Unified Adaptivity Optimization of Clock and Logic Signals

Shiyan Hu, Jiang Hu

Department of Electrical and Computer Engineering
Texas A&M University
College Station, Texas 77843
Email: {hushiyan, jianghu}@ece.tamu.edu

Abstract— **VLSI design is increasingly sensitive to variations which often degrade the parametric yield. Post-silicon tuning techniques can compensate for specific variations on the die and thus significantly improve the yield. Previous works on adaptivity optimization for post-silicon tuning focus on either logic signal tuning or clock signal tuning. This paper proposes the first unified adaptivity optimization on logical and clock signal tuning, which enables us to significantly save resource. In addition, it does not need any assumption on variation distributions.**

Our unified optimization is based on a novel linear programming formulation which can be efficiently solved by an advanced robust linear programming technique. Due to the discrete nature of the problem, the continuous solution obtained from linear programming is then efficiently discretized. This procedure involves binary search accelerated dynamic programming, batch based optimization, and Latin Hypercube sampling based fast simulation. Our experimental results demonstrate that up to 50% area cost reduction can be obtained by the unified optimization compared to optimization on logic or clock alone. In addition, the proposed discretization approach significantly outperforms the alternatives in terms of solution quality and runtime.

Keywords: Variation, Post-Silicon Tuning, Logic Signal Tuning, Clock Signal Tuning, Robustness

I. Introduction

With continuously shrinking features on the die, VLSI design is increasingly sensitive to variations such as manufacturing process variations. Consequently, circuit performance is no longer determined solely by deterministic values. It has significant uncertainty which needs to be considered in order to achieve high yield. There exist a handful set of approaches (e.g., [1], [2], [3]) which focus on performing statistical circuit optimization in the pre-silicon phase. That is, circuit parameters are determined in design time for yield optimization. With statistical variation models, they obtain the statistically optimized design and apply the design to all the dies. Although the optimized design is of good quality in statistical sense, the design is not necessarily ideal for each individual fabricated chip. Specific circuit parameter variations on the die cannot be mitigated. In addition, reliable statistical variation models are not easy to obtain [4].

In contrast to pre-silicon statistical optimizations, post-silicon tuning methodology can tune some circuit parameters after the chip is fabricated. This enables us to mitigate the specific circuit parameter variations on the individual chip to satisfy the design target. As a result, the timing yield can be significantly improved [5], [4].

Clearly, it is highly desirable to perform circuit adaptivity optimization for post-silicon tuning. Since making a circuit element post-silicon tunable necessarily introduces overhead, adaptivity optimization for post-silicon tuning aims to provide large tunability with small overhead. Previous works focus on either logic signal tuning (e.g., [5], [6], [4]) or clock signal tuning (e.g., [7], [8]). Note that some approaches (e.g., [6], [8]) also consider to perform gate sizing in design time, however, no joint tuning on logic and clock signal is performed in post-silicon phase. These approaches are effective, however, the resource utilization is not necessarily efficient since the interaction between logic circuit and clock network is not explored. Performing unified adaptivity optimization on clock and logic signals has the potential to significantly reduce overhead while still having large tunability for achieving yield target.

Common post-silicon tuning techniques include adaptive body biasing (ABB) [5] and tunable clock buffer (PST buffer) [7]. In this paper, as an illustration of our methodology, ABB is used to tune logic signals as in [5], [6], [4] and PST buffer is used to tune clock signals as in [7], [8]. Our approach can be easily extended to handle other tuning techniques.

In this paper, a unified adaptivity optimization technique on clock and logic signals is proposed. The new technique determines the location and the tuning range of each tunable circuit element. We propose methods for both continuous optimization and discrete optimization. Continuous approach assumes that each circuit element can be tuned to arbitrary precision. It involves a linear programming with uncertainty problem, which is solved by a robust linear programming approach with a parameter easily controlling the tradeoff between the worst-case design and the nominal design. Discrete approach discretizes the continuous solution, i.e., maps the tuning range of each tunable element to a permissible set of tuning ranges. It involves binary search accelerated dynamic programming, batch-based optimization, and Latin Hypercube sampling based fast simulations. Our main contribution is summarized as follows.

- According to the best of the authors' knowledge, this is the first work on unified adaptivity optimization for post-

1-4244-1382-6/07/$25.00 ©2007 IEEE

silicon tuning on clock and logic signals.

- In contrast to most previous works (e.g., [6], [4]), our approach computes the discrete solution in addition to the continuous solution. This is desirable due to the discrete nature of problem.
- Unlike many previous works (e.g., [6]), our new approach does not assume any variation distributions since reliable variation model is not easy to obtain in reality.

Computation cost is also an important issue. In our unified adaptivity optimization approach, the problem size is almost the same as adaptivity optimization on clock signal alone. This is due to that logic tuning is applied to the circuit block level but not to individual gate (compared to e.g., [8]). We have up to several dozens of circuit blocks. In addition, our continuous approach does not perform any Monte Carlo simulation and thus runs very efficiently. Even for the discretization approach which involves Monte Carlo simulations, since the search space is largely reduced due to being guided by continuous solution, together with various acceleration techniques, it still runs efficiently. Although the works of [7], [8] are also independent of assumptions on variation distributions, they tend to be slow as full-fledge Monte Carlo simulations are frequently called during their optimization procedures.

Our experiments demonstrate that the new continuous unified adaptivity optimization approach is consistently better than adaptivity optimization on logic or clock signal alone. One can achieve up to 50% area cost reduction by our approach. Our discretization approach also significantly outperforms the alternatives including nearest rounding approach and binary batch approach [7] in terms of yield, area and runtime.

II. Preliminaries and Motivation

In our adaptivity optimization approach, adaptive body biasing is applied to tune logic signals and PST buffer tuning is applied to tune clock signals. Note that as indicated in [4], due to well-spacing related layout rules and overhead issue, we apply ABB at circuit block level but not to individual device.

Since making a circuit element (e.g., a gate or a clock buffer) post-silicon tunable necessarily introduces overhead, adaptivity optimization for post-silicon tuning aims to provide large tunability with small overhead. Previous works focus on logic signal (e.g., [5], [6], [4]) or clock signal (e.g., [7], [8]) separately. Some approaches (e.g., [6], [8]) also consider to perform gate sizing in design time, however, no joint tuning on logic and clock signal is performed in post-silicon phase.

These approaches are effective, however, their resource utilization is not necessarily efficient since the interaction between logic circuit and clock network is not explored. Consider that we have a circuit which can achieve 99% yield by logic signal tuning alone. The observation is that some slack can be moved from non-timing critical part to timing critical part by introducing useful skew using clock signal tuning, and clock signal tuning is often cheaper than ABB (due to e.g., ABB is applied on a block of gates, a lot of more control signals need to be used, and ABB needs Digital-to-Analog converters [5]). As a result, usage of clock signal tuning would lead to less logic tuning and thus significantly reduce the overhead. On the other hand, solely relying on clock signal tuning often leads to small improvement on yield. This is the case since there may be many critical cycles in a circuit, and when these cycles are heavily overlapped, one can only move small amount of slack among flip-flops. In addition, a single clock buffer may drive many flip-flops and tuning it affects all of them. If some flip-flops are in timing critical paths, then only small amount of tuning can be applied to this clock buffer.

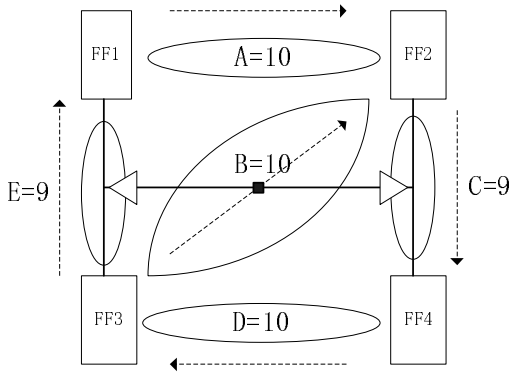

Fig. 1. A sequential circuit where the arrows show the signal flow directions. The central square is the clock source and the triangles are clock buffers.

The motivation of the unified optimization is further illustrated by the example of Figure 1. Figure 1 shows a sequential circuit consisting of 4 flip-flops $FF1, FF2, FF3, FF4$, two clock buffers, and five combinational paths A, B, C, D, E. The nominal delay for each combinational path is shown and all clock skews are zero. Suppose that the clock period is 10 and each combinational path has 10% variation off the nominal delay value. For simplicity, assume that we need to guarantee the worst-case design satisfying the timing constraint. Since three combinational paths A, B, D may have timing violations, we need to tune them by ABB. If clock tuning is also used, we can tune the right clock buffer to introduce 1 unit delay, i.e., $FF2$ and $FF4$ both have skew of 1. One can see that A, B do not need ABB due to the additional slack. That is, tuning the right clock buffer and D is sufficient, which certainly saves overhead. It is also clear that tuning clock signal alone may not make both A and D satisfy the timing constraint.

III. Overall Flow

A two-stage optimization approach is proposed to decide the location and the tuning range of each tunable element in the unified adaptivity optimization. The first stage is called *Continuous Optimization* which is to efficiently compute a post-silicon tunable design with the assumption that one can achieve arbitrary tuning precision for any tunable element. The second stage is called *Discretization*, which is to map the obtained continuous tuning range for each tunable element

$$\min \quad c_{b,12} + n_{fbb,12} - n_{rbb,12} \tag{1}$$

$$s.t. \quad (S_1 + d_1) + (T_{12} - n_{12}D_{p_{12}} - n'_{12}D_{p_{12}}) - (S_2 + d_2) \leq T_{cp} - T_{setup}, \tag{2}$$

$$(S_1 + d_1) + (t_{12} - n_{12}D_{p_{12}} - n'_{12}D_{p_{12}}) - (S_2 + d_2) \geq T_{hold}, \tag{3}$$

$$c_{b,12} = m_{b_1}C_{b_1} + c_{b_1} + m_{b_2}C_{b_2} + c_{b_2} + m_{b_3}C_{b_3} + c_{b_3}, \tag{4}$$

$$n_{fbb,12} = n_{12}C_{p_{12}} + c_{p_{12}}, \tag{5}$$

$$n_{rbb,12} = n_{12'}C_{p_{12}} + c_{p_{12}}, \tag{6}$$

$$d_1 = m_{b_3}D_{b_3} + m_{b_1}D_{b_1}, \tag{7}$$

$$d_2 = m_{b_3}D_{b_3} + m_{b_2}D_{b_2}, \tag{8}$$

$$0 \leq n_{12} \leq U_{n_{12}}, \tag{9}$$

$$-U_{n_{12}} \leq n'_{12} \leq 0, \tag{10}$$

$$0 \leq m_{b_1}, m_{b_2}, m_{b_3} \leq U_m, \tag{11}$$

$$S_1, S_2, T_{12}, t_{12} \text{ are random variables.} \tag{12}$$

into the permissible (i.e., discrete as in reality) set of tuning ranges.

IV. Continuous Optimization

A. Linear Programming Formulation

A sequential circuit is represented as a timing graph where each node represents a flip-flop and a directed edge from node u to node v represents the combinational logic paths from u to v. We only describe the mathematical formulation for a single edge (together with two nodes) in a timing graph. Other edges can be similarly handled. Suppose that in a clocked circuit, two flip-flops FF_1 and FF_2 are connected by combinational paths. The clock delay at FF_1 is S_1 and at FF_2 is S_2. The long (resp. short) critical combinational path delay between FF_1 and FF_2 is T_{12} (resp. t_{12}). All of S_1, S_2, T_{12}, t_{12} are random variables.

Without loss of generality, we assume that the combinational path p_{12} connecting FF_1 and FF_2 passes one circuit block. [4] demonstrates that in body biasing, delay reduction is linear with body voltage tuning. This fact is used here. To tune the path by ABB, we assume that the delay reduction due to body biasing along p_{12} is $n_{12}D_{p_{12}}$[1]. By linear fitting to the delay-area overhead data [4], $D_{p_{12}}$ is the slope of fitted line. For convenience, we call it "unit" delay reduction. n_{12} denotes the amount of ABB unit delay reduction applied to the circuit. The delay reduction comes with the area overhead due to e.g., control logic, extra well space, and extra power wires. It is measure by $n_{12}C_{p_{12}} + c_{p_{12}}$, where $C_{p_{12}}, c_{p_{12}}$ are also obtained from linear fittings. We call $C_{p_{12}}$ unit area overhead, and constant $c_{p_{12}}$ comes from e.g., shared control logics for the block of tunable elements. Similarly, we assume that the delay increase due to post-silicon tuning of PST clock buffer b_i is $m_{b_i}D_{b_i}$ with the area overhead $m_{b_i}C_{b_i} + c_{b_i}$ where D_{b_i} (resp. C_{b_i}) is the unit delay reduction (resp. the unit area overhead) for tuning a PST clock buffer, and c_{b_i} is constant indicating the overhead of control signals. The unified adaptivity optimization problem can be formulated as in Eqn. (1)-Eqn. (12).

Note that d_1 in Eqn. (7) and d_2 in Eqn. (8) are clock delay tuning at FF_1 and FF_2, respectively, and $n_{fbb,12}$ and $n_{rbb,12}$ are area overhead due to forward body biasing and reverse body biasing, respectively. $c_{b,12}$ is the area overhead due to clock tuning. They are introduced for the clarity of the formulation. S_1, S_2, T_{12}, t_{12} are the random variables obtained from statistical timing analysis. Only $m_{b_1}, m_{b_2}, m_{b_3}, n_{12}, n_{12'}$ are the decision variables in the formulation. Loosely speaking, $m_{b_1}, m_{b_2}, m_{b_3}$ are the amount of unit tuning applied on each clock buffer, n_{12} is the amount of unit forward body bias tuning applied on each circuit block (noting that they are non-negative), and n'_{12} is the amount of unit reverse body bias tuning applied on each circuit block (noting that they are non-positive). They can be any real values within certain ranges when solving the above linear programming problem. T_{cp} is the clock period, and T_{setup} and T_{hold} are the setup hold and the hold time of FF_2, respectively. Constants $U_{n_{12}}, U_m$ are the maximum tuning ranges which are allowed for each tunable element.

Eqn. (2) and Eqn. (3) contain random variables. To solve the linear programming with uncertainty, they will be cast into deterministic constraints using a technique proposed in the robust linear programming literature [9]. With theoretical guarantee, this technique enables us to specify a tradeoff parameter controlling the probability of constraint violation which is strongly related to timing yield. Such an approach has also been successfully used in [10] for clock scheduling. In our approach, we sample the subcircuits for simulations and use the results to decide the tradeoff parameter. That is, our usage of robust linear programming is adaptive. The details are omitted due to space limitation.

V. Discretization

In reality, it is very rare that a tunable delay element, either ABB or PST buffer, can be tuned continuously. In other words, the tuning is often allowed only for certain discrete steps. The second stage of the algorithm is to map the solution of the linear programming problem into a permissible (discrete) solution. Rounding up the tuning range of every tunable element to the continuous solution will obtain a discrete solution which satisfies the yield target but with more area cost overhead. In contrast, rounding down these tuning ranges could obtain a discrete solution with smaller area but not satisfying the yield target. With considerable size of the tuning step, nearest rounding may result in large error and a dedicated discretization algorithm is highly desirable. In our discretization process, we are to decide which one of the two choices should be used at each tunable element. This process consists of PST clock buffer tuning range rounding (i.e., clock rounding) and logic circuit tuning range rounding (i.e., logic rounding). Clock rounding is first performed and returns a set of solutions where all clock tuning ranges are discretized while logic tuning ranges are still continuous. Logic rounding is then

[1]Note that if reverse body biasing is used, the delay is increased, i.e., delay reduction becomes negative.

performed to some of the above solutions to discretize their logic tuning ranges.

A. Discretizating PST Clock Buffers

A dynamic programming approach is first used to determine discrete tuning range for each PST buffer in the clock tree. We define a *partial clock tuning solution* to be an incomplete determination for the discrete tuning ranges of all clock buffers. A partial clock tuning solution becomes *clock-complete* when the discrete tuning ranges of all clock buffers are determined. A clock buffer is *processed* if its discrete tuning range has been determined. The algorithm starts with the root of the clock tree, and performs a breadth-first traversal on the clock tree. During the process, we set the tuning range for each tunable clock buffer to each of two possible choices (up-rounding and down-rounding). The acceleration technique is necessary.

1) Solution Characterization: A set of partial solutions, denoted by $\mathcal{A}$, keep being updated during the process of dynamic programming. Each solution $\alpha \in \mathcal{A}$ is associated with a (C, Y) pair, where C denotes the cumulative area overhead and Y denotes the estimated yield. C is computed by summing the overhead of all processed PST buffers. To compute the yield of the circuit, discrete tuning range for every tunable element needs to be known. Since some of them are not processed, we will use their continuous tuning ranges for yield estimation. This makes sense as our goal for performing discretization is to obtain a discrete solution with yield and overhead close to the continuous solution.

2) Solution Propagation: Suppose that we are to decide the tuning range of a PST buffer b. A new solution α' will be formed for each of the two possible choices (up-rounding and down-rounding). Because all PST buffers are processed according to the breadth-first order, when b is processed, the cumulative area overhead can be updated by $C(\alpha') = C(\alpha) + C(b)$, where $C(b)$ is the area overhead due to b. $Y(\alpha')$ is obtained by simulations, precisely, a fast yield estimation through Latin Hypercube sampling based Monte Carlo simulations [11].

3) Acceleration by Pruning: The acceleration comes from the observation that we do not need to always update the yield of every partial solution during solution propagation.

After processing a node u, we obtain a set of partial solutions $\mathcal{A}_u$, each of which satisfies the yield target. Suppose that the next node to be processed is v. For each solution in $\mathcal{A}_u$, a new solution is generated by rounding up the PST buffer tuning range at v. Denote the resulting solution set by $\mathcal{A}_{v,+}$. Late yield update is applied on them. That is, their yields are not computed at this moment since we know that they must satisfy the yield target. For each solution in $\mathcal{A}_u$, a new solution is also generated by rounding down the PST buffer tuning range at v. Denote the resulting solution set by $\mathcal{A}_{v,-}$, which is sorted by C values. We are to perform yield estimation on $\mathcal{A}_{v,-}$ since some of them may not meet the yield target. Our yield estimation is performed in a binary search fashion. That is, we first estimate the yield for the middle solution. If it satisfies the yield target, the middle solution for the half solution set with smaller C will be tested. Otherwise, the current solution and the half solution set with smaller C will be pruned, and the middle solution for the half solution set with larger C will be tested. The process is repeated for $\log|\mathcal{A}_{v,-}|$ times. Denote the resulting solution set by $\mathcal{A}'_{v,-}$ and then $\mathcal{A}_v = \mathcal{A}_{v,+} \bigcup \mathcal{A}'_{v,-}$. During solution propagation, when the size of the solution set $\mathcal{A}$ is larger than a threshold w, top $w/2$ solutions with smallest C values are kept and all other solutions are pruned for further speedup.

B. Discretizing Logic Circuits

Section V-A returns a set of solutions, denoted by $\mathcal{A}$, which are sorted according to C values. For each solution in $\mathcal{A}$, clock tuning ranges have already been discretized while logic tuning ranges are not. This subsection deals with discretizing the logic tuning ranges and selecting which rounding solution to return.

The solution set $\mathcal{A}$ can be large and it is time consuming to perform logic rounding on each solution. Thus, similar to Section V-A.3, a binary search fashion algorithm is applied on the solution set $\mathcal{A}$. That is, the middle solution is first rounded (by logic rounding) followed by the middle solution in one half of $\mathcal{A}$. Finally, the solution (where both clock rounding and logic rounding have been performed) to be returned is the one with the smallest overhead while satisfying the yield target. We only describe how to round a single solution.

Recall that our body bias tuning is applied at the circuit block level (i.e., the whole circuit block has the same tuning range), we will discretize the tunable ranges for tunable circuit blocks. Each tunable circuit block is assigned with a timing criticality related reducibility which measures the possible area overhead reduction while still satisfying yield target. A large reducibility means the large possibility of area reduction. Since larger slack means that we have larger room for area reduction, and the area overhead is also proportional to the number of tunable gates, the following cost function is used for a path $\mathfrak{p}$ passing through a block B.

$$reducibility(\mathfrak{p}) = slack(\mathfrak{p}) \times tunablegates(\mathfrak{p}), \qquad (13)$$

where $slack(\cdot)$ is the sum of the slack of the gates along $\mathfrak{p}$ in B and $tunablegates(\cdot)$ is the number of tunable gates along $\mathfrak{p}$ in B. Note that the nominal slack is used as an estimation for the slack, which makes sense since variations in the block should have similar impact on all paths passing through it. Since all critical combinational paths are given as the input to our algorithm, the paths passing through a block can be easily identified. The reducibility of a block is then defined as the minimum reducibility cost for all paths passing through it since we need to guarantee that (almost) all paths could satisfy the timing constraint. That is,

$$reducibility(B) = \min_{\mathfrak{p}} reducibility(\mathfrak{p}). \qquad (14)$$

To perform logic rounding, a batch-based optimization technique is used after the reducibility costs for all blocks are computed. This technique is also used in [7] for clock tuning which is shown to be much more efficient than greedy approach.

C. Fast Simulations For Timing Yield Estimation

Discretization involves simulations for yield estimation. In simulations, the tuning range of a tunable element needs to be fixed. This is implemented as modifying the corresponding constraints in the linear programming formulation. For example, discretizing n_{12} in Eqn. (2) to 10 is implemented as setting $n_{12} \leq 10$ in Eqn. (9). During discretization, we may frequently estimate the new timing yield of the circuit (for not-yet discretized tuning ranges, their continuous tuning ranges are used). The commonly-used Monte Carlo simulation approach is very time consuming. Thus, fast Latin Hypercube (LH) sampling based Monte Carlo simulations are used. Compared to simple sampling, LH sampling allows us to sample the space more evenly to avoid crowded samples. By this, one can use much fewer samples in Monte Carlo simulations while still having high yield estimation accuracy [11]. Refer to [11] for the details.

VI. Experiments

The continuous linear programming algorithm and discretization algorithm are implemented in C++ and are tested on a Pentium IV computer with a 3.0GHz CPU and 2G memory. ISCAS'89 benchmark circuits and a cell library of $130nm$ technology are used in the experiments.

In our experiments, the yield target is set to 99.0%. In our continuous approach, yield estimation is implicitly used in the robust linear programming technique. In our discretizartion approach, Latin Hypercube sampling based Monte Carlo simulations are used for yield estimation. These yield estimations are very fast, but may have small errors. In order to compensate for such errors, we set the yield constraint in the optimization to be 99.5% which is slightly higher than our target of 99.0%. This *target compensation* idea can compensate for estimation errors and make our results satisfy the yield target in practice. After optimization, 5000 Monte Carlo simulations are performed to evaluate the timing yield for the obtained circuits. Their runtime is not included in the algorithms since they are not in optimization.

A. Continuous Adaptivity Optimization

Our first experiment is to investigate the difference between the unified optimization and the optimization on logic or clock signal adaptivity alone. Recall that our continuous optimization problem is solved using a robust linear programming technique. A single *tradeoff parameter* is used to achieve different cost-yield tradeoff and no Monte Carlo simulation is needed during optimization. The optimization for logic or clock adaptivity alone follows the above procedure except that additional constraints are introduced to ensure that the optimization is not performed for both clock and logic signals simultaneously. The results are summarized in Table I. Note that due to the target compensation idea, all results satisfy the yield target 99.0%. We make the following observations:

- Unified optimization saves large amount of area compared to optimization on logic signal or clock signal alone. For s5378, 50.5% area cost reduction is obtained.
- Clock Signal Adaptivity leads to small improvement on yield. Thus, linear programming often returns no feasible solutions for the same tradeoff parameter as in the unified optimization and Logic Signal Adaptivity. This is the case since there may be many critical cycles in a circuit, and when these cycles are heavily overlapped, there is only slight of amount of slack can be moved among flip-flops. In addition, a single clock buffer may drive many flip-flops and tuning it affects all of them. If some flip-flops are in timing critical paths, then only small amount of tuning can be applied to the clock buffer.
- Since the runtime only comes from formulating the linear program and solving it using the robust linear programming technique, all algorithms run very efficiently.

B. Discretization

We then perform discretization algorithm to the continuous solution obtained from the unified optimization. Since there is no previous work on unified optimization on the adaptivity of logic and clock signals, we compare our approach to the following two simple methods. The first approach is called Binary Batch algorithm which is in the same spirit as [7]. Initially, we plan to compare to a greedy algorithm where each circuit block is tuned separately. However, we found that this approach is intolerably slow. The batch based optimization [7] gradually increases the tuning range of a set of blocks where the blocks are picked according to the criticality. In order to further improve its efficiency, binary search is performed on the tuning steps (i.e., tuning range can be increased by various multiples of discrete tuning steps) to largely reduce the total number of Monte Carlo simulations. The second algorithm for comparison is a Nearest Rounding approach. In this approach, the tuning range of each tunable element is simply rounded to the nearest discrete tuning range. All approaches involve Latin Hypercube sampling based Monte Carlo simulations [11] and 200 LH samples are used in simulations for a yield estimation. Note that the yield constraint is pushed to 99.5% by our target compensation idea.

The discretization results are summarized in Table II. Note that the runtime for computing continuous solution is included for Nearest Rounding and Our Discrete Solution. We make the following observations.

- Nearest rounding often leads to large rounding error, i.e., timing yield is significantly decreased. For s5378, yield is below 90%.
- Since Binary Batch approach is not guided by our continuous solution, the solution quality is quite low compared to other approaches. It often doubles the area compared to the discretization approach. In addition, it takes much longer time to run. This is again due to that continuous solution is not used. For the other two approaches, we only have two choices (i.e., rounding-up or rounding-down) at each tunable element, which means that the search space has been greatly reduced.
- Our discretization approach achieves good balance between solution quality and runtime. Due to our target

TABLE I

CONTINUOUS OPTIMIZATIONS ON ISCAS'89 BENCHMARK CIRCUITS. #BK REFERS TO THE NUMBER OF BLOCKS AND #BUF REFERS TO THE NUMBER OF CLOCK BUFFERS. AREA REDUCTION IS OBTAINED BY COMPARING THE AREA OF OUR DISCRETE SOLUTION WITH THE MINIMUM AREA OF LOGIC SIGNAL ADAPTIVITY AND CLOCK SIGNAL ADAPTIVITY.

Circuit					Logic Signal Adaptivity			Clock Signal Adaptivity			Unified Adaptivity			
Name	#FF	#Gates	#Bk	#Buf	Area	Yield	CPU (s)	Area	Yield	CPU (s)	Area	Yield	CPU (s)	Area Red.
s838	32	446	25	45	185.1	99.7%	5.5	-	-	-	103.7	99.8%	5.5	43.9%
s1238	18	508	25	23	204.1	99.8%	1.1	218.5	99.8%	1.1	112.2	99.9%	1.1	45.0%
s1423	74	657	25	72	216.2	99.4%	51.2	-	-	-	206.5	99.7%	51.5	4.5%
s1488	6	653	25	16	230.8	99.1%	1.2	-	-	-	201.9	99.3%	1.2	12.5%
s5378	179	2779	25	128	207.5	99.3%	60.9	213.7	99.2%	61.2	102.7	99.5%	61.1	50.5%
s15850	534	9835	49	239	1081.7	99.4%	342.5	-	-	-	703.6	99.3%	341.7	34.9%
s35932	1728	16065	64	438	2077.5	100.0%	1007.8	-	-	-	1532.8	99.7%	1012.5	26.2%
s38417	1636	22179	64	393	1728.2	99.3%	570.5	-	-	-	1339.7	99.5%	569.4	22.5%
s38584	1426	19279	64	357	2593.2	99.7%	667.5	-	-	-	1820.8	99.5%	668.5	29.8%

TABLE II

DISCRETE SOLUTIONS FOR ISCAS'89 BENCHMARK CIRCUITS. RUNTIME FOR COMPUTING NEAREST ROUNDING AND DISCRETE SOLUTION INCLUDES THE RUNTIME FOR COMPUTING CONTINUOUS SOLUTIONS. AREA REDUCTION AND SPEEDUP ARE OBTAINED BY COMPARING TO BINARY BATCH.

Circuit	Binary Batch			Nearest Rounding			Our Discrete Solution				
Name	Area	Yield	CPU (s)	Area	Yield	CPU (s)	Area	Yield	CPU (s)	Speedup	Area Red.
s838	259.9	99.8%	552.7	86.5	90.5%	5.7	105.3	99.5%	242.0	2.3×	59.5%
s1238	222.0	99.9%	135.7	100.9	91.7%	1.4	115.0	99.7%	83.7	1.6×	48.2%
s1423	353.2	99.1%	685.3	229.3	97.5%	52.0	223.1	99.3%	248.5	2.8×	36.8%
s1488	545.9	99.3%	187.2	213.3	97.2%	1.6	214.6	99.4%	86.7	2.2×	60.7%
s5378	303.5	99.7%	935.8	89.8	89.2%	61.4	117.5	99.7%	465.8	2.0×	38.7%
s15850	1297.1	99.3%	4158.5	655.3	90.2%	346.7	778.0	99.5%	2310.2	1.8×	40.1%
s35932	2970.6	99.7%	19532.7	1729.2	95.8%	1024.3	1711.9	99.6%	9844.2	2.0×	57.6%
s38417	2513.1	99.8%	7822.1	1512.9	93.1%	576.8	1578.2	99.2%	4343.6	1.8×	37.2%
s38584	3189.2	99.8%	14102.5	1702.9	91.0%	676.8	1955.2	99.8%	5948.3	2.4×	38.7%

compensation idea and the effectiveness of LH sampling based Monte Carlo simulations, the exact yield always satisfies the target which is 99.0%. This is not the case for Nearest Rounding. For s1423 and s35932, our approach returns solutions with better yield and smaller area cost compared to Nearest Rounding. As our discretization approach maintains a set of solutions in computation (in both clock and logic rounding), it runs slower than Nearest Rounding approach. However, due to various acceleration techniques, our approach is still more efficient than [7]. Note that Binary Batch is actually a faster version of [7], and our discretization approach is consistently better than Binary Batch in terms of both area and runtime.

VII. CONCLUSION

To the best of the authors' knowledge, this work is the first one considering unified adaptivity optimization on logic and clock signals, which saves much area cost compared to optimization on logic or clock signals alone and it does need any assumption on variation distributions. Our continuous optimization is based on a novel linear programming formulation which is efficiently solved by a robust technique where no Monte Carlo simulation is needed. To compute the discrete solution, the continuous solution is used to guide the discretization process to greatly reduce search space. This process involves binary search accelerated dynamic programming, batch based optimization, and Latin Hypercube sampling based fast simulation. Our experimental results demonstrate that up to 50% area cost reduction can be obtained by the unified tuning and the discretization significantly outperforms the alternatives in terms of solution quality and runtime.

REFERENCES

[1] M. Mani, A. Devgan, and M. Orshansky, "An efficient algorithm for statistical minimization of total power under timing yield constraints," *in DAC*, pp. 309–314, 2005.

[2] A. Agarwal, K. Chopra, D. Blaauw, and V. Zolotov, "Circuit optimization using statistical static timing analysis," *in DAC*, pp. 321–324, 2005.

[3] J. Singh, V. Nookala, Z.-Q. Luo, and S. S. Sapatnekar, "Robust gate sizing by geometric programming," *in DAC*, pp. 315–320, 2005.

[4] S. Kulkarni, D. Sylvester, and D. Blaauw, "A statistical framework for post-silicon tuning through body bias clustering," *ICCAD*, pp. 39–46, 2006.

[5] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1396–1401, 2002.

[6] M. Mani, A. Singh, and M. Orshansky, "Joint design-time and post-silicon minimization of parametric yield loss using adjustable robust optimization," *ICCAD*, pp. 19–26, 2006.

[7] J.-L. Tsai, L. Zhang, and C.-P. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," *ICCAD*, pp. 575–581, 2005.

[8] V. Khandelwal and A. Srivastava, "Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation," *ISPD*, pp. 11–18, 2007.

[9] D. Bertsimas and M. Sim, "The price of robustness," *Operations Research*, vol. 52, no. 1, pp. 35–53, 2004.

[10] V. Nawale and T. Chen, "Optimal useful clock skew scheduling in the presence of variations using robust ILP-formulations," *ICCAD*, pp. 27–32, 2006.

[11] K. Fang, K. Fang, and L. Runze, "Design and modelling for computer experiments," *CRC Press*, 2005.