopted to speed-up the annealing process and thus more solution space can be explored.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 gives the detailed explanation of our performance yield-guaranteed high-level synthesis. Section 4 presents and discusses experimental results. We conclude this paper in the last section.

2. RELATED WORK

While the high-level synthesis is a well-studied problem [11, 12], none of work takes process variation into consideration. While the concept of variable latency components is considered by Raghunathan et al. [14], extra hardware is needed at the input side for predicting the latency at output in order to capture the latency variation based on the input pattern. Unfortunately, the influence of real variable latency introduced by process variation can not be captured by such an approach. High-level synthesis for low power has attracted significant attention [20, 21]; however, power savings from using deterministic supply voltages and fixed transistor parameters is not necessarily as effective as expected since process variation introduces a wide range of possible values on these originally fixed values. Physical information has also been included by incorporating floorplanning algorithms into HLS [15, 18, 19] in order to meet the driving force of deep submicron technologies. The relative location information from floorplanning is useful for determining the correlation of the intra-die variation among functional units; unfortunately, none of the above work makes use of this facility.

Some of recent HLS work [15, 16] target on temperature and reliability issues whereas the effect of process variation is still neglected. Many statistical timing analysis algorithms [2, 4] combating variability have been proposed and are effective when applied in gate-level synthesis. Although the same fundamental idea is to consider process variation during synthesis, the divergence raises in that the allocated resource can be shared and the sequencing order of tasks with respect to clock cycle time must be enforced in HLS, which makes performance yield-guaranteed high-level synthesis a unique problem.

To the best of our knowledge so far, it has not been attempted to incorporate process variation consideration into higher level synthesis tasks. In this work, we aim to accomplish this goal by integrating statistical static timing analysis, developing resource binding and clock selection algorithms so that the impact of process variation is minimized during high-level synthesis.

3. PERFORMANCE YIELD-GUARANTEED HIGH-LEVEL SYNTHESIS

In this section, a new high-level synthesis (HLS) framework is proposed to improve the *performance yield* of a design under process variation. The new HLS algorithm consists of resource allocation, binding, and scheduling. The clock selection facility is also equipped to guarantee the performance yield while satisfying a mix of area, latency, and clock optimization objectives. As the constrained scheduling problem and the resource selection problem have already been shown in the literature to be NP-complete [11], heuristics are needed to solve the optimization problem in a computationally efficient manner. In this work we have adopted a well-studied combinational optimization method, Simulated Annealing (SA) [13].

The complete flow of our HLS is depicted in Figure 1. First, we apply the clock slack and timing event minimization algorithm to determine the range of candidate clocks. One of three perturbation moves is then applied inside SA and followed by the scheduling algorithm. Next, we perform the statistical static timing analysis on the CDFG and find out latency distributions at the outputs of functional units. The performance yield constraint is enforced on the latency distributions and the cost function is evaluated. A fast annealing schedule is put into action to facilitate the whole simulated process until the terminating criterion is met. The best HLS solution encountered is generated at the final stage. Each subtask inside the flow will be explained in the following subsections.

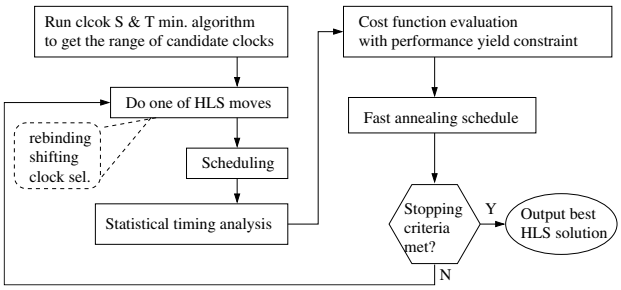

Figure 1: Flow of the SA-based HLS.

3.1 Statistical Static Timing Analysis

In order to consider the performance yield under process variability, statistical static timing analysis (SSTA) is needed. The effectiveness and efficiency of the methods of SSTA have been proposed and demonstrated by many recent researchers [2, 3, 4, 10]. However, most of these techniques mainly focus on continuous delay variables, which is not compatible with the discrete clock cycle time used in HLS. Thus, we need to first translate the delay of the functional unit in continuous form into that of in discrete form, and this will be explained in the following.

3.1.1 Discretizing Functional Unit Delay

Under the influence of process variation, the delay of a functional unit, instead of a deterministic value, is now of a distribution form. Figure 2 shows the delay variation for a carry-ripple adder implemented in 130 nm technology, with a 15% of inter-die and intra-die V_{th} variation. IBM has shown that the delay variation for different adder designs in 90 nm technology could be 21-27% of the mean values [22]. To model delay variation, we have adopted the discretizing approach proposed by Liou et al. [1]. Their approach was the first one to address statistical static timing analysis (SSTA) by using discretized probability density functions (PDFs) to handle delay probability distributions. While the algorithm was originally designated for modeling cell-based designs, we convert it to characterize the functional unit delay used in HLS. In this approach, a fixed time unit is needed during discretization. Since the clock cycle time (CCT) is a deterministic value and the selection of clock is integrated in our proposed HLS, it is suitable for discretizing continuous delay variables. We will show an example later to demonstrate that the choice of the clock cycle time has profound effects on the translated delay distributions.

In our analysis, the effect of process parameter variations on the delay of a functional unit is pre-characterized and accessed on the fly during statistical static timing analysis. The pre-characterization resource library contains statistical information on the delay of a functional unit when process variation is introduced.

3.1.2 SSTA for Control Data Flow Graph in HLS

In HLS, each operation (such as add and multiply) in CDFG is scheduled to one or more cycles (or control steps). Each control

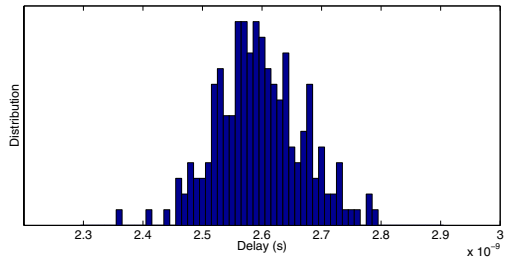

Figure 2: The delay variation of a carry-ripple adder implemented in 130nm with 15% of inter-die and intra-die V_{th} variation and 1000 Monte Carlo runs.

step corresponds to a time interval equal to the clock period. As a result of process variation, the functional units to implement the operations may have a large delay variation, and may not work at a particular clock frequency. Thus, under the effects of variability, statistical static timing analysis is essentially needed in HLS so as to meet the required performance yield.

The performance yield metric, which is the probability that the fabricated hardware can work at a particular clock frequency, is used to guide the high-level synthesis. For each functional unit FU_i, the delay probability distribution $D_i(t)$ can be provided either from simulation or from using gate-level statistical timing analysis. The calculation of the performance yield depends on module selection, clock selection, scheduling, and resource binding. For example as shown in Figure 3, the same CDFG can be either scheduled into 4 clock cycles with a much shorter clock cycle time T_{short}, or scheduled into 2 clock cycles with a longer clock cycle time T_{long}. The computational time is 4 $\times$ T_{short} and 2 $\times$ T_{long}, respectively.

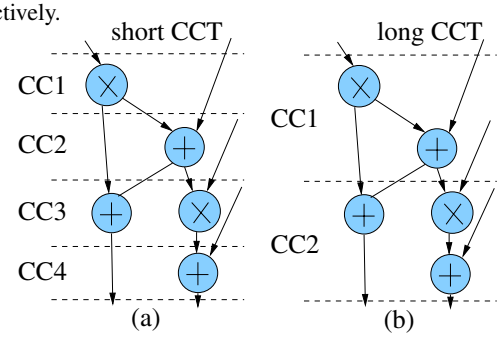

Figure 3: A CDFG has two different schedules, with different CCTs.

During statistical static timing analysis, the signal arrival time is calculated at each functional unit by propagating the delay distributions from the outputs of previous functional units. In the case of propagating one single event group, the sum operation is needed as in SSTA to combine the arrival delay distribution at the input of the functional unit with the functional unit delay distribution. This situation is shown in Figure 4, where a sum operation requires first performing shift with scaling and then grouping all probabilities of the same timing together. Note that, the notation for the probability of the output at time t is equal to the probability of outputting at t divided by the summation of the probabilities of all output events; e.g. probability of 1/4 for all timing events for the top-right event group in the figure.

In case of multiple inputs, the max operation is performed after the sum operations and the latest arrival time should dominate. The rationale behind this is based on *Bayes' Theorem*. The theorem states that $P(B) = \sum_i P(B|A = i) \cdot P(A = i)$, where

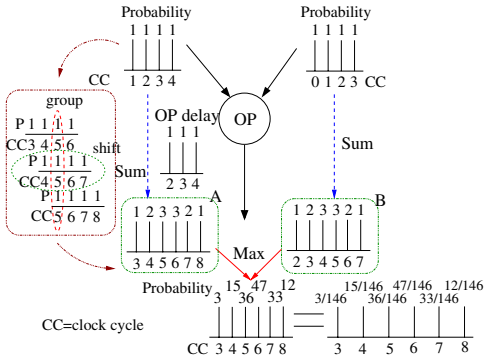

Figure 4: Statistical timing propagation on the operation node.

$P(B|A = i)$ refers to the probability of event B, given that $A = i$. As an example in the figure, the probability of timing event of 4 after the max operation is calculated as 15=2*(1+2)+3*(1)+2*3. The term, 2*(1+2), means that the probability of event group A dominates at output time 4. In order to make this assumption being true, we need to make sure unit B should arrive at timings smaller than 4; hence, the numbers inside the parenthesis accumulate such probabilities. Next, we need to count the probability if the event group B dominates, which is the term of 3*(1). Finally, the probabilities of both event groups arrive at the same time of 4, which is the term, 2*3. Other probabilities of different arrival times can be obtained in the similar fashion.

3.2 Clock Cycle Time Selection

One of the tasks in high-level synthesis is to determine the jobs needed to be done at each clock cycle. Therefore, the choice of clock cycle time plays an important role in HLS. Including the clock selection complicates the HLS problem and thus turns the problem into a three-dimensional problem (area, latency, and clock).

The delay of a functional unit is not deterministic under the influence of process variation and instead is a delay distribution with probabilities as shown in Figure 5. Based on different clock cycle times, the resulting probabilities and the required clock cycles will diversify accordingly. As two functional units, A and B, shown in Figure 5, for the clock cycle time of 12, unit A has two timing events. The first has the probability of 6/9 to finish in one clock cycle while the second has 3/9 probability, on the other hand, to finish in two clock cycles. In the case of CCT being 15, although the job in unit A can be done within one clock cycle, which means the unit is variation free under this CCT, functional unit B only requires CCT to be 12. Although both units are now free of variation, the slack of unit B is high and therefore the actual utilization rate of unit B is low. From this example, we know that slack only consideration is not sufficient to address the variation problem and the timing event should be taken into account as well.

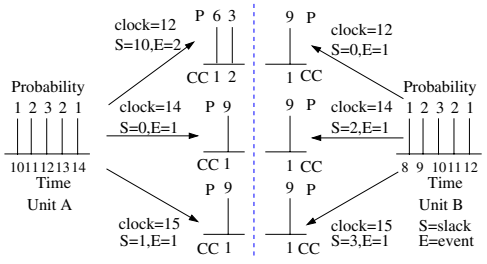

Figure 5: An example of clock selection.

However, randomly choosing the clock cycle time without first identifying possible candidate clocks leads to exhaustive searching, which is highly undesirable. Additionally, if the selected CCT

is too small, it may not be practically possible to implement the functional unit, and if the CCT is too large, then most of time functional units will stay idling. Thus, we have applied the clock slack minimization algorithm [5, 6] and modified it to include the concept of timing events that is directly resulted from the impact of the process variation. This algorithm is shown in Figure 6. In the experiment later, we will apply this algorithm with the smallest and with the fastest resource settings to determine the upper and the lower bounds for the clock cycle time. This CCT range is then used in our SA-based HLS as specified limits where clock perturbation function can vary within. Under this situation, our clock selection space contains both performance and area efficient solutions with minimum slack and less variation.

```
Clock slack and timing event minimization algorithm
begin
    best = 0;
    compute CLK_upper and CLK_lower;
    for all clk_i, CLK_lower ≤ clk_i ≤ CLK_upper
        compute slack(clk_i, t_j) and
        compute timing_event(clk_i, t_j) =
          ⌈(Delay_max(t_j) ÷ clk_i)⌉ − ⌈(Delay_min(t_j) ÷ clk_i)⌉ + 1,
        for all operators of type t_j;
        average_slack
            = [ (Σ_{j=1}^{M} Occur(t_j) × slack(clk_i, t_j)) / (Σ_{j=1}^{M} Occur(t_j)) ];
        average_timing_event
            = [ (Σ_{j=1}^{M} Occur(t_j) × timing_event(clk_i, t_j)) / (Σ_{j=1}^{M} Occur(t_j)) ];
        clock_suitability_{clk_i} =
          1 − ( ((average_slack * average_timing_event)) / clk_i );
        if(clock_suitability_(clk_i) > best)
                best = clock_suitability_(clk_i);
                CLK_best = clk_i
        endif;
    endfor
    return CLK_best
end
```

Figure 6: Outline of the clock slack and timing event minimization algorithm.

In the algorithm, CLK_{upper} is the largest delay value of all operation types while CLK_{lower} can be specified by the given resource library as long as a bistable transition is maintained. In our experiment setting, we first set the smallest delay value of all operation types as CLK_{lower}. Then, if the value of CLK_{upper}/c, where c is a constant, is smaller than CLK_{lower}, then CLK_{lower} = CLK_{upper}/c; otherwise, CLK_{lower} is not changed. From the experiment conducted, c=3 gives the best results. $Delay_{min}$ and $Delay_{max}$ represent the earliest and the latest arrival timing events in an event group, respectively. The calculation of $slack(clk_i, t_j)$ is formated as $((\lceil(Delay_{max}(t_j) \div clk_i)\rceil) \times clk_i)$ - $Delay_{max}(t_j)$ while $Occur(t_j)$ represents the number of occurrences of operator type j. Applying the clock slack and timing event algorithm to the example in Figure 5, we have *clock_suitability* 0.375 and 0.866 for CCTs 12 and 15, respectively, while the *clock_suitability* is 0.928 for the CCT of 14. This indicates that the selection of 14 for clock cycle time brings high utilization and less variation sensitive setting. The objective of adopting this algorithm is to provide a good starting point for our SA-based HLS in seeking a performance yield satisfied design.

3.3 Scheduling Algorithm

High-level synthesis involves three subtasks, which are resource allocation, binding, and scheduling. Both the resource allocation and the binding are taken care of by the moves used in our simulated annealing engine, which will be detailed in the following section. For the scheduling task, we have adopted the scheduling algorithm [8] for this purpose. The outline of the scheduling algorithm is depicted in Figure 7. Note that, we use the worst case delay for each functional unit when performing ASAP (As Soon As Possible) algorithm; that is, the latest arrival time of an event group.

In this algorithm, the latency is first determined by using ASAP scheduling and then the latency is denoted as "partition" afterward. The density is then assigned to each partition where the number of partitions equal to the length of the latency. The density is defined as $D_{ij} = c/(ALAP_i - ASAP_i + 1)$ where c is a constant. The meaning of density is to decide the probability that a node can be scheduled at one specific partition. So the total density for a particular partition j is given as $\sum_i^{\#nodes} D_{ij}$. Note that, only such node whose earliest time (by ASAP) is smaller than or equal to, and latest time (by ALAP) is larger than or equal to j will be considered. Next, we schedule the least free node i at the least density partition j and update the mobilities of unscheduled nodes. The mobility is defined as the difference of ALAP and ASAP. This process continues until the sequencing for all nodes is finalized. The objective of this scheduling algorithm is to distribute nodes evenly among partitions such that the number of resources used are minimized in the final design.

```
Scheduling algorithm
begin
    given CDFG and resource library;
    determine the minimum latency by using ASAP;
    for each resource type k
        done = false;
        while(not done);
            calculate the density for each partition;
            partition i = find the least dense partition;
            node j = find the least free node from partition i;
            schedule node j at current partition i;
            adjust the mobilities for unscheduled nodes;
            if(all nodes of type k scheduled)
                done = true;
            else
                done = false;
    endfor
end
```

Figure 7: Outline of the scheduling algorithm.

3.4 Simulated Annealing Engine

Simulated annealing (SA) is an iterative technique for solving optimization problems and has often been used in solving large-scale problems, such as floorplanning and placement. SA-based HLS is also in practice and is demonstrated in the recent literature [17, 18]. During the optimization process, one solution switches to another by using the perturbation operations in a well-defined way and thus the best solution can be obtained after evaluating a large number of different solution configurations. The inputs to our SA engine are the CDFG of an application and a resource library. The scheduling algorithm described above is used inside the SA engine to perform the actual scheduling on CDFG. Note that, the types of resources (resource allocation and binding) are chosen through the perturbation operation before the scheduling is started. The scheduler and statistical timing analyzer are then executed to evaluate the goodness of the current solution.

For high-level synthesis with the performance yield constraint, we have defined three perturbation operations which are listed below:

(1) Resource re-binding,
(2) Operation shift (single shift and all shift),
(3) Clock selection.

Each perturbation move will be detailed in the following subsections.

3.4.1 Resource Re-binding

The first operation is the resource re-binding which serves the role of resource allocation and binding in the traditional HLS. The purpose of this operation is to minimize the resulting latency variation and area consumption. Two cases for re-binding are considered and are shown in Figure 8. In the figure, node j is being considered for resource re-binding while node i is the predecessor of node j. Note that, all event groups, (1), (2), (3), and (4) show the resulting event distributions after executing the sum operation. For case A, we can easily tell the event group (2) is better since the latest arrival time and the event group are both smaller. However, it is not trivial to decide which one is better for case B. Hence, we introduce the product of standard deviation (σ) and mean (μ) to determine the selection of the resource. The reason why we use this product is that the standard deviation can tell how spread a distribution is while the mean value determines an average value of a distribution. Thus, the smaller $\sigma \times \mu$ value is preferred. The value of $\sigma \times \mu$ are 2.668 and 1.75, respectively, for event groups (1) and (2) in case A while are 2.668 and 2.75 for event groups (3) and (4) in case B. Thus, we choose event group (2) in case A while event group (3) in case B.

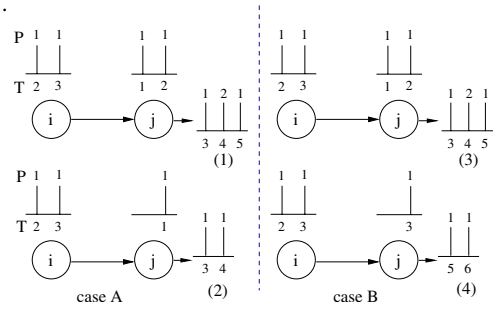

Figure 8: Two cases for re-binding operation.

3.4.2 Operation Shift

The second operation, operation shift, tries to move around the non-critical nodes of a CDFG such that the overall resource usage can be further reduced. One example is shown in Figure 9. The CDFG is shown in the left side of the figure. We can see that both nodes 8 and 9 are not on the critical path and thus their starting steps can be delayed to the later clock cycles so that the scheduling algorithm has the chance to find a lower area schedule. For operation shift, two different shifts, single shift and all shift, are defined. Single shift only moves one node at a time while all shift moves the node with its successors until the latest time constraint is violated. As shown in the figure assuming all nodes are of the same functional type, the number of units needed for CC1 and CC2 reduce from 4 to 3 units after all shift operation. Although nodes 8 and 9 are now scheduled at CC3 and CC4, respectively, the actual resource usage does not increase due to resource sharing. Thus, the shift operation can serve as the area reduction move.

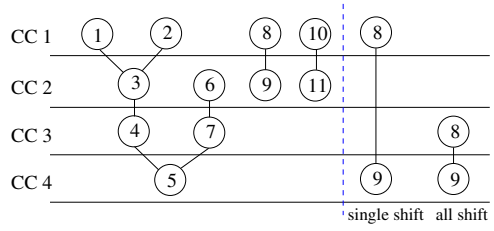

Figure 9: One CDFG example with two shift operations.

3.4.3 Clock Selection

The last move is the clock selection operation. As mentioned in section 3.2, the clock has profound effects on the resulting PDF of the distribution of a functional unit delay. Thus, incorporating automatic clock selection as one perturbation move can potentially mitigate the effect of process variation while achieving good area and latency tradeoffs. Note that, we have first used the clock slack and timing event minimization algorithm described in section 3.2 to find out the range in which the length of a clock can vary. That range is determined by mapping CDFGs to the fastest performance and the smallest area resources. Since a clock cycle time is randomly generated from the confined range, we have defined the term, SEP (average slack and event count product), to guide the selection of CCT during SA process. One thing to note is that the slack defined here is the average slack based on the probability associated with a timing event and is not the slack under the worst case delay. For example, the average slack of unit A for 12 CCT, shown in Figure 5, is 2.66=(1/9*2+1/9*1+3/9*0+2/9*11+1/9*10) while the slack is 10 for the worst case delay approach. The timing event count is calculated the same as the method mentioned in Figure 5. When one CCT is chosen from the candidate clocks, we first calculate its overall $SEP = \sum_i SEP(t_i)$ on all operation type t_i. The acceptance is approved if the current SEP is smaller than the SEP induced from the previous chosen clock cycle time. This process is iterated until we find a suitable CCT.

3.5 Performance Yield Constraint

A new scheduled, resource allocated CDFG solution is acquired when one of the three perturbations is applied. This new solution will be judged to determine it is accepted or not in two aspects, cost function and the current temperature of SA. The cost function is defined based upon optimization criteria and is normally related to area and latency constraints in high-level synthesis. Thus, the following equation is adopted to optimize both area and latency of a scheduled, resource allocated CDFG solution:

$$cost = \alpha * \frac{A}{A_{norm}} + (1 - \alpha) * \frac{L}{L_{norm}} \quad (1)$$

where A is the area needed to fulfill the scheduled CDFG application, A_{norm} is the average area, L is the required latency after scheduling, L_{norm} is the average latency, and α controls the weights of area and latency, respectively.

After one iteration of SA, latency distributions are available at the outputs of functional units of a CDFG. This distribution is represented as PDF and we can readily convert it into cumulative distribution function (CDF) by summation. The CDF of the latency delay is used in confining the performance yield. Based on the required performance yield, the probability of obtaining a fabricated die that meets this constraint, the maximum expected performance can be obtained. In order to satisfy a specific yield point while maintaining a reasonable area consumption, both factors must be included in the cost function.

However, the latency used in equation 1 does not consider process variation and is a deterministic value. Since the delays of functional units are now of distribution forms, the latency obtained through the traditional HLS is not correct and is either pessimistic or optimistic. Thus, incorporating processor variations into HLS optimization is of preference and is described following. After one iteration of resource allocation, binding, and scheduling, we apply SSTA on the CDFG to obtain the delay distributions at the outputs of functional units of a CDFG. For a given performance yield point, we can easily acquire the latency that meets the yield constraint by extracting the corresponding latency in CDF. Then, this latency

Benchmark	Fastest Performance			Smallest Area			SA-based Speed			
	Area	Total time(ns)	Run time(s)	Area	Total time(ns)	Run time(s)	Area	Total time(ns)	Run time(s)	Reduc. %
EW	400	143	1	240	253	1	360	143	22	10%
FIR	650	135	1	370	252	1	630	135	12	3%
$Diff$	875	60	1	495	112	1	715	60	5	18%
AR	1100	120	1	620	224	1	980	135	42	11%
dct_dif	1200	105	1	680	196	1	970	105	5	19%
$chemical$	1525	90	1	865	168	1	1300	90	69	14%

Table 1: The results of fastest performance and smallest area resource settings, and SA-based speed optimization.

with variation is multiplied with the chosen clock and forms one factor in the modified cost function. Under such setting, we can consider area, clock, and latency with variability at the same time. The modified cost function for HLS considering performance yield can be written as:

$$cost = \alpha * \frac{A}{A_{Max}} + (1 - \alpha) * \frac{T}{T_{Max}} \quad (2)$$

where T_{Max} and A_{Max} are the upper limits on the area and the completion time, which are given as design specifications. T is the product of selected clock and latency, where the latency is extracted from the satisfied performance yield point of the delay distribution. Thus, the SA-based HLS can proceed with the inclusion of the performance yield while optimizing the solution.

3.6 Fast Simulated Annealing Scheme

Although simulated annealing is a powerful technique to tackle various optimization problems, the excessive running time is its major weakness. In order to accelerate the whole searching process, we have adopted the fast simulated annealing scheme [7] and integrated into our algorithm.

4. EXPERIMENTAL RESULTS

In this section, we present experimental results for the performance yield-guaranteed high-level synthesis algorithm described in section 3. The algorithm was implemented in C++/STL and the experiment was carried out on an Intel Xeon based Linux machine. The resource library used in the experiment contains the delay distribution with probabilities and the area requirement for each functional unit. For this work, five different types of adders and multiplier are included in the resource library. Each functional unit was simulated in SPICE with Monte Carlo method to model both intra-die and inter-die variations. Different levels of variability in delay were explored ranging from 7-9% of σ/μ [22] and the distribution was truncated at the 3σ point. However, any delay distribution can be used in our framework. We have used six high-level synthesis benchmarks to evaluate the proposed approach; namely, a 16-point symmetric *FIR* filter, a 16-point elliptic wave filter (*EW*), an autoregressive lattice filter (*AR*), an algorithm for computing Discrete Cosine Transform (*dct_dif*) [24], a differential equation solver (*Diff*), and *chemical* is an IIR filter used in the industry.

In the first experiment, we demonstrate that the SA-based HLS is effective in reducing the area requirement while achieving the same timing performance. We first map the CDFG with the smallest area and the fastest performance resource settings, and then use the scheduling algorithm described in section 3.3 to assign each node of the CDFG to the corresponding control step. Note that, the scheduling algorithm [8] can distribute nodes evenly into different time steps such that the total resource usage is minimized. This result is shown in the columns of 2 to 7 of Table 1. Next, we use the SA-based HLS algorithm with two perturbation moves described in the previous section (resource re-binding and operation shift) and equation 1 as the cost function. We refer this approach as SA-based speed optimization afterward. The result of applying the SA-based HLS for speed optimization is shown in the columns of 8 to 11. The average area reduction is around 13% when compared to the scheduling with the fast resource setting. Thus, we know that the resource re-binding and the operation shift moves can improve the solution quality from the plain scheduling algorithm. The results shown here is to demonstrate that SA-based HLS is effective at extracting good HLS solutions. Note that, the effect of process variation is not considered here.

The second experiment we conduct is to demonstrate that when the clock selection move is deployed, the area can be further reduced at different performance yield points. For this experiment, we first use the area and the total completion time from the results of SA-based speed optimization and the fastest performance resource setting depicted in Table 1 as the upper limits. This is because that these values represent the best results so far we can obtain for the completion time and the area, respectively, and thus it is suitable to use them as the terms, T_{Max} and A_{Max}, in equation 2. The SA-based HLS is then executed with all three perturbation moves and with the performance yield constraint. Due to the space limitation, we only show the results of FIR, EW, and dct_fi benchmarks, which are listed in Table 2. In the table, T and A indicate the bound on the total completion time and the area requirement, #CC is the number of clock cycles needed, clk shows the clock length chosen from the algorithm, Total time is the product of clock cycle time and #CC, and the last two columns show the run time and the area improvement, respectively.

As can be seen from the table of FIR, we can still achieve 9.2% area reduction compared to that of 3% obtained from the SA-based speed optimization approach that does not use the clock selection move while achieving the same total completion time with the required performance yield. As we relax the performance yield and/or the total completion time (T) constraints, the area reduction is more significant. The same trend can be observed from other benchmarks. One thing to note from the tables of EW and dct_dif benchmarks is that the chosen CCT varies depending upon the performance yield with the area and the total completion time constraints. This does validate that the choice of the clock cycle time has profound impacts on the solution quality.

Figure 10 shows the area improvements of all benchmarks over the fastest performance resource configurations at different performance yields. In this figure, only the most rigid completion time (T) constraint is used. The average area reduction are 14%, 19%, and 24% for 95%, 90%, and 85% performance yield points, respectively. Note that, we actually achieve the same total completion time with 95% point as that of 100% so our approach does not sacrifice the performance to gain area reduction. Instead, our algorithm uses all three moves, resource re-binding, operation shift, and clock selection, to currently explore the solution space, and thus optimized solutions can be acquired.

5. CONCLUSION

In this paper, we have presented a performance yield-guaranteed high-level synthesis algorithm to consider the influence of process variation. Our SA-based HLS tackles the already cumbersome three-dimensional (area, latency, and clock) HLS problem with the inclusion of process variation. We have demonstrated that incorporating the clock selection and different heuristic moves in SA-based HLS can meet the area and the completion time constrains, effectively. Experimental results on several HLS benchmarks show that performance yield can be maintained while the savings in area are achieved compared to the worst-case designs.

Bounds T	Bounds A	Area	#CC	clk	Total time	run time(s)	Reduc. %
95% yield							
135	630	590	9	15	135	31	9.2%
150	630	520	9	15	135	31	20.0%
160	630	495	10	15	150	31	23.8%
90% yield							
135	630	565	9	15	135	32	13.0%
150	630	465	9	15	135	48	28.4%
160	630	430	10	15	150	31	33.8%
85% yield							
135	630	505	9	15	135	35	22.3%
150	630	455	10	15	150	32	30.0%
160	630	430	10	15	150	38	33.8%

(a) FIR filter.

Bounds T	Bounds A	Area	#CC	clk	Total time	run time(s)	Reduc. %
95% yield							
143	360	360	11	13	143	165	10.0%
155	360	360	11	13	143	132	10.0%
165	360	320	12	13	156	160	20.0%
90% yield							
143	360	350	11	13	143	167	12.5%
155	360	340	11	13	143	143	15.0%
165	360	270	11	15	165	133	32.5%
85% yield							
143	360	340	11	13	143	131	15.0%
155	360	270	11	14	154	91	32.5%
165	360	270	11	14	154	132	40.0%

(b) EW filter.

Bounds T	Bounds A	Area	#CC	clk	Total time	run time(s)	Reduc. %
95% yield							
105	970	910	7	15	105	78	24.1%
120	970	870	7	17	119	61	27.5%
130	970	785	7	17	119	76	34.5%
90% yield							
105	970	885	7	15	105	57	26.2%
120	970	810	7	16	112	50	32.5%
130	970	785	7	18	126	79	34.5%
85% yield							
105	970	870	7	15	105	79	27.5%
120	970	730	7	16	112	84	39.1%
130	970	710	8	15	120	77	40.8%

(c) dct_dif filter.

Table 2: Area reductions under different time, area, and yield bounds. (a)FIR, (b)EW, (c)dct_dif.

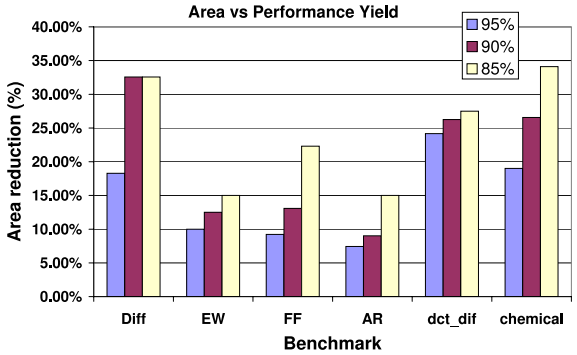

Figure 10: Area reductions under different yield points for all benchmarks.

6. REFERENCES

[1] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic. Fast Statistical Timing Analysis By Probabilistic Event Propagation. In *DAC*, 2001.

[2] A. Agarwal, D. Blaauw, V. Zolotov, S. Vrudhula. Statistical Timing Analysis using Bounds. In *DATE*, 2003.

[3] S. H. Choi, B. C. Paul, and K. Roy. Novel Sizing Algorithm for Yield Improvement under Process Variation in Nanometer Technology. In *DAC*, 2004.

[4] M. Mani, A. Devgan, and M. Orshansky. An Efficient Algorithm for Statistical Minimization of Total Power under Timing Constrains. In *DAC*, 2005.

[5] J. Ramanujam, S. Deshpande, J. Hong, and M. Kandemir. A Heuristic for Clock Selection in High-Level Synthesis. In *VLSID*, 2002.

[6] S. Naraynan and D. D. Gajski. System Clock Estimation based on Clock Slack Minimization. In *EURDAC*, 1992.

[7] T.-C. Chen and Y.-W. Chang. Modern Floorplanning Based on Fast Simulated Annealing. In *ISPD*, 2005.

[8] K. S. Hwang, A. E. Casavant, C.-T. Chang, and M. A. d'Abreu. Scheduling and Hardware Sharing in Pipelined Data Paths. In *ICCAD*, 1989.

[9] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. In *DAC*, 2003.

[10] A. Srivastava, D. Sylvester, and D. Blaauw. *Statistical Analysis and Optimization for VLSI: Timing and Power.* Springer, 2005.

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

[12] A. Raghunathan, N. K. Jha, and S. Dey. *High-Level Power Analysis and Optimization.* KAP, 1998.

[13] S. Kirpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. In *Science*, pp.671-680, 1983.

[14] V. Raghunathan, S. Ravi, and G. Lakshminarayana. High-level Synthesis using Variable-latency units. In *VLSID*, 2000.

[15] R. Mukherjee, S. O. Memik, and G. Memik. Temperature-Aware Resource Allocation and Binding in High-Level Synthesis In *DAC*, 2005.

[16] S. Tosun, et al. An ILP Formulation for Reliability-Oriented High-Level Synthesis. In *ISQED*, 2005.

[17] F. Su and K. Chakrabarty. Unified High-Level Synthesis and Module Placement for Defect-Tolerant Microfluidic Biochips. In *DAC*, 2005.

[18] A. Stammermann, et al. Binding, Allocation and Floorplanning in Low Power High-Level Synthesis. In *ICCAD*, 2003.

[19] Z. Gu, J. Wang, R. P. Dick, and H. Zhou. Incremental Exploration of the Combined Physical and Behavioral Design Space. In *DAC*, 2005.

[20] Y. Zhang, X. Hu, D. Z. Chen. Task Scheduling and Voltage Selection for Energy Minimization. In *DAC*, 2002.

[21] L. Zhang and N. K. Jha. Interconnect-aware High-level Synthesis for Low Power. In *ICCAD*, 2002.

[22] Norman J. Rohrer. Introduction to Statistical Variation and Techniques for Design Optimization. In *ISSCC Tutorial*, 2006.

[23] Dinesh Patil, et al. A New Method for Design of Robust Digital Circuits. In *ISQED*, 2005.

[24] K. R. Rao and P. Yip. *Discrete Cosine Transform.* Academic Press, 1990.