Post Silicon Clock Tuning System To Mitigate The Impact Of Process Variation On Performance

POST SILICON CLOCK TUNING SYSTEM TO MITIGATE THE IMPACT OF PROCESS VARIATION ON PERFORMANCE

A Thesis Presented

by

Kelageri Nagaraj

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2010

ELECTRICAL AND COMPUTER ENGINEERING

POST SILICON CLOCK TUNING SYSTEM TO MITIGATE THE IMPACT OF PROCESS VARIATION ON PERFORMANCE

A Thesis Presented

by

Kelageri Nagaraj

Approved as to style and content by:

_____
Sandip Kundu, Chair

_____
Wayne Burleson, Member

_____
Maciej Ciesielski, Member

_____
C. V. Hollot, Department Head
Electrical and Computer Engineering

ACKNOWLEDGEMENT

I would like to thank Professor Sandip Kundu for his motivation and guidance. I would also thank Intel for partially funding this project. I am also grateful to Professor Wayne Burleson and Professor Maciej Ciesielski for their time and for being on my thesis committee. I would like to extend my gratitude to all the members of Professor Sandip Kundu's research group at UMass for their valuable suggestions on the project. Finally, I would like to thank my wife Viju and my parents who stood by me when I needed them and encouraged me for the graduate studies.

ABSTRACT

POST SILICON CLOCK TUNING SYSTEM TO MITIGATE THE IMPACT OF PROCESS VARIATION ON PERFORMANCE

FEBRUARY 2010

KELAGERI NAGARAJ

B.E., E.C.E., NATIONAL INSTITUTE OF TECHNOLOGY, KARNATAKA, INDIA

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Sandip Kundu

Optical shrink for process migration, manufacturing process variation and dynamic voltage control leads to clock skew as well as path delay variation in a manufactured chip. Since such variations are difficult to predict in pre-silicon phase, tunable clock buffers have been used in several microprocessor designs. The buffer delays are tuned to improve maximum operating clock frequency of a design. This however shifts the burden of finding tuning settings for individual clock buffers to the test process. In this project, we describe a process of using tester measurements to determine the settings of the tunable buffers for recovery of performance lost due to process variations. Then we study the impact of positioning of tunable buffers in the clock tree. In course of our study it was observed that the greatest benefit from tunable buffer placement can be derived, when the clock tree is synthesized with future tuning considerations. Accordingly, we present a clock tree synthesis procedure which offers very good mitigation against process variation, as borne out by the results. The results show that without any design intervention, an average improvement of 9% is achieved by

our tuning system. However, when the clock tree is synthesized based on static timing information with tuning buffer placement considerations, much larger performance improvement is possible. In one example, performance improved by as much as 18%.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND MOTIVATION

1.1 Introduction

As integrated circuit (IC) technologies scale to 45nm and beyond, process variations are becoming increasingly critical for circuit performance. Systematic and random variation of the process, source voltage and temperature are the major problems which result in degrading of performance of the next generation microprocessor [1]. Critical path of the circuit determines the maximum frequency of the microprocessor. Figure 1 shows that as the number of the critical paths on a die increases, within-die delay variations among the critical paths cause both the standard deviation and the mean of the die frequency distribution to become smaller resulting in degradation of the performance of the circuit on the die [2].

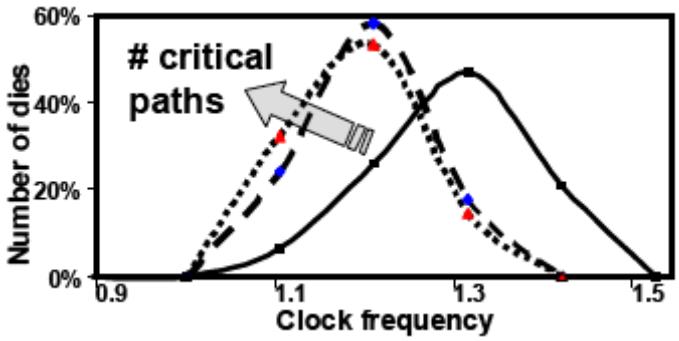


Figure 1: Die to Die critical path distribution [1]

A number of pre-silicon approaches based on statistical design optimization techniques have been proposed to mitigate the impact of process variation on the circuit performance. These approaches optimize the gate sizes and threshold voltage assignment to maximize the yield [1][2][3][4][5][6][7]. The basic principle behind these approaches

is to use statistical models to maximize the number of chips that will meet power and delay constraints in fabricated circuits. Regardless of how the design is optimized, it is inevitable that for some population of chips, the post-silicon performance will fall short of the expectation.

To address this problem, insertion of post-silicon tunable buffers have been proposed [8][9][10]. By controlling tuning inputs, the delay of these buffers can be varied to compensate for process variations, with the aim to maximize the circuit frequency. Typically, such buffers are inserted in the clock distribution network. By tuning the PST buffers (i) clock skew can be compensated and (ii) beneficial clock skew may be introduced to make a design run faster. While the first case is well known, the second case is also of equal importance and has been explained in section 2.1 with an example.

1.2 Motivation

The greatest challenge in post silicon tuning of clocks is not in design but in determining what tuning settings should be applied for maximizing performance. The answer to this question is also tied to the specific problem being addressed in this thesis.

Often a design is subjected to incremental optical shrink to increase its performance. Re-spinning the entire design to optimize its performance with respect to the new process introduces unacceptable delays. While without any design modification, the performance may fall short of expectation due to changes in clock skews. In such cases, the tuning settings may be determined once and applied to all dies. In this case, algorithmic determination of tuning settings is not very important as a trial and error approach may be used to find the best setting. The cost of trial and error approach is amortized over thousands to millions of dies and can be easily justified.

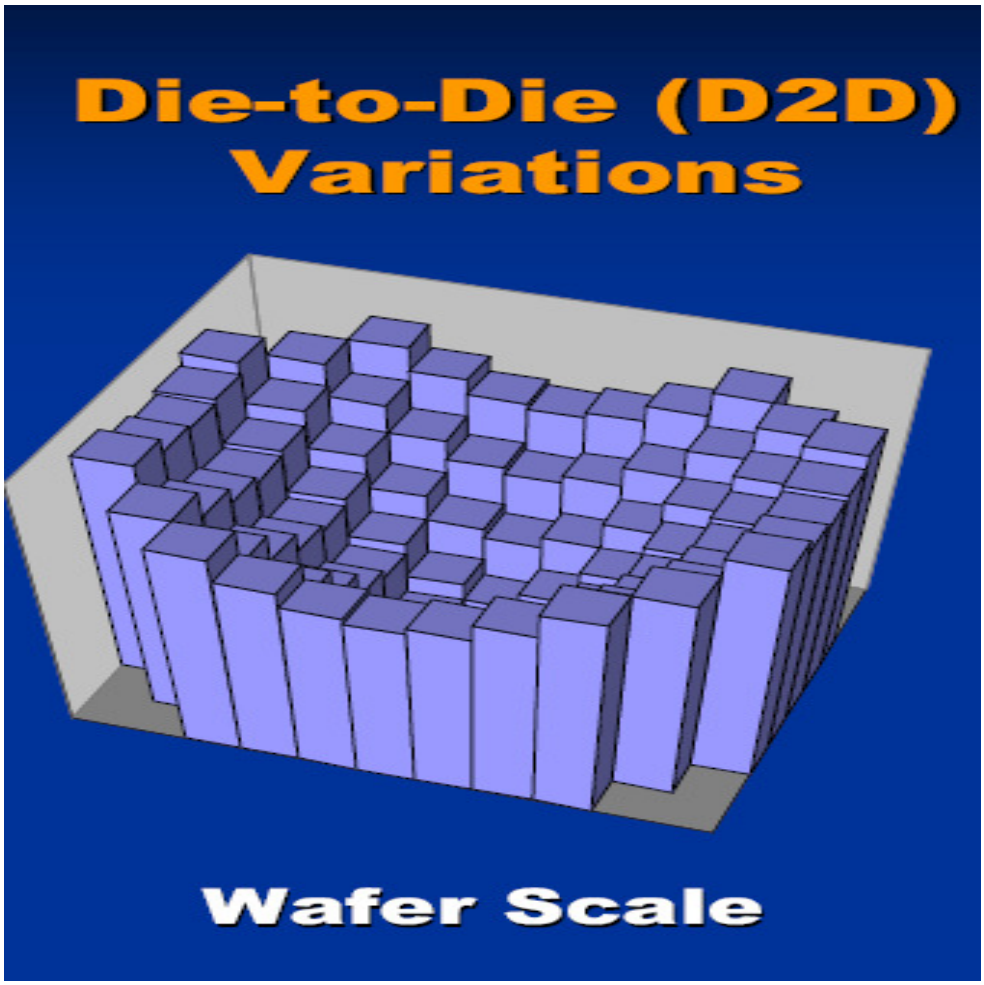


Figure 2: Inter die delay variations (Source: Intel)

Process variation affects different dies differently as shown in Figure 2. Accordingly, each die should have its unique tuning setting to maximize its performance. The solution is complicated by the fact that internal circuit delays are not known at the tester. The only information known to the observer are the input-output responses obtained from a tester. Thus, the data from the tester needs to be interpreted algorithmically to output tuning information. This is the subject of this project.

In this thesis, we describe a scheme that selects appropriate inputs through ATPG (Automatic Test Pattern Generation) process. The inputs are determined statically and

once during the design process. Adaptive test generation is expensive on tester time. This is why they are best avoided. The inference process for translating frequency failure data to tuning setting is kept simple. This means, detailed analysis of the internal circuit of the chip has to be avoided as much as possible. So our scheduling system just takes the externally manifested data as inputs to generate the list of candidates for tuning. The inputs to the scheduling system consist of a set of patterns and the output failure list generated by the tester. High performance microprocessors are *binned* to their operating frequency. Down binning a part results in a cost penalty. Thus, a binning-yield loss may be defined. By generating optimal tuning setting the goal of my thesis will be to minimize the binning yield loss.

1.3 Previous Work

In nano-CMOS designs, process variations result in a statistical spread in the achievable frequency, thereby causing some chips to fail from meeting the nominal target frequency. Borkar *et. al* [1] have suggested that as much as 30% frequency variation can be observed in high-performance microprocessors.

A lot of recent work focuses on statistical techniques for considering process variability during analysis and optimization. Statistical timing analysis has been used as a tool to predict the timing distribution of the designs [11][12]. Many approaches have been tried to utilize this information to perform statistical optimizations like gate sizing [1][4][5][6]. These techniques are used in the pre-silicon phase to counter variability.

Post-silicon tuning is another approach to reduce binning yield loss in circuits. This would allow a manufacturer to tune each chip to improve frequency. Recently, post-

silicon tunable (PST) clock-tree synthesis [8][9][10][13] has been proposed as one such approach that can be applied to high performance designs to correct timing violations.

Most of the work in this area focuses on design optimization. Chen *et. al* have proposed a timing driven post silicon tunable clock tree synthesis [14][15]. Srivastava *et. al* [16] have proposed a simultaneous gate sizing and PST buffer range determination using statistical timing. These approaches rely on available timing information and need delay models. E. G. Friedmen *et. al* [17] have proposed a methodology to take care of the effects of process variation on the clock distribution network by calculating the clock skew range of local data path, and using this skew to reduce the clock period. Fishburn [18] has proposed optimization of clock skew using linear programming. But these schemes do not take care of the PST buffer tuning.

By contrast, this thesis work is focused on finding appropriate settings for tuning elements based on information obtained from the tester. These settings increase the performance of the design and alleviate the effects of process variation. Our scheme does not rely on internal delays because such information is not easy to obtain from a chip without extensive and costly instrumentation.

The rest of the thesis is organized as follows: in chapter 2 the fundamental concept of clock tuning is explained. Chapter 3 talks about the critical path tracing concepts. Chapter 4 shows our methodology and first phase of results. Chapter 5 deals with the process variation and tuning system driven clock tree synthesis and illustrate the corresponding results using the stuck at and transition fault models. Chapter 6 explains the technique to generate clock tree structure based on static timing analysis. Chapter 7 describes the flow to generate the path delay test patterns which uses the critical path

information generated by the static timing analysis. Chapter 8 shows the results after incorporating the static timing based clock tree synthesis and path delay test model in our PST clock buffer tuning system.

CHAPTER 2

FUNDAMENTAL CONCEPT

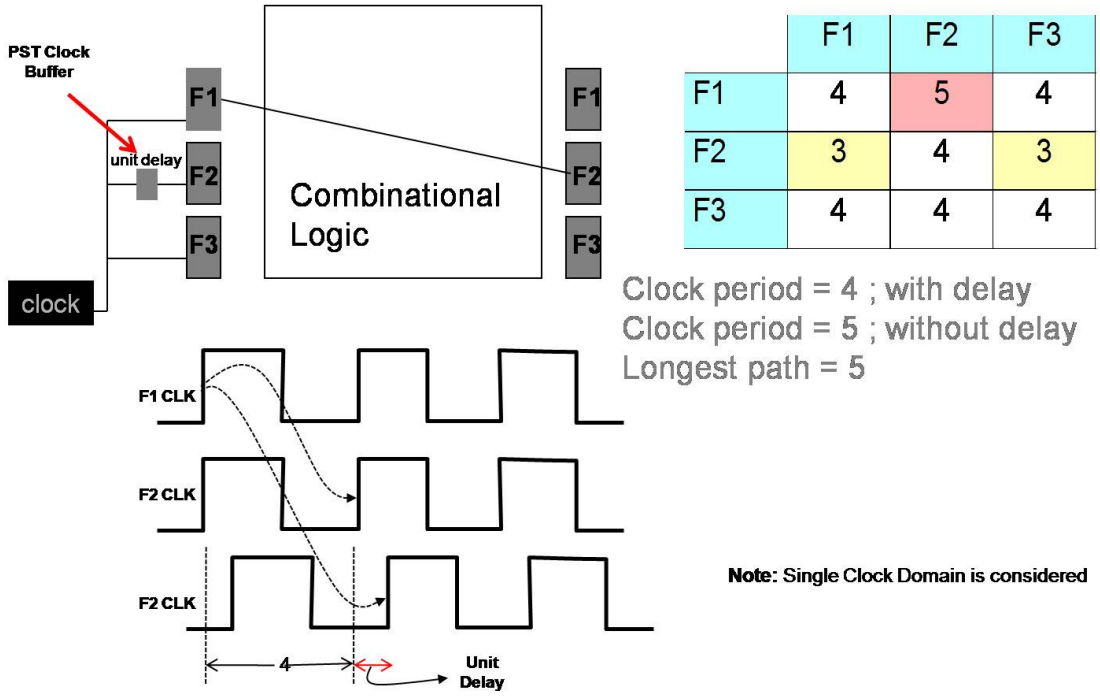
2.1 Background

A sequential design consists of a set of flip-flops and combinational logic between the flip-flops. Let us represent this with a graph G = (V, E), where V is the set of flip-flops and E is the set of edges representing the combinational logic between the flip-flops. Let *i* and *j* be two flip-flops and *cij* be the edge representing the combinational logic between them. Let *D(cij)* be the maximum delay of the combinational logic *cij*. Let *Ti* and *Tj* be the clock arrival time at *i* and *j*. In order to avoid timing violations, the flip-flop *j* should capture the data which was launched by the flip-flop *i*. For this to happen, following equation has to be satisfied.

$$Ti + D(cij) < Tclk + Tj - Tsetup \quad (1)$$

Where, *Tclk* is the clock period, *Tsetup* is the setup time of the flip-flop. PST buffers can control the clock arrival times *Ti* and *Tj*. Equation 1 can be satisfied either by reducing *Ti* or increasing *Tj*. This means that the PST buffers can be used to introduce beneficial clock skew to satisfy timing constraints. In other words, the timing slack is adjusted between the critical and non-critical paths. Similarly the hold time constraints can also be satisfied by using the PST buffers.

Example: It is well known that clock skew can degrade performance. However, clock skew can be beneficial as well. In this example, we illustrate a situation when clock skew can be beneficial.



	F1	F2	F3
F1	4	5	4
F2	3	4	3
F3	4	4	4

Figure 3: Flip-flops and combinational logic with flop to flop delay

In Figure 3, the delay from the flip-flop F1 to flip-flop F2 is 5 units. So the min clock period has to be 5 units for this circuit to work. The minimum clock period is reduced by adding a unit delay skew to the clock to the flip-flop F2. Under this condition the minimum clock period may be reduced to 4 units.

As this example illustrates, the clock tuning mechanism may be used to compensate for variation in path delays. Thus the power of this technique stretches beyond compensating for clock skews. In fact, the objective of this thesis is to tune the clock in a way that maximizes circuit performance and without any distinction as to which variation is being compensated.

2.2 Post Silicon Clock Tunable Buffers

The impact of process variation on the circuit performance is becoming serious in the nanometer designs. Existing successful timing methodology of reserving timing margins will not be beneficial to the timing yield in the nanometer designs as the timing uncertainties become larger. Reducing the impact of process variation on the circuit performance can enhance the timing yield. Use of post silicon clock tunable buffers in the clock tree is one of the suggested remedies. By inserting the PST buffers in the clock tree and redistributing the timing slacks in the adjacent timing paths, the cycle time goal of the circuit can be achieved. This is the crux of the Post Silicon Tuning System.

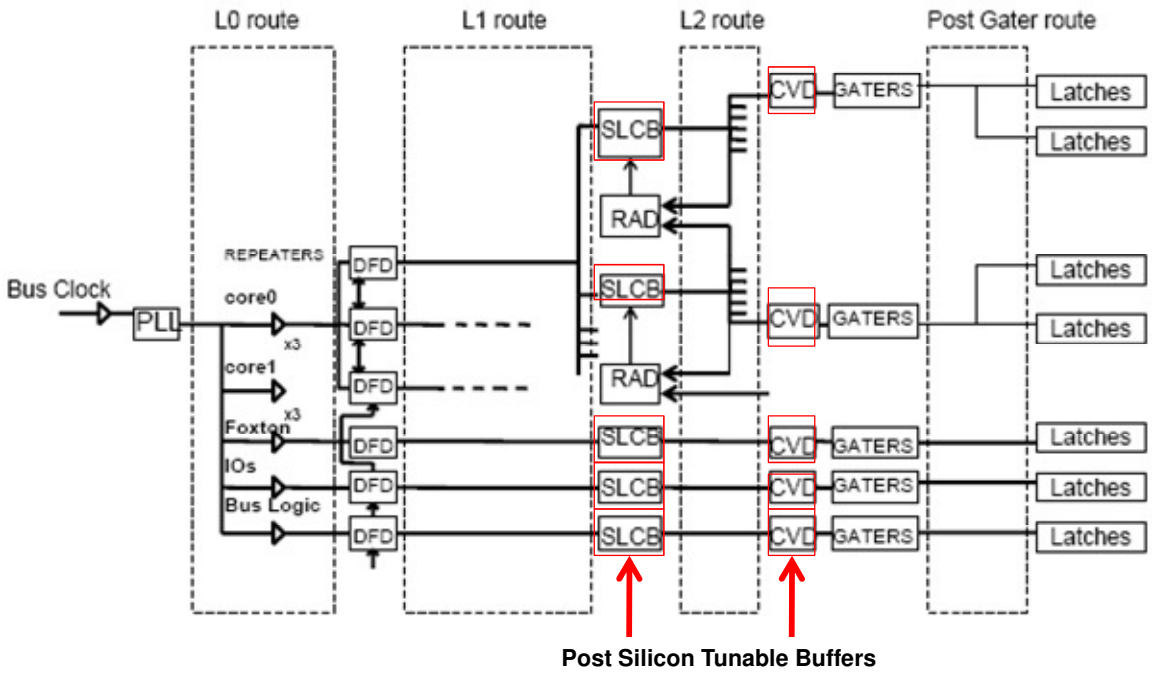


Figure 4: Clock distribution network of a dual-core Intel Itanium processor [19]

As shown in Figure 4, the clock distribution network of Intel's dual-core Itanium processor uses two levels of PST clock buffers to counter clock skews caused by process

variations and improve the timing yield [14]. The tunable Second Level Clock Buffers (SLCBs) at the terminals of L1 route can be dynamically adjusted by on-chip clock phase detection hardware to cancel clock skew variations. They can also be programmed from the Test Access Port (TAP) for timing optimization. There is also a second level of PST clock buffers at every terminal of the L2 route.

2.3 Post Silicon Clock Tunable Buffer Designs

There are several ways of implementing the PST buffers. Below we have explained two designs of PST buffers.

1) Figure 5 shows the design of a programmable delay line. The design consist of a 20 bit delay control register which can be controlled through a Test Access Port (TAP) Interface and banks of 20 passive loads which are connected to the intermediate nodes. The control bits in the 20 bit registers control the number of passive loads that gets connected in the intermediate nodes of the delay chain. The TAP interface is used to read the values from the control register and also to overwrite new values into the control register [20].

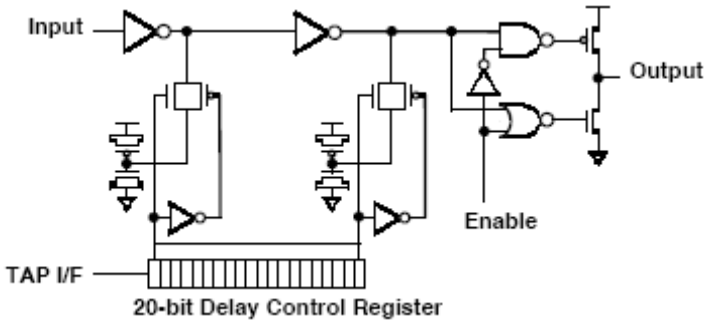


Figure 5: Post silicon clock tunable buffer [18]

2) Figure 6 shows the design of a programmable delay element design. The passive loads are in the form of transmission gates and the control signals choose the transmission gates which will appear in the clock delay line [15].

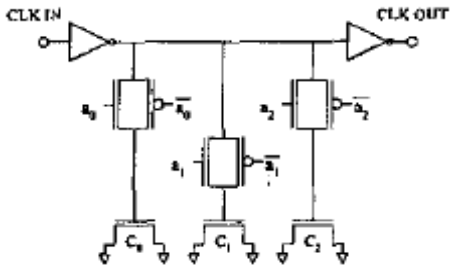


Figure 6: Post silicon clock tunable buffer [14]

2.4 Basic Testing Process to generate the tuning settings

We assume that the circuit under test is tested by scan patterns. In the test mode, a pattern is scanned into the device under test, following which a system clock is applied to launch the transition test. A second system clock captures the result into flip-flops, which is then scanned out. The basic test application method is based on starting with a relaxed system clock cycle with successive application of the same test pattern, using progressively shorter and shorter system clock periods until the circuit fails. This is repeated for all test patterns. When a pattern fails, corresponding clock period and failing outputs are noted. This is subjected to software analysis for targeting which clock buffers should be tuned and by how much.

It should be noted that this testing process is separate from ordinary pass/fail tests that are run for the manufacturing process. In this thesis work, it is assumed that the test is free from manufacturing defects and works fine at slow frequencies. Our objective is to improve bin yield at a higher frequency.

Initially, when a pattern p_i is applied, it is tested with a slow system clock, well beyond what the chip is rated for. This is used to capture the golden response for the pattern. Then the same pattern is applied repeatedly with progressively shorter periods until the output deviates from the golden response. It is not expected that when the chip fails, all output bits are faulty. In fact, only a few bits are expected to fail first. The failing outputs, together with actual inputs applied, produce useful information that help identify the clocks which need to be tuned to improve the performance. The analysis approach we use is based on critical path tracing which is described in the next chapter.

2.5 Summary

Takahashi et al. [10] propose a post-silicon clock timing adjustment method that adjusts the clock arrival times to compensate path delay variations. However, there is no systematic way to determine the tuning settings for the PST buffers in the clock trees [15]. This thesis tackles the problem of lack of systematic tuning system by providing a systematic way of generating the tuning settings for the PST buffers based on the Critical Path Tracing (CPT) concept which is explained in detail in the next chapter.

CHAPTER 3

TUNING SYSTEM USING CRITICAL PATH TRACING

3.1 Critical Path Tracing

Critical Path Tracing (CPT) was originally described by Abramovici *et. al*. [21] which was then applied for delay fault diagnosis by Girard *et. al* [22]. The basic critical path tracing algorithm uses six-valued algebra as shown in Figure 7. For two successive stimulus, the symbol S0 (S1) indicate that the signals stay glitch free at logic 0 (1), T0 (T1) indicates that the signal transitions from 1(0) to 0(1) while P1 (P2) indicates a signal that starts at 0(1) and stabilizes at 0(1), while it may glitch during intermediate periods.

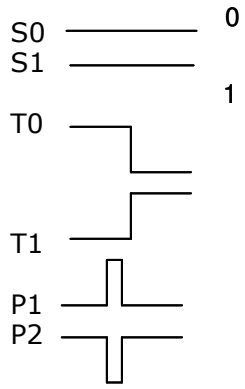


Figure 7: Six-valued algebra

CPT begins with the simulation of the failing patterns on combinational logic portion of the circuit based on 6-valued logic described above. Subsequently, a backward trace is performed from the failing outputs to identify the potential inputs that might have

contributed to the failure. Following are the rules for traversing backward through sensitizing gates.

1) If only one of the inputs has Dominant Logic Value (DLV) then that input is sensitive. For e.g. DLV for an AND or NAND gate is 0. So in six-valued algebra, transitioning to 0 (T0) will be the DLV. Similarly T1 will DLV for OR and NOR gates.

2) If all the inputs are non DLV then all the inputs are sensitive.

3) If all of the inputs have DLV then none of the inputs are sensitive.

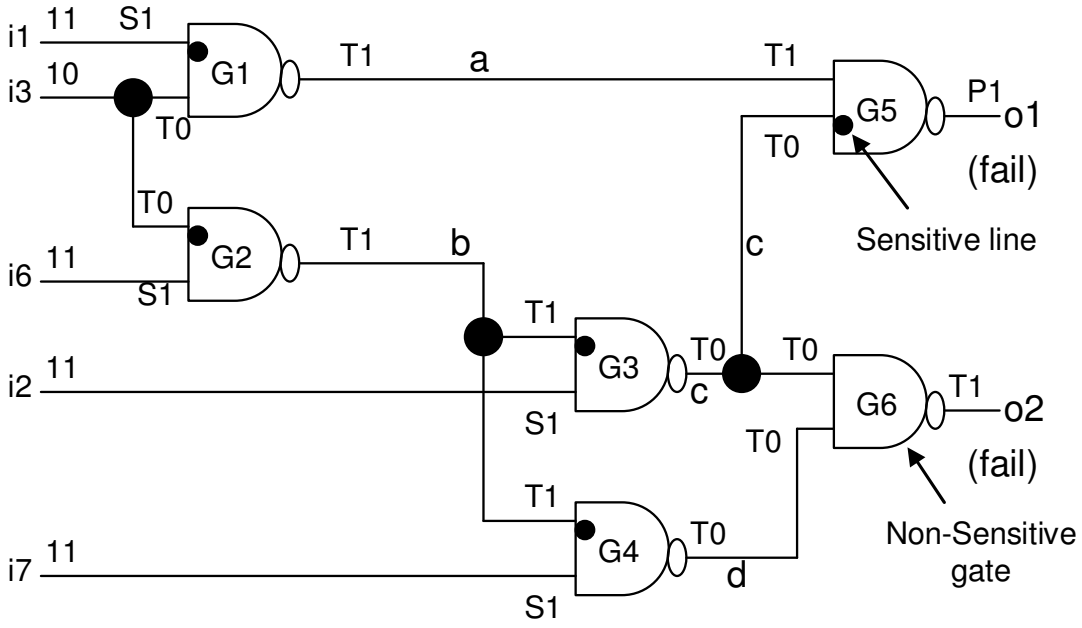


Figure 8: Critical Path Tracing (CPT) using sensitized gates

In Figure 8, we explain the process of Critical Path Tracing on C17 circuit from the ISCAS-85 benchmark circuit suite.

As shown in Figure 8, outputs o1 and o2 fail in the tester under input values displayed in the circuit. Our goal is to identify the critical inputs which control these two

outputs and generate a tuning setting which may potentially correct the failure at these two outputs. As shown in Figure 8, the input transitions from 11111 to 10111. For these two transitioning patterns, fault free simulation is done starting from the primary input and ending at the primary outputs. During the forward simulation process, sensitization rules are applied to mark the gates with sensitive inputs (marked with black dots).

The critical path tracing starts backwards from the two failed outputs o1 and o2. Tracing continues backward through the sensitive inputs of the gates until a primary input is reached. In Figure 8, for the output o1, CPT starts tracing back from the gate G5 and finds that the input c is sensitive and it traces to the gate G3. For G3 input b is sensitive and it traces to the gate G2. For G2, the input i3 is sensitive and since i3 is the primary input, the CPT has found the critical input for the output o1. So "c-b-i3" is the critical path for the output o1. Next, the CPT starts tracing from 2nd failed output o2. It starts back-tracing from the gate G6 and finds that G6 is a Non-sensitive gate as it does not have any sensitive inputs. This is a case of multiple path sensitization, which is an effect of re-convergent fan-out [21]. In such cases we use the concept of labeling proposed by Girard et al. [22]. The basic principle behind the labeling is to make sure that a fault at a stem can be propagated to the output of the re-convergent gate. Every input value having T0 (DLV for NAND) is assigned a label.

$$La(c) = \{(G6,1)\} \text{ and } La(d) = \{ (G6,2)\}$$

where, c and d are the input lines to the non-sensitive gate, G6. Next, the label (G6,1) is propagated through gate G3 and (G6,2) is propagated through gate G4. Since line b is the sensitive input for G3 and G4 and also a fan-out stem, its label is the union of $La\{c\}$ and $La\{d\}$.

$$La\{b\} = La\{c\} \cup La\{d\} = \{ (G6,1), (G6,2) \}$$

The labels generated by G6 are compared with each element of *La{b}* and if they are found equal, then stem b is considered to be the fan-out stem for the gate G6. Back tracing continues from stem b and propagates through G2. For G2, primary input i3 is the critical input and as explained earlier i3 becomes the critical input for the output o2 as well. So for the failed outputs o1 and o2, i3 is a critical input. Thus the delay at the outputs o1 and o2 can be controlled by the input i3.

If the inputs i1 through i5 are driven from flip-flops, while outputs o1 and o2 are driving the flip-flops, potential solutions we consider for improving performance are clock i3 early or clock o1 and o2 late. The flip-flop selection heuristics are described in the next section.

3.2 Tuning Target Selection

For the purpose of explaining the flip-flop selection heuristic, we assume that all flip-flops are tunable. This is of-course not the case in general. However, unless we can solve this problem, it is more difficult to solve the general problem. So we describe this solution briefly in the next section and in detail in chapter 4.

Once the CPT determines the critical inputs for all the patterns and all the outputs which failed at the tester, our tunings system evaluates these critical inputs and generates a tuning list consisting of flip-flops. By tuning the flip-flops in the list, we can see an improvement in the performance of the circuits. Table I shows a simple example to illustrate our tuning system.

Patterns	Early Tuning Candidates	Late Tuning Candidates
Pattern 1	FF1 early	FF3 late
Pattern 2	FF1early,FF2 early	FF4 late

Table I: Potential Early and Late tuning candidates

Figure 9 shows a circuit representation of the tuning setting for pattern 1. FF1 early means that PST buffer in the clock path to flip-flop 1 (FF1) has to be tuned such that the clock arrives early at FF1. Similarly FF3 late means the clock has to arrive late at flip-flop 3 (FF3).

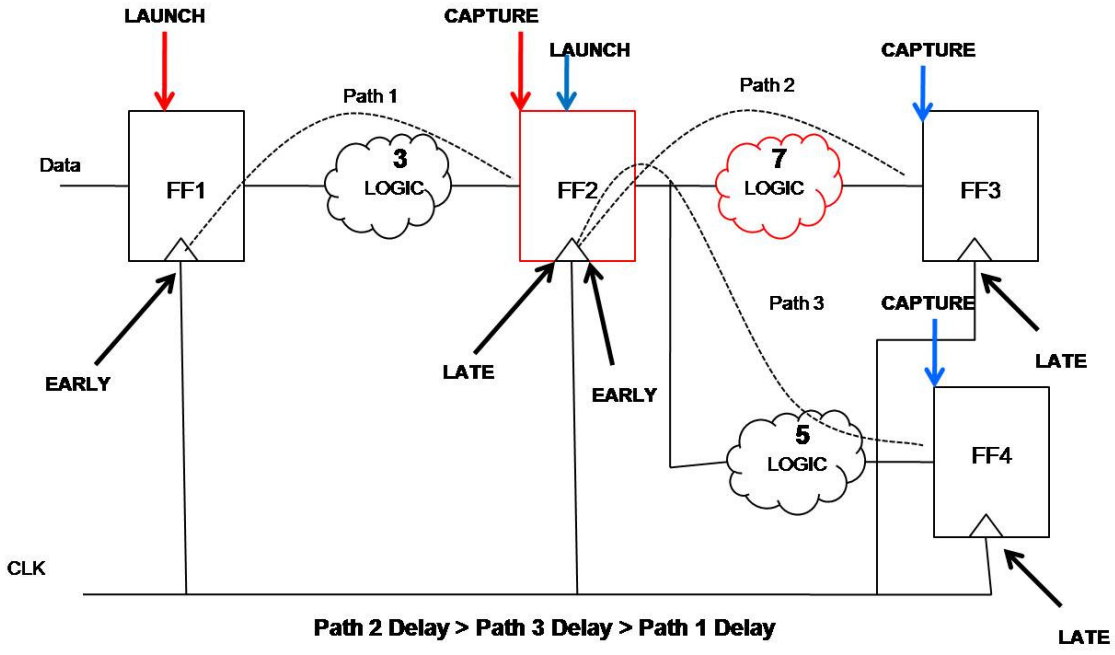


Figure 9: Early and Late Flip-Flop

At this time, we observe that tuning decisions are based on a small number of inputs. To determine the actual performance of the circuit, we apply a large number of patterns to identify the performance. This can be done on a tester. As long as the

performance gain is positive, the goal of frequency bin improvement is met. The details of selection of the list of flip flops are explained in detail in the chapter 4.

Next we describe the ATPG methodology for generating a small but effective set of patterns for tuning target selection.

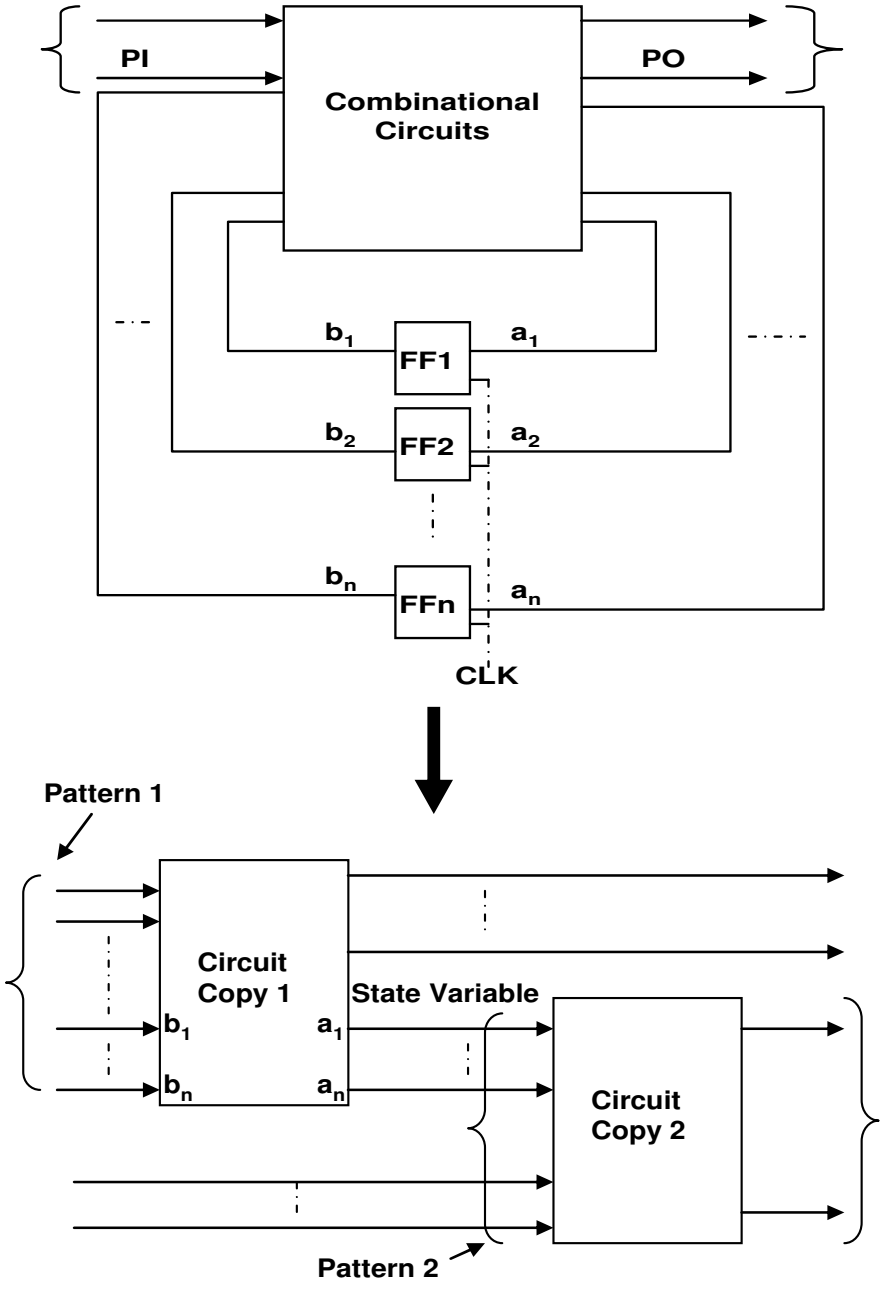


Figure 10: Iterative Logic Array Model

3.3 Critical Path Tracing for Sequential Circuits

The CPT explained above is good for the combinational circuits. To implement the CPT on sequential circuit, a concept known as Iterative Logic Array (ILA) is used which is explained below [23][13].

The set of transition patterns we apply should have the following characteristics:

- It must sensitize each input to an output, so that the effect of every input can be observed at the output.
- Every input should be sensitized for both values 0 and 1.
- Transitions must be generated by system clock.

To facilitate transition pattern generation with the above characteristics, we unroll the circuit as an Iterative Logic Array (ILA) [23][13]. This is illustrated in Figure 10. We make two copies of the combinational circuit because there are two time frames. The state variables which connect the 1^{st} copy of the circuit to the 2^{nd} copy are replaced by simple lines. The input to the flip-flops $a_1,a_2….a_n$ become the pseudo-output of the first copy and the respective flip-flop outputs $b_1,b_2…b_n$ become the pseudo-inputs. The pseudo-inputs will represent the present state and the pseudo outputs represent the next state [23][13].

Stuck-at faults are considered in the lines that replace the state elements between two copies of the circuit. A commercial ATPG is used for the pattern generation under input constraints in copy 1. The generated patterns satisfy the following properties:

- When the state inputs to the 1st copy of the circuit are scanned in and a system clock is applied, a transition pattern is generated at the stuck-at fault site
- Inputs to the 1st copy of the circuit become the initializing pattern and the input to the 2nd circuit copy which consists of the state variables and the primary input becomes the 2nd pattern of the ordered pair of the transition fault patterns.
- The output of the 2nd circuit copy becomes the expected outputs of the transition fault patterns.

CHAPTER 4

METHODOLOGY AND RESULTS

4.1 Experimental Setup

We use the experimental setup explained in [13]. Simulation experiments were conducted on ISCAS-89 benchmark circuits for seeking performance improvement opportunities. The experiments are double blind, where a commercial Verilog simulator is used in place of a tester, while the tuning target selection methodology uses the tester but has no access to the internals.

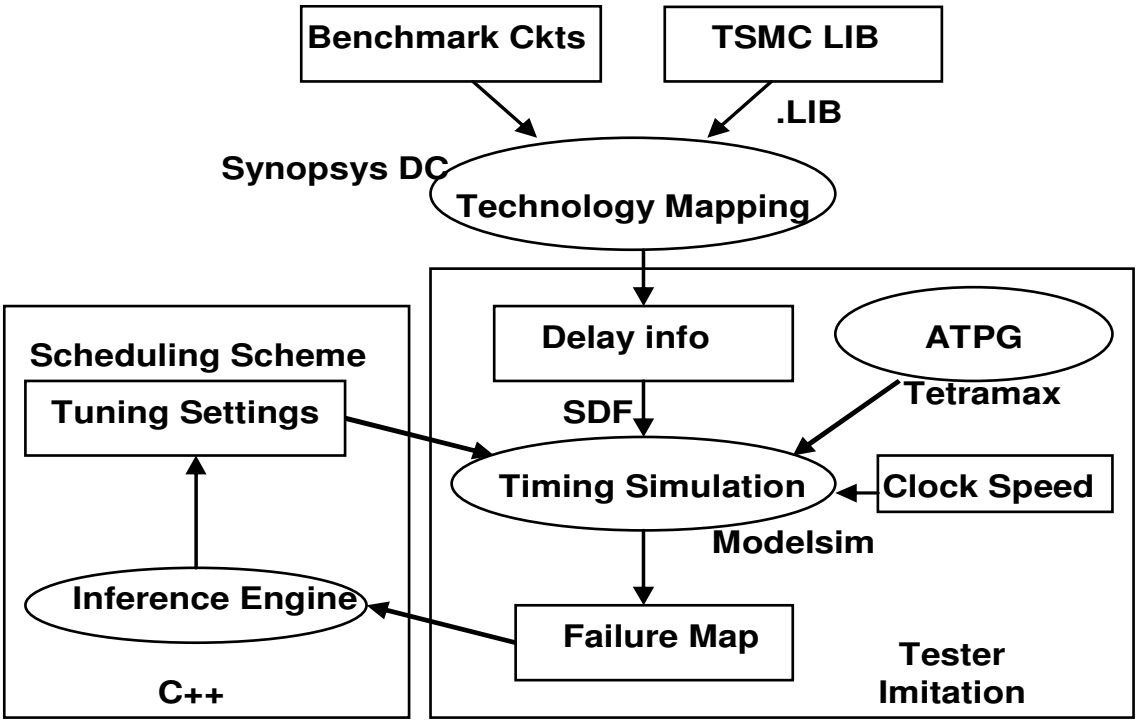


Figure 11: Clock Tuning Methodology

All target circuits were converted to scan circuits. It is assumed that every flip-flop is tunable. Tester failures are generated from Verilog simulation. Our entire methodology is shown in Figure 11. Each step in the flow is briefly explained in the

following sections. It should be noted here that we still haven't considered the process variation in benchmark circuits. Details of the circuits with process variation are explained in chapter 5. In this experimental setup, our goal is to use the CPT to generate the tuning list of flip flops for a circuit with no process variation and show that CPT shows performance benefit.

4.2 Synthesis

We used Synopsys™ Design Compiler to synthesize and map the benchmark circuits to the .25 micron TSMC technology. Design Compiler generates structural Verilog after synthesis and the delay information is extracted in the Standard Delay Format (SDF). (Note: We do not use this delay info in our tuning scheme. This is just used in the timing verification when we are imitating the tester environment). The structural Verilog and the SDF are used during the Verilog timing simulation.

4.3 ATPG Model and Approach

The circuit under test is unfolded into an ILA as described in the section 3.3. The transition fault patterns are generated using Synopsys™ Tetramax. The patterns generated are used in the Verilog timing verification to check the circuit for the timing uncertainties.

4.4 Verilog Timing Verification

We do Verilog timing verification to generate a list of output failures which fail to meet the targeted clock period or frequency. Figure 12 shows the clock timing waveform. At time t_0, the initializing pattern 1 is applied and at time t_2, the transition pattern 2 is

launched. The initial time period t_2-t_1 can be large to ensure that the circuit obtains the correct state. However, the time period t_4-t_2 has to be at the rated clock period at which the circuit is supposed to work. At time t_4, when the clock goes high, the functional data is captured and is compared against the expected output XPCT. If the captured primary output is same as the expected output XPCT, then the circuit is working at rated clock period for that pattern. If captured data is not equal to XPCT then there is a timing failure. The corresponding pattern and the failing primary outputs are added to the failed list. This process is repeated for all the ordered pair of the transition patterns. Above process is equivalent to applying patterns from a tester and taking a tester dump of the failed data. Following is an explanation of the above process with an example.

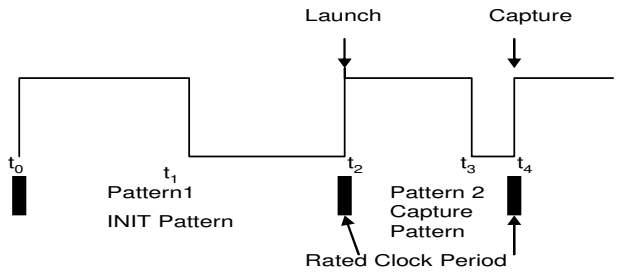


Figure 12: Timing Verification Clock Waveform

- Suppose after applying 10 test patterns, the corresponding minimum clock cycle times (in ps) are:

  700, 700, 626, 646, 674, 666, 630, 634, 700, 700

- We set a target clock period of under 640 ps. So the failure maps (failed pattern and the corresponding failed outputs) generated from 700, 700, 646, 674, 666, 700, 700 are used to generate the tuning setting for the PST buffers.

We have used Modelsim™ as the Verilog timing simulator and Standard Delay Format (SDF) for annotating delays to the gates.

4.5 Post Silicon Tuning System

As explained before, post silicon tuning is used to correct timing problems of the manufactured chips. Figure 13 shows the first part of the scheduling algorithm which takes failed patterns and failed output as the input. For all the failed patterns, it scans the circuit from primary input and sensitizes all the gates depending on the input values. Once the gates are sensitized, the Critical Path Tracing (CPT) function starts tracing back from the failed output. Using the critical path tracing algorithm, the critical inputs which are responsible for controlling the failed output are determined. The CPT Algorithm is explained in detail in chapter 3. Once the critical inputs are detected, the flip-flops connected to these inputs are added to a list called Early_list. Early_List consists of flip-flops which are the potential candidate for early tuning. By early tuning we mean that the clock will arrive early at this flip-flop.

Similarly the flip-flops connected to the failed output is added to the list called Late_list, which consist of the potential candidate for late tuning i.e. whose clocks have to be delayed. If a flip-flop is a potential candidate for both late and early tuning, then that flip-flop is not considered as a potential tuning candidate and it is removed from the late and early list, indicated by the remove_FF_late_early in the algorithm.

Scheduling algorithm :1
Input: Failed Patterns, failed output
Output: Early and Late Flip-Flop lists

```
1   foreach {failed pattern}
2   {
3       Sensitize_gate(); // Sensitize each gate with critical inputs
4       foreach {failed output}
5           {
6               Critical_input=CPT (PO);
7               Early_list = FF_of_critical_input;
8               Late_list = FF_of_failed_output;
9               if {FF_in_early==FF_in_late}
10                  {
11                   remove_FF_late_early();
12                  }
13          }
14  }
15   CPT (PO)
16    {
17      // if backtracing reaches primary input
18      // then that input is critical
19        If ( BackTrace(PO) == primary input)
20        return primary_input;
21    }
```

Figure 13: Scheduling Algorithm 1

Table II shows the result of the first part of the tuning system which generates the early and the late lists of flip-flops. These flip-flops are the potential candidates for tuning. Table II shows the tuning lists generated for 7 patterns. Ideally, taking the intersection of the tuning lists of all the patterns will give us a non empty intersection set, consisting of flip-flops. But in reality, we have observed that the probability of getting a non-empty intersection is very small and depends on the targeted clock period and the number of patterns which fail to achieve the targeted performance. Table II portrays a more realistic picture of what is actually observed.

Pattern	Potential Early Tuning Candidates	Potential Late Tuning Candidates
1	FF1,FF2,FF3	FF10,FF11
2	FF1,FF3	FF12,FF13
3	FF6,FF5	FF10,FF12
4	FF1,FF8	FF13,FF14,FF15
5	FF1,FF9	FF16,FF17
6	FF2,FF3,FF5	FF11,FF13
7	FF6,FF8	FF7,FF4

Table II: Early and Late Flip-Flop list

Scheduling algorithm :2
Input: early and late FF list
Output: Tuning list

```
1   Sort_list(FF_Early); // sort the early FF list
2   Sort_list(FF_Late); // sort the late FF list
3   Foreach FF in Early_list
4   {
5     Freq = Count_Repetition; //Count the repetition of FF in the list
6     object_early(FF).Freq=Freq;
7    }
8    Sort object_early(FF); //Sort the FF list with freq as key
9        Foreach FF in Late_list
10       {
11    //Count the repetition of FF in the list
12          Freq = Count_Repetition;
13          object_late(FF).Freq=Freq;
14         }
15   Sort object_late(FF); //Sort the FF list with freq as key
16  Tuning_list = FF with highest frequency;
```

Figure 14: Scheduling Algorithm 2

To solve the above dilemma, we assign weights to each potential tunable flip-flop depending on the frequency of their occurrence in the potential tunable list across patterns. We sort the list of potential tunable candidates with respect to their weights in descending order. The flip-flops at the top of the list will be used for tuning.

In Figure 14, the pseudo-code for assigning the weights to the potential tunable flip-flops is shown. Since we have used object oriented programming, the early and the late lists are stored in an array of objects. One of the member elements of this object is the frequency of occurrence of that FF in the failed patterns. The array of objects is sorted with respect to the frequency in descending order.

The scheduling scheme will generate a tuning list which will consist of the flip-flops picked from the top of the list. Table III shows a sorted list of weighted tuning list. The early tuning of flip-flop 1 (FF1) is the potential candidate in four patterns. Similarly early tuning of FF3 appears in three patterns. So suggested tuning list will be

1) Early Clock tuning for the flip-flop 1 (FF1)
2) Early Clock tuning for the flip-flop 3 (FF3)

Early	Freq	Late	Freq
FF1	4	FF10	2
FF3	3	FF11	2
FF2	2	FF12	2
…	…..	….	….

Table III: Final weighted list for tuning

We have used top 3 entries as the tuning list. But it can be varied depending on the design size and the patterns which fail. Once we generate the list, we repeat the timing verification process and verify that we are meeting the targeted performance. If we

meet the targeted performance level then the chip is good to go to the market. If not, then we iterate again through the scheduling scheme and generate another set of tuning list.

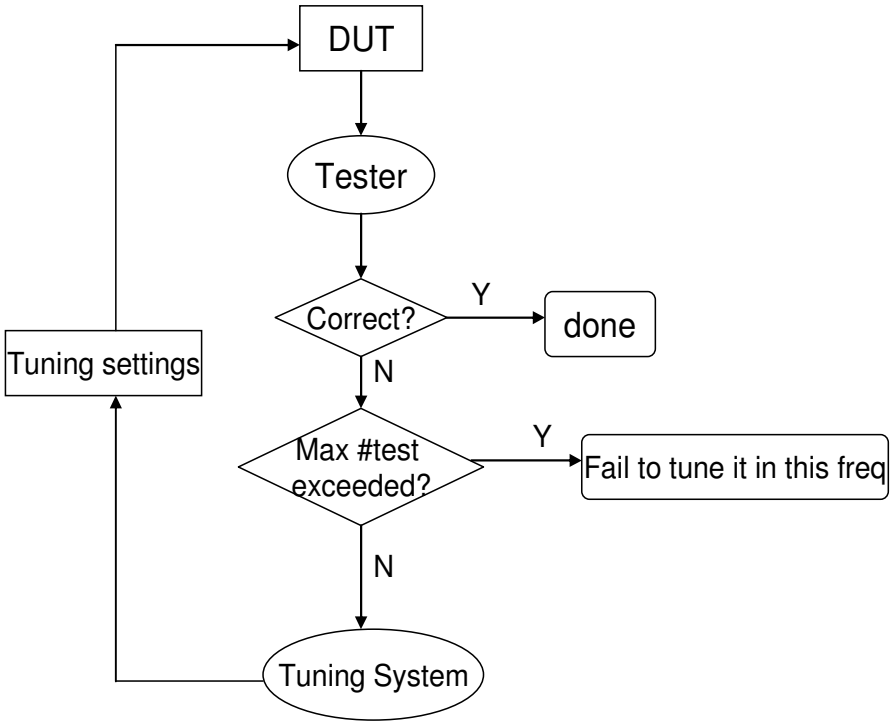


Figure 15: Iterative Test Flow

This process is iterated until it meets the max number of tests. If the scheduling scheme cannot improve performance within a predetermined maximum number of iterations then the chip tuning has failed. The flow chart shown in Figure 15 captures the flow, described so far.

4.6 Experimental Results

Our tuning system was implemented in C++ and it was tested on Pentium 4 machine with Linux as operating system. We have used ISCAS 89 benchmark circuits to test our tuning system. We synthesize the benchmark circuits using Synopsys™ Design

Compiler (DC) and generate the synthesized structural Verilog. We create an iterative array logic circuit and use Synopsys™ Tetramax to generate the transition fault patterns. We then run timing verification on the structural verilog with ATPG patterns as inputs to these circuits. The Standard Delay Format (SDF) file generated by Synopsys™ DC is used for delay annotation. The timing verification is done for the targeted clock frequency. The failed patterns and the corresponding failed outputs are taken through our tuning system, which generates the tuning list consisting of flip-flops. These flip-flops will have to be tuned to improve the performance. Using our tuning system we were able to see a performance gain in the range of 10% to 3% for ATPG patterns, as shown in Figure 16. Figure 17 shows the performance gain chart for the ATPG patterns in terms of the clock periods. In order to validate our result, we also did random pattern simulation and we saw a performance improvement in the range of 10% to 0.7%. The results are shown in Table IV.

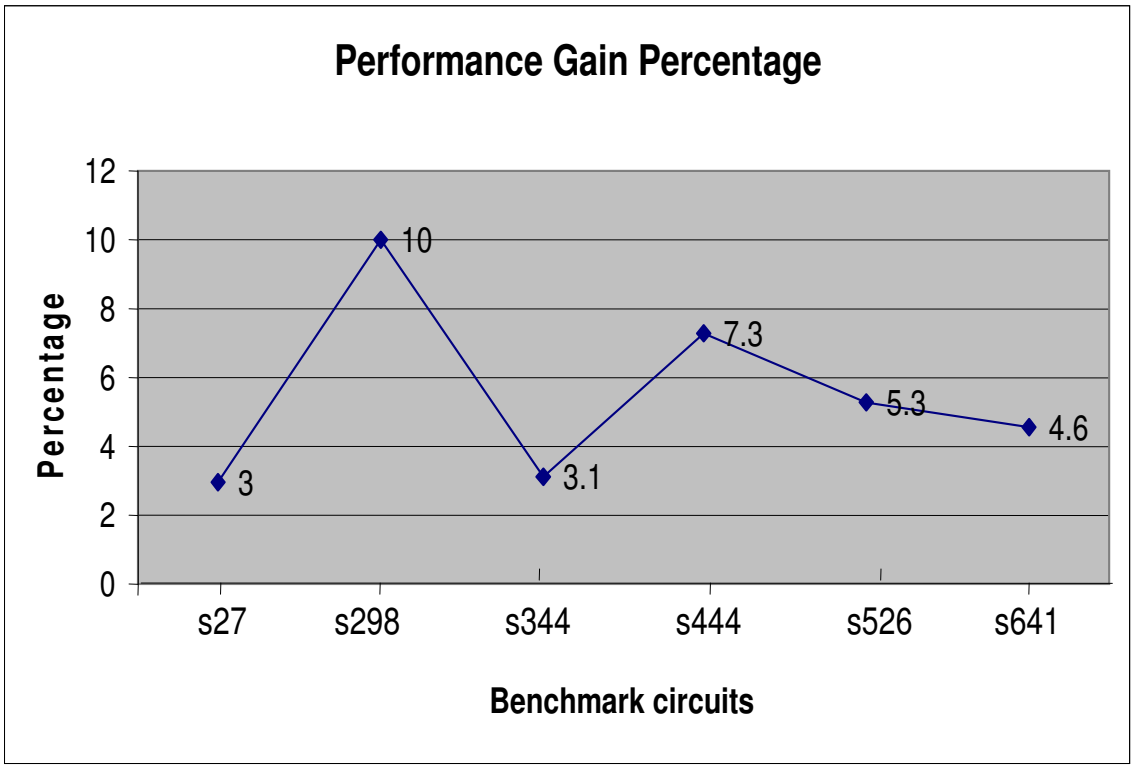


Figure 16: Performance gain percentage for ATPG patterns

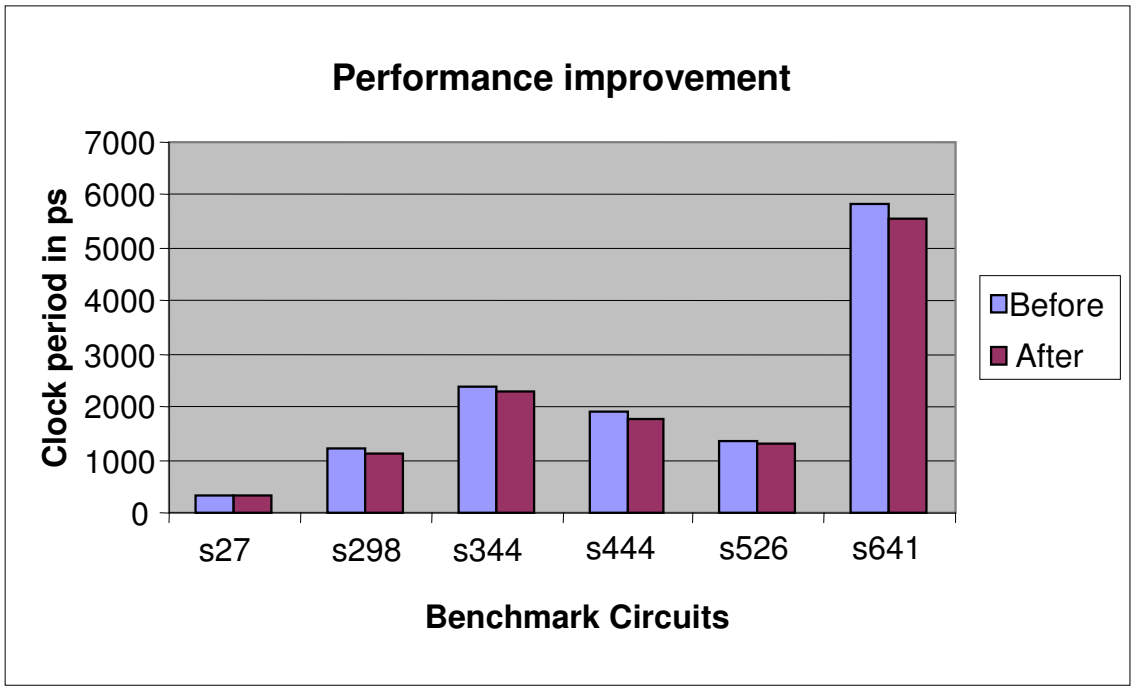


Figure 17: Performance Improvement for ATPG patterns

Benchmark Circuits	Before Tuning (ATPG) Min CLK Period in *ps*	After Tuning (ATPG) Min CLK Period in *ps*	% gain	Before Tuning 1000 Random Patterns in *ps*	After tuning 1000 Random patterns in *ps*	% gain	CPU Time in *s*
S27	340	330	3%	340	330	3%	0.016
S298	1220	1100	10%	1220	1100	10%	0.020
S344	2380	2304	3.1%	2415	2397	0.7%	0.032
S444	1900	1760	7.3%	2440	2390	2.04%	0.028
S526	1373	1300	5.3%	1373	1357	1.12%	0.052

Table IV: Performance gain table for ISCAS 89 circuits

In Table IV, the 2nd column shows the minimum clock period without the tuning settings. The 3rd column represents the new minimum clock period after the tuning settings were applied using the ATPG patterns. The 6th column represents the new clock period after using random patterns for timing simulation and tuning setting generation. So overall we see an improvement in the performance by just using the patterns and the corresponding failed output generated by the tester.

4.7 Summary

Post Silicon clock buffer tuning is one of the suggested remedies to improve the performance post manufacturing. We have presented a novel post silicon clock tuning system which uses only the external data available while testing, to generate the tuning

settings. Thus the proposed scheme will work on any design where the performance is degraded by process variation. Till now, we have not introduced the process variation into the circuits. This is tackled in the forthcoming chapters. On implementing the tuning settings generated by our tuning system, the benchmark circuits showed a significant 10% improvement in performance. Our present tuning system works on the assumption that each flip-flop has its own PST buffer and each flip-flop is tunable. But in reality there are a limited number of PST buffers and they are higher up in the clock tree rather than at the leaf level. The next chapter tackles this issue.

CHAPTER 5

MITIGATING THE IMPACT OF PROCESS VARIATION USING CPT

Previous chapters showed that the CPT successfully solved the problem of generating the tuning setting for the PST buffers. But we did not induce any process variation in the circuits. In this chapter we will induce process variation into the circuits and use our tuning system to mitigate the impact of process variation on performance.

It is a well known fact that the gate delays due to process variation in a circuit follow a normal distribution [1]. Process variation is introduced by perturbing the gate delays in a circuit. By repeating this process multiple times, multiple instances of the same circuits with process variation is created. We then generate tuning setting for each of the instances of the circuit and measure the performance benefits. In addition, we analyze the impact of PST buffer on the performance by placing them at different levels of clock tree. We use the clock tree synthesis technique used in [24]. The clock tree synthesis related details are explained in detail in the following section.

5.1 Clock Tree Synthesis for PST Buffers

One should note that Clock Tree Synthesis (CTS) is a design time activity, not a post-silicon activity. However, if the hardware tester is replaced by a timing simulator and what if questions are asked, sensitive flip-flops can be easily identified as explained in chapter 4. Please also note that these flip-flops take Boolean relationships into account

as well. With this assumption, following modification to critical path tracing is used to help design the clock tree.

Critical Path Tracing stops at a Flip-Flop. This does not mean that every flip-flop at the leaf level of the clock tree will have its own PST buffer. The PST buffers are placed at a higher level in the clock tree. So, if we tune a particular PST buffer, then its equivalent to tuning all the flip-flops in that particular sub tree. As shown in Figure 18, by tuning the buffer PST1, we are effectively tuning flip-flops FF1, FF2 and FF3. Thus by implication, we can derive which PST buffer(s) needs to be tuned and which flip-flops are tuned as a consequence.

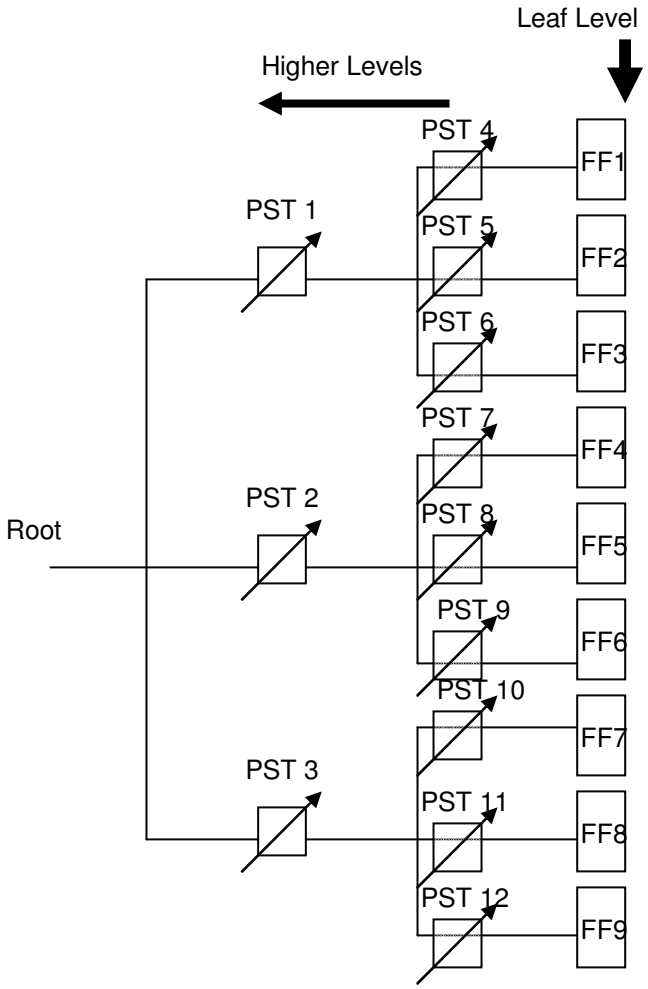


Figure 18: Clock Tree with PST buffer

```
Algorithm Clocktree_PST (CTS)
Input : standard clock tree with the flipflops at the leaf level
Ouput : Clocktree optimized for PST buffers
1 for each flip_flop in CTS
2 {    if {flip flop == Early_list_flip_flop}
3    {   //add flip-flop to the PST list which will be tuned early
4          FF_Early_PST_list=flip_flop;
5     }
6     if { flip flop == Late_list_flip_flop}
7       { // add flip-flop to the PST list which will be tuned late
8        FF_Late_PST_list=flip_flop;
9        }
10  }
11   // Place Early and Late PST and their corresponding FF in
12   // different clock subtrees
13   for each PST in CTS
14  {
15     clock_subtree1= FF_Early_PST_list;
16     clock_subtree2= FF_Late_PST_list;
17  }
```

Figure 19: Clock Tree Synthesis algorithm for PST buffer

From Table III in chapter 4, in order to tune FF1 flip-flop, we have to tune the PST buffer, PST1 in Figure 18. This results in the tuning of flip-flops FF1, FF2 and FF3. In order to tune a single flip-flop, we end up tuning additional flip-flops which may result in creating timing failures at the other outputs. As the placement of the PST buffers move to higher levels in the clock tree, the number of flip-flops which get tuned with the tuning of a single PST buffer increases. This increases the probability of failures at the outputs, as more number of flip-flops are being disturbed. This may result in not having any improvement in the performance from tuning. In order to correct this, we use our tuning

list and reshuffle the flip-flops in the clock tree such that the tuning element which were listed for early tuning are made to fall the same clock tree branch and the flip-flops which were listed for late tunings are put under different clock tree branch. This is a greedy algorithm for clock tree synthesis and is illustrated in Figure 19.

To accommodate the clock tree synthesis and process variation we introduced a minor change to our design methodology as shown in red in the Figure 20.

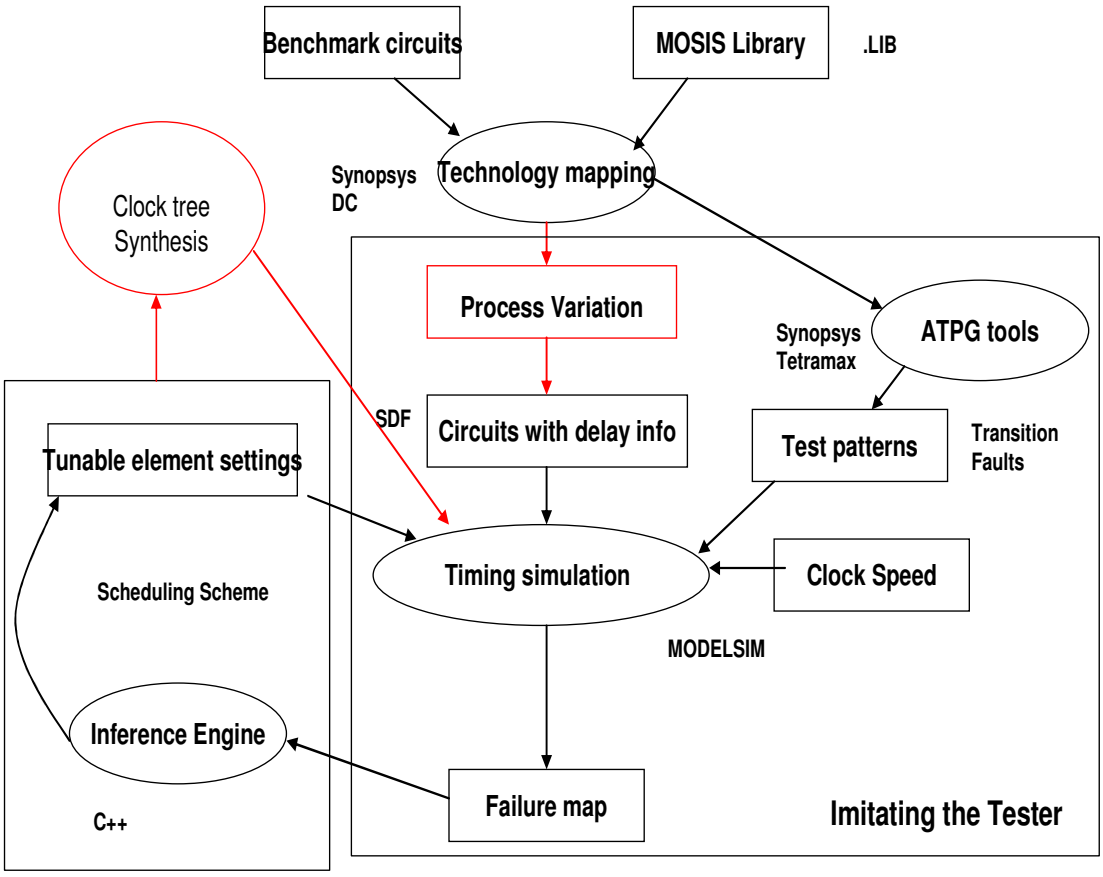


Figure 20: Modified Methodology with clock tree synthesis and process variation

By altering the clock tree based on our tuning system, we were able to see a performance gain of around 10% even when we tuned the PST buffers which were 2 levels above the leaf nodes of the clock tree. The important thing to note here is that the

clock-tree reshuffling is done the pre-silicon phase and this will help us in tuning the PST buffers in post silicon. In other words, the clock-tree synthesis is tuning system driven in the pre silicon phase. The results are explained in following section.

5.2 Experimental results

The tuning system is implemented in C++. We have used ISCAS 89 benchmark circuits to test our tuning system. As explained earlier, we created multiple instances of each circuit with the gate delays following a normal distribution.

The benchmark circuits are synthesized using Synopsys™ Design Compiler; transition test patterns are generated using Synopsys™ TetraMAX. We then run timing verification on the structural Verilog with ATPG patterns as inputs to these circuits. Timing verification is done for the targeted clock frequency and the failure patterns and the failure outputs are taken through our tuning system. The tuning system generates the tuning list consisting of the flip-flops, which have to be tuned to improve the performance.

Process variation is introduced by perturbing the gate delays in a circuit following a normal distribution. By repeating this process multiple times, multiple instances of circuits with process variation are created. The clock tree was generated as explained before. Table V and Table VI show the results for the ATPG and Random patterns respectively.

Ckts	Performance Improvement – Random Patterns											
	Leaf				L3				L2			
	Mu	σ	Min	Max	Mu	Σ	Min	Max	Mu	σ	Min	Max
S27	1.56	2.06	0	6.25	-	-	-	-	-	-	-	-
S298	8.16	2.13	5.51	11.5	3.7	2.13	1.35	7.35	4.44	3.04	0	7.3
S444	8.8	1.37	6	10	8.8	1.4	6	10.1	8.9	.83	7.6	10
S526	7.58	.69	6.5	8.7	7.8	1.12	5.3	8.7	0.17	0.53	0	1.7
S641	9.2	0.42	8.7	9.83	9.2	0.42	8.7	9.2	9.2	0.42	8.7	9.8

Table V: Performance improvement for ISCAS-89 benchmark circuits with PST buffers at different level for ATPG Patterns

Ckts	Performance Improvement – Random Patterns											
	Leaf				L3				L2			
	Mu	σ	Min	Max	Mu	Σ	Min	Max	Mu	σ	Min	Max
S27	1.56	2.06	0	6.25	-	-	-	-	-	-	-	-
S298	9.46	0.93	8.49	10.7	1.43	1.17	0	3.92	0.42	0.67	0	1.58
S444	8.24	1.31	5.64	9.43	8.31	1.37	5.65	9.7	8.7	1.47	5.7	10.1
S526	7.1	0.93	5.78	8.37	7.32	0.97	5.78	8.38	7.62	1.2	5.78	9.67
S641	4.12	0.90	2.89	5.7	5.12	0.88	4.15	7.02	7.6	1.2	5.51	9.22

Table VI: Performance improvement for ISCAS-89 benchmark circuits with PST buffers at different level for Random Patterns

Figure 21 plots the normal distribution of the minimum clock period for the different instance of the ISCAS benchmark circuit s298. The results are based on ATPG patterns.

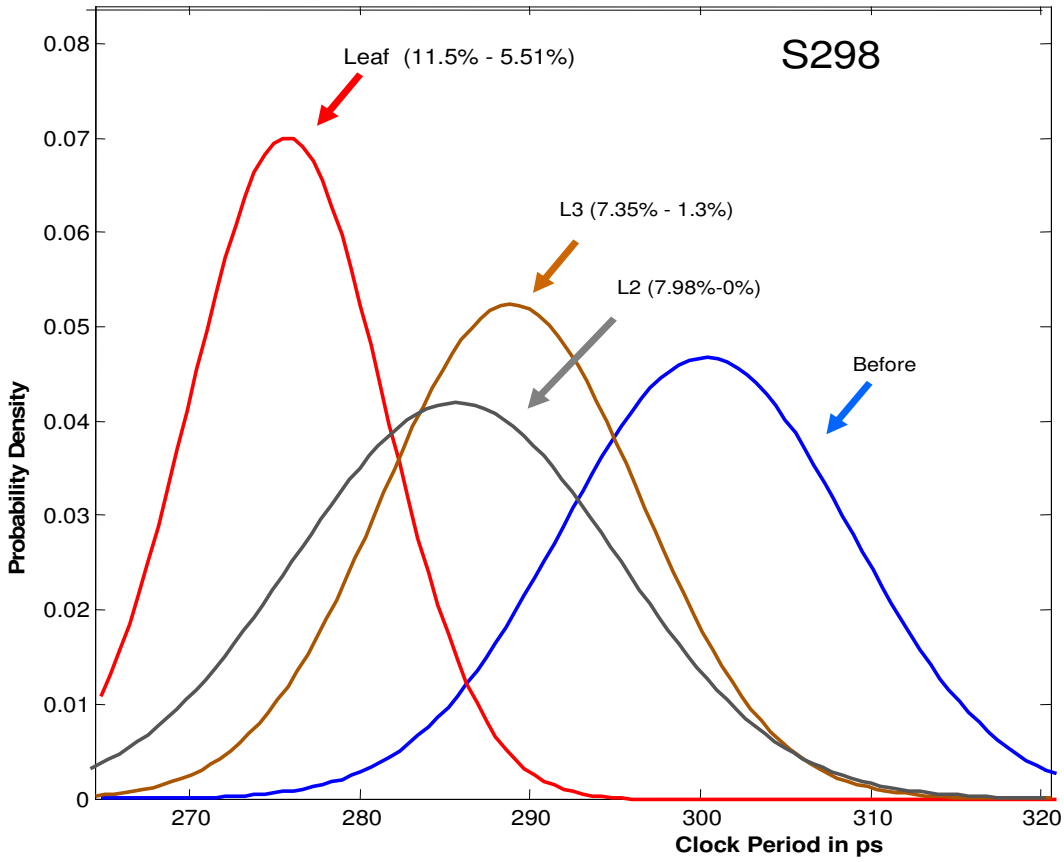


Figure 21: Tuning settings for s298 when the PST buffers were placed at different levels of clock tree- ATPG

In Figure 21, the distribution indicated by "before", is the distribution of minimum clock periods, when no tuning settings were applied. The distribution indicated by "Leaf", is the distribution when the tuning setting were applied, assuming that all the leaf nodes were tunable. This shows a performance improvement in the range of 11.51% to 5.5%. The distribution indicated by "L3", is the distribution when the tuning settings were applied to the PST buffers placed at one level above the leaf level (closer to the clock source). This showed a performance improvement in the range 7.35% to 1.3%. The decrease in the performance as compared to the leaf level, justifies our explanation in section 5.1, which indicated that by tuning the PST buffer at the higher level, we end up

disturbing unwanted flip-flops. Similarly, the distribution indicated by "L2" indicates that the tuned PST buffers were assumed to be two levels above the leaf level. This showed a performance improvement in the range of 7.35% - 0%. Again the explanation given in section 5.1 fits here.

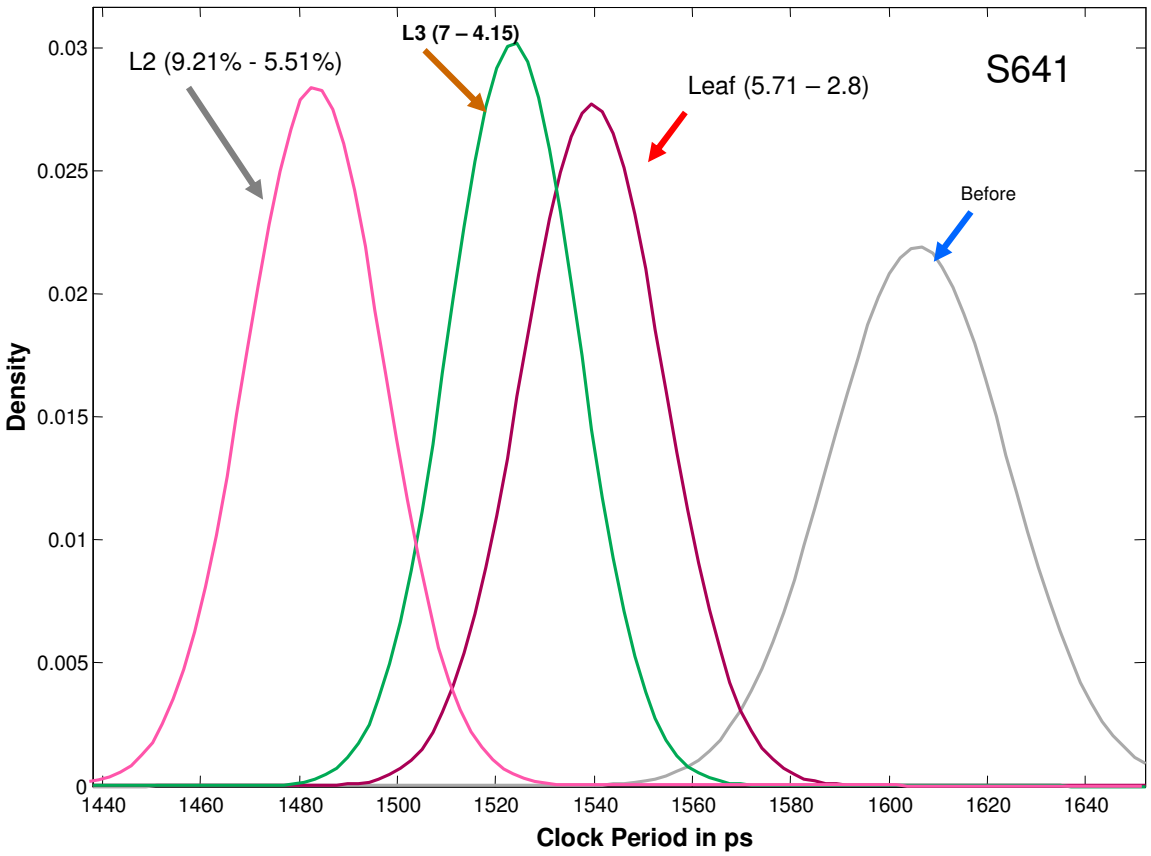


Figure 22: Tuning settings for s641 when the PST buffers were placed at different levels of clock tree- Random Patterns

Similar to Figure 21, Figure 22 shows the normal distribution of the minimum clock period for the s641 circuit for random patterns. Here it can be observed that the distribution for the level L2 is better than the level L3 and leaf. This is because tuning the PST buffer at level L2, constructively tuned the other flip-flops which were disturbed

when tuned by the PST at the lower level. To clarify, the tuning at different levels were done independently.

5.3 Summary

Post silicon clock buffer tuning is one of the available remedies for process variation. We have presented a novel post silicon clock tuning system which uses only the external data available to generate the tuning settings. Previous approaches were focused on delays but ignored Boolean relationship between inputs and outputs. By focusing on logic functionality, we take full advantage of Boolean relations between inputs and outputs. Using the tuning system, we also create an efficient clock tree in the pre silicon phase which offers further increase in performance. Dynamic simulation on benchmark circuits with circuit delays show around 9% average improvement in performance.

CHAPTER 6

CLOCK TREE SYNTHESIS USING STATIC TIMING ANALYSIS

In the previous chapters, we have shown from our experiments that by grouping the flip flops in different branches of the clock tree depending on the early and late tuning settings, we were able to generate constructive tuning settings and hence achieve performance benefit by PST buffer tuning. We want to approach the same problem using static timing analysis. We construct a clock tree which is static timing driven.

6.1 Static Timing Analysis

Static timing analysis is a method of validating the timing performance of a design by checking all possible timing paths for timing violations. Below are the brief explanations of some of the terms which are used in Static Timing Analysis.

- **Startpoint**: Startpoint is a place in the design where the data is launched by the clock edge. The startpoint of a path is a clock pin of a sequential element, or an input port of the design.
- **Endpoint:** Endpoint is a place in the design where the data is captured by the clock edge. The endpoint of a path is a data pin of a sequential element, or an output port of the design.
- **Timing Path:** A timing path is a point-to-point sequence through a design that starts at a register clock pin or an input port, passes through combinational logic elements, and ends at a register data input pin or an output port

- **Critical Path:** A critical path is a timing path with the maximum delay or the least timing slack. The critical path determines the maximum frequency of a design.

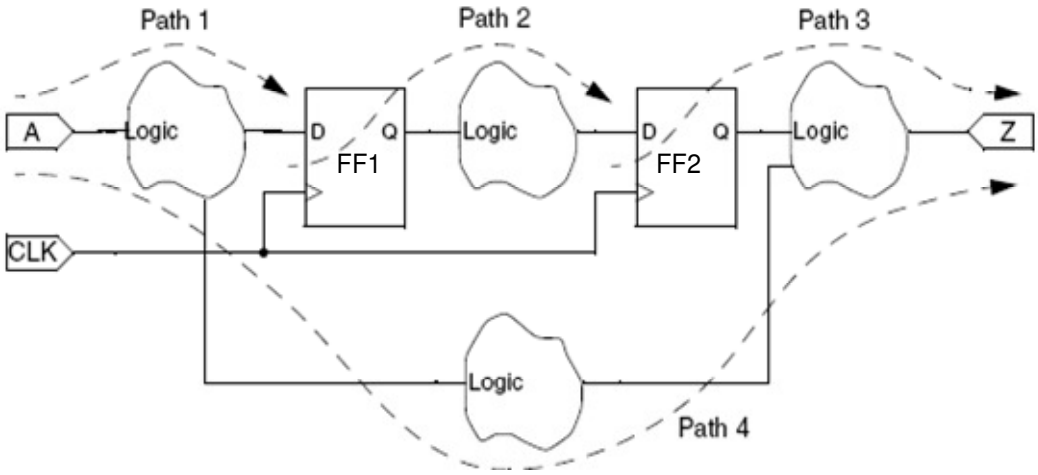


Figure 23: Timing Path [26]

In Figure 23, each logic cloud represents a combinational logic network. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point:

- Path 1 starts at an input port A and ends at the data input of a sequential element FF1.
- Path 2 starts at the clock pin of a sequential element FF1 and ends at the data input of a sequential element FF2.
- Path 3 starts at the clock pin of a sequential element FF2 and ends at an output port Z.
- Path 4 starts at an input port A and ends at an output port Z.

6.2 Static Timing Driven Clock Tree Synthesis

As explained above, a timing path will have a startpoint and endpoint. The data is launched by the startpoint and captured by the endpoint. The startpoint can be a clock pin of a sequential element (flip-flop) and the endpoint can be the data input of a sequential element (flip-flop). In our discussion, the startpoint sequential element is called a launch flip-flop and the endpoint sequential element is called the capture flip-flop. In Figure 23, flip-flop 'FF1' is the launch flip-flop and 'FF2' is the capture flip-flop.

In Figure 24, FF1 is the launch flip-flop and FF2 is the capture flip-flop for the path 1, FF2 is the launch flip-flop and FF3 is the capture flip-flop for the path 2, FF2 is the launch flip-flop and FF4 is the capture flip-flop for the path 3. Let us assume that delay of the path 2 is greater than the delay of path 3 and delay of path 3 is greater than the delay of path 1. Path 2 starting from the clock pin of flip-flop FF2 and ending at the data pin of the capture flop FF3 is the critical path for the circuit shown in Figure 24.

Let T_{launch} and $T_{capture}$ be the clock arrival time at launch and the capture flip-flops, respectively. In order to avoid timing violations, the capture flip-flop should capture the right data launched by the launch flip-flop. For this to happen, the following equation must be satisfied.

$$T_{launch} + D < T_{clk} + T_{capture} - T_{setup} \quad (2)$$

Here, T_{clk} is the clock period, T_{setup} is the setup time of the flip-flop D is the maximum delay of the combinational logic in the path. PST buffers can control the clock arrival times T_{launch} and $T_{capture}$. Equation 2 can be satisfied either by reducing T_{launch} or by

increasing $T_{capture}$. In other words, we can avoid the timing violation either by making the

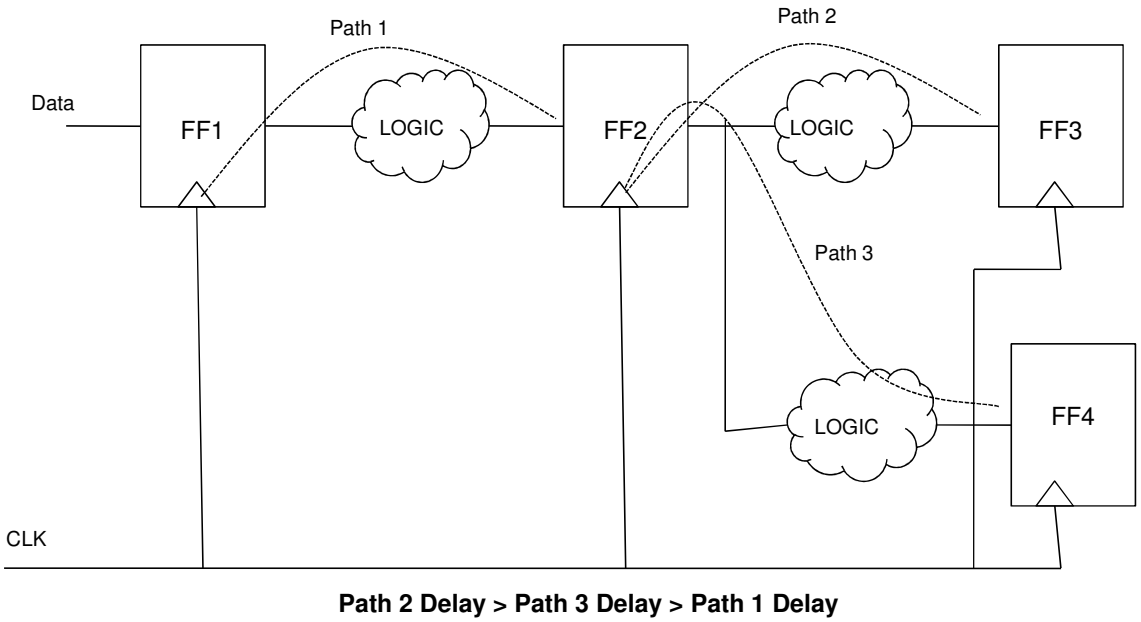


Figure 24: Timing Path example

clock to the launch flip-flop early or by making the clock to the capture flip-flop late.

This means that the PST buffers can be used to achieve the above goal. In order for the above scheme to work, the clock tree should be structured in such a way that all the launch flops should be in the clock tree branch which are targeted to receive the clock early and all the capture flip-flops should be in the clock tree branch which are targeted to receive the clock late. But there might be a scenario where a particular flip-flop can be a launch and the capture flip-flop when multiple timing paths are considered. As shown in Figure 24, flip-flop FF2 is a capture flip-flop for path 1 and the same flip-flop is the launch flip-flop for the path 2 and path 3.

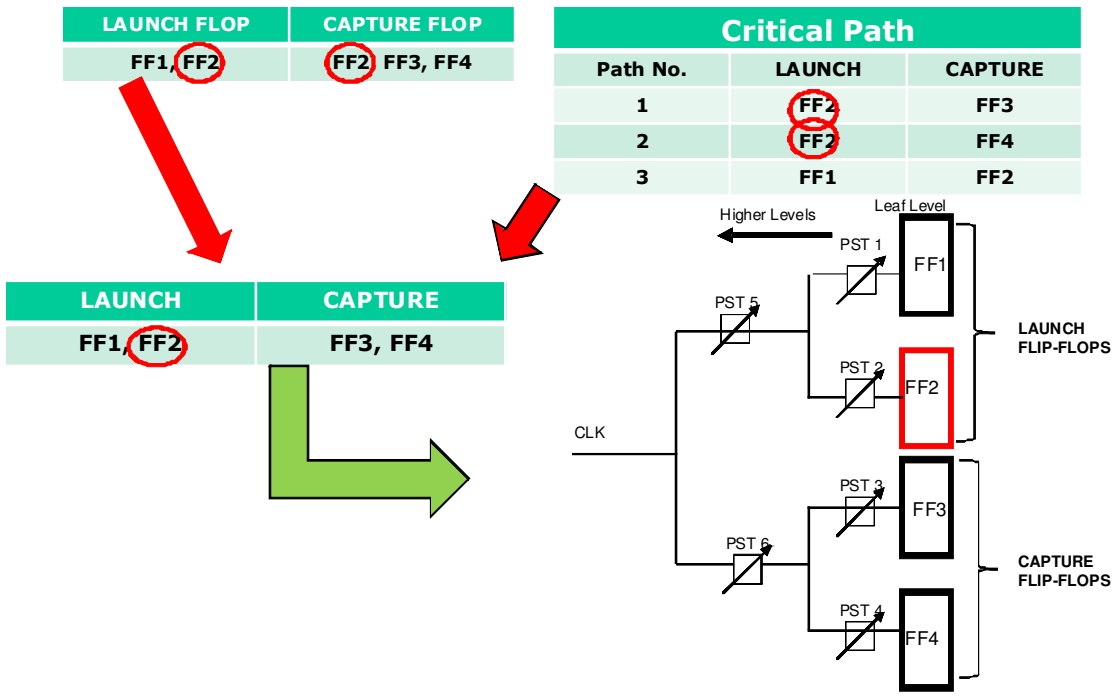


Figure 25: Static timing based clock tree

In order to avoid the above contention, we follow the following two rules while constructing the clock tree.

I. All the launch flip-flops should be in the clock tree branch which is targeted for the early clock tuning and all the capture flip-flops should be in the clock tree branch which targeted for late clock tuning.

II. In case of contention, critical path is used to decide whether a flip-flop falls in the early tuning clock tree branch or the late tuning clock tree branch.

Above two rules are explained clearly with the below example.

Example: As shown in Figure 24, flip-flop FF2 is the capture flip-flop for path 1 and is also the launch flip-flop for path 2 and path 3. In such case, we look for a critical path. In Figure 24, the delay of path 2 and path 3 is greater than the delay of the path 1. Flip-flop

FF2 is the launch flip-flop in the critical path and as a result the flip-flop FF2 falls under the early tuning clock tree branch. Critical path determines the maximum frequency of a circuit and the affect of process variation on critical path can affect the maximum frequency. As a result, critical path is considered for the clock tree synthesis.

Figure 25 shows the clock tree which is static timing based. Since FF1 and FF2 are the launch flip-flops, they are placed under the clock tree branch whose clock arrival time can be decreased by the PST buffer PST 5. Similarly the capture flip-flops are placed under the clock tree branch whose clock arrival time can be delayed by the PST buffer PST 6.

CHAPTER 7

PATH DELAY TEST MODEL

7.1 Path Delay Test

A path delay defect causes the cumulative propagation delay of a combinational path to increase beyond the specified time duration resulting in timing violations [25]. Path delay fault model is useful for testing and characterizing critical timing paths in a design [26]. Path delay fault tests exercise the critical paths at the full operating speed of the chip to detect whether the path is too slow because of manufacturing defects or process variations.

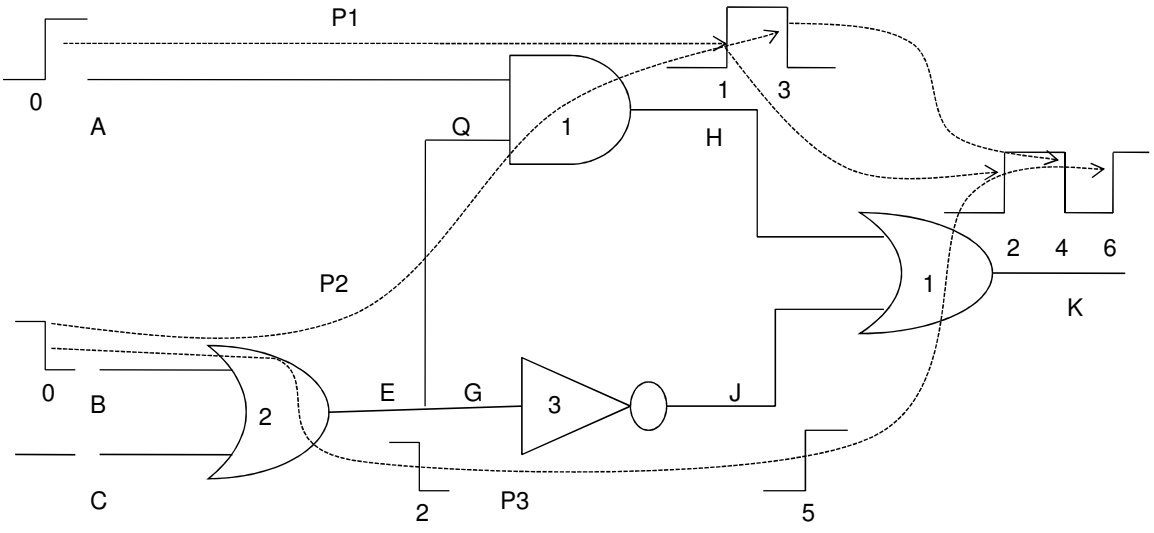


Figure 26: An example of transition propagation through paths [25]

Path delay fault may be because of physical defects which might increase the delay of the circuit on a chip. For example, incorrect field oxide thicknesses could lead to

slower signal propagation times, which could cause transitions along a critical path to arrive too late [25].

Example: Consider the circuit shown in Figure 26. All the gates have integer multiples of unit delays and the delays are marked on the gates. Signal waveforms are sketched at various nodes in the Figure 26 with the time of each transition shown under it. The time of input transitions is 0, making them the reference for all other transitions. As shown in the figure, output has three transitions brought via the three paths P1, P2 and P3. Let us assume that in this circuit the input and the output are synchronized with a clock of period T. From the Figure 26 it is clear that the critical path has a delay of 6 units along the path P3 in a fault free circuit. Suppose we choose T = 7, then any path will be faulty if its delay exceeds 7 units. All the 3 paths can exceed the delay of 7 units due to manufacturing defect resulting in timing violation. If the delay at the output exceeds only because of the path P3 then the input vector pair patterns of 010 and 100 should be able detect this fault as long as the delay is exceeding only because of path P3.

7.2 Non-Robust and Robust Path Delay Test

Non-robust path delay test guarantees to detect a path delay fault when no other path-delay fault is present. A robust path delay test guarantees to produce an incorrect value at the destination if the delay of the path under test exceeds a specified time interval, irrespective of the delay distribution in the circuit.

A path delay test uses a pair of input vectors to apply transition at the input of the path and then measures the output value after a specified clock period. For the path to be testable, the transition at the input of the path should control the transition at the output.

Consider the path-delay fault of transition to 0 at the input of the path P3 in the Figure 26. Signals B, E, G, J and K are called the on path signals. Signals that are not in the path P3 but feed the gates on the path are called the off-path signals. A non robust test consists of a vector-pair V1, V2 such that the change from V1 to V2 initiates the appropriate transition at the beginning of the path under test. In addition, all off path input signals for the path under test assume non-controlling values, which is 0 for an OR or NOR, and 1 otherwise in the steady state, following the application of the second vector V2. In other words, the non-path transitions shouldn't control the transitions on the path which is being tested. Path delay faults are tested using the following sequence:

- The first vector initializes the path before applying the launch event, typically a clock pulse.
- The second vector propagates a logic transition along the entire path.
- A second clock pulse, occurring one at the clock period of the design captures the resulting transition at the end of the path.

7.3 Methodology to Generate the Path-Delay Test Patterns

Path delay test pattern generation flow is shown in Figure 27. Following are the steps leading to the generation of the path delay patterns.

- ISCAS 89 benchmark circuits are synthesized using Synopsys Design Compiler® and mapped to the 45 nm technology.
- Static Timing Analysis (STA) is performed on the synthesized circuit using Synopsys PrimeTime®. It should be noted that the all the false paths are constrained in the STA. STA is done at the desired clock period and the paths

which have maximum delays (critical paths) are generated in a format which can be read by the Synopsys TetraMax®. The syntax used to define critical delay paths is shown in Figure 28.

- Path delay fault ATPG targets individual path delay faults and then simulates each test generated against the remaining undetected faults in the fault list using both robust and non-robust path delay fault models suitable for pass/fail manufacturing tests [26]. TetraMAX® reads in the critical path reported by PrimeTime® and then generates the path delay test vectors which are used for verilog timing simulations.

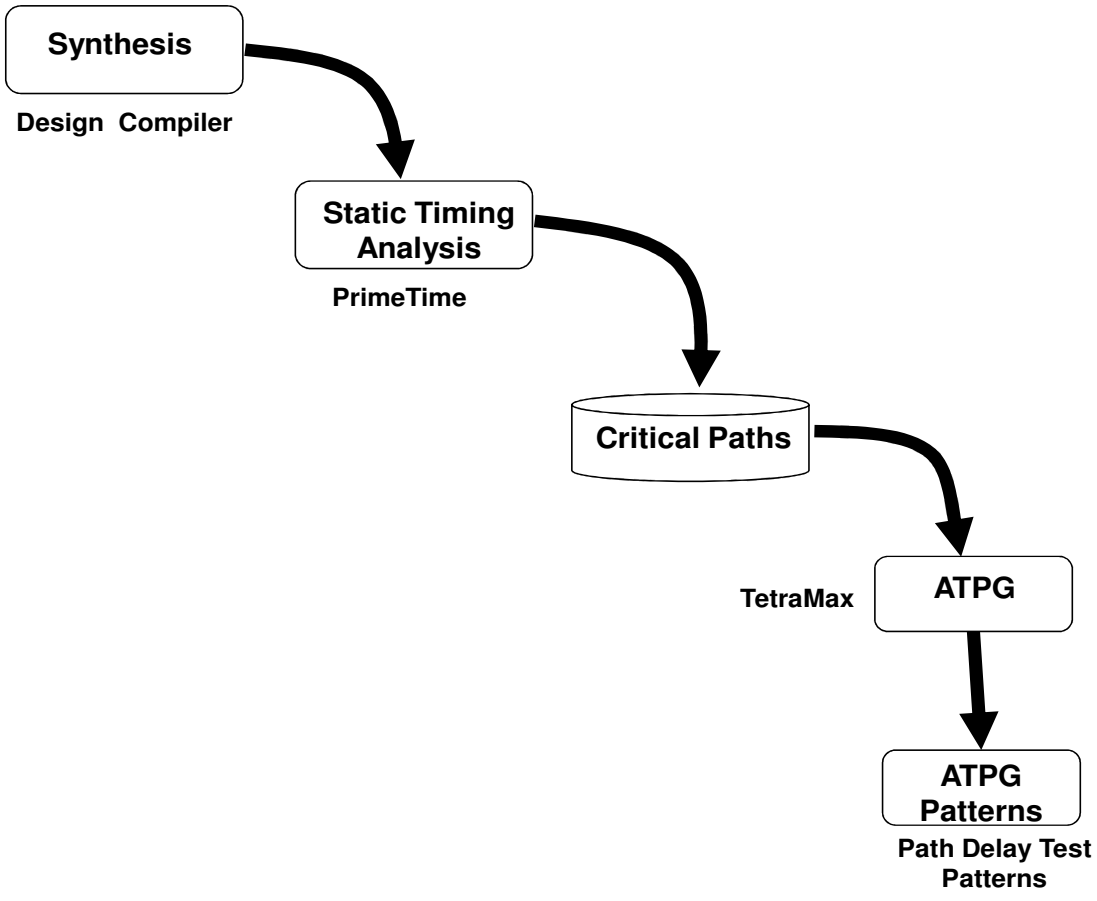


Figure 27: Path delay testing flow

```
$path {
        [ $name path_name ; ]
        [ $cycle required_time ; ]
        [ $slack slack_time ; ]
        [ $launch clock_name ; ]
        [ $capture clock_name ; ]
        $transition {
                pin_name1 ^ | v | = | ! ;
                pin_name2 ^ | v | = | ! ;
                …
                pin_nameN ^ | v | = | ! ;
        }
}
```

Where:

$name - Assigns a name to the delay path

$cycle - Time between launch and capture clock edges

$slack - Available time margin between the $cycle time and calculated delay of the path

$launch - Launch clock primary input to be used

$capture - Capture clock primary input to be used

$transition - (Required) Describes the expected transitions of path startpoint, output pins of path cells, and path_endpoint.

Figure 28: Syntax to describe the critical delay path used by TetraMax® [26]

Path delay testing is used to generate the effective set of ordered patterns which will is then used by our tuning system to generate the tuning settings. Path delay test guarantees to produce an incorrect value at the destination if the delay of the path under test exceeds the cycle time [25]. We have used these concepts and the patterns in our clock tuning system to generate the tuning settings for the PST buffer.

When considering for the path delay faults, only the paths on the critical paths are considered as these will be the one which will decide the cycle time of the circuit and any fault on these paths will impact the circuit performance. The results generated using the path delay patterns in the tuning system are explained in the next chapter.

CHAPTER 8

PST CLOCK TUNING RESULTS USING STATIC TIMING ANALYSIS AND PATH DELAY TEST MODEL

In chapter 6, we discussed using the static timing analysis for clock tree synthesis and in chapter 7, we discussed the generation of path delay test patterns to test the critical path of a circuit which decide the maximum frequency of a design. In this chapter we show the results after incorporating the concepts discussed in chapter 5 and chapter 6 into tuning system methodology to mitigate the impact of process variation on performance of a circuit.

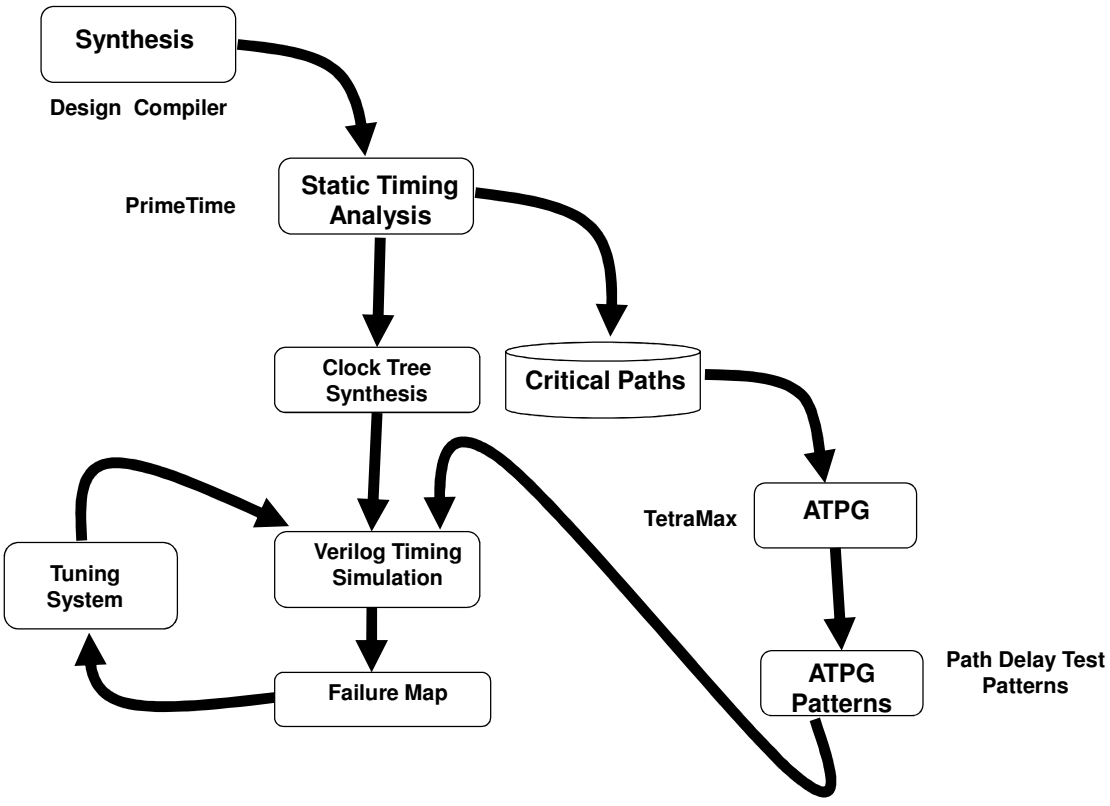


Figure 29: Clock tuning system methodology using path delay test patterns and static timing based clock tree synthesis

Figure 29 shows the experimental setup used in generating the results. As explained in chapter 4 and chapter 5, Design Compiler® is used to generate the synthesized benchmark circuit. Static Timing Analysis (STA) is performed on the synthesized circuit. As explained in chapter 6, STA is used to synthesize the clock tree which will assist in the clock tuning of PST buffer using our method. STA is also used to generate the critical path report which is required by the TetraMAX® to generate the path delay test patterns. As explained in chapter 7, TetraMAX® reads in the critical path report generated by PrimeTime® and generates the path delay test patterns. These patterns are used to test the critical path for any manufacturing defects which can affect the delay of the critical path and hence affect the maximum frequency of the circuit.

Timing verification is carried out on the structural verilog with the path delay test patterns as inputs to these circuits. Timing verification is carried out for the targeted clock frequency. The failed patterns and the corresponding failed outputs are taken through our tuning system, which generates the tuning list consisting of flip-flops. The generation of tuning list is explained in detail in section 4.5. The flip-flops in the tuning list are then tuned to improve the performance.

It should be noted that the process variation is modeled in the circuits the same way as explained in chapter 5. In addition, it should also be noted that the clock tree synthesis is a design phase activity and not a post silicon phase activity. The results are generated with the static timing based clock tree synthesis as explained in chapter 6.

8.1 Results

Before we explain the results we need to understand the following terms which will be used.

- **Before** : Performance of the circuit when the clock tuning is not used
- **Leaf**: Clock tree level with each flip-flop having its own PST buffer.
- **L3**: Clock tree level with PST buffer moved up one level from the leaf level.
- **L2**: Clock tree level with PST buffer moved up two level from the leaf level.

Ckts	Performance Improvement – Random Patterns											
	Leaf				L3				L2			
	Mu	σ	Min	Max	Mu	Σ	Min	Max	Mu	σ	Min	Max
S298	18.3	1.1	16.8	19.2	18.7	0.92	17.33	19.74	18.5	0.99	17.3	19.7
S526	9.8	0.17	9.6	10.2	9.9	0.17	9.58	10.17	9.9	0.17	9.6	10.2
S641	9.4	0.08	9.2	9.5	9.4	0.08	9.22	9.52	9.4	0.08	9.2	9.5
S1238	12.4	0.37	11.8	13.0	12.5	0.37	11.81	12.99	10.8	1.85	6.8	12.5

Table VII: Performance improvement with clock tuning for ISCAS 89 circuits and with path delay test patterns

Table VII shows the performance improvement for the ISCAS 89 circuits with our clock tuning technique. The patterns used here are the path delay test patterns. It should be noted that there is an average performance benefit of 18.3% for the s298 circuit. As explained in chapter 7, path delay fault model is useful for testing and characterizing critical path which decide the maximum frequency or minimum period of a circuit. The results shown in the chapter 5 were generated using the stuck-at fault model. The single stuck-at fault model (stuck-at-0 or stuck-at-1) plays an important part in manufacturing test. However, path delay fault is useful for achieving higher quality testing and as a result we see that the performance benefit achieved with the tuning setting is better in case of path delay test patterns.

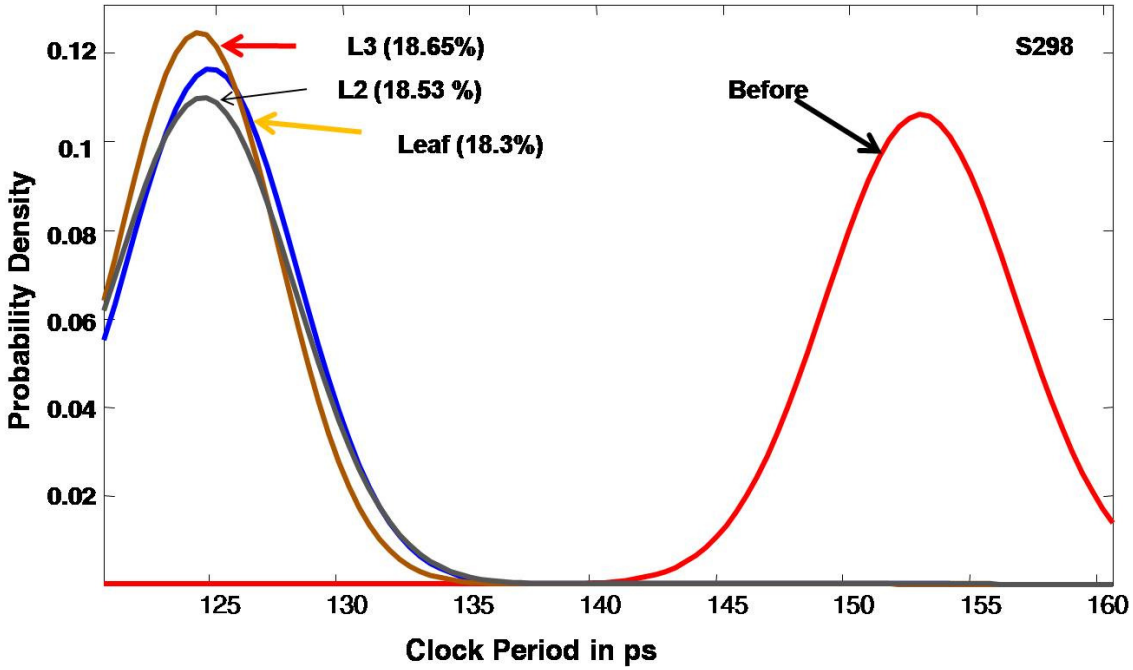


Figure 30: Normal distribution of clock period for the circuit s298 with and without the clock tuning with path delay test patterns

In Figure 30, the normal distribution for the clock period for the s298 ISCAS 89 benchmark is shown. The bell shaped curve marked as 'Before' indicates the normal distribution of minimum clock periods of s298 circuit with process variation modeled as explained in chapter 5. The 'Before' distribution is before the use of our PST clock buffer tuning system. This shows an average minimum clock period of 152.8 ps. The bell shaped curve marked as 'Leaf' indicates the normal distribution of minimum clock periods of s298 circuit with process variation. The 'Leaf' distribution is after using our PST clock buffer tuning system. In addition, all the leaf level nodes (flip-flops) of the clock tree synthesized using static timing are individually tunable. In other words, each flip-flop has its own PST buffer. This distribution has an average minimum clock period of 124.8ps which is improved by an average of 18% when compared to the one without clock tuning system.

The bell shaped curve marked as 'L3' indicates the normal distribution of minimum clock periods of s298 circuit with process variation. The 'L3' distribution is after using our PST clock buffer tuning system. For 'L3', the PST buffers are moved one level higher up in the clock tree from leaf level. In other words, each flip-flop will not have its own PST buffer but a bunch of flip-flops have a PST buffer. This distribution has an average minimum clock period of 124.3ps which is improved by an average of 18% when compared to the one without clock tuning system.

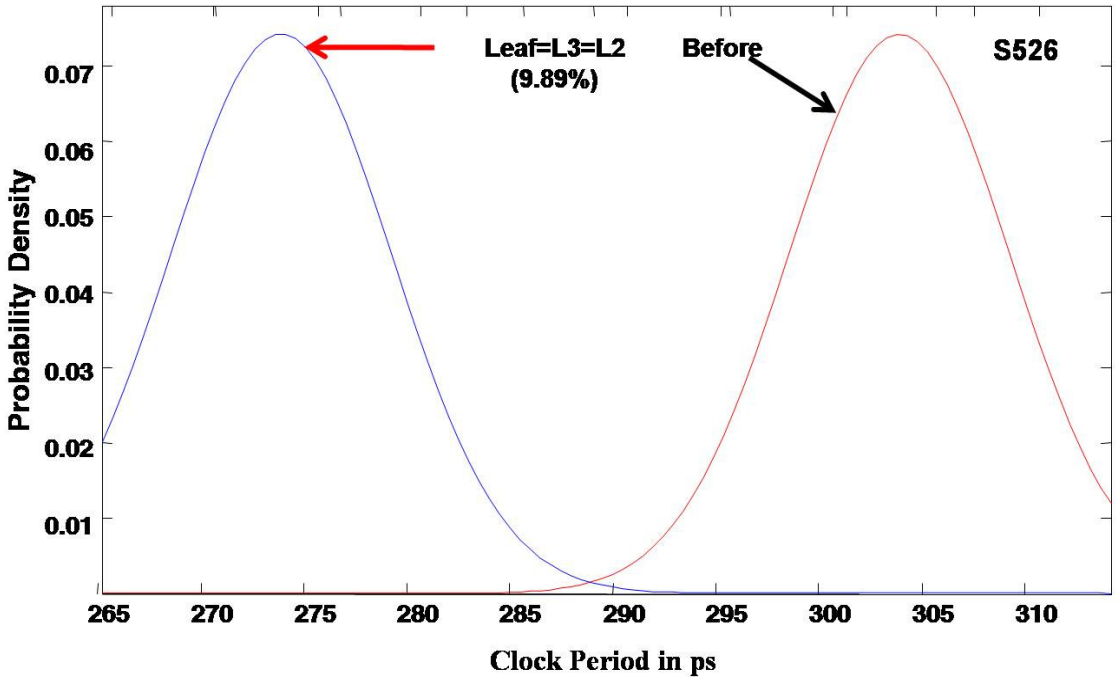


Figure 31: Normal distribution of clock period for the circuit s526 with and without the clock tuning with path delay test patterns

The bell shaped curve marked as 'L2' indicates the normal distribution of minimum clock periods of s298 circuit with process variation. The 'L2' distribution is after using our PST clock buffer tuning system. For 'L2', the PST buffers are moved two level higher up in the clock tree from leaf level. In other words, each flip-flop will not

have its own PST buffer but a bunch of flip-flops have a PST buffer. This distribution has an average minimum clock period of 124.5ps, which is improved by an average of 18% when compared to the one without clock tuning system.

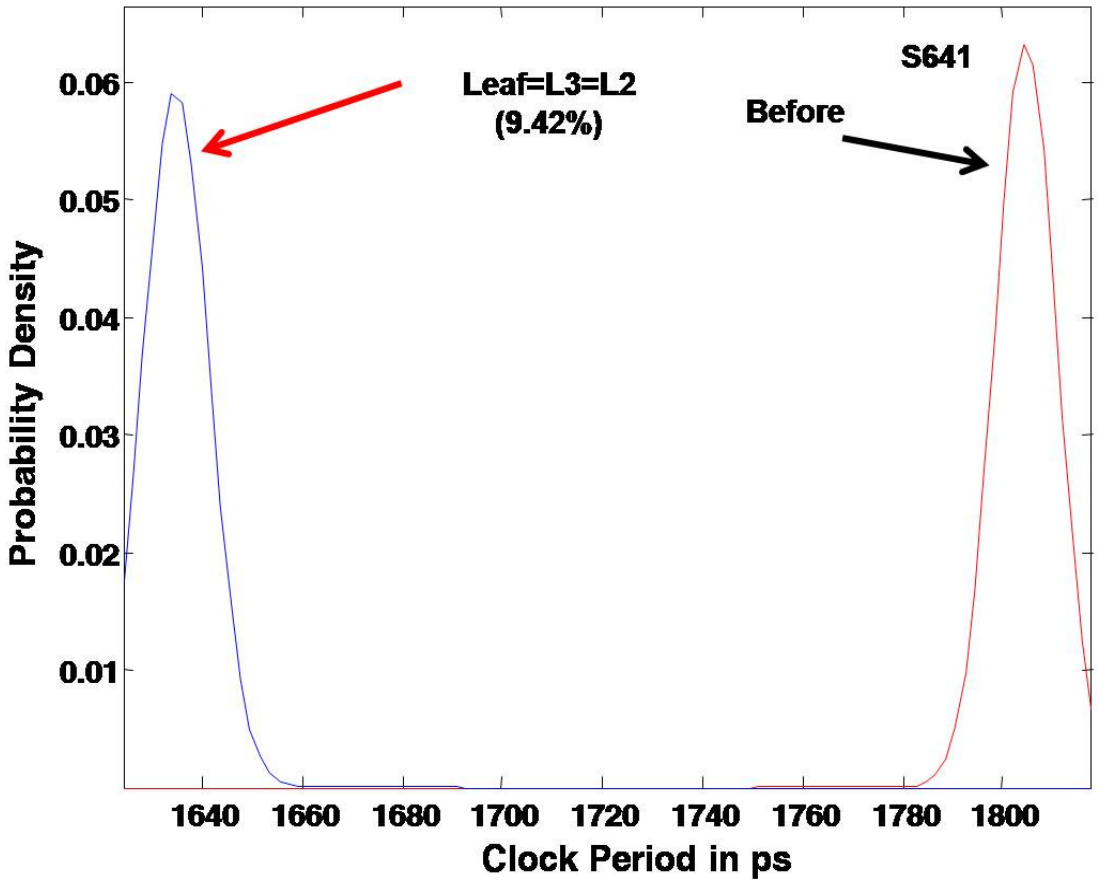


Figure 32: Normal distribution of clock period for the circuit s641 with and without the clock tuning with path delay test patterns

It should be noted that the change in the minimum clock period when the PST buffers are moved to different levels is attributed to the fact that some of the unwanted flip-flops get tuned and this was explained in chapter 5.

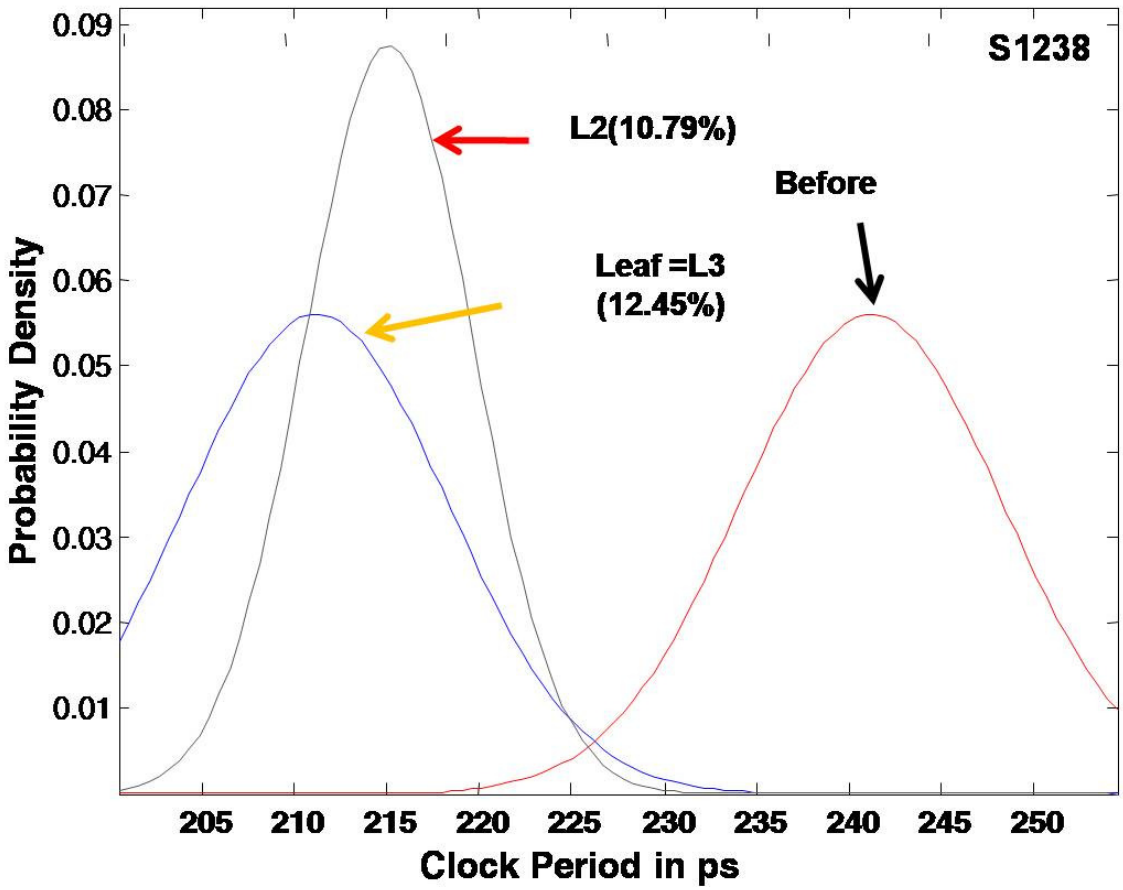


Figure 33: Normal distribution of clock period for the circuit s1238 with and without the clock tuning with path delay test patterns

Figure 31 and Figure 32 show the results for the circuit s526 and s641. In both of these circuits, performance improvement remains same at the different levels of the clock trees post clock tuning. This strongly attests our static timing based clock tree synthesis.

In Figure 33, the performance benefit seen at the higher level L2 of the clock tree is less because of the tuning of unwanted flip-flops. But we still see an average improvement of 11% for the s1238 circuit.

Figure 34 shows the performance improvement chart for the ISCAS 89 circuits after the use of our clock tuning system, static timing based clock tree synthesis and the

path delay test patterns. There was an average improvement of 18% to 9% in the performance after our PST buffer clock tuning concept was used.

Figure 34: Performance improvement with clock tuning for ISCAS 89 benchmark circuits

CHAPTER 9

SUMMARY

Process variation in chip fabrication and dynamic voltage control leads to clock skew as well as path delay variation in a manufactured chip. The thesis has outlined post-silicon clock buffer tuning as one of the available remedies for process variation. We have presented a novel post silicon clock tuning system which uses only the external data available to generate the tuning settings. Previous approaches were focused on delays but ignored Boolean relationship between inputs and outputs. By focusing on logic functionality, we have taken full advantage of Boolean relations between inputs and outputs. Using our tuning system, we also created an efficient clock tree structure in the pre silicon phase which offers further increase in performance. Dynamic simulation on benchmark circuits with circuit delays show around 9% average improvement in performance when the stuck at and transition fault models were used.

To further increase the performance benefit and assist our PST clock buffer tuning system, we constructed a clock tree structure which was based on static timing analysis. In addition, we used path delay fault model which is useful for testing and characterizing critical timing paths which determine the maximum frequency of a design. Using the static timing and path delay test model techniques our PST clock buffer tuning system showed a maximum performance improvement of 18%.

In this thesis, we were successful in showing that the Critical Path Tracing (CPT) based Post Silicon Tunable (PST) clock buffer tuning system mitigated the impact of process variation on the performance of a design.

BIBLIOGRAPHY

[1] S. Borkar *et al*., ”Parameter Variations and Impact on Circuits and Microarchitecture”. Proc. DAC, 2003

[2] M. Mani *et al.*, “An efficient algorithm for statistical minimization of total power under timing yield constraints,” Proc. DAC, 2005

[3] S. Nassif, “Delay variability: sources, impacts and trends,” Proc. ISSCC, 2000

[4] J. Singh et al., “Robust gate sizing by geometric programming,” Proc. DAC, 2005

[5] Khandelwal et al. “A General Framework for Accurate Statistical Timing Analysis Considering Correlations”. Proc. DAC, 2005

[6] A. Agrawal, K. Chopra, D. Blaauw, and V. Zolotov. ”Circuit Optimization Using Statistical Static Timing Analysis”. Proc. DAC, 2005

[7] D. Sinha, N. V. Shenoy, and H. Zhou. “Statistical Gate Sizing for Timing Yield Optimization”. Proc. ICCAD, 2005

[8] Tam, Stefan Rusu, Utpal Nagarji Desai, Robert Kim, Ji Zhang, and Ian Young. “Clock generation and distribution for the first IA-64 microprocessor”. Proc. IJSSC, 2000

[9] Patrick Mahoney, Eric Fetzer, Bruce Doyle, and Sam Naffziger. ”Clock distribution on a dual-core multithreaded Itanium-family processor”. Proc. ISSCC, 2005

[10] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi, “A post-silicon clock timing adjustment using genetic algorithms”. Proc. VLSI circuits, 2003

[11] H. Chang and S. Sapatnekar. “Statistical Timing Analysis considering Spatial Correlations Using a Single Pert-Like Traversal”. Procs. ICCAD, 2003

[12] Y. Zhan, A. J. Strojwas, X. Li, L. T. Pileggi, D. Newmark, and M. Sharma. “Correlation-aware Inbox statistical timing analysis ith non-gaussian delay distributions”. Procs. DAC, 2005

[13] K Nagaraj, S Kundu, “An Automatic post silicon tuning system for performance gain using tester measurements” Proc. ITC 2008

[14] Charlie Chung-Ping Chen *et. al* “Statistical timing analysis driven post-silicon-tunable clock-tree synthesis”, Proc. ICCAD 2005

[15] Jeng-Liang Tsai , DongHyun Baik , C. C. -P. Chen , K. K. Saluja, "A yield improvement methodology using pre- and post-silicon statistical clock scheduling" Proc. ICCAD, 2004

[16] V. Khandelwal and A. Srivastava "Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation," Proc. ISPD, 2007

[17] José Luis Neves , Eby G. Friedman, "Optimal clock skew scheduling tolerant to process variations" Proc. DAC, 1996

[18] Fishburn, J. P, "Clock skew optimization" IEEE Transactions on Computers , 1990

[19] Simo Patrick Mahoney, Eric Fetzer, Bruce Doyle and Sam Naffziger, "Clock distribution on a dual-core multi-threaded Itanium-family processor", Proc. ISSCC, 2005

[20] Simon Tam, Stefan Rusu, Utpal Nagarji Desai, Robert Kim, Ji Zhang, and Ian Young. "Clock generation and distribution for the first IA-64 microprocessor" IEEE Journal of Solid-State Circuits, 35(11):1545–1552,Nov 2000

[21] M. Abramovici, P. R. Menon and D Miller, "Critical Path Tracing: An Alternative to Fault Simulation," Proc. DAC, 1983

[22] Girard, P. Landrault, C. Prayossoudoyitch, "A reconvergent fanout analysis for the CPT algorithm used in delay-fault diagnosis" Proc. ETC, 1993

[23] P. Menon, Y. levendel, and M. Abramovici, "SCRIPT: A critical path tracing algorithm for synchronous sequential circuits", Proc. DAC, 1991

[24] K Nagaraj and Sandip Kundu, "A study of placement of post silicon tuning buffers for mitigating impact of process variation", Proc. DATE, 2009

[25] Bushnell, M., Agrawal, Vishwani, "Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits", ISBN: 978-0-7923-7991-1

[26] https://solvnet.synopsys.com