Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

Spring 6-30-2014

Probabilistic Analysis for Reliable Logic Circuits

Scott Blakely
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

 Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Recommended Citation

Blakely, Scott, "Probabilistic Analysis for Reliable Logic Circuits" (2014). *Dissertations and Theses.* Paper 1860.
https://doi.org/10.15760/etd.1859

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Probabilistic Analysis for Reliable Logic Circuits

by

Scott Blakely

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Xiaoyu Song, Chair
Marek Perkowski
Jingke Li

Portland State University
2014

©2014 Scott Blakely

Abstract

Continued aggressive scaling of electronic technology poses obstacles for maintaining circuit reliability. To this end, analysis of reliability is of increasing importance. Large scale number of inputs and gates or correlations of failures render such analysis computationally complex. This paper presents an accurate framework for reliability analysis of logic circuits, while inherently handling reconvergent fan-out without additional complexity. Combinational circuits are modeled stochastically as Discrete-Time Markov Chains, where propagation of node logic levels and error probability distributions through circuitry are used to determine error probabilities at nodes in the circuit. Model construction is scalable, as it is done so on a gate-by-gate basis.

The stochastic nature of the model lends itself to allow various properties of the circuit to be formally analyzed by means of steady-state properties. Formal verifying the properties against the model can circumvent strenuous simulations while exhaustively checking all possible scenarios for given properties. Small combinational circuits are used to explain model construction, properties are presented for analysis of the system, more example circuits are demonstrated, and the accuracy of the method is verified against an existing simulation method.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

1.1.1 Shrinking Feature Size

Traditionally, transient errors in circuits have been a result of issues such as power supply instability and mismatches between components [13]. However, continued reduction of device size has led to increased transient errors due to undesirable interference.

Feature size reduction results in an increased amount of charge that is stored in circuit nodes. Coupled with reduced noise margins, this can render circuits more vulnerable to manufacturing defects, transient faults due to noise, and radiation interference [1]. Minority carriers are created when neutrons or alpha particles collide with silicon. Current pulses of short duration can result from the minority carriers if they are collected by a p-n junction [1]. A current pulse resulting from the collision, referred to as a single-event upset (SEU), may cause a bit flip in a combinational logic node or memory component, possibly resulting in a soft error. A SEU can cause a soft error if it is greater than a cell's critical charge. With continued decreased feature size, attenuation is likewise decreased when errors propagates through the circuit. Lower energy particles, another byproduct of reduced feature sizes and voltage levels, also attribute to the occurrence of SEUs [5].

CMOS device dimensions are estimated to reach near the design limit of 50 nm by 2020 [12]. Exhausting the scalability of silicon-based circuit has made research and development of nanoscale electronics a valuable and relevant field. Recently multiple logic circuit technologies have demonstrated feasible potential for use in the not too distant future. Nanoscale architecture will likely have to expect, at least for the present time, device and failure rates of 10% or more [14].

While performance and cost, and more recently power consumption, have been the predominant concerns for digital system design, technology scaling has caused a large emphasis to be placed on reliability evaluation. As an emerging technology, it can be expected that nanoscale electronic devices and their interconnections will have significantly more reliability issues than current CMOS devices. Failures can occur during and after fabrication, making it difficult to implement a singular test and repair procedure. Reduced device dimensions lead to increased susceptibility of various phenomena interference.

Having to work with the increasing levels of error probability, developing an understanding of the likelihood of errors at specific nodes can aid in better fault-tolerance, error reduction and improved design of the system.

1.1.2 Application to Architecture

As previously mentioned, a fault in a circuit can result from physical properties, design methodology, or use of operation. A manifestation of a fault is referred to as a soft error, which in a digital circuit is an incorrect Boolean logic state. If an error alters the intended functionality of the output of the design, it causes a failure. A fault does not necessarily result in a soft error, and an error does not necessarily result in a failure.

Some present architectures methods to handle reliability issues predominantly focus on the failure of the system, namely fault-tolerant design. This entails testing and routing around failures, incorporating extra circuit elements into a design that can be used if other devices and connections fail, essentially using redundant logic for error prevention.

An effective fault tolerant system is able to automatically surmount the effects of faults by utilizing redundant circuitry. It can continue operating with inconsequential or non-degradation to performance or undesired alteration of data. This excess circuitry results in degraded testability and ability to handle failures over the life of the system. The extra circuitry also results in extra overhead cost and has questionable ability to handle continuous failures over the life of the circuit. Therefore placement of redundant circuit must be utilized critically and ideally as minimally as possible without compromising the quality of the design. Redundant circuitry is inevitably likely to be

desired in locations where there is a high probability of error, and at nodes that are most critical to the accuracy of the output of the circuit. Likewise, minimizing unnecessary redundant circuitry is desirable at nodes where failure probability is low and/or the effect of an error at a given node is negligible to the accuracy of the output of the circuit.

1.2 Summary of Paper

Due to increased probability of error compared to existing circuitry, it can be beneficial to model nanoscale circuits as probabilistic instead of deterministic. To that end, this paper presents a framework for modeling and analyzing the reliability of a sequential circuit using Markov Chains and probabilistic model checking. The objective is to aid in effective estimation of soft error probability.

Probabilistic model checking is a type of formal verification. For stochastic systems, it is used for analyzing reliability and performance measures. It is comprised of two primary parts:

- the construction of a formal model of a real-life system to be analyzed
- formal properties to check against the model revealing qualities of the system

The formal model is a stochastic one, where measures are quantified probabilistically as opposed to deterministically. Errors at nodes in the circuits are a function of two probabilistic elements; logic levels of primary inputs to the circuit, and noise manifested as errors originating at the outputs of gates.

Primary inputs to a combinational logic block are assumed to have a given probability of being either a logic high and a logic low. This probability then propagates through the circuit structure to allow for any node logic level of the circuit to be expressed probabilistically. Based on assumed physical properties of the gate, there is a given probability of failure at the output of each gate after the logical computation of that gate. The probability of an error at a given gate can also affect the probability of error of other gates, as it can propagate through the circuit.

Using the described probabilistic elements, a stochastic model is constructed for the logic levels of nodes in the circuit, and stochastic models are constructed for error estimation at each gate output. The stochastic models take the form of discrete-time Markov chains (DTMCs).

Representing the circuit as stochastic model allows for analysis of the system via formal verification. The formal verification takes the form of property specifications that are constructed with probabilistic computation tree logic (PCTL), a probabilistic temporal logic used to quantify or prove or disprove attributes of the circuit. This framework allows flexibility of analysis based upon the properties that are composed. Properties can reveal a wide variety of attributes of the circuit, from observability of specific errors at individual nodes, to the ability to see how errors at specific nodes affect errors at other nodes.

Briefly, the primary contributions of this paper are:

- A thorough method for stochastically modeling error probability of nodes of combinational logic circuits. The model inherently allows for handling simultaneous errors and reconvergent fan-out.
- A framework for using properties to analyze the steady-state error probabilities of the stochastic model. The analysis is flexible, scalable, and can reveal the relationship of failures of a specified node to failures of nodes in fan-in cone of the gate of interest.

1.3 Paper Organization

Chapter 2 covers the state of the art, briefly looking at various existing methodologies for reliability analysis of circuits, specifically probabilistic analysis.

Chapter 3 details the presented framework for modeling combinational circuits. First, error representation for the model is detailed, and then the method for error estimation at nodes is presented.

Simple combinational logic circuits are used as examples to demonstrate the reliability estimation. Discrete-time Markov chains are introduced, as they are the implemented stochastic model used to represent the combinational circuits. Construction of DMTCs

for error probability estimation is explained, and examples combinational circuits are shown.

In Chapter 4, a framework for model checking of the system is overviewed. Generic properties that can be analyzed are presented and briefly discussed. The methodology for estimating steady-state probabilities using DTMCs is detailed. The probabilistic temporal language used to form properties to verify against the model is defined in terms of syntax and semantics, and what it means to 'satisfy' a property is explained.

Chapter 5 presents case-studies. With the DTMCs required for the model established, a brief introduction to the PRISM probabilistic model checker is presented, where the software syntax is shown and code for the previously mentioned simple combinational circuits are demonstrated. Specific examples of property specifications in PRISM are proposed to present a framework for model checking. The accuracy of the model is verified by taking a simple circuit and comparing the simple circuit implemented as the presented model versus implementing the circuit as a hard coded monolithic DTMC where each combination of logic levels and probabilities of nodes comprise a state. The paper concludes with Chapter 6 briefly summarizing the work and presenting future possibilities. Compliable PRISM code for models of some of the examples circuits presented in the paper are included in the Appendices.

Chapter 2: State of the Art

Some recent approaches to reliability analysis for CMOS circuits have used a variety of probabilistic methods. [1] presents two algorithms that each assume independent gate failure. The first uses observability, as it quantifies the effect of a gate failure on the circuit output. It constructs a closed-form expression for the reliability of a circuit as a function of failure probabilities and observabilities of the gates. The method can be useful when single gate errors are more common than multiple gate errors. However, the primary drawback to the method is the limitation of not being able to model multiple simultaneous errors.

The second algorithm presented in [1] is a single-pass algorithm that is able to model multiple simultaneous errors. Gates are sorted topologically and processed in a single 'pass' from the inputs to the outputs. Sorting topologically allows for the effects of multiple gates errors in the transitive fan-in cone of a gate to be calculated at the input of the gate. The cumulative effect of errors at a gate output are estimated using (1) the joint signal probability distribution of the gate input, (2) the propagated error probabilities from the transitive fan-in at the gate inputs, and (3) the failure probability of the gate. Correlation coefficients are introduced to handle the possibility of reconvergent fan-out. Inevitably, the coefficients add an undesirable additional level of complexity to the algorithm.

In addition to the second algorithm, [1] also presents an upper bound that can be used on the single-pass algorithm to limit the maximum number of simultaneous gate failures. For a circuit, a sample space is of size $O(N^k)$, where N is the number of gates, and k is the upper bound for number of possible simultaneous gate failures.

[2] uses Binary Decision Diagrams (BDDs) and Algebraic Decision Diagrams (ADDs) for symbolic reliability analysis on combinational circuits. It focuses on analyzing the probability that a transient fault will manifest itself as an error on the output of the circuit.

[13] demonstrates reliability analysis through use of probabilistic transfer matrices (PTMs). Individual PTMs for gates interact to form an overall PTM for the circuit. This allows for the extraction and analysis of output probabilities, overall probability of error, and signal observability. The PTM method allows for simultaneous computations over all possible input combinations, calculating exact error probabilities, as opposed to relying on estimations from vector sampling. PTMs allow flexibility, and circuits are modeled at the logic level, using matrices for gates. As the probabilities of BSC errors for each gate are independent, the matrices can model a range of errors such as transient errors or stuck-at-faults.

However, to generate a PTM for an entire circuit, PTMs for gates or subcircuits are combined using tensor products of the individual PTMs. This results in extremely large matrices as if there are two matrices M1 and M2 with dimensions m × n and p × q, the resulting tensor product of M1 and M2, M1 $\otimes$M2, is an mp × nq matrix.

[14] presents a method for using Bayesian networks that can model large circuits. The Bayesian network is directed acyclic graph comprised of a set of random variables and their connectional dependencies. Gates are modeled using a conditional probability table (CPT). This models the probability of gate output signals being at a given Boolean logic state given the state of the input signal. The significant drawback to the Bayesian networks model is the undesired complexity from the inherently large CPTs.

[15] and [16] present methods using Markov Random fields (MRF). In the place of Boolean logic, a model is constructed of energy distribution functions derived from Gibbs distribution. Probabilities of energy levels at gate inputs and interconnects (a message passing algorithm) propagates the probability distributions from the inputs to the outputs of the Boolean system. [16] has MATLAB-based libraries for logic gates that calculate error probability distributions at nodes for specific input distributions. The MRFs model requires the minimization of a Gibbs distribution function with a significant number of variables for multilevel logic circuits. The resulting undesirable issue of the method, as with the PTM and the Bayesian models, is the complexity.

Chapter 3: Probabilistic Modeling for Reliability

3.1 Gate Error Estimation

3.1.1 Errors Due to Physical Attributes

Noise in a circuit can be caused by various sources, e.g. physical attributes of the circuit, such as crosstalk, terrestrial cosmic radiation, electromagnetic interference, etc. A binary symmetric channel (BSC) is used to model the noise at the output of each gate in a circuit. In the BSC model, a transmitter sends a binary bit, and a receiver receives the bit. The bit is expected to be received as it was sent. However, there exists a possibility that the polarity of the bit is erroneously flipped during the transmission [6], resulting in a soft error at the output of the gate.

For a given gate g, $g_{0\to1} \in \{0, 1\}$ represents the presence or absence of a soft error originating at the output of a gate due to noise, where a logic low was expected and the noise induced BSC results in a logic high. Likewise, $g_{1\to0} \in \{0, 1\}$ is the presence or absence of a soft error originating at the gate output due to noise, where a logic high was expected and the noise inducted BSC results in a logic low. $g_{0\to1} = 0$ and $g_{1\to0} = 0$ represent the absence of an error, while $g_{0\to1} = 1$ and $g_{1\to0} = 1$ represent the presence of an error.

3.1.2 Error at the Output of a Gate

For a given gate, there is not only the possibility of an error occurring at the output of the gate, but the possibility of an error propagating through from the fan-in cone of the gate to the output. If there is an error originating at the output of a gate, and an error propagated through to the output of the gate, their effects will advantageously cancel each other out, leaving the resulting output of the gate the same as intended error free value.

For a given gate, g, $\varepsilon g \in \{0, 1\}$, is an error-indicating variable, or comparator, for the error-incorporated gate output to the intended error-free logic level of the gate output. $\varepsilon g=0$ is the case where the error-incorporated output of gate g is the same as the error-free output. $\varepsilon g=1$ is the case where the error possibility-incorporated output of gate g is the same as the intended error-free output. εg is the exclusive disjunction between an error originating at the output of the gate and an error propagating through to the output of the gate from the input fan-in cone of the gate.

3.1.3 Examples

Figure 1: Inverter *b*, with primary input *a*

Figure 1 depicts an inverter with primary input *a,* and output *b*. As *a* is a primary input, it is assumed that there exists no possibility of an error (e.g. εa) propagating from the fan-in cone of *b* through to the output of *b*. $b_{0\to1}$ and $b_{1\to0}$, originating at the output of *b*, are noise dependent and are estimated based upon physical properties. εb, is the error-indicating variable, indicating the presence or absence of a resulting error at *b*. In Table 1, the "Error-free b" entry indicates the intended deterministic values of *b*, while the "Error-incorporated b" entry is the value of *b* taking into consideration the possibility of error.

From Table 1, it can be observed whenever $b_{0\to1}$=1 or $b_{1\to0}$ = 1, then εb=1. i.e. $\varepsilon b = b_{0\to1}/b_{1\to0}$

a	$b_{0\to1}/b_{1\to0}$	Error-free b	Error-incorporated b	εb
0	- / 0	1	1	0
0	- /1	1	0	1
1	0 / -	0	0	0
1	1 / -	0	1	1

Table 1: εb from inverter in Figure 3

Figure 2 depicts inverter *b,* feeding into a second inverter *c*.

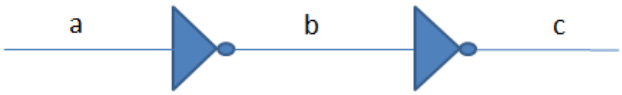


Figure 2: Two inverter circuit

As *b* is not a primary input, there exists a possibility it is propagating an error, εb, from the input fan-in cone of gate *c,* through to the output of *c.* εc, is the variable that represents the presence or absence of an error occurring at the output of *c,* regardless of whether is it a result of a propagated error from the fan-in cone of inverter *c,* or an error originating at the output of c, $c_{0\to1}$ or $c_{1\to0}$,.

εc, the error indicating variable at the output of gate can be expressed directly as a functions of:

- the input error-free logic levels of node in its fan-in cone, *b*
- the error-indicating variables of node in its fan-in cone, εb
- the probability of an error occurring at the output of gate *c*, $\Pr(c_{0\to1})$ /$\Pr(c_{1\to0})$

The origin of an error occurring at the output of c*,* is either from the fan-in cone εb*,* or from the output originating at the output of *c*, $c_{0\to1}=1/c_{1\to0}=1$. As shown in Table 2, if either the propagated error or the error originating at the output occur, it results in $\varepsilon c=1$.

But if both types of errors occur simultaneously, they advantageously cancel each other out, and εc=0. i.e. $\varepsilon c = \varepsilon b \oplus c_{0\to1}/c_{1\to0}$.

Error-free b	εb	$c_{0\to1}/c_{1\to0}$	Error-free c	Error-incorporated c	εc
0	0	-/0	1	1	0
0	0	-/1	1	0	1
0	1	0/-	1	0	1
0	1	1/-	1	1	0
1	0	0/-	0	0	0
1	0	1/-	0	1	1
1	1	-/0	0	1	1
1	1	-/1	0	0	0

Table 2: Resulting *εc* for Two Inverter Circuit

For another example, two-input NAND gate *d* is shown in Figure 3, with primary inputs *b* and *c*.

Figure 3: Two-input NAND gate *d* with primary inputs *b* and *c*

Since both inputs are primary inputs, there is never a case where an error is propagated from the input fan-in cone of *d* to the output of *d*. Therefore, εd=1 only occurs whenever only when $d_{0\to1}$=1 or $d_{1\to0}$=1, as shown in Table 3.

b	c	$d_{0\to1}/d_{1\to0}$	Error-free d	Error-incorporated d	εd
0	0	- / 0	1	1	0
0	0	- /1	1	0	1
0	1	- / 0	1	1	0
0	1	- /1	1	0	1
1	0	- / 0	1	1	0
1	0	- /1	1	0	1
1	1	0 / -	0	0	0
1	1	1 / -	0	1	1

Table 3: Resulting *εd* for Two-input NAND gate *d*.

As depicted in Figure 4, when an inverter is added to the circuit to feed into node c, it carreis with it the posibilty of an error at its output, *εc*. *εc* can affect the probability of the occurrence of *εd*, since it could propagate to the output of *d*, or cancel out either $d_{0\to1}$ or $d_{1\to0}$.

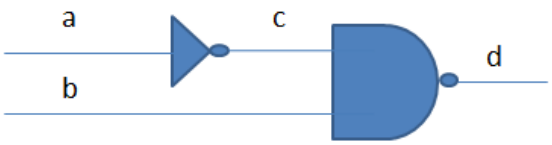


Figure 4: Invert and two-input NAND gate circuit

In the two inverter circuit example, whenever there was an error at the output of the first inverter, it would always propagate through to the output of the second inverter. For the circuit in Figure 4, when *εc*=1, it does not always propagate through to the output of *d*. As shown in Table 4, whenever *b*=0, regardless of the logic level of error-free *c*, when

εc=1, the error is not propagated to the output of *d*, because prior to considering the any error originating at the output of *d*, *d* will always be a logic high. εd can be expressed as

$$\varepsilon d = (b \wedge \varepsilon c \wedge \neg d_{0\to1}/\neg d_{1\to0}) \vee ((\neg b \vee \neg \varepsilon c) \wedge d_{0\to1}/d_{1\to0})$$

This example demonstrates that not all errors propagate from the fan-in cone to the output of a given gate.

b	**Error-free c**	**εc**	**$d_{0\to1}/d_{1\to0}$**	**Error-free d**	**Error-incorporated d**	**εd**
0	0	0	-/0	1	1	0
0	0	0	-/1	1	0	1
0	0	1	-/0	1	1	0
0	0	1	-/1	1	0	1
0	1	0	-/0	1	1	0
0	1	0	-/1	1	0	1
0	1	1	-/0	1	1	0
0	1	1	-/1	1	0	1
1	0	0	-/0	1	1	0
1	0	0	-/1	1	0	1
1	0	1	0/-	1	0	1
1	0	1	1/-	1	1	0
1	1	0	0/-	0	0	0
1	1	0	1/-	0	1	1
1	1	1	-/0	0	1	1
1	1	1	-/1	0	0	0

Table 4: Resulting *εd* for inverter and two-input NAND gate circuit

3.2 DTMCs

Discrete-time Markov chains (DTMCs) are essentially state transition diagrams augmented with probabilities. They are comprised of a set of states that represent possible configurations of the system. Transitions between the states are probabilistic and determined by discrete probabilistic distributions.

DTMCs are a memoryless random processes, where the next state is dependent only upon the current state and not on previous states. A DTMC can formally be defined as a tuple (S, s_{init}, **P**, L) where:

- S is a set of states, (e.g. {s_0, s_1, …, s_{n-1}}, where *n* is the number of states)
- $S_{init} \in S$ is the initial state
- P: S ×S →[0,1] is the transition probability matrix

  where $\forall s \in S \rightarrow \sum_{s' \in S} P(s,s') = 1$

- L : S →2^{AP} is function labeling states with atomic propositions [4]

Since **P** is a stochastic matrix:

- $s,s' \in S \rightarrow \mathbf{P}(s,s') \in [0,1]$ (5)
- $s \in S \rightarrow \Sigma_{s' \in S} \mathbf{P}(s,s') = 1$ (6)

Another way to represent a DTMC is as a series of random variables

$$\{X(k) \mid k=0, 1, \ldots\}$$

where X(k) is the state of the system at discrete-time step *k*.

Using this terminology, the Markov property, can be defined as

$$\Pr(X(k)= s_k \mid X(k-1)=s_{k-1}, \ldots, X(0)=s_{init}) = \Pr(X(k)= s_k \mid X(k-1)=s_{k-1}) \text{ [4]} \qquad (7)$$

This demonstrates the independence of the current state on the past. Equation (7) also assumes the DTMC is time-homogenous, where the probabilities are time independent.

3.3 Error Probability Estimation using DTMCs

For each gate in a combinational circuit, a DTMC is constructed to determine the error probability at the output of that gate. As previously established, the error probability at the output of a gate is a function of the error-free logic levels of the node(s) in its fan-in cone, the error-indicating variables of the nodes in its fan-in cone, and the presence or absence of an error originating at the output of the gate. A single DMTC is constructed for the sole purpose of estimating probabilities of error-free logic levels for every node in the circuit.

The states of a DTMC for the error probability of a logic gate are dependent upon both the states of the DTMC for the error-free logic levels of the nodes in the fan-in cone of the gate, and the states of the DTMCs for the error probability of the gates in the gate of interest's fan-in cone. This method of multiple DTMCs is used, as opposed to a single DTMC in order to make the model constructibly scalable. For a given combinational circuit, a single monolithic DTMC could be constructed to represent each unique combination of error-free node logic levels and error probabilities at the output of each gate, but this would render the model extremely difficult to create for large scale circuits.

3.3.1 DTMC for Error-Free Node Logic Level Estimation

Because a DTMC for the error probability of a given gate is dependent upon a DTMC for the error-free node logic level of a combinational circuit, the DTMC for the error free node logic levels of the circuit is explained first. The primary input(s) to the combinational circuit block are assumed to have known probabilities of being logic high and logic low. The probabilities from the primary inputs will propagate through to all the nodes in the circuit. The probability of a logic high occurring at a gate is simply the probability of the inputs to the gate being logic values that would result in a logic high at the output of the gate.

For the DTMC representing the error-free logic level of each node in a combinational circuit, each state is a unique combination of logic levels of all the nodes in the circuit. As logic levels are dictated by the values of the primary inputs, the number of possible states for the error-free node logic level DTMC is $2^{number\_of\_primary\_inputs}$.

Transitions between states are determined by the probabilities associated with the primary inputs being either a logic high or a logic low. i.e. the probability of transitioning to any state is the joint probability of the primary inputs associated with that state.

3.3.1.1 DTMC for Error-Free Node Logic Level Estimation Inverter Example

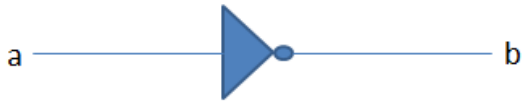


Figure 5: Inverter *b* with primary input *a*

For example of a DTMC for error-free node level logic estimation, we first start with the simplest possible circuit, a single inverter. Figure 5 shows inverter b, with primary input *a*. The DTMC is expressed as D = (S, s_{init}, **P**, L), where:

- S = {s_0, s_1}
- s_{init} = s_0 (arbitrary)

- P = $\begin{bmatrix} \Pr(\neg a) & \Pr(a) \\ \Pr(\neg a) & \Pr(a) \end{bmatrix}$

- AP = {¬a ∧ b, a ∧ ¬b}

    - $L(s_0)$ = {¬a ∧ b}

    - $L(s_1)$ = {a ∧ ¬b}

The Boolean values associated with each state are the logic levels of primary input *a* (e.g. state s_0 implies ¬*a*). As can be seen in the graphical representation of the DTMC in Figure 6, from either state, the DTMC transitions to the state s_0 = {¬a ∧ b}, with the probably of primary input *a*=0. Likewise, from either state, a transition to state s_1 = {a ∧ ¬b} occurs with the probability of primary input *a*=1.

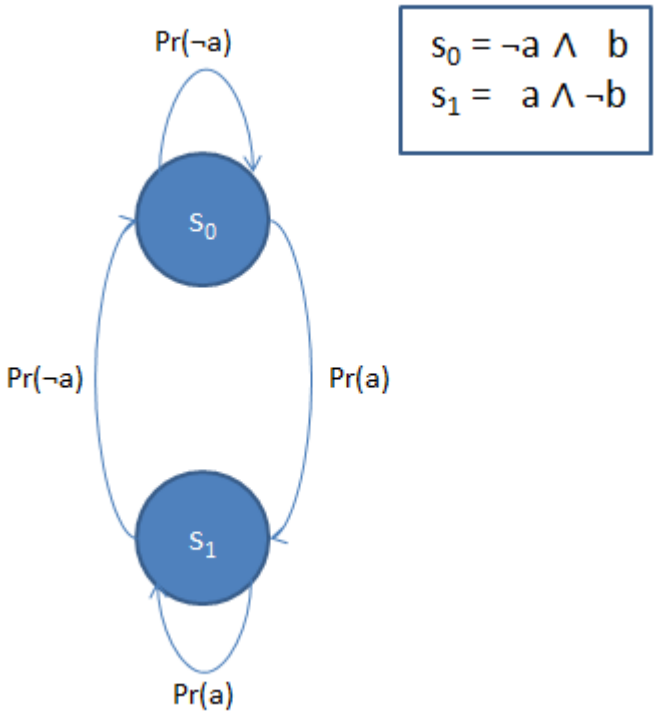


Figure 6: DTMC for error-free logic levels of inverter in Figure 5

3.3.2 DTMCs for Error Probability Estimation

For each gate in a combinational circuit, a DTMC is constructed for estimating the error probability at the output of that gate. For a given gate, a state of a DTMC minimally indicates the presence or absence of a resulting error at the output of that gate, where for a gate *g*, εg=0 represents the absence of an error, and εg=1 represents the presence of an error. For simplicity, this two state model is initially presented, but the number of states can be expanded to represent such characteristics such as:

- If there is an resulting error at the output of a gate, whether it is a 0→1 error or whether it is a 1→0 error.

- If there is a resulting error at the output of a gate, whether it originated at the output of the gate, or whether it propagated from the fan-in cone of the gate.

Transitions between the two states are comprised of unique combinations of the probabilities of node logic levels and errors of the inputs in the fan-in cone of a gate, as well as the probability of an error at the output of the gate. The type of the potential error originating at the output of a gate (e.g. $g_{0\to1,}$ or $g_{1\to0}$ for gate g) is dependent on node logic levels and error probabilities of the inputs in the fan-in cone of the gate. In the DTMCs for error probability estimation of each gate, this dependency is expressed using conditional probability. A transition of the model is expressed as given a certain combination of node logic levels and error probabilities from the input fan-in cone of the gate, that the probability of an absence or presence of an error originating at the output will determine to what state the model transitions.

The node logic levels of the inputs in the fan-in cone are derived from the error-free node level logic DTMC, and the error probabilities of the nodes in the fan-in gate, are derived from the DTMC for the error probability of the input node of mention. The probability of the absence or presence of an errors originating at the output of the gate

are noise dependent constants and are estimated based upon physical properties. e.g. for a given gate *g*, Pr($g_{0\to1}$) ∈ [0, 1].

3.3.2.1 DTMCs for Error Probability Estimation of Inverter Example

Inverter *b*, in Figure 5 is first used as an example of a DTMC for error probability estimation. Before constructing the DTMC, Table 5 is created to more clearly introduce the form of the expressions for transitions and states comprising the DTMC.

The DTMC for the error probability estimation of *b* has two states, $\varepsilon b=0$, and $\varepsilon b=1$. Transitions to these states are comprised of combinations of *a* and $b_{0\to1,}$ or $b_{1\to0.}$ As *a* is a primary input, there is no εa, an error that can be propagated through to the output from the fan-in cone of the gate. From any state, the disjunction of $\neg b_{1\to0} \mid \neg a$, and $\neg b_{0\to1} \mid a$ results in εb=0, and hence comprise the transition to the state εb=0. Likewise, the logical disjunction of $b_{1\to0} \mid \neg a$, and $b_{0\to1} \mid a$ results in εb=1, and hence comprises the transition to the state εb=1.

Expression results in εb=0	Expression results in εb=1
$\neg b_{1\to0} \mid \neg a$	$b_{1\to0} \mid \neg a$
$\neg b_{0\to1} \mid a$	$b_{0\to1} \mid a$

Table 5: Distinct expressions for error probability of inverter *b*

The DTMC for the error probability of *b*, *εb,* can be expressed as a tuple, $D_{\varepsilon b} = (S_{\varepsilon b}, s_{\varepsilon b\_init}, \mathbf{P}_{\varepsilon b}, L)$, where:

- $S_{\varepsilon b} = \{s_{\varepsilon b0}, s_{\varepsilon b1}\}$
- $s_{\varepsilon b\_init} = \varepsilon_{b0}$ (arbitrary)
- $\mathbf{P}_{\varepsilon b}$ =

$$\begin{bmatrix} (\Pr(\neg b_{1\to0})|\neg a) + (\Pr(\neg b_{0\to1})|a) & (\Pr(b_{1\to0})|\neg a) + (\Pr(b_{0\to1})|a) \\ (\Pr(\neg b_{1\to0})|\neg a) + (\Pr(\neg b_{0\to1})|a) & (\Pr(b_{1\to0})|\neg a) + (\Pr(b_{0\to1})|a) \end{bmatrix}$$

- AP = {¬εb , εb}
  - $L(s_{\varepsilon b0}) = \{\neg\varepsilon b\}$
  - $L(s_{\varepsilon b1}) = \{\varepsilon b\}$

This dependency of *εb* on the DTMC for *a* is depicted in Figure 7, as the logic level of *a* dictates whether $b_{1\to0}$ or $b_{1\to0}$.is needed for transitioning to *εb*=0 or *εb*=1. Figure 8 depicts the actual DTMC for *εb*, where conditional probability is used to express the dependency of state of $b_{1\to0}$ and $b_{1\to0}$.on the state of *a.*

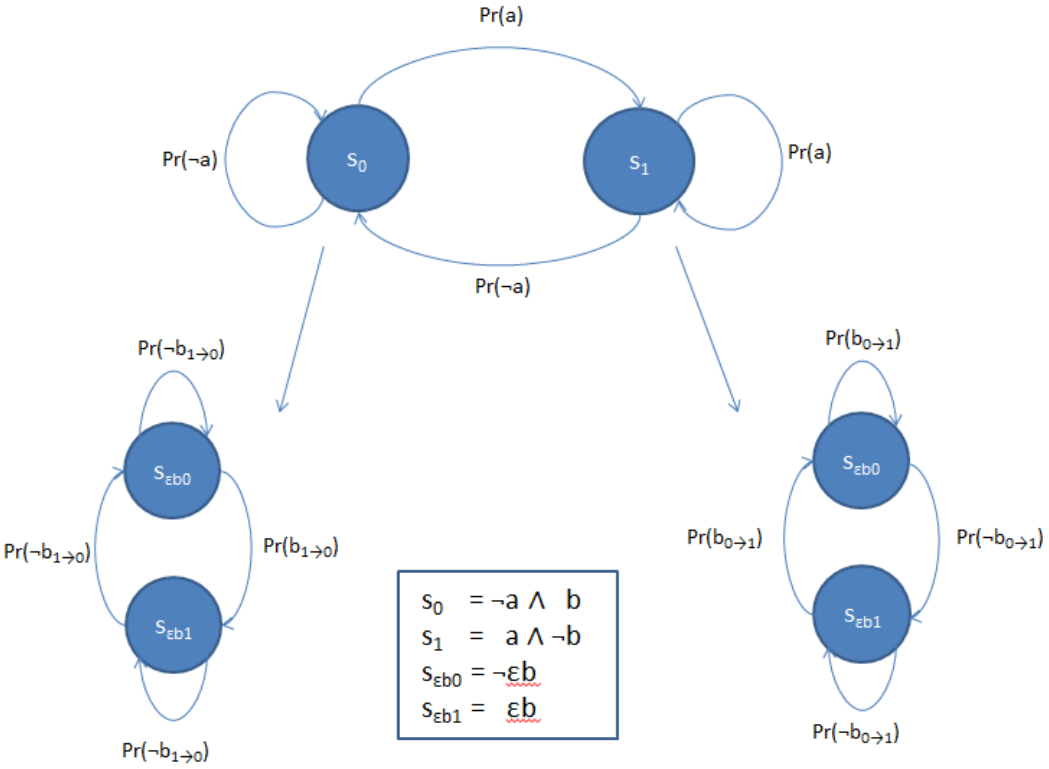


Figure 7: DTMCs for *εb*, the error probability of inverter *b*

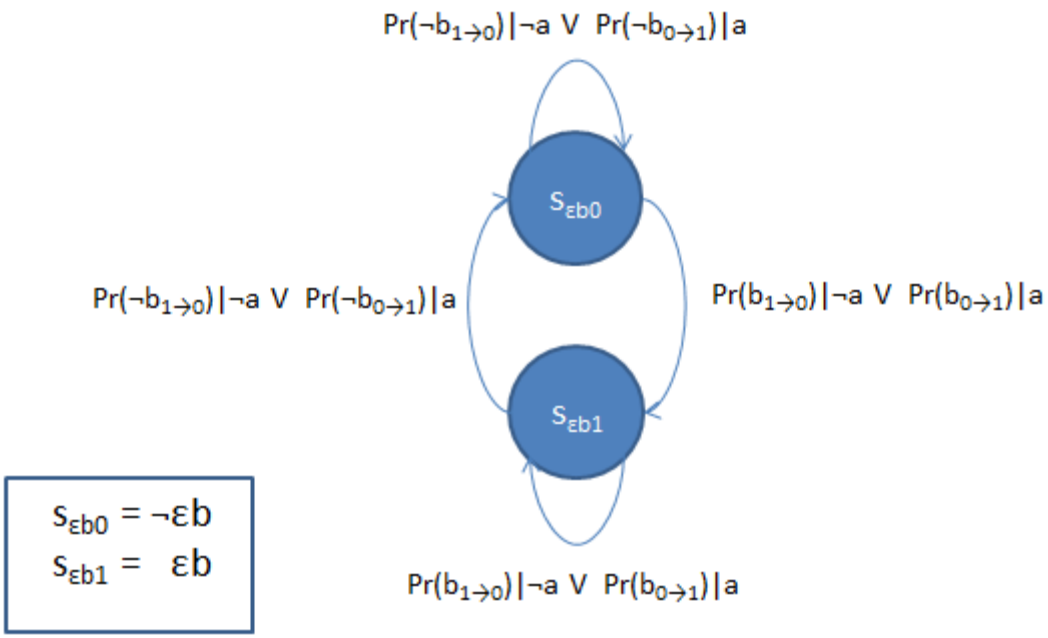


Figure 8: DTMC for *εb*, the error probability at the output of inverter *b*

The total number of states for the model is equal to $2^{(\text{number of primary inputs + number of gates})}$. In the case of the single inverter, $2^{(1+1)} = 4$ states.

3.3.2.2 DTMCs for Error Probability Estimation: Two Inverter Example

For the two inverter circuit shown in Figure 9, the DTMC for *εc*, the error probability at the output of inverter c, will not only depend on the DTMC for error-free logic levels of the circuit, but also the DTMC for *εb,* since an error at *b* could propagate to the output of *c*. The DTMC dependency is depicted in Figure 10.

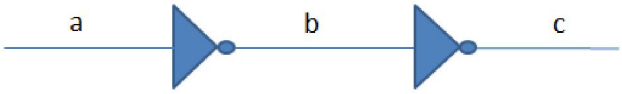


Figure 9: Two Inverter Circuit

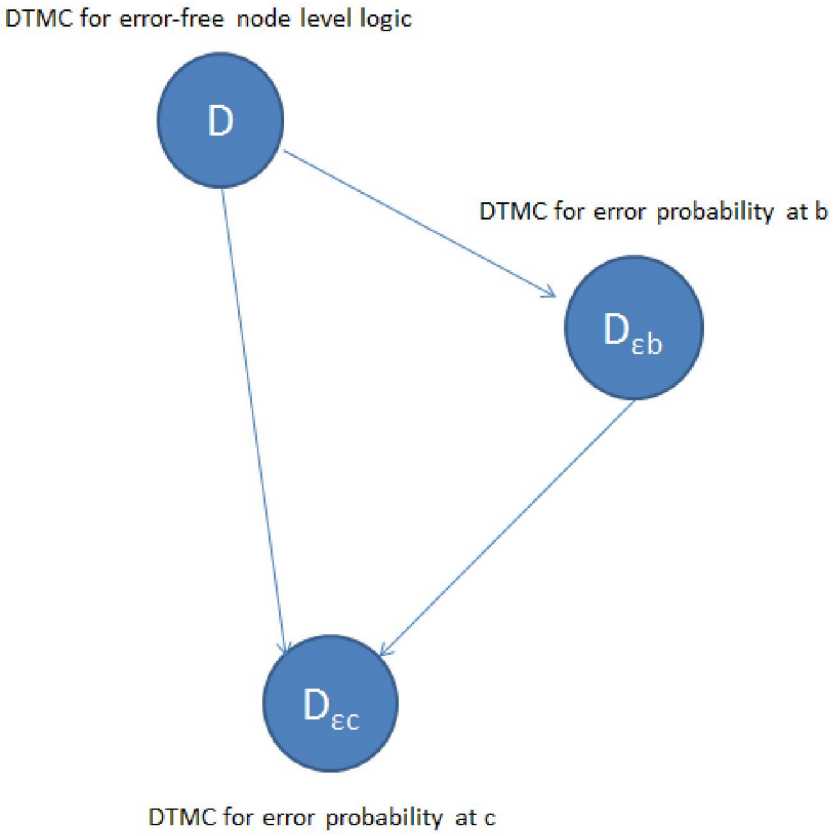


Figure 10: DTMC dependency for two inverter circuit in Figure 9

Since the DTMCs for εb and εc are both dependent on the DTMC for node level logic, we first construct the DTMC for node level logic. The DTMC for the error-free node logic level circuit is identical to the DTMC for the error-free node logic level circuit for the single inverter circuit in Figure 5, except the states include the variable for node *c*. The transitions remain unchanged, as they are solely dependent on the primary input to the circuit, which remains unchanged.

The DTMC is expressed as $D = (S, s_{init}, \mathbf{P}, L)$, where:

- $S = \{s_0, s_1\}$
- $s_{init} = s_0$ (arbitrary)
- $P = \begin{bmatrix} \Pr(\neg a) & \Pr(a) \\ \Pr(\neg a) & \Pr(a) \end{bmatrix}$
- $AP = \{\neg a \wedge b \wedge \neg c, a \wedge \neg b \wedge c\}$
    - $L(s_0) = \{\neg a \wedge b \wedge \neg c\}$
    - $L(s_1) = \{a \wedge \neg b \wedge c\}$

A graphical depiction of the DTMC for the error-free node logic levels of the two inverter circuit in Figure 9 is shown in Figure 11.

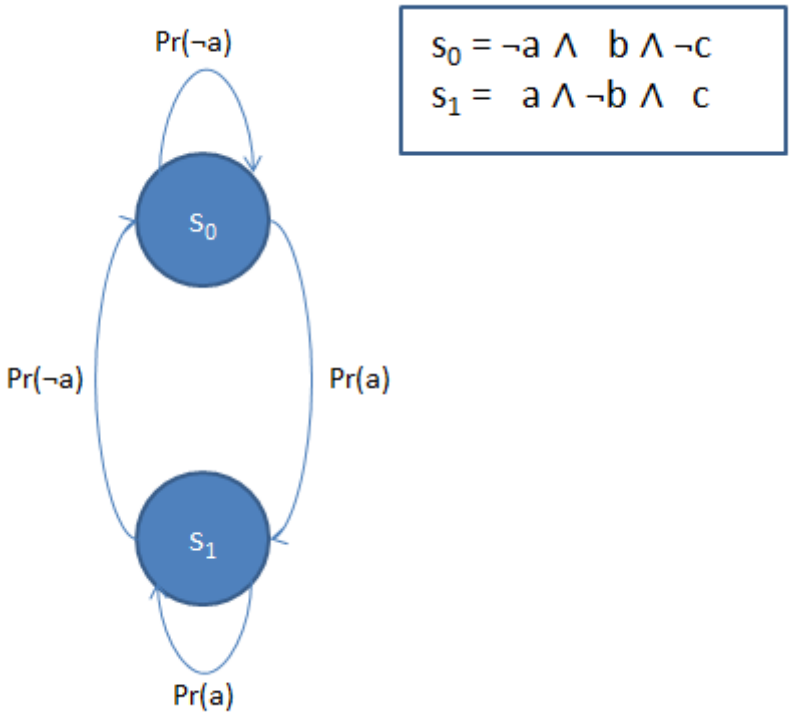


Figure 11: DTMC for error-free logic levels of two inverter circuit in Figure 6.

As construction of the DTMC for *εb* was already demonstrated when the single inverter circuit was examined, we proceed immediately to construction of the DTMC for *εc.* The DTMC for *εc* is constructed much the same as the DTMC for *εb,* with the primary difference being that the DTMC for *εc,* the error probability at the input of its fan-in gate, is one of the variables comprising transitions between states. As with Table 5 for *εb*, the left column of Table 6 shows expressions that result in ε*c*=0. Likewise, the expressions in the right column result in *εc*=1.

Expression results in εc=0	Expression results in εc=1
$\neg c_{1\to0} \mid (\neg b \wedge \neg \varepsilon b)$	$c_{1\to0} \mid (\neg b \wedge \neg \varepsilon b)$
$c_{0\to1} \mid (\neg b \wedge \varepsilon b)$	$\neg c_{0\to1} \mid (\neg b \wedge \varepsilon b)$
$\neg c_{0\to1} \mid (b \wedge \neg \varepsilon b)$	$c_{0\to1} \mid (b \wedge \neg \varepsilon b)$
$c_{1\to0} \mid (b \wedge \varepsilon b)$	$\neg c_{1\to0} \mid (b \wedge \varepsilon b)$

Table 6: Distinct expressions for error probability of inverter *c*

For the sole purposes of fitting more legibly into a transition probability matrix, the logical disjunction of the expressions that result in εc=0, is abbreviated as "Trsn_to_$s_{\varepsilon c0}$".

$$\text{Trsn\_to\_}s_{\varepsilon c0} = \Pr(\neg c_{1\to0}) \mid (\neg b \wedge \neg \varepsilon b) \vee \Pr(c_{0\to1}) \mid (\neg b \wedge \varepsilon b) \vee \Pr(\neg c_{0\to1}) \mid (b \wedge \neg \varepsilon b) \vee \Pr(c_{1\to0}) \mid (b \wedge \varepsilon b)$$

Likewise, the logical disjunction of the expressions that result in εc=1, is abbreviated as "Trsn_to_$s_{\varepsilon c1}$"

$$\text{Trsn\_to\_}s_{\varepsilon c1} = \Pr(c_{1\to0}) \mid (\neg b \wedge \neg \varepsilon b) \vee \Pr(\neg c_{0\to1}) \mid (\neg b \wedge \varepsilon b) \vee \Pr(c_{0\to1}) \mid (b \wedge \neg \varepsilon b) \vee \Pr(\neg c_{1\to0}) \mid (b \wedge \varepsilon b)$$

The DTMC for the error probability of *c*, εc, can be expressed as a tuple, $D_{\varepsilon c} = (S_{\varepsilon c}, s_{\varepsilon c\_init}, \mathbf{P}_{\varepsilon c}, L)$, where:

- $S_{\varepsilon c} = \{s_{\varepsilon c0}, s_{\varepsilon c1}\}$
- $s_{\varepsilon c\_init} = \varepsilon c_0$ (arbitrary)

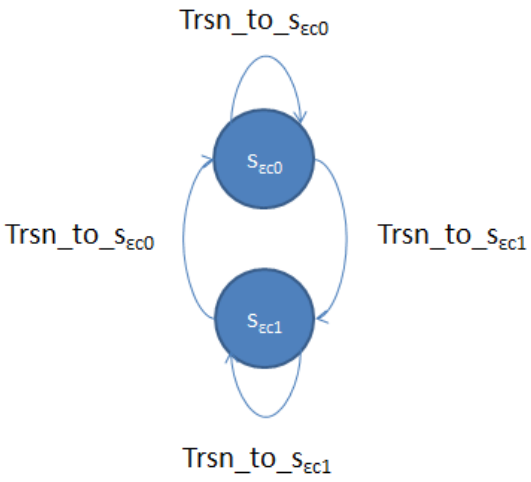
- $\mathbf{P}_{\varepsilon c} = \begin{bmatrix} \text{Trans\_to\_s}_{\varepsilon c0} & \text{Trans\_to\_s}_{\varepsilon c1} \\ \text{Trans\_to\_s}_{\varepsilon c0} & \text{Trans\_to\_s}_{\varepsilon c1} \end{bmatrix}$

- AP = {¬εc, εc}

    - $L(s_{\varepsilon c0}) = \{\neg\varepsilon c\}$

    - $L(s_{\varepsilon c1}) = \{\varepsilon c\}$

A graphical depiction of the DTMC for *εc* of the two inverter circuit is shown in Figure 12.



$\mathbf{s}_{\varepsilon c0} = \neg\varepsilon C$

$\mathbf{s}_{\varepsilon c1} = \varepsilon C$

$\mathbf{Trsn\_to\_s}_{\varepsilon c0} = Pr(\neg c_{1\to0}) | (\neg b \wedge \neg\varepsilon b) \vee Pr(c_{0\to1}) | (\neg b \wedge \varepsilon b) \vee Pr(\neg c_{0\to1}) | (b \wedge \neg\varepsilon b) \vee Pr(c_{1\to0}) | (b \wedge \varepsilon b)$

$\mathbf{Trsn\_to\_s}_{\varepsilon c1} = Pr(c_{1\to0}) | (\neg b \wedge \neg\varepsilon b) \vee Pr(\neg c_{0\to1}) | (\neg b \wedge \varepsilon b) \vee Pr(c_{0\to1}) | (b \wedge \neg\varepsilon b) \vee Pr(\neg c_{1\to0}) | (b \wedge \varepsilon b)$

Figure 12: DTMC for *εc*, the error probability at the output of inverter *c*

The total number of states for the model is equal to $2^{(\text{number of primary inputs + number of gates})}$. In the case of the two inverter circuit, $2^{(1+2)} = 8$ states.

3.3.2.3 DTMCs for Two-Input NAND Gate

A two input NAND gate *d* shown in Figure 13, with primary inputs *c*, and *b*, is used to show an example of constructing DTMCs for gate/circuits with multiple inputs.

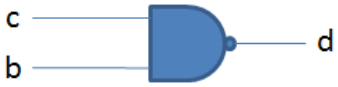


Figure 13: Two-Input NAND gate

As with the previous examples, first a DTMC for error-free node level logic is constructed. Each state of the DTMC comprises the logic levels in the circuit for a unique combination of primary inputs. Transitions to the states are the probabilities of unique combinations of the primary inputs. The resulting DTMC is expressed as D = (S, s_{init}, **P**, L), where:

- S = {s_0, s_{01}, s_{10}, s_{11}}
- $s_{init} = s_{00}$ (arbitrary)

- **P** = $\begin{bmatrix} \Pr(\neg c \wedge \neg b) & \Pr(\neg c \wedge b) & \Pr(c \wedge \neg b) & \Pr(c \wedge b) \\ \Pr(\neg c \wedge \neg b) & \Pr(\neg c \wedge b) & \Pr(c \wedge \neg b) & \Pr(c \wedge b) \\ \Pr(\neg c \wedge \neg b) & \Pr(\neg c \wedge b) & \Pr(c \wedge \neg b) & \Pr(c \wedge b) \\ \Pr(\neg c \wedge \neg b) & \Pr(\neg c \wedge b) & \Pr(c \wedge \neg b) & \Pr(c \wedge b) \end{bmatrix}$

- AP = {¬c ∧ ¬b ∧ d, ¬c ∧ b ∧ d, c ∧ ¬b ∧ d, c ∧ b ∧ ¬d}
    - $L(s_{00})$ = {¬c ∧ ¬b ∧ d}
    - $L(s_{01})$ = {¬c ∧ b ∧ d}
    - $L(s_{10})$ = {c ∧ ¬b ∧ d}
    - $L(s_{11})$ = {c ∧ b ∧ ¬d}

The Boolean values associated with each state are the logic levels of primary inputs *a* and *b* respectively (e.g. state s_{00} implies ¬*a* ∧ ¬*b*). Figure 14 is a graphical representation of the error-free node logic level DTMC.

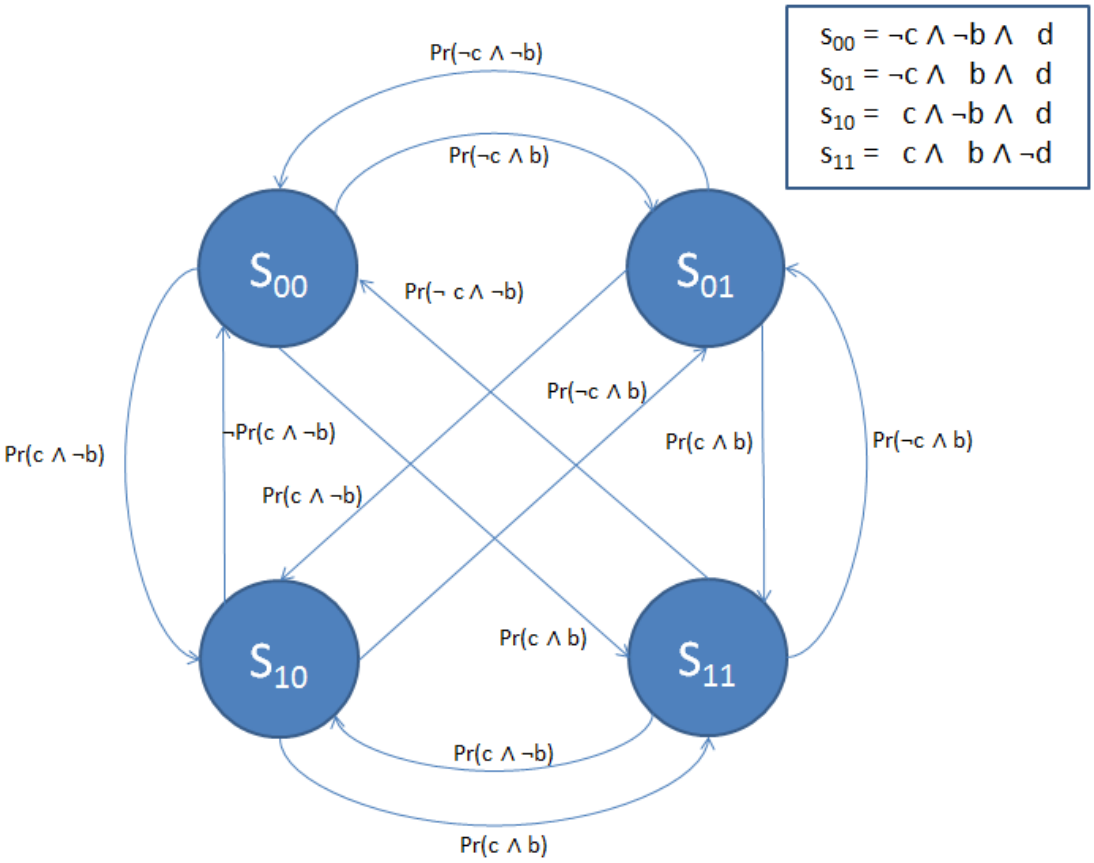


Figure 14: Error-free node logic level DTMC for the 2-input NAND gate *d.*

With the DTMC for error-free node level logic already constructed, we now construct the DTMC for εd. In Table 7, expressions resulting in εd=0 are shown in the left column, and expressions resulting in εd=1 in the right column. From either state, the logical disjunction of the expressions resulting in εd=0 comprises the transition to state $s_{\varepsilon d0}$. Likewise, from either state, the logical disjunction of the expressions resulting in εd=1 comprise the transition to state $s_{\varepsilon d1}$.

Expression results in εd=0	Expression results in εd=1
$\neg d_{1\to0} \mid (\neg c \vee \neg b)$	$d_{1\to0} \mid (\neg c \vee \neg b)$
$\neg d_{0\to1} \mid (c \wedge b)$	$d_{0\to1} \mid (c \wedge b)$

Table 7: *εd* for two-input NAND gate in Figure 12

The DTMC for the error probability of *d*, *εd,* can expressed as a tuple, $D_{\varepsilon d} = (S_{\varepsilon d}, s_{\varepsilon d\_init}, \mathbf{P}_{\varepsilon d}, L)$, where,

- $S_{\varepsilon d} = \{\varepsilon d_0, \varepsilon d_1\}$
- $s_{\varepsilon d\_init} = \varepsilon d_0$ (arbitrary)

$\mathbf{P}_{\varepsilon d}$ =

$$\begin{bmatrix} (\Pr(\neg d_{1\to0})|(\neg c \vee \neg b)) + (\Pr(\neg d_{0\to1})|(c \wedge b)) & (\Pr(d_{1\to0})|(\neg c \vee \neg b)) + (\Pr(d_{0\to1})|(c \wedge b)) \\ (\Pr(\neg d_{1\to0})|(\neg c \vee \neg b)) + (\Pr(\neg d_{0\to1})|(c \wedge b)) & (\Pr(d_{1\to0})|(\neg c \vee \neg b)) + (\Pr(d_{0\to1})|(c \wedge b)) \end{bmatrix}$$

- $AP = \{\neg\varepsilon d, \varepsilon d\}$
  - $L(s_{\varepsilon d0}) = \{\neg\varepsilon d\}$
  - $L(s_{\varepsilon d1}) = \{\varepsilon d\}$

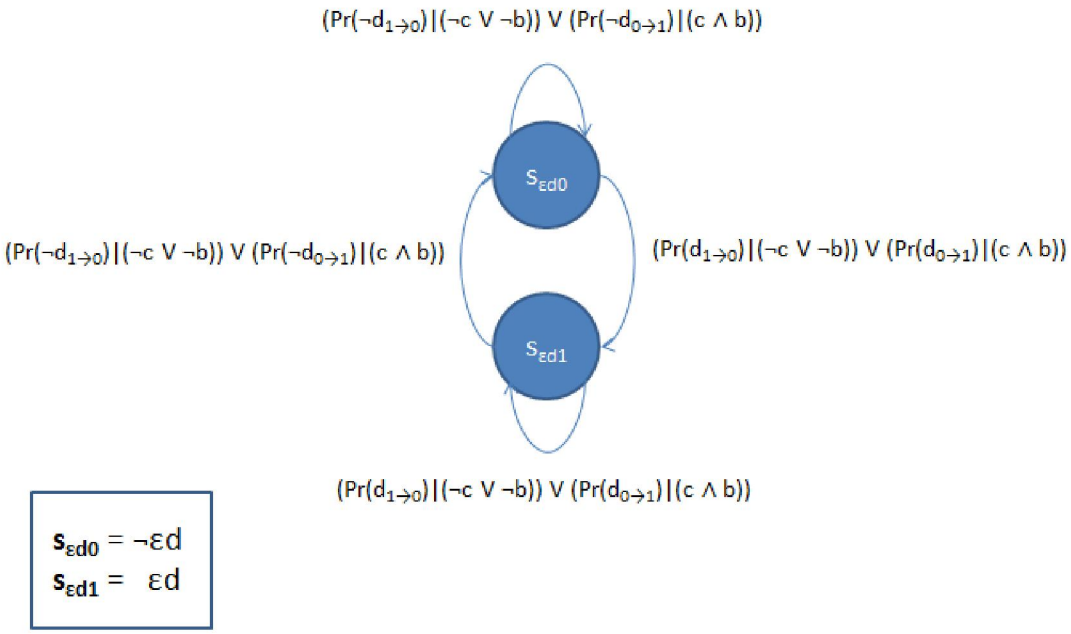


Figure 15: DTMC for *εd* of two-input NAND gate in Figure 13

The total number of states for the model is equal to $2^{(\text{number of primary inputs + number of gates})}$. In the case of the two input NAND gate, $2^{(2+1)} = 8$ states.

3.3.2.4 DTMCs for Inverter and Two-Input NAND Gate Circuit

Figure 16 depicts a simple combinational circuit, with primary inputs *a* and *b*.

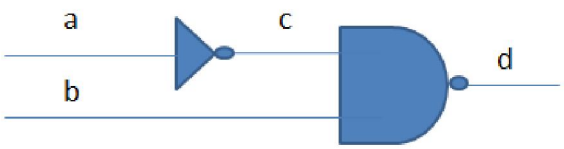


Figure 16: Inverter and two-input NAND circuit

The DTMC for the error-free node logic levels of the circuit is expressed as D = (S, s_{init}, **P**, L), where:

- S = {s_0, s_{01}, s_{10}, s_{11}}
- s_{init} = s_{00} (arbitrary)
- **P** = $\begin{bmatrix} \Pr(\neg a \wedge \neg b) & \Pr(\neg a \wedge b) & \Pr(a \wedge \neg b) & \Pr(a \wedge b) \\ \Pr(\neg a \wedge \neg b) & \Pr(\neg a \wedge b) & \Pr(a \wedge \neg b) & \Pr(a \wedge b) \\ \Pr(\neg a \wedge \neg b) & \Pr(\neg a \wedge b) & \Pr(a \wedge \neg b) & \Pr(a \wedge b) \\ \Pr(\neg a \wedge \neg b) & \Pr(\neg a \wedge b) & \Pr(a \wedge \neg b) & \Pr(a \wedge b) \end{bmatrix}$
- AP = {¬a ∧ ¬b ∧ c ∧ d, ¬a ∧ b ∧ c ∧ ¬d, a ∧ ¬b ∧ ¬c ∧ d, a ∧ b ∧ ¬c ∧ d}
    - L(s_{00}) = {¬a ∧ ¬b ∧ c ∧ d}
    - L(s_{01}) = {¬a ∧ b ∧ c ∧ ¬d}
    - L(s_{10}) = {a ∧ ¬b ∧ ¬c ∧ d}
    - L(s_{11}) = {a ∧ b ∧ ¬c ∧ d}

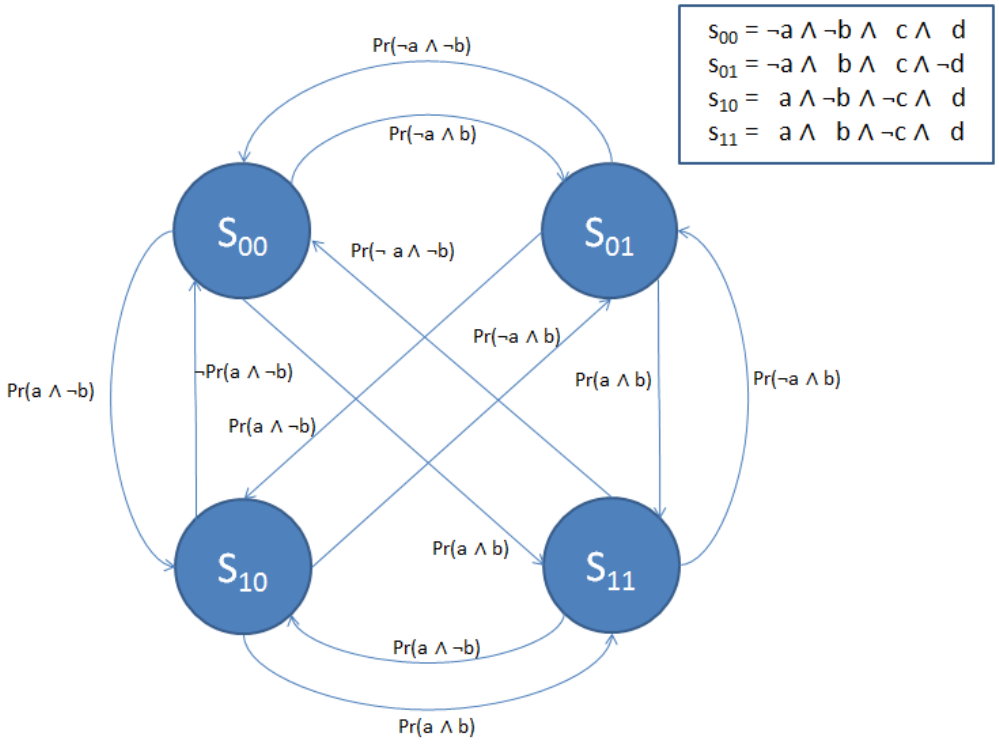


Figure 17: DTMC for error-free node logic levels of circuit in Figure 16.

With the DTMC for error-free node level logic already constructed, we now construct the DTMCs for *εc* and *εd.* The DTMC for inverter *εc* is constructed in the exact same way as the DTMC for the inverter from Figure 5, since the inputs to both the inverters are primary inputs. The DTMC for *εc* can then be expressed as a tuple, $D_{\varepsilon c} = (S_{\varepsilon c}, s_{\varepsilon c\_init}, \mathbf{P}_{\varepsilon c}, L)$, where:

- $S_{\varepsilon c} = \{s_{\varepsilon c0}, s_{\varepsilon c1}\}$
- $s_{\varepsilon c\_init} = \varepsilon_{c0}$ (arbitrary)
- $\mathbf{P}_{\varepsilon c} = \begin{bmatrix} (\Pr(\neg c_{1\to 0})|\neg a) + (\Pr(\neg c_{0\to 1})|a) & (\Pr(c_{1\to 0})|\neg a) + (\Pr(c_{0\to 1})|a) \\ (\Pr(\neg c_{1\to 0})|\neg a) + (\Pr(\neg c_{0\to 1})|a) & (\Pr(c_{1\to 0})|\neg a) + (\Pr(c_{0\to 1})|a) \end{bmatrix}$

- AP = {¬εc , εc}
    - $L(s_{\varepsilon c0}) = \{\neg \varepsilon c\}$
    - $L(s_{\varepsilon c1}) = \{\varepsilon c\}$

To aid in explaining the construction of the DTMC for *εd*, Table 8 shows expressions resulting in state *εd*=0 in the left column, and expressions resulting in state *εd*=1 in the right column. From either state, the logical disjunction of the expressions resulting in *εd*=0 comprises the transition to state $s_{\varepsilon d0}$. Likewise, from either state, the logical disjunction of the expressions resulting in *εd*=1 comprises the transition to state $s_{\varepsilon d1}$.

Expression results in εd=0	**Expression results in εd=1**
$d_{1\to0}$ | ((b ∧ ¬c ∧ ¬εc) V ¬b)	$d_{1\to0}$ | ((b ∧ ¬c ∧ ¬εc) V ¬b)
$d_{0\to1}$ | (b ∧ ¬c ∧ εc)	$\neg d_{0\to1}$ | (b ∧ ¬c ∧ εc)
$\neg d_{0\to1}$ | (b ∧ c ∧ ¬εc)	$d_{0\to1}$ | (b ∧ c ∧ ¬εc)
$d_{1\to0}$ | (b ∧ c ∧ εc)	$\neg d_{1\to0}$ | (b ∧ c ∧ εc)

Table 8: Distinct expressions for error probability of NAND *d*

For the sole purposes of fitting more legibly fit into a transition probability matrix, the logical disjunction of the expressions that result in *εd*=0, is abbreviated as "Trsn_to_$s_{\varepsilon d0}$".

$Trsn\_to\_s_{\varepsilon d0} = (Pr(\neg d_{1\to0}) | ((b \wedge \neg c \wedge \neg \varepsilon c) \vee \neg b))) \vee (Pr(d_{0\to1}) | (b \wedge \neg c \wedge \varepsilon c))$

$\vee (Pr(\neg d_{0\to1}) | (b \wedge \neg c \wedge \varepsilon c)) \vee (Pr(d_{1\to0}) | (b \wedge c \wedge \varepsilon c))$

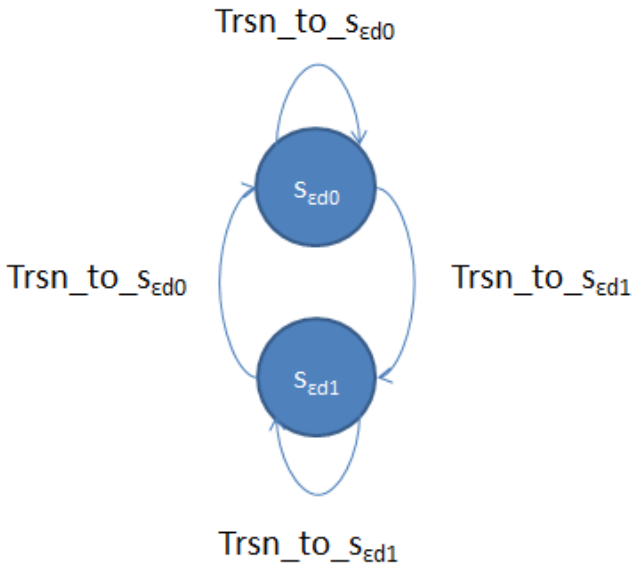
Likewise, the logical disjunction of the expressions that result in *εd*=1, is abbreviated as "$Trsn\_to\_s_{\varepsilon d1}$".

$Trsn\_to\_s_{\varepsilon d1} = (Pr(d_{1\to0}) | ((b \wedge \neg c \wedge \neg \varepsilon c) \neg b))) \vee (Pr(\neg d_{0\to1}) | (b \wedge \neg c \wedge \varepsilon c))$

$\vee (Pr(d_{0\to1}) | (b \wedge c \wedge \neg \varepsilon c)) \vee (Pr(\neg d_{1\to0}) | (b \wedge c \wedge \varepsilon c))$

The DTMC for the error probability of *d*, *εd,* can be expressed as a tuple, $D_{\varepsilon d} = (S_{\varepsilon d}, s_{\varepsilon dinit}, \mathbf{P}_{\varepsilon d}, L)$, where,

- $S_{\varepsilon d} = \{s_{\varepsilon d0}, s_{\varepsilon d1}\}$
- $\varepsilon d_{init} = \varepsilon d_0$ (arbitrary)
- $\mathbf{P}_{\varepsilon d} = \begin{bmatrix} \text{Trans\_to\_}s_{\varepsilon d0} & \text{Trans\_to\_}s_{\varepsilon d1} \\ \text{Trans\_to\_}s_{\varepsilon d0} & \text{Trans\_to\_}s_{\varepsilon d1} \end{bmatrix}$
- $AP = \{\neg \varepsilon d, \varepsilon d\}$
    - $L(s_{\varepsilon d0}) = \{\neg \varepsilon d\}$
    - $L(s_{\varepsilon d1}) = \{\varepsilon d\}$



$\mathbf{s_{\varepsilon d0}} = \neg\varepsilon d$

$\mathbf{s_{\varepsilon d1}} = \varepsilon d$

$\mathbf{Trsn\_to\_s_{\varepsilon d0}} = (Pr(\neg d_{1\to0}) \mid ((b \wedge \neg c \wedge \neg\varepsilon c) \vee \neg b))) \vee (Pr(d_{0\to1}) \mid (b \wedge \neg c \wedge \varepsilon c)) \vee (Pr(\neg d_{0\to1}) \mid (b \wedge \neg c \wedge \varepsilon c)) \vee (Pr(d_{1\to0}) \mid (b \wedge c \wedge \varepsilon c))$

$\mathbf{Trsn\_to\_s_{\varepsilon d1}} = (Pr(d_{1\to0}) \mid ((b \wedge \neg c \wedge \neg\varepsilon c) \vee \neg b))) \vee (Pr(\neg d_{0\to1}) \mid (b \wedge \neg c \wedge \varepsilon c)) \vee (Pr(d_{0\to1}) \mid (b \wedge c \wedge \neg\varepsilon c)) \vee (Pr(\neg d_{1\to0}) \mid (b \wedge c \wedge \varepsilon c))$

Figure 18: DTMC for *εd* from circuit in Figure 16

The total number of states for the model is equal to $2^{(\text{number of primary inputs + number of gates})}$. In the case of the two input NAND gate, $2^{(2+2)} = 16$ states.

3.4 Expanded DTMC State Model for Error Probability

The presented DTMCS for error probability of a gate have two states, reveling that at any step, the gate either has an error or is error-free. When there is an error, it may be desired to learn more about the error, for example:

- Whether the error was propagated to the output of the gate from the gate's fan-in cone, or if it originated at the output of the gate
- Whether the error was a logic low when a logic high was expected, or whether the error was a logic high when a logic low was expected.

To this end, a DMTC for error probability with additional variables is shown.

3.4.1 Set of States for Error-Free Output of Gate

A variable can be added to the DTCM for error probability indicating whether or not the error-free logic level of the node is supposed to be a logic low or logic high. For a gate, *g,* the set of states is referred to as *gef* ∈ {0, 1}, where *gef* =0 represents an intended logic low, and *gef*=1 represents an intended logic high.

3.4.2 Set of States for Propagated Error Only

A variable can be added for error probability indicating whether or not there is an error propagated through to the output of the node of interest. For a gate, *g*, the variable is referred to as *peg* ∈ {0, 1}, where *peg=0* represents the case where there is no propagated error, and *peg*=1 represents the case where there is a propagated error. The *pe* in *peg* stands for 'propagated error', and *g* is the name of the gate.

3.4.3 Two Inverter Example

Figure 20 depicts a two inverter circuit, with primary input *a*.

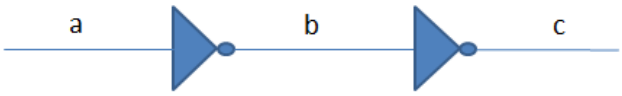


Figure 19: Two Inverter Circuit

Table 9 is constructed to more clearly depict the expanded state DTMC for the error probability of *c*. The table is identical to Table 2 for the DTMC, with the exception of the column for the states of *εc*.

Error-free b	εb	$c_{0\to1}/c_{1\to0}$	Error-free c	Error-incorporated c	εc	cef	pεc
0	0	-/0	1	1	0	1	0
0	0	-/1	1	0	1	1	0
0	1	0/-	1	0	1	1	1
0	1	1/-	1	1	0	1	1
1	0	0/-	0	0	0	0	0
1	0	1/-	0	1	1	0	0
1	1	-/0	0	1	1	0	1
1	1	-/1	0	0	0	0	1

Table 9: Resulting expanded state model for error probability in the Figure 19 circuit

From Table 9, it can be observed that

$$\varepsilon c = \varepsilon b \oplus c_{0\to1}/c_{1\to0}.$$

cef = c

pec = eb

The DTMC for error probably at gate *c* will now expand to eight states, since there are three Boolean variables, i.e. 2^3 states.

For the sole purposes of fitting more legibly into a transition probability matrix, the logical disjunctions of the expressions that result in unique combinations or *εc, cef,* and *pec* are abbreviated as "Trsn_to_$s_{\varepsilon c0}$", "Trsn_to_$s_{\varepsilon c1}$", … , and "Trsn_to_$s_{\varepsilon c7}$" .

$Trsn\_to\_s_{\varepsilon c0} = Pr(\neg c_{0\rightarrow 1}) | (b \wedge \neg \varepsilon b)$

$Trsn\_to\_s_{\varepsilon c1} = Pr(c_{1\rightarrow 0}) | (b \wedge \varepsilon b)$

$Trsn\_to\_s_{\varepsilon c2} = Pr(\neg c_{1\rightarrow 0}) | (\neg b \wedge \neg \varepsilon b)$

$Trsn\_to\_s_{\varepsilon c3} = Pr(\neg c_{0\rightarrow 1}) | (\neg b \wedge \varepsilon b)$

$Trsn\_to\_s_{\varepsilon c4} = Pr(c_{0\rightarrow 1}) | (b \wedge \neg \varepsilon b)$

$Trsn\_to\_s_{\varepsilon c5} = Pr(\neg c_{1\rightarrow 0}) | (b \wedge \varepsilon b)$

$Trsn\_to\_s_{\varepsilon c6} = Pr(c_{1\rightarrow 0}) | (\neg b \wedge \neg \varepsilon b)$

$Trsn\_to\_s_{\varepsilon c7} = Pr(\neg c_{0\rightarrow 1}) | (\neg b \wedge \varepsilon b)$

The DTMC for the error probability of c, *εc,* can then be expressed as a tuple, $D_{\varepsilon c} = (S_{\varepsilon c}, s_{\varepsilon c\_init}, \mathbf{P}_{\varepsilon c}, L)$, where:

- $S_{\varepsilon c} = \{s_{\varepsilon c0}, s_{\varepsilon c1, \dots}, s_{\varepsilon c7}\}$

- $s_{\varepsilon c\_init} = \varepsilon c_0$ (arbitrary)

- $\mathbf{P}_{\varepsilon c} = \begin{bmatrix} \text{Trans\_to\_s}_0 & \text{Trans\_to\_s}_1 & \dots & \text{Trans\_to\_s}_7 \\ \text{Trans\_to\_s}_0 & \text{Trans\_to\_s}_1 & \dots & \text{Trans\_to\_s}_7 \\ \dots & \dots & \dots & \dots \\ \text{Trans\_to\_s}_0 & \text{Trans\_to\_s}_1 & \dots & \text{Trans\_to\_s}_7 \end{bmatrix}$

- AP = {¬εc ∧ ¬cef ∧ ¬pec, εc ∧ cef ∧ ¬pec, ¬εc ∧ cef ∧ pec, εc ∧ cef ∧ pec, ¬εc ∧ ¬cef ∧ ¬pec, εc ∧ ¬cef ∧ ¬pec, ¬εc ∧ ¬cef ∧ pec, εc ∧ ¬cef ∧ pec}

    - $L(s_{\varepsilon c0})$ = { ¬εc ∧ ¬cef ∧ ¬pec}
    - $L(s_{\varepsilon c1})$ = {¬εc ∧ ¬cef ∧ pec}
    - $L(s_{\varepsilon c2})$ = {¬εc ∧ cef ∧ ¬pec}
    - $L(s_{\varepsilon c3})$ = {¬εc ∧ cef ∧ pec}
    - $L(s_{\varepsilon c4})$ = {εc ∧ ¬cef ∧ ¬pec}
    - $L(s_{\varepsilon c5})$ = {εc ∧ ¬cef ∧ pec}
    - $L(s_{\varepsilon c6})$ = {εc ∧ cef ∧ ¬pec}
    - $L(s_{\varepsilon c7})$ = {εc ∧ cef ∧ pec}

The total number of states for the model is equal to $2^{(\text{number of primary inputs + number of gates})}$. In the case of the two input NAND gate, $2^{(2+2)} = 16$ states.

3.5 Comparison of Presented Model vs Single DTMC for Error Probability Analysis

To verify the accuracy of the presented model for error probability analysis using DTMCs, a single DTMC is created for error probability analysis of gates in a combinational circuit, and the results are compared.

3.5.1 Single DMTC Construction for Error Probability

For a given combinational circuit, each state of a DTMC is a unique combination of error-free node logic levels and error indicating variables (e.g. εg for gate g) for each node in the circuit. From any state, transitions to each state are the logical conjunction of the probabilities of the primary inputs to the combinational circuit and the probabilities of the presence or absence of errors occurring at the output of each node (e.g. $\Pr(g_{0\to1})$ or $\Pr(g_{1\to0})$ for gate g that result in transitions to each unique state.

3.5.2 Single Inverter Example

Inverter b with primary input a is depicted in Figure 21. The error-free logic structure of the circuit is established prior to building any DTMC, therefore when a is a logic low, b is a logic high. Likewise, when a is a logic high, b is a logic low.

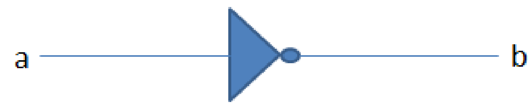


Figure 20: Inverter *b* with primary input *a*

The single DTMC for the node logic level and the error probability of *b* can be expressed as a tuple, $D_{circuit}$ = (S, s_{init}, $\mathbf{P}_{circuit}$, L), where:

- S = {s_0, s_1, s_2, s_3}
- $s_{init} = s_0$ (arbitrary)
- $\mathbf{P}_{circuit}$ =

$$\begin{bmatrix} \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \\ \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \\ \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \\ \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \end{bmatrix}$$

- AP = {a ∧ ¬εb, ¬ a ∧ ¬εb, a ∧ εb, ¬a ∧ εb}
  - L(s_0) = {a ∧ ¬εb}
  - L(s_1) = {¬a ∧ ¬εb}
  - L(s_2) = { a ∧ εb}
  - L(s_3) = {¬a ∧ εb}

3.5.3 Comparing $D_{circuit}$ vs $D_{\varepsilon b}$

When comparing the single DTMC (for node logic levels and error probability) for a circuit to that of DTMC for *εb* in section 3.4.1.1, it can be observed that the values of *εb* are the same for each of the corresponding states of the DTMCs. i.e. In both states $\underline{s_0}$ and $s_{\varepsilon b0}$, *εb*=0. In both states, s_1 and $s_{\varepsilon b1}$, *εb*=1, etc. In both DTMCs, since each state can be reached immediately reached from any state, if the transition probabilities are the same, the states of the two models will have equivalent steady-state probabilities for εb. $\mathbf{P}_{\mathbf{circuit}}$, the transition probability matrix for $D_{circuit}$, and $\mathbf{P}_{\varepsilon b}$, the transition probability matrix for *εb* are shown below.

- $\mathbf{P}_{\mathbf{circuit}}$ =

$$\begin{bmatrix} \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \\ \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \\ \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \\ \Pr(a) \wedge \Pr(\neg b_{0\to1}) & \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) & \Pr(a) \wedge \Pr(b_{0\to1}) & \Pr(\neg a) \wedge \Pr(b_{1\to0}) \end{bmatrix}$$

- $\mathbf{P}_{\varepsilon b}$ = $\begin{bmatrix} (\Pr(\neg b_{0\to1})|a) & (\Pr(\neg b_{1\to0})|\neg a) & (\Pr(b_{0\to1})|a) & (\Pr(b_{1\to0})|\neg a) \\ (\Pr(\neg b_{0\to1})|a) & (\Pr(\neg b_{1\to0})|\neg a) & (\Pr(b_{0\to1})|a) & (\Pr(b_{1\to0})|\neg a) \\ (\Pr(\neg b_{0\to1})|a) & (\Pr(\neg b_{1\to0})|\neg a) & (\Pr(b_{0\to1})|a) & (\Pr(b_{1\to0})|\neg a) \\ (\Pr(\neg b_{0\to1})|a) & (\Pr(\neg b_{1\to0})|\neg a) & (\Pr(b_{0\to1})|a) & (\Pr(b_{1\to0})|\neg a) \end{bmatrix}$

For $\mathbf{P}_{\varepsilon b}$, the probability of transitioning to *εb=0*, $Pr(\neg b_{0\to1})|\ a$, is equal to $Pr(\neg b_{0\to1}) \wedge P(a)$, since a logic high value for *a* is always and only the result of *Pr(a=1)* The steady-state probability of transitioning to *εb=0* is thus equal for DTMCs $D_{circuit}$ (single DTMC for entire circuit) and $D_{\varepsilon b}$. Likewise $Pr(\neg b_{1\to0})|\ \neg a$ can be expressed as $Pr(\neg b_{1\to0}) \wedge P(\neg a)$, etc. Thus

$D_{circuit}$, the single DTMC for the entire circuit (node logic levels and error probabilities) and $D_{\varepsilon b}$ (the DMTC for εb) yield equivalent steady-state values for states of εb.

3.5.4 Two Inverter Circuit Example

The two-inverter circuit with primary input a is depicted in Figure 21. The error-free logic structure of the circuit is established prior to building any DTMC, therefore when a=0, b=1, and likewise, when a=1, b=0.

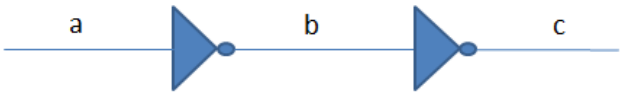


Figure 21: Two inverter circuit

Each unique combination of node logic levels and error probabilities at the output of each gate comprises a state. For the two-inverter circuit, the logic level and error probabilities comprising the states are variables are a, b, c, εb, and εc. Since the error-free logic levels of b and c change deterministically with the value of a, only a is mentioned. As shown in Table 10, the probabilities Pr(a), Pr($b_{0\to1}$)/Pr($b_{1\to0}$), and Pr($c_{0\to1}$)/Pr($c_{1\to0}$), determine the state the DTMC will transition to (i.e. the values of a, b, c, εb, and εc). In

this way, the logical conjunction of distinct combinations for *Pr(a), Pr($b_{0\to1}$)/Pr($b_{1\to0}$),* and *Pr($c_{0\to1}$)/Pr($c_{1\to0}$)* comprise the transitions. As the logic levels of *b* and *c* change deterministically with *a*, the values of *b* and *c* are omitted from Table 9.

Transition			**Resulting State**			
Pr(a)	**Pr($b_{0\to1}$)/Pr($b_{1\to0}$)**	**Pr($c_{0\to1}$)/Pr($c_{1\to0}$)**	**a**	**εb**	**εc**	**state #**
0	-/0	0/-	0	0	0	0
0	-/0	1/-	0	0	1	1
0	-/1	-/0	0	1	1	3
0	-/1	-/1	0	1	0	2
1	0/-	-/0	1	0	0	4
1	0/-	-/1	1	0	1	5
1	1/-	0/-	1	1	1	7
1	1/-	1/-	1	1	0	6

Table 10: Logic levels and error probabilities for two inverter circuit in Figure 21

For the sole purposes of fitting more legibly fit into a transition probability matrix, the logical disjunctions that comprise the transitions of each state in the DTMC are shown below:

$$\text{Trsn\_to\_}s_0 = \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) \wedge \Pr(\neg c_{0\to1})$$

$$\text{Trsn\_to\_}s_1 = \Pr(\neg a) \wedge \Pr(\neg b_{1\to0}) \wedge \Pr(c_{0\to1})$$

$$\text{Trsn\_to\_}s_2 = \Pr(\neg a) \wedge \Pr(b_{1\to0}) \wedge \Pr(c_{1\to0})$$

$$Trsn\_to\_s_3 = Pr(\neg a) \wedge Pr(b_{1\rightarrow 0}) \wedge Pr(\neg c_{1\rightarrow 0})$$

$$Trsn\_to\_s_4 = Pr(a) \wedge Pr(\neg b_{0\rightarrow 1}) \wedge Pr(\neg c_{1\rightarrow 0})$$

$$Trsn\_to\_s_5 = Pr(a) \wedge Pr(\neg b_{0\rightarrow 1}) \wedge Pr(c_{1\rightarrow 0})$$

$$Trsn\_to\_s_7 = Pr(a) \wedge Pr(b_{0\rightarrow 1}) \wedge Pr(c_{0\rightarrow 1})$$

$$Trsn\_to\_s_6 = Pr(a) \wedge Pr(b_{0\rightarrow 1}) \wedge Pr(\neg c_{0\rightarrow 1})$$

The DTMC for the error probabilities of the two inverter circuit in Figure 22, can be expressed as a tuple, $D_{circuit} = (S, s_{init}, \mathbf{P_{circuit}}, L)$, where:

- $S = \{s_0, s_{1, \ldots,} s_7\}$
- $s_{init} = s_0$ (arbitrary)
- $\mathbf{P_{circuit}} = \begin{bmatrix} Trans\_to\_s_0 & Trans\_to\_s_1 & \ldots & Trans\_to\_s_7 \\ Trans\_to\_s_0 & Trans\_to\_s_1 & \ldots & Trans\_to\_s_7 \\ \ldots & \ldots & \ldots & \ldots \\ Trans\_to\_s_0 & Trans\_to\_s_1 & \ldots & Trans\_to\_s_7 \end{bmatrix}$
- AP = {¬a ∧ ¬εb ∧ ¬εc, ¬a ∧ ¬εb∧ εc, ¬a ∧ εb ∧ εc, ¬a ∧ εb ∧ ¬εc,

  a ∧ ¬εb ∧ ¬εc, a ∧ ¬εb ∧ εc, a ∧ εb ∧ εc, a ∧ εb ∧ ¬εc}

  - $L(s_0) = \{\neg a \wedge \neg \varepsilon b \wedge \neg \varepsilon c\}$

- $L(s_1) = \{\neg a \wedge \neg \varepsilon b \wedge \varepsilon c\}$
- $L(s_2) = \{\neg a \wedge \varepsilon b \wedge \neg \varepsilon c\}$
- $L(s_3) = \{\neg a \wedge \varepsilon b \wedge \varepsilon c\}$
- $L(s_4) = \{a \wedge \neg \varepsilon b \wedge \neg \varepsilon c\}$
- $L(s_5) = \{a \wedge \neg \varepsilon b \wedge \varepsilon c\}$
- $L(s_6) = \{a \wedge \varepsilon b \wedge \neg \varepsilon c\}$
- $L(s_7) = \{a \wedge \varepsilon b \wedge \varepsilon c\}$

3.5.5 Comparing $D_{circuit}$ vs $D_{\varepsilon c}$

The logic level of *a* from $D_{circuit}$ is equivalent to the logic level of *cef* from $D_{\varepsilon c}$, when *a*=0, *b*=0, and *c*=0 (i.e. *cef*, the error-free logic level of *c*). Likewise, the logic level of *εb* from $D_{circuit}$ is equivalent to the logic level of *pec* from $D_{\varepsilon c}$, as *εb* is the propagated error entering into *c* (i.e. *pec*). These variable equivalents for $D_{\varepsilon c}$ and $D_{circuit}$ are shown in Table 10.

DTMC	$D_{\varepsilon c}$	$D_{circuit}$
Error-free node *c* equivalent	cef	a
Error at node *b* equivalent	pec	εb

Table 11: Variable equivalents for $D_{\varepsilon c}$ and $D_{circuit}$.

From $D_{\varepsilon c}$, the DTMC for εc, $s_{\varepsilon c0} = \neg\varepsilon c \wedge \neg cef \wedge \neg pec$, which is equivalent to $\neg\varepsilon c \wedge \neg a \wedge \neg\varepsilon b$, according to Table 11, and consequently equivalent to s_0.

Therefore,

$$s_{\varepsilon c0} = s_0$$

$$\neg\varepsilon c \wedge \neg cef \wedge \neg pec = \neg a \wedge \neg\varepsilon b \wedge \neg\varepsilon c$$

From $D_{\varepsilon c}$, the DTMC for εc, the probability of transitioning to εc=0 is:

$$Trsn\_to\_s_{\varepsilon c0} = Pr(\neg c_{0\to1}) | (b \wedge \neg\varepsilon b)$$

If *b* is a logic high and there is no resulting error at the output of b, i.e. εb=0, then there is no high to low error originating at the output of *b*, (i.e. $\neg b_{1\to0}$). Therefore, the case $b \wedge \neg\varepsilon b$, implies $\neg a \wedge \neg b_{1\to0}$.

$$Trsn\_to\_s_{\varepsilon c0} = Pr(\neg c_{0\to1}) | (b \wedge \neg\varepsilon b)$$

By substitution,

$$Trsn\_to\_s_{\varepsilon c0} = Pr(\neg c_{0\to1}) | (\neg a \wedge \neg b_{1\to0})$$

When considering steady-state probability, $Pr(\neg c_{0\to1}) | (\neg a \wedge\neg\ b_{1\to0})$ can be expressed as:

$$Pr(\neg c_{0\to1}) \wedge Pr(\neg a) \wedge Pr(\neg b_{1\to0}).$$

This is equivalent to $Trsn\_to\_s_0 = Pr(\neg a) \wedge Pr(\neg b_{1\rightarrow 0}) \wedge Pr(\neg c_{0\rightarrow 1})$. Therefore it is proven that $s_{\varepsilon c0}$ and s_0 are equivalent, and the probabilities of transitioning to each of those respective states, $Trsn\_to\_s_{\varepsilon c0}$ and $Trsn\_to\_s_0$ are equivalent as well.

Chapter 4: Probabilistic Model Checking

The established stochastic model presents a framework for properties to be checked against the model. Properties provide analysis of the stochastic model by verifying specified conditions or criteria. As the transitions of the model are probabilistic, the properties used to test it will reflect the inherent probabilistic nature of the system, in this way the utilization of the properties can be referred to as probabilistic model checking. The properties are based upon a probabilistic temporal logic, which examines the behavior of the system with respect to time.

A simple view of the probabilistic model checking process, depicted in Figure 22, consists of a probabilistic model checker, where the inputs are a probabilistic model, and probabilistic temporal logic specifications written for the model. The output of the model checker is the result from the application of the logic specifications to the model, verifying or disproving the specifications, or giving quantifiable results.

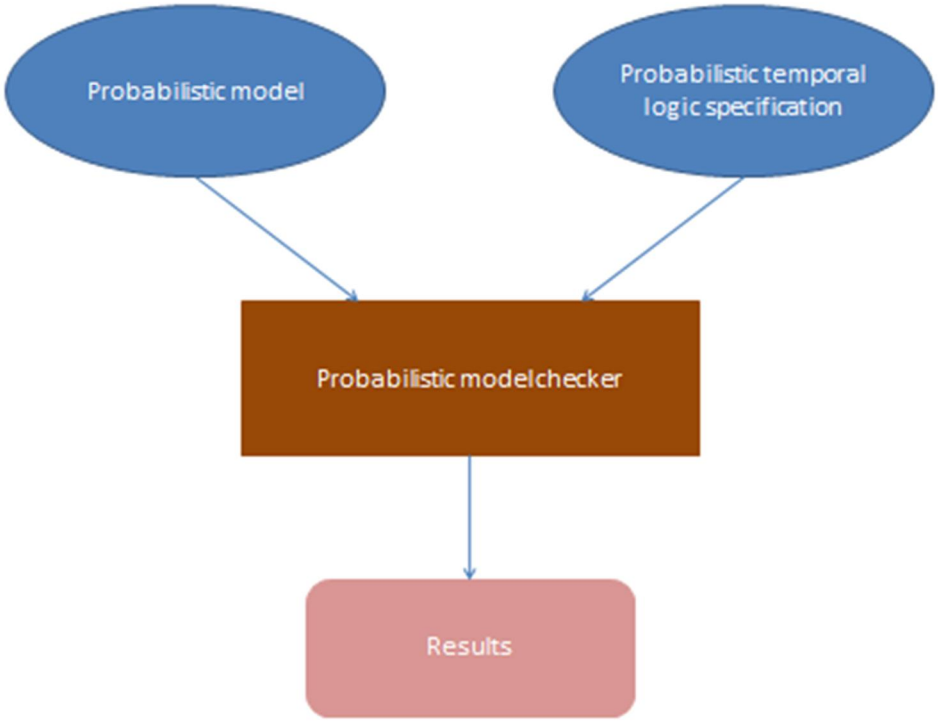


Figure 22: Probabilistic model checking flow

Once the description of the system has been modeled as a stochastic system, e.g. DTMCs, a formal specification of quantitative system properties, is used to analyze the circuit expressed using temporal logic. As Dutch computer scientist Edsger Dijkstra said "Program testing can be used to show the presence of bugs, but never to show their absence!"; property specifications must be formed with much thought to extract the desired information regarding reliability and performance properties. In order to analyze a DMTC, one or more temporal logic properties are required. The beauty of model checking is the depth and breadth of information extracted from the model is largely at the discrepancy of the composed properties.

All properties are expressed temporally to convey either transient or steady-state/long-run behavior.

Certain properties are proposed that extract potentially desirable information. Implementation of the properties using a probabilistic temporal language with model checker specific syntax is detailed later in Chapter 5. These properties are by no means exhaustive, but represent only a small sample set of various attributes of the system that can be tested.

For this project, the primary objective of the model checking and the composed properties is to estimate the error probability at specified nodes in the circuit. Properties are thus written to analyze the error probabilities with respect to the previously mentioned temporal measures.

Various nodes in the circuit with high probability of failure can be of interest. Or, the dependency of the output of the circuit on specific nodes can be relevant with regard to the placement of fault tolerant circuitry. An error at one node might have a greater effect on the output of the circuit than an error at another node. Nodes with higher probability of failure and/or nodes that have a higher effect on the output of the circuit could potentially be desirable places to implement redundant circuitry to circumvent potential failure

4.1 Steady-State Probabilities

Properties can be used to estimate any number of long-run behaviors probabilities of the model. As one of the more important attributes of the model to analyze can be the steady-state behavior of various nodes, steady-state estimations in DTMCs are briefly detailed.

4.1.1 Long-Run and Steady-State Probabilities

DTMCs can either be constructed to depict discrete time-steps to accurately model time or as time-abstract, where no information is assumed from transitions of the model. The presented combinational logic model is a time-abstract model, as propagation delay is not taken into account.

Transient probabilities are essentially only useful in discrete time-step models, however, long-run behavior can yield useful information in any DTMC.

The percentage of time, in the long run, that is spent in a specific state can be calculated by taking the limit:

$$\underline{\pi}_s = \lim_{k\to\infty} \underline{\pi}_{s,k}$$

where $\underline{\pi}_{s,k}$ is the transient state distribution at a given time k (or after k transitions), having started in state, s (i.e. $\pi_{s,k}(s')$ for all states s'). This limit, if it exists, is referred to as the limiting distribution.

To find out if the limiting distribution exists, firstly some terminology from [4] is introduced.

- Reachable: A state, *s,'* is reachable if, from *s,* there exists a finite path that ends in *s'*.

- Strongly Connected: $T \subseteq S$, is strongly connected if there exists a path from each state in *T* to every other state in T.

- Strongly Connected Component (SCC): An SCC is a maximally strongly connected set of states, where no superset of the set of states is strongly connected.

- Bottom Strongly Connected Component (BSCC): A set of states is a BSCC if it is (1) a SCC, and (2) there are no state outside the SCC is reachable from T.

- Irreducible: If all of a Markov chain's states belong to one BSCC, the Markov chain is considered irreducible.

- Periodic: A state *s*, with period *d*, is periodic if the greatest factor of the set $\{n \mid f_s^{(n)} > 0\} = d$, where $f_s^{(n)}$ is the probability of starting in *s* and returning to *s* in *n* steps. A Markov Chain with a period of one is aperiodic.

If a DTMC is finite, irreducible, and aperiodic, a limiting distribution will exist, and the limiting distribution is independent of initial state and distribution. Probabilities independent of initial distribution are referred to as steady-state probabilities, denoted simply as $\underline{\pi}$. Steady-state probabilities can be computed by using the linear equation system:

$$\underline{\pi} \cdot \mathbf{P} = \underline{\pi} \text{ and } \sum_{s \in S} \underline{\pi}(s) = 1 \quad [4]$$

The unique solution to the system will be the steady-state probability.

4.2 Probabilistic Computational Tree Logic

Verification of the system is performed formally by means of properties specifications that reveal, prove, or disprove specified functionality of a system. The properties take the form of probabilistic temporal logics, which allows for analysis of the behavior of how a system changes temporally.

Probabilistic Computational Tree Logic (PCTL), a branching-temporal logic, serves as the foundation for properties used in this probabilistic model checking. The probabilistic component of PCTL, the *P* operator, allows for quantitative extension on standard CTL quantifiers over paths. PCTL syntax is comprised of state and path formulas. State formulae are generated by context-free grammar, where for a state *s* of a DTMC (S, s_{init}, **P**, L), the grammar for PCTL state formulae is

$$\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid P_{\sim p}[\psi]$$

where *a* is an atomic proposition, ψ is a path formula, and $p \in [0,1]$ is a probability bound ~∈ [<, >, ≤, ≥], p ∈ [0,1]. [4]

The semantics for the DTMC state formulae are defined as

$s \models \text{true}$		always
$s \models a$	$\Leftrightarrow$	$a \in L(s)$
$s \models \phi_A \wedge \phi_B$	$\Leftrightarrow$	$s \models \phi_A$ and $s \models \phi_B$
$s \models \neg\phi$	$\Leftrightarrow$	$s \not\models \phi$

where ⊨ implies satisfaction.

For a path *ω* of an DTMC (S, s_{init}, **P**, L), the grammar for PCTL path formulae is

$$\Psi ::= X\phi \mid \phi \cup^{\leq k} \phi \mid \phi \cup \phi \mid$$

where k ∈ $\mathbb{N}$. [4]

In PCTL, path formulae only occur inside the *P* operator.

The semantics for DTMC path formulae are formally defined as

$\omega \models X\phi$	$\Leftrightarrow$	$\omega(1) \models \phi$
$\omega \models \phi_A \cup^{\leq k} \phi_B$	$\Leftrightarrow$	∃ i≤ k such that $\omega(i) \models \phi_B$

and $\forall j < l, \omega(j) \vDash \phi_A$

$\omega \vDash \phi_A \cup \phi_B \quad \Leftrightarrow \quad \exists k \geq 0, \omega(k) \vDash \phi_B$ and $\forall i < k\ \omega(i) \vDash \phi_A$

$\omega \vDash F\ \Phi \quad \Leftrightarrow \quad \exists k \geq 0, \omega(k) \vDash \Phi$

$\omega \vDash G\ \Phi \quad \Leftrightarrow \quad \forall i \geq 0\ \omega(i) \vDash \Phi$ [4]

4.3 Satisfiability

With the probabilistic model and the temporal language necessary to build the property specifications, determining how a property specification is satisfied or not satisfied has to be addressed. The inputs to the system, the DTMCs and the PCTL formula ϕ, and the output that is the set of states that satisfies ϕ, the state formula, are formally defined as

$$Sat(\phi) = \{s \in S \mid s \vDash \phi\}$$

Satisfaction can be an evaluation for a single state, $s \vDash \phi$, i.e. $s \in Sat(\phi)$, or of the entire system, $s \vDash \phi\ \forall s \in S$, i.e. $Sat(\phi) = S$.

For the non-probabilistic operators, satisfaction can be defined as

$Sat(true) = S$

$Sat(a) = \{ s \in S \mid a \in L(s) \}$

$Sat(\neg\phi) = S \backslash Sat(\phi)$

$Sat(\phi_A \wedge \phi_B) = Sat(\phi_A) \cap (\phi_B)$

$Sat(P_{\sim p}[\psi]) = \{ s \in S \mid Prob(s, \psi) \sim p \}$ [4]

Chapter 5: Case Studies

5.1 PRISM

PRISM is an open-source probabilistic model checker that can be used to analyze and model probabilistic behavior. Based at Oxford University Computing Laboratory, PRISM was initially developed at the University of Birmingham. It supports construction and analysis of discrete-time and continuous-time Markov chains, and Markov decision processes. PRISM's high-level Property Specification Language supports a number of temporal logics, PCTL, CSL, and LTL (for the model described, only PCTL is covered). In cases of state explosion, PRISM supports up to a maximum model size of 170 million states. [4]

Although briefly described earlier, an overview of the probabilistic model checking process, further depicted in Figure 23, is comprised of two phases:

- model construction, a description of the system
- model checking, a formal specification of the system's quantitative properties.

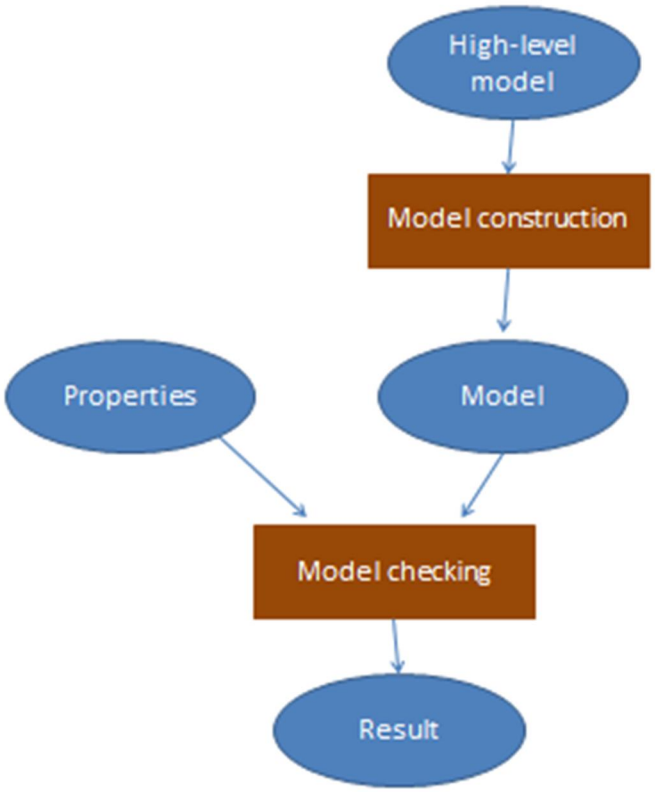


Figure 23: Probabilistic model checking process

The high-level model is the PRISM language description. Model construction takes PRISM language description and through matrix manipulation and a graph-based algorithm, constructs a set of reachable states. A model is then implemented as Markov models, in this case, DTMCs. Model checking is performed by applying properties formed as temporal logics to the model. The outcome of the properties expressed in formulas yield quantifiable results.

Probabilistic model checking, PRISM in particular, applies numerical computation to produce exact results. This stands in contrast to discrete-event simulation methods where results are approximated by taking the average of results from a large number of random samples. These utilized methods are comprised of graph theoretical algorithms and numerical computation. Graph theoretical algorithms are used to determine the set of reachable states in a model, or estimate qualitative properties. Numerical computation is used to calculate probabilities and reward values. The graph theoretical algorithms utilize graph structures of a Markov chain to determine qualitative properties and the reachability of states. [4]

Solutions to linear equations of systems, or calculation of transient probabilities of a Markov chain generally form the basis for numerical computation. Jacobi, Gauss-Seidel and SOR techniques are used for finding solutions for linear equation systems, while uniformisation is utilized for transient probabilities [4].

5.1.2 Model Specification

PRISM models are composed of modules and variables. A system is comprised of the parallel composition of interacting modules, with each module representing different components of the modeled system. A set of variables represents the state of each module.

The change of the state of modules is expressed as a set of guarded commands of the form:

[action] guard → probability: update

where:

- *action* is an optional label for reference
- *guard* is a predicate over the model variables
- *probability* is a real-valued expression
- *update* is of the form $(x'_1=u_1)$ & $(x'_2=u_2)$ & … & $(x'_k=u_k)$,

  where $x_1, x_2, \ldots, x_k$ are local variable belonging to a module and $u_1, u_2, \ldots u_k$ are expressions over all model variables [4].

More specifically, a full command in PRISM looks like the below example:

$$\underbrace{[update]}_{action}\ \underbrace{(b = true \wedge \varepsilon b = false)}_{guard} \rightarrow \underbrace{c10}_{probability} : \underbrace{(\varepsilon c' = true)}_{update} + \underbrace{1 - c10}_{probability} : \underbrace{(\varepsilon c' = false)}_{update};$$

A command is enabled if the state satisfies the predicate guard. A transition then updates the module's variables by taking into account its specified probability of occurrence.

5.1.3 Example Error Estimation Model in PRISM

Figure 24 depicts the single inverter *b,* used in Chapter 3, with primary input *a*. As *a* is a primary input, it is assumed that there is no probability of error at *a*. Figure 25

depicts an implementation of the single inverter circuit for error probability in compliable PRISM modeling language.

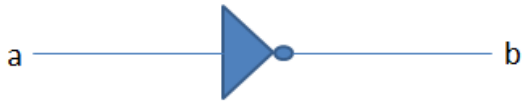


Figure 24: Single inverter *b* with primary input *a*

```
dtmc

const double pa = 0.5;    // probability of primary input 'a' being true (i.e. logic 1)
const double b01 = 0.1;   // probability of 0 -> 1 error originating at node 'c'
const double b10 = 0.1;    // probability of 1 -> 0 error originating at node 'c'

//------------------------- error-free circuit structure ----------------------------------------
formula b=true = a=false;   // b = INV(a)

//------------------------- input transitions ----------------------------------------------------
module primary_input
   a:bool;   // logic level of input node 'a'

   [update]a=false | a=true -> 1-pa:(a'=false) + pa:(a'=true);
endmodule


//------------------------- DTMCS for error estimation at output of inverter ‘b’ ---------
module inv_b
   eb :bool;  // error at 'b', where 'true' is an error, and 'false' is error-free

   // from any state, ‘eb’ transitions to ’true’ with probability 'b01' or ‘b10’
   [update]a=false -> b10:(eb'=true) + 1-b10:(eb'=false) ;
   [update]a=true -> b01:(eb'=true) + 1-b01:(eb'=false);
endmodule
```

Figure 25: Prism implementation for error estimation of single inverter *b*

In module *primary_input,* the logic level of primary input *a* toggles between *true* and *false* with a fifty percent probability of being in each state, where *true* and *false*

represent logic low and logic high respectively. The error-free circuit structure is established using formulas. Formula *b* represents the error-free logical functionality of inverter *b*. In PRISM, formulas are utilized to avoid duplication of code. The name of the formula can then be referenced as shorthand for the expression in any place an expression would be used. In the example, the formulas essentially creates a module or DTMC for *b*, where the value changes in accordance with the Boolean formula and the value of *a*. In this way, module *primary_input* in addition with formula *b* form the previously mentioned DTMC for error-free node logic levels of the circuit. Representation of the circuit structure through formulas aids in the scalability of the model.

Module *inv_b* is used to calculate the probability of error at the output of the inverter. Module *inv_b* comprises two commands for error estimation at the output of inverter *b,* as there is a command for each logic level of *a,* the input to the inverter.

The PRISM code in Figure 25 is compilable, and can be run by cutting and pasting the code into the 'model' window of the PRISM Model Checker.

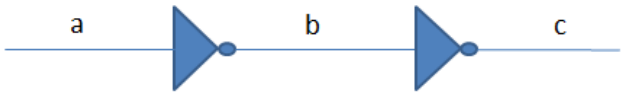


Figure 26: Two Inverter circuit

Figure 26 depicts the simple two-inverter circuit used in Chapter 3, with primary input *a* and output *c*. As this circuit is simply the single inverter with an additional inverter on the output, the PRISM model code is identical to the PRISM model for the single inverter circuit with the addition of formula *c*, the error-free logical representation of inverter *c*, and module *inv_c,* used for error estimation of *c.* In the first commands for *inv_c* the guards are conditions where an error is not propagated from inverter *b.* In the last two commands of module *inv_c,* the guards are conditions where an error is propagated from *b.*

```
dtmc

const double pa = 0.5;       // probability of primary input 'a' being true (i.e. logic 1)
const double b01 = 0.1;    // probability of 0 -> 1 error originating at node 'b'
const double b10 = 0.1;    // probability of 1 -> 0 error originating at node 'b'
const double c01 = 0.1;    // probability of 0 -> 1 error originating at node ‘c’
const double c10 = 0.1;    // probability of 1 -> 0 error originating at node 'c'

//------------------------- error-free circuit structure ----------------------------------------------
formula b=true = a=false;   // b = INV(a)
formula c=true = b=false;   // c = INV(b)


//------------------------- input transitions ----------------------------------------------------------
module primary_input
   a:bool;  // logic level of input node 'a'

   [update]a=false | a=true -> 1-pa:(a'=false) + pa:(a'=true);
endmodule

//------------------------- DTMCS for error estimation at output of inverter ‘b’ --------------
module inv_b
   eb :bool;  // error at 'b', where 'true' is an error, and 'false' is error-free

// from any state, ‘eb’ transitions to ’true’ with probability 'b01' or ‘b10’
   [update]a=false -> b10:(eb'=true) + 1-b10:(eb'=false);
   [update]a=true -> b01:(eb'=true) + 1-b01:(eb'=false);
endmodule


//------------------------- DTMCS for error estimation at output of inverter ‘c’ --------------
module inv_c
   ec:bool;  // error at 'c', where 'true' is an error, and 'false' is error-free
   cef:bool;  // error-free ‘c’, where 'true' is an error, and 'false' is error-free
   pec:bool;  //  'true’ is a propagated error to 'd', 'false' is no propagated error to 'd'.
// cases where no error propagated from inverter 'b'
   [update]b=true & eb=false -> c01:(ec'=true)&(cef'=false)&( pec'= false)+1-c01:(ec'=false)&(cef'= false)&
(pec' =false);
   [update]b=false & eb=false -> c10:(ec'=true)&(cef'=true)&(pec'=false)+ 1-c10:(ec'=false)
&(cef'=true)&(pec'=false);

// cases where error propagated from inverter 'b'
   [update]b=true  & eb=true -> c10:(ec'=false)&(cef'=false)&(pec'=true) + 1-c10:(ec'=true) &(cef'=false)&
(pec'=true);
   [update]b=false & eb=true -> c01:(ec'=false) &(cef'=true)&(pec'=true) + 1-
c01:(ec'=true)&(cef'=true)&(pec'=true);
endmodule
```

Figure 27: PRISM implementation for error estimation of two inverter circuit

5.2 Property Specifications

The general scope of how properties can be used for reliability analysis of the circuit was previously presented, but the implementation of them is realized through PRISM's property specifications. Operators from PCTL and some of its extensions form the foundation for PRISM's property specification language.

The inherent nature of probabilistic model checking lends itself to analysis of quantitative properties. It allows for testing the probability of a given event or occurrence. More complex properties can be expressed by combining the arithmetic equations with numeric values.

Operators *P*, *R*, and *S* form the primary constructs in the PRISM specification language which are used to extract temporal characteristics of the system. The *P* operator is the probability of an event occurrence. The *S* operator is the steady-state probability of an event. The *R* operator is the expected value of a random variable that is associated with a specified reward. As the presented model is time-abstract, only steady-state and rewards properties involving steady-state behavior yield viable results for large scale circuits.

As stated previously, the primary inquiries of interest involve the error probability of nodes as well as the dependency of the nodes on other nodes in the circuit (as discussed previously, the dependency of the error probability of a node on the error probability of nodes other than that of the nodes in its fan-in cone cannot be accurately

analyzed without additional variables). To that end, 15 property specifications in PRISM's property specification language are composed and shown in Table 12. The properties only represent a small subset of qualities of the circuit that can be analyzed. They largely serve as examples to represent some of the depth and breadth that PRISM's properties can analyze.

No.	Property
1	$P_{=?}[F^{I}\ \varepsilon c=true]$
2	$S_{=?}[\varepsilon=true]$
3	$S_{=?}[\varepsilon c=true\ \&\ pec=true]$
4	$S_{=?}[\varepsilon c=true\ \&\ cef=false]$
5	$S_{=?}[\varepsilon c=true\ \&\ pec=true\ \&\ cef=false]$
6	$P_{=?}\ [F\ \varepsilon c=true]$
7	$P_{=?}\ [G^{[\leq t]}\ \varepsilon c=false]$
8	$P_{=?}\ [\varepsilon c=false\ U^{I}\ \varepsilon c=true]$
9	$P_{=?}\ [\varepsilon c=false\ U\ \varepsilon c=true]$
10	$P_{=?}\ [\varepsilon c=false\ W\ \varepsilon c=true]$
11	$P_{=?}\ [F\ \varepsilon c=true]\ /\ P_{=?}\ [F\ \varepsilon c=true]$
12	$R_{\{\text{“}\ \varepsilon c\_total\text{”}\}=?}\ [C^{I}]$,
13	$R_{\{\text{“step”}\}=?}\ [F\ \varepsilon c=true]$
14	$R_{\{\text{“error”}\}=?}\ [S]$
15	$R_{\{\text{“error”}\}=?}\ [I=t]$

Table 12: Proposed properties for combinational circuits

5.2.1 Transient and Steady-State Probabilities

The transient behavior of the output of the circuit can be examined using the *P* operator and the 'eventually' operator F, expressed as F^{I} where *I* is the interval $[0 \leq t] \in \mathbb{R}$.

Property 1: $P_{=?}[F^{I}\, \varepsilon c=true]$

Property 1 asks what the probability is of an error at node *c*, occurring at some step during internal *I*.

The steady-state operator, *S*, is used in Property 2, where the question posed is the steady-state probability of an error at node *c*.

Property 2: $S_{=?}[\varepsilon c = true]$,

This can also be expanded to analyze multiple nodes using logic disjunction or conjunction. Property 3 demonstrates the expansion by asking the steady-state probability of an error at node *c* as a result of an error propagated from node *b*.

Property 3: $S_{=?}[\varepsilon c=true \ \& \ pec=true]$

Property 4 demonstrates another variation of the steady state example is to look at the steady-state probability of error at the output of the circuit when the expected value (error-free value) at *c* is a logic low.

Property 4: $S_{=?}[\varepsilon c=true \ \& \ cef=false]$

Property 5 is another steady-state property revealing the steady-state probability of an error at *c,* when an error-free logic low is expected, and the error is propagated from *b.*

Property 5: $S_{=?}$[εc=true & cef=false & pec=true]

5.2.2 Timing and Event Ordering

For transient analysis, the *F* operator is time-bounded by the interval *I*. However, there also exists a time-unbounded variant that is simply expressed as *F* without *I*, where F then becomes equivalent to $F^{[0,\infty)}$, this allows analysis over the lifetime of the system. The time-unbounded operator *F* can be used to estimate if an event will ever occur.

Property 6 poses the question if there exists the possibility that there will ever be an error at node *c*. In our model, this is a trivial question, but this could be relevant in a model where there exists no probability for an error to originate at the gate of interest, but possibly be propagated from other gates.

Property 6: $P_{=?}$ [F εc=true]

The 'always ' or 'global' operator, *G*, expresses a condition that remains true, as opposed to expressing a condition becoming true, as operator *F* does. An example of a possible relevant property with regard to the circuit could be Property 7, which asks what the probability is that there is no error at node *d* during interval *I.*

Property 7: $P_{=?}$ [G^{I} εc=false]

The 'until' operator, *U*, states that for two given conditions, the first condition is to remain true until the second condition becomes true. Property 8 poses the question of asking what the probability is that an error at node *c* will not occur until after interval *I*.

Property 8: $P_{=?}$ [εc=false U^{I} εc=true],

Property 9 is a slight modification of Property 8, asking what the probability is that a given node will fail before another node, specifically, the probability that there is an error at node *c* prior to an error at node *b*. Although indirect, this can demonstrate comparative susceptibility to error, as well as potential error dependency.

Property 9: $P_{=?}$ [εc=false U εb=true]

Similar to the 'until' operator, Property 10 demonstrates the 'weak until' operator, *W*, which can be used to ask what the probability is that an error at node *b*, if it occurs, occurs prior to an error at node *c*. The primary difference between the 'weak until' and the 'until' operator is that the 'weak until' operator also encompasses the scenario where no error at either node occurs.

Property 10: $P_{=?}$ [εc=false W εb=true]

5.2.3 Conditional Probability

Conditional probability can be used to express the dependence of one event upon another. With respect to the circuit, this can be particularly relevant where it is desired to find the probability of an error at one node in the circuit given an error at another node. Conditional probability can be examined by using the conjunction operator. For example, Property11 asks the question that at a given time interval, what is the probability of an error at node *c* given an error at node *b.*

Property 11: $P_{=?}\,[F^{I}\,\varepsilon c=\text{true}]\ \&\ P_{=?}\,[F^{I}\,\varepsilon b=\text{true}]$

Using this technique could show the level of dependence one node's error probability has on another node's error probability. This could assist in fault tolerance design, in that the nodes where being error free is most critical to the output of the circuit can be targets for including redundant circuitry to better assure the output would be unaffected by errors on nodes that have an especially strong influence on the accuracy of the output.

5.2.4 Reward-Based Properties

Rewards are used to detail quantitative properties [4]. In a Markov chain, rewards are the labeling of state and transitions with real values. In PRISM, rewards can be associated either with specified states or transitions.

The *R* operator represents the expected variable of a random variable defined using PRISM's rewards. There are four different variants of reward properties in PRISM

- **C**<=t: Cumulative
- **F**: Reachability
- **I**=t: Instantaneous
- **S**: Steady-state

where *t* is the step.

As there can be multiple reward structures, each type of reward can have an associated name or title, denoted here as *rew*, and used with the *R* operator as

$$R_{\{\text{“rew”}\}=?}\ [],$$

to denote the expected value of reward structure *rew*. For a simple example, a reward structure *ec_total* assigns a state reward of one to states where node *c* has an error.

```
rewards "ec_total"
        ec=true: 1;
endrewards
```

The accumulated amount of the reward over a period of time can be used as in Property 12 to ask the question of which is the expected cumulative number of steps node *c* has an erroneous output over time interval [0, t].

Property 12: $R_{\{\text{“ec\_total”}\}=?}[C^{\leq t}]$,

Property 12 can be expanded to help analyze the source of the error at node *c*. In addition to the reward structure *ec_total*, reward structures can be created to assign a state reward of one to states for each of the following unique cases:

- There is an error at *c* and *b* at the same step

```
rewards "ec_eb"
        (ec=true & eb=true): 1;
endrewards
```

- There is an error at *c* when a logic low was expected

```
rewards "ec_def_0"
        (ec=true & cef=false): 1;
endrewards
```

- There is an error at *c* when a logic high was expected

```
rewards "ec_def_1"
        (ec=true & cef=true): 1;
endrewards
```

Rewards can also be accumulated until a specified time and utilized to reveal such information as mean-time to failure. For example, a reward structure *steps* is given a value of one at each state.

```
rewards "steps"
        a=false:1;
        a=true:1;
endrewards
```

Property 13 asks the expected number of step prior to the first failure at node *c.*

Property 13: $R_{\{\text{“steps”}\}=?}$ [F εc=true]

The *S* operator can be utilized with rewards to give the steady-state of reward accumulation. If a reward structure error assigns a transition reward of one to every transition in the circuit where an error occurs, Property 14 asks the expected stead-state rate of error probability for all the nodes in the two inverter circuit.

```
rewards "total_errors"
        eb=true:1;
        ec=true:1;
endrewards
```

Property 14: $R_{\{\text{“total\_errors”}\}=?}$ [S]

Using reward structures to analyze steady-state properties can be seen as similar or redundant when compared to using the *S* operator to analyze properties as shown in Property 2, but it can be useful to double check or verify the accuracy of analysis.

Finally, the *R* operation can be used to show instantaneous values of rewards. Property 15 asks the total expected number of errors in the circuit at a given time step/instance. Where *t* is the step/instance at which the total number of expected errors is being estimated.

Property 15: $R_{\{\text{“total\_error”}\}=?}$ [I=t],

5.3 Presented DTMC Model vs Single DTMC Model

To verify accuracy, the modular DTMC model is compared to the singular DMTC model. Figure 28 depicts a simple circuit of an inverter feeding into one of the inputs of a two-input NAND-gate.

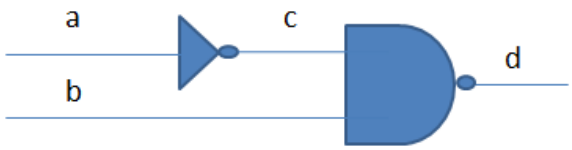


Figure 28: Inverter and two-input NAND circuit

Pr(a) and Pr(b) are each 0.5. $c_{0\to1}$, $c_{1\to0}$, $d_{0\to1}$ and $d_{1\to0}$ are all 0.1. Variants of the previously proposed properties are run against the model, with the results shown in Table 12.

No.	Property	Presented DTMC Model	Single DTMC Model
1	$P_{=?}[F^{\leq 20}$ εd=true]	0.9487	0.9510
2	$S_{=?}$[εd=true]	0.1400	0.1400
3	$S_{=?}$[εd=true & ped=true]	0.0500	0.0500
4	$S_{=?}$[εd=true & def=false]	0.0450	0.0450
5	$S_{=?}$[εd=true & ped=true & def=false]	0.0225	0.0225
6	$P_{=?}$ [F εd=true]	1.0000	1.0000
7	$P_{=?}$ [$G^{\leq 20}$ εd=false]	0.0513	0.0489
8	$P_{=?}$ [εd=false $U^{\leq 20}$ εc=true]	0.9487	0.9510
9	$P_{=?}$ [εc=false U εc=true]	0.7632	0.7368
10	$P_{=?}$ [εc=false W εc=true]	0.7632	0.7368
11	$P_{=?}$ [$F^{\leq 20}$ ped=true & εd=true]	0.6226	0.6415
12	$R_{\{\text{“ εc\_total”}\}=?}$ [$C^{\leq 100}$],	13.8200	13.8600
13	$R_{\{\text{“step”}\}=?}$ [F εc=true]	7.4285	7.1428
14	$R_{\{\text{“error”}\}=?}$ [S]	0.2400	0.2400
15	$R_{\{\text{“error”}\}=?}$ [I=17]	0.2400	0.2400

Table 13: Sample properties for circuit in Figure 16

The non-transient properties yield identical results for the two models (Properties 2, 3, 4, 5, 6, 14 and 15), but as the modular model is time-abstract, transient properties show some variance from the more accurate, but less scalable single model. Compilable code for modular DTMC and the single DTMC is located in Appendices A and B respectively.

5.4 Example of Presented DTMC Method handling Reconvergent Fan-out

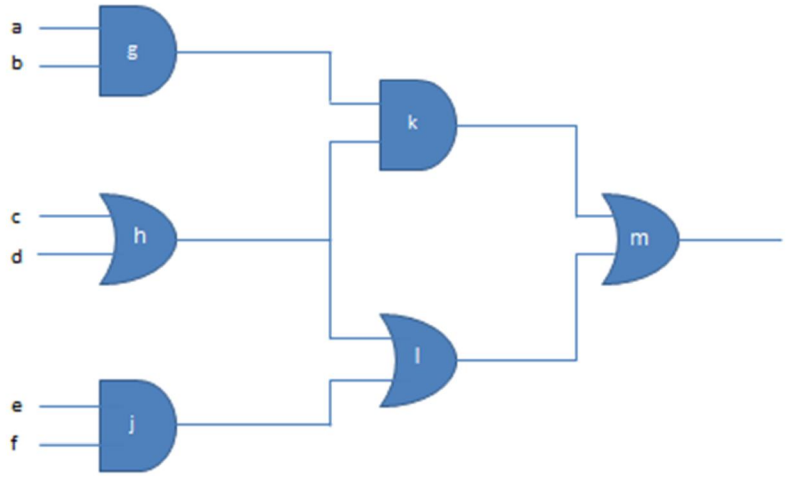


Figure 29: combinational circuit for comparison with First Pass method

For the depicted circuit in Figure 29, with a fifty percent probability of the inputs to the circuit being either a logic low or logic high, and a ten percent probability of errors originating at the output of each gate (for both 0→1 and 1→0 errors, Table 13 gives the resulting probabilities for ϵm using the First Pass method from [1] without correlation coefficients (incorporating correlation coefficients highly increases the accuracy of the First Pass method), and the proposed DMTC method (incorporating in correlation coefficients .

Model	Actual	First Pass Method without correlation coefficients	Presented DTMC Method
εm*, probability of error at *m	0.2267	0.2274	0.2267

Table 14: Results for εm from the circuit in Figure 29

In this presented model for the six gate circuit, there are $2^{\text{number of primary inputs + number of gates}} = 2^{6+6} = 4096$ states.

Chapter 6: Summary and Possible Future Work

This paper describes a framework for an accurate method for probabilistic analysis for logic circuits. The primary advantages of the model are:

- Complete verification, as opposed to partial verification provided by simulation
- Handling reconvergent fan-out without additional complexity
- Property specifications that allow for depth and breadth of analysis, including error dependency of nodes in the node of interest's fan-in cone.

While demonstrated on Boolean logic gates, the model can be modified to use on developing circuit technologies such as reversible and quantum gates. Possible future work could include extended the model to analyze sequential circuits, where inputs to the combinational logic would include feedback from circuit outputs.

References

[1] M. R. Choudhury, K. Mohanram. Reliability Analysis of Logic Circuits. In *IEEE Trans. Computer-Aided Design.* pp. 392-405. March 2009.

[2] N. Miskov-Zivanov, D. Marculescu. Circuit Reliability Analysis Using Symbolic Techniques. In *IEEE Trans. Computer-Aided Design.* pp. 2638-2649. December 2006.

[3] C.Y. Tsui, J. Monterio, M. Pedram. Power Estimation Methods for Sequential Logic Circuits. In *IEEE Trans. VLSI Systems (Volume 3, Issue: 3).* pp. 404-416. September 1995.

[4] M. Kwiatkowska, G. Norman and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of LNCS, pages 585-591, Springer, 2011.

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

[6] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies, C. E. Shannon and J. McCarthy, Eds. Princeton, NJ; Princeton University Press*. pp. 43-98, 1956.

[7] R. Alur, C. Courcoubeti, D.Dill. Model-checking for Probabilistic Real-time Systems. In *Proc. of the 18th ICALP*, LNCS 510, 1991.

[8] M. L. Puterman, L. Martin. Markov Decision Processes, Discrete Stochastic Dynamic Programming. *John Wiley & Sons, Inc*. New York.1994.

[9] M. Kwiatkowski, G. Norman, and D. Parker. Stochastic model checking. In *M. Bernardo and J. Hilston, editors, Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of LNBCS. Springer, 2007.

[10] D. Bhaduri, S. Shukla, P. Graham, M. Gokhale, Scalable Techniques and Tools for Reliability Analysis of Large Circuits. In *VLSI Design, 2007*. pp. 705-710. January 2007.

[11] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla. Evaluating the reliability of NAND multiplexing. In *IEEE Trans, Computer-Aided Design.* pp. 1629-1637. October 2005.

[12] T. Rejimon, S. Bhanja. Probabilistic Error Model for Unreliable Nano-logic Gates. In *IEEE-NANO 2006*. pp. 47-50. June 2006.

[13] S. Krishnaswamy, G.F. Viamontes, I.L. Markov, J.P. Hayes. Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices. In *Proc. of Design, Automation and Test in Europe, 2005.* pp. 282-287 Vol. 1. March 2005.

[14] T. Raejimon, S. Bhanja. Scalable Probabilistic Computing Models using Baynesian Networks. In *Proc. Int. Midwest Symp. Circuits and Systems.* pp. 712-715. August 2005.

[15] I. Bahar, J. L. Mundy, J. Chen. A Probabilistic-Based Design Methodology for Nanoscale Computation. In *Intl. Conf. on Computer Aided Design, 2003.* pp. 480-486. November 2003.

[16] D. Bhaduri, S. Shukla. NANOLAB: A Tool for Evaluating Reliability of Defect-Tolerant Nano Architectures, In *IEEE Trans. Nanotechnology Volume 4, Issue 4.* pp. 381-394, July 2005.

Appendix A: PRISM Code for Modular DTMC Model of Circuit from Figure 16

```
dtmc

const double pa = 0.5;        // probability of primary input 'a' being true (i.e. logic 1)
const double pb = 0.5;        // probability of primary input 'b' being true (i.e. logic 1)
const double c01 = 0.1;     // probability of 0 -> 1 error originating at node 'c'
const double c10 = 0.1;     // probability of 1 -> 0 error originating at node 'c'
const double d01 = 0.1;     // probability of 0 -> 1 error originating at node ‘d’
const double d10 = 0.1;     // probability of 1 -> 0 error originating at node 'd'


//------------------------- error-free circuit structure ----------------------------------------------------
formula c=true = a=false;                 // c = INV(a)
formula d=true = b=false&c=false;    // d = NAND(c,b)


//------------------------- input transitions ----------------------------------------------------------------
module circuit_inputs
   a:bool init false;  // logic level of input node 'a'

   [update]a=false -> 1-pa:(a'=false) + pa:(a'=true);
   [update]a=true -> 1-pa:(a'=false) + pa:(a'=true);
endmodule


module b_input
   b:bool init false;  // logic level of input node 'b'

   [update]b=false -> 1-pb:(b'=false) + pb:(b'=true);
   [update]b=true -> 1-pb:(b'=false) + pb:(b'=true);
endmodule


//------------------------- DTMCS for error estimation at output of inverter ‘c’ ----------------------------
module inv_c
   ec: bool init false;  // error at 'c', where 'true' is an error, and 'false' is error-free
   cef: bool init false;   // expected/error-free 'c': 'false' = logic low, 'true' = logic high
```

```
// from any state, ‘ec’ transitions to ’true’ with probability 'c01' or ‘c10’
   [update]a=false -> c10:(ec'=true)&(cef'=true) + 1-c10:(ec'=false)&(cef'=true);
   [update]a=true -> c01:(ec'=true)&(cef'=false) + 1-c01:(ec'=false)&(cef'=false);
endmodule

//-------------------------- DTMCs for error estimation at output of 2-input NAND ‘d’ ----------
----------
module nand_d
   ed: bool init false;  // error at 'd', where 'true' is an error, and 'false' is error-free
   def: bool init false;   // expected/error-free 'd': 'false' = logic low, 'true' = logic high
   ped: bool init false; // 'true; is a propagated error to 'd', 'false' is no propagated error
to 'd'.

  // from any state of ‘ed’, if there is an error propagated through ‘d’,
   // ‘ed’ transitions to ‘true’ with probability '1-d01' or ‘1-d10’
   [update]b=true & c=false & ec=true -> d01:(ed'=false)&(def'=true)&(ped'=true) + 1-
d01:(ed'=true)&(def'=true)&(ped'=true);
   [update]b=true & c=true  & ec=true -> d10:(ed'=false)&(def'=false)&(ped'=true) + 1-
d10:(ed'=true)&(def'=false)&(ped'=true);

   // from any state of ‘ed’, if there is no error propagated through ‘d’,
   // ‘ed’ transitions to ‘true’ with probability 'd01' or ‘d10’
   [update]b=false & ec=false        -> d10:(ed'=true)&(def'=true)&(ped'=false) + 1-
d10:(ed'=false)&(def'=true)&(ped'=false);
   [update]b=false & ec=true         -> d10:(ed'=true)&(def'=true)&(ped'=true) + 1-
d10:(ed'=false)&(def'=true)&(ped'=true);
   [update]b=true & c=false& ec=false -> d10:(ed'=true)&(def'=true)&(ped'=false) + 1-
d10:(ed'=false)&(def'=true)&(ped'=false);
   [update]b=true & c=true & ec=false -> d01:(ed'=true)&(def'=false)&(ped'=false) + 1-
d01:(ed'=false)&(def'=false)&(ped'=false);
endmodule


//-------------------------- reward structures ------------------------------------------------------

// assigns a reward of 1 anytime there is an error at 'd'
rewards "ed_total"
   ed=true : 1;
endrewards

// assigns a reward of 1 anytime there is an error at 'd' as a result of the propagated
error at 'c'
```

```
rewards "ed_ec"
   (ed=true &ped=true): 1;
endrewards

// assigns a reward of 1 anytime there is an error at 'd', when expecting a '0'
rewards "ed_def_1"
   (ed=true & def=true): 1;
endrewards

// assigns a reward of 1 anytime there is an error at 'd', when expecting a '1''
rewards "ed_def_0"
   (ed=true & def=false): 1;
endrewards

// assigns an reward of 1 to every time step
rewards "step"
   a=true: 1;
   a=false: 1;
endrewards

// assigns a reward to 1 anytime there is an error anywhere in the circuit
rewards "total_errors"
   ec=true: 1;
   ed=true: 1;

endrewards
```

Appendix B: PRISM Code for Singular DTMC Model of Circuit from Figure 16

```
dtmc

const double pa = 0.5;        // probability of primary input 'a' being true (i.e. logic 1)
const double pb = 0.5;        // probability of primary input 'b' being true (i.e. logic 1)
const double c01 = 0.10;      // probability of 0 -> 1 error originating at node 'c'
const double c10 = 0.10;      // probability of 1 -> 0 error originating at node 'c'
const double d01 = 0.10;      // probability of 0 -> 1 error originating at node ‘d’
const double d10 = 0.10;      // probability of 1 -> 0 error originating at node 'd'



//------------------------- error-free circuit structure ----------------------------------------------------
-----------
formula c=true = a=false;            // c = INV(a)
formula d=true = b=false&c=false;    // d = NAND(c,b)


//------------------------- input transitions ----------------------------------------------------------------
-----------
module circuit_inputs
   a:bool init false;  // logic level of input node 'a'
   b:bool init false;  // logic level of input node 'b'

 ec: bool init false;
 ed: bool init false;
   [update]a=false ->
((1-pa)*(1-pb)*(1-c10)*(1-d10)) :(a'=false)&(b'=false)&(ec'=false)&(ed'=false)+
((1-pa)*(1-pb)*(1-c10)*d10)     :(a'=false)&(b'=false)&(ec'=false)&(ed'=true)+
((1-pa)*(1-pb)*c10    *(1-d10)) :(a'=false)&(b'=false)&(ec'=true) &(ed'=false)+
((1-pa)*(1-pb)*c10    *d10)     :(a'=false)&(b'=false)&(ec'=true) &(ed'=true) +
((1-pa)*pb    *(1-c10)*(1-d01)) :(a'=false)&(b'=true) &(ec'=false)&(ed'=false)+
((1-pa)*pb    *(1-c10)*d01)     :(a'=false)&(b'=true) &(ec'=false)&(ed'=true)+
((1-pa)*pb    *c10    *(1-d10)) :(a'=false)&(b'=true) &(ec'=true) &(ed'=true)+
((1-pa)*pb    *c10    *d10)     :(a'=false)&(b'=true) &(ec'=true) &(ed'=false) +
(pa    *(1-pb)*(1-c01)*(1-d10)) :(a'=true) &(b'=false)&(ec'=false)&(ed'=false)+
(pa    *(1-pb)*(1-c01)*d10)     :(a'=true) &(b'=false)&(ec'=false)&(ed'=true)+
(pa    *(1-pb)*c01    *(1-d10)) :(a'=true) &(b'=false)&(ec'=true) &(ed'=false)+
(pa    *(1-pb)*c01    *d10)     :(a'=true) &(b'=false)&(ec'=true) &(ed'=true)+
(pa    *pb    *(1-c01)*(1-d10)) :(a'=true) &(b'=true)&(ec'=false)&(ed'=false)+
(pa    *pb    *(1-c01)*d10)     :(a'=true) &(b'=true)&(ec'=false)&(ed'=true)+
```

```
(pa    *pb    *c01    *(1-d01)) :(a'=true) &(b'=true)&(ec'=true) &(ed'=true)+
(pa    *pb    *c01    *d01)     :(a'=true) &(b'=true) &(ec'=true) &(ed'=false);

[update]a=true ->
((1-pa)*(1-pb)*(1-c10)*(1-d10)) :(a'=false)&(b'=false)&(ec'=false)&(ed'=false)+
((1-pa)*(1-pb)*(1-c10)*d10)     :(a'=false)&(b'=false)&(ec'=false)&(ed'=true)+
((1-pa)*(1-pb)*c10    *(1-d10)) :(a'=false)&(b'=false)&(ec'=true) &(ed'=false)+
((1-pa)*(1-pb)*c10    *d10)     :(a'=false)&(b'=false)&(ec'=true) &(ed'=true) +
((1-pa)*pb    *(1-c10)*(1-d01)) :(a'=false)&(b'=true) &(ec'=false)&(ed'=false)+
((1-pa)*pb    *(1-c10)*d01)     :(a'=false)&(b'=true) &(ec'=false)&(ed'=true)+
((1-pa)*pb    *c10    *(1-d10)) :(a'=false)&(b'=true) &(ec'=true) &(ed'=true)+
((1-pa)*pb    *c10    *d10)     :(a'=false)&(b'=true) &(ec'=true) &(ed'=false) +
(pa    *(1-pb)*(1-c01)*(1-d10)) :(a'=true) &(b'=false)&(ec'=false)&(ed'=false)+
(pa    *(1-pb)*(1-c01)*d10)     :(a'=true) &(b'=false)&(ec'=false)&(ed'=true)+
(pa    *(1-pb)*c01    *(1-d10)) :(a'=true) &(b'=false)&(ec'=true) &(ed'=false)+
(pa    *(1-pb)*c01    *d10)     :(a'=true) &(b'=false)&(ec'=true) &(ed'=true)+
(pa    *pb    *(1-c01)*(1-d10)) :(a'=true) &(b'=true)&(ec'=false)&(ed'=false)+
(pa    *pb    *(1-c01)*d10)     :(a'=true) &(b'=true)&(ec'=false)&(ed'=true)+
(pa    *pb    *c01    *(1-d01)) :(a'=true) &(b'=true)&(ec'=true) &(ed'=true)+
(pa    *pb    *c01    *d01)     :(a'=true) &(b'=true) &(ec'=true) &(ed'=false);


endmodule


//------------------------- reward structures ------------------------------------------------------

// assign a reward of 1 any step there is an error at 'd'
rewards "ed_total"
   ed=true : 1;
endrewards

// assign a reward of 1 any step there is an error at 'd' as a result of the propagated
error at 'c'
rewards "ed_ec"
   (ec=true & ed=true) : 1;
endrewards

// assign a reward of 1 any step there is an error at 'd', when expecting a '0'
rewards "ed_def_0"
  (a=false&b=true&ed=true): 1;
endrewards
```

```
// assign a reward of 1 any step there is an error at 'd', when expecting a '1'
rewards "ed_def_1"
   (a=true & ed=true): 1;
   (a=false& b=false&ed=true): 1;
endrewards

// assign an reward of 1 to every step
rewards "step"
   a=true:  1;
   a=false: 1;
endrewards

// assign a reward of 1 any step there is an error anywhere in the circuit
rewards "total_errors"
   ec=true: 1;
   ed=true: 1;
endrewards
```

Appendix C: PRISM Code for the Circuit from Figure 30

```
dtmc


// -------------------- (fixed) error probabilities ----------------------------

const double pa = 0.5;       //Prob(a=1)=0.5
const double pb = 0.5;       //Prob(b=1)=0.5
const double pc = 0.5;       //Prob(c=1)=0.5
const double pd = 0.5;       //Prob(d=1)=0.5
const double pe = 0.5;       //Prob(e=1)=0.5
const double pf = 0.5;       //Prob(f=1)=0.5

const double g01 = 0.1;     //Prob(g0->1)=0.1
const double g10 = 0.1;     //Prob(g1->0)=0.1
const double h01 = 0.1;     //Prob(h0->1)=0.1
const double h10 = 0.1;     //Prob(h1->1)=0.1
const double j01 = 0.1;     //Prob(j0->1)=0.1
const double j10 = 0.1;     //Prob(j1->1)=0.1
const double k01 = 0.1;     //Prob(k0->1)=0.1
const double k10 = 0.1;     //Prob(k1->1)=0.1
const double l01 = 0.1;     //Prob(l0->1)=0.1
const double l10 = 0.1;     //Prob(l1->1)=0.1
const double m01 = 0.1;     //Prob(m0->1)=0.1
const double m10 = 0.1;     //Prob(m1->1)=0.1


//--------------------------error-free circuit structure -------------
formula g=true  = a=true & b=true;  // g = a & b
formula h=true  = c=true | d=true;  // h = c | d
formula j=true  = e=true & f=true;  // j = e & f
formula k=true  = g=true & h=true;  // k = g & h
formula l=true  = h=true | j=true;  // l = h & j
formula m=true  = k=true | l=true;  // m = k & l




// ------------------------ input transitions ----------------

module circuit_inputs
```

```
   a:bool init false;  // logic level of input node 'a'
   b:bool init false;  // logic level of input node 'b'
   c:bool init false;  // logic level of input node 'c'
   d:bool init false;  // logic level of input node 'd'
   e:bool init false;  // logic level of input node 'e'
   f:bool init false;  // logic level of input node 'f'

   [update]a=false | a=true -> 1-pa:(a'=false) + pa:(a'=true);
   [update]b=false | b=true -> 1-pb:(b'=false) + pb:(b'=true);
   [update]c=false | c=true -> 1-pc:(c'=false) + pc:(c'=true);
   [update]d=false | d=true -> 1-pd:(d'=false) + pd:(d'=true);
   [update]e=false | e=true -> 1-pe:(e'=false) + pe:(e'=true);
   [update]f=false | f=true -> 1-pf:(f'=false) + pf:(f'=true);
endmodule


//-------------------------------------------------------------------------

module AND_g
   eg : bool init false;
   [update]a=false | b=false -> 1-g01:(eg'=false) + g01:(eg'=true);
   [update]a=true  & b=true  -> 1-g10:(eg'=false) + g10:(eg'=true);
endmodule

//-------------------------------------------------------------------------

module OR_h
   eh : bool init false;
   ei : bool init false;
   [update]c=false & d=false -> 1-h01:(eh'=false)&(ei'=false) + h01:(eh'=true)&(ei'=true);
   [update]c=true  | d=true  -> 1-h10:(eh'=false)&(ei'=false) + h10:(eh'=true)&(ei'=true);
endmodule

//-------------------------------------------------------------------------

module AND_j
   ej : bool init false;
   [update]e=false | f=false -> 1-j01:(ej'=false) + j01:(ej'=true);
   [update]e=true  & f=true  -> 1-j10:(ej'=false) + j10:(ej'=true);
endmodule
```

```
//-------------------------------------------------------------------------

module AND_k
   ek : bool init false;

   //error propagates through from transitive fan-in cone
   [update]g=false & h=false & eg =true & eh=true  -> k01:(ek'=false) + 1-k01:(ek'=true);
   [update]g=false & h=true  & eg=true & eh =false  -> k01:(ek'=false) + 1-k01:(ek'=true);
   [update]g=true  & h=false & eg =false & eh=true  -> k01:(ek'=false) + 1-k01:(ek'=true);
   [update]g=true  & h=true  &(eg=true | eh=true) -> k10:(ek'=false) + 1-k10:(ek'=true);


  // no error propagate trough from transitive fan-in cone
   [update]g=false & h=false &(eg =false | eh =false)-> k01:(ek'=true) + 1-k01:(ek'=false);
   [update]g=false & h=true  &(eg =false | eh=true) -> k01:(ek'=true) + 1-k01:(ek'=false);
   [update]g=true  & h=false &(eg=true | eh =false) -> k01:(ek'=true) + 1-k01:(ek'=false);
   [update]g=true  & h=true  & eg =false & eh =false  -> k10:(ek'=true) + 1-
k10:(ek'=false);
endmodule

 //-------------------------------------------------------------------------

module OR_l
   el : bool init false;
   //error propagates through from transitive fan-in cone
   [update]h=false & j=false &(ei=true | ej=true) -> l01:(el'=false) + 1-l01:(el'=true);
   [update]h=false & j=true  & ei =false & ej=true  -> l10:(el'=false) + 1-l10:(el'=true);
   [update]h=true  & j=false & ei=true & ej =false  -> l10:(el'=false) + 1-l10:(el'=true);
   [update]h=true  & j=true  & ei=true & ej=true  -> l10:(el'=false) + 1-l10:(el'=true);

  // no error propagate trough from transitive fan-in cone
   [update]h=false & j=false &  ei =false & ej =false  -> l01:(el'=true) + 1-l01:(el'=false);
   [update]h=false & j=true  & (ei=true | ej =false) -> l10:(el'=true) + 1-l10:(el'=false);
   [update]h=true  & j=false & (ei =false | ej=true) -> l10:(el'=true) + 1-l10:(el'=false);
   [update]h=true  & j=true  & (ei =false | ej =false) -> l10:(el'=true) + 1-l10:(el'=false);
endmodule

 //-------------------------------------------------------------------------

module OR_m
   em : bool init false;
   //error propagates through from transitive fan-in cone
```

```
   [update]k=false & l=false &(ek=true | el=true) -> m01:(em'=false) + 1-m01:(em'=true);
   [update]k=false & l=true  & ek =false & el=true  -> m10:(em'=false) + 1-
m10:(em'=true);
   [update]k=true  & l=false & ek=true & el =false  -> m10:(em'=false) + 1-
m10:(em'=true);
   [update]k=true  & l=true  & ek=true & el=true  -> m10:(em'=false) + 1-m10:(em'=true);

   //error propagates through from fan-in cone
   [update]k=false & l=false &  ek =false & el =false  -> m01:(em'=true) + 1-
m01:(em'=false);
   [update]k=false & l=true  & (ek=true | el =false) -> m10:(em'=true) + 1-
m10:(em'=false);
   [update]k=true  & l=false & (ek =false | el=true) -> m10:(em'=true) + 1-
m10:(em'=false);
   [update]k=true  & l=true  & (ek =false | el =false) -> m10:(em'=true) + 1-
m10:(em'=false);
endmodule
```