ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG

INSTITUT FÜR INFORMATIK
Lehrstuhl für Rechnerarchitektur
Prof. Dr. Bernd Becker

DIPLOMARBEIT

Simulation of Dynamic Effects of Resistive Open Defects

Alejandro Czutro
(Student-ID 1133719)

Supervision: Dr. Ilia Polian

17th April 2007

Erklärung

Hiermit erkläre ich, **Alejandro Czutro**, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe, und dass ich alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum	Unterschrift

Acknowledgements

I am grateful to Sandeep Gupta and Shahdad Irajpour of University of Southern California Los Angeles for providing deep insight into the implementation of their gate delay fault simulator [19].

Many thanks to Dr. Ilia Polian for the supervision of my work during the last six months; and to him and to Mr. Piet Engelke for meticulous proofreading of the present manuscript.

Finally, I would like to thank my wife, Mrs. Kinga Czutro, for supporting me during the last four years of study, and for working especially hard during the last six months, in order to permit me concentrate on this work and finish it on time.

Alejandro Czutro, Freiburg, 17th April 2007

Contents

List of Figures

List of Tables

1

Introduction

Nowadays, electronics belong to modern life more than ever before. The automotive industry is only one example of an area which cannot do without integrated circuit (IC) technology any more. At the same time, as the areas in which IC technology is necessary, expand steadily, the complexity of digital systems rises constantly. This, together with increased device operation speed and component miniaturisation as a response to the attempt of achieving higher performances, has as consequence that it is practically impossible to guarantee that a digital IC is defect-free.

Since a defective IC placed onto a printed circuit board usually makes the whole board and all other (possibly fault-free) components on it useless, the costs of shipping a defective IC may exceed the profit margin by far [34]. Due to this fact, the correct function of all delivered ICs must be verified by testing after they have been manufactured. Testing represents the single largest manufacturing expense in the semiconductor industry, costing over $40 billion a year (by 2003, [21]).

The general strategy consists in applying a set of input vectors to the circuit under test (CUT) and verifying if its outputs are as expected. However, digital testing comprises many more tasks, which are steadily being improved in order to: 1) meet the requirement of testing constantly more and more complex becoming ICs; 2) reduce the costs of testing. The two most important tasks are test pattern generation (TPG) and fault simulation.

TPG is the task of generating a set of input vectors (called test patterns) to be applied to the CUT. A set of test patterns is called test set. A test pattern p is said to detect a fault f, if the response of the fault-free circuit differs from the response of the circuit with the fault f, when p is applied to the circuit's inputs. A test set is said to detect a fault if at least one of the test patterns in the test set detects the fault. The effectiveness of TPG is

measured by the percentage of faults that are detected (covered) by applying the generated test set. This measure of the quality of the test set is called fault coverage.

Fault simulation is the process of determining the set of faults which are detected by applying a test set to a circuit. Fault simulation is necessary to measure fault coverage without having to apply the generated test set to an actual circuit. Fault simulation is also a very important part of the overall TPG process, which usually consists of several phases. The goal of each phase depends on the fault coverage achieved by the result of the preceding phase. Fault simulation of a test pattern p for a fault f consists of fault-free simulation followed by faulty-circuit simulation. Fault-free simulation is the process of determining the steady state of the fault-free CUT after applying p to its inputs. Faulty-circuit simulation determines the steady state of the CUT when it is affected by f after applying p to its inputs. The results of fault-free and faulty-circuit simulation are compared. If these differ, f is detected by p.

Among the typical defects encountered in today VLSI chips are missing or broken contacts, opens (broken lines), shorts and bridges (unintended contacts), contacts with too low conductance, surface impurities, etc. Some affect the functional behaviour of the circuit (i.e. cause the circuit to produce a wrong output or to produce an output which cannot be interpreted as a valid logical value), some affect its timing behaviour (i.e the circuit produces the right outputs, but these arrive too late), some do both, and others have further effects which do not need to be mentioned here. As it is not possible to list all defects that may occur in a circuit, so-called fault models are used to describe the faulty behaviour of a circuit having one or several defects.

The most popular fault model used in practice is the single stuck-at fault model [10], [14], [34, p. 16], [21]. Under this fault model, a single line in the circuit is assumed to be affected. That line has always a fixed logic value, regardless of what inputs are supplied to the circuit. Despite the assumption that only one single line can be affected, working under the stuck-at fault model detects a large amount of complex defects which are usually modelled by other fault models.

Due to the increased interconnection density of today's circuits, defects that affect the interconnect are becoming very frequent. Among these are opens and bridges, where opens constitute the major part of those defects which remain undetected during the test phase [32].

Traditionally, open defects were defined as unconnected nodes in the manufactured circuit that were connected in the original design. However,

open defects can also still connect the nodes, but only weakly, by introducing between the linked points a resistance that is higher than expected but finite [17], [32]. In [43], opens are classified into strong and weak ones. Strong opens are those with a very high resistance (more than 10 MΩ) and are thus treated as fully broken connections. These defects result in a memory behaviour at the fault site [45]. They can be modelled as stuck-open, stuck-on, or even stuck-at faults. In the past, strong opens have been studied extensively [37], [17], [7], [22].

Weak or resistive opens (i.e. opens with a low or moderate resistance less than 10 MΩ) manifest themselves as lines with an elevated resistance or as resistive vias and contacts. They still let the circuit work, but increase the delay of paths (sequences of lines connected by gates) going through the fault site [25], [29]. Not all these defects can be modelled effectively by the stuck-at fault model [2], [9], [8]. Thus, they are modelled by delay fault models which assume that one single gate or path is slow to propagate $0 \rightarrow 1$ or $1 \rightarrow 0$ transitions, delaying them by an additional unknown amount of time. Hence, resistive opens require a delay testing strategy. Delay testing strategies are characterised by the application of two successive test patterns (a test pair). The first test pattern brings the circuit into a known and stable state. The second pattern changes the state of the circuit. If the circuit's stabilisation after the state change takes place after a certain threshold time, the delay fault is detected. [43] shows that in modern deep sub-micron technology, the percentage of weak opens is high enough to require delay fault testing.

The amount of time by which the transitions are delayed is called the size of the delay fault. It depends on the open's resistance, which is an unpredictable random parameter [41]. A resistive open may be detected by a pair of test patterns if it has a certain resistance, while the same pair might not detect an open at the same location if the open has a different resistance. Since the resistance of an open in an actual manufactured circuit cannot be known a priori, it is necessary to generate more than one pair of test patterns to detect the open. Enlarging the range of resistances for which the open is detected is equivalent to increasing the probability that the open is detected if it is present.

The aim of this work was to design and implement a simulator which is able to compute, for each delay fault in a given fault list, a set of resistance ranges, for which the resistive open modelled by the fault is detected by a given test set. That set of resistance ranges is called the C-ADI ("Covered-Analogous Detection Interval", [40]) of the fault. Furthermore, the simulator should be able to compute the fault coverage the given test set achieves. The

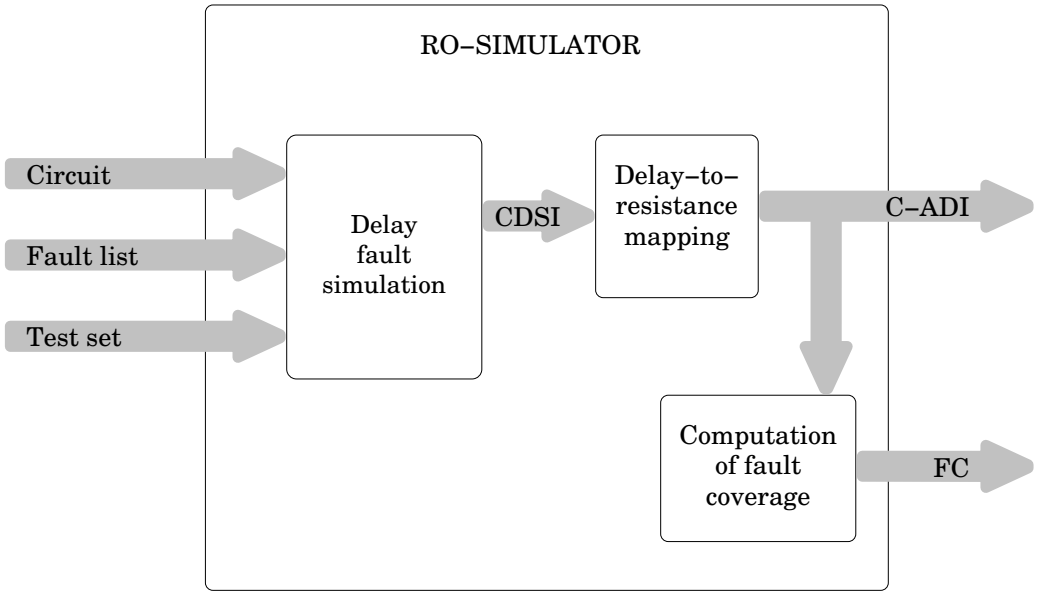


FIGURE 1.1: THE RO-SIMULATOR

proposed simulator is called **RO-simulator** (stands for "resistive-opens-simulator") throughout this thesis.

Figure 1.1 illustrates the overall structure of the RO-simulator. The input data of the simulator are a combinational circuit (a circuit without memory elements, which computes a Boolean function) or a sequential circuit (a circuit with memory elements, which implements a Mealy finite state machine), a list of delay faults and a set of test pattern pairs. The RO-simulator works in three phases. In Phase 1, delay fault simulation is performed. The result of this phase is a set of delay size intervals (one set for each fault) with the property that a fault is detected by the test set if its size is in one of its corresponding delay size intervals. The collection of sets of intervals is denoted by CDSI in Figure 1.1. In Phase 2, the set of intervals of delay fault sizes of each fault is transformed into the fault's C-ADI. This is one of the outputs of the RO-simulator. These data are also input of the third phase, in which the RO-simulator computes the achieved fault coverage. The fault coverage is computed based on C-ADI and expresses the probability that each resistive open is detected by the test set. Fault coverage turns into a probability because it depends on the opens' resistances. As was said before, the resistance of an open is an unpredictable value.

The implementation of Phase 1 of the RO-simulator is based on the delay fault simulation method RTM ("Ranges-type testing methodology") by Pramanick and Reddy [36]. That simulation method is called **`PR-simulator`** ("Pramanick-Reddy-simulator") in this thesis. The PR-simulator is an extension of a method introduced in [20], and is extended itself by Irajpour, Gupta and Breuer in [19]. In the RO-simulator's Phase 1, the PR-simulator's techniques for the description of dynamic behaviour of signals and for the propagation of fault effects through the circuit have been improved by calculating the cumulative fault effect of all inputs of a logic gate rather than considering all inputs explicitly. A further improvement of the PR-simulator, which was designed for combinational circuits, permits the RO-simulator to be also used on sequential circuits.

In Phase 2, the RO-simulator converts the set of delay size intervals of each fault into its C-ADI. The base for this conversion is a delay-to-resistance mapping $\delta \mapsto r$ with the meaning that a resistive open with resistance r produces a delay fault of size δ. In this first implementation of the simulator, a trivial delay-to-resistance mapping is used. This mapping assumes that the additional line delay caused by an open is linearly proportional to the open's resistance. Physically accurate mappings, such as one introduced in [24], can be easily integrated into the RO-simulator.

In Phase 3, the RO-simulator computes the fault coverage the given test set achieves. In order to do this, existing fault coverage metrics for resistive bridging faults [40], [38], [39], [23], [11], [12], [13] were adapted to the case of resistive opens.

This first implementation of the RO-simulator was tested on ISCAS 85 and ISCAS 89 circuits. Results are reported and analysed in this thesis.

The thesis is organised as follows. The next chapter introduces some conventions for this work and the basics on testing digital ICs. In Chapter 3 the requirements the RO-simulator must meet are formulated in a formal way, and the RO-simulator's overall structure is presented. Chapter 4 introduces some definitions which are necessary to understand Chapter 5, in which the algorithms of Phase 1 are presented in detail and illustrated by means of examples. In Chapter 6 fault coverage metrics are discussed. Experimental results are reported in Chapter 7. Chapter 8 concludes the work.

2

Preliminaries

While there is a wide variety of text books providing a good introduction into the area of testing digital ICs ([1], [21] and [4] are only a few examples), this chapter has been composed in order to introduce the basic concepts on testing which are needed to understand the present thesis. Additionally, the terminology used in all the other chapters is introduced.

Since digital testing is a discipline with a strong connection to the practice, new research topics typically emerge after new problems occur depending on the technology development. Much of the conducted research does not constitute a fundamental breakthrough. Instead, existing methods are gradually improved [34, Chapter 2]. That is also the characteristic of the work presented in this thesis. The RO-simulator combines several already existing techniques, while the original work consists in modifying those techniques to improve performance and to allow them to work together. The second purpose of this chapter is to help to distinguish between original work and those results obtained by other authors in the past. No result which can be found in this chapter has been obtained by ourselves. In all other chapters, non-original results are still referenced.

2.1 Circuits

A digital circuit is a device that processes input data and produces output data. A **`combinational`** circuit C with n inputs and m outputs **`implements`** or **`computes`** the Boolean function $g_C : \mathbb{B}^n \to \mathbb{B}^m$. For the example circuit in Figure 2.1, $n = 3$, $m = 2$ and g_C is $(a, b, c) \mapsto (a \wedge b, b \vee c)$.

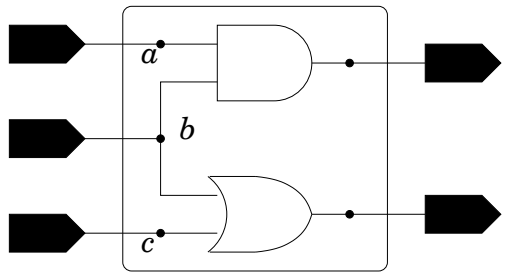


Figure 2.1: Example combinational circuit

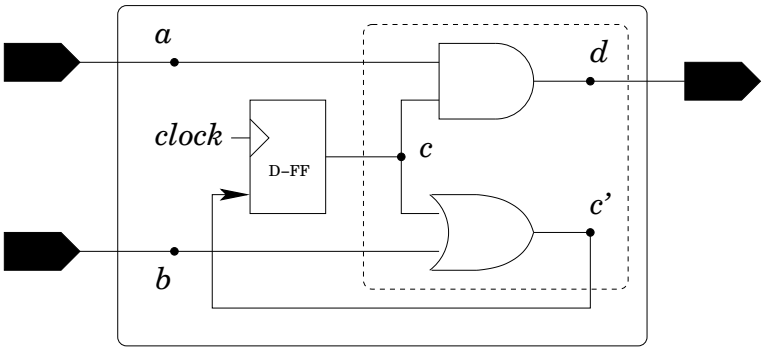


Figure 2.2: Example sequential circuit

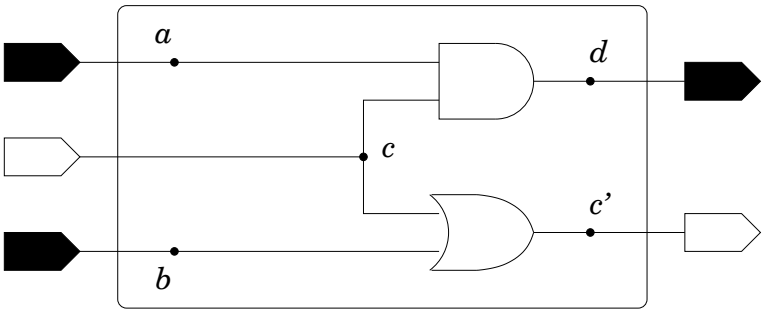


Figure 2.3: Example sequential circuit, simplified representation

Circuits which contain memory elements are called **sequential**. Figure 2.2 pictures an example sequential circuit. The box with the inscription "D-FF" is a D-flip-flop, a very common type of memory element.

Memory elements in sequential circuits are clocked, that means they are connected to a device which generates a **clock signal**. The clock signal oscillates between logic 1 and logic 0, normally with a 50% duty cycle (i.e. it produces a square waveform), and is used to synchronise the actions of all memory elements. These are designed such that they can store a new value only while the clock is high (or only when the clock is low, depending on the implementation). A **clock cycle** is composed of one falling (a $1 \to 0$ transition) and one rising edge (a $0 \to 1$ transition) of the clock and its length is called **clock period** or **clock sampling time**. This length is denoted by **TC**. In normal operation mode, new input vectors are applied to the sequential circuit once every clock cycle. A clock cycle is also called a **time frame**.

A sequential circuit C with n inputs, m outputs and k flip-flops can be regarded as implementation of a Mealy finite state machine with 2^k or less states. The states are encoded by the data stored in the flip-flops, while the combinational logic of the circuit (inside the dashed box in the example) computes the output function $g_C : \mathbb{B}^n \times \mathbb{B}^k \to \mathbb{B}^m$ and the transition function $\delta_C : \mathbb{B}^n \times \mathbb{B}^k \to \mathbb{B}^k$, which depend both on the inputs and on the present state. The g_C-values are sent to the outputs and the δ_C-values are stored back into the flip-flops. In the current example, $n = 2$, $m = 1$ and $k = 1$. Further, the machine has two states, 0 and 1, encoded by c. g_C maps $\big((a, b), (c)\big)$ to $(a \wedge c)$ and δ_C maps $\big((a, b), (c)\big)$ to $(c \vee b)$.

For some testing tasks, it is often convenient to ignore the memory elements and to only consider the sequential circuit's combinational core. The outputs of memory elements (e.g. c in Figures 2.2 and 2.3) are treated as additional inputs to the combinational core. These are called **secondary inputs**. The inputs of memory elements (e.g. c') are treated as additional outputs of the combinational core. These are called **secondary outputs**. Regular inputs and outputs (a, b and d) are then called **primary inputs** and **primary outputs**, respectively. Instead of handling the output function g_C and the transition function δ_C separately, only one global function $g_S : \mathbb{B}^{n+k} \to \mathbb{B}^{m+k}$, which is computed by the combinational core, needs to be considered. In the current example, g_S maps (a, b, c) to $(a \wedge c, c \vee b)$.

Each secondary input corresponds to exactly one secondary output, namely to the secondary output which feeds its source memory element. In a time frame i, the values of the secondary inputs are the values of their corresponding secondary outputs in time frame $i - 1$.

When simulating sequential circuits, the first time frame needs special treatment, as only the primary inputs and outputs are accessible from the outside of the circuit. There are techniques that make secondary inputs partially or fully accessible during test application, but these techniques have some drawbacks (see Section 2.3.2).

When secondary inputs and outputs are not accessible from the outside of the circuit, it is necessary to bring the circuit to a known state (i.e. to cause known values to be stored into the memory elements) before performing simulation. If the hardware implements a reset function, resetting the hardware will bring it to a known state. If no hardware reset is possible, one can try to find a so-called **synchronising** or **homing** sequence. In the example, the input $(1, 1)$ can be applied to the primary inputs of the circuit while the secondary input c has a defined but unknown value. Then, the value of the secondary output c' is 1, as OR gates always produce the output 1 if at least one of their inputs has the logic value 1. In the next time frame, a logic 1 is stored in the flip-flop. Altogether, applying the sequence (in this example, of length 1) of input vectors $(1, 1)$ brings the machine to a known state. Thus, $(1, 1)$ is a synchronising sequence. However, this technique is not infallible as it is not always possible to find such a sequence. Reading from a secondary output is possible using a similar technique. Suppose that the secondary output c' has the value 1 in time frame i. If the input $(1, 0)$ is applied to the primary inputs in time frame $i + 1$, the value of the primary output d is 1 in time frame $i + 1$. Analogously, if the value of c' is 0 in time frame i, the value of d is 0 in time frame $i + 1$. Thus, in order to read the secondary output c', it suffices to apply the input $(1, 0)$ to the primary inputs and to observe the primary output d in the following time frame.

Applying synchronising sequences requires simulating one or several time frames until all flip-flops have a known logic value. During those time frames, at least one flip-flop will have a defined but unknown logic value, which must be propagated through the combinational core. In order to make simulation possible, a three-valued algebra can be used. That algebra consists of logic 1, logic 0 and the **logic value X**. X stands for a defined but unknown logic value. 0 and 1 are then called **determinate** values.

This section is finalised with a list of conventions which are observed throughout this work. Figure 2.4 shows how circuits are represented graphically in this thesis and the following list introduces terminology used in the description of algorithms.

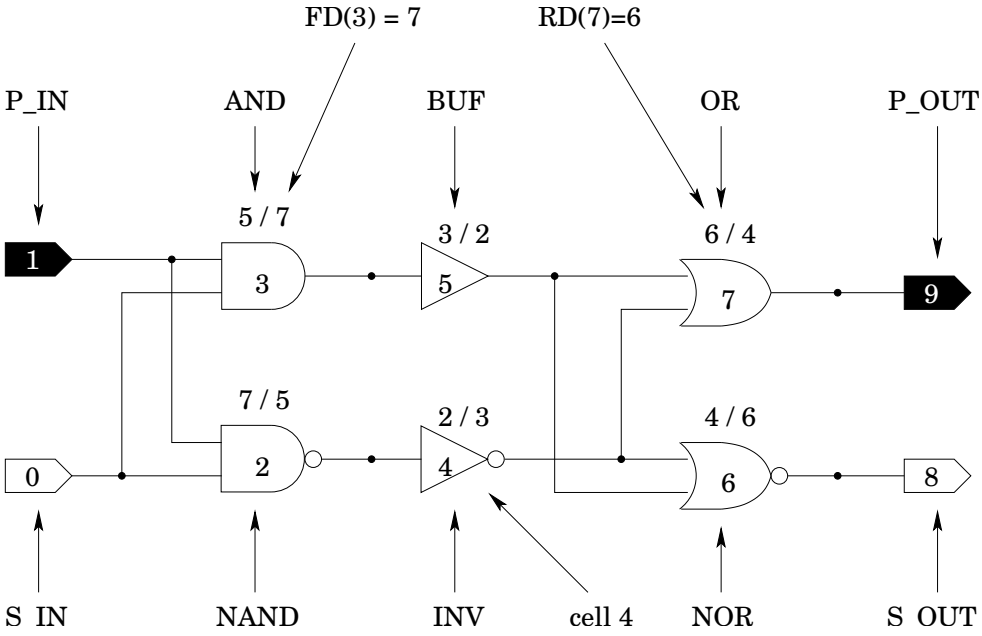


FIGURE 2.4: EXAMPLE CIRCUIT

- The implemented simulator uses a three-valued algebra consisting of the values **`0`**, **`1`** and **`X`**. X stands for a defined but unknown value. Let v be a logic value. Then, $\overline{v}$ denotes the logic inversion of v. $\overline{0} = 1$, $\overline{1} = 0$ and $\overline{\mathrm{X}} = \mathrm{X}$.

- Logic gates and primary and secondary input and output pins are called **`cells`**. Each cell in the circuit is identified uniquely by an integer between 0 and $k - 1$, where k is the number of cells in the circuit. Cells are ordered topologically, i.e. secondary and primary inputs are assigned the identification numbers 0 through $n - 1$, where n is the number of primary and secondary inputs in the circuit. Every other cell is assigned an identification number which is greater than those of its predecessors. That means, whenever it is necessary to perform an algorithm on all cells in the circuit, the algorithm starts with cell 0 and ends with cell $k - 1$. When processing any cell i, its inputs have already been processed.

- Input pins are also simply called inputs, output pins outputs.

- Each cell not being an input or an output is called a **`gate`** and is of

type INV, BUF, AND, OR, NAND, or NOR. A gate of type INV is an inverter, a gate with one input which computes the Boolean negation. A gate of type BUF is a buffer, a gate with one input which computes the identity function. Gates of type AND, NAND, OR or NOR have all at least two inputs. AND gates compute the Boolean conjunction, NAND gates the negation of the Boolean conjunction, OR gates the Boolean disjunction, and NOR gates the negation of the Boolean disjunction. All gates have exactly one output.

All algorithms in this work only distinguish between **`single-input`** and **`multiple-input`** gates. The behaviour of all single-input gates can be described depending on only one parameter called **`inversion`**. If a single-input gate is inverting, it produces the output $\overline{v}$ if its input is v. If the gate is not inverting it produces the output v if its input is v. Analogously, the behaviour of all multiple-input gates depends on three parameters called **`inversion`**, **`controlling value`** (abbreviated $\mathbf{CV}(\boldsymbol{c})$ for a gate c) and **`non-controlling value`** (abbreviated $\mathbf{NCV}(\boldsymbol{c})$). If at least one input of a multiple-input gate c has the logic value $\mathrm{CV}(c)$, then c produces the output value $\mathrm{CV}(c)$ ($\mathrm{NCV}(c)$ if c is inverting), no matter what logic values the other inputs have. In contrast, c can only produce the output $\mathrm{NCV}(c)$ ($\mathrm{CV}(c)$ if c is inverting) if all its inputs have the logic value $\mathrm{NCV}(c)$ at the same time.

	gate type	**inverting**	**cv**	**ncv**
single-input gates	BUF	no	-	-
	INV	yes	-	-
multiple-input gates	AND	no	0	1
	NAND	yes	0	1
	OR	no	1	0
	NOR	yes	1	0

Table 2.1: Types of gates and their parameters

Gates of type XOR are not treated in this work, as the rules explained above do not apply to them. Note that an XOR gate can be replaced

by several gates of the above types, as a XOR $b = \big((\overline{a} \wedge b) \vee (a \wedge \overline{b})\big)$ for all logic values a and b.

- In all graphical representations of circuits, gates are shown together with two numbers separated by a slash. For instance, those two numbers are 5 and 7 for gate 3 in the example. The first is the number of time units the gate needs to switch from 0 to 1 (**`produce a rising transition`**); this number is called **`rising delay time`** of c for a gate c and abbreviated by $\mathbf{RD}(\boldsymbol{c})$. The second is the number of time units the gate needs to switch from 1 to 0 (**`produce a falling transition`**); this number is called **`falling delay time`** of c for a gate c and abbreviated by $\mathbf{FD}(\boldsymbol{c})$.

- Instead of distinguishing between lines and their corresponding logic signals, lines are also simply called **`signals`**. Like cells, signals are identified uniquely by an integer between 0 and $l-1$, where l is the number of signals in the circuit. For all $i = 0, \ldots, l-1$, the identification number of the output signal of cell i is also i. This identification system for signals is well-defined as each signal has exactly one source cell and all gates and all input pins have exactly one output signal.

- Fanout stems are not discerned from fanout branches. A stem and all its branches are represented by only one common signal. (Example: signals 0, 1, 4 and 5 in Figure 2.4.)

- A signal which is the output of a primary input pin is also called a primary input, a signal which is the output of a secondary input pin also a secondary input, a signal which is the input of a primary output pin also a primary output and a signal which is the input of a secondary output pin also a secondary output.

- Let s be a signal and let c be its source cell. Then, the input signals of c are called **`predecessor signals`** of s. For example, signals 4 and 5 are predecessors of signal 7 in Figure 2.4.

- Let s be a signal and let c_1, c_2, ..., c_r be its drain cells. Then, the set containing the output signal of c_1, the output signal of c_2, ... and the output signal of c_r is called set of **`successor signals`** of s. For example, signals 2 and 3 are successors of signal 0 in Figure 2.4.

- Let s_1 and s_2 be signals. A **`path`** from s_1 to s_2 is the sequence of signals starting with s_1 and ending with s_2, where each signal in the

sequence is a successor signal of the previous signal in the sequence. For example, the path from signal 0 to signal 7 in Figure 2.4 is the sequence of signals 0, 2, 4 and 7.

- Let s be a signal. The **output cone** of s is the set containing all signals of every path between s and any primary or secondary output. s is not in its own output cone. For example, the output cone of signal 2 in Figure 2.4 is composed of signals 4, 6 and 7.

2.2 Basic principles of digital testing

2.2.1 Defects, faults and errors

In engineering, models bridge the gap between physical reality and mathematical abstraction. They are essential in the areas of design and test as they allow the development and application of analytical tools.

When testing digital ICs, it is necessary to understand the difference between **defects**, **faults** and **errors**. A defect is the unintended difference between the implemented hardware and its intended design. Note that not design mistakes are meant, but defects which come into existence during the manufacturing process. Some typical defects in VLSI chips are missing or broken contacts, shorts, contacts with too low conductance, surface impurities, etc. A fault is a formal representation of the defect. The wrong response of a defective system is an error. For example, Figure 2.5 pictures an AND gate whose input b is shorted to ground. This is the defect of the system. It can be represented by the fault b stuck-at-0, which means that signal b always has the logic value 0, independently of what value is produced by b's source cell. An error occurs if the input $(1, 1)$ is applied to the gate. Then, the gate produces the erroneous output value 0 instead of the expected 1.

How a manufacturing defect is represented by a fault is dictated by a so-called **fault model**. A fault model comprises a set of rules which specify how many faults can occur in the circuit and how these faults are defined regarding their occurrence site and the fault effect they induce. There is a wide variety of fault models created to describe the various effects that a circuit C can have. Among the list of possible effects of manufacturing defects are the following [29], [16], [34]:

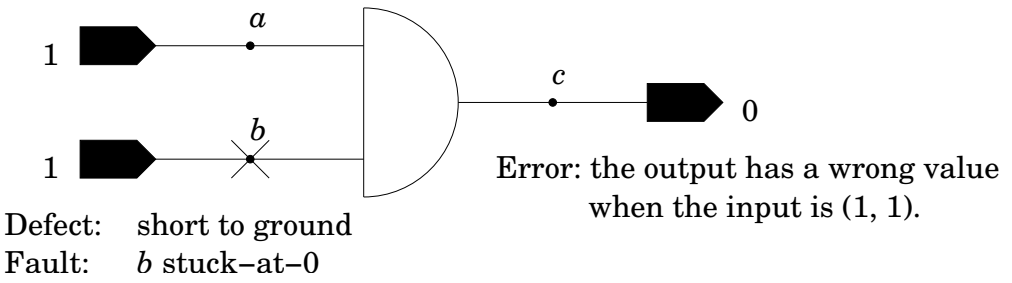


FIGURE 2.5: DEFECTS, FAULTS AND ERRORS

- The Boolean function g_C computed by C can be altered.
- The function computed by the circuit may become non-Boolean, i.e. at some output the circuit produces a voltage which cannot be clearly interpreted as logic 0 or logic 1.
- Some lines in the circuit can show a memory behaviour thus making a combinational circuit sequential.
- The timing of the information processing by the circuit can be affected.

The most important property of fault models is that they reduce the complexity of the problem to handle. While there are infinitely many possible defects (e.g. particle-induced defects cannot be listed completely as there are infinitely many possible particle shapes and the exact location on-chip of the particle is a continuous parameter), there are only finitely many faults in most fault models used in practice.

Currently, the most popular fault model is the **(single) stuck-at fault model** [10], [14], [34, p. 16], [21]. For each line s in the circuit, two stuck-at faults are defined: the s stuck-at-0 fault, meaning that line s always has the value 0; and the s stuck-at-1 fault, meaning that line s always has the value 1, independently of what value is produced by its source cell. Under this model, a fault evidently only affects the Boolean function the circuit computes. Since this fault model assumes that only one line in the circuit can be affected at the same time, the number of possible faults is linear in the number of lines of the circuit. Nevertheless, test process using the stuck-at fault model detects a large amount of defects which are usually modelled by more complex fault models (also fault models which allow more than one fault to occur simultaneously). Consider, for example, the circuit of

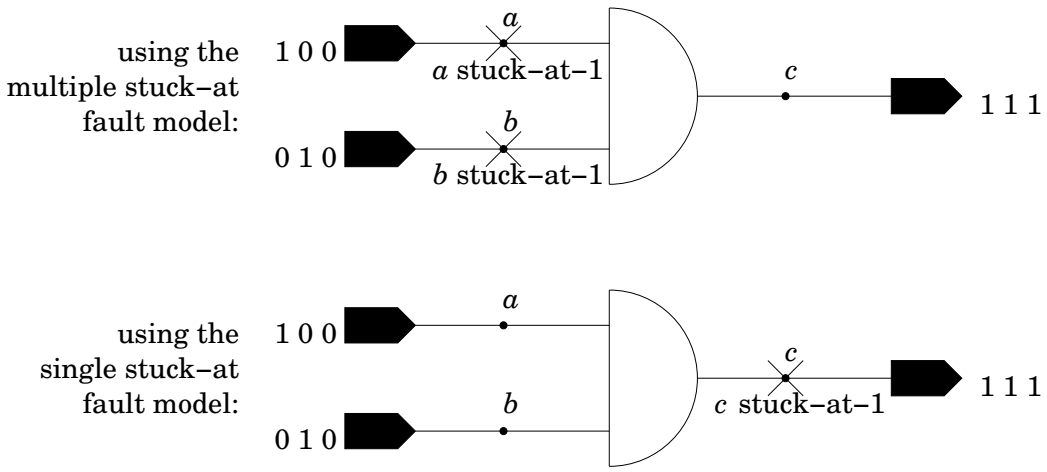


FIGURE 2.6: SINGLE AND MULTIPLE STUCK-AT FAULT MODELS

Figure 2.6. The multiple stuck-at fault model allows several stuck-at faults to occur at the same time. The upper part of the figure shows the circuit under the influence of two stuck-at faults ($f_1 : a$ stuck-at-1 and $f_2 : b$ stuck-at-1). These cause the following errors: when applying one of the input vectors $(1, 0)$, $(0, 1)$ or $(0, 0)$, the circuit produces a logic 1 instead of a logic 0. The lower part of the figure shows the circuit under the influence of only one stuck-at fault ($f_3 : c$ stuck-at-1). f_3 causes alone exactly the same errors as f_1 and f_2 do together. Thus, the combination of f_1 and f_2 can be detected using the same strategy as used to detect f_3 under the single stuck-at fault model.

2.2.2 Test application

Let C be a combinational circuit with n inputs and m outputs. The main concept of test application is best explained when considering a fault model which affects only the Boolean function $g_C : \mathbb{B}^n \to \mathbb{B}^m$ the circuit C computes (like the stuck-at fault model).

A set of input vectors or input patterns $P := \{p_1, p_2, \ldots, p_{r_P}\} \subseteq \mathbb{B}^n$ is called a **`test set`**; r_P is its **`size`**.

A set of faults $F := \{f_1, f_2, \ldots, f_{r_F}\}$ is called a **`fault list`**; r_F is its **`size`**. The list of all faults which need to be detected during the test process is called **`list of target faults`**.

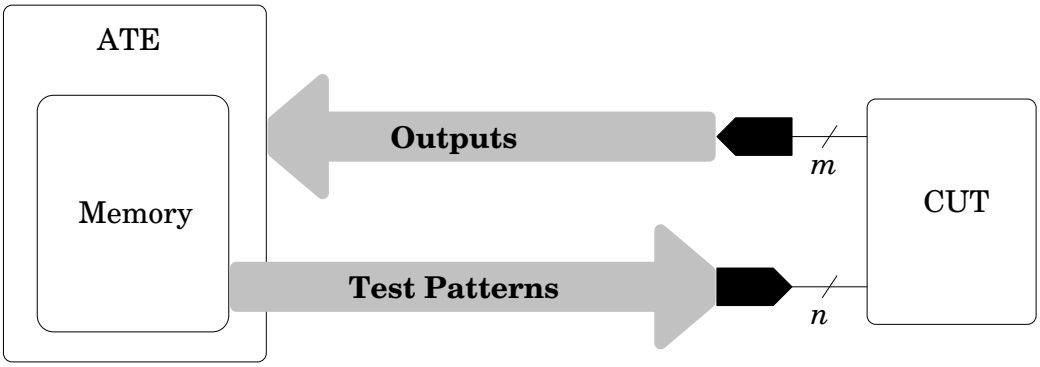


FIGURE 2.7: TEST APPLICATION

Let us consider a fault $f_i \in F$ and an input vector $p_j \in P$. If f_i is present, the Boolean function that C computes is altered. That means, C no longer computes g_C but a new Boolean function $g_C^{f_i}$. If $g_C(p_j) \neq g_C^{f_i}(p_j)$, p_j is said to **detect** f_i. Then, p_j is called a **test** for f_i. That means, by observing the circuit's response to the application of the input p_j, it is possible to determine whether fault f_i is present or not. For example, in Figure 2.6, the input vectors $(1,0)$, $(0,1)$ and $(0,0)$ are all tests for the fault c stuck-at 1 as they all produce the value 0 in the fault-free case and the value 1 in the faulty-circuit case. The test set P is said to detect f_i if at least one $p \in P$ detects f_i.

Once an appropriate test set has been generated (see Section 2.2.5), the testing process is simple to perform (cf. Figure 2.7). The **automatic test equipment** (ATE) has a memory module in which the test set and the responses of the fault-free circuit (obtained by simulation, see Section 2.2.4) are stored prior to test start. Then, each test set is applied to each actual manufactured circuit (**circuit under test**, CUT) which sends the generated output back to the ATE. The ATE compares the generated output with the expected fault-free output. If these two outputs differ, the circuit fails the test.

2.2.3 Fault coverage

The **fault coverage** is a measure to grade the quality of a test. In its most general form, it is defined as

$$\text{fault coverage of a test set } P = \frac{\text{number of faults } P \text{ detects}}{\text{size of target fault list}} \cdot 100\%.$$

There are many other possibilities to compute fault coverage. Depending on the used fault model, it may be necessary to consider several other factors when defining the detection metric, e.g. the probability for a fault to occur or the probability for a fault to be detected.

2.2.4 Simulation

2.2.4.1 Logic simulation

Logic simulation is the process of determining the steady-state logic values implied at each circuit line by the application of an input vector to the circuit.

In its simplest form, the zero-delay logic simulation of a vector $p := (v_1, v_2, \ldots, v_n) \in \mathbb{B}^n$ on a combinational logic block (combinational circuit or core of a sequential circuit) assigns each component v_i of the vector to the corresponding primary input and computes the new logic value implied at each line, where the lines are processed in topological order. These three steps are repeated for each vector $p \in P$.

There are several techniques to accelerate logic simulation. One such technique is the **event-driven** simulation. For example, when having to simulate two test vectors $p_1 := (0, 0, 0, 0, 0)$ and $p_2 := (1, 0, 0, 0, 0)$, the first vector must be simulated as usual. But when simulating p_2 it is not necessary to recompute the logic value of every line in the circuit. It is enough to recompute the values of those lines which are in the output cone of the first primary input, as that is the only input whose logic value is modified. Event-driven simulation is performed by assigning each primary input its new value as dictated by p_2. If its new value differs from its old value, its successor signals are inserted into a priority queue which orders the inserted signals topologically. Then, it is enough to recompute the logic values of those signals which are in the queue. Again, if the value of a signal s taken from the queue changes, s's successors have to be inserted into the queue. These steps are repeated until the queue is empty.

2.2.4.2 Fault simulation

Fault simulation is the process of determining the set of faults which are detected by applying a test set to a circuit. Fault simulation is performed to compute the fault coverage a test set achieves and as part of the test pattern generation process.

The basic fault simulation algorithm is depicted in Figure 2.9. In that pseudo-code, C^f stands for the faulty version of the circuit which is affected by fault f. $v(s)$ stands for the logic value a signal s has after performing the fault-free simulation. $v^f(s)$ stands for the logic value a signal s has after performing the faulty-circuit simulation on C^f. Obviously, this simple algorithm can also be applied when fault models other than the stuck-at fault model are used. Only the implementation of the fault-free simulation (line 9) and of the faulty-circuit simulation (line 11) algorithms depends directly on the used fault model.

Fault-free simulation of a test pattern p is the process of determining the steady state of a fault-free circuit after applying p to its inputs. In the case of the stuck-at model, fault-free simulation is equivalent with logic simulation. For other fault-models, a fault-free simulation may be very complex. Fault-free simulation is performed to compute the fault-free responses that have to be stored in the ATE memory.

Faulty-circuit simulation of a test pattern p is the process of determining the steady state of a circuit affected by a certain fault f after applying p to its inputs. Usually, faulty-circuit simulation is performed like fault-free simulation. Only processing the fault site is slightly different. Its logic value is not computed as that of all other lines; it is derived from the fault description. Consider, for example, the circuit in Figure 2.8. The fault-free simulation of $(0, 1)$ is as follows. First, 0 is assigned to the primary input a

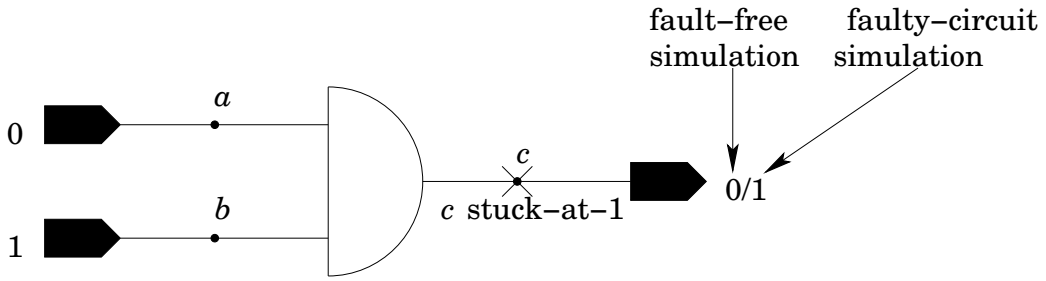


Figure 2.8: Example on fault simulation

and 1 is assigned to the primary input b. Then, the value of c is computed as value-of-a AND value-of-$b = 1$ AND $0 = 0$. In the faulty-circuit simulation, instead of reading the values of a and b, the logic 1 is directly assigned to signal c, as the fault c stuck-at-1 is present.

```
 1 SIMPLE_FAULT_SIMULATION
 2     Input:  a circuit C
 3             a list of target faults F
 4             a test set P
 5     Output: a list of detected faults F'
 6 BEGIN
 7     let F' be an empty list of faults
 8     for each test pattern p ∈ P ; do
 9         perform fault-free simulation of p
           and record the fault-free output
           v(s) for each signal s
10         for each fault f ∈ F ; do
11             perform faulty-circuit simulation
               of p on C^f
12             if v(z) ≠ v^f(z) for any primary output z ; then
13                 move f from F to F'
14             fi
15         done
16     done
17     return F'
18 END
```

FIGURE 2.9: ALGORITHM: SIMPLE FAULT SIMULATION

2.2.5 (Automatic) test pattern generation

Although test pattern generation is not the topic of this work, this section has been included in order to provide a more complete overview of the area of digital testing. The second aim of this section is making clear that fault simulation plays an important role during the TPG process.

Applying all possible 2^n input vectors to a circuit of n inputs would detect all existing defects which affect its static behaviour, without the necessity of using a fault model. However, this brute-force method is not feasible, as modern circuits have thousands of inputs. Even without taking the application feasibility into account, less test patterns mean a lower test application cost (time and tester memory costs). Hence, the process of finding a test set P with as less test patterns as possible that detects all faults in a target fault list F, or which at least achieves a high fault coverage, is essential. The term **(automatic) test pattern generation**, abbreviated (A)TPG, denotes this process.

Typically, the overall structure of a test pattern generation procedure consists of several phases [3, Chapter 3, p. 3.3/2]:

1) Low-cost, fault-independent test generation.

2) Identification of undetectable faults.

3) High-cost, deterministic, fault-oriented test generation.

4) Static test compaction.

In Phase 1, random test patterns are generated and simulated until the achievable fault coverage cannot be improved any more by adding more random patterns to the test set. In Phase 2, so-called redundant or undetectable faults are removed from the fault list. Redundant faults are those which do not affect the functionality of the circuit and remain thus undetected by every input vector. This topic is not relevant to this thesis and will not be explained any further. A standard text book like [21] or [1] can be consulted for more information on redundant faults, especially on how to handle their presence. In Phase 3, test patterns are generated, which detect the faults that remain undetected after the application of the test patterns generated in Phase 1. In Phase 4, some test patterns are removed from the test list in order to lower test application costs. However, test patterns may be only removed such that the achieved fault coverage does not fall.

```
 1 DETERMINISTIC_TPG_OVERALL_PROCEDURE
 2     Input:  a circuit C
 3             a list of target faults F
 4     Output: a test set P
 5             a list of undetectable faults F'
 6 BEGIN
 7     let P be an empty test set
 8     repeat
 9         choose any fault f ∈ F
10         try to generate a test pattern p that detects f
11         if p can be generated ; then
12             add p to P
13             perform fault-free simulation of p on C
14             for each fault f' ∈ F ; do
15                 perform faulty-circuit simulation
                   of p on C^f'
16                 if p detects f' ; then
17                     remove f' from F
18                 fi
19             done
20         else
21             move f from F to F'
22         fi
23     until F is empty
24     return P and F'
25 END
```

FIGURE 2.10: ALGORITHM: DETERMINISTIC TPG, OVERALL PROCEDURE

Phase 3 is performed using the algorithm in Figure 2.10. One chooses a target fault f and generates a test pattern p that detects it (line 10). The exact algorithm for the deterministic test generation procedure of line 10 depends on the used fault model. Since Phase 2 may fail to recognise some undetectable faults[1], it is possible that there is no test for the chosen fault f. If the deterministic test generation procedure (line 10) cannot find a test for f, f is marked as undetectable (line 21). If a test p is found, all faults which p detects are removed from the list, not only f (lines 14-19). This is done such that the test set's size remains low.

2.3 Delay fault modelling

In Section 2.2.1, some effects of manufacturing defects were listed. The last point in the list stated that some defects affected the timing of the information processing by the circuit. Defects that do not affect the logical behaviour of the circuit, but its timing, cannot be modelled using the stuck-at fault model. Instead, so-called **`delay fault`** models are used [21].

The most important delay fault models are the **`gate delay`** fault model [6], [35], the **`path delay`** fault model [44] and the **`segment delay`** fault model [18].

Gate delay faults: A circuit is said to have a (single) gate delay fault (GDF) in some gate, if the input or output of the gate is slow to propagate $0 \rightarrow 1$ (**`rising`**) or $1 \rightarrow 0$ (**`falling`**) transitions.

Path delay faults: A circuit is said to have a path delay fault (PDF), if the circuit has a path from a primary input s_i to a primary output s_o which is slow to propagate rising or falling transitions from s_i to s_o.

Segment delay faults: A circuit is said to have a segment delay fault (SDF), if the circuit has a path of a given length L (or shorter) from a signal s_1 to a signal s_2 which is slow to propagate rising or falling transitions from s_1 to s_2.

While the GDF model is very simple and works with the unrealistic assumption that only one site is fully affected while others are not affected at

[1]The algorithms used in Phase 2 are meant to be fast and low-cost, but are usually not complete. The problem of identifying redundant faults is co-NP-complete.

all, the PDF model is the more general of both, as it models the cumulative effect of the delay variations of the gates and lines along the path. However, since the number of paths in the circuit can be very large, this model may require much more time for test generation and application than the GDF model. The compromise between both extremes is the SDF model, which works with "short" paths. The parameter L can be chosen based on the available statistics about the types of manufacturing defects. The SDF model is more flexible than the GDF model but there are not as many faults as under the PDF model. A comparative study on delay fault models is published in [28].

Relevant for this thesis is the **line delay** fault model [27], which can be seen as a simplified version of the GDF model. A circuit is said to have a **(single) line delay fault** (LDF) if there is a line in the circuit which is slow to propagate rising or falling transitions, delaying them by an additional unknown, but fixed amount of time. That amount of time is always denoted by $\boldsymbol{\delta}$ in this thesis and is called the **size of the fault**. A fault is described by the fault site (the affected line) and by whether rising or falling transitions are affected. Faults affecting rising transitions are called **slow-to-rise** LDFs, those affecting falling transitions are called **slow-to-fall** LDFs. A slow-to-rise fault affecting the line or signal with identification number i is denoted in this thesis by $\boldsymbol{i}$**R**. A slow-to-fall fault affecting signal i is denoted by $\boldsymbol{i}$**F**.

2.3.1 Test application

Testing for delay faults requires the application of two successive test patterns (**a test pair**) to the CUT. The first pattern (**initialisation vector**) brings the CUT into a known and stable state. The second test pattern (**propagation vector**) excites the fault and propagates the fault effect to an output by inducing a rising or falling transition at one or more inputs of the CUT. If the circuit is correct, its state changes meeting the same timing constraints as the fault-free circuit. This testing method is known as **two-pattern testing**. Detailed information on two-pattern testing can be found in [4] and literature cited there.

For a slow-to-rise LDF f, exciting f means inducing at the fault site a rising transition that can be delayed if f is present. A slow-to-fall LDF is excited if a falling transition is induced at the fault site.

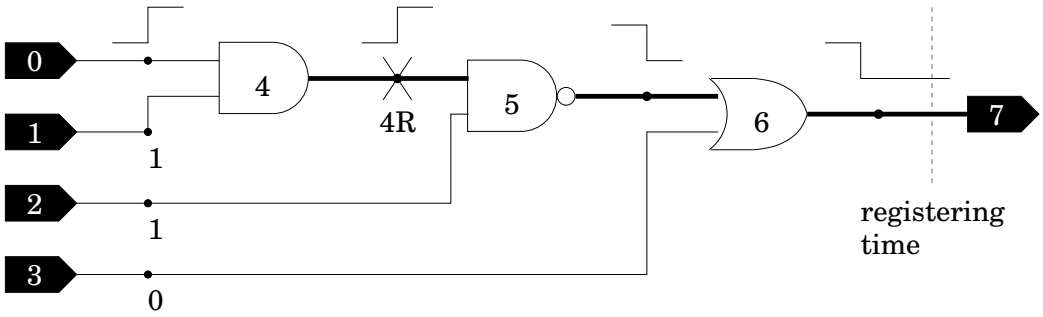


FIGURE 2.11: DETECTION OF AN LDF

Given a test pattern pair p, the initialisation vector is denoted by $\boldsymbol{p^1}$ and the propagation vector by $\boldsymbol{p^2}$.

A test pair p that detects an LDF iR (iF) must have the following properties:

- p^1 must induce the logic value 0 (1) at signal i.
- p^2 must induce the logic value 1 (0) at signal i thus launching a rising (falling) transition at the fault site.
- Both p^1 and p^2 must sensitise a path from the fault site to a primary output of the circuit, thus allowing the fault effect to become visible at an output.

A path is said to be sensitised if all its lines are sensible to the fault, i.e. if the behaviour of all its signals is different in the fault-free and the faulty-circuit case. Consider, for example, the circuit in Figure 2.11. The test pair 0110/1110 makes the path composed of bold lines sensible to fault 4R. When 0110/1110 is applied, signal 2 remains stable at logic 1 during the application of both vectors. As 1 is the non-controlling value of gate 5, the behaviour of gate 5's output depends exclusively on the behaviour of the fault site, signal 4. Analogously, signal 3 remains stable at logic 0, the non-controlling value of gate 6. Thus, the behaviour of the circuit's primary output depends exclusively on the behaviour of signal 5. Altogether, if the fault 4R is present, the induced rising transition at signal 4 is delayed. This delays signal 5's falling transition, which delays signal 6's falling transition. If the delay is large enough, the expected logic 0 will arrive at the output after the registering time, at which the wrong value 1 is read, thus proving the circuit to be faulty.

The "classical method" to apply a test pair p is the following: First, p^1 is applied to the CUT. All internal signals are allowed to stabilise (which is achieved in practice by holding the inputs constant for several cycles). Then, p^2 is applied to the inputs. In the next clock cycle the outputs are evaluated. If the value at any output does not correspond to the expected value, then the propagation must have been too slow and the CUT is proven faulty.

Delay fault tests must obviously be applied at a device's nominal speed (**`at-speed testing`**). In contrast to the classical test method, it is possible to apply all test pairs just one after another, without waiting for the circuit's signals to stabilise. An interesting empirical result is reported in [31]. By performing experiments with a sample of manufactured ICs, the authors found that omitting the waiting actually reduces the number of defective chips passing the test.

2.3.2 Application of two-pattern testing to sequential circuits

Most testing concepts remain largely unchanged when considering sequential instead of combinational circuits. Alas, that is not the case of two-pattern testing. When working with sequential circuits, two-pattern testing becomes considerably more difficult than when applying it to combinational circuits.

In the most general case, the secondary inputs and secondary outputs of a sequential circuit are not accessible from outside of the circuit. This circumstance may forbid the application of necessary values (values which are required to sensitise the path or to induce the proper transition at the fault site) to the secondary inputs; and, if the fault effect can only be propagated to a secondary output, the fault effect becomes unobservable.

It may be possible to write a necessary value into a memory element by applying the synchronising sequences technique (page 22). However, this technique does not always work. If the combinational core's structure is too complex, such a sequence may not exist at all. For circuits with a large number of secondary inputs, this writing approach becomes useless.

A better approach is to use a standard method for reducing testing problems for sequential circuits to ones for combinational circuits, namely **`scan design`**. See, for example, [34, Section 2.1.4] for a concise introduction; or [1], [21] for detailed information on this topic.

Figure 2.12 illustrates the main idea of scan design. The combinational core of a sequential circuit with 3 flip-flops is shown. There are two addi-

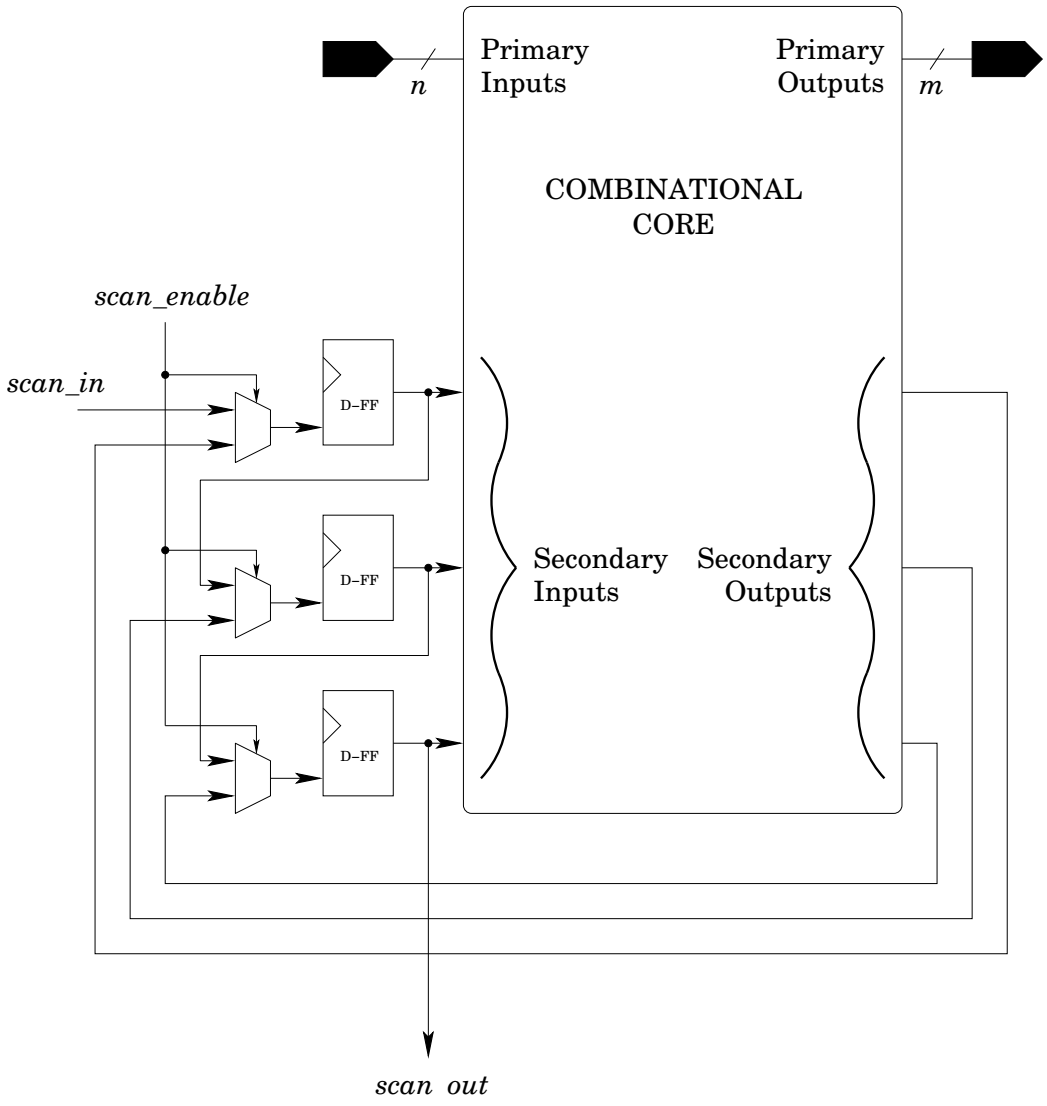


FIGURE 2.12: SCAN DESIGN

tional inputs, *scan_in* and *scan_enable*; and one additional output, *scan_out*. *scan_enable* controls additional multiplexers between the flip-flops. When *scan_enable* is inactive, the multiplexers are switched to let through the values from the core's secondary outputs, and the circuit is in its normal operation mode. When *scan_enable* is activated, the flip-flops are connected to form a chain called the **`scan chain`**. By applying values to the *scan_in* input, arbitrary values can be shifted into the flip-flops, while their content can be shifted out over the *scan_out* output.

When all flip-flops are part of the scan chain (**`full scan`**), applying arbitrary initialisation vectors is possible. However, scan does not solve the second problem that arises when applying two-pattern testing to sequential circuits. Since the two vectors of a test pair have to be applied in consecutive clock cycles, it is not possible to apply arbitrary propagation vectors, as the shifting of all values for the secondary inputs requires as much clock cycles as there are flip-flops.

Applying the test pair according to the "classical" test application method introduced in Section 2.3.1, which allows the effects of the application of the initialisation vector to stabilise over several clock cycles, does also not solve this problem. All the input values specified by the propagation vector must be applied simultaneously to all primary and secondary inputs of the CUT.

Hence, two-pattern testing can only be applied to a full-scan-circuit if the propagation vector is such that the values that it specifies for the secondary inputs are obtained through the functional path as response to the application of the initialisation vector.

Furthermore, each test pair can only be applied independently of all others in the test set, due to the additional clock cycles needed to scan in the initialisation vector and the additional clock cycles needed to scan out the secondary output values after the application of the propagation vector.

There are enhanced scan techniques that allow storing arbitrary values into several flip-flops simultaneously. However, this benefit is accompanied of a high hardware overhead.

2.4 Fault coverage under resistive fault models

In almost any semiconductor manufacturing technology, conducting wires connected in an unintended way are a prominent class of defects. Bridging fault models have been created to model this type of defects. In its more general form, this model assumes that two lines are bridged, e.g. due to a conducting contamination touching both lines, thus creating an unintended connection between them. If both lines conduct the same logic value (i.e. the same voltage level) there is no malfunction. However, if opposite logic values are present at the bridged lines, faulty behaviour may arise. Then, one or both bridged lines are affected. Depending on the assumptions made, the faulty behaviour can be described using a variety of different bridging fault models. Among these are the **resistive bridging fault** models, under which the bridge has a resistance. More detailed information on this topic can be found in [11], [12], [13] and [34, Section 2.3.4]. In this section, some definitions taken from that literature are introduced. These definitions will be needed in Chapter 6.

Working with resistive fault models requires taking into account the resistance, which is an unknown and unpredictable parameter [41]. A bridging fault may be detected by a test vector for one resistance, while the bridge between the same affected nodes may not be detected by the same vector if the bridge has a different resistance. For example, the resistance may be so high, that the fault cannot be excited. If the bridge conductance is too low, the bridged signals cannot affect each other.

This ambiguity changes the concept of fault coverage in fundamental manner. It is not longer possible to speak of a detected or an undetected fault. It is necessary to find a way of measuring the probability for the fault's detection, depending on the probability that the bridge has a certain resistance.

Renovell et al. [40], [38], [39] introduced the concept of an **Analogue Detectability Interval** (ADI). The simulation yields for each fault and each output a resistance range $[r_1; r_2]$, the ADI, such that the short modelled by the fault is detected by the test set if and only if its resistance r meets the condition $r_1 \leq r \leq r_2$. Typically, r_1 is 0 Ω, but this does not need to be the case. For circuits having reconvergencies and sequential circuits, the ADI may be the union of multiple disjoint intervals $[r_{1,1}; r_{1,2}] \cup [r_{2,1}; r_{2,2}] \cup \cdots \cup [r_{N,1}; r_{N,2}]$ [40].

Let the CUT have m outputs, and let f be a fault. For $i = 1, 2, \ldots, m$ and a test pattern p, $\boldsymbol{ADI_i(p)}$ denotes the ADI propagated to the i-th output by simulating p. Given a test set P, the **C-ADI** of f is defined as

$$\text{C-ADI}\,(f) := \bigcup_{p \in P} \bigcup_{i=1}^{m} ADI_i(p).$$

The C in C-ADI stands for "covered by the test set". The C-ADI of a fault is the union of all ranges of resistances for which the fault is detected by P.

There are several fault coverage definitions basing on C-ADI. In the original literature they are just called fault coverage. Here, the same notation and terminology as in [34, Section 2.3.4] is used.

Let $\rho(r)$ be the probability density function of the short resistance r. In [39] the Normal distribution is suggested to describe $\rho(r)$. The **pessimistic fault coverage** (P-FC) introduced in [39] is defined for one fault f as

$$\text{P-FC}\,(f) := \frac{\int_{\text{C-ADI}(f)} \rho(r)dr}{\int_{0}^{+\infty} \rho(r)dr} \cdot 100\%.$$

This definition relates the "fraction" of the ranges in which the fault is detected to the complete range from 0 to $+\infty$, weighted by ρ. For a fault list F, the average fault coverage is defined as

$$\text{P-FC}\,(F) := \frac{\sum_{f \in F} \text{P-FC}\,(f)}{|F|}$$

In [40], a second definition is proposed. **G-ADI** is defined as C-ADI of an exhaustive test set. G stands for "global". The corresponding fault-coverage definition is

$$\text{G-FC}\,(f) := \frac{\int_{\text{C-ADI}(f)} \rho(r)dr}{\int_{\text{G-ADI}(f)} \rho(r)dr} \cdot 100\%.$$

This definition can be considered to be exact, but an exhaustive test set consists of 2^n vectors for circuits with n inputs. Thus, G-ADI can only be measured for circuits with relatively few inputs.

The third fault coverage definition was introduced by Walker in [23]. **E-FC** (E means "excitation") is defined as

$$\text{E-FC}\,(f) := \frac{\int\limits_{\text{C-ADI}(f)} \rho(r)dr}{\int\limits_{0}^{R_{\max}} \rho(r)dr} \cdot 100\%,$$

where $R_{\max}$ is the maximum size the resistance may have as to excite the fault.

A fourth fault coverage definition was introduced in [12]. **O-FC** (O stands for "optimistic") is defined as

$$\text{O-FC}\,(f) := \begin{cases} 0\% & \text{if C-ADI}\,(f) = \emptyset \\ 100\% & \text{else} \end{cases}.$$

Under O-FC, it is enough that a fault is detected for any resistance in order to regard the fault as detected.

The following relationship is shown in [12]:

$$\text{P-FC}\,(f) \leq \text{E-FC}\,(f) \leq \text{G-FC}\,(f) \leq \text{O-FC}\,(f)\,.$$

For a fault list F, the average fault coverages G-FC (F), E-FC (F) and O-FC (F) are defined analogously to P-FC (F).

3

Simulating Dynamic Effects of Resistive Opens

The aim of the work presented in this thesis is designing and implementing a simulator for resistive opens. These opens are also called weak and have a resistance of less than 10 MΩ [43]. Resistive opens are defects which manifest themselves as lines with an elevated resistance or as resistive vias and contacts. They still let the circuit work, but increase the delay of paths going through the fault site [25], [29]. In this work, resistive opens are modelled using the single line delay fault model. In [36] they are modelled as gate delay faults. Modelling them as line delay faults instead, does not constitute a large modification. This just allows the fault to occur at the circuit's inputs, which are not fed by gates.

When modelling a resistive open with a resistance r by an LDF, the fault's size δ depends on r. r is an unpredictable random parameter [41]. A simulator is to be designed and implemented that accepts the following arguments:

- a combinational or sequential circuit C (gate-level net-list and rising and falling delay times of each gate type) with n primary inputs and m primary outputs;
- a list F of LDFs, from now on called **`the fault list`**;
- and a set P of test pairs, from now on called **`the test set`**;

The simulator shall accomplish the following tasks:

- perform fault simulation and compute C-ADI (f) for each fault f in F;

- compute the fault coverage achieved by the application of P to C.

Let $f \in F$ be an LDF modelling an open with resistance r. Analogously to the definition in Section 2.4, C-ADI (f) is a set of possibly **disjoint** ranges of resistances. It has the form

$$\text{C-ADI}\,(f) := \big\{[r_{1,1}; r_{1,2}], [r_{2,1}; r_{2,2}], \ldots, [r_{N,1}; r_{N,2}]\big\},$$

where $r_{i,1} \in \mathbb{N}$ and $r_{i,2} \in \{r_{i,1}, r_{i,1}+1, \ldots\} \cup \{+\infty\}$ for all $i = 1, 2, \ldots, N$; and the property that P detects f if and only if $r \in [r_{i,1}; r_{i,2}]$ for some $i = 1, 2, \ldots, N$. The ranges in C-ADI (f) are called **detection resistance ranges** of f.

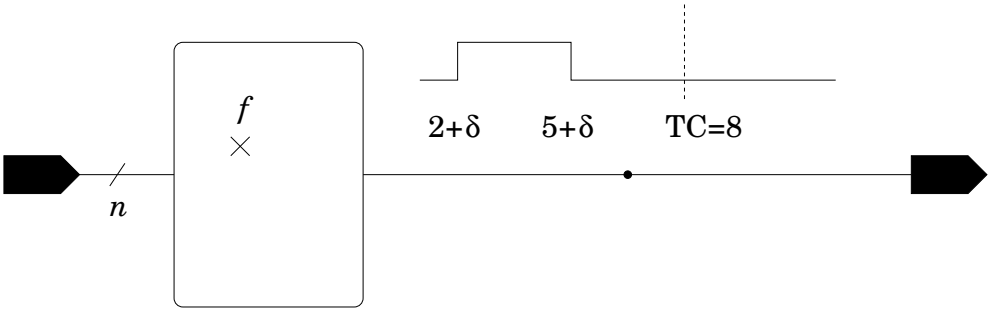


FIGURE 3.1: DETECTION INTERVAL OF AN LDF

What is the purpose of computing C-ADI (f) of a fault f? Let us consider the application of a test pair p on the example circuit shown in Figure 3.1. In the fault-free case, the circuit's output signal is stable at logic 0 until time 2. The application of p[2] induces a 1-pulse between time 2 and time 5. After time 5, the output remains stable at logic 0. The clock sampling time TC is 8. Let a fault f of size δ cause the 1-pulse at the circuit's output to begin at time $2+\delta$ and to end at time $5+\delta$. p detects f only if the presence of f causes the circuit's output to have the logic value 1 (the wrong output value) at time TC, which is the time the circuit's output values are registered at.[1] That means, p detects f if and only if $2+\delta \leq \text{TC} < 5+\delta$, i.e. if $3 < \delta \leq 6$. δ will meet this constraint only if the modelled open's resistance r is in a

[1]Usually, a combinational circuit does not stand alone. Several combinational blocks are part of a system and synchronised using a universal clock. For the purpose of communication among the combinational blocks, their outputs may be stored into register banks which are writable at times TC, $2 \cdot TC$, $3 \cdot TC$, etc.

certain range $[r_1; r_2]$. As was already said, r is an unpredictable parameter, so it is not guarantee-able that r is in $[r_1; r_2]$ for every actual manufactured circuit that has the open. The open will remain undetected by p if not. In order to improve fault coverage, it may be necessary to find additional test pairs which detect the open if its resistance is in ranges other than $[r_1; r_2]$.

Thus, simulating the test set and computing the C-ADI of faults is essential during TPG as an indicator of whether more test pairs have to be generated to achieve an acceptable fault coverage.

Throughout this thesis, our simulator is called **`RO-simulator`** (stands for "resistive-opens-simulator").

3.1 Overall structure of the RO-simulator

The RO-simulator works in three phases:

1) Delay fault simulation is performed and for each fault f in F a so-called **`detection set of delay size intervals`**, abbreviated $\mathrm{DS}_{\boldsymbol{del}}\,(\boldsymbol{f})$, is computed. It has the form

$$\mathrm{DS}_{del}\,(f) := \big\{[a_{1,1}; a_{1,2}], [a_{2,1}; a_{2,2}], \ldots, [a_{M,1}; a_{M,2}]\big\},$$

where $a_{i,1} \in \mathbb{N}$ and $a_{i,2} \in \{a_{i,1}, a_{i,1} + 1, \ldots\} \cup \{+\infty\}$ for all $i = 1, 2, \ldots, M$; and the property that P detects f if and only if f's size δ is in $[a_{i,1}; a_{i,2}]$ for some $i = 1, 2, \ldots, M$. The intervals in $\mathrm{DS}_{del}\,(f)$ are called **`detection delay size intervals`** of f.

2) For each fault f in F, C-ADI (f) is computed based on $\mathrm{DS}_{del}\,(f)$.

3) The overall fault coverage that P achieves is computed based on C-ADI (f) of all faults $f \in F$.

Figure 3.2 shows how the simulator works. In that pseudo-code, C^f stands for the faulty version of the circuit which is affected by fault f.

Lines 10 through 18 correspond to Phase 1 of the RO-simulator. Phase 1's structure is exactly that of a "classical" fault simulation algorithm (cf. Figure 2.9 on page 32). Here, instead of just marking the faults as detected, their detection delay size intervals are computed (line 15) since, according to the

```
1  SIMULATION_OF_DYNAMIC_EFFECTS_OF_RESISTIVE_OPENS
2      Input:  a combinational circuit C
3              a fault list F
4              a test set P
5      Output: an array RES holding C-ADI(f) for each f ∈ F
6              the achieved fault coverage FC
7  BEGIN
8      let RES be the array that will
       hold C-ADI(f) for each f ∈ F.
9      let DEL be the array that will
       hold DS_del(f) for each f ∈ F.
10     for each test pair p ∈ P ; do
11         perform fault-free simulation of p
12         for each fault f ∈ F ; do
13             if f is excited by p ; then
14                 perform faulty-circuit simulation
                   of p on C^f
15                 determine detection delay size intervals
                   of f under p and add them to f's
                   detection set of delay size
                   intervals DEL[f]
16             fi
17         done
18     done
19     for each fault f ∈ F ; do
20         convert DEL[f] into RES[f]
21     done
22     compute FC out of RES
23     return RES and FC
24 END
```

FIGURE 3.2: ALGORITHM: OVERALL PROCEEDING OF THE RO-SIMULATOR

fault model, a fault is detected only with a certain probability depending on its size.

An example on how the fault-free simulation (line 11) works is in Section 5.1.1, a detailed description in Section 5.2.1. An example on how the faulty-circuit simulation (line 14) works for a test pattern p and a fault f, is in Section 5.1.2, a detailed description in Section 5.2.2. An example on how the computation of a set of detection intervals (line 15) works for a test pattern p and a fault f, is in Section 5.1.3, a detailed description in Section 5.2.3. The obtained detection intervals tell what size f must have such that p detects it. The final set of detection intervals for f is the union of the sets obtained for each test pair.

A short remark must be made on line 13 in Figure 3.2, although this has been mentioned in the preceding chapter. A fault iF is excited by a test pair $p := (p^1, p^2)$ if the application of p^1 causes signal i to stabilise at logic 1 while the application of p^2 causes signal i to stabilise at logic 0. A fault iR is excited if the application of p^1 causes signal i to stabilise at logic 0 while the application of p^2 causes signal i to stabilise at logic 1.

Finally, it is necessary to say that the RO-simulator's Phase 1 is based on the delay fault simulation method (**`PR-simulator`**) in [36]. However, techniques and algorithms of the PR-simulator were modified or extended. These modifications will be explained later. If those modifications have been introduced by other authors in the past, we are not aware of that. Chapters 4 and 5 deal in detail with the implementation of Phase 1 of the RO-simulator.

In Phase 2 (lines 19 through 21 in Figure 3.2), C-ADI (f) is computed out of $\mathrm{DS}_{del}(f)$. This is done for each fault f independently of all other faults.

Given a fault f and its detection set of delay size intervals, $\mathrm{DS}_{del}(f) := \{[a_{1,1}; a_{1,2}], [a_{2,1}; a_{2,2}], \ldots, [a_{M,1}; a_{M,2}]\}$, it is enough to find a map κ from the set of delay fault sizes to the set of resistances, such that $\kappa(\delta) = r$ if an open with resistance r causes a delay of δ. Then, C-ADI $(f) = \{[\kappa(a_{1,1}); \kappa(a_{1,2})], [\kappa(a_{2,1}); \kappa(a_{2,2})], \ldots, [\kappa(a_{M,1}); \kappa(a_{M,2})]\}$.

Deriving the exact, physically accurate map is out of the scope of this work. Proposals for the mapping κ exist [24], but in this first implementation of the RO-simulator, a linear mapping is used. In future it will be replaced by more accurate models as soon as these become available. The RO-simulator was implemented such that new mapping models can be easily integrated into it.

In Phase 3 (line 22 in Figure 3.2), the achieved fault coverage is computed depending on the detection sets of resistance ranges. This topic is discussed in Chapter 6. Due to the linear mapping mentioned above, the fault coverage figures presented in Chapter 7 directly depend on the detection sets of delay fault intervals.

3.2 Application of RO-simulation to sequential circuits

Obviously, only Phase 1's implementation depends on whether the given circuit is combinational or sequential. The delay-to-resistance mapping performed in Phase 2 is done by local analysis and does not depend on the CUT being combinational or sequential. The fault coverage computation performed in Phase 3 is also independent of this question.

Phase 1 implements delay fault simulation which requires two-pattern testing (cf. Section 2.3). As was explained in Section 2.3.2, two-pattern testing cannot be applied to all sequential circuits, but it can be applied to full-scan-capable circuits. However, there is the restriction that each test pair must be applied independently of all others in the test set, and that the propagation vector of each test pair is such that, the values it specifies for the secondary inputs are obtained through the functional path as response to the application of the initialisation vector.

The RO-simulator can work with both combinational and sequential circuits. The delay fault algorithms which form Phase 1 were all enhanced as to accept and process secondary inputs and secondary outputs.

However, the RO-simulator can only be used on full-scan-capable sequential circuits, as the secondary inputs are treated as fully accessible during the application of each initialisation vector and the secondary outputs are treated as fully observable during the application of each propagation vector. Enhanced-scan-capability is not a prerequisite.

4

More Preliminaries

In Chapter 5 Phase 1's design and implementation is discussed. In order to keep the chapters of this thesis rather short and thus, more easy to read, this chapter has been included. This chapter introduces some definitions which are necessary to understand the following chapter; as well as the basics on waveforms and so-called signal descriptors. These are the data structures used to specify the simulation algorithms of the next chapter.

4.1 Definitions and conventions

When building a simulator for delay faults, it is necessary to establish a set of rules that dictate how to represent and handle time and other concepts that depend on time. This set of rules is presented in this section. None of the concepts presented here constitutes original work. These concepts are used in several works that deal with the timing behaviour of circuits. However, other authors may use a different terminology.

4.1.1 Time issues and test application

In this work, points in time, time interval lengths and LDF sizes are represented by integers greater or equal 0. Additionally, there is a point in time denoted by $-\infty$. This point in time plays a special role which will be explained below. The base time unit of the RO-simulator are picoseconds. For simplicity, time units are mainly left out in this thesis.

For simplicity in the case of combinational circuits, and in order to be able to handle sequential circuits, the consecutive application of several test pairs is not considered in this work. Each test pair is assumed to be applied independently of all others in the test set.

A signal s is said to be **`stable`** after the application of a test vector when s's logic value does not change any more.

When simulating a test pair p, the classical method presented on page 37 is used. After the application of p^1, all signals are allowed to stabilise before applying p^2. The point in time, at which all signals in the circuit become stable after having applied p^1, is denoted by $-\infty$. p^2 is applied at time 0.

It would also be possible to apply p^1 at any time t and p^2 at time $t + \mathrm{TC}$. But there is no difference in proceeding the first or the second way. Since this work deals with delay faults which can only be excited by inducing a transition at the fault site, only the effects set in motion by applying p^2 are of interest.

In this context, four parameters are defined for each signal s in the circuit. These parameters are also defined in [20] and [36].

IV (s): `Initial value` of s. This is the stable logic value of s after the application of p^1 (in other words, s's logic value at time $-\infty$).

FV (s): `Final value` of s. This is the stable logic value of s after the application of p^2 in the **fault-free** simulation.

EAT (s): `Earliest arrival time` of s. This is the first point in time at which the value of s changes from $\mathrm{IV}(s)$ to a different value as a reaction to the application of p^2 in the **fault-free** simulation.

LST (s): `Latest stabilisation time` of s. This is the last point in time at which the value of s changes to $\mathrm{FV}(s)$ as a reaction to the application of p^2 in the **fault-free** simulation.

Consider the example in Figure 3.1 (page 46). Let s be the depicted circuit's only primary output signal. Then, $\mathrm{IV}(s)$ and $\mathrm{FV}(s)$ are both 0, $\mathrm{EAT}(s)$ equals 2 (in the fault-free simulation, δ equals 0) and $\mathrm{LST}(s)$ equals 5.

Note that $\mathrm{EAT}(s)$ and $\mathrm{LST}(s)$ have a very important meaning. For any signal s, its logic value is guaranteed to be stable between time $-\infty$ and time $\mathrm{EAT}(s)$ and for ever after time $\mathrm{LST}(s)$. Thus, if a signal s's logic value never changes in reaction to the application of p^2 in the **fault-free** simulation, $\mathrm{EAT}(s)$ have to be defined as $+\infty$ and $\mathrm{LST}(s)$ as $-\infty$.

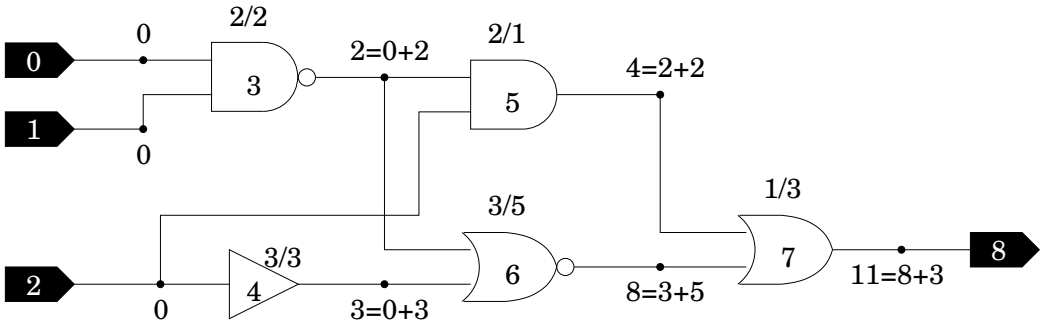


FIGURE 4.1: COMPUTING PLST: AN EXAMPLE

A path is said to be **stable** after the application of an input vector when all signals in the path are stable. The guaranteed time after which a path has stabilised is called **latest stabilisation time** (LST) of the path.

PLST (stands for "**l**atest **s**tabilisation **t**ime of any **p**ath") is defined as the guaranteed time after which all paths in the fault-free circuit have stabilised after applying any input vector. I.e. PLST is the largest path LST among all paths in the circuit. Consider, for example, the circuit in Figure 4.1. Suppose that all signals are stable. Then, let an unknown input vector be applied at time 0. Signals 0, 1 and 2 become stable at time 0 as they are primary inputs. Signal 3 becomes stable at time 2 as all its predecessors become stable at time 0 and gate 3 needs 2 time units to produce both rising and falling transitions. Analogously, signal 4 becomes stable at time 3. In the worst case, signal 5 becomes stable at time $4 = 2 + 2$, as the latest point in time at which both its predecessor signals are guaranteed to be stable is time 2; and because 2 time units is the largest time gate 5 needs to produce a transition. Analogously, the worst case stabilisation time of signal 6 is $8 = 3 + 5$, and the worst case stabilisation time of signal 7 is $11 = 8 + 3$. Thus, PLST equals 11. Figure 4.2 depicts the algorithm used to compute PLST. The primary inputs are assigned the latest stabilisation time 0 (line 8). For each other signal, its latest stabilisation time is computed as the sum of the latest time at which all its predecessors are stable and the largest time its source gate needs to produce a transition (line 10). Finally, PLST is the largest latest stabilisation time among all outputs of the circuit (line 13).

The clock period length TC is usually chosen as PLST·(100+SF)%, where SF is a **safety margin**. In this work, SF is set to 20, i.e. TC := 1.2·PLST. In the example of Figure 4.1, TC is set to 13.

```
 1 PLST_computation
 2     Input:  a circuit C containing NS-many signals
 3     Output: an amount of time representing PLST
 4 BEGIN
 5     initialise an array LST of length NS
 6     for i = 1 to NS ; do
 7         if signal i is a primary or secondary input ; then
 8             set LST[i] := 0
 9         else
10             set LST[i] := max{ LST[j] | signal j is a
                                          predecessor signal
                                          of signal i }
                             + max{RD(i), FD(i)}
11         fi
12     done
13     return max{ LST[j] | signal j is a
                            primary or secondary
                            output }
14 END
```

FIGURE 4.2: ALGORITHM: COMPUTATION OF PLST

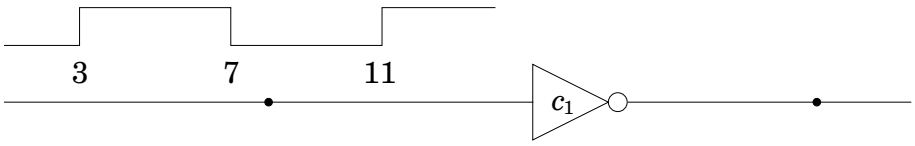


FIGURE 4.3: EXAMPLE ON INPUT INTERVALS OF A SINGLE-INPUT GATE

4.1.2 Input and output intervals

In order to make notions and algorithms introduced in following sections more easy to describe, the concept of input and output intervals is defined in this section. If other authors have defined this concept in this form in the past, we are not aware of that.

Consider the single-input gate c_1 in Figure 4.3. The behaviour of c_1's input signal can be described using 4 intervals: an interval beginning at time $-\infty$ and ending at time 3, during which the signal has the logic value 0, denoted by $0@]-\infty;3[$; a second interval $1@[3;7[$; a third interval $0@[7;11[$; and a fourth interval $1@[11;+\infty[$.

It is obvious that the behaviour over time of any signal in the circuit can be described by a set of **disjoint** intervals analogous to that in this example.

Let c be a single-input gate. The **`set of output intervals`** of c is the set of disjoint intervals describing the behaviour of c's output signal. The **`set of input intervals`** of c is the set of disjoint intervals describing the behaviour of c's input signal.

Once the input intervals of c are known, it is easy to compute c's output intervals. Each input interval induces exactly one output interval according to the delay rules introduced in the following section.

Now consider the case that c is a multiple-input gate. Like before, the **`set of output intervals`** of c is the set of disjoint intervals describing the behaviour of c's output signal.

If c has n input signals, c's overall input could be described by n sets of disjoint input intervals. However, there is a method to reduce the n sets of input intervals of a multiple-input gate to only one set of input intervals. This allows to compute that gate's set of output intervals using the same simple algorithm as in the single-input case.

Let c be a multiple-input gate. The **`set of input intervals`** of c is the set of disjoint intervals obtained according to the following rules:

- c has an input interval $\mathrm{CV}(c)\,@[x;y[$ if, at all times t with $x \leq t < y$, at least one input signal of c has the value $\mathrm{CV}(c)$.

- c has an input interval $\mathrm{X}@[x;y[$ if, at all times t with $x \leq t < y$, no input signal of c has the value $\mathrm{CV}(c)$ and at least one input signal of c has the value X.

- c has an input interval $\mathrm{NCV}(c)\,@[x;y[$ if all input signals of c have the value $\mathrm{NCV}(c)$ at all times t with $x \leq t < y$.

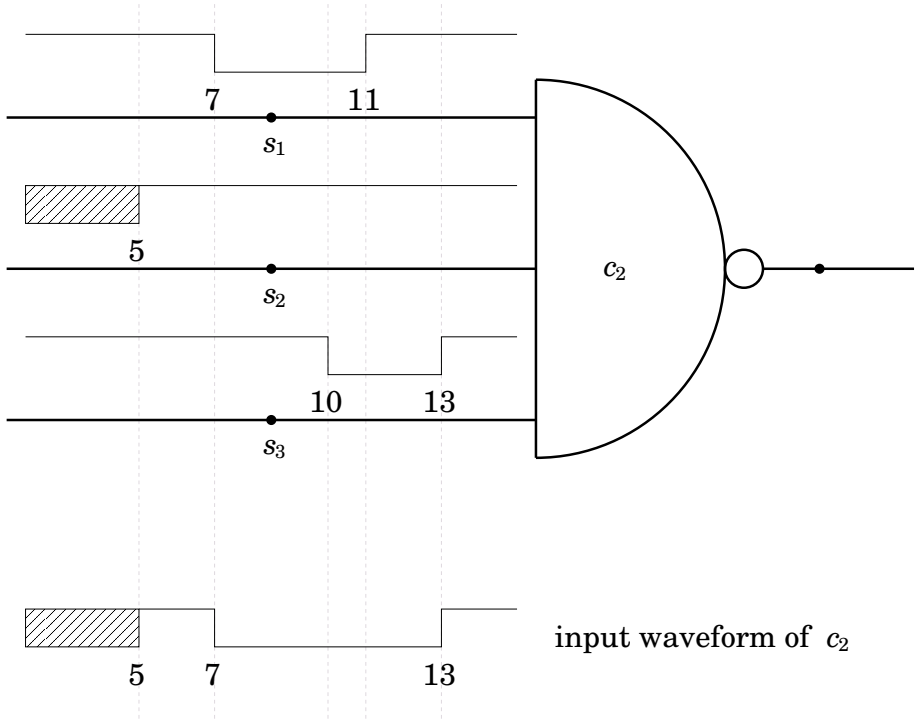


FIGURE 4.4: EXAMPLE ON INPUT INTERVALS OF A MULTIPLE-INPUT GATE

Consider, for example, the multiple-input gate c_2 in Figure 4.4. Signal s_2 has the unknown value X until time 5. At the same time, signals s_1 and s_3 have the value 1. As 1 is NCV (c_2), c_2's input between time $-\infty$ and time 5 depends only on signal s_2. Thus, the first input interval of c_2 is X@$]-\infty; 5[$. Between time 5 and time 7 all three signals have the value 1. Thus, the second input interval of c_2 is $1@[5; 7[$. Between time 7 and time 11 signal s_1 has the logic value 0, CV (c_2). Between time 10 and time 13 signal s_3 has the logic value 0. Altogether, between time 7 and time 13 there is always an input signal of c_2 which has the logic value 0. Thus, the third input interval of c_2 is $0@[7; 13[$. After time 13 all three input signals are stable at logic 1. Thus, the fourth input interval of c_2 is $1@[13; +\infty[$.

Altogether, the set of input intervals of c_2 comprises the intervals X@$]-\infty; 5[$, $1@[5; 7[$, $0@[7; 13[$ and $1@[13; +\infty[$. These intervals are represented by the waveform in the lowest part of Figure 4.4. These four intervals are enough to determine c_2's output intervals. Each interval induces exactly one output interval according to the delay rules that follow in the next section.

4.1.3 Delay model

The term **`delay model`** shall not be confounded with the term delay fault model. A delay model is a set of rules specifying how and when a gate responds to the change of one or more of its inputs.

[20] introduces the **`distributed delay model`**, which tries to reflect the fact that delays in actual fabricated circuits vary between upper and lower bounds, especially due to manufacturing process variations. The model has the following characteristics:

1) The delay through a gate may vary depending on whether the gate produces a rising or a falling transition.

2) Different gates of the same type may have different delays.

3) A gate has some inertia in responding to changes in its input values. That means that certain input pulses are filtered out in the gate response if they are too short.

Here, a simplified version of the distributed delay model is used:

1) The delay through a gate c is defined by its rising delay time $\mathrm{RD}(c)$ and by its falling delay time $\mathrm{FD}(c)$.
Let s be c's output signal. Given the set of input intervals of c, the set of output intervals is computed in the following way:

   (i) Each input interval with value 1 or value 0 induces one output interval according to the rules listed in Table 4.1.

   (ii) s is defined to have the unknown value X at all times t which are not in any interval computed in the first step.

   The table is read in the following way: consider, for example, the table's first row. An input interval $0@[x; y[$ induces the output interval $0@[x+\mathrm{FD}(c)\,; y+\mathrm{RD}(c)\,[$ if the gate c is not inverting (if c is inverting, the table's second row determines the output interval). However, the output interval is only induced if the additional condition $x+\mathrm{FD}(c) \leq y$ is met. If this additional condition is not met, no output interval is induced at all. The intuition behind this additional condition is best explained by means of an example. Let I be a 1-interval and let c be non-inverting. I begins with a rising transition τ_1 and ends with a falling transition τ_2. τ_1 induces a rising transition τ_3 at c's output; but

input int.	***c* inv.**	**add. condition**	**output interval**
$0@[x;y[$	no	$x + \mathrm{FD}(c) \leq y$	$0@[x + \mathrm{FD}(c) ; y + \mathrm{RD}(c) [$
$0@[x;y[$	yes	$x + \mathrm{RD}(c) \leq y$	$1@[x + \mathrm{RD}(c) ; y + \mathrm{FD}(c) [$
$1@[x;y[$	no	$x + \mathrm{RD}(c) \leq y$	$1@[x + \mathrm{RD}(c) ; y + \mathrm{FD}(c) [$
$1@[x;y[$	yes	$x + \mathrm{FD}(c) \leq y$	$0@[x + \mathrm{FD}(c) ; y + \mathrm{RD}(c) [$

TABLE 4.1: DELAY MODEL

if τ_2 arrives before τ_3 has been produced, it is not possible to guarantee that c is still able to produce τ_3.
In the table, x is allowed to be $-\infty$ and y is allowed to be $+\infty$. Additionally, we define $\pm\infty + k = \pm\infty$ where k is $\mathrm{RD}(c)$ or $\mathrm{FD}(c)$.

2) The second property of the distributed delay model can be modelled as an increase or decrease from the nominal delay by a maximum of σ [20, p. 79]. For simplicity, this extension is not used in this work. Here, $\mathrm{RD}(c)$ and $\mathrm{FD}(c)$ are single fixed integers.

3) The gate's inertia (Property 3 of the distributed delay model) is accounted for to some extent by this simplified model's first property. Let us illustrate this with an example (see Figure 4.5). Let c be an AND gate, let $\mathrm{RD}(c)$ be 7, $\mathrm{FD}(c)$ be 3. The input interval $1@[t;t+a[$ induces the output interval $1@[t+7;t+a+3[$, but only if $t+7 \leq t+a$. That means that the output interval is induced only if $a \geq 7$. I.e. 1-pulses shorter than 7 time units are filtered out.

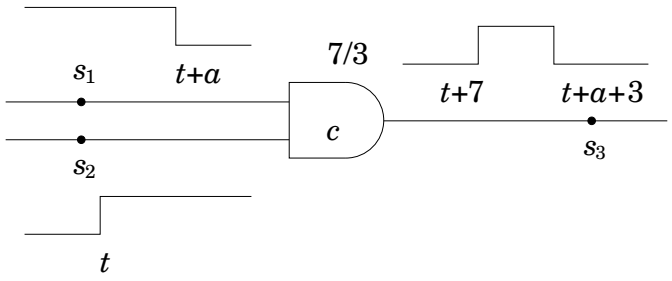


FIGURE 4.5: EXAMPLE ON THE APPLICATION OF THE DELAY MODEL

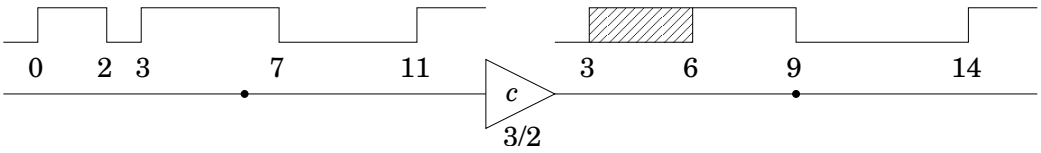


FIGURE 4.6: EXAMPLE ON THE APPLICATION OF THE DELAY MODEL

This section is finalised with a further example of the application of the delay model. Let c be the single-input gate in Figure 4.6. $\mathrm{RD}(c)$ is 3 and $\mathrm{FD}(c)$ is 2. The task is to compute the set of output intervals of c. The set of input intervals of c is composed of the intervals $0@]-\infty;0[$, $1@[0;2[$, $0@[2;3[$, $1@[3;7[$, $0@[7;11[$ and $1@[11;+\infty[$.

Since c is not inverting, only the first and the third delay rules (i.e. the first and third rows of Table 4.1, resp.) are needed. $0@]-\infty;0[$ induces the output interval $0@]-\infty+\mathrm{FD}(c);0+\mathrm{RD}(c)[$, i.e. $0@]-\infty;3[$. $1@[0;2[$ is filtered out since the additional condition $0+\mathrm{RD}(c)\leq 2$ is not met. $0@[2;3[$ is also filtered out since the additional condition $2+\mathrm{FD}(c)\leq 3$ is not met. $1@[3;7[$ induces the output interval $1@[3+\mathrm{RD}(c);7+\mathrm{FD}(c)[$, i.e. $1@[6;9[$. $0@[7;11[$ induces the output interval $0@[7+\mathrm{FD}(c);11+\mathrm{RD}(c)[$, i.e. $0@[9;14[$. Finally, $1@[11;+\infty[$ induces the output interval $1@[11+\mathrm{RD}(c);+\infty+\mathrm{FD}(c)[$, i.e. $1@[14;+\infty[$.

Until now, the output intervals $0@]-\infty;3[$, $1@[6;9[$, $0@[9;14[$ and $1@[14;+\infty[$ have been determined. The union of these intervals does not include the time between time 3 and time 6. Thus, the interval $X@[3;6[$ has to be added to c's set of output intervals.

4.2 Waveforms

Waveforms are a very common way of describing the behaviour of signals, not only because waveforms are very intuitive, but also because they can be implemented easily. In previous sections of this chapter waveforms were already used to represent graphically the sets of intervals introduced in section 4.1.2. Basically, a waveform is not more than a more compact representation of a set of intervals. In this section some formal definitions on waveforms are presented. Waveforms will be needed to describe the fault-free simulation algorithm in Chapter 5. None of the following concepts is new, but other authors may use a different terminology or different operations on waveforms.

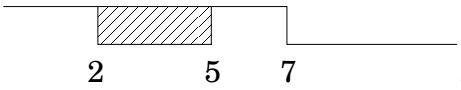
4.2.1 Initial definitions

Let s be a signal and let t be a point in time. Then, $\boldsymbol{s(t)}$ denotes the logic value signal s has at time t. t is allowed to be $-\infty$. Then, $s(-\infty)$ is the value signal s stabilises at after the application of the initialisation vector of the currently treated test pair.

The **waveform of a signal** s is defined as the tuple or sequence of pairs

$$\boldsymbol{W_s} := \big((t, s(t))\big|t = -\infty \text{ or the logic value of } s \text{ changes at time } t\big),$$

where the pairs (also called **transitions**) are ordered by time. For example, the waveform



,

which represents graphically the set of intervals

$$\big\{1@]-\infty;2[,\ \mathrm{X}@[2;5[,\ 1@[5;7[,\ 0@[7;+\infty[\big\},$$

is formalised as the tuple $W := W_s = \big((-\infty, 1),\ (2, \mathrm{X}),\ (5, 1),\ (7, 0)\big)$.

$\boldsymbol{|W|}$ denotes the **number of transitions** of W, i.e. the number of pairs W is composed of. In the current example, $|W_s| = 4$.

Let i be in $\big\{1, 2, \ldots, |W|\big\}$. Then, $\boldsymbol{t_W(i)}$ denotes the first component of the i-th pair in W, i.e. the i-th point in time at which the waveform undergoes a transition. In the current example, $t_W(1) = -\infty$, $t_W(2) = 2$, $t_W(3) = 5$ and $t_W(4) = 7$.

Let t be a point in time. Then, $\boldsymbol{v_W(t)}$ denotes the logic value the waveform has at time t. We define further $v_W(+\infty) := v_W\big(t_W(|W|)\big)$, i.e. $v_W(+\infty)$ denotes the last logic value the waveform changes to. In the current example, $v_W(-\infty) = 1$, $v_W(5) = 1$, $v_W(6) = 1$ and $v_W(+\infty) = 0$.

Note that the parameters defined in Section 4.1.1 can be extracted directly from the waveform:

- $\mathrm{IV}\,(s) = v_{W_s}(-\infty)$
- $\mathrm{FV}\,(s) = v_{W_s}(+\infty)$
- $\mathrm{EAT}\,(s) = \begin{cases} t_{W_s}(2) & \text{if } |W_s| \geq 2 \\ +\infty & \text{else} \end{cases}$
- $\mathrm{LST}\,(s) = t_{W_s}(|W_s|)$

In the current example, $\mathrm{IV}\,(s) = 1$, $\mathrm{FV}\,(s) = 0$, $\mathrm{EAT}\,(s) = 2$ and $\mathrm{LST}\,(s) = 7$.

4.2.2 Intersection of waveforms

In Section 4.1.2 it was demonstrated that it is possible to describe the overall input of a multiple-input gate c with n inputs using only one set of input intervals instead of n sets corresponding to its n input signals. Since a waveform is none but a compact representation of a set of intervals, it is also possible to describe the overall input of c using only one waveform which is called the **`input waveform`** of c.

In order to be able to define the input waveform of a multiple-input gate, it is necessary to first introduce the intersection of waveforms.

Let c be a 2-input gate, and let s_1 and s_2 be its input signals. Let W_1 and W_2 be their waveforms.

The **`0-intersection`** of W_1 and W_2 is defined as

$$W_1 \cap_0 W_2 := \left(\big(t, \max_0(v_{W_1}(t), v_{W_2}(t)\big) \;\middle|\; \begin{array}{l} \text{There is a } v \text{ with} \\ (t,v) \in W_1 \text{ or} \\ (t,v) \in W_2. \end{array} \right),$$

where $\max_0$ is defined by the following table:

$\max_0$	0	X	1
0	0	0	0
X	0	X	X
1	0	X	1

For example, let $W_1 := \big((-\infty, 1),\ (2, \mathrm{X}),\ (5, 1),\ (7, 0)\big)$ and $W_2 := \big\{(-\infty, 0),\ (1, 1),\ (3, 0),\ (5, \mathrm{X}),\ (8, 0),\ (10, 1)\big\}$. The 0-intersection of the two waveforms is
$W_3 := W_1 \cap_0 W_2 = \big((-\infty, 0),\ (1, 1),\ (2, \mathrm{X}),\ (3, 0),\ (5, \mathrm{X}),\ (7, 0)\big)$.

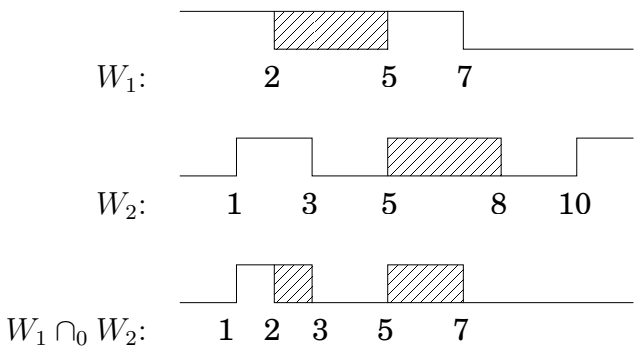


Figure 4.7: Example: 0-intersection of waveforms

These waveforms are represented graphically as shown in Figure 4.7. As the graphic shows, the intersection waveform has the value 0 at all times at which W_1 or W_2 or both have the value 0. W_3 has the value X at all times at which neither W_1 nor W_2 have the value 0 while W_1 or W_2 or both have the value X. Finally, W_3 has the value 1 only at those times at which both W_1 and W_2 have the value 1. Thus, W_3 represents exactly the set of input intervals that one gets by applying the rules of page 55 on the sets of intervals of s_1 and s_2 if $\mathrm{CV}(c) = 0$.

Analogously, the **`1-intersection`** of W_1 and W_2 is defined as

$$W_1 \cap_1 W_2 := \left(\big(t, \max_1(v_{W_1}(t), v_{W_2}(t)\big) \;\middle|\; \begin{array}{l} \text{There is a } v \text{ with} \\ (t,v) \in W_1 \text{ or} \\ (t,v) \in W_2. \end{array} \right),$$

where $\max_1$ is defined by the following table:

$\max_1$	1	X	0
1	1	1	1
X	1	X	X
0	1	X	0

For example, the 1-intersection of W_1 and W_2 is $W_4 := W_1 \cap_0 W_2 = \big((-\infty, 1),\ (3, \mathrm{X}),\ (5, 1),\ (7, \mathrm{X}),\ (8, 0),\ (10, 1)\big)$. These waveforms are represented graphically as shown in Figure 4.8. As the graphic shows, W_4 represents exactly the set of input intervals that one gets by applying the rules of page 55 on the sets of intervals of s_1 and s_2 if $\mathrm{CV}(c) = 1$.

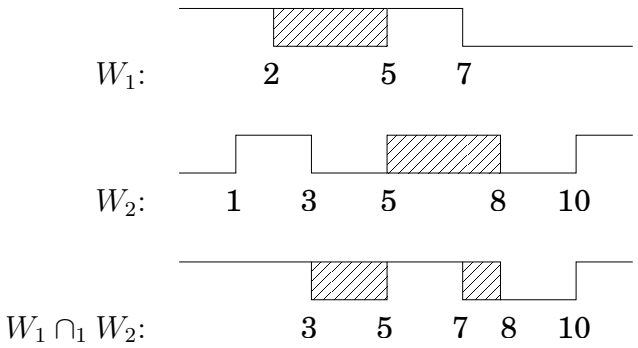


FIGURE 4.8: EXAMPLE: 1-INTERSECTION OF WAVEFORMS

For $val \in \{0,1\}$, $n \geq 2$ and n waveforms W_1, W_2, ..., W_n, we define

$$W_1 \cap_{val} W_2 \cap_{val} \cdots \cap_{val} W_n := ((W_1 \cap_{val} W_2) \cap_{val} \dots) \cap_{val} W_n.$$

Finally, it is obvious that the input waveform W of a multiple-input gate c with n inputs represented by waveforms W_1, W_2, ..., W_n, is computed in the following way:

- If CV $(c) = 0$, W is set to $W_1 \cap_0 W_2 \cap_0 \cdots \cap_0 W_n$.
- If CV $(c) = 1$, W is set to $W_1 \cap_1 W_2 \cap_1 \cdots \cap_1 W_n$.

For completeness it has to be mentioned that the input waveform of a single-input gate equals the waveform of its only input signal.

4.2.3 Translation of waveforms

In this section the translation of waveforms is defined. This operation is necessary to compute the output waveform of a gate c out of its input waveform.

Let W be a waveform, and let rd and fd be delays (amounts of time). Then, $\boldsymbol{T_{rd,fd}(W)}$ denotes the waveform which results from delaying all rising edges of W by rd and all falling edges by fd, where pulses of W which are too short according to the used delay model (see Table 4.1 on page 58) are filtered out. Note that, if W is the input waveform of a gate c of type BUF, then $T_{\mathrm{RD}(c),\mathrm{FD}(c)}(W)$ describes c's output waveform.

For example, let $W_1 := \big((-\infty,0),\ (0,1),\ (2,0),\ (3,1),\ (7,0),\ (11,1)\big)$. Then, $W_2 := T_{3,2}(W_1) = \big((-\infty,0),\ (3,\mathrm{X}),\ (6,1),\ (9,0),\ (14,1)\big)$ is the waveform that results from delaying W_1's rising edges by 3 time units and W_1's falling edges by 2 time units. 1-pulses shorter than 3 time units and 0-pulses shorter than 2 time units are filtered out according to the delay model. These waveforms are represented graphically as shown in Figure 4.9.

Observing the graph, it is easy to see how W_2 is computed. Each pair in W_1 induces one pair in W_2:

- The first pair of W_1 is $(-\infty, 0)$. This is also the first pair of W_2 as $-\infty + 2 = -\infty$.
- The second pair of W_1 is $(0, 1)$ and represents a rising edge which must be delayed by 3 time units. Thus, the second pair of W_2 must have

the form $(3, val)$. As the next transition of W_1, $(2, 0)$, arrives before time 3, it cannot be guaranteed that W_2 really gets the logic value 1 at time 3; hence, $val \neq 1$. However, it is also not possible to guarantee that W_2 will remain stable at logic 0. Thus, val has to be set to X, the unknown logic value. The second pair of W_2 is $(3, \mathrm{X})$.

- The third pair of W_1 is $(2, 0)$ and represents a falling edge which must be delayed by 2 time units. Thus, the third pair of W_2 must have the form $(4, val')$. As the next transition of W_1, $(3, 1)$, arrives before time 4, it cannot be guaranteed that W_2 really gets the logic value 0 at time 4; hence, $val' \neq 0$. Since the value of W_2 before time 4 is unknown, val' is also unknown. The third pair of W_2 is $(4, \mathrm{X})$. However, as this pair and the preceding one, $(3, \mathrm{X})$, specify the same logic value, X, this pair can be removed. W_2 remains with only two pairs: $(-\infty, 0)$ and $(3, \mathrm{X})$.

- The fourth pair of W_1 is $(3, 1)$ and represents a rising edge which must be delayed by 3 time units. Thus, the next pair of W_2 is $(3 + 3, 1)$, i.e. $(6, 1)$. Now, W_2 is composed of three pairs: $(-\infty, 0)$, $(3, \mathrm{X})$ and $(6, 1)$.

- The fifth pair of W_1 is $(7, 0)$ and represents a falling edge which must be delayed by 2 time units. Thus, the next pair of W_2 is $(7 + 2, 0)$, i.e. $(9, 0)$. Now, W_2 is composed of four pairs: $(-\infty, 0)$, $(3, \mathrm{X})$, $(6, 1)$ and $(9, 0)$.

- The sixth pair of W_1 is $(11, 1)$ and represents a rising edge which must be delayed by 3 time units. Thus, the next pair of W_2 is $(11 + 3, 1)$, i.e. $(14, 1)$. Finally, W_2 is composed of five pairs: $(-\infty, 0)$, $(3, \mathrm{X})$, $(6, 1)$, $(9, 0)$ and $(14, 1)$.

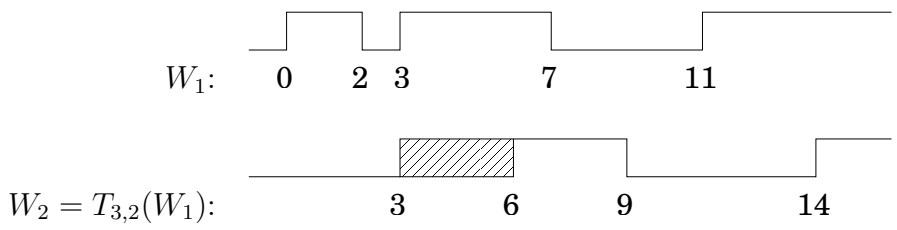


FIGURE 4.9: EXAMPLE: TRANSLATION OF WAVEFORMS

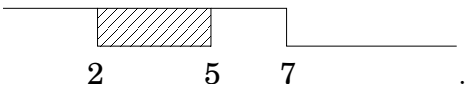
Comparing this example with that in Figure 4.6, it can be seen that the translation of the input waveform of a non-inverting gate c yields the same result as computing the set of output intervals of c as introduced in Section 4.1.3.

4.2.4 Inversion of waveforms

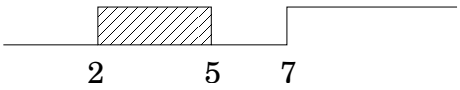
$\overline{W}$ denotes the **`inverse waveform`** of W. It is defined as

$$\overline{W} := \big\{(t, \overline{v}) \big| (t, v) \in W\big\}.$$

For example, let W be the waveform $\big((-\infty, 1),\ (2, \mathrm{X}),\ (5, 1),\ (7, 0)\big)$, which is represented graphically in the following way:



Its inverse waveform



has at all times t the logic inverse of the value W has at that time t. In this example, the inverse waveform is formalised as

$$\overline{W} = \big\{(-\infty, 0),\ (2, \mathrm{X}),\ (5, 0),\ (7, 1)\big\}.$$

The examples in this section (4.2) were chosen such as to illustrate all important aspects of the algorithms on waveforms (0-intersection, 1-intersection, translation and inversion). We renounce to describe them in detail using pseudo-code. Due to the simple and compact representation of waveforms as sequences of pairs, the algorithms are easy to implement.

4.3 Signal descriptors

In Section 4.1.2 we saw that the behaviour over time of a signal can be described by a set of intervals of the form $val@[x; y[$. In Section 4.2 we saw that waveforms can be used to represent sets of intervals in a more compact way. In this section, a method to represent the behaviour of signals in a faulty circuit is presented. In a circuit having an LDF of size δ, some signals have one or more of their transitions delayed by δ (with respect to the fault-free case). Such signals are called **fault-affected**. It would be possible to describe the behaviour of those signals using waveforms composed of two types of pairs, pairs of the form (x, val) and pairs of the form $(x + \delta, val)$. However, Chapter 5 will show that waveforms include information which is not needed during faulty-circuit simulation. For example, only determinate portions of the waveform (portions with logic value 0 or 1) are relevant during faulty-circuit simulation, since a fault can only be detected if it induces a wrong but known value at an output of the circuit. Furthermore, the only part of the waveform of a **non-fault-affected** signal (none of its edges is delayed by δ) which is relevant during faulty-circuit simulation is its last transition.

In this section the concept of **signal descriptors** is introduced. The signal descriptor of a signal s, denoted by $\boldsymbol{SD_s}$, is a set of **description intervals** describing the behaviour of s during faulty-circuit simulation of one fixed test pair and one fixed LDF of size δ. The description intervals in signal descriptors are an enhancement of the intervals defined in Section 4.1.2. Signal descriptors were first defined in [20] (though without first defining the simple intervals of Section 4.1.2), and used with modifications in [36] and [19]. The signal descriptors used in this work are most similar to those in [36]. However, there are some modifications which will be pointed at later.

4.3.1 Initial definitions

Let C be the CUT, f an LDF of size δ, C^f the faulty version of C which has the fault f, and let p be a fixed test set (p^1 denotes the initialisation vector, p^2 the propagation vector).

A signal descriptor is a set of description intervals, where each description interval I has the form $\boldsymbol{val@[left; right[}\ \ \boldsymbol{\delta_{min} \leq \delta \leq \delta_{max}}$. val is 0 or 1, $\delta_{\min} \in \mathbb{N}$, $\delta_{\max} \in \{\delta_{\min}, \delta_{\min}+1, \delta_{\min}+2, \dots\} \cup \{+\infty\}$. $left$ is of the form $\boldsymbol{x}$ where $x \in \mathbb{N} \cup \{-\infty\}$, or of the form $\boldsymbol{x + \delta}$ where $x \in \mathbb{N}$. $right$ is of the form $\boldsymbol{y}$ where $y \in \{x+1, x+2, \dots\} \cup \{+\infty\}$, or of the form $\boldsymbol{y + \delta}$ where

$y \in \mathbb{N}$. For improved readability, in this thesis we write:

$val@[left; right[$ instead of $val@[left; right[\ \ 0 \leq \delta < +\infty$,
$val@[left; right[\ \ \delta \geq \delta_{\min}$ instead of $val@[left; right[\ \ \delta_{\min} \leq \delta < +\infty$,
$val@[left; right[\ \ \delta \leq \delta_{\max}$ instead of $val@[left; right[\ \ 0 \leq \delta \leq \delta_{\max}$.

Let $I : val@[left; right[\ \ \delta_{\min} \leq \delta \leq \delta_{\max}$ be a description interval in SD_s for a signal s. The meaning of I is the following: when applying the test pair p to C_f, signal s has the logic value val at all times t with $left \leq t < right$, but only if $\delta_{\min} \leq \delta \leq \delta_{\max}$. For example, if the description interval $1@[4+\delta; 7+\delta[\ \ \delta \leq 47$ is contained in SD_s, this means, that s has the logic value 1 between time 4 and time 7 if C is fault-free; or between time $4+\delta$ and time $7+\delta$ if C has the fault f and f has size $\delta \leq 47$.

The description interval $val@[left; right[\ \ \delta_{\min} \leq \delta \leq \delta_{\max}$ is called **fault-affected** if $left$ is of the form $x + \delta$ or if $right$ is of the form $y + \delta$. Otherwise, the description interval is called **non-fault-affected**. A signal descriptor is called **fault-affected** if it contains at least one fault-affected description interval. Noting that actual fault effect propagation takes place under p^2, if s is non-fault-affected, the only part of its waveform which is necessary to consider in the computation of SD_s is the stable final-valued one stretching from LST (s) to $+\infty$ (cf. example in Section 5.1.2 and [36, page 83]). Thus, the signal descriptor of a non-fault-affected signal s always contains exactly one description interval of the form FV (s) $@[x; +\infty[$ for an $x \geq$ LST (s).

The example in Section 5.1.2 will illustrate the fact that some description intervals can only come into existence if δ is within a certain range $[\delta_{\min}; \delta_{\max}]$. This range is called the **constraint on** the description interval. The constraint on a description interval I is denoted by K_I (a notation including a C would be more intuitive, but C's are already used to denote circuits). In [36] constraints are only of the form $\delta \geq \delta_{\min}$, while [19] introduces two constraints per interval, an optimistic and a pessimistic one. We stick to the idea in [36] and use only one constraint per interval, however this only constraint is extended to have the already mentioned form $\delta_{\min} \leq \delta \leq \delta_{\max}$. $\delta_{\min}$ and $\delta_{\max}$ depend on the form of the description interval. How they are computed is presented in Section 5.2.2.

Let c be a gate, let s be its output signal and let s_1, s_2, ..., s_n be its input signals. Let SD_s, SD_1, SD_2, ..., SD_n be their corresponding signal descriptors. SD_s is called the **output signal descriptor** of c and all description intervals in SD_s are called **output description intervals** of c. SD_1, SD_2, ..., SD_n are called the **input signal descriptors** of c and all their description intervals are called **input description intervals** of c.

4.3.2 cv-intervals and ncv-intervals

Throughout this section, let c be a multiple-input gate, let s be its output signal and let s_1, s_2, ..., s_n be its input signals. Let SD_s, SD_1, SD_2, ..., SD_n be their corresponding signal descriptors.

In Section 4.2, we saw that it is possible to describe the overall input of c using only one waveform, namely the intersection of W_{s_1}, W_{s_2}, ..., W_{s_n}. Since the structure of description intervals is far more complex than that of simple intervals defined in Section 4.1.2, it is very difficult to formulate a meaningful definition of the intersection of signal descriptors. However, this is not necessary. In order to be able to compute the output description intervals of c, it is enough to define two more simple concepts, cv-intervals and ncv-intervals. cv- and ncv-intervals are used in [36], however without giving them a name and without specifying how to compute them. The following definitions constitute original work.

For each $i = 1, 2, \ldots, n$, let $SD_i^{\mathrm{CV}(c)}$ be the set of all description intervals in SD_i, which have the form $\mathrm{CV}(c)\,@[left_I, right_I[\ \ K_I$. Analogously, let $SD_i^{\mathrm{NCV}(c)}$ be the set of all description intervals in SD_i, which have the form $\mathrm{NCV}(c)\,@[left_I, right_I[\ \ K_I$.

4.3.2.1 cv-intervals

The **set of cv-intervals** of c is the set $\bigcup_{i=1}^{n} \boldsymbol{SD_i^{\mathrm{CV}(c)}}$, i.e. the set of all input intervals of c which have the value $\mathrm{CV}(c)$. Remember that if an input signal of c has the logic value $\mathrm{CV}(c)$, this always induces the output value $\mathrm{CV}(c)$ ($\mathrm{NCV}(c)$ if c is inverting), independently of the logic values of all other signals. Thus, cv-intervals have the following property: an output description interval of c, which is induced by a cv-interval $I : \mathrm{CV}(c)\,@[left_I; right_I[\ \ K_I$ in SD_i for an $i = 1, 2, \ldots, n$, is independent of how all other input signals of c (s_1, s_2, ..., s_{i-1}, s_{i+1}, ..., s_n) behave between time $left_I$ and $right_I$. Hence, each cv-interval of c describes a portion of the overall input of c.

4.3.2.2 Intersection of description intervals

The definition of ncv-intervals shall be formulated such that each ncv-interval describes a portion of the overall input of c. Remember that c can produce the output $\mathrm{NCV}(c)$ ($\mathrm{CV}(c)$ if c is inverting) only if all its input signals have the logic value $\mathrm{NCV}(c)$ simultaneously. In order to be able to express this simultaneousness, the intersection of description intervals must be defined.

Let signals s_a and s_b be the inputs of a two-input gate g and let SD_a and SD_b be their signal descriptors. Let val be a determinate logic value (0 or 1). Let $I : val@[left_I; right_I[\ \ \delta^I_{\min} \leq \delta \leq \delta^I_{\max}$ be a description interval in SD_a and let $J : val@[left_J; right_J[\ \ \delta^J_{\min} \leq \delta \leq \delta^J_{\max}$ be a description interval in SD_b. Then, $\boldsymbol{I \cap J}$ denotes the **intersection** of I and J and is regarded as input description interval of g. Let $K := I \cap J$ be of the form $val@[left_K; right_K[\ \ \delta^K_{\min} \leq \delta \leq \delta^K_{\max}$. The meaning of K is that both inputs of g have the logic value val at all times t with $left_K \leq t < right_K$, but only if $\delta^K_{\min} \leq \delta \leq \delta^K_{\max}$. This concept is easily extended to the case that g has more than two inputs, as $I_1 \cap I_2 \cap \cdots \cap I_n := ((I_1 \cap I_2) \cap \ldots) \cap I_n$.

Note that the intersection of I and J is only well-defined if both intervals have the same logic value. Thus, for the computation of K, the logic value of I and J is ignored. Computing K consists in determining all points in time which belong to both I and J; and determining all δ values which fulfil both constraints $\delta^I_{\min} \leq \delta \leq \delta^I_{\max}$ and $\delta^J_{\min} \leq \delta \leq \delta^J_{\max}$, and which meet natural constraints imposed by the form of $left_K$ and $right_K$.

As $left_I$, $left_J$, $right_I$ and $right_J$ are of various forms, several cases need to be handled.

Computation of $\boldsymbol{left_K}$ and $\boldsymbol{\delta^K_{\min}}$ There are four cases:

Case 1: $left_I = x_I$ and $left_J = x_J$.
$left_K$ is set to $\max\{x_I, x_J\}$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}\}$.

Case 2: $left_I = x_I + \delta$ and $left_J = x_J + \delta$.
$left_K$ is set to $\max\{x_I, x_J\} + \delta$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}\}$.

Case 3: $left_I = x_I$ and $left_J = x_J + \delta$.
If $x_I \leq x_J$, $left_K$ is set to $x_J + \delta$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}\}$. Else, $left_K$ is set to $x_J + \delta$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}, x_I - x_J\}$.

Case 4: $left_I = x_I + \delta$ and $left_J = x_J$.
If $x_J \leq x_I$, $left_K$ is set to $x_I + \delta$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}\}$. Else, $left_K$ is set to $x_I + \delta$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}, x_J - x_I\}$.

Computation of $right_K$ and $\delta^K_{\max}$ There are four cases:

Case 1: $right_I = y_I$ and $right_J = y_J$.
$right_K$ is set to $\min\{y_I, y_J\}$ and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}\}$.

Case 2: $right_I = y_I + \delta$ and $right_J = y_J + \delta$.
$right_K$ is set to $\min\{y_I, y_J\} + \delta$ and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}\}$.

Case 3: $right_I = y_I$ and $right_J = y_J + \delta$.
If $y_I \leq y_J$, $right_K$ is set to y_I and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}\}$. Else, $right_K$ is set to $y_J + \delta$ and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}, y_I - y_J\}$.

Case 4: $right_I = y_I + \delta$ and $right_J = y_J$.
If $y_J \leq y_I$, $right_K$ is set to y_J and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}\}$. Else, $right_K$ is set to $y_I + \delta$ and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}, y_J - y_I\}$.

For example, let I be $val@[17; 27 + \delta[$ $2 \leq \delta \leq 10$. In Figure 4.10, I is represented as a bold line beginning at time 17. $right_I$ is $27 + \delta$, where $2 \leq \delta \leq 10$. That means that the right limit of I may be between time $27 + 2 = 29$ and time $27 + 10 = 37$ depending on the fault's size δ. This is represented by the grey region in the figure. The bold line representing I ends at time 37 which is the latest time for I's right limit.

Let J be $val@[10 + \delta; 36[$ $3 \leq \delta \leq 11$. $left_J$ is $10 + \delta$, where $3 \leq \delta \leq 11$. That means that the left limit of J may be between time $10 + 3 = 13$ and time $10 + 11 = 21$ depending on the fault's size δ. This is represented by the grey region in the figure. J is represented as a bold line beginning at time 13, which is the earliest time for J's left limit, and ending at time 36.

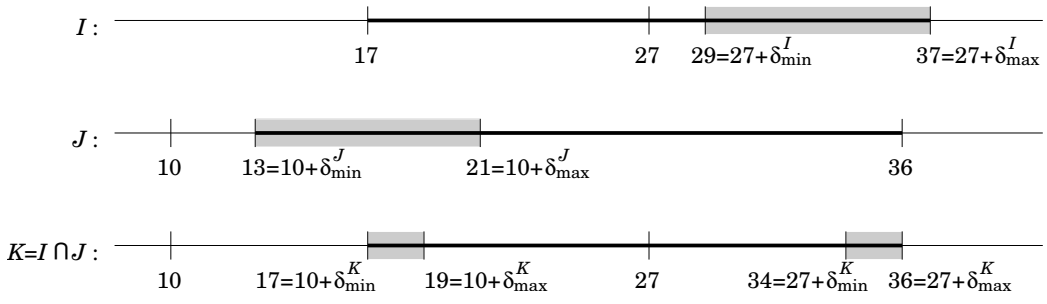


FIGURE 4.10: INTERSECTION OF DESCRIPTION INTERVALS WITH THE SAME LOGIC VALUE

First, $left_K$ and $\delta^K_{\min}$ are computed. $left_I = 17$ and $left_J = 10 + \delta$. Here Case 3 is applied. Since $17 \not\leq 10$, $left_K$ is set to $10 + \delta$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}, x_I - x_J\} = \max\{2, 3, 17 - 10\} = 7$. The intuition behind this is the following: Due to $\delta^K_{\min} = 7$, it is guaranteed that K's left limit is not before time $10 + 7 = 17 = left_I$. That means, for all valid δ values, K will not contain any points which are not contained in I.

Then, $right_K$ and $\delta^K_{\max}$ are computed. $right_I = 27 + \delta$ and $right_J = 36$. Here Case 4 is applied. Since $36 \not\leq 27$, $left_K$ is set to $27 + \delta$ and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}, y_J - y_I\} = \min\{10, 11, 36 - 27\} = 9$. The intuition behind this is the following: Due to $\delta^K_{\max} = 9$, it is guaranteed that K's right limit is not after time $27 + 9 = 36 = right_J$. That means, for all valid δ values, K will not contain any points which are not contained in J.

Finally, K is $val@[10 + \delta; 27 + \delta[\;\; 7 \leq \delta \leq 9$.

A second example will show that it is not always possible to intersect two intervals. Let I be $val@[22; 27 + \delta[\;\; 2 \leq \delta \leq 10$ and let J be $val@[35 + \delta; 52[\;\; 3 \leq \delta \leq 11$. Already their graphical representation in Figure 4.11 shows that I and J cannot be intersected, since both bold lines do not have any point in common. However, let us compute K as was done before.

First, $left_K$ and $\delta^K_{\min}$ are computed. $left_I = 22$ and $left_J = 35 + \delta$. Here Case 3 is applied. Since $22 \leq 35$, $left_K$ is set to $35 + \delta$ and $\delta^K_{\min}$ to $\max\{\delta^I_{\min}, \delta^J_{\min}\} = \max\{2, 3\} = 3$.

Then, $right_K$ and $\delta^K_{\max}$ are computed. $right_I = 27 + \delta$ and $right_J = 52$. Here Case 4 is applied. Since $52 \not\leq 27$, $left_K$ is set to $27 + \delta$ and $\delta^K_{\max}$ to $\min\{\delta^I_{\max}, \delta^J_{\max}, y_J - y_I\} = \min\{10, 11, 52 - 27\} = 10$.

Finally, K is $val@[35 + \delta; 27 + \delta[\;\; 3 \leq \delta \leq 10$.

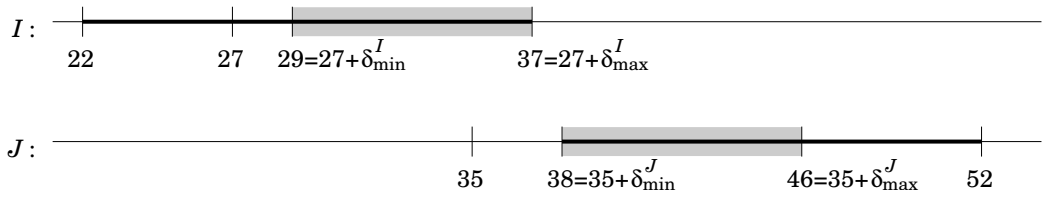


FIGURE 4.11: INTERSECTION OF DESCRIPTION INTERVALS WITH THE SAME LOGIC VALUE

It is easy to see that $val@[35+\delta; 27+\delta[\;\; 3 \leq \delta \leq 10$ is not a valid description interval, because $35+\delta \not< 27+\delta$ for all δ with $3 \leq \delta \leq 10$. A time interval whose left limit is not earlier than its right limit cannot be valid.

Two description intervals I and J are said to be **intersect-able** if their intersection $K := I \cup J$ is a valid description interval.

The best way of determining if I and J are intersect-able is by computing $K := I \cup J$ using the rules introduced on page 69 and checking if K has the following properties:

Property 1: $\delta^K_{\min} \leq \delta^K_{\max}$ must be true.

Property 2: Depending on its form, K must meet one of the following three conditions:

Case 1: K is of the form $val@[x_K; y_K[\;\; K_K$ or $val@[x_K+\delta; y_K+\delta[\;\; K_K$.
$x_K < y_K$ must be true.

Case 2: K is of the form $val@[x_K+\delta; y_K[\;\; \delta^K_{\min} \leq \delta \leq \delta^K_{\max}$.
$x_K + \delta^K_{\max} < y_K$ must be true. If $\delta^K_{\max} = +\infty$, y_K must be $+\infty$.

Case 3: K is of the form $val@[x_K; y_K+\delta[\;\; \delta^K_{\min} \leq \delta \leq \delta^K_{\max}$.
$x_K < y_K + \delta^K_{\min}$ must be true.

4.3.2.3 ncv-intervals

The **set of ncv-intervals** of c is defined as:

$$\left\{ I \;\middle|\; \begin{array}{l} \text{There is an } I_1 \in SD_1^{\mathrm{NCV}(c)}, \text{ an } I_2 \in SD_2^{\mathrm{NCV}(c)}, \ldots \text{ and an} \\ I_n \in SD_n^{\mathrm{NCV}(c)}, \text{ such that } I = I_1 \cap I_2 \cap \cdots \cap I_n. \end{array} \right\}.$$

Each ncv-interval of c represents an interval of time during which all inputs of c have the logic value NCV (c) simultaneously. Thus, ncv-intervals of c induce output description intervals with value NCV (c) (CV (c) if c is inverting).

The computation is performed using the algorithm of Figure 4.12. It intersects each description interval in $SD_1^{\mathrm{NCV}(c)}$ with each description interval in $SD_2^{\mathrm{NCV}(c)}$. Those intersections which are valid description intervals are intersected with all description intervals in $SD_3^{\mathrm{NCV}(c)}$. This step is repeated until the description intervals in $SD_n^{\mathrm{NCV}(c)}$ have been processed.

```
 1 COMPUTE_NCV_INTERVALS
 2     Input:  a multiple-input gate c with n inputs
 3             c's input signal
               descriptors SD_1, SD_2, ..., SD_n
 4     Output: a set NCVINT of ncv-intervals of c.
 5 BEGIN
 6     let TMPSET be an empty set of intervals
 7     set NCVINT := SD_1^NCV(c)
 8     for i = 2 to n ; do
 9         set TMPSET := ∅
10         for each interval I ∈ SD_i^NCV(c) ; do
11             for each interval J ∈ NCVINT ; do
12                 if I and J are intersect-able ; then
13                     set TMPSET := TMPSET ∪ {(I ∩ J)}
14                 fi
15             done
16         done
17         set NCVINT := TMPSET
18     done
19     return NCVINT
20 END
```

FIGURE 4.12: ALGORITHM: COMPUTING THE SET OF NCV-INTERVALS

4.3.2.4 An example on cv- and ncv-intervals

Let c be an AND gate with two input signals described by signal descriptors SD_1 and SD_2. Let:

- $SD_1 := \{0@]-\infty; 50+\delta[, \quad 1@]50+\delta; +\infty[\}$
- $SD_2 := \{1@]30; +\infty[\}$

Then, the set of cv-intervals of c is $\{0@]-\infty; 50+\delta[\}$, namely the set of all input description intervals of c which have the logic value 0 ($= \mathrm{CV}(c)$).

The set of ncv-intervals of c is $\{1@]50+\delta; +\infty[\}$. $1@]50+\delta; +\infty[$ is the intersection of $1@]50+\delta; +\infty[$, and $1@]30; +\infty[$. Once more, the meaning of the ncv-interval $1@]50+\delta; +\infty[$ is that all input signals of c have the logic value 1 ($= \mathrm{NCV}(c)$) between time $50+\delta$ and time $+\infty$.

5

Delay Fault Simulation

In this chapter the simulation algorithm corresponding to Phase 1 on page 47 (lines 10 through 18 in Figure 3.2) is discussed. As was mentioned before, this algorithm constitutes a simulation method for delay faults which was already presented in [20] and extended in [36] and [19]. Our implementation bases on the work in [36]. However, we made some modifications. Some of these were already mentioned in the previous chapter. We will explain the other modifications later.

5.1 Illustrative example

Let us illustrate how the simulation works by applying the algorithm to the example circuit of Figure 5.1. Thus, especially some principles of the non-trivial faulty-circuit simulation (line 14 in Figure 3.2) will become clear before introducing the whole algorithm in a formal way.

The circuit under test is denoted by C. C^f denotes the circuit under the effect of an LDF f. Let the test set consist of only one test pair: $\boldsymbol{p^1 := 010/p^2 := 111}$; and let the fault list contain only one fault: **2R** (i.e. the fault delays signal 2's rising transitions by δ) . Finally, let TC be 13 ($\approx 1.2 \cdot$ PLST, cf. Figure 4.1).

5.1.1 Fault-free simulation

Fault-free simulation (line 11 in Figure 3.2) of p is performed first. The fault-free simulation works with waveforms as this permits the simulation

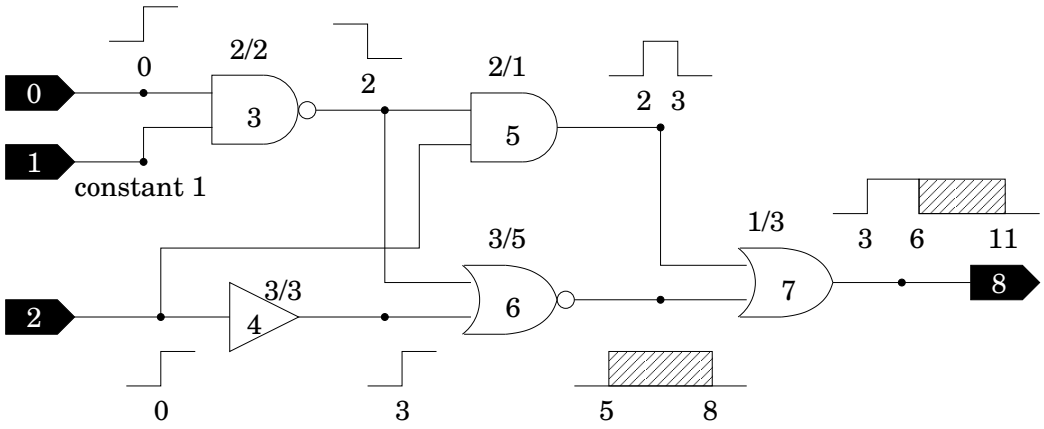


FIGURE 5.1: SIMULATION EXAMPLE: FAULT-FREE SIMULATION

of two test patterns with only one circuit pass. Additionally, operations on waveforms are easy to implement. The waveforms shown in Figure 5.1 are the result of the fault-free simulation in the example. They are processed in topological order.

For $i = 0, 1, \ldots, 7$, let W_i denote the waveform of signal i.

Signal 0: Since signal 0 is a primary input, its waveform is derived directly from the test pair. Following the conventions introduced in Section 4.1.1, W_0 starts at time $-\infty$ with logic value 0 (p^1 specifies a 0 for signal 0) and changes its logic value to 1 at time 0 (p^2 specifies a 1).

Signal 1: This is also a primary input. Both p^1 and p^2 specify a 1 for signal 1. Thus, its waveform starts at time $-\infty$ with logic value 1 and does not change any more.

Signal 2: Same as signal 0.

Signal 3: This signal is the output of the NAND gate with input signals 0 and 1. W_3 starts at time $-\infty$ with the value $\mathrm{IV}(0)$ NAND $\mathrm{IV}(1) = 0$ NAND $1 = 1$. Signal 0's rising edge at time 0 induces an output falling edge at time $0 + \mathrm{FD}(3) = 2$.

Signal 4: Gate 4 is a buffer. Its input waveform's only edge is a rising one, so the output waveform results from shifting the input waveform by $\mathrm{RD}(4) = 3$ time units to the right.

Signal 5: This signal is the output of the AND gate with input signals 2 and 3. The gate's input waveform is the 0-intersection of W_2 and W_3, i.e. $\big((-\infty, 0),\ (0, 1),\ (2, 0)\big)$. W_5 starts at time $-\infty$ with the value 0 since the gate is non-inverting. Then, the input waveform's interval $1@[0; 2[$ induces the output interval $1@[0 + \mathrm{RD}\,(5)\,; 2 + \mathrm{FD}\,(5)\,[$, i.e. a 1-interval between time 2 and time 3.

Signal 6: This signal is the output of the NOR gate with input signals 3 and 4. The gate's input waveform is the 1-intersection of W_3 and W_4, i.e. $\big((-\infty, 1),\ (2, 0),\ (3, 1)\big)$. W_6 starts at time $-\infty$ with the value 0 since the gate is inverting. The input waveform's falling edge at time 2 should induce an output rising edge at time $2 + \mathrm{RD}\,(6) = 5$. However, as an input rising edge arrives at time 3, i.e. before time 5 (see Table 4.1), it is not possible to guarantee that the output rising edge can be produced at time 5. Since it is also not possible to guarantee that the signal remains stable at logic 0, an output X-interval begins at time 5. The input rising edge at time 3 induces an output falling edge at time $3 + \mathrm{FD}\,(5) = 8$. After time 8, W_6 does not change any more.

Signal 7: This signal is the output of the OR gate with input signals 5 and 6. The gate's input waveform is the 1-intersection of W_5 and W_6, i.e. $\big((-\infty, 0),\ (2, 1),\ (3, 0),\ (5, \mathrm{X}),\ (8, 0)\big)$. W_7 starts at time $-\infty$ with the value 0 as the gate is non-inverting. The input rising edge at time 2 induces an output rising edge at time $2 + \mathrm{RD}\,(7) = 3$. The input falling edge at time 3 should induce an output falling edge at time $3 + \mathrm{FD}\,(7) = 6$. However, time 6 is later than the arrival time of the input interval $\mathrm{X}@[5; 8[$. Thus, instead of a rising edge, there is the beginning of an output X-interval at time 6. The input X-interval ends with a falling edge at time 8. This induces an output falling edge at time $8 + \mathrm{FD}\,(7) = 11$.

5.1.2 Faulty-circuit simulation

Now faulty-circuit simulation of p has to be performed on $C^{2\mathrm{R}}$ (line 14 in Figure 3.2). In the faulty-circuit simulation the behaviour of signals is described using signal descriptors. Figure 5.2 shows the signal descriptors which result from the faulty-circuit simulation. The fault-affected signals are marked with a circle. Like waveforms, signal descriptors are processed in topological order.

For $i = 0, 1, \ldots, 7$, let SD_i denote the signal descriptor of signal i.

Signals 0 and 1: These signals are not fault-affected since they are not in the fault site's output cone. Hence, the signal descriptor can be derived from the fault-free waveform. The signal descriptor of such a non-fault-affected signal s contains only one interval of the form $\mathrm{FV}(s)\,@[\mathrm{LST}(s); +\infty[$. All other waveform intervals (which end all before $\mathrm{LST}(s)$) are insignificant to fault propagation.

Signal 2: Signal 2 is an input. In the fault-free case, signal 2 has the logic value 0 between time $-\infty$ and time 0, at which the change to value 1 takes place. In the faulty circuit, the change is delayed by δ. Thus, the first interval in the signal descriptor is $0@]-\infty; 0+\delta[$. With an analogous reasoning, the second interval is $1@[0+\delta; +\infty[$.

Signal 3: Same as signal 1.

Signal 4: Signal 4 is also fault-affected. As signal 2 is signal 4's only predecessor signal, SD_4 is derived from SD_2 by translating SD_2's intervals by $\mathrm{RD}(4)$. Signal 2's description interval $0@]-\infty; 0+\delta[$ turns into signal 4's description interval $0@]-\infty; 0+\mathrm{RD}(4)+\delta[$, i.e. $0@]-\infty; 3+\delta[$. Signal 2's description interval $1@[0+\delta; +\infty[$ turns into signal 4's description interval $1@[0+\mathrm{RD}(4)+\delta; +\infty[$, i.e. $1@[3+\delta; +\infty[$.

Signal 5: Since 0 is the controlling value of cell 5, the input description interval $0@[2; +\infty[$ induces the output description interval $0@[3; +\infty[$, independently of what SD_2 looks like. This output description interval is independent of δ and guarantees that signal 5 has the "right" value ($\mathrm{FV}(5)$) at the "right" time ($\mathrm{LST}(5)$) and that the value does not change any more, no matter to what extent the faulty input signal is affected by 2R. Altogether, a signal s with source cell c having a non-fault-affected predecessor signal whose final value is $\mathrm{CV}(c)$, is always non-fault-affected; even if s is in the fault site's output cone.

Signal 6: Since 1 is the controlling value of cell 6, signal 4's description interval $1@[3+\delta; +\infty[$ is a cv-interval of cell 6 and induces the output description interval $0@[8+\delta; +\infty[$. The ncv-interval $0@[2; 3+\delta[$, which is the intersection of signal 3's $0@[2; +\infty[$ and signal 4's $0@]-\infty; 3+\delta[$ induces the output description interval $1@[2+\mathrm{RD}(6); 3+\mathrm{FD}(6)+\delta[$, i.e. $1@[5; 8+\delta[$. Now, note that this last interval can only be induced if $2+\mathrm{RD}(6) \leq 3+\delta$, i.e. if $\delta \geq 2$. The output interval only exists if

the LDF 2R has a size of at least 2 time units. This example shows that it is not always possible to derive fault descriptors from fault-free waveforms.

Signal 7: The computation of SD_7 is analogous to that of SD_6. Just note that signal 7's interval $1@[6; 11 + \delta[$, which has no natural constraint on δ (as $5 + \mathrm{RD}(7) \leq 8 + \delta$ for all $\delta \geq 0$), carries the constraint $\delta \geq 2$. That is because this interval is exclusively induced by signal 6's interval $1@[5; 8 + \delta[$ which can only exist (and thus induce other intervals) if the constraint $\delta \geq 2$ is fulfilled. That means, interval constraints are inherited, unless the heir carries an own stronger constraint.

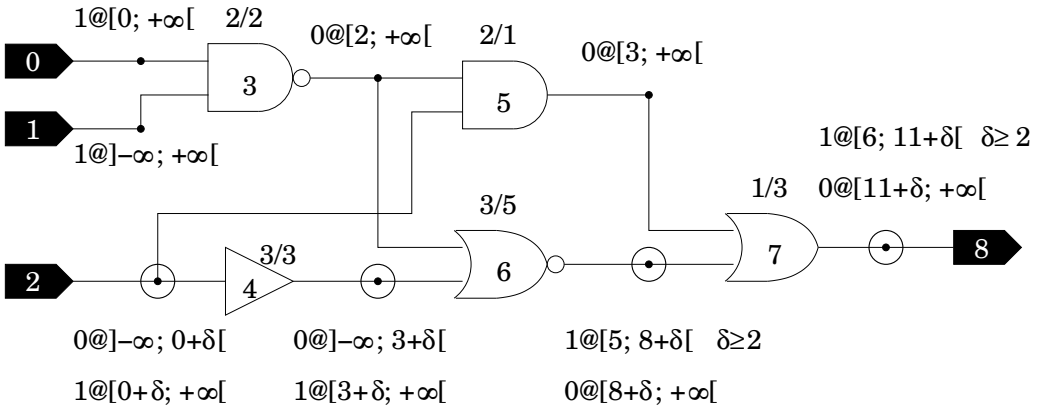


FIGURE 5.2: SIMULATION EXAMPLE: FAULTY-CIRCUIT SIMULATION

5.1.3 Computing the detection set of delay size intervals

Only fault-affected circuit outputs are considered to compute the detection intervals. In the example, there is only one (fault-affected) output in the circuit: signal 7.

In the fault-free case, the final value of signal 7 is 0. In the faulty-circuit case, there is only one fault-affected interval with the a wrong and known value, namely $1@[6; 11 + \delta[$, which is only valid if $\delta \geq 2$. If this interval ends after time TC, the value of signal 7 will be wrong at time TC and the fault

will be detected. Only $11 + \delta > \text{TC}$ is needed for detection, i.e. that $\delta > 2$. Thus, 2R is detected by the simulated test pair if its size δ is in the detection interval $[3; +\infty[$.

Finally, the set of detection intervals of delay sizes of 2R is $\big\{[3; +\infty[\big\}$.

Note that if there were more test pairs in the test set, the faulty-circuit simulation of each test pair p would yield a different set S_p of detection intervals for 2R. At the end, the final set of detection intervals is the union of all S_p.

5.2 The delay fault simulation algorithm

As was already mentioned before, the delay fault simulation algorithm presented in this section bases on that presented in [36]. However, in this work, operations of waveforms (cf. Section 4.2) are different from those in [36]; and the signal descriptors used here (cf. Section 4.3) are an extension of those in [36]. Hence, the details of fault-free and faulty-circuit simulation presented in this section constitute original work.

5.2.1 Fault-free simulation

The fault-free simulation (line 11 in Figure 3.2) consists in computing a waveform for every signal (cf. 5.1.1). This is done in topological order.

Let p be the simulated test pair. The waveform W_s of a signal s is computed depending on the type of s's source cell c.

Case 1: c is a primary or secondary input.

In this case, the waveform is defined by the test pair p. If p^1 specifies value v_1 and p^2 specifies value v_2 for signal s, W_s is set to $\big((-\infty, v_1),\ (0, v_2)\big)$.

Case 2: c is a single-input gate.

If c is a single-input gate, its input waveform equals the waveform W of its only input signal. Then, if c is not inverting, $W_s = T_{\text{RD}(c),\text{FD}(c)}(W)$ (cf. end of second paragraph of Section 4.2.3). If c is inverting, c's input waveform W must be inverted before translating it, i.e. $W_s = T_{\text{RD}(c),\text{FD}(c)}\left(\overline{W}\right)$.

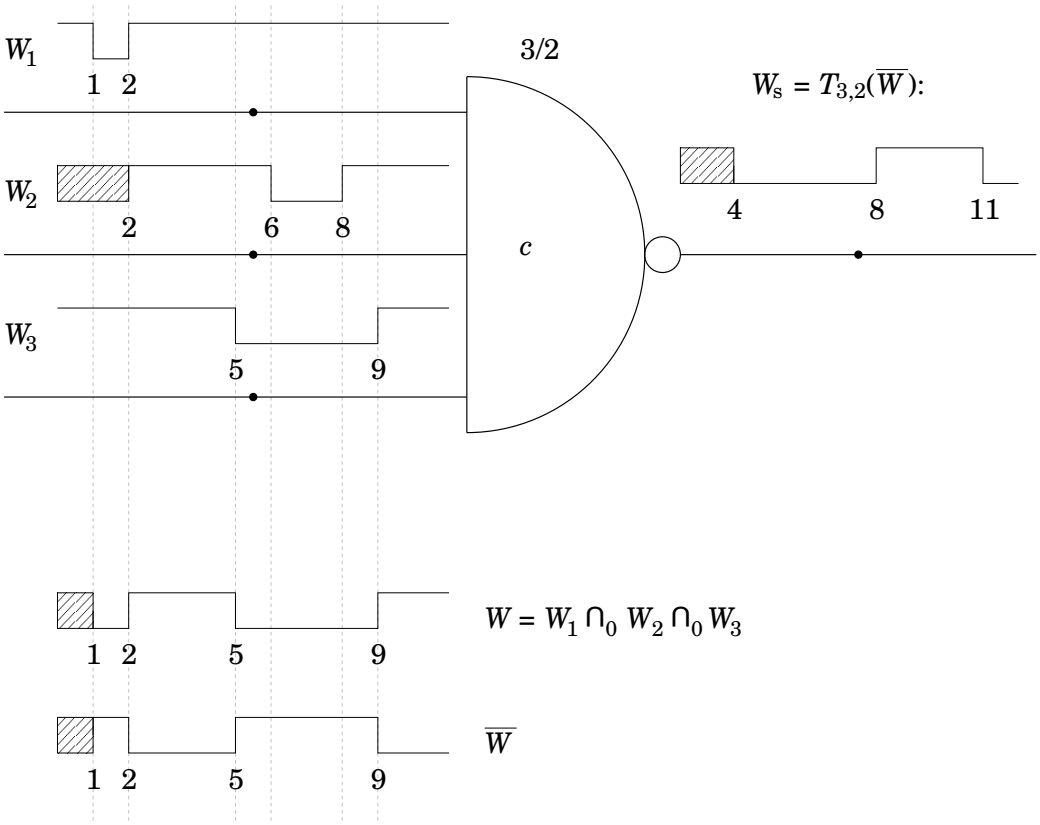


FIGURE 5.3: COMPUTING THE OUTPUT WAVEFORM OF A NAND GATE

Case 3: c is a multiple-input gate.

If c is a multiple-input gate, let $W_1, W_2, \ldots, W_n$ be the waveforms of c's n input signals. Then, according to Section 4.2.2, c's input waveform is given by

$$W := \begin{cases} W_1 \cap_1 W_2 \cap_1 \cdots \cap_1 W_n & \text{if CV}\,(c) = 1 \\ W_1 \cap_0 W_2 \cap_0 \cdots \cap_0 W_n & \text{if CV}\,(c) = 0 \end{cases}.$$

As in Case 2, $W_s = T_{\text{RD}(c),\text{FD}(c)}(W)$ if c is not inverting, and $W_s = T_{\text{RD}(c),\text{FD}(c)}\left(\overline{W}\right)$ if c is inverting.[1]

[1]Remember that this was the aim of defining input waveforms in Section 4.2.2. Once the input waveform of c is known, Case 2 and Case 3 become identical.

Figure 5.3 pictures an example. In the example, gate c is a NAND gate with three inputs s_1, s_2 and s_3, described by waveforms W_1, W_2 and W_3. The waveform shown beneath c is its input waveform. It has the value 0 whenever at least one of c's input signals have the value 0; and the value 1 whenever all input signals of c have the value 1. Until time 1, s_1 and s_3 have the logic value 1 and are thus neutral. W depends until time 1 on s_2. Since the value of s_2 is unknown until time 2, W is at X until time 1.

Since c is inverting, its input waveform W must be inverted. The inverse waveform $\overline{W}$ is shown underneath W. Finally, $\overline{W}$'s rising edges are delayed by $\mathrm{RD}(c) = 3$ time units and $\overline{W}$'s falling edges are delayed by $\mathrm{FD}(c) = 2$ time units to get c's output waveform W_s. Pulses that are too short, like $1@[1;2[$ are filtered out in the output.

5.2.2 Faulty-circuit simulation

Let f be an LDF of size δ. The faulty-circuit simulation of a test pair p on the faulty version C^f of the circuit (line 14 in Figure 3.2) consists in computing a signal descriptor for every signal (cf. 5.1.2). This is done in topological order.

When computing the signal descriptor SD_s of a signal s, it is necessary to distinguish several cases. In all cases, let c be the source cell of s. Furthermore, we define that $\pm\infty + k = \pm\infty$, where k is $\mathrm{RD}(c)$, $\mathrm{FD}(c)$, δ, $\mathrm{RD}(c) + \delta$ or $\mathrm{FD}(c) + \delta$.

Case 1: s is not in the fault site's output cone (e.g. signal 0 in Section 5.1.2).

In this case, s is non-fault-affected. Its signal descriptor contains only one description interval. SD_s is set to $\{\mathrm{FV}(s)\,@[\mathrm{LST}(s)\,;+\infty[\}$ (cf. Section 4.3.1).

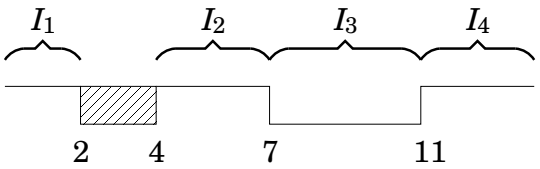


FIGURE 5.4: COMPUTING THE FAULT SITE'S SIGNAL DESCRIPTORS

Case 2: s is the fault site (e.g. signal 2 in Section 5.1.2).

Let $W_s := \big((t_1, v_1), (t_2, v_2), \ldots, (t_N, v_N)\big)$ be the fault-free waveform of s and let $t_{N+1} := +\infty$.

Case 2.1: f is slow-to-rise.

In the faulty-circuit case, it is necessary to additionally delay all rising edges by δ. More precisely, one begins with an empty signal descriptor. Then, for each $i = 1, \ldots, N$, if v_i is 0, the description interval $0@[t_i; t_{i+1} + \delta[$ is inserted into SD_s; if v_i is 1, the description interval $1@[t_i + \delta; t_{i+1}[$ is inserted into SD_s; if v_i is X, nothing is done.

Case 2.2: f is slow-to-fall.

In the faulty-circuit case, we just need to additionally delay all falling edges by δ. More precisely, one begins with an empty signal descriptor. Then, for each $i = 1, \ldots, N$, if v_i is 1, the description interval $1@[t_i; t_{i+1} + \delta[$ is inserted into SD_s; if v_i is 0, the description interval $0@[t_i + \delta; t_{i+1}[$ is inserted into SD_s; if v_i is X, nothing is done.

For example, let $W_s := \big((-\infty, 1), (2, X), (4, 1), (7, 0), (11, 1)\big)$ (the waveform's graphical representation is depicted in Figure 5.4), and let f be slow-to rise. The corresponding signal descriptor is:

$$SD_s = \left\{ \begin{array}{ll} 1@]-\infty; 2[& (I_1) \\ 1@]4+\delta; 7[& (I_2) \\ 0@]7; 11+\delta[& (I_3) \\ 1@]11+\delta; +\infty[& (I_4) \end{array} \right\}.$$

In [36], only the first portion of the waveform between time $-\infty$ and $\mathrm{EAT}(s)$ and the last portion between $\mathrm{LST}(s)$ and time $-\infty$ are turned into description intervals. In this case, this methodology would yield the signal descriptor

$$\left\{ \begin{array}{ll} 1@]-\infty; 2[& (I_1) \\ 1@]11+\delta; +\infty[& (I_4) \end{array} \right\}.$$

Turning all determinate waveform intervals into description intervals, as is done here, is obviously more exact. In [19], this concept is also used; however, both falling and rising edges are delayed by δ, as [19] works with GDFs which are slow-to-rise and slow-to-fall at the same time.

Case 3: s is in the fault site's output cone.

Three sub-cases need to be handled. Note that, if s is in the fault site's output cone, c must be a gate.

Case 3.1: c is a single-input gate (e.g. signal 4 in Section 5.1.2).

If c is a single-input gate, s has exactly one predecessor signal s'. One constructs the signal descriptor SD_s starting with an empty one. For each description interval I in $SD_{s'}$, a new description interval J is inserted into SD_s according to the rules listed in Table 5.1 (cf. also Table 4.1).

In the table, v stands for a determinate logic value (0 or 1). $\overline{v}$ denotes the logic inverse of v. $D(v)$ denotes the delay that c needs to produce the output v, i.e. $D(0) = \mathrm{FD}\,(c)$ and $D(1) = \mathrm{RD}\,(c)$.

The table is read in the following way. Take, for example, Rule 2. An input description interval of the form $I : 1@[x+\delta; y+\delta[\ \ K_I$ induces the output description interval $J : 1@[x+\mathrm{RD}\,(c)+\delta; y+\mathrm{FD}\,(c)+\delta[\ \ K_I$ if c is not inverting; however, only if the additional condition $x+\mathrm{RD}\,(c) \leq y$ is met. This additional condition is imposed by the used delay model. If the additional condition is not met, no output description interval is induced at all. The constraint on the output description interval J is denoted by K_I. This means, that J inherits I's constraint.

In Rules 3, 4, 5, 6, 9, 10, 11 and 12, there is no additional condition. Instead, the output description interval's constraint K_J must be adjusted such that the used delay model is respected:

Rule 3: The first constraint on J is $K_1 : x + \delta < y$ as the left limit of a time interval must be earlier than its right limit. The second constraint $K_2 : x + D(v) + \delta \leq y$ derives from the delay model. The third constraint on J is the inherited constraint K_I. Finally, $K_J = K_1 \wedge K_2 \wedge K_I$, where the intersection $K_\alpha \wedge K_\beta$ of two constraints $K_\alpha :\ \alpha_1 \leq \delta \leq \alpha_2$ and $K_\beta :\ \beta_1 \leq \delta \leq \beta_2$ is defined as $\max\{\alpha_1, \beta_1\} \leq \delta \leq \min\{\alpha_2, \beta_2\}$.

Rule 5: The first constraint on J is $K_1 : x < y + \delta$ as the left limit of a time interval must be earlier than its right limit. The second constraint $K_2 : x + D(v) \leq y + \delta$ derives from the delay model. The third constraint on J is the inherited constraint K_I. Finally, $K_J = K_1 \wedge K_2 \wedge K_I$.

Rule	I	c inverting	additional condition	J
1	$v@[x;y[\ K_I$	no	$x + D(v) \leq\mid y$	$v@[x + D(v); y + D(\overline{v})[\ K_I$
2	$v@[x + \delta; y + \delta[\ K_I$	no	$x + D(v) \leq\mid y$	$v@[x + D(v) + \delta; y + D(\overline{v}) + \delta[\ K_I$
3	$v@[x + \delta; y[\ K_I$	no	-	$v@[x + D(v) + \delta; y + D(\overline{v})[\ K_J$
4	$v@[x + \delta; +\infty[\ K_I$	no	-	$v@[x + D(v) + \delta; +\infty[\ K_J$
5	$v@[x; y + \delta[\ K_I$	no	-	$v@[x + D(v); y + D(\overline{v}) + \delta[\ K_J$
6	$v@]-\infty; y + \delta[\ K_I$	no	-	$v@]-\infty; y + D(\overline{v}) + \delta[\ K_J$
7	$v@[x;y[\ K_I$	yes	$x + D(\overline{v}) \leq\mid y$	$\overline{v}@[x + D(\overline{v}); y + D(v)[\ K_I$
8	$v@[x + \delta; y + \delta[\ K_I$	yes	$x + D(\overline{v}) \leq\mid y$	$\overline{v}@[x + D(\overline{v}) + \delta; y + D(v) + \delta[\ K_I$
9	$v@[x + \delta; y[\ K_I$	yes	-	$\overline{v}@[x + D(\overline{v}) + \delta; y + D(v)[\ K_J$
10	$v@[x + \delta; +\infty[\ K_I$	yes	-	$\overline{v}@[x + D(\overline{v}) + \delta; +\infty[\ K_J$
11	$v@[x; y + \delta[\ K_I$	yes	-	$\overline{v}@[x + D(\overline{v}); y + D(v) + \delta[\ K_J$
12	$v@]-\infty; y + \delta[\ K_I$	yes	-	$\overline{v}@]-\infty; y + D(v) + \delta[\ K_J$

Table 5.1: Rules for the creation of output description intervals

Rule 9: The first constraint on J is $K_1 : x + \delta < y$ as the left limit of a time interval must be earlier than its right limit. The second constraint $K_2 : x + D(\overline{v}) + \delta \leq y$ derives from the delay model. The third constraint on J is the inherited constraint K_I. Finally, $K_J = K_1 \wedge K_2 \wedge K_I$.

Rule 11: The first constraint on J is $K_1 : x < y + \delta$ as the left limit of a time interval must be earlier than its right limit. The second constraint $K_2 : x + D(\overline{v}) \leq y + \delta$ derives from the delay model. The third constraint on J is the inherited constraint K_I. Finally, $K_J = K_1 \wedge K_2 \wedge K_I$.

Rules 4, 6, 10 and 12: Here J inherits I's constraint. I.e. $K_J = K_I$.

Case 3.2: c is a multiple-input cell and at least one of its input signals s' is non-fault-affected and $\mathrm{FV}(s') = \mathrm{CV}(c)$.

As was already explained in Section 5.1.2 (see signal 5), s is non-fault-affected although it is in the fault site's output cone. Thus, its signal descriptor contains only one description interval. $SD_s = \{\mathrm{FV}(s)\,@[\mathrm{LST} + D(\mathrm{FV}(s)); +\infty[\}$, where LST is defined as

$$\mathrm{LST} := \min\left\{\mathrm{LST}(s') \,\middle|\, \begin{array}{l} s' \text{ is a non-fault-affected input signal} \\ \text{of } c \text{ and } \mathrm{FV}(s') = \mathrm{CV}(c). \end{array}\right\}.$$

Case 3.3: c is a multiple-input cell and either none of its non-fault-affected input signals has $\mathrm{CV}(c)$ as fault-free final value, or all its input signals are fault-affected (e.g. signals 6 and 7 in Section 5.1.2).

We saw in Section 4.3.2 that the part of c's overall input which is relevant to faulty-circuit simulation, is described by the set of cv-intervals of c and by the set of ncv-intervals of c. In this case, one begins with an empty signal descriptor SD_s. Then, for each cv-interval I an output description interval J is inserted into SD_s. The same is done with all ncv-intervals. J is computed according to the rules listed in Table 5.1 (cf. also Table 4.1).

5.2.3 Computing the detection set of delay size intervals

Once there is a signal descriptor for each output, it is possible to compute the detection set of delay size intervals of f (line 15 in Figure 3.2).

One starts with an empty set D of detection intervals. Then, for each primary or secondary output s with a fault-affected signal descriptor SD_s (those signals are all in the fault site's output cone), we define the set SD_s^{bad} as the set of those description intervals during which signal s has the wrong logic value (i.e. $\overline{\text{FV}(s)}$, not X). Finally, for each description interval I in $\bigcup_{s \text{ primary or secondary output}} SD_s^{\text{bad}}$, an interval $DET_I := [\delta_{\min}; \delta_{\max}]$ of delay fault sizes is inserted into D. The meaning of DET_I is that f is detected by the current test pair p if f's size δ is in DET_I. Once D has been constructed, the last step consists in checking for non-empty intersections of intervals in D and replacing all such intersecting intervals by their unions, in order to produce a set containing only disjoint intervals.

Given a description interval I, the corresponding detection interval of fault sizes DET_I is computed according to the following rules:

Case 1: I has the form $\overline{\text{FV}(s)}@[left_I; y[\ \ K_I$.

In this case, I definitely ends before time TC, so the detection interval DET_I is empty.

Case 2: I is of the form $\overline{\text{FV}(s)}@[x; y+\delta[\ \ \delta_{\min}^I \leq \delta \leq \delta_{\max}^I$ or of the form $\overline{\text{FV}(s)}@]-\infty; y+\delta[\ \ \delta_{\min}^I \leq \delta \leq \delta_{\max}^I$.

This case is illustrated in Figures 5.5 through 5.8. I is represented by a bold line starting at time x. I's right limit lies between $y+\delta_{\min}^I$ and $y+\delta_{\max}^I$, depending on δ. The signal has a wrong (known) value during the interval I. Thus, the fault is detected if $x \leq \text{TC} < y+\delta$, i.e. if the signal has a wrong value at time TC.

If $\text{TC} < y+\delta_{\min}^I$ (Figure 5.5), I ends after time TC for all valid δ values. Thus, f is detected for all $\delta \in [\delta_{\min}^I; \delta_{\max}^I]$. DET_I is set to $[\delta_{\min}^I; \delta_{\max}^I]$.

Else, if $\text{TC} = y+\delta_{\min}^I$ (Figure 5.6), I ends right before time TC if δ equals $\delta_{\min}^I$. δ must be greater than $\delta_{\min}^I$ in order I to end after time TC. Since we work with integer values, DET_I is set to $[\delta_{\min}^I + 1; \delta_{\max}^I]$.

Else, if $\text{TC} < y+\delta_{\max}^I$ (Figure 5.7), DET_I is set to $[\text{TC} - y + 1; \delta_{\max}^I]$.

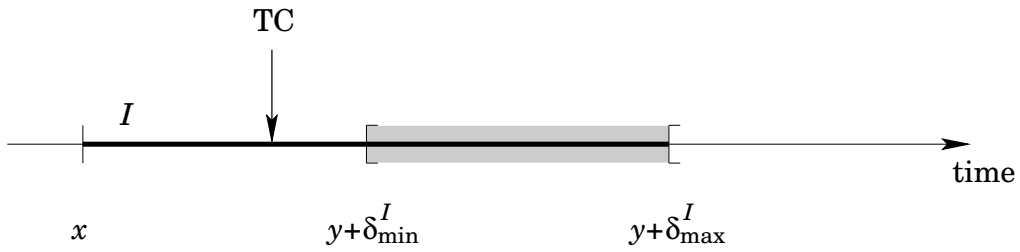


FIGURE 5.5: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

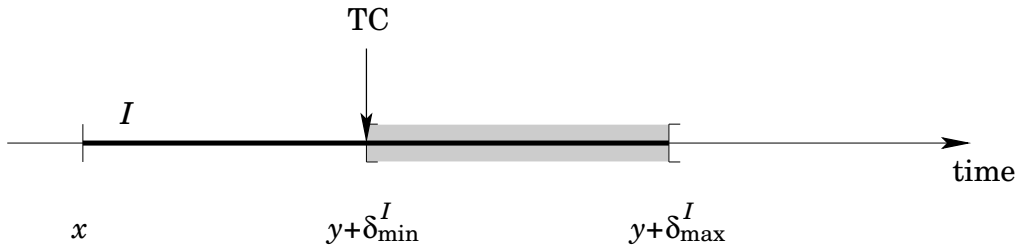


FIGURE 5.6: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

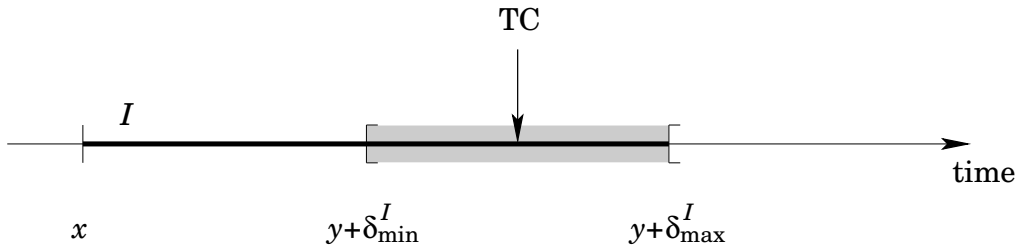


FIGURE 5.7: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

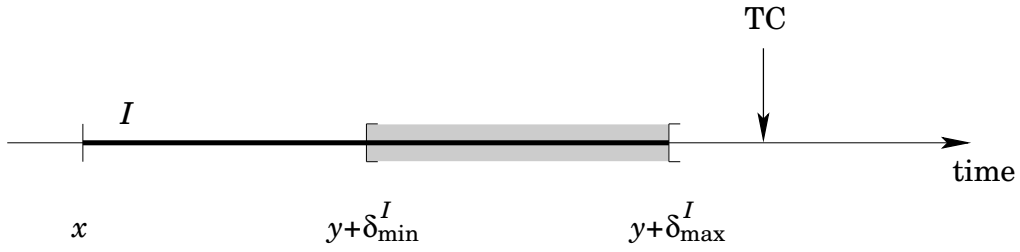


FIGURE 5.8: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

Else, i.e. if $y + \delta^I_{\max} \leq$ TC (Figure 5.8), I ends before TC for all valid values of δ. Hence, DET_I is empty.

Case 3: I has the form $\overline{\mathrm{FV}\,(s)}@[x+\delta; y+\delta[\ \ \delta^I_{\min} \leq \delta \leq \delta^I_{\max}$.

This case is illustrated in Figures 5.9 through 5.14. I is represented by a bold line. I's left limit lies between $x + \delta^I_{\min}$ and $x + \delta^I_{\max}$, depending on δ. I's right limit lies between $y + \delta^I_{\min}$ and $y + \delta^I_{\max}$, depending on δ. The signal has the wrong output value during the interval I. Thus, the fault is detected if $x + \delta \leq \mathrm{TC} < y + \delta$, i.e. if the signal has a wrong value at time TC.

As I has two variable limits, $\delta_{\min}$ and $\delta_{\max}$ are computed separately. $\delta_{\max}$ is computed first:

If TC $< x + \delta^I_{\min}$ (Figure 5.9), I begins after time TC for all valid δ values. The fault remains undetected. Hence, DET_I is empty.

Else, if TC $< x + \delta^I_{\max}$ (Figure 5.10), $\delta_{\max}$ is set to TC $- x$. For δ's larger than TC $- x$, I begins after time TC and the fault remains undetected.

Else (Figure 5.11), I begins before time TC for all valid δ values. Hence, $\delta_{\max}$ is set to $\delta^I_{\max}$.

Then, $\delta_{\min}$ is computed:

If TC $< y + \delta^I_{\min}$ (Figure 5.11), I ends after time TC for all valid δ values. Thus, f is detected for all $\delta \in [\delta^I_{\min}; \delta^I_{\max}]$. $\delta_{\min}$ is set to $\delta^I_{\min}$.

Else, if TC $= y + \delta^I_{\min}$ (Figure 5.12), I ends right before time TC if δ equals $\delta^I_{\min}$. δ must be greater than $\delta^I_{\min}$ in order I to end after time TC. Since we work with integer values, $\delta_{\min}$ is set to $\delta^I_{\min} + 1$.

Else, if TC $< y + \delta^I_{\max}$ (Figure 5.13), $\delta_{\min}$ is set to TC $- y + 1$.

Else, i.e. if $y + \delta^I_{\max} \leq$ TC (Figure 5.14), I ends before TC for all valid values of δ. Hence, DET_I is empty.

Finally, DET_I is set to $[\delta_{\min}, \delta_{\max}]$ if $\delta_{\min} \leq \delta_{\max}$, else $DET_I := \emptyset$.

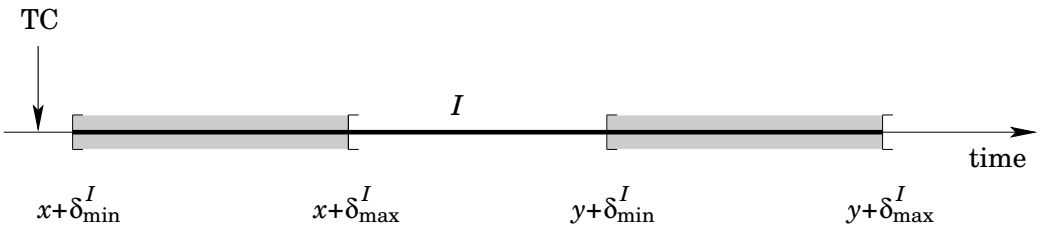


FIGURE 5.9: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

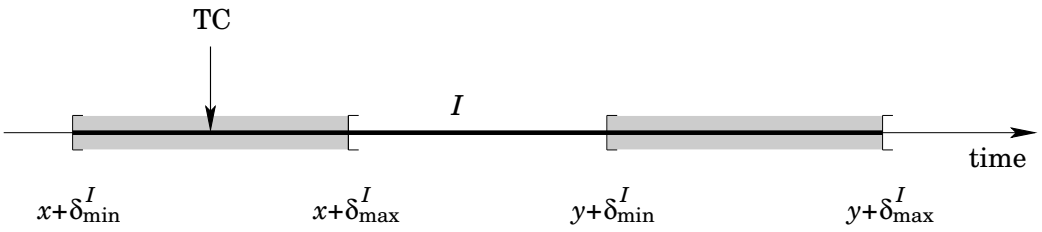


FIGURE 5.10: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

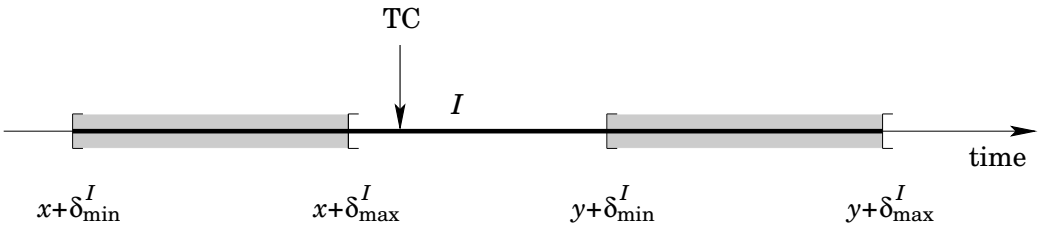


FIGURE 5.11: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

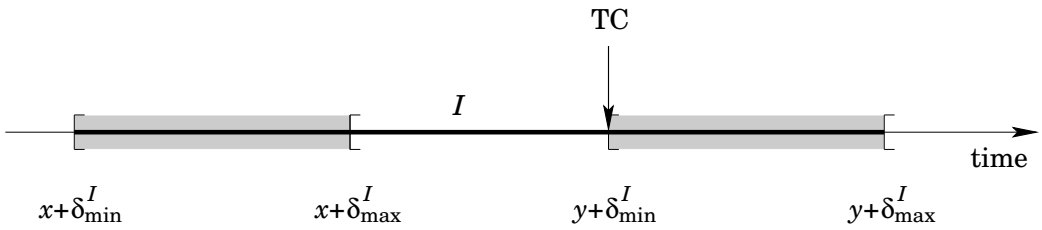


FIGURE 5.12: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

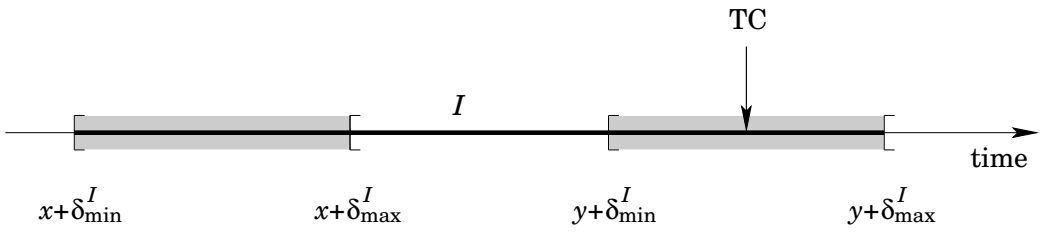


FIGURE 5.13: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

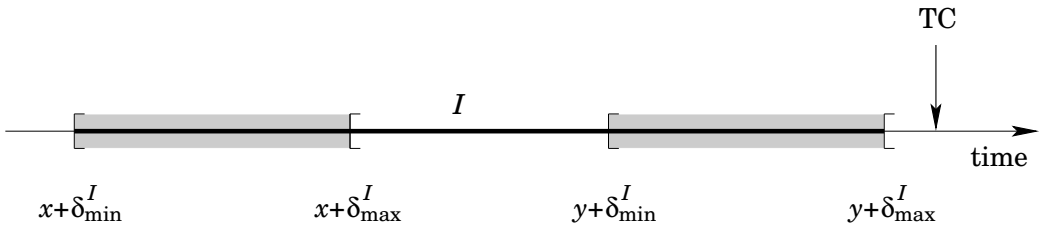


FIGURE 5.14: COMPUTING THE DETECTION INTERVALS OF DELAY SIZES

6

Fault Coverage

Measuring the fault coverage that the simulated test set achieves corresponds to the RO-simulator's Phase 3 (see page 47). In this chapter the two metrics which the RO-simulator uses to compute fault coverage are presented.

6.1 Introduction

In Section 2.4, the definition of four fault coverage metrics for the simulation of resistive bridging faults was introduced. Those fault coverage definitions reflected the fact that a resistive bridging fault g may be detected by a given test set only if its resistance is in C-ADI (g), the union of one or more disjoint ranges of resistances. The four fault coverage metric definitions were the following:

- P-FC $(g) := \dfrac{\int\limits_{\text{C-ADI}(g)} \rho(r)dr}{\int\limits_{0}^{+\infty} \rho(r)dr} \cdot 100\%$
- G-FC $(g) := \dfrac{\int\limits_{\text{C-ADI}(g)} \rho(r)dr}{\int\limits_{\text{G-ADI}(g)} \rho(r)dr} \cdot 100\%$
- E-FC $(g) := \dfrac{\int\limits_{\text{C-ADI}(g)} \rho(r)dr}{\int\limits_{0}^{R_{\max}} \rho(r)dr} \cdot 100\%$

- O-FC $(g) := \begin{cases} 0\% & \text{if C-ADI}\,(g) = \emptyset \\ 100\% & \text{else} \end{cases}$

where $\rho(r)$ is the probability density function of the bridge resistance r.

Since the detection of resistive opens, as in the case of resistive bridging faults, depends on the open's resistance; and since in both cases the resistance is a stochastic parameter, these fault coverage metrics are also applicable to our work with resistive opens.

6.2 Fault coverage metrics for resistive opens

Throughout the rest of this chapter, let F be the fault list and P be the set of test pairs simulated in Phase 1 of the RO-simulator. Let $f \in F$ be an LDF of size δ modelling a resistive open with resistance r. Let C-ADI (f) denote the detection set of resistance ranges computed in Phase 2 of the RO-simulator.

The drawback of P-FC is that it is too pessimistic as it assumes that, for each resistance $r' \in [0, +\infty[$, a test pair p exists which detects f if $r = r'$.

G-FC is the most accurate metric as it takes into account the set G-ADI $(f) = \{r' \,|\text{A test pair } p \text{ exists which detects } f \text{ if } r = r'\}$. However, computing this set is infeasible in most practical cases.

Thus, neither P-FC nor G-FC are used by the RO-simulator.

The first metric the RO-simulator uses is O-FC, exactly as it is defined above.

The second metric the RO-simulator uses is simply called **`fault coverage`** (FC). In order to define it, the same principle is used which E-FC uses to avoid the pessimism of P-FC without being as computationally complex as G-FC: the denominator's integral goes over only one resistance range with finite limits. Before formulating an equation for FC, some definitions have to be introduced:

$D_{\min}$: This is the smallest size which δ may have such that f can be detected by any test pair. In this work $D_{\min}$ is set to TC − PLST.

$D_{\max}$: This is a number that is large enough as to assume that a fault having that size will be detected in any case. In this work $D_{\max}$ is set to $5 \cdot$ PLST.

R_{min}: This is the size the open's resistance must have in order to produce a delay fault of size $D_{\min}$. Assuming that the delay-to-resistance mapping is monotonically rising, $R_{\min}$ is the smallest size the open's resistance must have such that it can be detected by any test pair.

R_{max}: This is the size the open's resistance must have in order to produce a delay fault of size $D_{\max}$.

RC-ADI (f): RC-ADI stands for "restricted C-ADI" and is defined for a fault f as[1]

$$\text{RC-ADI}(f) := \left\{ I \setminus]R_{\max}, +\infty[\;\middle|\; I \in \text{C-ADI}(f) \right\}.$$

That means that RC-ADI of a fault is the same as its C-ADI, but all resistances greater than $R_{\max}$ are ignored.

Then, FC is defined as

$$\text{FC}(f) := \frac{\int\limits_{\text{RC-ADI}(f)} \rho(r)dr}{\int\limits_{R_{\min}}^{R_{\max}} \rho(r)dr} \cdot 100\%.$$

ρ is the probability density function of the open resistance. Ideally, ρ should be obtained from the technology data. As these data are not available to us at the moment, we rely on [43], where the authors report a uniform distribution for resistances of up to 10 MΩ. Hence, ρ is defined as

$$\rho(r) := 1 \text{ for all } r \in [0; 10\ \text{M}\Omega].$$

Note that, although ρ's domain is not (as usual) $\mathbb{R}^+$, FC is well-defined, since $R_{\max} \leq 10$ MΩ in all cases (if an open's resistance is greater than 10 MΩ, the open is no longer considered a weak open [43]).

6.3 An example

The following example illustrates how to compute the fault coverage for a fault f. For simplicity, let us assume that the delay-to-resistance mapping maps n picoseconds to n milli-Ohm for all $n \in \mathbb{N}$.

[1] Remember that C-ADI was defined in the context of the RO-simulator as a set of ranges, and not like in [40] as a union of ranges.

Let PLST be 150 ns, let TC be 180 ns (TC $= 1.2 \cdot$ PLST). Then, $D_{\min} =$ TC $-$ PLST $=$ 30 ns and $R_{\min}$ is 30 Ω. Let $D_{\max}$ be 750 ns ($D_{\max} = 5 \cdot$ PLST). Then, $R_{\max}$ is 750 Ω.

Let the C-ADI of a fault f be the set of resistance ranges $\big\{[200\ \Omega; 310\ \Omega],\ [500\ \Omega; +\infty[\big\}$. Then, RC-ADI of f is $\big\{[200\ \Omega; 310\ \Omega],\ [500\ \Omega; 750\ \Omega]\big\}$. Finally,

$$\begin{aligned}
\mathrm{FC}\,(f) &= \frac{\int\limits_{\text{RC-ADI}(f)} \rho(r)dr}{\int\limits_{R_{\min}}^{R_{\max}} \rho(r)dr} \cdot 100\% \\
&= \frac{\int\limits_{200}^{310} 1\,dr + \int\limits_{500}^{750} 1\,dr}{\int\limits_{30}^{750} 1\,dr} \cdot 100\% \\
&= \frac{(310-200)+(750-500)}{750-30} \cdot 100\% \\
&= 50\%\ .
\end{aligned}$$

Figure 6.1 illustrates the integral $\int_{\text{RC-ADI}(f)} \rho(r)dr$ (the union of all dark shaded regions) and the integral $\int_{R_{\min}}^{R_{\max}} \rho(r)dr$ (the union of all dark and light shaded regions). It can be seen, that FC (f) relates the fraction of those ranges in which the open is detected to the observed "maximal" range from $R_{\min}$ to $R_{\max}$.

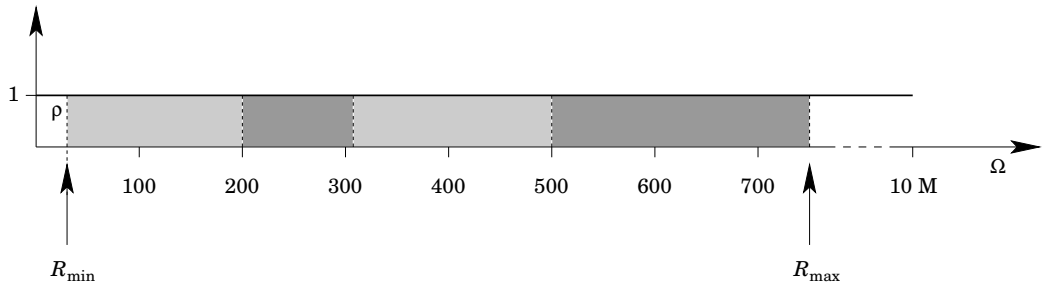


Figure 6.1: Example: computing fault coverage

6.4 Overall fault coverage

The fault coverage for the fault list F is defined as the average over all faults in the list (cf. Section 2.4):

- $\text{O-FC}(F) := \dfrac{\sum_{f \in F} \text{O-FC}(f)}{|F|}$
- $\text{FC}(F) := \dfrac{\sum_{f \in F} \text{FC}(f)}{|F|}$

If the probability that each fault occurs in the circuit (i.e. the probability that the open which the fault models is present) is known, a more exact fault coverage metric is obtained by weighting the average of the single fault coverages by those probabilities. However, as these data are also not known, the above definition is used by the RO-simulator.

7

Experimental Results

The RO-simulation was applied to ISCAS 85 (combinational) and ISCAS 89 (sequential) benchmark circuits.

In all cases, one test set composed of 100 randomly generated test pairs was simulated. In all cases the fault list included all LDFs, i.e. iR and iF for each signal i in the circuit.

Table 7.1 lists the rising delay and the falling delay times (in picoseconds) of each gate type. These fixed values were used for the simulation of all benchmark circuits. In the table, ANDN denotes N-input gates of type AND (analogously for gates of type OR, NAND or NOR). Currently, we do not dispose of timing information regarding the cell library for the used ISCAS benchmarks. Hence, all delay times in the table were chosen at random such as to be between 1 and 7 nanoseconds. The simulator's implementation can read these delay data from file. Thus, there is no difficulty in using a different set of delay times if these become available.

All measurements were performed on an Intel® XEON™ 2.00GHz machine with 2GB RAM running Debian Linux.

Tables 7.2 and 7.3 list the benchmark circuits' properties and an overview of all obtained results. The meaning of the column labels is the following:

Signals: This is the number of signals in the circuit.

Faults: This is the number of simulated faults.

Depth: This is the circuit's depth. If one wants to assess the worst-case time which the simulator needs to perform fault-free and faulty-circuit simulation, the circuit's depth may be a better reference measure than the circuit's size expressed as number of signals.

Gate type	**Rising delay**	**Falling delay**	**Gate type**	**Rising delay**	**Falling delay**
BUF	3544	1551	INV	2932	5362
AND2	4282	1811	NAND2	1509	5836
AND3	3002	4302	NAND3	3307	1468
AND4	3710	2345	NAND4	5048	4249
AND5	4374	4460	NAND5	2002	3980
AND6	6896	2517	NAND6	5689	4070
AND7	3939	4130	NAND7	1136	5756
AND8	5876	1618	NAND8	4058	3174
AND9	4671	3666	NAND9	3182	3387
AND10	1690	1304	NAND10	3316	6173
AND11	1323	3196	NAND11	5641	1831
OR2	4175	1933	NOR2	2977	1482
OR3	3536	2552	NOR3	3361	6149
OR4	3451	4293	NOR4	1567	2159
OR5	5444	4133	NOR5	6713	6592
OR6	5451	5013	NOR6	6727	2667
OR7	4695	5631	NOR7	5824	2659
OR8	2475	4506	NOR8	2870	5153
OR9	3525	1367	NOR9	4559	2075
OR10	4123	1996	NOR10	6065	4747
OR11	5529	1997	NOR11	1894	3737

Table 7.1: Rising and falling delay times of each gate type

PLST: This is the guaranteed time in picoseconds after which all paths in the fault-free circuit have stabilised after modifying any primary or secondary input.

TC: This is the clock sampling time in picoseconds. All experiments were run setting TC to 1.2 times PLST.

O-FC: The measured optimistic fault coverage, i.e. the percentage of faults that were detected for at least one resistance.

FC: The measured fault coverage as defined in Section 6.4.

Time: This is the time in seconds which was needed to perform the simulation of all 100 test pairs on the machine mentioned above.

Memory peak: This is the highest amount of RAM (in MB) which the simulator used at any time during the process.

The diagrams in Figures 7.1 and 7.2 show how the needed time and memory depend on the number of simulated faults (in this case, also on the circuit's number of signals) and on the circuit's depth, respectively. The amount of needed time and memory clearly rises only linearly as the number of treated faults rises. Meanwhile, there is no identifiable relation between circuit depth and use of computing resources.

The diagrams in Figures 7.3 and 7.4 show how the fault coverage depends on the number of simulated faults and on the circuit's depth, respectively. The fault coverage values are uniformly distributed around their respective average values and the dispersion is not large.

Here, it was observed, that a fault coverage of slightly more than 50% was achieved with only 100 randomly generated test pairs. In order to learn more about this topic a further experiment was performed on c5315 (second largest combinational circuit) and on s35932 (third largest sequential circuit). Two hundred random test pairs were simulated for each one of these two circuits. The achieved fault coverage was measured after the simulation of each test pattern, in order to see how fault coverage depends on the number of random test patterns. The diagrams in Figures 7.5 and 7.6 report the obtained results. In both cases, only about thirty to forty test pairs were needed to achieve a fault coverage of more than 60%. Furthermore, hundred test patterns were enough to achieve a fault coverage of more than 70% (80% according to the optimistic metric).

Circuit	Signals	Faults	Depth	PLST	TC	O-FC (%)	FC (%)	Time	Mem. peak
c0017	11	22	5	17508	21009	100.00	90.69	1	0.06
c0095	32	64	6	20820	24983	100.00	94.85	1	0.10
c0880	443	886	26	116524	139828	90.86	78.47	14	0.94
c1908	913	1826	42	209834	251800	66.05	56.17	40	1.77
c3540	1719	3438	49	231594	277912	64.55	55.15	166	4.17
c5315	2485	4970	51	256032	307238	91.57	76.55	148	5.68
c7552	3719	7438	45	229785	275741	85.67	72.45	194	8.76
s00027	17	34	8	27994	33592	97.06	87.25	1	0.07
s00208	122	244	13	50849	61018	68.04	59.23	4	0.27
s00298	136	272	11	38996	46795	79.78	68.95	4	0.30
s00386	172	344	13	51161	61393	66.57	60.33	5	0.36
s499	175	350	14	61399	73678	14.29	12.50	5	0.36
s00382	182	364	11	47584	57100	74.73	65.33	7	0.39
s00344	184	368	22	91337	109604	89.95	76.96	5	0.40
s00349	185	370	22	91337	109604	89.73	76.78	6	0.40
s00400	186	372	11	47584	57100	73.39	64.09	6	0.40
s00444	205	410	13	58782	70538	64.15	55.42	6	0.43
s00526	218	436	11	38996	46795	56.43	48.84	7	0.46
s00510	236	472	14	49028	58833	80.51	71.16	7	0.49
s00420	252	504	15	59199	71038	33.14	29.20	8	0.51
s00832	310	620	12	49878	59853	43.23	37.54	10	0.60
s00820	312	624	12	49878	59853	42.95	37.32	10	0.62
s635	320	640	129	607130	728555	31.41	25.12	9	0.62

Table 7.2: Experimental results: tabular overview

Table 7.3: Experimental results: tabular overview

Circuit	Signals	Faults	Depth	PLST	TC	O-FC (%)	FC (%)	Time	Mem. peak
s00641	433	866	76	376483	451779	81.53	67.93	13	0.90
s00953	440	880	18	72860	87431	51.60	44.24	14	0.86
s00713	447	894	76	380275	456329	78.08	64.99	14	0.94
s00838	512	1024	19	75899	91078	13.77	11.98	15	0.97
s938	512	1024	19	75899	91078	16.80	14.71	16	0.97
s01238	540	1080	24	104838	125805	59.36	49.99	16	1.04
s01196	561	1122	26	117209	140650	57.40	48.11	17	1.06
s01494	661	1322	19	77886	93463	71.94	64.03	20	1.22
s01488	667	1334	19	77886	93463	71.97	64.10	21	1.26
s01423	748	1496	61	261925	314309	72.00	58.24	24	1.52
s1512	866	1732	32	129628	155553	64.38	53.81	27	1.71
s3271	1714	3428	30	140204	168244	88.80	75.41	293	6.63
s3384	1911	3822	62	260511	312613	92.26	76.06	65	3.99
s3330	1961	3922	31	136175	163409	64.03	54.38	61	3.70
s05378	2993	5986	27	125531	150637	63.29	55.75	97	5.71
s09234	5844	11688	60	281200	337439	43.61	36.18	264	10.72
s13207	8651	17302	61	304632	365558	57.06	46.67	382	16.13
s15850	10383	20766	84	422286	506743	62.21	50.90	485	18.96
s35932	17828	35656	31	153305	183965	89.47	78.37	733	36.03
s38584	20717	41434	58	294145	352973	73.36	60.09	1051	38.97
s38417	23843	47686	49	229613	275535	77.88	65.34	992	44.21
AVG:	**2608**	**5216**	**33**	**150037**	**180044**	**67.16**	**57.76**	**120**	**5.13**

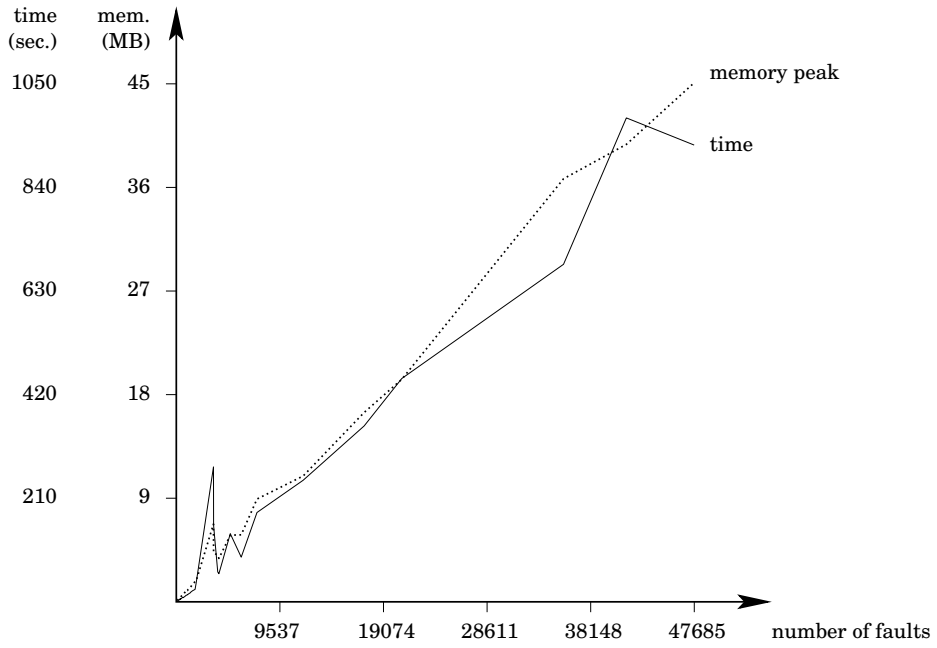


FIGURE 7.1: USE OF RESOURCES DEPENDING ON NUMBER OF FAULTS

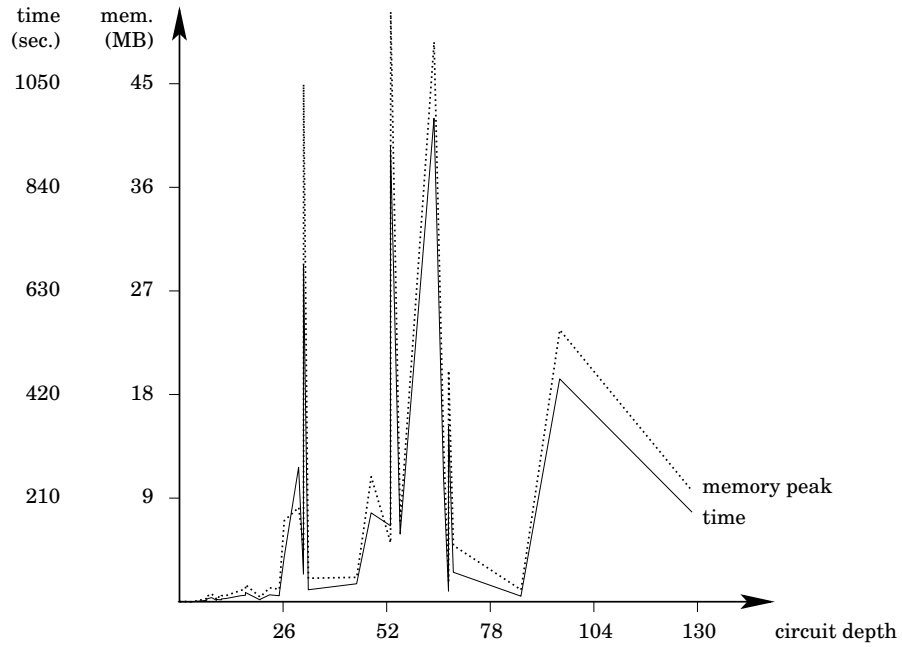


FIGURE 7.2: USE OF RESOURCES DEPENDING ON DEPTH OF CIRCUIT

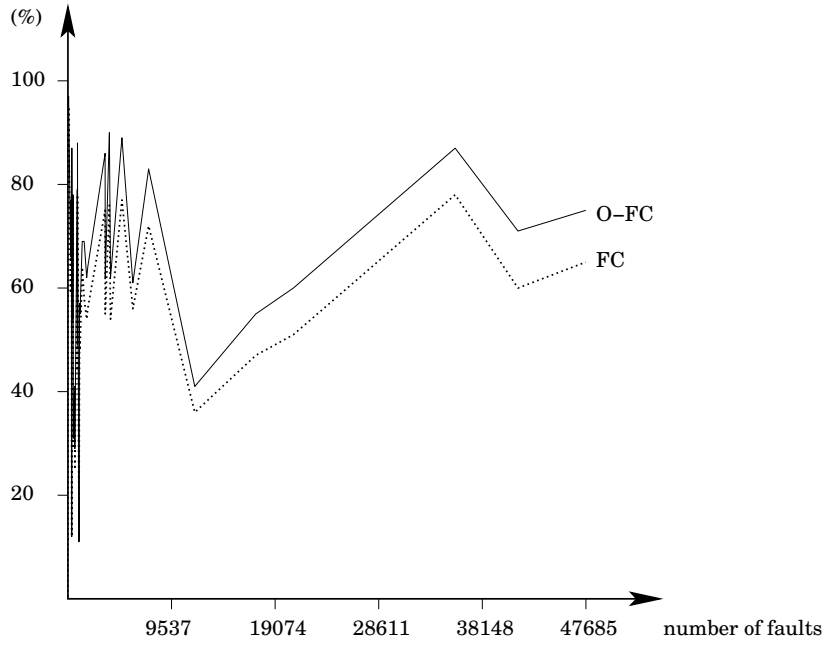


FIGURE 7.3: FAULT COVERAGE DEPENDING ON NUMBER OF FAULTS

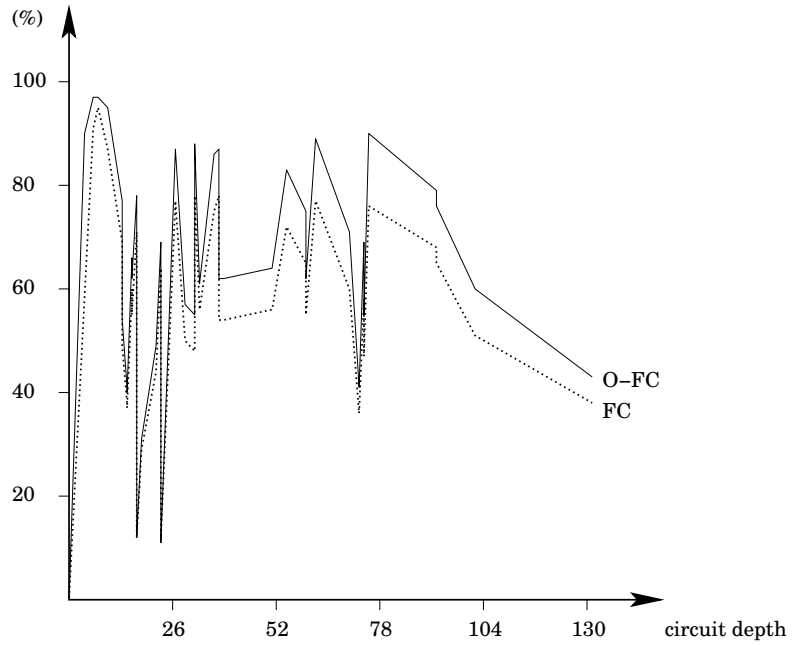


FIGURE 7.4: FAULT COVERAGE DEPENDING ON DEPTH OF CIRCUIT

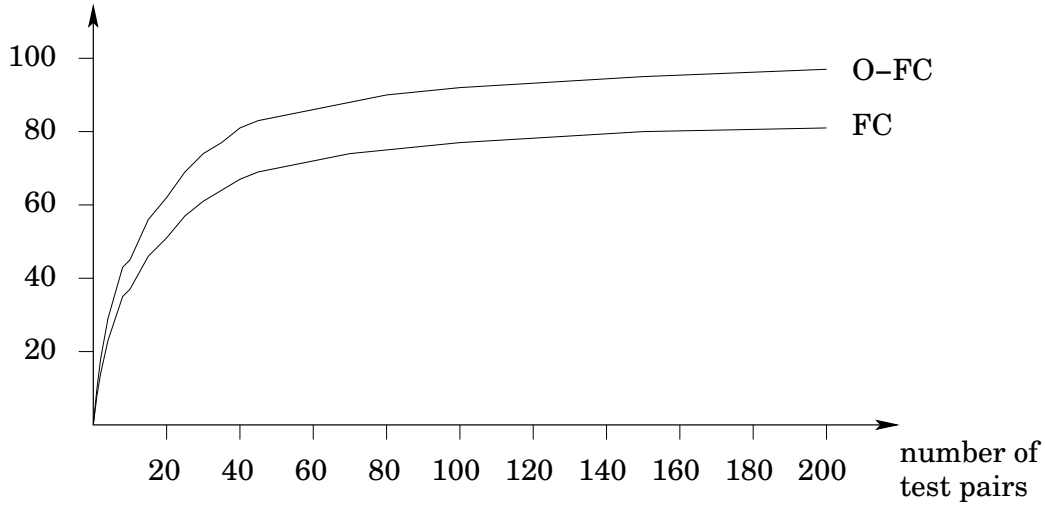


FIGURE 7.5: FAULT COVERAGE DEPENDING ON NUMBER OF TEST PAIRS, CIRCUIT C5315

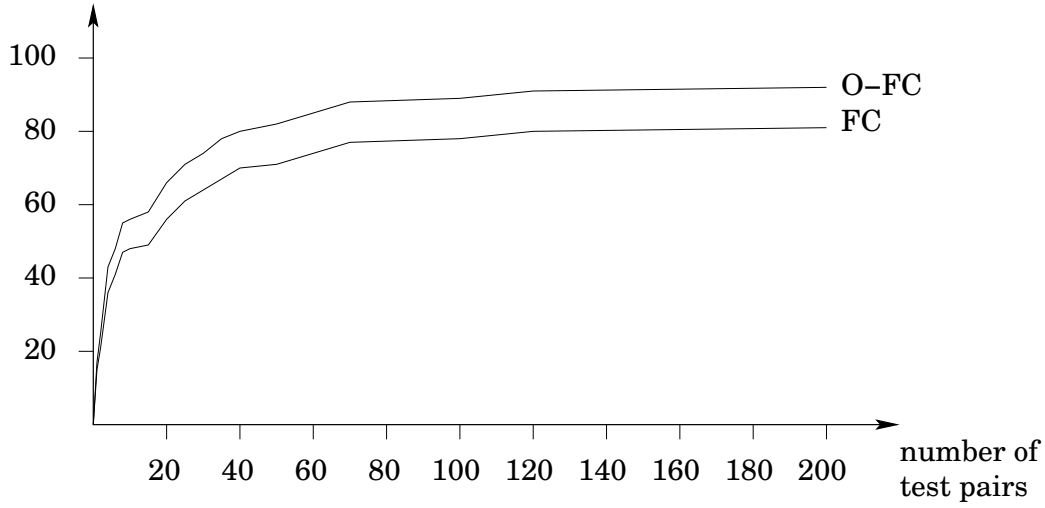


FIGURE 7.6: FAULT COVERAGE DEPENDING ON NUMBER OF TEST PAIRS, CIRCUIT S35932

An interesting observation is made when analysing Figures 7.3 through 7.6. The gap between O-FC and FC is rather small; in fact, O-FC never exceeds FC by more than about 10 percentage points. This means that, in the average, all faults which were detected for any resistance, could also be detected for a large range of resistances.

An analysis of the simulator's log files brought forth that all faults that were detected for any resistance had the range $[R_{\max}, +\infty[$ in their C-ADI. Thus, ignoring that range for the computation of FC does not influence in a negative way the value $\mathrm{FC}(f)$ for any detected fault f.

Unfortunately, we are not aware of other authors having already proposed a similar simulator for resistive opens. Hence, it was not possible to compare the obtained results with achievements of other authors.

8

Conclusions

Resistive opens are frequent defects in modern deep sub-micron technologies. They do not always affect the logic value of signal nodes, but cause signal changes to delay. Although these delays may be small enough as to make the fault hard-to-detect, it is very important to test for this kind of defects.

The resistance of an open is an unpredictable parameter, but it is essential to characterise the behaviour of the defect. In this work, a simulator for resistive opens was designed and implemented. It computes each fault's C-ADI, i.e. a set of ranges of resistances such that the open is detected by the test set if its resistance is in one of those ranges. These simulation results are especially useful during test pattern generation, as they tell if it is necessary to generate more test pairs that assure that the fault can be detected for a wide range of resistances.

The simulator works in three Phases. All three phases were presented and the method of Phase 1 was treated extensively. Additionally, possibilities of measuring fault coverage based on ADI were discussed. A new fault coverage metric was defined.

Experiments were performed on ISCAS 85 and ISCAS 89 circuits, the results were reported. One conclusion was that the use of computing resources depends only linearly on the circuit's size.

Future work includes the integration of the technology-based delay-to-resistance mappings for Phase 2 of the RO-simulator, performance profiling and increasing interoperability with existing tools for static resistive bridging faults like that presented in [12].

Appendix A

Contents of the Attached CD-ROM

Directory tree:	**Contents of directory:**
Diplomarbeit	
|-- bin	Makefile, README, executable files and configuration files needed to compile and run the simulator
|-- include	header files and documentation on implemented classes
|-- lib	compiled library and object files
|-- share	
| |-- Benchmarks	benchmark files
| |-- Documents	support documents and some of the foreign works mentioned in the bibliography
| |-- Experiments	
| | |--Inputs	complete lists of faults and lists of randomly generated test pattern pairs used to perform the experiments
| | |--Outputs	complete lists of faults and log files generated by the experiments
| |-- Thesis	this document
| | |-- Sources	pdfLaTeX-sources
| | |-- Graphics	included pdf-graphics
'-- src	source files of main applications, libraries and library testing routines

Bibliography

[1] *M. Abramovici, M.A. Breuer, A.D. Friedman.* Digital systems testing and testable design. Computer Science Press, 1990.

[2] *R.C. Aitken.* Nanometer technology effects on fault models for IC testing. IEEE Transactions on Computer, vol. 32, no. 11, pp. 46-51, November 1999.

[3] *B. Becker, I. Polian.* Testen von digitalen ICs. Course held at the Albert-Ludwigs-University of Freiburg, Winter Term 2003/04.

[4] *M.L. Bushnell, V.D. Agrawal.* Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits. Kluwer Academic Publishers, 2001.

[5] *A. Campbell, E. Cole, C. Henderson, M. Taylor.* Case history: failure analysis of a CMOS SRAM with an intermittent open contact. 17th International Symposium for Testing and Failure Analysis, Los Angeles, CA, November 1991.

[6] *J.L. Carter, V.S. Iyengar, B.K. Rosen.* Efficient test coverage determination for delay faults. Proceedings 1987 IEEE International Test Conference, pp. 418-427, 1987.

[7] *V.H. Champac, A. Zenteno.* Detectability conditions for interconnection open defects. IEEE VLSI Test Symposium, pp. 305-311, 2000.

[8] *V.H. Champac, J. Figueras.* Testability of floating gate defects in sequential circuits. Proceedings of the 13th IEEE VLSI Test Symposium, pp. 202-207, 2005.

[9] *K.T. Cheng, A. Krstic.* Current directions in automatic test-pattern generation. IEEE Transactions on Computer, vol. 32, no. 11, pp. 58-64, November 1999.

[10] *R.D. Eldred.* Test routines based on symbolic logical statements. Journal of the ACM, 6(1), pp. 33-36, 1959.

[11] *P. Engelke, I. Polian, M. Renovell, B. Becker.* Automatic test pattern generation for resistive bridging faults. Proceedings of the Ninth IEEE European Test Symposium, pp. 160-165, 2004.

[12] *P. Engelke, I. Polian, M. Renovell, B. Becker.* Simulating resistive-bridging and stuck-at faults. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, i. 10, pp. 2181-2192, October 2006.

[13] *P. Engelke, I. Polian, M. Renovell, B. Seshadri, B. Becker.* The pros and cons of very-low-voltage testing: an analysis based on resistive bridging faults. Proceedings of the 22nd IEEE VLSI Test Symposium, pp. 171-178, April 2004.

[14] *J.M. Galey, R.E. Norby, J.P. Roth.* Techniques for the diagnosing of switching circuit failures. Symposium on Switching Circuit Theory and Logical Design, pp. 152-160, March 1961.

[15] *C.D. Graas, H.A. Le, T.A. Rosi.* Correlations between initial via resistance and reliability performance. 35th Annual Proceedings IEEE International Reliability Physics Symposium, April 1997.

[16] *C. Hawkins, J. Soden, A. Righter, F.J. Ferguson.* Defect classes - an overdue paradigm for CMOS IC testing. International Test Conference, pp. 413-425, 1994.

[17] *C.L. Henderson, J.M. Soden, C.F. Hawkins.* The behavior and testing implications of CMOS IC open circuits. International Test Conference, pp. 302-303, 1991.

[18] *K. Heragu, J.H. Patel, V.D. Agrawal.* Segment delay faults: a new fault model. VLSI Test Symposium, pp. 32-39, 1996.

[19] *S. Irajpour, S.K. Gupta, M.A. Breuer.* Multiple tests for each gate delay fault: higher coverage and lower test application cost. Proceedings 2005 IEEE International Test Conference, pp. 9-17, November 2005.

[20] *V.S. Iyengar, B.K. Rosen, J.A. Waicukauski.* On computing the sizes of detected delay faults. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 9, no. 3, pp. 299-312, March 1990.

[21] *N. Jha, S. Gupta.* Testing of digital systems. Cambridge University Press, 2003.

[22] *H. Konuk.* Voltage and current-based fault simulation for interconnect open defects. Transactions On Computer-Aided Design of ICs and Systems, vol. 18, pp. 1768-1779, December 1999.

[23] *C. Lee, D.M.H. Walker.* PROBE: a PPSFP simulator for resistive bridging faults. VLSI Test Symposium, pp. 105-110, 2000.

[24] *Z. Li, X. Lu, W. Qiu, W. Shi, D.M.H. Walker.* A circuit level fault model for resistive opens and bridges. VLSI Test Symposium, pp. 379-384, 2003.

[25] *J.C.M. Li, E.J. McCluskey.* Testing for tunneling opens. Proceedings of the International Test Conference 2000, pp. 85-94, 2000.

[26] *E. Lindbloom, J.A. Waicukauski, B. Rosen, V. Iyengar.* Transition fault simulation by parallel pattern single fault propagation. International Test Conference, pp. 542-549, 1986.

[27] *A.K. Majhi, J. Jacob, L.M. Patnaik, V.D. Agrawal.* On test coverage of path delay faults. VLSI Design, pp. 418-421, 1996.

[28] *A.K. Majhi, V.D. Agrawal.* Tutorial: delay fault models and coverage. VLSI Design, pp. 364-369, 1998.

[29] *W. Maly.* Realistic fault modeling for VLSI testing. Design Automation Conference, pp. 173-180, 1987.

[30] *W. Maly, P.K. Nag, P. Nigh.* Testing oriented analysis of CMOS ICs with opens. Proceedings 1988 IEEE International Conference on Computer Aided Design, pp. 344-347, 1988.

[31] *E.J. McCluskey, C.W. Tseng.* Stuck-fault tests versus actual defects. International Test Conference, pp. 336-343, 2000.

[32] *W. Needham, C. Prunty, E.H. Yeoh.* High voltage microprocessor test escapes, an analysis of defects our tests are missing. Proceedings 1998 IEEE International Test Conference, pp. 25-34, October 1998.

[33] *E.S. Park et al.* Statistical delay fault coverage and defect level for delay faults. Proceedings 1988 IEEE International Test Conference, pp. 492-499, September 1988.

[34] *I. Polian.* On non-standard fault models for logic digital circuits: simulation, design for testability, industrial applications. VDI Fortschritt-Berichte, Reihe 20, Nr. 377, VDI-Verlag, Düsseldorf, 2004.

[35] *A.K. Pramanick, S.M. Reddy.* On the detection of delay faults. Proceedings 1988 IEEE International Test Conference, pp. 845-856, 1988.

[36] *A.K. Pramanick, S.M. Reddy.* On the fault coverage of gate delay fault detecting tests. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 1, pp. 78-94, January 1997.

[37] *M. Renovell, G. Cambon.* Topology dependence of floating gate faults in MOS integrated circuits. Electronics Letters, vol. 22, pp. 152-153, January 1986.

[38] *M. Renovell, P. Huc, Y. Bertrand.* CMOS bridge fault modeling. VLSI Test Symposium, pp. 392-397, 1994.

[39] *M. Renovell, P. Huc, Y. Bertrand.* The concept of resistance interval: a new parametric model for resistive bridging faults. VLSI Test Symposium, pp. 184-189, 1995.

[40] *M. Renovell, F. Azaïs, Y. Bertrand.* Detection of defects using fault model oriented test sequences. Journal of Electronic Testing: Theory and Applications, 14:13-22, 1999.

[41] *M. Renovell, M. Comte, I. Polian, P. Engelke, B. Becker.* Analyzing the memory effect of resistive open in CMOS random logic. Design and Test of Integrated Systems in Nanoscale Technology 2006, pp. 251-256, September 2006.

[42] *R. Rodríguez-Montañés, E.M.J.G. Bruls, J. Figueras.* Bridging defects resistance measurements in a CMOS process. International Test Conference, pp. 892-899, 1992.

[43] *R. Rodríguez-Montañés, P. Volf, J. Pineda de Gyvez.* Resistance characterization for weak open defects. IEEE Design & Test, v. 19 n. 5, p. 18-26, September 2002.

[44] *G.L. Smith.* Model for delay faults based upon paths. Proceedings 1985 IEEE International Test Conference, pp. 342-349, 1985.

[45] *J. Soden, R. Treece, M. Taylor, C. Hawkins.* CMOS IC stuck-open fault electrical effects and design considerations. Proceedings 1989 IEEE International Test Conference, pp. 423-430, 1989.

[46] *C.W. Starke.* Built-in test for CMOS circuits. International Test Conference, pp. 309-314, 1984.

[47] *N.N. Tendolkar.* Analysis of timing failures due to random AC defects in VLSI modules. Proceedings 22nd ACM/IEEE Design Automation Conference, pp. 709-714, June 1985.