Spring 6-21-2023

Stochastic Worst-Case Test Vectors for ASIC Devices in Single Event Environment

Mostafa Hemeda
mostafa.hemeda@aucegypt.edu

The American University in Cairo

Graduate Studies

Stochastic Worst-Case Test Vectors for ASIC Devices in Single Event Environment

A THESIS SUBMITTED BY

Mostafa Abdelmohsen Hemeda

TO THE

Electronics and Communications Engineering Department

SUPERVISED BY

Prof. Ahmed Abou-Aouf

May 2023

in partial fulfillment of the requirements for the degree of
Master of Science

Declaration of Authorship

I, Mostafa Abdelmohsen Hemeda, declare that this thesis titled, "Stochastic worst-case test vectors for ASIC devices in Single Event Environment" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Mostaha Abdelmohsen Hemeda

Date:

17/4/2023

The American University in Cairo
School of Sciences and Engineering

"Stochastic Worst-Case Test Vectors for ASIC Devices in Single Event Environment"

A Thesis Submitted by

Mostafa Abdelmohsen Hemeda

To the

Department of Electronics and Communications Engineering

May 14, 2023

In partial fulfillment of the requirements for the degree of

Master of Science in Electronics and Communications Engineering
Has been approved by

Dr. Ahmed Abou-Auf (Advisor)
Professor, Department of Electronics and Communications Engineering
American University in Cairo

Ahmed A. Abou-Auf

Dr. Yehea Ismail (Internal Examiner)
Professor and Chair, Department of Electronics and Communications Engineering
American University in Cairo

Dr. Yehea Ismail

Dr. Amr Wassal (External Examiner)
Professor and Chair, Department of Computer Engineering
Cairo University

Amr Wassal

Dr. Sherif Abdel Azeem (Moderator)
Professor and Graduate Program Director, Department of Electronics and Communications Engineering
American University in Cairo

Sherif Abdel Azeem

Sherif Abdel Azeem
Graduate Program Director

May 16, 2023
Date

Dean

May 21, 2023
Date

Abstract

Charged particles and energetic particles can impact the integrated circuit, referred to as single event effects (SEE). Nuclear reactors and space radiation can produce these particles. These effects can negatively affect the reliability and performance of electronics. When SEE occurs, a transient current is created, which can cause electronic devices to have incorrect outputs and ultimately fail. As a result, ensuring the reliability of ASIC circuits is a significant concern.

This thesis discusses the different fault types, then discuss the soft error and, in particular, the Single Event Transient (SET) and its causes and models. Then, this thesis proposes a new model to get the probability of error of standard cells using a pre-characterized cell module. The proposed method guarantees that no logical masking and over-estimation of the probability of error has occurred. Furthermore, it suggests calculating the error probability of the whole combinational circuit in application-specific integrated circuits (ASIC) using the Krishnaswamy method and a newly proposed method. Finally, this thesis presents a new approach to getting the worst-case vector based on Krishnaswamy and a newly proposed method. The flow gets the most probable vector as the worst-case vector from a set of vectors generated from the Automated Test Pattern Generation (ATPG) tool. In addition, a new error metric is discussed to find the worst-case vector based on two-norm.

Acknowledgments

I would like to convey my utmost gratitude and appreciation to my thesis advisor, Prof. Ahmed Abou-Auf, for his unwavering support, invaluable guidance, and remarkable patience throughout the duration of my thesis.

I would like to thank my examiners for devoting time to review the thesis and providing valuable comments.

Also, I would like to thank all my colleagues that offered help and support throughout the research work, especially Eng. Mohamed Wael, Eng. Ahmed Ibrahim and Eng. Mostafa Abdelaziz.

Contents

List of Figures

List of Tables

List of Abbreviations

SET	**Single Event-Transient**
SEE	**Single Event Effects**
EM	**Electromigration**
IC	**Integrated Circuits**
ESD	**Electrostatic Discharge**
ATPG	**Automated Test Pattern Generation**
ASIC	**Application-Specific Integrated Circuits**
BEOL	**Back End Of Line**
SEU	**Single Event Upset**

List of Symbols

I_{SET}	SET current
$Q_{Critical}$	The critical charge for SET
M	PTM matrix
J	ITM matrix
A	Output stimulus probability matrix for faulty circuit
B	Output stimulus probability matrix for ideal circuit

Chapter 1

Circuit Faults

According to Moore's law, the transistor keeps scaling down until it reaches the nanometer range. Scaling down transistors, in turn, introduced some new effects that were not considered before, like RLC parasitics, on-chip variations, and IR drop. In addition, the complex fabrication process of advanced nodes can cause defects in the fabricated chips. Furthermore, process variation in advanced nodes became a serious problem. These effects are known as Deep Submicron effects (DSM). Furthermore, the high cost of testing the chips makes it vital to study the possible faults to manufacture fault-free chips and make sure that the fabricated chip is testable. For example, FIGURE 1 indicts the cost of fabrication per transistor is decreasing over the years, but the cost of testing per transistor is almost the same. [1]

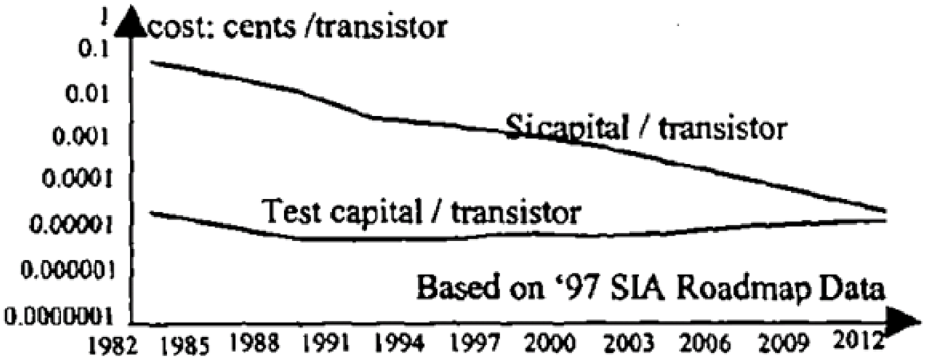


FIGURE 1 Cost of Fabrication vs Cost of Testing[1]

In general, circuit faults can be divided into two main categories:

1) Permanent faults: these can cause the circuit to break permanently, such as stuck-at faults, IDDQ.

2) Transient faults: these faults can cause the circuit to fail momentarily, such as Single-Event-Transient (SET), due to radiation, electromagnetic interference,

electromagnetic discharge, or power noise.[2]

1.1 Permanent Faults

Any fault, in general, is the abstraction of failure in a circuit in a particular node. Permanent faults can happen due to the mal-fabrication of the circuit interconnect, which causes stuck-at faults. Furthermore, any device mal-fabrication can cause more current to flow across it than expected. This can cause IDDQ faults.

1.1.1 Stuck at Faults

For example, if the 2-input AND gate has 2-input terminals A and B and one output terminal C. Stuck-at Faults can happen due to shorting a certain node to VDD (stuck-at 1) or to ground (stuck at 0). If any terminals are tied to VDD, it is called a stuck-at-1 fault. On the other hand, if they are tied to zero, it is called stuck at zero faults. This can damage the circuit functionality, which is $C = AB$. This problem is becoming more severe due to the need for a less resistance interconnect, such as a cooper can cause faults such as stuck-at faults due to shorts[3]. Moreover, these defects can be caused by:

1- Process Defects:

Process defects are caused by fabrication processes like shorts, opens, or latch-up.

2- Material Defects:

Material defects are related to materials used in fabrication, such as metals and silicon, as the material can have imperfections that will cause faults.

3- Aging Defects:

Some metals can become open due to aging effects over time. This is called electromigration (EM). This is caused when current flows in a net with a value greater than the maximum value. EM is solved by widening interconnect. Furthermore, temperature, mechanical stress, or radiation can cause the dielectric to break, which can cause to ruin the transistor's functionality.

4- Packaging Defects:

Some faults can happen due to packaging if there is a mismatch between the core pads and wire bonding which can be caused unwanted parasitics that will cause the circuit to fail.

1.1.2 IDDQ Based Faults

When the input is static, no current must be drawn out from the circuit (as the leakage current is negligible). So, if some current is taken from the VDD for some static input, this circuit must be faulty, as shown in FIGURE 2. So, the main concept of this method is to measure the output current and observe the quiescent supply current. If the current is large, a fault issue can cause failure or reliability [4].

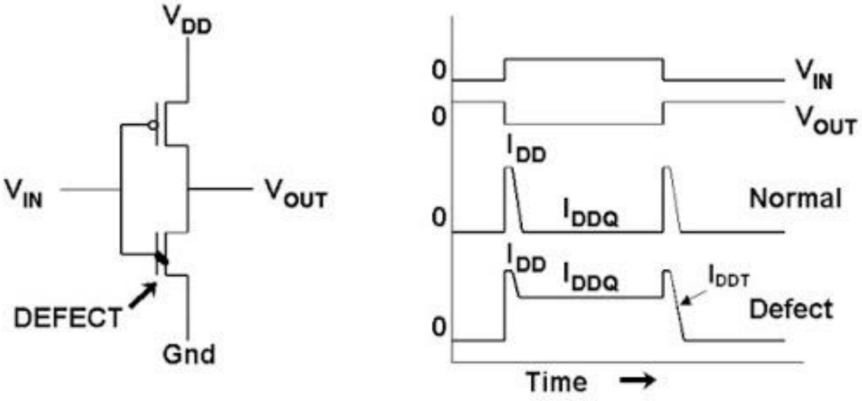


FIGURE 2 IDDQ example

Example of errors :

1) Bridging error: If the NMOS transistor is off, and PMOS is on because the input is 0 in a certain branch, but the NMOS is on, and PMOS is off in another branch, and then there is a direct path from the two on transistors as shown in FIGURE 3 A then the current will flow. This can be captured by IDDQ test.

2) Floating node detection:

The floating node can take any value regardless of the input. So, it can take a value that can drive current in this branch, as shown in FIGURE 3 B.

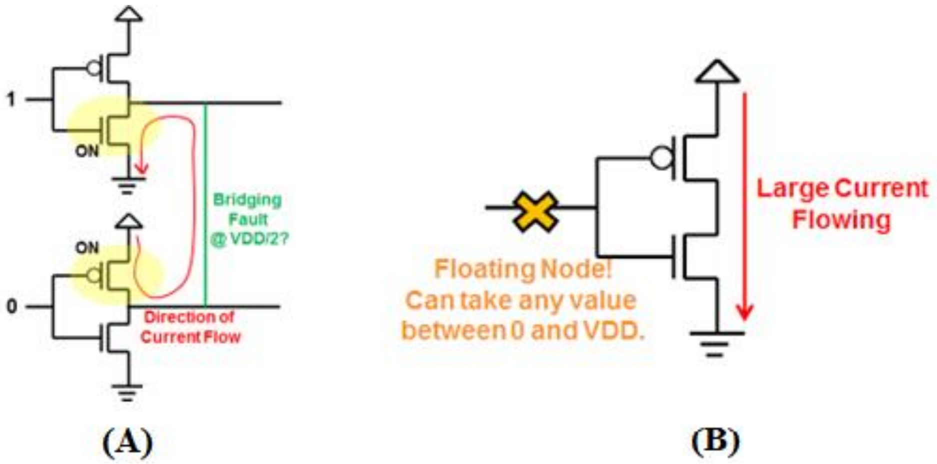

1
ON
Bridging Fault @ VDD/2?
ON
0
Direction of Current Flow
(A)
Large Current Flowing
Floating Node! Can take any value between 0 and VDD.
(B)

FIGURE 3(A) Example of Bridging error and (B) Example of floating node Error

This method has a lot of advantages as it simply depends only on measuring the current and can catch a lot of defects. Area overhead is minimum. On the other hand, it takes less time than scan chains and is inefficient for technology with large leakage. So Table 1 summarizes the IDDQ testing process. So, if there was a high current at a high frequency, that is most probably because of a defect.

Table 1 Decision of fault based on IDDQ method

I_{DD}	F_{MAX}	**Decision on IC**
H	H	Good -Fast
H	L	Defect
L	H	Unlikely
L	L	Good -Slow

1.2 Transient Faults

Transient faults are a type of fault that causes the circuit to malfunction instantaneously. These errors are called soft errors in a circuit. There are various reasons for transient errors, such as:

1) Cosmic rays

High-energy particles from space, such as protons and neutrons, can collide with the atoms in the semiconductor material and create a charge imbalance in the chip that affects the state of the device, causing the logic to change. In fact, 95% of soft errors are caused by cosmic rays. Although cosmic rays' leading cause is not reaching Earth, the resultant alpha particles that hit the Earth's surface cause SET. For example, in New York, the flux of neurons is 14 $\frac{neurons}{cm^2h}$. It has been noted that cosmic rays decay with distance, meaning that the nearer the earth's surface, the fewer neurons will reach the ground. In other words, well-isolated chips and computers have fewer soft errors than regular chips. IBM has studied the effect of cosmic rays on chips and found that concrete shielding will cause a significant reduction in SET [5]. For example, an experiment was done to measure the transient error rate at to be 5,950 failures in time per billion hours per DRAM chip. On the other hand, when the same test setup was moved to a shielded underground vault of rock that effectively eliminated all cosmic rays, no soft errors were measured [6].

2) Alpha particles:

Alpha particles are emitted by radioactive materials in the environment and can cause soft errors in VLSI. These particles can be a big problem in space and nuclear power systems. Furthermore, space and nuclear power devices need high reliability as they are costly to produce, and such systems have no tolerance for error. [5]

3) Neutron radiation:

In a space or nuclear environment such as a nuclear reactor, neutrons can trigger nuclear reactions in the semiconductor material causing defects. Neutron beams can cause a brust that can affect the condition of nearby device’s state. Neutron beams damage electronics and can affect measurements. [7]

4) Power supply fluctuations:

Rapid changes in the power supply voltage or current can cause transient errors in the device's function. Electromagnetic interference and power spikes near the power sources can cause rapid change. These fluctuations are very sensitive for analog designs. For example, for a voltage-controlled oscillator (VCO), power fluctuations can cause the variable capacitors to charge, including the drain-bulk capacitance, which causes shifting of the output frequency of the VCO and its gain, which can damage the functionality momentary[8].

5) Electrostatic discharge:

Electrostatic discharge (ESD) can cause soft errors in VLSI devices by creating a charge imbalance in the circuit. ESD occurs when an object with a high electrical charge comes into contact with a conductor or semiconductor material. This contact can cause a sudden discharge of static electricity, which can generate a high-energy pulse of electrons (unwanted current). When the ESD pulse reaches a VLSI device, it can induce a transient voltage that can alter the device's state. The voltage generated by the ESD pulse can be much higher than the operating voltage of the device, which can cause an unintended state transition in the circuit. This high voltage can lead to soft errors in the device's data output. ESD can also cause physical damage to the device, such as melting or puncturing the metal interconnects or oxide layers which can lead to permanent damage or failure of the device. To prevent ESD-induced soft errors, designers can use various techniques, such as implementing ESD protection circuits, grounding and shielding processes, and anti-static devices and procedures during device handling and manufacturing. ESD protection circuits can be designed to shunt the high-energy ESD pulse away from the sensitive circuits and discharge the charge safely to the ground. Grounding and shielding can reduce the buildup of static charges in the environment. At the same time, anti-static devices and procedures can minimize the risk of ESD discharge during device handling and manufacturing [9].

6) Process variations:

Process variation can have a significant impact on soft errors in VLSI devices. It refers to the natural variations that occur during the manufacturing process, which can cause differences in the electrical properties of the devices. These variations may include temperature, pressure, doping, and chip location in the silicon, which can cause variations in the charge collection and generation mechanisms, affecting the device's susceptibility to soft errors. To mitigate the effects of process variation on soft errors, designers can use techniques such as redundancy and error-correcting codes. Redundancy involves using additional hardware (replica logic) to duplicate critical circuits or data paths, which can provide a backup in case of a soft error. In this technique, the same logic is used more than once, and there is a decision logic based on majority voting to decide the correct logic. Error-correcting codes use mathematical algorithms to detect and correct errors in the data, which can improve the system's reliability.[10]

7) Temperature fluctuations:

Rapid temperature changes can cause soft errors in VLSI devices. Temperature fluctuations can be caused by thermal noise, thermal runaway, or other factors. The experimental result from TCAD of 130nm technology causes the signal event transient width to increase with the temperature increase, as shown in FIGURE 4.[10]

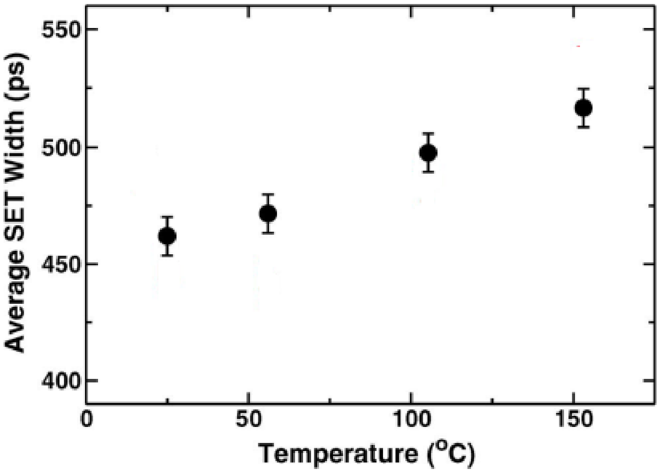


FIGURE 4 Effect of temperature on the average SET width[10]

1.3 Fault Modeling

The ability to accurately model circuit faults is critical to diagnosing and troubleshooting electronics as well as preventing these errors. In this context, fault modeling refers to the process of identifying, characterizing, and simulating circuit faults to gain a deeper understanding of how they occur and how they can be mitigated. This involves creating models that capture the behavior of faulty circuits and developing simulators and test methods to evaluate the performance of these models.[11]

1.3.1 Stuck-at modeling

In order to model stuck-at-fault for a particular node in a netlist, the node in the circuit will be tied to a fixed value which will be zero for stuck and zero faults and one for stuck-at-one faults. For any node in a certain design, each node can be stuck at 0 or stuck at 1 or a fault-free node. This fact gives each node a 3 degree of freedom, so for each circuit, the number of possible combinations will be 3^n where n is the number of nodes in the circuit. This is a huge number and will take a long time to compute. If it is assumed that only a single event happens at a time, the number of combinations will be reduced to only $2n$. The scope of this thesis will focus on a single fault.

The single stuck-at model assumes the following: -

1- Only one fault happens at a time.

2- The node is tied to 0 or 1 permanently.

3- The fault can happen in the inputs or outputs of the gate.

Stuck-at modeling aims to find the test vector that will propagate the error to the observable output. For example, FIGURE 5 represents a faulty circuit with the output of OR gate stuck at 1. To make the error propagate, the algorithm needs the inputs of the OR gate to make the output 0. So the inputs of the OR gate must be 0 and 0. The fault must be propagated to the observable output of the circuit, so the other input of AND2 must be 1 to prevent the masking. So the other two inputs of AND1 must be 1 and 1. So to test the stuck-at error in the output of OR gate, the input must be 1100. Any test modeling algorithm must assume that only one node is stuck and 1 and find the pattern that will it propagate. Then, it will assume it is stuck at 0 and find the other patterns. In addition, all the nodes of the circuit will be assumed to be faulty, and all the patterns will be test patterns generated.

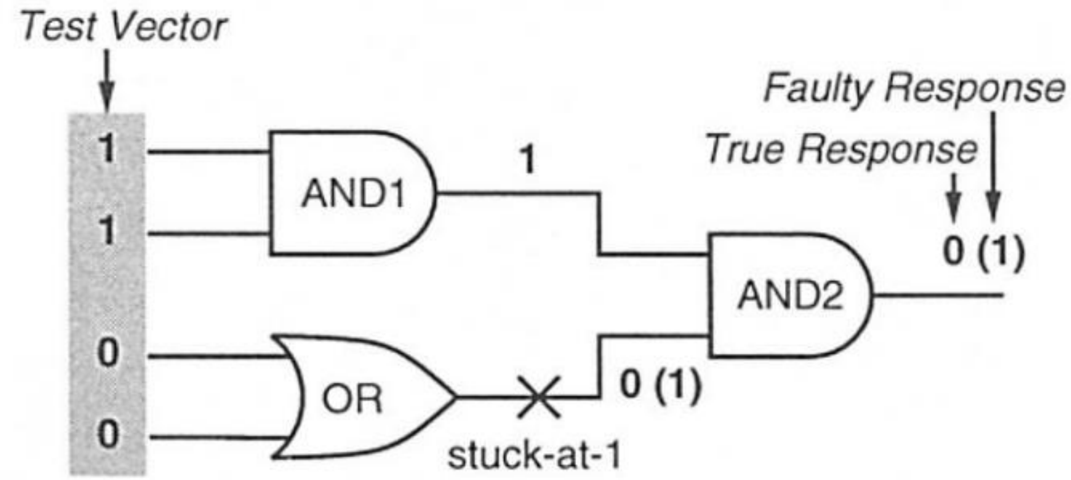


FIGURE 5 Testing stuck-at-1 fault[12]

1.3.2 Fault Equivalence

Two faults are equivalent if they will have the same effects on the circuit output using the same input pattern. For example, if the response of a circuit for a certain test vector (V_1, V_2) are expected to be $f_0(V_1)$ and $f_0(V_2)$ the output of the circuit, if it has a single error in a certain node, is $f_1(V_1)$ And if the error in the other node is $f_2(V_2)$. Then, if the two faults are equivalent, then V_1must equal to V_2 as following

$$f_0(V_1) \oplus f_1(V_1) = 1 \quad (1)$$

$$f_0(V_2) \oplus f_2(V_2) = 1 \quad (2)$$

This means that there are output is different, and no logical masking happened. And $f_0(V_1)$ is different than $f_1(V_1)$. In addition, $f_0(V_2)$ is different than $f_2(V_2)$. If $V_1 = V_2 = V$ then, the two faults are equivalent as follows

$$\big(f_0(V) \oplus f_1(V)\big) \oplus (f_0(V) \oplus f_2(V)) = 1 \quad (3)$$

Simplifying :

$$f_1(V) \oplus f_2(V) = 0 \quad (4)$$

So the two faults are equivalent.

Patterns can not distinguish between equivalent faults. For example, a stuck-at-0 in any of the inputs of the AND gate is equivalent to each other and to a stuck-at-0 at the output of the gate. In addition, a stuck-at-0 at the inverter input is equivalent to a stuck-at-1 at the inverter output. FIGURE 6 illustrates the different equivalent faults for standard cells. The dotted arrow means that the fault is equivalent. For example, for NOR gate, the stuck-at-1 at any input is equivalent to each other, and the stuck-at-0 at the output.

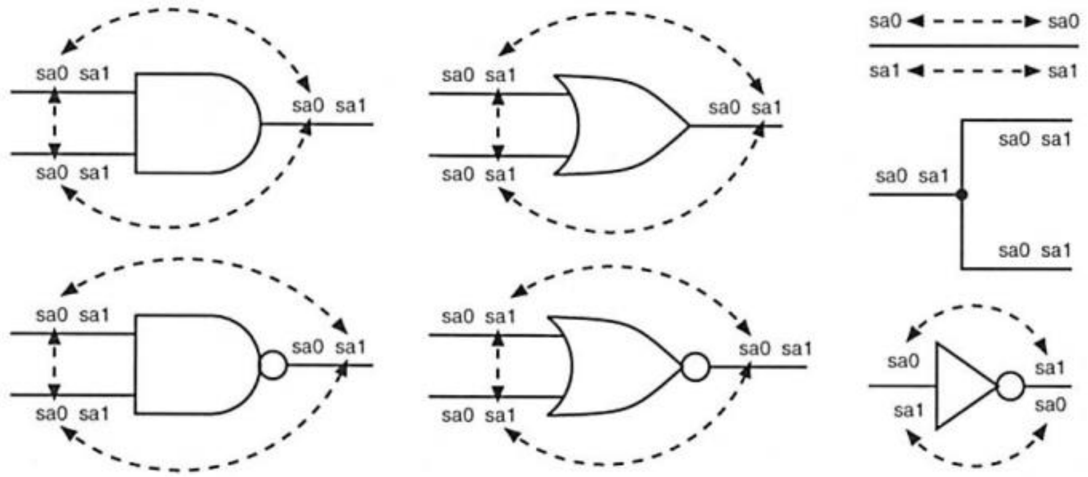


FIGURE 6 Equivalent cells example[12]

The number of distinct stuck-at faults is around 50-60% for any typical circuit, which makes any tool needs to find the distinct stuck-at faults before generating the test vectors. [12]

1.4 ATPG

Automatic Test Pattern Generation (ATPG) is a process used in designing and manufacturing integrated circuits to generate test patterns that can be used to detect faults in the circuit. ATPG aims to produce a set of test patterns that can detect the largest number of faults in the circuit while minimizing the number of patterns required and the time it takes to test the circuit. This is essential because it ensures that the manufactured circuits meet the required specifications and are defect-free.

A possible way to generate the test patterns is to try all the possible inputs for a circuit and test them; this process is inefficient for large circuits. For example, for a 64-bit adder, which will have around 129 inputs (64 for the first input and 64 bit for the 2nd input, and 1 bit to determine whether it is addition or subtraction), the number of test patterns needed will be around 2^{129}. If each test takes 1 ns to compute, the algorithm will need 2^{121} second which is $5.26 * 10^{27}$ years. This is impossible to compute. The process of ATPG involves several steps, including fault modeling, test generation, and simulation. Fault modeling identifies potential faults in the circuit and creates a model of each fault. This model includes information about the fault's location and type of fault, such as a stuck-at or bridging fault. Test generation involves using fault models to create test patterns to detect faults. Simulation is then used to test the circuit using the generated patterns and to identify any faults that were not detected. There are many ATPG methods, each with advantages and disadvantages:

1- Random test pattern generation:

This method includes generating random patterns without specific knowledge of the circuit being tested. While this method is simple and easy to implement, it is not very good at detecting faults and may require multiple testing procedures to achieve proper fault coverage. A typical random test algorithm starts by assuming all the probability of inputs to be equal. Then, the algorithm generates a random vector, sees the fault that it simulates, and saves the tested nodes. Then, the algorithm checks the coverage, which is the nodes and faults percentage tested in a circuit. If the coverage is still small, the algorithm generates another test vector to get new faults and test other nodes. If no new faults is tested, the vector is discarded, changing the probability of inputs to get a new vector. Finally, once the target coverage is reached, the algorithm is halted. FIGURE 7 represents the flow chart of a random algorithm[13].

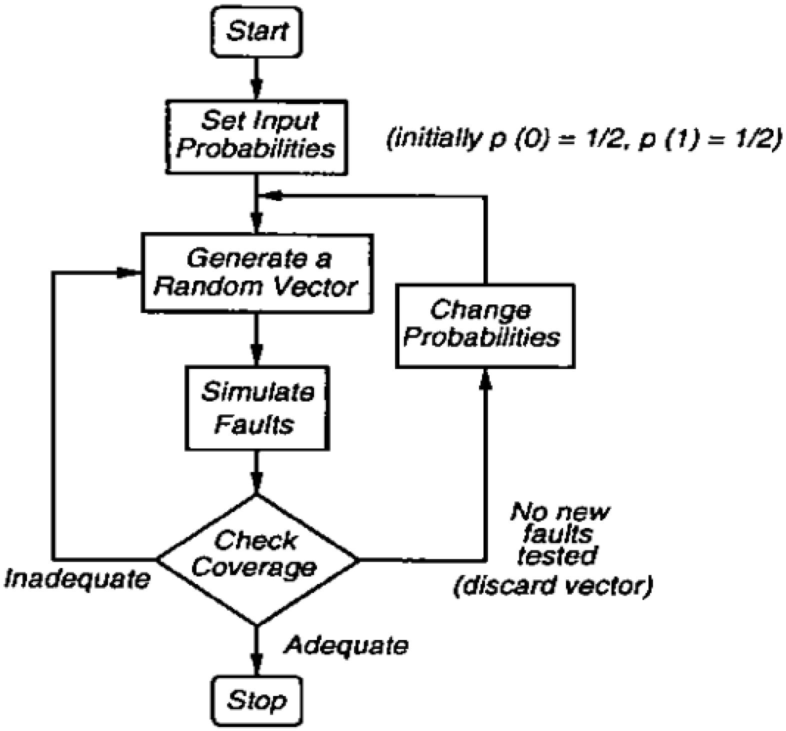
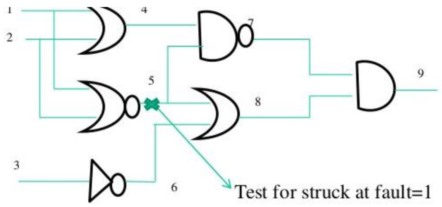


FIGURE 7 Random test algorithm[13]

2- Deterministic test pattern generation:

Another approach is deterministic pattern generation, where test patterns are generated using predefined algorithms and rules. This approach is more capable of detecting errors than random sample generation but can be more complex and time-consuming to implement. One example of such an algorithm is the D algorithm which is an efficient and effective method for generating test patterns for single stuck-at faults in a circuit.

The D-algorithm starts with performing a fault simulation of the circuit using a fault model. This fault simulation generates a set of fault tables that identify the potential faults in the circuit and their corresponding fault values. The fault model used in the D-algorithm assumes that all faults are single stuck-at faults, which means that a single node in the circuit is always stuck at either a 0 or stuck at 1. Then, pattern generation is done by backtracking. In D-algorithm, one can use D-algorithm to donate D to the node under test. If the node tested is supposed to be 1, then its value is donated to D. If it is supposed to be 0, its value is donated to $\overline{D}$. For example, FIGURE 8 illustrates a test case of how D- algorithm works; if test node 5 is the node under test if it is stuck-at-1, the algorithm chooses the path to node 5 from the input and puts input to an initial value (1 0) as it shall produce 0 (to test stuck-at-one). Then, the algorithm wants to propagate the fault to the circuit output. So the output shall be $\overline{D}$. So, node 5 must propagate either by node 8 or 7, let's say from 5 to 8, so node 6 must be zero. Then, for the value to be propagated to node 9, node 7 must be 1. Then, backtracing is done to find the inputs of the circuit. If there is a contradiction to the assumption, the algorithm refuses the assumed path and tries another path. For example, in FIGURE 8, after backtracing, the algorithms found the inputs is 0 0, which contradicts the initial assumption.



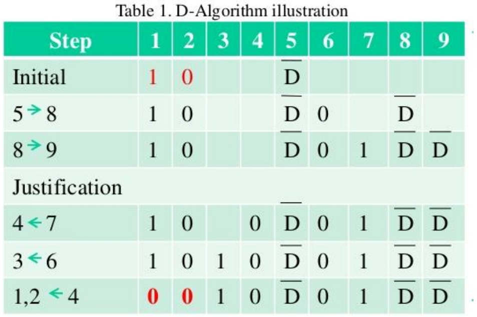
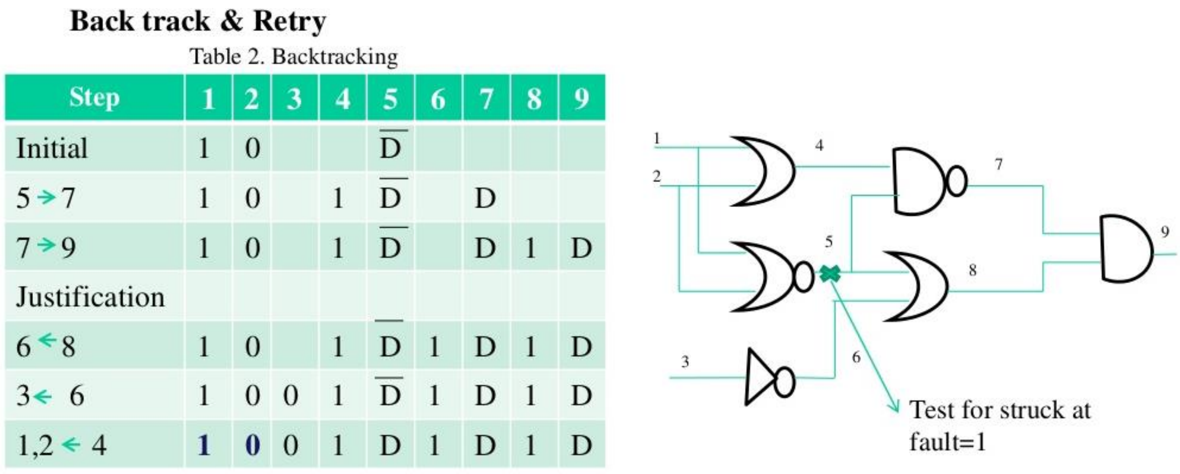


Table 1. D-Algorithm illustration

Step	1	2	3	4	5	6	7	8	9
Initial	1	0			$\overline{D}$				
5→8	1	0			$\overline{D}$	0		$\overline{D}$	
8→9	1	0			$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$
Justification									
4←7	1	0		0	$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$
3←6	1	0	1	0	$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$
1,2 ←4	**0**	**0**	1	0	$\overline{D}$	0	1	$\overline{D}$	$\overline{D}$

FIGURE 8 D-Algorithm illustration[14]

FIGURE 9 represents the retry for the second path, which is node 5 propagates the fault through node 7 to the output node. The algorithm found that the inputs agreed with the assumption. So this pattern will be the correct test pattern to test if node 5 is stuck-at-1. Then, this method is done for all the nodes to get all the test patterns. [14]



Back track & Retry

Table 2. Backtracking

Step	1	2	3	4	5	6	7	8	9
Initial	1	0			$\overline{D}$				
5 → 7	1	0		1	$\overline{D}$		D		
7 → 9	1	0		1	$\overline{D}$		D	1	D
Justification									
6 ← 8	1	0		1	$\overline{D}$	1	D	1	D
3 ← 6	1	0	0	1	$\overline{D}$	1	D	1	D
1,2 ← 4	1	0	0	1	D	1	D	1	D

FIGURE 9 D-algorithm backtrack and retry[14]

So the process of the D-algorithm can be summarized as

1- Initialize the gate inputs to detect the fault at a certain node.

2- Select the other node values so the fault will propagate to the output through a certain path.

3- Then backtrack to get the input's value and check for contradiction.

4- If a contradiction is found, repeat from 2 with a different path till no contradiction is found.

5- If no contradiction is found, record the test pattern and the fault tested and move to other nodes.

3- Hybrid test pattern generation:

This method tried to take advantage of both stochastic and deterministic approaches to make the algorithm faster and more efficient. For example, the algorithm could use a deterministic method to generate test patterns for the most critical faults in the circuit and then use a stochastic method to generate test patterns for the remaining faults. This will reduce the coverage of the algorithm to reduce the runtime. Another way is using probabilistic approaches, including deterministic approaches to produce a seed of initial test vectors, then stochastic approaches to get the remaining test vectors. One example of a hybrid pattern generation algorithm is the Genetic Algorithm-based ATPG (GA-ATPG). GA-ATPG is a probabilistic algorithm that uses a combination of deterministic and stochastic methods. It starts by using a deterministic algorithm to generate a set of initial test patterns. These test patterns are then used as the initial population for the genetic algorithm, which generates test patterns from the initial population by mutation and cross-over. Then, it evaluates the test vector by the number of faults detected. The algorithm stops if the maximum number of iterations is met or the target coverage is met. [15]

So, choosing the appropriate algorithm requires a tradeoff between the runtime and coverage. Several commercial ATPG tools are available, including Synopsys' TetraMAX, Mentor Graphics' Tessent FastScan, and Cadence's Encounter Test. These tools provide a range of features, including support for different fault models, automatic test pattern generation, and fault simulation.

Chapter 2

Single Event Transient

Single-event Transient (SET) is the dominant source of soft errors in application-specific integrated circuits (ASIC). SET is becoming more prevalent upon scaling down the transistor size. Because the downsizing of the transistor scales down the supply voltage, which makes it more susceptible to SEE. Furthermore, all the electronic devices in the nominal conditions is exposed to radiation effects which is the main cause of SET. So, it has become necessary to investigate the SEE effects on electronic devices and try to minimize its effect to increase the reliability performance of any ASIC circuit [16]. It is worth noting that this phenomenon is not only special for CMOS electronics but also common for other technologies such as memristors[17].

The SET effects can cause real reliability challenges for circuit design as they can cause serious faults. For example, IBM reported 5,950 transient faults per billion hours for its DRAM in normal conditions [5]. Furthermore, it can happen in a vital system where it can cause a real problem. For example, in 2008, SEE affected an Airbus when it was traveling from Singapore to Australia. The SEE caused the computer to believe that the plane's angle of attack changed, and to adjust it back, the computer moved the cord of the airbus, causing injury to passengers. This made the plane manufacturer take extra strict measures to prevent this incident from happening again. This accident indicates the necessity of understanding the SEE effects in the reliability of ASIC design, especially in a highly radiative area like nuclear plants or space[16]. Furthermore, circuits controlling people's lives, like hospitals and transportation, must resist SET.

This chapter will explain the SET’s physical causes and how it affects CMOS devices. Then, modeling the SET effects will be explained in a circuit model and device model.

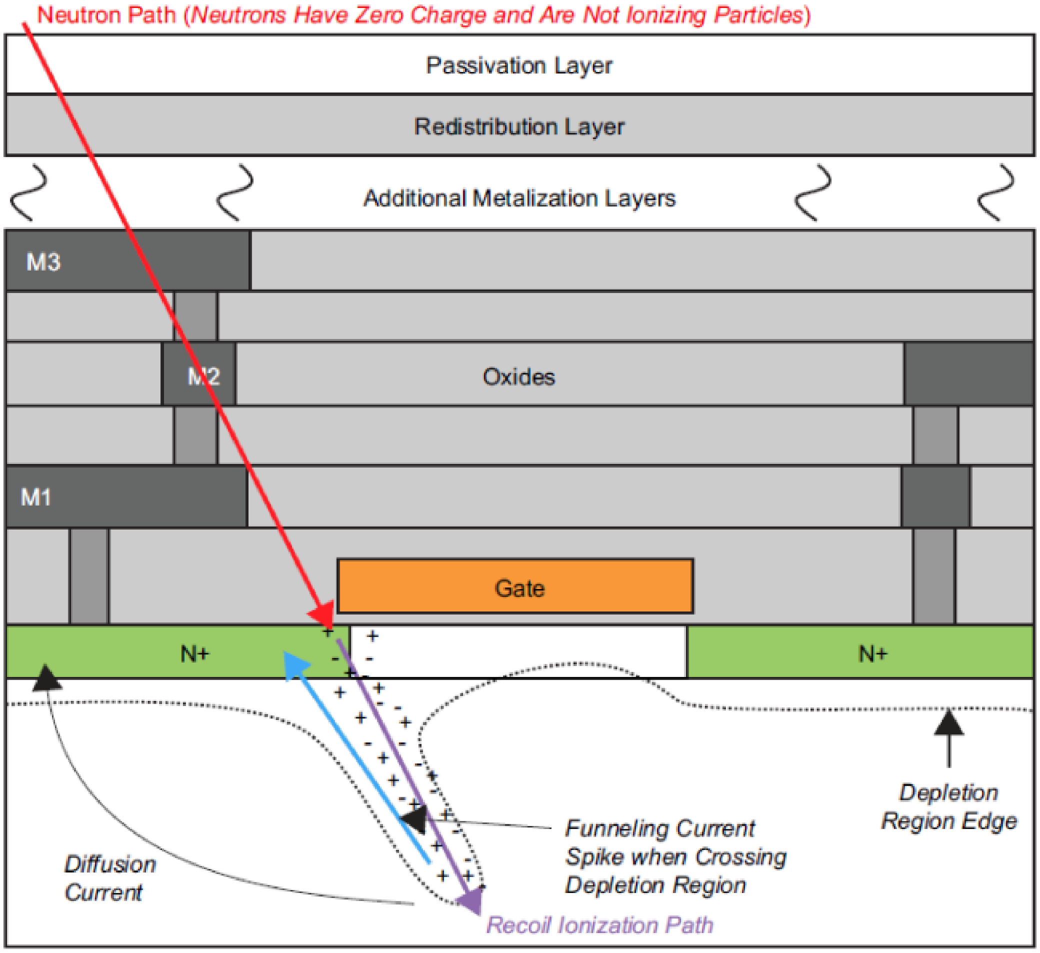
2.1 SET Causes

In normal conditions, outer space has highly energetic rays. These rays comprise high-energy particles (89% protons, 10% helium nuclei, and 1% other nuclei). These energetic particles are called cosmic rays. When cosmic rays enter the earth’s atmosphere, it interacts with atoms in the atmosphere producing highly energetic neutrons. These neutrons tend to collide with other molecules in the atmospheric layers, but it reaches the earth's surface. So the neutron flux, which is the number of neutrons per cm^2 per hour, is high near the space and low on the surface. The number of neutron fluxes is exponentially increasing when the attitude is increasing. For example, Table 2 represents the neutron flux recorded in New York City[5]. This is why, at high altitudes, airplanes are more exposed to SETs than computers.

Table 2 Influence of attitude in neutron flux in New York[18]

Attitude (feet)	Neutron flux ($\frac{n}{cm^2 h}$)	Neutron flux relative to the sea-level
0	14	1
1000	18.2	1.3
2000	23.4	1.7
3000	29.9	2.1
4000	37.9	2.7
5000	47.6	3.4
10000	134.6	9.6
12500	212.5	15.2
15000	322.6	23
17500	472.4	33.7
20000	668.5	47.8
22500	916.7	65.5
25000	1220.9	87.2
30000	2001.1	142.9

These neutrons have no charge, but it is very energetic. So they can collide with the molecules in the die, causing energetic ions to flow. These ions can cause a charge distribution, producing an electron-hole current when a neutron hits near the gate of a transistor. For example, FIGURE 10 represents an energetic neutron hitting near the device gate, causing charge imbalance and creating electron-hole pairs in the substrate. These electron-hole pairs cause the depletion region to deform, causing a funneling current spike in the depletion region, which leads to diffusion current, which can cause the device to work in an unwanted manner.



Neutron Path (Neutrons Have Zero Charge and Are Not Ionizing Particles)
Passivation Layer
Redistribution Layer
Additional Metalization Layers
M3
M2
Oxides
M1
Gate
N+
N+
Depletion Region Edge
Funneling Current Spike when Crossing Depletion Region
Diffusion Current
Recoil Ionization Path

FIGURE 10 An energetic neutron hits an NMOS Device

Another possible source of SET is the alpha particles. Alpha particles are mainly produced from materials adjacent to a chip, solder, and packaging materials. Alpha particles are driven from the Back End Of Line (BEOL) metalization and packaging material. FIGURE 11 represents the typical packaging used in CMOS, as shown lead solder and plastic packaging are used, which could emit alpha particles. For example, Lead (Pb-210), which is an isotope from stable Pb and used in packaging, is chemically indistinguishable from normal Pb, but Pb-210 is a huge contributor to alpha particle generation in CMOS. So, some engineers suggest avoiding using lead in CMOS fabrication. However, lead-free solder does not guarantee a chip that does not have alpha particles. For example, tin (Sn), used as a replacement for lead, has alpha particles component as it has impurities that are generated during refining. In fact, IBM found that Sn emits as many alpha particles as Pb or higher. There are continuous efforts to have as low alpha-emitting solder as possible. The typical alpha emission rates are 5-50 $\frac{\boldsymbol{n}}{\boldsymbol{cm^2h}}$[19]. Another source of alpha particles can be packing material such as underfill, overmolds, organic packages, and ceramic packages could be a source of alpha-particle. The typical range of typical alpha emission rates is 1-5 $\frac{\boldsymbol{n}}{\boldsymbol{cm^2h}}$. [19].

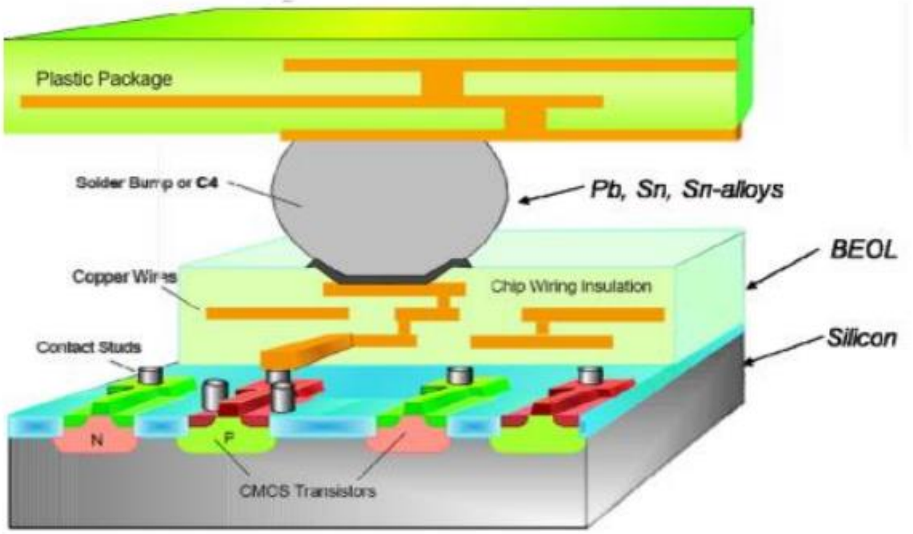


FIGURE 11 Packaging in CMOS circuit[20]

These alpha particles cause direct ionization in the active device and cause hole-electronic pair, which causes the depletion region to deform, causing a funneling current spike in the depletion region and leading to the diffusion current similar to neutron particles, as seen in FIGURE 12.

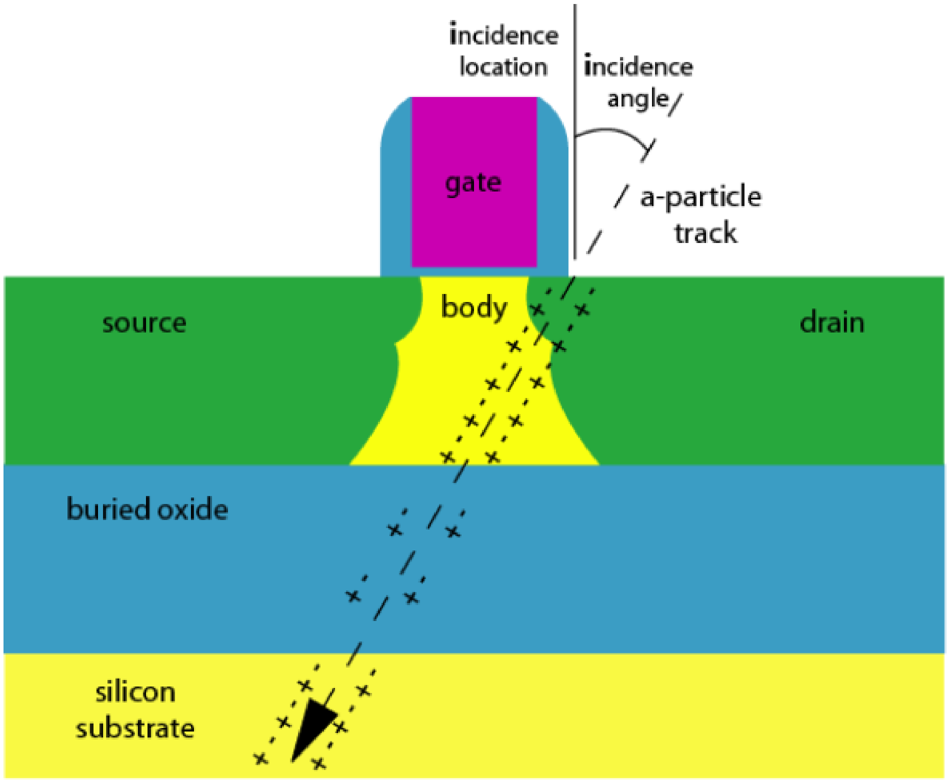


FIGURE 12 A charged alpha particle hits an active device[20]

2.2 SET Modeling

As seen in the previous section, SET causes undesired currents to flow in the affected node, which changes the voltage of the affected node due to the trapped charges in the substrate. If the change is big enough, it can cause the logical voltage of the node to flip. This node-flipped voltage can cause the fault to propagate through standard cells in a typical IC in a way similar to stuck-at fault but for a limited time. This fault is called Single Event Upset (SEU), happening when the fault of the output of the gate is flipped from 0 to 1 or from 1 to 0 as seen in FIGURE 13. When SET happens in the NMOS device while it is OFF, a current I_{set} follows across the terminal (as if NMOS was ON), causing current from the load. So that the charge across the capacitor of the output load decreases, leading to decreasing the voltage across the capacitor. If the $I_{set}$was big enough, it could take enough charges across the capacitor to flip the logic from 1 to 0. This amount of charge is called the critical charge $Q_{critical}$. So, this causes the SEU to propagate from input to output. This operation can happen throughout the circuit until it reaches the observable output[21].

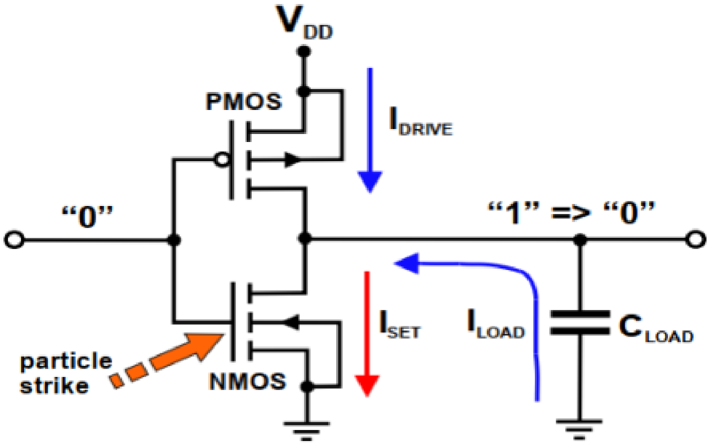


FIGURE 13 SET propagation to output in inverter[21]

SET modeling can be divided into:

1- Device modeling.

2- Circuit macro modeling.

3- Circuit micro modeling.

4- Mixed mode Simulation.

2.2.1 SET Device modeling

Device modeling is the most accurate method to model SET. It is based on TCAD simulation. The device is drawn in TCAD, which will have all the characteristics of the device, like doping concentration, gate length, the shape of the device, the material used, etc. In addition, information about the neutron particle's energy and position must be known. For example, in [22], Katunin et al. simulated the effect of SET on a logical element of matching on the Spaced Transistor Groups Digital Integrated Circuit Elements (STG DICE), which is a way in which transistors are arranged in groups with empty spaces in between as shown in FIGURE 14. The SET was injected in different positions, and the SET of 20–60

MeV×cm2/mg was simulated. Then for input 0101 the output was recorded.

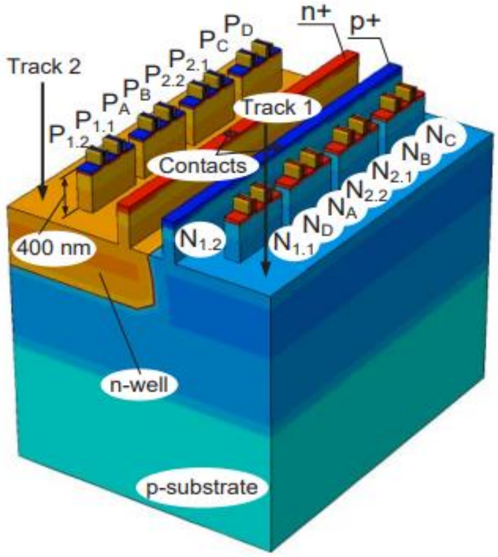


FIGURE 14 STG DICE TCAD Simulation[22]

FIGURE 15 represents the simulation of the output when the input was 0101, and the output was expected to be 1 when the SET source had 20-60 MeV×cm2/mg. As expected, it is noticed that the output voltage amplitude and pulse width increase with particle energy.

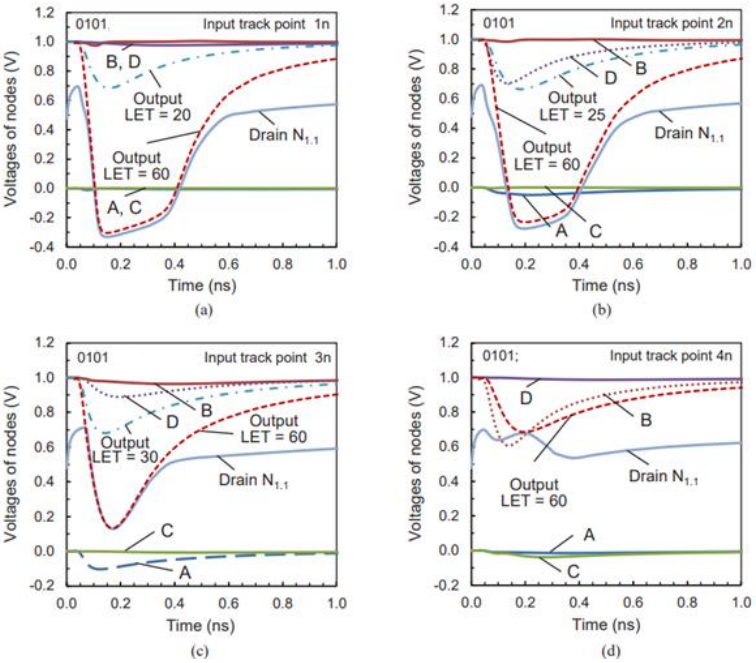


FIGURE 15 Output of STG when changing the location and the energy of the hitting particles[22]

2.2.2 SET Macro Modeling (Gate-Level)

The macro modeling is the simplest form to model SET as it is based on injecting current between different gates without modifying the gate structure or the gate terminals, as seen in FIGURE 16. Because of its simplicity of implementation using TCAD simulation, it is the most common model to use.

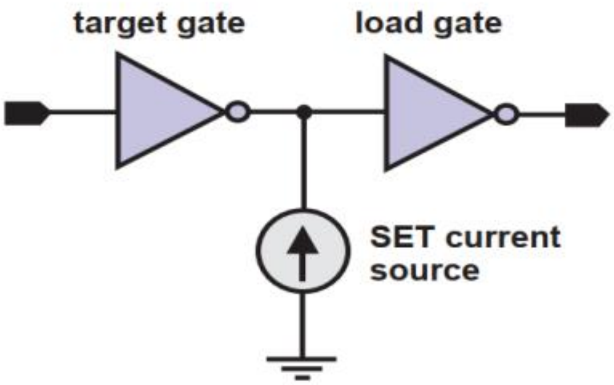


FIGURE 16 SET Macro modeling [21]

2.2.3 SET Micro Modeling (Transistor-Level)

The micro-modeling of SET is the transistor-level model. It simulates the actual SET by a current source; for example, if SET hits the drain of a transistor, it means that there are electron-hole pairs resulting in funneling current in the substrate. So, a current source between the drain and substrate is added to model this phenomenon, as seen in FIGURE 17 [21]. Unfortunately, this needs editing in the transistor models, which is restricted mostly to foundries.

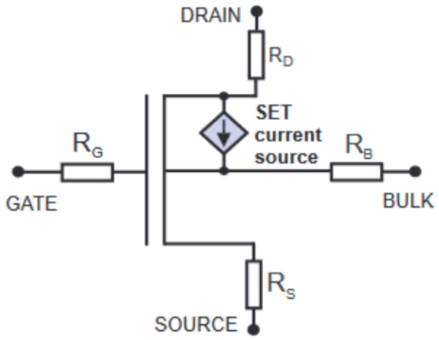


FIGURE 17 Micro-Modeling of SET[21]

2.2.4 Mixed Mode Simulation

The mixed signal mode depends on simulating the SET on the needed device using TCAD simulation and the rest of the design in SPICE. For example, in [23], Dodd made a mixed simulation between the SPICE simulator and TCAD simulator called Davinci, as seen in FIGURE 18. A SET strike with linear energy transfer (LET) varies from 1-30 Mev-cm^2/mg and is simulated to find the critical particle energy that will cause SET at the output, as seen in FIGURE 19 for bulk and Silicon On Insulator (SOI). The main problem with this method is that it requires doping, device material, and fabrication parameters, which is a foundry secret.

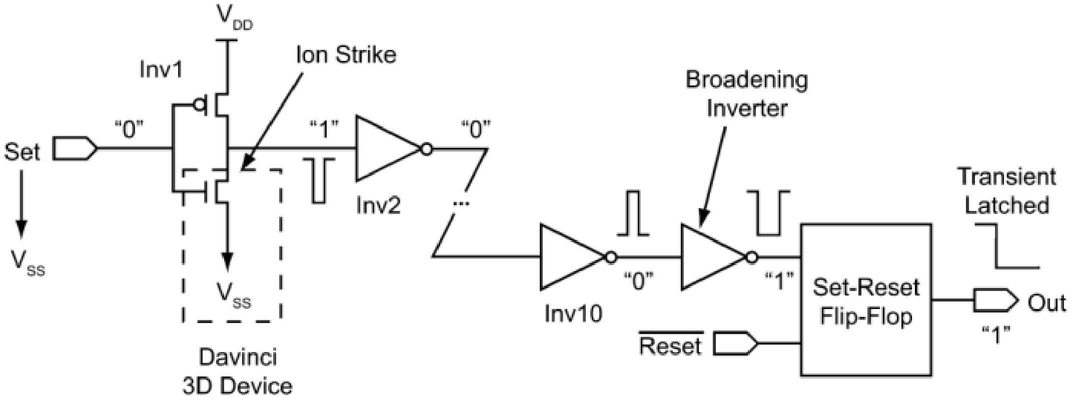


FIGURE 18 Mixed-level simulation of 10 cascaded inverters[23]

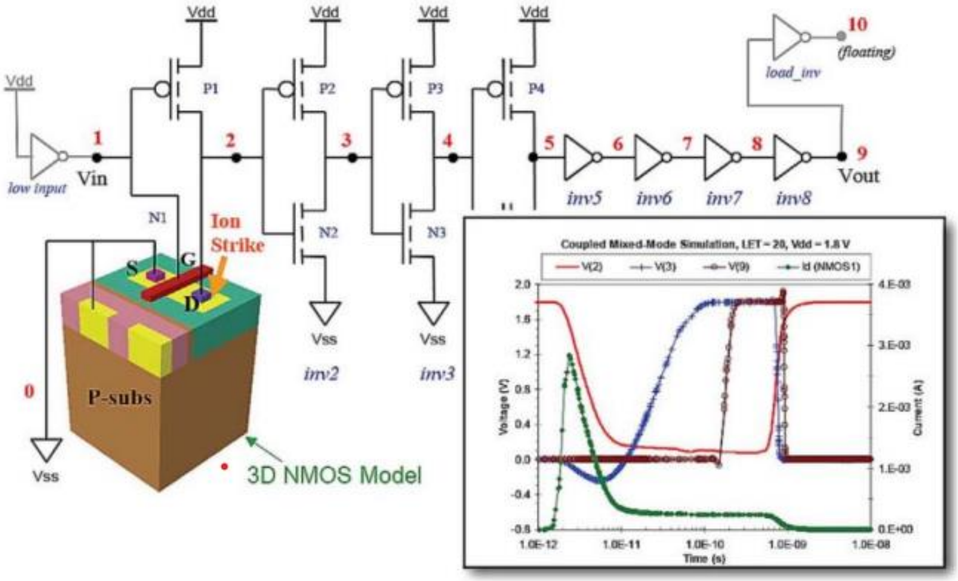


FIGURE 19 Mixed model simulation between 3D NMos and SPICE Simulation[23]

The struck drain voltage was recorded for both SOI and bulk process. It was found that the more energy the ions have, the more amplitude and pulse-width time they will cause. In addition, It was found that SOI is more resistant to SOI than bulk, especially the amplitude, as seen in FIGURE 20 [23]. Moreover, as seen in the next section, this experiment shows how the SET should be represented in SPICE.

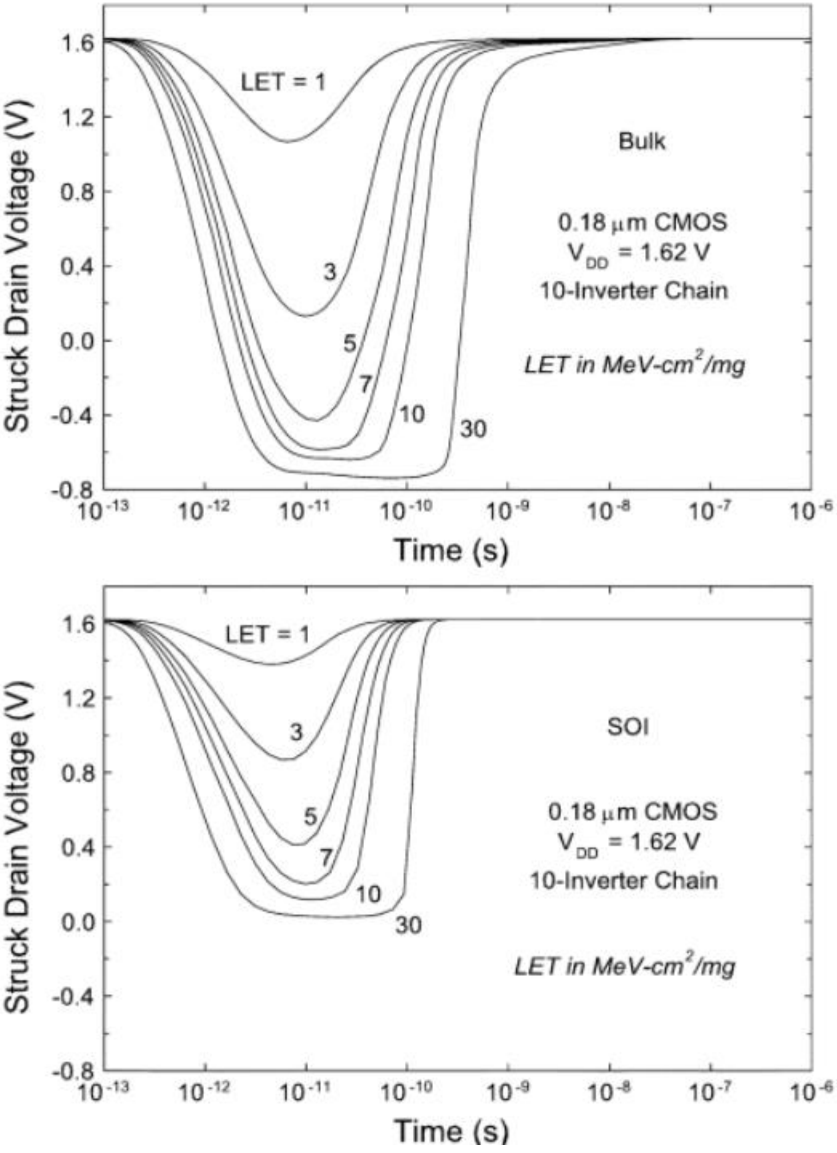


FIGURE 20 Stuck Drain voltage for different LET[23]

Then, the propagation of the effect of the SET was recorded to see the effect of cascading, as shown in FIGURE 21. It was found that cascaded inverters help in shielding the effect in SET. And for the 10 cascaded inverters, after 7 Mev-cm2/mg, the effect of SET appears at the observable output causing logical fault[23].

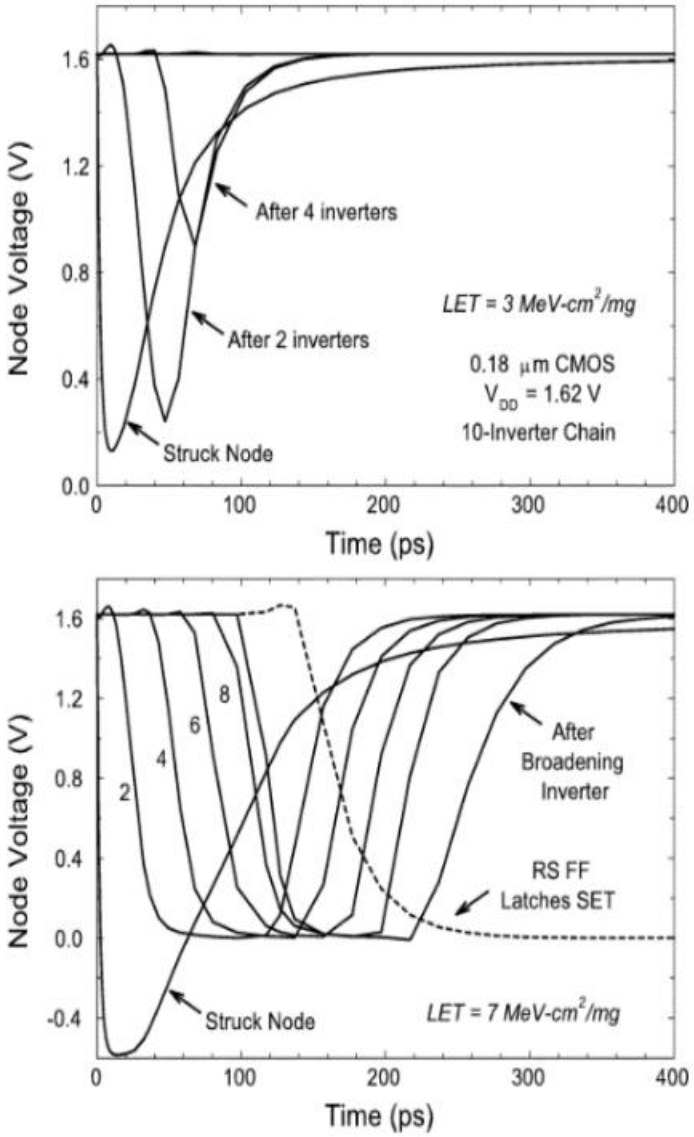


FIGURE 21 Effect of Propagation through gates [23]

2.3 SET Pulse Models

There are two methods to model the SET:

1- The current pulse model that is used micro and macro modeling.

2- The voltage pulse modeling that can be used in macro modeling instead of the current source.

2.3.1 Current source model

Double-Exponential Current Model is the most commonly used model for the SET current source. It was developed by [24] as follows

$$I_{SET}(t) = I_0\left(e^{\alpha t} - e^{-\beta t}\right) \tag{5}$$

Where I_0 is the amplitude of the pulse height, and α is the reciprocal of the fall time of the junction in FIGURE 22. And β is the reciprocal of the time constant for establishing the ion track.

$$\beta = \frac{k\varepsilon_0\varepsilon_r}{q\mu N_D} \tag{6}$$

Where k in the Boltzmann universal constant, N_D is the donor dopant concentration, $\varepsilon_0\varepsilon_r$ is the permittivity of the junction, μ is the electron mobility, and q is the single-electron charge.

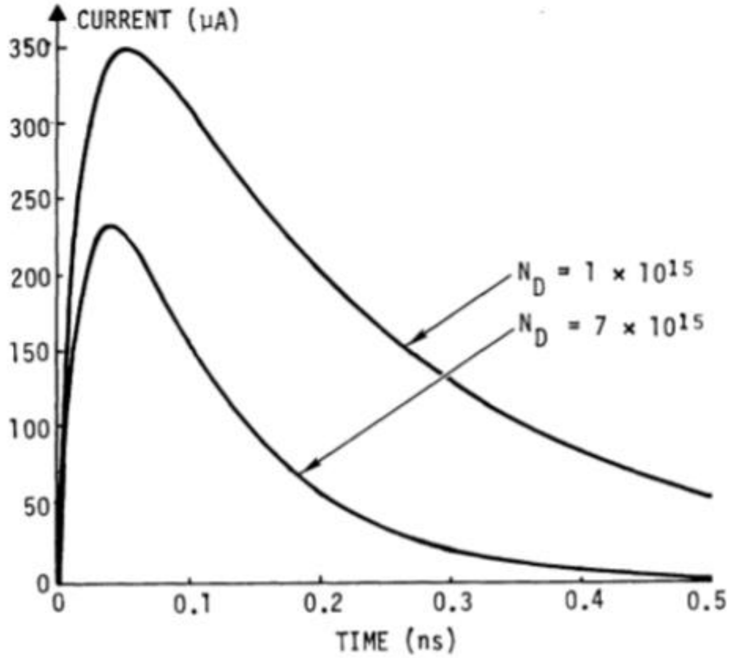


FIGURE 22 Double-Exponential Current Model [24]

2.3.2 Double Sinusoidal Voltage Pulse

As shown in the previous section, the SET causes the drain voltage to be like an imperfect square wave, as shown in FIGURE 23. This idea was proposed by [25].

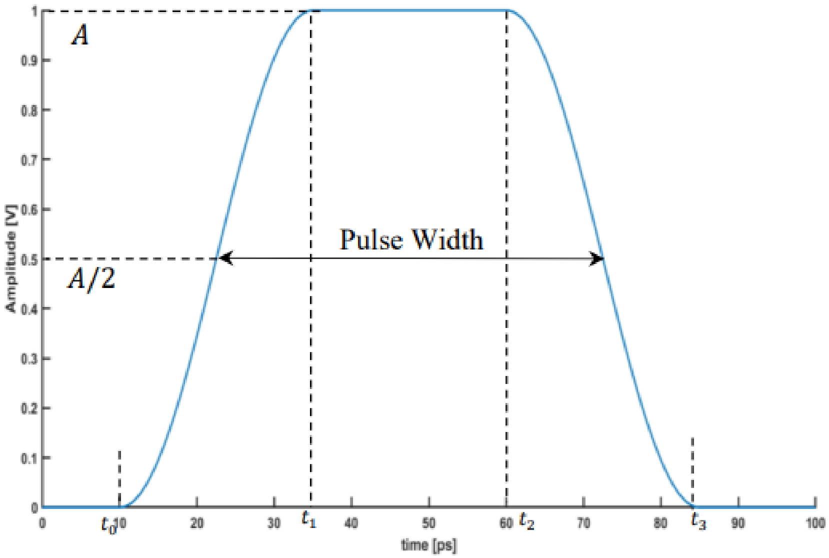


FIGURE 23 Double sinusoidal Voltage Pulse[25]

Equation (7) represents the Double Sinusoidal Voltage Pulse where A is the maximum voltage value, and the pulse width is when $V(t) = \frac{A}{2}$. Parameters t_0, t_1, t_2, t_3 *and* ω are parameters to define the SET shape.

$$V(t) = \begin{cases} 0 & t \le t_0 \\ \frac{A}{2}\left(\sin\left(\omega(t - t_0) - \frac{\pi}{2}\right) + 1\right) & t_0 \le t \le t_1 \\ A & t_2 \le t \le t_3 \\ \frac{A}{2}\left(\sin\left(\omega(t - t_2) + \frac{\pi}{2}\right) + 1\right) & t_2 \le t \le t_3 \\ 0 & t_3 \le t \end{cases} \tag{7}$$

Chapter 3

Characterization and SET Modeling for Standard Cell

This chapter illustrates how standard cells are characterized, which indicates how the amplitude and pulse width propagating from the input of the gate to its output can be predicted. Furthermore, the generated method is used to calculate the probability of SEU happening for each standard cell in chapter 5. If the output has the logic flipped, it is called SEU. If not, then the out will be resistant to particle hit. The characterization was done using the Double Sinusoidal Voltage Pulse and macro modeling of each standard cell.

The simulation was done for XFAB® XH018 technology, but the same methodology can be applied to any technology. Firstly the Double Sinusoidal Voltage Pulse was performed at the input of the gates, and the input voltage (V_{in}), input pulse width (tw_{in}) output voltage (V_{out}) and output pulse width (tw_{out}) were recorded. The circuit design was as FIGURE 24. Then, swapping V_{in} and tw_{in} was done and the V_{out} and tw_{out}values were recorded accordingly. Finally, prediction models were done using [26] and [27] to best fit to predict any V_{in} and TW_{in}.

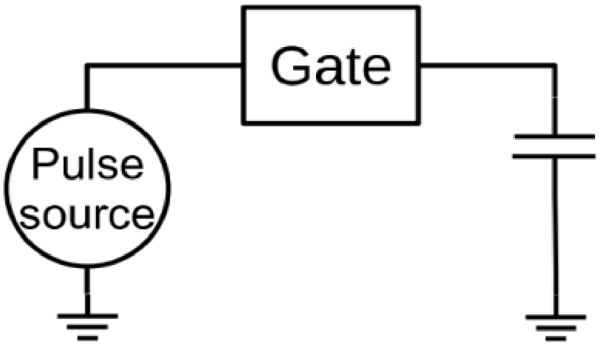


FIGURE 24 Standcell characterization testbench[27]

Fault Modeling

This chapter will discuss two models of the output voltage (V_{out}) and output pulse width (tw_{out}) in terms of the input voltage (V_{in}) and pulse width. (tw_{in}).

3.1 Analytical Model

The first model is an analytical model developed in [26]. The model describes the relationship between the output voltage and input voltages as a sigmoid surface as follows

$$\frac{V_{out}}{V_{dd}} = \frac{1}{1 + e^{-k(V_{in}-V_0)}} \tag{8}$$

Where k and V_0 determine the shape of the sigmoid surface and can be found as follows:

$$k = c\left(1 - e^{-\frac{tw_{in}}{T}}\right) \tag{9}$$

$$V_0 = V_{DC}\left(1 + \left(\frac{t_{d1}}{tw_{in}}\right)^{\alpha}\right) \tag{10}$$

Where V_{DC} is the voltage value where the input voltage is the same as the output voltage. The other coefficients $c,\ \alpha,\ T,\ t_{d1}$ are the fitting parameters obtained by fitting the sigmoid surface. These parameters differ between each standard cell gate.

Equation (11) predicts the output pulse width (tw_{out})) in terms of the input voltage (V_{in}) and pulse width. (tw_{in}) as follows:

$$tw_{out} = a\, tw_{in} + t_0\, e^{\frac{-tw_{in}}{t_i}} + b \tag{11}$$

Where a and b can be found as follows:

$$a = a_0 + a_1 . V_{in} \tag{12}$$

$$b = b_0 + b_1 . V_{in} \tag{13}$$

Where a_0 , a_1 , b_0 , b_1 are fitting parameters. Typical plots of the voltage and pulse width as seen in FIGURE 25 [26].

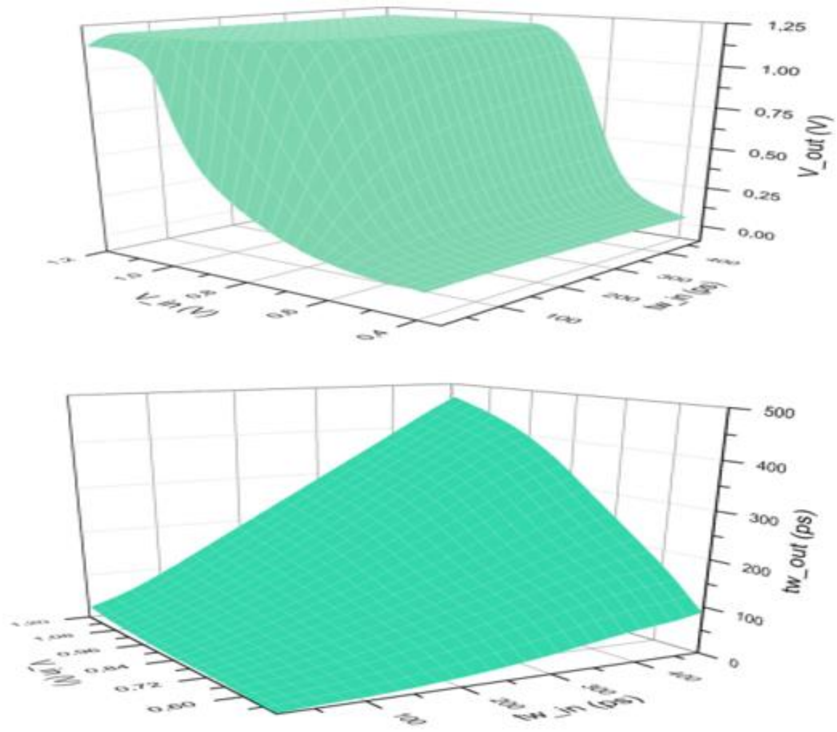


FIGURE 25 Voltage and pulse width transfer function[26]

So for each new technology, simulation must be done using gate-level simulation like Cadence Virtuoso. The model of FIGURE 24 is used, and the value of capacitance is assumed to be a minimum-size inverter which provides the worst-case scenario as increasing the capacitance value will attenuate the SET effect. Then input voltage and pulse width values are swept, and output voltage and pulse width are recorded. Finally, best-fitting techniques are used to get each standard cell's parameter values. So, for each new input voltage and pulse width, the model of [26] can predict the output voltage and pulse width.

3.2 Cubic Interpolation Model

A new numerical model was done by [27]. In his model, after simulating every standard cell using Cadence Virtuoso using ocean script by sweeping the input voltage and pulse width, the data is given to MATLAB, and data fitting was done using cubic interpolation as it gives smooth interpolation with minimal error. Finally, all the standard cell models are saved as a library that can be used to predict the output pulse width and voltage.

In general, cubic interpolation gives more accurate results than [26]. In [27]'s work, a comparison between the two models was made foe Xfab 65nm technology. For example, FIGURE 26 indicates the difference between models [26] and [27]. While [26] smooths the surface and gives an analytical prediction for output voltage and pulse width, it has more error than [27].

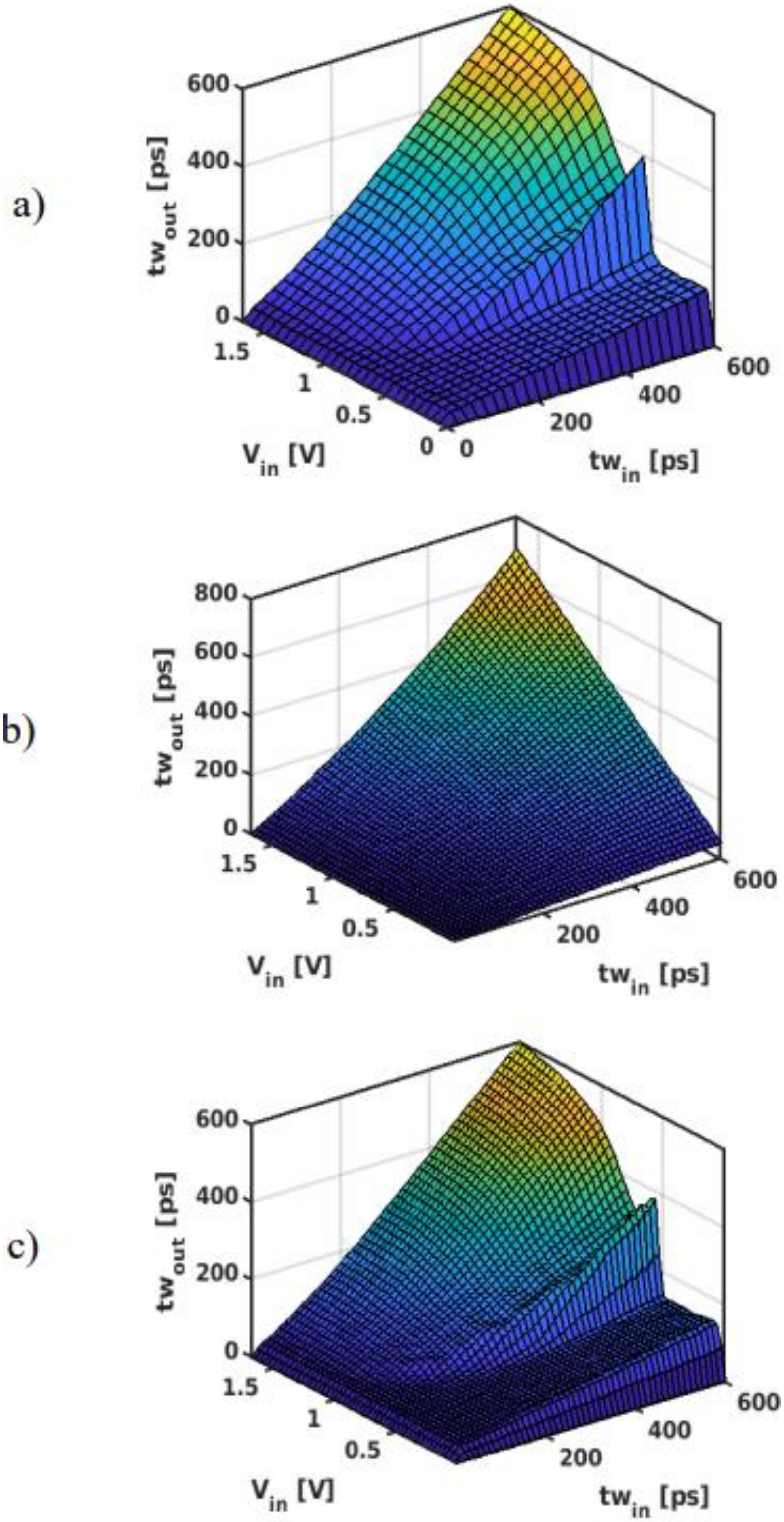


FIGURE 26 Pulse output width from NO2HDSVTX4 cell characterization. (a) simulation result, (b) model from [26], and (c) cubic interpolation.[27]

Chapter 4

ASIC Circuit Probability of Error and Worst Case Vector

This chapter illustrates how a complete ASIC combinational circuit is characterized, which indicates how the probability of error is propagated from inputs to observable outputs based on the probability of error of each standard cell. Then, the overall probability is calculated using the two methods. The first method is developed in [29] by Krishnaswamy. The second method is the novel method developed in this thesis.

In order to find the probability of error of a certain ASIC circuit, both methods used the same probabilistic transfer matrix (PTM) and ideal transfer matrix (ITM) to express the probability of error in a standard cell and the truth table of the standard cell.

4.1 PTM and ITM

The probability Transfer Matrix (PTM) was first used in the context of conditional probability theory [30]. This idea used a simple matrix-based method to convey the probability of error corresponding with each input. The PTM essentially offers a practical method for expressing conditional probabilities as matrices, making analyzing complex systems easier, especially in digital circuits and information theory. The PTM's capacity to express the probability of error for each input is one of its main features. It is possible to get insights into conditional probabilities by displaying them in a matrix format. PTM dimension is 2^N X 2^M where N and M are the numbers of inputs and output, respectively, each element of the PTM represents the conditional probability for a certain output represented by that column number, given that a certain input, represented by row number, has occurred $\left(PTM_{i,k} = P(out = k - 1 \mid input = i - 1\right)$. The Ideal Transfer Matrix (ITM) uses the same fundamental idea as the PTM but presupposes that each standard cell has a 0% probability of fault. The ITM may be used to determine a system's ideal performance. ITM is directly related to the truth table of each standard cell, as the P(1) column is the truth table of the standard cell, whereas P(0) is complementary to the truth table. For example, equation (14) represents the ITM and PTM of the 2-input AND gate. The element (1,1) represents the conditional probability that the output is 0 given that the input is 0,0 $(PTM_{11} = P(out = 0\ |input = 0{,}0)$[31]. PTM can be done easily by first getting the ITM from the truth table, then replacing each zero with p and one with $1 - p$ where p is probability of error that will be explained in chapter 5. Table 3 represents the PTM and ITM of the commonly used standard cells.

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} (1-p) & p \\ (1-p) & p \\ (1-p) & p \\ p & (1-p) \end{bmatrix} \tag{14}$$

Table 3 PTM and ITM for famous commonly used cell

Standard Cell	ITM	PTM
2-Input AND	$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1-p & p \\ 1-p & p \\ 1-p & p \\ p & 1-p \end{bmatrix}$
2-Input NAND	$\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} p & 1-p \\ p & 1-p \\ p & 1-p \\ 1-p & p \end{bmatrix}$
2-Input OR	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1-p & p \\ p & 1-p \\ p & 1-p \\ p & 1-p \end{bmatrix}$
2-Input NOR	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} p & 1-p \\ 1-p & p \\ 1-p & p \\ 1-p & p \end{bmatrix}$
2-Input XOR	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1-p & p \\ p & 1-p \\ p & 1-p \\ 1-p & p \end{bmatrix}$
2-Input XNOR	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} p & 1-p \\ 1-p & p \\ 1-p & p \\ p & 1-p \end{bmatrix}$
Buffer	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$
Inverter	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}$

So, In the current work, a MATLAB code was built to analyze buffer, inverter, N-input AND, N-input OR, N-input XOR, N-input NAND, N-input NOR, and N-input XNOR for XFAB® XH018 technology. FIGURE 27 represents the proposed PTM generation flow used in MATLAB in this work. Firstly, normal random input vectors for each standard cell used in the design are generated. Then each cell is characterized to get the probability of error. Then, ITM was generated by the truth table for each cell. Finally, PTM is generated from ITM and this standard cell's corresponding probability of error.

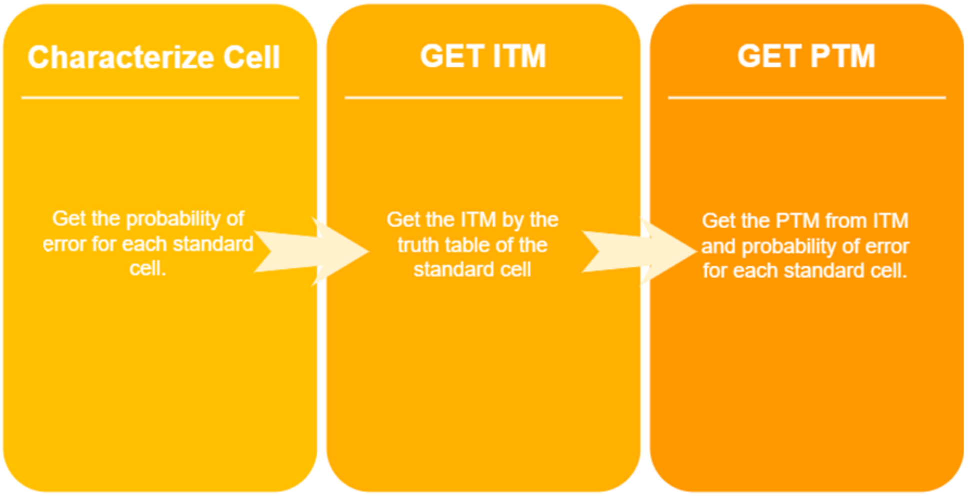
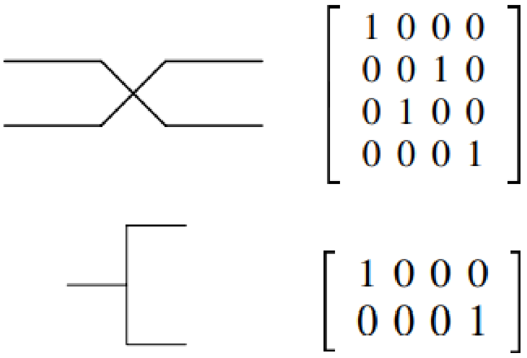

Characterize Cell
Get the probability of error for each standard cell.
GET ITM
Get the ITM by the truth table of the standard cell
GET PTM
Get the PTM from ITM and probability of error for each standard cell.

FIGURE 27 PTM generation flow for each standard cell

It is worth noting that Krishnaswamy's method requires some special ITM to be calculated. These special ITMs are for fanout and wire swapping, as shown in FIGURE 28 [29]. In addition, gerenic fanout detection and wire swapping method were done to get the ITM for these special ITM.



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

FIGURE 28 special ITM for wire swapping and fanout.[29]

4.2 Krishnaswamy's Method

Krishnasway depends on the PTM algebra, which uses the PTM and ITM for each standard cell. Then, using matrix operation, the accumulative PTM and ITM for ASIC circuit.

4.2.1 PTM and ITM Algebra

In order to calculate the full circuit PTM and ITM, The pre-characterized PTM/ITM will be used. If two cells g_1, g_2 with PTM $P_1 and\, P_2$ are in series, their combined PTM will be $P_1\, P_2$ as equation (15) indicates.

$$P_{total} = P_1\, P_2 \tag{15}$$

For example, if two buffers are cascaded as in FIGURE 29. The total PTM of the system will be

$$P_{total} = P_1\, P_2 = \begin{bmatrix} 0.82 & 0.18 \\ 0.18 & 0.82 \end{bmatrix}$$

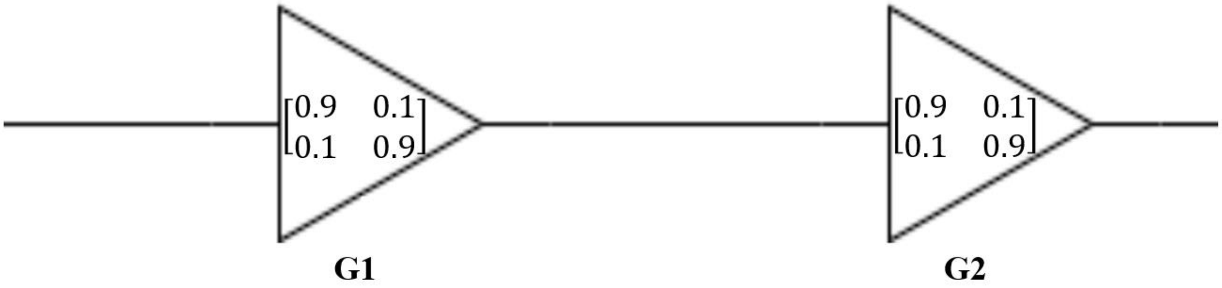

$\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$
$\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$
G1
G2

FIGURE 29 Cascaded buffer PTM example

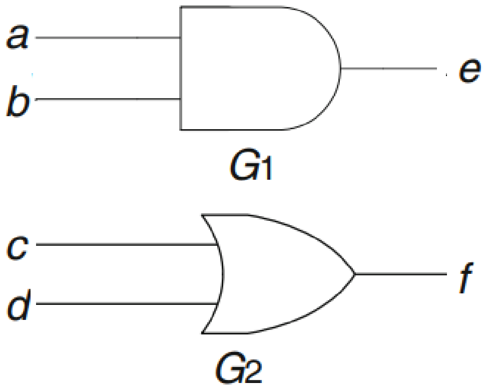
This makes sense as increasing the depth of a path makes it more probable to get hit by SET, increasing the probability of error.

On the other hand, if the cells were parallel, their combined PTM or ITM would be $P_1 \otimes P_2$ where $\otimes$ represents the Kronecker tensor product[29] as (16) indicates.. The Kronecker tensor product combines two matrices to form a larger matrix representing the transfer matrix for a composite system consisting of multiple subsystems.

$$P_{total} = P_1 \otimes P_2 \tag{16}$$

For example, FIGURE 30 indicates the PTM calculation for 2-Input AND in parallel with 2-

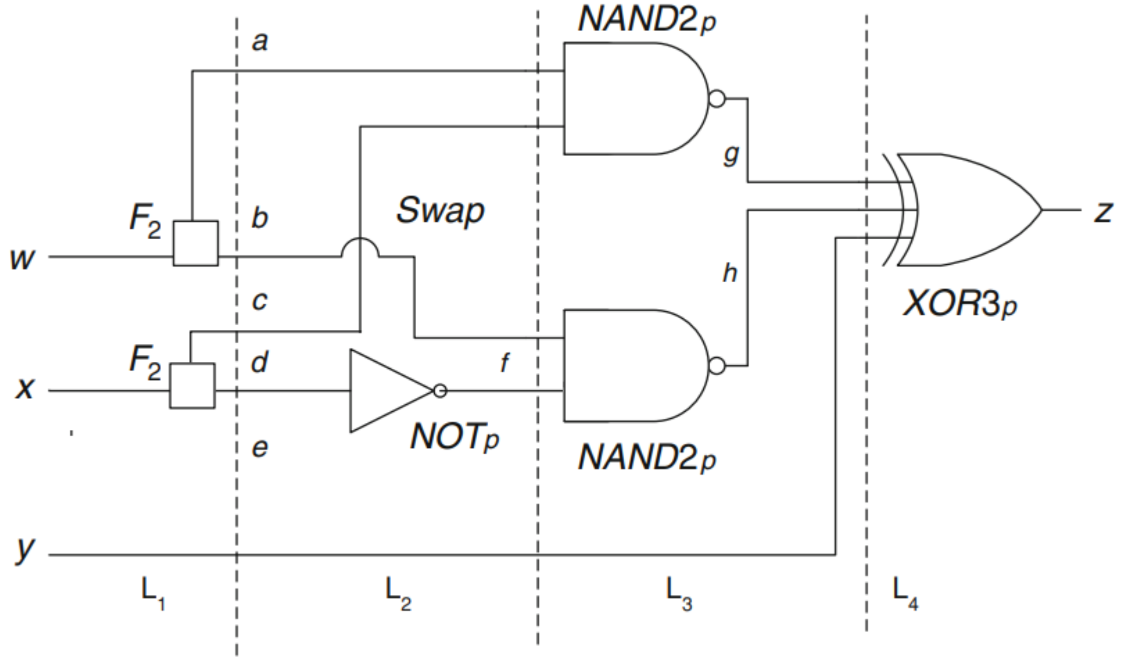
Input OR. The resultant matrix is now 16x4, as excepted, as the system has 4 inputs and 2 outputs.



$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

FIGURE 30 Example of PTM calculation of 2 parallel gates

In order to calculate a complete combinational circuit, Krishnasway proposed that the PTM and ITM of each standard cell, wire-swapping, and fanout will be calculated first. Then, the circuit will be divided into levels; each level will be composed of a parallel element that has ITM or PTM. Then, for each level, the resultant PTM or ITM will be calculated using equation (16). Then each level will be composed of cascaded subsystems, so the overall PTM or ITM of the system will be calculated by equation (15). For example, FIGURE 31 represents a complex system. The first step is to detect the fanout and swapped wires, get their ITMs, treat them as standard cells, and get the PTM and ITM of standard cells. Then, a partitioning algorithm must be done to get cells that are in parallel. Then, for each level, PTM and ITM are calculated by parallel rule. Finally, a cascaded rule will be implied to get the whole circuit ITM and PTM.



$$(F_2 \otimes F_2 \otimes I)(I \otimes swap \otimes NOT_p \otimes I)(NAND2_p \otimes NAND2_p \otimes I)(XOR3_p)$$

FIGURE 31 A completed combinational circuit PTM calculated example[31]

As seen before, the algorithm needs special ITM calculation of generic fanout and wire-swapping in a circuit. In addition, matrix multiplication and manipulation are big computational problems for big circuits.

4.2.2 Circuit Fidelity

Circuit fidelity is used to evaluate the reliability of a circuit. It measures the similarity between an ITM and a corresponding PTM of a certain system. Therefore, it is a measure of the reliability of the circuit. It can be done by element multiplication (dot product) of M and J equation (17) states, where M, J are the PTM and ITM of the whole circuit, respectively, and **v** is a row vector that represents the probability of input. For example, if the circuit in FIGURE 31 has only 000 and 111 as input with the same probability, the v will be [0.5 0 0 0 0 0 0 0.5] [31].

$$fidelity(v, M, J) = \|\mathbf{v}(M.* J)\| \tag{17}$$

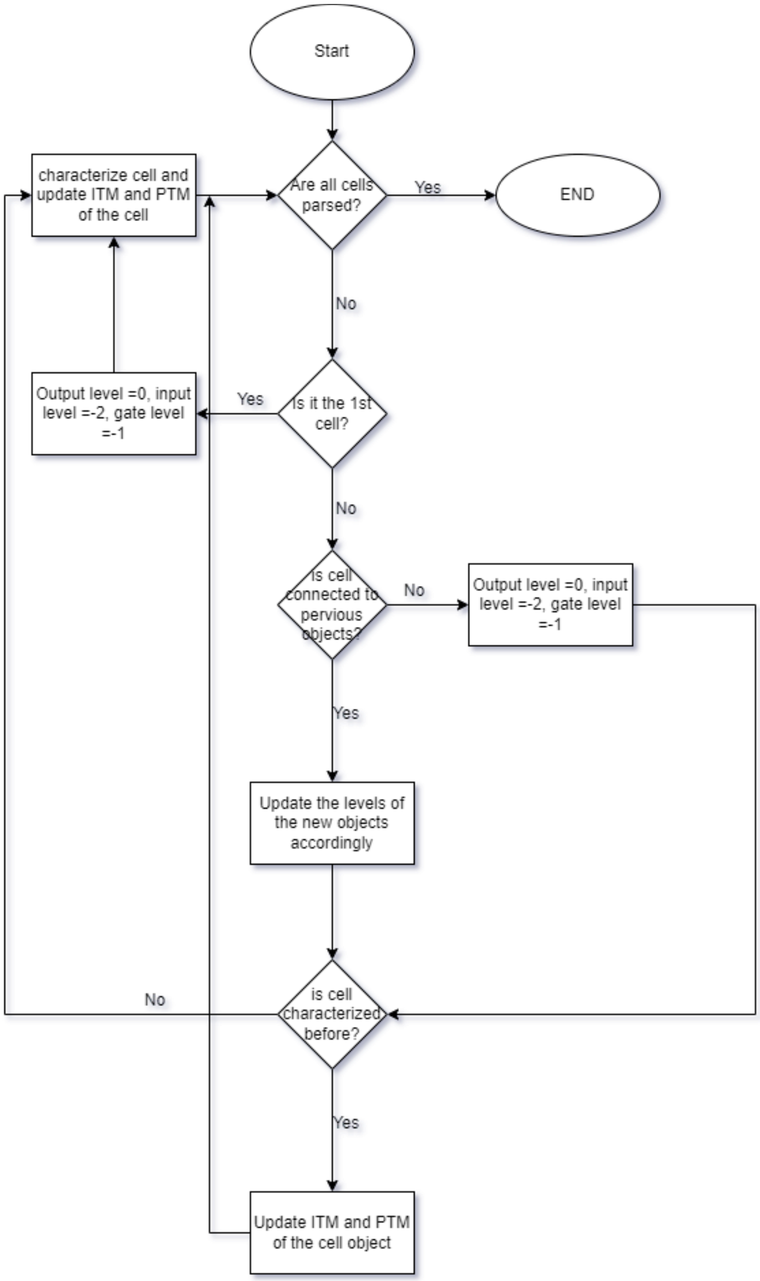
$$\mathrm{P_{err}} = 1 - \|\mathbf{v}(M.* J)\| \tag{18}$$

In order to calculate the probability of error of a certain input, it will be 1-fidelity, as indicated in (18).

4.2.3 Algorithm Implementation

In order to implement the previous algorithm, a gate-level Verilog parser was implemented. The code parses the gate-level netlist three times. The 1st time is to get the circuit hierarchy, get the standard cells used in the design, and characterize them. The second time is to manipulate the fanout and treat them as a dummy gate with ITM only. Finally, the third time is to make sure of the circuit hierarchy with fanout and wire-swapping. Each gate and node is considered an object with ITM, PTM, and level properties.

The first time, the first gate's output node is assumed to be in level 0, so the gate will be in its previous level (level -1), and its input will be in level -2. A normal node, which is not a fanout of swapped, will have PTM and ITM of identity, whereas the gate will have ITM and PTM characterized by the code. Characterization parameters were as follows; It is assumed that the average amplitude of the SET input is 0.9 V (which is half of VDD), with a deviation of 0.1 V. Additionally, the average width of the SET input is 300 ps with a deviation of 100 ps. In addition, the critical voltage and pulse width are 0.9 V and 300 ps, respectively. Then, any new gate parsed will have a new level based on its connection until a full circuit hierarchy is known. For example, if a new gate is connected to the output of the 1st gate, the gate object will have (level =2) and so on. If a gate is connected to nodes that do not level property as the input nodes are not connect to any of the previous objects, it is assumed to be in level 0, so the gate will be in its previous level (level -1), and its input will be in level -2. This assumption will be verified and corrected in the last round of parsing. FIGURE 32 represents the flowchart of the parser.



Start
Are all cells parsed?
Yes
END
characterize cell and update ITM and PTM of the cell
No
Output level =0, input level =-2, gate level =-1
Yes
Is it the 1st cell?
No
Is cell connected to pervious objects?
No
Output level =0, input level =-2, gate level =-1
Yes
Update the levels of the new objects accordingly
No
is cell characterized before?
Yes
Update ITM and PTM of the cell object

FIGURE 32 The first round of the parser flow chart

In the second round of parsing, fanouts are detected. So dummy objects are added with the knowledge of the hierarchy. For example, if a dummy object is added to level -5, all the objects in -5 or below level will be reduced by 1. Furthermore, the fanout object will be in level -5 parallel with dummy objects representing a dummy node (with ITM of PTM of identity). Finally, in the final round of parsing, the new hierarchy is checked, and it is ensured that the inputs of any gate in the design are on the same level and the levels are consistent.

After parsing the circuit, some circuit manipulation is done in order to order the objects according to level and prepare ITM and PTM calculations. Finally, PTM and ITM calculations are done for each level by tensor product as they are parallel, and the PTM and ITM of the system will be the product of all levels of PTM and ITM. Finally, to get the circuit fidelity, all the input vectors are assumed to have the same probability of error. So, the fidelity (F) as a matrix will be as equation (19).

$$F(M, J) = M.* J \tag{19}$$

4.2.4 Algorithm Validation

As mentioned before, ITM represents the truth table of the circuit. So, a testbench for these circuits was built using SystemVerilog for serval gate-level netlists. A random input vector was tested for the SystemVerilog code (which is the golden model) and the MATLAB code. Then, the output vector of the SystemVerilog was compared to the ITM from MATLAB. A perfect match between the two codes was achieved. As the ITM calculation is the same for PTM, PTM and ITM calculation was verified.

4.2.5 Worst-Case Vector Exploration

The worst case vector is the input vector with the most likely to occur. So, the probability of error of input vectors will be found, and the most likely input vector that can cause SEU will be considered the worst-case vector. Alternatively, the algorithm could get the vectors that will have the lowest fidelity, as equation (20) indicates where V is in binary, M is the PTM, and J is the ITM of the circuit. So the worst-case vector will be the index that will have the lowest fidelity noting that the index of the matrix starts with zero.

$$V = \arg_{min} \|(M.* J)\| \tag{20}$$

For example, if all gates in the circuit in FIGURE 31 have a probability of error of 0.1, then

$$F(M,J) = M.*J = \begin{bmatrix} 0.756 & 0 \\ 0 & 0.756 \\ 0.756 & 0 \\ 0 & 0.756 \\ 0 & 0.705 \\ 0.705 & 0 \\ 0 & 0.705 \\ 0.705 & 0 \end{bmatrix}$$

So the worst case vector can be 000, 001, 010, 011 with a probability of error 0.244.

Finally, a MATLAB code was written to get the worst-case vector and the code was tested for ISCAS85 benchmark circuits.

Chapter 5

Standard Cell Probability of Error

In soft error, the SEU is not like a stuck-at error as it has a probability of occurrence in each environment. Previously, the standard cell probability did not account for logical masking and depended on the probability of the strike [31]. However, this chapter proposes a new method to obtain the probability of error in each standard cell is modeled that can be used for PTM analysis based on the normal distribution of SET voltage and pulse width. In addition, the new proposed method account for logical masking for each standard cell. Previously, the input voltage and pulse width of SET are assumed to be deterministic. However, the input voltage and pulse width are not deterministic as they change according to the particle energy and the hit's physical position. So, the Double Sinusoidal Voltage Pulse is modified as the model's amplitude was assumed to have a normal distribution with a known mean V_{mean} and standard division σ_V. In addition, the model's pulse width was assumed to have a normal distribution with a known mean tw_{mean} and standard division σ_{tw} as seen in FIGURE 33.

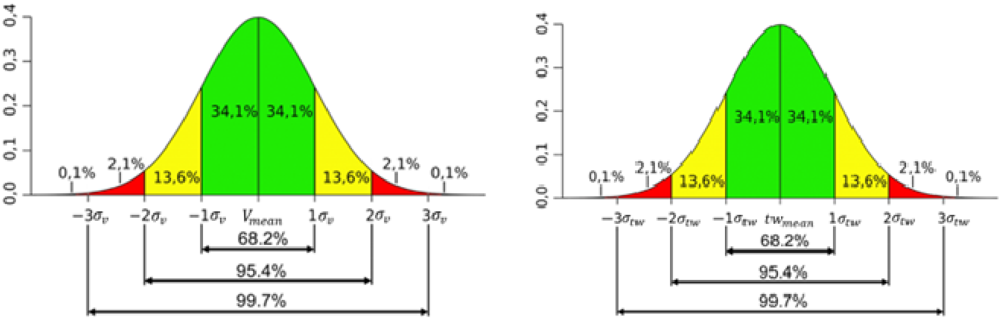


FIGURE 33 Voltage and pulse width normal distribution

So, the vector of input voltages and pulse widths that are normally distributed were generated using MATLAB. Then, for each input vector, the V_{out} and tw_{out} are calculated using [27] and then recorded. If these V_{out}, and tw_{out} is larger than the critical voltage and pulse width V_{cr}, tw_{cr} respectively, the input voltage will be recorded. Finally, the error probability will be the number of critical inputs that caused SET error over the total number of input vectors

$$P_{SET} = \frac{No\ of\ critical\ input\ vectors}{No\ of\ input\ vectors} \tag{21}$$

P_{SET} will be very pessimistic as it does not account for logical masking. Logical masking is when the input of a logic gate that has no logical fault propagates, as seen in FIGURE 34. For example, when one of the AND inputs is zero, the output will be zero regardless if there is a SET in any of its other inputs [28].

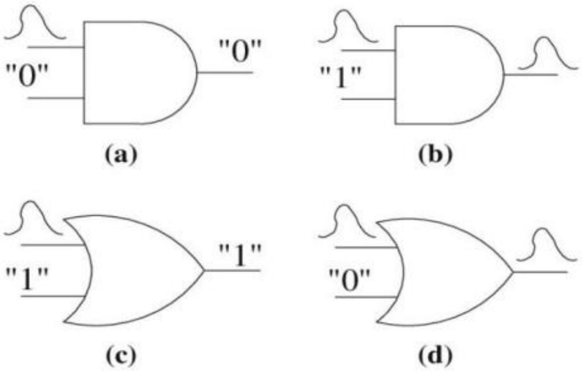


FIGURE 34 Illustration of logical masking[28]

So, to account for logical masking and get more accurate results, this thesis proposes including the probability that there is no logical masking.

5.1 AND gate probability of error

For a 2-input AND gate with A and B terminals as input, it is assumed that all the input has equal probability. Table 4 illustrates the calculation of the probability of error of the 2-AND gate. The proposed model considers all the possible transitions of inputs and only take care of the one that will cause the SET to propagate.

Table 4 AND gate probability of error calculation

A	B	out	SET	Prob of input
0→0	0→0	0→0	No	1/16
0→1	0→0	0→0	No	1/16
0→0	0→1	0→0	No	1/16
0→1	0→1	0→1	YES	1/16
0→0	1→1	0→0	No	1/16
0→1	1→1	0→1	YES	1/16
0→0	1→0	0→0	NO	1/16
0→1	1→0	0→0	NO	1/16
1→1	0→0	0→0	No	1/16
1→1	0→1	0→1	YES	1/16
1→0	0→0	0→0	NO	1/16
1→0	0→1	0→0	NO	1/16
1→1	1→1	1→1	NO	1/16
1→1	1→0	1→0	YES	1/16
1→0	1→1	1→0	YES	1/16
1→0	1→0	1→0	YES	1/16

For 2-input AND, SET will only happen if SET causes the input to go from A to go from 0 to 1 while B is constant at 1 or vice versa. In addition, multiple event-transient can happen if A, B are switched at the same time with a probability of P_{SET}^2 as the SET will be assumed independent. Furthermore, a closed-form expression was introduced N-input AND gate, the multiple events transients will be as (22).

$$P_{AND\ N} = 2 * \sum_{k=1}^{M} \frac{\binom{N}{k} {P_{SET}}^{k}}{2^{2N}} \tag{22}$$

Where M is the number of event-transient wanted to take into consideration, and N is the number of input terminals. M could take any value from 1 to N. In the current work, Single event transient was only. So $M = 1$ is used, and equation (22) will be reduced to (23).

$$P_{AND\ N} = \frac{N * P_{SET}}{2^{2N-1}} \tag{23}$$

5.2 NAND gate probability of error

For a 2- input NAND gate with A,B terminals as input, it is assumed that all the input has equal probability. Table 5 illustrates the calculation of the error probability of 2-NAND gate.

Table 5 NAND gate probability of error calculation

A	B	out	SET	Prob of input
0→0	0→0	1→1	No	1/16
0→1	0→0	1→1	No	1/16
0→0	0→1	1→1	No	1/16
0→1	0→1	1→0	YES	1/16
0→0	1→1	1→1	No	1/16
0→1	1→1	1→0	YES	1/16
0→0	1→0	1→1	NO	1/16
0→1	1→0	1→1	NO	1/16
1→1	0→0	1→1	No	1/16
1→1	0→1	1→0	YES	1/16
1→0	0→0	1→1	NO	1/16
1→0	0→1	1→1	NO	1/16
1→1	1→1	0→0	NO	1/16
1→1	1→0	0→1	YES	1/16
1→0	1→1	0→1	YES	1/16
1→0	1→0	0→1	YES	1/16

For 2-input NAND will have the same error probability formula as AND gate, and equations (22) and (23) will still be valid.

5.3 OR gate probability of error

For a 2- input OR gate with A,B terminals as input, it is assumed that all the input has equal probability. Table 6 illustrates the calculation of the probability of error of 2-OR gate. The proposed model considers all the possible transitions of inputs and only take care of the one that will cause the SET to propagate.

Table 6 2-input OR gate probability of error calculation

A	B	Out	SET	Prob of input
0→0	0→0	0→0	NO	1/16
0→1	0→0	0→1	YES	1/16
0→0	0→1	0→1	YES	1/16
0→1	0→1	0→1	YES	1/16
0→0	1→1	1→1	NO	1/16
0→1	1→1	1→1	NO	1/16
0→0	1→0	1→0	YES	1/16
0→1	1→0	1→0	NO	1/16
1→1	0→0	1→1	NO	1/16
1→1	0→1	1→1	NO	1/16
1→0	0→0	1→0	YES	1/16
1→0	0→1	1→1	NO	1/16
1→1	1→1	1→1	NO	1/16
1→1	1→0	1→1	NO	1/16
1→0	1→1	1→1	NO	1/16
1→0	1→0	1→0	YES	1/16

For 2-input OR will have the same error probability formula as AND and NAND gate, and equations (22) and (23) will still be valid.

5.4 NOR gate probability of error

For a 2- input NOR gate with A,B terminals as input, it is assumed that all the input has equal probability. Table 7 illustrates the calculation of the probability of error of 2-NOR gate. The proposed model considers all the possible transitions of inputs and only take care of the one that will cause the SET to propagate.

Table 7 2-input NOR gate probability of error calculation

A	B	out	SET	Prob of input
0→0	0→0	1→1	NO	1/16
0→1	0→0	1→0	YES	1/16
0→0	0→1	1→0	YES	1/16
0→1	0→1	1→0	YES	1/16
0→0	1→1	0→0	NO	1/16
0→1	1→1	0→0	NO	1/16
0→0	1→0	0→1	YES	1/16
0→1	1→0	0→0	NO	1/16
1→1	0→0	0→0	NO	1/16
1→1	0→1	0→0	NO	1/16
1→0	0→0	0→1	YES	1/16
1→0	0→1	0→0	NO	1/16
1→1	1→1	0→0	NO	1/16
1→1	1→0	0→0	NO	1/16
1→0	1→1	0→0	NO	1/16
1→0	1→0	0→1	YES	1/16

For 2-input NOR will have the same error probability formula as AND, NAND and OR gate, and equations (22) and (23) will still be valid.

5.5 Buffer gate probability of error

A buffer gate only has 1 terminal, so the SET at the input will propagate to the output. So the probability of error of the buffer gate will be the same as the calculated SET.

$$P_{Buffer} = \mathrm{P_{SET}} \qquad (24)$$

5.6 Inverter gate probability of error

An inverter gate only has 1 terminal, so the SET at the input will propagate to the output without logical masking. So the probability of error of the inverter gate will be the same as the calculated SET.

$$P_{INV} = \mathrm{P_{SET}} \qquad (25)$$

5.7 XOR gate probability of error

For a 2- input XOR gate with A,B terminals as input, it is assumed that all the input has equal probability. Table 8 illustrates the calculation of the probability of error of 2-XOR gate. The proposed model considers all the possible input transitions and only takes care of the one that will cause the SET to propagate.

Table 8 2-input XOR gate probability of error calculation

A	B	out	SET	Prob of input
0→0	0→0	0→0	NO	1/16
0→1	0→0	0→1	YES	1/16
0→0	0→1	0→1	YES	1/16
0→1	0→1	0→0	NO	1/16
0→0	1→1	1→1	NO	1/16
0→1	1→1	1→0	YES	1/16
0→0	1→0	1→0	YES	1/16
0→1	1→0	1→1	NO	1/16
1→1	0→0	1→1	NO	1/16
1→1	0→1	1→0	YES	1/16
1→0	0→0	1→0	YES	1/16
1→0	0→1	1→1	NO	1/16
1→1	1→1	0→0	NO	1/16
1→1	1→0	0→1	YES	1/16
1→0	1→1	0→1	YES	1/16
1→0	1→0	0→0	NO	1/16

For 2-input XOR, SET will only propagate if a single-event transient happens when A is inverted from B. So odd number of single events transient must happen for N-input XOR so that the SET will propagate. So, the closed-form expression was introduced N-input XOR gate and the multiple events transients will be as (26) (22).

$$P_{XOR\,N} = 2^{N} * \sum_{k=1}^{M} \frac{\binom{N}{2k-1} {P_{SET}}^{2k-1}}{2^{2N}} \qquad (26)$$

Where M is the number of event-transient wanted to take into consideration, and N is the number of input terminals. M could take any value from 1 to $N-1$. In the current work, Single event transient was only. So $M = 1$ is used, and equation (26) will be reduced to (27)

$$P_{XOR\ N} = \frac{N * P_{SET}}{2^N} \tag{27}$$

5.8 XNOR gate probability of error

For a 2- input XNOR gate with A,B terminals as input, it is assumed that all the input has equal probability. Table 9 illustrates the calculation of the probability of error of 2-XNOR gate. The proposed model considers all the possible transitions of inputs and only take care of the one that will cause the SET to propagate.

Table 9 2-input XNOR gate probability of error calculation

A	B	out	SET	Prob of input
0→0	0→0	1→1	NO	1/16
0→1	0→0	1→0	YES	1/16
0→0	0→1	1→0	YES	1/16
0→1	0→1	1→1	NO	1/16
0→0	1→1	0→0	NO	1/16
0→1	1→1	0→1	YES	1/16
0→0	1→0	0→1	YES	1/16
0→1	1→0	0→0	NO	1/16
1→1	0→0	0→0	NO	1/16
1→1	0→1	0→1	YES	1/16
1→0	0→0	0→1	YES	1/16
1→0	0→1	0→0	NO	1/16
1→1	1→1	1→1	NO	1/16
1→1	1→0	1→0	YES	1/16
1→0	1→1	1→0	YES	1/16
1→0	1→0	1→1	NO	1/16

For 2-input XNOR, SET will only propagate if a single-event transient happens when A is inverted from B same as XOR. So, the closed-form expression was introduced *N*-input XNOR gate, and the multiple events transients will be the same as XOR as (26) and (27).

5.9 Proposed Method Significance

The proposed method is a very simple methodology to get the probability of error based on the normal distribution of the SET voltage and pulse width. This assumption is physical, as the voltage and pulse width depend on the location of the particle hit near the gate. In addition, the energy of SET is not deterministic and can vary. So, the proposed method can help in getting the standard cell probability of error without the need for characterization. In addition, the proposed method accounts for logical masking, which gives a more realistic value for the probability of error for each standard cell. Moreover, a closed formula for the probability of error for different standard cells with N number of pins has been obtained. These formulas are valid for multiple event transient environments.

Chapter 6

New Probabilistic Circuit Testing Algorithm

A novel probabilistic circuit testing algorithm is presented based on the ITM, and PTM explained before. The presented algorithm does not depend on special ITM like the fanout and wire-swapping. Furthermore, it does not require matrix multiplication and tensor product. This method overcomes the limitation of the Krishnaswamy method for big circuits. This algorithm will allow the testing of large circuits with a huge number of inputs and outputs in a feasible time. This requires a new model for the circuit node and a new way to calculate the probability of error at the output of each gate. Then, the propagation of the probability of error from input to output will be done using the proposed algorithm.

6.1 New Node Model

Previously, any digital node in a circuit could take 0 or 1 only. This is a well-known concept that is applied in all electronics EDA tools. In the proposed work, it is suggested that any node can take both values with a probability. So, now any node will have $[p(0)\ p(1)]$, which is called stimulus probability in our thesis. So if the node is ideal and the circuit is error-free, 0 in the old concept is mapped to [1 0] as this means that $p(0) = 1$, and 1 in the old concept is mapped to [0 1] as this means that $p(1) = 1$. This new proposal will give a new degree of freedom for a faulty node as it can easily express the probability of error at a given node in terms of probability.

6.2 Probability of Stimulus at Standard Cell's Outputs

This thesis introduces a new method for determining the probability of a stimulus for each output of a standard cell, which is crucial for the recently introduced node model. To achieve this, the PTM of each standard cell and utilize a conditional probability model is used, as described by equations (28) and (29), to determine the probability of a stimulus for the cell's output.

Here, N represents the number of inputs, and k is in binary. $P[k]$ is the product of the probabilities of each of the cell's inputs since inputs are assumed independent. For example, in a 2-input gate with terminals A and B, $P[input = 11] = P[A = 1] * P[B = 1]$. Moreover, $P[out = 0 \mid input = k\ ]$ can be found in the first column of PTM and in the (k+1) row ($PTM_{k+1,1}$), while and $P[out = 1 \mid input = k]$ can be found in the second column of PTM and in the (k+1) row($PTM_{k+1,2}$).

$$P[out = 0] = \sum_{k=0}^{N-1} P[out = 0 \mid input = k] * P[k] \tag{28}$$

$$P[out = 1] = \sum_{k=0}^{N-1} P[out = 1 \mid input = k] * P[k] \tag{29}$$

For example, for a generic 2-input gate

$$P[out = 0] = \mathrm{P[A = 0, B = 0] * M_{1,1} + P[A = 1, B = 0] * M_{2,1} + P[A = 0, B = 1] * M_{3,1} + P[A = 1, B = 1] * M_{4,1}}$$

$$P[out = 1] = \mathrm{P[A = 0, B = 0] * M_{1,2} + P[A = 1, B = 0] * M_{2,2} + P[A = 0, B = 1] * M_{3,2} + P[A = 1, B = 1] * M_{4,2}}$$

Where M is the PTM of the standard cell

6.3 Propagation of Probability of Error

The proposed algorithm works in the same manner as any gate-simulating tool; the only change is that the node will have stimulus probability $[p(0)\, p(1)]$, and instead of each gate having a truth table, PTM will be used with each input stimulus of probability to get the output stimulus probability. So, the algorithm must work for a certain input vector. In addition, it assumes that the input is ideal (no error). Then, the input vector is converted to the new node model. Afterward, the algorithm calculates the probability of a stimulus for the cells connected to the circuit's input. This process is then repeated for the cells that are connected to the new nodes until the primary outputs are reached. For example, FIGURE 35 represents the gate level netlist of C17 of ISCAS85. Initially, the stimulus probability for O[0] and O[1] in gates 1 and 2 is examined, respectively. Subsequently, the existing stimulus probability of nodes is used to calculate the stimulus probability for O[2] and O[3] in gates 4 and 3, respectively. Finally, the stimulus probability for O[4] and O[5] is determined in gates 5 and 6, which correspond to the primary outputs of C17.

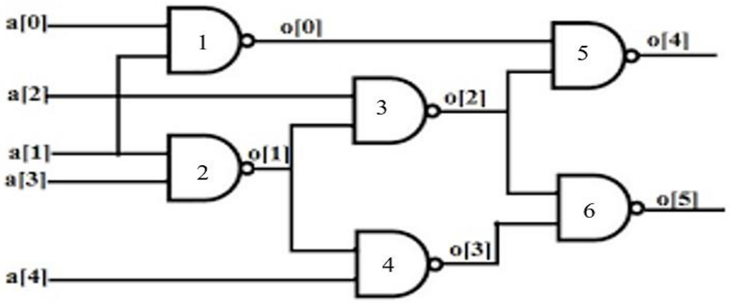


FIGURE 35 Gate Level Netlist of C17 ISCAS85 Example

6.4 Worst-Case Vector Exploration

In order to find the worst-case vector, the algorithm can choose the worst-case pattern according to different metrics:

6.4.1 Reconstructing ITM and PTM.

In this metric, PTM and ITM are reconstructed from their definition. $PTM_{i,k} = P(out = k - 1 \,|input = i - 1)$, so for each input vector, the $P(out = k - 1)$ is calculated by the product of the probability of their stimulus probability if all the nodes are assumed to be independent. For example, if PTM_{13} is desired in the circuit in FIGURE 35, the input will be 00000, and $P(out = 2)$ will be calculated which will be $P(o[4] = 1, o[5] = 0) = P(o[4] = 1) * P(o[5] = 0)$ and so on. In addition, the ITM will be reconstructed in the same manner. Then, circuit fidelity will be calculated, and the worst-case vector will be found in the same way as equation (20) in section 4.2.5. The main problem of this method is for a huge circuit that has a huge number of outputs as it will take time to get all PTM and ITM. For example, for the C7552 circuit that has 108 output, the size of PTM for a single input vector will be $1\text{x}2^{108}$ which will be infeasible. In addition, in real circuits, there will be dependencies, and some nodes will have the same input such as in fanout cases. So, getting the output pattern probability of errors will be harder. So a new metric that does not depend on the reconstructed PTM and ITM and depends only on the node probability of error will be more useful for the proposed algorithm.

6.4.2 Second Norm

The new proposed metric that is used by the proposed flow is the two-norm. The two norm measures the difference between the probability of the primary output's stimulus and the ideal probability by calculating the Euclidean distance between the flawed outputs and the error-free outputs. In order to accomplish this, two matrices named A and B are generated with dimensions of M x 2, where M refers to the number of outputs. A is constructed by stacking the stimulus probability of output of the faulty outputs, whereas B is constructed by stacking the error-free output. Both matrices have two columns only, where the first column denotes the probability of zero, and the second column denotes the probability of one for each output. Subsequently, the two-norm of (A-B) is calculated, and equation (30) is applied to document how the pattern influences the overall error for each test vector. The test vector with the highest second norm is chosen as the worst-case vector.

$$E = \|A - B\| \tag{30}$$

Furthermore, a weight factor can be added if some outputs are more crucial in the circuit, as equation (31) indicates. This can be applied to clock pins or important design pins.

$$E = \|\mathbf{W}.(A - B)\| \tag{31}$$

In addition, the weight factor can be the mean ratio between input and output SET pulse (Δ) that was done in [32] by propagating the SET to primary outputs and then dividing it by input SET for each test vector. In [32], the SET pulse width and voltage at the output were calculated by the propagation of pulse width and voltage from a node under test to observable output.

$$\Delta = \frac{mean \int SET_{out}}{mean \int SET_{in}} \tag{32}$$

The main problem of the proposed method is the exhaustive search by inserting all the possible patterns. In order to face this issue, an automatic test pattern generation (ATPG) tool was used to produce a limited number of test vectors with the best coverage ratio. Specifically, Tessent FastScan is used to generate all the test patterns. From these patterns, a limited number of vectors were selected to obtain the worst-case vector. FastScan considers every node a potential faulty node and creates a series of test vectors to achieve maximum coverage, representing the percentage of nodes tested. The generated test patterns produce a statistical report indicating the total coverage of the entire set. In addition, FastScan creates a separate file summarizing the status of each net in the circuit and whether it is detected or not.

6.5 Algorithm Implementation

The same condition of cell characterization is applied in 4.2 . In addition, the parse used is the same parser that is used in 4.2.3, but the algorithm doesn't need the fanout or wire-swapping detection. So, The parser only parses once to know the circuit hierarchy. In addition, the manipulation step is integrated with the parser. Then, the algorithm calculates the stimulus probability for circuit output for a certain input. FIGURE 36 represents the algorithm flow chart. Firstly, FastScan is used to generate all the input test patterns, then MATLAB will be used to parse these test patterns and circuit hierarchy. Then, the stimulus probability at the observable output is calculated for each test pattern. Then, the two-norm will be used as the error metric. Finally, the pattern associated with the highest value of the second norm will be considered the worst-case vector, as the weight factor in our work is assumed to be all the same.

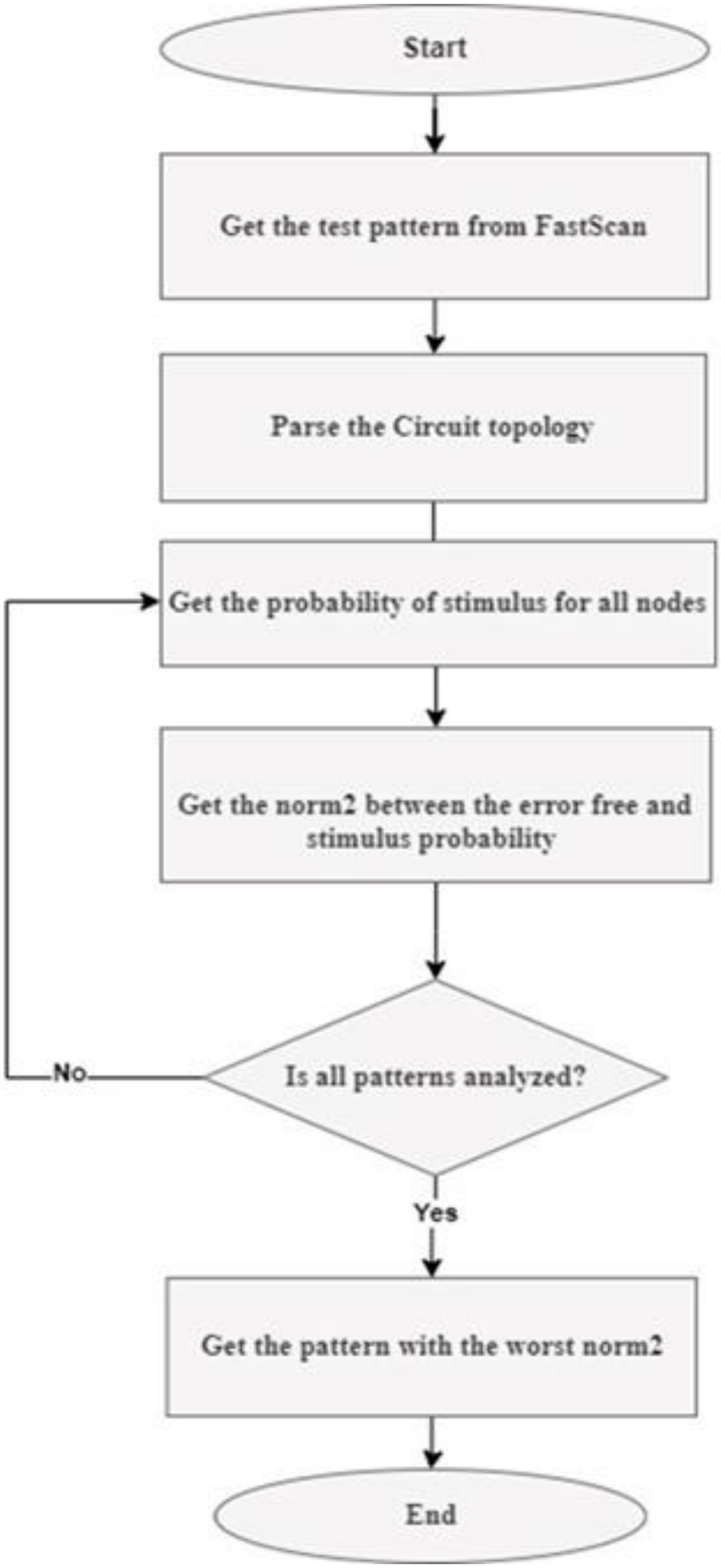

Start
Get the test pattern from FastScan
Parse the Circuit topology
Get the probability of stimulus for all nodes
Get the norm2 between the error free and stimulus probability
Is all patterns analyzed?
No
Yes
Get the pattern with the worst norm2
End

FIGURE 36 Novel algorithm flow chart

6.6 Algorithm Validation

In order to validate the new algorithm, two methods have been done. The first method is to test random input vectors and compare the output of gate-level simulation using MATLAB and SystemVerilog. The test has been done to check the error-free pattern at the output and map [1 0] to 0 and [0 1] to 1. Then, the output pattern was compared to the output of the SystemVerilog. The validation was done on different ISCAS85 circuits, and it was a 100% match. The second method was done for a small circuit like C17 of ISCAS85 by reconstructing the ITM from the new algorithm and comparing it to Krinshnaswamy's ITM. Both ways restarted in a perfect match. Since the algorithm uses the same steps for error-free and faulty gates, the algorithm is validated. For example, for small circuit like C17 in FIGURE 35, Table 10 represents a comparison between the SystemVerilog output and MATLAB output. As stated previously, [1 0] is mapped to 0, and [0 1] is mapped to 1. So that the output of MATLAB is the same as the SystemVerilog. This means the error-free algorithm matches with SystemVerilog, which means the stimulus probability of the node is correct for faulty gates as they use the same methodology.

Table 10 Comparison between MATLAB ideal circuit output VS SystemVerilog for C17

Input Patterns	**SystemVerilog**		**MATLAB**	
	O[4]	O[5]	O[4]	O[5]
0 1 1 1 1	0	0	[1 0]	[1 0]
0 0 1 0 1	0	1	[1 0]	[0 1]
1 0 0 1 1	0	1	[1 0]	[0 1]
1 1 0 0 0	1	1	[0 1]	[0 1]
1 0 1 0 0	1	0	[0 1]	[1 0]

6.7 Results and Discussion

Table 11 illustrates the algorithms' worst-case test vectors time and value of the second norm for the worst-case vector for different ISCAS85 benchmarks using XFAB® XH018 technology. A laptop with a 12th with 16 GB RAM and Gen Intel(R) Core(TM) i7-12700H processor runs the MATLAB code for both algorithms. The small circuits produced the same worst-case pattern using both algorithms. However, Krinshnaswamy's method encountered memory issues for large circuits and could not compute the worst-case test vector, while the novel algorithm successfully produced results quickly. The standard cells' probability of errors was calculated when the SET mean amplitude was 0.9 V, with a standard deviation of 0.1 V, and a mean width of 300 ps, with a standard deviation of 100 ps. Additionally, the critical voltage and pulse width are 0.9 V and 300 ps, respectively.

Table 11 New Algorithm Computational Result Result

#	*Benchmark Circuit*	*No. Test Vectors*	**No of inputs**	**No of outputs**	**No of Cells**	**Proposed Method**	
						Computational time	**Norm2**
1	C17	5	5	2	6	0.0033 sec	0.070765
2	C432	120	36	7	116	0.37 sec	1.789386
3	C7552	282	207	108	990	11 sec	4.467548
4	C499	186	41	32	174	0.79 sec	1.259853
5	C1908	220	33	25	243	1.91 sec	2.285325
6	C880	134	60	26	252	1.24 sec	1.69

The worst case vector using the new methodology for C17 is 5h0F, and for C432 is 36hF5BA91293. In addition, for C7552, the worst-case vector is 207h5EE38D17FD8CF189EB8547793B3EEBFB3BEA65AABA0EB06FE044. For C499, there are four worst-case vectors with the same Norm2 value; the vectors are as follows: 41h 0A25BBACB3C, 41h 132BBBA977C,41h 1FE9F2BBB1E, and 41h132BBBA96BE. For C1908, the worst-case vector is 33h16DA901F6. Finally, For C880, the worst-case vector is 60hADFBC6C00F18A10.

The novel algorithm is faster as it doesn't need matrix multiplication or constructing a huge matrix like Krinshnaswamy's method. For example, for the C7552 circuit, the ITM or PTM matrix size will be 2^{108} x 2^{207} which is infeasible. In addition, the new algorithm does not need to identify the fanout or wire-swapping. For small circuits, the new algorithm is 10x faster than the Krishnaswamy algorithm and produces the same worst-case vector.

It is worth noting that FASTSCAN produces vectors that span the whole circuit faults and nodes with maximum possible coverage, and the produced vectors are not equivalent. In most cases, the first number of worst-case vectors can be equivalent. So that, first worst-case vectors can be equivalent and not appear in FASTSCAN, but the worst-case one will be there. So, for example, for the C17 circuit, FASTSCAN produces only five vectors. However, they are not the top five vectors that have the highest probability of error. This can be explained in terms of equivalent faults discussed earlier in 1.4 , as the top five vectors are equivalent.

Table 12 represents the probability of error for patterns that came from FASTSCAN using Krishnaswamy and the proposed methods. PTM of the C17 circuit was reconstructed as done in 6.4.1. In most cases, the probability of error was the same, but some cases were different when the output of the circuit was the same. This is because the reconstruction of PTM is based on the assumption that the circuit outputs are independent of each other, which is not true for most of the circuits. For example, for C17 in FIGURE 35, o[2] is input for both NAND gates, so outputs are dependent on each other. So the reconstruction of the PTM of the circuit is not valid in most cases. So, it is recommended that the norm2 metric is used for our proposed method as it will overcome this issue.

Table 12 The output pattern probability of error for different input vectors for C17 using Krishnaswamy method and the proposed method

Input Patterns	**Output Patterns**	***Krishnaswamy Method***	***Proposed Method***
0 1 1 1 1	0 0	0.0469	0.0691
0 0 1 0 1	0 1	0.0467	0.0466
1 0 0 1 1	0 1	0.0467	0.0466
1 1 0 0 0	1 1	0.0316	0.0464
1 0 1 0 0	1 0	0.0392	0.0391

If one level of dependency is removed, the results will be more close to each other. For example, if the level of dependency at the output is removed by combining the 2 NAND gates at the output by using PTM algebra like FIGURE 37. The PTM of the red block will be $(\mathrm{I} \otimes \mathrm{F}_2 \otimes \mathrm{I})(\mathrm{NAND}_2 \otimes \mathrm{NAND}_2)$. Appling the same concept of conditional probability, the new output probability of error at the output will be obtained as in Table 13.

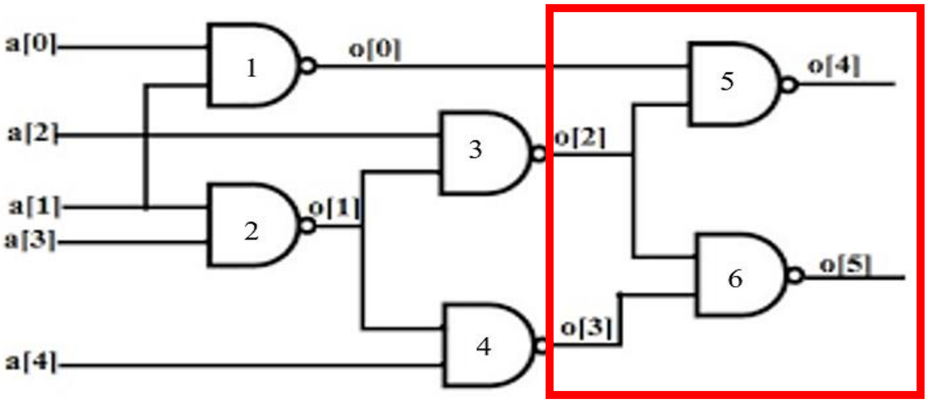


FIGURE 37 Removing dependency at the output for C17

Table 13 The output pattern probability of error for different input vectors for C17 using Krishnaswamy method and the proposed method after removing one level of dependency.

Input Patterns	Output Patterns	*Krishnaswamy Method*	*Proposed Method*
0 1 1 1 1	0 0	0.0469	0.054
0 0 1 0 1	0 1	0.0467	0.0467
1 0 0 1 1	0 1	0.0467	0.0467
1 1 0 0 0	1 1	0.0316	0.0316
1 0 1 0 0	1 0	0.0392	0.0392

The level of dependencies is caused by the fanout that causes some nodes to depend on each other, so when the level of dependencies decreases, the probability of error gets closer to each other. So, norm2 is the more convenient metric for our proposed method, as it gets the probability of error of each node. On the other hand, Krishnaswamy's method gets the probability of each output vector.

6.8 Significance of The Proposed Algorithm

The proposed method provides a simple way to get each node stimulus probability in the same manner as a normal gate simulator tool does. In addition, the new module makes getting the probability of error of each node feasible rather than the output pattern probability of error that is obtained from Krishnaswamy's method. Furthermore, the new model requires a new metric to get the worst-case vector which is based only on the stimulus probability of output nodes. Furthermore, due to the simplicity of the operation, the code is much faster (around 10x faster) and takes much less memory than

Krishnaswamy's method. So, analysis of big gates with many inputs and outputs is feasible. Finally, both method gets similar worst-case vectors as the two methods for small circuits, while for large circuits, which was infeasible using Krishnaswamy, it is now feasible using the proposed method.

Chapter 7

Conclusion and Future Work

This work presents the different types of circuit faults. Then, soft circuits fault, particularly SET, was discussed. The physical causes of SET were discussed. Then different electrical models of SET characterization are discussed. In addition, this work presents a novel method to find the probability of error for each standard cell. Furthermore, this thesis proposed a new algorithm to calculate the error probability at the observable outputs and the worst-case test vector in the ASIC circuit. The new model uses a realistic probability of error of standard cells to account for logic masking. All technology standard cell circuits were characterized and then used to characterize complete ASIC gate-level netlist using our new methodology and Krishnaswamy's method. The worst-case pattern is calculated using both methods. The main difference between the proposed and Krishnasawmy methods is that the proposed method gets the probability of error of every single node in the circuit, while Krishnaswamy gets the probability of error for patterns in the observable outputs. The flow aims to help circuit designers find the worst-case vector without performing real-time simulations. It can also be performed either in the design or signoff phases. The designer must aim to signoff with the smallest possible probability of error to avoid transient error as possible, especially for circuits operated in radiative environments. The proposed algorithm analyzes test vectors from the ATPG tool to obtain the worst—case test vector. The proposed flow helps circuit designers immune their circuits to transient faults and guides them during lab experiments because it is compatible with the typical ASIC development and implementation flow. Furthermore, a worst-case vector is related to the worst SET energy, as the worst-case power that the SET can inject, as SET is most likely to occur, means that the energy has been transmitted from the input to the output.

To complete our work, a generic parser that can parse sequential circuits must be done and that divides paths into combinational paths that the new method can work into.

Furthermore, a test chip must be fabricated to test the SET in the laboratory and verify the worst-case vector.

References

[1] K.-T. Chueng, S. Dey, M. Rodgers, and K. Roy, "Test challenges for deep sub-micron technologies," in *Proceedings of the 37th conference on Design automation - DAC '00*, Los Angeles, California, United States: ACM Press, 2000, pp. 142–149. doi: 10.1145/337292.337353.

[2] C. Zhao, X. Bai, and S. Dey, "Evaluating Transient Error Effects in Digital Nanometer Circuits," *IEEE Trans. Rel.*, vol. 56, no. 3, pp. 381–391, Sep. 2007, doi: 10.1109/TR.2007.903288.

[3] A. K. Stamper, T. L. McDevitt, and S. L. Luce, "Sub-0.25-micron interconnection scaling: damascene copper versus subtractive aluminum," in *IEEE/SEMI 1998 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop (Cat. No.98CH36168)*, Boston, MA, USA: IEEE, 1998, pp. 337–346. doi: 10.1109/ASMC.1998.731585.

[4] B. Straka, H. Manhaeve, J. Vanneuville, and M. Svajda, "A fully digital controlled off-chip I/sub DDQ/ measurement unit," in *Proceedings Design, Automation and Test in Europe*, Paris, France: IEEE Comput. Soc, 1998, pp. 495–500. doi: 10.1109/DATE.1998.655904.

[5] J. F. Ziegler, "Terrestrial cosmic rays," *IBM J. Res. & Dev.*, vol. 40, no. 1, pp. 19–39, Jan. 1996, doi: 10.1147/rd.401.0019.

[6] D. Timothy J., "A white paper on the benefits of chipkill-correct ecc for pc server main memory," *Dell1997AWP*, 1997.

[7] D. Munteanu and J. L. Autran, "Terrestrial neutron-induced single events in GaN," *Microelectronics Reliability*, vol. 100–101, p. 113357, Sep. 2019, doi: 10.1016/j.microrel.2019.06.049.

[8] X. Chen, Q. Guo, H. Yuan, and Y. Guo, "A Single-Event Transient Radiation Hardened Low-Dropout Regulator for LC Voltage-Controlled Oscillator," *Symmetry*, vol. 14, no. 4, p. 788, Apr. 2022, doi: 10.3390/sym14040788.

[9] K. Feng, S. Vora, R. Jiang, E. Rosenbaum, and S. Vasudevan, "Guilty As Charged: Computational Reliability Threats Posed By Electrostatic Discharge-induced Soft Errors," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy: IEEE, Mar. 2019, pp. 156–161. doi: 10.23919/DATE.2019.8715149.

[10] V. Ferlet-Cavrois, L. W. Massengill, and P. Gouker, "Single Event Transients in Digital CMOS—A Review," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 1767–1790, Jun. 2013, doi: 10.1109/TNS.2013.2255624.

[11] A. A. Abou-Auf, "Total-Dose Worst-Case Test Vectors for Leakage Current Failure Induced in Sequential Circuits of Cell-Based ASICs," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 4, pp. 2189–2197, Aug. 2009, doi: 10.1109/TNS.2009.2019275.

[12] "Fault Modeling," in *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, in Frontiers in Electronic Testing, vol. 17. Boston: Kluwer Academic Publishers, 2002, pp. 57–80. doi: 10.1007/0-306-47040-3_4.

[13] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electron. Comput.*, vol. EC-16, no. 5, pp. 567–580, Oct. 1967, doi: 10.1109/PGEC.1967.264743.

[14] shankardas deepti Bharat, "ATPG methods and algorithms," 2012. [Online]. Available: https://www.slideshare.net/deeptishankardas/aptg-methods-and-algorithms

[15] P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "An automatic test pattern generator for large sequential circuits based on Genetic Algorithms," in *Proceedings., International Test Conference*, Washington, DC, USA: Int. Test Conference, 1994, pp. 240–249. doi: 10.1109/TEST.1994.527955.

[16] R. C. Baumann, "Landmarks in Terrestrial Single-Event Effects." NSREC Short Course,

2013. [Online]. Available: http://www-vlsi.es.kit.ac.jp/SERconf/2014/Baumanall.pdf

[17] A. Abubakr, A. Ibrahim, Y. Ismail, and H. Mostafa, “The Impact of Soft Errors on Memristor-Based Memory,” in *2017 New Generation of CAS (NGCAS)*, Genova, Italy: IEEE, Sep. 2017, pp. 229–232. doi: 10.1109/NGCAS.2017.72.

[18] N. Michael, “Radiation Environments.” [Online]. Available: https://www.iroctech.com/library/working-environments/

[19] D. F. Heidel *et al.*, “Alpha-particle-induced upsets in advanced CMOS circuits and technology,” *IBM J. Res. & Dev.*, vol. 52, no. 3, pp. 225–232, May 2008, doi: 10.1147/rd.523.0225.

[20] M. Gordon and K. Rodbel, “Single-Event Upsets and Microelectronics.” 2015.

[21] M. Andjelkovic, A. Ilic, Z. Stamenkovic, M. Krstic, and R. Kraemer, “An overview of the modeling and simulation of the single event transients at the circuit level,” in *2017 IEEE 30th International Conference on Microelectronics (MIEL)*, Nis: IEEE, Oct. 2017, pp. 35–44. doi: 10.1109/MIEL.2017.8190065.

[22] Y. V. Katunin and V. Ya. Stenin, “TCAD simulation of single-event transients in the 65-nm CMOS element of matching for a content-addressable memory,” in *2017 25th Telecommunication Forum (TELFOR)*, Belgrade: IEEE, Nov. 2017, pp. 1–4. doi: 10.1109/TELFOR.2017.8249392.

[23] P. E. Dodd, M. R. Shaneyfelt, J. A. Felix, and J. R. Schwank, “Production and propagation of single-event transients in high-speed digital logic ICs,” *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 3278–3284, Dec. 2004, doi: 10.1109/TNS.2004.839172.

[24] G. C. Messenger, “Collection of Charge on Junction Nodes from Ion Tracks,” *IEEE Trans. Nucl. Sci.*, vol. 29, no. 6, pp. 2024–2031, Dec. 1982, doi: 10.1109/TNS.1982.4336490.

[25] S. Barcelo, X. Gili, S. A. Bota, and J. Segura, “An SET propagation EDA tool based on analytical glitch propagation model,” in *2013 14th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Oxford, United Kingdom: IEEE, Sep. 2013, pp. 1–5. doi: 10.1109/RADECS.2013.6937388.

[26] X. Gili, S. Barcelo, S. A. Bota, and J. Segura, “Analytical modeling of Single Event Transients propagation in combinational logic gates,” in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*, Seville, Spain: IEEE, Sep. 2011, pp. 408–411. doi: 10.1109/RADECS.2011.6131416.

[27] M. Ahmed, “Fault Modeling and Test Vector Generation for ASIC Devices Exposed to Space Single Event Environment,” The American Univercity in Cairo, Cairo, 2021. [Online]. Available: https://fount.aucegypt.edu/etds/1645/

[28] C. Lazzari, G. Wirth, F. L. Kastensmidt, L. Anghel, and R. A. da L. Reis, “Asymmetric transistor sizing targeting radiation-hardened circuits,” *Electr Eng*, vol. 94, no. 1, pp. 11–18, Mar. 2012, doi: 10.1007/s00202-011-0212-8.

[29] S. Krishnaswamy, I. L. Markov, and J. P. Hayes, “Logic Circuits Testing for Transient Faults,” in *European Test Symposium (ETS’05)*, Tallinn, Estonia: IEEE, 2005, pp. 102–107. doi: 10.1109/ETS.2005.27.

[30] K. N. Patel, I. L. Markov, and J. P. Hayes, “Evaluating Circuit Reliability Under Probabilistic Gate-Level Fault Models,” *Proceedings of the International Workshop on Logic Synthesis (IWLS)*, May 2003.

[31] S. Krishnaswamy, I. L. Markov, and J. P. Hayes, *Design, Analysis and Test of Logic Circuits Under Uncertainty*, vol. 115. in Lecture Notes in Electrical Engineering, vol. 115. Dordrecht: Springer Netherlands, 2013. doi: 10.1007/978-90-481-9644-9.

[32] M. Wael, A. Ibrahim, M. Abdelaziz, A. Elsafty, and A. A. Abou-Auf, “Single Event

Transient Sensitivity Analysis and Worst-Case Test Vector Identification for ASICs," in *2022 22th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Venice, Italy: IEEE, Oct. 2022, pp. 1–4.

Appendix 1

This appendix describes and lists the different scripts used throughout this work.

1) FastSCAN Code.

This code is used to find the input vectors that uncover all the circuit faults with maximum coverage. All the patterns were saved in all.patterns, which will be used later.

```
system date
set system mode atpg
set fault type transition
add fault -all
create patterns -auto

report faults -all > all.fault_list
report statistics > all.stats
save patterns all.patterns - Ascii -replace

exit
```

2) MATLAB Code.

MATLAB was used to perform the modeling and find the probability of error at the circuit output, then find the worst-case vector.

I. Top-Level

Use_pattern is used to control the usage of our new method if it is 1, 0 will use the Krishmaswa,y method. 2 is used for both methods.

```
%%%LIB has 1 column is cell name, 2rd is P of fault, 3th is ITM
%%%4th is inputs, 5th are outputs
global Lib CKT;
v_in_mean=0.9;v_in_std=0.1;
tw_in_mean=300;tw_in_std=100;
    V_critical=0.9;
    Tw_critical=300;
    %Number_of_points_tw = 50;Number_of_points_v=10;
 models_dir='./models';
%%
%%%CKT has 1 column is cell name/node name,2nd is stage, 3th is ITM/PTM,4th
%%%is position, 5th is either 0 which is gate,1 which is input, 2 which is
%%%output,-1 is dummay gate , 6 is the number of repetion, 7 cell name, 8th
%%%prob of error
 use_pattern=1;

inputs=["N1" "N2" "N3" "N6" "N7"];
outputs = ["N22" "N23"];
CKT_name = "C:\Masters\Thesis\c17\syn_output\c17_syn.v";
```

```
pattern_file = "C:\Masters\Thesis\c17\all.patterns";

% CKT_name = "C:\Masters\Thesis\c432\syn_output\c432_syn.v";
%  outputs =  ["N223" "N329" "N370" "N421" "N430" "N431" "N432"];
%  inputs= ["N1" "N4" "N8" "N11" "N14" "N17" "N21" "N24" "N27" "N30" "N34" "N37"
"N40" "N43" "N47" "N50" "N53" "N56" "N60" "N63" "N66" "N69" "N73" "N76" "N79" "N82"
"N86" "N89" "N92" "N95" "N99" "N102" "N105" "N108" "N112" "N115"];
% pattern_file = "C:\Masters\Thesis\c432\all.patterns";

% CKT_name = "C:\Masters\Thesis\c7552\syn_output\c7552_syn.v";
% outputs =  ["N387" "N388" "N478" "N482" "N484" "N486" "N489" "N492" "N501" "N505"
"N507" "N509" "N511" "N513" "N515" "N517" "N519" "N535" "N537" "N539" "N541" "N543"
"N545" "N547" "N549" "N551" "N553" "N556" "N559" "N561" "N563" "N565" "N567" "N569"
"N571" "N573" "N582" "N643" "N707" "N813" "N881" "N882" "N883" "N884" "N885" "N889"
"N945" "N1110" "N1111" "N1112" "N1113" "N1114" "N1489" "N1490" "N1781" "N10025"
"N10101" "N10102" "N10103" "N10104" "N10109" "N10110" "N10111" "N10112" "N10350"
"N10351" "N10352" "N10353" "N10574" "N10575" "N10576" "N10628" "N10632" "N10641"
"N10704" "N10706" "N10711" "N10712" "N10713" "N10714" "N10715" "N10716" "N10717"
"N10718" "N10729" "N10759" "N10760" "N10761" "N10762" "N10763" "N10827" "N10837"
"N10838" "N10839" "N10840" "N10868" "N10869" "N10870" "N10871" "N10905" "N10906"
"N10907" "N10908" "N11333" "N11334" "N11340" "N11342" "N241_O"];
% inputs= ["N1" "N5" "N9" "N12" "N15" "N18" "N23" "N26" "N29" "N32" "N35" "N38"
"N41" "N44" "N47" "N50" "N53" "N54" "N55" "N56" "N57" "N58" "N59" "N60" "N61" "N62"
"N63" "N64" "N65" "N66" "N69" "N70" "N73" "N74" "N75" "N76" "N77" "N78" "N79" "N80"
"N81" "N82" "N83" "N84" "N85" "N86" "N87" "N88" "N89" "N94" "N97" "N100" "N103"
"N106" "N109" "N110" "N111" "N112" "N113" "N114" "N115" "N118" "N121" "N124" "N127"
"N130" "N133" "N134" "N135" "N138" "N141" "N144" "N147" "N150" "N151" "N152" "N153"
"N154" "N155" "N156" "N157" "N158" "N159" "N160" "N161" "N162" "N163" "N164" "N165"
"N166" "N167" "N168" "N169" "N170" "N171" "N172" "N173" "N174" "N175" "N176" "N177"
"N178" "N179" "N180" "N181" "N182" "N183" "N184" "N185" "N186" "N187" "N188" "N189"
"N190" "N191" "N192" "N193" "N194" "N195" "N196" "N197" "N198" "N199" "N200" "N201"
"N202" "N203" "N204" "N205" "N206" "N207" "N208" "N209" "N210" "N211" "N212" "N213"
"N214" "N215" "N216" "N217" "N218" "N219" "N220" "N221" "N222" "N223" "N224" "N225"
"N226" "N227" "N228" "N229" "N230" "N231" "N232" "N233" "N234" "N235" "N236" "N237"
"N238" "N239" "N240" "N242" "N245" "N248" "N251" "N254" "N257" "N260" "N263" "N267"
"N271" "N274" "N277" "N280" "N283" "N286" "N289" "N293" "N296" "N299" "N303" "N307"
"N310" "N313" "N316" "N319" "N322" "N325" "N328" "N331" "N334" "N337" "N340" "N343"
"N346" "N349" "N352" "N355" "N358" "N361" "N364" "N367" "N382" "N241_I" ];
% pattern_file = "C:\Masters\Thesis\c7552\all.patterns";

%% core
if (use_pattern==1)

Parse_CKT_pattern(CKT_name,models_dir,outputs,v_in_mean,v_in_std,V_critical,tw_in_m
ean,tw_in_std,Tw_critical);
    Cal_Worst_case_pattern2(pattern_file,inputs,outputs)
elseif(use_pattern==0)
    [all_gates_name,all_gates_name_no_in,all_gates_name_no_out]=
Parse_CKT5(CKT_name,models_dir,outputs,v_in_mean,v_in_std,V_critical,tw_in_mean,tw_
in_std,Tw_critical);

maniplate_CKT3(all_gates_name,all_gates_name_no_in,all_gates_name_no_out,outputs);
    Cal_Worst_case(inputs,outputs);
else
```

```
    %% debug

Parse_CKT_pattern(CKT_name,models_dir,outputs,v_in_mean,v_in_std,V_critical,tw_in_mean,tw_in_std,Tw_critical);
    Cal_Worst_case_pattern2(pattern_file,inputs,outputs)
    [all_gates_name,all_gates_name_no_in,all_gates_name_no_out]=
Parse_CKT5(CKT_name,models_dir,outputs,v_in_mean,v_in_std,V_critical,tw_in_mean,tw_in_std,Tw_critical);

maniplate_CKT3(all_gates_name,all_gates_name_no_in,all_gates_name_no_out,outputs);
    Cal_Worst_case(inputs,outputs);
end
```

II. Parse CKT for Krinshnaswamy's method

For Krinshnawamy's method, the code parse the gate level 3 times

```
function  [all_gates_name,all_gates_name_no_in,all_gates_name_no_out]=
Parse_CKT5(CKT_name,models_dir,outputs,v_in_mean,v_in_std,V_critical,tw_in_mean,tw_in_std,Tw_critical)
global Lib CKT;
Lib=cell(0,0);
MyFolderInfo = dir([models_dir,'/*_model.mat']);
for jj=1:size(MyFolderInfo,1)
load([models_dir,'/',MyFolderInfo(jj).name]);
cell_name = string(erase(MyFolderInfo(jj).name,'_model.mat'));
Lib{jj,1}=cell_name;
end
first_parse=0;gates_position_in=[];gates_aranged=strings(0,0);
for parse_counter=1:3
if (parse_counter==1)
    first_parse=1;
else
    first_parse=0;
end
CKT =cell(1,4);

%%%

fid = fopen(CKT_name);
my_line =  fgetl(fid);
intial_stage=0;
%for i=1:length(outputs)
 %  CKT{1}(i)=outputs(i);
  % CKT{2}(i)= intial_stage;
   %CKT{3}(i)={eye(2)};
   %CKT{4}(i)= i; %% 1 is the bottom, 2 is higher
%end
%counter=length(outputs)+1;

%%StD cells names
STD_names=strings(1,size(Lib,1));
for i=1:size(Lib,1)
```

```
    STD_names(1,i)= Lib{i,1};
end
counter=1;
repetion=1;
dummy_counter=0;
all_gates_name=[];all_gates_inputs={};all_gates_outputs={};gates_total_number=1;
all_gates_name_no_in=[];
all_gates_name_no_out=[];
while ischar(my_line)
    % Search for the input of the gate
    my_line = regexprep(my_line,"\(","  ");
    my_line = regexprep(my_line,"\)","  ");
    my_line = regexprep(my_line,",","");
    Line_spiltted= split(my_line);

    if(length(Line_spiltted)>2)
        index_in_lib= find(STD_names==Line_spiltted{2});
    if (index_in_lib)
    %match_IN = regexp(my_line,'//  /(\w*)\s+(\w*)','tokens');
        if (size(Lib(index_in_lib,:),2)==1)||(isempty((Lib{index_in_lib,2})))

charcell(Lib{index_in_lib,1},v_in_mean,v_in_std,V_critical,tw_in_mean,tw_in_std,Tw_cr
itical,index_in_lib);
        end
    %checking gate is connected to output
    Outputs_gate= Line_spiltted(rem(find(Line_spiltted ==
Lib{index_in_lib,4}),length(Line_spiltted))+1);
    Inputs_gate =Line_spiltted(rem(find(Line_spiltted ==
Lib{index_in_lib,5}),length(Line_spiltted))+1);
    index_output= find (outputs == (Line_spiltted{(find(Line_spiltted ==
Lib{index_in_lib,4}))+1}));
    %% saving inputs and outputs of gate %%19/11
    dummy_input=[];dummy_output=[];
    for i=1:length(Inputs_gate)
        dummy_input=[dummy_input " " Inputs_gate{i}];
    end
    for i=1:length(Outputs_gate)
        dummy_output=[dummy_output " " Outputs_gate{i}];
    end
    all_gates_inputs{gates_total_number}=dummy_input;
        all_gates_outputs{gates_total_number}=dummy_output;

        gates_total_number=gates_total_number+1;

    %%
    All_nodes=strings(1,size(CKT,1));
    if(size(CKT,1) >1)
            for i=1:size(CKT,1)
                All_nodes(1,i)= CKT{i,1};
            end
    end
    out_index= find(Outputs_gate==All_nodes);
    [~,in_index_all]
=ind2sub(size(Inputs_gate==All_nodes),find(Inputs_gate==All_nodes));
```

```
    %%for maniplate_CKT.m
    all_gates_name=[all_gates_name " " string(Line_spiltted{3})] ;
    all_gates_name_no_in=[all_gates_name_no_in length(Inputs_gate)];
    all_gates_name_no_out=[all_gates_name_no_out length(Outputs_gate) ];
    %%
%my_line
    % saving gates
    if (counter == 1) ||
((~isempty(Outputs_gate==outputs))&&isempty(out_index)&&isempty(find(Inputs_gate==All
_nodes)))
         CKT{counter,1}=string(Line_spiltted{3});
         CKT{counter,2}=intial_stage-1;
         CKT{counter,3}= Lib{index_in_lib,3};
         CKT{counter,8}= Lib{index_in_lib,2};
         CKT{counter,5}= 0;
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           Position=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
           Position=1;
         end
         CKT{counter,4}= Position; %% position is high
         counter=counter+1;
         Position_in=Position;
         Position_out=Position;
         for i=1:length(Inputs_gate)
         CKT{counter,1}=Inputs_gate(i);
         CKT{counter,2}=intial_stage-2;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_in; %% position is high
         CKT{counter,5}= 1;
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
         Position_in=Position_in+1; %% each input is in different position
         end
         for i=1:length(Outputs_gate)
         CKT{counter,1}=Outputs_gate(i);
         CKT{counter,2}=intial_stage;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_out; %% position is high
         CKT{counter,5}= 2; %% output
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
         Position_out=Position_out+1;
         end
    elseif (out_index)
        stage_temp= CKT{out_index,2};
        %%19/11
%         cell=CKT{out_index,7}(1);
%         cell_input=find(all_gates_name==cell)/2;
%         if ~isempty(cell_input)
%         other_inputs=
all_gates_inputs{cell_input}(find((all_gates_inputs{cell_input}~=Outputs_gate)&
all_gates_inputs{cell_input}~= " "));
%         gate_connected_other_inputs=[];
%         other_inputs_index= find(other_inputs==All_nodes);
%         for j=1:length(other_inputs)
%             index_dummy = find(CKT{other_inputs_index(j),7} == cell)
%             if (CKT{other_inputs_index(j),4}(index_dummy)
<CKT{out_index,4}(index_dummy))
%                 for i=1:length(all_gates_name)
%                     if ~isempty(all_gates_outputs{i}==other_inputs)
%                        Position_temp=Position_temp+all_gates_name_no_in(i);
%                     end
%                 end
%             end
%         end
%         end
        %%
        CKT{out_index,5} = [CKT{out_index,5} 2]; %it is now output
         CKT{counter,1}=string(Line_spiltted{3});
         CKT{counter,2}=stage_temp-1;
         CKT{counter,3}= Lib{index_in_lib,3};
         CKT{counter,8}= Lib{index_in_lib,2};
         CKT{counter,5}= 0;
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           Position_temp=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
            Position_temp= min( CKT{out_index,4});% min of position
         end
         Position_in=Position_temp;
         CKT{counter,4}= Position_temp; %% position is high
                 counter=counter+1;

%% add dummy cell to other inputs if not connected to cell

cell_name =CKT{out_index,7};

    inout=zeros(1,size(CKT,1));

    for i =1:size(CKT,1)
        if ((~isempty(find(CKT{i,5}==1)))&(isempty(find(CKT{i,5}==2))))
                inout(i)=1;
        end
    end
%index_dummy_cell=find(nodes_cell&(inout==1));
if (first_parse)
    index_dummy_cell=[];
else
index_dummy_cell=find((inout==1));
end

for i=1:length(index_dummy_cell)
```

```
    %%insert dummy cell
    if (length(CKT{index_dummy_cell(i),4})>1)
        dummy=CKT{index_dummy_cell(i),7}(1);
    else
       dummy= CKT{index_dummy_cell(i),7};
    end
        CKT{counter,1}= dummy+ "_dummy"+dummy_counter;
         CKT{counter,2}=CKT{index_dummy_cell(i),2}-1;
         CKT{counter,3}= eye(2);
         CKT{counter,5}= -1;
         CKT{counter,6}= 1;
         CKT{counter,7}= dummy + "_dummy"+dummy_counter ;
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           CKT{counter,4}=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
            CKT{counter,4}= CKT{index_dummy_cell(i),4}(1); %% position is high
         end
         counter=counter+1;
         %%update inputs
        CKT{counter,1}= CKT{index_dummy_cell(i),1};
         CKT{counter,2}=CKT{index_dummy_cell(i),2}-2;
         CKT{counter,3}= eye(2);

         CKT{counter,5}= CKT{index_dummy_cell(i),5};
         CKT{counter,6}= 1;
         CKT{counter,7}= dummy + "_dummy"+dummy_counter ;
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           CKT{counter,4}=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
            CKT{counter,4}= CKT{index_dummy_cell(i),4}(1); %% position is high
         end
         counter=counter+1;

        CKT{index_dummy_cell(i),1}=CKT{index_dummy_cell(i),1}+ "_dummy"+dummy_counter
;
                 CKT{index_dummy_cell(i),2}=CKT{index_dummy_cell(i),2};
         CKT{index_dummy_cell(i),3}=CKT{index_dummy_cell(i),3};
         CKT{index_dummy_cell(i),6}=CKT{index_dummy_cell(i),6};
         CKT{index_dummy_cell(i),5}=[1 2];
         dummy_counter=dummy_counter+1;
end
%% update inputs
    All_nodes=strings(1,size(CKT,1));
    if(size(CKT,1) >1)
            for i=1:size(CKT,1)
                All_nodes(1,i)= CKT{i,1};

            end
    end
             for i=1:length(Inputs_gate)
             index_in= find((Inputs_gate(i)==All_nodes));
             if (isempty(index_in))
         CKT{counter,1}=Inputs_gate(i);
         CKT{counter,2}=stage_temp-2;
```

```
     CKT{counter,3}= eye(2);
     CKT{counter,4}= Position_in; %% position is high
     CKT{counter,5}= 1;
     CKT{counter,6}=repetion;
     CKT{counter,7}=string(Line_spiltted{3});
     counter=counter+1;
      Position_in=Position_in+1;
       else %%% update reption and stage
        CKT{index_in,6} =CKT{index_in,6}+1;
         if (CKT{index_in,2}>(stage_temp-2))
           CKT{index_in,2}=(stage_temp-2);
         end
         temp=zeros(2,2*CKT{index_in,6});temp(1,1)=1;temp(end,end)=1;
         CKT{index_in,3}=temp;
         CKT{index_in,4} = [CKT{index_in,4} Position_in ];
         CKT{index_in,7} = [CKT{index_in,7} " " string(Line_spiltted{3}) ];
         Position_in =Position_in+1;
       end
     end
  elseif (~isempty(in_index_all))
     %in_index_all=[];
     pos_temp=zeros(size(in_index_all));
for j=1:length(in_index_all)
  pos_temp(j) = CKT{in_index_all(j),2};
end
[dummy_temp in_index_index]=min(pos_temp);
in_index=in_index_all(in_index_index);
    stage_temp= CKT{in_index,2};
        %% 19/11

%Position_temp=all_gates_name_no_in(find(all_gates_name==CKT{in_index,7})/2)+max(CKT{in_index,4});
    %%
    CKT{in_index,5} = [CKT{in_index,5} 1]; %it is now input
     CKT{counter,1}=string(Line_spiltted{3});
     CKT{counter,2}=stage_temp+1;
     CKT{counter,3}= Lib{index_in_lib,3};
    CKT{counter,8}= Lib{index_in_lib,2};
     CKT{counter,5}= 0;
     CKT{counter,6}= repetion;
     CKT{counter,7}=string(Line_spiltted{3});

     if(~isempty(find(gates_aranged==CKT{counter,7})))
       Position_temp=  gates_position_in(find(gates_aranged==CKT{counter,7}));
     else
     Position_temp= CKT{in_index,4}+1;
     end
      Position_in=Position_temp;
      CKT{counter,4}= Position_temp; %% position is high
     Position_out=Position_temp;
         counter=counter+1;

%%add dummy cell to other inputs if not connected to cell
cell_name =CKT{in_index,7};
```

```
    inout=zeros(1,size(CKT,1));

    for i =1:size(CKT,1)
        if ((~isempty(find(CKT{i,5}==2)))&(isempty(find(CKT{i,5}==1))))
                inout(i)=1;
        end
    end
if (first_parse)
    index_dummy_cell=[];
else
    index_dummy_cell=[];
%index_dummy_cell=find((inout==1));
end

for i=1:length(index_dummy_cell)
        CKT{counter,1}= cell_name+ "_dummy";
         CKT{counter,2}=CKT{index_dummy_cell(i),2}+1;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= CKT{index_dummy_cell(i),4}; %% position is high
         CKT{counter,5}= -1;
         CKT{counter,6}= 1;
         CKT{counter,7}= CKT{index_dummy_cell(i),7} + "_dummy" ;
         counter=counter+1;
        CKT{counter,1}= CKT{index_dummy_cell(i),1};
         CKT{counter,2}=CKT{index_dummy_cell(i),2}+2;
         CKT{counter,3}= CKT{index_dummy_cell(i),3};
         CKT{counter,4}= CKT{index_dummy_cell(i),4}; %% position is high
         CKT{counter,5}= CKT{index_dummy_cell(i),5};
         CKT{counter,6}= CKT{index_dummy_cell(i),6};
         CKT{counter,7}= CKT{index_dummy_cell(i),7} + "_dummy" ;
         counter=counter+1;

        CKT{index_dummy_cell(i),1}=CKT{index_dummy_cell(i),1}+ "_dummy" ;
                 CKT{index_dummy_cell(i),2}=CKT{index_dummy_cell(i),2};
         CKT{index_dummy_cell(i),3}=eye(2);
         CKT{index_dummy_cell(i),6}=1;
end
%%
         for i=1:length(Inputs_gate)
             index_in_other= find((Inputs_gate(i)==All_nodes));
             if (isempty(index_in_other))
         CKT{counter,1}=Inputs_gate(i);
         CKT{counter,2}=stage_temp;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_in; %% position is high
         CKT{counter,5}= 1;
         CKT{counter,6}=repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
            Position_in=Position_in+1;
             else %%% update reption and stage
                CKT{index_in_other,6} =CKT{index_in_other,6}+1;
                 if (CKT{index_in_other,2}>(stage_temp))
```

```
                     CKT{index_in_other,2}=(stage_temp);
                 end
                 temp=zeros(2,2*CKT{index_in_other,6});temp(1,1)=1;temp(end,end)=1;
                 CKT{index_in_other,3}=temp;
                 CKT{index_in_other,4} = [CKT{index_in_other,4} Position_in ];
                 CKT{index_in_other,7} = [CKT{index_in_other,7} " "
string(Line_spiltted{3}) ];
                 Position_in =Position_in+1;
             end
         end
        for i=1:length(Outputs_gate)
             index_out= find((Outputs_gate(i)==All_nodes));
             if (isempty(index_out))
         CKT{counter,1}=Outputs_gate(i);
         CKT{counter,2}=stage_temp+2;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_out; %% position is high
         CKT{counter,5}= 2;
         CKT{counter,6}=repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
            Position_out=Position_out+1;
             else %%% update reption and stage
                CKT{index_out,6} =CKT{index_out,6}+1;
                 if (CKT{index_out,2}>(stage_temp))
                     CKT{index_out,2}=(stage_temp);
                 end
                 temp=zeros(2,2*CKT{index_out,6});temp(1,1)=1;temp(end,end)=1;
                 CKT{index_out,3}=temp;
                 CKT{index_out,4} = [CKT{index_out,4} Position_in ];
                 Position_out =Position_out+1;
             end
         end


    else
     fprintf("Case is not considered");
    end


     %%%%%%%%%% to be continued dealing with nodes
    end
    end
    my_line = fgetl(fid);
end
%% maniplate CKT

j=1;nodes_reptead=[];nodes_reptead_index=[];nodes_reptead_gates={};nodes_reptead_posi
tion={};
stages =[]; gates_aranged=strings(0,0);gates_no_in=[];gates_no_out=[];
for i=1:length(CKT)
    overwrite=0;
    if(length(CKT{i,4})>1)
        nodes_reptead= [nodes_reptead CKT{i,1}];
        nodes_reptead_position(j)={CKT{i,4}};
        nodes_reptead_index=[nodes_reptead_index i];
        nodes_reptead_gates(j)= {CKT{i,7}};
        j=j+1;
    end
    %% arrange the cells in CKT stages
    stage_index=find(stages==CKT{i,2});
    if (isempty(stage_index)&(CKT{i,5}<1))
        stages=[stages CKT{i,2}];
        stage_index=length(stages);
    end
    if (CKT{i,5}<1)

        if (~isempty((find(all_gates_name == CKT{i,7}))))
            if (CKT{i,4}<=size(gates_no_in,1))&(stage_index <=
size(gates_no_in,2))&(gates_no_in(CKT{i,4},stage_index)~=0)
                overwrite=1;
            old_row_in=zeros(1,size(gates_no_in,2));old_row_out=old_row_in;

old_row_in(stage_index)=gates_no_in(CKT{i,4},stage_index);old_row_out(stage_index)=ga
tes_no_out(CKT{i,4},stage_index);

gates_no_in(CKT{i,4},stage_index)=all_gates_name_no_in((find(all_gates_name ==
CKT{i,7}))/2);
             gates_no_out(CKT{i,4},stage_index)=
all_gates_name_no_out((find(all_gates_name == CKT{i,7}))/2);

gates_no_in=[gates_no_in(1:CKT{i,4},:);old_row_in;gates_no_in(CKT{i,4}+1:end,:)];

gates_no_out=[gates_no_out(1:CKT{i,4},:);old_row_out;gates_no_out(CKT{i,4}+1:end,:)];
            else

gates_no_in(CKT{i,4},stage_index)=all_gates_name_no_in((find(all_gates_name ==
CKT{i,7}))/2);
             gates_no_out(CKT{i,4},stage_index)=
all_gates_name_no_out((find(all_gates_name == CKT{i,7}))/2);
            end
        else
            %dummy_gate
            if (CKT{i,4}<=size(gates_no_in,1))&(stage_index <=
size(gates_no_in,2))&(gates_no_in(CKT{i,4},stage_index)~=0)
                overwrite=2;
            old_row_in=zeros(1,size(gates_no_in,2));old_row_out=old_row_in;

old_row_in(stage_index)=gates_no_in(CKT{i,4},stage_index);old_row_out(stage_index)=ga
tes_no_out(CKT{i,4},stage_index);
            gates_no_in(CKT{i,4},stage_index)=1;
            gates_no_out(CKT{i,4},stage_index)=1;

gates_no_in=[gates_no_in(1:CKT{i,4},:);old_row_in;gates_no_in(CKT{i,4}+1:end,:)];

gates_no_out=[gates_no_out(1:CKT{i,4},:);old_row_out;gates_no_out(CKT{i,4}+1:end,:)];
            else
            gates_no_in(CKT{i,4},stage_index)=1;
            gates_no_out(CKT{i,4},stage_index)=1;
            end
        end
        if (overwrite)
            old_row=strings(1,size(gates_no_in,2));
            old_row(stage_index)=gates_aranged(CKT{i,4},stage_index);
            gates_aranged(CKT{i,4},stage_index)=CKT{i,7};

gates_aranged=[gates_aranged(1:CKT{i,4},:);old_row;gates_aranged(CKT{i,4}+1:end,:)];
        else
            gates_aranged(CKT{i,4},stage_index)=CKT{i,7};
        end
    end
end
gates_position_in=zeros(size(gates_aranged));
gates_position_in(1,:)=1;
for i=2:size(gates_aranged,1)

        gates_position_in(i,:)=sum(gates_no_in(1:i-1,:),1)+1;

end
end
end
```

III. Parse CKT for new method method

In this parser, the code parse once as swapping and fanout is no longer needed. In addition, the manipulation of the circuit is added

```
%%%LIB has 1 column is cell name, 2rd is P of fault, 3th is ITM
%%%4th is inputs, 5th are outputs
% clearvars -except Lib
function  []=
Parse_CKT_pattern(CKT_name,models_dir,outputs,v_in_mean,v_in_std,V_critical,tw_in_mea
n,tw_in_std,Tw_critical)
global Lib CKT;
Lib=cell(0,0);
assign_counter=0;
MyFolderInfo = dir([models_dir,'/*_model.mat']);
for jj=1:size(MyFolderInfo,1)
load([models_dir,'/',MyFolderInfo(jj).name]);
cell_name = string(erase(MyFolderInfo(jj).name,'_model.mat'));
Lib{jj,1}=cell_name;
end
Lib{end+1,1}="assign";Lib{end,2}=0;Lib{end,3}=eye(2);Lib{end,4}=".Q";Lib{end,5}=".A";

first_parse=0;gates_position_in=[];gates_aranged=strings(0,0);
```

```
for parse_counter=1:1
if (parse_counter==1)
    first_parse=1;
else
    first_parse=0;
end
CKT =cell(1,4);

%%%%

fid = fopen(CKT_name);
my_line =  fgetl(fid);
intial_stage=0;
%for i=1:length(outputs)
 %  CKT{1}(i)=outputs(i);
  % CKT{2}(i)= intial_stage;
   %CKT{3}(i)={eye(2)};
   %CKT{4}(i)= i; %% 1 is the bottom, 2 is higher
%end
%counter=length(outputs)+1;

%%StD cells names
STD_names=strings(1,size(Lib,1));
for i=1:size(Lib,1)
    STD_names(1,i)= Lib{i,1};
end
counter=1;
repetion=1;
dummy_counter=0;
all_gates_name=[];all_gates_inputs={};all_gates_outputs={};gates_total_number=1;
all_gates_name_no_in=[];
all_gates_name_no_out=[];
while ischar(my_line)
    % Search for the input of the gate
    my_line = regexprep(my_line,"\("," ");
    my_line = regexprep(my_line,"\)"," ");
    my_line = regexprep(my_line,",","");
    my_line = regexprep(my_line,";","");
    Line_spiltted= split(my_line);

    if(length(Line_spiltted)>2)
        index_in_lib= find(STD_names==Line_spiltted{2});
        if(index_in_lib==(size(Lib,1)))
           dummy= Line_spiltted;
           Line_spiltted{3}=['asssign_dummy',num2str(assign_counter),'_gate'];
           Line_spiltted{4}='.A';Line_spiltted{5}=dummy{5};
           Line_spiltted{6}='.Q';Line_spiltted{7}=dummy{3};
           assign_counter=assign_counter+1;
           clearvars dummy;
%            my_line = fgetl(fid);
%             continue
        end
    if (index_in_lib)
    %match_IN = regexp(my_line,'//  /(\w*)\s+(\w*)','tokens');
        if (size(Lib(index_in_lib,:),2)==1)||(isempty((Lib{index_in_lib,2})))
            if(index_in_lib~=(size(Lib,1)))

charcell(Lib{index_in_lib,1},v_in_mean,v_in_std,V_critical,tw_in_mean,tw_in_std,Tw_cr
itical,index_in_lib);
            end
        end
    %checking gate is connected to output
    Outputs_gate= Line_spiltted(rem(find(Line_spiltted ==
Lib{index_in_lib,4}),length(Line_spiltted))+1);
    Inputs_gate =Line_spiltted(rem(find(Line_spiltted ==
Lib{index_in_lib,5}),length(Line_spiltted))+1);
    index_output= find (outputs == (Line_spiltted{(find(Line_spiltted ==
Lib{index_in_lib,4}))+1}));
    %% saving inputs and outputs of gate %%19/11
    dummy_input=[];dummy_output=[];
    for i=1:length(Inputs_gate)
        dummy_input=[dummy_input " " Inputs_gate{i}];
    end
    for i=1:length(Outputs_gate)
        dummy_output=[dummy_output " " Outputs_gate{i}];
    end
    all_gates_inputs{gates_total_number}=dummy_input;
        all_gates_outputs{gates_total_number}=dummy_output;

        gates_total_number=gates_total_number+1;

    %%
    All_nodes=strings(1,size(CKT,1));
    if(size(CKT,1) >1)
            for i=1:size(CKT,1)
                All_nodes(1,i)= CKT{i,1};
            end
    end
    out_index= find(Outputs_gate==All_nodes);
    [~,in_index_all]
=ind2sub(size(Inputs_gate==All_nodes),find(Inputs_gate==All_nodes));
    %%for maniplate_CKT.m
    all_gates_name=[all_gates_name " " string(Line_spiltted{3})] ;
    all_gates_name_no_in=[all_gates_name_no_in length(Inputs_gate)];
    all_gates_name_no_out=[all_gates_name_no_out length(Outputs_gate) ];
    %%
%my_line
    % saving gates
    if (counter == 1) ||
((~isempty(Outputs_gate==outputs))&&isempty(out_index)&&isempty(find(Inputs_gate==All
_nodes)))
         CKT{counter,1}=string(Line_spiltted{3});
         CKT{counter,2}=intial_stage-1;
         CKT{counter,3}= Lib{index_in_lib,3};
         CKT{counter,8}= Lib{index_in_lib,2};
         CKT{counter,9}=string(Inputs_gate);
         CKT{counter,10}=string(Outputs_gate);
         CKT{counter,5}= 0;
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           Position=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
           Position=1;
         end
         CKT{counter,4}= Position; %% position is high
         counter=counter+1;
         Position_in=Position;
         Position_out=Position;
         for i=1:length(Inputs_gate)
         CKT{counter,1}=Inputs_gate(i);
         CKT{counter,2}=intial_stage-2;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_in; %% position is high
         CKT{counter,5}= 1;
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
         Position_in=Position_in+1; %% each input is in different position
         end
         for i=1:length(Outputs_gate)
         CKT{counter,1}=Outputs_gate(i);
         CKT{counter,2}=intial_stage;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_out; %% position is high
         CKT{counter,5}= 2; %% output
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
         Position_out=Position_out+1;
         end
    elseif (out_index)
        stage_temp= CKT{out_index,2};
        %%19/11
%         cell=CKT{out_index,7}(1);
%         cell_input=find(all_gates_name==cell)/2;
%         if ~isempty(cell_input)
%         other_inputs=
all_gates_inputs{cell_input}(find((all_gates_inputs{cell_input}~=Outputs_gate)&
all_gates_inputs{cell_input}~= " "));
%         gate_connected_other_inputs=[];
%         other_inputs_index= find(other_inputs==All_nodes);
%         for j=1:length(other_inputs)
%             index_dummy = find(CKT{other_inputs_index(j),7} == cell)
%             if (CKT{other_inputs_index(j),4}(index_dummy)
<CKT{out_index,4}(index_dummy))
%                 for i=1:length(all_gates_name)
%                     if ~isempty(all_gates_outputs{i}==other_inputs)
%                        Position_temp=Position_temp+all_gates_name_no_in(i);
%                     end
%                 end
%             end
%         end
%         end
        %%
        CKT{out_index,5} = [CKT{out_index,5} 2]; %it is now output
         CKT{counter,1}=string(Line_spiltted{3});
         CKT{counter,2}=stage_temp-1;
         CKT{counter,3}= Lib{index_in_lib,3};
         CKT{counter,8}= Lib{index_in_lib,2};
         CKT{counter,9}=string(Inputs_gate);
         CKT{counter,10}=string(Outputs_gate);
         CKT{counter,5}= 0;
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           Position_temp=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
            Position_temp= min( CKT{out_index,4});% min of position
         end
         Position_in=Position_temp;
         CKT{counter,4}= Position_temp; %% position is high
                 counter=counter+1;

%% add dummy cell to other inputs if not connected to cell

cell_name =CKT{out_index,7};

    inout=zeros(1,size(CKT,1));

    for i =1:size(CKT,1)
        if ((~isempty(find(CKT{i,5}==1)))&(isempty(find(CKT{i,5}==2))))
                inout(i)=1;
        end
    end
%index_dummy_cell=find(nodes_cell&(inout==1));
if (first_parse)
    index_dummy_cell=[];
else
index_dummy_cell=find((inout==1));
end

for i=1:length(index_dummy_cell)
    %%insert dummy cell
    if (length(CKT{index_dummy_cell(i),4})>1)
        dummy=CKT{index_dummy_cell(i),7}(1);
    else
       dummy= CKT{index_dummy_cell(i),7};
    end
        CKT{counter,1}= dummy+ "_dummy"+dummy_counter;
         CKT{counter,2}=CKT{index_dummy_cell(i),2}-1;
         CKT{counter,3}= eye(2);
         CKT{counter,5}= -1;
         CKT{counter,6}= 1;
         CKT{counter,7}= dummy + "_dummy"+dummy_counter ;
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           CKT{counter,4}=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
            CKT{counter,4}= CKT{index_dummy_cell(i),4}(1); %% position is high
         end
         counter=counter+1;
         %%update inputs
        CKT{counter,1}= CKT{index_dummy_cell(i),1};
         CKT{counter,2}=CKT{index_dummy_cell(i),2}-2;
         CKT{counter,3}= eye(2);

         CKT{counter,5}= CKT{index_dummy_cell(i),5};
         CKT{counter,6}= 1;
         CKT{counter,7}= dummy + "_dummy"+dummy_counter ;
         if(~isempty(find(gates_aranged==CKT{counter,7})))
           CKT{counter,4}=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
            CKT{counter,4}= CKT{index_dummy_cell(i),4}(1); %% position is high
         end
         counter=counter+1;

        CKT{index_dummy_cell(i),1}=CKT{index_dummy_cell(i),1}+ "_dummy"+dummy_counter
;
                 CKT{index_dummy_cell(i),2}=CKT{index_dummy_cell(i),2};
         CKT{index_dummy_cell(i),3}=CKT{index_dummy_cell(i),3};
         CKT{index_dummy_cell(i),6}=CKT{index_dummy_cell(i),6};
         CKT{index_dummy_cell(i),5}=[1 2];
         dummy_counter=dummy_counter+1;
end
%% update inputs
    All_nodes=strings(1,size(CKT,1));
    if(size(CKT,1) >1)
            for i=1:size(CKT,1)
                All_nodes(1,i)= CKT{i,1};

            end
    end
             for i=1:length(Inputs_gate)
             index_in= find((Inputs_gate(i)==All_nodes));
             if (isempty(index_in))
         CKT{counter,1}=Inputs_gate(i);
         CKT{counter,2}=stage_temp-2;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_in; %% position is high
         CKT{counter,5}= 1;
         CKT{counter,6}=repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
            Position_in=Position_in+1;
             else %%% update reption and stage
                CKT{index_in,6} =CKT{index_in,6}+1;
```

```
                 if (CKT{index_in,2}>(stage_temp-2))
                     CKT{index_in,2}=(stage_temp-2);
                 end
                 temp=zeros(2,2*CKT{index_in,6});temp(1,1)=1;temp(end,end)=1;
                 CKT{index_in,3}=temp;
                 CKT{index_in,4} = [CKT{index_in,4} Position_in ];
                 CKT{index_in,7} = [CKT{index_in,7} " " string(Line_spiltted{3}) ];
                 Position_in =Position_in+1;
             end
         end
    elseif (~isempty(in_index_all))
         %in_index_all=[];
         pos_temp=zeros(size(in_index_all));
for j=1:length(in_index_all)
    pos_temp(j) = CKT{in_index_all(j),2};
end
[dummy_temp in_index_index]=min(pos_temp);
in_index=in_index_all(in_index_index);
        stage_temp= CKT{in_index,2};
                %% 19/11

%Position_temp=all_gates_name_no_in(find(all_gates_name==CKT{in_index,7})/2)+max(CKT{
in_index,4});
        %%
        CKT{in_index,5} = [CKT{in_index,5} 1]; %it is now input
         CKT{counter,1}=string(Line_spiltted{3});
         CKT{counter,2}=stage_temp+1;
         CKT{counter,3}= Lib{index_in_lib,3};
                 CKT{counter,8}= Lib{index_in_lib,2};
         CKT{counter,9}=string(Inputs_gate);
         CKT{counter,10}=string(Outputs_gate);
         CKT{counter,5}= 0;
         CKT{counter,6}= repetion;
         CKT{counter,7}=string(Line_spiltted{3});

         if(~isempty(find(gates_aranged==CKT{counter,7})))
           Position_temp=  gates_position_in(find(gates_aranged==CKT{counter,7}));
         else
         Position_temp= min(CKT{in_index,4})+1;
         end
          Position_in=Position_temp;
          CKT{counter,4}= Position_temp; %% position is high
         Position_out=Position_temp;
                 counter=counter+1;

%%add dummy cell to other inputs if not connected to cell
cell_name =CKT{in_index,7};

    inout=zeros(1,size(CKT,1));

    for i =1:size(CKT,1)
        if ((~isempty(find(CKT{i,5}==2)))&(isempty(find(CKT{i,5}==1))))
                inout(i)=1;
        end
    end
if (first_parse)
    index_dummy_cell=[];
else
    index_dummy_cell=[];
%index_dummy_cell=find((inout==1));
end

for i=1:length(index_dummy_cell)
        CKT{counter,1}= cell_name+ "_dummy";
         CKT{counter,2}=CKT{index_dummy_cell(i),2}+1;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= CKT{index_dummy_cell(i),4}; %% position is high
         CKT{counter,5}= -1;
         CKT{counter,6}= 1;
         CKT{counter,7}= CKT{index_dummy_cell(i),7} + "_dummy" ;
         counter=counter+1;
        CKT{counter,1}= CKT{index_dummy_cell(i),1};
         CKT{counter,2}=CKT{index_dummy_cell(i),2}+2;
         CKT{counter,3}= CKT{index_dummy_cell(i),3};
         CKT{counter,4}= CKT{index_dummy_cell(i),4}; %% position is high
         CKT{counter,5}= CKT{index_dummy_cell(i),5};
         CKT{counter,6}= CKT{index_dummy_cell(i),6};
         CKT{counter,7}= CKT{index_dummy_cell(i),7} + "_dummy" ;
         counter=counter+1;

        CKT{index_dummy_cell(i),1}=CKT{index_dummy_cell(i),1}+ "_dummy" ;
                 CKT{index_dummy_cell(i),2}=CKT{index_dummy_cell(i),2};
         CKT{index_dummy_cell(i),3}=eye(2);
         CKT{index_dummy_cell(i),6}=1;
end
%%
         for i=1:length(Inputs_gate)
             index_in_other= find((Inputs_gate(i)==All_nodes));
             if (isempty(index_in_other))
         CKT{counter,1}=Inputs_gate(i);
         CKT{counter,2}=stage_temp;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_in; %% position is high
         CKT{counter,5}= 1;
         CKT{counter,6}=repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
            Position_in=Position_in+1;
             else %%% update reption and stage
                CKT{index_in_other,6} =CKT{index_in_other,6}+1;
                 if (CKT{index_in_other,2}>(stage_temp))
                     CKT{index_in_other,2}=(stage_temp);
                 end
                 temp=zeros(2,2*CKT{index_in_other,6});temp(1,1)=1;temp(end,end)=1;
                 CKT{index_in_other,3}=temp;
                 CKT{index_in_other,4} = [CKT{index_in_other,4} Position_in ];
                 CKT{index_in_other,7} = [CKT{index_in_other,7} " "
string(Line_spiltted{3}) ];
```

```
                 Position_in =Position_in+1;
             end
         end
        for i=1:length(Outputs_gate)
             index_out= find((Outputs_gate(i)==All_nodes));
             if (isempty(index_out))
         CKT{counter,1}=Outputs_gate(i);
         CKT{counter,2}=stage_temp+2;
         CKT{counter,3}= eye(2);
         CKT{counter,4}= Position_out; %% position is high
         CKT{counter,5}= 2;
         CKT{counter,6}=repetion;
         CKT{counter,7}=string(Line_spiltted{3});
         counter=counter+1;
            Position_out=Position_out+1;
             else %%% update reption and stage
                CKT{index_out,6} =CKT{index_out,6}+1;
                 if (CKT{index_out,2}>(stage_temp))
                     CKT{index_out,2}=(stage_temp);
                 end
                 temp=zeros(2,2*CKT{index_out,6});temp(1,1)=1;temp(end,end)=1;
                 CKT{index_out,3}=temp;
                 CKT{index_out,4} = [CKT{index_out,4} Position_in ];
                 Position_out =Position_out+1;
             end
         end


    else
     fprintf("Case is not considered");
    end


     %%%%%%%%%% to be continued dealing with nodes
    end
    end
    my_line = fgetl(fid);
end
%% maniplate CKT

j=1;nodes_reptead=[];nodes_reptead_index=[];nodes_reptead_gates={};nodes_reptead_posi
tion={};
stages =[]; gates_aranged=strings(0,0);gates_no_in=[];gates_no_out=[];
for i=1:length(CKT)
    overwrite=0;
    if(length(CKT{i,4})>1)
        nodes_reptead= [nodes_reptead CKT{i,1}];
        nodes_reptead_position(j)={CKT{i,4}};
        nodes_reptead_index=[nodes_reptead_index i];
        nodes_reptead_gates(j)= {CKT{i,7}};
        j=j+1;
    end
    %% arrange the cells in CKT stages
    stage_index=find(stages==CKT{i,2});
    if (isempty(stage_index)&(CKT{i,5}<1))
        stages=[stages CKT{i,2}];
        stage_index=length(stages);
    end
    if (CKT{i,5}<1)

        if (~isempty((find(all_gates_name == CKT{i,7}))))
            if (CKT{i,4}<=size(gates_no_in,1))&(stage_index <=
size(gates_no_in,2))&(gates_no_in(CKT{i,4},stage_index)~=0)
                overwrite=1;
            old_row_in=zeros(1,size(gates_no_in,2));old_row_out=old_row_in;

old_row_in(stage_index)=gates_no_in(CKT{i,4},stage_index);old_row_out(stage_index)=ga
tes_no_out(CKT{i,4},stage_index);

gates_no_in(CKT{i,4},stage_index)=all_gates_name_no_in((find(all_gates_name ==
CKT{i,7}))/2);
             gates_no_out(CKT{i,4},stage_index)=
all_gates_name_no_out((find(all_gates_name == CKT{i,7}))/2);

gates_no_in=[gates_no_in(1:CKT{i,4},:);old_row_in;gates_no_in(CKT{i,4}+1:end,:)];

gates_no_out=[gates_no_out(1:CKT{i,4},:);old_row_out;gates_no_out(CKT{i,4}+1:end,:)];
            else

gates_no_in(CKT{i,4},stage_index)=all_gates_name_no_in((find(all_gates_name ==
CKT{i,7}))/2);
             gates_no_out(CKT{i,4},stage_index)=
all_gates_name_no_out((find(all_gates_name == CKT{i,7}))/2);
            end
        else
            %dummy_gate
            if (CKT{i,4}<=size(gates_no_in,1))&(stage_index <=
size(gates_no_in,2))&(gates_no_in(CKT{i,4},stage_index)~=0)
                overwrite=2;
            old_row_in=zeros(1,size(gates_no_in,2));old_row_out=old_row_in;

old_row_in(stage_index)=gates_no_in(CKT{i,4},stage_index);old_row_out(stage_index)=ga
tes_no_out(CKT{i,4},stage_index);
            gates_no_in(CKT{i,4},stage_index)=1;
            gates_no_out(CKT{i,4},stage_index)=1;

gates_no_in=[gates_no_in(1:CKT{i,4},:);old_row_in;gates_no_in(CKT{i,4}+1:end,:)];

gates_no_out=[gates_no_out(1:CKT{i,4},:);old_row_out;gates_no_out(CKT{i,4}+1:end,:)];
            else
            gates_no_in(CKT{i,4},stage_index)=1;
            gates_no_out(CKT{i,4},stage_index)=1;
            end
        end
        if (overwrite)
            old_row=strings(1,size(gates_no_in,2));
            old_row(stage_index)=gates_aranged(CKT{i,4},stage_index);
            gates_aranged(CKT{i,4},stage_index)=CKT{i,7};
```

```
gates_aranged=[gates_aranged(1:CKT{i,4},:);old_row;gates_aranged(CKT{i,4}+1:end,:)];
        else
            gates_aranged(CKT{i,4},stage_index)=CKT{i,7};
        end
    end
end
gates_position_in=zeros(size(gates_aranged));
gates_position_in(1,:)=1;
for i=2:size(gates_aranged,1)

        gates_position_in(i,:)=sum(gates_no_in(1:i-1,:),1)+1;

end
end
end
```

IV. Characterize Cell

In this function, the characterization of cell is done to get the probability of error, ITM and PTM of the circuit.

```
function [] =
charcell(cell_name,v_in_mean,v_in_std,V_critical,tw_in_mean,tw_in_std,Tw_critical,Lib
_counter)
rng('default') % For reproducibility
    single_Event=1;

global Lib
Number_of_points_tw=50;
Number_of_points_v=10;
load(['./models/'+cell_name+'_model.mat']);
cell_name_char=char(cell_name);
%% Maximum tw_in is 600, Maximum v_in is 1.8

%assume indpendace of the inputs

    tw_in_v=normrnd(tw_in_mean,tw_in_std,[Number_of_points_tw 1]);
    v_in_v=normrnd(v_in_mean,v_in_std,[Number_of_points_v 1]);

    v_th=1.8;
    tw_th=600;
    P=0;
for i=1:length(v_in_v)
    for j=1:length(tw_in_v)
        tw_out = tw_model1(tw_in_v(j),v_in_v(i));
        v_out = v_model1(tw_in_v(j),v_in_v(i));
        if (tw_out>Tw_critical) && (v_out >V_critical)
            P=P+1;
        end
    end
end
P=P/(length(v_in_v))/length(tw_in_v);
%% AND CELLS
if (string(cell_name_char(1:3)) == 'AND')
    Lib{Lib_counter,1}=cell_name;
    temp=zeros(2^(str2double(cell_name_char(4))),1);
    temp(end)=1;
    Lib{Lib_counter,3}=[~temp temp];
    Lib{Lib_counter,4} = [".Q"];
    No_of_inputs=str2double(cell_name_char(4));
    No_of_prob = 2^(No_of_inputs*2);
    P_t=0;
    % SET can happen for each node or no node at all, SET will cause error and no
logical masking for 0->1 1 and 0 ->1 1  with prob P and 0->1 o->1 with prob P^2
    % for AND P_t=sum(P^iii*nCiii /No_of_prob)
    %%prob of the out switch from 0->1
    if (single_Event==1)
        end_counter=1;
    else
        end_counter=No_of_inputs;
    end
    for iii=1:end_counter
    P_t=P_t+P^(iii)*nchoosek(No_of_inputs,iii)/No_of_prob;
    end
    %%prob of the out switch from 1->0 (same for 0->1 for and)
    P_t=2*P_t;
    Lib{Lib_counter,2}=P_t;
    if (str2double(cell_name_char(4)) == 2)
        Lib{Lib_counter,5} = [".A" ".B"];
    elseif     (str2double(cell_name_char(4)) == 3)
        Lib{Lib_counter,5} = [".A" ".B" ".C"];
    elseif     (str2double(cell_name_char(4)) == 4)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D"];
    elseif     (str2double(cell_name_char(4)) == 5)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E"];
    elseif     (str2double(cell_name_char(4)) == 5)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E" ".F"];
    else
        error("No of input is not in the code please add it")
    end

elseif (string(cell_name_char(1:2)) == 'NA')
        Lib{Lib_counter,1}=cell_name;
    temp=zeros(2^(str2double(cell_name_char(3))),1);
    temp(end)=1;
    Lib{Lib_counter,3}=[temp ~temp];
    Lib{Lib_counter,4} = [".Q"];
    No_of_inputs=str2double(cell_name_char(3));
    No_of_prob = 2^(No_of_inputs*2);
    % SET can happen for each node or no node at all, SET will cause error and no
logical masking for 0->1 1 and 0 ->1 1  with prob P and 0->1 o->1 with prob P^2
    % for NAND P_t=sum(P^iii*nCiii /No_of_prob)
    %%prob of the out switch from 0->1
    P_t=0;
    if (single_Event==1)
        end_counter=1;
    else
        end_counter=No_of_inputs;
    end
    for iii=1:end_counter
    P_t=P_t+P^(iii)*nchoosek(No_of_inputs,iii)/No_of_prob;
    end
    %%prob of the out switch from 1->0 (same for 0->1 for nand)
    P_t=2*P_t;
    Lib{Lib_counter,2}=P_t;
    if (str2double(cell_name_char(3)) == 2)
        Lib{Lib_counter,5} = [".A" ".B"];
    elseif     (str2double(cell_name_char(3)) == 3)
        Lib{Lib_counter,5} = [".A" ".B" ".C"];
    elseif     (str2double(cell_name_char(3)) == 4)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D"];
    elseif     (str2double(cell_name_char(3)) == 5)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E"];
    elseif     (str2double(cell_name_char(3)) == 6)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E" ".F"];
    else
        error("No of input is not in the code please add it")
    end

%% OR NOR cells
elseif (string(cell_name_char(1:2)) == 'OR')
    Lib{Lib_counter,1}=cell_name;
    temp=ones(2^(str2double(cell_name_char(3))),1);
    temp(1)=0;
    Lib{Lib_counter,3}=[ ~temp temp];
    Lib{Lib_counter,4} = [".Q"];
    No_of_inputs=str2double(cell_name_char(3));
    No_of_prob = 2^(No_of_inputs*2);
    P_t=0;
    % SET can happen for each node or no node at all, SET will cause error and no
logical masking for 0->1 1 and 0 ->1 1  with prob P and 0->1 o->1 with prob P^2
    % for OR P_t=sum(P^iii*nCiii /No_of_prob)
    %%prob of the out switch from 1->0
    if (single_Event==1)
        end_counter=1;
    else
        end_counter=No_of_inputs;
    end
    for iii=1:end_counter
    P_t=P_t+P^(iii)*nchoosek(No_of_inputs,iii)/No_of_prob;
    end
    %%prob of the out switch from 1->0 (same for 0->1 for and)
    P_t=2*P_t;
    Lib{Lib_counter,2}=P_t;
    if (str2double(cell_name_char(3)) == 2)
        Lib{Lib_counter,5} = [".A" ".B"];
    elseif     (str2double(cell_name_char(3)) == 3)
        Lib{Lib_counter,5} = [".A" ".B" ".C"];
    elseif     (str2double(cell_name_char(3)) == 4)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D"];
    elseif     (str2double(cell_name_char(3)) == 5)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E"];
    elseif     (str2double(cell_name_char(3)) == 6)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E" ".F"];
    else
        error("No of input is not in the code please add it")
    end


%%
elseif (string(cell_name_char(1:2)) == 'NO')
    Lib{Lib_counter,1}=cell_name;
    temp=ones(2^(str2double(cell_name_char(3))),1);
    temp(1)=0;
    Lib{Lib_counter,3}=[ temp ~temp];
    Lib{Lib_counter,4} = [".Q"];
    No_of_inputs=str2double(cell_name_char(3));
    No_of_prob = 2^(No_of_inputs*2);
    P_t=0;
    % SET can happen for each node or no node at all, SET will cause error and no
logical masking for 0->1 1 and 0 ->1 1  with prob P and 0->1 o->1 with prob P^2
    % for OR P_t=sum(P^iii*nCiii /No_of_prob)
    %%prob of the out switch from 1->0
    if (single_Event==1)
        end_counter=1;
    else
        end_counter=No_of_inputs;
    end
    for iii=1:end_counter
    P_t=P_t+P^(iii)*nchoosek(No_of_inputs,iii)/No_of_prob;
    end
    %%prob of the out switch from 1->0 (same for 0->1 for and)
    P_t=2*P_t;
    Lib{Lib_counter,2}=P_t;
    if (str2double(cell_name_char(3)) == 2)
        Lib{Lib_counter,5} = [".A" ".B"];
    elseif     (str2double(cell_name_char(3)) == 3)
        Lib{Lib_counter,5} = [".A" ".B" ".C"];
    elseif     (str2double(cell_name_char(3)) == 4)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D"];
    elseif     (str2double(cell_name_char(3)) == 5)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E"];
    elseif     (str2double(cell_name_char(3)) == 6)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E" ".F"];
    else
        error("No of input is not in the code please add it")
    end
```

```
%% XOR
elseif (string(cell_name_char(1:2)) == 'EO')
    Lib{Lib_counter,1}=cell_name;
    No_of_inputs=str2double(cell_name_char(3));

    N= 0: 2^(No_of_inputs)-1;
    B=de2bi(N,'left-msb');
    temp=B(:,1);
    for i=1:No_of_inputs-1
        temp=xor(temp,B(:,i+1));
    end
    Lib{Lib_counter,3}=[~temp temp];
    Lib{Lib_counter,4} = [".Q"];
    No_of_prob = 2^(No_of_inputs*2);
    P_t=0;
    % SET can happen for each node or no node at all, SET will cause error and no
logical masking for 0->1 1 and 0 ->1 1  with prob P and 0->1 o->1 with prob P^2
    %%prob of the out switch from 1->0
    if (single_Event==1)
        end_counter=1;
    else
        end_counter=No_of_inputs;
    end

    for iii=1:2:end_counter
    P_t=P_t+P^(iii)*nchoosek(No_of_inputs,iii)/No_of_prob;
    end
    P_t=(2^No_of_inputs)*P_t;
    Lib{Lib_counter,2}=P_t;
    if (str2double(cell_name_char(3)) == 2)
        Lib{Lib_counter,5} = [".A" ".B"];
    elseif     (str2double(cell_name_char(3)) == 3)
        Lib{Lib_counter,5} = [".A" ".B" ".C"];
    elseif     (str2double(cell_name_char(3)) == 4)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D"];
    elseif     (str2double(cell_name_char(3)) == 5)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E"];
    elseif     (str2double(cell_name_char(3)) == 6)
        Lib{Lib_counter,5} = [".A" ".B" ".C" ".D" ".E" ".F"];
    else
        error("No of input is not in the code please add it")
    end
elseif (string(cell_name_char(1:2)) == 'IN')
    Lib{Lib_counter,1}=cell_name;
    Lib{Lib_counter,2}=P;
    Lib{Lib_counter,3}= [0 1;1 0];
    Lib{Lib_counter,4} = [".Q"];
    Lib{Lib_counter,5} = [".A"];
 elseif (string(cell_name_char(1:2)) == 'BU')
    Lib{Lib_counter,1}=cell_name;
    Lib{Lib_counter,2}=P;
    Lib{Lib_counter,3}= eye(2);
    Lib{Lib_counter,4} = [".Q"];
    Lib{Lib_counter,5} = [".A"];
```

```
else
    error("This cell is not characterized, please add it to the code")
end
end
```

V. Manipulation of Circuit

In this function, the circuit is manipulated to sort the circuit according to level and get the ITM of the swapping wire.

```
function
[]=maniplate_CKT3(all_gates_name,all_gates_name_no_in,all_gates_name_no_out,outputs)
global CKT
clearvars -except CKT all_gates_name all_gates_name_no_in all_gates_name_no_out
outputs

j=1;nodes_reptead=[];nodes_reptead_index=[];nodes_reptead_gates={};nodes_reptead_posi
tion={};
stages =[]; gates_aranged=strings(0,0);gates_no_in=[];gates_no_out=[];
stage_array=[];    max_stage=[];nodes_reptead_stage=[];
for i=1:length(CKT)
    if(length(CKT{i,4})>1)
        nodes_reptead= [nodes_reptead CKT{i,1}];
        nodes_reptead_position(j)={CKT{i,4}};
        nodes_reptead_index=[nodes_reptead_index i];
        nodes_reptead_gates(j)= {CKT{i,7}};
        nodes_reptead_stage(j)=CKT{i,2};
        j=j+1;
    end
    %% arrange the cells in CKT stages
    stage_index=find(stages==CKT{i,2});
    if (isempty(stage_index)&(CKT{i,5}<1))
        stages=[stages CKT{i,2}];
        stage_index=length(stages);
    end
    if (CKT{i,5}<=0)
    gates_aranged(CKT{i,4},stage_index)=CKT{i,7};
        if (~isempty((find(all_gates_name == CKT{i,7}))))

gates_no_in(CKT{i,4},stage_index)=all_gates_name_no_in((find(all_gates_name ==
CKT{i,7}))/2);
             gates_no_out(CKT{i,4},stage_index)=
all_gates_name_no_out((find(all_gates_name == CKT{i,7}))/2);
        else
            %dummy_gate
            gates_no_in(CKT{i,4},stage_index)=1;
            gates_no_out(CKT{i,4},stage_index)=1;
        end
    end
    %%
    stage_index=find(stage_array==CKT{i,2});
   if (isempty(stage_index))
        stage_array=[stage_array CKT{i,2}];
        stage_index=length(stage_array);
        max_stage=[max_stage  max(CKT{i,4})];
    end
    if (max(CKT{i,4})> max_stage(stage_index))
         max_stage(stage_index)= max(CKT{i,4});
    end
end

%% Make sure that all the outputs are in the last stage and observable
output_stage=max(stage_array);
for kk=1:length(CKT)

    if(~isempty(find(outputs==CKT{kk,1}))&&(CKT{kk,2}~=output_stage))
        output_name= outputs(find(outputs==CKT{kk,1}));
        pos=CKT{kk,4}(1);
        CKT{kk,1}=[CKT{kk,1} + "_dummynet"];
        current_stage=CKT{kk,2};
       stage_max=max_stage(stage_array==CKT{kk,2});
        for j=1:length(CKT)
             if(CKT{j,2}<=current_stage)
                 CKT{j,2}=CKT{j,2}-1;
             end
        end
        for j=1:length(stage_array)
           if (stage_array(j)<output_stage && stage_array(j)>current_stage)
                 CKT{end+1,1}=[output_name+ "_dummynet_stage"+j];
                 CKT{end,2}=stage_array(j);
                 CKT{end,3}=eye(2);
                 CKT{end,4}=max_stage(j)+1;
           end
        end
        for j=1:stage_max
              CKT{end+1,1}=[output_name+ "_dummynet"+j];
            if(j==pos)
                CKT{end,3}=[1 0 0 0 ; 0 0 0 1];
                CKT{end,4}=[j,stage_max+1];
            else
                CKT{end,3}=eye(2);
                CKT{end,4}=j;
            end
             CKT{end,2}=current_stage;
        end
         CKT{end+1,1}=output_name;
         CKT{end,2}=output_stage;
         CKT{end,3}=eye(2);
         CKT{end,4}=max_stage(stage_array==output_stage)+1;


j=1;nodes_reptead=[];nodes_reptead_index=[];nodes_reptead_gates={};nodes_reptead_posi
tion={};
stages =[]; gates_aranged=strings(0,0);gates_no_in=[];gates_no_out=[];
```

```
stage_array=[];    max_stage=[];nodes_reptead_stage=[];
for i=1:length(CKT)
    if(length(CKT{i,4})>1)
        nodes_reptead= [nodes_reptead CKT{i,1}];
        nodes_reptead_position(j)={CKT{i,4}};
        nodes_reptead_index=[nodes_reptead_index i];
        nodes_reptead_gates(j)= {CKT{i,7}};
        nodes_reptead_stage(j)=CKT{i,2};
        j=j+1;
    end
    %% arrange the cells in CKT stages
    stage_index=find(stages==CKT{i,2});
    if (isempty(stage_index)&(CKT{i,5}<1))
        stages=[stages CKT{i,2}];
        stage_index=length(stages);
    end
    if (CKT{i,5}<=0)
    gates_aranged(CKT{i,4},stage_index)=CKT{i,7};
        if (~isempty((find(all_gates_name == CKT{i,7}))))

gates_no_in(CKT{i,4},stage_index)=all_gates_name_no_in((find(all_gates_name ==
CKT{i,7}))/2);
             gates_no_out(CKT{i,4},stage_index)=
all_gates_name_no_out((find(all_gates_name == CKT{i,7}))/2);
        else
            %dummy_gate
            gates_no_in(CKT{i,4},stage_index)=1;
            gates_no_out(CKT{i,4},stage_index)=1;
        end
    end
    %%
    stage_index=find(stage_array==CKT{i,2});
   if (isempty(stage_index))
        stage_array=[stage_array CKT{i,2}];
        stage_index=length(stage_array);
        max_stage=[max_stage  max(CKT{i,4})];
    end
    if (max(CKT{i,4})> max_stage(stage_index))
         max_stage(stage_index)= max(CKT{i,4});
    end
end


    end
end
%%

for i=1:length(nodes_reptead)
  stage_index= find(stage_array== nodes_reptead_stage(i));
  No_element_in_stage=max_stage(stage_index);
  temp_ITM=speye(2^No_element_in_stage);
  node_interchange=sort(cell2mat(nodes_reptead_position(i)));
 N=(1:2^No_element_in_stage)-1;
 %node_interchange(1)+1 as the swap is between the repeated node and the
 %other node
 b = de2bi(N,'left-msb');
 b_new=[b(:,1:node_interchange(2)) b(:,node_interchange(1)+1)
b(:,node_interchange(2)+1:end) ];
 b_new(:,node_interchange(1)+1)=[];
 N_new=bi2de(b_new,'left-msb')+1;
 %temp_ITM(N_new,:)=eye(2^No_element_in_stage);
 temp_ITM(:,N_new)=speye(2^No_element_in_stage);

 for j=1:length(CKT)
     if(CKT{j,2}<=nodes_reptead_stage(i))
         CKT{j,2}=CKT{j,2}-1;
     end
 end
  CKT{length(CKT)+1,2} = nodes_reptead_stage(i);
  CKT{length(CKT),3} = temp_ITM;
  CKT{length(CKT),4} = 1;
  CKT{length(CKT),1} = nodes_reptead(i)+'_swaped';
  for j=1:length(stage_array)
      if (stage_array(j)<=nodes_reptead_stage(i))
          stage_array(j)=stage_array(j)-1;
      end
  end
  stage_array(length(stage_array)+1)=nodes_reptead_stage(i);
  max_stage(length(stage_array))=1;
  current_stage=nodes_reptead_stage(i);
  for j=1:length(nodes_reptead_stage)
      if (nodes_reptead_stage(j)<=current_stage)
          nodes_reptead_stage(j)=nodes_reptead_stage(j)-1;
      end
  end
end
%% get cells position
% gates=[];gates_pos=[];gates_no_in=[];gates_no_out=[];
% for i=1:length(nodes_reptead_gates)
%     updated_gates=[];
%     for j=1:length(nodes_reptead_gates{i})
%         if ((nodes_reptead_gates{i}(j) ~= " " ) >0)
%             gates = [ gates nodes_reptead_gates{i}(j)];
%             updated_gates=[ updated_gates nodes_reptead_gates{i}(j)];
%         end
%     end
%     nodes_reptead_gates{i}=updated_gates;
% end
% for j=1:length(gates)
% for i=1:length(CKT)
%     if((CKT{i,1})== gates(j))
%         gates_pos=[gates_pos CKT{i,4}];
%         continue
%     end
% end
% gates_no_in = [gates_no_in all_gates_name_no_in((find(all_gates_name ==
gates(j)))/2)];
```

```
% gates_no_out = [gates_no_out all_gates_name_no_out((find(all_gates_name ==
gates(j)))/2)];
% end
end
```

VI. Calculate The Worst Case using Krishnawamy’s method

In this function, the worst-case vector using Krinshnawamy’s method

```
%% calculate ITM/PTM
function []=Cal_Worst_case(inputs,outputs)
global CKT
tic
Result_ITM=1;Result_PTM=1;
ALL_CKT_STAGES=zeros(1,size(CKT,1));
for i=1:size(CKT,1)
    ALL_CKT_STAGES(i)=CKT{i,2};
end
[ALL_CKT_STAGES,index_sorted]=sort(ALL_CKT_STAGES);
CKT_2=CKT(index_sorted,:);
unique_stages=sort(unique(ALL_CKT_STAGES));
counter=1;kron_result_ITM=cell(1,length(unique_stages));
kron_result_PTM=kron_result_ITM;
output_sorted=strings(0);input_sorted=output_sorted;

for i=min(unique_stages):max(unique_stages)

   min_index= find(ALL_CKT_STAGES==i,1,'first'); %%1st index
   max_index= find(ALL_CKT_STAGES==i,1,'last'); %%last index
   SUB_CKT= CKT_2(min_index:max_index,:);
   ALL_SUBCKT_STAGES=zeros(1,size(SUB_CKT,1));
   if(i== min(unique_stages))
        input_sorted=strings(size(SUB_CKT,1),1);
    elseif(i== max(unique_stages))
        output_sorted=strings(size(SUB_CKT,1),1);
    end
   for j=1:size(SUB_CKT,1)
     ALL_SUBCKT_STAGES(j)=min(SUB_CKT{j,4});
   end
    [ALL_SUBCKT_STAGES,index_sorted]=sort(ALL_SUBCKT_STAGES);
    SUB_CKT=SUB_CKT(index_sorted,:);
    CKT_2(min_index:max_index,:)=SUB_CKT;
    kron_result_ITM{counter}=sparse(1);
    kron_result_PTM{counter}=sparse(1);
    for j=1:size(SUB_CKT,1)
     kron_result_ITM{counter} =  kron(kron_result_ITM{counter},sparse(SUB_CKT{j,3}));
     matrix_temp=SUB_CKT{j,3};
     if  ~(isempty(SUB_CKT{j,8}))
         matrix_temp(matrix_temp==0)= SUB_CKT{j,8};
         matrix_temp(matrix_temp==1)= 1-SUB_CKT{j,8};
     end
```

```
          kron_result_PTM{counter} =
kron(kron_result_PTM{counter},sparse(matrix_temp));

    if(i== min(unique_stages))
        input_sorted(j)=SUB_CKT{j,1};
    elseif(i== max(unique_stages))
        output_sorted(j)=SUB_CKT{j,1};
    end

    end
    Result_ITM=Result_ITM*kron_result_ITM{counter};
    Result_PTM=Result_PTM*kron_result_PTM{counter};
    counter=counter+1;

end
%% Compute fidelity
fidelity = Result_ITM .* Result_PTM;
fidelity_min=min(fidelity(fidelity>0));
[x_min ~] = find(fidelity == fidelity_min);

error= 1-fidelity_min;
x_min= x_min-1;
Worst_vector=de2bi(x_min,'left-msb');


sorting=zeros(length(inputs),1);
Worst_vector_new=Worst_vector;
for i = 1: length(inputs)
    sorting(i)=find(inputs ==input_sorted(i));
    Worst_vector_new(:,sorting(i))=Worst_vector(:,i);
end

for i=1:size(Worst_vector,1)
% fprintf('Worst case vector for inputs: [')
% fprintf('%s ', input_sorted);
% fprintf('] with prob of error,%d \n', error);
% fprintf('is [')
% fprintf('%g ', Worst_vector(i,:));
% fprintf(']\n');
fprintf('Worst case vector for inputs: [')
fprintf('%s ', inputs);
fprintf('] with prob of error,%d \n', error);
fprintf('is [')
fprintf('%g ', Worst_vector_new(i,:));
fprintf(']\n');
end
toc
end
```

VII. Calculate The Worst Case using patterns

In this function, the worst-case vector is extracted among patterns that are generated from FastScan. Norm method is a variable that controls the usage of the proposed new metric.

```
 %% calculate ITM/PTM
function []=Cal_Worst_case_pattern2(pattern_file,inputs,outputs)
global CKT;
norm2_method=1;
tic
%  b=1:2^length(inputs);b=b-1;Patterns=de2bi(b,'left-msb');Coverage='100%';
%  Patterns= [0 1 1 1 1; 1 1 1 1 1];
%  Patterns= zeros(1,length(inputs));
 [Patterns,Coverage]= read_patterns(pattern_file);
 if (~norm2_method)
    ITM_Results=zeros(size(Patterns,1),2^length(outputs));PTM_Results=ITM_Results;
 end
norm2=zeros(size(Patterns,1),1);
for jjj=1:size(Patterns,1)
    Pattern=Patterns(jjj,:);
nodes=inputs;
nodes_prob=cell(length(nodes),1);
for i=1:length(nodes)
    if (Pattern(i)==0)
        nodes_prob{i}=[1 0];
    else
        nodes_prob{i}=[0 1];
    end
end
%%

ALL_CKT_STAGES=zeros(1,size(CKT,1));
for i=1:size(CKT,1)
    ALL_CKT_STAGES(i)=CKT{i,2};
end
[ALL_CKT_STAGES,index_sorted]=sort(ALL_CKT_STAGES);
CKT_2=CKT(index_sorted,:);
unique_stages=sort(unique(ALL_CKT_STAGES));
counter=1;kron_result_ITM=cell(1,length(unique_stages));
kron_result_PTM=kron_result_ITM;
output_sorted=strings(0);input_sorted=output_sorted;


for i=min(unique_stages):max(unique_stages)

   min_index= find(ALL_CKT_STAGES==i,1,'first'); %%1st index
   max_index= find(ALL_CKT_STAGES==i,1,'last'); %%last index
   SUB_CKT= CKT_2(min_index:max_index,:);
   ALL_SUBCKT_STAGES=zeros(1,size(SUB_CKT,1));
   if(i== min(unique_stages))
```

```
        input_sorted=strings(size(SUB_CKT,1),1);
    elseif(i== max(unique_stages))
        output_sorted=strings(size(SUB_CKT,1),1);
    end
   for j=1:size(SUB_CKT,1)
     ALL_SUBCKT_STAGES(j)=min(SUB_CKT{j,4});
   end
    [ALL_SUBCKT_STAGES,index_sorted]=sort(ALL_SUBCKT_STAGES);
    SUB_CKT=SUB_CKT(index_sorted,:);
    CKT_2(min_index:max_index,:)=SUB_CKT;
    %kron_result_ITM{counter}=sparse(1);
    %kron_result_PTM{counter}=sparse(1);
    for j=1:size(SUB_CKT,1)
     %kron_result_ITM{counter} = 
kron(kron_result_ITM{counter},sparse(SUB_CKT{j,3}));
          %kron_result_PTM{counter} = 
kron(kron_result_PTM{counter},sparse(matrix_temp));

    if(i== min(unique_stages))
        input_sorted(j)=SUB_CKT{j,1};
    elseif(i== max(unique_stages))
        output_sorted(j)=SUB_CKT{j,1};
    end

    end
    %Result_ITM=Result_ITM*kron_result_ITM{counter};
    %Result_PTM=Result_PTM*kron_result_PTM{counter};
    counter=counter+1;
end
%%

PTM_nodes=nodes_prob;
ITM_nodes=nodes_prob;
for i=min(unique_stages):max(unique_stages)
       min_index= find(ALL_CKT_STAGES==i,1,'first'); %%1st index
   max_index= find(ALL_CKT_STAGES==i,1,'last'); %%last index
   SUB_CKT= CKT_2(min_index:max_index,:);
   for j=1:size(SUB_CKT,1)
       if (SUB_CKT{j,5}==0)
           if(length(SUB_CKT{j,9})==1)
               1;
           end
           PTM_nodes_in=zeros(length(SUB_CKT{j,9}),2);ITM_nodes_in=PTM_nodes_in;
           for k=1:length(SUB_CKT{j,9})
              Pattern_index= find(SUB_CKT{j,9}(k)==nodes);
              if (isempty(Pattern_index))
                   1
              end
              ITM_nodes_in(k,:)=ITM_nodes{Pattern_index,:};
              PTM_nodes_in(k,:)=PTM_nodes{Pattern_index,:};
           end
           nodes(end+1)=SUB_CKT{j,10};
           ITM_nodes{end+1}= cal_prob_cell2(ITM_nodes_in,SUB_CKT{j,3});
              matrix_temp=double(SUB_CKT{j,3});
```

```
             if  ~(isempty(SUB_CKT{j,8}))
%                  SUB_CKT{j,8}=0.01;
                 matrix_temp(matrix_temp==0)= SUB_CKT{j,8};
                 matrix_temp(matrix_temp==1)= 1-SUB_CKT{j,8};
             end
           PTM_nodes{end+1}= cal_prob_cell2(PTM_nodes_in,matrix_temp);
       end
   end

end


PTM_nodes_out=zeros(length(outputs),2);ITM_nodes_out=PTM_nodes_out;

for k=1:length(outputs)
    out_index= find(outputs(k)==nodes);
    if (isempty(out_index))
                   1
    end
    ITM_nodes_out(k,:)=ITM_nodes{out_index,:};
    PTM_nodes_out(k,:)=PTM_nodes{out_index,:};
end
if (~norm2_method)
    ITM_Results(jjj,:)= cal_PTM_cell(ITM_nodes_out)';
    PTM_Results(jjj,:)= cal_PTM_cell(PTM_nodes_out)';
else
    norm2(jjj)=norm(ITM_nodes_out - PTM_nodes_out,2);
end
end

if (~norm2_method)
    fidelity = ITM_Results .* PTM_Results;
    fidelity_min=min(fidelity(fidelity>0));
    error= 1-fidelity_min;
    [x_min ~] = find(fidelity == fidelity_min);
    Pattern=Patterns(x_min,:);
else
    [dummy x_max] =max(norm2);
    [x_max ~] = find(norm2 == (norm2(x_max)));
    Pattern=Patterns(x_max,:);
    error=norm2(x_max);
end




% x_max-x_min


for i=1:size(Pattern,1)
fprintf('Worst case vector for inputs: [')
fprintf('%s ', inputs);
if (~norm2_method)
    fprintf('] with prob of error,%d \n', error);
```

```
else
    fprintf('] with norm2 of error,%d \n', error);
end


fprintf('is [')
fprintf('%g ', Pattern(i,:));
fprintf('] with coverage %s \n',Coverage);

end
toc
end
```

VIII. Calculate Output Probability

In this function, the output probability of the cell is calculated using our new method

```
function  output_prob= cal_prob_cell2(input_prob,PTM)
% in this function we assume that each cell have only one output
%input_pro is [ p(0) p(1)] for each input
N=1:size(PTM,1);
 b = de2bi(N-1,'left-msb')+1;
      Prob_of_input=ones(size(PTM,1),1);
    for i=1:size(b,1)

        for j=1:size(input_prob,1)
        Prob_of_input(i)=Prob_of_input(i)*input_prob(j,b(i,j));
        end
    end
output_prob=0;
for i=1:size(PTM,1)



output_prob=output_prob+PTM(i,:)*Prob_of_input(i);


end
end
```

IX. Calculate the PTM of the circuit

In this function, the PTM and ITM can be calculated from stimulus probability

```
function  Prob_of_output= cal_PTM_cell(output_prob)
% in this function we assume that each cell have only one output
%input_pro is [ p(0) p(1)] for each input
N=1:(2^size(output_prob,1));
```

```
 b = de2bi(N-1,'left-msb')+1;
      Prob_of_output=ones(2^size(output_prob,1),1);
    for i=1:size(b,1)

        for j=1:size(output_prob,1)
        Prob_of_output(i)=Prob_of_output(i)*output_prob(j,b(i,j));
        end
    end

% Prob_of_output=1;
%     for i=1:size(output_prob,1)
%
%   Prob_of_output =  kron(Prob_of_output,output_prob(i,:));
%     end
% end
```