FastYield: Variation-Aware, Layout-Driven Simultaneous Binding and Module Selection for Performance Yield Optimization

Gregory Lucas, Scott Cromar, Deming Chen
Department of Electrical and Computer Engineering
University of Illinois, Urbana-Champaign
e-mail: {gmlucas2, scromar2, dchen}@illinois.edu

Abstract – While technology scaling has presented many new and exciting opportunities, new design challenges have arisen due to increased density, and delay and power variations. High-level synthesis has been touted as a solution to these problems, as it can significantly reduce the number of man hours required for a design by raising the level of abstraction. In this paper, we propose a new variation-aware high-level synthesis binding/module selection algorithm, named *FastYield*, which takes into consideration multiplexers, functional units, registers, and interconnects. Additionally, FastYield connects with the lower levels of the design hierarchy through its inclusion of a timing driven floorplanner guided by a statistical static timing analysis (SSTA) engine which is used to modify/enhance the synthesis solution. FastYield is able to incorporate spatial correlations of process variations in its optimization, which are shown to affect performance yield. On average, FastYield achieves a clock period that is 14.5% smaller, and a performance yield gain of 78.9%, when compared to a variation-unaware algorithm. By making use of accurate timing information, FastYield's rebinding improves performance yield by an average of 9.8% over the initial binding, for the same clock period. To the best of our knowledge, this is the first high-level synthesis binding/module selection algorithm that is layout-driven and variation aware.

I. Introduction

Aggressive technology scaling to the deep sub-micron realm has resulted in significant variations in fabricated device parameters. In turn, these parameter variations have caused many traditional circuit design and analysis techniques to become inadequate. To overcome this obstacle, a shift in the design paradigm from the worst-case deterministic design to a statistical or probabilistic design is critical. A new era of statistical design techniques has begun to emerge where circuit parameters such as delay and power are no longer modeled as deterministic values, but are represented as probability density functions. These statistical design techniques are leading to reclamation of lost performance and yield that has been occurring when using deterministic design techniques.

The shift to probabilistic design methodologies has produced a number of gate-level variation-aware optimization techniques [1][2]. While progress at the gate-level is encouraging, the large productivity gains available in high-level synthesis (HLS) make it attractive and necessary to address the issue of process variations at a higher level of abstraction.

In this paper, we propose a novel variation-aware simultaneous binding and module selection algorithm, named FastYield, which maximizes the performance yield of the resulting circuit. We connect our synthesis engine to the layout closely, so layout information can be accurately back-annotated to the synthesis and introduce useful synthesis transformations. Synthesis and layout are iterated until the performance gain is maximized. The major contributions of our paper are summarized below:

1) A simultaneous binding and module selection algorithm that considers registers, multiplexers, functional units, interconnects, and spatially correlated process variations.

2) A timing-driven, simulated annealing-based, statistical floorplanner that considers interconnect delay and spatial correlation between all units in the design.

3) An iterative functional unit rebinding based on timing analysis information and register criticality.

The rest of the paper is organized as follows: Section II introduces related work on recent high-level synthesis algorithms; Section III presents statistical functional unit modeling; Section IV presents the details of the FastYield algorithm; Section V presents experimental results; Section VI concludes this paper with future directions.

II. Related Work

HLS is a well-studied topic [3][4][5][6]. Much work has been done in the areas of scheduling, resource allocation, and binding. A number of works, such as [7], have addressed the topic of simultaneous binding and floorplanning, but with no consideration of spatial correlation or variability. Huang et al. [8] presented a binding algorithm based on bipartite weighted matching. However, their algorithm does not address the critical issues of module selection and delay variability. Likewise, most of the work in HLS has ignored the issue of process variation as it has not been an important issue, but that has begun to change in the past few years with the move to deep submicron processes. We will mainly introduce variation-aware HLS work here, which is an emerging area of research.

Hung et al. [9] offer a simultaneous scheduling, binding, and allocation algorithm based on simulated annealing. The simulated annealing algorithm seeks to reduce the overall latency while meeting a performance yield requirement. However, the algorithm does not consider multiplexer use or interconnect delay, both of which can significantly contribute to the clock period of the unit.

Jung et al. [10] propose a timing variation-aware HLS algorithm which improves resource sharing. While the algorithm is effective, it ignores multiplexers and interconnects, and also relies on the assumption that functional units (FUs) are independent of each other in its yield calculation given by:

$$yield = \prod_{k=1}^{n} P(FU_k < T_{clk})$$

where n is the number of functional units, and T_{clk} is the chosen clock period. As has been shown in [11] and [12], and as our results show, correlation among process parameters has an effect on the performance yield.

Lastly, Wang et al. [13] propose a simulated annealing based method to consider both power yield and timing yield during HLS. They use a number of different simulated annealing moves combined with a cost function that penalizes the design if it exceeds a power or timing yield constraint. Spatial correlation and interconnect delay are not considered.

III. Resource and Correlation Modeling

Modeling resources at a higher level of abstraction is critical to attaining an accurate HLS solution. We employ a Monte Carlo based method to *pre-characterize* the functional units. Two types of variation are considered, random variation and correlated variation (or systematic variation). The characterization flow for each unit begins with logic synthesis followed by placement and routing using Synop-

978-1-4244-2749-9/09/$25.00 ©2009 IEEE

sys Design Compiler and Cadence SOC Encounter. The characterization was performed on a recently released 45nm standard cell library provided in the design kit from [14]. From the place and route information, the delay of the unit and placement of the individual gates are extracted.

Using Monte Carlo analysis, we then characterize the units by specifying a correlated, θ_{cor}, and independent, θ_{ind}, percentage of delay variation for each gate in the resource with respect to its nominal delay value. For each Monte Carlo run, the critical path of the circuit is then found by running a deterministic timing analysis (we used Synopsys PrimeTime). By plotting the critical path for each Monte Carlo run, the mean, μ_{FU}, and standard deviation, σ_{FU} of the delay distribution is built.

To consider spatial correlation during the binding algorithm, we define two types of delay variation, inter-unit delay variation and intra-unit delay variation. Inter-unit delay variation is defined to be correlated across units, while intra-unit delay variation is defined to be independent across units. The components of inter- and intra-unit delay variation are calculated as percentages of the standard deviation that was found from the Monte Carlo analysis of the resource. Equations (1) and (2) show the calculation of the intra- and inter-unit delay standard deviations.

$$\sigma^2_{intra} = \sigma^2_{FU} \times \theta_{ind} / (\theta_{ind} + \theta_{cor}) \quad (1)$$

$$\sigma^2_{inter} = \sigma^2_{FU} \times \theta_{cor} / (\theta_{ind} + \theta_{cor}) \quad (2)$$

We support different structural implementations of the same arithmetic operation. These implementations provide different delay and area tradeoff characteristics and offer opportunities for better design space exploration targeting higher performance yield given a specific resource or area constraint.

IV. FastYield Algorithm Description

In this section we will present the FastYield binding/module selection algorithm. FastYield seeks to improve performance yield through a multiplexer- and interconnect-aware delay reduction strategy. Performance yield evaluated at a clock period *t*, *PY(t)*, is defined as:

$$PY(t) = P(r_1 \le t, r_2 \le t, ..., r_n \le t) \quad (3)$$

where *PY(t)* is the probability that $r_1, r_2, \ldots, r_n$ meet the clock period requirement, and r_n represents the probability distribution of register *n*. We assume all delays are jointly Gaussian with an associated covariance matrix, i.e. they are correlated.

The algorithm has three major components: 1) an initial resource allocation and binding; 2) a timing driven floorplanner, which performs both a timing driven placement as well as a statistical static timing analysis (SSTA); and 3) a FU rebinding which incorporates timing analysis information from component 2. FastYield seeks to improve the synthesis solution through iteratively feeding back accurate, floorplan and interconnect-aware, statistical timing information to the rebinding step.

One of the strengths of FastYield lies in its use of a process correlation model during timing analysis. Enabled by the floorplan, interconnect delay and multiplexer delays are considered during each SSTA step. Performance yield is calculated at the end of each timing analysis to evaluate the success of the algorithm, and the algorithm exits when no further improvement is seen in the binding/module selection solution. Each of the main components of FastYield is described next.

A. Initial Binding

The inputs to the algorithm include: 1) a scheduled control data flow graph (CDFG), 2) a resource library, and 3) an area constraint. The resource library contains all the resources – including FUs, multiplexers, and registers – as well as the pre-characterization data for each. FastYield performs an initial allocation and binding in three steps: First, a minimal set of registers is allocated and bound to a set of variables (variables are outputs of operations). Second, a combined FU allocation and binding takes place. Third, a minimized set of multiplexers is allocated. We name this section Initial Binding to differentiate from the Rebinding procedure to be covered later.

A.1. Register Allocation and Binding

Register binding is accomplished in a manner similar to that described in [8], where a minimal set of registers is allocated, and variables are bound by solving a weighted bipartite graph.

A.2. Initial Functional Unit Allocation and Binding

Once the registers are allocated and variables are bound to them, FUs are allocated and operations assigned to them one control step at a time. First, control steps are ranked according to the equation:

$$Rank_{cstep} = diversity \times numOPs \quad (4)$$

where *diversity* is the number of different types of operations in the control step, and *numOPs* is the number of operations assigned to the control step. The control steps are then processed from the highest ranked to the lowest ranked. This strategy is similar to the 'first fit decreasing' heuristic used in bin packing problems. The items are put in descending order according to their volumes (in this case rank), and then packed one at a time in an effort to make the packing as close to optimal as possible.

The cluster of control step operations to be bound is placed into a set, O_{cstep}, and the available FUs are put into a set FU_{av}. On the first control step to be bound, the set of available FUs consists of, for each operation in the control step, one instance of each FU in the resource library that is compatible with that operation (see Fig. 1). This initial allocation ensures that each operation can bind to any of the compatible FUs in the resource library. In subsequent control steps, FU_{av} is trimmed of any FUs that, if allocated, would exceed the area constraint, with the qualification that a sufficient number of FUs of each type has been allocated to accommodate a successful binding solution. In this way FastYield produces a binding solution that meets the area constraint, while also enabling module selection.

A weighted bipartite graph is constructed where each vertex represents either an operation ($o_i \in O_{cstep}$) or a FU ($fu_j \in FU_{av}$), and there is an edge, e_{ij}, between each operation, o_i, and FU, fu_j, which can perform the operation, with a corresponding weight. Edge weights are based on multiplexer creation due to the already bound registers. If two operations share the same input register, it is advantageous to bind the two operations to the same FU, because no multiplexer is needed (which will in effect potentially reduce the path delay). Likewise, if two operations that share the same output register are bound to the same FU, no multiplexer is needed at the registers input port (again having a positive effect on the delay reduction). The initial binding weight, $w_{ij\_initial}$, corresponding to each edge, e_{ij}, is calculated below:

$$w_{ij\_initial} = 1 / estDelay(i, j) \quad (5)$$

$$estDelay(i, j) = \mu_{fu_j} + \mu_{mux_{in}} + \mu_{mux_{out}} + 3 \times \sqrt{\sigma^2_{fu_j} + \sigma^2_{mux_{in}} + \sigma^2_{mux_{out}}} \quad (6)$$

where, μ is the mean, σ is the standard deviation, mux_{in} is the multip-

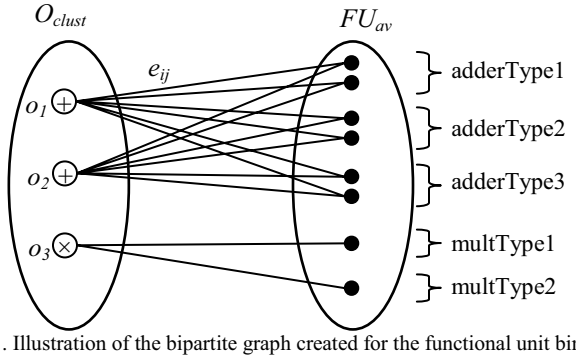

Fig. 1. Illustration of the bipartite graph created for the functional unit binding of the first control step.

lexer that would be created at the input of the FU if the operation were bound to it, and mux_{out} is the multiplexer that would be created at the input of the output register if the operation were bound to the FU. This weight calculation effectively incorporates the statistical behaviors of all the involved components in the circuit paths, putting a higher weight on the shorter delay paths. The maximum weight solution is then found to minimize the delay, and the operations are bound to FUs for the control step. After all the control steps are processed, FUs and registers are connected with allocated multiplexers.

B. Statistical Timing Driven Floorplanner

The timing-driven floorplanner is run after each binding iteration is completed to evaluate the performance yield of the solution. As has been shown in previous work, [12][15], and as we show in the experimental results section, spatial correlation of variation in parameters such as gate length can have an impact on the variance of the timing of a circuit. To achieve accurate timing results it is important that spatial correlation among units is considered during statistical timing analysis.

B.1. Unit Correlation Model

To complement the high level synthesis resource modeling, we propose a novel unit-based correlation model. In this model, each functional unit, register, or multiplexer is assigned a unit number and the correlation between each unit is found based on the distance between the center points of the units using a correlation function that meets the requirements of [12] so that the correlation matrix for the circuit is positive-semi definite, a requirement for the SSTA approach that we use.

This model is beneficial to high level synthesis as it complements the proposed resource modeling (Section III), and also takes into account the different sizes of functional units. On the other hand, a grid based model, as used in [15], does not complement the unit characterization since it is possible for functional units to be split across different grid regions, which complicates both the unit characterization process and the correlation calculation.

Fig. 2 shows an example of the unit correlation model. Two multipliers (1 and 2), an adder (3), and a register (4) are labeled in the picture. It can be seen that when an adder and a register (small area) are placed next to each other, the correlation is higher than when two multipliers (large area) are placed next to each other. This scenario can be accurately modeled using our unit-based model. Our model can also be viewed as an extension of the grid based model where each logic gate/functional unit is its own grid.

The proposed correlation model, in conjunction with the inter-unit and intra-unit variation found during the resource characterization, allows correlated variation to be represented at a higher level of abstraction with accuracy and runtime efficiency.

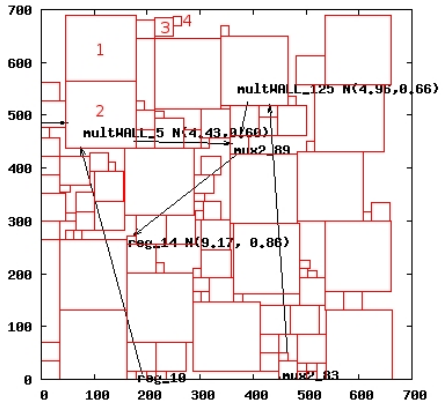

Fig. 2. Sample floorplan showing data connections.

B.2. SSTA Algorithm

To obtain layout information during the timing analysis we use a modified version of the Parquet floorplanner [16]. The modified flooplanner employs a simulated annealing approach where, after a number of unit moves, a statistical timing analysis is performed to evaluate the solution. Fig. 3 shows the pseudo code for the timing driven floorplanner.

The method for statistical timing analysis considering spatial correlation is based on the work of Chang et al. [15]. This work relies on principal component analysis (PCA) to transform a set of correlated random variables into a new set of independent random variables.

To perform PCA, a correlation matrix for the binding solution is found using the unit correlation model described above. The interconnect delay between the units is modeled based on distance. Since no detailed routing information is available, we model the connection between two functional units as a two-pin net with the length being the Manhattan distance between the two connecting terminals of the FUs. The mean Elmore delay with optimal buffer placement is then found using (7) which follows from the results of [17]**:**

$$\mu_{wire} = 2.5 \times \sqrt{R_{buff} C_{buff} R_{length} C_{length} l^2} \tag{7}$$

$$\sigma_{wire} = \alpha \times \mu_{wire} \tag{8}$$

where μ_{wire} is the mean wire delay, R_{buff} is the output resistance of the buffer, C_{buff} is the input capacitance of the buffer, R_{length} is the resistance per unit length, C_{length} is the capacitance per unit length, and l is the net length. The standard deviation of the wire length is calculated using (8) where α is a percentage of wire variation. α is found in accordance with the results from [18] as follows[1]:

$$\alpha = 0.3836 \times \exp(-0.1537 h) \tag{9}$$

where h is the optimal buffer size as calculated by [17]. We consider the wire variation to be independent across wires.

B.3. Floorplanner Cost Function

The cost function for simulated annealing moves in the floorplanner is given by:

$$Z \sim N(\mu_z, \sigma_z) = \max(reg_1(\mu_1,\sigma_1), reg_2(\mu_2,\sigma_2), ..., reg_n(\mu_n,\sigma_n))$$

$$T_R = \frac{\mu_z + \sigma_z}{\mu_{best} + \sigma_{best}}$$

$$Cost = \alpha * area + \beta * T_R \tag{10}$$

where $\max(reg_1(\mu_1,\sigma_1), reg_2(\mu_2,\sigma_2),..., reg_n(\mu_n,\sigma_n))$ represents the statistical max operation [15] on the timing distributions at the inputs to all output registers (pseudo primary outputs), μ_{best} and σ_{best} represent the mean and standard deviation of the best solution found so far, and α and β are weighting parameters. The T_R cost is then found by adding the mean and standard deviation of the max distribution, normalized by the mean and standard deviation of the best solution. In the calculation of T_R, we chose to use the sum of the mean and standard deviation since the result corresponds to the required clock frequency for an ~84% yield, for which we target in this study. After a

```
Parquet:
While (time > time_cool){
      Perform_moves(num_moves);
      Calc_wire_delay();
      Calc_Correlation();
      Perform_PCA();  //principle component analysis
      Perform_timing_analysis();
      Calculate_Cost();
}
```

Fig. 3. Timing driven floorplanner pseudo code.

[1] We plotted the equation based on the buffer size vs. wire variation data reported in [18].

specified number of moves, a timing analysis is performed on all paths in the design as described in Fig. 3.

Upon completion of the timing analysis, the delay probability density function (pdf) for each register is known. The distributions, as well as the required clock frequency for an 85% performance yield are then passed back to FastYield for the criticality analysis of the rebinding step. Fig. 2 shows the example floorplan obtained from the timing driven floorplanner, with the arrows representing the flow of data through the critical path.

C. Rebinding

Functional unit rebinding is performed after the initial solution has been analyzed by the timing driven floorplanner, and then continues in an iterative fashion until the floorplanner reports that no improvement has been made. The rebinding algorithm works by determining which functional units along the critical paths are causing the majority of the delay. It then attempts to reduce the delay in two ways: one, by swapping slower FUs on critical paths for faster FUs; and two, by rebinding individual operations on the critical paths. Fig. 4 shows the pseudo code for the rebinding algorithm, which will be explained next.

C.1. Register and Functional Unit Ranking

The algorithm begins by ranking the output registers in order of their criticality. The slowest register is identified by finding the worst case delay based on the mean and standard deviation from the floorplanning information. The rank of register r is then calculated:

$$RegRank_r = \frac{\mu_r + 3\sigma_r}{\mu_{slowest} + 3\sigma_{slowest}} \quad (11)$$

where μ_r and σ_r are the mean and standard deviation for register r, and $\mu_{slowest}$ and $\sigma_{slowest}$ are the mean and standard deviation of the slowest register. The registers are then ordered according to their criticality, or rank, starting from the most critical.

With the registers ranked, the algorithm then proceeds to rank each FU that is connected to each register. The rank of FU k connected to register r is found by:

$$FURank_k = RegRank_r \times (0.5 \times \frac{\mu_k}{\mu_r} + 0.5 \times \frac{\sigma_k}{\sigma_r}) \quad (12)$$

where μ_r and σ_r are the mean and standard deviation for output register r, μ_k and σ_k are the mean and standard deviation of the FU, and $RegRank_r$ is the rank for register r. The $RegRank_r$ weight provides an estimate of the global impact the register has on the overall clock period, while the ratio of the means and standard deviations considers how much the overall mean or variance of the functional unit impacts the final timing at the register. The end of section C.3 will present an example of how this ranking is accomplished.

C.2. Swapping Critical Functional Units

The rebinding algorithm examines the set of allocated FUs, and based on their rank, finds any higher ranked FUs that are slower than lower ranked, faster FUs of the same type, and swaps them. The net effect is to place the fastest FUs on the most critical paths. If no FUs meet the criteria for swapping, the rebinding proceeds to the next step. If FUs are swapped then the timing analysis is re-run before rebinding proceeds.

```
Rebind {
  Calc_reg_and_FU_ranks();
  If(Swap_critial_FUs()) Break;
  Order_rebind_operations(Orebind);
  for_each (op in Orebind) {
    Calc_op_to_FU_weights();
    Bind_largest_weight_pair();
    Estimate_timing();
    Recalc_reg_and_FU_ranks();
  }
}
```

Fig. 4. Rebinding algorithm pseudo code.

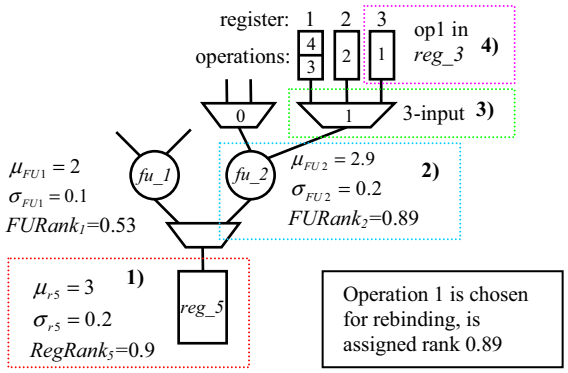

Fig. 5. Example of FU ranking and the selection of operations for rebinding

C.3. Selection of Operations to be Rebound

The rankings of the registers and FUs are used in the selection of particular operations that will be rebound. Operations are chosen that both contribute to a critical path delay, and have the potential to reduce that delay. Briefly, this is done as follows: First, a set of output registers are selected for their delay criticality based on their rank. For each chosen register, the FU connected to it with the highest rank (denoting its greater contribution to the criticality of the register) is selected, and an operation, or multiple operations, that are bound to that FU are selected to be rebound. The criterion for selection of the particular operations associated with each FU to be rebound is the operation's potential, if rebound, to reduce multiplexer size on that critical path. The example in the next paragraph serves to clarify the process.

An example of the register and FU ranking, and operation selection, is illustrated in Fig. 5. The method is presented step-by-step: The slowest register has a mean μ = 3.1, and a standard deviation σ = 0.3. 1) Based on the slowest register information, by (11) register 5 is found to have a rank of 0.9. (Register 5 is determined to be critical based on its rank.) 2) The FUs connected to register 5 are ranked according to (12). *fu_2* is found to have a higher rank than *fu_1*, so it is from *fu_2* that an operation, or operations, will be selected for rebinding. 3) The inputs to *fu_2* are examined, and port 1 is found to have a larger multiplexer than port 0. 4) The registers connected to the inputs of the 3-input multiplexer are evaluated. One of the three registers has two variables bound to it, and the other two have one variable bound to them. Since fewer variables bound to a register is preferred (more likely to reduce the multiplexer size if moved), register 3 is randomly chosen from the two registers with only one variable bound to them. The operation corresponding to that register/variable, operation 1 in this case, is assigned the rank of the target FU, and chosen for rebinding. This same process is carried out for each critical register, and the selected operations (along with their ranking) are placed in the set O_{rebind} to be rebound.

C.4. Operation Rebinding

The rebinding is performed for each operation $o_i \in O_{rebind}$, one operation at a time, starting with the operation with the highest rank. Previous bindings that have not been selected for rebinding are left untouched. For a given operation, o_i, a rebind weight is calculated for each FU, fu_j, in the previously allocated FU set. The weight, w_{ij_rebind}, for each operation FU pair is calculated as follows:

$$w_{ij\_rebind} = \frac{w_{ij\_rebindPrevious}}{\max(w_{i\_rebind})} \times (1 - FURank_j) \quad (13)$$

where $w_{ij\_rebindPrevious}$ is the weight of the operation-to-FU pair in the previous iteration of rebinding (or the initial binding if this is the first iteration), $\max(w_{i\_rebind})$ is the maximum weight from all of the operation-to-FU pairs, and $FURank_j$ is the FU rank as described

earlier. The first part of the weighting considers the likelihood operation o_i had of being assigned to fu_j in the previous binding. If o_i was close to being assigned to fu_j during the previous binding, then rebinding o_i to fu_j will be a good choice, if the rank of the FU is low (meaning it currently is not a part of the critical path). The second part of the equation adds this rank consideration to the weight.

The operation-to-FU pair with the largest weight is then chosen, and that operation is bound to the FU. The process repeats for each operation that belongs to the set O_{rebind}. However, after each operation is rebound it is possible that the multiplexer size has changed, which in turn reduces the critical path of the circuit and changes the ranks of the registers. Therefore, after each operation is rebound, a fast estimated timing analysis is performed on the paths that are affected by the rebinding of the operation and the register and FU ranks are recalculated. After every operation in O_{rebind} has been processed, one iteration of rebinding is complete and the solution is sent to the floorplanner for analysis.

V. Experimental Results

In this section we present a number of results that demonstrate the importance of considering process variation and correlation during high-level synthesis, and the effectiveness of FastYield at accomplishing these tasks. FastYield reads in a benchmark, which has been pre-scheduled with list scheduling, and a resource library, and runs it through the initial binding, floorplanning and timing analysis, and rebinding. The resource library contains the pre-characterized resources, which include FUs, multiplexers, and registers. The resources were pre-characterized with 10% random variation and 10% spatially correlated variation with a correlation distance of 1 mm (such assumptions are compatible with the variation predictions laid out in [19]). The characterization was performed on a 45nm library provided in the design kit from [14].

A number of data-intensive benchmarks are used in our experiments with FastYield. The benchmark control data flow graphs include several different DCT algorithms such as *pr*, *wang*, and *dir*, and a couple of DSP programs such as *chem*, *mcm* and *honda* [20]. The benchmarks are profiled in Table 1. Each node in the benchmarks is either an addition/subtraction or a multiplication.

A. Spatial Correlation in Timing Analysis

In order to show the importance of considering spatially correlated process parameters during the timing analysis we performed a floorplanning and timing analysis on the same binding solution with spatial correlation and θ_{ind} from equation (1) set to 0 (Corr), and without correlation (No Corr) with $\theta_{ind} = 1$. Setting $\theta_{ind} = 0$ makes $\sigma_{inter}^2 = \sigma_{FU}^2 = 1$. This makes it possible for all the FU variation to be correlated between units, however, the actual correlation between the FUs is still found based on the distance between them. The results are shown in Table 2. Columns 2 and 3 show the clock period obtained for an 85% yield with Corr and No Corr, respectively. Column 4 shows the reduction in clock period of the Corr result over the No Corr result, which averages 4.22%. Column 5 reports the performance yield (PY) gain of Corr over No Corr, which averages about 14.25%. That is, for the No Corr clock period given, one would expect to achieve an 85% PY based on the No Corr timing analysis, but would achieve a 85% + 14.25% = 99.25% PY based on the Corr timing analysis. In other words, timing analysis without consideration of correlated process parameters is conservative compared to correlated timing analysis. This shows the importance of using spatial correlation information to guide the floorplanner, as well as performing accurate SSTA.

TABLE 1
Benchmark Profiles

Benchmark	No. of PIs	No. of POs	No. of Adds	No. of Mults	Total No. of Edges
chem	20	10	171	176	731
dir	8	8	84	64	314
honda	9	2	45	52	214
mcm	8	8	64	30	252
pr	8	8	26	16	134
steam	5	5	105	115	472
wang	8	8	26	22	134

B. FastYield Compared to BindBWM and Rebinding Improvement

We compare the results of FastYield after rebinding (FY Rebind) to an enhanced version of the weighted bipartite graph based binding (here referred to as BindBWM) of Huang, et al. [8]. The enhancements to [8] include module selection and the ability to specify an area constraint, making the comparison demonstrative of the performance yield gains that can be achieved when considering process variation during binding. The same schedules, area constraints, and library were used in both algorithms. We also compare FY Rebind performance to the performance attained by FastYield before rebinding (FY Initial) to show the effect of timing information on the rebinding solution. In all of the benchmarks, the same number of adders and multipliers were allocated in the binding solution for BindBWM, FY Initial, and FY Rebind.

Table 3 summarizes the experimental results. Columns 2, 4, and 6 give the clock periods for each BindBWM, FY Initial and FY Rebind respectively. Columns 3 and 5 give the PY attainable by the respective binding solutions if clocked at the 85% PY clock of FY Rebind. Fig. 6 demonstrates this graphically by plotting the cumulative density functions (cdf's) for the different binding results of *chem* (pdf's are inset). If clocked at the 85% PY clock period of FY Rebind, FY Initial and BindBWM have PY's of 67.7% and 12.5%, respectively.

In Table 3, Column 7 gives the total FY runtime in minutes. Columns 8 and 10 give the FY Rebind percentage reduction in clock period when compared to BindBWM and FY Initial, respectively. Columns 9 and 11 give the PY gain (in percent) of FY Rebind over BindBWM and FY Initial, respectively. This means that if BindBWM

TABLE 2
Correlation vs. No-Correlation Experimental Results

Benchmark	85% Yield Clk (ns)		Corr reduction in Clk over No Corr (%)	Corr 85% PY Gain over No Corr (%)
	Corr	No Corr		
chem	5.91	6.20	4.70	14.97
dir	4.91	5.14	4.49	14.98
honda	5.14	5.30	3.03	14.37
mcm	4.09	4.28	4.35	10.56
pr	4.45	4.66	4.51	14.99
steam	5.54	5.80	4.51	14.89
wang	4.91	5.11	3.98	14.99
Average			**4.22**	**14.25**

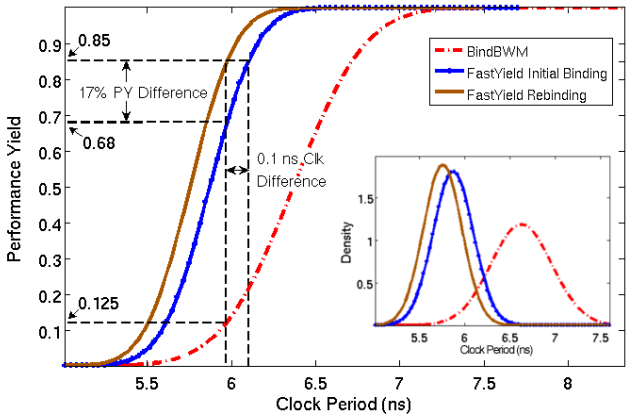

Fig. 6. *Chem* delay distributions of BindBWM, FY Initial, and FY Rebind.

TABLE 3
FastYield Experimental Results

	BindBWM		FastYield Initial		FastYield Rebind		Comparison			
Benchmark	85% Yield Clk (ns)	PY at FY Rebind 85% Clk (%)	85% Yield Clk (ns)	PY at FY Rebind 85% Clk (%)	85% Yield Clk (ns)	Total FY Run Time (min)	FY Rebind reduction in Clk over BindBWM (%)	FY Rebind 85% PY Gain over BindBWM (%)	FY Rebind reduction in Clk over FY Initial (%)	FY Rebind 85% PY Gain over FY Initial (%)
chem	6.9	12.5	6.1	67.7	6.0	75	14.17	72.5	2.35	17.3
dir	5.8	1.5	4.9	70.9	4.8	43	16.71	83.5	1.76	14.1
honda	5.7	8.1	4.9	82.6	4.9	28	14.39	76.9	0.32	2.4
mcm	4.9	11.4	4.3	78.0	4.2	40	14.57	73.6	3.34	7.0
pr	5.2	0.1	4.5	70.1	4.3	24	16.47	84.9	3.04	14.9
steam	6.2	7.6	5.5	76.3	5.5	64	11.88	77.4	1.14	8.7
wang	5.3	1.6	4.7	80.8	4.6	16	13.29	83.4	0.95	4.2
Average							**14.50**	**78.9**	**1.84**	**9.8**

or FY Initial were clocked at the 85% PY clock period of FY Rebind, they would have a PY smaller than 85% by the given amount.

By considering process variation and layout, FY Rebind is able to reduce the clock period of the benchmarks by an average of 14.5% and increase the performance yield an average of 78.9%, when compared to BindBWM. It is also able to improve clock period and PY an average of 1.84% and 9.8%, respectively, over FY Initial.

In some cases the amount of clock period improvement that rebinding can achieve is limited by the number of the type of unit that is on the critical path. For example, if there are 4 allocated multipliers, all of which are found to be critical, then rebinding cannot offer much improvement. However, if only 3 of 4 allocated multipliers are found to be critical then rebinding can offer more improvement.

Often, though, even if the reassignment of operations has a small effect on mean clock period, it can have a large impact on the variance of the clock period, thus improving the PY significantly. This can be seen in Fig. 6, where there is a large improvement in the delay cdf between the BindBWM and FY Rebind. This explains the results in column 3 of Table 3, where we see that when BindBWM is clocked at the 85% PY clock value of FY Rebind, the PY is very small. The difference between FY Rebind and FY Initial is not as drastic, but there are two key improvements. First, the mean of the pdf has been shifted to a lower clock value. Second, the variance has been reduced. Combining these two improvements results in a significant PY jump for a relatively minor change in the mean clock period (17% PY difference in the example).

VI. Conclusions and Future Work

We have presented a new variation-aware algorithm, FastYield, for simultaneous binding and module selection. FastYield incorporates many competing factors into its algorithms that are not found in previous variation-aware algorithms. It considers register, multiplexer, and functional unit usage as well as spatial correlation among the resources during SSTA embedded in a floorplanner. The importance of spatial correlation during SSTA was demonstrated. On average, FastYield achieves an 85% performance yield clock period that is 14.5% smaller, and a performance yield gain of 78.9%, when compared to a variation-unaware and layout-unaware algorithm based on [8]. Also, by making use of accurate timing information, FastYield's rebinding improves performance yield by an average of 9.8% over the initial binding, for the same clock period. This result shows that by performing statistical layout-driven synthesis, substantial gains in performance yield can be made. Future work includes making scheduling variation-aware as well. Simultaneous register and functional unit binding considering process variation will also be considered.

VII. Acknowledgement

This work is partially supported by NSF grant CCF 07-02501.

References

[1] M. Guthaus, N. Venkateswaran, C. Visweswariah, V. Zolotov, "Gate Sizing Using Incremental Parameterized Statistical Timing Analysis," Intl Conference on CAD, 2005.

[2] I. Lin, T. Ling, Y. Chang, "Statistical Circuit Optimization Considering Device and Interconnect Process Variations," Intl Workshop on System Level Interconnect Prediction, 2007.

[3] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[4] D. Gajski, N. Dutt, A. Wu, *High-level synthesis: Introduction to chip and system design*, Kluwer Academic Publishers, 1992.

[5] A. Raghunathan, N. K. Jha, S. Dey, *High-level power analysis and optimization*, Kluwer Academic Publishers, 1998.

[6] R. Camposano, W. Wolf, *High-level VLSI synthesis*, Springer-Verlag New York, 2001.

[7] Y. M. Fang, D. F. Wong, "Simultaneous Functional-Unit Binding and Floorplanning," Intl Conference on CAD, 1994.

[8] C. Huang, Y. Chen, T. Lin, Y. Hsu, "Data Path Allocation Based on Bipartite Weighted Matching," DAC, 1990.

[9] W. L. Hung, X. Wu, Y. Xie, "Guaranteeing Performance Yield in High-Level Synthesis," Intl Conference on CAD, 2006.

[10] J. Jung, T. Kim, "Timing Variation-Aware High Level Synthesis," Intl Conference on CAD, 2007.

[11] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, C. Spanos, "Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization," ISQED, 2005.

[12] J. Xiong, V. Zolotov, L. He, "Robust Extraction of Spatial Correlation," Intl Symposium on Physical Design, 2006.

[13] F. Wang, G. Sun, Y. Xie, "A Variation Aware High Level Synthesis Framework," DATE, 2008.

[14] OSU Design Flows for MOSIS SCMOS_SUBM Design Flow FreePDK 45nm Variation-Aware Design Flow, http://vcag.ecen.okstate.edu/projects/scells/

[15] S. Chang, S. Sapatnekar, "Statistical Timing Analysis Considering Spatial Correlations using a Single PERT-like Traversal," Intl Conference on CAD, 2003.

[16] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design," *IEEE Trans. on VLSI Systems*, vol. 11(6), pp. 1120-1135, December 2003.

[17] H.B. Bakoglu, J.D. Meindl, "Optimal Interconnection Circuits for VLSI," *IEEE Trans. on Electron Devices*, May 1985.

[18] N. NS, T. Bonifield, A. Singh, C. Bittlestone, U. Narasimha, "BEOL Variabiliy and Impact on RC Extraction," DAC, 2005.

[19] S. Nassif, "Design for variability in DSM technologies," ISQED, 2000.

[20] M. B. Srivastava, M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput," *Trans. on VLSI Systems*, 1995.