PROXIMITY OPTIMIZATION FOR ADAPTIVE CIRCUIT DESIGN

A Thesis

by

ANG LU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jiang Hu
Committee Members,	Peng Li
	Anxiao Jiang
Head of Department,	Miroslav M. Begovic

December 2015

Major Subject: Computer Engineering

ABSTRACT

The performance growth of conventional VLSI circuits is seriously hampered by various variation effects and the fundamental limit of chip power density. Adaptive circuit design is recognized as a power-efficient approach to tackling the variation challenge. However, it tends to entail large area overhead if not carefully designed. This work studies how to reduce the overhead by forming adaptivity blocks considering both timing and physical proximity among logic cells. The proximity optimization consists of timing and location aware cell clustering and incremental placement enforcing the clusters. Experiments are performed on the ICCAD 2014 benchmark circuits, which include case of near one million cells. The experiment results prove that during clustering, location proximity among logic cells are equally important as the timing proximity among logic cells. Compared to alternative methods, our approach achieves 25% to 75% area overhead reduction with an average of 0.6% wirelength overhead, while retains about the same timing yield and power consumption.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND OVERVIEW

1.1 Introduction

Variability, such as process variations and device aging, and power are notorious barriers to the progress of VLSI technology. Process variations are introduced during the fabrication of die. Two dies, though they are designed to be the same, but during fabrication, there will be some differences, these differences are categorized as inter-die variation. Also, on the same die, even two MOSFETs are designed to be the same, there will be some differences in their gate width, length, and oxide thickness. These differences are categorized as intra-die variation. As technology aggressively scaled and size of die grows, circuits has been increasingly susceptible to process variations. On the other hand, aging degradation has become a major reliability issue in sub-130nm technologies. Negative Bias Temperature Instability (NBTI) manifests itself as a temporal increase in the threshold voltage of a PMOS transistor, thereby causing circuit delays to degrade over time and exceed their specifications [4]. A dual effect is Negative Bias Temperature Instability (NBTI) for NMOS transistor.

The compound effect of process variations and device aging is even more difficult to deal with. Variations demand extra power for timing margins and therefore exacerbate power dissipation. On the other hand, increasingly tight power budget seriously hinders design techniques for variation tolerance. Adaptive circuit design is an approach to getting out of this difficult situation.

An adaptive circuit, as the Figure 1.1 shows, contains sensors that detect timing variations. Broadly speaking, there are two kinds of sensors: critical path replica [6] and canary flip-flop [8]. The sensor outputs control certain tuning knobs, such as body bias [9] and supply voltage change [5], such that timing variations are corrected.

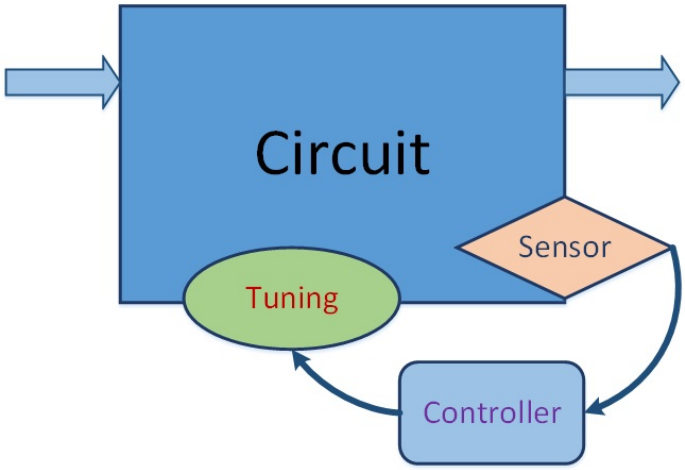

Figure 1.1: A scratch of a most basic adaptive circuit.

Unlike conventional methods, which allocate extra power and timing margins according to the worst case variations, an adaptive circuit spends additional power only when timing variation is actually observed. Adaptive design is conceptually more power-efficient than conventional designs, however, it entails area overhead on sensors, tuning circuits and control wires. If not carefully designed, the overhead can be quite significant. For example, a naïve implementation of the voltage interpolation technique [5] can double chip power grid.

There are two mainstream tuning methods, adaptive body bias (ABB), and adaptive supply voltage (ASV). ABB takes advantage of the body effect phenomenon to modulate the V_{th} of a MOSFET. It is composed by forward body bias (FBB) and reverse body bias (RBB). FBB lowers the V_{th}. It increases the speed of the MOSFET but increases leakage power. On the contrary, RBB reduces leakage power at the cost of an increase delay. The idea of ASV is also easy to understand. Instead of one V_{DD} to be the supply voltage of the whole circuit like conventional circuits do, there will be two V_{DD}'s, one high V_{DD} and one low V_{DD}. High V_{DD} means more power dis-

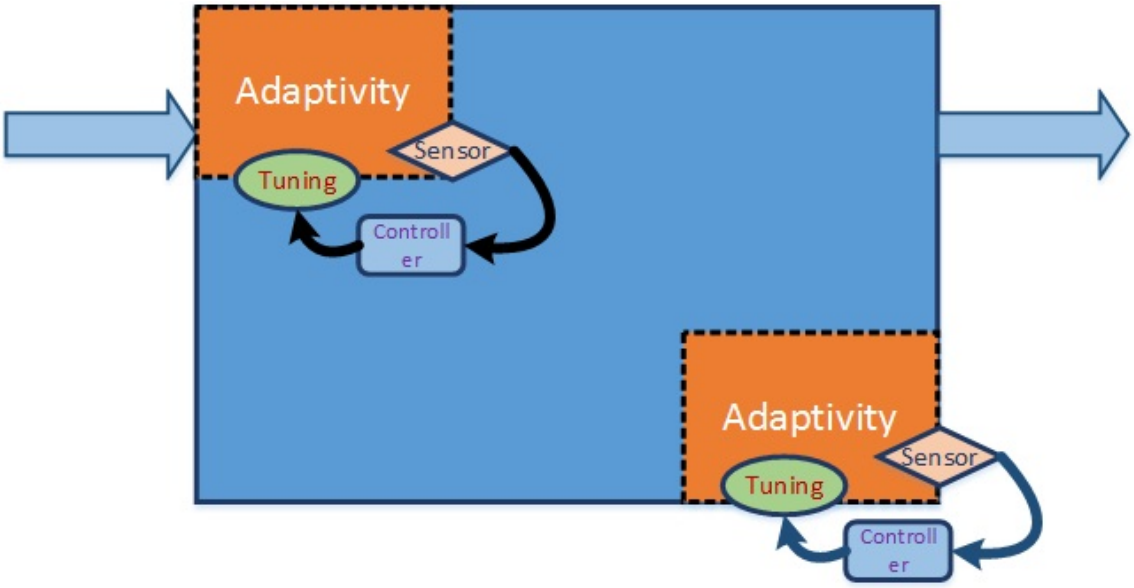

Figure 1.2: A scratch of a most basic adaptive circuit, but grouped into two clusters.

sipation and better performance, while low V_{DD} means less power and longer delay. Both methods are trade-off between power and performance.

Indeed, the overhead issue is a key reason that prevents a wide application of adaptive design techniques. The efficiency of an adaptive design highly depends on how its adaptivity blocks are formed. The circuit cells within one block share the same sensors, control and tuning knobs. Ideally, one prefers to put cells that need similar tuning actions into the same block. As such, a small number of sensors and knobs can cover a large circuit and the amortized overhead is relatively low. At the same time, cells in a block need to be physically close to each other or form a contiguous region. As the Figure 1.2 shows, each orange area represents a cluster. In this figure, there are two clusters, and compare to Figure 1.1, there are much less sensors and knobs needed. Separating cells of the same block far apart would at least cause unnecessarily large control wire overhead. Overall, adaptivity blocks should be formed according to both timing proximity and spatial proximity of cells.

The adaptivity block generation problem is studied in [2] for adaptive body bias. It estimates timing proximity among cells by Monte Carlo simulation of body bias assignment. The assignment assumes that the body of each cell can be individually controlled and is implemented by quadratic programming. After the simulation, the probability distribution of body bias tuning for each cell is obtained. Then, cells with similar distributions and are highly correlated are clustered to form a block. It is observed that the spatial correlation among tuning actions of different cells is similar as physical proximity among the cells. To ensure that cells in the same block are located in a contiguous region, incremental placement change is performed using the CAPO placer [7]. This is a pioneer work that demonstrates the importance of adaptivity clustering. However, it has a few drawbacks. First, Monte Carlo simulation of quadratic programming is very time consuming and difficult to scale to large cases. For example, a 32K-cell case, which is fairly small from the point of view of modern IC design, costs nearly four and half hours runtime in [2]. Second, it is not described how Capo [7] enforces spatial continuity among cells in a cluster. Third, although timing and spatial proximity are correlated, the difference between them cannot be neglected. For example, the timing-only clustering [2] results in 3% wirelength overhead, which is not trivial. The clustering method of [2] is integrated with cell sizing in [10].

In this work, we propose a balanced approach: cell clustering with consideration of both timing and spatial proximity. To make the clustering more scalable, the consideration of timing is based on timing analysis result instead of Monte Carlo simulation of quadratic programming. The clustering is followed by an incremental placement that enforces the spatial continuity of cells in a cluster. The placement is formulated and solved by min-cost network flow model that minimizes total cell movement. Experiments are performed on the ICCAD 2014 incremental timing-

driven placement contest benchmark suites, which include circuit of near one million cells. Compared to timing-only and location-only clustering, our approach achieves $\frac{1}{4}$ to $\frac{3}{4}$ area overhead reduction with an average of 0.6% wirelength overhead. At the same time, it retains about the same timing yield and power consumption.

1.2 Overview of our design flow

The input to our method is a combinational logic circuit, timing constraints, and adaptivity policy. An overview of this design flow is sketched in Figure 1.3. As shown in the figure, the design flow is composed by three stages.

In stage 1, cell placement (including detailed placement) and timing analysis are performed. It is represented by a graph $G = (V, E)$, where the node set V indicates cells and edges E imply fanin/fanout among cells. Stage 2 is the main part of our project. It is composed by two phases: phase I: Timing and location aware cell clustering and phase II: cluster driven incremental placement. The clustering algorithm partitions cells into blocks, where cells in a block have similar timing and location characteristics. The incremental placement further forces cells of each cluster to form a contiguous region. After detailed placement, we check the increase in wire-length, if the wire-length increases over τ%, then we re-run the clustering and incremental placement algorithm. Then in stage 3, the result of our method is fed to the adaptive circuit optimization [11], which decides if to assign adaptivity to each block and simultaneously performs gate sizing.

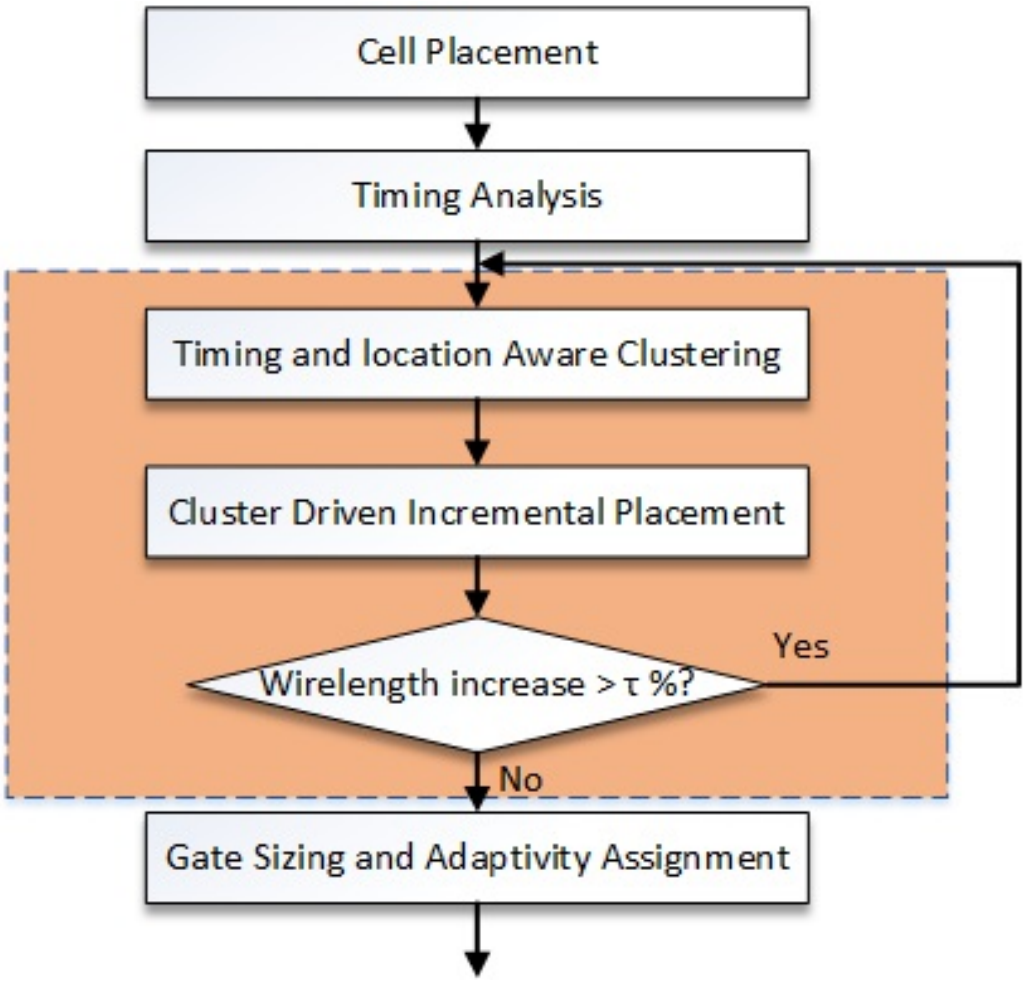

Figure 1.3: Overview of proposed design flow.

2. METHODOLOGY

Our design flow compose of a three-stage optimization flow to determine the clusters to which gates are assigned as well as the specific adaptivities for gates. In this chapter, I will describe each stage in detail. But since stage 2 is the main stage in our design flow, I will explain it more specifically.

2.1 Generate a general placement and perform timing analysis on the placement

This is stage I in our design flow. First, perform a global placement on the benchmark circuits. The placement tool we used in this paper is CAPO. The global placement will then be used to provide placement information for timing analysis. The timing analysis tool will collect timing information, which will then be used in later stages.

2.1.1 Perform global placement using CAPO placement tool

In this sub-step, we perform global placement using CAPO placement tool [7]. CAPO is chosen in our design flow due to the fact that it is a fast and high-quality routability-driven placer for standard-cell ASICs. It reads the LEF/DEF format of the benchmark suites we use. It performs global placement with recursive bisection using leading-edge multi-level partitioner. And perform detailed placement with optimal branch-and-bound partitioner and placer to eliminate overlap. After CAPO is performed, the placement provides the physical location of each gates needed for timing analysis and clustering. Moreover, the placement is legal and routable everywhere, which means later when incremental placement is performed in stage 2, no

effort will be made in eliminating overlap.

2.1.2 Perform statistical static timing analysis

In this step, statistical Static Timing Analysis (STA) is performed to provides the timing information needed for k-means clustering, which includes *nominal slack* of each gate, *criticality* of each gate, and *critical path delay* of the circuit. The definition of gate's criticality is: *the change of the critical path delay of the circuit as a result to the change of the gate's delay.* Since adaptivity is operated according to observed variations, variations need to be taken into consideration in the timing analysis. Traditional STA techniques lie in their deterministic nature so it is not suitable for our application. Monte Carlo simulation is the very accurate, however, the runtime is too long. Statistical STA lies in between STA and Monte Carlo simulation. It treats delays not as fixed numbers, but as probability density functions (PDFs), taking the statistical distribution of parametric variation into consideration while analyzing the circuit. It is accurate enough [3], while the runtime of SSTA is still acceptable.

The statistical STA tool we applied [3] derived from the timer introduced in [1]. It computes the distribution of circuit delay while considering spatial correlations. Spatial correlations between gates are modeled by storing them in a grid-based correlation matrix. Process variation we considered including gate length variation, whose standard deviation σ is 5% of nominal value, and gate width variation with σ begin 2.7% of nominal value. In order to manipulate the complicated correlation structure, the Principal Component Analysis (PCA) technique is employed to transform the sets of correlated parameters into sets of uncorrelated ones. The statistical timing

computation is then performed with a PERT (Program Evaluation and Review Technique) -like circuit graph traversal [1]. As tested on ISCAS'85 circuits, errors of the statistical STA results compare to Monte Carlo simulation results on mean (μ) and standard deviation (σ) are 0.04% and 5.7% respectively [3], which are acceptable for our application.

2.2 K-means clustering and incremental placement

This is stage II in our design flow. In this stage, we use an efficient clustering technique to group gates into a number of clusters. And then, incremental placement is performed based on the clustering result to move gates within the same cluster to a contiguous region, so that they could be tuned with an identical adaptivity configuration.

2.2.1 Timing and location aware cell clustering

We start with an example in Figure 2.1 to illustrate that considering only timing in clustering like [2] is insufficient. The argument in [2] is that timing correlations highly depend on spatial correlations. As such, spatial proximity is largely addressed by considering only timing proximity. This statement is often true, however, it is not difficult to find counter examples that easily happen in practice. In Figure 2.1, there are two timing critical paths: one from gate $A1$ to $A2$ and the other from $B1$ and $B2$. Along path A (B), forward body bias (FBB) of either gate $A1$ ($B1$) or $A2$ ($B2$) can fix timing error. If FBB of NAND gates is slightly more efficient than FBB of NOR gates, the quadratic programming in [2] may mostly choose FBB of $A1$ and $B2$ at the same time. Then, the clustering of [2] would put $A1$ and $B2$ into the same cluster although they are spatially far apart. The subsequent incremental placement

must move those cells for a long distance to bring them together. Consequently, wirelength is significantly increased. Moreover, the large cell moves may invalidate the original timing analysis result. For the example in Figure 2.1, a better solution is to cluster $A1$ ($A2$) with $B1$ ($B2$).

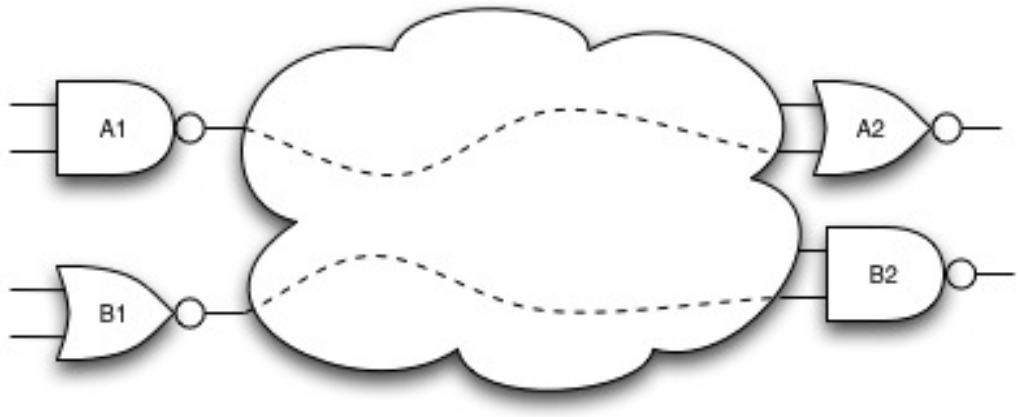

Figure 2.1: An example for clustering.

In our clustering, both timing and spatial proximity are considered. Straightforwardly, spatial proximity between two cells are estimated by the Manhattan distance between them.

Timing proximity is much more complex. Ideally, it should indicate the probability that two cells take the same tuning actions. Monte Carlo simulations of quadratic programming like [2] serves this purpose well, but is too expensive to use. Therefore, we resort to a simple surrogate metric that include two factors. The first factor is the timing slacks at cells. If two cells have similar slack, then it is more likely that they would take the same tuning actions. In the cases where tuning knobs are only for high performance and not for low power, e.g., only forward body bias, when two

cells have huge slack, none of them would take tuning action regardless how large their slack difference is. Hence, a cell g_i is characterized by *capped slack* defined as

$$\hat{s}_i = \min(slack_i, \theta) \tag{2.1}$$

where θ is a constant threshold. To further account for timing variability, the slack here is based on nominal delay plus scaled σ (standard deviation) of the delay. It is conceivable that a cell with large σ is more likely to be tuned when it has the same nominal delay as others.

The second factor in the surrogate timing proximity is sensitivity, which is defined as ratio of slack increase by tuning a cell versus the tuning cost, i.e.,

$$\psi = \frac{Critical\_Path\_Slack\_Increase}{Tuning\_Cost} \tag{2.2}$$

where the tuning cost can be power increase or area overhead of adaptivity. When two cells have similar timing slack, the sensitivity may make a difference on if to take tuning action. Indeed, it makes sense for tuning policies to favor change on cells with relatively large sensitivities.

Overall, the distance between g_i at (x_i, y_i) and g_j at (x_j, y_j) in the clustering is defined as

$$d_{i,j} = \alpha \cdot |\hat{s}_i - \hat{s}_j| + \beta \cdot |\psi_i - \psi_j| + \gamma \cdot (|x_i - x_j| + |y_i - y_j|) \tag{2.3}$$

where α, β and γ are constant parameters. Usually, the value of β is much smaller than α.

Based on the distance defined above, we adopt Lloyd's K-means algorithm for the clustering. To make the description complete, we summarize the main steps of

this algorithm. It starts with K arbitrary means or centers. Then, each element is assigned the cluster with nearest center. After the assignment, the centers are updated with the centroids of the clusters. This assignment and center update procedure is repeated till the within-cluster sum of distance (WCSD) converges to the minimum. WCSD is defined by

$$\sum_{i=1}^{K} \sum_{\vec{x} \in C_i} |\vec{x} - \vec{\mu}_i| \tag{2.4}$$

where C_i is a cluster, $\vec{x}$ is an element and $\vec{\mu}_i$ is the mean or center for cluster C_i. Unlike the original Lloyd's algorithm, which is based on Euclidean distance, we use Manhattan distance to match the layout convention in VLSI circuits. The value of K is decided empirically [11]. Moreover, we allow K to be changed according to clustering results. If two clusters are very near to each other, they are merged and K is therefore decreased.

2.2.2 Cluster driven incremental placement

After the clustering, a small number of cells are often located away from the majority cells of their own clusters. For example, in Figure 2.2, where clusters are indicated by colors, two blue cells and one orange cell are away from their clusters. We call them *alien cells*. Due to alien cells, control wires for a cluster must span a relatively large region. Moreover, tuning overhead, such as extra power lines in voltage interpolation [5], is also increased by the spreading out of clusters.

The purpose of incremental placement is to move alien cells back to their clusters such that each cluster forms a compact and contiguous region. An alien cell g_i^k belonging to cluster C_k can be moved to an empty space among majority cells of C_k.

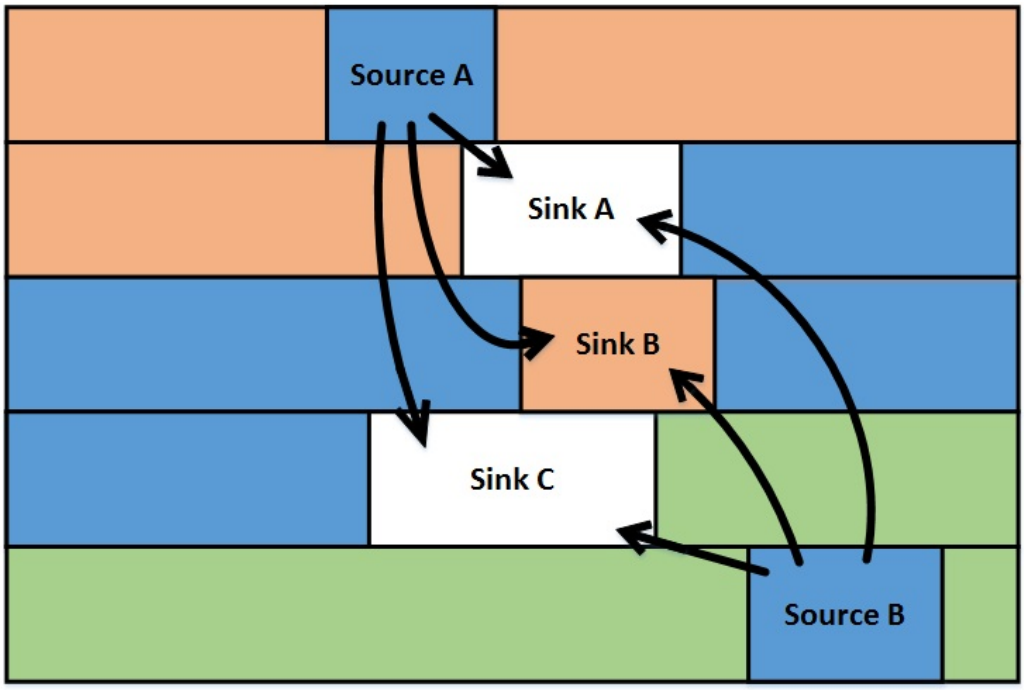

Figure 2.2: An example of cell placement after clustering. The white regions are empty.

Alternatively, it can be moved to the position of another alien cell g_j^l that belongs to another cluster C_l but sits within cluster C_k, as g_j^l will be moved out of cluster C_k sooner or later. For example, the blue cell in top row of Figure 2.2 can be moved to the location of the orange cell in the middle row. Of course, these moves are allowed only if the size of empty space or g_j^l is no less than that of g_i^k. In order to retain the original design as much as possible, the total cell movement needs to be minimized at the same time.

In essence, the incremental placement is a min-cost assignment problem – assigning alien cells to empty or potentially empty space. In general, an assignment problem can be solved through min-cost network flow model. However, there is a pitfall. That is, if one attempts to move all alien cells simultaneously in a network

flow model, it is difficult to ensure that an alien cell is moved to its own cluster, not other clusters. In fact, this is a multi-commodity network flow problem, which is NP-complete. On the other hand, this issue is not difficult to circumvent, simply by processing one cluster at a time. Specifically, alien cells belonging to one cluster are collected back using min-cost flow model.

Since the clusters are processed one at a time, we need to find the order for processing them. The order is based on cluster porosity, i.e., the percentage of space can be used by its alien cells. A cluster with low porosity is processed first. White space between two clusters can be claimed by either cluster. Processing low porosity (high density) clusters first would allow them to have high priority for taking white space between clusters. Evidently, if the overall placement density is not high, this order does not matter.

Now we describe the network flow model for moving alien cells belonging to cluster C_k back to C_k. The network is a directed graph $G' = (V', E')$. The node set V' is composed by the following types of nodes:

- *Source node*: Each source node corresponds to an alien cell g_i^k that needs to be moved to cluster C_k.
- *Sink node*: Each sink node indicates (1) a contiguous empty space inside or adjacent with cluster C_k, or (2) an alien cell g_j^l that sits inside C_k.
- *Super source* S: This is a virtual node and there is an edge from S to every source node.
- *Super sink* T: This is a virtual node and there is an edge from every sink node to T.

There are three types of edges in E'.

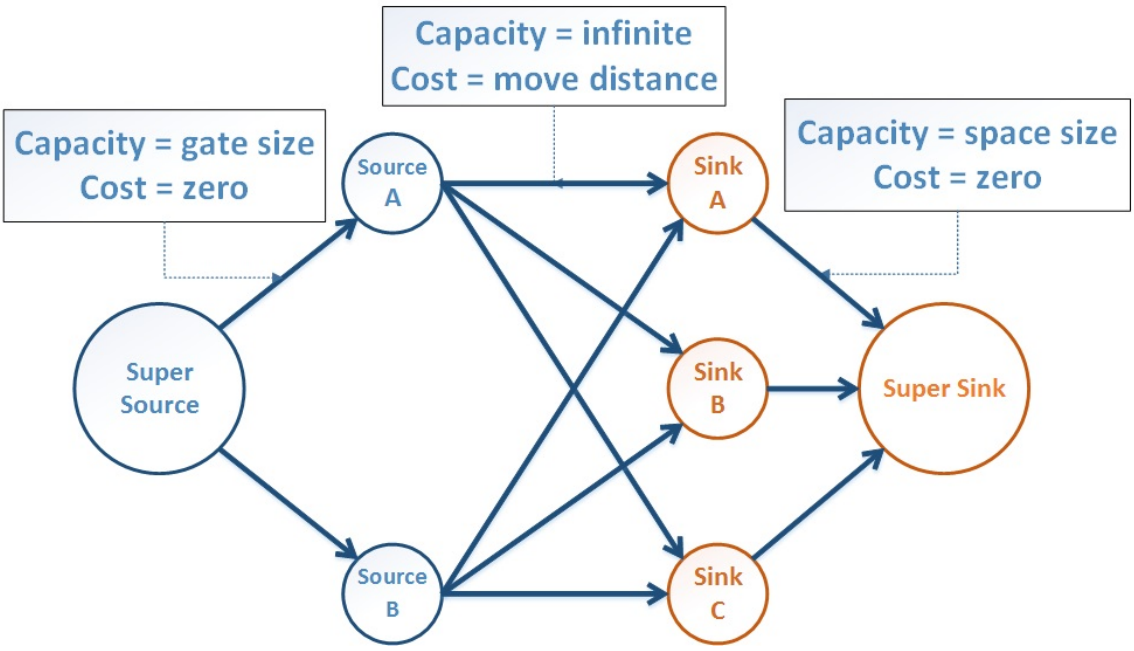

Figure 2.3: Network of max flow min cost problem

- *From S to source nodes*: Each such edge has capacity equal to the size of corresponding alien cell, and cost of 0.
- *From source to sink node*: There is an edge between every pair of source and sink nodes. Its capacity is infinity and its cost is equal to the distance of moving the corresponding alien cell to C_k.
- *From sink node to T*: Each such edge has capacity equal to the size of corresponding empty space or alien cell that does not belong to C_k. The edge cost is 0.

Figure 2.3 shows the network flow model for moving the two blue alien cells in Figure 2.2. The flow constraint is equal to the total size of alien cells to be moved. In practice, placement density is rarely near 100% and the percentage of alien cells is small. Hence, there is usually plenty of white space accommodating the moves. After

the model is formulated, the Edmonds-Karp algorithm [12] is performed to obtain min-cost flow solution. The algorithm can guarantee to find the optimal solution in polynomial time. In the solution, the flow on each edge from source to sink node tells how to move an alien cell. The incremental placement algorithm flow is outlined in the pseudo code below.

Algorithm 1 Incremental Placement

input : cell placement and clusters $\mathcal{C} = \{C_1, C_2, ...\}$
output : cell placement where each cluster forms contiguous region

Sweep each cell row to identify alien cells and empty space
for Each cluster $C_i \in \mathcal{C}$ **do**
1. Find alien cells belonging to C_i and usable space in/around C_i
2. Build min-cost network flow model for moving the alien cells back to C_i
3. Solve the min-cost network flow problem using the Edmonds-Karp algorithm

end for

In implementation, we need to identify alien cells, the clusters they belong to and the empty space within (or adjacent with) clusters. Since circuit designs mostly use standard cells and cells are placed in rows, the identification is done by scanning individual rows. By checking if a consecutive set of cells belong to the same cluster, one can detect potential alien cell. If a cell does not have left or right neighbors from the same cluster, the rows right above and below it are also examined to see if it has neighbor above/below that belongs to the same cluster. Also, since in min-cost problem, the algorithm will not see a cell as a whole, there is a chance a cell c_i is split into multiple pieces. To handle this problem, we simply find which space the biggest piece of c_i is assigned to, and put the whole cell into that place. It doesn't necessarily result in an overlap, but if there is one, we will shift the cells around c_i to

make more room for c_i. In this way, we make sure the placement is legal everywhere.

2.3 Gate sizing and adaptivity assignment

This is stage III in our design flow. In this stage, a circuit has been partitioned into clusters $C = \{C_1, C_2, ...\}$, one needs to decide if to assign adaptivity to each cluster C_i. By assigning adaptivity, a cluster is able to autonomously detect and compensate timing variations and therefore has robustness to variations. Adaptivity comes with area overhead due to sensor and tuning circuit. The adaptivity assignment tool we use [11] has the following inputs: circuit G composed by clusters $C = \{C_1, C_2, ...\}$, timing yield constraint Υ, and adaptivity area constraint Ω. The algorithm iteratively assigns or de-assigns adaptivity for a cluster. Depending on if timing yield $Y(G)$ evaluated by an statistical STA satisfies a given constraint Υ, each iteration may be in either timing mode or overhead mode. The iteration continues as long as there is still improvement on timing yield, area, or power dissipation. The output of the algorithm is adaptivity assignment $\Phi(C_i)$ for each $C_i \in C$.

In their approach, they arbitrarily partition a circuit by a $n * n$ framework, gates within the same tile are assigned to a cluster. What we do differently is, in our application, we import the clustering result of timing and location aware clustering algorithm to assign gates to clusters. In the next chapter, we report results of our design flow as well as comparison between results gotten using their location-aware clustering method to show the effectiveness of our method.

3. EXPERIMENTS

3.1 Experiment design and setup

The entire flow of Figure 1.3 is evaluated in the experiments. The initial placement (including detailed placement) is done using the Capo placer [7]. All the other steps in Figure 1.3 are implemented by C++ language. The timing analysis after the initial placement follows the method of PCA-based statistical static timing analysis [1]. The wirelength is evaluated according to half-perimeter of net bounding boxes. The last step gate sizing and adaptivity assignment uses the method of [11]. The timing yield of adaptive designs is estimated by the technique of [3]. All the implementations run on an AMD Opteron processor with 2.2GHz frequency, 4GB memory and Linux operating system.

The experiments are performed on ICCAD 2014 Incremental Timing-Driven Placement Contest benchmark suites [13]. Adaptive body bias is employed as platform of adaptive circuit design. Please note that our method can be applied to other types of adaptive design, such as voltage interpolation [5]. We assume canary flip-flop [8] based delay variation sensors. The control signals incur only several dozens of nets, whose wirelength is negligible in circuits with hundreds of thousands of nets. We did routing for a few cases and found that the control wirelength accounts for less than 0.1% of total wirelength. In the experiments, we only focus on the wirelength overhead arising from the clustering and incremental placement. The area overhead from adaptive circuits mostly includes sensor area and gate area increase due to triple-well process for body bias. The number of clusters is empirically chosen in a range from 10 to 25. Please note our clustering algorithm can autonomously adjust the number of clusters. The timing is estimated according to RC switch model and the Elmore

model. The power model is the same as that in [2]. Gate length variations with standard deviation of 5% nominal value are considered.

The following approaches are compared in the experiments.

- *Over-design*: This is the conventional non-adaptive circuit design. It does not have sensors, control or tuning circuits, and therefore cannot adapt to variations. It applies identical amount of power among all chips according to the worst case variation.
- *Location-aware clustering*: This is an implementation of the flow in Figure 1.3, but only spatial proximity is considered in the clustering step. As such, each cluster forms a contiguous region without the need of incremental placement.
- *Timing-aware clustering*: This is an implementation of the flow in Figure 1.3, but only timing proximity is considered in the clustering step, i.e., $\gamma = 0$ for the clustering distance defined in Equation (2.3). This implementation tries to emulate the approach of [2] in a broad sense, but in a simpler manner.
- *Ours*: This is our complete flow in Figure 1.3 based on timing and location aware clustering.

3.2 Experimental results

All methods are tested under several different timing constraints and the average results are shown in Table 3.3 and 3.4. The results in Table 3.3 are from experiments with only forward body bias (FBB). One can see that all methods achieve about the same timing yield and adaptive design can save about 26% power compared to over-design. Our method results in 26% less area overhead than the location-aware clustering method. Compared to the timing-aware clustering, our method not only reduces area overhead by 31% but also incurs much less wirelength overhead. The

average wirelength overhead from our method is only 0.6%, which is about 95% less than that from the timing-aware clustering.

Table 3.4 summarizes results from experiments with both FBB and RBB (Reverse Body Bias). The observation is similar to Table 3.3 except that area overhead reduction from our method is 78% on average compared to location-aware clustering. For circuit *mgc_matrix_mult*, the adaptive design leads to area decrease. This is because the optimization in the last step of Figure 1.3 may downsize cells.

Flow computing runtime data for different methods with FBB are outlined in Table 3.1. Even for the largest case *netcard*, which has near one million cells, our complete flow takes about 3 hours. This is much faster than the approach of [2], which spends near 4.5 hours to process a small circuit with 32K cells. The over-design flow has only the initial placement and timing analysis part of Figure 1.3. By comparing with the runtime of our complete flow, one can tell that the initial placement and timing analysis account for about 1/3 of total runtime. The location-aware clustering method does not include incremental placement. A simple calculation tells that the incremental placement causes about 1/3 of entire runtime, and the clustering plus adaptivity optimization also costs about 1/3 of total runtime.

Table 3.1: Runtime of the first set of comparison with only forward body bias (FBB)

		Baseline	Location-aware clustering	Ours
Circuit	#gates	Rutime (s)	Rutime (s)	Rutime (s)
mgc_edit_dist	130674	693	1698	2197
mgc_matrix_mult	155341	1020	1794	2318
vga_lcd	164891	609	1381	3615
b19	219268	1419	3111	3882
leon3mp	649191	2544	4369	7880
leon2	79286	3133	4953	8477
netcard	958792	3729	7363	11307
Average	438920	1878	3524	5754

Experiment is performed to investigated the impact of weight factors α, β and γ in Equation (2.3), which defines the distance for clustering. The experiment is conducted on circuit *mgc_matrix_mult* with FBB, and the result is shown in Table 3.2. In this Table, the column of AP is for adaptive power, which is the average power increase due to the tuning from zero body bias to forward body bias. The minimum area overhead ΔArea is 1653, which is significantly lower than that in Table 3.3. This is because the timing constraint for Table 3.2 is relatively loose while the area overhead in Table 3.3 is an average from multiple experiments including those with tight timing constraints. The wirelength overhead ΔWire is mostly decided by the ratio between α and γ. Not surprisingly, wirelength overhead is quite remarkable when this ratio is large. The timing yield results for these different weight factors are very similar.

Table 3.2: Impact of weight factors in clustering distance (Equation (2.3))for circuit *mgc_matrix_mult* with FBB. Adaptive power is denoted by AP.

α	β	γ	# clusters	AP	ΔArea	ΔWire
0	1	1	23	3707	4111	0%
5	1	1	24	3500	1851	0.61%
6.5	1	1	22	7764	8538	0.76%
7.2	1	1	22	9582	10585	0.82%
15	1	1	12	10376	10903	6.7%
20	1	1	12	6686	8423	8.5%
1	1	0	20	3771	2533	9.5%
5	0	1	24	3500	1851	0.61%
5	2	1	24	3377	1653	0.60%
5	4	1	24	4330	3673	0.55%

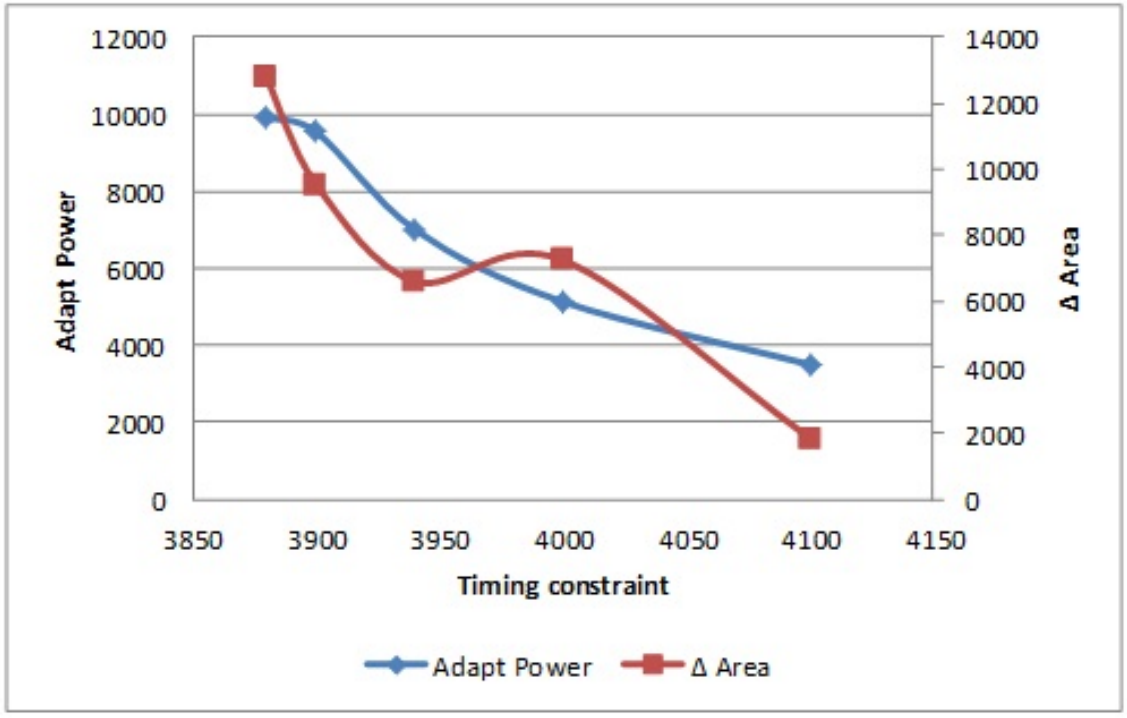

Figure 3.1: Power/area - timing trade-off for *mgc_matrix_mult* with FBB.

In another experiment, timing constraint is varied to observe the effect on power and area overhead for circuit *mgc_matrix_mult* with FBB. The power here only includes the adaptive power, which is incurred due to body bias change. The trade-off curves are depicted in Figure 3.1. It is as expected that power/area increases as timing constraint is tightened. The optimization and adaptivity tuning are carried out in a way to obtain similar timing yield.

Table 3.3: Experimental results with only forward body bias (FBB). Total area overhead, power overhead, and wire-length increase are denoted by ΔA, ΔPwr, and $\Delta Wire$, respectively.

		Over-design	Location-aware Clustering			Timing-aware Clustering				Ours				
Circuit	#gates	Yield	Power	Yield	ΔArea	Power	Yield	ΔArea	Power	ΔWire	Yield	ΔArea	Power	ΔWire
mgc_edit_dist	130674	99.9%	806862	99.0%	13200	677812	99.1%	6416	679502	8%	99.2%	5279	678047	0.0%
mgc_matrix_mult	155341	99.9%	1280460	99.4%	12534	954764	99.7%	13982	952605	8%	99.4%	7603	949890	0.6%
vga_lcd	164891	99.9%	1025100	99.1%	7048	821001	99.6%	8639	820743	11%	99.3%	4776	820007	0.6%
b19	219268	99.9%	1994560	99.2%	12793	1501780	99.1%	7343	1493885	9%	98.8%	6695	1493931	0.6%
leon3mp	649191	99.9%	5206540	98.9%	25356	3730190	99.3%	27124	3732427	14%	98.5%	21401	3727880	0.9%
leon2	794286	99.9%	6063358	98.6%	33516	4351986	99.2%	29367	4349844	13%	98.2%	25998	4347273	0.7%
netcard	958792	99.9%	7026287	99.1%	56990	5376300	99.7%	79730	5385104	14%	99.3%	46900	5369778	0.9%
Average	438920	99.9%	3343310	99.0%	23062	2487691	99.4%	24657	2487330	11%	99.0%	16950	2483829	0.6%
Normalized			1		1.36	0.744		1.45	0.744	1		1	0.743	0.05

Table 3.4: Experimental results with Adaptive Body bias (ABB). Total area overhead, power overhead, and wire-length increase are denoted by ΔA, ΔPwr, and $\Delta Wire$, respectively.

		Over-design		Location-aware Clustering			Timing-aware Clustering				Ours			
Circuit	#gates	Yield	Power	Yield	ΔArea	Power	Yield	ΔArea	Power	ΔWire	Yield	ΔArea	Power	ΔWire
mgc_edit_dist	130674	99.9%	806862	98.9%	10485	674719	98.8%	7207	674920	8%	98.9%	6004	674593	0.0%
mgc_matrix_mult	155341	99.9%	1280460	99.2%	-1532	941004	99.7%	1476	942688	8%	99.2%	-1982	940700	0.6%
vga_lcd	164891	99.9%	1025100	99.0%	24379	817091	99.5%	4674	819654	11%	99.0%	3492	816888	0.6%
b19	219268	99.9%	1994560	99.4%	30641	1476531	97.6%	7720	1480902	9%	99.4%	5319	1479105	0.6%
leon3mp	649191	99.9%	5206540	98.1%	55381	3635033	98.9%	25343	3644075	14%	98.1%	19509	3625009	0.9%
leon2	794286	99.9%	6063358	98.6%	149873	4267482	98.3%	26735	4243923	13%	98.6%	23986	4264651	0.7%
netcard	958792	99.9%	7026287	98.2%	174698	5287475	99.7%	58021	5292620	14%	99.0%	40056	5274198	0.9%
Average	438920	99.9%	3343310	98.8%	63418	2442762	98.9%	18739	2442683	11%	98.9%	13769	2435021	0.6%
Normalized			1		4.61	0.731		1.36	0.731	1		1	0.728	0.05

4. CONCLUSION

Adaptive circuit design is a well-demonstrated technique for robust VLSI systems. However, it is rarely applied in realistic products due to the lack of optimization tool support, especially the tools for managing adaptivity overhead. In this work, we propose a methodology and algorithmic techniques for clustering and incremental placement to reduce overhead of adaptive circuit design. In clustering, physical proximity is not explicitly considered before. Now though our design flow we prove that during clustering, location is equally important as timing. The clustering considers both timing and spatial proximity, and is much faster than its previous work. The incremental placement is realized by iterative min-cost network flow algorithm. Experimental results from benchmark circuits show that our approach significantly reduces area overhead while maintains the same power and timing performance. Experimental results from benchmark circuits confirm the effectiveness of our approach, our approach significantly reduces area overhead while maintains the same power and timing performance. It incurs 95% less wirelength overhead than the approach of timing-only clustering.

REFERENCES

[1] H. Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 621, 2003.

[2] S. H. Kulkarni, D. M. Sylvester, and D. T. Blaauw. Design-time optimization of post-silicon tuned circuits using adaptive body bias. *IEEE Transactions on Computer-Aided Design*, 27(3):481–494, March 2008.

[3] R. Kumar, B. Li, Y. Shen, U. Schlichtmann, and J. Hu. Timing verification for adaptive integrated circuits. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1587–1590, 2015.

[4] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Adaptive techniques for overcoming performance degradation due to aging in digital circuits. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 284–289, 2009.

[5] X. Liang, G. Y. Wei, and D. Brooks. Revival: a variation-tolerant architecture using voltage interpolation and variable latency. *IEEE Micro*, 29(1):127–138, January 2009.

[6] Q. Liu and S. S. Sapatnekar. Capturing post-silicon variations using a representative critical path. *IEEE Transactions on Computer-Aided Design*, 29(2):211–222, February 2010.

[7] J. A. Roy and I. L. Markov. ECO-system: embracing the change in placement. *IEEE Transactions on Computer-Aided Design*, 26(12):2173–2185, December 2007.

[8] T. Sato and Y. Kunitake. A simple flip-flop circuit for typical-case designs for DFM. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 539–544, 2007.

[9] J. W. Tschanz, J. T. Kao, S. G. Narendra, R. Nair, D. A. Antoniadis, A. P. 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*, 37(11):1396–1402, November 2002.

[10] C. Zhuo, D. Blaauw, and D. Sylvester. Variation-aware gate sizing and clustering for post-silicon optimized circuits. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 105–110, 2008.

[11] H. He, J. Wang, and J. Hu. Collaborative gate implementation selection and adaptivity assignment for robust combinational circuits. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 122–127, 2015.

[12] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows: theory, algorithms, and applications. Prentice Hall, Upper Saddle River, NJ, 1993.

[13] M. C. Kim, J. Hu, and N. Viswanathan. ICCAD-2014 CAD contest in incremental timing-driven placement and benchmark suite. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 361–366, 2014.