Enablement of Optimal and Fast Timing Convergence for High speed I/O designs

Major Project

Submitted in Partial Fulfilment of the Requirements
for the Degree of
Master of Technology(M.Tech.)
in
VLSI Design
by

Saifee Shabbir S.
12MECV32

Department of Electronics & Communication Engineering
Institute of Technology
Nirma University
Ahmedabad
May-2014

Enablement of Optimal and Fast Timing Convergence for High speed I/O designs

Major Project
Submitted in Partial Fulfilment of the Requirements
for the Degree of
Master of Technology(M.Tech.)
in
VLSI Design
by

Saifee Shabbir S.
12MECV32

Dr. Amisha Naik
Internal Guide

Mr. Kaushik Saiprasad
External Guide

Department of Electronics & Communication Engineering
Institute Of Technology
Nirma University
Ahmedabad
December-2013

Declaration

This is to certify that

1. I, Shabbir Saifee, a student of semester IV Master of Technology in VLSI Design, Nirma University, Ahmedabad hereby declare that the project work "Enablement of Optimal and Fast Timing Convergence for High speed I/O designs "has been independently carried out by me under the guidance of Mr. Kaushik Saiprasad, Tech-Lead, Intel Technology India Private Limited, Bangalore and Dr. Amisha Naik, Professor, Department of VLSI Design, Nirma University, Ahmedabad. This Project has been submitted in the partial fulfillment of the requirements for the award of degree Master of Technology(M.Tech.) in VLSI Design, Nirma University, Ahmedabad during the year 2013 - 2014.

2. I have not submitted this work in full or part to any other University or Institution for the award of any other degree.

Shabbir Saifee
12MECV32

Certificate

This is to certify that the Major Project Part-I entitled "Enablement of Optimal and Fast Timing Convergence for High speed I/O designs" submitted by Shabbir Saifee (12MECV32), towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. Amisha Naik
Internal Project Guide,
Institute of Technology,
Nirma University, Ahmedabad

Mr. Kaushik Saiprasad
External Project Guide, Tech-Lead,
Intel Technology India Pvt. Ltd.,
Bangalore

Mr. Dilip Kothari
Section Head, M.tech. EC
Institute of Technology,
Nirma University,Ahmedabad

Prof. P N Tekwani
Head of EE Dept
Institute of Technology,
Nirma University,Ahmedabad

Date: Place:Ahmedabad

Certificate

This is to certify that Mr. Shabbir Saifee, 12mecv32, a student of M.Tech. (VLSI Design), Institute of Technology, Nirma University was working with this organization since 26/6/2014 and carried out his thesis work titled "Enablement of Optimal and Fast Timing Convergence for High speed I/O designs". He was working in the IPG, HIP DA-L division under the supervision of Mr. Kaushik Saiprasad, Tech Lead, Intel Technology. He has successfully completed his assigned work and is allowed to submit his dissertation. We wish him all the success in the future.

Mr. Kaushik Saiprasad
External Project Guide, Tech-Lead,
Intel Technology India Pvt. Ltd.,
Bangalore

Date: Place:Bangalore

Acknowledgement

With immense pleasure, I would like to present this report on the dissertation work related to "Enabling Optimal and Fast Timing Convergence for High speed I/O designs". I am very thankful to all those who helped me for the successful completion of the dissertation and for providing valuable guidance throughout the project work.

First of all, I would like to thank Mr. Thameem Syed for providing me with an opportunity to work at this esteemed organization and for continuous encouragement and motivation. I would like to express my deepest thanks and gratitude to Mr. Kaushik Saiprasad for providing constant support and guidance required to understand the project work and the platform required to carry out the project work.

I would also like to thank my internal guide , Dr. Amisha Naik, Professor, VLSI Design, Institute of Technology, Nirma University, Ahmedabad for giving valuable support for project work.

I also owe my colleagues at Intel, special thanks for helping me on this path and for making project at Intel more enjoyable.

Shabbir Saifee
12MECV32

Abstract

The ASIC devices are often composed of third-party IP (Intellectual Property), custom or semi-custom functional blocks, fab-vendor memory macros, standard cell logic, etc. As design sizes increase and customers migrate to static timing analysis solutions that incorporate delay calculation using parasitic information and signal integrity analysis, capacity and runtime issues for full-chip analysis becomes increasingly important.

For Static Timing Analysis, timing abstractions of designs for complex blocks or IP blocks can improve capacity and runtime while preserving reasonable accuracy. Usually, the digital blocks of an IP are modeled into a library and analog parts are used as it is for timing analysis. This report addresses the issue and implementation of bounding the analog modules of an IP into a timing model or a Liberty Syntax file. For the design convergence, making the flows optimal is required. The effect of variation is observed on the design parameters.

Proprietary Note: The names of the tools PrimeTime and IC Compiler provided by Synopsys that are mentioned in my thesis are the proprietary names and copyrights of Synopsys and I have used these tools as part of my internship work at Intel.

Contents

List of Figures

Chapter 1

Introduction

The STA (Static Timing Analysis) refers to the method of validating the timing performance of a design by checking all possible paths for timing violations. It is a complete and exhaustive verification of all timing checks of a design. Other timing analysis methods such as simulation can only verify the portions of the design that get exercised by stimulus. Verification through timing simulation is only as exhaustive as the test vectors used. To simulate and verify all timing conditions of a design with 10-100 million gates is very slow and the timing cannot be verified completely. Thus, it is very difficult to do exhaustive verification through simulation. Static timing analysis on the other hand provides a faster and simpler way of checking and analyzing all the timing paths in a design for any timing violations. Given the complexity of present day ASICs, which may contain 10 to 100 million gates, the static timing analysis has become a necessity to exhaustively verify the timing of a design.

1.1 Background and problem statement

An Intellectual Property (IP) is made up of many blocks connected together. Some of these blocks are analog in nature and some are digital. The timing analysis of the digital blocks is easier than analog blocks. Also the analog blocks are more complex in design and analysis than digital. There are smaller libs that are present inside the analog blocks. Along with the other digital/analog logic, these libs form an analog block. There are multiple analog blocks in an IP interfaced with other digital blocks. These analog blocks are also called Custom Macros (CM's).

In the present process of sign-off, at the design level, the digital and analog blocks are put together and the STA is performed. The parameters that are needed to be provided to the tool are Verilog netlists, the constraints for each and every block, a flat RC extraction i.e. spef for the whole design, the smaller libs in the analog blocks and other parameters required. Hence in order to do the analysis of the analog blocks, we have to open them up and provide the Verilogs for the digital parts along with smaller analog libs. Because of this , the runtime is huge. Also in case of violations the resolution time and turnaround time is high. As a result the number of iterations for sign-off increases, hence the time to market. Now as the no of transistors increases, CM's become more complex and timing analysis becomes more and more difficult.

The solution to all these problems is to box up the CM's. PrimeTime provides a facility to generate the timing models for a particular block which can be used at the top hierarchy of the design instead of the netlist and all the smaller intcin libs. Hence at the top level, we can replace the analog blocks with their corresponding timing models or libs and perform the timing analysis. This way the number of iterations that are required can significantly be reduced. This report deals with the extraction, modeling and use of these timing models.

The sign-off process is very important in ASIC design. The complete design is performed in an implementation tool and then it is taken to a timing analysis tool to perform sign-off. Presently, there is a big difference in timing numbers between these tools. The idea is to introduce the Advanced On Chip Variation derates in both tools and try to bring the timing closer.

1.2 Introduction to STA

In a CMOS digital design flow, the static timing analysis can be performed at many different stages of the implementation. Figure 1 shows a typical flow.

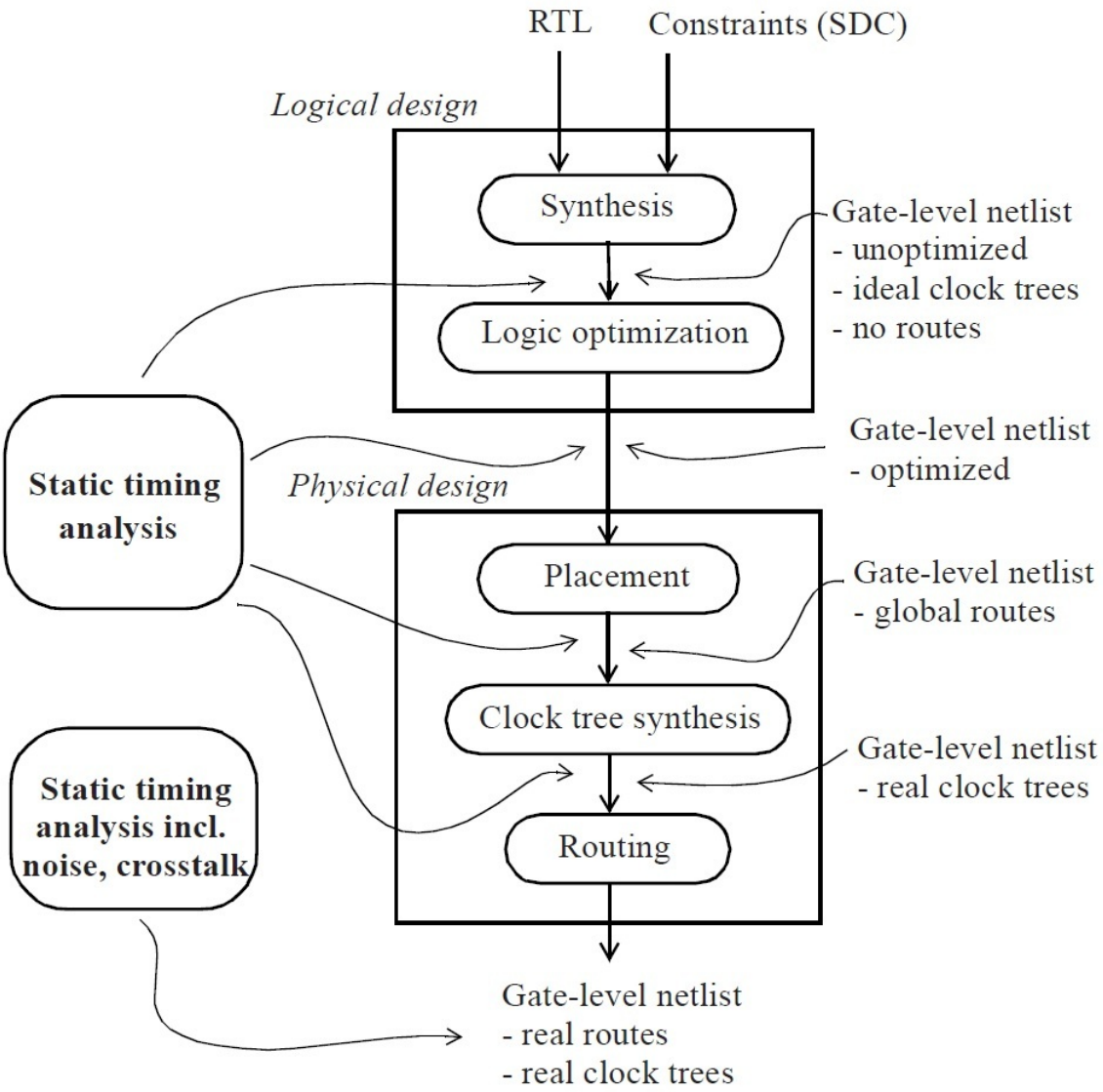

Figure 1.1: STA in ASIC design

STA is rarely done at the RTL level as, at this point, it is more important to verify the functionality of the design as opposed to timing. Also not all timing information is available since the descriptions of the blocks are at the gate level; the STA is used to verify the timing of the design. STA can also be run prior to performing logic optimization - the goal is to identify the worst or critical timing paths. STA can be rerun after logic optimization to see whether there are failing paths still remaining that need to be optimized, or to identify the critical paths.

At the start of the physical design, clock trees are considered as ideal, that is, they have zero delay. Once the physical design starts and clock trees are built, STA can be performed to check the timing again. In physical implementation, the logic cells are connected by interconnects metal traces. The parasitic RC (Resistance and Capacitance) of the metal traces impact the signal path delay through these traces. In a typical nanometer design, the parasitics of the interconnect can account for the majority of the delay and power dissipation in the design.

At the logical design phase, ideal interconnect may be assumed since there is no physical information related to the placement; there may be more interest in viewing the logic that contributes to the worst paths. Another technique used at this stage is to estimate the length of the interconnect using a wireload model. The wireload model provides estimated RC based on the fanouts of a cell. Before the routing of traces are finalized, the implementation tools use an estimate of the routing distance to obtain RC parasitics for the route. Since the routing is not finalized, this phase is called the global route phase to distinguish it from the final route phase. In the global route phase of the physical design, simplified routes are used to estimate routing lengths, and the routing estimates are used to determine resistance and capacitance that are needed to compute wire delays. During this phase, one cannot include the effect of coupling.

After the detailed routing is complete, actual RC values obtained from extraction are used and the effect of coupling can be analyzed. However, a physical design tool may still use approximations to help improve run times in computing RC values. An extraction tool is used to extract the detailed parasitics (RC values) from a routed design. Such an extractor may have an option to obtain parasitics with small runtime and less accurate RC values during iterative optimization and another option for final verification during which very accurate RC values are extracted with a larger runtime.

The STA is static since the analysis of the design is carried out statically and does not depend upon the data values being applied at the input pins. This is in contrast to simulation based timing analysis where a stimulus is applied on input signals, resulting behavior is observed and verified, then time is advanced with new input stimulus applied, and the new behavior is observed and verified and so on.

Given a design along with a set of input clock definitions and the definition of the external environment of the design, the purpose of static timing analysis is to validate if the design can operate at the rated speed. That is, the design can operate safely at the specified frequency of the clocks without any timing violations. Figure 2 shows the basic functionality of static timing analysis. The DUA is the design under analysis. Some examples of timing checks are setup and hold checks. A setup check ensures that the data can arrive at a flip-flop within the given clock period. A hold check ensures that the data is held for at least a minimum time so that there is no unexpected pass-through of data through a flip-flop: that is, it ensures that a flip-flop captures the intended data correctly. These checks ensure that the proper data is ready and available for capture and latched in for the new state.

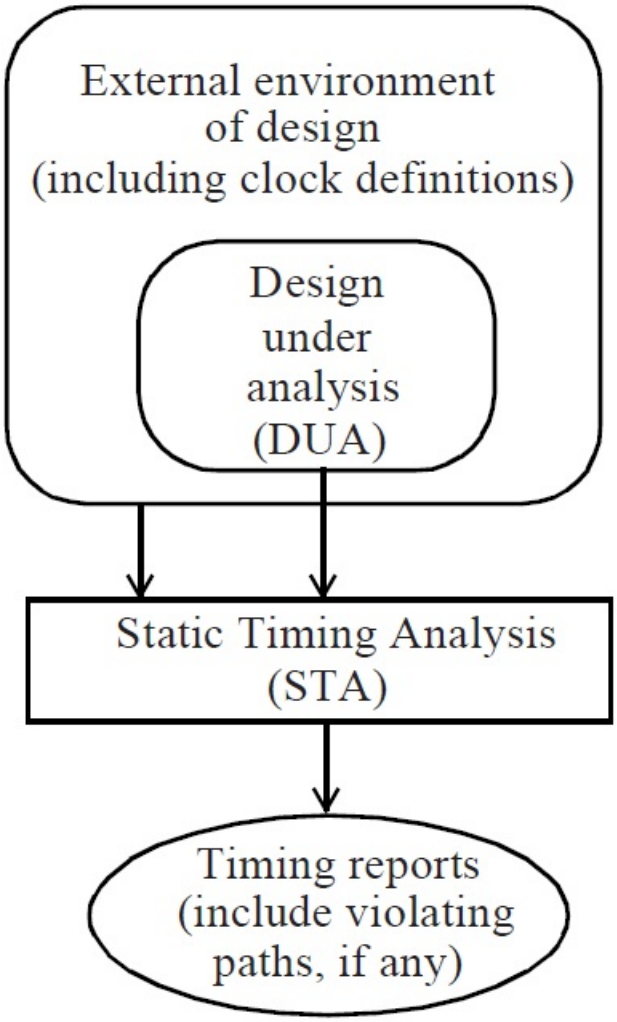

Figure 1.2: STA basic flow

1.3 Noise and Crosstalk

In semiconductor devices, metal interconnect traces are typically used to make the connections between various portions of the circuitry to realize the design. As the process technology shrinks, these interconnect traces have been known to affect the performance of a design. For deep submicron or nanometer process technologies, the coupling in the interconnect induces noise and crosstalk - either of which can limit the operating speed of a design. While the noise and coupling effects are negligible at older generation technologies, these play an important role in nanometer technologies. Thus, the physical design should consider the effect of crosstalk and noise and the design verification should then include the effects of crosstalk and noise.

The crosstalk noise refers to unintentional coupling of activity between two or more signals. The crosstalk noise is caused by the capacitive coupling between neighboring signals on the die. This results in switching activity on a net to cause unintentional effects on the coupled signals. The affected signal is called the victim, and the affecting signals are termed as aggressors. Note that two coupled nets can affect each other, and

often a net can be a victim as well as an aggressor. Figure 3 shows an example of a few signal traces coupled together. The distributed RC extraction of the coupled interconnect is depicted along with several drivers and fanout cells. In this example, nets N1 and N2 have Cc1 + Cc4 as coupling capacitance between them, whereas Cc2 + Cc5 is the coupling capacitance between nets N2 and N3.

There are two types of noise effects caused by crosstalk glitch, which refers to noise caused on a steady victim signal due to coupling of switching activity of neighboring aggressors, and change in timing (crosstalk delta delay), caused by coupling of switching activity of the victim with the switching activity of the aggressors.

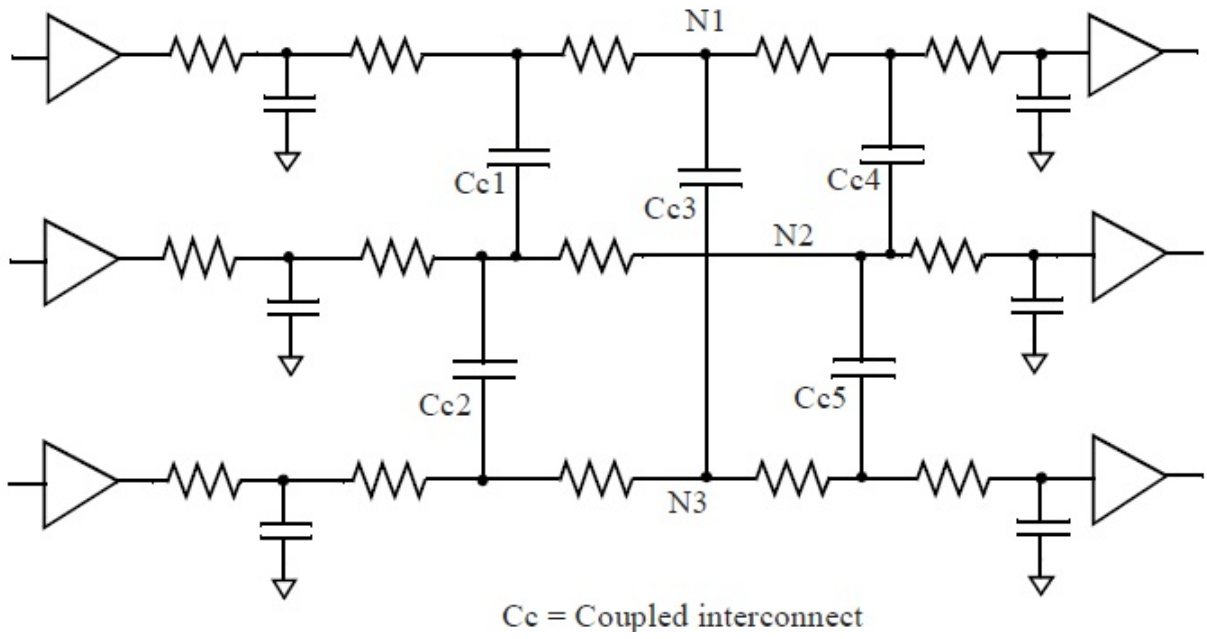

Figure 1.3: Coupled interconnects

There are several reasons why the noise plays an important role in the deep submicron technologies:

- Increasing number of metal layers: For example, a 0.25mm or 0.3mm process has four or five metal layers and it increases to ten or higher metal layers in the 65nm and 45nm process geometries.
- Vertically dominant metal aspect ratio: This means that the wires are thin and tall unlike the wide and thin in the earlier process geometries. Thus, a greater proportion of the capacitance is comprised of sidewall coupling capacitance which maps into wire to wire capacitance between neighboring wires.
- Higher routing density due to finer geometry: Thus, more metal wires are packed in close physical proximity.
- Larger number of interacting devices and interconnects: Thus, greater number of active standard cells and signal traces are packed in the same silicon area causing a lot more interactions.

- Faster waveforms due to higher frequencies: Fast edge rates cause more current spikes as well as greater coupling impact on the neighboring traces and cells.
- Lower supply voltage: The supply voltage reduction leaves little margin for noise.

1.4 Setting up the STA environment

Specification of correct constraints is important in analyzing STA results. The design environment should be specified accurately so that STA analysis can identify all the timing issues in the design. Preparing for STA involves amongst others, setting up clocks, specifying IO timing characteristics, and specifying false paths and multicycle paths. Most digital designs are synchronous where the data computed from the previous clock cycle is latched in the flip-flops at the active clock edge. Consider a typical synchronous design shown in Figure 4. It is assumed that the Design Under Analysis (DUA) interacts with other synchronous designs. This means that the DUA receives the data from a clocked flipflop and outputs data to another clocked flip-flop external to the DUA.

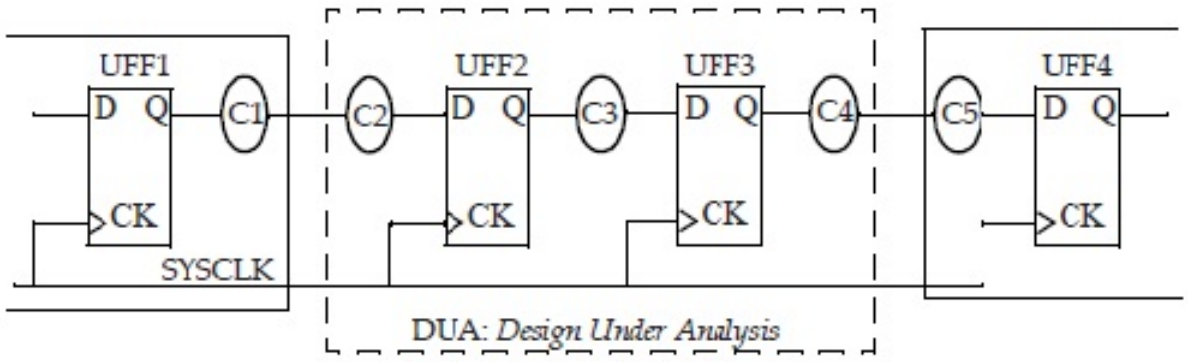

Figure 1.4: STA Environment

To perform STA on this design, one needs to specify the clocks to the flip-flops, and timing constraints for all paths leading into the design and for all paths exiting the design. The example in Figure 4 assumes that there is only one clock and C1, C2, C3, C4, and C5 represent combinational blocks. The combinational blocks C1 and C5 are outside of the design being analyzed. In a typical design, there can be multiple clocks with many paths from one clock domain to another.

The tool used for the purpose of timing analysis is Synopsys PrimeTime. PrimeTime is a full-chip, gate-level static timing analysis tool targeted for complex, multimillion-gate designs. It accepts design information in a wide range of industry-standard formats, including gate-level netlists in .db, Verilog, and VHDL formats; delay information in Standard Delay Format (SDF); parasitic data in Standard Parasitic Exchange Format (SPEF), Synopsys Binary Parasitic Format (SBPF) and Reduced Standard Parasitic Format (RSPF) formats; and timing constraints in Synopsys Design Constraints (SDC) format.

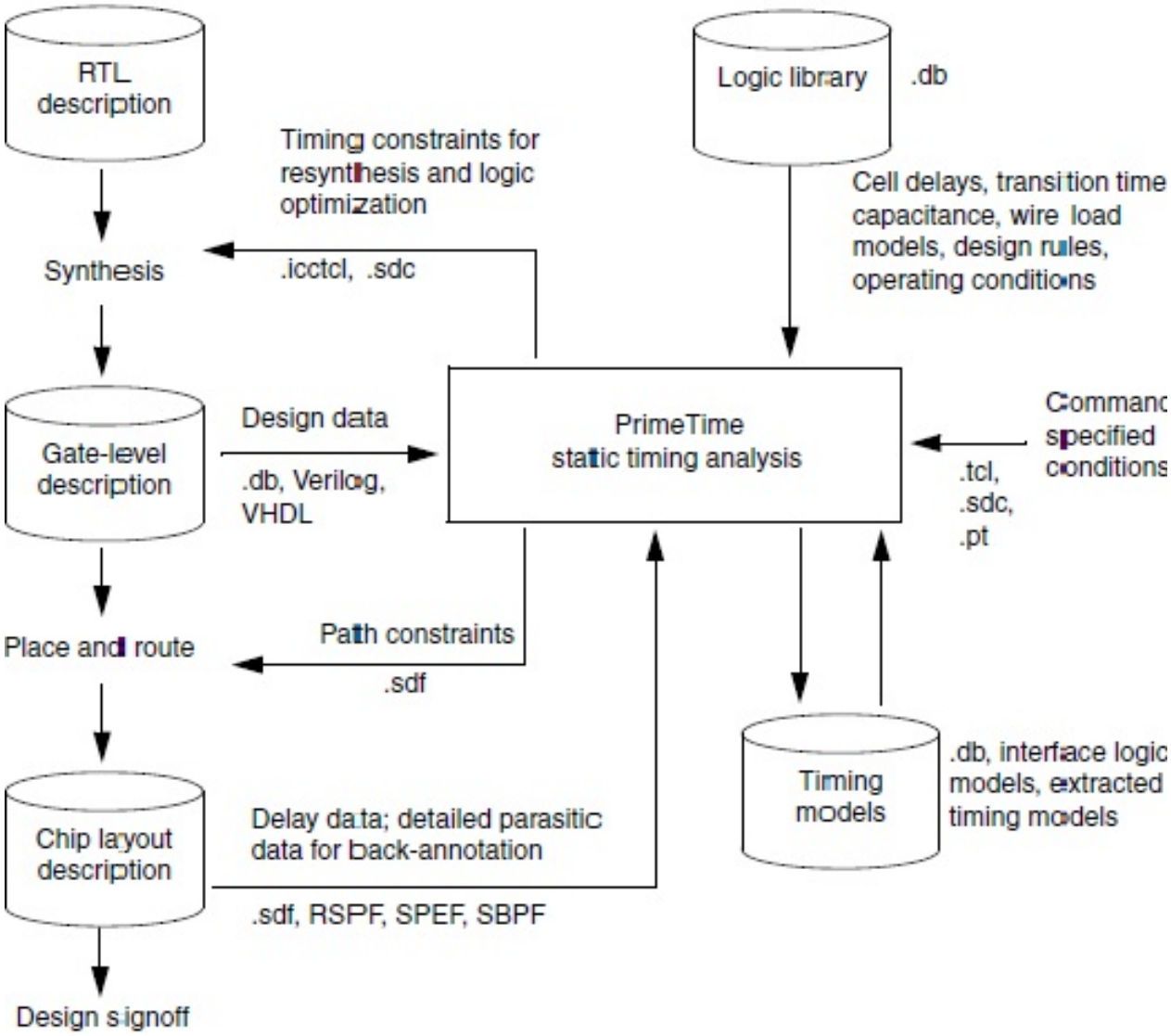

Figure 1.5: Timing Flow

Figure 5 shows how PrimeTime is used in a typical synthesis flow. Starting from an RTL design description, a synthesis tool generates a gate-level design description. PrimeTime reads this description and verifies the design timing using information provided in the logic library. If PrimeTime finds any timing violations, the design needs to be resynthesized using new timing constraints (generated by PrimeTime) to fix the conditions that are causing the timing violations. When the gate-level design is free of timing violations, you can proceed to placement and routing. This produces a chip layout database from which accurate delay information or detailed parasitic information can be extracted. This data, when back-annotated on the design in PrimeTime, results in a physically-accurate timing analysis. A successful validation of the circuit timing at this point leads to signoff of the completed design.

From the figure 5 we can gather that the RTL generated is synthesized into a gate level netlist using a synthesis tool. The design is then translated in terms of pre-defined standard cells and libraries. This serves as a design input to the tool in the form of a Verilog or VHDL format files. Also the standard cells and libraries have to be provided in the form of .db and .lib files. These files contain the cell delays, transition times,

capacitances, operating conditions (PVT), design rules and other information in them in the form of NLDM tables.

The Synopsys Design Constraint (.sdc) file is also an important input to the tool. Basically, the sdc files contain the setup information wrt design. They are:

- Clock definitions at various pins/ports and the period and waveforms of the clock.
- Setting the input transition values at various pins/ports
- Setting up false paths between various parts of design.
- Setting derates to various clock paths and data paths.
- Various other types of constraints wrt design and technology.

The timing models that are extracted for the bigger modules are also instantiated along with the standard cell libs and Verilog netlists. They can be in the form of Interface Logic Models (ILMs) or Extracted Timing Models (ETMs).

The parasitic information of the circuit is vital in order to calculate the detailed delays of the nets and determine accurate timing information. The tool basically accepts the three types of parasitic file formats:

- Reduced Standard Parasitic Format (RSPF)
- Detailed Standard Parasitic Format (DSPF)
- Standard Parasitic Exchange Format (SPEF)

Usually the SPEF is used in order to provide the RC information of the nets. It is very complete and compact way of representing the parasitics. The RC information is back annotated in the design and real time delay values can be used to calculate the delay arcs.

This chapter so far we have seen the basic introduction of timing, setting up the STA environment and timing flow used for it. In the next chapter we will look over the standard cell libraries and how timing and noise are characterized in a .lib.

Chapter 2

Standard cell libraries, timing and noise characterization

The ASIC design involves the use of standard cell libraries and timing models for the purpose of design and timing analysis. The timing information of any cell or block is represented in the .lib files in the Non-Linear Delay Model (NLDM) format. The noise representation is done in the form of Composite Current Source (CCS) models. This chapter will cover the basics of the NLDM and CCS models.

2.1 Non-Linear Delay Models

The cell timing models are intended to provide accurate timing for various instances of the cell in the design environment. The timing models are normally obtained from detailed circuit simulations of the cell to model the actual scenario of the cell operation. The timing models are specified for each timing arc of the cell. The delay for the timing arc of any cell is dependent on two parameters; 1) the input transition time of the signal and 2) The capacitance load at the output pin of the cell. The delay values have a direct correlation with the load capacitance the larger the load capacitance, the larger the delay.

Most of the cell libraries include table models to specify the delays and timing checks for various timing arcs of the cell. The table models are referred to as NLDM and are used for delay, output slew, or other timing checks. The table models capture the delay through the cell for various combinations of input transition time at the cell input pin and total output capacitance at the cell output. An NLDM model for delay is presented in a two-dimensional form, with the two independent variables being the input transition time and the output load capacitance, and the entries in the table denoting the delay. An example of such a table for a typical inverter cell is:

```
pin (OUT) {
        max_transition : 1.0;
        timing() {
                related_pin : "INP1";
                timing_sense : negative_unate;
                cell_rise(delay_template_3x3) {
                        index_1 ("0.1, 0.3, 0.7"); /* Input transition */
                        index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
                        values ( /* 0.16 0.35 1.43 */ \
                        /* 0.1 */ "0.0513, 0.1537, 0.5280", \
                        /* 0.3 */ "0.1018, 0.2327, 0.6476", \
                        /* 0.7 */ "0.1334, 0.2973, 0.7252");
                        }
                cell_fall(delay_template_3x3) {
                        index_1 ("0.1, 0.3, 0.7"); /* Input transition */
                        index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
                        values ( /* 0.16 0.35 1.43 */ \
                        /* 0.1 */ "0.0617, 0.1537, 0.5280", \
                        /* 0.3 */ "0.0918, 0.2027, 0.5676", \
                        /* 0.7 */ "0.1034, 0.2273, 0.6452");
```

Figure 2.1: Inverter NLDM table

In the above example, the delays of the output pin OUT are described. This portion of the cell description contains the rising and falling delay models for the timing arc from pin INP1 to pin OUT, as well as the max_transition allowed time at pin OUT. There are separate models for the rise and fall delays (for the output pin) and these are labeled as cell_rise and cell_fall respectively. The type of indices and the order of table lookup indices are described in the lookup table template delay_template_3x3.

```
lu_table_template(delay_template_3x3) {
        variable_1 : input_net_transition;
        variable_2 : total_output_net_capacitance;
        index_1 ("1000, 1001, 1002");
        index_2 ("1000, 1001, 1002");
        }
```

Figure 2.2: Lookup table template

The input transition and the output capacitance can be in either order, that is, variable_1 can be the output capacitance. However, these designations are usually consistent across all templates in a library. This lookup table template specifies that the first variable in the table is the input transition time and the second variable is the output capacitance. The table values are specified like a nested loop with the first index (index_1) being the outer (or least varying) variable and the second index (index_2) being the inner (or most varying) variable and so on. There are three entries for each variable and thus it corresponds to a 3-by-3 table. In most cases, the entries for the table are also formatted like a table and the first index (index_1) can then be treated as a row index and the second index (index_2) becomes equivalent to the column index. The index values (for example 1000) are dummy placeholders which are overridden by the actual index values in the cell_fall and cell_rise delay tables.

The setup and hold constraints for a synchronous pin of a sequential cell are normally described in terms of two-dimensional tables as illustrated below. The example below shows the setup and hold timing information for the data pin of a flip-flop.

```
pin (D) {
direction : input;
. . .
timing () {
        related_pin : "CK";
        timing_type : "setup_rising";

        rise_constraint ("setuphold_template_3x3") {
                index_1("0.4, 0.57, 0.84"); /* Data transition */
                index_2("0.4, 0.57, 0.84"); /* Clock transition */
                values( /* 0.4 0.57 0.84 */ \
                /* 0.4 */ "0.063, 0.093, 0.112", \
                /* 0.57 */ "0.526, 0.644, 0.824", \
                /* 0.84 */ "0.720, 0.839, 0.930");

        fall_constraint ("setuphold_template_3x3") {
                index_1("0.4, 0.57, 0.84"); /* Data transition */
                index_2("0.4, 0.57, 0.84"); /* Clock transition */
                values( /* 0.4 0.57 0.84 */ \
                /* 0.4 */ "0.762, 0.895, 0.969", \
                /* 0.57 */ "0.804, 0.952, 0.166", \
                /* 0.84 */ "0.159, 0.170, 0.245");
                }
        }
}


timing () {
        related_pin : "CK";
        timing_type : "hold_rising";

        rise_constraint ("setuphold_template_3x3") {
                index_1("0.4, 0.57, 0.84"); /* Data transition */
                index_2("0.4, 0.57, 0.84"); /* Clock transition */
                values( /* 0.4 0.57 0.84 */ \
                /* 0.4 */ "-0.220, -0.339, -0.584", \
                /* 0.57 */ "-0.247, -0.381, -0.729", \
                /* 0.84 */ "-0.398, -0.516, -0.864");
                }

         fall_constraint ("setuphold_template_3x3") {
                index_1("0.4, 0.57, 0.84"); /* Data transition */
                index_2("0.4, 0.57, 0.84");/* Clock transition */
                values( /* 0.4 0.57 0.84 */ \
                /* 0.4 */ "-0.028, -0.397, -0.489", \
                /* 0.57 */ "-0.408, -0.527, -0.649", \
                /* 0.84 */ "-0.705, -0.839, -0.580");
                }
}
```

Figure 2.3: Setup and Hold tables

2.2 Composite Current Source Models (CCS)

The timing models, such as NLDM, represent the delay through the timing arcs based upon output load capacitance and input transition time. In reality, the load seen by the cell output is comprised of capacitance and interconnect resistance. The interconnect resistance becomes an issue since the NLDM approach assumes that the output loading is purely capacitive. Even with non-zero interconnect resistance, these NLDM models have been utilized when the effect of interconnect resistance is small. As the feature size shrinks, the effect of interconnect resistance can result in large inaccuracy as the waveforms become highly non-linear. Various modeling approaches provide additional accuracy for the cell output drivers. Broadly, these approaches obtain higher accuracy by modeling the output stage of the driver by an equivalent current source. The CCS timing models provide the additional accuracy for modeling cell output drivers by using a time-varying and voltage-dependent current source. The timing information is provided by specifying detailed models for the receiver pin capacitance and output charging currents under different scenarios.

2.2.1 Receiver Pin capacitance

The receiver pin capacitance can be specified with the help of a one dimensional table as follows:

```
pin (IN) {
        . . .
        receiver_capacitance1_rise ("Lookup_table_4") {
        index_1: ("0.1, 0.2, 0.3, 0.4"); /* Input transition */
        values("0.001040, 0.001072, 0.001074, 0.001085");
}
```

Figure 2.4: Receiver Pin capacitance

The index_1 specifies the indices for input transition time for this pin. The one-dimensional table in values specifies the receiver capacitance for rising waveform at an input pin for the leading portion of the waveform. Similar to the receiver_capacitance1_rise shown above, the receiver_capacitance2_rise specifies the rise capacitance for the trailing portion of the input rising waveform. The fall capacitances (pin capacitance for falling input waveform) are specified by the attributes receiver_capacitance1_fall and receiver_capacitance2_fall respectively.

2.2.2 Output Current

The non-linear timing is represented in terms of output current. The output current information is specified as a lookup table that is dependent on input transition time

and output load. The output current is specified for different combinations of input transition time and output capacitance. For each of these combinations, the output current waveform is specified. Essentially, the waveform here refers to output current values specified as a function of time.

```
pin (OUT) {
    . . .
    timing () {
    related_pin : "IN" ;
    . . .
    output_current_fall () {
        vector ("LOOKUP_TABLE_1x1x5") {
        reference_time : 5.06; /* Time of input crossing threshold */
            index_1("0.040"); /* Input transition */
            index_2("0.900"); /* Output capacitance */
            index_3("5.079e+00, 5.093e+00, 5.152e+00,
            5.170e+00, 5.352e+00");/* Time values */
            /* Output charging current: */
            values("-5.784e-02, -5.980e-02, -5.417e-02,
            -4.257e-02, -2.184e-03");}
```

Figure 2.5: Output current specification

The index_1 and index_2 refer to the input transition time and the output load used and index_3 is the time. The index_1 and index_2 (the input transition time and output capacitance) can have only one value each. The index_3 refers to the time values and the table values refer to the corresponding output current. Thus, for the given input transition time and output load, the output current waveform as a function of time is available.

2.2.3 CCSN model and parameters

The CCB refers to the source-drain channel connected portion of a cell. For example, single stage cells such as an inverter, nand and nor cells contain only one CCB - the entire cell is connected through using one channel connection region. Multi-stage cells such as and cells, or or cells, contain multiple CCBs.

The CCSN models are usually specified for the first CCB driven by the cell input, and the last CCB driving the cell output. These are specified using steady state current, output voltage and propagated noise models. For single stage combinational cells such as nand and nor cells, the CCS noise models are specified for each timing arc. These cells have only one CCB and thus the models are from input pins to the output pin of the cell. An example model for nand cell is:

```
pin (OUT) {
. . .
timing () {
related_pin : "IN1";
. . .
ccsn_first_stage() { /* First stage CCB */
        is_needed : true;
        stage_type : both; /*CCB contains pull-up and pull-down*/
        is_inverting : true;
        miller_cap_rise : 0.8;
        miller_cap_fall : 0.5;
        dc_current (ccsn_dc) {
                index_1 ("-0.9, 0, 0.5, 1.35, 1.8"); /* Input voltage */
                index_2 ("-0.9, 0, 0.5, 1.35, 1.8"); /* Output voltage*/
                values ( \
                "1.56, 0.42, . . ."); /* Current at output pin */}
. .
        output_voltage_rise () {
        vector (ccsn_ovrf) {
                index_1 ("0.01"); /* Rail-to-rail input transition */
                index_2 ("0.001"); /* Output net capacitance */
                index_3 ("0.3, 0.5, 0.8"); /* Time */
                values ("0.27, 0.63, 0.81");}.
. .}
        output_voltage_fall () {
        vector (ccsn_ovrf) {
                index_1 ("0.01"); /* Rail-to-rail input transition */
                index_2 ("0.001"); /* Output net capacitance */
                index_3 ("0.2, 0.4, 0.6"); /* Time */
                values ("0.81, 0.63, 0.27");}.
. .}
        propagated_noise_low () {
        vector (ccsn_pnlh) {
                index_1 ("0.5"); /* Input glitch height */
                index_2 ("0.6"); /* Input glitch width */
                index_3 ("0.05"); /* Output net capacitance */
                index_4 ("0.3, 0.4, 0.5, 0.7"); /* Time */
                values ("0.19, 0.23, 0.19, 0.11");
}
        propagated_noise_high () {
. . .
}}
```

Figure 2.6: CCSN example

The attribute ccsn_first_stage indicates that the model is for the first stage CCB of the nand cell. As mentioned before, the nand cell has only one CCB. The attribute is_needed is almost always true with the exception being that for nonfunctional cells such as load cells and antenna cells. The stage_type with value both specifies that this stage has both pull-up and pull-down structures. The miller_cap_rise and miller_cap_fall represent the Miller capacitances1 for the rising and falling output transitions respectively.

The dc_current tables represent the DC current at the output pin for different combinations of input and output pin voltages. The index_1 specifies the input voltage

and index_2 specifies the output voltage. The values in the two-dimensional table specify the DC current at the CCB output. The input voltages and output currents are all specified in library units (normally Volt and mA).

The output_voltage_rise and output_voltage_fall constructs contain the timing information for the CCB output rising and falling respectively. These are specified as multi-dimensional tables for the CCB output node. The multidimensional tables are organized as multiple tables specifying the rising and falling output voltages for different input transition time and output net capacitances. Each table has index_1 specifying the rail-to-rail input transition time rate and index_2 specifying the output net capacitance. The index_3 specifies the times when the output voltage crosses specific threshold points. In each multi-dimensional table, the voltage crossing points are fixed and the time values when the CCB output node crosses the voltage are specified in index_3.

The propagated_noise_high and propagated_noise_low models specify multidimensional tables which provide noise propagation information through the CCB. These models characterize the crosstalk glitch (or noise) propagation from an input to the output of the CCB. The characterization uses symmetric triangular waveform at the input. The multi-dimensional tables for propagated_noise are organized as multiple tables specifying the glitch waveform at the output of the CCB. These multi-dimensional tables contain: 1. input glitch magnitude (in index_1), 2. Input glitch width (in index_2), 3. CCB output net capacitance (in index_3), and 4. time (in index_4).

2.3 NLDM table look up and delay calculations

As illustrated above, an inverter cell with an NLDM model has the following tables:

- Rise delay
- Fall delay
- Rise transition
- Fall transition

Given the input transition time and output capacitance of such a cell, as shown in Figure 12, the rise delay is obtained from the cell_rise table for 15ps input transition time (falling) and 10fF load, and the fall delay is obtained From the cell_fall table for 20ps input transition time (rising) and 10fF load.

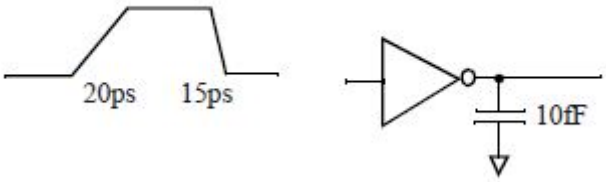

Figure 2.7: Inverter delay calculation example

The info about the inverting nature of the cell is present in the field called timing_sense which specifies whether the timing arc is positive_unate or negative_unate. For the example inverter cell, the timing arc is negative_unate which implies that the output pin transition direction is opposite (negative) of the input pin transition direction. Thus, the cell_rise table lookup corresponds to the falling transition time at the input pin.

If the input transition time and the output capacitance correspond to a table entry, the table lookup is trivial since the timing value corresponds directly to the value in the table. The example below corresponds to a general case where the lookup does not correspond to any of the entries available in the table. In such cases, two-dimensional interpolation is utilized to provide the resulting timing value. The two nearest table indices in each dimension are chosen for the table interpolation.

Consider the table lookup for fall transition (example table specified above) for the input transition time of 0.15ns and an output capacitance of 1.16pF. The corresponding section of the fall transition table relevant for two-dimensional interpolation is reproduced below.

```
fall_transition(delay_template_3x3) {
 index_1 ("0.1, 0.3 . . .");
 index_2 (". . . 0.35, 1.43");
 values ( \
  ". . . 0.1937, 0.7280", \
  ". . . 0.2327, 0.7676"
  . . .
```

Figure 2.8: Delay table for inverter

In the formulation below, the two index_1 values are denoted as x1 and x2; the two index_2 values are denoted as y1 and y2 and the corresponding table values are denoted as T11, T12, T21 and T22 respectively. If the table lookup is required for (x0, y0), the lookup value T00 is obtained by interpolation and is given by:

T00 = x20 * y20 * T11 + x20 * y01 * T12 + x01 * y20 * T21 + x01 * y01 * T22
where,

x01 = (x0 - x1) / (x2 - x1)
x20 = (x2 - x0) / (x2 - x1)
y01 = (y0 - y1) / (y2 - y1)
y20 = (y2 - y0) / (y2 - y1)

Substituting 0.15 for index_1 and 1.16 for index_2 results in the fall_transition value of:

T00 = 0.75 * 0.25 * 0.1937 + 0.75 * 0.75 * 0.7280 + 0.25 * 0.25 * 0.2327
+ 0.25 * 0.75 * 0.7676
= 0.6043

Note that the equations above are valid for interpolation as well as extrapolation - that is when the indices (x0, y0) lie outside the characterized range of indices. As an example, for the table lookup with 0.05 for index_1 and 1.7 for index_2, the fall transition value is obtained as:

T00 = 1.25 * (-0.25) * 0.1937 + 1.25 * 1.25 * 0.7280 + (-0.25) * (-0.25) * 0.2327
+ (-0.25) * 1.25 * 0.7676
= 0.8516

Hence, we can understand how the delay values are calculated by looking up trans and cap values from the tables.

In this chapter, we discussed the concept of NLDM and CCSN tables and also how the delay values are calculated. In the next chapter we'll discuss how the timing libs are generated and different concepts of it.

Chapter 3

Timing abstraction

In any ASIC Design, the design is made up of many analog as well as digital blocks. Timing models are generated for the complex and big logic blocks from PrimeTime in order to increase the accuracy and reduce the runtime of the whole design. These models are either generated by using Interface Logic Models (ILM), Extracted Timing Models (ETM) or Quick models. The ETMs are usually used because they are more accurate, compact and very advantageous from the security point of view of an IP vendor. This chapter covers the details of ETM models, their extraction, validation and corresponding methodology.

3.1 Timing Models

A typical chip can contain synthesized logic, netlist-based cores, and predesigned custom blocks, as shown in figure 14.

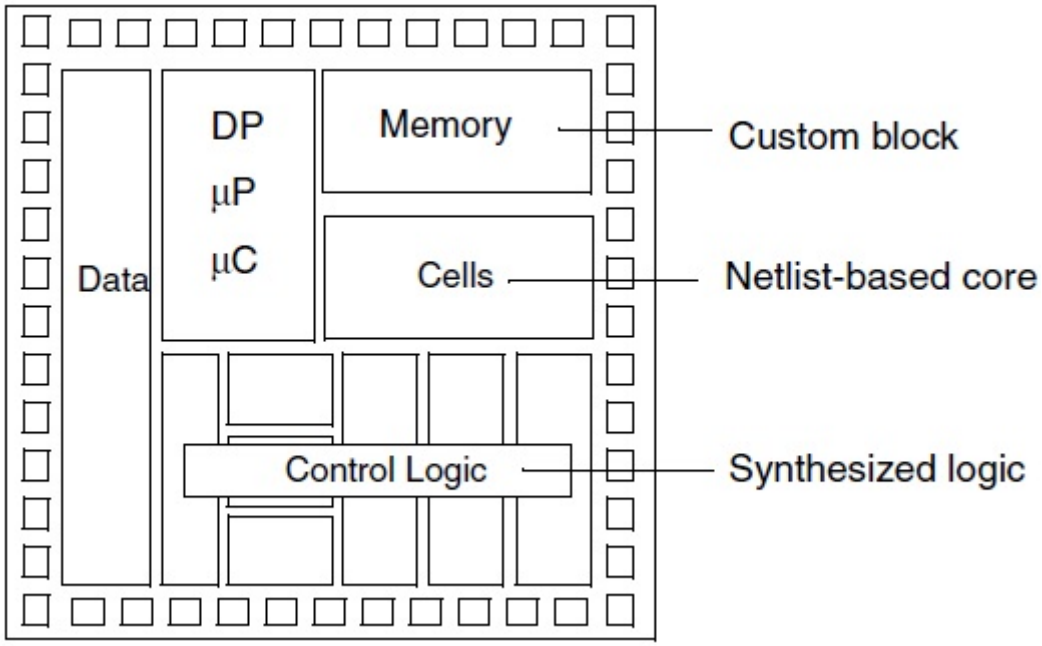

Figure 3.1: ASIC structure

Before you can perform static timing analysis on a chip using PrimeTime, every leaf cell must have a timing model. For static timing purposes, a leaf cell can be a simple macro cell (such as a NAND, NOR, or flip-flop) or a complex block (such as a RAM or microprocessor).

Synthesized logic is modeled as a netlist containing gates such as NANDs, NORs, and flip-flops. The gates are modeled using the standard Synopsys modeling language. They are then compiled into a logic library database (.db) file using Library Compiler. This logic library is the same library used by Design Compiler. A predesigned custom block is defined at the transistor level and imported into the chip as a fixed unit, such as a RAM or microprocessor block. Because a gate-level netlist does not exist for this type of block, the Liberty modeling language can be used to describe the timing behavior of the block.

PrimeTime provides three kinds of modeling techniques for creating timing models:

- Interface Logic Models (ILM's)
- Quick Timing Models (QTM's)
- Extracted Timing Models (ETM's)

The ILMs are the kinds of models which capture the cells that are at the boundary of the particular design. Hence the name Interface logic models. It captures the flops that are connected to the input and output pins and also the combinational logic. It generates a Verilog for such config and corresponding extraction in the form of parasitics file (.spef). As it exposes the boundary info and the elements present, it is not usually used because it does not provide required security wrt design that is required for an IP design particularly. Also the accuracy of the model is not up to the mark. Hence in order to avoid all of this the ETMs are used.

3.2 Extracted Timing Models

PrimeTime can generate a static timing model for the current design from its gate-level netlist. The generated model has the same timing behavior as the original netlist, and can be used in place of the original netlist in a hierarchical timing analysis. Using an extracted timing model has these advantages:

- The generated model is usually much smaller than the original netlist. When you use extracted models in place of netlists in PrimeTime, you can significantly reduce the time needed to analyze a large design.
- Using a model in place of a netlist prevents a user from seeing the contents of the block, allowing the block to be shared while protecting the intellectual property of the block creator.

The ETMs are an abstraction of the block using sequential and combinational timing arcs. NLDM tables are extracted for each of the timing arcs whose delay is a function of input transitions and output loads. The extraction process requires the block level netlist, logic libraries and clocks and constraints. PrimeTime takes all of these as inputs and generates a timing model for that block.

3.2.1 Model Extraction

Timing model extraction creates a timing arc for each path in the design from an input port to a register, an input port to an output port, and from a register to an output port. Figure 7 and 8 show an example of a gate-level design named simple and the model extracted from the netlist.

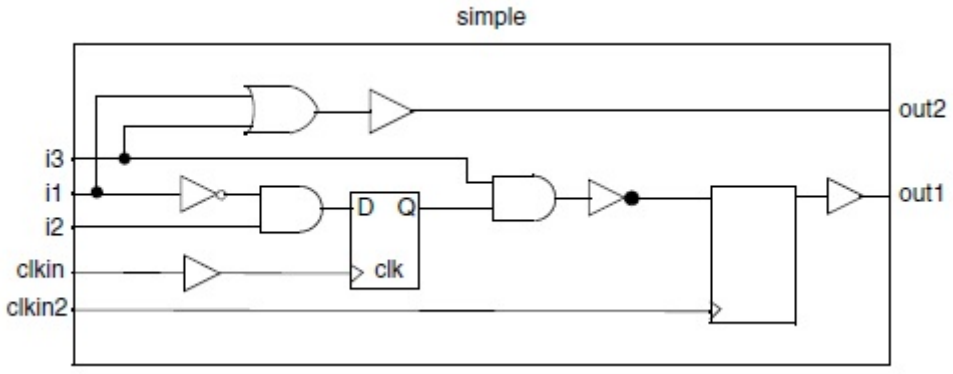

Figure 3.2: ETM generation-netlist

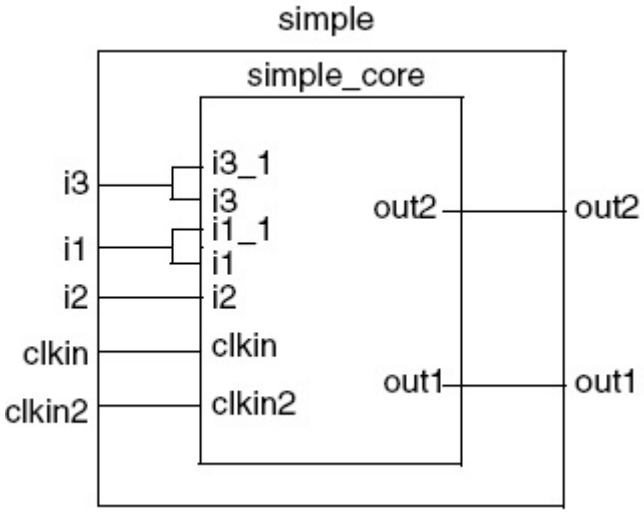

Figure 3.3: ETM generated model

The generated timing model is another design containing a single leaf cell, as shown in Figure 3. The core cell is connected directly to input and output ports of the model design. This cell contains the pin-to-pin timing arcs of the extracted model. Figure 4 shows the timing arcs of the core cell. These arcs are extracted from the timing paths of the original design.

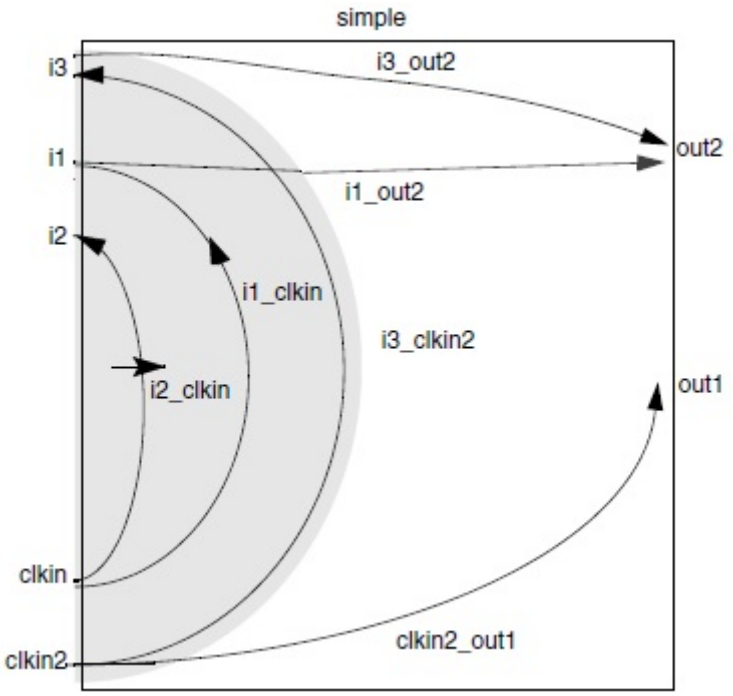

Figure 3.4: ETM model arcs

The delay data in the timing arcs is accurate for a range of operating environments. The extracted delay data does not depend on the specific values from input transition times, output capacitive loads, input arrival times, output required times, and so on. When the model is used in a design, the arc delays vary with the input transition times and output capacitive loads. This is called a context-independent model because it works correctly in a variety of contexts.

The characteristics of the extracted model depend on the operating conditions and wire load model in effect at the time of extraction. However, clocking conditions and external constraints do not affect the model extraction process. But for the validation of the model, these parameters are very important as the model is sensitive to the changes in them. Also at the time of model extraction it should be made sure that the design for which the model is being extracted is free from any violations. The extract_model is the command used for the extraction and it supports a number of switches to include the presence of latches at the interface, noise of the design. It also generates a test design model which can be used for the validation purposes. The model after generation can be further validated in order to see the functionality under same working constraints and conditions.

3.2.2 Model Validation

The extracted model has to be validated against the original netlist for the timing characteristics. During validation, there might be timing mismatches between the model and netlist. These failures could occur because of setup issues, pessimism in graph-based analysis, or limitations in modeling. The validation can be done in two ways 1. Auto

validation; 2. Manual validation. PrimeTime provides the option of the auto validation which can be used while extracting the model. It extracts the model and automatically validates it wrt netlist design. The other option is to manually validate the model.

For the manual validation, PrimeTime has provided two commands: write_interface_timing and compare_interface_timing which can be used to generate various reports and do the validation.

- **write_interface_timing**

  This command is used to write the timing report for the design wrt to the interface. It goes through all the elements that are present at the interface and generates a timing report for each of them. The parameters in the report include the worst path timing arc of each pin/port wrt to the clock, the lumped and total capacitances, transition times at each pin and design rules. The multiple arcs that can be included are setup, hold, min_seq_delay, max_seq_delay, recovery, removal, clock_gating_setup, clock_gating_hold, min_combo_delay and max_combo_delay.

- **compare_interface_timing**

  The previous command is used at the netlist level as well as model level. It generates two reports at the netlist as well as model level which can be compared in order to find the differences in the parameters.

By doing this procedure we can make sure that the model that is generated can be used at the hierarchy level instead of the whole netlist. Violations can be seen in the comparison reports which can be resolved by looking at the individual paths and changing certain parameters that can alter the delays in the design.

3.2.3 Flow/Methodology

The model extraction and validation basically follows the following flow/methodology:

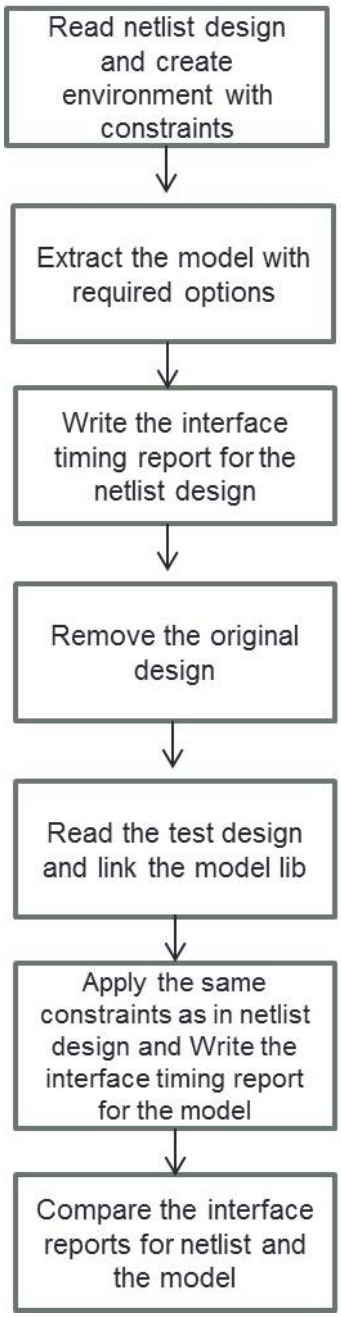

Figure 3.5: ETM methodology

- **Load the design with netlist**

  The initial step is to load the design as is with the netlist, constraints and the spef. The netlists of all the small and big modules that are required in the timing analysis of a design have to be included here in form of Verilog. The standard cell libraries have also to be included based on which the synthesized netlist is generated. The constraints that are to be specified are clock definitions, the transition time values to the specific pins/ports, setting up the false paths, timing derates to the clock and data paths, the other PrimeTime related files etc. The spef is the parasitic RC data extraction for the whole design. Hence, the whole environment has to be setup like this in PrimeTime.

- **Extract model**

  Now that the design is setup, we can extract the model for the design. We have to make sure that the design is free from violations and that many constraints like false paths, multicycle paths, clock definitions, latch declarations, removing the unnecessary constraints etc. are specified properly. Then we can extract the model with the required options. The model can be extracted using the command extract_model with appropriate switches.

- **Write the interface report**

  The interface timing report is now generated for the netlist design. The input/output constraints like input and output delays have to be applied before writing the report.

- **Model Instantiation**

  Now that the model has been extracted we will have to validate it. In order to do that we have to instantiate it in place of the netlist design. Hence we remove the original design and link the created model lib to the test design that is created. We have to apply the same constraints that we applied in the netlist design and create the same environment for the model design.

- **Interface report generation and comparison**

  Now we can write the interface timing report for the model design provided we have specified the same input/output delay constraints to that of the netlist design. The report will contain the values for the same parameters mentioned above. Then we can compare the interface timing reports for the netlist and model design.

- **Remove violations**

  The comparison report will show violations if any and we can debug further in order to determine which parameters are changing the incremental delays in the timing reports. The report_etm_arc is a command that is used at the netlist level design in order to determine the path delays for the netlist as well as the extracted model. The probable parameters that can be responsible for the changes can be derates, transition time and capacitance. By making appropriate changes, we can remove the violations.

This methodology can be incorporated in the main flow and automation can be brought in action in order to resolve the violations observed.

Chapter 4

Implementation vs. Sign-off Timing Correlation

The basic process of an IP design includes the physical design steps that are carried out in any physical implementation tool. After the physical design is finished and the design is optimized for timing, it is taken to Timing analysis tool for timing sign-off. The timing between implementation and sign-off is varying because of several parameters like AOCV derates, fill, spef etc. Also the correlation can be based on the factors like buffers/inverters added while the timing optimization is done during implementation and sign-off and increase in design area. The purpose is to reduce the timing gap between these two and introduce the use of AOCV derates in physical design and look at the impact of it on the timing and area requirements.

4.1 Introduction

The IP design follows a flow that includes the physical design steps like Floorplan, placement, CTS, hold fixing, routing, DRC and LVS checks that are performed in IC Compiler. This design after completion of all these steps is taken to PrimeTime for timing sign-off. The ECO (Engineering Change Order) is performed here and timing is optimized to make the design timing violation free. The changes in the design are taken back to ICC and the design again undergoes the physical design steps. The timing between ICC and PT can be very different because of certain parameters like AOCV, spef and fill. The description of some parameters can be given as follows:

4.1.1 Advanced On-Chip Variation derates (AOCV)

Usually, global flat derates value are applied to a design to add margin to account for On Chip Variation. While this traditional approach is a reasonable method to bind the process variations for 130-nm and 90-nm designs, it can add excessive and pessimistic margins for designs at smaller geometries and result in over-design, reduced design performance, and longer design cycles. Advanced OCV (AOCV) analysis naturally extends OCV analysis and delivers an improved method of adding margin in the design. AOCV models the random and systematic variations across an IC that affect timing by using variable derating factors that consider the location and the logic depth of each path being analyzed. By using context-specific derating values instead of a single global derating value, you can reduce excessive design margins and have fewer timing violations.

The pessimism-reducing approach of AOCV minimizes unnecessary over-design and enables you to reach timing closure more quickly and with greater confidence. PrimeTime uses derating tables to specify the AOCV information. PrimeTime calculates and applies variable derating factors that consider the location and the depth of each path being analyzed. AOCV analysis uses metrics such as path-depth and location based bounding box to calculate a context-specific AOCV derating factor to apply to the path, replacing the use of a global derating factor.

To perform graph-based AOCV analysis, PrimeTime chooses conservative values of path-depth and location-based bounding box to bind the worst-case path through a cell. PrimeTime uses the AOCV depth and bounding box metrics to lookup the appropriate derating factor from the derate tables and applied to the timing arc. The AOCV metrics computation is different for graph-based and path-based AOCV. In graph-based AOCV, the AOCV metrics are calculated for all paths through each timing arc. In path-based mode, the AOCV metrics are calculated specific to the path being analyzed.

4.1.2 AOCV file format

You must specify the AOCV derating tables in the AOCV file format. The following table types are supported:

- One-dimensional tables in either depth or distance

- Two-dimensional tables in both depth and distance

PT supports the two dimensional tables containing both distance and depth. ICC doesnt support this type of AOCV specification. It supports only single row from the specified two dimensional tables. The example of an AOCV file can be given as:

```
version: 1.0
object_type: design
rf_type: rise fall
delay_type: cell net
derate_type: early
object_spec:
depth: 0 1 2 3
distance: 100 200
table: 0.87 0.93 0.95 0.96 \
       0.83 0.85 0.87 0.90
```

Figure 4.1: AOCV file format

The design when taken for sign-off, AOCV derates are applied and timing analysis is performed. But when physical design is performed the flat derates are applied. As a result there is a difference in sign-off and implementation timing. So there are a number of experiments that are performed in order to check the timing between them and also the effects on the design area and other parameter changes when AOCV derates are introduced in implementation. The experiments can be broadly categorized into two categories:

- Introducing AOCV in implementation and compare the timing between it and sign-off.
- Observing the number of buffers required for the timing fix, timing numbers and area increase.

4.2 Application of AOCV during implementation and impact on timing

The physical design steps till now included the flat derates. Now we will introduce the AOCV derates in design steps and observe the effects of it on timing. Also the sign-off sessions will have the AOCV derates applied for all the experiments. Now for the same design we will run two sessions; 1. Whole design flow with flat derates applied and 2. With AOCV derates applied. This has been done in order to clearly see the slack differences and compare them.

There is a small procedure that has to be followed in order to properly annotate these derates in the design flow. Since the tool doesnt support the full 2 dimensional tables we have to specify the row number of the table that we want to annotate and use for the respective cells. The command to read the aocv files is given as:

```
read_aocvm <aocv_file_name>
```

This command is modified as follows:

```
read_aocvm distance_row <no. of the row> -<min/max corner> <file_name>
```

This command specifies the no. of the row that is to be read, for a min or max corner from a specified aocv file. The no of row specification is important because the value of AOCV derates increase/decrease diagonally depending upon the type of the cell it is used on. Hence with the help of this we can control the derate value applied to the cells.

Also while setting up the design and sign-off sessions, there are certain variables that are to be set to a certain value in order to enable/disable AOCV analysis in the tool. The setting of these variables is very important. The variables are listed in the table below with their corresponding values in the tools.

Variables	Sign-off	Implementation
timing_aocvm_analysis_mode	combined_launch_capture_depth	Combined_clock_and_data_metrics
timing_aocvm_ocv_precedence_ compatibility	True	True
Timing_aocvm_enable_analysis	True	True
Timing_remove_clock_reconver gence_pessimism	True	True
Pba_aocvm_only_mode	True	-Not Available-

Figure 4.2: Important required variables

4.2.1 Methodology

The following methodology is used in order to conduct the experiments and get the desired results.

- Generate the final databases from design sessions (with and w/o AOCV derates). Pick up the spef, UPF, Verilog netlist and the constraints file .sdc from the databases.

- Run the sign-off sessions for the setup and hold corners for both the databases. Now for the timing comparisons, we will keep PT as benchmark and compare the ICC slacks to it. To make sure that we are comparing the slacks of same paths, generate the timing slacks for a large number of paths and list out the startpoint, startpoint clock, endpoint and endpoint clock for each path.

- For each of these pairs, well generate a timing report in ICC and compare its slack wrt PT. After generating a histogram of all the slack differences for all the paths, well be able to see how close Implementation timing is to sign-off.

- This procedure has to be followed separately for both setup and hold corners and respective histograms have to be generated to see the timing correlation for each.

4.2.2 Results

Now in order to perform these experiments, two implementations of the same design were fired with AOCV enabled in one and flat derate in another. Also timing sessions were setup for setup and hold corners for each design session. The results that I found are as follows. The following figures show the setup histograms.

R1	R2	ACT	CUM	PER
N/A	0	0	0	0.00
<	-600	57	57	0.89
-550	-600	0	57	0.00
-500	-550	18	75	0.28
-450	-500	9	84	0.14
-400	-450	0	84	0.00
-350	-400	0	84	0.00
-300	-350	0	84	0.00
-250	-300	0	84	0.00
-200	-250	0	84	0.00
-150	-200	0	84	0.00
-100	-150	2	86	0.03
-90	-100	0	86	0.00
-80	-90	4	90	0.06
-70	-80	0	90	0.00
-60	-70	6	96	0.09
-50	-60	26	122	0.41
-40	-50	128	250	2.00
-30	-40	216	466	3.38
-20	-30	439	905	6.87
-10	-20	668	1573	10.46
0	-10	575	2148	9.00
10	0	997	3145	15.61
20	10	105	3250	1.64
30	20	135	3385	2.11
40	30	289	3674	4.52
50	40	676	4350	10.58
60	50	854	5204	13.37
70	60	786	5990	12.31
80	70	306	6296	4.79
90	80	54	6350	0.85
100	90	13	6363	0.20
150	100	24	6387	0.38
200	150	0	6387	0.00
250	200	0	6387	0.00
300	250	0	6387	0.00
350	300	0	6387	0.00
400	350	0	6387	0.00
>	400	0	6387	0.00

Figure 4.3: Setup histogram without AOCV

R1	R2	ACT	CUM	PER
N/A	0	0	0	0.00
<	-600	57	57	0.89
-550	-600	27	84	0.42
-500	-550	0	84	0.00
-450	-500	0	84	0.00
-400	-450	0	84	0.00
-350	-400	0	84	0.00
-300	-350	0	84	0.00
-250	-300	0	84	0.00
-200	-250	0	84	0.00
-150	-200	0	84	0.00
-100	-150	0	84	0.00
-90	-100	0	84	0.00
-80	-90	0	84	0.00
-70	-80	0	84	0.00
-60	-70	0	84	0.00
-50	-60	0	84	0.00
-40	-50	0	84	0.00
-30	-40	0	84	0.00
-20	-30	0	84	0.00
-10	-20	0	84	0.00
0	-10	218	302	3.39
10	0	1450	1752	22.54
20	10	825	2577	12.83
30	20	189	2766	2.94
40	30	282	3048	4.38
50	40	712	3760	11.07
60	50	1232	4992	19.15
70	60	885	5877	13.76
80	70	360	6237	5.60
90	80	140	6377	2.18
100	90	33	6410	0.51
150	100	22	6432	0.34
200	150	0	6432	0.00
250	200	0	6432	0.00
300	250	0	6432	0.00
350	300	0	6432	0.00
400	350	0	6432	0.00
>	400	0	6432	0.00

Figure 4.4: Setup histogram with AOCV

The above figures show the setup histogram for the PT session from each ICC session. The 1st and 2nd columns show the range in which the slack of the path is falling. The 3rd column shows the number of paths falling in that range. The 4th column shows the accumulative no of paths and the 5th shows the percentage of the particular range of the total paths. The first figure is from the session with flat derates and second is that with AOCV derates. As we look at Figure 2, we can see that the setup for all the paths for the session with flat derates is uniformly distributed and almost half of the paths lie in the negative slack region. That means that the implementation slack is more than the other. Now if we look at the setup values with AOCV then a significant improvement can be seen in the slack differences. The distribution has become tighter than the other and the slacks have come in the positive part of the distribution that means formers slack is becoming closer to that of the latters. Now well look at the hold results.For the timing experiments, the hold fixing has been done during implementation and ECO has been done during sign-off. Also for all the experiments, AOCV has been enabled. The below figures show the histograms for the hold corners with and without AOCV.

R1	R2	ACT	CUM	PER
N/A	0	0	0	0.00
<	-600	0	0	0.00
-550	-600	0	0	0.00
-500	-550	0	0	0.00
-450	-500	8	8	0.12
-400	-450	1	9	0.01
-350	-400	0	9	0.00
-300	-350	0	9	0.00
-250	-300	0	9	0.00
-200	-250	0	9	0.00
-150	-200	0	9	0.00
-100	-150	5	14	0.07
-90	-100	1	15	0.01
-80	-90	0	15	0.00
-70	-80	6	21	0.09
-60	-70	11	32	0.16
-50	-60	25	57	0.36
-40	-50	139	196	2.00
-30	-40	310	506	4.47
-20	-30	1289	1795	18.58
-10	-20	2165	3960	31.21
0	-10	1765	5725	25.45
10	0	1025	6750	14.78
20	10	186	6936	2.68
30	20	0	6936	0.00
40	30	0	6936	0.00
50	40	0	6936	0.00
60	50	0	6936	0.00
70	60	0	6936	0.00
80	70	0	6936	0.00
90	80	0	6936	0.00
100	90	0	6936	0.00
150	100	0	6936	0.00
200	150	0	6936	0.00
250	200	0	6936	0.00
300	250	0	6936	0.00
350	300	0	6936	0.00
400	350	0	6936	0.00
>	400	0	6936	0.00

Figure 4.5: Hold histogram without AOCV

From the figure we can see that majority of the paths fall in the negative slacks region. That means that the formers slacks are more than latters. i.e. Implementation results are more pessimistic than sign-off. Now we will look at the results for AOCV enabled design session.

R1	R2	ACT	CUM	PER
N/A	0	0	0	0.00
<	-600	0	0	0.00
-550	-600	0	0	0.00
-500	-550	0	0	0.00
-450	-500	9	9	0.13
-400	-450	0	9	0.00
-350	-400	0	9	0.00
-300	-350	1	10	0.01
-250	-300	0	10	0.00
-200	-250	0	10	0.00
-150	-200	0	10	0.00
-100	-150	0	10	0.00
-90	-100	0	10	0.00
-80	-90	0	10	0.00
-70	-80	0	10	0.00
-60	-70	0	10	0.00
-50	-60	0	10	0.00
-40	-50	0	10	0.00
-30	-40	0	10	0.00
-20	-30	0	10	0.00
-10	-20	0	10	0.00
0	-10	1142	1152	16.28
10	0	1034	2186	14.74
20	10	524	2710	7.47
30	20	3556	6266	50.70
40	30	740	7006	10.55
50	40	8	7014	0.11
60	50	0	7014	0.00
70	60	0	7014	0.00
80	70	0	7014	0.00
90	80	0	7014	0.00
100	90	0	7014	0.00
150	100	0	7014	0.00
200	150	0	7014	0.00
250	200	0	7014	0.00
300	250	0	7014	0.00
350	300	0	7014	0.00
400	350	0	7014	0.00
>	400	0	7014	0.00

Figure 4.6: Hold histogram with AOCV

From the above figure it is evident that AOCV enablement brings the slacks in Implementation closer to sign-off ultimately leading to bringing the majority of the paths in the positive slacks range. Hence we can conclude that the timing between sign off and implementation is becoming closer and tighter by introduction of AOCV in design steps.

4.3 Impact of AOCV on buffer count and TNS/WNS

During the physical design steps, after the Clock Tree Synthesis (CTS) stage the tool performs a hold optimization. In this stage, for hold optimization, it adds buffers to all the paths in order to decrease the slacks and thus reducing the Worst Negative Slack (WNS) and Total Negative Slack (TNS). In order to determine how the AOCV usage is impacting the buffer count and WNS TNS of design, I ran multiple sessions of the same design with different configuration.

4.3.1 Configuration of experiments

The Implementation sessions were configured in four different aspects keeping the AOCV enablement and hold fixing to be variables.

- AOCV enabled and hold fixing performed (AOCV_HOLD).
- AOCV enabled and hold fixing not performed (AOCV_NOHOLD).
- AOCV disabled and hold fixing not performed (NOAOCV_NOHOLD).
- AOCV disabled and hold fixing performed (NOAOCV_HOLD).

These configurations define the basic platform on which we can determine how the AOCV application is affecting the hold optimization, and whether or not not fixing hold in Implementation helps to our purpose. Whether or not AOCV is applied or hold fixing is done,, when the design is taken for sign-off, ECO is performed which optimizes the design timing with AOCV enabled. This will also lead to changes in the buffer count and TNS/WNS of the design.

In the AOCV tables, as you move from the first element towards the last element diagonally, the derate value increases. By default, design tool takes the last row of the table unless specified like discussed in the section 4.2 above. Hence we will specifically provide the number of row to be 0 in order to make slacks less pessimistic and avoid over-optimization.

4.3.2 Flow/Methodology

- According to the configuration, change the settings in the flow such that the desired design parameters are obtained. For example, if we want a design with AOCV_NOHOLD configuration, then make sure that the variables are being set to their values properly and that the AOCV files are being sourced completely. Also make sure that the design does not undergo the hold optimization stage.
- After the design is finished, dump out a qor report. This report contains the number of buffers/inverters present in the design. Also it contains the TNS/WNS of the design at that stage.
- Setup the timing sessions for each of these design sessions as discussed earlier for both setup and hold corners. These timing sessions have AOCV enabled with the variables set to the values specified.
- Now perform the ECO on the design, so that the timing optimization is done. Here also buffer addition/removal is done here in order to optimize hold. Generate a qor report and determine the number of buffers added and the final TNS and WNS. The entire path timing reporting is done path based in PT because it gives a clearer picture regarding the timing.

4.3.3 Results

After doing all the experiments, we have the values of number of buffers that are added for each configuration at the respective stage and also TNS and WNS for each of them. They can be tabulated as follows:

Configuration	PT-Buffers	ICC-Buffers	Total Buffers	Pre-Fix-PT-hold WNS/TNS		Post-fix-PT-hold WNS/TNS	
				WNS	TNS	WNS	TNS
AOCV_HOLD	364	18390	35700	-1100	-8069	-273.7	-544
AOCV_NOHOLD	3643	7312	24291	-1096	-45234	-273	-800
NOAOCV_HOLD	1705	13000	30264	-1097	-23960	-273	-690
NOAOCV_NOHOLD	3570	7300	24300	-1096	-43133	-274	-689

Figure 4.7: Buffer count and timing values

Configuration	Pre-ICC		Post- ICC		Pre-Fix-PT-setup WNS/TNS		Post-fix-PT-setup WNS/TNS	
	WNS	TNS	WNS	TNS	WNS	TNS	WNS	TNS
AOCV_HOLD	-303	-7932	-86.2	-2392	-1087	-60200	-1080	-60200
AOCV_NOHOLD	-337	-150155	-131	-26800	-2048	-97570	-2020	-102200
NOAOCV_HOLD	-92	-2736	-64	-1400	-1088	-57122	-1080	-57100
NOAOCV_NOHOLD	-294	-49600	-62	-3367	-2079	-95361	-2070	-101500

Figure 4.8: Timing values

The above tables depict the results for the multiple experiments performed. The pre and post hold values are from PT, taken before and after the ECO is performed. Same thing is valid for setup values. The pre ICC values are obtained from the final database of the design before it is taken to sign-off. After performing ECO, changes are reverted back to implementation tool for design improvement and the post ICC values

are obtained after the changes are implemented faithfully.

The buffer counts refer to the number of buffers added at a particular stage in PT or ICC. PT buffers are the number of buffers added in PT after performing ECO. Same goes for the ICC when it fixes hold in one of its design stage. The total is the total numbers of buffers in a design after all the fixes are performed.

Now well look at the results wrt the buffers added and the final timing values obtained. Of all the sessions, the hold values for the sessions where hold fixing is done in ICC are the best. The remaining two sessions have bad values for hold timing. So from a practical standpoint, these configurations are less useful than others.

Now the other two sessions where the hold fixing is performed has AOCV enabled and disabled. The session where it is enabled, has better hold nos. than the other one. But, there is another side of it. If we look at the buffer count of both the sessions, the latter is far better than the former. There is an almost 5k difference in buffer count between them and that too for the same hold values. So this is a negative point for the former. Also the setup values and ICC hold values for the latter is better than former.

So when there are area considerations, the latter has an edge over the former and should be used for the design purposes. Hence, we can conclude that even though the AOCV application brings the timing between sign-off and implementation closer, the buffer count values and final timing values are affected to some extent and hence the AOCV application cannot be used during design in ICC.

Chapter 5

Conclusion

The IP group delivers the quality IPs to various vendors for integration to the SoCs and processor chips. Now as the complexity of IPs and technology is growing leaps and bounds, there has arisen a need to reduce the sign off time and delivery time of the IP. The IP block contains various digital and analog blocks/CBBs (Custom Building Blocks) put together. The signoff is done by putting the digital and CBBs opened up in the form of Verilogs. This leads to a greater runtime, huge usage of resources, large memory requirements and large violation resolution time. As a result, the idea of boxing up the CBBs has come into picture.

This can be done by generating the timing models for the CBBs individually. The generated blocks can be validated then using the derived ETM validation methodology so that we can make sure that the lib when used at the upper level of hierarchy will not cause any violations. Hence during signoff process these libs for the CBBs can be directly used in place of the whole netlist design.

The advantages of boxing up the CBBs are many; firstly the runtime for the whole design comes down significantly. Secondly the resource usage and memory requirements also decrease. The SPEF that has to be generated at the main design level does not contain the CBB SPEFs; the constraints that are used also contain the most abstract constraints for the CBBs. As a result, the complexity of the STA environment also decreases.

Also, the whole designing today is done using automated tools and a number of parameters are incorporated while using them. One of such tools is PrimeTime, which is primarily focused to focus on the timing part of a design. The timing sign-off is usually done in PT.

In order to reduce the iterations during sign-off and bring the ICC timing closer to PT, several parameters were tested like introduction to AOCV in ICC. The results were generated by doing multiple experiments and it was concluded that even though the overall ICC timing is becoming closer to PT, it is affecting the other design parameters like area and timing values. Hence the use of AOCV will not be incorporated in the flow.

References

1) Synopsys,*PrimeTime User Guide*,H-2013.
2) J.Bhasker, Rakesh Chadha, *Static Timing Analysis for Nanometer Designs.*
3) Synopsys,*ETM Appnote 1.0.*
4) Intelpedia.
5) Synopsys,*IC Compiler User Guide*,H-2013.
6) Synopsys,*AOCVM Appnote 1.0.*
7) INTEL Internal Documents.
8) Synopsys,*CCS Noise Liberty Syntax.*