Identification of Soft-Error at Gate Level

Ghaith Bany Hamad

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical & Computer Engineering)

at

Concordia University

Montréal, Québec, Canada

April 2011

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Ghaith Mohammad Bany Hamad

Entitled: "Identification of Soft-Error at Gate Level"

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. S. Williamson

_____ Examiner, External
Dr. L. Wang (CIISE) To the Program

_____ Examiner
Dr. G. Cowan

_____ Supervisor
Dr. Y. Savaria

_____ Supervisor
Dr. O. Ait Mohamed

Approved by: _____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

_____20_____ _____
Dr. Robin A. L. Drew
Dean, Faculty of Engineering and
Computer Science

ABSTRACT

Identification of Soft-Error at Gate Level

Ghaith Bany Hamad

Due to shrinking feature size and significant reduction in noise margins, as we are moving into very deep sub-micron technology, circuits have become more susceptible to manufacturing defects, noise-related transient faults and interference from radiation. Traditionally, soft errors have been a much greater concern in memories than in logic circuits. However, due to technology scaling, logic circuits have become equally susceptible to soft errors. Moreover, enhanced usage of commercial off the shelf (COTS) electronic components for avionics has also increased the importance of analyzing soft errors in hardware circuits. Conventionally, understanding soft error glitches requires circuit level modeling, which requires information available only at late stages in the design flow. Instead of this approach some researchers have produced modeling techniques using Reduced Order Binary Decision Diagrams (ROBDD) and Algebraic Decision Diagrams (ADD), which does allow analyzing soft error at an earlier stage in design flow. In this thesis, a new methodology for modeling soft errors glitch propagation path using Multiway Decision Graphs is introduced. This modeling technique is applicable on both combinational and asynchronous circuits. The proposed glitch propagation path modeling technique jointly takes care of logical and electrical masking. Our methodology involves new ways of injecting glitches including glitch injection in feedback paths of asynchronous circuits. This work presents a complete framework to exhaustively provide all the possible sequences of signals that lead to the possibility of glitch propagation to the primary output in combinational and asynchronous circuits. In addition, a new tool is developed based on the proposed methodology called Soft Error Glitch-Propagating Path Finder (*SEGP-Finder*) to automate the identification

of these sequences of signals. This work helps designers identify the vulnerable circuit paths at the logic abstraction level. Also, this methodology allows designers to apply radiation tolerance techniques on reduced sets of possibilities. By applying our methodology on different combinational and asynchronous circuits an improvement in terms of possible-fault injection vectors is observed. As an example, approximately 8% of all the possible input vectors and sequences is required for obtaining exhaustive glitch propagation path identification in a representative implementation of a bundled data asynchronous circuit. To the best of our knowledge, this is the first time MDG based decision diagram based soft error identification approach is proposed for combinational and asynchronous circuits.

ACKNOWLEDGEMENTS

It has been an amazing experience to accomplish my Master's thesis in the Hardware Verification Group (HVG) at Concordia. It certainly would not have happened without the support and guidance of several people to whom I owe a great deal.

First of all, I would like to thank my supervisor, Dr. Otmane Ait Mohamed. It is he who offered me the opportunity to join the group. He was fully supportive, understanding, involved and present during all the phases of my research. I have learned many things from him in regard to research, academia, and life in general.

Secondly, I sincerely thank Prof. Yvon Savaria, for co-supervising my research work. Also, I would like to thank Dr. Rafay Hasan, this thesis would not have been possible without his guidance, his expert advice, his support and encouragements. He introduced me to the topic of this thesis and guided me in the right direction.

Next, let me thank all the members of HVG for their help and encouragement. Their friendship brought me a warm environment in the lab. Especially, I thank the two most helpful people, Naeem Abbasi and Zaid Al Bayati.

Last but not least, I thank my family for their constant moral support and their prayers. They are the people who are closest to me and suffered most for my higher study abroad. Their support was invaluable in completing this thesis.

To my parents, Brothers; Dr.Laith , Eng.Saif, Dr.Qais and Dr.Yazen, My sisters.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ADD	Algebraic Decision Diagram
AQX	Affected by aggressor-to-Quiet-line crosstalk
ASM	Abstract State Machine
BDD	Binary Decision Diagram
BD	Bundle Data
DAG	Directed Acyclic Graph
DI	Delay Insensitive
FOL	First Order Logic
GALS	Globally Asynchronous Locally Synchronous
ITRS	International Technology Roadmap for Semiconductors
MDG	Multiway Decision Graph
ROBDD	Reduced Order Binary Decision Diagram
SEE	Single Event Effects
SET	Single Event Transit
SEU	Single Event Upset
SER	Soft Error Rate
SRAM	Static Random-Access Memory

Chapter 1

Introduction

Digital designs have been growing fast in size and complexity over the past four decades. As this complexity grows, reliability is becoming an increasingly major concern for designers especially for mission critical systems such as avionic, medical and banking applications. Fault tolerant design methods have been gaining increased importance to provide more reliable designs. To allow the efficient design of a system that can tolerate faults, a first natural step includes understanding the source of induced errors, and most importantly, their analysis and modeling for the purpose of guiding the design process.

A fault is an incorrect logic behavior that result from some physical defect, imperfection, or flaw in the hardware or software part of the system. According to their source or duration, faults classified as permanent, transient or intermittent. Reliability issues in modern deep sub-micron technologies have aggravated because designs implemented with scaled technologies have become more venerable to disturbances induced by crosstalk and soft-errors. Therefore, there is a growing need for understanding the effect of soft errors in digital design at an early stage in the design flow. This work is an attempt in this direction.

1.1 Soft Error

Soft errors can be defined as circuit errors caused due to excess charge carriers induced primarily by external radiation, e.g., cosmic rays or alpha particles. If these radiation events cause a charge generation large enough to perturb the logic value on the output of a gate, a single-event transient (SET) is generated. If a SET is propagated and latched into a memory element then it is called single event upset (SEU).

Soft errors typically affect logic circuits in various ways. These faults can cause an error in the system by changing the internal state, even though they last only for a short time. The rate at which the soft error occurs is called Soft Error Rate (SER). It is usually measured in FIT (Failure In Time), which is the number of failures per 10^9 device hours. Researchers have shown that SER in logic is posing a threat now [57] and will increase in occurrences rate by orders of magnitude within the next few years [39]. As the technology is shrinking the possibility of occurrence of soft errors in combinational circuit is getting as high as the possibility in SRAMs [2].

It is well known that alpha particles cause soft errors. Different techniques have been proposed to reduce the soft error rate due to alpha particles such as: reducing the number of alpha particles emitted by the package; coating the chip surface with a film that blocks alpha particle irradiation; and better design of the memory device to make it less sensitive to alpha-induced soft errors. However, even when reducing or attempting to eliminate alpha particle, soft errors are still there. Recently, it has been found that cosmic ray neutrons also cause soft errors even at ground level [43]. Approximately 95% of the particles capable of causing soft errors are energetic neutrons [56]. Neutrons are uncharged and cannot disturb a circuit on their own, but the notably undergo neutron capture by the nucleus of atoms in a chip. This process may result in the production of charged secondaries, such as alpha particles and oxygen nuclei, which can then cause soft errors.

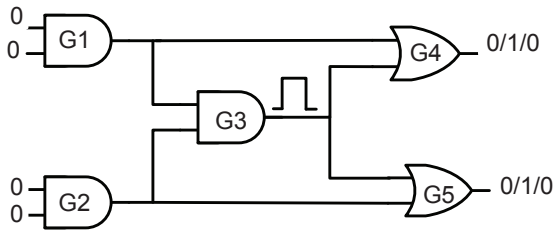

Figure 1.1: SET effects.

Figure 1.1 reports an example of SET: the circuit is primary inputs are set to 0; thus the expected output value is 0 on both G4 and G5 outputs. When G3 is struck by a particle with sufficient energy, its output switches to 1, for a duration long enough so that it may propagate through to the primary outputs. As a result, we observe a transition on both G4 and G5, whose outputs are set to 1. As soon as the SET effects disappear, the outputs switch back to the expected value.

In combinational logic there are three masking effects (logical, electrical and latching-window masking) that prevent the soft fault glitch from causing an error. Modeling soft faults in combinational logic is always combined with modeling these three masking effects. In the following section we discuss in more detail the importance of modeling and detecting soft errors in combinational and asynchronous circuits and the main challenges facing modeling the soft error in these circuits.

1.2 Problem Formulation

1.2.1 The Problem in Combinational Circuits

A digital circuit is defined as combinational if its steady-state output is completely determined by the present inputs as shown in Figure 1.2. A combinational circuit consists of input variables, logic gates and output variables. The logic gates accept signals from the inputs and generate signals to the outputs. This process transforms

binary information from the given input data to a required output data. Combinational circuits are employed extensively in the design of digital systems. Most digital systems are made up of standard combinational blocks, such as adders, multipliers, decoders etc. They perform specific digital functions commonly needed in the design of digital systems. Some combinational circuits, such as decoders, encoders and multiplexers, can be used to control other devices such as decoder circuits [15, 16], three-state buffers [17], register circuits [17], bus circuits, read / write memory operations [18] and others. Decoder and other combinational circuits can be used in mobile system, wireless networks and can be applied for other related communication systems [19, 20]. Decoder logic is essential for control units and memories [21].

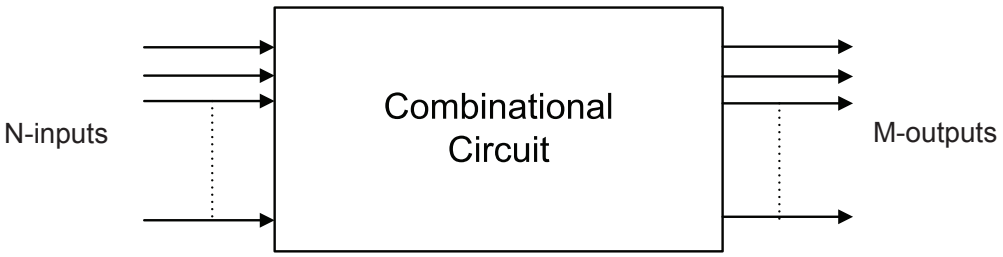

Figure 1.2: A popular representation of combinational circuits.

Traditionally, soft errors have been of greater concern in memories than in logic circuits, because of the small cell size of memories. In contrast to this, three factors prevented logic from becoming more susceptible to soft errors:

1. Logical masking - to be latched, a SET needs to be on the sensitized path from the location where it originates to the latch;

2. Electrical masking - a SET needs to create a pulse that has a duration and amplitude large enough to reach the latches. Due to the electrical properties of the gates the pulse (glitch) passes through, it can be attenuated and even completely masked before it reaches the latch.

3. Latching-window masking- if the pulse reaches the latch and appears at its input "on time", depending on its amplitude and duration, it can be latched.

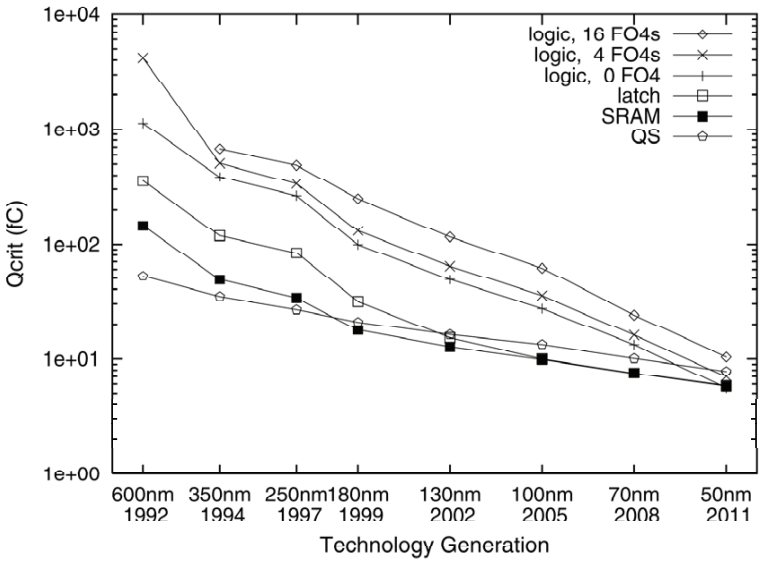

Figure 1.3: Critical charge for SRAM/latch/logic [8].

However, as technology continues to scale, logic circuits are becoming much more susceptible to soft errors [8] as shown in Figure 1.3. This figure shows a decreasing trend in Q_{CRIT}. Where Q_{CRIT} is defined as the minimum amount of induced charge required at a circuit node to cause a voltage pulse to propagate from that node to the output and be of sufficient duration and magnitude to be reliably latched. A higher Q_{CRIT} means fewer soft errors. Smaller feature sizes and lower voltage levels allow lower energy particles to cause SETs. Figure 1.3 shows this trend, where Q_{CRIT} of logic circuits continues to reduce with scaling in technology. Hence the logic circuits in sub-100 nm era have become vulnerable to soft errors even at terrestrial altitudes [38]. Therefore, soft error failure rates in combinational logic are expected to become very important in the future and it may even exceed soft error rates in memories [22].

1.2.2 The Problem in Asynchronous Circuits

A combinational circuit does not store any data for the future time. In actual implementation, these circuits contain logic gates without feedback signals. An asynchronous circuit, on the other hand, has combinational feedback which can store signal states. Its output, therefore, depends on both primary inputs and internal states. The internal states, in turn, may depend upon previous primary inputs.

Asynchronous circuits are becoming more prevalent. Most commonly they occur in the interfaces and the glue logic that binds the components of a system. According to ITRS 2009, 25% of the global signals in integrated circuits will be asynchronous handshakes by 2015 [5]. Asynchronous logic is adaptable to delay variations and components designed to function asynchronously can be more easily composed. Some of the often mentioned advantages of asynchronous circuits are speed, low energy dissipation, modular design, immunity to metastable behavior, freedom from clock skew, and low susceptibility to electromagnetic interference [23]. Figure 1.4 shows the difference between asynchronous and synchronous feedback. In synchronous circuits:

- Synchronous feedback must wait for the clock.
- Always behaves as described in its state table.
- Input signals must not change when the clock does.

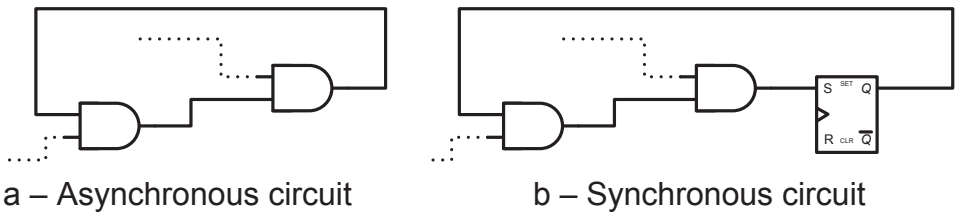

Figure 1.4: Illustrates synchronous and asynchronous feedback.

While in asynchronous circuits:

- Asynchronous feedback comes immediately with only gate delays.
- Inputs can come at any time.
- Circuits may not behave as described in the state table.

The design, verification, and testing of asynchronous circuits are complex problems. When modeling asynchronous circuits, certain requirements have to be taken into consideration due to the special nature of those circuits. These requirements are:

1. The initial values of inputs are governed by the relevant asynchronous protocols.
2. Asynchronous circuits have feedback paths from the outputs to the inputs. In the verification or the simulation of asynchronous circuits, the conditions imposed by the protocol on these paths must be considered.

1.3 Thesis Contributions

In most of the previous work, electrical masking is analyzed through simulation [2], while logical masking is analyzed by path tracing [1, 2, 3]. In comparison to [1, 2, 3, 4], where latching-window, electrical and logical masking are analyzed separately and assumed independent, our approach provides a unified treatment of electrical and logical masking, while including their joint dependency on input patterns and circuit topology.

In our work, by using Multiway Decision Graphs (MDGs), this information is instead implicitly included inside the decision diagram, and therefore allows for efficient concurrent computation of output error susceptibility caused by hits on various internal nodes.

In this work, we propose a new way to verify the glitch propagation by performing invariant checking. This checking includes running reachability analysis and checking the glitch propagation at each reachable state.

Because asynchronous circuits are becoming increasingly common due to their introduction as an interfacing mechanism in cross clock domain communication, it is extremely valuable to develop a technique that deals with modeling soft error glitch propagation in asynchronous circuits at a higher abstraction level. In terms of reviews of related work, proposed methodology and discussions, we believe our contribution can be specified as:

- Initially this thesis proposes a new methodology to identify the effect of logical masking of soft-faults and glitch propagation in both combinational and asynchronous circuits using MDG. As yet, such techniques have not been explored for soft error modeling or for analysis of glitch propagation in asynchronous circuits.

- This work elaborated on a way around to keep the functionality of the asynchronous systems while injecting glitches in feedback paths of asynchronous circuits.

- This work provides a complete framework for modeling soft faults and glitch propagation as well as a methodology to introduce them into asynchronous and combinational circuits.

- We extended the proposed methodology to jointly capture the effects of both electrical and logical masking of the Soft-Error glitch propagation for asynchronous and combinational circuits.

- A new tool is developed to automate the proposed methodologies called soft error glitch propagation finder (*SEGP-Finder*).

1.4 Thesis Outline

The rest of the thesis is organized as follows:

- In Chapter 2, we present some of the related works in the areas of Soft-Error, the structure of Multiway Decision Graph (MDG), the MDG tool followed by the MDG model checking approach and asynchronous circuits and its importance in modern deep sub-micron technologies.

- In Chapter 3, we discuss our proposed methodology for the identification of soft-error glitches in both combinational and asynchronous circuits and provide step by step description of the methodology. In this chapter, the methodology considers only logical masking.

- In Chapter 4, we discuss our proposed methodology for modeling the effect of both electrical and logical masking in combinational and asynchronous circuits.

- In Chapter 5, we explain the automation of the proposed methodology and the mechanics of the tool *SEGP-Finder*. This chapter also contain the experimental results showing the validation of our approach.

- We conclude the thesis by summarizing our work. Also we provide some future research directions in Chapter 6.

Chapter 2

Preliminaries and Related Work

In this chapter, we provide some background information necessary to understand the remaining chapters along with some related works. We start by providing some related research work in modeling soft errors. We then describe the underlying formal logic of MDG, the *Abstract State Machine* (ASM) and the MDG structure. Finally, we provide an introduction to the MDG tool, the MDG model checker, and the invariant specifications.

2.1 Soft Error Analysis and Modeling

Faulty systems (buggy in digital systems) can be very dangerous and very expensive; especially system that have safety critical applications such as Magnetic Resonance Imaging (MRI) machines, space shuttles, microprocessors and so on. This increases the need for fault diagnosis and fault-tolerance-driven design methodologies [9].

Due to this need, a lot of research work has been done on analyzing and modeling the effect of soft-errors [3, 32-35]. One of the earliest approaches was to inject the fault at a certain internal node of the design and then run simulate the circuit for different input sequences.

In most of the previous work, modeling the electrical and logical masking is

done separately such as [2, 31]. In [2], the analysis of electrical masking for each path is performed within HSPICE simulator, and logical masking is analyzed for each input vector and each path separately, by flipping the logic value of each node.

Using a mathematical model in order to analyze the propagation of a transient fault through a chain of combinational gates has been proposed in [31]. Their work was focused on estimating electrical masking on the sensitized path in the circuit, while logical and latching-window masking were not included.

In [36], an independent computation of the soft error masking factors; logical, electrical, and latching-window masking to find the soft-error tolerance of the circuit is proposed. Some other works focused on modeling only one of the masking effects such as in [37] where the focus is only on modeling the logical masking effect of the circuit for given gate output probabilities, without considering electrical and latching window masking.

In [10], the authors addressed the topic differently by proposing a modeling technique using Reduced Order Binary Decision Diagram (ROBDD) and Algebraic Decision Diagram (ADD), and combining the effects of logical, electrical and time masking. ADD is used for glitch modeling and Binary Decision Diagram (BDD) checks for the sensitization path. However, ROBDD suffers from state space explosion. Also this technique uses two decision diagrams, due to the limited availability of data types, to elaborate the glitch scenario.

In [30], a soft error modeling tool, FASER, is introduced that uses a modified BDD called event BDD to analyze glitch propagation. Since BDDs based techniques suffer from state space explosion problem, FASER [30] tries to resolve this issue by using partitioning. However, the proposed partitioning methodology is very constrained.

2.2 Asynchronous Circuits

In this work, we use the term "*Asynchronous*" to refer to circuits designed without clocks, also known as Self-Timed circuits, where the clock is replaced by handshaking signals. Asynchronous interface circuits are indispensable in many real-time digital systems [7, 41]. It is used to describe a variety of design styles, which use different assumptions about circuit properties. Two commonly used protocols in asynchronous circuit are:

- Bundled data protocol [9, 10, 11], which uses 'conventional' data processing elements with completion indicated by a locally generated delay model.
- Delay-insensitive (DI) data-encoded protocol [8, 12, 13, 45], where arbitrary delays through circuit elements can be accommodated. The latter style tends to yield circuits which are larger than bundled data implementations, but which are insensitive to layout and parametric variations and are thus "correct by design".

In order to verify the behavior of this kind of asynchronous circuits, understanding the protocol the asynchronous circuit is following is very important. Then the design can be verified under certain condition based on the used protocol. The next chapter discusses the implementation of our proposed technique on asynchronous circuits in more details.

These asynchronous circuits have unbounded gate delay assumption, which provides them with inherent tolerance to a broad class of delay faults. However, in certain cases, faults occurring in asynchronous circuits can have catastrophic effects due to the event ordering constraints and might cause circuit failure and can sometimes lead to deadlock [46].

The authors in [53, 54] proposed a metric, sensitive time, to evaluate the sensitivity of asynchronous circuits to transient faults and developed several harderning

techniques for Quasi delay insensitive (QDI) circuits with full duplication of circuit parts and synchronization of replicated results through C-elements [55].

The problem of soft errors in asynchronous burst-mode machines (ABMMs) has been discussed in [44], and two solutions have been proposed. The first solution is an error tolerance approach, which leverages the inherent functionality of Muller C-elements, along with a variant of duplication, to suppress all transient errors. The second solution is an error mitigation approach, which leverages a newly devised soft-error susceptibility assessment method for ABMMs, along with partial duplication, to suppress a carefully chosen subset of transient errors.

Control circuits in an asynchronous design are comprised mostly of Muller C-elements. In [47], soft error analysis of four popular CMOS implementations of the Muller C-element have been presented. The analysis shows that Safety Integrity Level (SIL) implementation has the best soft error resilience. Optimization techniques to improve the soft error resilience of C-elements are proposed in [47].

2.3 GP Modeling For Asynchronous Handshake Schemes

The **GP set** is the set of conditions that allows a glitch to propagate, in different logic gates and Muller C-elements [28]. Figure 2.1 shows the GP sets for the NOT, AND, OR, and Muller C-elements. Using the GP sets in modeling glitches is initially proposed for modeling crosstalk glitch in asynchronous circuits which is the work done in [49]. In the following we explain this work to better understand of the theory behind the GP sets. In the next chapter, we explain how the GP sets can be used to model the soft error glitch as well.

In the related modeling framework, an aggressor line (AL) is a signal line which performs a transition from logic level logic 0 to logic 1 (or logic 1 to logic 0), denoted by T (or T'). T inflicts a crosstalk glitch G (G') on the victim line (VL). Glitch on

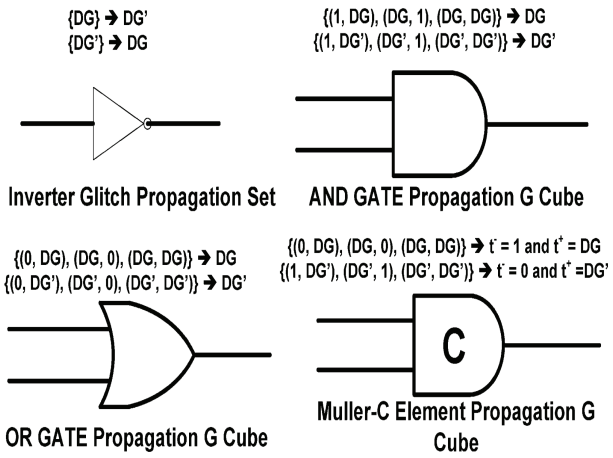

Figure 2.1: GP sets for NOT, AND, OR, and Muller C-element [49].

a particular VL, for example A, due to T (T') in AL is represented as $G_A(G_A)$. VL returns to its stable state after a bounded delay, $\triangle t_G$. In the context of asynchronous handshake schemes in globally asynchronous locally synchronous (GALS) affected by aggressor-to-quiet-line crosstalk (AQX), VL can glitch (i.e., G or G') only if VL and AL are at the same logic level before T (T'). Typically, channels linking two mutually asynchronous communicating modules are vulnerable to AQX because of the physical length of the required interconnections.

AL_in (VL_in) and AL_out (VL_out), respectively, denote whether the AL (VL) signal is an inbound or outbound signal to the module. AL_I (VL_I) and AL_O (VL_O) represent the corresponding input and output signals which are transiting (glitching) toward T (G), respectively. Opposite transitions (glitches) are represented as their complements (AL_I', AL_O', VL_I', VL_O'). This notation is summarized in Table 2.1.

DG (DG') is a symbol used in our modeling method to represent composite logic values of the form v/vg, where v and vg are values of the same signal in the glitch-free and the erroneous circuit, respectively. The composite logic values that represent possible error propagation 1/G' and 0/G are denoted by the symbols DG and DG', respectively. The terminology was adopted as an extension of that used in the

Table 2.1: Summary of AL and VL notations.

	Wire Name	Direction of the signal on the wire	Logic Level '1' on the signal wire
Aggressor	AL	AL_in, AL_out	AL_I, AL_O
Victim	VL	VL_in, VL_out	VL_I, VL_O

D-algorithm [50, 51] at the foundation of testing methodologies.

In this work, we use the GP sets to model the soft error glitch. There are some differences between soft error and crosstalk glitch. While crosstalk glitch can occur at the primary input, soft error glitch only happens at the internal, and if that internal node is primary input for combinational circuits then it can occur at the primary input. Also, G, G' is reactive to T, T' in crosstalk glitch modeling, while G, G' can occur irrespective of T, T' in soft error. In the next chapter, we discuss modeling soft error glitch using GP sets in detail.

2.4 Multiway Decision Graph (MDG)

2.4.1 Abstract State Machine (ASM)

MDG is an extension of Binary Decision Diagram (BDD) in the sense that it represents and manipulates a subset of first-order logic formulae suitable for large data path circuits. One of the advantages of MDG over the other decision graph is that a data value can be represented by a single variable of abstract sort, rather than by concrete Boolean variables, and a data operation can be represented by an uninterpreted function symbol. MDG and ROBDD are alike in the sense that both require a fixed order of node labels along all paths. In ROBDD the entire variable are Boolean. But in MDG every signal/variable must belong to an appropriate sort, also a type definition must be provided for all functions. The MDG operations and verification procedures are packaged as a set of tools and implemented in *prolog*

[14] providing facilities for hardware verification: invariant checking, equivalence checking and model checking. This work utilizes MDG to identify soft error glitch propagation in asynchronous handshakes, which are extensively used for communication among CDC modules.

In MDG, a state machine is described using finite sets of input, state and output variables, which are pair-wise disjoint. The behavior of a state machine is defined by its transition/output relations including a set of reset states. An abstract description of the state machine, called *Abstract State Machine* [24], is obtained by letting some data input, state or output variables be of an abstract sort, and the datapath operations be uninterpreted function symbols. As ROBDDs are used to represent sets of states and transition/output relations for finite state machines (FSM), MDGs are used to compactly encode sets of (abstract) states and transition/output relations for ASMs. This technique replaces the *implicit enumeration technique* [25] with the *implicit abstract enumeration* [11].

2.4.2 Structure

MDGs are graph representation of a class of quantifier-free and negation-free first order many sorted formulae. It subsumes the class of Bryant's (ROBDDs) [6] while accommodating abstract data and Uninterpreted Function symbols. MDG can be seen as a *Directed Acyclic Graph* (DAG) with one root, whose leaves are labeled by formulae of the logic True (T) [11], such that:

1. Every leaf node is labeled by the formula T, except if the graph G has a single node, which may be labeled T or F.

2. The internal nodes are labeled by terms, and the edges issuing from an internal node v are labeled by terms of the same sort as the label of v.

Following is an example: Let graph G represent Boolean formula $(\neg x \wedge F0) \vee (x \wedge F1)$ Where, *F0* and *F1* are the Boolean formulas represented by the sub-graphs *G0* and *G1* respectively. In many sorted first-order logic the graph G can be viewed as representing a formula: $((x = 0) \wedge F0) \vee (x = 1) \wedge F1))$.

Three possible generalizations of G and the corresponding formulas are shown in Figure 2.2. *F0*, *F1* and *F2* are first-order formulas represented by the sub-graphs *G0*, *G1* and *G2* respectively:

1. From G to G': $x \in \{0, 1\} \longrightarrow x \in \{0, 2, 3\}$, and graph G' represents the formula
   $((x = 0) \wedge F0) \vee ((x = 2) \wedge F1) \vee ((x = 3) \wedge F2)$.

2. From G to G'': $x \in \{0, 1\} \longrightarrow x \in \{a, y, f(a, y)\}$, and graph G'' represents the formula
   $((x = a) \wedge F0) \vee ((x = y) \wedge F1) \vee ((x = f(a, y)) \wedge F2)$.

3. From G to G''': $x \in \{0, 1\} \longrightarrow g(x) \in \{0, 2, 3\}$, and graph G''' represents the formula
   $((g(x) = 0) \wedge F0) \vee ((g(x) = 2) \wedge F1) \vee ((g(x) = 3) \wedge F2)$.

The above generalized decision graph G', G'' and G''' are examples of *Multiway Decision Graphs* (MDGs). As in ordinary many-sorted *First Order Logic* (FOL), terms are made out of sorts, constants, variables, and function symbols. Two kinds of sorts are distinguished: *concrete* and *abstract*. *Concrete* sort is equipped with finite enumerations, lists of individual constants. *Concrete* sorts are used to represent control signals. *Abstract* sort has no enumeration available. A signal of an *abstract* sort represents a data signal.

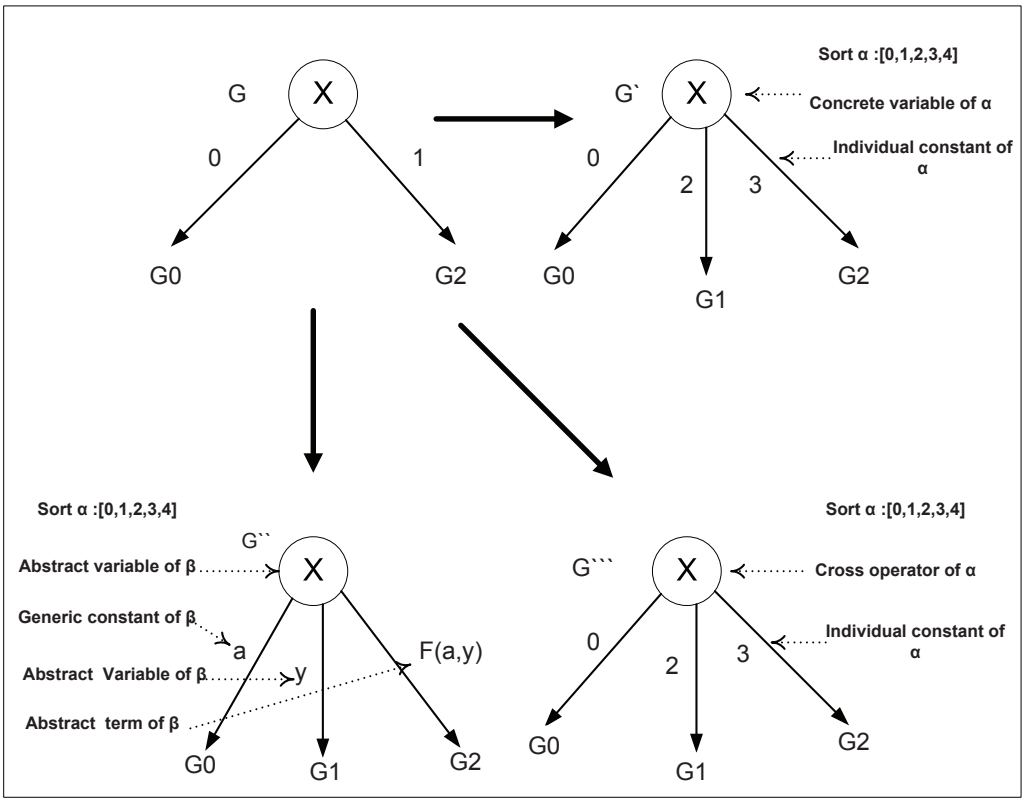

Figure 2.2: BBDs to MDGs

2.4.3 The MDG-Tool sets

The MDG-tool [14] is a well known academic tool. It supports invariant checking, sequential equivalence checking, and model checking. The MDG tool uses a *prolog-style* hardware description language called the MDG-HDL [11]. MDG-HDL supports structural, behavioral and mixed styles of coding. A structural specification is usually a netlist of components connected by signals. A behavioral description consists of a tabular representation of the transition and output relations in the form of a truth table.

The first step in the verification is to describe the design specifications and implementations using MDG-HDL, as shown in Figure 2.3. The following input files are needed to describe the design in MDG:

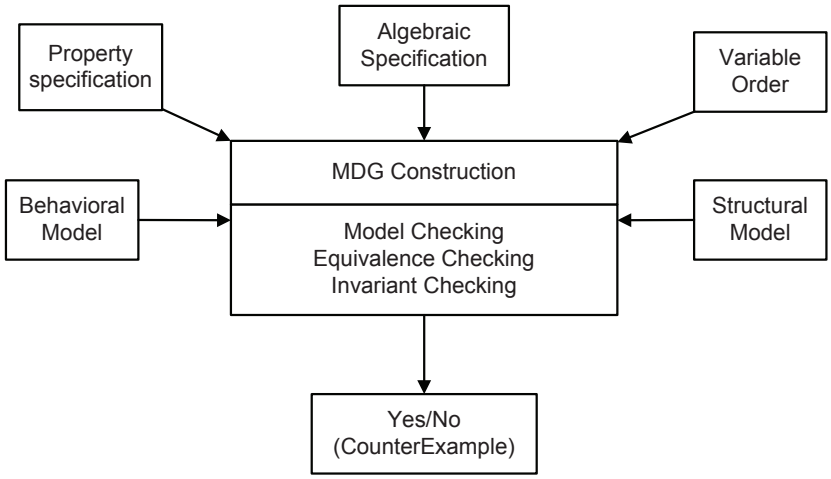

Figure 2.3: The Structure of the MDGs-tool.

- The *algebraic specification* file defines sorts, function types and generic constants used in hardware descriptions. And if necessary, it also includes the rewrite rules which partially interpret the otherwise uninterpreted function symbols.

- The *symbol order* file provides the *custom* (user-defined) *symbol order* for all the variables and cross-operators which would appear in MDGs.

- The circuit description file declares signals and their sort assignments, component network, outputs and the mapping between state variables and next state variables. There is a special component construct *table* which is the tabular representation for behavioral descriptions. For sequential circuits, we also give the set of initial states and the transition/output relation partitioning strategy.

- The *invariant specification* file defines the invariant to be checked during the reachability analysis.

2.4.4 Using MDG for Reachability Analysis

The presence of uninterpreted symbols in the logic means that we must distinguish between a state machine M and its abstract description D in the logic. This is called Abstract State Machine, a state machine given an abstract description in terms of *Directed Formulas* DFs, or equivalently MDGs, as defined in [11, 58].

Definition 1. An abstract description of a state machine M is a tuple $D = (X, Y, Z, Y', IS, Tr, Or)$, where:

- X : finite set of input variables,
- Y : finite set of state variables,
- Z : finite set of output variables,
- IS : MDG of type $U0 \longrightarrow Y$, where U_0 is a set of disjoint abstract variables, IS is the abstract description of the set of initial states,
- Tr : MDG of type $X \cup Y \longrightarrow Y'$. Tr is the abstract description of the transition relation,
- Or : MDG of type $X \cup Y \longrightarrow Z$. Or is the abstract description of the output relation.

Algorithm 1 shows how the analysis of the reachable states of M is performed based on the abstract description D. The algorithm is initialized by the construction of the initial MDG structure in Lines 1-3. In line 4-10, within the while loop, the set of reachable states is computed. When the *frontier* set (Q) becomes empty (F), the *while* loop terminates. A new MDG input is produced in line 6. In line 7, next state is computed by the function *next_state* using the RelP operation, that takes the MDGs representing the set of inputs, the current state and the transition relation as assignment, respectively. In line 8, The function $frontier$, computes

the set difference using the PbyS operation, that approximates the set difference between the newly reachable state in the current iteration from the reachable state in the first iteration. Finally, the set of all reachable states so far is computed, in line 9.

Algorithm 1 MDG Reachability Analysis

```
1: R := IS;
2: Q := IS;
3: i := 0;
4: while Q ≠ F do do
5:    i := i + 1;
6:    IN := new_inputs(i); -Produce new inputs
7:    NS := next_states(IN,Q, Tr); Compute next state
8:    Q := frontier(NS,R); Set difference
9:    R := union(R,Q); Merge with set of states reached previously
10: end while
```

2.4.5 Invariant Specification in MDG

An invariant file specifies the invariant condition to be checked during reachability analysis [27]. An invariant condition can be specified by a combinational circuit whose output signals are named by the variables that occur in the condition. By convention, an assignment of values to those variables satisfies the condition iff the outputs of the combinational circuit take those values for some assignment of values to the inputs. An MDG representing the invariant is obtained from the MDG representing the functionality of the combinational circuit by existentially quantifying the concrete inputs. The variables representing abstract inputs are left in the graph as implicitly quantified secondary variables [26].

For example, for the equivalence checking of two ASMs, we need to specify the equality of two corresponding signals as an invariant. This is expressed by the simple fork as shown in Figure 2.4 (a). The fork may yield different MDGs depending on the sort of the signals. If u, x and y are of the Boolean sort, then

u is existentially quantified and we get the MDG as shown in Figure 2.4 (b) which simply represents $x = y$. If x and y are of an abstract sort, then we get an MDG as shown in Figure 2.4 (c) which represents the formula $(x = u) \wedge (y = u)$. Taking the secondary variable u to be existentially quantified, the invariant is $\exists u((x = u) \wedge (y = u))$, which is logically equivalent to $x = y$. This combinational circuit is described completely in an invariant specification file, including the following predicates: *signal/2, component/2, outputs/1* and *ordercond/1*, which gives the node order for the variables and the cross-function symbols appeared in the circuit.

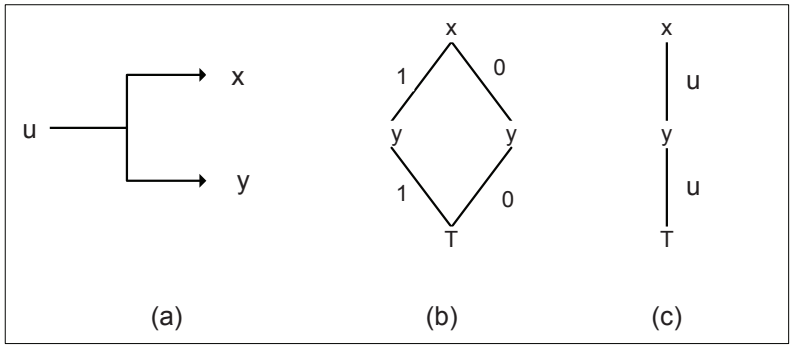

Figure 2.4: Invariant specification in MDG tool.

In this Chapter, we presented some of the basics required for better understanding the rest of chapters of this thesis. We provided the basics of Multiway Decision Graph, MDG tool, GP sets, and asynchronous circuits. We have presented some of the works related to soft error modeling in combinational and asynchronous circuits. Furthermore, we also mentioned how our work differs from the related work. In the next chapter, we describe our proposed methodology in detail.

Chapter 3

Identification of Soft-Error Glitch path: Considering Logical Masking

In this chapter, we present the proposed methodology for the identification of soft-errors in both combinational and asynchronous circuits. In this chapter, we deal with the identification of soft error and modeling *logical* masking in logic circuits. Logical masking has been explained in chapter 1. In section 3.1, we explain the main steps of the proposed methodology. Section 3.2 discusses how we use the Glitch Propagation (GP) sets to identify the soft error glitch and to model the logical masking effect. We explain the new proposed technique for glitch injection at vulnerable nodes, also the breaking and the initializing of the feedback in asynchronous circuits in Section 3.3. Section 3.4, explains how the invariant checking is used to verify the glitch propagation property. The implementation of our approach on combinational logic and asynchronous circuits is discussed in detail in section 3.5 and 3.6 respectively.

3.1 The Proposed Methodology

This section presents an overview of the proposed technique; the flow chart of our methodology is shown in Figure 3.1. This figure shows that our methodology requires structural specifications as input. The next step is to inject a glitch at the vulnerable nodes by modifying these structural specifications. The following step is to examine the possibility of glitch propagation using formal verification techniques. This methodology provides the number of nodes that are prone to glitches and generates counterexamples. These examples are the vulnerable conditions under which glitches on a certain node propagate. The following sections further elaborate on this flow chart.

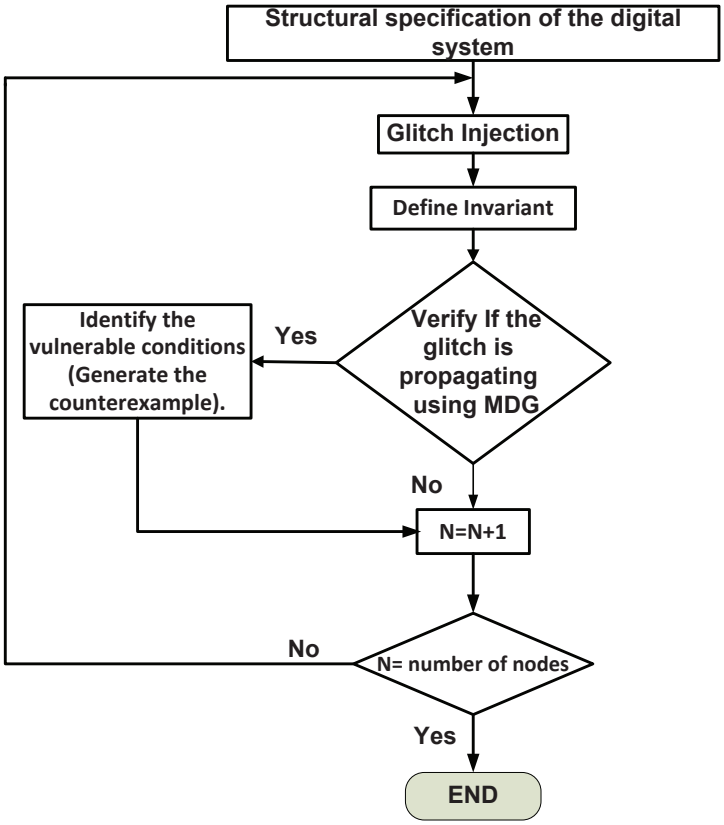

Figure 3.1: The flow chart of the proposed technique.

3.2 Glitch Propagation (GP) sets

In section 2.3, we defined the GP set as a set of conditions that allow glitches to propagate in different logic gates and Muller C-element. Also we discussed how the GP sets used to model the crosstalk glitch in asynchronous circuits. The next subsections explain how we combine the GP sets with the soft-error glitch in order to build a new truth table for different logic gates and and Muller C-element.

3.2.1 Muller C-element

Table 3.1 shows the expansion of the GP sets in the form of a truth table. Each GP set's truth table has four possible values $0, 1, G, G'$. If the glitch is from logic 0 towards logic 1 then it is represented as G and G' represents the inverse case. Composite value, DG represents that an un-affected node would have the value $'1'$ while the glitch will force this node to glitch, G'. Hence, DG represents $1/G'$. Similar explanation elaborates DG', which represents $0/G$. Table 3.1 shows that if the output of the C-element is logic $'1'$ before the occurrence of a composite logic value of DG in any or both inputs of the *C-element*, then, due to the glitch, the output of the C-element will have a composite logic value of DG after the gate propagation delay at time $t+$.

Table 3.1: The truth table for the C-element.

C-element	0		1		G		G'	
	t-	t+	t-	t+	t-	t+	t-	t+
0	0	0	0	0	0	0	0	0
	1	0	1	1	1	1	1	DG
1	0	0	0	1	0	DG'	0	0
	1	1	1	1	1	1	1	1
G	0	0	0	DG'	0	NTS	0	NTS
	1	1	1	1	1	NTS	1	NTS
G‘	0	0	0	0	0	NTS	0	NTS

3.2.2 Other Logic Gates

For all the logic gates (such as: *AND, OR, NOT, NAND, XOR etc.*) we apply the same principle used in the previous section. For example, for the *AND* gate if one of the inputs is zero then the output is zero without caring about the other inputs. So the glitch can not propagate from the *AND* gate if one of the other inputs is zero. We similarly expanded all the GP sets provided in [12] for all the logic gates and came up with similar truth tables, which are shown in Table 3.2.

Table 3.2: Truth table for the AND, OR, and NOT gates.

	0		1		DG		DG‘		
	AND	OR	AND	OR	AND	OR	AND	OR	NOT
0	0	0	0	1	0	DG	0	DG’	1
1	0	1	1	1	DG	1	DG’	1	0
DG	0	DG	DG	1	DG	DG	DG’	DG	DG
DG’	0	DG’	DG’	1	1	DG’	DG	DG’	DG’

3.3 Glitch Injection and Initialization

In order to insert glitches, we first need to translate the structural specification into an MDG model. The MDG model contains declaration of the signals and their type assignments, the component network, the outputs, and the mapping between state variables and next state variables. In this work, we used an MDG based table, which is the tabular representation for behavioral descriptions. Glitches are inserted using the library of GP sets, which is created a priori. This library contains the GP sets in the form of truth tables, with the logic values introduced in Table 3.1 and 3.2. We modeled those truth tables in MDG while keeping the information provided in the structural specifications. This method can be easily ported to any combinational logic circuit.

To inject glitches at all the possible nodes in a design, we used multiplexers, which can select between a normal mode and a glitch mode during the simulation. If a soft error needs to be analyzed at a particular node, it simply chooses the glitch mode (defined as an enumerated data type), otherwise the normal operation is carried out. Hence, the functionality of the circuit is preserved under the normal mode. After building the MDG model and injecting the glitch, the next step is to find if the glitch is able to propagate to the primary outputs.

3.4 Invariant Checking

To write an invariant, one needs to know the semantics of MDG-HDL. Since this language was devised for sequential circuits, therefore, it includes the notion of register at the end of the design. In our modeling of asynchronous circuits, these registers always appear at the output of the design and hence do not disturb the logic of the circuit. This also helps writing the invariant to check whether the glitch can propagate to the primary output of the circuit or not. This can be written in temporal logic as $AG((reg\_out = 0)||(reg\_out = 1))$, which means *"In all the reachable states, the output of the register is equal to 1 or 0"*. We can apply the same invariant to all the asynchronous circuits as long as we are using the same semantics of our defined enumerated data type. The next step is to check whether the glitch is propagating for our particular glitch injection scenario or not. This is obtained by performing a reachability analysis. This analysis checks that the invariants at all the reachable state of the circuit are exhaustively analyzed for glitch propagation to the primary output. Provided the invariant fails then counterexamples are provided which we classify as vulnerable sequences to glitch propagation conditions.

3.5 Soft Error Analysis on Combinational circuits

In this section, to illustrate the proposed technique, we apply our soft error analysis on one combinational circuit.

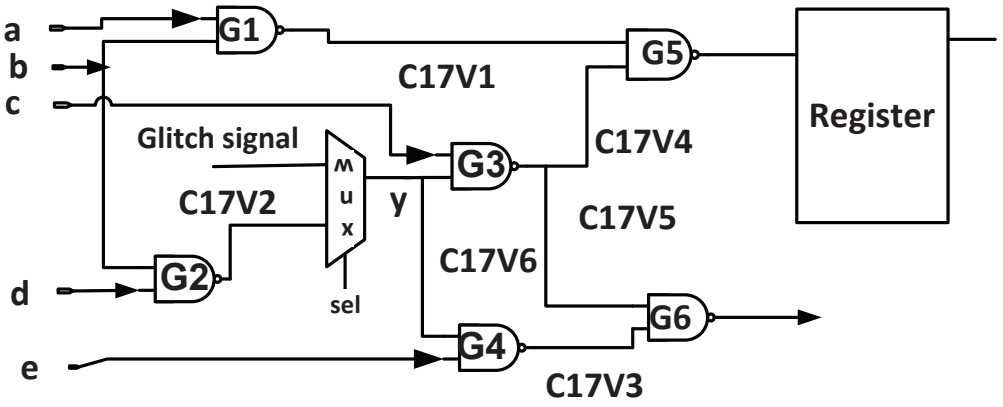

Figure 3.2: C17 with the glitch injected between G2 and G3.

Figure 3.2 shows the implementation of our technique on the *ISCAS-85* benchmark circuit C17. A glitch is introduced in the circuit using a 2:1 multiplexer at a potentially vulnerable internal node of the circuit. The glitch is inserted by having one of the inputs of the multiplexer as a glitch signal and the other input as a normal signal. Figure 3.2 shows an example implementation with a glitch inserted between two nand gates, *G2* and *G3*. A register is connected to one of the primary output where we expect glitch propagation. Then the next step is to perform the invariant checking, where MDG run reachability analysis and for each reachable state it checks the invariant on the outputs. The same invariant is used as discussed in Section 3.4. In case the invariant checking fails our proposed method provides counterexamples. One such case is shown in Figure 3.3, which shows the counterexample when the glitch is injected at node *C17V2* in Figure 3.2. In our implementation, a critical sequence is defined as the sequence of input signals that allow the glitch at an internal node to propagate to the primary outputs.

In Figure 3.3, it is shown that, starting from the initial state of the output

```
=== The Counterexample ===
-------- Assumptions --------
-
-------- Initial state --------
-
reg_output_signal = 1
-
-------- Clock cycle 1 --------
  -- The symbolic input --
-
a = 0
c = 1
select = 1
glitch_signal = dgp
-
  -- The symbolic state --
reg_output_signal = dg
-------- Clock cycle 2 --------
  -- The symbolic input --
-
  -- Symbolic Output --
flag = 1
=== End of counterexample ===
-
Generating counter example took 0.010 seconds.
```

Figure 3.3: One of the counterexamples for the C17.

register, the MDG tool provides the symbolic inputs which are the critical sequence of inputs that allows the glitch on the node *C17V2* to propagate to the primary output. The critical sequence for the previous case is when the input signal c is at logic '1', signal a at logic '0', and the select signal for the multiplexer is at logic '1'. The verification time and the time needed to generate the counterexample are between 10 and 20 millisecond. Table 3.3 shows the critical sequences for all the internal nodes *(C17V1-C17V6)*. In chapter 5, we propose a full automation of all these steps.

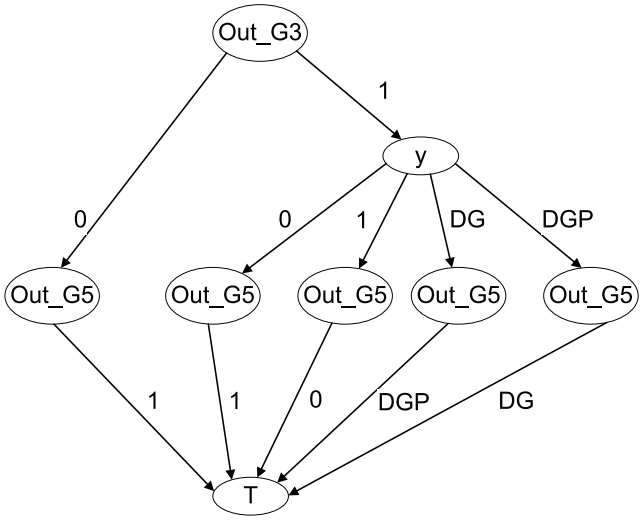

Figure 3.4: MDG decision diagram for path $G1 \longrightarrow G5$ from circuit C17.

Table 3.3: Node wise Vulnerable Conditions for C17.

Internal node	Vulnerable Conditions
C17V1	c = 0
C17V2	c = 1, a = 0
C17V3	c = 0
C17V4	a = 0, b = 0
C17V5	e = 0
C17V6	e = 1 , c = 0

Our method utilized MDG which simplifies the use of decision diagram. Previous approaches utilize different techniques for sensitization paths and glitch modeling. One such case is [10] where ADD is used for the glitch modeling and the BDD checks for the sensitization path. Instead MDG in conjunction with GP sets allows us to simply define a truth table, similar to Table 3.1 and 3.2, for each gate in the circuit that not only depicts sensitization but also encapsulates glitch modeling information. This is achievable because of enumerated data type of MDG and glitch propagation modeling approach described in [12]. By modeling the circuit in MDG using this kind of truth table, we are able to define both the glitch effect and the sensitized path using one decision diagram as shown in Figure 3.4.

Figure 3.4 shows the MDG decision diagram for path $G1 \longrightarrow G5$ of the *C17* circuit of *ISCAS'85* benchmark. In our work, out of one decision diagram we are able to model the logical masking and model the acceptable amplitude and width needed by the glitch to propagate. As an example, modeling of the path sensitization in Figure 3.4. *Out_G3* and *out_G5* in Figure 3.4 represent the output of gate 3 and 5 in Figure 3.2. It is shown that when *out_G3* is equal to *0* then the *out_G5* will be 1 irrespective of the glitch value at the other input of *G5*. To model the glitch we defined a new data type called glitch_type which can be *(0, 1, DG, DG')*. The Y signal in Figure 3.4 is an example of a glitch signal.

3.6 Soft Error Analysis on Asynchronous Circuits

As we explained in section 2.2, a digital system with asynchronous interfaces for interaction between synchronous modules is known as a globally asynchronous locally synchronous (GALS) systems. Several asynchronous interfaces have been proposed for GALS systems. Broadly, based on their handshake protocol, these interfaces can be divided into two classes, the bundled data protocol and delay-insensitive (DI) data-encoded protocol. This section provides glitch propagation analysis for

representative circuits of both protocols.

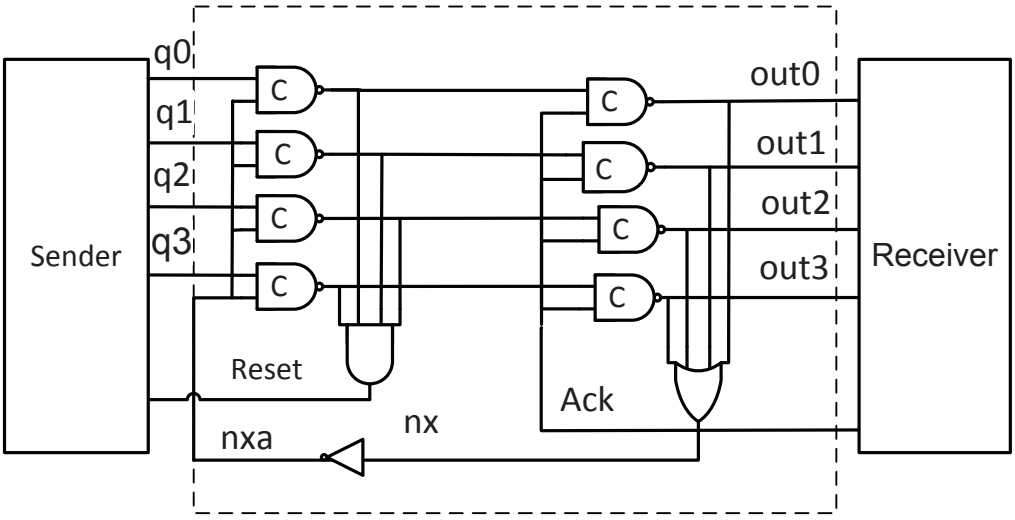

Figure 3.5: Hardware implementation of the data-encoded DI scheme.

3.6.1 Delay Insensitive (DI) Asynchronous Circuit

In the Delay-Insensitive design style there is no need for timing analysis, giving designs that operate correctly regardless of the delay in the interconnecting wires. 1-of-4 data-encoded DI circuit is shown in Figure 3.5. The *nx0*-to-*nx3* group of signals is set to high or logic "1" as an initial condition. This group of signals is represented as *nx[0−3]* in the rest of this thesis. Similarly, *q[0−3]* represents the *q0*-to-*q3* group of signals, and *out[0−3]* represents the out0-to-out3 group of signals. According to the 1-of-N DI data-encoded protocol, of which 1-of-4 is the special case shown in Figure 3.5, only one of the input lines of in*[0−3]* can go high at a given time. When any of these input signals becomes high (to transfer some data), the corresponding line in the group *nx[0−3]* is pulled down to logic 0.

The soft error analysis of this kind of the asynchronous circuits is similar to combinational circuit implementation, we added multiplexer at the internal nodes to add glitches in asynchronous circuits as well. However, overall, the implementation

of our technique to asynchronous circuits is more involved. This is mainly due to the feedback path required for initial conditions to follow the asynchronous protocols. Figure 3.6 shows the asynchronous delay insensitive (DI) circuit after implementing the proposed methodology. Here, an initialization mechanism is introduced as a combination of multiplexers, registers, and decoders. We also needed to break the feedback path in order to comply with the MDG-tool. In order to keep the functionality of the asynchronous circuit after breaking the feedback path, we added a combination of multiplexer and register. This addition is to initiate the input part of the feedback which is the *nxa* signal in Figure 3.6. By controlling the initial value of the feedback signal, we are able to keep the functionality of the circuit.

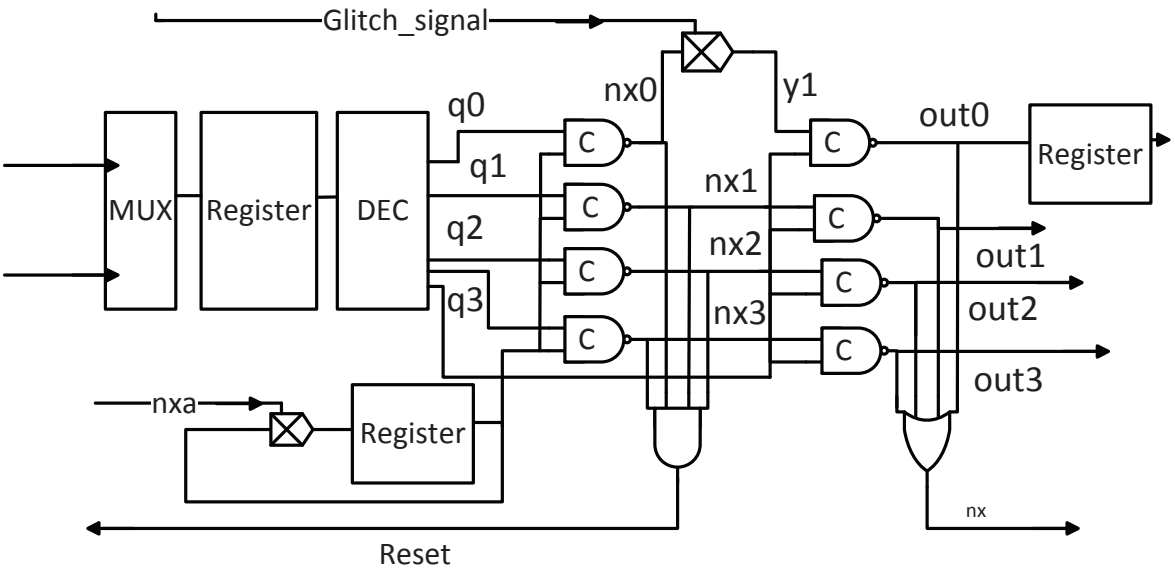

Figure 3.6: Asynchronous Delay Insensitive (DI) circuit after the implementation

3.6.2 Bundled Data Protocol based Asynchronous Circuit

The term *bundled-data* refers to a situation where the data signals use normal Boolean levels to encode information, and where separate request and acknowledge wires are bundled with the data signals. Such as the circuit in Figure 3.7.

Immediately upon activation by a switching event on *Den* it issues a request for a clock stretch *Ri+* which gets acknowledged by *Ai+*. When the clock is ensured to be and remain low, the external handshake cycle on *Rp/Ap* gets processed and subsequently the clock may resume again.

By following the same steps applied to the DI circuit, we implemented our methodology on the bundled-data-protocol based circuit. Figure 3.8 shows the asynchronous bundled data circuit after implementing our methodology. The inputs of the multiplexer are all the possible input sequences that the input can have based on the bundle data protocol [29]. After one of the initial sequences is chosen by the multiplexer, it will be stored in the register. The decoder will give a value for each input of the circuit based on the chosen initial sequence. *BDV1* to *BDV8* are the potential vulnerable internal nodes where we inserted the glitches.

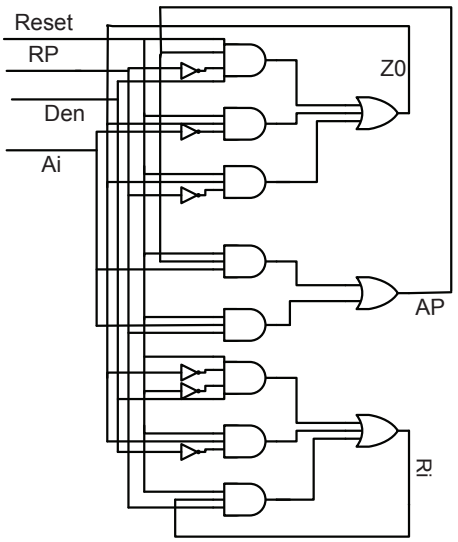

Figure 3.7: Asynchronous Bundle Data circuit without the initialization technique.

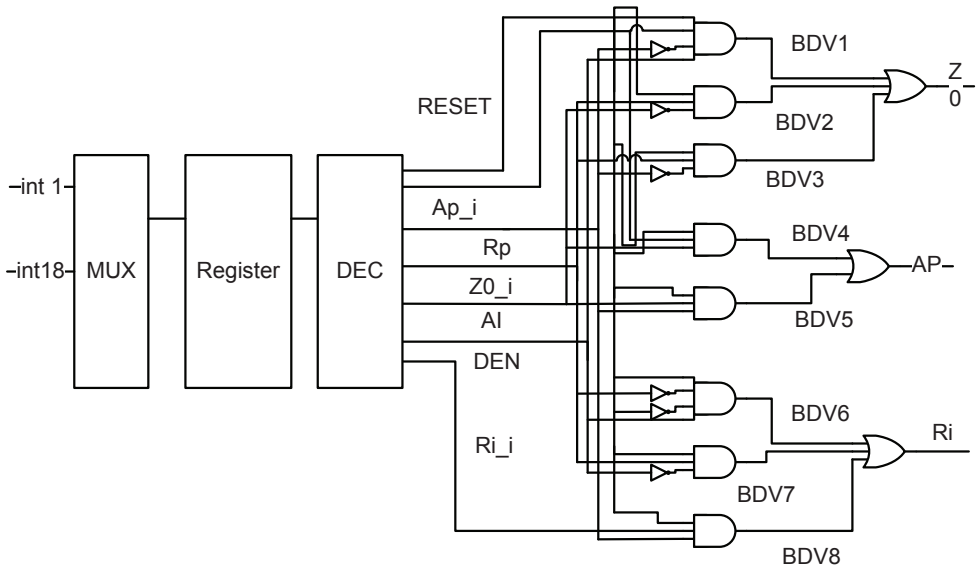

Figure 3.8: Asynchronous Bundle Data circuit with the initialization technique.

3.7 Summary

In this chapter, we presented the proposed methodology for the identification of soft-errors in both combinational and asynchronous circuits. Also we provided step by step description of our methodology with examples. We proposed a new technique for the glitch injection at vulnerable nodes and the breaking and the initializing of the feedback in asynchronous circuits. We discussed the implementation of our approach on combinational logic and asynchronous circuits. Later on in this thesis, we discuss the experimental results of the implementation of this methodology. An automation of the proposed methodology will be proposed in chapter 5. The goal of the proposed methodology of this chapter is to model the effect of logical masking in logic circuits. In the next chapter, we extend this methodology in order to model the both electrical and logical masking.

Chapter 4

Identification of Soft Error Glitch Propagation Path: Considering Electrical and Logical Masking

In the previous chapter, we explained our methodology to model the effect of logical masking in both combinational and asynchronous circuits. In this chapter, we explain how the proposed methodology in the previous chapter can be extended to model the effect of both electrical and logical masking. In Section 4.1, we give a brief description of electrical masking effect in logic circuits. Assumptions and notations required to identify the soft error glitch propagation, while considering electrical masking along with logical masking, are explained in Section 4.2. In Section 4.3, these assumptions are used to model the combined effect of electrical and logical masking utilizing the concept of GP sets. For implementing these modeling concepts over combinational and asynchronous circuits we proposed a methodology which is elaborated in Section 4.4.

4.1 Electrical Masking

As explained in chapter 1, there are three types of masking that can prevent a transient glitch in combinational logic from propagating to the primary outputs: logical masking, electrical masking, and latch window or time masking [39, 40]. Logical masking happens when one of the other inputs of a gate is in controlling state (e.g., 0 for a NAND gate) so that the transient is blocked. Latch window masking means that the arrival of the transient pulse is outside of the latching window for the sequential elements. Electrical masking happens when the voltage transient resulting from a particle strike is attenuated by subsequent logic gates because of the electrical property of logic gates [15]. This work primarily deals with the identification and the verification of soft-error glitch propagation in combinational and asynchronous circuits, therefore modeling the latching time windows masking effect is not discussed.

The possibility of glitch propagation in systems, which are subject to logical and electrical masking, depends on following factors. First the glitch magnitude at the output, which is a function of the initial amplitude of the glitch and the attenuation on the sensitized paths and the second factor is the duration of the glitch. Some notations are defined and a few assumptions are made in order to accurately model these effects, which are explained in the next section.

4.2 Assumptions and Notations

When the glitch hits the internal nodes, it requires certain threshold amplitude and duration to charge or discharge the node. Due to electrical masking the amplitude and the interval of the soft error glitch reduce as it propagates through the design. Sometimes this reduction in the amplitude and the interval prevents glitch from reaching the output. Traditionally, electrical simulations are used to understand the effects of electrical masking. In this work, this information is extracted beforehand

and utilized at logic level of abstraction. Using the results of electrical simulations it is safe to conclude that regular FO4 (fan-out of 4) gates may completely mask the glitch propagation if it is applied to four similar cascaded gates [52]. Keeping this fact in consideration, following we provide assumptions regarding the glitch characteristics as it propagates through the design to model the electrical masking.

Classification of glitches to facilitate modeling of electrical masking: As a soft error hits the internal node of the design it can create glitches of different amplitude and duration. In order to accommodate different levels of possible glitches we classify them based on the mentioned parameters. It is assumed that the glitch with the most strength (i.e both the amplitude and duration is well above the threshold value of the subsequent gates) can pass the complete depth of the combinational circuit without losing its strength. This type of glitch is given the name *G_dp4* in our analysis. The glitch with next highest amplitude and duration, which is subjected to attenuation due to electrical masking, is classified into three further categories namely *G_dp3, G_dp2, G_dp1.* The numeric value at the end of each glitch is attributed to how many gates these glitches may propagate in the worst case. For simplicity of analysis we further assumed that if *G_dp3* occurs and passes to the first level of gates, the attenuated strength of the glitch can be represented by *G_dp2.* Similarly *G_dp1* may represent the occurrence of glitch of strength *G_dp1* at that node or attenuated strengths of *G_dp3* and *G_dp2* after passing through 2 gates and 1 gate respectively as shown in Figure 4.1. The assumption that the glitch travels a depth of approximately three gates is inspired by the modeling provided in [52]. Formally the classification of glitches is provided below, and further elaborated in Figure 4.2.

1. *G_dp4*: If the amplitude is within ($a \geq a1$), and its time duration is within the interval ($\triangle t2 < d < \triangle t1$) as shown in Figure 4.2, then the glitch propagates through all the cascaded gates without loosing its strength. This is the worst

case where the glitch has the highest amplitude and duration.

2. *G_dp3*: If the amplitude is within ($a3 < a < a2$), and its time duration is within the interval ($\triangle t3 < d < \triangle t2$) as shown in Figure 4.2, then the glitch propagates through two cascaded gates.

3. *G_dp2*: If the amplitude is within ($a4 < a < a3$), and its time duration is within the interval ($\triangle t4 < d < \triangle t3$) as shown in Figure 4.2, then the glitch propagates through one cascaded gate.

4. *G_dp1*: If the amplitude is within ($0 < a < a4$), and its time duration is within the interval ($0 < d < \triangle t4$) as shown in Figure 4.2, then the glitch does not propagate at all.

Putting Logical and Electrical Masking Together: Figure 4.1 shows the case when *G_dp3* amplitude is ($a3 < a < a2$) and the duration is ($\triangle t3 < d < \triangle t2$). This glitch is injected at the output of gate G1. This glitch now is controlling the next gate (G2) if and only if the other input of G2 is at logic '1'; otherwise it will be logically masked as we explained in the previous chapter. As the glitch propagates through G2, electrical masking reduces the amplitude and the interval and hence the output of G2 shows a glitch of strength *G_dp2* only. As the glitch reaches at

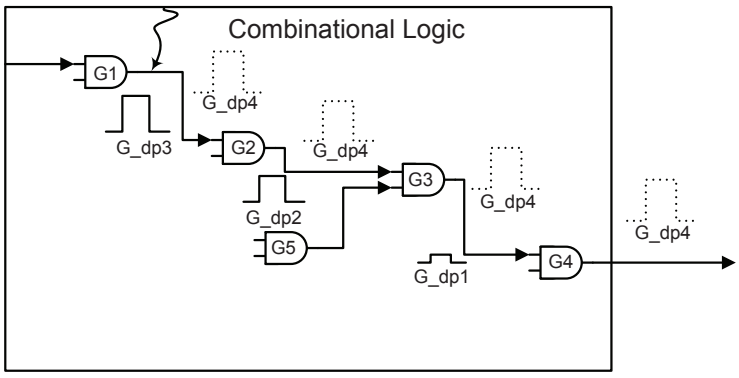

Figure 4.1: Glitch propagation with electrical masking.

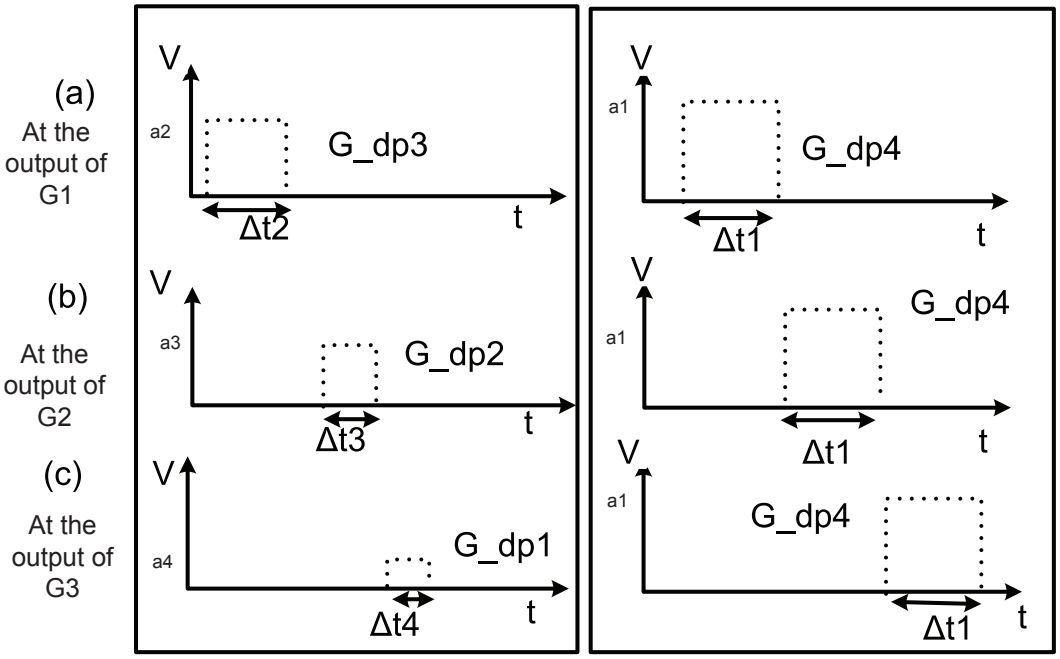

Figure 4.2: A glitch at the (a) output of initial gate G1, (b) output of gate G2, (c) output of gate G3 on sensitized path.

the input of G2 with level *G_dp2* assuming that the output G5 is logic '1' then the glitch characteristics reduction from the propagation through G3 is represented by moving the glitch from *G_dp2* to *G_dp1*. In the next sections we combine these assumptions with the GP sets in order to build the truth table for each logic gate.

4.3 Modeling Electrical and Logical masking

In section 3.2, we used the GP sets to build the truth table for different logic gates and *Muller C-element* in order to model logical masking. In this section, we extend this principle to model both electrical and logical masking. Table 4.1 shows the combination of electrical and logical masking by extending the truth tables, which was previously made for the logical masking only. This table shows the corresponding output of AND gate for all possible inputs combinations. For example, if the glitch is injected at one of the inputs of AND gates with level *G_dp3* then provide the

other input is at logic '0' the glitch will not be able to propagate to the output due to logical masking. If the other input is logic '1' one then the glitch will propagates to the output due to logical masking. Whereas if the non-glitch input is at lo logic '1' then the glitch on the other input may propagate to the output.

There are two possible transition scenarios with soft error glitch, either the original signal transits from logic '0' towards '1' or logic '1' towards '0'. It is assumed that the glitch signals are represented as *G_dp4, G_dp3, G_dp2,* and *G_dp1* in the former case, and they are represented as glitch signals *G_dp4', G_dp3', G_dp2'* and *G_dp1'* in the later case. Both cases are included in Table 4.1. For example, if one of the inputs for the AND gate is logic '1', and the other input is *G_dp3 or G_dp3'* then the output will be *G_dp2/G_dp2'*. The same principle is used to build Table 4.2 which shows the corresponding outputs for all the possible inputs combinations in the case of OR gate.

In order to illustrate this technique, we applied it on the circuit in Figure 4.1. This circuit is a combination of different AND gates. As a first step a glitch is injected at one of the internal node. Here we injected it at the output of *G2 (out_G2)*. Injecting the glitch at the internal node is done using multiplexers as explained before. Next step is to define the behaviour for each gate by extracting the truth table for it from Table 4.1.

Table 4.1: The truth table for AND gate.

	0	**1**	**G_dp4/G_dp4'**	**G_dp3/G_dp3'**	**G_dp2/G_dp2'**	**G_dp1/G_dp1'**
0	0	0	0/0	0/0	0/0	0/0
1	0	1	G_dp4/ G_dp4'	G_dp2/ G_dp2'	G_dp1/ G_dp1'	0/ 1
G_dp4/ G_dp4'	0/ 0	G_dp4/ G_dp4'	G_dp4/ G_dp4'	G_dp4/ G_dp4'	G_dp4/ G_dp4'	G_dp4/' G_dp4'
G_dp3/ G_dp3'	0/ 0	G_dp2/ G_dp2'	G_dp4/ G_dp4'	G_dp2/ G_dp2'	G_dp2/ G_dp2'	G_dp2/ G_dp2'
G_dp2/ G_dp2'	0/ 0	G_dp1/ G_dp1'	G_dp4/ G_dp4'	G_dp2/ G_dp2'	G_dp1/ G_dp1'	G_dp1/ G_dp1'
G_dp1/ G_dp1'	0/ 0	0/ 1	G_dp4/ G_dp4'	G_dp2/ G_dp2'	G_dp1/ G_dp1'	1/ 0

One of the contributions this work is to propose an easier representation by modeling both logical and electrical masking in one decision diagram, which is the multiway decision graph. Figure 4.3 shows the multiway decision graph (MDG) for the circuit in Figure 4.1 at the output of *G3*. Signals *out_G5*, *out_G3* and out_G2 present the outputs of *G5*, *G3* and G2 in Figure 4.1 respectively. This decision graph is similar to what we have in Table 4.1 for the case of AND gates. As shown in this figure G3 is an AND gate which has two inputs; the first one is coming from the output of G5, and the second one is coming from the output of *G2*. Starting with *out_G5* if it is logic 0 then the output of *G3* is logic '0' regardless of the value of *out_G2* which is the glitch signal. But if *out_G5* is equal logic '1' then the output of *G3* is dependent on the output of *G5*. In the last case, the glitch can propagate to the output of *G3* if it has sufficient amplitude and duration. Figure 4.3 shows all the possible cases of *out_G5*, *out_G2* and the corresponding output of *G3 (out_G3)*.

Table 4.2: The truth table for OR gate.

	0	**1**	**G_dp4/G_dp4'**	**G_dp3/G_dp3'**	**G_dp2/G_dp2'**	**G_dp1/G_dp1'**
0	0	1	G_dp4 /G_dp4'	G_dp2 /G_dp2'	G_dp1 /G_dp1'	0 /1
1	0	1	1/1	1/1	1/1	1/1
G_dp4 /G_dp4'	G_dp4 /G_dp4'	1 /1	G_dp4 /G_dp4'	G_dp4 /G_dp4'	G_dp4 /G_dp4'	G_dp4 /G_dp4'
G_dp3 /G_dp3'	G_dp2 /G_dp2'	1 /1	G_dp4 /G_dp4'	G_dp2 /G_dp2'	G_dp2 /G_dp2'	G_dp2 /G_dp2'
G_dp2 /G_dp2'	G_dp1/ /G_dp1'	1 /1	G_dp4/ /G_dp4'	G_dp2 /G_dp2'	G_dp1 /G_dp1'	G_dp1 /G_dp1'
G_dp1 /G_dp1'	0 /1	1 /1	G_dp4 /G_dp4'	G_dp2 /G_dp2'	G_dp1 /G_dp1'	0 /1

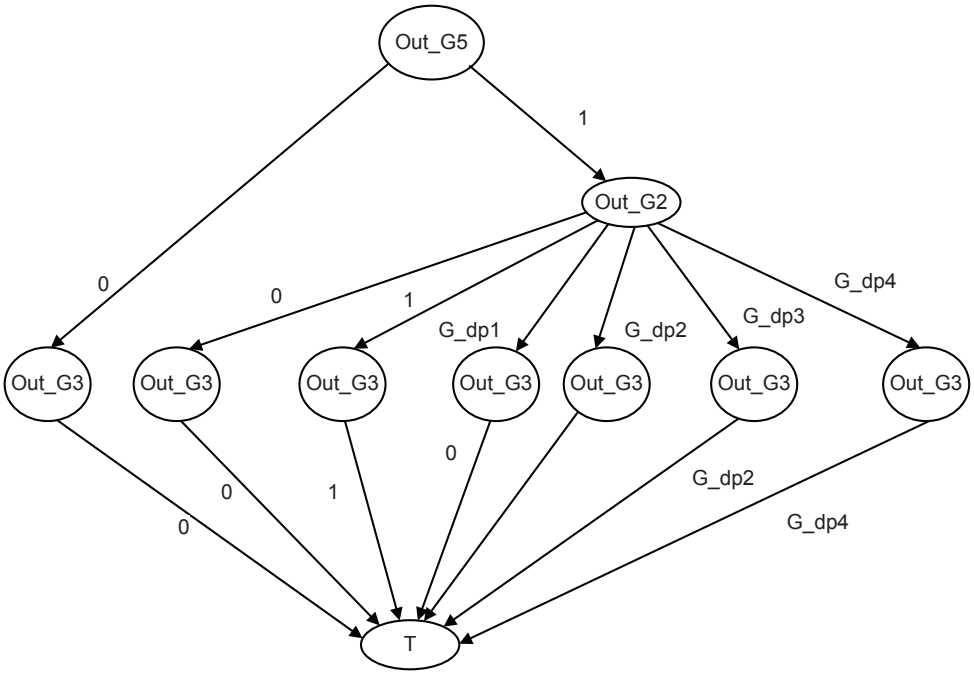

Figure 4.3: Multiway Decision Graph.

4.4 The Proposed Methodology

This section presents an overview of the proposed technique. Figure 4.4 shows the flow chart of our methodology for integrating electrical and logical masking. This is similar to Figure 3.1, with the exception of introduction of the glitch depth block. Our methodology requires structural specifications as input, which is the MDG-HDL description of the design. The next step is to specify the initial glitch depth to be injected. This addition is done by defining a new signal type called glitch type signal. This signal type can take any of the following values: *1, 0, G_dp1, G_dp2, G_dp3, G_dp4.* After that the next step is the glitch injection at the vulnerable nodes of the design by modifying the structural specifications. The glitch injection mechanism is similar to the one adapted in the previous chapter, where multiplexers are added at the internal nodes. These multiplexers can choose among the normal mode and the various possible glitch injection values. These glitches are injected for

any of the desired level, among *G_dp1* to *G_dp4*, as and when required.

After injecting the design with glitches the next logical step is to identify the conditions under which the glitches at a particular internal node may propagate to the primary output. Invariant checking, a formal verification approach, is applied to examine the possibility of glitch propagation. An invariant, similar to what is used in Section 3.4, is applied over to perform this verification. It can be written in temporal logic as:

$$AG((reg\_out = 1) \vee (reg\_out = 0))$$

Which means for all the reachable state in the design the output is always logic '1' or logic '0'.

In case the glitch can reach the output, which means that both electrical and logical masking did not prevent the glitch from propagating. In this case our technique provides the user with a counterexample. The designer can infer the following from the counterexample:

- The critical sequences, which are the input sequences that allow the glitch to propagate to the output. With these sequences logical masking alone cannot prevent the glitch from reaching the output.
- The minimum glitch depth required by the glitch signal at the vulnerable nodes in order to be able to propagate and reach the primary output.

The flow chart in Figure 4.4 shows that our technique allows the user to verify the glitch propagation with different glitch depths. Start with verifying the glitch propagation at the first level *(G_dp1)* then after that verify the glitch propagation with different depth level such as: *G_dp3, etc.*

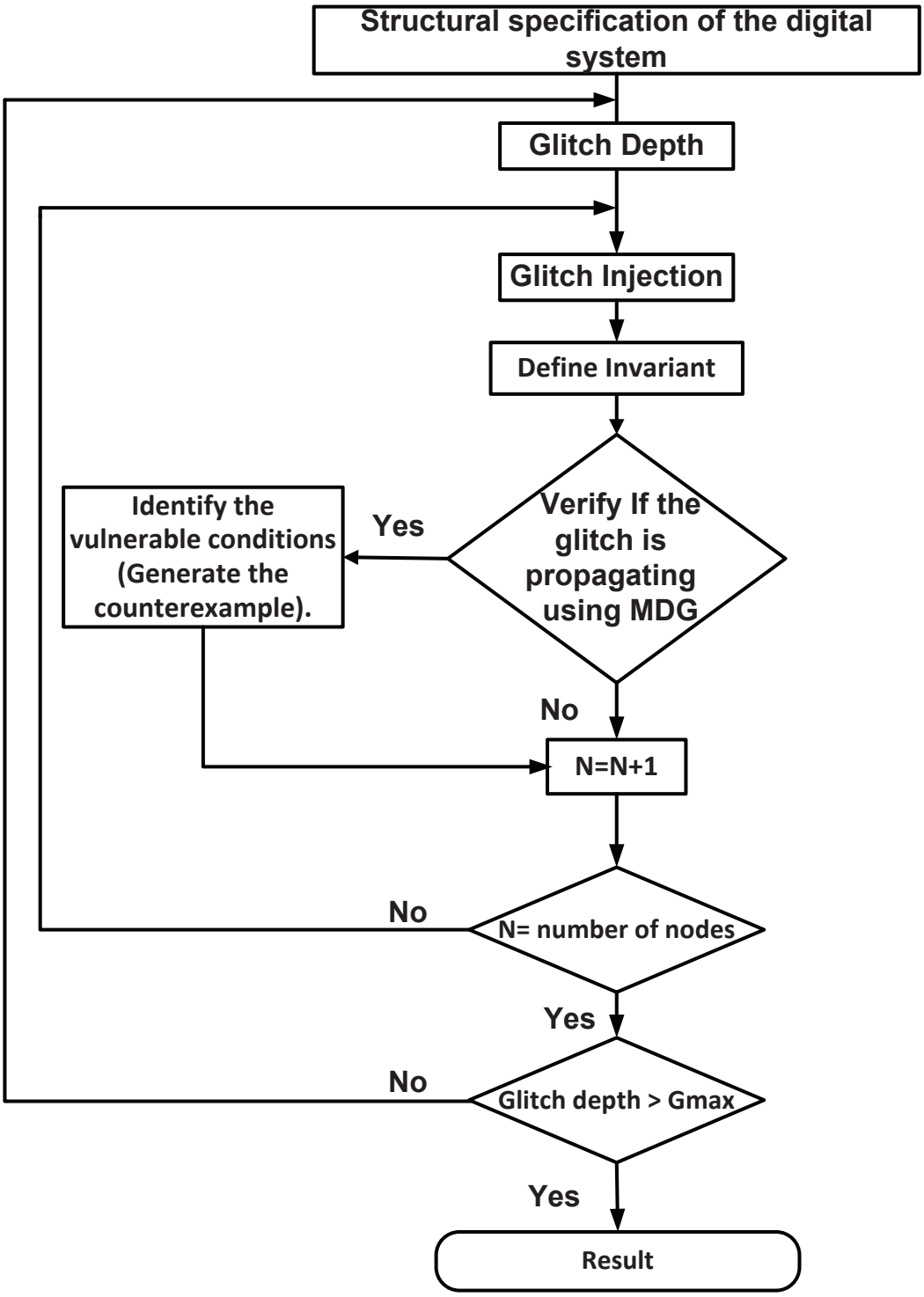

Figure 4.4: The flow chart for the proposed methodology.

4.5 Summary

In this chapter, we proposed a new methodology to model the combine effect of electrical and logical masking. We extended the proposed methodology in chapter 3 to model the effect of both masking. We explained the proposed methodology in detail. We discussed in detail the identification of soft error glitch, we built our own assumption and notation for the identification of the glitch. Then we defined new truth tables for each logic gates along with GP sets concepts. The implementation of this methodology will be explained later in this thesis. In order to make the proposed methodologies applicable on large design with large number of internal nodes. In the next chapter we propose a full automation of the proposed methodologies.

Based on our observation we assumed that *G_dp4* is maximum stages required glitches to completely attenuated. The number of cascaded gates that the glitch can propagate through in our work is mainly depending on the following:

- The initial value of the glitch which it is injected at the internal node. For example, if the glitch is injected with *G_dp3* then the glitch can propagate through 2 gates in the propagation path before it reaches to *G_dp1* as shown in Figure 4.1. If the glitch at level *G_dp2* then maximum it can propagate through 1 gates before it reaches to *G_dp1*.

- As an initial assumption, we assumed that all the gates have the same size. Different behaviour for each gate is possible by creating different truth tables for each gate to define its own behaviour.

Chapter 5

SEGP-Finder: Automating Identification of Soft Error Glitch-Propagating Paths(SEGP)

In this chapter, we proposes a new tool, soft error glitch-propagating path finder *SEGP-Finder* to identify the propagation of soft error at gate level. *SEGP-Finder* models electrical and logical masking effects and verify the glitch propagation by implementing previous proposed methodology. The applicability of the tool over combinational and asynchronous circuits is illustrated by implementing 8-bit adders, multipliers, and the Self-timed multiple-group pipeline asynchronous handshake circuits.

5.1 Identification of Soft-Error at Gate-Level

In order to explain the automation of the proposed technique, we are providing usage of the files that are required by MDG. Our automation tool (described in next section) modifies these files; therefore this information is helpful in understanding the contextual details:

- The *algebraic specification* file defines sorts, function types and generic constants used in hardware descriptions. In our work, this file contains the definition of the glitch sort signal, where 0, 1, *DG, DGP* are the possible values for this signal in the design when logical masking is only modeled. G_dp1, G_dp2, G_dp3, G_dp4 are the possible values in case of modeling both electrical and logical masking.

- The *symbol order* file provides the custom (user-defined) symbol order for all the variables and cross-operators which would appear in MDG model.

- The *circuit description* file declares signals and their sort assignments, component network, outputs and the mapping between state variables and next state variables.

- The *invariant specification* file defines the invariant to be checked during the reachability analysis. E.g. the invariant (written in temporal logic) $AG((reg\_out = 0)||(reg\_out = 1))$ means *"In all the reachable states, the output of the register is equal to 1 or 0".*

5.2 SEGP-Finder

This section explains the features of the proposed tool (*SEGP-Finder*). This tool implements the methodology in the previous chapter. Flow chart of *SEGP-Finder* is shown in Figure 5.1. It illustrates that the input of our tool is the Gate-level MDG-HDL model of the design combined with the GP sets and the glitch identification. There are four main stages shown in Figure 5.1. The first one is to annotate the MDG Model where modifications are applied on the design file to accommodate possible soft error glitches at the vulnerable nodes. Output of the annotated MDG Model is given to the invariant generator, which provides the MDG-invariant. Next is the MDG verification step where we perform the invariant checking in MDG.

Finally, *SEGP-Finder* analyzes the results and generates result file. The following sections discuss the details of these stages.

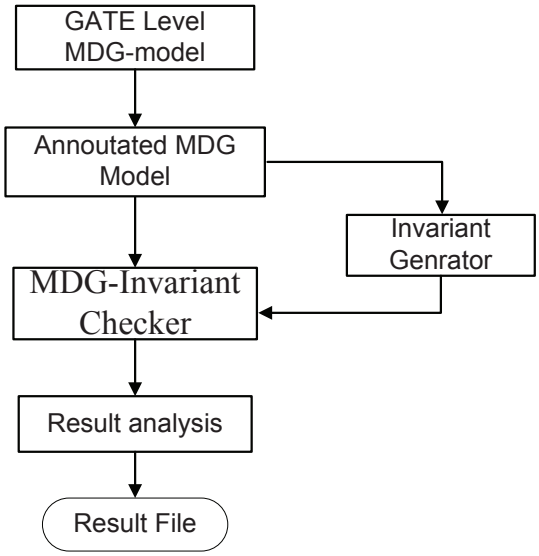

Figure 5.1: The flow chart of the *SEGP-Finder*.

5.3 Annotate the MDG Model

Annotating the effects of soft error glitches in the design is implemented by annotator, shown in Figure 5.2. As the name suggests, annotator prepares the design written in MDG-HDL to perform verification. The goal here is to modify the MDG-HDL model of the design to accommodate soft error glitches. This is done by modifying the MDG input files as explained in Section 2 and graphically shown in Figure 5.2.

Next step is to inject the glitch for each internal node. It shows that for each new description file annotator injects the glitch at one internal node. This is a recursive process done for all the internal nodes. As an example, for *"Spec_1"* our program injects the glitch at the first internal node. The annotator adds a

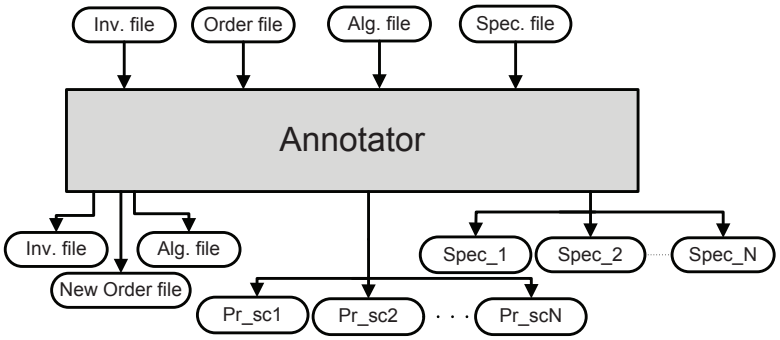

Figure 5.2: The annotator.

multiplexer (mux) component to the design in *"Spec_1"*. This introduction of mux component is made possible by modifying the subsequent connecting gates. To illustrate fully the working algorithm, consider the small circuit example in Figure 5.3 (*ISCAS-85* benchmark circuit *C17* with glitch inserted at N1). To perform this glitch injection at N1, the annotator opens the *"Spec_1"* and adds a mux component to the design at N1 which, in MDG-HDL, is the following:

component(mux, mux(sel(select) ,inputs([(0,N1), (1,y)]) , output(n_select_init))).

In MDG every signal in the design is sequentially arranged in the order file which is the representation format of net-list in MDG. The signals related to the new mux component need to be inserted in the order file. The annotator creates a copy of the original order file. Next it opens the new order file and adds these new glitch signals to the order list of the design.

In order to run the MDG to do our verification one time the user has to guide or interact with MDG nine times. It is very hard to do this for large design with large number of internal nodes. In this tool we handle this issue by writing a *prolog* script for each glitch verification case. The number of these scripts is equal to the number of the internal nodes. These scripts are automatically generated without any interference from the user.

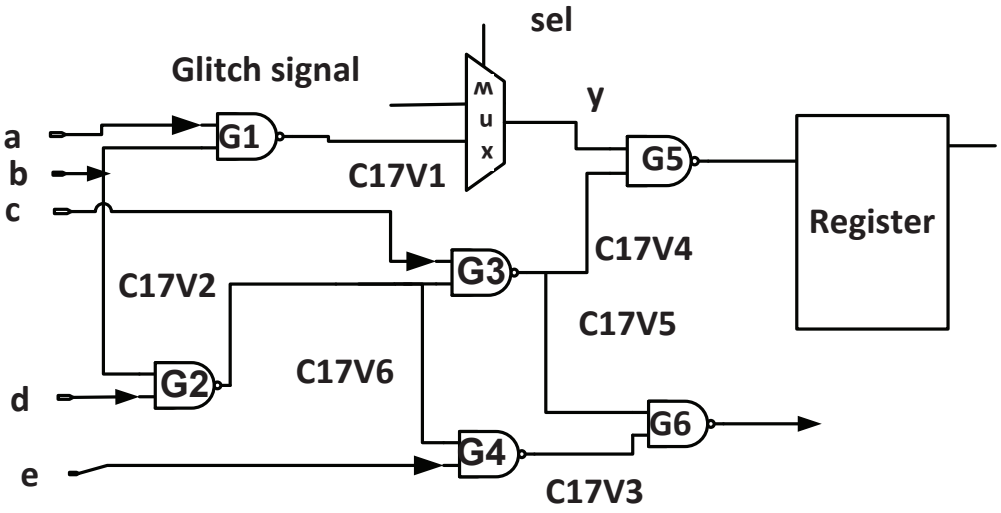

Figure 5.3: C17 with the glitch injected between G1and G5.

5.4 MDG Invocation and Result Organization

The proposed approach includes running all the *prolog* scripts in sequence. This can be done by writing a shell script. This script is generated automatically by the annotator and it performs two main functions. The first function is to run all the *prolog* scripts as shown in the top box of Figure 5.4. In the second step MDG verification results for all the iterations are stored into separate files. Therefore the number of result files is equal to the number of the internal nodes. Afterwards, as shown in Figure 5.4, an analysis is run on these result files. Here the tool extracts the critical sequence, the counterexamples, along with verification time information from each result file. Next section further discusses these results.

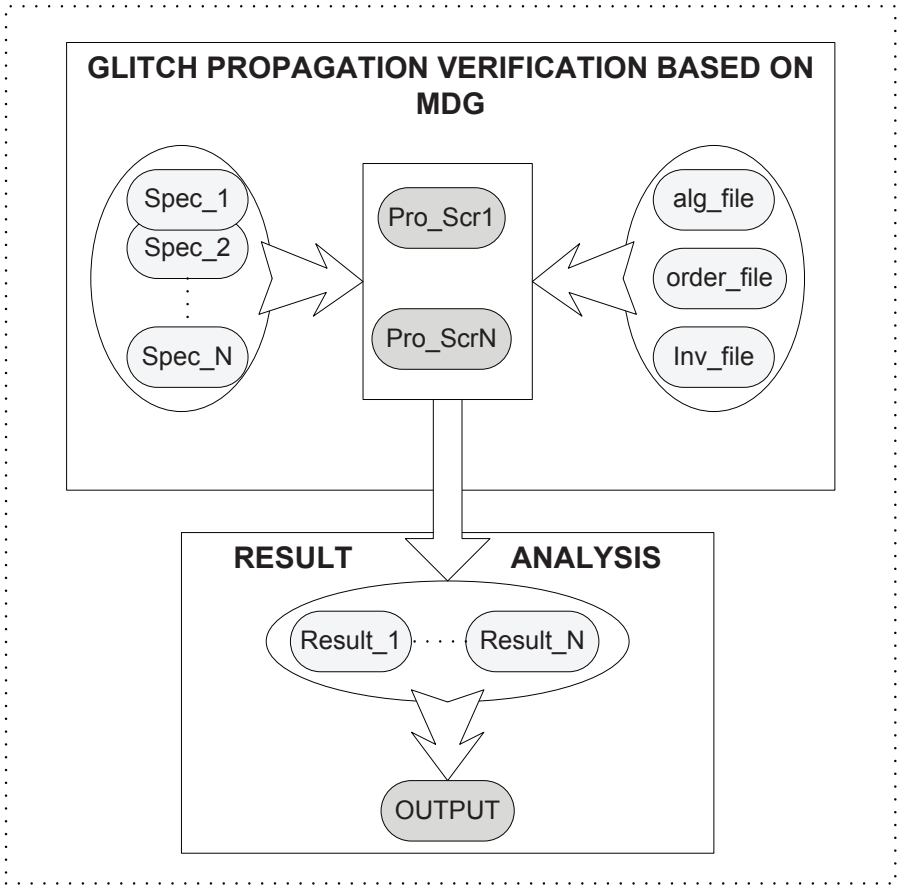

Figure 5.4: User Transparent MDG Invocation and Result Analysis.

5.5 Experimental Results

We performed our verification using a SUN RAY2 computer, with the SUSE Linux Enterprise Server 10 operating system, over an Intel core i7-860 processor.

5.5.1 Implementation

Logical Masking

For the Combinational circuits starting with the *C17* circuit, for each design we injected glitches at every internal node of the circuit and invariant checking is performed in each case.

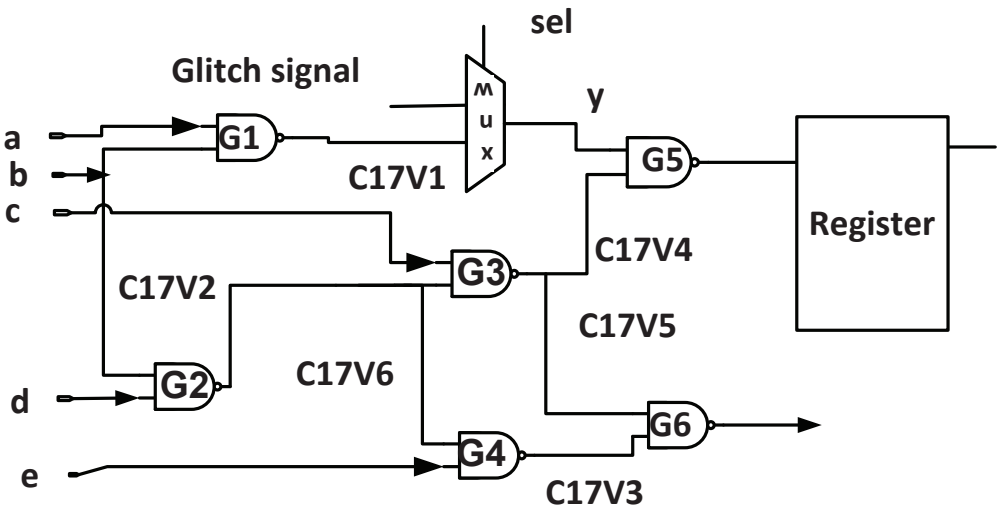

Figure 5.5: C17 with the glitch injected between G1 and G5.

```
=== The Counterexample ===
-------- Assumptions --------
-
-------- Initial state --------
reg_output_signal = 1
-------- Clock cycle 1 --------
  -- The symbolic input --
c = 0
select = 1
glitch_signal = dgp
  -- The symbolic state --
reg_output_signal = dg
-------- Clock cycle 2 --------
  -- The symbolic input --
-
  -- Symbolic Output --
flag = 1
=== End of counterexample ===
Generating counter example took 0.010 seconds.
```

Figure 5.6: One of the counterexamples for the C17.

As we explained before, our proposed method provides counterexamples in case the invariant checking fails. One such case is shown in Figure 5.6, which shows the counterexample when the glitch is injected at node *C17V1* in Figure 5.3. In Figure 5.6, it is shown that, starting from the initial state of the output register, the MDG tool provides the symbolic inputs which are the critical sequence of inputs that allows the glitch on the node *C17V1* to propagate to the primary output. The critical sequence for the previous case is when the input signal *c* is at logic '0' and the *select* signal for the multiplexer is at logic '1'. The verification time and the time needed to generate the counterexample are between 10 and 20 millisecond. *SEGPfinder* collects all the counterexamples for the target circuit and provide the user back with table that includes all the results such as Table 5.1 for the C17 circuit. Table 5.1 shows the critical sequences of all the internal nodes of the *C17* circuit *(C17V1-C17V6)* .

Table 5.1: Node wise vulnerable conditions for C17.

Internal node	Vulnerable Conditions
C17V1	c = 0
C17V2	c = 1, a = 0
C17V3	c = 0
C17V4	a = 0, b = 0
C17V5	e = 0
C17V6	e = 1 , c = 0

The second example on the combinational circuit implementation is the 4-bit adder which is shown in 5.7. By giving the MDG model of the 4-bit adder as input to our tool Table 5.2 is the output. This table shows the corresponding critical sequence or vulnerable condition for each internal node. *a0 = 0, b0 = 0, select = 1, glitch_signal = dg* is the critical sequence when the glitch is injected at node *N2* in Figure 5.7. Based on the result shown in Table 5.2 logical masking effect alone is not able to prevent the glitch from propagating. Later in this section, we show

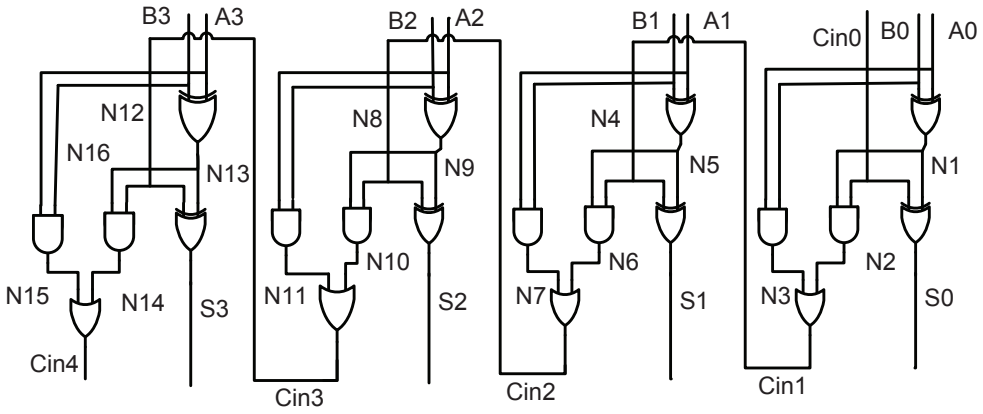

Figure 5.7: Gate-Level representation of 4-bit adder circuit.

the modeling of the electrical masking effect in this circuit, then we verify if both masking effects can prevent the glitch propagation.

Table 5.2: Node wise vulnerable conditions for 4-Bit adder circuit.

Node name	Verification time (sec.)	Vulnerable Condition
N1	0.37	select = 1, glitch_signal = dg, cin0= 0.
N2	0.34	select = 1, glitch_signal = dg, a0 = 0, b0=0.
N3	0.34	select = 1, glitch_signal = dgp, cin0= 0.
N4	0.34	select = 1, glitch_signal = dgp, a1 = 0, b1 = 0.
N5	0.33	select = 1, glitch_signal = dg, cin1 = 0.
N6	0.33	select = 1, glitch_signal = dgp, a1 =0, b1 =1.
N7	0.33	select = 1, glitch_signal = dg, cin1= 0.
N8	0.36	select = 1, glitch_signal = dg, a2 = 0, b2=0.
N9	0.34	select = 1, glitch_signal = dg, cin2= 0.
N10	0.36	select = 1, glitch_signal = dgp, a2 = 0, b2 =1.
N11	0.37	select = 1, glitch_signal = dg, a2= 0, b2=0.
N12	0.39	select = 1, glitch_signal = dg, a3 = 0, b3 = 0.
N13	0.38	select = 1, glitch_signal = dgp,cin3 = 0.
N14	0.37	select = 1, glitch_signal = dgp,a3 = 0, b3 = 0.
N15	0.37	select = 1, glitch_signal = dgp,cin3= 0.
N16	0.35	select = 1, glitch_signal = dgp,cin3= 1,a3=0, b3=1.

For asynchronous circuits, we not only have to consider the input values but also the order in which signals get asserted. For example, according to the *1-of-N* DI data-encoded protocol, of which *1-of-4* is the special case shown in Figure 3.6, only one of the input lines can go high at a given time. When any of these input signals becomes high (to transfer some data), the corresponding line in the signal group *nx[0-3]* is pulled down to logic "0". Based on this protocol, the possible input sequences *(Z0-Z3)* are obtained. We call these possible input sequences initial input sequences of the multiplexer in Figure 3.6. Based on the chosen initial input sequence from the multiplexer, the decoder gives certain values for each input of the circuit. To ease reading, we have provided these initial sequences in Table 5.3.

Table 5.3: The initial sequences for the DI circuit.

Initial sequence	The input value					
	q0	q1	q2	q3	nxa	out_ack
Z0	1	0	0	0	1	0
Z1	0	1	0	0	1	0
Z2	0	0	0	0	0	1
Z3	0	0	0	0	1	0

Table 5.4 shows the critical sequence for all the internal nodes for the DI circuit under all the possible input sequences. The previous state output of the C-element X is represented by *t_IsX*. *Select2* is the selection signal of the multiplexer at the internal node. Table 5.4 shows that the glitches at internal node can propagate under all the possible input sequences if the previous value of the output C-element is logic *'1'*.

The vulnerable nodes for the bundle data circuit in Figure 3.8 are identified as *BDV1* to *BDV8*. For the introduction of glitches at each node, there is a specific critical sequence for the glitch to propagate. Such as, introducing glitch at *BDV1*

Table 5.4: Node wise vulnerable conditions for DI asynchronous circuit.

Internal Node	Initial sequences	Vulnerable Conditions
Nx0	Z0	t_ls5 = 0, glitch_signal = dgp, select2 = 1.
Nx0	Z1	t_ls5 = 1, glitch_signal = dg, select2 = 1.
Nx0	Z2	t_ls5 = 1, glitch_signal = dg, select2 = 1.
Nx0	Z3	t_ls5 = 1, glitch_signal = dg. select2 = 1.
Nx1	Z0	t_ls6 = 0, glitch_signal = dgp select2 = 1.
Nx1	Z1	t_ls6 = 1, glitch_signal = dg select2 = 1.
Nx1	Z2	t_ls6 = 1, glitch_signal = dg, select2 = 1.
Nx1	Z3	t_ls6 = 1, glitch_signal = dgp select2 = 1.
Nx2	Z0	t_ls7 = 0, glitch_signal = dgp select2 = 1.
Nx2	Z1	t_ls7 = 1, glitch_signal = dg select2 = 1.
Nx2	Z2	t_ls7 = 1, glitch_signal = dgp select2 = 1.
Nx2	Z3	t_ls7 = 1, glitch_signal = dg select2 = 1.
Nx3	Z0	t_ls8 = 0, glitch_signal = dgp select2 = 1.
Nx3	Z1	t_ls8 = 1, glitch_signal = dg select2 = 1.
Nx3	Z2	t_ls8 = 1, glitch_signal = dgp select2 = 1.
Nx3	Z3	t_ls8 = 1, glitch_signal = dg select2 = 1.

in Figure 3.8, then the counterexample shows that the critical sequence is *Reset=1, Z0=1, Ai=1 or Reset=1, Z0=1, Rp=1.* We applied our MDG model and performed our verification methodology for all the identified vulnerable nodes of the circuit and a log of critical sequences was taken. We obtained a table similar to Tables 5.1 and 5.4 for bundled data protocol as well, but because of space limitation, we did not put it here.

Figure 5.8 shows the *self-timed multiple-group pipeline* circuit [48] to be verified using our technique. In order to verify the glitch propagation for this asynchronous circuit we started with verifying the completion detection part which is shown in Figure 5.9. A gate level representation of the 3-of-6 completion detection is shown in Figure 5.9. We injected the glitch at the inputs of the completion detections shown in Figure 5.9. Table 5.5 shows partial results of this design when the initial values of all primary inputs are zero.

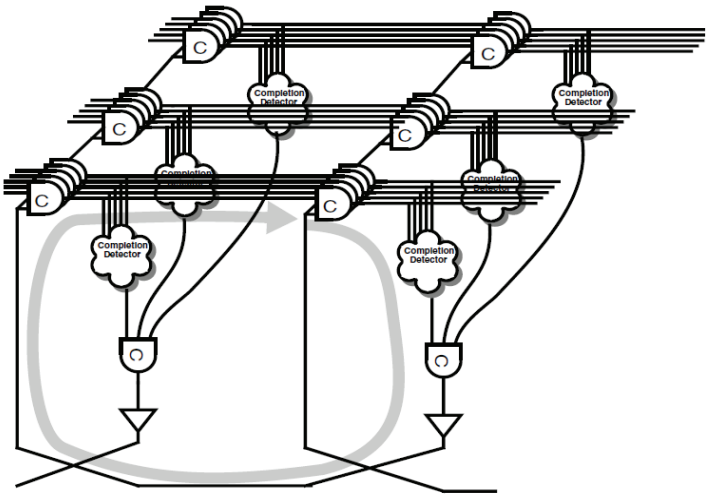

Figure 5.8: Self-timed multiple-group pipeline [48]

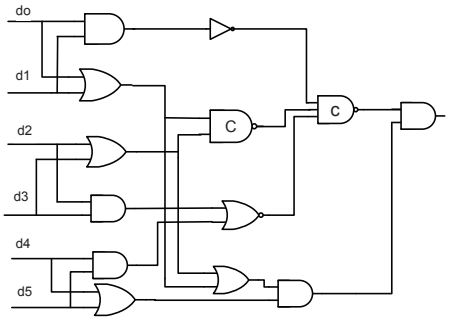

Figure 5.9: 3-of-6 completion detection.

Electrical Masking

In our tool *SEGP-Finder* we implemented the proposed methodology which we discussed in chapter 4, in order to model the electrical masking. Here we show our tool results for modeling electrical masking on different combinational and asynchronous circuits. Starting with the 4-bit adder circuit shown in Figure 5.10. A gate level representation of the 4-bit adder is shown in Figure 5.7. As we explained earlier the 4-bit adder consists of multiple full adder blocks. So one efficient way to verify this design is to verify one of the full adders. We injected the glitch at one internal node of the full adder circuit then we verified the glitch propagation for all the outputs.

Table 5.5: Node wise vulnerable conditions for the self-timed multiple-group pipeline circuit.

Node name	Verification time (sec.)	Vulnerable Condition
D0	0.70	select=1,glitch_signal=dg,feed1=0,in1=0, in3=0,in4=0,in5= 0,t_ls1=1,t_ls2=0,t_ls3=1
D1	0.73	select=1,glitch_signal=dg,feed1=0,in2=0, in3=0,in1=0,in5=0,t_ls1=1,t_ls2=0,t_ls3= 1.
D2	0.73	select=1,glitch_signal=dg,feed1=0,in1=0, in2= 0,in4=0,in5=0,t_ls1=1,t_ls2=0,t_ls3=1.
D3	0.74	select=1,glitch_signal=dg,feed1=0,in1= 0, in2 = 0,in3=0,in5 = 0,t_ls1 = 1,t_ls2 = 0,t_ls3 = 1.
D4	0.75	select =1, glitch_signal=dg, feed1=0,in1=0, ,in2 =0,in3=0,in4=0,in6=1,t_ls9=1,t_ls2=0,t_ls3 = 1.
D5	0.70	select=1,glitch_signal=dgp,feed1=0,in1=0,in2=0, in3=0,in4=0,in5=1,t_ls8 = 1,t_ls2 =0,t_ls3= 1.

Out of the results shown in Table 5.6, we are able to find that the glitch can propagate from the full adder where it is injected to the next full adder based on the glitch assumption we have. By keeping the glitch in the same position as before (at node N1) and checking the carry of the second full adder in Figure 5.10 we noticed that the invariant checking a success which means that the glitch cannot reach that point due to the electrical masking. Because the 4-bit adder is a combination of full

Table 5.6: The result when a glitch the glitch inserted at the output of *N1*.

primary output	Glitch depth at the output	Vulnerable Conditions
S0	G_dp2	select = 1, glitch_signal = G_dp3, cin0 = 0.
C0	G_dp1	select = 1, glitch_signal = G_dp3, cin0 = 1, a0 = 1,b0 = 0.
S1	G_dp1	select = 1, glitch_signal = G_dp3, cin0 = 1,a0=0, a1 = 0, b1 = 0.
C2	Safe	Safe

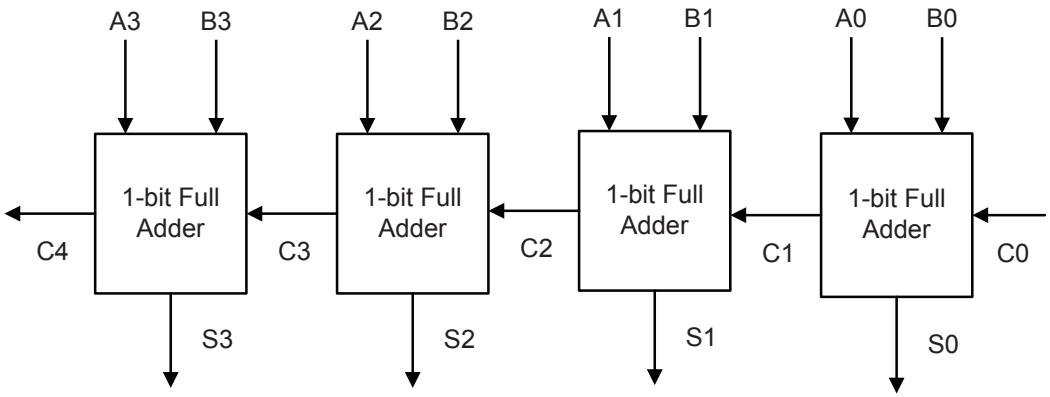

Figure 5.10: 4-bit adder circuit.

adder blocks we do not need to verify the glitch propagation for all the full adders. Verifying one block saves time and reduces complexity.

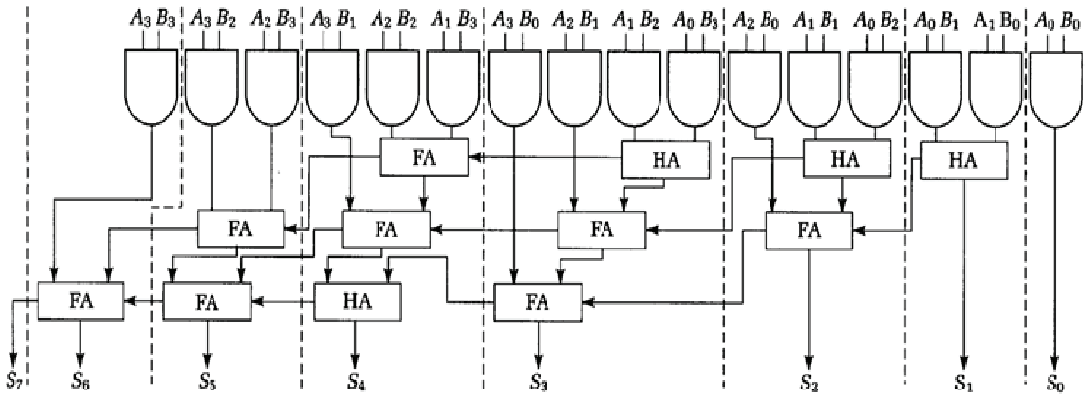

Figure 5.11: 4-bit multiplier circuit.

A combinational multiplier is a good example for showing how our technique can be used to verify electrical masking because we have sufficient depth. The 4-bit multiplier circuit is shown in 5.11. The scheme that is shown in Figure 5.11 is often referred to as ripple carry since each more significant column in the sum must wait for the carries to be computed in the less significant columns before its corresponding sum bit can be computed.

In the figure above the two bits to be added enter from the top, any carry in

from the right enters from the right, and any carry out exits from the left of each block. The output from the bottom of a block is the sum. The least significant output bit, S0 (the first column), involves only two input bits and is computed as the simple output of an AND gate. This operation cannot generate a carry out. The next output bit, S1, involves the sum of two partial products. A half adder is used to form the sum since there can be no carry in from the first column; however, a carry out can be produced. The third output bit, S2, is formed from the sum of three (1-bit) partial products plus a possible carry in from the previous bit. This operation requires two cascaded adders (one half adder and one full adder) to sum the four possible input bits (three partial products and one possible carry in from the right). The remaining output bits are formed similarly. Because in some columns we must add more than two binary numbers, there may be more than one carry out generated to the left. Similar to the 4-bit adder, we injected the glitch with different depth and for all the internal nodes.

5.5.2 Discussion

In this section, a comparison is provided between our proposed soft-error estimation technique and other contemporary techniques. Conventionally, the total number of tested input sequences for the glitch propagation can be found using the following formula:

$$f(x, y) = 2^x * y \tag{5.1}$$

Where x is the number of primary input and y is the number of internal nodes (vulnerable nodes). For example, the total number of tested input sequences for the *C17* is 192 because it has 5 input and 6 internal nodes. Our methodology allows the designer to determine the smallest set of input sequences that are responsible for soft-error glitch propagation. Table 5.8 illustrates this fact; as it can be seen,

the required number of the inputs to the sequence to be verified is reduced in all the cases. For example, for the *C17*, instead of verifying the circuit for 192 input sequences, using our technique, we need to verify the circuit under only 6 sequences. The results shown in all the tables in this section can be provided to the designers to analyze and possibly avoid the soft error glitch propagation using logic design and circuit techniques. These results are also beneficial to DFT groups so that it can be made sure that these logic paths have ample controllable and observable points. This information can help a protocol developer to refrain from using the vulnerable paths as much as possible. Another advantage of our tool is the possibility of applying partially instead of doing it for all the internal nodes. It helps a designer in investigating only part of the design; partial verification is of course time efficient. The fact that *SEGP-Finder* has been successfully applied to relatively large circuits proves the scalability of the technique.

Table 5.7: Node wise vulnerable conditions for 4-Bit adder circuit.

Node name	Verification time (sec.)	Vulnerable Condition
N1	0.37	select = 1, glitch_signal = G_dp3, cin0= 0.
N2	0.34	select = 1, glitch_signal = G_dp3, a0 = 0, b1 = 0.
N3	0.34	select = 1, glitch_signal = G_dp3, cin0 = 0.
N4	0.34	select = 1, glitch_signal = G_dp3, a1 = 0, b1 = 0.
N5	0.33	select = 1, glitch_signal = G_dp3, cin1 = 0.
N6	0.33	select = 1, glitch_signal = G_dp3, a1 = 0, b1 = 1.
N7	0.33	select = 1, glitch_signal = G_dp3, cin1 = 0.
N8	0.36	select = 1, glitch_signal = G_dp3, a2 = 0, b2 = 0.
N9	0.34	select = 1, glitch_signal = G_dp3, cin2 = 0.
N10	0.36	select = 1, glitch_signal = G_dp3, a2 = 0, b2 = 1.
N11	0.37	select = 1, glitch_signal = G_dp3, a2 = 0, b2 = 0.
N12	0.39	select = 1, glitch_signal = G_dp3, a3 = 0, b3 = 0.
N13	0.38	select = 1, glitch_signal = G_dp3, cin3 = 0.
N14	0.37	select = 1, glitch_signal = G_dp3, a3 = 0, b3 = 0.
N15	0.37	select = 1,glitch_signal = G_dp3, cin3= 0.
N16	0.35	select = 1, glitch_signal = G_dp3, cin3 = 1,a3 = 0, b3 = 0.

Table 5.8: Identifiable vulnerable input sequences.

	Number of internal nodes	Required number of inputs		% of require inputs	verification time
		Available inputs	Proposed		
C17	6	192	6	3.125	1.8
4-bit adder	20	5120	20	0.39	6.2
DI	4	256	16	6.25	5.6
Bundle data	8	1024	80	7.8125	28

5.6 Summary

In this chapter, *SEGP-Finder* has been proposed, which is a new tool to identify the propagation soft error glitch at gate level. We explained how *SEGP-Finder* models electrical and logical masking effects and verify the glitch propagation by implementing previous proposed methodology. *SEGP-Finder* has been tested over many combinational and asynchronous circuits. Considerably large designs, such as adders and multipliers, have been implemented with little intervention requirement, emphasizes the efficiency and scalability of this tool. It is shown that the counter-examples, generated from this methodology, can be exploited at various design abstraction levels. Based on the results indicate that *SEGP-Finder* is fast and accurate compared with simulation based techniques. The overall verification time is linearly related to the number of internal nodes.

Chapter 6

Conclusion and Future Work

We proposed a novel method to identify paths that can propagate soft faults and glitches in both combinational and asynchronous circuits. This technique provides the critical sequence of inputs for both combinational and asynchronous circuits at an early stage of the design cycle. It helps designers applying radiation tolerance techniques on limited parts of data paths. The proposed technique also considerably reduces the number of required fault injection vectors. Only 7.8 % and 6.25 % of the total input sequences need to be injected to characterize the complex bundled data protocol based and 1-of-4 DI based asynchronous circuits, respectively.

A methodology for automating the Identification of paths propagating soft faults and glitches is presented. The proposed automation tool, *SEGP-Finder* has been tested over many combinational and asynchronous circuits. Fair size designs, such as an 8-bit adder and the 4-bit multiplier, have been analyzed with little human intervention, which is a promising result with respect to the efficiency and scalability of this tool. It is shown that the counterexamples, generated with this methodology, can be exploited at various design abstraction levels.

To the best of our knowledge, this is the very first time multiway decision graphs have been used to model, analyze, and verify glitch related behaviour of

digital circuits. The proposed technique offers many benefits. It provides the designer information about what to expect in case of a soft error occurrence in the circuit. Based on this information, the designer may alter the design, insert mitigation mechanisms at particular nodes rather than introducing complete redundancy (as is the case with triple module redundancy TMR), or simply avoid vulnerable design scenarios.

The proposed methodology and tool cannot handle sequential circuits at this time. So in order to make it more general, we plan to extend it in the future to include sequential circuits. Also this work does not contain timing analysis since our verification core is the MDG does not support timing analysis. Therefore, latch window masking was not modeled in this work, because modeling this masking effect requires knowledge of the glitch arrival time window. Extending the work in this thesis can be done along different paths, such as extending the methodology to sequential circuits. This can be done by proposing a new technique to model the SR-Latches components and other sequential component.

Building on our work, soft error analysis at higher abstraction levels can also be performed. Starting from the gate level soft error analysis for some of the most commonly used combinational blocks in digital design, new GP sets for bigger blocks can be defined, then, from these GP sets, we can build new truth tables to define the behaviour for the combinational blocks. After that, a library containing the soft error defined behaviour for all basic combinational blocks can be built and used for soft error analysis.

In order to explain our last idea, a 4-bit adder circuit which is shown in Figure 5.10 can be constructed from a number of 1-bit full adders, so instead of verifying the soft error propagation for the whole design at the gate level. We can use the proposed methodology to verify glitch propagation for a 1-bit full adder at the gate level. Out of this verification, we can extract the GP sets for this circuit. Then the next step would be to build the a truth table defining the behaviour of this circuit

based on the extracted GP sets. The last step is to store the new defined behaviour in our library. In the future if we want to verify 4-bit or 8-bit adders or an adder of any size , what we need to do is just to call the 1-bit full adder models from the library and connect them and then the verification can be performed at the RTL level. The same approach can be applied to other circuits such as multipliers.

Finally, we believe this thesis is an important milestone towards building a complete environment for modeling, analysis, and verification of soft error in logic circuits.

Bibliography

[1] C. Zhao, X. Bai, and S. Dey. A scalable soft spot analysis methodology for compound noise effects in nano-meter circuits. In Proc. of ACM IEEDE sign Automation Conference (DAC), p. 894-899, June 2004.

[2] M. Zhang and N. R. Shanbhag. A Soft Error rate Analysis (SERA) Methodology. In Proc. of ACM/IEEE International Conference on Computer Aided Design (ICCAD), p.111-118, :2004.

[3] Y. S. Dhillon, A. U. Diril, and A. Chatterjee. Soft-Error Tolerance Analysis and Optimization of Nanometer Circuits. In Proc. of Design, Automation and Test in Europe (DATE), pp. 288-293, March 2005.

[4] S. Krishnaswamy, G. F. Viamonte, I. L. Markov, and J. P. Hayes. Accurate Reliability Evaluation and Enhancemenvt in a Probabilistic Transfer Matrices. In Proc. of Design, Automation and Test in Europe (DATE), pp. 282-287, March 2005.

[5] ITRS 2009, available online at http://www.itrs.net/Links/2009ITRS/Home2009.htm

[6] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. Computers, IEEE Transactions C-35 Issue:8, pp.677-691, Aug. 1986.

[7] Andrew Royal and Peter Y. K. Cheung. Globally Asynchronous Locally Synchronous FPGA Architectures. Field Programmable Logic and Applications, LNCS 2778, pp:356-364,

[8] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, L. Alivis. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. in IEEE Proceeding of the International Conference on Dependable Systems and Networks (DSN'02), June 2002, pp. 389 - 398.

[9] K. Mohanram and N. A. Touba. Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits. In Proc. of International Test Conference (ITC), pp. 893-901, 2003.

[10] N. Miskov-Zivanov, D. Marculescu. MARS-C: Modeling and Reduction of Soft Errors in Combinational Circuits. in Design Automation Conference, DAC'06, pp. 767 - 772.

[11] F. Corella, Z. Zhou, X. Song, M. Langevin, E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. Formal Method in System Design, Kluwer, 1997, pp. 7 - 46.

[12] S. R. Hasan, N. Belanger, Y. Savaria, M. O. Ahmad. Crosstalk Glitch Propagation Modeling for Asynchronous Interfaces in Globally Asynchronous Locally Synchronous Systems. IEEE Transaction on Circuits and Systems, Vol. 57, No. 8, August, 2010, pp. 2020 - 2031.

[13] F. Corella, Z. Zhou, X. Song, M. Langevin, E. Cemy, Miultiway Decision Graphs for Automated Hardware verification. Journal of Formal Methods in System Design. Available as IBM research report RC19676(87224), July 1994.

[14] Z. Zhou and N. Boulerice.MDG Tools (V1.0) User's Manual. University of Montreal, Dept. of Information and Operation Research. 1996.

[15] S. Brown and Z. Vranesic. Fundamentals of Digital Logic with VHDL Design. 2nd Edn., McGraw-Hill Higher Education, USA., ISBN: 0072499389, pp: 939. 2005.

[16] J. F. Wakerly. Digital Design-Principles and Practices. 4th Edn., Pearson Prentice Hall, USA., ISBN: 0132128381, pp: 859. 2006.

[17] M. Morris Mano. Computer System Architecture. 3rd Edn., Prentice Hall, USA., ISBN: 0-13-175738-5, pp: 525. 1993.

[18] M. Morris Mano. Digital Design. 3rd Edn., Prentice Hall, USA., ISBN: 0-13-062121-8.2002.

[19] C. Keawsai, K. Sripimanwat and A.Lasakul. Modified register exchange method of viterbi decoder for 3GPP mobile system. ECTI, Pataya, Thailand. 2004 $http://www.kmitl.ac.th/dslabs/download/paper/viterbidecoder_ecti2004.pdf$

[20] L. Zhang, Y.J. Cheng and X. Zhou. Rate avalanche: Effects on the performance of multi-rate 802.11 wireless networks. Simulat, 2009 Model. Pract. Theor., 17: 487-503. DOI: 10.1016/j.simpat.2008.09.003

[21] R. Williams. Computer Systems Architecture. Addison-Wesley, ISBN: 0201648598. 2001.

[22] P. Hazucha and C. Svensson. Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate.IEEE Transactions on Nuclear Science, Vol. 47, No. 6, pages2586-2594, Dec. 2000.

[23] S. Hauck, S. Burns, G. Borriello, C. Ebeling. An FPGA For Implementing Asynchronous Circuits. IEEE Design & Test of Computers, Vol. 11, No. 3, pp. 60-69, Fall, 1994.

[24] F. Corella, M. Langevin, E. Cerny, Z. Zhou and X. Song State enumeration with abstract descriptions of state machines. In Proc. IFIP WG 10.5, 1995.

[25] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In Proceedings of the international

workshop on Automatic verification methods for finite state systems, pp. 365 373, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[26] Z. Zhou, X. Song, F. Corella, E. Cerny, and Mi. Langevin. Description and verification of RTL designs using multiway decision graphs. pp.575 - 580. 1995.

[27] Y. Xu. MDG model checker users manual. Technical report, 1999.

[28] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. in Proceedings of an International Symposium on the Theory of Switching, pp. 204-243 Harvard University Press, Apr. 1959

[29] J. Muttersbach, T. Villiger, and W. Fichtner. Practical design of globally-asynchronous locally-synchronous systems. in Proc. 6th Int. Symp. Adv. Res. ASYNC, Apr. 2000, pp. 52-59.

[30] B. Zhang, W. Wang, M. Orshansky. FASER: Fast Analysis of Soft Error Susceptibility for Cell-Based Designs. pp.755-760, 7th International Symposium on Quality Electronic Design (ISQED'06), 2006.

[31] M. Omana, G. Papasso, D. Rossi, and C. Metra. A Model for Transient Fault Propagation in Combinatorial Logic. In Proc. of the 9th IEEE International On-Line Testing SymposiumI, OLTS03, pp. 111-115, July 2003.

[32] C. Boleat,G. Colas. Overview of soft errors issues in aerospace systems. On-Line Testing Symposium, 2005. IOLTS 2005. 11th IEEE International. pp. 299-302

[33] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson. On Latching Probability of Particle Induced Transients in Combinational Networks. In Proc. of Fault-Tolerant Computing Symposium, pp. 340-349, 1994.

[34] J.F. Ziegler et al. IBM experiments in Soft Fails in Computer Electronics (1978-1994). In IBM Journal of Research and Development, Vol 40, pp. 3-18, 1996.

[35] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim. Robust System Design with Built-In Soff-Error Resilience. IEEEC omputeMr agazine,V ol. 28, No. 2, pp. 43-52, February 2005.

[36] Y. S. Dhillon, A. U. Diril, and A. Chatterjee. Soft-Error Tolerance Analysis and Optimization of Nanometer Circuits. In Proc. of Design, Automation and Test in Europe (DATE), pp. 288-293, March 2005.

[37] S. Krishnaswamy, G. F. Viamonte, I. L. Markov, and J. P. Hayes. Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices. In Proc. of Design, Automation and Test in Europe (DATE), pp. 282-287, March 2005.

[38] R.C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. In IEEE Transactions on Device and Materials Reliability Volume 5, Issue 3, Sept. 2005, pp. 305 - 316.

[39] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic. Proc. Intel Conf. Dependable Systems and Networks, pp. 389-398, June 2002.

[40] T. Karnik, P. Hazucha, and J. Patel. Characterization of Soft Errors Caused by Single Event Upsets in CMOS Process. IEEE Trans. Dependable and Secure Computing, vol. 1, no. 2, pp. 128-143, April-June 2004.

[41] C. Frieda, R. Manohar. Fault Detection and Isolation Techniques for Quasi Delay-Insensitive Circuits. In Proc. of International Conference on Dependable Systems and Networks, Italy, June 28 - July 01, 2004.

[42] Harling. Embedded DRAM Has a Home in the Network Processing World. Integrated System Design, 3 August 2001.

[43] T. Karnik, , P. Hazucha. Characterization of soft errors caused by single event upsets in CMOS processes. Dependable and Secure Computing, IEEE Transactions on. pp.128 - 143. April-June 2004

[44] S. Almukhaizim, S. Feng, E. Love, Y. Makris. Soft-Error Tolerance and Mitigation in Asynchronous Burst-Mode Circuits. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. pp. 869 - 882, July 2009.

[45] K. Mohanram and N. A. Touba. Cost-effective approach for reducing soft error rate in logic circuits. In IEEE Int. Test Conf., 2003, pp. 893901.

[46] S. Almukhaizim, Y. Makris, Y.-S. Yang, and A. Veneris. Seamless integration of SER in rewiring-based design space exploration. In IEEE Int. Test Conf., 2006, pp. 29.3.129.3.9.

[47] B. Vaidyanathan, Y. Xie, V. Narayanan, H. Zheng. Soft Error Analysis and Optimizations of C-elements in Asynchronous Circuits. The Second Workshop on System Effects of Logic Soft Errors (SELSE), 2006.

[48] W.J. Bainbridge, W.B. Toms, D.A Edwards, S.B. Furber. Delay-insensitive, point-to-point interconnect using m-of-n codes. Asynchronous Circuits and Systems, 2003. Proceedings. Ninth International Symposium on. May, 2003. pp. 132 - 140.

[49] S.R. Hasan, N. Belanger, Y. Savaria, M.O. Ahmad. Crosstalk-Glitch Gating: A Solution for Designing Glitch-Tolerant Asynchronous Handshake Interface Mechanisms for GALS Systems. Circuits and Systems I: Regular Papers, IEEE Transactions on. pp. 2696 - 2707. 20 May 2010.

[50] M. Abramovici, M. A. Breuer, and A. D. Friedman. Digital Systems Testing and Testable Designs. Piscataway, NJ: IEEE Press, 1990.

[51] J. P. Roth. Diagnosis of automata failures: A calculus and a method. IBM J. Res. Develop., vol. 10, no. 4, pp. 278291, Jul. 1966.

[52] M. J. Bellido-Diaz, J. Juan-Chico, A. J. Acosta, M. Valencia, and J.L.Huertas. Logical modelling of delay degradation effect in static CMOS gates. IEEE Proc-Circuits Devices Syst., 147(2):107117, April 2000.

[53] Y. Monnet, M. Renaudin, and R. Leveugle. Asynchronous circuits transient faults sensitivity evaluation. in DAC 2005, Anaheim, CA, Jun. 2005, pp. 863868.

[54] Y. Monnet, M. Renaudin, and R. Leveugle. Asynchronous circuits sensitivity to fault injection. In Proc. 10th IEEE Int. On-Line Testing Symp., 2004, pp. 121126.

[55] Y. Monnet, M. Renaudin, and R. Leveugle. Hardening techniques against transient faults for asynchronous circuits. in Proc. 11th IEEE Int. On-Line Test. Symp., 2005, pp. 129134.

[56] J.F. Ziegler. Terrestrial cosmic rays. IBM Journal of Research and Development, Vol. 40, no. 1, pp. 19-40, Jan 1996.

[57] R. C. Baumann. The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In Digest of International Electron Devices Meeting, 2002, pp. 329332.

[58] S. Rasmi H. Abed. The Verification of MDG Algorithms in the HOL Theorem Prove. PhD thesis, Concordia University, Canada, 2008.