**Université de Montréal**

# Waveform Narrowing: A Constraint-Based Framework for Timing Analysis

par
Maroun Kassab

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade
de Philosophiæ Doctor (Ph. D.)
en informatique

Mars, 2002

QA
76
U54
2002
1.025

Université de Montréal

# Waveform Narrowing: A Constraint-Based Framework for Timing Analysis

by
Maroun Kassab

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

A thesis submitted to the Faculty of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science

March, 2002

**Université de Montréal**
Faculté des études supérieures

Cette thèse intitulée:

# Waveform Narrowing: A Constraint-Based Framework for Timing Analysis

présentée par

Maroun Kassab

a été évaluée par un jury composé des personnes suivantes:

|  |  |
|---|---|
| Marc Feeley | président-rapporteur |
| Eduard Cerny | directeur de recherche |
| François-Raymond Boyer | membre du jury |
| John Hayes | examinateur externe |
| François Perron | représentant du doyen de la FES |

# Résumé

**Mots Clés**

Faux chemins, ITGE, Vérification formelle, Circuits logiques séquentiels, Vérification des aspects temporels.

La vérification des aspects temporels des circuits logiques synchrones est NP-dur à cause du problème des faux chemins dans les circuits combinatoires. Les méthodes de vérification basées exclusivement sur les propriétés topologiques du circuit sont trop pessimistes, et les méthodes exactes ont une complexité exponentielle de temps d'exécution au pire cas. Nous présentons dans cette thèse une méthode basée sur la satisfaction des contraintes, ayant une complexité linéaire d'espace, et une complexité de temps qui peut être quasi-linéaire, $n \times \log(n)$, quadratique ou exponentielle, tout dépendant du niveau de précision requis. La méthode consiste à modéliser le circuit, les conditions de fonctionnement, et les contraintes temporelles par un système de contraintes qui est consistant si et seulement si les contraintes temporelles ne sont pas respectées. Le système de contraintes contient un ensemble de variables, prenant valeurs de leurs domaines respectifs, et un ensemble d'opérateurs de contraintes dont chacun opère sur un sous-ensemble des variables. Le système est résolu partiellement en appliquant répétitivement les opérateurs de contraintes, éliminant des valeurs qui ne font partie d'aucune solution, jusqu'à ce qu'il atteigne le point fixe, où ce n'est plus possible de changer les domaines des variables. Lorsque la résolution résulte en une variable ayant un domaine vide, on déduit que le système est incohérent et par conséquent les contraintes temporelles sont respectées; autrement, on ne peut rien conclure.

La méthode conduit à des résultats faux négatifs dans le cas où la résolution partielle se terminerait avec des domaines non vides, et que le système est en réalité incohérent. Nous avons développé deux méthodes polynomiales pour réduire ce pessimisme:

- Le concept des *dominateurs temporels*, des nœuds clés dans le circuit, ayant des domaines qui peuvent être réduits suite à des conditions nécessaires déduites en examinant la fonction globale du circuit;

- Une procédure de corrélation spatiale qui permet de renforcer partiellement la fonction globale du circuit sur les nœuds ayant des branchements convergents, en restreignant leurs domaines à des ondes qui se stabilisent à la valeur logique 0, puis à 1, et en combinant les résultats.

Nous avons aussi développé une procédure de décision qui permet de trouver une solution du système de contraintes (vecteur de test qui viole les contraintes temporelles) ou de prouver que le système est effectivement incohérent.

Lorsque appliquée sur les benchmarks standards ISCAS'85, la méthode a trouvé les bornes supérieures des délais des circuits correspondant aux délais exacts. En plus, à l'exception du circuit c6288, la procédure de décision a trouvé des vecteurs de test pour tous les circuits avec un nombre remarquablement restreint de retours en arrière.

On a rendu l'implantation plus complète et robuste afin de pouvoir tester la méthode sur des circuits industriels. Le vérificateur résultant fut testé sur un circuit industriel de 122 milles portes logiques, et a prouvé qu'au pire cas, la marge de sécurité de la contrainte d'établissement des bascules est en fait 17.459% du temps de cycle, comparée à 8.74% déduite par une analyse topologique.

# Abstract

**Keywords**

Timing Verification, False Path Problem, VLSI, Formal Verification, Synchronous Sequential Circuits.

Verifying the timing properties of VLSI circuits is NP-Hard due to the false path problem, and considering just the topological delay of the circuit is too conservative and may result in unnecessary redesign efforts. We present in this thesis a timing verification method based on *Waveform Narrowing*. The method has linear space complexity, and has controllable time complexity that can be virtually linear, $n \times \log(n)$, quadratic, or exponential, depending on the required level of accuracy. The method consists of modeling the circuit, the timing constraints, and the operating conditions as a constraint system that is consistent if and only if the timing constraints are violated. The constraint system is composed of a finite set of variables that take values from their respective domains, and a set of relational constraint operators, each operating on a subset of the variables. The system is solved partially by repeatedly applying the constraints, removing from the domains values that are not part of any solution, until the greatest fixpoint is reached. If we end up with empty domains, we conclude that the timing constraints are satisfied; otherwise, no conclusion can be drawn.

The method results in false negative answers when we end up with non-empty domains, and yet the constraint system has no solution. To reduce this pessimism we developed two polynomial techniques:

The *Timing dominators* concept that determines key circuit nets for which the domains can be narrowed as a consequence of necessary conditions deduced from the global circuit function;

*Spatial correlation* procedure that enforces partially the global circuit function by restricting the domains of selected reconvergent fan-outs to waveforms stabilizing at 0 and 1, and then merging the results.

Also, we developed a case analysis procedure able to find a test vector or to prove that no violation is possible.

When tested on ISCAS'85 benchmarks, the method found tight upper bounds that correspond to exact circuit delays for all circuits. Moreover, except for c6288, the case analysis procedure found test vectors for all circuits with a remarkably low number of backtracks!

We extended the method by implementing capabilities necessary to verify industrial designs. The resulting timing verifier was tested successfully on a 122K-gate industrial circuit, and proved that its relative safe margin is 17.459% of the clock cycle instead of 8.74% reported by topological analysis.

# CONTENTS

**List of Figures** .......... xi

**List of Tables** .......... xviii

**List of Symbols, Keywords, and Abbreviations** .......... xix

**Acknowledgments** .......... xxiii

**Preface** .......... xxv

**Chapter I Introduction** .......... 1
- 1.1 Design Flow .......... 2
- 1.2 Post-Fabrication Testing .......... 3
- 1.3 Timing Verification .......... 4
  - 1.3.1 Simplified Model of a Synchronous Sequential Circuit .......... 5
  - 1.3.2 Constraints of Memory Elements .......... 5
  - 1.3.3 Verification of a Flip-Flop's Constraints .......... 8
  - 1.3.4 Synchronizing Clocks .......... 9
  - 1.3.5 Component Delays .......... 10
    - 1.3.5.1 Factors Affecting Component Delays .......... 10
    - 1.3.5.2 Delay Models and Components Correlation .......... 12
  - 1.3.6 Maximal Delay of a Combinational Circuit .......... 12
  - 1.3.7 Minimal Delay of a Combinational Circuit .......... 15
  - 1.3.8 The False Path Problem .......... 16
- 1.4 Original Contributions of this Thesis.......... 16
- 1.5 Plan of the Thesis .......... 18

**Chapter II Literature Review** .......... 19
- 2.1 Methods Based on Path Enumeration .......... 21
  - 2.1.1 Static Sensitization .......... 23
  - 2.1.2 The Brand-Iyengar Criterion .......... 24
  - 2.1.3 Co-Static Sensitization .......... 25
  - 2.1.4 Viable Sensitization .......... 25
  - 2.1.5 Floating Mode Sensitization .......... 26
- 2.2 Reduction to a Test Generation Problem .......... 26
- 2.3 Mixed Boolean-Linear Programming .......... 28
- 2.4 Hierarchical Method .......... 29
- 2.5 Method Based on Constraint Satisfaction .......... 30
- 2.6 Summary .......... 31

**Chapter III Method Based on Waveform Narrowing .................... 34**
3.1 Overview of the WN Method .................... 34
3.1.1 Inconsistency Property .................... 34
3.2 Formalism for Timing Verification .................... 35
3.2.1 Abstract Waveforms .................... 35
3.2.2 Abstract Signals Domain .................... 36
3.2.3 Other Domains .................... 36
3.2.4 Timing Verification .................... 37
3.3 Combinational Circuit Example .................... 39
3.4 Sequential Circuit Example .................... 43
3.5 Relational Constraints .................... 50
3.5.1 Delay Constraints .................... 54
3.5.2 Delayless Gate Constraints .................... 58
3.5.2.1 Buffer Gate (Identity) .................... 59
3.5.2.2 Inverter Gate (NOT) .................... 59
3.5.2.3 AND Gate .................... 60
3.5.2.4 OR - NAND - NOR Gates Constraints .................... 73
3.5.2.5 Exclusive OR Gate (XOR) .................... 74
3.6 Combinational Circuit Delay Verification / Basic Modeling .................... 79
3.7 Reduction of Pessimism .................... 84
3.7.1 Static Learning .................... 84
3.7.2 Spatial Correlation .................... 85
3.7.3 Global Timing Implications .................... 88
3.7.4 Static Timing Dominators .................... 91
3.7.5 Dynamic Timing Dominators .................... 92
3.8 Case Analysis .................... 95
3.8.1 Controllability Measure .................... 98
3.8.2 Requirement Propagation / Back Trace Procedure .................... 100
3.8.3 Case Analysis Procedure .................... 105
3.9 Experimental Results .................... 111
3.10 Conclusions .................... 115

**Chapter IV Advanced Modeling .................... 116**
4.1 Clock Definition Formalism .................... 118
4.2 Modeling Delays .................... 120
4.2.1 Normalized Delay .................... 121
4.2.2 Delay Correlation Networks .................... 123
4.3 Modeling Building Blocks .................... 124
4.3.1 Domains .................... 125
4.3.2 Primitive Constraints .................... 126
4.4 Cell Library Modeling .................... 132
4.5 Handling of Combinational Loops .................... 133
4.6 Timing Verifier .................... 134
4.7 Experimental Results .................... 137
4.8 Conclusions .................... 140

**Chapter V Conclusions** .......................................... **141**
5.1 Comparison with Other Methods .......................................... 143
5.2 A Disadvantage .......................................... 144
5.3 Original Contributions of this Thesis .......................................... 144
5.4 Future Work .......................................... 146

**References** .......................................... **148**

**Appendix A (Algorithms)** .......................................... **xxvi**
A.1 evaluateConstraintSystem() .......................................... xxvii
A.2 getOrderedReconvergentFanouts(gate) .......................................... xxix
A.3 doSpatialCorrelation(order, s) .......................................... xxx
A.4 correlateReconvergentFanouts(gate_set) .......................................... xxxii
A.5 getTimingDominators( output ) .......................................... xxxiv
A.6 evaluateConstraintSystemTD( output ) .......................................... xxxv

**Appendix B (Cell Modeling)** .......................................... **xxxvi**
B.1 Hard Primitives Used to Model Cell Libraries. .......................................... xxxvii
B.2 Two Input AND Gate Cell Model Example .......................................... xliv
B.3 FLIP-FLOP Gate Cell Model Example .......................................... xlvi

# LIST OF FIGURES

**Figure 1**
Design flow of integrated circuits. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .2

**Figure 2**
Stuck-at-1 fault testing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

**Figure 3**
Delay fault testing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

**Figure 4**
Post-fabrication tests. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

**Figure 5**
Sequential synchronous circuit. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5

**Figure 6**
Rising edge-triggered flip-flop. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6

**Figure 7**
An incorrect 1 bit counter:
Hold time constraint of the flip-flop is violated . . . . . . . . . . . . . . . . . . . . . . . . . . .6

**Figure 8**
High level sensitive latch. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7

**Figure 9**
Hold time constraint violation prevented by the use of a transparent latch. . . . . . . .7

**Figure 10**
Unfolding clock cycles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8

**Figure 11**
Sequential circuit controlled by two harmonically related clocks. . . . . . . . . . . . . . .9

**Figure 12**
Delay of an inverter. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

**Figure 13**
P (left) and N type transistors. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

**Figure 14**
Silicon wafer. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

**Figure 15**
Examples of input stimuli. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .14

**Figure 16**
Difference between floating and sequences of vectors maximal delays. . . . . . . . .14

**Figure 17**
A circuit with false paths (numbers on gates represent delays):
when B stabilizes to 0 (1), G stabilizes to 1 after 5 (3) time units.
Its floating delay is 5, whereas its topological delay is 7 . . . . . . . . . . . . . . . . . . . .16

**Figure 18**
A path statically sensitized. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .23

**Figure 19**
A true path of length 3 identified as false by static sensitization criterion. . . . . . .24

**Figure 20**
A false path of length 6 identified as true by co-static sensitization (a = 0). . . . . .25

**Figure 21**
A circuit to be transformed to the equivalent normal form. . . . . . . . . . . . . . . . . . .27

**Figure 22**
Removing fan-outs results in a tree circuit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .27

**Figure 23**
The result of pushing back the inverters. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .27

**Figure 24**
Normal form representation.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .28

**Figure 25**
(a): Abstract signal domain.
(B): Clock edge domain.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .35

**Figure 26**
A delayless AND gate constraint operator.
(a) Initial domains.
(b) Effect on the gate output Y.
(c) Effect on the input A . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .38

**Figure 27**
(a): Combinational circuit with a false path.
(b): Combinational circuit example: step 0. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .39

**Figure 28**
Combinational circuit example: step 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .40

**Figure 29**
Combinational circuit example: step 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .40

**Figure 30**
Combinational circuit example: step 3. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .40

**Figure 31**
Combinational circuit example: step 4. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .41

**Figure 32**
Combinational circuit example: step 5. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .41

**Figure 33**
Combinational circuit example: step 6. . . . . . . . . . 42

**Figure 34**
Combinational circuit example: step 7. . . . . . . . . . 42

**Figure 35**
Frequency divider. . . . . . . . . . 43

**Figure 36**
Sequential circuit example: step 0. . . . . . . . . . 44

**Figure 37**
Sequential circuit example: step 1. . . . . . . . . . 45

**Figure 38**
Sequential circuit example: step 2. . . . . . . . . . 45

**Figure 39**
Sequential circuit example: step 3. . . . . . . . . . 45

**Figure 40**
Sequential circuit example: step 4. . . . . . . . . . 45

**Figure 41**
Sequential circuit example: step 5. . . . . . . . . . 45

**Figure 42**
Sequential circuit example: step 6. . . . . . . . . . 46

**Figure 43**
Sequential circuit example: step 7. . . . . . . . . . 46

**Figure 44**
Sequential circuit example: step 8. . . . . . . . . . 46

**Figure 45**
Sequential circuit example: step 9. . . . . . . . . . 47

**Figure 46**
Sequential circuit example: step 10. . . . . . . . . . 47

**Figure 47**
Sequential circuit example: step 11. . . . . . . . . . 47

**Figure 48**
Sequential circuit example: step 12. . . . . . . . . . 47

**Figure 49**
Sequential circuit example: step 13. . . . . . . . . . 47

**Figure 50**
Sequential circuit example: step 14. . . . . . . . . . 48

**Figure 51**
Sequential circuit example: step 15. . . . . . . . . . 48

**Figure 52**
Sequential example: step 16. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .48

**Figure 53**
Sequential example: step 17. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .48

**Figure 54**
Sequential example: step 18. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .48

**Figure 55**
Sequential example: step 19. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .49

**Figure 56**
Sequential example: step 20. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .49

**Figure 57**
Waveform narrowing for steps 6 through 20. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .49

**Figure 58**
An incompatible value. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .53

**Figure 59**
A delayed binary waveform. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .55

**Figure 60**
Clock delay constraint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .55

**Figure 61**
Data delay constraint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .56

**Figure 62**
Effect on Y.$w_0$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .57

**Figure 63**
Effect on A.$w_0$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .58

**Figure 64**
Effect on D. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .58

**Figure 65**
Buffer Constraint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .59

**Figure 66**
NOT Constraint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .59

**Figure 67**
ANDing two binary waveforms. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .60

**Figure 68**
AND Constraint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .60

**Figure 69**
AND Constraint ( A.$w_0$, B.$w_0$, Y.$w_0$ ) - effect on Y.$w_0$. . . . . . . . . . . . . . . . . . . . . .62

**Figure 70**
AND Constraint ( A.$w_0$, B.$w_0$, Y.$w_0$ ) - effect on A.$w_0$ and B.$w_0$. . . . . . . . . . . . . .62

**Figure 71**
AND Constraint ( A.$w_0$, B.$w_1$, Y.$w_0$ ) - effect on Y.$w_0$. . . . . . . . . . . . . . . . . . . . . .64

**Figure 72**
AND Constraint ( A.$w_0$, B.$w_1$, Y.$w_0$ ) - effect on A.$w_0$. . . . . . . . . . . . . . . . . . . . . .65

**Figure 73**
AND Constraint ( A.$w_0$, B.$w_1$, Y.$w_0$ ) - effect on B.$w_1$. . . . . . . . . . . . . . . . . . . . . .66

**Figure 74**
AND Constraint ( A.$w_1$, B.$w_1$, Y.$w_1$ ) - effect on Y.$w_1$. . . . . . . . . . . . . . . . . . . . . .68

**Figure 75**
AND Constraint ( A.$w_1$, B.$w_1$, Y.$w_1$ ) - effect on A.$w_1$. . . . . . . . . . . . . . . . . . . . . .69

**Figure 76**
AND Constraint Example. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .71

**Figure 77**
AND Constraint Example: $C_{AND}(0|_{-\infty}^{8}, 0|_{-\infty}^{6}, 0|_{7}^{8})$ . . . . . . . . . . . . . . . . . . . . . . . .71

**Figure 78**
AND Constraint Example: $C_{AND}(0|_{-\infty}^{8}, 1|_{-\infty}^{5}, 0|_{7}^{8})$ . . . . . . . . . . . . . . . . . . . . . . . .71

**Figure 79**
AND Constraint Example: $C_{AND}(1|_{-\infty}^{10}, 0|_{-\infty}^{6}, 0|_{7}^{8})$ . . . . . . . . . . . . . . . . . . . . . .72

**Figure 80**
AND Constraint Example: $C_{AND}(1|_{-\infty}^{10}, 1|_{-\infty}^{5}, 1|_{9}^{10})$ . . . . . . . . . . . . . . . . . . . . . .72

**Figure 81**
Domains contents after the AND constraint is applied. . . . . . . . . . . . . . . . . . . . . . .72

**Figure 82**
XORing two binary waveforms. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .74

**Figure 83**
XOR Constraint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .74

**Figure 84**
XOR Constraint ( A.$w_0$, B.$w_0$, Y.$w_0$ ) - effect on Y.$w_0$. . . . . . . . . . . . . . . . . . . . . .76

**Figure 85**
XOR Constraint ( A.$w_0$, B.$w_0$, Y.$w_0$ ) - effect on A.$w_0$. . . . . . . . . . . . . . . . . . . . . .77

**Figure 86**
NAND-4 constraint model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .79

**Figure 87**
False path circuit of Hrapcenko. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .82

**Figure 88**
Static Learning on the circuit c17. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .84

**Figure 89**
Learned constraint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .85

**Figure 90**
Spatial correlation on reconvergent fan-out $e_3$.
(a) Domains before spatial correlation
(b) Domains when $e_3$ is restricted to waveforms stabilizing at 1
(c) Domains when $e_3$ is restricted to waveforms stabilizing at 0
(d) Domains after spatial correlation on $e_3$ (union of (b) and (c) ). . . . . . . . . . . . . .86

**Figure 91**
16-bit carry skip adder. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .89

**Figure 92**
Carry out of 16-bit carry skip adder.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .90

**Figure 93**
Circuit path from X to C7.
NAND gates are represented as separate delays
and delayless NAND constraints. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .90

**Figure 94**
Timing dominators of output_57 of c1908-ALL-GATES. . . . . . . . . . . . . . . . . . . . . . .95

**Figure 95**
(1) Requirements impossible to satisfy.
(2) Decision tree when b is chosen the last.
(3) Decision tree when b is chosen the first.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .97

**Figure 96**
Requirement propagation.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .97

**Figure 97**
Controllability measure for circuit c17. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .100

**Figure 98**
Back Trace example.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .101

**Figure 99**
Timing check ($\xi$, $T$, 61) for a carry skip circuit. . . . . . . . . . . . . . . . . . . . . . . . . .105

**Figure 100**
Topological partitioning for decision making. . . . . . . . . . . . . . . . . . . . . . . . . . . .106

**Figure 101**
Topological partitioning for phase 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .108

**Figure 102**
Clocking scheme example. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .119

**Figure 103**
Normalized delay.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .121

**Figure 104**
Gate interval delay, function of J and N. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .121

**Figure 105**
Four point slack computation. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .122

**Figure 106**
Slack for multiple data paths. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122

**Figure 107**
Simple delay correlation network. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123

**Figure 108**
One-dimensional position dependent delay correlation network. . . . . . . . . . . . . . 123

**Figure 109**
Two-dimensional position dependent delay correlation network. . . . . . . . . . . . . . 124

**Figure 110**
Two-to-one multiplexer.
(1) AND - OR logic model.
(2) OR-AND logic model.
(3) Timing model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131

**Figure 111**
Combinational cell modeling example.
(1) Cell logic function.
(2) Cell constraint model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

**Figure 112**
Flip-Flop cell modeling example.
(1) Cell logic function.
(2) Cell constraint model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133

**Figure 113**
Combinational loop example.
(a) C and E are part of a combinational loop.
(b) Loop aware constraint for E.
(c) Loop aware constraint for C. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134

**Figure 114**
Timing Verifier Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134

**Figure 115**
Experimental time complexity of the basic WN method on ISCAS'85 . . . . . . xxviii

**Figure 116**
Experimental time complexity of doSpatialCorrelation on ISCAS'85 . . . . . . . xxxi

**Figure 117**
Experimental time complexity of each system evaluation of the
spatial correlation on ISCAS'85 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxiii

# List of Tables

**Table I**
Floating algebra . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

**Table II**
Sensitization criteria for delayless two-input AND gate.. . . . . . . . . . . . . . . . . . . .22

**Table III**
Substitution for Controlling / Non-Controlling /
Controlled / Non-Controlled values.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .73

**Table IV**
Results on the ISCAS'85 benchmark suite. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112

**Table V**
Summary of the results on the ISCAS'85
NOR-GATES benchmark suite . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113

**Table VI**
False path elimination summary. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114

**Table VII**
Results on an industrial benchmark for best-case delay annotation. . . . . . . . . . . .138

**Table VIII**
Results on an industrial benchmark for worst-case delay annotation. . . . . . . . . . 139

# LIST OF SYMBOLS, KEYWORDS, AND ABBREVIATIONS

**AND**
The logic And gate function.

**AS**
The space of *Abstract Signals*.

**ATPG**
Automatic test pattern generation.

**AW**
The space of *Abstract Waveforms*.

**BDD**
Binary decision diagram.

**BW**
The space of Binary Waveforms.

**C++**
An object oriented programming language.

**CMOS**
Complementary metal-oxide semiconductor.

**CPU**
Central processing unit.

**CRYSTAL**
A transistor-level timing verifier developed by Ousterhout.

**EDA**
Electronic design automation.

**FAN**
An algorithm for test pattern generation developed by Fujiwara and Shimono.

**FLIP-FLOP**
An edge-triggered memory element.

**HDL**
Hardware description language.

**IC**
Integrated circuit.

**ISCAS**
International Symposium on Circuits and Systems.

**ITGE**
Intégration à très grande échelle.

**Kbytes**
1024 bytes.

**Kgates**
Thousand gates.

**Mbytes**
1048576 bytes.

**MIPS**
Million instructions per second.

**NAND**
The Inverted logic And gate function.

**ND**
*Normalized Delay*.

**NMOS**
N-type metal-oxide semiconductor transistor.

**NOR**
The Inverted logic Or gate function.

**NOT**
The logic Inverter gate function.

**NP-Complete**
Non deterministic polynomial complete.

**NP-hard**
Non deterministic polynomial hard.

**OR**
The logic Or gate function.

**PERT**
A method based on graph topological ordering for analyzing critical paths.

**PLL**
Phase locked loop.

**PMOS**
P-type metal-oxide semiconductor transistor.

**PODEM**
An algorithm for test pattern generation developed by Goel.

**RAM**
Random access memory.

**RealIntervals**
The space of real intervals.

**SCALD**
Structured Computer-Aided Logic Design, a computer program containing a gate-level timing verifier developed by McWilliams.

**SCOPE**
A method for calculating the controllability measure.

**SDF**
Standard Delay Format, an industry standard for component delay representation.

**SPICE**
A computer program for circuit simulation.

**TV**
A transistor-level timing verifier developed by Jouppi.

**Verilog**
A hardware description language.

**VHDL**
A hardware description language.

**VLSI**
Very large scale integration.

**WN**
*Waveform Narrowing*.

**XNOR**
The Inverted logic Exclusive-Or gate function.

**XOR**
The logic Exclusive-Or gate function.

for Berthe, Alice, and Lory-Ann

# ACKNOWLEDGMENTS

This work is part of the Power and Timing Verification Research Project in LASSO laboratory at the Université de Montréal, supervised by Professor Eduard Cerny.

First, I would like to express my gratitude to my research advisor Professor Eduard Cerny, without whom this work could not have come to accomplishment. Professor Cerny is an extraordinary wealth of knowledge and energy, exemplary in his dedication to making projects successful. Apart from his leadership and the strict work progress verification, he provided the necessary industrial relations, directing the project toward applications that are applied to the real industrial world. He provided me with invaluable technical advice through deep discussions of the subject matter of my research, enlarging substantially my knowledge in Electronic Design Automation and mathematics. Not to mention that he provided me with the opportunity to participate in many technical conferences.

- Thanks to all those who participated in the Power and Timing Verification Project. Especially Jindrich Zejda, M. Sidi Aourid, and Aisha Bensalem.
- Nortel Networks provided office space and access to proprietary information and machines to conduct experiments. More importantly, Thomas Krodel, Carol Bachelu, and Spiros Boucouris, experts in the field of timing verification, provided precious technical assistance and help in an exceptionally friendly professional environment!
- Nortel Networks and Functionality Inc. provided us with InCore data structure software that includes Verilog and SDF parsers.
- Parts of the experiments were done on workstations on loan from the Canadian Microelectronics Corp.

- Funding for the project was provided partly by Natural Sciences and Engineering Research Council of Canada (NSERC) - Nortel North America Cooperative R&D Grant No. 169880.

Additional funding for my studies was provided by the following organizations:

- Ministère de l'Éducation du Québec - aide financière aux études;
- Université de Montréal, entente de financement.
- NSERC Industrial Postgraduate Scholarship in association with Nortel Networks.

Last but not least, I would like to thank all my friends, family, and parents, especially Berthe who, despite all the difficulties we encountered, provided all the needed support.

# PREFACE

Electronic Design Automation (EDA) is a multi-billion-dollar industry that is struggling to keep pace with the increasing capability of silicon technology that is still following the Moore's Law. A common cause of increasing complexity is design methodology that is not uniform. EDA tools need to handle excessive number of special cases that make it impossible to hard-code their user interface. Most design tools are command line utilities, or shared libraries integrated using scripting extension languages like Tcl, which makes the learning curve steep and makes the design methodology open. In fact, too much of the capabilities of the tools are exposed to designers who can easily abuse them. The complexity of the user interface makes the EDA industry adopt methods that are easy to code, validate, integrate, and use. For instance, although too pessimistic, topological analysis is still the basis of many commercial timing verifiers because of its simplicity. The result is an industry that is expanding horizontally; little effort is available for advanced research.

Academia is, however, more relaxed, and one can still dedicate effort to investigate a new method, hoping to provide new scientific solutions that EDA and the scientific community in general may benefit from. This thesis presents the results of the investigation we conducted on a new timing verification method based on waveform narrowing, inspired by logic constraint programming and interval arithmetic.

It was a great challenge to write this thesis as it deals with complex subjects related to optimization techniques, electrical engineering, and to computer science. We did our best to make the thesis highly illustrated, self-contained, requiring no in-depth knowledge in any of the subjects mentioned, however, mathematical maturity is a prerequisite for the understanding of this work.

# CHAPTER I INTRODUCTION

Advances in very large scale integration (VLSI) technology have made digital electronics essential for a wide spread of applications. In the 1960's, circuits were fabricated using millimetric scale discrete components such as transistors, resistors and capacitors. Recently, Motorola succeeded in fabricating a microprocessor device using a new 0.1 micron process technology [1]. Current standard fabrication processes use 0.35, 0.25, 0.18, and 0.11 microns for the transistor size, enabling the integration of a complete system containing millions of transistors on a single silicon chip.

The increasing complexity led the designers' community to formalize the design flow, and made essential the use of computer assisted design and verification tools. Simulation based verification of gigantic designs has become practically impossible due to the exponential state explosion problem (e.g. A processor with 16 registers, 32 bits each, has $2^{512}$ possible states). Consequently, formal verification, an approach exploiting mathematical techniques in order to prove certain functional properties without having to enumerate all existing states, has emerged. Numerous problems are subject to formal verification, such as the equivalence of two state machines or two logic circuits, proof that a system stays alive (no deadlock), or verifying that timing constraints of memory elements are satisfied in a sequential circuit.

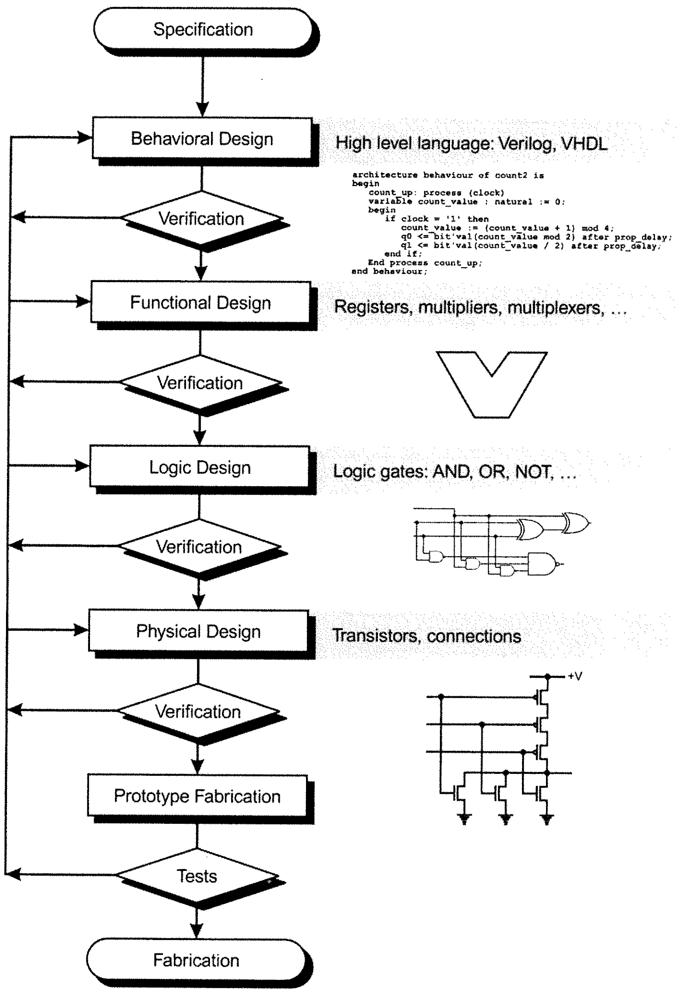
The subject of this thesis is the development of a "static" timing verifier for synchronous sequential circuits. The attribute "static" is used instead of formal to comply with timing verification literature.

The remainder of this chapter introduces the principal subject as well as related subjects that are necessary to the understanding of the subsequent parts of the thesis.

## 1.1 Design Flow

Figure 1 shows a simplified design flow of integrated circuits. Starting from specifications (function, response time, power dissipation, etc.), the design process follows a descending hierarchical approach. Each level of abstraction has to be verified as functionally equivalent to the one above it, and, that different constraints such as power dissipation and response time are satisfied.

Obviously, the process is iterative, and the purpose of design and verification tools is to minimize the number of iterations, especially at lower levels, where the fabrication of prototypes is very expensive.



```
architecture behaviour of count2 is
begin
   count_up: process (clock)
   variable count_value : natural := 0;
   begin
      if clock = '1' then
         count_value := (count_value + 1) mod 4;
         q0 <= bit'val(count_value mod 2) after prop_delay;
         q1 <= bit'val(count_value / 2) after prop_delay;
      end if;
   End process count_up;
end behaviour;
```

***Figure 1***
*Design flow of integrated circuits.*

### Different Levels of Abstraction:

**Behavioral Level:** The system is defined using a high-level hardware description language (HDL) such as Verilog or VHDL.

**Register Transfer Level:** The description of the system in terms of functional blocks (multipliers, multiplexers, registers, etc.) This description is obtained using a compiler for the behavioral level, i.e., the high-level language, or by manual translation.

**Logic Gate Level:** The definition of the system is in terms of logic gates and memory elements (AND, OR, NOT, flip-flops, etc.). The logic synthesizer generates this description from the Register Transfer Level.

**Transistor Level:** The lowest level in the hierarchy. This level is an assembly of transistors and metal connectors. A technology mapper is used to rewrite the logic gate level in terms of gates from a well-characterized library (e.g., TGC1000 from Texas Instruments). Then, placement and routing tools are applied to generate the physical layout.

## 1.2 Post-Fabrication Testing

The production of integrated circuits is a complex lithographic process that yields a success ratio lower than 1. The success rate varies depending on the process technology and the chip area.

Two major models of fabrication defects are used:

**Stuck-at Fault**: the output of a logic gate g is said to be stuck-at-1 (0) if its logic level remains 1 (0) regardless of its input levels. For example, the output of an AND gate is 1, even if one of the inputs is 0. To detect such a defect, a test vector must be applied to the circuit inputs, propagating the fault to at least one of the outputs. Fig. 2 shows a simple example.

**Delay Fault**: a gate can have an excessive delay due to a fabrication defect. A common cause of such a defect is an excessive interconnect resistance caused by a very thin open that still permits conduction by tunneling effect. To detect such a defect, two test vectors must be applied successively to the circuit inputs in order to trig-

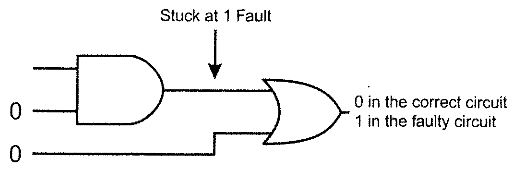


**Figure 2**
*Stuck-at-1 fault testing.*

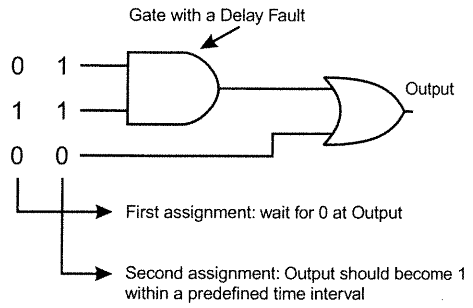

Gate with a Delay Fault
0 1
1 1
0 0
Output
First assignment: wait for 0 at Output
Second assignment: Output should become 1 within a predefined time interval

**Figure 3**
*Delay fault testing.*

ger a transition that propagates to a circuit output through the defect, as shown in the example of Fig. 3.

Finally, for a specific design, a set of test vectors is generated to detect possible defects. Manufactured chips are tested before delivery as indicated in Fig. 4. Automatic Test Pattern Generation (ATPG) is a domain that has been widely studied in the last 30 years [31-55]. Contributions in this domain offer a rich set of heuristics [32,33,35] that help resolve the satisfiability of a Boolean formula.

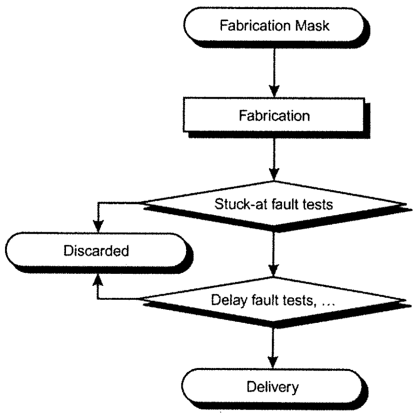


**Figure 4**
*Post-fabrication tests.*

## 1.3 Timing Verification

Verifying timing properties of a synchronous sequential circuit consists of determining whether it functions properly at a certain clock frequency (e.g. 500 MHz), or determining the maximal safe clock frequency. This verification can be applied to any level of abstraction. The highest precision is obtained only after the actual physical design is available, where the different circuit components are precisely characterized. Before reaching this stage, approximate delays are assumed for each circuit component and interconnect.

Note that the violation of timing constraints of circuit components depends on the clock frequency, functioning semantics, and circuit topology. The following section defines default semantics for synchronous sequential circuits, adopted throughout this thesis, unless stated otherwise.

## 1.3.1 Simplified Model of a Synchronous Sequential Circuit

A model of a combinational logic circuit is a directed acyclic graph. Nodes represent logic gates and each arc represents a connection (connects the output of a gate to the input of another gate). The circuit input (output) terminals are represented by nodes with no incident (exiting) arcs.

A synchronous sequential circuit is an implementation of a state machine. The state is stored in a register and the next state is calculated by a combinational logic circuit as shown in Fig. 5. The state register consists of memory elements for which the memorizing action is triggered by the system clock.

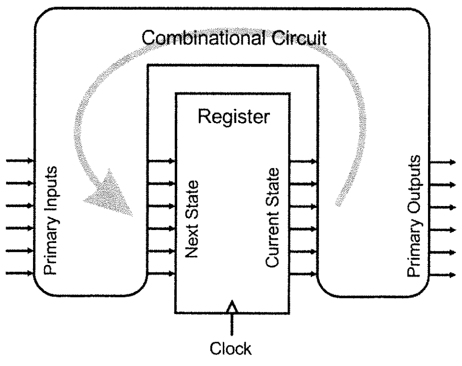


***Figure 5***
*Sequential synchronous circuit.*

The default operation semantics assumes that a new state is calculated every clock cycle. Therefore, the clock period must be long enough for the combinational circuit to finish its calculations and for the memory elements to store the new state at the next active clock edge.

## 1.3.2 Constraints of Memory Elements

Two types of memory elements are used: Edge-sensitive flip-flops, and level-sensitive transparent latches. The most common ones used for state memorization are the flip-flops sensitive to the rising edge of the clock (transition from 0 to 1, see Fig. 6). When the logic level of the clock CLK changes from 0 to 1, the logic level of input D is memorized and becomes present at the output Q. Like every physical device, the memorization operation is not instantaneous; its timing behavior is characterized by three parameters [12]:

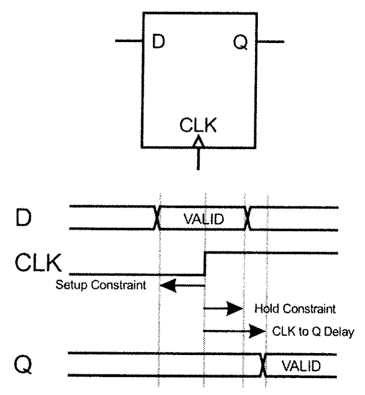


*Figure 6*
*Rising-edge-triggered flip-flop.*

1) **CLK to Q Delay**: This delay represents the time necessary for the memorized logic level to get to the output Q.

2, 3)**Setup Time ($t_S$), Hold Time ($t_H$) Constraints**: They represent the timing constraints on the logic level of D to be memorized correctly. D must be valid (stable) in the interval $[t_R - t_S, t_R + t_H]$ where $t_R$ is the arrival time of the rising edge of CLK.

The timing verifier must check that setup and hold constraints are satisfied for each flip-flop.

Note that the setup time constraint tends to be violated when higher clock frequency is used, whereas, hold time constraint tends to be violated due to the circuit topology, resulting in a system that does not function properly with any given clock frequency. In the circuit of Fig. 7, the flip-flop does not memorize correctly the new state computed by the inverter due to the premature disappearance of the previous value. Violation of the hold time constraint is due to very short paths in the combinational circuit, and to clock skew (non-simultaneous arrival time of the clock edge at different flip-flops).

Flip-flops sensitive to the falling clock edge (sensitive to the transition from 1 to 0 of the clock) are defined in a similar manner.

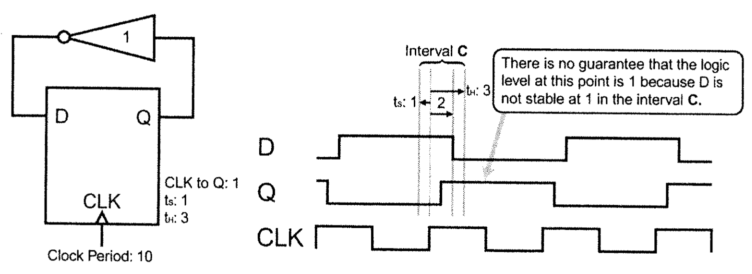

1
D
Q
CLK
CLK to Q: 1
tS: 1
tH: 3
Clock Period: 10
Interval C
tS: 1
2
tH: 3
There is no guarantee that the logic level at this point is 1 because D is not stable at 1 in the interval C.
D
Q
CLK

*Figure 7*
*An incorrect 1 bit counter: Hold time constraint of the flip-flop is violated.*

The behavior of the level sensitive latch (transparent latch) is slightly different. Consider the latch sensitive to the high level (see Fig. 8), when the clock C is at 1, the output follows the input (Q = D). When C falls to 0, Q keeps its last value. Its temporal behavior is characterized by four parameters [12]:

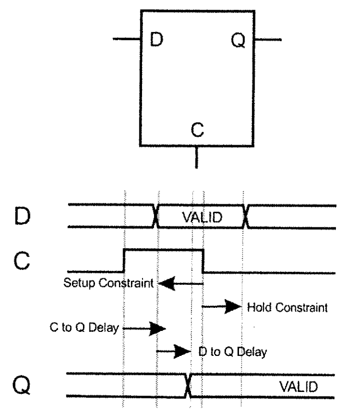


***Figure 8***
*High level sensitive latch.*

1) **C to Q Delay**: Time before Q starts to follow D, after C rises to 1.

2) **D to Q Delay**: When C = 1, it is the events (logic level change) propagation time from D to Q.

3, 4) **Setup Time, Hold Time Constraints**: The same definition as for the falling edge flip-flop (the edge of the clock after which the latch keeps its last value).

The low level sensitive latch is defined similarly.

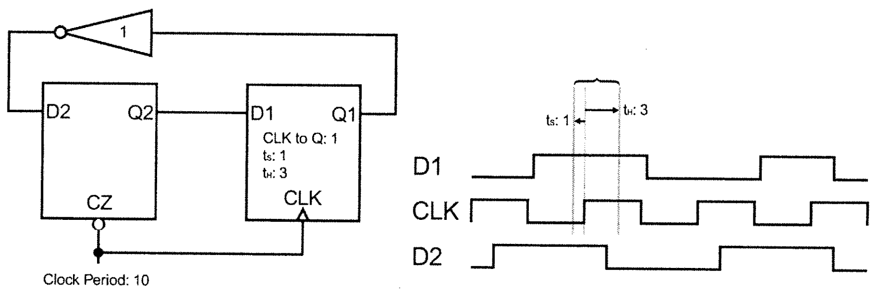


***Figure 9***
*Hold time constraint violation prevented by the use of a transparent latch.*

Transparent latches are frequently used in industrial circuits to avoid violation of hold time constraints of edge-triggered flip-flops. Fig. 9 shows a revised version of the circuit in Fig. 7: the low level sensitive latch prevents the premature disappearance of the previous state by delaying it to the falling edge of the clock. When CLK rises to 1, the value of Q1 becomes D1 after 1 time unit (CLK-to-Q1), then D2 becomes $\overline{Q1}$ after 1 time unit, but Q2 keeps its old value until CZ falls back to 0. Of course, the latch timing constraints must also be satisfied.

### 1.3.3 Verification of a Flip-Flop's Constraints

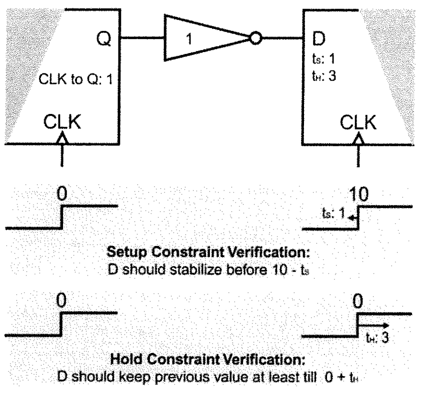

Q
CLK to Q: 1
CLK
1
D
$t_S$: 1
$t_H$: 3
CLK
0
10
$t_S$: 1
Setup Constraint Verification:
D should stabilize before 10 - $t_S$
0
0
$t_H$: 3
Hold Constraint Verification:
D should keep previous value at least till 0 + $t_H$

*Figure 10*
*Unfolding clock cycles.*

According to the operation semantics of the synchronous sequential circuit, the new value that each flip-flop stores has been computed from values stored on the previous clock cycle. To verify the constraints of a flip-flop, the combinational sub-circuit that does the computation is observed and the flip-flop setup and hold time constraints are studied separately. Considering the example in Fig. 7, only one flip-flop is present and the sub-circuit that does the computation is the inverter. The flip-flop is broken down in two parts: one receives the data computed by the combinational circuit, and the other injects a value into that same circuit (see Fig. 10). For a period of 10 units of time, data is injected at time 0, and sampled at D at time 10.

Verifying the setup time constraint comes down to verifying that D stabilizes before 10 - $t_S$, meaning that it is the maximal delay of the combinational circuit that comes into play. On the other hand, to check the hold time constraint, assuming that the flip-flop injects a value at time 0, it must be verified that the previous value of D does not disappear before $t = 0 + t_H$. In this case, it is the minimal delay of the combinational circuit that has to be considered.

In summary, the problem of deciding whether the setup or hold constraints are satisfied comes down to comparing the maximal and minimal circuit delays to certain given values.

Chapter 2 explains the major methodologies for computing the maximal combinational circuit delay proposed in literature.

### 1.3.4 Synchronizing Clocks

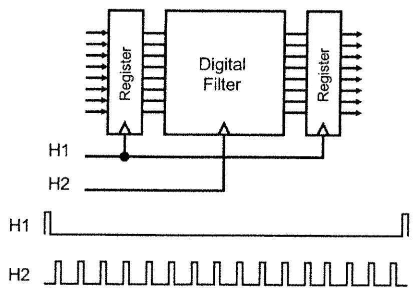


***Figure 11***
*Sequential circuit controlled by two harmonically related clocks.*

In Section 1.3.1, the synchronous sequential circuit was introduced without exploring thoroughly the subject of synchronizing clocks. Designers of synchronous circuits face many practical and conceptual problems:

**Maximal Fan-out Capability**: In a real circuit, it is not possible for a single gate output to drive an indefinite number of gates without risking loss of data. Therefore, the clock signal is distributed to a number of buffers; each can then send it to a number of memory elements and / or other buffers.

**Clock Skew**: To avoid violations of the hold time constraints, delays are inserted in the clock tree to minimize the time window in which the memory elements are activated.

**Multiple Clocks**: For certain applications, such as the digital filter shown in Fig. 11, it is useful to use multiple clocks. In this particular case, for every voice sample, many cycles are needed to perform the filtering function.

**Gated Clocks**: power dissipation in Complementary Metal Oxide Semiconductor (CMOS) circuits, is directly related to the number of signal transitions (change of logic level) on circuit nodes. To reduce power dissipation, logic gates are inserted to prevent the clock from propagating to the parts of the circuit that are irrelevant to the current operation.

A timing verifier must be able to deal with multiple and gated clocks without excessive user intervention.

### 1.3.5 Component Delays

The delay of a component is the time necessary for it to respond to a stimulus. Logic gate delays are related to fabrication technology. A CMOS circuit is essentially a network of P and N transistors interconnected by metal connections. Transistors act as switches (closed or open-circuit).

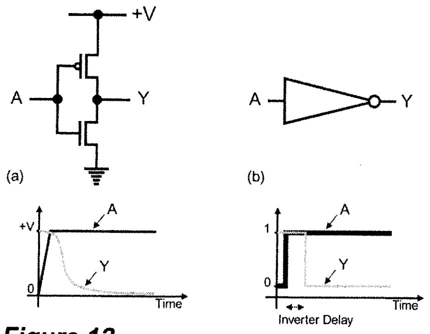


***Figure 12***
*Delay of an inverter.*

**Gate delays**: The logic level at the output of a gate changes from 1 to 0 when the paths that lead to +V become all open and at least one path to ground (0 volt) becomes closed. Transistor switching time is not negligible, the electrical signal changes gradually instead of instantaneously. Fig. 12(a) shows a CMOS implementation of an inverter with its electrical response to a rising transition at its input. Fig. 12(b) shows the logic abstraction of this circuit. One particularity of CMOS gate delays is that they are inertial: two successive events (impulse) at the input of a gate that have time separation (pulse width) less than the gate delay do not affect the output; it is absorbed by the inertial gate delay.

**Interconnect delays**: as opposed to gate delays, interconnect delays are not inertial. They propagate events as waves.

#### 1.3.5.1 Factors Affecting Component Delays

**Gate Delays:**

Gates do not have fixed delay values; their dynamic properties are affected by many factors:

**Fabrication Process**: The silicon atom is an element of group 4 on the periodic table. It forms liaisons with four other atoms to make a non-conducting crystal. To construct transistors, the silicon chip is doped with atoms of group 3 such as gallium to get positive charge carriers (type P regions), and with atoms from group 5 (arsenic) to get negative charge carriers (N type regions). Fig. 13 shows the structure of Metal Oxide Semiconductor (MOS) transistors of types P and N. Feature size and doping density variations introduce uncertainty in the transistor switching time. The fabrication of chips is done on silicon wafers (see Fig.14). It is generally accepted that component features vary depending on the position within a wafer and from one wafer to another. On the same wafer, if A is a point where the doping density is $d_A$, then the density at a point B is contained within $[d_A - f(AB)\ ,\ d_A + f(AB)]$, where $f$ is a positive non decreasing function such that $f(0) = 0$. The latter translates into a correlation law between the different gate delays on the same silicon chip. This law will be later elaborated by considering the effect of the fabrication process along with the effects of power supply and temperature. The relationship between doping density and gate delay is monotone, it can be increasing or decreasing depending on the process technology.

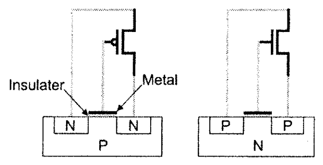


***Figure 13***
*P (left) and N type transistors.*

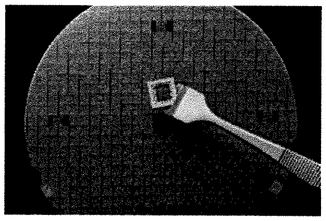


***Figure 14***
*Silicon wafer.*

**Supply Voltage**: Fluctuations in supply voltage implies a change in gate delays. Generally, the delay increases as voltage decreases.

**Temperature**: Ambient temperature and transistor switching activity cause temperature variations in the different regions of the silicon chip. Generally, the clock distribution network is at the highest temperature. A gate delay increases as the temperature rises.

**Slew Rate**: When a signal at the input of a gate switches slowly, the gate response is also slower, resulting in a higher delay.

**Interconnect Delays:**

Metal connections are fabricated by depositing a conductor on the silicon substrate using a chemical process. This is done over several layers due to the fact that circuits are generally non-planar. An interconnect delay is affected by the depositing process, the chips geometry, and the temperature.

### 1.3.5.2 Delay Models and Components Correlation

Integrated Circuit (IC) manufacturers specify the variation margins of the different factors influencing gate delays. Fabrication process, electric power supply and temperature are the most noticeable. A delay value is defined by a set of three values ($d_{min}$, $d_{nom}$, $d_{max}$). The actual delay is contained within the interval [$d_{min}$ , $d_{max}$].

$d_{min}$: Minimal value of the delay with respect to the three factors.

$d_{nom}$: Nominal value, defined over a predetermined condition of the three factors (e.g., temperature 25°, supply voltage 5 volts and doping density d).

$d_{max}$: Maximal value of the delay with respect to the three factors.

The effect of the influencing factors on gate delays of the same silicon chip, apart from signal slew rate, makes it impossible for different gates to have arbitrary values within the associated intervals [$d_{min}$, $d_{max}$]. For example, when the delay of a gate A has its maximum value $d_{maxA}$, the delay of a gate B is necessarily in the interval [$d_{maxB} - \varepsilon$, $d_{maxB}$]. $\varepsilon$ is a positive value that depends on the degree of correlation, a number between 0 and 1 (0: no correlation, 1: 100% correlation). An elaborate delay correlation model is presented in chapter 4.

### 1.3.6 Maximal Delay of a Combinational Circuit

A combinational logic circuit $\xi$ with $m$ inputs and $n$ outputs implements a logic function $f: B^m \rightarrow B^n$ where $B = \{0, 1\}$. Due to gate and interconnect delays, the response of

$\xi$, following an input stimulus, is not instantaneous and a certain lapse is necessary before the outputs stabilize at valid values.

**Definitions and Terminology:**

| A | B | NOT(B) | AND(A,B) |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | $x$ | $x$ | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | $x$ | $x$ | $x$ |
| $x$ | 0 | 1 | 0 |
| $x$ | 1 | 0 | $x$ |
| $x$ | $x$ | $x$ | $x$ |

***Table I**: Floating algebra.*

**Path**: a path of $\xi$ is an alternating sequence of connections and gates which are connected one to the next.

**Path Length**: the length of a path is the sum of the delays of its gates and connections.

**Maximal Topological Delay**: the length of the longest (slowest) path of $\xi$.

**Input Vector**: an input vector for $\xi$ is an element of $B^m \times R$. This represents logic values applied at the inputs of $\xi$ at a certain time $t$. $(e_1,e_2,...,e_m,t)$ is written $(e_1,e_2,...,e_m)_t$.

**Sequence of Input Vectors**: a sequence of $k$ input vectors for $\xi$, $(v_{t1}, v_{t2}, ..., v_{tk})$, is an element of $(B^m \times R)^k$, it is assumed that $t_i < t_{i+1}$ $i$=1,$k$-1. Furthermore, when $v_{ti}$ is applied at time $t_i$, it remains applied until time $t_{i+1}$. Obviously, $v_{tk}$ remains applied indefinitely.

**Floating Algebra**: it is the Boolean algebra augmented with an unknown uncorrelated value, denoted $x$. Table I defines the logic functions NOT and AND over $F = \{0, 1, x\}$ (other functions are deduced in a straight-forward fashion). Many methods use this algebra to define and compute the delay of a combinational circuit. In that case, $\xi$ computes $f: F^m \rightarrow F^n$.

**Maximal Delay of Sequences of Vectors**: the maximal delay of sequences of vectors of $\xi$, denoted $dmax^S(\xi)$ is: $dmax^S(\xi)$ = minimum $\{t \geq 0 \mid \forall\, sv = (v_{-\infty}, ..., v_0)$, all outputs of $\xi$ are stable after time $t$ when $sv$ is applied$\}$. Intuitively, it is the maximal time for $\xi$ to compute its logic function for every possible value of an input vector applied at time 0, assuming the inputs were previously arbitrarily changing.

**Maximal Transition Delay**: The maximal transition-mode delay of circuit $\xi$, denoted $dmax^T(\xi)$ is: $dmax^T(\xi) = \text{minimum } \{t \geq 0 \mid \forall\ (v_{-\infty}, v_0)$, all outputs of $\xi$ are stable after time $t$ when $(v_{-\infty}, v_0)$ is applied$\}$. It is the maximal time for $\xi$ to compute its logic function for every possible value of an input vector $v_0$ applied at time 0, assuming the inputs were previously stable at arbitrary levels.

**Maximal Floating Delay**: The maximal floating delay of circuit $\xi$, denoted $dmax^F(\xi)$ is: $dmax^F(\xi) = \text{minimum } \{t \geq 0 \mid \forall\ (v_{-\infty}, v_0) \in ((\{x\}^m \times \{-\infty\}), (B^m \times \{0\}))$, all outputs of $\xi$ are stable after time $t$ when $(v_{-\infty}, v_0)$ is applied$\}$. Note that the circuit outputs stabilize at values in $B$.

Fig. 15 shows examples of input signals for each of the previously defined delays. Note that the transition delay is not a valid model for sequential circuits because the state register changes value at every clock cycle. Devadas et al determined in [82] that the transition delay is valid when it is greater than $(\frac{1}{2} \times$ topological delay).

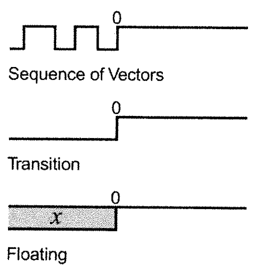


***Figure 15***
*Examples of input stimuli.*

Lam and Brayton have shown in [102] that, for real circuits, floating and sequences of vectors delays are equal. The difference exists only in artificial cases such as the one shown in Fig. 16. In a real circuit, the probability of having two paths of exactly the same length is null. Consequently, the correlation between X and Y in the circuit in Fig. 16 is highly improbable.

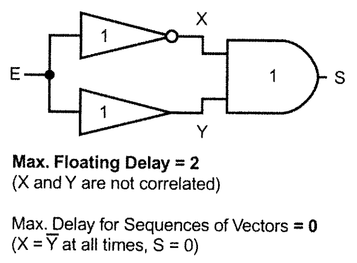

X
E
1
1
1
S
Y
**Max. Floating Delay = 2**
(X and Y are not correlated)
Max. Delay for Sequences of Vectors = **0**
(X = $\overline{Y}$ at all times, S = 0)

***Figure 16***
*Difference between floating and sequences of vectors maximal delays.*

**Monotone Speed-up Property**: A method for maximal circuit delay calculation that uses fixed delay values (max.) instead of intervals is said to satisfy the monotone speed-up property if the following holds:

Let $C$ be a circuit for which the method calculates a max. delay of $d$. The method calculates a max. delayless than or equal to $d$ for any circuit $C'$ that is the same as $C$ except that some of its components are faster.

In other words, speeding-up components of a circuit $C$ does not result in a slower one (as calculated by the method).

### 1.3.7 Minimal Delay of a Combinational Circuit

Definitions of minimal delays for a combinational circuit $\xi$ are expressed in a symmetrical fashion with maximal delays.

**Minimal Topological Delay**: the minimal topological delay of $\xi$ is the length of its shortest path.

**Minimal Delay of Sequences of Vectors**: the minimal delay of sequences of vectors of circuit $\xi$, denoted $dmin^S(\xi)$ is: $dmin^S(\xi)$ = maximum $\{t \geq 0 \mid \forall\ sv = (v_{-\infty}, v_0, \dots)$, all outputs of $\xi$ are stable before $t$ when $sv$ is applied$\}$. Intuitively, it is the time before which the outputs of $\xi$ remain stable after applying $v_0$ and assuming that, subsequently, inputs change arbitrarily.

**Minimal Transition Delay**: the minimal transition delay of circuit $\xi$, denoted $dmin^T(\xi)$ is: $dmin^T(\xi)$ = maximum $\{t \geq 0 \mid \forall\ (v_{-\infty}, v_0)$, all outputs of $\xi$ are stable before $t$ when $(v_{-\infty}, v_0)$ is applied$\}$.

**Minimal Floating Delay**: The minimal floating delay of circuit $\xi$, denoted $dmin^F(\xi)$ is: $dmin^F(\xi)$ = maximum $\{t \geq 0 \mid \forall\ (v_{-\infty}, v_0) \in ((B^m \times \{-\infty\}), (\{x\}^m \times \{0\}))$, all outputs of $\xi$ remain stable before $t$ when $(v_{-\infty}, v_0)$ is applied$\}$.

### 1.3.8 The False Path Problem

Fig. 17 shows a circuit that has a floating delayless than its topological delay. This phenomenon is caused by the fact that, in general, not all signal paths in a circuit can propagate transitions (so called false paths). When the longest paths are false, e.g., path A-C-D-F-G in the circuit of Fig. 17, the actual circuit delay is less than its topological delay. Hrapcenko [57] presented early an extended discussion on the subject, and proved that even minimal circuits may have their longest paths false. He noticed also that false paths appear naturally in accelerated carry-skip adders. This phenomenon makes the problem of deciding whether the maximal circuit delay is less than a certain value NP-complete, as shown in [76].

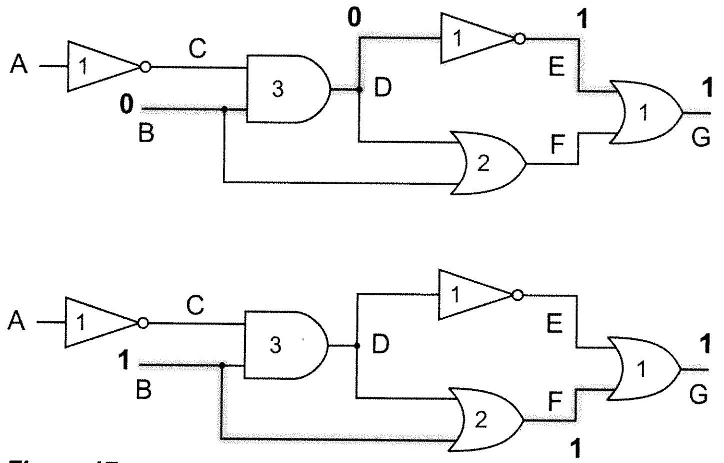


***Figure 17***
*A circuit with false paths (numbers on gates represent delays): when B stabilizes to 0 (1), G stabilizes to 1 after 5 (3) time units. The circuit's floating delay is 5, whereas its topological delay is 7.*

## 1.4 Original Contributions of this Thesis

The major contributions of this thesis are summarized as follows:

- Established the mathematical foundations of the *waveform narrowing* method for the purpose of floating-mode delay calculation, the original method was formulated around the transition-mode.

- Developed a spatial correlation procedure that was effective in reducing the pessimism of the method on standard and industrial benchmark circuits.

- Developed the *Timing Dominators* concept that was very successful in eliminating false violations with minimal added execution time complexity ($n \times \log(n)$).

- Developed a case analysis procedure able to find a test vector, or prove that no violation is possible. The procedure is guided by heuristics inspired by ATPG techniques, namely the controllability measure of [23] and the FAN algorithm of [33]. The procedure uses a novel partitioning strategy based on timing dominators.

In order to provide support for state of the art industrial circuits, we extended the method as follows:

- Developed an intuitive formalism able to express arbitrary complex clocking schemes, along with a procedure to deduce correct default edge selection for setup verification.

- Defined a delay correlation domain based on three-valued delay annotation (min, typ, max) using the novel concept of *normalized delays*. The resulting constraints can be used to build complex correlation networks able to model arbitrary complex component delay correlation, like position dependence, rising-delay vs. falling-delay, etc.

- Defined more than 70 constraint primitives able to model industrial cell libraries.

- Developed a hard multiplexer primitive that reduces the inherent pessimism of the floating delay model.

- Developed and automated a general concept for modeling combinational cells. And added cell aware constraints that remove the pessimism induced by path delays of unknown polarities.

- Added support for automatic handling of combinational loops, still present in some synchronous industrial designs.

- Implemented an industrial-grade version of the timing verifier in the object oriented language C++, and evaluated the Waveform Narrowing method on industrial circuits provided by Nortel Networks.

## 1.5 Plan of the Thesis

Chapter 2 presents an overview of the methods for computing maximal circuit delays proposed in the literature.

Chapter 3 contains the proposed timing verification method, along with the results on the standard ISCAS'85 benchmark suite.

Chapter 4 presents the extensions we implemented to enable the method to be applied to state of the art industrial designs, along with the results on industrial circuits provided by Nortel Networks.

Chapter 5 concludes the thesis.

# CHAPTER II LITERATURE REVIEW

In this chapter we review the major methodologies for computing the maximal delay in combinational logic circuits as presented in the literature. These methods have evolved from a simple topological sort of the PERT project by Kirkpatrick and Clark in 1966 [56] to more recent methods that consider the circuit functionality and automatically eliminate false paths.

## Motivation

In the early 1980's, the increasing complexity of logic circuits made the use of simulators such as SPICE [9] futile for timing verification. SPICE solves differential equations to deduce the waveform at the output of a circuit, given a precise waveform at each input. Execution time was estimated to about a minute for each circuit transistor on a typical computer of that era. To compute the transition delay of a circuit, an exponential number of simulation runs is required in order to account for all possible situations. Therefore, a trade-off had to be made between execution time and precision.

## First Approach: Limited Worst Case Simulation - Case Analysis

In 1980, McWilliams introduced SCALD [58], a gate-level timing verifier. The combinational circuit is represented by a directed acyclic graph. Each node represents a gate and is associated with a delay value. Gate algebra is defined over a set of seven values {0, 1, rising-transition, falling-transition, stable, changing, unknown}. A signal is defined by a list of values and time intervals, e.g. [stable for 1 ns, varying for 2 ns, stable the rest of the cycle]. Each gate signal is computed using a lookup table, keeping track of minimal and maximal event times. The evaluation is achieved using an event driven simulator that triggers the evaluation of a gate when all its inputs are ready (this scheduling technique is referred to as path tracing in literature). The algorithm complexity is linear, function of the number of connections. The advantage of this method over a simple topological sort is the possibility to perform case analysis by allowing the user to specify constant values (0 or 1) on certain nodes. Case analysis can eliminate false paths, however, its abuse can underestimate the circuit delay as will be seen later when static sensitization is discussed.

Case analysis has been adopted by many verifiers such as TV [60] and CRYSTAL [62], both of which function at the transistor level. Other systems such as Hitchcock's [59] allow the user to explicitly specify false paths. With the increasing complexity of digital circuits, however, manual identification of false paths became very hard and error prone.

## Automatic Identification of False Paths

Automatic false path elimination techniques are classified as follows:

1) **Path Enumeration** [63, 66, 70, 72, 73, 75, 76, 78, 82, 87, 88, 89, 110, 117]: these methods search systematically all circuit paths and use sensitization criteria (defined later) to decide whether a path is false.

2) **Reduction to a Test Generation Instance** [86,93]: these methods reduce the problem of deciding whether the maximal circuit delay is greater than a certain value to an instance of the problem of generating a test vector for multi-stuck-at faults.

3) **Approximate**: [108,114] present approximate methods that construct Boolean expressions using a subset of the involved variables, and solve them by symbolic methods.

4) **Optimization**: [85,102] present a mixed Boolean-linear programming formulation, [98,109,112,118] present a formalism based on constraint satisfaction.

## 2.1 Methods Based on Path Enumeration

These methods search systematically to find the longest circuit path that is not false, the so-called longest sensitized path. Sensitization criteria as explained thoroughly in the following sub-paragraphs are predicates used to decide whether an event at a gate input can propagate to its output. The generic algorithm that stops when the longest true path is found is a best-first search. The first step is to mark each gate with the length of the longest path that reaches an output, starting from the gate itself. The marks are then used to guide the search: paths are incrementally built, one gate at a time, starting at the inputs, and keeping the paths in a priority queue which returns the potentially longest path when an output is reached. Details of the algorithm are given in [66]. Simple depth-first search is used when all true paths are to be found.

These methods compute the maximal circuit delay using the maximal delay values for gates and connections instead of their interval of uncertainty $[d_{min}, d_{max}]$. The following definitions are required to simplify the presentation:

**Side Input**: for a gate *g* belonging to a path *p*, inputs of *g* that do not lie on *p* are side inputs of *p*.

**Controlling Value**: a controlling value of a logic gate is a logic value which, if applied at an input, determines the output independently from the other inputs, e.g., 0 for an AND gate.

**Non-Controlling Value**: a non-controlling value of a logic gate is a logic value that is not controlling.

Consider a path *p* containing *n* gates $g_1, g_2, \ldots, g_n$. A sensitization criterion is a conjunction

$$\boxed{\Phi = \varphi(g_1) \wedge \varphi(g_2) \wedge \ldots \wedge \varphi(g_n)}$$

which evaluates to true if the path is sensitizable. $\varphi$ is a predicate which depends on the gate type. It evaluates to true if the last event at the input *E* that is part of *p*, propagates to

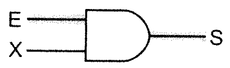
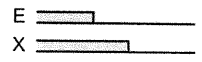
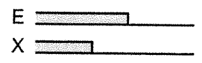
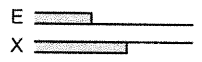
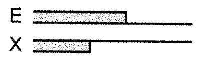
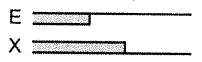
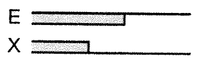
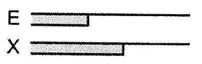
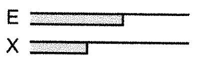
|  | Conser-vative | Static | Brand & Iyengar | Co-Static | Viability | Floating |
|---|---|---|---|---|---|---|
|  | ✔ | ✘ U | ✔ | ✔ | ✔ | ✔ |
|  | ✘ | ✘ | ✘ | ✔ V | ✘ | ✘ |
|  | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|  | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|  | ✔ N | ✘ | ✔ N | ✘ | ✔ N | ✘ |
|  | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
|  | ✔ N | ✔ N | ✔ N | ✔ N | ✔ N | ✘ |
|  | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

***Table II***
*Sensitization criteria for delayless two-input AND gate.*
*E: gate input belonging to the path which we check whether it is false.*
*X: gate input not belonging to the path.*
*S: gate output.*
*✘: last event at E does not propagate to S.*
*✔: last event at E propagates to S.*
*N: the decision does not affect the outcome of computing the max. delay using the criterion.*
*V: cause of overestimating the delay.*
*U: cause for underestimating the delay.*
*Brand & Iyengar: last event of X is determined using topological information only.*

the gate output $S$. $T_E$ is the occurrence time of the last event on $E$, $T_X$ is the occurrence time of the last event on a side input $X$. Note that the sensitization criteria are expressed using partial information: The last events at gate inputs. Therefore, they are only heuristics and do not necessarily reflect reality. A safe sensitization criterion is one that does not cause delay under-estimation if it is used for maximal circuit delay computation. Table II summarizes the sensitization criteria defined for a delayless two-input AND gate. The column labeled Conservative represents a safe criterion: the only case where the last event on $E$ does not propagate to $S$ is when $X$ stabilizes to a controlling value earlier. It is important to note that, when $E$ stabilizes to a non-controlling value earlier than $X$, the last event on $S$ propagates from $X$. In such case, the last event on $E$ is irrelevant for computing the maximal circuit delay. It is this property that makes the Floating-Mode sensitization criterion safe and equivalent to the Conservative one when it comes to computing the maximal circuit delay, though they differ when paths shorter than the circuit delay are checked for sensitizability.

The following sections define the predicate $\varphi$ in terms of the following:

- *g*: Gate for which $\varphi$ is defined;
- *S*: The output of *g*;
- *E*: Input of *g* that belongs to the path that is being checked for sensitizability;
- $T_E$: The occurrence time of the last event on *E;*
- *X*: A side input of E;
- $T_X$: The occurrence time of the last event on *X;*

## 2.1.1 Static Sensitization

A path *p* is statically sensitized if there exists an input vector that sets the side inputs of each gate *g* of *p* to non-controlling values (with respect to *g*).

$$\boxed{\varphi(g) = X \text{ is non-controlling}}$$

The shaded path of Fig. 18 is statically sensitized: all side inputs are set to non-controlling values.

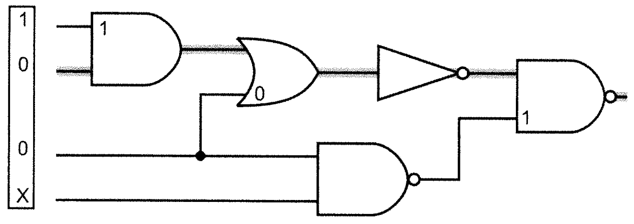


***Figure 18***
*A path statically sensitized.*

Proposed in [66], this criterion can cause an under-estimation of the maximal circuit delay as shown in [72] and [88]. Fig. 19 shows a circuit with a true path of length 3, identified by this criterion as false. All gates have delays of 1. Non controlling values for the side inputs of the path {b, d, f, g} are a = 1, c = 0, and e = 1. Although a = 1 implies that e = 0, it is possible for the falling transition at b to travel to g because of the dynamic behavior of the circuit. In fact, before time 0, we have a = 1, b = 1, c = 0, d = 1, e = 0, f = 1, and g = 1. Both a and b change values from 1 to 0 at time 0, causing d to fall to 0 at time 1, and e to rise to 1 at time 1, causing g to fall to 0 at time 2. Then f falls to 0 at time 2, causing g to rise to 1 at time 3.

This example shows the inadequate use of case analysis based on constant logic values, as proposed in SCALD and other tools.

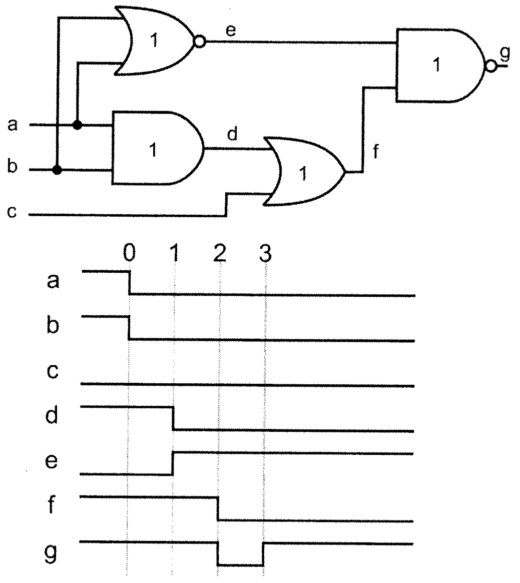


***Figure 19***
*A true path of length 3 identified as false by static sensitization criterion.*

### 2.1.2 The Brand-Iyengar Criterion

To avoid the delay under-estimation problem of static sensitization, the authors of [63] suggested imposing non-controlling values only on the side inputs $X$ for which $Top_X < Top_E$ where $Top_X$ $(Top_E)$ is the longest path from a primary input to $X$ $(E)$. Therefore,

$$\boxed{\varphi(g) \;=\; (Top_X \geq Top_E) \vee X \text{ is non-controlling}}$$

The column Brand & Iyengar in Table II gives the wrong impression that this criterion is equivalent to the conservative one. The fact of the matter is that it is not as tight. The pessimism is caused by the fact that there may be cases where $Top_X \geq Top_E$ and yet $T_X < Top_E$ (the longest path from a primary input to $X$ is false). In such case, Brand-Iyengar criterion evaluates to true, while the conservative one requires $X$ to stabilize to a non-controlling value.

### 2.1.3 Co-Static Sensitization

This criteria is proposed by Devadas et al in [93]. For the last event to propagate from $E$ to $S$, if $E$ stabilizes at a non-controlling value, then $X$ must stabilize to a non-controlling value also.

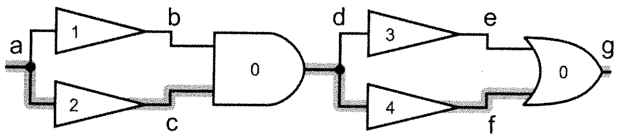


***Figure 20***
*A false path of length 6 identified as true by co-static sensitization (a = 0).*

$$\boxed{\varphi(g) = (E \text{ is controlling}) \vee (X \text{ is non-controlling})}$$

Unlike the static sensitization, this criterion does not under-estimate the circuit delay, however, is does over-estimate it, as can be seen in the example of Fig. 20. The Numbers on gates represent their delays. A value of 0 at the input a implies 0 on all circuit nodes, satisfying the criterion for the path {a, c, d, f, g} with a delay of 6. However, it is impossible for a transition to travel along this path. In fact, when a falls to 0 at time 0, b follows at time 1, and d follows b instantaneously at time 1. In this case, the last transition at g is at most at time 5. In the case of a rising transition at a, d follows after 2 time units, but g, controlled by the short path {d, e} because 1 is a controlling value for the OR gate, follows d after 3 time units instead of 4.

### 2.1.4 Viable Sensitization

Suggested by McGeer and Brayton [72], for the last event to propagate from $E$ to $S$, $X$ must stabilize to a non-controlling value before $T_E$, or $X$ stabilizes after $T_E$.

$$\boxed{\varphi(g) = (T_X \geq T_E) \vee (X \text{ stabilizes to a non-controlling value})}$$

This is the most conservative criterion, it is the negation of a sufficient condition that makes a path not sensitized: a side input that stabilizes to a controlling value earlier.

### 2.1.5 Floating Mode Sensitization

This criterion is proposed by Chen and Du [71]. The last event at $E$ propagates to $S$ according to the following rules:

when $E$ stabilizes to a controlling value, $T_X \geq T_E$ or (X stabilizes to a non-controlling value and $T_X < T_E$)

when $E$ stabilizes to a non-controlling value, $X$ must stabilize to a non-controlling value and $T_X < T_E$.

The maximal floating delay of a combinational circuit defined in section 1.3.6 is compatible with this criterion. It is widely used by timing verifiers due to its simplicity and the fact that it determines a tight upper bound for the maximal circuit delay. Furthermore, this bound is an upper bound for the family of circuits that have gate delays contained in the interval [0 , maximal gate delay], that is, the criterion satisfies the so called monotone speed-up property. Note that this property is satisfied in all previously stated sensitization criteria, except static sensitization.

Another sensitization criterion, which has not been stated, is dynamic sensitization. This criterion computes the maximal transition delay defined in section 1.3.6, taking into consideration the instantaneous signal values. This criterion does not possess the monotone speed-up property. An interval delay model has to be used instead of simply the maximal delay values, which makes the computation too complex to be practical.

## 2.2 Reduction to a Test Generation Problem

Ashar et al [86], and Devadas et al [93] proposed two similar methods to reduce the problem of deciding whether a circuit delay is greater than $\delta$ to an instance of a multi stuck-at-fault test generation problem (see [31-42] for details on test generation). These methods use the properties of the transformation of a logic circuit into a two-level circuit, a disjunction of conjunctions, known as the equivalent normal form. This transformation is illustrated on the example of Fig. 21 extracted from [93] (numbers on gates are identifiers). First, all fan-outs are eliminated by duplicating common structures, resulting in the tree circuit of Fig. 22. Then, the inverters are pushed back to the inputs, applying DeMorgan's law resulting in the circuit of Fig. 23. The normal form logic expression representing the circuit is:

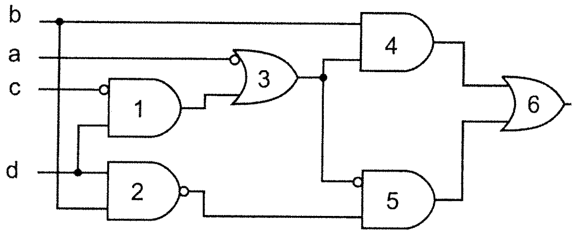


***Figure 21***
*A circuit to be transformed to the equivalent normal form.*

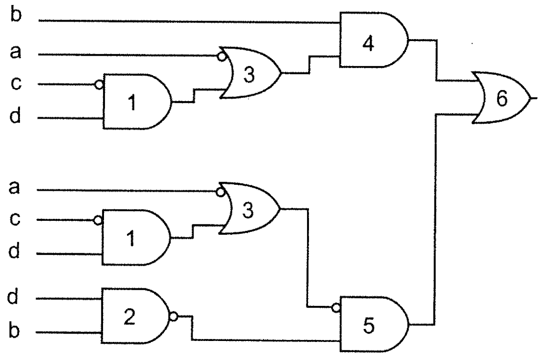


***Figure 22***
*Removing fan-outs results in a tree circuit*

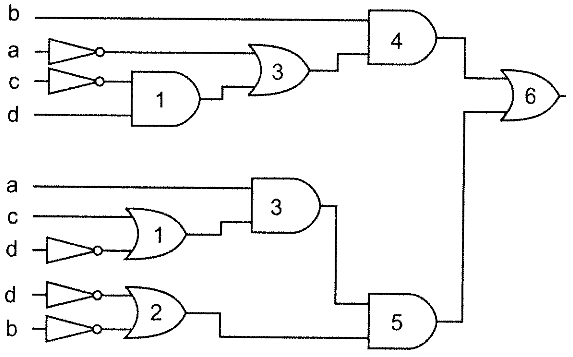


***Figure 23***
*The result of pushing back the inverters.*

$NFE = (b_{\{4,6\}} \wedge \bar{a}_{\{3,4,6\}}) \vee$

$(b_{\{4,6\}} \wedge \bar{c}_{\{1,3,4,6\}} \wedge d_{\{1,3,4,6\}}) \vee$

$(a_{\{3,5,6\}} \wedge c_{\{1,3,5,6\}} \wedge \bar{d}_{\{2,5,6\}}) \vee$

$(a_{\{3,5,6\}} \wedge c_{\{1,3,5,6\}} \wedge \bar{b}_{\{2,5,6\}}) \vee$

$(a_{\{3,5,6\}} \wedge \bar{d}_{\{1,3,5,6\}} \wedge \bar{d}_{\{2,5,6\}}) \vee$

$(a_{\{3,5,6\}} \wedge \bar{d}_{\{1,3,5,6\}} \wedge \bar{b}_{\{2,5,6\}})$

*NFE* is represented by the circuit of Fig. 24. Each variable represents an individual path of the original circuit; the indices contain the identifiers of the path gates. To make it equivalent to the original circuit, a delay equal to the path length is placed at each input and gates are assumed delayless. Suppose that the maximal floating delay of the circuit is to be computed for the rising transition at the output, i.e., $\tau$ = maximum time separating the application of an input vector $v$ and the last event at the output for all $v$ such that $ENF(v) = 1$ (obviously, the last event is a rising transition). Assuming that $\tau > \delta$, there exists an input vector $v$ such that, referring to the circuit of Fig. 24:

1) all AND gates belonging only to paths shorter than or equal to $\delta$ stabilize at 0;

2) at least one AND gate belonging to a path longer than $\delta$ stabilize at 1.

The input vector $v$ is a test vector for multi stuck-at-0 faults injected at each of the AND gates inputs belonging to a path longer than $\delta$.

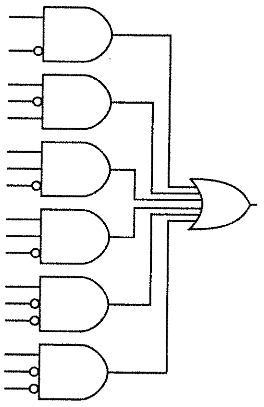


*Figure 24*
*Normal form representation.*

To compare the maximal floating delay for the falling transition, the method is simply applied to the inverted circuit..

Devadas et al suggest an algorithm in [93] that uses the original circuit without modification at the cost of modifying the way the test generation is done. Ashar et al [86] use standard tools for test-generation at the cost of modifying the circuit, which in the worst case doubles in size by the process of pushing back the inverters to the inputs. The execution time for the circuit c1908 from the ISCAS'85 [3] benchmark is 3675 seconds for the method in [93] on a 10 MIPS machine, and it is 800 seconds for the one in [86] on a 20 MIPS machine.

## 2.3 Mixed Boolean-Linear Programming

Lam et al suggested in [85] a Mixed Boolean-Linear programming formalism to compute the exact circuit delay using the interval delay model for the gates. The dynamic behavior of the circuit is represented by a timed Boolean function. For example, a two-input AND gate with a delay of 2 is represented by $s(t) = a(t-2)$ AND $b(t-2)$ where $t$ is time, $s$ is the output, $a$ and $b$ are the inputs. The function representing the circuit behavior at its output is $f(t, x_1, ..., x_n, d_1, ..., d_m)$ where $x_1, ..., x_n$ are the inputs, $d_1, ..., d_m$ are the gate delays and $t$ is time. The computation of the maximal delay is formulated as follows:

$$Delay = max \ \ t \text{ such that}$$

$$f(t, x_1, ..., x_n, d_1, ..., d_m) \neq f(\infty, x_1, ..., x_n, d_1, ..., d_m) \quad (1)$$

$$d_{min_i} \leq d_i \leq d_{max_i}$$

To resolve (1) for certain values of $t$ and the delays $d_i \in [d_{min_i}, d_{max_i}]$ $f(t, x_1, ..., x_n, d_1, ..., d_m)$ XOR $f(\infty, x_1, ..., x_n, d_1, ..., d_m)$ is represented by a Binary Decision Diagram (BDD, see [15-21]) and checked for satisfiability. A BDD is a directed acyclic graph representation of a Boolean function. The disadvantage of BDDs is their size that is exponential in the worst case (in terms of the number of variables), encountered with multipliers for example. The execution time of this method on circuit c1908 from the ISCAS'85 benchmark is 12140 seconds on a 38 MIPS machine.

## 2.4 Hierarchical Method

Yalcin and Hayes [107] proposed a method applicable at different levels of abstraction of a circuit. They represent the delay of the combinational circuit with $n$ inputs and $m$ outputs with a matrix of dimensions $n \times m$. The element $(i, j)$ represents the delay from input $i$ to output $j$. This delay is a set of pairs $(\psi, t)$ where $\psi$ represents the conditions that have to be satisfied to have a delay $t$ from input $i$ to output $j$. The circuit matrix is deduced by operations on the matrices corresponding to its building blocks. Satisfiability of conditions $\psi$ is evaluated symbolically by manipulating BDDs. The excessive requirements for execution time and memory inclined the authors to follow an approximate approach in [108], by restraining the conditions $\psi$ to depend on a limited number of signals, the controlling lines. In the case of a multiplier, however, there are no controlling lines and this method simply computes the topological delay.

## 2.5 Method Based on Constraint Satisfaction

Cerny and Zejda proposed in [98] a method based on Waveform Narrowing (WN). Using the circuit description and the operating conditions, a constraint system is built and partially resolved using an event driven mechanism.

A constraint system is composed of a finite set of variables $\{X_1, X_2,..., X_n\}$ that take values from their respective domains $D_1, D_2,..., D_n$, and a set of relational constraint operators $\{C_1, C_2,..., C_m\}$, each operating on a subset of the variables. A domain $D_k$ of a variable $X_k$ initially contains the set of all possible values $X_k$ can take. A solution of the constraint system is an assignment for all the variables, from their respective domains, that makes the system consistent, i.e., all the constraints are satisfied. When a constraint operator $C_k$ is applied, it removes from the domains of the associated variables values that are not compatible, i.e., values that are not part of any solution. The resulting system contains the same original set of solutions.

Modeling a timing verification problem using Waveform Narrowing concept consists of:

- Defining domains that represent sets of binary waveforms;
- Defining domains that represent sets of delay values;
- Defining constraint operators that represent logic gate functions. When applied, the constraint operators remove from the domains associated with the gate terminals the values that do not satisfy the local gate constraint, regardless of the global circuit function;
- Building a constraint system based on the logic circuit description, the component timing properties (delays / timing constraints), and the operating conditions (clock frequency). The constraint system should have a solution if and only if the timing constraints are violated;

- Resolve the system partially by repeatedly applying the constraint operators until the greatest fixpoint is reached; it is no more possible to change the domains by applying the constraint operators. If the resolution results in empty domains, we conclude that the timing constraints are satisfied. Otherwise, no conclusion can be drawn.

This method is an attractive framework for timing verification as it can efficiently handle component delay correlation, eliminates false paths, and models the transition and floating-mode delays. The method can give false negative results because the constraint system resolution is partial, however, it is possible to tighten the results by investing more processing time. The method is the basis for our work that is detailed in Chapters 3 and 4.

## 2.6 Summary

Timing verification is a critical phase in the design flow of VLSI circuits. In the emerging system-on-a-chip technology, timing verifiers are faced with multi-million-gate chips that need to be verified in hours. Therefore, quasi-linear complexity is imposed on any commercially acceptable timing verification algorithm. Unfortunately, the timing verification problem is NP-Complete [76]. The complexity of the problem is caused by the fact that, in general, not all signal paths in a circuit can propagate transitions (so called false paths). Many techniques have been developed to deal with the false path problem. Algorithms based on path enumeration suffer from poor performance, as they may have to enumerate a very large number of paths, however, it is possible to improve the performance by memorizing inconsistencies between sub-paths [110]. In [86,93] the authors reduced the problem of comparing the circuit delay with $\delta$ to an ATPG problem. In [85] a method based on timed Boolean functions and a BDD representation was formulated, however, it may experience exponential space explosion for certain circuits. To cope with the increasing complexity, the research community tends to offer approximate solutions, as is the case in [108] where the entries of the conditional delay matrix are restricted to be expressed using the controlling lines, smoothing out the other variables. The group at the Université de Montréal has developed a method based on waveform narrowing

[98,112,118] inspired by constraint logic programming using relational interval arithmetic [14]. Unlike other methods, it can efficiently handle component delay correlation [112] and adapt to different circuit-delay modes (transition or floating).

A recent publication [119] is particularly interesting. It exposes the approach used to verify the timing properties of the 600 MHz Alpha processor. This processor is built on a one square centimeter silicon chip, containing roughly 18 million transistors. Their approach is summarized as follows:

1) Uses a fixed delay model (nominal value) for components
2) Uses SPICE [9] to simulate the clock signal
3) The longest circuit path is responsible for its delay

Note that no automatic false path elimination technique was used. Having to deal with a circuit of such size, even a quadratic algorithm is not acceptable. Moreover, the use of a fixed delay model is motivated by the fact that, when the timing verifier does not handle component delay correlation, the use of interval delay model introduces excessive pessimism. Finally, the Alpha design group had to program their own timing verifier in order to be able to handle non-standard design techniques, such as gated clocks, necessary to achieve the 600 MHz clock frequency.

The main drawback that keeps exact false path elimination methods from serious industrial use is their inherent computational complexity:

- Methods based on BDD evaluation have an exponential space complexity in the worst case. For example no method was able to build a BDD for the multiplier c6288 from ISCAS-85 benchmark suite without exhausting the system memory. But they can apply existential abstraction of variables to reduce complexity for approximate (upper bound) values, as is the case in [108] where the variables are limited to the controlling lines.

- Methods that reduce the problem to an instance of a test generation problem have exponential time complexity in the worst case. For instance, these methods experience an excessive execution time for simple circuits like c1908 from the ISCAS-85 benchmark suite.

- Methods based on path enumeration may have to enumerate an exponential number of paths. For instance, c6288 has more than $10^{18}$ paths!

Beyond the computational complexity issue, a methodology for circuit delay computation is required to be easy to integrate with existing EDA tools. The objective of our work is to provide a method that has the characteristics mandatory for industrial use, such as:

- Quasi-linear time and space complexity.
- Possibility to tighten results of critical circuit nodes by investing processing time.
- Supports interval delays and component delay correlation.
- Supports delays as defined in Standard Delay Format (SDF) [13].
- Supports gated clocks.
- Supports correctly transparent latches.
- Supports complex clocking schemes.
- Provides easy and automated cell library modeling.

The following chapter contains the proposed timing verification method, along with the results on the standard ISCAS'85 benchmark suite.

# CHAPTER III METHOD BASED ON WAVEFORM NARROWING

In this chapter, we present the proposed timing verification method based on the work of Cerny and Zejda in [98]. After explaining the basic idea, two examples are used to develop intuition. The method is then formalized and further extended using conservative reduction techniques and a case analysis procedure. The original method was formulated around the transition-mode delay, and implemented no pessimism reduction techniques.

## 3.1 Overview of the WN Method

The waveform narrowing method is a custom constraint programming system adapted for timing verification. A constraint system is composed of a finite set of variables $\{X_1, X_2,..., X_n\}$ that take values from their respective domains $D_1, D_2,..., D_n$, and a set of relational constraint operators $\{C_1, C_2,..., C_m\}$, each operating on a subset of the variables. A domain $D_k$ of a variable $X_k$ initially contains the set of all possible values $X_k$ can take. A solution of the constraint system is an assignment for all the variables, from their respective domains, that makes the system consistent, i.e., all the constraints are satisfied. When a constraint operator $C_k$ is applied, it removes from the domains of the associated variables values that are not compatible, i.e., values that are not part of any solution. The resulting system contains the same original set of solutions.

### 3.1.1 Inconsistency Property

If applying repeatedly the constraints results in one of the domains becoming empty, the constraint system has no solution. However, if all the domains are non-empty, the system may still have no solution.

## 3.2 Formalism for Timing Verification

Formulating a circuit delay computation problem by means of a constraint system involves defining signal domains (sets of binary waveforms), delay domains (intervals), and constraints representing logic gate functions. The variables and the relational constraints represent the signal values of circuit nets and the logic gate functions, respectively. The specific circuit-delay mode and the timing constraints being verified introduce further restrictions on the domains.

### 3.2.1 Abstract Waveforms

- A binary waveform is a mapping $f{:}R \to \{0, 1\}$.
- The space of all binary waveforms is $BW = \{f{:}\ R \to \{0, 1\}\}$.
- An abstract waveform is a subset of $BW$ defined as
  $w = v|_{lmin}^{max} = \{f \in BW \mid \exists t' \in [lmin, max]\ f(t') \neq v \ \wedge\ \forall t > t'\ f(t) = v\}$.
  $v|_{lmin}^{max}$ contains the binary waveforms that are stable at value $v$ after time $max$ and undergo the last transition at or after time $lmin$.
  **Example:**
  $0|_{10}^{10}$ contains the binary waveforms that: undergo 1 to 0 transition exactly at time 10, stable at 0 after time 10, and can be anything before time 10.
  **Infinite bounds extension:**
  $v|_{-\infty}^{+\infty}$ contains the set of all binary waveforms that stabilize at $v$.
  $v|_{-\infty}^{-\infty}$ contains the binary waveform that is stable at $v$ for all finite values of time.
  Note that $v|_{-\infty}^{+\infty} \cap v|_{-\infty}^{-\infty} = v|_{-\infty}^{-\infty}$ and $0|_{-\infty}^{+\infty} \cap 1|_{-\infty}^{+\infty} = \phi$ by definition.
- The abstract waveform space is $AW = \{v|_{lmin}^{max} \mid lmin, max \in R \wedge v \in \{0, 1\}\}$ $\cup\ \{0|_{-\infty}^{+\infty}, 1|_{-\infty}^{+\infty}, 0|_{-\infty}^{-\infty}, 1|_{-\infty}^{-\infty}\}$.

References to $v$, $lmin$ and $max$ of an abstract waveform $w$ are denoted $w.v$, $w.lmin$ and $w.max$, respectively. $w.v$ is the ***class*** and $[w.lmin, w.max]$ is the ***last-transition interval*** of

$w$. If $w.lmin > w.max$ then $[w.lmin, w.max]$ is empty and $w$ itself is also empty, denoted $w = \phi$.

Abstract waveforms are used in the formalization of maximal circuit delay verification for which the relevant signal waveforms are the ones that do stabilize to a final logic value at a finite value of time.

## 3.2.2 Abstract Signals Domain

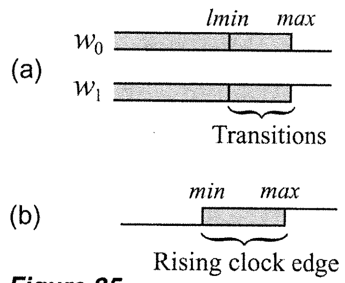


***Figure 25***
*(a): Abstract signal domain.*
*(b): Clock edge domain.*

The objective is to define a domain model for the set of possible binary waveforms that an electrical signal, abstracted to the timed Boolean domain, can take values from. In timing verification, signals are considered in a finite time interval. A signal is normally unstable for a certain period of time, and then it stabilizes to a final binary value. Therefore, it is convenient to compose the domain using two abstract waveforms, one containing the waveforms stabilizing at 0, the other containing the ones stabilizing at 1. This subdivision is a key element when it comes to defining the relational constraints that model the gate functions such as AND, OR, NOT, etc.

- An abstract signal S is a pair of abstract waveforms $(w_0, w_1) \mid w_0.v = 0$ and $w_1.v = 1$ .
- The space of all abstract signals is $AS = \{w \in AW \mid w.v = 0\} \times \{w \in AW \mid w.v = 1\}$. Domains of the variables representing digital signals are elements of $AS$.

References to $w_0$ and $w_1$ of an abstract signal S are denoted S.$w_0$, S.$w_1$, respectively. Fig. 25 (a) shows the graphical representation of the abstract signals domain used in the examples throughout this thesis.

## 3.2.3 Other Domains

Beside abstract signals, clock domains are represented as the interval uncertainty of the occurrence time of the relevant clock edge (Fig. 25 (b)), delays are represented as intervals.

### 3.2.4 Timing Verification

Given a logic circuit and operating conditions, we build a constraint system that is consistent, i.e. has a solution, if and only if the timing constraints are violated. The system is built as follows:

- For each circuit net $N_k$ we associate a variable $X_k$ that takes values from its domain $D_k$. $D_k$ initially contains the set of all possible values (waveforms) $X_k$ can take.

  For each logic gate $G_k$ driving a net $N_{k1}$ (gate output), and driven by nets $\{N_{k2}, ..., N_{kh}\}$ (gate inputs), we associate a constraint operator $C_k$ that operates on the domains $D_{k1}, D_{k2}, ..., D_{kh}$ from which the variables $X_{k1}, X_{k2}, ..., X_{kh}$ take values, respectively. Let $f : BW^{h-1} \rightarrow BW$ be the timed Boolean function $G_k$ implements. An assignment to the variables $\{X_{k1}, X_{k2}, ..., X_{kh}\}$ satisfies the gate constraint if $X_{k1} = f(X_{k2}, ..., X_{kh})$. When applied, the constraint operator $C_k$ removes from each domain $D_{ki}$ (narrows $D_{ki}$) the values that, when assigned to $X_{ki}$, do not satisfy the gate constraint for all possible assignments to the other variables.

**Constraint Operators:** the constraint operator $C_f$ of a two input gate implementing the timed Boolean function $f$ ($A$, $B$: domains of the inputs, $Y$: domain of the output) is defined in terms of the forward and partial inverse functions defined as follows:

**Forward Function:**

$$forward^Y(A, B) = \{f(a, b) \mid a \in A, b \in B\}$$

**Partial Inverse functions:**

$$partialInverse^A(B, Y) = \{a \in BW \mid \exists\ b \in B \text{ and } y \in Y, y = f(a, b)\}$$

$$partialInverse^B(A, Y) = \{b \in BW \mid \exists\ a \in A \text{ and } y \in Y, y = f(a, b)\}$$

**Constraint Operator:** the constraint operator is defined as $C_f(A, B, Y) = (A', B', Y')$ such that:

$$Y' = Y \cap forward^Y(A, B)$$

$$A' = A \cap partialInverse^A(B, Y)$$

$$B' = B \cap partialInverse^B(A, Y)$$

Generalization to multiple input gates is trivial.

**Example:** Fig. 26 shows the results of applying the constraint operator of a delayless *AND* gate. the *and* function is a mapping $and : BW \times BW \rightarrow BW$, for $a, b \in BW$ $and(a, b) = y$ such that $y(t) = a(t)$ and $b(t)$ $\forall\ t \in R$.

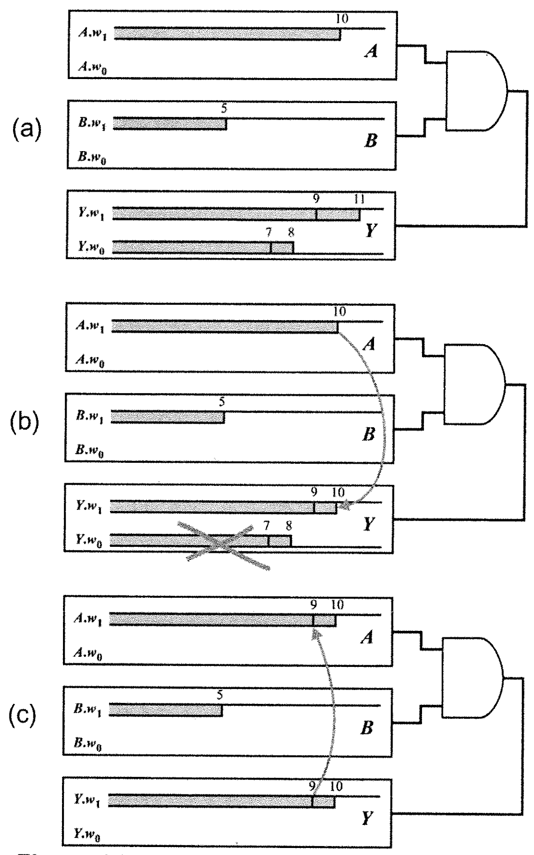


***Figure 26***
*A delayless AND gate constraint operator.*
*(a) Initial domains.*
*(b) Effect on the gate output Y.*
*(c) Effect on the input A.*

**Initial state:** Fig. 26(a) shows the initial domains of the gate inputs A and B, and the gate output Y, before the *AND* constraint operator is applied:

$A = (\phi, 1|_{-\infty}^{10})$, contains the waveforms stable at 1 after time 10;
$B = (\phi, 1|_{-\infty}^{5})$, contains the waveforms stable at 1 after time 5;
$Y = (0|_7^8, 1|_9^{11})$, contains the binary waveforms stable at 1 after time 11, having transitions in [9,11], and the waveforms stable at 0 after time 8, having transitions in [7,8];

**Effect on the output *Y*:** Fig. 26(b) shows how the gate output is narrowed by deductions derived from the gate function: Let $a \in A$, $b \in B$, and $y \in Y$ waveforms that satisfy the *AND* constraint: $y = and(a, b)$. Since $a$ is stable at 1 after time 10, and $b$ is stable at 1 after time 5, $y$ is necessarily stable at 1 after time 10. Therefore, $Y$ is narrowed to $(\phi, 1|_9^{10})$: waveforms unstable after time 10 and those stabilizing at 0 are removed.

**Effect on *A*:** Fig 26(c) shows the narrowing performed on the input $A$. Let $y \in Y$, $\exists\ t \in [9, 10]\ |\ y(t) = 0$. Since any waveform in $B$ is stable at 1 after 5, waveforms in $A$ that satisfy the *AND* constraint are $a \in A \mid a(t) = 0$. Therefore, waveforms in $A$ that are stable at or after time 9 are removed, resulting in $A = (\phi, 1|_9^{10})$.

**Effect on *B*:** no narrowing is possible on $B$ because, $\forall\ b \in B, \exists\ a \in A$ and $y \in Y$ such that $y = and(a, b)$. For example,

$$a(t) = y(t) = \begin{cases} 0 \text{ for } t \le 10 \\ 1 \text{ otherwise} \end{cases} \text{ satisfy } y = and(a, b)\ \forall\ b \in B.$$

- When delays are represented as intervals instead of fixed values, additional variables and domains are associated with circuit component delays, along with appropriate constraints that handle component delay correlation.

The system is partially resolved by an event-driven engine that repeatedly applies the constraint operators until the greatest fixpoint is reached. If we get empty domains we conclude that the timing constraints are satisfied (no violation), otherwise, no conclusions can be drawn.

A formal presentation of the WN method is presented later, after developing intuition by means of the following two examples.

## 3.3 Combinational Circuit Example

Consider the circuit in Fig. 27(a) where the numbers on gates represent their maximal delays. The operating conditions and timing requirements imposed on the circuit are:

- Input data (A and B) stable after time 0 (operating conditions).

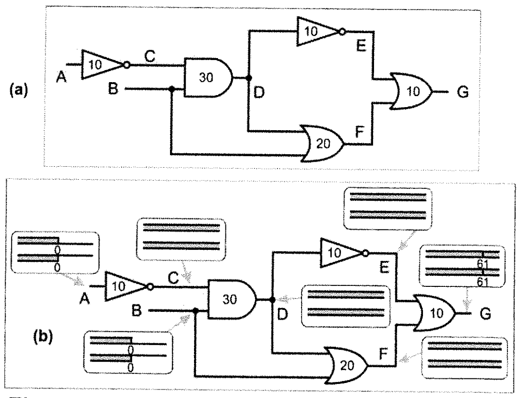


***Figure 27***
*(a): Combinational circuit with a false path.*
*(b): Combinational circuit example: step 0.*

- Output (G) stabilizes at time 61 or earlier. That is a constraint; therefore, transitions at or after time 61 at G violate the timing requirements.

A graph of constraints is built isomorphic to the circuit, as mentioned in Section 3.2.4. The constraint associated with a logic gate is formulated around its forward logic function and its partial inverse operating on sets of waveforms. The constraint system resolution is depicted in Figures 27-34.

- **Step 0**
  Initially assign the set of all possible waveforms $(0|_{-\infty}^{+\infty}, 1|_{-\infty}^{+\infty})$ to the internal nets C, D, E, and F. Inputs A and B are restricted to waveforms stable after time 0: $(0|_{-\infty}^{0}, 1|_{-\infty}^{0})$. The output G is restricted to what violates the timing requirements: transitions at or after time 61: $(0|_{61}^{+\infty}, 1|_{61}^{+\infty})$.

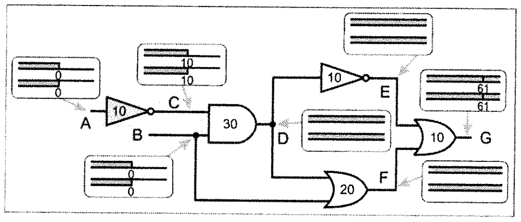


*Figure 28*
*Combinational circuit example: step 1.*

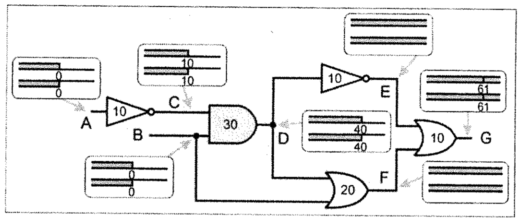


*Figure 29*
*Combinational circuit example: step 2.*

- **Step 1**
  Applying the inverter constraint operating on {A, C} removes from C (narrows C) the waveforms having transitions after time 10. Obviously, transitions after 10 at C cannot be caused by a signal that is stable after 0 at A, provided the inverter between A and C has a maximal delay of 10.

- **Step 2**
  Applying the AND constraint operating on {B, C, D} removes from D the waveforms having transitions after time 40.

- **Step 3**
  Applying the Inverter constraint operating on {D, E} removes from E the waveforms having transitions after time 50.

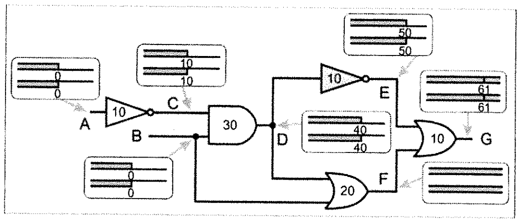


*Figure 30*
*Combinational circuit example: step 3.*

- **Step 4**

  Applying the OR constraint operating on {B, D, F} removes from F the waveforms having transitions after time 60.

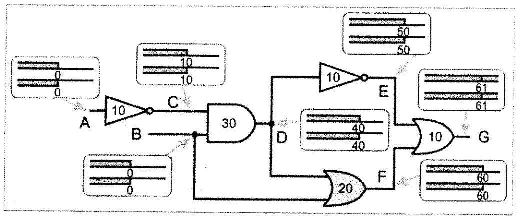


*Figure 31*
*Combinational circuit example: step 4.*

- **Step 5**

  The OR constraint operating on {E,F,G} has a more complex behavior as the gate's partial inverse function comes into play:

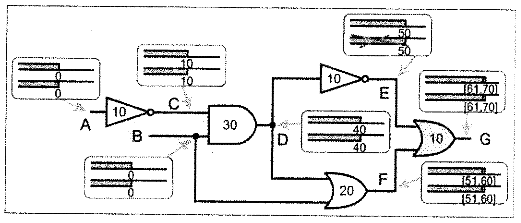


*Figure 32*
*Combinational circuit example: step 5.*

**Forward function:** since the gate maximal delay is 10, and waveforms at its inputs are stable after time 60, applying the constraint removes from G the waveforms having transitions after 70 (the circuit topological delay).

**Partial inverse:** any waveform in E that is stable at 1 (controlling value for OR) after time 50 causes waveforms at G to be stable after 60, none of which is in the domain of G. Therefore these waveforms do not make part of any solution, they are removed. Furthermore, waveforms at F stable at time 51 and after cause waveforms at G to be stable at 61 and after, none of which is in the domain of G. Therefore, they are removed form F which ends-up with waveforms stable after 60, having transitions in [51,60].

It is important to get familiar with the meaning of narrowing: Step 5 narrowed the domain of E from $(0|_{-\infty}^{50}, 1|_{-\infty}^{50})$ to $(0|_{-\infty}^{50}, \phi)$, removing the waveforms stabilizing at 1. Then it narrowed the domain of F from $(0|_{-\infty}^{60}, 1|_{-\infty}^{60})$ to $(0|_{51}^{60}, 1|_{51}^{60})$, removing the waveforms stabilizing at or after 51. Narrowing $v|_{lmin}^{max}$ either decreases the value of *max* (removes late transitions), or increases the value of *lmin* (removes early stabilization), or both.

- **Step 6**
  Applying the OR constraint operating on {B,D,F} involves also its partial inverse function: waveforms in the domain of B stabilizing at 1 (controlling for OR) are removed as they cause at F waveforms stable after time 20, none of which is in its domain. Also, waveforms of D stabilizing at time 31 or earlier are removed as they cause waveforms at F having no transitions at or after time 51.

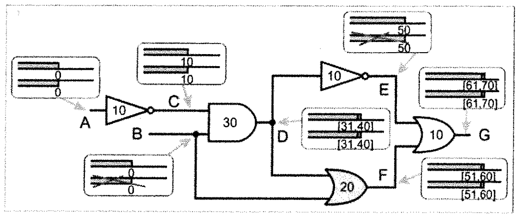


***Figure 33***
*Combinational circuit example: step 6.*

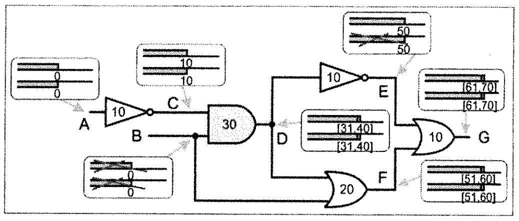


***Figure 34***
*Combinational circuit example: step 7.*

- **Step 7**
  Finally, applying the AND constraint operating on {B,C,D} removes from B the waveforms stabilizing at 0 (controlling for AND) as they cause waveforms at D stable after time 30, none of which is in its domain. We end up with an empty domain. Therefore, the constraint system has no solution, and it is impossible to violate the timing requirements in the context of the current operating conditions (inputs stable after time 0). This proves that no transition is possible at or after time 61, consequently, the path A-C-D-F-G of length 70 is false.

It is important to note that the system is resolved using an event driven mechanism. Each time a domain is narrowed, all the constraints operating on it are re-applied. The operating conditions and the timing constraints trigger the initial operations. The system then iterates until no domain is narrowed further, until the greatest fixpoint is reached.

In this example, only the maximal gate delay is used, and no clock signal is involved. The objective was to illustrate false path elimination using the waveform narrowing formalism. The following example illustrates how component delay correlation is handled through a simple example involving no complex logic functionality.

## 3.4 Sequential Circuit Example

Fig. 35 illustrates a simple frequency divider circuit composed of one edge sensitive flip-flop, one inverter, and one clock buffer. The logic behavior is as follows: Q flips its value at each rising edge of the clock, resulting in a 50% duty cycle signal at Q having half the clock frequency. Considering the physical components behavior, the timing constraints of the flip-flop (setup / hold) have to be satisfied; otherwise, the circuit response is unpredictable. The setup constraint verification is illustrated next, using the following parameters:

- Clock period: 22
- Clock buffer delay: [10,20]
- Inverter delay: [10,20]
- CLK-to-Q flip-flop delay: 0
- D / CLK setup constraint: 0
  D has to stabilize at least 0 time units before the occurrence time of the clock edge.

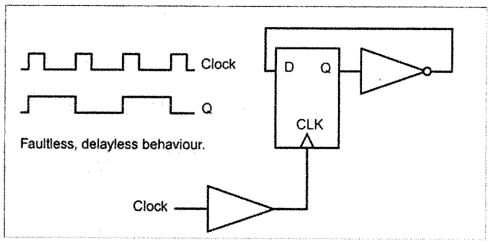


***Figure 35***
*Frequency divider.*

- Component delay correlation factor: 10%
  This means that, although the circuit delays can have values in [10,20], once one is specified as d, the other is in [d - 1, d + 1]. (10% of [10,20] interval width is 1).

As explained in Section 1.3.3, two clock cycles are unfolded. The flip-flop is broken down into two parts: one injects a logic value at the arrival time of the active clock edge of one cycle; the other samples the data at the next clock cycle (after one period).

The constraint system that is used to model the verification of the setup constraint of the flip-flop is depicted in Fig. 36. The clock period is modeled by a buffer that has a fixed delay value equal to the clock period of 22. Since the logic function is very simple, the graphical representation of the data signal domain shows only the dynamic behavior. Three domain types are used in this example:

- Interval delay domains, shown on the clock buffer and the inverter;

- Clock domains representing the interval of uncertainty of the arrival time of the rising edge. The clock falling edge is irrelevant in this particular case.
- Abstract signal domains for Q and D.

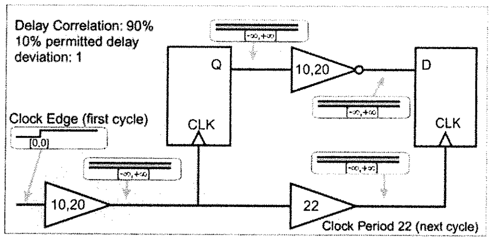


***Figure 36***
*Sequential circuit example: step 0.*

Initial domain values are specified as follows:

- Q and D contain all possible waveforms: $(0|_{-\infty}^{+\infty}, 1|_{-\infty}^{+\infty})$.
- Internal clock domains, CLK at both cycles, contain all possible rising edges $[-\infty\ ,\ +\infty]$.
- The primary input clock domain contains a single rising edge at time 0.
- Delay domains for the inverter and the clock buffer contain all possible delay values [10,20].

**Constraint Operator of the delayless Flip-Flop D/CLK half:** when the flip-flop D / CLK constraint (the part that samples data the next cycle) is applied, it removes from the domains of D and CLK the values that do not violate its setup constraint. It removes from CLK the edges arriving after the stabilization time of D, and removes from D the waveforms stabilizing before the earliest arrival time of the clock edge at CLK.

**Constraint Operator of the delayless Flip-Flop Q/CLK half:** when the flip-flop CLK / Q constraint is applied, it removes from Q the waveforms unstable after the latest clock edge, and removes from CLK the edges arriving before the latest transition at Q. The rational behind this definition is that the data signal at Q stabilizes CLK-to-Q units of time (0 in this case) after the arrival time of the active clock edge. Therefore, the latest transition at Q cannot happen after the arrival time of the clock edge.

Figures 37-56 depict the constraint system resolution. Each step explains the results of applying the shaded constraint.

- **Step 1**

  Applying the clock buffer constraint operator narrows the CLK domain to [10,20].

- **Step 2**

  CLK / Q constraint operator removes from Q the waveforms that are unstable after time 20.

- **Step 3**

  The inverter constraint operator removes from D the waveforms that are unstable after time 20 + 20 = 40.

- **Step 4**

  The period delay narrows the next cycle clock domain to 22 + [10,20] = [32,42].

- **Step 5**

  The D / CLK setup constraint operator has effect on both domains, D and CLK. The setup constraint is violated only when the clock edge occurs before the stabilization time of D. Therefore, clock edges occurring in ]40,42] do not violate the setup constraint, and thus the domain of CLK is narrowed to [32,40]. The same is true for the waveforms at D stabilizing before time 32 (before the arrival time of any clock edge). Therefore, the domain of D is narrowed to $(0|_{32}^{40}, 1|_{32}^{40})$. In fact, the

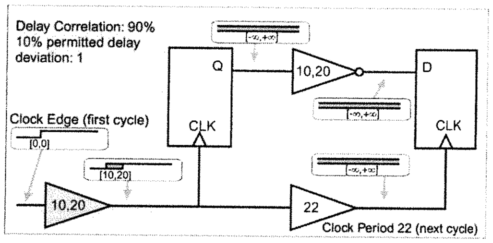


***Figure 37***
*Sequential circuit example: step 1.*

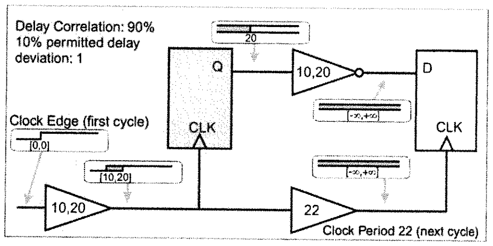


***Figure 38***
*Sequential circuit example: step 2.*

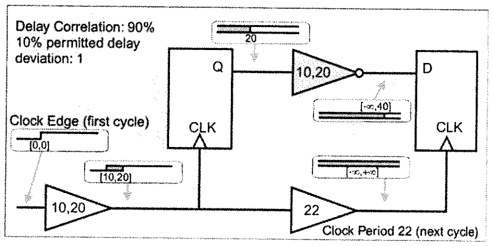


***Figure 39***
*Sequential circuit example: step 3.*

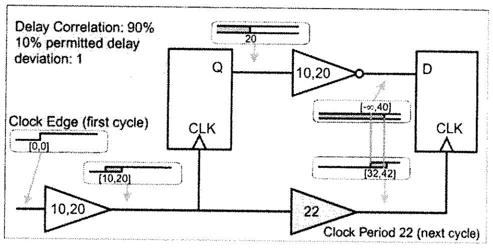


***Figure 40***
*Sequential circuit example: step 4.*

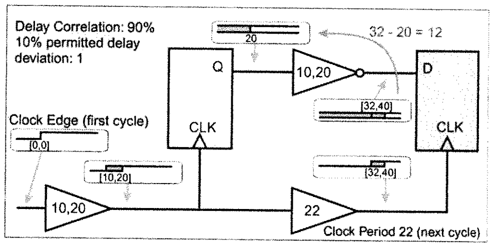


***Figure 41***
*Sequential circuit example: step 5.*

(delayless) setup constraint operator does an interval intersection operation between the domains of D and CLK.

- **Step 6**

  The inverter constraint operator has a backward effect on the domain of Q. Since all waveforms at D have transitions at or after time 32, the waveforms at Q should all have transitions at or after 32 - 20 = 12. Therefore, the waveforms at Q stabilizing before time 12 are removed; the domain of Q is narrowed to $(0|_{12}^{20}, 1|_{12}^{20})$.

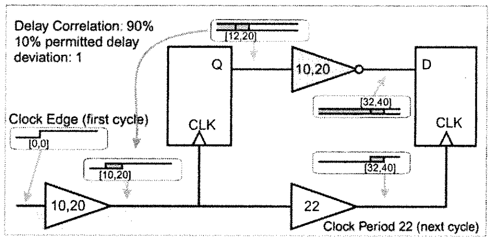


***Figure 42***
*Sequential circuit example: step 6.*

- **Step 7**

  The CLK / Q constraint operator has also a backward effect on the domain of CLK. Since all the waveforms at Q have transitions at or after time 12, the clock edge cannot happen before 12. Therefore [10,12[ is removed from the domain of CLK.

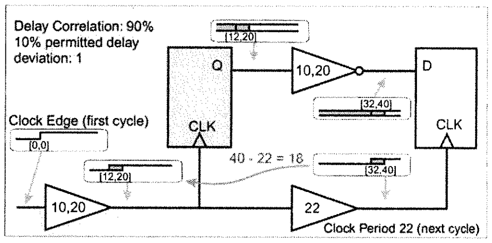


***Figure 43***
*Sequential circuit example: step 7.*

- **Step 8**

  The clock period constraint operator also has a backward effect on the clock domain of the previous cycle. Remember that the clock period is 22, and the time lapse between successive rising edges of the clock is exactly 22 time units. The latest arrival time of the clock on the next cycle is 40; therefore, the latest arrival time of the clock at the previous cycle is 40 - 22 = 18. It follows that the clock signal domain must be narrowed to [12,18].

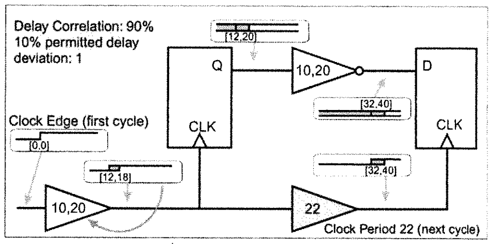


***Figure 44***
*Sequential circuit example: step 8.*

- **Step 9**

  The buffer delay constraint operator affects its delay domain: a delay value in [10,12[ or ]18,20] causes the clock edges to fall outside of [12,18]. Therefore the buffer delay domain should be narrowed to [12,18].

- **Step 10**

  At this point, the delay correlation factor of 10% comes into play. Since delays cannot differ by more than 1 unit of time, a delay of [12,18] at the clock buffer implies an inverter delay in [12,18] + [-1,+1] = [11,19].

- **Step 11**

  The CLK / Q constraint operator removes from Q the waveforms unstable after 18, the latest arrival time of the clock.

- **Step 12**

  The inverter constraint operator removes from D the waveforms unstable after 18 + 19 = 37.

- **Step 13**

  The clock period constraint operator removes from the domain of the clock on the next cycle the edges occurring in [32,34[, since the earliest edge on the previous cycle occurs at time 12.

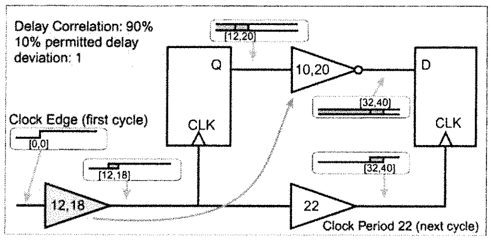


***Figure 45***
*Sequential circuit example: step 9.*

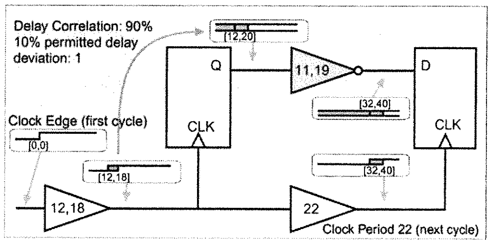


***Figure 46***
*Sequential circuit example: step 10.*

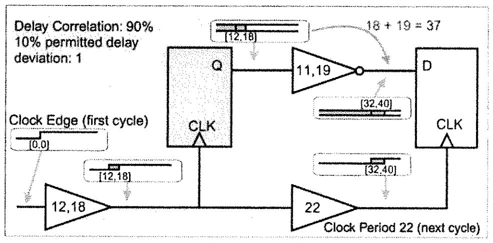


***Figure 47***
*Sequential circuit example: step 11.*

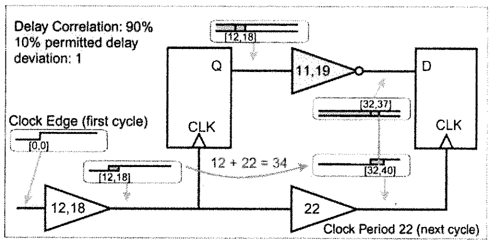


***Figure 48***
*Sequential circuit example: step 12.*

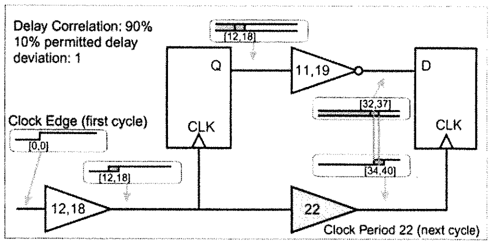


***Figure 49***
*Sequential circuit example: step 13.*

- **Step 14**

  D / CLK setup constraint operator narrows D and CLK to their interval intersections: $(0|_{34}^{37}, 1|_{34}^{37})$ for D, and [34, 37] for CLK.

- **Step 15**

  The inverter constraint operator removes from Q the waveforms that are stable before 34 - 19 = 15. Q becomes $(0|_{15}^{18}, 1|_{15}^{18})$.

- **Step 16**

  The clock period constraint operator removes from the clock domain of the previous cycle the edges occurring after 37 - 22 = 15.

- **Step 17**

  CLK / Q constraint operator removes from Q the waveforms unstable after time 15, and removes from CLK the edges occurring before 15.

- **Step 18**

  Once again, the clock buffer delay domain is affected; it is narrowed to the single value of 15, as the clock domain at its output contains the single edge occurring at time 15.

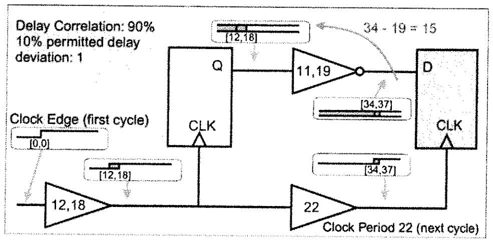


***Figure 50***
*Sequential circuit example: step 14.*

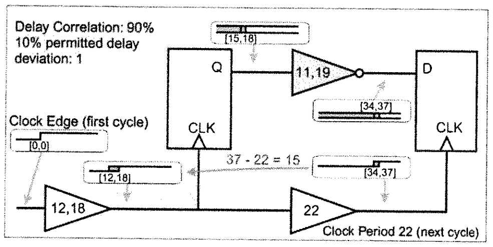


***Figure 51***
*Sequential circuit example: step 15.*

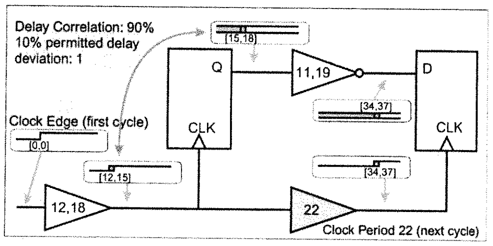


***Figure 52***
*Sequential circuit example: step 16.*

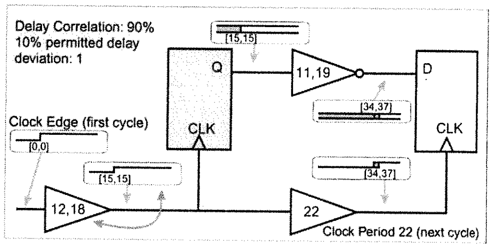


***Figure 53***
*Sequential circuit example: step 17.*

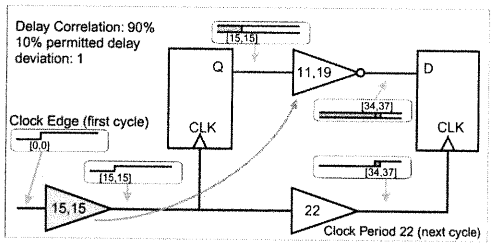


***Figure 54***
*Sequential circuit example: step 18.*

- **Step 19**

  The delay correlation of 10% narrows the inverter delay domain to [15,15] + [-1,+1] = [14,16].

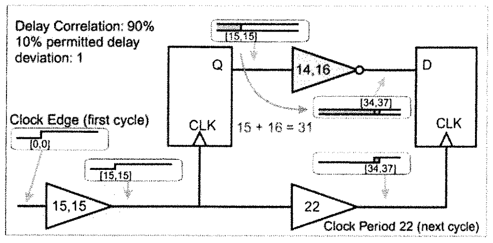


*Figure 55*
*Sequential circuit example: step 19.*

- **Step 20**

  Applying the inverter constraint operator narrows the domains of Q and D to $\phi$. In fact, waveforms at Q that are stable after time 15 cause waveforms at D to be stable after time 15 + 16 = 31, none of which is in the domain of D. Therefore, we can conclude that the constraint system has no solutions, and the setup constraint of the flip-flop is satisfied.

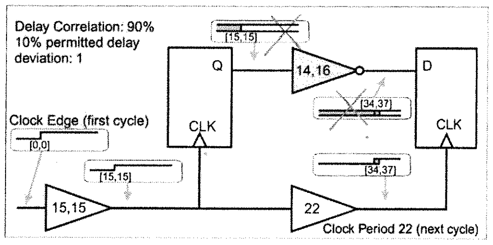


*Figure 56*
*Sequential circuit example: step 20.*

Fig. 57 illustrates graphically the domains evolution for steps 6 to 20. The height of the shaded area represents the domain interval width. Note that during evaluation, domains never get enlarged. They either remain the same or get narrowed.

**Summary:** to prove that the setup constraint of the flip-flop is satisfied, the constraint system narrows the domains by keeping only the values that violate the constraint. If we end up with empty domains, we can conclude that the setup constraint is satisfied.

The next Section formalizes the notion of a relational constraint operator, and defines constraint operators for the basic symmetrical logic gates such as AND, OR, XOR, etc.

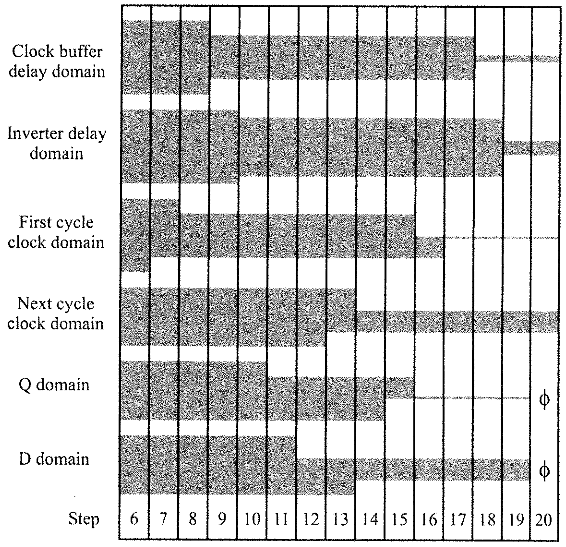


*Figure 57*
*Waveform narrowing for steps 6 through 20.*

## 3.5 Relational Constraints

The relational constraints defined on abstract signals, clock domains, and delay domains make extensive use of interval arithmetic and set operations such as union and intersection.

**Interval arithmetic:**

A real interval $[a, b]$ is a subset of $R$ defined as $[a, b] = \{x \in R | (x \geq a \wedge x \leq b)\}$. When $a > b$ the interval is empty, denoted $\phi$.

Interval operators are derived from arithmetic and set operators. For an arithmetic operator $\otimes$ that can be anything like $-$ , $+$ , $\div$ , or $\times$ , the corresponding interval arithmetic operator is: $[a, b] \otimes [c, d] = \{m = x \otimes y \ | \ x \in [a, b] \wedge y \in [c, d]\}$.

- **Addition:**
  $[a, b] + [c, d] = [a + c, b + d]$ when both intervals are not empty. Otherwise, $[a, b] + \phi = \phi + [a, b] = \phi$.
  $k + [a, b] = [a, b] + k = [k, k] + [a, b] = [k + a, k + b]$ when $[a, b]$ is not empty. Otherwise, $k + \phi = \phi + k = \phi$.

- **Negation:**
  $-[a, b] = [-b, -a]$ when $[a, b]$ is not empty. Otherwise, $-\phi = \phi$.

- **Subtraction:**
  $[a, b] - [c, d] = [a, b] + (-[c, d])$

  **Examples:**

  $[1, 5] + [2, 4] = [3, 9]$

  $[3, 9] - [2, 4] = [-1, 7]$. Note that the result is "larger" than $[1, 5]$.

Note that interval operators corresponding to arithmetic commutative operators are also commutative.

- **Union:**
  $[a, b] \cup [c, d] = [min(a, c), max(b, d)]$ when both intervals are not empty. Otherwise, $[a, b] \cup \phi = \phi \cup [a, b] = [a, b]$.

  Note that the interval union is not equivalent to the set union. For instance, $[1, 2] \cup [5, 6] = [1, 6]$, a set union does not include $]2, 5[$. In fact, the interval union is defined as to allow representing the result as an interval.

- **Intersection:**
  $[a, b] \cap [c, d] = [max(a, c), min(b, d)]$ when both intervals are not empty. Otherwise, $[a, b] \cap \phi = \phi \cap [a, b] = \phi$.

**Domain Operators:**

Arithmetic and set operators are defined on signal (clock, data) and delay domains (intervals) using the interval operations defined previously. Since an empty set ($\phi$) results when any of the operands is empty, the following defines the cases when the operands are non-empty:

- **Addition:**
  **Abstract Waveform + Delay:** $(AW \times RealIntervals) \rightarrow AW$
  For $v|_{lmin}^{max} \in AW$, $[d_{min}, d_{max}] \in (R^{+} \times R^{+})$ | $d_{min} \le d_{max}$ and $lmin \le max$
  $v|_{lmin}^{max} + [d_{min}, d_{max}] = v|_{lmin + d_{min}}^{max + d_{max}}$.
  It corresponds to the interval addition between $[lmin, max]$ and $[d_{min}, d_{max}]$.

  **Abstract Signal + Delay:** $(AS \times RealIntervals) \rightarrow AS$
  For an *abstract signal S* and a delay $[d_{min}, d_{max}] \in (R^{+} \times R^{+})$,
  $S + [d_{min}, d_{max}] = (S.w_0 + [d_{min}, d_{max}], S.w_1 + [d_{min}, d_{max}])$.

- **Subtraction:**
  **Abstract Waveform - Delay:** $(AW \times RealIntervals) \rightarrow AW$
  For $v|_{lmin}^{max} \in AW$, $[d_{min}, d_{max}] \in (R^{+} \times R^{+})$ | $d_{min} \le d_{max}$ and $lmin \le max$
  $v|_{lmin}^{max} - [d_{min}, d_{max}] = v|_{lmin - d_{max}}^{max - d_{min}}$.

It corresponds to the interval subtraction between $[lmin, max]$ and $[d_{min}, d_{max}]$.

**Abstract Signal - Delay:** $(AS \times RealIntervals) \rightarrow AS$

For an *abstract signal S* and a delay $[d_{min}, d_{max}] \in (R^{+} \times R^{+})$,

$S - [d_{min}, d_{max}] = (S.w_0 - [d_{min}, d_{max}], S.w_1 - [d_{min}, d_{max}])$.

- **Union:**

**Abstract Waveforms:** $(AW \times AW) \rightarrow AW$

For $v|_{lmin_1}^{max_1}, v|_{lmin_2}^{max_2} \in AW, v \in \{0, 1\}$,

$v|_{lmin_1}^{max_1} \cup v|_{lmin_2}^{max_2} = v|_{min(lmin_1, lmin_2)}^{max(max_1, max_2)}$

It corresponds to the interval union between their *last transition intervals*.

**Abstract Signals:** $(AS \times AS) \rightarrow AS$

For the *abstract signals* $S_1, S_2$,

$S_1 \cup S_2 = (S_1.w_0 \cup S_2.w_0, S_1.w_1 \cup S_2.w_1)$.

- **Intersection:**

**Abstract Waveforms:** $(AW \times AW) \rightarrow AW$

For $v|_{lmin_1}^{max_1}, v|_{lmin_2}^{max_2} \in AW, v \in \{0, 1\}$,

$v|_{lmin_1}^{max_1} \cap v|_{lmin_2}^{max_2} = v|_{max(lmin_1, lmin_2)}^{min(max_1, max_2)}$

It corresponds to the interval intersection between their *last transition intervals*.

**Abstract Signals:** $(AS \times AS) \rightarrow AS$

For the *abstract signals* $S_1, S_2$,

$S_1 \cap S_2 = (S_1.w_0 \cap S_2.w_0, S_1.w_1 \cap S_2.w_1)$.

**Note**: for an *Abstract signal S, $S.w_0$ ($S.w_1$)* is used to denote both the *abstract waveform $S.w_0$ ($S.w_1$)* and the real interval $[S.w_0.lmin, S.w_0.max]$ ($[S.w_1.lmin, S.w_1.max]$).

**Relational Constraint Operators:**

**Definition 1:** let $S_1, S_2, S_3, ..., S_n$ *be n* sets, and *f* a relation $f \subseteq S_1 \times S_2 \times ... \times S_n$.

The ***complete relational constraint operator*** based on $f$ is a mapping $\hat{C}_f : 2^{S_1} \times 2^{S_2} \times ... \times 2^{S_n} \to 2^{S_1} \times 2^{S_2} \times ... \times 2^{S_n}$ defined as follows:

for $D_1 \subseteq S_1, D_2 \subseteq S_2, ..., D_n \subseteq S_n$,

$\hat{C}_f(D_1, D_2, ..., D_n) = (\hat{D}_1, \hat{D}_2, ..., \hat{D}_n)$ such that, for each $i \in \{1, 2, ..., n\}$

$\hat{D}_i = \{x_i \in D_i \mid \text{for all } j \neq i\ \exists x_j \in D_j, (x_1, x_2, ..., x_n) \in f\}$.

$\{x \in D_i \mid x \notin \hat{D}_i\}$ is the set of ***incompatible*** values of the domain $D_i$ in the context of the relation *f*. In fact, when applied to a set of domains, the relational constraint operator is said to strip out their *incompatible* values. $\hat{D}_i$ is the set of values ***compatible*** with the relation *f*. Note that, in the context of a constraint system, a value in $\hat{D}_i$ is not necessarily part of a solution because all constraints have to be satisfied not just $\hat{C}_f$.

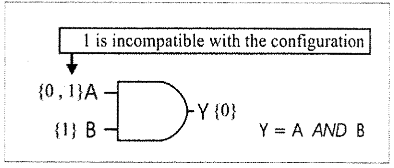


*Figure 58*
*An incompatible value.*

**Example:**

Let $f = \{(0, 0, 0), (0, 1, 0), (1, 0, 0), (1, 1, 1)\}$. In fact $f$ represents the logic function *AND*, and its elements are $(x, y, AND(x, y))$.

Referring to Fig. 58, $D_A = \{0, 1\}$, $D_B = \{1\}$, and $D_Y = \{0\}$

$\hat{C}_{AND}(\{0, 1\}, \{1\}, \{0\}) = (\{0\}, \{1\}, \{0\})$ because

$(0, 1, 0) \in AND$, but $(1, 1, 0) \notin AND$. In fact, no value in B can be combined with the value 1 in A so that Y results in the value 0, and therefore, the value 1 in A is incompatible with the constraint.

**Property:** let $S_1, S_2, S_3, ..., S_n$ *n* sets, a relation $f \subseteq S_1 \times S_2 \times ... \times S_n$ and $\hat{C}_f$ its ***complete relational constraint operator.*** $\forall\ D_1 \subseteq S_1, D_2 \subseteq S_2, ..., D_n \subseteq S_n$, and $D_i = D_{i1} \cup D_{i2}$,

$\hat{C}_f(D_1, ..., D_i, ..., D_n) = \hat{C}_f(D_1, ..., D_{i1}, ..., D_n) \cup \hat{C}_f(D_1, ..., D_{i2}, ..., D_n)$.

The union of $(E_{1,1}, ..., E_{1,n}), (E_{2,1}, ..., E_{2,n}) \in 2^{S_1} \times ... \times 2^{S_n}$ is defined as $(E_{1,1} \cup E_{2,1}, ..., E_{1,n} \cup E_{2,n})$.

**Definition 2:** let $S_1, S_2, ..., S_n$ be $n$ sets, and $f$ a relation $f \subseteq S_1 \times S_2 \times ... \times S_n$, and let $\hat{C}_f : 2^{S_1} \times 2^{S_2} \times ... \times 2^{S_n} \to 2^{S_1} \times 2^{S_2} \times ... \times 2^{S_n}$ be its *complete constraint operator.* Let $E_1 \subseteq 2^{S_1}, E_2 \subseteq 2^{S_2}, ..., E_n \subseteq 2^{S_n}$ such that $\forall i \in \{1, 2, ..., n\}$, $\forall A \in 2^{S_i}$, $\exists B \in E_i \mid A \subseteq B$; and let $C_f$ be a mapping $C_f : E_1 \times E_2 \times ... \times E_n \to E_1 \times E_2 \times ... \times E_n$. $C_f$ is said to be an ***optimal constraint operator*** based on $f$ iff for every $(D_1, D_2, ..., D_n) \in E_1 \times E_2 \times ... \times E_n$ the following holds:

$$\text{for } \hat{C}_f(D_1, D_2, ..., D_n) = (\hat{D}_1, \hat{D}_2, ..., \hat{D}_n)$$
$$\text{and } C_f(D_1, D_2, ..., D_n) = (D'_1, D'_2, ..., D'_n)$$
$$\forall i \in \{1, 2, ..., n\},\ \hat{D}_i \subseteq D'_i \ \wedge \ \forall X \in E_i,\ \hat{D}_i \subseteq X \Rightarrow D'_i \subseteq X.$$

Put in intuitive terms, $D'_i$ is the unique "smallest" member of $E_i$ that contains $\hat{D}_i$. Note that, when it exists, an optimal constraint operator is unique.

The concept of optimal constraint operator is important for cases where it is not possible to represent efficiently members of $2^{S_i}$. For example, in the context of timing verification where logic functions are defined on binary waveforms, the logic constraints are defined on the space of *abstract waveforms* (*AW*) instead of the space of *binary waveforms* (*BW*). The reason for this is practical: representing arbitrary members of $2^{BW}$ may require excessive space, whereas, representing an *abstract waveform* require one bit for the final stable value and two numbers for the *last-transition-interval* (see Section 3.2.1). In this case, it is not always possible for constraint operators to strip out all the *incompatible* values (waveforms) simply because the resulting binary waveform set has to be represented as an *abstract waveform*. Therefore, it is desirable to define *optimal constraint operators* on abstract waveforms.

### 3.5.1 Delay Constraints

**Delay Function:** a delay function is a mapping $del : BW \times R^+ \to BW$, for $f \in BW$ and $d \in R^+$, $del(f, d) = g$ such that $\forall t \in R$, $g(t) = f(t-d)$.

Example: let $f \in BW$ such that

$$f(t) = 0 \text{ for } t < 0$$
$$f(t) = 1 \text{ for } t \geq 0$$

$del(f, 5) = g$ defined as

$g(t) = 0$ for $t < 5$

$g(t) = 1$ for $t \geq 5$ (see Fig. 59)

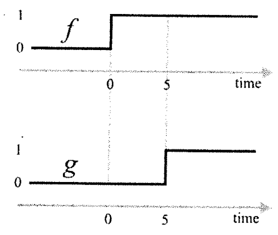


***Figure 59***
*A delayed binary waveform.*

Delay elements in circuits propagate data and clock signals. The following defines delay constraints operating on clock and data domains.

**Clock Delay Constraint Operator:** the clock delay constraint operator operates on three domains (see Fig. 60):

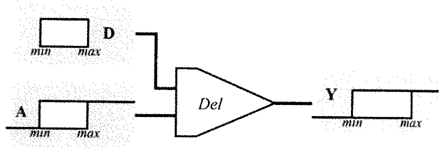


***Figure 60***
*Clock delay constraint.*

1) $A$ is the domain corresponding to the clock signal driving the input of the delay element. It is the set of clock edges, represented as an interval (each clock edge is represented by its occurrence time).

2) $Y$ is the domain corresponding to the clock signal at the output of the delay element, also represented as an interval.

3) $D$ is the domain of the delay values, represented as an interval.

Let's calculate the bounds for each domain, considering only the other two:

$$Y' \subseteq \{del(a, d) | (a \in A), d \in D\}$$
$$Y' \subseteq \{a + d | (a \in A), d \in D\}$$
$$Y' \subseteq A + D \qquad (1)$$

$$A' \subseteq \{a \in R \ | del(a, d) \in Y \text{ for } d \in D\}$$
$$A' \subseteq \{y - d \ | \ y \in Y \text{ and } d \in D\}$$
$$A' \subseteq Y - D \qquad (2)$$

$$D' \subseteq \{d \in D \mid d + a = y \text{ for } a \in A \text{ and } y \in Y\}$$
$$D' \subseteq \{y - a \mid y \in Y \text{ and } d \in D\}$$
$$D' \subseteq Y - A \qquad (3)$$

The clock delay constraint operator is a direct consequence of (1), (2), and (3):

$$\boxed{\begin{aligned} &ClockDelay(A, D, Y) = (A', D', Y') \text{ such that,}\\ &\quad A' = A \cap (Y - D)\\ &\quad Y' = Y \cap (A + D)\\ &\quad D' = D \cap (Y - A) \end{aligned}}$$

**Property:** the clock delay constraint operator is *optimal.*

Let $\hat{C}_{del}$ be the complete clock delay constraint operator. By following the same steps as for *ClockDelay*, we find that $\ddot{C}_{del}(A, D, Y) = (A', D', Y')$. In fact, when the domains of $A$, $B$, and $Y$ are initially intervals, the delay constraint operator is equivalent to the complete relational constraint operator based on *del.*

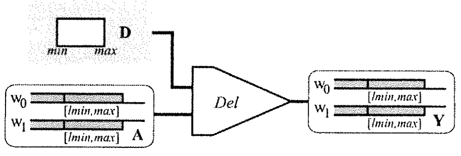


*Figure 61*
*Data delay constraint.*

**Data Delay Constraint Operator:** the data delay constraint operator operates on three domains (see Fig. 61):

1) $A$ is the domain corresponding to the data signal driving the input of the delay element. It is the set of binary waveforms represented as an *abstract signal* (See section 3.2.2). $A.w_0$ ($A.w_1$) is the set of binary waveforms in A stabilizing at 0 (1) having at least one transition in the interval $A.w_0$ ($A.w_1$).

2) $Y$ is the domain corresponding to the data signal at the output of the delay element, also represented as an *abstract signal.* $Y.w_0$ ($Y.w_1$) is the set of binary waveforms in $Y$ stabilizing at 0 (1), having at least one transition in the interval $Y.w_0$ ($Y.w_1$).

3) $D$ is the domain of the delay values, represented as an interval.

The delay relation is the same for clock delay, except it is defined on *abstract signal* domains for $A$ and $Y$. Let $C_{del}$ be its *constraint operator.* We have

$$A = A.w_0 \cup A.w_1 \text{ and } Y = Y.w_0 \cup Y.w_1 \text{. Therefore,}$$
$$C_{del}(A, D, Y) = C_{del}(A.w_0, D, Y) \cup C_{del}(A.w_1, D, Y)$$
$$C_{del}(A.w_0, D, Y) = C_{del}(A.w_0, D, Y.w_0) \cup C_{del}(A.w_0, D, Y.w_1)$$
$$C_{del}(A.w_1, D, Y) = C_{del}(A.w_1, D, Y.w_0) \cup C_{del}(A.w_1, D, Y.w_1)$$

Since waveforms stabilizing at 0 (1) in $A$ result in waveforms stabilizing at 0 (1) in $Y$ we have:

$$C_{del}(A.w_0, D, Y.w_1) = (\phi, \phi, \phi)$$
$$C_{del}(A.w_1, D, Y.w_0) = (\phi, \phi, \phi)$$

Therefore,

$$C_{del}(A, D, Y) = C_{del}(A.w_0, D, Y.w_0) \cup C_{del}(A.w_1, D, Y.w_1)$$

The delay constraint operator defined on *abstract signals* is a composition of the same operator defined on abstract waveforms. Let's calculate the bounds for the domains of $C_{del}(A.w_0, D, Y.w_0)$:

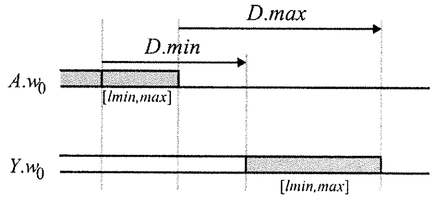


***Figure 62***
*Effect on Y.w₀.*

1) Effect on Y:

Let $a \in A.w_0$ then

$$a(t) = 0 \text{ for } t > A.w_0.max$$
$$\exists\, t_1 \in A.w_0 \mid a(t_1) = 1$$

for $d \in D$, the waveform in $Y$ compatible with $a$ and $d$ is $y$ defined as:

$y(t) = a(t-d)$. Therefore,

$$y(t) = 0 \text{ for } t > A.w_0.max + d$$
$$y(t_1 + d) = 1$$

In other terms, since any waveform in $A$ is stable at 0 after time $A.w_0.max$, any waveform in $Y$ having transitions after time $A.w_0.max + D.max$ is incompatible (see Fig. 62). Also, any waveform in $Y$ stabilizing before time $A.w_0.lmin + D.min$ is incompatible. Therefore,

$$Y'.w_0 \subseteq A.w_0 + D$$

2) Effect on $A$: Since any waveform in $Y.w_0$ is stable after time $Y.w_0.max$, any waveform in $A.w_0$ having transitions after $Y.w_0$.max - $D.min$ is incompatible (see Fig. 63). Also any waveform in $A.w_0$ stabilizing before $Y.w_0.lmin$ - $D.max$ is incompatible.

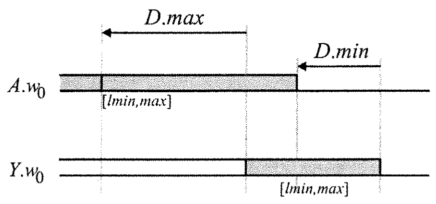


***Figure 63***
*Effect on A.w*$_0$.

Therefore,

$$A'.w_0 \subseteq Y.w_0 - D$$

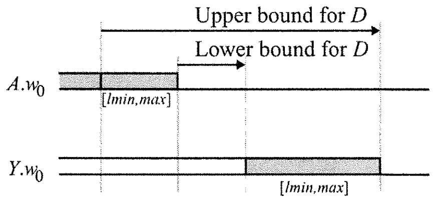


***Figure 64***
*Effect on D.*

3) Effect on $D$: Any delay value (in $D$) less than $Y.w_0.lmin$ - $A.w_0.max$ is incompatible as it results in waveforms stabilizing before $Y.w_0.lmin$ in $Y$ (see Fig. 64). Also a delay value greater than $Y.w_0.max$ - $A.w_0.lmin$ is incompatible as it results in waveforms having transitions after $Y.w_0.max$ in $Y$. Therefore,

$$D' \subseteq Y.w_0 - A.w_0$$

Finally, the data delay constraint operator is defined on $A$, $D$, and $Y$ as follows:

$$\boxed{\begin{aligned} &DataDelay(A, D, Y) = (A', D', Y') \text{ such that,} \\ &\quad Y' = Y \cap ((Y.w_0 + D), (Y.w_1 + D)) \\ &\quad A' = A \cap ((Y.w_0 - D), (Y.w_1 - D)) \\ &\quad D' = D \cap ((Y.w_0 - A.w_0) \cup (Y.w_1 - A.w_1)) \end{aligned}}$$

**Property:** the data delay constraint operator is *optimal.*

### 3.5.2 Delayless Gate Constraints

This section defines the constraint operators associated with delayless symmetrical logic functions BUFFER, INVERTER, AND, and XOR. The other symmetrical functions such as OR, NAND, NOR, and XNOR are derived from AND, XOR and NOT.

### 3.5.2.1 Buffer Gate (Identity)

**Buffer Function:** the buffer function, *buf*, is a mapping $buf: BW \rightarrow BW$, for $f \in BW$ $buf(f) = f$.

**Buffer Constraint Operator:** the buffer constraint operator operates on two domains of the same type (clock or data) (see Fig. 65):

1) $A$ is the domain corresponding to the data or clock signal driving the input of the buffer.

2) $Y$ is the domain corresponding to the data or clock signal at the output of the buffer.

$$\boxed{\begin{array}{l} C_{buf}(A, Y) = (A', Y') \text{ such that:} \\ \quad Y' = Y \cap A \\ \quad A' = A \cap Y \end{array}}$$

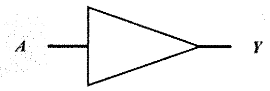


***Figure 65***
*Buffer Constraint.*

**Property:** the Buffer constraint operator is *optimal.*

### 3.5.2.2 Inverter Gate (NOT)

**NOT Function:** the inverter function, NOT, is a mapping $not: BW \rightarrow BW$, for $f \in BW$ $not(f) = g$ such that $g(t) = \overline{f(t)}$ $\forall\, t \in R$.

The inverter function results in waveforms stabilizing at 0 (1) when the input stabilizes at 1 (0).

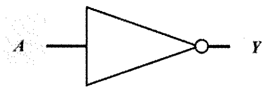


***Figure 66***
*NOT Constraint.*

**NOT Constraint Operator:** the NOT constraint operator operates on two *abstract signal* domains (see Fig. 66):

1) $A$ is the domain corresponding to the data signal driving the input of the inverter.

2) $Y$ is the domain corresponding to the data signal at the output of the inverter.

$C_{NOT}(A, Y) = (A', Y')$ such that:

$$Y'.w_0 = Y.w_0 \cap A.w_1 \qquad Y'.w_1 = Y.w_1 \cap A.w_0$$
$$A'.w_0 = A.w_0 \cap Y.w_1 \qquad A'.w_1 = A.w_1 \cap Y.w_0$$

**Property:** the NOT constraint operator is *optimal.*

The NOT constraint operator is also defined on clock signals, where a rising (falling) transition at the input results in a falling (rising) transition at the output. In such case, the clock signal domain is defined as two sets of transitions: rising, and falling ones.

### 3.5.2.3 AND Gate

**AND Function:** the AND function is a mapping $and: BW \times BW \rightarrow BW$, for $a, b \in BW$ $and(a, b) = y$ such that $y(t) = a(t)$ and $b(t)$ $\forall\, t \in R$.

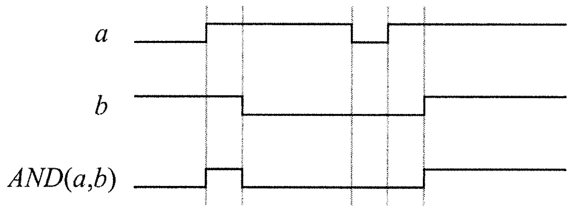


**Figure 67**
*ANDing two binary waveforms.*

Fig. 67 shows an example of ANDing two binary waveforms. Note that a transition at $a$ ($b$) is reflected at $y$ only if $b$ ($a$) is stable at 1 (non-controlling for AND) during the transition.

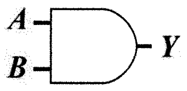


**Figure 68**
*AND Constraint.*

**AND Constraint Operator:** the AND constraint operator operates on three *abstract signal* domains (see Fig. 68):

1) A, B are *abstract signal* domains for the inputs of the AND gate.

2) Y is an *abstract signal* domain of the output of the AND gate.

The AND constraint is much more complex than the previously defined single input gates constraints (Delay, Buffer, and Inverter). The AND constraint is broken down into simple cases of a single *abstract waveform* for each case of possible relative positions of the *last transition intervals*.

$$
\begin{aligned}
C_{AND}(A, B, Y) &= C_{AND}(A.w_0, B, Y) \cup C_{AND}(A.w_1, B, Y)\\
C_{AND}(A.w_0, B, Y) &= C_{AND}(A.w_0, B.w_0, Y) \cup C_{AND}(A.w_0, B.w_1, Y)\\
C_{AND}(A.w_1, B, Y) &= C_{AND}(A.w_1, B.w_0, Y) \cup C_{AND}(A.w_1, B.w_1, Y)\\
C_{AND}(A.w_0, B.w_0, Y) &= C_{AND}(A.w_0, B.w_0, Y.w_0) \cup C_{AND}(A.w_0, B.w_0, Y.w_1)\\
C_{AND}(A.w_0, B.w_1, Y) &= C_{AND}(A.w_0, B.w_1, Y.w_0) \cup C_{AND}(A.w_0, B.w_1, Y.w_1)\\
C_{AND}(A.w_1, B.w_0, Y) &= C_{AND}(A.w_1, B.w_0, Y.w_0) \cup C_{AND}(A.w_1, B.w_0, Y.w_1)\\
C_{AND}(A.w_1, B.w_1, Y) &= C_{AND}(A.w_1, B.w_1, Y.w_0) \cup C_{AND}(A.w_1, B.w_1, Y.w_1)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
C_{AND}(A, B, Y) = \quad & C_{AND}(A.w_0, B.w_0, Y.w_0) \cup\\
& C_{AND}(A.w_0, B.w_0, Y.w_1) \cup\\
& C_{AND}(A.w_0, B.w_1, Y.w_0) \cup\\
& C_{AND}(A.w_0, B.w_1, Y.w_1) \cup\\
& C_{AND}(A.w_1, B.w_0, Y.w_0) \cup\\
& C_{AND}(A.w_1, B.w_0, Y.w_1) \cup\\
& C_{AND}(A.w_1, B.w_1, Y.w_0) \cup\\
& C_{AND}(A.w_1, B.w_1, Y.w_1)
\end{aligned}
$$

$C_{AND}(A.w_0, B.w_0, Y.w_1) = (\phi, \phi, \phi)$ because waveforms at an AND input stabilizing at 0 (controlling value for AND) result in waveforms stabilizing at 0 at the output $Y$.

For similar reasons,

$$
\begin{aligned}
C_{AND}(A.w_0, B.w_1, Y.w_1) &= (\phi, \phi, \phi),\\
C_{AND}(A.w_1, B.w_0, Y.w_1) &= (\phi, \phi, \phi), \text{ and}\\
C_{AND}(A.w_1, B.w_1, Y.w_0) &= (\phi, \phi, \phi)
\end{aligned}
$$

Therefore,

$$
\boxed{
\begin{aligned}
& C_{AND}(A, B, Y) =\\
& \qquad C_{AND}(A.w_0, B.w_0, Y.w_0) \cup\\
& \qquad C_{AND}(A.w_0, B.w_1, Y.w_0) \cup \qquad\qquad \textbf{(A0)}\\
& \qquad C_{AND}(A.w_1, B.w_0, Y.w_0) \cup\\
& \qquad C_{AND}(A.w_1, B.w_1, Y.w_1)
\end{aligned}
}
$$

Each subset is considered separately, and domains are considered non-empty, as the case of empty domains is trivial.

- $C_{AND}(A.w_0, B.w_0, Y.w_0) = (A'.w_0, B'.w_0, Y'.w_0)$

**Effect on $Y.w_0$:**

Fig. 69 enumerates the six possibilities for the overlap of the *last transition intervals* of $A.w_0$ and $B.w_0$. since waveforms in $A$ and $B$ stabilize at 0, waveforms in Y are stable at 0 after time $min(A.w_0.max, B.w_0.max)$. This is trivial since 0 is a controlling value for the AND gate. However, except when $A.w_0 = B.w_0 = 0|_T^T$, it is possible for $Y$ to contain the binary waveform stable at 0 at all time. This happens when we have waveforms $a$ in $A$, and $b$ in $B$ that are never at 1 at the same time. Therefore there is no restriction on whether waveforms in Y have transitions in some interval, and hence,

$$\text{if}(\ A.w_0 = B.w_0 = 0|_T^T\ )\ \text{then}$$
$$Y'.w_0 \subseteq 0|_T^T$$
$$\text{else}$$
$$Y'.w_0 \subseteq 0|_{-\infty}^{min(A.w_0.max,\ B.w_0.max)}$$

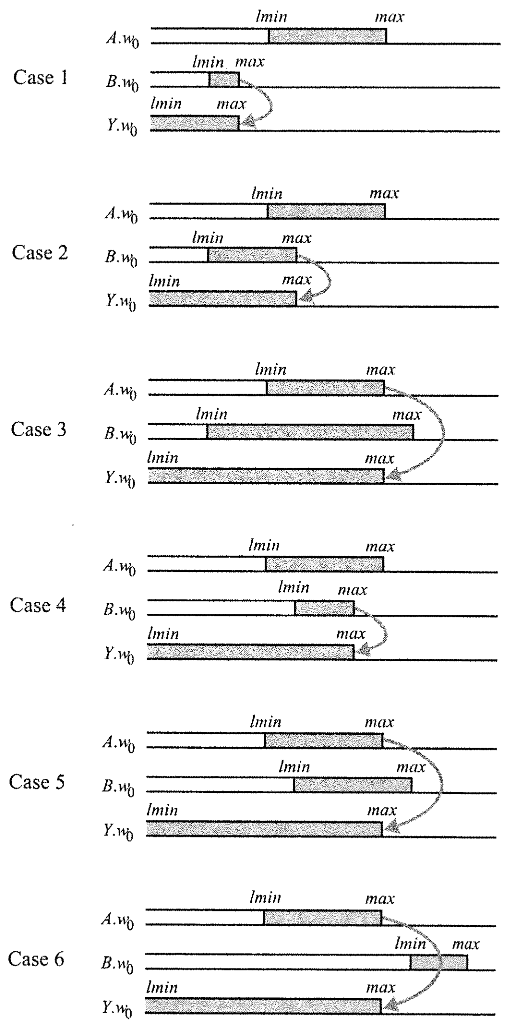


*Figure 69*
*AND Constraint ( A.w0, B.w0, Y.w0 ) - effect on Y.w0.*

**Effect on $A.w_0$ and $B.w_0$:**

referring to Fig. 70, waveforms in $A.w_0$ and $B.w_0$ stabilizing before $Y.w_0.lmin$ are incompatible with the constraint because they result in waveforms

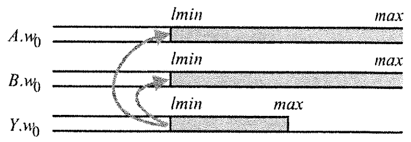


*Figure 70*
*AND Constraint ( A.w0, B.w0, Y.w0 ) - effect on A.w0 and B.w0.*

in $Y$ having no transitions at or after $Y.w_0.lmin$. In fact, let $y \in Y.w_0$, $\exists\, t \in Y.w_0$ such that $y(t) = 1$. Therefore, waveforms $a$ in $A$, and $b$ in $B$ compatible with $y$ are such that $a(t) = 1$ and $b(t) = 1$. On the other hand, there is no restriction on whether $a$ or $b$ has to stabilize after a certain time, because, for example, while $a$ stabilizes before $Y.w_0.max$, $b$ can have transitions after time $Y.w_0.max$ and yet $a$, $b$, and $y$ are compatible ($y = AND(a, b)$). Hence,

$$A'.w_0 \subseteq 0|_{Y.w_0.lmin}^{+\infty} \quad \text{and}$$
$$B'.w_0 \subseteq 0|_{Y.w_0.lmin}^{+\infty}.$$

Therefore,

```
$C_{AND}(A.w_0, B.w_0, Y.w_0) = (A'.w_0, B'.w_0, Y'.w_0)$ such that,
if( $A.w_0 = B.w_0 = 0|_T^T$ ) {
    $Y'.w_0 = Y.w_0 \cap 0|_T^T$
} else {
    $Y'.w_0 = Y.w_0 \cap 0|_{-\infty}^{min(A.w_0.max,\, B.w_0.max)}$
}
$A'.w_0 = A.w_0 \cap 0|_{Y.w_0.lmin}^{+\infty}$
$B'.w_0 = B.w_0 \cap 0|_{Y.w_0.lmin}^{+\infty}$
// if any becomes empty, the others follow
if( $A'.w_0 = \phi$ ) { $B'.w_0 = \phi$; $Y'.w_0 = \phi$;}
if( $B'.w_0 = \phi$ ) { $A'.w_0 = \phi$; $Y'.w_0 = \phi$;}
if( $Y'.w_0 = \phi$ ) { $B'.w_0 = \phi$; $A'.w_0 = \phi$;}          (A1)
```

**Note:** $C_{AND}(A.w_0, B.w_0, Y.w_0)$ as defined in (A1) is not equivalent to the complete relational constraint operator based on *AND*. For example, $C_{AND}(A, B, Y) = (A, B, Y)$ for $A = 0|_{10}^{20}$, $B = 0|_{10}^{30}$, and $Y = 0|_{10}^{20}$. Let $x$ be the binary waveform defined as: $x(25) = 1$ and $x(t) = 0$ for $t \neq 25$. We have $x \in B$ but it is incompatible with *AND* in the context of $A$ and $Y$. The AND constraint operator could not narrow $B$ from $0|_{10}^{30}$ to $0|_{10}^{20}$ because the binary waveform $b$ defined as $b(10) = 1$, $b(30) = 1$, $b(t) = 0$ for $t \notin \{10, 30\}$ is compatible with AND in the context of $A$ and $Y$.

- $C_{AND}(A.w_0, B.w_1, Y.w_0) = (A'.w_0, B'.w_1, Y'.w_0)$

**Effect on $Y.w_0$:**

Fig. 71 enumerates the six cases of overlap of the *last transition intervals* of $A.w_0$ and $B.w_1$. In all cases, $Y.w_0.max$ is bounded by $A.w_0.max$. Since the waveforms in $A.w_0$ are stable at 0 after time $A.w_0.max$, all compatible waveforms in $Y$ are stable at 0 after $A.w_0.max$. On the other hand, for Cases 2-6, the binary waveform stable at 0 in $Y.w_0$ is compatible: Let $a \in A.w_0$ such that,

$$a(t) = 1 \text{ for } t = A.w_0.lmin$$
$$a(t) = 0 \text{ otherwise,}$$

and let $b \in B.w_1$ such that,

$$b(t) = 0 \text{ for } t \le B.w_1.max$$
$$b(t) = 1 \text{ otherwise.}$$

For Cases 2-6 $A.w_0.lmin \le B.w_1.max$; therefore, $a$ and $b$ are never 1 at the same time, and ANDing them results in the stable 0 in $Y.w_0$. In Case 1, however, the waveforms in $A.w_0$ have transitions after the waveforms in $B.w_1$ stabilized at 1. Hence, waveforms in $Y.w_0$ stabilizing at 0 before $A.w_0.lmin$ are incompatible. Therefore,

if ( $A.w_0.lmin > B.w_1.max$ ) then

$$Y'.w_0 \subseteq A.w_0$$

else

$$Y'.w_0 \subseteq 0|_{-\infty}^{A.w_0.max}$$

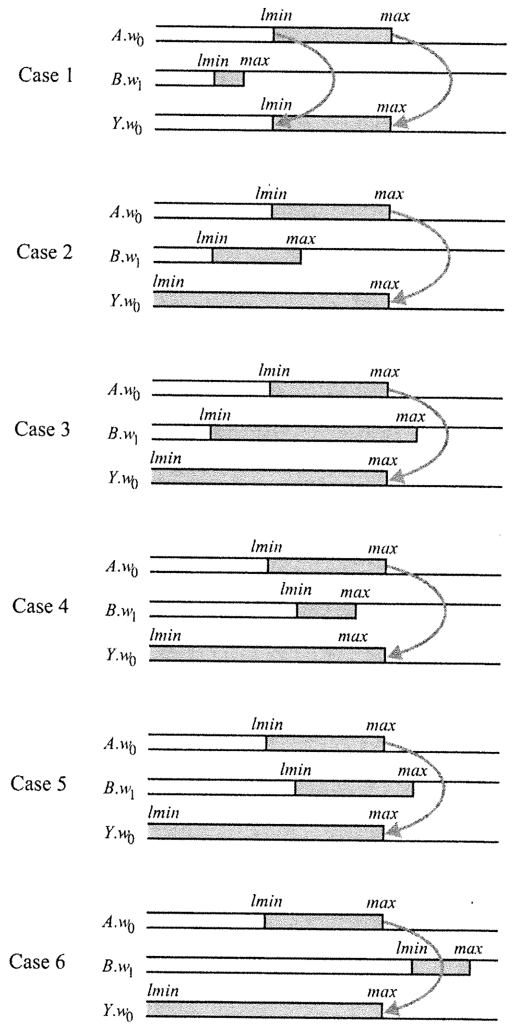


***Figure 71***
*AND Constraint ( $A.w_0$, $B.w_1$, $Y.w_0$ ) - effect on $Y.w_0$.*

**Effect on $A.w_0$:**

Fig. 72 enumerates the six cases of overlap of the *last transition intervals* of $B.w_1$ and $Y.w_0$. Waveforms in $A.w_0$ stabilizing before $Y.w_0.lmin$ are incompatible with the constraint because they result in waveforms in $Y$ having no transitions at or after $Y.w_0.lmin$. In fact, let $y \in Y.w_0$, $\exists\, t \in Y.w_0$ such that $y(t) = 1$. Therefore, waveforms $a$ in $A$, and $b$ in $B$ compatible with $y$ are such that $a(t) = 1$ and $b(t) = 1$. On the other hand, waveforms in $A.w_0$ having transition after $max(Y.w_0.max, B.w_1.max)$ are incompatible with the constraint as they result in waveforms in $Y.w_0$ unstable after time $Y.w_0.max$. Hence,

$$A'.w_0 \subseteq 0|_{Y.w_0.lmin}^{max(Y.w_0.max,\, B.w_1.max)}$$

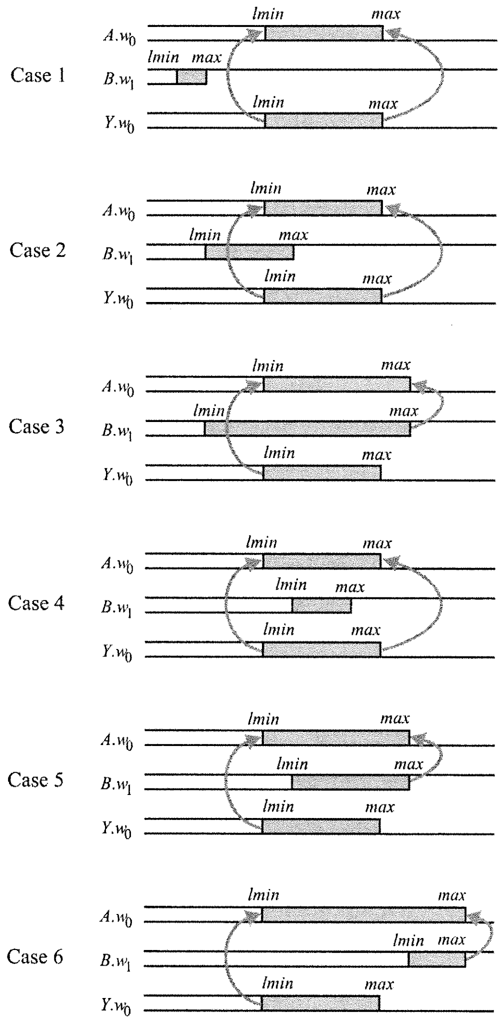


***Figure 72***
*AND Constraint ( A.w0, B.w1, Y.w0 ) - effect on A.w0.*

**Effect on $B.w_1$:**

Fig. 73 enumerates the six cases of overlap of the *last transition intervals* of $A.w_0$ and $Y.w_0$. Case 1, where $A.w_0.max < Y.w_0.lmin$, is inconsistent because no waveform in $Y.w_0$ is stable at 0 after time $A.w_0.max$. For Cases 2-5, no assumption can be made on how waveforms in $B.w_1$ should behave. In fact, let $a \in A.w_0$ and $y \in Y.w_0$ such that,

$$a(t) = 1 \text{ when } t = max(A.w_0.lmin, Y.w_0.lmin),$$
$$a(t) = 0 \text{ otherwise,}$$

and $\quad y = a$.

The binary waveform in $B.w_1$ stable at 1 is compatible with $a$ and $y$. Also, the binary waveform in $B.w_1$ stable at 1 at and before time $max(A.w_0.max, Y.w_0.max)$, and having transitions elsewhere is also compatible with $a$ and $y$. Therefore, for Cases 2-5, $B'.w_1 \subseteq 1|_{-\infty}^{+\infty}$. Case 6 is different, let $a \in A.w_0$, $\exists\, t \in A.w_0$, $a(t) = 1$. Since waveforms in $Y.w_0$ are stable at 0 after time $Y.w_0.max$, any waveform $b$ in $B.w_1$ compatible with $a$ is such that $b(t) = 0$. Therefore, $B'.w_1 \subseteq 1|_{A.w_0.lmin}^{+\infty}$.

The effect on $B.w_1$ is as follows:

```
if ( $A.w_0.max < Y.w_0.lmin$ ) then
    $B'.w_1 \subseteq \phi$
else
    if ( $A.w_0.lmin > Y.w_0.max$ ) then
        $B'.w_1 \subseteq 1|_{A.w_0.lmin}^{+\infty}$
    else
        $B'.w_1 \subseteq 1|_{-\infty}^{+\infty}$
```

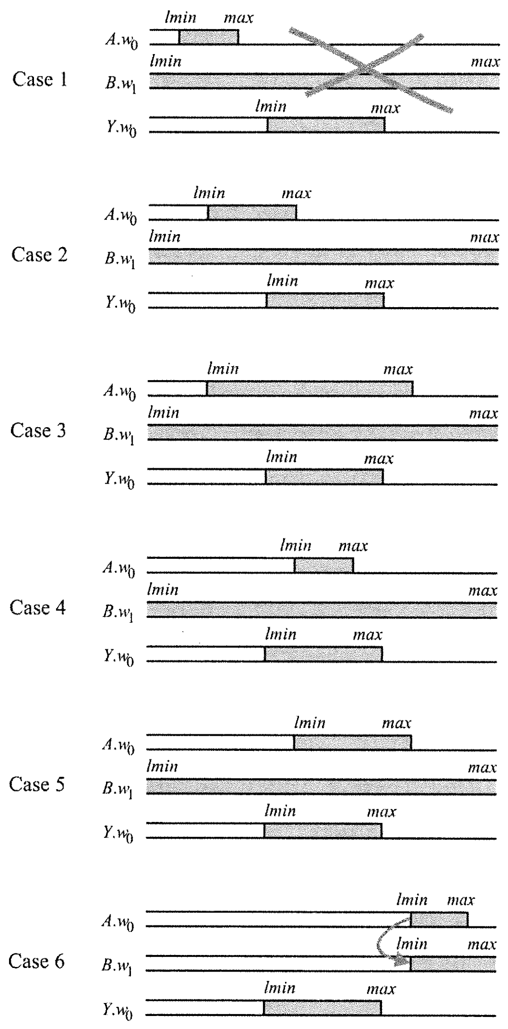


***Figure 73***
*AND Constraint ( A.w0, B.w1, Y.w0 ) - effect on B.w1.*

Finally,

```
$C_{AND}(A.w_0, B.w_1, Y.w_0) = (A'.w_0, B'.w_1, Y'.w_0)$ such that,

if( $A.w_0.lmin > B.w_1.max$ ) {
    $Y'.w_0 = Y.w_0 \cap A.w_0$
} else {
    $Y'.w_0 = Y.w_0 \cap 0|_{-\infty}^{A.w_0.max}$
}
$A'.w_0 = A.w_0 \cap 0|_{Y.w_0.lmin}^{max(Y.w_0.max, B.w_1.max)}$
if( $A.w_0.max < Y.w_0.lmin$ ) {
    $B'.w_1 = \phi$
} else {
    if( $A.w_0.lmin > Y.w_0.max$ ) {
        $B'.w_1 = B.w_1 \cap 1|_{A.w_0.lmin}^{+\infty}$
    } else {
        $B'.w_1 = B.w_1$
    }
}
// if any becomes empty, the others follow
if( $A'.w_0 = \phi$ ) {$B'.w_1 = \phi$; $Y'.w_0 = \phi$;}
if( $B'.w_1 = \phi$ ) {$A'.w_0 = \phi$; $Y'.w_0 = \phi$;}
if( $Y'.w_0 = \phi$ ) {$B'.w_1 = \phi$; $A'.w_0 = \phi$;}          (A2)
```

- $C_{AND}(A.w_1, B.w_0, Y.w_0) = (A'.w_1, B'.w_0, Y'.w_0)$

  This case is symmetrical to $C_{AND}(A.w_0, B.w_1, Y.w_0)$.

- $C_{AND}(A.w_1, B.w_1, Y.w_1) = (A'.w_1, B'.w_1, Y'.w_1)$

**Effect on $Y.w_1$:**

Fig. 74 enumerates the six cases of overlap of the *last transition intervals* of $A.w_1$ and $B.w_1$. In all cases, waveforms in $Y.w_1$ having transitions after $max(A.w_1.max, B.w_1.max)$ are incompatible. Also, waveforms in $Y.w_1$ have transitions at or after $max(A.w_1.lmin, B.w_1.lmin)$. This is caused by the fact that waveforms in $A.w_1$ and $B.w_1$ are at a controlling value (0) some where in their respective *last transition intervals*. Therefore,

$$Y'.w_1 \subseteq 1|_{max(A.w_1.lmin, B.w_1.lmin)}^{max(A.w_1.max, B.w_1.max)}$$

**Effect on $A.w_1$:**

Fig. 75 enumerates the six cases of overlap of the *last transition intervals* of $B.w_1$ and $Y.w_1$. In Case 1, waveforms in $Y.w_1$ have transitions after the waveforms in $B.w_1$ stabilized at 1. Therefore, waveforms in $A.w_1$ stabilizing at 1 before $Y.w_1.lmin$ are incompatible. For all cases, waveforms in $A.w_1$ having transition after $Y.w_1.max$ are incompatible as they result in no waveform in $Y.w_1$. Case 6 is inconsistent as all waveforms in $B.w_1$ are at a controlling value (0) at certain time in $B.w_1$, while the waveforms in $Y.w_1$ are stable at 1 at the same time. $A.w_1$ is bounded as follows:

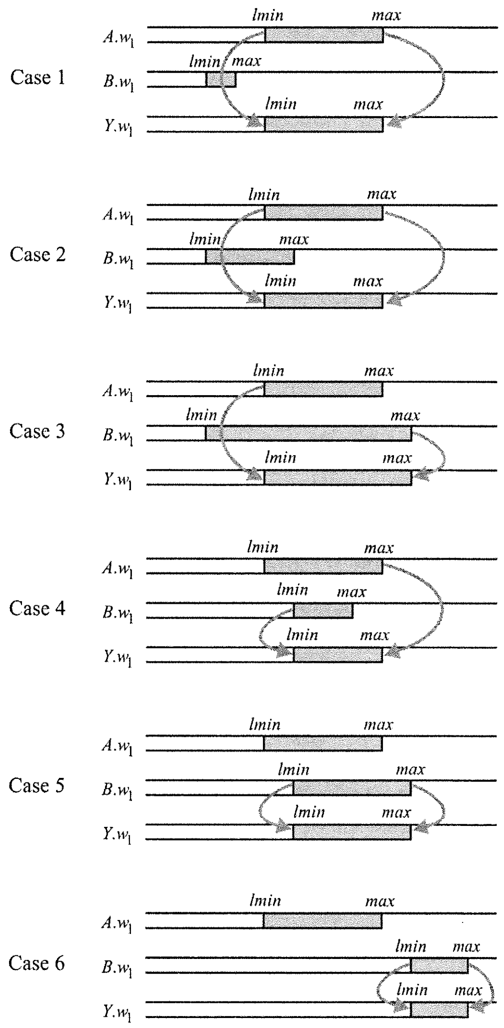


*Figure 74*
*AND Constraint ( A.w1, B.w1, Y.w1 ) - effect on Y.w1.*

```
if ( $B.w_1.lmin > Y.w_1.max$ ) then
    $A'.w_1 = \phi$
else
    if ( $Y.w_1.lmin > B.w_1.max$ ) then
        $A'.w_1 \subseteq 1|_{Y.w_1.lmin}^{Y.w_1.max}$
    else
        $A'.w_1 \subseteq 1|_{-\infty}^{Y.w_1.max}$
```

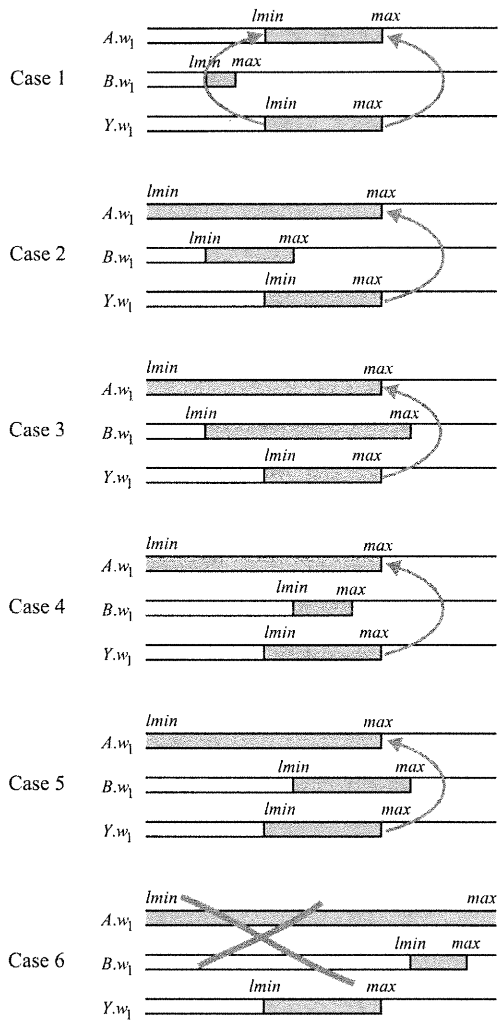


***Figure 75***
*AND Constraint ( $A.w_1$, $B.w_1$, $Y.w_1$ ) - effect on $A.w_1$.*

**Effect on $B.w_1$ :**

This case is symmetrical to the case of $A.w_1$.

Finally, combining the bounds on $A.w_1$, $B.w_1$, and $Y.w_1$ gives:

```
$C_{AND}(A.w_1, B.w_1, Y.w_1) = (A'.w_1, B'.w_1, Y'.w_1)$ such that,

$Y'.w_1 = Y.w_1 \cap 1|_{max(A.w_1.lmin,\, B.w_1.lmin)}^{max(A.w_1.max,\, B.w_1.max)}$
if( $B.w_1.lmin > Y.w_1.max$ ) {
    $B'.w_1 = \phi$; // the others become empty later
} else {
    if( $Y.w_1.lmin > B.w_1.max$ ) {
        $A'.w_1 = A.w_1 \cap 1|_{Y.w_1.lmin}^{Y.w_1.max}$
    } else {
        $A'.w_1 = A.w_1 \cap 1|_{-\infty}^{Y.w_1.max}$
    }
    if( $A.w_1.lmin > Y.w_1.max$ ) {
        $A'.w_1 = \phi$; // the others become empty later
    } else {
        if( $Y.w_1.lmin > A.w_1.max$ ) {
            $B'.w_1 = B.w_1 \cap 1|_{Y.w_1.lmin}^{Y.w_1.max}$
        } else {
            $B'.w_1 = B.w_1 \cap 1|_{-\infty}^{Y.w_1.max}$
        }
    }
}
// if any becomes empty, the others follow
if( $A'.w_1 = \phi$ ) {$B'.w_1 = \phi$; $Y'.w_1 = \phi$;}
if( $B'.w_1 = \phi$ ) {$A'.w_1 = \phi$; $Y'.w_1 = \phi$;}
if( $Y'.w_1 = \phi$ ) {$B'.w_1 = \phi$; $A'.w_1 = \phi$;}        (A3)
```

(A0), (A1), (A2), and (A3) constitute the algorithm for the 2-input AND gate constraint operator.

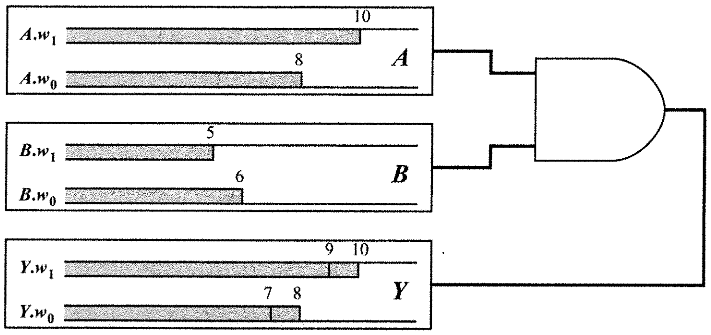


*Figure 76*
*AND Constraint Example.*

Example: Fig. 76 depicts a situation of three domains operated on by the AND constraint:

$$A = (0|_{-\infty}^{8}, 1|_{-\infty}^{10})$$
$$B = (0|_{-\infty}^{6}, 1|_{-\infty}^{5})$$
$$Y = (0|_{7}^{8}, 1|_{9}^{10})$$

(A0) implies:

$$C_{AND}(A, B, Y) = (A', B', Y') =$$
$$C_{AND}(0|_{-\infty}^{8}, 0|_{-\infty}^{6}, 0|_{7}^{8}) \cup$$
$$C_{AND}(0|_{-\infty}^{8}, 1|_{-\infty}^{5}, 0|_{7}^{8}) \cup$$
$$C_{AND}(1|_{-\infty}^{10}, 0|_{-\infty}^{6}, 0|_{7}^{8}) \cup$$
$$C_{AND}(1|_{-\infty}^{10}, 1|_{-\infty}^{5}, 1|_{9}^{10})$$

Fig. 77 illustrates the case for $C_{AND}(0|_{-\infty}^{8}, 0|_{-\infty}^{6}, 0|_{7}^{8})$, it is impossible for waveforms in $Y$ to have transitions after time 6 because the waveforms in $B$ are stable at 0 (controlling) after time 6. Therefore,
$C_{AND}(0|_{-\infty}^{8}, 0|_{-\infty}^{6}, 0|_{7}^{8}) = (\phi, \phi, \phi)$.

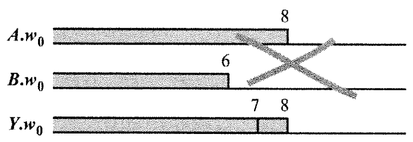


*Figure 77*
*AND Constraint Example:* $C_{AND}(0|_{-\infty}^{8}, 0|_{-\infty}^{6}, 0|_{7}^{8})$

Fig. 78 illustrates the case for $C_{AND}(0|_{-\infty}^{8}, 1|_{-\infty}^{5}, 0|_{7}^{8})$, since waveforms in A stabilize at a controlling value (0). Waveforms that stabilize

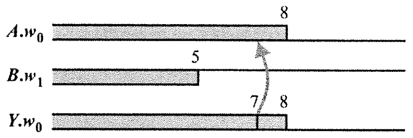


*Figure 78*
*AND Constraint Example:* $C_{AND}(0|_{-\infty}^{8}, 1|_{-\infty}^{5}, 0|_{7}^{8})$

before 7 are incompatible as they result in waveforms in Y having no transitions at or after 7. Therefore, $C_{AND}(0|_{-\infty}^{8}, 1|_{-\infty}^{5}, 0|_{7}^{8}) = (0|_{7}^{8}, 1|_{-\infty}^{5}, 0|_{7}^{8})$.

Fig. 79 illustrates the case for $C_{AND}(1|_{-\infty}^{10}, 0|_{-\infty}^{6}, 0|_{7}^{8})$. This is an inconsistent situation as it is impossible for waveforms in Y to have transitions at or after 7, while waveforms in B stabilize at 0 after 6. Therefore, $C_{AND}(1|_{-\infty}^{10}, 0|_{-\infty}^{6}, 0|_{7}^{8}) = (\phi, \phi, \phi)$.

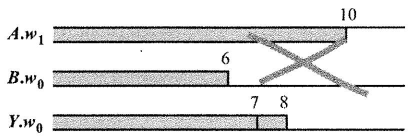


***Figure 79***
*AND Constraint Example:* $C_{AND}(1|_{-\infty}^{10}, 0|_{-\infty}^{6}, 0|_{7}^{8})$

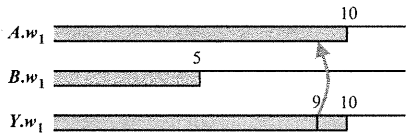


***Figure 80***
*AND Constraint Example:* $C_{AND}(1|_{-\infty}^{10}, 1|_{-\infty}^{5}, 1|_{9}^{10})$

Fig. 80 illustrates the case for $C_{AND}(1|_{-\infty}^{10}, 1|_{-\infty}^{5}, 1|_{9}^{10})$. Since waveforms in B are stable at 1 when the waveforms in Y have transitions at or after 9, waveforms in A stabilizing before 9 are incompatible. Therefore, $C_{AND}(1|_{-\infty}^{10}, 1|_{-\infty}^{5}, 1|_{9}^{10}) = (1|_{9}^{10}, 1|_{-\infty}^{5}, 1|_{9}^{10})$.

Hence, $C_{AND}(A, B, Y) = (A', B', Y') =$

$$\begin{aligned}
&(\phi, \phi, \phi) \cup \\
&(0|_{7}^{8}, 1|_{-\infty}^{5}, 0|_{7}^{8}) \cup \\
&(\phi, \phi, \phi) \cup \\
&(1|_{9}^{10}, 1|_{-\infty}^{5}, 1|_{9}^{10}) = ((0|_{7}^{8}, 1|_{9}^{10}), (\phi, 1|_{-\infty}^{5}), (0|_{7}^{8}, 1|_{9}^{10}))
\end{aligned}$$

as shown in Fig. 81.

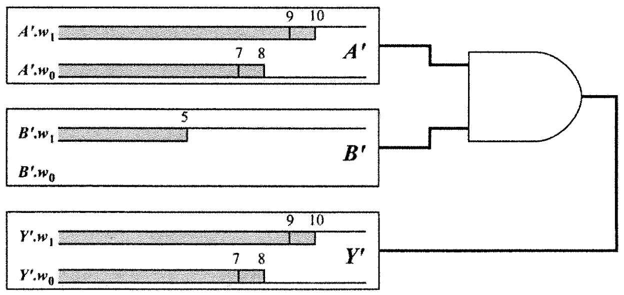


***Figure 81***
*Domains contents after the AND constraint is applied.*

### 3.5.2.4 OR - NAND - NOR Gates Constraints

The members of the family of AND, NAND, OR, and NOR gates have similar behavior. They all have:

- **controlling values**, 0 for AND / NAND, 1 for OR / NOR;
- **non-controlling values**, 1 for AND / NAND, 0 for OR / NOR;
- **controlled values**, i.e., the value of the gate output when one of the inputs is at a controlling value: 0 for AND / NOR, 1 for NAND / OR;
- **non-controlled values**, i.e., the value of the gate output when all inputs are at non-controlling values: 1 for AND / NOR, 0 for AND / OR.

Although it is possible to model these gates using AND and NOT constraints, it is more efficient to have a hard model for each. This is accomplished by parameterizing the AND constraint algorithm using the concept of Controlling / Non-Controlling / Controlled / Non-Controlled values. Table III shows the *abstract waveform* substitutions to get the constraints for NAND / OR / NOR from the constraint of the AND gate.

| AND | NAND | OR | NOR |
|---|---|---|---|
| $A.w_0$ | $A.w_0$ | $A.w_1$ | $A.w_1$ |
| $A.w_1$ | $A.w_1$ | $A.w_0$ | $A.w_0$ |
| $B.w_0$ | $B.w_0$ | $B.w_1$ | $B.w_1$ |
| $B.w_1$ | $B.w_1$ | $B.w_0$ | $B.w_0$ |
| $Y.w_0$ | $Y.w_1$ | $Y.w_1$ | $Y.w_0$ |
| $Y.w_1$ | $Y.w_0$ | $Y.w_0$ | $Y.w_1$ |

***Table III***
*Substitution for Controlling / Non-Controlling / Controlled / Non-Controlled values.*

For example, to get the OR gate constraint, replace the *abstract waveforms* in (A0), (A1), (A2), and (A3) as follows:

| | | |
|---|---|---|
| $A.w_0$ becomes $A.w_1$ | $A'.w_0$ becomes $A'.w_1$ | $A.w_0 \cap 0\|_x^y$ becomes $A.w_1 \cap 1\|_x^y$ |
| $A.w_1$ becomes $A.w_0$ | $A'.w_1$ becomes $A'.w_0$ | $A.w_1 \cap 1\|_x^y$ becomes $A.w_0 \cap 0\|_x^y$ |
| $B.w_0$ becomes $B.w_1$ | $B'.w_0$ becomes $B'.w_1$ | $B.w_0 \cap 0\|_x^y$ becomes $B.w_1 \cap 1\|_x^y$ |
| $B.w_1$ becomes $B.w_0$ | $B'.w_1$ becomes $B'.w_0$ | $B.w_1 \cap 1\|_x^y$ becomes $B.w_0 \cap 0\|_x^y$ |
| $Y.w_0$ becomes $Y.w_1$ | $Y'.w_0$ becomes $Y'.w_1$ | $Y.w_0 \cap 0\|_x^y$ becomes $Y.w_1 \cap 1\|_x^y$ |
| $Y.w_1$ becomes $Y.w_0$ | $Y'.w_1$ becomes $Y'.w_0$ | $Y.w_1 \cap 1\|_x^y$ becomes $Y.w_0 \cap 0\|_x^y$ |

### 3.5.2.5 Exclusive OR Gate (XOR)

**XOR Function:** the XOR function is a mapping $xor : BW \times BW \rightarrow BW$, for $a, b \in BW$ $xor(a, b) = y$ such that, $\forall\ t \in R$,

$y(t) = 0$ if $a(t) = b(t)$,

$y(t) = 1$ otherwise.

Fig. 82 shows an example of XORing two *binary waveforms*. At time 2, *a* and *b* changed values simultaneously, resulting in no transition for XOR(*a*,*b*). In fact, a transition at *a* (*b*) is reflected at *y* only if *b* (*a*) is stable during the transition.

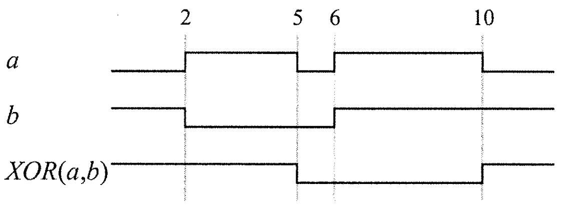


***Figure 82***
*XORing two binary waveforms.*

**XOR Constraint Operator:** the XOR constraint operator operates on three *abstract signal* domains (see Fig. 83):

***Figure 83***
*XOR Constraint.*

1) A, B are *abstract signal* domains for the inputs of the XOR gate.

2) Y is an *abstract signal* domain of the output of the XOR gate.

The XOR constraint is simpler than the AND counterpart because it does not have controlling values. In fact, 0 and 1 are both non-controlling. The presentation of the XOR constraint follows the same exhaustive case analysis of the *abstract waveforms* and the overlap of their *last transition intervals.*

$$C_{XOR}(A, B, Y) = C_{XOR}(A.w_0, B, Y) \cup C_{XOR}(A.w_1, B, Y)$$
$$C_{XOR}(A.w_0, B, Y) = C_{XOR}(A.w_0, B.w_0, Y) \cup C_{XOR}(A.w_0, B.w_1, Y)$$
$$C_{XOR}(A.w_1, B, Y) = C_{XOR}(A.w_1, B.w_0, Y) \cup C_{XOR}(A.w_1, B.w_1, Y)$$
$$C_{XOR}(A.w_0, B.w_0, Y) = C_{XOR}(A.w_0, B.w_0, Y.w_0) \cup C_{XOR}(A.w_0, B.w_0, Y.w_1)$$
$$C_{XOR}(A.w_0, B.w_1, Y) = C_{XOR}(A.w_0, B.w_1, Y.w_0) \cup C_{XOR}(A.w_0, B.w_1, Y.w_1)$$
$$C_{XOR}(A.w_1, B.w_0, Y) = C_{XOR}(A.w_1, B.w_0, Y.w_0) \cup C_{XOR}(A.w_1, B.w_0, Y.w_1)$$
$$C_{XOR}(A.w_1, B.w_1, Y) = C_{XOR}(A.w_1, B.w_1, Y.w_0) \cup C_{XOR}(A.w_1, B.w_1, Y.w_1)$$

Therefore,

$$\begin{aligned} C_{XOR}(A, B, Y) = \; & C_{XOR}(A.w_0, B.w_0, Y.w_0) \cup \\ & C_{XOR}(A.w_0, B.w_0, Y.w_1) \cup \\ & C_{XOR}(A.w_0, B.w_1, Y.w_0) \cup \\ & C_{XOR}(A.w_0, B.w_1, Y.w_1) \cup \\ & C_{XOR}(A.w_1, B.w_0, Y.w_0) \cup \\ & C_{XOR}(A.w_1, B.w_0, Y.w_1) \cup \\ & C_{XOR}(A.w_1, B.w_1, Y.w_0) \cup \\ & C_{XOR}(A.w_1, B.w_1, Y.w_1) \end{aligned}$$

$C_{XOR}(A.w_0, B.w_0, Y.w_1) = (\phi, \phi, \phi)$ because waveforms at both XOR input stabilizing at 0 result in waveforms stabilizing at 0 at the output $Y$.

For similar reasons, we have

$$C_{AND}(A.w_1, B.w_1, Y.w_1) = (\phi, \phi, \phi),$$
$$C_{AND}(A.w_1, B.w_0, Y.w_0) = (\phi, \phi, \phi), \text{ and}$$
$$C_{AND}(A.w_0, B.w_1, Y.w_0) = (\phi, \phi, \phi).$$

Therefore,

$$\boxed{\begin{aligned} & C_{XOR}(A, B, Y) = \\ & \qquad C_{XOR}(A.w_0, B.w_0, Y.w_0) \cup \\ & \qquad C_{XOR}(A.w_0, B.w_1, Y.w_1) \cup \qquad\qquad \textbf{(X0)} \\ & \qquad C_{XOR}(A.w_1, B.w_0, Y.w_1) \cup \\ & \qquad C_{XOR}(A.w_1, B.w_1, Y.w_0) \end{aligned}}$$

Each subset is considered separately, and all domains are considered non-empty, as the case of empty domains is trivial.

- $C_{XOR}(A.w_0, B.w_0, Y.w_0) = (A'.w_0, B'.w_0, Y'.w_0)$

**Effect on** $Y.w_0$**:**

Fig. 84 enumerates the six cases of overlap of the *last transition intervals* of $A.w_0$ and $B.w_0$. In all cases, waveforms in $Y.w_0$ having transitions after $max(A.w_0.max, B.w_0.max)$ are incompatible. Also, for case 1 (6), waveforms in $A.w_0$ ($B.w_0$) happened to have transitions when the waveforms in $B.w_0$ ($A.w_0$) are already stable at 0. Therefore, waveforms in $Y.w_0$ stabilizing before $A.w_0.lmin$ ($A.w_0.lmin$) are incompatible. $Y.w_0$ is bounded as follows:

```
if( $A.w_0.max < B.w_0.lmin$ ) then
    $Y'.w_0 \subseteq 0|_{B.w_0.lmin}^{B.w_0.max}$
else
    if( $B.w_0.max < A.w_0.lmin$ ) then
        $Y'.w_0 \subseteq 0|_{A.w_0.lmin}^{A.w_0.max}$
    else
        $Y'.w_0 \subseteq 0|_{-\infty}^{max(A.w_0.max,\ B.w_0.max)}$
```

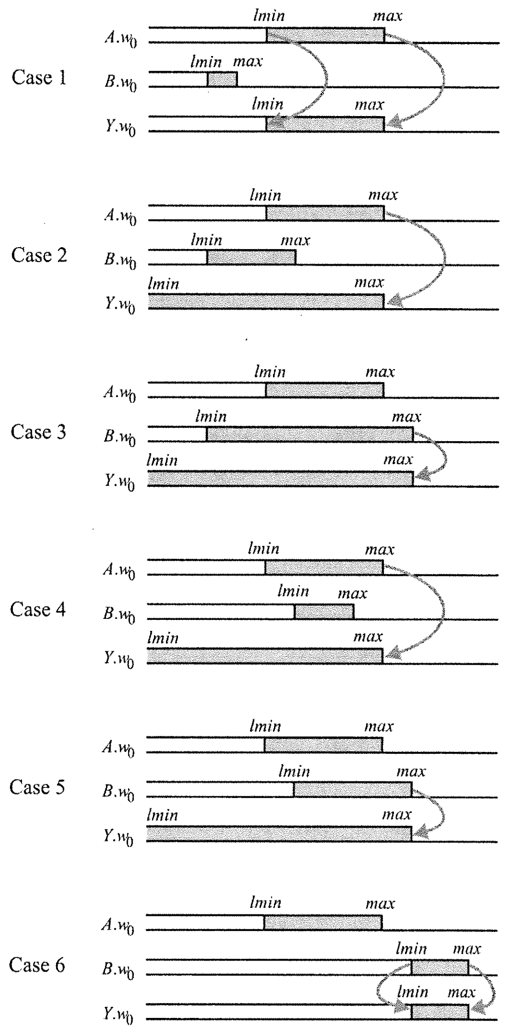


Figure 84
XOR Constraint ( $A.w_0$, $B.w_0$, $Y.w_0$ ) - effect on $Y.w_0$.

**Effect on** $A.w_0$**:**

Fig. 85 enumerates the six cases of overlap of the *last transition intervals* of $B.w_0$ and $Y.w_0$. In Case 1, the waveforms in $Y.w_0$ happen to have transitions after the waveforms in $B.w_0$ stabilized. Therefore, the waveforms in $A.w_0$ stabilizing before

$Y.w_0.lmin$ are incompatible. Case 6 is symmetrical to Case 1. On the other hand, the waveforms in $A.w_0$ cannot have transitions after the waveforms in both $B.w_0$ and $Y.w_0$ have stabilized. $A.w_0$ is bounded as follows:

```
if ( $B.w_0.max < Y.w_0.lmin$ ) then
    $A'.w_0 \subseteq 0|_{Y.w_0.lmin}^{Y.w_0.max}$
else
    if ( $Y.w_0.max < B.w_0.lmin$ ) then
        $A'.w_0 \subseteq 0|_{B.w_0.lmin}^{B.w_0.max}$
    else
        $A'.w_0 \subseteq 0|_{-\infty}^{max(B.w_0.max,\ Y.w_0.max)}$
```

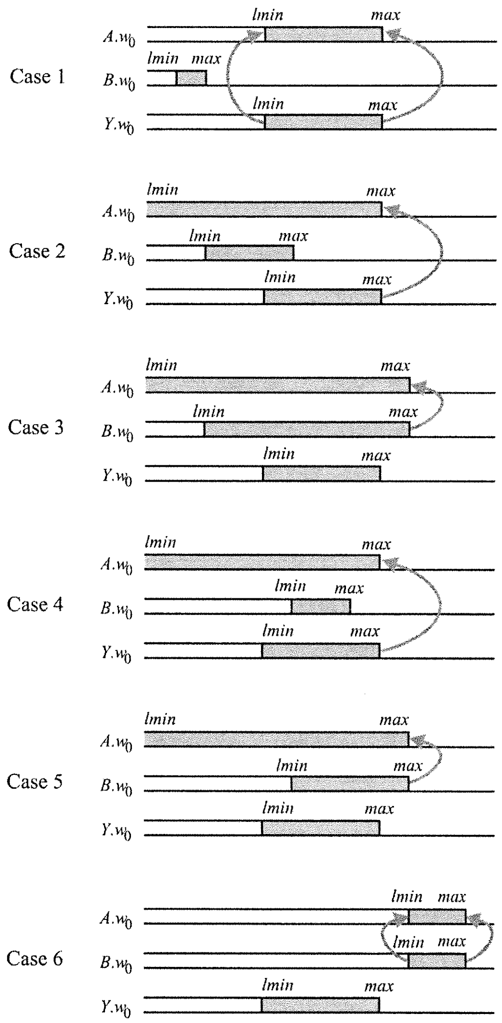


***Figure 85***
*XOR Constraint ( A.w0, B.w0, Y.w0 ) - effect on A.w0.*

**Effect on $B.w_0$:**

The effect on $B.w_0$ is symmetrical to the case for $A.w_0$.

In fact, all the cases of the XOR gate are symmetrical. A transition present at an input results in a transition at the output provided the other input does not change at the same time.

Therefore,

```
$C_{XOR}(A.w_0, B.w_0, Y.w_0) = (A'.w_0, B'.w_0, Y'.w_0)$ such that:

if( $A.w_0.max < B.w_0.lmin$ ) {
    $Y'.w_0 = Y.w_0 \cap 0|_{B.w_0.lmin}^{B.w_0.max}$
} else {
    if( $B.w_0.max < A.w_0.lmin$ ) {
        $Y'.w_0 = Y.w_0 \cap 0|_{A.w_0.lmin}^{A.w_0.max}$
    } else {
        $Y'.w_0 = Y.w_0 \cap 0|_{-\infty}^{max(A.w_0.max,\, B.w_0.max)}$ } }
if( $B.w_0.max < Y.w_0.lmin$ ) {
    $A'.w_0 = A.w_0 \cap 0|_{Y.w_0.lmin}^{Y.w_0.max}$
} else {
    if( $Y.w_0.max < B.w_0.lmin$ ) {
        $A'.w_0 = A.w_0 \cap 0|_{B.w_0.lmin}^{B.w_0.max}$
    } else {
        $A'.w_0 = A.w_0 \cap 0|_{-\infty}^{max(B.w_0.max,\, Y.w_0.max)}$
    } }
if( $A.w_0.max < Y.w_0.lmin$ ) {
    $B'.w_0 = B.w_0 \cap 0|_{Y.w_0.lmin}^{Y.w_0.max}$
} else {
    if( $Y.w_0.max < A.w_0.lmin$ ) {
        $B'.w_0 = B.w_0 \cap 0|_{A.w_0.lmin}^{A.w_0.max}$
    } else {
        $B'.w_0 = B.w_0 \cap 0|_{-\infty}^{max(A.w_0.max,\, Y.w_0.max)}$ } }

if( $A'.w_0 = \phi$ ) {$B'.w_0 = \phi$; $Y'.w_0 = \phi$;}
if( $B'.w_0 = \phi$ ) {$A'.w_0 = \phi$; $Y'.w_0 = \phi$;}
if( $Y'.w_0 = \phi$ ) {$B'.w_0 = \phi$; $A'.w_0 = \phi$;}          (X1)
```

The algorithm for the constraint $C_{XOR}(A.w_0, B.w_1, Y.w_1) = (A'.w_0, B'.w_1, Y'.w_1)$ is deduced from **(X1)** by the following substitution:

| | | |
|---|---|---|
| $A.w_0$ becomes $A.w_0$ | $A'.w_0$ becomes $A'.w_0$ | $A.w_0 \cap 0|_x^y$ becomes $A.w_0 \cap 0|_x^y$ |
| $B.w_0$ becomes $B.w_1$ | $B'.w_0$ becomes $B'.w_1$ | $B.w_0 \cap 0|_x^y$ becomes $B.w_1 \cap 1|_x^y$ |
| $Y.w_0$ becomes $Y.w_1$ | $Y'.w_0$ becomes $Y'.w_1$ | $Y.w_0 \cap 0|_x^y$ becomes $Y.w_1 \cap 1|_x^y$ |

The algorithms for $C_{XOR}(A.w_1, B.w_0, Y.w_1) = (A'.w_1, B'.w_0, Y'.w_1)$ and $C_{XOR}(A.w_1, B.w_1, Y.w_0) = (A'.w_1, B'.w_1, Y'.w_0)$ are derived from **(X1)** in a similar way. Also the Inverted XOR gate (XNOR) is symmetrical to XOR, we simply exchange $Y.w_0$ and $Y.w_1$.

## 3.6 Combinational Circuit Delay Verification / Basic Modeling

**Timing Check:** a ***timing-check*** is a tuple $\sigma = (\xi, s, \delta)$ where $\xi$ is a combinational circuit, $s$ is a primary output of $\xi$, $\delta$ is a delay value, and the primary inputs of $\xi$ are considered stable after time 0. It represents the following timing verification decision problem:

***Does the output s of circuit $\xi$ have a delay greater than or equal to $\delta$?***

The *timing-check* $\sigma = (\xi, s, \delta)$ is transformed into a constraint system that is consistent iff the output $s$ has a delay greater than or equal to $\delta$. The constraint system is built around the sub-circuit graph corresponding to the nodes that have arcs leading to $s$ (the fan-in cone of $s$). The circuit is transformed into an equivalent form that contains only gates that have a hard constraint model (Delays, Buffers, Inverters, 2-input-AND, etc.). In fact, this is done in a bottom-up fashion: the logic gates of the technology cell library of the circuit is modeled first, then each cell instance is replaced by its model. Hence the transformation is straightforward and involves no complexity overhead, the cell library is modeled once and used for all circuits designed for the cell technology. Fig. 86 shows how a four-input-NAND gate is modeled by a constraint network, 2-input delayless AND and NAND constraints are cascaded, and the gate delay is modeled by a delay constraint placed at the gate output. The constraint model contains variables associated with the original gate terminals $A$, $B$, $C$, $D$, and $Y$, and variables for the internal nodes $I_1$, $I_2$, and $I_3$, and a variable for the delay value $DV$. In order to simplify the presentation of the remainder of this chapter, we consider that circuits are composed of one and

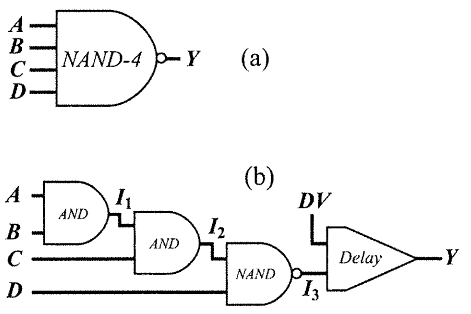


*Figure 86*
*NAND-4 constraint model.*

two-input symmetrical fixed delay gates. The delay constraint is embedded (hard coded) with the gate constraints making the constraint network isomorphic to the circuit graph. In fact, this is suitable for combinational circuit delay verification where no clock is used. Although the gate constraints have no notion of inputs and outputs (they are actually relations), the graph of constraints is directed, according to the combinational circuit graph.

Let $\xi(\{Gate_1, Gate_2, \ldots, Gate_m\}, \{Net_1, Net_2, \ldots, Net_n\})$ be the circuit of $m$ gates and $n$ nets where each gate is connected to a subset of the nets. We build a constraint system composed of $n$ variables $X_1, X_2, \ldots, X_n$ associated with the $n$ domains $D_1, D_2, \ldots, D_n$, respectively, and $m$ relational constraints $C_1, C_2, \ldots, C_m$, where $C_i$ operates on the domains corresponding to the variables of the nets connected to $Gate_i$. The initial values for all the domains of the constraint system of $\sigma$ are $(0|_{-\infty}^{+\infty}, 1|_{-\infty}^{+\infty})$ so as to contain any possible *BW.* For *floating-mode* delay calculation, we restrict the primary inputs to the set of waveforms that are stable after time 0: $(0|_{-\infty}^{0}, 1|_{-\infty}^{0})$. To verify whether the output *s* has a delay greater than or equal to $\delta$, we restrict the signal domain of *s* to the waveforms having transitions at or after time $\delta$, i.e., $D_s = (0|_{\delta}^{+\infty}, 1|_{\delta}^{+\infty})$.

The constraint system is tightened (solved) by repeatedly applying the gate constraint operators until no *narrowing* (change) of any domain is possible, i.e., the greatest fixpoint of the system of equations induced by the constraint system is reached. We implemented this iterative computation efficiently using an event-driven scheduler. The function **evaluateConstraintSystem()** in Appendix A (A.1) evaluates the constraint system.

**Timing-Check Compatible Waveforms:** Given a *timing-check* $\sigma = (\xi, s, \delta)$ and its corresponding constraint system composed of the variables $X_1, X_2, \ldots, X_n$, their respective domains $D_1, D_2, \ldots, D_n$, and the constraints $C_1, C_2, \ldots, C_m$, a *binary waveform* $w \in D_k$ is said to be ***σ-compatible*** iff it is part of a solution, i.e., iff there is a waveform in each $D_i$, $i \neq k$, such that with *w* from $D_k$, the constraint system is satisfied. *w* is said to be ***σ-incompatible*** if it is not *σ-compatible.*

For combinational circuits with multiple outputs, each output is considered with the sub-circuit corresponding to its fan-in cone included in its timing check.

**Theorem 1:** Given a *timing-check* $\sigma = (\xi, s, \delta)$, the fixpoint of the evaluation of its corresponding constraint system is reached in a finite number of steps.

***Proof:*** This is a consequence of the domains' discrete representation and the monotonicity of the constraint operators. Since each domain is represented using a finite number of 32 bit integers, the number of values a domain can take is finite. Moreover, if $D_i$ and $D_{i+1}$ are the values of a domain $D$ at iterations $i$ and $(i + 1)$, respectively, we have $D_{i+1} \subseteq D_i$ (according to the definition of the constraint operators). The time domain of interest for a given $\sigma = (\xi, s, \delta)$ is [0 , *top*], *top* is the topological delay of the circuit $\xi$. The possible values for the *lmin* and *max* parameters of $D.w_0$ and $D.w_1$ are in {-∞, 0, 1, ..., *top*, +∞}. In fact, $-\infty$ and $+\infty$ can be represented as -1 and (*top*+1), respectively. The number of ways a non-empty interval [*lmin*, *max*] can be narrowed is $((max - lmin + 1) \times (max - lmin + 2)/2) + 1$, a positive number that decreases when (*max* - *lmin*) decreases. For instance, the interval [2,3] can be narrowed to one of 4 possibilities: [2,3], [2,2], [3,3], and $\phi$. Let now $f_D(i)$ be the number of ways a domain $D$ can be narrowed at iteration $i$. $f_D{:}N \rightarrow N$ is a finite non-increasing discrete function. Therefore, $\exists\, n_0 \in N \mid \forall\, n > n_0, f_D(n) = f_D(n_0)$. The minimum number after which the functions $f_D$ of all domains converge is the number of steps required to reach the fixpoint, and this number is finite.

The time complexity of **evaluateConstraintSystem()** is very difficult to establish. In fact, it has a worst case of enumerating all integer values of a large interval. This worst case happens when the constraint system has circular implications such as the case of the sequential circuit example in section 3.4. Consider Step 5 of this example. The setup constraint narrowed the clock domain of the next cycle from [32, 42] to [32, 40]. In real cases, however, delays are scaled to fine resolutions. Therefore, if a domain is narrowed one time unit each iteration, e.g., from [6000000, 8000000] to [6000000, 7999999], it may need to enumerate the integer values of a very large interval before the fixpoint is reached. When such a slow convergence is encountered, the evaluation is stopped and the constraint system is considered consistent. This worst case behavior is not encountered in combinational circuits where clocks are not considered. In fact, if the gates constraints are evaluated once in topological order, followed by another evaluation in reverse topological order, the system become "very close" to its greatest fix point. For combinational circuits, the time complexity of **evaluateConstraintSystem()** is virtually linear with the circuit size. However, the algorithm is very pessimistic in the presence of reconvergent fan-outs.

**Theorem 2:** Let $\sigma = (\xi, s, \delta)$ be a *timing-check*, and $D_s$ the domain of $s$ after its constraint system has been evaluated. If $D_s = \{\phi, \phi\}$ then no transition is possible on output $s$ at or after time $\delta$.

This is a direct consequence of the way the gate constraints are defined. The constraints never eliminate *compatible binary waveforms* from any domain, although they may include some *incompatible* ones to allow representing the family of waveforms by an *abstract signal* value. Moreover, when a domain becomes empty, its value propagates to all other domains because no value is *compatible* with an empty set of waveforms.

When the evaluation reaches a fixpoint with non-empty domains, the domains may still contain only *incompatible binary waveforms* which in this case represents a false negative answer to the question: "Is the circuit output $s$ stable at and after time $\delta$?". Only inconsistency of the constraint system is the exact answer, implying that the output $s$ is stable at and after time $\delta$.

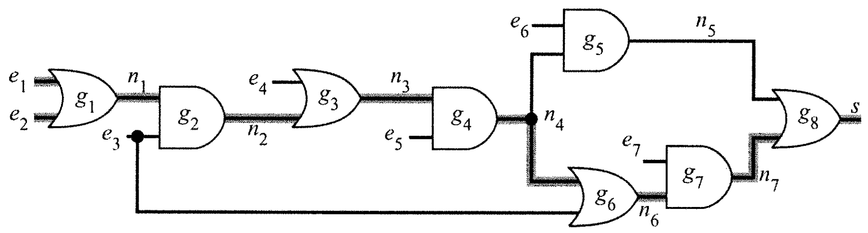


*Figure 87*
*False path circuit of Hrapcenko.*

**Hrapcenko false-path example:** Consider the *timing-check* $\sigma = (\xi, s, 61)$ where $\xi$ is the circuit of Fig. 87 [57]. Assuming a delay of 10 for each gate, the topological delay of $\xi$ is *top* $= 70$ and its *floating-mode* delay is known to be 60, because the path $\{n_1, g_2, n_2, g_3, n_3, g_4, n_4, g_6, n_6, g_7, n_7, g_8, s\}$ is false. This can be easily proven for this particular example: it is caused by $g_2$ and $g_6$ sensitization conflict, while $e_3$ is required to be 1 for $g_2$, it has to be 0 for $g_6$. Let's prove that the *floating* circuit delay is less than 61 by resolving the constraint system associated with the *timing check* $\sigma$. Let

$D_{e_1}, D_{e_2}, D_{e_3}, D_{e_4}, D_{e_5}, D_{e_6}, D_{e_7}, D_{n_1}, D_{n_2}, D_{n_3}, D_{n_4}, D_{n_5}, D_{n_6}, D_{n_7}, D_s$ be the domains associated with the variables of the corresponding nets. The initial domain values are:

- Inputs stable after time 0: $D_{e_i} = (0|_{-\infty}^{0}, 1|_{-\infty}^{0}), i \in \{1, 2, 3, 4, 5, 6, 7\}$
- All waveforms for internal nodes: $D_{n_i} = (0|_{-\infty}^{+\infty}, 1|_{-\infty}^{+\infty}), i \in \{1, 2, 3, 4, 5, 6, 7\}$
- Waveforms that violate the timing constraints at the output, transitions at or after time 61: $(0|_{61}^{+\infty}, 1|_{61}^{+\infty})$

Applying the constraint operators of the gates yields:

$g_1 \Rightarrow D_{n_1} = (0|_{-\infty}^{10}, 1|_{-\infty}^{10})$ the maximal delay of $g_1$ is 10; therefore, no transition is possible on $n_1$ after time 10.
$g_2 \Rightarrow D_{n_2} = (0|_{-\infty}^{20}, 1|_{-\infty}^{20})$
$g_3 \Rightarrow D_{n_3} = (0|_{-\infty}^{30}, 1|_{-\infty}^{30})$
$g_4 \Rightarrow D_{n_4} = (0|_{-\infty}^{40}, 1|_{-\infty}^{40})$
$g_5 \Rightarrow D_{n_5} = (0|_{-\infty}^{50}, 1|_{-\infty}^{50})$
$g_6 \Rightarrow D_{n_6} = (0|_{-\infty}^{50}, 1|_{-\infty}^{50})$
$g_7 \Rightarrow D_{n_7} = (0|_{-\infty}^{60}, 1|_{-\infty}^{60})$
$g_8 \Rightarrow D_s = (0|_{61}^{70}, 1|_{61}^{70}), D_{n_5} = (0|_{-\infty}^{50}, \phi), D_{n_7} = (0|_{51}^{60}, 1|_{51}^{60})$: the *last-transition-interval* on $s$ is propagated to $n_7$ and the controlling waveforms on $n_5$ are removed because they "block the way" on $n_7$;
$g_7 \Rightarrow D_{n_6} = (0|_{41}^{50}, 1|_{41}^{50}), D_{e_7} = (\phi, 1|_{-\infty}^{0})$
$g_6 \Rightarrow D_{n_4} = (0|_{31}^{40}, 1|_{31}^{40}), D_{e_3} = (0|_{-\infty}^{0}, \phi)$
$g_4 \Rightarrow D_{n_3} = (0|_{21}^{30}, 1|_{21}^{30}), D_{e_5} = (\phi, 1|_{-\infty}^{0})$
$g_3 \Rightarrow D_{n_2} = (0|_{11}^{20}, 1|_{11}^{20}), D_{e_4} = (0|_{-\infty}^{0}, \phi)$
$g_2 \Rightarrow D_{n_1} = (0|_{1}^{10}, 1|_{1}^{10}), D_{e_3} = (\phi, \phi)$. This proves that no transition is possible on $s$ at or after t = 61, therefore, the circuit *floating delay* is less than 61.

## 3.7 Reduction of Pessimism

The basic waveform narrowing method is based on local consistency techniques. Constraints consider each gate as isolated, ignoring the global circuit function. Therefore, the system evaluation may result in false negative answers when we end-up with non-empty domains and yet the constraint system has no solution. To reduce this pessimism we incorporated additional constraints based on necessary conditions determined by considering the global circuit function.

### 3.7.1 Static Learning

*Static learning* [35] is a technique widely used to speed up test pattern generation. It consists of determining a table of global logic implications using the negation of forward logic implications. For example, consider the circuit c17 (ISCAS'85 benchmark) of Fig. 88:

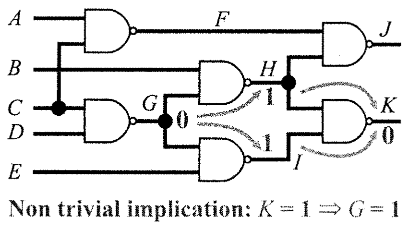


***Figure 88***
*Static Learning on the circuit c17.*

$$G = 0 \;\Rightarrow\; (H = 1) \wedge (I = 1) \;\Rightarrow\; K = 0$$
$$\text{Therefore, } \neg(K = 0) \;\Rightarrow\; \neg(G = 0)$$

$K = 1 \;\Rightarrow\; G = 1$ is a non-trivial global implication that is not deduced using local constraint operators. In fact, $K = 1$ can be caused by either $(H = 0) \wedge ((I = 0) \vee (I = 1))$ or $(I = 0) \wedge ((H = 0) \vee (H = 1))$. Therefore, no deduction can be made from a local stand point. Note that $H = 0 \;\Rightarrow\; G = 1$ is not considered a learned implication because it is trivial. In fact, it can be determined using the local constraint operator of NAND. A good heuristic to distinguish trivial and non-trivial implications is to keep only what was caused by non-controlling values. For instance, $\neg(K = 0) \;\Rightarrow\; \neg(G = 0)$ is considered non-trivial as $K = 0$ is caused by non-controlling values at the NAND inputs.

*Static learning* is implemented in a pre-processing stage that determines implications and adds new constraint operators to the constraint system. When a *class* ($w_0$ or $w_1$) becomes empty during constraint system resolution, *learning* tables are used to impose *class* restrictions on other domains wherever possible. For instance, the added learned constraint in Fig. 89 is defined as:

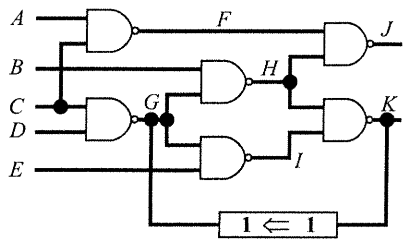


**Figure 89**
*Learned constraint.*

$$\text{if}\,(\,K.w_0 \;=\; \phi\,)\;\text{then}$$
$$G \;=\; G \cap (\phi,\, 1|_{-\infty}^{+\infty})$$

That is, if there is only the $w_1$ waveform on *K*, it must be that there is only the $w_1$ waveform on *G*.

## 3.7.2 Spatial Correlation

Another way to reduce pessimism is to enforce spatial correlation on reconvergent fan-outs. The spatial correlation procedure is as follows:

1) Select a reconvergent fan-out X;
2) Save the circuit domains as ALL;
3) Restrict the domain of X to waveforms stabilizing at 1, and evaluate the system;
4) Save the circuit domains as SC1 and restore ALL;
5) Restrict the domain of X to waveforms stabilizing at 0, and evaluate the system;
6) Merge the circuit domains with SC1;

Spatial correlation is performed on reconvergent fan-outs in topological order. Beside reducing the pessimism of the method, this procedure enables the calculation of a tight upper bound of the circuit delay instead of resolving a decision problem that consists of comparing the circuit delay to a certain value. For instance, let's apply the spatial correlation to $e_3$ of the false path example of Hrapcenko without imposing any constraint on the output *s* (see Fig. 90):

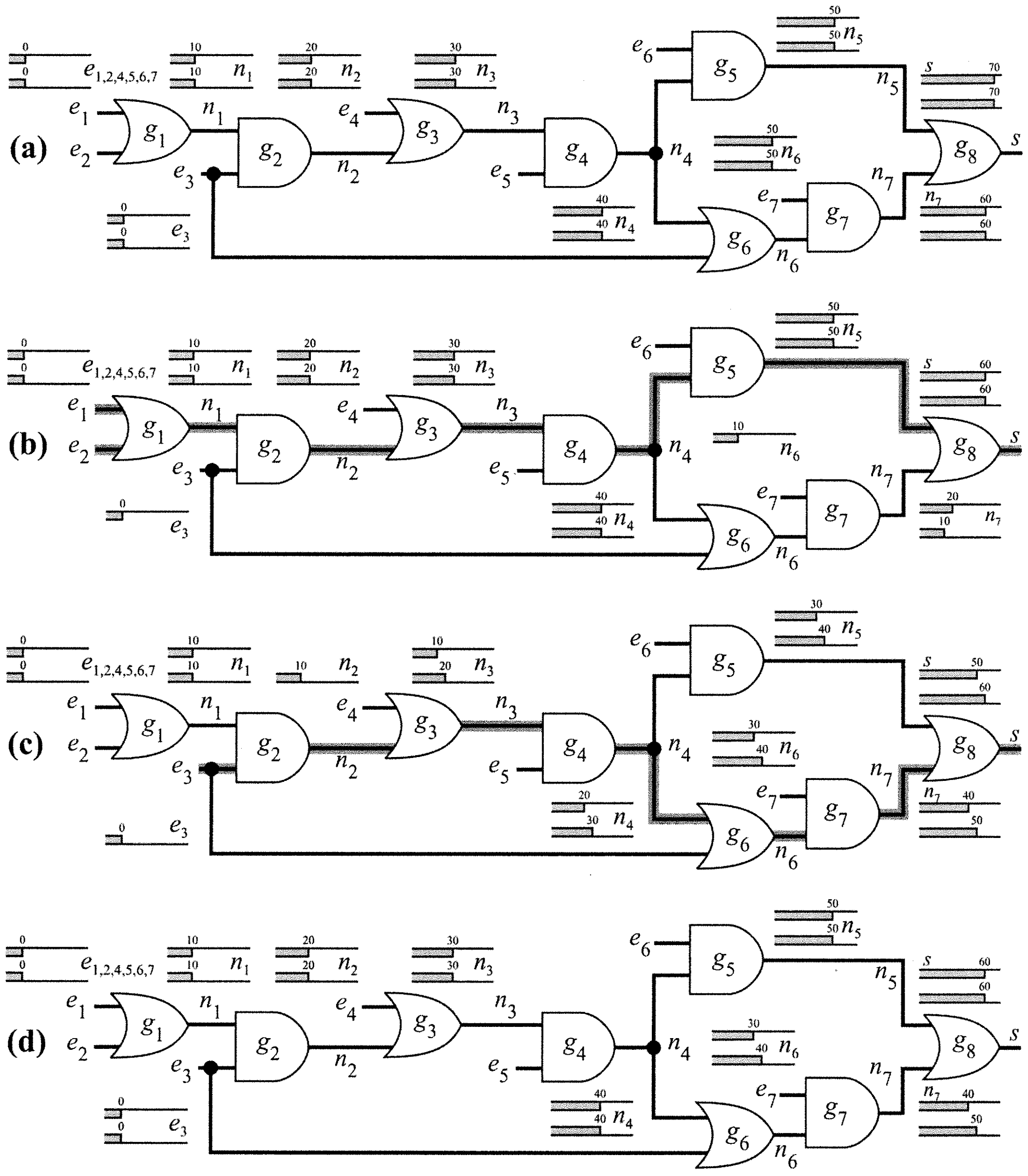


***Figure 90***
*Spatial correlation on reconvergent fan-out $e_3$.*
*(a) Domains before spatial correlation*
*(b) Domains when $e_3$ is restricted to waveforms stabilizing at 1*
*(c) Domains when $e_3$ is restricted to waveforms stabilizing at 0*
*(d) Domains after spatial correlation on $e_3$ (union of (b) and (c) )*

**Initial Domains:**

- Inputs stable after time 0: $D_{e_i} = (0|_{-\infty}^{0}, 1|_{-\infty}^{0}), i \in \{1, 2, 3, 4, 5, 6, 7\}$

- All waveforms for internal nodes: $D_{n_i} = (0|_{-\infty}^{+\infty}, 1|_{-\infty}^{+\infty}), i \in \{1, 2, 3, 4, 5, 6, 7\}$

- All waveforms for the output $s$: $D_s = (0|_{-\infty}^{+\infty}, 1|_{-\infty}^{+\infty})$

Applying the constraint operators of the gates yields (Fig. 90 (a) ):

$g_1 \Rightarrow D_{n_1} = (0|_{-\infty}^{10}, 1|_{-\infty}^{10})$ $\qquad$ $g_2 \Rightarrow D_{n_2} = (0|_{-\infty}^{20}, 1|_{-\infty}^{20})$

$g_3 \Rightarrow D_{n_3} = (0|_{-\infty}^{30}, 1|_{-\infty}^{30})$ $\qquad$ $g_4 \Rightarrow D_{n_4} = (0|_{-\infty}^{40}, 1|_{-\infty}^{40})$

$g_5 \Rightarrow D_{n_5} = (0|_{-\infty}^{50}, 1|_{-\infty}^{50})$ $\qquad$ $g_6 \Rightarrow D_{n_6} = (0|_{-\infty}^{50}, 1|_{-\infty}^{50})$

$g_7 \Rightarrow D_{n_7} = (0|_{-\infty}^{60}, 1|_{-\infty}^{60})$ $\qquad$ $g_8 \Rightarrow D_s = (0|_{-\infty}^{70}, 1|_{-\infty}^{70})$

all we can conclude here is that the circuit floating delay is $\leq 70$.

Save all domains as ALL;

Remove waveforms stabilizing at 0 in $e_3$ (Fig. 90 (b) )

$g_2 \Rightarrow D_{n_2} = (0|_{-\infty}^{20}, 1|_{-\infty}^{20})$ $\qquad$ $g_6 \Rightarrow D_{n_6} = (\phi, 1|_{-\infty}^{10})$

$g_7 \Rightarrow D_{n_7} = (0|_{-\infty}^{10}, 1|_{-\infty}^{20})$ $\qquad$ $g_8 \Rightarrow D_s = (0|_{-\infty}^{60}, 1|_{-\infty}^{60})$

the circuit floating delay is $\leq 60$ in this case.

Save all domains as SC1 and restore ALL;

Remove waveforms stabilizing at 1 in $e_3$ (see Fig. 90 (c) )

$g_2 \Rightarrow D_{n_2} = (0|_{-\infty}^{10}, \phi)$ $\qquad$ $g_3 \Rightarrow D_{n_3} = (0|_{-\infty}^{20}, 1|_{-\infty}^{10})$

$g_4 \Rightarrow D_{n_4} = (0|_{-\infty}^{30}, 1|_{-\infty}^{20})$ $\qquad$ $g_5 \Rightarrow D_{n_5} = (0|_{-\infty}^{40}, 1|_{-\infty}^{30})$

$g_6 \Rightarrow D_{n_6} = (0|_{-\infty}^{40}, 1|_{-\infty}^{30})$ $\qquad$ $g_7 \Rightarrow D_{n_7} = (0|_{-\infty}^{50}, 1|_{-\infty}^{40})$

$g_8 \Rightarrow D_s = (0|_{-\infty}^{60}, 1|_{-\infty}^{50})$

the circuit floating delay is $\leq 60$ in this case also.

Merge all domains with SC1; (Fig. 90 (d) )

Spatial correlation on stem $e_3$ tightened the domains of $n_6$, $n_7$, and $s$. In fact, it proved that the circuit floating delay is $\leq$ 60.

Note that, unlike the case analysis based on constant logic values [50, 60, 62], this procedure is safe and conservative.

**The reconvergent fan-outs are identified as follows:**

- Perform a breadth first search backward from the output (e.g., from s in Fig. 90), mark each node visited multiple times as a reconvergent fan-out;
- Perform a depth first search backward from the output, visiting the deeper structures first, and gather each node marked as reconvergent fan-out when its sub-structure is already gathered. (e.g., returns $\{e_3, n_4\}$ when applied to the circuit of Fig. 90, $e_3$ is deeper than $n_4$).

Appendix A (A.2) contains the function **getOrderedReconvergentFanouts(gate)** that returns the reconvergent fan-outs in topological order, deeper nodes first. The time complexity of this algorithm is linear with the graph size.

The procedure **correlateReconvergentFanouts(gate_set)** in Appendix A (A.4) performs a generalized spatial correlation that does $2^p$ system evaluations corresponding to the restriction of $p$ domains to waveforms stabilizing at 0 and 1 ($p$ is the number of gates of **gate_set**). The procedure **doSpatialCorrelation(order,s)** in Appendix A (A.3) determines the reconvergent fan-outs $\{r_1, r_2, ..., r_k\}$ of the logic cone of the output $s$; then it calls **correlateReconvergentFanouts(gate_set)** for ($k$ - *order* + 1) gate sets corresponding to $\{r1, r2,..., r_{order}\}$, $\{r2,..., r_{order+1}\}$, ... , $\{r_{k - order + 1},..., r_k\}$. The total number of system evaluations is $(2^{order} \times (k - order + 1))$. When $order \leq 6$ (a constant) and $k \in O($ graph size $)$ The overall time complexity of **doSpatialCorrelation(order, s)** is virtually quadratic for combinational circuits.

## 3.7.3 Global Timing Implications

In this section we introduce the notion of *static* and *dynamic timing dominators* to identify global implications related to the existence of transitions at or after a certain time on some key circuit nodes.

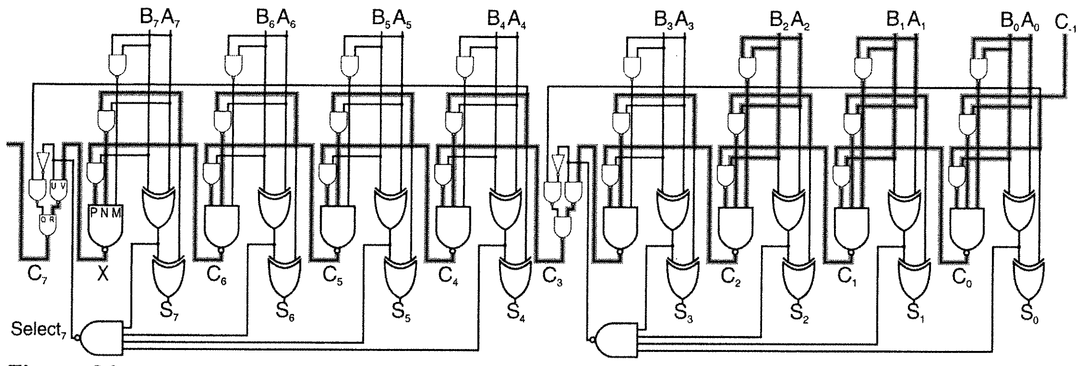


***Figure 91***
*16-bit carry skip adder.*

The propagation of the *last-transition interval* is the main mechanism that proves timing properties. In the Hrapcenko's false path example (Fig. 87), only one path was the potential carrier of the transitions at the output $s$. There was no ambiguity in deciding which net is the cause of the violation when the gate constraints were individually applied: at gate $g_8$ the constraint was able to decide that net $n_5$ cannot be the cause of the violation because its waveforms stabilize too early, $max(D_{n_5}.w_0.max, D_{n_5}.w_1.max) + 10 < min(D_s.w_0.lmin, D_s.w_1.lmin)$. This is why $1|_{-\infty}^{50}$ (stabilized too early at a controlling value) was narrowed to $\phi$ in $n_5$ and the *last-transition interval* was propagated to $n_7$. In more complex circuits, such as the carry skip adder in Figure 91, we may not be able to make such an unambiguous decision. Consider for example the output $C_7$. Suppose that the maximum path length from $C_2$ to $C_7$ is 750 and that the domain of $C_7$ contains waveforms that have transitions at or after time 750, i.e., $\sigma = (\xi_1, C_7, 750)$. Also, suppose that the paths from $C_7$ to the primary inputs that are longer than 750 constitute the shaded sub-graph of Fig. 91. The waveforms in the domains of the unshaded nets do not have transitions that may propagate to the *last-transition interval* of $C_7$, they stabilize too early. Fig. 92 shows a magnified view of the paths from X to $C_7$, and Fig. 93 (a) shows its constraint sub-graph. Referring always to Fig. 93, (b) shows how $g_1$ propagates the *last-transition interval* from $C_7$' to R, and removes the waveforms stabilizing at the controlling value from Q because they stabilize too early, (c) shows a similar behavior for $g_2$. In (d), the behavior of $g_3$ is different. It still removes the waveforms stabilizing at the controlling value from M because they stabilize too early, and propagates the *last-transition interval* from X.$w_1$ to P.$w_0$ and N.$w_0$, but it cannot propagate X.$w_0$ to P.$w_1$ or N.$w_1$ because the waveform stable at 1 in each domain is compatible with the constraint. The behavior on $g_3$ leads a more pessimistic result than necessary for this example. However, by examining the shaded sub-graph of Fig. 91, we can see that it converges to single nets on $C_6$, $C_5$, $C_4$, $C_3$, and $C_2$. Considering that the transitions in the last transition interval of $C_7$ traveled along the shaded nets, the waveforms in $C_6$ stabilizing before 750 minus the maximum path length from $C_6$ to $C_7$ cause the constraint system to be inconsistent, they are not part of any solution, therefore, they can be removed. In fact, $g_3$ could not decide whether the transitions in $X.w_0$ come from P or N, but examining the circuit graph we can decide that they necessarily come from $C_6$. $C_6$ is narrowed to waveforms having transitions at or after time 750 - the maximum path length from $C_6$ to $C_7$. The domains on nets $C_5$, $C_4$, $C_3$, and $C_2$ are narrowed in a similar way. These nets are defined in the next sections as *timing dominators*.

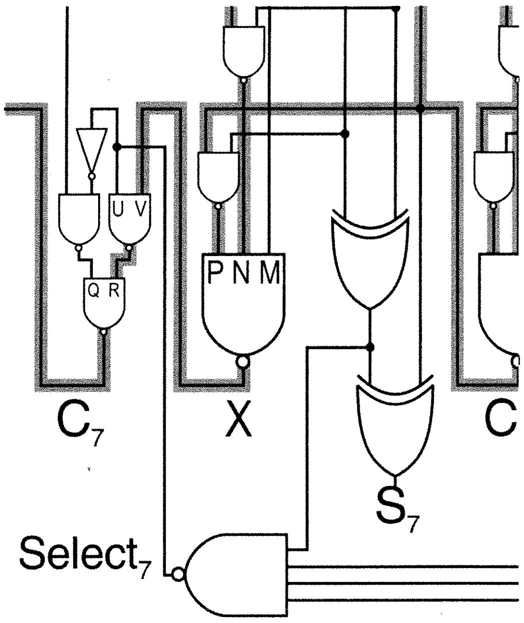


***Figure 92***
*Carry out of 16-bit carry skip adder.*

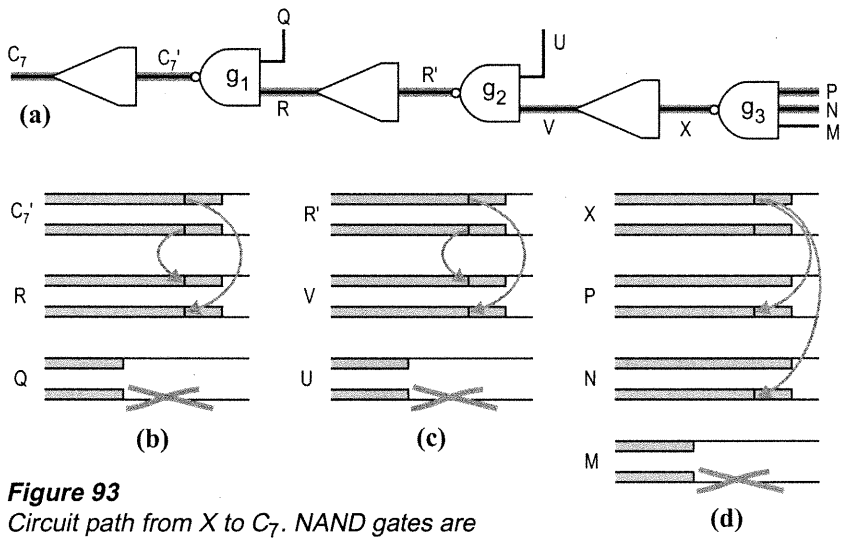


***Figure 93***
*Circuit path from X to $C_7$. NAND gates are represented as separate delays and delayless NAND constraints.*

### 3.7.4 Static Timing Dominators

**Static Carrier:** A net $x$ of $\xi$ is a ***static carrier*** of $\sigma = (\xi, s, \delta)$ iff $\exists$ path in $\xi$ containing $x$ and $s$ of length greater than or equal to $\delta$. Obviously, violation of the timing requirement (transitions at or after time $\delta$ at $s$) is caused by signal transitions traveling along the static carrier nets.

**Static Carrier Circuit:** The sub-circuit composed of the *static carriers of* $\sigma = (\xi, s, \delta)$ and their driving gates of $\xi$ is the ***static-carrier circuit*** of $\sigma$.

For example, the *static-carrier-circuit* of $\sigma = (\xi_1, C_7, 750)$ where $\xi_1$ is the circuit of Figure 91, is the sub-circuit of $\xi_1$ composed of the shaded nets and their driving gates.

**Static Timing Dominators:** Let $\Psi$ be the *static-carrier circuit* of $\sigma = (\xi, s, \delta)$. Let $\Psi'$ be a directed acyclic graph derived from $\Psi$ as follows: each net in $\Psi$ corresponds to a vertex in $\Psi'$; each gate in $\Psi$ with k inputs $x_1, x_2, \ldots, x_k$ and one output $x_0$ corresponds to k edges, from the vertex corresponding to $x_0$ to those corresponding to $x_i$, i=1 to k. We add a terminal vertex **T** to $\Psi'$, and we add an edge to **T** from each vertex of an input of $\Psi$. $\Psi'$ is a directed acyclic graph with one source vertex **S** (corresponding to $s$) and one sink vertex **T**. The nets of $\Psi$ corresponding to the dominators [2] of **T**, i.e., the vertices that lie on every path from **S** to **T**, are said to be the ***static timing dominators*** of $\sigma$.

For example, for $\sigma_1 = (\xi_1, C_7, 750)$ where $\xi_1$ is in Figure 91, $C_7$, X, $C_6$, $C_5$ are some of the *static timing dominators* of $\sigma_1$.

**Lemma:** Let $d$ be a *static dominator* of $\sigma = (\xi, s, \delta)$. Waveforms on $d$ that are stable at and after time ($\delta$ - maximum path length from $d$ to $s$) are $\sigma$*–incompatible*, i.e., make the constraint system inconsistent.

The proof is trivial, it follows from Lemma 6.1 in [8].

### 3.7.5 Dynamic Timing Dominators

The propagation of the *last-transition interval* of the output to the *static timing dominators* of the circuit represents global necessary assignments. Additional global implications can be determined by analyzing the contents of the *abstract signal* domains. This section presents a generalization of the concept of *static timing dominators*.

***k*-Dynamic Carrier:** Let $\sigma = (\xi, s, \delta)$ be a *timing-check*, and $C$ its associated constraint system. Let $D_s$ be the domain associated with output $s$. If $D_s \neq (\phi,\phi)$ then $s$ is said to be a ***0-dynamic carrier*** of $\sigma$. If net $y$ is a *k-dynamic-carrier* and it is the output of gate $g$ with max. delay $d_{max,}$ then an input net $x$ of gate $g$ is a *k'-dynamic-carrier* of $\sigma$ where $k' = (k + d_{max})$, provided that the domain $D_x$ of $x$ satisfies $D_x \cap (0|_{\delta - k'}^{+\infty}, 1|_{\delta - k'}^{+\infty}) \neq (\phi, \phi)$.

**Dynamic Carrier:** A net $x$ is said to be ***dynamic carrier*** of $\sigma$ iff $\exists\ k \geq 0$ such that $x$ is a *k-dynamic carrier* of $\sigma$.

**Dynamic Carrier Circuit:** Let $\Psi$ be the circuit composed of the *dynamic carriers of* $\sigma = (\xi, s, \delta)$ and their driving gates of $\xi$. $\Psi$ is said to be the ***dynamic-carrier circuit*** of $\sigma$.

**Dynamic Distance:** Let $\Psi$ be the *dynamic-carrier circuit* of $\sigma = (\xi, s, \delta)$ and $x$ be a net of $\Psi$. The maximum length of paths in $\Psi$ from $x$ to $s$ is the ***dynamic distance*** of $x$.

Intuitively, the *dynamic distance* of $x$ is the maximum time a transition at $x$ takes to reach $s$, and is equal to the highest value $k$ such that $x$ is a *k-dynamic carrier* of $\sigma$. In fact, the concept of *dynamic carriers* is formulated by necessary conditions for a net to be the cause of a violation of the timing check, and the domain of a net that is not a *dynamic carrier* of $\sigma = (\xi, s, \delta)$ does not contain transitions that propagate to the *last-transition interval* of $s$.

**Dynamic Timing Dominators:** Let $\Psi$ be the *dynamic-carrier circuit* of $\sigma = (\xi, s, \delta)$. Let $\Psi'$ be a directed acyclic graph derived from $\Psi$ as follows: each net in $\Psi$ corresponds to a vertex in $\Psi'$; each gate in $\Psi$ with k inputs $x_1, x_2, \ldots, x_k$ and one output $x_0$ corresponds to k edges, from the vertex corresponding to $x_0$ to those corresponding to $x_i$, i=1 to k. We add a terminal vertex **T** to $\Psi'$, and we add an edge to **T** from each vertex of an input of $\Psi$. $\Psi'$ is a directed acyclic graph with one source vertex **S** (corresponding to $s$) and one sink vertex **T**. The nets of $\Psi$ corresponding to the dominators [2] of **T**, i.e., the vertices that lie on every path from **S** to **T**, are said to be the ***dynamic timing dominators*** of $\sigma$.

**Theorem 3:** For a *timing-check* $\sigma = (\xi, s, \delta)$ and a *dynamic timing dominator d*, let $k$ be the largest integer such that $d$ is *k-dynamic carrier* of $\sigma$. The waveforms in $d$ that are stable at and after time $(\delta - k)$ are *$\sigma$–incompatible*, i.e., they necessarily make the constraint system inconsistent.

***Proof:*** Theorem 3 is a direct consequence of the fact that any net $x \notin \Psi$ is not the source of a timing violation, i.e., $D_x$ do not contain transitions that propagate to within the *last-transition interval* of $D_s$. This fact can be proven by contradiction: Suppose that there is a path $p = (x, g_{k_0}, n_{k_1}, \ldots, n_{k_p}, g_{k_p}, s)$ such that the domain of $x$ contains transitions that propagate along $p$ to the *last-transition interval* of $D_s$. This implies that the same property is true for all the nets of $p$. Then $n_{k_p}$ has transitions at or after time $(\delta - \text{max. delay of } g_{k_p})$ and consequently $n_{k_p}$ is $(\delta - \text{max. delay of } g_{k_p})$*-dynamic carrier* of $\sigma$. Similarly $x$ is $(\delta - \textit{length of } p)$*-dynamic carrier* of $\sigma$. $x \in \Psi$ contradicts the original assumption. This proves that transitions on internal nets of the circuit that cause transitions on $s$ at or after time $\delta$ (a violation of the timing check) *if any* are originating from $\Psi$. Any waveform on a *dynamic dominator* that is stable at and after time $(\delta - \text{its } \textit{dynamic-distance})$ makes the constraint system inconsistent.

**Corollary:** Let $d$ be a *dynamic dominator* of $\sigma = (\xi, s, \delta)$ and $k$ the *dynamic distance* of $d$. Narrowing the domain of $d$ by intersecting it with $(1|_{\delta-k}^{+\infty}, 0|_{\delta-k}^{+\infty})$ maintains all the solutions of the original system.

The proof follows from Theorem 3.

To determine the timing dominators, a back trace is started at the output. The fan-in gates that are in the *dynamic carrier circuit* are scheduled on a priority queue that returns the closest gate to the output when queried. Each scheduled gate keeps track of its *dynamic distance* which is updated when the gate is scheduled multiple times. A node $d$ is a dynamic dominator if the queue becomes empty after $d$ is removed from it. The function **getTimingDominators(output)** in Appendix A (A.5) is a high level algorithm that determines the timing dominators. The worst case time complexity of the algorithm is $n \times \log(n)$ ($n$ is the circuit graph size). In practice, the apparent complexity is close to linear as the number of gates scheduled at the same time is small due to practical circuit topologies, and to the fact that, in most cases, the *dynamic carrier circuit* is a small fraction of the logic circuit. In fact, it is possible to determine the timing dominators in linear time by first determining the *dynamic carrier circuit*, but this turned out to be more expensive than the use of a priority queue in most cases. The function **evaluateConstraintSystemTD(output)** in Appendix A (A.6) is a high level algorithm that evaluates the constraint system applying the additional narrowing on the timing dominators.

Timing dominators concept is a very important contribution to the waveform narrowing method, it leads to a powerful mechanism for reducing the pessimism of the method. For example, the c1908-ALL-GATE circuit of the ISCAS'85 benchmark suite is a traditionally difficult case for combinational circuit delay calculation, and yet its timing properties were proven very efficiently when timing dominators were determined. For example, the output output_57 has a topological delay of 400. We restricted its domain to waveforms having transitions at or after time 241 and the constraint system ended up with

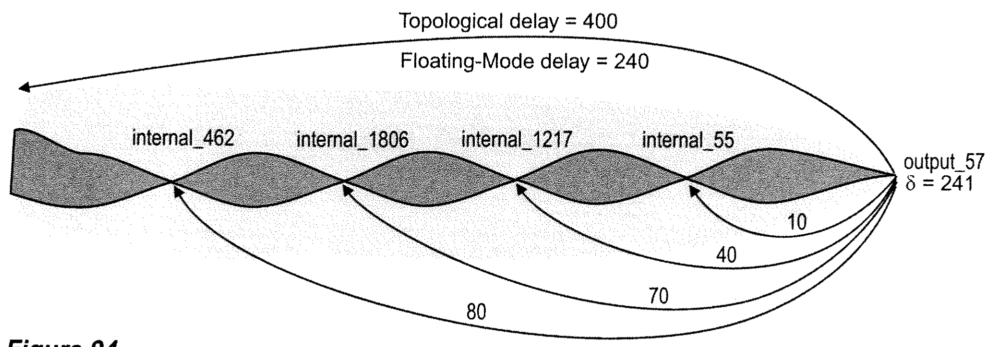


***Figure 94***
*Timing dominators of output_57 of c1908-ALL-GATES.*

non-empty domains. This particular timing check has four timing dominators beside the output itself, no narrowing was performed on three of them (internal_462, internal_1806, and internal_1217) by the original method (see Fig. 94). After narrowing the domains of these dominators to waveforms having transitions at or after time $\delta - DynamicDistance$ (e.g., 161 for internal_462) the constraint system evaluation resulted in empty domains, proving that the actual floating delay of output_57 is $< 241$.

## 3.8 Case Analysis

When the evaluation of the constraint system associated with a timing-check $\sigma = (\xi, s, \delta)$ ends up with non-empty domains, we cannot definitely conclude that a violation is possible. This section presents an algorithm for case analysis that does a decision tree traversal by selecting nets, and restricting their domains to waveforms stabilizing at either 0 or 1. The objective is to find a test vector or to prove that the constraint system is actually inconsistent, i.e., has no solution. A test vector is found when each circuit domain is *decided*, i.e., is non-empty and contains waveforms stabilizing at either 0 or 1 exclusively. Obviously, we are dealing with an NP-Hard problem. Therefore, the algorithm relies heavily on heuristics, and has exponential time complexity in the worst case. The outcome of case analysis can be one of the following three:

- CS_INCONSISTENT, the constraint system has no solution, no timing violation is possible. This happens when the constraint system ends up with empty domains on all decision tree leaves;
- TEST_VECTOR, all circuit nodes were successfully restricted to one abstract waveform (0 or 1), and the constraint system is consistent. In this case we consider that the assignment of the primary inputs is a test vector that violates the timing requirements;
- ABANDONED, the case analysis is abandoned due to excessive number of backtracks.

The general scheme for the decision tree traversal is depicted in the following recursive algorithm:

```
FindTestVector() {
    Evaluate the constraint system;
    if ( result is empty domains ) return CS_INCONSISTENT;
    if ( all nets are decided ) return TEST_VECTOR;
    select one net N that is not yet decided and a value V;    // V = 0 or 1
    save system state;
    restrict N to V;
    result = FindTestVector();
    if ( result == CS_INCONSISTENT ) {
        restore system state;
        restrict N to $\overline{V}$;
        result = FindTestVector()
    }
    return result;
}
```

The key for good performance is an appropriate net selection heuristic. As a general rule of thumb, obvious easy decisions should be made as late as possible in the decision process. In fact, when made early in the decision tree, decisions that have conflicting

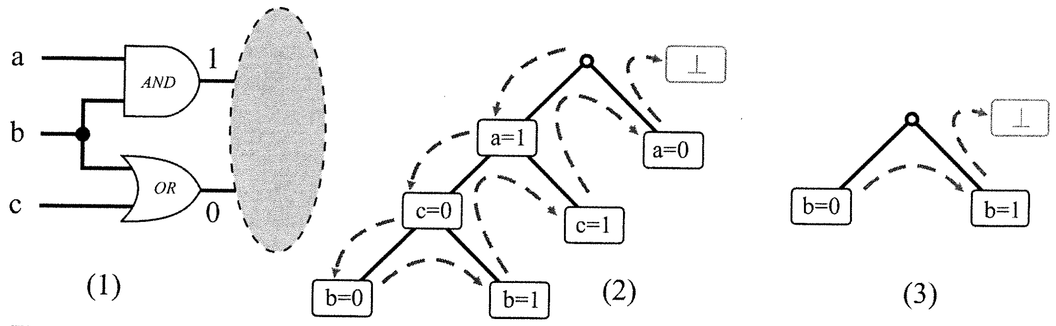


***Figure 95***
*(1) Requirements impossible to satisfy. (2) Decision tree when b is chosen the last.*
*(3) Decision tree when b is chosen the first.*

implications are likely to prune a large part of the search space. Consider for example the circuit in Fig. 95(1) for which our objective is to find an assignment for a, b, and c that sets the AND gate output to 1, and the OR gate output to 0. Six decisions are needed to prove that the requirements are impossible to satisfy when b is decided the last (see Fig. 95(2)). In contrast, only two decisions are needed when b is selected first (see Fig. 95(3)). Conflicting implications can be determined easily by propagating backward the requirements. Fig. 96 shows how this is done for the previous example, the AND gate requires 1 for a and b, whereas the OR gate requires 0 for b and c. Therefore, b is required to be at 0 and 1, hence it is likely to have conflicting implications. This is the basis for a very successful strategy used in test pattern generation. The back trace mechanism was introduced by Goel [32] as the PODEM algorithm, and then refined by Fujiwara and Shimono [33] in the FAN algorithm as the multiple back trace procedure. The PODEM algorithm back traced objectives all the way to primary inputs, one path at a time, according the following rules:

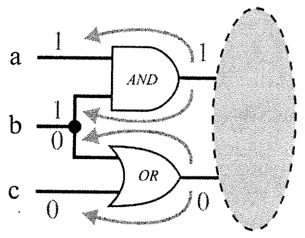


***Figure 96***
*Requirement propagation.*

- If our objective can be satisfied by setting a gate input to a controlling value, e.g., 0 for AND / NAND gates, then choose that input that can be "most easily" set.
- If our objective can be satisfied by setting all gate inputs to a non-controlling value, e.g., 1 for AND / NAND, then start with that input that is the "hardest" to set.

The multiple back trace procedure follows multiple paths in a reverse levelized order. Decisions are made on fan-out branches that receive conflicting requirements, e.g., fan-out b in the example of Fig. 96. We adopted the multiple back trace strategy to select nets and decisions. The objectives to satisfy, however, are derived from a timing verification problem rather than test pattern generation one. For the timing check $\sigma = (\xi, s, \delta)$, case analysis main goal is to find a test vector that produce transitions later than $\delta$. Therefore, we need to sensitize one path in $\Psi$, the *dynamic carrier circuit* of $\sigma$. This is done incrementally, starting from the output s, favoring the longest paths in $\Psi$.

Section 3.8.1 defines the controllability measure we used to distinguish between "easy" and "hard" to set nets; Section 3.8.2 defines the back trace procedures; and Section 3.8.3 contains the case analysis algorithm.

### 3.8.1 Controllability Measure

*Controllability* is a heuristic measure associated with circuit nodes. It represents the cost for finding a test vector that, when applied to primary inputs, sets the node to 0 or 1. For instance, suppose that it is desirable to set the output of an AND gate to 0, this can be achieved by setting any of the gate's inputs to 0. The controllability measure is used to choose the easiest. *Controllability* is a subject fairly well studied in literature [22]-[30], we used SCOPE [23].

SCOPE controllability is defined as a couple ($C_0$, $C_1$), and calculated as follows:

- Primary inputs are assigned ($C_0$=1, $C_1$=1)
- For a circuit net driven by a gate G, $C_x$ = (minimum cost of input assignments that produce $x$) + 1.

for an AND gate where Y is the output, A and B are the inputs:

$C_0(Y) = \min(C_0(A), C_0(B)) + 1$

$C_1(Y) = C_1(A) + C_1(B) + 1$

for a NAND gate where Y is the output, A and B are the inputs:

$C_0(Y) = C_1(A) + C_1(B) + 1$

$C_1(Y) = \min (C_0(A), C_0(B)) + 1$

for an OR gate where Y is the output, A and B are the inputs:

$C_0(Y) = C_0(A) + C_0(B) + 1$

$C_1(Y) = \min (C_1(A), C_1(B)) + 1$

for a NOR gate where Y is the output, A and B are the inputs:

$C_0(Y) = \min (C_1(A), C_1(B)) + 1$

$C_1(Y) = C_0(A) + C_0(B) + 1$

for a XOR gate where Y is the output, A and B are the inputs:

$C_0(Y) = \min (C_1(A) + C_1(B) , C_0(A) + C_0(B)) + 1$

$C_1(Y) = \min (C_0(A) + C_1(B) , C_1(A) + C_0(B)) + 1$

for an XNOR gate where Y is the output, A and B are the inputs:

$C_0(Y) = \min (C_0(A) + C_1(B) , C_1(A) + C_0(B)) + 1$

$C_1(Y) = \min (C_1(A) + C_1(B) , C_0(A) + C_0(B)) + 1$

for a NOT gate where Y is the output, A is the input:

$C_0(Y) = C_1(A) + 1$

$C_1(Y) = C_0(A) + 1$

for a BUFFER or DELAY gate where Y is the output, A is the input:

$C_0(Y) = C_0(A) + 1$

$C_1(Y) = C_1(A) + 1$

This measure represents the number of nodes in the equivalent fan-out-free circuit that need to be set to a specific value in order to set the gate output to 0 or 1. *Controllability* functions for other gates can be easily formulated.

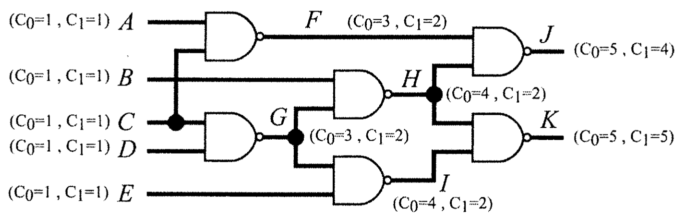


***Figure 97***
*Controllability measure for circuit c17.*

Fig. 97 shows the controllability measures calculated for the circuit c17 of the ISCAS'85 benchmark suite.

## 3.8.2 Requirement Propagation / Back Trace Procedure

Logic requirements are presented in [33] as *objectives*: an *objective* is a triplet (*Net*, $n_0$(*Net*), $n_1$(*Net*)). *Net* is the net identifier where the *objective* is attached, $n_0$(*Net*) ($n_1$(*Net*)) is the number of times *Net* is required to be set to 0 (1). *Objectives* serving our purpose have different semantics for $n_0$(*Net*) and $n_1$(*Net*). $n_0$(*Net*) ($n_1$(*Net*)) is the potential length of the path that is likely to be enabled if the domain of *Net* is restricted to waveforms stabilizing at 0 (1). *Objectives* are propagated from a gate output to its inputs (*back traced*) according to the gate type and the *controllability* measures of the gate inputs. During backtracing, objectives are put in a priority queue that returns the objective attached to the net that is the closest to the circuit primary output (in our case, the circuit has only one output). Backtracing proceeds by pulling an *objective* out of the priority queue, calculating the *objectives* for the inputs of the gate driving the *objective's* net, and inserting back the calculated *objectives* in the queue. When an *objective* is being inserted in the queue, and another one for the same net is already queued, they are merged into one that gets the largest values. For instance, merging (*Net*, 100, 200) and (*Net*, 50, 300) results in (*Net*, 100, 300). This deviates from the back trace procedure defined in [33] where the merged objective gets the sum of the values.

The following algorithms calculate the objectives for the inputs of the different logic gate types. $C_0$(X), $C_1$(X) are the controllability measures for the net X.

for a BUFFER or a DELAY gate where Y is the output, A is the input:

```
$n_1(A) = n_1(Y)$
$n_0(A) = n_0(Y)$
```

for a NOT gate where Y is the output, A is the input:

```
$n_1(A) = n_0(Y)$
$n_0(A) = n_1(Y)$
```

for an AND gate where Y is the output, A and B are the inputs:

```
$n_1(A) = n_1(Y)$
$n_1(B) = n_1(Y)$
if( $C_0(A) < C_0(B)$ ) {
    $n_0(A) = n_0(Y)$
    $n_0(B) = 0$
} else {
    $n_0(B) = n_0(Y)$
    $n_0(A) = 0$
}
```

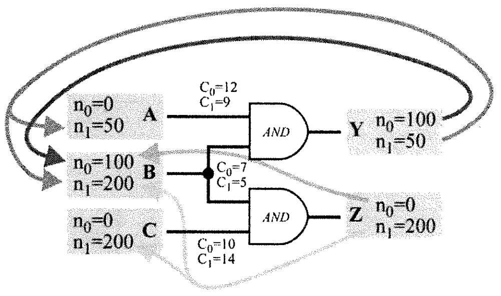


***Figure 98***
*Back Trace example.*

Fig. 98 shows a back trace example: (Y, 100, 50) is back traced as (A, 0, 50) and (B, 100, 50); (Z, 0, 200) is back traced as (B, 0, 200) and (C, 0, 200); the objectives at B are merged as (B, 100, 200). B has a conflicting requirements, the case analysis sets B to 1 as a first attempt, to favor the path of length 200.

for a NAND gate where Y is the output, A and B are the inputs:

```
$n_1(A) = n_0(Y)$
$n_1(B) = n_0(Y)$
if( $C_0(A) < C_0(B)$ ) {
    $n_0(A) = n_1(Y)$
    $n_0(B) = 0$
} else {
    $n_0(B) = n_1(Y)$
    $n_0(A) = 0$
}
```

for an OR gate where Y is the output, A and B are the inputs:

```
$n_0(A) = n_0(Y)$
$n_0(B) = n_0(Y)$
if( $C_1(A) < C_1(B)$ ) {
    $n_1(A) = n_1(Y)$
    $n_1(B) = 0$
} else {
    $n_1(B) = n_1(Y)$
    $n_1(A) = 0$
}
```

for a NOR gate where Y is the output, A and B are the inputs:

```
$n_0(A) = n_1(Y)$
$n_0(B) = n_1(Y)$
if( $C_1(A) < C_1(B)$ ) {
    $n_1(A) = n_0(Y)$
    $n_1(B) = 0$
} else {
    $n_1(B) = n_0(Y)$
    $n_1(A) = 0$
}
```

for a XOR gate where Y is the output, A and B are the inputs:

```
if( $C_0(A) + C_0(B) < C_1(A) + C_1(B)$ ) {
    if( $C_0(A) + C_1(B) < C_1(A) + C_0(B)$ ) {
        $n_0(A) = \max(n_0(Y), n_1(Y))$
        $n_0(B) = n_0(Y)$
        $n_1(A) = 0$
        $n_1(B) = n_1(Y)$
    } else {
        $n_0(A) = n_0(Y)$
        $n_0(B) = \max(n_0(Y), n_1(Y))$
        $n_1(A) = n_1(Y)$
        $n_1(B) = 0$
    }
} else {
    if( $C_0(A) + C_1(B) < C_1(A) + C_0(B)$ ) {
        $n_0(A) = n_1(Y)$
        $n_0(B) = 0$
        $n_1(A) = n_0(Y)$
        $n_1(B) = \max(n_0(Y), n_1(Y))$
    } else {
        $n_0(A) = 0$
        $n_0(B) = n_1(Y)$
        $n_1(A) = \max(n_0(Y), n_1(Y))$
        $n_1(B) = n_0(Y)$
    }
}
```

The back trace procedures for XOR and XNOR gates are more complicated as 0 and 1 are both non-controlling values. They were defined in [35].

for an XNOR gate where Y is the output, A and B are the inputs:

```
if( $C_0(A) + C_0(B) < C_1(A) + C_1(B)$ ) {
    if( $C_0(A) + C_1(B) < C_1(A) + C_0(B)$ ) {
        $n_0(A) = \max( n_0(Y), n_1(Y) )$
        $n_0(B) = n_1(Y)$
        $n_1(A) = 0$
        $n_1(B) = n_0(Y)$
    } else {
        $n_0(A) = n_1(Y)$
        $n_0(B) = \max( n_0(Y), n_1(Y) )$
        $n_1(A) = n_0(Y)$
        $n_1(B) = 0$
    }
} else {
    if( $C_0(A) + C_1(B) < C_1(A) + C_0(B)$ ) {
        $n_0(A) = n_0(Y)$
        $n_0(B) = 0$
        $n_1(A) = n_1(Y)$
        $n_1(B) = \max( n_0(Y), n_1(Y) )$
    } else {
        $n_0(A) = 0$
        $n_0(B) = n_0(Y)$
        $n_1(A) = \max( n_0(Y), n_1(Y) )$
        $n_1(B) = n_1(Y)$
    }
}
```

### 3.8.3 Case Analysis Procedure

The main idea is to compute the *initial objectives* so as to set those nets which are inputs of gates in the *dynamic-carrier circuit* $\Psi$ of $\sigma$ that are not *dynamic carriers* to a *non-controlling* value regarding the gates they feed into $\Psi$. This strategy is justified by the following reasoning: The timing violation at output $s$ is originating in $\Psi$, hence we need to sensitize the paths in $\Psi$. Let's illustrate how the initial objectives are established on the

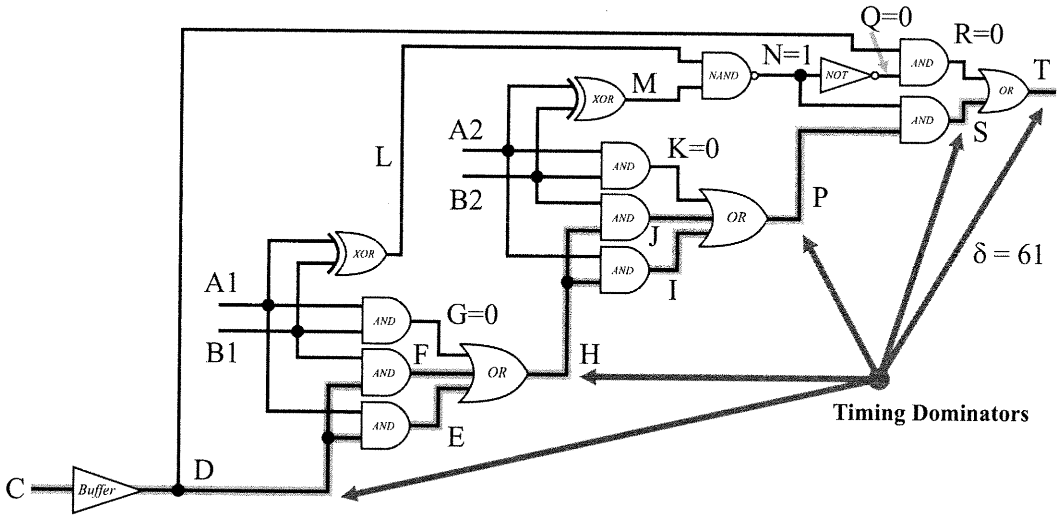


***Figure 99***
*Timing check* $(\xi, T, 61)$ *for a carry skip circuit.*

example of Fig. 99. This is a carry skip circuit having a topological delay of 70 (all gates have a delay value of 10). The timing check compares the circuit delay with 61. The constraint system evaluation ended with non-empty domains. The *dynamic carrier circuit* is the shaded sub-circuit, it contains six timing dominators: T, S, P, H, D, and C. The following is a partial list of domain contents:

$$T = (0|_{61}^{70}, 1|_{61}^{70}) \qquad S = (0|_{51}^{60}, 1|_{51}^{60}) \qquad P = (0|_{41}^{50}, 1|_{41}^{50})$$
$$H = (0|_{21}^{30}, 1|_{21}^{30}) \qquad D = (0|_{1}^{10}, 1|_{1}^{10}) \qquad C = (0|_{-9}^{0}, 1|_{-9}^{0})$$

N is restricted to waveforms stabilizing at 1, non-controlling for AND. We denote this as N = 1 in order to simplify the presentation.

N = 1 Q = 0 R = 0 K = 0 G = 0

The remaining nets have undecided domains (both classes, $w_0$ and $w_1$, are non-empty). R and Q need no justification, they are not considered as objectives. However, N, K, and G are still *unjustified*. A net is *unjustified* if it is possible to restrict the domains of the inputs of its driving gate to single no empty classes ($w_0$ or $w_1$) and end up with an inconsistent situation. The following objectives are selected:

| | |
|---|---|
| (N, $n_0 = 0$, $n_1 = +\infty$) | $+\infty$ is used to indicate that this objective is mandatory to satisfy |
| (K, $n_0 = +\infty$, $n_1 = 0$) | |
| (G, $n_0 = +\infty$, $n_1 = 0$) | |
| (A2, $n_0 = 0$, $n_1 = 70$) | A2 is feeding an AND gate in the *dynamic carrier circuit*, the longest path that is potentially enabled by setting A2 to 1 is of length 70. |
| (B2, $n_0 = 0$, $n_1 = 70$) | |
| (A1, $n_0 = 0$, $n_1 = 70$) | |
| (B1, $n_0 = 0$, $n_1 = 70$) | |

Once these seven objectives are put in the priority queue, the basic idea is to start the backtracing process, and to apply decisions to the domains of the fan-out nets for which the objective have both $n_0 \neq 0$ and $n_1 \neq 0$. The decision is to restrict the domain to waveforms stabilizing at 0 (1) when $n_0 > n_1$ ($n_0 \leq n_1$). The actual algorithm is more complex, and is presented later in this section. Let us start first by stating the algorithm that computes the initial objectives:

The algorithm is parameterized by a starting and ending points, both of them are timing dominators. ComputeInitialObjectives determines the objectives that sensitize the sub-paths of the *dynamic carrier circuit* that lie between start and end (see Fig. 100).

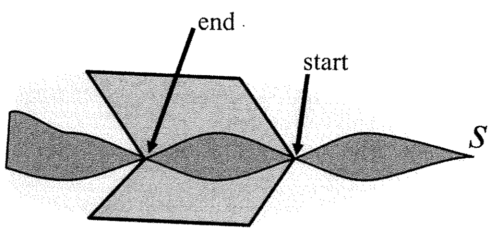


***Figure 100***
*Topological partitioning for decision making.*

```
function ComputeInitialObjectives($(\xi, s, \delta)$, start, end) {
    compute $\Psi$ the dynamic-carrier-circuit of $(\xi, s, \delta)$;
    if (end $\neq$ none) remove end from $\Psi$;
    restrict $\Psi$ to the cone of start;
    List = {gates in $\Psi$ having inputs not in $\Psi$}
    H = $\phi$;          // H is a heap returning objectives in reversed topological order
    for each gate G in List do {
        for each net N, input to G not in $\Psi$ do {
            if domain of N contains both classes then {        // N is not yet decided
                if G has a controlling value then {            // i.e., if G is not xor or xnor
                    if the non-controlling value is 0 then
                        add the objective (N, dynamic-distance of N + max(N), 0) to H;
                            // max(N) is the largest of $N.w_0.max$ and $N.w_1.max$
                    else add the objective (N, 0, dynamic-distance of N + max(N)) to H;
                    // when H already contains an objective corresponding to N, adding the new objective
                    // results in updating the already included one so as to contain the largest
                    // of $n_0$ and $n_1$.
                }
            }
            else {          // N is already decided
                for each unjustified net N' in the cone of N do {
                    if (the non-empty class of N' is 0) then add (N',$+\infty$,0) to H;
                    else add (N',0,$+\infty$) to H;
                }
            }
        }
    }
    return H;
}
```

In the context of test pattern generation, backtracing is restarted a minimal number of times. In our case such a strategy resulted in poor performance because decisions on nets may have profound effect on $\Psi$, the source of the violation. The back trace is restarted each time the size of the decision stack changes as a result of backtracks. Moreover, decisions are performed in 4 phases:

**Phase 1:** Let $d_0, d_1, \ldots, d_k$ be the consecutive *dynamic-dominators* of $(\xi, s, \delta)$ computed before any decision is taken, $(d_0 = s)$. Let $\xi_{d_i, d_{i+1}}$ be the sub-circuit of $\xi$ composed of the fan-in cone of $d_i$ excluding $d_{i+1}$. We fix the *class* value of nets in $\xi_{d_i, d_{i+1}}$, $i = 0$ to $k$-1, using the function MakeDecisions($(\xi, s, \delta)$, $d_i$, $d_{i+1}$). Then, we fix the *class* of nets in the fan-in cone of $d_k$ using MakeDecisions($(\xi, s, \delta)$, $d_k$, *none*). MakeDecisions fixes the *class* only on those fan-out nets with conflicting objectives, i.e., both $n_0$ and $n_1 \neq 0$.

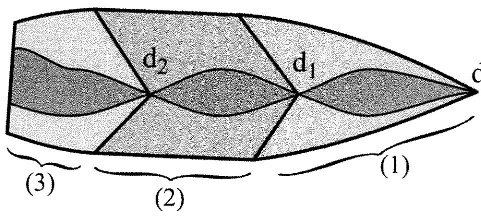


***Figure 101***
*Topological partitioning for phase 1.*

Fig. 101 shows a partitioning example for three *timing dominators*, $d_0$, $d_1$, and $d_2$. Phase 1 fix the class of fan-out nodes in regions (1) first, then (2) then (3).

**Phase 2:** We perform decisions on the whole circuit using MakeDecisions($(\xi, s, \delta)$, *s*, *none*).

**Phase 3:** We perform decisions on the whole circuit using MakeAllDecisions($(\xi, s, \delta)$). Decisions in phases 2 and 3 are taken on fan-outs with conflicting requirements, i.e., $n_0$ and $n_1$ are both non-zero, whereas decisions here are taken on all fan-outs.

**Phase 4:** We perform decisions on the primary inputs after complete back trace from all *unjustified* nets. An output of a gate G is *unjustified* iff its domain is restricted to one *class* and if we can intersect the domain on each input with $(0|_{-\infty}^{+\infty}, \phi)$ or $(\phi, 1|_{-\infty}^{+\infty})$ to get non-empty input domains that are inconsistent with the gate constraint.

## Case analysis algorithm:

```
function CaseAnalysis($(\xi, s, \delta)$) {
1.  compute dynamic-dominators of $(\xi, s, \delta)$;
    for each successive pair of dominators $d_i$ and $d_j$, starting from $s$ do {
        if (MakeDecisions($(\xi, s, \delta)$, $d_i$, $d_j$) = = CS_INCONSISTENT) return CS_INCONSISTENT;
    }
    if (MakeDecisions($(\xi, s, \delta)$, $d$, none) = = CS_INCONSISTENT) return CS_INCONSISTENT;
        // decisions on the cone of the deepest dynamic-dominator d.
2.  if (MakeDecisions($(\xi, s, \delta)$, $s$, none) = = CS_INCONSISTENT) return CS_INCONSISTENT;
        // decisions on the whole circuit
3.  if (MakeAllDecisions($(\xi, s, \delta)$) = = CS_INCONSISTENT) return CS_INCONSISTENT;
4.  Compute objectives for all unjustified nets;
    Perform complete back trace, to the primary inputs;
    while (decision stack not empty and not all inputs decided) make decision for a primary input;
    if (decision stack is empty) return CS_INCONSISTENT; else return TEST_VECTOR;
}
```

```
function MakeDecisions( $(\xi, s, \delta)$ , start, end) {
    H = ComputeInitialObjectives( $(\xi, s, \delta)$ , start, end);
    while (H not empty) {
        remove the objective for the net closest to s from H and place it into obj;
        if (N the net of obj is a fan-out) {
            if (both $n_0$ and $n_1 \neq 0$) {
                if (both classes of the domain of N are not empty) {
                    if ($n_0 > n_1$) then remove, i.e., make empty, class 1 of the domain of N;
                    else remove class 0;
                    while (constraint system is inconsistent) {
                        backtrack;
                        if (decision stack empty) return CS_INCONSISTENT;
                    }
                    if (the decision stack changed size because of backtracks)
                    H = ComputeInitialObjectives( $(\xi, s, \delta)$ , start, end);
                }
                if (N is an output of a gate G) {
                    if (class 1 of the domain of N is empty) back trace (N, $+\infty$ ,0);
                    else back trace (N,0, $+\infty$ );
                    // back trace puts the computed objectives corresponding
                    // to inputs of G on H
                }
            }
            else {
                if (N is an output of a gate G) {
                    if (both classes of the domain of N are not empty) back trace obj;
                    else {
                        if (class 1 of the domain of N is empty) back trace (N, $+\infty$ ,0);
                        else back trace (N,0, $+\infty$ );
                    }
                }
            }
        }
        else {
            if (N is an output of a gate G) {      // i.e., not an input
                if (both classes of the domain of N are not empty) back trace obj;
                else {
                    if (class 1 of the domain of N is empty) back trace (N, $+\infty$ ,0);
                    else back trace (N,0, $+\infty$ );
                }
            }
        }
    }
}
```

```
function MakeAllDecisions($(\xi, s, \delta)$) {
    H = ComputeInitialObjectives($(\xi, s, \delta)$, $s$, none);
    while (H not empty) {
        remove the objective for the net closest to $s$ from H and place it into obj;
        if (N the net of obj is a fan-out) {
            if (both classes of the domain of N are not empty) {
                if ($n_0 > n_1$) then remove, i.e., make empty, class 1 of the domain of N;
                else remove class 0;
                while (constraint system inconsistent) {
                    backtrack;
                    if (decision stack empty) return CS_INCONSISTENT;
                }
                if (the decision stack changed size because of backtracks) {
                    if (MakeDecisions($(\xi, s, \delta)$, $s$, none) = = CS_INCONSISTENT)
                            return CS_INCONSISTENT;
                    H = ComputeInitialObjectives($(\xi, s, \delta)$, $s$, none);
                }
            }
            if (N is an output of a gate G) {
                if (class 1 of the domain of N is empty) back trace (N,$+\infty$,0);
                else back trace (N,0,$+\infty$);
                // back trace puts the computed objectives corresponding
                // to inputs of G on H
            }
        }
        else {
            if (N is an output of a gate G) {
            if (both classes of the domain of N are not empty) back trace obj;
            else {
                if (class 1 of the domain of N is empty) back trace (N,$+\infty$,0);
                else back trace (N,0,$+\infty$);
            }
        }
    }
}
```

The next section presents the experimental results of applying the basic waveform narrowing method and the pessimism reduction techniques to the ISCAS'85 benchmark suite.

## 3.9 Experimental Results

The experiments were executed on a Sun SPARCstation 10 (120 MIPS). The basic constraint system evaluation without global implications on *timing dominators* was able to eliminate timing check violations for the circuits c5315 and c7552 of the NOR-gate implementations of the ISCAS'85 benchmarks [11] with delays of 10 on the outputs of all gates. The global implications on *timing dominators* eliminated timing violations from c1908 and c3540. Spatial correlation of order 1 eliminated timing check violations from c2670 and c6288. The case analysis found test vectors for all circuits except c6288. Table IV contains the detailed checks and Table V summarizes the results. Note that the value of δ for which a test vector is found represents the exact floating-mode delay of the circuit when the constraint system is inconsistent for (δ + 1) on all outputs. The columns of Table IV contain, from left to right, the following information: 1) the circuit name, 2) the output on which the timing check was done, 3) the max. topological delay of the output, 4) the max. topological delay of the circuit, 5) the timing constraint δ on the output, 6) the result of the first evaluation of the constraint system before the use of global implications on *timing dominators*, 7) the result after the use of global implications on *timing dominators*, 8) the result after spatial correlation of order 1 is applied, 9) the number of backtracks in the case analysis, 10) the result of case analysis, and 11) the total CPU time. Not included in Table IV is the timing check performed on a 16 bit carry-skip adder, partly shown in Fig. 91. The adder has a topological delay of 2000 and a floating-mode delay of 1000. This was determined in 25 seconds of CPU time after a total of 1636 backtracks. For δ =1001 the case analysis proved that the constraint system is inconsistent on all outputs, and for δ=1000 found a test vector.

The global implications on *timing dominators* proved to be very effective in the case of the traditionally difficult c1908 circuit. It proved that output 57_912 (topological delay of 340) has a delayless than or equal to 200 in 0.76 seconds. This particular check has 5 *timing dominators* and no narrowing was performed on 3 of them by the original method. Spatial correlation may present a processing overhead when applied by default, however, it is necessary and proved to be efficient when the constraint system evaluates to non-empty domains and yet no violation is actually possible as in the cases of c2670 and c6288. In these two cases the false paths are caused by reconvergent fan-outs that are *dynamic carriers.* Without applying spatial correlation, case analysis failed to prove the inconsistency because decisions on *dynamic carriers* are done in Phase 3, too far in the decision tree.

| CIRCUIT | OUTPUT | OUTPUT MAX. TOP. | CIRCUIT MAX. TOP. | $\delta$ | BEFORE G.I.T.D. | AFTER G.I.T.D. | SPATIAL CORREL. | C.A. #BTRCK | C.A. RESULT | CPU (s) |
|---|---|---|---|---|---|---|---|---|---|---|
| c17 | 23GAT_10 | 50 | 50 | $50^E$ | P | P | P | 0 | V | 0.05 |
| c432 | 432GAT_195 | 190 | 190 | $190^E$ | P | P | P | 1 | V | 18.82 |
| c499 | OD31_211 | 250 | 250 | $250^E$ | P | P | P | 5 | V | 7.10 |
| c880 | 880GAT_440 | 200 | 200 | $200^E$ | P | P | P | 0 | V | 3.06 |
| c1355 | 1355GAT_558 | 270 | 270 | $270^E$ | P | P | P | 1 | V | 8.17 |
| c1908 | 69_908 | 320 | 340 | 311 | N | - | - | - | - | 0.07 |
| c1908 | 72_909 | 320 | 340 | 311 | N | - | - | - | - | 0.07 |
| c1908 | 57_912 | 340 | 340 | 201 | P | N | - | - | - | 0.76 |
| c1908 | 72_909 | 320 | 340 | $310^E$ | P | P | P | 5 | V | 11.58 |
| c2670 | 225_1424 | 250 | 250 | 241 | P | P | N | 0 | N | 3.67 |
| c2670 | 225_1424 | 250 | 250 | $240^E$ | P | P | P | 7 | V | 17.07 |
| c3540 | 405_1717 | 410 | 410 | 391 | P | N | - | - | - | 2.86 |
| c3540 | 402_1718 | 410 | 410 | 391 | P | N | - | - | - | 2.26 |
| c3540 | 402_1717 | 410 | 410 | $390^E$ | P | P | P | 3 | V | 56.00 |
| c5315 | 658_2483 | 460 | 460 | 451 | N | - | - | - | - | 0.78 |
| c5315 | 690_2484 | 460 | 460 | 451 | N | - | - | - | - | 0.78 |
| c5315 | 658_2484 | 460 | 460 | $450^E$ | P | P | P | 16 | V | 21.97 |
| c6288 | 6288GAT_2447 | 1230 | 1230 | 1221 | P | P | N | 0 | N | 56.36 |
| c6288 | 6288GAT_2447 | 1230 | 1230 | $1220^U$ | P | P | P | A | A | A |
| c7552 | 338_3716 | 380 | 380 | 371 | N | - | - | - | - | 0.34 |
| c7552 | 399_3717 | 380 | 380 | 371 | N | - | - | - | - | 0.38 |
| c7552 | 399_3717 | 380 | 380 | $370^E$ | P | P | P | 1 | V | 8.34 |

***Table IV***
*Results on the ISCAS'85 benchmark suite*

**Legend for Tables IV, V, and VI:** (G.I.T.D. stands for Global Implications on Timing Dominators.)
P: Possible violation of the timing-check constraint.
N: No violation of the timing-check constraint is possible.
V: Test vector found.
- (dash): Procedure not used (was not necessary).
A: Abandoned due to excessive number of backtracks.
($^E$): Value represents exact floating-mode delay. In this case timing checks for all outputs having topological delay greater than or equal to the value are included in Table IV.
($^U$): Value represents upper bound on the maximal floating-mode delay.

Apart from c6288, case analysis heuristics proved to be extremely efficient considering the very low number of backtracks.

| CIRCUIT | #INPUTS | #OUTPUTS | #GATES | #NOR | CIRCUIT MAX. TOP. | FLOATING DELAY | C.A. #BTRCK | TOTAL CPU (s) |
|---|---|---|---|---|---|---|---|---|
| c17 | 5 | 2 | 26 | 6 | 50 | $50^E$ | 0 | 0.05 |
| c432 | 36 | 7 | 386 | 129 | 160 | $190^E$ | 1 | 18.82 |
| c499 | 41 | 32 | 1126 | 370 | 250 | $250^E$ | 5 | 7.10 |
| c880 | 60 | 26 | 744 | 244 | 200 | $200^E$ | 0 | 3.06 |
| c1355 | 41 | 32 | 1206 | 474 | 270 | $270^E$ | 1 | 8.17 |
| c1908 | 33 | 25 | 1216 | 426 | 340 | $310^E$ | 5 | 12.48 |
| c2670 | 233 | 140 | 1946 | 584 | 250 | $240^E$ | 7 | 20.74 |
| c3540 | 50 | 22 | 2134 | 840 | 410 | $390^E$ | 3 | 61.12 |
| c5315 | 178 | 123 | 3718 | 1351 | 460 | $450^E$ | 16 | 23.53 |
| c6288 | 32 | 32 | 4800 | 2352 | 1230 | $1220^U$ | 0 | 56.36 |
| c7552 | 207 | 108 | 5806 | 2023 | 380 | $370^E$ | 1 | 9.06 |

***Table V***
*Summary of the results on the ISCAS'85 NOR-GATES benchmark suite*

The columns of Table V contain, from left to right, the following information: 1) the circuit name, 2) the number of circuit inputs, 3) the number of circuit outputs, 4) the number of gates, 5) the number of NOR gates the circuit has, 6) the circuit topological delay, 7) the circuit floating delay as calculated by the waveform narrowing method, 8) the number of backtracks of case analysis that was needed to find a test vector, 9) the total CPU time for the timing checks that were necessary to prove the circuit timing property (floating delay). Note that the circuits have only NOR gates, buffers and inverters. For instance, c6288 has a total of 4800 gates, 2352 of them are NOR gates, the remaining gates (2448) are buffers and inverters.

Compared to other methods, waveform narrowing proved to be extremely efficient on the traditionally difficult example of c1908. For instance, the method of [85] took 12140 seconds on a 38 MIPS workstation. This represents 3844 seconds on a 120 MIPS, about 300 times slower than our method. For the same circuit, the method of [93] took 3675 second on a 10 MIPS workstation, still about 24 times slower than our method when scaled to 120 MIPS.

| CIRCUIT | CIRCUIT MAX. TOP. | FLOATING DELAY | TOTAL CPU (s) |
|---|---|---|---|
| c1908 | 340 | $310^U$ | 0.86 |
| c2670 | 250 | $240^U$ | 3.67 |
| c3540 | 410 | $390^U$ | 5.12 |
| c5315 | 460 | $450^U$ | 1.56 |
| c6288 | 1230 | $1220^U$ | 56.36 |
| c7552 | 380 | $370^U$ | 0.72 |

***Table VI***
*False path elimination summary*

After all, comparing execution time for exact results between different methods does not tell the whole story. Let's not forget that the calculation of combinational circuit delays is an NP-Hard problem, any exact method has an exponential time complexity in the worst case. Therefore, all methods rely on heuristics biased toward resolving efficiently certain types of circuit topologies.

The important properties a timing verification method should have are execution time predictability, reasonable memory requirements, and implicit false path elimination. Exact methods are not necessary in most cases. In fact, the electronic design automation (EDA) industry is busy building timing verifiers that can be integrated in the design flow easily, and little emphasis is given to false path elimination. The customers simply cannot afford to wait unpredictable amounts of time before obtaining the results, not to mention that the memory requirement of the method is critical for large industrial designs. In fact, modern EDA physical design tools, e.g., IBM's ChipBench, have design and verification tools integrated in one user interface. The designer can modify the physical design and get timing feedback right after, all design data remain in core along with the data structure needed for timing verification. These facts put the waveform narrowing method at a great advantage.

Table VI summarizes the false path implicit elimination of our method, the case analysis execution is stripped out. The fact of the matter is that only 10% of the execution time is spent on eliminating the false paths and determining a tight delay upper bound that corresponds to the exact circuit delay. The remaining 90% is simply spent on proving that the upper bound we found is in fact the exact delay. Beside the case analysis, our method has a virtually $n \times \log(n)$ time complexity ($n$ is the circuit size) when the *timing dominators* procedure is used, and a virtually quadratic time complexity when the spatial correlation procedure is used. Moreover, the memory requirement of our method is minimal, the data structure is basically a circuit graph representation, and a domain stack for each circuit net. The domain stack can be limited to a maximum of three domain instances. The program implementation required on the average about 400 bytes for each circuit gate.

## 3.10 Conclusions

In this chapter, we established the mathematical foundations of the waveform narrowing method, and defined elaborate constraints for basic primitives such as symmetrical gates, AND, XOR, NOT, BUFFER, DELAY, etc. We showed how the novel global implications on timing dominators efficiently reduced the pessimism of the method. Further refinements were achieved by enforcing spatial correlation on reconvergent stems and by the case analysis procedure that aims at finding exact results.

At this point, we have a method that proved to be very efficient on the standard ISCAS'85 benchmark suite. However, it is very desirable to assess its effectiveness on real world industrial circuits. The software implementation, part of the Power and Timing Verification Project at the Université de Montréal, is suitable to run standard benchmark circuits, but it does not have the capabilities needed to run industrial designs. It needs adequate capability for cell library modeling, standard delay back-annotation, complex clocking schemes, etc. In order to test the method on industrial circuits, we simply had to rewrite the timing verifier from scratch. Chapter 4 explains the work we did to bring the waveform narrowing method closer to industrial use.

# CHAPTER IV ADVANCED MODELING

The previous Chapter presented a comprehensive constraint-based framework for timing analysis that can be applied to circuits composed of symmetrical logic gates having simple timing properties. When the timing environment consists of a single-phase clock signal, it is trivial to reduce the problem of verifying setup and hold constraints of the flip-flops to a delay calculation problem of a combinational circuit. However, when faced with real world industrial designs, complex pragmatic aspects need to be addressed, namely:

**Test Cases:** VLSI chips have multiple modes of operation, e.g., test mode, scan mode, functional mode, etc. Therefore, multiple test cases are presented to the timing verifier; each one consists of a set of constants to be applied to primary inputs, and a set of harmonically related clocks. The constants configure the chip in a specific mode of operation, causing the clocks to be routed to specific sub-sets of the flip-flops. While defining constants is trivial (instance pin = logic value), defining harmonically related clocks require a concise and elaborate syntax.

**Standard Delay Annotation:** industrial component delays are specified as triplets (minimum, typical, maximum). Since the typical delay value does not occupy a fixed relative position in the interval [minimum, maximum] over all circuit components, abstracting the delays to their [minimum, maximum] intervals leads to errors.

**Cell Libraries:** the building blocks of an industrial design are components chosen from a technology cell library that usually contains non-symmetrical and tristate logic gates, e.g., multiplexers, tristate buffers, etc. Moreover, cell component delays are specified as selective i/o path delays for each type of transition at an output, caused by an event at a specific input. For example, consider a tristate buffer having an output Y and two inputs A and G, operating according to the following truth table:

| A | G | Y |
|---|---|---|
| 0 | 0 | Z (high impedance state) |
| 1 | 0 | Z |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

The output Y can hold three values (0, 1, and Z). Therefore, six transition types are possible at Y: 01, 10, 0Z, Z0, 1Z, and Z1; i/o path delays are specified as follows:

| | |
|---|---|
| A to Y for 01 | (event propagation delay from A to Y, when the resulting event at Y is a rising transition) |
| A to Y for 10 | |
| G to Y for 0Z | |
| G to Y for Z0 | |
| G to Y for 1Z | |
| G to Y for Z1. | |

Consequently, an extended set of constraint operator primitives is needed to allow the modeling of cell library components.

**Combinational Loops:** although discouraged by modern design methodologies, industrial designs may still contain combinational loops that the timing verifier needs to identify and handle properly.

Section 4.1 presents the syntax adopted to specify harmonically related clocks; Section 4.2 addresses delay modeling and component delay correlation using triplets (min, typ, max); Section 4.3 defines an extended set of constraint operator primitives necessary to model cell library components; Section 4.4 presents the cell library modeling process; Section 4.5 illustrates how combinational loops are handled; Section 4.6 presents a high-level architecture of the resulting timing verifier; Section 4.7 presents the results of verifying a real world industrial design; and Section 4.8 concludes the chapter.

## 4.1 Clock Definition Formalism

An important piece of information presented to timing verifiers in test cases is the definition of clock domains, harmonically related clocks, and the functional semantics of the circuit. It is imperative that the timing verifier distinguishes which clock edge is sampling the computed new circuit state, and which one is injecting the old state to be used by the combinational circuit to compute the new one. This section defines simple formalism but powerful enough to express arbitrary complex clocking schemes.

Test cases are specified using a simple regular grammar syntax. We describe the part that defines harmonically related clocks, and edge selection clauses that specify which clock edge to use at each flip-flop when performing setup verification. Note that it is irrelevant to specify edges for a synchronizing latch; its constraint selects the pulse that propagates data without violating its setup/hold constraint. The clock definitions are based on absolute time scale.

**Notation:**

`::=` denotes the relation "is a";
Balanced parentheses enclose syntax elements;
`+  ::=` one or more occurrences of the preceding syntax element;
`{}::=` one of the enclosed, comma separated, syntax elements;
C++ data types and comments are used inside definitions.

A set of harmonically related clocks is specified as:

```
(CLOCKS
  (BASE_PERIOD float_base_period)
  (CLOCK clock_name
    (DRIVER pin_driver)
    (MULT unsigned_mult)
    (DIV unsigned_div)
    {
      {(RISE float_ideal_time [float,float])
        (FALL float_ideal_time [float,float])}+,
      {(FALL float_ideal_time [float,float])
        (RISE float_ideal_time [float,float])}+
    }
  )+
)
```

Each clock has a name, a driver, a clock period **base_period * mult / div**, and an even number of alternating transitions, each defined by an **ideal_time** and an interval of uncertainty for the edge occurrence time. **ideal_time** is a real number reflecting simultaneity and edge ordering.

**Example:** the clocking scheme of Fig. 102 is represented as:

```
(CLOCKS
 (BASE_PERIOD 10.0)
 (CLOCK clock_A
  (DRIVER driver_A)
  (MULT 3)(DIV 1)
  (RISE 0 [-0.1,0.1])
  (FALL 10 [11.1,11.2])
 )
 (CLOCK clock_B
  (DRIVER driver_B)
  (MULT 4)(DIV 1)
  (RISE 20 [19,21])
  (FALL 30 [30.5,32])
 )
)
```

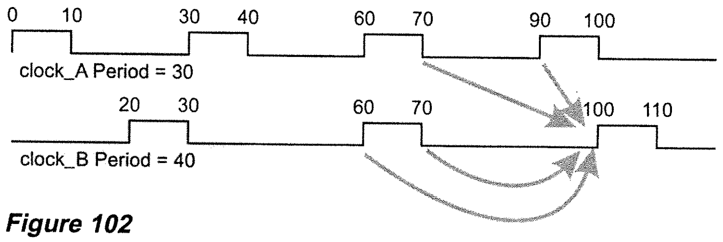


***Figure 102***
*Clocking scheme example.*

Examining the **MULT/DIV** entries, the timing verifier determines that the clocking scheme period is 120 (4 **clock_A** cycles, or 3 **clock_B** cycles). If no edge selection is provided in the test case, it generates default setup edge selection for all transitions in one clocking scheme period. For instance, when the new state is sampled by edge 100 (see Fig. 102), the timing verifier selects the closest relevant edge before it for old state data injection, 90, 70, or 60, depending on which clock is driving the flip-flop, and which edge polarity the flip-flop is sensitive to. An edge selection is represented by **ideal_time [translation]** where **ideal_time** is one defined in the clock definition;

**translation** is an integer representing the number of clock periods the edge is translated by. The following is a part of the generated default setup edge selection clause.

```
(SETUP_EDGES
 (DEST clock_A 0[0]
  // data destination flip-flop clocked by
  // clock_A edge 0 in cycle 0
  (SOURCE clock_A
   (EDGE 0[-1])(EDGE 10[-1]))
   // if data source flip-flop is clocked
   // by clock_A, select edge 0
   // or edge 10 in cycle -1, whichever
   // is relevant
  (SOURCE clock_B
   (EDGE 20[-1])(EDGE 30[-1]))
 )
 ...
)
```

If the user provides edge selection clauses, the timing verifier uses them and generates no defaults. Correct edge selection depends on correct comparison and translation of **ideal_time**, which is represented as a double precision floating point number, a representation that may cause slight errors as a result of conversion and arithmetic operations. Consequently, software compensation measures were taken (fuzzy comparison).

## 4.2 Modeling Delays

Gate delays are defined and annotated as triplets (*min*, *typ*, *max*), representing delays for predefined operating conditions and manufacturing parameters (supply voltage, junction temperature, process). It is generally accepted that, for example, when the delay of a gate $g_1$ is assigned its *typical* value, the delay of a gate $g_2$ is assigned a narrow interval around its own *typical* value. A similar scheme is used for *min*. and *max*. delays. *Typical* delays do not occupy a fixed relative position in the interval [*min*, *max*] for all gates; therefore, the simplistic correlation approach used in the sequential example in section 3.4 is incorrect. In this section we address this problem by introducing the concept of normalized delays.

### 4.2.1 Normalized Delay

*Normalized delay* is a real value in the interval [-1, +1]. For a delay annotation d = (*min*, *typ*, *max*), we define a mapping $DtoN_d : [min, max] \rightarrow [-1, 1]$ such that $DtoN_d([min,max])$ is the two line segments (*min*, -1) to (*typ*,0) and (*typ*,0) to (*max*,1), illustrated in Fig. 103. We define also the function $NtoD_d$ as the inverse of $DtoN_d$.

The *delay correlation degree* $\Delta$ is a real number in the interval [0,1]: 0 for no correlation, 1 for 100% correlation. The *normalized deviation* for a delay correlation degree $\Delta$ is $J = 2(1 - \Delta)$, a real number in [0,2]. J represents the maximum deviation between any two correlated delays at the normalized scale. For a specific normalized delay value N and deviation J, a gate interval delay is determined as $NtoD_g([N - J, N + J] \cap [-1, 1])$, see Fig. 104 (a). Fig. 104 (b) shows the gate delay bounds, function of the normalized delay N and J. It is a polygon ABCDEFGH. Path delays are sums of gate delays; therefore, they are bounded by a similar polygon, and when setup constraint verification is based on topological analysis (no false path elimination), it is sufficient to consider four cases for N: -1 + J, -J, J, and 1 - J to get a conservative figure for the slack (timing margin before

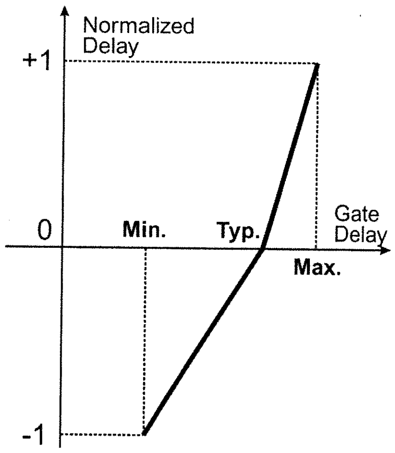
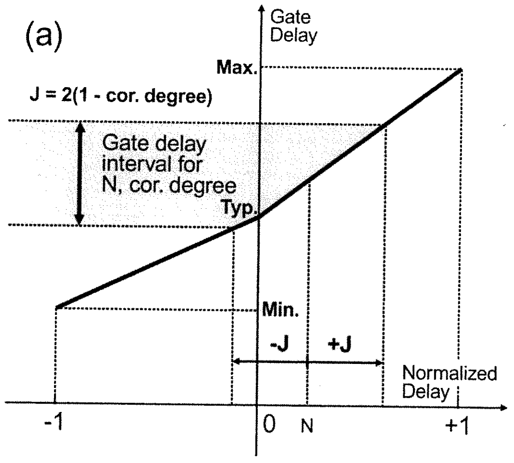


***Figure 103***
*Normalized delay.*



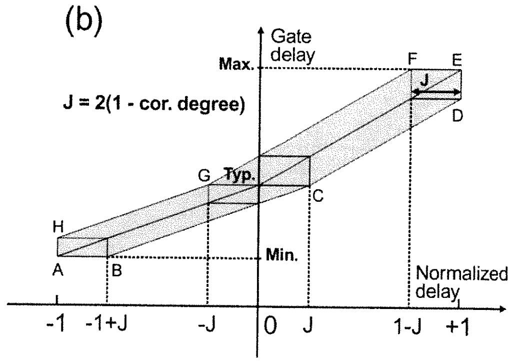


***Figure 104***
*Gate interval delay, function of J and N.*

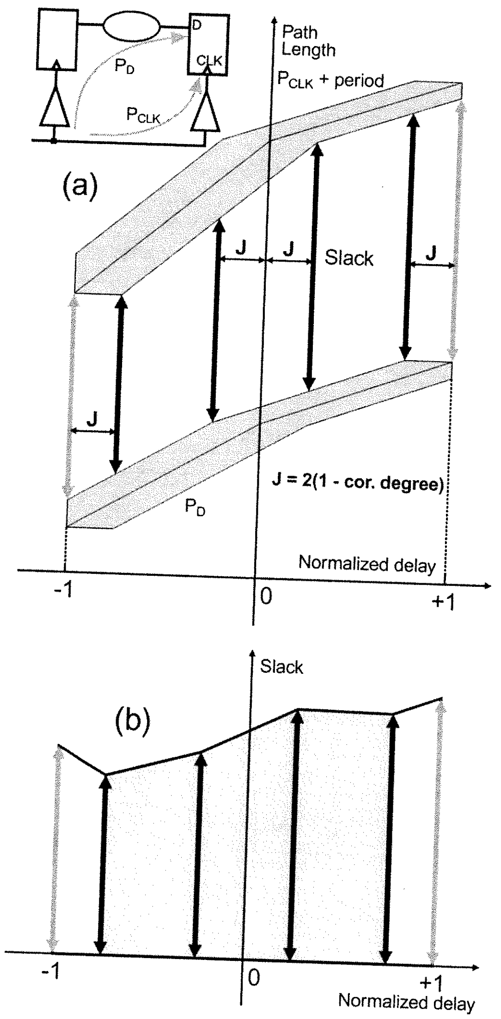


***Figure 105***
*Four point slack computation.*

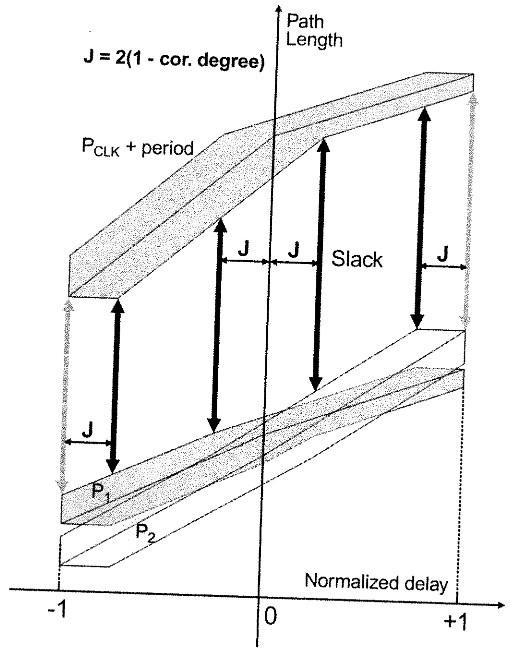


***Figure 106***
*Slack for multiple data paths.*

setup violation happens at a flip-flop). Fig. 105(a) illustrates the case for a single data path: the minimum clock path delay and the maximum data path delay change linearly between the four evaluation points, therefore the slack, shown in Fig. 105(b), changes also linearly (Note that N = -J or N = J does not represent the worst case). In the case of multiple data paths, see Fig. 106 for example, the line segment joining the highest end points of a set of line segments between -J and +J is always above all of them. Consequently, considering that the slack changes linearly between -J and J is conservative. In general, it is not clear whether this property holds when the timing verifier eliminates false paths and the circuit contains delay dependent false paths.

### 4.2.2 Delay Correlation Networks

Consider a set of delays $\{d_1, d_2, ..., d_p\}$, correlated by a degree $\Delta$. The correlation relation is implemented using a single normalized delay ND, and a correlation constraint for each delay, operating on the delay and ND (see Fig. 107). For a delay $d_k$ and its associated $DtoN_k$ and $NtoD_k$ The constraint is defined using $J = 2(1 - \Delta)$ as follows:

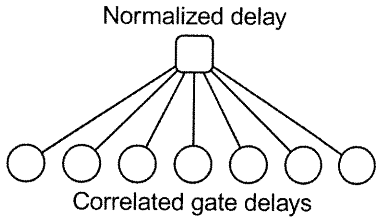


***Figure 107***
*Simple delay correlation network.*

- $ND' = ND \cap (DtoN_k(d_k) + [-J , J])$
- $d_k' = d_k \cap NtoD_k(ND)$

In other words,

- when a delay $d_k$ changes as a result of applying constraints, ND is narrowed to $(DtoN_k(d_k)+ [-J,J])$;
- when ND changes, all correlated delays $d_m$ are narrowed to $NtoD_m(ND)$.

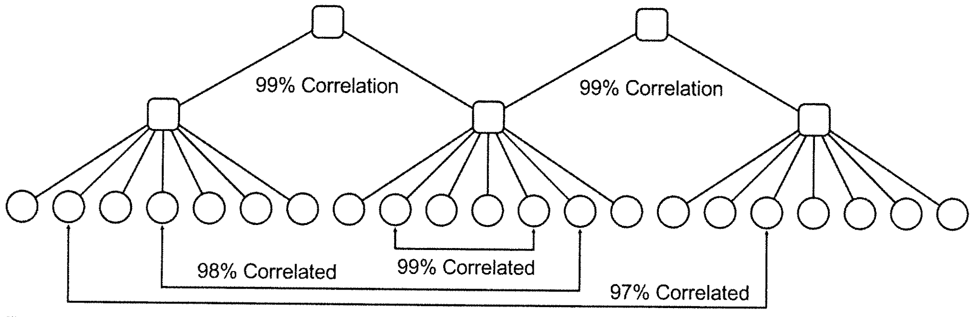


***Figure 108***
*One-dimensional position dependent delay correlation network.*

A normalized delay is no different from a delay, and it is possible to use multiple correlated NDs. Fig. 108 and 109 illustrate position dependent correlated delays using two level NDs. Arbitrary complex correlation schemes can be represented by correlation networks. Apart from supporting position dependent delay correlation, our timing verifier supports correlation between rising delays; between falling delays; and between rising and falling delays, as they may be inter-correlated with different strengths. However, more research is needed in process characterization to take full advantage from the offered flexibility. In general, delays associated with components belonging to different manufacturing steps are correlated with lesser strengths than those belonging to the same one.

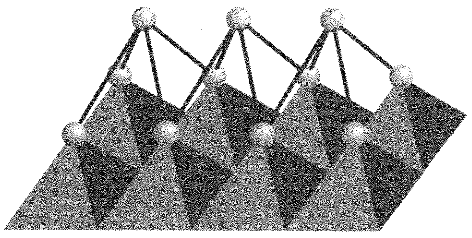


***Figure 109***
*Two-dimensional position dependent delay correlation network.*

## 4.3 Modeling Building Blocks

Modeling industrial cell libraries, the building blocks of industrial designs, requires many types of primitives and domains. Cell delays are back annotated in the Standard Delay Format (SDF) as multiple triplets (min, typ, max), one for each combination of input terminal and event type at the output. For example, consider a two input AND gate for which A and B are the inputs, and Y is the output. Four path delay triplets are defined for such a gate:

- A delay to be used for an event at A that causes a 0-to-1 transition at Y
- A delay to be used for an event at A that causes a 1-to-0 transition at Y
- A delay to be used for an event at B that causes a 0-to-1 transition at Y
- A delay to be used for an event at B that causes a 1-to-0 transition at Y

Moreover, industrial designs contain tristate gates that can have high impedance states (Z) at their outputs. Therefore, the data domain should support this new state in addition to the usual binary values 0 and 1. This leads to heterogeneous domain types in constraint models. For instance, the domain type at a tristate buffer output contains three intervals (for 0, 1, and Z), whereas a domain at an input contains only two. Also, delays for tristate gates are defined for six types of transitions: 0 to1, 1 to 0, 0 to Z, Z to 1, 1 to Z, and Z to 0. The remainder of this section summarizes the domain types and primitives we implemented as building blocks for developing cell libraries.

### 4.3.1 Domains

The following domains are implemented:

- **TVintervalDelayDomain**
  Delay domain used when a single interval is needed for gate or interconnect delays.
- **TVrfDelayDomain**
  Delay domain for 0-to-1 and 1-to-0 transitions (two intervals).
- **TVonOffDelayDomain**
  Delay domain for 0-to-Z, Z-to-1, 1-to-Z, Z-to-0 transitions (four intervals). This is suitable for delays from tri-state control inputs to switch the output between normal binary states and high impedance.
- **TVcorrelationDomain**
  Normalized delay domain.
- **TVrfCorrelationDomain**
  Normalized delay suitable to correlating rising and falling delays separately.
- **TVclockDomain**
  Clock domain. It contains a number of transitions depending on how the clock is defined in the clocking scheme. Each transition is defined as a polarity flag (RISE, or FALL), an ideal time, and an interval of uncertainty for the clock edge.
- **TVsetupDomain**
  Abstract signal domain, it contains two intervals, $w_0$ and $w_1$ representing $\left(0\Big|_{w_0.min}^{w_0.max}, 1\Big|_{w_1.min}^{w_1.max}\right)$.

- **TVsetupDomainZ**

  Abstract signal domain that accounts for high impedance state. It contains three intervals, $w_0$, $w_1$, and $w_z$ representing $\left(0\big|_{w_0.min}^{w_0.max}, 1\big|_{w_1.min}^{w_1.max}, z\big|_{w_z.min}^{w_z.max}\right)$.

### 4.3.2 Primitive Constraints

The following logic primitives are used internally by the timing verifier, and used to describe cell library models. Appendix B (B.1) lists the primitives necessary for cell modeling, along with their terminal and delay names.

- **interconnectDelay**

  A delay primitive for interconnect delays. It is distinguished from gate delays, so that the timing verifier can correlate its delay correctly if the user chooses to correlate interconnect delays separately.

- **nonInvertingDelay**

  A delay primitive annotated as rising (R) and falling (F) delays. It applies R to $w_1$ and F to $w_0$. Used at a cell input to model positive path delay (rising transition at input causes rising transition at output, e.g., input of an AND gate).

- **invertingDelay**

  A delay primitive annotated as rising (R) and falling (F) delays. It applies F to $w_1$ and R to $w_0$. Used at a cell input to model negative path delay (rising transition at input causes falling transition at output, e.g., input of a NAND gate).

- **unknownDelay**

  A delay primitive annotated as rising (R) and falling (F) delays. It applies R $\cup$ F to $w_1$ and $w_0$. Used at a cell input to model unknown path delay (rising transition at input may cause a rising or falling transition at output, e.g., input of a XOR gate).

- **fs0zRz0FDelay, fs0zFz0RDelay, fs0zUz0UDelay, fs1zRz1FDelay, fs1zFz1RDelay, fs1zUz1UDelay, onRoffFDelay, onFoffRDelay, unknownZDelay, tristateDelay**

These delay primitives handle tristate gate delays, they differ by how annotated delays are applied.

- **buf, not, and2, and3, ..., or2, or3, ..., nand2, nand3, ..., nor2, nor3, ..., xor2, xor3, ..., xnor2, xnor3, ...**
  Delayless symmetrical gates.

- **mux2**
  Multiplexer primitive. The multiplexer constraint is discussed in detail later.

- **nMux2**
  Inverted output multiplexer primitive.

- **deviceInput, deviceInOut, deviceOutput**
  Primitives for device inputs, outputs, and input/outputs.

- **vss, vdd, highZ**
  Primitives that provide constants (vss = 0, vdd = 1, highZ = Z).

- **passive, passiveZ**
  To hide circuit nodes that have no timing assumptions, e.g., connections that are driven exclusively by clocks not harmonically related to the current clock, the timing verifier uses the passive primitives. Passive and passiveZ always hold the constant domains $(0|_{-\infty}^{-\infty}, 1|_{-\infty}^{-\infty})$ and $(0|_{-\infty}^{-\infty}, 1|_{-\infty}^{-\infty}, z|_{-\infty}^{-\infty})$, respectively. When the setup constraint of a D flip-flop is being verified, the timing verifier traces the logic cone that drives D and breaks the connections that have no timing assumptions. The broken connections are then routed to passive or passiveZ depending on whether high impedance state is needed. Passive primitives disallow narrowing on the domain they hold to prevent timing optimism.

- **bufif0, bufif1, notif0, notif1**
  Tristate primitives. Their outputs hold tristate domains (**TVsetupDomainZ**), whereas their inputs expect binary domains (**TVsetupDomain**).

- **pullup, pullupDeviceInOut, pulldown, pulldownDeviceInOut, pullupdown**
  Primitives needed to accommodate circuit nets that are driven by multiple tristate gates. In such cases, all net drivers are routed to the inputs of the pull primitive which becomes the driver of the net sinks. Pull primitives expect tristate domains at inputs, they hold a binary domain at the output.

- **latchHQ, latchLQ, latchHQZ, latchLQZ**
  Transparent latch primitives. They differ by whether the output is inverted or not, and by the clock sensitivity. The latch constraint is delay dependent. Actually, latches are delay annotated. The latch constraint determines the earliest clock window that makes the latch transparent, respecting its setup timing requirement. For example, consider the high level sensitive non-inverting latch (latchHQ) as defined in Section 1.3.2 (see Fig. 8): D is the data input; C is the clock defined as a rising edge, falling edge, and a period; Q is the output; C-to-Q is the time delay before Q starts to follow D, after C rises to 1; D-to-Q is the events propagation delay from D to Q, when C = 1; S is the setup time constraint. The latch constraint operator operates on {C, D, Q, C-to-Q, D-to-Q, S}, and it is defined as follows:

**Effect on Q.$w_0$ and D.$w_0$:**

```
    R = uncertainty interval of the rising clock edge;
    F = uncertainty interval of the falling clock edge;
    period = clock period;
    SafeCycle = min $\{n \in Z \mid D.w_0.max < F.min + period \times n - S.max\}$;
The actual falling edge interval for which the setup constraint is satisfied is:
    AF = F + period $\times$ SafeCycle;
    AR = uncertainty interval of the rising edge that occurs just before the edge of AF;
    if(D.$w_0$.$lmin$ > AR.max){
        // latch is transparent for the last-transition-interval of D.$w_0$.
        // the latch behaves like a buffer
        Q'.$w_0$ = Q.$w_0$ $\cap$ (D.$w_0$ + D-to-Q)
        D'.$w_0$ = D.$w_0$ $\cap$ (Q.$w_0$ − D-to-Q)
    }else{
```

$max0 = \max\{AR.max + \text{C-to-Q}, D.w_0.max + \text{D-to-Q}\};$

$Q'.w_0.max = Q.w_0.max \cap 0|_{-\infty}^{max0}$

}

The effect on $Q.w_1$ and $D.w_1$ is symmetrical.

- **dffHQ, dffLQ**

  Primitives that inject logic values into the combinational circuit. They are used to model the output half of a D flip-flop. dffHQ is triggered by a rising clock edge, whereas dffLQ is triggered by a falling one. The dff relational constraint is very simple. Let *C* be the clock edge interval, and *Q* the abstract signal at the output:

$$\begin{aligned} Q'.w_0 &= Q.w_0 \cap 0|_{-\infty}^{C.max} \\ Q'.w_1 &= Q.w_1 \cap 1|_{-\infty}^{C.max} \\ C' &= C \cap [min(Q.w_0.lmin, Q.w_1.lmin), +\infty] \end{aligned}$$

- **HsetupHold, LsetupHold**

  Flip-flop setup constraints. This models the input half of a D flip-flop. HsetupHold is sensitive to the rising clock edge, while LsetupHold is sensitive to the falling one. The relational constraint is very simple. Let *C* be the clock edge interval, and *D* the abstract signal to check for possible overlap with the clock:

$$\begin{aligned} D'.w_0 &= D.w_0 \cap 0|_{C.min}^{+\infty} \\ D'.w_1 &= D.w_1 \cap 0|_{C.min}^{+\infty} \\ C' &= C \cap [-\infty, max(Q.w_0.max, Q.w_1.max)] \end{aligned}$$

- **dummy**

  used as a place holder with no functionality.

- **reset, resetZ, preset, presetZ**

  Primitives used to model active low/high reset/preset lines for flip-flops and latches.

The following primitives handle delay values and component delay correlation. They are used internally by the timing verifier.

- **TVconstantDelayHolder**
  Holds a delay domain that is not bound by component delay correlation.

- **TVdelayCorrelation**
  Single interval *normalized delay* primitive, used to bind a set of delay values by a correlation degree.

- **TVrfDelayCorrelation**
  Dual interval *normalized delay* primitive, used to correlate rising and falling delays separately.

- **TVintervalDelayValue**
  Holds a single interval delay.

- **TVrfDelayValue**
  Holds two interval delays, Rising and Falling.

- **TVooDelayValue**
  Holds four interval delays, 0-to-Z, Z-to-1, 1-to-Z, Z-to-0.

- **TVshDelayValue**
  Holds setup and hold constraints (annotated like delays) for flip-flops and latches.

- **TVnegIntervalDelayValue, TVnegRFDelayValue, TVnegOODelayValue, TVnegSHDelayValue**
  These primitives hold delay values similar to their "positive" counterparts. However, the delay correlation mechanism is reversed. For example, when delays in "positive" delay primitives get their *max* values, delays in these primitives get their *min* values. These primitives may be useful to model delays in mixed technologies where, for example, higher temperature can cause some components to be slower, and some others to be faster.

## Multiplexer Primitive

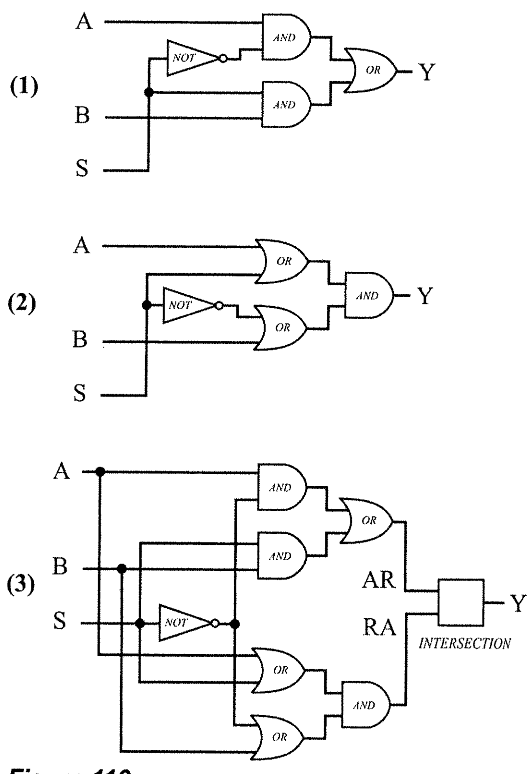


***Figure 110***
*Two-to-one multiplexer.*
*(1) AND - OR logic model.*
*(2) OR-AND logic model.*
*(3) Timing model.*

Multiplexers have complex non-symmetrical behavior. Two-to-one multiplexers have three inputs: A, B, and S, and one output Y. When S = 0, Y = A, and when S = 1, Y = B. Special attention is given to the multiplexer primitive as it is very pessimistic to use its equivalent logic model as a timing one.

The circuits of Fig. 110(1) and (2) are two logic models equivalent to the multiplexer logic function, nevertheless, they have different *floating-mode* timing properties. Consider the case when A and B are both stable at 0. The output of the AND-OR circuit (1) is stable at 0 regardless of S, whereas, the constraint model of the OR-AND model (2) fail to get to such a conclusion. When A and B are stable at 1, however, the OR-AND circuit behaves correctly, while the AND-OR model fail. The fact of the matter is that the paths from S to Y are false when A and B are stable at the same logic value, a situation the *floating-mode* delay model fail to uncover.

Since the AND-OR and the OR-AND have complementary behaviors, one succeeds when the other fail, we opted to use the intersection of both cases. The hard coded multiplexer constraint model is depicted graphically in Fig. 110(3), it corresponds to the *sequences of vectors* delay model. The *INTERSECTION* constraint is defined as follows:

INTERSECTION(RA, AR, Y) = (RA', AR', Y') such that

$$Y' = Y \cap AR \cap RA,$$

$$RA' = Y \cap AR \cap RA,$$

$$AR' = Y \cap AR \cap RA.$$

## 4.4 Cell Library Modeling

The modeling of combinational cells is straightforward. Interconnect delays are placed at the cell inputs, each interconnect delay is followed by path delays, one for each reachable cell output. The cell logic structure is mapped to the predefined primitives counterparts. The exception is when the cell contains logic structures that drive multiple outputs. In such cases, the common structure is duplicated in order to make the path delays independent, as illustrated in the example of Fig. 111. The path delay types are chosen according to the path polarity from the cell input to the cell output in question. For instance, path delays of an AND gate are of type nonInvertingDelay, because a rising transition at the output is only caused by rising ones at inputs. Inverting and non-inverting path polarities cause no accuracy problem for the cell timing model. However, unknown path polarity is pessimistic, because the worst-case delay is applied at the cell input regardless of the transition type at the output. In fact, the delay element is not aware of the state at the cell output. To remove this pessimism, the timing verifier adds a cell-aware constraint. It consists of the following procedure (Note that the rising/falling delay from input A to output Y is the delay separating an event at A and the resulting rising/falling transition at Y):

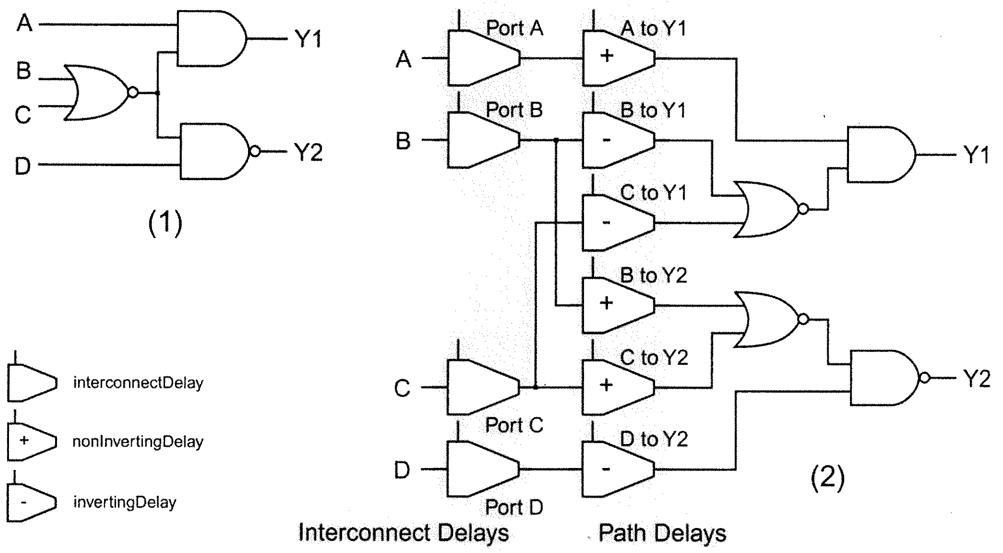


***Figure 111***
*Combinational cell modeling example. (1) Cell logic function. (2) Cell constraint model.*

1) Save the domains of the cell (internal and terminals);

2) Apply the constraints using the rising path delays;

3) Remove from the output the waveforms stabilizing at 0;

4) Exchange the domains with the saved ones;

5) Apply the constraints using the falling path delays;

6) Remove from the output waveforms stabilizing at 1;

7) Merge the domains with the saved ones.

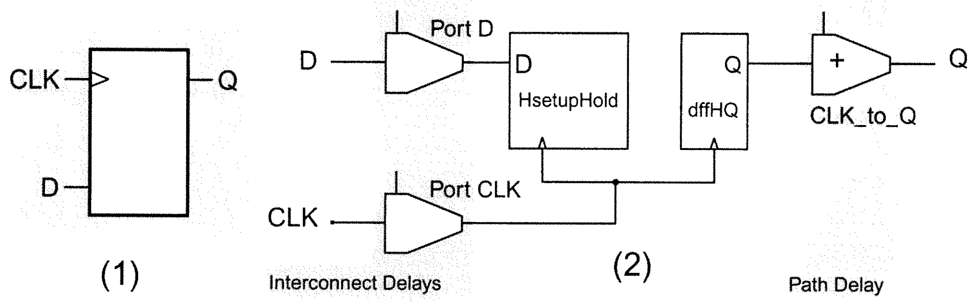


***Figure 112***
*Flip-Flop cell modeling example. (1) Cell logic function. (2) Cell constraint model.*

We automated the combinational cell constraint model generation to prevent human mistakes and omissions from corrupting the timing verification results. The sequential cells are hand crafted. Fig. 112 shows how a flip-flop is modeled. Interconnect delays are put at the inputs, a timing check is put between the clock and the data input signal, dffHQ primitive is used to drive the clock-to-Q path delay that is placed at the output this time. The setup constraint value is annotated to HsetupHold primitive. Appendix B (B.3) contains the real definition of a similar flip-flop.

## 4.5 Handling of Combinational Loops

Although discouraged by synchronous design methodologies, combinational loops are still used. Therefore, the timing verifier needs to deal with them. Our timing verifier determine all circuit loops and resolves them by imposing additional constraints on loop nodes. An added constraint binds the stabilization time of the node using the worst-case arc (path length) from the loop input lines as shown in the example of Fig. 113:

(a) C and E are part of a combinational loop (in fact E = B);

(b) additional constraint for E: limit E to waveforms stabilizing after *max*, the worst case of A propagated through path $g_1$-$g_2$, and D propagated through path $g_2$;

(c) additional constraint for C: limit C to waveforms stabilizing after *max*, the worst case of D propagated through path $g_2$-$g_1$, and A propagated through path $g_1$.

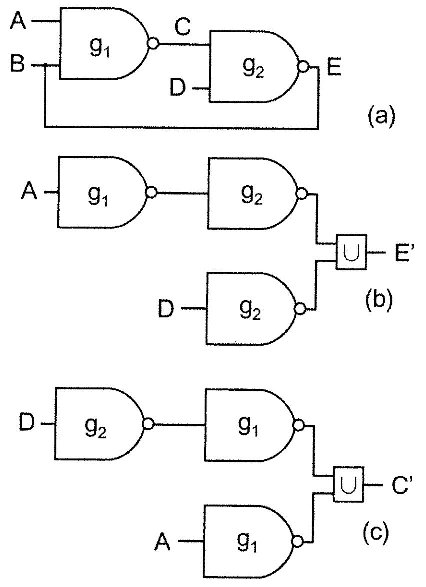


***Figure 113***
*Combinational loop example.*
*(a) C and E are part of a combinational loop.*
*(b) Loop aware constraint for E.*
*(c) Loop aware constraint for C.*

## 4.6 Timing Verifier

The timing verifier is written in C++ (about 60000 lines of code), uses a Verilog compiler provided by Nortel Networks, and is composed of three components (see Fig. 114):

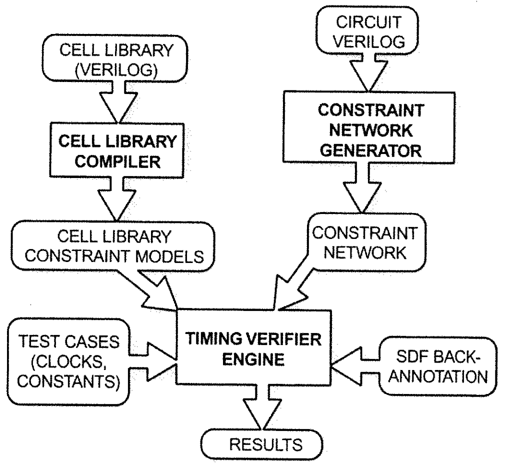


***Figure 114***
*Timing Verifier Architecture*

- Cell Library Compiler, generates models for combinational cells;

- Constraint Network Generator, generates constraint networks (in terms of cell models) from Verilog circuit descriptions;

- Core Waveform Narrowing Timing Verifier, supporting SDF back-annotation.

Input constraints that select particular circuit operation modes or test cases can be presented to the timing verifier as sets of constants to be applied to certain circuit nodes, and as clock domain definitions. The syntax is very simple, the following is a commented example:

```
(STV_TIMING
 (DESIGN "falsepath") // usually the Verilog module name
 (DATE "Wed Jan 26 13:30:35 2000")
 (PROGRAM "")
 (VERSION "")
 (DIVIDER .)  // hierarchy divider, it can be either . or /
 (SYSTEM_TIME_UNIT ns)
 (SYSTEM_TIME_SPAN 100) // this tells the timing verifier to scale delays to
                        // integers so as to accommodate at least 100 ns max values.
 (TEST_CASE "test 1"    // we can define multiple test cases, e.g., "Functional", "Scan Test", ...
  (CONSTANTS
   (DRIVER falsepath.instance1.Y  H)   // apply 1 to falsepath.instance1.Y
   (DRIVER falsepath.instance2.Y  H)
   (DRIVER falsepath.instance3.Y  L)   // apply 0 to falsepath.instance3.Y
  )
  (CLOCKS
   (BASE_PERIOD 5)
   (CLOCK clock
    (DRIVER falsepath.clk)
    (MULT 1)
    (DIV 1)
    (RISE 0 [0,0])
    (FALL 2.5 [2.5,2.5])
   )
  )
  (SETUP_EDGES
   (DEST clock 0[0]
    (SOURCE clock
     (EDGE 0[-1])(EDGE 2.5[-1])
    )
   )
  )
 )
)
```

**The verification is done as follows:**

```
verifyTestCases() {
. load the cell libraries, constraint network, and test cases;
. if SDF file is specified {
. . load and annotate delays;
. } else {
. . use default delays specified in the libraries;
. }
. for each test case do {
. . generate default clocks edge selection clauses if not provided by the user;
. . apply and propagate constants to circuit nodes;
. . strip inactive parts of the circuit;
. . trace and initialize clock trees;
. . determine testable setup checks;
      // an untestable check is one that is driven exclusively by
      // logic that is not controlled by any defined clock.
      // Testable checks may still be controlled partly by logic with
      // unknown timing properties, this part is routed toward passive
      // constraints so that we can verify the part we know about.
. . for each edge selection clause do {
. . . select the checks that are controlled by the edge;
. . . initialize the constraint system;
        // initialization includes combinational loop
        // detection and additions of appropriate loop constraints if any
. . . evaluate the constraint system with no constraint on any check;
. . . while (true) {
. . . . restrict the constraint system to the check having the worst slack;
. . . . refine the check results using pessimism reduction techniques;
. . . . break if we still get the same slack;
. . . }
. . . print results for the edge selection;
. . } // for each edge selection clause
. } // for each test case
} // verifyTestCases()
```

## 4.7 Experimental Results

The timing verifier was tested on two industrial circuits provided by Nortel Networks. The first circuit is a small well-characterized Nortel benchmark used mainly for tool evaluation and debugging purposes. It has 2651 cell instances (TGC1000 Texas Instrument cell library) and 1160 timing checks. The test case has four clock domains and 13 constants. After constants propagation, the clock tree reached 1120 timing checks; all of them are found to be testable. The results were consistent with the results of Mirage, the timing verifier used at Nortel Networks. The execution required 5.8 Mbytes and 0.891 seconds on a PC (P-III, 866 Mhz) detailed as:

Loading libraries: 0.12 second
Loading design: 0.15 second
Loading test cases: 0 second
Loading delays: 0.3 second
Verification: 0.311 second.

This circuit was very useful for debugging our software, it has typical industrial complexity: uses synchronizing latches, flip-flops, tristate gates, etc., it is small enough to trace easily, and it has known timing properties. This small benchmark has no value when it comes to evaluating false path elimination effectiveness. In fact, its logic cones are tiny, and the eliminated false paths did not affect the final timing results. Its value, however, is in validating our clock edge selection algorithms, as well as the basic timing verification algorithms.

The other circuit we checked is a 122 Kgates synchronous design (34 inputs, 131 inouts). The experiment was done on a Sun Ultra 10 workstation with 512 Mbytes of RAM. The test case has 13 constants and 4 harmonically related clock domains (period = 100 ns). After constants propagation, the defined clocks reached 57118 timing checks. 55 checks were determined irrelevant (data is constant, Reset/Preset active, or data not selected for latching in a scannable flip-flop). 57063 timing checks are testable, all checked for setup constraint. Two delay annotation files were available: one has worst-case delays; the other has best-case delays. We checked the design against each annotation. The results are presented in Tables VII and VIII. The tables columns contain, the timing check identifier, followed by slack double columns under **R** (timing slack for rising transitions) and **F** (timing slack for falling transitions). Each execution required 145 Mbytes of RAM, and the following CPU times for file loading across network file system:

Loading libraries: 1.65 seconds
Loading test cases: 0.01 seconds
Loading design (30 Mbytes): 101.33 seconds
Loading delays (50 Mbytes): 243.25 seconds.

| Best Case | No Pessimism Reduction 1.51 minutes | | Spatial Correlation Order 1 0.2 minute | | Spatial Correlation Order 2 0.7 minute | | Spatial Correlation Order 3 0.9 minute | | Spatial Correlation Order 5 4.8 minutes | | Spatial Correlation Order 8 20.79 minutes | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Check** | **R** | **F** | **R** | **F** | **R** | **F** | **R** | **F** | **R** | **F** | **R** | **F** |
| DFF_01 | 36.04 | 35.53 | 39.25* | 39.26* | 39.25 | 39.26 | 39.25 | 39.29* | 39.25 | 39.29 | 39.25 | 39.29 |
| DFF_02 | 41.15 | 40.49 | 41.67* | 40.49 | 41.67 | 40.49 | 41.67 | 40.49 | 41.70* | 40.49 | 41.70 | 40.49 |
| DFF_03 | 41.24 | 40.79 | 41.83* | 40.79 | 41.83 | 40.79 | 41.83 | 40.79 | 41.83 | 40.79 | 41.83 | 40.79 |
| DFF_04 | 41.45 | 40.86 | 42.01* | 40.86 | 42.01 | 40.86 | 42.01 | 40.86 | 42.05* | 40.86 | 42.05 | 40.86 |
| DFF_05 | 41.49 | 41.04 | 42.04* | 41.04 | 42.04 | 41.04 | 42.04 | 41.04 | 42.05* | 41.04 | 42.05 | 41.04 |
| DFF_06 | 41.63 | 41.09 | 42.05* | 41.09 | 42.05 | 41.09 | 42.05 | 41.09 | 42.09* | 41.09 | 42.09 | 41.09 |
| DFF_07 | 41.68 | 41.11 | 42.24* | 41.11 | 42.24 | 41.11 | 42.24 | 41.11 | 42.27* | 41.11 | 42.27 | 41.11 |
| DFF_08 | 41.70 | 41.12 | 42.25* | 41.12 | 42.25 | 41.12 | 42.25 | 41.12 | 42.27* | 41.12 | 42.27 | 41.12 |
| DFF_09 | 41.63 | 41.16 | 42.17* | 41.16 | 42.17 | 41.16 | 42.17 | 41.16 | 42.21* | 41.16 | 42.21 | 41.16 |
| DFF_10 | 41.90 | 41.17 | 42.49* | 41.17 | 42.49 | 41.17 | 42.49 | 41.17 | 42.49* | 41.17 | 42.49 | 41.17 |
| DFF_11 | 41.83 | 41.19 | 42.36* | 41.19 | 42.36 | 41.19 | 42.36 | 41.19 | 42.39* | 41.19 | 42.39 | 41.19 |
| DFF_12 | 42.09 | 41.32 | 42.48* | 41.32 | 42.48 | 41.32 | 42.48 | 41.32 | 42.51* | 41.32 | 42.51 | 41.32 |
| DFF_13 | 42.10 | 41.36 | 42.53* | 41.36 | 42.53 | 41.36 | 42.53 | 41.36 | 42.55* | 41.36 | 42.55 | 41.36 |
| DFF_14 | 42.15 | 41.40 | 42.71* | 41.40 | 42.71 | 41.40 | 42.71 | 41.40 | 42.72* | 41.40 | 42.72 | 41.40 |
| DFF_15 | 42.01 | 41.57 | 42.48* | 41.57 | 42.48 | 41.57 | 42.48 | 41.57 | 42.48 | 41.57 | 42.48 | 41.57 |
| DFF_16 | 41.58 | 41.78 | 42.05* | 41.78 | 42.05 | 41.78 | 42.05 | 41.78 | 42.05 | 41.78 | 42.05 | 41.78 |
| DFF_17 | 42.39 | 41.63 | 43.43* | 42.24* | 43.43 | 42.24 | 43.43 | 42.24 | 43.43 | 42.24 | 43.43 | 42.24 |
| DFF_18 | 42.09 | 41.92 | 42.55* | 41.92 | 42.55 | 41.92 | 42.55 | 41.92 | 42.55 | 41.92 | 42.55 | 41.92 |

***Table VII***
*Results on an industrial benchmark for best-case delay annotation.*

Apart from loading files, the first slack double column under “No Pessimism Reduction” corresponds to verifying all 57063 setup checks using the basic waveform narrowing method without using the pessimism reduction techniques. The double columns under “Spatial Correlation Order *n*” correspond to performing pessimism reduction only on the cases shown in the tables. Pessimism reduction was run independently for each column, i.e., the basic method is applied to all 57063 checks, than pessimism reduction with spatial correlations of order *n* is applied. Note that, in this particular example, little benefit was gained by setting the spatial correlation order to higher than 1. The shaded table elements correspond to the cases for which the higher order made a difference.

| Worst Case | No Pessimism Reduction 1.46 minutes | | Spatial Correlation Order 1 0.51 minute | | Spatial Correlation Order 2 0.75 minute | | Spatial Correlation Order 3 1.24 minutes | | Spatial Correlation Order 4 1.4 minutes | | Spatial Correlation Order 8 13 minutes | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Check** | R | F | R | F | R | F | R | F | R | F | R | F |
| DFF_01 | 10.92 | 8.740 | 17.79* | 17.45* | 17.79 | 17.45 | 17.79 | 17.45 | 17.79 | 17.45 | 17.79 | 17.45 |
| DFF_19 | 22.04 | 20.96 | 22.04 | 20.96 | 22.04 | 20.96 | 22.04 | 20.96 | 22.04 | 20.96 | 22.04 | 20.96 |
| DFF_02 | 21.05 | 21.18 | 21.05 | 21.18 | 21.05 | 21.18 | 21.05 | 21.18 | 21.05 | 21.18 | 21.05 | 21.18 |
| DFF_03 | 22.40 | 21.07 | 22.40 | 21.07 | 22.40 | 21.07 | 22.40 | 21.07 | 22.40 | 21.07 | 22.40 | 21.07 |
| DFF_20 | 22.11 | 21.07 | 22.11 | 21.07 | 22.11 | 21.07 | 22.11 | 21.07 | 22.11 | 21.07 | 22.11 | 21.07 |
| DFF_21 | 22.55 | 21.45 | 22.55 | 21.45 | 22.55 | 21.45 | 22.55 | 21.45 | 22.55 | 21.45 | 22.55 | 21.45 |
| DFF_22 | 22.59 | 21.49 | 22.59 | 21.49 | 22.59 | 21.49 | 22.59 | 21.49 | 22.59 | 21.49 | 22.59 | 21.49 |
| DFF_18 | 23.18 | 21.54 | 23.18 | 21.54 | 23.18 | 21.54 | 23.18 | 21.54 | 23.18 | 21.54 | 23.18 | 21.54 |
| DFF_23 | 22.69 | 21.60 | 22.69 | 21.60 | 22.69 | 21.60 | 22.69 | 21.60 | 22.69 | 21.60 | 22.69 | 21.60 |
| DFF_24 | 22.67 | 21.60 | 22.67 | 21.60 | 22.67 | 21.60 | 22.67 | 21.60 | 22.67 | 21.60 | 22.67 | 21.60 |
| DFF_15 | 23.01 | 21.71 | 23.01 | 21.71 | 23.01 | 21.71 | 23.01 | 21.71 | 23.01 | 21.71 | 23.01 | 21.71 |
| DFF_25 | 22.38 | 21.73 | 22.38 | 21.73 | 22.38 | 21.73 | 22.38 | 21.73 | 22.38 | 21.73 | 22.38 | 21.73 |
| DFF_26 | 22.82 | 21.74 | 22.82 | 21.74 | 22.82 | 21.74 | 22.82 | 21.74 | 22.82 | 21.74 | 22.82 | 21.74 |
| DFF_27 | 22.82 | 21.76 | 22.82 | 21.76 | 22.82 | 21.76 | 22.82 | 21.76 | 22.82 | 21.76 | 22.82 | 21.76 |
| DFF_28 | 22.5 | 21.86 | 22.5 | 21.86 | 22.5 | 21.86 | 22.5 | 21.86 | 22.5 | 21.86 | 22.5 | 21.86 |
| DFF_04 | 23.52 | 21.89 | 23.52 | 21.89 | 23.52 | 21.89 | 23.52 | 21.89 | 23.52 | 21.89 | 23.52 | 21.89 |
| DFF_10 | 23.22 | 21.92 | 23.22 | 21.92 | 23.22 | 21.92 | 23.22 | 21.92 | 23.22 | 21.92 | 23.22 | 21.92 |
| DFF_13 | 23.13 | 21.96 | 23.13 | 21.96 | 23.13 | 21.96 | 23.13 | 21.96 | 23.13 | 21.96 | 23.13 | 21.96 |
| DFF_11 | 22.99 | 21.96 | 22.99 | 21.96 | 22.99 | 21.96 | 22.99 | 21.96 | 22.99 | 21.96 | 22.99 | 21.96 |

***Table VIII***
*Results on an industrial benchmark for worst-case delay annotation.*

The results for this example are very significant; they prove that false path elimination should not be ignored by industry because it is necessary and affordable. For this example, the waveform narrowing method proved that the slack lower bound for setup checks is in fact 17.459 ns as compared to 8.74 ns reported by topological analysis. The clock period in the test case is 100 ns, therefore, the relative safe margin proved to be 17.459% of the clock cycle instead of 8.74% !

The resources required by the program are minimal. The memory requirement and execution time for the basic method without pessimism reduction scales linearly with circuit size, and the constant factor of our implementation is very efficient. Although the time complexity of pessimism reduction procedures is bounded by the spatial correlation, which is virtually quadratic, we expect it to grow much slower with circuit size. This belief is backed by two facts:

1) The complexity of the timing checks does not grow with circuit size. Timing check cones are limited to a limited number of logic levels. Usually, deep logic structures are broken into pipelines to keep up with the appealing higher clock frequencies. In fact, bigger circuits contain more checks not more complex ones.

2) When a timing check still has the worst calculated slack after it is refined by applying the techniques of pessimism reduction, there is no need to go any further.

In our example, only 19 timing checks out of 57063 have their slack less than 23 ns in the worst-case delay annotation (Table VIII). After the worst-case DFF_01 is refined from 8.74 to 17.45 ns, it is still the worst case. Therefore, there is no need to go any further, and pessimism reduction execution time is actually less than 2 seconds rather than the 0.51 minute for spatial correlation of order 1. The remaining checks were refined simply to collect more data for the sake of this illustration.

## 4.8 Conclusions

In this chapter, we presented the basic concepts behind delay correlation schemes using three valued delays (min, typ, max), accommodating cell position dependence and the ability to support precise process characterization. We also presented the extensions that we implemented for the method to become applicable to state of the art industrial circuits. We showed the hard primitive constraints, which are the building blocks for cell library modeling. We presented the concepts behind combinational cell modeling supporting separate rising and falling i/o path delays. A simple and intuitive, yet very powerful, clock definition formalism was presented, able to express arbitrary complex clocking schemes. The resulting timing verifier was tested successfully on a real world industrial synchronous design; false paths were eliminated successfully.

# CHAPTER V CONCLUSIONS

We presented in this thesis an elaborate timing verification method based on *Waveform Narrowing* proposed initially by Cerny and Zejda in [98]. The method is in fact a custom constraint programming system adapted to timing verification of logic circuits. It consists of modeling the circuit timing constraints and operating conditions as a constraint system that is consistent, i.e., has a solution, iff the timing constraints are violated. The constraint system is composed of a finite set of variables $\{X_1, X_2,..., X_n\}$ which take values from their respective domains $D_1$, $D_2$,..., $D_n$, and a set of relational constraint operators $\{C_1, C_2,..., C_m\}$, each operating on a subset of the variables. A variable represents either a circuit net, or a gate or interconnect delay. Net domains contain sets of binary waveforms, whereas delay domains are intervals. A constraint operator is a logic gate function defined over sets of waveforms and interval delays. A domain $D_k$ of a variable $X_k$ contains initially the set of all possible values $X_k$ can take. A solution of the constraint system is an assignment for all the variables, from their respective domains, that makes the system consistent, i.e., all the constraints are satisfied. When a constraint $C_k$ is applied, it removes from the domains of the associated variables values that are not compatible, i.e., values that are not part of any solution. The system is then solved by repeatedly applying the constraints until the greatest fixpoint is reached. If we end up with empty domains, we conclude that the timing constraints are satisfied; otherwise, no conclusion can be drawn.

The foundation of the waveform narrowing method is based on local consistency techniques. Constraints consider each gate as isolated, ignoring the global circuit function. Therefore, the system evaluation may result in false negative answers when the resulting domains are not empty, and yet the constraint system has no solution. To reduce this pessimism we developed two polynomial techniques:

- *Timing dominators* concept, which determines key circuit nets for which the domains can be narrowed as a consequence of necessary conditions based on the global circuit function;

- Spatial correlation procedure, which enforces partially the global circuit function by restricting the domains of selected reconvergent fan-outs to waveforms stabilizing at 0 and 1, and then merging the results.

Also, we developed a case analysis procedure able to find a test vector (solution of the constraint system, proof of timing violation), or prove that no violation is possible.

When tested on the ISCAS'85 benchmark suite, the original method eliminated violations from the c5315 and c7552 circuits. The use of *timing dominators* alone eliminated violations from the traditionally difficult c1908, and from c3540. *Timing dominators*, combined with spatial correlation, eliminated violations from c2670 and also the traditionally difficult multiplier, c6288. In summary, *timing dominators* and spatial correlation techniques made the *Waveform Narrowing* method determine tight circuit delay upper bounds that correspond to the exact circuit delays for all ISCAS'85 circuits. Moreover, except for c6288, the case analysis procedure found test vectors for all circuits with a remarkably low number of backtracks!

Motivated by the success of the method when tested on standard benchmark circuits, we wanted to evaluate its effectiveness on real world industrial circuits. The task, however, turned out to be complex and required us to rewrite the software from scratch. We implemented appropriate capabilities for cell library modeling, standard delay back-annotation, complex clocking schemes, etc.

The resulting timing verifier was tested on a 122K-gate industrial circuit, provided by Nortel Networks. The results were very significant because they proved that false path elimination should not be ignored by industry because it is necessary and affordable. For this instance, our method proved that the slack lower bound for setup checks is in fact 17.459 ns as compared to 8.74 ns reported by topological analysis. The clock period in the test case is 100 ns, therefore, the relative safe margin proved to be 17.459% of the clock cycle instead of 8.74% !

## 5.1 Comparison with Other Methods

Although our method has a very competitive execution time as compared to other methods, we believe that the real comparison comes from other points of view:

1) Timing verification is an NP-Hard problem. Therefore, any exact method has exponential time complexity in the worst case. All algorithms that aim at finding test vectors rely on heuristics biased toward resolving certain types of circuits. Commercial timing verifiers need approximate false path elimination methods that have predictable execution time, and are statistically efficient. Our method proved to have this property on all tested circuits. In fact only 10% of the reported execution time for the ISCAS'85 circuits (except c6288) were used for false path elimination. The remaining 90% were used to prove that the delay upper bound we determined is actually the exact circuit delay. The results for the industrial circuit were even better; they showed that the execution time required by our method grows virtually linearly with the circuit size, although the spatial correlation is quadratic. In fact pessimism reduction techniques need to be applied only to timing checks having the worst slacks, and bigger circuits contain more checks, not more complex ones. For our example of 57063 checks, only 19 of them had slacks less then 23% of the clock period before pessimism reduction.

2) The memory requirement is critical in today's integrated electronic design automation tools. Our method requires a simple data structure that represents the graph of constraints, and a domain stack for each net that can be limited to three domain instances. The implementation averaged at 1188 bytes per industrial gate. This requirement can be brought down if fixed delay model is used instead of the three valued delay model. In contrast, any method that attempts to build binary decision diagrams to resolve the satisfiability of a Boolean function $f$ has an exponential space complexity in the worst case (in terms of the number of variables of $f$)!

3) Our method has a unique delay correlation model that is able to express complex correlation schemes like two-dimensional position dependence.

4) As a verification environment, the waveform narrowing is very flexible and intuitive. In fact the constraint system resembles a simulation environment, except that the gates operate on sets of waveforms instead of instantaneous values. The constraint network that does the verification, and the circuit description, are one entity. No dynamic structures need to be created for each individual check. Clock trees are traced on the fly, and timing checks that verify the integrity of gated clocks can be inserted very easily.

## 5.2 A Disadvantage

A disadvantage of the waveform narrowing method is its complexity. In fact, it takes a lot of effort before one can develop intuition about waveform narrowing. Therefore, a new software developer allocated to maintain and enhance a timing verifier based on waveform narrowing needs to go through a steep learning curve.

## 5.3 Original Contributions of this Thesis

The major contributions of this thesis are summarized as follows:

- Established the mathematical foundations of the *waveform narrowing* method for the purpose of floating mode delay calculation, the original method was formulated around the transition mode.

- Developed the spatial correlation procedure that was effective in reducing the pessimism of the method on standard and industrial benchmark circuits. The added execution time complexity is quadratic.

- Developed the *Timing Dominators* concept that was very successful in eliminating false violations with minimal added execution time complexity ($n \times \log(n)$).
- Developed a case analysis procedure able to find a test vector, or prove that no violation is possible. The procedure is guided by heuristics inspired by ATPG techniques, namely the controllability measure of [23] and the FAN algorithm of [33]. The procedure uses a novel partitioning strategy based on timing dominators.

In order to provide support for state of the art industrial circuits, we extended the method as follows:

- Developed an intuitive formalism able to express arbitrary complex clocking schemes, along with a procedure to deduce correct default edge selection for setup verification.
- Defined a delay correlation domain based on three-valued delay annotation (min, typ, max) using the novel concept of normalized delays. The resulting constraints can be used to build complex correlation networks able to model arbitrary complex component delay correlation, like position dependence, rising-delay vs. falling-delay, etc.
- Defined more than 70 constraint primitives able to model industrial cell libraries.
- Developed a hard multiplexer primitive that reduces the inherent pessimism of the *floating mode* delay model;
- Developed and automated a general concept for modeling combinational cells. And added cell aware constraints that remove the pessimism induced by path delays of unknown polarities;
- Added support for automatic handling of combinational loops, still present in some synchronous industrial designs;
- Implemented an industrial-grade version of the timing verifier in the object oriented language C++, and evaluated the waveform narrowing method on industrial circuits provided by Nortel Networks.

## 5.4 Future Work

Two objectives were in mind when we started work on the timing verification method based on *Waveform Narrowing*: First, to reduce its pessimism, then to assess its suitability for industrial use. While we succeeded in both, there is still a lot of gaps that need to be filled.

From an evaluation point of view:

- We could not test all the capabilities of our timing verifier due to data unavailability. For instance, it is not clear how circuit components are related. Like, how rising delays, affected mostly by PMOS transistors, are correlated to falling delays, affected mostly by NMOS transistors. Or how interconnect delays belonging to different metal layers are correlated, etc. We believe that component delay correlation is a subject that needs to be researched from a process characterization perspective.

On another front, our timing verifier is missing important features needed by commercial tools. They are summarized as follows:

- Support for slew dependent delays.
- Support for latch based designs. Modern design methodologies favor transparent latches to store the circuit state, due to their advantage in preventing race conditions.
- Support for derived clocks. Test cases presented to timing verifiers predefine derived clocks based on manual calculations or spice simulations that do not necessarily reflect the current configuration of delay annotation. For example, consider a multi-phase clocking scheme that is generated using a master low frequency clock that drives a PLL frequency multiplier, which in turn drives a clock generation circuit. Defining the clock generated by the PLL in the test case is no cause of errors, however, defining the clocks generated by the clock generator involves using discrete component delay values that are not valid in all configurations, making the clock phases relative positions incorrect.

- We believe that the spatial correlation procedure can be enhanced further by using the heuristics used by case analysis instead of simply using topological sorting on reconvergent fan-outs. Limiting the fan-outs selected for spatial correlation to the ones with conflicting requirements for sensitizing the longest paths would enhance the effectiveness of higher spatial correlation orders.

# REFERENCES

## Miscellaneous

[1] "Industry's first 0.1 micron production silicon," *Motorola Press Release*, Feb. 1999, http://www.mot-sps.com/news_center/press_releases/PR990201A.html

[2] R. E. Tarjan: "Finding Dominators in Directed Graph," *SIAM Journal on Computing*, vol. 3, pp. 62-89, 1974.

[3] F. Brglez, H. Fujiwara: "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *International Symposium on Circuits and Systems*, pp. 695-698, 1985

[4] R. E. Bryant: "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Transactions on Computers*, vol. 33, pp. 160-177, February 1984.

[5] J. Hayes: "Digital Simulation with Multiple Logic Values," *IEEE Transactions on Computer-Aided Design*, vol. CAD-5, no. 2, pp. 274-283, 1986.

[6] J. P. Hayes: "An Introduction to Switch-Level Modeling," *IEEE Design and Test*, vol. 4, pp. 18-25, August 1987.

[7] V. Chandramouli, et al: "AFTA: A Formal Delay Model for Functional Timing Analysis," *Design, Automation and Test in Europe*, pp. 350-355, 1998.

[8] S. Devadas, et al: "Event Suppression: Improving the Efficiency of Timing Simulation for Synchronous Digital Circuits," *IEEE Transactions on CAD*, vol. 12, pp. 814-822, June 1994.

[9] L. W. Nagel: "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Memo ERL-M520, *Department of Electrical Engineering and Computer Science, University of California*, Berkeley, May 1975.

[10] G. DeMicheli: Synthesis and Optimization of Digital Circuits, *McGraw-Hill*, New York, 1994.

[11] P. Van Hentenryck: Constraint Satisfaction in Logic Programming, *MIT Press*, Cambridge, MA, 1989.

[12] TGC1000/TEC1000 5-V CMOS Array, *Texas Instrument*, 1993.

[13] Standard Delay Format Specification 3.0, *Open Verilog International*, 1995

[14] F. Benhamou, W. J. Older: "Applying Interval Arithmetic to Real, Integer and Boolean Constraints," *Journal of Logic Programming*, vol. 32, no. 1, pp. 1-24, 1997.

**Logic Function Representation / Manipulation**

[15] Sheldon B. Akers: "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. c-27, no. 6, pp. 509-516, June 1978.

[16] R. Bryant: "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. c-35, no. 8, pp. 677-691, Aug. 1986.

[17] M. Fujita, H. Fujisawa, and N. Kawato: "Evaluation and Improvement of Boolean Comparison Method Based on Binary Decision Diagrams," *IEEE International Conference on Computer-Aided Design*, pp. 2-5, 1988.

[18] S.I. Minato, N. Ishiura, S. Yaijma: "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *27TH Design Automation Conference*, pp. 52-57, 1990.

[19] H. T. Liaw, C. S. Lin: "On the OBDD Representation of General Boolean Functions," *IEEE Transactions on Computers*, pp. 661-664, 1992.

[20] S. Devadas: "Comparing Two-Level and Ordered Binary Decision Diagram Representations of Logic Functions," *IEEE Transactions on Computer-Aided Design*, pp. 722-723, May 1993.

[21] J. Gergov, C. Meinel: "Mod-2-OBDD's: A Generalization of OBDD's and EXOR-Sum-of-Products," *Proceedings of the IFIP Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 1993.

**Controllability / Visibility Heuristics**

[22] Lawrence Goldstein: "Controllability/Observability Analysis of Digital Circuits," *IEEE Transactions on Circuits and Systems*, vol. CAS-26, pp. 685-693, September 1979.

[23] Lawrence Goldstein: "SCOAP: Sandia Controllability/Observability Analysis Program," *Proceedings of the 17TH Design Automation Conference*, pp. 190-196, June 1980.

[24] Vishwani Agrawal, Sharad Seth, M. Mercer: "Testability Measures - What do they tell us?" *International Test Conference*, Philadelphia, Digest of pp., pp. 391-396, 1982.

[25] F. Berglez, P. Pownall, R. Hum: “Application of Testability Analysis: From ATPG to Critical Path Tracing,” *Proceedings 1984 of the International Test Conference*, pp. 705-712, October 1984

[26] A. Lioy, M. Mezzalam: “On Parameters Affecting ATPG Performance,” *Proceedings of CompEuro 1987*, pp. 394-397, 1987.

[27] Sharad Seth, Vishwani Agrawal: “An Exact Analysis for Efficient Computation of Random-Pattern Testability in Combinational Circuits,” *Proceedings of the 16$^{TH}$ Annual International Symposium on Fault-Tolerant Computing Systems*, pp. 318-323, 1986.

[28] Wen Ching Wu, Chung Len Lee: “A Probabilistic Testability Measure for Delay Faults,” *28$^{TH}$ ACM/IEEE Design Automation Conference*, pp. 440-445, 1991.

[29] Sarma Sastry, Amitava Majumdar: “A Branching Process Model for Observability of Combinational Circuits,” *28$^{TH}$ ACM/IEEE Design Automation Conference*, pp. 452-457, 1991.

[30] S. Devadas, A. Ghosh and K. Keutzer: "An Observability-Based Code Coverage Metric for Functional Simulation", *Proceedings of the International Conference on Computer-Aided Design*, November 1996.

**Automatic Test Pattern Generation**

[31] J. Paul Roth: “Diagnosis of Automata Failures: A Calculus and a Method,” *IBM Journal*, pp. 278-291, July 1966

[32] Prabhakar Goel: “An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits,” *IEEE Transactions on Computers*, vol. c-30, no. 3, pp. 215-222, March 1981.

[33] Hideo Fujiwara, Takeshi Shimono: “On the Acceleration of Test Generation Algorithms,” *IEEE Transactions on Computers*, vol. c-32, no. 12, pp. 1137-1144, December 1983.

[34] F. Maamari, J. Rajski: “Reconvergent Fanout Analysis and Fault Simulation Complexity of Combinational Circuits,” *IEEE Electronics Letters*, vol. 23, no. 21, pp. 1131-1133, October 1987.

[35] Michael Schulz, Erwin Trischler, Thomas Sarfert: “SOCRATES: A Highly Efficient Automatic Test Pattern Generation System,” *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 1, pp. 126-137, January 1988.

[36] Fadi Maamari, Janusz Raiski: “A Reconvergent Fanout Analysis for Efficient Exact Fault Simulation of Combinational Circuits,” *18$^{TH}$ International Fault Tolerant Symposium*, pp. 122-126, June 1988.

[37] John Calhoun, Franc Brglez: "A Framework and Method for Hierarchical Test Generation," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 45-67, January 1992.

[38] Thomas Sarfert et al: "A Hierarchical Test Pattern Generation System Based on High-Level Primitives," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 34-44, January 1992.

[39] Wolfgang Kunz, Dhiraj Pradhan: "Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits," *International Test Conference*, pp. 816-825, 1992.

[40] Hyoung Min, Hwei-tsu Luh, William Rogers: "Hierarchical Test Pattern Generation: A Cost Model and Implementation," *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems*, vol. 12, no. 7, pp. 1029-1039, July 1993.

[41] Irith Pomeranz, Lakshmi Reddy, Sudhakar Reddy: "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1040-1049, July 1993

[42] H. Cox, J. Rajski: "On Necessary and Nonconflicting Assignments in Algorithmic Test Pattern Generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 515-530, April 1994

**Delay Automatic Test Pattern Generation**

[43] Chin Jen Lin, Sudhakar Reddy: "On Delay Fault Testing in Logic Circuits," *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, no. 5, pp. 694-703, September 1987.

[44] Jacob Savir, William McAnney: "Random Pattern Testability of Delay Faults," *IEEE Transactions on Computers*, vol. 37, no. 3, pp. 291-300, March 1988.

[45] Karl Fuchs, Franz Fink, Michael Shulz: "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 10, pp. 1323-1335, October 1991.

[46] Eun Sei Park, Ray Mercer: "An Efficient Delay Test Generation System for Combinational Logic Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 7, pp. 926-938, July 1992.

[47] Alexander Saldanha, Robert Brayton, Alberto Sangiovanni-Vincentelli: "Equivalence of Robust Delay-Fault and Single Stuck-Fault Test Generation," *$29^{TH}$ ACM/IEEE Design Automation Conference*, pp. 173-176, 1992.

[48] Irith Pomeranz, Sudhakar Reddy: "At-Speed Delay Testing of Synchronous Sequential Circuits," *29$^{TH}$ ACM/IEEE Design Automation Conference*, pp. 177-181, 1992.

[49] Debashis Bhattacharya, Prathima Agrawal, Vishwani Agrawal: "Delay Fault Test Generation for Scan/Hold Circuits using Boolean Expressions," *29$^{TH}$ ACM/IEEE Design Automation Conference*, pp. 159-164, 1992.

[50] Kwang-Tng Cheng, Srinivas Devadas, Kurt Keutzer: "Delay-Fault Test Generation and Synthesis for Testability Under a Standard Scan Design Methodology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 8, pp. 1217-1231, August 1993.

[51] Karl Fuchs, Michael Pabst, Torsten Rössel: "RESIST: A Recursive Test Pattern Generation Algorithm for Path Delay Faults Considering Various Test Classes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 12, pp. 1550-1561, December 1994.

[52] Wuudiann Ke, P. R. Menon: "Path-Delay-Fault Testable Nonscan Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 5, pp. 576-581, May 1995.

[53] Irith Pomeranz, Sudhakar M. Reddy, Prasanti Uppaluri: "NEST: A Nonenumerative Test Generation Method for Path Delay Faults in Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 12, pp. 1505-1515, December 1995.

[54] U. Sparmann, D. Luxenburger, K.-T. Cheng, and S.M. Reddy: "Fast Identification of Robust Dependent Path Delay Faults," *32$^{ND}$ Design Automation Conference*, pp. 119-125, 1995.

[55] Kwang-Ting Cheng, Hsi-Chuan Chen: "Classification and Identification of Nonrobust Untestable Path Delay Faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 8, pp. 845-853, August 1996.

**Timing Verification**

[56] T. I. Kirkpatrick and N. R. Clark: "PERT as an Aid to Logic Design," *IBM Journal of Research and Development*, vol. 10, pp. 135-141, 1966.

[57] V. H. Hrapcenko: "Depth and Delay in a Network," *Soviet Math. Dokl.*, vol. 19, pp. 1006-1009, 1978.

[58] T. M. McWilliams: "Verification of Timing Constraints on Large Digital Systems," *17$^{TH}$ Design Automation Conference*, pp. 139-147, 1980.

[59] R. B. Hitchcock, et al: "Timing Analysis of Computer Hardware," *IBM Journal of Research and Development*, vol. 26, pp. 100-105, January 1982.

[60] N. P. Jouppi: "TV: An NMOS Timing Analyzer," *Proceedings of the 3$^{RD}$ Caltech VLSI Conference*, pp. 71-86, March 1983.

[61] N. P. Jouppi: "Timing Analysis for NMOS VLSI," *20$^{TH}$ Design Automation Conference*, 1983.

[62] J. K. Ousterhout: "A Switch-Level Verifier for Digital MOS VLSI," *IEEE Transactions on CAD*. vol. 4, pp. 336-349, 1985.

[63] D. Brand, V. S. Iyengar: "Timing Analysis Using Functional Relationships," *Proceedings of the International Conference on Computer-Aided Design*, pp. 126-129, 1986.

[64] M. R. Dagenais, N. C. Rumin: "Circuit-Level Timing Analysis and Design Verification of High-Performance MOS Computer Circuits," *International Conference on Computer Design*, October 1986.

[65] M. R. Dagenais, N. C. Rumin: "Timing Analysis and Verification of Digital MOS Circuits," *Proceedings of CompEuro 1987*, pp. 242-245, May 1987.

[66] J. Benkoski, E. Meersch, L. Claesen, and H. De Man: "Efficient Algorithms for Solving The False Path Problem in Timing Verification," *Proceedings of the International Conference on Computer-Aided Design*, pp. 44-47, 1987.

[67] N. P. Jouppi: "Timing Analysis and Performance Improvement of MOS VLSI Designs," *IEEE Transactions on CAD*, vol. 6, pp. 650-665, July 1987.

[68] D. E. Wallace, C. H. Sequin: "ATV: An Abstract Timing Verifier," *25$^{TH}$ Design Automation Conference*, pp. 154-159, 1988.

[69] P. Das, et al: "Hierarchical Timing View Generation Including Accurate Modeling for False Paths," *Proceedings of the Custom Integrated Circuits Conference*, pp. 13.3.1-13.3.4, 1989.

[70] S. Perremans, L. Claesen, H. De Man: "Static Timing Analysis of Dynamically Sensitizable Paths," *26$^{TH}$ Design Automation Conference*, pp. 568-573, 1989.

[71] Du, D.H.C., S.H.C. Yen, and S. Ghanta: "On the General False Path Problem in Timing Analysis," *26$^{TH}$ Design Automation Conference*, pp. 555-560, 1989.

[72] P. C. McGeer, R. K. Brayton: "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network," *26$^{TH}$ Design Automation Conference*, pp. 561-567, 1989.

[73] J. Benkoski, E. V. Meersch, and H. De Man: "Timing Verification Using Statically Sensitizable Paths," *IEEE Transactions on CAD*, vol. 9, pp. 1073-1084, 1990.

[74] A. R. Martello, S. P. Levitan, D. M. Chiarulli: "Timing Verification Using HDTV," *$27^{TH}$ Design Automation Conference*, pages 118-123, 1990.

[75] Y. C. Ju, R. A. Saleh: "Incremental Techniques for the Identification of Statically Sensitizable Critical Paths," *$28^{TH}$ Design Automation Conference*, pp. 541-546, 1991.

[76] P. C. McGeer, R. K. Brayton: Integrating Functional and Temporal Domains in Logic Design, *Kluwer Academic Publishers*, 1991.

[77] P. C. McGeer, R. K. Brayton: "Timing Analysis and Delay-Fault Test Generation Using Path-Recursive Functions," *International Conference on Computer Aided Design*, pp. 180-183, 1991.

[78] J. P. M. Silva, et al: "FPD- An Environment for Exact Timing Analysis," *International Conference on Computer Design*, pp. 212-215, 1991.

[79] R. Steward, J. Benkoski: "Static Timing Analysis Using Interval Constraints," *International Conference on Computer Aided Design*, pp. 308-311, 1991.

[80] P. Ashar, S. Dey, S. Malik: "Exploiting Multi-Cycle False Paths in the Performance Optimization of Sequential Circuits," *Proceedings of the International Conference on Computer-Aided Design*, pp. 510-517, 1992.

[81] C. Ramachandran, F. J. Kurdahi: "Combined Topological and Functionality Based Delay Estimation Using a Layout-Driven Approach for High Level Applications," *European Design Automation Conference*, pp. 72-78, 1992.

[82] Srinivas Devadas, Kurt Keutzer, Sharad Malik, Albert Wang: "Certified Timing Verification and the Transition Delay of a Logic Circuit," *$29^{TH}$ Design Automation Conference*, pp. 549-555, 1992.

[83] P. Johannes, et al: "Performance Through Hierarchy in Static Timing Verification," *$12^{TH}$ World Computer Congress*, vol. 1, pp. 703-709, Madrid, 1992.

[84] A. Kuehlmann, R. A. Bergamaschi: "Timing Analysis in High-Level Synthesis," *European Design Automation Conference*, pp. 349-354, 1992.

[85] W.K.C. Lam, R.K. Brayton, and A.L. Sangiovanni-Vincentelli: "Circuit Delay Models and Their Exact Computation Using Timed Boolean Functions," *$30^{TH}$ Design Automation Conference*, pp. 128-134, 1993.

[86] P. Ashar, Sharad Malik, and S. Rothweiler: "Functional Timing Analysis using ATPG," *Proceedings of the European Design Automation Conference*, pp. 506-510, 1993.

[87] H. Chang, J. A. Abraham: "VIPER: An Efficient Vigorously Sensitizable Path Extractor," *30$^{TH}$ Design Automation Conference*, pp. 112-117, 1993.

[88] H. C. Chen, D. H. C. Du: "Path Sensitization in Critical Path Problem." *IEEE Transactions on CAD*, vol. 12, pp. 196-207, February 1993.

[89] J. P. M. Silva, K. A. Sakallah: "Concurrent Path Sensitization in Timing Analysis," *European Design Automation Conference*, pp. 196-199, 1993.

[90] J. P. M. Silva, K. A. Sakallah: "An Analysis of Path Sensitization Criteria," *International Conference on Computer-Aided Design*, pp. 68-72, 1993.

[91] T. Kamel: "Design and Implementation of a Static Timing Analyzer Using CLP (BNR)," *internal report of Computing Research Lab, Bell-Northern Research*, 1993.

[92] Yves Blaquière, Michel R. Dagenais, and Yvon Savaria: "A New Accurate and Hierarchical Timing Analysis Approach," *Proceedings of the European Conference on Design Automation*, pp. 449-454, 1993.

[93] Srinivas Devadas, Kurt Keutzer, and Sharad Malik: "Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms," *IEEE Transactions on CAD*, vol. 12, pp. 1913-1923, December 1993.

[94] Shiang-Tang Huang, Tai Ming Parng and Jyuo Min Shyu: "A Polynomial-Time Heuristic Approach to Approximate a Solution to the False Path Problem," *30$^{TH}$ Design Automation Conference*, pp. 118-122, 1993.

[95] S. Devadas, K. Keutzer, S. Malik, and A. Wang: "Computation of Floating Mode Delay in Combinational Circuits: Practice and Implementation," *IEEE Transactions on CAD*, vol. 12, pp. 1924-1936, December 1993.

[96] R. Iris Bahar, Hyunwoo Cho, Gary D. Hachtel, Enrico Macii, and Fabio Somenzi: "Timing Analysis of Combinational Circuits Using ADD's," *Proceedings of the European Design and Test Conference*, pp. 625-629, 1994.

[97] S. Bhattacharya - S. Dey - F. Brglez: "Provably Correct High-Level Timing Analysis Without Path Sensitization," *Proceedings of the International Conference on Computer-Aided Design*, pp. 736-742, 1994.

[98] E. Cerny, J. Zejda: "Gate-Level Timing Verification Using Waveform Narrowing," *Proceedings of the European Design Automation Conference*, pp. 374-379, 1994.

[99] A. Gupta, D. P. Siewiorek: "Automated Multi-Cycle Symbolic Timing Verification of Microprocessor-Based Design," *31$^{ST}$ Design Automation Conference*, pp. 113-119, 1994.

[100] S.-T. Huang, T.-M. Parng, and J.-M. Shyu: "Timed Boolean Calculus and its Applications in Timing Analysis," *IEEE Transactions on Computer Assisted Design*, vol. 13, pp. 318-337, March 1994.

[101] William K. C. Lam, Robert K. Brayton and Alberto L. Sangiovanni-Vincentelli: "Exact Minimum Cycle Times for Finite State Machines," *31ST Design Automation Conference*, pp. 100-105, 1994.

[102] W. K. C. Lam, R. K. Braytom: Timed Boolean Functions - a Unified Formalism for Exact Timing Analysis, *Kluwer Academic Publishers*, 1994.

[103] R. P. Lloppis: "Exact Path Sensitization in Timing Analysis," *European Design Automation Conference*, 1994.

[104] J. P. M. Silva, K. A. Sakallah: "Efficient and Robust Test Generation-Based Timing Analysis," *International Symposium on Circuits and Systems*, pp. 303-306, 1994.

[105] S.Z. Sun and David H.C. Du: "Efficient Timing Analysis for CMOS Circuits Considering Data Dependent Delays," *International Conference on Computer Design*, pp. 156-159, 1994.

[106] J. P. Silva, K. A. Sakallah: "Dynamic Search-Space Pruning Techniques in Path Sensitization, *31ST ACM/IEEE Design Automation Conference*, pp. 705-711, 1994.

[107] H. Yalcin, J. P. Hayes: "Hierarchical Timing Analysis Using Conditional Delays," *International Conference on Computer-Aided Design*, pp. 371-377, 1995,

[108] H. Yalcin, J. P. Hayes, K. A. Sakallah: "An Approximate Timing Analysis Method for Datapath Circuits," *International Conference on Computer-Aided Design*, pp. 114-118, 1996.

[109] E. Cerny: "Gate-Level Timing Verification with Component Delay Correlation," *internal report, Université de Montréal*, 1996.

[110] C. Oh, R. Mercer: "Efficient Logic-Level Timing Analysis Using Constraint-Guided Critical Path Search," *IEEE Transactions on VLSI Systems*, vol. 4, no. 3, pp. 346-354, September 1996.

[111] H. Yalcin, J. P. Hayes: "Event Propagation Conditions in Circuit Delay Computation," *ACM Transactions on Design Automation of Electronic Systems*, July 1997.

[112] S. Aourid, E. Cerny: "CLP-Based Gate Level Timing Verification with Delay Correlation," *IEEE/ACM International Workshop on Logic Synthesis*, Granlibakken, 1997.

[113] S. V. Venkatesh, et al: "Timing Abstraction of Intellectual Property Blocks," *Custom Integrated Circuits Conference*, 1997.

[114] Y. Kukimoto, W. Gosti, A. Saldanha, and R. K. Brayton: "Approximate Timing Analysis of Combinational Circuits under the XBD0 Model," *Proc. IEEE International Conference on Computer-Aided Design*, San Jose, CA, 1997.

[115] M. Sivaraman, A. J. Stojwas: "Timing Analysis Based on Primitive Path Delay Fault Identification," *Proc. IEEE International Conference on Computer-Aided Design*, 1997.

[116] N. Kobayashi, S. Malik: "Delay Abstraction in Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Syatems*, vol. 16, no. 10, pp. 1205-1213, 1997.

[117] Luís Guerra e Silva, João P. Marques Silva, Luís Miguel Silveira and Karem A. Sakallah: "Satisfiability Models and Algorithms for Circuit Delay Computation," *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (Tau)* 1997.

[118] M. Kassab, E. Cerny, S. Aourid and T. Krodel: "Propagation of Last-Transition Time Constraints in Gate-Level Timing Analysis," *Design, Automation and Test in Europe*, pp. 796-802, 1998.

[119] Emily Shriver, Dale Hall, Nevine Nassif, Nadir Rahman, Nick Rethman, Gill Watt, and Jim Farrell: "Timing Verification of the 21264: A 600MHz Full-Custom Microprocessor," *Proceedings of the International Conference on Computer Design*, 1998.

[120] Maroun Kassab, Eduard Cerny, Thomas Krodel, "A Gate-Level Static Timing Verifier Based on Waveform Narrowing," *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (Tau) Four Seasons Hotel, Austin,* TX, pp. 84-90, December 2000.

# APPENDIX A

# Algorithms

## A.1 evaluateConstraintSystem()

This function evaluates the constraint system in an event driven fashion.
**Returns:**

CS_CONSISTENT if the evaluation ended with non-empty domains
CS_INCONSISTENT if the evaluation ended with empty domains

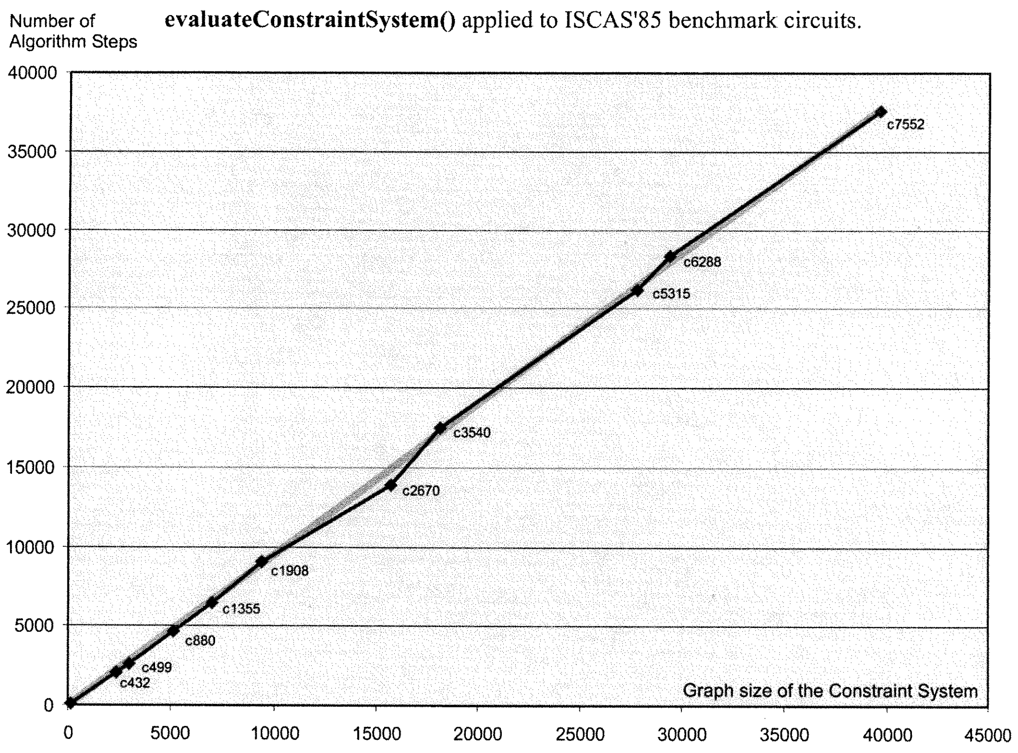
**Notes:**

queue is a global queue for event scheduling

```
evaluateConstraintSystem() {
  // initially queue contains the constraints operating on the domains that were
  // previously changed.
  while (! queue.empty() ){
    constraint = queue.removeFirst();
    apply constraint;
    if ( a domain becomes empty ) return CS_INCONSISTENT;
    schedule the constraints operating on the modified
        domains on queue (if they're not scheduled already);
  }
  return CS_CONSISTENT;
}
```

The worst case time complexity of evaluateConstraintSystem() is $O(N \times 2^B)$ where $N$ is the number of domains, and $B$ is the number of bits used to represent a domain. This worst case may happen when each application of a constraint operator results in narrowing one domain by one time unit, and when the constraint system has circular dependencies like the sequential example in section 3.4.

The worst-case time complexity highly over-estimates the actual execution time of the algorithm. In fact, when component delay correlation is not used, the constraint system has no circular dependencies and the experimental time complexity is virtually linear with the constraint system graph size. Figure 15 shows the experimental time complexity when the algorithm is applied to the ISCAS'85 benchmark circuits.



**Initial domains:**
Inputs: waveforms stable after time 0
Others: all possible values.

Component delay correlation is not used.

**Graph Size:** it is the number of Constraint Operators (vertices) plus the sum of their numbers of inputs (edges).

| Circuit | Graph Size | #Steps |
|---|---|---|
| c432 | 2367 | 2095 |
| c499 | 2988 | 2598 |
| c880 | 5149 | 4620 |
| c1355 | 6956 | 6462 |
| c1908 | 9411 | 9048 |
| c2670 | 15675 | 13915 |
| c3540 | 18115 | 17465 |
| c5315 | 27815 | 26196 |
| c6288 | 29356 | 28328 |
| c7552 | 39581 | 37588 |

***Figure 115***
*Experimental time complexity of the basic WN method on ISCAS'85.*

## A.2 getOrderedReconvergentFanouts(gate)

This function returns an array of reconvergent fan-outs in topological order, deeper structures first.
**returns:**
Array of reconvergent fan-outs
**Notes:**
gate is a primary output (a setup timing check)

```
getOrderedReconvergentFanouts(gate){  // gate is a primary output
  queue.reset();
  queue.insert(gate);
  gate.setProperty(PROP_IS_VISITED);
  while( ! queue.empty() ) { // visit the graph from the output, breadth first to mark the RF
    g = queue.removeFirst();
    for ( fi=0; fi < g.nbLogicFanin(); fi++ ) {
      fanin = g.fanin(fi);
      if ( fanin.hasProperty(PROP_IS_VISITED) ) {
        fanin.setProperty(PROP_IS_RECFANOUT);
      }else{
        fanin.setProperty(PROP_IS_VISITED);
        queue.insert(fanin);
      }
    }
  }
  reconvergent_fanouts.reset(); // to receive the ordered reconvergent fan-outs
  stack.reset();
  stack.push(gate);
  gate.unsetProperty(PROP_IS_VISITED);
  while(!stack.empty()){  // visit the graph to collect the RF in topological order, deepest first
    gg = stack.top();
    depth_sorted_gates.reset(); // to receive the ordered fan-ins of gg
    for ( i=0; i<gg.nbLogicFanin(); i++ ) {
      if ( gg.fanin(i).hasProperty(PROP_IS_VISITED) ) {
        depth_sorted_gates.insert( gg.fanin(i) );
      }
    }
    depth_sorted_gates.sort();  // visited fanin gates are sorted by depth of sub-graph (deepest first)
    for( i=0; i<depth_sorted_gates.size(); i++ ) {
      stack.push(depth_sorted_gates(i));
      depth_sorted_gates(i).unsetProperty(PROP_IS_VISITED);
    }
    if(gg == stack.top()){ // finished visiting its fan-ins
      stack.pop();
      if( gg.hasProperty(PROP_IS_RECFANOUT) ) {
        reconvergent_fanouts.insert(gg);
        gg.unsetProperty(PROP_IS_RECFANOUT);
      }
    }
  }
  return reconvergent_fanouts;
}
```

The worst-case time complexity of getOrderedReconvergentFanouts(gate) is linear with the graph size of the fan-in cone of "gate".

## A.3 doSpatialCorrelation(order, s)

This function performs spatial correlation. It basically perform exhaustive simulation, 'order' nodes at a time on reconvergent fan-outs in topological order, deeper structures first.
**Returns:**

CS_CONSISTENT when the constraint system evaluation ends with non-empty domains
CS_INCONSISTENT when the evaluation ends with empty domains

```
doSpatialCorrelation(order, s){
  rec_fanouts = getOrderedReconvergentFanouts(s)
  unsigned i=0;
  unsigned last;
  do{
    last = min( i + order , rec_fanouts.size() );
    cor_gs.reset();
    for( gi = i; gi < last; gi++ ){
      cor_gs.insert(rec_fanouts_(gi));
    }
    i++;
    if( correlateReconvergentFanouts(cor_gs) == CS_INCONSISTENT){
      return CS_INCONSISTENT;
    }
  }while(last < rec_fanouts.size());
  return CS_CONSISTENT;
}
```

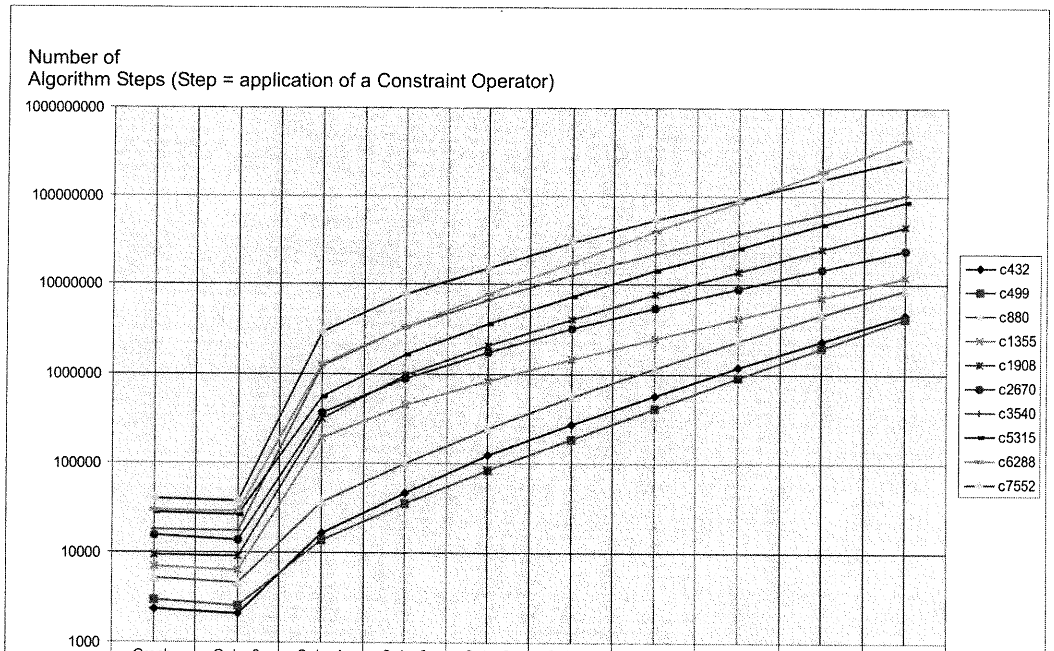
The worst-case time complexity of doSpatialCorrelation(order, s) is

$$O((2^{order} \times (k - order + 1)) \times F)$$

where $k$ is the number of reconvergent fan-outs, and $F$ is the worst-case time complexity of evaluateConstraintSystem(). Since $k \in O(\text{Graph Size})$ and $order \in O(1)$, the spatial correlation worst-case is $O((\text{Graph Size}) \times F)$.

Figure 116 shows the experimental time complexity for the spatial correlation of degrees 1-8 applied to ISCAS'85 circuits.

Figure 117 shows the average number of times a constraint operator is applied when evaluateConstraintSystem is called from within correlateReconvergentFanouts(cor_gs). It is much less expensive that the initial evaluation where all initial internal domains contained all possible values. This confirms a quadratic experimental time complexity for a given correlation order.



| | Graph | Order 0 | Order 1 | Order 2 | Order 3 | Order 4 | Order 5 | Order 6 | Order 7 | Order 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| c432 | 2367 | 2095 | 16562 | 46621 | 123350 | 270809 | 562508 | 1164980 | 2300393 | 4520612 |
| c499 | 2988 | 2598 | 13810 | 35520 | 82023 | 185222 | 410306 | 908142 | 1963933 | 4217005 |
| c880 | 5149 | 4620 | 36185 | 99803 | 237798 | 541836 | 1143818 | 2300068 | 4551995 | 8735885 |
| c1355 | 6956 | 6462 | 191827 | 445394 | 820414 | 1452710 | 2477619 | 4161947 | 7093237 | 12198308 |
| c1908 | 9411 | 9048 | 314694 | 944718 | 2074415 | 4083844 | 7651178 | 13984597 | 25366229 | 45645153 |
| c2670 | 15675 | 13915 | 358160 | 877088 | 1748071 | 3155216 | 5399300 | 8962487 | 14920732 | 24776599 |
| c3540 | 18115 | 17465 | 1198184 | 3257744 | 6700467 | 12667585 | 22487322 | 38295050 | 63371953 | 104003436 |
| c5315 | 27815 | 26196 | 542400 | 1619078 | 3579360 | 7393630 | 14285818 | 26205559 | 48089971 | 87331355 |
| c6288 | 29356 | 28328 | 1283939 | 3265353 | 7656360 | 17636758 | 39737685 | 87431022 | 193831441 | 421383734 |
| c7552 | 39581 | 37588 | 2919051 | 7807019 | 15411856 | 30368697 | 53733612 | 91168710 | 154591938 | 264195874 |

***Figure 116***
*Experimental time complexity of doSpatialCorrelation on ISCAS'85.*
*Graph: the graph size of the constraint system.*
*Order 0: basic method only is applied, no spatial correlation.*
*Order n: basic method is applied, then doSpatialCorrelation(n,s) is called.*

## A.4 correlateReconvergentFanouts(gate_set)

This function does “exhaustive simulation” on the domains of the gates in gate_set.
**Returns:**

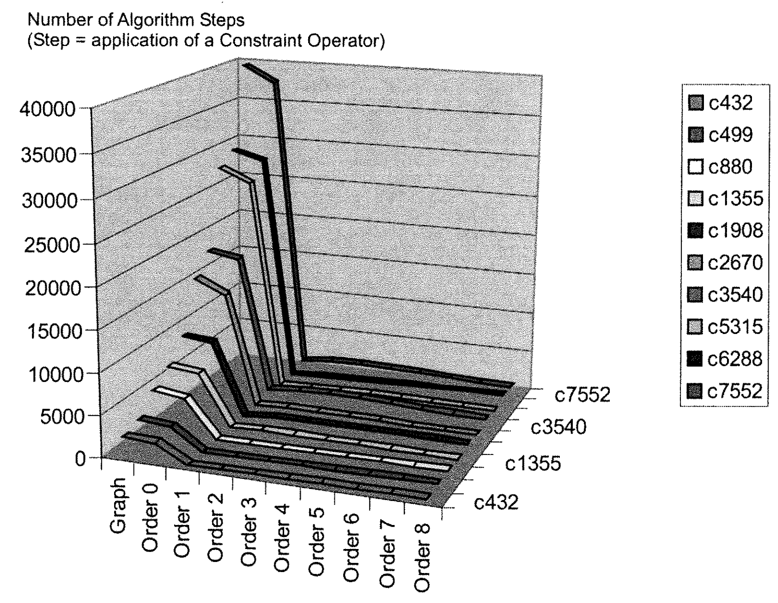
CS_CONSISTENT when the constraint system evaluation ends with non-empty domains
CS_INCONSISTENT when the evaluation ends with empty domains

```
correlateReconvergentFanouts(gate_set){
  int last = gate_set.size() ^ 2;
  save the constraint system state in STATE_1;
  set all domains of STATE_2 to empty; // where to accumulate the results
  bool consistent = false;
  for( i=0; i < last ; i++ ){
    restore domains from STATE_1;
    for( g=0; g<gate_set.size(); g++ ){
      if ( ( i & (g ^ 2) ) == 0 ) {
        remove w1 from domain of gate_set(g);
      }else{
        remove w0 from domain of gate_set(g);
      }
    }
    if(evaluateConstraintSystem() == CS_CONSISTENT){
      accumulate system domains in STATE_2; // union
      consistent = true;
    }
  }
  restore domains from STATE_2;
  if ( consistent ) return CS_CONSISTENT;
  return CS_INCONSISTENT;
}
```

The worst-case time complexity of correlateReconvergentFanouts(gate_set) is

$$O(2^{\text{gate\_set.size()}} \times F)$$

where $F$ is the worst-case time complexity of evaluateConstraintSystem(). Figure 117 shows the experimental time complexity observed for evaluateConstraintSystem() when it is called from within correlateReconvergentFanouts(gate_set).



| | Graph | Order 0 | Order 1 | Order 2 | Order 3 | Order 4 | Order 5 | Order 6 | Order 7 | Order 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| c432 | 2367 | 2095 | 81 | 126 | 174 | 195 | 206 | 216 | 216 | 215 |
| c499 | 2988 | 2598 | 95 | 142 | 174 | 204 | 232 | 262 | 289 | 317 |
| c880 | 5149 | 4620 | 126 | 192 | 237 | 275 | 294 | 299 | 299 | 289 |
| c1355 | 6956 | 6462 | 358 | 425 | 396 | 353 | 303 | 256 | 219 | 189 |
| c1908 | 9411 | 9048 | 397 | 609 | 674 | 667 | 627 | 575 | 523 | 472 |
| c2670 | 15675 | 13915 | 379 | 476 | 480 | 435 | 374 | 311 | 260 | 216 |
| c3540 | 18115 | 17465 | 1020 | 1402 | 1448 | 1372 | 1221 | 1042 | 864 | 710 |
| c5315 | 27815 | 26196 | 320 | 495 | 552 | 573 | 556 | 511 | 469 | 427 |
| c6288 | 29356 | 28328 | 431 | 556 | 656 | 757 | 855 | 941 | 1044 | 1136 |
| c7552 | 39581 | 37588 | 1108 | 1495 | 1481 | 1462 | 1295 | 1100 | 933 | 798 |

***Figure 117***
*Experimental time complexity of each system evaluation of the spatial correlation on ISCAS'85.*
*Graph: the graph size of the constraint system.*
*Order 0: basic method only is applied, before spatial correlation.*
*Order n: number of steps needed to perform evaluateConstraintSystem() when it is called in correlateReconvergentFanouts(gate_set). Note that here it is much less expensive than the initial system evaluation because changes to few domains propagate to a small region of the constraint system.*

## A.5 getTimingDominators( output )

This function determines the Timing Dominators
**Returns:**
    array of timing dominators
**Notes:**
    'output' is a setup timing check, it is actually a timing dominator.

```
getTimingDominators( output ) {
  dominator_set.reset();
  heap.reset();
  dominator.gate = output;
  dominator.lmin = output.lmin();
  heap.insert( dominator );
  while (! heap.empty() ) {
    dominator = heap.removeNodeClosestToOutput();
    if (heap.size() == 0) { // what we removed from heap is a dominator
      dominator_set.insert(dominator);
    }
    for ( i = 0; i < dominator.gate.nbLogicFanin(); i++){
      if ( dominator.gate.fanin( i ).max() >=
                 dominator.lmin - dominator.gate.maxDelay() ){
        // dominator.gate.fanin( i ) is a possible cause of the violation
        if(dominator.gate.fanin( i ) is a primary input) return dominator_set;
        new_dominator.gate = dominator.gate.fanin( i );
        new_dominator.lmin = dominator.lmin - dominator.gate.maxDelay();
        if ( heap contains a node N such that N.gate == new_dominator.gate ){
          if (N.lmin > new_dominator.lmin){
            N.lmin = new_dominator.lmin;
          }
        }else{
          heap.insert( new_dominator );
        }
      }
    }
  }
  return dominator_set;
}
```

The worst-case time complexity of getTimingDominators( output ) is the same as heap sort: $O(n \times \log(n))$ where $n$ is the graph size of the fan-in cone of "output".

## A.6 evaluateConstraintSystemTD( output )

This function evaluates the constraint system and applies narrowing on Timing Dominators
**Returns:**

CS_CONSISTENT when the constraint system evaluation ends with non-empty domains

CS_INCONSISTENT when the evaluation ends with empty domains

```
evaluateConstraintSystemTD( output ) {
  // initially queue contains the constraints operating on the domains that were
  // previously changed.
  if ( queue.empty() ) return CS_CONSISTENT;
  if ( evaluateConstraintSystem() == CS_INCONSISTENT) return CS_INCONSISTENT;
  dominators = getTimingDominators( output );
  for each dominator d of dominators {
    intersect domain of d.gate with $(0|_{d.lmin}^{+\infty}, 1|_{d.lmin}^{+\infty})$
    if domain of d.gate changed then schedule all constraints operating on it on queue;
  }
  return evaluateConstraintSystemTD();
}
```

# APPENDIX B

# Cell Modeling

## B.1 Hard Primitives Used to Model Cell Libraries.

Primitives have **Named** INPUTs, OUTPUTs, INOUTs, and DELAYs.
When used in cell descriptions, terminal names are used to specify how primitives are connected, delay names are used for delay annotation.

**deviceInput**
OUTPUT: Y

**deviceOutput**
INPUT: A

**deviceInOut**
INOUT: A

**vss**
OUTPUT: Y

**vdd**
OUTPUT: Y

**highZ**
OUTPUT: Y

**pullup**
INOUT: A

**pullupDeviceInOut**
INOUT: A

**pulldown**
INOUT: A

**pulldownDeviceInOut**
INOUT: A

**pullupdown**
INOUT: A

**bufif0**
OUTPUT: Y
INPUT: A
INPUT: GZ

**bufif1**
OUTPUT: Y
INPUT: A
INPUT: G

**notif0**
OUTPUT: Y
INPUT: A
INPUT: GZ

**notif1**
OUTPUT: Y
INPUT: A
INPUT: G

**and2**
OUTPUT: Y
INPUT: A
INPUT: B

**and3**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C

**and4**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C
INPUT: D ... **and26**

**or2**
OUTPUT: Y
INPUT: A
INPUT: B

**or3**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C

**or4**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C
INPUT: D ... **or26**

**nand2**
OUTPUT: Y
INPUT: A
INPUT: B

**nand3**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C

**nand4**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C
INPUT: D ... **nand26**

**nor2**
OUTPUT: Y
INPUT: A
INPUT: B

**nor3**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C

**nor4**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C
INPUT: D ... **nor26**

**xor2**
OUTPUT: Y
INPUT: A
INPUT: B

**xor3**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C

**xor4**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C
INPUT: D ... **xor26**

**xnor2**
OUTPUT: Y
INPUT: A
INPUT: B

**xnor3**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C

**xnor4**
OUTPUT: Y
INPUT: A
INPUT: B
INPUT: C
INPUT: D **... xnor26**

**mux2**
OUTPUT: Y
INPUT: S
INPUT: A
INPUT: B

**nMux2**
OUTPUT: Y
INPUT: S
INPUT: A
INPUT: B

**buf**
OUTPUT: Y
INPUT: A

**not**
OUTPUT: Y
INPUT: A

**latchHQ**
OUTPUT: Q
INPUT: C
INPUT: D
DELAY: C_TO_Q
DELAY: D_TO_Q
DELAY: SETUP_HOLD

**latchLQ**
OUTPUT: Q
INPUT: CZ
INPUT: D
DELAY: CZ_TO_Q
DELAY: D_TO_Q
DELAY: SETUP_HOLD

**latchHQZ**
OUTPUT: QZ
INPUT: C
INPUT: D
DELAY: C_TO_QZ
DELAY: D_TO_QZ
DELAY: SETUP_HOLD

**latchLQZ**
OUTPUT: QZ
INPUT: CZ
INPUT: D
DELAY: CZ_TO_QZ
DELAY: D_TO_QZ
DELAY: SETUP_HOLD

**interconnectDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**tristateDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**nonInvertingDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**invertingDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**unknownDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**fs0zRz0FDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**fs0zFz0RDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**fs1zRz1FDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**fs1zFz1RDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**onRoffFDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**onFoffRDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**unknownZDelay**
OUTPUT: Y
INPUT: A
DELAY: A_TO_Y

**dffHQ**
OUTPUT: Q
INPUT: CLK

**dffLQ**
OUTPUT: Q
INPUT: CLKZ

**dummy**
INPUT: A

**reset**
INPUT: Y
INPUT: CLR

**resetZ**
INPUT: Y
INPUT: CLRZ

**preset**
INPUT: Y
INPUT: PRE

**presetZ**
INPUT: Y
INPUT: PREZ

**HsetupHold**
INPUT: CLK
INPUT: D
DELAY: SETUP_HOLD

**LsetupHold**
INPUT: CLKZ
INPUT: D
DELAY: SETUP_HOLD

## B.2 Two Input AND Gate Cell Model Example

The model definition contains:

1) **Header**: cell name, library name, version, date, vendor, program. timescale;
2) **Instance Declaration**: named instances with default delay values when applicable;
3) **Drivers Section**: specifies the driver of each input terminal
4) **SDF Delay Mapping**: how SDF delay annotations are mapped.

Interconnect port delays are the input instances.
IOPATH delays are driven by the interconnect delays.
Primitives at the cell pins are named after the pin names.

For multiple output cells, common logic structures are duplicated to make the
IOPATH delays independent.

```
(STV_LIBDEF
// Header
 (CELL "AN210")
 (LIBRARY "Library Name")
 (VERSION "1.0")
 (DATE "Thu Oct 21 15:36:12 1999")
 (VENDOR "Semi Conductor Vendor")
 (PROGRAM "cdf") // the program that generated this description
 (TIMESCALE 1ns) // time scale for default delays

// Instance Declaration

// PORT delays

 (INSTANCE A INPUT        // INPUT keyword marks the primitive instance as a cell input
  (PRIMITIVE "interconnectDelay")
  (DELAY "A_TO_Y"(0.0:0.0:0.0)) // NULL default
     // instances of primitives with delays specify the default delay values to be used
     // if no SDF annotation is performed
 )
 (INSTANCE B INPUT
  (PRIMITIVE "interconnectDelay")
  (DELAY "A_TO_Y"(0.0:0.0:0.0)) // NULL default
 )

// path delays to Y

 (INSTANCE A_Y
  (PRIMITIVE "nonInvertingDelay")
  (DELAY "A_TO_Y"(0.332246:0.390227:0.582283)(0.280749:0.343123:0.515448))
 )
 (INSTANCE B_Y
  (PRIMITIVE "nonInvertingDelay")
  (DELAY "A_TO_Y"(0.316855:0.378051:0.57481)(0.326919:0.401621:0.595931))
 )

// gates to Y

 (INSTANCE Y OUTPUT     // OUTPUT keyword marks the primitive instance as a cell output
  (PRIMITIVE "and2")
 )
// Drivers Section
 (DRIVERS

// drivers to Y

  (INSTANCE Y
   A(A_Y)B(B_Y)
     // pin "A" of instance "Y" is driven by instance "A_Y"
     // pin "B" of instance "Y" is driven by instance "B_Y"
  )
  (INSTANCE A_Y
   A(A)
  )
  (INSTANCE B_Y
   A(B)
  )
 )

// SDF Delay Mapping
 (TARGET_DELAY "PORT A"
  (INSTANCE A "A_TO_Y")
   // "PORT A" SDF delay annotation goes to instance "A", delay "A_TO_Y"
 )
 (TARGET_DELAY "PORT B"
  (INSTANCE B "A_TO_Y")
 )
 (TARGET_DELAY "IOPATH A Y"
  (INSTANCE A_Y "A_TO_Y")
 )
 (TARGET_DELAY "IOPATH B Y"
  (INSTANCE B_Y "A_TO_Y")
 )
 (TARGET_DELAY "DEVICE Y"
  (INSTANCE A_Y "A_TO_Y")
  (INSTANCE B_Y "A_TO_Y")
 )
 (TARGET_DELAY "DEVICE"
  (INSTANCE A_Y "A_TO_Y")
  (INSTANCE B_Y "A_TO_Y")
 )
)
```

## B.3 FLIP-FLOP Gate Cell Model Example

```
(STV_LIBDEF
 (CELL "FlipFlop")
 (LIBRARY "LibName")
 (VERSION "1.0")
 (DATE "Wed Oct 20 11:40:41 1999")
 (VENDOR "VendorName")
 (PROGRAM "cdf")
 (TIMESCALE 1ns)

// PORT delays

 (INSTANCE CLK INPUT
  (PRIMITIVE "interconnectDelay")
  (DELAY "A_TO_Y"(0.0:0.0:0.0)) // NULL default
 )
 (INSTANCE D INPUT
  (PRIMITIVE "interconnectDelay")
  (DELAY "A_TO_Y"(0.0:0.0:0.0)) // NULL default
 )

// path delays to Q

 (INSTANCE Q OUTPUT
  (PRIMITIVE "nonInvertingDelay")
  (DELAY "A_TO_Y"(0.672096:0.881856:1.351029)(0.758338:1.037886:1.606180))
 )

// gates to Q

 (INSTANCE DFF
  (PRIMITIVE "dffHQ") // positive
 )

// timing check

 (INSTANCE check_D_CLK
  (PRIMITIVE "HsetupHold") // positive
  (DELAY "SETUP_HOLD"(0.660000:0.800000:1.250000)(0.420000:0.420000:0.420000))
 )

 (DRIVERS
  (INSTANCE Q
   A(DFF)
  )
  (INSTANCE DFF
   CLK(CLK)
```

```
    )
    (INSTANCE check_D_CLK
      CLK(CLK)
      D(D)
    )
  )
  (TARGET_DELAY "PORT D"
    (INSTANCE D "A_TO_Y")
  )
  (TARGET_DELAY "PORT CLK"
    (INSTANCE CLK "A_TO_Y")
  )
  (TARGET_DELAY "IOPATH CLK Q"
    (INSTANCE Q "A_TO_Y")
  )
  (TARGET_DELAY "DEVICE Q"
    (INSTANCE Q "A_TO_Y")
  )
  (TARGET_DELAY "DEVICE"
    (INSTANCE Q "A_TO_Y")
  )
  (TARGET_DELAY "SETUPHOLD D CLK"
    (INSTANCE check_D_CLK "SETUP_HOLD")
  )
)
```