University of Balearic Islands
Department of Physics
Electronic Systems Group

PhD. Thesis

An advanced Framework for efficient IC optimization based on analytical models engine

Author: Salvador Barceló Adrover
Director: Dr. Jaume Segura

Palma de Mallorca, 2013

Esta tesis doctoral ha sido realizada por el Sr. Salvador Barceló Adrover bajo la dirección del Dr. Jaume Segura Fuster, del Departamento de Física de la Universidad de las Islas Baleares (UIB).

Palma de Mallorca, 6 de noviembre de 2013

Dr. Jaume Segura Fuster

El director de tesis

Salvador Barceló Adrover

El doctorando

La realización de esta tesis doctoral ha estado financiada por el proyecto TEC20011-25017 del Ministerio de Ciencia e Innovación y ha recibido una ayuda para apoyar a grupos de investigación competitivos de la Consejería de Educación, Cultura y Universidades del Gobierno Balear (2011-2013), patrocinada por la Comunidad Autónoma de las Illes Balears y cofinanciada con fondos FEDER. También ha contado con una beca de formación de personal investigador (FPI) del Ministerio de Ciencia e Innovación (BES-2006-11943).

GOVERN DE LES ILLES BALEARS

MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

FEDER Fondo Europeo de Desarrollo Regional

Resumen de la tesis

El escalado de la tecnología CMOS ofrece grandes ventajas tales como el aumento de la densidad de integración o la frecuencia de operación a la par que proporciona una reducción del consumo de energía y el coste por transistor. Esta carrera por la integración genera nuevos desafíos relacionados con efectos adversos, algunos son de nueva aparición mientras que otros se agravan respecto a nodos anteriores . Uno de tales efectos adversos de importancia creciente es el impacto de los llamados Single Event Transient o SET . Dado que el escalado de la tecnología reduce la capacidad parásita de los nodos internos, la tensión de alimentación y el retardo de puerta, la importancia relativa del efecto de las partículas ionizantes aumenta debido a que la misma cantidad de carga es capaz de inducir un transitorio de tensión con mayor probabilidad de propagarse dentro del circuito.

Además de las técnicas basadas en redundancia, muchas soluciones de mitigación se basan en el aumento de la robustez intrínseca del circuito frente a eventos transitorios. Aunque estas estrategias no pueden evitar el problema completamente, reducen su impacto hasta límites aceptables dependiendo del ámbito de aplicación del circuito.

En base a estos desafíos, la presente tesis desarrolla y analiza un conjunto de herramientas orientadas a la evaluación de la sensibilidad a la propagación de eventos SET en circuitos microelectrónicos. Las herramientas de procesamiento son capaces de manejar circuitos con una alta complejidad de forma eficiente.

Después de desarrollar un sistema compacto de lógica específica que mejora el rendimiento de los algoritmos construidos para propagar las transiciones dentro del circuito y manejar de forma eficiente la propagación de SETs, se presentan diversas técnicas de simplificación, partición y encapsulación de bloques en circuitos.

Se ha desarrollado un algoritmo eficiente de búsqueda de caminos sensibilizables cuya eficacia se ha demostrado sobre circuitos tipo benchmark de tamaño considerable. Se ha demostrado también que el tiempo de retardo de un camino dado depende de los vectores de sensibilización aplicados a las puertas complejas que forman parte del mismo. En algunos casos, la variación de retardo debida a los diferentes vectores de sensibilización es comparable a las variaciones de retardo

atribuibles a las variaciones paramétricas del proceso.

El motor del algoritmo de análisis de SET a nivel de circuito lo constituye un modelo analítico de propagación de SET a nivel de puerta lógica (desarrollado en el marco de otra tesis doctoral en desarrollo) implementado mediante un ajuste polinómico. Los coeficientes del modelo son extraídos para cada tecnología a partir del tratamiento automatizado de la biblioteca de puertas lógicas correspondiente. El proceso de extracción determina el conjunto de parámetros óptimo para cada puerta de la biblioteca, constituyendo un proceso clave para una estimación precisa de la propagación de SETs a nivel de circuito.

Después de verificar la descripción de la propagación de SET a través de un trabajo exhaustivo de simulación a nivel eléctrico, utilizando circuitos de tipo benchmark sintetizados en tecnologías CMOS comerciales, se han propuesto varias métricas de propagación de SETs considerando el impacto de los enmascaramientos lógico, eléctrico y combinado lógico-eléctrico. Estas métricas proporcionan una vía de análisis para cuantificar tanto las regiones que son más susceptibles a propagar eventos SET hacia las salidas como el conjunto de salidas más susceptibles de producir eventos SET.

La aplicación de la herramienta a circuitos tipo benchmark grandes han demostrado la capacidad del conjunto de herramientas en el ámbito de la estimación de la propagación de SETs. En base a los indicadores desarrollados, la herramienta es capaz de identificar la lista de nodos internos que son más susceptibles a propagar un SET, considerando tanto los efectos de enmascaramiento eléctrico como lógico. Los resultados pueden ser ponderados por la probabilidad lógica de que un cierto camino se encuentre sensibilizado a partir de los vectores de entrada del circuito . Del mismo modo, la herramienta también proporciona información acerca de los nodos de salida del circuito con una mayor probabilidad de producir un SET en ambientes de radiación específicos.

Un análisis adicional permite determinar exhaustivamente el efecto de ensanchamiento/filtrado del pulso inyectado de forma exhaustiva en todos los nodos del circuito.

La aplicación exhaustiva de la herramienta al análisis de los circuitos tipo benchmark grandes demuestra la viabilidad de las mismas para el tratamiento de grandes circuitos obteniendo la información de interés.

En numerosos casos, el conjunto de algoritmos desarrollados han proporcionado mejores resultados que las herramientas comerciales consideradas estándar en ámbitos industriales.

Abstract

CMOS IC technology scaling provides many advantages like integration density increase, higher operating frequency while providing reduced power consumption and cost per transistor. Such integration race generates new challenges related to adverse side effects some of them being new, while others are exacerbated from previous technology nodes. One of such adverse effect that grows in importance is the impact of Single Event Transients (SETs). As technology scaling reduces the parasitic capacitance, supply voltage and the gate delay, the relative importance of ionizing particles increases because the same amount of charge is capable of inducing a voltage transient that may propagate within the circuit.

Apart of redundancy-based techniques, many mitigation solutions are based on increasing the circuit intrinsic robustness to soft-error effects. Although these strategies cannot avoid the problem completely, they reduce the soft-error impact to an acceptable limit that depends on the circuit application.

Based on these challenges, this thesis develops and evaluates a complete framework for SET propagation sensitivity. The framework comprises a number of processing tools capable of handling circuits with high complexity in an efficient way.

After developing a compact specific logic system to enhance the performance of the algorithms constructed to propagate transitions within the circuit and handle efficiently SET propagation, various simplification, partitioning and encapsulation techniques have been detailed and analyzed to enhance the overall framework operation.

A quite efficient true path finding algorithm has been constructed and its efficacy demonstrated on large benchmark circuits. It has been also shown that the delay of a given path depends on the sensitization vectors applied to the complex library gates within the path. In some cases, the delay variation due to different sensitization vectors is comparable to the path delay caused by process parameters variations. Such an improvement over the path delay computation, links such delay estimation to the specific sensitization vector and to the verification of the path being a true path, representing a significant improvement over commercial tools.

The framework developed engine is an SET analytical propagation model (developed within

another Ph.D. thesis under development) incorporated as a polynomial implementation. Polynomial coefficients are extracted for each technology by automatically processing the associated gate library. An optimal parameter set is obtained for each library gate, being a key process for accurate SET propagation estimation.

After verifying the SET propagation description through extensive electrical simulations over benchmark circuits synthesized on commercial CMOS technologies, various SET propagation metrics have been proposed considering the impact of logic masking, electric masking and combined logic-electric masking. Such metrics provide a valuable vehicle to grade either in-circuit regions being more susceptible of propagating SET events toward the circuit outputs or circuit outputs more susceptible to produce SET events.

The tool application to large benchmark circuits has shown the framework capabilities in the SET propagation estimation domain. Based on the developed metrics, the tool is capable of identifying the list of circuit internal nodes most suitable to propagate an SET accounting for both the electrical and logical masking effects. Results can be weighted by the logic probability of a node being activated from the circuit input vectors. Similarly, the tool also provides information about the circuit output nodes with a higher probability of producing an SET under specific radiation environments. An additional tool analysis determines exhaustively the effect of pulse broadening/filtering once a specific SET event is induced at each circuit node.

Exhaustive application to large benchmark circuits demonstrates the framework feasibility to treat huge circuits providing the parameters of interest. In many instances, the developed framework has been shown to output better results than industry-standard commercial tools.

Index

Chapter 1: Introduction....1
1.1.Motivation and objectives....2
1.2.Document organization....4

Chapter 2: Timing analysis and SET propagation....5
2.1.Timing analysis....5
2.2.Delay modeling....13
2.2.1.Empirical modeling....14
2.2.1.1. Lookup Table (LUT)....14
2.2.1.2. Polynomial model....15
2.2.2.Extraction process....16
2.2.3.Multivariable polynomial model....17
2.2.4.Computational advantages....18
2.3.Effective capacitance....19
2.3.1.Capacitance components....19
2.3.1.1. Transistor-level components....20
2.4.SET Propagation....23

Chapter 3: Framework core elements....29
3.1.Definitions....29
3.2.Logic System....32
3.2.1.Theoretical concepts....32
3.2.2.State of the art....33
3.2.3.Developed Logic System....34
3.2.4.Dual logic system....38
3.3.Basic operations....41
3.3.1.Sensitization....41
3.3.2.Implication....46
3.3.3.Justification....48
3.4.Sensitization Algorithms....51
3.4.1.Stepwise algorithm....51
3.4.1.1. Branch point....54
3.4.1.2. Main routine....56
3.4.1.3. Implication....57
3.4.1.4. Justification....59
3.4.2.Full path algorithm....63
3.4.3.Path graph creation....67

Chapter 4: Preprocessing techniques and Framework structure....69
4.1.Simplification techniques....70
4.2.Partitioning techniques....70
4.2.1.Auxiliary routines....70

4.2.1.1. Touch related nodes algorithm....70
4.2.2.Separate independent sections....71
4.2.3.Partitioning by Output....73
4.2.4.Partitioning by Input....74
4.2.5.Partitioning by Input-Output....76
4.3.Encapsulation techniques....81
4.3.1.Principle of the technique....81
4.3.2.Specific techniques....83
4.3.2.1. Gates with special input configuration....83
4.3.2.2. Non-Stem nodes....85
4.3.2.3. Reconvergent fan-out....86
4.3.2.4. Single input gates....88
4.3.2.5. Gates sharing all inputs nodes....89
4.3.2.6. Gates partially sharing input nodes....90
4.4.Application....92
4.4.1.Identification of repeated structures....92
4.4.2.Path enumeration....95
4.5.Results....97
4.6.Framework....102
4.6.1.Cell manager module....103
4.6.2.Circuit analyzer....105
4.6.3.Circuit preprocessor....105
4.6.4.Path searching engine....105
4.6.5.Path analyzer....106
4.6.6.External tool interpreter....107

Chapter 5: Framework application to Timing Analysis....109
5.1.Polynomial Delay Model....110
5.1.1.Multidimensional extraction process....113
5.1.2.Gate simulation process....117
5.1.3.Delay model accuracy verification....121
5.2.Effective capacitance extraction....123
5.2.1.Delay-based effective capacitance extraction....125
5.3.Delay dependency with sensitization vector....127
5.3.1.Gate-level analysis....128
5.3.2.Transistor level analysis....133
5.3.3.Circuit-level relevancy....136
5.3.4.Sensitization vector impact on timing analysis....137
5.3.4.1. Test circuit....137
5.3.4.2. Benchmark circuits....139
5.3.4.3. Relevance and comparison to other effects....141
5.4.Algorithms for Timing Analysis....144
5.4.1.Exhaustive path identification....144
5.4.2.Timing estimation using path graph....149
5.4.3.Timing estimation using path graph and simplification techniques....152

Chapter 6: Framework application to SET propagation estimation (SENSET).....157
6.1.SET Propagation model.....158
6.1.1.Model parameters extraction.....159
6.2.SET Propagation Sensitivity.....162
6.2.1.Logic sensitivity.....163
6.2.2.Electrical sensitivity.....166
6.2.3.Circuit level SET sensitivity metrics.....167
Node SET Sensitivity (NSS).....167
Output SET sensitivity (OSS).....169
SET Output width distribution.....170
6.3.SET through reconvergence.....171
6.4.Results.....173
6.4.1.Tool Accuracy.....173
6.4.2.SENSET Analysis.....175
6.4.2.1. Output sensitivity.....175
6.4.2.2. Node SET sensitivity.....179
6.4.2.3. Output width distribution.....183

Chapter 7: Conclusions and future work.....191

References.....193

A. Benchmark circuits.....202

Index of Figures

Figure 2.1: General circuit structure....6
Figure 2.2: Timing diagram....8
Figure 2.3: Transition delay....14
Figure 2.4: Propagation delay....14
Figure 2.5: MOSFET parasitic capacitances....20
Figure 2.6: Inverter capacitances....22
Figure 2.7: Logic masking....25
Figure 2.8: Time masking....25
Figure 3.1: 2-bit carry bypass adder....31
Figure 3.2: Hasse diagram of 10-valued logic system [33]....34
Figure 3.3: Semi-undetermined value....35
Figure 3.4: Hasse diagram....37
Figure 3.5: Static-hazard....37
Figure 3.6: Hasse diagram of dual value logic system....39
Figure 3.7: Logic system example....40
Figure 3.8: Minimum sensitization conditions....43
Figure 3.9: Forward implication....47
Figure 3.10: Backward implication....47
Figure 3.11: Mixing values....48
Figure 3.12: Static glitch....49
Figure 3.13: Stepwise algorithm....52
Figure 3.14: Sensitization graph....53
Figure 3.15: Branch point....55
Figure 3.16: Setting values....57
Figure 3.17: Implication flowchart....58
Figure 3.18: Justification algorithm flowchart....60
Figure 3.19: Flowchart of full path sensitization algorithm....64
Figure 3.20: Full path example....64
Figure 3.21: Flowchart of multi-option verification....65
Figure 3.22: Flowchart of multi-option assignment....66
Figure 3.23: Non-divergent In-Out graph....67
Figure 3.24: Path graph from input....68
Figure 3.25: Path graph from output....68
Figure 4.1: Touch nodes algorithm....71
Figure 4.2: Independent sections....72
Figure 4.3: Partitioning by Output....74
Figure 4.4: Partitioning by input....75
Figure 4.5: Partitioning by Input-Output....77
Figure 4.6: Justification tree 1....79
Figure 4.7: Justification tree 2....79
Figure 4.8: Pre-sensitization....80
Figure 4.9: Subcircuit encapsulation....82
Figure 4.10: Redundant inputs gate....83
Figure 4.11: XNOR2 implemented with a multiplexer....84
Figure 4.12: Non-stem nodes....85
Figure 4.13: Reconvergence....87
Figure 4.14: Single input gate....89
Figure 4.15: Shared input gates....90

Figure 4.16: Partially shared input nodes....91
Figure 4.17: Example circuit....95
Figure 4.18: Path graph before simplification....96
Figure 4.19: Simplified path graph....96
Figure 4.20: General Framework Structure....102
Figure 4.21: Cell manager diagram....104
Figure 5.1: Rising input propagation delay variation....111
Figure 5.2: Rising output transition time....111
Figure 5.3: Flowchart of the model extraction algorithm....116
Figure 5.4: Gate delay simulation....117
Figure 5.5: Transition generation....118
Figure 5.6: Real transition....119
Figure 5.7: Mean error and number of coefficients....119
Figure 5.8: Mean error and number of coefficients....120
Figure 5.9: C extraction circuit....126
Figure 5.10: Gate AO22....128
Figure 5.11: Gate OA12....128
Figure 5.12: Gate CB4I6....129
Figure 5.13: Gate AOI212....129
Figure 5.14: Gate AO22 transistor-level schematic....134
Figure 5.15: Gate OA12 transistor-level schematic....134
Figure 5.16: Internal currents of AO22....136
Figure 5.17: Test circuit....138
Figure 5.18: Example circuit....149
Figure 5.19: Circuit graph....150
Figure 5.20: Path graph....151
Figure 5.21: Algorithm flowchart....152
Figure 6.1: SET pulse characteristics....158
Figure 6.2: SET model extraction circuit....160
Figure 6.3: Example circuit....164
Figure 6.4: Graph of example circuit....164
Figure 6.5: Electrical propagation....166
Figure 6.6: Reconvergence....171
Figure 6.7: SET propagation through reconvergence....172
Figure 6.8: Output sensitivity c3540 (100ps)....175
Figure 6.9: Output sensitivity c3540 (150ps)....176
Figure 6.10: Output sensitivity c5315 (100ps)....177
Figure 6.11: Output sensitivity c5315 (150ps)....177
Figure 6.12: Output sensitivity c7552 (100ps)....178
Figure 6.13: Output sensitivity c7552 (150ps)....178
Figure 6.14: c3540 Minimum pulse width....179
Figure 6.15: c5315 Minimum width....180
Figure 6.16: c7552 Minimum width....180
Figure 6.17: c3540 Minimum pulse height....181
Figure 6.18: c5315 Minimum height....181
Figure 6.19: c7552 Minimum Height....181
Figure 6.20: c3540 Internal node Electrical & Logic sensitivity....182
Figure 6.21: c5315 Internal node Electrical & Logic sensitivity....182
Figure 6.22: c7552 Internal node Electrical & Logic Sensitivity....183
Figure 6.23: c3540 Output width distribution (100ps)....184

Figure 6.24: c3540 Output width distribution (150ps).....184
Figure 6.25: c5315 output width distribution (100ps).....185
Figure 6.26: c5315 Output width distribution (150ps).....185
Figure 6.27: c7552 Output width distribution (100ps).....186
Figure 6.28: c7552 Output width distribution (150ps).....186
Figure 6.29: c3540 Injected-Output width.....187
Figure 6.30: c3540 Injected-Output width.....188
Figure 6.31: c3540 Injected-Output width.....188
Figure 6.32: c3540 Injected-Output width.....189
Figure 6.33: c7552 Injected-Output width.....189
Figure 6.34: c7552 Injected-Output width.....190

Index of Tables

Table 2.1: Capacitance components....21
Table 3.1: 10-valued logic system....34
Table 3.2: Initial set of values....35
Table 3.3: Not gate propagation....35
Table 3.4: Or Propagation....36
Table 3.5: And Propagation....36
Table 3.6: Dual logic system initial values....38
Table 3.7: Composite dual values....38
Table 3.8: Minimum sensitization condition....43
Table 3.9: Example gates....44
Table 3.10: AND3 steady values sensitization....44
Table 3.11: AND3 Minimum condition sensitization....44
Table 3.12: OA12 Steady values sensitization....45
Table 3.13: OA12 Relaxed steady value sensitization....45
Table 3.14: OA12 Minimum condition sensitization....45
Table 3.15: XOR2 sensitization table....46
Table 3.16: Justification table. Steady values....49
Table 3.17: NAND2 justification options for semi-undetermined values....50
Table 3.18: NAND2 justification options for transitions....50
Table 3.19: Specific functions for reversible algorithm....53
Table 3.20: Justification table for OA12....59
Table 3.21: Justification table for AND2 gate....59
Table 4.1: Independent sections....73
Table 4.2: Redundant input gates....85
Table 4.3: Block repetition after non-stem simplification....93
Table 4.4: Block repetition examples....94
Table 4.5: #Instances for iterative simplifications of b17....94
Table 4.6: ISCAS c6288....98
Table 4.7: ISCAS c7552....98
Table 4.8: ITC'99 b17....98
Table 4.9: ITC'99 b20....99
Table 4.10: ITC'99 b21....99
Table 4.11: Path graph creation....100
Table 4.12: Structural paths enumeration....101
Table 5.1: Delay comparison vs electrical simulation (130nm)....122
Table 5.2: Delay comparison vs electrical simulations (90nm)....122
Table 5.3: Delay comparison vs electrical simulations (65nm)....122
Table 5.4: RC Constant capacitance extraction....124
Table 5.5: AO22 Propagation Table....130
Table 5.6: OA12 Propagation Table....130
Table 5.7: CB4I6 Propagation Table....131
Table 5.8: AOI212 Propagation Table....131
Table 5.9: Propagation delay variation for AO22....132
Table 5.10: Propagation delay variation for OA12....132
Table 5.11: Propagation delay variation for CB4I6....132
Table 5.12: Propagation delay variation for AOI212....133
Table 5.13: Circuit-level multi-sensitization impact....137
Table 5.14: Delay vs Input vector for the simple circuit in Fig. 5.17....139

Table 5.15: Sensitization vector impact on critical path identification....140
Table 5.16: Path delay variation....141
Table 5.17: Complex gates per path....143
Table 5.18: Backtrack limit influence for ISCAS c7752....145
Table 5.19: Forward algorithm. Criterion 1....146
Table 5.20: Forward algorithm criterion 2....146
Table 5.21: Forward algorithm criterion 3....146
Table 5.22: Backward algorithm Criterion 1....147
Table 5.23: Backward algorithm Criterion 2....147
Table 5.24: Backward algorithm Criterion 2....147
Table 5.25: Forward algorithm with justification at end Criterion 1....148
Table 5.26: Forward algorithm with justification at end Criterion 2....148
Table 5.27: Forward algorithm with justification at end Criterion 3....148
Table 5.28: Critical path identification (preprocessing)....153
Table 5.29: Slowest true path identification....154
Table 6.1: Most sensitizable paths....174
Table 6.2: Random selected paths....174
Table A.1: ISCAS'85 Benchmark circuits....203
Table A.2: ITC'99 Benchmark circuits....204

Chapter 1: Introduction

With no doubt, the integrated electronic technology industry is advancing extremely fast. Even with the CMOS technology dead end predictions [1], the MOSFET transistor dimensions are shrinking each year and the number of transistors integrated within a die increases constantly [2], as was predicted by the Moore's law in 1965 [3]. This has lead to spectacular figures like some of today's commercial circuits exceeding one billion transistors integrated together in the same piece of silicon [4]. Such an evolution reduces the design margins, magnifies some side effects that were negligible in previous technologies while new adverse physical phenomena come into the picture. All these events increase the relevance of design improvement, circuit verification at multiple design stages, and testing, to ensure that a given circuit meets all operation constraints required. However, this task becomes more and more challenging due to the ever-increasing overall complexity.

Design-for-testability (DFT) is a valuable vehicle in making test complexity manageable, but even with the aid of such techniques some specific circuit verifications remain unaffordable. An advanced test and verification plan is essential given its significant economic impact on the final circuit cost. Other vital stages of the design flow are related to circuit optimization: an overall die area reduction improves the manufacturing yield reducing the cost per circuit, while power emerged as a key technology scaling restraining parameter due to thermal issues in high-end applications, and a limiting parameter of portable devices that require a very low power circuits to maximize the battery life.

One of the physical mechanisms that threaten current and future technology nodes reliability is

the impact of Single Event Effects (SEEs) [5]. As technology scaling reduces the circuit nodes parasitic capacitance, lowers the supply voltage and shrinks gate delay, the relative impact of ionizing particles increases because the same amount of injected charge is capable of inducing a voltage transient that may propagate within the circuit and/or induce a memory upset.

The impact of ionizing radiation on circuits behavior has been an issue deeply studied specially for circuits operating in hostile environments with high radiation levels, specially aerospace applications lacking the protection against cosmic radiation provided by the atmosphere and the earth magnetic field. Traditionally, radiation-hardening techniques were adopted almost exclusively for applications running in hostile environments. The high sensitivity to radiation of today technologies has shifted this view since commercial devices are susceptible of being affected by ionizing radiation even at sea level due to the technology miniaturization. This trend is motivating the adoption of procedures to consider the soft-error susceptibility caused by particle impacts within the design flow of current consumer electronic circuits.

The main impact of soft-errors caused by ionizing particles affected traditionally circuit memory subsystems. The adoption of circuit redundancy and error correction codes (ECC) for critical memory systems has accomplished maintaining the soft-error rate (SER) associated to memory elements within tolerable limits despite the technology evolution. However, the SER associated to the combinational logic has experimented a considerable increase with technology scaling, since their impact is favored by such scaling.

Apart of redundancy-based techniques, many mitigation solutions are based on increasing the circuit intrinsic robustness to SEEs. Although these strategies cannot avoid the problem completely, they reduce the soft-error impact to an acceptable limit that depends on the final application.

1.1. Motivation and objectives

The growing impact of transient effects caused by radiation phenomena in combinational circuits has motivated an increasing interest in the development of efficient Single Event Transients (SET) description and mitigation techniques. Although the basic mechanisms governing SET propagation within combinational blocs have been extensively studied and are well known, the development of efficient propagation models suitable for nanometer technologies is of enormous interest nowadays. SET modeling has an inherent difficulty related to the complexity of describing the propagation of a non-purely digital perturbation within large or complex digital blocs where electrical-level

descriptions are not suitable. SET propagation descriptions must accomplish various conditions for them to be efficiently adopted as valid descriptions in industrial environments [6] including accuracy, compactness and simplicity.

However, although efficient compact models are available [7], their adoption by the research community is conditioned to their practical application within circuit-level analysis in an affordable way. In this context, the development of efficient EDA tools capable of covering the gap between the gate level model verification – typically accomplished through electrical-level simulations not suitable for large blocks – and the realistic complex circuit domain is lacking. Such a framework is capital to advance in the overall application and evaluation of SET mitigation techniques, as they require quick and efficient ways of evaluating various circuit alternatives, as well as being capable of determining the best option between different solutions.

Such achievement is complex since it is not a merely implementation of a given analytical model within a tool given the complexity of today IC designs. A practical solution requires an efficient implementation of a complete framework capable of tracing true paths within a circuit, accurately accounting for the propagation delay, efficiently managing complex circuits and providing a powerful information about circuit SET sensitivity according to various design abstraction levels.

The creation of such a framework is the focus of the work developed in this thesis, exploiting the benefits of compact modeling descriptions developed within the research group where this thesis has been developed. Although the main objective is focused on developing specific SET propagation analysis tools, when integrating such components within existing commercial tools, secondary objectives have been found as the work has been carried over. The low efficiency and low accuracy of some commercial tool modules have motivated the development of specific framework elements oriented to efficiently computing standard tasks like efficient true path enumeration, efficient gate and path delay computation and efficient handling of highly complex circuits.

The final objective is to provide the circuit designer with a valuable tool to analyze the circuit SET sensitivity in terms of SET propagation for various design abstraction levels. The framework must be capable of being used either by gate-level design and/or synthesis applications as well as by block-level integration designers and tools. In this way, the framework must be capable of providing in-circuit information by grading internal nodes in terms of their SET sensitivity, as well as detailing block-level analysis when treating the circuit as a box.

Following the established tool evaluation methods, the framework is aimed to be validated through electrical-level simulations applied through standardized benchmark circuits synthesized on a wide set of commercial and open-source technologies. Application on large benchmark circuits will allow evaluating the framework capabilities compared to “de facto” standardized commercial tools.

1.2. Document organization

This work is divided in seven chapters organized as follows:

Chapter 2 provides a general view of the main topics covered in this thesis. It introduces basic concepts about timing analysis and SET propagation through a combinational circuit, and some theoretical foundations relative to analytical modeling techniques and logic gates capacitances.

In chapters 3 and 4 are detailed the algorithms and techniques included in the framework for combinational circuits processing.

Chapter 3 is focused on identifying paths capable to propagate a transition through a combinational logic block. It starts by defining the basic concepts about paths through a circuit and then explains the new logic system developed for true path identification. The algorithms for path identification are described step by step detailing each individual task.

Chapter 4 presents solutions for the limitations of the algorithms of chapter 3 when are applied to very large circuits, and explains a set of circuit simplification techniques to reduce the circuit complexity allowing to process complex circuit design in a reasonable time.

In chapters 5 and 6 show the application of the framework developed to solve two key tasks for a proper design flow of a reliable digital circuit.

Chapter 5 presents an analytical delay model based on the mathematical concepts introduced in Chapter 2, and its application in combination with the path identification techniques to timing analysis.

Chapter 6 presents how the framework components are applied in combination with an analytical SET propagation model to estimate the SET propagation capability of a combinational circuit providing SET propagation sensitivity metrics. This metrics may help to improve the designs tolerance to radiation induced effects.

Finally, chapter 7 exposes the conclusions drawn from this work and the future work.

Chapter 2: Timing analysis and SET propagation

This chapter provides a global insight of basic concepts discussed in detail in the following chapters, with the objective of establishing the foundations of the work developed in this thesis. The chapter starts presenting the essential concepts related to timing analysis being a key step in the design of a synchronous digital circuit. It follows with the mathematical theory on which the analytical modeling technique implementation used in this work is based. Finally the basics about the SET propagation through a combination block are introduced.

2.1. Timing analysis

Timing analysis is a key step in the design flow of synchronous digital circuits, validating the proper timing performance of a circuit design [8]. Its significance and complexity increases with technology scaling due to new physical phenomena appearing in nanometer technologies [9][10] and the increase in integration density.

A synchronous digital circuit is intended to operate at a given clock rate and timing analysis is responsible to verify if the combinational blocks delays meet the timing constraints imposed by the system clock frequency and the characteristics of sequential elements. Theoretically, such timing constraints can be verified through a detailed circuit simulation, but such simulation are too slow that in practice remain completely unaffordable for large circuit designs due to their excessive computational resources requirements. Therefore the timing analysis is performed using simplified delay models seeking a balance between accuracy and computation time.

Circuit synthesis is performed according to multiple constraints set by the designer like area, power and timing. Timing analysis is used to guide the synthesis selecting the proper logic gates to implement the expected logic function accomplishing the design constraints. However, timing constraints remains among the most important design constraint since if they are not meet, the circuit is unable to operate correctly at intended clock frequency.

The timing analysis objective during the design flow is to ensure that the correct logic value will be present at the data input of each sequential element when the clock edge arrives, allowing that the memory elements capture the correct logic values.

Fig. 2.1 shows a generic structure of a sequential circuit where a combinational logic block is located between two sets of latching elements, the input latches and the output latches, both controlled by a clock signal (*Clk*). Input latches apply a logic vector at the combinational block inputs keeping these values stable during one clock cycle. Output latches capture the logic values arriving at the block outputs, their outputs constitute the inputs to the following combinational block (not shown in the Figure). To guarantee correct circuit operation, the circuit response to an input vector at a given clock edge must provide valid stable values at data input of the output latches before the next clock edge arrives.

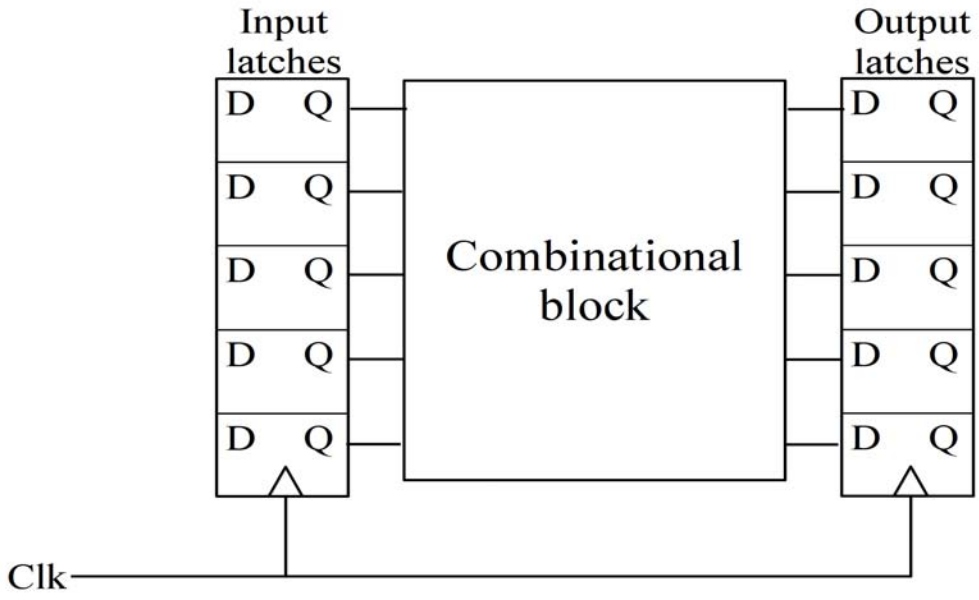


Figure 2.1: General circuit structure

Fig. 2.2 shows a timing diagram to illustrate the circuit operation. *FFI* is one of the input latches and *FFO* is an output latch. T_{Clk} is the clock period, i.e. time interval between two consecutive clock

edges, therefore the operating frequency is $f = \frac{1}{T_{Clk}}$.

The sequence of events represented in Fig. 2 can be divided in three steps as follow:

(1) A clock edge arrival latches the data at the input D of the FFI (FFI/D) placing the captured data at its output Q (FFI/Q). The time required by a latch to set the captured value at the output Q after the triggering clock edge is denoted as t_{dff}, i.e. the delay of the memory element. The time instant at which the data is placed at the inputs of the combinational block is referred as *launch time*.

(2) Once the logic values are stable at the inputs of the combinational block at *launch time*, the change is propagated through the logic until the outputs of the combinational block. The *arrival time* is the time instant at which the outputs of the combinational block take the correct logic value. The difference between *arrival time* and the *launch time* is the delay of the combinational logic block.

(3) The next clock edge triggers the capture of the output values, however the latches require that the data to be captured be stable at input D before the clock edge arrival. The amount of time the data must be stable before the clock edge arrival is called *setup time* (t_{setup}) and depends on the specific latch characteristics. Correct data must be stable at the data input of the latches the *setup time* before the clock edge, this instant is referred as *required time*, i.e. is the instant at which the correct logic must show up at the combinational block outputs for proper operation. After the clock edge the data must remain stable at the inputs of the latches an amount of time called *hold time* (t_{hold}) to be properly captured.

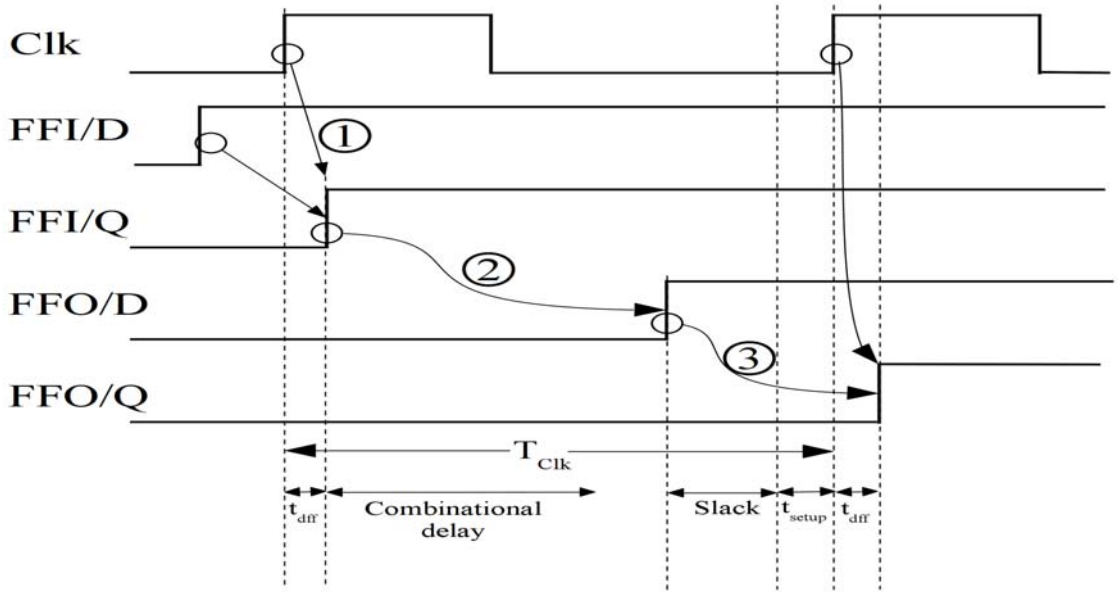


Figure 2.2: Timing diagram

The time between the *arrival time* and the *required time* is called *slack (2.1)*, this is the amount of time by which the timing constraint is met.

$$slack = required\ time - arrival\ time \tag{2.1}$$

If the *slack* is positive, as shows the example of Fig. 2.2, then the signal arrives earlier than the *required time*, and therefore timing constraint is met. However if the *slack* is negative then the signal arrives later than the *required time*, producing a timing constraint violation. If the *slack* is exactly zero then the timing constraint is theoretically met, however in the practice the *slack* must be greater than zero to ensure the proper operation, accounting for parameter variations and timing inaccuracies.

This type of timing check is referred as setup time check, although it is not the unique timing check performed by a timing analysis tool. Timing checks usually performed during the timing analysis are:

- Setup time check: Verifies that the data arrives soon enough before the clock edge to be correctly captured, i.e. the signal is stable at the outputs before the *required time*. This is the most common type of timing analysis, involving the longer paths though the combinational block that determines the larger propagation delays.
- Hold time check: Verifies that the data remains valid enough time to satisfy the *hold time* of the latches. This type of analysis involves the shortest paths through the logic ensuring that

the data does not change too early.

Hold time violation verification is mandatory to ensure the correct circuit operation, since a *hold time* violation may cause a circuit malfunction due to a corrupt data captured by an output latch. This type of timing violations involves exclusively the shortest paths through a combinational block and therefore do not impose any restriction on the circuit operating frequency. Therefore, *hold time* violations are easy to identify and solve since they can be only caused by extremely short paths and are easily solved by adding a buffer or increasing the delay of some path gate causing the violation. For this reason, out attention will be concentrated on the setup time check.

Setup time checking is quite more complex than hold time checking since it requires identifying the largest propagation delay through the combinational block, involving the concept of critical path [11]. The critical path is defined as the path from an input node to an output node of a combinational logic block having the maximum delay [12]. The critical path delay determines the circuit maximum operating frequency.

The simplest way to estimate the maximum delay through a combinational logic block is to take the longest topological path computed by assigning a delay value to each gate in the circuit and adding the delay of the gates traversed by a path. This is a quick way of identifying the largest delay through a logic block, and may be useful as a first approximation to the maximum operating frequency permitted. However, in many cases the largest topological path is a false path being non-sensitizable meaning that a transition can never be propagated through this path. Consequently a maximum circuit delay overestimation may lead to a pessimistic timing analysis [13].

The path really determining the maximum block delay is the longest true path that in some cases may be considerably shorter than the longest topological path, since all longest paths may be false. Identification of the longest true path allows performing a more accurate timing analysis. This may allow increasing the operating frequency or keeping the frequency and reducing the circuit area and power consumption thanks to the use of weaker gates, i.e. logic gates with lower conductance having worst delay and consequently requiring less area and power consumption.

A pessimistic timing analysis occurs when the path considered to be critical is really a false path slower than the longest true path. This overestimates the maximum delay, although the correct circuit operation is guaranteed. Otherwise, if the worst true path is not correctly identified, and the path considered critical is not really the slowest true path an underestimation of the maximum delay is obtained. This corresponds to an optimistic timing estimation and may give rise to a circuit

malfunction. Thus, the correct identification of the critical path through a combinational block is a key step to perform a precise timing analysis.

A proper timing analysis must ensure that the circuit meets the timing constraints under different conditions and therefore must consider delays variations. Since the delay through a given path depends on multiple factors, then the circuit critical path may change depending on these factors. The elements that impact the delay through a path can be divided in two categories depending on its nature:

- Internal:
  - Parameter variations
  - Aging
  - In-circuit Noise
- External
  - Temperature
  - Supply voltage
  - External perturbations

The internal factors are specific of each circuit sample and operation, while the external factors depend on the operating environment conditions. Parameter variations occur between samples of the same circuit design due ti imperfections of the manufacturing process, and are a static factor since they do not change over time. There are two types of parameter variations:

- Die-to-die: The physical parameters of the devices, like dimensions or doping levels, vary between two samples of the same circuit even when are manufactured by the same process.
- Intra-die: Different regions of a single circuit suffer different parameter deviations in addition to the die-to-die variations. Intra- or within-die variations are due to the statistical nature of some manufacturing steps.

Circuit aging, is a degradation of the circuit components that in general worsens the circuit performance unlike the parameter variation that may produce circuits faster than the mean. In general, aging affects differently each circuit sample, since it depends strongly on factors like operating temperature since many aging effects involves thermally activated physical mechanisms [14][15]. However, despite its time dependent nature the aging in general affects the circuit slowly, requiring a long period of operation to experience important circuit performance degradation, specially compared to the external factors that present a very dynamic behavior [16].

Noise mechanisms may couple internal circuit nodes or induce supply/ground fluctuations that are highly operation dependent. Noise mechanisms like capacitive and inductive coupling are physically well understood and in theory could be accurately described and their influence on delay is highly dependent on the circuit operation. Circuit complexity and the dependence of such mechanisms on circuit operation prevent a practical description of these mechanisms that in practice are modeled as random in nature, adding to parameter variations [17].

The external delay variations factors are very dynamic as they can vary considerably during the circuit operation in short periods of time. The supply voltage should be stable in general however in addition to circuit activity there may be external effects that cause voltage drop effect. The voltage drop can affect the entire circuit or be localized to specific regions. Beside the unwanted effects, in modern circuits the supply voltage is intentionally lowered depending on the circuit activity to reduce power consumption and heating. Temperature is also a highly dynamic factor that depends on the environment temperature, the heat generated by the circuit itself and the cooling mechanisms to dissipate this heat.

All factors together contribute to variations in the propagation delays through a circuit that are different from one sample of the circuit to another and depending on the environment conditions and the circuit activity.

In summary, timing analysis must verify that the circuit meets the timing constraints in the worst case conditions, at least for the range of operating conditions imposed to the design. Depending on the way to consider the variations there are two types of timing analysis:

- STA (Static Timing Analysis): Computes the delays in a deterministic way i.e. considering static conditions. Possible variations on the static conditions are accounted by simulating

multiple sets of conditions. Usually the STA uses a strategy called corner analysis that provides a conservative result since corners settings are sets of extreme conditions. Therefore corner analysis guarantees the proper operation of the circuit under the worst possible conditions although this leads to a pessimistic analysis. The situation where all variables take the worst possible value for all components of the circuit is very unlikely.

- SSTA (Statistical Static Timing Analysis): Computes the delays through a circuit using probability distributions instead of deterministic values, giving a distribution of possible circuit outcomes rather than a single value. In general SSTA provides a less pessimistic prediction than the corner analysis at the cost of more complex process and larger runtime. The increase of parameter variations has motivated a considerable growth of this research field.

Independently of the strategy chosen and the variables considered, an accurate timing analysis requires the ability of identifying the set of true paths suitable of becoming a critical depending on the operating conditions. The importance of critical path identification relies on the fact that the delay difference from one path to another may be larger than the variation produced by the operating conditions and parameter variations. Thus this work is focused on precise path identification.

2.2. Delay modeling

The propagation delay through a combinational circuit determines the maximum frequency of operation of this circuit since the output signals must be correct and stable when the output memory element captures this value. If the propagation delay of a path is larger than the clock cycle, then the memory element will most probably capture an incorrect value [18]. These kinds of errors are called delay faults, and may be difficult to identify during the design stage [19].

To ensure that a circuit design will operate at a designated frequency, or to estimate the maximum frequency at which it could operate, a timing analysis of the design is required. This analysis must be performed during the design stage before manufacturing given the costs associated to an incorrect timing operation. Electrical simulation of complete real circuits is unaffordable due to the excessive computation resources required. This is solved through delay models that sacrifice accuracy to gain in computational speed. Some published delay models work at the transistor-level allowing their application to full custom designs and usually requires complex modeling techniques. Since many designs are completed through a synthesis process based on standard cell libraries, the delay model used in this Thesis works at the standard cell level.

Before introducing the mathematical details of the model and the algorithms to extract the required parameters, some well-known basic definitions about cell-level delays are detailed.

Definition: The propagation delay of a gate in a digital circuit is the time required by a signal to pass through the gate from one input to its output. This delay is given as the time lapse between the instant at which the input transition crosses the 50% of the supply voltage, and when the output transition crosses the same point, independently of the transitions direction. Fig. 2.4 shows a representation of the propagation delay through an inverter.

Definition: The transition time or slew time, is time required to change the voltage of a signal from its initial to its final value. The transition time is measured as the time between the instants when the signal crosses the 10% and the 90% of the supply voltage, for a rising transition and the opposite for a falling transition. Fig. 2.3 depicts a rising transition and its transition time.

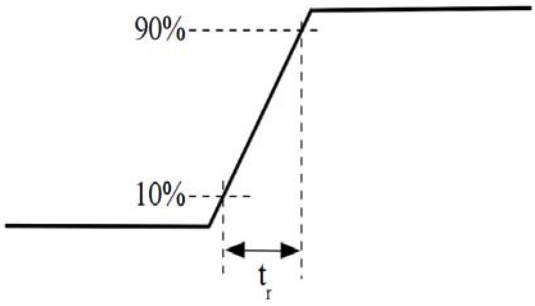


Figure 2.3: Transition delay

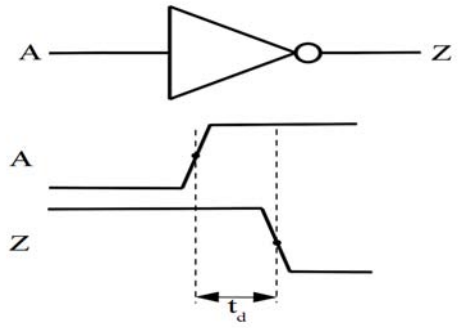


Figure 2.4: Propagation delay

2.2.1. Empirical modeling

Physically based delay models are derived more or less directly from the equations governing the voltage and current expressions that describe logic gate transitions. Technology scaling increases the transistor current expressions complexity as the device is miniaturized due to the side effects impacting its behavior. Such a complexity increase has an impact on physically based delay models whose growth makes them difficult to handle [20]. One alternative to overcome the high complexity of models based on physical descriptions is the use of empirical models extracted directly from the circuit behavior instead of its physical principles. Various alternatives have been proposed as exposed next.

2.2.1.1. Lookup Table (LUT)

A widely used strategy is the Lookup table (LUT) that involves tabulating a set of values for the magnitude of interest. Each value correspond to a combination of the considered variables, thus, the dimensionality of the table depends on the number of variables considered. In this approach, the data is discrete and the magnitude of interest is known for a discrete set of the input variables values. The result for any other value is obtained through interpolation algorithms that may range from a simple lineal interpolation to more complex interpolations techniques. This is equivalent to having piecewise model with a function that depends on the interpolation algorithm used. Despite all the benefits of LUT techniques, an analytical model has some advantages over LUT. Depending on the model analytical expression, the computation time may be faster than the interpolations required by LUT methods. The memory space required to store the model data is in general much smaller for analytical models and depends on the LUT size compared to the number of parameters of the analytical model. To accomplish the same accuracy through both methods, the LUT must have a considerable size resulting in a larger memory requirement. However, the main advantage of

an analytical model is the capability of being mathematically manipulated as for example differentiated, providing a measure of the impact of a fluctuation on a given variable.

Various analytical methods are typically used to model any magnitude with any number of variables. Some of them are detailed next.

2.2.1.2. Polynomial model

As was stated in 1712 by the Britannic mathematician Brook Taylor, any differentiable function can be represented by an infinite sum of terms that are calculated from the values of the function's derivatives at a single point. If the infinite series is truncated in a finite order, the result is an approximation of the function in some neighborhood. The order where the series is truncated determines the approximation to the real function. The Taylor series may be compactly written as (2.2)

$$f(x+x_0)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}\cdot x^n \tag{2.2}$$

A more practical representation of the polynomial function is given by (2.3).

$$y=f(x)=\sum_{i=0}^{n}P_i\cdot x^i \tag{2.3}$$

Equation (2.4) shows the equivalence between the polynomial parameters and the Taylor series terms.

$$P_i=\frac{f^{(i)}(0)}{i!}:0\le i\le n \tag{2.4}$$

To model some physical magnitude using this analytical expression, the parameters of the polynomial (P_i) must be extracted from the empirical data, or in the specific case of the digital circuit analysis, from electrical-level simulations results. A great advantage of the polynomial approach is that it does not require a numerical process to fit the data to the function. Instead, the parameters can be computed analytically, using a linear systems solver for which very efficient computation algorithms exists.

2.2.2. Extraction process

Starting with a set of empirical data (y) regarding to one variable (x), as shows (2.5), a polynomial regression of order n, can be accomplished by solving a linear system represented by the matrix equation (2.6).

$$y = f(x) \tag{2.5}$$

$$\underline{\underline{A}}_{(n+1)x(n+1)} \cdot \underline{P}_{(n+1)} = \underline{B}_{(n+1)} \tag{2.6}$$

Below are the details about the matrix $\boldsymbol{A}$ and both vectors P and B of (2.6). As shown in (2.7), matrix A is formed by sums of powers of the values of the independent variable x. The elements of vector P are directly the polynomial coefficients we are searching for. Vector B contains sums of products between the values of the dependent variable y, and powers of the independent variable x.

$$\begin{aligned} &\underline{\underline{A}}_{(n+1)x(n+1)} = \{a_{ij}\} \ : \ a_{ij} = \sum_{k=0}^{m-1} x_k^{(i+j)} \\ &\underline{P}_{(n+1)} = \{p_i\} \\ &\underline{B}_{(n+1)} = \{b_i\} \ : \ b_i = \sum_{k=0}^{m-1} x_k^i \cdot y_k \\ &\forall\, 0 \le i, j \le n+1 \end{aligned} \tag{2.7}$$

Where m is the number of samples of the data to be adjusted, and n is the order of the polynomial. I.e., the maximum polynomial expression exponent since the first exponent is 0, and the number of coefficients is $n+1$. An extended representation of matrix A, and vectors P and B are shown below.

$$\underline{\underline{A}}_{(n+1)x(n+1)} = \begin{bmatrix} m & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^n \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \cdots & \sum x_i^{n+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \cdots & \sum x_i^{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x_i^n & \sum x_i^{n+1} & \sum x_i^{n+2} & \cdots & x_i^{2n} \end{bmatrix} \quad \underline{P}_{n+1} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} \quad \underline{B}_{n+1} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \\ \vdots \\ \sum x_i^n y_i \end{bmatrix} \tag{2.8}$$

The order of the polynomial that must be used depends on two factors, the specific form of the data to adjust, and the accuracy desired.

2.2.3. Multivariable polynomial model

So far we have only considered one variable, while modeling the behavior of a circuit, in general, requires multiple variables. To include more variables into the model, the polynomial regression can be used hierarchically, i.e., each parameter of the polynomial is a function of another variable, which can also be adjusted to a polynomial form, and so on. As an example, (2.9) shows a function of two variables (x and y) fitted to a polynomial expression regarding to variable x, where each coefficient P_i is a function of y.

$$f(x,y)=\sum_{i=0}^{n_x} P_i(y)\cdot x^i \tag{2.9}$$

As shown in (2.10), each P_i coefficient is also expressed as a polynomial. The order of the polynomials $(n_x,\ n_{y_i}: 0\le i\le n_x)$ may be different for each case depending of the specific characteristics of each function. Finally (2.11) gives the general expression for 2-variable function fitted to a polynomial expression. This expression for two variables can be easily generalized to any number of variables.

$$P_i(y)=\sum_{j=0}^{n_{y_i}} P_{ij}\cdot y^j \quad \forall P_i: 0\le i\le n_x \tag{2.10}$$

$$f(x,y)=\sum_{i=0}^{n_x}\sum_{j=0}^{n_{y_i}} P_{ij}\cdot x^i\cdot y^j \tag{2.11}$$

To extract the coefficients for a multivariable model, the process is the same than the one explained for a single variable, simply applying it with respect to the first variable for each value of the second variable, resulting in a set of coefficients for each value of the second variable. In the second step these coefficients are fitted with respect to the second variable, obtaining a set of coefficients for each coefficient of the first step, and so on if there are more than two variables. The final result is a matrix of coefficients, with a number of dimensions equal to the number of variables. Depending on the order used for each regression some elements of this matrix can be null.

2.2.4. Computational advantages

The multi-variable polynomial model provides some computational advantages. The model coefficients can be represented using a matrix, with as much dimensions as variables considered. Since the matrix algebra has a wide use in the data processing field, this method benefits from the advances in the matrix computation. There are highly efficient matrix algorithms and even libraries that offer parallel computing to exploit current multi-core processors capabilities.

This kind of model allows performing a partial pre-computation by assigning a constant value to any of the considered variables. For instance, if the model depends on *n* variables, but in a specific application some of them have a constant value, the model may be preprocessed to simplify it. This preprocessing step reduces the matrix dimensions for each variable with a constant value assigned. Equation (2.12) shows the analytical expressions for a function with 3 variables, where a constant value is assigned to the variable *y*, resulting in a function with 2 variables. The 3-dimensional matrix P has been converted to a 2-dimensional matrix Q.

$$f(x,y,z)=\sum_i\sum_j\sum_k P_{ijk}\cdot x^i y^i z^k \;\rightarrow\; y=k \;\rightarrow\; f(x,z)=\sum_i\sum_k Q_{ik}\cdot x^i z^k$$
$$\rightarrow\quad Q_{ik}=\sum_j P_{ijk}\cdot k^j \tag{2.12}$$

Another computational advantage is that the model can be easily differentiated respect to any of its variables, to obtain the rate of change of the function. (2.13) shows an example of polynomial of two variables (*x, y*), with an *m by n* parameter matrix (*P*). The function is differentiated respect to variable *y*, getting a new polynomial function, where the parameter matrix (*Q*) was reduced to *m* by *n-1*, and each new parameter (Q_{ij}) can be easily computed from the original ones (P_{ij}).

$$f(x,y)=\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} P_{ij}\cdot x^i y^i \;\rightarrow\; \frac{\partial f(x,y)}{\partial y}=\sum_{i=0}^{m-1}\sum_{j=1}^{n-1} P_{ij}\cdot x^i\cdot j\cdot y^{j-1}=\sum_{i=0}^{m-1}\sum_{j=0}^{n-2} Q_{ij}\cdot x^i\cdot y^j$$
$$\rightarrow\quad Q_{ij}=(i+1)\cdot P_{i(j+1)} \qquad \forall\, 0\le i<m,\ 0\le j<n-1 \tag{2.13}$$

2.3. Effective capacitance

The capacitance of each circuit node is key when analyzing its dynamic behavior as it determines the signal time evolution. Specifically, the capacitance has a direct relation to the voltage variation in time and the amount of charge required to change the voltage level of a node. In this way, the efficacy of a circuit behavior prediction is strongly conditioned to the accurate estimation of the nodes capacitance. Some of the most relevant digital circuit analysis that require accurate capacitance estimation are:

- Timing analysis
- Power consumption
- Crosstalk analysis
- SET propagation

When working with digital circuits synthesized using a standard cell library the most interesting capacitance value is that of the input nodes of each cell, because these kinds of circuits are typically analyzed at the cell-level. Even in some cases, the designer has no access to the internal implementation of the cell, becoming impossible to perform an analysis at transistor or physical-level.

The main issue with the input capacitance of a CMOS cell is that it is a dynamic value depending on the voltage at each node. Therefore the value of the input capacitance varies during a transition, and even depend on the voltage transition speed. Accounting for this dependence -such as is done by SPICE-like simulation- implies adopting highly complex models resulting in an extremely time consuming simulation. Therefore, it is desirable to compute a capacitance steady that models as accurate as possible the dynamic behavior of the real capacitor. From now on we will refer as *effective capacitance* to the equivalent steady value of the dynamic capacitance of a cell input. The components that contribute to the input capacitance of a CMOS logic gate are detailed next.

2.3.1. Capacitance components

The input capacitance of a standard cell has two main contributions when describing an isolated cell, i.e., without considering any capacitive effect of the surrounding cells and wires of the circuit. The first contribution, at the transistor-level, is due to the MOSFET parasitic capacitors of the transistors forming the cell. A second contribution at the cell-level comes from the capacitors

formed by the layout layers (metals, poly-silicon and diffusion areas) and the insulating oxide between them. Besides these two contributions, in a real circuit, there are more capacitances to consider for a precise analysis, but these depend on the specific circuit topology and must be extracted at the circuit-level, falling outside the standard cell-modeling domain. In any case, the gates capacitances provide the more relevant contribution, except for the interconnect dominated sections such as clock trees and buses.

2.3.1.1. Transistor-level components

Regarding to the transistor-level contribution, basically a typical MOSFET transistor has five parasitic capacitors, as depicted in Fig. 2.5. They can be divided into two groups depending on if they are formed by an oxide between two conductors, or by a reverse polarized junction.

- Oxide capacitances: C_{gd}, C_{gs}, C_{gb}.
- Junction capacitances: C_{db}, C_{sb}.

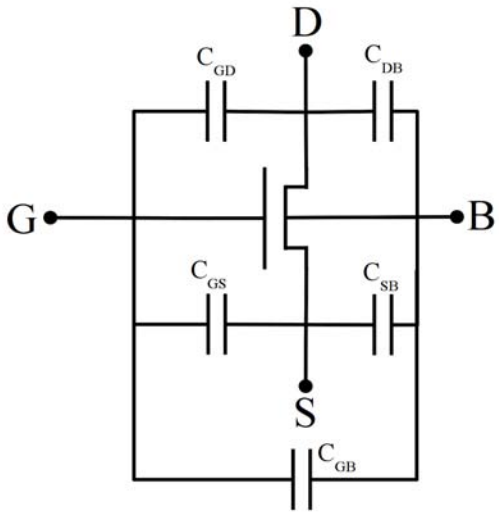


Figure 2.5: MOSFET parasitic capacitances

The oxide capacitances value depend on the transistor operation region, except for the component due to the gate and both diffusion regions (source and drain) overlap. This capacitance component is constant independently of the operation region, depends only on the overlapped area and the oxide coefficient, as shown in (2.14).

$$C_{overlap} = C_{ox} \cdot W \cdot L_D \tag{2.14}$$

where C_{ox} is the capacitance coefficient of the oxide, W is the transistor width, and L_D is the overlapping length between the gate and the diffusion.

The components depending on the operation region are due to the capacitor created between the gate and the channel being reason for its dependence on the operation region due to the channel shape variation. Table 3.1 summarizes the value of each oxide capacitance depending on the operation region specifying both the overlapping and channel components.

Table 2.1: Capacitance components

Capacitance	Cut-off	Linear	Saturation
C_{GB}	$C_{ox}WL$	0	0
C_{GD}	$0 + C_{ox}WL_D$	$0.5C_{ox}WL + C_{ox}WL_D$	$C_{ox}WL_D$
C_{GS}	$0 + C_{ox}WL_D$	$0.5C_{ox}WL + C_{ox}WL_D$	$(2/3)C_{ox}WL + C_{ox}WL_D$

The junction capacitances value is even more complex than the oxide capacitances. In addition to the junction area, their value depends on the doping coefficient of the semiconductor and the voltage of each junction side. However, since these capacitances are created between the substrate and the diffusion areas (drain and source), their contribution to the input capacitance of a cell is small, since the input of a CMOS cell is always connected to the gate of the transistors.

During a rising or falling transition, the transistors that form a CMOS gate pass through the three regions of operation. Some transistors transition from cut-off to saturation, and others do in the opposite direction. The oxide capacitances change their value during the transition, together with the junction capacitances that have a voltage dependent value, giving the dynamic equivalent capacitance of the CMOS cell. Fig. 2.6 shows a schematic of a CMOS inverter, including the parasitic capacitors affecting its operation. Some of the transistors parasitic capacitors have no impact because they have both terminals shorted, like the nMOS bulk-source capacitance.

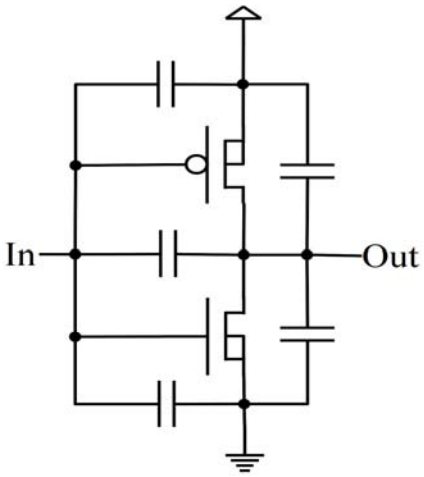


Figure 2.6: Inverter capacitances

As already stated, the electrical-level simulators, use extremely complex transistor models that include the capacitive parasitic effects. However, this level of accuracy is paid with a huge computation time, due to the complexity of the model equations. These kinds of simulations are unaffordable for large circuits, and require to be simplified at the expense of a lower accuracy in the estimation.

2.4. SET Propagation

A soft-error in a digital circuit is a circuit behavior alteration that occurs only during the circuit operation, usually caused by a dynamic event that impact circuit operation. Soft-errors are not associated to any physical defect in the circuit structure due to manufacturing errors or to inherent aging effects. Furthermore, soft-errors are usually not reproducible due to its dynamic nature and, since there is no physical defect in the circuit structure, they cannot be detected through a periodic circuit test, and its time occurrence cannot be predicted.

A source of soft-errors whose importance is growing due to technology scaling is alpha-particle and neutron strikes. Main sources are either contaminants from the circuit encapsulation, or cosmic radiation, i.e., high-energy particles coming from the space. Circuits operating outside the atmosphere protection, like space probes and artificial satellites, are specially affected by soft-errors. However, IC miniaturization is making soft-errors a concern even for sea-level applications.

When an ionizing particle strikes the diffusion areas of a semiconductor device it generates free electron-hole pairs that are drained in opposite directions by internal electric fields, and may result in an injected charge at a specific node. Such an injected charge, modeled through a current source, may result in a short duration voltage pulse. If the particle strike impacts a memory circuit, or a register latch, then the voltage pulse induced by the ionizing particle may flip the stored value, generating a corruption of the stored data. This type of soft-error is referred to as a Single-Event-Upset (SEU), and may be an important cause of errors in the memory circuits, especially those that must operate on a hostile environment like space missions. On the other hand, if a particle impacts a combinational logic block, it generates a transient voltage pulse at the combinational circuit node, and in general is referred to as a Single-Event-Transient (SET). Typically SEU and SET effects are categorized as Single Event Effects (SEEs).

The possibility of an SEE to induce an error depends on various factors. If the ionizing particle impacts a memory element, the possibility of generating an error depends mainly on the relationship between the amount of charge injected and the electrical strength of the latch logic gates. The injected charge must be enough to force a memory element flip, surpassing the current draining capacity of the cell. In the case of an SET at an internal node of a combinational block, there are additional masking factors that may prevent such a perturbation to trigger a circuit error [21]. An SET may cause an error only if it propagated within the combinational logic until reaching a register that must capture the perturbation initiated by the SET.

SET effects have been extensively studied and the conditions for an SET to propagate within a circuit are well understood [22][23][24][25]. There are three masking mechanism, that may be classified into two categories. The first category includes the masking mechanisms related to the SET propagation within the circuit, i.e., the *electrical masking* and the *logic masking*. The second category is related to SET capturing at a memory element, i.e., the *time masking*. Then, multiple conditions need to be concurrently satisfied for an SET to induce a SEU:

1. *Electrical masking*: The voltage perturbation must have the appropriated electrical characteristics to traverse the circuit logic gates, and be capable of switching the memory element state. This masking mechanism depends on the relationship between the induced pulse, the logic gates electrical characteristics and the nodes capacitance. Logic gates with short delay values and small loads will pass narrow pulses, while slow gates or heavy loaded ones will filter short pulses.
2. *Logic masking*: There must be a logic path sensitized for the perturbation to travel from its originated node to a memory element. If the pulse is generated at an internal node, and none of the logic gates that such node feeds, has the appropriate logic input values that propagate the perturbation toward the output, then the pulse simply vanishes without any impact on the circuit operation. This masking mechanism depends on the logic gate type and the logic values applied at the combinational block inputs. An example of logic masking is shown in Fig. 2.7.
3. *Time masking*: The voltage perturbation generated within the circuit must arrive at some circuit output within a time window during which the connected memory element is transparent. A pulse just reaching a memory element input does not imply a logic error, unless it reaches the register within the capture time window. The time window size is related to the memory element setup and hold times. Fig. 2.8 shows an example of two SETs at the input node of a register, the first one is captured, while the second is masked as it falls outside the capture window.

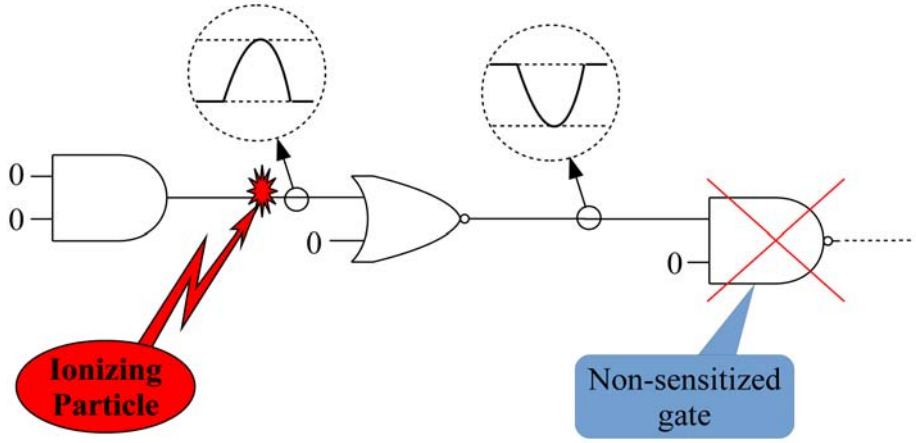


Figure 2.7: Logic masking

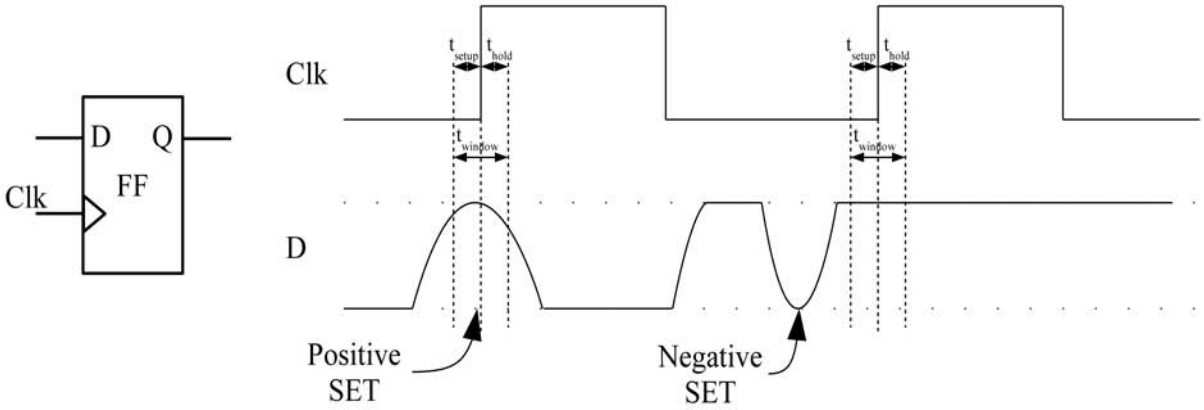


Figure 2.8: Time masking

The described masking mechanisms prevent that all SETs generated within a circuit ends up causing an SEU. However technology evolution exacerbates the IC susceptibility to SEEs caused by SETs. In the case of memories, the soft-error-rate (SER) does not increase as much as in combinational logic with technology scaling, because critical memory systems are usually protected through redundancy techniques. In general error-correcting-codes (ECC) are quite effective to protect memory elements against this type of errors without introducing an excessive overhead. However, the combinational blocks SER increases at each technology node, just as predicted years ago [26], reaching and even surpassing the error rate of unprotected memory elements in current technologies for the reasons detailed next.

When a high-energy particle strikes on a circuit node it produces an amount of charge at the node parasitic capacitance that modifies the voltage level of the affected node. The amount of charge injected depends on the particle energy while the charge impact on the circuit operation depends on the parasitic capacitance value and the circuit supply voltage. The capacitance definition reveals that the voltage variation is directly proportional to the charge and inversely proportional to the capacitance (2.15).

$$C = \frac{Q}{V} \rightarrow \begin{cases} V \propto Q \\ V \propto \frac{1}{C} \end{cases} \tag{2.15}$$

In the past technologies, where the parasitic capacitances were relatively large because of the device and interconnect sizes, the amount of charge injected by a particle represented a small contribution to the node voltage perturbation. These technologies also had a relatively high supply voltage compared to the voltage spike induced by the particle that was in general negligible. However, technology scaling shrinks device dimensions - reducing the nodes parasitic capacitance – and dictates also a circuit supply voltage reduction. The small node parasitic capacitances of today nanometer technologies entail that the charge induced by an ionizing particle results in a considerable voltage variation compared to the actual supply voltage values. Therefore SET generation in current technology CMOS ICs is increasing [27]. Moreover, in addition to SET rate generation increase, the masking mechanisms become less effective as technology scales down. Electrical masking effectively reduces mainly for two reasons. First, supply voltage reduction and node capacitance decrease, contributes to the SET propagation, and second, faster logic gates contribute less to filter short-duration glitches due to the propagation delay decrease. In addition, the increase of circuit clock frequency makes time masking less severe since the memory elements remain transparent more frequently, increasing the probability that a register captures an invalid value due to an SET. Therefore circuit SER estimation and improvement of circuit hardiness techniques becomes an important step during design phase.

Most existing works dealing with SET propagation probability are mainly based on computing the circuit Soft-Error-Rate (SER) considering statistical techniques [28][29][30][31]. Some use a fixed amount of injected charge for the SET generation, while others consider the injected charge to be a random value, or even take a range of charge values. Many different methods are used to

model the injected charge for SER computation. There are many SER computation techniques that provide statistical metrics of the probability that a circuit suffers an error due to an SET propagated and captured by a memory element. However, none of them offers a deterministic metric about the minimum electrical characteristics required to allow a pulse to propagate from a given node until a circuit output.

A key issue in the development of circuit-hardening techniques with respect to SETs is related to identifying the nodes having the highest sensitivity to SET (i.e. the weakest sites in the circuit), together with the determination of the possible paths through which an SET can propagate toward a memory element.

One of this work objectives, is to estimates the SET propagation through a circuit by computing, for each circuit node, the minimum electrical characteristics (width and height) of a voltage pulse capable of traveling toward a circuit output, independently of the pulse cause. The independence of the pulse cause makes it suitable for describing any kind of induced voltage pulse propagation in a circuit.

The minimum electrical characteristics required to allow a pulse to be propagated depends only on static parameters, like the node capacitance and logic gates conductance. A specific mode developed in [7] will be used in this work.

Chapter 3: Framework core elements

This chapter presents the core elements of the EDA Framework developed. The chapter starts by introducing some basic definitions related to paths within a combinational logic block that will be used in the next sections. The concept of logic system is described, detailing the logic system specially developed to perform efficient path identification. This logic system has the special feature of treat both transitions at each node at a time. Each basic operation associated to the path identification techniques is detailed individually. These operations conform the basics of the customizable algorithms detailed in this chapter.

3.1. Definitions

For a combinational circuit C = {PI, PO, W}, let

PI = $\{I_i\}$ be the set of primary inputs,

PO = $\{O_i\}$ be the set of primary outputs

W = $\{w_i\}$ be the set of the wires, i.e. internal nodes.

Structural path: a structural path P_S, is a sequence of nodes ξ_i (including I_i, O_i and w_i) through a combinational block starting at a primary input node and ending at a primary output node, by

traversing the logic gates always from input to output (3.1). A structural path can be a true path or a false path as defined next.

$$P_S=\{\xi_1,\xi_2,\ldots,\xi_{n-1},\xi_n\} \;:\; \xi_1\in PI,\;\; \xi_i\in W \;\; 2\le i\le n-1,\;\; \xi_n\in PO \tag{3.1}$$

False path: if a structural path cannot be sensitized, i.e., a transition applied at the input node cannot be propagated through the path until reaching the output, then the structural path is a false path. A path cannot be sensitized if some gate within the path cannot be sensitized due to a logic incompatibility.

True path: a structural path is a true path if there exists an input vector capable of sensitizing a structural path, i.e. it is possible to propagate a transition through it.

Functional path: a functional path is a combination of a true path and a specific transitions sequence through the path nodes. Note that, for each true path there are at least two functional paths: one having a transitions sequence starting with a rising transition at the primary input node, and the other starting with a falling transition. The transition direction (rising or falling) at each circuit node depends on the gate type traversed, and specifically if it is an inverting or non-inverting gate. Since an exclusive-or gate (XOR or XNOR) polarity depends on the logic value settled at he inputs not being part of the functional path, then each exclusive-or in a true path may contribute with up to four functional paths.

Therefore, if $P_S=\{\xi_1,\xi_2,\ldots,\xi_{n-1},\xi_n\}:\forall\,\xi_i\in C$ is a structural path through the circuit *C*, then $P_f=\{\xi_1^{t_1},\xi_2^{t_2},\ldots,\xi_{n-1}^{t_{n-1}},\xi_n^{t_n}\}$ (where $t_i\in\{r,f\}$ with r indicating a rising transition, and f a falling transition.) is a functional path as long as the path can be sensitized, otherwise it is a false path.

Sensitization vector: a sensitization vector is each logic input pattern that sensitizes a functional path. Each functional path may have multiple sensitization vectors since in general logic gates can be sensitized by more than one vector, and the logic values required to propagate a transition through a gate can be obtained in multiple ways within a circuit.

On-path input: logic gate input through which a given path crosses the gate.

Off-path inputs: logic gate inputs that do not belong to the path. Gate sensitization depends on the logic values applied to the off-path inputs.

For clarity, the previous definitions are illustrated on a real combinational circuit. Fig. 3.1 shows the structure of a 2-bit carry-bypass adder.

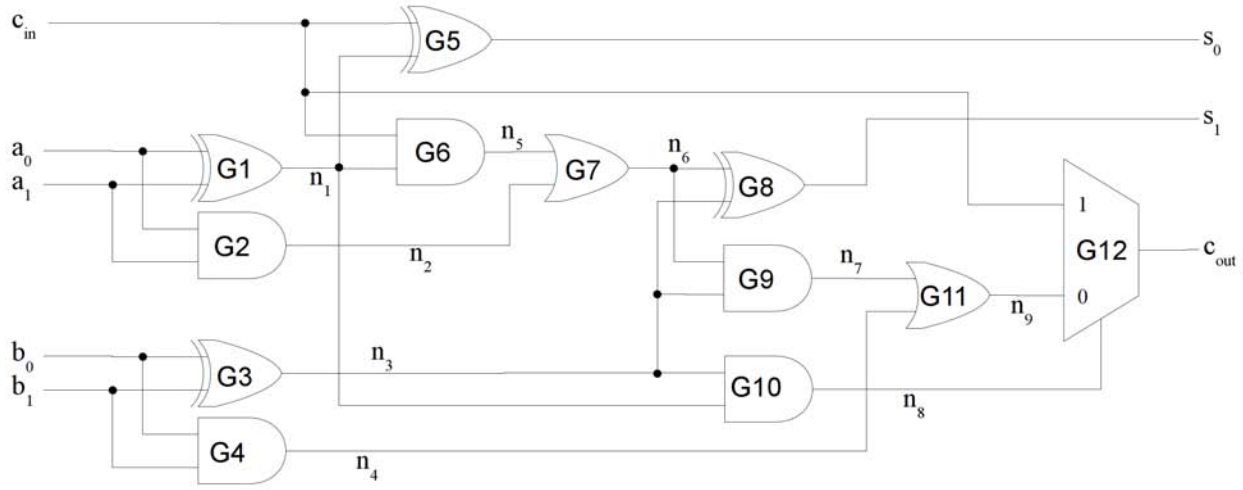


Figure 3.1: 2-bit carry bypass adder

Taking two structural paths P_{S1} and P_{S2} as example

$P_{S1} = \{C_{in}, n_5, n_6, n_7, n_9, C_{out}\}$

$P_{S2} = \{C_{in}, n_5, n_6, S_1\}$

It can be easily verified that P_{S1} is a false path. It passes through gates G6 and G9 that require a logic 1 at nodes n1 and n3 respectively to be sensitized, since this is the non-controlling value for an AND gate. Such assignments imply a logic 1 at node n8, that makes multiplexer (G12) to select the input connected at node C_{in}, impairing a transition propagation from node n9 to the output C_{out}.

The structural path P_{S2} passes through an XOR gate (G8). Therefore, there are four transition sequences of transitions through this path:

$$\{C_{\text{in}}^{r}, n_5^{r}, n_6^{r}, S_1^{r}\} \qquad \{C_{\text{in}}^{f}, n_5^{f}, n_6^{f}, S_1^{f}\}$$

$$\{C_{\text{in}}^{r}, n_5^{r}, n_6^{r}, S_1^{f}\} \qquad \{C_{\text{in}}^{f}, n_5^{f}, n_6^{f}, S_1^{r}\}$$

All four sequences are sensitizable, meaning that there are four functional paths for this

structural path. Moreover, each functional path has multiple sensitization vectors.

A transition propagation through gate G6 requires a logic 1 at node n1, while propagation through gate G7 requires a logic 0 at node n2. These requirements can be accomplished in two ways:

$$\left.\begin{array}{l} a_0=0, b_0=1 \\ a_0=1, b_0=0 \end{array}\right\} \rightarrow n_1=1, n_2=0$$

3.2. Logic System

3.2.1. Theoretical concepts

Digital circuits operation is based on Boolean algebra formalism defined by three basic logic operations (inversion, AND and OR) operating on a two-valued element set consisting on what is referred to as logic values (0, 1). Most of today digital circuits implement such logic values through voltage levels. However, although digital circuits work internally with only two states, circuit analysis typically uses a more sophisticated logic system, called logic algebra, consisting of more than two logic values and usually referred to as *Multiple-valued logic*. Some examples of logic system used in digital simulations, where some values can be unknown, are: a three-valued logic system {0, 1, X}, or a 5-valued logic system {0, 1, D, $\overline{D}$, X}. These multiple-values logic systems are commonly used in algorithms that generate patterns for stuck-at faults test. The extra values represent problem-specific states, and provide an abstraction level that helps simplifying circuit analysis. For example, the unknown value *X* allows carrying out a digital simulation of a circuit without the need of knowing all input nodes values. In general, the set of logic values used depends on the conditions of the specific problem to solve.

An algorithm to derive a logic system for a given application was published in [32]. Together with the generation algorithm also provides a completeness theorem that is used to ensure that the logic system is complete.

Relying on general terminology and notation used in the literature, a *v*-valued logic, where *v* is the total number of values in the algebra, is denoted as L_v. The logic system values can be partitioned into basic values (B_v), and composite values (C_v), then $L_v = \{B_v, C_v\}$. Each composite value represents a set of basic values. The notation for a logic system is summarized in (3.2).

$$\left.\begin{array}{r} \textit{Basic values}: \ B_v=\{b_i\} \\ \textit{Composite values}: \ C_v=\{c_i\} \end{array}\right\} \rightarrow \ L_v=\{B_v, C_v\}$$
$$c_i=\{b_{i1}, b_{i2}, \ldots, b_{\text{in}}\} \ : \ b_{ij}\in B_v, c_i\in C_v \tag{3.2}$$

The algorithm to derive a complete logic system for a specific application presented in [32] can be summarized in the following steps:

- The first step consists in defining the requirements for the specific logic system application, and, based on these application requirements an initial set of values is produced. These are the basic values of the logic algebra.
- Second, each of the initial set of values combination is evaluated using the basic logic functions {*Not*, *Or*, *And*}. Complex logic functions are not required as they can be decomposed in basic ones.
- The result of evaluating the logic functions using the initial values provides new values not present in the initial set. These new values are composite values and complete the overall set of values. The evaluation process of the previous step continues using the new values as input values, until no more new values are obtained.

3.2.2. State of the art

As commented earlier, there is a high number of logic algebras for many kinds of applications available in the literature. These range from logic systems with only 3 values [33], to complex logic algebras with a large number of values, like 23-valued and 41-valued system presented in [32]. As an example, Table 3.1 lists the values of a logic system published in the context of path delay test generation [33][34]. Fig. 3.2 shows the Hasse diagram of this logic system. A Hasse diagram is a common way to represent a logic system, and the relationships between the composite values and the basic values covered by each composite value.

Table 3.1: 10-valued logic system

Value	Description
0s	Stable 0
1s	Stable 1
$0\bar{s}$	Final value 0 but unstable
$1\bar{s}$	Final value 1 but unstable
0x	Final value 0, stable or unstable
1x	Final value 1, stable or unstable
$X\bar{s}$	Unknown value but unstable
$X\bar{s}^{0}$	Unknown value but not stable 1
$X\bar{s}^{1}$	Unknown value but not stable 0
X	Unknown value

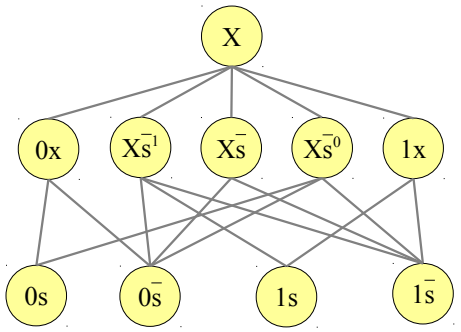


Figure 3.2: Hasse diagram of 10-valued logic system [33]

3.2.3. Developed Logic System

The purpose of the logic system developed in this work is to determine if a given perturbation can propagate through a logic path. Therefore, the focus of the logic system is to allow representing a transition, and determine the likelihood that such transition passes through a logic gate. From the simplest point of view, the main characteristic of a logic transition is a signal starting with one logic value and ending ups with the opposite one. Then, an easy way to represent a transition is through the start and end values. This representation can be extended to other values besides transitions, and is the basis of the logic system developed in this work.

As was stated, the first step is to determine the initial set of values based on the specific requirements. Starting from the two basic logic states "0" and "1", and the concept of initial and final value, the possible combinations are constructed. The values are represented using two characters, the first for the initial state and the second for the final one ignoring what happens in the middle. The initial set of values is completed with a fully undetermined value, i.e., values where its initial and final values are unknown. Table 3.2 shows the initial set of values, formed by four basic values and a composite value that represents any of the basic ones.

Table 3.2: Initial set of values

Initial state	Final state	Value	Compacted notation	Basic values	Description
0	0	00	0	{00}	Steady 0
1	1	11	1	{11}	Steady 1
0	1	01	R	{01}	Rising transition
1	0	10	F	{10}	Falling transition
X	X	XX	X	{00, 11, 01, 10}	Unknown

Once the initial set of values has been established. The next step is to evaluate the three basic functions using each combination. Table 3.3 shows the propagation through a *NOT* gate, in this case all the output values are members of the initial set.

Table 3.3: Not gate propagation

In	Out
00 (0)	11 (1)
11 (1)	00 (1)
01 (R)	10 (F)
10 (F)	01 (R)
XX (X)	XX (X)

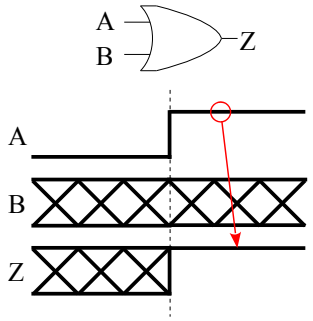


Figure 3.3: Semi-undetermined value

Tables 3.4 and 3.5, present the result of propagating the initial set through *OR* and *AND* gates respectively. Due to the commutative property of both functions only half of the table is given for each gate. As shown, there are four new values called semi-undetermined values since they represent a case where only the initial or the final value is known, but not both. Fig. 3.3 gives an example of this kind of composite values. An OR gate with a rising transition at input A and an unknown value at input B produces a determined final value even if there is an unknown at input B, since a logic "1" is a controlling value for the OR gate. However, the initial value is undetermined because there is no controlling value nor all values determined. Each of these new values, represents a set of basic values, as shown in (3.3) and (3.4).

Table 3.4: Or Propagation

OR		Input B				
		00 (0)	**11 (1)**	**01 (R)**	**10 (F)**	**XX (X)**
Input A	**00 (0)**	00	11	01	10	XX
	11 (1)		11	11	11	11
	01 (R)			01	11	X1
	10 (F)				10	1X
	XX (X)					XX

Table 3.5: And Propagation

AND		Input B				
		00 (0)	**11 (1)**	**01 (R)**	**10 (F)**	**XX (X)**
Input A	**00 (0)**	00	00	00	00	00
	11 (1)		11	01	10	XX
	01 (R)			01	00	0X
	10 (F)				10	X0
	XX (X)					XX

$$\{X1\}=\{11,01\}$$
$$\{1X\}=\{11,10\} \tag{3.3}$$

$$\{X0\}=\{00,10\}$$
$$\{0X\}=\{00,01\} \tag{3.4}$$

If the logic functions are evaluated again using the new values, no additional new values are obtained. Then, the logic system generation process is finished, getting a 9-value logic system that can be demonstrated to be a complete system using the completeness theorem published in [32]. Next, the complete set of values is shown dividing the values in basic (3.5) and composite (3.6) values, while the Hasse diagram for this logic system is depicted in Fig 3.4.

$$B = \{00, 11, 01, 10\} \tag{3.5}$$

$$C = \{0X, X0, 1X, X1, X\} \tag{3.6}$$

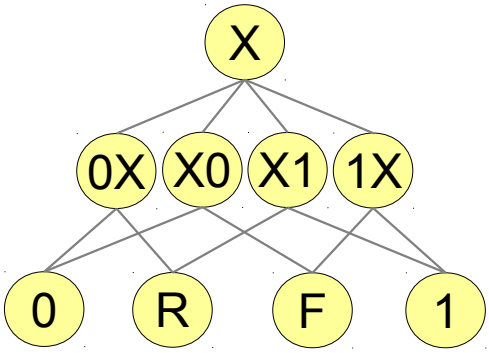


Figure 3.4: Hasse diagram

Since this logic system represents the values exclusively as an initial and final state, there may be an issue when there are opposite transitions at both gate inputs, because depending on the arrival time of each transition a static-hazard can occur. An example is shown in Fig. 3.5 for the case when two opposite transitions arrive to an OR gate. As shown, the hazard presence depends on the relative arrival time between the two transitions. Given that this logic system primary goal is simplicity and efficiency, this situation is not considered. Accounting for hazards in transition propagation results in a huge complexity increase due to its time-dependent nature. A workaround to detect such kind of situations would consist in applying post-processing only when the hazard generation is possible.

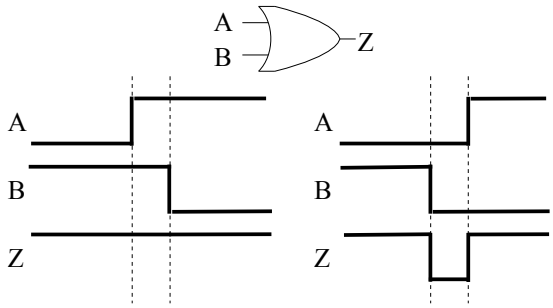


Figure 3.5: Static-hazard

3.2.4. Dual logic system

The logic system introduced can be expanded to consider both transitions at a time, instead of using a traditional two-step process, consisting in applying first one transition and then the opposite one. Both transitions are considered simultaneously by converting the logic system to dual values. Joining two values of the previous logic system creates such dual values that are represented by using four characters, the first value initial and final state, and the second value initial and final state. Starting from the *R* and *F* values that represent a single transition, we define two dual values *RF* and *FR*, that represent both transitions simultaneously. *FR* is the result of inverting *RF* value.

Thus, the initial set of values of the dual logic system are:

Table 3.6: Dual logic system initial values

4-value representation	Compacted representation	Description
00 00	0	Steady 0
11 11	1	Steady 1
01 10	RF	Rise / Fall transitions
10 01	FR	Fall / Rise transitions
XX XX	X	Undetermined

Applying the same techniques described in the previous section using the initial set of dual values we obtain four composite values. Each value is a combination of two previous logic system composite values. Table 3.7 shows these new values using the 4-values representation, and the compacted representation.

Table 3.7: Composite dual values

4-value representation	Compacted representation
0X X0	0X0
X0 0X	X0X
1X X1	1X1
X1 1X	X1X

Similarly to the previous case, it can be demonstrated that this set of values applied to each logic function does not generates new values. Finally, Fig. 3.6 shows the Hasse diagram of the dual logic system that treats both transitions simultaneously, with the same number of values than the simple logic system.

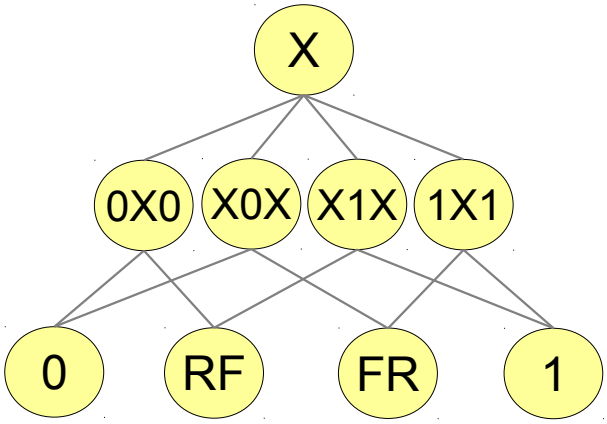


Figure 3.6: Hasse diagram of dual value logic system

Fig. 3.7 illustrates the logic system application to finding true paths, and specially the ability of detecting incompatibilities in advance, thanks to the semi-undetermined values.

(1) All nodes of the circuit are initialized to undetermined value (*X*).

(2) The *RF* value is assigned to a circuit input node and is forward propagated. Even although only one value is fixed, the semi-undetermined values allow the propagation.

(3) Then, the value *1X1* at the NAND gate output makes impossible to propagate the transition through the first input of the OR gate. The logic system allows detecting this incompatibility when only one input values is fixed.

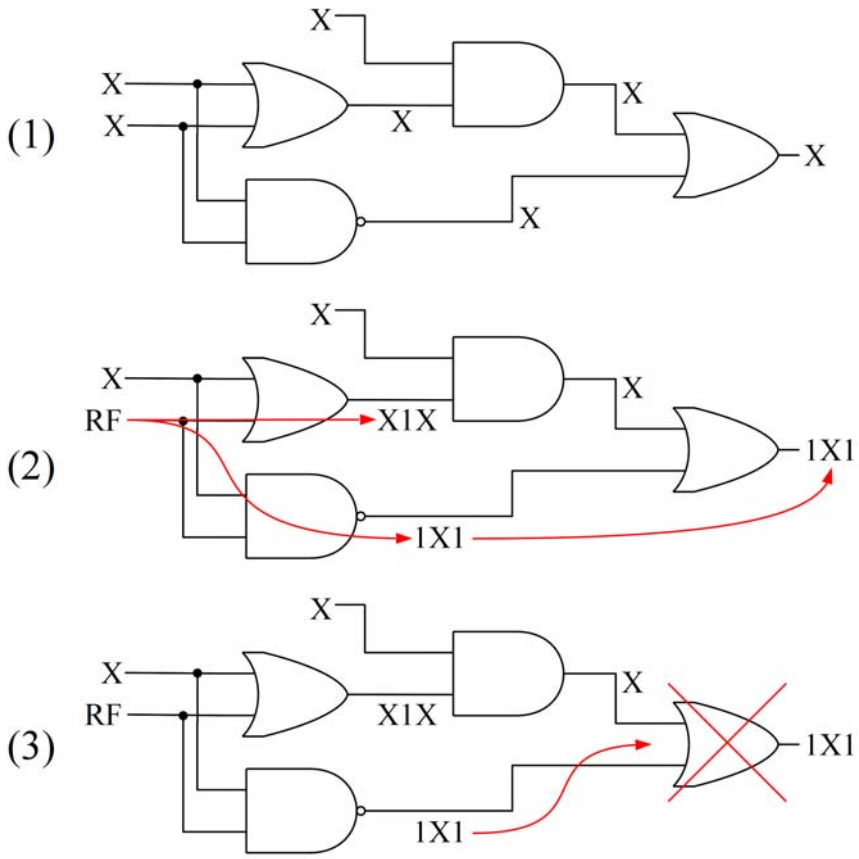


Figure 3.7: Logic system example

3.3. Basic operations

This section details each basic operation involved in the path sensitization algorithms.

3.3.1. Sensitization

Gate sensitization, is the process of applying adequate logic values to the off-path inputs to propagate a transition from the on-path input to the gate output. I.e., setting the gate in transparent mode with respect to the input making a transition [35]. The logic values required at the off-path inputs depend on the gate type and in some cases on the on-path input. Furthermore, the sensitization of a gate can be accomplished using different sensitization criteria that also determine the values to apply at the off-inputs.

Regarding the sensitization conditions, logic gates can be classified in three categories (single input logic gates are not considered, since sensitization makes no sense for these gates):

- Basic logic gates: gates implementing a primitive logic function: OR, AND, NOR and NAND gates. These gates are characterized by having a single sensitization vector for each input. Independently of the number of inputs of the gate, all off-path inputs must always have a determined logic value and, in general, the value required is the same for each off-path input. The exception to the last rule are the logic gates implementing basic functions with some inverted input, like AND2A, i.e., a 2-input AND gate with the input A inverted. These gates are typically found in standard cell libraries.
- Complex gates: gates that combine primitive logic functions in a single CMOS structure that reduce the number of transistors required to perform the logic function compared to the interconnection of multiple basic gates. Typically, complex gates comprise a combination of few primitive functions, although more complicated functions like full-adder or multiplexer are also used. Concerning the sensitization, complex gates have in general multiple sensitization vectors for each input node. Some complex gates can be sensitized without applying logic values to all the off-path inputs, unlike the basic gates.
- Exclusive gates: The XOR and XNOR gates have the special property that the output transition does not depends exclusively of the input transition, but also depends on the off-path input values. In this way, a logic gate of this category can act as an inverting or non-inverting gate depending on the logic values feeding the off-path inputs.

The sensitization criterion can be established to fulfill the requirements of the specific analysis to be carried out, and determine the off-path logic values required to sensitize a gate under specific conditions. Depending on the requirements imposed the sensitization criterion the specific conditions may be more or less restrictive. Some general sensitization criteria are provided next, however the specific requirements can be configured to obtain application specific conditions.

1. Full-determined steady sensitization: All off-path inputs take fully determined steady logic values, and cannot change. The paths sensitized by using this sensitization criterion are fully independent of the off-path delays.
2. Relaxed steady sensitization: this criterion is a relaxed version of the previous one. In this case, steady logic values are required only for a reduced set of off-path inputs to guarantee the sensitization of the gate, allowing undetermined values in the rest of the off-path inputs. This criterion makes only sense for complex gates, since basic gates require all their off-path inputs to be determined for sensitization. Like in the previous criterion, the paths sensitized by this way are sensitizable independently of the off-path delays. These paths are referred to in the literature as *Robust testable paths* [36].
3. Minimum condition sensitization: the conditions imposed are the minimum capable of sensitizing the gate, without imposing steady logic values at the off-path inputs neither determining values at all inputs. Thus, semi-undetermined logic values can be used to always guarantee the proper transition propagation.

Minimum condition sensitization:

The minimum condition criterion exploits the advantages of the logic system semi-undetermined values with the objective of applying the most flexible logic value capable of sensitizing the gate. Each primitive logic function has a controlling logic value that uniquely defines the output value independently of the other inputs values. Therefore, when a gate has a controlling value at the on-path input, the off-path inputs can have undetermined values. Fig. 3.8 shows this characteristic for a rising and a falling transition applied to AND and OR logic functions.

As shown in Fig. 3.8, the off-path input of a logic gate can be undetermined when the on-path input takes a controlling value, either for the initial or the final value depending on the transition

direction and the logic function. Table 3.8 gives the logic values required at the off-path inputs of the primitive functions for both transitions.

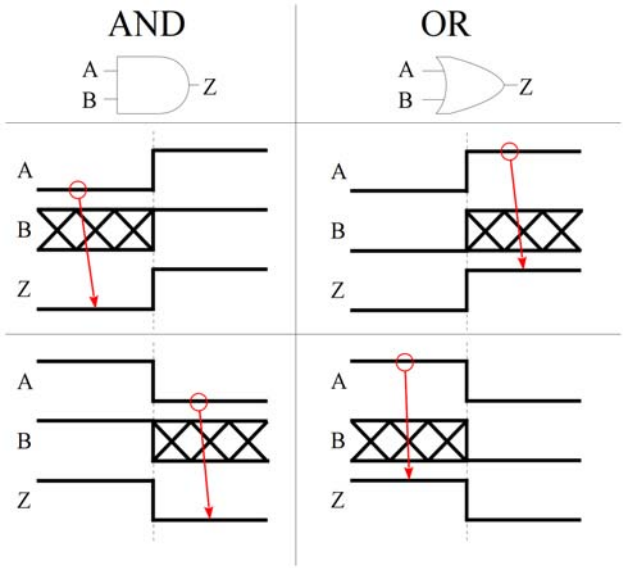


Figure 3.8: Minimum sensitization conditions

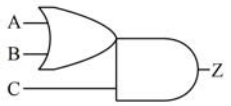
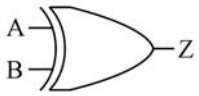
Table 3.8: Minimum sensitization condition

	AND	**OR**
R	X1	0X
F	1X	X0

This kind of sensitization is conditioned to the off-path delays, since if the off-path input is a transition (X1 can remain steady 1 or perform a rising transition) this would result in a transitions at both gate inputs. When both inputs transition the instant of the output transition depends on the input transitions direction and their relative arrival time. If the input transition goes from a controlling value to a non-controlling value, then the last input transition forces the output transition. However, if the input transitions change from non-controlling value to a controlling one, then the first input transition induces the output change.

To get a more detailed insight of these sensitization criteria, they are detailed for each logic gate category using the 9-valued logic system developed in this work. Table 3.9 shows the three gates chosen to illustrate the sensitization criteria for each category, including name, logic function and symbol.

Table 3.9: Example gates

Basic gates	AND3	$Z = A * B * C$	
Complex gates	OA12	$Z = (A + B) * C$	
Exclusive gates	XOR2	$Z = A \oplus B$	

Basic gates (AND3):

The non-controlling logic value of an AND gate is 1, therefore to sensitize an input with a steady value this gate requires a logic 1 at all off-path inputs. Since a basic gate requires a determined value at all inputs, there is no difference between the criterion 1 and 2. The sensitization table for both criteria is given in Table 3.10.

Table 3.10: AND3 steady values sensitization

A	B	C	Z
RF	1	1	RF
1	RF	1	RF
1	1	RF	RF

Table 3.11 corresponds to the sensitization table for the minimum condition criterion applied to an AND3 gate and shows the criteria relaxation with respect the previous table. Steady values have been replaced by semi-undetermined values following the minimum conditions requirements. As in the others criteria, the basic gates, requires the same logic values at all off-path inputs.

Table 3.11: AND3 Minimum condition sensitization

A	B	C	Z
RF	X1X	X1X	RF
X1X	RF	X1X	RF
X1X	X1X	RF	RF

Complex gates (OA12):

As explained earlier, complex gates may have multiple sensitization vectors for each input. In the example case, only one of the three inputs has more than one sensitization vector, as shown in the tables. Depending on the complex gate specific structure, the number of sensitization vectors for each input varies. For gate OA12 it is clear that any input combination forcing a logic 1 at the internal OR gate output sensitizes input C. Then, in the case of the first criterion shown in Table 3.12, three input vectors sensitize input C.

Table 3.12: OA12 Steady values sensitization

A	B	C	Z
RF	0	1	RF
0	RF	1	RF
1	0	RF	RF
0	1	RF	RF
1	1	RF	RF

The second sensitization criterion simplifies the cases with multiple sensitization vectors, because requires only the minimal set of values fully determined. Thus, the logic 1 required at the OR gate output necessary to sensitize input C, can be accomplished with a 1 at any of the inputs, independently of the value of the other input, as shown in Table 3.13.

Table 3.13: OA12 Relaxed steady value sensitization

A	B	C	Z
RF	0	1	RF
0	RF	1	RF
1	X	RF	RF
X	1	RF	RF

Table 3.14 gives the sensitization vectors for the minimum condition criterion, where the steady values of the previous table have been replaced by their semi-undetermined equivalents.

Table 3.14: OA12 Minimum condition sensitization

A	B	C	Z
RF	0X0	X1X	RF
0X0	RF	X1X	RF
X1X	X	RF	RF
X	X1X	RF	RF

Exclusive gate (XOR2):

Exclusive gates do not have controlling and non-controlling values, thus any value sensitizes the gate. As shown in Table 3.15 when a transition arrives at input A, it is propagated independently of the input B value, but this value determines the output transition direction. For this type of gate there is no difference between the sensitization criteria, since an exclusive gate requires that all input values are fully determined to propagate a transition.

Table 3.15: XOR2 sensitization table

A	B	Z
RF	0	RF
RF	1	FR
0	RF	RF
1	RF	FR

3.3.2. Implication

Implication is the process of assigning as many logic values as possible to combinational block nodes to determine logic values without the need of taking any decision. This is, setting all logic values that are uniquely determined by the logic values already assigned [37]. This is a vital step for an efficient path sensitization, because maximizing the number of nodes with a logic value assigned helps identifying logic conflicts as early as possible, minimizing the number of options in the cases where a decision must be taken [38].

The implication procedure may be separated in two components depending on the implication direction.

- *Forward implication*: propagates the logic values assigned at internal nodes toward output nodes of the gates fed by these nodes. The process continues until there are no more logic values that can be uniquely determined. Fig. 3.9 shows an example of forward implication: a 0 has been set to node B to sensitize the topmost OR gate, implying a 0 at node E, since it is the controlling value of the AND gate. The forward implication ends here because a logic 0 at node E does not uniquely imply any value at the OR gate output. The forward implication is a quite simple process since it only requires evaluating the gate logic function.
- *Backward implication*: The backward implication, works in the opposite direction than the forward implication by assigning values at the gate input nodes, whenever the input values are uniquely determined by the current output value. This process is slightly more complex

than the forward implication, since in this case there are in general many justification options compatible with the current values that must be determined. Fig. 3.10 is an example of backward justification. To sensitize the OR gate, a 0 is assigned to node D. There is only one way to justify the 0 at node D, assigning a logic 1 to nodes B and C.

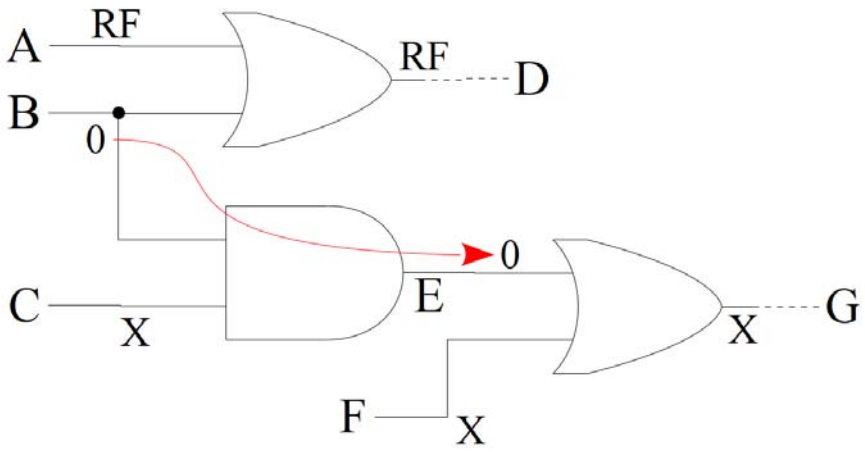


Figure 3.9: Forward implication

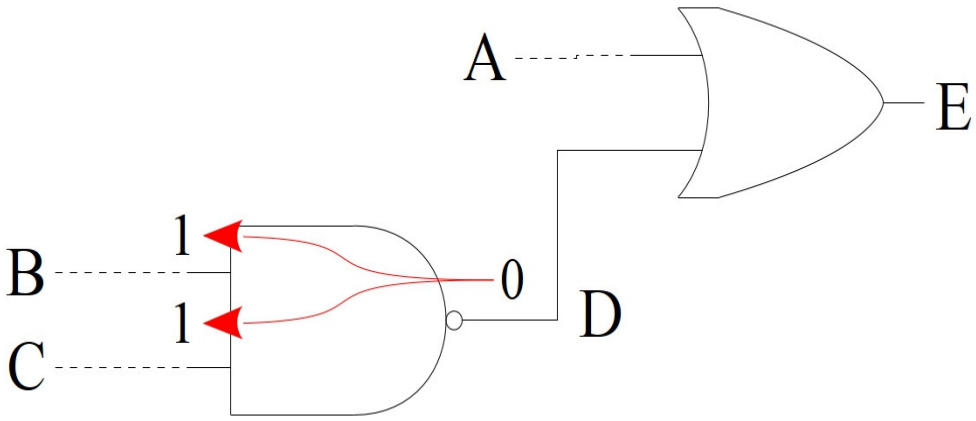


Figure 3.10: Backward implication

3.3.3. Justification

Justification is the process of corroborating the logic values assigned to internal nodes by logic values assigned to the combinational block primary inputs. The sensitization process assigns logic values to the off-path inputs of a gate to allow propagating of a transition through it. However, this is not enough to ensure that the gate can be sensitized. A logic value assigned to an internal node must be justified, guaranteeing that is possible to set this logic value at the internal node, from values applied to primary inputs of the combinational block without any logic conflict.

The backward implication performs a justification limited to the nodes with a value that are uniquely determined. The rest of the nodes with assigned logic values that are not uniquely fixed must be justified by taking some decisions, backtrack if a logic conflict is encountered and trying another option.

The justification process may be required even when a node has been already justified in a previous step. As long as the current value is less restrictive and compatible with the required value, then the value resulting from the intersection between them (the current value and the required value) is the logic value that must be justified. Fig. 3.11 shows an example where an OR gate must be sensitized to propagate a transition through input *A*. The off-path input *B* already has a logic value assigned (X0X). According to the minimum condition criterion, the value required to sensitize the OR gate is 0X0. Then, both values X0X and 0X0 are compatible, and the intersection value is 0, as shows Fig. 3.11. Thus, the value that must be justified in this case is 0, since the currently justified value (X0X) does not ensure that a 0 will be assigned.

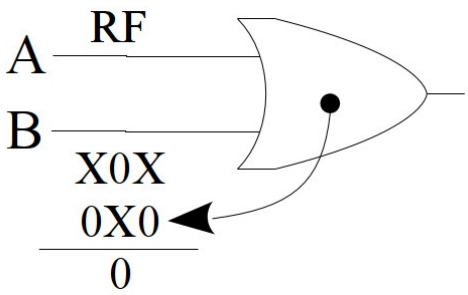


Figure 3.11: Mixing values

Similarly to sensitization, a justification table is constructed for each circuit logic cell. The justification tables must be generated for each possible logic value at the gate output node. Therefore, these tables will be generated by all the values of the logic system except for the fully-undetermined value (X), for which the justification makes no sense. We take a NAND2 as an

example to illustrate the justification tables.

Initially we show the fully-determined steady values (0 and 1) justification included in Table 3.16. As shown in the backward implication example, there is only one way of justifying a 0 at the NAND gate output since this output values requires a non-controlling value (1) at all input nodes. Instead, the justification of a 1 can be accomplished in multiple ways, as shows the Table 3.16. The first two options assign a steady 0 to one of the inputs, but the last two options, shaded in the table, assign semi-undetermined values to both gate inputs. The reason to mark these options is that they can induce glitches. The input combination {X0X, 0X0} is compatible with {FR, RF}, and as shown in Fig. 3.12, if the falling transition arrives before the rising one, a static glitch will be produced at the output. However, if transitions arrive in the reverse order the output takes a steady value. Thus, the real output value is delay dependent. Then, to avoid completely such type of behavior and derive a path sensitization fully free of glitches, the justification combinations susceptible to produce glitches can be discarded. Therefore, similarly to the sensitization criterion, the justification criteria is configurable and can be user defined.

Table 3.16: Justification table. Steady values

Z	A	B
0	1	1
1	0	X
	X	0
	X0X	0X0
	0X0	X0X

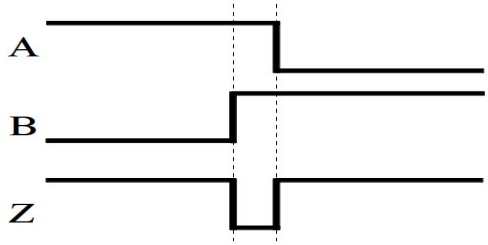


Figure 3.12: Static glitch

If the sensitization criterion used applies exclusively steady values and the justification is also configured to assign only steady logic values, then no more justification options are required for a NAND2 gate. However, for other strategies the justification algorithm requires the justification options for all the output logic values. Tables 3.17 and 3.18 include the justification combinations for the semi-undetermined logic values and for the transitions respectively.

Table 3.17: NAND2 justification options for semi-undetermined values

Z	**A**	**B**
X0X	X1X	X1X
0X0	1X1	1X1
X1X	X0X	X
	X	X0X
1X1	0X0	X
	X	0X0

Table 3.18: NAND2 justification options for transitions

Z	**A**	**B**
RF	FR	1X1
	1X1	FR
FR	RF	X1X
	X1X	RF

3.4. Sensitization Algorithms

The sensitization algorithms determine which paths are capable of propagating a transition through a combinational block, and which input vectors must be applied to the primary inputs to enable the propagation through each path [39]. This objective can be accomplished using various methods.

The framework developed provides two types of sensitization algorithms fully customizable to adapt to each specific analysis or circuit structure. The first algorithm is a stepwise algorithm that exhaustively identifies true paths through a combinational block, exploiting the fact that many paths share common sections. The second sensitization algorithm is intended to be applied to a given structural path previously identified exploiting the fact that all gates to be crossed are known in advance.

3.4.1. Stepwise algorithm

The stepwise sensitization algorithm follows a strategy to exploit the fact that many paths through a circuit share common subpaths. Therefore, each common subpath can be sensitized only once for all paths passing through it, and if a subpath turns out to be non-sensitizable then all paths sharing this subpath are non-sensitizable. In this way a large number of paths can be discarded at once, resulting in an overall algorithm efficiency improvement. This strategy has been extensively used in the delay faults testing domain [36]. However, in this work, this strategy is the base to implement a highly customizable generic algorithm that will allow it to be easily adapted to a wide range of applications. The algorithm walks through the circuit structure following a Depth-First-Search (DFS) strategy, since explores as far as possible along each branch before backtracking to take another branch. Even so, the framework provides another version of the algorithm that uses the breadth-first search (BFS) strategy, where neighbor nodes are processed first before going deeper in the circuit [40]. This variant of the algorithm is less efficient, especially in memory usage, as is well known in graph theory. Nevertheless, for certain special situations it may be the best choice.

The generic stepwise algorithm flowchart is depicted in Fig. 3.13. It starts at an initial node and advances node by node until a final node is reached, or the main routine returns false when processing the current node. If the node being analyzed is a branch point (i.e. various options can be taken from this point), the process state is saved to allow returning to this point and take the next option when the current option is completely explored. The states are saved in a stack structure:

each time that the algorithm has to backtrack it jumps to the last saved branch point. When the algorithm jumps back to a branch point, if the option taken is the last option for this branch point, this point is removed from the stack of saved states.

After checking if the current node is a branch point and the state saved if necessary, the next step is the main routine of the algorithm. This main routine can be totally customized to be adapted to specific requirements. However, the main routine generally tries to sensitize the next gate, and then performs the implication of the assigned values whenever the sensitization finishes successfully. The next step to perform is the justification, although it may be interesting to carry out only sensitization and implication steps, and letting justification at the end instead of at each step.

If the main routine returns false (i.e. a logic incompatibility is found) then the algorithm jumps to the last saved point taking the next possible option, all the paths sharing the current subpath are discarded at once.

If the main routine result is true, then the next node becomes the new current node and the process is repeated until all options have been explored.

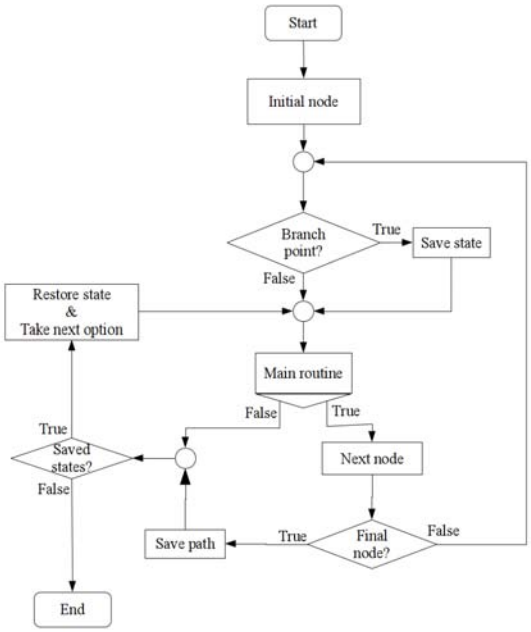


Figure 3.13: Stepwise algorithm

Fig. 3.13 shows a graph representing an example circuit structure. Nodes B, C, G and E are branch points, since the algorithm can take multiple paths. The red line alongside the graph shows how the algorithm travels the structure, going as far as possible before backtrack to the last branch point to take another route.

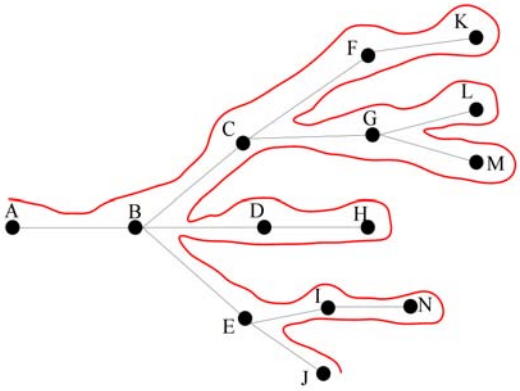


Figure 3.14: Sensitization graph

The generic algorithm can be applied in different ways thanks to the ability of customizing each task. Each flowchart box depicted in Fig. 3.13 can be designed to adapt the algorithm to each circuit analysis requirements where the algorithm can be useful. One of the most relevant ways in which the algorithm can be modified to a specific use, is the ability of working in both directions. I.e., starting at an input node and advancing forward through the circuit until reaching the output nodes, or reversely, starting at an output node and moving backward until reaching the input nodes. Both variants are capable of provide the same results. However, depending on the circuit structure and mainly on the analysis carried out, one version may be more efficient or more suitable than the other.

Table 3.19 gives the operation performed at each flowchart box of Fig. 3.13 for each direction.

Table 3.19: Specific functions for reversible algorithm

	Forward	**Backward**
Initial node	Primary input node	Primary output node
Is final node?	Is an output?	Is an input?
Is a branch point?	Is fan-out stem or multi-output gate or multi-sensitizable gate	Multi-input gate or multi-sensitizable gate
Next node	Output of the sensitized gate	Input of the sensitized gate

There is not a definitive answer to which option is better (forward or backward) since it is highly circuit structure dependent. Forward sensitization exploits the fact that many paths share their initial section independently of the output at which they end. Backward sensitization presents a similar characteristic, since many paths ending at a given output may share their final stretch, and even more, the last gate to an output node is shared by all paths ending at this output.

In the backward version almost all nodes are branch points, except those that are the output of a single input gate, resulting that there are more branch points where the state must be saved than in the forward direction version. However, each time that the process state must be saved, the current subpath is shared by all paths from this point. While this increases the number of operations to store and restore the state, it also increases the amount of paths sharing sections. This fact can be either an advantage or a disadvantage with a strong dependency on the circuit structure.

3.4.1.1. Branch point

As outlined, a branch point is a node from which the stepwise algorithm can follow different routes or use different sensitization options. Therefore, exploring all possible paths requires the state be stored to allow coming back to this point and take another option. The conditions to become a branch point depend mainly on the algorithm direction, but it may also depend on another configurable conditions.

We will refer to Fig. 3.15 to illustrate the branch point concept for different cases, taking node NA as the current node.

In the forward algorithm case, node N_A can be reached either from N1 or N2 after sensitizing gate G1. There are three characteristics that make N_A a branch point.

- Node NA is connected to three gates, it has fan-out is 3, therefore the path may continue through the gates G2, G3 or G4.
- Gate G4 is a half-adder and thus has two outputs. Consequently the path can go from NA to N5 or N6 when traversing gate G4.
- Gate G3 is a complex gate and has multiple sensitization vectors for the input connected to NA, as described in section 3.3.1, adding more options to the paths emanating from the node NA.

The options given by the two first characteristics lead to different structural paths, and are mandatory to explore all possible structural paths. However, the multiple sensitization characteristic only contributes to considering the specific sensitization conditions of each gate traversed, it can be removed as a branch point condition if this issue is not required.

The backward algorithm can arrive to node NA either from the output nodes of the gates G2, G3 and G4 (N3, N4, N5 or N6). Due to the reverse direction, there is only one case where a node is not a branch point: when it is the output of a single input gate (an inverter or a buffer). In all other

cases, there will be at least as many options as number of inputs of the next gate. Then in this case there are two features to consider.

- The current node is the output of a multi-input gate -like G1 in the example- then the path can continue through either of the gate inputs.
- The next gate in the path is a complex gate and there is an interest in getting all possible sensitization conditions.

Following the same previous analysis, only the former characteristic leads to different structural paths, while the other is necessary only when the sensitization conditions are relevant. Thus, depending on the analysis performed, there maybe no interest in exhaustive exploration of all sensitization conditions, and it is only required to determine which paths are sensitizable. Is these cases the branch point conditions can be limited to those that traverse different structural paths.

In the cases where the multiple sensitization vectors for complex gates were considered, the compatibility of each sensitization pattern with the current logic values is verified during the branch point processing. Proceeding in this way prevents storing and restoring the process state in the case that a gate has multiple sensitization vectors but only one is compatible with the current state. Therefore, since this verification must be performed anyway, it entails an algorithm performance improvement and a memory usage reduction.

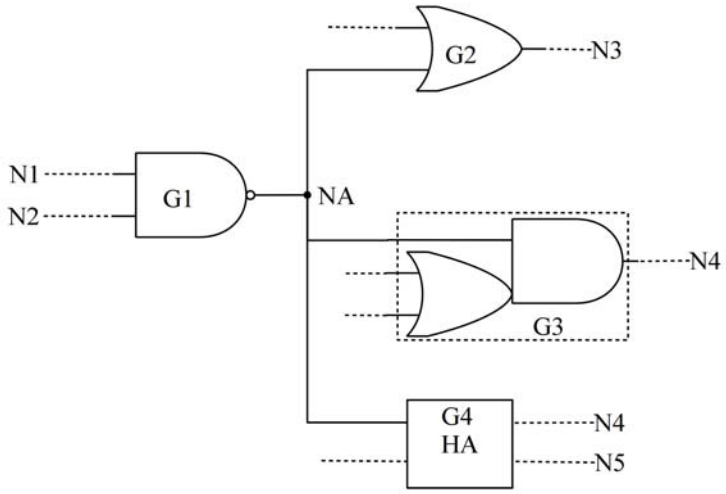


Figure 3.15: Branch point

3.4.1.2. Main routine

The main algorithm routine, is also fully customizable like any other algorithm component. The next gate sensitization in the path is the main task in this step, since the algorithm objective is to identify true paths. Furthermore, apart from the sensitization step, there are other tasks are performed in the main routine, predominantly the implication, i.e., setting all logic values that are uniquely determined by the current assignments, and the justification.

Implication is an important step since it maximizes the number of logic values without taking decisions, helping in an early detection of logic incompatibilities and thus improving the algorithm performance. Therefore, in general there is no reason for not performing the implication just after the sensitization.

The justification is also required to guarantee that a path is sensitizable, but unlike the implication, that improves the performance without introducing decisions, the justification is a considerably more complex task, and requires choosing between multiple options, and keep track of the choices made to allow a backtrack if the current options leads to a logic conflict. Then, there are two options regarding the justification process.

1. Including the justification in the main routine, and perform the justification at each step, or
2. Wait until the end of the path to justify all the unjustified values accumulated during the process.

Both options have advantages and drawbacks. Option 1 has the main advantage that justifying the values at each node can detect non-sensitizable paths in advance without tracing the entire path. Its main disadvantage is that some valid choices at a given point of the process may become incompatible in the next algorithm step, forcing a backtrack and a modification of the option chosen. This is the main advantage of Option 2 since all logic values uniquely determined are already assigned to their nodes, reducing the number of possible justification options, and consequently the number of backtracks required.

Furthermore, the justification at the end may be interesting even when it is less efficient than the justification at each step. Using this algorithm variant, waiting at the end of the path to perform the justification may be useful in some cases, such as determining the probability of activation of a given path, i.e., computing how many input vectors allow sensitizing the path. An exact computation of this metric, is usually an unaffordable task except for small circuits although it can be approximated. Therefore, if all justification decisions are delayed until the algorithm reaches the

path end, then all logic values assigned before the justification step are necessary to guarantee the path sensitization, and none of the values that requires a choice has been already settled.

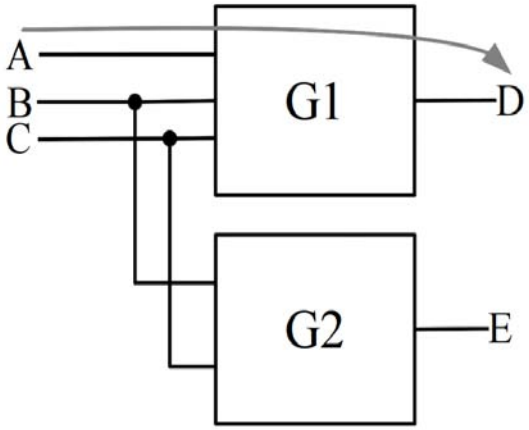


Figure 3.16: Setting values

3.4.1.3. Implication

The implication step is not performed immediately after each value assignment, instead the node is marked and subsequently the implication applies to each marked node. This procedure reduces the number of implications to be performed. Fig. 3.16 shows an example where a path through gate G1, represented by an arrow, requires assigning values to nodes B and C. If the implication is performed immediately after each assignment, the forward implication procedure through gate G2 will be performed twice, first for node B assignment and then for node C.

Besides the forward implication, marked nodes are also candidates for the backward implication and justification. While the forward implication is a direct process requiring exclusively the evaluation of a gate logic function taking the current input values, the backward implication is a quite more complex process that requires determining which justification options are compatible with the current logic state. Therefore, the implication is performed in two stages, starting by the forward implication followed by the backward implication.

Fig. 3.17 shows the implication procedure flowchart indicating the two stages of the process. The algorithm starts selecting the gates to be evaluated, avoiding repeated propagation, and then perform the propagation of these gates. If some logic conflict is detected then the process ends, otherwise the algorithm continues with the backward propagation.

In the backward implication, for each marked node, the algorithm first determines how many of

the justification combinations are compatible with the current state. There are only two possible results for this process.

- There is only one justification option, or only one of them is compatible with the current values. Then, this node will be justified immediately.
- There are more than one compatible justification options. In this case the node is added to a list of unjustified nodes to be justified later during the justification step. It may happen that some logic values are uniquely determined, if the compatible options share a value at a given node. Table 3.20 shows the justification table of a gate OA12 when the output value is 1. As highlighted in the table, all options share the value of input C. Therefore, this value is uniquely determined and will be assigned although multiple justification options exists. In the example, the value of input C is shared by all options, but even if there is no value shared by all options, a shared value among the compatible options may exists.

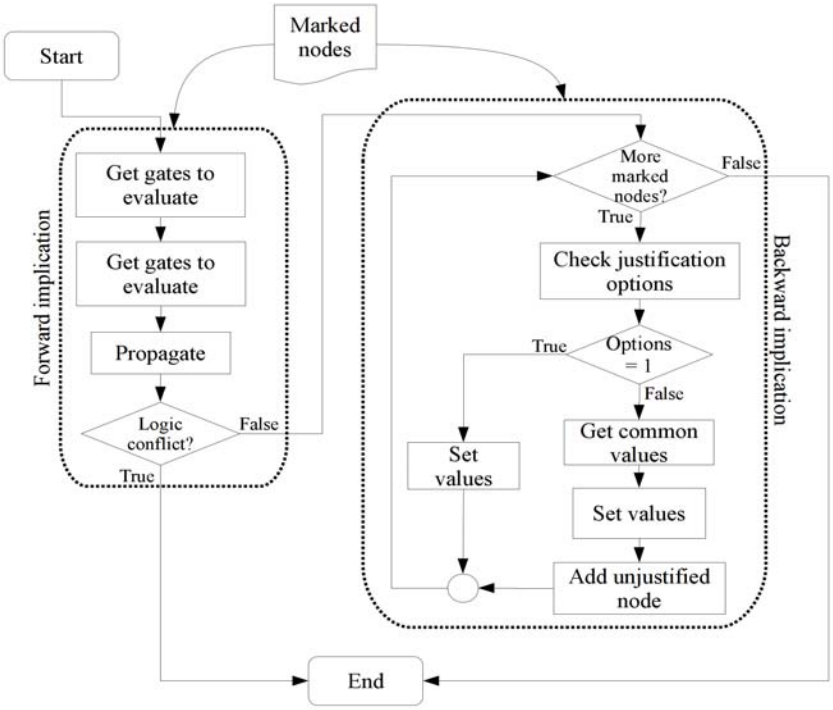


Figure 3.17: Implication flowchart

Table 3.20: Justification table for OA12

A	B	C	Z
1	X	1	1
X	1	1	
1X1	X1X	1	
X1X	1X1	1	

The case where none of the justification options is compatible will never occur, since the logic system and the forward implication avoid assigning an unjustifiable value to a node. To illustrate this situation assume that the algorithm must to assign a value at the output of an AND2 gate, Table 3.21 gives its justification values for the output values 0 and 1. Then if a 0 can be assigned at the output this ensures that both inputs are not 1 simultaneously. If this would have happen, the forward implication would have assigned a 1 to the output node, avoiding the initial assignment.

Table 3.21: Justification table for AND2 gate

A	B	Z
0	X	0
X	0	
0X0	X0X	
X0X	0X0	
1	1	1

3.4.1.4. Justification

Justification is the most complex step, it involves taking decisions, keeping the trace of the options chosen to allow a backtrack if the current choices lead to a logic incompatibility, and giving the opportunity of trying other combination of options.

The implication step determines which justification options are compatible with the current logic values creating a list of unjustified nodes and their compatible options. This is the starting point of the justification algorithm.

Fig. 3.18 Flowchart shows how justification algorithm loops until all unjustified nodes are justified as long as there are no logic conflicts. At each loop the algorithm tries to justify one

unjustified node, starting by the first compatible option. Early identification of compatible options carried out by the implication process, simply compares the current logic values with each justification vector but does not perform the implication of these values. Therefore, the fact that an option was identified as compatible does not guarantee its compatibility. However, it provides a first estimation of which justification options must be attempted.

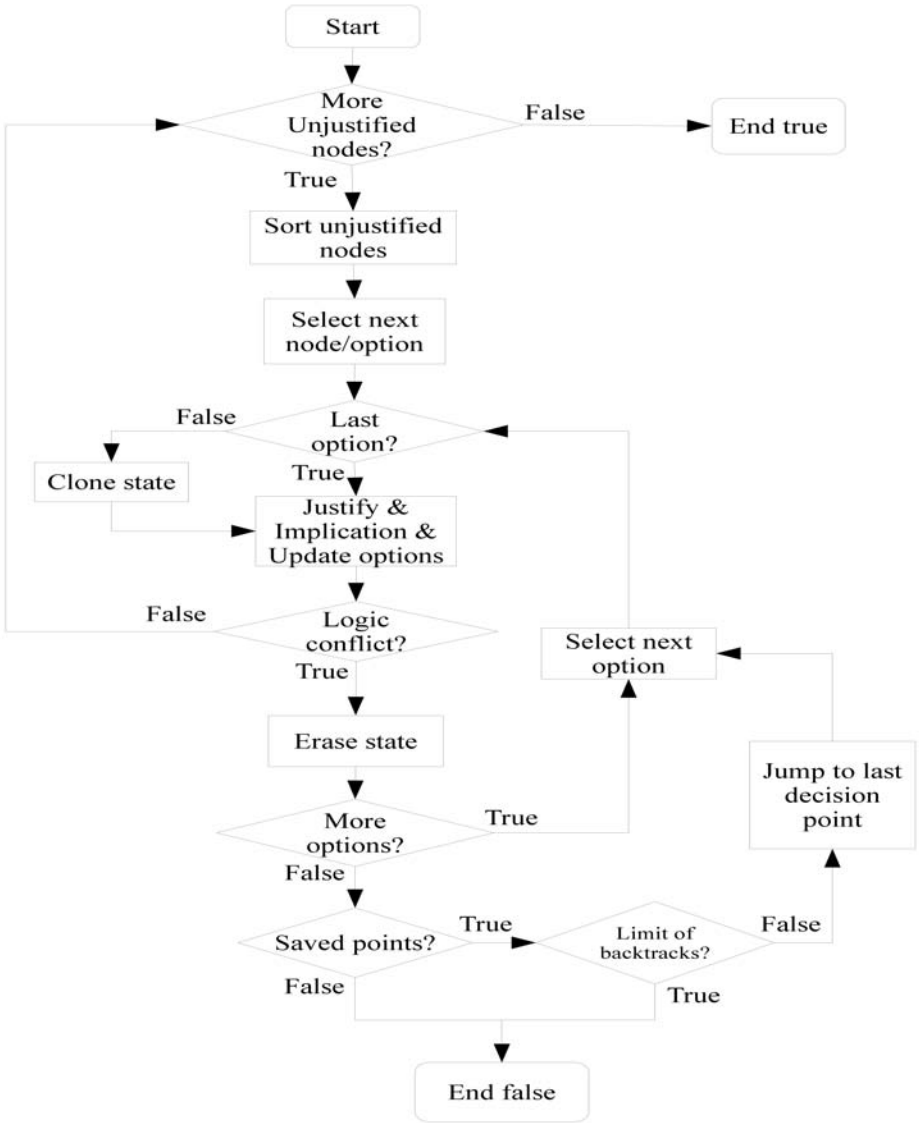


Figure 3.18: Justification algorithm flowchart

First, the unjustified nodes are sorted based on their number of compatible options starting the justification by the nodes with less compatible options. A node with many options offers more

flexibility than a node with only two options, therefore in general the strategy of justifying first the nodes with less options reduces the number of backtracks. If two unjustified nodes have the same number of compatible options, then the node with higher deep goes before. The deep of a node is the distance in number of gates from this node to any primary input. Therefore a node with a higher deep usually requires more complex justification due to its higher distance to the inputs, improving the performance if it is justified first. Since all unjustified nodes must be justified, the more efficient way is to begin by the more difficult nodes.

During the framework development various options were tested using benchmark circuits to determine which strategy provides better results in general. The experiments performed demonstrated that sorting the unjustified nodes is significant, since omitting this step decreased the algorithm performance in all tested cases. The algorithm performance was also shown to depend on the criterion used to sort the unjustified nodes. After examination of various unjustified nodes sorting options, we found that the sorting criterion based on the number of compatible options provides the better results in general. However, since the framework development was focused on the ability of customizing each algorithm, the criterion used to sort the unjustified nodes can also be user-defined.

Besides sorting the unjustified nodes, the compatible options for each one are also sorted. They are sorted in function of the number of logic values to be set, beginning by the options with fewer requirements. Since each logic value assigned represents a new unjustified node, except if the node is a primary input, the best strategy is to minimize the number of assignments.

At each loop iteration the algorithm tries to justify the next node in the queue, beginning by the first option. The current state is cloned to allow backtracking and take another option if necessary, except if the current option is the last option for this node. In this case there is no reason to create a copy of the state.

The following step is the main routine of the justification algorithm, i.e. setting the logic values required to justify the current node. Furthermore to set the logic values, this process performs the implication of the assigned values, adding to the queue the new unjustified nodes that result from the current justification. If during implication there is no logic conflict, then the unjustified node queue is updated. This process verifies again the compatible options for each unjustified node. Due to the values assigned to justify the current node, the compatible options may change. The expected results for the updating step of each unjustified node are as follows:

- The compatible options remain unchanged.
- There are fewer compatible options. Some of the options previously compatible became incompatible. If the compatible options are reduced to only one, then the implication procedure is applied to this node and it is erased from the queue of unjustified nodes.
- The node is already justified. The values assignments performed to justify a given node can implicitly justify another node. In this case the node is dropped from the queue.

The case where there are no compatible options will never occur because this situation would create a logic conflict during the implication step of the values that avoid this justification.

Performing this process each time that a node is justified may appear very time consuming at glance, however the experiments performed with benchmark circuits show that this process improves considerably the algorithm performance, specially when the unjustified nodes are sorted again after the update.

Once the main routine has finished, if no logic conflict is detected, then the algorithm continues with the next unjustified node, until all nodes in the queue have been processed. However, if some task of the main routine detects a logic conflict, then the rest of the main routine is skipped and the current state is erased. After that, if there are more options for the current node then the algorithm repeats the main routine using the next option. Otherwise, if all options for the current node have been explored then the algorithm discards the current state and backtracks to the last node that offers an alternative option, if exists. In the case that there are no alternatives, the justification algorithm finishes with a false return value.

Although the justification is successfully completed, the stored states must be kept because a justification that is currently compatible might become incompatible due to the next gate sensitization, requiring discarding the current justification options and trying another one.

Since justification can be extremely time consuming due to the huge number of combinations than can exist for large circuits, this process must be limited. Justification is limited by setting a backtrack threshold. Each time that the algorithm discards the state and backtracks to a previous node for another option, a backtrack counter is increased. If the backtrack counter reaches the threshold, then the justification algorithm ends. The backtrack threshold is an adjustable parameter of the justification algorithm.

3.4.2. Full path algorithm

The full-path sensitization algorithm is another strategy included in the framework. This algorithm requires a structural path previously identified instead of performing an exhaustive path identification through the entire combinational block. The stepwise algorithm is focused on identifying as much as possible true paths by processing a given circuit section, exploiting the fact that many paths have sections in common. However, depending on the nature of the analysis carried out, the interest does not rely on an exhaustive path identification. Moreover, the exhaustively exploration of all true paths in large circuits may be very time consuming. In these cases the sensitization can be focused on a set of structural paths previously selected. Then, this algorithm tries to sensitize a given structural path.

Additionally, this algorithm may be used to refine the results obtained by the stepwise algorithm. For example, in a path with gates having multiple sensitization vectors, maybe not all combinations can be justified due to the backtrack threshold. However, all sensitization combinations could be true. In a case like this, if all sensitization combinations are of interest then the full path algorithm can be applied to verify the sensitization combinations discarded by the stepwise algorithm due to the backtrack limit, getting more precise results.

While the stepwise algorithm takes advantage of the fact that many paths share common subpaths, the full path algorithm exploits the fact that all gates that must be traversed are known in advance. Therefore, to exploit this feature the algorithm starts by setting all logic values that are common to all possible sensitization options to sensitize the entire path.

As shows the flowchart of Fig. 3.19, the first step of the algorithm is a loop that applies for each gate in the path. In each loop iteration, if the processed gate has only one sensitization vector then this sensitization vector is applied. Otherwise if the gate has multiple sensitization vectors only the logic values that are common to all sensitization vectors are assigned, if any.

To illustrate this loop, Fig. 3.20 is an abstract representation of a path where each box represents a gate and the number inside is the number of sensitization options for this gate. As shows the figure the third gate has three sensitization vectors, but sharing a logic 1 at first off-path input. Therefore, the first loop of the algorithm will set the sensitization vectors for the second and fifth gate, since they have only one option, and the shared logic value of the third gate. There are two gates, first and fourth, that have two sensitization vectors without shared values, hence no value will be assigned in these cases. Even without implication or justification, there exists the possibility of logic conflicts

during this process, since it may happen that two gates share some off-path input nodes. In this case the algorithm ends, and the structural path is a false path.

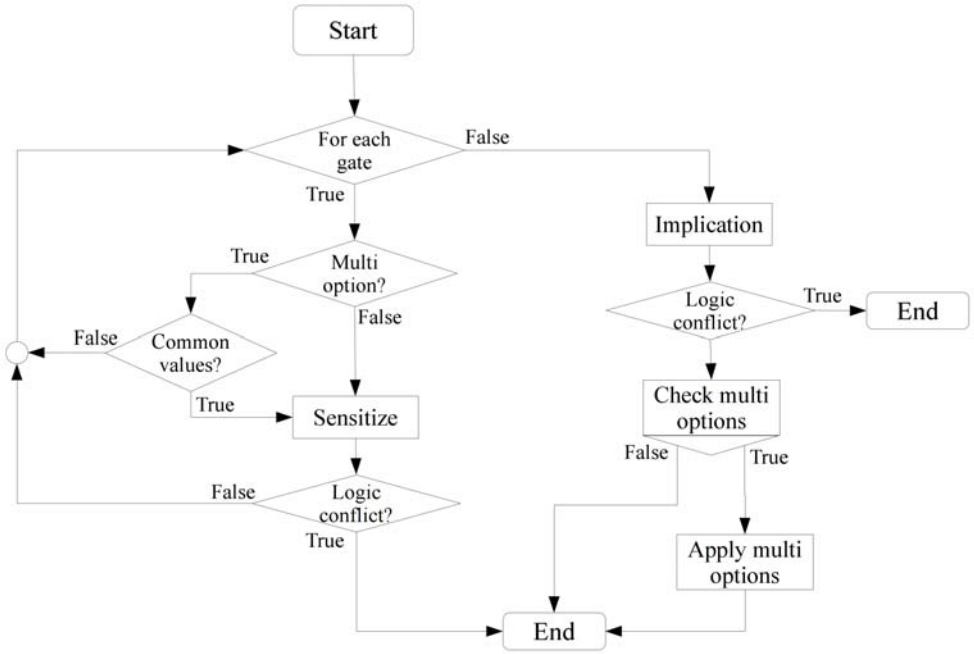


Figure 3.19: Flowchart of full path sensitization algorithm

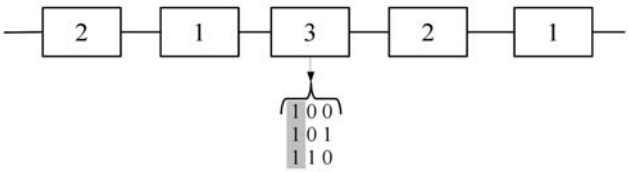


Figure 3.20: Full path example

If the first loop has ended successfully the next step is the implication of all logic values set in the previous step. If a logic conflict is detected during the implication, then the process finishes with a negative result.

Once the implication ends successfully, all logic values required to sensitize the path that do not imply decisions have been set. The following step verifies again the gates with multiple sensitization options; this is done because the logic values assigned during the implication may prevent some sensitization options due to logic conflicts.

The flowchart for this algorithm section is depicted in Fig. 3.21. As shown in the figure, a loop checks each multi-option gate, verifying which sensitization options are compatible with the current logic values. Hence, three cases can occur:

- None of the sensitization options is compatible with the current values. Therefore, the path cannot be sensitized.
- There is only one option compatible. Hence, the corresponding values are set and the implication is performed. If no logic conflict is detected the algorithm goes to the next gate, otherwise the path is false.
- More than one option is compatible. However, if not all options are compatible, then some values can be shared between the options. Otherwise the shared values would have been detected in the first stage. Therefore if there are shared values, the algorithm performs the assignment and the implication.

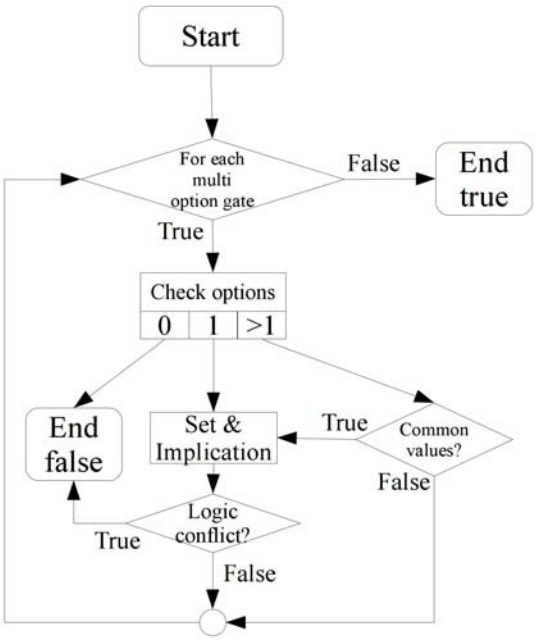


Figure 3.21: Flowchart of multi-option verification

Finally if logic conflict is detected during this second step then there is no way to sensitize this path. On the other hand the algorithm passes to the final step.

The final algorithm step completes the path sensitization trying the sensitization options that are still compatible, justifying all values. However, similarly to the stepwise algorithm, the full path algorithm is customizable. Thus, the final step can be performed in two ways, depending on if the interest is simply determining if the path is sensitizable, or if an exhaustive examination of all sensitization combinations is preferred.

Both methods are quite similar, except that in the former case the algorithm ends when one combination proves that the path is true, rather than exploring all combinations.

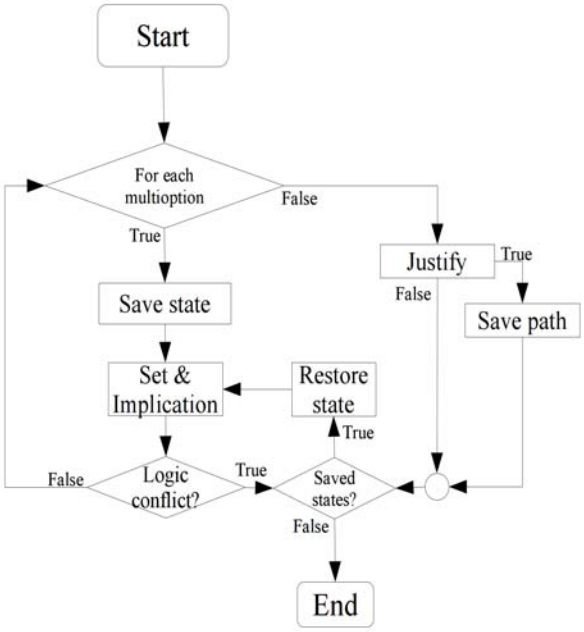


Figure 3.22: Flowchart of multi-option assignment

The flowchart for the version that explores all combinations is shown in Fig. 3.22. The process uses a stepwise strategy to avoid repeated work. At each loop iteration the algorithm saves the current state, assigns the values for the current gate and option, and performs the implication.

If a logic incompatibility is detected, the current state is discarded, the last saved state is restored and the algorithm tries the next option for the gate. The process goes on, until all gates are sensitized or all combinations result incompatible. When all gates have been sensitized without logic incompatibilities, then the final process, the justification of all unjustified nodes, starts.

Finally if the justification can be performed, then the path is saved. In all cases, the algorithm jumps back to the last saved state to explore the remaining sensitization combinations.

3.4.3. Path graph creation

Following with the modularity and flexibility philosophy of the framework a set of algorithms to create graphs representing the paths through a combinational circuit are provided [41]. Subsequently the graph may be used by other algorithms to accomplish multiple objectives. Various path graph creation strategies are available; the graph can be created either from inputs or from outputs, and can be divergent or convergent depending on the specific requirements.

Figs. 3.23 to 3.25 show three path graphs for the 2-bit carry-bypass adder of Fig. 3.1. Fig. 3.23 shows a convergent graph for the paths starting at input node a_0 and ending at output node C_{out}. In this structure each circuit node appears only once being a direct representation of circuit structure. However an exhaustive path analysis is difficult to perform on this structure and can be solved with a divergent graph.

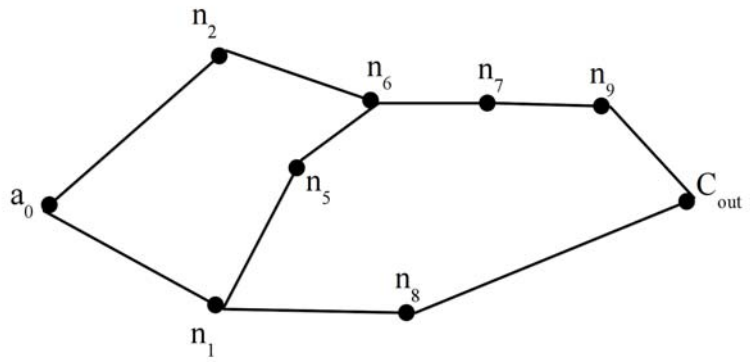


Figure 3.23: Non-divergent In-Out graph

The graphs in Figs. 3.24 and 3.25 are divergent where each structural path has its own branch by repeating the shared sections. In the first case the graph emanates from input node a_0 and the graph in Fig. 3.25 originates from the output node C_{out}.

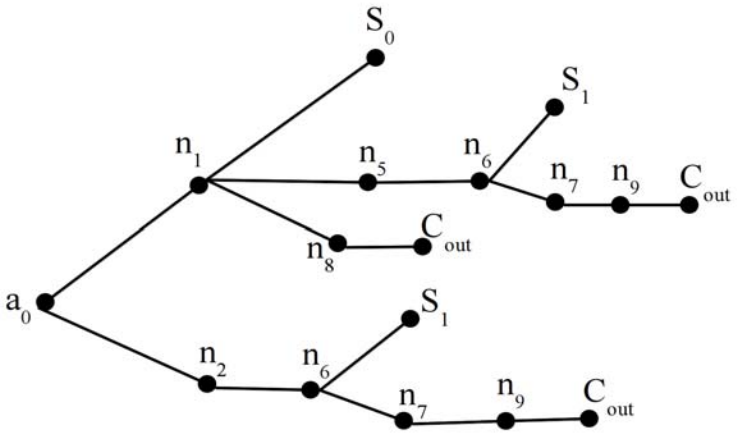


Figure 3.24: Path graph from input

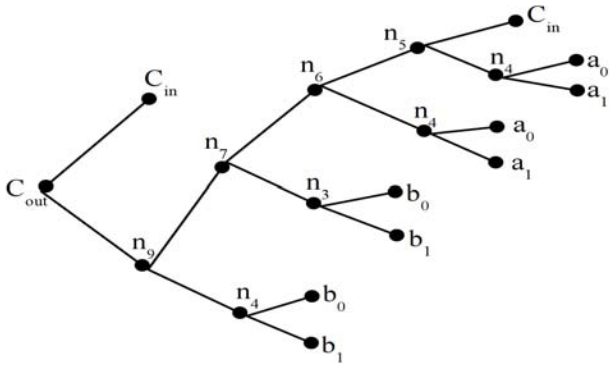


Figure 3.25: Path graph from output

Divergent graphs require considerably more resources than convergent ones, however a divergent graph provides advantages for algorithms traversing the graph to perform some task, like computing the propagation delay through each path. A divergent graph keeps all path information simplifying the task of estimating the propagation delay through each structural path. The same reasoning can be applied to other analysis like SET propagation.

Actually the stepwise sensitization algorithms follow this structure without explicitly generating the graph. Each time that a branch is proved non-sensitizable and the algorithm discards the branch then this is equivalent to prune the branch from the graph, removing all ramifications hanging from that branch. However, although the stepwise algorithms does not create explicitly the graph for the circuit under test, there are cases where the graph creation is a useful strategy or even a requirement. Once a path graph has been created the sensitization techniques can be applied to prune some branches and simplify the graph.

In following chapters will show some path graph applications.

Chapter 4: Preprocessing techniques and Framework structure

Many algorithms have been developed over the years to identify the paths through a combinational circuit, each one improving the performance of its predecessors, while adopting its own data structures to keep the path information and the current state of the process consistent [36] [42][43]. However, the increase of algorithms efficiency together with the increase of processors computational power, can not fight against the fast increase of circuit complexity, due to the exponential nature of the problem.

An exhaustive identification of all functional paths is unaffordable for most of today circuits since the number of paths grows exponentially with circuit size, making this a problem computationally prohibitive. This task is the typical NP-complete problem identified during the early years of test pattern generation for digital circuits [44]. These issues are not exclusive of paths identification problem and are extensible to other kinds of circuit analysis algorithms.

This work presents an alternative solution to circuit complexity analysis that instead of trying to increase the algorithm efficiency it reduces the circuit complexity, without information loss regarding the circuit structure. The techniques presented in this work are general and can be applied not only to the critical path problem, but also to other domains like circuit design improvement through area or power reduction [45].

4.1. Simplification techniques

The framework developed in this Thesis provides a set of tools intended to simplify the circuit under test before applying any posterior analysis algorithm. Such preprocessing techniques are aimed at reducing the circuit complexity with the objective of increasing the algorithm speed when applied to highly complex circuit designs. However, not only the speed is relevant, some circuit designs are unaffordable even for the most efficient algorithms if applied without a previous simplification stages. Preprocessing techniques also allow identifying specific issues of a design that might be improved by redesigning a portion of it. Moreover, preprocessing techniques might be also used to isolate specific circuit regions, allowing to focus a specific analysis while discarding unrelated elements.

The techniques exposed are classified into the following categories:

- Partitioning: These techniques are intended to separate regions of a combinational block to be treated independently in a plain structure.
- Encapsulation: This procedure wraps certain circuit parts into sub-circuits and treats them as a single block, hiding the internal structure. Encapsulation can be applied iteratively creating a hierarchical structure.

4.2. Partitioning techniques

4.2.1. Auxiliary routines

4.2.1.1. Touch related nodes algorithm

The touching related nodes algorithm is a simple strategy to determine which nodes can be directly affected by a given node, or which nodes can directly affect such node value. This algorithm is just an auxiliary routine used internally by the partitioning techniques. Depending on the direction in which the nodes are touched, we refer to *forward touch* when the nodes are touched towards the outputs. Otherwise if the nodes are touched towards the inputs we refer to a *backward touch*. Fig. 4.1 shows an example of both kinds of touch, from the node *A*.

Solid arrows represents the *forward touch:* all the output nodes of the gates having node *A* as input are touched. This includes all nodes that can be directly affected by a change in the logic value of node *A*. The region of a circuit identified by this technique is referred as the *output cone* of node *A*, i.e., all the nodes that can be reached from the current node.

The *backward touch* is represented by dashed arrows in Fig. 4.1. In this case all the input nodes of the gate from which node *A* emanates are touched. Thus, touching all nodes such that a change in their value can directly affect to the node *A*. In this case the region of a circuit formed by all nodes from which the current node can be reached, is referred to as the *input cone* of the node A.

DEFINITION 1: The *output cone* of a node is the set of all nodes that can be reached from this node.

DEFINITION 2: The *input cone* of a node is the set of all nodes from which the considered node can be reached.

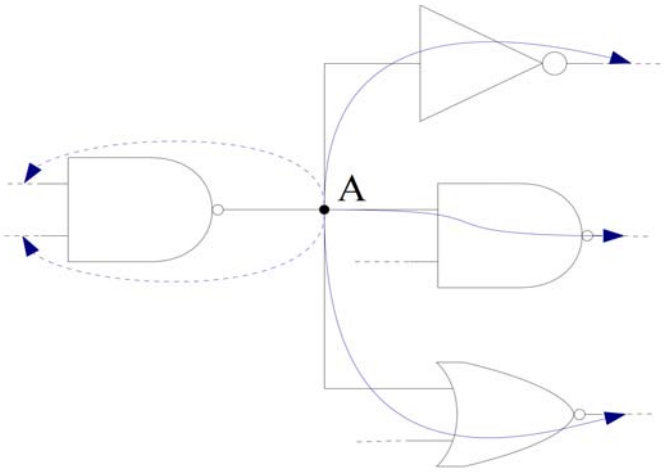


Figure 4.1: Touch nodes algorithm

In both cases, the touch is propagated forward or backward from each node that has been touched. This process continues until reaching an end node. The end node is in general an output node for the *forward touch* and an input node for the *backward touch*. However, any other node can be defined as the end node of the algorithm if the specific process wants to be restricted to a limited circuit region. Both techniques are used individually or in combination in many of the circuit preprocessing techniques available in the framework developed.

4.2.2. Separate independent sections

In general, not all inputs and outputs of a combinational block are related to each other, making it possible to analyze and/or optimize unrelated circuit regions separately. This method consists in finding combinational circuit sections that are fully independent from the rest of the circuit. I.e., any logic value applied to any of a section inputs, will never impact the gates and node values outside such section, under any condition. The success of this technique is very circuit dependent, because

many circuits have no independent sections. However, if independent sections are present, in a given circuit, each section can be treated as an individual circuit, and processed concurrently. Fig. 4.2 shows an example of a very simple circuit with two independent sections. Input nodes N4 and N5 have no relationship to the rest of the circuit except with the node N8 that is only affected by N4 and N5. Thus, the gate G3 and its nodes are totally independent of the rest of the circuit, and may be treated independently.

Typically, circuits with independent sections have one large section that includes almost the entire circuit, plus some very small sections. For this reason, this technique usually has a small impact on the circuit complexity reduction. The advantage is that the algorithm to separate these sections is very simple and fast.

Although the circuit complexity simplification can be usually slight, its simplicity makes it worthwhile. Its impact on the algorithms efficiency is mainly a memory usage reduction to store the circuit structure and the state information. Usually, the algorithms use an array to track each circuit node state during analysis. Although the node number reduction achieved by this technique may be small, its effect is magnified if a large number of state arrays must be stored, or if the algorithms constantly make copies of these arrays during the circuit processing.

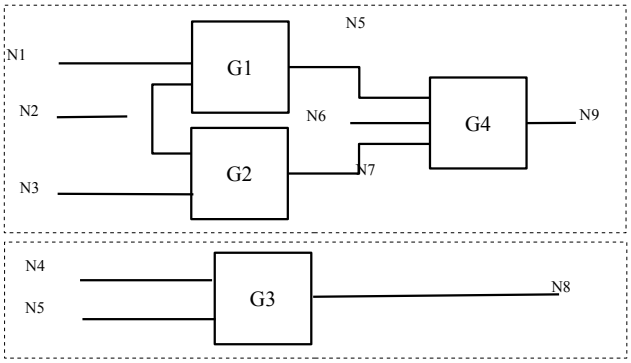


Figure 4.2: Independent sections

To determine the independent sections of a combinational block, the tool leans on the related nodes touch algorithms. Starting at the first block input node, all nodes are touched in both forward and backward direction. Touch is propagated until there are no more pending nodes to be propagated. When this process ends, all the combinational block nodes that have not been touched are fully independent from the circuit region covered by all touched nodes. The same process repeats iteratively, starting by the first input node that was not touched in the previous step, until there are no remaining untouched nodes.

Table 4.1 gives the results of this technique applied to some benchmark circuits, showing the number of nodes in the main circuit section, the number of nodes of the secondary sections, and the percentage of nodes outside the main section with respect to the total number of nodes.

Table 4.1: Independent sections

Circuit	# of nodes		Percentage of node outside the main section
	Main section	Secondary sections	
c880	209	10	4.57%
c2670	389	167	30.04%
c5315	656	57	7.99%
c7552	860	20	2.27%
b12	656	19	2.81%
b17	11833	141	1.18%

As shown, in general the percentage of the circuit that can be separated is small, with some exception like the ISCAS c2670. Moreover, in all cases the runtime required to separate the sections is absolutely negligible, reaching some tenths of a second in the worst case.

4.2.3. Partitioning by Output

In many instances, the interest of a specific circuit behavior is related to a specific output node, or a subset of output nodes. For example, the soft-error-rate (SER) analysis is a candidate, since the interest relies on estimate the probability that a soft-error will be produced at a given output node. In these cases, the circuit structure can be pruned eliminating all the circuit regions that have no impact on the output of interest behavior. Such a situation is equivalent to selecting the input cone of the selected output node.

In general, sensitization algorithms propagate forward the logical values assigned to the nodes to maximize identifying the logic values that can be uniquely determined to detect logical conflicts as early as possible and improving the algorithm efficiency. This process is called *forward implication* in the literature. Then, discarding the circuit regions outside the input cone, avoids propagating the logic values through gates that are irrelevant for the analysis being carried out. In addition memory requirements to store the circuit structure are reduced together with the state information used by the algorithm. Since the output cones of each output are independent, they can be processed concurrently.

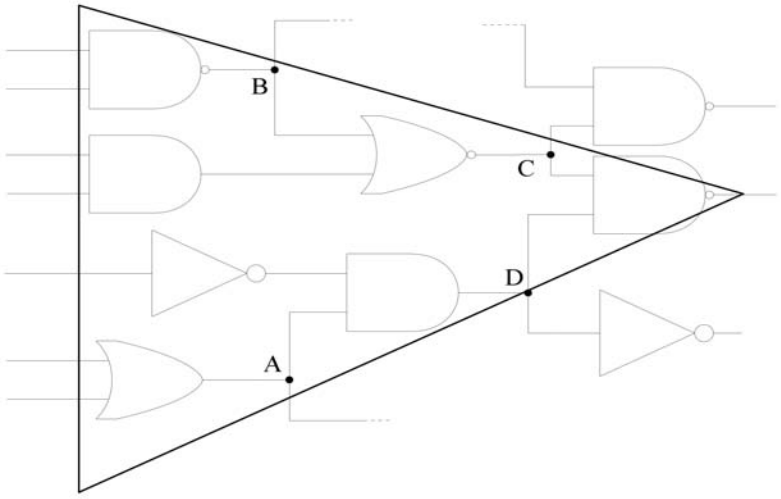


Figure 4.3: Partitioning by Output

Fig. 4.3 shows the schematics of a circuit portion with a showing line the input cone of one of its outputs. As shown, the circuit the fan-out of the nodes A, B, C, D is two, however after selecting the output cone, the fan-out of all these nodes reduces to one. The nodes fan-out reduction has in general a significant impact over the computation time required to process the circuit, and reduces the number of nodes where the algorithm must take a decision.

The input cone of a given output node is easily identified by using the backward touch technique starting at the output node toward the input nodes. The result is a set of touched nodes. If the interest lies of more than one output node simultaneously, the equivalent circuit can be easily identified by performing the intersection between the set of each output touched nodes. Once after this step, all non-touched nodes are discarded, as well as all the gates related to these nodes.

As explained above, the sensitization algorithms may work in both directions, therefore when the sensitization is applied to an input cone of a given output node, is the kind of problem where the backward sensitization may be more suitable than the forward version.

4.2.4. Partitioning by Input

The case of partitioning a circuit by an input is slightly different from that of an output, in the sense that the previous technique involves uniquely the input cone of the considered output node. However, selecting the circuit region related to a given input node involves not only its output cone, but also all nodes required to justify the logic values of the off-path inputs of the gates belonging to the output cone. I.e., the input cone of each node located within the output cone of the input node selected.

Fig. 4.4 shows the same circuit portion used to illustrate output selection technique, where the input selection for the node E is represented. The dark shading area identifies the node output cone, All the nodes required to identify the paths starting at this node must then be added as shown by the light shading area that involves the nodes required to justify the logic values imposed to sensitize paths through the output cone.

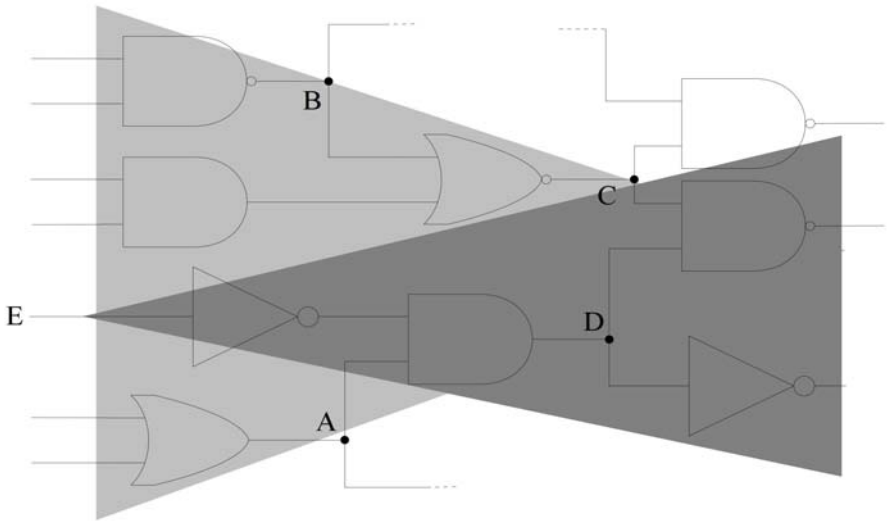


Figure 4.4: Partitioning by input

In general, the input selection technique involves a larger region of the circuit than the output selection technique. In some cases the entire circuit can be required.

By using this technique, the processing algorithms avoid propagating values through regions of the circuit that are unrelated to the area of interest. However, there is a way to simplify the processing of the paths starting at this node, by considering both regions represented in the figure in different manner: all the nodes that can be crossed by a path are located inside the output cone, while the other region is involved only in the justification of the logic values required to sensitize the paths. We refer this last area as *Justification region*. Then both regions, the output cone and the justification region may be treated separately.

Furthermore, if this method is combined with the technique of separating independent sections, then in some cases the justification area can be divided into various fully independent blocks, even when the whole combinational block cannot be divided in independent sections. That is because justification region encompasses only a portion of the circuit outside the output cone. Referring to the example of the Fig. 4.4, it is easy to verify that the justification region can be separated in two

independent sections. One section include one single gate, and the other with three gates.

DEFINITION 3: Justification region, is a combinational block region that during the current objective is used only to justify the logic values assigned to nodes inside the region being analyzed until the input nodes.

4.2.5. Partitioning by Input-Output

The previous techniques can be combined to get the highest efficiency. To determine the paths that can cross a combinational block from a given input toward a given output, both cones can be used to delimit the region through which any path of interest travels. This allows concentrating the effort in the specific region of interest, discarding unrelated gates, while dividing each block part that can be treated independently from the rest of the circuit.

Fig. 4.5 shows an abstract representation of a combinational block where the input and output of interest have been highlighted. To identify the common paths, the combinational block is divided in various regions.

Fig. 4.5 shows the output cone of the input considered, and the input cone of the output node of interest. The output cone corresponds to the area with vertical stripes, and the input cone have been marked with horizontal stripes. The area defined by the intersection of both cones delimits the path region, i.e., all paths from the input to the output considered pass exclusively through this region.

DEFINITION 4: Path region, is the region of a combinational block that covers all nodes through which a path can pass for a specific problem defined by an input and/or an output.

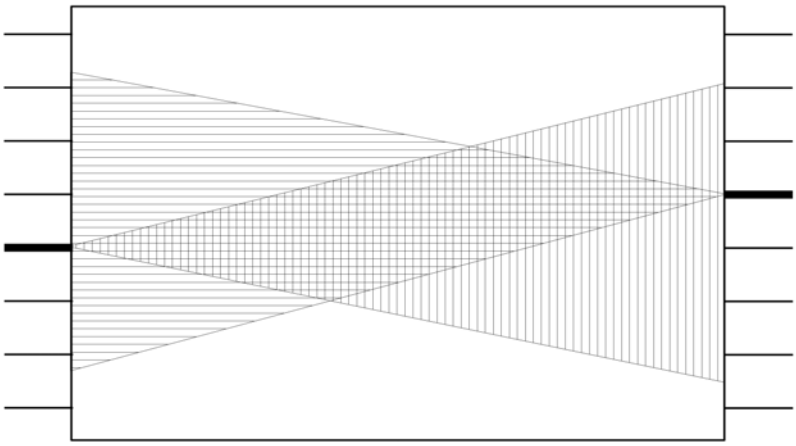


Figure 4.5: Partitioning by Input-Output

The partitioning process of a combinational block may be described in an analytical way using set theory:

- If n_i is a node of a combinational block,
- then $C=\{n_i\}:0\leq i\leq N_{nodes}$ is the set of all nodes of the combinational bloc.
- $CONE_{In}(n_i)$ = set of all nodes inside the input cone of the node n_i.
- $CONE_{Out}(n_i)$ = set of all nodes inside the output cone of the node n_i.

Thus, the difference between the set of all nodes and the input cone of an output node Z (4.1), determines the region that can be discarded.

$$C \setminus CONE_{\text{In}}(Z) = Ur(Z) \tag{4.1}$$

Then, $Ur(n_i)$ is the unrelated region, i.e., the set of all nodes non related with the node output n_i.

As has been advanced, the intersection between the input cone and the output cone defines the path region (4.2).

$$CONE_{\text{Out}}(A) \cap CONE_{\text{In}}(Z) = P(A, Z) \tag{4.2}$$

Where P(A, Z), is the *Path region* between the nodes A and Z.

All the nodes lying within the input cone of the output node, that do not belong to the path region, define the justification region for these pair of input and output nodes (4.3).

$$CONE_{\text{In}}(Z) \setminus P(A, Z) = J(A, Z) \tag{4.3}$$

Where J(A, Z) is the *Justification region* of the nodes {A, Z}.

Finally the combinational block can be partitioned into three disjoint regions, the path region, the justification region and the unrelated region. Once the combinational block has been partitioned, the unrelated region can be discarded, since it is irrelevant for the problem to be solved. The path region and the justification region are both required to determine the sensitizable paths through the circuit, however they are treated separately.

The justification region can be preprocessed using the algorithm to identify independent sections. As already mentioned, although the combinational block under test may not have independent sections, the justification region represents only a part of the whole circuit and might have locally independent sections connected through gates that are located outside the justification region, making the entire circuit to have no independent sections.

The fact that the justification region can be divided in independent sections helps to simplify the justification process, since each section has its own justification structure. This, in turn, reduces the memory required by the structures and allows processing each one concurrently. The justification algorithm uses a tree to keep track the nodes that must be justified and the options available to justify each node. Similar techniques are used generally in others path finding algorithms.

To illustrate the process, assume that during a sensitization of a path there are 3 nodes to be justified (n1, n2, n3), and each one can be justified in two ways (a, b). Fig. 4.6 shows an abstract representation of a justification tree for this situation. Before identify which justification options are incompatible between them, the tree must keep all possible justification combinations. If some of the tree nodes are unrelated, their justification options can never be incompatible between them. This situation cannot be detected during the tree construction without including an additional process. However, if the justification region consists of separated regions treated independently, then unrelated justifications are located in different justification structures. Fig. 4.7 shows the same case than Fig. 4.6 assuming that nodes n1 and n3 belong to one section and n2 to an independent section. As shown in the figures, the resulting justification trees after separating the independent sections are simpler than the original one, requiring less memory to store it, and even more, each

tree can be computed simultaneously.

An additional advantage of circuit partitioning is that some sections can be pre-justified. This is: before the sensitization step, the justification sections can be processed, identifying in advance combinations of output values that are incompatible. This allows discarding directly such combinations in the sensitization step. The suitability of this technique depends on the block to pre-justify size. Depending on the block size, the justification can be fully performed, determining the validity of each combination, or partially identifying only some combinations as compatible or incompatible, those that are easily identified. However, if the block is too large this technique can be unaffordable.

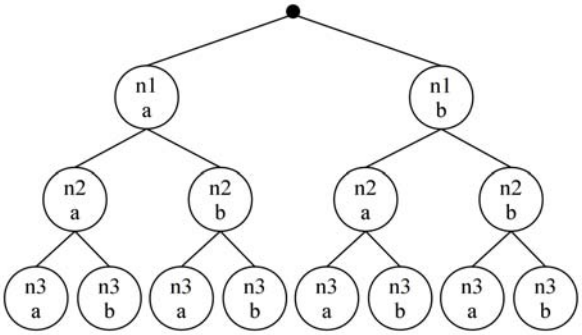


Figure 4.6: Justification tree 1

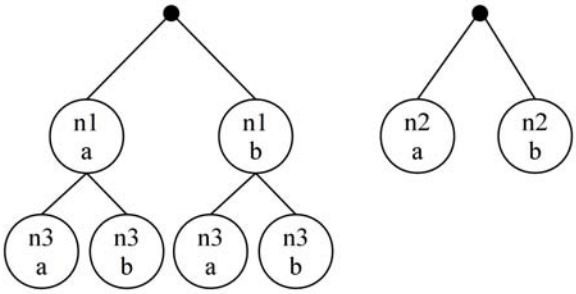
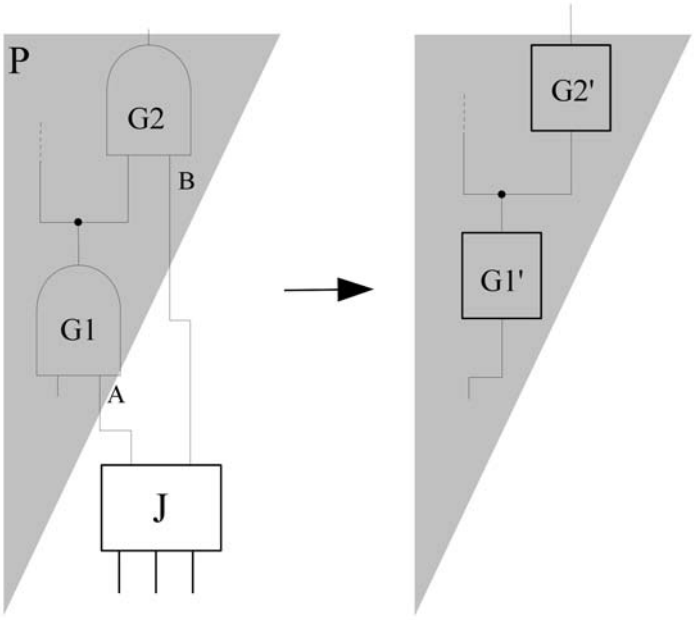


Figure 4.7: Justification tree 2

The pre-justification of a block may allow simplifying the path region before starting its analysis. An example of this case is depicted Fig 4.8. P represents the path region, and J a justification section. If J can be pre-justified demonstrating that both output nodes can be set to 1 simultaneously, then the gates G1 and G2 may always be sensitized. Thus, there is no reason to check its sensitization during the path analysis, and the gates can be considered single input gates as shown in Fig. 4.8b.



(a) *(b)*

Figure 4.8: Pre-sensitization

4.3. Encapsulation techniques

The encapsulation techniques are another approach to reduce the complexity of a combinational block. These techniques are based on packaging certain parts of the circuits into sub-circuits, taking such sub-circuits as single gates. Therefore, the problem of finding paths through the circuit reduces to finding the paths through each sub-circuit, and the paths through the global compacted circuit. This technique may reduce exponentially the number of paths to be computed, making the task of identify the paths through a combinational circuit possible.

4.3.1. Principle of the technique

In the worst circuit scenario, the order of magnitude of the number of paths through a combinational logic block grows exponentially with the circuit size (i.e. the number of nodes). Therefore, a reduction of the node number may drastically reduce the number of paths. In addition, to circuit node reduction, the technique developed identifies repeated circuit structures labeling their encapsulation under the same block.

With this objective, has been developed a set of algorithms that package specific circuit parts into sub-circuits. Each sub-circuit can be analyzed in a very short time, while contributing to an exponential reduction of the main circuit complexity providing an exponential reduction of the paths number. For instance, Fig. 4.9 represents a large circuit divided into four sub-circuits. The number of paths going from the input node *In-1* to the output node *Out-1*, can be extremely large. Assume that each sub-circuit (*SC-X*) has 100 paths, from one of its inputs to one of its outputs. This means that potentially there could be 2,000,000 paths from *In-1* to *Out-1* if the entire circuit is considered, as shows (4.4). However, if the circuit is partitioned into independent sub-circuits, and the path-finding algorithm is applied individually to each sub-circuit and the sub-circuits are used as if they were simple gates to compute the overall circuit paths, then the number of paths to be computed reduces exponentially, and the task becomes affordable.

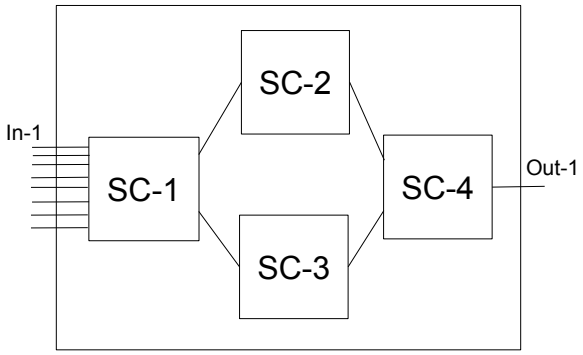


Figure 4.9: Subcircuit encapsulation

The following expressions show the number of paths to compute for the original circuit (4.4), and for the case of using sub-circuits (4.5). As shows (4.4), the total number of paths of the plain circuit from *In-1* to *Out-1* are the paths that pass trough {*SC-1*,*SC-2*,*SC-4*} plus the paths through {*SC-1*,*SC-3*,*SC-4*} giving a large number of paths. In the second case (4.5), each sub-circuit is processed independently, and then is processed the main circuit. In this case, the number of paths that must be calculated is reduced to the sum of the paths of each sub-circuit, plus the paths of the global circuit. In this very simple example, the number of paths that must be computed is reduced by several orders of magnitude. In a real circuit, the situation usually not as favorable, but as demonstrate the results presented later the reduction are very significant.

$$nPaths_{SC-1} \cdot (nPaths_{SC-2} + nPaths_{SC-3}) \cdot nPaths_{SC-4} = 2{,}000{,}000 \tag{4.4}$$

$$nPaths_{SC-1} + nPaths_{SC-2} + nPaths_{SC-3} + nPaths_{SC-4} + nPaths_{global} = 402 \tag{4.5}$$

In addition, if the original circuit has a number of repeated structures, then some of the identified sub-circuits have identical structure, and the corresponding sub-circuit will be only processed once. This helps in a further reduction of the circuit structure to be processed, and identifies repeated structures that can be optimized independently of the main circuit, even at the full-custom level. Depending on how many times each structure is repeated within the circuit, the effort of a full-custom design of the sub-circuit may lead to a large improvement of the main circuit area, power and delay.

The following sections describe in detail each circuit simplification technique adopted to reduce the circuit complexity, without losing information required to compute the delays and other

strategical parameters for design verification and optimization [46]. These techniques then can be applied iteratively, since the circuit resulting from applying of one of these techniques may has an appropriate structure to apply a technique that could not be applied before the first simplification.

4.3.2. Specific techniques

Following will be explained each encapsulation technique that identifies specific structures inside a circuit, encapsulating these structures inside a sub-circuit, reducing the number of nodes and logic gates of the circuit. Such techniques are independent and can be applied in any order and even iteratively.

4.3.2.1. Gates with special input configuration

In certain cases, automatic synthesis tools generate a structure with standard cells that have some of its inputs connected to the same node or to a node with steady logic value. These cases correspond to gates with special input configurations.

The first special case is composed by gates having various inputs connected to the same node, and are called redundant-input-gates. Fig. 4.10 shows an example of this case where node N3 is connected to two G1 gate inputs. This gate is converted into a new gate (MG1) without repeated inputs. There are cases where a NAND2 gate, or a NOR2 gate, is used to implement an inverter, by connecting both inputs to the same node. This is an example of case where this technique can be applied.

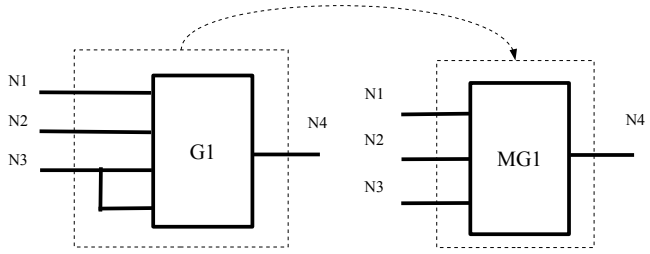


Figure 4.10: Redundant inputs gate

As examples, there are two cases that appears considerably in the synthesized circuit used as test bench. The example shown in the Fig. 4.11 corresponds to a multiplexer with the input D0 inverted, where both data inputs have been shorted, that is a very simple way to select between the direct or inverted value of a given signal. The function implemented by this configuration is equivalent to an

XNOR2, however, this is an efficient way to implement a XNOR2 function. Even, some technologies implement the XNOR2 gates by using this configuration internally.

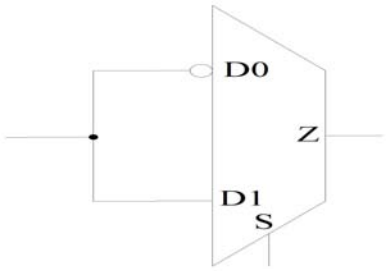


Figure 4.11: XNOR2 implemented with a multiplexer

This technique does not reduce the number of circuit nodes or gates. However, the affected node fan-out is reduced, and may become a non-stem-node, (i.e. a node with fan-out equal to 1), if the involved gate is the only one being connected to that node. If this occurs, this node is ready to be simplified using the non-stem-node simplification explained next.

In some cases, when a gate has inputs connected to the same node, the equivalent logic function can be implemented using a smaller gate. In his case, the design can be optimized by replacing the cell by another one that implements the equivalent logic function. In addition to the area reduction, the capacitance of the node connected to two inputs of the same gate is reduced.

This technique improves the algorithm to identify functional paths because the sensitization table for the new gate is simpler than the original due to the lower number of inputs. The fact that two or more original gate inputs have always the same logical value, reduces the overall number of input values combinations.

Another similar case occurs with gates having some input connected to a steady logic value (VDD or GND). In this case, the gate with a fixed input is converted to a gate with an equivalent logic operation, eliminating the fixed value node. Typically, this simplification has not a great impact on the circuit complexity, and eliminates the nodes with steady logic values simplifying the logic table.

Table 4.2 shows the number of redundant input gates for some benchmark circuits. As shows the table redundant input gates are not abundant, but this special case must be considered by algorithms. However encapsulating these gates allows to implement algorithms ignoring this special

configuration that results in simpler algorithms. Thus, the performance improvement occurs in each gate of the circuit and not only in the encapsulated gates. All algorithm components can be simplified: sensitization, implication, branch point process and justification.

Table 4.2: Redundant input gates

Circuit	# of redundant input gates
c7552	14
b17	16
b18	22
b19	65
b22	13

4.3.2.2. Non-Stem nodes

Non-stem-nodes, i.e. the nodes with a fan-out equal to 1, do not imply any decision during the process of finding paths since there are no branches to consider, nor states to be annotated for later analysis. Therefore, these nodes may be eliminated, reducing the number of nodes and therefore the amount of memory required to store the paths and the circuit state although they don't reduce the number of paths. This simplification also reduces the number of gates, because the two gates connected to that node are fused into one macro-gate. In this case the simplification impacts the tool memory requirements, the circuit structure, the nodes through which each path passes, and the tables that keep the logic value of each node of the circuit.

Fig. 4.12 shows an example of this situation, where node N5 is a non-stem node. Once after the simplification, node N5 disappears and gates G1 and G2 are merged into a macro-gate MG1.

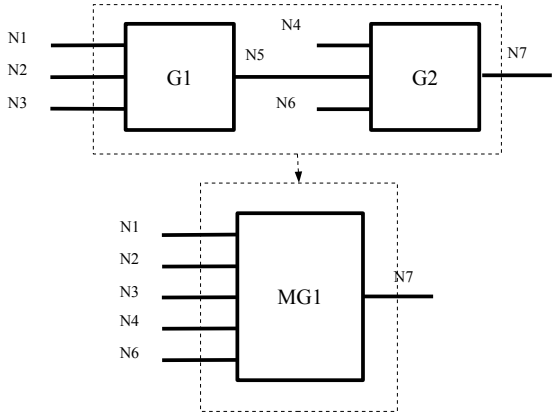


Figure 4.12: Non-stem nodes

The main advantage of this simplification is that, due to the new macro-gate, the two gates sensitization and justification is performed in one step. Also, the sensitization and justification tables for the macro-gate can be smaller than the two original tables, because some input combinations may become incompatible when the two gates are treated together as one single gate.

In some cases, when the involved gates share an input node, the fan-out of that node is reduced, increasing the impact of the simplification. For example, if nodes N3 and N4 in Fig. 4.12 are the same node, then the new gate MG1 would have only four inputs instead of five, and the bifurcation would be hidden inside the macro-gate. In this case, if the fan-out of the node was two, the new node would have a fan-out equal to one and could be simplified using the same technique in the next step.

The simplification ratio of this technique is: -1 node and -1 gate for each non-stem node, plus the additional effects if some input nodes are shared between the gates merged.

4.3.2.3. Reconvergent fan-out

Reconvergent fan-out simplification is a significant technique used to reduce the circuit complexity given its impact on the path finding algorithms [47]. A reconvergent fan-out is the structure where all paths starting from a given node converge to a single node. Fig. 4.13 shows an example of reconvergent fan-out. The node N2 has a fan-out equal to 3, and all paths starting from that node pass through the node N10, this is a reconvergent fan-out. This simplification technique converts all gates and nodes from the start node to the node where all paths converge to a single macro-gate. The example of the Fig. 4.13 is quite simple, however even in this simple example six gates and four nodes are converted to a single macro-gate. Additionally, the starting node N2 with fan-out equal to 3 becomes a non-stem-node, and can be simplified using the non-stem node technique.

In some cases it is possible that reconvergences between two nodes too far away to be found exists since the complexity of searching for reconvergences increases exponentially with the length of the paths from start node to reconvergence node. This situation does not represent a limitation, since the hierarchical nature of the simplification algorithms, allows identifying this situation in subsequent passages of the algorithm once after other simplifications that reduce the circuit complexity have been applied.

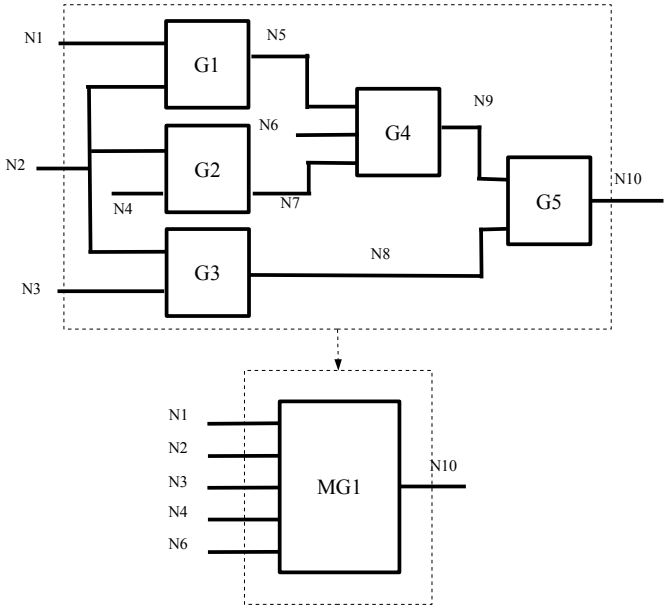


Figure 4.13: Reconvergence

This technique provides a large reduction on the circuit complexity, reduces the number of nodes, the number of gates, and the fan-out of the input nodes of the macro-gate.

A convergent block has only one output node regardless of the number of gates that form it. Therefore all paths through the block share its final gate. The stepwise sensitization algorithm is unaware of this fact having great impact on the algorithm performance, since if the shared gate is sensitized in advance then some work could be avoided. I.e, many paths through the reconvergence may be false paths due to the shared gate, and the stepwise algorithm must trace all paths to discover it.

Some existing algorithms are able to discover this fact and act in consequence, however this requires algorithms specially designed to consider this cases and therefore, this increases in the algorithm complexity. Otherwise if the reconvergent structure is encapsulated inside a block and treated as if it was a single gate, its complexity is hidden inside the block and the algorithms do not require to consider these kind of special cases, allowing simpler algorithms.

This structure behavior is discussed in detail in the SET propagation section (6) given its inherent interest.

4.3.2.4. Single input gates

The gates that have only one input, i.e. buffers and inverters, do not imply any decision during the sensitization and justification process. They do not require sensitization, and the justification of an output value is direct without any decision. From the logical point of view, the buffers could be eliminated directly, however given its impact on the path delay they cannot be eliminated without loss of information, although can be merged with other gates.

Since non-stem nodes can be simplified with a technique previously explained, this technique considers that the input and output nodes of the single input gate have a fan-out larger than 1.

Fig. 4.14 shows this simplification method explained in two steps. In the first step, the single input gate is cloned at each output stem providing the output nodes of the created virtual gates with a fan-out equal to 1. In the second step, these nodes are simplified by applying the non-stem node simplification technique that merges the single input gates with the following gate.

By applying this technique, the number of nodes and gates of the circuit reduces by one for each single input gate. However, as shown in Fig. 4.14 the node N2 fan-out increases because this node was merged with the node N5 due to the withdrawal of the gate G2. Concentrating the fan-out at one node improves the performance of some algorithms. Since, this reduces the number of times that the algorithm must save the state, and increases the number of times that this saved state will be reused, one time for each gate that follows the stem node. For this reason, the tables that keep the current state of the algorithm can be optimized before being stored, and due to the increased reutilization of the stored states, the computing time required for the optimization of the tables is productive. On the contrary, when the fan-out is small the time required to optimize the tables before storing is counterproductive.

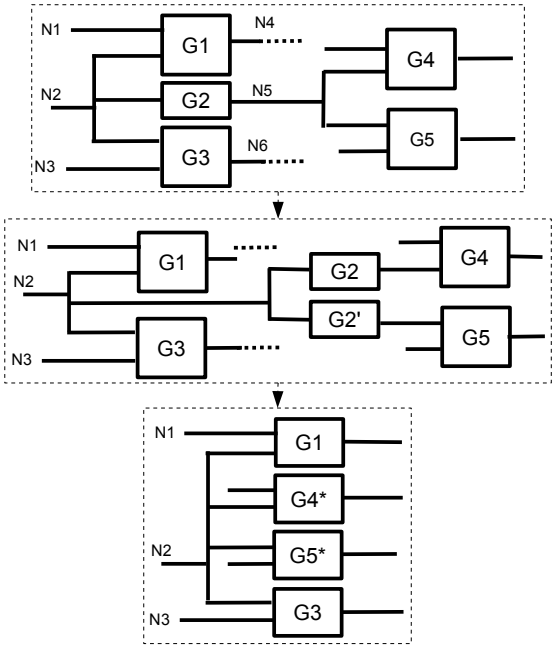


Figure 4.14: Single input gate

4.3.2.5. Gates sharing all inputs nodes

This encapsulation technique is less general than the previous ones, in the sense that only some circuit designs have the structures required by this simplification methods, being very circuit dependent. This technique is focused to the case of various logic gates that share all their input nodes, and in turn these nodes do not feed other gates. To illustrate this type of structure Fig. 4.15 shows an example where two logic gates (G2, G3) share the input nodes (N5 and N6) feeding their both inputs. Depending on the circuit structure it's also possible to find out gates that share more than two input nodes, or more than two gates sharing their input nodes. As shows the figure, this method encapsulates into a block the gates sharing their input nodes, plus the gates driving the shared nodes since this technique requires that the shared nodes do not feed other gates.

The number of gates that share all their input nodes and the number of input nodes shared, determines the impact of this simplification technique over the circuit complexity. In general, if there are n_G gates sharing n_N nodes, then the total number of gates wrapped is $n_G + n_N$, and number of nodes wrapped are n_N. Once applied this method, the circuit reduces the number of nodes by n_N and gates by $n_G + n_N - 1$.

The impact of this method over the circuit complexity, can be greater if some of input nodes of the wrapped gates are the same. In the example shown in figure, the block has four input nodes, however if the inputs N1 and N4 were the same node, then the resulting macro-gate would have

only three inputs, and the fan-out branch of this node was hidden into the macro-gate.

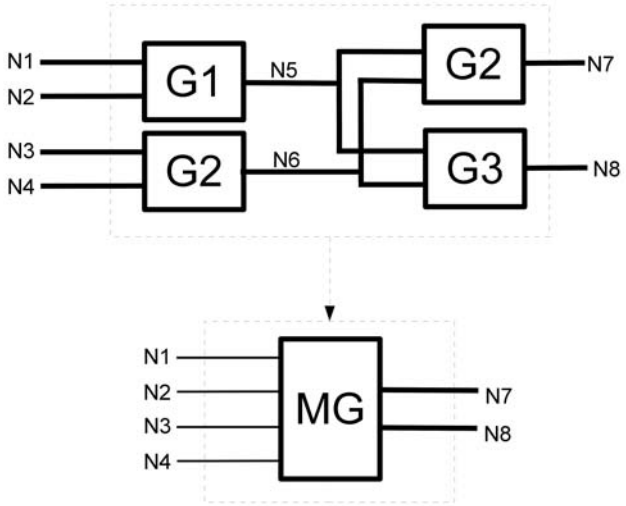


Figure 4.15: Shared input gates

4.3.2.6. Gates partially sharing input nodes

This is a less restrictive version of the previous technique. In this case the technique is more effective if applied to nodes with a large fan-out. Fig. 4.16 shows a simple example of this case, since the cases observed in real circuits are considerably more complex. In the example of Fig. 4.16, we can see that the node N7 have a fan-out equal to 4, and all the fan-out gates of the node N8 are also fan-out gates of N7. This allows to wrap the gates G4-G7 into a block, together with their input nodes and the gates that drives the nodes that all their fan-out gates are inside the block, i.e. G1 and G2 in the example.

As already been said, the real cases are more complex. In real circuits, after apply some of the other techniques, there are nodes with large fan-out, and that nodes shared their fan-out gates with a high number of nodes, resulting blocks with a large number of gates, with a great impact over the global circuit complexity.

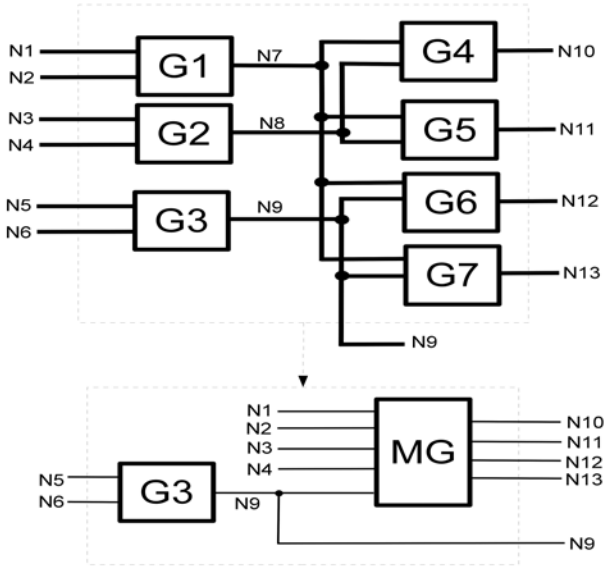


Figure 4.16: Partially shared input nodes

4.4. Application

The set of techniques presented in the previous sections are intended to improve circuit analysis by partitioning the circuit in different sections and/or encapsulating some circuit parts into subcircuits reducing its overall complexity. The specific techniques to apply and the sequence in which must be applied depend on the circuit structure and the nature of the analysis to perform.

Circuit partitioning techniques are mainly useful to focus the analysis to specific regions of a design avoiding to manage the entire circuit when the interest is in a specific circuit region. As a practical example, to perform the SET propagation analysis presented in chapter 6 was used the technique of partitioning by output since to compute the SET sensitivity we are interested on the paths ending at a given output node.

On the other hand, the encapsulation techniques allows to analyze each circuit block individually, and then the results of this analysis are used to simplify the entire circuit analysis. For very large size and complex circuits, the encapsulation techniques may yield a hierarchical structure of blocks, i.e. a block may be internally constituted by other blocks and so on. These techniques also takes advantage of repeated structures in some circuits. These cases are identified by the encapsulation techniques, and the repeated structures are analyzed only once, reducing the computation resources required to process the entire design.

Since the main objective is to reduce the circuit complexity in general, theses techniques can be applied to many kinds of design analysis like synthesis optimization, timing analysis, test pattern generation, SER analysis and SET propagation, and in general any other analysis that can benefit from the circuit complexity reduction.

4.4.1. Identification of repeated structures

When the encapsulation techniques are applied to a given circuit, some of the subcircuits created have an identical internal structure, thus they are instances of a same block [48][49][50]. One of the most relevant applications for the ability to identify repeated structures is circuit synthesis optimization. I.e. if a block is repeated many times in a circuit, this block may be optimized individually, even using full custom design of the layout. The full custom design of a block that repeats many times within the design, may improve considerably area, power consumption and delay of the whole circuit [51].

The experimental tests show that, the most complex circuits usually have structures that repeat

many times. A good example is the ISCAS circuit c6288, a 16-bit multiplier representing a difficult architecture for the path tracing and timing analysis algorithms [43]. However, this circuit has a very regular structure with a high number of repeated substructures, as will be shown in the results.

To illustrate the identification of repeated blocks Table 4.3 shows the number of instances of the six block with higher repetition for a selection of circuits after apply the non-stem-nodes simplification. Same block number in different circuits do not necessarily refer to the same subcircuit.

Table 4.3: Block repetition after non-stem simplification

Circuit	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**
c5315	39	6	4			
c6288	95	7	7	6	3	
c7552	11	5	4	4	4	4
b14	58	48	33	31	24	15
b15	129	33	30	29	18	15
b17	400	130	104	80	64	59
b18	337	314	294	180	120	118
b20	55	26	24	23	21	18
b21	46	24	22	19	19	18
b22	66	63	49	33	30	28

As show the results in the table some circuits have a large number of instances of the same block. Circuits b17 and b18 show a quantitative higher number of block repetition with 400 and 337 times respectively.

As example Table 4.4 gives the internal components, i.e. the standard cells, that form some of the blocks with higher repetition from the results of Table 4.4. The results show that there is a block consisting of two standard cells (AOI22 and OAI211) with higher repetition in various circuits, and was proved that these blocks are identical in all cases. The reason for the huge repetition of this configuration falls outside of the topic of this thesis, however could be a consequence of the strategies followed by the synthesis algorithms.

Table 4.4: Block repetition examples

Circuit	# Instances	Block components
b17	400	AOI22 + OAI211
	130	AOI12 + OAI21
b18	337	AOI22 + OAI211
	314	AOI22 + MUXI21
b15	129	AOI22 + OAI211
c6288	95	FA1 + FA1

Due to the high number of instances, if this structure were optimized by a full-custom design creating a new cell equivalent to this block the impact over the whole circuit could be considerable.

The previous results about the block repetition were obtained after one simplification step, however the block repetition also occurs at higher levels. Therefore, applying iteratively the non-stem simplification technique shows that there are larger blocks repeated. As an example Table 4.5 shows the number of instances of the eight blocks exhibiting higher repetition after each simplification step applied to the circuit b17. The results in the table shows that the number of instances reduces at each step but the blocks generated are larger at each step. Concretely the block 2A consists of one instance of the block 1A plus one standard cell. Many of the blocks obtained in the four step consist of two blocks created at previous steps.

Table 4.5: #Instances for iterative simplifications of b17

	Step 1	Step 2	Step 3	Step 4
# Instances	400 (1A)	325 (2A)	51	29
	130	68	40	22
	104	61	36	12
	80	56	20	11
	64	48	6	6
	59	46	5	6
	47	21	4	5
	40	21	4	4

4.4.2. Path enumeration

Many design analysis techniques, mainly algorithms to identify functional paths for timing analysis or test pattern generation, require enumerating structural paths, or constructing a graph that represents the topological paths through the circuit [41]. The framework provide algorithms to perform these tasks, however both methods are unaffordable for many real circuits, due to the exponential growth in the number of paths of a combinational block. However, as shown in Section 4.1, when some circuit parts are encapsulated the number of paths through the circuit (or the number of nodes of a graph) is also reduced exponentially, and becomes affordable.

To illustrate how the encapsulation techniques can help to analyze a circuit Fig. 4.17 shows an a portion of example circuit. The region inside the box is a reconvergent structure since all paths starting from N1 pass through N4.

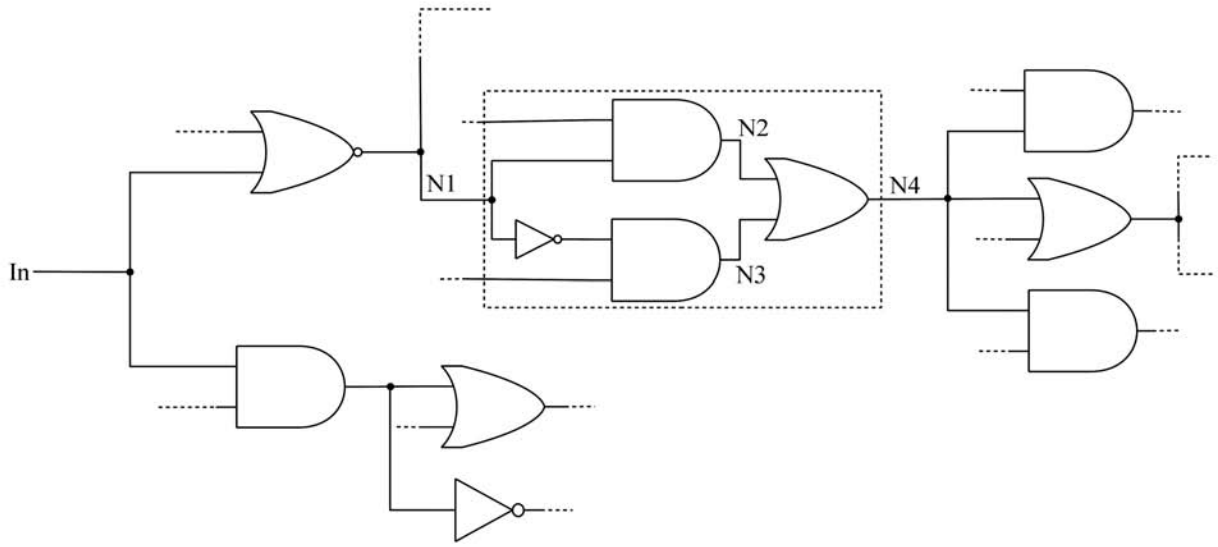


Figure 4.17: Example circuit

The path graph for this circuit from the node *In* generated as was explained in section is shown in figure 4.18. The region delimited by the box corresponds to the reconvergent structure highlighted in the circuit schematic of Fig. 4.17. As shows the graph there are two structural paths from node N1 to N4, one through node N2 and the other through node N3.

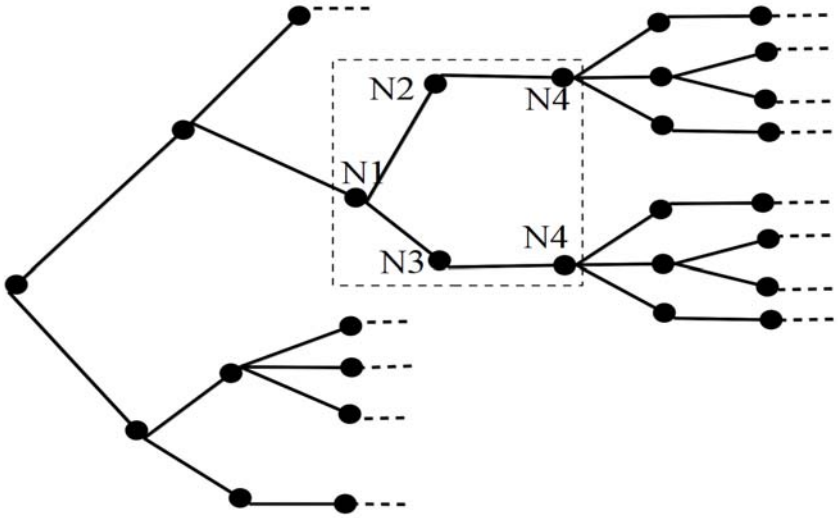


Figure 4.18: Path graph before simplification

If the circuit of the example is simplified by using the technique to encapsulate reconvergent structures then both paths through the reconvergence are hidden inside the block, with the consequent simplification to path graph. Fig. 4.19 shows the path graph for the simplified circuit, as shown the figure there is only one path from N1 to N4 simplifying all branches hanging from node N4.

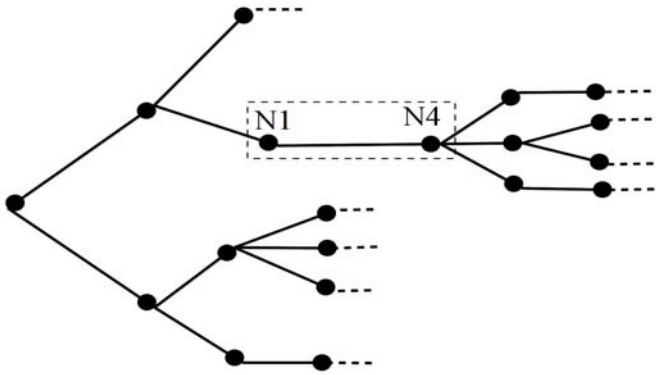


Figure 4.19: Simplified path graph

In this example there is a single block with a simple internal structure, however in real applications the blocks involved may be large subcircuits, even with a hierarchical structure with other blocks inside it. In this cases number of paths through the block may be large, and some of them could be identified as false analyzing the subcircuit isolatedly, and the path graph can be pruned.

4.5. Results

The techniques exposed were applied to the larger ISCAS'85 and ITC'99 benchmark. We first analyzed the circuit complexity, and its evolution when applying iteratively the encapsulation techniques developed. Since a well-defined circuit complexity metric is not available, we took the number of wires and gates, and the number of fan-out branches to quantify a measure of the circuit size and complexity. This represents a valid metric to evaluate the impact of the encapsulation techniques because the number of input and output nodes remains constant; only the circuit internal wires changes when applying the encapsulation methods. However the number of input and output nodes may vary in the case of apply some of the partitioning techniques.

To give an insight of how the encapsulation techniques allow to create blocks sharing their internal structure is defined the reuse factor representing an average of the number of times that each block is repeated in the circuit, weighted by the size of the block, expressed as the number of standard cells that the block wraps.

Tables 4.6-4.10 show the simplification results for the two largest ISCAS'85 and three ITC'99 circuits. Each column represents a simplification step. The first step applies the simplification of redundant inputs gates, the impact of this technique is very small as shown by the results. The second step is the reconvergence identification. Then, the non-stem nodes simplification was applied, being the step providing a larger impact on the overall number of gates. Finally the single-input gates simplification step is applied, performed in two stages. The main values used to represent the circuit size and complexity are: the number of wires, i.e. internal nodes, and gates. The last row shows the reuse factor, representing the repetition of structures in the circuit. As shown, the single-input gates expansion has a negative impact of the circuit size, but represents an intermediate step. Results show that the reuse factor has an inverted tendency with respect to the circuit complexity, specially for the c6288 circuit due to its very regular structure with a large number of repeated blocks.

Table 4.6: ISCAS c6288

	Original circuit	Redundant input gates	Reconvergent structures	Non-stem nodes	Expand single in gates	Non-stem nodes
#Inputs	32					
#Outputs	32					
#Wires	775	775	763	13	23	8
#Gates	602	602	509	17	27	12
#Stem nodes	150	150	138	21	16	16
#Branches	683	682	658	38	38	34
Reuse factor	1.00	1.00	1.02	35.41	22.67	51.00

Table 4.7: ISCAS c7552

	Original circuit	Redundant input gates	Reconvergent structures	Non-stem nodes	Expand single in gates	Non-stem nodes
#Inputs	167					
#Outputs	54					
#Wires	576	576	518	293	459	215
#Gates	630	630	572	347	513	269
#Stem nodes	424	423	379	365	288	288
#Branches	862	847	784	696	696	602
Reuse factor	1.00	1.00	1.10	1.82	1.55	2.96

Table 4.8: ITC'99 b17

	Original circuit	Redundant input gates	Reconvergent structures	Non-stem nodes	Expand single in gates	Non-stem nodes
#Inputs	1354					
#Outputs	1485					
#Wires	9213	9213	9013	4465	9978	3290
#Gates	10647	10647	10447	5919	11432	4744
#Stem nodes	5881	5877	5777	5758	4610	4610
#Branches	20628	20605	20419	19837	19837	19327
Reuse factor	1.00	1.00	1.02	1.80	1.41	3.41

Table 4.9: ITC'99 b20

	Original circuit	Redundant input gates	Reconvergent structures	Non-stem nodes	Expand single in gates	Non-stem nodes
#Inputs	522					
#Outputs	513					
#Wires	3412	3412	3317	1917	3845	1431
#Gates	3869	3869	3774	2392	4320	1906
#Stem nodes	2476	2476	2427	2390	1954	1954
#Branches	6520	6514	6425	6052	6052	5936
Reuse factor	1.00	1.00	1.03	1.62	1.34	3.04

Table 4.10: ITC'99 b21

	Original circuit	Redundant input gates	Reconvergent structures	Non-stem nodes	Expand single in gates	Non-stem nodes
#Inputs	521					
#Outputs	512					
#Wires	3519	3519	3423	1912	3822	1424
#Gates	3936	3936	3840	2365	4275	1877
#Stem nodes	2480	2479	2432	2392	1946	1946
#Branches	6553	6546	6454	5996	5996	5847
Reuse factor	1.00	1.00	1.03	1.66	1.37	3.12

Tables 4.6-4.10 show the evolution in the circuit size, although the main interest is on its impact over the circuit analysis techniques efficiency.

We illustrate this effect for two tasks: a path graph construction and structural path enumeration. The two analysis were performed using first the original circuit without any simplification, and then using the simplified circuit.

Table 4.11 presents the results for path graph creation, providing the memory used and CPU time required for the simplification step and for the graph creation. As can be seen in some cases the CPU time for simplification plus the processing is larger than processing the original circuit, but

must be taken into account that the simplification step must only be performed once. As shown, some circuits cannot be processed in its original configuration due to an excessive consumption of memory and CPU. These cases are marked with the tag "killed". The circuit c6288 provides the most impressive results due to its highly regular structure.

Table 4.11: Path graph creation

Circuit		**Memory (MiB)**	**CPU Time (s)**	
			Simplification	**Processing**
c6288	**O**	2742	-----	Killed
	S	12	0	3
b14	**O**	2405	-----	11
	S	464	3	8
b15	**O**	456	-----	3
	S	221	7	8
b17	**O**	1708	-----	6
	S	756	24	2
b20	**O**	2475	-----	Killed
	S	1312	10	17
b21	**O**	2631	-----	Killed
	S	1201	11	17

The enumeration paths results are given in Table 4.12, including the memory consumed, the CPU time required and the number of paths identified, taken into account that in the simplified version the number of paths is less than in the original case due to the hierarchical nature. The second column shows the portion of the circuit analyzed until the process was killed due to memory depletion. As occurs in the previous results, the circuit c6288 presents the largest difference for both cases tested.

Table 4.12: Structural paths enumeration

Circuit		Killed at input	Memory (MiB)	CPU Time (s)	Paths
c6288	**O**	15 / 32	2182	94	26,359,541
	S	Complete	12	4	2,026
b14	**O**	Complete	1634	71	9,381,264
	S	Complete	645	22	4,283,940
b15	**O**	Complete	289	21	2,365,298
	S	Complete	206	20	2,140,734
b17	**O**	10925 / 1357	2003	3641	18,499,297
	S	Complete	1357	1051	11,923,450
b20	**O**	497 / 523	2226	157	12,453,676
	S	Complete	1829	101	11,626,730
b21	**O**	389 / 523	2113	86	12,395,361
	S	Complete	1574	93	10,922,976

4.6. Framework

The entire EDA framework has been developed keeping special attention to modularity and customizability of each component, with the objective of providing a quite high flexible tool easily adaptable to a vast set of applications. This framework is an evolution of a tool presented in [52]. The global framework structure is shown in Fig. 4.20, where each module is listed together with the relationships between the modules, as well as the main fluxes of data for both input and output.

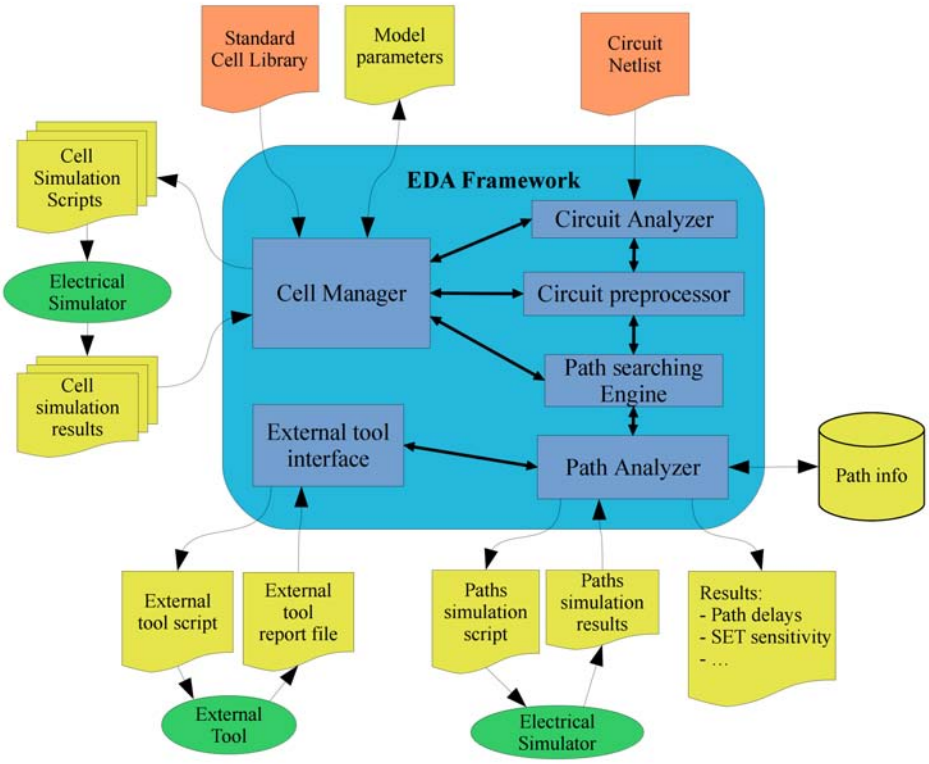


Figure 4.20: General Framework Structure

The high framework flexibility is of special interest in the research field, where the development of new design strategies or modeling techniques require a way to test them and verify its effectiveness against existing tools or models.

The framework comprises the following modules:

- Cell manager
- Circuit analyzer
- Circuit preprocessor

- Path searching engine
- Path analyzer
- External tool interface

4.6.1. Cell manager module

The cell manager is responsible for managing all the information related to the gates/cells used in the circuits, and for performing task related to them. The module objective is to import a standard cell library from a file or a database and create data structures representing the cells to provide the information required by the circuit analyzer to link the circuit with the cell library. Besides, this module is also the responsible of creating the logic tables required by sensitization algorithms, and the models used to estimate the circuit operation.

To perform its work the module leans on two submodules:

- The logic engine provides all the tools required to perform logic operations, manipulate boolean expressions and generate logic tables required by the algorithms.
- The model engine provides a set of mathematical algorithms intended to fit data to analytical expressions. These algorithms are used to extract model parameters from simulation results.

Fig. 4.21 provides the cell manager detailed structure, including each submodule. Configuration parameters are in green boxes, while input and output data are in yellow.

The logic engine creates the logic functions for each cell from the specifications provided by the standard cell library file. Generally the cell library files provides the logic function of each cell in a textual form, for example for an AND2 cell the library file gives the logic function as Z=A*B. Therefore, the logic engine provides a Boolean algebra processor capable of interpreting a Boolean expression, simplifying if necessary, and generating a function to implement such Boolean expression. Each required logic function is customized at compilation-time, i.e. the logic engine creates and compiles the code to implement the function. Therefore, since the logic functions code is compiled instead of using a runtime configuration, the execution of the algorithms involving the cell logic functions is faster.

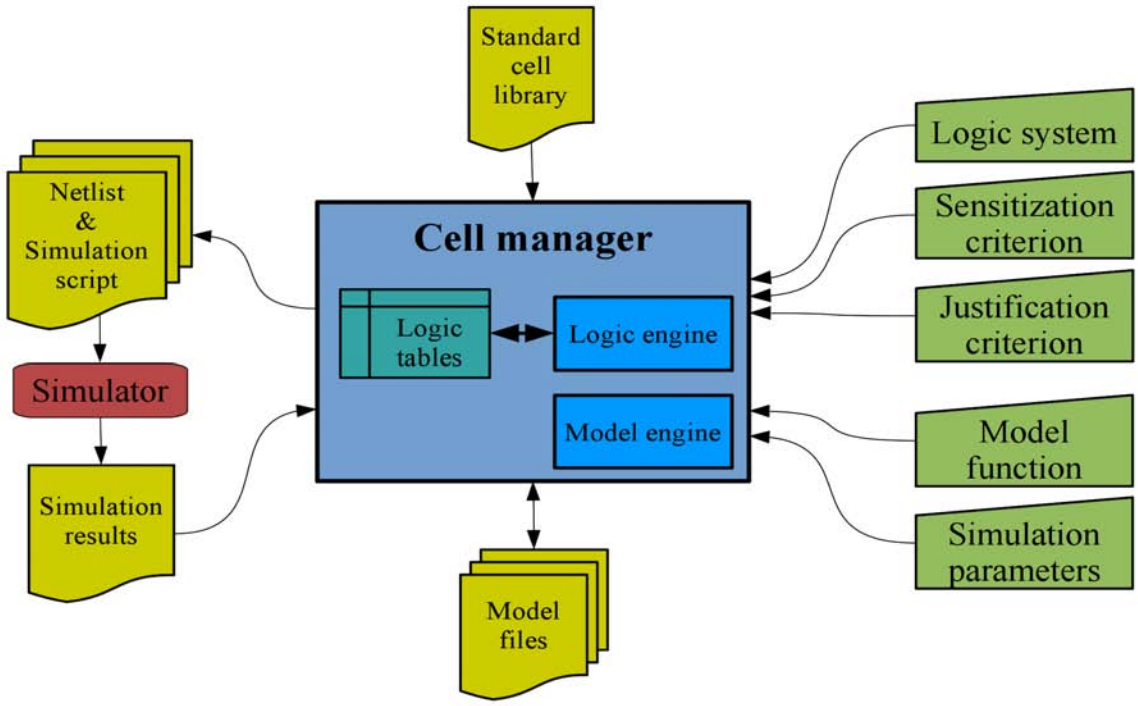


Figure 4.21: Cell manager diagram

The logic engine Boolean algebra processor also computes the equivalent logic function for the blocks created by the encapsulation techniques, since these blocks are treated as cells.

Besides the generation of the logic functions, the logic engine creates the logic tables required by the sensitization and justification processes. The standard cells performing the same logic function, (i.e. same gate type with variations in gate strength), share the logic tables avoiding redundant information.

The logic engine can be customized to accommodate the logic system and the criteria used to generate the sensitization and justification tables. Predefined configurations exist for each sensitization criteria detailed in section 3.3.1, however the system allows configuring other criteria.

The cell manager module also provides a set of tools intended to fit analytical models to simulation results. To accomplish this goal the module is capable of generating transistor-level netlists to simulate each cell of a library using a circuit structure according to the model requirements. These netlists are created using a netlist template provided by the user specifying a generic circuit structure independent of the standard cell to be simulated, allowing an automatic netlists generation independently of the type of model being developed. Moreover, together with the netlists, the cell manager creates a set of scripts to perform electrical simulations of such netlists.

The parameters to be varied during simulations, like temperature or supply voltage, as well as the interval of values to use are also configurable parameters.

Thereafter the module loads the result files generated by the electrical simulator, and extracts the model parameters for each standard cell, assisted by the engine model and using the model function provided as a parameter.

4.6.2. Circuit analyzer

The circuit analyzer reads the circuit netlist, and using the information provided by the cell manager module creates a data structure that represents the circuit. Such structure is then used by other framework algorithms. This module is capable of importing a circuit netlist in the most common formats, mainly Verilog (IEEE 1364) [53], however has been developed in a way that allows easily incorporate interpreters for different netlist formats.

4.6.3. Circuit preprocessor

This module can be used to preprocess the circuit structure before perform an analysis, with the objective of focusing the analysis at a specific circuit region, or to simplify the circuit structure, allowing to easily process large combinational blocks. The set of tools provided by this module includes all the functions to perform partitioning and encapsulations, as detailed in section 4.1.

The partitioning techniques can be combined to select the circuit region of interest, and generate a circuit structure for the selected region. Furthermore, the encapsulation techniques can be applied iteratively in any order with only some exceptions, and generate a hierarchical structure. Thus, the techniques to be applied depend on the circuit size and structure, and can be selected at each step in depending on the previous steps result. This module is assisted by the cell manager module that is capable of creating the equivalent logic functions, and the logic tables for the blocks as if they are individual cells. This is possible except for large blocks, that require to be treated differently.

4.6.4. Path searching engine

This module includes the customizable algorithms detailed in section 3.4, together with the individual components used to personalize each step of the algorithms. Allowing to use a wide range of strategies to identify which structural paths are sensitizable paths (i.e. identify the true paths through a combinational block), provide the sensitization input vectors for each true path, etc.

This module is organized as if were a building kit, with individual components that can

assembled in almost any configurations. Therefore a set of processing components are combined into a generic algorithm to obtain the desired functionality. Once all components required by the algorithm chosen are established, the module allows to generate a compile-time customized version of the algorithm to achieve a better performance respect to use a runtime customization.

The results generated by the algorithms of this module are transferred to the path analyzer module, which is the responsible to manage the path information, providing different ways to store the path information for further analysis.

4.6.5. Path analyzer

This module provides a wide set of tools to work with the path information generated by the path searching engine or imported from an external source either generated by the tool itself or by third-party tools. The module includes data structures and functions to work with the information in multiple ways, allowing to operate over:

- Structural paths.
- Functional paths with the corresponding sensitization vectors.
- Subpaths, i.e. portions of paths.

The path information can be used to compute a wide range of results, in some cases assisted by cell manager that provides the model information. Some of the main results that can be computed from the path information are:

- Gate and path delays.
- Critical path identification.
- SET propagation and SET sensitivity analysis.
- Path activation probability.

However, since the framework has been implemented with aim of high flexibility, it allows to easily include any kind of model, and extend the path information processing, generating results adapted to each specific application.

4.6.6. External tool interpreter

These modules add the interoperability feature with third-party tools, allowing an automatic generation of scripts to be used to by external tools and import the results generated by the third-party tools. This feature allows incorporating additional processing steps in the workflow, and also can be used to verify the results provided by the tool, or the estimation obtained by a model. This is an interesting feature for the research, since allows verifying the behavior of a new model technique, or a new processing strategy, compared with a reference tool.

Chapter 5: Framework application to Timing Analysis

Chapter 2 introduced the basic concepts regarding timing analysis and its importance during the design flow. Chapter 2 also describes a general methodology for implementing analytical model considering multiple variables. This chapter explains how such a general modeling strategy is implemented to model the propagation delay through a logic gate, and how the model parameters can be extracted from electrical-level simulations. Accurate delay modeling requires precise node capacitance estimation, therefore this chapter presents a method to extract an effective capacitances.

Then it is shown how the delay model is used in combination with the path identification algorithms detailed in Chapter 3 is applied to perform timing analysis on benchmark circuits.

It is demonstrated the importance of considering the specific logic vector applied to sensitize each path for efficient delay estimation.

5.1. Polynomial Delay Model

The first version of delay model presented, is a deterministic version that does not considers the parameter variations caused by the manufacturing process, assuming that each standard cell device has its nominal values [54][55]. This model consists of two functions (5.1), one for the propagation delay (t_d) and the other for the output transition time (t_{out}), as have been defined previously. For a CMOS cell both functions depend on multiple variables [56]. The variables taken into account for this delay model are:

- Output capacitive load
- Input transition time
- Temperature
- Supply voltage

$$\begin{cases} t_d = f\left(C_{load}, t_{in}, T, V_{DD}\right) = \sum_i \sum_j \sum_k \sum_l P_{ijkl} C_{load}^i \cdot t_{in}^j \cdot T^k \cdot V_{DD}^l \\ t_{out} = f\left(C_{load}, t_{in}, T, V_{DD}\right) = \sum_i \sum_j \sum_k \sum_l Q_{ijkl} C_{load}^i \cdot t_{in}^j \cdot T^k \cdot V_{DD}^l \end{cases} \quad (5.1)$$

Each variable considered by the model has a different impact over the delay. Figs. 5.1 and 5.2 show the dependency of the propagation delay and the output transition time respectively, around a reference working point for an inverter of 65nm CMOS commercial technology. The reference working point was chosen with the following values:

- C_{Load} = 2.4 fF
- t_{in} = 40 ps
- T = 25 ºC
- V_{DD} = 1.2 V (Nominal supply voltage of the technology)

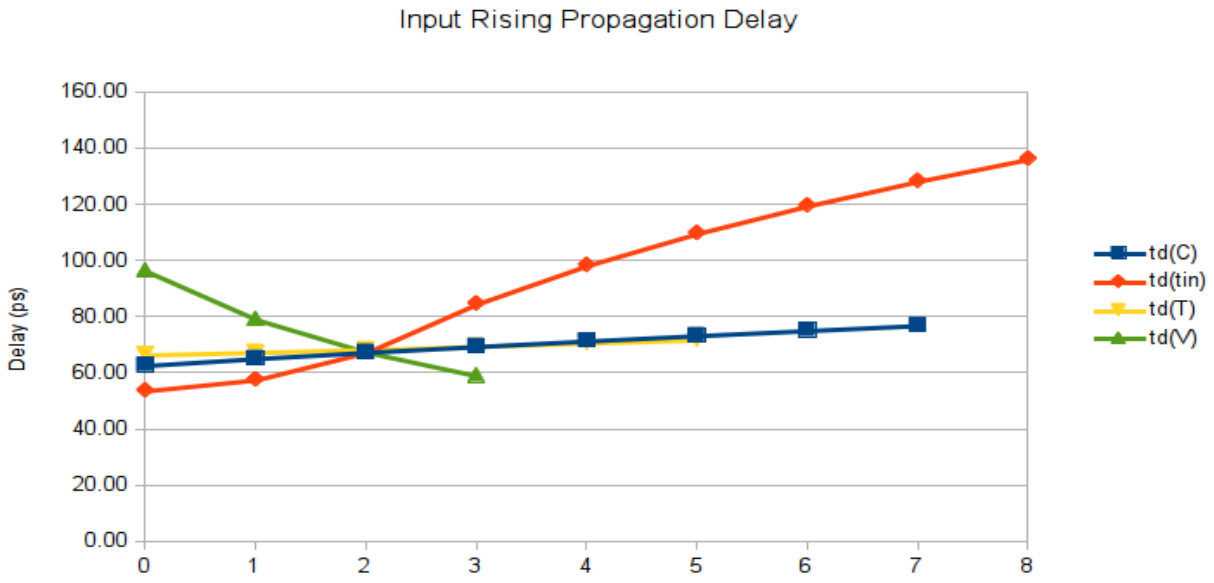


Figure 5.1: Rising input propagation delay variation

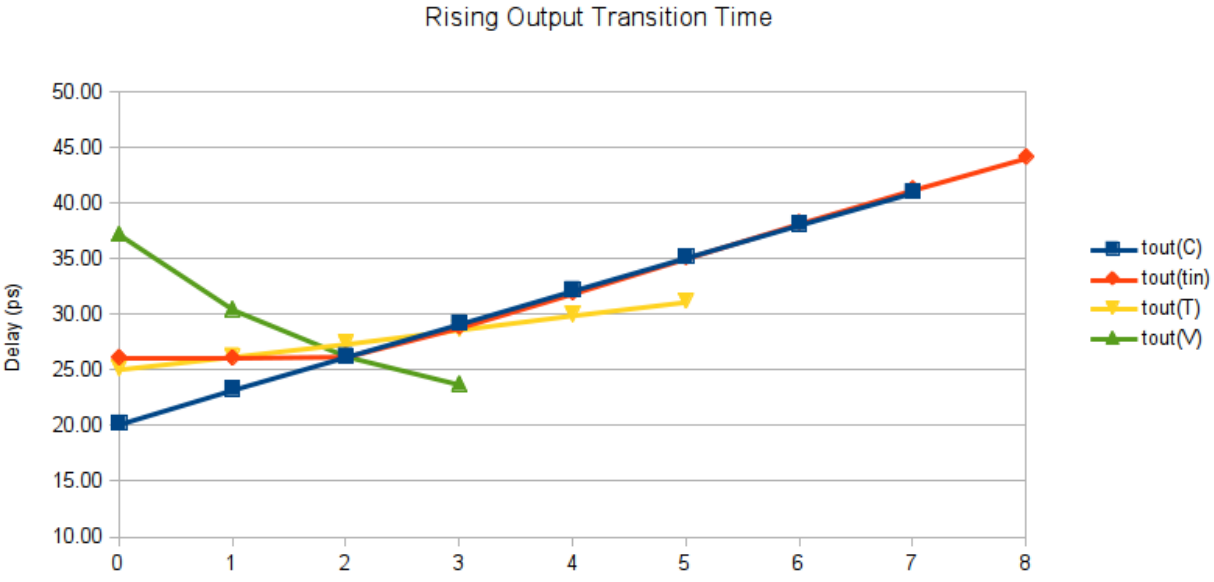


Figure 5.2: Rising output transition time

Equations (5.2) gives the value of variables in function of the x-axis value.

$$\begin{aligned} C &= 0.8\text{fF} \cdot (X+1) \\ t_{in} &= 80\text{ps} \cdot X + 20\text{ps} \\ T &= 25\,^{o}C \cdot X \\ V_{DD} &= 0.1\text{V} \cdot X + 1\text{V} \end{aligned} \tag{5.2}$$

It is shown that the input transition time is the parameter with a higher impact on the propagation

delay. The temperature, that varies from 0 ºC to 125 ºC, has a small impact compared to the other parameters. As expected, the supply voltage is the only parameter with an opposite tendency.

According to the specific application of the model, some of the variables can take a constant value, like a timing analysis where temperature variations are not considered. In view of the gate delays characteristics, the multivariable polynomial function presented previously fits perfectly to model the propagation delay and the output transition time. Thus, both functions use a 4 variable polynomial model, as shown in (5.1).

On a CMOS standard cell, the propagation delay from an input to an output varies depending on the input-output pair used, and also depends on whether the transition is a rising edge or a falling edge, resulting in a set of 4 coefficient matrices for each input-output pair, as outlined below.

- t_d^r : Propagation delay for an input rising transition.
- t_d^f : Propagation delay for an input falling transition.
- t_{out}^r : Output transition time for a rising transition.
- t_{out}^f : Output transition time for a falling transition.

For the standard cells that implement a primitive logic function, there is only one input vector that sensitizes each input to propagate a transition to the output. However, in the case of complex cells, previously detailed in the logic sensitization section, where in general each input can be sensitized by more than one input vector, both propagation delay and output transition time depend on the sensitization vector applied to the gate. This fact must be taken into account since knowing the input vector used to sensitize the gate is required for accurate delay estimation. Therefore, the complex gates require a set of 4 coefficient matrices for each sensitization vector of each input-output pair. Summarizing, the number of coefficient matrices for a standard cell are:

- Primitive cells: $4 \cdot n_{\text{in}} \cdot n_{\text{out}}$
- Complex cells: $4 \cdot \sum_{\text{in}=1}^{n_{\text{in}}} \sum_{\text{out}=1}^{n_{\text{out}}} SensVec(\text{in}, out)$

Where *SensVec(i, j)* is the number of input vectors that sensitizes the path from the input *i* to the

output *j*, and n_{in} and n_{out} are the numbers of inputs and outputs of the cell.

In this work we only consider the cases with steady values in all inputs except the input being propagated toward the output. Modeling multiple transitions at the gate inputs requires a more complex analysis not considered in this work. If the delay between the transitions is large enough, the propagation delay of the gate may be computed considering exclusively the first dominant transition, since in this case the other inputs can be treated as having a constant value. On the other hand, if the transitions are almost simultaneously the propagation delay is affected, and depends on multiple factors that are very complex to model. Theses factors are: the difference of arrival time between the dominant transition and the others transitions, the slope and the direction of the secondary transitions.

In the next sections the impact of the sensitization vector on the propagation delay of the cells will be analyzed at different levels of abstraction, to show the root causes of this variation and the importance of considering it for an accurate timing analysis [52][57].

5.1.1. Multidimensional extraction process

In a real application of the type of model presented before, should seek a compromise between accuracy and the model coefficients matrices size. The number of coefficients, not only affects to the memory requirement to store the model, but also to the computation complexity to evaluate the model during circuit design analysis. Since any model is only an approximation a given inaccuracy always exist. Therefore, the process developed to extract the coefficients for the multi-variable and multipurpose polynomial model is conditioned by a set of parameters that controls the regression, and ensure the best balance between accuracy and complexity.

The parameters that influence the extraction process are:

- Maximum polynomial order: Determines the maximum number of coefficients to be used for each polynomial regression.
- Minimum quadratic correlation coefficient: It is the minimum quadratic correlation coefficient. It compares two data arrays, and provides insight about equality between the two data sets taking a value in the range [-1,1]. A quadratic correlation coefficient of 1 indicates that the two data sets match perfectly. This parameter is used to control model accuracy, but depending on the specific type of function, a quadratic correlation coefficient

near to 1 may require a large polynomial order. To control this, the maximum polynomial order imposed, dominates over the quadratic correlation. Equation (5.3) is the expression used to compute the quadratic correlation coefficient.

$$r^2 = \frac{\sum_{i=0}^{n} (y_i - \overline{y}) \cdot (y'_i - \overline{y'})}{\sum_{i=0}^{n} (y_i - \overline{y})^2 \cdot \sum_{i=0}^{n} (y'_i - \overline{y'})^2} \tag{5.3}$$

where y is the original data being a function of variable x, and y' is the estimation made by the model, i.e., $y' = f(x)$ where f is the model function. To have an ideally adjustment the result must be $y' = y$.

- Maximum relative error: This parameter fixes the maximum error accepted between the original data and the data generated by the model. This maximum error cannot be an absolute value, because it must have the same significance independently of the order of magnitude of the data. The relative error is computed using (5.4).

$$Max.\ Relative\ Error = \underset{0 \le i \le n}{Max} \left(\left| \frac{y'_i - y_i}{y_i} \right| \right) \tag{5.4}$$

- Minimum relative range: As shown in (5.5), the relative range is computed as the maximum value minus the minimum value divided by the mean value. If the relative range is less than the minimum imposed, the variation with the current variable is considered negligible, and the data may be approximated as a constant using the mean value.

$$Relative\ Range = \left| \frac{Max(y) - Min(y)}{\overline{y}} \right| \tag{5.5}$$

Once the parameters that control the extraction process have been established, then the process itself will be explained. To achieve the desired accuracy with a minimal polynomial order, the approximation process increases the order iteratively until the maximum imposed order or the accuracy requirements are reach. The same process is repeated iteratively as many times as variables considered. The output data of each stage becomes the input data for the next stage.

Each process step starts by checking the relative range of the input data, and – as exposed earlier

– if it's lower than the minimum imposed, the data is approximated by the mean value. This checking minimizes the number of regressions performed, and therefore the number of coefficients generated, if the dependence with the variable is negligible. This step can be considered an order 0 regression. It is easy to demonstrate that a polynomial regression of order 0 is equivalent to computing the mean value of the data, as shown in (5.6).

$$\left.\begin{aligned} A &= \sum_{k=0}^{m-1} x_k^0 = m \\ P &= p_0 \\ B &= \sum_{k=0}^{m-1} x_k^0 y_k = \sum_{k=0}^{m-1} y_k \end{aligned}\right\} \rightarrow A \cdot P = B \rightarrow p_0 = \frac{\sum_{k=0}^{m-1} y_k}{m} \tag{5.6}$$

If the relative range is beyond the threshold required, a linear regression is carried out (i.e. a polynomial regression with order 1). After the regression, the algorithm uses the coefficients obtained to generate an estimation of the original sequence of values. This generated sequence, is used to compute both the quadratic correlation coefficient and the maximum relative error between the input data and the estimation. If these two parameters comply with the requirements imposed, then the step is finished; if not, the regression is repeated increasing the polynomial order by 1, and so on until the error requirements are fulfilled, or the maximum order allowed is reach. The flowchart of the extraction process is depicted in Fig. 5.3.

The whole process described is carried out using multidimensional matrices, where the dimensions depends on the number of variables. At each step, for each input matrix, the process generates *m* – the order of the polynomial regression – output matrices with a dimension lowered by 1 with respect to the input matrix. Expression (5.7) shows the general form of each step, where a polynomial regression of order *m* with respect to variable x_1 is applied to an *n*-dimensional matrix *f* and the result are *m+1* (*n-1*)-dimensional matrices f_i.

$$\underbrace{f(x_1, x_2, \ldots, x_n)}_{\text{1 n-dim Matrix}} \rightarrow \underbrace{\sum_{i=0}^{m} f_i(x_2, \ldots, x_n) \cdot x_1^i}_{\text{m+1 (n-1)-dim Matrices}} \tag{5.7}$$

Finally, with the objective of achieving the best results with the smallest possible order, the process described is carried out using all possible cases for the sequence in which the variables are

taken for regression, i.e. the quality of the result may be different if the process starts with the variable x_1 and ends up with x_n instead of starting with x_n and ending with x_1. Although this could be time consuming for a large number of variables this step is to be executed only once per technology library, and ensures an optimal result. The way to determine which is the best sequence of variables, consists in comparing the original data with the estimations generated using the model function and the coefficients extracted. The case where the generated matrix is more similar to the original in terms of relative error is selected as the best combination.

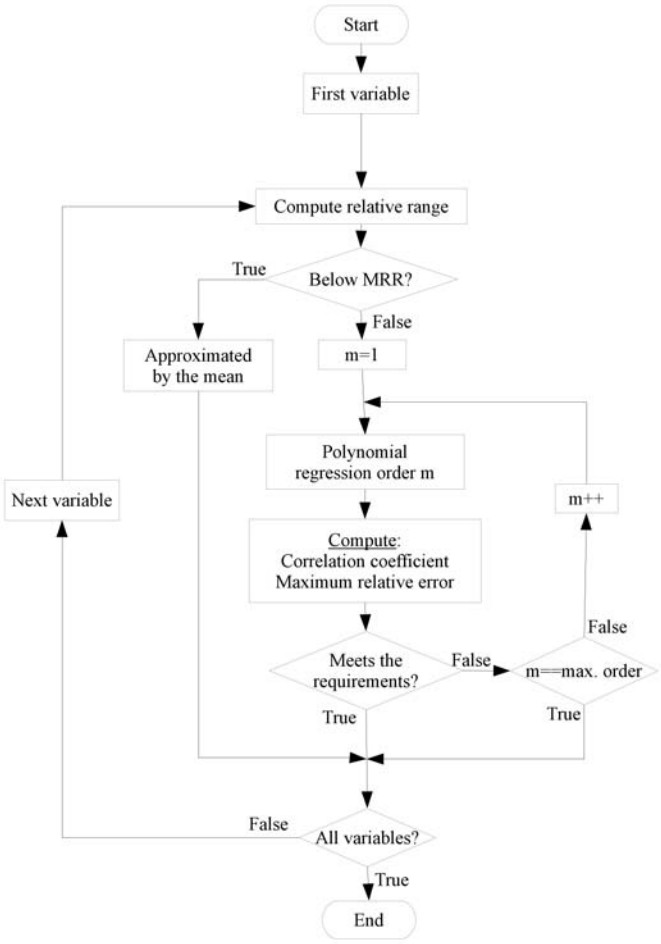


Figure 5.3: Flowchart of the model extraction algorithm

5.1.2. Gate simulation process

Delay model parameters are extracted from electrical simulation results using the algorithm exposed in section 5.1.1. The electrical simulations from which the model parameters are obtained, are done automatically and systematically for a given technology library, and consist of a set of iterative simulations. Each iteration uses a different combination of values for each variable considered, for which the propagation delay and output transition time for rising and falling input transition are determined. Such an iterative simulations are repeated for each gate input and each input vector that sensitizes that input. Fig. 5.4 shows the circuit structure used to simulate the delay through a gate: the gate is simply loaded with a capacitance and while a pulse source is applied to the active input and a sensitization input vector is applied to the side inputs.

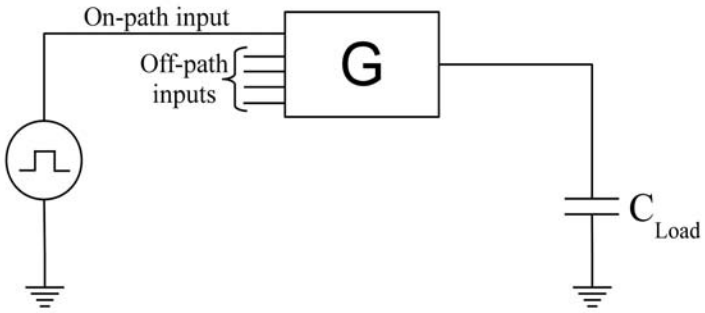


Figure 5.4: Gate delay simulation

The input pulse source is not an ideal pulse because an ideal pulse causes an excessive overshooting at the output waveform, due to its discontinuities at the derivative. One way to generate a real transition is using an ideal pulse source followed by a buffer and using the output of the buffer as source. However, this method does not allow controlling the transition time. To avoid the side effects of an ideal pulse, having a more realistic pulse resembling to a real transition that allows to controlling the transition time, we use an input source that generates a pulse using a sinusoidal function. The waveform generated is fully differentiable and presents a non-constant slope, as a real transition caused by a CMOS gate.

The analytical expression for a rising transition is shown in (5.8).

$$f(t)=\begin{cases}0 & : \quad t\le t_0\\ \dfrac{V_{DD}}{2}\left(\sin\left(\dfrac{K}{t_r}(t-t_0)-\dfrac{\pi}{2}\right)+1\right) & : \quad t_0\le t\le t_1\\ V_{DD} & : \quad t_1\le t\end{cases} \tag{5.8}$$

Where $K=2\cdot\arcsin(0.8)=1.85459$ and t_r is the transition time.

Fig. 5.5 depicts two transition waveforms, one corresponding to an ideal pulse source and the other is the result of the sinusoidal-based function (5.8). As shown the function provides a much more realistic waveform than an ideal pulse, avoiding the angular points that cannot be differentiated, and that induce an overshooting effect due to the abrupt derivative change. Comparing these waveforms with the one shown in the Fig. 5.6, that corresponds to a real transition at the output of an inverter, it may be concluded that the sinusoidal-based function generates a waveform very similar to a real transition produced at the output of a gate.

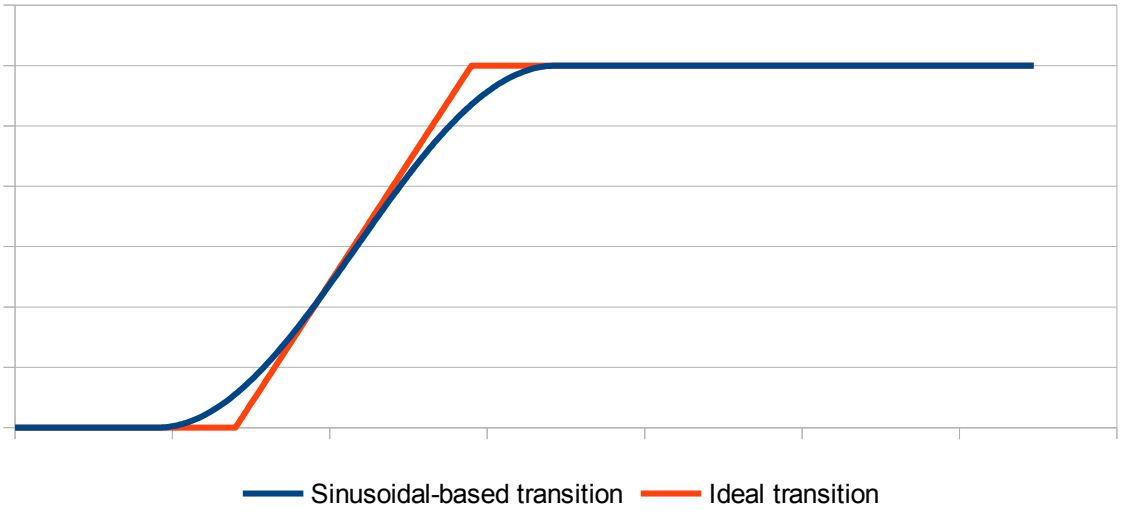


Figure 5.5: Transition generation

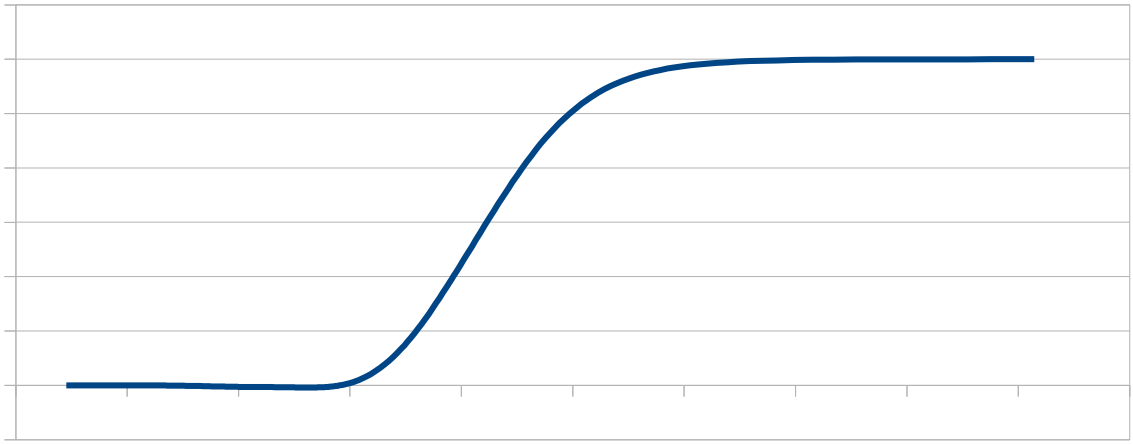


Figure 5.6: Real transition

As already explained, the algorithm to extract the multivariable polynomial model parameters, explores all the sequences in which the variables can be considered to obtain the best possible results. Fig. 5.7 represents the mean relative error and the number of model parameters for each variable sequence, using a maximum order of 2, and Fig. 5.8 shows the same results for the case of maximum order 3.

From the results represented in these graphs it is shown that the best case in terms of mean error does not corresponds to the case with more parameters. As expected, using a greater value for the maximum order, provides better values for the mean error, as shows Fig. 5.8 compared to Fig. 5.7.

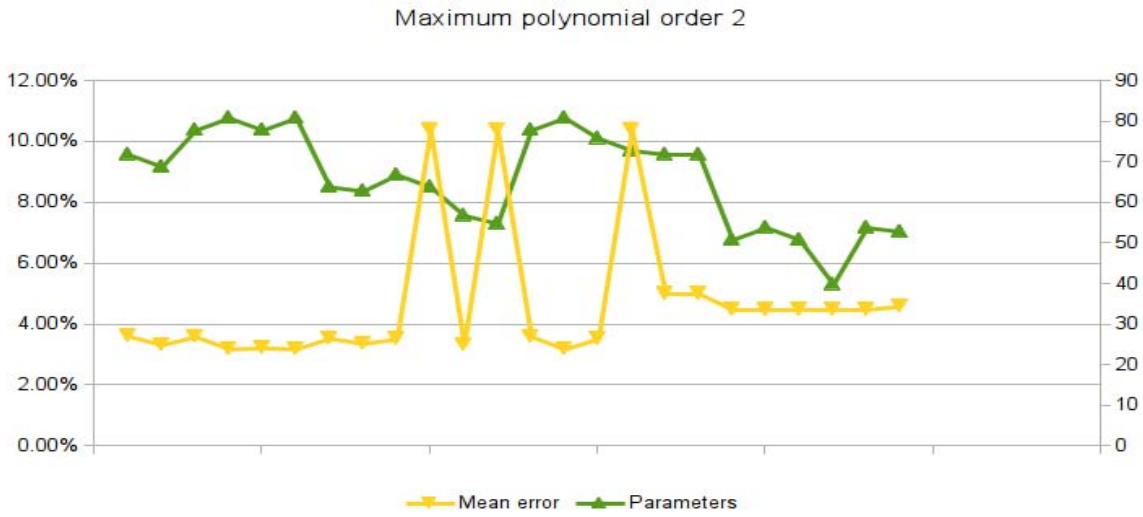


Figure 5.7: Mean error and number of coefficients

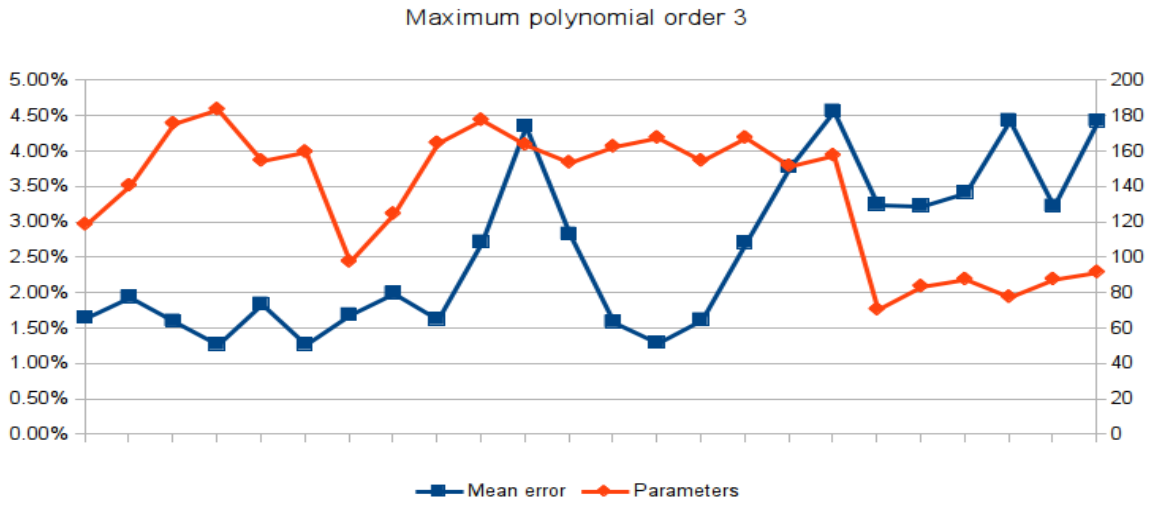


Figure 5.8: Mean error and number of coefficients

Previously, section 2.2.4 introduced the concept of matrix coefficients pre-computation for the polynomial model when some variable takes a constant value. The delay model depends on four variables, the output capacitance load, the input transition time, the temperature and the supply voltage. According to the application requirements some of the variables may take a constant value, and create coefficient matrices for specific situations.

For a digital circuit a structure where each gate of a circuit has its own coefficient matrix depending on the cell used for each gate can be created. Each gate coefficient matrix can be simplified using the technique of partial computing. Using the 4 variables model, results in each gate having a 4D coefficient matrix for each variable that takes a constant value, reducing the number of dimensions by 1.

The first model variable is the output gate load that depends on the input gates connected to this node and the contribution of the interconnect determined mainly by the circuit structure. Then the node capacitance can be considered constant, and the coefficient matrix of each gate can be simplified respect to the output load variable.

The supply voltage of each gate usually has a constant value although in some circuits the supply voltage varies dynamically, and in some analysis the voltage drops caused by the circuit activity must be taken into account. However, for most analysis the supply voltage is assumed constant. In these situations the gates coefficient matrix can be reduced respect to the supply voltage variable, even for circuits with regions with different supply voltages. Each gate coefficient matrix is simplified using its own supply voltage. To analyze the impact of the voltage drops on the delay, the

coefficient matrices of the gates is simplified using a voltage map, where each gate may use a different voltage value. The voltage map can be generated using an application that estimates the voltage drops based on the circuit activity estimations.

Temperature can be treated similarly to the supply voltage. The coefficient matrices can be reduced using a constant temperature value for all the gates, or by applying a thermal map where each gate operates at different temperate. Specific tools may perform an estimation of the heating of each circuit zone.

The input transition time of a gate depends on the path considered, and all the variables that impact the previous gates. For this reason this variable cannot be considered constant for a given gate. A possible exception is the first gate of a path if the transition applied at the input of the circuit is considered constant.

Based on the previous arguments, using a constant value for the capacitance of each node, and applying thermal and voltage maps, the 4D matrix of each gate can be reduced to a vector of coefficients. This reduction is performed once for each gate, and helps to improve the timing analysis speed avoiding repeated computations. If a timing analysis must be performed at various operating conditions, various sets of coefficient matrices can be precomputed using different values, and used simultaneously.

5.1.3. Delay model accuracy verification

To verify the proper delay estimation performed by the analytical model the delays computed by the model are compared with the estimation provided by a commercial tool and delays measured from electrical simulations. Delay estimations are compared at gate-level and path-level.

Tables 5.1, 5.2 and 5.3, provide the error in the delay estimation given by the tool developed and the commercial tool, when compared to electrical simulations as a reference. These tables contain the mean and maximum delay errors for the entire path and for an individual gate. Results show that the delay model used to estimate the gate propagation delay provides more accurate results than the commercial tool considered.

In all the cases investigated the polynomial model provides better delay estimations than the look-up table model used by the commercial tool, even considering a first order model. The analytical form of the model reduces considerably the computation time leading to faster delay estimations.

Table 5.1: Delay comparison vs electrical simulation (130nm)

	Developed model				Commercial tool			
ISCAS Circuit	**Mean path error**	**Max path error**	**Mean gate error**	**Max gate error**	**Mean path error**	**Max path error**	**Mean gate error**	**Max gate error**
c17	1.92%	4.61%	1.91%	5.26%	9.94%	21.16%	8.63%	24.16%
c432	1.24%	2.59%	6.02%	11.42%	6.76%	7.53%	17.22%	44.11%
c499	3.31%	5.20%	6.44%	9.21%	4.11%	4.12%	11.70%	25.37%
c880a	2.11%	7.38%	4.63%	6.99%	3.31%	7.11%	13.78%	64.13%
c1355	2.67%	8.46%	3.41%	7.19%	3.79%	6.98%	14.25%	56.78%
c1908	1.66%	3.65%	4.13%	14.02%	7.39%	8.71%	17.99%	61.60%
c2670	0.59%	1.08%	4.10%	16.57%	8.95%	27.89%	15.31%	306.95%
c3540	3.04%	5.63%	5.33%	13.84%	5.10%	5.10%	19.45%	109.07%
c5315	6.31%	7.41%	6.32%	19.43%	10.60%	13.59%	17.75%	53.62%
c6288	2.39%	7.86%	3.50%	18.24%	10.59%	22.66%	15.38%	82.24%
c7552	5.38%	9.67%	7.24%	11.58%	11.59%	21.17%	16.23%	58.45%

Table 5.2: Delay comparison vs electrical simulations (90nm)

	Developed model				Commercial tool			
ISCAS Circuit	**Mean path error**	**Max path error**	**Mean gate error**	**Max gate error**	**Mean path error**	**Max path error**	**Mean gate error**	**Max gate error**
c17	2.93%	5.12%	3.21%	5.84%	19.92%	40.58%	18.42%	42.08%
c432	4.87%	8.87%	6.36%	20.41%	17.92%	20.23%	24.39%	73.58%
c499	6.73%	11.08%	7.96%	17.36%	16.15%	19.05%	24.77%	136.92%
c880a	6.21%	9.67%	6.74%	15.68%	18.31%	53.27%	19.71%	97.03%
c1355	4.31%	7.67%	6.39%	12.43%	17.83%	29.64%	36.47%	53.21%
c1908	2.88%	5.33%	6.59%	19.21%	17.67%	21.76%	28.55%	98.82%
c2670	2.32%	3.52%	5.17%	16.38%	16.26%	29.64%	21.71%	231.97%
c3540	4.11%	6.87%	6.21%	12.09%	15.89%	31.45%	26.87%	66.88%
c5315	5.64%	7.63%	5.16%	13.56%	18.52%	28.79%	20.36%	60.09%
c6288	3.55%	6.61%	4.94%	11.34%	13.25%	23.74%	18.56%	58.34%
c7552	6.31%	8.82%	5.61%	10.98%	16.23%	39.25%	23.34%	67.87%

Table 5.3: Delay comparison vs electrical simulations (65nm)

	Developed model				Commercial tool			
ISCAS Circuit	**Mean path error**	**Max path error**	**Mean gate error**	**Max gate error**	**Mean path error**	**Max path error**	**Mean gate error**	**Max gate error**
c17	4.30%	8.24%	4.13%	8.24%	29.91%	59.99%	28.20%	59.99%
c432	7.82%	9.75%	9.29%	10.53%	29.09%	32.94%	31.56%	103.06%
c499	2.94%	4.64%	4.69%	11.81%	28.20%	33.99%	37.84%	248.47%
c880a	1.19%	7.65%	3.68%	13.49%	33.32%	99.43%	25.64%	129.93%
c1355	0.75%	2.40%	4.04%	12.27%	27.95%	34.82%	39.11%	136.04%
c1908	4.05%	5.96%	6.24%	16.86%	23.57%	31.39%	28.11%	156.99%
c2670	2.35%	6.81%	5.67%	14.51%	19.87%	29.60%	21.01%	49.58%
c3540	3.98%	7.61%	9.33%	19.12%	25.67%	40.12%	32.11%	77.43%
c5315	5.87%	8.64%	8.81%	15.49%	31.01%	57.64%	26.28%	81.42%
c6288	3.29%	8.75%	7.81%	17.63%	23.47%	62.37%	34.69%	64.58%
c7552	5.42%	11.01%	8.43%	20.09%	26.33%	42.11%	33.84%	67.12%

5.2. Effective capacitance extraction

In chapter 2 was introduced the theoretical concepts related to the input capacitance of a CMOS logic gate. Then, after introducing the main contributions to the input capacitance of a CMOS standard cell, the next step is to evaluate different approaches to easily model all of these effects using a steady capacitance value.

An effective input capacitance value for a standard cell input can be obtained by multiple methods, since this effective value is only an approximation to the real capacitive effect, there is not an exact way to extract its value, and depending on the specific use of the effective capacitance, one method may provide better results than other.

Various methods were analyzed, to evaluate the one working better when used to estimate circuit delays. The term Characterized-Gate (CG) will be used to refer to the standard cell from which we get the input capacitance. Various strategies are detailed:

- Charge integration: A method widely used in commercial tools for calculating the logic cells capacitance is by integrating the input current into the gate during a transition. With this method, the capacitance value can be computed using the basic definition of the capacitance, given by (5.9), as the quotient between the charge stored and the voltage variation between the capacitor terminals. The dynamic behavior of the real capacitive to be described may provide a value that depends on the conductance used to charge (discharge) the node.

$$C = \frac{Q}{V} \tag{5.9}$$

  As already was shown by [58] this results technique provides an overestimation of the capacitance value up to 33%, depending of the specific technology when applied to timing analysis

- RC time constant: This technique is based on the theoretical definition of the time constant (τ) of an RC circuit, being the product of the resistance value and the capacitance value (5.10). The RC constant is the time required to charge or discharge the capacitor at 63.2 percent, through the resistor. This method works applying a voltage pulse through a resistance to the input of the CG, where the cell acts as the capacitor of the RC circuit. By

measuring the charging (discharging) time of the input node, the equivalent capacitance value can be computed easily from the measured time and the resistance value. The main issue of this technique is which value must been chosen for the resistance. The capacitance value computed does not remain constant when the resistance value changes.

$$\tau = R \cdot C \qquad v(t) = V_{DD} \cdot \left(1 - e^{-\frac{1}{\tau}t}\right) \tag{5.10}$$

Test experiments carried over exhibit a capacitance estimation variation reaching values beyond 100% when considering different resistance values (Table 5.4 shows the relative capacitance difference for various resistance values for two 65nm standard cells). These results make clear the invalidity of this technique to estimate the effective capacitance of a CMOS gates.

- Empirical adjustment of propagation delay: The previous methods are based on theoretical concepts of a capacitance to get an effective value. However, the dynamic nature of the transistor capacitance does not match the theoretical concepts derived for a static capacitance and do not provide accurate results. The fully empirical method adjusts the cell capacitance value to the one providing the observed delay.

  The empirical method finds the capacitive load for which a reference gate exhibits the same propagation delay than when it is loaded with an instance of the CG. This way to extract the capacitance value is specific to the timing analysis since its value is adjusted comparing the propagation delay of a gate. Such a capacitance estimation can be unsuitable for other applications aside from propagation time estimation.

Table 5.4: RC Constant capacitance extraction

	HS65_LS_AND2X4		HS65_LS_NOR2X3	
	Input A	Input B	Input A	Input B
10 Ω	–	–	–	–
50 Ω	+40.03%	+40.97%	+46.86%	+42.96%
100 Ω	+88.87%	+91.85%	+107.93%	+97.65%
200 Ω	+196.38%	+203.06%	+239.02%	+216.37%

Thanks to its delay-based nature, the empirical method provides the higher accurate delay

estimation compared to electrical-level simulations, and is the method used from now on. Next, this method is described in detail.

5.2.1. Delay-based effective capacitance extraction

The generic circuit scheme used to extract the effective capacitance of an input node of a given standard cell is shown in Fig. 5.9. A voltage pulse is applied to the inputs of two instances of the same cell type (Ref_1 and Ref_2). One of these references gates (Ref_1) is loaded with one instance of the CG, while Ref_2 has a pure capacitive load. The goal is to adjust the value of the capacitor, until the propagation delay of both reference gates matches. Evidently, if the reference gate has more than one input, the logical values applied to the other inputs must allow propagating the transition through the gate. To avoid this issue, the best solution is to choose a gate with a single input as a reference gate, like an inverter or a buffer.

The effective capacitance is iteratively approximated, each process step simulates the circuit and compares the propagation delay through the two reference gates. The capacitor value is increased or decreased depending on if the delay through Ref_1 is greater or lesser than the Ref_2, and the simulation repeats until the difference in the propagation delay through the two branches is below a defined tolerance limit. To approach progressively to the capacitance value that equals the delay, the differential applied to the capacitance value is smaller at each simulation step. When this iterative process ends, the final capacitor value is taken as the effective capacitance of the gate under test.

Simulations carried out demonstrate that the effective capacitance value obtained for a given cell, have a negligible dependence with the input transition time applied to the input of the reference gates. Resulting in a mean variation under 0.5% for both extreme values tested. The type of cell used as a reference gate has a larger impact on the capacitance estimation, than the input transition time. The method was tested for all combinational cells in a commercial 65nm CMOS library, using buffers of different size as reference gates. The variation on the estimated value was about a 5% for the worst case tested, and a mean value of the variation below 2%. However, this is for the two extreme cases, and can be reduced by taking an intermediate value.

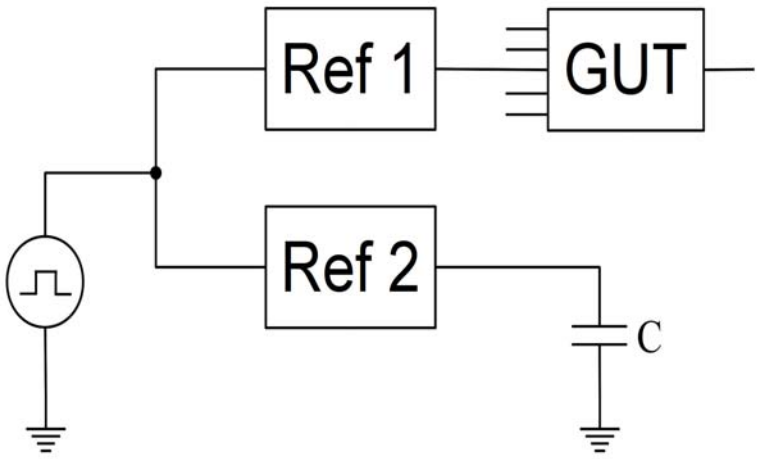


Figure 5.9: C extraction circuit

5.3. Delay dependency with sensitization vector

When a circuit design is synthesized using standard cells, CAD algorithms are designed to reduce circuit area, power consumption and propagation delays in addition to optimizing other parameters. To accomplish this goal, synthesis tools use library complex gates, i.e. circuit structures that combine primitive logic functions like NOT, AND, OR, NAND, NOR, in a single CMOS structure that reduces the number of transistors required to perform a given logic function. Typically, complex gates comprise a combination of few primitive functions although more complex functions like full-adders or multiplexers are also common. In the context of timing analysis, a typical characteristic of complex gates vs. basic gates is that, in general, it is possible to find more than one vector that sensitizes each gate input, while single gates have typically only one sensitization vector [59]. In this work we show that the gate delay when propagating a transition through a given input of a complex gate may vary significantly depending on the input vector used to sensitize such an input with the consequent impact on the circuit-level timing computation. This delay variation is shown to be not negligible, being similar to the delay variation caused by process parameter fluctuations. We also show that the sensitization vector impact on the delay is also of the same order of magnitude than the delay due to the interconnect system.

In some works, complex gates are converted to primitive gates prior to timing analysis thus applying the delay model to basic gates [60]. This methodology may be a source of inaccuracies since the circuit used for simulation has a topology that differs from the actual circuit structure being finally manufactured [61]. Other works analyze the delay of complex gates through a transistor-level approach [62][63][64], or using a current source model (CSM) [65], providing good accuracy at the cost of very complex expressions that result in a slow computation time at the circuit level [66]. Moreover, some works use alternative gate delay models based on neural networks [67]. However, the gate delay modeling techniques achieving a better tradeoff between computation time, accuracy and flexibility are based on analytical models [68]. In addition, many works related to timing analysis do not consider specifically the case of complex gates [69][70][71] [72], although some of them could be easily extended to include complex gates while others could have it more difficult to consider this effect.

5.3.1. Gate-level analysis

Without lose of generality, we illustrate the delay dependence with the sensitization vector using four complex gates included in almost all standard cell libraries. The results and arguments provided for these four gates may be extrapolated to any complex cell, even in cells with a more complex structure than the cells used as examples the impact over the delay may be more relevant. The four gates analyzed are:

- AO22 (referred to as AO2N in some technologies), being a four input gate that implements the logic function in (5.13), whose logic symbol and the CMOS transistor topology is shown in Fig. 5.10. Table 5.5 shows the sensitization vectors for each gate input. The logic value "T", represents a transition either rising.

$$Out = A * B + C * D \tag{5.11}$$

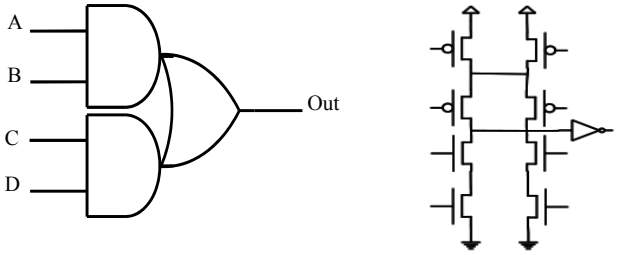


Figure 5.10: Gate AO22

- OA12 (AO7N in some technologies), being a three input gate for which only one of its inputs has multiple input vectors to sensitize the gate. The gate logic function is given by (5.13), its symbol and transistor topology are shown in Fig. 5.11, and the sensitization vectors in Table 5.6.

$$Out = (A + B) * C \tag{5.12}$$

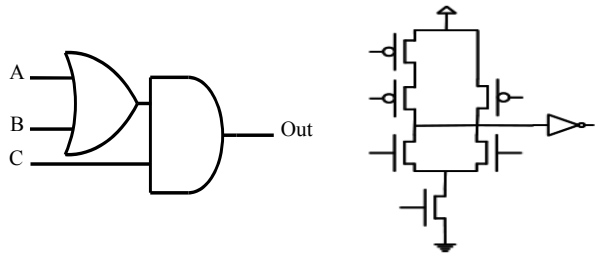


Figure 5.11: Gate OA12

- CB4I6 (also known as AO20N), a four input gate with an increasing number of sensitization vectors for each input. Its logic function is given by (5.14), and symbol and transistor topology are shown in Fig. 5.12, and its sensitization vectors in the Table 5.7.

$$Out = (A + B) * C + D \tag{5.13}$$

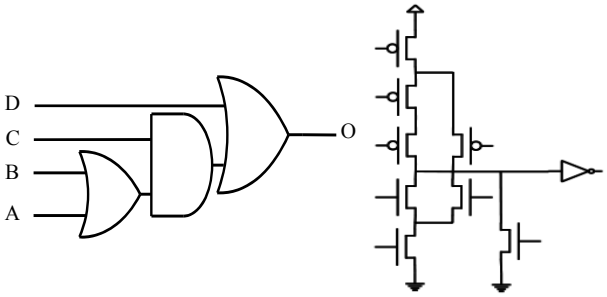


Figure 5.12: Gate CB4I6

- AOI212 (also known as AO10), a five input gate. This is an inverting gate, thus, do not has an inverter at its output. Its logic expression is given by (5.14), Fig. 3.14 shows its symbol and CMOS transistor topology, and its sensitization vectors in the Table 5.8.

$$Out = \overline{A * B + C * D + E} \tag{5.14}$$

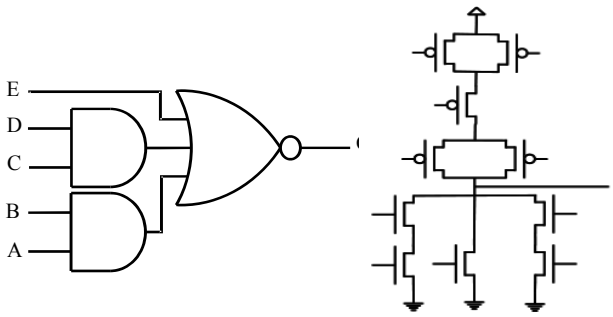


Figure 5.13: Gate AOI212

Table 5.5: AO22 Propagation Table

	A	B	C	D	Out
Vector A1	T	1	0	0	T
Vector A2	T	1	1	0	T
Vector A3	T	1	0	1	T
Vector B1	1	T	0	0	T
Vector B2	1	T	1	0	T
Vector B3	1	T	0	1	T
Vector C1	0	0	T	1	T
Vector C2	1	0	T	1	T
Vector C3	0	1	T	1	T
Vector D1	0	0	1	T	T
Vector D2	1	0	1	T	T
Vector D3	0	1	1	T	T

Table 5.6: OA12 Propagation Table

	A	B	C	Out
Vector A1	T	0	1	T
Vector B1	0	T	1	T
Vector C1	1	0	T	T
Vector C2	0	1	T	T
Vector C3	1	1	T	T

Table 5.7: CB4I6 Propagation Table

	A	B	C	D	Out
Vector A1	T	0	1	0	T
Vector B1	0	T	1	0	T
Vector C1	1	0	T	0	T
Vector C2	0	1	T	0	T
Vector C3	1	1	T	0	T
Vector D1	0	0	0	T	T
Vector D2	0	1	0	T	T
Vector D3	1	0	0	T	T
Vector D4	1	1	0	T	T
Vector D5	0	0	1	T	T

Table 5.8: AOI212 Propagation Table

	A	B	C	D	E	Out
Vector A1	T	1	0	0	0	$\overline{T}$
Vector A2	T	1	0	1	0	$\overline{T}$
Vector A3	T	1	1	0	0	$\overline{T}$
Vector B1	1	T	0	0	0	$\overline{T}$
Vector B2	1	T	0	1	0	$\overline{T}$
Vector B3	1	T	1	0	0	$\overline{T}$
Vector C1	0	0	T	1	0	$\overline{T}$
Vector C2	0	1	T	1	0	$\overline{T}$
Vector C3	1	0	T	1	0	$\overline{T}$
Vector D1	0	0	1	T	0	$\overline{T}$
Vector D2	0	1	1	T	0	$\overline{T}$
Vector D3	1	0	1	T	0	$\overline{T}$
Vector E1	0	0	0	0	T	$\overline{T}$
Vector E2	0	0	0	1	T	$\overline{T}$
Vector E3	0	0	1	0	T	$\overline{T}$
Vector E4	0	1	0	0	T	$\overline{T}$
Vector E5	0	1	0	1	T	$\overline{T}$
Vector E6	0	1	1	0	T	$\overline{T}$
Vector E7	1	0	0	0	T	$\overline{T}$
Vector E8	1	0	0	1	T	$\overline{T}$
Vector E9	1	0	1	0	T	$\overline{T}$

With the objective to illustrate to the variation of the propagation delay of the gates when change the sensitization vector, we carried extensive electrical simulations to compute the gate delays through each input for all the sensitization vectors for three commercial CMOS technologies (130nm, 90nm and 65nm) at nominal supply voltage and 25 ºC. Each gate was loaded with 4 gates equal to the one being analyzed.

Tables 5.9 to 5.12 show some of the delay results obtained when propagating both rising and falling transition through each input of the four complex gates considered in this analysis, for a sample of the sensitization vectors of each gate. For each gate, the Vector X1 (with X being A, B, C, D or E) delay is taken as a reference to which of the other Vectors delay Xi, (i > 1) are referred.

Table 5.9: Propagation delay variation for AO22

	130nm		90nm		65nm	
	In Rise	**In Fall**	**In Rise**	**In Fall**	**In Rise**	**In Fall**
Vector A1	118.23	117.90	58.93	64.18	82.58	82.66
Vector A2	122.68	146.28	61.96	78.11	85.68	99.18
Vector A3	118.49	137.80	58.95	71.87	82.67	94.81
A2 vs A1	3.76%	24.07%	5.14%	21.70%	3.75%	19.98%
A3 vs A1	0.22%	16.88%	0.03%	11.97%	0.11%	14.69%
Vector B1	121.32	125.90	57.97	65.95	84.74	87.69
Vector B2	125.88	157.00	60.82	80.58	87.88	105.53
Vector B3	121.59	148.44	58.06	74.33	84.85	101.14
B2 vs B1	3.76%	24.71%	4.90%	22.18%	3.71%	20.34%
B3 vs B1	0.23%	17.91%	0.15%	12.70%	0.13%	15.33%
Vector C1	146.16	139.68	74.20	75.89	100.80	101.86
Vector C2	143.54	166.78	74.31	88.55	101.25	118.66
Vector C3	139.05	158.27	71.32	84.03	98.37	112.74
C2 vs C1	-1.79%	19.41%	0.14%	16.69%	0.45%	16.49%
C3 vs C1	-4.86%	13.31%	-3.89%	10.73%	-2.41%	10.68%
Vector D1	146.43	144.41	71.81	79.55	96.19	98.42
Vector D2	143.96	173.14	71.67	93.38	96.54	115.65
Vector D3	139.38	164.76	68.76	88.75	93.72	109.96
D2 vs D1	-1.69%	19.90%	-0.20%	17.38%	0.36%	17.50%
D3 vs D1	-4.82%	14.10%	-4.24%	11.58%	-2.57%	11.73%

Table 5.10: Propagation delay variation for OA12

	130nm		90nm		65nm	
	In Rise	**In Fall**	**In Rise**	**In Fall**	**In Rise**	**In Fall**
Vector C1	122.36	136.54	61.79	77.88	76.32	63.77
Vector C2	108.45	129.56	55.19	73.82	71.66	61.56
Vector C3	98.72	131.85	52.61	75.24	67.27	62.40
C2 vs C1	-11.37%	-5.11%	-10.68%	-5.21%	-6.10%	-3.46%
C3 vs C1	-19.31%	-3.43%	-14.85%	-3.38%	-11.85%	-2.15%

Table 5.11: Propagation delay variation for CB4I6

	130nm		90nm		65nm	
	In Rise	**In Fall**	**In Rise**	**In Fall**	**In Rise**	**In Fall**
Vector D1	156.84	139.84	67.63	78.33	81.98	82.75
Vector D2	159.12	169.82	70.84	97.39	79.73	85.45
Vector D3	153.89	169.41	68.91	96.95	80.73	86.44
Vector D4	146.45	163.44	67.19	93.33	76.82	81.97
Vector D5	147.93	182.45	64.24	96.51	84.78	116.65
D2 vs D1	1.45%	21.44%	4.74%	24.33%	-2.74%	3.26%
D3 vs D1	-1.88%	21.15%	1.88%	23.78%	-1.52%	4.45%
D4 vs D1	-6.62%	16.88%	-0.65%	19.16%	-6.30%	-0.95%
D5 vs D1	-5.68%	30.48%	-5.02%	23.21%	3.42%	40.97%

Table 5.12: Propagation delay variation for AOI212

	130nm		90nm		65nm	
	In Rise	**In Fall**	**In Rise**	**In Fall**	**In Rise**	**In Fall**
Vector E1	106.23	119.58	63.06	73.25	59.32	79.88
Vector E2	105.95	144.12	62.78	87.61	57.96	93.30
Vector E3	107.24	150.96	63.99	89.38	57.70	89.84
Vector E4	101.03	144.72	61.83	87.60	59.65	98.85
Vector E5	100.64	171.08	61.49	103.59	58.27	114.31
Vector E6	102.08	179.20	62.89	105.99	57.97	109.95
Vector E7	99.70	138.21	60.68	82.74	59.38	93.67
Vector E8	99.38	163.06	60.39	97.76	58.00	108.26
Vector E9	100.64	171.11	61.63	100.15	57.72	103.92
E2 vs E1	-0.26%	20.52%	-0.45%	19.60%	-2.29%	16.79%
E3 vs E1	0.95%	26.23%	1.47%	22.02%	-2.74%	12.46%
E4 vs E1	-4.89%	21.02%	-1.96%	19.59%	0.56%	23.74%
E5 vs E1	-5.27%	43.06%	-2.50%	41.42%	-1.77%	43.10%
E6 vs E1	-3.90%	49.85%	-0.27%	44.69%	-2.27%	37.64%
E7 vs E1	-6.15%	15.58%	-3.78%	12.96%	0.11%	17.26%
E8 vs E1	-6.45%	36.36%	-4.24%	33.46%	-2.23%	35.51%
E9 vs E1	-5.27%	43.09%	-2.28%	36.72%	-2.70%	30.08%

Results in Tables show propagation delay variations with the input sensitization vector that reach up to 50% (49.85%) depending on the gate structure, input transition and technology. The delay variation for the 65nm technology may get to up to 43% (Vector E5 vs. Vector E1 for gate AOI212 propagating a falling input transition) suggesting that this variation may induce a large variance at the circuit level. The values of delay variation in function of the sensitization vector presented in Tables 5.9-5.12, demonstrates that this factor must be considered to perform an accurate delay estimation.

The cell AOI212 presents the largest variations of all cases of the table, that is because this is an inverting gate, while the others are non-inverting. The non-inverting cells are constructed using an inverting cells followed by an inverter. The effects of the sensitization vector are manifested mainly on the first stage, and the output inverter reduces the impact on the cell delay.

5.3.2. Transistor level analysis

We investigated the root cause of the delay variations with the sensitization vector to get insight on this phenomenon through a transistor-level analysis. This analysis is carried on the two first gates considered since it was observed that the delay variation root cause is common to all gates.

The complex gates considered implement non-inverting functions, and require an output inverter for a CMOS implementation. Such inverter does not influence the delay variation with the sensitization vector and therefore it is not considered in the transistor-level analysis. Fig. 5.15 shows the transistor-level analysis for gate AO22 and represents the three input vectors that propagate a falling transition through Input A. A solid cross on a transistor indicates that such device is OFF, while a solid arrow close to a device indicates that such transistor is ON. A dashed cross or arrow represents that such a transistor makes a transition and indicates the final state once the switching input is at its final state (i.e. a dashed arrow indicates a transistor that switched from OFF to ON, while a dashed cross indicates a transistor that changed from ON to OFF).

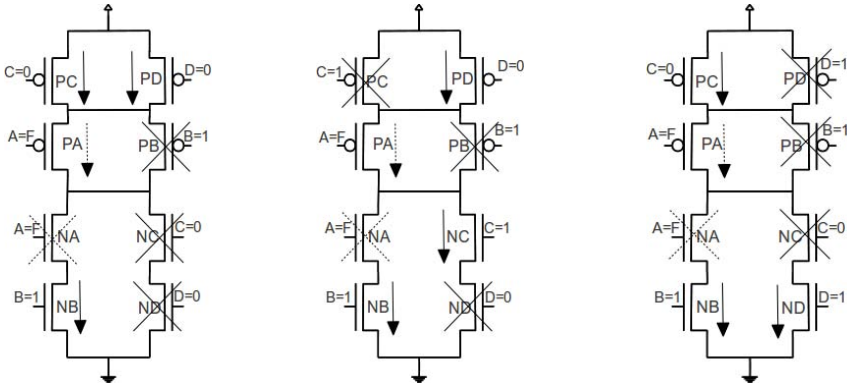


Figure 5.14: Gate AO22 transistor-level schematic and current paths for each sensitization vector. (output inverter not shown)

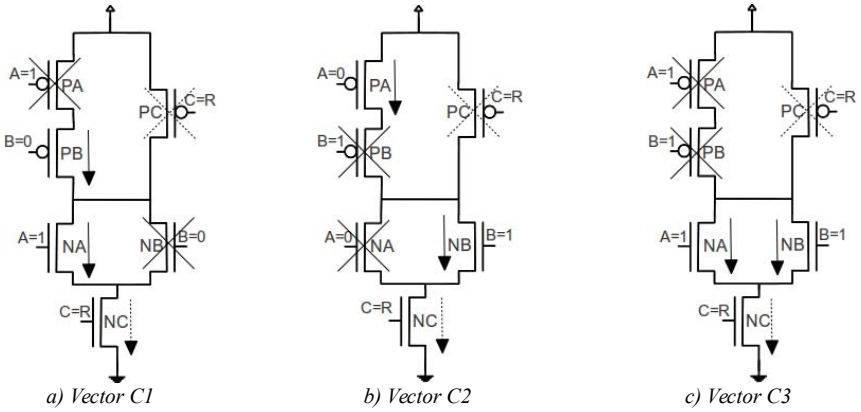


Figure 5.15: Gate OA12 transistor-level schematic and current paths for each sensitization vector. (output inverter not shown)

Results in Table 5.18 show that the transition in Fig. 5.14a corresponds to the fastest case, while Fig. 5.14b corresponds to the slowest one. As shown in Fig. 5.14, the current charging the output node must pass always through transistor PA. In the fastest case, both parallel transistors PC and PD are ON, allowing a higher current through PA, leading to a quicker charge of the output node. In the

other two cases only one of the two top parallel transistors (either PC or PD) is ON resulting in less current available to charge the output and hence resulting in a bigger delay. The relative delay difference between Vector A2 and Vector A3, is due to the transistor NC being ON for Vector A2 and OFF for Vector A3. When this device is ON, it creates an additional current path that charges internal parasitic capacitors. Such an additional current is taken from the current driven by the pMOS devices that therefore does not contribute to charge the output resulting in a comparatively longer transition.

To better illustrate this effect, Fig. 136 shows the dynamic currents through the pMOS, nMOS and output node in gate AO22 for the three input vectors discussed. The solid line represents the current through the pMOS transistors charging the output node, while the dashed line represents the current through the nMOS transistors drained from the output node. The dotted line is the difference between previous currents, and corresponds to the net current charging the output node. Vector A1 (Fig. 5.14a) corresponds to the fastest transition and exhibits the larger output load component as it drives the larger current through the pMOS transistor, and the smaller current through the nMOS ones (this later component being equal to the one in Vector A3). The slowest transition (in Fig. 5.14b) has a current component through the nMOS transistors being almost twice than that of the other two cases. This current reduces the effective current that charges the output node as shown in Fig. 5.16b.

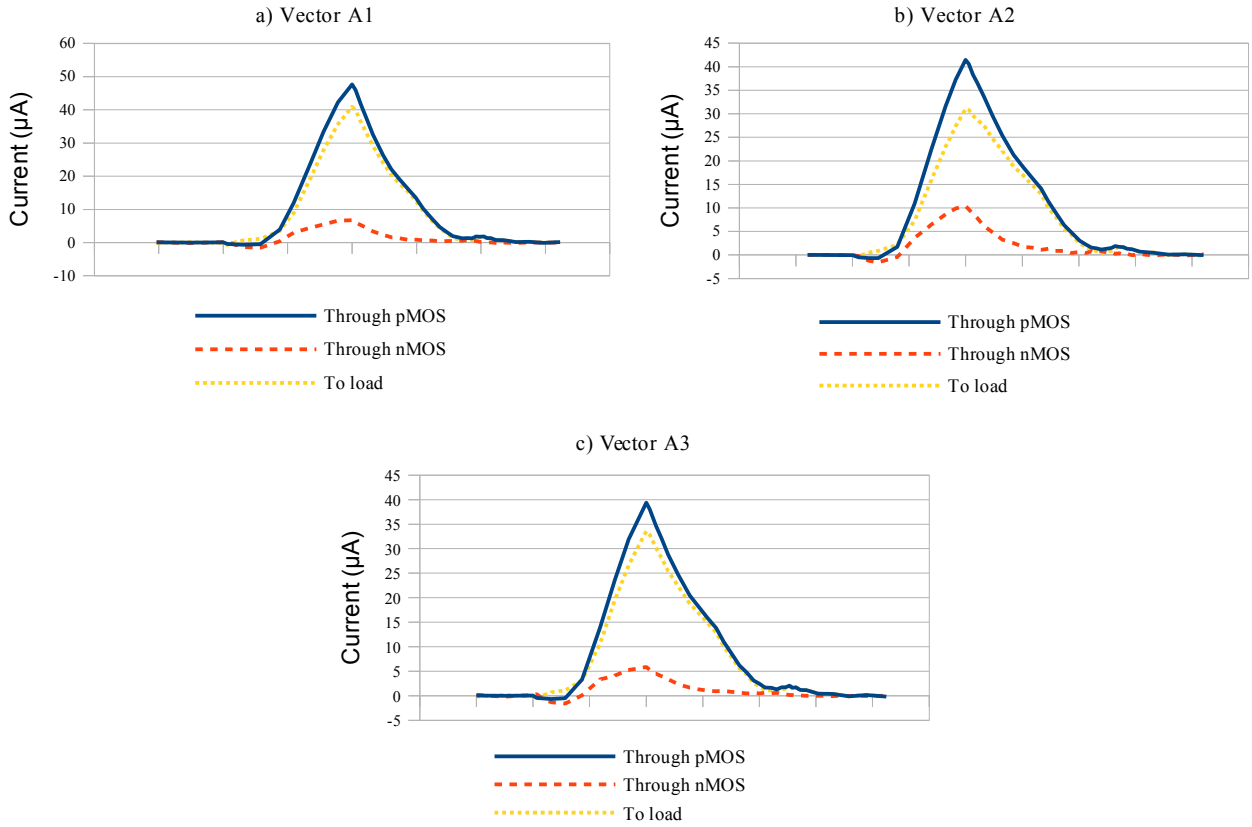


Figure 5.16: Internal currents of AO22

The gate OA12 behavior is analogous to the AO22 case. Fig. 5.15 shows the transistor-level diagram for each sensitization vectors that pass a rising transition at input C toward the gate output. Fig. 5.15c corresponds to the fastest transition. For this input vector, transistors NA and NB are both ON, increasing the current available through NC with respect to the other two cases where only NA or NB are ON. The Vector C2 transition (Fig. 5.15b) shows a delay slightly larger than that for Vector C1 (in both cases only one nMOS transistor is ON in the parallel structure) since transistor PB is ON increasing the amount of charge that must be drained from the output node when discharging the internal parasitic capacitors.

The analysis carried over in this section, together with the results shown in Table 5.18, highlight that if a logic gate has more than one sensitization vector for a given input, it is important to consider which input vector is actually applied to sensitize such input to the gate when performing timing analysis.

5.3.3. Circuit-level relevancy

As an initial experiment to analyze the impact of the multiple vector sensitization at the circuit level, we took the 1000 slowest paths of the ISCAS'85 benchmark circuits and computed how many

of such paths contained multiple sensitization vectors. Results are given in Table 5.13 showing that, for the large circuits (starting from the ISCAS c499) in almost all cases, the first 1000 slowest paths contain multiple-input gates highlighting the relevancy that this phenomenon might have at the circuit level. This is due to fact that technology libraries include many complex gates, typically used by the synthesis algorithms to reduce area, delay and power.

Table 5.13: Circuit-level multi-sensitization impact

Circuit	Slowest Path is multi-input	# of Multi-Input into 1000 Slowest paths
c17	No	24
c432	Yes	774
c499	Yes	1000
c880	Yes	1000
c1355	Yes	1000
c1908	Yes	1000
c2670	Yes	1000
c3540	Yes	1000
c5315	Yes	1000
c6288	Yes	1000
c7552	Yes	990

5.3.4. Sensitization vector impact on timing analysis

To illustrate the significance of the sensitization vector for an accurate path delay estimation, firstly is shown the behavior of a path identification algorithm not considering the specific vector on a test circuit and then, the results for benchmark circuits are compared with a commercial tool, showing the relevance of this consideration.

5.3.4.1. Test circuit

We first report initial results on a simple circuit shown in Fig. 5.17 to illustrate how the developed algorithm works compared to a commercial tool in the case of path with multiple sensitization vectors. The critical path of the sample circuit in Fig. 5.17 passes through input A of an AO22 complex gate (shown in a dashed box).

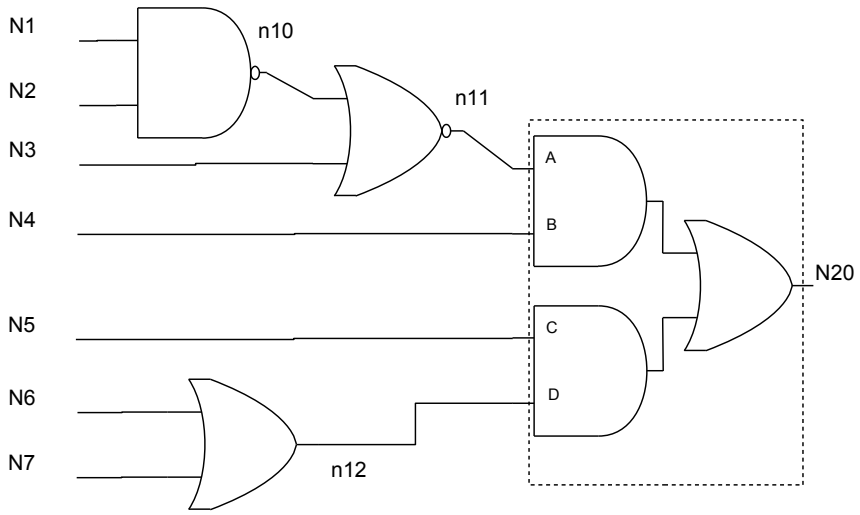


Figure 5.17: Test circuit

The easiest way to sensitize the complex gate leads to the smaller propagation delay for this path, although it is also possible to sensitize the gate with an input vector that exhibits a larger delay. The commercial tool correctly provides the critical path that propagates a falling edge through nodes N1-n10-n11-N20, as expected. The input vector used to sensitize the critical path is:

N1=F N2=1 N3=0 N4=1 N5=0 N6=X N7=X

and corresponds to the easiest option that assigns a logic 0 to node N5 and therefore doesn't require assigning n12 nor justifying its value to an input node. Setting N5 to 0 provides the shortest way to sensitize the AO22 gate, but ignores another case having a larger propagation delay for such path. This can be obtained sensitizing gate AO22 with a vector that results in a larger delay. This second vector requires a more complex justification process to assign logic values until reaching an input node.

The algorithm developed in the framework provides two paths passing through the same nodes and starting with a falling transition, each with different input vector. One is the same vector provided by the commercial tool, while the second one is:

N1=F N2=1 N3=0 N4=1 N5=1 N6=0 N7=0

Table 5.14 provides the delay obtained from electrical simulations of the critical path for the two input vectors. It is shown that the additional path provided by the tool developed exhibits a delay increase of 8.6% with respect to the one given by the commercial tool. Such an erroneous estimation is due to not considering the multiple sensitization vectors of complex gates. The tool proposed in this work identifies correctly the path with larger delay.

Table 5.14: Delay vs Input vector for the simple circuit in Fig. 5.17

Input vector	Delay (ps)
N1=F, N2=1, N3=0, N4=1, N5=1, N6=0, N7=0	106.16
N1=F, N2=1, N3=0, N4=1, N5=0, N6=X, N7=X	97.73

Once a simple illustrative example is given, we report the results obtained in benchmark circuits, and then compare the delay variation due to different sensitization vectors with other effects like process parameters or the impact of the interconnect capacitance.

5.3.4.2. Benchmark circuits

To prove the importance of consider the specific sensitization vector during the path delay estimation this section is focused on the delay variation with the input vector for complex gates, and therefore the results are focused on analyzing the delay of the paths having more than one sensitization input vector due to complex gates. We tested the pat identification algorithms developed using the ISCAS and ITC benchmarks circuits.

To generate the results we first determined the paths having more than one sensitization vector. Then the tool generated a script for the commercial tool to explore a set of these paths, and import the report generated [73][74]. With this information, we compared the delay estimation and the input values assigned to the complex gates within each path, to those generated by the developed tool and the electrical simulations. Finally, we computed the percentage of paths for which the commercial tool identified correctly the input vector providing the larger delay. Each path obtained was electrically simulated to verify that it was really a true path and to determine the input vector providing the larger delay.

Table 5.15 shows the results summarizing the ability to identify the input vector inducing the worst-case delay for each path, for both the developed method that considers the sensitization vector and the commercial tool used as an example of algorithm that uses a minimum effort sensitization. The table is organized as follows: the first column identifies the circuit, while the second column provides the CPU time required by the tool to find all true paths, plus the corresponding sensitization vectors and extract the 100 slowest multi-vector paths. The third

column shows the total number of sensitization logic vectors reported by the tool, and the fourth column the number of input vectors reported by the developed tool that sensitize the functional paths considered, i.e. the 100 slowest paths that have more than one sensitization vector (except for the small c17). We call a functional path to a sequence of nodes (i.e. a structural path), with a specific sequence of transitions along the path. Each structural path may have more than one functional path, and each functional path may have more than one sensitization vector. The CPU time used by the commercial tool is given in the fifth column and the sixth column values correspond to the number of sensitization vectors reported by the commercial tool for the true paths considered.

Table 5.15: Sensitization vector impact on critical path identification

		Developed framework			**Commercial tool**			
Circuit		**CPU Time (s)**	**# of Sensitization Logic Vectors**	**# Logic Vectors for 100 slowest multi-vector paths**	**CPU Time (s)**	**LogicVectors**	**Correctly identified Sensitization vectors**	
ISCAS'85	**c17**	0.00	32	24	1.49	25	6	75.00%
	c432	20.93	9864	226	263	498	23	37.10%
	c499	90.50	278346	400	31029	796	0	0.00%
	c880	8.14	215690	3098	1419	389	8	17.02%
	c1355	317.07	45448	200	83477	0	0	0.00%
	c1908	22.49	167988	398	2245	176	1	7.69%
	c2670	67.93	280856	219036	2991	26	0	0.00%
	c3540	413	1184190	2640	245762	0	0	0.00%
	c5315	2563	1401378	6794	3773	0	0	0.00%
	c6288	22566	5790748	231026	47569	0	0	0.00%
	c7552	432	186994	238	2029	1204	43	87.76%
ITC'99	**b14**	32848	4268462	5894	52104	0	0	0.00%
	b15	7980	3139370	8648	25002	247	23	56.10%
							Mean	21.59%

To give a metric of the accuracy to identify correctly the worst sensitization vector, the sensitization vectors are grouped in functional paths (i.e paths with same sequence of nodes and transitions on each node, but with different sensitization vector and propagation delay), and for each functional path considered the sensitization vectors are compared.

Finally, the last two columns of Table 5.15 show the number and percentage respectively of functional paths for which the minimum effort algorithm provides the input vector that produces the

worst delay for that functional path. These results show the inefficiency of not considering the specific sensitization vector during the delay computation highlighting the impact of the delay variation due to the sensitization vector for complex gates. In many cases the commercial tool simply finds the case for which the complex gate input assignations are easier to justify instead of exploring all the possibilities.

The algorithm developed in this work explores all possible input vectors for each path, and therefore it identifies correctly the worst delay for each path. The results in the last column of Table 5.15 show that if the delay variation with the input vector is not considered, the estimation of the worst delay for each path is quite poor, obtaining only a mean value of 21.59% of paths correctly estimated.

5.3.4.3. Relevance and comparison to other effects

The delay variation due to the sensitization vector are compared to the delay variations caused by other effects like process parameter fluctuations or the interconnect system. Such analysis is key to determine the relative significance of this phenomenon compared to other important delay variation sources. We carried this comparison for various combinational ISCAS circuits to estimate the relative impact at the circuit level. Table 5.16 shows the relative delay variations obtained for the c432 ISCAS circuit as an example. The first row shows the delay variation due to the sensitization vector that gets up to 30%.

Table 5.16: Path delay variation

		Delay variation
Sensitization vector		29.98%
Interconnect size (Oversized vs ideal)		19.84%
Parameter variations	**Nominal vs Best Case**	27.29%
	Worst Case vs Nominal	32.70%
	Worst Case vs Best Case	47.72%

To estimate the delay due to the interconnect system, we compared the nominal delay of the ISCAS c432 using a timing simulator that neglected the impact of the interconnect load to another simulation of the same circuit for which the interconnect was estimated assuming a 10X area increase. Such analysis provides and estimation about the impact of the interconnect delay for circuits having long wires. The second row in Table 5.16 shows the relative delay variation between both circuit versions whose difference is mainly due to the interconnect system. Such delay variation is 10% smaller than the delay variation due to the sensitization vector variation. We used

the Synopsys® Design Vision tool to estimate the area of the synthesized circuit and the impact of the additional wiring on delay [75][76].

The last three rows of the table show the process induced delay variation taking the 65nm CMOS commercial technology provided corner values. In the worst-case variation scenario (worst-case corner vs. best-case corner) the delay relative variation gets up to 47%. This represents the highest variation value as it corresponds to the wider fluctuation range being a non-realistic overestimated case. A worst-case vs. nominal-case variation analysis shows a 33% relative variation being of the same order of magnitude that the variation due to the sensitization vector.

The analysis carried in the previous section considered the delay variation impact with the sensitization vector assuming the impact of a single complex gate on a given path. We have shown that, for a path with only one gate that can be sensitized by multiple vectors, a relatively large gate delay variation of about 35% may result in an overall path delay variation of about 10%. Obviously, a given gate-level delay variation will translate to a smaller path-delay variation since the contribution of a single gate is softened when compared to the overall path. This effect will obviously depend on the relative path length, and possibly on the specific position of the complex gate within the path.

We investigated the effect of having more than one complex gate in a given path, and its combined impact on the overall path. We saw that the variation effect of one complex gate can amplify the effect of another complex gate, because variation affects not only the propagation delay, but also the output slew time. As an example a path having two complex gates, the first one with a small variation of about 2.6% has an impact on the next complex gate, whose delay variation increases from 34% to 54%, resulting in an overall path delay variation of 16%. To get an idea of the importance of this effect, we observed that this path exhibits a delay variation of about 2.6% due to a temperature increase from 25 ºC to 60 ºC.

To evaluate the relative impact of this effect at the circuit level, we analyzed how often this situation might occur within a given circuit by analyzing how many of the 1000 slowest paths contained two or more complex gates. We also computed, for the same set of paths, the mean number of complex gates per path. Results in Table 5.17 show that this is a common situation, and that the mean number of complex gates per path is beyond 3 for the ISCAS circuits larger than the c432.

Table 5.17: Complex gates per path

Circuit	>= 2 complex gates	Mean complex gates / path
c432	488	1.49
c499	1000	5.04
c880	1000	4.35
c1355	1000	3.78
c1908	994	3.58
c2670	967	4.44
c3540	1000	7.01
c5315	1000	7.2
c7552	993	5.36

These results demonstrate that the effect of sensitization vector on the path delay cannot be ignored, and has an impact comparable to others side effects.

5.4. Algorithms for Timing Analysis

The following sections show how the multiple sensitization algorithms can be combined with the simplification techniques to perform true path identification on combinational circuits. These results may be used for multiple purposes, since there are many cases where true path identification through a combinational block and the input vectors required to sensitize these paths are required. Some of them are:

- Timing analysis.
- Circuit synthesis optimization.
- Delay fault test pattern generation.
- SET propagation analysis.

5.4.1. Exhaustive path identification

Exhaustive path identification consists in identifying as many true paths as possible together with the input vectors that sensitize these paths. This can be accomplished using any version of the stepwise algorithm detailed in section 3.4.1.

As has been shown in section 5.3.1, the different input vector used to sensitize complex gates lead to significant variations on the propagation delay [52][57]. Therefore the input vector must be considered for accurate path delay estimation, and the exhaustive path identification process tries to identify all possible sensitization vectors for each true path.

It is obvious that an exhaustive path exploration cannot be achieved for very large circuit designs due to the exponential nature of the number of paths through a circuit categorized as a NP-problem [44]. However the high efficiency of the stepwise algorithm developed combined with circuit simplification techniques makes this objective affordable in a reasonable time for circuits with thousands of gates. In any case, a backtrack limit can be imposed as detailed in the description of the algorithms (3.4.1.43.4.1), to avoid excessive runtime when the algorithm is applied to complex circuits. In this case some true paths can be discarded as if they were false paths although they might actually be true paths.

Experimental results show that, in general, the number of true paths discarded due to backtrack

limit represents a small portion of the total number of true paths through the circuit. The total number of true paths and all sensitization vectors for each true path can be found performing an exhaustive path identification without backtrack limit, although this task may require a huge computation time. Then, the results show that in general an increase of the backtrack limit leads to a large increase in the runtime at a small increase in the number of true paths. This indicates that it is possible to run a preliminary analysis limiting the backtrack to small value achieving a fast execution without incurring in a great lost of identified paths.

Table 5.18: Backtrack limit influence for ISCAS c7752

Backtrack limit	**True paths**		**Sensitization vectors**		**Total backtracks**		**Aborted due to backtrack limit**		**CPU Time (s)**	
5	6,094	---	67,117,888	---	271,583,015	---	51,381,250	---	3,452	---
10	6,174	+1.31%	67,157,686	+0.06%	480,834,854	+77.05%	33,385,701	-35.02%	6,269	+81.60%
1,000	6,200	+1.74%	67,272,336	+0.23%	2,215,316,187	+715.7%	1,519,598	-97%	12,042	+248.8%

Table 5.18 shows how the backtrack limit impacts the exhaustive path identification for ISCAS c7552. The table includes the results for three values of the backtrack limit (5, 10 and 1000), showing that the increase in the number of true paths and sensitization vectors are small while the runtime suffers a large increase. As shown in Table 5.18 and as expected the number of abortions due to the backtrack limit decreases considerably when the limit is increased. However, this reduction is not reflected on sensitization vectors identified. Therefore a large number of aborted cases actually lead to a logic conflict.

Typically long paths traverse many logic gates and consequently may have a lot of sensitization vectors some of which may be discarded due to the backtrack limit. Given the sensitization vector impact over the path delay it is interesting to obtain all sensitization vectors for the longest paths since some discarded vectors may lead to the largest delay through the circuit. This can be accomplished by marking the paths when the process is aborted due to the backtrack limit, and then adding a post-processing step to complete the sensitization vectors for the paths with longest delays.

Tables 6.1-6.1 shows the results of the exhaustive path identification algorithm for some benchmark circuits. Results are presented for different versions of the stepwise algorithm: forward, backward and forward version with the justification at the end of the path. The results are also given for the three sensitization criteria detailed in section 3.3.1.

Table 5.19: Forward algorithm. Criterion 1

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	8	32	0	0	0	0
c432	1474	22798	4167397	3580	16	6252
c499	3284	216600	21530322	17876	121	30248
c880	2831	132438	151806	30	2	34680
c1355	3248	358386	20969048	20100	119	47140
c1908	4888	1142162	23638299	15082	140	146768
c2670	1459	70176	2871832	0	40	35112
c3540	8259	628622	18345841	13698	160	171108
c5315	5756	5550518	168642519	137176	617	1852775

Table 5.20: Forward algorithm criterion 2

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	9	26	0	0	0	0
c432	1598	15944	2682140	2365	11	5872
c499	6509	425400	21530322	17876	127	54984
c880	4809	170740	81928	9	1	34676
c1355	3900	425072	22754656	21636	131	54456
c1908	8762	1179648	14021949	7784	89	151088
c2670	4046	445700	417396	0	37	106732
c3540	36331	3074478	79748793	57669	719	810344
c5315	9674	3445656	5843788	4279	130	1157124

Table 5.21: Forward algorithm criterion 3

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	9	26	0	0	0	0
c432	22822	350204	2431014	1275	20	30996
c499	7040	3818340	47328070	29246	475	459920
c880	4939	679842	171140	16	6	80564
c1355	4463	3424658	56014844	39355	436	413076
c1908	18034	8515074	20118919	8398	410	1058456
c2670	4636	21027112	8747115	4342	289	4830208

Table 5.22: Backward algorithm Criterion 1

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	8	32	2	0	0	0
c432	1324	21084	1294490	1084	6	6492
c499	3483	222638	10372717	10545	70	31460
c880	2782	126238	2449518	1696	16	34676
c1355	2021	168014	10077719	9952	65	24852
c1908	4176	377040	18146261	11343	93	51356
c2670	1459	69808	366452	22	7	35108
c3540	3308	59716	7605232	6112	87	38336
c5315	5467	2776434	227693100	195318	1659	961428

Table 5.23: Backward algorithm Criterion 2

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	9	26	1	0	0	0
c432	1430	14254	776283	664	3	5720
c499	6968	443840	21455792	21093	144	57932
c880	4732	160094	2169632	1206	16	34680
c1355	2235	178612	10966046	10836	70	26132
c1908	7726	484794	21967053	12753	114	64408
c2670	4046	439436	3780163	1828	46	106363
c3540	10976	221540	20640998	16427	230	67528
c5315	9464	3047378	45026080	19913	526	1044912

Table 5.24: Backward algorithm Criterion 2

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	9	32	0	0	0	0
c432	29928	521622	24378983	21847	140	4600
c499	8097	3570130	108930052	103995	996	424900
c880	5107	725346	4346727	2757	37	97732
c1355	4752	2836592	92755529	88809	818	340712
c1908	12748	5557564	152069262	107085	1086	688684

Table 5.25: Forward algorithm with justification at end Criterion 1

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	8	32	0	0	0	0
c432	1469	22468	5897394	4849	25	6300
c499	3433	207632	20794752	20584	142	29060
c880	2831	132466	119721	16	2	34676
c1355	2984	251064	88988968	85880	526	34172
c1908	4744	1024352	131029781	80839	663	131544
c2670	1459	70176	3010	0	13	35108
c3540	8173	604858	51921889	42179	506	164976

Table 5.26: Forward algorithm with justification at end Criterion 2

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c17	1595	15746	3723826	3167	16	5932
c432	6866	415456	41589504	41168	297	53248
c499	4809	170672	163247	43	4	34696
c880	3546	286928	110217348	106924	665	38536
c1355	8585	1087636	106880356	60961	656	140264
c1908	4046	445226	2176617	237	31	105992
c2670	35932	2922426	257897648	212724	2465	772724

Table 5.27: Forward algorithm with justification at end Criterion 3

	Paths	Input vectors	Backtracks	Backtrack limited	Time (s)	Memory (kiB)
c432	22826	349880	2960434	1386	24	30788
c499	7040	3720912	112318736	98040	1030	445580
c880	4939	678674	984536	648	17	91028
c1355	4456	3336184	125640128	112296	908	400224
c1908	17963	8355352	125365092	89962	1242	1043408
c2670	4636	21018034	23487967	9634	881	4816500

As outlined in the introduction to timing analysis, ensuring that a design meets the timing constraints imposed by the specifications requires to focus the timing analysis on the critical paths, i.e. the true paths with largest propagation delay. Therefore there is no reason to identify all

sensitizable paths to verify that the timing constraints are meet. However, this information can be used to perform circuit optimizations thanks to the detailed timing information.

In the next sections the exhaustive path identification combined to other techniques to perform the path identification on large circuit designs is shown.

5.4.2. Timing estimation using path graph

This technique uses the path graph structure detailed in section 3.4.3 that represents all the structural paths through a circuit and allows to easily identify the longest structural paths. Once a path has been identified then the full path sensitization algorithm is applied to verify if it is actually a true path. If the path is false, then it is discarded and the process continues with the next structural path. This process repeats until the required number of true paths has been identified.

This strategy avoids performing an exhaustive path identification focusing the effort in the paths with largest propagation delay, allowing to be used with circuits unaffordable for the exhaustive path identification algorithm.

Figure 5.18 shows an example circuit used to illustrate how the critical paths can be found using a path graph. The numbers shown inside the gates represent the propagation delay for each gate input. These numbers are merely for illustration and are replaced by a delay model in a real application for more accurate results.

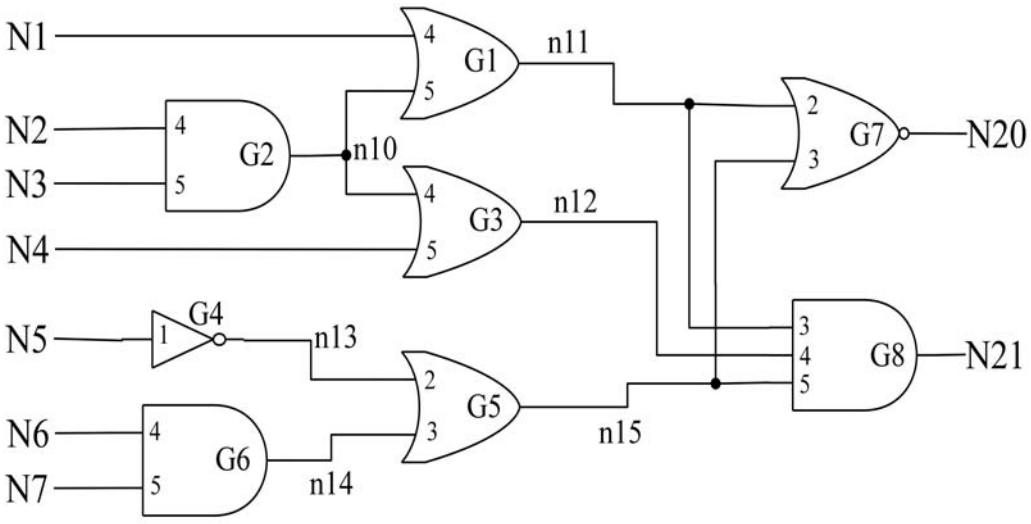


Figure 5.18: Example circuit

Firstly the limitations of using a graph representing directly the circuit structure instead of a path graph will be exposed, since a path graph requires much more memory than a circuit graph, then the reasons to use the first strategy will be shown.

The structural path with the largest delay can be easily identified by simply propagating through the circuit the delay required by a signal to reach each node in the worst case and then tracing the path from the output node with the largest arrival time.

Figure 5.19 shows a graph representing the circuit of Fig. 5.18. The number pairs near to each node are the minimum and the maximum delay to reach this node. For example the values for the node n11 are (4, 10): 4 is the minimum delay to arrive to node n11 from node N1 through input A of gate G1, and the maximum value (10) is the time required through the subpath {N3, n10, n11} providing the largest delay to arrive at node n11.

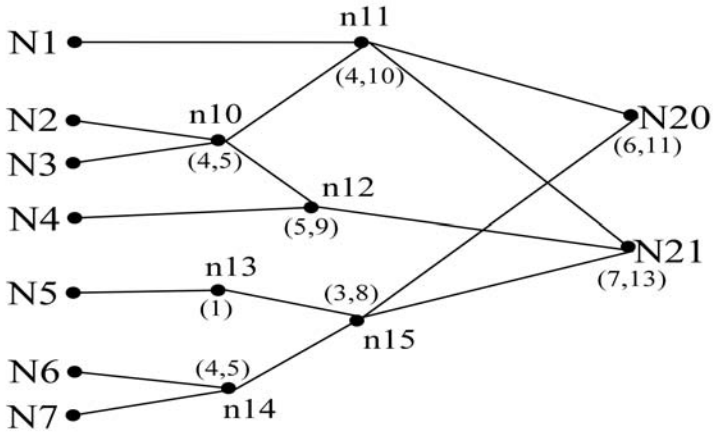


Figure 5.19: Circuit graph

Applying this method it is easy to identify the output node where the largest structural path ends. As shown in Fig. 5.19 the maximum delay for the output nodes N20 and N21 are 12 and 13 respectively, then the slowest path ends at node N21. Tracing back the maximum delays through the graph it can be identified three structural paths having a delay equal to 13 (Since the example uses integer values for gate delays there are multiple path with exactly the same propagation delay).

{N3, n10, n11, N21}

{N3, n10, n12, N21}

{N7, n14, n15, N21}

This method allows to easily identifying the paths with worst delay, however to identify the next

slowest paths (as example the path {N2, n10, n12, N21} has a delay equal to 12) more information than the one presented in the graph is required. Therefore if we are interested in a set of slowest paths the delay information for each graph node must be more complex. Besides if a path results non-sensitizable, in general, the graph cannot be pruned dropping exclusively the false path identified.

The aforementioned problems can be solved using a path graph instead of a circuit graph. As was detailed in section 3.4.3 a path graph is a graph representing each structural path through a circuit. Although the generation of a path graph requires more computational resources than the previous structure it has many advantages.

Figure 5.20 shows the path graph for the circuit in Fig. 5.18. As shown there are various independent graphs for each input node allowing for an independent processing. The numbers in parentheses represent the propagation delay to reach each node. In this case the graph contains the delay for each specific path instead of uniquely the maximum and minimum values.

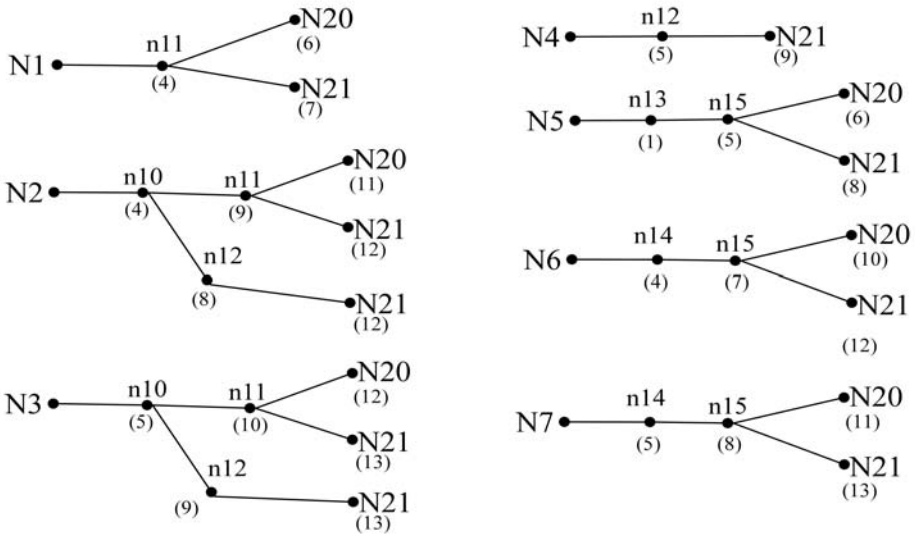


Figure 5.20: Path graph

Using this structure the task of identifying a set of paths with larger delay is immediate. As an example, the graph shows that there are three paths with a delay equal to 13, four paths with a delay of 12 and two paths with a delay equal to 11, and so on. In addition if a subpath is recognized as non-sensitizable the graph can be pruned dropping all paths sharing the non-sensitizable subpath without affecting other paths. Therefore, full path sensitization algorithm can be applied to structural paths beginning with the one with largest delay until the specific number of true paths has been found.

5.4.3. Timing estimation using path graph and simplification techniques

The technique presented in the previous section can be used to identify the critical paths for a circuits too large and complex to be processed using the exhaustive path identification. However, the path graph for very large circuits may require an excessive amount of memory. This limitation can be solved with the help of the simplification techniques detailed in section 4.1.

Figure 5.21 shows the flowchart of the algorithm used for very large circuits combining the path graph, simplification techniques and exhaustive path identification.

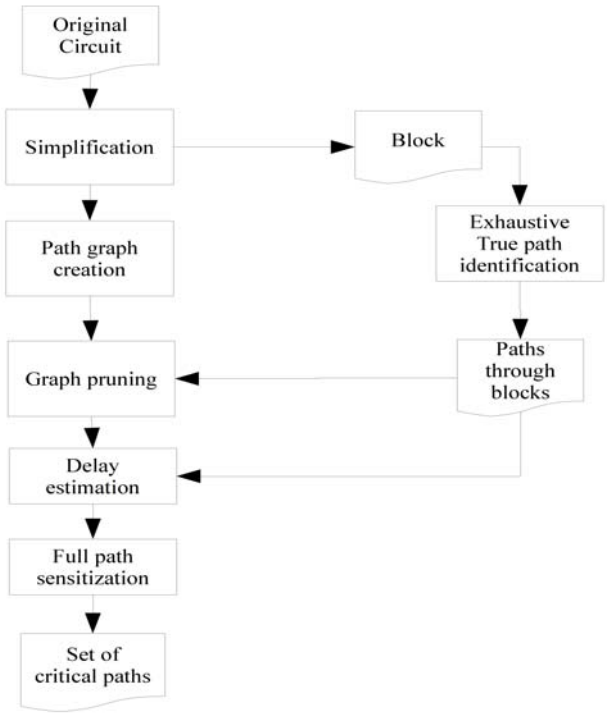


Figure 5.21: Algorithm flowchart

The first algorithm step reduces the circuit by iteratively applying a selection of simplification techniques depending on the specific circuit structure and analysis performed. If the interest relies on a specific set of inputs and/or outputs the circuit can be partitioned before begin simplified by the encapsulation techniques. The outcome of this step is a simplified circuit structure and a set of blocks created by encapsulating portions of the circuit. For huge circuits the simplification techniques creates a hierarchical structure where each block contains other blocks.

Each individual block is small enough to be processed using the exhaustive path identification algorithm. The information about the true paths and sensitization vectors for each block will be used

in next algorithm steps to compute the path delay through the global circuit. To avoid redundant computations in the case of repeated structures the blocks with multiple instances are analyzed only once.

Once the circuit has been simplified it is ready for path graph creation. The simplifications performed allow creating the graph even for very large and complex circuit designs. Once after the graph creation, the information about the paths through each block is used to prune the path graph, since if a block does not have any true paths from a given input to an output then the paths sharing this subpath can be pruned simplifying the graph.

The path graph and the path information for each block are used to compute an estimation of the path delays. The candidates to be critical paths are selected from the graph in the same way as in the previous section. Then the algorithm tries to sensitize these paths using the full path sensitization algorithm combined with the sensitization information generated for each block. This process repeats until a specified number of true paths has found.

Table 5.28 and 5.29 show the results of applying the algorithm depicted in Fig. 5.21 to identify the slowest true paths for large circuit designs. Table 5.28 shows the time required for each preprocessing step, while Table 5.29 shows the time required to identify a given number of slowest true paths.

Table 5.28: Critical path identification (preprocessing)

Circuit	**Preprocessing time (s)**			
	Simplification	**Path graph creation**	**Block processing**	**Total**
c6288	0	3	1435	1438
c7552	1	2	485	488
b14	3	8	931	942
b15	7	8	1046	1061
b17	24	2	1324	1350
b18	158	58	1832	2048
b19	101	37	1763	1901
b20	10	17	536	563
b21	11	17	729	757

Table 5.29: Slowest true path identification

Circuit	Time required to identify true paths (s)		
	25 slowest paths	**100 slowest paths**	**250 slowest paths**
c6288	3146	7618	16325
c7552	221	1246	5218
b14	284	761	1222
b15	546	1846	4158
b17	991	3759	7021
b18	1864	4725	10047
b19	2146	7215	19745
b20	716	2413	7894
b21	512	1346	2548

Chapter 6: Framework application to SET propagation estimation (SENSET)

The general concepts about the propagation of SETs induced by ionizing particle impacts on a combinational logic block were introduced in Chapter 2. A SET propagating through a circuit reaching an output node may be the source of a soft-error. Therefore to guarantee the reliability of a circuit design the susceptibility to propagate an SET cannot be ignored during the design flow, specially for circuits intended to work in a hostile environment with high level of radiation like aerospace applications.

In this chapter the integration of the tools included in the framework with an analytical SET propagation model developed within our research group will be detailed. The objective of this integration is determining the circuit internal nodes and outputs with highest sensitivity to SET propagation, providing a valuable information for circuit designers.

6.1. SET Propagation model

The tool to be developed requires estimating both logic and electrical masking that an SET suffers through a path from the node where the SET is originated until an output node by describing how the pulse traverses each logic gate. While logic filtering does not require any specific model, electrical masking descriptions needs an accurate description. A simple and accurate propagation model was published in [7], and will constitute the core of the tool presented in this work.

The electrical SET propagation model characterizes the SET pulse through two parameters: the pulse width and the pulse height. The pulse height (V_{max}) is the maximum voltage variation with respect to the correct voltage (either GND or V_{DD}), and the width (t_w) is the pulse duration measured at the pulse half height (V_{max} / 2). Fig. 6.1 shows a voltage pulse with the two metrics used to characterize it. This model considers height and width parameters when propagating the pulse, while other models typically don't consider the pulse height assuming that all pulses span the entire supply voltage range [77]. This model is fully analytical and fully continuous, being highly effective for inclusion in CAD environments.

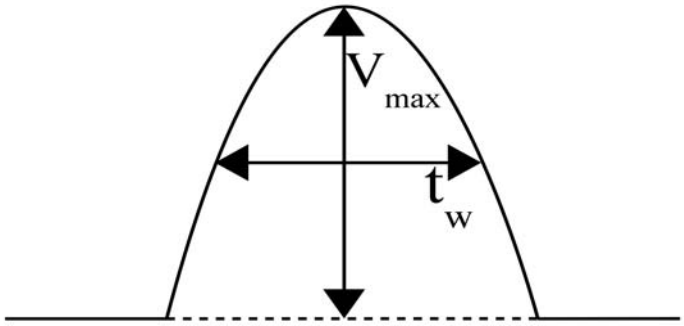


Figure 6.1: SET pulse characteristics

The model behaves like a transfer function, providing the characteristics of the pulse at the logic gate output (i.e. a pair V_{out}, tw_{out}) once the input pulse (V_{in}, tw_{in}) parameters are given. This transfer function is related to the gate type, and the specific gate input through which the pulse is propagated. For each logic gate input, the model consists of two analytic functions, one for each output parameter (V_{out}, tw_{out}). Model equations are reported in (6.1) and (6.2). A detailed explanation of the analytical expressions, and physics details about the SET propagation was presented in [7].

$$V_{out}(V_{in}) = \frac{V_{DD}}{1 + e^{-k\cdot(V_{in} - V_0)}} \tag{6.1}$$

$$tw_{out}(tw_{in}) = a \cdot tw_{in} + b + t_0 \cdot e^{-\frac{tw_{in}}{t_i}} \tag{6.2}$$

Both transfer functions (output height and output width) depend on the two input SET parameters through their coefficients -*k* and V_0 in (6.1), and *a* and *b* in (6.2)- for input pulse width and height respectively. These coefficients depend on the output load capacitance.

As shown in (6.2), the width output pulse equation is a linear expression plus an exponential term while coefficients *a* and *b* have a linear relationship with the input pulse height, and depend on the output load capacitance. The exponential term is negligible except for very narrow pulses, and as demonstrated experimentally [6] this term may be ignored with an acceptable loss of accuracy. These characteristics make the multivariable polynomial model ideal to compute the output pulse width simplifying the model integration into the software tool and the model parameters extraction process. The expression used for SET propagation within the propagation tool is given in (6.3). The polynomial model is also used to determine coefficients (*k* and V_0) of the height transfer function that depend on two variables (input pulse width and output load capacitance). Finally, the model requires 3 coefficient matrices, one 3-dimensional matrix for tw_{out} (*TW*), and two 2-dimensional matrices for the coefficients *k* and V_0.

$$tw_{out}(tw_{in}, V_{in}, C_{Load}) = \sum_i \sum_j \sum_k TW_{ijk} \cdot tw_{in}^i \cdot V_{in}^j \cdot C_{Load}^k \tag{6.3}$$

$$k(tw_{in}, C_{Load}) = \sum_i \sum_j K_{ij} \cdot tw_{in}^i \cdot C_{Load}^j$$

$$V_0(tw_{in}, C_{Load}) = \sum_i \sum_j V_{0ij} \cdot tw_{in}^i \cdot C_{Load}^j \tag{6.4}$$

6.1.1. Model parameters extraction

Three coefficient matrices parameters are extracted for each gate type in the technology library, the input through which the pulse passes, and the pulse polarity (rising or falling).

All coefficients are extracted from electrical-level simulations, following a process similar to the one developed for the delay model. An automatic process simulates the SET propagation through each library gate input. Fig. 6.2 shows the circuit schematic used to determine each input model coefficients for each logic gate. An iterative process varies the output capacitance and the input

pulse height and width. Results consist of two 3-dimensional matrices, one for the output height and one for the output width, with the values for each combination of the input pulse height, width and output load.

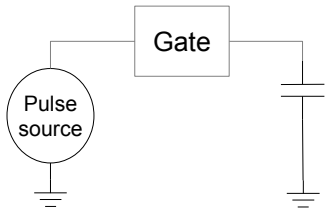


Figure 6.2: SET model extraction circuit

The specific pulse waveform applied to the logic gate input influences the output pulse, becoming a source of inaccuracy for the overall estimation if the input pulse waveform is not realistic. The pulse induced by an ionizing particle impact is usually described through a double exponential waveform, where the rising transition has a quick slope and the falling one is softer. However, such a waveform appears only at the impact node; when it is propagated through logic gates the waveform generated at each gate output presents, in general, an almost symmetric waveform. This is because standard cells are typically designed to have similar output slew time for both rising and falling transitions. A realistic input pulse that prevents an excessive output voltage waveform overshoot as induced by an ideal pulse is obtained by an input source generating a waveform based on sinusoidal transitions (6.5). This waveform is a modification of the one presented in section S'ha produït un error: No s'ha trobat la font de referència used to extract the delay model parameters.

Although the waveform generated by this function has a shape different to the waveform induced by a particle impact, its shape is very similar to the waveform of a SET after passing through a gate. Therefore, the SET propagation estimation may suffer a larger inaccuracy in the first gate traversed, due to the difference in the input pulse shape. However, as the SET traverses multiple gates, this drawback only affects the first gate traversed, and may be solved by using two extracted model coefficients sets using two different types of input waveforms. A first set would be used for the pulse shape induced by a particle impact, and the second set for the pulse after crossing one gate. However, the improvement in estimation accuracy does not compensate the complexity increase, the simulation runtime required to extract the coefficients, and the memory increase required to store an additional set of model coefficients. It must be also considered that the induced pulse shape varies depending on the type of particle impacting the circuit. Then, problem solving may require

using a variety of pulse shapes to describe various kind of particles, increasing even more the complexity. Thus, we consider a single pulse waveform for all cases.

$$V(t)=\begin{cases} 0 & : \quad t \le t_0 \\ \frac{A}{2}\cdot\left(\sin\left(\omega\cdot(t-t_0)-\frac{\pi}{2}\right)+1\right) & : \ t_0 \le t \le t_1 \\ V_{DD} & : \ t_2 \le t \le t_3 \\ \frac{A}{2}\cdot\left(\sin\left(\omega\cdot(t-t_2)+\frac{\pi}{2}\right)+1\right) & : \ t_2 \le t \le t_3 \\ 0 & : \quad t_3 \le t \end{cases} \tag{6.5}$$

Where A is the maximum pulse amplitude, t_0 is the initial delay time before the pulse starts, and t_1, t_2, t_3 and ω determine the pulse width and the rising and falling slopes. This expression provides a simple control of the transition time, pulse width and height, being a requisite for the model extraction process.

6.2. SET Propagation Sensitivity

The SET propagation sensitivity (SPS) is a metric defined for a circuit subpath – a subpath being a path starting at an internal node, and ending at a circuit output – that accounts for the combination of the subpath logic and electrical masking effects. The SPS can be used to estimate the likelihood that an SET induced at a given circuit node propagates within the circuit until reaching a memory element that may result in a soft-error. The SPS does not consider the probability that an ionizing particle impact generates or not an SET, only the likelihood that an SET will be propagated, once it has been generated. This section describes how the SET propagation model outlined in Section 6.1 is integrated into the software tool, how the SPS is computed and how it is used to identify the internal nodes and circuit outputs with higher sensitivity to propagate or register an SET respectively.

Since an SET that does not reach a circuit output will not cause an SEE, the sensitivity analysis is performed starting at the output nodes and going backward inside the circuit. Given that SET *time masking* is not related to the SET propagation within the circuit (it only affects its capture by a memory element), then only *logic* and *electrical masking* are considered.

For *logic masking*, the number of gates in a subpath determines how many nodes must be set to a specific logic value to allow the SET propagation. Therefore, the longer the path length, the higher the logic requirements for the SET to be propagated, implying that the probability of a specific subpath activation is inversely proportional to the number of logic conditions to be satisfied. It may even happen that some subpaths are non-sensitizable (i.e. their logic probability is zero) due to logic incompatibilities between the in-path gate sensitization requirements. Therefore, in general, *logic masking* increases with the length of the subpath that the SET must cross until reaching an output.

Electrical masking description is more complex. Typically a pulse whose duration is shorter than the gate delay gets filtered, otherwise the pulse is propagated maintaining its width approximately equal, as long as the height of the pulse is sufficiently large. If a logic gate has different rising and falling transition times, then the pulse may experience a broadening effect. However, in general, library standard cells have balanced transition times, and a significant number of gates must be crossed by an SET to experience broadening. Broadening may also happen when an SET enters a reconvergent path and there is a specific relationship between the reconvergent path delays. Therefore, with some exceptions, it can be considered that the electrical masking mechanism

increases with the path length, because a longer path implies a higher probability that some gate filters the SET.

The combination of logic and electrical masking mechanisms provides the SPS that can be used to determine the subpaths with higher probability to propagate an SET until an output node. In general these higher-probability paths are short subpaths.

The quantification of each subpath sensitivity ending at an output node within a combinational circuit to propagate an SET is performed in two steps. The first algorithm step determines the logic sensitivity by computing the sensitization probability based on how many input vectors can logically sensitize each subpath. The second algorithm step computes the electrical sensitivity of each subpath, starting by the paths with higher logic probability. This second step uses the SET propagation model to compute relevant electrical propagation information. This covers determining the minimum characteristics of an SET at the internal node for it to arrive to an output node with a specified width and/or height, or determining the electrical characteristics of an SET reaching an output node for a given SET specified at an internal node.

6.2.1. Logic sensitivity

The logic sensitivity determination is a key initial step since if a given subpath cannot be logically sensitized it will never propagate an SET independently of the electrical characteristics of the perturbation and the path logic gates. Logic masking acts as filter eliminating the paths that cannot be sensitized, and thus reducing the number of paths to be included in the electrical sensitivity analysis.

As stated earlier, since an SET can only cause a soft-error if it reaches an output node, the most relevant subpaths are typically short subpaths, and therefore logic and electrical sensitivity computation starts at the output nodes, moving toward the internal nodes. Furthermore when a subpath is non-sensitizable, or its logic sensitization probability is very low, all longer paths passing through this subpath may be discarded, making unnecessary moving deeper into the circuit, and reducing the number of SET propagation computations to perform.

Thus, the logic sensitivity computation algorithm works using the stepwise sensitization algorithm applied in reverse order (3.4.1), i.e., starting at the output nodes and moving toward inside the combinational block. In this case the stepwise sensitization algorithm applied uses the breadth-first search (BFS) strategy since highest sensitivity subpaths are typically short subpaths.

BFS is more efficient for this task as it processes first the current node neighbor nodes, instead of going deeper until the end of the branch before backtracking, as does the depth-first search (DFS).

In this way the analysis concentrates on the best candidate subpaths, ensuring that in general the subpaths with higher logic sensitivity are identified, even if a maximum of subpaths to be processed is imposed to avoid excessive runtime.

For clarity, the algorithm will be detailed using the example circuit of Fig. 6.3 without loose of generality. Assume that the circuit in Fig. 6.3 is a part of a larger circuit with multiple outputs. Since the sensitivity analysis is focused on the outputs, the tool simplifies the circuit, leaning on the circuit partitioning technique, and takes only the part related to the output being analyzed. To illustrate the method a graph representing of the circuit structure starting at the output node is shown in Fig. 6.4. Thus, the algorithm traverses the circuit in the following manner:

1. Start at output node Z.
2. Node *a*. Compute sensitivity for the path {*a, Z*}.
3. Node *b*. Compute sensitivity for the path {*b, Z*}.
4. Node *c*. Compute sensitivity for the path {*c, a, Z*}.
5. Node *g*. Compute sensitivity for the path {*g, a, Z*}.
6. Node *d*. Compute sensitivity for the path {*d, b, Z*}.
7. And so on.

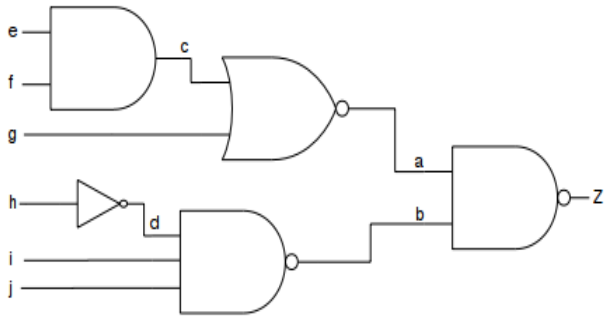


Figure 6.3: Example circuit

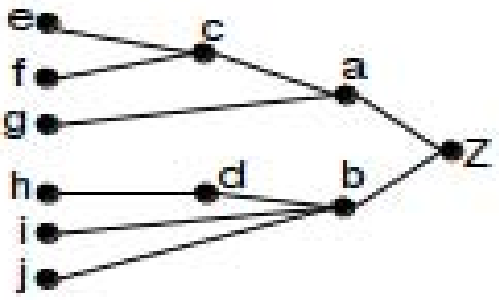


Figure 6.4: Graph of example circuit

At each step the logic sensitivity is computed incrementally, i.e., to compute the sensitivity of the path $\{c, a, Z\}$, the algorithm uses the previously computed sensitivity for the path $\{a, Z\}$. This approach improves the algorithm efficiency reusing information instead of sensitizing the entire subpath each time.

To determine the subpath logic sensitivity the algorithm first sensitizes the gates that must be crossed by applying non-controlling logic values to the side inputs. Then, the logic values assigned are justified towards the input nodes. If a logic incompatibility is detected, then the subpath under analysis and all subpaths passing through it are discarded. If the subpath can be sensitized, its logic probability is determined based on the number of logic conditions that must be fulfilled to guarantee its sensitization.

Even if the subpath can be sensitized, when the logic sensitivity of a path is below a given threshold, the algorithm discards this branch considering that all paths hanging from the current node have even worse sensitivity, and are negligible. This technique helps to avoid computing all possible subpaths through a circuit, focusing the analysis on those subpaths with higher probabilities to cause an error.

The logic sensitivity of a subpath is determined accounting for the number of logic conditions that must be fulfilled to allow the subpath to be sensitized. For this quantification, it is assumed that each possible input logic vector has exactly the same probability to occur, i.e., each primary input may have a logic value 0 or 1 with a 50% probability. This assumption is an approximation, since for a specific task when in field operation not all input vectors have the same probability, and even some input values combinations may never appear during a real circuit operation. To allow a more realistic estimation of the activation probability, the information generated by the sensitization algorithm can be combined with the information about the probabilities of the input vectors supplied by another tool.

All logically rated subpaths are sorted according to their sensitization probability. Given the random nature of an ionizing particle impact on a circuit, the subpaths with higher probability of being activated are considered first in the electrical sensitivity computation.

6.2.2. Electrical sensitivity

Once the logic sensitivity is used to get a graded path list, the electrical sensitivity is computed for each subpath in the list. In this phase the tool determines the SET electrical characteristics to allow a proper propagation under different conditions. The electrical sensitivity is computed in two directions: the forward direction determines the SET characteristics (V_{in}, tw_{in}) at the subpath initial node required to meet a specific output SET characteristics (V_{out}, tw_{out}), while the backward direction computes the output SET characteristics for a given initial SET.

In the forward case the tool computes the minimum width and height pulse capable of propagating from the subpath initial node reaching the output node with a width and height surpassing the minimums required to be captured.

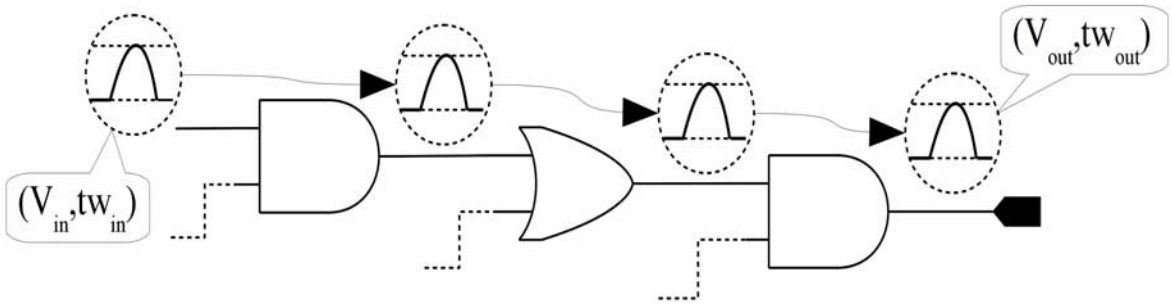


Figure 6.5: Electrical propagation

The process to determine the minimum initial SET starts by applying a pulse with maximum height (V_{DD}) with a sufficiently large width to be propagated by any library gate. The SET is circulated through each subpath using the SET propagation model until an output is reached. Then, the pulse is iteratively narrowed until the SET perturbation does not reach the output or the width and height at the output node are below the thresholds imposed. The value obtained corresponds to tw_{min}, and is used as a measure of the subpath electrical sensitivity to an SET. The minimum electrical characteristics required at an output node is a configurable parameter that, in general, depends on the memory element characteristics.

Subsequently, a V_{DD} pulse having wide-enough duration is propagated, and its height is iteratively reduced until it doesn't propagate toward the output. The minimum pulse height arriving to an output is called V_{min}.

The tw_{min} and V_{min} values provide a quantification of each subpath electrical sensitivity to propagate an SET once the output SET characteristics are determined (mainly fixed by the latch

element connected at the circuit output). This metric gives an insight about the minimum induced SET required at the initial node to produce an output SET capable of being captured by the memory element. Such a minimum can be compared with the typical SET pulses expected for that technology once the final application environment is known and the variety of expected ionizing radiation is estimated. The tool information is key to determine the intrinsic SET filtering circuit capabilities.

As explained earlier determining the minimum characteristics of the initial SET requires an iterative process until the output threshold is not fulfilled. On the other hand, computing the output SET width and height for given initial SET requires only a single computation that propagates the SET along the subpath. However, the output SET characteristics can be further detailed by injecting multiple initial SETs. This metric allows determining the output SET characteristics for a given induced SET.

6.2.3. Circuit level SET sensitivity metrics

Depending on the design stage, the SPS can be used as a metric to rank the circuit internal nodes, or a metric to categorize circuit output nodes. Nodes with the smallest minimum SET pulses will be more prone to propagate any perturbation toward the circuit output and should be the designer focus when hardening the circuit block. SPS can be also used to determine the circuit output being more susceptible to produce an SET, this information is valuable when using a given circuit as a design element that cannot be internally modified. The SPS metric is therefore used to compute either the SET Node Sensitivity (SNS), or the SET Output Sensitivity (SOS).

Node SET Sensitivity (NSS)

The NSS of each internal node to propagate an SET is computed by setting the same SET width threshold for all circuit outputs, or *Output Width Threshold*, and determining for each circuit subpath the minimum SET pulse at the subpath input producing such an output width threshold at the subpath end (note that all subpaths considered end at a circuit output). For each circuit node, the minimum SET pulse for all subpaths starting at such node leading to the output with threshold are averaged considering exclusively the electrical sensitivity. To include the logic probability in the metric the minimum SET values are first weighted by their logic sensitivity, and then averaged. NSS is a useful metric for block-level design as it provides a relative metric of SET propagation for all the nodes within the circuit.

We refer to the specific node sensitivity metrics as *Node SET Sensitivity–Electrical* (NSS-E) when is considered exclusively the electrical sensitivity, and *Node SET Sensitivity–Electrical & Logica*l (NSS-EL) if the electrical sensitivity is combined with the logic activation probability.

Note that the NSS metrics for a given node are computed using the sensitivity information of all subpaths starting at the considered node independently of the output node where the subpaths ends.

The NSS-E is computed as the mean of the minimum SET characteristics of each subpath (6.6) and (6.7). This metric considers only the sensitizable subpaths but grants the same weight to each subpath independently of its logic activation probability.

$$NS-E_{width}(node)=\frac{\sum_{\forall\, p_i\in Paths(node)} t_{w_{min}}(p_i)}{n_{paths}(node)} \tag{6.6}$$

$$NS-E_{height}(node)=\frac{\sum_{\forall\, p_i\in Paths(node)} V_{min}(p_i)}{n_{paths}(node)} \tag{6.7}$$

Where *Paths(node)* is the set of paths starting at *node* and n_{paths}*(node)* is the number of subpaths emanating from *node*.

This metric gives an insight of the ability of a node to propagate an SET until an output, since if a node NSS-E is high this indicates that, in average, such node requires a wide pulse for proper propagation. Otherwise a node with a small NSS-E implies that almost any SET induced at this node will be capable of reaching an output.

To combine the electric and logic sensitivities, the NSS-EL is computed as the sum of the logic probability of all subpaths capable of propagating a given SET normalized to the sum of all subpaths logic probability even if they unable to electrically propagate the SET (6.8). Such normalization is required to avoid values beyond 100% as it is possible to sensitize more than one subpath with each input vector.

$$NS-EL(node)=\frac{\sum_{\forall\, p_i\in E(node)} P_{Logic}(p_i)}{\sum_{\forall\, p_i\in Paths(node)} P_{Logic}(p_i)} \tag{6.8}$$

Where, *E(node)* is the set of all paths from *node* capable of electrically propagating a given SET until an output node, and *Paths(node)* is the set of paths starting at *node*.

***Output SET sensitivity* (OSS)**

The output SET sensitivity is obtained by injecting the same SET pulse width at each circuit node and determining how many of such SETs arrive at each circuit output with a pulse width larger than the output width threshold. These values are weighted by the logic probability of each path. The OSS provides a metric indicating which circuit outputs are more likely to produce an SET.

The OSS is computed by accounting for the SPS of the subpaths ending at each output independently of the initial node. The OSS can be computed either considering only the electrical sensitivity or including the logic probability of each subpath obtaining two OSS metrics: the Electrical Sensitivity (OSS-E) and the Electrical and Logical Sensitivity (OSS-EL).

The *Electrical Sensitivity* (OSS-E) is computed as the ratio between the number of subpaths capable of propagating the injected SET pulse until the corresponding output node, and the total number of subpaths reaching such output (6.9).

$$OS-E(Z)=\frac{|Paths_{Electric}(Z)|}{|Paths(Z)|} \tag{6.9}$$

Where $Paths_{Electric}(Z)$ is the set of paths capable to electrically propagate an SET until the output Z, and $Paths(Z)$ is the set of all paths ending at the output Z.

The OSS-E does not consider the logic probability except for the fact that non-sensitizable subpaths are discarded; the sensitizable subpaths are weighted independently of their logic probability. Therefore, the OSS-E metric provides an insight about the number of subpaths through which an SET may reach a circuit output, independently of their probability of being logically activated.

The *Electrical and Logic Sensitivity* (OSS-EL), considers the logic probability of a subpath to be activated by weighting each subpath by its logic probability and normalizing the result to avoid OSS-EL values beyond 100% (6.10). This possibility is specially potential for subpaths being a portion of longer subpaths, for example: each input vector sensitizing a subpath of three gates $\{G_1, G_2, G_3\}$ also sensitizes the two gates subpath $\{G_2, G_3\}$ and the single gate subpath $\{G_3\}$.

$$OS-ELS(out)=\frac{\sum_{\forall p_i \in Paths_{Elec}(out)} P_{Logic}(p_i)}{\sum_{\forall p_i \in Paths(out)} P_{Logic}(p_i)} \quad (6.10)$$

Where p_i is a subpath, $Paths_{Elec}(out)$ is the set of paths ending at *out* through which an SET is electrically propagated, and *Paths(out)* is the set of all paths ending up at output *out* independently of their electrical propagation.

SET Output width distribution

The SET output width distribution provides the probability of each pulse width at the output node for the same SET injected at each circuit internal node. The distribution is constructed creating a histogram of each subpath output width ending at a given output node, weighted by the logic probabilities.

The information provided by this metric is of interest when some kind of SET filtering technique is applied at the circuit outputs. The metric value provides an indication of the SET width probability expected at each circuit output node for a given induced SET.

Depending on the specific design flow task of a circuit the metric that results more interesting may be different. During the design of the combinational block the interest may lie on identify which nodes have the greater sensitivity to propagate an SET with the objective to redesign some areas of the block to reduce its sensitivity. On the other hand, if the block is already designed and must be included in a larger design, the more interesting metric is the output sensitivity, due to the fact that maybe the block cannot be redesigned, and the design of the system must be focused on the behavior of the output nodes of each block included.

6.3. SET through reconvergence

Fig. 6.6 shows an example of a real reconvergent structure extracted from an ISCAS benchmark circuit. There are two reconvergences in this block, all paths starting at inputs B and D converge to the output node Z.

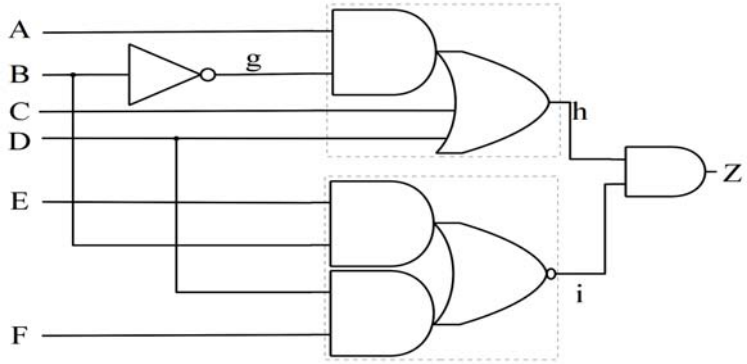


Figure 6.6: Reconvergence

If an SET is induced at one of the reconvergent inputs and the other inputs have the appropriate logic values, then the SET can be propagated through two different paths arriving at both inputs of the output logic gate. The relative difference between pulse arrival time at these inputs depends on the propagation delay of each path.

Fig. 6.7 shows a timing diagram for an induced positive SET at the input B, assuming that the logic values at the inputs are: A = 1, B = 0, C = 0, D = 0, E = 1, F = X. Under this conditions the induced pulse is propagated through both paths {B, g, h} and {B, i} reaching the output gate.

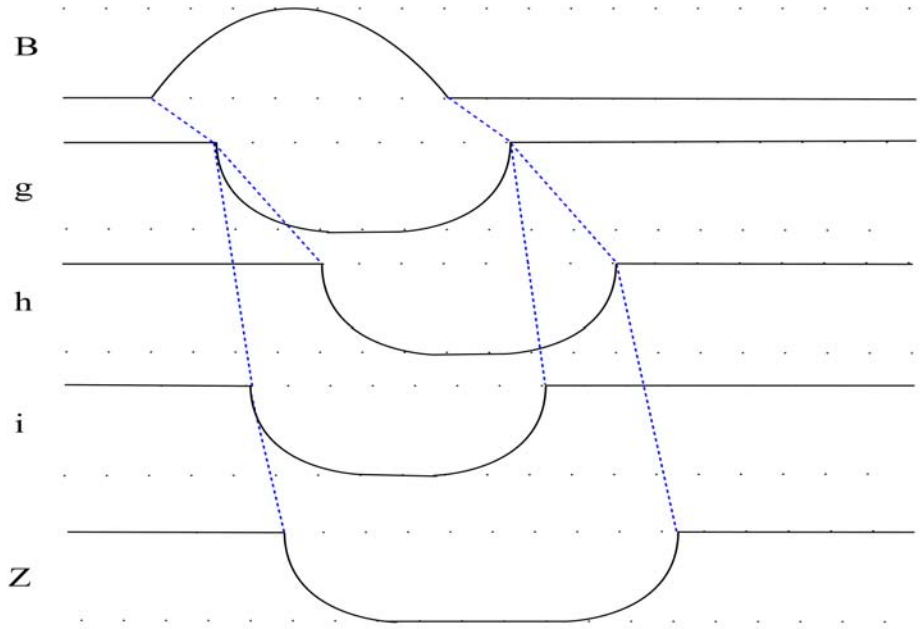


Figure 6.7: SET propagation through reconvergence

As shown in the timing representation the Path {B, g, h} has a larger delay than {B, I}, and therefore the pulse arrives at node *i* earlier than at node *h*. Both pulses overlap generating a wider SET at the output *Z*. In this case both SETs arrive at the last gate with the same polarity, however if one pulse is inverted then the effect is a narrowing of the SET. The pulse may even be completely filtered, since a reconvergent structures is one of the techniques used as filters for SET tolerant circuits.

Considering these situations increases considerably the complexity of the algorithms for SET propagation. The framework developed identifies reconvergent structures and uses the encapsulation technique to hide the reconvergence to the higher-level algorithm. In this way, the main algorithm does not handle such complexity.

The reconvergent block is analyzed as a smaller circuit. If the number of gates within the reconvergent block is small, the SET propagation is analyzed automatically through electrical simulations. In this case, the electrical-level extractor is responsible to describe the possible pulse broadening or filtering. However, if the recovergent block is large, electrical-level characterization is not feasible, and the SET is propagated through the reconvergent paths separately determining the parameters of each reconvergent SET. Then the delay through each path is estimated, and the resulting SET is constructed according to the relative delay between the reconvergent paths.

In this way the impact of path reconvergence on the SET broadening or filtering is described in detail either through electrical-level characterization (very short reconvergent paths), or through the delay model. In any of the cases, the recovergent structure is hidden within a block for the main algorithm.

6.4. Results

We verified SENSET operation on various combinational benchmark circuits including six ISCAS-85 benchmark circuits (c432, c499, c880a, c1355, c1908 and c2670) and four ITC'99 circuits (b14, b15, b17, b18) synthesized using a 65nm CMOS commercial technology. Three additional ISCAS-85 circuits (c3540g, c5315g and c7552g) were synthesized using the 45nm Nangate Open Cell Library from Si2 [78].

6.4.1. Tool Accuracy

For each benchmark circuit we initially verified the correct tool operation by comparing their results to electric-level simulations. The tool automatically generated scripts to simulate a set of subpaths with the electrical simulator and compare the analytical model behavior against the electrical simulation for each subpath. The simulation injected a pulse at the subpath start node and measured the height and width at each node until an output node was found or the pulse disappeared (was filtered out). Recursively electrical simulations were performed to determine the values of V_{min} and tw_{min} as detailed previously. Then, the tool imported the simulation results and compared the minimum SET values with those obtained from the analytical model to determine the method accuracy. To guarantee that the comparison between the model estimation and the electrical simulation result took into account all kind of paths, the subpaths tested were divided into two sets. The first set was composed by paths with higher logic sensitivity, while the second set included randomly selected paths, excluding those paths already included in the first set.

Tables 3.3 and 4.9 compare the results between the electrical simulations and the analytical model. Table I includes the results from the set of the most sensitizable subpaths, while Table 4.9 shows the results for the set of randomly selected subpaths. The first column shows the circuit name. The mean error in the estimation for both voltage and timing parameters appears in columns 2 and 3, while columns 4 and 5 contain the maximum error. The column tagged "Correct prediction" gives the percentage of subpaths for which the model and the simulation match to predict if an SET reaches the output or not.

Table 6.1: Most sensitizable paths

Circuit	Mean error		Max. error		Correct prediction
	V_{min}(V)	tw_{min}(ps)	V_{min}(V)	tw_{min}(ps)	
c432	0.022	12.46	0.300	85.00	95.6%
c499	0.019	7.20	0.200	15.00	98.4%
c880	0.026	7.34	0.200	43.00	94.8%
c1355	0.034	10.30	0.100	26.00	96.0%
c1908	0.020	7.99	0.200	16.00	99.0%
c2670	0.017	11.03	0.100	25.00	98.0%
c3540g	0.008	3.00	0.100	28.00	93.2%
c5315g	0.018	6.10	0.100	21.00	95.6%
c7552g	0.005	2.80	0.100	12.00	99.2%
b14	0.023	9.55	0.100	31.00	93.0%
b15	0.012	14.74	0.200	25.00	89.0%
b17	0.018	10.91	0.100	38.00	90.0%
b18	0.022	16.85	0.100	28.00	87.0%

Table 6.2: Random selected paths

Circuit	Mean error		Max. error		Correct prediction
	V_{min}(V)	tw_{min}(ps)	V_{min}(V)	tw_{min}(ps)	
c432	0.033	6.36	0.200	68.00	95.6%
c499	0.021	2.02	0.200	28.00	94.0%
c880	0.015	4.56	0.100	45.00	95.6%
c1355	0.013	3.91	0.100	25.00	89.0%
c1908	0.019	5.08	0.100	39.00	91.0%
c2670	0.012	13.16	0.100	37.00	99.0%
c3540g	0.007	2.70	0.100	20.00	93.6%
c5315g	0.013	3.90	0.100	27.00	90.4%
c7552g	0.008	2.80	0.100	27.00	88.4%
b14	0.022	14.16	0.100	44.00	94.0%
b15	0.020	15.96	0.100	43.00	92.0%
b17	0.025	14.32	0.200	31.00	97.0%
b18	0.015	18.36	0.100	48.00	86.0%

As shown in the Tables, the model correctly predicts if an SET can reach the output in more than 90% in the vast majority of the cases studied. Mean error results show that the estimation of the minimum pulse capable of reaching an output are quite accurate, both in height and width. Therefore Tables 6.1 and 4.9, provide an insight about the accuracy of the propagation model to estimate the minimum electrical characteristics of an SET to reach an output node. Such an accuracy is key to validate the results of metric estimations performed in the remaining sections of this chapter.

6.4.2. SENSET Analysis

Once the tool accuracy has been verified for a wide set of benchmark circuits, we show the SET sensitivity analysis carried over various benchmark circuits to illustrate the variety of analysis that can be performed using such tool.

6.4.2.1. Output sensitivity

Figs. 6.8-6.13 report the OSS for three ISCAS circuits (c3540, c5315 and c7552) when injecting a 100ps (Figs. 6.8, 6.10 and 6.12 respectively) and a 150ps (Figs. 6.9, 6.11 and 6.13) SET pulse at each internal circuit node. The graph *x*-axis lists the circuits outputs node numbers, showing two values per output corresponding to the *Electrical* (OSS-E) and *Electrical & Logic* (OSS-EL) sensitivity.

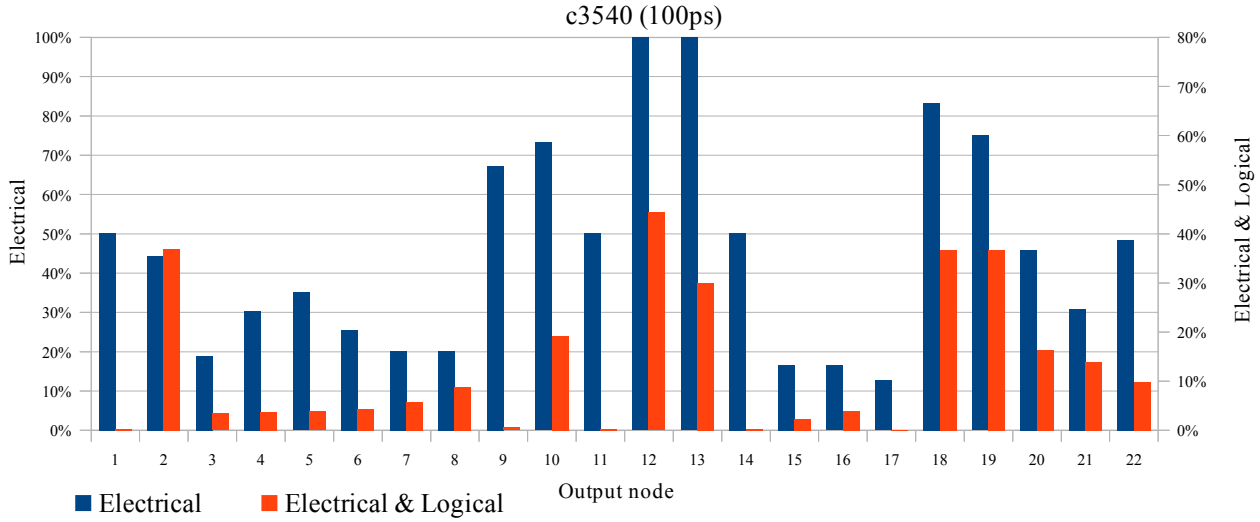


Figure 6.8: Output sensitivity c3540 (100ps)

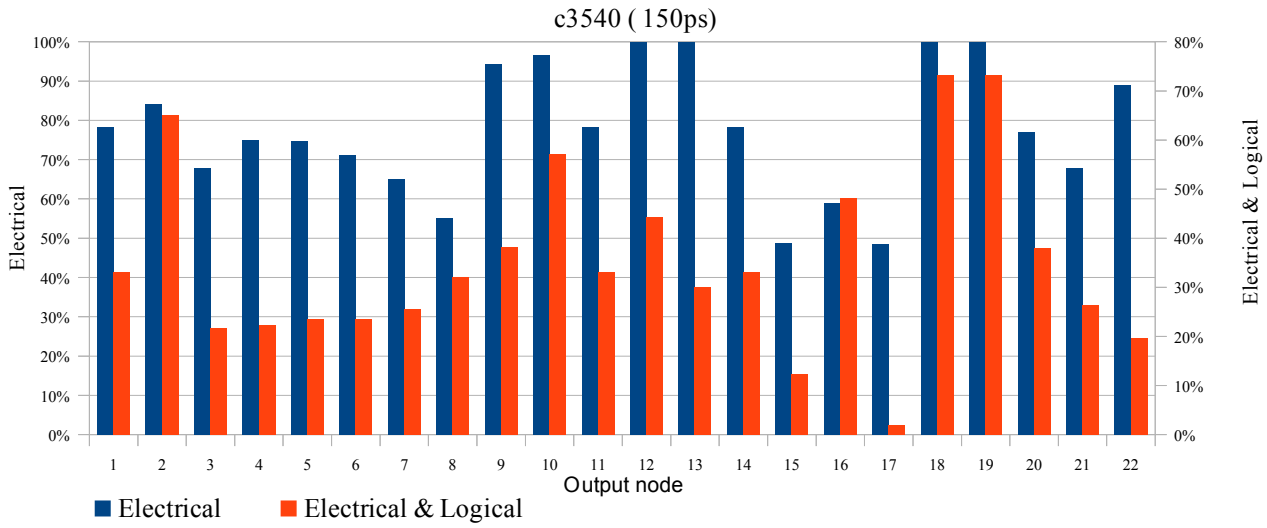


Figure 6.9: Output sensitivity c3540 (150ps)

Both OSS-E and OSS-EL output sensitivity metrics provide important SET propagation information. As an example, output #9 of the c3540 circuit (Fig. 6.8) has a quite large electrical sensitivity to propagate a 100ps pulse, but there is a quite small set of logic vectors that sensitize such path, therefore its although OSS-E is high, the OSS-EL is one of the lowest. This indicates that output #9 has a large number of subpaths capable of propagating a 100ps SET from electrical point of view, but these subpaths have very low probability of being logically activated. An example of the opposite behavior is output #2, whose OSS-E value lies within the mean of all OSS-E values, but its OSS-EL takes the second highest value.

Comparing the results for an injected 100ps (Fig. 6.8) and 150ps (Fig. 6.9) pulse, it is observed that the OSS-E is clearly higher for wider pulse in all cases (as would be expected) since the electrical masking is reduced with the pulse width. As example outputs #15, #16 and #17 have a very low OSS-E for a 100ps injected SET; however for a 150ps their OSS-E is quite larger. The OSS-EL also increases with a wider SET since larger number of subpaths are capable of propagating the injected SET.

Outputs #12 and #13 show similar results for both pulse width because all subpaths ending at these outputs are capable of propagate a 100ps pulse and hence a wider one.

There are cases where the OSS-EL shows a larger increase than the OSS-E like output #11, because some subpaths with higher logic probability are unable to propagate a 100ps pulse but propagates a 150ps SET.

Results for circuit c5315 (Figs. 6.10 and 6.11) shows a similar behavior than the previous circuit. However for this circuit there are a set of output nodes for which the pulse injected are unable of being propagated.

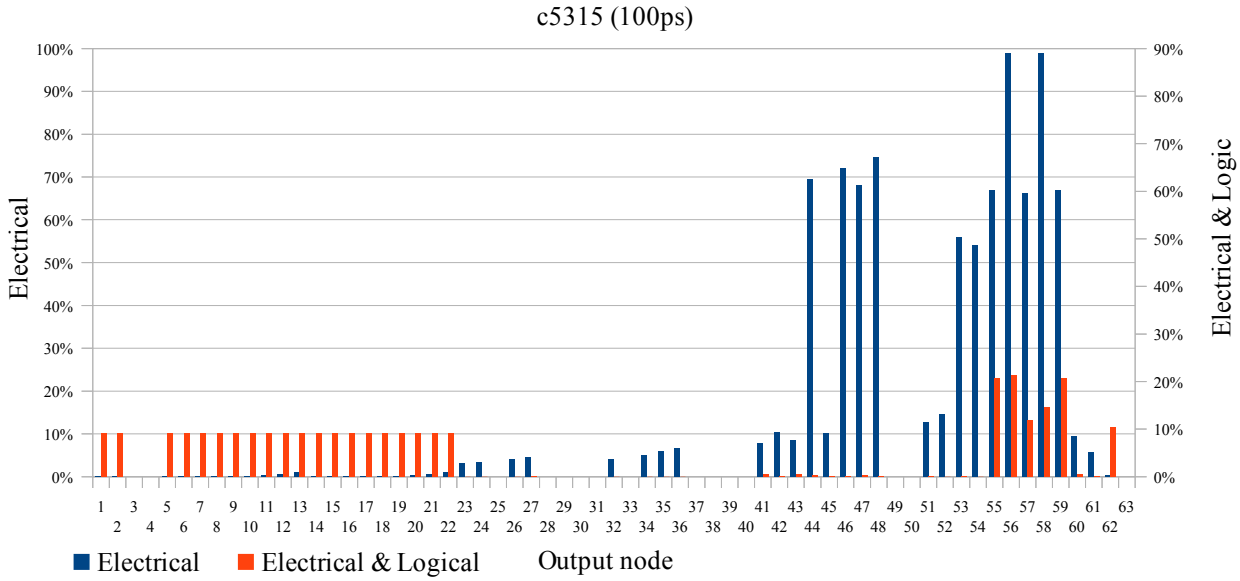


Figure 6.10: Output sensitivity c5315 (100ps)

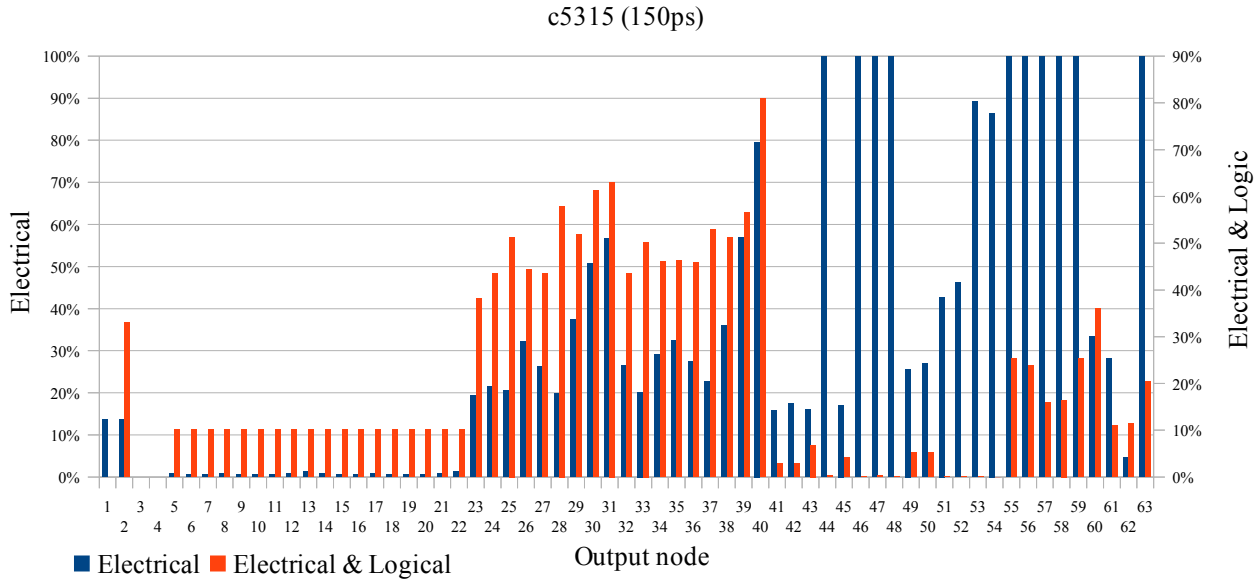


Figure 6.11: Output sensitivity c5315 (150ps)

The behavior obtained for circuit c3540 is also observed in circuit c7552 that has a higher number of circuit outputs. Note that some c7552 outputs (from #47 to #50 in Fig. 6.12) have an OSS-E beyond 70%, but their OSS-EL is almost 0%. On the other hand, output #32 has an OSS-E

slightly beyond 50%, while its OSS-EL is beyond 60% indicating its high probability of producing an SET when considering both electrical and logic filtering.

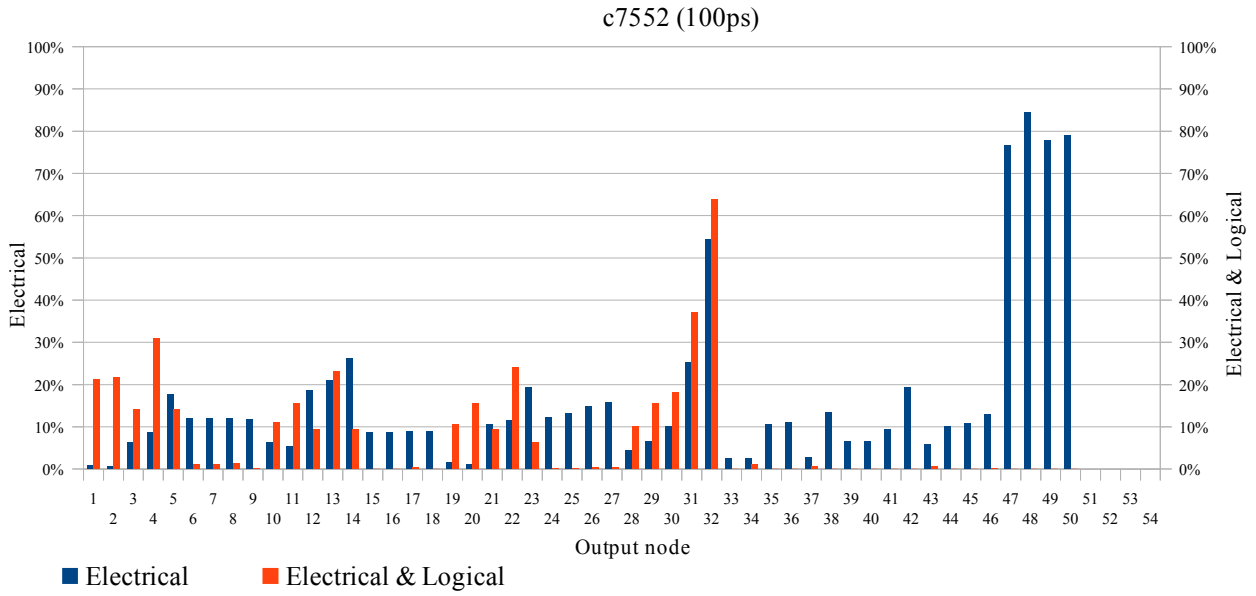


Figure 6.12: Output sensitivity c7552 (100ps)

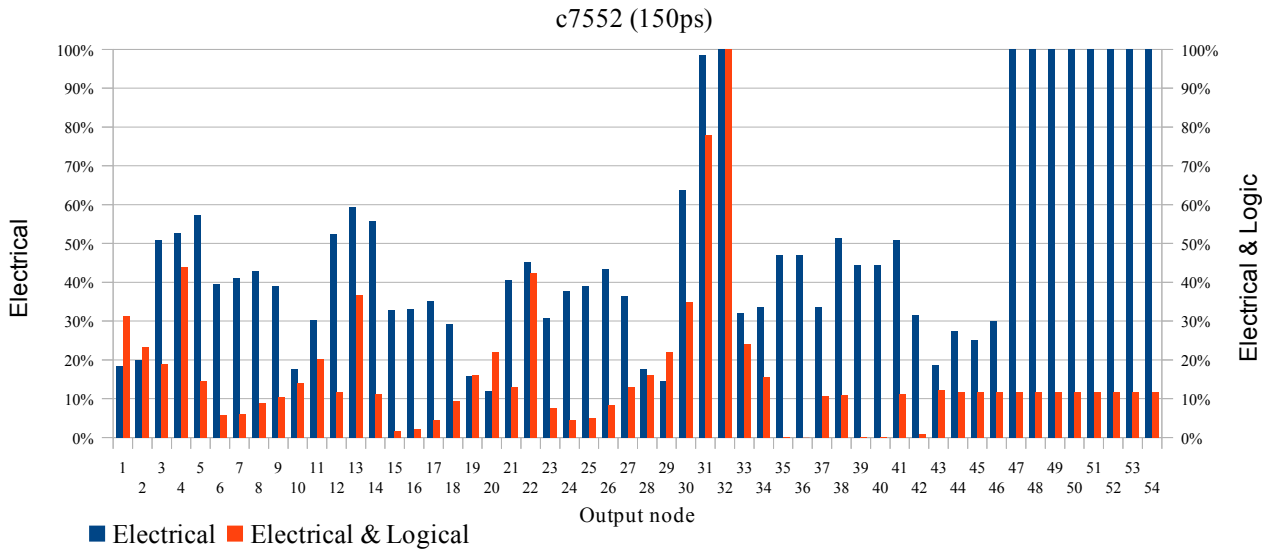


Figure 6.13: Output sensitivity c7552 (150ps)

Comparison of Figs. 6.12 and 6.13 shows that 100ps pulses are not propagated to output nodes #51 to #54, however 150ps pulses are propagated through all subpaths ending at these outputs, as shows Fig. 6.13. This behavior is due to the subpath last logic gate (whose output constitutes the output node) that filters a 100ps pulse but not the 150ps one.

6.4.2.2. Node SET sensitivity

To report the internal node sensitivity we set a threshold for the SET electrical characteristics at the output nodes, considered the minimum SET that would be captured by memory elements. We chose an approximated value for illustration purposes of the method developed. A specific analysis would require a detailed analysis of the memory elements in the technology library. This section results have been obtained by setting a voltage threshold of V = 0.9V and t_w = 80ps, for a technology with a supply voltage of V_{DD} = 1.1 V. The threshold width is approximately twice the delay of a library inverter.

The results are reported for three benchmark circuits (c3540, c5315 and c7552) and since the number of internal nodes is very large the graphs only shows the fifty internal nodes with a larger number of subpaths reaching an output node.

With these conditions Figs. 6.14 to 6.16 show the mean minimum width computed according to (5.7) for the each benchmark circuit respectively. As shown in the graphs, the mean minimum pulse widths are located approximately around the output width threshold value, because in general the subpaths propagate the SET keeping its width. However, when the mean value is below 80ps it means that the subpaths for this node in general broaden the SET pulses. On the other hand, the nodes with large mean value are dominated by subpaths that narrow the pulses.

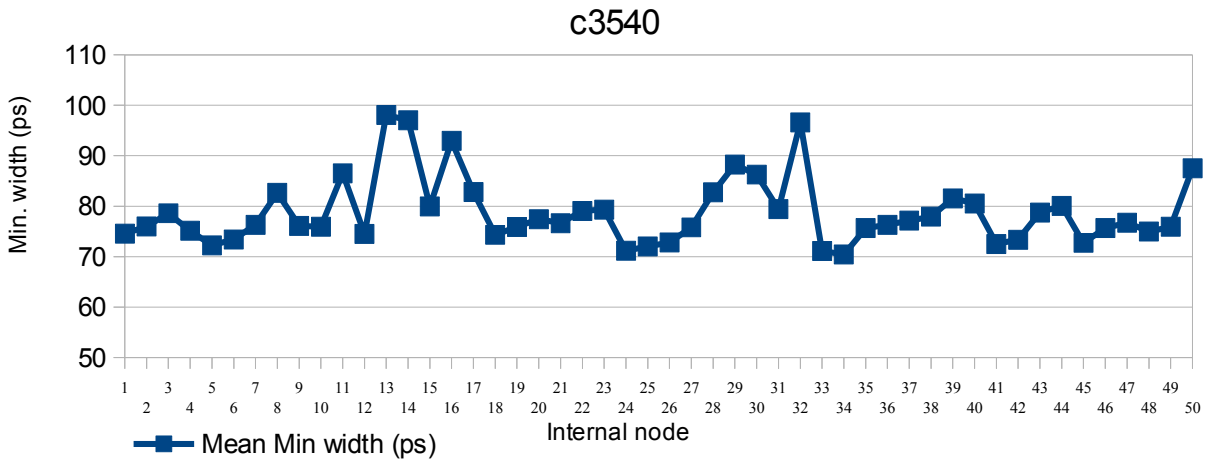


Figure 6.14: c3540 Minimum pulse width

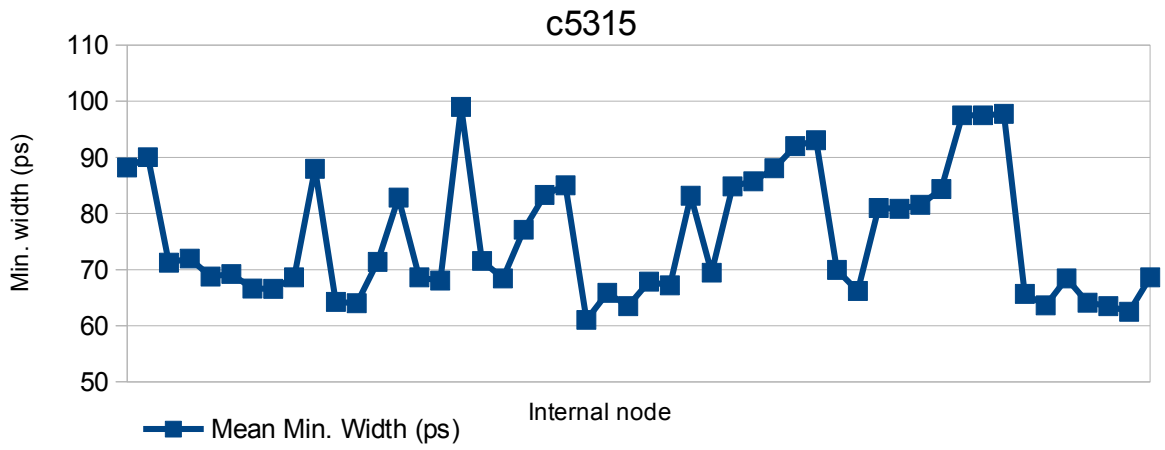


Figure 6.15: c5315 Minimum width

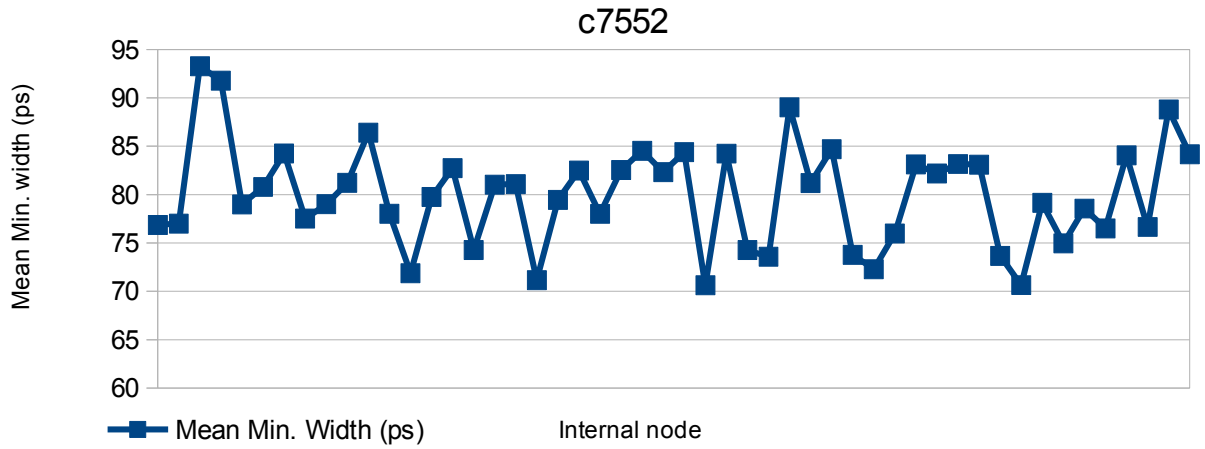


Figure 6.16: c7552 Minimum width

Figs. 6.17 to 6.19 show the average value of the minimum height required to reach an output surpassing the output threshold for an injected SET having a 150ps width. These values have a maximum at the supply voltage, since the case of an injected SET surpassing this value has not been considered.

This case is quite different than the previous one since if the pulse is sufficiently wide and the height is above the next logic gate threshold voltage, then logic gate propagating the pulse raises the outputs to the supply voltage.

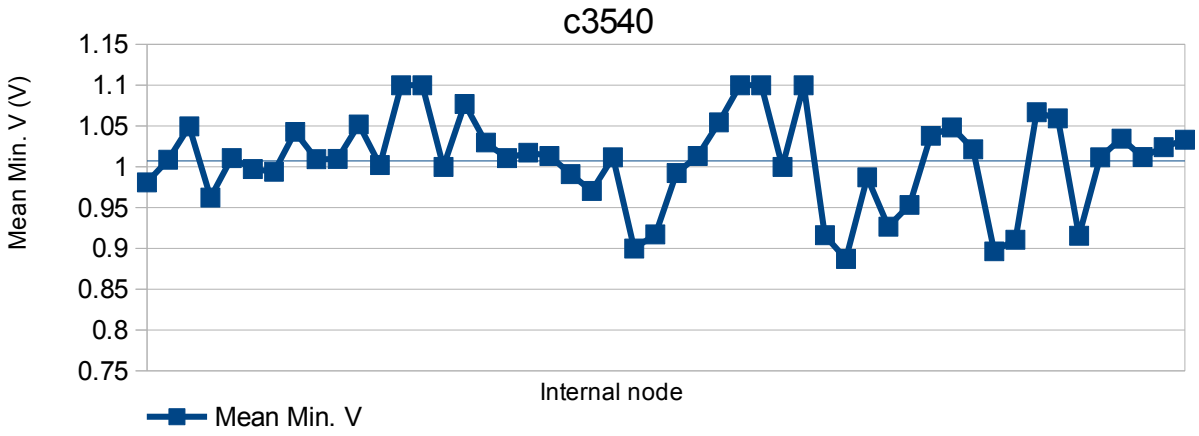


Figure 6.17: c3540 Minimum pulse height

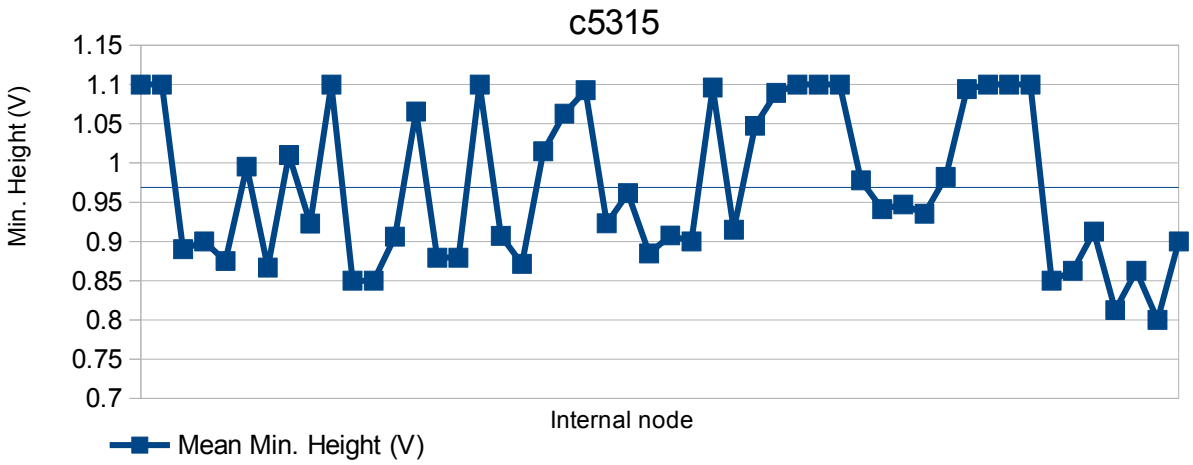


Figure 6.18: c5315 Minimum height

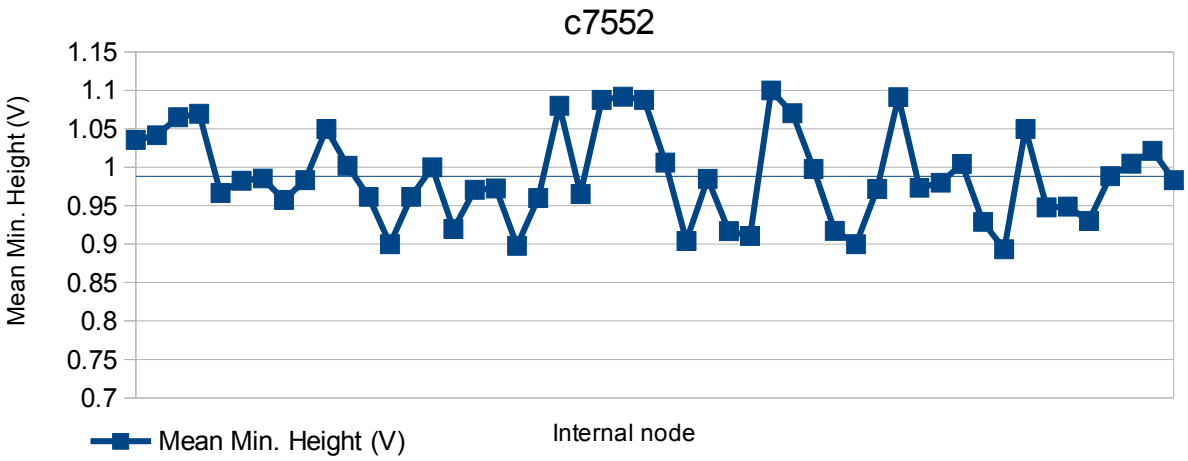


Figure 6.19: c7552 Minimum Height

All previous results have been obtained considering only electrical parameters. Fig. 6.20 to 6.22 show the results considering the logic sensitivity by using the expression (6.8). The internal nodes included in these results are the same for the electrical sensitivity metrics.

These results show that some internal nodes exhibit a high NES that gets weighted down when considering both electrical and logic SET filtering. This information is valuable when designing or synthesizing a specific logic block.

For instance, nodes #47 - #49 of circuit c3540 have a small minimum width value as shown in Fig. 6.14, however when the logic sensitivity in also considered its combined sensitivity falls to low values as shows Fig. 6.20. Similar situations are observed for the other circuits.

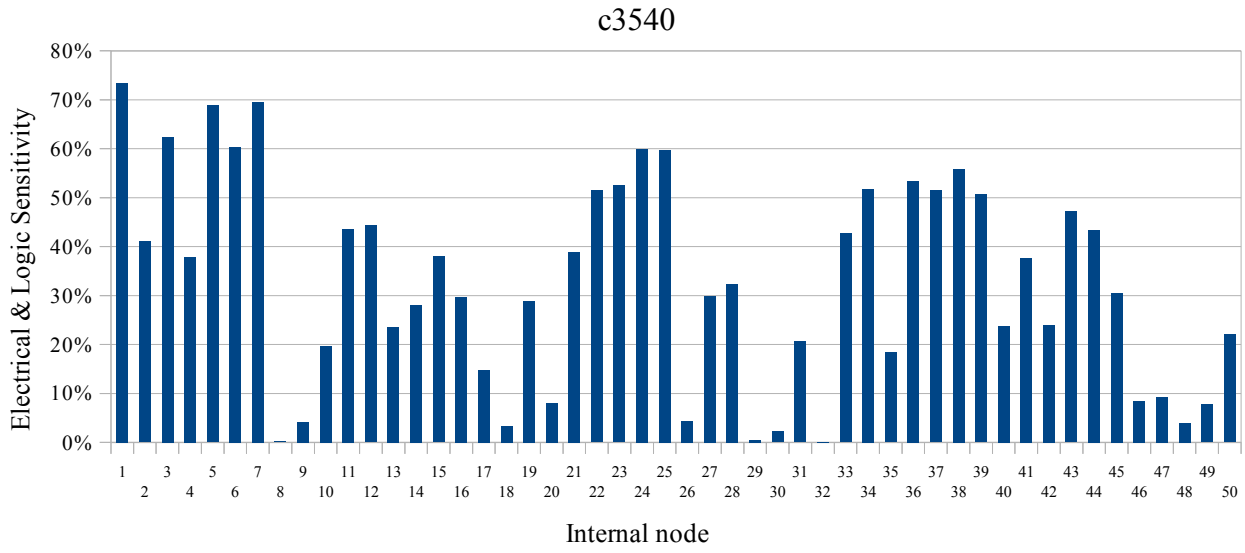


Figure 6.20: c3540 Internal node Electrical & Logic sensitivity

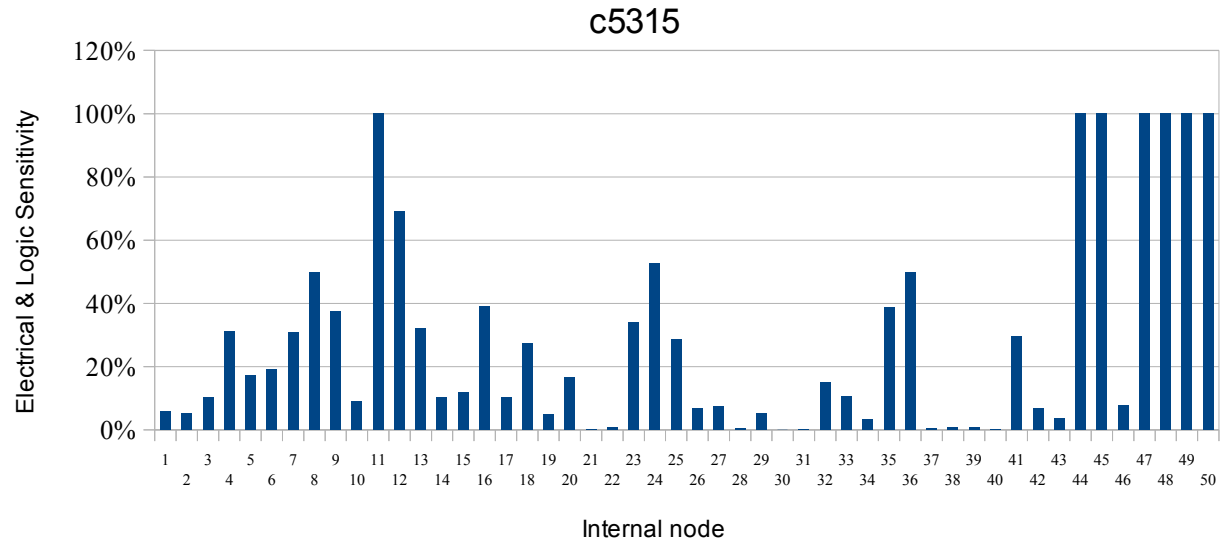


Figure 6.21: c5315 Internal node Electrical & Logic sensitivity

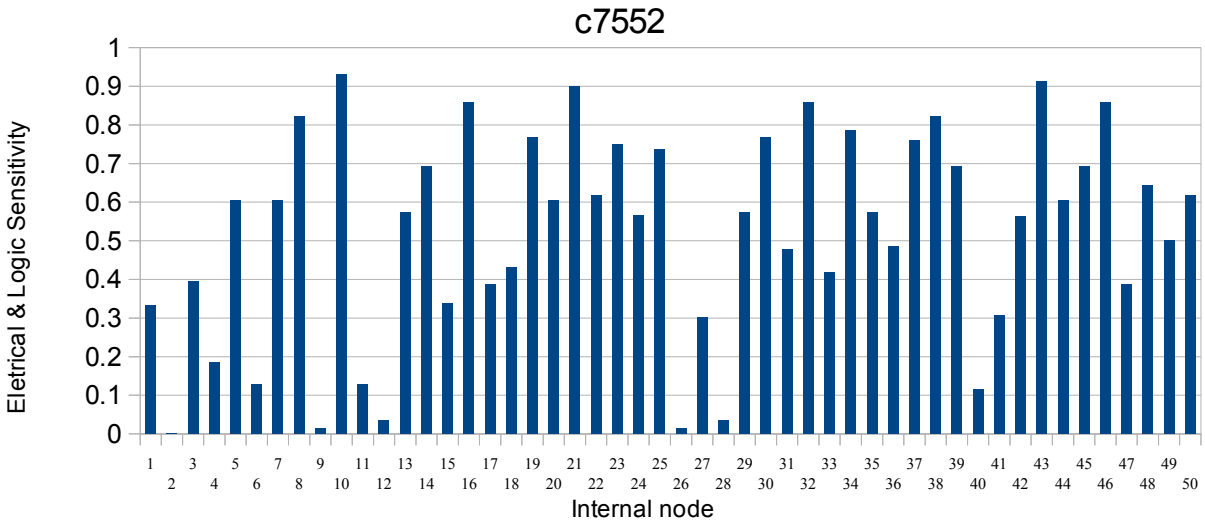


Figure 6.22: c7552 Internal node Electrical & Logic Sensitivity

6.4.2.3. Output width distribution

Figs. 6.23 and 6.24 shows the output width distribution for the benchmark circuit c3540 when a pulse of 100ps (Fig. 6.23) and 150ps (Fig. 6.24) is exhaustively injected at each circuit internal node. Each color represents an output node, the x axis is the SET width reaching the output node and the y axis is the probability computed as the sum of the logic probabilities of each subpath propagating the injected SET until output node arriving with a width inside a given range.

As shown in the Figs, in general the pulse output width with higher probability is located around the width value of the injected pulse. As shown for this circuit only few cases reach an output with a pulse width lower than injected pulse width for the 100ps case. However when the injected pulse is of 150ps, the probability that a narrow pulse reaches an output is higher. This is because 100ps SET pulse is completely filtered before arrives to the output node for many subpaths, while a 150ps pulse is capable of traversing much more subpaths.

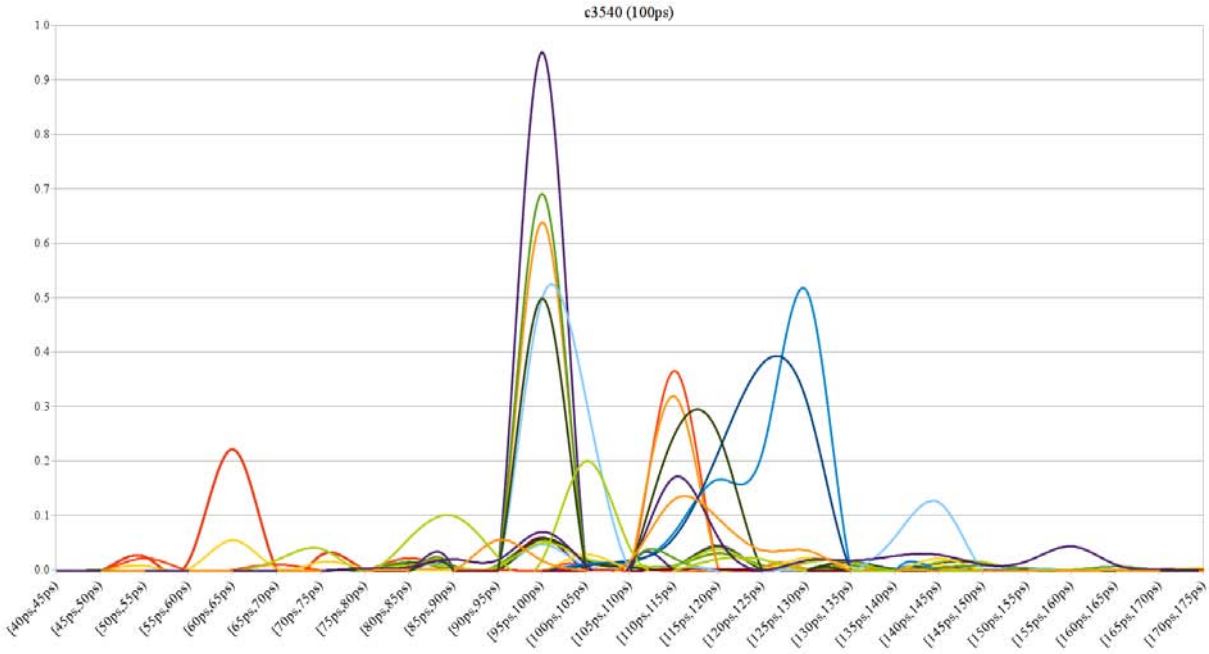


Figure 6.23: c3540 Output width distribution (100ps)

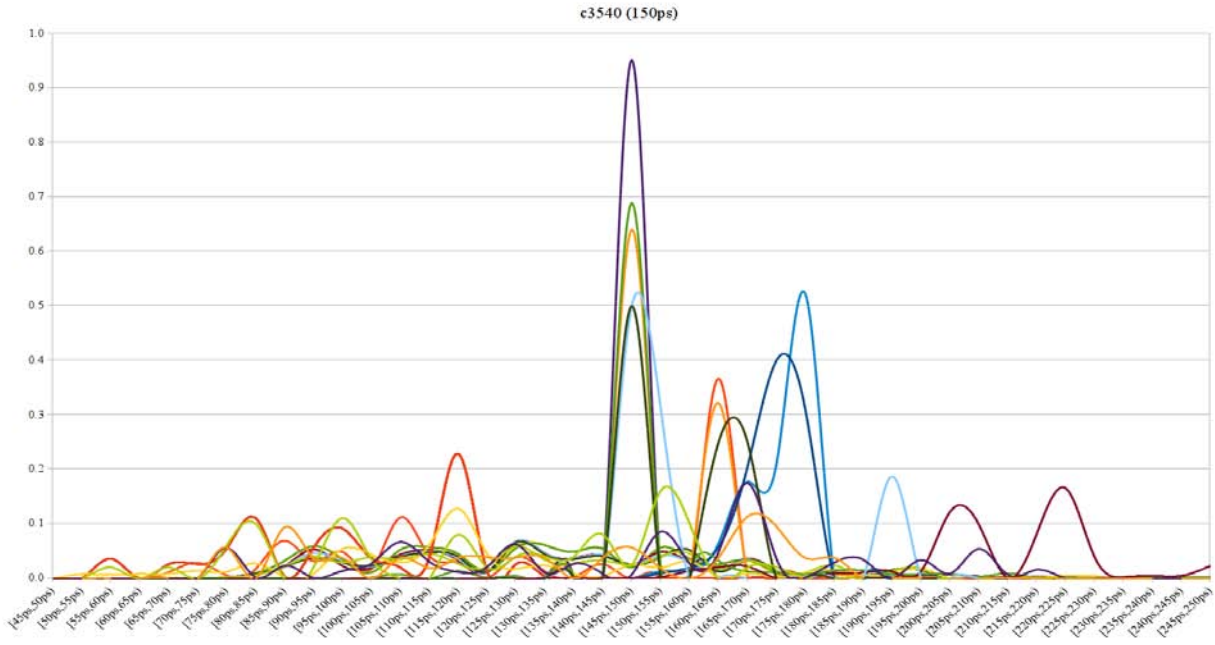


Figure 6.24: c3540 Output width distribution (150ps)

Figs. 6.25 to 6.28 show the output width distribution for the benchmark circuits c5315 (Figs. 6.25 and 6.26) and c7552 (Figs. 6.27 and 6.28) when injecting a pulse of 100ps and 150ps. The Figs. show a similar behavior than the ones of the circuit c3540.

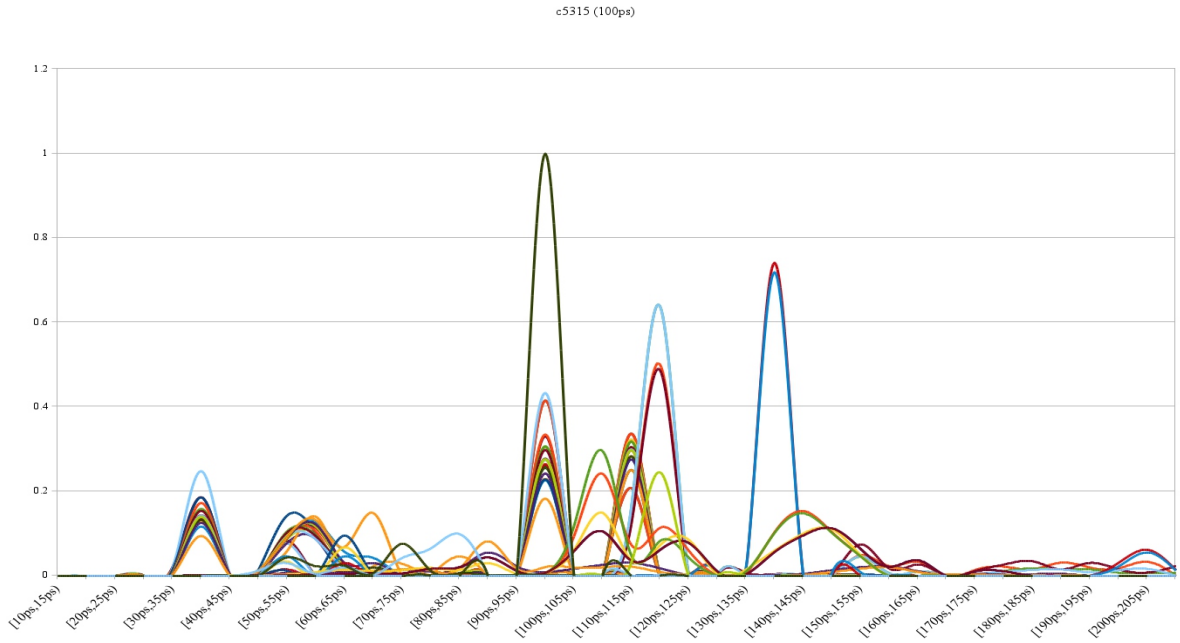


Figure 6.25: c5315 output width distribution (100ps)

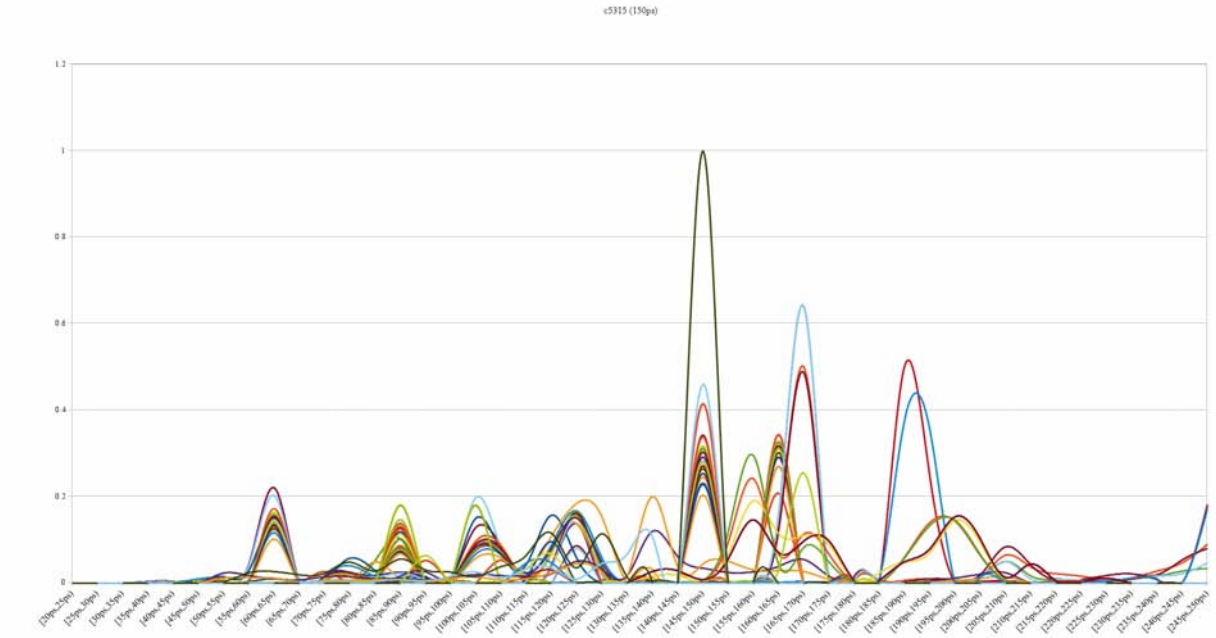


Figure 6.26: c5315 Output width distribution (150ps)

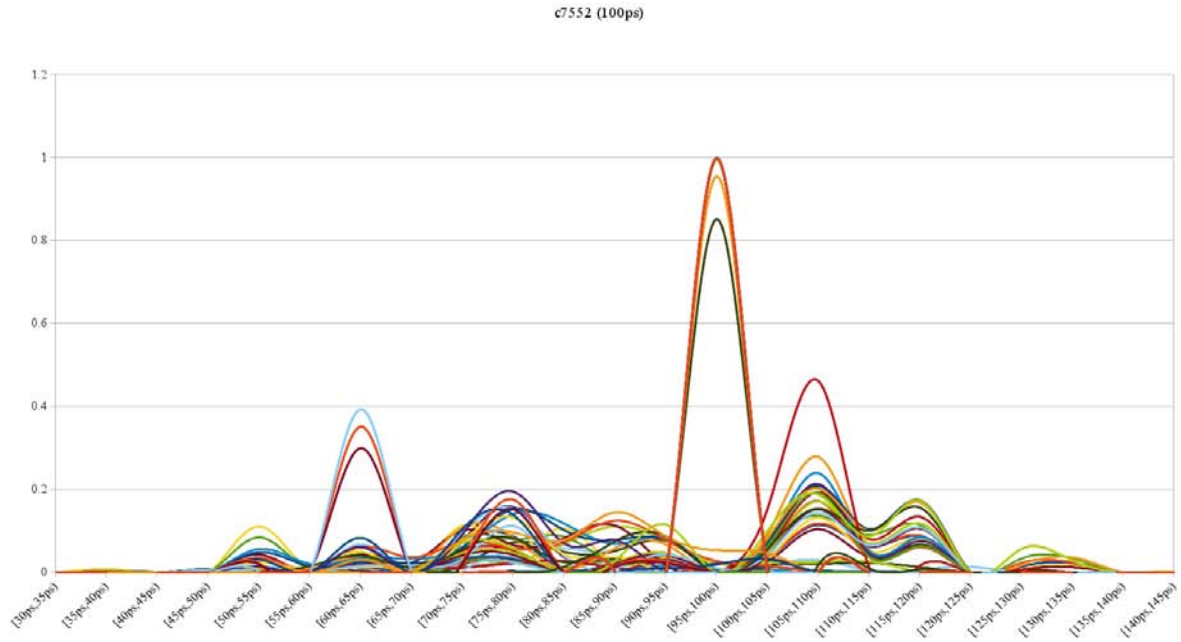


Figure 6.27: c7552 Output width distribution (100ps)

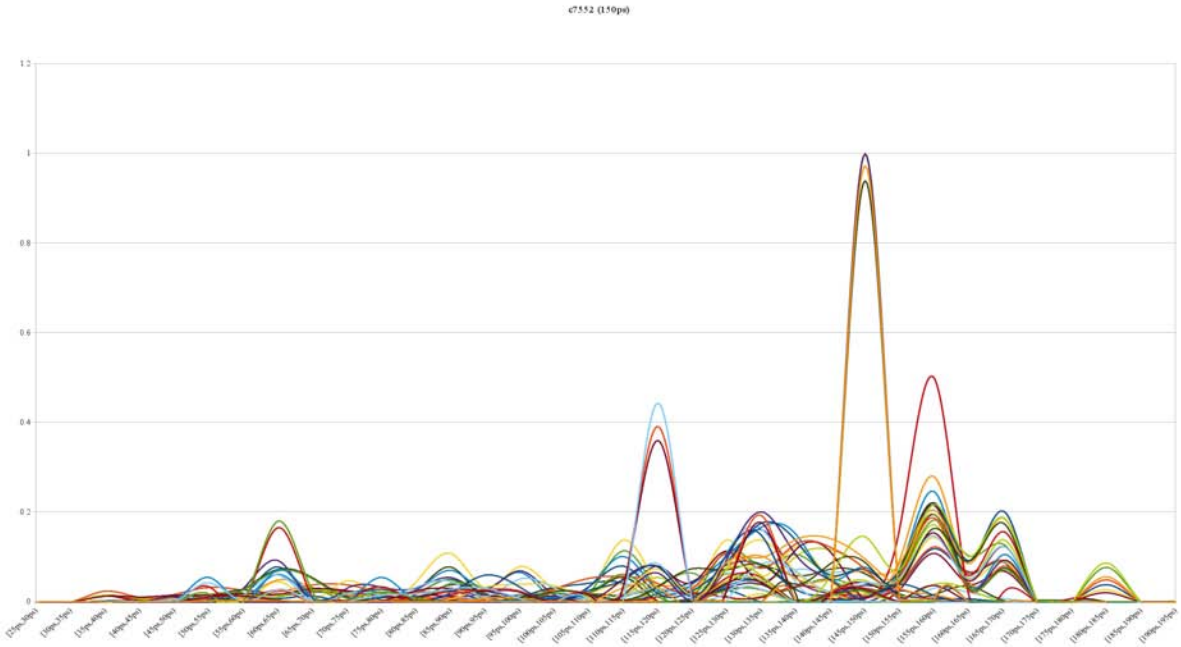


Figure 6.28: c7552 Output width distribution (150ps)

To get a wider information about the output width distribution we performed an injected pulse width sweep, obtaining a relationship between this width and the output pulse width. Figs. 6.29 to 6.34 show a 3D representation of these results for a various output nodes of the c3540 and c7552 benchmark circuits. These graphs plots the probability of each output pulse width related to the injected width.

Generalizing the previous results both graphs show that the higher output SET width probabilities are located on the line defined by $tw_{in} = tw_{out}$, i.e. the case where the SET traverses the path keeping its width.

The graph in Fig. 6.29 shows that for this output there is no pulse broadening, since the probability to get an output pulse wider than the injected pulse is practically zero. However, there are various pulses that suffers a narrowing.

Fig. 6.30 shows the results for another output node that presents a different behavior, since in this case pulse broadening is observed.

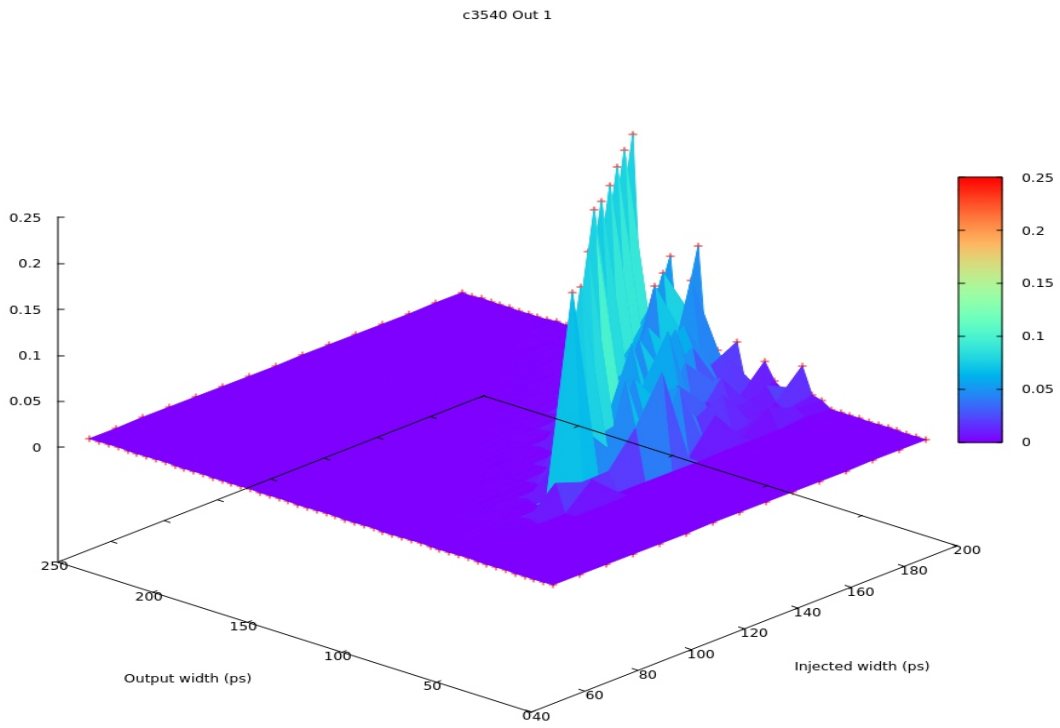


Figure 6.29: c3540 Injected-Output width

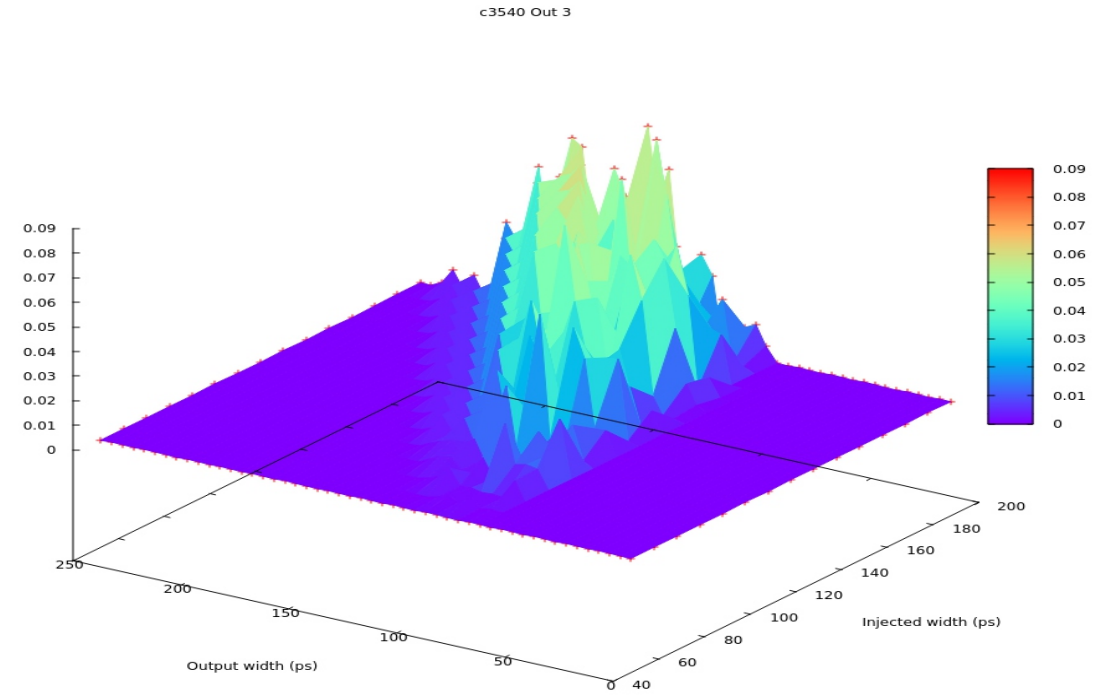


Figure 6.30: c3540 Injected-Output width

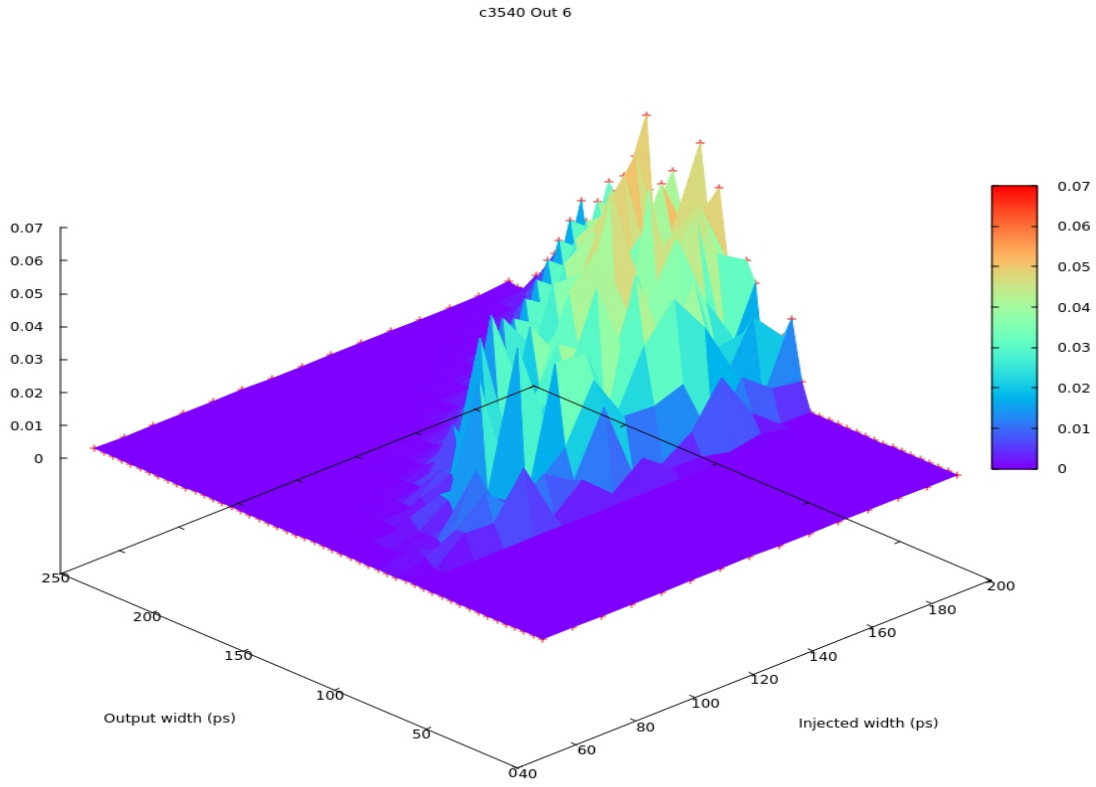


Figure 6.31: c3540 Injected-Output width

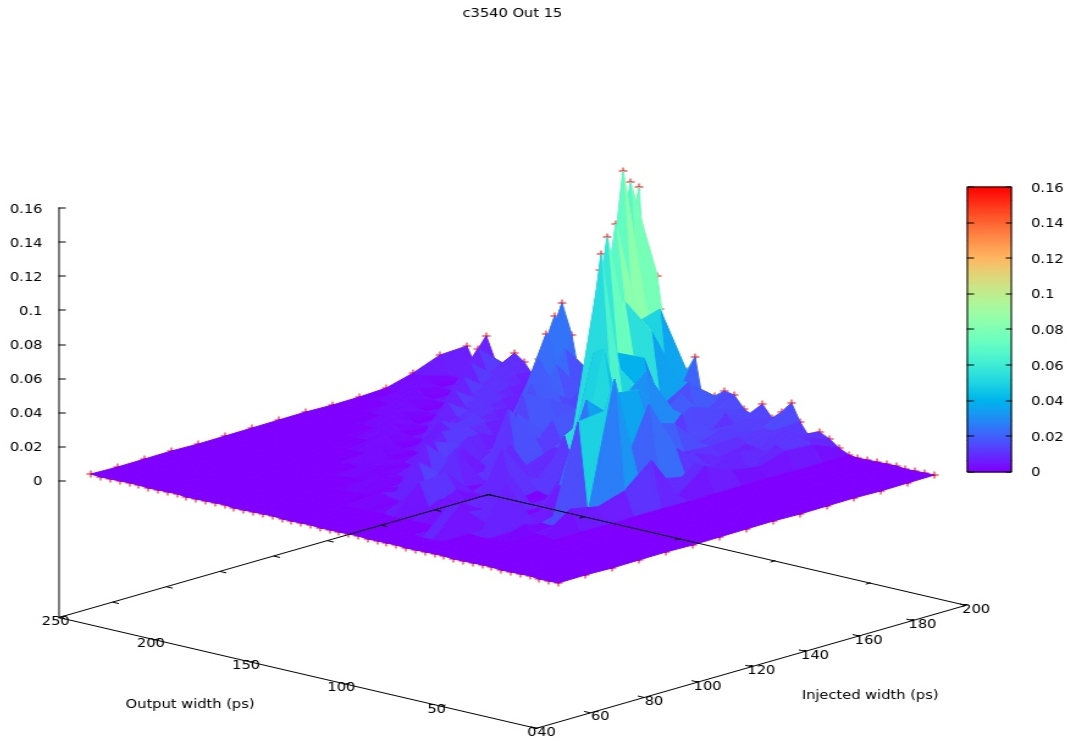


Figure 6.32: c3540 Injected-Output width

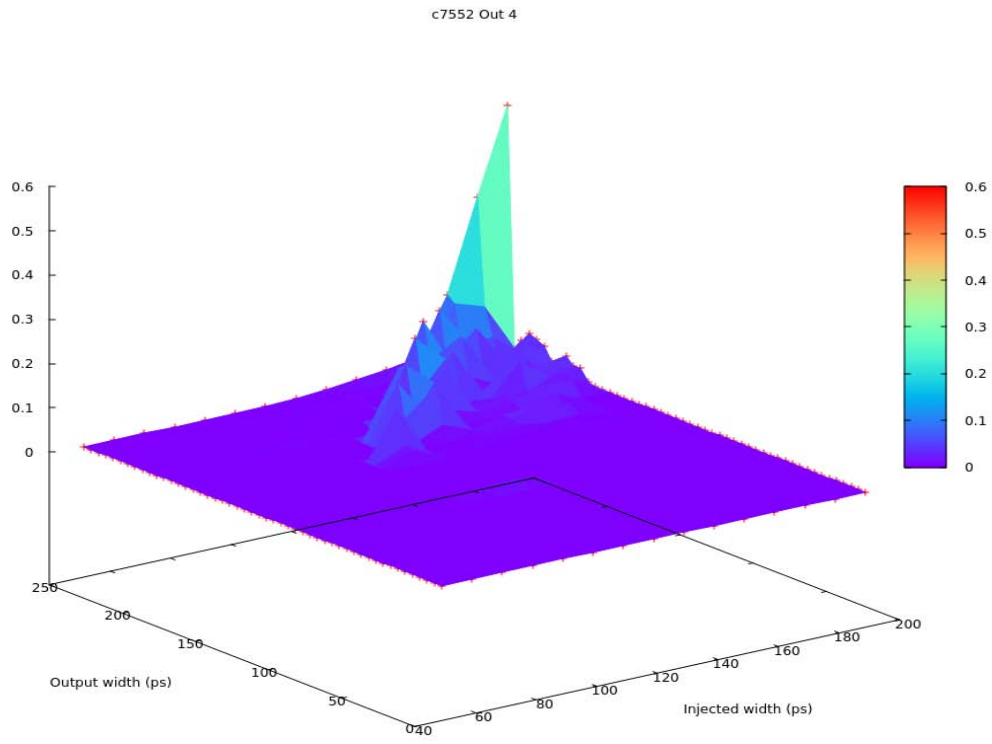


Figure 6.33: c7552 Injected-Output width

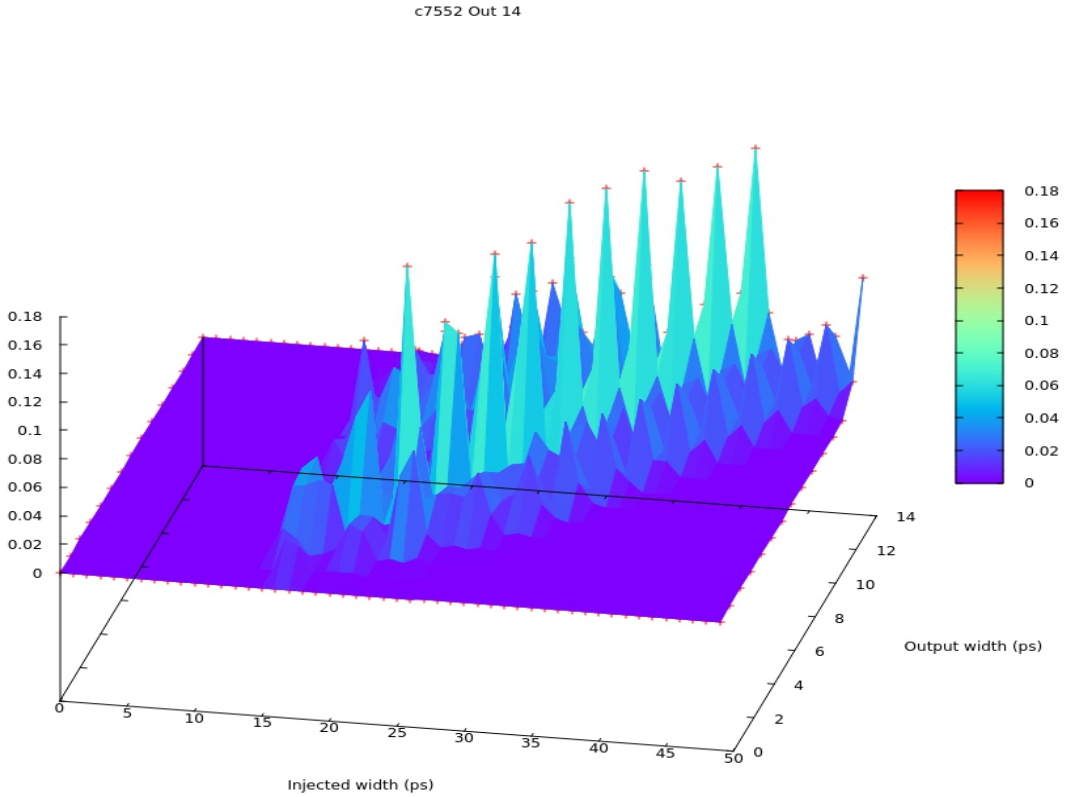


Figure 6.34: c7552 Injected-Output width

Chapter 7: Conclusions and future work

A complete framework for SET propagation sensitivity has been presented and evaluated. The framework comprises a number of processing tools capable of handling circuits with high complexity in an efficient way. This goal has been achieved through the application of appropriated circuit pre-processing, partitioning and collapsing techniques suited for each particular situation. Basic circuit analysis tasks have been developed in light of the low efficiency exhibited by commercial tools.

In this way, a quite efficient true path finding algorithm has been constructed and its efficacy demonstrated on large benchmark circuits. Results show that this module is capable of determining the required number of true paths for circuits where commercial tools fail to provide a list. Moreover we have found that the delay value of a given path depends on the sensitization vectors applied to the complex library gates. We have shown that in some cases delay variation due to different sensitization vectors may get up to 43% at the path level, being comparable to the path delay caused by process parameters variation. Such an improvement over the path delay computation, links such delay estimation to the specific sensitization vector and to the verification of the path being a true path, representing a significant improvement over commercial tools.

A compact specific logic system has been developed to enhance the performance of the algorithms constructed to propagate transitions within the circuit and handle efficiently SET propagation. Various simplification, partitioning and encapsulation techniques have been detailed

and analyzed to enhance the overall framework operation.

A polynomial implementation of an analytical delay model has been incorporated in the framework. After an automated gate technology library processing the required model parameters are extracted obtaining the optimal configuration for each gate. Such an extraction process is key for accurate SET propagation analysis.

SET propagation results have been thoroughly verified through extensive electrical simulations over benchmark circuits synthesized on commercial CMOS technologies. Results demonstrate an excellent SET propagation prediction.

Various SET propagation metrics have been proposed considering the impact of logic masking, electric masking and combined logic-electric masking. Such metrics provide a valuable tool to grade either in-circuit regions being more susceptible of propagating SET events toward the circuit outputs or circuit outputs more susceptible to produce SET events.

The SENSET tool application to large benchmark circuits has shown the framework capabilities in the SET propagation estimation domain. Based on the developed metrics, the tool is capable of identifying the list of circuit internal nodes most suitable to propagate an SET accounting for both the electrical and logical masking effects. Results can be weighted by the logic probability of a node being activated from the circuit input vectors. Similarly, the tool also provides information about the circuit output nodes with a higher probability of producing an SET under specific radiation environments. An additional tool analysis is capable of exhaustively determining the effect of pulse broadening/filtering once a specific SET event is induced at each circuit node.

The work presented here establishes the foundations for a future circuit analysis tool oriented not only to evaluate the weakest regions of a circuit in terms of SET propagation susceptibility, but also to automatically suggest circuit design modifications oriented to enhance circuit robustness against SET propagation.

References

[1] T.H. Ning, "Directions for silicon technology as we approach the end of CMOS scaling", 10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp. 3, 1-4 November 2010

[2] "International Technology Roadmap for Semiconductors (ITRS)". Edition 2011

[3] Gordon E. Moore, "Cramming more components onto integrated circuits", Electronic, vol. 38, no. 8, pp. 114, April 19 1965

[4] Riedlinger, R.J.; Bhatia, R.; Biro, L.; Bowhill, B.; Fetzer, E.;Gronowski, P.; Grutkowski, T, "A 32nm 3.1 billion transistor 12-wide-issue Itanium® processor for mission-critical servers", IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 84-86, 20-24 February

[5] R.C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies", IEEE Transactions on Device and Materials Reliability, vol. 5, no. 3, pp. 305-316, September 2005

[6] S. Barceló, X. Gili, S. A. Bota and J. Segura, "An SET propagation EDA tool based on analytical glitch propagation model", 14th European Conference on Radiation and Its Effects on Components and Systems (RADECS), pp. , September 2013

[7] X. Gili, S. Barceló, S.A. Bota, J. Segura, "Analytical Modeling of Single Event Transients Propagation in Combinational Logic Gates", IEEE Transactions on Nuclear Science, vol. 59, no. 4, pp. 971-979, August 2012

[8] D. Blaauw, K. Chopra, A. Srivastava and L. Scheffer, "Statistical timing analysis: from basic principles to state of the art", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 4, pp. 589-607, April 2008

[9] I. Keller, K. Ho Tam and V. Kariat, "Challenges in gate level modeling for delay and SI at 65nm and below", Design Automation Conference (DAC), pp. 468-473, June 2008

[10] S.R. Nassif, "Modeling and forecasting of manufacturing variations", Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 145-149, 2001

[11] A. Bosio, P. Girard, S. Pravossoudovitch, P. Bernardi and M.S. Reorda, "An Exact and Efficient Critical Path Tracing Algorithm", IEEE International Symposium on Electronic

Design, Test and Application, pp. 164-169, 2010

[12] Dong Xiang; Boxue Yin; Chakrabarty, K, "Compact Test Generation for Small-Delay Defects Using Testable-Path Information", Asian Test Symposium (ATS), pp. 424-429, 23-26 November 2009

[13] Shihheng Tsai and Chung-Yang Haung, "A false-path aware formal static timing analyzer considering simultaneous input transitions", Design Automation Conference (DAC), pp. 25-30, July 2009

[14] Lee, P.M.; Garfinkel, T.; Ko, P.-K.; Chenming Hu, "Simulating the competing effects of P- and N-MOSFET hot-carrier aging in CMOS circuits", IEEE Transactions on Electron Devices, vol. 41, no. 5, pp. 852-853, May 1994

[15] Kai-Chiang Wu; Marculescu, D., "Aging-aware timing analysis and optimization considering path sensitization", Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1-6, 14-18 March 2011

[16] Kumar, S.V.; Kim, C.H.; Sapatnekar, S.S., "Adaptive Techniques for Overcoming Performance Degradation Due to Aging in CMOS Circuits", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 4, pp. 603-614, April 2011

[17] Charles Hawkins and Jaume Segura, Introduction to Modern Digital Electronics. 1891121073

[18] Chin Jen Lin and S.M. Reddy, "On Delay Fault Testing in Logic Circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 6, no. 5, pp. 694-703, September 1987

[19] Kee Sup Kim, S. Mitra, P.G. Ryan, "Delay defect characteristics and testing strategies", IEEE Design & Test of Computers, vol. 20, no. 5, pp. 8-16, Sept-Oct 2003

[20] M. Hashimoto, Y. Yamada, H. Onodera, "Equivalent waveform propagation for static timing analysis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, no. 4, pp. 498-508, April 2004

[21] D. Kobayashi, K. Hirose, V. Ferlet-Cavrois, D. McMorrow, T. Makino, H. Ikeda, Y. Arai, M. Ohno, "Device-Physics-Based Analytical Model for Single-Event Transients in SOI CMOS Logic", IEEE Transactions On Nuclear Science, vol. 56, no. 6, pp. 3043-3049, 2009

[22] P.E. Dodd, M.R. Shaneyfelt, J.A. Felix and J.R. Schwank, "Production and propagation of

single-event transients in high-speed digital logic ICs", IEEE Transactions on Nuclear Science, vol. , no. 51, pp. 3278-3284, 2004

[23] Y. S. Dhillon, A. U. Diril, A. Chatterjee, A.D. Singh, "Analysis and Optimization of Nanometer CMOS Circuits for Soft-Error Tolerance", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14, no. , pp. 512-524, 2006

[24] L.W. Massengill, P. W. Tuinenga, "Single-Event Transient Pulse Propagation in Digital CMOS", IEEE Transactions on Nuclear Science, vol. , no. 55, pp. 2861-2871, 2008

[25] G.I. Wirth, M.G. Vieira, E.H. Neto, F.L. Kastensmidt, "Generation and Propagation of Single Event Transients in CMOS Circuits", IEEE Design and Diagnostics of Electronic Circuits and Systems, vol. , no. , pp. 196-201, 2006

[26] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burguer, L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic", International Conference on Dependable Systems and Networks (ICDSN), vol. , no. , pp. 389-398, 2002

[27] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage and T. Turflinger, "Digital Single event transient trends with technology node scaling", IEEE Transactions on Nuclear Science, vol. 53, no. , pp. 3462-3465, 2006

[28] Rajaramant, R. ; Kim, J.S. ; Vijaykrishnan, N. ; Xie, Y. ; Irwin, M.J. , "SEAT-LA: a soft error analysis tool for combinational logic", 19th International Conference on Embedded Systems and Design, pp. , 2006

[29] Ming Zhang ; Shanbhag, N.R. , " Soft-Error-Rate-Analysis (SERA) Methodology ", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 10, pp. 2140-2155, 2006

[30] Bin Zhang ; Wei-Shen Wang ; Orshansky, M. , "FASER: fast analysis of soft error susceptibility for cell-based designs", 7th International Symposium on Quality Electronic Design, pp. 760, 2006

[31] Rao, R.R. ; Blaauw, D. ; Sylvester, D. , "Soft Error Reduction in Combinational Logic Using Gate Resizing and Flipflop Selection", International Conference on Computer-Aided Design (ICCAD), pp. 502-509, 206

[32] S. Bose, P. Agrawal and V. D. Agrawal, "Deriving Logic Systems for Path Delay Test Generation", IEEE Transactions on Computers, vol. 47, no. 8, pp. 829-845, August 1998

[33] K. Fuchs, F. Fink, M.H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults", IEEE Transactions on Computer-Aided Design, vol. 10, no. , pp. 1323-1335, October 1991

[34] Huawei Li ; Yue Zhang ; Xiaowei Li , "Delay test pattern generation considering crosstalk-induced effects", 12th Asian Test Symposium, pp. 178-183, 2003

[35] Hideo Fujiwara, Logic Testing and Design for Testability.

[36] Fuchs, K.; Pabst, M.; Rossel, T., "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. 1550-1562, December 1994

[37] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms", IEEE Transactions on Computers, vol. C-32, no. 12, pp. 1137-1144, December 1983

[38] M.H. Schulz, E. Trischler and T.M Sarfert, "SOCRATES: a highly efficient automatic test pattern generation system", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 7, no. 1, pp. 126-137, 1988

[39] P. 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

[40] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to Algorithms. Second Edition. 0-262-03293-7

[41] Zijian He, Tao Lv, Huawei Li, Xiaowei Li, "Graph partition based path selection for testing of small delay defects", 15th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 499-504, 18-21 January 2010

[42] Lei Wu, Walker D.M.H., "A fast algorithm for critical path tracing in VLSI digital circuits", 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), pp. 178-186, October 2005

[43] Wangqi Qiu; Walker, D.M.H., "Testing the path delay faults of ISCAS85 circuit c6288", Proceesings of 4th International Workshop on Microprocessor Test and Verification: Common Challenges ans Solutions, pp. 19-24, May 2003

[44] H. Fujiwara and S. Toida, "The complexity of fault detection: An approach to design for testability", Proceedings of 12th International Symposium on Fault Tolerant Computing, pp.

101-108, June 1982

[45] S. Barceló, X. Gili, S. Bota and J. Segura, "Circuit reuse identification pre-processing tool for efficient CAD analysis", Proceedings of Conference on Design of Integrated Circuits and Systems (DCIS), pp. , November 2013

[46] Choudhury, M.; Mohanram, K., "Timing-driven optimization using lookahead logic circuits", 46th ACM/IEEE Design Automation Conference (DAC), pp. 390-395, July 2009

[47] Shiy Xu; Edirisuriya, E., "A new way of detecting reconvergent fanout branch pairs in logic circuits", 13th Asion Test Symposium, vol. , no. , pp. 354-357, 15-17 November 2004

[48] Chowdhary, A.; Kale, S.; Saripella, P.K.; Sehgal, N.K.; Gupta, R.K., "Extraction of functional regularity in datapath circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 9, pp. 1279-1296, September 1999

[49] Sreenivasa Rao, D.; Kurdahi, F.J., "Partitioning by regularity extraction", 29th ACM/IEEE Proceedings of Design Automation Conference, pp. 8-12, June 1992

[50] Chowdhary, A.; Kale, S.; Saripella, P.K.; Sehgal, N.K.; Gupta, R.K., "Extraction of functional regularity in datapath circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 9, pp. 1279-1296, September 1999

[51] Kutzschebauch, T., "Efficient logic optimization using regularity extraction", Proceedings of International Conference on Computer Design, pp. 487-493, 2000

[52] S. Barceló, X. Gili, S. Bota and J. Segura, "An efficient and scalable STA tool with direct path estimation and exhaustive sensitization vector exploration for optimal delay computation", Design, Automation & Test in Europa (DATE), pp. , 2011

[53] IEEE P1800, "SystemVerilog Working Group", , vol. , no. , pp. ,

[54] S. Wen-Tsong and W. Wanalertlak, "An advanced cell polynomial-base modeling for logic synthesis", IEEE international SOC Conference (SOCC), pp. 393-396, September 2003

[55] Feng Wang and Shir-Shen Chiang, "Scalable polynomial delay model for logic and physical synthesis", ICDA, pp. , August 2000

[56] e Silva, L.G.; Zhenhai Zhu; Phillips, J.R.; Miguel Silveira, L., "Variation-Aware, Library Compatible Delay Modeling Strategy", IFIP International Conference on Very Large Scale Integration, pp. 122-127, 16-18 October 2006

[57] S. Barceló, X. Gili, S. Bota and J. Segura, "Sensitization input vector impact on propagation delay for nanometer CMOS ICs: analysis and solutions", IEEE Transactions on VLSI circuits, vol. , no. , pp. , October 2013

[58] Kouno, T.; Hashimoto, M.; Onodera, Hidetoshi, "Input Capacitance Modeling of Logic Gates for Accurate Static Timing Analysis", Asian Solid-State Circuits Conference, 2005, pp. 453-456, November 2005

[59] T. El Motassadeq, V. Sarathi, S. Thameem and M. Nijam, "SPICE versus STA tools: challenges and tips for better correlation", IEEE International SOC Conference (SOCC), pp. 325-328, 2009

[60] H. Yaun-chung, C. Hsi-chuan, S. Shangzhi and D.H.C. Du, "Timing analysis of combinational circuits containing complex gates", In proceedings of International Conference on Computer Design: VLSI in Computers and Processors (ICCD), pp. 407-412, October 1998

[61] J.L. Guntzel, A.C. Medina Pinto, E. d'Avila, R. Reis, "ATG-based timing analysis of circuits containing complex gates", In Proceedings of 13th Symposium on Integrated Circuits and Systems Design, pp. 21, 2000

[62] C. Ting-Wei, C.Y.R. Chen and Wei-Yu Chen, "An efficient gate delay model for VLSI design", 25th International Conference on Computer Design (ICCD), pp. 450-455, October 2007

[63] J. Xue, D. Al-Khalili and C.N. Rozon, "A normalized intrinsic delay model of static CMOS complex gates for deep submicron technologies", Notheast Workshop on Circuits ans Systems (NEWCAS), pp. 17-20, June 2004

[64] Raja, S. ; Varadi, F. ; Becer, M. ; Geada, J., "Transistor level gate modeling for accurate and fast timing, noise, and power analysis", 45th ACM/IEEE Design Automation Conference (DAC), pp. 456-461, 2008

[65] N. Menezes, C. Kashyap, C. Amin, "A “true” electrical cell model for timing, noise, and power grid verification", 45th ACM/IEEE Design Automation Conference (DAC), pp. 462-467, 8-13 June 2008

[66] Seckin, U.; Yang, C.-K.K., "A Comprehensive Delay Model for CMOS CML Circuits", IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 55, no. 9, pp. 2608-2618, October 2009

[67] B.P. Das, V. Janakiraman, B. Amrutur, H.S. Jamadagni, N.V. Arvind, "Voltage and temperature scalable gate delay and slew models including intra-gate variations", 21st VLSI Design VLSID 2008, pp. , 2008

[68] B. Lasbouygues, R. Wilson, N. Azemard, P. Maurine, "Temperature- and voltage-aware timing analysis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 4, pp. 801-815, April 2007 2007

[69] Ming-Ting Hsieh, Shun-Yen Lu, Jing-Jia Liou, A. KiFli, "High quality pattern generation for delay defects with functional sensitized paths", 17th Asian Test Symposium ATS '08, pp. 131-136, 24-27 November 2008

[70] Xiang Lu, Zhuo Li, Wangqi Qiu, D.M.H. Walker, Weiping Shi, "Longest-path selection for delay test under process variation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 12, pp. 1924-1929, December 2005 2005

[71] Kai-Chiang Wu, D. Marculescu, "Aging-aware timing analysis and optimization considering path sensitization", Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1-6, 14-18 March 2011

[72] K. Christou, M.K. Michael, S. Neophytou, "Identification of critical primitive path delay faults without any path enumeration", 28th VLSI Test Symposium (VTS), pp. 9-14, 19-22 April 2010

[73] PrimeTime® Advanced Timing Analysis User Guide, Version E-2010.12, December 2010, Synopsys®, Inc.

[74] Synopsys ®Timing Constraints and OptimizationUser Guide. Version D-2010.03, March 2010, Synopsys®, Inc.

[75] Design Vision User Guide, Version D-2010.03, March 2010. Synopsys®, Inc.

[76] Design Compiter Optimization Reference Manual. Version D-2010.03, March 2010. Synopsys®, Inc.

[77] Y.S. Dhillon, A.U. Diril and A. Chatterjee, "Soft-error tolerance analysis and optimization of nanometer circuits", Proceedings of Design, Automation and Test in Europe (DATE), pp. 288-293, 7-11 March 2005

[78] Si2 (Silicon Integration Initiative), "Nangate Open Cell Library", , vol. , no. , pp. ,

[79] Mourad, S.; McCluskey, E.J., "On benchmarking digital testing systems", Proceedings of International Test Conference - New Frontiers in Testing, vol. , no. , pp. 997, 12-14 September 1988

[80] Basto, L., "First results of ITC'99 benchmark circuits", IEEE Design & Test of Computers, vol. , no. , pp. 54-59, Jul/Sep 2000

[81] Hansen, M.C.; Yalcin, H.; Hayes, J.P., "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering", IEEE Design & Test of Computers, vol. 16, no. 3, pp. 72-80, 1999

[82] S. Davidson, "ITC'99 Benchmark Circuits: Preliminary Results", Proceedings of International Test Conference (ITC), pp. 1125, 1999

A. Benchmark circuits

Qualification of circuit processing algorithms requires a test bench to grade their operation and the efficiency of the techniques used. Various benchmark circuits sets have been proposed in the field of the electronic design automation (EDA) tools development to provide a common framework where to check the algorithms behavior using circuit structures similar to real circuits. One of the most widely used benchmark circuits suite in the literature since their introduction in 1985 are the ISCAS'85 circuits [79][80], a set of combinational circuits published at the International Symposium of Circuits and Systems (ISCAS). They were based on industrial designs and provided in gate-level netlist format. Their high-level function was kept secret for confidentiality, and to provide a set of combinational blocks to be viewed as a bunch of logic gates without a specific high-level function. However multiple reverse engineering works have unveiled its high-level details [81].

The ISCAS'85 were followed by a second set of benchmark circuit published in 1989, referred to as the ISCAS'89. They are sequential circuits, instead of purely combinational blocks as the ISCAS'85. However, due to the quick evolution of the integration technology, ISCAS circuits became too simple for testing modern design tools, and a new set of benchmark circuits were introduced during the 1999 Int. Test Conference [82]. The ITC'99 benchmark circuits included designs from industry and universities, and have been extensively studied in the literature [80].

The techniques and algorithms presented in this work will be illustrated using circuits from the ISCAS'85 and ITC'99 benchmarks.

Benchmark circuits are provided in netlist form using generic logic gates. In this work, to get a realistic environment where to test our tools, the circuits have been synthesized on various technologies using the commercial tool *Design Compiler* from *Synopsys*® [75][76]. The synthesis tool was configured to make a high-effort synthesis with the objective of getting a realistic synthesis, since in real applications any achievable reduction in area, delay or power consumption usually will be exploited. The details of the circuits synthesized on a 65nm CMOS commercial technology are provided in Tables A.1 and A.2. Table A.1 includes details about the ISCAS'85 circuits, while Table A.2 details the ITC'99 ones. Both Tables provide the circuit name, its high-level function (in general obtained by reverse engineering), and the number of elements, nodes and gates, of each circuit.

A. Benchmark circuits

The number of nodes is detailed in categories:

- Input nodes: primary inputs.
- Output nodes: primary outputs.
- Wire nodes, circuit internal nodes (i.e. not Input nor Output nodes).

The number of input and output nodes of each circuit is determined by its high-level design, however, the number of wires and gates depends on the technology and parameters used to perform the circuit synthesis. For example if the synthesis is performed using only basic gates (NOT, OR, AND, NOR, NAND), the number of gates of the circuit will be higher than if the synthesis uses complex gates.

To get a wide test bench, in addition to commercial 130nm, 90nm and 65nm CMOS technologies we have also included a 45nm Nangate Open Cell Library from Si2 [78].

Table A.1: ISCAS'85 Benchmark circuits

Circuit	Function	Input	Output	Wire	Nodes	Gates
c17	Handy-level simple circuit	5	2	1	8	3
c432	27-channel interrupt controller (Priority decoder)	36	7	74	117	81
c499	32-bit SEC (ECAT)	41	32	105	178	137
c880	8-bit ALU	60	26	133	219	159
c1355	32-bit SEC (ECAT)	41	32	107	180	139
c1908	16-bit SEC/DED (ECAT)	33	25	139	197	164
c2670	12-bit ALU and controller	233	130	187	550	316
c3540	8-bit ALU and controller	50	22	409	481	431
c5315	9-bit ALU and selector	171	102	420	700	522
c6288	16-bit Multiplier	32	32	775	839	602
c7552	32-bit Adder/Comparator	207	94	576	877	670

Table A.2: ITC'99 Benchmark circuits

Circuit	Function	Input	Output	Wire	Nodes	Gates
b01	FSM (compare serial flows)	6	6	15	27	21
b02	FSM (recognizes BCD numbers)	5	5	11	21	16
b03	Resource arbiter	36	32	43	111	75
b04	Compute min and max	79	73	118	270	180
b05	Elaborate the contents of a memory	37	61	218	316	279
b06	Interrupt handler	6	9	19	34	28
b07	Count points on a straight line	47	49	165	261	212
b08	Find inclusions in sequences of numbers	32	24	45	101	69
b09	Serial to serial converter	31	31	38	100	69
b10	Voting system	29	18	76	123	94
b11	Scramble string with variable cipher	39	33	257	329	289
b12	1-player game	129	132	415	675	547
b13	Interface to meteorologic sensors	59	54	82	195	135
b14	Viper processor (subset)	247	238	1694	2179	1921
b15	80386 processor (subset)	454	460	3020	3934	3465
b17	Three copies of b15	1354	1485	9136	11975	10573
b18	Two copies of b14 and two of b17	3333	3321	24010	30664	26718
b19	Two copies of b14 and two of b17	6616	6621	46421	56658	52133
b20	A copy of b14 and a modified version of b14	523	514	3420	4457	3882
b21	Two copies of b14	523	514	3307	4344	3753
b22	A copy of b14 and two modified versions of b14	733	726	4724	6186	5310