Reduction of Crosstalk Pessimism with Consideration of Logic and Timing Correlations

Murthy Palla

Universität Bremen

2009

Reduction of Crosstalk Pessimism with Consideration of Logic and Timing Correlations

Vom Fachbereich für Physik und Elektrotechnik
der Universität Bremen

zur Erlangung des akademischen Grades eines
DOKTOR-INGENIEUR (Dr.-Ing.)
genehmigte Dissertation

von

Murthy Palla
aus Ganapavaram, Indien

Referent: **Prof. Dr.-Ing. Walter Anheier**
Korreferent: **Prof. Dr.-Ing. Rolf Drechsler**

Eingereicht am: 22-06-2009
Tag des Promotionskolloquiums: 24-06-2009

"Fill the brain with high thoughts, highest ideals, place them day and night before you, and out of that will come great work."

Swamy Vivekananda

Acknowledgments

This thesis is an outcome of years of research, during which I worked with a number of people whose assorted contributions helped me in bringing it to its current state. I take the pleasure to express my deep gratitude to all of them in this relatively short acknowledgment.

In the first place, I would like to convey my gratitude to Prof. Dr. Walter Anheier for his supervision, advice and guidance in every respect. His unflinching support and encouragement was a source of great enthusiasm and vitality. I am very much indebted to him.

I am very much grateful to Prof. Dr. Rolf Drechsler for agreeing to be my second supervisor. He has always been a backbone for this research. His involvement in this research has strengthened my roots and nourished my intellectual maturity, which I will benefit from through the rest of my life.

I express my sincere thanks to Mr. Jens Bargfrede of Infineon Technologies AG, Munich, Germany for mentoring and guiding me at Infineon. This thesis would not have been possible without his continuous support and encouragement. I am grateful to him in every possible way for being an excellent source of inspiration.

My thanks go to Mrs. Irmtraud Rugen-Herzig of Infineon Technologies AG for being an excellent manager and always supporting me in performing high quality research at Infineon. I am very much indebted to my colleague Klaus Koch for introducing me to this field of work and encouraging me in all respects during my stay with Infineon. I also express my sincere thanks to my other colleagues Michael Mirbeth, Robert Häussler, Harald Kinzelbach, Heinrich Häner, Petra Nordholz, Engelhart Jörg and Adalbert Perbandt for being very friendly and helpful during my stay with Infineon. I specially thank Hannellore Boehm-Burgardt for being so friendly and excellently organizing all the administrative issues related to my work with Infineon.

I thank my colleagues Stephan Eggersglüß, Daniel Tille, Robert Wille, Andere Sülflow, Sebastian Offermann, Ajoy Palit and Kishore Duganapalli of University of Bremen for being open for knowledge sharing and collaborations. My discussions with them have been a source of new ideas which helped me in shaping this thesis as an excellent piece of work.

I am very much thankful to all my friends who stand as a reason for the beauty of my life for staying with me with a helping hand in times of hardship and happiness. Special mention should be made to the continuous encouragement and support of Chakrapani Bommaraju during my stay in Germany.

I do not attempt to thank my parents, who have been very caring, lovely and unconditionally supporting throughout my life. Instead, I thank God for blessing me with such wonderful parents. I warmly thank my brother, sister and their better halves for always praying for my success and well-being, and for being such a nice family.

Words cannot express my appreciation to my wife Veena for being so lovely, caring and staying beside me with a deep sense of togetherness. I thank her parents and sister for their unequivocal support and affection.

Abstract

Accurate estimation of crosstalk has become a key issue in Static Timing Analysis of modern deep–submicron (DSM) digital circuits. The inherent logic and timing properties of the circuit are often neglected in the crosstalk estimation process resulting in an overly pessimistic analysis. False Noise Analysis aims at estimating the worst realizable crosstalk that is logically and temporally valid. Although the false noise problem has been widely studied, due to its NP–hard nature, it still lacks a very efficient solution. In this thesis, this problem of false noise is studied in detail and three novel techniques which improve the speed and accuracy of this process are proposed. The contributions of this thesis can be classified into two sections: speed improvement techniques and conservative analysis techniques:

Speed improvement techniques: The novel speed improvement techniques called Simple Aggressor Ordering (SAO) and Adaptive Bounding (AB) exploit problem specific knowledge to prune a large portion of the infeasible search space of the underlying branch and bound algorithm. The beneficence of these methods lies in the fact that they simultaneously improve the speed, solvability and accuracy (stemming from the ability to solve all the cases, which are otherwise infeasible).

Conservative analysis techniques: One of the drawbacks of the state-of-the-art techniques to solve the false noise problem is that they are either non–conservative or inefficient. This issue is addressed by the novel Timing Arc Based Logic Analysis (TABLA) technique that uses dedicated solvers to solve the temporal and logical problems. TABLA uses timing arcs as basic elements to perform an efficient temporal logic analysis employing the min–max timing model using dedicated solvers for logic and timing.

The proposed techniques have been implemented and tested in an industrial environment on several industrial and benchmark circuits, and results are provided.

Kurzfassung

Die genaue Abschätzung des Übersprechens ist ein Scilülerythema der statischen Laufzeitanalyse digitaler Schaltungen moderner Technologien. Die inhärenten logischen und zeitlichen Eigenschaften der Schaltung werden bei der Abschätzung des Übersprechens oft vernachlässigt, was zu einer übermässig pessimistischen Analyse führt. Die False-Noise-Analyse hat das Ziel das logisch und zeitlich gültige, maximale Übersprechen zu bestimmen. Obwohl die False-Noise-Analyse vielfach untersucht wurde, fehlt aufgrund der NP-Vollständigkeit des Problems weiterhin eine sehr effiziente Lösung. In dieser Arbeit wird das Problem der False-Noise-Analyse im Detail untersucht und drei neuartige Techniken zur Verbesserung der Geschwindigkeit und Genauigkeit des Verfahrens vorgeschlagen. Die Beiträge dieser Arbeit lassen sich in zwei Abschnitte unterteilen, die Verbesserung der Ausführungsgeschwindigkeit und konservative Analyse-Techniken :

Verbesserung der Ausführungsgeschwindigkeit: Die neuartigen Ansätze zur Verbesserung der Ausführungsgeschwindigkeit, Einfache Aggressor Sortierung (engl. Simple Aggressor Ordering – SAO) und Adaptive Schrankenbestimmung (engl. Adaptive Bounding – AB), nutzen problemspezifisches Wissen zur Reduzierung eines grossen Teils des Suchraums des zugrundeliegenden Branch-and-Bound Algorithmus. Das Bestechende an diesen Methoden ist die Tatsache, dass sie gleichzeitig die Ausführungsgeschwindigkeit, Genauigkeit und Lösbarkeit verbessern.

Konservative Analyse-Techniken: Einer der Nachteile der bekannten Verfahren zur Lösung des False-Noise-Problems ist, dass sie entweder nicht konservativ oder ineffizient sind. Beide Probleme werden in dieser Arbeit durch den neuartigen Ansatz der Timing Arc Basierten Logik Analyse (TABLA) gelöst. TABLA verwendet eine zeitbehaftete Modellierung der Schaltvorgänge (engl. timing arc) als Grundelement, um eine effiziente temporale Logikanalyse basierend auf dem min-max Zeit-Modell mittels dedizierter Lösungsverfahren für Logik- und Zeitbereich durchzuführen.

Die beschriebenen Techniken wurden anhand mehrerer Industrie- und Vergleichsschaltkreise getestet, und die Ergebnisse dazu werden vorgestellt.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the early years of digital computing, the operating frequencies were so low that the digital signals could actually be seen as zeros and ones. Analog modeling of digital signals was not necessary at that time. With the advances in technology and voracious market needs, operating frequencies have increased, feature sizes have decreased and wire dimensions have reduced. This has resulted in an increase of parasitic couplings among the interconnect wires making crosstalk effects more and more significant. At the same time, increased frequencies and reduced threshold voltages have further worsened the scenario by increasing the influence of crosstalk in producing delay and noise, thus making the DSM designs more susceptible to crosstalk. This has made crosstalk a word of great significance in the semiconductor industry today. Accurate estimation of crosstalk without giving any room for pessimism has become essential for an early design closure. The problem of reduction of crosstalk pessimism with consideration of logic and timing correlations is a hot topic of research [1, 2, 4, 9, 14, 19, 32, 33, 34, 40, 42, 49, 50, 51, 52, 53, 58, 61, 64, 70].

This PhD research aims at providing efficient and effective false noise reduction solutions to a proprietary STA tool, which targets to perform a highly accurate timing analysis. All the assumptions made by the new techniques proposed in this thesis have high practical relevance and are in line with the requirement to maintain high accuracy. The real challenge of this research work lies in the fact that it has to solve several underlying NP-hard problems in a very cost effective manner, while making no simplistic assumptions.

The rest of this chapter is organized as follows: Section 1.1 briefly reviews crosstalk, its effects and explains how they are measured. Section 1.2 explains the problem

of false noise with an example. The major goals of this thesis are discussed in Section 1.3. Section 1.4 explains the organization of this thesis.

1.1 Crosstalk

Crosstalk is defined as the noise induced by the parasitic coupling between on-chip wires. In other words, it is the interaction of two or more adjacent lines due to parasitic coupling between a switching line (aggressor) and the (victim) line where the noise is induced. Crosstalk can basically be of three types: capacitive, inductive and resistive. However, it is a common practice to consider only capacitive crosstalk as it is more significant than the other two types for the current generation digital circuits. In this thesis, we consider only capacitive crosstalk.

In this section, the basics of crosstalk are explained. For in-depth discussions on crosstalk analysis, the reader is referred to [15, 30, 56, 57, 58, 65].

1.1.1 Effects of crosstalk

Depending on the switching state of the victim relative to that of its aggressor(s), crosstalk can induce either noise or delay on the victim line. Both these effects are explained with examples in the following sections.

Crosstalk noise

A silent victim with its aggressors switching would have a glitch on its signal which can result in a functional (logical) failure if it crosses the switching threshold of the subsequent cell. This noise on the victim line due to crosstalk is called *crosstalk noise*.

Consider the example in Figure 1.1, where the victim is silent at logic 0 while the aggressor is switching from logic 0 to logic 1. C_x is the parasitic coupling capacitance between the two nets. A crosstalk glitch occurs when a victim (typically with low driver strength) is influenced to change its state by its aggressor(s) (typically of higher driver strength and thus a sharp switching rate). A functional error occurs potentially if the amplitude of this glitch is higher than the switching threshold of the subsequent cell. The type of glitch formed on the victim depends

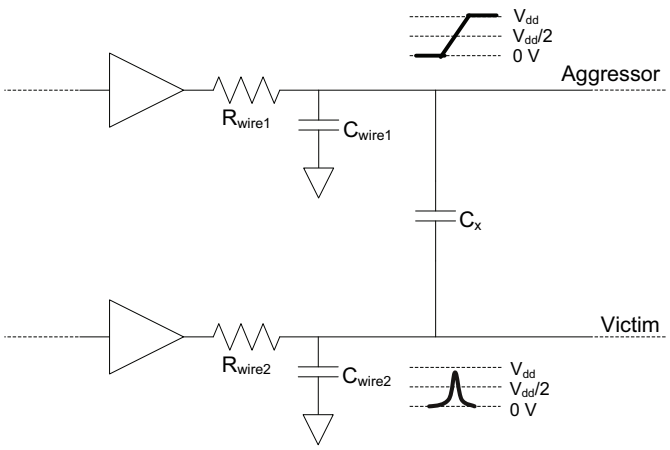


FIGURE 1.1: Crosstalk induced glitch

on the states of the victim and its aggressor(s). Different types of crosstalk glitches can be seen in Figure 1.2.

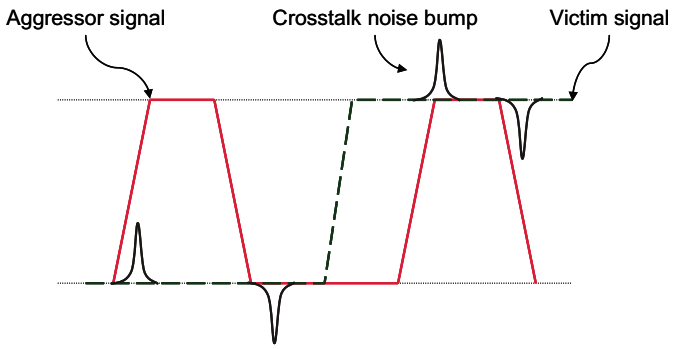


FIGURE 1.2: Crosstalk noise types

It can be observed in Figure 1.2 that if the victim is silent at logic 0 and the aggressor is switching from logic 0 to logic 1, there will be 'low noise'. If the victim is silent at logic 0 and the aggressor is switching from logic 1 to logic 0, there will be a 'low undershoot'. Similarly, when the victim is silent at logic 1, we can observe 'high noise' and 'high overshoot' depending on the switching state of the aggressor.

Crosstalk delay

If the victim is switching in the vicinity of the switching time of its aggressors, there can be a delay or speed-up of the victim signal depending on the direction of its switching relative to that of its aggressors. The difference in delay with and without crosstalk is called *crosstalk delay*.

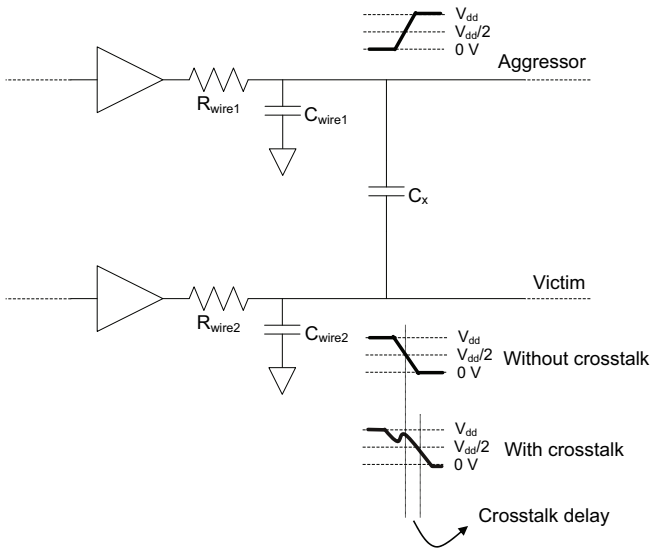


FIGURE 1.3: Crosstalk induced signal delay

Consider the example in Figure 1.3, where the victim is switching from logic 1 to logic 0 while the aggressor is switching from logic 0 to logic 1 and C_x is the parasitic coupling capacitance between the two nets. A delay in the victim signal due to crosstalk can be observed.

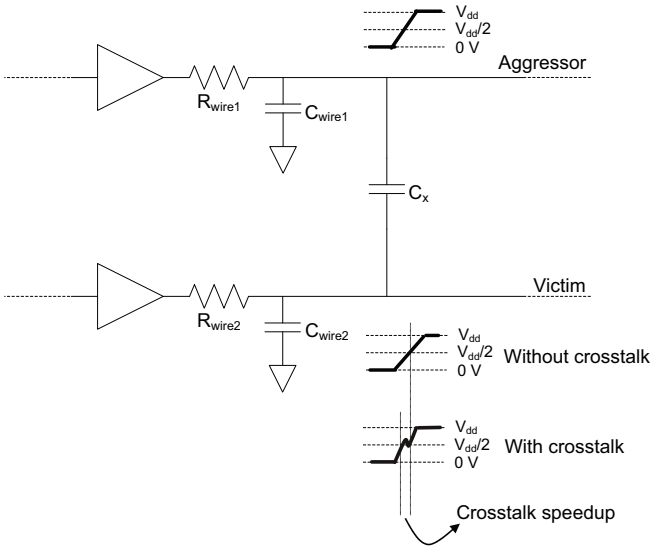


FIGURE 1.4: Crosstalk induced signal speed-up

Similarly, an example of signal speed-up due to crosstalk is depicted in Figure 1.4, where both the victim and its aggressor are switching from logic 1 to logic 0 leading to a speed-up in the signal, as shown in the figure. Even though this delay is negative, it is, in general, called *crosstalk delay*. This delay (or speed-up) propagates through the successive cells on the path and can result in timing violations.

In literature, the terms delay noise and glitch noise are often used for crosstalk delay and crosstalk noise respectively. To avoid any confusion, the terms *crosstalk delay* and *crosstalk noise* are used throughout this thesis for delay due to crosstalk and noise glitches due to crosstalk, respectively, unless specifically mentioned.

1.2 False noise

Conventional crosstalk models often consider all the potential aggressors of any given victim net or path as valid. This unrealistic worst–case crosstalk model leads to an overly conservative and pessimistic estimation of the circuit crosstalk. The amount of overestimated crosstalk (noise, delay or slew change) is called *false noise*.

The source of overall pessimism in crosstalk analysis stems from the independent accounting of each aggressor and victim net coupling. Timing and logic correlations which can render certain switching scenarios impossible are simply ignored. Industrial strength crosstalk analysis tools try to avoid the most obvious blunders by considering the simple logic correlations arising from single inverter and buffer cells. Yet this is far from being sufficient as is apparent from [33, 35, 32, 1, 50, 64, 52, 53, 9]. What needs to be done is to find the aggressor set that has a maximum crosstalk impact on the victim consisting only of those aggressors that can logically and temporally induce crosstalk simultaneously.

To demonstrate the idea of false noise, we consider the circuit in Figure 1.5 consisting of a victim V and three aggressors a_1, a_2 and a_3. In case of V switching from logic 0 to 1, the aggressors will increase the victim's delay, if they switch from logic 1 to 0 in the vicinity of V's switching time. Conventional crosstalk analysis algorithms consider that the worst–case would occur with all the three aggressors switching in the same direction at the same time without considering whether this switching scenario is possible at all. However, a brief check of the circuit rules out this scenario as not all aggressors can simultaneously assume logic 1 or logic 0, which implies that they cannot simultaneously switch in the same direction. So, a constraint should be laid pruning such unrealistic scenarios for crosstalk analysis.

False noise not only distorts the crosstalk analysis of the affected net, but also that of subsequent circuit elements. The impact of this error propagation depends on the crosstalk effect considered. While crosstalk noise can be attenuated by subsequent cells, crosstalk delay never vanishes but sums up in the overall path

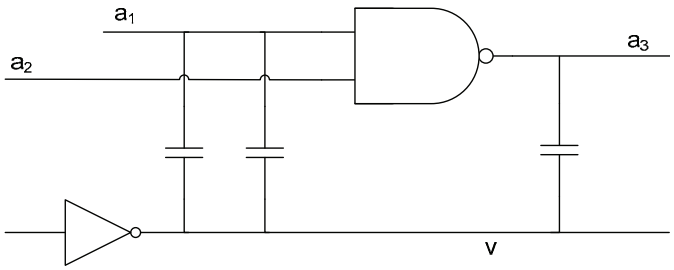


FIGURE 1.5: False noise example with impossible aggressor combination $\mathbf{a_1}$, $\mathbf{a_2}$ and $\mathbf{a_3}$ of victim $\mathbf{V}$.

delay and slack [33]. False calculated slew change due to crosstalk even distorts the analysis of subsequent cells in the path and, finally, the timing check calculation.

1.3 Goals of the thesis

The motivation for this work stemmed up from a successful implementation, testing and improvement of the false noise analysis methods using logic correlations proposed in [33, 35], during the Masters research work done in [48]. The following are the major goals of this PhD research:

1. *Development of speed-improvement algorithms:* A major drawback of several state-of-the-art techniques for false noise analysis is their speed. As a result, false noise analysis is performed as a post-processing technique that is run only on a small set of critical nets or paths [33, 35, 64, 52]. To change this scenario, the second goal of this research is to develop new algorithms that improve the speed of the false noise analysis process.

2. *Development of conservative false noise analysis techniques that can handle glitches:* Although the false noise problem can be greatly simplified by making the zero-delay assumption, as done in [33, 34, 35], such a solution is not acceptable as it does not consider glitches and hence not conservative. The third goal of this research is to circumvent this scenario by the development of new methods to that are conservative, and at the same time effective and efficient.

1.4 Organization of the thesis

This rest of this thesis is organized as follows:

Chapter 2 discusses various topics related to the background of the false noise problem like circuit delay models, deduction of fanin cone, the Boolean satisfiability (SAT) problem etc.

Chapter 3 discusses the state-of-the-art techniques for false noise analysis.and analyzes them.

Chapter 4 discusses about the complexity of the false noise problem and proposes two novel techniques called Simple Aggressor Ordering and Adaptive Bounding for improving its speed.

Chapter 5 discusses about the optimism arising from the zero-delay model and explain a study conducted on the generation and propagation of glitches in standard cells. It briefly discusses about the integration of logic and timing to perform a conservative false noise analysis.

Chapter 6 proposes and explains in detail a novel technique called Timing Arc Based Logic Analysis (TABLA) to perform a conservative false noise analysis. It also explains about the dynamic simulation framework used to verify the assumptions made by TABLA to perform a conservative false noise analysis.

Chapter 7 gives the results of the proposed techniques and analyzes them.

Chapter 8 concludes the thesis with a summary and outlook.

Chapter 2

Review of Related Models and Techniques

This chapter discusses various background related topics of the false noise problem. Section 2.1 discusses about various circuit delay models that can be used for the false noise analysis. Section 2.3 explains how to figure out the coupled fanin cone of any given crosstalk configuration. Section 2.4 explains about the modeling of crosstalk to measure its impact on a victim due to a specified set of its aggressors. Section 2.5 explains about the modularization of the false noise analysis problem. The techniques to analyze false noise used in this thesis are heavily based on the Boolean satisfiability (SAT) solvers. The SAT problem reviewed in Section 2.6.

2.1 Circuit delay models

Circuit delay models can be classified into four types: zero-delay, fixed delay, min-max delay and unbounded delay models. These models are discussed briefly in this section. For a more detailed discussion, the reader is referred to [41].

Zero-delay model

In this model, all the combinatoric cells and interconnects of the circuit are assumed to have a zero delay. The outputs of any gate of the circuit are assumed to be available immediately after the inputs are applied. As a result, any signal in the circuit can switch at most once. This assumption simplifies the analysis and

makes it easy to convert the problem into a combinational optimization problem. It is valid for domino circuits and other glitch free circuits designed using special gate sizing techniques.

Fixed delay model

In this model, the delay on any timing arc of a cell is considered to be constant and not dependent on any factors like input slew, output capacitance etc. Delays on the interconnects are usually considered as zero. Its delay could be incorporated by adding buffer of equal delay. This model is intermediate to the zero-delay model and min-max delay model in terms of speed, accuracy and conservatism. This is a good alternative to the min-max delay model, especially when speed is of utmost importance and the results are not much deviant from the min-max delay model for the problem in hand. This model partially considers the problem with glitches discussed in Chapter 5.

Min-max delay model

In this model, delays in the form of min-max windows are considered. This leads to a more conservative analysis as it is the most general model which does not give scope to any optimism. Depending on the problem, techniques using this model can be very costly in terms of CPU time.

Unbounded delay model

The unbounded delay model [43] assumes an indefinite delay on the cells and interconnects of the circuit. In other words, this model assumes infinite timing windows. As no explicit information on timing needs to be handled along with logic, the modeling of logic gets very simple. On the other hand, because of the overly conservative assumption of infinite timing windows, this model is less restrictive. Hence, it cannot be very effective in terms of false noise reduction, unless it is combined with a method that considers more accurate timing windows.

2.2 Switching window vs. arrival time window

An arrival time window of a port for a specific type of transition (rise or fall) indicates the earliest and the latest time points at which the signal on it can cross the $V_{dd}/2$ voltage level. A switching window is the arrival time window extended on both sides by 50% of maximum time taken by the specified transition on this port (i.e., rise or fall).

2.3 Coupled fan–in cone

For the analysis of a victim net, it is not necessary to consider the logic constraints of the entire circuit. It is sufficient to consider that part of the circuit that can influence the victim and the aggressor signals. This part of the circuit is obtained by finding the fan–in cones of the victim and its aggressors, and merging them [64]. This selection is called the coupled fan–in cone of the victim. Coupled fan–in cones are often very large for victim nets near the primary outputs or the inputs of sequential cells.

2.4 Modeling crosstalk

The problem of measuring the impact of crosstalk on a victim with a specified set of its aggressors has been widely studied. The range of its solutions varies from a complex non-linear optimization approach to a linear aggressor bump alignment method. Although the non-linear optimization approach is highly accurate it is highly time consuming and hence is not used in any commercial tools. The linear aggressor bump alignment method on the other hand has been proven to be of a reasonable accuracy.

Although a highly accurate measurement of crosstalk effects is essential for the final noise or delay estimation, it is sufficient to have a good analytical model for an abstract level analysis like the false noise analysis. In what follows, the approximate crosstalk estimation technique using aggressor alignment and exact crosstalk estimation techniques using optimization algorithms are discussed briefly.

2.4.1 Using aggressor alignment

This is a heuristic method to estimate crosstalk noise or delay under the assumption of a linear crosstalk model by the application of superposition principle. In this method, the crosstalk bumps due to the individual aggressors on a silent victim are initially obtained using SPICE simulations. The overall effect of crosstalk due to all the aggressors switching simultaneously is then measured by adjusting the switching times of the aggressors in such a way that all the crosstalk bumps due to each of the individual aggressors are aligned and running another SPICE simulation.

2.4.2 Using optimization techniques

The worst impact of crosstalk on a victim can be estimated using optimization techniques to find the worst-case alignment of aggressors[1]. Such a procedure to estimate worst-case crosstalk is very accurate, although it is time consuming. In this thesis, for the purpose of accurate crosstalk analysis on certain cases explained in Chapter 7, a limited-memory quasi-Newton optimization method called L–BFGS-B [74], which is available on our proprietary STA tool is used.

2.5 Modularization of false noise analysis

False Noise Analysis, in a broader perspective, can be divided into two modules – the optimization and constraint satisfiability check modules. These modules are independent of each other, as depicted in Figure 2.1. This kind of modularization helps attacking the optimization and constraint satisfiability problems independently in such a way that the improvements in each module are orthogonal, but benefit from each other.

2.5.1 Optimization

The optimization module deals with the combinational optimization of crosstalk to find the maximum realizable configuration of aggressors. This can be done by

[1]The word optimization can either mean minimization or maximization depending on the situation. In this case, by optimization, the maximization of the worst-case crosstalk effects is meant.

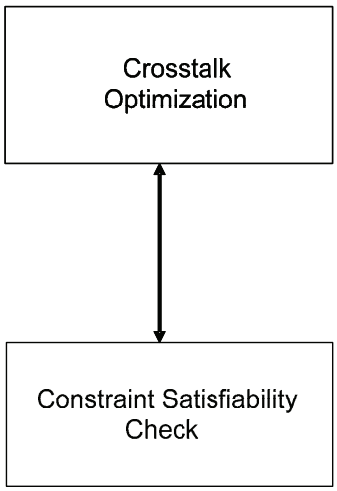


FIGURE 2.1: Modules of the false noise problem

exploring a binary decision tree to implicitly or explicitly check all the possible combinations of aggressors and find the optimal one.

Optimizer uses the services of the constraint satisfiability check module to find whether a potential solution is valid or not. Its efficiency depends on how effectively it uses the problem specific knowledge and feedback from the constraint satisfiability checker to minimize the search space. Various techniques for the efficient optimization of false noise are discussed in detail in Chapter 4.

2.5.2 Constraint satisfiability check

This module deals with the verification of the inherent constraints of the circuit to check if a given crosstalk configuration (queried by the optimization module) is valid. The process of checking the validity of constraints depends on the type of circuit delay model considered.

For the zero-delay model, this can be done by deducing the logic constraints of the circuit, feeding them into a SAT solver and then querying it if a particular configuration is valid. However, for more complicated models, an analysis of both the temporal and logical behavior of the circuit must be done. While circuit unrolling discussed in Section 2.7 is a straight forward means to achieve this, it is often very inefficient, especially for applications like false noise analysis and false path analysis, where a large number of time-frames have to be used. In this thesis, a more sophisticated approach to achieve this is proposed in Chapter 6.

2.6 The Boolean Satisfiability (SAT) problem

The SAT problem has its significance in false noise analysis because the constraint satisfiability problem (discussed in detail in section 2.5.2) can be formulated as a SAT problem. As the constraint satisfiability check occurs in an inner loop of the optimization process, it is essential to have fast algorithms to do this efficiently. This can be done by using a state–of–the–art SAT solver.

Boolean Satisfiability (SAT) is the problem of finding if an assignment of the variables of a given boolean formula that make the formula evaluate to *true* exists. A formula for which such an assignment exists is said to be satisfiable (SAT), or unsatisfiable (UNSAT) otherwise.

Definition 1: Let $x_1, x_2 \cdots x_n$ be n boolean variables and $f(x_1, x_2 \cdots x_n)$ be a boolean formula. The formula $f(x_1, x_2 \cdots x_n)$ is said to be satisfiable if there exists a boolean vector X_{SAT} which is a mapping from $\{x_1, x_2 \cdots x_n\}$ to $\{0, 1\}^n$ such that $f(X_{SAT}) = true$.

For an unsatisfiable case, the formula evaluates identically to *false* for all possible variable assignments. The f problem is the first known NP-complete problem, as proved by Stephen Cook in 1971 [18].

SAT solvers have evolved continuously for a period of more than 4 decades and have seen great advances in the recent years. Many SAT solvers like zChaff, MiniSat etc. are publicly available [25, 46].

In what follows, various concepts related to SAT solvers that stand behind their efficiency are briefly discussed. A detailed discussion of these is out of the scope of this thesis. The fundamentals of SAT solvers are explained in detail in [71] and [73].

2.6.1 The basic DPLL algorithm

The DPLL or Davis-Putnam-Logemann-Loveland algorithm [21, 20] is an algorithm that forms the basis for today's most efficient complete SAT solvers. It is a back-tracking algorithm that runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if this simplified formula is satisfiable. If so, the original formula is satisfiable. Otherwise, the literal is assigned the opposite truth value and the same procedure is repeated.

The DPLL algorithm enhances the back-tracking algorithm by the use of two techniques:

- ***Unit propagation:*** If there is a unit clause, i.e., a clause with a single literal, then it has only a single possible assignment that makes it satisfiable. Hence, no choice is necessary in this case, and in practice, this helps in avoiding a large part of the search space.
- ***Pure literal elimination:*** A propositional variable occurring with only one polarity in the formula is called a pure variable. These variables can always be assigned in a way that makes all the clauses containing it satisfiable. This can help in reducing the size of the formula significantly.

2.6.2 Boolean constraint propagation

Boolean constraint propagation (BCP) is a very important mechanism of today's SAT solvers. The functionality of a BCP engine is to detect unit clauses and conflict clauses after a variable assignment. There are various methods to implement the BCP mechanism. A few of them are counter based mechanism, using head/tail lists and 2-literal watching. The 2-literal watching mechanism has become very popular and is used in most of the efficient SAT solvers today.

2.6.3 Conflict analysis and learning

The most effective feature of SAT solvers that make them very efficient is their conflict analysis and learning mechanism. When a conflicting clause is encountered during the BCP mechanism, the solver needs to backtrack and undo all the decisions that led to this conflict. Conflict analysis is a procedure to identify the reason for a conflict and add knowledge to the SAT solver that forbids all the solutions that lead to the same conflict.

Techniques like non-chronological backtracking are very powerful in the deduction of very efficient conflict clauses or learnt clauses. These clauses are redundant to the satisfiability problem itself, however they often help the SAT solver in pruning significant amount of search space.

2.6.4 Other techniques

State-of-the-art SAT solvers use a number of heuristics to improve their efficiency. Some of these are preprocessing, random restart, activity heuristics, variable ordering etc. A discussion on these techniques is out of the scope of this thesis. For a detailed discussion on these topics, an interested is reader is referred to [59, 46, 26, 24].

2.7 Circuit unrolling technique

Circuit unrolling is a popular technique used to perform temporal logic analysis of circuits. It transforms a sequential problem that involves both timing and logic into a combinational satisfiability problem. This process involves in transforming the circuit to be analyzed into a larger circuit by making a copy of the original circuit for each time frame, enabling it to be analyzed as a circuit with zero-delay on its gates and interconnects. The variables of the unrolled circuit are attributed with the time stamps to which they belong. This process increases the size of the circuit by the number of time slices. This technique is often used in formal verification [6].

The circuit unrolling technique can be applied to perform temporal logic analysis to reduce false noise or to identify false paths. For problems of larger size, this approach leads to a potential exponential blow-up of variables and clauses.

This technique is often inefficient to use this technique because of the following reasons:

1. After circuit unrolling, the number of variables in the temporal logic analysis problem becomes the number of variables in the original circuit times the number of time slices i.e., if the original circuit has n variables and if m time slices are used, then the number of variables in the unrolled circuit becomes $n \times m$. Since the temporal logic analysis problem is NP-hard, the increase in number of variables by a large factor badly effects the efficiency of the solution.

2. The process of circuit unrolling converts the temporal logic analysis problem into a large combinational problem. However, this transformation is done

without the knowledge of the circuit and timing. Hence, it effectively becomes a brute-force approach that becomes highly inefficient for problems of large size.

Chapter 3

Related Work

In this chapter, the state-of-the-art techniques for false noise analysis.are discussed and analyzed. Section 3.1 discusses the Simple Logic Implication based approach to reduce false noise. Section 3.2 discusses the logic constraints based approach to reduce crosstalk delay noise pessimism. Section 3.4 explains a back propagation approach to perform temporal logic analysis. Section 3.5 discusses about the timing slicing based approach that uses circuit unrolling to solve the temporal logic analysis problem. Section **??** explains about an event propagation and storing mechanism that can answer queries related to the circuit behavior. The Tendency Graph Approach is reviewed in 3.3. Finally, Section 3.6 discusses about the shortcomings of these approaches and explains how this thesis solves these issues.

3.1 SLI based approach

This method [32] aims at finding a maximal aggressor set called Maximum Realizable Aggressor Set (MRAS) that produces maximum realizable crosstalk on a victim net without violating the inherent logical properties of the circuit. The logic properties of the circuit are represented in the form of simple logic implications (SLI), which use the notation

$$(a = \alpha) \rightarrow (b = \beta) \tag{3.1.1}$$

where $\alpha, \beta \in \{0, 1\}$. This means that the information that $a = \alpha$ logically implies that $b = \beta$.

This method, described in [32], initially formulates the problem using zero-delay logic implications which are valid only when the circuit is stationary, i.e., before or after all the transitions occur. Analysis using zero-delay logic implications is conservative only for glitch free circuits [32]. Methods to extend the approach to use timed logic constraints are proposed at the end.

To obtain all the SLIs of the circuit, first, the logic implications of all the simple gates of the circuit are derived and then propagated using the methods *direct implication* and *lateral propagation*. It is proved that if a circuit consists of AND, OR and INVERTER gates, the logic function of the circuit is completely specified by the full set of SLIs of the circuit. This means that consideration of all the SLIs of the circuit implies consideration of all logic properties for the crosstalk noise analysis.

Once all the SLIs are obtained, they are translated into a constraint graph, which is an undirected graph with all potential aggressors to the victim net under consideration as vertices. An edge is drawn between two vertices if they cannot switch simultaneously to produce the desired crosstalk effect. Each of the vertices is assigned a weight (≥ 0), which represents the strength of the aggressor, i.e., a proportion to the amount of crosstalk that the aggressor can produce. The problem is now reduced to the problem of finding the maximum weight independent set (MWIS) of vertices from the constraint graph, which is NP-complete.

As the SLIs are based on the zero-delay assumption, this method cannot handle glitches in the circuit and hence is not conservative. In order to circumvent this, [32] proposes timed SLIs, which consider also the circuit timing in order to catch glitches. However, the timed SLIs become inefficient and ineffective for medium and large industrial designs.

The method assumes linear crosstalk models, i.e., the amount of crosstalk on a victim net due to a set of aggressors is assumed to be proportional to the sum of the individual amounts of crosstalk due to each aggressor. Influence of slews of the victim and aggressor lines is not considered. SLIs can relate only two signals at a time. It would often be advantageous to obtain the relation among more than two. This can be done by using logic correlations as described in [33]. This method is explained in the next section.

3.2 Logic constraints based approach

3.2.1 Introduction

This method [33] also aims at finding a maximal aggressor set called maximum realizable aggressor set (MRAS) that produces maximum realizable crosstalk delay on a victim path (not just on a victim net) without violating the inherent logic constraints of the circuit. It considers only false noise in worst-case crosstalk delay estimation. It assumes linear delay models, i.e., the crosstalk delay due to a set of aggressors simultaneously acting on the victim is calculated as the sum of the crosstalk delays due to each of the aggressors individually. The overall delay on a victim path is assumed to be the sum of the individual delays on each victim net. Different aspects of this method are explained in various subsections below.

3.2.2 Generation of logic constraints

Logic constraints are impossible combinations of signals. They indicate whether a particular combination of signals is possible or not. Mathematically, a logic constraint is a boolean term with one or more literals, either inverted or uninverted, in a conjunctive form. If the signal values are such that the term gives a logic 1, then the signal combination is not possible.

The first step in this approach is to find all the logic correlations of the circuit signals. As it is cumbersome to find the set of all the allowed combinations, a set of prohibited combinations is found. All the prohibited combinations (logic constraints) are specified in the form of a boolean equation, in such a way that a set of signals satisfying this equation is prohibited. Extraction of logic constraints can be done from either a gate level or a transistor level circuit. First, the logic constraints of the signals of a gate or transistor are found and these logic constraints are forward and backward propagated to find other logic constraints of the circuit using resolution method [54] described in [35].

3.2.3 Estimation of worst possible crosstalk

Once all the logic constraints of the circuit are found, the problem is reduced to finding a maximum weight independent set (MWIS) of a hypergraph with each

logic constraint as a hyperedge. This problem is then formulated as an optimization problem which tries to maximize the overall delay on the victim path which is given by the equation below:

$$\triangle D_{max} = max \sum_{a_{i,j} \in A_R \subseteq A} \triangle D_{i,j} \tag{3.2.1}$$

where $\triangle D_{max}$ is the maximum possible delay on the victim path, $\triangle D_{i,j}$ is the delay on i^{th} victim net due to its j^{th} aggressor, A_R is the maximum realizable aggressor set and A is the set of all aggressors of the whole path. It can be noted that this formulation is possible due to the assumption that the overall delay on a victim net due to all the aggressors acting simultaneously is the sum of the delay on it due to each aggressor acting individually. The branch and bound algorithm, described in 4.1, is used to find an exact solution to this problem in [33]. This algorithm is a depth-first search algorithm which drops unnecessary branches when there is no scope of finding a better solution in that direction based on a fail criterion. This algorithm finds the exact solution, but has an exponential complexity in worst-case.

3.2.4 Approximate heuristics for speed improvement

Due to its exponential complexity, using the exact solution approach becomes difficult as the number of aggressors increases. For these cases, certain heuristic approaches have been proposed in [33], which are based on the divide and conquer principle. These heuristics divide the set of aggressors into subsets and solve the problem for each of these subsets independently. However, if the aggressors of these subsets have any logic constraints in common, it leads to an over-approximation of crosstalk. Additional heuristics propose a method to consider the common logic constraints of the subsets by converting them into simple logic implications (SLIs) which leads to results which are a little more accurate.

3.3 Tendency Graph Approach (TGA)

TGA, proposed in [50] is based on the branching heuristic 'fail first' which states: "To succeed, try first where you are most likely to fail". Fail first suggests to order the elements fed to the branch and bound algorithm in a way that unnecessary

branches are dropped out at an earlier stage, thus saving unnecessary exploration of search space.

TGA increases the speed of branch and bound algorithm by trying to minimize the effort required to determine that an insoluble subtree is indeed insoluble. However, since branch and bound has an exponential worst-case complexity, for problems with a very large number of elements (typically greater than 50), the algorithm may not be feasible even after applying the heuristic. In such cases, additional, yet approximate solutions which aim at partitioning the set of aggressors into smaller sub-sets are discussed in [33].

TGA is designed to improve the speed of the logic constraints approach proposed in [33]. As the approach that used SAT solvers handles constraints differently, TGA is not applicable for such cases. Instead, the AB and SAO approaches proposed and discussed in this thesis can be used.

3.4 Back propagation approach

3.4.1 Introduction

This method, described in [9, 52, 53], uses a four-variable Boolean logic to characterize signal transitions in a time interval, and formulates Boolean satisfiability between aggressors and a victim under the min-max delay model for gates. It performs a temporal logic analysis to consider both logic and functionality of the circuit in the process of elimination of false noise. It constructs a timed-SAT formula on the fanin cone of a path/endpoint based on the required transition analysis.

3.4.2 Functionality and timing

In order to perform a temporal logic analysis, this method introduces four timed binary variables y_T^0, y_T^1, y_T^r, y_T^f for each wire y in time interval $T = [t_1, t_2]$ to indicate whether the wire is low, high, rising or falling, respectively at some time in $[t_1, t_2]$. The boundary between logic values 0 and 1 is defined by $V_{dd}/2$ voltage level. y_T^r (y_T^f) is true if there exists a pair $t_1, t_2 \in T : t_1 < t_2$ such that $y_{t_1}^0$ and $y_{t_2}^1$ ($y_{t_1}^1$ and $y_{t_2}^0$) are true. The nature of this specification implies that $y_T^r \Rightarrow y_T^0 y_T^1$, $y_T^f \Rightarrow y_T^0 y_T^1$and that $y_T^r y_T^f$ is not necessarily 0. In other words, for a net to rise

or fall in time interval T, both the variables y^0_T and y^1_T must be true. Also, it is possible that a net can both rise and fall in the same time interval, implying that the glitches are caught using this model.

The signal values and time intervals are propagated backwards towards the inputs[1]. This is done with the use of propagation conditions derived from the functionality of the cells. The propagation conditions for a two input AND gate $y = ab$ are given below:

$$y^1_T = a^1_{T-(D^r_a \cup D^f_a)} b^1_{T-(D^r_b \cup D^f_b)} \quad (3.4.1)$$

$$y^0_T = a^0_{T-(D^r_a \cup D^f_a)} + b^0_{T-(D^r_b \cup D^f_b)} \quad (3.4.2)$$

$$y^r_T = a^r_{T-D^r_a} b^1_{T-(D^r_b \cup D^f_b)} + b^r_{T-D^r_b} a^1_{T-(D^r_a \cup D^f_a)} \quad (3.4.3)$$

$$y^f_T = a^f_{T-D^f_a} b^1_{T-(D^r_b \cup D^f_b)} + b^f_{T-D^f_b} a^1_{T-(D^r_a \cup D^f_a)} \quad (3.4.4)$$

Once all the transition and steady value intervals are propagated to inputs, they are pruned by taking an intersection with input arrival windows because a circuit is insensitive to input changes outside arrival windows. The overlap of back-propagated and arrival time window constitutes the active portion of the window for the circuit.

3.4.3 Modeling noise in logic

The purpose of the worst case noise analysis is to determine the maximum delay/speedup possible at net transitions because of crosstalk. After the noise analysis is done, the switching windows of every net are padded by the amount of noise that can occur. When constructing a timed-Boolean formula for a circuit behavior using logic of intervals and backward window propagation, the noise delays are considered in two ways – worst case noise delays or refined noise delays.

Worst case noise delays

This is a straight-forward method in which a noise impact is taken into account by including worst-case noise values into pin-to-output delays. This is efficient

[1]Unless specified otherwise, throughout this thesis, the word inputs is used to mention primary inputs and sequential cell outputs. This word is used in the sense that they are the inputs of the fan-in-cone under consideration

from the standpoint of complexity but does not provide any room for an accurate analysis of noise sources.

Refined noise delays

In this case, the noise possible on the nets of interest is explicitly considered. The is done by discretizing the maximum possible noise with a predefined step $\triangle$ and deriving Boolean conditions for the feasibility of every interval. The maximum noise that can occur is determined by the maximum discrete noise value for which the corresponding SAT clause is satisfiable.

3.4.4 Input conditions

When required time intervals are propagated backward from the analyzed nets, multiple time intervals might be obtained at an input, introducing many timed-Boolean variables. It is assumed that for every cycle, an input can switch at most once. Therefore, the timed boolean variables for inputs must satisfy the following conditions:

- **Persistence:** Variables at adjacent points in time of the same net must have the same logic value unless there is a transition event between the two points.
- **Monotonicity:** There is at most one transition event.

3.4.5 Pruning the size of the satisfiability formulation

The side of the circuit that needs to be processed for the false noise analysis of a given path or cone L is defined by the combinational logic in the coupled fan-in of all the nets from L together with their aggressors. Moving backward, propagated windows are multiplied because of discretizing the delay and due to reconvergent paths. This results in exponential number of switching windows and timed Boolean variables propagated. To keep the size of the SAT formula under control two techniques are used:

- **Temporal pruning:** If a backward propagated window for required times is outside the bounds of switching windows for arrival times, then the propagated window is dropped.
- **Interval merging:** This technique selectively merges a pair of disjoint switching windows T_a and T_b to create a new window $T_{new} = T_a \cup T_b$. A predefined threshold on the maximum number of windows at the output of a single net is assumed. Interval merging is applied to keep the number of timing windows not exceed the threshold. As this technique introduces an over-approximation in the overall analysis, it is conservative.

3.4.6 Delay noise analysis

For a given path (or cone of logic) L, finding its true delay reduces to the derivation of a SAT formula for L and every interval for delay or speedup at L, and then finding the maximum delay or speedup for which the formula is satisfiable. The analysis takes as an input a set of noisy switching windows sw^n for all nets together with their noisy pin-to-output delays, which are available as a result of conventional worst-case noise analysis. The procedure starts from a terminal net g_n of l and recursively constructs SAT formula over the input variables.

For efficiency, the refined noise delay propagation is applied only to the nets from L. For all the side-inputs, the worst case noise delay propagation is considered.

3.5 Time-sliced logic approach

3.5.1 Introduction

This method, described in [64], uses a timing analysis technique where circuit functionality, delay and crosstalk are simultaneously considered using time-sliced Boolean logic. The timing analysis framework used in this method tracks timing and logic assumptions along every path to ensure that they are consistent. It does so by creating a Boolean equation at each time point that must be feasible for the output to transition at that given time. The worst case timing on the critical path endpoint is calculated to be the minimum and maximum time slices where the timed Boolean logic on its coupled fanin cone is satisfiable.

While this framework is accurate if fine time steps are used, it also generates a very large number of equations to evaluate.

3.5.2 Timing analysis framework

The method employs discretized timing windows to break down a timing window into bins to keep track of the possibility of one or more switching events occurring within each time bin. This reduces pessimism in coupling analysis due to the decrease in aggressor and victim timing overlap possibilities over the original monolithic timing window.

As part of the logic framework, all side-input states that sensitize a delay arc are exhaustively enumerated and stored. Using a two-input AND gate as an example, the following logic equations can be written as

$$Y_R = A_R(B_1 + B_R) + B_R(A_1 + A_R) \tag{3.5.1}$$

$$Y_F = A_F(B_1 + B_F) + B_F(A_1 + A_F) \tag{3.5.2}$$

where A and B are inputs and Y is the output. The 0,1, R and F subscripts refer to the low, high, rising and falling states in a four-variable Boolean logic. As shown in the above equations, all input combinations in four-variable logic are considered. For an N-input gate, there are 4^N possible input combinations.

The logic conditions required for a gate output to be either steady-state low or steady-state high are calculated from the gate logic function using the same four-variable logic representation. All combinations of 0, 1, R and F on all inputs are exhaustively enumerated to form the steady-state logic function for the output. Using a two-input AND gate example as above, the equations for the output low and high conditions can be written as

$$Y_0 = A_0 + B_0 \tag{3.5.3}$$

$$Y_1 = A_1 \cdot B_1 \tag{3.5.4}$$

It should be noted that A_0 is not equivalent to $\overline{A_1}$, as $\overline{A_1}$ implies A_R, A_F or A_0.

Each time slice is represented by a logic variable for each of the rise (R), fall (F), steady-state low (0) and steady-state high (1) states. These states are not mutually exclusive, as within a logic time slice all four states may exist in sequence.

A gate output's time slice logic conditions are represented by a gate input's time slice logic variables that are a gate delay d earlier in time. All possible arcs must be enumerated. From this, the time slice logic tables are built for all nets in the coupled fan-in cone. Using this, the satisfiability at the output cone is verified.

3.5.3 Coarse logic gridding

With small grid sizes of typically 5ps, the number of SAT clauses become too large,resulting in extremely slow run times and memory overflows. The runtime of this method is improved by merging adjacent time slice logic conditions into coarser logic grids while maintaining the timing grid at 5ps to preserve timing accuracy. This leads to some additional pessimism as the switching events within a coarse logic grid are not logically distinguishable from one another.

Three additional techniques are employed in this method to control the number of logic equations. These are listed below:

1. The number of logic grids per net are limited by automatically scaling the logic grid size based on the timing window width.
2. Entire logic subtrees with significant timing slacks are truncated.
3. Logic trace back for weak aggressors is avoided.

3.6 Conclusions

The SLI based approach discussed in Section 3.1 uses simple logic implications of the circuit to solve the false noise problem. The logic constraints based approach 3.2 considers logic correlations of the circuit and performs a resolution principle [54] based false noise analysis. The Tendency Graph Approach explained in Section 3.3 provided a speed improvement of this procedure. These approaches suffer from loss of accuracy or degradation of performance because of a restricted application or an uncontrolled application, respectively, of the resolution principle. In this thesis, this problem is solved using a state-of-the-art SAT solver, which can handle the underlying Boolean satisfiability problem very efficiently as explained in Section 2.6. Two novel techniques called SAO and AB proposed in this thesis provide a drastic improvement of speed of this SAT-based false noise analysis.

The back propagation and the time sliced logic approaches discussed in Section 3.4 and Section 3.5, respectively, use the min-max circuit delay model to perform a conservative false noise analysis, which can handle glitches. For this purpose, an analysis of timing, logic and their interdependence must be performed. Such an analysis is often called temporal logic analysis. The problem of temporal logic analysis to verify the validity of a crosstalk scenario can be described as a satisfiability problem. This problem is solved in [14, 52, 64] by converting it into a Boolean SAT instance, which leads to a potential exponential blow-up of variables and clauses, and then solving it using a state-of-the-art SAT solver [25, 46]. These approaches often suffer from degraded performance due to the loss of high level semantics of the underlying timing problem and large SAT instances.

A novel technique called Timing Arc Based Logic Analysis (TABLA) proposed in this thesis efficiently handles the temporal logic analysis problem by using dedicated solvers for timing and logic, contrary to these approaches. To account for the interdependence of timing and logic, these solvers are tightly coupled by the use of common variables and data structures, which is possible because of the suitability of timing arcs to handle both logic and timing. As the higher level timing information is handled without any transformation into Boolean logic, this method maintains utmost accuracy without trading it off for runtime.

Chapter 4

Techniques for Efficient False Noise Analysis

As the optimization of false noise requires the exploration of a binary search tree, it has a complexity of $O(2^n)$, where n is the number of aggressors, in the worst case. This necessitates the application of heuristics to improve its efficiency in an average case. In this chapter, two novel techniques called Simple Aggressor Ordering (SAO) and Adaptive Bounding (AB) that improve the speed of the optimization process without trading off accuracy are proposed.

The rest of the chapter is organized as follows. Section 4.1 describes the basic branch and bound algorithm. Section 4.4.2 and Section 4.3 explain the SAO and AB techniques designed to improve the speed of false noise analysis. The AB technique is designed to work with the zero-delay model. However, the same concept can be extended to work with TABLA discussed in Chapter 6.

4.1 Branch and bound – the core algorithm

Branch and bound is an optimization method, mostly used for non–convex NP–hard problems. It explores the solution search space by pruning off non–promising or futile regions as it proceeds, based on a bounding criterion and constraint satisfiability. In the case of false noise analysis, the branch and bound explores a binary search tree of aggressors where each node of the tree indicates a specific selection of the aggressors already traversed. The validity of any branching, or, in other words, the corresponding crosstalk configuration, is determined using

the constraint satisfiability check module[1] (see Section 2.5). The currently best solution is kept track of during the entire process.

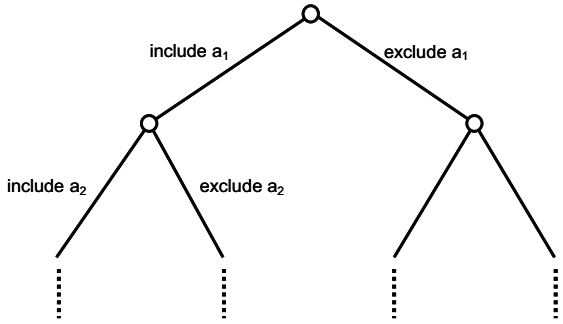


FIGURE 4.1: A sample branch and bound decision tree.

Figure 4.1 depicts an example of a binary tree explored by the branch and bound algorithm. The tree is traversed recursively in a depth–first fashion. During the traversal, at each node, all further possible combinations (with the decisions taken at the higher levels fixed) which include and exclude the current node are explored by recursive calls to its sub–nodes. However, this search is implicit in some cases where it can be proven that further exploration is not necessary. In other words, before any further recursive calls to the branch and bound method, it is verified using the bounding criterion that a further exploration of the sub–tree is useful. The sub–tree of any node is explored further only if it can potentially yield a better solution than the current best.

Pruning can also be done based on the validity of configurations. If at any stage, a decision either to include or to exclude an aggressor is identified to be invalid, the sub–tree corresponding to this decision branch is pruned.

4.2 Using SAT solver for false noise analysis

As discussed in Chapter 3, [33, 35] are the seminal papers that discuss about using logic correlations to perform false noise analysis using resolution principle [54]. The drawbacks of these methods, discussed in Chapter 3, can be avoided by using a state-of-the-art SAT solver [25, 46] to solve this problem. Integration of a SAT solver in to the false noise analysis flow is a simple process that involves in the generation of logic constraint clauses, adding them to the SAT solver and

[1]For the zero-delay model, a constraint satisfiability check module is nothing but the SAT solver to which the logic constraints are fed. For the more complex case of temporal logic solver, TABLA discussed in Chapter 6 plays this role.

then using to SAT solver to check the logical validity of a particular scenario under consideration. It has been observed that this approach is more efficient than the classical logic correlations approach proposed in [33, 35].

4.3 Simple Aggressor Ordering (SAO)

4.3.1 Procedure

The order in which the aggressors are traversed plays a vital role in the amount of solution search space that can be pruned. A tendency based ordering of aggressors is proposed in [50]. This approach is based on the switching tendencies of aggressors which can be calculated from the logic constraints correlating them. As this is a considerable overhead with the use of SAT solvers, a different ordering technique called Simple Aggressor Ordering, which orders the aggressors in the descending order of their strengths is proposed and used in this thesis.

Experiments have shown that this kind of ordering is very much essential not just for speed improvement, but in many cases also for feasibility. If the decisions on stronger aggressors are done in the beginning, many decisions on less significant aggressors can be avoided if their addition does not yield any improvement. The latter can be verified by computation of the bounding criterion.

4.4 Learning in branch and bound

4.4.1 Introduction

In this thesis, the idea of learning in branch and bound, which can aid the Adaptive Bounding strategy discussed in Section 4.4.2 is introduced. Learning in branch and bound can be achieved by using a conflict–driven SAT solver, which is based on learning from conflicts occurring during its search for a satisfiable solution. The idea of conflict–based learning is very effective because it drives the search process in such a way that the conflicts that once occurred do not occur again.

During the constraint satisfiability check, if an aggressor configuration is found to be invalid, the set of conflicting aggressors can be deduced from the conflict information obtained from the SAT solver. This information is stored in a database

in the form of a list of conflicting aggressor sets for any given aggressor. As we see in the next section on Adaptive Bounding, this information can be exploited to calculate very tight upper bounds in the branch and bound process.

4.4.2 Adaptive Bounding (AB)

The power of branch and bound lies in its ability to prune futile search spaces. This can be improved by using a bounding strategy that calculates very tight upper bounds. Adaptive Bounding is such a technique that helps branch and bound to efficiently use knowledge that is already acquired during the process by adaptively taking care that a conflict that already occurred is considered in the estimation of upper bounds. This helps the Branch and Bound to efficiently foresee the necessity to explore an unexplored sub–tree.

The branch and bound algorithm used in the thesis is based on the following properties of which the first two are inspired from [33]. Let $W()$ be the weight function which gives the overall crosstalk induced by a set of aggressors.

Procedure 4.1 ADAPTIVE_BNB: Pseudocode for the Adaptive Branch and Bound algorithm

Input: An aggressor vector ordered using SAO, $A_v = \{a_1, a_2, \cdots, a_n\}$; Index of the aggressor to be processed, i; Current aggressor selection, A_s; Current MRAS, A_r; Weight function $W()$ to find the overall influence of a given aggressor set

Output: Current MRAS, A_r

```
1: Set unprocessed aggressor set A_u = {a_i, a_{i+1}, ..., a_n}
2: if A_u = ∅ return A_r
3: Set A_s' = A_s ∪ {a_i}; Set A_u' = A_u − {a_i}
4: if aggressor configuration A_s' is valid then
5:       W_UB = ADAPTIVE_UPPER_BOUND(A_s', A_u')
6:       if W(A_s') > W(A_r) then A_r = A_s'
7:       if W_UB > W(A_r) then
8:             A_r = ADAPTIVE_BNB(A_v, i + 1, A_s', A_r)
9: else
10:      Get conflicting aggressor set, A_c from SAT solver
11:      Add A_c to conflict database
12: W_UB = ADAPTIVE_UPPER_BOUND(A_s, A_u')
13: if W_UB > W(A_r) then
14:      A_r = ADAPTIVE_BNB(A_v, i + 1, A_s, A_r)
15: return A_r
```

Property 1: Let A be the set of all potential aggressors. If $A_r \subseteq A$ is any set of realizable aggressors and A_R is the maximum realizable aggressor set (MRAS), then $W(A_r) \leq W(A_R)$.

Property 2: Let A be the set of all potential aggressors, and A_1 and A_2 be two disjoint subsets of A, such that $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$. If $A_{r_1} \subset A_1$, and $A_r \subset A$ are realizable aggressor sets such that $W(A_r) > W(A_{r_1} \cup A_2)$, then the maximum realizable aggressor set (MRAS) A_R is not a subset of $A_{r_1} \cup A_2$, i.e., $A_R \not\subset A_{r_1} \cup A_2$.

Property 3: Let A be the set of all potential aggressors and $A_r \subset A$ be a set of realizable aggressors. Assume that $A_c \subset A$ is a set of conflicting aggressors of which aggressor a has the least strength. If $A_s \subset A$ is any selection of aggressors such that $A_c \subset A_s$ and $W(A_r) > W(A_s - \{a\})$, then the maximum realizable aggressor set (MRAS) A_R is not a subset of A_s, i.e., $A_R \not\subset A_s$.

Procedure 4.2 ADAPTIVE_UPPER_BOUND: Pseudocode to find upper bound using the Adaptive Bounding technique

Input: Aggressor set A_s (current aggressor selection); Aggressor set A_u (unprocessed aggressors); Weight function $W()$ to find the overall influence of a given aggressor set

Output: Adaptive Upper Bound, W_{UB}

```
1: Let A_UB = A_s
2: for each aggressor a of A_u do
3:     Let S_Ac be the set of all known conflicts of a
4:     Set conflictFlag = false
5:     for each conflict aggressor set A_c of S_Ac do
6:         if A_c − {a} ⊂ A_UB then
7:             Set conflictFlag = true;
8:             Break
9:     if conflictFlag = false then add a to A_UB
10: return W(A_UB)
```

As explained in Section 4.1, during its process, the branch and bound algorithm needs to calculate an upper bound on the crosstalk due to a set of aggressors, say A_s, obtained from the current selection and a set of unprocessed aggressors, say A_u. A non–learning branch and bound ends up in finding the overall crosstalk due to $A_s \cup A_u$. However, this is not realistic if any of the conflicting aggressor sets from the conflict database are present in $A_s \cup A_u$.

Adaptive Bounding is based on Property 3, described in Section 4.4.2. With the Adaptive Bounding technique, only a maximal set of aggressors from A_u are considered along with A_s to calculate the upper bound. As the learning process in

the branch and bound is supported by the conflict–driven mechanism of the SAT solvers, application of this technique does not add any considerable overhead to the optimization process. The Adaptive Bounding process is described in pseudocode in Procedure 4.2.

4.5 Implementation and results

The proposed techniques have been implemented in C++ and tested in an industrial environment on several ISCAS 89 benchmark circuits. The details of the implementation and results of these approaches are provided in Chapter 7.

Chapter 5

Conservative False Noise Analysis

As discussed in the previous chapters, a basic requirement of False Noise Analysis is that it is conservative. A non-conservative approach is not acceptable as it could result in an optimistic situation where the predicted worst case crosstalk scenario is better than the actually possible worst case. The solutions to the false noise problem which rely on the zero-delay circuit model are optimistic as they do not consider glitches or hazards of the circuit.

This chapter discusses the optimism arising from the zero-delay model, the properties of glitches and the different types of constraints of the circuit that can be considered in order to perform a conservative false noise analysis.

5.1 Analysis of the zero-delay model

As explained in Section 2.1, the zero-delay model assumes a zero-delay on the gates and interconnects of the circuit, which means that signal transitions propagate instantaneously through the circuit. As a result, the time line can be divided into two periods: pre–transition and post–transition. So, the problem can be solved as two separate logic problems for the pre–transition and post–transition states, after which, both the results are analyzed together to check if the logic is satisfiable.

As any signal can switch at most once according to this model, glitches in the circuit that might occur due to the inputs switching at different times are not caught. In the next section (Section 5.2), an example explaining how the presence of a glitch can impair the analysis using zero-delay model is explained.

5.2 Glitch problem

This section explains with example the optimism arising in false noise analysis as a result of neglecting the dynamic behavior of the circuit. Consider the circuit of Figure 5.1. Net V is the victim and nets B and C are its aggressors. If the cell delays of the circuit are neglected, one can come to the conclusion that the nets B and C should always assume opposite values and hence should switch in opposite directions. This implies that only one of the two aggressors B and C can induce crosstalk on the victim V. This is not quite true when the delays depicted in the circuit diagram are considered. For example, if a glitch exists at net A, because of different propagation delays to nets B and C a fall transition can happen on both B and C simultaneously as depicted in Figure 5.2. Thus, filtering one of the two aggressors B and C based on a simplistic zero delay model would lead to optimism, if hazards are present on the circuit.

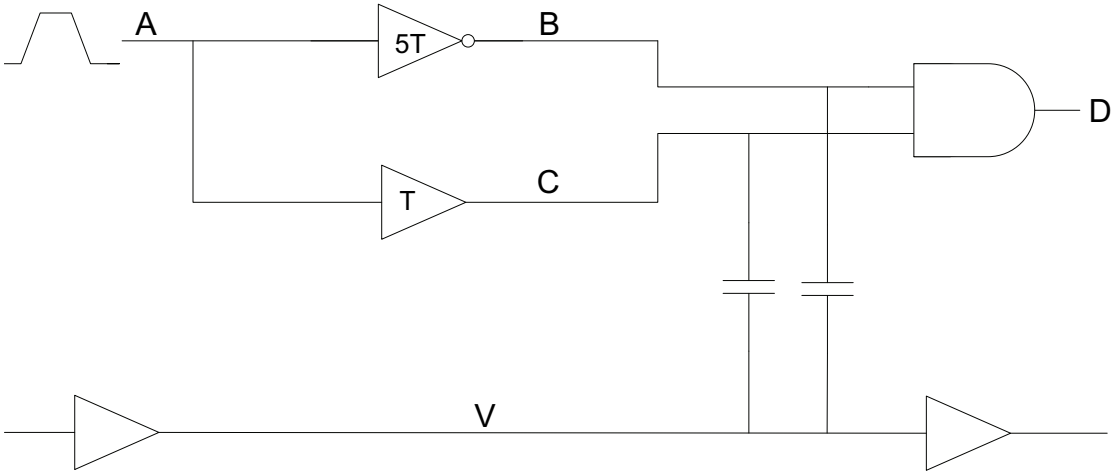


FIGURE 5.1: Glitch problem - example circuit

5.3 Properties of glitches (hazards)

The properties of glitches have been studied and analyzed. They are explained in the following subsections.

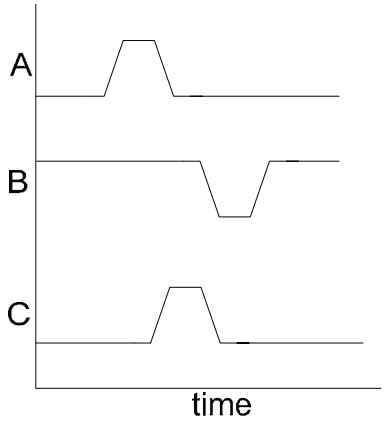


FIGURE 5.2: Glitch problem – waveforms

5.3.1 Glitch generation

A glitch or a hazard occurs at the output of a standard cell if its inputs switch simultaneously in such a way that they change the state of the output for a short period of time. However, simultaneous input switching does not always result in the generation of a glitch or a hazard. A glitch occurs only if certain conditions are met at the inputs of the glitch producing cell.

As standard cells are, in general, designed to be glitch free for single input switching, a glitch can occur only if two or more inputs simultaneously switch. Because of the inertia of cells, they do not produce glitches for inputs switching with a switching time difference of less than a certain threshold. This threshold depends on the timing arcs of the cell and can be characterized. An example on how glitches are generated and propagated is explained in Section 5.3.3.

5.3.2 Glitch propagation

Depending on the rise and fall times of a cell, a pulse applied at the input of it can dilate or shrink. Consider the example in Figure 5.3. The pulse widens because the rise time t_r is less than the fall time t_f. Similarly, in the example of Figure 5.4, the rise time t_r is greater than the fall time t_f and hence the pulse shrinks.

Because of the low-pass filtering effect of the combinational cells in a circuit, many glitches that are not wide, even though they reach V_{dd}/GND, get attenuated and filtered in the process of their propagation.

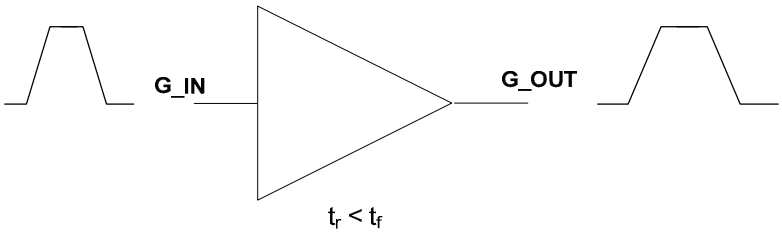


FIGURE 5.3: Example – pulse dilation

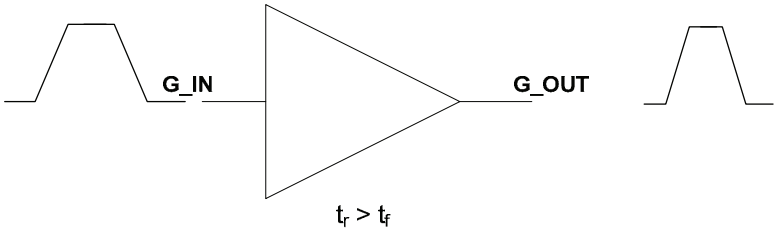


FIGURE 5.4: Example – pulse shrinkage

5.3.3 Experimental study on glitches

In order to depict the process of glitch generation and propagation, the circuit in Figure 5.1 was simulated by applying a rise transition at pin A and a fall transition at pin B. The switching time of A was kept constant, while the switching time of B was varied. This process was repeated for three different slews of 50ps, 125ps and 250ps on the inputs. From the results of these simulations shown in Figure 5.6, Figure 5.7 and Figure 5.8, it can be observed that a glitch is not generated for the instances in which the switching time difference is less than a few ps, depending on the slew. It can also be noted that many of the glitches applied at G_IN of the circuit in Figure 5.5 are much attenuated or filtered at G_OUT.

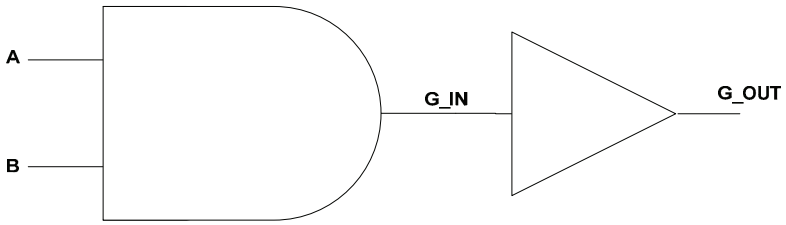


FIGURE 5.5: Inertia of glitches - example circuit

A study has been performed on 382 combinational cells of a 90nm in–house standard cell library to understand the generation and propagation of glitches due to 2-input simultaneous switching. The experimental setup used for this study automatically creates a circuit similar to the one in Figure 5.5, in which the buffer is replaced by the cell under test is used. For these circuits, the non-switching inputs

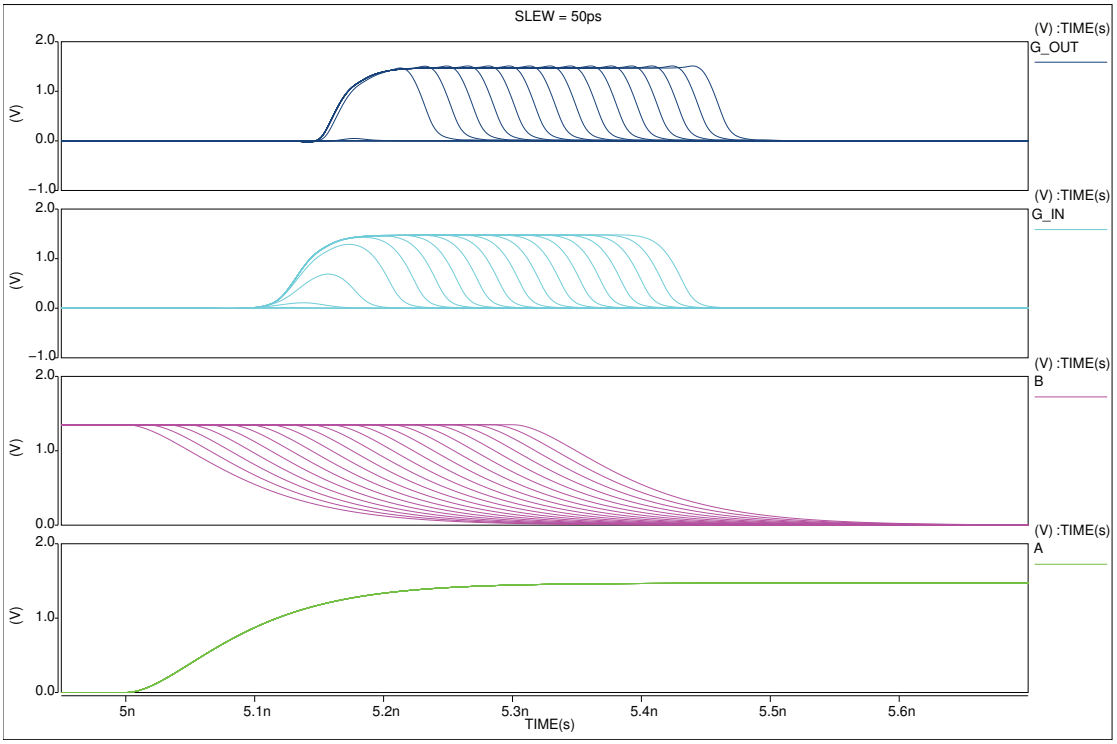


FIGURE 5.6: Glitch inertia – minimum input switching time difference – slew = 50ps

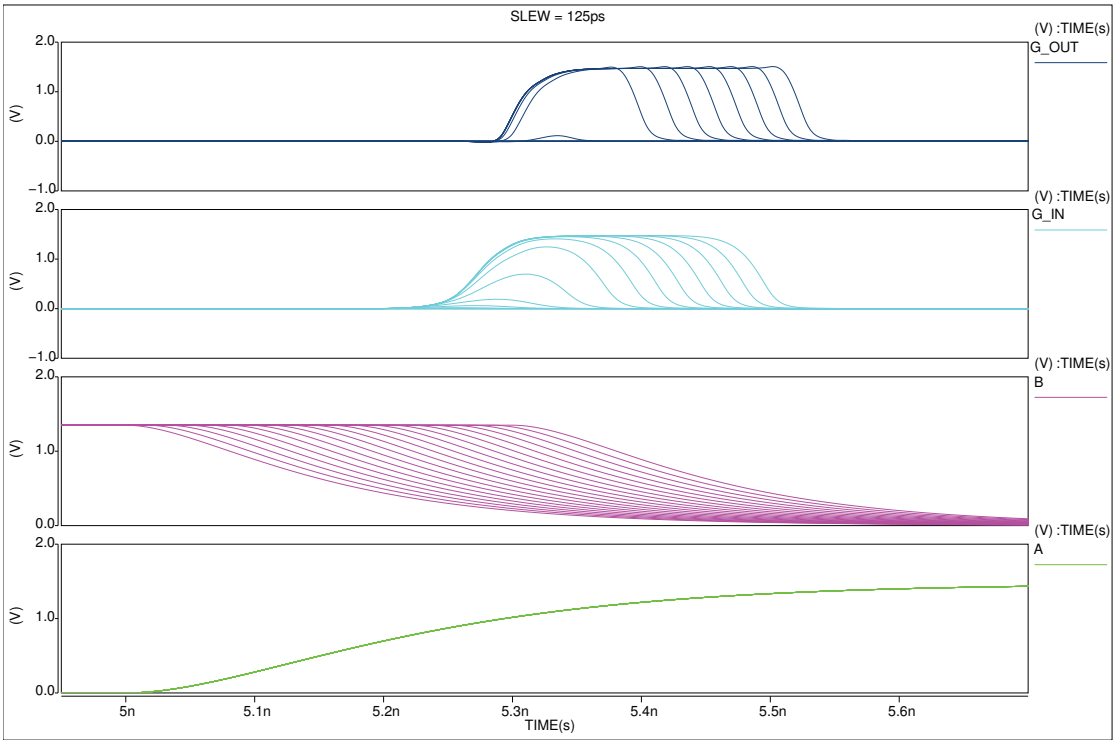


FIGURE 5.7: Glitch inertia – minimum input switching time difference – slew = 125ps

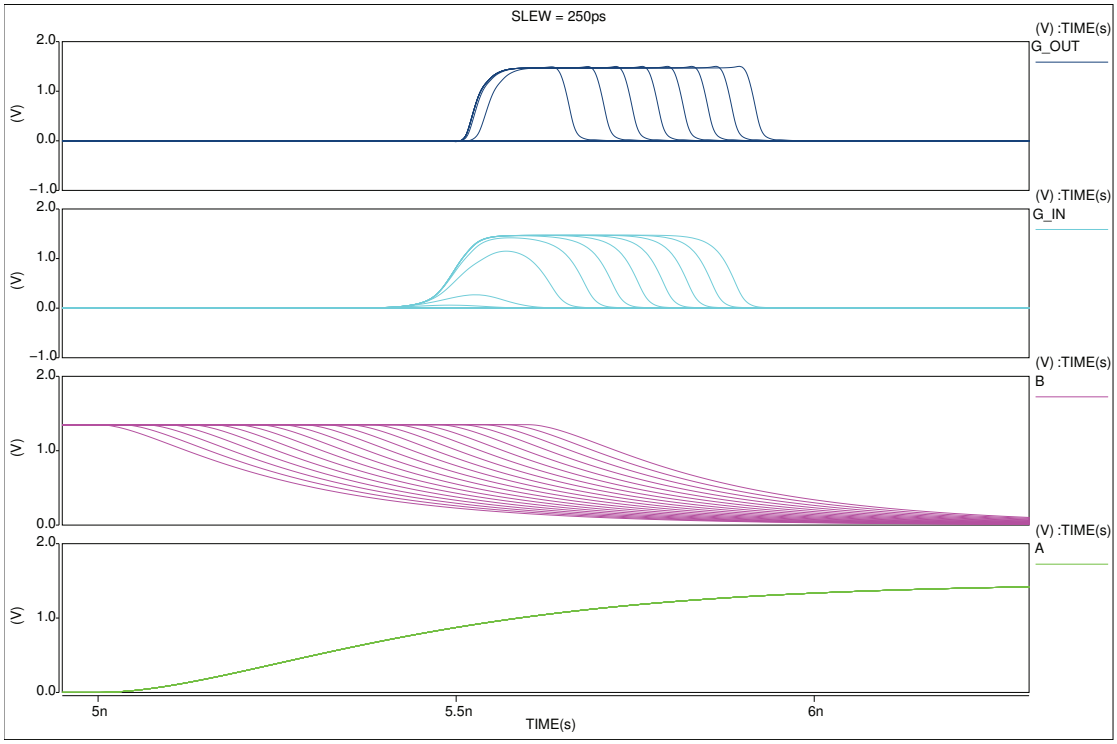


FIGURE 5.8: Glitch inertia – minimum input switching time difference – slew = 250ps

of the cell under test are fed with their non-controlling values. By using a bisection algorithm, these circuits are simulated to find the minimum input switching time at the inputs (A and B in Figure 5.5) that is required to see a glitch of $50\%V_{dd}$ width greater than 1ps. The $50\%V_{dd}$ width of the input glitch for this case gives the maximum width of the glitch that can be filtered by the standard cell. This analysis was performed for three different slews are the inputs – 50ps, 125ps and 250ps.

Figure 5.9, Figure 5.10 and Figure 5.11 give the distribution of the library cells by means of the minimum input switching time difference that is required to see a glitch at the output that is wider than 1ps. For these histograms, time bins of width 5ps each are used. The average minimum input switching time difference for all the 382 cells tested using input slews of 50ps, 125ps and 250ps is found to be 59.45ps, 82.91ps and 118.76ps, respectively.

Figure 5.12, Figure 5.13 and Figure 5.14 give the distribution of the library cells by means of the maximum width of a glitch that can propagate through them and produce an output glitch that is thinner than 1ps. For these histograms, time bins of width 5ps each are used. The average maximum glitch width that can be

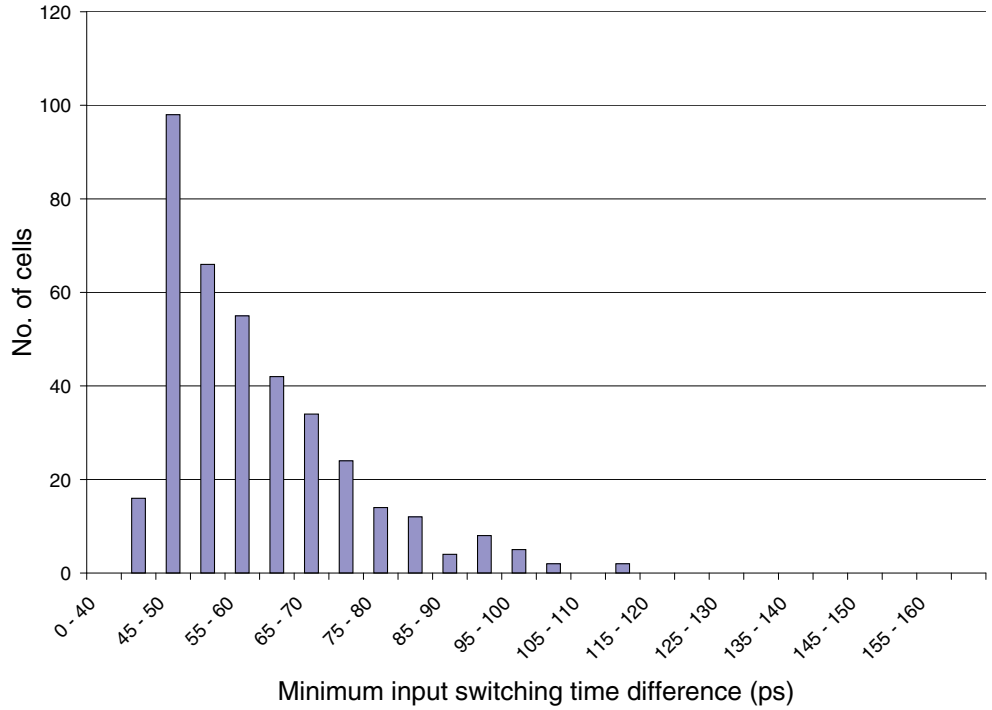


FIGURE 5.9: Glitch inertia histogram – minimum input switching time difference – slew = 50ps

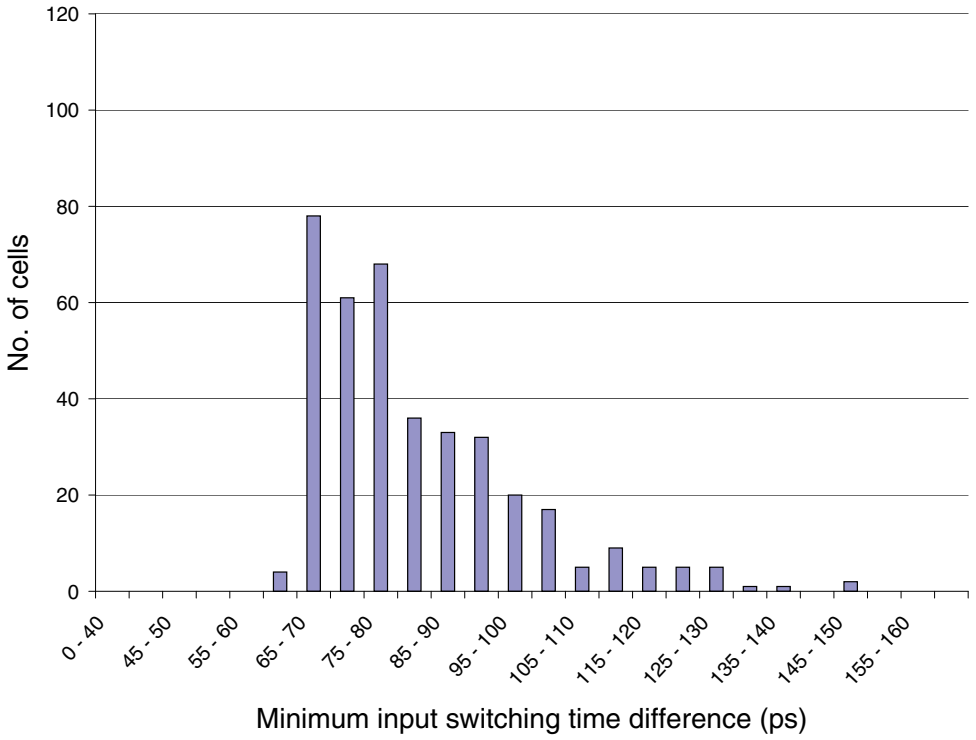


FIGURE 5.10: Glitch inertia histogram – minimum input switching time difference – slew = 125ps

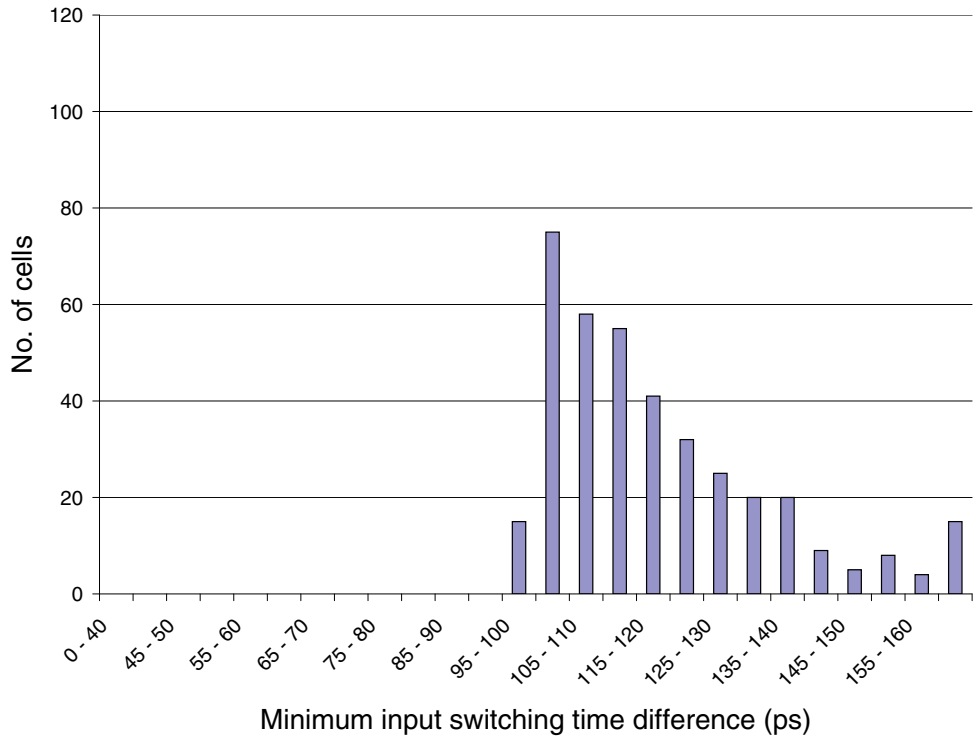


FIGURE 5.11: Glitch inertia histogram – minimum input switching time difference – slew = 250ps

filtered for all the 382 cells tested using input slews of 50ps, 125ps and 250ps is found to be 52.46ps, 86.52ps and 127.49ps, respectively.

The inertia that a cell applies for the generation or propagation of a glitch can be understood from these histograms. It is also evident from these histograms and the plots in Figure 5.6, Figure 5.7 and Figure 5.8 that an increasing slew has a restrictive effect on the generation or propagation of glitches.

The outcome this analysis is in line with the study of multiple input switching done in [11], which indicates that there should be a minimum separation between the inputs for the output to complete its transition.

For further reading on the properties of glitches, their propagation and consideration in various types of analysis, the reader is referred to [11, 16, 17, 28, 69, 43]

5.4 Types of constraints

In this section, we discuss various types of constraints of the circuit that can be considered to perform a conservative false noise analysis.

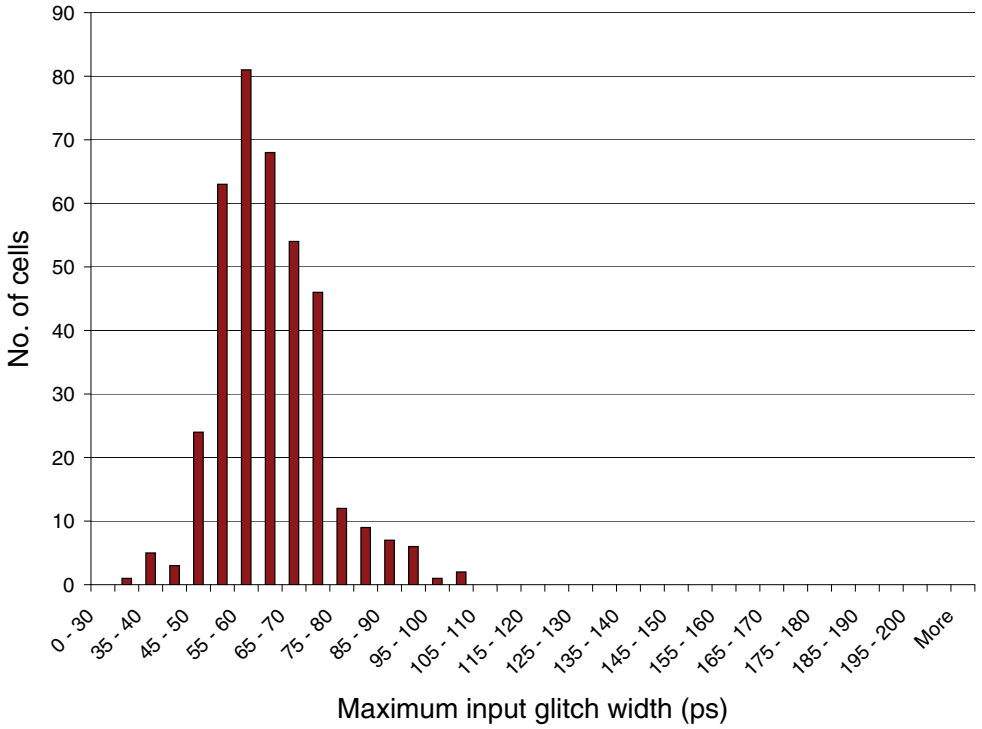


FIGURE 5.12: Glitch inertia histogram – maximum input glitch width – slew = 50ps

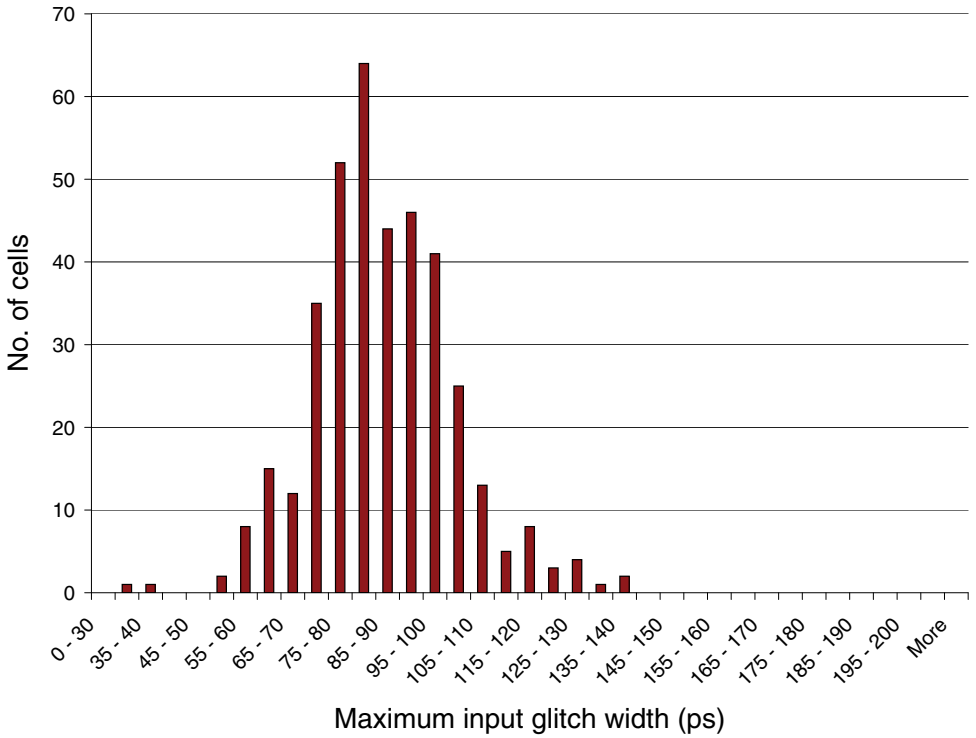


FIGURE 5.13: Glitch inertia histogram – maximum input glitch width – slew = 125ps

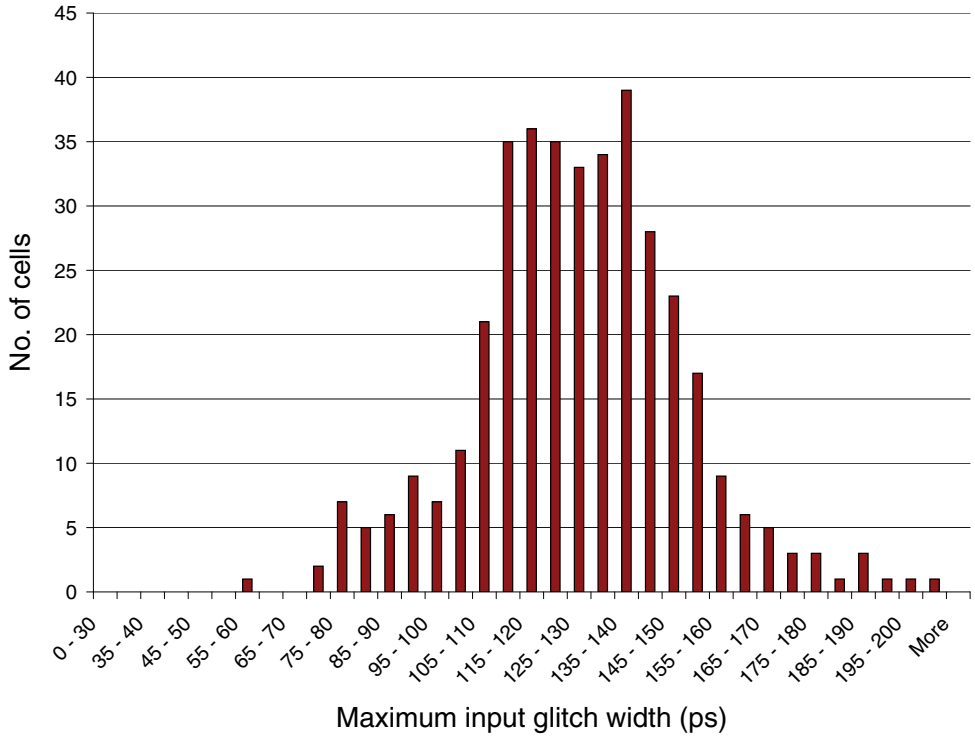


FIGURE 5.14: Glitch inertia histogram – maximum input glitch width – slew = 250ps

5.4.1 Switching windows

A switching window (see definition in Section 2.2) of a port for a specific type of transition (rise or fall) indicates the earliest and the latest time points at which such a transition could occur. For an aggressor to be able to induce crosstalk on its victim, its switching window should overlap with the switching window of the victim. All aggressors whose switching windows do not overlap with the switching window of the victim can be filtered.

5.4.2 Reconvergence correlation

If two aggressors, or an aggressor and its victim have a common predecessor, the calculation of the timing windows of one net with respect to the other can be done more accurately, by throwing away the unnecessary uncertainty at the common predecessor. Such a relation between the timing windows of the two aggressors or the aggressor–victim pair is called reconvergence correlation. This has been handled in [58].

5.4.3 Logic correlations

Because of the inherent logical properties of any circuit, some of its signal combinations are prohibited. Such correlations forced by the logic of the circuit are called logic correlations. For a conservative analysis, the logic correlations used should not rule out the possibility of glitches by using simplistic assumptions like the zero-delay model. For example, see the constraints deduced in the Timing Arc Based Logic Analysis explained in Chapter 6.

5.4.4 Side-input constraints

Side-input constraints correlate the timing and logical properties of the circuit. They constrain the validity of timing arcs based on their side-input conditions. As will be explained in Chapter 6, these constraints can be used to calculate precise timing windows called logical timing windows, which play a vital role in the reduction of crosstalk pessimism.

Chapter 6

Timing Arc Based Logic Analysis (TABLA)

A pure logic analysis under the zero delay assumption does not guarantee conservatism because it can be optimistic in the presence of any glitches in the circuit. To capture the effect of glitches, a detailed analysis that assumes a conservative delay model should be used. A commonly known way to handle glitches in circuits is by using the circuit unrolling technique described in Section 2.7. However, circuit unrolling is a very costly and limited accuracy technique aiming at the conversion of the temporal validity problem into a logical satisfiability (SAT) problem.

Hence, it is essential to have a method that can efficiently handle the timing and logic constraints of the circuit. Timing Arc Based Logic Analysis (TABLA) is one such procedure that aims at analyzing the temporal and logical behavior of a circuit simultaneously in a conservative way to check if a particular crosstalk scenario is valid. It uses to separate, but tightly coupled solvers to simultaneously verify the validity of timing and logic of the circuit avoiding the necessity to use the highly complicated circuit unrolling technique. TABLA offers the flexibility to perform a fast, sub-optimal (but, still conservative) analysis that considers STA timing windows that are independent of the logical model of the circuit, or a slow, but optimal analysis that considers logical timing windows calculated using a logically valid input switching vector obtained from a logic solver[1].

The fundamental difference between TABLA and other approaches in [33, 35, 32, 1, 50, 64, 52, 53, 9] is that TABLA exploits the logical behavior of timing arcs

[1]Throughout this thesis, the term 'logic solver' is used as an alternative to 'SAT solver'.

of the cells of the circuit, whereas the others deduce logic correlations among the nets of the circuit.

In this chapter, the details of this method are discussed. Throughout this chapter, we take the example of the simple circuit in Figure 6.2 to demonstrate the concept in detail.

6.1 Drawbacks of circuit unrolling

The technique of circuit unrolling is discussed in Section 2.7. As explained there, the size of the converted problem, i.e., the number of variables in the unrolled SAT instance is equal to the number of time frames (or, time slices) multiplied by its original size. In other words, the number of variables in the unrolled SAT instance is

$$n_{unrolled} = n_{frames} \times n_{original} \tag{6.1.1}$$

where $n_{original}$ is the number of variables in the original SAT instance and n_{frames} is the number of time frames considered. Considering the exponential complexity of the SAT problem, this can be a very inefficient solution, if the number of time frames is large, because of a potential blowup of the number of variables. A large number of time fames is required to maintain reasonable accuracy. The solution gets highly inaccurate when the time slicing is not fine enough. The main drawback of the circuit unrolling technique is that it handles timing data very inefficiently by converting it into a boolean SAT instance. This leads to two interdependent problems:

Loss of information: Conversion of the continuous timing data into a boolean SAT instance leads to loss of information, which is dependent on the width of the time frames (or time slices) being considered. It is often the case [64, 52, 53, 9] that the number of time frames that can be used in practice is limited, above which the method gets infeasible. The direct consequence of this is the limited accuracy of the obtained results.

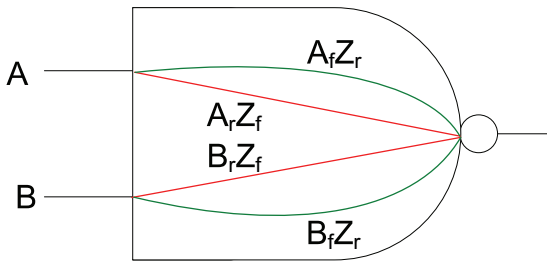
Run time: Because of the exponential complexity of the SAT problem, increase in the number of variables can drastically increase the run time. A possible trade-off here is to reduce the number of time slices used, which affects the accuracy, on the other hand.

Because of the interdependence, an attempt to reduce the loss of information leads to an increased run-time and vice versa.

6.2 What is a timing arc?

A timing arc is a pin-to-pin timing path through one gate. Each timing arc has a start pin and an end pin. The start pin can be an input pin or an inout pin, and the end pin can be an output pin or an inout pin[2].

The set of all timing arcs of any given cell is an exclusive list of all possible paths through it. Depending on the complexity of the gate, there can be timing arcs with similar transition scenarios, but with different side input conditions. Consider the example of a NAND gate, which has four timing arcs as depicted in Figure 6.1.



$A_r Z_f$ – A rise & Z fall
Condition: $B = 1$

$A_f Z_r$ – A fall & Z rise
Condition: $B = 1$

$B_r Z_f$ – B rise & Z fall
Condition: $A = 1$

$B_f Z_r$ – B fall & Z rise
Condition: $A = 1$

FIGURE 6.1: Example – Timing arcs and side input conditions of a NAND gate

Each timing arc represents a possible transition scenario at an input-output pin pair of its corresponding logic gate, and also the logical conditions under which such a transition scenario can actually happen. Further, it also represents the possible latency of the cell for this input-output transition scenario. This property of a timing arc makes it suitable for a closer integration of the logic and timing analyses of a circuit.

6.3 Circuit timing graph

A circuit timing graph is a directed acyclic graph (DAG) with its vertices representing the ports of the circuit and edges representing the timing arcs or nets

[2]The only exception are constraint timing arcs, such as setup or hold constraint timing arcs, which start and end at input pins. These are not considered in this approach because we consider only combinational logic clouds starting from primary inputs or sequential cell outputs, and ending at primary outputs or sequential cell inputs.

connecting them. Its vertices and edges are assigned properties of their respective circuit elements that are required for TABLA. Using a graph structure to store the circuit information is helpful for the quick retrieval of data in a topological order. It also enables the usage of various existing graph algorithms such as topological sorting, graph coloring etc.[3]

Although the graph operations of TABLA can be explained using formal modeling, a descriptive approach to explain them is used in this thesis.

In Section 6.3.1, we see how the mapping of information from the circuit elements to the circuit timing graph is done. In Section 6.3.2, we take an example to see how the timing graph of a sample circuit is built.

6.3.1 Mapping the circuit information

In this section, we see how the circuit information is mapped on to the circuit timing graph, and also how the vertices and edges of the graph are classified.

Vertices

Each port[4] of the circuit has two corresponding vertices in the circuit timing graph: one representing 'rise' transition and the other representing 'fall' transition. The 'rise' vertices are attributed the rise timing information such as the rise timing window, the maximum time taken for the rise transition etc. Similarly, the 'fall' vertices are attributed the fall timing information.

The two vertices (rise and fall) belonging to any port of the circuit are called its paired vertices. They are paired in such a way that each of them stores the 'vertex id' of the other. This helps in an easy traversal through the circuit structure.

Based on their position in the graph, vertices are further classified into four types – input, net receiver, timing arc receiver and virtual:

- An *input vertex* is a vertex corresponding to a primary input.

[3]Most of these algorithms are available for direct usage from graph libraries like BGL (Boost Graph Library), LEDA (Library of Efficient Data Types and Algorithms) etc.

[4]A port is an interface to an instance of a cell or a module through which it is connected to other instances. Each port corresponds to a specific pin of the cell or module represented by its instance. A port can be of *input*, *output* or *inout* type, depending on the direction of its data flow.

- A *net receiver vertex* is a vertex whose corresponding port is the receiver port of a circuit net.
- A *timing arc receiver vertex* is a vertex corresponding to a port, which is connected to the 'to' pin of a timing arc of any cell in the circuit. In other words, its corresponding port is the output port of a cell instance.
- A *virtual vertex* is an imaginary vertex that is connected to the input vertex or another imaginary vertex that helps in making the circuit timing graph a rooted graph.

Each vertex is assigned an edge literal that represents the type of transition its corresponding port assumes. An edge literal can represent five possible types of signal states – rise, fall, low, high and undefined. Hence, the state of a port assumes can be identified by getting the edge literal information of any of its two corresponding vertices.

In the course of TABLA, each vertex of the circuit timing graph can either be colored or uncolored. A vertex is colored, if it is not up-to-date and needs to be processed. It is uncolored, if it is up-to-date and need not be processed.

Edges

Edges of the circuit timing graph correspond to either a timing arc or a net branch, which are the basic delay elements of a circuit. Each timing arc of the circuit has a corresponding edge in the timing graph. Similarly, each net of the circuit has its corresponding edges, twice the number of its receiver ports, in number, connecting the 'rise' driver vertex to the 'rise' receiver vertices and the 'fall' driver vertex to the 'fall' receiver vertices.

Edges of the circuit timing graph can be classified into three categories – timing arc, net branch and virtual:

- A *timing arc edge* is an edge corresponding to a timing arc of the circuit.
- A *net branch edge* is an edge corresponding to a branch of a net of the circuit. Each net branch has two edges in the circuit timing graph corresponding to it, connecting the fall-fall and rise-rise vertices of the respective ports.
- A *virtual edge* is an imaginary edge connecting two virtual vertices.

Each path of the circuit can be represented as the combination of different timing arcs through which it passes. Each edge is attributed with the minimum and the maximum delay possible through it considering the worst case situations in a conservative manner. With this information, it is possible to calculate the minimum and maximum possible delays on any path in the circuit timing graph, and hence indirectly the minimum and maximum delays on the circuit path it corresponds to. Hence, the delay on a path can be conservatively calculated by summing up the delays on the individual edges it passes through, i.e., by adding up either the max or the min delays.

6.3.2 Example of a circuit and its timing graph

In this section, we consider the sample circuit in Figure 6.2 to demonstrate how its timing graph looks and how the information of the circuit is mapped on to the graph.

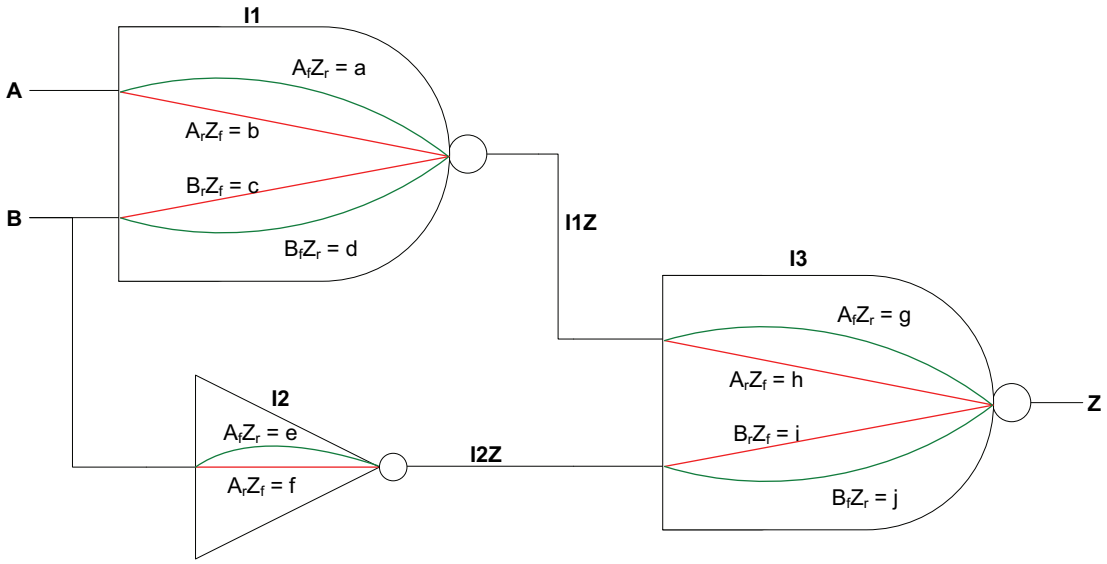


FIGURE 6.2: Example – A circuit with two NAND gates and an inverter

Figure 6.2 consists of two NAND gates and an inverter. It has 2 primary input ports, 8 internal ports and a primary output port. Also, it has a total of 10 timing arcs; 4 for each of the NAND gates and 2 for the inverter. The names of the instances, ports and timing arcs are shown below:

Instances	:	NAND gates	–	$I1$, $I3$
		Inverter	–	$I2$
Ports	:	Primary input	–	A, B

		Internal	–	$I1/A$, $I1/B$, $I1/Z$, $I2/A$, $I2/Z$, $I3/A$, $I3/B$, $I3/Z$
		Primary output	–	Z
Nets	:	A	–	$A \rightarrow I1/A$
		B	–	$B \rightarrow I1/B$, $B \rightarrow I2/A$
		$I1Z$	–	$I1/Z \rightarrow I3/A$
		$I2Z$	–	$I2/Z \rightarrow I3/B$
		Z	–	$I3/Z \rightarrow Z$
Timing arcs	:	$A_f Z_r$	–	a, e, g
		$A_r Z_f$	–	b, f, h
		$B_r Z_f$	–	c, i
		$B_f Z_r$	–	d, j

Figure 6.3 depicts the circuit timing graph of this circuit. Each of the ports of the circuit has two corresponding vertices in the graph, one corresponding to 'rise' (drawn in green color) and the other corresponding to 'fall' (drawn in red color). For differentiation from other vertices, vertices belonging to primary input ports are drawn in rectangular shape. Similarly, vertices belonging to internal ports and primary output are drawn in oval shape. The vertices drawn in circular shape are virtual vertices, which are imaginary and are required to make the circuit timing graph a rooted graph.

While the virtual vertices do not provide any information other than their name, each of the non-virtual vertices provides the following information:[5]

1. Name of the port to which it belongs to.
2. The kind of transition it corresponds to, i.e., rise or fall. This information is encoded in the color of its boarder. A vertex with red border represents fall transition on its corresponding port and similarly, a vertex with green border represents rise transition.
3. The maximum time taken for the transition it represents.
4. Whether or not the vertex needs to be processed. If the vertex is filled in color, then it needs to be processed. If it is uncolored, then either the vertex is not under consideration or it is up-to-date. The details are discussed in Section 6.6.4.

[5]Some of the information that the circuit timing graph holds is not depicted in the graph shown in Figure 6.3 because of space constraints

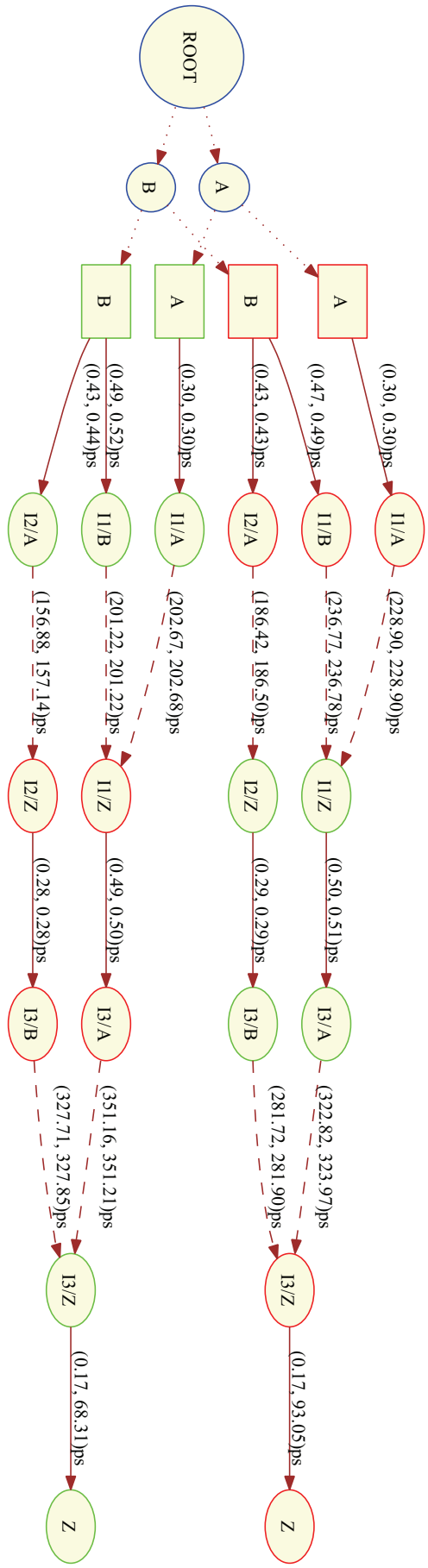


FIGURE 6.3: Timing graph of the circuit in Figure 6.2 [a][b][c][d]

[a]All the delays provided are specific to the technology used, which is not of any significance in the discussion here.
[b]Because of space constraints all the data that the circuit timing graph holds is not depicted in this figure.
[c]Vertex notation: circle – virtual vertex, rectangle – input vertex, oval – net receiver or timing arc receiver vertex
[d]Edge notation: dotted line – virtual edge, solid line – net branch edge, dashed line – timing arc edge

5. After each vertex is processed, it stores the logical timing window of its corresponding port for the transition it represents. A logical timing window is different from an STA timing window in the sense that it considers the logical and temporal validity of timing arcs that are considered for its calculation. The procedure to calculate logical timing windows is discussed in Section 6.6.5.

As mentioned in the previous section, three types of edges exist. The solid edges in the timing graph represent net branches. The dotted edges, which are connected to the virtual vertices on either of their ends represent virtual edges. And, the dashed edges connecting two internal port vertices represent timing arcs. The type of transitions at the input and output of a timing arc can be observed from the colors of the vertices it connects. For example, if the 'from' vertex of a timing arc edge is red (green), it means that the timing arc represents a fall (rise) transition at its input. Similarly, if its 'to' vertex is red(green), it means that the timing arc represents a fall (rise) transition at its output. The color of the edges does not have any significance.

Each of the non-virtual timing arc edges provides the following information:

1. The Id of the timing arc it corresponds to.
2. The minimum and maximum delays possible on the timing arc.
3. Although it is not depicted because of the space constraints, they also provide information on the side input condition. They carry the information on which vertex/vertices (i.e., the port/ports represented by it/them) should assume which logic value/values for the timing arc to be valid.

Similarly, each of the net branch edges provides the following information:

1. The minimum and maximum delays possible on it.
2. The kind of transition that corresponds to the edge. This can be identified by the border colors of the two vertices connected by it, which should essentially be the same. If they are red in color, then the edge propagates a fall transition. Similarly, if they are green in color, then the edge propagates a rise transition.

The virtual edges connecting a virtual vertex and a primary input vertex provide the Id of the virtual timing arcs they represent.

6.4 Key idea

As discussed in Chapter 3, the basic drawback of all the existing approaches that aim at reducing false noise [33, 35, 32, 1, 50, 64, 52, 53, 9] is that they are either not conservative or not efficient. The main reason for this is that they either use simplistic non-conservative delay models or they attempt to convert the whole logic and timing problem into a SAT problem, as explained in Section 3.6.

Timing data can be handled better by using a dedicated timing solver rather than a logic solver. Hence, instead of converting timing information into a boolean SAT instance, TABLA uses two separate solvers for logic and timing. It uses a less restrictive SAT solver that assumes the unbounded delay model (see Section 2.1) and a timing solver that assumes the min-max delay model designed to verify the solutions provided by the SAT solver.

TABLA performs a logically driven timing analysis that verifies the underlying logic and timing constraints with the use of two independent logic and timing solvers. The logic solver acts as a master entity and proposes solutions that are valid in the logical perspective to the timing solver. The timing solver then checks if the logically valid solution is valid also temporally. If so, the crosstalk configuration is considered as valid. Otherwise, the timing solver indicates this to the logic solver, which would then verify if another logically valid solution exists. If so, it is handed over to the timing solver and the loop continues. Otherwise, the configuration is considered invalid.

Thus, TABLA searches (implicitly or explicitly) for all possible switching scenarios under the unbounded delay assumption and checks if they can be sensitized temporally under the more realistic min-max delay assumption (each of the timing arcs of the circuit assumes a delay within its min-max bounds). The higher level flow of TABLA is depicted in Figure 6.4. It should be noted that TABLA verifies the validity of a given crosstalk scenario. In other words, TABLA corresponds to the 'Constraint Satisfiability Checker' module of the false noise analysis flow (see Section 2.5).

6.5 Logic solver

As mentioned earlier, the basic element of TABLA is a timing arc. The timing arcs of a cell represent both its functional and temporal behavior. Hence, it is

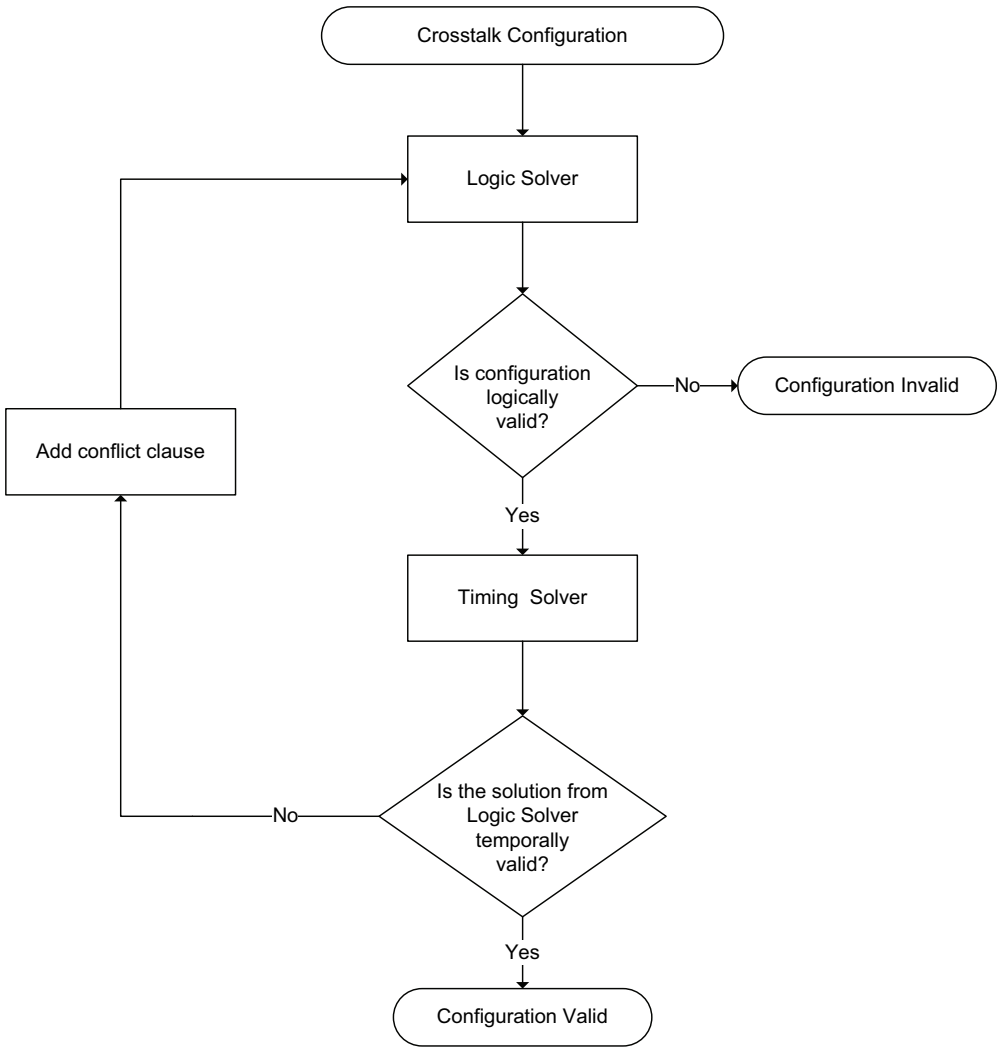


FIGURE 6.4: TABLA: top level flow chart

understood that they can be used to efficiently model the temporal logic analysis problem in both logical and temporal domains.

The logic solver considers correlations among the timing arcs of a circuit and proposes logically valid solutions to the timing solver (described in Section 6.6) for the verification of their temporal validity. If a logically valid solution of the logic solver is found to be temporally invalid, then a conflict clause is added to the logic solver that forbids this solution and possibly other similar solutions.

In order to make sure that glitches (hazards) of the circuit are considered by TABLA, and that the underlying SAT instance is simple, unbounded delay model is used by the logic solver. As a result, the modeling of logic gets very simple. On the other hand, because of this overly conservative assumption, this model is less restrictive with respect to false noise reduction, unless it is combined with a

method that considers more accurate timing windows. In the context of TABLA, the clauses for logic solver are deduced from the circuit under the unbounded delay assumption. The logically valid solutions obtained from the logic solver, which may in reality be invalid if accurate timing windows are considered, are verified by a timing solver for their temporal validity.

6.5.1 Modeling the logic problem using timing arcs

In TABLA, each of the timing arcs of the circuit under consideration is represented by a boolean variable. In the logic domain, this variable can be related to the variables of other timing arcs that need to be 'true', if its corresponding timing arc has to be active. As we will see in Section 6.6, in the temporal domain, only the timing arcs that can be logically active are considered to calculate accurate and more realistic timing windows than the STA timing windows. These timing windows are called 'logical timing windows' as they consider the inherent logic of the circuit and are conditional to an input switching vector, which in turn is obtained from the logic solver.

6.5.1.1 Variable assignment

The activity of each of the timing arcs of the circuit under consideration is represented using a boolean variable. Assignment of logic 1 to a boolean variable implies that its corresponding timing arc is active and similarly, the assignment of logic 0 implies that the timing arc is not active.

In order to control the signal values on the input ports (primary inputs or flip-flop outputs) and the aggressors of a given crosstalk configuration, additional variables are defined. Imaginary timing arcs are assumed at the inputs (primary inputs or flip-flop outputs), and the boolean variables assigned to these timing arcs that control the input switching values are called virtual timing arc variables. Similarly, a boolean variable per aggressor is assigned to control its activity.

To store the static values on nets that have all the timing arcs connected to them inactive, we define a boolean variable for each net of the circuit. The value on this variable is invalid if any of the timing arcs connected to its corresponding net is active. The values of these variables are forced by the side-input conditions of active timing arcs and the circuit logic itself. This will make sure that two timing

arcs whose side-input conditions contradict each other are not active at the same time.

The number of variables of the underlying SAT instance is in the order of the sum of the number of timing arcs and the number of nets of the circuit, and is hence very low compared to the number of variables required in the circuit unrolling process.

6.5.1.2 Deduction of constraints

Constraints that verify the logical consistency of timing arcs are deduced. We consider three kinds of constraints: timing arc constraints, logic constraints and side-input constraints. In what follows, the procedure to deduce these three types of constraints is explained.

Timing arc constraints: Timing arc constraints involve timing arc variables and they check the consistency of timing arcs. They can be further classified into two types: Back tracking constraints and single switching constraints.

Back tracking constraints ensure that a timing arc is active only if at least one of the timing arcs connected to its input is active. This is achieved by making backward implications from any timing arc to the corresponding timing arcs of the immediately preceding instances of its fan-in cone.

Consider the example of timing arcs e and f of the circuit in Figure 6.5. If timing arc e has to be active, then either timing arc b or timing arc c should be active. Similarly, if timing arc f has to be active, then either timing arc a or timing arc d should be active. Hence, we have to capture the implications $e \Rightarrow b + c$ and $f \Rightarrow a + d$. They can be written in the form of constraints $(\overline{e} + b + c)$ and $(\overline{f} + a + d)$.

Single switching constraints ensure that the signals on primary inputs and sequential cell outputs switch only once. Let us consider that the inputs of the NOR gate in the circuit in Figure 6.5 are primary inputs. The number of transitions possible on this port can be restricted to one by adding the constraint clauses $(\overline{a} + \overline{b})$ and $(\overline{c} + \overline{d})$ to the CNF.

Logic constraints: Logic constraints involve the net variables and they check the logical consistency of the static values assigned to the nets of the circuit. Consider the NAND gate in Figure 6.1. As depicted, assume that A, B,

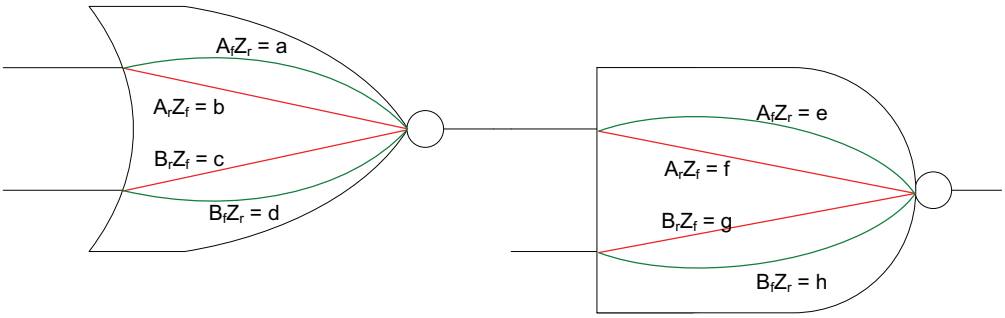


FIGURE 6.5: Example – deduction of constraints for TABLA

and Z are its net variables. Following the logic of the NAND gate, three constraints can be deduced:

1. If $A = 0$, then Z must assume a logic 1. Similarly, if $Z = 0$, then A must assume a logic 1. From this, we deduce the constraint $(A + Z)$.
2. If $B = 0$, then Z must assume a logic 1. Similarly, if $Z = 0$, then B must assume a logic 1. This can be put in the form of the constraint $(B + Z)$.
3. If $A = 1$ and $B = 1$, then Z must assume a logic 0. Following this, we get the constraint $(\overline{A} + \overline{B} + \overline{Z})$.

Logic constraints for other logic gates can be deduced in a similar way, i.e., by checking the restrictions a logic gate has on the values on its pins. Constraints for complex gates can be deduced by applying resolution principle [54] on the constraints of the basic gates they consist of.

Side-input constraints: Side-input constraints verify if the side-input conditions of the corresponding timing arcs are satisfied. They involve both timing arc and net variables. These constraints verify the logical consistency of timing arcs when the side-inputs assume static values. Their significance is explained with example in Section 6.5.3.

The constraint clauses for the entire circuit are deduced in a pre-processing step and are stored in the database. This information is then retrieved for the verification of individual crosstalk constellations in the actual procedure.

6.5.1.3 Capability to consider glitches

As it was already mentioned, the unbounded delay model is less restrictive on the circuit behavior compared to the min-max, fixed or zero delay models. It does not make any assumptions that restrict the occurrence of glitches in the circuit under consideration. In the timing arc based logic modeling used in TABLA, a 'rise-fall' timing arc, if active, implies only that its corresponding input and output should rise and fall, respectively. It does not put any restriction on when they should rise or fall. Also, the constraint clauses generated do not restrict the number of transitions possible on any port of the circuit, except for the input ports (primary input ports and sequential cell output ports), which are restricted to switch at most once. Hence, this model has the capability to consider glitches of the circuit for false noise analysis.

6.5.2 Deduction of constraints on the sample circuit

In this section, we continue with the example of the circuit in Figure 6.2 explained in Section 6.3 to demonstrate the procedure to build a circuit timing graph. Let us assume that the virtual timing arcs shown in Figure 6.3 with Ids 10, 11, 12 and 13 be represented by variables A_r, A_f, B_r and B_f respectively. The deduction of different kinds of constraint clauses is explained below:

Timing Arc Constraints As mentioned in Section 6.5.1.2, there are two types of timing arc constraints: back tracking constraints and single switching constraints. The back tracking constraints for the circuit in hand are deduced as explained below:

Consider instance I1 (NAND gate):

- For timing arc a to be valid, the virtual timing arc A_r has to be active. Hence, we have the implication $a \rightarrow A_r$ from which we get the constraint clause $(\overline{a} + A_r)$.
- For timing arc b to be valid, the virtual timing arc A_f has to be active. Hence, we have the implication $b \rightarrow A_f$ from which we get the constraint clause $(\overline{b} + A_f)$.
- For timing arc c to be valid, the virtual timing arc B_f has to be active. Hence, we have the implication $c \rightarrow B_f$ from which we get the constraint clause $(\overline{c} + B_f)$.

- For timing arc d to be valid, the virtual timing arc B_r has to be active. Hence, we have the implication $d \rightarrow B_r$ from which we get the constraint clause $(\overline{d} + B_r)$.

Consider instance I2 (Inverter):

- For timing arc e to be valid, the virtual timing arc B_r has to be active. Hence, we have the implication $e \rightarrow B_r$ from which we get the constraint clause $(\overline{e} + B_r)$.
- For timing arc f to be valid, the virtual timing arc B_f has to be active. Hence, we have the implication $f \rightarrow B_f$ from which we get the constraint clause $(\overline{f} + B_f)$.

Consider instance I3 (NAND gate):

- For timing arc g to be valid, either of the timing arcs b and c has to be active. Hence, we have the implication $g \rightarrow (b + c)$ from which we get the constraint clause $(\overline{g} + b + c)$.
- For timing arc h to be valid, either of the timing arcs a and d has to be active. Hence, we have the implication $h \rightarrow (a + d)$ from which we get the constraint clause $(\overline{h} + a + d)$.
- For timing arc i to be valid, the timing arc e has to be active. Hence, we have the implication $i \rightarrow e$ from which we get the constraint clause $(\overline{i} + e)$.
- For timing arc j to be valid, the virtual timing arc f has to be active. Hence, we have the implication $j \rightarrow f$ from which we get the constraint clause $(\overline{j} + f)$.

Single switching constraints ensure that the primary inputs assume at most one transition. For the primary inputs A and B, they are deduced as explained below:

- Primary input A is connected to the virtual timing arcs A_r and A_f. The situation in which both of them are simultaneously active can be restricted by the constraint $(\overline{A_r} + \overline{A_f})$.
- Primary input B is connected to the virtual timing arcs B_r and B_f. The situation in which both of them are simultaneously active can be restricted by the constraint $(\overline{B_r} + \overline{B_f})$.

Logic constraints Logic constraints constrain the net variables according to the logic of the circuit. As described in the previous section, each logic gate has a specific set of constraint clauses. In the case of the circuit in Figure 6.2, we have the below logic constraints:

Instance I1 (NAND gate) : $(A + I1Z)$, $(B + I1Z)$, $(\overline{A} + \overline{B} + \overline{I1Z})$
Instance I2 (Inverter) : $(B + I1Z)$, $(\overline{B} + \overline{I2Z})$
Instance I3 (NAND gate) : $(I1Z + Z)$, $(I2Z + Z)$, $(\overline{I1Z} + \overline{I2Z} + \overline{Z})$

Side-input constraints The side-input constraints impose the side-input conditions of the timing arcs to make sure that any logical inconsistency is avoided. Side-input constraints of a side-input are active only if none of the timing arcs connected to it are active. The side input condition should be verified, if none of the timing arcs connected to the side input are active. For the circuit in Figure 6.2, side-input constraints are deduced in the following way:

Consider instance I1 (NAND gate):

- Timing arc a has the side-input condition $B = 1$. Hence, if B has a static 0 on it, i.e., if $c = 0$, $d = 0$ and $B = 0$, then timing arc a cannot be active. This can be put in the form of constraint as $(B + c + d + \overline{a})$.
- Timing arc b also has the same side-input condition, i.e., $B = 1$. Hence, the side-input constraint concerning b is $(B + c + d + \overline{b})$.
- Timing arc c has the side-input condition $A = 1$. Hence, if A has a static 0 on it, i.e., if $a = 0$, $b = 0$ and $A = 0$, then timing arc c cannot be active. This can be put in the form of constraint as $(A + a + b + \overline{c})$.
- Timing arc d also has the same side-input condition, i.e., $A = 1$. Hence, the side-input constraint concerning d is $(A + a + b + \overline{d})$.

Consider instance I2 (Inverter):

- It does not have any side-input constraints because it has only one input.

Consider instance I3 (NAND gate):

- Timing arc g has the side-input condition $I2Z = 1$. Hence, if $I2Z$ has a static 0 on it, i.e., if $i = 0$, $j = 0$ and $I2Z = 0$, then timing arc g cannot be active. This can be put in the form of constraint as $(I2Z + i + j + \overline{g})$.
- Timing arc h also has the same side-input condition, i.e., $I2Z = 1$. Hence, the side-input constraint concerning h is $(I2Z + i + j + \overline{h})$.

- Timing arc i has the side-input condition $I1Z = 1$. Hence, if $I1Z$ has a static 0 on it, i.e., if $g = 0$, $h = 0$ and $I1Z = 0$, then timing arc i cannot be active. This can be put in the form of constraint as $(I1Z+g+h+\overline{i})$.
- Timing arc j also has the same side-input condition, i.e., $I1Z = 1$. Hence, the side-input constraint concerning j is $(I1Z + g + h + \overline{j})$.

6.5.3 Significance of side-input constraints and logic constraints

In this section, the significance of side-input constraints and logic constraints in verifying the logical consistency of timing arcs when the side-inputs assume static values is explained. Continuing with the example circuit in Figure 6.2, let us consider the path A—$I1/A$—$I1/Z$—$I3/A$—$I3/Z$—Z. Also, let us assume that the nets B and $I2Z$ have static values on them, i.e., the timing arcs c, d, e, f, i and j are inactive. In other words, $c = d = e = f = i = j = 0$.

There are two possibilities in which this path can be sensitized, which are discussed below:

1. $a = 1$, $h = 1$*:* Because of the single switching constraint $(\overline{a} + \overline{b})$, b must be equal to 0. Now, for the timing arc a to be active, its side-input constraint $(B + c + d + \overline{a})$ needs to be satisfied. Since, $c = d = 0$ as per our initial assumption, the constraint evaluates to $B = 1$. Because of the logic constraint $(\overline{B} + \overline{I2Z})$, $I2Z$ evaluates to logic 0.

   For timing arc h to be active, its side-input constraint $(I2Z + i + j + \overline{h})$ should be satisfied. However, since $I2Z = i = j = 0$, h evaluates to logic 0, which means that we have a conflict. Hence, this possibility should be ruled out.

2. $b = 1$, $g = 1$*:* Because of the single switching constraint $(\overline{a} + \overline{b})$, a must be equal to 0. For timing arc b to be active, its side-input constraint $(B+c+d+\overline{b})$ must be satisfied. Since, $c = d = 0$ as per our initial assumption, B evaluates to logic 1. Applying the logic constraint $(\overline{B} + \overline{I2Z})$, we have $I2Z = 0$.

   Now, for timing arc g to be active, the side-input condition $(I2Z+i+j+\overline{g})$ needs to be satisfied. However, since $I2Z = i = j = 0$, g evaluates to logic 0, which leads to a conflict.

Hence, the path A—$I1/A$—$I1/Z$—$I3/A$—$I3/Z$—Z is a false path under the assumptions made. It would not be possible to identify this path as a false path without the aid of the logic and side-input constraints.

6.6 Timing solver

Timing solver evaluates the logically valid solutions to the false noise problem provided by the logic solver for their temporal validity. As these solutions were proven to sensitize the crosstalk scenario under the unbounded delay assumption made by the logic solver, it is now required to verify if they hold also under the more realistic min-max delay assumption.

The timing solver performs an STA kind of propagation of timing windows from inputs (primary inputs or sequential cell outputs) towards the output (primary outputs or sequential cell inputs) in a topological order. However, it differs from STA in that it checks the logical and temporal validity of timing arcs for their consideration in the calculation of timing windows. If the timing windows of the victim and its aggressors in the crosstalk configuration under consideration overlap, then the configuration is considered as valid. Otherwise, the logic solver is asked to propose a new solution.

As discussed in Section 2.1, the unbounded delay assumption under which the logic analysis is performed by the logic solver is very pessimistic. The timing solver throws away this pessimism to the maximum possible extent by the consideration of the accurate min-max delay model. It performs a temporal sensisizability check on the solutions provided by the logic solver. This check is performed for each solution proposed by the logic solver until a valid solution is found or if it is found that no such solution exists. After each temporal sensitizability check that proves that the solution proposed by the logic solver is temporally invalid, a conflict clause is added to it in order to forbid this solution, as well as other solutions inconsistent with the conflict clause.

The concepts of the timing solver are explained using the example of the circuit in Figure 6.2. Let us consider a crosstalk fall delay scenario, where the victim makes a fall transition and its aggressors make a rise transition with net A of the circuit as the victim and net Z is an aggressor of it. The coupled fan-in cone of this crosstalk cluster consists of all the predecessors of the fall vertex of net A and the

rise vertex of net Z, as shown in Figure 6.6. Figure 6.6 shows the fan-in timing graph of this crosstalk configuration before and after processing.

In this section, we see in detail how the timing solver functions. The list of various topics discussed in this section is given below:

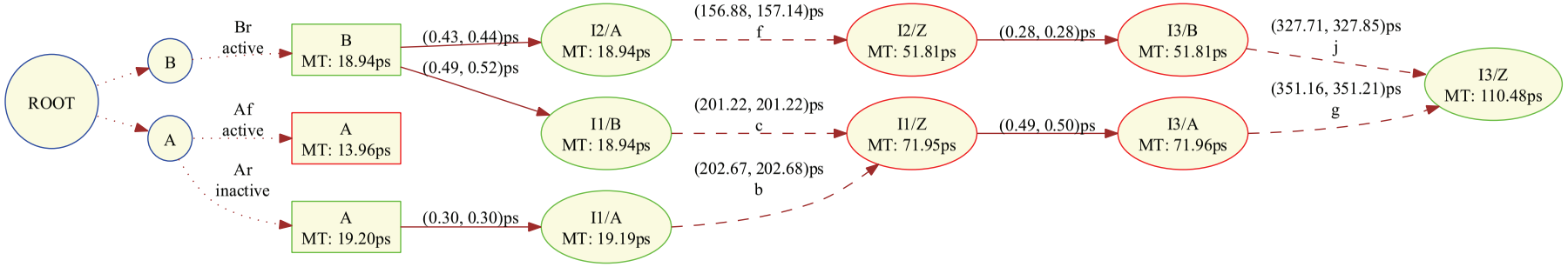
1. Availability of the timing information required by the timing solver.
2. Ordering of the vertices of the timing solver.
3. Identification of the input switching vector corresponding to the logically valid solution proposed by the logic solver.
4. Coloring of vertices to avoid redundancy in calculations
5. Propagation of the input switching vector through the vertices of the fan-in timing graph.
6. Checking whether a timing arc can be active
7. Checking if the logical timing windows of the crosstalk scenario overlap.
8. Deduction of conflict clauses

6.6.1 Availability of timing information

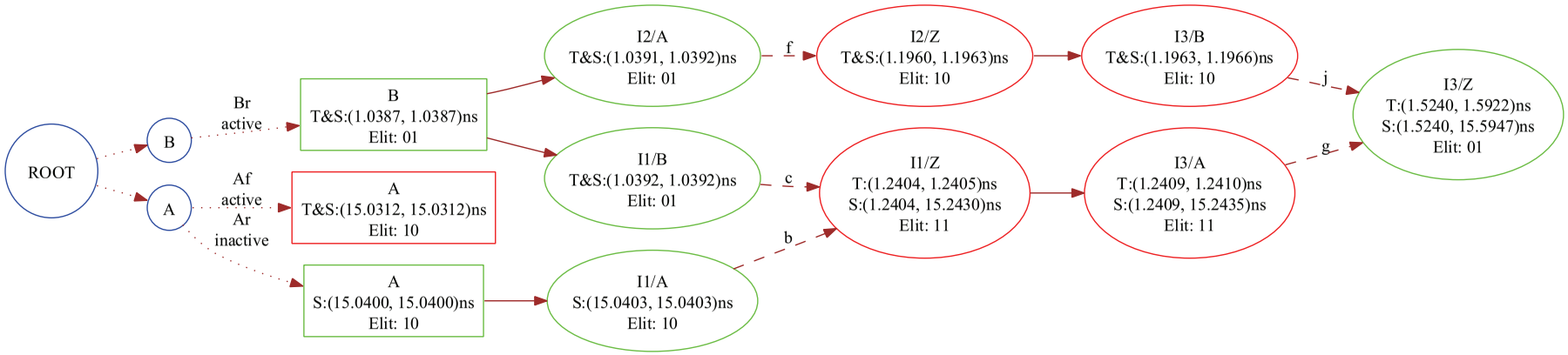
Before making the assumption that some information is available, it is essential to check if such an assumption is realistic. In this section, it is explained how the timing information required for TABLA to evaluate logical timing windows which are more accurate than the static timing windows can be obtained.

In TABLA, the following timing information on circuit elements is required:

1. **Min-max delays on each of the timing arcs of the circuit:** This information can be obtained in two ways:
   (a) *by pre-characterization of cells:* Cells may be pre-characterized to obtain the minimum and maximum possible delays on each of their timing arcs by assuming the extreme conditions. The information obtained through this procedure may not be very accurate, but is still conservative.



(a) Before processing



(b) After processing

FIGURE 6.6: Example – Fan-in timing graph of the fall delay crosstalk constellation of the circuit in Figure 6.2 with net A as victim and net Z as its aggressor

Abbreviations: MT – Maximum transition time, T – TABLA arrival time window, S – STA arrival time window
To avoid redundancy, edge delay information is omitted in Figure (b)
Vertex notation: circle – virtual vertex, rectangle – input vertex, oval – net receiver or timing arc receiver vertex
Edge notation: dotted line – virtual edge, solid line – net branch edge, dashed line – timing arc edge

(b) *by keeping the information obtained from the simulations during STA:* By sorting the events to be processed using the timing sort algorithm [63], it can be ensured that the information required for the processing of any net is available before its processing. It should be noted here that it is required to keep all the events generated during the STA, and not just the extreme events, for this purpose. As the number of events is proportional to the number of timing arcs of the circuit, although it does not lead to a memory explosion, it adds a memory overhead. The min-max delay on each of the timing arcs of the circuit under consideration is obtained from a proprietary STA tool and is stored in the database.

2. **Min-max delays on the nets of the circuit:** Calculation of crosstalk is a chicken-egg problem. However, it has been proven that this can be solved non-iteratively with reasonable accuracy using the timing sort algorithm [63]. This information can also be obtained in two ways:

   (a) *by modeling the net branches as buffers:* A conservative approximation of the minimum and maximum delays possible on a net branch can be used to model the net branches as buffers. This may not be very accurate, but still conservative.

   (b) *by keeping the information from simulations:* As mentioned, based on the timing sort algorithm [63], it can be assumed that the min-max delays of all the nets in the coupled fan-in cone of any net are available at the time it is processed. The min-max delays on the interconnects are obtained from the STA tool as they are available and are stored in the database.

3. **Maximum transition time of the ports of the circuit:** This information is required for the calculation of switching windows from timing windows (see Section 2.2) of the ports in the coupled fan-in cone of the crosstalk constellation. This information can be obtained from the STA tool used to calculate the timing windows.

6.6.2 Ordering of the vertices of the timing graph

The propagation of timing information from inputs (primary inputs or sequential cell outputs) towards the outputs (primary outputs or sequential cell inputs) in

timing solver is done in a very similar fashion to STA except that the timing solver verifies if a timing arc can temporally be active before considering it for propagation of timing windows. Hence, the vertices of the fan-in timing graph are ordered topologically using topological sorting. The vertices are then processed in this order.

A possible topological order to process the vertices of the fan-in timing graph shown in Figure 6.6 is $ROOT_v \rightarrow B_v \rightarrow B_r \rightarrow I2/A_r \rightarrow I2/Z_f \rightarrow I3/B_f \rightarrow I1/B_r \rightarrow A_v \rightarrow A_f \rightarrow A_r \rightarrow I1/A_r \rightarrow I1/Z_f \rightarrow I3/A_f \rightarrow I3/Z_r$ (The subscripts r, f and v indicate that the vertices are of *rise*, *fall* and *virtual* type, respectively). In what follows, we assume that this order has been used to process the vertices of this fan-in timing graph.

6.6.3 Identification of the input switching vector

The input switching vector specific to the solution provided by the logic solver can be identified by checking the values on the boolean variables corresponding to the virtual timing arcs in the fan-in timing graph. As it can be observed from the circuit timing graph in Figure 6.3, each input (primary input or sequential cell output) has two virtual timing arcs connected to it. If one of these two timing arcs is active (at most one timing arc at the inputs can be active because of the single switching constraints), then the input is assumed to make the transition corresponding to that timing arc. Otherwise, the input is assumed to be static. In this case, the value this net would take is obtained from the value on the net variables. If the net variable is at logic 1, then the input assumes logic 1. Otherwise, it assumes logic 0.

Consider again the example of the fan-in cone of circuit in Figure 6.2 shown in Figure 6.6. Assume that net A is the victim and net $I3/Z$ is its aggressor. Also, assume that the fall delay crosstalk scenario is considered. The fan-in timing graph depicts the example of a particular case where the solution provided by the logic solver dictates that the virtual timing arcs with ids 12, 11 and 10 are *active*, *active* and *inactive*, respectively. From this, it can be identified that the solution under consideration (obtained from the logic solver) indicates that the input ports A and B are falling and rising respectively.

6.6.4 Coloring of vertices to avoid redundant calculations

TABLA uses a vertex coloring mechanism to avoid redundancies in calculation of timing windows. A vertex can be either colored or uncolored. A colored vertex indicates that it is ready for processing, and an uncolored vertex indicates that the information on it is either up-to-date or that the vertex need not be processed. Initially, all the vertices of the fan-in graph are marked colored.

An exception to the coloring mechanism holds for the input vertices. These vertices are always processed and are used to identify if the subsequent vertices connected to them need to be processed. This can be done by checking if the current input switching vector changes this input from the value it has obtained from the previous input switching vector. The key is that if the input has not changed, then the vertices connected to its out-edges are not influenced by this vertex. If it is identified that the input has changed from its previous value, then the timing window on it is updated and the vertices connected to its out-edges are colored to indicate that they need to be updated too. Similarly, after processing any colored vertex, all the vertices connected to its out-edges are colored to indicate that they also need to be processed.

From Figure 6.6, which shows the fan-in timing graph of the example described above before and after processing, it can be figured out that all the vertices of the graph are colored before processing and that they are uncolored after processing. Now, if the timing solver has to verify other input vectors from the logic solver, it will make the best use of the already calculated information, by using the coloring mechanism. This saves time because of avoiding redundant calculations.

6.6.5 Propagation of timing windows

As described in Section 6.3, each timing graph vertex is associated with the following attributes, which depend on the logically valid solution from the logic solver being verified by the timing solver:

- A timing window corresponding to its transition type
- An edge literal representing the type of edge (rise, fall, low, high, undefined) on the associated circuit port

Processing of a vertex involves the calculation/update of these two attributes. After topologically ordering the vertices of the fan-in timing graph as described in Section 6.6.2, they are processed one after the other.

The processing of any vertices depends on its type. As explained in Section 6.6.4, vertices other than the input vertices are processed only if they are colored. In other words, all uncolored vertices other than the input vertices that come for processing in the topological order are skipped without being processed, following the information from their color that they are already up-to-date. The procedure to process different kinds of vertices is explained below:

Input vertices Each input vertex has a corresponding virtual timing arc connected to it. The values attributed to the timing window and the edge literal of an input vertex depends on states (active or inactive) of the timing arcs corresponding to the input vertex and its paired vertex. There are three possibilities:

1. *Virtual timing arc connected to the current input vertex is active:* This automatically implies that the timing arc connected to the paired vertex of the current input vertex is inactive because of their single switching constraint. Hence, the timing window of the input vertex corresponding to its transition type (rise or fall) is obtained from the STA tool. Similarly, its edge literal is set to the transition type (rise or fall) corresponding to it.
2. *Virtual timing arc connected to the paired vertex of the current input vertex is active:* This automatically implies that the timing arc connected to the current input vertex is inactive because of the single switching constraint corresponding to it. Hence, the timing window of the input vertex corresponding to its transition type (rise or fall) is set to NULL, indicating that such a transition does not occur. The edge literal is set to the transition type (rise or fall) corresponding to the paired vertex of the current input vertex.
3. *Virtual timing arcs connected to both the current input vertex and its paired vertex are inactive:* In this case, the port corresponding to the input vertex does not assume any transition, i.e., it remains static. Hence, the timing window of the input vertex corresponding to its transition type (rise or fall) is set to NULL, indicating that such a transition does not occur. The static value that the port corresponding to this input

vertex assumes is obtained from the net variable corresponding to the net connected to it. The edge literal of the current input vertex is set to low or high depending on the value of this net variable.

After the attributes of the input vertex are set, depending on whether it is colored or not, its out-edges are colored. Each of these cases is explained below:

- *If the input vertex is colored:* This means that the vertex was never processed earlier. Hence, the two attributes (timing window and edge literal) are assigned values as explained above. The vertices connected to the out-edges of this vertex are colored in order to indicate that they need to be updated.
- *If the input vertex is uncolored:* This means that the vertex was processed earlier. Now, we calculate the two attributes (timing window and edge literal) as explained above and check if they differ from the previous assignments. If so, the attributes are updated and the vertices connected to the out-edges of this vertex are colored in order to indicate that they need to be updated. Otherwise, it is recognized that the input switching scenario for this input has not changed and hence the vertices connected to the out-edges of this vertex are not influenced by this vertex. Hence, they are not touched.

Net receiver vertices A net receiver vertex needs to be processed only if it is colored. The timing window on this vertex is obtained by delaying the timing window on its immediate predecessor by the min-max delay possible on the edge connecting them. Let the timing window on the immediate predecessor be $(t_{pre_{min}}, t_{pre_{max}})$ and the delay on the connecting edge be (d_{min}, d_{max}). Then, the timing window on this vertex is calculated as

$$(t_{min}, t_{max}) = (t_{pre_{min}} + d_{min}, t_{pre_{max}} + d_{max}) \tag{6.6.1}$$

The edge literal of a net receiver vertex is set to the edge literal value of its immediately preceding vertex. After this vertex is processed, the vertices connected to its out-edges are colored to indicate that they need to be updated.

Timing arc receiver vertices A timing arc receiver needs to be processed only if it is colored. The timing arc attached to the timing arc receiver vertex is checked if it can be active as explained in Section 6.6.6. If it is found that

it can be active, then the timing window of this vertex is calculated using equation 6.6.1 used for the processing of net receiver vertices. If the timing arc is temporally inactive but logically valid as per the SAT model from the logic solver, then a conflict clause is added to the logic solver to block this and other similar scenarios. Such a conflict that invalidates a timing arc to propagate timing windows is called a propagation conflict. The procedure to deduce these conflicts is explained in Section 6.6.8.1.

The edge literal value of a timing arc receiver vertex is calculated by using the function of the cell to which it belongs to. As the vertices are processed in a topological order, the values of the edge literals of all the inputs of this cell are already available at the time of this calculation.

After this vertex is processed, the vertices connected to its out-edges are colored to indicate that they need to be updated.

Virtual vertices Virtual vertices are imaginary vertices and hence do not need to be processed.

Consider again the example of the fan-in timing graph depicted in Figure 6.6. Processing the vertices of the timing graph 6.6(a) in the topological order shown in Section 6.6.2 as described in the procedure above, we obtain the timing graph depicted in Figure 6.6(b). The STA arrival time windows are provided just for reference and are not related to the algorithm being explained. By comparing the logical timing windows calculated by the timing solver and the STA timing windows of all the ports, it could be understood how TABLA throws away pessimism in the STA timing windows by calculating timing windows specific to the logically valid solutions from the logic solver. For example, the logical and STA timing windows of the port $I3/Z$ for rise transition are $(1.524, 1.5922)ns$ and $(1.524, 15.5947)ns$ respectively, which clearly shows the gain in accuracy and, as a result, reduction in pessimism, by using TABLA.

6.6.6 Checking if a timing arc can be active

In TABLA, before any timing window is propagated through a timing arc, it is checked if the timing arc can be active. This is done by verifying if the timing window to be propagated has a non-zero width and by checking the satisfiability of the side-input condition of the timing arc with the accurate timing information available. A timing arc is identified to be inactive if at least one of the below two conditions is met:

1. The timing window to be propagated through the timing arc has a zero-width, i.e., if there is no event to be propagated.

2. The switching times of the side-inputs of the timing arc input are in such a way that the side-input condition of the timing arc is violated. This can happen if a side-input can be proven to assume a controlling value during the period at which the timing arc input can switch.

Let us take the example of the timing graph in Figure 6.6(b) to see how a timing arc is verified, if it can be active. Here we see how the timing arc g, connecting the fall vertex of $I3/A$ and the rise vertex of $I3/Z$, is checked for its validity. From Figure 6.6(b), we deduce the following information on ports $I3/A$ and $I3/B$.[6]

$$\text{Port } I3/A \implies \begin{cases} \text{Fall timing window} & = & (1.2409, 1.2410)ns \\ \text{Max. transition time (40-60)} & = & 71.96ps = 0.07196ns \\ \text{Fall switching window} & = & (1.0610, 1.4209)ns \\ \text{Pre-transition value} & = & 1 \\ \text{Post-transition value} & = & 1 \end{cases}$$

$$\text{Port } I3/B \implies \begin{cases} \text{Fall timing window} & = & (1.1963, 1.1966)ns \\ \text{Max. transition time (40-60)} & = & 51.81ps = 0.05181ns \\ \text{Fall switching window} & = & (1.0668, 1.3261)ns \\ \text{Pre-transition value} & = & 1 \\ \text{Post-transition value} & = & 0 \end{cases}$$

For timing arc g to be active, its side-input (port $I3/B$) should be able to assume a non-controlling value on it at any time during the switching window of port $I3/A$. Checking the switching windows of the ports $I3/A$ and $I3/B$ above, it can be observed that they overlap. Clearly, it cannot be proven that port $I3/B$ assumes the controlling logic 0 on it during the entire switching window $[(1.0668, 1.3261)ns]$ of port $I3/A$. Hence, timing arc g is considered active.

Now, to make an example where timing arc g can be proven to be inactive, let us assume that the delay from inputs till the port $I3/A$ is $1ns$ larger and hence, the fall switching window of the port $I3/A$ is $(2.0610, 2.4209)ns$. In this case, the timing data on ports $I3/A$ and $I3/B$ becomes as displayed below:

[6]Switching window is arrival time window expanded on both sides by 2.5 times the maximum 40-60 transition time. The pre- and post-transition states are obtained from the edge literals shown in Figure 6.6(b).

$$
\text{Port } I3/A \implies \begin{cases} \text{Fall timing window} & = & (2.2409, 2.2410)ns \\ \text{Max. transition time (40-60)} & = & 71.96ps = 0.07196ns \\ \text{Fall switching window} & = & (2.0610, 2.4209)ns \\ \text{Pre-transition value} & = & 1 \\ \text{Post-transition value} & = & 1 \end{cases}
$$

$$
\text{Port } I3/B \implies \begin{cases} \text{Fall timing window} & = & (1.1963, 1.1966)ns \\ \text{Max. transition time (40-60)} & = & 51.81ps = 0.05181ns \\ \text{Fall switching window} & = & (1.0668, 1.3261)ns \\ \text{Pre-transition value} & = & 1 \\ \text{Post-transition value} & = & 0 \end{cases}
$$

From the switching information of port $I3/B$, it becomes apparent that it remains static at logic 0 after the time point $1.3261ns$. However, from the fall switching window of port $I3/A$, it can switch only in the time period $(2.0610, 2.4209)ns$. It can be observed that port $I3/B$ assumes the controlling value (logic 0) during this period. Hence, timing arc g is inactive, and the fall timing window on port $I3/A$ is not propagated. Further, if this timing arc is active from the solution provided by the logic solver, then a conflict clause is added to it in order to block this scenario.

6.6.7 Checking if logical timing windows overlap

After the propagation of the input switching vector corresponding to the logically valid solution from the logic solver from inputs (primary inputs or sequential cell outputs) towards outputs (primary outputs or sequential cell inputs) as explained in Section 6.6.5, it is now required to check if the switching windows obtained from the logical timing windows calculated by TABLA overlap in such a way that the crosstalk scenario under consideration is sensitized.

6.6.8 Deduction of conflict clauses

Conflict clauses are clauses added to the logic solver to forbid a solution or a set of solutions of the SAT instance. Deduction of efficient and small conflict clauses is often the key to improve the speed of the logic solver, and in our case, the speed of the whole false noise analysis procedure itself. A powerful conflict clause should include within itself all the obvious solutions that can be forbidden.

As explained in the previous sections, TABLA is a procedure that uses a timing solver to verify all the solutions proposed by the logic solver. Each time a timing solver identifies a case (i.e., a logically valid solution) to be temporally invalid, it adds a conflict clause to the logic solver that forbids this solution and also solutions similar to this, which can already be identified as invalid. The power of the conflict clauses deduced decide the efficiency of the algorithm, as they decide how many logically valid solutions can already be identified as temporally invalid, thus avoiding the necessity to use the timing solver to check them. Hence, conflict analysis plays a very important role in TABLA.

As will be shown in the subsequent sections on propagation conflicts and top-level conflicts, the first step in deducing a conflict clause is to identify the location of the conflict, or to find the set of conflicting vertices. This is followed by the process to identify the reason for the conflict by back-tracking from the conflicting vertices to the input vertices. This procedure is explained step-by-step in the pseudocode listed in Procedure 6.1

Procedure 6.1 BACK_TRACK_TO_FIND_REASON: Algorithm to back track the fan-in cone to find the reason for conflict

Input: The set of conflicting vertices, V_c;
Output: A SAT clause, C to identify the reason for the conflict

```
1: for each vertex v of V_c do
2:     if vertex v is already processed then
3:         continue
4:     if vertex v is an input vertex then
5:         Let e_tarc be the timing arc variable corresponding to v
6:         if e_tarc is active then
7:             Add literal ¬e_tarc to C
8:         else
9:             Add literal e_tarc to C
10:    else
11:        for each in-edge e of vertex v do
12:            if e is a net-branch edge then
13:                Vertex set V = {v}
14:                C = C ∪ BACK_TRACK_TO_FIND_REASON(V)
15:            else
16:                if e is active then
17:                    C = C ∪ BACK_TRACK_TO_FIND_REASON(V)
18:                else
19:                    Add literal e to C
20:    Mark vertex v as processed
21: return C
```

There are two types of conflicts: side-input conflicts and top-level conflicts. They are discussed in detail in separate sub-sections below:

6.6.8.1 Propagation conflicts

A propagation conflict is said to have occurred if a timing window cannot be propagated through a timing arc that is active according to the logically valid SAT model being verified by the timing solver for temporal validity. There are two types of propagation conflicts: side-input conflicts and zero-width conflicts.

Side-input conflicts: A side-input conflict occurs if any timing arc of the circuit that is active as per the logically valid solution from the logic solver cannot actually be valid because one of its side-inputs does not satisfy the side-input condition after the consideration of the timing windows on it.[7] The conflict clause deduced in this case should indicate the logic solver that the input switching vector obtained from the solution provided by it has lead to the case where the timing arc under consideration cannot be active due to its specific side-input that violates the side-input condition. This can be done by including all the conditions that lead to this conflict, i.e., the input timing arcs that, either by being active or inactive, influenced or caused the conflict and the conflicting timing arcs themselves to the conflict clause.

Deduction of a side-input conflict clause involves the subsequent steps:

1. *Finding the location of the conflict:* To find out where the conflict has occurred, the vertices at which the conflict is observed, i.e., the from vertex of the timing arc that is found to be invalid and the vertex corresponding to the side-input port of this timing arc that lead to this conflict are identified. As the timing information on these vertices is in conflict, these vertices are called conflicting vertices.
2. *Identifying the cause of the conflict:* After finding the location of the conflict, the cause for the conflict is identified by back-tracking the decisions made by the logic solver that lead to this situation. This can be done by using the algorithm listed in Procedure 6.1.

[7] The logic solver has missed this constraint because it assumes that any side-input that either has a rise or a fall transition on it always satisfies the side-input condition involving it. Now, it has been identified by the timing solver that this is not true with the accurate timing information it has.

Consider once again the example of the fan-in timing graph depicted in Figure 6.6. Assume that timing arc g is inactive because of the timing on its side-input port $I3/B$. In order to avoid this scenario in future, a conflict clause that blocks it has to be added to the logic solver. As this conflict arises from the timing on the vertices $I3/A_f$ and $I3/B_f$, it is required to back-track through the timing arcs connecting the conflicting vertices to the input vertices to find out the reason for the conflict using the algorithm listed in Procedure 6.1. The order in which the algorithm proceeds in our current example is depicted in Figure 6.7 with step numbers above the vertices processed.

Let C be a variable to store the conflict clause. The algorithm starts back-tracking from the conflicting vertex $I3/B_f$. As it does not encounter any inactive timing arcs, it proceeds till the input vertex B_r. At this point, the literal $\overline{B_r}$ is added to clause C making it $(\overline{B_r})$. Since, this finishes the processing of all the fan-in vertices of $I3/B_f$, the algorithm now proceeds with the second conflicting vertex, i.e., $I3/A_f$.

Vertex $I3/A_f$ has two timing arcs (b and c) connected to it. As timing arc b is inactive, a literal indicating this is added to clause C making it $(\overline{B_r} + b)$. Now, since timing arc c is active, we have to back-track it towards the input vertices. This ends up at the vertex B_r, which was already traversed, and hence need not be considered. This ends the back-tracking process.

Now, the literal corresponding to the conflicting timing arc, i.e., the timing arc g is added to C, making it $(\overline{B_r} + b + \overline{g})$. Adding this conflict clause to the logic solver forbids the current scenario. This can be understood by setting the values of timing arcs B_r and b to logic 1 and logic 0, respectively, which forces the timing arc g to evaluate to logic 0 in order to make the clause satisfiable (see Section 2.6).

Zero-width conflicts: A zero-width conflict is said to have occurred if a timing arc of the circuit that is active as per the logically valid solution from the logic solver cannot actually be active because the input port corresponding to this timing arc has a zero width timing window. The conflict clause deduced in this case indicates the logic solver that the input switching vector obtained from the solution provided by it has lead to the case where the timing arc under consideration cannot be active because it does not have any timing window to be propagated, i.e., the input port of the timing arc does not switch at all.

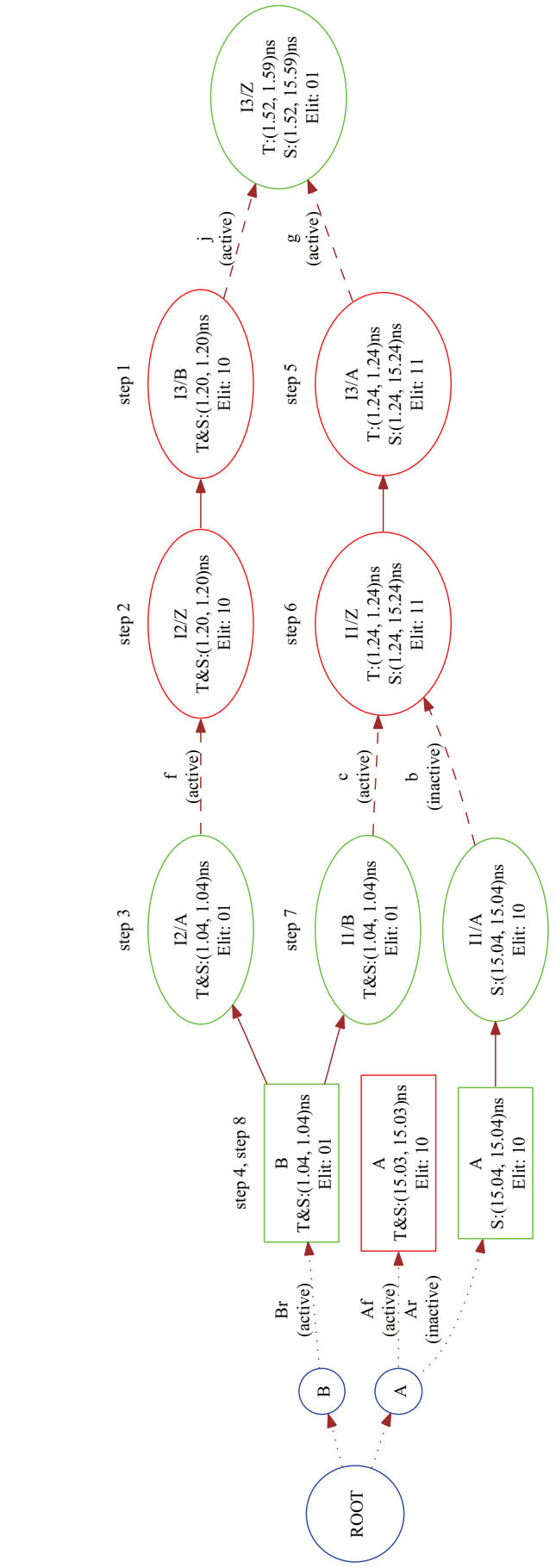


FIGURE 6.7: Back-tracking the fan-in timing graph to find reason for the side-input conflict[abcde]

[a]Abbreviations: MT – Maximum transition time, T – TABLA arrival time window, S – STA arrival time window
[b]To avoid redundancy, edge delay information is omitted. See Figure 6.6(a) for this information.
[c]All the delays provided are specific to the technology used, which is not of any significance in the discussion here.
[d]Vertex notation: circle – virtual vertex, rectangle – input vertex, oval – net receiver or timing arc receiver vertex
[e]Edge notation: dotted line – virtual edge, solid line – net branch edge, dashed line – timing arc edge

An important point to be noted here is that zero-width conflicts need not be added to the logic solver as they are redundant. The reason for this comes directly from the fact that a port has a zero-width timing window on it because all the immediately preceding timing arcs connected to it are inactive (either because of a side-input conflict or from the solution from the logic solver itself). Hence, the zero-width conflict should take care that at least one of these immediately preceding timing arcs has to be active, if the timing arc connected that propagates the window on it has to be active. However, this is already taken care by the timing arc constraints themselves (see Section 6.5.1.2), and hence redundant.

6.6.8.2 Top level conflicts

A top level conflict is said to have occurred when the timing windows of the inputs (primary inputs or flipflop outputs) corresponding to the input switching vector obtained through a logically valid solution from the logic solver, when propagated forward towards the crosstalk cluster (victim and its aggressor ports) do not overlap in such a way that the crosstalk scenario under test is sensitized. In other words, a top level conflict occurs when the logically valid solution proposed by the logic solver is found to be temporally invalid because two or more of the aggressor/victim switching windows do not overlap.

Similar to the side-input conflicts, a top level conflict should also indicate the logic solver that the input switching vector obtained from the solution provided by it has lead to the case where two or more switching windows of the crosstalk cluster (victim and its aggressor ports) do not overlap in such a way that all the aggressors of the crosstalk configuration under test are simultaneously active. This can be done by including all the conditions that lead to this conflict, i.e., the timing arcs that, either by being active or inactive, influenced or caused the conflict and the states of the conflicting aggressor/victim ports themselves to the conflict clause.

There can be several top-level conflicts for each temporally invalid configuration depending on the number of aggressor/victim pairs whose switching windows do not overlap. A top level conflict on an aggressor-victim pair forbids the situation by indicating the logic solver that the current solution (and possibly solutions similar to it) make(s) the aggressor under consideration invalid for inducing crosstalk of the given type on the victim. Similarly, a top level conflict on an aggressor-aggressor pair forbids the situation by indicating the logic solver that the current

solution (and possibly solutions similar to it) avoid the two aggressors under consideration from simultaneously inducing crosstalk on the victim.

6.6.9 Inertial delay model

In an inertial delay model, the output signal of the device exhibits an inertia to change its state. This inertia must be overcome for it to change its state. The inertia value is typically the delay through the cell. If there are any spikes, pulses or glitches at the input that have a width smaller than the inertia of the cell, the output signal value does not change. According to this model, many hypothetical glitches can be filtered and hence the analysis can be made more accurate.

Sections 5.3 discusses the conditions under which a glitch can be generated and propagated with experimental results of an analysis performed on a standard cell library of Infineon. In TABLA, a glitch (or hazard) is assumed to occur only if the switching time difference between two inputs is greater than or equal to zero. Considering the results of the experimental analysis performed, this is a conservative assumption since all the cells are identified to require at least a switching time difference of greater than 59.45 ps to generate a glitch of height greater than 50% V_{dd}.

6.7 Overall flow of TABLA

The overall flow of TABLA is described in various steps below:

1. Find the coupled fan-in cone of the aggressors and/or victim that are to be analyzed.
2. Assign a boolean variable to each of the timing arcs of all the gates in the coupled fan-in cone. For example, consider the circuit in Figure 6.5. It can be observed that each of the timing arcs is assigned a variable.
3. Deduce timing arc, side-input and logic constraints and feed them into the logic solver. This SAT instance is used to perform a conservative logic analysis that considers glitches, because of the underlying unbounded delay model.
4. Add assumptions that activate the timing arc variables corresponding to the aggressors and/or victim under consideration. For example, if it has to be

verified if a 'rise' transition is possible on the Z pin of the inverter in the circuit in Figure 6.5, the assumption clause would be (a), which means that a should be restricted to logic 1 in the final SAT solution, if such a solution exists.

5. Check for the satisfiability of the SAT instance. If the instance is UNSAT, the configuration is invalid even after glitches are considered. Otherwise, proceed to the next step.

6. Check for the temporal sensitizability of the switching scenario proposed by the logic solver. The switching scenario describes what kind of transitions take place at each of the nets of the fan-in cone. To check the temporal validity of this, we start from the input ports and propagate timing windows towards the output ports. During this process, we verify if the timing arcs that are active as per the switching scenario can indeed be active to make the switching windows of the victim and its aggressors overlap. If so, the switching scenario is valid. Otherwise, proceed to the next step.

7. Add a new learnt clause to the logic solver to invalidate the previous solution. Proceed to step 5.

6.8 Verification of TABLA

The real challenge behind the proposal of a novel technique lies in the verification of the validity of all the assumptions on which it is based for practical purposes. In this thesis, a framework that performs dynamic simulations on test circuits is used to justify the assumptions of TABLA, based on which it calculates logical timing windows. Comparison of logical timing windows, which are calculated stage-wise in the STA fashion which dynamic simulations is a very accurate means to comment the correctness of the method and on any deviations they have from the actual switching times.

The process of STA itself is a simplistic approximation of the complex and time-consuming dynamic analysis of circuits using simulations. Although it makes many conservative assumptions that take care of various extreme situations, the timing windows calculated by STA are often non-conservative when compared to the results of the dynamic simulations. This can be explained by the fact that multiple input switching (MIS) and varying side-input signals have more impact on cell delays in the emerging deep-submicron technologies. As STA assumes single

input switching (SIS), the timing windows calculated by it can occasionally differ from the results of dynamic simulations.

The rest of this chapter is organized as follows: Section 6.8.1 explains the framework used to perform dynamic simulations. Section 6.8.2 explains how the timing windows of TABLA and STA are validated by referring to the results of the dynamic simulations. Section 7.3.2 discusses the results of checking the validity of the TABLA and STA timing windows using dynamic simulations on two test circuits.

6.8.1 Dynamic simulation framework

The dynamic simulation framework verifies if the decisions made by TABLA in determining the validity of an input switching vector to trigger a particular crosstalk scenario under consideration is conservative. This can be done by verifying if the switching of the victim and its aggressors happens within the logical timing windows calculated by TABLA for any given crosstalk scenario.

The verification process is invoked after each input switching vector from the logic solver is analyzed for its validity by the timing solver. As explained in Chapter 6, the timing solver finds the crosstalk fan-in cone and then propagates the input switching vector from inputs to the victim and its aggressors by calculating logical timing windows. These timing windows are then used to determine if the victim and its aggressors can simultaneously switch to produce a maximal crosstalk impact. To find out when exactly the victim and its aggressors switch if the input switching vector under consideration is applied to the circuit, the entire fan-in cone is simulated using SPICE and the input switching vector is fed to it. This setup is depicted in Figure 6.8.

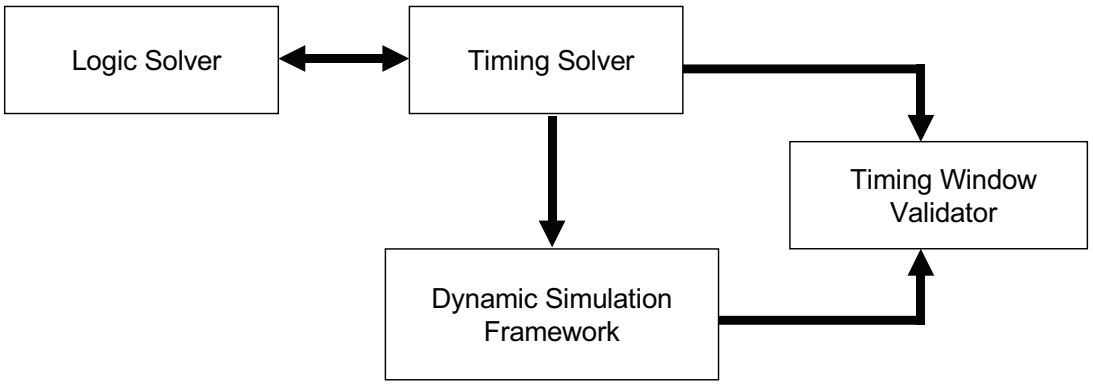


FIGURE 6.8: Setup for verification of TABLA

As the switching times of the victim and its aggressors can vary with the switching times of the inputs, a simulation is performed at each corner of the input timing windows. In other words, if n inputs switch according to the input switching vector, then all the 2^n possible extreme switching conditions are tested. For the timing windows to be valid, the victim and its aggressors should not switch outside these windows for all the possible input switching corners.

6.8.2 Timing window validator

The timing window validator, depicted as a block in Figure 6.8, verifies if the TABLA and STA timing windows are in line with the actual switching times. If the actual switching time, which is obtained from the dynamic simulation framework, is within the switching window being tested, then the switching window is considered to be valid. Otherwise, the amount of deviation from the actual switching time is calculated using the formulae given in equations 6.8.1 and 6.8.2.

Let (t_{early}, t_{late}) be the switching window on a given port for a specific transition (rise or fall). Assume that t_{switch} is the time at which the signal on the port crosses the 50% V_{dd} point, and t_{slew} is the 40–60 slew of the transition, as obtained from the results of the simulation. For the actual switching to happen within the switching window, two conditions have to be satisfied:

$$\begin{aligned} \text{condition A} &: t_{early} < t_{switch} - 2.5 \times t_{slew} \\ \text{condition B} &: t_{late} > t_{switch} + 2.5 \times t_{slew} \end{aligned}$$

If either or both of these conditions are not satisfied, then the switching window is determined to be deviating from the actual switching time. The deviation is calculated as follows:

- **If condition A fails:** Percentage deviation is calculated as

$$d = \frac{[t_{early} - t_{switch} + 2.5 \times t_{slew}] \times 100}{t_{early}} \tag{6.8.1}$$

- **If condition B fails:** Percentage deviation is calculated as

$$d = \frac{[t_{switch} + 2.5 \times t_{slew} - t_{late}] \times 100}{t_{late}} \tag{6.8.2}$$

These formulae make the deviation appear more significant for timing windows near the inputs and less significant for the timing windows near the outputs. However, they serve as a means to understand the approximate amount of deviation caused by factors like multiple input switching and side input conditions different to those used in STA.

As the estimation of deviation for cases with timing windows of small width becomes inaccurate, only cases in which timing windows have a width greater than or equal to 2.5 times the maximum 40–60 transition time possible on the port for the particular transition type are considered.

Chapter 7

Results and Analysis

The false noise reduction techniques proposed in this thesis have been implemented in C++ in an industrial environment into a framework called False Unrealistic Noise Detector and Analyzer (FUNDA) and integrated into a proprietary STA tool.

The crosstalk analysis on various circuits is done for the fall delay scenario, where the victim makes a falling transition while the aggressors are rising. Similar results can be produced for other crosstalk scenarios. For the process of finding the worst realizable aggressor set, a linear model for estimating crosstalk that considers coupling capacitance of an aggressor with its victim as the weight factor is used. In other words, it is assumed that the coupling capacitance of an aggressor with its victim is directly proportional to the crosstalk impact of the aggressor. A more realistic weight factor can be used without impacting the functionality of the techniques proposed. The validity of using a linear model for estimating crosstalk is justified experimentally in [33].

In this chapter, the results of various approaches proposed in this thesis are described in separate sections.

7.1 Aggressor statistics

In this section, we briefly see the statistics of the aggressors on the ISCAS89 benchmark circuits which are used for the analysis of the proposed SAO, AB and TABLA techniques. These circuits have been implemented in a 90nm technology in an industrial environment. No filtering of aggressors based on their coupling

capacitance is done. Figure 7.1 gives the information on the average and maximum number of aggressors on each of the benchmark circuits used for the false noise analysis. The circuits in the figure are ordered in the increasing order of their size. From this histogram, it can be observed that these circuits, despite of being small, are strongly affected by crosstalk. To highlight the efficiency of the proposed methods and maintain highest possible accuracy, no simplification methods like aggressor lumping or aggressor set splitting are employed as in the case of [33, 35, 64, 52].

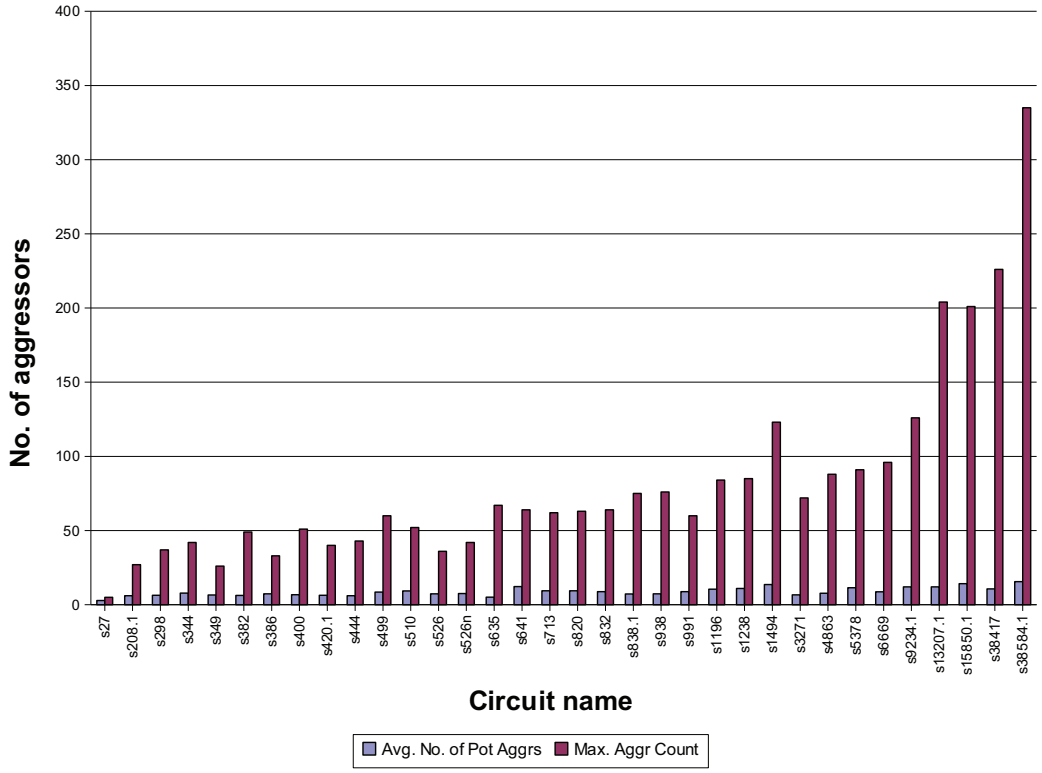


FIGURE 7.1: Aggressor count histogram

7.2 Adaptive Bounding and Simple Aggressor Ordering

The AB and SAO techniques described in Section 4.4 and Section 4.3 have been integrated into the FUNDA framework and tested on several ISCAS89 benchmarks. The experiments were conducted on a SUN sparc Sun–Fire–V890 workstation. All the nets that see cross–coupling with other nets are considered. No aggressors are filtered based on coupling capacitance ratios or absolute values as is done by

some commercial STA tools. Such simplifications are possible and would have a positive speed impact on this approach. Also, no approximate heuristics that trade–off accuracy and speed are used, contrary to [33]. With the application of such heuristics, one can expect a further improvement in speed, of course, with a loss of accuracy.

A comparison of the results of SAT based false noise analysis with and without the use of the proposed SAO and AB techniques has been provided in Table 7.1, Table 7.2 and Table 7.3.

Table 7.1 provides the solvability of various benchmark circuits. Columns 2 to 5 of this table provide the number of nets of each circuit processed versus the number of unsolved nets for each case. A net is considered unsolved if the number of recursive branch and bound calls during the optimization process exceeds an upper limit of 10,000. It can be observed that the application of both SAO and AB (referred as SAT+SAO+AB in Table 7.1) made all the nets solvable for all the tested circuits. Columns 6 to 8 of this Table give the percentage of unsolved nets that are declared infeasible.

Table 7.2 gives the results on the reduction of pessimism in terms of number aggressors and without the application of SAO and AB. Columns 2 to 4 give the average reduction of pessimism of each net. It can be observed that the amount of pessimism reduced per net is larger when SAO and AB are applied. This is a direct consequence of their ability to solve all the nets thus yielding optimal results.

Table 7.3 give the run time of false noise analysis on various circuits with and without the application of SAO and AB. Columns 2 to 4 signify the speed improvement achieved using the proposed methods. The speed has been measured in terms of the CPU time used. As already mentioned, a net is declared as unsolvable by a particular approach if it required more than 10,000 branch and bound (BnB) calls. For each infeasible case the BnB call count is set to be 10,000 and the CPU time to reach this stage is measured. The efficiency of each of the proposed techniques SAO and AB in successfully filtering futile search spaces during the optimization process can clearly be observed.

The results clearly show the efficiency of the proposed techniques in effectively increasing solvability and speed, and reducing pessimism. These techniques can also be applied on the top of the logic constraints approach proposed in [33] with considerable improvement in speed. However, this is not recommended as the

Circuit name	No. of nets						
	proce-ssed	unsolved			% unsolved		
		SAT	SAT+SAO	SAT+SAO +AB	SAT	SAT+SAO	SAT+SAO +AB
s208.1	72	0	0	0	0	0	0
s298	92	2	0	0	2.17	0	0
s349	101	1	0	0	0.99	0	0
s386	111	9	0	0	8.11	0	0
s382	116	1	1	0	0.86	0.86	0
s344	122	3	0	0	2.46	0	0
s420.1	124	2	0	0	1.61	0	0
s444	129	3	0	0	2.33	0	0
s713	155	8	1	0	5.16	0.65	0
s400	155	2	0	0	1.29	0	0
s635	158	2	1	0	1.27	0.63	0
s526	160	5	0	0	3.13	0	0
s499	161	3	1	0	1.86	0.62	0
s641	163	15	1	0	9.2	0.61	0
s526n	167	6	0	0	3.59	0	0
s510	182	15	0	0	8.24	0	0
s820	239	24	3	0	10.04	1.26	0
s938	243	4	1	0	1.65	0.41	0
s838.1	246	5	2	0	2.03	0.81	0
s832	272	25	6	0	9.19	2.21	0
s991	294	4	0	0	1.36	0	0
s1238	354	41	7	0	11.58	1.98	0
s1196	378	36	7	0	9.52	1.85	0
s1494	431	59	27	0	13.69	6.26	0
s3271	775	17	1	0	2.19	0.13	0
s9234.1	909	73	30	0	8.03	3.3	0
s5378	977	84	2	0	8.6	0.2	0
s4863	984	58	5	0	5.89	0.51	0
s6669	1248	72	1	0	5.77	0.08	0
s13207.1	2310	187	57	0	8.1	2.47	0
s15850.1	2415	191	94	0	7.91	3.89	0
s38584.1	5580	599	213	0	10.73	3.82	0
s38417	6763	516	135	0	7.63	2	0

TABLE 7.1: Problem solvability using the SAT, SAO and AB techniques

basic logic constraints approach is either less accurate or slower than the SAT based approach even without the application of SAO and AB.

It can be observed from the tables that the SAO and AB techniques never had a negative impact on solvability, speed or pessimism reduction. The significant improvement of these techniques on solvability and speed can be attributed to their ability to prune infeasible search space of the branch and bound algorithm

Circuit name	% Reduction in aggressor count		
	SAT	SAT+SAO	SAT+SAO+AB
s208.1	22.01	22.01	22.01
s298	19.41	20.49	20.49
s349	15.24	15.59	15.59
s386	18.89	22.70	22.70
s382	23.95	23.94	24.34
s344	20.31	21.15	21.23
s420.1	21.76	22.33	22.33
s444	23.55	24.53	24.53
s400	12.08	13.47	13.72
s713	25.94	26.53	26.53
s635	21.29	21.45	21.71
s526	20.80	21.90	21.90
s499	36.43	36.85	37.27
s641	10.36	12.68	12.88
s526n	22.79	24.40	24.40
s510	29.43	34.05	34.05
s820	23.19	27.49	28.22
s938	19.64	20.08	20.21
s838.1	18.11	18.49	18.81
s832	22.83	25.93	27.11
s991	13.21	13.60	13.60
s1238	22.82	27.54	28.67
s1196	25.59	29.67	30.79
s1494	23.44	27.35	31.01
s3271	19.64	20.34	20.38
s9234.1	16.11	17.55	18.79
s5378	14.19	16.52	16.60
s4863	10.95	12.33	12.48
s6669	8.15	9.27	9.29
s13207.1	16.02	17.63	18.54
s15850.1	12.38	13.28	14.41
s38584.1	11.39	12.76	13.64
s38417	11.01	12.28	12.78

TABLE 7.2: Reduction in crosstalk pessimism in terms of aggressor count using the SAT, SAO and AB techniques

based. The improvement in reduction of pessimism is due to their ability to solve more cases without exceeding the cutoff limit.

Application of simplistic assumptions like aggressor filtration based on their coupling capacitance, splitting of aggressors into subsets, limiting the number of logic correlations or lumping of weak aggressors can always improve the speed of the proposed techniques. As this is not the primary goal of this thesis, an analysis on

Circuit name	CPU time (in sec)			% Improvement
	SAT	SAT+SAO	SAT+SAO+AB	SAT vs. SAT+SAO+AB
s208.1	1.07	0.12	0.07	93.46
s298	3.36	0.24	0.14	95.83
s349	3.52	0.18	0.12	96.59
s386	13.49	0.55	0.29	97.85
s382	2.98	1.98	0.2	93.29
s344	7.69	0.5	0.21	97.27
s420.1	3.47	0.67	0.17	95.1
s444	6.2	0.36	0.25	95.97
s713	8.45	0.91	0.32	96.21
s400	18.42	2.99	0.55	97.01
s635	4.92	2.11	0.38	92.28
s526	13.22	0.74	0.3	97.73
s499	9.43	2.8	0.48	94.91
s641	44.15	5.54	0.79	98.21
s526n	14.64	1.27	0.41	97.2
s510	27.98	2.14	0.61	97.82
s820	48.8	15.23	1.07	97.81
s938	14.87	5.47	0.98	93.41
s838.1	16.86	5.92	0.87	94.84
s832	59.33	17.44	1.33	97.76
s991	18.86	2.67	1.27	93.27
s1238	118	37.05	3.43	97.08
s1196	112	36.33	3.62	96.76
s1494	195	96.98	7.75	96.03
s3271	134	16.99	5.42	95.96
s9234.1	588	290	18.63	96.83
s5378	699	92.71	14.01	98
s4863	506	90.23	18.42	96.36
s6669	952	112	22.08	97.68
s13207.1	3147	1284	103	96.74
s15850.1	3653	1860	148	95.95
s38584.1	24048	9809	661	97.25
s38417	24826	8472	596	97.6

TABLE 7.3: CPU time spent on false noise analysis using the SAT, SAO and AB techniques

how much speed improvement one can gain using these assumptions is not done in this work.

Because of the underlying zero-delay model, this approach of performing a pure logic correlation analysis leads to optimism in the case of presence of glitches. It can be applied on specially designed glitch free circuits, for instance, the domino circuits. As the concepts of SAO and AB, however, are not dependent on the

zero-delay model, they can be applied also on other approaches that consider more accurate circuit delay models.

7.3 Timing Arc Based Logic Analysis

TABLA has been implemented in C++ and integrated into the FUNDA framework. It has been tested on the ISCAS89 benchmark circuits and an industrial design in order to analyze its efficiency and effectiveness in the reduction of false noise. The experiments were run single–threaded on a Sun spark solaris10 workstation with 64GB RAM, 16 CPUs of 1.84 GHz each and a CPU factor of 84.

Table 7.4 show the results of pessimism reduction in terms of number of aggressors on the benchmark circuits. Column 1 and 2 of this table show the percentage reduction in aggressor count for each of these circuits by using the STA switching windows alone, and along with TABLA, respectively for the filtration of false crosstalk. Column 3 of this table shows the improvement provided by TABLA on the top of the STA switching windows. It should be noted that reduction of a small percentage of crosstalk is often sufficient to identify a critical path as a non-violating path.

The improvement of TABLA in reduction of crosstalk pessimism achieved over the STA window based filtration shown in Table 7.4 gives the average improvements for each circuit. As it will be evident from the results on the actual benefit of TABLA in terms of timing slack in Section 7.3.1, the actual improvement for certain nets can be as large as 100%. Such an improvement plays a crucial role in fixing violations on critical paths.

Table 7.5 gives the run time and solvability of TABLA on various benchmark circuits. Columns 1 to 3 give the name of the circuit, number of nets processed and the number of nets that were left insolvable when a cutoff time of 10 seconds per net was used. It can be observed that most of the nets were solved within the cutoff limit.

Columns 4 to 6 of Table 7.5 give the total CPU time spent on processing all the nets of the corresponding circuits using large conflict clauses deduced by considering all the input literals of the fanin cone vs. the powerful conflict clauses deduced by using Procedure 6.1 as explained in Section 6.6.8, and the run time improvement achieved by using powerful conflict deduction mechanism. The CPU times shown

Circuit name	% Reduction in aggressor count		
	using STA windows	using TABLA	Improvement by TABLA
s208.1	24.21	33.89	9.68
s298	31.18	42.98	11.80
s344	37.60	47.30	9.70
s349	39.10	48.60	9.50
s382	39.74	46.38	6.65
s386	33.80	43.08	9.28
s400	35.01	45.72	10.71
s420.1	31.43	40.26	8.83
s444	30.63	42.20	11.57
s499	39.92	52.96	13.04
s510	30.77	45.39	14.62
s526	28.17	40.02	11.84
s526n	30.28	42.57	12.29
s635	32.65	44.28	11.63
s641	40.45	43.84	3.39
s713	33.24	39.65	6.41
s820	29.50	47.00	17.50
s832	29.89	43.50	13.61
s838.1	38.64	45.73	7.09
s938	37.58	45.58	8.00
s991	31.74	35.19	3.45
s1196	32.40	42.94	10.54
s1238	29.94	41.33	11.39
s1494	31.17	44.63	13.46
s3271	30.76	38.81	8.05
s4863	39.36	42.45	3.09
s5378	43.28	48.43	5.15
s6669	37.04	41.82	4.77
s9234.1	45.59	49.58	3.99
s13207.1	44.71	50.23	5.52
s15850.1	51.34	56.01	4.67
s38417	45.00	47.76	2.76
s38584.1	41.51	45.90	4.39

TABLE 7.4: Reduction of pessimism in terms of aggressor count for TABLA

are rounded of to two decimal places, while the exact values are used to calculate the improvements shown in Column 6.

The use of powerful conflict clauses, in general, provides improvement also in terms of solvability and pessimism reduction because of the cut–off time. This is a direct consequence of the fact that TABLA can solve more nets and reduce

Circuit name	No. of nets			CPU time (in mins)		
	processed	unsolved	% unsolved	large conflicts	powerful conflicts	% improvement
s208.1	72	0	0.00	0.02	0.01	61.64
s298	92	0	0.00	0.02	0.02	32.09
s344	122	1	0.82	0.40	0.17	57.15
s349	101	0	0.00	0.32	0.16	48.70
s382	116	1	0.86	0.48	0.30	38.53
s386	111	0	0.00	0.05	0.03	37.37
s400	154	1	0.65	0.63	0.26	58.40
s420.1	124	0	0.00	1.75	0.28	84.05
s444	127	1	0.79	1.20	0.27	77.62
s499	161	20	12.42	13.70	4.78	65.09
s510	181	0	0.00	1.41	0.11	92.06
s526	160	0	0.00	1.57	0.23	85.59
s526n	167	0	0.00	2.01	0.24	88.21
s635	158	3	1.90	7.75	1.53	80.22
s641	163	9	5.52	8.64	2.36	72.65
s713	155	3	1.94	4.82	0.85	82.37
s820	239	1	0.42	4.19	0.65	84.59
s832	272	2	0.74	4.46	0.72	83.92
s838.1	246	11	4.47	7.27	2.03	72.06
s938	242	18	7.44	8.24	2.89	64.87
s991	294	3	1.02	5.86	1.00	82.94
s1196	377	23	6.10	18.72	5.68	69.67
s1238	352	35	9.94	18.70	7.20	61.49
s1494	431	24	5.57	7.65	5.11	33.14
s3271	771	19	2.46	11.53	5.08	55.90
s4863	983	47	4.78	48.41	10.72	77.85
s5378	976	47	4.82	65.31	10.02	84.66
s6669	1245	81	6.51	57.88	15.89	72.55
s9234.1	909	58	6.38	52.45	13.80	73.68
s13207.1	2307	181	7.85	121.58	40.48	66.70
s15850.1	2412	140	5.80	136.54	33.08	75.77
s38417	6744	681	10.10	503.00	153.98	69.39
s38584.1	5570	612	10.99	468.00	146.74	68.65

TABLE 7.5: Solvability and CPU time for TABLA

more pessimism in the given time, with the use of powerful conflict clauses. The solvability and pessimism reduction values provided in Table 7.5 and Table 7.4 correspond to TABLA using the powerful conflict clause generation mechanism.

Net #	Aggr. count			Crosstalk delay (ps)			Improvement	
	unfiltered	STA Window Overlap	TABLA	unfiltered	STA Window Overlap	TABLA	in ps	%
1	5	5	0	91	91	0	91	100
2	17	17	0	96	96	0	96	100
3	7	7	1	116	116	40	76	65
4	9	9	1	144	144	40	105	73
5	13	13	2	145	145	59	85	59
6	9	4	2	148	92	8	84	57
7	9	4	2	154	95	9	87	56
8	9	7	1	192	191	7	184	96
9	9	7	1	203	202	7	195	96
10	16	10	1	251	224	115	109	44
11	16	10	1	254	227	116	111	44
12	12	11	2	258	228	138	89	35
13	16	9	1	264	130	52	78	29
14	16	9	1	270	131	53	78	29
15	16	9	1	270	131	53	78	29

TABLE 7.6: Results of TABLA on selected critical nets of an industrial design.

7.3.1 Crosstalk analysis using TABLA

To demonstrate the benefit of TABLA in terms of crosstalk delay on a real circuit, it was run on selected critical nets of an industrial design with about 65K standard cells. An L–BFGS-B[74] based optimization tool was used to accurately align the aggressors and calculate the crosstalk delay on these nets without any false noise filtering, with aggressor filtering based on switching window overlap and with TABLA. For these experiments unbounded optimization was used, i.e., aggressor alignment was not restricted by timing windows. The bump height produced by an aggressor on its victim was used as aggressor weight in the branch and bound as opposed to the coupling capacitance based weighing mechanism used in [49].

Table 7.6 shows the results of TABLA on 15 nets of the industrial design that gain a maximum benefit from false noise reduction. It highlights the significance of false noise analysis in reduction of crosstalk pessimism for nets that are worst affected by crosstalk. Columns 2–4 give the number of aggressors considered by the different approaches, columns 5–7 show the corresponding crosstalk delays and columns 8 and 9 the improvement that TABLA achieved over switching window overlap. A significant additional amount of crosstalk is filtered by TABLA as compared to the filtration based on STA switching window overlap.

These results clearly indicate the significant benefit in terms of timing slack to be gained by the reduction of false noise. Many state-of-the-art temporal logic analysis methods [64, 52] resort to simplifications such as coarse time slices, interval merging or aggressor lumping to achieve solvability whereas TABLA solves most of the nets with a 10s cutoff limit without such simplifications indicating its efficiency.

7.3.2 Experimental validation of TABLA

As dynamic simulation is a time consuming process, especially for large circuits, only two circuits (s298 and s6669 of the ISCAS 89 benchmarks) are used for this analysis. The results indicate that a considerable number of timing windows calculated by TABLA and STA deviate from the switching times obtained from dynamic simulations. However, it has been observed that most of the cases in which TABLA windows deviate, STA windows also show a deviation. The below table gives a summary of the analysis performed:

Circuit name	s298	s6669
No. of cases tested	286	12540
No. of failed cases TABLA	71	5137
No. of failed cases STA	71	5045

TABLE 7.7: Result summary – Verification of TABLA

As mentioned earlier, the large number of deviations observed in the above analysis is found to be attributable to Multiple Input Switching and difference in side-input conditions for the cells involved. The error histograms for these circuits are shown in Figure 7.2 and Figure 7.3.

As it can be observed from the histograms, for most of the cases the percentage of error is less than 1%, which is very much acceptable. Also, the deviations are a result of the single input switching limitation of STA. As TABLA uses these timing information obtained from STA, these errors are carried. It can also be observed that the number of deviations seen in TABLA are slightly more than those in STA. These cases result from a situation where STA is lucky due to its pessimism.

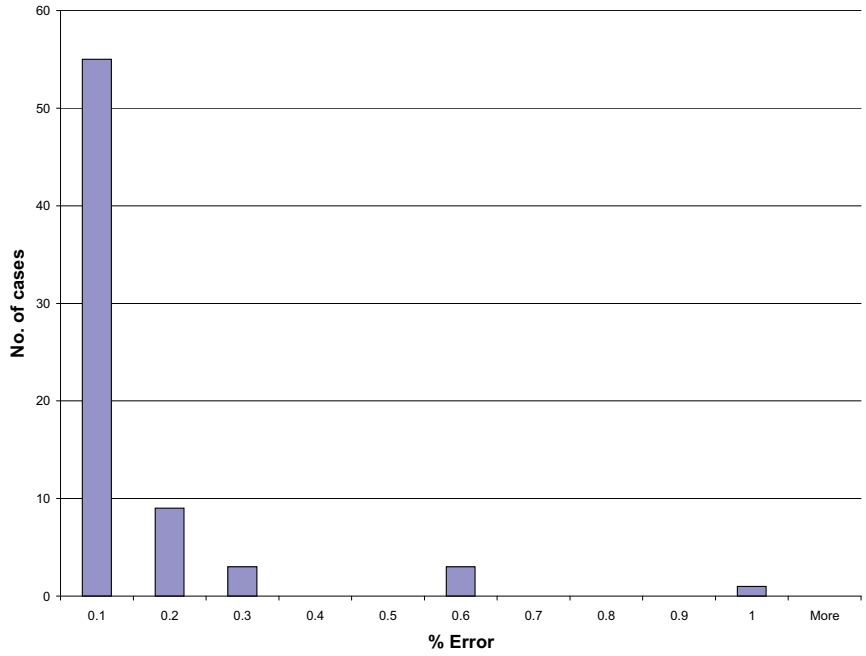


FIGURE 7.2: Verification of TABLA – deviation histogram – circuit s298

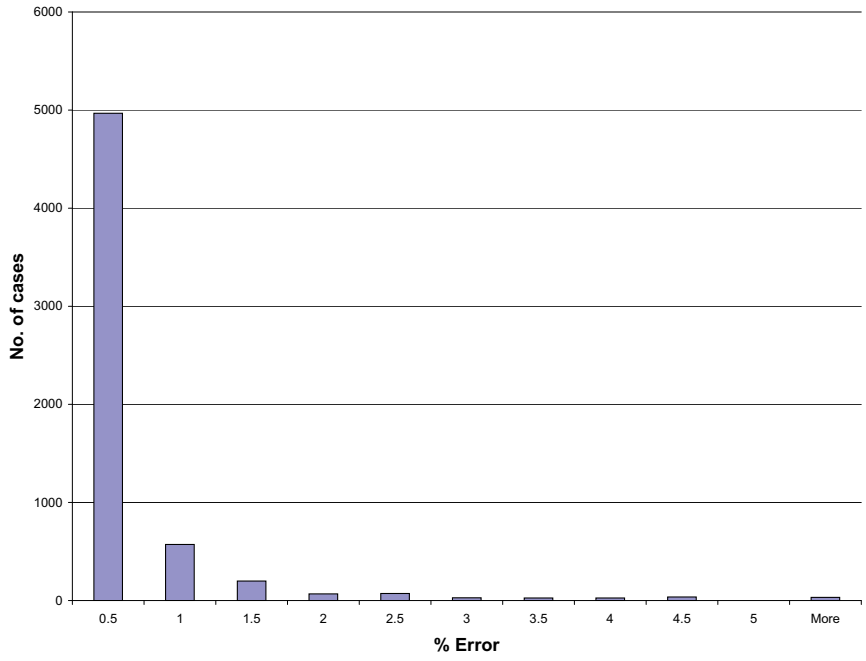


FIGURE 7.3: Verification of TABLA – deviation histogram – circuit s6669

7.4 Analysis of the results

The results of SAO, AB and TABLA on various industrial and benchmark circuits indicate their improvements on the state-of-the-art approaches. A quantitative

comparison of results with other related approaches like [33, 35, 64, 52] is not possible because of uncommon implementations of the tested circuits, which leads to different sets of aggressors, different amounts of crosstalk etc.

The dominance of the proposed approaches can be appreciated by considering several facts described in the below subsections:

7.4.1 SAO and AB

The seminal work on consideration of logic constraints to perform false noise analysis under the zero-delay assumption was done in [33, 35]. These papers explicitly state that they require to selectively apply resolution principle [54], because an uncontrolled application would lead to an explosion of deduced constraints. This leads to an incomplete analysis. However, because of the efficiency of the state-of-the-art SAT solvers, they can efficiently solve this problem optimally without dropping any constraints. An average speed improvement of 96.24% achieved by SAO and AB over the SAT solver based false noise analysis clearly indicates their dominance on the state-of-the-art methods.

7.4.2 TABLA

Temporal logic analysis of circuits to perform a conservative false noise analysis that can handle glitches was done in [64, 52]. These papers explicitly state that they resort approximation methods like aggressor lumping, coarse gridding of time slices, interval merging etc. [52] indicates that this approach can handle aggressor sets of size about 5. Additionally, false noise analysis is performed as a post-processing step in these approaches.

Contrary to these approaches, it can be observed from the results that TABLA was able to solve most of the cases without making any approximate heuristics. It can be observed from Section 7.1 that the circuits have a large number of potential aggressors that have to be processed. With the use of two dedicated solvers for timing and logic, TABLA efficiently blocks infeasible solutions using powerful conflict clauses deduced from the timing solver.

Chapter 8

Conclusions and Outlook

8.1 Summary

This thesis presented three novel techniques called SAO, AB and TABLA to perform an efficient false noise analysis. The first two techniques aid in improving the speed of the false noise analysis and third technique can be used to perform an efficient temporal logic analysis of a circuit.

Chapter 1 and Chapter 2 give an introduction to the false noise problem and various related background topics like circuit delay models, SAT solvers and circuit unrolling technique. The state-of-the-art in false noise analysis is discussed in Chapter 3. Various approaches like the SLI approach [32], logic constraints approach [33, 35], back propagation approach [52] and time-sliced logic approach [64] are discussed in detail. Their performance and accuracy bottle necks are discussed.

Two novel techniques, SAO and AB, to improve the speed of false noise analysis using the zero-delay model are proposed in Chapter 4. SAO is an aggressor ordering technique and AB is a technique to adaptively calculate tighter upperbounds for the branch-and-bound algorithm using the conflict information obtained from the SAT solver. These techniques provide a drastic improvement in speed, solvability and accuracy of the analysis compared to an analysis using SAT solver that does not use these techniques.

The drawbacks of using a non-conservative delay model like the zero-delay model are discussed in Chapter 5. The properties of glitches are studied on an in-house standard cell library. It has been experimentally shown that glitches generate and

propagate through a standard cell only if the input switching time difference and the width of the input glitch, respectively are greater than a certain threshold.

A novel technique called TABLA that uses the accurate min–max timing model that handles glitches implicitly due to its timing arc based approach is proposed and explained in detail in Chapter 6. Because of its conflict driven approach as well as the tight coupling of two domain specific solvers used in TABLA, it handles the false noise problem very efficiently.

The proposed techniques are integrated into a false noise analysis framework called FUNDA, which serves as an add-on to an in-house STA tool. These techniques are tested on several ISCAS89 benchmarks as well as an industrial design. They are demonstrated to achieve great benefit in terms of reduction of crosstalk pessimism and efficiency of false-noise-analysis.

8.2 Major contributions

The goals of this thesis are discussed in Section 1.3. In this section, a list of major contributions of this thesis is given explaining the extent to which these goals are achieved.

1. *Solving the logic correlations problem using SAT solver:* The solution to the logic correlations problem proposed by [33, 35] was infeasible for many cases and require the application of approximate heuristics that find a sub-optimal solution to the problem. In this thesis, this problem was solved using a state-of-the-art SAT solver which significantly improved the solvability of the problem.

2. *SAO and AB:* Although the use of SAT solver was significantly helpful to improve the speed of false noise analysis, it was still inefficient for certain cases. With the use of SAO and AB techniques, a great improvement in speed, solvability and accuracy was achieved.

3. *Experimental study on properties of glitches:* The zero-delay assumption used in [33, 35, 50, 49] is conservative only for specially designed glitch free circuits or domino circuits as explained in [34]. To be conservative, false noise analysis must have the capability to consider glitches. A study of glitches on an in-house standard cell library of Infineon was conducted. The analysis is later used to support the assumptions used in TABLA.

4. *TABLA:* This technique uses timing arcs as basic elements to perform an efficient temporal logic analysis. While many state-of-the-art temporal logic analysis methods [64, 52] resort to simplifications such as coarse time slices, interval merging or aggressor lumping to achieve solvability, because of its efficient handling of timing and logic information, TABLA solves most of the nets with a 10s cut–off limit without such simplifications indicating its efficiency. The assumptions of TABLA have been experimentally validated using a dynamic simulation framework that compares the timing windows of TABLA with the actual switching times obtained from a SPICE simulation. The results indicated that there are slight deviations of the timing windows of TABLA from the actual switching times. However, it is understood that these deviations originate from the single input switching assumption of STA, from which the timing data of TABLA is obtained. As evident from the error histograms provided, most of these deviations are below 5% and hence negligible.

8.3 Outlook

A considerable advance of the state-of-the-art has been achieved with the three novel techniques SAO, AB and TABLA proposed in this thesis. However, there is still a great requirement and scope for further improvements in this field of research. Some of the directions, which are believed to yield improvements, are listed below:

1. *Hybrid approach:* The pure logic analysis under zero–delay assumption is very efficient in terms of run time, especially with the use of techniques like AB and SAO. This can be exploited by applying this analysis only to identify the cases for which a more accurate analysis is necessary. TABLA being costlier than this approach is then performed only on these cases. This can potentially save a lot of time because the detailed analysis can be avoided for most of the simple cases. The effectiveness of such an approach is another issue that has to be verified. A research in this direction can be useful.

2. *Deduction of more powerful conflicts:* As explained in Section 6.6, the efficiency of TABLA depends on the power of the conflict clauses. A powerful conflict clause should be as short as possible, thus blocking a large set of

infeasible solutions. Research towards using problem specific knowledge like the circuit topology or properties of glitches to deduce more powerful conflict clauses can greatly improve the run time of TABLA.

3. *Minimum cost satisfiability:* The problem of minimum cost satisfiability (MinCostSAT) is a hot topic of research today. MinCostSAT is a variation of the Boolean satisfiability problem that minimizes the cost of the satisfying assignment. Attempts have been made to bring the branch & bound algorithm and the SAT mechanism closer in such a way that the branch & bound benefits from the knowledge of the SAT solver [29]. [27] provides methods to convert pseudo-Boolean constraints (linear constraints over boolean variables) which can be solved using a SAT solver. These methods can be used to solve the MinCostSAT problem by iteratively minimizing the cost of the satisfying assignment until no better satisfiable assignment can be found. Testing the efficiency of these approaches to solve the problem of false noise analysis and working on enhancements is a promising area of research.

List of Abbreviations

AB	Adaptive Bounding
BCP	Boolean Constraint Propagation
BGL	Boost Graph Library
BnB	Branch and Bound
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DPLL	Davis-Putnam-Logemann-Loveland
FNA	False Noise Analysis
FUNDA	False Unrealistic Noise Detector and Analyzer
LC	Logic Constraint
LEDA	Library of Efficient Data Types and Algorithms
MIS	Multiple Input Switching
MRAS	Maximum Realizable Aggressor Set
MT	Maximum Transition Time
MWIS	Maximum Weight Independent Set
NP	Non-Polynomial
SAO	Simple Aggressor Ordering
SAT	Boolean Satisfiability
SIS	Single Input Switching

SLI	Simple Logic Implications
STA	Static Timing Analysis
TABLA	Timing Arc Based Logic Analysis
TGA	Tendency Graph Approach

Bibliography

[1] R. Arunachalam, R. D. Blanton, and L. T. Pileggi. False coupling interactions in static timing analysis. In *Design Automation Conference*, 2001.

[2] R. Arunachalam, K. Rajagopal, and L. T. Pileggi. TACO: Timing analysis with coupling. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 266–269, New York, NY, USA, 2000. ACM.

[3] R. I. Bahar, H. Cho, G. D. Hachtel, E. Macii, and F. Somenzi. Timing analysis of combinational circuits using ADDs. In *European Design and Test Conference*, pages 625–629, Mar 1994.

[4] M. Becer, V. Zolotov, R. Panda, A. Grinshpon, I. Algol, R. Levy, and C. Oh. Pessimism reduction in crosstalk noise aware STA. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design*, pages 954–961, Washington, DC, USA, 2005. IEEE Computer Society.

[5] K. P. Belkhale and A. J. Suess. Timing analysis with known false sub graphs. In *International Conference on Computer-Aided Design*, pages 736–740, 1995.

[6] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, London, UK, 1999. Springer-Verlag.

[7] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27:589–607, April 2008.

[8] D. Blaauw, R. Panda, and A. Das. Removing user specified false paths from timing graphs. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 270–273, New York, NY, USA, 2000. ACM.

[9] D. Chai, A. Kondratyev, Y. Ran, K. H. Tseng, Y. Watanabe, and M. Marek-Sadowska. Temporofunctional crosstalk noise analysis. In *Design Automation Conference*, pages 860–863, 2003.

[10] S. Chakraborty, D. I. Dill, and K. Y. Yun. Min-max timing analysis and an application to asynchronous circuits. In *Proceedings of the IEEE*, volume 87, pages 332–346, 1999.

[11] V. Chandramouli and K. A. Sakallah. Modeling the effects of temporal proximity of input transitions on gate propagation delay and transition time. In *DAC '96: Proceedings of the 33rd annual conference on Design automation*, pages 617–622, New York, NY, USA, 1996. ACM.

[12] K. Chandrasekar and M. S. Hsiao. Integration of learning techniques into incremental satisfiability for efficient path-delay fault test generation. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1002–1007, Washington, DC, USA, 2005. IEEE Computer Society.

[13] H. Chen and D. Du B. Lu. Static timing analysis with false paths. In *International Conference on Computer Design*, pages 541–544, 2000.

[14] P. Chen and K. Keutzer. Towards true crosstalk noise analysis. In *ICCAD '99: Proceedings of the 1999 IEEE/ACM International Conference on Computer-Aided Design*, pages 132–138, Piscataway, NJ, USA, 1999. IEEE Press.

[15] P. Chen, Y. Kukimoto, and K. Keutzer. Refining switching window by time slots for crosstalk noise calculation. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pages 583–586, New York, NY, USA, 2002. ACM.

[16] K. Chung, T. Kim, and C. L. Liu. A non-zero delay model for glitch analysis in logic circuits. In *IEEE Midwest Symposium on Circuits and Systems*, volume 3, pages 1244–1247, 2000.

[17] K. Chung, T. Kim, and C. L. Liu. G-vector: A new model for glitch analysis in logic circuits. In *Journal of VLSI Signal Processing Systems*, volume 27, pages 235–251, 2001.

[18] S. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[19] D. Das, K. Killpack, C. Kashyap, A. Jas, and H. Zhou. Pessimism reduction in coupling-aware static timing analysis using timing and logic filtering. In *Asia and South Pacific Design Automation Conference*, pages 486–491, 2008.

[20] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[21] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[22] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Computation of floating mode delay in combinational circuits: Theory and algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12:1913–1923, 1993.

[23] D. H. Du, S. H. Yen, and S. Ghanta. On the general false path problem in timing analysis. In *DAC '89: Proceedings of the 26th ACM/IEEE conference on Design automation*, pages 555–560, New York, NY, USA, 1989. ACM.

[24] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *International Conference on Theory and Applications of Satisfiability Testing*, 2005.

[25] N. Eén and N. Sörensson. An extensible SAT solver. In *International Conference on Theory and Applications of Satisfiability Testing*, 2003.

[26] N. Eén and N. Sörensson. MiniSat–A SAT solver with conflict–clause minimization. In *International Conference on Theory and Applications of Satisfiability Testing*, 2005.

[27] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[28] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. In *Symposium on Switching Circuit Theory and Logical Design*, volume 00, pages 111–120, 1964.

[29] Z. Fu and S. Malik. Solving the minimum-cost satisfiability problem using SAT based branch-and-bound search. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, pages 852–859, New York, NY, USA, 2006. ACM.

[30] R. Gandikota, K. Chopra, D. Blaauw, D. Sylvester, M. Becer, and J. Geada. Victim alignment in crosstalk aware timing analysis. In *ICCAD '07: Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 698–704, Piscataway, NJ, USA, 2007. IEEE Press.

[31] S. Garg, S. Tata, and R. Arunachalam. Static transition probability analysis under uncertainty. In *International Conference on Computer Design*, pages 380–386, 2004.

[32] A. Glebov, S. Gavrilov, D. Blaauw, S. Sirichotiyakul, C. Oh, and V. Zolotov. False noise analysis using logic implications. In *International Conference on Computer-Aided Design*, pages 515–521, 2001.

[33] A. Glebov, S. Gavrilov, R. Soloviev, V. Zolotov, M. R. Becer, C. Oh, and R. Panda. Delay noise pessimism reduction by logic correlations. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design*, pages 160–167, Washington, DC, USA, 2004. IEEE Computer Society.

[34] A. Glebov, S. Gavrilov, V. Zolotov, C. Oh, R. Panda, and M. Becer. False-noise analysis for domino circuits. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 784–789, 2004.

[35] A. Glebov, S. Gavrilov, V. Zolotov, R. Panda, C. Oh, and D. Blaauw. False–noise analysis using resolution method. In *International Symposium on Quality Electronic Design*, 2002.

[36] L. Guerra e Silva, J. Marques-Silva, L. M. Silveira, and K. A. Sakallah. Satisfiability models and algorithms for circuit delay computation. *ACM Trans. Des. Autom. Electron. Syst.*, 7(1):137–158, 2002.

[37] Robert B. Hitchcock, Sr. Timing verification and the timing analysis program. In *DAC '82: Proceedings of the 19th conference on Design automation*, pages 594–604, Piscataway, NJ, USA, 1982. IEEE Press.

[38] S. Huang, T. Parng, and J. Shyu. Timed boolean calculus and its applications in timing analysis. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 13, pages 318–337, 1994.

[39] K. J. Keller, H. Takahashi, K. K. Saluja, and Y. Takamatsu. On reducing the target fault list of crosstalk-induced delay faults in synchronous sequential circuits. In *Proceedings of the IEEE International Test Conference 2001*, pages 568–577, Washington, DC, USA, 2001. IEEE Computer Society.

[40] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli. Digital sensitivity: predicting signal interaction using functional analysis. In *International Conference on Computer-Aided Design*, pages 536–541, 1996.

[41] W. K. C. Lam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Circuit delay models and their exact computation using Timed Boolean Functions. In *DAC '93: Proceedings of the 30th international conference on Design automation*, pages 128–134, New York, NY, USA, 1993. ACM.

[42] R. Li, A. Shey, and M. Laudes. Incorporating logic exclusivity (LE) constraints in noise analysis using gain guided backtracking method. In *ICCAD '08: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 783–789, Piscataway, NJ, USA, 2008. IEEE Press.

[43] M. K. Michael and S. Tragoudas. Generation of hazard identification functions. In *International Symposium on Quality Electronic Design*, pages 419–424, 2003.

[44] A. Mondal and P. P. Chakrabarti. Reasoning about timing behavior of digital circuits using symbolic event propagation and temporal logic. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 25, pages 1793–1814, 2006.

[45] A. Mondal, P. P. Chakrabarti, and C. R. Mandal. A new approach to timing analysis using event propagation and temporal logic. In *Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 1198–1203, 2004.

[46] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, 2001.

[47] M. Orshansky and A. Bandyopadhyay. Fast statistical timing analysis handling arbitrary delay correlations. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 337–342, New York, NY, USA, 2004. ACM.

[48] M. Palla. Methods for detection of invalid crosstalk aggressor configurations. Master's thesis, Darmstadt University of Technology, June 2005.

[49] M. Palla, J. Bargfrede, K. Koch, W. Anheier, and R. Drechsler. Adaptive branch and bound using SAT to estimate false crosstalk. In *ISQED '08: Proceedings of the 9th international symposium on Quality Electronic Design*, pages 508–513, Washington, DC, USA, 2008. IEEE Computer Society.

[50] M. Palla, K. Koch, J. Bargfrede, M. Glesner, and W. Anheier. Reduction of crosstalk pessimism using Tendency Graph Approach. In *International Conference on Computer Design*, 2006.

[51] M. Palla, K. Koch, J. Bargfrede, M. Glesner, and W. Anheier. False noise analysis using branch & bound and SAT. In *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU 2008)*, 2008.

[52] Y. Ran, A. Kondratyev, K. Tseng, Y. Watanabe, and M. Marek-Sadowska. Eliminating false positives in crosstalk noise analysis. In *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, volume 24, pages 1406–1419, 2005.

[53] Y. Ran, A. Kondratyev, Y. Watanabe, and M. Marek-Sadowska. Eliminating false positives in crosstalk noise analysis. In *Design, Automation and Test in Europe Conference and Exhibition*, volume 2, 2004.

[54] J. A. Robinson. A machine-oriented logic based on the resolution principle. In *Journal of the ACM*, volume 12, pages 23–41, 1965.

[55] S. Roy, P. P. Chakrabarti, and P. Dasgupta. Bounded delay timing analysis using boolean satisfiability. In *VLSID '07: Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference*, pages 295–302, Washington, DC, USA, 2007. IEEE Computer Society.

[56] S. S. Sapatnekar. On the chicken-and-egg problem of determining the effect of crosstalk on delay. In *Proc. IEEE 8th Topical Meeting on Electrical Performance of Electronic Packaging (EPEP-99)*, pages 245–248, 1999.

[57] D. Satyendra. Crosstalk delay analysis in very deep-submicron VLSI circuits. Master's thesis, Southern Methodist University, May 2004.

[58] S. Shrivastava, R. Pratap, H. Parameswaran, and M. Verma. Crosstalk analysis using reconvergence correlation. In *Asia and South Pacific Design Automation Conference*, pages 79–83, 2006.

[59] J. P. Marques Silva and K. A. Sakallah. GRASP—a new search algorithm for satisfiability. In *ICCAD '96: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.

[60] D. Sinha and H. Zhou. A unified framework for statistical timing analysis with coupling and multiple input switching. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design*, pages 837–843, Washington, DC, USA, 2005. IEEE Computer Society.

[61] D. Tadesse, D. Sheffield, E. Lenge, R. I. Bahar, and J. Grodstein. Accurate timing analysis using SAT and pattern-dependent delay models. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 1018–1023, 2007.

[62] G. Theodoridis, S. Theoharis, D. Soudris, and C. Goutis. Switching activity estimation under real-gate delay using timed Boolean functions. In *IEE proceedings. Computers and digital techniques*, volume 147, pages 444–450, 2000.

[63] B. Thudi and D. Blaauw. Non-iterative switching window computation for delay noise. In *Design Automation Conference*, pages 390–395, 2003.

[64] K. Tseng and M. Horowitz. False coupling exploration in timing analysis. In *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, volume 24, pages 1795–1805, 2005.

[65] K. Tseng and V. Kariat. Static noise analysis with noise windows. In *Design Automation Conference*, pages 864–868, 2003.

[66] J. Vygen. Slack in static timing analysis. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 25, pages 1876–1885, 2006.

[67] T. Xiao and M. Marek-Sadowska. Functional correlation analysis in crosstalk induced critical paths identification. In *Design Automation Conference*, pages 653–656, 2001.

[68] T. Xiao and M. Marek-Sadowska. Using temporal and functional information in crosstalk aware static timing analysis. In *Journal on VLSI Design, Hindawi Publishing House*, volume 15, pages 647–666, 2002.

[69] L. Xun and M. C. Papaefthymiou. A statistical model of input glitch propagation and its application in power macromodeling. In *MWSCAS-2002: Midwest Symposium on Circuits and Systems*, 2002.

[70] J. Yang, J. Kim, J. Choi, M. Yoo, and J. Kong. Elimination of false aggressors using the functional relationship for full-chip crosstalk analysis. In *International Symposium on Quality Electronic Design*, pages 344–347, 2003.

[71] L. Zhang. *Searching for truth: techniques for satisfiability of boolean formulas*. PhD thesis, Princeton, NJ, USA, 2003. Adviser-Malik,, Sharad.

[72] L. Zhang, Y. Hu, and C. C. Chen. Block based statistical timing analysis with extended canonical timing model. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 250–253, New York, NY, USA, 2005. ACM.

[73] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *International Conference on Computer Aided Verification*, 2002.

[74] C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal. L–BFGS–B: A limited memory FORTRAN code for solving bound constrained optimization problems. In *Tech. Report, NAM-11, EECS Department, Northwestern University*, 1995.

[75] V. Zolotov, D. Blaauw, S. Sirichotiyakul, M. Becer, C. Oh, R. Panda, A. Grinshpon, and R. Levy. Noise propagation and failure criteria for VLSI designs. In *International Conference on Computer-Aided Design*, pages 587–594, 2002.

Curriculum Vitae

Murthy Palla

Born on 12th June 1980 in Ganapavaram, India

1995−1997	Intermediate Vignan Co-operative Jr. College, India
1997−2001	Study of Bachelor of Technology Electronics and Communication Engineering Nagarjuna University, India
2001	Bachelor of Technology
2002−2005	Study of Master of Science Information and Communications Engineering Darmstadt University of Technology, Darmstadt, Germany
2005	Master of Science
2005−2009	Doctoral Research Assistant Electrical Engineering and Information Technology University of Bremen, Bremen, Germany
2009	Doctor of Engineering