Fall 1-13-2022

Single Event Transient Sensitivity Measurement and Worst-Case Test Vector Exploration for ASIC Devices Exposed to Space Single Event Environment

Mohamed Wael
m_wael@aucegypt.edu

The American University in Cairo

School of Science and Engineering

Single Event Transient Sensitivity Measurement and Worst-Case Test Vector Exploration for ASIC Devices Exposed to Space Single Event Environment

A Thesis Submitted to

Electronics and Communications Engineering Department

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

by Mohamed Wael Mohamed Nady

Under the Supervision of Prof. Ahmed Abou-Auf

Fall 2021

ACKNOWLEDGMENT

I would like to express my honest appreciation to my advisor Prof. Ahmed Abou-Auf for his continuous assistance, mentorship, and support throughout my Thesis.

I would like to thank my examiners Prof. Amr Wassal and Prof. Yehea Ismail, 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, exceedingly Eng. Ahmed Ibrahim, Eng. Ayman Mohamed, Eng. Abdelaziz El-Safty and Eng. Mostafa Abdelaziz.

ABSTRACT

Space radiation and nuclear reactors produce single event effects (SEE) in electronic circuits and impact their performance. The SEE phenomena cause circuits and electronic devices to fail by producing faulty results. Therefore, today's circuit's reliability is a significant concern for all circuit designers.

This thesis suggests a new automated flow to measure the single-event-transient (SET) effects in combinational circuits in application-specific integrated circuits (ASIC) while reaching full fault coverage. The developed flow characterizes the whole circuit nodes by identifying the most sensitive paths to the propagated SET pulses from the node under test to an observable primary output, causing single event upsets (SEUs).

The flow generates test vectors to reach the combinational circuit's highest possible fault coverage percentage. The generated test vectors guarantee that no logical masking for detected SET faults. Then, it analyzes each test vector independently to detect different sensitized paths possible for SET fault propagation. Then, the flow searches for the most sensitive path from the node under test to an observable primary output while measuring the minimal SET pulse characteristics that would produce SEU's. This approach also suggests a new enhanced metric is to identify which test vector enhances the propagated SET pulse within a combinational circuit, which is vital to find worst-case test vectors.

.

Table of Contents

Table of Figures

List of Tables

Chapter 1

Single Event Effects

1.1 Introduction

Single event effects (SEE) in electronics are the dominant source of soft errors in today's commercial integrated circuit technology [1]. Also, commercial electronics operating in normal conditions like those existing in today's cell phones suffer from radiative effects that degrade their reliability. The SEEs have a profound impact on the failure rates in current commercial products, which is why manufacturers and designers always investigate SEEs. In the latest microelectronic products where additional hardening and mitigation techniques are applied to reduce soft error rates, it is found that SEEs are the major contributor to reducing their reliability performance [1].

SEE impacts all electronics either implemented for critical or non-critical applications. It might be acceptable to expect a high failure rate for a non-critical microelectronic device. However, it is crucial to rely on highly reliable circuits whose failure rates are minimal as possible for applications related to safety, health, and life-required circuits. Unmitigated SEEs lead to significant soft error rates causing product failures. Companies that did not design reliable microelectronic devices for their applications would suffer market losses. For example, the phenomenon of SEE reached business news when Forbes, a highly reputable business magazine, reported that a mysterious glitch appeared in high-end servers manufactured by Sun Microsystems for no reason. These sudden glitches appeared due to cosmic rays affecting their SRAM memory inside their servers, generating a high soft failure rate enough, causing problems for Sun Microsystems' customers [2]. The root cause of the problem is that a low-energy neutron activates ^{10}B-doped glass in SRAM cells, increasing the soft error rate in the servers, causing Sun Microsystems to suffer from a significant revenue loss.

Another reported example in 2008, an Air-bus plane was traveling from Singapore to Australia, relying on its autopilot at an altitude of 37000 ft. One of the primary inertial references started generating wrong values of the "angle of attack.", see Figure 1 [3], suggesting that the plane faced a stall. The "angle of attack" is a crucial

parameter that should always be correct to make safe decisions. When SEE caused a soft error, the computer believed that the chord line needed to be adjusted suddenly to keep the airplane steady. The computer then lowered the airplane's nose, suddenly injuring more passengers and crew in the process [4].

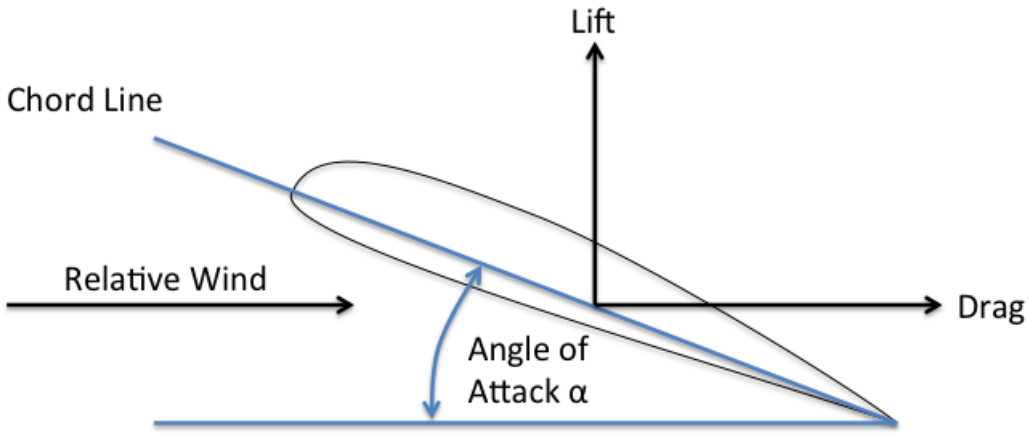


Figure 1, Wind direction and the angle of attack relative to the airplane chord line [3]

The aircraft manufacturer started implementing extra strict measures to prevent this accident from occurring again even though the units met the manufacturer's specifications earlier [1]. Therefore, understanding SEEs is an essential step for designers and manufacturers while building reliable electronic devices for critical applications such as automotive, airplanes, satellites, health ..etc.

Through Chapter 1, a summary of SEE in combinational circuits, besides simulation and modeling approaches to SET. Chapter 2 presents the circuit-level characterization method implemented in this work. In Chapter 3, different types of VLSI faults and an introduction to the SET fault are presented. The SET sensitivity flow is explained in Chapter 4. Chapter 5 suggests a new metric to compare SET testing vector in terms of SET enhancement while SET pulse propagates through the circuit. Chapter 6 suggests future work based on this thesis.

1.2 Errors classification in Integrated Circuits

There are three types of error events in manufactured microelectronic devices: soft, hard and intermittent. A soft error happens when an energetic radiative particle hits a circuit, causing charge disturbance in the affected element, leading to the corruption of

the saved data. After a certain period, the circuit functions as well as expected, and the radiation still did not damage the affected element. On the other hand, the hard error happens when the device itself is permanently damaged and can not function as it is supposed to do anytime in the future. For example, power devices are susceptible to hard errors due to SEEs [1].

Furthermore, the third type is the intermittent error that happens when certain conditions exist; then, the same device cannot function as it is supposed to. However, in normal conditions, it operates well. The intermittent error always happens at the exact location, but soft errors hit random locations and are not predictable.

Another cause of soft errors is electromagnetic interference generated from capacitive and inductive elements in the circuit. Circuits suffer from noise and cross-talks among different wires due to the nature of parasitic interconnects. Induced electromagnetic noise and cross-talks increase in the circuits when interconnect signals change at very high frequencies. Therefore, designers tend to genuinely shield the design and make sure that capacitive cross talks are below an acceptable threshold by applying decoupling techniques and separating noisy sources from the noise-sensitive circuit modules to reduce the induced faults in the circuit [5]. In the thesis, the main focus will be on radiative induced SEEs where an energetic radiative particle hits the circuit and induces a current pulse disturbing the circuit. The affected gate will return to function typically, but the functionality is not correct during the fault time window, causing a transient event. Sometimes, the circuit could take multiple hits affecting not only one gate but multiple gates, especially when the operating circuit is near radioactive stations or in outer space. The high-speed neutrons besides alpha particles are the primary sources of SEEs in microelectronics.

High-energy particles reach earth from outer space consisting of primary protons (89%), hydrogen nuclei, helium nuclei (10%), and other heavy metal nuclei (1%), including uranium [6]. These energetic particles are known as cosmic rays. When cosmic rays enter the earth's atmosphere, they collide with the atoms in the atmosphere, causing many nuclear interactions. As a result of these interactions, high-speed neutrons are generated in the atmosphere [7]. The high-speed neutrons tend to collide more with available air molecules in the air and lose their kinetic energy after each collision. High-

speed neutrons are abundant near outer space, but fewer neutrons exist when they are close to the surface of the earth. The neutron flux, the number of neutrons per time and per area, increases with the distance away from the earth's surface. For example, neutron flux in New York City is 13 neutrons per centimeter squared per hour $\frac{neutrons}{cm^2 * hour}$, but its 144 $\frac{neutrons}{cm^2 * hour}$ at an altitude 10 thousand feet above New York City [7].

Neutron particles are neutral not have any charges. However, neutrons are very energetic, and they can collide with other molecules in the die resulting in generating energetic ions in the affected circuit. The energetic ions have a high speed, creating a charge disturbance while penetrating the CMOS gates' depletion region. This phenomenon results in the electron-hole generation and induces current flow in the CMOS gate. In NMOS, the electrons will move towards the high voltage while holes will

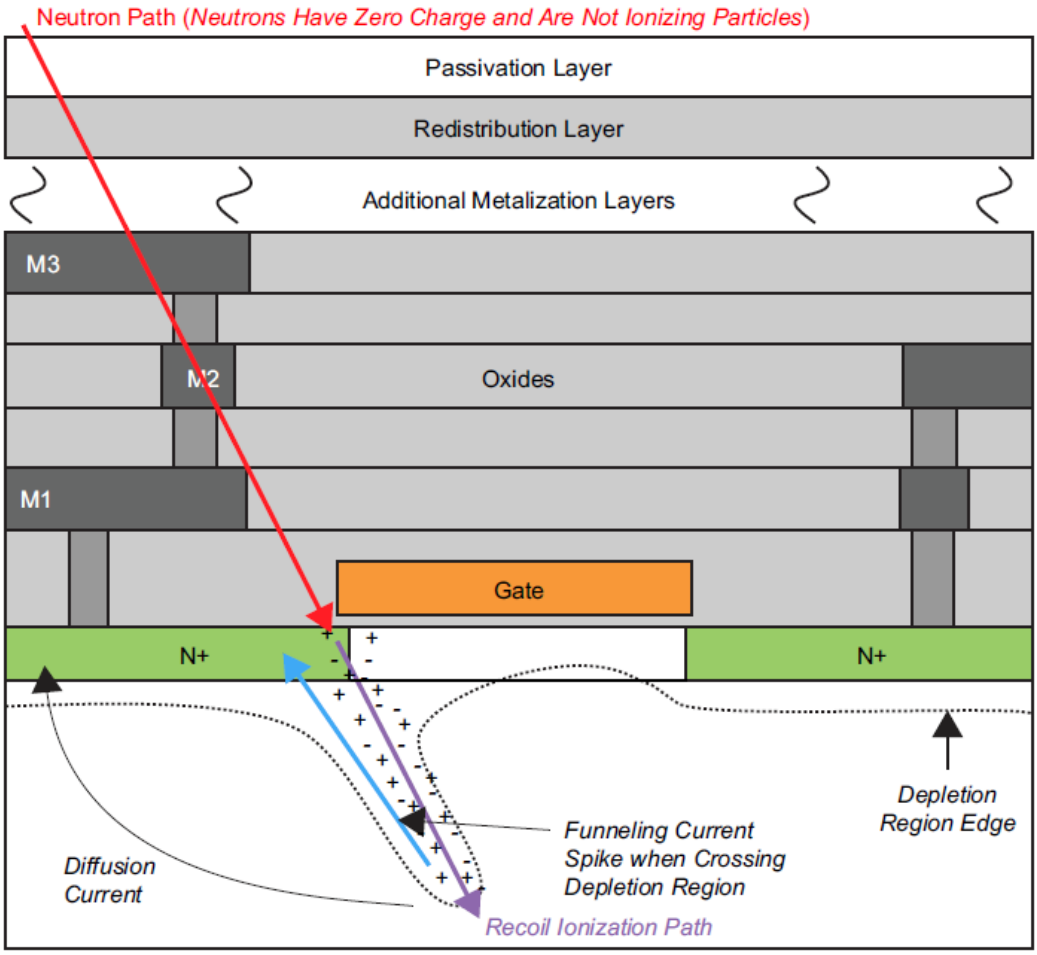


Figure 2 An energetic neutron hits an active NMOS device [7].

move towards the lower voltage transistor terminal; see Figure 2. Alpha particles α, helium nuclei, are generated because of the radioactive decay of packaging materials. They are charged particles able to cause ionization track in microelectronic devices, generating free electrons and holes, as seen in Figure 3. Alpha particles have low energy compared to neutrons, so the only source of alpha particles can penetrate the circuit until the silicon depletion region comes from materials used in die manufacturing.

The chip's primary sources of alpha particles are circuit solder bumps, metals, the under-fill material used in flipped chips, and other packaging materials see Figure 4. The mold compound material used in wire bonding is the typical source for these alpha

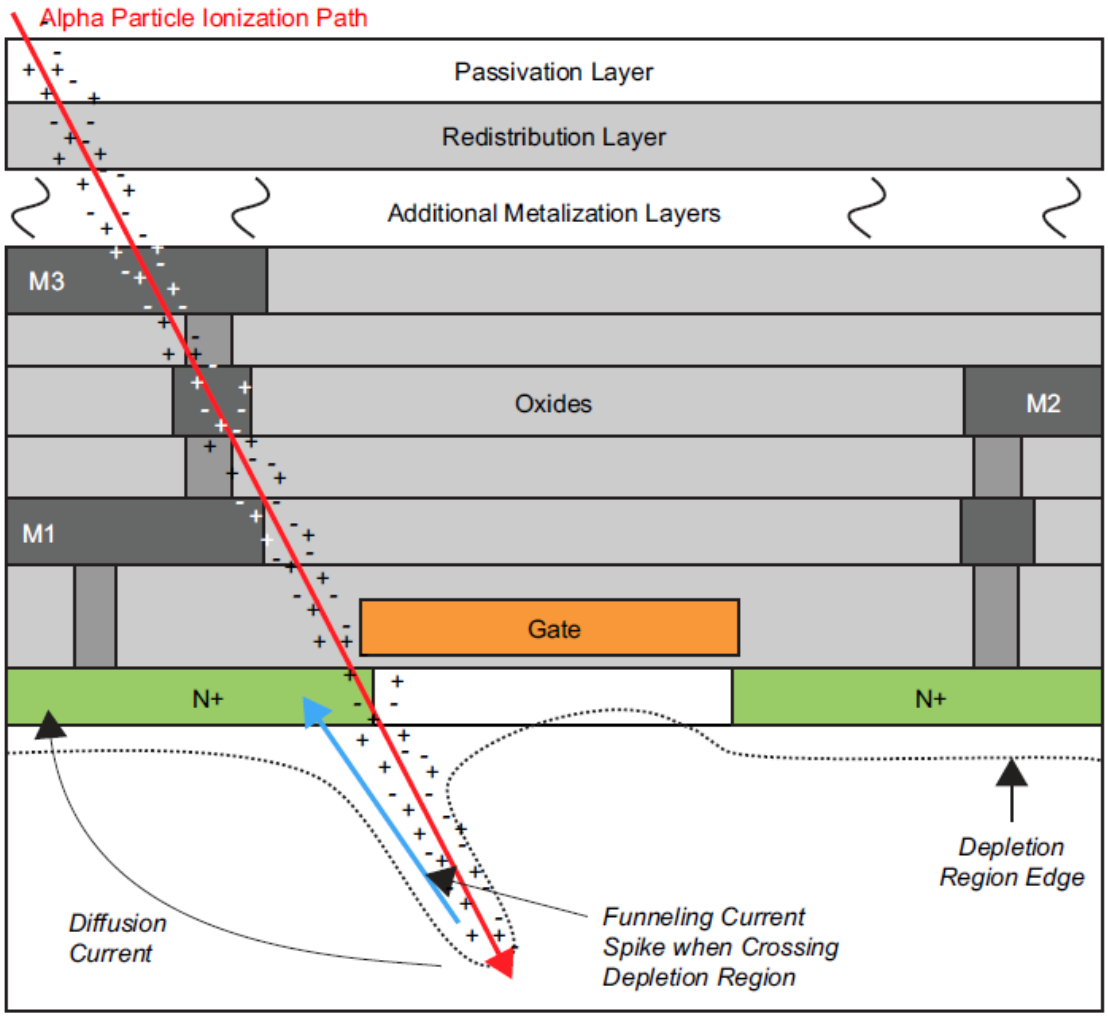


Figure 3, A charged alpha particle hits an active device [7].

particles, as indicated in Figure 4. The single event upset (SEU) effect due to alpha particles emission is measured to be 1.5 $\frac{\alpha}{hour}$ $on\ a\ 300\ mm\ dia.wafer$ [8].

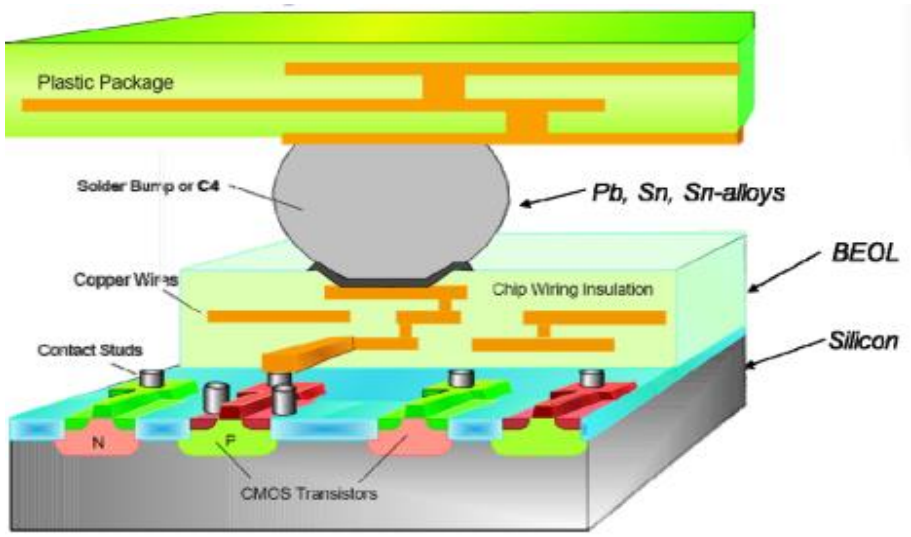


Figure 4 The IC chip structure contains solder bumps, and metals which are the primary sources of alpha particles [8]

1.3 SET Phenomenon and Modelling

The Single Event Transient happens when an external incident particle strikes a circuit node and alters the charges in this node. This effect causes the voltage to change at this node. The charge trapped in this node will cause an additional current in this node. In digital integrated circuits (ICs), this would cause the logic gates voltage to be affected and sometimes be changed, causing a single event upset (SEU) when the voltage value at the output of the gate switches from 0 to 1 or vice versa.

The single event transient (SET) can happen due to radiation coming from space, nuclear reactors, research facilities, and radiative medical setups [9]. Due to the random nature of such radiation, radiative energetic ionized particles could simply strike the surrounding circuit. These ionized particles inject extra charges in the circuit node, and it could easily perturb the voltage at the affected nodes. This perturbation in the voltage waveform is described as a SET. The perturbation changes the shape of the voltage waveform during a certain amount of time, causing a voltage transient at the affected circuit node.

1.3.1 Physical Mechanism of SET—Device-level

Some essential points in CMOS devices need to be highlighted to understand the physical mechanism behind SET. First, the reverse-biased junction in CMOS is very sensitive to ion strikes. Also, the junction in the NMOS transistor between n+ and p well is more affected by ion strikes than $P+$ to n N-well in PMOS because separation and collection of electrons are more significant since the nature of electrons' mobility is larger than holes. Furthermore, manufacturers usually use the p-type bulk, which enhances the collection of charges to appear deeper inside the p-bulk substrate [1].

In Figure 5, a highly energetic ion strikes an NMOS transistor. The ion propagates through the n+|p junction that is reverse biased, leading to the deposition of the charge electron-hole pairs along the particle's track in the device. The local electric field between the reverse-biased n+|p junction collects the carriers rapidly, generating a large voltage/current transient. Furthermore, Electric field distortion in the biased region occurs along the ion track and causes the depletion region to be extended in a funnel structure. The extension of the depletion region in the bulk region increases the collected

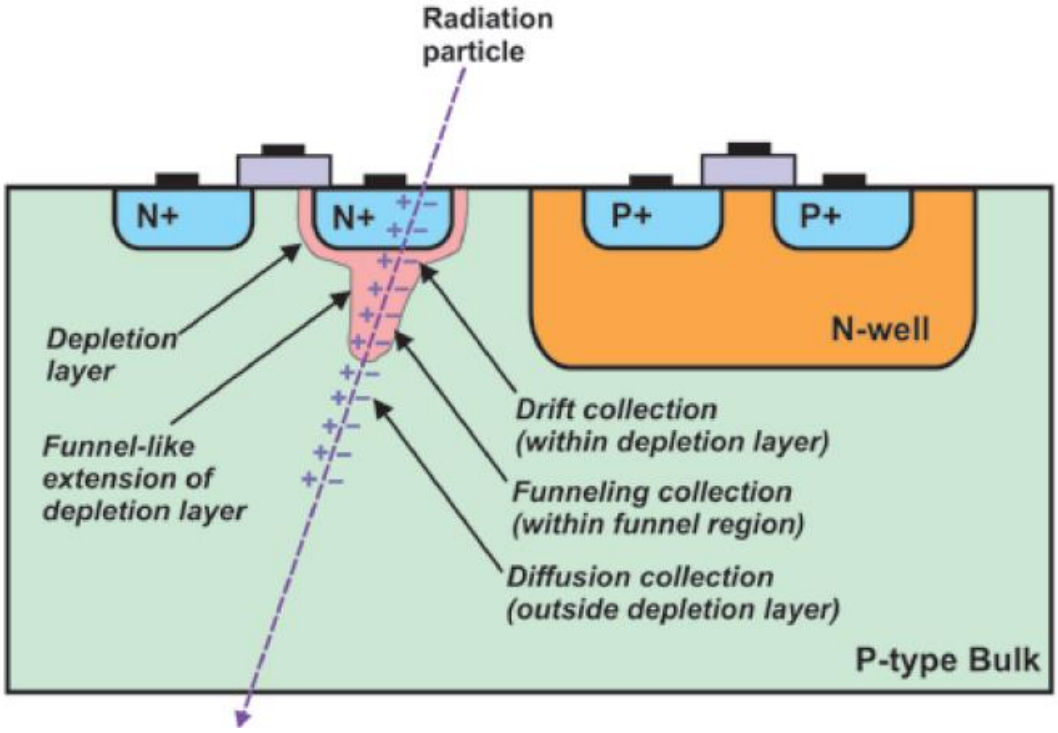


Figure 5 Extra current at drain because of a radiative particle hit. [1]

charge by drifting. The extended funnel volume is a function of doping concentration in the substrate. In the "prompt" phase, the charge collection usually takes tens of pico-seconds.

Then, a second phase occurs when electrons start to diffuse in time into the depletion region. This process continues till all additional carries are collected, recombined, or diffused. The diffusion collection of charges process is much slower than the prompt collection of them, as illustrated in Figure 6 [1].

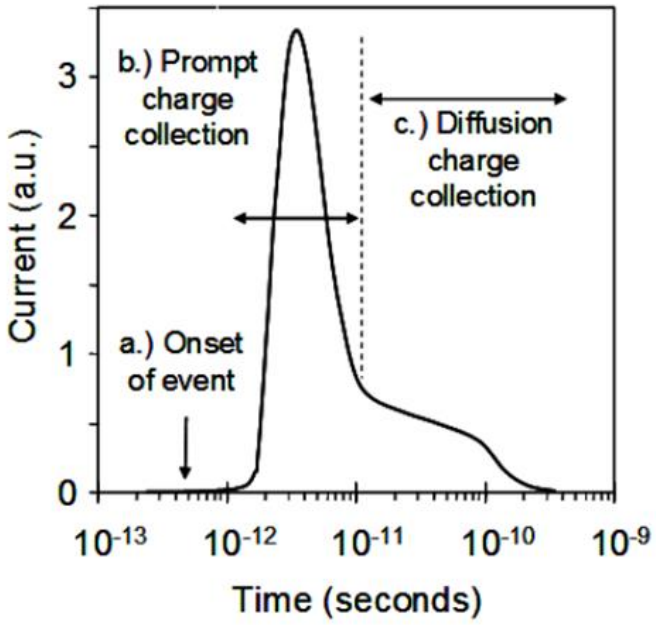


Figure 6 The current waveform after a strike by a radiative particle [1]

1.3.2 Physical Mechanism of SET—Gate Level

In Figure 7, the CMOS inverter has a low input voltage when a highly energetic ionized particle strikes the NMOS transistor. The deposited charges caused by the incident particle are collected as described in the previous section and introduce a new SET current I_{set}. The load capacitance charges could now go to the ground through the ON NMOS, and the voltage value of the load capacitance starts to change. The PMOS is still in ON state and starts to drive current I_{drive} to dissipate the SET current besides the load current I_{load}.

The output voltage is deformed, and it can be displayed as a pulse waveform. In the above case, the SET perturbation would cause a pulse from high "1" towards low "0". If it has a large swing and operates for a considerable period, this pulsed-wave causes the logic value to be transitioned from 1 to 0.

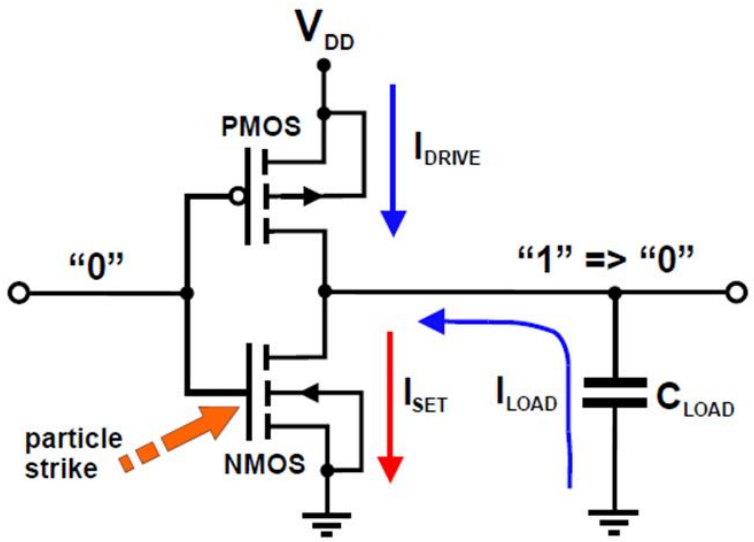


Figure 7 A new current pulse generated by an incident radiative particle [9]

The amount of charge that appeared after particle radiation hit that can transform the output voltage value from 1 to 0 or from 0 to 1 is called the critical charge $Q_{critical}$. Therefore, the standard cell gates are immune to the SET phenomenon if they exhibit high $Q_{critical}$ values, and hence, only high energetic particles can produce single event upsets (SEU). If the induced charges by the radiative particle hit are not sufficient for the output signal voltage to flip, electrical masking occurs, and no SET pulse propagates through the gate [10].

1.3.3 Logical masking and fault propagation

In a large combinational circuit, studying the conditions to facilitate the single event transient error propagation through the IC circuit is another crucial aspect. The logic function of the AND gate and the input voltage values of the gate control the propagation of the SET fault at the gate output. For example, if a radiative particle strikes one of the input terminals of an AND gate while it has a "0" low voltage before the SET pulse, the radiative particle will cause a pulsed voltage waveform from "0" to "1" for a period t_{set} . Suppose the non-affected AND gate terminal has a "1" high voltage, and the SET pulse is presented in the second terminal. In that case, the AND gate output exhibits SET pulse, as illustrated in Figure 8-b. Then, the gate output will return to its original logic value "0" after the end of the SET effect. On the other hand, if the non-affected terminal has a "0" voltage, it will not permit the SET pulse to appear in the gate output, as illustrated in Figure 8-a.

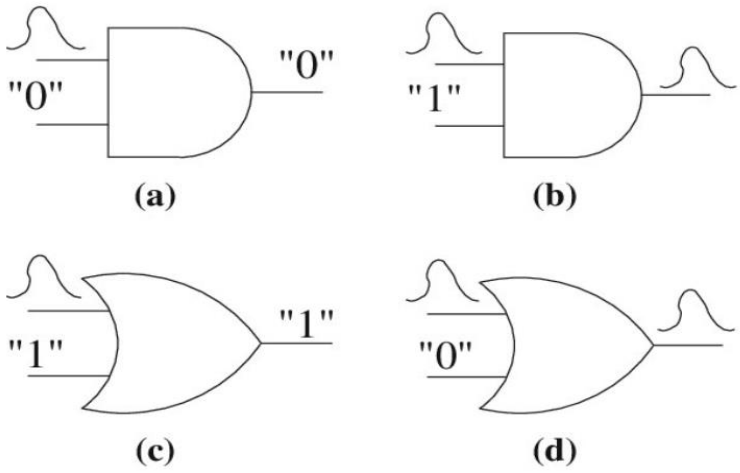


Figure 8 SET propagation dependance on the logic values at the gate inputs. "a" & "c" illustrates that the SET propagation while "b & "d" no SET propagation.

The same scenario applies to different logic gates but with different input logic values. In Figure 8-c and Figure 8-d, the SET hits a NOR gate, and the affected node has a low "0" voltage value. If the other terminal has a high "1" voltage value, the SET pulse input will not propagate through the gate, and the output voltage will remain "1". On the contrary, if the voltage is "0", as in Figure 8-d, the SET will propagate.

Therefore, the logic function of the gates is also an essential factor determining the propagation of the SET fault in the circuit. Therefore, “Logical Masking” exists when the logic input stops the input SET pulse from propagating to the gate output [11].

1.3.4 SET at P-MOS and N-MOS

An experimental characterization experiment was conducted in [12] to distinguish the charge collection mechanism of the SET phenomenon happening in either N-MOS and P-MOS transistors in CMOS logic gates. In [12], they fabricated two custom CMOS circuits for N-hits and P-hits. Each circuit consists of a repetitive 100 unique blocks. The N-hit circuit consists of two inverters connected each at the terminals of a NAND gate, as shown in Figure 9. The input to each block is supposed to be voltage high “1”. On the other P-hit circuit, the two inverters are connected to a NOR gate while the voltage input is low “0”, as shown in Figure 10. The chip used 65nm bulk CMOS technology, and It was tested using a heavy-ion setup. The block design will logically mask ion-strikes inducing SET pulsed current at the inverter’s active area. Therefore, the only effect of propagating through the block is the ion hitting the active areas of the NAND and NOR gates. Considering the N-hit circuit, if the ion hits one of the inverters, causing the logic output to change from “0” to “1”, the other inverter should maintain its “0” low voltage value and prevent inverter SET induced pulse from propagation. Therefore, only ions hitting “off” NMOS transistors in the NAND gate would generate a SET pulse and change the block output voltage. The same methodology also applies to the P-hit block, with only ions hitting “off” P-MOS transistors to change the P-it block voltage.

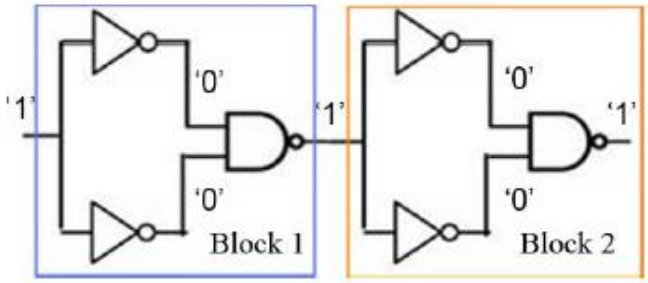


Figure 9 Two blocks of N-hit circuits connected in series [12]

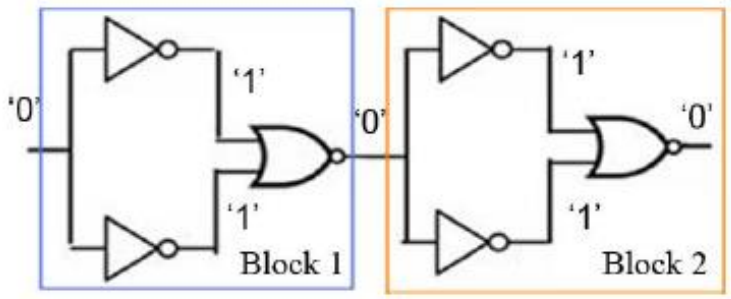


Figure 10 Two blocks of P-hit circuits connected in series [12]

The inverter, NAND, and NOR gates were designed to have the same current drive to lower the differences between both blocks [12]. The custom design of the two layouts considers separating both inverters away from each other by 3.5 μm to prevent charge sharing in the active area between these two inverters [13]. A layout of two N-hit blocks illustrating how the backend design separates two inverters from each other was introduced in [14], as depicted in Figure 11. If both inverters were not separated, the heavy-ion could simultaneously induce SET pulsed current at both inverters, changing the block logic value. Also, each block is spaced 2.5 μm behind the consecutive one to prevent charge sharing between NAND gates and the following inverters of the next block in a phenomenon known as "SET quenching" [15].

In order to calculate the induced SET pulse width, an on-chip measurement circuit consisting of a long inverter chain is also implemented. The pulse width is to be measured in terms of inverter stage delays [12]. The inverter chain consists of 80 consecutive inverters when each inverter delay is $25ps$. Therefore, the pulse width will be multiple of the inverter delay. The measurement circuit could measure SET pulses ranging from 12.5 μs to 2 ns.

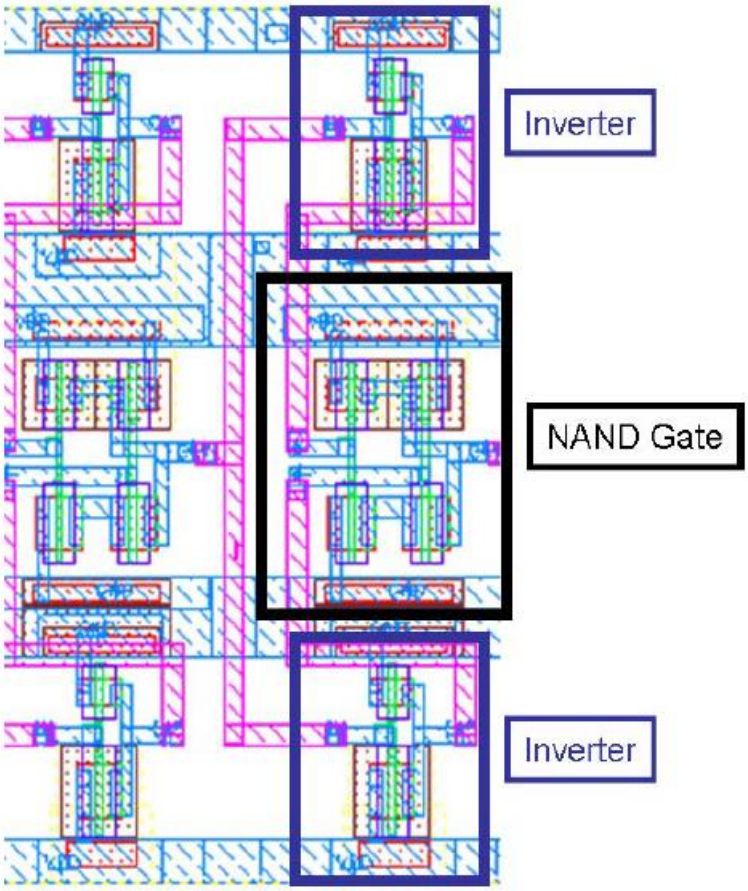


Figure 11 The layout of a N-hit block [14]

Experimental testing with heavy ions with different linear energy transfer (LET) starting from 21.2 $\frac{MeV\ cm^2}{mg}$ to 58.8 $\frac{MeV\ cm^2}{mg}$. The energy of ions was measured at the silicon surface and the SET propagation pulse width. In Figure 12, the box plot shows the mean, minimum, maximum, and ± 1 standard deviation of the measured SET pulse width data versus different input heavy-ion energies used in the experiment. It appears that at lower LETs, the SET pulse width originating from N-hits is larger than those P-hits. The reason is that the perturbation of the well voltage is not considerable and only charge collection exists due to drift and diffusion charge collection contributes to the SET pulse.

On the other hand, the perturbation of the well voltage becomes more significant after increasing the incident ion energy. The input heavy-ion penetrates deeply in the well, causing bipolar parasitics to contribute significantly to the charge collection process.

Considering the n-well in the PMOS transistor, holes collected by the drain or the substrate cause electrons to lower the well potential, resulting in a lower source-well potential barrier, defined as parasitic bipolar action [12]. The parasitic bipolar action increases the induced current collected at the drain, resulting in larger SET pulse width measured in P-hit circuits larger than pulses from N-hit circuits. Circuits built using dynamic logic circuits, where only one type of transistor is used while designing the circuit, benefitted from this experiment because they could choose their type of transistor due to the expected operating conditions.

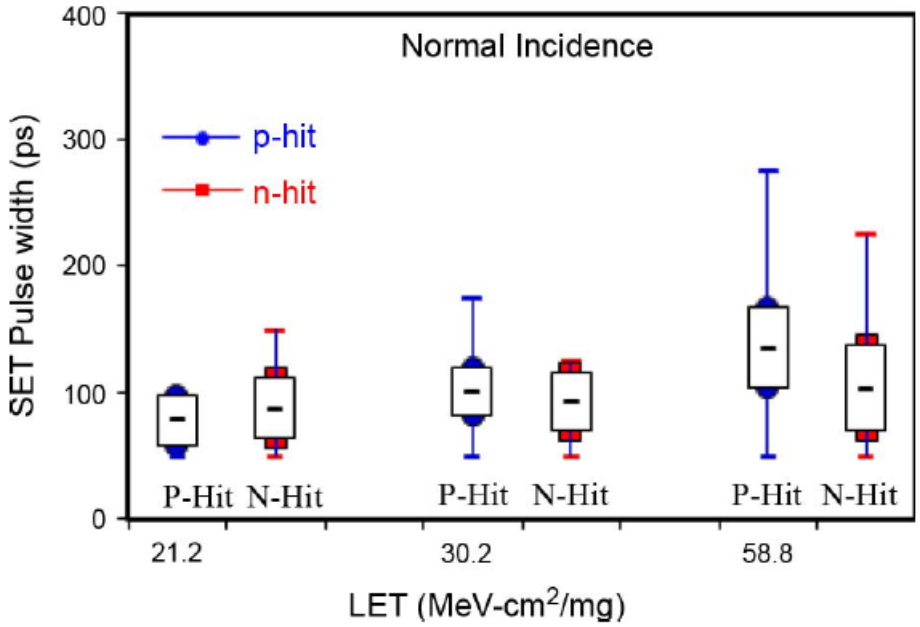


Figure 12 A box plot of SET pulse width corresponding to LET ranging from 21.1 to 58.8 $\frac{MeV\ cm^2}{mg}$ [12]

1.3.5 Technology Scaling Trends in SET

Two chips were fabricated using 130-nm and 90-nm CMOS technology to study the effect of transistor scaling for single event transient [16]. Their design consisted of a custom inverter target circuit and SET pulse capture circuit, and they were tested using

heavy ions. The operating voltage of both circuits was the same, 1.2 V. The operating conditions were the same except for the transistor scaling of both technologies.

Figure 13 compares the measured SET pulses as a function of the incident LET of the heavy ions. At low LET values, there is a significant difference between both technologies. The 90-nm circuit observes larger SET pulse width than the 130-nm circuit. On the other side where LET values are high, the results are comparable between the two technologies in terms of the SET pulse widths. Figure 13 proves that the SET effects are increasing with smaller technologies. In smaller technologies, the wider SET pulses prove that new future nodes will be more sensitive to SET events.

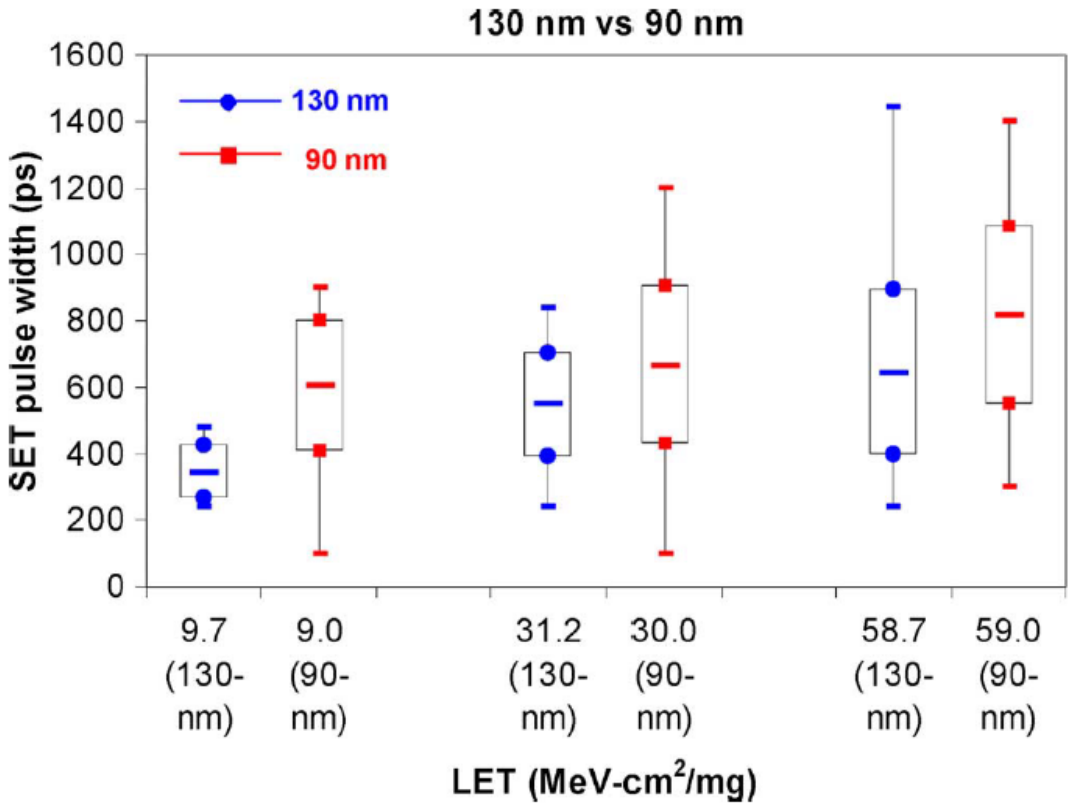


Figure 13 A comparison between 130nm and 90nm in terms of LET [16].

1.4 Simulation and Modelling of SET

Controlled irradiation experiments are essential to study the influence of energetic particles on operating circuits. Tested devices in the circuit should consider only the effect of the energetic particles after bombarding them with laser or heavy ions with a previously known energy and start to monitor the induced waveforms afterward.

However, there are considerable challenges to lab irradiation experiments [17]. First, controlling the bombarded particle to strike a specific position is a challenging operation that requires expensive characterization tools [12]. New small node technology increases the challenge level because of scaling down the transistors. Researchers tend to increase the device area as possible to increase controllability and prevent the energetic particle from unintentionally affecting another device [16]. Second, special circuit design requirements impose an extra load of difficulties. Monitoring the induced SET current or voltage pulses cannot be done using external probes because the induced pulse will change after moving through the highly parasitic wires. Therefore, built-in self-test circuits are proposed to measure SET pulse widths on the same chip using a chain of inverters [14]. Finally, the custom chip requires fabrication and sometimes custom packaging, so extra enhancements or refinements are unavailable. In brief, this lab testing requires time and resources, and it is not easy to implement.

1.4.1 Classification of SET Modeling and Simulation Approaches.

1.4.1.1 Device-Level Simulation

On the other hand, doing a computer simulation of the circuits under test by modeling the SET effects on the circuits is the optimal approach used by everyone. There are different levels at which researchers study the SET effects. First, device-level simulation studies the SET effects by applying detailed physics mechanisms between the interactions of the energetic particle and the induced charges in the active device. It is the most accurate level of simulation [9]. Still, it can only be applied to a small number of connected transistors, not on a circuit level, because of the enormous computational resources required to perform such a simulation. Device-level simulation can only be applied to develop an electrical model or closed analytical equation to the charge sharing mechanism in the device. For example, TCAD simulations were implemented to study the quenching of SET in circuits in [15]. The developed TCAD structure in [15] to study the SET quenching is shown in Figure 14. Overall, the developed simulation will produce an abstract model of the SET-induced current or voltage waveform in the affected active devices and be used in SPICE circuit-level simulations.

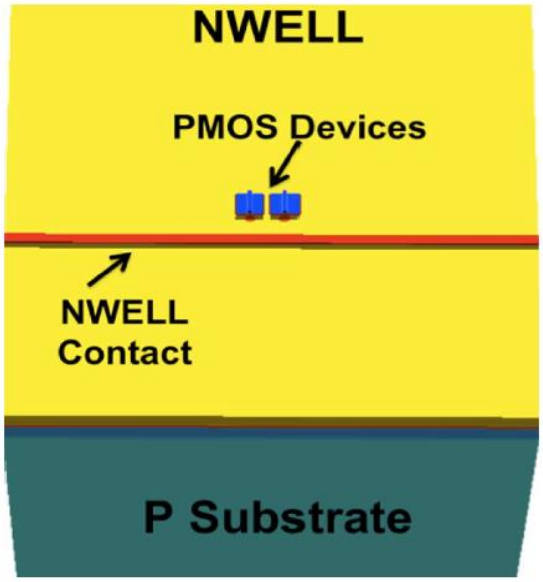


Figure 14 A 3D TCAD model of two adjacent PMOS transistors [15]

1.4.1.2 Circuit-Level Simulation.

The second level of SET simulation is SPICE simulations of circuit-level designs. The SPICE simulation, circuit-level, consists of two mechanisms for introducing the induced SET pulse in the circuit. The first mechanism is Micro-Modelling, where a pulsed current source is embedded inside the transistor under test in this mechanism [9], as shown in Figure 16. The effect of the SET pulse is monitored using real-time SPICE simulations at the circuit level.

The second mechanism is known as Macro-Modelling, where a pulsed current source is injected at a circuit net between two different standard gates [9], as shown in Figure 15. This mechanism also uses real-time SPICE simulations, the SET pulse in the tested circuit, and monitors the propagated SET pulse until a primary output if no logical masking occurs.

The Macro-Modelling mechanism is widely used in the literature because it is easier to implement in CAD simulators. The Micro-Modelling mechanism requires a modification to the predefined transistor model file produced by the foundry, which sometimes does not allow its users to make such changes. In this work, the Macro-Modelling approach is to be used while applying circuit characterization or validating the SET effects on the whole combinational circuits.

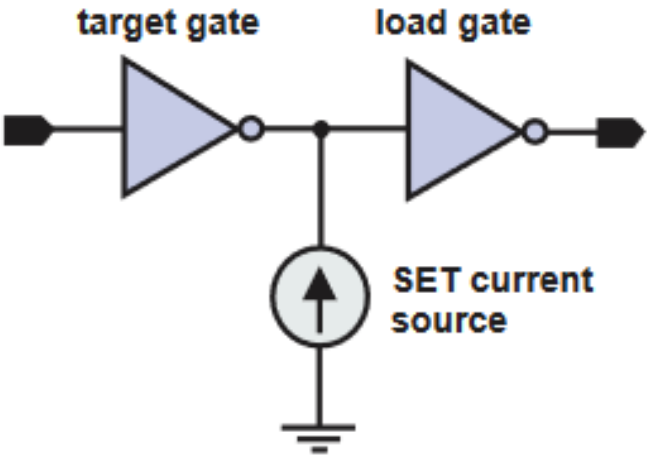


Figure 15 Macro-modelling of SET pulse in circuit-level [9].

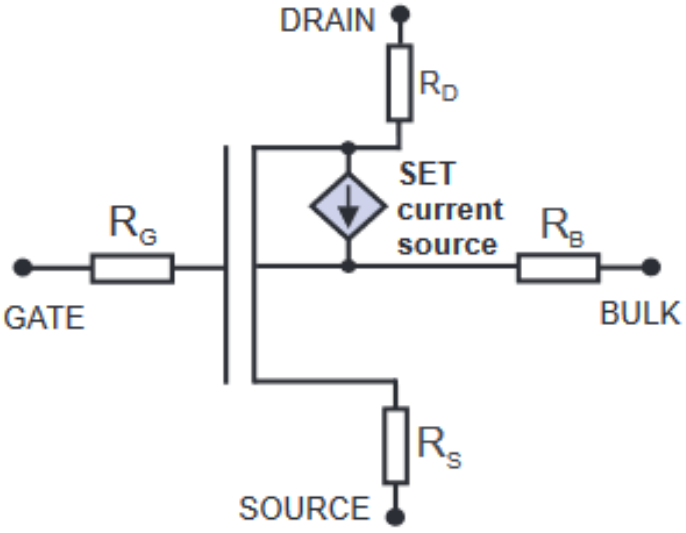


Figure 16 Micro-modelling of SET pulse inside a transistor under test [9].

1.4.1.3 Mixed-Mode Simulation.

Sometimes, mixed-mode simulation is applied to take advantage of both previous schemes. An example of employed mixed-mode simulation is presented in [18]. They designed a chain consisting of four inverters. The energetic particle strikes the off-state NMOS transistor of the second inverter. The 3D-TCAD simulation was applied only on the affected NMOS to study the SET propagation in the four inverter chain shown in Figure 17, while the other inverters were simulated using SPICE simulation.

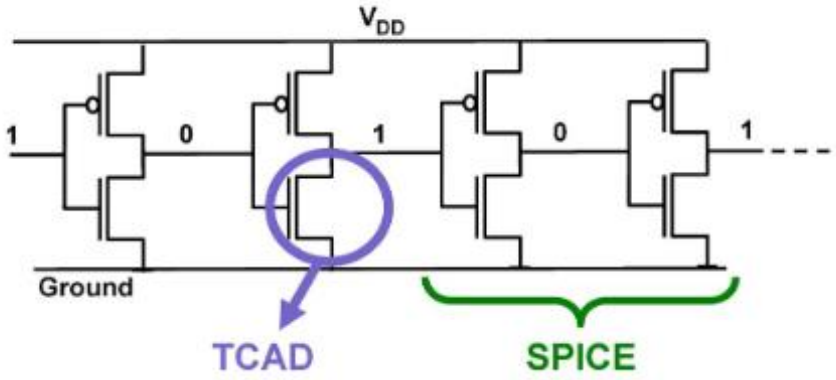


Figure 17 A mixed-mode simulation example [18].

Another vital contribution of the mixed-mode simulation was presented in [19]. The mixed-mode simulation performed by Davinci, a commercial mixed-level device and circuit simulator, was able to compute the voltage and current waveforms induced by the energetic particles. Previously, researchers could only measure the induced SET pulse width using inverter delay chains. Therefore, applying the mixed-mode simulation was a revolutionary step. The model implemented in [19] consists of a 10-inverter chain, followed by a broadening inverter and a set-reset latch. The broadening inverter consists of a strong NMOS and weak PMOS to stretch the negative-going (transition from VDD to GND) transient pulses. The latch's objective is to measure the propagated SET pulse. The input of the 10-inverter chain is connected to ground "0", as shown in Figure 18. The simulations included both bulk and SOI transistors.

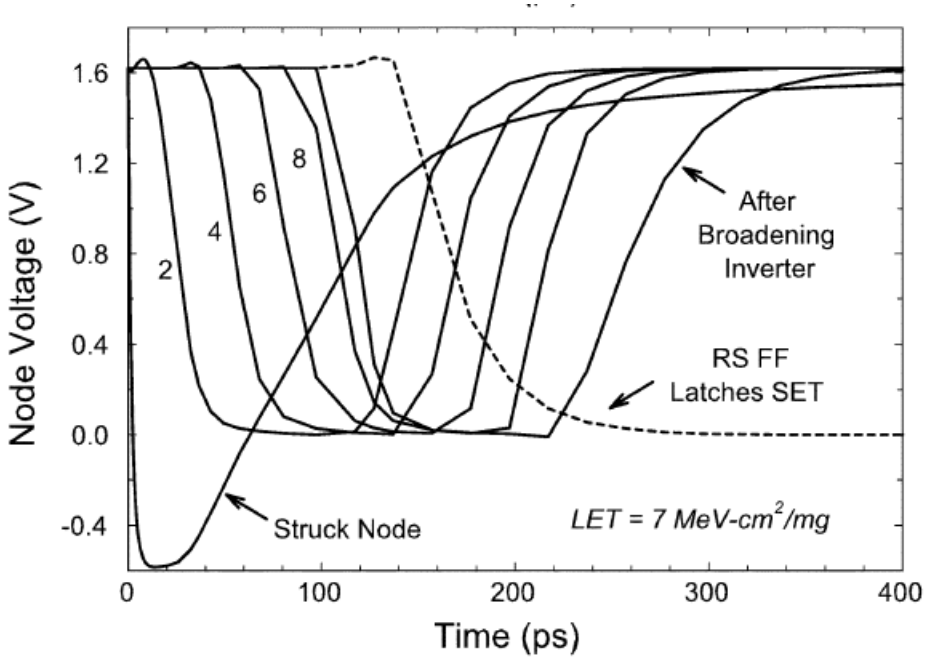


Figure 19 The SET voltage waveform propagation from inv1 to the set-rest latch in a 180nm bulk CMOS technology [19].

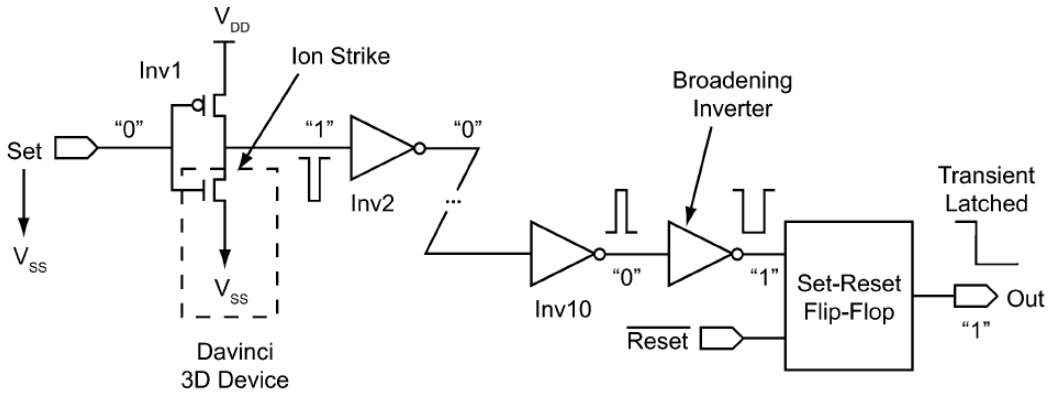


Figure 18 The schematic of the design implanted on Davinci mixed-mode simulator.

An energetic ion hits the off-state NMOS in the first inverter. The NMOS transistor is simulated using the device-level scheme, while the circuit-level simulation is applied to the other devices. The SET propagates through the inverter chain and is latched at the end of the circuit, meaning it is captured by a sequential element leading to a single event upset (SEU). With $LET = 7\ MeV - cm^2/mg$, the mixed-mode simulation shows that propagated SET is captured by the set-rest latch causing a single event upset (SEU), as shown in Figure 19. Therefore, any particle with a LET of more than 7 can also cause SEU. Also, it shows that the propagated SET pulse is wider than the induced SET pulse at the struck node by 10 to 15 percent [19].

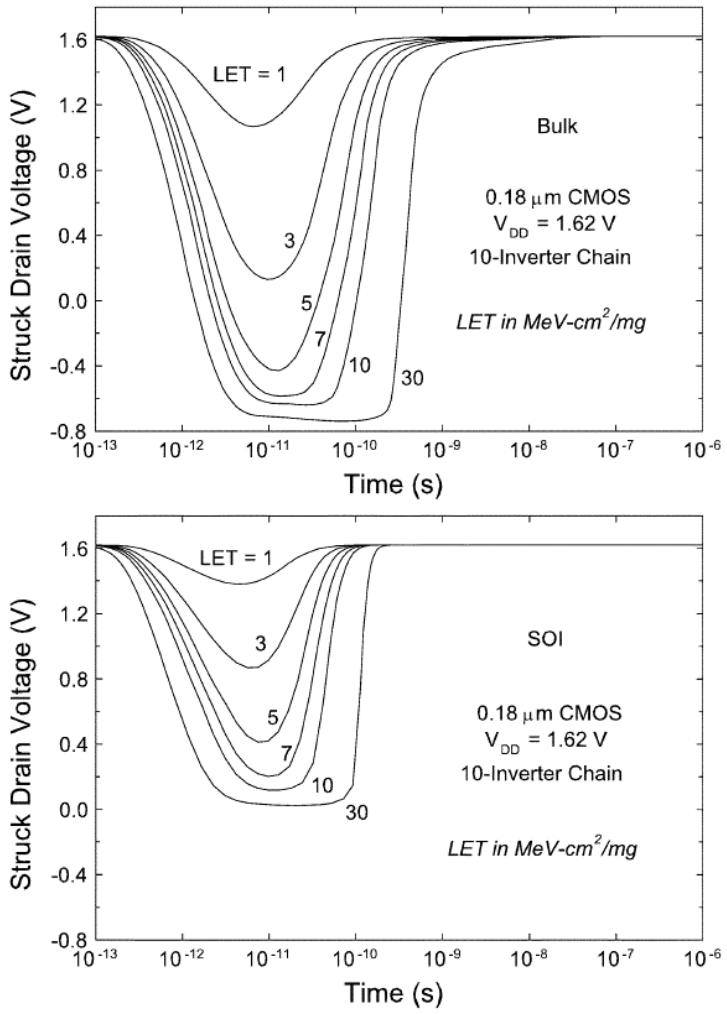


Figure 20 Drain voltage waveform at different LET for bulk and SOI 180 nm CMOS transistor [19].

The mixed-mode simulation gives a degree of freedom to change the energy of the radiative particle, so the voltage waveform resulting from the different energy bombardment was generated, as shown in Figure 20.

Also, the mixed-mode simulation predicted the critical LET value required for SET propagation in the circuit. The critical LET is plot versus different transistor feature sizes for bulk and SOI CMOS in Figure 21. It is shown that scaling transistor down causes CMOS to be more susceptible to energetic particles strikes and bulk CMOS below 100 nm are susceptible to alpha particles hit. Furthermore, SOI CMOS shows more robustness to SET strikes.

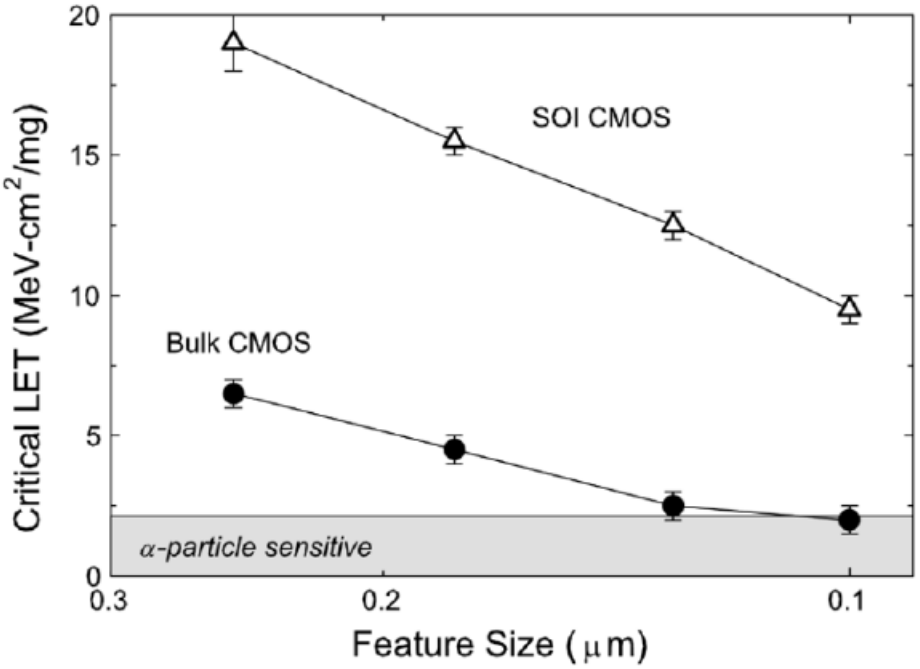


Figure 21 Critical LET of SET propagation versus different transistor feature sizes.

In Figure *22*, an important plot is presented, showing the resulting SET pulse width at the struck node versus the linear energy transfer of the striking, energetic particle [19].

The plot shows different responses of different technology nodes, including the 180nm bulk CMOS technology.

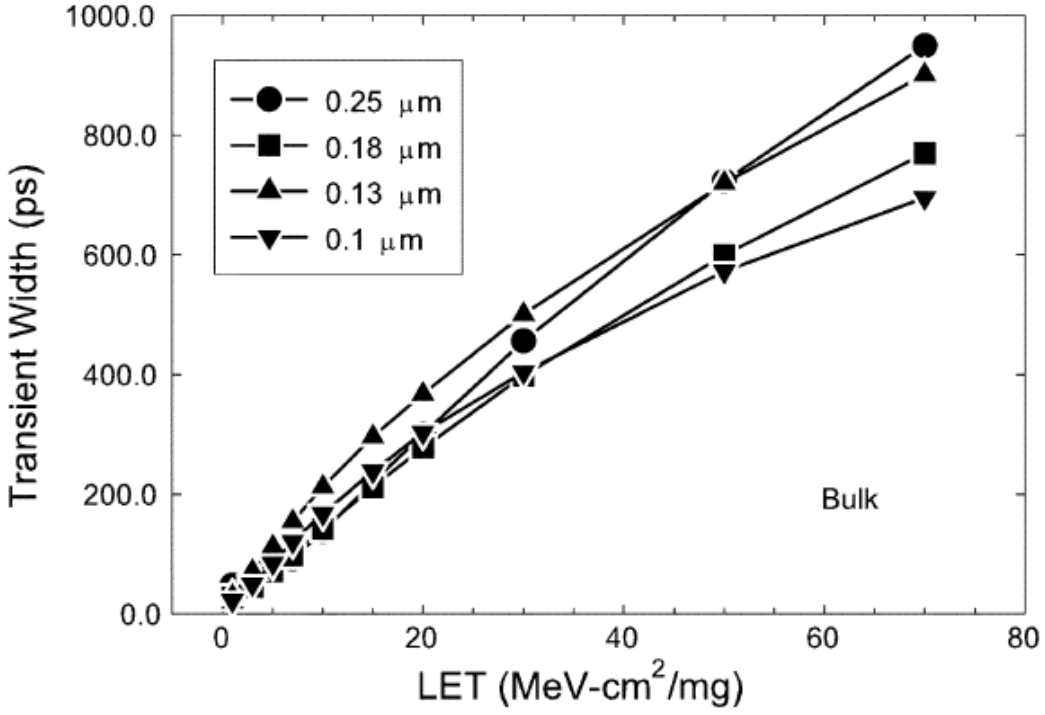


Figure 22 SET pulse width vs LET

1.5 The SET Current Source Model.

The current pulse developed in [20] is considered the most used current source model for SET simulations. The SET current can be modeled according to the following equation:

$$I_{SET}(t) = \frac{Q_{COLL}}{\tau_f - \tau_r}\left(e^{-\frac{t}{\tau_f}} - e^{-\frac{t}{\tau_r}}\right)$$

Where Q_{coll} is the collected charge, τ_f is the collection time constant of the junction, τ_r is the ion-track establishment time constant. The τ_f is responsible for the fall time of the pulse while τ_r is responsible for the rise time. A plot of the current pulse is illustrated in Figure 23.

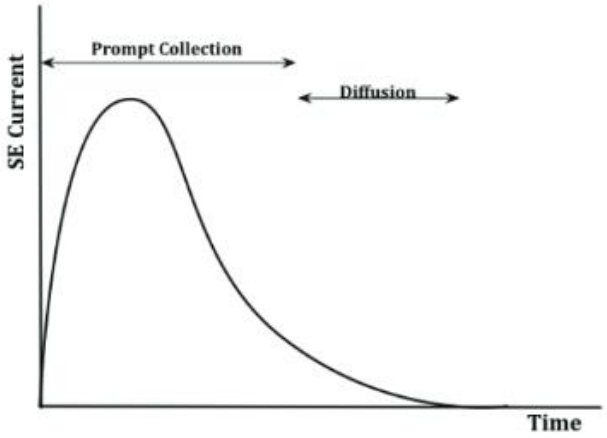


Figure 23 The induced SET current pulse developed in [18]

There are different current source models developed by researchers, such as Freeman's current model [21], Hu's current model [22], diffusion current model [23] ..etc. However, the current model used in this work is based on the recent work done in [24]. An ideal SET voltage pulse is generated using double sinusoidal transition. The voltage waveform follows the following equation:

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

$V(t)$ is the voltage value at time t. $t, t_0, t_1, t_2, t_3, and\ \omega$ are the controlling parameters to pulse waveform. A is the SET pulse height, while the pulse width is the duration between two points in the pulse where both have voltage value $V(t) = A/2$. The double sinusoidal voltage waveform is illustrated in a VerilogA module developed in [25]. It is used to apply SET-induced pulse, as shown in Figure 24.

The voltage waveform does not imitate the one produced by the ionizing particle as the double exponential current model. However, it follows the same waveform after the induced charge perturbation traverses a logic gate [24].

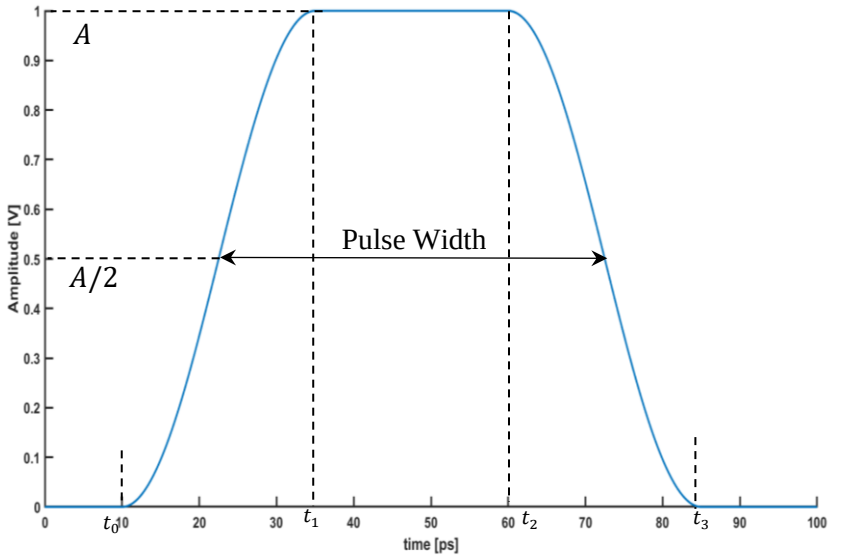


Figure 24 Voltage waveform using double sinusoidal current source pulse [24].

Chapter 2

Standard Cell Characterization

2.1 Introduction.

This chapter illustrates how standard cells are characterized in terms of SET pulsed currents. Each standard cell needs to be stimulated electrically to determine if the induced SET current due to the energetic particle strike at the standard cell inputs will traverse through the gate and appear at the gate's output or not. If the SET pulse did not appear at the gate's output, it means that the gate is not affected by the hit, and the gate stops the SET pulse from propagation.

These simulations are performed at the circuit level, and the SET current models explained in section 1.5 will be utilized to model the induced current at the standard cell's input [25]. A testbench is constructed for each standard cell in the library, as shown in Figure 25. The characterization requires that each standard cell be subjected to multiple SET pulses with different pulse height A *or* V_{in} and pulse widths Tw_{in}. The standard cell performance is recorded after each transient simulation conducted on the cell under test. Instead of storing the whole circuit's output waveform, only values of the propagated pulse height V_{out} and pulse width Tw_{out} are stored. If the input SET current pulse does not traverse throught the tested cell (gate), the V_{out} and Tw_{out} are equal to zero.

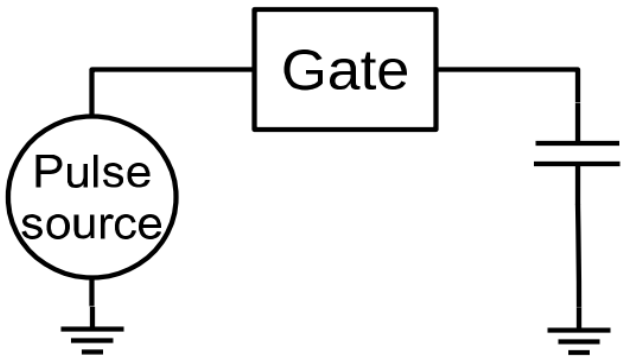


Figure 25 A standard cell (gate) characterization test bench.

The input SET pulses are performed using a sweep over a range of V_{in} and Tw_{in} in SPICE simulations. The Cadence Spectre circuit simulator is used to perform the required sweeps since each sweep is an independent transition simulation. The corresponding output waveforms are stored then analyzed by measuring the SET output characteristic of V_{out} and Tw_{out} as in Figure 25. After multiple sweeps, a set of input SET current pulse and their corresponding propagated SET characteristics are collected.

2.2 Applied Fault Model

In [24], a fault model for the SET output pulse propagating through a standard cell is proposed after applying numerous transient simulations using a test bench circuit similar to the one in Figure 25. The fault model consists of two transfer functions for both $V_{out}\ and\ Tw_{out}$ based on the input SET characteristics V_{in} and Tw_{in}. The transfer functions of the output voltage permutation is given by the next analytical equations:

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

σ_V is a sigmoid surface function, and $k, and\ V_o$ controls the shape of the sigmoid surface and are dependant on the input SET pulse width Tw_{in} characteristic as shown in the following equations:

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

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

The V_{DC} coefficient is the voltage value where the input voltage equals the output voltage in the function. The rest parameters c, T, t_{d1}, and α fitting parameters are used to fit σ_V to the data generated by multiple transient simulations.

The output pulse width Tw_{out} equation is given by:

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

Where $a\ and\ b$ are given by:

$$a = a_0 + a_1.V_{in}$$

$$b = b_0 + b_1.V_{in}$$

The a_0 , a_1 , b_0 , b_1 are technology-dependent parameters. A plot of both $V_{out}$and Tw_{out} functions for an inverter build using 65nm CMOS technology, as shown in Figure 26.

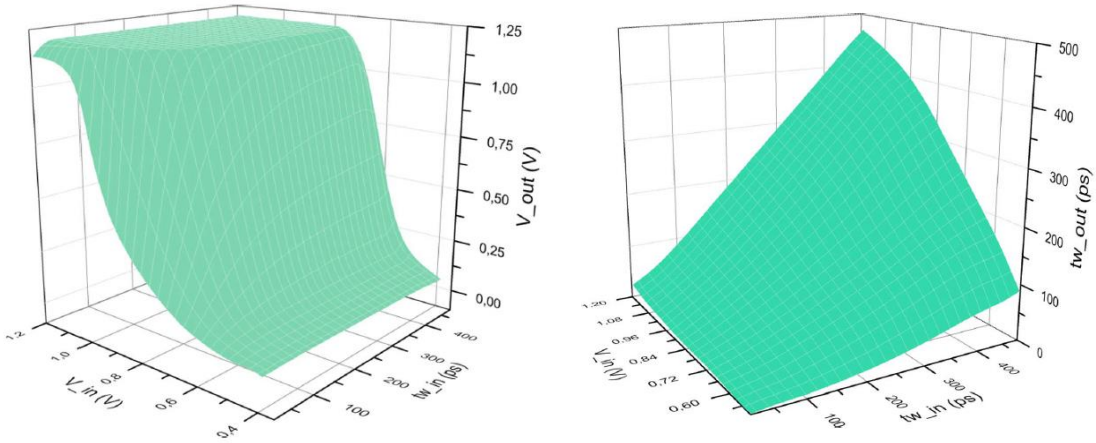


Figure 26 SET output pulse transfer functions of 65nm CMOS inverter.

However, in [24], a different model fitting is applied on the collected SET characteristics by Cadence simulations to produce a SET fault model for each standard cell. The model fitting presented in [25] produced accurate results than the model developed in [24] for 180nm CMOS technology. The new fitting model generates two interpolated equations of the SET output pulse using MATLAB algorithm; hence, more accurate results than the analytical transfer function presented in [24]. The proposed fitting is based on cubic interpolation to provide a smooth interpolation for both transfer functions. A comparison of the simulated date from Cadence transient simulation, the analytical model in [24], and the cubic interpolation developed in [25] is illustrated in Figure 27.

The characterization described above relies on performing many transient simulations of different input SET characteristics of height and pulse width. Increasing the number of input sweep values enhances the fitting functions to find an optimal fault model. The characterization step requires computational memory resources; however, it is a one-time step for the whole technology that can be implemented using a generic

automation script, as shown in [25]. The fault model would not change as long as the physical properties of the standard cell remain constant.

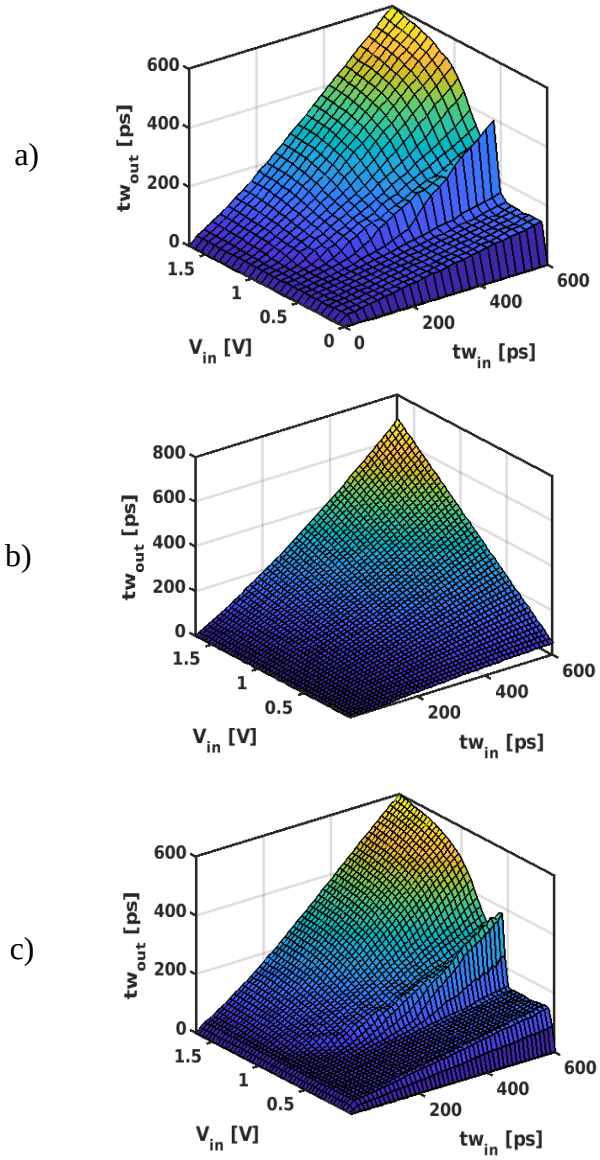


Figure 27 The transfer function of the SET output pulse width of the NO2HDSVTX4 cell in 180nm CMOS. (a) simulation result, (b) model from [22] and (c) cubic

2.3 Characterization Review.

In [24] and [25], they used the minimum standard cell as a loading instance at the output of the gate under test. The capacitance at the tested gate output affects the SET output characteristics V_{out} and Tw_{out} because increasing the load capacitance leads to attenuation of the output SET pulse; hence, lower measured values of V_{out} and Tw_{out}.

In the SET sensitivity work explained in Chapter 4, the load capacitance will be minimum representing the worst-case scenario. Furthermore, the height of the transient pulse from 0→1 and from 1→ 0 is assumed to be equivalent while performing characterization. Therefore, they applied only one type of transient pulse which from 0→1 through all the work in [24] and [25].

However, choosing only the smallest inverter as the typical loading capacitive element to perform standard cell characterization is very pessimistic because cells in ASIC chips can be followed by larger loading capacitive cells most of the time. Larger capacitive load results in less SET V_{out}. The graphs in Figure 28 show different transfer functions plotting after the characterization of the NA2HDLLX0 standard cell when the loading inverter width changes from INHDLLX0 to INHDLLX2.

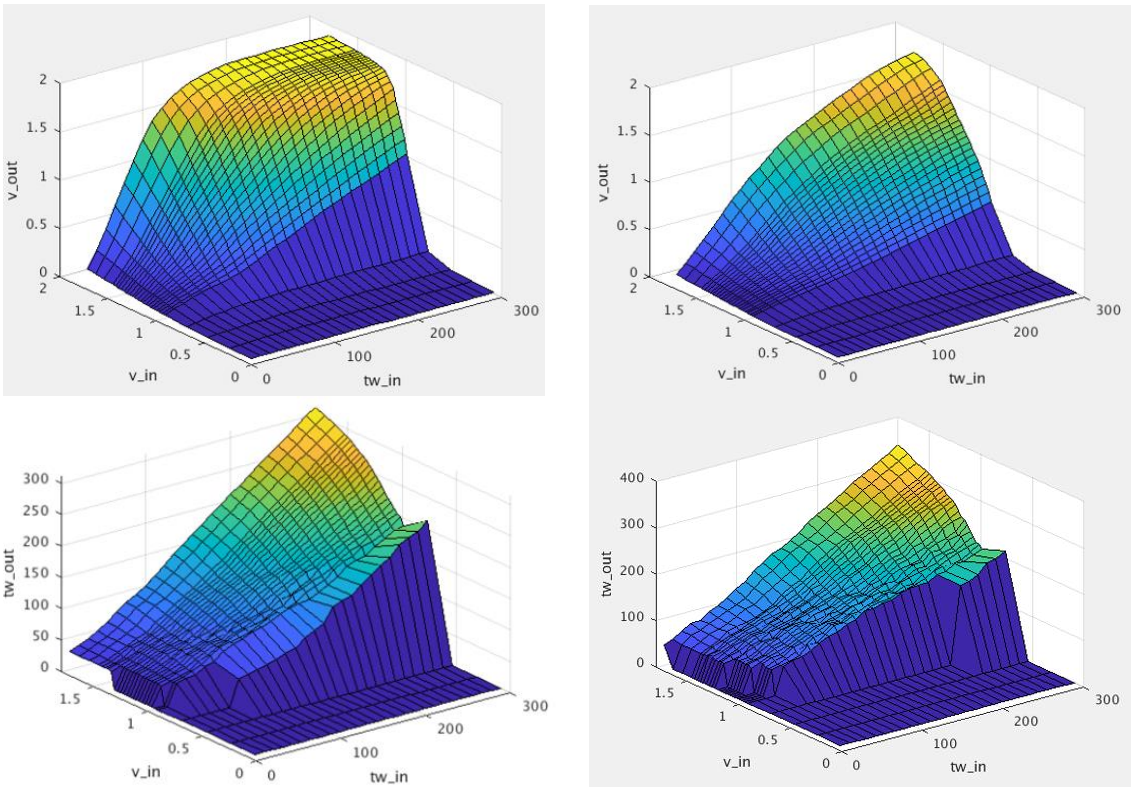
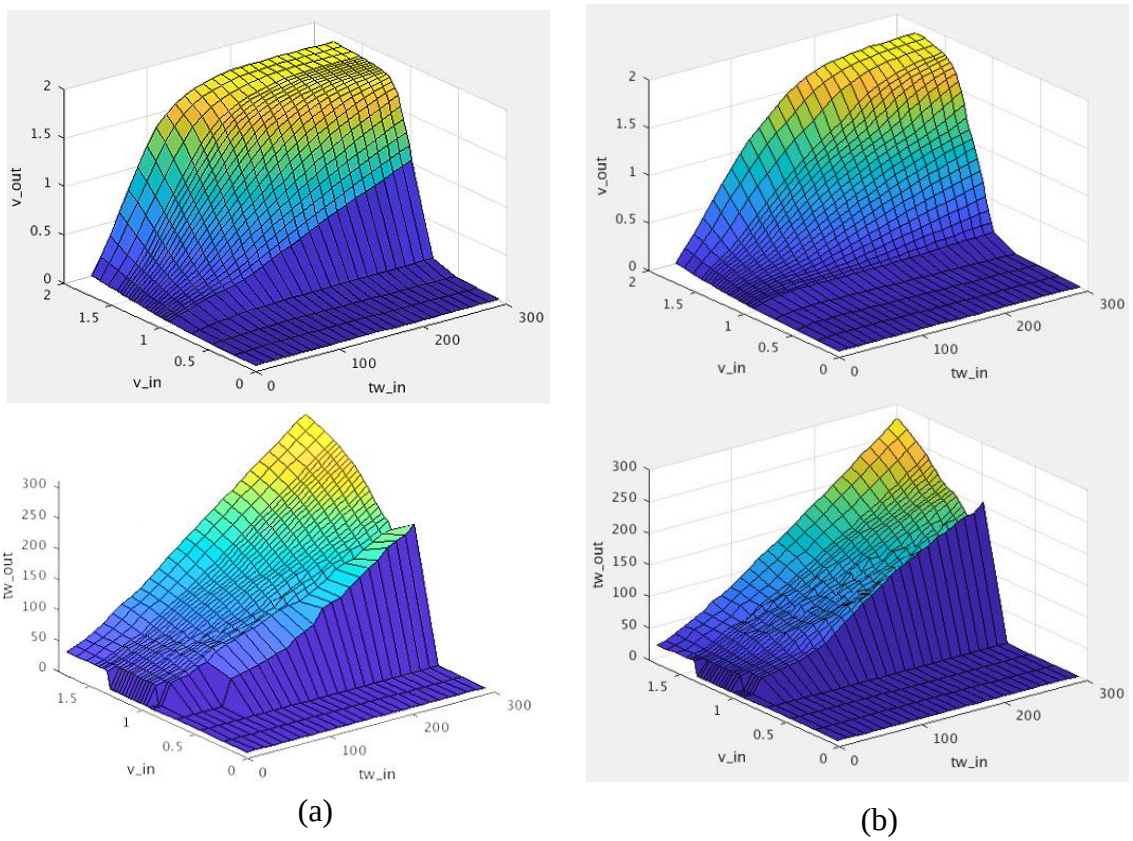


Figure 28 A comparison between using different loading inverters while characterizing NA2HDLLX0 cell.

Furthermore, another experiment was performed to test the assumption that the A voltage height difference is the same for transitions from 0 → 1 and 1 → 0. The similar loading inverter INHDLLX0 characterizes the Same NA2HDLLX0 standard cell, but the logic input to the gates is different. In the first case of 0 → 1 input to NAND gate, the pin A, where SET current source is connected, has a logic value "0," and pin B has logic value 1. In contrast, inputting 1 → 0 to NAND gate, the pin A, where SET pulse occurs, is connected initially to logic "1", and B is connected to logic 1.

The graphs in Figure 29 show different plots of the SET output characteristics after the characterization of the NA2HDLLX0 standard cell for both transitions. In Figure 30, the difference is a plot between:

$$V_{out}(0 \rightarrow 1) - V_{out}(1 \rightarrow 0) \;\&\; Tw_{out}(0 \rightarrow 1) - Tw_{out}(1 \rightarrow 0)$$



(a) (b)

Figure 29 The SET characteristics after characterization of NA2HDDLLX0 when transition (a) from 0 to 1 (b) from 1 to 0

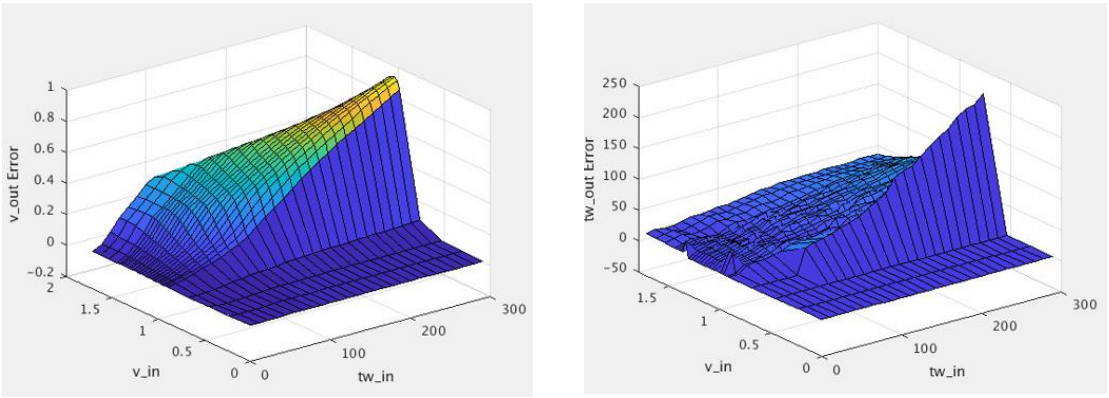


Figure 30 Error between SET from 0 to 1 and SET from 1 to 0 in NA2HDLLX0 with load INVHDLLX0

2.4 Discussion.

Characterization of SET output characteristics of each standard cell is affected by load capacitance and input logic. Characterization of standard cells using the smallest inverter as a capacitive load is the most pessimistic case because a small capacitance after the gate output requires a small charge to fill the capacitor. That is why the output voltage value increases rapidly. Characterization is required to consider different load capacitors to have less pessimistic results. The pessimistic results will influence the SET sensitivity measurements. Assuming all transitions are the same influence also influences the SET sensitivity measurements performed later in section 4.3.

Characterization of each standard cell for different SET values, capacitive loads, and logic input values is a computationally complex operation. It will require time and significant memory and computation resources to characterize each gate. However, these operations can be performed to generate a 3D matrix for each possible input value to construct a fault model using the known liberty format. It is a massive advantage if two lookup tables are constructed to model SET_rise and SET_fall characteristics. The input to each lookup table is ($SET\ V_{in}$, $SET\ Tw_{in}$, C_{load}) and the results are ($SET\ V_{out}$, $SET\ Tw_{out}$). Furthermore, using the liberty format for SET is compatible with today's Static timing analysis.

Chapter 3

Types of VLSI Faults

3.1 Introduction

The current trend in the electronics industry is to enhance the performance of electronic chips by scaling them down. Decreasing the transistor size causes the transistors to operate faster and enables designers to put more transistors in the chip; hence, improving the performance of complex operations by achieving higher frequencies while consuming less power. However, there are many challenges to scaling down transistors due to process variations and manufacturing defects. These defects cause unplanned scenarios in the fabricated chips not accounted for during design. The process variations and manufacturing defects are defined as Deep Sub Micron (DSM) effects. These effects are increasing and becoming more complex with every smaller technology. They could be due to ion migration, crystal imperfections, contact degradation, dielectric breakdown ..etc. [26]. These unpleasant effects cause severe cross-coupling capacitances and inductances among circuit interconnects, substrate noise, thermal noise, dynamic and static voltage drops, and electromigration. These defects affect the functionality and the performance of the circuit and can cause chips to fail [27].

Therefore, researchers tend to build fault models to predict the effects of these defects while designing the chip and protecting the chips from them. The fault is considered as an abstraction of the investigated failure not intended for the design. The fault can be built at any circuit level, such as transistor or gate level. The objective of the fault model is to model a large percentage of the defects at a higher abstraction value, reducing the complexity and number of operations required to investigate each defect. For example, a circuit consists of interconnected gates, and when an interconnect or a gate is not functioning correctly, a fault happens [28]. The Fault model is the essential step in generating test vectors. There are many types of faults, such as stuck-at faults, delay faults, Redundant Faults, Initialization faults ..etc. The two of most concerns regarding SET are the stuck-at-faults and the transition fault, which is a type of delay fault [29].

3.2 Stuck-At Faults

The most popular fault model is the stuck-at fault. The stuck-at fault is a functional fault applied to a single Boolean function (standard cell gate) or multiple connected standard cell gates. The single stuck-at fault is the simplest abstract fault, and it has two types: stuck-at-0 and stuck-at-1. The effect of the fault is the same as if the faulty net is connected directly to VDD or ground. However, it is not a physical defect model meaning that the net in the fabricated chip is shorted to VDD or ground. If the logic is fixed at 1 or "high," it means that a logical error happens when the net or the node exhibits logic value 0 or "low," and vise versa.

The single stuck-at-fault considers only one faulty node in the circuit; hence, the number of faults in the circuit will be 2n, assuming the number of nets is n. The fault is considered permanent since the node will consistently exhibit the fault logic value, such as stuck-at-1 will always have logic 1. The fault can be at any net in the circuit.

In Figure 31, a single-stuck-at fault is assumed at node G. G is the output of AND gate, while A & F are the inputs to it. To test if G is stuck-at-0, both A & F must exhibit a logic 1. This scenario is known as fault excitation, which can activate the fault stuck-at-0 at node G through the application of logic 1 at the same net [30]. The fault needs to appear at the output of the combinational circuit and propagates through the following gates to an output port. In the same example in Figure 31, there is only one OR gate between node G and the output Y. The OR gate must be transparent to the logic value of G, so the other input node to it should exhibit a logic 0. Therefore, if G is stuck-at-0, Y will be 0, and if it has logic value 1 it will be 1. The Inputs at A, B, and C are responsible for activating the fault and propagating it to the output node; thus, the logic values 100 used are considered a test vector.

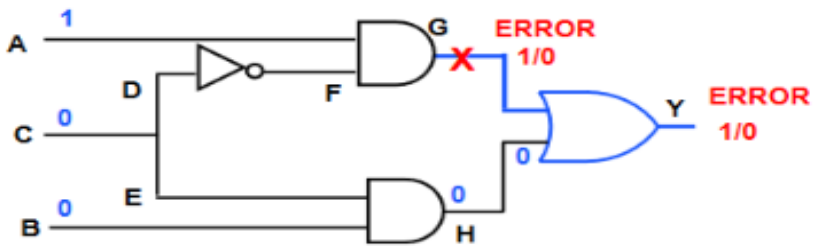


Figure 31 Test vector 100 is responsible for detecting a stuck-at-0 fault at node G [21].

3.3 Delay Faults

The typical stuck-at-fault assumes that a net that is already stuck-at-0 consistently exhibits a 0 logic value and not 1. Therefore, these defects will remain permanent and tend to have this stuck-at-0 fault for an infinite time; thus, they will have an infinite time delay. Therefore, circuits exhibiting stuck-as faults suffer from an infinite delay. Unfortunately, some manufacturing defects do not change the functionality of the circuit gates. Still, they introduce an extra delay to the nodes, causing the logic transition from low-to-high or high to low to take more time. An example of manufacturing defects is undesired electrical connections between two or more nets in the integrated circuit as a result of extra conducting materials or missing insulating materials. This scenario is well-known as bridging [31] [32]. These manufacturing defects exist because of the following scenarios:

1. Under-design power grids are causing large IR drops.
2. Interconnects are close, so capacitive and inductive couplings happen [27].
3. Extreme statistical variation in geometry.
4. Significant gate threshold variations.

Bridging can also happen after fabricating the integrated circuit because of oxide-surface conduction, lateral charge spreading, and electromigration. In Figure 32, there are two examples of bridging within the integrated circuits affecting logic transition. The circuit on the left shows a resistive path from Vdd to an output net. The Circuit shows slow-to-

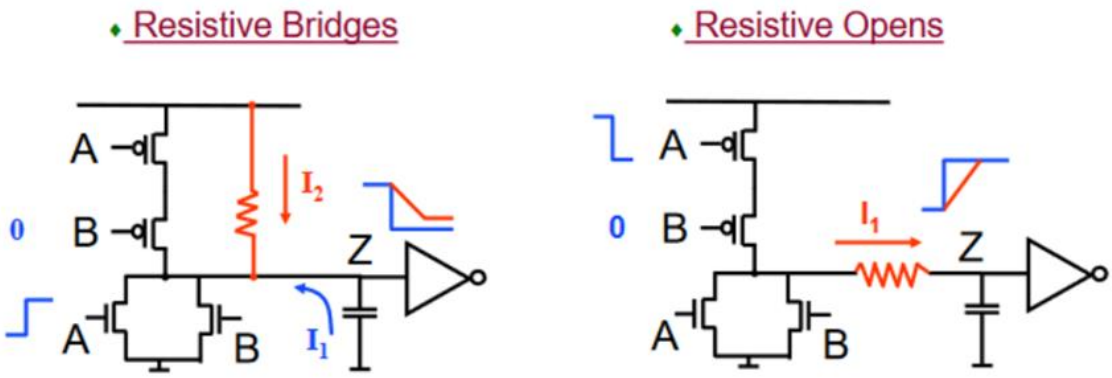


Figure 32 The NOR gate output has a resistive path causing a slow to fall fault [24].

fall faults at the output of the NOR gate when the logic on A changes from 0 to 1. While the circuit on the right exhibits a large resistive path between the output of the NOR gate and the input of the inverter, causing both 0 to 1 and 1 to 0 transitions to be delayed [33]. The introduced extra delay of all possible faults should be less than the clock period so that the circuit could function correctly.

The typical stuck-at faults are not capable of detecting such delay faults in the circuits since they can only detect faults that have an infinite delay to change the logic values from 0 to 1 or 1 to 0, as described early in section 3.2. Defects such as resistive power supply lines, process variations, and coupling faults cannot be detected by the stuck-at-faults because the logic values of the faulty nets are not fixed at a specific logic value. Defects that cause the wrong timing behavior of the circuit are modeled using delay fault models. The two most common delay models are transition fault and path delay fault. A typical test applicable for delay faults must have a sequence of two test vectors T_1 & T_2. The first test vector would initialize the faulty node to the opposite value, and the circuit is allowed to hold its states till it is stable. Then, the second test vector is applied, causing the faulty node to change its value from 0 to 1 in case of a slow to-rise fault. After the specified clock period, a measurement of the logic value is applied at the output ports or latches.

3.3.1 Transition Fault Model

The transition fault model assumes that the extra delay caused by a transition on a net is significant so that the delay of every timing path passing through this net exceeds the clock period [34]. In Figure 33, an example of a slow-to-rise fault at node C is under investigation. In the beginning, an initialization vector "001" is applied at the input ports a, b and d, respectively, before time t_1 and the "c" node exhibits a 0 logic value. At time t_2, a test vector "101" is applied to cause the transition from 0 to 1 to propagate from node "a" through node "c" till it reaches the output "e". If the circuit is fault-free, the logic value 1 appeared on input port "a" will appear quickly at output port "e" at time t_3, as illustrated by the solid line in Figure 33. However, the fault assumes that a considerable delay occurs at node "C", so the logic value will not change from 0 to 1 after this delay period. Therefore, the significant delay period will affect the logic

transition at node “e” so that when a logic measurement would take place at the time t_3 “e” would exhibit a logic value 0 or low instead of 1, as illustrated by the dashed line.

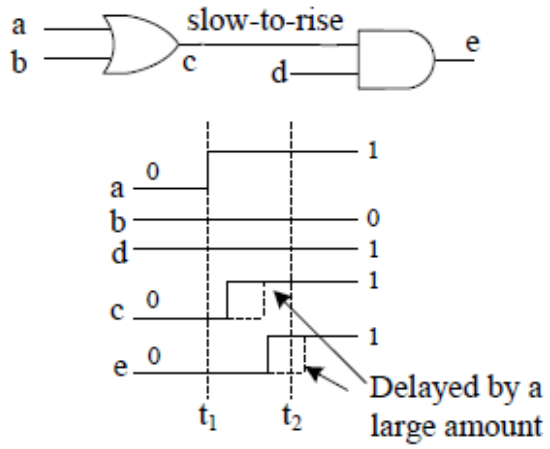


Figure 33 A transition fault example [25].

The delay fault is a function of the logic of the input pins. Each net can have two delay faults slow-to-rise and slow-to-fall faults. The count of the possible delay faults is $2n$, assuming that n is the number of nets in the integrated circuit.

The methodology of generating test vector pairs for transition faults is straightforward. In case of slow-to-rise faults (0 to 1), an initialization test vector is required to put a 0 logic value at the node under test. Then a stuck-at-0 test vector for the node under test is applied afterward. While the slow-to-fall fault (1 to 0) is the initial test vector to put a logic value 1 to the node under test. Then, a stuck-at-1 test vector at the node under test is applied.

The transition fault has many advantages since it detects many defects produced by cross-coupling and bridging. It also uses the stuck-at-fault test vectors to its benefit, meaning it is very friendly with a majority of CAD tools. On the other hand, transition faults miss some small delay defects. These delay defects are modeled using the path delay fault model described in the next section.

3.3.2 The Path Delay Fault

A path is a series of connected gates starting from a primary input and ending with a primary output. The path delay fault model starts to collect the small delays from the start of the path till the end of the path. A path delay fault model happens when the cumulative path delay is larger than the clock period and causes a faulty timing behavior. Therefore, the path delay fault is different than the transition delay fault.

An example of the path delay fault is illustrated in Figure 34. An initialization vector 0010 is applied to the input ports “a”, “b”, “d”, and “f”, respectively. Then, a vector 1010 is then applied at the time t_2. Pin “a” is the only pin to exhibit a logic transition from 0 to 1. According to the circuit functionality, the logic transition should appear in the path [a – c – e – g]. If the circuit is fault-free, the transition should propagate quickly and before the measurements conducted at the time t_3, as depicted in the solid line. However, if a path delay fault exists, the transition delay after each consecutive gate adds to the existing delay of the circuit, causing a timing violation at the time t_2, as indicated by the dashed line.

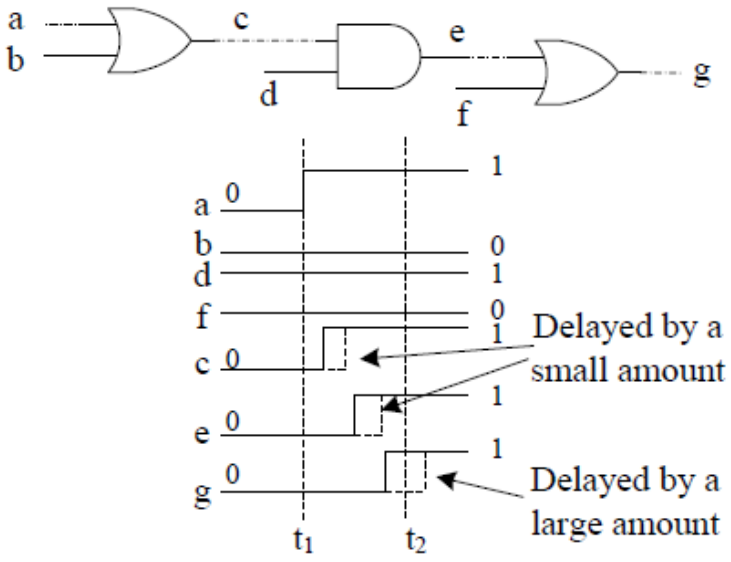


Figure 34 An example of path delay fault [25].

3.4 SET fault

The SET fault at a circuit node exhibits a logic transition from 0 to1 and back to 0 again and vice versa. The induced SET pulse will propagate through the circuit gates till it reaches a primary output. An induced current pulse propagates from the node under test to a primary output through the circuit gates when a SET fault happens. For the SET pulse to be present at the output, an input test vector should also sensitize the propagation path. Such a test vector is a stuck-at-fault test vector. If a stuck-at test vector is applied, the SET pulse has a path to propagate and appear at primary outputs. However, The SET happens for a certain period; a stuck-at test vector can still be manifested to propagate a transient fault and monitor the SET fault at a primary output.

A real-time simulation is required to measure the SET pulse characteristic. In order to validate the results generated by FastScan algorithms, a Spectre simulation test was performed on one of the smallest benchmarks available online (ISCAS85 C17) [35]. The circuit was synthesized first using Genus, a Cadence synthesis tool. Then, Cadence Virtuoso read the netlist and generated the schematic shown in the following figure. The input and output nodes are recognized as circuit ports

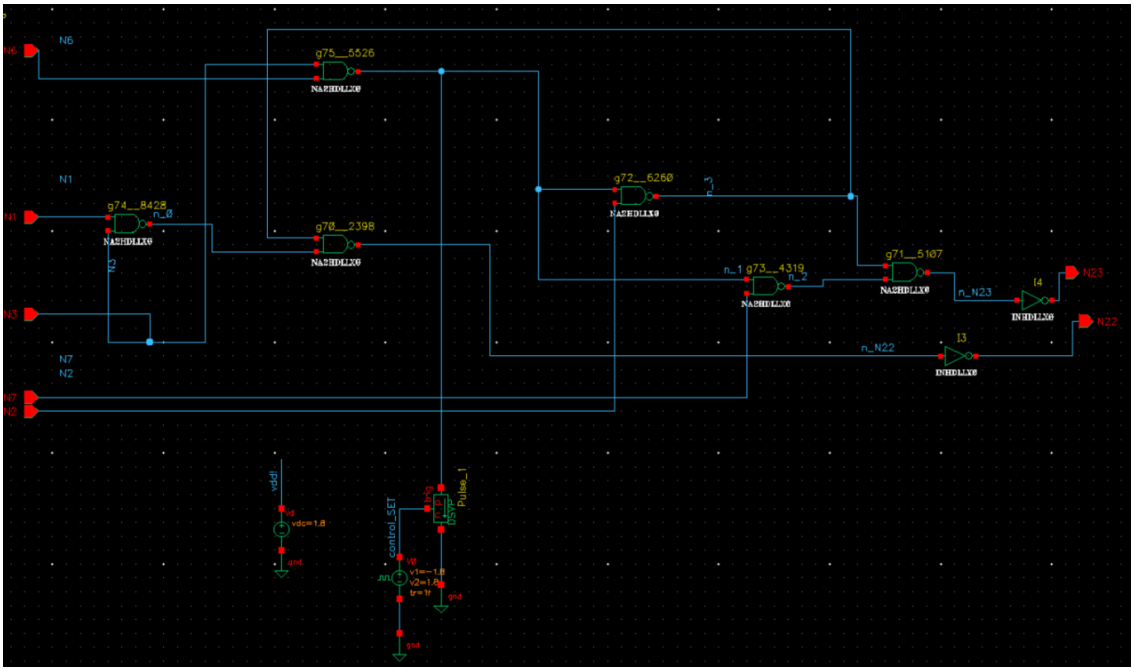


Figure 35 Building testbench to the C17 benchmark netlist on Cadence Virtuoso.

The conditions to have a SET fault are a large induced SET current pulse and input vectors that can sensitize at least one path to the output. In Figure 35, the circuit diagram to test the C17 benchmark against the SET fault is presented. In this test, a SET current source named DSVP is used to insert a SET pulse with a height of 1.8 volts and width of 180 picoseconds at net “n_1”. A stuck-at 0 test vector for net n_1 {10011} is applied at the primary input ports. A transient simulation is performed on the circuits. Then, the voltage waveform of the nets is plotted as in Figure 36. The induced SET pulse at net n_1 affects the circuit performance. Net “n_2” exhibits a logic transition from 1 to 0 and back to 1 again after the SET pulse duration ends. Also, the output net “n_N23” has a logic transition from 0 to 1 and back to 0 again. The waveform at “n_N23” is not correct for almost 140 picoseconds. If a logic measurement is conducted during this period, it is clear that a faulty timing violation would occur.

Figure 36 The voltage waveform of C17 circuit when a SET fault at net n_1.

The SET fault does not occur for an infinite time like stuck-at faults, so it is closer to its behavior to the transition fault. However, to test SET faults, the initialization vector is not required since the root cause of the SET fault is the energetic particle that induces SET current pulse in the circuit and not a typical primary input transition as in the transition delay. The required test vector is the deduced stuck-at vector because it can sensitize a path and make the SET observable at a primary output.

Chapter 4

SET Sensitivity Flow

4.1 Introduction

A SET sensitivity flow is presented in this section, starting by generating possible SET propagating paths from nodes under test to primary outputs. Then, starting analyzing the produced paths with graph theory and measure SET sensitivity of each node in the circuit, producing SET sensitivity report besides identifying immune nodes. The whole flow is shown in Figure 37.

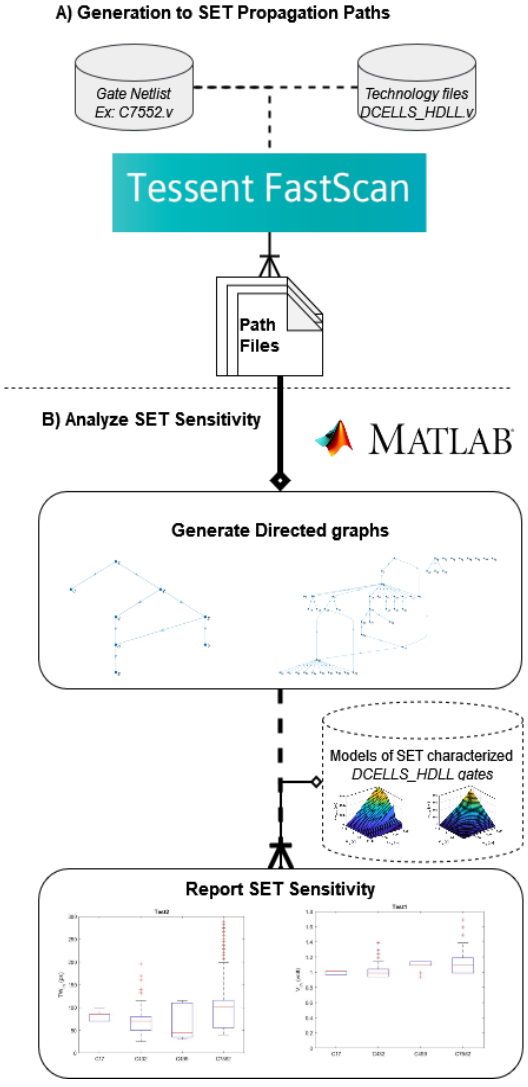


Figure 37 The SET sensitivity flow.

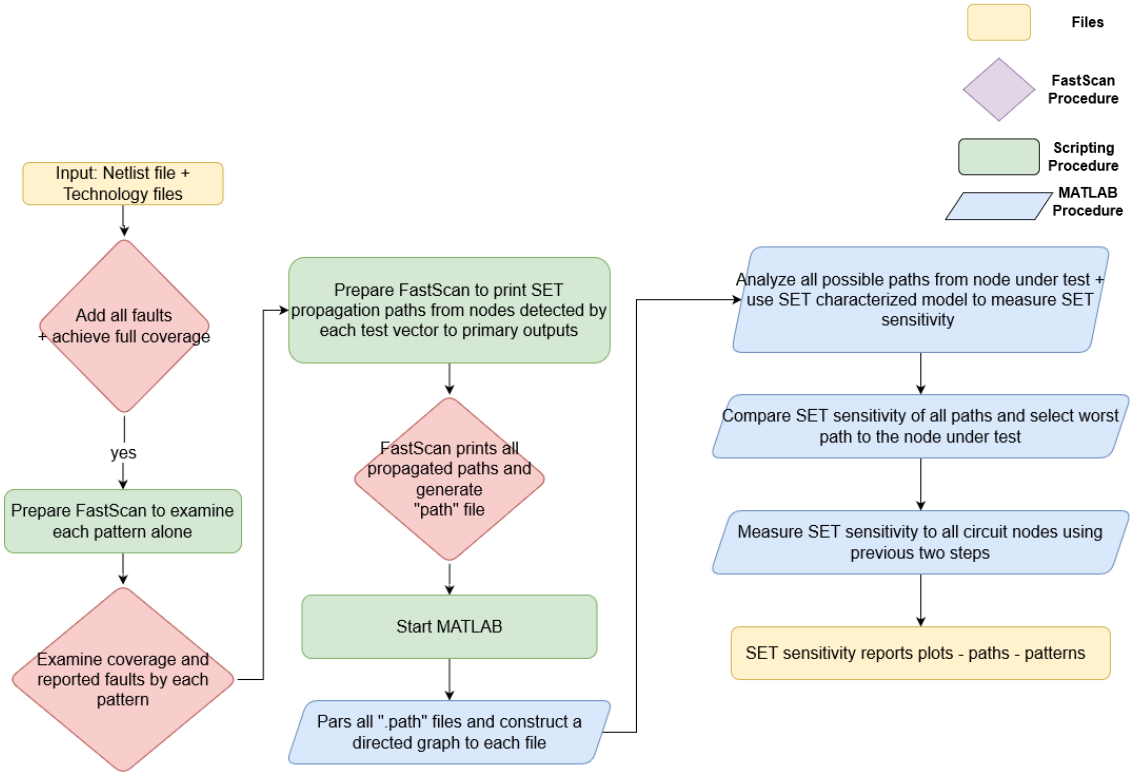


Figure 38 Steps and procedures applied in SET sensitivity flow.

In Figure 38, a helicopter view of the SET sensitivity measurement flow is presented. The flow starts by running FastScan to generate test vectors to achieve full coverage of SET faults for all circuit nodes. Then, FastScan is utilized to analyze the effect of each test vector after performing some scripting procedures. Furthermore, FastScan generates possible sensitized paths from nodes under test to at least one primary output node.

MATLAB starts to analyze FastScan generated paths. A directed graph is constructed for each path file. MATLAB algorithm extracts all possible propagating paths from the node under test to the primary output, using graph theory techniques. Then, the MATLAB algorithm measures the SET sensitivity of the node under test. After analyzing all circuit nodes, the MATLAB algorithm generates SET sensitivity reports for the whole circuit. The whole flow is explained in detail in this chapter.

4.2 FastScan Flow.

4.2.1 Introduction.

Tessent FastScan, a Siemens digital industries software, is an automatic test pattern generation (ATPG) responsible for analyzing different types of circuit fault models in integrated circuits such as stuck-at faults, delay faults ...etc. It is also capable of analyzing the circuit under test using user-preferred input test patterns. It can also produce input patterns to detect as many faults as possible and achieve full coverage. The most significant advantage of Tessent FastScan is automating the testing flow and its ability to be customized based on the user preferred flow [36].

There are two possibilities for SET transitions to occur at any circuit node. First, if a node is at 1, “high” state and SET event occurs, changing the node voltage to 0, “low” state. Second, if the node is at 0, “low” state, and SET transforms it to 1, “high” state. A logic simulator is required to suggest testing vectors to help circuit designers to observe such faults at primary outputs. FastScan can be employed to generate test patterns to study circuits in terms of possible SET faults because the nature of the SET fault is somehow very similar to the transition faults. Still, we are interested in the second stuck-at generated vector, as described earlier in section 3.4. In this work, FastScan operates on synthesized Verilog netlists. Therefore, FastScan requires both the circuit under test and necessary technology files illustrating the equivalent Boolean functions of the synthesized gates beside the name of the top module to identify the primary I/Os.

FastScan analyzes the logic of the input circuits starting from the top module. It is also asked to detect all possible transition faults (SET faults) for all circuit nodes. The tool starts to generate a two-input vector pair that would detect the faults at all nodes because all nodes were added to the possible fault sites in the circuit. The physical meaning is that when an energetic particle hits one of the nodes, given that the suitable test pattern is applied, an induced current pulse will propagate through the circuit to at least a primary output port. The induced transient pulse can change the logic at the output port for a sufficient period, causing a single event upset. This work considers only a single hit or fault during a specified period.

Finally, the FastScan generates a text file describing the primary input and output ports. It also writes the logic values for the input vector pairs used and the expected logic values for the primary outputs. The following figures show the type of reports generated by FastScan, reporting fault coverage, test vectors, and detected nets.

In this step, it is assumed that an energetic particle could hit the node under test and change the voltage value for a sufficient time so that one can observe the change at a primary output. Also, since most of the analysis is made to combinational circuits, it is possible to ignore latching masks. When there are sequential gates, latching masks should be considered.

```
ASCII_PATTERN_FILE_VERSION = 3;

SETUP =

    declare input bus "PI" = "/N1", "/N2", "/N3", "/N6", "/N7";

    declare output bus "PO" = "/N22", "/N23";

end;

SCAN_TEST =

    pattern = 0  clock_sequential ;
    force   "PI" "10111" 0;
    force   "PI" "01101" 1;
    measure "PO" "11" 2;

    pattern = 1  clock_sequential ;
    force   "PI" "01001" 0;
    force   "PI" "10000" 1;
    measure "PO" "00" 2;

    pattern = 2  clock_sequential ;
    force   "PI" "01110" 0;
    force   "PI" "11010" 1;
    measure "PO" "11" 2;

    pattern = 3  clock_sequential ;
    force   "PI" "11110" 0;
    force   "PI" "10011" 1;
    measure "PO" "01" 2;
```

Figure 39 The head of ”all.pattern” file generated for the C17 benchmark circuit.

```
type    code    pin_pathname
----    ------  --------------------------------
  1     DS      /g74__8428/Q
  0     EQ      /g74__8428/A
  0     EQ      /g74__8428/B
  0     EQ      /N1
  1     EQ      /g70__2398/B
  1     DS      /N1
  1     EQ      /g74__8428/A
  1     DS      /g72__6260/Q
  0     EQ      /g72__6260/B
  0     EQ      /g72__6260/A
  0     EQ      /N2
  1     DS      /N2
  1     EQ      /g72__6260/B
  0     DS      /N3
  1     DS      /N3
  1     DS      /g75__5526/Q
  0     EQ      /g75__5526/B
  0     EQ      /g75__5526/A
  0     EQ      /N6
  1     DS      /N6
  1     EQ      /g75__5526/B
  1     DS      /g73__4319/Q
  0     EQ      /g73__4319/B
  0     EQ      /g73__4319/A
  0     EQ      /N7
  1     EQ      /g71__5107/B
  1     DS      /N7
  1     EQ      /g73__4319/B
  1     DS      /N22
```

Figure 40 The generated “all.fault” report, illustrating the detection status of each net.

```
           Statistics Report
            Stuck-at Faults
---------------------------------------
Fault Classes              #faults
                           (total)
---------------------  ---------------
  FU (full)                    50
  -------------------  ---------------
  DS (det_simulation)     50 (100.00%)
---------------------------------------
Coverage
  -------------------
  test_coverage               100.00%
  fault_coverage              100.00%
  atpg_effectiveness          100.00%
---------------------------------------
#test_patterns                    5
#simulated_patterns               6
CPU_time (secs)                 0.6
---------------------------------------
```

Figure 41 The "all.stats" report generated by FastScan.

4.2.2 FastScan Flow

It is essential to get the conditions at which a SET could propagate from the faulty node to at least one primary output without possible logical masking in the combinational circuit under investigation to start the sensitivity flow. The role of FastScan is to generate test vectors that can show the SET faults at the outputs. The whole flow of FastScan, as illustrated in the flow chart in Figure 42.

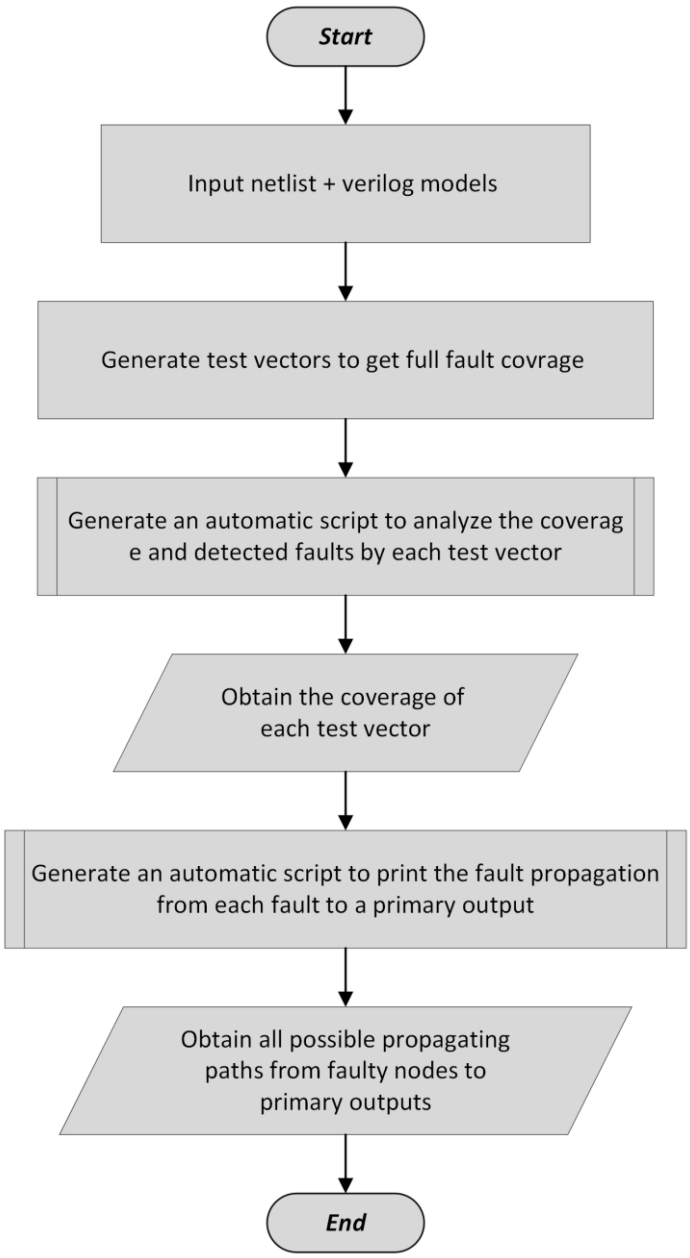


Figure 42 A flow chart of the FastScan flow

4.2.2.1 Initial FastScan Run.

The first run of FastScan takes all nodes as possible faults nodes. The software will generate a set of test vectors trying to reach maximum coverage. A statistic report is also generated indicating the overall coverage of the whole set of the generated test patterns. FastScan generates another file, summarizing the status of each net in the circuit and whether it is detected or not.

4.2.2.2 Second FastScan Run.

The first run of FastScan only gives information about the collective effect of the whole generated test vectors. Furthermore, information about the sensitized paths during each test vector is not achieved. Therefore, the Second run of FastScan is essential to get more information about the individual effect of each test vector generated in the first run. However, the second run of FastScan requires some modifications to add each pattern individually to it. Therefore, an automation step is applied using a Perl script to build a custom second run on FastScan. The Perl script role is to:

I. Parse all previous patterns and generate a pattern file specified for each test vector.
II. Build a custom FasScan (.do) file that will run FastScan (n) times, where (n) is the number of the generated test vectors from the initial run. Each time, there will be one test vector applied to the netlist.

This approach will help analyze the effects of each test vector in terms of its coverage and which nodes the test vector detects. Another statistic file and fault file will be generated after applying each test vector. The statistic file measures the coverage achieved by the applied test vector. This information is crucial to the MATLAB approach while comparing test vectors to identify the worst-case test vector. The fault file records the faulty detected nodes and the output ports where the SET fault should appear. Still, the second run does not directly give the sensitized paths information required because FastScan does not have a direct command to type the sensitized path directly. Unfortunately, the FastScan needs another run after indicating the detected nodes and asking explicitly the FastScan to identify the logic path from the node under test to the output.

4.2.2.3 Third FastScan Run

Before running FastScan for the third time, additional information is required to be added in the FastScan (.do) file. Using FastScan command [37]

[report_gates -path <fault node> <detected_output_port>]

An automation Perl script is required to add this information explicitly in FastScan (.do) file. The Perl script tasks are:

I. Open the fault file generated by each test vector and extract the faulty detected nodes and the detected primary output.

II. Write an automated FastScan script to apply first the test vector and then type the report_gates command from each fault node to the observable primary output port.

III. Save the reported sensitized paths in a .path file with a unique convection such as [pat<pattern_numer>_from_<fault_node>_to_<primary_output>.path]

The third run of FastScan is the final one because the FastScan now will produce the sensitized path required to measure the SET sensitivity. Unfortunately, this way produces a lot of similar paths, and sometimes there are empty paths. The empty paths because the Perl script types command asking FastScan to report all possible paths from the detected nodes to a primary output at which the transition fault can be observed. That is why some filtration processing is applied afterward.

A vital notation is that if there is no possible test pattern detecting a SET fault at a particular node, this node is undetectable at the primary output. This node has logical masking preventing the SET fault from observing at the primary output. The previous scenario shows a trade-off between hardening the design against SET faults versus the tendency to increase the observability of the combinational circuit.

4.2.3 FastScan Flow Results Description.

Different benchmarks netlists are investigated in this section, starting from the simplest netlist C17 to the largest netlist C7552. C432 benchmark is a channel interrupt controller. C499 is a 32-Bit single error correction (SEC) circuit. C7552 is a 32-bit adder/comparator, arithmetic logic unit, responsible for different 16 logic functions [38]. Using FastScan, SET fault coverage of 100% was achieved after synthesizing the ISCAS85 netlists with XFAB 180nm HDLL technology library using Genus synthesize

tool. The number of nets is extracted from the synthesis tool (Genus), while the number of faults is twice the number of standard cell pins, as illustrated in Figure 40.

Table 1 FastScan results of different ISCAS85 benchmarks.

#	*Source benchmark*	*Test Vectors*	*Number Of nets*	*Total Faults*	*Detected Faults*	*Coverage percentage*	*Number of path files*
1	C17	5	11	50	50	100%	88
2	C432	57	152	810	810	100%	9302
3	C499	93	215	1206	1203	99.75%	65462
4	C7552	141	1197	6562	6560	99.97%	808138

4.2.4 Validating FastScan results using C17

This section conducts a validation experiment on the C17 circuit, shown previously in Figure 35. The experiment applies a large enough SET characteristic pulse with voltage height equal to 1.8 volts and pulse width equal to 300 picoseconds at all nodes in the C17 while applying the test vectors generated by FastScan. All waveforms show a logic transition at the corresponding detected primary output port due to the propagated SET fault. Since all waveforms experience a logic transition from low to high or from high to low in the graph experiment validates FastScan results using real-time simulation.

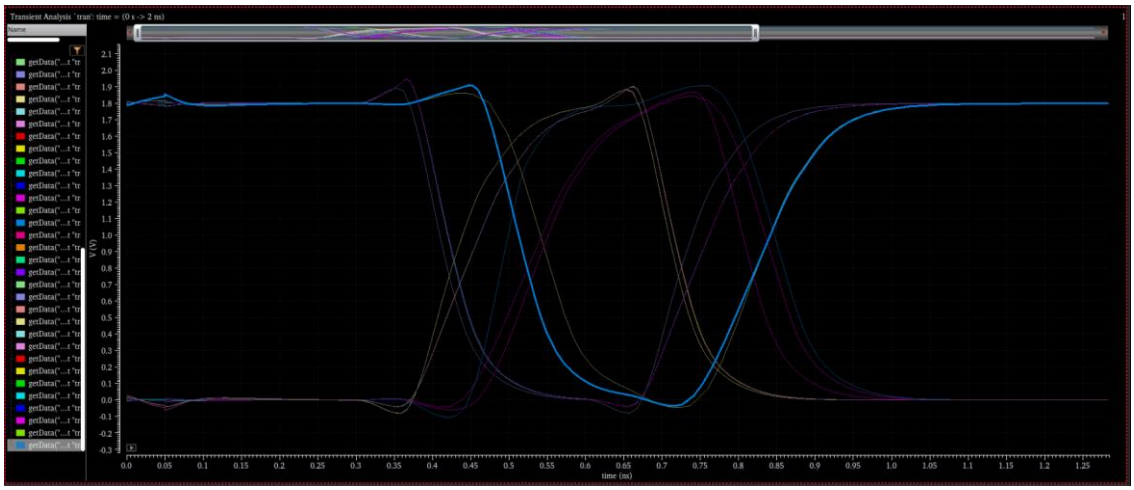


Figure 43 All voltage waveforms measured at output primary ports after applying a large SET pulse on each node under test.

4.3 MATLAB sensitivity measurement flow

The MATLAB flow objective is to measure and store the SET sensitivity of all nets related to the combinational circuit under investigation. The final analysis result is a table full of the minimum SET characteristics for each circuit net that can result in a SET propagation to a primary output and produce a single event upset. Before the beginning of the analysis, there are multiple information or files required:

I. Knowledge of different possible sensitized paths from the node under test to at least one primary output.
II. Cell map to identify the reference standard cells of each instance in the netlist.
III. SET characterized models for the standard cells mentioned in the combinational circuit.
IV. The user provides the primary input and output ports.

The first requirement of a set of sensitized paths is produced from FastScan using the flow described early. The FastScan reports all gates between the node under test and the primary output specified early in the command. The FastScan does not care if a logic transition happens while propagating through each gate. Therefore, the MATLAB algorithm does this extra analysis while reading the path file because it is crucial to get a propagating path.

The second requirement is performed while performing synthesis of the nets under test. The synthesis tool is asked to build a map file reporting the standard cell reference for each instance in the generated gate netlist.

The third requirement is performed early by exploring all or necessary standard cells used in synthesis and characterizing them in terms of the SET fault. This requirement does not depend on any preferred characterized models as any standard cell models can be implemented in the algorithm. However, the upcoming work uses the characterized model proposed by [25] since it records less error percentage with real-time simulations performed on cadence virtuoso. The fourth requirement is I/O ports knowledge; the algorithm requires knowing the input and output ports while parsing the path files to help build a directed graph, as explained in the following sections.

Another essential premise is that the load instance used for characterization is the smallest instance load available in the technology. For the XFAB HDLL library, the smallest cell to be used as a load is INVHDLLX0, an inverter with width x0. This premise has a massive impact on results because the smallest cell has a small load capacitance, so the SET characteristic measured at the cell's output will be high compared to other cells in the technology. The sensitivity measurement will be on a pessimistic case scenario because of the use of the smallest inverter. For results to be less pessimistic, fault modeling of the library standard cells should be modeled with other different cell loads in the technology, which requires massive computational resources to finish such characterization in a reasonable time. However, the sensitivity flow is generic, and if such models exist, results will match real-time simulations.

The MATLAB flow starts with parsing the cell reference map of the gate netlist. Also, all generated path files from FastScan are parsed and used to construct a direct graph. Each directed graph is analyzed to identify the possible SET propagating path from the faulty node to the primary output specified in the file. The cell map is used to identify the reference standard cells of each cell gate. Characterization of each path is performed using the input SET model. Then, the flow identifies the worst propagating path in terms of SET sensitivity. The following sections explain the performed operations in the MATLAB flow and their significance. Furthermore, a flowchart summarizing the steps applied in MATLAB flow is presented in Figure 44.

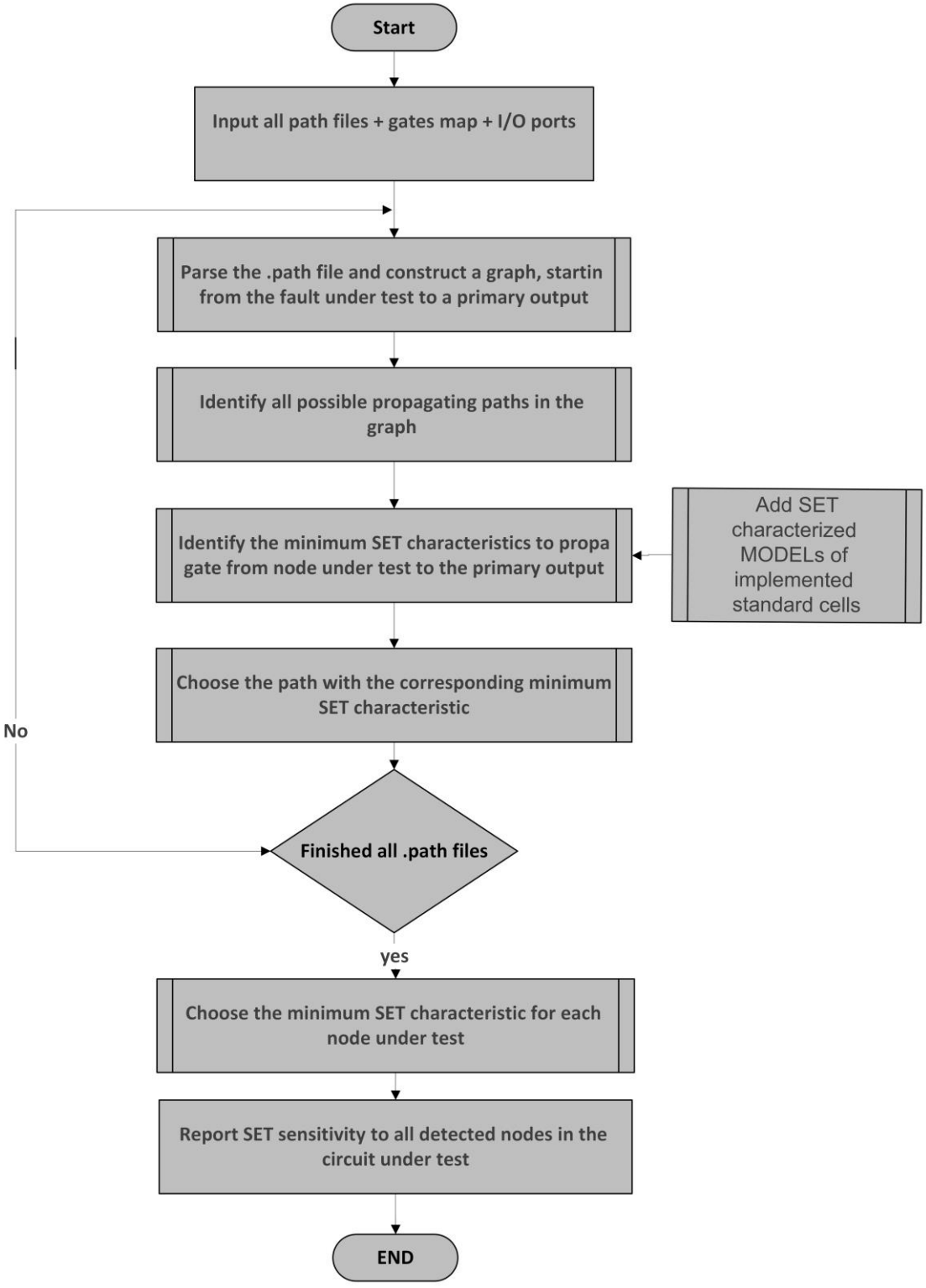


Figure 44 A Summary of the implemented steps in the MATLAB sensitivity flow.

4.3.1 Analyze “path” Files.

The MATLAB algorithm uses the generated path files from FastScan to identify the propagation of the logic values through the combinational gates until it reaches a primary output. The algorithm starts with parsing all generated path files from FastScan. Each file contains all possible logic transitions from the node under test to a primary output. In large circuits, there are many possible paths for the fault to propagate from the node under test to the primary output, given this applied pattern. It is also possible that this node under test has other path files because its fault is observable at other primary outputs, using the same applied test pattern. Also, some patterns could share the observation of the same node so that there are sometimes other paths that could be a replica of old path files or they are new ones. In brief, there are multiple possible paths for the faulty node to be present at least one primary output. The best-case approach to cover all cases would be an exhaustive test to cover all possible patterns and analyze each sensitized path. However, this approach is redundant besides being time and computationally consuming. The optimal approach is to analyze the available propagating paths from a generated set of test vectors and identify the worst propagating path among them and save its sensitivity.

The MATLAB flow could parse paths produced from synthesized netlists and non-synthesized netlists written in the format of ISCAS85 benchmark circuits. Two different writing styles were analyzed; hence two different parsers are implemented. The flow starts collecting all cell names exhibiting a logic transition at their inputs or outputs. Since the transition type is essential, the MATLAB code also records the transition values either started from 0 or 1. FastScan sometimes reports unnecessary cells that do not exhibit a logic transition but share a circuit node with other propagating cells. Therefore, the code is implemented to filter out those unnecessary cells and records only gates with the logic transition.

A simple example of a path file is in Figure 45. The propagating path starts from the input primary port and propagates through three gates. The MATLAB follows the propagating signal from the primary input and numerates the gates through which logic transition occurs either from (000-111) or from (111-000). MATLAB enumerates the I/O ports and the cell gates during parsing, as indicated in table1.

```
//  ------------------------------------------------------------------------
//  Begin path trace between /N3 (3) and /N23 (13).
//  ------------------------------------------------------------------------
//  /N3  primary_input
//      N3      O  (000-111)  /g75__5526/A  /g74__8428/B
//  /g75__5526  NA2HDLLX0
//      A       I  (000-111)  /N3
//      B       I  (000-111)  /N6
//      Q       O  (111-000)  /g73__4319/A  /g72__6260/A
//  /g73__4319  NA2HDLLX0
//      A       I  (111-000)  /g75__5526/Q
//      B       I  (111-111)  /N7
//      Q       O  (000-111)  /g71__5107/B
//  /g72__6260  NA2HDLLX0
//      A       I  (111-000)  /g75__5526/Q
//      B       I  (111-111)  /N2
//      Q       O  (000-111)  /g71__5107/A  /g70__2398/A
//  /g71__5107  NA2HDLLX0
//      A       I  (000-111)  /g72__6260/Q
//      B       I  (000-111)  /g73__4319/Q
//      Q       O  (111-000)  /N23
//  /N23  primary_output
//      N23     I  (111-000)  /g71__5107/Q
//  Number gates in trace = 6.
```

Figure 45 A generated FastScan path file from "N3" PI to PO "N23"

Table 2 Node Enumeration of the path cells

NAME	NODE NUMBER
"/N3"	1
"/G75__5526"	2
"/G74__8428"	3
"/G73__4319"	4
"/G72__6260"	5
"/G71__5107"	6
"/G70__2398"	7
"/N23"	8

4.3.2 Extract SET Propagating Paths.

The MATLAB code also records the circuit connections from the node under test to the primary output. These connections are essential while constructing a circuit-directed graph. The objective of the directed graph is to extract all possible paths from the node under test until the primary output. A MATLAB function is implemented to quickly go over the graph and return an array of all possible paths.

There are some gates at which no propagating signals appear, so the gates are left as leaf nodes, while other gates that experience logic propagation are considered parent nodes for the following paths. Each graph should end up with a leaf of a primary output node. The corresponding graph to the path file shown in Figure 45 is presented in Figure 46. Each number presented in the graph represents a cell gate in the path file. The directed graph shows the electrical connections and the hierarchy of the sensitized path.

In this example, there are two paths from the primary input "/N3" to the primary output "/N23". The MATLAB function analyzes the graph and extracts both paths. A directed graph algorithm is applied to extract all possible paths from nodes 1 to 8. In the graph, there are two possible paths to SET propagation. The first one is from [1 – 2 – 4 – 6 – 8], and the second path is [1 – 2 – 5 – 6 – 8]. The SET pulse propagates through three gates in each possible path in Figure 46, excluding the I/O ports.

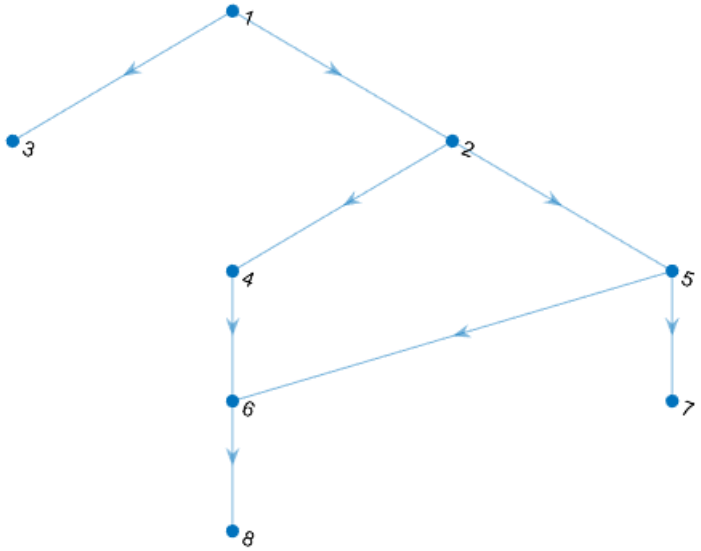


Figure 46 A directed graph to the path file shown in the Figure44

The MATLAB then stores the extracted paths after transforming node numbers to their original names in the gate netlist. With the information given from the cell map file produced from the synthesis step, the cell gates are translated to their original reference standard cell gate. For example, the cell ‘g75__5526’ is translated to NA2HDLLX0 standard cell. This translation is essential to the next step of characterizing the SET's possible propagating paths. Another complex directed graph from C432 from node 1 to node 23 is presented in Figure 47.

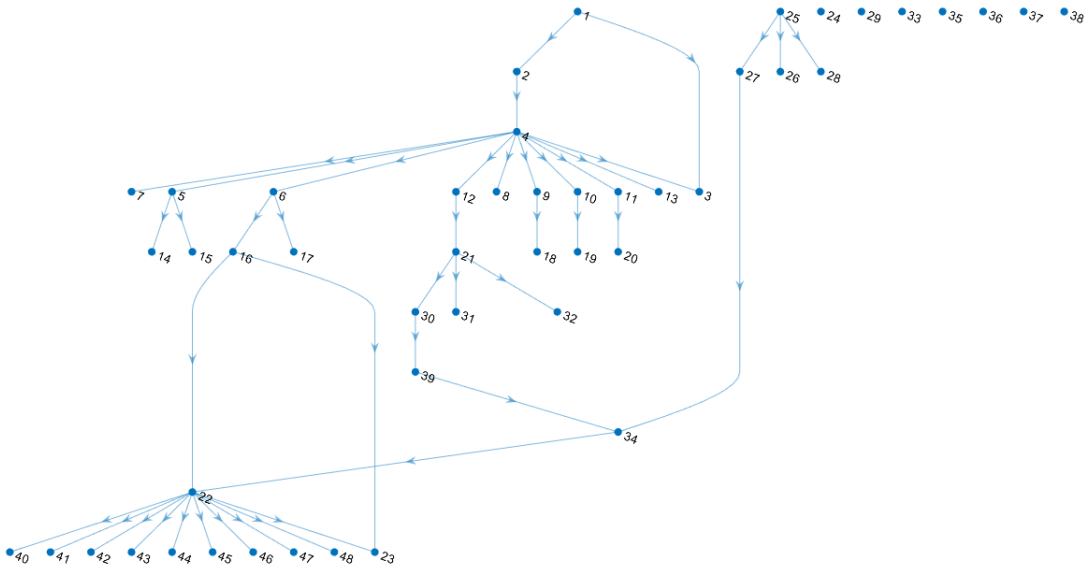


Figure 47 An example of a complex directed graph built for a path in C432

4.3.3 Apply SET Tests

The node under test will likely have more than one propagating path to the specified primary output in the path file. Therefore, it is essential to analyze all these paths and record the worst in terms of the minimum SET characteristics required to make a single event upset at the primary output. Two tests proposed in [31] will be manifested to measure nodes' sensitivity. In [39], there is no mention of the SET characteristics to obtain an SEU at the primary output. In this work, when the SET height at the primary output is larger than VDD/2, the SET input characteristics cause an SEU.

The assumed input SET characteristics are applied to the SET model of the first cell gate. The output of the cell gates is used as input to the SET model of the second cell gate, and their output will be used for the third gate and so on. This operation stops at the

last cell gate in the path. A flow chart of the operation to characterize each path or consecutive cell gates is presented in Figure 48.

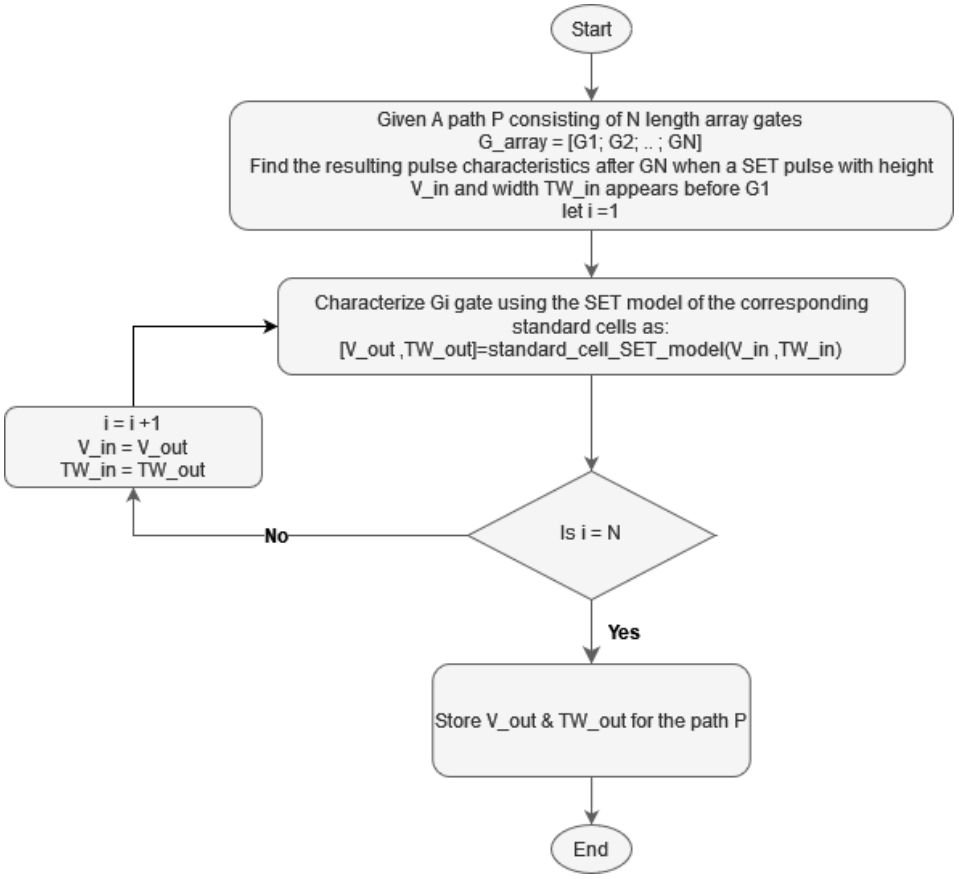


Figure 48 A flow chart identifies the procedure to characterize SET through consecutive gates.

4.3.3.1 SET Test1

The first test applies the maximum pulse width input used while characterizing the library standard cells. In our case, the maximum pulse width used is 300 picoseconds. The SET test searches for the minimum SET height or voltage possible besides the 300 pico width to produce a single event upset (SEU) at the output. Starting from a small voltage, continue increasing the input voltage value until an SEU occurs. The operation of measuring the SET output pulse characteristics is mentioned in Figure 48. In the end, the minimum voltage value V_{min} recorded is to be selected. Regarding this test, the worst propagating path among possible N paths is the corresponding path which records the minimum voltage.

$$P_{worst} = argmin_i \{Vmin_i\};\ \forall\ i\ \in [0, N]$$

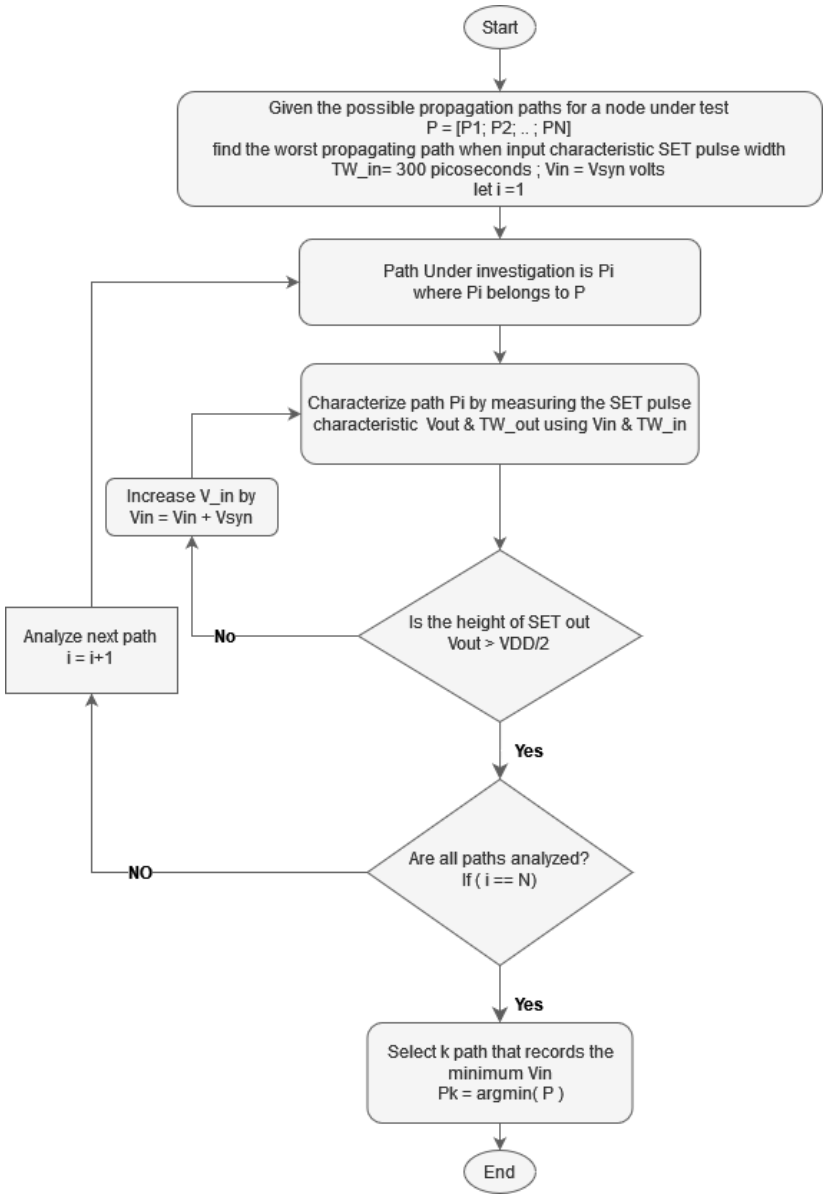


Figure 49 The flow chart of applied SET test 1.

4.3.3.2 SET Test 2

The second test applies VDD, the maximum voltage available in CMOS, used while characterizing the library standard cells. The SET test searches for the minimum SET pulse width using defined models to produce a single event upset (SEU) at the output. Starting from small pulse width, the code continues increasing input SET value until an SEU occurs. In the end, the minimum input pulse width recorded (tw_{in}) is to be selected

as the tw_{min} required to cause an SEU. Regarding this test, the worst propagating path among possible N paths is the corresponding path which records the minimum voltage.

$$P_{worst} = argmin_i \{tw\_in_i\};\ \forall\ i\ \in [0, N]$$

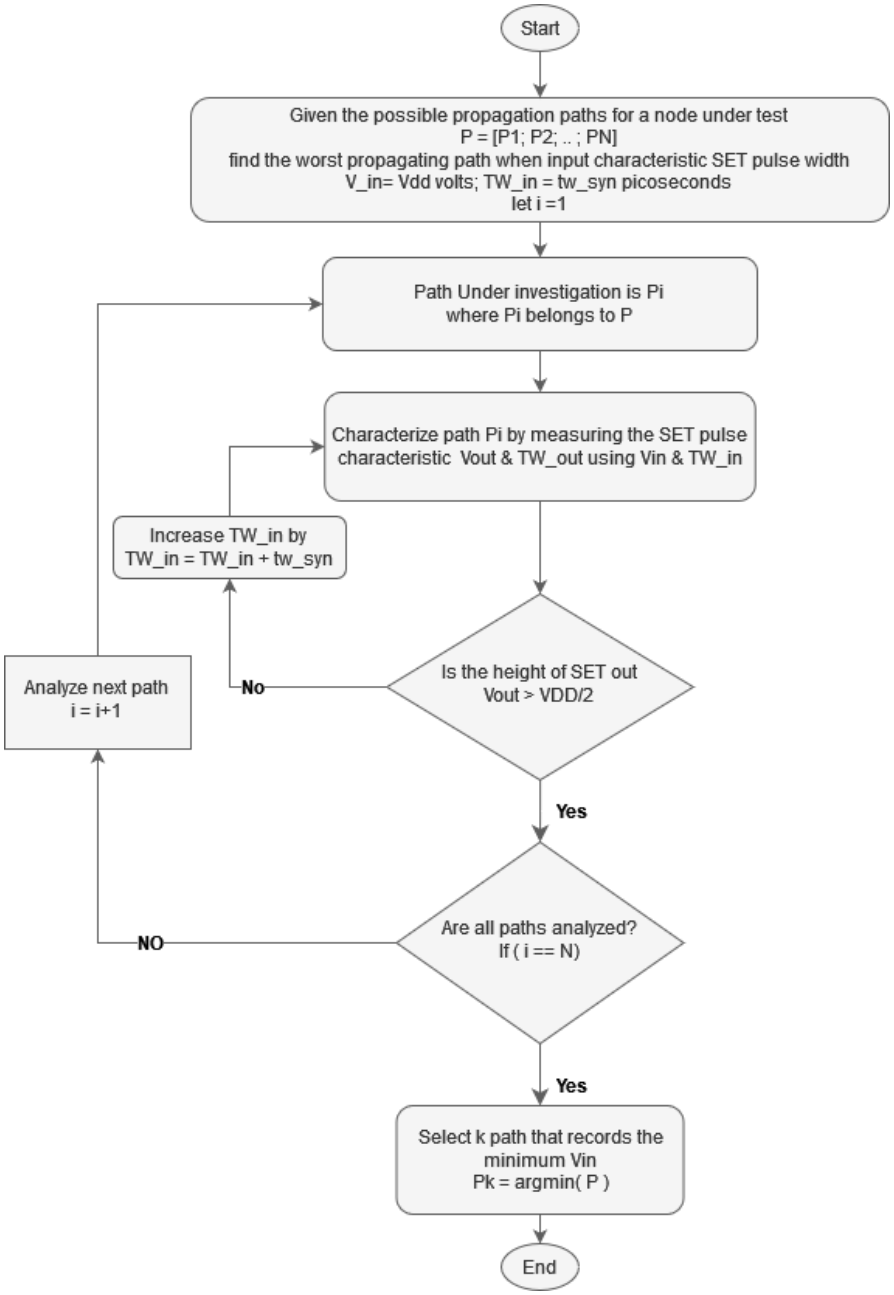


Figure 50 The flow chart of the second SET test

4.3.4 Characterization of Combinational Circuits.

When the nets of the combinational circuit under test are fully tested, a report contains the most sensitive nets, and the least sensitive nets are produced. If nets have different SET propagation paths from different patterns, the worst SET propagation path is to be selected as a sensitive measure to the node. In the end, the MATLAB algorithm starts to analyze the results and record the worst tested SET propagating path for each node. As mentioned in section 4.3, the results are based on pessimistic SET standard cell models. Therefore, real-time simulations should produce SET pulses at the output with less than or equal pulse characteristics measured in MATLAB.

Table *3* summarizes the result of characterizing benchmark circuits in ISCAS 85 circuits when the pulse width is constant. At the same time, a sweep of the SET pulse changes till an SEU appears at the primary output. The nets in each benchmark recorded different sensitivity measures. Some nets are away from the primary output and report lower sensitivity voltage values. The minimum, mean, and maximum sensitivity values are reported with how many nets report each value. Furthermore, Figure 51 shows a boxplot representing the SET sensitivity of test1 for all nets in the ISCAS85 benchmarks.

Table 3 Test1 SET sensitivity results.

#	*Source benchmark*	*Number of nets*	*Sensitivity*	*N.O nets repetitions*	**Test 1** *Min.* V_{in} $(Tw_{in} = 300ps)$
1	C17	11	Most sensitive	3	0.965
			Least Sensitive	6	1.015
2	C432	152	Most sensitive	61	0.94
			Mean sensitive	26	1.04
			Least Sensitive	2	1.39
3	C499	215	Most sensitive	32	0.94
			Middle sensitive	81	1.09
			Least Sensitive	55	1.14
4	C7552	1197	Most sensitive	439	0.99
			Middle sensitive	342	1.09
			Least Sensitive	29	1.89

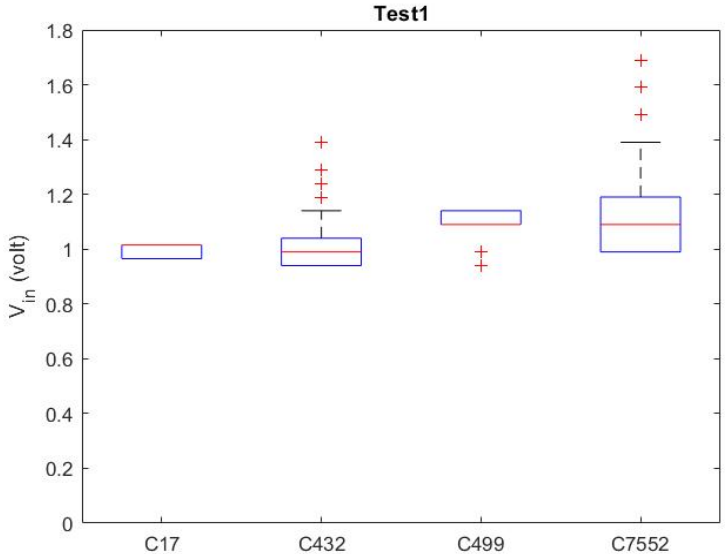


Figure 51 A box plot for the SET sensitivity results from

In Table *4*, the SET sensitivity results for Test2 are presented. The minimum induced SET pulse width required to produce an SEU at the primary output for each node is recorded. Figure 52 shows a boxplot representing the SET sensitivity of test1 for all nets in the ISCAS85 benchmarks.

Table 4 Test2 SET sensitivity results

#	***Source benchmark***	***Number of* nets**	***Sensitivity***	***N.O paths repetitions***	**Test 2 *Min.* Tw_{in} ($V_{in} = 1.8\ V$)**
1	C17	11	Most sensitive	3	70
			Middle sensitive	5	85
			Least sensitive	1	97.5
2	C432	152	Most sensitive	2	25
			Middle sensitive	15	70
			Least sensitive	1	195
3	C499	215	Most sensitive	16	30
			Middle sensitive	15	85
			Least sensitive	27	115
4	C7552	1197	Most sensitive	133	40.2
			Middle sensitive	115	101
			Least sensitive	29	304

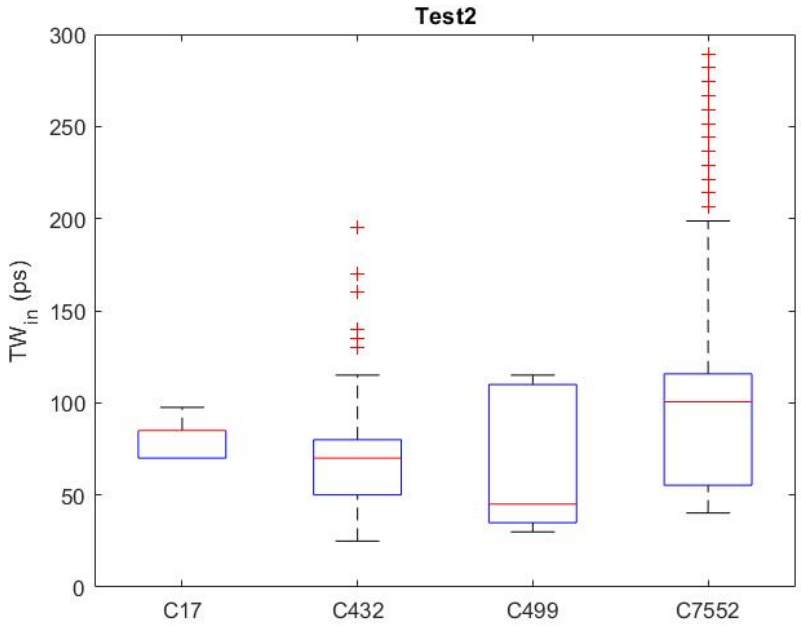


Figure 52 A boxplot of the sensitivity results for Test2

4.4 Discussion

In [19], a graph between the SET transient pulse width measured at the struck node versus the corresponding linear energy transfer of the hit particle was introduced for multiple Bulk CMOS technology nodes. In Figure 53, the 180nm plot is extracted

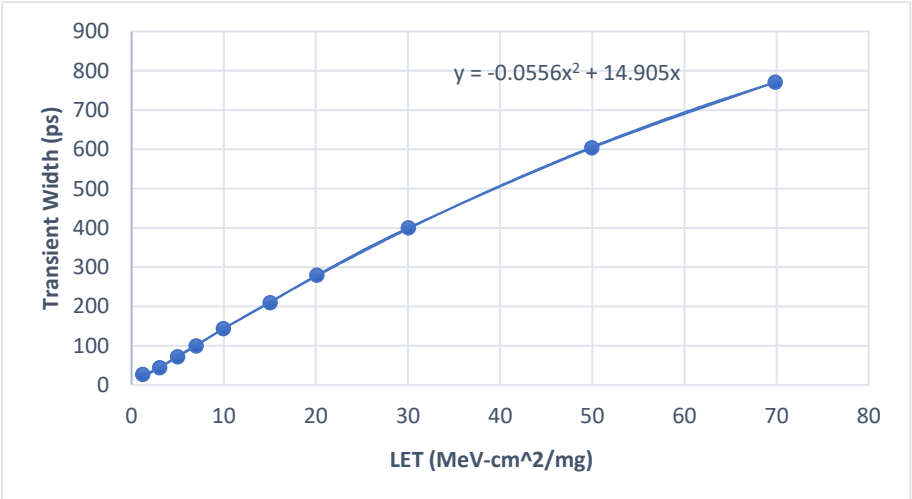


Figure 53 SET pulse width versus LET specific for 180nm Bulk CMOS technology

using the online tool, WebPlotDigitizer [40], for better visibility, while the original graph is presented in Figure 22.

Furthermore, ions used for experimental results at the cyclotron facility at Texas A&M University reported in [16] are presented in Table 5. The corresponding transient pulse width extracted from the plot graph in Figure 53 is also added. The plot represents a golden reference to measure the relation between the struck particle energy with the SET pulse width induced at the struck node.

Table 5 The energy of the experimental ions and their expected pulse width.

Ion	LET ($MeV\frac{cm^2}{mg}$)	Transient Pulse $\boldsymbol{Tw_{in}}$ $(\boldsymbol{ps})$
Ne	1.8	27.009144
Ar	5.7	86.764944
Kr	20.6	330.637416
Xe	40.7	698.734344

The SET pulse width obtained from the MATLAB sensitivity flow in Test2 will be compared against the SET pulse width of the experimental lab ions reported in Table 5. The number of nets immune to SET effects due to these ions is written in Table 6. The Xe ion with a pulse width of almost 700 picoseconds is not used because the window of pulse width for the characterized standard cell was 300 picoseconds.

Table 6 Characterization of benchmark circuits relative to lab ions

#	***Source benchmark***	***Number of* nets**	***Number of nets immune to***		
			Ne	**Ar**	**Kr**
1	C17	11	9	1	0
2	C432	152	142	29	0
3	C499	215	170	59	0
4	C7552	1197	1044	566	29

4.5 Results Validation

A generic MATLAB script is used to perform an exhaustive test on Cadence Virtuoso to validate the MATLAB flow. The simulation inputs are the minimum SET pulse characteristics predicted in Test1 and Test2. All SET faults predicted by the pessimistic MATLAB flow were simulated. SET faults from 0 to 1 and from 1 to 0 appear on the real-time simulation plot windows in Figure 54. In the graph, an exhaustive simulation was performed on all faults predicted in the C17 benchmark using the produced patterns in FastScan.

The majority of the faults predicted in MATLAB do not appear at the primary output because of the pessimistic characterization model since C17 only consists of NA2HDLLX0 cells. The characterization based on the capacitance of the smallest inverter is a very rough condition since there is no accounting for the fanout capacitance or the larger load capacitance for standard cells other than NA2HDLLX0. Transitions that reach more than $0.5 * VDD$ are predicted by MATLAB with their pulse width.

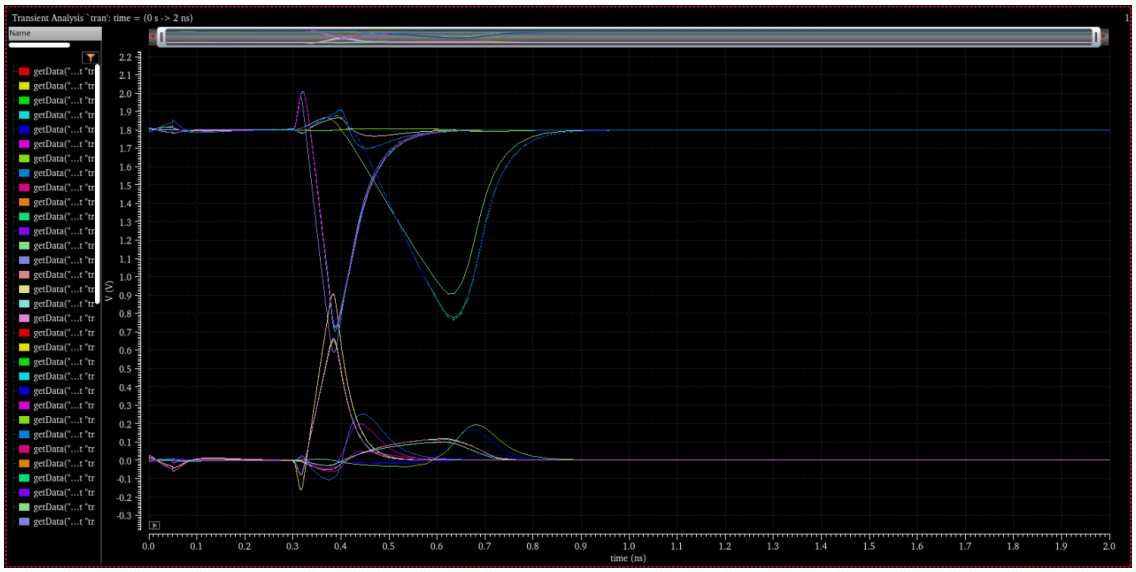


Figure 54 Real-time simulations of C17 benchmark

Chapter 5

Identify Worst-Case Patterns to SET Faults

5.1 Introduction

In [39], Barcelo used two tests to measure the minimum SET sufficient for the SET pulse to propagate its way to an observable output. The first test is sweeping over the voltage values while maintaining the pulse width at a significant value. The second test maintains the voltage constant, usually at VDD, and sweeps over the pulse width. The sweeping stops once there is no more SET pulse at an observable output. The same two tests were performed in the SET sensitivity flow.

Every trial of the cells characterization process aims to calculate the propagated SET height (voltage) and width (period). Measuring the final SET pulse that would appear at the output port is already performed in the MATLAB flow besides measuring the minimum input SET pulse at the fault node to determine its SET sensitivity. Each pattern can detect (n) SET faults out of all existing nodes (N). Furthermore, MATLAB already measures the SET sensitivity of each node using the two tests mentioned in [39]. However, the sweeping starts from a small value until a SET pulse appears at the observable output, and the search for SET propagation paths is not performed exhaustively. The pulse characteristics of the inserted SET pulse at different nodes under test are recorded in addition to the pulse characteristics of the resulting SET pulse at the output [41].

There was no attempt to characterize functional patterns and analyze their SET sensitivity in the literature. Therefore, the objective is to find a methodology that would help characterize all test patterns and define the worst-case pattern. This approach has significant benefits while designing hardened circuits to SET because it will guide circuit designers about possible patterns that will increase the probability of SET faults. It can also propose excellent test patterns to be used while conducting lab SET experiments. The proposed methodology to characterize different circuit patterns and find the worst-case vector pair is presented in the following section.

In [24], Gili presented a SET propagation model for CMOS logic gates. The model consists of two analytical equations to quantify the possible SET output characteristic (width and height) in terms of the input SET characteristics. The output characteristics are plotted against the input characteristics after characterizing a single CMOS 65 nm inverter, as shown in the following figures. Each cell gate can be characterized independently and have its models that describe how the CMOS gate behaves in response to the different input SET waveforms. These different tests are plotted as shown in the following graphs in the case of the CMOS 65nm inverter.

As described before, only two tests were used to measure SET sensitivity in-circuit nodes. Each CMOS gate will only be analyzed in two planes in the 3D plots presented. The two planes are when:

$$V_{in} = V_{DD} \qquad \& \qquad Tw_{in} = T_{max}$$

Where T_{max} is the maximum pulse width used while characterizing cell gates

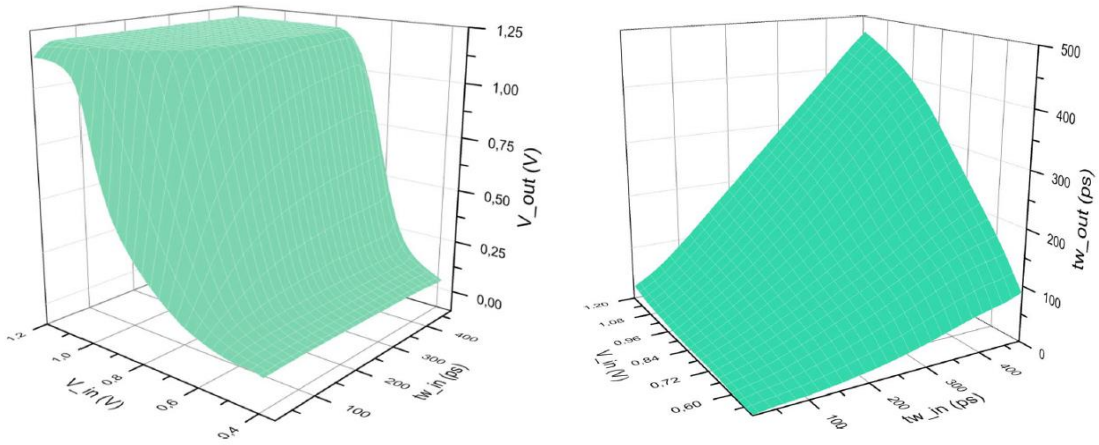


Figure 55 A plot of the SET transfer functions

In the following example, a propagating path consists of N consecutive CMOS gates, as shown in Figure 56. The gates will be analyzed for the two different tests. The following plots describe the plots used for the SET characterized SET for teach gate. The set of planes used in the two tests is shown in Figure 57. The first test starts with $V_{in} = V_{VDD}$ while sweeping over the input pulse width that would result in a SET pulse at the output, plots [a,b] represent the two plans used at the first CMOS buffers, and their output will be used in plots [c,d] and so on until the last CMOS gate, as shown in Figure 57.

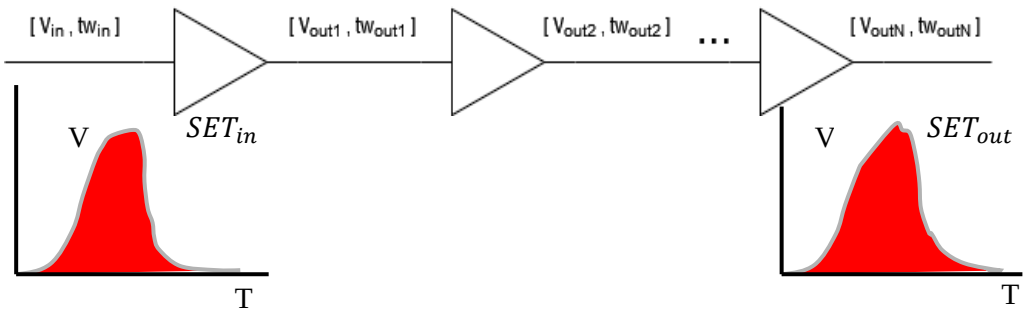


Figure 56 An example of a SET pulse propagating through N buffers

TEST1

$when\ V_{in} = V_{DD},\ tw_{in};$ $V_{in} = V_{out1}, tsw_{in} = tw_{out1};$ $V_{in} = V_{outN-1}, t_{in} = tw_{outN-1}$

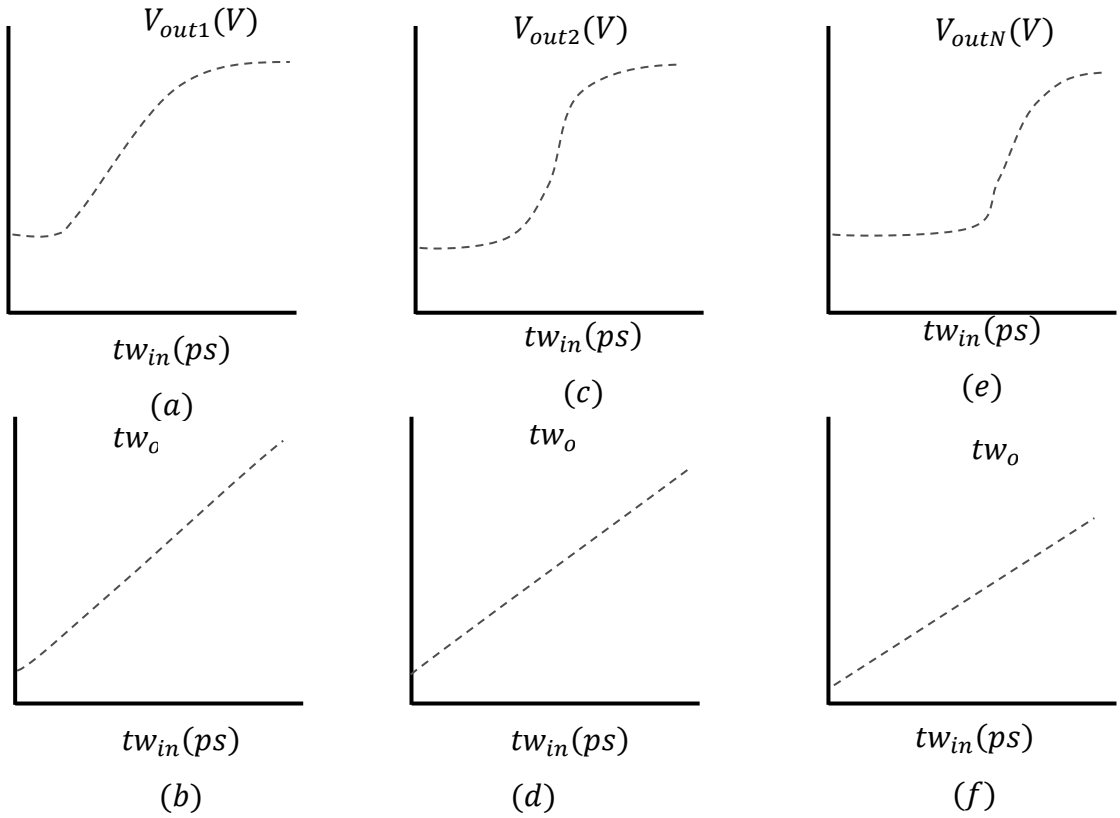


Figure 57 Illustrative plot of Test1 (a & b) are the planes used at 1st gate. (c & d) planes used at 2nd gate. (e & F planes used at N gate)

The Second test starts with $tw_{in} = tw_{max}$ while sweeping over the input pulse height V_{in} would result in a SET pulse at the output. Plots [a,b] represent the two plans used at the first CMOS buffers, and their output will be used in plots [c,d] and so on until

the last CMOS gate, as shown in Figure 58. The illustrative plot shows that SET pulse starts to fade at the second gate and be small at the N^{th} gate; however, if the the voltage at the output of N^{th}gate larger than VDD, the set pulse width causes a single event upset.

TEST2

$when\ tw_{in} = tw_{max}, V_{in}\ ;\ tw_{in} = tw_{out1}, V_{in} = V_{out1}\ ;\quad tw_{in} = tw_{outN-1}, v_{in} = V_{outN-1}$

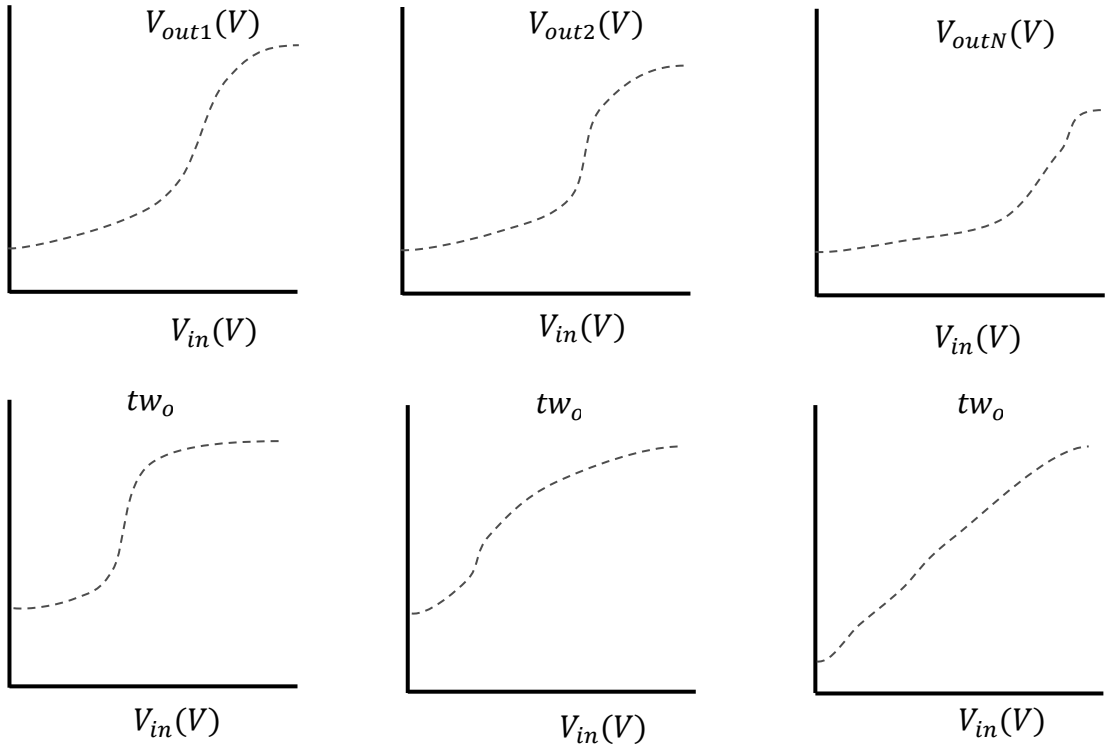


Figure 58 Illustrative plot of Test2 (a & b) are the planes used at 1st gate. (c & d) planes used at 2nd gate. (e & F planes used at N gate)

5.2 Generalization of SET Testing.

Using only two tests to start characterizing CMOS gates restricts investigation to only two planes from the whole 3D space. That is why it is possible to add more plans (tests) until all 3D space is covered. Applying this approach to get the worst-case pattern is very significant.

After analyzing each pattern and identifying the SET sensitivity of nodes under test, it is possible to get all the height and width of minimum output SET to each node

that the investigated pattern can detect. It is essential to measure the area under the V, T curve of the output SET pulse detected from each node under test. The mean of these output SET pulses could be considered a measure of the average SET pulses that occur when the SET pulse strikes while this pattern is applied in the circuit. Also, the minimum input SET characteristic could be employed to calculate the average input SET pulse that can cause propagation of SET pulse to an observable output leading to a single event upset. The ratio is a key distinguishing factor between the average input SET pulse and the output SET pulse. It helps identify how much the SET pulse can increase or decrease after propagating in the circuit.

5.3 Pattern Comparison

Constructing a figure of merit is required to compare different test patterns and identify the worst-case candidate among them. The two factors should be unitless, as mentioned before. They also have to be able to quantify the SET effect in the circuit under test. The first factor is the pattern coverage ratio. It is very intuitive to include pattern coverage because it directly measures how many circuit nodes can be affected when a SET happens while this pattern is applied. Each pattern has a coverage ratio that is always less than or equal to 1. The coverage ratio is measured firstly in the flow using FastScan during the second iteration when each pattern is analyzed alone to know the detected faults given while the pattern is applied. The coverage ratio is:

$$coverage\ ratio = \frac{detected\ nets\ by\ the\ pattern}{all\ nets}$$

The second factor should consider how much change in SET pulse while it propagates through circuit gates until it reaches an observable output. Therefore, one can define Δ as the mean ratio between input and output SET pulse as follows:

$$\text{pulse } \Delta\ ;\ (\Delta) = \frac{mean\ \int SET_{out}}{mean\ \int SET_{in}} = \frac{mean\ [\ \sum_{all\ detected\ nodes}\ (width*height)_{out}\ ]}{mean\ [\ \sum_{all\ detected\ nodes}\ (width*height)_{in}\ ]}$$

The Δ expression is a direct indication of how much change occurs for SET pulse through circuit gates. If Δ value is greater than 1, the majority of the measured SET pulses from each node under test until the observable outputs continuously increase. It also indicates that this pattern sensitizes circuit paths that enhance the SET propagated pulse. On the other hand, if Δ is smaller than 1, it reflects that the measured pattern turns

on CMOS gates that resist the propagation of SET pulses and try to reduce their effect in the circuit. The higher the Δ expression, the more critical the pattern becomes.

Each test used contributes to a different Δ value for the pattern because each test is independent of other tests. Therefore, adding more tests and investigating different Δ values for the patterns help differentiate more among them and make it easier to identify the worst-case candidate pattern. In the example of the two tests applied earlier, each pattern has two Δ values. The worst-case pattern should always be an enormous value, indicating that the pattern is the least resisting of the SET propagation through effective CMOS gates.

The second unitless factor is the sum of all Δ values measured after each applied test. If we have $N\ tests$ for a particular pattern, then the second unitless factor is:

$$\sum_{j=0}^{j=N} \Delta_j$$

Finally, the expression of the figure of merit for a pattern is the coverage factor times the enhancement factor:

$$covergae\ ratio * \sum_{j=0}^{j=N} \Delta_j$$

Therefore, the worst-case test vector is given by:

$$worst\ case = \ \text{arg}max_{pattern} \left\{ coverage\ ratio_{pattern} * \left[\sum_{j=0}^{j=N} \Delta_j \right]_{pattern} \right\}$$

In Table 7, the three most patterns with high coverage ratio, Δ values, and worst-case metric values are presented for the benchmark circuits previously analyzed in Chapter 4. The test vector with the most coverage percentage does not necessarily enhance the SET propagation; however, other patterns that achieve average coverage percentage show high SET propagation enhancement. The metric in the last column presents how patterns can be compared in terms of both coverage and SET enhancement. The table also shows patterns with higher coverage percentages, and his SET enhancement values are the best candidates to be worst-case test vectors.

The test vector suggested by the presented flow can be utilized for experimental trials to test SET pulses while applying this pattern. The same method can be applied to analyze the most applied vectors in combinational circuits and expect their performance against SET phenomena.

Table 7 Test vectors characterization for ISCAS85 benchmarks.

#	*Source benchmark*	*No. Test Vectors*	*Coverage Ratio of highest 3 patterns*		*Δ values of highest 3 patterns*		*Worst case metric of highest 3 patterns*	
			Pattern number	**Ratio**	**Pattern number**	**Value**	**Pattern number**	**Value**
1	C17	7	Pat1	0.34	Pat5	1.2127	Pat1	0.3573
			Pat3	0.34	Pat2	1.1593	Pat6	0.3483
			Pat6	0.32	Pat6	1.0885	Pat3	0.3476
2	C432	57	Pat54	0.1864	Pat3	3.4738	Pat3	0.5061
			Pat3	0.1457	Pat28	3.1550	Pat35	0.3321
			Pat12	0.1383	Pat26	3.0937	Pat55	0.3292
3	C499	93	Pat50	0.2255	Pat62	3.6242	Pat50	0.6437
			Pat3	0.1965	Pat65	3.488	Pat3	0.5565
			Pat12	0.1907	Pat12	3.4095	Pat28	0.5513
4	C7552	141	Pat49	0.2033	Pat23	4.4260	Pat129	0.6446
			Pat94	0.1914	Pat134	4.2620	Pat88	0.6111
			Pat119	0.1894	Pat135	4.1533	Pat49	0.5909

Chapter 6

Conclusion and Future Work

In this work, a review was made for ASIC standard cell characterization flow in terms of input SET and automated flow to cover all possible faulty nodes in combinational circuits fully, and generating sensitized paths for SET faults is tested. Also, full use of sensitized paths and SET characterized standard cells to measure entire combinational circuits SET sensitivity is also presented along with a validation flow performing real-time simulation to validate the flow results. A new metric studying SET enhancement to compare different test vectors is also presented and tested. Although the proposed analysis focuses on combinational circuits, it can also be expanded to include sequential circuits by applying circuit partitioning methods.

Reviewing the available characterization methods showed that SET characterization is only performed on pessimistic cases; however, it can be applied to produce a SET lookup table considering different capacitive loads and different SET transitions. Path sensitization automation flow extracts sensitized path for all circuit nodes aiming to achieve full SET fault coverage in combinational circuits using a commercial ATPG tool. Furthermore, an implemented SET sensitivity measurement tool is introduced to characterize full combinational circuits, illustrating SET fault analysis for propagating paths and extracting the worst propagating path for each node under test. A full automation flow to validate the pessimistic sensitivity results is implemented to execute real-time simulations on Cadence virtuoso with minimal human intervention. The proposed sensitivity flow would help circuit designers to immune their circuits to SET events and guide them during lab experiments.

To fully use the sensitivity flow, a more extensive set of SET characterized standard cells should be considered to obtain less pessimistic results. A liberty implementation to the extracted SET characteristics should be available and used while doing static timing analysis to study the SET robustness using the EDA tool and identify critical timing paths after place and route or synthesis.

References

[1] R. C. Baumann, “Landmarks in Terrestrial Single-Event Effects 2013 NSREC Short Course,” p. 94.

[2] “Sun Screen,” Dec. 10, 2000. [Online]. Available: https://www.forbes.com/forbes/2000/1113/6613068a.html?sh=a8582e61627b

[3] “Angle of Attack.” http://www.aviationchief.com/. [Online]. Available: http://www.aviationchief.com/angle-of-attack.html

[4] “In-flight upset 154 km west of Learmonth, WA 7 October 2008 VH-QPA Airbus A330-303.” Australian Transport Safety Bureau. [Online]. Available: https://www.atsb.gov.au/media/3532398/ao2008070.pdf

[5] A. Vittal and M. Marek-Sadowska, “Crosstalk Reduction for VLSI,” vol. 16, no. 3, p. 9, 1997.

[6] “Cosmic rays: particles from outer space.” CERN. [Online]. Available: https://home.cern/science/physics/cosmic-rays-particles-outer-space

[7] “Introduction to Single-Event Upsets,” p. 10, 2013.

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

[9] 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, Oct. 2017, pp. 35–44. doi: 10.1109/MIEL.2017.8190065.

[10] S. Krishnaswamy, I. L. Markov, and J. P. Hayes, “Logic Circuit Testing for Transient Faults,” p. 6.

[11] 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.

[12] S. Jagannathan *et al.*, “Independent Measurement of SET Pulse Widths From N-Hits and P-Hits in 65-nm CMOS,” *IEEE Trans. Nucl. Sci.*, p. 5595524, Dec. 2010, doi: 10.1109/TNS.2010.2076836.

[13] O. A. Amusan *et al.*, “Mitigation Techniques for Single-Event-Induced Charge Sharing in a 90-nm Bulk CMOS Process,” *IEEE Trans. Device Mater. Relib.*, vol. 9, no. 2, pp. 311–317, Jun. 2009, doi: 10.1109/TDMR.2009.2019963.

[14] M. J. Gadlage, J. R. Ahlbin, B. L. Bhuva, L. W. Massengill, and R. D. Schrimpf, “Single event transient pulse width measurements in a 65-nm bulk CMOS technology at elevated temperatures,” in *2010 IEEE International Reliability Physics Symposium*, Garden Grove (Anaheim), CA, USA, 2010, pp. 763–767. doi: 10.1109/IRPS.2010.5488736.

[15] J. R. Ahlbin, L. W. Massengill, B. L. Bhuva, B. Narasimham, M. J. Gadlage, and P. H. Eaton, “Single-Event Transient Pulse Quenching in Advanced CMOS Logic Circuits,” *IEEE Trans. Nucl. Sci.*, vol. 56, no. 6, pp. 3050–3056, Dec. 2009, doi: 10.1109/TNS.2009.2033689.

[16] B. Narasimham *et al.*, “Characterization of Digital Single Event Transient Pulse-Widths in 130-nm and 90-nm CMOS Technologies,” *IEEE Trans. Nucl. Sci.*, vol. 54, no. 6, pp. 2506–2511, Dec. 2007, doi: 10.1109/TNS.2007.910125.

[17] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage, and T. Turflinger, "Digital Single Event Transient Trends With Technology Node Scaling," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 6, pp. 3462–3465, Dec. 2006, doi: 10.1109/TNS.2006.886044.
[18] P. M. Gouker *et al.*, "Effects of Ionizing Radiation on Digital Single Event Transients in a 180-nm Fully Depleted SOI Process," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 6, pp. 3477–3482, Dec. 2009, doi: 10.1109/TNS.2009.2034153.
[19] 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.
[20] G. C. Messenger, "Collection of Charge on Junction Nodes from Ion Tracks," *IEEE Transactions on Nuclear Science*, vol. 29, no. 6, pp. 2024–2031, Dec. 1982, doi: 10.1109/TNS.1982.4336490.
[21] L. B. Freeman, "Critical charge calculations for a bipolar SRAM array," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 119–129, Jan. 1996, doi: 10.1147/rd.401.0119.
[22] C. Hu, "Alpha-particle-induced field and enhanced collection of carriers," *IEEE Electron Device Letters*, vol. 3, no. 2, pp. 31–34, Feb. 1982, doi: 10.1109/EDL.1982.25467.
[23] T. Merelle *et al.*, "Criterion for SEU occurrence in SRAM deduced from circuit and device Simulations in case of neutron-induced SER," *IEEE Transactions on Nuclear Science*, vol. 52, no. 4, pp. 1148–1155, Aug. 2005, doi: 10.1109/TNS.2005.852319.
[24] X. Gili, S. Barcelo, S. à A. Bota, and J. Segura, "Analytical Modeling of Single Event Transients Propagation in Combinational Logic Gates," *IEEE Trans. Nucl. Sci.*, vol. 59, no. 4, pp. 971–979, Aug. 2012, doi: 10.1109/TNS.2012.2187071.
[25] A. Mohamed, "Fault Modeling and Test Vector Generation for ASIC Devices Exposed to Space Single Event Environment," 2021. [Online]. Available: https://fount.aucegypt.edu/cgi/viewcontent.cgi?article=2669&context=etds
[26] K. K. Saluja, "TESTING AND TESTABLE DESIGN OF DIGITAL SYSTES," Jan. 13, 2012. [Online]. Available: https://www.ee.iitb.ac.in/~viren/Courses/2012/EE709/Lecture4.pdf
[27] Y. Zorian, S. Dey, and M. J. Rodgers, "Test of future system-on-chips," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD - 2000. IEEE/ACM Digest of Technical Papers (Cat. No.00CH37140)*, San Jose, CA, USA, 2000, pp. 392–398. doi: 10.1109/ICCAD.2000.896504.
[28] G. Fuchs, "Implications of VLSI Fault Models and Distributed Systems Failure Models — A Hardware Designer's View," Dagstuhl Seminar Proceedings, 2009, p. 7. [Online]. Available: https://core.ac.uk/download/pdf/62914078.pdf
[29] "Stuck-open and Stuck-on Faults." http://ece-research.unm.edu/jimp/vlsi_test/slides/html/faults2.html
[30] P. Janak, "Stuck-At Fault: A Fault Model for the next Millennium?," Aug. 22, 2005. [Online]. Available: https://web.stanford.edu/class/ee386/public/stuck_at_fault_6per_page

[31] J. Plusquellic, “Failures in Integrated Circuits,” Aug. 28, 2207. http://ece-research.unm.edu/jimp/vlsi_test/slides/html/defects1.html
[32] Zhuo Li, Xiang Lu, Wangqi Qiu, Weiping Shi, and D. M. H. Walker, “A circuit level fault model for resistive opens and bridges,” in *Proceedings. 21st VLSI Test Symposium, 2003.*, Napa, CA, USA, 2003, pp. 379–384. doi: 10.1109/VTEST.2003.1197678.
[33] A. K. Majhi and V. D. Agrawal, “Tutorial: Delay Fault Models and Coverage,” p. 6.
[34] B. Yao, “Transition Faults and Transition Path Delay Faults: Test Generation, Path Selection, and Built-In Generation of Functional Broadside Tests,” 2013.
[35] “Benchmarks - iscas85,” Jan. 02, 2007. http://www.pld.ttu.ee/~maksim/benchmarks/iscas85/verilog/
[36] Siemens, “Tessent FastScan: Automatic test pattern generation.” Siemens, 2019. [Online]. Available: https://resources.sw.siemens.com/en-US/fact-sheet-tessent-fastscan
[37] D. Storie, “FastScan and FlexTest Reference Manual.”
[38] M. C. Hansen, H. Yalcin, and J. P. Hayes, “Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering,” *IEEE Des. Test. Comput.*, vol. 16, no. 3, pp. 72–80, Sep. 1999, doi: 10.1109/54.785838.
[39] 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, Sep. 2013, pp. 1–5. doi: 10.1109/RADECS.2013.6937388.
[40] “WebPlotDigitizer.” https://apps.automeris.io/wpd/
[41] S. Sayil, “A survey of circuit-level soft error mitigation methodologies,” *Analog Integr Circ Sig Process*, vol. 99, no. 1, pp. 63–70, Apr. 2019, doi: 10.1007/s10470-018-1300-8.

Appendix A

This appendix records the necessary scripts used all over this work.

I. Perl

Perl is a scripting language that defines an automatic script's repeated execution. It is derived from the IEEE 1364 Verilog HDL specification. Perl is used to generate custom FastScan do files to analyze each test vector independently and obtain its corresponding covered nets due to its application. Also, it is used to generate another custom FastScan do file to write down the possible paths from the node under test to primary outputs as described in Chapter 4.

A. Perl script to produce a FastScan file to analyze each test vector.

```
#!/usr/bin/perl
use strict;
use warnings;
open (PAT_FILE, 'all.patterns') or die("Could not open  file.");
my $setup_flag = 0;
my @setup;
my $count = 0;
my $end = "end;\n";
my $pattern_flag = 0;
my @pattern;
my $pat_file_name = "patterns/my_ext_pat";
my $extension = ".txt";
my $name;
while (my $line = <PAT_FILE>) {
# get the setup part
    if ( $line =~ /SETUP =/ ) {
      $setup_flag = 1;
      #print ("reached setup \n");
    }
    if ($setup_flag == 1) {
     push(@setup,$line);
    if ( $line =~ /SCAN_TEST =/ ) {
    #print ("reached SCAN_TEST\n");
     $setup_flag = 0;
    }
    }
# get patterns
    if ( $line =~ /pattern/ ) {
```

```
            $pattern_flag = 1;
            #print ("reached pattern \n", $count);
        }
        if ($pattern_flag == 1) {
         push(@pattern,$line);
        if ($line =~ /measure/) {
    # need to erase all pattern info
            $name = $pat_file_name . $count . $extension;
            open(DES, '>', $name) or die $!;
            #print($name);
            push( @pattern, $end);
            print ( DES @setup);
            print ( DES @pattern);
            @pattern = ();
            $pattern_flag = 0;
                $count = $count + 1;
        }
        }
    }
    close (PAT_FILE);
    # write tut.do
    my $dofile = "tut.do";
    my $ext_file_name;
    my $cnt = 0;
    open(DO_FILE, '>', $dofile) or die $!;
    print( DO_FILE "set system mode atpg \n");
    while ($cnt < $count) {
        print( DO_FILE "set pattern source external ");
        $ext_file_name = $pat_file_name . $cnt . $extension;
        print( DO_FILE $ext_file_name);
        print( DO_FILE "\n");
        print( DO_FILE "set fault type Transition \n");
        print( DO_FILE "add faults -all \n");
        print( DO_FILE "run \n");
        print( DO_FILE "report statistics > stats/$cnt.stats \n");
        print( DO_FILE "write fault faults/mypats.flt$cnt -replace  \n\n\n");
        $cnt = $cnt + 1;
    }
    print( DO_FILE "save patterns Junk_pattern_to_quit.patterns - Ascii -replace\nexit
    \n");
    close(DO_FILE);
    exit;
```

B. Perl script to generate custom FastScan file to generate paths from fault under test to primary ouptuts.

```
#!/usr/bin/perl
use strict;
use warnings;
open(my_compare_FILE, '>', "compare_files.txt") or die $!;
print( my_compare_FILE "pat_file \t\t flt_file \n");
my @pat_files   = glob("patterns/my_ext_pat*.txt");
my @fault_files = glob("faults/mypats.flt*");
my $num_files = scalar @pat_files;
my $num_files2 = scalar @pat_files;
my $print_files = 1;
if ($num_files != $num_files2) {
    print "the pattern files do not match the fault files";
}
if ($print_files == 1) {
foreach my $pat_file (@pat_files) {
    print( my_compare_FILE "$pat_file \t");
    print "$pat_file \n";
}
foreach my $flt_file (@fault_files) {
    print( my_compare_FILE "$flt_file \n");
    print "$flt_file \n";
}
}
close(my_compare_FILE);
my @primary_outputs;
my @nodes;
my @observable_outputs;
my $end_observable_outputs = 0;
open(DO_FILE, '>', "generate_paths.do") or die $!;
print( DO_FILE "set system mode atpg \n");
open( OUT_ports, '>', "my_outs.txt") or die $!;
for (my $cnt = 0; $cnt < $num_files; $cnt = $cnt + 1) {
    print "$cnt \n";
# open pattern_file + record outputs
    open(PAT, @pat_files[$cnt]);
    while (my $line = <PAT>) {
        if ($line =~ /declare output bus "PO" =/ ) {
            while ($end_observable_outputs == 0) {
                my @temp_outputs = ( $line =~ m/"(\/[^\s]+)"/g) ; #  /N22
                push( @primary_outputs,@temp_outputs);
                my $size_a = scalar @primary_outputs;
```

```
                    foreach my $flt_file (@primary_outputs) {
                        print "$flt_file --> $line\n";
                    }
                    if ($line =~ /end;/) {
                        $end_observable_outputs = 1;
                    }
                    $line = <PAT>
                }
                $end_observable_outputs = 0;
            }
        }
        close(PAT);
        foreach my $pat_file (@primary_outputs) {
            print( OUT_ports "$pat_file \t");
            print "$pat_file \n";
        }
        my $size_a = scalar @primary_outputs;
        print( OUT_ports "$size_a");
        print( OUT_ports "\n");
    # open fault_file + record detected outputs
        open(FLT, @fault_files[$cnt]);
        while (my $line = <FLT>) {
            if ($line =~ /0,  DS,|1,  DS,/) {
                my @node = ($line =~ m/"(\/[^\s]+)"/g);
                #print ("@node\n");
                push( @nodes,@node);
            }
        }
        foreach my $net (@nodes) {
            if ({ map { $_ => 1 } @primary_outputs }->{$net}) {
                push( @observable_outputs,$net);
            }
        }
    # start filter observable outputs from nodes (nodes - observable_outputs = new_nodes)
        my %total;
        $total{$_} = 1 for @nodes;
        delete $total{$_} for @observable_outputs;
        @nodes = keys %total;
    # start  wiritng the DO file for the FASTSCAN
        print( DO_FILE "set pattern source external @pat_files[$cnt] \n");
        print( DO_FILE "set fault type Transition \n");
        print( DO_FILE "add faults -all \nrun \n");
        print( DO_FILE "set_gate_report pattern 0 \n");
        print( DO_FILE "set_gate_level design \n");
```

```
            my $l_nodes = scalar @nodes;
            my $l_obs_outputs = scalar @observable_outputs;
            for (my $n_cnt = 0; $n_cnt < $l_nodes; $n_cnt = $n_cnt + 1) {
                for (my $n_outpts = 0; $n_outpts < $l_obs_outputs; $n_outpts = $n_outpts + 1)
        {
                    my $file_ext = "pat" . $cnt . "\_from_" . @nodes[$n_cnt] . "_to_" .
        @observable_outputs[$n_outpts] . ".path";
                    $file_ext =~ tr/\//_/;
                    print( DO_FILE "report_gates -path @nodes[$n_cnt]
        @observable_outputs[$n_outpts] > paths/$file_ext \n");
                }
            }
            print (DO_FILE "\n");
            close(FLT);
            @observable_outputs=();
            @primary_outputs=();
            @nodes=();
        }
        print( DO_FILE "save patterns Junk_pattern_to_quit.patterns - Ascii -replace\nexit
        \n");
        close(DO_FILE);
        close(OUT_ports);
```

II. MATLAB

MATLAB analyzes the path files generated from FastScan and constructs directed graphs to obtain possible SET propagation paths. Also, MATLAB utilizes the standard cell characterization step by producing SET models. Implemented algorithms in MATLAB to extract possible SET propagating paths and measure SET sensitivity with the help of SET characterized gate models. In addition, it is also used in producing a testing OCEAN script to validate the MATLAB results with real-time simulations performed on Cadence Virtuoso.

i. The Top (set_charactrize)

```
clear;clc;
tic;
%%%%%%%%%%%% Parameters
global v_syn; v_syn = 0.1; % volt
global t_syn; t_syn = 7.55;   % pico second

%%%%%%%%%%%% Parameters not to change for SET synsitivity
global tw_max_period; tw_max_period=300;
global tw_min_period; tw_min_period=10;
```

```
global v_max; v_max=1.8;
global v_min; v_min=v_max/20;

%%%%%%%%%%% Models directory
global models_path; models_path = 'new';

global synth; synth = 1;
% HDL VDD 1.8
% NA2_3VX0 3.0

%%%%%%%%%%%
circuit_name = string("c432");
if circuit_name == string("c17")
    path_dir = pwd + "/synth_paths/c17/paths";
    stats_path = pwd + "/synth_paths/c17/stats";
    cell_map_dir = pwd + "/synth_paths/c17/syn_output/c17_cell_map.txt";
    input_ports = ["N1" "N2" "N3" "N6" "N7"];
elseif circuit_name == string("c17_DS1")
    path_dir = pwd + "/synth_paths/c17_1_DS_only/paths";
    stats_path = pwd + "/synth_paths/c17_1_DS_only/stats";
    cell_map_dir = pwd + "/synth_paths/c17_1_DS_only/c17_cell_map.txt";
    input_ports = ["N1" "N2" "N3" "N6" "N7"];
elseif circuit_name == string("c432")
    path_dir = pwd + "/synth_paths/c432/paths";
    stats_path = pwd + "/synth_paths/c432/stats";
    cell_map_dir = pwd + "\synth_paths\c432\syn_output\c432_cell_map_x4.txt";
    input_ports = ["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"];
else
    path_dir = pwd + "/synth_paths/c432_dontuse/paths";
    stats_path = pwd + "/c432_stats";
    cell_map_dir = pwd + "\synth_paths\c432_dontuse\c432_cell_map_x2.txt";
end
files = dir(fullfile(path_dir, '*.path')) ;
SET_results = cell(length(files),16);
SET_results_new = cell(length(files),16);
non_physical_paths = 0;
no_output_transition = 0;
no_path_transition = 0;
no_start_gate_in_file = 0;
no_TO_DEBUG_FILES = 0;
starts_from_output_pin =0;
TO_DEBUG_FILES = [];
```

```
TO_DEBUG_start_gate = [];
n_files_100x = length(files) / 100;
current_mod = 0;
% build cell mapping --> map cell in a netlist name to reference name
[cell2ref] = parse_cell_map(cell_map_dir);
fault_type = cell(length(files),2);
% loop over all file pattern files
for n_file = 594737:length(files)%594737  451836  283038:length(files)
    n_file;
    current_path_file = path_dir + "/" +files(n_file).name;
    str=split(files(n_file).name,["_from_","_to_",".path"]);
    SET_results(n_file,1)={files(n_file).name};
    SET_results(n_file,2)={str{1}};
    SET_results(n_file,3)={str{2}};
    SET_results(n_file,4)={str{3}};
    fault_type(n_file,1);
    fault_type(n_file,1) = SET_results(n_file,1);
    % generate propagation paths
    if (synth == 0)
        [prop_path_gates,status] = parse_pathfile_2(current_path_file);
    else
        [prop_path_gates,status,DS_type] =
parse_pathfile_6(current_path_file,input_ports);
    end
    SET_results(n_file,16) = {DS_type};
    SET_results(n_file,11) = {status};
    if (isempty(prop_path_gates))
        switch(status)
            case 'no_gates'
                non_physical_paths = non_physical_paths +1;
            case 'no_output_transition'
                no_output_transition = no_output_transition +1;
            case 'no_path_transition'
                no_path_transition = no_path_transition +1;
            case 'it is output pin'
                starts_from_output_pin = starts_from_output_pin +1;
            case 'no_start_gate in path file'
                no_start_gate_in_file = no_start_gate_in_file + 1;
                TO_DEBUG_start_gate = [TO_DEBUG_start_gate n_file];
            case 'ok'
                no_TO_DEBUG_FILES = no_TO_DEBUG_FILES + 1;
                TO_DEBUG_FILES = [TO_DEBUG_FILES n_file];
        end
        continue;
```

```
    end
    %%% transform gates to one model
    dut_cells = cell(size(prop_path_gates,1),size(prop_path_gates,2));
        dut_cells = {};
        for path = 1 : size(prop_path_gates,1)
            emptyCells = cellfun(@isempty,prop_path_gates(path,:));
%             prop_path_gates(emptyCells)=[];
            for i=1:(length(prop_path_gates(path,:))-sum(emptyCells))
                %char(prop_path_gates(path,i));
                dut_cells{path,i} = cell2ref(char(prop_path_gates(path,i)));
            end
        end
    if (length(dut_cells) ~=0 )
        [SET_results, SET_results_new] =
measure_synsitivity(SET_results,SET_results_new,n_file,dut_cells);
    end
    % display progress
     new_mod = floor(n_file./n_files_100x);
     if (new_mod > current_mod)
         current_mod = new_mod;
         disp( ['completed ',num2str(current_mod),'% files'])
     end
end
disp( ['there are ',num2str(non_physical_paths),' files that have no gates in
files'])
disp( ['there are ',num2str(no_output_transition),' files that have
no_output_transition'])
disp( ['there are ',num2str(no_path_transition),' files that have
no_path_transition'])
disp( ['there are ',num2str(no_TO_DEBUG_FILES),' files that have of status while no
paths exist'])
disp( ['there are ',num2str(no_start_gate_in_file),' files that have no mentioned
start exist'])
real_paths = -(non_physical_paths +no_output_transition + no_path_transition +
no_TO_DEBUG_FILES + no_start_gate_in_file) + size(SET_results,1);
real_paths_percent = 100 * (1 - (non_physical_paths +no_output_transition +
no_path_transition) / size(SET_results,1));
SET_results(TO_DEBUG_FILES,:) = [];
SET_results_new(TO_DEBUG_FILES,:) = [];
% filter out (no gates + no_output_transition + no_path_transition)
SET_results_real = SET_results(find(string(SET_results(:,11)) == 'ok'),:);

%% Analyze the number of faults
% Number of Detected faults after removing outputs in all. Faults should
```

```
% equal these DS_faults of received files.
DS_faults = unique(string(SET_results(:,3)));
DS_faults_that_ok = unique(string(SET_results_real(:,3)));
%% adjust 'SET_results_new' for all values in graph
SET_results_new(:,1:4) = SET_results(:,1:4);
SET_results_new(:,15) = SET_results(:,11);
SET_results_new(:,16) = SET_results(:,16);
SET_results_real_new = SET_results_new(find(string(SET_results_new(:,15)) ==
'ok'),:);
paths_with_absolute_immune_to_technology =
SET_results_real_new(find(cell2mat(SET_results_real_new(:,12)) > 150),:);
paths_not_immune_to_technology =
SET_results_real_new(find(cell2mat(SET_results_real_new(:,6)) > 1.3),:);
save("c432")
```

ii. Parse path file (parse_pathfile_6)

```
function [prop_path_gates,status,fault_type_invistigated] =
parse_pathfile_6(pathfile_dir,input_ports)
    str=split(pathfile_dir,["_from__","_to_",".path"]);
    start_gate_port = {str{2}};
    idx = ismember(input_ports,start_gate_port);
    idx = sum(idx);
    not_include_first_gate = 0;
    starts_from_input_pin = 0;
    start_gate_port = char(start_gate_port);
    if (idx == 0)
        if (start_gate_port(end) == 'Q')
           not_include_first_gate = 1;
        elseif (regexp(start_gate_port(end),"[ABCD]") == 1)
            starts_from_input_pin =1 ;
        end
        start_gate_port = start_gate_port(1:end-2);
    end
    % 1st open for file
    status = 'ok';
    gate2num = containers.Map('KeyType','char','ValueType','double');
    num2gate = containers.Map('KeyType','double','ValueType','char');
    collect_parent = [];
    collect_child = [];
    collect_transition = []; % starts from 0 --> means start from 1 to 0 -- DS1 --
                        % while starts from 1 --> means start from 0 to 1 -- DS0
    PI_flag = 0;
```

```
    remove_primary_input_node = 0;
    first_gate = 0;
    primary_output = 0;
    no_path_exists = 0;

    % start parsing
    fid = fopen(pathfile_dir);
    my_line =  fgetl(fid);
    while ischar(my_line)
        % Search for the input of the gate
        match_IN = regexp(my_line,'//  /(\w*)\s+(\w*)','tokens');
        if (~isempty(match_IN))
            % get current parent gate name + value
            gate = string(match_IN{1}(1,1));
            type = string(match_IN{1}(1,2));
            [gate2num,num2gate,parent_idx] =
return_gate_id(gate2num,num2gate,gate);
            if (isempty(collect_child) && PI_flag)
                collect_parent=[1];
                collect_child=[parent_idx];
                PI_flag = 0;
                first_gate = parent_idx;
            elseif (isempty(collect_child))
                first_gate = parent_idx;
            end
            if (type == "primary_input")
                PI_flag = 1;
                remove_primary_input_node = 1;
            end
            if (type == "primary_output")
                primary_output = parent_idx;
            end
        end
        %%% parse the nodes after ("primary_input")
        [primary_match_OUT, port_output_nodes] = regexp(my_line,'O  \(\d*-
\d*\)  ','match','split');
         if (PI_flag && ~isempty(primary_match_OUT))
            PI_flag = 0;
            tokens =
regexp(string(port_output_nodes(2)),'/(\w+)/[ABCDEF]\d*','tokens');
            child_v = zeros(1,length(tokens));
            for i=1:length(tokens)
                [gate2num,num2gate,child_v(i)] =
return_gate_id(gate2num,num2gate,string(tokens(i)));
```

```
            end
            parent_v = ones(1,length(tokens)) * parent_idx;
            collect_parent = [collect_parent parent_v];
            collect_child = [collect_child child_v];
            [primary_match_OUT, port_output_nodes] = regexp(my_line,'\(\d*-
\d*\)','match','split');
            if (primary_match_OUT == "(000-111)")
                my_transition = ones(1,length(child_v));
                collect_transition = [collect_transition my_transition];
            elseif (primary_match_OUT == "(111-000)")
                my_transition = zeros(1,length(child_v));
                collect_transition = [collect_transition my_transition];
            end
         end

        %%% parse the line
        [match_OUT, output_nodes] = regexp(my_line,'//      Q      O  \((000-
111|111-000)\)  ','match','split');
        if (~isempty(match_OUT))
            output_nodes(1)=[];
            tokens = regexp(string(output_nodes(1)),'/(\w+)/[ABCDEF]\d*','tokens');
            tokens2 = regexp(string(output_nodes(1)),'/N(\w+)','tokens');
            if (length(tokens2) > 0)
                for i=1:length(tokens2)
                    tokens2(i) = {"N" + string(tokens2(i))};
                end
                tokens = [tokens tokens2];
                tokens2 = {};
            end
            if (length(tokens) == 0 )
                tokens = regexp(string(output_nodes(1)),'/(\w+)','tokens');
            end
            child_v = zeros(1,length(tokens));
            for i=1:length(tokens)
                [gate2num,num2gate,child_v(i)] =
return_gate_id(gate2num,num2gate,string(tokens(i)));
            end
            parent_v = ones(1,length(tokens)) * parent_idx;
            collect_parent = [collect_parent parent_v];
            collect_child = [collect_child child_v];
            [primary_match_OUT, port_output_nodes] = regexp(my_line,'\(\d*-
\d*\)','match','split');
            if (primary_match_OUT == "(000-111)")
                my_transition = ones(1,length(child_v));
```

```
                collect_transition = [collect_transition my_transition];
            elseif (primary_match_OUT == "(111-000)")
                my_transition = zeros(1,length(child_v));
                collect_transition = [collect_transition my_transition];
            end
        end
        % get the number of gates
        tokens = regexp(string(output_nodes(1)),'//  Number gates in trace =
(\d+)\.','tokens');
        if( ~isempty(tokens) && str2double(string(tokens(1))) == 0)
            display('no path exists');
            no_path_exists = 1;
        end
        my_line = fgetl(fid);
    end
    fclose(fid);
    if (no_path_exists)
        prop_path_gates = [];
        status = 'no_gates';
        fault_type_invistigated='';
        return;
    end
    if (length(collect_parent) ~= length(collect_child))
        error('nodes are not matched correctly');
    end
    graph = digraph(collect_parent, collect_child);
    primary_output;
    first_gate;
    % find if the output point has a transiotion (000-111) or (111-000) and
reported in graph ?
    if (~(findnode(graph,primary_output)))
        prop_path_gates = [];
        status = 'no_output_transition';
        fault_type_invistigated='';
        return;
    end
    check_if_key_exists = isKey(gate2num,start_gate_port);
    if (check_if_key_exists == 0)
        prop_path_gates = [];
        status = 'no_start_gate in path file';
         fault_type_invistigated='';
        return;
    end
    first_gate = gate2num(start_gate_port);
```

```
    if (remove_primary_input_node)
        idx = ismember(input_ports,start_gate_port);
        idx = sum(idx);
        if (idx == 0)
            remove_primary_input_node = 0;
        end
    end
    %%
    % find if there is a path existing
    %plot(graph);
    short_path = shortestpath(graph, first_gate, primary_output);

    if (~isempty(short_path))
        % get all paths from first_gate to primary_output
        pth=pathof(graph,first_gate,primary_output);

        prop_path_gates = cell(length(pth),1);
        for i = 1 : length(pth)
            my_path = cell2mat(pth(i));
            %% TODO remove path when primary input
            collect_parent;
            collect_child;
            collect_transition;
            if (remove_primary_input_node == 1)
               for j = 2 : length(my_path)-1 % to remove the (primary input [1] +
primary output [end]) from the path
                    prop_path_gates(i,j-1) = {num2gate(my_path(j))};
                end
            else
               for j = 1 : length(my_path)-1 % to remove the (primary output [end])
from the path
                    prop_path_gates(i,j) = {num2gate(my_path(j))};
               end
            end

        end
    else
        status = 'no_path_transition';
        prop_path_gates = {};
    end
    if (size(prop_path_gates,1) == 0)
    end
    if  (status == "no_path_transition")
```

```
        fault_type_invistigated='';
    elseif (size(prop_path_gates,1) == 0)
        status = 'it is output pin';
        fault_type_invistigated='';
        return;
    else
        if (not_include_first_gate == 1)
            prop_path_gates = prop_path_gates(:,2:end);
        end
        if (not_include_first_gate || remove_primary_input_node ||
starts_from_input_pin)
            transition_from_parent = collect_transition(find(collect_parent ==
first_gate));
            fault_type_invistigated =  transition_from_parent(1);
        else
            transition_to_child = collect_transition(find(collect_child ==
first_gate));
            fault_type_invistigated =  transition_to_child(1);
        end
    end
    end
    function di_graph_with_names(num2gate,collect_parent,collect_child)
        collect_parent_name = cell(1,length(collect_parent));
        collect_child_name = cell(1,length(collect_child));
        for point = 1:length(collect_parent)
            collect_parent_name(point) = {num2gate(collect_parent(point))};
            collect_child_name(point) = {num2gate(collect_child(point))};
        end
        graph_name = digraph(collect_parent_name, collect_child_name);
        plot(graph_name);
    end
```

iii. Measure SET sensitivity (measure_synsitivity)

```
function [SET_results,SET_results_new] =
measure_synsitivity(SET_results,SET_results_new,n_file,dut_cells)
    global v_syn t_syn v_max v_min tw_max_period tw_min_period;
    local_SET_results = cell(1,4);
    % local_SET_results_new = {min v_in, tw_max_period, V_out, t_out} {v_max, min
t_in, v_out, t_out}
    local_SET_results_new = cell(1,8); % new one to include {test} {test2}
    for path = 1:size(dut_cells,1)
        %% Test1
        found_v_min = 0;
```

```
        v_in = v_min;
        [~,v_th] = charactrize_path(tw_max_period,v_in,dut_cells(path,:));
        % record min voltage synsitivity
        while (found_v_min == 0)
            [tw_out_v,v_min_out] =
charactrize_path(tw_max_period,v_in,dut_cells(path,:));
            if (v_min_out > v_max/2)
                found_v_min = 1;
                break;
            end
            v_in = v_syn + v_in;
            if (v_in > v_max)
                found_v_min = 1;
                break;
            end
        end
        local_SET_results(path,1) = {v_in};local_SET_results(path,2) = {v_min_out};
        local_SET_results_new(path,1) = {v_in} ;     local_SET_results_new(path,2)
= {tw_max_period};
        local_SET_results_new(path,3) = {v_min_out} ;local_SET_results_new(path,4)
= {tw_out_v};
        %% Test2
        % record min pulse width
        found_t_min = 0;
        tw_in = tw_min_period;
        [t_th,~] = charactrize_path(tw_in,v_max,dut_cells(path,:));
        while (found_t_min == 0)
            [tw_min_out,v_out_t] = charactrize_path(tw_in,v_max,dut_cells(path,:));
            if ((tw_min_out > tw_max_period/4) && (v_out_t > v_max/2))
                found_t_min = 1;
                break;
            end
            tw_in = t_syn + tw_in;
            if (tw_in > tw_max_period)
                found_t_min = 1;
                break;
            end
        end
        local_SET_results(path,3) = {tw_in};local_SET_results(path,4) =
{tw_min_out};
        local_SET_results_new(path,5) = {v_max} ;  local_SET_results_new(path,6) =
{tw_in};
        local_SET_results_new(path,7) = {v_out_t} ;local_SET_results_new(path,8) =
{tw_min_out};
```

```
    end

    %%%%%%%%%% record the v_min when large time period exists.
    [~,v_min_index]= min(cell2mat(local_SET_results(:,1)));
    SET_results(n_file,6 : 7) = local_SET_results(v_min_index,1:2);
    SET_results(n_file,5)={length(dut_cells(v_min_index,:))};

    %%%%%%%%%% record the t_min when full VDD exists.
    [~,t_min_index]= min(cell2mat(local_SET_results(:,3)));
    SET_results(n_file,9 : 10) = local_SET_results(t_min_index,3:4);
    SET_results(n_file,8)={length(find(~cellfun(@isempty,dut_cells(t_min_index,:)))
)};
    %% SET_results_new
    [~,v_min_index]= min(cell2mat(local_SET_results(:,1)));
    SET_results_new(n_file,5)={length(dut_cells(v_min_index,:))};
    SET_results_new(n_file,6 : 9) = local_SET_results_new(v_min_index,1:4);
    [~,t_min_index]= min(cell2mat(local_SET_results(:,3)));
    SET_results_new(n_file,11 : 14) = local_SET_results_new(t_min_index,5:8);
    SET_results_new(n_file,10)={length(find(~cellfun(@isempty,dut_cells(t_min_index
,:))))};


    end
```

iv. Characterize each propagating path (characterize_path)

```
function [tw_out,v_out] = charactrize_path(tw_in,v_in,dut_cells)
    global v_syn t_syn v_max v_min tw_max_period tw_min_period models_path;
    tw_temp = tw_in;
        v_temp=v_in;
        for i = 1:length(dut_cells)
            if (isempty( dut_cells{i}))
                continue;
            end
            dut_name = dut_cells{i};
            if (strcmp(models_path,'new'))
                load(['./new_models/',dut_name,'_model.mat']);
            else
                load(['./models/',dut_name,'_model.mat']);
            end
            if (strcmp(dut_name,'NA2_3VX0')) % (dut_name == 'NA2_3VX0')
                v_out = v_model(tw_temp,v_temp);
```

```
                tw_out = tw_model(tw_temp,v_temp);
            else
                v_out = v_model1(tw_temp,v_temp);
                tw_out = tw_model1(tw_temp,v_temp);
            end
            if (v_out <= v_min)
                v_out = v_min;
            elseif (v_out >= v_max)
                v_out = v_max;
            end
            if (tw_out <= tw_min_period)
                tw_out = tw_min_period;
            elseif (tw_out >= tw_max_period)
                tw_out = tw_max_period;
            end
            tw_temp = tw_out;
            v_temp=v_out;
        end
    end
```

v. Charactrize each pattern (measure_pattern_mean_areas_test_ratios)

```
function [my_patterns] =
measure_pattern_mean_areas_test_ratios(SET_results_real_new,patterns_count,index_al
l_group)

%we have two tests 1: test V height in  --  T width in
%the idea is to get the change of areas in 1st trial and 2nd trial and the worst
will be the max_xum :)
%my_patterns =    {count --> should be coverage} {mean(area_in (V_in*t_in)),
mean(area_in (V_out*t_out)), mean_A_in/mean_A_out}
%                                                 {mean(area_in (V_in*t_in)),
mean(area_in (V_out*t_out)), mean_A_in/mean_A_out}
%                                                    {sum of 2 areas} {area1 /
area2} (sum teas1_A_ratio + teas2_A_ratio)
%              1 --> pattern0   {}
%              2 --> pattern1   {}
%              N --> pattern{N) {}

properties_count = 10; % this variable responsible for the number of properties
needed to compare
my_patterns = zeros(patterns_count, properties_count);

for i=1:patterns_count
    idx_nodes_under_tests = find(index_all_group == i);
    pat_invistigated = SET_results_real_new(idx_nodes_under_tests,:);

    % delta height and width
    area_in_t1  = cell2mat(pat_invistigated(:,6)) .*
cell2mat(pat_invistigated(:,7));
```

```
    area_out_t1 = cell2mat(pat_invistigated(:,8)) .*
cell2mat(pat_invistigated(:,9));

    area_in_t2  = cell2mat(pat_invistigated(:,11)) .*
cell2mat(pat_invistigated(:,12));
    area_out_t2 = cell2mat(pat_invistigated(:,13)) .*
cell2mat(pat_invistigated(:,14));

    % 1 - put the count of nodes under test
    my_patterns(i,1) = length(idx_nodes_under_tests);

    % 2 - put the mean (V_in  * tin)  -t1
    % 3 - put the mean (V_out * tout) -t1
    % 4 - put mean_A_in/mean_A_out

    % 5 - put the mean (V_in  * tin)  -t2
    % 6 - put the mean (V_out * tout) -t2
    % 7 - put mean_A_in/mean_A_out -t2
    my_patterns(i,2) = mean(area_in_t1);
    my_patterns(i,3) = mean(area_out_t1);
    my_patterns(i,4) = my_patterns(i,3)/my_patterns(i,2);

    my_patterns(i,5) = mean(area_in_t2);
    my_patterns(i,6) = mean(area_out_t2);
    my_patterns(i,7) = my_patterns(i,6)/my_patterns(i,5);

    % 8 - put the sum of 2 area -out
    % 9 -  put the area ratio b/w area1/area2 -out
    % 10 - put the sum of 2 area ratios
    my_patterns(i,8) =  my_patterns(i,3) +  my_patterns(i,5);
    my_patterns(i,9) =  my_patterns(i,3) / my_patterns(i,5);
    my_patterns(i,10) =  my_patterns(i,4) +  my_patterns(i,7);

end
```

III. Ocean

OCEAN script analyzes the benchmark after adding some edits to the benchmark gate netlist by changing the connection of the SET pulse source to the netlist. The primary output where the SET pulse upset occurs is monitored. The SET pulse characteristic at the output port is measured as the SET height and pulse width.

i. Ocean Script

```
sh("mkdir /home/vlsi/Desktop/wael_validation/c17/netlists")
sh("rm -rf /home/vlsi/Desktop/wael_validation/c17/netlists/*")
sh("mkdir /home/vlsi/Desktop/wael_validation/c17/output")
sh("rm -rf /home/vlsi/Desktop/wael_validation/c17/output/*")
; save a backup for the netlist
sh("mv
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist_backup")
```

```
sh("cp -rf
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist_backup
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist")
out_f = outfile( "../c17_DS1.out" "w" )


;-----------------------------------------------------;

; copy the original netlist to a my directory - apply netlist modification -
return it back to simulation directory \n
sh("cp -rf
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist
/home/vlsi/Desktop/wael_validation/c17/netlists/netlist_pat1_from__g73__4319
_A_to__N23")
sh("sed -i --null-data 's/Pulse_1 (n_1 0 control_SET)/Pulse_1 (n_1 0
control_SET)/g'
/home/vlsi/Desktop/wael_validation/c17/netlists/netlist_pat1_from__g73__4319
_A_to__N23")
sh("cp -rf
/home/vlsi/Desktop/wael_validation/c17/netlists/netlist_pat1_from__g73__4319
_A_to__N23
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist")
;; row : 4 ;;;; test : 1 ;;

simulator( 'spectre )
design(  "/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schemati
c/netlist/netlist")
resultsDir( "/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schem
atic" )
modelFile(

'("/home/vlsi/Desktop/xh018_xenv_dir/XKIT_ROOT_DIR/xh018/cadence/v8_1/spectr
e/v8_1_2/lpmos/xh018.scs" "tm")

'("/home/vlsi/Desktop/xh018_xenv_dir/XKIT_ROOT_DIR/xh018/cadence/v8_1/spectr
e/v8_1_2/lpmos/param.scs" "3s")

'("/home/vlsi/Desktop/xh018_xenv_dir/XKIT_ROOT_DIR/xh018/cadence/v8_1/spectr
e/v8_1_2/lpmos/config.scs" "default")
)
;;;;;;;;;;;;;; variables in ADEL ;;;;;;;;;;;;;
;;;; these variables are equal --> A = set_height
desVar(   "A" 1.015 )
desVar(   "set_height" 1.015 )
;;;; these variables are equal --> tw_in = set_width
desVar(   "tw_in" 300p  )
desVar(   "set_width" 300p  )
;;;; these two are equal for SET_PULSE --> start_from_0_or_1 = ref
desVar(   "start_from_0_or_1" 1.8 )
desVar(   "ref" 1.8 )
;;;;;;;;;;;;;; variables to stimuli ;;;;;;;;;;;;
desVar(   "t_pulse" 50p  )
desVar(   "t_period" 2000p )
;;;;;;;;;;;;;; variables to stimuli ;;;;;;;;;;;;
desVar(   "t0" 300p  )

stimulusFile( ?xlate nil
```

```
"/home/vlsi/Desktop/wael_validation/c17/stimulus/pat_1_graphical_stimuli.scs
")
analysis('tran ?stop "2000p"  )
envOption(
    'analysisOrder  list("tran")
)
; first run
temp( 27 )
run()
selectResult( 'tran )
plot(getData("n_N23"))
out=outfile("/home/vlsi/Desktop/wael_validation/c17/output/pat1_from__g73__4
319_A_to__N23_test1_outputs_waveform.out" "w" )
ocnPrint(?output out ?numberNotation 'engineering VT("n_N23"))
inv_wave = getData( "n_N23")
fprintf(out_f " 1 4 1 ")
;; row : 4 ;;;; test : 2 ;;

simulator( 'spectre )
design(  "/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schemati
c/netlist/netlist")
resultsDir( "/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schem
atic" )
modelFile(

'("/home/vlsi/Desktop/xh018_xenv_dir/XKIT_ROOT_DIR/xh018/cadence/v8_1/spectr
e/v8_1_2/lpmos/xh018.scs" "tm")

'("/home/vlsi/Desktop/xh018_xenv_dir/XKIT_ROOT_DIR/xh018/cadence/v8_1/spectr
e/v8_1_2/lpmos/param.scs" "3s")

'("/home/vlsi/Desktop/xh018_xenv_dir/XKIT_ROOT_DIR/xh018/cadence/v8_1/spectr
e/v8_1_2/lpmos/config.scs" "default")
)
;;;;;;;;;;;;;; variables in ADEL ;;;;;;;;;;;;;
;;;; these variables are equal --> A = set_height
desVar(   "A" 1.8 )
desVar(   "set_height" 1.8 )
;;;; these variables are equal --> tw_in = set_width
desVar(   "tw_in" 85p  )
desVar(   "set_width" 85p  )
;;;; these two are equal for SET_PULSE --> start_from_0_or_1 = ref
desVar(   "start_from_0_or_1" 1.8 )
desVar(   "ref" 1.8 )
;;;;;;;;;;;;;; variables to stimuli ;;;;;;;;;;;;
desVar(   "t_pulse" 50p  )
desVar(   "t_period" 2000p )
;;;;;;;;;;;;;; variables to stimuli ;;;;;;;;;;;;
desVar(   "t0" 300p  )

stimulusFile( ?xlate nil
"/home/vlsi/Desktop/wael_validation/c17/stimulus/pat_1_graphical_stimuli.scs
")
analysis('tran ?stop "2000p"  )
envOption(
    'analysisOrder  list("tran")
)
; first run
temp( 27 )
run()
selectResult( 'tran )
plot(getData("n_N23"))
```

```
out=outfile("/home/vlsi/Desktop/wael_validation/c17/output/pat1_from__g73__4
319_A_to__N23_test2_outputs_waveform.out" "w" )
ocnPrint(?output out ?numberNotation 'engineering VT("n_N23"))
inv_wave = getData( "n_N23")
fprintf(out_f " 1 4 2 ")
sh("cp -rf
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist_backup
/home/vlsi/Desktop/wael_validation/simulation/c17/spectre/schematic/netlist/
netlist")

;-------------------------------------------------------;
```