MASTER

Test generation for current testing

Kolks, T.H.S.T.M.

Award date:
1991

Link to publication
Test Generation for Current Testing

T.H.S.T.M. Kolks

Master Thesis Report
author: T.H.S.T.M. Kolks
supervisor: Prof. dr. ing. J.A.G. Jess
coach: ir. J. Pineda
Eindhoven, the Netherlands
Februari 1991

The department of Electrical Engineering of the Eindhoven University of Technology does not accept any responsibility regarding the contents of student projects and graduation reports.
Summary

Current testing is found to be a very powerful tool for testing CMOS IC's because it is capable of detecting a large set of manufacturing defects. It requires a different methodology to generate a test set compared to traditionally used, functional failure test sequences. This paper presents a theory to sensitize some classes of defects in order to make them detectable using current testing. Based on this theory, a test generation heuristic is presented. This heuristic is able to generate short test sequences for large sets of defects.
Contents

Introduction .................................................. 1

Current Testing .............................................. 2
2.1 Process oriented Fault Modeling ....................... 2
   2.1.1 Defects in a Manufacturing Environment .......... 3
   2.1.2 Faults in a Manufacturing Environment .......... 4
   2.1.3 Defect-Fault Relation ............................ 5
   2.1.4 Quality of Testing ............................... 5
2.2 Current Testing .......................................... 6
   2.2.1 Currents in CMOS ICs with Shorts ................. 6
   2.2.2 Currents in CMOS ICs with Opens ................. 9
2.3 Advantages and Limitations of Current Testing ....... 9

Sensitization of Defects .................................... 11
3.1 Basic features and definitions of static CMOS
   circuits ................................................... 11
3.2 Bridges .................................................. 14
   3.2.1 Undetectable Bridging Defects .................... 14
3.3 Open Gates ............................................. 16
3.4 Open Drains and Sources .............................. 17
   3.4.1 Undetectable Opens .............................. 20

Test Generation ............................................. 21
4.1 Concepts and Definitions ................................ 21
4.2 Manipulating Boolean Functions ....................... 24
4.3 Optimal Test Set Generation ......................... 24
4.4 Heuristic Test Set Generation ....................... 26
4.5 Implementation notes .................................. 27
4.6 Experimental Results ........................................ 29
Conclusions and Discussion ..................................... 32
  5.1 Conclusions .................................................. 32
  5.2 Discussion ................................................... 32
      5.2.1 Defect Dominance ..................................... 33
      5.2.2 Other Heuristics ....................................... 34
Appendix .................................................................. 35
References ........................................................... 37
Chapter 1

Introduction

Very Large Scale Integration (VLSI) has enabled us to implement very complex circuits on a single chip. The advantages of VLSI chips are obvious, however, they do pose a problem. That problem is testing of such chips to ensure that they operate properly. Testing of chips has become an increasing part in integrated circuit manufacturing as far as time and costs are considered. Therefore, more attention has been paid to the area of digital system testing.

One of the methods for testing IC's, arising from this interest, is current testing. It is a kind of IC testing suitable for CMOS technology, the dominating technology over the last years. Current testing was found to be a very powerful tool for its ability to test a large class of manufacturing defects. Due to recent developments, it has been made possible to make use of so called Built-in Current testing (BIC). This provides an excellent observability of the circuit under test, making testing possible at a high speed.

However, current testing requires a different test generation methodology than traditional voltage-oriented testing. This report formulates an approach for test generation for current testing. Algorithms are described and the result of an implementation of these algorithms are evaluated. The organization of this report is as follows. First, in chapter 2, the aspects of current testing and its properties are stated. In chapter 3, ways to make defects detectable are presented. Then, in chapter 4, algorithms for test generation are shown and the results of implementing those algorithms are considered. Finally, a possibility to improve test generation for current testing is discussed.
Chapter 2

Current Testing

Failures of VLSI circuits are caused by manufacturing process induced defects. Typically, these defects are described using a functional-fault modeling technique. The physical reality in the IC, though, is in many cases complicated and actual defects may cause circuit behaviour that is far from the state that can be represented by the traditional fault models. Therefore, tests generated using these methods may lack of quality.

Current testing is a powerful testing method for CMOS IC's. In this method, the device under test is monitored for its current flow through the power supply terminals while input stimuli are applied. Abnormal currents are an indication for defects in the circuit. Current testing uses a different fault model which emphasizes the defect rather then the fault. It appears that a vast majority of process induced defects can be detected using current testing. Hence, tests obtain a higher degree of quality.

2.1 Process oriented Fault Modeling

The outcome of IC production may differ from its expected result due to process disturbances. These disturbances can manifest themselves as deformations in the IC topology. The result of such deformations sometimes leads to unwanted behaviour of the product. In this paragraph, basic reasons for this difference are briefly explained as well as their impact on IC performance and testing strategies.
2.1.1 Defects in a Manufacturing Environment

The manufacturing of an IC consists of several process steps. The outcome of a step is subjected to process disturbances. These process disturbances are random phenomena [3], which may result in a permanent modification of the physical characteristics of a fabricated IC structure. Disturbances may have a local or a global nature. The term "global" is used when the entire manufacturing wafer is affected by the disturbance. Global disturbances are usually easily detected during the manufacturing process of integrated circuits. The term "local" is used to describe phenomena in a region much smaller then the wafer, often to the size of a single device. Local disturbances may lead to defects. A defect is a discrepancy between the circuit design and its physical realization, affecting an area smaller then the size of a single device. They are often called spot defects. Although there are many sources of spot defects, and therefore a lot of possible kinds of spot defects, they can in general be classified as [2]:

- Shorts caused by:
  - Spots of extra or conducting or semiconducting material in conducting layers (horizontal shorts);
  - Spots of missing insulating material (vertical shorts);
- Breaks caused by:
  - Spots of missing conducting or semiconducting material (horizontal breaks);
  - Spots of extra insulating materials in the windows of insulating layers;
- New (parasitic) devices;
- Devices with a degraded performance;

These four classes are expected to cover most defects occurring in IC manufacturing. Two examples of these defects are illustrated in Fig. 2.1 and 2.2.
2.1.2 Faults in a Manufacturing Environment

A defect may result in a fault. A fault is a discrepancy between the actual circuit performance and its expected performance. In general, three types of faults are categorized, on the basis of the duration for which they manifest themselves:

- A *permanent fault* results in a stable deviation.

- A *intermittent fault* appears regularly but is not present continuously.
§2.1.2 Faults in a Manufacturing Environment

— A transient fault appears only once and is therefore untracable.

2.1.3 Defect-Fault Relation
Looking at the definitions of defect and fault, there are several important things that can be noticed. First, a fault, according to its definition, does not always have to be regarded as a deviation in the functional behaviour of the circuit. That is, not only a difference in the logical output of the device in comparison with a correct one, is of importance.

Secondly, a defect does not have to result in a fault. It could be a so called benevolent defect. Also, it is possible that a defect is not important a the time being, but may eventually result in a fault.

These aspects are common to the relation between defects and faults. Usually, the relation between a defect and its fault is described using a fault model. A fault model maps a set of defects onto a set of fault types, thereby forcing a relation between these two. Traditionally, the stuck-at fault model is used. In the stuck-at fault model, it is assumed that a defect causes a line in the circuit to behave as if it is permanently at logic 0 or logic 1. The stuck-at model is often used to generate a circuit test. However, these test sequences have limitations which are inherent to that fault model[2]. Taken together, such test sequences have following drawbacks:

— Detection of nonexisting faults;
— Some existing faults are not detected;
— Inefficiently used by not taking into account the likelihood of occurrence of various faults and combinations of multiple faults;

Other fault models that are being used, like the stuck-open and the stuck-on fault model, have the same limitations. It should be clear that the ability of the fault model to represent defects, when used in test generation, is of major importance regarding the efficiency and the usefulness of the test sequence.

2.1.4 Quality of Testing
An ideal test sequence is a test that would reveal all defects that cause or may cause in future any malfunctioning of the circuit under test.
Since it is not possible to perform such a test, or even come up with one, it is suitable that some kind of measure is given to indicate the power of a test.

A very good way to express quality of a test is in terms of defect detectability. Defect detectability can be stated as a measure of probability that a fully tested IC contains no undetected defects. This definition stresses the need to detect defects instead of faults. A upcoming detail in this matter is whether or not to recognize benevolent defects, but this seems merely a detail.

Often, with the use of traditional fault models, the term fault coverage is applied to express the quality of testing. Fault coverage is defined as the percentage of the assumed faults that will be detected in applying the test. As the definition suggests, the principle of fault coverage presumes a fault model. Hence, the actual quality of the test is not only depending on the fault coverage factor itself but also on the quality of the fault model. Therefore, the use of the fault coverage factor only is not suitable to describe the quality of testing.

2.2 Current Testing

Current testing is a potential methodology for testing CMOS circuits. It is a fairly good testing method because it emphasizes the occurrence of a defect rather then a fault.

Testing for currents is based on the assumption that a defect-free circuit's energy consumption is determined by its topology and the input signals. Therefore, the power supply currents of an IC are well determined by both circuit topology and input stimuli. In CMOS circuits, there are no steady state Vdd-Gnd paths and consequently, no static current, except for some small junction leakages. Defects may cause abnormal power supply currents.

We start with a more detailed analysis of defects inducing elevated currents. Two of the most important classes of defects are considered here: regions with either extra or missing material which cause shorts or opens.

2.2.1 Currents in CMOS ICs with Shorts

Shorts are likely to occur in defective CMOS circuits. They are
§2.2.1 Currents in CMOS ICs with Shorts

sometimes refered to as bridges. It is very likely that due to this bridges abnormal power supply current is caused. This can be explained as followed.

In a correctly designed digital CMOS IC, a node in steady state has a potential about equal to Vdd or Gnd. It is also possible that a node is in a high impedance state. Most part of the nodes can be toggled to another state when applying a suitable input vector. Two nodes can be called logically equivalent if there isn't a set of input vectors, other then the empty set, that will put these nodes in opposite state. Suppose now that there is a bridge between two nodes that are not logically equivalent. Because they are not logically equivalent, there is at least one input vector that puts these nodes into opposite states. Therefore, there will be a voltage drop across the bridge which will cause a steady current to flow. This current indicates the presence of this defect. Figure 2.3 shows a circuit with a bridge between two nodes. A typical waveform of the supply current of the circuit without the defect can be seen in the Fig. 2.4. This waveform was obtained using a SPICE simulation. The circuit was also simulated with the defect. The response is drawn in Fig. 2.5 and shows the expected result.

Figure 2.3. Circuit simulated using SPICE
It is important to state that in this manner general bridging defects can be detected. This includes:

- gate oxide pinholes:
  - gate - substrate shorts;
  - gate - drain shorts;
  - gate - source shorts;
  - gate - P/N well shorts;
- drain-source bridges;
- leakage defects;
§2.2.1 Currents in CMOS ICs with Shorts

There is also a class of shorts that cannot be detected. This class contains pairs of nodes that are logically equivalent, but may also contain other pairs of nodes. Exactly which bridging defects are not detectable is explained in chapter 3. This class is understood to be very small and insignificant from the practical point of view [5].

2.2.2 Currents in CMOS ICs with Opens

To test for opens in a circuit using the method of current testing is more difficult. The question is whether or not an open results in an abnormal current. So far, the greater part of research has been devoted to the consequences of the functional behaviour of a circuit under test under influence of opens. Recently, the influence of opens on currents has been investigated [6]. It was based on a new MOS transistor charge model [7]. It showed, in theory as well as in practice, that some kinds of opens in CMOS often can be detected. These kinds of opens are:

- Open gates;
- Open sources and drains;

The result of this has been confirmed [4].

The general thought in case of an open gate is, that it is not stuck open but, under circumstances, weakly conducting. It can therefore be detected using current testing.

It is also possible that due to an open drain or source, extra current will flow through the power terminals.

However, to perform a SPICE simulation on these opens is somewhat difficult, because it it is based on a MOS model different from that of the model used in SPICE [8]. A deeper thought on the mechanism which will result in this is given in chapter 3. Here, it is enough to state that current testing can be used to detect open gate defects, and can under circumstances detect open sources and drains.

2.3 Advantages and Limitations of Current Testing

As explained in the previous paragraphs, current testing can be used to detect a vast majority of defects actually occurring in IC manufacturing process. In this, it has an advantage over traditional
testing methods which test, as stated, for faults under the assumption of a fault model which is not always correct.

Another advantage is that it is not in need of extra observability as some other testing methods are.

One of the earlier main disadvantages of current testing was the monitoring of the supply current. This required sophisticated equipment to reduce influences of noise. Also, it made current testing a very slow method. By using Built-In-Current testing (BIC) these problems have been overcome [9]. BIC implies a current sensor which has to be build onto the chip. Detection of abnormal currents is done by the BIC sensors which output a single pass or fail bit. By using BIC, the test rate can be made high, also there is no need for any evaluation pass of any output like in SCAN path testing methods [1].

It has been presented that the actual test length of current testing is smaller then that of stuck-at tests [4]. This is a result of the fact that for stuck-fault vectors more restrictions are required. They are constructed from a so called fault sensitizing step followed by a fault propagation step. For current testing only a sensitizing step is needed. Hence, less restrictions are placed upon the inputs to sensitize the defect, making it possible to intersect several input vectors to sensitize multiple defects by applying one vector.

A disadvantage is that it is only usable for CMOS technology and in practice it requires the use of a BIC sensor. But one can conclude that it is a powerful tool in testing the outcome of manufacturing processes.
Chapter 3

Sensitization of Defects

In this chapter methods are presented to sensitize certain defects so they will be detectable using current testing. Some basic definitions are stated first.

3.1 Basic features and definitions of static CMOS circuits

In this report static CMOS circuits are regarded, which consist of transistors connected by nets. A net is referred to as a node. CMOS static gates consist of a network of p-channel transistors, the *P-part*, and a network of n-channel transistors, the *N-part*. There is no restriction to the use of series-parallel gates only. Fig. 3.1 presents a sketch of a circuit.

![Figure 3.1. General static CMOS Circuit](image)
The circuit is driven by primary inputs, called inputs, while the functional response can be monitored at the primary outputs, simply called outputs. Along with that, we define a input_set as the set of all nodes which are inputs. We also define a output_set as the set of all primary output nodes. We can use fig. 3.1 to present some other definitions.

A switch node is the node connecting a P-part and a N-part of a gate while not being an output node or an input node. The switch_set is the set of all switch nodes. All nodes are present in the total_set.

Further, we discern two special nodes. First, a node is called a Vdd Node if it is always at the Vdd potential. Secondly, a node always at the Gnd potential is called a Gnd node. Note that these definitions can be applied to the power terminals.

An internal node is a node that is neither a switch node, nor an input node, nor an output node nor a Vdd node, nor a Gnd node. Hence, internal nodes are the nodes inside the P-parts and the N-parts.

Two nodes are called connected if there is at least one path of conducting transistors between them.

A node is called Vdd connected if it is connected with a Vdd node. A node is called Gnd connected if it is connected with a Gnd node.

In the steady state, each node of an IC must be in one of four states:

- **Vdd state**: The node is Vdd connected and not Gnd connected.
- **Gnd state**: The node is Gnd connected and not Vdd connected.
- **HZ (High Impedance)**: The node is neither Vdd connected nor Gnd connected.
- **X state**: The node is both Vdd connected and Gnd connected.

In a static combinational network, the state is completely defined by the state of its nodes. The state of a node is completely defined by the state of the input nodes. Boolean vectors can be used to describe the states of these inputs assuming that an input node may only be in the Vdd state or the Gnd state. Hence, a certain state of a node can be described by a set of input vectors which will force that node to that specific state. Such a set of input vectors is represented by a boolean function. This boolean function is a boolean equation on the input
§3.1 Basic features and definitions of static CMOS circuits

nodes, and is true and only true for an input vector which is part of the set represented by that function. We call such a boolean function a formula. We can use these formulas to define the state of a node. Let $F_i^{Vdd}$ be a formula which forces the node $i$ to be Vdd connected. In the same way, let $F_i^{Gnd}$ be the formula which causes the node $i$ to be Gnd connected. Subsequent definitions can be stated for $F_i^{HZ}$ and $F_i^{X}$. From their definitions follows:

$$F_i^{HZ} = F_i^{Vdd} + F_i^{Gnd}$$  \hspace{1cm} (3.1)

and

$$F_i^{X} = F_i^{Vdd} \cdot F_i^{Gnd}$$  \hspace{1cm} (3.2)

In these definitions, " $\cdot$ " represents the boolean multiplication of two formulas. The operator " $+$ " is used for the boolean summation of two operands. The bar above an operand indicates its complement. To describe the repeated appliance of the multiplication and summation operator, the symbols " $\prod$ " and " $\Sigma$ " are used, respectively.

The P-part of a CMOS gate controls the Vdd connectability of the gate's outcome, while the N-part controls the Gnd connectability. In a correctly designed CMOS circuit without defects, the switchnode (or output node) of a gate can never be simultaneously connected to Vdd and Gnd. Therefore, there should never be a path between a Vdd node and a Gnd node resulting in a static current. This shows that in a correctly designed and defect free static CMOS circuit, the X state does not occur. Hence,

$$\forall i \in \text{total.set} : F_i^{X} = 0$$  \hspace{1cm} (3.3)

We define a circuit node to be an output or an input or a switch node. All circuit nodes are present in the circuit_set, which is consequently the union of the output_set and the input_set and the switch_set. Circuit nodes are the nodes that can be seen when regarding a circuit at gate level instead of transistor level. In correctly designed, defect free, combinational CMOS circuit, such a node may never be in a HZ state. Hence:

$$\forall i \in \text{circuit.set} : F_i^{HZ} = 0$$  \hspace{1cm} (3.4)

Together with eq 3.1 this results in:

$$\forall i \in \text{circuit.set} : F_i^{Gnd} + F_i^{Vdd} = 1$$  \hspace{1cm} (3.5)
For fully complementary MOS gates (FCMOS), eq 3.5 leads to:
\[
\forall_{\text{circuit set}} \ : \ F_i^{\text{Gnd}} = F_i^{\text{Vdd}}
\] (3.6)

### 3.2 Bridges

A bridging defect between two nodes causes the occurrence of a X state, for if one node is Vdd connected and the other is Gnd connected, the bridge between those nodes puts them both in the X state. This X state can be detected using current testing. Hence, testing for a bridge between two nodes can be done by forcing the two nodes into opposite states Gnd and Vdd.

We can derive a formula to detect a bridging defect between a node i and a node j and name it test formula \( f_{\text{bridge}}^{(i,j)} \). It is obvious that:
\[
f_{\text{bridge}}^{(i,j)} = (F_i^{\text{Vdd}} \cdot F_j^{\text{Gnd}}) + (F_i^{\text{Gnd}} \cdot F_j^{\text{Vdd}})
\] (3.7)

Such a test formula represents all the input vectors which will sensitize its bridging defect. To find one such an input vector, the test formula only needs to be satisfied.

#### 3.2.1 Undetectable Bridging Defects

As already stated in an earlier chapter, there are some bridging defects that can not be detected. Take as an example a bridging defect between two nodes that are logically equivalent. Because they are logically equivalent, they can never be forced into the opposite states. Theorem 3.1 states, when a bridging defect can not be detected:

**Theorem 3.1** A bridging defect between node i and node j can not be detected using current testing, iff:
\[
F_i^{\text{Vdd}} \leq (F_i^{\text{Vdd}} + F_i^{\text{HZ}}) \land F_i^{\text{Gnd}} \leq (F_i^{\text{Gnd}} + F_i^{\text{HZ}})
\] (3.8)

The "\( \leq \)" operator is used to indicate that the set represented by one formula is a subset of the set represented by the other formula. The translation of this operator into boolean operations on formulas is trivial.

**Proof:** A bridging defect between an node i and a node j can not be detected iff its test formula equals zero:
\[
f_{\text{bridge}}^{(i,j)} = 0
\]
§3.2.1 Undetectable Bridging Defects

\[(F_{i}^{\text{Vdd}} \cdot F_{j}^{\text{Gnd}}) + (F_{i}^{\text{Gnd}} \cdot F_{j}^{\text{Vdd}}) = 0\]
\[
\iff
\] \[(F_{i}^{\text{Vdd}} \cdot F_{j}^{\text{Gnd}}) = 0 \land (F_{i}^{\text{Gnd}} \cdot F_{j}^{\text{Vdd}}) = 0\]
\[
\iff
\] \[F_{i}^{\text{Vdd}} \leq \overline{F_{j}^{\text{Gnd}}} \land F_{j}^{\text{Gnd}} \leq \overline{F_{j}^{\text{Vdd}}}\]

Since \(F_{i}^{\text{Vdd}}, F_{j}^{\text{Gnd}},\) and \(F_{j}^{\text{HZ}}\) are disjoint sets whose union includes all possible states, then

\[
\overline{F_{j}^{\text{Gnd}}} = F_{j}^{\text{Vdd}} + F_{j}^{\text{HZ}}\]
and
\[
\overline{F_{j}^{\text{Vdd}}} = F_{j}^{\text{Gnd}} + F_{j}^{\text{HZ}}\]

Resulting in:

\[
F_{i}^{\text{Vdd}} \leq (F_{j}^{\text{Vdd}} + F_{j}^{\text{HZ}}) \land F_{j}^{\text{Gnd}} \leq (F_{j}^{\text{Gnd}} + F_{j}^{\text{HZ}})\]

Fig 3.2 shows an example of a pair of nodes for which no test vector can be found. Note that this is not a trivial example since the nodes are not logically equivalent.
3.3 Open Gates

In [6] is stated that a MOS transistor with an open gate is not a transistor stuck-open. It appears that such a transistor is weakly "on" whenever the drain is Vdd connected and the source is Gnd connected in case of a n-channel transistor. If it is a p-channel transistor, the drain should be Gnd connected and the source should be Vdd connected. Due to the fact that the transistor starts to conduct, an X state occurs. Fig 3.3 presents an example.
To present a formula which sensitizes this defect we consider a transistor $t$, its drain node $\text{drain}$, source node $\text{source}$ and gate node $\text{gate}$. Note that since there is no physical difference between the source and the drain of a MOSFET, it is sufficient that these are driven into opposite states.

Then, for a n-channel MOS transistor we name a test formula which sensitizes a floating gate as $T_{\text{open}_G}^{<1,N>}$. Hence,

$$T_{\text{open}_G}^{<1,N>} = \text{Gnd}_{\text{gate}} \cdot (\text{Vdd}_{\text{drain}} \cdot \text{Gnd}_{\text{source}} + \text{Gnd}_{\text{drain}} \cdot \text{Vdd}_{\text{source}})$$

(3.9)

For a p-channel MOS transistor a similar test formula, $T_{\text{open}_G}^{<1,P>}$, is defined and:

$$T_{\text{open}_G}^{<1,P>} = \text{Vdd}_{\text{gate}} \cdot (\text{Vdd}_{\text{drain}} \cdot \text{Gnd}_{\text{source}} + \text{Gnd}_{\text{drain}} \cdot \text{Vdd}_{\text{source}})$$

(3.10)

These test formulas force the drain and source of the transistor into opposite states while the gate should turn the transistor off, if there is no open gate defect present in the transistor. To find a test vector that will sensitize a open gate defect, eq. 3.9 or 3.10 has to be satisfied.

### 3.4 Open Drains and Sources

To discuss the sensitization of this kind of opens, lets look at some of their aspects first. In fig. 3.4 a circuit is drawn, which is infected by an open source in the N-part of the NOR-gate:
Obviously, the switchnode of the NOR gate still can be Vdd connected through the correct P-part. Applying the vector \((00x)\) for the inputs \(ABC\) will result in pulling the switchnode to Vdd. However, if the inputs are set to \((011)\), then there is no Gnd connectability, as well as there is no Vdd connectability resulting in a HZ state for the switchnode. Initially, the node will contain its previously obtained value, the reason why these defects are sometimes referred to as *memory defects*. However, due to the leakage currents of the diodes in sources and drains the node is slowly discharged if the latter vector is applied long enough. This effect has been predicted and measured by several papers \([6, 10]\). The time for the switchnode to switch due to the discharging currents, can, under normal circumstances, be more then ten seconds.

The consequence is, that the transistors driven by the switchnode are driven by, in this case, a very slowly falling voltage. Now, if the voltage of the switchnode is above the threshold voltage of the n-channel transistor but below the threshold voltage of the p-channel transistor current will flow through the NAND gate. This abnormal current can be detected by current testing.

Same reasoning is valid for p-channel open sources as well as open drains. Normally, it will take a long time before the switchnode reaches a voltage for which both following transistors will start to
§3.4 Open Drains and Sources

conduct, if at all, therefore making it an impractical testing method. Fortunately, the delay time can be drastically influenced by light [6], hence making testing for this class of opens usable in practice. Note that no significant current passes through the NOR gate because of the open.

Now, we are ready to derive test formulas for an open source or open drain defect. It may be clear that two test patterns are needed for an open test, one to initialise the switchnode, the other to let it be pulled up or down. Therefore, we define a test formula to test a open drain/source defect in n-channel transistor \( t \) with drainnode drain, sourcenode source and gatenode gate as \( \mathcal{S}_{\text{open DS}}^{<t, N>} \). It consists of a initialisation formula \( IN^{<t, N>} \) followed immediately, in time, by a pull down formula \( GO^{<t, N>} \) as in:

\[
\mathcal{S}_{\text{open DS}}^{<t, N>} = < IN^{<t, N>} , GO^{<t, N>} >
\] (3.11)

Similarly, we define a test formula \( \mathcal{S}_{\text{open DS}}^{<t, P>} \) for a p-channel transistor \( t \), with nodes drain, gate and source, as:

\[
\mathcal{S}_{\text{open DS}}^{<t, P>} = < IN^{<t, P>} , GO^{<t, P>} >
\] (3.12)

Now, we would like to define the initialisation formulas and the pull up or down formulas. The first is easy enough. We define switch as the switchnode of the gate which includes the transistor \( t \). To notice, "gate" is now not used in the sense of "gate of an transistor". Then, the initialisation formula consists merely of the formula to pull this node up or down. Hence,

\[
IN^{<t, N>} = F_{\text{switch}}^{Vdd}
\] (3.13)

and

\[
IN^{<t, P>} = F_{\text{switch}}^{Gnd}
\] (3.14)

To compute the \( GO \) formula, we remark the following. First, the switch of that gate must only be Vdd or Gnd connected by a path which will contain the transistor. Otherwise, the switch will be immediately connected to a Vdd node or a Gnd node, so the voltage of the switch falls or rises at once. We will state such a formula by \( \mathcal{S}_{\text{switch}}^{Gnd} \) for some path or \( \mathcal{S}_{\text{switch}}^{Vdd} \) for some other path.

Secondly, the gates driven by the switch, in the example it is the NAND gate, should get the "chance" to conduct an amount of current.
That is, their should be an $X$ state in the switch nodes of those gates. Those nodes will be called $driven\_switches$ of $t$ and call a formula that achieves this $\mathcal{G}_{\text{driven\_switches}}^X$. In the example this was achieved by setting input $C$ to $1$. We remark that in case there are several gates driven by a switch, it is sufficient if only one of them is bound to that demand.

Now we can define $GO_{<t,N>}$ and $GO_{<t,P>}$:

$$GO_{<t,N>} = \mathcal{G}_{\text{Gnd},t\in\text{path}} \bullet \left( \sum_{i \in \text{driven\_switches of } t} \mathcal{G}_i^X \right) \quad (3.15)$$

or

$$GO_{<t,P>} = \mathcal{G}_{\text{Vdd},t\in\text{path}} \bullet \left( \sum_{i \in \text{driven\_switches of } t} \mathcal{G}_i^X \right) \quad (3.16)$$

Although it seems that we have now derived a way to sensitize open drains and open sources, the results are not valid. Two mechanisms can make this way of testing invalid [14], namely:

— Circuit delays;
— Charge sharing;

The latter is of minor importance since it can and will be avoided by adding some simple constraints to the circuit to be designed. However, circuit delays can prevent a two pattern test from detecting the stuck open unless it is very carefully derived. Also, there are circuits for which a valid test set does not exist at all. In practice, some opens probably will be detected, so we will contain these formulas, keeping in mind the relative power of them.

### 3.4.1 Undetectable Opens

If an open gate is undetectable, the test formulas derived in eq. 3.9 or eq. 3.10 should be equal to zero. This, however, is not to be expected in a correctly designed, combinational CMOS circuit.

Regarding the open sources and drains, there are some opens that cannot be detected even if the test is valid. Those are the opens for which eq. 3.15 or 3.16 are equal to zero. Open drains in gates containing an output node, for instance can not be detected, because they do not possess gates which are driven by them. Therefore, there is no exceptional current to be measured.
Chapter 4

Test Generation

In this chapter, algorithms are described that generate a test for current testing. Both an optimal and a heuristic approach is regarded, but only a heuristic is more precise depicted. Computation on formulas is done using Reduced Ordered Binary Decision Diagram's (ROBDD's). Also, an implementation is briefly described and run time results are considered, using an ISCAS '85 benchmark.

4.1 Concepts and Definitions

A certain set of defects are considered. These defects are sensitized according to their sensitizing equations. An algorithm that will handle this starting set of defects is shown in Algorithm 4.1.

The result of this algorithm is a set of satisfiable defects, i.e. defects for which their test formulas are not equal to zero. We are not interested in specifying which defect is related to such a test formula, the identification of a defect is of minor importance, so when we talk about a set of defects $D$ we can imagine $D$ as $\{1,2,\ldots,N\}$ where each integer of the set uniquely represents some satisfiable defect. For now it is also unimportant to have any knowledge about whether or not such a test formula contains the initialization vector or the pull up or down vector in case of an open drain or source. All we have to know is that such a satisfied test formula sensitizes something. We denote such a general test formula for a defect $k$ by $r_k$.

Suppose we have two test formulas, $r_k$ and $r_l$. Then the defects they represent can be possibly detected by one single test vector. This is the
Algorithm 4.1. Algorithm to Compute the Test Formulas

```
Start_Set  /* initial set of defects */
D = ∅     /* resulting set of defects */
F_set = ∅ /* resulting set of test sets */

foreach (defect ∈ Start_Set) {
    \( \mathcal{F} = \) sensitize (defect);
    if (\( \mathcal{F} \neq 0 \))
        print ("Not detectable!");
    else {
        D = D \cup defect;
        F_set = F_set \cup \mathcal{F};
    }
}
```

case if we choose a test vector that satisfies both \( \mathcal{F}_k \) and \( \mathcal{F}_l \). Since a formula represents a set of vectors, it is obvious that that single test vector is member of the set described by \( \mathcal{F}_k \bullet \mathcal{F}_l \). We denote a test formula that represent the set of input vectors which will detect both defect \( k \) and \( l \) at the same time as \( \mathcal{F}_{(l,k)} \). Hence,

\[
\mathcal{F}_{(l,k)} = \mathcal{F}_k \bullet \mathcal{F}_l \quad (4.1)
\]

Two defects \( l \) and \( k \) can be detected by a single test vector iff:

\[
\mathcal{F}_l \bullet \mathcal{F}_k \neq 0 \quad (4.2)
\]

The expansion of eq 4.1 and 4.2 to multiple defects is trivial. A multiple defect test formula, simply called a multiple formula, for detecting the set of defects \( D \) is given by \( \mathcal{F}_D \) and:

\[
\mathcal{F}_D = \prod_{i \in D} \mathcal{F}_i \quad (4.3)
\]

A test vector for the set of defects \( D \) can be found if \( \mathcal{F}_D \) is unequal to zero, in which case we say that \( \mathcal{F}_D \) is satisfiable. Else, we say that it is not satisfiable. We talk about a maximum clique formula \( \mathcal{F}_S \) of a set \( D \), if \( S \) is the biggest subset of \( D \) for which \( \mathcal{F}_S \) is still satisfiable.

A test vector for a set \( D \) can be found iff:

\[
\prod_{i \in D} \mathcal{F}_i \neq 0 \quad (4.4)
\]
A consequence from eq. 4.4 follows from the fact that multiplying is both communicative and associative:

$$\mathcal{F}_D \neq 0 \Rightarrow \forall S \subseteq D \mathcal{F}_S \neq 0 \quad (4.5)$$

To obtain more knowledge about how to generate a satisfiable multiple formula, we temporarily introduce a graph according to the following definition. Consider a undirected graph $G = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ representing single defect test formulas for a certain set of defects $V$. There exists an undirected edge between between nodes $\mathcal{F}_i$ and $\mathcal{F}_j$, $i, j \in \mathcal{V}$, iff $\mathcal{F}_i \cdot \mathcal{F}_j \neq 0$. We call this graph a intersection graph. From eq. 4.6 it follows that a satisfiable multiple defect test formula $\mathcal{F}_S$ for $S \subseteq V$ induces a complete subgraph in the graph $G$ for the set of nodes $S$. Unfortunately, this property not appears to he enough to make $\mathcal{F}_S$ satisfiable. To proof this, let us look at the example depicted in fig. 4.1.

It represents the intersection graph for five single test formulas.

$$\begin{align*}
\mathcal{F}_1 &= a \\
\mathcal{F}_2 &= b \\
\mathcal{F}_3 &= c \\
\mathcal{F}_4 &= \bar{a} + \bar{b} \\
\mathcal{F}_{\{1,2,3,4\}} &= 0
\end{align*}$$

**Figure 4.1.** Intersection Graph Representation for Single Defect Test Sets

The graph in Fig. 4.1 is a complete graph. However, the result $\mathcal{F}_{\{1,2,3,4\}}$ is equal to zero. So for these 4 defects there exists no multiple test formula which would detect them all. This implies that the existance of a satisfiable multiple defect formula can not be predicted by looking at its intersection graph. A multiple defect formula has to be computed first, before it is known to be satisfiable or not. For this reason, the at first sight promising concept of the intersection graph will not be used any further nor will be any other representation.
So far, we have looked at defects and their test formulas which could be detected by one single test vector only. To deal with two test pattern defects we have to introduce in some kind of way the aspect of time. To do this, we can give a more specific definition of the test set. A test set is a ordered set of input vectors. A ordering between to vectors \( v_1 \) and \( v_2 \) is written as \( <v_1, v_2> \) and means that \( v_1 \) is a vector which is immediately followed by \( v_2 \). Each vector is placed in a timeslot, so the length of the test set is equal to the number of timeslots it occupies.

With these concepts and definitions we are able to describe some algorithms in a more clear way.

### 4.2 Manipulating Boolean Functions

It is clear that generating test vectors for current testing is depending on the efficient manipulation of boolean functions since defect sensitizing is described using these functions and test generation algorithms are also dealing with formulas. The boolean functions are represented using reduced, ordered, binary decision diagrams [12]. Manipulating ROBDD's requires a computational time depending on the size of the ROBDD's. And although it is a NP-complete problem, and the size of the ROBDD can be exponential in the worst case, many practical functions remain small. Hence, it is often possible to deal with boolean equations in a efficient way.

It is known that the size of ROBDD's is depending on the ordering chosen for its variables. The problem of choosing an optimal variable ordering is itself co NP-complete but there is a simple heuristic based upon the topology of a circuit, giving good results. This heuristic is depicted in Algorithm 4.1. This algorithm was used to obtain a variable ordering.

### 4.3 Optimal Test Set Generation

When we want an optimal test sequence for a given set \( D \) of satisfiable defects we are looking for the smallest possible set of vectors which will cover all defects of \( D \). Of course there may be many different solutions of equal length but one of them will do. An example is given in Fig. 4.2. The reader can easily verify that this is only one of all solutions.
Algorithm 4.2. Heuristic for Input Ordering

compute the level of each circuit node in the graph, according to:
level (input node) = 1;
level (other circuit nodes) = max (its fanin nodes) + 1;

order_list = empty;

sort output nodes in decreasing order of levels.

foreach (output_node)
    fanin_order (output_node, order_list);

/* definition */
fanin_order(node, order_list) {
    if (node e: order_list) {
        sort fanins of this gate in decreasing order of level
        foreach (fanin)
            fanin_order (fanin, order_list);
        append (order_list, node);
    }
}

\[ \mathcal{F}_1 = a \]
\[ \mathcal{F}_2 = b \]
\[ 1 : \mathcal{F}_1 \bullet \mathcal{F}_2 \bullet \mathcal{F}_3 \quad a \bullet b \bullet c \]
\[ \mathcal{F}_3 = c \]
\[ 2 : \mathcal{F}_4 \quad \bar{c} \]
\[ \mathcal{F}_4 = \bar{c} \]

Figure 4.2. Example of an Optimal Test Set

Finding an optimal test sequence is obviously a set covering problem. We are looking for the smallest set \( T \) of test vectors such that \( T \) contains at least one element of the test vector sets. This statement is at least true for one pattern defects. It is equal to the Hitting Set problem which is NP-complete [11]. Two pattern tests do not seem to make the problem less difficult.

We are using boolean functions, and to find if there is a vector that is element of other boolean functions, we intersect them to check the outcome according to eq. 4.3. Hence, one could come up with a strategy in which to find the maximum clique test formula out of a set of defect
test formulas, then deleting the detected test formulas from the set and repeating that process. So the idea is to look for the largest cliques in the set of test formulas. However, this idea will not always result in an optimal solution as shown in the following example.

\[ T_1 = a \]
\[ T_2 = b \quad \text{Optimal Test Set} \]
\[ T_3 = c \]
\[ T_4 = d \cdot \bar{a} \quad 1 : a \cdot b \cdot c \quad b \cdot c \cdot d\bar{a} \cdot e\bar{a} \]
\[ T_5 = e \cdot \bar{a} \quad 2 : d\bar{a} \cdot e\bar{a} \cdot \bar{a}bc \quad \bar{a}bc \]
\[ T_6 = \bar{a} \cdot \bar{b} \cdot \bar{c} \quad 3 : \quad a \]

**Figure 4.3.** Optimal Test Set versus Maximum Clique Method

The strategy to choose, could be one of branch and bound. The bound could be simply chosen the size of the solution found so far. This method, however, has as a serious drawback. While starting a new computation which could possibly come up with a shorter solution, multiple test formulas are computed which could have already been computed in earlier times. And the computation of multiple test formulas can be a very expensive operation.

Another possibility is to compute all possible maximum clique formulas in advance. The problem that is left then is a NP-complete Minimum Cover problem [11], however, no operations have to be performed on test formulas anymore. This approach has the disadvantage that it computes maximum cliques that are never used or only partially used.

It seems clear that an optimal solution algorithm must combine the two approaches. This can be done by, during computation, keeping up a sort of memory function. This memory function holds earlier computed results in a some kind of easy accessible way. The total amount of operations on test formulas, represented by boolean equations, could thereby be reduced somewhat.

### 4.4 Heuristic Test Set Generation

We present a very straightforward algorithm in this paragraph. The idea of the algorithm is to place a test formula in a timeslot in a greedy
§4.4 Heuristic Test Set Generation

kind of way, i.e. place it in the first timeslot possible, starting with the first one.

One deviation is made in this strategy regarding to defects requiring a two pattern test. This is because they are likely to have a bigger impact on the final size of the solution. But more important, they require more operations on boolean equations representing the test formulas then single pattern defects, since both test formulas of the defect have to fit. If the first formula, the initialisation formula, can be placed in a timeslot \( x \), then the second formula has to be placed in timeslot \( x+1 \). If the latter is not possible, the computation for the placement of the first formula was a wast of time. So the little deviation is, that first, this class of defects are placed. When this is done, the test formulas of the single pattern defects are placed. This algorithm is shown in Algorithm 4.3.

The run time order of this algorithm, can be expressed in the number of operations performed on boolean functions representing the test sets. When we have a set \( D \) of defect formulas and the solution is found in \( T \) vectors, \( |D| \geq T \), vectors then there appears to be a worst case situation if there is only one test formula of a defect in every timeslot, hence \( |D|=T \). Obviously in that case, \( \sum_{n=1}^{n=T} n \) "and"-operations are performed on boolean functions. If all formulas can be placed in the first timeslot, \( T = 1 \), the number of operations is equal to \( |D| \). The algorithm shown in Algorithm 4.3 could be improved somewhat, by adding some extras. For instance, one could try to optimize the placement of the two pattern defects. Another possibility is, in case the last occupied timeslot is only occupied by one defect, to swap this defect with another, already placed defect, and trying to place the new defect in a other timeslot. The resulting test set would be one less in length.

4.5 Implementation notes

The software program written to generate a test for current testing, reads a file which holds the netlist description of a circuit for which to perform the test for. The input format is shown in Appendix 1. It builds a graph consisting of nodes, which represent the nets in the netlist and the transistors, and edges, which are pointers from nets to transistors and visa versa. The graph of the circuit is stored in a hashtable. The
Algorithm 4.3. Heuristic for Test Sequence Generation

DFS = \{ <IN_1, GO_1>, <IN_2, GO_2>, ..., <IN_K, GO_K> \}  
/* Set of test formulas for two pattern defects */

SFS = \{ J_1, J_2, ..., J_N \}  
/* Set of test formulas for single pattern defects */

Test_Seq = array[1..max_time_slot];

/* Place members of DFS first */
foreach (<J_in, J_go> ∈ DFS) {
    x = 1
    step1 :
    temp1 = Test_Seq[x] • J_in;
    if (temp1 ≠ 0) {
        temp2 = Test_Seq[x+1] • J_go;
        if (temp2 ≠ 0) {
            Test_Seq[x] = temp1;
            Test_Seq[x+1] = temp2;
            continue;
        }
    }
    else {
        x = x + 1;
        goto step1;
    }
}

/* Place all members of SFS */
foreach (J ∈ SFS) {
    x = 1
    step2 :
    temp = Test_seq[x] • J;
    if (temp ≠ 0) {
        Test_Seq[x] = temp;
        continue;
    }
    else {
        x = x + 1;
        goto step2;
    }
}
transistors, which are in fact nothing more than three pointers to their
drain, gate and source, are stored in an array.

After the graph has been constructed, an input ordering is performed
according to Algorithm 4.1. Then, the formulas $F_{i}^{\text{Gnd}}$ and $F_{i}^{\text{Vdd}}$ are
determined, for every node $i \in \text{circuit\_nodes}$ (see chapter 3). Note that,
presuming an FCMOS circuit, eq. 3.6 holds, therefore only $F_{i}^{\text{Vdd}}$ is
computed. The reason for this is that for computation of a formula for a
internal node or some test formula not the whole graph has to be
walked through but only computations have to be performed on circuit
nodes in the neighbourhood.

The defect sensitization is done according to the equations in chapter 3.
Which defects are handled is depending on which option is chosen at
the start up of the program. The most important option is that for the
layout driven defects. They are stored in a file which the user offers.
This file is expected to come from a yield extractor which was applied to
the layout of the circuit to be tested. The format of this file is also
shown in Appendix 1. Other options are the computation of all possible
bridging defects in the circuit or all stuck at faults. Both options try to
compute all open defects. In case the latter options are switched on, the
defects are stored in an file by the program for the user. Every
sensitized defect is represented in the program by its ROBDD pointer
and its identification number in a structure. Defects needing two
formulas, in this case the open drains and sources, have two of that
structures with the same identification number. If a defect can not be
covered, this is printed to standard error.

Finally, the heuristic algorithm of Algorithm 4.5 is used to create a
test. To represent the cliques computed, a set representation using
bitvectors is used. At the end, the computed cliques are satisfied, to
obtain a input vector. The result and some statistics are printed in a
file for the user.

4.6 Experimental Results

The previously described program, using algorithms and definitions
presented in this rapport, was implemented in C on a HP-9000/835
minicomputer. To present some performance results, the ISCAS '85
benchmark was used which contains a gate level description of
combinational circuits. Those gate level circuits were translated into a
FCMOS transistor netlist.

A problem arises from the fact that there are no defect files for these circuits which are layout driven. And, to perform a test to compute all bridging defects would take much too long since the circuits can be very large, and will result in thousands of defects.

However, to obtain at least some results, a defect input file was generated in a random way containing only bridging defects. The number of defects was a fixed percentage of the number of nets in the circuit. Also, not all the circuits of the benchmark were used, since a few of them are very big and would take an enormous amount of time. For each circuit, multiple randomly generated defect files were applied. The results are shown in Table 4.1.

<table>
<thead>
<tr>
<th>File</th>
<th># Defects</th>
<th># Transistors</th>
<th>Time</th>
<th># Vectors</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>c17</td>
<td>2</td>
<td>24</td>
<td>0:00.1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>c432</td>
<td>23</td>
<td>824</td>
<td>5:35.3</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>c432</td>
<td>23</td>
<td>824</td>
<td>4:29.5</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>c432</td>
<td>23</td>
<td>824</td>
<td>4:13.8</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>c880</td>
<td>49</td>
<td>1802</td>
<td>3:28.8</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>c880</td>
<td>49</td>
<td>1802</td>
<td>1:32.3</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>c880</td>
<td>49</td>
<td>1802</td>
<td>2:41.2</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>c1908</td>
<td>88</td>
<td>3446</td>
<td>26:51.3</td>
<td>5</td>
<td>100%</td>
</tr>
<tr>
<td>c1908</td>
<td>88</td>
<td>3446</td>
<td>34:50.1</td>
<td>5</td>
<td>100%</td>
</tr>
<tr>
<td>c1908</td>
<td>88</td>
<td>3446</td>
<td>30:28.5</td>
<td>4</td>
<td>98.9%</td>
</tr>
</tbody>
</table>

Although this run time test does not tell us much about the runtime of this program in reality, some things can be concluded from this test.

It revealed for instance the need for a layout driven defect input, so "impossible" defects are not sensitized, therefore saving CPU time.

Also it appeared that a lot of sensitized defects can be collapsed, that is, they imply each other and can therefore be omitted. When these defects are omitted from the defect file, less operations will be performed on boolean formulas, hence, saving CPU time.

When looking at the result file, two things can be noticed. First, the length of the test set appears to be very small, as was expected.
Secondly, most of the defects are placed in the first timeslot of the test set. In most cases, over 40% was placed in the first timeslot thus making the worst case analysis of the heuristic algorithm a theoretical one.

We emphasize that a layout driven and collapsed defect set is obliged, to draw any conclusion regarding the runtime of this implementation.
Chapter 5

Conclusions and Discussion

This chapter is devoted to present the conclusions that can be drawn from this rapport. Some aspects which came up during the research are discussed.

5.1 Conclusions

From the theoretical reasoning and practical work presented in this report, we conclude the following:

- Current testing will detect a vast majority of in a manufacturing environment induced defects.

- Test sets for current testing are very small compared to the number of defects they are able to test.

- To avoid the test generation to be very time consuming it is important to let the set of defects which should be tested for be as small as possible. This can be achieved by satisfying the following:
  
    i) The set of defects should be extracted from the layout of the circuit.

    ii) The number of defects should be reduced by applying techniques to collapse defects.

5.2 Discussion

Some aspects are discussed now, which could be used to improve test generation for current testing as presented in this report.
5.2.1 Defect Dominance
The approach followed in this report was made under the assumption of an appropriate fault model compared to traditional fault models. According to that model, defects are sensitized which result in test formulas which represent the set of input vectors detecting a defect. The number of test formulas obtained using the approach of this report, is equivalent to the number of defects presented.

The advantage of formulas is that they can be intersected to obtain a shorter test set. The intersection, though, may take exponential time. Therefore, it is important to reduce the number of test formulas. The technique used to achieve this, is called defect dominance. If for two defects $I$ and $k$ their test formulas $F_I$ and $F_k$ are related as in $F_I \subseteq F_k$, then defect $I$ is said to be dominated by defect $k$. Consequently, it is not important to produce a test vector for defect $I$ since a test vector for $k$ will always detect defect $I$. The implication operation can be done rather efficiently by using a different algorithm [12]. However, it is better to first reduce the number of defects, before they are actually sensitized, then doing this after the test formulas are constructed.

In the approach presented in this paper, test formulas are depending on the primary inputs of the circuit. Let us look now at the circuit of fig. 5.1.

![General Circuit with Circuit Variables](image)

**Figure 5.1.** General Circuit with Circuit Variables

A different method would be to first describe the defects 1 and 2 in terms of circuit variables $S_1, ..., S_k$. This would result in a much smaller
ROBDD for this defects, hence, dominating defects can be discovered much faster then describing them in primary inputs. After omitting dominated defects, the resulting defects can be described in circuit variables $S_{k+1}, \ldots, S_n$, including the defects 3 and 4. The process repeats itself.

Practical test revealed that there may be a significant number of dominated defects in a normal defect set, therefore unnecessarily consuming CPU time and test set length. Using this method, the test generating for current testing can be improved.

5.2.2 Other Heuristics

The heuristic for actually generating the test set, Algorithm 4.3, is a greedy one. The goal of this, is to reduce the resulting test length. The results obtained, made clear that the test lengths are indeed very small. This holds for sure when comparing this results to results one would probably get from using test based on a stuck at model. Hence, it is not necessary to "hunt" for a as short as possible solution.

This suggest that heuristics can be derived that result in a less smaller test length then now is obtained, but require less computation time. A heuristic what would result in this, is to try to place only a limited number of test formulas in a certain timeslot. Hence, fewer redundant operations are performed that would result in a not satisfiable multiple test formula. This seems to be justified when noticing that in some runtime cases the actual test set generating algorithm consumed more CPU time then the construction of the test formulas of the defects.

Applying such algorithms is expected to result in longer, but still very efficient test set, with preferable run times.
Appendix

Input Formats
Netlist Input Formats

< type > = P | N;
< node > = unsigned int;
then,
< line > ::= T Drain Gate Source;
where T of < type > and Drain, Gate, Source of < node >.

Defect Input Formats

< line > ::= B Node1 Node2 | /* Bridge */
            S Transistor | /* Open Drain/Source */
            G Transistor ; /* Open Gate */

where Node1, Node2, Transistor of unsigned int.
References


