Static Timing Analysis for Nanometer Designs

A Practical Approach

J. Bhasker • Rakesh Chadha

Static Timing Analysis for Nanometer Designs
A Practical Approach

Springer

J. Bhasker
eSilicon Corporation
1605 N. Cedar Crest Blvd.
Suite 615
Allentown, PA 18103, USA
jbhasker@esilicon.com

Rakesh Chadha
eSilicon Corporation
890 Mountain Ave
New Providence, NJ 07974, USA
rchadha@esilicon.com

ISBN 978-0-387-93819-6 e-ISBN 978-0-387-93820-2
DOI: 10.1007/978-0-387-93820-2

Library of Congress Control Number: 2009921502

Printed on acid-free paper.

springer.com

Contents

❑

Preface

Timing, timing, timing! That is the main concern of a digital designer charged with designing a semiconductor chip. What is it, how is it described, and how does one verify it? The design team of a large digital design may spend months architecting and iterating the design to achieve the required timing target. Besides functional verification, the timing closure is the major milestone which dictates when a chip can be released to the semiconductor foundry for fabrication. This book addresses the timing verification using static timing analysis for nanometer designs.

The book has originated from many years of our working in the area of timing verification for complex nanometer designs. We have come across many design engineers trying to learn the background and various aspects of static timing analysis. Unfortunately, there is no book currently available that can be used by a working engineer to get acquainted with the details of static timing analysis. The chip designers lack a central reference for information on timing, that covers the basics to the advanced timing verification procedures and techniques.

The purpose of this book is to provide a reference for both beginners as well as professionals working in the area of static timing analysis. The book

is intended to provide a blend of the underlying theoretical background as well as in-depth coverage of timing verification using static timing analysis. The book covers topics such as cell timing, interconnect, timing calculation, and crosstalk, which can impact the timing of a nanometer design. It describes how the timing information is stored in cell libraries which are used by synthesis tools and static timing analysis tools to compute and verify timing.

This book covers CMOS logic gates, cell library, timing arcs, waveform slew, cell capacitance, timing modeling, interconnect parasitics and coupling, pre-layout and post-layout interconnect modeling, delay calculation, specification of timing constraints for analysis of internal paths as well as IO interfaces. Advanced modeling concepts such as composite current source (CCS) timing and noise models, power modeling including active and leakage power, and crosstalk effects on timing and noise are described.

The static timing analysis topics covered start with verification of simple blocks particularly useful for a beginner to this area. The topics then extend to complex nanometer designs with concepts such as modeling of on-chip variations, clock gating, half-cycle and multicycle paths, false paths, as well as timing of source synchronous IO interfaces such as for DDR memory interfaces. Timing analyses at various process, environment and interconnect corners are explained in detail. Usage of hierarchical design methodology involving timing verification of full chip and hierarchical building blocks is covered in detail. The book provides detailed descriptions for setting up the timing analysis environment and for performing the timing analysis for various cases. It describes in detail how the timing checks are performed and provides several commonly used example scenarios that help illustrate the concepts. Multi-mode multi-corner analysis, power management, as well as statistical timing analyses are also described.

Several chapters on background reference materials are included in the appendices. These appendices provide complete coverage of SDC, SDF and SPEF formats. The book describes how these formats are used to provide information for static timing analysis. The SDF provides cell and interconnect delays for a design under analysis. The SPEF provides parasitic information, which are the resistance and capacitance networks of nets in a

design. Both SDF and SPEF are industry standards and are described in detail. The SDC format is used to provide the timing specifications or constraints for the design under analysis. This includes specification of the environment under which the analysis must take place. The SDC format is a defacto industry standard used for describing timing specifications.

The book is targeted for professionals working in the area of chip design, timing verification of ASICs and also for graduate students specializing in logic and chip design. Professionals who are beginning to use static timing analysis or are already well-versed in static timing analysis can use this book since the topics covered in the book span a wide range. This book aims to provide access to topics that relate to timing analysis, with easy-to-read explanations and figures along with detailed timing reports.

The book can be used as a reference for a graduate course in chip design and as a text for a course in timing verification targeted to working engineers. The book assumes that the reader has a background knowledge of digital logic design. It can be used as a secondary text for a digital logic design course where students learn the fundamentals of static timing analysis and apply it for any logic design covered in the course.

Our book emphasizes practicality and thorough explanation of all basic concepts which we believe is the foundation of learning more complex topics. It provides a blend of theoretical background and hands-on guide to static timing analysis illustrated with actual design examples relevant for nanometer applications. Thus, this book is intended to fill a void in this area for working engineers and graduate students.

The book describes timing for CMOS digital designs, primarily synchronous; however, the principles are applicable to other related design styles as well, such as for FPGAs and for asynchronous designs.

Book Organization

The book is organized such that the basic underlying concepts are described first before delving into more advanced topics. The book starts

with the basic timing concepts, followed by commonly used library modeling, delay calculation approaches, and the handling of noise and crosstalk for a nanometer design. After the detailed background, the key topics of timing verification using static timing analysis are described. The last two chapters focus on advanced topics including verification of special IO interfaces, clock gating, time borrowing, power management and multi-corner and statistical timing analysis.

Chapter 1 provides an explanation of what static timing analysis is and how it is used for timing verification. Power and reliability considerations are also described. Chapter 2 describes the basics of CMOS logic and the timing terminology related to static timing analysis.

Chapter 3 describes timing related information present in the commonly used library cell descriptions. Even though a library cell contains several attributes, this chapter focuses only on those that relate to timing, crosstalk, and power analysis. Interconnect is the dominant effect on timing in nanometer technologies and Chapter 4 provides an overview of various techniques for modeling and representing interconnect parasitics.

Chapter 5 explains how cell delays and paths delays are computed for both pre-layout and post-layout timing verification. It extends the concepts described in the preceding chapters to obtain timing of an entire design.

In nanometer technologies, the effect of crosstalk plays an important role in the signal integrity of the design. Relevant noise and crosstalk analyses, namely glitch analysis and crosstalk analysis, are described in Chapter 6. These techniques are used to make the ASIC behave robustly from a timing perspective.

Chapter 7 is a prerequisite for succeeding chapters. It describes how the environment for timing analysis is configured. Methods for specifying clocks, IO characteristics, false paths and multicycle paths are described in Chapter 7. Chapter 8 describes the timing checks that are performed as part of various timing analyses. These include amongst others - setup, hold and asynchronous recovery and removal checks. These timing checks are intended to exhaustively verify the timing of the design under analysis.

Chapter 9 focuses on the timing verification of special interfaces such as source synchronous and memory interfaces including DDR (Double Data Rate) interfaces. Other advanced and critical topics such as on-chip variation, time borrowing, hierarchical methodology, power management and statistical timing analysis are described in Chapter 10.

The SDC format is described in Appendix A. This format is used to specify the timing constraints of a design. Appendix B describes the SDF format in detail with many examples of how delays are back-annotated. This format is used to capture the delays of a design in an ASCII format that can be used by various tools. Appendix C describes the SPEF format which is used to provide the parasitic resistance and capacitance values of a design.

All timing reports are generated using PrimeTime, a static timing analysis tool from Synopsys, Inc. Highlighted text in reports indicates specific items of interest pertaining to the explanation in the accompanying text.

New definitions are highlighted in **bold**. Certain words are highlighted in *italics* just to keep the understanding that the word is special as it relates to this book and is different from the normal English usage.

Acknowledgments

We would like to express our deep gratitude to eSilicon Corporation for providing us the opportunity to write this book.

We also would like to acknowledge the numerous and valuable insights provided by Kit-Lam Cheong, Ravi Kurlagunda, Johnson Limqueco, Pete Jarvis, Sanjana Nair, Gilbert Nguyen, Chris Papademetrious, Pierrick Pedron, Hai Phuong, Sachin Sapatnekar, Ravi Shankar, Chris Smirga, Bill Tuohy, Yeffi Vanatta, and Hormoz Yaghutiel, in reviewing earlier drafts of the book. Their feedback has been invaluable in improving the quality and usefulness of this book.

Last, but not least, we would like to thank our families for their patience during the development of this book.

Dr. Rakesh Chadha
Dr. J. Bhasker

January 2009

CHAPTER

1

Introduction

This chapter provides an overview of the static timing analysis procedures for nanometer designs. This chapter addresses questions such as, what is static timing analysis, what is the impact of noise and crosstalk, how these analyses are used and during which phase of the overall design process are these analyses applicable.

1.1 Nanometer Designs

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 submi-

J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*,
DOI: 10.1007/978-0-387-93820-2_1, © Springer Science + Business Media, LLC 2009

cron or nanometer process technologies[1], 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.

1.2 What is Static Timing Analysis?

Static Timing Analysis (also referred as STA) is one of the many techniques available to verify the timing of a digital design. An alternate approach used to verify the timing is the timing simulation which can verify the functionality as well as the timing of the design. The term *timing analysis* is used to refer to either of these two methods - static timing analysis, or the timing simulation. Thus, timing analysis simply refers to the analysis of the design for timing issues.

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 1-1 shows the basic functionality of static

1. Deep submicron refers to process technologies with a feature size of 0.25µm or lower. The process technologies with feature size below 0.1µm are referred to as *nanometer technologies*. Examples of such process technologies are 90nm, 65nm, 45nm, and 32nm. The finer process technologies normally allow a greater number of metal layers for interconnect.

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.

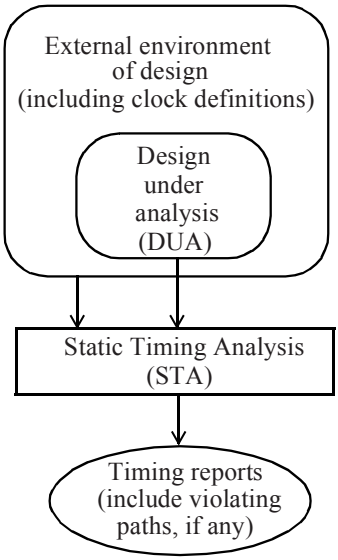

Figure 1-1 *Static timing analysis.*

The more important aspect of static timing analysis is that the entire design is analyzed once and the required timing checks are performed for all possible paths and scenarios of the design. Thus, STA is a complete and exhaustive method for verifying the timing of a design.

The design under analysis is typically specified using a hardware description language such as VHDL[1] or Verilog HDL[2]. The external environment, including the clock definitions, are specified typically using SDC[3] or an equivalent format. SDC is a timing constraint specification language. The timing reports are in ASCII form, typically with multiple columns, with each column showing one attribute of the path delay. Many examples of timing reports are provided as illustrations in this book.

1.3 Why Static Timing Analysis?

Static timing analysis 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[4], which may contain 10 to 100 million gates, the static timing analysis has become a necessity to exhaustively verify the timing of a design.

Crosstalk and Noise

The design functionality and its performance can be limited by noise. The noise occurs due to crosstalk with other signals or due to noise on primary inputs or the power supply. The noise impact can limit the frequency of

1. See [BHA99] in Bibliography.
2. See [BHA05] in Bibliography.
3. Synopsys Design Constraints: It is a defacto standard but a proprietary format of Synopsys, Inc.
4. Application-Specific Integrated Circuit.

operation of the design and it can also cause functional failures. Thus, a design implementation must be verified to be robust which means that it can withstand the noise without affecting the rated performance of the design.

Verification based upon logic simulation cannot handle the effects of crosstalk, noise and on-chip variations.

The analysis methods described in this book cover not only the traditional timing analysis techniques but also noise analysis to verify the design including the effects of noise.

1.4 Design Flow

This section primarily describes the CMOS[1] digital design flow in the context used in the rest of this book. A brief description of its applicability to FPGAs[2] and to asynchronous designs is also provided.

1.4.1 CMOS Digital Designs

In a CMOS digital design flow, the static timing analysis can be performed at many different stages of the implementation. Figure 1-2 shows a typical flow.

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 behavioral level. Once a design at the RTL level has been synthesized to 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

1. Complimentary Metal Oxide Semiconductor.
2. Field Programmable Gate Array: Allows for design functionality to be programmed by the user after manufacture.

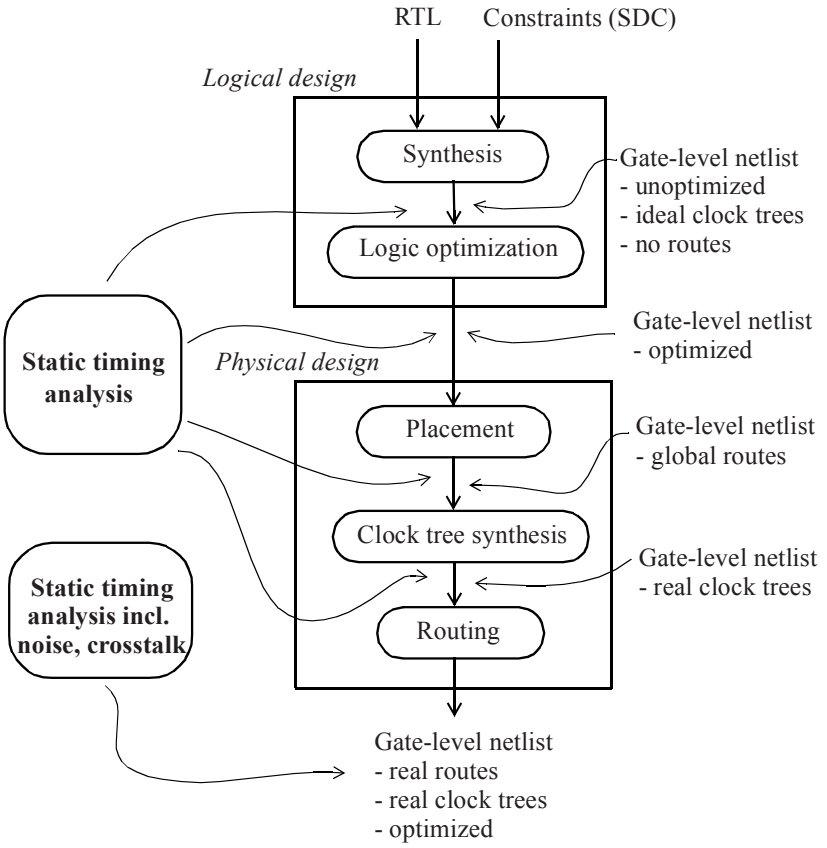

Figure 1-2 *CMOS digital design flow.*

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 after clock trees are built, STA can be performed to check the timing again. In fact,

during physical design, STA can be performed at each and every step to identify the worst paths.

In physical implementation, the logic cells are connected by interconnect metal traces. The parasitic RC (**R**esistance and **C**apacitance) 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. Thus, any analysis of the design should evaluate the impact of the interconnect on the performance characteristics (speed, power, etc.). As mentioned previously, coupling between signal traces contributes to noise, and the design verification must include the impact of the noise on the performance.

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 can not 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.

To summarize, the static timing analysis can be performed on a gate-level netlist depending on:

i. How interconnect is modeled - ideal interconnect, wireload model, global routes with approximate RCs, or real routes with accurate RCs.

ii. How clocks are modeled - whether clocks are ideal (zero delay) or propagated (real delays).

iii. Whether the coupling between signals is included - whether any crosstalk noise is analyzed.

Figure 1-2 may seem to imply that STA is done outside of the implementation steps, that is, STA is done after each of the synthesis, logic optimization, and physical design steps. In reality, each of these steps perform integrated (and incremental) STA within their functionality. For example, the timing analysis engine within the logic optimization step is used to identify critical paths that the optimizer needs to work on. Similarly, the integrated timing analysis engine in a placement tool is used to maintain the timing of the design as layout progresses incrementally.

1.4.2 FPGA Designs

The basic flow of STA is still valid in an FPGA. Even though routing in an FPGA is constrained to channels, the mechanism to extract parasitics and perform STA is identical to a CMOS digital design flow. For example, STA can be performed assuming interconnects as ideal, or using a wireload model, assuming clock trees as ideal or real, assuming global routes, or using real routes for parasitics.

1.4.3 Asynchronous Designs

The principles of STA are applicable in asynchronous designs also. One may be more interested in timing from one signal in the design to another as opposed to doing setup and hold checks which may be non-existent. Thus, most of the checks may be point to point timing checks, or skew

checks. The noise analysis to analyze the glitches induced due to coupling are applicable for any design - asynchronous or synchronous. Also, the noise analysis impact on timing, including the effect of the coupling, is valid for asynchronous designs as well.

1.5 STA at Different Design Phases

At the logical level (gate-level, no physical design yet), STA can be carried out using:

i. Ideal interconnect or interconnect based on wireload model.

ii. Ideal clocks with estimates for latencies and jitter.

During the physical design phase, in addition to the above modes, STA can be performed using:

i. Interconnect - which can range from global routing estimates, real routes with approximate extraction, or real routes with signoff accuracy extraction.

ii. Clock trees - real clock trees.

iii. With and without including the effect of crosstalk.

1.6 Limitations of Static Timing Analysis

While the timing and noise analysis do an excellent job of analyzing a design for timing issues under all possible situations, the state-of-the-art still does not allow STA to replace simulation completely. This is because there are some aspects of timing verification that cannot yet be completely captured and verified in STA.

Some of the limitations of STA are:

i. *Reset sequence*: To check if all flip-flops are reset into their required logical values after an asynchronous or synchronous reset. This is something that cannot be checked using static timing analysis. The chip may not come out of reset. This is because certain declarations such as initial values on signals are not synthesized and are only verified during simulation.

ii. *X-handling*: The STA techniques only deal with the logical domain of logic-0 and logic-1 (or high and low), rise and fall. An unknown value *X* in the design causes indeterminate values to propagate through the design, which cannot be checked with STA. Even though the noise analysis within STA can analyze and propagate the glitches through the design, the scope of glitch analysis and propagation is very different than the *X* handling as part of the simulation based timing verification for nanometer designs.

iii. *PLL settings*: PLL configurations may not be loaded or set properly.

iv. *Asynchronous clock domain crossings*: STA does not check if the correct clock synchronizers are being used. Other tools are needed to ensure that the correct clock synchronizers are present wherever there are asynchronous clock domain crossings.

v. *IO interface timing*: It may not be possible to specify the IO interface requirements in terms of STA constraints only. For example, the designer may choose detailed circuit level simulation for the DDR[1] interface using SDRAM simulation models. The simulation is to ensure that the memories can be read from and written to with adequate margin, and that the DLL[2], if any, can be controlled to align the signals if necessary.

1. **D**ouble **D**ata **R**ate.
2. **D**elay **L**ocked **L**oop.

vi. *Interfaces between analog and digital blocks*: Since STA does not deal with analog blocks, the verification methodology needs to ensure that the connectivity between these two kinds of blocks is correct.

vii. *False paths*: The static timing analysis verifies that timing through the logic path meets all the constraints, and flags violations if the timing through a logic path does not meet the required specifications. In many cases, the STA may flag a logic path as a failing path, even though logic may never be able to propagate through the path. This can happen when the system application never utilizes such a path or if mutually contradictory conditions are used during the sensitization of the failing path. Such timing paths are called *false paths* in the sense that these can never be realized. The quality of STA results is better when proper timing constraints including false path and multicycle path constraints are specified in the design. In most cases, the designer can utilize the inherent knowledge of the design and specify constraints so that the false paths are eliminated during the STA.

viii. *FIFO pointers out of synchronization*: STA cannot detect the problem when two finite state machines expected to be synchronous are actually out of synchronization. During functional simulations, it is possible that the two finite state machines are always synchronized and change together in lock-step. However, after delays are considered, it is possible for one of the finite state machines to be out of synchronization with the other, most likely because one finite state machine comes out of reset sooner than the other. Such a situation can not be detected by STA.

ix. *Clock synchronization logic*: STA cannot detect the problem of clock generation logic not matching the clock definition. STA assumes that the clock generator will provide the waveform as specified in the clock definition. There could be a bad optimization performed on the clock generator logic that causes, for example, a large delay to be inserted on one of the paths that