LOW COST POWER AND SUPPLY NOISE ESTIMATION AND CONTROL IN SCAN TESTING OF VLSI CIRCUITS

A Dissertation

by

ZHONGWEI JIANG

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

DOCTOR OF PHILOSOPHY

December 2010

Major Subject: Computer Engineering

LOW COST POWER AND SUPPLY NOISE ESTIMATION AND CONTROL IN SCAN TESTING OF VLSI CIRCUITS

A Dissertation

by

ZHONGWEI JIANG

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

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Duncan M. Walker
Committee Members,	Rabinarayan Mahapatra
	Vivek Sarin
	Jiang Hu
Head of Department,	Valerie E. Taylor

December 2010

Major Subject: Computer Engineering

ABSTRACT

Low Cost Power and Supply Noise Estimation and Control in

Scan Testing of VLSI Circuits. (December 2010)

Zhongwei Jiang, B.S., Nanjing University of Posts and Telecommunications, China;

M.S., Shanghai Jiao Tong University, China

Chair of Advisory Committee: Dr. Duncan M. Walker

Test power is an important issue in deep submicron semiconductor testing. Too much power supply noise and too much power dissipation can result in excessive temperature rise, both leading to overkill during delay test. Scan-based test has been widely adopted as one of the most commonly used VLSI testing method. The test power during scan testing comprises shift power and capture power. The power consumed in the shift cycle dominates the total power dissipation. It is crucial for IC manufacturing companies to achieve near constant power consumption for a given timing window in order to keep the chip under test (CUT) at a near constant temperature, to make it easy to characterize the circuit behavior and prevent delay test over kill.

To achieve constant test power, first, we built a fast and accurate power model, which can estimate the shift power without logic simulation of the circuit. We also proposed an efficient and low power X-bit Filling process, which could potentially reduce both the shift power and capture power. Then, we introduced an efficient test pattern reordering algorithm, which achieves near constant power between groups of patterns. The number of patterns in a group is determined by the thermal constant of the chip. Experimental

results show that our proposed power model has very good correlation. Our proposed X-Fill process achieved both minimum shift power and capture power. The algorithm supports multiple scan chains and can achieve constant power within different regions of the chip. The greedy test pattern reordering algorithm can reduce the power variation from 29-126% to 8-10% or even lower if we reduce the power variance threshold.

Excessive noise can significantly affect the timing performance of Deep Sub-Micron (DSM) designs and cause non-trivial additional delay. In delay test generation, test compaction and test fill techniques can produce excessive power supply noise. This can result in delay test overkill. Prior approaches to power supply noise aware delay test compaction are too costly due to many logic simulations, and are limited to static compaction.

We proposed a realistic low cost delay test compaction flow that guardbands the delay using a sequence of estimation metrics to keep the circuit under test supply noise more like functional mode. This flow has been implemented in both static compaction and dynamic compaction. We analyzed the relationship between delay and voltage drop, and the relationship between effective weighted switching activity (WSA) and voltage drop. Based on these correlations, we introduce the low cost delay test pattern compaction framework considering power supply noise. Experimental results on ISCAS89 circuits show that our low cost framework is up to ten times faster than the prior high cost framework. Simulation results also verify that the low cost model can correctly guardband every path's extra noise-induced delay. We discussed the rules to set different constraints in the levelized framework. The veto process used in the compaction can be also applied to other constraints, such as power and temperature.

DEDICATION

To my parents and my family:

without their support, this would not have been possible.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Duncan M. (Hank) Walker, for his guidance, patience and continuous support throughout my doctoral study. I would also like to thank him for guiding me in my dissertation research with such dedication and consideration, and never failing to pay attention to any details of my work. His technical insight, his novel ideas and his encouragement are all essential to this work. This dissertation would never have been accomplished without his technical and editorial advice.

I would like to extend my gratefulness to my committee: Dr. Jiang Hu, Dr. Vivek Sarin and Dr. Rabinarayan Mahapatra. They provided a lot of valuable suggestions and personal encouragement, and I learned a lot from talking to them.

Thanks to my teammates Zheng Wang, Sivakumar Ganesan, Shayak Lahiri, and Karthik Tamilarasan. I learned a lot from them in the past several years. I want to thank Jing Wang for her help and advice on my research. Another special thanks goes to Wangqi Qiu and Lei Wu for their help on the industrial project on testing a microprocessor.

My research was funded in part by Semiconductor Research Corporation (SRC) and by the National Science Foundation (NSF). I thank them for their financial support.

Finally, I want to acknowledge the love and support of my parents and my family. They were always there for me whenever I had problems, and they always shared my happiness for every progress I made. I am deeply indebted to them, more than my words can ever express.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

LIST OF TABLES

Page

Page

Page

1. INTRODUCTION

Test power is an important issue in deep submicron (DSM) semiconductor testing. Too much power supply noise and too much power dissipation can result in excessive temperature rise, both leading to overkill during delay test. Scan-based test has been widely adopted as one of the most commonly used VLSI testing methods. The test power during scan testing comprises shift power and capture power. During Launch-on-Shift (LOS) or Launch-on-Capture (LOC) test, the power consumed during the shift cycles dominates the total power dissipation, since there is a large amount of signal switching during the scan-in/out process for most scan architectures. Capture power is dissipated only during the capture cycle, and so is much smaller than the shift power. For example, if the scan chain is longer than a thousand scan cells, the shift power could be one thousand times larger than the capture power. Since the shift power is expensive to compute during the shift-in and shift-out process, we need a simple and fast model to estimate it. The power dissipation during different phases of the test process is hard to predict, but it is crucial for IC manufacturing companies to achieve near constant power consumption during a given timing window, in order to keep the chip under test (CUT) at a near constant temperature to avoid exceptional behavior or even over-kill. In addition, if the CUT has linear temperature rise, it is easy to characterize the circuit behavior during each test phase. Industry data shows that the signal delay rises 35-55%

This dissertation follows the style and format of *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

for a 100^{o}C rise in 65 nm technology. If we can predict the temperature at a given test pattern, we can adjust the capture clock timing to avoid overkill.

Prior work [1][2][3][4][5][6][7][8][9][10] proposed methods to reduce the capture power and keep the power supply noise to a low level during compaction or test generation. A static compaction technique was proposed to control scan power [11]. Test-vector ordering heuristics have been proposed, but they were only concerned with minimizing power, at a high computational cost [12]. They do not consider how to keep the test power constant.

Recently, a technique called *Preferred Fill* [13] was proposed which fills the X (don't care) bits in a test pattern by using the signal probability. Only a single pass is required to compute the signal probability for the entire circuit, and the approach achieves very good capture power reduction. Shift power can be minimized by using *Adjacent Fill*, in which X bits are filled with the adjacent 0/1 value.

Since accurately computing the shift power requires $N \cdot M$ cycles of logic simulation if M is the number of bits in a scan chain and N is the number of test patterns, it is obvious that this is not feasible for large circuits. Prior work [11] proposed using scan chain switching to estimate the shift power, but did not fully consider the structure of the circuit, which limited correlation to logic simulation results.

In order to achieve constant test power, first, we need a fast and accurate power model, which can estimate the shift power without logic simulation of the circuit. In addition, we need an efficient and low power X-bit Filling process, which can reduce both the shift power and capture power. Then, we need an efficient test pattern

reordering algorithm, which achieves near constant power between groups of patterns. The number of patterns in a group is determined by the thermal time constant of the chip. The X-Fill process that we propose combines Preferred Fill, Adjacent Fill and Random Fill to achieve both minimum shift power and capture power. The algorithm supports multiple scan chains and can achieve constant power within different regions of the chip. The greedy test pattern reordering algorithm can reduce the power variation from 29-126% to 8-10% or even lower if we reduce the power variance threshold.

The traditional test pattern compaction process achieves a high compaction rate, but does not check the supply noise of each pattern. High compaction will generate higher power patterns that may produce excessive power supply noise. The excessive switching in the circuit supply network will cause a voltage drop and consequently a delay increase on signal paths, potentially violating the timing specification. The approach in [14] proposed a static compaction technique, which controls the supply noise so that paths do not exceed their timing specification due to noise. This approach is a post-processing step based on the un-compacted patterns and the target paths corresponding to each pattern. It shows good correlation compared to circuit simulation and it was verified with silicon results [4]. The supply noise and delay estimation in [14] was based on a low cost power supply noise model and delay model. The major problem of this approach is the tremendous number of logic simulations. We enhanced this approach by proposing a levelized supply noise estimation framework, which drastically reduces the simulation time. The other drawback of [14] is that it is a post-processing step after ATPG. Dynamic compaction [15] during ATPG achieves significantly higher test pattern

compaction compared to static compaction. Dynamic compaction combines paths together based on their necessary assignments, without fault simulation. This algorithm was incorporated into the KLPG ATPG algorithm and significantly reduced pattern count without coverage loss. We have incorporated the new low cost supply noise estimation framework into dynamic compaction.

1.1 Test Power

Test power is an important issue in deep submicron semiconductor testing. Too much power supply noise and too much power dissipation can result in excessive temperature rise, both leading to overkill during delay test. In this work, we focus on power dissipation during the scan-in/out process, since this dominates total power dissipation during scan-based testing. For example, if the scan chain is longer than a thousand scan cells, the shift power could be a thousand times larger than capture power and the capture power is neglectable. The power dissipation during different phases of the testing process are hard to predict but it is crucial for IC manufacturing companies to achieve near constant power consumption in a given timing window in order to keep the chip under test (CUT) at a near constant temperature to avoid exceptional behavior or even over-kill. Also, if the CUT has linear temperature rise, it is easy to characterize the circuit behavior during each phase of the testing. We can compute the temperature at each test pattern and adjust the capture clock timing to avoid overkill. Industry data shows that the signal delay rises by 35-55% for a 100^{o}C rise, in 65nm technology.

Prior work [1]-[12] proposed methods to reduce the capture power and keep the

power supply noise at a low level during compaction or test generation. The work in [11] proposed a static compaction technique to control scan power. The work in [12] proposed test-vector ordering heuristics but only concerns about minimizing power and the computational complexity is very high. They did not consider how to keep the test power constant.

Recently, a technique called *Preferred Fill* [13] was proposed that fills the X (don't care) bits in a test pattern using signal probability, to minimize unnecessary switching activity during the launch cycle. It only needs one pass to compute the signal probabilities for the whole circuit, and achieves very good capture power reduction.

Once Preferred Fill has been used, Adjacent Fill can be used to fill the remaining X bits. In Adjacent Fill, the X bits are filled with the previous 0/1 (care bit) value loaded into the scan chain. This minimizes transitions on the scan chain outputs as it is shifted, with a corresponding reduction in circuit activity. We will these two techniques in our X-Fill process.

Since accurately computing the shift power requires N·M cycles of logic simulation where M is the number of bits in a scan chain and N is the number of test patterns, it is obvious that this is infeasible for large circuits. Prior work [11] proposed using scan chain switching to estimate the shift power, but they did not consider circuit statistics, reducing the accuracy of the estimation.

A test pattern reordering algorithm was proposed in [16] which achieves near constant test power across the chip. However, the greedy reordering algorithm has some shortcomings, such that it could fall into an infinite loop if there is an extremely high

power or low power pattern. In addition, the algorithm can only deal with single scan chain, while typical industrial circuits have many parallel scan chains.

We extended the work in [17] and improved the robustness of the reordering algorithm. We also added multiple scan chain support. The most important addition is the ability to achieve constant power within a giving region of a chip, as well as for the chip as a whole. Section 2 of this dissertation introduces an efficient test pattern compaction technique that was used to prepare test data for our algorithm. In Section 3, we used a modified version of Preferred Fill combined with Adjacent Fill in order to minimize both Capture and Shift Power. Section 4 introduces a shift power estimation heuristic that can efficiently estimate the shift power in terms of Weighted Switching Activity (WSA) without using logic simulation. We also describe the influence of the number of scan chains on the correlation between the chain power (scan chain switching) and shift power (circuit switching). Efficient greedy test-pattern re-ordering algorithms will be shown in Subsection 2.4 and Subsection 2.5 that can achieve near constant power dissipation both across chip and within region. Very good simulation results for KLPG delay test for ISCAS89 and ITC99 circuits under different power constraints are presented in Subsection 2.6. The variation in power is reduced from 29-126% to 8-10%. Our work appears to be the first to target both near-constant shift power while at the same time minimizing both shift power and capture power.

1.2 Power Supply Noise in Delay Test

Delay testing has become increasingly important due to reduced timing margins and

increased clock rates. Small delay defects can be tested using the path delay fault model [18]. However, as the semiconductor technology is scaled, designs are becoming more sensitive to various noise sources [19], such as leakage noise, crosstalk and power supply noise. Too much power supply noise can result in excessive noise-induced circuit delay increase, leading to overkill during delay test.

Several techniques have been proposed for estimating power supply noise during timing analysis [20][21]. These methods focused on supply network and circuit models to achieve reasonable accuracy. Jiang et al. [22] proposed a vector independent approach using genetic algorithms to estimate the worst-case noise-induced delay. Liou et al. [23] proposed an estimation method based on a statistical timing analysis framework.

Most prior work in testing while considering power supply noise adopts a vector-less strategy due to the high simulation cost of the power supply noise model on large circuits. Tirumurti et al. [24] proposed added power noise to a generalized fault model [25]. Pant et al. [26] proposed a vector-less approach for computing the maximum path delay under power supply fluctuations. Krstic et al. [27] used a vector-based approach to generate the maximum power supply noise on one path at a time. However, the resulting maximum noise could be considerably greater than the mission-mode worst-case noise. Moreover, the method may be in competition with other goals, such as crosstalk generation, that may have greater impact on path delay. Lee et al. [28] introduced a novel test pattern generation framework for inducing maximum crosstalk effects on delay-sensitive paths and Ma et al [29] proposed a layout-aware pattern generation for maximizing supply noise effects on critical paths. The motivation of this work is

maximizing noise, which is not consistent with our goal of achieving mission-mode noise.

Previous work [30] introduced a simplified power region model and circuit switching model. Good delay estimation results were verified by circuit simulation and measurement on ISCAS89 and industrial circuits during static test compaction. The major drawback of this approach was the large number of logic simulations required. A new dynamic compaction procedure [31] for path delay test reduced pattern count by as much as 4x over static compaction, but at the cost of producing some very high noise patterns that could result in test overkill.

Our prior work [32] demonstrated a realistic low cost delay test static compaction framework which used a levelized estimation metric to speed up the work in [30]. This approach shows up to 5x speed up over the previous work, but did not provide a practical approach to determine the different algorithm parameters. In addition, since dynamic compaction [31] has shown great advantage over static compaction, it requires us to further expand the supply noise analysis work to dynamic compaction during ATPG.

In this work, we focus on power supply noise modeling and estimation during delay test pattern compaction, for both static and dynamic compaction. We first introduce a realistic levelized low cost static compaction flow for delay test by reusing the noise and delay model in [30], and then we combined the low cost flow into dynamic compaction [31]. Experimental results on ISCAS89 circuits show that our low cost framework is up to 5x faster than the prior high cost framework [30]. Simulation results also verify that

the low cost model can correctly guardband the extra noise-induced delay of every path. Subsection 3.1 summarizes our delay model and circuit switching model, which is based on [30]. Then we analyze the relationship between delay and voltage drop, and the relationship between effective weighted switching activity (WSA) and voltage drop. Based on these correlations, we introduce the low cost delay test pattern static compaction framework considering power supply noise in Subsection 3.2. In Subsection 3.3 this framework is integrated with dynamic test compaction. Subsection 3.4 gives the rules for parameter setting that used in the compaction flow. A pseudo-functional test with power analysis is shown in Subsection 3.5. Experimental results together with further discussion are given in Subsection 3.6, and conclusions in Subsection 3.7.

2. CONSTANT POWER DISSIPATION

2.1 Compaction

The original test patterns were generated by a K-Longest Path per Gate (KLPG) delay fault ATPG tool named *CodGen* [33]. It generated launch-on-capture (LOC) robust path delay tests targeting the longest rising and falling transition path through every line in the circuit. Since it will generate one pattern for each longest path, in order to save simulation time, we must compact the patterns.

For test pattern compaction for ISCAS89 circuits, we implemented a greedy static compaction algorithm. Vectors are considered one by one in the order they are generated, and combined with the first compatible vector in the compacted vector list. For example, if we have two vectors V1=(0XX1X0XX) and V2=(X0XX100X), we check each bit of same position of the vectors and see whether the two bits are compatible. The common rule is that X is compatible with both 0 and 1; 0 is only compatible with 0; 1 is only compatible with 1. The first bit of V1/V2 is 0/X, so the compacted bit will be 0; the second bit of V1/V2 is X/0, so the compacted bit will be 0. The same process goes on after the last bit has been compacted. After the bit-checking finished, we have the final compacted vector V3=(00X1100X). If we change the first bit of V2 to 1, then V1 and V2 are not compatible because the first bit is not compatible.

Figure 1 is the flow chart of the compaction procedure in our experiment. This compaction process is brute force because it does not consider supply noise issue and we try to minimize the pattern count. The initial patterns after compaction tend to have more

bit transitions than the later patterns. We term compaction that does not consider supply noise *Force Compaction*.

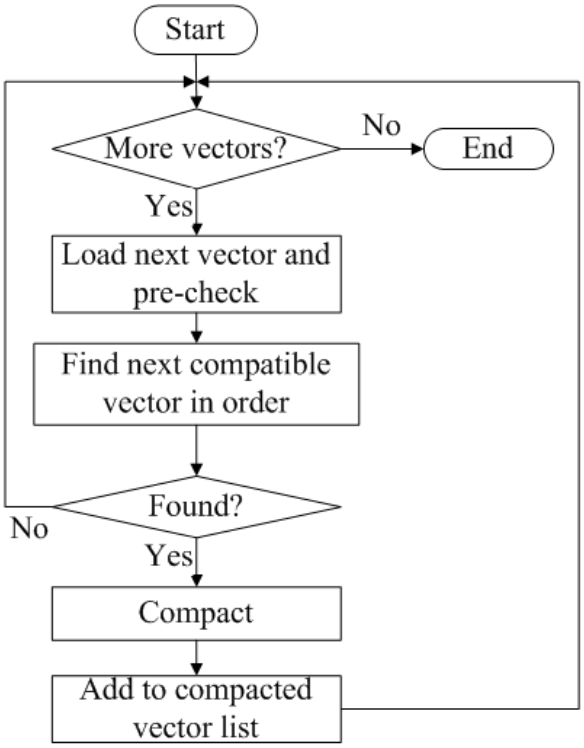

Figure 1. Static Compaction Flow

We use dynamic compaction [15] for ITC99 circuits. This compacts paths together based on their necessary assignments, without fault simulation. Rather than working on one pattern at a time, the algorithm considers a pool of paths that are currently being compacted into a set of patterns. Each new path generated is compared against this path pool. This algorithm was incorporated into the KLPG algorithm and significantly reduced pattern count without coverage loss.

The data from Table 1 show the difference of the number of compacted vectors between original *CodGen* and static compaction. We can see a tremendous reduction of patterns after compaction, especially for larger circuits such as s35932, s38417, b18 and b19. A high compaction rate minimizes test data volume and test application time. However, the compaction process may generate some extremely high power (noise) test patterns. To solve this problem, we propose an X-Fill process in the next subsection.

Table 1. Compaction Results

Circuit	# gates	# scan cells	# bits in each pattern	# Paths (Patterns) from ATPG	# Compacted Patterns
s5378	2958	179	214	1799	407
s9234	5808	211	247	2376	790
s13207	8589	638	700	3220	909
s15850	10306	534	611	2646	470
s35932	17793	1728	1763	9762	36
s38417	23815	1636	1664	14917	948
s38584	20679	1426	1464	9724	525
b15	8816	449	486	4486	1506
b17	32192	1415	1453	19165	3290
b18	114561	3320	3358	58858	5434
b19	231266	6642	6667	114688	5319
b20	20172	490	523	20351	6234
b21	20517	490	523	20443	6579
b22	29897	735	768	30489	8090

2.2 X-Fill

After compaction, we have many fewer test patterns, but more than 95% of the bits are still don't care (X) bits. In the next step, we compute the signal probability using the Preferred Fill [13] technique. The idea of Preferred Fill is to use the signal probability to set the X bits. Let the vector pair of one pattern be <V1, V2> and V1={PI1, PPI1}, V2={PI2, PPI2}. The outputs after applying V1(V2) is O1(O2) and we have O1={PO1, PPO1}, O2={PO2, PPO2}. Here PI means Primary Input, PPI means Pseudo-Primary Input, PO means Primary Output and PPO means Pseudo-Primary Output. For Launch-On-Capture (LOC) test, PPI2=PPO1.

At first, Preferred Fill will fill all the X values of PPI1. In the original Preferred Fill algorithm, a bit of the PPI1 that has a 1-probability close to 0.5 will be randomly filled, but we will use Adjacent Fill. Adjacent Fill will cause the least number of scan chain output transitions when the output of current pattern is shifting out and the next pattern is shifting in. Since the power during test is mainly the shift power, not the capture power, Adjacent Fill significantly reduces overall power dissipation. The X-Fill procedure is very fast since the signal probabilities can be computed in only one pass and filling all of the test patterns can be completed in several seconds.

Once the scan patterns are filled, we then fill the X values of PI1 and PI2. We use minimum transition fill if the bits in the same position are not both X. Then we use random fill. For example, if PI1=0XX1X11X, PI2=X0XX10XX, we fill the first bit of PI2 to 0 since the first bit of PI1 is 0, then we fill the second bit of PI1 to 0 since the second bit of PI2 is 0 and so on. After this step finished, we have PI1=00X1111X and

PI2=00X1101X. In the next step, we randomly fill the remaining X bits (but they should be the same in both PI1 and PI2). If the random values for the first X is 0 and for the second X is 1, finally we have PI1=00011111 and PI2=00011011.

The circuit response to a test pattern is crucial to our shift power estimation, since these values will be shifted out, causing switching activity. By giving V1 as input, we can compute the PPO of the circuit then assign it to the PPI part of V2, given the use of LOC test. For the X bits of PI of both V1 and V2, we first use Minimal-Transition Fill, then random fill to finish the X filling process.

Once a fully-filled vector V2 is available, PPO2 is computed using logic simulation. This step is required since the computation of shift power needs two parts: the PPO2 of the first pattern P1, and the PPI1 of the next pattern P2. The next subsection will describe in detail how to compute shift power. The pseudo code of the entire X Fill algorithm is shown below.

Algorithm **X-Fill** ()

1 Compute signal probability *prob* of all PPI1;
2 For each test pattern in the list, do
3 For each pin *p* of PPI1 which has X value
4 if (*prob* < 0.5) then *p* = 0
5 else if (*prob* > 0.5) then *p* = 1
6 else Adjacent Fill *p*
7 For each pin *p* of PI1 which has X value
8 Fill *p* according to the value of *p* in PI2
9 For each pin *p* of PI2 which has X value
10 Fill *p* according to the value of *p* in PI1

```
11  For each pin p of PI1 and PI2 which has X value
12      Randomly fill p
13  Do logic simulation to fill all X values of PPI2 by applying V1 as input
14  Do logic simulation to compute PPO2 by applying V2 as input
```

2.3 Shift Power Estimation

In this work, we use Weighted Switching Activity (WSA) to estimate the power. The WSA of a node is the number of state transitions at the driving gate multiplied by (1+fan-out of the gate). The WSA of the entire circuit is obtained by summing the WSA of all the gates in the circuit.

The capture power is a small part of the total test power, since each time a bit of the output result is shifted out and a bit of the test pattern is shifted in to the scan chain; the transitions in the scan chain will propagate to the entire circuit. It is approximately true that given a circuit of scan chain length 100, the shift power will be around 100 times the capture power. Therefore, our work will only focus on heuristics for keeping the shift power constant.

The precise calculation of shift power is straightforward. Given two consecutive test patterns: <V1, V2> and <V1', V2'>, first do logic simulation to compute the output O1 response to vector V2. The output O1 will be shifted out and at the same time vector V1' is shifted in. Each time a shift occurs, logic simulation computes the WSA of the entire circuit. We already compute O1 in the X-Fill step. But we still have to compute the circuit WSA as each bit in O1 is shifted out and each bit in te PPI1 part of V1' is shifted

in.

It is obvious that this precise calculation is not feasible for large circuits since we cannot afford to simulate $N \cdot M$ times (N is the number of patterns; M is the length of scan chain). Previous work [34] indicated that the WSA in the whole circuit is proportional to the switching in the scan chain. We improve on that prior work by considering the fan-out of each scan cell, i.e. the scan chain WSA. This increases the correlation between chain and shift power. A scan cell with higher fan-out causes more circuit switching when it transitions, and most switching happens in the first few logic levels. We use ISCAS89 and ITC99 benchmark circuits as samples and the results are listed in Table 2. Here 'Shift Power' is computed by aggregating the WSA across all scan chain shifts. The 'Chain Power' is computed by aggregating all the scan chain transitions multiplied by (1+fan-out of scan cell).

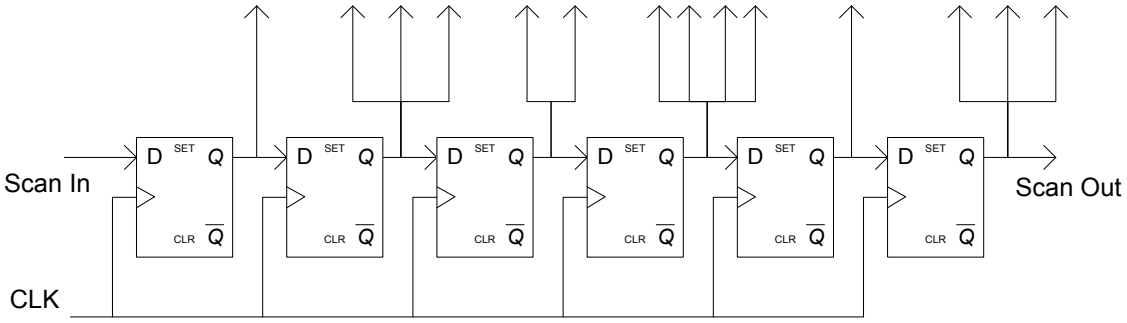

Figure 2. Scan Chain Example

For example, if there is a transition between two adjacent bits at scan cell i and the fan-out of this cell is fi, then one shift of this bit will increase the WSA in the chain by

(1+*fi*). For example, if O1=010010 and PPI1=100100, let us assume the bits are shifted from right to left and the fan-out of each scan cell is (132413) as shown in Figure 2. For simplicity, we use D flip-flops to represent the scan cells. The first 2 bits (from left to right) of O1 are (01) and there is one transition between them, so shifting out bit 2 of O1 will cause 1+1=2 WSA because it only shifted through the first cell. The second and third bits of O1 are (10), there is one transition between them, so shifting out bit 2 of O1 will cause (1+1)+(1+3)=6 WSA because the fan-out of the first and second cells are 1 and 3 respectively and we have to aggregate them when the transition shifted through the first and second cell. The computation of WSA when shifting in PPI1 is a little different from shifting O1. For example, when the transition between the first and second bit of PPI1 is shifted in, it will pass through scan cells 2,3,4,5 and 6. Then the WSA produced by it is (1+3) + (1+2) + (1+4) + (1+1) + (1+3) = 18.

The CPU time to compute the shift power for KLPG tests for circuit s38417 is nearly 3 hours, while computing the chain power takes approximately 20 seconds. More data will be shown in Subsection 2.6. Table 2 shows the correlation between Shift Power and Chain Power for ISCAS89 benchmark circuits. We simulate the Shift Power pattern by pattern using the compacted patterns in Table 1. For all listed ISCAS89 benchmarks, the correlation is above 90% and for s38417 and s38584, the correlation is close to 100%. For ITC99 circuits, the correlation is good except for circuit b18. Although b18 has lower correlation, we will still use the chain power to estimate the shift power in the experimental results in Subsection 2.6. These show that the power variance and standard deviation dropped tremendously for all of the circuits during Pattern-Reordering, which

gives some confidence in the usage of chain power to estimate shift power. Pattern-Reordering will be discussed in Subsection 2.4 and 2.5.

Table 2. Relationship Between Shift Power and Chain Power using WSA (Computed per Pattern)

Circuit	# scan chains	Ave Capture Power	Ave Shift Power (y)	Ave Scan Chain Power (x)	Equation	Correlation (R^2)
s5378	1	1589	318554	31295	y=9.012x+3.7e4	0.909
s9234	1	2009	718326	66871	y=10.26x+3.2e4	0.967
s13207	1	3398	3438625	413164	y=6.787x+6.3e4	0.980
s15850	1	2743	2392278	273651	y=7.578x+3.2e4	0.932
s35932	1	16986	15130622	2692987	y=4.020x+4e6	0.955
s38417	1	14507	16872503	1526040	y=9.254x+3e6	0.997
	2	14507	8557740	774681	y=9.262x+1e6	0.996
	4	14507	4344864	394478	y=9.312x+6.7e4	0.996
s38584	1	5498	11943562	2281070	y=4.779x+1e6	0.994
b15	1	3849	959586	144832	y=4.921x+2.5e4	0.951
b17	1	11292	9430342	1602988	y=4.879x+2e6	0.988
b18	5	28532	18298468	2913364	y=4.399x+5e6	0.530
	10	28532	9374047	1487643	y=4.597x+3e6	0.542
	20	28532	4973402	800478	y=4.333x+1e6	0.545
b19	9	55288	45711184	6707544	y=5.064x+1e7	0.816
	18	55288	22437717	3366618	y=4.634x+7e6	0.825
	27	55288	14834816	2243794	y=4.741x+4e6	0.845
b20	1	15892	5523859	254798	y=16.30x+1e6	0.927
b21	1	15831	5558590	251898	y=16.82x+1e6	0.917
b22	1	20615	11662497	589127	y=15.08x+3e6	0.925

We also conducted experiments by vary the number of scan chains to determine the influence of scan chain count on test power estimation. We only changed the number of scan chains on circuits s38417, b18 and b19 since the other benchmark circuits had too few scan cells. From Table 2 we can see that the average capture power is not related to the number of scan chains. However, the average shift power and average chain power is almost inverse proportional to the number of scan chains. The reason is that for more chains, fewer clock cycles are required to shift the test patterns in and results out.

Figure 3 shows the parallel vector bit shifting for multiple scan chains. Here shift power and chain power actually refer to energy consumption, since formally speaking, power is the energy consumed in a giving time. Our goal is to keep this nearly constant. The correlation between shift power and chain power changes little change with different number of scan chains. Many scan chains corresponds to shorter scan chains and is preferable for designs using test compression.

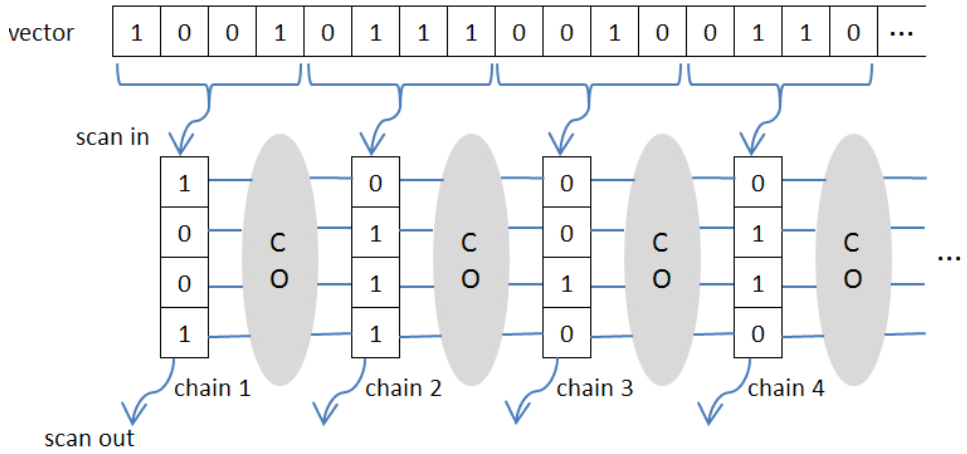

Figure 3. Parallel Vector Bit Shifting for Multiple Scan Chains

Figure 4 shows the power correlation for circuit s38417. The correlation is near 100 percent. The chain power is by far the most promising metric for us to estimate shift power and the most important thing is that the computation cost is very low compared to logic simulation.

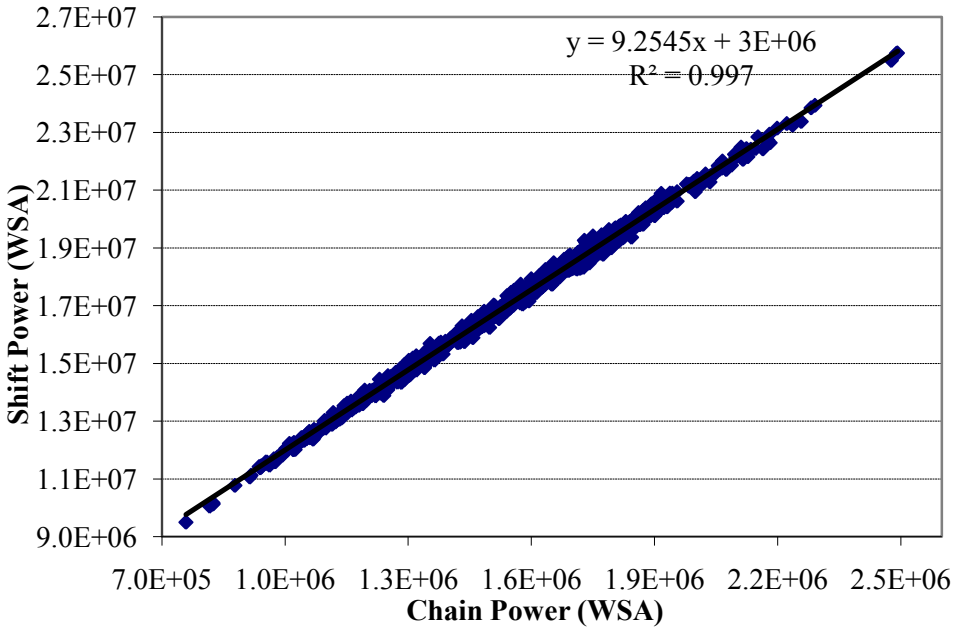

Figure 4. Power Correlation for s38417 (per pattern)

2.4 Chip-wise Test Pattern Reorder

After all vectors are filled, we will start re-ordering to achieve constant power. The test pattern application time is small compared to the chip thermal time constant. The thermal time constant is usually 1-10ms for about a 1^{o}C rise. For a 500-bit scan chain shifting at 100 MHz, the scan in/out time is only 5μs. Even if we consider 10 patterns in

a group, the 50μs application time is still less than 1 ms. Therefore, we can group patterns together and reorder these groups to achieve constant power. In our work we define the pattern group or time window as 10 patterns. The algorithm attempts to equalize the power between groups. We set a power variance bound (*pvb*) that defines the permissible power variation between each pattern group. If the power of all groups is within in the bound, we can say that the power is constant. In our experiments, we typically set *pvb* to 0.05 which means a +/-5% variation is allowed between the highest and lowest power pattern groups.

The reordering algorithm shown on the next page uses a greedy approach. It differs from the initial version in [16], because if there is an extremely high power pattern and an extremely low power pattern, we will continually swap those two patterns and never achieve close to the optimal solution. In addition, in the original algorithm, if a pattern swap cannot achieve constant power in a group, it will go on to the next group without trying to find another swap candidate. The new algorithm introduces an exclusive list and a swap-check process to solve this problem. Detailed information is given below.

The algorithm first randomly shuffles all the patterns because after compaction, the initial patterns always tend to have more power than the later patterns. Randomly shuffled patterns eliminate this bias, and so form a good starting point for the reordering. It then computes the power of each pattern *k* using the transitions in the chain, stored as *PP[k]*. Then the power of all patterns in a group *i* is stored as *PG[i]*. The average power of all groups is computed and stored as *ave*. This initialization procedure is summarized in the following pseudo code.

Chip-wise-Initialize ()

1 Random shuffle all patterns;
2 Compute Chain power *PP[k]* of each pattern *k*;
3 Group patterns according to predefined time window (10);
4 Compute power *PG[i]* of each group *i*;
5 Compute average power *ave* of all groups;
6 Set *iteration* to 0;

For each iteration of the algorithm, we start from the first group and proceed to the last group and check whether the total power of that group resides in the range (1+/-*pvb)*ave*. If it is higher than (1+*pvb*)**ave*, we pick the pattern *m* where *PP[m]* has the highest power in the group and meets the following constraints:

1. *PP[m]* is higher than the average power of all patterns, which is *ave*/10 in our experiment;
2. *PP[m]* should not be in the *exclude list*.

For each group *i* during one iteration, we maintain an *exclude list* that contains all patterns in group *i* that cannot find a pattern in another group to swap with. This list will be initialized each time we start swapping patterns for a new group. Then we tried to find another group *j* where *PG[j]* is lowest among all other groups. We will pick the lowest power pattern *t* in group *j* as a candidate to swap with pattern *m*. This approach could make the power more even between groups, since it makes more attempts, compared to the one attempt in [17].

The change of power induced by swapping pattern *m* and *t* is calculated as: *change = PP[m]-PP[t]*.

The difference of power of group *i PG[i]* and *ave* is calculated as *diff1=PG[i]-ave.*

The difference of power of group *j PG[j]* and *ave* is calculated as *diff2= ave- PG[j]*.

It is obvious that *diff1* and *diff2* are positive values according to our selection criteria. We then will check that swapping of *m* and *t* does not fall into the six illegal cases given below. If after checking all the patterns in group *j*, we still cannot find a legal pattern *t* to swap with *m*, we put pattern *m* into the *exclude list* which means that we can't find a pattern in group *j* to swap with it. The reason we do the following check is to ensure that the swapping will not make the original power of group *i* and *j* worse. This can happen if *change* is very high. The approach in [17] does not perform this checking and will increase the power variation for the following cases.

Case 1: If *change* > 2**diff1* which means *PG[i]* deteriorated because *diff1'=change-diff1* is larger than *diff1*. If *change* > 2**diff2* and *diff2>pvb*ave* as Figure 5 shows, *PG[j]* also deteriorated because *diff2'=change-diff2* is larger than *diff2*. We reject this swap.

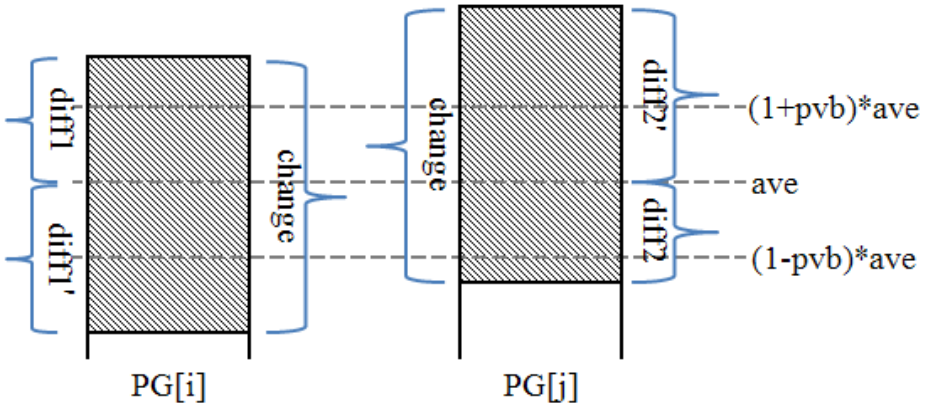

Figure 5. Case 1 of Swap-Check

Case 2: If *change* > 2**diff1* which means *PG[i]* deteriorated. If *change*<2**diff2* and *diff2>pvb*ave* as Figure 6 shows, *diff2'= change-diff2*. Since *diff2'* is less than *diff2*, the improvement of *PG[j]* would be *Im[j] = diff2-diff2' = 2*diff2 - change*. The deterioration of *PG[i]* is *De[i] = diff1'-diff1 = change-2*diff1*. If *Im[j]<De[i],* we reject this swap.

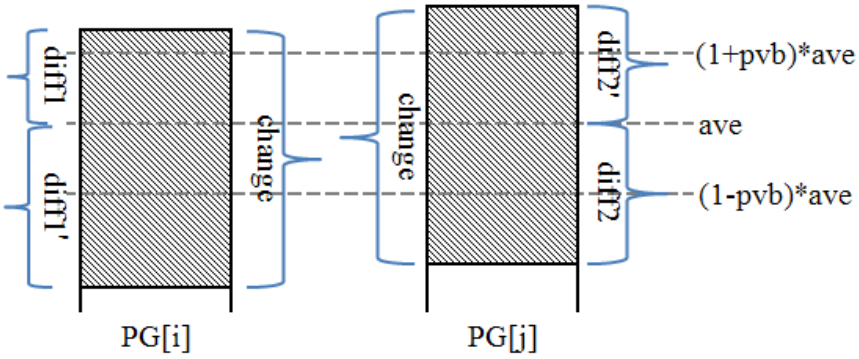

Figure 6. Case 2 of Swap-Check

Case 3: If *change* > 2**diff1* which means *PG[i]* deteriorated. If *change*>2**diff2* and *diff2<=pvb*ave* as Figure 7 shows, *PG[j]* also deteriorated. We reject this swap.

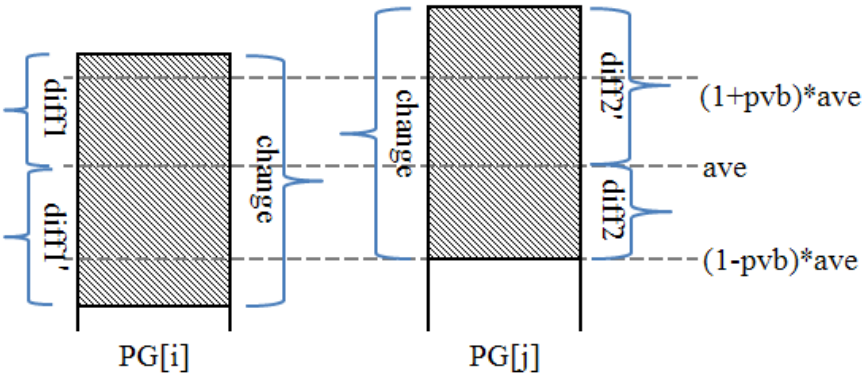

Figure 7. Case 3 of Swap-Check

Case 4: If *change* < 2**diff1* which means *PG[i]* improved. If *change-diff1-pvb*ave>0,* which means *PG[i]* becomes less than *(1-pvb)*ave* after swap. *Im[i] = diff1-diff1' = 2*diff1 -change*. If *diff2>pvb*ave* as Figure 8 shows, *PG[j]* deteriorated, *De[j] = diff2'-diff2 = change-2*diff2*. If *Im[i] < De[j],* we reject this swap.

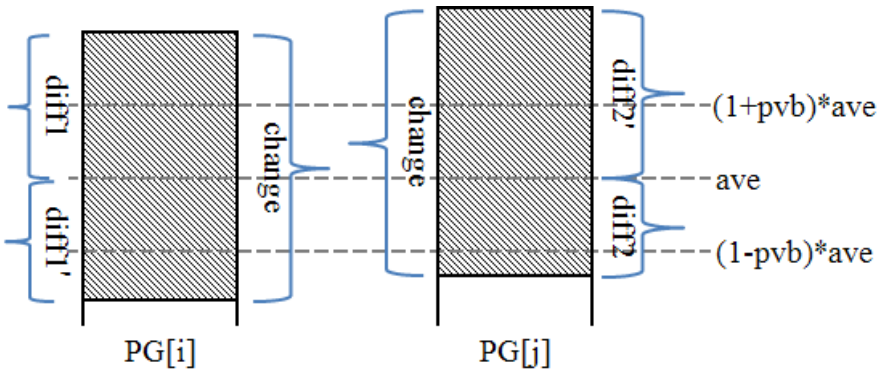

Figure 8. Case 4 of Swap-Check

Case 5: If *change* < 2**diff1* which means *PG[i]* improved. If *change-diff1-pvb*ave>0,* which means *PG[i]* become less than *(1-pvb)*ave* after swap. *Im[i] = diff1-diff1' = 2*diff1- change*. If *diff2<=pvb*ave* as Figure 9 shows, *PG[j]* deteriorated, *De[j]* is obviously larger than *Im[i]* so we reject this swap.

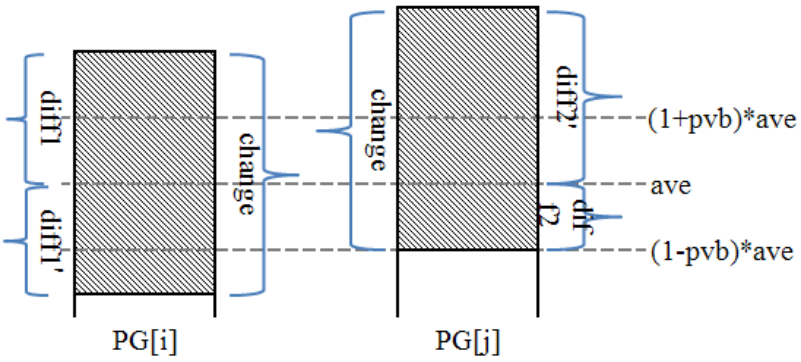

Figure 9. Case 5 of Swap-Check

Case 6: If *change-diff1-pvb*ave<=0,* which means *PG[i]* improved. If *change-diff1>diff2,* which means that *De[j]* is larger than *Im[i]* as Figure 10 shows, we reject this swap.

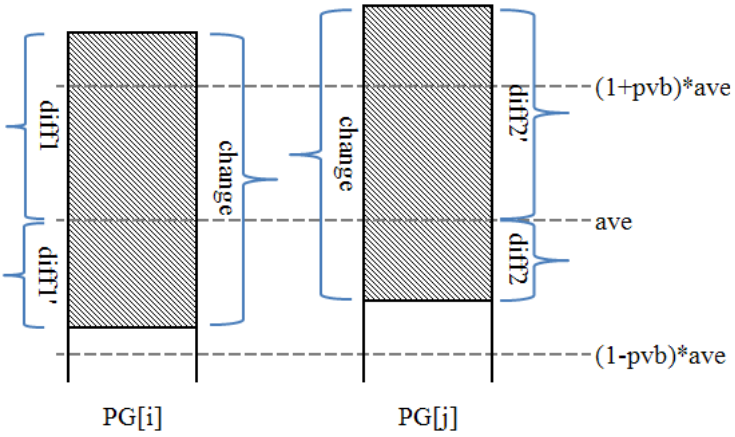

Figure 10. Case 6 of Swap-Check

The pseudo code of *Swap-Check()* listed below checks all of the six rules and if the swap does not violate any of them, the function returns false. If any rule is violated, the function returns true. If pattern *t* passes the rule checking by calling *Swap-Check()*, we proceed to swap with pattern *m*, and re-compute the chain power for pattern *m-1, m, t-1,*

t since the shift power computation of vector *i* is dependent on the next vector to be shifted in. For example, computing shifting power for vector *m-1* needs the PPI1 of pattern *m* because we are shifting out the PPO2 of pattern *m-1* and shifting in the *PPI1* of pattern *m*. Then we update the total power of the affected group and re-compute the average group power *ave,* because the power of the affected groups has changed.

```
boolean Swap-Check (diff1, diff2, change)
1   if (change - 2*diff1 > 0) {
2      if (diff2 > pvb*ave ) {
3         if (change >= 2*diff2)
4            return true;  //case 1
5         else if (change - diff2 - pvb*ave >= 0)
6            if (2*diff2 - change < change - 2*diff1)
7               return true; //case 2
8      }
9      else  return true;  //case 3
10  }
11  else if (change - diff1 - pvb*ave > 0) {
12     if (diff2 > pvb*ave) {
13        if (2*diff1 - change < change – 2*diff2)
14           return true;  //case 4
15     }
16     else  return true;  // case 5
17  }
18  else if (change - diff1 > diff2)
19     return true; //case 6
20  return false; //default
```

Similarly, if *PG[i]* is lower than (1-*pvb*)**ave*, we follow steps similar to when *PG[i]* is higher than (1+*pvb*)**ave*, making sure to select the lowest power pattern *m* in group *i* and that power *PP[m]* is lower than *ave/10*; Also, find group *j (j≠i)* where *PG[j]* is the highest among all groups; find pattern *t* which *PP[t]* is the highest in group *j* and *PP[t]* is more than *PP[m]*.

This process will stop when constant power is achieved or the total number of iterations exceeds a pre-defined *timeout* value. The following are the pseudo codes of the Pattern-Reorder algorithm and the sub-routine to check the legality of swapping two patterns which is called *Swap-Check()*. Note that a variable called *attempts* is used during swapping for each group *i*. It is set to 5 (= half the group size) which is the number of attempts to select and swap patterns in the group. The reason why we introduced this loop variable is that we try to even out the group power *PG[i]* as best as we can during each iteration. Experiments showed good results after we added this variable.

The pseudo code of chip-wise pattern reordering algorithm is summarized as follows.

```
Algorithm Chip-wise-Pattern-Reorder ()
1  Chip-wise-Initialize();
2  while iteration < timeout and power is not constant, do{
3     Increment iteration by 1;
4     Initialize the exclude list;
5     For each group i, do{
6  start:
7        if PG[i] >  (1+pvb)*ave {
8           Set attempts = 0;
```

```
9       while (true) {
10        if PG[i] < (1+pvb)*ave,
11          break;  // PG[i ] is constant
12        if (attempts < 5){//try 5 swaps to even PG[i]
13          Increment attempts by 1;
14          Set diff1= PG[i]-ave;
15          Select the highest power pattern m in group i which is not in
16            exclude list and power PP[m] is higher than ave/10;
17          if m is not found
18            break;
19          Find group j (j≠i) which PG[j]is the lowest among all groups;
20          Set t = first pattern in group j;
21          Set swapped = false; // a flag to mark if pattern t found
22          For each pattern n in group j, do {
23            // Find pattern t which PP[t] is the lowest in
24            // group j and PP[t] is less than PP[m].
25            Set change = PP[m]- PP[n];
26            If change <= 0 || PP[n] >= PP[t], continue;
27            Set diff2= ave - PG[j];
28            if Swap-Check (diff1, diff2, change);
29              continue; //swap illegal
30            else {
31              set t = n;
32              set swapped = true; // pattern t found
33            }
34          } //end for
35          if (swapped = false){
36            //can't find pattern in group j to swap with pattern m
37            Put pattern m into exclude list and goto start;
```

```
38              }else {
39                  //swap pattern m and t
40                  Re-compute Chain power for pattern m-1, m, t-1, t;
41                  Re-compute power for group i, j;
42                  Update ave;
43              }
44            }// end if
45        }//end while
46      }//end if
47      else if PG[i] < (1-pvb)*ave{
48        //follow the similar steps as above, make sure to pick up
49        //the lowest power pattern m in group i and power
50        //PP[m] is lower than ave/10; find group j (j≠i) which
51        //PG[j]is the highest in all groups; find pattern t  which
52        //PP[t] is the highest in group j and PP[t] > PP[m].
53      }//end if
54    }//end for
55 }//end while
```

2.5 Region-wise Test Pattern Reorder

For large circuits, we found some regions of circuits that always had more switching than other regions, even if the total power is constant. We call those regions ‘hot spots’. In test mode, we want to keep the power dissipation in each region constant in addition to keeping the total power constant. This is obviously a harder problem because we have to know the layout information of the circuit and want to keep the power in each region

to be constant. Intuitively, if we have reordered patterns that can achieve chip-wise constant power, we cannot guarantee that this pattern order can achieve region-wise constant power. On the contrary, if we have region-wise constant power patterns, we are sure the chip-wise power is constant because of the following proof.

Assume we have *n* regions and each region has constant power (within +/-*pvb*). Assume we have *m* pattern groups. Suppose the power of group *g* in region *r* is *PG[r][g]* and we have $1 \le r \le n$, $1 \le g \le m$. Then the power of group *g* in for the whole chip is the sum of the power in all regions. We have the following two equations:

$$\sum_{1 \le r \le n} PG[r][g] \le \max_{1 \le r \le n}(PG[r][g]) \times n \le (1 + pvb) \times ave$$

$$(1 - pvb) \times ave \le \min_{1 \le r \le n}(PG[r][g]) \times n \le \sum_{1 \le r \le n} PG[r][g]$$

Since we already have region-wise constant power, which means that the max and min power of each region is within the +/- *pvb* range, the chip-wise power should also be constant. Here we only focused on evening out the pattern-to-pattern power variation within each region, not the power between regions, since some regions will inherently have more switching activity than others.

The algorithm *Region-wise-Pattern-Reorder()* is similar like *Chip-wise-Pattern-Reorder()*. *Region-wise-Initialize()* is called first to initialize the power for each region of each group. *Chip-wise-Pattern-Reorder()* is called instead of random shuffle in *Chip-wise-Initialize()* because we think starting from the patterns that achieves chip-wise constant power is a good starting point of our region-wise algorithm. Obviously region-wise reordering is more costly than chip-wise reordering. Two-dimensional arrays

PP[r][k] and *PG[r][i]* are used to store the region-wise group power per pattern and per group. We also need to store the average power *ave[r]* for each region *r*.

Region-wise-Initialize ()

1 **Chip-wise-Pattern-Reorder** ();
2 Compute Chain power *PP[r][k]* of each pattern *k* in each region *r*;
3 Group patterns according to predefined time window (10);
4 Compute power *PG[r][i]* of each group *i* in each region *r*;
5 Compute average power *ave[r]* of all groups in each region *r*;
6 Set *iteration* to 0;

Then, we call *FindRegion()* to even out the power of the region that has the most variance from the average power of that region, then switch to the next region until the power for this group is even among all regions. If we cannot find a pattern to swap, we go to next group. Here the array *var[r]* is computed by subtracting the group power *PG[r][i]* by *ave[r]* where *r* is the region ID and *i* is the group ID. A simple sort is used here to find the largest *var[r]* by its absolute value. The region which has the largest absolute value of *var[r]* is returned as our target region.

FindRegion (*i*)

1 //For group i, compute the power difference of each region from the average
2 for each region *r*, do
3 *var[r] = PG[r][i]-ave[r]*;
4 Sort *var[r]* decreasingly by it's absolute value;
5 Return the first region *r* in the *var* list;

We added a function called *Swap-Check-Region()* to check if a swap between pattern *m* and *n* for evening the power of region *i* does not deteriorate the power variation for

any regions other than group *i*. The process shown below is similar to *Swap-Check()* in chip-wise reordering. First, we check the power change after swapping pattern *m* and *n* and save it as variable *diff*. Then we check whether the region power variance *var[j]* (which is computed in function *FindRegion()*) is above the *pvb*ave[j]*. If yes, this means region *j* is a high power region with *diff* less than zero. This indicates that the swap will make the power variation in region *j* higher, so we reject this swap. For the case that *var[j]* is less than negative *pvb*ave[j]*, which means region *j* is a lower power region and *diff* is larger than zero. This indicates that the swap will make the power for region *j* lower, so we also reject this move.

```
boolean Swap-Check-Region (i,m,n)
1   for any other region j other than i, do {
2      Set diff = PP[j][m]- PP[j][n];
3      if (var[j] > pvb*ave[j]&& diff <0)
4         return true;
5      if (var[j] < -pvb*ave[j]&& diff >0)
6         return true;
7   }
8   return false; //default
```

It is critical to mention that in line 6 of algorithm *Region-wise-Pattern-Reorder()*, we will check the power variance *n* times (*n* is the number of regions). And in line 7, we call *FindRegion()* to even the power of the maximum power variance region. Each time we find a pattern to swap, we need to make sure the 6 rules defined in Subsection 2.5 are followed by calling *Swap-Check()* in line 30. The value to be passed in to the function is

the region-wise power variance between *ave[r]* and *PG[r][j]*, not the chip-wise power variance between *ave* and *PG[j]*. The iterations will end when the power is constant or a pre-defined timeout occurs.

```
Algorithm Region-wise-Pattern-Reorder ()
1  Region-wise-Initialize();
2  while iteration < timeout and power is not constant, do{
3     Increment iteration by 1;
4     Initialize the exclude list;
5     For each group i, do{
6        For each region r, do {
7           r = FindRegion(i); //Find target region to even;
8  start:
9           if PG[r][i] >  (1+pvb)*ave[r] {
10             Set attempts = 0;
11             while (true) {
12                if PG[r][i] <  (1+pvb)*ave[r]
13                   break;  // PG[r][i] is constant
14                if (attempts < 5){//try 5 swaps to even PG[r][i]
15                   Increment attempts by 1;
16                   Set diff1= PG[r][i]-ave[r];
17                   Select the highest power pattern m in group i which is not in
18                      exclude list and power PP[r][m] is higher than ave/10;
19                   if m is not found
20                      break;
21                      Find group j (j≠i) that PG[r][j] is the lowest among all groups;
22                   Set t = first pattern in group j;
23                   Set swapped = false; // a flag to mark if pattern t found
```

```
24              For each pattern n in group j, do {
25                // Find pattern t which PP[r][t] is the lowest in
26                // group j and PP[r][t] is less than PP[r][m].
27                Set change = PP[r][m]- PP[r][n];
28                if change <= 0 || PP[r][n] >= PP[r][t], continue;
29                Set diff2= ave[r] – PG[r][j];
30                if (Swap-Check (diff1, diff2, change)
31                      || Swap-Check-Region(r, m, n) )  continue;
32                else {
33                   set t = n;
34                   set swapped = true; // pattern t found
35                }
36              }// end for
37              if (swapped = false){
38                //can't find pattern in group j to swap with pattern m
39                Put pattern m into exclusive list and goto start;
40              }else {
41                 //swap pattern m and t
42                 Re-compute Chain power for pattern m-1, m, t-1, t;
43                 Re-compute power for group i, j;
44                 Update ave[r];
45              }
46            }// end if
47          }//end while
48        }//end if
49        else if PG[r][i] < (1-pvb)*ave[r]{
50          //follow the similar steps as above, make sure to pick up
51          //the lowest power pattern m in group i and power
52          //PP[r][m] is lower than ave[r]/10; find group j (j≠i) which
```

```
53            //PG[r][j] is the highest in all groups; find pattern t  which
54            //PP[r][t] is the highest in group j and PP[r][t] > PP[r][m].
55          }//end if
56        }//end for
57    }//end for
58 }//end while
```

2.6 Experimental Results

The algorithm was implemented by C++ and run on a Windows XP PC with Intel Core 2 Duo processor (2.66GHz) and 4GB memory. Figure 11 is the complete flow chart of the procedures discussed above. It starts from reading the netlist and uncompacted test patterns, then compacting patterns and filling X bits using the algorithm in Subsection 2.2, then reordering the patterns by using the algorithm in Subsection 2.4 and 2.5. If we reorder patterns for region-wise constant power, we need to read in the layout information that describes cell placement. The reordering algorithm is independent from the X-Fill algorithm and compaction algorithm used to generate the patterns. Thus, it can be used on other test patterns, such as transition fault patterns.

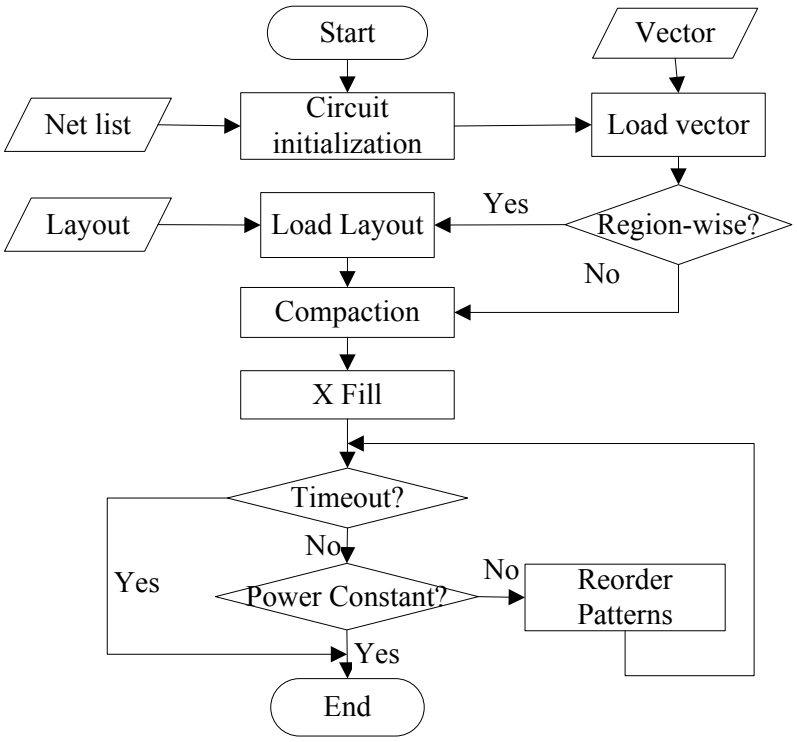

Figure 11. Constant Power Flow

As we can see from Table 3 and Table 4, our algorithm greatly reduces the power variation by reducing the Max power and increasing the Min power. Column "Initial Chain Power" is the power computed before re-ordering and column "Final Chain Power" is the power after reordering. The column 'Reorder Time' is the CPU time for reordering patterns, which requires only 1 second to reorder more than 500 patterns for circuit s38584. For the other small circuits, the total time is rounded up to 1 second. The reordered patterns reduce the overall Max/Ave (Min/Ave) from 176.18% (50.61%) to 104.50% (95.45%) and the Standard Deviation/Ave dropped from 20.66% to 2.64% for s38584. The variance between Max/Min dropped from around 126% to 9%. Note that after compaction, for circuit s35932, we only have 36 patterns which could only been

Table 3. Estimation Results for Chip-wise Constant Power Algorithm (Part 1)

Circuit	# Patterns (# Groups)	# scan chains	Initial Chain Power (before Reorder)		
			Ave(WSA)	(Max-Min)/Ave	Stdev/Ave
s5378	407(40)	1	312948	28.84%	8.53%
s9234	790(79)	1	668711	46.48%	12.02%
s13207	909(90)	1	4131639	29.35%	4.95%
s15850	470(47)	1	2736509	73.30%	16.84%
s35932	36(3)	1	26929867	93.74%	49.03%
s38417	948(94)	1	15260402	82.08%	17.49%
		2	7746811	78.33%	16.98%
		4	3944785	77.12%	16.46%
s38584	525(52)	1	22810696	125.57%	20.66%
b15	1506(150)	1	1448320	107.17%	15.49%
b17	3290(329)	1	16029879	56.01%	11.22%
b18	5434(543)	5	29133638	44.79%	9.56%
		10	14876425	46.08%	9.81%
		20	8004780	44.32%	9.94%
b19	5319(531)	9	67075436	41.65%	8.79%
		18	33666184	42.28%	9.20%
		27	22437942	44.90%	10.18%
b20	6234(623)	1	2547979	78.23%	14.28%
b21	6579(657)	1	2518977	70.77%	13.24%
b22	8090(809)	1	5891267	56.43%	11.36%

Table 4. Estimation Results for Chip-wise Constant Power Algorithm (Part 2)

Circuit	# scan chains	Final Chain Power (after Reorder)			Iterations	Total Time (m:s)	Reorder Time (m:s)
		Ave(WSA)	(Max-	Stdev/Ave			
s5378	1	312471	8.04%	2.61%	1	00:03	00:01
s9234	1	669484	9.91%	2.86%	1	00:07	00:01
s13207	1	4135834	8.69%	2.03%	1	00:15	00:02
s15850	1	2733073	9.75%	2.79%	1	00:07	00:01
s35932	1	24638404	5.40%	2.83%	1	00:02	00:01
s38417	1	15225594	9.44%	2.85%	1	00:48	00:11
	2	7727072	9.86%	2.76%	1	00:52	00:13
	4	3932966	9.00%	2.78%	1	00:55	00:15
s38584	1	22727131	9.06%	2.64%	2	00:23	00:06
b15	1	1450203	9.64%	3.01%	3	00:14	00:03
b17	1	16025113	9.77%	2.76%	1	03:30	01:17
b18	5	29128507	9.21%	2.47%	1	19:48	07:19
	10	14874256	9.87%	2.63%	1	20:08	07:18
	20	8003301	9.87%	2.76%	2	20:00	07:13
b19	9	67058303	9.63%	2.52%	1	39:57	15:31
	18	33660060	9.45%	2.60%	1	39:47	15:12
	27	22432003	9.95%	2.76%	1	39:59	15:22
b20	1	2547109	9.95%	2.59%	2	03:18	01:19
b21	1	2518064	9.91%	2.59%	1	03:46	01:28
b22	1	5888644	9.74%	2.52%	2	07:29	02:58

assembled to 3 pattern groups. The high compaction rate of our static compaction process could potentially produce extremely high power patterns and very low power patterns in a group. This is why the routine *Swap-Check* was introduced in our reordering algorithm.

We also conducted experiments by changing the number of scan chains for s38417, b18 and b19. The improvement in Max-Min variance and Standard Variation are almost independent of the number of scan chains.

The number of pattern groups is computed by dividing the pattern number by 10 and truncating the remainder because the remainder patterns would not be able to fill a full time window. This is not essential to the algorithm, since it computers per-pattern statistics for each group, and so can handle groups with different pattern counts. We do not calculate the shift-in power for the first pattern because initially the chain is preset to all 0's or all 1's, which would have very low shift power. Our time window starts from the shift in of the second test pattern.

In order to show the correctness of our power estimation, we do logic simulation to compute the Total Shift Power for each circuit to see whether the reordered patterns achieve constant shift power. Here we do logic simulation each time we shift in/out a bit from the scan chain. Table 5 and Table 6 show the corresponding Shift Power compared to the Chain Power in Table 3 and Table 4. The time cost to compute shift power is so high that for the largest ITC99 circuit b19 with 9 scan chains, it cost more than 144 CPU hours to compute the initial shift power, and then this cost is repeated to compute the final shift power. Note that this is performed only as an evaluation of the final results,

Table 5. Simulation Results for Chip-wise Constant Power Algorithm (Part 1)

Circuit	# Patterns (# Groups)	# scan chains	Initial Shift Power (before Reorder)			
			Ave(WSA)	(Max-Min)/Ave	Stdev/Ave	Time (h:m:s)
s5378	407(40)	1	3185536	31.06%	7.94%	0:00:25
s9234	790(79)	1	7183262	47.48%	12.13%	0:01:43
s13207	909(90)	1	34386247	26.17%	4.17%	0:10:10
s15850	470(47)	1	23922777	60.93%	14.02%	0:04:57
s35932	36(3)	1	151306220	74.42%	39.10%	0:03:06
s38417	948(94)	1	168725032	68.88%	14.72%	2:43:39
		2	85577398	65.91%	14.33%	1:39:23
		4	43448643	65.10%	14.02%	58:01
s38584	525(52)	1	119435620	112.80%	18.70%	1:15:19
b15	1506(150)	1	9595860	77.16%	11.25%	0:16:28
b17	3290(329)	1	94303420	45.60%	9.29%	20:14:54
b18	5434(543)	5	182984680	36.24%	7.13%	41:42:05
		10	93740465	36.76%	7.01%	26:05:14
		20	49734021	39.35%	7.26%	18:02:10
b19	5319(531)	9	457111842	35.45%	6.67%	144:53:48
		18	224499510	35.39%	6.76%	85:52:41
		27	148348164	39.14%	7.64%	57:45:39
b20	6234(623)	1	55238589	62.75%	10.99%	7:41:22
b21	6579(657)	1	55585899	49.99%	10.48%	8:12:53
b22	8090(809)	1	116624969	45.47%	8.88%	24:16:33

Table 6. Simulation Results for Chip-wise Constant Power Algorithm (Part 2)

Circuit	# scan chains	Final Shift Power (after Reorder)			
		Ave(WSA)	(Max-Min)/Ave	Stdev/Ave	Time
s5378	1	3183594	9.89%	2.48%	0:00:26
s9234	1	7187976	10.53%	2.78%	0:01:43
s13207	1	34412756	7.60%	1.69%	0:09:37
s15850	1	23908167	13.25%	2.85%	0:04:48
s35932	1	135100344	15.07%	7.70%	0:02:50
s38417	1	168457613	8.48%	2.38%	2:42:28
	2	85422031	8.54%	2.29%	1:38:01
	4	43343391	7.77%	2.35%	0:57:41
s38584	1	119184448	9.06%	2.51%	1:13:05
b15	1	9606710	10.26%	2.55%	0:16:34
b17	1	94281909	9.51%	2.32%	20:39:38
b18	5	182933251	15.04%	2.46%	42:20:29
	10	93742759	14.20%	2.47%	25:50:07
	20	49635595	14.48%	2.56%	17:51:14
b19	9	457141343	10.75%	2.08%	144:26:14
	18	224583866	9.92%	2.04%	85:15:59
	27	148396215	10.77%	2.27%	58:06:35
b20	1	55232269	12.14%	2.31%	7:52:33
b21	1	55594752	13.10%	2.44%	8:22:05
b22	1	116636580	12.55%	2.19%	24:16:09

rather than during the reordering. If we reorder patterns using full logic simulation, the execution time would be infeasible. Our estimation algorithm requires only 40 minutes and the results correlate well to logic simulation. For circuit s38417 with one scan chain, the estimation time is only 48 seconds compared to more than 160 minutes for simulation (will need twice that time to compute both initial and final power). For other circuits listed, our estimation also performs very well, at much lower CPU cost.

Our proposed greedy reordering algorithm also shows close correlation between Shift Power and Chain Power. For circuit s38417 with 1 scan chain, Table 3 and Table 4 show the estimated results that the (Max-Min)/Ave is 9.44% and Stdev/Ave is 2.85% after reordering. Using simulation, from Table 5 and Table 6 we can see that the (Max-Min)/Ave is 8.48% which is within the +/-5% bound and Stdev/Ave is 2.38%. For circuit b17 with 1 scan chain, the estimated results show that the (Max-Min)/Ave is 9.77% and Stdev/Ave is 2.76% after reordering. The simulation results show that the (Max-Min)/Ave is 9.51%, which is also within the +/-5% bound and Stdev/Ave is 2.32% after reordering.

We also executed experiments using different values of the power variation bound (*pvb*) for the larger circuits s38417, s38584, b17, b18, b19, b21 and b22. The results are summarized in Table 7 and Table 8. First, we can see that even if we reduce the *pvb* to 1% for most circuits, our algorithm still can still reorder patterns in a short time. Since the number of scan chains has little impact on the simulation results, we use 1 scan chain for the smaller circuits, 10 chains for b18 and 18 chains for b19, in order to reduce simulation time. The column ‘Total Time’ consists of the time to read in scan chain,

Table 7. Estimation and Simulation Results for Different Power Variance Bound (pvb) in Chip-wise Constant Power Algorithm (Part 1)

Circuit	# scan chains	pvb	Total Time (m:s)	Reorder Time (m:s)	Reorder Iterations	Final Chain Power		
						Ave(WSA)	(Max-Min)/Ave	Stdev/Ave
s38417	1	3%	00:49	00:12	2	15218815	5.73%	1.70%
		2%	00:50	00:13	3	15216761	3.66%	0.98%
		1%	00:51	00:14	14	15210029	1.74%	0.50%
s38584	1	3%	00:24	00:06	4	22735206	5.37%	1.60%
		2%	00:25	00:07	8	22729259	3.54%	1.13%
		1%	00:26	00:08	13	22742189	1.93%	0.62%
b17	1	2%	03:56	01:20	14	16029610	3.70%	1.05%
		1%	04:12	01:35	20	16026938	1.98%	0.57%
b21	1	2%	03:36	01:13	19	2518275	3.87%	1.13%
		1%	03:51	01:28	21	2518120	1.99%	0.60%
b22	1	2%	07:42	02:54	7	5888778	3.96%	1.11%
		1%	07:49	03:02	14	5888084	1.99%	0.59%
b18	10	3%	20:08	07:18	2	14874277	5.93%	1.72%
b19	18	3%	39:49	15:14	1	33661516	5.95%	1.65%

Table 8. Estimation and Simulation Results for Different Power Variance Bound (pvb) in Chip-wise Constant Power Algorithm (Part 2)

Circuit	# scan chains	pvb	Final Shift Power		
			Ave(WSA)	(Max-Min)/Ave	Stdev/Ave
s38417	1	3%	168394083	5.53%	1.47%
		2%	168371311	3.66%	0.88%
		1%	168317122	2.12%	0.50%
s38584	1	3%	119249912	6.22%	1.56%
		2%	119217402	5.17%	1.27%
		1%	119271968	3.42%	0.79%
b17	1	2%	94306017	4.34%	0.97%
		1%	94291941	3.73%	0.62%
b21	1	2%	55606503	9.77%	1.68%
		1%	55603244	8.89%	1.55%
b22	1	2%	116636381	8.86%	1.42%
		1%	116628884	7.90%	1.25%
b18	10	3%	93748252	13.77%	2.21%
b19	18	3%	224592057	7.05%	1.39%

netlist and un-ordered test patterns, the time to reorder patterns and the time to output reordered patterns. If we look at the 'Final Shift Power' column, we can see that after computing the shift power by simulation, the correlation between Chain Power and Shift Power is very good even when the *pvb* is 1%. For example, for s38417, when *pvb*=1%, the '(Max-Min)/Ave' and 'Stdev/Ave' of 'Final Shift Power' is 2.12% and 0.5% respectively which is very close to 1.74% and 0.5%. Keep in mind that the actual

variation experienced by the chip will be even smaller, since the pattern group application time is much less than the chip thermal time constant.

When we reduce the value of *pvb*, the reorder time and reorder iterations increased accordingly. For example, we need only 2 iterations to even out the power for s38417 when *pvb* is set to 3%, but we need up to 14 iterations when *pvb* is 1%. The reorder time also increased from 12 to 14 seconds. Note that the number of pattern swaps in each iteration is not equal, so that the number of iterations is not linear to the reorder time.

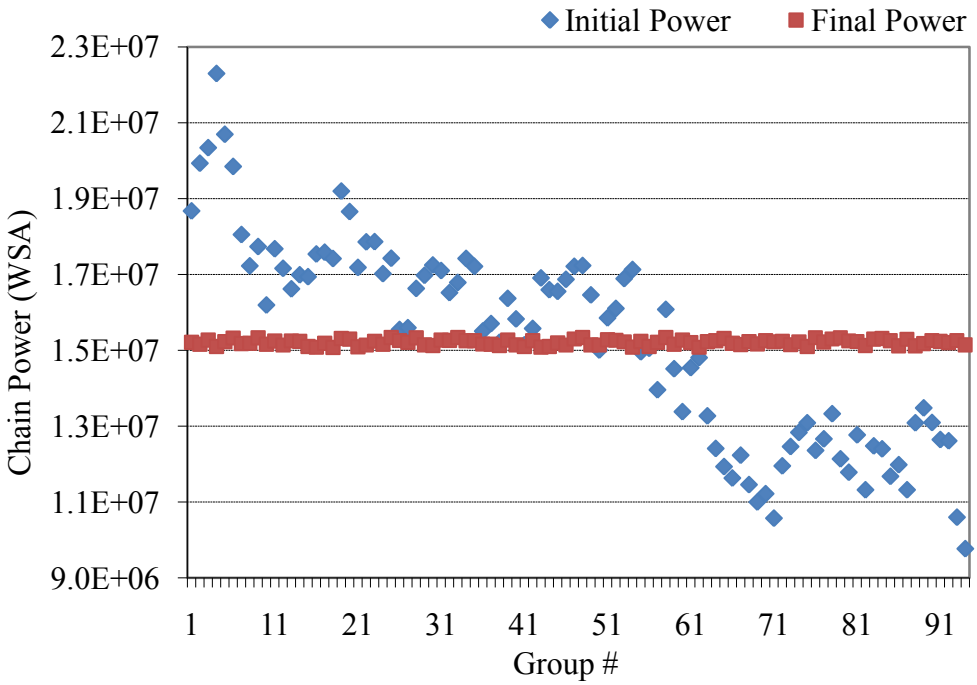

Figure 12. Chip-wise Constant Power Estimation Result for s38417 (pvb=1%)

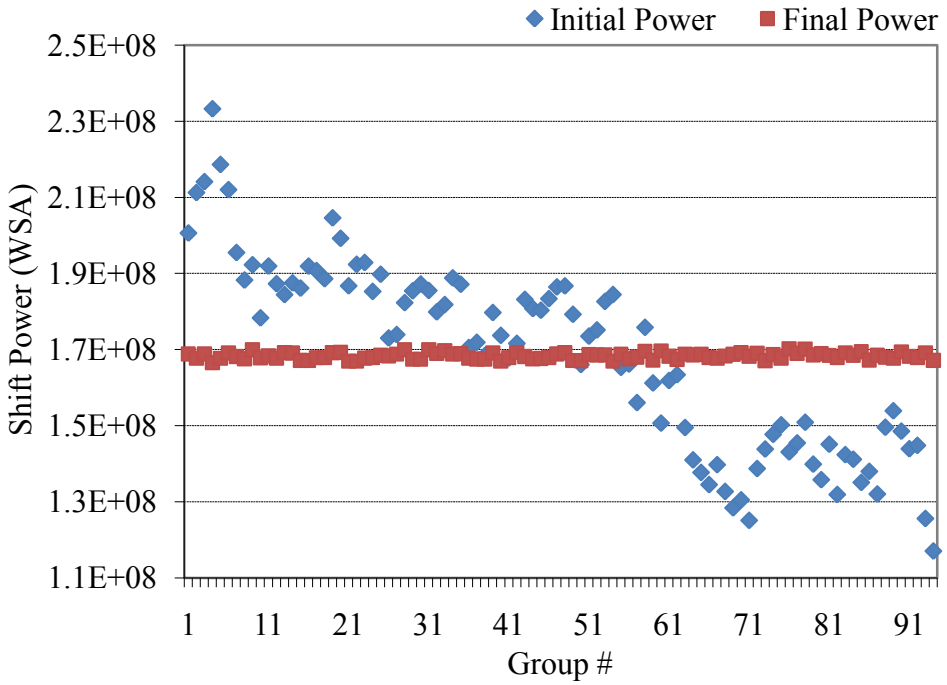

Figure 13. Chip-wise Constant Power Simulation Result for s38417 (pvb=1%)

Figure 12 shows the estimation result of s38417 when *pvb* is set to 1%. It is easy to see the tremendous change in Chain Power before and after reordering. The chain power is almost constant between groups.

Figure 13 shows the simulation result of s38417, running logic simulation on the patterns before and after reordering, to verify our algorithm correctness. We can see that the final total shift power is near constant compared to the initial total shift power. Figure 12 and Figure 13 also showed the power distribution of statically compacted test patterns – the initial patterns are high power and the later patterns are relatively low power.

Note that our pattern reordering algorithm is not capable of reducing the power variation within a group, which means that when we shift the time window along with the time line, the power consumption within a time window will change and the power variation between windows might increase.

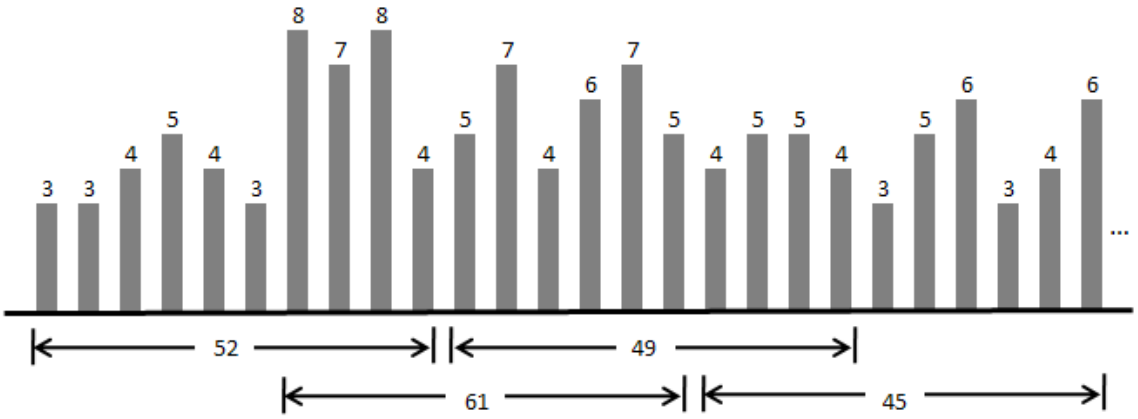

Figure 14. 10 Patterns/Group, Time Window = 10 Patterns, Average Power = 50

Figure 14 shows an example of 10 patterns per group and the time window is the time needed to apply 10 patterns. Although we can achieve constant power for the first two groups, when we shift the window six patterns along the time line, the group power within the two consecutive windows is larger than before. The reason is that the last several patterns in group 1 and the first several patterns in group 2 have higher power. When we shifted the window, the new group 1 happens to have included all those high power patterns and the new group 2 happens to have some low power patterns. To deal with this situation, we can run our algorithm for a small number pattens per group

compared to the actual time window. Given a time window of 100 patterns and if we can have constant power for every 10 patterns' group, the variation of power when shifting the time window for 100 patterns will be much smaller than a time window of 10 patterns.

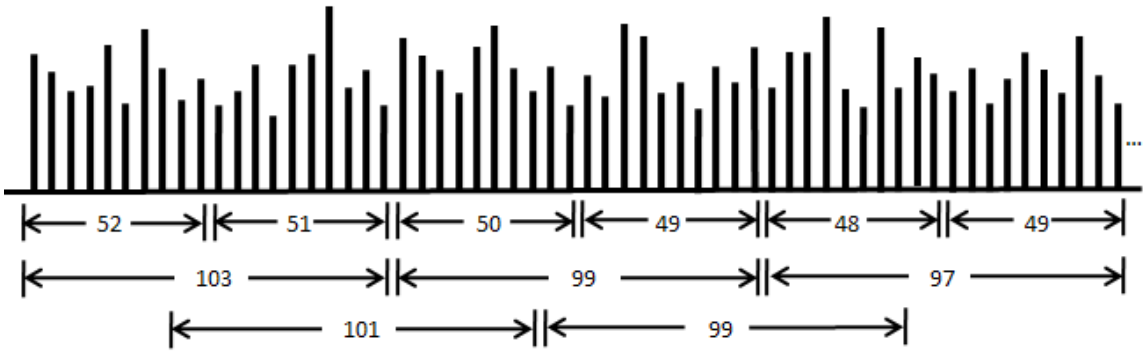

Figure 15. 10 Patterns/Group, Time Window = 20 Patterns, Average Power = 50

Figure 15 shows an example of constant power of 10 patterns per group and the time window is 20 patterns. It shows that when we shift the window, the variation of power in the window is much less than before. If the time window is 50 patterns or even more, the power variations while shifting the time window will be even less.

Table 9. Estimation Results for 50 Patterns per Group in Chip-wise Constant Power Algorithm (pvb=1%) (Part 1)

Circuit	# Patterns (# Groups)	# scan chains	Initial Chain Power (before Reorder)		
			Ave(WSA)	(Max-Min)/Ave	Stdev/Ave
b17	3290(32)	1	80141712	39.77%	10.37%
b21	6579(65)	1	12599386	48.79%	11.31%
b22	8090(80)	1	29478473	39.98%	10.35%

Table 10. Estimation Results for 50 Patterns per Group in Chip-wise Constant Power Algorithm (pvb=1%) (Part 2)

Circuit	Final Chain Power (after Reorder)			Iterations	Total Time (h:m:s)	Reorder Time (h:m:s)
	Ave(WSA)	(Max-Min)/Ave	Stdev/Ave			
b17	80169996	1.98%	0.58%	1	03:42	01:20
b21	12589563	1.79%	0.52%	2	03:36	01:23
b22	29442332	1.87%	0.51%	1	07:34	03:02

Table 9 and Table 10 show the estimation result of three circuits when we use 50 patterns per group instead of the previous 10 patterns per group. Here we set *pvb* to 1% and all circuits use 1 scan chain. Compared to Table 7 and Table 8, the iterations need to reorder patterns drops significnatly because more patterns are grouped, resulting in less power variation. This reduces the iterations needed to even out the power across groups. From the viewpoint of the thermal time constant, 50 patterns would be applied in 0.25 ms, assuming a 500 bit scan chain and 100 MHz scan rate. This is still less than the thermal time constant. Intuitively, the larger the pattern group, the easier it is to achieve constant power. Table 11 and Table 12 show the corresponding simulated results. It can be seen that the shift power (Max-Min)/ave variation and standard deviation are closedto the estimated results, which means using chain power to estimate shift power is a good metric.

Table 11. Simulation Results for 50 Patterns per Group in Chip-wise Constant Power Algorithm (pvb=1%) (Part 1)

Circuit	# Patterns (# Groups)	# scan chains	Initial Shift Power (before Reorder)			
			Ave(WSA)	(Max-Min)/Ave	Stdev/Ave	Time (h:m:s)
b17	3290(32)	1	471425682	31.82%	8.58%	20:14:54
b21	6579(65)	1	278029059	38.26%	8.94%	8:12:53
b22	8090(80)	1	583473664	30.73%	8.06%	24:16:33

Table 12. Simulation Results for 50 Patterns per Group in Chip-wise Constant Power Algorithm (pvb=1%) (Part 2)

Circuit	Final Shift Power (after Reorder)			
	Ave(WSA)	(Max-Min)/Ave	Stdev/Ave	Time (h:m:s)
b17	471636925	1.96%	0.50%	20:15:14
b21	277959273	3.53%	0.78%	8:22:05
b22	583111762	3.21%	0.67%	24:16:09

Table 13 and Table 14 show the estimation results for the Region-wise constant power algorithm with *pvb* set to 5% and 10 patterns per group. We only listed the two largest ISCAS89 circuits and one of the largest circuits in ITC99 since the smaller circuits do not have enough gates to divide into regions. The layouts of these circuits were created using Cadence SOC Encounter with TSMC 180 nm technology. The number of scan chains for s38417, b17 and b19 is 1, 1 and 18 respectively. Since s38417

Table 13. Estimation Results for Region-wise Constant Power Algorithm (pvb=5%, timeout=200, 10 Patterns per Group) (Part 1)

Circuit	Region ID	# Scan Cells	Chain Power before Reorder			Chain Power after Chip-wise		
			Ave	(Max-Min) /Ave	Stdev/Ave	Ave	(Max-Min) /Ave	Stdev/Ave
s38417	1	441	4337023	81.73%	17.57%	4324136	9.34%	2.53%
	2	417	3689107	83.44%	17.57%	3678538	10.25%	2.59%
	3	396	3689289	83.02%	17.86%	3678449	9.35%	2.60%
	4	382	3544983	80.10%	16.98%	3534597	9.08%	2.41%
b17	1	253	4779645	56.99%	11.34%	4777707	12.60%	2.99%
	2	307	3394306	57.43%	11.57%	3392797	11.22%	2.94%
	3	432	4042883	55.79%	11.18%	4041393	10.74%	2.86%
	4	423	3813045	54.31%	10.92%	3811470	13.41%	2.89%
b19	1	1043	5526596	39.40%	7.80%	5526027	19.36%	3.29%
	2	1146	4405387	50.63%	13.17%	4404758	23.71%	4.56%
	3	610	4470274	64.21%	15.46%	4470294	27.58%	5.23%
	4	389	1862794	38.80%	7.28%	1862459	22.32%	3.28%
	5	443	2646726	59.21%	13.80%	2646181	29.51%	4.95%
	6	880	3029939	48.05%	11.84%	3029556	23.23%	3.94%
	7	813	5396912	33.23%	7.72%	5396485	18.91%	3.29%
	8	710	2941721	49.78%	9.98%	2941285	23.26%	4.10%
	9	608	3385835	50.07%	9.61%	3385461	19.27%	3.65%

Table 14. Estimation Results for Region-wise Constant Power Algorithm (pvb=5%, timeout=200, 10 Patterns per Group) (Part 2)

Circuit	Region ID	Chain Power after Region-wise Reorder			Region-wise Iterations	Total Time (m:s)	Total Reorder Time (m:s)
		Ave	(Max-Min) /Ave	Stdev/Ave			
s38417	1	4324136	8.86%	2.26%	2	01:01	00:22
	2	3678538	8.62%	2.25%			
	3	3678449	8.31%	2.32%			
	4	3534597	8.24%	2.13%			
b17	1	4777707	9.53%	2.48%	2	04:27	01:23
	2	3392797	9.70%	2.40%			
	3	4041393	9.49%	2.37%			
	4	3811470	9.50%	2.43%			
b19	1	5526081	8.89%	2.10%	8	52:03	20:12
	2	4404752	9.85%	2.34%			
	3	4470339	9.71%	2.58%			
	4	1862471	9.02%	2.15%			
	5	2646169	9.56%	2.45%			
	6	3029520	9.55%	2.16%			
	7	5396455	9.65%	2.38%			
	8	2941293	9.45%	2.40%			
	9	3385442	8.57%	2.23%			

and b17 are small, we just use 1 chain but b19 has more than 200K gates and 6600+ scan cells. We use multiple scan chains both because this is realistic and it reduces the simulation time.

We divided the layout of s38417 and b17 to 4 regions of the same size (a 2 by 2 division). Since the die of b19 is much larger, we divide it into 9 regions (a 3 by 3 division). The column 'Region ID' identifies different regions and column '# Scan Cells' indicates how many scan cells are in that region. It can be seen from Table 13 and Table 14 that the regions with more scan cells most times has more scan chain power because more scan cells have potentially more switching activity than regions with fewer scan cells. On the other hand, the fan-out of scan cells and the switching in the scan chain are not equal between regions, so more scan cells cannot guarantee more WSA in the scan chain. For example, region 2 of s38417 has more scan cells than region 3 but less average chain power. The column 'Chain Power after Chip-wise Reorder' shows the power of different regions after chip-wise reordering. We saw that the chip-wise reordering algorithm could not achieve constant power for region 2 of s38417 and all regions in b17 and b19. The column 'Region-wise Iterations' shows how many iterations we need in the region-wise constant power algorithm. The column 'Total Reorder Time' shows the total time during reordering including both chip- and region-wise reordering. For b19, we can see that the chip-wise reordering still left large variations within each region but when those variations across regions are added together, we can have constant power over the chip because the low power and high power regions canceled out. After Region-wise reordering, the power variation in each region of b19 becomes constant. For example, the (Max-Min)/Ave and Stdev/Ave of region 3 is 64.21% and 15.46% initially, then reduces to 27.58% and 5.23% respectively after Chip-wise reordering and finally shrinks to 9.71% and 2.58% respectively after Region-wise reordering.

Table 15. Simulation Results for Region-wise Constant Power Algorithm (pvb=5%, timeout=200, 10 Patterns per Group) (Part 1)

Circuit	Region ID	Shift Power before Reorder				Shift Power after Chip-wise Reorder			
		Ave	(Max-Min)/Ave	Stdev/Ave	Time (h:m:s)	Ave	(Max-Min)/Ave	Stdev/Ave	Time (h:m:s)
s38417	1	37797308	75.07%	16.40%	3:23:29	37698602	8.89%	2.39%	3:17:41
	2	41313969	67.90%	14.43%		41220638	8.23%	2.12%	
	3	40979987	69.64%	14.86%		40888432	8.49%	2.18%	
	4	42863191	63.58%	13.41%		42786472	8.22%	1.94%	
b17	1	21802858	46.94%	9.78%	20:35:19	21796151	11.86%	2.65%	20:36:23
	2	25717510	46.89%	9.36%		25709607	9.73%	2.47%	
	3	21079347	44.79%	9.19%		21074909	9.20%	2.44%	
	4	23619558	43.15%	8.90%		23613076	11.39%	2.46%	
b19	1	24178722	31.40%	6.54%	87:23:55	24178168	17.43%	2.76%	87:31:08
	2	18310477	42.93%	11.58%		18307831	21.69%	4.06%	
	3	21087664	56.34%	13.96%		21089830	25.01%	4.71%	
	4	24174107	50.49%	9.42%		24192498	25.45%	3.73%	
	5	31063344	40.25%	8.48%		31079009	17.42%	2.94%	
	6	21009688	40.26%	9.30%		21026229	18.20%	3.15%	
	7	28294492	27.97%	5.69%		28293852	14.10%	2.30%	
	8	23059753	48.99%	7.90%		23095508	21.83%	3.34%	
	9	29903466	51.14%	9.29%		29908170	21.45%	3.83%	

Table 16. Simulation Results for Region-wise Constant Power Algorithm (pvb=5%, timeout=200, 10 Patterns per Group) (Part 2)

Circuit	Region ID	Shift Power after Region-wise Reorder			
		Ave	(Max-Min) /Ave	Stdev/Ave	Time (h:m:s)
s38417	1	37698621	8.21%	2.15%	3:15:45
	2	41220802	7.47%	1.86%	
	3	40888453	7.66%	1.94%	
	4	42786492	7.34%	1.72%	
b17	1	21796151	10.01%	2.21%	20:32:48
	2	25709483	9.50%	2.06%	
	3	21074916	8.55%	2.02%	
	4	23612997	9.27%	2.09%	
b19	1	24178592	7.67%	1.71%	87:30:48
	2	18308219	10.27%	2.17%	
	3	21090074	10.15%	2.38%	
	4	24192466	17.97%	3.08%	
	5	31080053	10.55%	1.87%	
	6	21027518	11.23%	1.99%	
	7	28293600	8.32%	1.70%	
	8	23095128	12.72%	2.41%	
	9	29908894	18.95%	3.27%	

Table 15 and Table 16 show the simulation results based on the estimation results in from Table 13 and Table 14. The column ‘Shift Power before Reorder’ shows the shift

power before reorder. Column 'Shift Power after Chip-wise Reorder' and 'Shift Power after Region-wise Reorder' shows the power after chip-wise and region-wise reorder. Compared to the time for estimation, the simulation time is much longer and infeasible for industrial circuits. After this verification step, we can see that the actual shift power of each region after reordering had less variation than the initial value, which confirms the value of our power estimation metric. For circuit b19, the (Max-Min)/Ave and Stdev/Ave of region 3 is 56.34% and 13.96% initially, then reduces to 25.01% and 4.71% respectively after Chip-wise reordering and finally shrinks to 10.15% and 2.38% respectively after Region-wise reordering. For regions 4 and 9 of b19, the final power variation is 17.97% and 18.95%, which is well above the +/-5% *pvb*, mainly because of the correlation between shift power and chain power is not perfect. However, compared to the original and chip-wise reordering results, our region-wise reordering results are much better in terms of controlling the power within each region.

Table 17. Chip-wise Shift Power Comparison Between Chip-wise and Region-wise Reorder Algorithm

Circuit	Chip-wise Shift Power after Chip-wise Reorder			Chip-wise Shift Power after Region-wise Reorder		
	Ave	(Max-Min) /Ave	Stdev/Ave	Ave	(Max-Min) /Ave	Stdev/Ave
s38417	168457613	8.48%	2.38%	162667295	6.63%	1.81%
b17	94281909	9.51%	2.32%	92193547	8.52%	2.00%
b19	224583866	9.92%	2.04%	221174543	5.67%	1.10%

We also computed the chip-wise power by aggregating the power of each region after region-wise reordering to investigate the influence of the region-wise reordering algorithm on the whole chip based on the results in Table 15 and Table 16. Table 17 shows that the region-wise reordered patterns can achieve better constant chip-wise power than the original chip-wise algorithm. For circuit b19, the (Max-Min)/Ave and Stdev/Ave are 9.92% and 2.04% respectively after chip-wise reordering which shrink to 5.67% and 1.10% respectively after region-wise reordering.

2.7 Enhancement Approaches

The constant power flow has some shortcomings. The first problem is that for some circuits the greedy reordering algorithm cannot achieve a tight *pvb* specification. One observation is that there are some extremely low and high power patterns in the pattern set that make it hard to find a group to put them into to achieve constant power. One way to reduce the number of high power patterns is called *Veto-Compaction*, which is described in Subsection 2.7.1. Another way to reduce the number of low power patterns is called *Noise-Injection* which will be shown in Subsection 2.7.2.

The second problem is that the power estimation model shown in Subsection 2.3 does not work very well for circuits b14, b18, and b19. It may be that some control signals deep in the logic turn on or off many gates. Alternatively, there might be many gates in some circuit levels that are un-evenly distributed compared to other levels. We want to create new metrics to more accurately model the shift power. An approach called *Level-Sim* [17] will be demonstrated in Subsection 2.7.3. It takes the first several levels

of gates from scan chain into account when computing the WSA. This approach achieves higher accuracy when using more levels, but at higher CPU cost. To further address the problem, two other techniques are given in Subsection 2.7.4 and 2.7.5. One is called Toggle Probablistic Analysis considering Single Input Change (TPASIC), which assumes only 1 output of the scan chain toggles, with all other scan chain values held constant with a 50% chance of being 0 and 50% chance of being 1. A preprocessing step computes the WSA for the fan-out cone of each toggling scan cell. This step comprises *N* calculations for an *N*-cell scan chain. Then, we can estimate the shift power for each pattern by summing the fan-out WSA for each toggling scan cell. This technique assumes that toggling fan-out cones do not interact. This technique improves the correlation of b18 to 62%, as shown in Table 2. Another technique called TPASIC considering Adjacent Fill (TPASICAF) was developed. This differs from TPASIC by considering the effects of Adjacent Fill. The difference is that the scan cells besides the toggling value are filled using Adjacent Fill. This will have less average WSA than TPASIC because it is not possible to have other scan cells toggle. So it is less likely to overestimate the shift power WSA. Experiments show that TPASICAF can further increase the correlation to for b18 to 73% compared to only 54% in the original approach in Subsection 2.3. However, we also found that using TPASIC and TPASICAF in pattern reordering, only small improvements in (Max-Min)/Ave and Std/Ave are achieved (as measured by simulation). This suggests that roughly a 60% correlation is good enough to achieve nearly constant power. In addition, since WSA itself is an

estimation of power, it is sufficient to use the fast and accurate enough metric in Subsection 2.3 for power estimation.

2.7.1 Veto Compaction

As described previously, un-compacted test patterns are generated by *CodGen* [33] and then compacted using a greedy forward-order static compaction. This is termed Force Compaction (Force-Comp). This procedure could generate very high-power patterns, if many paths can be packed into a test pattern. We want to minimize the creation of these patterns, since they make it difficult to achieve constant test power. In order to do that, we do a fast pre-check for each pattern: if the transition count (TC) of the two vectors is within a predefined threshold, we can allow the compaction to proceed, else another pattern pair is considered for compaction. The pre-check step is a rough prediction of whether the pattern has high power. The *transition count threshold* (TCT) can be set by experience and it will be the only parameter to influence the compacted vector number in our experiment. We term this step Veto Compaction (Veto-Comp).

Figure 16 is the flow chart of the proposed compaction procedure in our experiment. We set TCT to be 0.05·(the number of bits in each vector). In other words, if more than 5% of the bits in a pattern will transition, this compaction is vetoed. The data below shows the increase in compacted patterns using Veto Compaction.

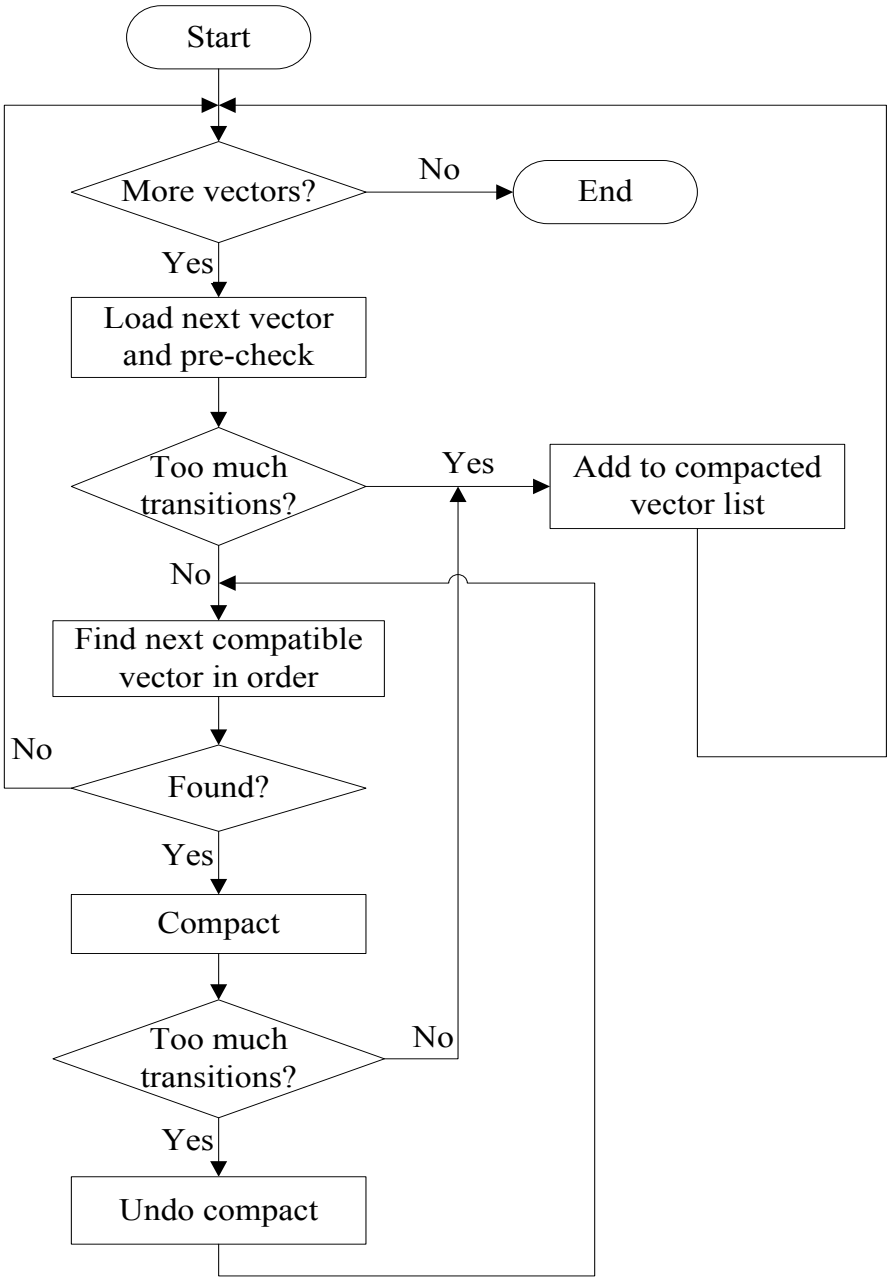

Figure 16. Veto Compaction Flow Chart

Table 18. Pattern Count Comparison (TCT = 0.05)

Circuit	Scan Chain Length	Initial # Patterns	# Patterns after Compaction		
			Force	Veto	% increase for Veto
s15850	534	2646	470	470	0.00%
s38417	1636	14917	948	963	1.58%

Table 19. Transition Count Comparison (Force-Comp vs. Veto-Comp)

Circuit	Transition Count in Pattern					
	Force-Comp			Veto-Comp		
	Ave	Max	Standard Deviation	Ave	Max	Standard Deviation
s15850	10.49	121	12.19	10.57	30	9.13
s38417	57.93	324	29.58	57.21	83	21.5

Table 20. Power Reduction after Using Veto-Comp (vs. Force-Comp)

Circuit	% drop of Max Power		% drop of (Max-Min) Power	
	Capture Power	Shift Power	Capture Power	Shift Power
s15850	17.39%	19.70%	50.95%	19.01%
s38417	4.68%	11.49%	7.18%	9.17%

From Table 18 and Table 19 we can see that Veto-Comp only caused a small increase in pattern count, but caused a large reduction in the maximum transition count and transition count variation. Table 20 shows the power variation reduction after using

Veto-Comp. It can be seen that not only the Max capture power but also the Max shift power were reduced. In addition, the power variation (Max-Min) is also greatly reduced. For s15850, the (Max-Min) for shift power dropped nearly 20%. The results of using these Veto-Comp patterns in pattern reordering will be shown below.

2.7.2 Noise Injection

There may also be some cases with extremely low power patterns that makes it difficult for the test pattern reordering algorithm to find patterns during each swap iteration. We minimize the occurrence of low power patterns using an approach called Noise-Injection. This approach is embedded in the X-Fill process discussed in Subsection 2.2. The modified X-Fill algorithm called *X-Fill-NoiseInject* is shown below.

Algorithm **X-Fill-NoiseInject** ()

1 Pre-Compute the Transition count (*tc[i]*) for each un-filled pattern *i*;
2 Compute the average transition count as *trans_ave*;
3 Compute signal probability *prob* of all PPI1;
4 For each test pattern in the list, do
5 For each pin *p* of PPI1 which has X value
6 if (*prob* < 0.5) then *p* = 0
7 else if (*prob* > 0.5) then *p* = 1
8 else if (*tc[i]* >= *tcb*trans_ave*) then Adjacent Fill *p*
9 else Random Fill *p*; //Noise was injected here
10 For each pin *p* of PI1 which has X value
11 Fill *p* according to the value of *p* in PI2
12 For each pin *p* of PI2 which has X value
13 Fill *p* according to the value of *p* in PI1

```
14  For each pin p of PI1 and PI2 which has X value
15      Randomly fill p;
16  Do logic simulation to fill all X bits of PPI2 by applying V1 as input
```

The major difference from the original X-Fill algorithm is line 1, 2, 8 and 9. Line 1 and 2 first compute the transition count for each pattern and keep a record of the average transition count. During the Preferred Fill process [13] starting at line 5, if the signal probability is 0.5, we first check whether the transition count of this pattern is below a bound (defined by value *tcb*trans_ave, tcb* is set to 0.5 in our experiments), if not, we do Adjacent Fill as before; if yes, we will execute the noise injection approach. The noise injection could have different format and for different patterns we can adjust the rate of injected noise, but for simplicity, we use random fill in our experiments. For example, if a pattern is {01XXX10}, then in normal Adjacent Fill, it would become {0111110}, but after noise injection, it could be {0110010}, two new transitions between the third and fourth bit and between the fifth and sixth bit are introduced. The noise injected brings the power level of the low power pattern up to a higher level, which also could make the constant power algorithm execute faster. Experimental results on ISCAS89 and ITC99 circuits are shown below.

The column 'Force' in Table 21 shows the time/iterations using patterns after Force-Comp and 'Veto' stands for using the patterns after Veto-Comp. We can see that for s38417, the iterations dropped from 14 for Force compacted patterns to 5 for Veto compacted patterns. When we inject noise to Force compacted patterns, the iterations dropped to 8. When we inject noise into the Veto compacted patterns, the iterations

dropped to 4. Since Veto-Comp reduces the Max power and Noise-Inject increases the Min power, using Veto+NoiseInject has the best running time.

Table 21. Constant Power Algorithm Results Comparison for ISCAS89 Circuits

Circuit	pvb	Reorder Time (m:s)				Iterations			
		Force	Veto	Force+ NoiseInject	Veto+ NoiseInject	Force	Veto	Force+ NoiseInject	Veto+ NoiseInject
s15850	1%	0:02	0:01	0:01	0:01	12	7	8	4
s38417	1%	0:14	0:06	0:08	0:05	14	5	8	4

Table 22. Constant Power Algorithm Results Comparison for ITC99 Circuits

Circuit	pvb	Reorder Time(m:s)		Iterations	
		Dynamic	Dynamic+NoiseInject	Dynamic	Dynamic+NoiseInject
b18	2%	Timeout	07:34	Timeout	5
	1%	Timeout	08:01	Timeout	8
b19	2%	Timeout	19:30	Timeout	2
	1%	Timeout	20:13	Timeout	13

The column ‘Dynamic’ in Table 22 shows the time/iterations using patterns after Dynamic Compaction and column ‘Dynamic+NoiseInject’ shows the results that applied NoiseInject into the dynamic compacted patterns. We can see that the Noise Injection approach can produce reordered patterns in a short time while the original patterns without any noise injection cannot meet the *pvb* (1% or 2%) within the timout value (500 in our experiments). We did not conduct experiments with Veto-Comp for b18 and b19 for dynamic compaction.

2.7.3 Level-Sim

Our power estimation approach used in Subsection 2.3 is to estimate the total shift power in the CUT from the WSA in the scan chain. This approach works well for most ISCAS89 and ITC99 circuits but not very well for circuit b14 and b18, with a power correlation below 60%. Here a new approach called Level-Sim can take the next several levels of the circuit into account to increase the accuracy of power estimation.

Table 23. Level-Sim Results for b14 (4800 Patterns)

Level	Correlation	Time (sec)
1	0.568	11
3	0.589	174
5	0.605	198
7	0.645	207
9	0.698	215
11	0.767	240
13	0.812	333
15	0.909	389
17	0.926	482
19	0.978	577
21	0.988	728
61	1	3070

The basic idea of Level-Sim is to do logic simulation for the first n (n << logic depth of the circuit) levels of gates and compute the WSA to be used in the constant power algorithm. Logic simulation is expensive, but if we limit the simulation to the first

several levels, it can be affordable. The simulation results for b14 are shown in Table 23. It can be seen that the scan chain power has only 56.8% correlation with the total shift power. We increase the simulated levels by 2 each time and the correlation increased correspondingly, and is almost 1 at 20 levels, which is only 1/3 of the logic depth.

From all of the other benchmark circuits in ISCAS89 and ITC99, if the correlation is above 80%, the power estimation approach can achieve very good simulation results. For b14, an 80% correlation corresponds to 12-13 logic levels that must be simulated. Although Level-Sim needs much more time than the scan chain power estimation (Level=1), it is an order of magnitude faster than full logic simulation.

2.7.4 Toggle Probabilistic Analysis Considering SIC (TPASIC)

The major issue raised from Level-Sim is the high computational cost for simulation. In addition, it is not possible to determine how many levels to simulate to achieve sufficient correlation, without running a series of experiments. One technique to address this problem is taking the signal toggling of all levels into account by using a probabilistic analysis.

The analysis is comprised of three parts. The first step is to assume that one scan input is toggling (either rising or falling) and all the other scan cells are stable at random values. The second step is to do a pre-calculation of the WSA of the whole circuit for each of the scan cells toggling (N times where N is the number of scan cells) considering the probability. The WSA calculated in this manner is termed the Pseudo-WSA or PWSA. For each scan cell, we calculate PWSA by by propagating the toggle at the scan

through its fan-out cone. Note that there will be 2 calculations as we are considering both rising and falling toggle. The final step is to do a pattern by pattern analysis by taking all the scan cell toggles into account. The idea is to simply sum the PWSA of all scan cells that are toggling in that shift cycle, and then for all shift cycles in the pattern. The aggregated PWSA will be the estimated shift power of this pattern.

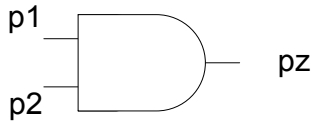

Figure 17. Toggling Probability Analysis for 2-Input AND Gate

For better understanding of this technique, Figure 17 shows a 2-input AND gate. p1 is the probability that input1 toggles (either rising or falling). To compute the toggling probability of the 2-input AND gate, there are three cases to be considered:

Case 1: p1 and p2 are both rising or both falling, which occurs with probability (p1/2)·(p2/2) ·2.

Case 2: p1 is toggling, keep p2 stable and non-controlling, with probability p1·(1-p2)/2.

Case 3: p2 is toggling, keep p1 stable and non-controlling, with probability p2·(1-p1)/2.

The final toggling probability of the output is:

$$\mathrm{pz} = \frac{\mathrm{p1}}{2} \cdot \frac{\mathrm{p2}}{2} \cdot 2 + \frac{\mathrm{p1} \cdot (1 - \mathrm{p2})}{2} + \frac{\mathrm{p2} \cdot (1 - \mathrm{p1})}{2}$$

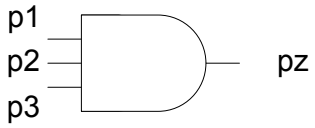

Figure 18. Toggling Probability Analysis for 3-Input AND Gate

Figure 18 shows a 3-input AND gate. p1 is the probability that input1 toggles (either rising or falling). To compute the toggling probability of 3-input AND gates, there are seven cases:

Case 1: p1, p2 and p3 are both rising or both falling, with probability of (p1/2) ·(p2/2) ·(p3/2) ·2.

Case 2: p1 is toggling, keep p2&p3 stable and non-controlling, with probability of p1·((1-p2)/2) ·((1-p3)/2).

Case 3: p2 is toggling, keep p1&p3 stable and non-controlling, with probability of p2· ((1-p1)/2) · ((1-p3)/2).

Case 4: p3 is toggling, keep p1&p2 stable and non-controlling, with probability of p3·((1-p1)/2) ·((1-p2)/2).

Case 5: p1 and p2 are toggling in the same direction (both rising or falling), p3 is non-controlling, with probability of (p1/2) ·(p2/2) ·2· ((1-p3)/2).

Case 6: p1 and p3 are toggling in the same direction (both rising or falling), p2 is stable and non-controlling, with probability of (p1/2) · (p3/2) ·2· ((1-p2)/2).

Case 7: p2 and p3 are toggling in the same direction (both rising or falling), p1 is stable and non-controlling, with probability of (p2/2) · (p3/2) ·2· ((1-p1)/2).

So the final toggling probability of the output (and also of the gate itself) will be:

$$pz = \frac{p1}{2} \cdot \frac{p2}{2} \cdot \frac{p2}{2} \cdot 2 + \frac{p1 \cdot (1 - p2) \cdot (1 - p3)}{2} + \frac{p2 \cdot (1 - p1) \cdot (1 - p3)}{2} + \frac{p3 \cdot (1 - p1) \cdot (1 - p2)}{2} + \frac{p1 \cdot p2 \cdot (1 - p3)}{2} + \frac{p1 \cdot p3 \cdot (1 - p2)}{2} + \frac{p2 \cdot p3 \cdot (1 - p1)}{2}$$

Similar formula can be made for gates with more than 3 inputs and other type of primitive gates.

To compute WSA with probability (PWSA) for each scan cell, we set the toggling probability of this cell to 1 and the toggling probability of all other scan cells and PIs to 0. This is a Single-Input-Change (SIC) vector. Thus we call this technique Toggle Probabilistic Analysis considering SIC (TPASIC).

Then by using the previous described formulae, we can compute the toggling probability of all gates. Summing together the probabilities in the scan cell fan-out cone, we get the PWSA of each scan cell:

$$\text{PWSA of Each Scan Cell} = \sum_{i=0}^{n} p_i \cdot (1 + \text{Fanout}_i)$$

The drawback of this approach is that it has potential to overestimate power as seen from Figure 19. The fanout PWSA of launch scan cell L1 overlaps with the fanout PWSA of launch scan cell L7. The overlap is colored grey. However, considering the low care bit density of test patterns, the overlap effect should be minimal.

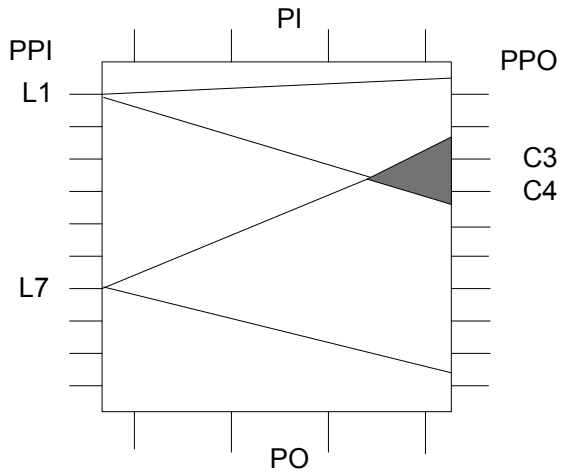

Figure 19. Fanout Cone Overlap

The computational complexity for computing the toggling probablity is O(# of scan cells). While computing the PWSA, we did not use time-consuming simulation such as used in Level-Sim [19]. The experimental results using this technique will be shown in together with Subsection 2.7.5 for comparison.

2.7.5 TPASIC Considering Adjacent Fill (TPASICAF)

The improvement of using the probabilistic technique that is shown in Subsection 2.7.4 over the original metric 3 is visible, but still not good enough for b14, b18, and even for b19. That is because we did not consider the X-Fill effect. In fact, the X-Fill process described in Subsection 2.2 uses Adjacent Fill for all the left over X-bits after Preferred Fill. The computation of the fan-out cone WSA of each scan cell can be more accurately computed by setting the stable scan cell values using adjacent fill. This technique is the same as described in Subsection 2.7.4 except we assume only one scan

input change during pre-calculation and all the other scan cell value are filled using adjacent fill. We will call this technique TPASIC with Adjacent Fill (TPASICAF).

Table 24. Power Correlation Comparison of Different Metrics

Circuit	# scan chains	Correlation using Chain WSA	Correlation using TPASIC	Correlation using TPASICAF
s1488	1	95.40%	93.03%	95.28%
s5378	1	90.90%	89.35%	89.20%
s9234	1	96.70%	95.58%	96.19%
s13207	1	98.00%	97.99%	97.84%
s15850	1	93.20%	84.65%	87.83%
s35932	1	95.50%	87.93%	88.30%
s38417	4	99.60%	99.36%	99.30%
s38584	1	99.40%	98.62%	98.89%
b14	1	56.80%	64.41%	68.32%
b15	1	95.10%	92.15%	93.14%
b17	1	98.80%	97.85%	98.57%
b18	10	54.20%	61.86%	73.02%
b19	9	81.60%	85.99%	94.41%
b20	1	92.70%	90.90%	91.38%
b21	1	91.70%	88.77%	89.75%
b22	1	92.50%	89.89%	89.38%

Table 24 shows the improvement of using TPASIC and TPASICAF over the original scan chain WSA metric in terms of power correlation between simulated shift power and estimated shift power. It can be seen that TPASICAF is overall the best technique among

the three. Specifically for b14, there is 11.54% increase and for b18, there is a 18.82% increase for TPASICAF over Chain WSA. The improvement of TPASICAF over TPASIC is also noticeable in b14, TPASICAF has a 3.91% improvement over TPASIC, and for b18, TPASICAF has a 11.16% improvement over TPASIC. For b19, the improvement of TPASICAF over Chain WSA is 12.81%. For some other benchmark curcits, TPASICAF has slightly worse correlation than Chain WSA. For s15850, the degradation is 5.37%. But this side effect does not influence the pattern reordering result because experimental results showed that a correlation of over 80% is good enough because WSA itself is an estimation of real power consumption.

The constant power result after applying the different power estimation model can be seen in Table 25 where the improvement of TPASIC over Original is not very much. But after TPASICAF is applied, the improvement is visible. The power variation is represented in terms of (Max-Min)/Ave and SD/Ave where SD stands for standard deviation.

Table 25. Constant Power Results Comparison

Circuit	Original		TPASIC		TPASICAF	
	(Max-Min)/Ave	SD/Ave	(Max-Min)/Ave	SD/Ave	(Max-Min)/Ave	SD/Ave
b14	22.36%	4.13%	23.48%	3.71%	20.93%	3.71%
b18	14.92%	2.52%	13.91%	2.51%	12.79%	2.48%
b19	9.92%	2.04%	9.85%	1.98%	9%	1.97%

2.8 Conclusions

In this work, we introduced an X-bit filling technique that targets minimizing both shift power and capture power. Then we proposed an efficient power estimation algorithm based on the power model that estimates shift power from chain power. Finally, a chip-wise and a region-wise test pattern reordering algorithms are shown which generate re-ordered vectors and achieved near constant power. We then showed techniques to improve the results for circuits where the simple power estimation model did not work well. Our future work will be dealing with reducing power variations between different test patterns and further improving the correlation between shift power and chain power.

3. SUPPLY NOISE IN DELAY TEST

3.1 Delay Modeling and Analysis

3.1.1 Power Region Model

Much previous work [35][36][37] has been published on transient power grid analysis. However, RLC or RC network analysis is much too expensive for compaction. Therefore, we make several approximations to simplify the problem. Power grid analysis [24] of bumped chips shows that the supply voltage impact of a switching transient is contained within a local area, since most current flows through nearby pads. Therefore we assume that the supply voltage within a region (e.g. between a set of power pads) is uniform, and the voltage of each region is independent of each other. Hence, voltage drop for any gate in the region is identical. In addition, all switching activities across the region are equivalent, and any switching events outside the region can be neglected.

As manufacture technology shrinks in the DSM era, *dI/dt* effects becomes more and more important as shown in [38][39]. In this research, we only consider power supply noise caused by IR (resistive) voltage drop in the on-chip power grid. This permits modeling the power grid as an RC network. To accurately model and analyze *LdI/dt* (inductive) drop, a RLC network is necessary, which is computationally too expensive [27][40]. We use a power region model similar to that in [30], as shown in Figure 20. C_d is the distributed decoupling capacitance in a region, and C_p is the total parasitic capacitance of devices and interconnects within the region connected to the power

supply network in the current clock cycle. All switching gates that draw current from the supply within this region during the clock cycle are modeled as time-varying current sources $I_{switching\_i}$. The switching current model is discussed in next subsection. $I_{on\text{-}chip}$ is the current from the on-chip capacitance, and $I_{off\text{-}chip}$ is the current from off chip.

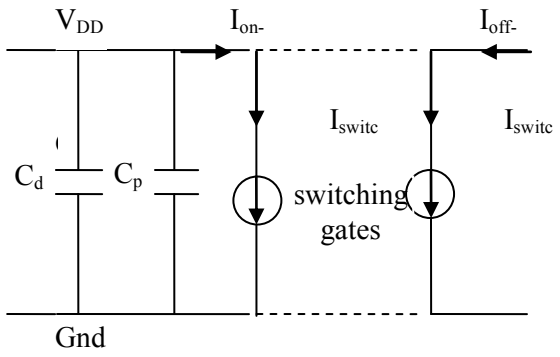

Figure 20. Simplified Power Supply Model in a Region [30]

The maximum regional voltage drop ΔV_{max} during a clock cycle is:

$$\Delta V_{max} = (\int I_{on\text{-}chip}) / (C_d + C_p)$$

$$= (\sum \int I_{switching\_i} - \int I_{off\text{-}chip}) / (C_d + C_p) \qquad (1)$$

We assume that $\int I_{switching\_i}$ occurs over the time of the nominally longest path delay during that clock cycle. After the switching transitions, V_{DD} recovers through $I_{off\text{-}chip}$ to V_{DDinit} at the start of the next cycle.

3.1.2 Circuit Switching Model

We must calculate $\int I_{switching\_i}$ for each logic gate in order to compute ΔV_{max}. Tirumurti [24] created a table of peak power and ground currents for different values of gate output

load and input slope by simulation. We adopt a similar strategy which was used in [30] where a lookup table was created from circuit simulation for all types of primitive gates with different number of inputs. For example, for a NAND gate, we generated data for 2, 3 and 4 inputs NAND gates, similar data was also generated for AND, OR, NOR, NOT gates. Figure 21 shows a typical waveform for an inverter. This waveform is approximated as triangular if the load is small, otherwise as a trapezoid, in order to compute the total charge of each transition. For simplicity, we are not considering ground bounce so the actual capacitance charging occurs only when a rising transition appears. To analysis the extra delay induced by voltage drop along a path, we should compute the capacitance charge over the gates that are on the target path.

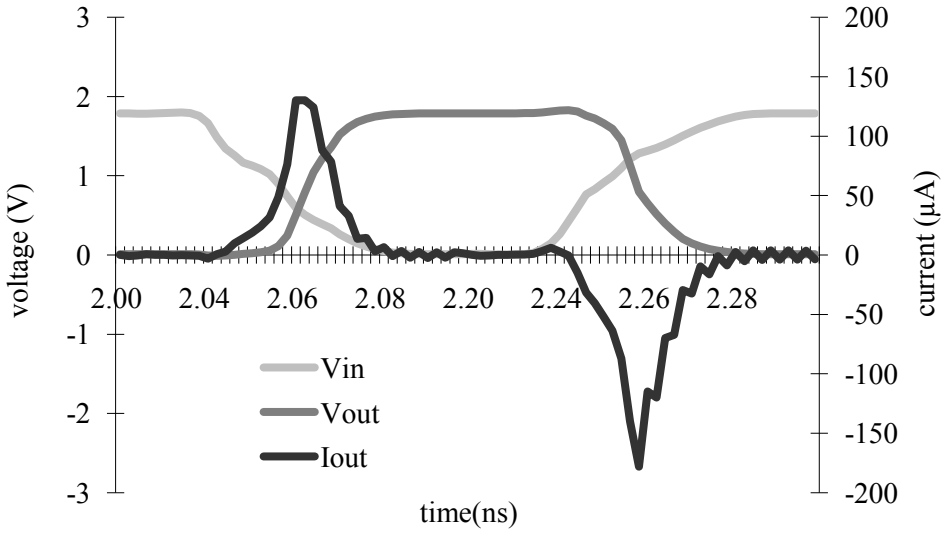

Figure 21. A Current Waveform for an Inverter

3.1.3 Delay vs. Supply Voltage Drop

Several models been proposed for cell delay functions including power supply voltage. Bai [41] proposed using a quadratic delay equation that is a function of the supply voltage, input slope and output load capacitance. He also suggested linear functions of supply voltage if the voltage drop is not too large. The error of this linear model was estimated to be less than 5%. Hence, our model of rising transition delay increase is as follows:

$$\Delta delay / delay = \delta \Delta V / V_{DD} \qquad (2)$$

where *delay* is the nominal delay, ΔV is the estimated voltage drop at the cell, and V_{DD} is the ideal supply voltage. A table of coefficients δ under different output loads and input slopes is obtained by simulation for each cell type. The accuracy of these models was verified with circuit simulation on circuit s1488 [30] and from measurement on industrial design [42].

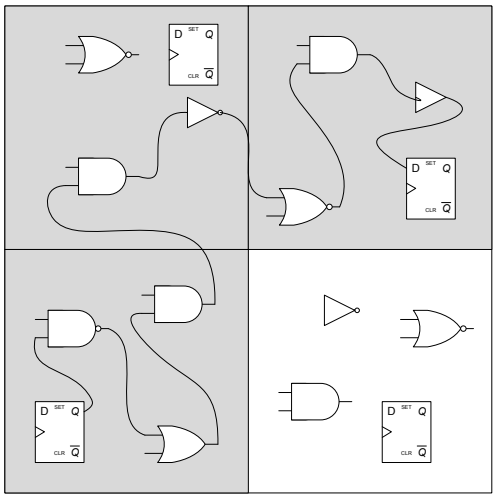

Figure 22. Effective Regions Associated with a Path

We conducted experiments using these models to determine the correlation between voltage drop in the effective regions and delay increase. Here the effective regions are the power regions that the circuit path under test traverses. The three gray regions in Figure 22 shows a chip divided into four power regions (shown here as rectangular for illustration). The regions colored gray are the effective regions for the path shown. The path starts from a scan cell in the lower left region and ends at another scan cell in the upper right region. By the definition of region construction, only the voltage drop in these three regions can affect the delay of the target path. The size of each region is determined by the RC time constant of the power grid.

Figure 23 shows the correlation of voltage drop in effective regions to modeled delay increase for ISCAS89 circuit s38417 for more than 14K paths generated from a delay test ATPG [18], with minimum transition fill of the don't care bits. The correlation is 0.97, which shows that voltage drop is a good estimation of extra delay and voltage drop can be used as a guardband of delay. Since computing voltage drop is computationally less expensive than computing delay, if we know the percentage drop of voltage, we can decide if we have to veto the compaction because of the excessive noise brought to by it.

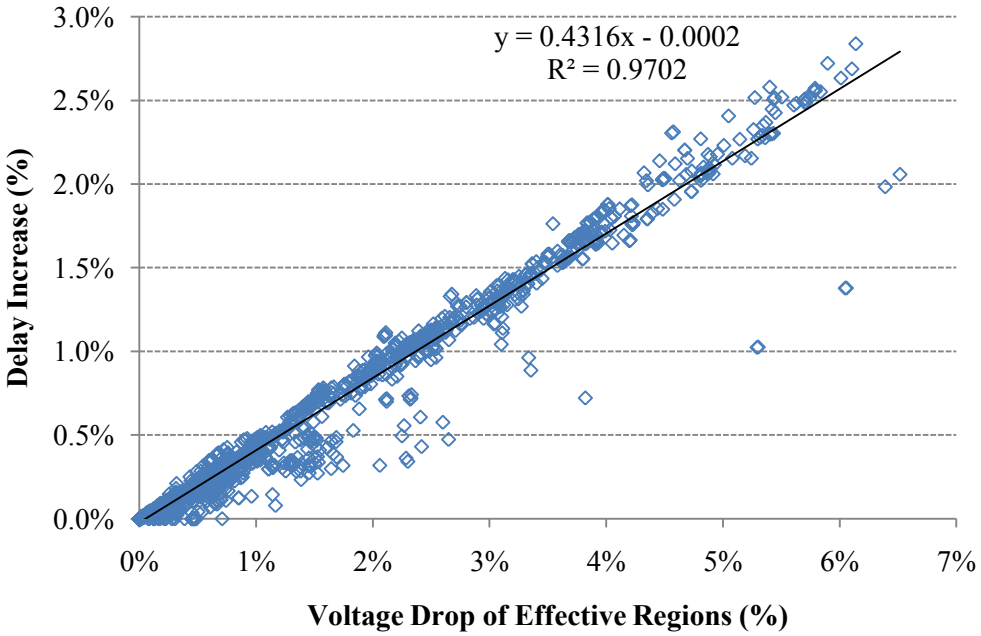

Figure 23. Voltage Drop vs. Delay Increase for s38417

3.1.4 Supply Voltage Drop vs. Effective WSA

Weighted switching activity (WSA) can be used to estimate test power [43].The WSA of a node is the number of state transitions at the gate multiplied by (1+fan-out of the gate). The WSA of the entire circuit is obtained by aggregating the WSA of all the gates in the circuit.

WSA is also a good metric to estimate voltage drop. We conducted experiments to find the correlation between regional voltage drop and the effective WSA. Here effective WSA means the WSA in those regions traversed by the target path. We only consider rising transitions because most supply droop is caused by charging the load capacitance.

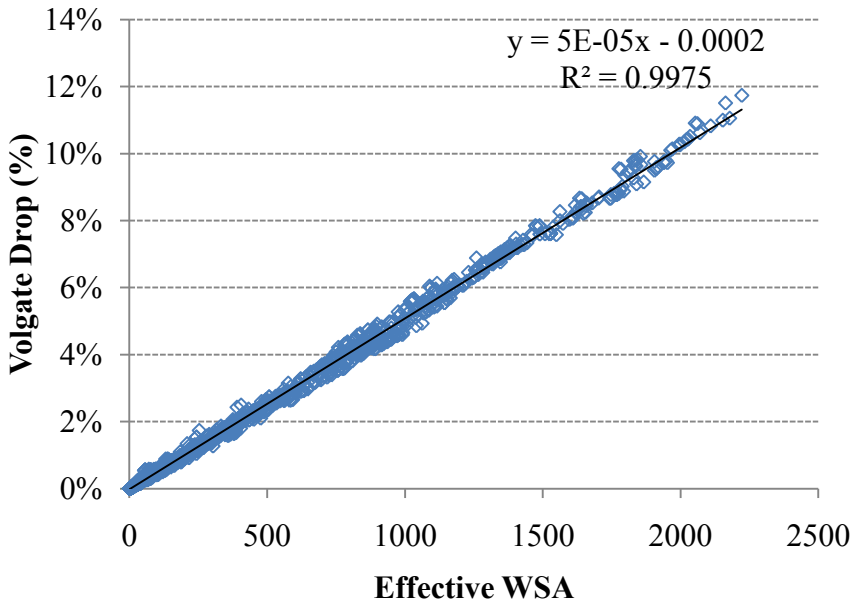

Figure 24. Voltage Drop vs. Effective WSA for s38417

Figure 24 shows near perfect correlation between voltage drop and effective WSA for s38417. We can see that for this circuit, the correlation is almost 100% which provides confidence for us to use effective WSA to estimate/guardband voltage drop which eventually guardbands delay.

In order to compute WSA of the whole circuit, we need to know the test vector pair with all don't care bits filled, and we need to perform a logic simulation before we know which gates has rising transition. Logic simulation is computational expensive which should be used as less frequently as possible. However, if we do not have a good WSA estimation metric, we cannot avoid whole circuit simulation.

On the other hand, since we have to know all the rising transitions before computing voltage drop which means logic simulation is a prerequisite to compute voltage, if we

know the threshold of WSA which corresponds to a threshold of voltage drop, we can skip the voltage computation step which increases the speed of test compaction as well.

3.1.5 Delay Distribution Analysis

Prior work [30] did not distinguish the path length during compaction, so much time was spent unnecessarily checking short paths, and rejecting compaction attempts that did not increase circuit delay. Figure 25 shows the delay distribution of the paths for circuit s38417 in Figure 23. The cell-to-cell Standard Delay Format (SDF) delay was generated using Synopsys *PrimeTime* with 180 nm technology. We can see that many paths are short enough that noise-induced delay will not cause them to exceed the delay of the critical path, and so they can be ignored during compaction.

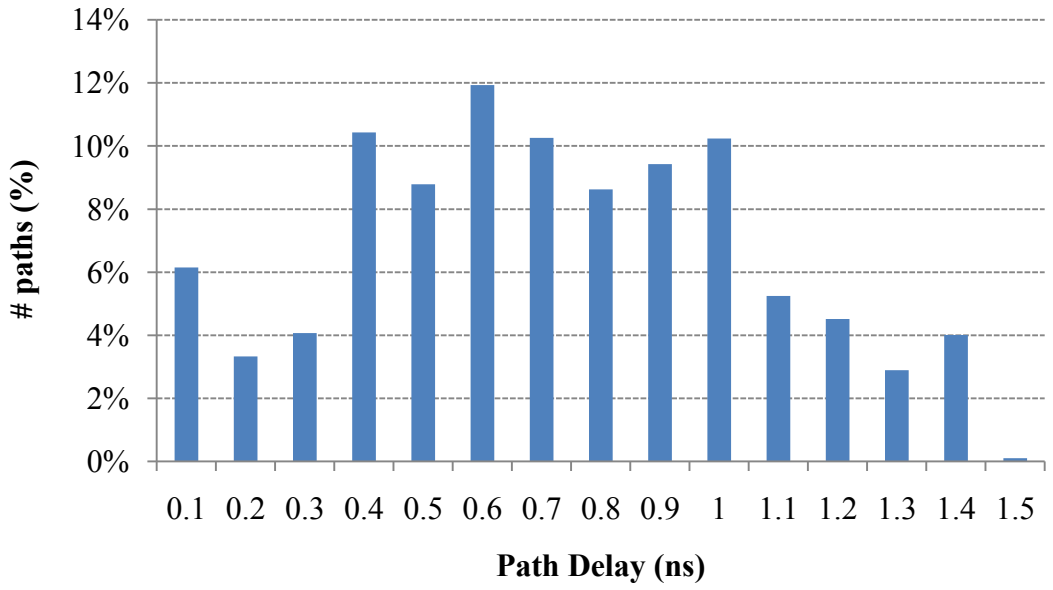

Figure 25. Path Delay Distribution for s38417

As patterns are compacted, one test pattern can contain tests for many paths. As explained above, we will only focus on all of the longer paths tested in that pattern. In such way, we could greatly reduce the delay calculation time by reducing the search range to those long paths while the prior work [30] considered all paths including those short paths. The 'long' paths are those paths that are longer than a threshold which can be a fraction of the maximum length of all paths. During static compaction process, since we know all the paths and test patterns, we can set the threshold before compaction. But during dynamic compaction, since we don't have all the paths generated before compaction, we have to find a global longest path first before compaction. This can be done by searching all structural longest path and justify all the side inputs along this path until we find a two vector pair to test the path.

For the example in Figure 25, the percentage of long paths (path delay > 1ns) is very small so if we only considering those long paths, the speed up of compaction should be huge. Note that this circuit is just a special case, different circuits have different distribution that some of them could have huge percentage of long paths. Even for those circuits whose long paths are dominant, our heuristics would still work better than the old one [30] with experimental results shown in Subsection 3.5.

3.2 Low Cost Supply Noise-Aware Delay Test Static Compaction

We improved on the high cost delay test static compaction algorithm in [30] by exploiting the correlations discussed above. Figure 26 shows our proposed delay test compaction framework that consists of two major steps, with each step having a four-

level estimation flow embedded. The initial test set is one pattern per path which is generated from an ATPG engine [18].

Step 1: Uncompacted paths are loaded in the order generated and a pre-check is performed. Before doing any delay estimation, we will fill the don't care bits for each pattern. The care bit density of each uncompacted pattern is at most a few percent for most circuits. Experience also shows that random fill causes noise that is usually much worse than mission mode [44] and minimal transition fill will potentially have the minimal delay impact so we used minimal transition fill for each vector before analyzing the noise of each vector. Note that the filling process here is not a 'real' filling because after analysis finished, we have to 'unfill' it to restore its original value before compaction. This 'pseudo-filling' takes place each time we begins delay analysis of any vector, both the new vector come into process or the old vector in the compacted list that

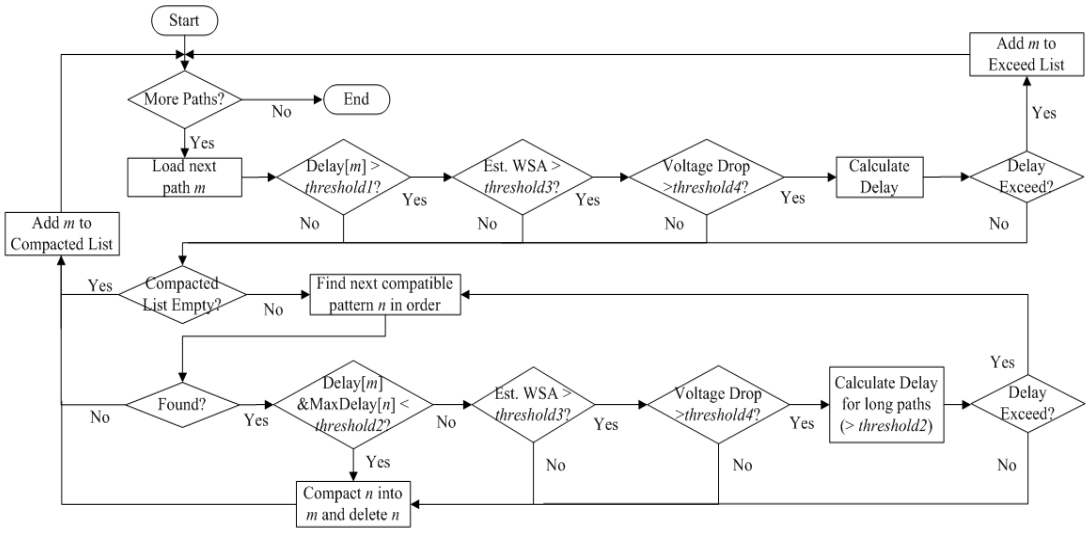

Figure 26. Levelized Low Cost Static Compaction Flow for Delay Test Considering Power Supply Noise

has many vectors been compacted into. During this checking flow, we used a so-called levelized low cost estimation approach, with each level having higher accuracy at higher computational cost. In Level 1, we only check if the SDF delay of this path *m* is too long (set by *threshold1*), if not, we go to Step 2; if yes, we start Level 2, where we estimate the WSA of the pattern to test path *m* without logic simulation. If the WSA is within a limit (*threshold3*), we go to Step 2, otherwise we will go to Level 3, which is similar to the approach in [28]. Logic simulation is used to compute voltage drop and estimate delay. So this level is high cost compared to previous levels. The voltage drop can be easily computed after logic simulation because we know which cells will have rising transitions and how much charge will be consumed during load capacitance charging. If the voltage drop threshold (*threshold4*) is not exceeded, we go to Step 2, otherwise we go to Level 4. Level 4 computes the path delay. If the path delay is above a threshold (delay constraint), this vector is too noisy all by itself, so we put it on an 'Exceed List'. The high supply noise level of vectors on this list is due to ATPG, rather than compaction. Such vectors should be rare, given the low care bit density in path delay test vectors.

Step 2: We try to find a compatible pattern *n* for pattern (path) *m* from Step 1. If the SDF delay of the longest target paths for patterns *m* and *n* are both smaller than *threshold2*, from our previous knowledge, they can safely be compacted. The reason is that two very short paths being compacted will not generate extra delay sufficient to slow the circuit. Here *threshold2* is smaller than *threshold1*, since during compaction, the care bit density and gate switching increases and we want to set a lower threshold to

catch them. If the delay is larger than *threshold2*, we will follow an approach similar to step 1. Note that during each compaction, we will do a 'pseudo-compaction' step to compact pattern *m* and *n* to be a new pattern *n'* before analyzing the WSA, Voltage Drop and delay. If any of the analysis shows negative results, we will discard the compaction and also the new pattern *n'*. Actually *n'* will be the real compacted patterns that be put into compacted list if it passed the delay checking and *n* will be deleted. In Level 4 of this step, we will compute the delay of the long paths using delay look-up tables. If the supply noise level for patterns *n* and *m* together is within limits, compaction is performed and the new vector is added to the set of compacted vectors. If the compaction is rejected, the next compatible vector is considered. We need a fast model to estimate the effective WSA without doing logic simulation. We have tried to use scan chain WSA [43] but the scan chain WSA during the capture cycle does not have good correlation to the WSA in the circuit. The reason that [43] has good correlation is that they are computing the cycle. Prior work [30] used the transition count of each vector pair as a supply noise pre-check, but our simulations show this is not very accurate. To deal with the low correlation issue, [45] proposed a technique called 'Level-Sim' to simulate the circuit for the first several logic levels. Significant correlation improvements were shown on some ITC99 circuits. However, the question is then how to decide the number of levels to simulate. The Level-Sim time is also much higher than computing the scan chain WSA. Therefore, in our static compaction flow, we do not use WSA as a delay estimate. We do use WSA in dynamic compaction because the ATPG

[18] has information about necessary assignments that improves the accuracy of the WSA estimate, while static compaction only has knowledge of the vector pair.

3.3 Supply Noise-Aware Delay Test Dynamic Compaction

Dynamic compaction [31] has been used in KLPG delay test ATPG [18] that shows up to 3x reduction of pattern count over static compaction. The pattern count after dynamic compaction is comparable to the number of transition fault tests, while achieving higher test quality. We modified the supply noise framework described above and embedded it into the dynamic compaction algorithm. The basic idea of dynamic compaction is that for each path that is recently generated by ATPG, we retain the set of necessary assignments (NAs), rather than primary input justification values, since the NAs are unique to each path. When checking two paths for compatibility, the NAs are first checked, and if they are compatible, then a *direct implication* [18] is done to verify compatibility. A direct implication on a gate is one where the input or output value of that gate can be determined from other values assigned to that gate. If direct implication was successful, then a PODEM-based *final justification* [18] is performed to find a vector pair that sensitizes this path. If justification is successful, the new pattern is placed into a Path Pool [31], with each pattern retaining knowledge of the set of paths it contains. After we check pattern compatibility, we perform the noise check before we accept this compaction.

The supply noise aware dynamic compaction flow is shown in Figure 27. The major difference from the compaction flow in [31] is that two checking steps marked in dark

have been added. The first one is called ‘Initially Too Noisy’ which essentially did the Step 1 check which has been depicted in Figure 26. If this step fails which means the newly generated pattern itself is too noisy, we will simply write out this pattern and go on to the next one. This step is still necessary in dynamic compaction because if we neglect this step, some high noise patterns will be appended to the Path Pool which will potentially be compatible candidates during compaction that none of the following noise check could pass. This would consequently induce huge number of redundant noise computation time. The other process embedded is called ‘Pass Supply Noise Check’ which has been added between ‘Pass Justification’ and ‘Update P with F’. It performed the Step 2 operation in Figure 26., If a pattern in the Path Pool fails the check *M* times in a row, then we write out the pattern that the pointer *P* points to. In our experiments, we set *M* to 1000 and the Path Pool size to 5000. Theoretically the higher *M* and Pool size are, the higher the compaction rate and CPU time are. Those values are set by experience and a tradeoff between compaction rate and CPU time.

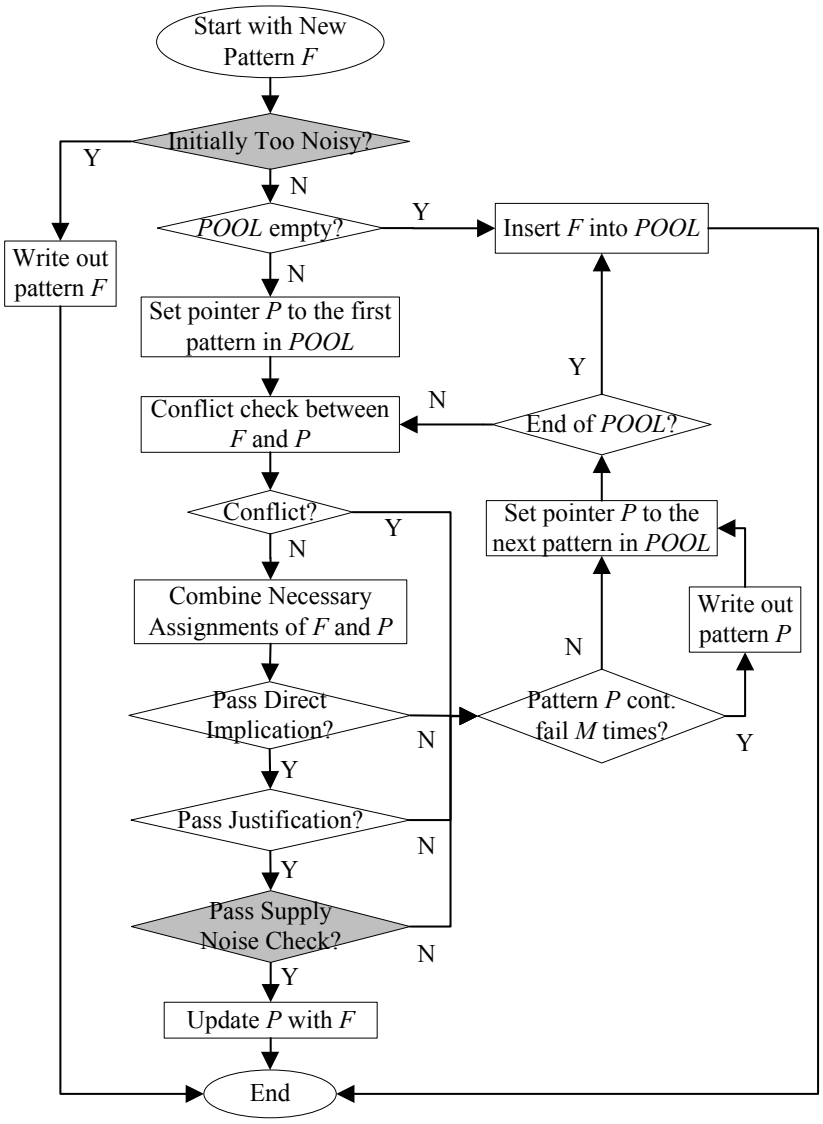

Figure 27. Power Supply Noise-Aware Delay Test Dynamic Compaction Flow

Since dynamic compaction is performed during ATPG, we know the necessary assignments (NAs) of all the internal gates along the new path being considered for compaction. We performed experiments to find the correlation between the WSA of the NAs and the entire circuit. Figure 28 shows the correlation for s38417. The correlation is

high enough that WSA of NAs can be used to estimate whole circuit WSA, which eventually can estimate delay. Note that logic simulation is not required here since WSA of NAs can be used as a guardband for WSA of whole circuit. But in static compaction flow (Figure 26) we have to do logic simulation to compute WSA. From Figure 23 and Figure 28, we can determine *threshold3*. The data in Figure 28 is only available after ATPG is completed, not when we need it during dynamic compaction. A set of long paths can be generated to estimate the maximum WSA. Then we can set *threshold3* to be a fraction of this maximum WSA.

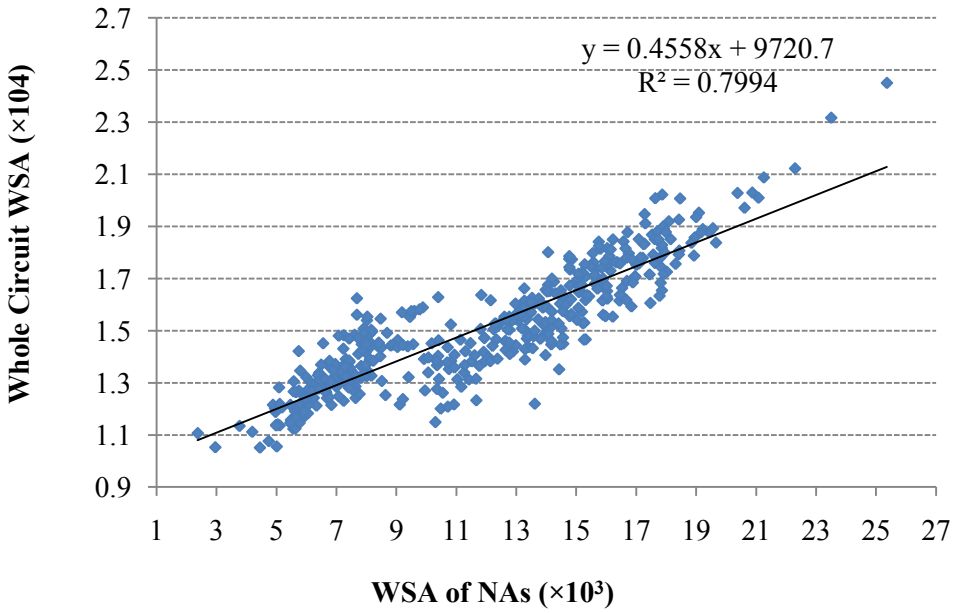

Figure 28. Correlation Between WSA of Whole Circuit and NAs for s38417

3.4 Parameter Setting

As discussed from previous subsections, there are totally 4 parameters used in the compaction flow: *threshold1*, *threshold2*, *threshold3* and *threshold4*. Here *threshold1* is

used to guardband the path length during initial check, *threshold2* is used to guardband path length during compaction, *threshold3* is used in guardbanding WSA and *threshold4* is used to guardband voltage drop. The following rules are proposed on how to set those parameters.

Rule 1: *threshold1* should be set to 75% of the delay of the longest testable path or the delay of system clock period. However, to be conservative, a smaller *threshold1* can be used for the accurate calculation of excessive delay. This recommendation is based on the experimental results shown in Figure 29. The delay increase was caused by compaction and for all the paths generated for s38417, we can see that the delay increase is within 4% to 8% of max delay. Setting *threshold1* to 75% is safe enough to prevent estimation escape since the max delay increase is less than 20%.

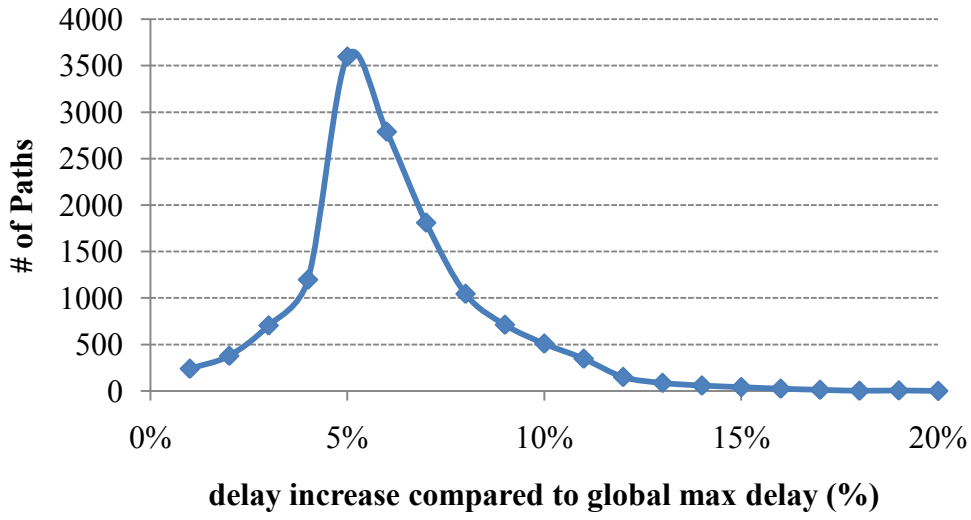

Figure 29. Delay Increase Distribution for Paths in s38417

Rule 2: *threshold2* should be set to 50% of the delay of the longest testable path or the delay of system clock period. The reason that *threshold2* is smaller than *threshold1* is that during compaction, one pattern can test multiple paths which makes the supply noise of all tested paths higher. To be conservative, setting a smaller value for *threshold2* can catch those path delay escapes that pass the *threshold1* due to compaction.

Rule 3: *threshold4* can be estimated by doing a cell delay library simulation before compaction. Just as there is a correlation between voltage drop and delay as shown in Figure 23, we can do a pre-simulation for our delay model by using a sample set of test patterns. For most libraries, we expect to see a correlation similar to Figure 23. The cell delay library could come from SPICE or any other simulation tool. For example, suppose we have a relationship between voltage drop (x) and delay (y) of $x = 2 \cdot y$ with very good correlation (>90%). The formula to set *threhold4* will be *threshold4* = $2 \cdot$ *delay_constraint*. Then if we set the *delay_constraint* to 5% of nominal delay, then we can set *threshold4* to be $2 \cdot 5\%$ which is 10% of nominal supply voltage. However, if the correlation is not very high, say less than 70%, we could conservatively reduce *threshold4*, say to $1.5 \cdot 5\%$ which is 7.5% of nominal supply voltage.

Rule 4: *threshold3* is set by using the correlation between WSA and voltage drop as shown in Figure 24. The results from this figure could come from simulation from a sample of test patterns. The only requirement is to find the trend and correlations. We do not have to simulate all the patterns to get the trend. In order to set *threshold3*, we need to set *threshold4* first because eventually *threshold3* is used to filter the delay, not the voltage and we can use *threshold4* as a bridge for *threshold3* to guardband delay.

Suppose the WSA (x) and voltage drop (y) has a relationship of x = 2000·y with good correlation (>90%) and *threshold4* is 10%. Then the formular to set *threshold3* will be *threshold3* = 2000 · *threshold4*. Then we can set *threshold3* to be 2000 · 10% = 200. However, if the correlation is lower, say less than 70%, we could conservatively reduce *threshold3*, say to 1500·10% which is 150.

Experimental results in Subsection 3.6 show the effects of different parameter settings.

3.5 Pseudo Functional Test Power Analysis

3.5.1 Pseudo Functional Test

Traditional at-speed test can over-test a chip because the supply droop during the capture cycle can slow down the circuit elements. The authors in [46] show observations of a burst of 30 at-speed clock pulses after a period of quiescence.

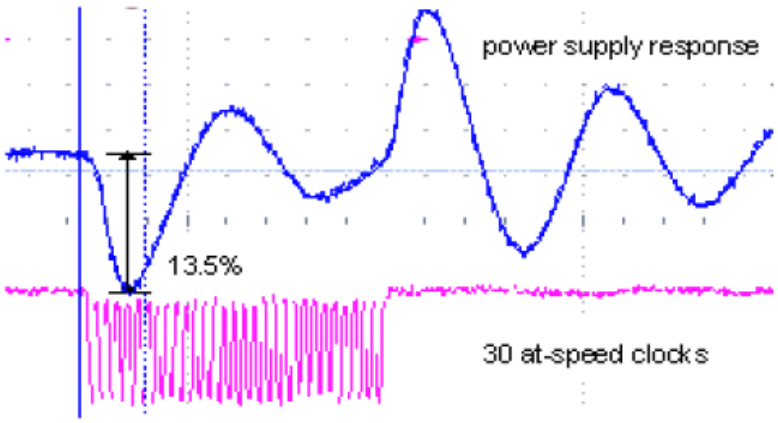

Figure 30. Oscilloscope Droop Measurement [46]

Figure 30 shows that there is a large droop event corresponding to the start of the clock burst, with a typical ringing shape. There is also a power supply overshoot corresponding to the end of the clock burst, as the circuit experiences a sudden drop in current demand. One of the options that has been proposed to reduce the effect of power supply droop is to increase the length of the capture sequence and apply the first few capture cycles at slower speeds.

The experiments in [46] also show that in all the slow/fast capture sequences, the actual droop seen during the at-speed clocks is dramatically reduced and shows that this method can be effective in achieving its goal. It should be noted that longer capture sequences increase test generation complexity significantly. Moreover, ATPG tools need to understand that the slow capture cycles are not catching any speed defects and need to account for this during fault grading. We call this approach *pseudo functional test* and we will analysis the power in terms of WSA in the next subsection.

3.5.2 Multicycle Capture Power

We conducted experiments on ITC99 benchmark circuit b19 by analyzing the WSA of 10 at-speed clock cycles. b19 was selected because it is the largest benchmark circuits we have which comprises 200k gates. Ideally, we should use ATPG tool such as *CodGen* to generate test patterns that launch pseudo-functionally N-1 cycles and capture delay defects at the Nth cycle. Due to the unavailability of CodGen's multiple cycle ATPG functionality, we only use 2 cycle Launch-on-Capture test patterns.

Figure 31 shows the average WSA for b19 using 5319 dynamic compacted patterns for 10 cycles. We can see that at the first launch cycle, Preferred Fill has much lower WSA than random fill due to the former technique's process of minimizing the capture power. However, from the 2nd launch cycle forward until the 10th cycle, we cannot see much difference by using Preferred Fill. In addition, since we are hold the PIs constant during the 10 cycles, the WSA is gradually falling off to a steady state where the WSA is very low compared to the 1st cycle. This is partially due to the switching activity intentionally made by ATPG to sensitize the path in the 1st cycle gradually faded away after multiple cycles.

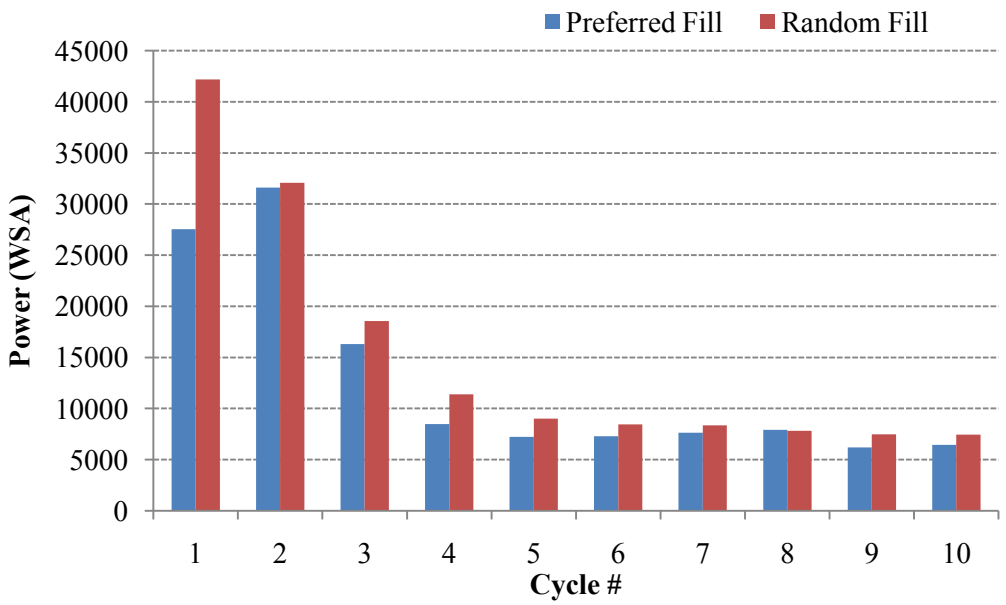

Figure 31. Average WSA for b19

The other finding from this is that we need new X-Fill technique to boost the WSA of the 10th cycle, which is the actually capture cycle. This will be part of our future work.

3.6 Experimental Results

The realistic low cost delay test compaction framework was implemented in C++ and running on a Windows platform with Core 2 Duo 2.66 GHz CPU and 2GB DDR2 memory. The circuit layout was created using SOC Encounter and 180 nm technology. By default, we set *threshold1* to be 0.75 and *threshold2* to be 0.5 of maximum delay of all paths. We set *threshold3* to 0.5 of maximum WSA and *threshold4* to be the same as the circuit delay constraint, since we want to be conservative in using the correlation shown in Figure 23. Ideally, the higher the threshold we set, the faster the compaction flow will be, but the greater the risk of creating test patterns that exceed the desired noise levels.

The path delay patterns are generated with the *CodGen* K-Longest Path per Gate (KLPG) ATPG [18] tool. It is used to generate the 2K longest paths through each line in the circuit, with K paths having a rising transition at the fault site and K paths having a falling transition. In this work we will use K=1.

Table 26 and Table 27 show the static compaction results for four ISCAS89 circuits by comparing the low cost framework discussed in the paper and the high cost framework [30]. For each benchmark circuit, we conducted experiments on several different delay constraints whose value can be seen in column ‘Delay Constraint’. Initially, we used a greedy forward-order procedure to compact all the patterns without considering supply noise. We term this procedure “force compaction.” It corresponds to ‘No’ in the ‘Delay Constraint’ column. We term the noise-aware compaction approach “veto compaction” because we will veto any compaction that violates our delay

Table 26. Low Cost Delay Estimation Framework During Static Compaction for ISCAS89 Circuits

Circuit	# Paths	Delay Constraint	Low Cost Framework				
			Total Time (m:s)	Delay Estimate Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
s1488	167	3%	0:01	0:01	85	5	89
		5%	0:01	0:01	83	4	90
		7%	0:01	0:01	82	3	91
		10%	0:01	0:01	79	0	94
		18%	0:01	0:01	79	0	94
		No	0:01	0:01	79	0	0
s15850	2415	3%	0:14	0:11	483	15	3279
		5%	0:13	0:10	481	11	2874
		7%	0:12	0:09	480	8	2660
		8%	0:08	0:05	467	0	1556
		16%	0:08	0:05	467	0	1556
		No	0:03	0:00	467	0	0
s35932	9442	3%	2:42	1:35	122	62	24295
		5%	2:22	1:15	72	20	17440
		7%	1:52	0:45	49	1	13535
		8%	1:44	0:37	46	0	11187
		No	1:07	0:00	46	0	0
s38417	14405	3%	3:32	2:39	1093	157	19338
		5%	3:10	2:17	996	30	16679
		7%	2:57	2:04	977	1	14860
		8%	2:56	2:03	977	0	14826
		14%	2:56	2:03	977	0	14826
		No	0:53	0:00	977	0	0

Table 27. High Cost Delay Estimation Framework During Static Compaction For ISCAS89 Circuits

Circuit	# Paths	Delay Constraint	High Cost Framework				
			Total Time (m:s)	Delay Estimate Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
s1488	167	3%	0:01	0:01	117	46	217
		5%	0:01	0:01	106	33	228
		7%	0:01	0:01	98	24	236
		10%	0:01	0:01	93	16	241
		18%	0:01	0:01	79	0	260
		No	0:01	0:01	79	0	0
s15850	2415	3%	0:19	0:16	602	203	4280
		5%	0:21	0:18	539	87	4319
		7%	0:21	0:18	510	53	4320
		8%	0:21	0:18	503	40	4329
		16%	0:22	0:19	467	0	4362
		No	0:03	0:00	467	0	0
s35932	9442	3%	37:50	36:43	1421	1010	311979
		5%	9:41	8:34	249	133	71223
		7%	3:59	2:52	71	6	29574
		8%	2:31	1:24	51	0	18557
		No	1:07	0:00	46	0	0
s38417	14405	3%	30:24	29:31	1941	960	210646
		5%	14:54	14:01	1265	275	100713
		7%	7:36	6:43	1103	129	49280
		8%	6:15	5:22	998	17	38957
		14%	4:24	3:31	977	0	24974
		No	0:53	0:00	977	0	0

constraint. Column ‘Total Time’ is the total compaction time while “Delay Estimate Time’ is only the time used in the delay estimation including logic simulation and table lookup. Column ‘# Patterns After Compaction’ is the total number of patterns after compaction which includes all the ‘# Exceed Paths’ which are the paths that are initially too noisy and are put into the ‘Exceed List’ in Step 1 of Figure 25. The number of logic simulations made during delay estimation is shown in column ‘# Simulations’.

Since the high cost model in [30] considers all paths as target paths without looking at their nominal delay, it will reject many compactions, even though the same amount of extra delay for those short paths will not actually cause a timing issue. For example, as shown in Figure 32, suppose the delay bound of a circuit is 1 *ns* and the maximum path delay is 0.9 *ns*. Then for safe compaction, we should set the delay constraint to be 11% of max delay, which is 0.1 *ns*. However, a compaction is still safe if it adds 0.3 *ns* extra delay to a short path #1 with nominal delay of 0.5 *ns*. If the extra 0.3 *ns* is added to the 0.9 *ns* path #2, it would violate the delay constraint, so this compaction should be vetoed. During static compaction, we know the maximum nominal delay of all paths, so we know the threshold of how much extra delay we can tolerant. As a result, the low cost model can reduce unnecessary simulations and accept some compactions that were rejected by [30].

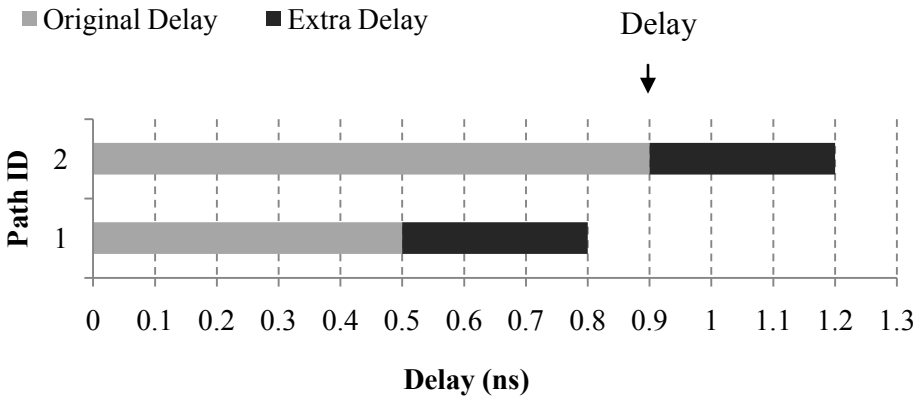

Figure 32. Delay Constraint Effect on Different Paths

The high cost framework has used some redundant delay estimation pruning technique by simply overlook the delay calculation if the two compatible patterns being compacted have very small transition count [47]. The transition count threshold was set by experience that could be simply as a tiny fraction of the total number of bits in a vector. However, this pre-check is prone to delay underestimation because small transition count cannot guarantee short path. For example as shown in Figure 33, the vector pair to test path B has higher transition count (2) than the transition count (1) of vector pair to test path A, but path B in fact is shorter than path A. This applies to both robust and non-robust transition fault test. Therefore, in our low cost framework, we did not use transition count as a threshold.

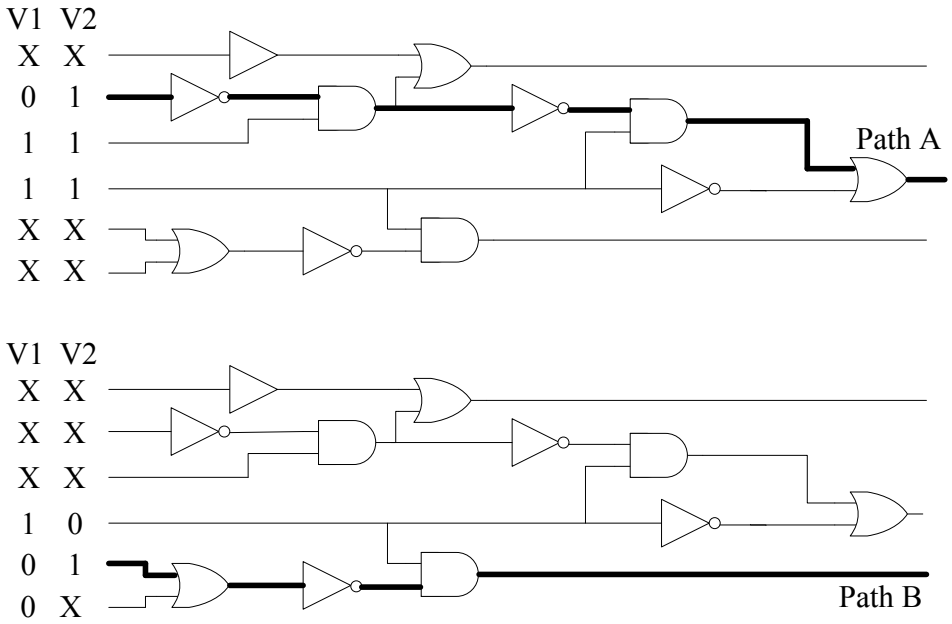

Figure 33. Vector Pair Transition Count on Different Paths

For all the delay increase constraints considered in Table 26 and Table 27, the low cost model has smaller pattern count after compaction than the high cost model. For circuit like s38417 with a 5% delay increase constraint, the low cost model is 5x faster than the high cost model in delay estimation. For a 3% delay constraint, the low cost model's pattern count is almost half the size of the high cost model. The number of simulations also has been greatly reduced. For a 3% delay constraint, more than 210K simulations are need in the high cost model but only around 45K simulations are used in the low cost model. Also, the high cost model needs the constraint to be relaxed to 15% constraint to generate the 977 patterns of the force compaction model, while the low cost model achieves this pattern count while meeting a 7% constraint. For circuit like s35932, the high cost framework with 3% delay constraint generates 1421 patterns with 1010

originally failed patterns that are more than 10 times the number of the low cost framework. The major problem is that the high cost model considers delay constraints *per path* while low cost framework considers delay constraints *globally*.

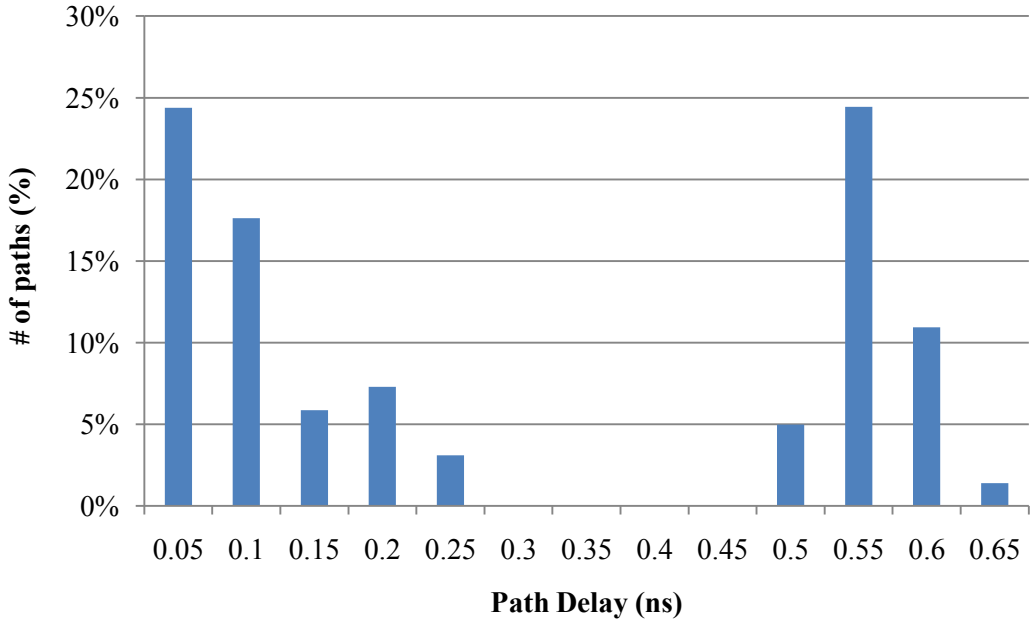

Figure 34. Path Delay Distribution for s35932

Figure 34 shows the path delay distribution of 9442 paths of circuit s35932 that were generated from ATPG where many are short paths. It is interesting that most paths are either short or long with no paths in the middle range of delay (0.3~0.45 ns). For those paths that are shorter than 0.1 ns delay, the high cost framework would probably think some of them are too noisy during initial check and put them to originally failed patterns while low cost framework would not. In addition, this circuit has an extremely high compaction rate compared to other circuits, partially due to the high portion of short

paths with low care bit density which on the other hand increases the compatibility of two vectors.

The delay constraint used in [30] is applied per path, not the globally longest path. For example, it will reject compaction if the extra delay of path *P* is over a delay constraint (normally a fraction, or *x*% of the original delay of path *P*). Realistically, the delay constraint should be set a percent of the global longest path that determines the clock cycle of the circuit. In our experiments, we will veto any compaction if the extra delay is over *x*% of the global longest path delay. This information is available for static compaction since we already have the complete path list but not available for dynamic compaction unless all the paths are generated. However, *CodGen* [18] has the ability to generate the global longest path first before it generates other paths such that we can use that information to set our delay constraint.

In order to further show the efficiency of low cost framework, we applied the delay constraint similarly on high cost framework using the globally longest path delay. Table 28 and Table 29 show the experimental results on s38417 with different delay constraint.

Table 28. Low Cost Framework During Static Compaction for s38417 With Same Delay Constraint Metric Applied
(Threshold1=0.75, Threshold2=0.5, Threshold3=0.5, Threshold4=Delay Constraint)

Delay Constraint	Low Cost Framework				
	Total Time (m:s)	Delay Estimate Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
3%	3:32	2:39	1093	157	19338
4%	3:24	2:31	1026	74	18313
5%	3:10	2:17	966	30	16679
6%	2:58	2:05	983	9	14912
7%	2:57	2:04	977	1	14860
8%	2:56	2:03	977	0	14826

Table 29. High Cost Framework During Static Compaction for s38417 With Same Delay Constraint Metric Applied
(Threshold1=0.75, Threshold2=0.5, Threshold3=0.5, Threshold4=Delay Constraint)

Delay Constraint	High Cost Framework				
	Total Time (m:s)	Delay Estimate Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
3%	5:40	4:47	1093	157	29436
4%	5:23	4:30	1026	74	28414
5%	5:10	4:17	966	30	26771
6%	4:58	5:05	983	9	25010
7%	4:57	5:04	977	1	24979
8%	4:57	5:04	977	0	24948

In order to verify that our low cost framework is more realistic than high cost model, we also simulated the delay of all the paths after static compaction by exporting one pattern per target path. It is an important step to see after compaction, whether all the paths' delays are within timing bound because a higher compaction rate would induce higher supply noise. For circuit s38417, before compaction, the max path delay is 1.44 ns, and the 5% delay constraint is set to 0.07 ns, which increases the delay bound to 1.51 ns.

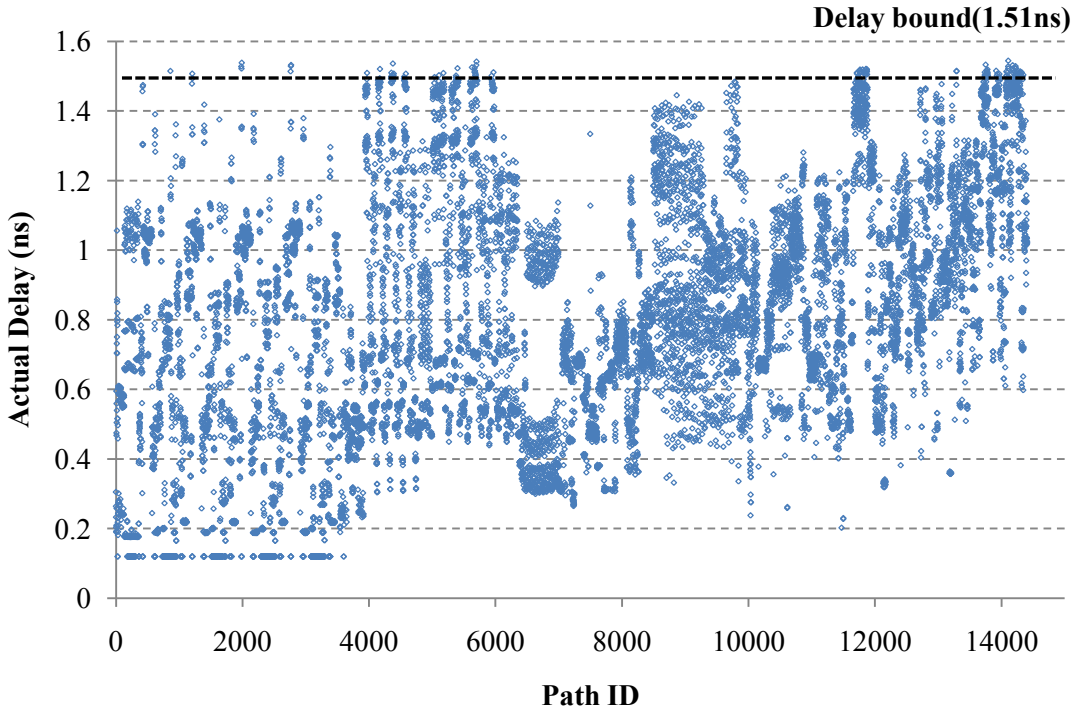

Figure 35. Actual Path Delay After Compaction for s38417

Figure 35 shows that our low cost model can keep the max path delay of almost all paths within the delay bound except for those 30 patterns that are originally too noisy. Those noisy patterns are coming from ATPG while not from compaction because in

order to test those paths, the excessive switching in those effective regions will reduce the voltage and introduce extra delay that goes beyond the delay constraints. However, we still have to test those paths by relaxing the delay constraint or make the circuit running at a slower speed.

Figure 36 shows that the extra delay induced by compaction. We can see that for some short paths, we can tolerate them having extra delay larger than 0.07 ns while for long paths, they must strictly obey the constraint. While for high cost framework, it is obvious that all extra delay would be constrained to within 0.07ns.

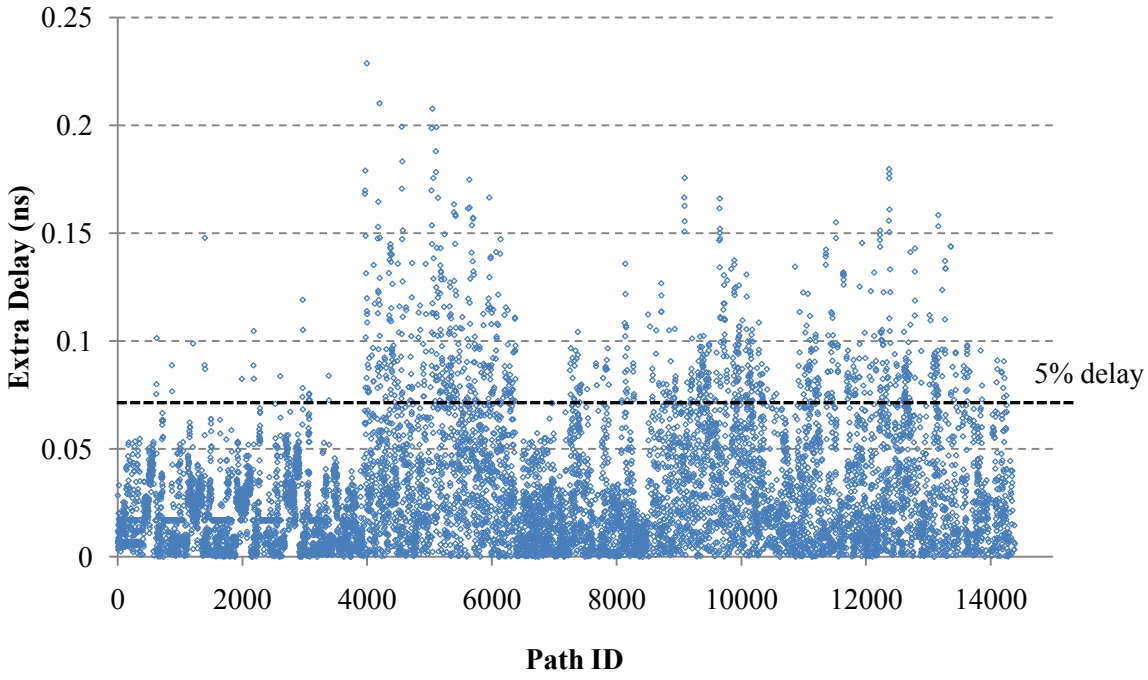

Figure 36. Extra Path Delay After Compaction for s38417

We also analyzed the effect of different thresholds on the compaction speed. Table 30 shows that by changing *threshold1* and *threshold2* on 3% delay constraint with

threshold4 fixed to 3%, the number of simulations differs accordingly. When we fixed *threshold1* to 0.75 but changed *threshold2*, we can see that the higher the *threshold2*, the less the number of simulations because we are reducing the number of patterns under delay estimation by filtering out even more short paths. Similar results can be seen if we fixed *threshold2* to 0.5 and altered *threshold1*. The smaller *threshold2* is, the higher the number of simulations is needed because we enlarged the number of patterns for delay estimation which includes shorter paths. Note that from the data in Table 29, we can see that our default setting of threshold pair [*threshold1*, *threshold2*] which is [0.75, 0.5] is pessimistic because for the highest 'working' pair which is [0.8, 0.7] we set, the number of patterns after compaction is the same but the delay estimation time is 58 seconds or 15% less. Here 'working' means the number of patterns after compaction is the same with [0.75, 0.5]. On the other hand, we should limit the value of *threshold1* or *threshold2* in case the compaction flow underestimates the delay which can be seen in the threshold pair [0.9, 0.7]. For this case, we are too optimistic about the guardband of threshold1 that for some paths that are shorter than 90% but longer than 80% of the longest path, they could bring excessive switching in the circuit that slows down the transition propagation that goes beyond the delay constraint we set. The '# Exceed Paths' shows 23 patterns less than the threshold pair [0.8, 0.7] which means some patterns skip the delay check and were added into the final compaction pattern set. The side effect of those patterns is that they could be compatible with other patterns but they will fail the delay check every time. The redundant delay checking will bring more simulations and which explains the higher running time that did no good to our

compaction. So be conservative, we should set the *threshold1* and *threshold2* to smaller values. To our experience, [0.75, 0.5] is a good conservative pair.

Table 31 shows the effect of different *threshold4* on the results of static compaction using low cost framework. Here we keep the delay constraint to be 3% of max delay, *threshold1* to be 0.75 and *threshold2* to be 0.5. From Figure 23 we can see that *threshold4* could be any value less than twice the value of delay constraint which is 6% for s38417. The experimental results show that setting *threshold4* to 3% is very conservative as it vetoes many compaction which is safe for supply noise where we can see directly from column '# Exceed Paths'. However considering the inaccuracy of delay model we used, setting a smaller *threshold4* can guarantee that we are not overlooking any extra delay caused by supply noise.

Table 30. Low Cost Delay Estimation During Static Compaction for s38417 with Different Threshold1 and Threshold2 (Delay Constraint=3%, Threshold4=3%)

Threshold1	Threshold2	Total Time (m:s)	Delay Estimate Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
0.55	0.5	3:57	3:04	1093	157	22780
0.65	0.5	3:42	2:49	1093	157	20631
0.75	0.5	3:32	2:39	1093	157	19338
0.75	0.6	3:23	2:30	1093	157	18467
0.75	0.7	2:54	2:01	1093	157	14350
0.8	0.7	2:47	1:54	1093	157	13882
0.9	0.7	9:07	8:14	1091	134	62061

Table 31. Low Cost Delay Estimation During Static Compaction for s38417 with Different Threshold3
(Delay Constraint=3%, Threshold1=0.75, Threshold2=0.5)

Threshold4	Total Time (m:s)	Delay Estimate Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
3:32	2:39	1093	157	19338	3:32
3:32	2:39	1093	157	19338	3:32
3:33	2:40	1091	151	19514	3:33
3:34	2:41	1089	147	19865	3:34

Dynamic compaction (DC) results on four ISCAS89 circuits can be seen in Table 32 and Table 33. As with static compaction (SC), the high cost framework is slower due to more simulations. Similar like experiments on static compaction, before applying the frameworks, we conducted force compaction that did not consider supply noise effects. For example, for s38417, if we use the low cost model, a 10% constraint produces 389 patterns in less than 5 extra CPU minutes. At a 7% constraint, only 4 additional test patterns are generated. For a 3% delay constraint, 353 extra patterns are generated, using considerably more CPU time. The large CPU time increase is due to the many patterns near their delay increase thresholds, requiring many more simulations. In addition, each time a compaction was rejected due to noise, we must find another compatible pattern and pass both the direct implication and final justification phases, which require significant time. Column ‘Extra Time’ includes all these efforts and delay estimation. For high cost model, more than half an hour extra time was spent for 3% delay constraint

which is almost 2x slower than low cost model and the pattern count is more than twice of the low cost model. For delay constraint like 5% and 7%, delay estimation of low cost model is still 2x faster than high cost model together with huge reduction of pattern count. By comparing Table 25 and Table 31, we can see that DC needs more execution time but has smaller pattern count than SC. Also in both cases the low cost framework works well until the delay constraint becomes so stringent that many patterns are close to the constraint, and so require detailed analysis. The reason that DC has fewer simulations than SC for the 3% delay constraint, even though it has a higher compaction rate, is that the if two short paths are compacted during DC that do not need noise estimation, but they are not compatible in SC while they are compatible with a long path, then we need to estimate supply noise in SC for the compacted pattern. But for most cases, DC should have more number of simulations than SC due to the high compaction rate which need more number of delay estimations each time two patterns are compacted together.

Table 32. Low Cost Delay Estimation Framework During Dynamic Compaction

Circuit	Delay Constraint	Low Cost Framework				
		Total Time (m:s)	Extra Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
s1488	3%	0:02	0:01	66	5	100
	5%	0:02	0:01	64	4	101
	7%	0:02	0:01	63	3	102
	10%	0:02	0:01	60	0	104
	16%	0:02	0:01	60	0	104
	No	0:01	0:00	60	0	0
s15850	3%	1:49	0:30	294	15	4146
	5%	1:46	0:27	291	11	3607
	7%	1:40	0:21	291	8	3068
	8%	1:36	0:17	279	0	2338
	16%	1:35	0:16	274	0	2285
	No	1:19	0:00	274	0	0
s35932	3%	9:26	5:09	108	62	21000
	5%	8:27	4:10	57	20	17534
	7%	7:57	3:40	35	1	15455
	8%	7:37	3:20	31	0	14561
	No	4:17	0:00	28	0	0
s38417	3%	40:34	18:03	742	295	36983
	5%	35:05	12:34	469	63	20146
	7%	33:04	10:33	393	0	15979
	10%	32:22	9:51	389	0	15749
	15%	32:22	9:51	389	0	15749
	No	22:31	0:00	389	0	0

Table 33. High Cost Delay Estimation Framework During Dynamic Compaction

Circuit	Delay Constraint	High Cost Framework				
		Total Time (m:s)	Extra Time (m:s)	# Patterns After Compaction	# Exceed Paths	# Simulations
s1488	3%	0:03	0:02	96	46	238
	5%	0:03	0:02	87	33	247
	7%	0:03	0:02	79	24	255
	10%	0:03	0:02	74	16	240
	16%	0:03	0:02	60	0	270
	No	0:01	0:00	60	0	0
s15850	3%	1:45	0:25	447	203	4554
	5%	1:48	0:27	350	87	4592
	7%	1:45	0:24	318	53	4514
	8%	1:46	0:25	310	40	4580
	16%	1:46	0:25	274	0	4556
	No	1:19	0:00	274	0	0
s35932	3%	38:35	34:18	1406	1010	144534
	5%	21:52	17:35	247	133	68855
	7%	10:31	6:14	66	6	25207
	8%	8:40	4:23	42	0	18234
	No	4:17	0:00	28	0	0
s38417	3%	54:46	32:15	1427	960	80969
	5%	46:42	24:11	698	275	48982
	7%	39:08	16:37	527	129	31480
	10%	34:44	12:13	413	17	28721
	15%	34:28	11:57	389	0	25536
	No	22:31	0:00	389	0	0

3.7 Conclusions

In this work, we have introduced a realistic low cost delay test compaction flow that guardbands the circuit delay using a sequence of estimation metrics. Significant improvements are demonstrated over prior work using benchmark circuits. Current work targets larger designs. The veto compaction process can be also applied as a guardband for other constraints, such as test power, where similar approaches have been demonstrated [45]. Finally, this work only considers on-chip IR drop. In the future, we want to also consider off-chip *LdI/dt* effects during ATPG and compaction.

4. SUMMARY AND FUTURE WORK

In this work, we first proposed a constant test power dissipation flow that combines X-Fill, shift power estimation and test pattern reordering algorithm together. The flow is independent of the X-Fill algorithm. Our work proposed an X-Fill algorithm that minimizes both capture power and shift power using Preferred Fill [13] and Adjacent Fill. Weighted Switching Activity (WSA) was used as a power estimation metric. The shift power estimation metric was also enhanced by probabilistic analysis. The constant power was achieved using pattern reordering as we are not using X-Fill or ATPG itself to even out the power. Good experimental results prove our reordering algorithm's effectiveness and correctness. However, we still have to solve the poor estimation of shift power estimation for circuits b14 and b18. In addition, since WSA itself is an estimation of power, we need silicon results to show the real power dissipation after applying our reordered patterns.

We also proposed a realistic power supply noise-aware delay test compaction framework that has great improvement over prior work [1]. This framework used a series of thresholds to guardband the excessive delay induced by compaction and tried to minimize the unnecessary simulations which on the other hand greatly speeds up the algorithm. We also discussed how to set those thresholds to make this framework theoretical and practical. We have conducted experiments by combining the framework to both static compaction and dynamic compaction. The next step is to consider *dI/dt*

effects because on our current circuit model, we only considered IR drop. This will require more complicated circuit model.

REFERENCES

[1] J. Wang, X. Lu, W. Qiu, Z. Yue, S. Fancler, W. Shi and D. M. H. Walker, "Static Compaction of Delay Tests Considering Power Supply Noise." in *Proc*. *IEEE VLSI Test Symposium*, 2005, pp. 235-240.

[2] W. Li, S. M. Reddy and I. Pomeranz, "On Reducing Peak Current and Power During Test," in *Proc*. *IEEE Computer Society Annual Symposium on VLSI,* Tampa, FL, May 2005, pp. 156-161.

[3] X. Lin, J. Rajski, I. Pomeranz, and S. M. Reddy, "On Static Test Compaction and Test Pattern Ordering for Scan Designs", in *Proc*. *IEEE International Test Conference*, 2001, pp. 1088-1097.

[4] J. Wang, D. M. H. Walker, et al, "Power Supply Noise in Delay Testing", in *Proc*. *IEEE International Test Conference*, Charlotte, NC, Oct. 2006, pp. 1-10.

[5] P.Girard, "Low Power Testing of VLSI Circuits: Problems and Solutions", in *Proc. International Symposium on Quality Electronic Design*, 2000, pp. 173-179.

[6] S.Gerstendorfer and H. J. Wunderlich, "Minimized Power Consumption for Scan Based BIST", in *Proc*. *IEEE International Test Conference*, 1999, pp. 77-84.

[7] X. Wen, Y. Yamashitam, S. Kajihara, L. T. Wang, K. K. Saluja and K. Kinoshita, "On Low-Capture-Power Test Generaion for Scan Testing", in *Proc*. *IEEE VLSI Test Symposium*, 2005, pp. 265 – 270.

[8] W. Li, S. M. Reddy and I. Pomeranz, "On Reducing peak current and power during test," in *Proc. IEEE Computer Society Annual Symposium on VLSI*, 2005, pp. 156-161.

[9] X. Wen, Y. Yamashita, S. Morishima, S. Kajihara, L.T. Wang, K. K. Saluja and K. Kinoshita, "Low-capture-power test generation for scan-based at-speed testing," in *Proc. IEEE International Test Conference*, 2005, pp. 1019-1028.

[10] S. Sharifi, J. Jaffari, M. Hosseinabady, A. Afsali-Kusha and Z. Navabi, "Simultaneous Reduction of Dynamic and Static Power in Scan Structures", in *Proc. DATE*, 2005, pp. 846-851.

[11] R. Sankaralingam, R. R. Oruganti and N. A. Touba, "Static Compaction Techniques to Control Scan Vector Power Dissipation," in *Proc*. *IEEE VLSI Test Symposium*, Apr. 2000, pp. 34-40.

[12] V. Dabholkar, S. Chakravarty, I. Pomeranz and S. M. Reddy, "Techniques for Minimizing Power Dissipation in Scan and Combinational Circuits During Test Application", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 17, No. 12, December 1998.

[13] S. Remersaro, X. Lin, Z. Zhang, S. M. Reddy, I. Pomeranz and J. Rajski, "Preferred Fill: A Scalable Method to Reduce Capture Power for Scan Based Designs," in *Proc*. *IEEE International Test Conference*, 2006, pp. 1-10.

[14] J. Wang, D. M. H. Walker, et al, A Vector-Based Approach for Power Supply Noise Analysis in Test Compaction, in *Proc*. *IEEE International Test Conference*, 2005

[15] Z. Wang, D. M. H. Walker, "Dynamic Compaction for High Quality Delay Test", in *Proc. IEEE VLSI Test Symposium*, Apr. 2008.

[16] Z. Jiang, D. M. H. Walker, “An Efficient Algorithm for Achieving Chip-wise Constant Test Power”, in *IEEE Workshop on Defect and Data Driven Testing*, 2008

[17] Z. Jiang, D. M. H. Walker, “Enhancement Approaches for Constant Test Power Algorithm", *International Test Synthesis Workshop*, Austin, TX, 2009

[18] W. Qiu, et al, “K Longest Paths Per Gate (KLPG) Test Generation for Scan-Based Sequential Circuits,” *IEEE Int’l Test Conf*., Charlotte, NC, Oct. 2004, pp. 223-231.

[19] K. L. Shepard and V. Narayanan, “Noise in Deep Submicron Digital Design,” *IEEE Int’l Conf. Computer Aided Design*, San Jose, CA, Nov. 1996, pp. 524-531.

[20] Y.-S. Chang, S. K. Gupta and M. A. Breuer, “Analysis of Ground Bounce in Deep Sub-Micron Circuits,” *IEEE VLSI Test Symp.*, Monterey, CA, Apr. 1997, pp. 110-116.

[21] H. H. Chen and D. D. Ling, “Power Supply Noise Analysis Methodology for Deep Submicron VLSI Chip Design,” *ACM/IEEE Design Auto. Conf.*, Anaheim, CA, June 1997, pp. 638-643.

[22] Y.-M. Jiang and K.-T. Cheng, “Analysis of Performance Impact Caused by Power Supply Noise in Deep Submicron Devices,” *ACM/IEEE Design Auto. Conf.*, New Orleans, LA, June 1999, pp. 760-765.

[23] J.-J. Liou, A. Krstic, Y.-M. Jiang and K.-T. Cheng, "Modeling, Testing, and Analysis for Delay Defects and Noise Effects in Deep Submicron Devices," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 6, Jun. 2003, pp. 756-769.

[24] C. Tirumurti, S. Kundu, S. Sur-Kolay and Y.-S. Chang, "A Modeling Approach for Addressing Power Supply Switching Noise Related Failures of Integrated Circuits," *Design, Automation and Test in Europe*, Paris, France, Feb. 2004, pp. 1078-1083.

[25] S. T. Zachariah, Y.-S. Chang, S. Kundu and C. Tirumurti, "On Modeling Cross-talk Faults," *Design, Automation and Test in Europe*, Munich, Germany, Mar. 2003, pp. 490-495.

[26] S. Pant, D. Blaauw, V. Zolotov, S. Sundareswaran and R. Panda, "Vectorless Analysis of Supply Noise Induced Delay Variation," *IEEE/ACM Int'l Conf. Computer Aided Design*, San Jose, CA, Nov. 2003, pp. 184-191.

[27] A. Krstic, et al, "Pattern Generation for Delay Testing and Dynamic Timing Analysis Considering Power Supply Noise Effects," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 20, no. 3, Mar. 2003, pp. 416-425.

[28] J. Lee and M. Tehranipoor, "A Novel Test Pattern Generation Framework for Inducing Maximum Crosstalk Effects on Delay-Sensitive Paths", *IEEE Int'l Test Conf.,* Santa Clara, CA, Oct. 2008, pp.1-10.

[29] J. Ma, J. Lee and M. Tehranipoor, "Layout-Aware Pattern Generation for Maximizing Supply Noise Effects on Critical Paths", *IEEE VLSI Test Symp.*, Santa Cruz, CA, May. 2009, pp. 221-226.

[30] J. Wang, et al, "A Vector-based Approach for Power Supply Noise Analysis in Test Compaction," *IEEE Int'l Test Conf.*, Austin, TX, Nov.2005, pp. 517-526.

[31] Z. Wang and D. M. H. Walker, "Dynamic Compaction for High Quality Delay Test", *IEEE VLSI Test Symp.*, San Diego, CA, Apr. 2008, pp. 243-248.

[32] Z. Jiang, Z. Wang, J. Wang and D. M. H. Walker, "Realistic Low Cost Framework for Supply Noise-Aware Delay Test Static Compaction", *IEEE Workshop on Defect and Data Driven Testing*, Austin, TX, Nov. 2009.

[33] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi and H. Balachandran, "K Longest Paths Per Gate (KLPG) Test Generation for Scan-Based Sequential Circuits," in Proc. IEEE International Test Conference, 2004, pp. 223-231.

[34] R. Sankaralingam, R. R. Oruganti and N. A. Touba, "Static Compaction Techniques to Control Scan Vector Power Dissipation," in Proc. IEEE VLSI Test Symposium, Apr. 2000, pp. 34-40.

[35] S. R. Nassif and J. N. Kozhaya, "Fast Power Grid Simulation," *ACM/ IEEE Design Auto. Conf.*, Los Angeles, CA, Jun. 2000, pp. 156-161.

[36] H. Qian, S. R. Nassif and S. S. Sapatnekar, "Random Walk in a Supply Network," *ACM/IEEE Design Auto. Conf*, Anaheim, CA, Jun. 2003, pp. 93-98.

[37] Z. Zhu, B. Yao and C.-K. Cheng, "Power Network Analysis Using an Adaptive Algebraic Multigrid Approach," *ACM/IEEE Design Auto. Conf*, Anaheim, CA, Jun. 2003, pp. 105-108.

[38] P. Pant, J. Zelman, "Understanding Power Supply Droop during At-Speed Scan Testing", *IEEE VLSI Test Symp*., Santa Cruz, CA, May. 2009, pp. 227-232.

[39] B. Nadeau-Dostie, K. Takeshita and J. Côté, **"**Power-Aware At-Speed Scan Test Methodology for Circuits with Synchronous Clocks," *IEEE Int'l Test Conf*, Santa Clara, CA, Oct. 2008, pp. 1-10.

[40] Y.-M. Jiang, K. -T. Cheng and A. -C. Deng, "Estimation of Maximum Power Supply Noise for Deep Sub-Micron Designs," *IEEE Symp. on Low Power Electronics and Design*, Monterey, CA, Aug. 1998, pp. 233-238

[41] G. Bai, S. Bodda and I. N. Hajj, "Static Timing Analysis Including Power Supply Noise Effect on Propagation Delay in VLSI Circuits," *IEEE Design Auto. Conf.*, Las Vegas, NV, June 2001, pp. 295-300.

[42] J. Wang, et al, "Power Supply Noise in Delay Testing", *IEEE Int'l Test Conf.*, Santa Clara, CA, Oct. 2006, pp. 1-10.

[43] Z. Jiang and D. M. H. Walker, "An Efficient Algorithm to Achieve Constant Test Power", *IEEE Workshop on Defect and Data Driven Testing*, Santa Clara, CA, Oct. 2008.

[44] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash and M. Hachinger, "A Case Study of IR-Drop in Structured At-Speed Testing," *IEEE Int'l Test Conf.*, Charlotte, NC, Sept. 2003, pp. 1098-1104.

[45] Z. Jiang and D. M. H. Walker, "Enhancement Approaches for Constant Test Power Algorithm", *IEEE Int'l Test Synth. Workshop*, Austin, TX, Mar. 2009.

[46] P. Pant and J. Zelman, "Understanding Power Supply Droop During At-Speed Scan Testing", *IEEE VLSI Test Symposium*, 2009, pp. 227-232.

[47] A. Kokrady and C. Ravikumar, "Fast, Layout-Aware Validation of Test Vectors for Nanometer-Related Timing Failures," *IEEE International Conference on VLSI Design*, Bombay, India, Jan. 2004, pp. 597-602

VITA

Zhongwei Jiang

Computer Science and Engineering Dept., Texas A&M University,

College Station, TX 77840

E-mail: jzweiwei@gmail.com

Zhongwei Jiang was born in Nanjing, China. He obtained his B.S. in electrical engineering from Nanjing University of Posts and Telecommunications, Nanjing, China in July 2002, M.S. in computer engineering from Shanghai Jiao Tong University, Shanghai, China in July 2004 and a Ph.D. in computer engineering from Texas A&M University, College Station, Texas, in December 2010. He worked for HP as a software engineer for 2 years. His research interests are delay testing, automatic test pattern generation, power and supply noise analysis. He is a member of the IEEE.