NOVEL STATIC TIMING ANALYSIS APPROACH FOR

CUSTOMIZED DESIGN FABRIC

by

Jiajia Wang

APPROVED BY SUPERVISORY COMMITTEE:

_____

Dr. Carl Sechen, Chair

_____

Dr. William Swartz

_____

Dr. Dian Zhou

NOVEL STATIC TIMING ANALYSIS APPROACH FOR CUSTOMIZED DESIGN FABRIC

by

JIAJIA WANG, BS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN

ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

August 2018

ACKNOWLEDGMENTS

I was very lucky to have completed my master thesis in one of the best VLSI team in the world. In the three years of study and research, my advisor and my teammate were always with me whenever I was in difficulty. I am also thankful to whoever accompany me in these three years at UT Dallas.

Specifically, I would like to first say thank you to Dr. Carl Sechen for providing an opportunity to study in his outstanding research group. In this team, I involved deeply in the state of art circuit design and research that help me grow up to who am I today. I won't forget the encouragement that he gave me every week and thank him for making the research so enjoyable. I would also like to thank Dr. William Swartz and Dr. Dian Zhou for your help in my study and also serving as my committee member. I was really happy to share and discuss my idea my work with all of you.

I would also never forget my nice lab mates Jingxiang Tian, Xianyu Xu and Bo Hu that work with me in the same lab for such a long time. I would not forget the scene that we are beating the deadline, writing papers and joking around.

Finally, I would like to thank my parents for continuously loving me for many years without any return. Your love just like the rain and sunshine that help me grow happy and healthy. Thanks!

April 2018

NOVEL STATIC TIMING ANALYSIS APPROACH FOR

CUSTOMIZED DESIGN FABRIC

Jiajia Wang, MSEE
The University of Texas at Dallas, 2018

Supervising Professor: Carl Sechen

Synopsys' PrimeTime is widely used for static timing analysis and timing signoff solutions nowadays. However, PrimeTime has its two limitation. Firstly, it is unable to work on some customized fabric where the circuit consist not only the standard cells and wires as in conventional circuit. Secondly, PrimeTime reports the timing of the design not as accurate in deep nano-technology. The error could sometimes be as large as 15% compare to the real case. In this article, we proposed a novel STA approach that combines the HSpice simulation and PrimeTime analysis to provide timing accuracy that is comparable to HSpice. This approach is very flexible that could work on any non-standard customized circuit including SRAM cell or customized circuit that containing pass transistors. The thesis also introduced a customized design circuit named Field Programmable Transistor Array (FPTA) as test case to practically proved the feasibility of the new STA approach. The simulation result shows that the proposed approach could work on non-standard circuit and provide the accuracy that is extremely close to Spice simulation.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

ABSTRACT v

LIST OF FIGURES viii

LIST OF TABLES x

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 STATIC TIMING ANALYSIS OVERVIEW 2

2.1 Static/Dynamic Timing Analysis 2

2.2 Synopsys PrimeTime STA 2

2.3 Timing Path 2

2.4 Delay Calculation 4

2.5 Example of Delay Calculation 5

CHAPTER 3 FIELD PROGRAMMABLE TRANSISTOR ARRAY (FPTA) 6

3.1 Introduction of FPGA 6

3.2 Conventional FPGA Architecture 6

3.3 FPTA Architecture 7

3.4 Switch Box 8

3.5 Example of a Mapped Circuit on FPTA 9

CHAPTER 4 MOTIVATION FOR IMPLEMENTING THE STA APPROACH 12

4.1 Two Reasons to Implement the New Approach 12

4.2 Inaccuracy of NLDM Model and Complex CCS Model 12

4.3 Limitation on Simulating Customized Architecture 13

CHAPTER 5 NOVEL STA APPROACH FLOW 15
5.1 Introduction of the Novel STA Approach 15
5.2 Step 1 – Levelization 15
5.3 Step 2 – Targeting the Driver Receiver Pair 16
5.3 Step 3 – Generating a New Design 18
5.4 Step 4 – HSpice Simulation and Waveform Saving 20
5.5 Step 5 – Generating the New Library 22
CHAPTER 6 TESTED DESIGN AND SIMULATION RESULT 26
6.1 Test Design 26
6.2 Step 1 – Levelization 27
6.3 Step 2 – Targeting the Driver Receiver Pair 28
6.4 Step 3 – Generating a New Design 29
6.5 Step 4 – HSpice Simulation and Waveform Saving 31
6.6 Step 5 – Generating the New Library 33
CHAPTER 7 PRIMETIME RESULT 35
7.1 PrimeTime Running Procedure and Script 35
7.2 PrimeTime Result 36
7.3 Result Accuracy and Error Percentage 37
CHAPTER 8 CONCLUSION 39
REFERENCES 40
BIOGRAPHICAL SKETCH 41
CURRICULUM VITAE

LIST OF FIGURES

Figure 2.1 Timing Paths in a Sequential Logic Design....3

Figure 2.2 Timing Paths in a Combinational Cloud....4

Figure 2.3 Combinational Logic Cloud with RC network....5

Figure 3.1 Conventional FPGA Structure....7

Figure 3.2 Boolean Logic of a Conventional FPGA and its LUT Structure....7

Figure 3.3 A Basic Logic Unit of the FPTA (left) and the Logic Unit of Mapping a NOR3 Gate (right)....8

Figure 3.4 Switch Box Architecture in FPTA....9

Figure 3.5 Architure of Logic Unit with a Mapped Circuit....10

Figure 3.6 Gate Level Circuitry on a FPTA Mapping Architecture....11

Figure 4.1 NLDM Diver Receiver model (left) and CCS Driver Receiver Model (right)....12

Figure 5.1 A Levelized Gate Level Circuit....17

Figure 5.2 Driver Receiver Pair in FPTA....17

Figure 5.3 Driver Receiver Pair in Original FPTA....18

Figure 5.4 Driver Receiver Pair in New Generated Design....18

Figure 5.5 Diver Load Pair in FPTA Fabric with Saved PWL Waveform....21

Figure 5.6 Two Input Signals with Same Slew Rate can Result in Different Delay at the Output....21

Figure 5.7 Timing Table in a NLDM Library....23

Figure 5.8 Timing Table in a Newly Generated NLDM Library....24

Figure 6.1 Gate Level Verilog Code of the Test Design....26

Figure 6.2 Schematic of the Test Design....26

Figure 6.3 Levelized Test Design....27
Figure 6.4 Saved Results of Levelized Design 27
Figure 6.5 Grouped Driver Load Pair in the Test Design 28
Figure 6.6 Driver Load Pair File 28
Figure 6.7 Original Circuit of the Testing Design....29
Figure 6.8 Newly Generated Testing Design 30
Figure 6.9 Gate Level Netlist of the Original Testing Design 31
Figure 6.10 Gate Level Netlist of the Newly Generated Testing Design....31
Figure 6.11 Example of a Simple FPTA Design Architecture....32
Figure 6.12 New Generated Timing Library of Test Design 34
Figure 7.1 PrimTime Input TCL Script 35
Figure 7.2 PrimeTime Report of the Test Design 36
Figure 7.3 Test Design Imported into the PrimeTime....36

LIST OF TABLES

Table 2.1 Different Timing Paths in a Design ……..……3

Table 7.1 Gate Delay using HSpice and PrimeTime and the Delay Difference Percentage …...38

CHAPTER 1

INTRODUCTION

Synopsys' PrimeTime static timing analysis tool provides a single, golden, trusted signoff solution for timing, signal integrity, power and variation-aware analysis. It is one of most popular and trusted STA tool in industry in recent years. However, PrimeTime is unable to be used in some custom design circuitry. For example, If the connection wires in a custom design has pass gates built in, PrimeTime can hardly report the wire delay from .spef file. It is because there is no fixed delay value for a transistor or a pass gate. On the other hand, even though people trust the timing solution from PrimeTime, it's not always as accurate as expected. Sometimes, the error can be as large as 15% compare to the golden HSpice simulation. Due to these motivations, a novel STA approach is proposed in the article. Unlike the conventional method which takes the gate delay from the cell library and also calculates the wire delay from the timing information in .spef file, the new approach combined the wire and its driver, which are obtained from HSpice simulation, delay together. By importing the delays of all the drivers and its connecting wire into PrimeTime, an accurate timing report of the design will be reported from PrimeTime. In Chapter 2, the conventional STA background was introduced. Chapter 3 gives an example of a custom circuit design which is unable to be used by conventional STA method for timing sign off. Chapter 4 lists two motivations for implementing the novel STA approach. Chapter 5 splits the new approach into five steps and explains each step in detail. In Chapter 6, a test case is given to run the five steps and to verify the feasibility of the new approach. Chapter 7 gives the implemented result of the new approach and compares it with the golden HSpice result.

CHAPTER 2

STATIC TIMING ANALYSIS OVERVIEW

2.1 Static/Dynamic Timing Analysis

Timing Analysis is a critical part in VLSI design flow. The circuit can be analyzed in a way of dynamic or static. Dynamic Timing Analysis (DTA) check the functionality of a design by applying input vector and verifies the output vectors. It runs multiple simulations for logic operation thus timing consuming. While Static Timing Analysis (STA) is a fast method to compute and validate the timing performance on digital circuits. It computes the worst case delay on critical path and report timing violations by checking all possible paths in a design. STA does not verify the design functionality.

2.2 Synopsys PrimeTime STA

Synopsys' PrimeTime static timing analysis tool provides a single, golden, trusted signoff solution for timing, signal integrity, power and variation-aware analysis. It is one of most popular and trusted STA tool in industry in recent years. In PrimeTime, people can perform design checks includes setup/hold violation, data-to-data timing constrains, worst case delay on critical path, design rules, minimum clock period and so on.

2.3 Timing Path

PrimeTime first breaks down the design into different timing paths. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point. As the Table 2.1 shows, the starting point can be an input port or a clock pin of a sequential element. The

end point can be a data input of sequential element or an output port. Figure 2.1 shows an example of timing path in a sequential logic design with four paths in total. Path1 starts from an input port A and ends at data input port D of a sequential element D Flip Flop (DFF). Path2 and Path3 has the start point of DFF Clock pin and has end point of DFF input port and Output port respectively. Path4 starts the same point as Path1 but ends at an output port Z. The logic block in the diagram indicates a combinational logic cloud which usually contains multiple paths. The figure 2.2 shows an example of a combinational logic cloud that contains two paths. One path goes from an invertor to a NOR gate. Another path starts from the same invertor goes through a NAND gate and fed into the NOR gate. Obviously, the second path is longer than the first path because it contains one extra gate delay. PrimeTime will take the longer path to the consideration for maximum delay calculation. In contrast, the shorter path will be used for the minimum delay calculation.

Table 2.1. Different Timing Path in a Design

Path	Startpoint	Endpoint
Path 1	Input port	Data input of sequential element
Path 2	Clock pin of a sequential element	Data input of a sequential element
Path 3	Clock pin of a sequential element	Output port
Path 4	Input port	Output port

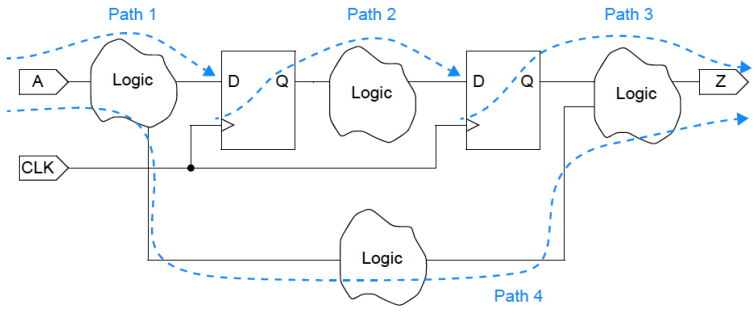

Figure 2.1. Timing Paths in a Sequential Logic Design

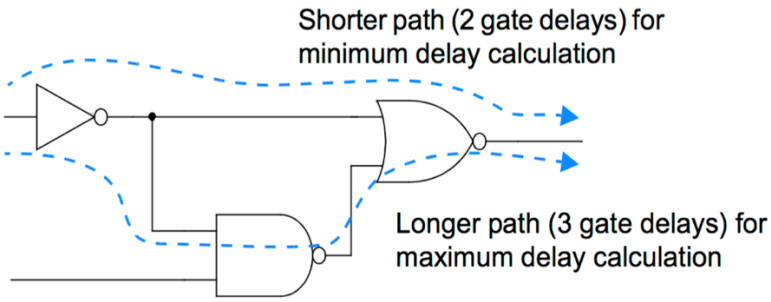

Figure 2.2. Timing Paths in a Combinational Cloud

2.4 Delay Calculation

After breaking a design down into different timing paths, PrimeTime work on the paths independently and calculate the delay along each path. When calculating the path delay, PrimeTime take the total delay of cell delay and net delay in the path. Cell delay is the mount of delay from the input of a gate to the output of a gate. PrimeTime usually reads a cell delay from a look-up table in a cell library. Typically, the cell delay in the table is a function of input transition and output load. PrimeTime is able to look for the delay in the table based on a giving set of input transition and output load. As the delay typically linearly increases with the increasing of these two factors, the table provides accurate delay by interpolating or extrapolating among the delay values according to these two factors. The Non-Linear Delay Model (NLDM) library will be introduced later in the paper.

Net delay is the amount of delay on a wire that connect the output of a gate to the input of next gate. For post layout timing analysis, PrimeTime uses specific time values back-annotated from a Standard Delay Format (SDF) file or uses detailed parasitic resistance and capacitance data back-annotated from Standard Parasitic Exchange Format (SPEF), Galaxy Parasitic Database

(GPD), Detailed Standard Parasitic Format (DSPF), Reduced Standard Parasitic Format (RSPF) file to calculate the net delay.

2.5 Example of Delay Calculation

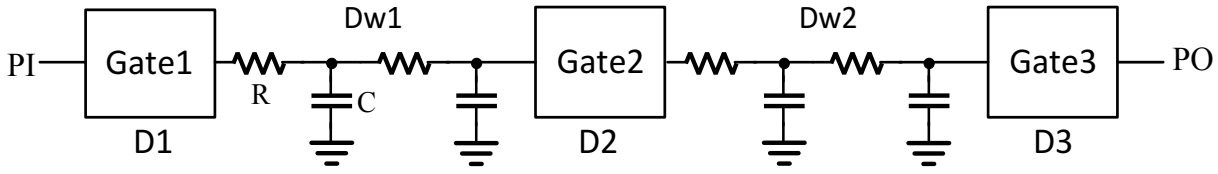

Figure 2.3. Combinational Logic Cloud with RC Network

Figure 2.3 shows how PrimeTime calculates the delay in a combinational logic cloud. From primary input (PI) to primary output (PO), signal passes three gates and two nets. In this example, we used NLDM library to have the cell delay D1, D2 and D3 by specifying the input transition at PI and the loads connected to each cell. We also get the parasitic resistance and capacitance (which forms pi model) from SPEF file for net delay calculation. PrimeTime has its own secret algorithm to work on those pi models and get the net delay Dw1 and Dw2. Thus, the total delay in the example design is the sum of all cell delay and net delay which is D1+D2+D3+Dw1+Dw2.

CHAPTER 3

FIELD PROGRAMMABLE TRANSISTOR ARRAY (FPTA)

3.1 Introduction of FPGA

In this chapter, a Novel Field Programmable Transistor Array (FPTA) architecture is introduced. The FPTA works similar to conventional Field Programmable Gate Array (FPGA) whose function is to prebuilt the logic blocks and program the routing resources. It is widely used nowadays in industry because of its reprogrammable feature which provide the capability for prototyping ASIC design before tape out. FPGA also provide a capability to update the hardware remotely after the product is sold, a feature that is becoming popular in customer product. Conventional FPGA implement logic blocks by configuring look up tables (LUTs) in Configurable Logic Blocks (CLBs).

3.2 Conventional FPGA Architecture

The left of Figure 3.1 shows the basic architecture of a FPGA. The I/O blocks located on the outer edges that encircles the inner blocks, where blue squares are the interconnection switches and white squares are logic blocks. By looking at a detailed graph on the right, the interconnected switches can be programmed to form the routing path among the logic blocks. Logic blocks are the basic memory element, it can be configured and stores the Boolean combinations of the target cells.

Figure 3.2 shows an example of LUT structure in a conventional FPGA. It takes Boolean combinations of three inputs a, b and c in the truth table as the address of the SRAM cell and stores

the correspond output y value in the memory. The inputs then act as a select signal in the multiplexer to choose the correct output value as current output state.

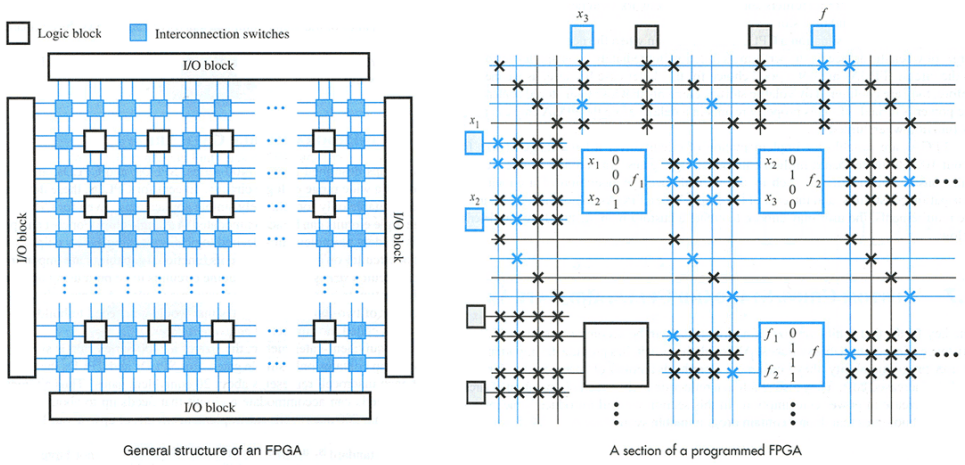

Figure 3.1. Conventional FPGA Structure

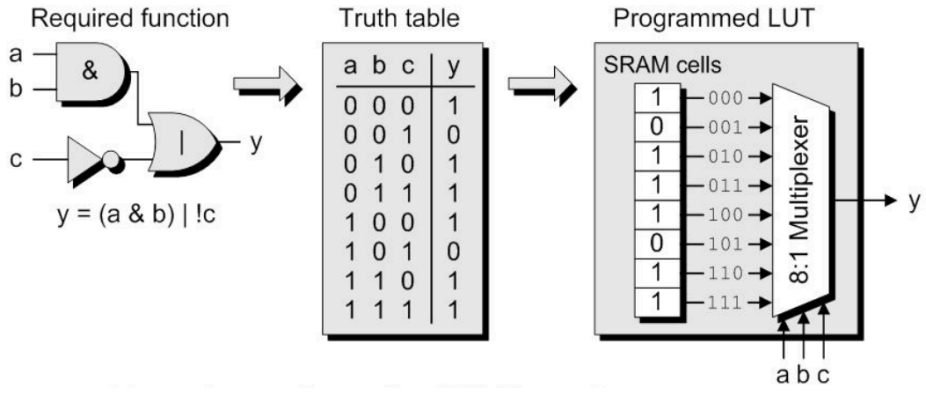

Figure 3.2. Boolean Logic of a Conventional FPGA and its LUT Structure

3.3 FPTA Architecture

Instead of utilizing the conventional LUT as Configurable Logic Block (CLB), the basic logic unit of the novel fabric FPTA consists of carefully arranged rows and columns of transistors as CLB. Each transistor can be carefully programmed to form the target logic function. As Figure 3.3

shows, the FPTA basic unit contained twelve PMOS at top and twelve NMOS at bottom which are colored in black. Each of these blacked transistor can be programmed either logic high or low to turn on or off. There are another eight inner transistors colored blue can only be programmed with control signal to transmit or isolate signals. The rest of transistors can be fed in with both input signal or control signal. A NOR3 example is shown on the right of Figure 3.3, the input signal A, B and C are mapped on the selected PMOS and NMOS. Some intermediate pass gate transistors are needed to be properly programmed to necessarily connect the selected transistors. The advantage of the novel FPTA CLB architecture is its flexibility, which makes possible of utilizing much less number of transistors and area to implement the same design, compared to conventional FPGA which has to consume the whole LUT to implement relative simple gate.

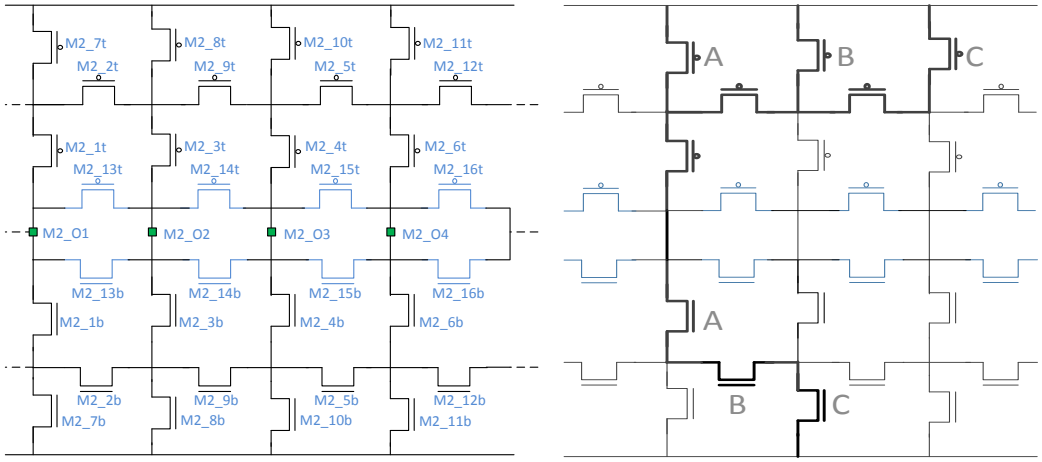

Figure 3.3. A Basic Logic Unit of the FPTA (left) and the Logic Unit of Mapping a NOR3 Gate (right)

3.4 Switch Box

Switching boxes which is used to configure the CLB is put on top and bottom of CLB. The switching boxes consists of metals and transistor. Only NMOS transistors are employed to connect

metals, which largely reduced area of the switch box. Keepers are necessary to keep the full logic level on the wire because of the use of NMOS. Figure 3.4 shows the example of the switch box floor plan. Each dot at the crossing point of two metals refers to a switch which is a NMOS transistor. When a switch was programmed with a logic high, two metals was connected through the drain and source of the NMOS transistor. To reduce the routing congestion and maximized utilization of area, different metal layer from metal 2 up to metal 4 are used.

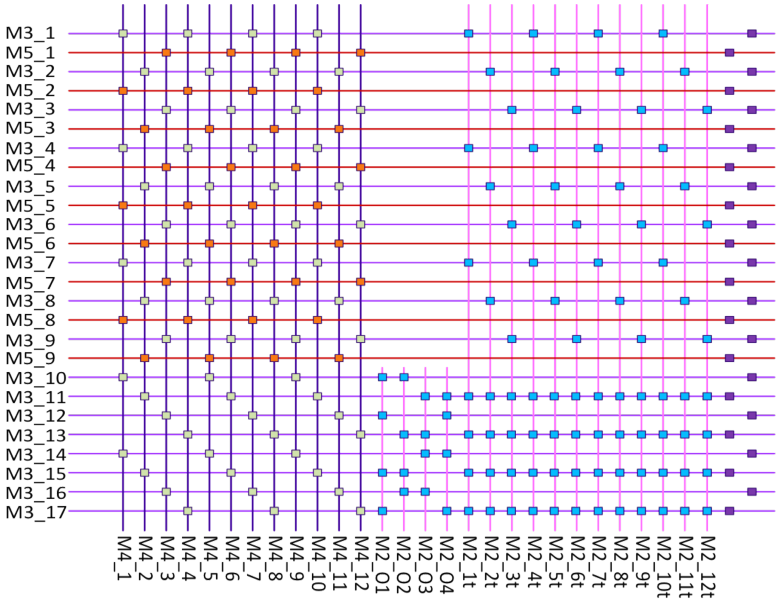

Figure 3.4. Switch Box Architecture in FPTA

3.5 Example of a Mapped Circuit on FPTA

Fig. 3.5 shows a simple combinational logic consists of a NAND2 followed by an inverter and the routing paths. The NAND2 gate occupies two columns in CLB while inverter occupies one. The output pin of NAND2 located in the middle of CLB. NAND2 output signal transmit to the next stage through two paths. One path goes up through upper switch box and fed into PMOS

of the inverter, the other goes down through bottom switch box and fed into the NMOS of the inverter.

Figure 3.6 is the equivalent gate level circuitry extracted from the CLB after mapping the logic gate into the FPTA shown in Figure 3.5. The output of the NAND2 go through two paths. The upper path goes through the upper switch box and feed into the PMOS of the inverter, while the lower path connects into NMOS. The router decides the signal routing and the number of pass transistor on each path. However, to minimize the delay from driver to its load, the algorithm is designed to equalize the delay on the two paths as much as possible.

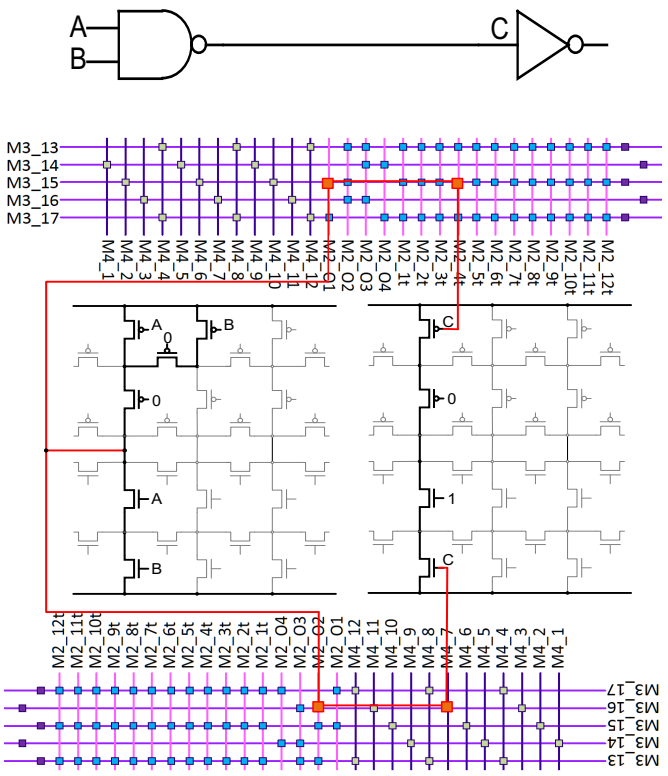

Figure 3.5. Architecture of Logic Unit with a Mapped Circuit

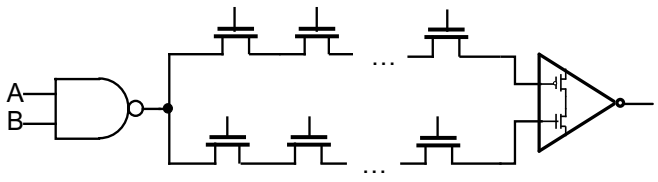

Figure 3.6. Gate Level Circuitry on a FPTA Mapping Architecture

CHAPTER 4

MOTIVATION FOR IMPLEMENTING THE STA APPROACH

4.1 Two Reasons to Implement the New Approach

The reasons we cannot simply use Synopsys PrimeTime to do the Timing Analysis can be concluded into three main aspects. Firstly, to ensure the delay accuracy, Composite Current Source (CCS) model library is used rather than Non-Linear Delay Model (NLDM) library for advance technology. However, CCS model saves the real simulation waveforms in the library which largely increase the library complexity and requires more memory usage. Thus, the new approach we proposed combines the pros in both CCS model and NLDM model. It uses the NLDM library format which allows the less complexity and less memory usage. At the same time, by using different methodologies, it increases the accuracy of the library dramatically. Secondly, PrimeTime is not universal to be used for timing sign off in some design such as SRAM design or some customized circuitry which has customized elements rather than standard cells in a conventional circuit design.

4.2 Inaccuracy of NLDM Model and Complex CCS Model

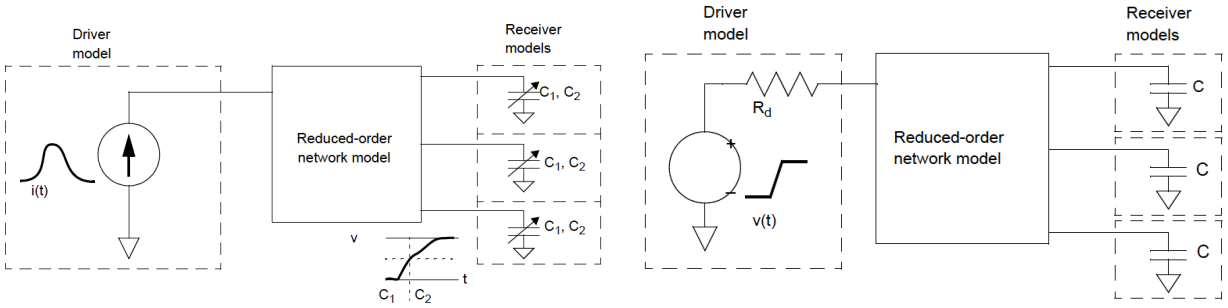

Figure 4.1. NLDM Diver Receiver model (left) and CCS Driver Receiver Model (right)

NLDM is widely used in PrimeTime, however it is not as accurate as expected as it fails to taken into account the wire resistor and gate miller effect. NLDM is an earlier established model that uses a linear voltage ramp in series with a resistor as the driver. The receiver model is a capacitor that represents a capacitive load which can be applied with different capacitive values. However, miller effect induces capacitance change at the receiver model, thus, a fixed capacitance load in NLDM model unable to carry out an accurate delay due to miller effect. To increase the accuracy in the cell delay, the Composite Current Source (CCS) model library is used for nanometer technologies rather than Non-Linear Delay Model (NLDM) library. CCS model uses timing-varying current source driver which is able to accurately handle high-impedance nets and other non-monotonic behavior. Moreover, its receiver model uses two different capacitor value which provides better approximation of the Miller Effect. Even though CCS model improves the accuracy of the delay information, its library model saves waveforms of current source behavior and more lookup tables due to different capacitive loads, which also dramatically increases the library complexity and slowdown the simulation speed.

4.3 Limitation for Timing Sign Off in Customized Design

An important limitation of the conventional timing analysis method is that they can hardly provide timing for customized design. A simple example is the FPTA design introduced in Chapter 3. In Figure 3.6, there are two signal paths, which consists of metal connection and pass gate transistor, between the output of the NAND2 and input of the inverter. This unique architecture makes it difficult for PrimeTime to analyze. Since the signal is split into two independent paths, two signals will propagate with different speed before they merge together again. The correlation at the end of two signal paths with different delay make it difficult to model accurately. In addition,

pass transistor with nonlinear resistance are needed in the signal path for the programmable purpose. The modeling of the pass transistor is also challenging for Primetime. Since it has no drivability, it cannot be considered as a gate. Another way of modeling it is to count it as a RC network like RC network in the SPEF file. If there are pass transistors connect among the signal path as figure shows, we would need to provide PrimeTime the Resistance value and Capacitance value of the pass transistor as well. However, the nonlinearity of the pass transistor makes it impossible to model it accurately. The resistance of the pass transistor in the linear region is shown below:

$$\mathrm{Ron} = \frac{1}{\mu \mathrm{Cox} \frac{\mathrm{W}}{\mathrm{L}} (\mathrm{VGS} - \mathrm{VT})}$$

While in Saturate region the resistance becomes:

$$\mathrm{Ron} = \frac{1}{-\frac{\mathrm{VDD}}{2}} \int_{\mathrm{VDD}}^{\frac{\mathrm{VDD}}{2}} \frac{\mathrm{V}}{\mathrm{IDSAT}(1+\lambda \mathrm{V})}\, \mathrm{dV} \approx \frac{3}{4} \frac{\mathrm{VDD}}{\mathrm{IDSAT}} \left(1 - \frac{7}{9}\lambda \mathrm{VDD}\right)$$; Where $\mathrm{IDSAT} = \frac{\mathrm{W}}{\mathrm{L}} [(\mathrm{VDD} - \mathrm{VT})\mathrm{VDAST} - \frac{\mathrm{V2DAST}}{2}]$

From the equation above, it's clearly that the transistor resistance, Ron, is highly depending on the voltage on its gate and source and the operation regions of the transistor. No fixed resistance value could be found for the pass transistor since its operation go through all the nonlinear condition. The capacitance of the pass gate is also very nonlinear. These nonlinearities not only affect the propagation delay, but also contribute the distortion to the waveform making the calculation inaccurate.

CHAPTER 5

NOVEL STA APPROACH FLOW

5.1 Introduction of the Novel STA Approach

Due to limitations described in Section 3, a novel STA approach, which is able to simulate custom design accurately and is compatible with Primetime, was researched, developed and implemented. It utilizes industry 'golden standard' HSpice simulator to accurately measure the delay of different path between the driver and receiver, then collect and sort the simulation waveforms and delay information to create a new cell library for Primetime. The design is then manipulated into a new design that contains the library gates only from the newly made cell library. PrimeTime is able to do the STA by analyzing the new design and reading the delays from new library. The approach is split into five steps. The following sections will introduce each steps in detail.

5.2 Step 1 – Levelization

Levelizing a design is a procedure to rearrange a digital design such that the lower level components always propagate to a higher level component. In such a way, the design can be analyzed and simulate level by level to avoid unnecessary repeated evaluation. It largely simplifies the simulation orders thus improve speed.

The simple example showed in the Figure 5.1 is a levelized gate level circuit. X1 and X2 need to be ready before G3 get simulated. In other words, G1 and G2 need to be simulated before G3. Such process that determines the order of simulation is what we called levelization. Levelization assigns the level number to each gate and the corresponding nets connecting to it. The algorithm

computes the level of a gate by adding one to the maximum level number of its input nets. The nets connect to the primary input pins will always be labeled as level 0 and the nets that connect to the output pin of a gate will have the same level number as the gate.

For example, in Figure 5.1, all the nets that connect to the primary inputs A, B, C, D and E was labeled as level 0. Since G1 and G2 have the inputs only from level 0 nets, the level of G1 and G2 will be labeled as one (which is 0 plus 1). Net X1 and X2 will follow the same level number as its driver gate G1 and G2, thus they are both grouped into level 1. G3 has one level higher than X1 and X2, so it belongs to level 2 and X3 has the same level as G3. The level of G4 is one level higher than the maximum level number of its input nets, thus G4 is level 3, which is one level higher than X3. In a similar way, G5 labeled as level 3.

Levelization not only sort the gate in the order that help speeding up the simulation process, but is also an indispensable step for our new approach. By levelizing the design, a gate has its inputs only from its lower level. In this way, the outputs from its previous stage can be directly fed into the current stage as input signals. It dramatically increases simulation accuracy on our customized design architecture. More details will be explained in the following chapter.

5.3 Step 2 – Targeting the Driver Receiver Pair

The purpose of step 2 is to find out all the driver and its receiver pairs in a design. The other advantage of levelizing a design is the contribution to this step. After levelization, we can easily find the driver-receiver pair level by level. It saves a lot of time since the receiver is always in one or more lower level than its receiver. If a gate is defined as a driver, then all its fan-out gates are defined as its receivers. We take the driver and one of its receivers as a driver-receiver pair. This step is repeated until all the receivers are grouped together with the driver as a pair.

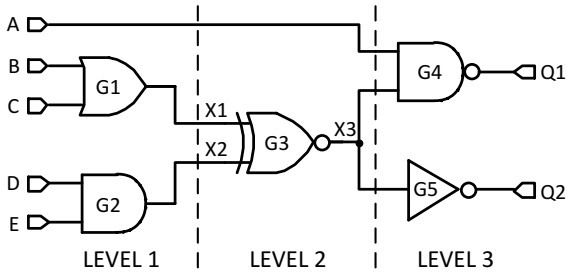

Figure 5.1 A Levelized Gate Level Circuit

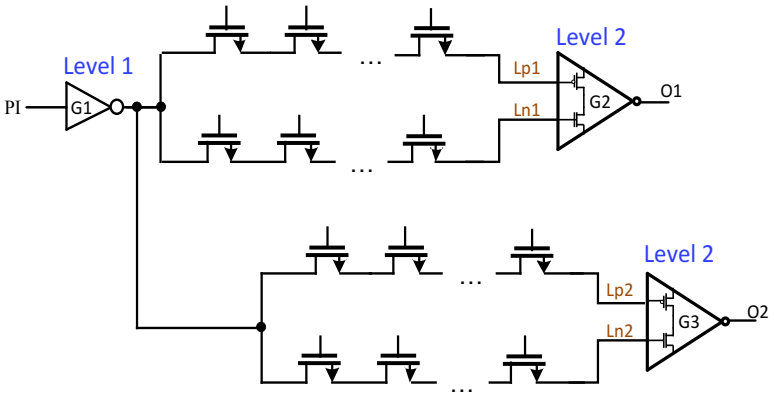

Figure 5.2 Driver Receiver Pair in FPTA

Take the Figure 5.1 again as an example. If G1 is consider as a driver, then its fan-out load G3 will be its corresponding receiver. G1-G3 will be a driver receiver pair. Another driver-receiver pair is G2-G3 if G2 is consider as a driver. In the similar way, if G3 is now consider as a driver, its fan-out G4 and G5 are its receivers. Thus, G3-G4 and G3-G5 are two independent driver receiver pairs. Another example is shown in Figure 5.2, which is a real circuit of a three cascaded inverters that map into the FPTA fabric. Obviously, G1 is in level 1 and G2, G3 are in level 2. As discussed in previous chapter, FPTA has special architecture that the signal was split and transmit through two paths, P-path and N-path, and then was fed into pmos and nmos of the next gate respectively. If we ignore the P-path and N-path, the inputs of G2 and G3 are actually connected

to the output of G1. If we consider G1, who is at level 1, as a driver, then G2 and G3 should be considered as the receivers of G1. In this design, there are totally two driver-receiver pairs, that is G1-G2 and G1-G3.

5.3 Step 3 – Generating a New Design

After we have found all the driver-receiver pairs in a design and the gates interconnect, the next step is to generate a new gate level design based on the information we have in step 1 and step 2. The new generated design will replace the old gate level design and will be inputted into PrimeTime for timing analysis. This is also a crucial step in the proposed approach.

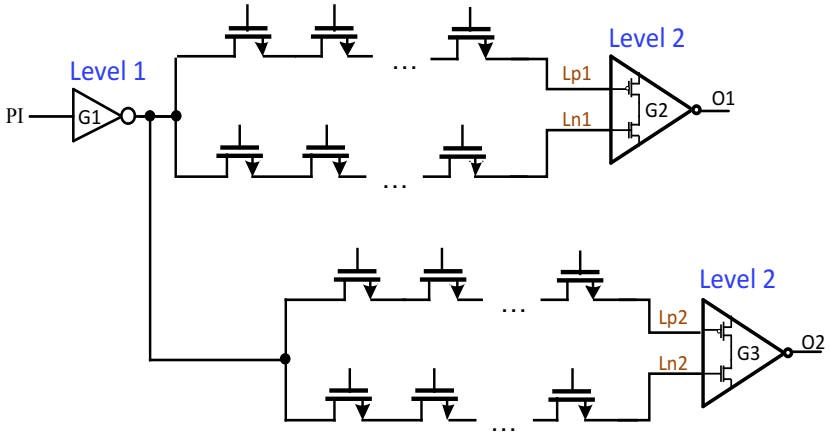

Figure 5.3 Driver Receiver Pair in Original FPTA

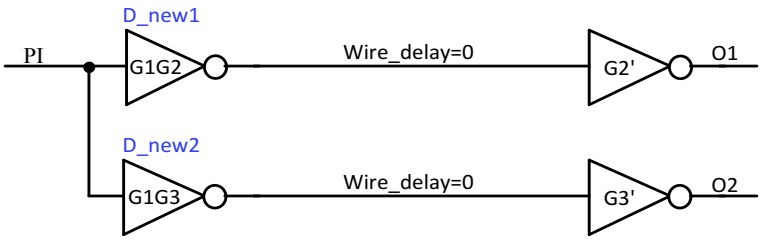

Figure 5.4 Driver Receiver Pair in New Generated Design

The generation of the new design starts from the first level driver-receiver pair in the original levelized design by copying the driver as its original cell type and rename it with a different and unique name to differentiate with all other cells that being copied in the new design. Then rename the wire that connected to the output pin of the copied driver. Repeat the step until all the driver-receiver pair is being copied and renamed.

A simple example of this step would be an inverter driver that connects to two inverter loads as shown in Figure 5.3. In step 2, the driver G1 was paired with G2 and G3 respectively as two driver-receiver pairs. The first step to generate the new design is to copy inverter G1 and rename it as 'G1G2' as shown in Figure 5.4. The new cell name "G1G2" is the combination of the original driver names the corresponding receiver name. The wire connected to G1 will be renamed to 'G1_G2'. The 'G1_G2' name will be used for all the wires that connected to the output of G1G2. Since G2 directly connects to the primary output O1, it will be copied and rename as G2'. Similarly, receiver and driver pair G1-G3 could be reorganized to another path. The copied inverter driver will be named as G1G3 and the wire connection will be renamed as 'G1_G3' as also shown in Figure 5.4. As G3 is connected to the primary output O2, it is renamed as G3' in the new design and is still connected to the output O2. The gate level verilog netlist for the original design is shown below:

```
Module old_design (PI, O1, O2);
        input PI;
        output O1, O2;
        wire wire;
        INV G1 ( .A(PI), .O(wire));
        INV G2 ( .A(wire), .O(O1));
        INV G3 ( .A(wire), .O(O2));
endmodule
```

After being processed through Step 3, the new generated gate level verilog design netlist is as following:

```
Module new_design (PI, O1, O2);
        input PI;
        output O1, O2;
        wire G1_G2, G1_G3;

        INV G1G2 ( .A(PI), .O(G1_G2));
        INV G2' ( .A(G1_G2), .O(O1));
        INV G1G3 ( .A(PI), .O(G1_G3));
        INV G3' ( .A(G1_G3), .O(O2));

Endmodule
```

Compared to the old design, each driver load pairs have its own signal path in the new design. For example, In the old design, G1 is a shared driver of G2 and G3. In the new design, each load has its own driver G1G2 and G1G3. The newly generated driver will store the path delay of the original driver-receiver pair from HSpice simulation, which provides PrimeTime the accurate path delay information that is needed in STA.

5.4 Step 4 – HSpice Simulation and Waveform Saving

The new design circuitry is generated in Step 3, which needs the delay information from the original design to complete the modeling. In this step, we simulate the design using HSpice to collect delay information of each driver load on each signal path in the old design. When simulating the driver load pair of the first level in levelized design, we input the ideal Piece Wise Linear (PWL) at primary input port. And then simulate and record the output waveform at load input pin. This saved waveform will be used as input waveform for the simulation on driver load pair in second level. Vice Versa, the design will be simulated level by level from primary inputs to primary outputs.

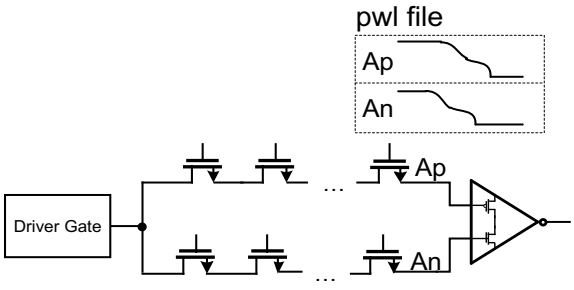

Figure 5.5 Diver Load Pair in FPTA Fabric with Saved PWL Waveform

Figure 5.5 shows a driver-receiver pair example on our FPTA fabric. We assume the gate of the driver in the first level is connected with primary input pins. As our customized fabric has unique design feature as shown in Figure 5.5, the output signal of the driver was split into two paths where the Ap path connects to pmos of the load and the An path connects to the nmos of the load. However, PrimeTime only accept delay calculation on the single signal path. This algorithm requires the FPTA to be simulated on both signal paths and abandon the fasted one when considering the worst case delay calculation. Thus we measure the delay from input of driver gate to the gate of both pmos and nmos of the receiver inverter. And a Perl script will be run to choose the maximum delay and take it as the path delay of the driver load pair.

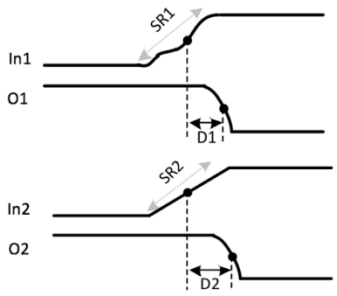

Figure 5.6 Two Input Signals with Same Slew Rate can Result in Different Delay at the Output

To improve the accuracy of delay obtained from HSpice simulation, the output PWL waveforms at An and Ap are saved in file after simulation and will be used as input signals for

simulation of next level when the inverter are acting as a driver. The reason is shown as in Figure 5.6. In1 is the real PWL waveform from early simulation and O1 is the corresponding output waveform while In2 shows the ideal PWLwaveform and O2 is its output waveform. SR1 and SR2 are the input slew rate for the In1 and In2 respectively, which are equal. D1 and D2 are the delays of the two cases. The real PWL signal has some distortion that may be caused from multiple reasons such as miller effect or design elements (e.g., Keeper in FPTA fabric). Even though they have the same input slew rates, the non-linear input PWL signal will result a different delay compare to the linear PWL. By using the real simulated waveforms from early stage, the distortion can be included and simulated. On the other hand, the signals traveling on p path and n path will arrives loads at different time, feeding in the signals with real delay difference can largely increase the accuracy for the simulation on next stage. Moreover, even though the NLDM library is used for gate delay calculation in the design, the HSpice simulation captured the Miller Effect which make the library as accurate as CCS model.

5.5 Step 5 – Generating the New Library

After we collect the delay information on each path, we are going to generate the library for the cells in the new design. The library model we use is Non Linear Delay Model. The NLDM library stores the delay information into lookup tables. It is a two-dimensional array where one array is input slew rate and the other one is output capacitance of the cell. The cell delay is then searched by the specific array values. An output timing arc related to an input pin contains four tables shown in Figure 5.7. Table ‘cell_fall’ and ‘cell_rise’ stores falling delay and rising delay of the cell, where falling/rising delay measures the delay from the 50% of the input signal to the 50% of falling/rising edge of the output signal. The other two table ‘fall_transition’ and ‘rise_transition’

represent falling transition time from 80% to 20% (which can be redefined by user in the library header) and rising transition time from 20% to 80% of the output waveform. The two-dimensional array ‘index_1’ and ‘index_2’ in each table refers to the different input slew rates and output capacitances respectively. The numbers in ‘values’ represents the delay information corresponding to different index number. If an index number is not listed in the lookup table, then the tool will interpolate or extrapolate to find the delay information based on the index number. If we want to know the rise transition of a cell with input slew rate 0.2727 and output load 0.01, let’s take the library in Figure 5.8 for example, the corresponding delay information in lookup table will be 0.06553. If we change the input slew rate from 0.2727 to some number between 0.2727 and 1.195, by interpolation, we will get the rise transition to be a number between 0.06553 and 0.08054.

```
timing() {
  related_pin : "A" ;
  timing_sense : negative_unate ;
  timing_type : combinational ;

  cell_fall(tmg_ntin_oload_5x2) {
    index_1("0.01, 0.2727, 1.195, 2.956, 5.7");
    index_2("0.01, 0.09");
    values("0.1505, 0.1536, 0.164, 0.1839, 0.2148",\
           "1.011, 1.014, 1.026, 1.045, 1.076");
  }

  cell_rise(tmg_ntin_oload_5x2) {
    index_1("0.01, 0.2727, 1.195, 2.956, 5.7");
    index_2("0.01, 0.09");
    values("0.1732, 0.1762, 0.1872, 0.2082, 0.2404",\
           "1.299, 1.303, 1.314, 1.334, 1.366");
  }

  fall_transition(tmg_ntin_oload_5x2) {
    index_1("0.01, 0.2727, 1.195, 2.956, 5.7");
    index_2("0.01, 0.09");
    values("0.05555, 0.05939, 0.07327, 0.09986, 0.1416",\
           "0.07812, 0.08105, 0.09078, 0.1124, 0.1488");
  }

  rise_transition(tmg_ntin_oload_5x2) {
    index_1("0.01, 0.2727, 1.195, 2.956, 5.7");
    index_2("0.01, 0.09");
    values("0.06127, 0.06553, 0.08054, 0.1092, 0.1536",\
           "0.08783, 0.09056, 0.1008, 0.1229, 0.1613");
  }
}
```

Figure 5.7 Timing Table in a NLDM Library

```
timing() {
  related_pin : "A" ;
  timing_sense : negative_unate ;
  timing_type : combinational ;

        cell_fall(tmg_ntin_oload_3x10) {
          index_1("0.01, 0.1 ,5");
          index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
          values("3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500",\
                 "3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500",\
                 "3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500, 3.20500");

      }

        cell_rise(tmg_ntin_oload_3x10) {
          index_1("0.01,  0.1 ,5");
          index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
          values("2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400",\
                 "2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400",\
                 "2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400, 2.25400");
      }

        fall_transition(tmg_ntin_oload_3x10) {
          index_1("0.01, 0.1 ,5");
          index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
          values("0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 2.18300",\
                 "0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100",\
                 "0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100, 0.86100");
      }

        rise_transition(tmg_ntin_oload_3x10) {
          index_1("0.01,  0.1 ,5");
          index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
          values("2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300",\
                 "2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300",\
                 "2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300, 2.18300");
      }
    }
  }
}
```

Figure 5.8 Timing Table in a Newly Generated NLDM Library

Figure 5.8 shows the new generated library for a modified cell. As mentioned, the timing information in the lookup table was generated from HSpice simulation. It is accurate and trusted as it includes all the noise factor such as cross talk and miller effect. When looking at the library, there is an interesting phenomenon that the timing in each lookup table does no change with

changing of the input slew rate and output loads. The is due to the novel approach when generating the library. When simulating the driver load pair, we have a fixed load for each driver load pair in a design. Thus, no matter what the 'index_2' value, we should always search for a fixed delay and a transition. Although the primary input transitions defined by user can varies, the transition delay on the higher level of driver load pairs are always fixed. That is because in our custom design, there are a long wire and repeaters between each driver and its load which is used to minimize the small transition difference of the signals.

In traditional timing library, as we increase the index vector range and reduce its interval, the accuracy of the timing information can be increased. However, for the new generated library, because of its unique characteristics as we discussed earlier, the size of the library could be largely reduced while keeping its accuracy unaffected. In this way, the proposed approach not only reduces the memory size, but also increases the efficiency when there is no complicated interpolation and extrapolation calculation involved.

CHAPTER 6

TESTED DESIGN AND SIMULATION RESULT

6.1 Test Design

Some test benches are run to verify the results of the proposed approach. This chapter will be focusing on the details of one test bench and its simulation result. At each step, some status of the test bench will be given and its comparison data from golden simulation will also be provided to show the performance of the proposed approach.

To make it easier to understand, a simple RTL design was chosen as the example. It contains only six standard cells: two invertors, two NOR gates, a NAND gate and a AOI21, which is shown in Figure 6.2. These gates are commonly used standard cells for synthesizing a design. Figure 6.1 shows the RTL gate level netlist and Figure 6.2 shows its schematic view.

```
module group_bench (a, b, d, e, in, out1, out2 );

   input a, b, d, e, in;
   output out1, out2;
   wire i, j, k, m, l, o1, o2, o3, o4;

   AOI21 G1 ( .A(k), .B(m), .C(e), .O(l) );
   NAND2 G2 ( .B(j), .A(i), .O(k) );
   INV G3 ( .A(a), .O(i) );
   NOR2 G4 ( .A(l), .B(m), .O(out2) );
   NOR2 G5 ( .B(d), .A(j), .O(m) );
   INV G6 ( .A(b), .O(j) );

endmodule
```

Figure 6.1 Gate Level Verilog Code of the Test Design

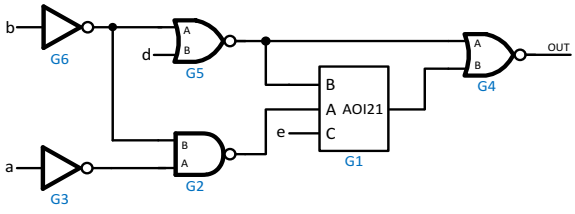

Figure 6.2 Schematic of the Test Design

From Figure 6.1 and Figure 6.2, we can easily see that the design contains four primary input pins a, b, d and e that is fed into different gate in different level. There is only one output pin 'out' connects to the output of the NOR gate. The six gates are named as G1, G2 until G6.

6.2 Step1 – Levelization

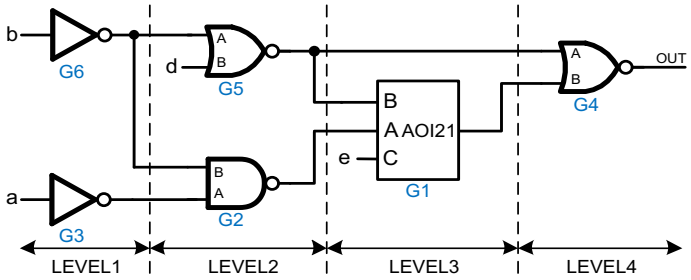

Figure 6.3 Levelized Test Design

As previous chapter described, the first step is to levelize the design into different levels from primary input to primary output. A Perl script is written up to take a gate level design, levelize it and save it to 'levelization.txt' file based on the connection between each gates. Figure 6.3 shows the result of the levelizing the testing design into four levels and Figure 6.4 shows what is inside the file 'levelization.txt' after running the Perl script.

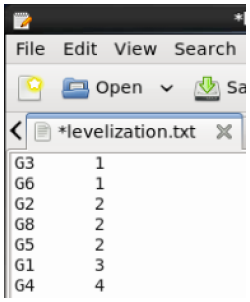

Figure 6.4 Saved Results of Levelized Design

From the Figure 6.4, we can easily see that there are totally four levels where G3 and G6 are labeled as Level 1, G2, G8 and G5 are labeled as Level 2, G1 and G4 are in Level 3 and Level 4 respectively.

6.3 Step 2 – Targeting the Driver Receiver Pair

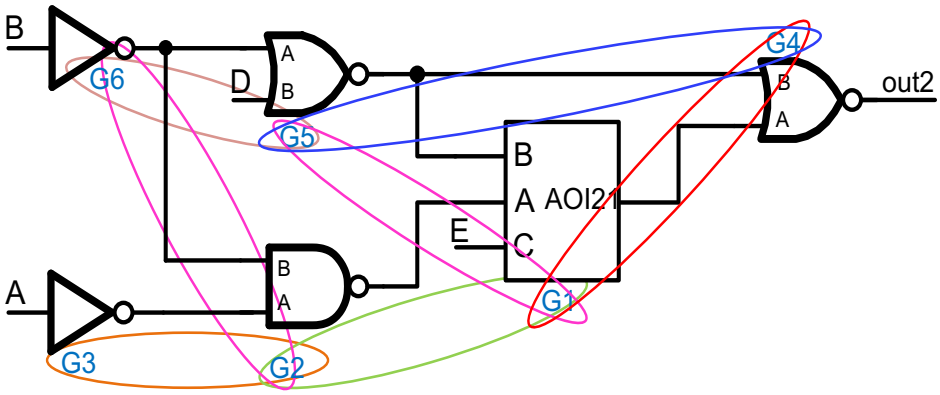

Figure 6.5 Grouped Driver Load Pair in the Test Design

*driver_load.txt

```
NOR2;G4;out2
INV;G3;G2|A
NOR2;G5;G1|B
NOR2;G5;G4|B
AOI21;G1;G4|A
NAND2;G2;G1|A
INV;G6;G2|B
INV;G6;G5|A
```

Figure 6.6 Driver Load Pair File

After levelization, we will simulate all the objects according to the levelized netlist. The objects are nothing but all the driver load pairs in the design. A Perl script can automatically look for the driver load pair in the design level by level from first level to the last one. Those driver

load pair will be saved in a temporary file called ‘driver_load.txt’. Figure 6.6 shows an example of a generated ‘driver_load.txt’ file and Figure 6.5 shows its symbol view of the grouped pairs.

From the graph of the symbol view, each color of the ellipse circles indicates one group of driver load pair. Obviously, there are seven different objects or pairs in this design. One should note that a gate can be the driver in different driver load pairs and a gate can be the load in different pairs as well. For example, G6 is the driver in both pair G6-G2 and G6-G5. This is because the output of G6 is connected two different load G5 and G2. Giving another example, G4 is both the load of pair G5-G4 and G1-G4 as the input A of G4 is connected to the output of G5 and the input B of G4 is connected to the output of AOI21. The contents in the ‘driver_load.txt’ contains three information and each information is separated by semicolon. The first column is the driver gate type, for example, INV, NOR2 or NAND2. The second column is the driver and third column is the load and the pin name that is connected to the driver. The gate name of the load and its pin name is separated by ‘|’. Let’s take the second row as an example, it shows that the driver gate G3 is an INV, and it is connected to the pin A of G2. Thus, this is a driver load pair G3-G2.

6.4 Step3 – Generate New Design (.v) and New Library (.lib) for STA

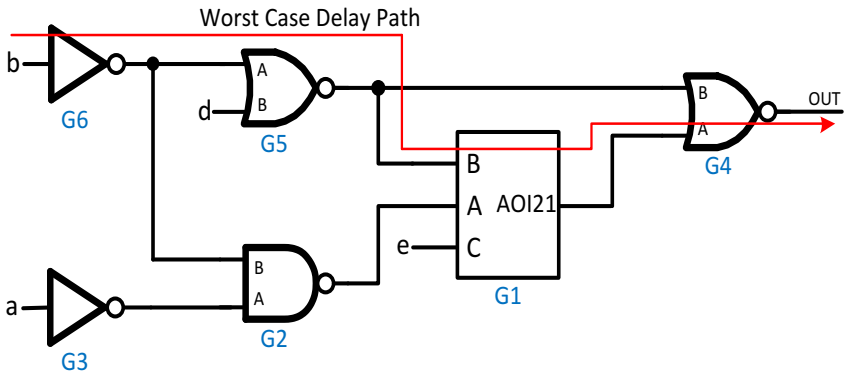

Figure 6.7 Original Circuit of the Testing Design

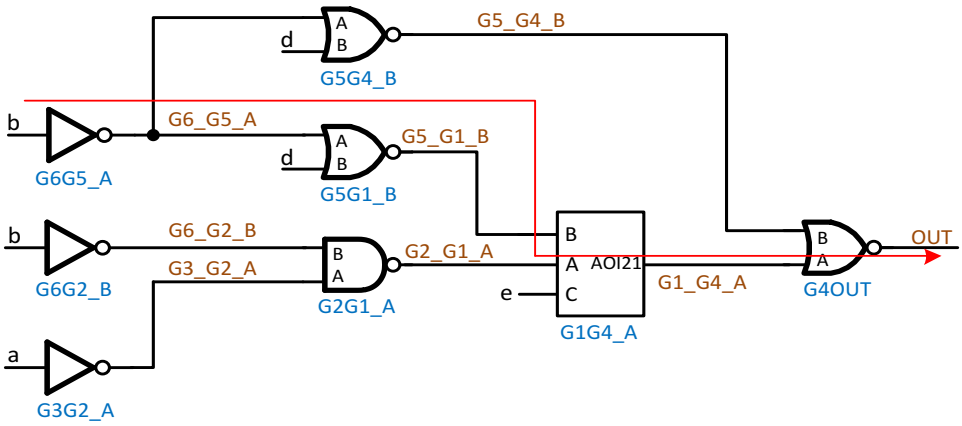

Figure 6.8 Newly Generated Testing Design

After we have all the driver load pairs listed, we will need to generate new design which has all the paths individually listed without any mutual driver and load. In such a way, the delay on each individual signal paths can be represented in the library of each driver gate.

The same test design is taken here as an example. Figure 6.7, let's take G6 in the original design as an example. From step 2, we know that driver G6 has two loads which give rise to the two driver load pair G6-G5 and G6-G2. Thus in the newly generated design as shown in Figure 6.8, the inverter G6 was copied twice and renamed as G6G5_A and G6G2_B. where the letter A stands for the pin name of the load that driver G6 connected to. This change converts a mutual signal path on G6 into two individual and orthogonal signal paths which allows the driver gates possess different delays. Let's take G1 as another example. Since G1 has only one load G4, thus in the new generated design, G1 was only copied once and named as G1G4_A. And it is still connected to the only load NOR2 named as G4OUT in the new design. Not only the gates has been copied and renamed in the new design, but also the wires that connects the driver and load has copied and renamed.

Figure 6.9 above shows the gate level netlist of original design. Figure 6.10 shows the gate level netlist of new generated design. We can see that all the gates and nets has been copied and renamed. The number of gates increases because each driver with multiple loads are turned into multiple paths.

```
module group_bench (a, b, d, e, in, out1, out2 );

   input a, b, d, e, in;
   output out1, out2;
   wire i, j, k, m, l, o1, o2, o3, o4;

   AOI21 G1 ( .A(k), .B(m), .C(e), .O(l) );
   NAND2 G2 ( .B(j), .A(i), .O(k) );
   INV G3 ( .A(a), .O(i) );
   NOR2 G4 ( .A(l), .B(m), .O(out2) );
   NOR2 G5 ( .B(d), .A(j), .O(m) );
   INV G6 ( .A(b), .O(j) );


endmodule
```

Figure 6.9 Gate Level Netlist of the Original Testing Design

```
module group_bench (a, b, d, e, in, out1, out2 );

   input a, b, d, e, in;
   output out1, out2;


      NOR2_G4out2  G4out2 (   .A(G1_G4_A), .B(G5_G4_B), .O(out2)  );
      INV_G3G2_A  G3G2_A (   .A(a), .O(G3_G2_A)  );
      NOR2_G5G1_B  G5G1_B (   .B(d), .A(G6_G5_A), .O(G5_G1_B)  );
      NOR2_G5G4_B  G5G4_B (   .B(d), .A(G6_G5_A), .O(G5_G4_B)  );
      AOI21_G1G4_A  G1G4_A (   .A(G2_G1_A), .B(G5_G1_B), .C(e), .O(G1_G4_A)  );
      NAND2_G2G1_A  G2G1_A (   .B(G6_G2_B), .A(G3_G2_A), .O(G2_G1_A)  );
      INV_G6G2_B  G6G2_B (   .A(b), .O(G6_G2_B)  );
      INV_G6G5_A  G6G5_A (   .A(b), .O(G6_G5_A)  );
endmodule
```

Figure 6.10 Gate Level Netlist of the Newly Generated Testing Design

6.5 Step 4 – HSpice Simulation and Waveform Saving

To obtain the worst case delay, the delay on each path between driver and load needs to be simulated. The delay path was obtained by HSpice simulation from the input driver to its load.

And then a delay library was generated based on the simulation result. This delay is nothing but the total delay of driver and the wire connected to the driver. Thus, this new generated delay library was used for giving the delay information of the gate in the new design.

Let's take the driver G6 in Figure 6.7 as example. First we need to simulate and obtain the delay from primary input b through G6 to the input A of G5. This delay will be used to represent the delay of G6G5_A in the new design. Since the wire delay is included into the cell delay, the wire G6_G5_A delay will be marked as 0. In such a way, the path delay in the original design was successful mimic to the new design and expressed in the library. Again, to get the delay of G6G2_B in the new design, we simulate the delay from primary input b through G6 to the input pin B of G2. And then mark the wire delay of G6_G2_B as 0. By repeatedly doing this until we obtain all the paths delay in the design, we will be able to create a new gate library for calculating the worst case delay in PrimeTime.

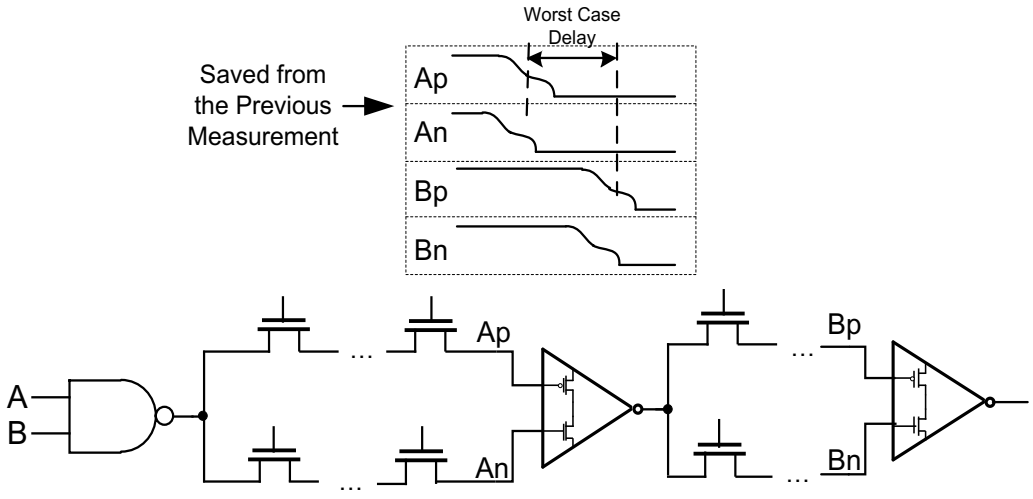

Figure 6.11 Example of a Simple FPTA Design Architecture

What need to be paid attention to is the architecture of our FPTA design. It is different from the ASIC design or conventional FPGA design. Figure 6.11 shows an example of the FPTA design

architecture. As we explained in Chapter 3, there are two signal paths between each driver and load. One signal path is connecting to the PMOS of the load and another one is connecting to the NMOS. This unique architecture requires the HSpice simulation on each path and taking the worst delay of the two. We also need to note that, as the two signal paths are not identical, the signals will arrive at the PMOS and NMOS of the load at different time. Thus, if we give the same input signal at the PMOS and NMOS of a driver pin, then the delay accuracy will be affected dramatically. To improve the delay accuracy, the simulation waveform at input of the load, An and Ap in Figure 6.11 for example, will be saved into a file. Those waveforms will be used as the input signals of simulation on next level where the load becomes a driver. After the simulation is done for a current level, the saved input waveforms can be deleted to save the memory, which is a benefit that levelization brings us. Without the levelization, the waveforms will need to be saved until the simulation of the highest level is done.

6.6 Step 5 – Generating the New Library

When the delay information was grasped from simulation on each level, the script runs to generate the new library for each gate on the same level immediately.

Figure 6.12 shows an example of the generated delay lookup tables. The tables contain rising/falling transitions of the gate and the rising/falling delays. This delay table eliminate the interpolation and extrapolation calculation when reading a delay number, since it is a fixed delay number regardless of the change in the input transition and output load. This is because there are always one or more fixed loads connecting to a driver during the simulation. What's more, due to the unique architecture where is always a long wire bridging each driver load pair, it gives rise to

a result that the transition of a signal arrives at a gate are always the same. These are the factors that makes the new library simple and easy to read.

```
cell_fall(tmg_ntin_oload_3x10) {
  index_1("0.01, 0.1 ,5");
  index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
  values("4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800",\
         "4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800",\
         "4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800, 4.19800");

}

cell_rise(tmg_ntin_oload_3x10) {
  index_1("0.01,  0.1 ,5");
  index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
  values("2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600",\
         "2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600",\
         "2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600, 2.68600");
}

fall_transition(tmg_ntin_oload_3x10) {
  index_1("0.01, 0.1 ,5");
  index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
  values("1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 2.52900",\
         "1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900",\
         "1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900, 1.55900");
}

rise_transition(tmg_ntin_oload_3x10) {
  index_1("0.01,  0.1 ,5");
  index_2("0, 0.003451, 0.01557, 0.0387, 0.07474, 0.1253, 0.1919, 0.2757, 0.378, 0.5");
  values("2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900",\
         "2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900",\
         "2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900, 2.52900");
}
```

Figure 6.12 New Generated Timing Library of Test Design

CHAPTER 7

PRIMETIME RESULT

7.1 PrimeTime Running Procedure and Script

```
set search_path "."

set library_file {fake_gate_try2.db}
set link_library $library_file
set target_library $library_file



read_verilog ./new_design.v

#read_parasitics -format SPEF  "no_tran.spef"
#read_parasitics -format SPEF  "no_tran_conly.spef"

set timing_slew_propagation_mode best_slew
set timing_report_unconstrained_paths true
#set_driving_cell -lib_cell "INV1" -input_transition_rise 0.263 -input_transition_fall 1.15
[all_inputs]

check_timing
report_timing -transition -delay  max_fall -capacitance -input_pins -net  -significant_digit 5
#report_timing -transition -rise_through G1G4_A/B  -capacitance -input_pins -net  -significant_digit 5
#source dump_timing.tcl
#source dump_slack.tcl
exit
```

Figure 7.1 PrimTime Input TCL Script

Up to now, we have finished the five steps and get all the data we need to import to PrimeTime. The next step is to import the new generated design (.v) from step 3 and new generated library (.lib) from step 5 into PrimeTime, then the PrimeTime will report the result for the critical path and worst case delay of the design. Figure 7.1 shows the TCL script to initialize the PrimeTime for the test case.

In the script, 'fake_gate_try2.db' is the database format of the new generated library. 'New_design.v' is the new generated Verilog design from the approach. From the script, we can see that there are no SPEF file included as the input since all the delay of the wires in the new

design was marked to zero. Thus, we do no need to consider any wire information in the SPEF here.

7.2 PrimeTime Result

Figure 7.2 shows the result of the PrimeTime for the test case. Figure 7.3 shows the gate level of the test case, on which the critical path was labeled with the red line. From the result form, we can get the information of gate fan-out, input cap, input transition, gate delay and the path delay. The critical path start from the input port b, pass through G5G1_B, G1G4_B, and ends at OUT port. From Figure, we can see that the delay of G6G5_A is 4.198ns, the delay of G5G1_B and G1G4_B are 2.951ns and 0.615ns respectively. The total delay or worst case delay for the test case is 8.2846ns. The delay of the nets connecting to each of the gate and the input cap are all zeros.

```
****************************************


  Startpoint: b (input port)
  Endpoint: out2 (output port)
  Path Group: (none)
  Path Type: max

  Point                        Fanout    Cap      Trans       Incr       Path
  ---------------------------------------------------------------------------
  input external delay                                      0.00000    0.00000 r
  b (in)                                          0.00000   0.00000    0.00000 r
  b (net)                         2  0.00000
  G6G5_A/A (INV_G6G5_A)                           0.00000   0.00000    0.00000 r
  G6G5_A/O (INV_G6G5_A)                           1.55900   4.19800    4.19800 f
  G6_G5_A (net)                   2  0.00000
  G5G1_B/A (NOR2_G5G1_B)                          1.55900   0.00000    4.19800 f
  G5G1_B/O (NOR2_G5G1_B)                          2.49700   2.95100    7.14900 r
  G5_G1_B (net)                   1  0.00000
  G1G4_A/B (AOI21_G1G4_A) <-                      2.49700   0.00000    7.14900 r
  G1G4_A/O (AOI21_G1G4_A)                         0.50560   0.61500    7.76400 f
  G1_G4_A (net)                   1  0.00000
  G4out2/A (NOR2_G4out2)                          0.50560   0.00000    7.76400 f
  G4out2/O (NOR2_G4out2)                          0.39570   0.52060    8.28460 r
  out2 (net)                      1  0.00000
  out2 (out)                                      0.39570   0.00000    8.28460 r
  data arrival time                                                    8.28460
  ---------------------------------------------------------------------------
  (Path is unconstrained)


1
#source dump_timing.tcl
#source dump_slack.tcl
exit
```

Figure 7.2 PrimeTime Report of the Test Design

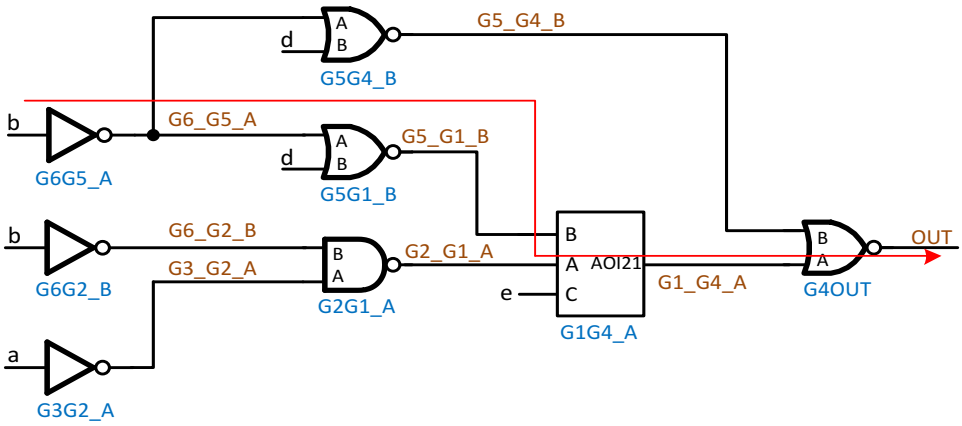

Figure 7.3 Test Design Imported into the PrimeTime

7.3 Result Accuracy and Error Percentage

To verify the result of the approach that reported from PrimeTime, a golden delay is needed for the new STA to compare with. HSpice is a trusted simulator that widely used in research and industry nowadays. It gives accuracy delay simulation before tape out. Thus we used HSpice to measure the delay on the critical path and compare it with the one from our approach. The expectation of the proposed approach is to make the difference as small as possible and limit it below 1%. In Table 1, it shows the gate names on critical path, along with their delay grasped from HSpice and the new approach, as well as the difference percentage of the delay between the two. The error percentage for each gate calculated by using the absolute value of difference between HSpice golden delay and the new STA result divided by the HSpice golden delay, the formula is given as follows:

$$\text{Error Percentage of a gate}(\%) = \frac{|\text{HSpice golden delay} - \text{New STA result}|}{\text{HSpice golden delay}} * 100\%$$

We can see from Table 7.1 that the gate error percentage varies from 0.19% to 2.33%. The overall delay error percentage on the entire critical path is calculated by:

$$Error_{\text{overall}}\ \% = \frac{|Total\ HSpice\ golden\ delay - Total\ New\ STA\ result|}{Total\ HSpice\ golden\ delay} * 100\%$$

$$Error_{\text{overall}}\% = \frac{|(4.19n + 2.95n + 0.615n + 0.512n) - (4.16n + 2.96n + 0.601n + 0.52n)|}{4.16n + 2.96n + 0.601n + 0.52n} * 100\%$$

$$= 0.42\ \%$$

On the critical path in this test design, our approach has only 0.42% delay difference with the HSpice simulation. This meets our expectation that the overall error percentage is below 1%.

Table 7.1 Gate Delay using HSpice and PrimeTime and the Delay Difference Percentage

Gate Name	HSpice Golden Delay	New STA Result	Error Percentage
G6G5_A	**4.16n**	**4.19n**	**0.72%**
G5G1_B	**2.96n**	**2.95n**	**0.34%**
G1G4_A	**0.601n**	**0.615n**	**2.33%**
G4out2	**0.52n**	**0.521n**	**0.19%**

CHAPTER 8

CONCLUSION

This thesis proposed a novel approach for static timing analysis on both ASIC design and custom circuits. The approach is split into five steps and is introduced step by step based on a specific customized design. The article describes four characters of the new implemented STA approach. Firstly, it can be widely used in not only ASIC design, but also some non-standard customized circuitry. Secondly, it is high efficient since it supports multicore HSpice running at the same time. Moreover, it is a memory efficient design because of the levelization, only one level of data need to be stored at a time. Most importantly, the approach has high accuracy since it directly maps the HSpice simulation result to PrimeTime. To practically verify the approach, a custom FPTA design in 130nm was used as a test case. The timing analysis of a custom FPTA using our proposed STA approach was shown step by step in this thesis. In chapter 7, the PrimeTime final report of the critical path using our proposed method was compared with HSpice golden delay. The difference between the two is as small as 0.42%. It statistically proved that the new STA method was implemented with enough accuracy and it is as trustable as HSpice simulation.

REFERENCES

[1] PrimeTime User Manual, Synopsys Inc, Mountain View, CA, 2017

[2] J. Tian, G. R. Reddy, J. Wang, W. Swartz, Y. Makris, C. Sechen, "A Field Programmable Transistor Array Featuring Single-Cycle Partial/Full Dynamic Reconfiguration," in Automation and Test in Europe (DATE), 2017, pp. 1336-1341

[3] N. Alaraje, Programmable Logic Device, (2018) [Online]. Available: http://slideplayer.com/slide/3943612/13

BIOGRAPHICAL SKETCH

Jiajia Wang was a research assistant worked on several projects such as Field Programmable Transistor Array and novel Timing Analysis Method in Nano Design Lab during three years of master program study at The University of Texas at Dallas. While at UT Dallas, she acted as a join author and published her first paper on DATE Conference. She also joined MediaTek as a design researcher internship for six months working on low power SRAM design and on chip FPGA integration. Before she joined UT Dallas, Jiajia Wang studied in Singapore for 3 years and transferred to Saint Louis University for further study, and received her bachelor's degree in Saint Louis University.

CURRICULUM VITAE

JIAJIA WANG

EDUCATION

M.S.E.E Circuit and System **Sep. 2014 – Feb. 2018**
The University of Texas at Dallas (UT Dallas)
B.S. Electrical Engineering **Feb. 2012 – Dec. 2013**
Saint Louis University (SLU)
Diploma. Electronic, Computer and Communication **Mar. 2008 – Mar. 2011**
Nanyang Polytechnic Singapore (NYP)

WORK EXPERIENCE

Digital IC Design Researcher Intern **May. 2017 – Dec. 2017**
MediaTek Austin
Research Assistant **Sep. 2014 – May. 2017**
Nanometer Design Laboratory of UT Dallas
VLSI Teaching Assistant **Sep. 2015 – May. 2017**
UT Dallas

AWARDS AND PROFESSIONAL AFFILIATIONS

Research Assistantship, UT Dallas 2015-2016
Dean's Honors list, **Saint Louis University** 2012-2013
Tau Beta Pi - National Engineering Honor Society 2013
Jesuit transfer scholarship, **Saint Louis University** 2011-2013
Admitted to Director list for outstanding Academic Performance 2008-2011

PUBLICATION

1. A Field Programmable Transistor Array Featuring Single-Cycle Partial/Full Dynamic Reconfiguration, Jingxiang Tian, Gaurav Rajavendra Reddy, **Jiajia Wang**, William Swartz, Yiorgos Makris, Carl Sechen, *Automation and Test in Europe (DATE), 2017*