

QA  
76.9  
.F38  
A55  
2010

# **OPTIMIZED SWITCH-LEVEL SOFT ERROR DETECTION BASED ON ADVANCED SWITCH-LEVEL MODELS**

by

**Jalal Mohammad Chikhe**

Maîtrise de science de la modélisation, de l'information et des systèmes

Mention Électronique, électrotechnique, automatique et systèmes

University of Paul Sabatier, Toulouse, France, 2007

A project report  
presented to Ryerson University  
in partial fulfillment of the  
requirements for the degree of  
Master of Engineering  
in the Program of  
Electrical and Computer Engineering  
Toronto, Ontario, Canada, 2010

©Jalal Mohammad Chikhe 2010

PROPERTY OF  
RYERSON UNIVERSITY LIBRARY

## **AUTHOR'S DECLARATION**

I hereby declare that I am the sole author of this project or dissertation. I authorize Ryerson University to lend this project or dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this project or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

# ABSTRACT

Project Title:

**Optimized switch-level soft error detection based on advanced switch-level models**

Project submitted by:

**Jalal Mohammad Chikhe**

**Optimization Problems Research and Applications Laboratory (OPR-AL)**

**Master of Engineering, Ryerson University, 2010**

Project Directed by:

**Dr. Reza Sedaghat**

Due to the reduction of transistor size, modern circuits are becoming more sensitive to soft errors. The development of new techniques and algorithms targeting soft error detection are important as they allow designers to evaluate the weakness of the circuits at an early stage of the design. This project presents an optimized implementation of soft error detection simulator targeting combinational circuits. The developed simulator uses advanced switch level models allowing the injection of soft errors caused by single event-transient pulses with magnitudes lesser than the logic threshold. The ISCAS'85 benchmark circuits are used for the simulations. The transients can be injected at drain, gate, or inputs of a logic gate. This gives clear indication of the importance of transient injection location on the fault coverage. Furthermore, an algorithm is designed and implemented in this work to increase the performance of the simulator. This optimized version of the simulator achieved an average speed-up of 310 compared to the non-algorithm based version of the simulator.

# ACKNOWLEDGEMENT

At first I would like express my gratitude to Dr Sedaghat for his advice, guidance and patience. I am grateful to have him as a mentor because his support helped me progress in my research. Being part of Optimization Problems Research and Applications Laboratory (OPR-AL) was one a most creative and energetic experience. I would like to thank the members of the (OP-RAL) for making me very comfortable when joining the team and for offering their help and support.

I dedicate this project to my parents. Their love and encouragement helped me through this journey.

# TABLE OF CONTENTS

<b>Abstract.....</b>	<b>iv</b>
<b>Acknowledgement.....</b>	<b>v</b>
<b>Table of Contents.....</b>	<b>vi</b>
<b>List of Tables.....</b>	<b>ix</b>
<b>List of Figures.....</b>	<b>xii</b>
<b>Nomenclature.....</b>	<b>xiv</b>
<b>Chapter 1      Introduction.....</b>	<b>1</b>
1.1. Background.....	1
1.2. Motivation.....	3
1.3. Summary of Contribution.....	3
1.4. Project Organization.....	4
<b>Chapter 2      Main Functions of the Simulator.....</b>	<b>5</b>
2.1. Verilog Strength and Logic Levels.....	6
2.2. Switch-Level Functions.....	9
2.2.1. funC Function.....	9
2.2.2. funN Function.....	10
2.2.3. funP Function .....	11
2.2.4. Capacitance Functionality.....	12
2.2.5. Example of Transient Injection at Drain of a Switch Applied to NAND2 Gate.....	13
<b>Chapter 3      Fundamental Architecture and Data Structure of the Simulator.....</b>	<b>15</b>
3.1. Programming Language.....	15
3.2. Simulation Environment.....	16
3.2.1. Data Input.....	17
3.2.2. Simulator Flow Diagram.....	18
3.2.3. Data Output.....	20

3.2.4. Processing Module.....	23
3.3. Data Structure of Netlist Gate-Level (netlist_g).....	24
3.4. Processing and Storing Arrays of Gate-Level.....	26
3.5. Data Structure of Netlist Switch-Level (netlist_s).....	28
3.5.1. Coding Architecture of netlist_s.....	28
3.5.2. NOT Gate.....	29
3.5.3. BUFFER Gate.....	32
3.5.4. NAND2 – AND2 Gates.....	33
3.5.5. NAND3 – AND3 Gates.....	36
3.4.6. NOR2 – OR2 Gates.....	38
3.4.7. NOR3 – OR3 Gates.....	40
3.5.8. Data Structure Switch-Level of C17 Circuit.....	43
3.6. Processing and Storing Arrays of Switch-Level.....	46
 Chapter 4      Detailed Representation of SimV <sub>1</sub> Simulator.....	 48
4.1. Flow Diagram of SimV <sub>1</sub> .....	48
4.2. Flow Diagram of Module 1.....	52
 Chapter 5      Detailed Representation of SimV <sub>2</sub> Simulator.....	 57
5.1. Gate-Level Resolution Function.....	57
5.3. Switch-Level Data Input Organization.....	63
5.3. Concept of the Algorithm Used in SimV <sub>1</sub> Simulator.....	63
5.3.1. 1 <sup>st</sup> Solution Based on C17 Circuit Example.....	67
5.3.2. 2 <sup>nd</sup> Solution Based on C17 Circuit Example.....	69
5.3.3. 3 <sup>rd</sup> Solution Based on C17 Circuit Example.....	70
5.3.4. 4 <sup>th</sup> Solution Based on C17 Circuit Example.....	71
 Chapter 6      Experimental Results.....	 75
6.1. Overview of the Simulation Applied Data.....	75
6.1.1. Benchmarks.....	75
6.1.2. Information on the Applied Data.....	77

6.2. Fault Coverage Due to Transient Injection at Different	
Location types.....	77
6.2.1. Injection at Gate of a Switch.....	77
6.2.2. Injection at Drain of a Switch.....	81
6.6.3. Injection at Inputs of a Logic Gate.....	83
6.3. Fault Coverage Versus Applied Test Vectors.....	85
6.4. Timing Results.....	87
6.6.1. Matlab Profile of SimV <sub>1</sub> and SimV <sub>2</sub> Simulators.....	87
6.6.2. Timing Results of SimV <sub>1</sub> Simulator.....	88
6.6.3. Timing Results of SimV <sub>2</sub> Simulator.....	90
6.6.4. Performance Study of the Developed the Simulators.....	93
Chapter 7	
Conclusion.....	96
Publications.....	98
Bibliography.....	99

# LIST OF TABLES

Table 1: Verilog coding representation of logic and strength levels and their decimal	
Representation .....	7
Table 2: map_det of C17 circuit .....	21
Table 3: map_detATV of C17 circuit .....	23
Table 4: Coding of the gate types .....	25
Table 5: Data structure of C17 netlist gate-level (netlist_g) before and after net reorganisation	
.....	26
Table 5(a): Data structure of C17 netlis gate-level before net reorganisation .....	26
Table 5(b): Data structure of C17 netlist gate-level after net reorganisation .....	26
Table 6: The main storing and indexing arrays needed to process C17 circuit gate-level .....	27
Table 7: Coding of the switch-level functions .....	29
Table 8: Coding and data structure of switch-level NOT gate .....	32
Table 9: Coding and data structure of switch-level BUFFER gate .....	33
Table 10: Data structure of switch-level NAND2 and AND2 gates .....	34
Table 11: Coded data of switch-level NAND2 and AND2 gates .....	35
Table 12: Data structure of switch-level NAND3 and AND3 gates .....	36
Table 13: Coded data of switch-level NAND3 and AND3 gates .....	37
Table 14: Data structure of switch-level NOR2 and OR2 gates .....	39
Table 15: Coded data of switch-level NOR2 and OR2 gates .....	40
Table 16: Data structure of switch-level NOR3 and OR3 gates .....	41
Table 17: Coded data of switch-level NOR3 and OR3 gates .....	41
Table 18: Data structure of netlist switch-level of C17 circuit .....	45



Table 19: Array of the storing processed data of C17 circuit switch-level .....	47
Table 20: Architecture of Array <sub>injD</sub> drain location array applied to NAND3 .....	56
Table 21: Example of gate-level resolution function applied to NAND2 gate: Gate <sub>1</sub> of C17 circuit covering the effect of the flipping bit of the output of the logic gate on the circuit for a set of 12 test vectors .....	61
Table 22: Data structure of collected data gate-level based on the resolution function applied to C17 circuit .....	63
Table 23: Structure of the collected data of NOT gate applied to SimV <sub>1</sub> * simulator .....	65
Table 24: Example of how the algorithm applied to SimV <sub>2</sub> simulator builds the solution maps map_det and map_detATV for C17 circuit .....	73
Table 25: map_det and map_detATV maps for C17 circuit .....	74
Table 26: Detailed logic gate information of ISCAS'85 benchmark circuits .....	76
Table 27: Information on ISCAS'85 benchmark circuits .....	77
Table 28: Simulation results for injection at gate of a switch for C5315 circuit .....	80
Table 29: Simulation results of F <sub>cov</sub> for transient injection at gate of a switch for ISCAS'85 benchmark circuits .....	81
Table 30: Simulation results for transient injection at drain of a switch for C5315 circuit .....	82
Table 31: Simulation results of F <sub>cov</sub> for transient injection at drain of a switch for ISCAS'85 benchmark circuits .....	83
Table 32: Simulation results for transient injection at inputs of a logic for C5315 circuit .....	84
Table 33: Simulation results of F <sub>cov</sub> for transient injection at inputs of a logic gate for ISCAS'85 benchmark circuits .....	84

Table 34: Timing results extracted from SimV <sub>1</sub> simulator based on transient injection at drain of a switch .....	89
Table 35: Timing results extracted from SimV <sub>2</sub> simulator .....	91
Table 36: Performance and speedup of SimV <sub>2</sub> versus SimV <sub>1</sub> based on all applied test vectors and based on transient injection at drain of a switch .....	94
Table 37: Performance and speedup of SimV <sub>2</sub> versus SimV <sub>1</sub> based on one test vector and based on transient injection at drain of a switch .....	95

# LIST OF FIGURES

Figure 1: Strength levels versus Voltage levels adapted from [16] .....	8
Figure 2: Flow Diagram of funC Function .....	10
Figure 3: Flow Diagram of funN Function .....	11
Figure 4: Flow Diagram of funP Function .....	12
Figure 5: Example of NAND2 gate in switch-level .....	13
Figure 6: Example of NAND2 gate in switch-level where the transient “11111” is injected at the drain of P1: PMOS .....	14
Figure 7: Flow diagram of the simulation environment .....	17
Figure 8: The simulator data input for C17 Circuit .....	18
Figure 9: Flow diagram of the Simulator .....	20
Figure 10: C17 Circuit before and after net reorganisation .....	24
Figure 10(a): C17 Circuit - Before net reorganisation .....	24
Figure 10(b): C17 Circuit - After net reorganisation .....	24
Figure 11: How the function code “4” processes the data .....	29
Figure 12: Switch-Level of BUFFER and NOT .....	31
Figure 13: Switch-Level of NAND2 and AND2 .....	35
Figure 14: Switch-Level of NAND3 and AND3 .....	38
Figure 15: Switch-Level of NOR2 and OR2 .....	39
Figure 16: Switch-Level of NOR3 and OR3 .....	43
Figure 17: Flow diagram of SimV <sub>1</sub> simulator .....	51
Figure 18: Flow diagram of the transient injection and detection Module 1 .....	55

Figure 19: Structure of the 3D array of gate-level resolution function applied to C17 circuit ...	62
Figure 20: Data structure of some logic gates extracted from SimV <sub>1</sub> * simulator .....	66
Figure 21: F <sub>cov</sub> for gate, drain of a switch and inputs of a logic gate for ISCAS'85 benchmarks circuits .....	85
Figure 22: Plot of F <sub>cov</sub> based on transient injection at drain of a switch versus applied test vectors for C499, C2670, C3515 and C7552 benchmarks circuits .....	86
Figure 23: Matlab Profile Display showing the execution time of SimV <sub>1</sub> functions .....	88
Figure 24: Matlab Profile Display showing the execution time of SimV <sub>2</sub> functions .....	88
Figure 25: SimV <sub>1</sub> Simulation time for ISCAS'85 benchmark circuits based on transient injection at drain of a switch .....	90
Figure 26: SimV <sub>2</sub> Simulation execution time for ISCAS'85 benchmark circuits .....	92

# NOMENCLATURE

<b>SET</b>	Single-Event Transient
<b>SEE</b>	Single-Event Effects
<b>CUT</b>	Circuit Under Test
<b>SimV<sub>1</sub></b>	Non-algorithm based soft error detection simulator
<b>SimV<sub>2</sub></b>	Algorithm based soft error detection simulator (Optimized Simulator)
<b>funC</b>	The function resolving the connection node
<b>funN</b>	The function representing the NMOS switch
<b>funP</b>	The function representing the PMOS switch
<b>Signal<sub>1L</sub></b>	Logic code of incident signal 1 (Same topology for Signal 2)
<b>Signal<sub>1s</sub></b>	Strength code of incident signal 1 (Same topology for Signal 2)
<b>OutC</b>	State code of output of funC
<b>OutC<sub>s</sub></b>	Strength code of funC output
<b>OutC<sub>L</sub></b>	Logic code of funC output
<b>Drain<sub>L</sub></b>	Logic code of the signal at drain
<b>Drain<sub>s</sub></b>	Strength code of the signal at drain
<b>Gate<sub>L</sub></b>	Logic code of the signal at gate
<b>Gate<sub>s</sub></b>	Strength code of the signal at gate of a switch
<b>Source<sub>L</sub></b>	Logic code of the signal at source of a switch
<b>Source<sub>s</sub></b>	Strength code of the signal at source of a switch

<b>Org<sub>m</sub></b>	Reorganizing, coding and parsing program of data inputs
<b>SimV<sub>1</sub>*</b>	SimV <sub>1</sub> simulator applied to all the logic gate types independently
<b>Prof<sub>m</sub></b>	Profiling maps generated by SimV <sub>1</sub> or SimV <sub>2</sub>
<b>Prof<sub>m</sub>*</b>	The 3D profiling maps generated by SimV <sub>1</sub> *
<b>map<sub>det</sub></b>	The 2D Profiling map of fault detection
<b>map<sub>detATV</sub></b>	The 2D Profiling map of the record of test vectors achieving transient detection
<b>T<sub>Loc</sub></b>	Transient location
<b>T<sub>L</sub></b>	Transient logic code
<b>T<sub>s</sub></b>	Transient strength code
<b>netlist<sub>g</sub></b>	Array based netlist gate-level
<b>Data<sub>in_g</sub></b>	Array of the addresses of the primary inputs
<b>Data<sub>out_g</sub></b>	Array of the addresses of the primary outputs
<b>Data<sub>inout_g</sub></b>	Storage array of the values of the connection links of CUT
<b>Stat<sub>v</sub></b>	Status of the content of the corresponding address (i.e. 0 or 1)
<b>netlist<sub>s</sub></b>	Array based netlist switch-level
<b>Cap<sub>v</sub></b>	Capacitance value at the node
<b>Cap<sub>s</sub></b>	Updated capacitance value at the node
<b>Add<sub>inc1</sub></b>	Incrementation start address of a logic gate of CUT (i.e. used to determine the address of VDD and GND).
<b>Add<sub>inc2</sub></b>	The remainder address incrementation of a logic gate of CUT without VDD and GND address
<b>Out<sub>g</sub></b>	Output of a gate

<b>Inc<sub>a</sub></b>	The first incrementation address of the links of the switch-level data structure
<b>Add<sub>cg</sub></b>	The last address of the last link corresponding to the previous logic gate
<b>Data_inout_s</b>	Storage array of the values of the connection links of CUT switch-level
<b>Cap<sub>e</sub></b>	The capacitance value exists for 1 and does not exist for 0 value
<b>L<sub>S</sub></b>	Strength code of a link
<b>L<sub>L</sub></b>	Logic code of a link
<b>var<sub>f</sub></b>	Variable of injected transient types
<b>nF</b>	The maximum number of injected transient types
<b>var<sub>l</sub></b>	Variable of locations of transient injection
<b>nL</b>	The maximum number locations of transient injection
<b>var<sub>v</sub></b>	Variable of index of the used test vector
<b>nV</b>	The maximum number of applied test vectors
<b>Init<sub>f</sub></b>	Initializes var <sub>f</sub> to 1.
<b>Init<sub>l</sub></b>	Initializes var <sub>l</sub> to 1.
<b>Init<sub>v</sub></b>	Initializes var <sub>v</sub> to 1.
<b>init<sub>all</sub></b>	Initializes init <sub>f</sub> , init <sub>l</sub> and init <sub>v</sub>
<b>det(var<sub>f</sub>,var<sub>l</sub>)</b>	Coefficients of map <sub>det</sub>
<b>detATV(var<sub>f</sub>,var<sub>l</sub>)</b>	Coefficients of map <sub>detATV</sub>
<b>Comp</b>	The results of comparing faulty and non-faulty CUT
<b>init<sub>ns</sub></b>	Initializes var <sub>ns</sub> , block B1 and B2 of Module 1 of SimV <sub>1</sub>

<b>Array<sub>injD</sub></b>	The 2D array of transient injection locations of drain location array
<b>var<sub>ns</sub></b>	Variable representing the row indexes of netlist <sub>s</sub>
<b>nS</b>	The maximum number of rows in netlist <sub>s</sub>
<b>Logic<sub>c</sub></b>	Array of gate-level resolution function used in SimV <sub>2</sub>
<b>In<sub>G</sub></b>	Decimal representation of logic combination + 1
<b>Out<sub>G</sub></b>	Logic value at the output of the logic gate
<b>Out<sub>GF</sub></b>	Flipped bit of logic value at the output of the logic gate
<b>Stat<sub>po</sub></b>	Status of the primary output due to Out <sub>GF</sub>
<b>Multi<sub>SInG</sub></b>	Multiplication of In <sub>G</sub> by Stat <sub>po</sub>
<b>n<sub>g</sub></b>	Number of logic gates in CUT
<b>N<sub>pg</sub></b>	Number of progressive logic gates in CUT
<b>TV<sub>n</sub></b>	The index of applied test vector similar to var <sub>v</sub>
<b>map<sub>det</sub>*</b>	The 3D Profiling map of fault detection for a switch-level logic gates
<b>map<sub>detATV</sub>*</b>	The 3D Profiling map of the record of the output of a switch-level logic gate
<b>n<sub>T.d</sub></b>	Total number of injected transients based on test vectors elimination
<b>n<sub>T.all</sub></b>	Total number of injected transients without test vectors elimination
<b>F<sub>cov</sub></b>	The fault coverage or soft error coverage
<b>nL<sub>d</sub></b>	The number of transient injection locations corresponding to detection
<b>T(Parameter)</b>	Execution time of the parameter number



<b><math>T_{\text{cycle}}</math></b>	Execution time to run the simulator for one transient injection and at one transient injection location and for one test vector
<b><math>T(\text{RF}_{\text{Gate-level}})</math></b>	The simulation execution time of the resolution function gate-level
<b><math>T(\text{P}_{\text{Algorithm}})</math></b>	The simulation execution time to apply the algorithm

# **Chapter 1**

## **Introduction**

### **1.1. Background**

As the dimension of transistors are reduced to nanometre scale, modern integrated circuits have the advantage of operating at low power and can achieve high speed characteristics, thus they are becoming more sensitive to external radiation [1]. This work focuses on soft errors in combinational circuits caused by single-event transients (SETs) which belong to Single-Event Effects (SEEs) category. When a vulnerable node within a combinational logic is hit by a cosmic particle, the produced disturbance might propagate to the primary output of the circuit [2]. This can create a soft error if the faulty output data are latched. The developed simulator in this work assumes that the faulty output data are always latched. Furthermore, this work does not apply to sequential circuits; it is mainly developed for combinational circuits. Moreover, due to the reduction of technology scaling, the internal electrical masking and latching-window masking are diminishing as modern digital circuits operate at a higher clock frequency compared to their predecessor. Due to the susceptibility of modern digital circuits to SETs, the research trend has shifted toward SETs in combinational logic.

The developed simulator for soft error detection in this work, operates by injecting a fault at certain location of the circuit under test (CUT), applies a set of test vector at the primary inputs of CUT and by comparing the faulty and non-faulty primary outputs of CUT, the simulator can determine whether there is a fault detection or not. There are different levels of abstraction that can be used for soft error detection including gate-level, electrical-level and switch-level. Gate-level soft error detection models are bit-flip based [3] [4] and the inputs of a logic gate are used as transient injection location [4] [5]. These models can't mimic the complex analog behavior of the transient propagation as the internal nodes of CUT can't be accessed [6]. As a result the simulators based on these models can be very fast and less reliable in terms of accuracy. The simulators for soft error detection based on electrical-level are reliable in terms of accuracy and very time consuming in terms of simulation speed but not feasible for complex digital circuits. There are examples of electrical-level simulators of soft error detection in the literature such as [7]. Switch-level soft error simulation based models can be a tradeoff between gate-level abstraction and electrical-level abstraction. Switch-level can model important characteristics in MOS circuits such as charge sharing and variation in driven strengths [8] [9]. The switch-level models used in this work are based on Verilog strength and logic levels and are provided by Verilog Hardware Descriptor Language Reference Manual [10]. These models can imitate important phenomena of electrical-level such as variations in driven strengths due to different voltage levels induced by SETs and the effect of the storage strength of the node capacitance. These switch-level models were previously used in different work such as in [11] where static faults are injected at gate of a switch.

## 1.2. Motivation

Due to progress in technology, modern circuits are becoming more sensitive to soft errors. The use of simulation based soft error detection is important as it allows detection of design weakness at early stage of the design. The use of switch-level models for soft error detection simulation is an alternative to electrical-level and gate-level models. Simulation soft error detection based on switch-level models can overcome the speed limitation and feasibility of complex digital circuits of electrical-level models and the accuracy limitation of gate-level models. Several soft error simulators using various techniques are available in the literature such as [12] [13]. The developed simulator in this work uses Verilog strength and logic levels models to simulate transient injection in combinational circuits. The developed simulator is array based and is programmed in Matlab scripting language. Furthermore, an algorithm is developed to significantly speed up the simulation process when compared to the non-algorithm based simulator based on the same switch-level models.

## 1.3. Summary of Contribution

The objective of this project is to present a soft error switch-level based simulator. This simulator has the ability to inject SETs of different strengths at different locations of a combinational circuit. The contribution to the project is summarized as following:

- Coding the main switch-level functions that allow modeling of CUT. These functions use models based on Verilog strength and logic levels [10] allowing the injection of transients of different logic levels and strength levels. These transients are classified in 23 types.
- Parsing the netlist gate-level of CUT expressed in Verilog into data structure gate-level.
- Parsing the data structure gate-level of CUT into switch-level data structure.

- The developed simulator can inject the transients at different location types such as drain, gate or inputs of a logic gate. The soft error coverage (or fault coverage) can be determined for any of these location types. This feature of the simulator shows the effect of the transient injection location type on the soft error coverage.
- The results of the simulation are stored in arrays called the profiling maps. These maps allow the determination of the status of the transient injection location versus the injected transient type and the test vector used to detect the corresponding transient.
- The created profiling maps are used to build statistical results based on the used test vectors. The progress of the fault coverage due to injected transients can be plotted against the order of the applied test vectors allowing the evaluation of the efficiency of the applied test vectors.
- Development of an algorithm allowing the design of new soft error detection simulator SimV<sub>2</sub>. This simulator achieved a significant speed-up compared to SimV<sub>1</sub> non-algorithm based developed in this work as well
- Timing equations are determined based on run time of the developed simulators SimV<sub>1</sub> and SimV<sub>2</sub>
- Analytical study is conducted based on the experimental results of fault coverage and simulation run time.

## 1.4. Project Organization

The remainder of the chapters is organized as follows: Chapter 2 discusses the main switch-level models implemented in this work. Chapter 3 explains the coding of CUT in gate-level and switch-level data structures and provides a large view of the simulation environment. Chapter 4 describes the architecture of the non-algorithm based simulator SimV<sub>1</sub>. Chapter 5 explains how

the developed algorithm is applied to SimV<sub>2</sub> simulator and describes its architecture. Chapter 6 presents the results of the experiments extracted from SimV<sub>1</sub> and SimV<sub>2</sub> simulators. Chapter 7 presents the conclusion of this work followed by a cited publication and references.

## **Chapter 2**

### **Main Functions of the Simulator**

#### **2.1. Verilog Strength and Logic Levels**

The switch-level soft error models in this work use the Verilog strength and logic levels [10]. Table 1 [14] shows how the logic levels and strength levels are structured and coded. There are 3 logic levels 0, 1, U, called respectively logic 0, logic 1, and unknown. These logic levels are associated with 7 levels of strength varying from small to supply. The 2-bit logic code represents the logic level and the 3-bit strength code represents the strength level. The state code combines the logic level and strength level codes and they are coded in 5-bit. This represents the state of the signals of the switch-level models. Logic levels Z and X have special significance. The logic code xx means that the logic of the signal has no significance as the strength level is below small strength level, thus the state codes associated to Z and X are respectively 00000 and 10000 representing “High Impedance” and the OFF state of CMOS switch. In addition, table 1 shows the decimal representation of the logic and strength codes. This representation is used for the simulation program.

Table 1: Verilog coding representation of logic and strength levels and their decimal representation

Logic Level	Verilog Strength Level	Logic Code	Strength Code	State code	Decimal Representation	
					Logic Code	Strength Code
0	Supply	00	111	00111	0	7
0	Strong	00	110	00110	0	6
0	Pull	00	101	00101	0	5
0	Large	00	100	00100	0	4
0	Weak	00	011	00011	0	3
0	Medium	00	010	00010	0	2
0	Small	00	001	00001	0	1
1	Supply	01	111	01111	1	7
1	Strong	01	110	01110	1	6
1	Pull	01	101	01101	1	5
1	Large	01	100	01100	1	4
1	Weak	01	011	01011	1	3
1	Medium	01	010	01010	1	2
1	Small	01	001	01001	1	1
U	Supply	11	111	11111	3	7
U	Strong	11	110	11110	3	6
U	Pull	11	101	11101	3	5
U	Large	11	100	11100	3	4
U	Weak	11	011	11011	3	3
U	Medium	11	010	11010	3	2
U	Small	11	001	11001	3	1
Z	High Z	xx	000	00000	0	0
X	Don't care	xx	000	10000	2	0

Voltage levels and strength levels of signals are related as shown in figure 1. There is a direct relationship between voltage levels and strength levels for 1.8 V supply voltage [16]. Switch-level strength-based models using Verilog logic and strength levels were previously used in



different works such as [11] where static faults are injected at gate of a switch. Similar switch-level models are used to study the delay introduced by resistive faults [14]. These switch-level models were used for soft error detection for the first time in the cited publication [15].

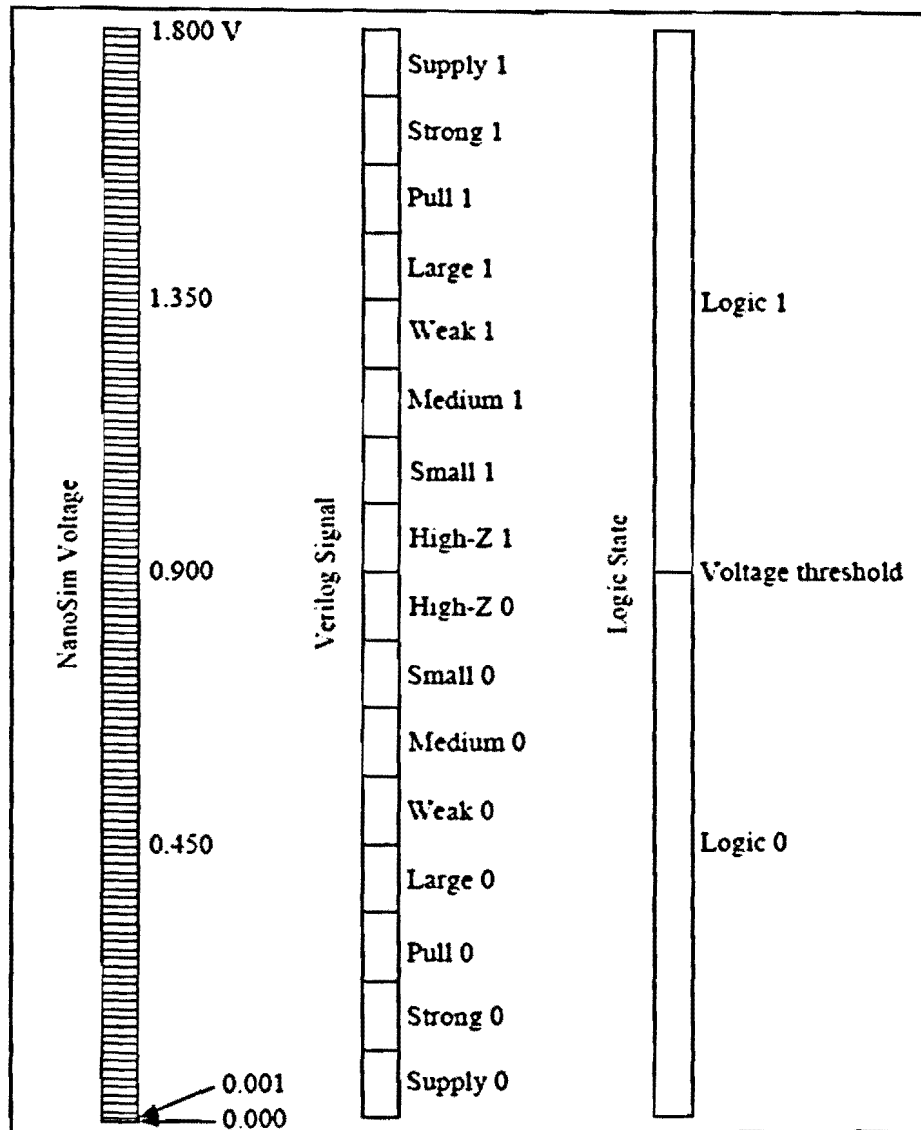


Figure 1: Strength levels versus Voltage levels adapted from [16].

## 2.2. Switch-Level Functions

The signals processed by the switch-level models are based on 5-bit coding representing the state code which combines the logic and strength levels as shown in Table 1. In the absence of transients, the strength code of all signals are assumed to be at Supply strength level “111”. The switch-level models in this work are based on Verilog signal resolution rules described as follows:

- When a switch is OFF, the signal at drain of a switch takes the state code “10000” and hence this signal does not participate in resolving the output at the connection node.
- When a switch is ON, the signal at drain of a switch will take the state code of the signal at the source of this switch.
- When the signal at gate of a switch has a logic code unknown (i.e. U) or state code of high impedance, then the state code of the signal at drain of this switch will be “11111” which is the unknown logic code and supply strength code.

### 2.2.1. funC Function

The flow diagram shown in Figure 2 describes the functionality of the connection node. This function resolves the output of the connection node for two incident signals presented at this node, Signal1 and Signal2. Signal1<sub>L</sub> and Signal2<sub>L</sub> represent respectively the logic code of the incident signals 1 and 2 at the node. Signal1<sub>S</sub> and Signal2<sub>S</sub> represent respectively the strength code of the incident signals 1 and 2 at the node. OutC represents the output of funC. OutC<sub>L</sub> and OutC<sub>S</sub> represent respectively the logic and strength codes of funC output.

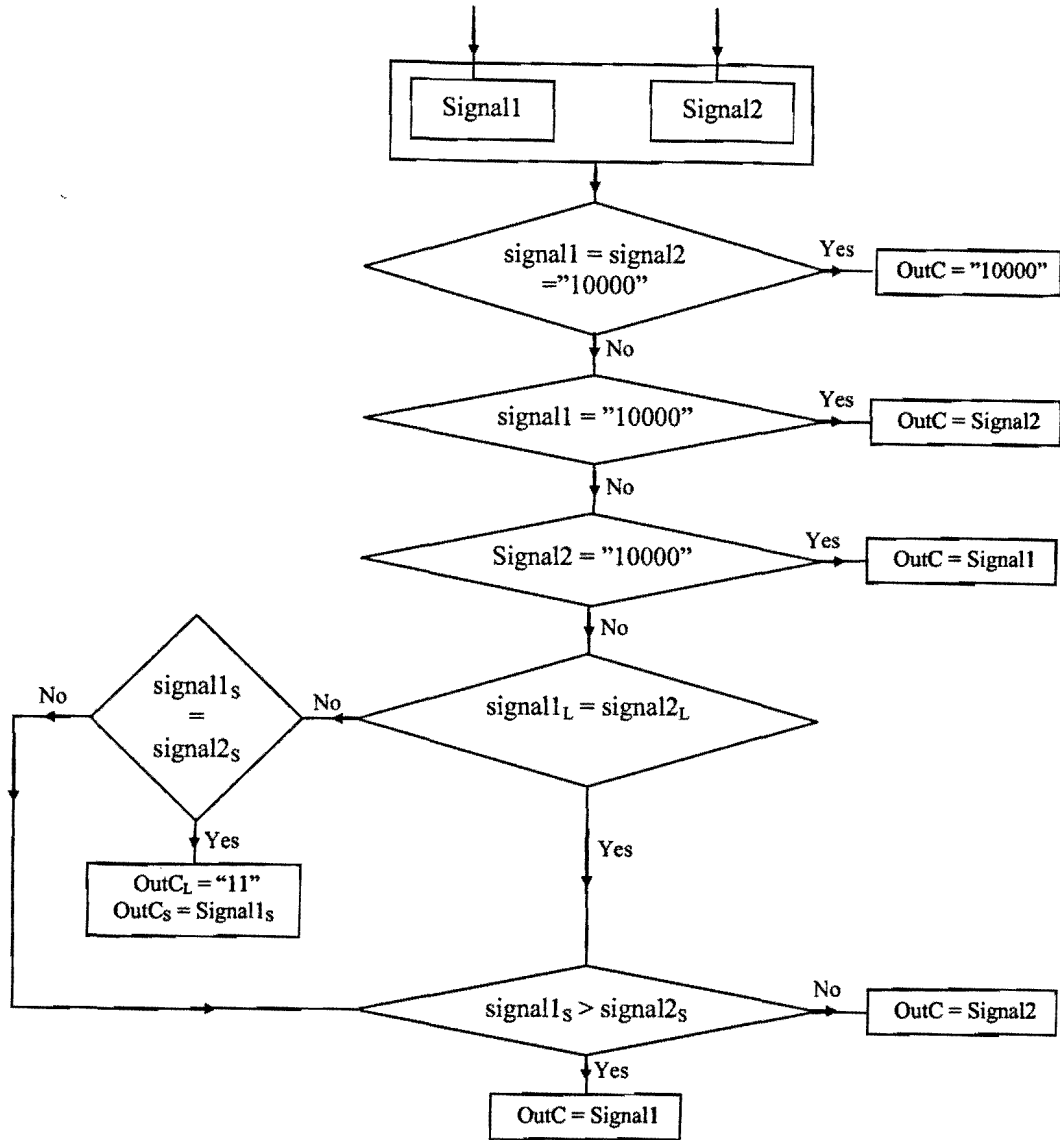


Figure 2: Flow Diagram of funC Function

**OutC<sub>L</sub>**: Logic code of funC output ; **OutC<sub>S</sub>**: Strength code of funC output;  
**Signal1<sub>L</sub>** , **Signal2<sub>L</sub>** : Logic code of the incident signals 1 and 2 at the node  
**Signal1<sub>S</sub>** , **Signal2<sub>S</sub>** : strength code of the incident signals 1 and 2 at the node

## 2.2.2. funN Function

This function is modeled based on Verilog rules described earlier. The flow diagram shown in Figure 3 describes the functionality of funN which represents the NMOS switch. Drain<sub>L</sub> and

Drain<sub>s</sub> represent respectively, the logic and strength code of the signal at drain. Gate<sub>L</sub> and Gate<sub>s</sub> represent respectively the logic and strength code of the signal at gate. Source<sub>L</sub> and Source<sub>s</sub> represent respectively the logic and strength code of the signal at source

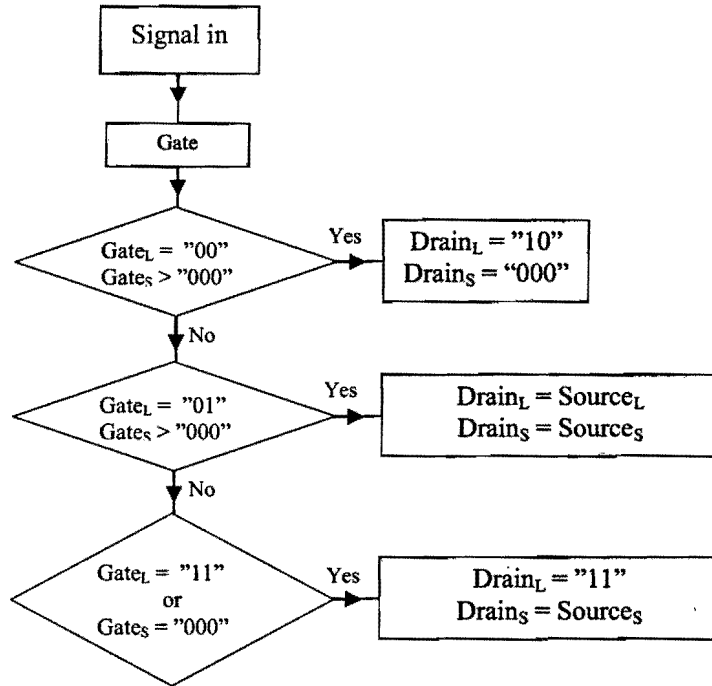


Figure 3: Flow Diagram of funN Function

**Gate<sub>L</sub>, Drain<sub>L</sub>, Source<sub>L</sub>:** Logic code of signals at gate, drain and source respectively  
**Gate<sub>s</sub>, Drain<sub>s</sub>, Source<sub>s</sub>:** Strength code of signals at gate, drain and source respectively

### 2.2.3. funP Function

Similarly, funP function is modeled based on Verilog rules described earlier. The flow diagram shown in Figure 4 describes the functionality of funP which represents the PMOS switch.

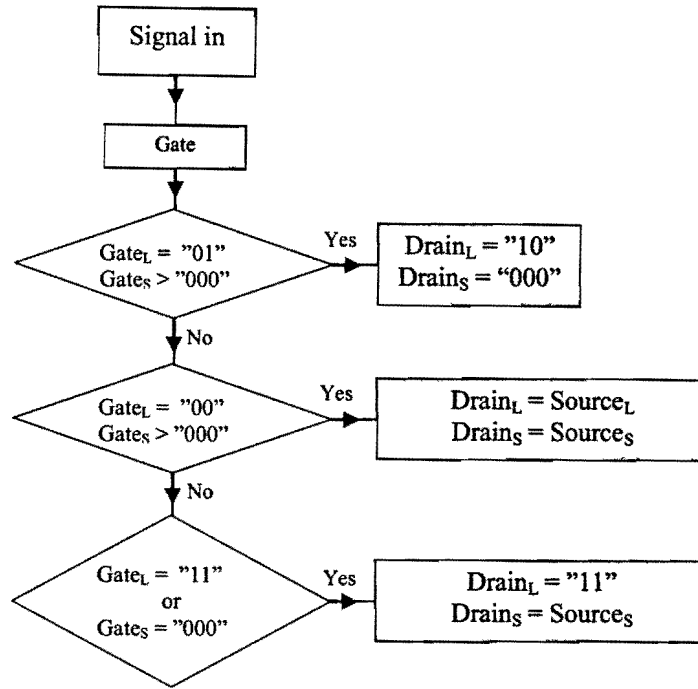


Figure 4: Flow Diagram of funP Function

$Gate_L$ ,  $Drain_L$ ,  $Source_L$ : Logic code of signals at gate, drain and source respectively.  
 $Gate_S$ ,  $Drain_S$ ,  $Source_S$ : Strength code of signals at gate, drain and source respectively.

## 2.2.4. Capacitance Functionality

Due to technology scaling and the shrinking size of transistors, the charge of capacitance values are weak [10]. The modeled capacitance in this work takes the strength code "001" (i.e. strength level "small"). The logic value of a capacitance takes the logic value of the output of the resolved connection node.

## 2.2.5. Example of Transient Injection at Drain of a Switch Applied to NAND2 gate

Figure 5 shows NAND2 gate translated in switch-level. The main functions used to resolve the output Y of NAND2 gate are funP, funN and funC. The capacitance values  $C_1$  and  $C_2$  are updated after applying funC at the node.

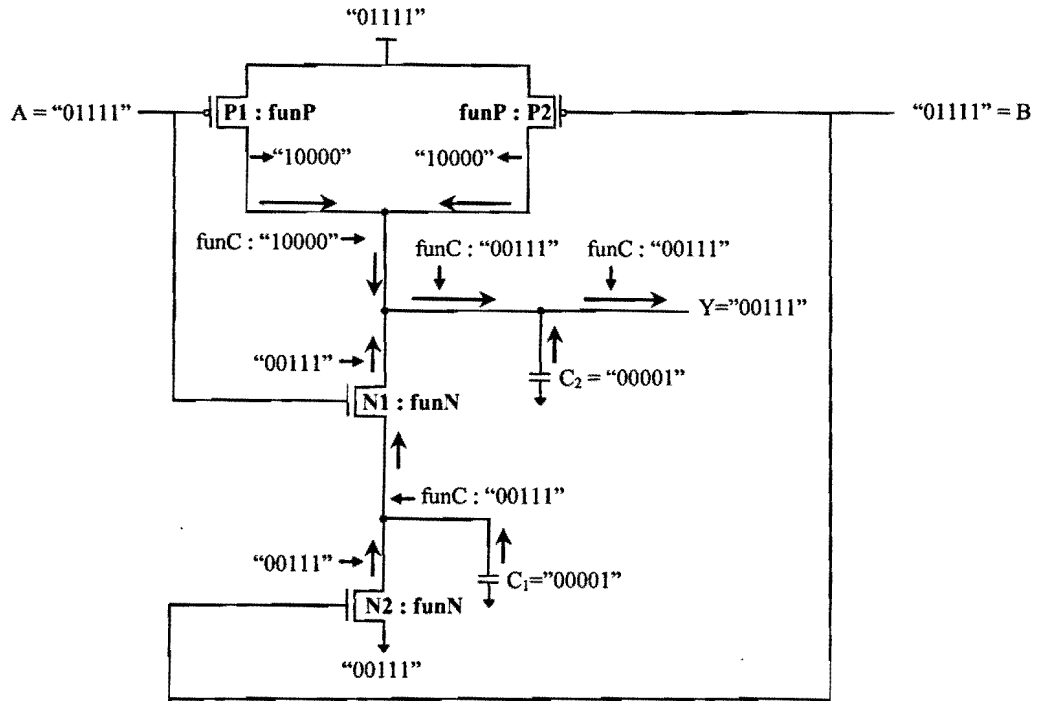


Figure 5: Example of NAND2 gate in switch-level

Figure 6 shows the same example of NAND2 gate when a transient "11111" is injected at the drain of P1 : PMOS switch. This transient injection results in soft error as the resolved output Y of the NAND2 gate changes to "11111". This example shows how the capacitance values are updated. According to the capacitance functionality, the capacitance strength code is "001" and its logic takes the logic of the resolved connection node. The "Before" transient injection and

“After” transient injection values of  $C_1$  and  $C_2$  show an example of how these capacitances update their values.

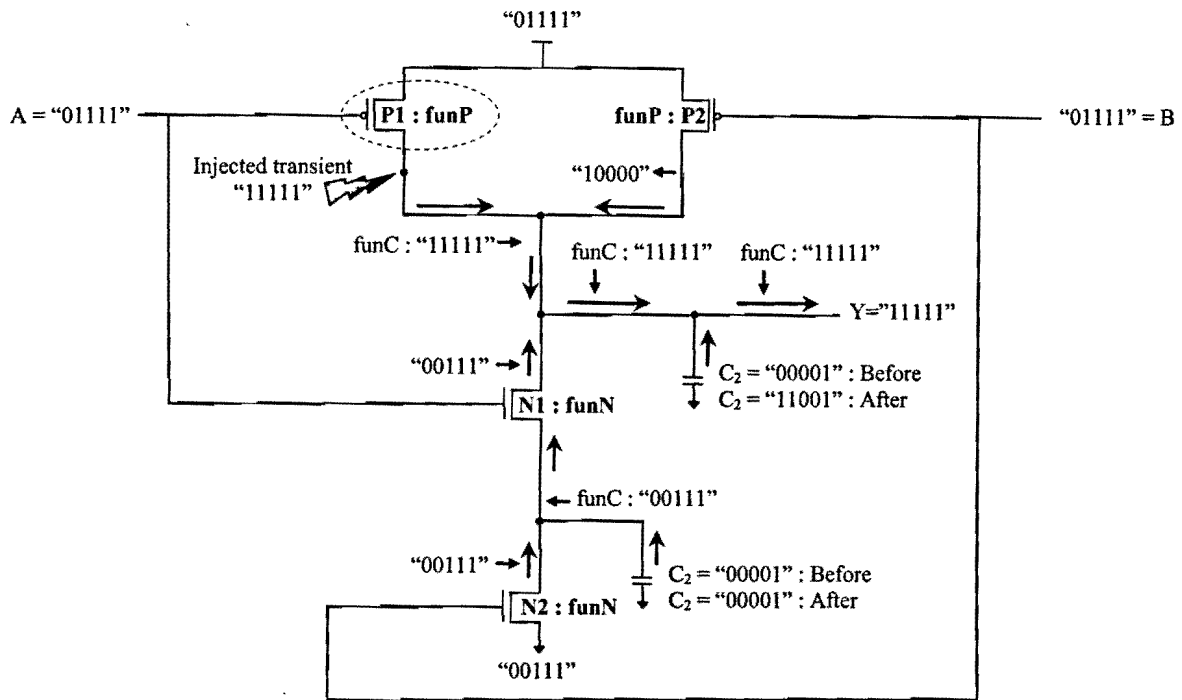


Figure 6: Example of NAND2 gate in switch-level where the transient “11111” is injected at the drain of P1: PMOS

## **Chapter 3**

# **Fundamental Architecture and Data Structure of the Simulator**

This chapter explains in details the architecture and the data structure of the simulation environment. The simulator is array based and a coding structure is used to code CUT in gate and switch-level. Examples are provided along this chapter to clarify the idea of how the data structures are coded.

### **3.1. Programming Language**

All the programs developed in this work are coded in Matlab using m-file format. A large amount of data is processed through the simulation and the use of an optimized programming language is important. Matlab scripting language is a combination of different languages offering simple, efficient and optimized functions targeting array based applications and offers different tools such as the Profile option.. This tool allows the user to optimize the code as it offers a detailed execution time report for the full program line by line. This option along other tools, help improve the speed of the developed program.



## 3.2. Simulation Environment

The flow diagram in Figure 7 presents a large view of the simulation environment. There are three stages for the simulation and can be classified as follows:

- **Data Input:** Contains the files representing CUT, the test vectors and the injected transient types. The test vectors in this work are based on compaction algorithm [18] and are design for gate-level circuits. These test vectors are used for the first time on the switch-level models used in this work for the purpose of soft error simulation. There are different formats and are loaded into the simulator (i.e. Block Program). A detailed explanation of the content of these files is provided in section 3.2.1.
- **Program:** Represents the processing part of the simulation environment and is coded entirely in Matlab scripting language. Different sections are dedicated for this block.
- **Statistical Results:** Represents the different results extracted from the simulation. These detailed results are used to conduct statistical study on the performance of the simulator, test vectors and the effect of transient injection location type on the fault coverage. Chapter 6 is based on these results.

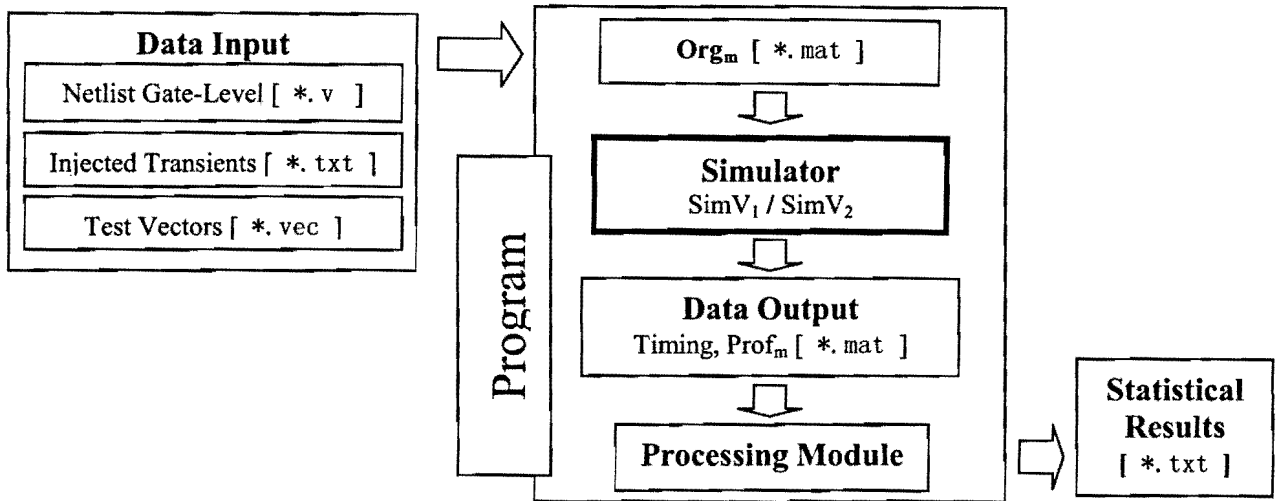


Figure 7: Flow diagram of the simulation environment

### 3.2.1. Data Input

The different file formats shown in figure 8 target the C17 benchmark circuit. Any other combinational circuit can be loaded into the simulator similarly as long as its format is expressed in the same manner as the C17 sample. These files are as follows:

- **C17\_TV.vec:** Represents the test pattern for the C17 circuit. Number 5 indicates the number of inputs of the circuit. The notation END indicates the end of the test vectors. This data is accessed line by line where the most significant bit is devoted to the first input of the circuit. Every bit of the data input is expressed in decimal and represent the logic level. The strength level is assumed to be “111” which is 7 in decimal. The inputs of CUT are then formatted in a logic and strength level format.
- **C17\_netlist\_g.v:** Represents the netlist gate-level of C17 circuit expressed in Verilog format. Any circuit can be loaded into the simulator in the same format.

- **Transients.txt:** Represents the decimal format of the transient types injected into CUT. The logic and strength levels are represented respectively by the column on the left and right.

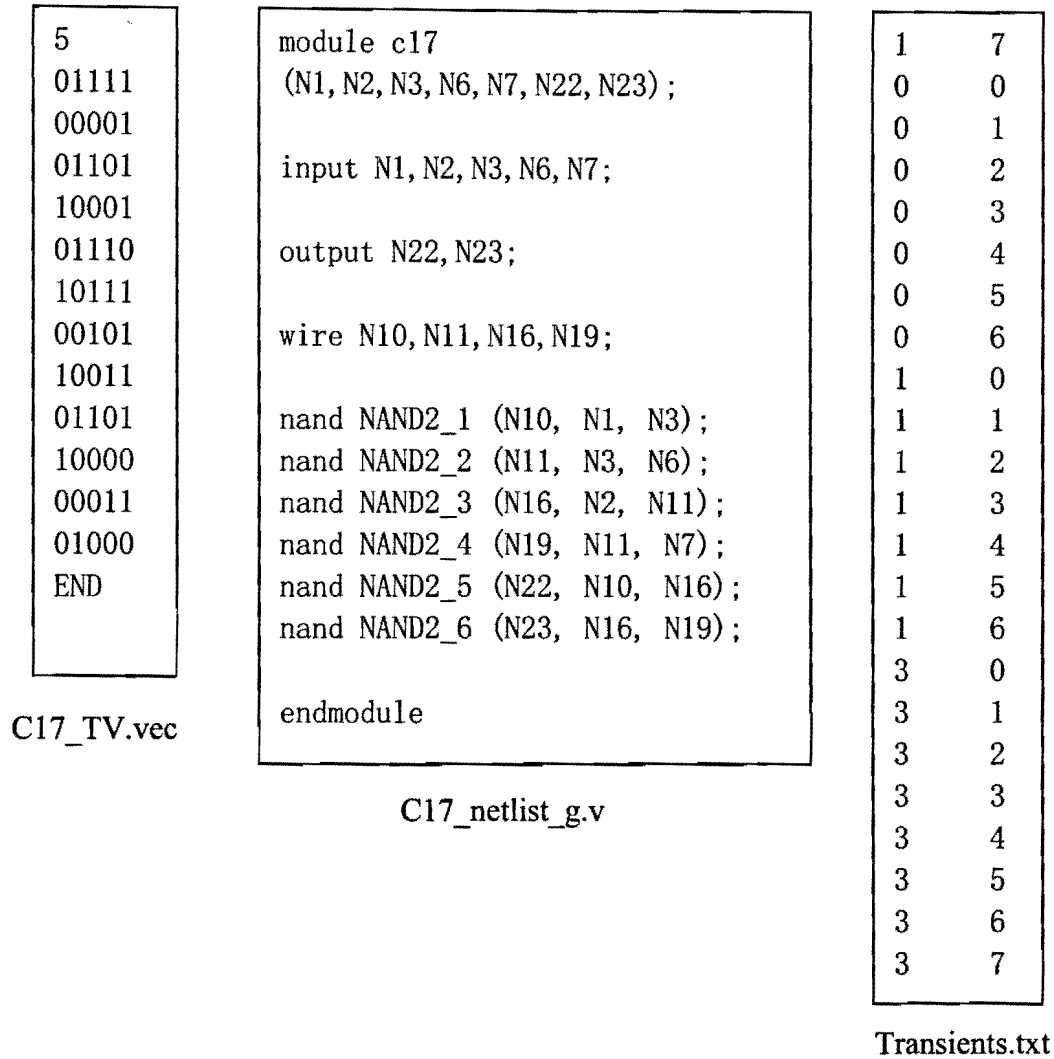


Figure 8: The simulator data input for C17 Circuit

### 3.2.2. Simulator Flow Diagram

The flow diagram in figure 9 shows a highlight of the simulation architecture. There are two developed simulators in this work; SimV<sub>1</sub> and SimV<sub>2</sub>. Chapter 4 and 5 provide respectively,

detailed explanation on  $\text{SimV}_1$  and  $\text{SimV}_2$ . The various blocks of this flow diagram are as follows:

- **Data Input:** Explained in section 3.2.1.
- **Org<sub>m</sub>:** Reorganizes, parses and codes the data input. Sections 3.3, 3.4, 3.5 and 3.6 provide detailed explanations of how these data structures are coded.
- **Case1:** Represents  $\text{SimV}_1$  non algorithm based version of the simulator.
- **Case2:** Represents  $\text{SimV}_2$  algorithm based version of the simulator. The block titled "Data Storage RUN: ONCE" processes all the logic gate types (i.e. AND2, NAND2, etc) independently by running  $\text{SimV}_1^*$  simulator on these gates. Exhaustive test vectors are applied for the simulation and the results are stored in 3D profiling maps (i.e.  $\text{Prof}_m^*$ ) in a \*.mat format file. Detailed explanation of  $\text{Prof}_m^*$ ,  $\text{SimV}_2$  and  $\text{SimV}_1^*$  is provided in chapter 5. The block "Data Storage RUN: ONCE" will run only one time and generates a \*.mat file that is used for any circuit simulated by  $\text{SimV}_2$ , thus its run time is not part of the simulation time.
- **Data Output:** Provides the results of the simulation in profiling maps arrays  $\text{Prof}_m$  and the simulation execution time (i.e. Timing). Section 3.2.3 is dedicated to the structure of the profiling maps  $\text{Prof}_m$ .

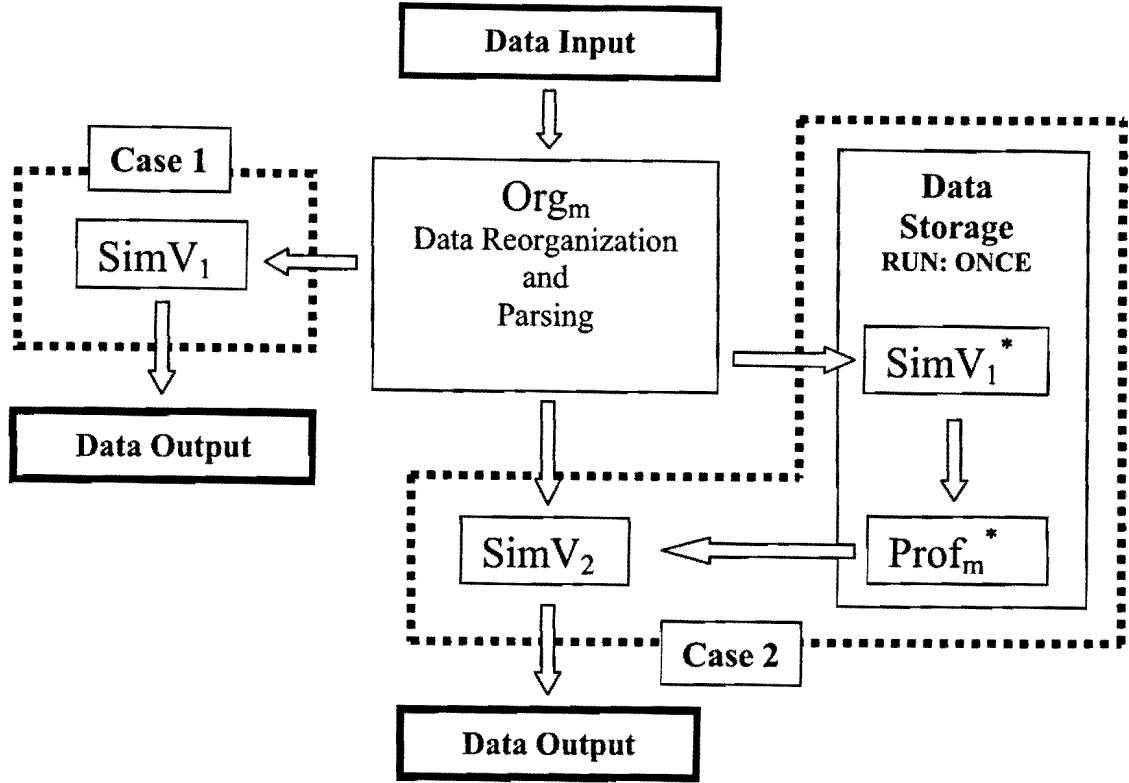


Figure 9: Flow diagram of the Simulator

### 3.2.3. Data Output

The data output of the simulators are stored in 2D arrays and are called profiling maps "Prof<sub>m</sub>" due to the way these arrays store the results of the simulation. The C17 circuit profiling maps are used as example of the data output. Table 2 shows the profiling map map\_det representing the fault detection due to transient injection at a drain of a switch. This table is organized as follows:

- **T<sub>Loc</sub>**: Represents the location of drain or gate as transient injection location for C17 circuit. There are 24 drains or gates in C17 circuit where the transient can be injected.
- **T<sub>L</sub>**: Represents the transient logic code expressed in decimal

- $T_S$ : Represents the transient strength code expressed in decimal

The values in table 2 are either 1 or 0 representing respectively, detection or non-detection of the transient injection. For example:

[Transient location: " $T_{Loc} = 5$ ", Transient value: " $T_L:T_S$ " = "1:7", Results = ( 1 ) ]

Explanation: The corresponding injected transient is detected at location (5).

Table 2: map\_det of C17 circuit

		$T_{Loc}$											
		1	2	3	4	5	6	7	8	.....			
$T_L$	$T_S$												
1	7	1	1	1	1	1	1	1	1				
0	0	0	1	0	0	0	1	0	0				
0	1	1	1	0	0	1	1	0	0				
0	2	1	1	0	0	1	1	0	0				
0	3	1	1	0	0	1	1	0	0				
0	4	1	1	0	0	1	1	0	0				
0	5	1	1	0	0	1	1	0	0				
0	6	1	1	0	0	1	1	0	0				
1	0	0	1	0	0	0	1	0	0				
1	1	0	1	1	1	0	1	1	1				
1	2	0	0	1	1	0	0	1	1				
1	3	0	0	1	1	0	0	1	1				
1	4	0	0	1	1	0	0	1	1				
1	5	0	0	1	1	0	0	1	1				
1	6	0	0	1	1	0	0	1	1				
3	0	0	1	0	0	0	1	0	0				
3	1	1	1	1	1	1	1	1	1				
3	2	1	1	1	1	1	1	1	1				
3	3	1	1	1	1	1	1	1	1				
3	4	1	1	1	1	1	1	1	1				
3	5	1	1	1	1	1	1	1	1				
3	6	1	1	1	1	1	1	1	1				
3	7	1	1	1	1	1	1	1	1				

Table 3 shows map\_detATV profiling map. This array is organized similarly to map\_det array. The results in this table represent the index of the test vector used to detect the injected transient in case of detection. For example:

- [ Transient location: " $T_{Loc} = 5$ ", Transient value: " $T_L:T_S$ " = "0:1", Results = ( 8 ) ]

Explanation: The 8<sup>th</sup> test vector detects the injected transient "0:1" at location (5).

- [ Transient location: " $T_{Loc} = 1$ ", Transient value: " $T_L:T_S$ " = "0:0", Results = ( 12 ) ]

Explanation: All the 12 test vectors are used and the injected transient "0:0" at location (1) is not detected. map\_det shows (0) as results indicating a non detection of transient injection at this location.

The sum of all the values of map\_detATV represents the number of times the circuit is simulated in other terms; this is called the total number of transients.

The sum of all the values of map\_det represents the number of detected faults. The number of coefficient of map\_det array is called the number of injected faults. The soft error coverage is evaluated based on ( EQ – 3 – 1 ) equation:

$$Soft\ Error\ Coverage\ (\%) = \frac{Number\ of\ detected\ faults}{Number\ of\ injected\ faults} \cdot 100 \quad (EQ - 3 - 1)$$

Table 3: map\_detATV of C17 circuit

$T_{Loc}$		1	2	3	4	5	6	7	8	.....	21	22	23	24	
$T_L$	$T_S$	1	7	1	1	4	1	2	2	1	1	2	2	1	1
0	0	12	4	12	12	12	12	3	12	12	12	12	2	12	12
0	1	1	4	12	12	8	3	12	12	12	12	12	12	2	12
0	2	1	4	12	12	8	3	12	12	12	12	12	12	2	12
0	3	1	4	12	12	8	3	12	12	12	12	12	12	2	12
0	4	1	4	12	12	8	3	12	12	12	12	12	12	2	12
0	5	1	4	12	12	8	3	12	12	12	12	12	12	2	12
0	6	1	4	12	12	8	3	12	12	12	12	12	12	2	12
1	0	12	4	12	12	12	3	12	12	12	12	12	12	2	12
1	1	12	4	6	6	12	3	1	1	1	1	12	2	1	1
1	2	12	12	6	6	12	12	1	1	1	1	12	12	1	1
1	3	12	12	6	6	12	12	1	1	1	1	12	12	1	1
1	4	12	12	6	6	12	12	1	1	1	1	12	12	1	1
1	5	12	12	6	6	12	12	1	1	1	1	12	12	1	1
1	6	12	12	6	6	12	12	1	1	1	1	12	12	1	1
3	0	12	4	12	12	12	3	12	12	12	12	12	2	12	12
3	1	1	4	6	6	8	3	1	1	1	1	12	2	1	1
3	2	1	4	6	6	8	3	1	1	1	1	12	2	1	1
3	3	1	4	6	6	8	3	1	1	1	1	12	2	1	1
3	4	1	4	6	6	8	3	1	1	1	1	12	2	1	1
3	5	1	4	6	6	8	3	1	1	1	1	12	2	1	1
3	6	1	4	6	6	8	3	1	1	1	1	12	2	1	1
3	7	1	1	4	1	1	1	1	1	1	1	1	1	1	1

### 3.2.4. Processing Module

This block processes the profiling maps; map\_det and map\_detATV in order to calculate the soft error coverage and other statistical results. This module provides the Statistical Results mentioned in figure 7. All the results in chapter 6 are provided by this module.



### 3.3. Data Structure of Netlist Gate-Level (netlist\_g)

This section explains the data structure and the coding architecture of gate-level netlist of CUT. The schematic of C17 circuit shown in figure 10(a) translates the Verilog netlist gate-level (i.e. input file of the simulator). Every logic gate is coded accordingly to table 4. Based on these codes, the first column of table 5(a) is code (11) representing a NAND2 gate and the second column represents the address of the output. Section 3.4 explains in detail how these addresses are used. Columns in1 to in10, are the addresses of the inputs of the logic gates. An address "0", indicates that the corresponding input does not exist. This structure is the main idea behind the simulation. Any combinational circuit can be parsed into this structure. There are 10 possible inputs indicating that this simulator can handle circuits built with up to 10 logic gates. Table 4 shows the different gate types and their corresponding codes.

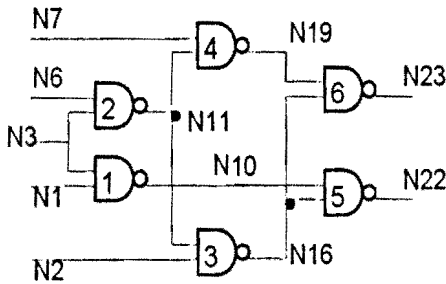


Figure 10(a): C17 Circuit  
Before net reorganisation

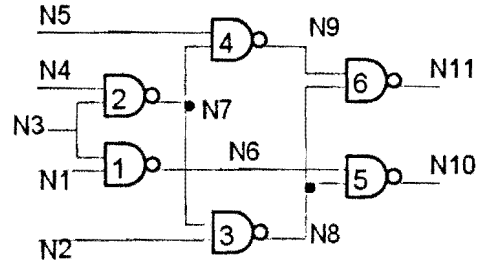


Figure 10(b): C17 Circuit  
After net reorganisation

Figure 10: C17 Circuit before and after net reorganisation

Based on the information of Table 5(a), the maximum number of addresses in this 2D array is 23. The block  $Org_m$  mentioned in section 3.2.2. reorganizes these addresses in a way that their numbers will be minimized when there are gaps between these numbers. Figure 10(b) and Table 5(b) show the results of the addresses reorganization. The maximum number of addresses after

reorganization is 11. This operation is useful as it reduces the size of the storage array and resulting in maximizing the speed of the simulation. The storage array Data\_inout\_g as shown in table 6, its size depends on the number of nets in the circuit and as the index of the number of the nets is reduced, the size of the storage array is reduced. Section 3.4. explains in detail the structure of this storage array.

Table 4: Coding of the gate types

Gate Type	Gate Code	Gate Type	Gate Code
BUFFER	1	AND10	21
NOT	2	AND9	22
		AND8	23
NAND10	3	AND7	24
NAND9	4	AND6	25
NAND8	5	AND5	26
NAND7	6	AND4	27
NAND6	7	AND3	28
NAND5	8	AND2	29
NAND4	9		
NAND3	10	OR10	30
NAND2	11	OR9	31
		OR8	32
NOR10	12	OR7	33
NOR9	13	OR6	34
NOR8	14	OR5	35
NOR7	15	OR4	36
NOR6	16	OR3	37
NOR5	17	OR2	38
NOR4	18		
NOR3	19	XOR2	39
NOR2	20		

Table 5: Data structure of C17 netlist gate-level (netlist\_g) before and after net reorganisation

Table 5(a): Data structure of C17 netlis gate-level  
before net reorganisation

Gate Type	Output	in1	in2	in3	in4	in5	in6	in7	in8	in9	in10
11	10	1	3	0	0	0	0	0	0	0	0
11	11	3	6	0	0	0	0	0	0	0	0
11	16	2	11	0	0	0	0	0	0	0	0
11	19	11	7	0	0	0	0	0	0	0	0
11	22	10	16	0	0	0	0	0	0	0	0
11	23	16	19	0	0	0	0	0	0	0	0

Table 5(b): Data structure of C17 netlist gate-level  
after net reorganisation

Gate Type	Output	in1	in2	in3	in4	in5	in6	in7	in8	in9	in10
11	6	1	3	0	0	0	0	0	0	0	0
11	7	3	4	0	0	0	0	0	0	0	0
11	8	2	7	0	0	0	0	0	0	0	0
11	9	7	5	0	0	0	0	0	0	0	0
11	10	6	8	0	0	0	0	0	0	0	0
11	11	8	9	0	0	0	0	0	0	0	0

### 3.4. Processing and Storing Arrays of Gate-Level

This section explains how a circuit is processed and its primary output is resolved. Table 6 shows the following:

- Data\_in\_g: Contains the addresses of the primary inputs of C17 circuit
- Data\_out\_g: Contains the addresses of the primary output of C17 circuit
- Data\_inout\_g: Contains the resolved values of every net of the circuit. Address Index is the address of input or output of any logic gate as shown in Table 5(b). L<sub>L</sub> and L<sub>S</sub> represent

respectively the logic and strength codes of CUT connection links.  $Stat_v$  represents the status of the content of the corresponding address. " $Stat_v = 1$ " indicates that the value corresponding to this address is not resolved and similarly, when " $Stat_v = 0$ ", this indicates that the value corresponding to this address is resolved. The values of the address index from 1 to 5 represents the addresses of the primary inputs as shown in  $Data\_in\_g$  and the  $L_L$  and  $L_S$  corresponding to these values are the first applied test vectors "0111" as shown in figure 8 and consequently the corresponding  $Stat_v$  values are filled by zeros.

This is the main idea of how the data is stored when processing the gate-level data structure. When a CUT is entirely processed and the primary outputs of the circuit are resolved all the values of  $Stat_v$  become "0".

Table 6: The main storing and indexing arrays needed to process C17 circuit gate-level

Data inout g					
Data in g	Data out g	Address Index	Stat <sub>v</sub>	L <sub>L</sub>	L <sub>S</sub>
1	10	1	0	0	7
2	11	2	0	1	7
3		3	0	1	7
4		4	0	1	7
5		5	0	1	7
		6	1	0	0
		7	1	0	0
		8	1	0	0
		9	1	0	0
		10	1	0	0
		11	1	0	0

## 3.5. Data Structure of Netlist Switch-Level (netlist\_s)

### 3.5.1 Coding Architecture of netlist\_s

The main idea of the coding structure of netlist switch-level is similar to the coding structure of netlist gate-level. The switch-level of CUT is composed of funP, funN and funC representing respectively the switches P, N and the function of the connection node. These functions were previously explained in chapter 2. The capacitance is represented by  $Cap_v$  (i.e. Capacitance value) and  $Cap_s$  represents the updated capacitance value at the node. Table 7 uses these notations and is organized as follows:

- **Function Code:** Represents the codes of the functions in column two.
- **Function:** Represents the main functions of the switch level funN, funP, funC and (funC + Caps). Function (funC + Caps) performs two operations as follows:
  - **1<sup>st</sup> operation:** Resolves the connection node by applying funC on the input (in) and the previous value of the capacitance  $Cap_v$  (i.e. Capacitance value). The resolved output is represented by OutC.
  - **2<sup>nd</sup> operation:** Updates the capacitance value  $Cap_v$  by  $Cap_s$  (i.e. Capacitance store) value. The functionality of  $Cap_s$  is based on the explanation provided on the capacitance in section 2.2.4. Figure 11 shows a flow diagram of (funC +  $Cap_s$ ) function where OutC is resolved by funC and the  $Cap_v$  is updated by  $Cap_s$
- **Data 1:** Represents the output of the function (i.e Drain for funP and funN)
- **Data 2, 3:** Represents the inputs of the functions (i.e. the inputs funP, funN and funC).

Table 7: Coding of the switch-level functions

Function Code	Function	Data 1	Data 2	Data3
1	funP	Drain	Gate	Source
2	funN	Drain	Gate	Source
3	funC	OutC	in <sub>1</sub>	in <sub>2</sub>
4	funC + Cap <sub>s</sub>	OutC	in	Cap <sub>v</sub>

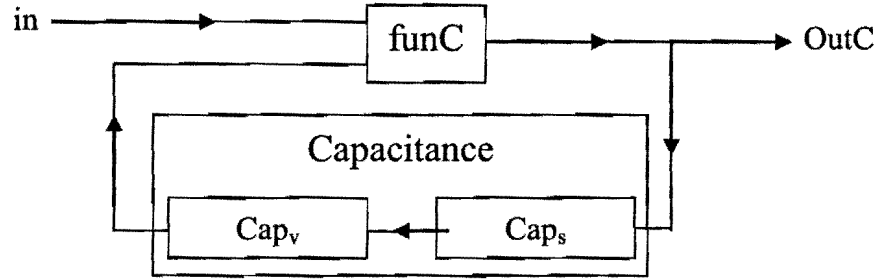


Figure 11: How the function code “4” processes the data

A structure based on this coding architecture is created for every single logic gate of table 4 (i.e. Gates type). Org<sub>m</sub> block shown previously will build a switch-level netlist based on these structures for any combinational circuit. The following sections of this chapter focus on providing detailed information on the logic gates switch-level coding structure.

### 3.5.2. NOT Gate

Figure 12 shows the switch-level of NOT gate. Table 8 shows the coding and data structure of switch-level NOT gate. The main concept is to affect addresses to the links created for the switch-level (i.e. D(Drain), G(Gate), S(Source), OutC, Cap<sub>v</sub>, VDD and GND). These links represent the basic elements of the switch-level structure. The equation (EQ – 3 – 2) evaluates the incrementation start address for a logic gate.

$$\text{Add}_{\text{inc1}} = [ \text{Number of input(s) of the logic gate} + 1 ] \quad (\text{EQ} - 3 - 2)$$

$\text{Add}_{\text{inc1}}$  represents the incrementation start address. The NOT gate shown in table 8 represents the organization of the data structure and the results of the coding of these structures. The numbers of “Coded Data” represent the addresses for Data1, Data2 and Data3. The “Function” column is similar to the structure created for gate-level and it is used to identify the operators. (i.e. funN, funP, funC and (funC + Caps)). The “Coded Data” and “Data structure” are organized as follows:

- **(in)** and **(Out<sub>g</sub>)** represent respectively the input and output of NOT gate. The addresses **(1)** and **(2)** are respectively allocated to these elements.
- **VDD** and **GND** (i.e. Supply and ground) get respectively the addresses **(3)** and **(4)** as the incrementation of the addresses starts from  $\text{Add}_{\text{inc1}} = 2$ . The remainder incrementation of the addresses of the links of the switch-level of this logic gate will start from  $\text{Add}_{\text{inc2}}$  as shown in (EQ – 3 – 3).  $\text{Add}_{\text{inc2}}$  represents the starting incrementation address for the remainder elements of this logic gate.

$$\text{Add}_{\text{inc2}} = \text{Add}_{\text{inc1}} + 2 = 4 \quad (\text{EQ} - 3 - 3)$$

- The remainder elements of this table are coded as follows:
  - **D[1], D[2]**: Represents respectively the drains of P and N switches numbered by order of priority access. The order is important because of the dependency of the switches to resolve the output of the logic gate. As the program processes the information in sequence, it is important to have these functions in certain order for coherent results when resolving the output of the logic gate.

- OutC [1]: Represents the output of funC function and it resolves the incident signals D[1] and D[2].
- $Cap_v$  : The capacitance value at the node.
- The *Asterix* sign followed by a number ( i.e. \* number ) has a particular meaning as it helps construct the addresses order. For example the notation ( \*1 D[1] ) indicates that the corresponding elements are increasingly numbered at these locations. The repeating elements in the table will simply keep their numbers. Once these numbers are completed, they will be assigned to the coded data table ( i.e. [( \* number ) + Add<sub>inc2</sub> ] ) starting from top to bottom and left to right. Based upon this data structure table, the program will be able to build the “Coded Data” table.

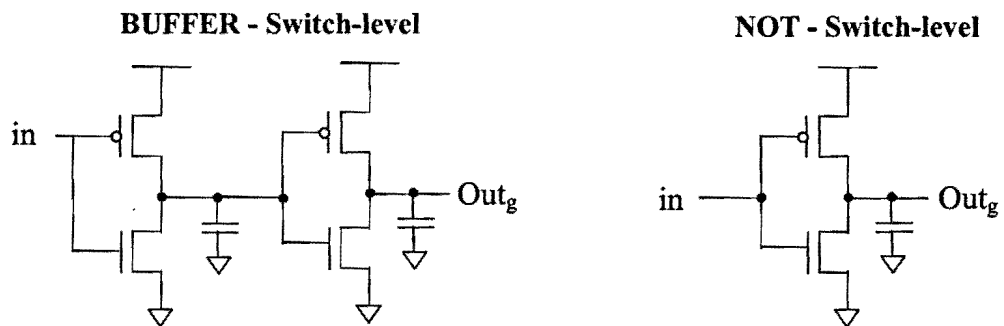


Figure 12: Switch-Level of BUFFER and NOT



Table 8: Coding and data structure of switch-level NOT gate

NOT - Switch-level				
Index	Function	Data 1	Data 2	Data3
1	funP	*1 D[1]	in	VDD
2	funN	*2 D[2]	in	GND
3	funC	*3 OutC[1]	1 D [1]	2 D [2]
4	funC + Cap <sub>s</sub>	Out <sub>g</sub>	3 OutC[1]	*4 Cap <sub>v</sub> [1]

Structure

Data

1	1	5	1	3
2	2	6	1	4
3	3	7	5	6
4	4	2	7	8

Coded

Data

### 3.5.3. BUFFER Gate

Figure 12 shows the switch-level of BUFFER gate. Table 9 shows the coding and data structure of switch-level NOT gate. This gate is coded similarly to NOT gate.

Table 9: Coding and data structure of switch-level BUFFER gate

BUFFER - Switch-level				
Index	Function	Data 1	Data 2	Data3
1	funP	*1 D[1]	in	VDD
2	funN	*2 D[2]	in	GND
3	funC	*3 OutC[1]	1 D [1]	2 D [2]
4	funC + Cap <sub>s</sub>	*4 OutC[2]	3 OutC[1]	*5 Cap <sub>v</sub> [1]
5	funP	*6 D[3]	4 OutC[2]	VDD
6	funN	*7 D[4]	4 OutC[2]	GND
7	funC	*8 OutC[3]	6 D [3]	7 D [4]
8	funC + Cap <sub>s</sub>	Out <sub>g</sub>	8 OutC[3]	*9 Cap <sub>v</sub> [2]

Data Structure

1	1	5	1	3
2	2	6	1	4
3	3	7	5	6
4	4	8	7	9
5	1	10	8	3
6	2	11	8	4
7	3	12	10	11
8	4	2	12	13

Coded Data

### 3.5.4. NAND2 – AND2

Table 10 shows the data structure of switch-level NAND2 – AND2 gates. Table 11 shows the coded data of switch-level of NAND2 – AND2 gates. Figure 13 shows the switch-level of NAND2 – AND2 gates. These gates are coded in the same manner as NOT except minor changes. The reason these two gates are coded on the same table is because the only difference between NAND2 and AND2 is a NOT gate. The notation on the table indicates that starting from index 7, the two gates NAND2 and AND2 have different structures.

Furthermore, the notation S [1] is added to the data structure and it is used as input and output depending on the used function. This notation is added for logic gates that have switches in series (i.e. A drain of a switch close to the ground will become the source of the next switch).

Table 10: Data structure of switch-level NAND2 and AND2 gates

[ NAND2 - AND2 ] - Switch-level				
Index	Function	Data 1	Data 2	Data3
1	funP	*1 D[1]	in <sub>1</sub>	VDD
2	funP	*2 D[2]	in <sub>2</sub>	VDD
3	funC	*3 OutC[1]	1 D [1]	2 D [2]
4	funN	*4 D[3]	in <sub>2</sub>	GND
5	funC + Cap <sub>s</sub>	*5 S[1]	4 D [4]	*6 Cap <sub>v</sub> [1]
6	funN	*7 D[4]	in <sub>1</sub>	5 S[1]
7	funC	*8 OutC[2]	3 OutC[1]	7 D[6]
8	funC + Cap <sub>s</sub>	*9 OutC[3]	8 OutC[2]	*10 Cap <sub>v</sub> [2]
9	funP	*11 D[5]	9 OutC[3]	VDD
10	funN	*12 D[6]	9 OutC[3]	GND
11	funC	*13 OutC[4]	11 D [5]	12 D [6]
12	funC + Cap <sub>s</sub>	Out <sub>g</sub>	13 OutC[4]	*14 Cap <sub>v</sub> [3]
8	funC + Cap <sub>s</sub>	Out <sub>g</sub>	8 OutC[2]	*9 Cap <sub>v</sub> [2]

AND2

NAND2

Data Structure

Table 11: Coded data of switch-level NAND2 and AND2 gates

AND2 - Switch-level					NAND2 - Switch-level			
Index	Function Code	Data			Function Code	Data		
		1	2	2		1	2	2
1	1	6	1	4	1	6	1	4
2	1	7	2	4	1	7	2	4
3	3	8	6	7	3	8	6	7
4	2	9	2	5	2	9	2	5
5	4	10	9	11	4	10	9	11
6	2	12	1	10	2	12	1	10
7	3	13	8	12	3	13	8	12
8	4	14	13	15	4	3	13	14
9	1	16	14	4				
10	2	17	14	5				
11	3	18	16	17				
12	4	3	18	19				

Coded Data

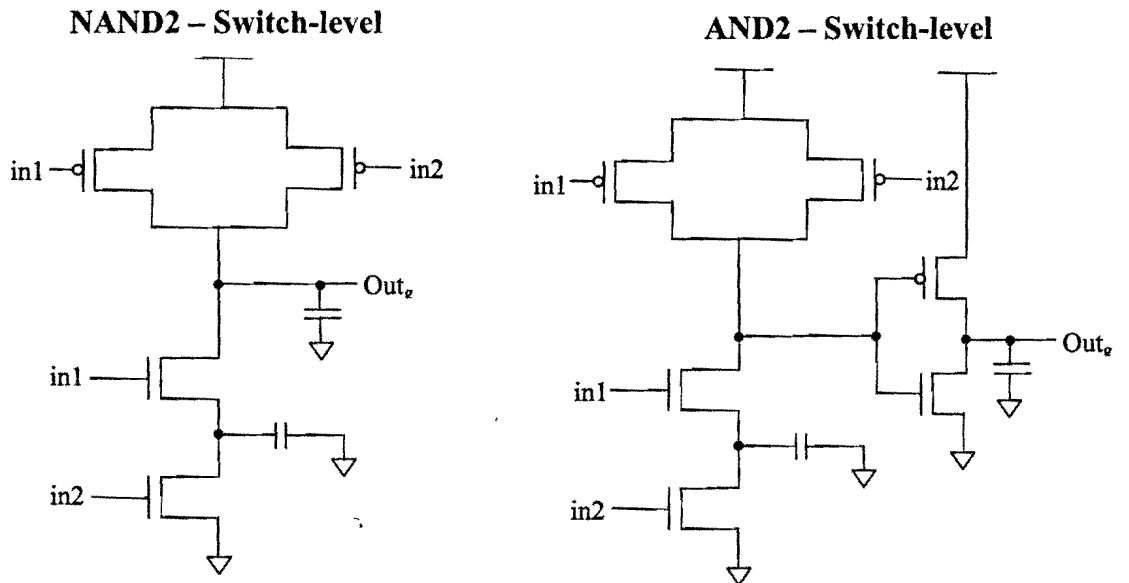


Figure 13: Switch-Level of NAND2 and AND2

### 3.5.5. NAND3 – AND3

Table 12 shows the data structure of switch-level NAND3 – AND3 gates. Table 13 shows the coded data of switch-level of NAND3 – AND3 gates. Figure 14 shows the switch-level of NAND3 – AND3 gates. These gates are coded in the same manner as NAND2 – AND2.

Table 12: Data structure of switch-level NAND3 and AND3 gates

[ NAND3 - AND3 ] - Switch-level				
Index	Function	Data 1	Data 2	Data3
1	funP	*1 D[1]	in <sub>1</sub>	VDD
2	funP	*2 D[2]	in <sub>2</sub>	VDD
3	funP	*3 D[3]	in <sub>3</sub>	VDD
4	funC	*4 OutC[1]	1 D [1]	2 D [2]
5	funC	*5 OutC[2]	3 D[3]	4 OutC[1]
6	funN	*6 D[4]	in <sub>3</sub>	GND
7	funC + Cap <sub>s</sub>	*7 S[1]	6 D [4]	*8 Cap <sub>v</sub> [1]
8	funN	*9 D[5]	in <sub>2</sub>	7 S[1]
9	funC + Cap <sub>s</sub>	* 10 S[2]	9 D [5]	* 11 Cap <sub>v</sub> [2]
10	funN	*12 D[6]	in <sub>1</sub>	10 S[2]
11	funC	*13 OutC[3]	5 OutC[2]	12 D[6]
12	funC + Cap <sub>s</sub>	*14 OutC[4]	13 OutC[3]	*15 Cap <sub>v</sub> [3]
13	funP	*16 D[7]	14 OutC[4]	VDD
14	funN	*17 D[8]	14 OutC[4]	GND
15	funC	*18 OutC[5]	16 D [7]	17 D [8]
16	funC + Cap <sub>s</sub>	Out <sub>g</sub>	18 OutC[5]	*19 Cap <sub>v</sub> [4]
12	funC + Cap <sub>s</sub>	Out <sub>g</sub>	13 OutC[3]	*14 Cap <sub>v</sub> [3]

Data Structure

AND3

NAND3

Table 13: Coded data of switch-level NAND3 and AND3 gates

AND3 - Switch-level					NAND3 - Switch-level				
Index	Function	Data 1	Data 2	Data 2	Function	Code	Data 1	Data 2	Data 2
	Code								
1	1	7	1	5	1	7	1	5	Coded Data
2	1	8	2	5	1	8	2	5	
3	1	9	3	5	1	9	3	5	
4	3	10	7	8	3	10	7	8	
5	3	11	9	10	3	11	9	10	
6	2	12	3	6	2	12	3	6	
7	4	13	12	14	4	13	12	14	
8	2	15	2	13	2	15	2	13	
9	4	16	15	17	4	16	15	17	
10	2	18	1	16	2	18	1	16	
11	3	19	11	18	3	19	11	18	
12	4	20	19	21	4	19	19	20	
13	1	22	20	5	4	4	19	20	
14	2	23	20	6					
15	3	24	22	23					
16	4	4	24	25					

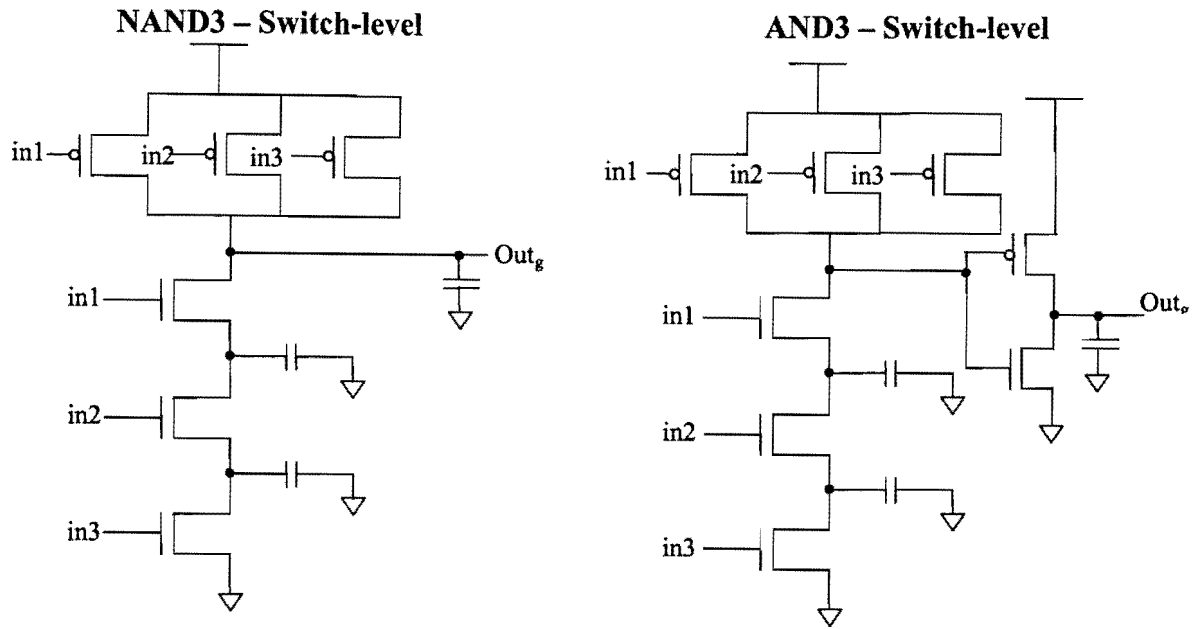


Figure 14: Switch-Level of NAND3 and AND3

### 3.5.6. NOR2 – OR2

Table 14 shows the data structure of switch-level NOR2 – OR2 gates. Table 15 shows the coded data of switch-level of NOR2 – OR2 gates. Figure 15 shows the switch-level of NOR2 – OR2 gates. These gates are coded in the same manner as NAND2 – AND2.

Table 14: Data structure of switch-level NOR2 and OR2 gates

[ NOR2 - OR2 ] - Switch-level				
Index	Function	Data 1	Data 2	Data3
1	funN	*1 D[1]	in <sub>1</sub>	GND
2	funN	*2 D[2]	in <sub>2</sub>	GND
3	funC	*3 OutC[1]	1 D [1]	2 D [2]
4	funP	*4 D[3]	in <sub>1</sub>	VDD
5	funC + Cap <sub>s</sub>	*5 S[1]	4 D [4]	*6 Cap <sub>v</sub> [1]
6	funP	*7 D[4]	in <sub>2</sub>	5 S[1]
7	funC	*8 OutC[2]	3 OutC[1]	7 D[6]
8	funC + Cap <sub>s</sub>	*9 OutC[3]	8 OutC[2]	*10 Cap <sub>v</sub> [2]
9	funP	*11 D[5]	9 OutC[3]	VDD
10	funN	*12 D[6]	9 OutC[3]	GND
11	funC	*13 OutC[4]	11 D [5]	12 D [6]
12	funC + Cap <sub>s</sub>	Out <sub>g</sub>	13 OutC[4]	*14 Cap <sub>v</sub> [3]

8	funC + Cap <sub>s</sub>	Out <sub>g</sub>	8 OutC[2]	*9 Cap <sub>v</sub> [2]	NOR2
---	-------------------------	------------------	-----------	-------------------------	------

8	funC + Cap <sub>s</sub>	Out <sub>g</sub>	8 OutC[2]	*9 Cap <sub>v</sub> [2]	OR2
---	-------------------------	------------------	-----------	-------------------------	-----

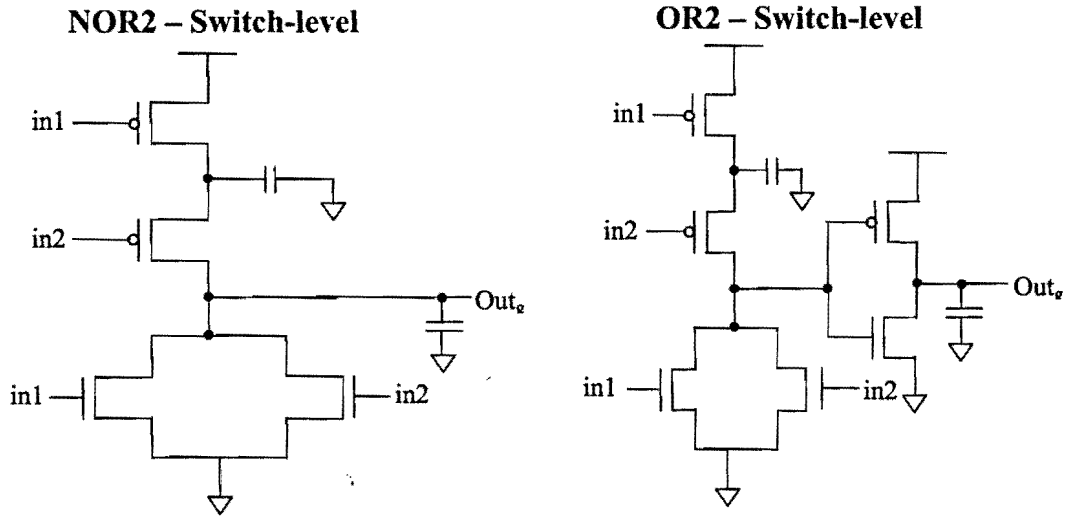


Figure 15: Switch-Level of NOR2 and OR2



Table 15: Coded data of switch-level NOR2 and OR2 gates

OR2 - Switch-level					NOR2 - Switch-level			
Index	Function Code	Data			Function Code	Data		
		1	2	2		1	2	2
1	2	6	1	5	2	6	1	5
2	2	7	2	5	2	7	2	5
3	3	8	6	7	3	8	6	7
4	1	9	1	4	1	9	1	4
5	4	10	9	11	4	10	9	11
6	1	12	2	10	1	12	2	10
7	3	13	8	12	3	13	8	12
8	4	14	13	15	4	14	13	15
9	1	16	14	4	1	16	14	4
10	2	17	14	5	2	17	14	5
11	3	18	16	17	3	18	16	17
12	4	3	18	19	4	3	18	19

Coded Data

### 3.5.7. NOR3 – OR3

Table 16 shows data structure of switch-level NOR3 – OR3 gates. Table 17 shows the coded data of switch-level of NOR3 – OR3 gates. Figure 16 shows the switch-level of NOR3 – OR3 gates. These gates are coded in the same manner as NAND2 – AND2.

Table 16: Data structure of switch-level NOR3 and OR3 gates

[ NOR3 - OR3 ] - Switch-level				
Index	Function	Data 1	Data 2	Data3
1	funN	*1 D[1]	in <sub>1</sub>	GND
2	funN	*2 D[2]	in <sub>2</sub>	GND
3	funN	*3 D[3]	in <sub>3</sub>	GND
4	funC	*4 OutC[1]	1 D [1]	2 D [2]
5	funC	*5 OutC[2]	3 D[3]	4 OutC[1]
6	funP	*6 D[4]	in <sub>1</sub>	VDD
7	funC + Cap <sub>s</sub>	*7 S[1]	6 D [4]	*8 Cap <sub>v</sub> [1]
8	funP	*9 D[5]	in <sub>2</sub>	7 S[1]
9	funC + Cap <sub>s</sub>	* 10 S[2]	9 D [5]	* 11 Cap <sub>v</sub> [2]
10	funP	*12 D[6]	in <sub>3</sub>	10 S[2]
11	funC	*13 OutC[3]	5 OutC[2]	12 D[6]
12	funC + Cap <sub>s</sub>	*14 OutC[4]	13 OutC[3]	*15 Cap <sub>v</sub> [3]
13	funP	*16 D[7]	14 OutC[4]	VDD
14	funN	*17 D[8]	14 OutC[4]	GND
15	funC	*18 OutC[5]	16 D [7]	17 D [8]
16	funC + Cap <sub>s</sub>	Out <sub>g</sub>	18 OutC[5]	*19 Cap <sub>v</sub> [4]
12	funC + Cap <sub>s</sub>	Out <sub>g</sub>	13 OutC[3]	*14 Cap <sub>v</sub> [3]

OR3

NOR3

Data Structure

Table 17: Coded data of switch-level NOR3 and OR3 gates

OR3 - Switch-level					NOR3 - Switch-level			
Index	Function Code	Data			Function Code	Data		
		1	2	2		1	2	2
1	2	7	1	6	2	7	1	6
2	2	8	2	6	2	8	2	6
3	2	9	3	6	2	9	3	6
4	3	10	7	8	3	10	7	8
5	3	11	9	10	3	11	9	10
6	1	12	1	5	1	12	1	5
7	4	13	12	14	4	13	12	14
8	1	15	2	13	1	15	2	13
9	4	16	15	17	4	16	15	17
10	1	18	3	16	1	18	3	16
11	3	19	11	18	3	19	11	18
12	4	20	19	21	4	4	19	20
13	1	22	20	5				
14	2	23	20	6				
15	3	24	22	23				
16	4	4	24	25				

Coded Data

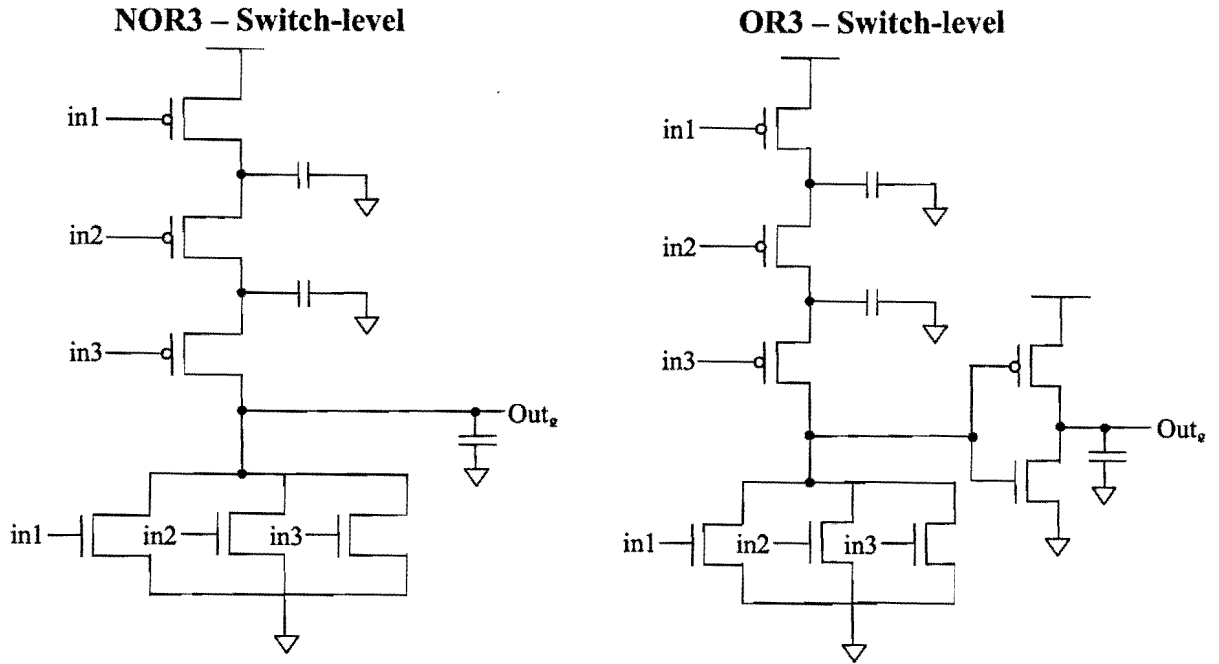


Figure 16: Switch-Level of NOR3 and OR3

### 3.5.8 Data Structure Switch-Level of C17 Circuit

Table 18 shows how C17 circuit is coded in switch-level. This circuit has 6 NAND2 gates. The coding of switch-level is based on incrementing the addresses of the gate level structure. The coding of the gates is slightly different. The following shows how C17 circuit is coded:

- Gate Type: Represents the logic gate types of the circuit (i.e. Number 11 indicates that the corresponding coded logic gate is NAND2 gate starting from address index 1 to address index 8). The zeros indicate simply that the corresponding switch-level codes are for the last logic gate type number. The zeros are user default values to fill the Gate Type arrays. It is simply for programming purposes.
- First NAND2 gate [Address Index = [1:8]]:

$$\text{Add}_{\text{inc1}} (\text{C17}) = [\text{Total Number of Gates} + \text{Primary inputs}] = 11 \quad (\text{EQ} - 3 - 4)$$

$$\text{Add}_{\text{inc2}} = \text{Add}_{\text{inc1}} + 2 = 13 \quad [ 2: \text{Added addresses for VDD and GND} ] \quad (\text{EQ} - 3 - 5)$$

$$\text{Inc}_a = \text{Add}_{\text{inc2}} + 1 = 14 \quad (\text{EQ} - 3 - 6)$$

$\text{Inc}_a$ : The first incrementation address of the links of the switch-level data structure

- Second NAND2 gate:

$$\text{Add}_{\text{cg}} = 22 \quad [ \text{The maximum address of the previous logic gate} ]$$

$$\text{Inc}_a = \text{Add}_{\text{inc2}} + 1 = 23$$

$\text{Add}_{\text{cg}}$ : Represents the last address of the last link corresponding to the previous logic gate.

- The coding of the remaining logic gates is similar to “Second NAND2 gate” until all the logic gates are coded in switch-level.

The program can process any combinational circuit and build the netlist switch-level coded data structure based on the same structure explained in this chapter.

Table 18: Data structure of netlist switch-level of C17 circuit

Netlist_s ( C17 )					
Arrya Index	Function Code	Data 1	Data 2	Data 3	Gate Type
1	1	14	1	12	11
2	1	15	3	12	0
3	3	16	14	15	0
4	2	17	3	13	0
5	4	18	17	19	0
6	2	20	1	18	0
7	3	21	16	20	0
8	4	6	21	22	0
.....					
8	1	23	3	12	11
9	1	24	4	12	0
10	3	25	23	24	0
11	2	26	4	13	0
12	4	27	26	28	0
13	2	29	3	27	0
14	3	30	25	29	0
15	4	7	30	31	0
.....					
41	1	59	8	12	11
42	1	60	9	12	0
43	3	61	59	60	0
44	2	62	9	13	0
45	4	63	62	64	0
46	2	65	8	63	0
47	3	66	61	65	0
48	4	11	66	67	0

### 3.6. Processing and Storing Arrays of Switch-Level

The processing arrays used to store the values of the processed links of the switch-level circuit are stored in an array called: `Data_inout_s`. This array structure is similar to gate-level `Data_inout_g`. Table 19 shows `Data_inout_s` of C17 circuit. This array is in 3D and it is built as follows:

- $Cap_e$ : Indicates that the value of the corresponding address index is a capacitance.

$$Cap_e = 1 \quad [\text{This address index corresponds to a capacitance}]$$

$$Cap_e = 0 \quad [\text{This address index does not correspond to a capacitance}]$$

- $Stat_v$ : Indicates that the corresponding value at this address index is resolved or calculated and stored
- $L_L, L_S$ : As explained in section 3.4. These are respectively the logic and strength codes of a link. This represents the value at the corresponding link.
- There are 12 arrays forming `Data_inout_s`. Every array corresponds to a test vector. There are 12 test vectors. The capacitance values are calculated and stored in `Data_inout_s` for every test vector. When a transient is injected, the previous values of the capacitances is ready to apply ( $funC + Cap_s$ ) at the node.

Table 19: Array of the storing processed data of C17 circuit switch-level

Data inouts (C17)				
Address Index	Cap <sub>e</sub>	Stat <sub>v</sub>	L <sub>L</sub>	S <sub>L</sub>
1	0	1	0	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	0
5	0	1	0	0
6	0	1	0	0
7	0	1	0	0
8	0	1	0	0
9	0	1	0	0
10	0	1	0	0
11	0	1	0	0
12	0	1	0	0
13	0	1	0	0
14	0	1	0	0
15	0	1	0	0
16	0	1	0	0
17	0	1	0	0
18	0	1	0	0
19	1	0	0	1
20	0	1	0	0
21	0	1	0	0
22	1	0	1	1
.....				
59	0	1	0	0
60	0	1	0	0
61	0	1	0	0
62	0	1	0	0
63	0	1	0	0
64	1	0	0	1
65	0	1	0	0
66	0	1	0	0
67	1	0	0	1



## Chapter 4

### Detailed Representation of SimV<sub>1</sub> Simulator

#### 4.1. Flow diagram of SimV<sub>1</sub>

Figure 17 shows the full flow diagram of SimV<sub>1</sub>. The different parameters and blocks in this flow diagram are as follows:

- **var<sub>f</sub>**: Represents the variable of injected transient types, this number varies from 1 to nF (i.e. nF = 23 is the maximum number of injected transient types)
- **var<sub>l</sub>**: Represents the variable of the locations of transient injection, this number varies from 1 to nL (i.e. nL is the maximum number of locations of transient injection, for C17 circuit “nL = 24” ).
- **var<sub>v</sub>**: Represents the variable of index of the used test vectors, this number varies from 1 to nV (i.e. nV represents the maximum number of applied test vectors, for C17 circuit “nV = 24”)
- **init<sub>f</sub>, init<sub>l</sub>, init<sub>v</sub>**: Initialize respectively var<sub>f</sub>, var<sub>l</sub> and var<sub>v</sub> to “1” when they are called.
- **init<sub>all</sub>**: Initialize the blocks init<sub>f</sub>, init<sub>l</sub> and init<sub>v</sub> when it is called.

- $\text{det}(\text{var\_f}, \text{var\_l})$  ,  $\text{detATV}(\text{var\_f}, \text{var\_l})$ : Represents the coefficients of the profiling mapping arrays  $\text{Prof}_m$  (i.e.  $\text{map\_det}$  and  $\text{map\_detATV}$ ) explained in section 3.2.3. Their values are updated based on the input information (i.e.  $\text{var\_f}$ ,  $\text{var\_l}$  and  $\text{var\_v}$ ) as shown in their corresponding blocks in the flow diagram.
- Module1: Represents the heart of the simulator. This module receives at its inputs the transient types, the transient injection locations and the test vector. The output of this module is either 1/0 indicating detection or no detection based on the input parameters. The simulator can use the drain, gate and the inputs of a logic gate as transient injection location type. This chapter will focus on the drain only. A gate or inputs of a logic gate locations types are coded in similar manner to a drain as transient location type. The next section is dedicated for this module.

The program activates  $\text{init}_{all}$  and the variables  $\text{var\_f}$ ,  $\text{var\_l}$  and  $\text{var\_v}$  are initialized then Module 1 will process these values for CUT and the results of the simulation will be  $\text{Comp} = 1/0$ . There are two options:

- $\text{Comp} = 1$ : Signifies that the injected transient is detected. The coefficient arrays take the following values:  $\text{det}(1,1)=1$  and  $\text{detATV}(1,1)=1$ . This indicates that the first test vector detected the first transient type at the first location.  $\text{Prof}_m$  (i.e.  $\text{map\_det}$  and  $\text{map\_detATV}$ ) maps are updated by  $\text{det}$  and  $\text{detATV}$  values. The cycle is ended by initializing  $\text{var\_f}$ ,  $\text{var\_l}$  and  $\text{var\_v}$ . This concept allows the simulation to run faster as the simulator will stop testing the circuit for the remaining test vectors when a transient injection is detected. Module 1 will restart to process the next set of values of  $\text{var\_f}$ ,  $\text{var\_l}$  and  $\text{var\_v}$  until all  $\text{var\_f}$  and  $\text{var\_l}$  values are processed.

- $\text{Comp} = 0$ : Signifies that the injected transient is not detected. The array coefficients are not updated until detection of transient injection. When all the test vectors are applied and “ $\text{Comp} = 0$ ”, the coefficient arrays will become:  $\text{det}(1,1)=0$  and  $\text{detATV}(1,1)=nV$ . This indicates that none of the test vectors detected the first transient type at the first location. The  $\text{Prof}_m$  (i.e.  $\text{map\_det}$  and  $\text{map\_detATV}$ ) maps are updated by  $\text{det}$  and  $\text{detATV}$  values. The cycle is ended by initializing  $\text{var\_f}$ ,  $\text{var\_l}$  and  $\text{var\_v}$ .

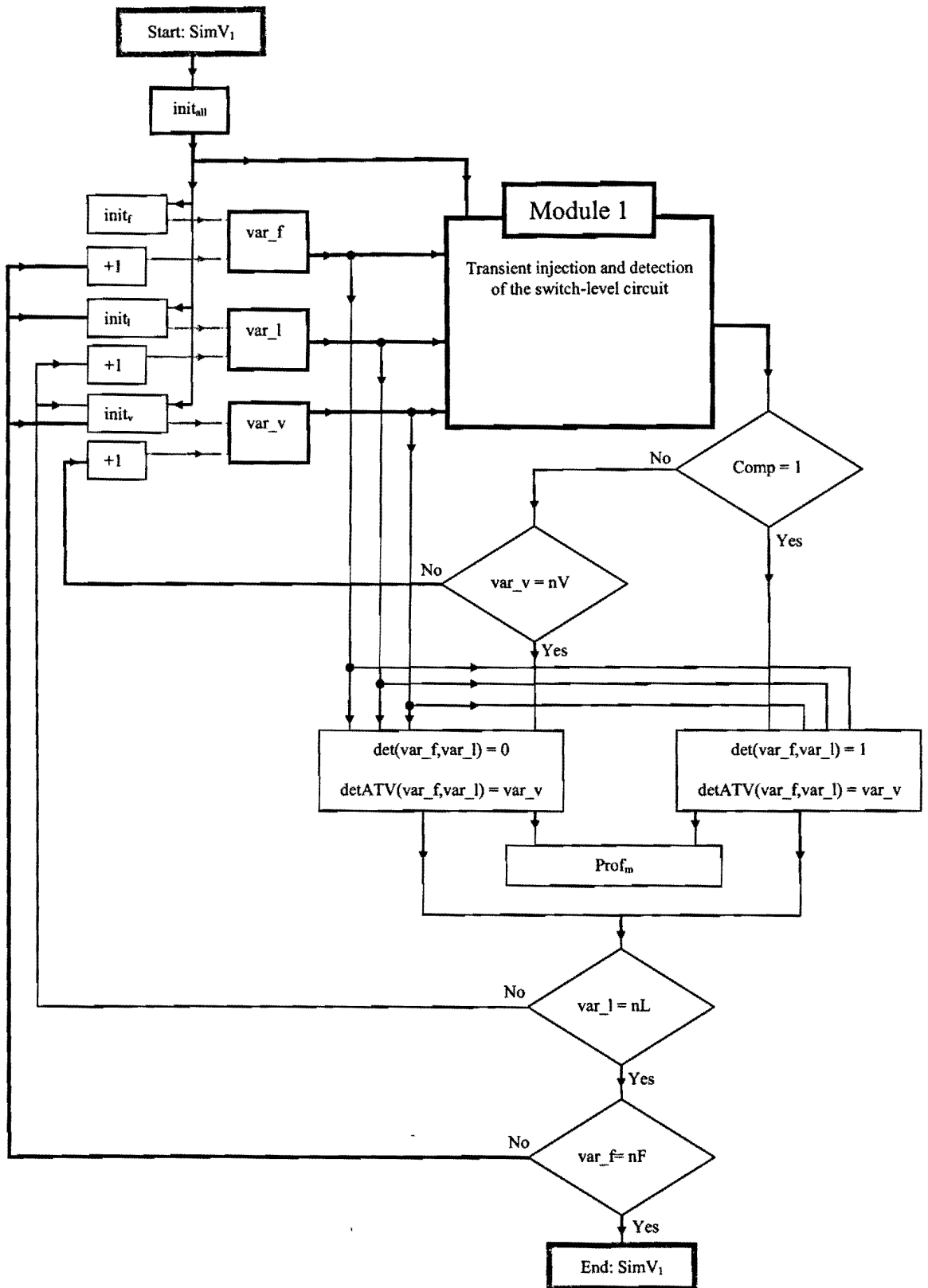


Figure 17: Flow diagram of SimV<sub>1</sub> simulator

## 4.2. Flow Diagram of Module1

This module is the main processing unit of the simulator SimV<sub>1</sub>. Its inputs are var<sub>f</sub>, var<sub>l</sub>, var<sub>v</sub> and init<sub>all</sub>. These variables represent respectively the transient type variable, the transient injection location variable, the index of the test vector variable and the initialization block of all these variables at the start of the program. The output of this module is either 1/0 indicating whether the injected transient is detected or not.

Figure 18 shows the flow diagram of this module and its different blocks and its functionality are explained in detail as follows:

- At the start of the simulation, init<sub>all</sub> initializes init<sub>ns</sub>. This block initializes var<sub>ns</sub>, block B1 and B2:
  - var<sub>ns</sub>: Represents the index of a row of netlist<sub>s</sub> array. When this variable is initialized, its value is set to 1.
  - Block B1: Updates Data<sub>inout\_s</sub> array by storing the test vector indexed by var<sub>v</sub> and updates the capacitances values corresponding to the processed test vector. Section 3.6 explains the structure of Data<sub>inout\_s</sub> and shows an example of this array for C17 circuit. When this block is initialized, a new var<sub>v</sub> value is loaded and Data<sub>inout\_s</sub> is updated.
  - Block B2: Updates Array<sub>injD</sub> based on the transient injection location var<sub>l</sub>.

Array<sub>injD</sub> is shown in table 20. This table shows an example of NAND3 logic gate. There are 6 locations for transient injection in this example as shown in Array<sub>loc</sub> where this array shows the addresses of possible transient injection locations in

netlist\_s. The program at every value of var\_ns will check if at the specified array location var\_1, the corresponding array [ ArrayinjD (var\_1(var\_ns)) = 1 ] is true. Then the transient can be injected at the corresponding address in Data\_inout\_s. The remainder of the flow diagram will provide further explanation of this process of transient injection and detection.

- Block B3, B4 and B5 Operate as follows: For the specified var\_ns value, the corresponding netlist\_s(var\_ns) row is selected to be processed. var\_ns determines the row to be accessed when B4 is called. The function of the selected row is then resolved and the output of this row is stored in Data\_inout\_s.
  - The remainder of the process is organized as follows: Array<sub>injD</sub>(var\_1(var\_ns)) determines the transient injection location based on var\_1 location. The corresponding array of var\_1 as shown in figure 20, has zeros except at one index address. There are two options:
    - Array<sub>injD</sub>(var\_1(var\_ns)) = 1: Data\_inout\_s is updated by the transient value (var\_f) at the corresponding address location. There are 2 options based on this solution:
      - var\_ns ≠ nS: var\_ns is incremented by 1. This indicates that the next row of netlist\_s is going to be resolved. This process will be in loop until all the rows of netlist\_s are resolved.
- nS: Represents the last row of netlist\_s.
- var\_ns = nS: The primary outputs of CUT are resolved. Block B7 is then called to compare the results of the primary outputs with and without the transient injection. Detection and no detection status are represented

respectively by the codes 1 and 0. After this stage,  $\text{init}_{\text{ns}}$  is called to restart the same process with a new set of  $\text{var}_v$ ,  $\text{var}_l$  and  $\text{var}_f$

- $\text{Array}_{\text{injD}}(\text{var}_l(\text{var}_{\text{ns}})) = 0$ :  $\text{Data\_inout\_s}$  of block B6 is bypassed. The remainder of the flow diagram is the same as the previous condition:
  - $\text{var}_{\text{ns}} \neq \text{nS}$ :  $\text{netlis\_s}$  is incremented (loop)
  - $\text{ns} = \text{nS}$ : B7 is called,  $\text{init}_{\text{ns}}$  is initialized and a new set of  $\text{var}_v$ ,  $\text{var}_l$  and  $\text{var}_f$  are processed.

The functionality of  $\text{SimV}_1$  explained in this chapter gives an idea of how this simulator is implemented. The different results of execution time, fault coverage and other statistical results based on the simulation are presented in Chapter 6.

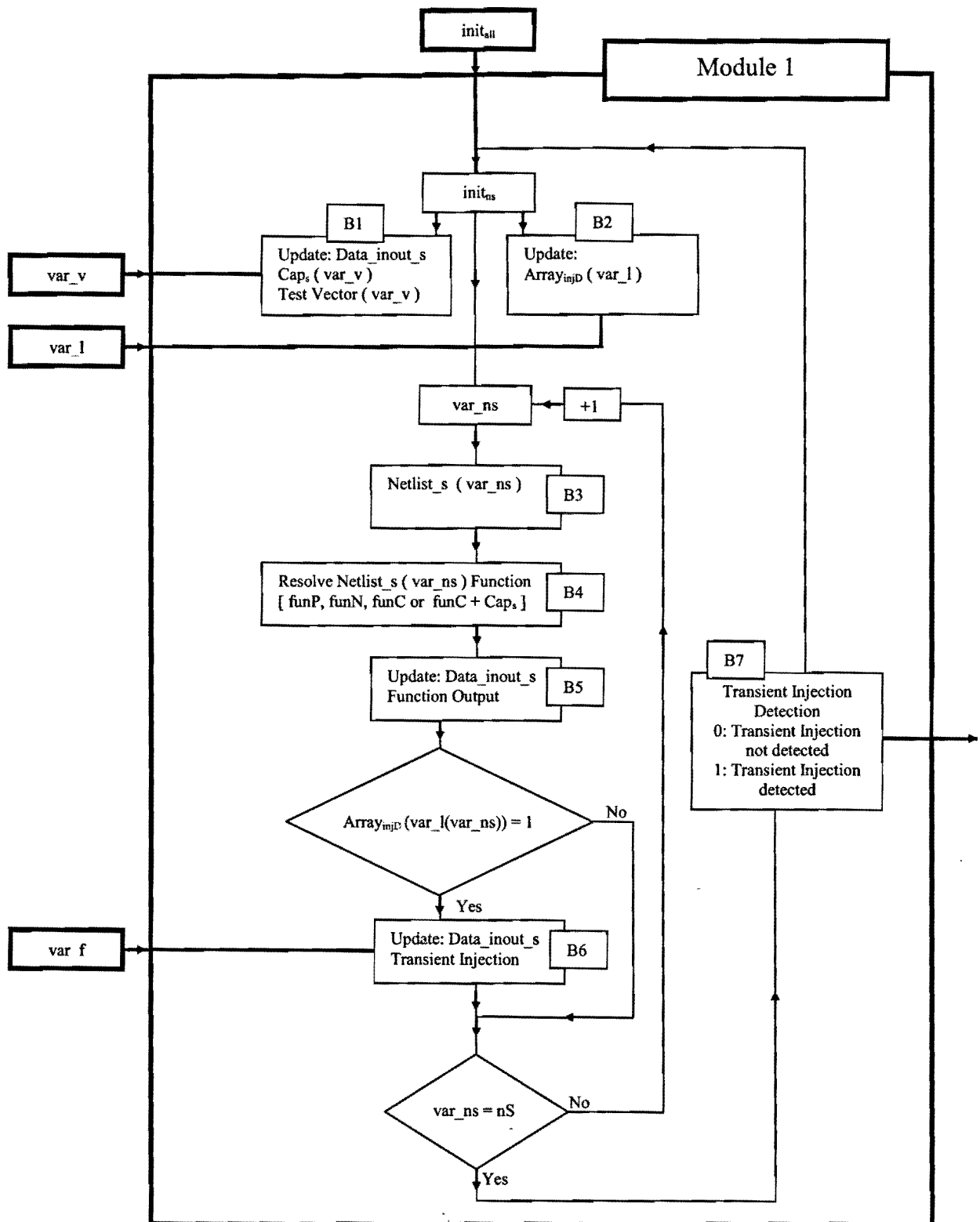


Figure 18: Flow diagram of the transient injection and detection **Module 1**



Table 20: Architecture of Array<sub>injD</sub> drain location array applied to NAND3

NAND3 - Switch-level						
Array <sub>loc</sub>	Index	Function Code	Function	Data 1	Data 2	Data2
			Code			
1	1	funP	1	7	1	5
2	2	funP	1	8	2	5
3	3	funP	1	9	3	5
6	4	funC	3	10	7	8
8	5	funC	3	11	9	10
10	6	funN	2	12	3	6
	7	funC + Caps	4	13	12	14
	8	funN	2	15	2	13
	9	funC + Caps	4	16	15	17
	10	funN	2	18	1	16
	11	funC	3	19	11	18
	12	funC + Caps	4	4	19	20

Coded Data

Array <sub>injD</sub> ( var <sub>1</sub> ( var <sub>ns</sub> ) )						
var <sub>1</sub>	1	2	3	4	5	6
Index [ var <sub>ns</sub> ]	1	0	0	0	0	0
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	1	0	0
7	0	0	0	0	0	0
8	0	0	0	0	1	0
9	0	0	0	0	0	0
10	0	0	0	0	0	1
11	0	0	0	0	0	0
12	0	0	0	0	0	0

## **Chapter 5**

### **Detailed Representation of SimV<sub>2</sub> Simulator**

SimV<sub>2</sub> is an optimized version of SimV<sub>1</sub> simulator. An algorithm is designed for SimV<sub>2</sub> to speed up its run time. This simulator is able to achieve a significant speedup in comparison to SimV<sub>1</sub>. This chapter explains in detail the different modules of this simulator and the concept of the developed algorithm that allows the simulator to run faster than SimV<sub>1</sub> simulator. Section 5.1 and 5.2 gives detailed explanation of two types of data structures needed for the simulation. Section 5.3 explains the concept of the developed algorithm by providing a step by step simplified explanation based on C17 circuit. SimV<sub>1</sub> is part of SimV<sub>2</sub> simulator and it is used only one time in order to build the switch-level data for SimV<sub>2</sub> simulator.

#### **5.1. Gate-Level Resolution Function**

This function is based on collecting detailed information on the logic gates of CUT. All the data processed at this level are in gate-level. The collected information is the main key for the applied algorithm. Table 20 shows the type of information collected through this stage. This

table shows an example of the first NAND2 logic gate of C17 circuit. The different parameters and the data structure of this array are explained as follows:

- $Logic_c$ : Signifies that the corresponding array collects the logic combination at the inputs of the corresponding logic gate. At every applied test vector at the primary inputs of C17 circuit, this array will store the logic combination of the inputs of the corresponding logic gate.
- $Gate_1$ : Represents the first gate of C17 circuit.
- $in_1$  to  $in_{10}$ : Represents the possible inputs of a gate. For the NAND2 gate there are only 2 inputs, thus the first two inputs  $in_1$  and  $in_2$  are used and the other inputs are left blank.
- $in_1$  and  $in_2$ : Stores the logic combination of the inputs of NAND2 gate for the 12 applied test vectors at the primary inputs of C17 circuit.
- Array Row Index: Represents the index of the applied test vectors.
- $In_G$ : Represents: [Decimal representation of logic combination + 1]. This combination is shown in  $in_1$  and  $in_2$ .
- $Out_G$ : Represents the logic value at the output of NAND2 gate based on the logic combination at the input of this gate.
- $Out_{GF}$ : Represents  $NOT(Out_G)$ .
- $Stat_{po}$ : Represents the status of the primary output of C17 circuit for the corresponding test vector when the output of NAND2 gate is flipped for the same applied test vector. This value can be 1 or 0 corresponding to detection or no detection due to the flipped bit at the output of this NAND2 gate.
- $Multi_{InG}$ : Represents the multiplication of  $In_G$  by  $Stat_{po}$ .

Figure 19 shows C17 circuit and the structure of the 3D arrays based on table 20 arrays. These arrays are generated by processing C17 circuit in gate-level and by applying the gate-level resolution function. There are different features implemented into the program that helped build these arrays faster. The use of gate level allows applying the test vectors as a set of bits. All the test vectors are applied at the same time to the circuit. Another feature that helped speed this process is called progressive gates. The following technique helped improve the speed of resolving the primary outputs of the circuit and builds Logic<sub>c</sub> arrays. This technique is based on reducing the number of processed gates by one every time a new Logic<sub>c</sub> array is built. For C17 circuit, the number of times these logic gates are processed is shown as follows:

- Reduction of number of processed logic gate is applied:

$$N_{pg} = \frac{n_g(n_g+1)}{2} + n_g = \frac{6(6+1)}{2} + 6 = 27 \quad (\text{EQ} - 5 - 1)$$

$$N_{pg} = 6 + 6 + 5 + 4 + 3 + 2 + 1 = 27$$

$N_{pg}$ : Represents the number of progressive logic gates to be processed in order to build all the Logic<sub>c</sub> arrays of CUT. The first “6” (i.e.  $n_g = 6$ ) logic gates are used as a reference solution in the program. The remainder of the logic gates is processed in a progressive manner.

- Reduction of number of processed gate is not applied:

$$N_{pg} = n_g^2 = 6^2 = 36 \quad (\text{EQ} - 5 - 2)$$

There is a significant difference between the two calculations and this difference becomes more important for bigger circuits.

The main idea behind this process is based on how the resolved values of the outputs of the logic gates are stored in Data\_inout\_g array. When the first gate is resolved, its output is stored in Data\_inout\_g and Logic<sub>c</sub> array of Gate 1 is created. For Gate 2, the only information that has to be changed is the flipping bit at the output of Gate 2 in order to resolve Logic<sub>c</sub> array of Gate 2. This does not affect Gate 1, thus Gate 1 information is permanently stored in Data\_inout\_g and it is used to resolve the primary outputs of the circuit and build Logic<sub>c</sub> array of Gate 2. The same concept is applied for the remainder of the logic gates and every time a new Logic<sub>c</sub> array is built, the number of processed gates is reduced by one.

Table 22 shows the needed information of Logic<sub>c</sub> arrays to be used for SimV<sub>2</sub> simulator. This table shows the data structure of collected data gate-level based on the resolution function applied to C17 circuit.

Table 21: Example of gate-level resolution function applied to NAND2 gate: Gate<sub>1</sub> of C17 circuit covering the effect of the flipping bit of the output of the logic gate on the circuit for a set of 12 test vectors

Logic <sub>C</sub> - NAND2 gate : Gate1															
Array Column Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Array Row Index	in <sub>1</sub>	in <sub>2</sub>	in <sub>3</sub>	in <sub>4</sub>	in <sub>5</sub>	in <sub>6</sub>	in <sub>7</sub>	in <sub>8</sub>	in <sub>9</sub>	in <sub>10</sub>	In <sub>G</sub>	Out <sub>G</sub>	Out <sub>GF</sub>	Stat <sub>PO</sub>	Multi <sub>SinG</sub>
1	0	1									2	1	0	1	2
2	0	0									1	1	0	1	1
3	0	1									2	1	0	0	0
4	1	0									3	1	0	1	3
5	0	1									2	1	0	1	2
6	1	1									4	0	1	1	4
7	0	1									2	1	0	1	2
8	1	0									3	1	0	1	3
9	1	0									3	1	0	1	3
10	1	0									3	1	0	1	3
11	0	0									1	1	0	1	1
12	0	0									1	1	0	0	0
1 <sup>st</sup> Stage														2 <sup>nd</sup> Stage	

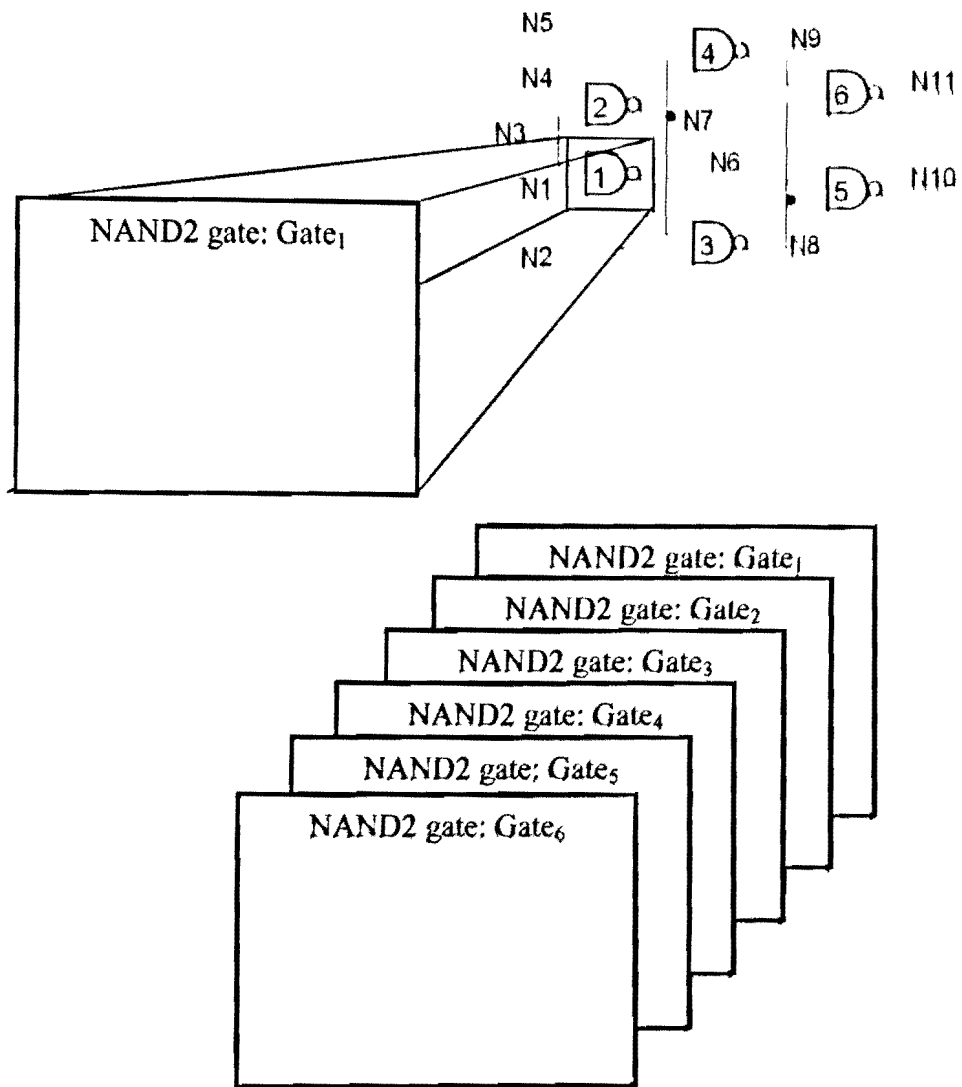


Figure 19: Structure of the 3D array of gate-level resolution function applied to C17 circuit

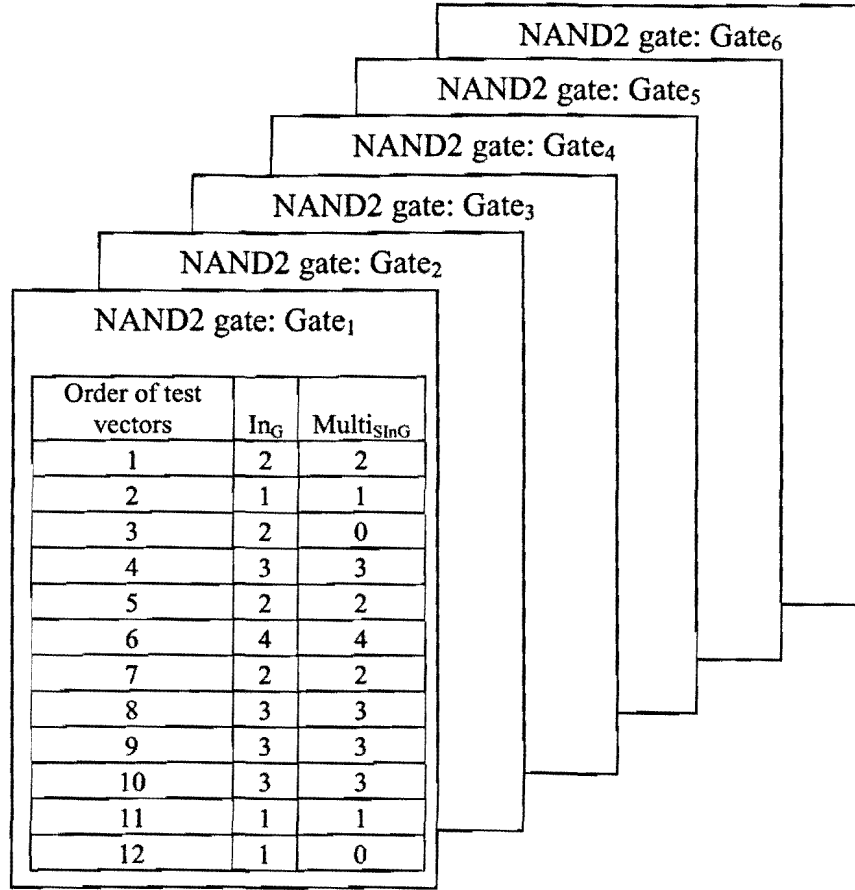


Table 22: Data structure of collected data gate-level based on the resolution function applied to C17 circuit

## 5.2. Switch-Level Data Input Organization

This data is built based on SimV<sub>1</sub>\* simulator. All the logic gate types are processed through SimV<sub>1</sub> simulator for transient injection and detection at switch-level. Every logic gate is processed as an independent circuit. This simulator is slightly different from SimV<sub>1</sub> as when a fault is detected, the simulator will keep running until all the test vectors are used. The test vectors applied to these logic gates are exhaustive test vectors and their numbers can be 2 test vectors for NOT or BUFFER gates, and 1024 test vectors for NAND10 or OR10 gates. SimV<sub>1</sub>\* will run all the test vectors and injects the transients at all locations at every logic gate. The results of processing these gates are stored in 3D Prof<sub>m</sub>\* arrays. These arrays are similar to Prof<sub>m</sub>



arrays built through SimV<sub>1</sub> simulator at the exception of little modification based on how the data is stored. These arrays are 3D arrays as every array slice corresponds to the index of the applied test vector TV<sub>n</sub> (i.e. similar notation as var\_v). It is the same analogy for map\_detATV\* and map\_det\*, the only difference with these maps and map\_det and map\_ATVdet are as follows:

- map\_detATV\* : Stores the output value of the logic gate for every injected transient and every transient injection location. There is an array created for every TV<sub>n</sub>.
- map\_det\*: Stores the status of the output of the logic gate as 1/0 corresponding to detection or no detection. There is an array created for every TV<sub>n</sub>. This map is similar to map\_det

Table 23 shows 2 slices of Prof<sub>m</sub>\* arrays corresponding to 2 test vectors. These arrays are based on applying 8 transient types. A NOT gate has only 2 transient injection locations, therefore, T<sub>Loc</sub> = [1:2]. This table shows map\_det of NOT gate as well. This array can be built from map\_det\*: 1 and 2. map\_det = [ map\_det\*: 1 ] OR [ map\_det\*: 2].

map\_det of the logic gate is stored as it is needed for SimV<sub>2</sub> as well. Every logic gate will generate 3 array types: map\_det, map\_det\* and map\_det\*. Figure 20 shows the structure of Prof<sub>m</sub>\* arrays for different logic gate types. Based on NOT logic gate example, T<sub>Loc</sub> = [1:2]. There are two slices of arrays for map\_det\* and map\_detATV\* as TV<sub>n</sub> = 2. Every logic gate has one array of map\_det. For NAND3 logic gate example, T<sub>Loc</sub> = [1:6]. There are 8 slices of arrays for map\_det\* and map\_detATV\* as TV<sub>n</sub> = 8. These 3D arrays are all stored in a \*.mat format file and are used in SimV<sub>2</sub> simulator.

NOT – TV <sub>n</sub> : 2									
		T <sub>Loc</sub>				T <sub>Loc</sub>			
		1	2			1	2		
T <sub>L</sub>	T <sub>S</sub>								
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
1	1	0	3	0	1	0	1	0	1
3	1	0	3	0	1	0	1	0	1
1	7	0	1	3	3	1	1	0	1
3	7	0	0	0	0	0	0	0	0
0	6	0	1	0	1	0	1	0	1
1	6	0	0	0	0	0	0	0	0
map_detATV* : 2		map_det* : 2							

NOT – TV <sub>n</sub> : 1									
		T <sub>Loc</sub>				T <sub>Loc</sub>			
		1	2			1	2		
T <sub>L</sub>	T <sub>S</sub>								
0	0	1	1	0	0	0	0	0	0
0	1	3	1	1	0	1	0	1	0
1	1	1	1	0	0	0	0	0	0
3	1	3	1	1	0	1	0	1	0
1	7	0	3	1	1	1	1	1	1
3	7	3	3	1	1	1	1	1	1
0	6	0	1	1	0	1	0	1	0
1	6	1	1	0	0	0	0	0	0
map_detATV* : 1		map_det* : 1							

NOT Gate			
		T <sub>Loc</sub>	
		1	2
T <sub>L</sub>	T <sub>S</sub>		
0	0	0	0
0	1	1	0
1	1	0	1
3	1	1	1
1	7	1	1
3	7	1	1
0	6	1	0
1	6	0	1
map_det			

Table 23: Structure of the collected data of NOT gate applied to SimV<sub>1</sub>\* simulator

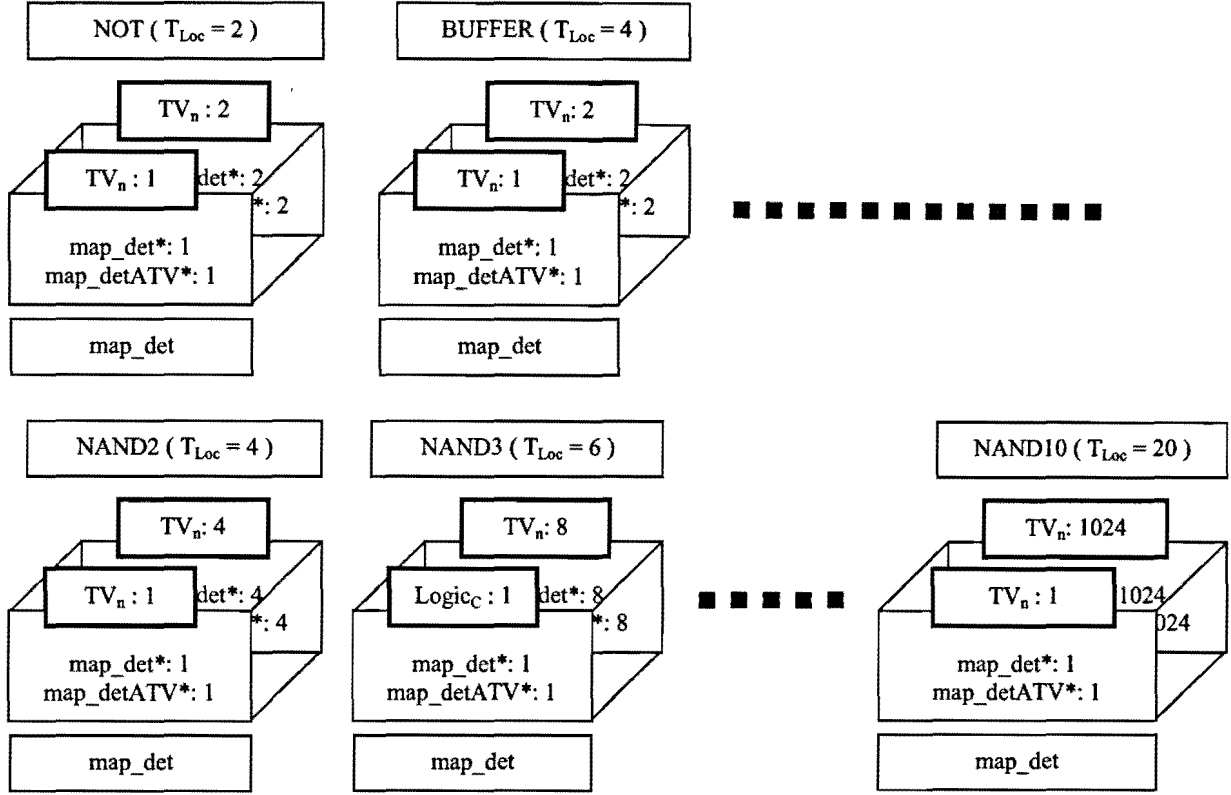


Figure 20: Data structure of some logic gates extracted from SimV1\* simulator

### 5.3. Concept of the Algorithm Used in SimV2 Simulator

This simulator uses the data generated from the gate-level resolution function mentioned in section 5.1 and the switch-level 3D array switch-level logic gates mentioned in section 5.2. This algorithm is able to trace back the information of transient injection and detection and build the same profiling maps (i.e. map\_det and map\_detATV) generated by SimV1. The example of C17 circuit is used in this section showing how this concept is implemented in order to build map\_det and map\_detATV.

Table 24 shows the different arrays used to resolve C17 circuit. This example shows how 4 different transients are injected into CUT and how they are detected in case of detection. Sections 5.3.1 to 5.3.4 give detailed explanation of these 4 transients. The steps of how the

transient detection is tracked are marked by different contrasts in the table for each one of these 4 transients. This allows better solution traceability and provides a clear explanation on how these map\_det and map\_detATV are built. C17 circuit has 6 NAND2 logic gates. This example traces 4 solutions for [ NAND2 - Gate<sub>1</sub> ] and it is the first logic gate of C17 circuit. The table [ NAND2 - Gate<sub>1</sub> ] shows one of the arrays shown in table 22. [ NAND2 Gate ] table shows four arrays of map\_det\* and map\_detATV\* and one array of map\_det. Each of these arrays has the same information on T<sub>Loc</sub> (i.e. T<sub>Loc</sub> = [1:4] corresponding to 4 locations). [ map\_detATV C17 ] table shows T<sub>Loc</sub> = [1:24] and the contrasted positions on this table represent the solutions found by applying SimV<sub>2</sub>. The transient locations type for this example is drain of a switch and only 8 transient types are used. The simulator can run all the transient types. Based on the experiment results, some transients can be collapsed and be represented by one transient type, therefore these 8 transient types represents the 23 different transient types. The following sub-sections present four sample of transient injection at four drain locations of C17 circuit. The steps explained in these sub-sections provide detailed explanation on how the program resolves the profiling maps Prof<sub>m</sub> of C17 circuit.

### 5.3.1 1<sup>st</sup> Solution Based on C17 Circuit Example

T<sub>Loc</sub>: The location of the transient = 1.

Array address of the transient type is 2, corresponding to T<sub>L</sub> = 0, T<sub>S</sub> = 1.

For all the steps when referred to map\_det, map\_detATV, map\_det and map\_detATV, all these arrays are accessed at the same T<sub>Loc</sub> and index of transient type.

- Step 1: Check map\_det of [ NAND2 ] table

- If the corresponding value is 0, then go to step 4.

Solution:  $\text{map\_detATV} = nV$ ,  $\text{map\_det} = 0$ .

- If the corresponding value is 1, then go to Step 2.
- Step 2 :Check the first  $\text{map\_det}^*$  of [ NAND2 Gate ] table that contains [1] as value
  - 1<sup>st</sup> [1] value is at  $\text{map\_det}^*: 2$ . This means that  $\text{TV}_n = 2$ .
  - The corresponding array location for [  $\text{map\_detATV}^*: 2$  ] array is value [3]
  - Solutions are:  $\text{map\_det}^*: 2 : [1]$  ,  $\text{map\_detATV}^*: 2 : [3]$
  - Go to step 3.

- Step 3: Check [ NAND2 – Gate<sub>1</sub> ] table.

- If  $\text{map\_detATV}^*$  solution is [3] then check the columns:

[ Order of test vectors ] and [  $\text{In}_G$  ]

- Find the first  $\text{In}_G$  value where:  $\text{In}_G = 2$
- Record the value of [ Order of test vectors ] corresponding to the first

[  $\text{In}_G = 2$  ] where 2 corresponds to  $\text{TV}_n = 2$ . In this case:

Order of test vectors = 1

- If  $\text{map\_detATV}^*$  solution is [0/1] then check the columns:

[ Order of test vectors ] and [  $\text{Multi}_{\text{In}_G}$  ]

- Step 4: Final solution

$\text{map\_det}$  (Array address of the transient type,  $T_{\text{Loc}}$ )

map\_detATV (Array address of the transient type,  $T_{Loc}$ )

Solution for C17: map\_detATV (2,1) = 1 ( At  $T_{Loc} = 1$  ,  $T_L = 0$  ,  $T_S = 1$  )

Solution for C17: map\_det (2,1) = 1 ( At  $T_{Loc} = 2$  ,  $T_L = 0$  ,  $T_S = 1$  )

These solutions are shown as well in table 25.

### 5.3.2. 2<sup>nd</sup> solution of based on C17 circuit example

$T_{Loc}$ : The location of the transient = 2.

Array address of the transient type is 1, corresponding to  $T_L = 0$  ,  $T_S = 0$ .

- Step 1: Check map\_det of [ NAND2 ] table

The corresponding value is 1. Go to Step 2.

- Step 2 :Check the first map\_det\* of [ NAND2 Gate ] table that contains [1] as value
  - 1<sup>st</sup> [1] value is at map\_det\*: 3. This means that  $TV_n = 3$ .
  - The corresponding array location for [ map\_detATV\*: 3 ] array is value [3]
  - Solutions are: map\_det\*: 3 : [1] , map\_detATV\*: 3 : [3]
  - Go to step 3.
- Step 3: Check [ NAND2 – Gate<sub>1</sub> ] table.
  - If map\_detATV\* solution is [3] then check the columns:  
[ Order of test vectors ] and [  $In_G$  ]
    - Find the first  $In_G$  value where:  $In_G = 3$

- Record the value of [ Order of test vectors ] corresponding to the first [ InG = 3 ] where 3 corresponds to  $TV_n = 3$ . In this case:

Order of test vectors = 4

- Go to step 4 for the final solution.

- Step 4: Final solution

map\_det (Array address of the transient type,  $T_{Loc}$ )

map\_detATV (Array address of the transient type,  $T_{Loc}$ )

Solution for C17: map\_detATV (1,2) = 4                      ( At  $T_{Loc} = 2$  ,  $T_L = 0$ ,  $T_S = 0$  )

Solution for C17: map\_det (1,2) = 1                      ( At  $T_{Loc} = 2$  ,  $T_L = 0$ ,  $T_S = 0$  )

These solutions are shown as well in table 25.

### 5.3.3. 3<sup>rd</sup> Solution Based on C17 Circuit Example

$T_{Loc}$ : The location of the transient = 3.

Array address of the transient type is 2, corresponding to  $T_L = 0$ ,  $T_S = 1$ .

- Step 1: Check map\_det of [ NAND2 ] table

If the corresponding value is 0, then go to step 4.

Solution: map\_detATV = nV = 12, map\_det = 0.

- Step 2 : Not applicable: N/A
- Step 3: Not applicable: N/A
- Step 4: Final solution

map\_det (Array address of the transient type,  $T_{Loc}$ )

map\_detATV (Array address of the transient type,  $T_{Loc}$ )

Solution for C17: map\_detATV (2,3) = 1 ( At  $T_{Loc} = 3$  ,  $T_L = 0$ ,  $T_S = 1$  )

Solution for C17: map\_det (2,3) = 1 ( At  $T_{Loc} = 3$  ,  $T_L = 0$ ,  $T_S = 1$  )

These solutions are shown as well in table 25.

### 5.3.4. 4<sup>th</sup> Solution Based on C17 Circuit Example

$T_{Loc}$ : The location of the transient = 4.

Array address of the transient type is 8, corresponding to  $T_L = 1$ ,  $T_S = 6$ .

- Step 1: Check map\_det of [ NAND2 ] table

If the corresponding value is 1, then go to Step 2.

- Step 2 :Check the first map\_det\* of [ NAND2 Gate ] table that contains [1] as value
  - 1<sup>st</sup> [1] value is at map\_det\*: 4. This means that  $TV_n = 4$ .
  - The corresponding array location for [ map\_detATV\*: 4 ] array is value [1]
  - Solutions are: map\_det\*: 4 : [1] , map\_detATV\*: 4 : [1]
  - Go to step 3.
- Step 3: Check [ NAND2 – Gate<sub>1</sub> ] table.

If map\_detATV\* solution is [0/1] then check the columns:

[ Order of test vectors ] and [ Multis<sub>inG</sub>]



- Find the first  $\text{Multi}_{\text{SInG}}$  value where:  $\text{Multi}_{\text{SInG}} = 4$
- Record the value of [ Order of test vectors ] corresponding to the first

[ $\text{Multi}_{\text{SInG}} = 4$ ] where 4 corresponds to  $\text{TV}_n = 4$ . In this case:

Order of test vectors = 6

- Step 4: Final solution

$\text{map\_det}$  (Array address of the transient type,  $T_{\text{Loc}}$ )

$\text{map\_detATV}$  (Array address of the transient type,  $T_{\text{Loc}}$ )

Solution for C17:  $\text{map\_detATV}(6,3) = 6$  ( At  $T_{\text{Loc}} = 3$  ,  $T_L = 1$ ,  $T_S = 6$  )

Solution for C17:  $\text{map\_det}(6,3) = 1$  ( At  $T_{\text{Loc}} = 3$  ,  $T_L = 1$ ,  $T_S = 6$  )

These solutions are shown as well in table 25.

Table 24: Example of how the algorithm applied to SimV<sub>2</sub> simulator builds the solution maps map\_det and map\_detATV for C17 circuit

NAND2 Gate																		
Array Address	T <sub>L</sub>	T <sub>S</sub>	T <sub>Loc</sub>				T <sub>Loc</sub>				T <sub>Loc</sub>				T <sub>Loc</sub>			
			1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	0	0	1	1	1	1	1	1	1	1	1	3	1	1	0	0	0	0
2	0	1	1	1	1	1	1	1	1	1	1	3	1	1	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	1	3	1	1	0	0	3	3
4	3	1	1	1	1	1	3	1	1	1	1	3	1	1	0	0	3	3
5	1	7	3	3	1	3	0	3	1	3	3	0	3	1	0	0	1	1
6	3	7	3	3	1	3	3	3	1	3	3	3	3	3	3	3	3	3
7	0	6	1	1	1	1	0	1	1	1	1	0	1	1	0	0	0	0
8	1	6	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
			map_detATV* : 1				map_detATV* : 2				map_detATV* : 3				map_detATV* : 4			


Array Address	T <sub>L</sub>	T <sub>S</sub>	T <sub>Loc</sub>				T <sub>Loc</sub>				T <sub>Loc</sub>				T <sub>Loc</sub>			
			1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
4	3	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1
5	1	7	1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1
6	3	7	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	6	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
8	1	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
			map det* : 1				map det* : 2				map det* : 3				map det* : 4			


Array Address	T <sub>L</sub>	T <sub>S</sub>	T <sub>Loc</sub>											
			1	2	3	4	.....	24	1	2	3	4	.....	24
1	0	0	4				.....		0	1	0	0	.....	
2	0	1		12			.....		1	0	0		.....	
3	1	1					.....		0	1	1	1	.....	
4	3	1					.....		1	1	1	1	.....	
5	1	7					.....		1	1	1	1	.....	
6	3	7					.....		1	1	1	1	.....	
7	0	6					.....		1	1	0	0	.....	
8	1	6				6	.....		0	0	1	1	.....	
			map_detATV C17											
			map_det NAND2											

NAND2 - Gate <sub>1</sub>		
Order of test vector	InG	Multisig
1		2
2	1	1
3	2	0
4	3	3
5	2	2
6	4	4
7	2	2
8	3	3
9	3	3
10	3	3
11	1	1
12	1	0

Table 25: map\_det and map\_detATV maps for C17 circuit

Array Address	TL	TS
1	0	0
2	0	1
3	1	1
4	3	1
5	1	7
6	3	7
7	0	6
8	1	6

T <sub>Loc</sub>						
1	2	3	4	.....	24	
12	<del>4</del>	12	12	.....	12	
	4	12	12	.....	12	
12	4	6	6	.....	1	
1	4	6	6	.....	1	
1	1	4	1	.....	1	
1	1	4	1	.....	1	
1	4	12	12	.....	12	
12	12	<del>6</del>	6	.....	1	
map_detATV C17						

T <sub>Loc</sub>						
1	2	3	4	.....	24	
0	<del>1</del>	0	0	.....	0	
	1	0	0	.....	0	
0	1	1	1	.....	1	
1	1	1	1	.....	1	
1	1	1	1	.....	1	
1	1	1	1	.....	1	
1	1	0	0	.....	0	
0	0	<del>1</del>	1	.....	1	
map_detATV C17						

Section 5.3.1 to section 5.3.4 gave detailed examples of 4 transient injections of different transient types and at different drain locations. These examples reflect the concept of the algorithm and how CUT is resolved to build the profiling maps Prof<sub>m</sub>.

## **Chapter 6**

### **Experimental Results**

The results of the experiments are based on ISCAS'85 benchmark circuits. Different experiments are presented in this chapter based on the simulation of transient injection at drain, gate and inputs of a logic gates. A performance analysis based on simulation execution time of SimV<sub>1</sub> and SimV<sub>2</sub> simulators is conducted.

#### **6.1. Overview of the Simulation Applied Data**

##### **6.1.1. Benchmarks**

Detailed information on ISCAS'85 benchmark circuits [17] is shown in table 26. The parameters of this table are as follows:

Gate Types: Represents the different logic gates of these circuits.

Number of Gates: Represents the number of gates for each Gate Type.

Table 26: Detailed logic gate information of ISCAS'85 benchmark circuits

C17	Gate Types	NAND2							
	Number of Gates	6							

C432	Gate Types	NOT	NAND2	NOR2	AND9	XOR2	NAND4	AND8	NAND3
	Number of Gates	40	64	19	3	18	14	1	1

C499	Gate Types	XOR2	AND2	NOT	AND4	OR4	AND5		
	Number of Gates	104	40	40	8	2	8		

C880	Gate Types	NAND4	AND3	NAND2	NAND3	AND2	OR2	NOT	NOR2
	Number of Gates	13	12	60	14	105	29	63	61

	Gate Types	BUFFER							
	Number of Gates	26							

C1908	Gate Types	NOT	NAND2	BUFFER	AND2	AND3	NAND4	NAND3	NAND8
	Number of Gates	277	347	162	30	12	2	1	3

	Gate Types	AND4	NAND5	AND5	AND8	NOR2			
	Number of Gates	2	24	16	3	1			

C2670	Gate Types	BUFFER	AND2	NOT	AND4	AND3	NAND2	OR2	OR4
	Number of Gates	272	203	321	11	112	254	51	22

	Gate Types	NOR2	AND5	OR3	OR5				
	Number of Gates	12	7	2	2				

C3540	Gate Types	BUFFER	NOT	OR2	AND2	NAND2	NAND3	AND3	NOR2
	Number of Gates	223	490	35	410	274	17	76	25

	Gate Types	AND4	NAND4	OR3	NOR3	AND5	NOR8	OR4	
	Number of Gates	10	7	56	27	2	16	1	

C5315	Gate Types	BUFFER	AND2	NOT	NAND2	AND4	OR2	AND3	OR3
	Number of Gates	313	319	581	454	27	95	359	50

	Gate Types	OR4	NOR2	AND5	OR5	NOR3	NOR4	AND9	
	Number of Gates	61	19	11	8	6	2	2	

C6288	Gate Types	AND2	NOT	NOR2					
	Number of Gates	256	32	2128					

## 6.1.2. Information on the Applied Data

Table 27 shows detailed information of ISCAS'85 benchmark circuits. The information extracted from these circuits is provided by the simulator.

The test vectors used for the simulations are based on compaction algorithm [18]. These test vectors are designed for logic-level circuits and are used for the first time in this work.

Table 27: Information on ISCAS'85 benchmark circuits

Benchmarks ISCAS'85	Number of primary inputs	Number of primary outputs	Number of Gates	Number of switches	Number of test vectors
C17	5	2	6	24	12
C432	36	7	160	896	27
C499	41	32	202	2180	52
C880	60	26	383	1802	16
C1908	33	25	880	3446	106
C2670	233	140	1269	5668	44
C3540	50	22	1669	7504	84
C5315	178	123	2307	11262	37
C6288	32	32	2416	14368	12
C7552	207	108	3513	15400	73

## 6.2. Fault Coverage Due to Transient Injection at Different Location types

This work is pioneer in terms of soft error detection based on switch-level models and there is no available data in the literature for comparison. The results in this work are based on the developed simulators.

### 6.2.1. Injection at Gate of a Switch

The results of simulation of transient injection at gate of a switch applied to C5315 circuit are shown in table 28. The simulator processing unit has the ability to show detailed results for different injected transient types. The various parameters of this table are explained as follows:

- $n_{T,d}$ : Represents the total number of injected transients for all transient locations and based on test vectors elimination (i.e. When an injected transient is detected, the simulator stops, records the information of the injected transient and proceeds with the next transient injection location). This method is applied to SimV<sub>1</sub> simulator. SimV<sub>2</sub> simulator has the ability to build the same information as it provides the same profiling maps as SimV<sub>1</sub>.  $n_{T,d}$  is determined by (EQ – 6 – 1). The parameters of this equation are explained in previous sections at the exception of  $var\_f$ . This parameter in this expression is constant and based on the calculation of one transient type.  $var\_f$  takes one value between 1 and 23 which is the injected transient type used for the corresponding calculation in this expression.

$$n_{T,d} = \sum_{var\_f = [1:23], var\_l = 1}^{nL} map\_det(var\_f, var\_l) \quad (EQ - 6 - 1)$$

- $n_{T,all}$ : Represents the total number of injected transients for all transient locations and based on all applied test vectors without test vector elimination (i.e. When an injected transient is detected, the simulator doesn't stop, and records the information of the injected transient and when the last test vector is applied then the simulator proceeds with the next transient injection location).  $n_{T,all}$  is determined by (EQ – 6 – 2) for one transient type. The parameters of this expression are explained in previous sections.

$$n_{T,d} = nV \cdot nL \quad (EQ - 6 - 2)$$

- $F_{cov}$ : Represents the fault coverage corresponding to every transient type, applied to all transient injection locations and based on the applied test vectors.
- $nL$ : Represents the number of transient injection locations. This number is the same as the number of switches of CUT as shown in Table 27.
- $nL_d$ : Represents the number of transient injection locations corresponding to detection. The  $F_{cov}$  can be determined based on  $nL$  and  $nL_d$  for every transient type.
- $n_{T,dSell}$ : Represents the number of injected transients for all transient locations and based on test vector elimination. This parameter is similar to  $n_{T,d}$ . This number applies only to the selected transient types. Table 28 shows that some transient types can be collapsed in one type as their results are equivalent. The equivalent transient types are differentiated in the table by different contrasts. Furthermore the 23 transient types based on the gate transient location type can be collapsed in 4 transient types resulting in a significant reduction of simulation execution time. The results of the remaining transient types are rebuilt based on these four selected transient equivalent types.
- Total: Represents the results based on all the transient types or the selected transient types such as for  $n_{T,dSell}$ .
- $T(Parameter)$ : Represents the execution time needed to process the parameter variable. (e.i.  $T(n_{T,d})$  represents the execution time to run  $n_{T,d}$ ). The simulation execution time is expressed in seconds.

Table 29 shows the fault coverage  $F_{cov}$  for all the ISCAS'85 benchmark circuits for injection at a gate of a switch.  $F_{cov}$  is shown for every single fault type. The total shows the fault coverage for all the applied fault types.



Table 28: Simulation results for injection at gate of a switch for C5315 circuit

Gate - C5315

$T_L$	$T_S$	$n_{Td}$	$n_{Tall}$	$F_{cov}$	$nL$	$nL_d$	$T(n_{Td})$	$T(n_{Tall})$	$n_{TdSel}$	$T(n_{TdSel})$
1	7	28521	416694	0.99991	11262	11261	28327.53	413867.42	28521	28327.53
0	0	17280	416694	1	11262	11262	17162.78	413867.42	17280	17162.78
0	1	60186	416694	0.99592	11262	11216	59777.74	413867.42	60186	59777.74
0	2	60186	416694	0.99592	11262	11216	59777.74	413867.42	0	0.00
0	3	60186	416694	0.99592	11262	11216	59777.74	413867.42	0	0.00
0	4	60186	416694	0.99592	11262	11216	59777.74	413867.42	0	0.00
0	5	60186	416694	0.99592	11262	11216	59777.74	413867.42	0	0.00
0	6	60186	416694	0.99592	11262	11216	59777.74	413867.42	0	0.00
1	0	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
1	1	86652	416694	0.95294	11262	10732	86064.21	413867.42	86652	86064.21
1	2	86652	416694	0.95294	11262	10732	86064.21	413867.42	0	0.00
1	3	86652	416694	0.95294	11262	10732	86064.21	413867.42	0	0.00
1	4	86652	416694	0.95294	11262	10732	86064.21	413867.42	0	0.00
1	5	86652	416694	0.95294	11262	10732	86064.21	413867.42	0	0.00
1	6	86652	416694	0.95294	11262	10732	86064.21	413867.42	0	0.00
3	0	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
3	1	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
3	2	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
3	3	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
3	4	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
3	5	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
3	6	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
3	7	17280	416694	1	11262	11262	17162.78	413867.42	0	0.00
<b>Total</b>		1082349	9583962	0.98665	259026	255569	1075007.06	9518950.74	192639	191332.26

Table 29: Simulation results of  $F_{cov}$  for transient injection at gate of a switch for ISCAS'85 benchmark circuits

Gate - $F_{cov}$											
$T_L$	$T_S$	C17	C432	C499	C880	C1908	C2670	C3540	C5315	C6288	C7552
1	7	1	1	0.9885	1	1	0.9915	0.9997	0.9999	0.3351	0.9928
0	0	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
0	1	1	0.9888	0.9404	0.9878	0.9997	0.9287	0.9884	0.9959	0.1640	0.9719
0	2	1	0.9888	0.9404	0.9878	0.9997	0.9287	0.9884	0.9959	0.1640	0.9719
0	3	1	0.9888	0.9404	0.9878	0.9997	0.9287	0.9884	0.9959	0.1640	0.9719
0	4	1	0.9888	0.9404	0.9878	0.9997	0.9287	0.9884	0.9959	0.1640	0.9719
0	5	1	0.9888	0.9404	0.9878	0.9997	0.9287	0.9884	0.9959	0.1640	0.9719
0	6	1	0.9888	0.9404	0.9878	0.9997	0.9287	0.9884	0.9959	0.1640	0.9719
1	0	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
1	1	0.9167	0.8214	0.8528	0.9417	0.9689	0.8705	0.9428	0.9529	0.2233	0.9373
1	2	0.9167	0.8214	0.8528	0.9417	0.9689	0.8705	0.9428	0.9529	0.2233	0.9373
1	3	0.9167	0.8214	0.8528	0.9417	0.9689	0.8705	0.9428	0.9529	0.2233	0.9373
1	4	0.9167	0.8214	0.8528	0.9417	0.9689	0.8705	0.9428	0.9529	0.2233	0.9373
1	5	0.9167	0.8214	0.8528	0.9417	0.9689	0.8705	0.9428	0.9529	0.2233	0.9373
1	6	0.9167	0.8214	0.8528	0.9417	0.9689	0.8705	0.9428	0.9529	0.2233	0.9373
3	0	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
3	1	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
3	2	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
3	3	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
3	4	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
3	5	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
3	6	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
3	7	1	1	1	1	1	0.9979	0.9999	1	0.7415	0.9971
Total		0.9783	0.9505	0.9455	0.9816	0.9918	0.9463	0.9820	0.9867	0.4380	0.9747

## 6.2.2. Injection at Drain of a Switch

Similarly the data of transient injection at drain of a switch is organized in a similar manner to transient injection at gate of a switch. Table 30 shows detailed results of C5315 for transient injection at drain of a switch. Table 31 shows the fault coverage  $F_{cov}$  for ISCAS'85 benchmark circuits for injection at drain of a switch

Table 30: Simulation results for transient injection at drain of a switch for C5315 circuit

Drain - C5315										
$T_L$	$T_S$	$n_{T,d}$	$n_{T,all}$	$F_{cov}$	$nL$	$nL_d$	$T(n_{T,d})$	$T(n_{T,all})$	$n_{TdSel}$	$T(n_{TdSel})$
1	7	33443	416694	1	11262	11261	13779.58	171691.18	33443	13779.58
0	0	367790	416694	0.14	11262	1625	151541.18	171691.18	367790	151541.18
0	1	236380	416694	0.5	11262	5623	97396.08	171691.18	236380	97396.08
0	2	256505	416694	0.49	11262	5573	105688.22	171691.18	256505	105688.22
0	3	256505	416694	0.49	11262	5573	105688.22	171691.18	0	0.00
0	4	256505	416694	0.49	11262	5573	105688.22	171691.18	0	0.00
0	5	256505	416694	0.49	11262	5573	105688.22	171691.18	0	0.00
0	6	256505	416694	0.49	11262	5573	105688.22	171691.18	0	0.00
1	0	367790	416694	0.14	11262	1625	151541.18	171691.18	0	0.00
1	1	185221	416694	0.64	11262	7253	76316.94	171691.18	185221	76316.94
1	2	251310	416694	0.5	11262	5627	103547.71	171691.18	251310	103547.71
1	3	251310	416694	0.5	11262	5627	103547.71	171691.18	0	0.00
1	4	251310	416694	0.5	11262	5627	103547.71	171691.18	0	0.00
1	5	251310	416694	0.5	11262	5627	103547.71	171691.18	0	0.00
1	6	251310	416694	0.5	11262	5627	103547.71	171691.18	0	0.00
3	0	367790	416694	0.14	11262	1625	151541.18	171691.18	0	0.00
3	1	53811	416694	1	11262	11251	22171.84	171691.18	53811	22171.84
3	2	53811	416694	1	11262	11251	22171.84	171691.18	0	0.00
3	3	53811	416694	1	11262	11251	22171.84	171691.18	0	0.00
3	4	53811	416694	1	11262	11251	22171.84	171691.18	0	0.00
3	5	53811	416694	1	11262	11251	22171.84	171691.18	0	0.00
3	6	53811	416694	1	11262	11251	22171.84	171691.18	0	0.00
3	7	13966	416694	1	11262	11262	5754.44	171691.18	13966	5754.44
Total		4434321	9583962	0.63	259026	163780	1827081.32	3948897.24	1398426	576196.00

Table 31: Simulation results of  $F_{cov}$  for transient injection at drain of a switch for ISCAS'85 benchmark circuits

Drain - $F_{cov}$											
$T_L$	$T_S$	C17	C432	C499	C880	C1908	C2670	C3540	C5315	C6288	C7552
1	7	1	1	0.98624	1	1	0.9435	0.9915	0.9999	0.5702	0.9915
0	0	0.25	0.1942	0.1633	0.1421	0.17847	0.1251	0.1349	0.1443	0.0178	0.1349
0	1	0.5	0.4587	0.46468	0.5	0.49942	0.4854	0.4899	0.4993	0.2037	0.4899
0	2	0.5	0.4554	0.44495	0.5	0.49739	0.3553	0.4603	0.4948	0.0035	0.4603
0	3	0.5	0.4554	0.44495	0.5	0.49739	0.3553	0.4603	0.4948	0.0035	0.4603
0	4	0.5	0.4554	0.44495	0.5	0.49739	0.3553	0.4603	0.4948	0.0035	0.4603
0	5	0.5	0.4554	0.44495	0.5	0.49739	0.3553	0.4603	0.4948	0.0035	0.4603
0	6	0.5	0.4554	0.44495	0.5	0.49739	0.3553	0.4603	0.4948	0.0035	0.4603
1	0	0.25	0.1942	0.1633	0.1421	0.17847	0.1251	0.1349	0.1443	0.0178	0.1349
1	1	0.75	0.6942	0.64037	0.6421	0.67847	0.6159	0.6264	0.644	0.1111	0.6264
1	2	0.5	0.4933	0.4578	0.5	0.49942	0.373	0.4706	0.4996	0.0003	0.4706
1	3	0.5	0.4933	0.4578	0.5	0.49942	0.373	0.4706	0.4996	0.0003	0.4706
1	4	0.5	0.4933	0.4578	0.5	0.49942	0.373	0.4706	0.4996	0.0003	0.4706
1	5	0.5	0.4933	0.4578	0.5	0.49942	0.373	0.4706	0.4996	0.0003	0.4706
1	6	0.5	0.4933	0.4578	0.5	0.49942	0.373	0.4706	0.4996	0.0003	0.4706
3	0	0.25	0.1942	0.1633	0.1421	0.17847	0.1251	0.1349	0.1443	0.0178	0.1349
3	1	1	0.9587	0.94174	1	0.99942	0.9762	0.9814	0.999	0.297	0.9814
3	2	1	0.9587	0.94174	1	0.99942	0.9762	0.9814	0.999	0.297	0.9814
3	3	1	0.9587	0.94174	1	0.99942	0.9762	0.9814	0.999	0.297	0.9814
3	4	1	0.9587	0.94174	1	0.99942	0.9762	0.9814	0.999	0.297	0.9814
3	5	1	0.9587	0.94174	1	0.99942	0.9762	0.9814	0.999	0.297	0.9814
3	6	1	0.9587	0.94174	1	0.99942	0.9762	0.9814	0.999	0.297	0.9814
3	7	1	1	1	1	1	0.9989	0.999	1	0.8854	0.999
<b>Total</b>		0.652	0.6187	0.59763	0.6334	0.63886	0.5616	0.6111	0.6323	0.1576	0.6111

### 6.2.3. Injection at Inputs of Logic Gate

The data for transient injection at inputs of a logic gate are based on a bit flip where the logic level is flipped at the inputs of a gate. The signals considered for these experiments are of a logic code 0 or 1 and strength code 7. The injected transient is  $T_L = 1$  and  $T_S = 7$  (i.e. When this transient is injected at an input of a logic gate and if the logic at this input is 0 then the logic will become 1 (Flipped) and similarly for logic 1, it becomes logic 0). This experiment will allow

comparing the fault coverage with transient injection at drain and gate of a switch. Basically at the location of the tested input of a logic gate, the logic level signal is flipped then the fault detection is determined based on the non-faulty and faulty information of the primary outputs of CUT. Table 32 shows detailed results of injection at inputs of a logic gate for C5315 circuit. Table 33 shows  $F_{cov}$  results for ISCAS'85 benchmark circuits.

Table 32: Simulation results for transient injection at inputs of a logic for C5315 circuit

Inputs - $F_{cov}$ - Bit Flip									
$T_L$	$T_S$	$n_{T,d}$	$n_{T,all}$	$F_{cov}$	$nL$	$nL_d$	$T(n_{T,d})$	$T(n_{T,all})$	$T(n_{T,dSel})$
1	7	20564	162282	0.99977	4386	4385	20553	162191	20552.5

Table 33: Simulation results of  $F_{cov}$  for transient injection at inputs of a logic gate for ISCAS'85 benchmark circuits

Inputs - $F_{cov}$ - Bit Flip											
$T_L$	$T_S$	C17	C432	C499	C880	C1908	C2670	C3540	C5315	C6288	C7552
1	7	1	1	1	1	1	0.7997	0.98226	0.999772	0.000833	0.9823

Figure 21 shows the results of  $F_{cov}$  for transient injection at drain, gate and inputs of a logic gate. The plot shows that for injection at gate of a switch and inputs of a logic gate, the fault coverage is very close, on the other hand,  $F_{cov}$  for injection at drain shows a significant difference compared to injection at drain of a switch and inputs of a logic gate. C6288 represents an exception in terms of fault coverage and does not follow the same trend as the remaining CUTs in this table. This circuit is a 240 full and half adder cells arranged in a 15x16 matrix. The number of fanouts in this circuit is very large and this makes the circuit redundant and less prone to transients as the results show in this table.

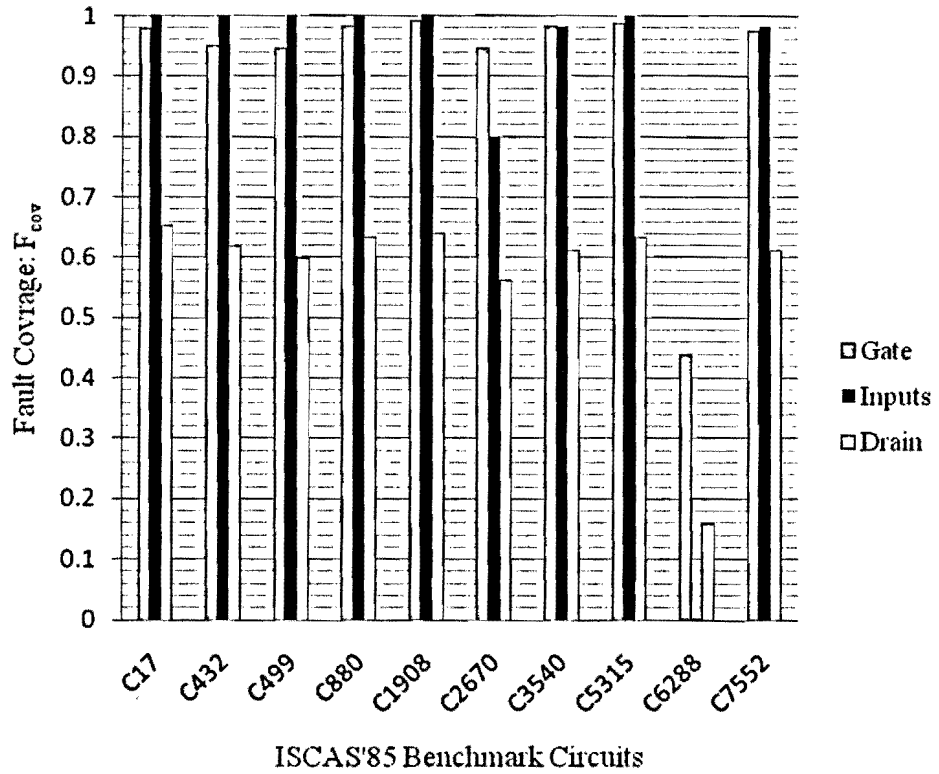


Figure 21:  $F_{cov}$  for gate, drain of a switch and inputs of a logic gate for ISCAS'85 benchmarks circuits

### 6.3. Fault Coverage Versus Applied Test Vectors

Figure 22 shows the efficiency of the applied test vectors on the fault coverage based on injection at drain of a switch for C499, C2670, C3540 and C7552. This feature of the simulator can be used to evaluate the performance of any applied test vectors.

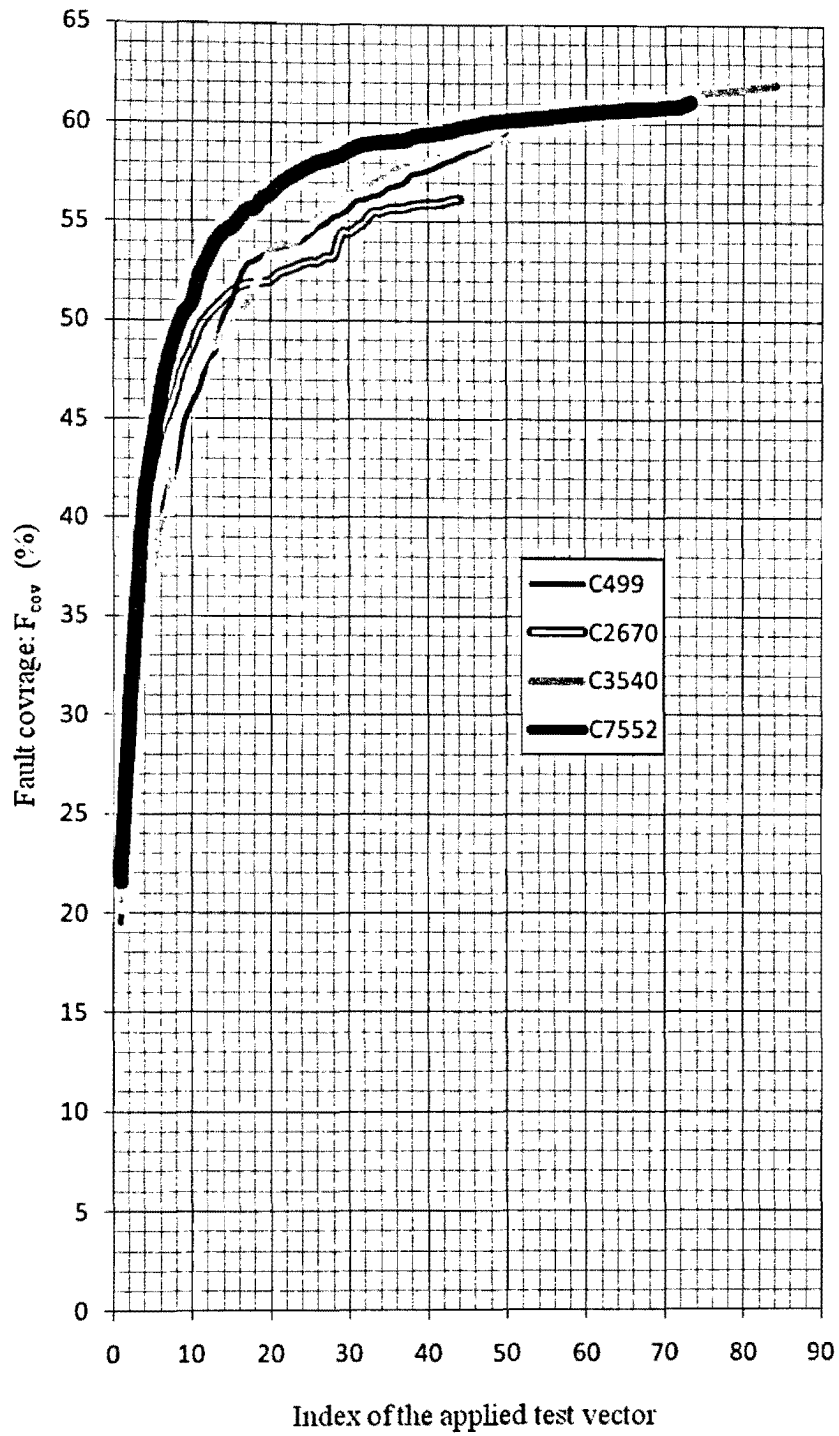


Figure 22: Plot of  $F_{cov}$  based on transient injection at drain of a switch versus applied test vectors for C499, C2670, C3515 and C7552 benchmarks circuits

## 6.4. Timing Results

### 6.6.1. Matlab Profile of SimV<sub>1</sub> and SimV<sub>2</sub> Simulators

Figure 23 shows the main functions of SimV<sub>1</sub>. This image is extracted from Matlab Profile Display showing the execution time for the main functions of this simulator. The parameter “calls” represents the number of times funC, funP and funN are called to run SimV<sub>1</sub> simulator. The number of calls of funC can be determined as shown in (EQ – 6 – 3).

$$\text{funC Number of Calls} = n_{T.d} * nL = 139846 * 11262 = 1574945652 \quad (\text{EQ} - 6 - 3)$$

$$nL = 11262 \quad , \quad n_{T.d} = 139846$$

Figure 24 shows the main functions of SimV<sub>2</sub>. This image is extracted from Matlab Profile Display showing the execution time for the main functions of this simulator. This image shows that the main functions of SimV<sub>2</sub> are very different of SimV<sub>1</sub> and there are no more switch-level functions. The parameter “calls” represent the number of times the corresponding functions are called in the program. The different functions of SimV<sub>2</sub> are as follows:

- **f\_gate\_res**: Represents the gate-level resolution function. This number is built based on the number of progressive gates:  $N_{pg}$
- **f\_res\_solution\_out**: Represents the number of ones of map\_det based on all the independent switch-level logic gates.
- The remainder of the functions is reserved for programming purpose.



Function Name	Calls	Total Time
<u>funC</u>	1.574907e+10	157711.891 s
<u>funP</u>	7.874537e+09	61772.127 s
<u>funN</u>	7.874537e+09	61640.323 s
<u>mean</u>	1	0.010 s

Figure 23: Matlab Profile Display showing the execution time of SimV<sub>1</sub> functions

Function Name	Calls	Total Time
<u>Circuit_gate_Level</u>	1	3857.081 s
<u>f_gate_res</u>	2664585	1395.605 s
<u>f_nb_In_Gates</u>	2664585	34.947 s
<u>f_res_solution_out</u>	59574	6.269 s
<u>f_bin2int</u>	2307	2.420 s
<u>fliplr</u>	85359	1.150 s

Figure 24: Matlab Profile Display showing the execution time of SimV<sub>2</sub> functions

## 6.6.2. Timing Results of SimV<sub>1</sub> Simulator

Table 34 shows the execution time for ISCAS'85 benchmark circuits based on transient injection at drain of a switch for “nF = 8” and “nF = 23”. The speedup column shows an average speedup of 3.2 of “nF = 8” versus “nF = 23”. The execution time corresponding to “nV = 1” (i.e. One test vector) is used to plot the number of switches versus the simulation execution time

for one test vector as shown in Figure 25. The data of this plot are quadratic.

(EQ – 6 – 4) shows the fitting equation extracted from Matlab of the quadratic curve.

$$T_{SimV_1} = 3.1 * 10^{-4} * X^2 - 7.6 * 10^{-2} * X + 1.3 * 10^2 \quad (EQ - 6 - 4)$$

$T_{simV_1}$ : Represents the execution time

X: Represents the number of switches in CUT

Table 34: Timing results extracted from SimV<sub>1</sub> simulator based on transient injection at drain of switch

Benchmarks	n <sub>T,d</sub> nF = 8	n <sub>T,d</sub> nF = 23	T <sub>cycle</sub>	T(nF = 8)	T(nF = 23)	T(nF = 8) nV = 1	Speedup nF 8 vs 23
C17	1074	3433	0.00205	2.206688525	7.05359563	0.3944918	3.196461825
C432	89226	289969	0.03301	2945	9570.73841	236.58754	3.249826284
C499	399840	1286082	0.08312	33235.24	106900.87	1449.6363	3.216491597
C880	108883	344287	0.06761	7361.11	23275.7591	974.60358	3.161990393
C1908	1148576	3651811	0.1342	154142.04	490083.022	3699.701	3.179424783
C2670	901650	2907137	0.21302	192071.61	619285.182	9659.2858	3.224241114
C3540	2063684	6563123	0.28719	592667.5	1884857.23	17240.535	3.180294561
C5315	1398426	4434321	0.41203	576196	1827081.32	37122.418	3.170937182
C6288	1058249	3414302	0.54873	580689.64	1873519.18	63072.859	3.226369219
C7552	3605618	11456616	0.58	2091258.44	6644837.28	71456	3.17743477

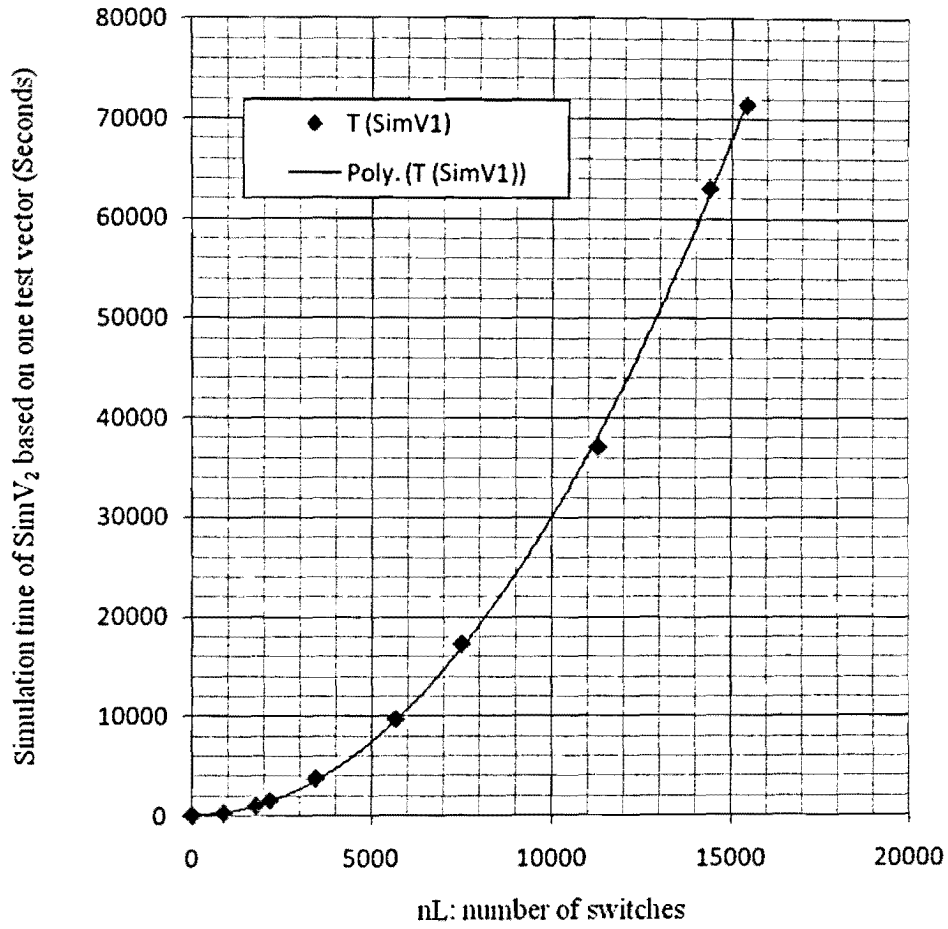


Figure 25: SimV<sub>1</sub> Simulation time for ISCAS'85 benchmark circuits based on transient injection at drain of a switch

### 6.6.3. Timing Results of SimV<sub>2</sub> Simulator

Table 35 shows the execution time for ISCAS'85 benchmark circuits based on transient injection at drain of a switch. The different parameters in this table are as follows:

$T(RF_{\text{Gate-level}})$ : Represents the simulation execution time of the resolution function gate-level

$T(P_{\text{Algorithm}})$ : Represents the simulation execution time needed apply the algorithm in order to build the profiling maps of SimV<sub>2</sub> simulator.

$T(\text{SimV}_2)$ : The execution time to run  $\text{SimV}_2$  simulator for all test vectors.

$T(\text{SimV}_2)$ ,  $nV=1$ : The execution time to run  $\text{SimV}_2$  simulator for one test vector.

The execution time of  $T(P_{\text{Algorithm}})$  is insignificant in comparison to  $T(RF_{\text{Gate-level}})$  as shown in Table 35. Figure 26 shows the plot of  $T(\text{SimV}_2)$  versus  $N_{pg}$  (i.e. the number of progressive logic gates). This plot is not a good quadratic approximation due to different reasons. One of the reasons can be caused by accessing the arrays of the gates as the different processed gates have different sizes in terms of inputs and this can result in a large amount of data to access creating an irregularity of used gate types due to the type of CUT. Equation ( EQ – 6 – 5 ) shows the fitting equation extracted from Matlab.

$$T_{\text{SimV}_2} = 8.7 * 10^{-12} * X^2 + 2.4 * 10^{-5} * X + 0.99 \quad (\text{EQ} - 6 - 5)$$

$T_{\text{simV}_2}$ : Represents the execution time of  $\text{SimV}_2$

$X$ : Represents  $N_{pg}$

Table 35: Timing results extracted from  $\text{SimV}_2$  simulator

Benchmarks	$T(RF_{\text{Gate-level}})$	$T(P_{\text{Algorithm}})$	$T(\text{SimV}_2)$	$nV$	$T(\text{SimV}_2)$ , $nV = 1$	$N_{pg}$
C17	0.1	0.16	0.26	12	0.021666667	27
C432	13.43	1.06	14.49	27	0.536666667	27260
C499	227.8	2.12	229.92	52	4.421538462	191889
C880	36.64	1.47	38.11	16	2.381875	73919
C1908	927.35	3.91	931.26	106	8.785471698	388520
C2670	1721.74	4.3	1726.04	44	39.22818182	807084
C3540	4116.64	6.79	4123.43	84	49.08845238	1395284
C5315	3857.41	7.98	3865.39	37	104.47	2664585
C6288	1968.34	7.21	1975.55	12	164.6291667	2922152
C7552	35222.64	11.15	35233.79	73	482.6546575	6175854

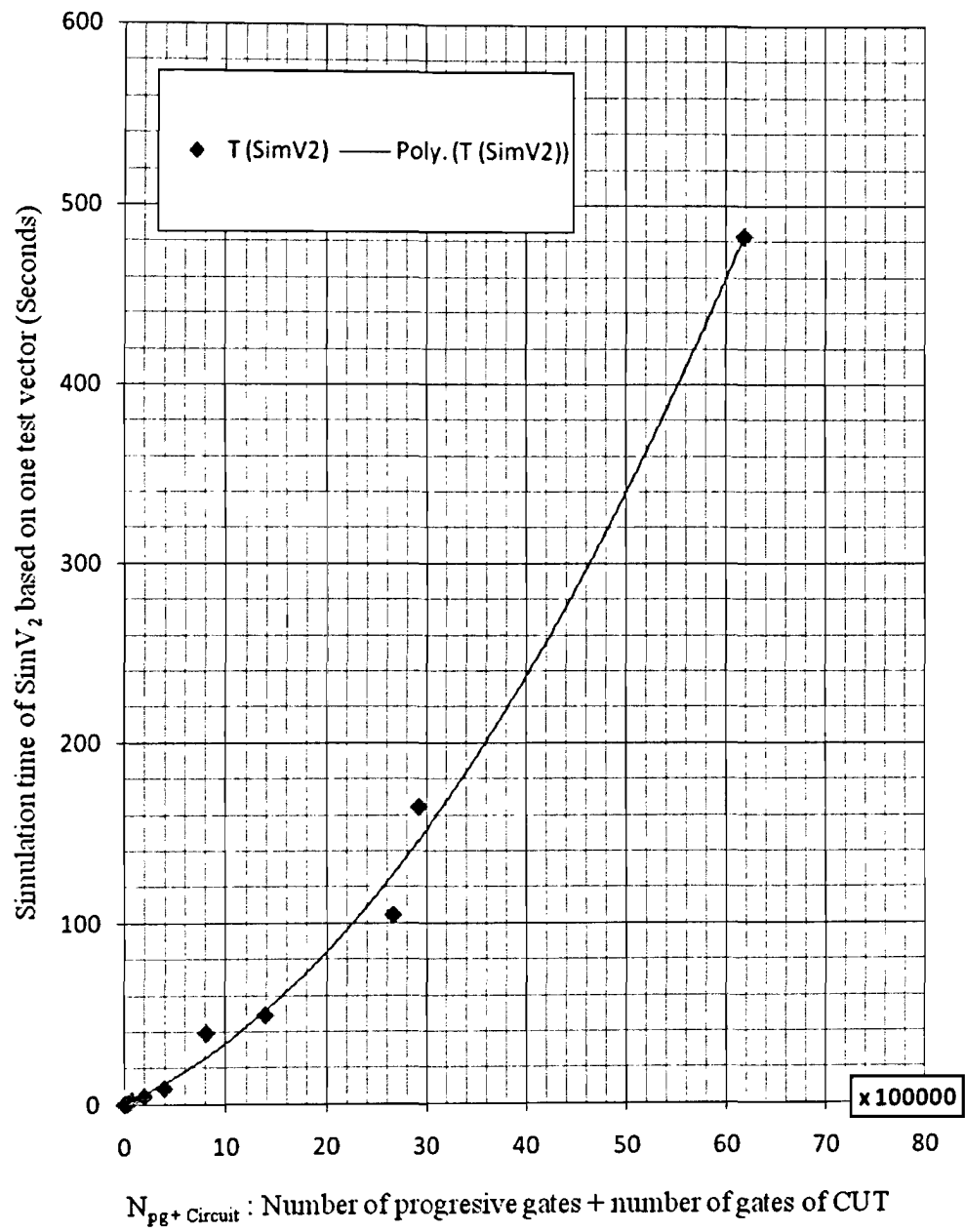


Figure 26: SimV<sub>2</sub> Simulation execution time for ISCAS'85 benchmark circuits.

#### **6.6.4. Performance Study of the developed Simulators**

Table 36 shows the execution time for SimV<sub>1</sub> and SimV<sub>2</sub> simulators based on all the applied test vectors and for transient injection at drain of a switch. The table shows an average speedup of 147 of SimV<sub>2</sub> versus SimV<sub>1</sub>.

Table 37 shows the execution time for SimV<sub>1</sub> and SimV<sub>2</sub> simulators based on one test vector and for transient injection at drain of a switch. The table shows an average speedup of 320 of SimV<sub>2</sub> versus SimV<sub>1</sub>.

The speedup of SimV<sub>2</sub> versus SimV<sub>1</sub> seems to reduce when a larger set of test vectors is used. Even with a large number of test vectors, SimV<sub>2</sub> achieves a significant speedup in comparison to SimV<sub>1</sub> simulator.

Table 36: Performance and speedup of SimV<sub>2</sub> versus SimV<sub>1</sub> based on all applied test vectors and based on transient injection at drain of a switch

Timing comparison based on all applied test vectors			
Benchmarks	T (SimV <sub>1</sub> )	T (SimV <sub>2</sub> )	Speed Up
C17	2.20668853	0.26	8.487263558
C432	2945	14.49	203.2436163
C499	33235.24	229.92	144.5513222
C880	7361.11	38.11	193.1542902
C1908	154142.04	931.26	165.5198763
C2670	192071.61	1726.04	111.2787711
C3540	592667.5	4123.43	143.7316748
C5315	576196	3865.39	149.0654242
C6288	580689.64	1975.55	293.9382147
C7552	2091258.44	35233.79	59.35377488
Average Speed Up			147.232423

Table 37: Performance and speedup of SimV<sub>2</sub> versus SimV<sub>1</sub> based on one test vector and based on transient injection at drain of a switch

Timing comparison based on one test vector			
Benchmarks	T (SimV <sub>1</sub> )	T (SimV <sub>2</sub> )	Speed Up
C17	0.3944918	0.021666667	18.2073137
C432	236.587542	0.536666667	440.8463507
C499	1449.63632	4.421538462	327.8579009
C880	974.603582	2.381875	409.1749489
C1908	3699.70099	8.785471698	421.1158048
C2670	9659.28585	39.22818182	246.2333302
C3540	17240.5346	49.08845238	351.213651
C5315	37122.4182	104.47	355.3404635
C6288	63072.859	164.6291667	383.1208059
C7552	71456	482.6546575	148.0478825
Average Speed Up			<b>310.115845</b>



## Chapter 7

### Conclusion

The advanced switch-level models used in this work are implemented for soft error detection and can inject transients of different strength and logic levels at any location type in a combinational circuit. An algorithm is developed in this work to speed up the simulation execution time. The experiments conducted on ISCAS'85 benchmarks circuits led to the following conclusions:

- The soft error coverage based on gate of a switch and Inputs of a logic gate are in the same range and achieved an average of 0.9. On the other hand and for transient injection at drain of a switch the average fault coverage is 0.6.
- The developed algorithm based simulator in this work achieved a speedup of 310 for all the test vectors and 147 for one test pattern versus the developed non-algorithm based simulator in this work
- The algorithm based simulator is divided into two modules, gate-level resolution function and the algorithm module. The execution time of the algorithm module is insignificant in comparison to the gate-level resolution function, thus the bottle-neck of this simulator is the

gate-level module. The execution time of the simulator can be improved as the literature is filled with various optimized gate-level based simulators and the combination of this algorithm with an optimized gate-level simulator can add a significant speedup to the simulation.

## PUBLICATIONS

1. Kalkat, P.K.; Sedaghat, R.; Chikhe, J.M.; Javaheri, R.; , "Soft error injection using advanced switch-level models for combinational logic in nanometer technologies," *Microelectronics (ICM), 2009 International Conference on* , vol., no., pp.332-335, 19-22 Dec. 2009

## BIBLIOGRAPHY

- [1] R. C. Baumann, "Single event effects in advanced CMOS Technology," in Proc. IEEE Nuclear and Space Radiation Effects Conf. Short Course Text, 2005.
- [2] M. P. Baze, S. P. Buchner, "Attenuation of single event induced pulses in CMOS combinational logic," *IEEE Transactions on Nuclear Science*, vol. 44, pp. 2217–2223, December 1997.
- [3] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, Lorenzo Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic", Proceedings of the 2002 International Conference on Dependable Systems and Networks, p.389-398, June 2002.
- [4] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Exploiting Circuit Emulation for Fast Hardness Evaluation", *IEEE Transactions on Nuclear Science*, Volume 48, Issue 6, pp. 2210-2216, December 2001.
- [5] S. Kundu, M.D.T. Lewis, I. Polian, B. Becker, "A soft error emulation system for logic circuits," *Conference on Design of Circuits and Integrated Systems*, page 137, 2005.
- [6] Kim, S., Iyer, R. K., "Impact of Device Level Faults in a Digital Avionic Processor", Proc. AIAA/IEEE 8th Digital Avionics Systems Conference (DASC), Oct 17-20, 1988, pp. 428-436.
- [7] V. Degalahal et al., "SESEE: soft error simulation and estimation engine," *Proceedings of MAPLD International Conference*, 2004.

- [8] J.P. Hayes, "Pseudo-Boolean Logic Circuits," *IEEE Transactions on Computers*, vol. 35, no. 7, pp. 602-612, July 1986.
- [9] Dahlgren P, Liden P, "Efficient Modeling of Switch-Level Networks Containing Undetermined Logic Node States", *Proceedings of IEEE/ACM Int. Conf. on CAD*, pp. 746-752, November 1993.
- [10] Verilog Hardware Descriptor Language Reference Manual (LRM) DRAFT, IEEE-STD 1364, 1995.
- [11] Ejlali A, Miremadi SG, "FPGA-based fault injection into switch-level models", *Journal of microprocessors and microsystem, Elsevier Science*, 28(5-6): pp.317-27, April 2004.
- [12] T.A. DeLong, B.W. Johnson, J.A. Profeta lii, A fault injection technique for VHDL behavioral-level models, *IEEE Design and Test of Computers* 13 (4) (1996) 24-33.
- [13] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool," *Proceedings of 24th International Symposium on Fault-Tolerant Computing*, 1994, pp. 336-344.
- [14] Reza Javaheri, Reza Sedaghat," Dynamic Strength Scaling for Delay Fault Propagation in Nanometer Technologies", *14<sup>th</sup> International CSI conference*, 2009.
- [15] Kalkat, P.K.; Sedaghat, R.; Chikhe, J.M.; Javaheri, R.; , "Soft error injection using advanced switch-level models for combinational logic in nanometer technologies," *Microelectronics (ICM), 2009 International Conference on* , vol., no., pp.332-335, 19-22 Dec. 2009

[16] [http://www.asicnorth.com/images/asicNorth\\_automated\\_analog\\_verification.pdf](http://www.asicnorth.com/images/asicNorth_automated_analog_verification.pdf)

Relationship between strength levels and voltage levels for 1.8 supply voltage.

[17] <http://www.eecs.umich.edu/~jhayes/iscas/>

ISCAS'85 benchmark circuits.

[18] Hamzaoglu I, Patel J.H, "Test set compaction algorithms for combinational circuits",

*IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*,

Volume 19, Issue 8, Page(s):957 – 963, August 2000.

⑥ BL-21-44