EINDHOVEN UNIVERSITY OF TECHNOLOGY

Automatic generation of in-circuit tests for board assembly defects

Citation for published version (APA):
van Schaaijk, H., Spierings, M., & Marinissen, E. J. (2018). Automatic generation of in-circuit tests for board assembly defects. In *Proceedings - 2nd IEEE International Test Conference in Asia, ITC-Asia 2018* (pp. 13-18). Article 8462941 Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/ITC-Asia.2018.00013

DOI:
10.1109/ITC-Asia.2018.00013

Document status and date:
Published: 15/08/2018

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 15. Aug. 2024

Automatic Generation of In-Circuit Tests for Board Assembly Defects

Harm van Schaaijk[1,2]
Martien Spierings[1]
Erik Jan Marinissen[2,3]

[1] Prodrive Technologies
Science Park Eindhoven 5501
5692 EM Eindhoven
The Netherlands
harm.van.schaaijk@prodrive-technologies.com
martien.spierings@prodrive-technologies.com

[2] Technische Universiteit Eindhoven
Den Dolech 2
5612 AZ Eindhoven
The Netherlands
h.a.h.v.schaaijk@student.tue.nl
e.j.marinissen@tue.nl

[3] IMEC
Kapeldreef 75
3001 Leuven
Belgium
erik.jan.marinissen@imec.be

Abstract — **The components and the solder joints that are made during assembly to hold components to their printed circuit board can suffer from defects and therefore need to be tested. Many research papers on board-assembly testing focus on boundary scan test, processor-controlled test, or other powered digital testing techniques that mostly ignore the indispensable passive circuits and that can incur damage that could have been avoided by executing a non-powered test first. In-circuit testing is a non-powered test method that applies stimuli and measures responses using probe needles. However, often used self-learning solutions for designing these tests need a known-good-board, entailing significant disadvantages. In this paper, a software tool is described that automatically generates in-circuit tests based on the product design files, without requiring probe access on every net. Furthermore, the tool indicates where on the board fault coverage is not maximal, and hence where extra probe access will improve the test quality.**

1 Introduction

In board assembly, electronic components are soldered onto a printed circuit board (PCB). Boards contain a mix of passive and active discrete components, as well as (digital) integrated circuits (ICs). A typical board at Prodrive Technologies contains 2500 passive, 200 active, and 150 IC components. We distinguish components based on through-hole technology (THT) and surface-mount technology (SMT). THT components, of which the leads are inserted into holes in the PCB, have been largely replaced by the much smaller SMT components which are placed on top of the board.

Board assembly suffers from defects, and hence the assembled PCBs need to be tested. The bare PCB as well as the components are assumed to be fault-free, typically achieved by manufacturing tests by their respective suppliers. Hence, what need to be tested after board assembly is whether or not the components are correctly soldered onto the PCB. Defects considered are expressed in the PCOLA-SOQ [1] score; this method assigns a coverage score to Presence, Correctness, Orientation, Liveliness, and Alignment properties of each device on the board and Short, Open, and Quality score to all the connections on the board.

Many research papers on board-assembly testing focus on "*non-intrusive*" testing techniques [2] such as boundary scan tests [3] or processor-controlled tests. However, such papers typically ignore that discrete components are often an indispensable part of a board design (e.g, in a power supply unit). Passive components can typically only be verified to be 'present' with such non-intrusive test techniques. Furthermore, these non-intrusive methods typically require the board to be powered and therefore might incur damage that could have been avoided by executing a non-powered test first.

In-Circuit Testing is a board-assembly test method in which components are electrically tested through probing with an ICT tester. Such a tester has a bed-of-nails, i.e., a fixture with probe needles in locations corresponding to the probe locations of the board-under-test. These probes can be connected via a switch matrix to a parametric test instrument (e.g., a source measurement unit (SMU)). Ideally, every component-under-test (CUT) has individual probe access, i.e., its two leads can both be probed, and does not have false paths. If that is indeed the case, the CUT can simply be measured. However, this is not true in the reality of actual boards. Probing can be done on (1) the leads of THT components, and/or on (2) dedicated probe pads. Probing on the leads of SMT components is not recommended, as for example an open solder connection might be masked under the temporary force of the probe needle. Consequently, we have a limited number of probe locations, due to which we are sometimes forced to measure multiple components together.

So-called "self-learning" solutions [4] characterize a board based on the impedance patterns between the available probe locations, and compare those to the impedance patterns obtained for a known-good ("golden") board. Serious drawbacks of self-learning are the identification of a golden board and the fact that the test coverage of the various components on the board remains undetermined.

In this paper, we describe the first stage of the development of a software tool that automatically generates, based on a boards design netlist, in-circuit tests for board assembly defects of passive components, to be applied through a bed-of-nail tester. In this stage only passive components are considered. Support for other components will be implemented in future iterations. We use graph-theoretic algorithms to identify probe locations and assignment of test signals to those probe locations such that (1) the granularity of the clusters is as small as possible, and (2) any alternative ('false') paths between two target probe points are neutralized by guarding them with additional probes. The software tool also indicates where on the PCB fault coverage is not maximal, and hence where design-for-test (here: additional probe points) can help to improve the test quality.

This paper is organized as follows. In Section 2 the inputs of the tool are described. Section 3 describes how the netlist is converted into a graph. Subsequently, in Section 4, the test generation algorithms are described. Finally, Section 5 shows that the solution proposed in this paper is actually viable by applying it on actual boards.

2 System Inputs

The test is generated based on the the design files of the device-under-test (DUT). The data that is needed to describe the DUT consists of components information, such as the reference designator (refdes) that is used to distinguish each component instance (In this paper IEEE Std 315-1975 [5] is used), the component type (e.g. resistor, capacitor) to determine what type of test is needed, the value and tolerance of the component which is needed to verify if the correct component

is placed on the board and how accurately to measure. Furthermore the mounting technology of the component to determine if the pins of the components can be probed, and the nets that describe the interconnections between the components on the board are needed.

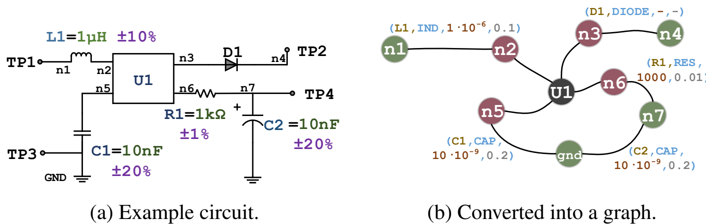
The possibilities to test components are limited by the accuracy and options of the board tester. To make the tool independent of different board testers, a configuration file describing the capabilities of the hardware should be provided to the tool. This file should contain: (1) the voltage and frequency range of the signal generators of the board tester, (2) the modes of the signal generators (e.g. constant voltage, constant current), (3) the sensitivity of the tester to determine if a voltage or current is large enough to measure reliably, (4) The impedance ranges to determine if a resistive, capacitive or inductive component can be measured by the hardware of the tester, (5) The impedance sweet spot in which the tester yields the most reliable tests, (6) the maximum guard ratio, i.e. the amount of current flowing through the false paths versus the amount of current flowing through the path that we try to measure.

3 Netlist Handling

For further computation, we convert the EDIF 2 [6] netlist (containing a set of components $\mathcal{C}$ and nets $\mathcal{N}$) into an undirected multigraph $G = (V, E)$, where V denotes the set of vertices and E the set of edges. Contrary to what one perhaps intuitively would expect, we found that in our application representing components by edges (as passive components have an in/out degree of two) and nets by vertices (as both nets and vertices can have an arbitrary degree) works best.

Multi-pin components, such as transistors and ICs, do not fit this representation; since the internal structure of ICs is unknown, representing these components by numerous edges between all the connected nets would blow up the number of edges, while these edges do not have a defined meaning. Therefore these components are represented by a vertex as well, with edges representing the pins of the component.

Furthermore, it is important to represent the properties of the nets and components in the graph. Therefore the graph is extended with a list of all nets that have probe access $P \subseteq V$ and a function $L : E \rightarrow \{l \mid l = (r, c, v, t, p)\}$ which yields the label of an edge, containing the refdes r and component type c, alongside the value v and tolerance t for passive components or the pin p for active components.



(a) Example circuit. (b) Converted into a graph.

Figure 1: Netlist handling example.

An example netlist and its corresponding graph are shown in Figure 1. Nets which have probe access are indicated as green vertices, while nets without probe access are shown as red vertices. The edge labels of the components are shown next to the corresponding edge. Edge labels of pins are omitted.

4 Test Generation for Passive Components

After the graph is generated tests should be generated for the components in the graph. Since not every component can be tested in isolation, either due to components being in parallel or due to a lack of probe access, components might have to be tested in groups of two or more components. Therefore we introduce the notion of a structure. A structure is defined as the smallest group of components that have to be tested together. A structure containing a single component is defined as an isolated structure, all other structures are defined as compound structures. Since tests will be generated for each structure, this paper uses the term structure-under-test (SUT) instead of CUT. In order to generate tests for the structures, first a pre-processing step is performed on the graph that determines all the information about the structures that is needed to generate tests. After that, tests will be mapped onto the structures.

4.1 Graph Pre-Processing

In the pre-processing step it is determined: (1) which components have to be tested together, i.e. which components will be grouped in a structure, (2) the basic electrical properties of each structure, (3) the circuit surrounding each structure that might influence the measurement, and (4) The highest voltage that can safely be applied to a net.

For one test a structure might be the SUT, while for the next test that same structure might be part of a false path. In both scenarios it is important to know the overall impedance of the structure (e.g. to determine the guard ratio). Furthermore, since each component inside of the structure has a tolerance, the impedance of the structure has a maximum deviation that is important to know when mapping tests onto the structure.

Since the structures are tested while already assembled onto the PCB, other structures than the SUT might form a conducting path between the measurement probes of the SMU. These false paths will likely influence the test results if no precautions are taken. Hence, for each structure we would like to known which structures are part of such a false path and how to mitigate the influence of these paths.

Finally, even though this paper focuses on generating tests for passive components, the active components cannot be ignored. The active components on the board might start influencing a test if its PN-junctions are excited or might even get damaged by applying a too high voltages to a net. In order to prevent both, the maximum voltage that should be used by the SMU should be determined.

In order to get this data, the graph is first converted into an *impedance graph*, that describes the impedances between nets. Next the unreachable components in this graph are combined into reachable structures using a *collapse* procedure. For each of these structures in the graph the false paths are determined, which are then placed in a new graph. In some cases these subgraphs can be collapsed even further. Next for each structure the *minimal guard set* is determined, which describe all possible ways to mitigate the influence of the false paths. Finally the active components are considered again in order to determine the maximum voltage that can be used safely. These steps are explained in detail in Sections 4.1.1 to 4.1.5. Finally Section 4.1.6 shows an example of the pre-processing procedure.

4.1.1 Impedance Graph

An impedance graph is defined as a graph where the nets are represented as vertices, and the edges represent a (frequency dependent) impedance between the nets connected by the edge.

We assume that if a the voltage that is is used to test passive components is low enough (e.g. 200mV), the influence of the active components can be ignored since the PN-junctions in the components are not excited. For all passive components c the impedance $Z_c(f)$ as a function of the frequency f can be determined.

To create the (simple) impedance graph, first all vertices that represent a net are copied to a new graph. Next based on the component type and the value of the label of each edge a new impedance edge e_i is created. Next the edge is added to the impedance graph if there is not already an edge e_{exist} between the vertices, otherwise e_i and e_{exist} are combined into a new edge e_{new}, for which it holds that $Z_{\text{new}}(f)$ will be equal to the impedances $Z_i(f)$ and $Z_{\text{exist}}(f)$ in parallel. Algorithm 1 shows the pseudo code.

```
Data: Graph G = (V, E)
Result: Impedance graph G_i = (V', E')
   // Copy all vertices that represent a net
1  G_i := ({v|V ∈ N} , ∅);
   // Combine all parallel edges
2  foreach e = {v_1, v_2} ∈ E do
3  |   e_i := GetImpedanceEdge(e);
4  |   if e_exist := G_i.GetEdge(v_1,v_2) ≠ null then
5  |   |   e_new := CombineParallel(e_i, e_exist);
6  |   |   G_i.SwapEdge(e_exist, e_new);
7  |   else
8  |   |   G_i.AddEdge(e);
9  |   end
10 end
```

Algorithm 1: Impedance graph algorithm.

4.1.2 Graph Collapsing

Next the graph is collapsed, The goal of the collapsing process is to combine non reachable structures into as much reachable compound structures as possible. It does this while maintaining a valid representation of the netlist, i.e. expanding the compound structures will yield the original graph.

Algorithm 2 shows the pseudo code of the collapse algorithm. It consists of three steps. First all vertices are copied from G to G'. Thereafter all edges that are connected to a vertex with a degree of two that do not have probe access are combined. Note that the collapsing process constantly joins structures that are either in series or in parallel. This allows to determine the impedance of the collapsed edge easily.

```
Data: Impedance graph G_i = (V, E)
Result: Collapsed graph G_c = (V' ⊆ V, E')
   // Copy all vertices and edges
1  G_c := (V, E);
   // Collapse nets without probe access and a degree d(·) = 2
2  ℂ := {v | v ∈ V' ∧ v ∉ P ∧ d(v) = 2};
3  while ℂ ≠ ∅ do
4  |   v := v ∈ ℂ;
5  |   e_1 = {v, v_a} := AdjacentEdges(v)[1];
6  |   e_2 = {v, v_b} := AdjacentEdges(v)[2];
7  |   e_new := CombineSeries(e, e_2);
8  |   if e_exist := G_c.GetEdge(v_1,v_2) ≠ null then
9  |   |   e_new' := CombineParallel(e_exist, e_new);
10 |   |   G_c.SwapEdge(e_exist, e_new');
11 |   else
12 |   |   G_c.RemoveEdges(e_1, e_2);
13 |   |   G_c.AddEdge(e_new);
14 |   end
   |   // Degree of v_a and v_b might be changed
15 |   if v_a ∉ P ∧ d(v_a) = 2 then ℂ.Add(v_a);
16 |   if v_b ∉ P ∧ d(v_b) = 2 then ℂ.Add(v_b);
17 end
```

Algorithm 2: Collapse algorithm.

4.1.3 False Paths

A structure is tested by connecting a source probe to one of the vertices of the structure, a measurement probe to the other, and executing a test routine. However, since the structures are tested while having been soldered onto a PCB, there might be paths other than the SUT that allow current to flow from source to measurement. These paths are called false paths, which are defined as:

Definition 1. *A path p is a false path of an edge $e = \{s, m\}$ if and only if p is a s, m-path (denoted as $s \overset{p}{\rightsquigarrow} m$), and p is a simple path i.e. $\forall_v[v \in p : \#_v(p) = 1]]$.*

Note that false paths can only be simple paths; a net cannot have multiple voltages at the same time and current cannot flow through it more than once. A closer look at these false paths reveals an interesting property, viz. that edges being in a false path of the SUT is both a reflexive and symmetric relation. If one observes that each false s, m-path of an edge $e = \{s, m\}$ together with e itself forms a simple cycle, the rest of the proof is trivial. From this it follows that the connected components can be split into subgraphs that for each edge e contains e itself and all of its false paths.

This split happens to be equal to the biconnected components [7] of the graph. Biconnected components (BCC) of a graph are the sections held together by the so called articulation points. Articulations points and BBCs are defined as:

Definition 2. *A vertex v is an articulation point $v \in \mathbb{A}(G)$ of a graph G if and only if $\exists_{u,w}[u, w \in V(G) \land u \neq w : \forall_p[u \overset{p}{\rightsquigarrow} w : v \in p]]$*

Definition 3. *For two BCCs $B_1, B_2 \in \mathbb{B}(G)$ with $B_1 \neq B_2$ of a graph G it holds that $\exists_v[v \in \mathbb{A}(G) : \forall_p[u \in B_1 \overset{p}{\rightsquigarrow} v \in B_2 : v \in p]]$*

Theorem 1 shows the proof that the set of edges consisting of the SUT and edges in its false paths is equal to the biconnected component that holds the SUT. So, to find all edges that are part of a false path of a structure, a biconnected component algorithm, such as the one proposed by John Hopcroft and Robert Tarjan [8], can be used.

Theorem 1. *For a graph G, let $e \in E(G)$, where $\mathbb{P}_e$ is the sets of all false paths of e, then: $\{e_1 | e \in \mathcal{B} \in \mathbb{B}(G) \land e_1 \in \mathcal{B}\} = e \cup \mathbb{P}_e$*

Proof. Let $e = \{s, m\} \in E(G)$ be a SUT. Now, if a random traversal following s, n-path $p \not\ni e$ is started, then anywhere in the path it either holds that an articulation point has been crossed i.e. $\exists_v[v \in p \land v \in B \in \mathbb{B}(G) : s \notin B \lor m \notin B]$ meaning that it is neither possible to return to s nor m without crossing the articulation point again, or not.

- An articulation point v has been crossed:
  - By the definition of biconnected components the path has reached some other biconnected component.
  - Assume that p is part of a false path of e. A false path of e concatenated with e itself creates a simple cycle, meaning that there are at least two distinct s, p_n-paths. This contradicts with the fact that v is on every s, p_n-path and therefore it can be concluded that p cannot be part of some false path of e.
- No articulation point has been crossed:
  - By the definition of biconnected components this path lies within a single biconnected component.
  - There exists a s, p_n-path, namely p, and thus a simple s, p_n-path p_s equal to p with its cycles removed. There also has to exist a p_n, m-path q since no articulations are crossed, and thus a simple p_n, m-path q_s. It follows that there has to exist a path $\hat{p} = s \overset{p_s}{\rightsquigarrow} p_n \overset{q_s}{\rightsquigarrow} m$. However, for $\hat{p}$ to be a false path of e, p_s has to be distinct from q_s otherwise $\hat{p}$ is not simple. So, assume that $\hat{p}$ cannot be simple, then there has to exist a vertex $v \in p_s$ that is on every possible n, m-path. This implies that an articulation has been crossed, contradicting the fact that p reached p_n without crossing one. By contradiction it holds that there has to exist a simple path $\hat{p}$, from which it follows that $\hat{p}$ is a false path of e, and therefore that all edges in p_s are part of a false path of e. Since for ever vertex $v \in p$ there exists a simple s, v-path it also holds that all edges in p are part of a false path of e.

It can be concluded that if a path crosses an articulation point it has reached a different biconnected component and it can no longer be a

false path of e. As long as the path does not cross an articulation point and is therefore within a biconnected component, the edges of that path have to be part of a false path of e. Therefore it can be concluded that the set of edges inside the same biconnected component as e is equal to e together with the edges in the false paths of e. □

4.1.4 Minimal Guard Set

If a false path is not guarded it likely will influence the measurement. However, if too many nets are guarded in a false path, components are shorted between two guarded nets, which negatively influences the guard ratio. Moreover, the ideal set of nets that are guarded depends on the selected frequency of the SMU e.g. in a path consisting of a capacitor, a resistor, and an inductor it is likely better to guard before the resistor for low frequencies, while for high frequencies the impedances would be better balanced if guarded after the resistor.

Data: Biconneced component $B = (V, E)$ and SUT $s = \{v_s, v_m\} \in E$
Result: Minimal Guard Set of s

1 B.RemoveEdge(SUT);
2 **call** GetSetsRecursive(∅,∅);
3 B.AddEdge(SUT);
4 **Function** *GetSetsRecursive(Selected, DoNotSelect)*
5   $v_{\text{impassable}}$:= Selected;
6   p_{short} := B.GetShortestPath(v_s, v_m, $v_{\text{impassable}}$);
7   **if** p_{short} = *null* **then**
8   guardCandidates := $\{v | v \in p_{\text{short}} \land v \in \mathcal{P}\} \setminus$ DoNotSelect;
9   **if** *guardCandidates* == ∅ **then**
10   IgnoreInvalid(Selected);
11   **end**
12   **foreach** $v \in$ *guardCandidates* **do**
13   selected' := Selected ∪ v;
14   doNotSelect' := DoNotSelect ∪ guardCandidates;
15   **call** GetSetsRecursive(selected', doNotSelect');
16   **end**
17   **else**
18   ReportSolution(Selected);
19   **end**

Algorithm 3: Minimal Guard Set Algorithm.

The goal of the minimal guard set algorithm is to find all possible sets of vertices that guard all false paths with as few components shorted between guards as possible. Essentially, for every structure $e = \{s, m\}$ that needs to be tested, the algorithm has to find all minimal s, m-vertex-cuts of the biconnected component of e.

The pseudo code of how this can be accomplished is shown in Algorithm 3. The algorithm works by exhaustively finding the shortest path parallel to the SUT, and selecting a vertex from that path to guard and mark that vertex as impassible, and marking the rest of the vertices as *do not select*. If after some iterations no parallel path can be found, then the selected edges form a valid guard set (since there does not exist any paths that are not guarded). However, if there is still a path, but none of the vertices in that path can be probed, the (partial) solution is invalid (since there is a false path that could not be guarded) and can be ignored.

Constantly selecting the shortest path and making it impossible to select other vertices from that path ensures that no components will be shorted between guard probes. Selecting a net n_1 to guard a path $p_s \ni n_1$ might automatically guard a longer path $p_l \ni n_1$, thus first guarding the longer path with a net $n_2 \neq n_1$ would result p_l to be guarded more than once. Furthermore, first selecting n_1 and then $n_2 \in p_s$ would result p_s to be guarded more than once.

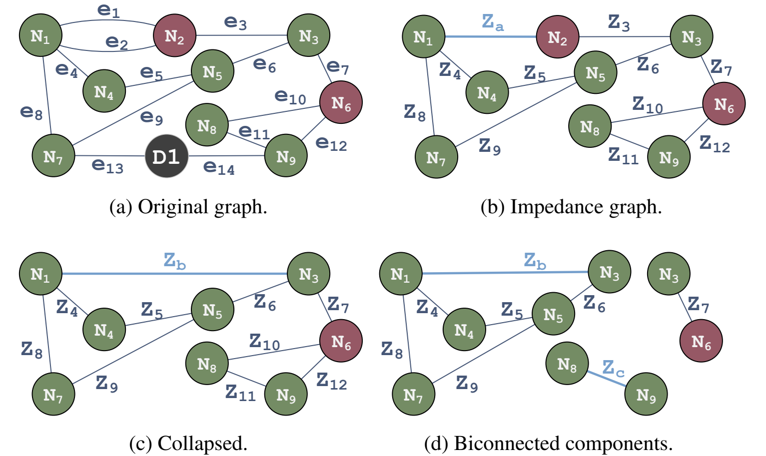
4.1.5 Maximal Voltage

The maximum voltage that can be safely applied to a net is typically limited by the active components that can be reached from that net. As mentioned in Section 4.1.1 a maximal safe voltage can be assumed in order to prevent damage. However, if all paths between a source probe and active components are forced to a potential of ∼0V, this is not necessary. Guard and measurement probes exactly do this.

Nets can be determined to be limited by a safe voltage by verifying that an active component can be reached from the source probe without passing a guard or measurement probe. This can be simply executed by starting a Depth First Search (DFS) from the source vertex in the original graph with the guarded vertices and measurement vertex marked as impassible. If the DFS can reach an active component, the voltage should be limited to the assumed maximal voltage. If no active components can be reached, the voltage is only limited by the capabilities of the board tester.

4.1.6 Pre-Processing example

Figure 2 shows an example of the pre-processing steps, applied to the graph shown in Figure 2(a). Figure 2(b) shows the original graph converted to an impedance graph, in which parallel edges e_1 and e_2 are combined and the active component D1 is removed. Since vertex N_2 now has a degree of two and does not have probe access, the two edges Z_a and Z_3 are collapsed, as shown in Figure 2(c). Finally the biconnected components are determined (and collapsed again) in order to get the false paths of each structure, as shown in Figure 2(d).



(a) Original graph. (b) Impedance graph.

(c) Collapsed. (d) Biconnected components.

Figure 2: Pre-processing example.

Figure 2(c) and Figure 2(d) make clear why collapsing has to performed twice; once before determining the BCCs to make sure that non-reachable components are not separated, and after as well, since edges Z_{10}, Z_{11}, and Z_{12} can be collapsed into a single structure since $d(N_6) = 2$ in the biconnected components.

If we want to find the minimal guard set of for example compound structure Z_b, then the first shortest path we find is N_4, N_5 and the seconds path N_7, N_5. Therefore the algorithm will yield minimal guard sets $\{\{N_5\}, \{N_4, N_7\}\}$. Using the first set vertex D1 can be reached from N_1 without passing a guarded or measured (N_3) vertex and the maximal voltage to test Z_b should therefore be limited to the maximal assumed safe voltage. This is not the case if the latter guard option would be used.

4.2 Test Mapping

Isolated structures contain only a single component, and therefore it is easy to create a library of tests that can be mapped on the isolated structures of the DUT to cover all the cases. However, for compound structures this is not true since these can consists of multiple components, in multiple different combinations viz. the number of compound structures that can be found is (theoretically) infinitely big. To maximize the test coverage a generic test method is needed that tries

to map a test on a structure if there is no test found in the test library. We deploy both solutions to get the best of both worlds.

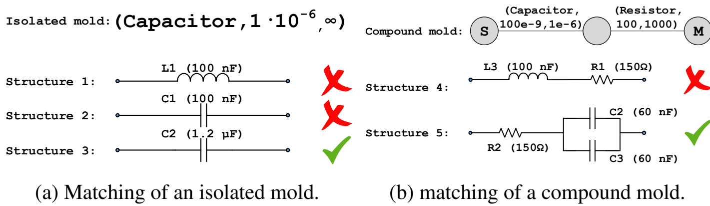
4.2.1 Library Test Mapping

The library test mapping procedure maps tests from a pre-defined test library onto the structures of the DUT. This library contains product-independent tests, which all specify the following: (1) if the test is suitable for mapping on structures that have a false path, (2) a so called *mold* which specifies which combinations of components can be covered by the test, (3) a test mapping procedure, and (4) a test effectiveness report.

Only if a structure complies with the mold the mapping algorithm will continue to try to map the test onto the structure. Since there are both isolated and compound structures, there are also isolated and compound molds. Isolated molds are just a description of which component type and the range of the value that can be covered by the test. The compound molds are small graphs with a component type and a value range in all edge labels, and a definition of which of the vertex S will be connected to the source probe, and which vertex M to the measurement probe.

If an (expanded) structure $e = \{v_1, v_2\}$ and a mold are isomorphic [9] while preserving either $S \mapsto v_1 \wedge M \mapsto v_2$ or $S \mapsto v_2 \wedge M \mapsto v_1$, and all of the labels of the mold and structure match, then the mold matches and test generation procedure continues. The test generation procedure itself (e.g. choosing the right test type, frequency for the SMU, etc. is out of the scope of this paper).

Figure 3(a) shows an example of a isolated mold. The mold specifies that the component has to be a capacitor with a value of at least $1\mu F$. Unlike Structure 3, both Structures 1 and 2 do not comply with this mold since either the value or component type does not match. Figure 3(b) shows an example of a compound mold. The mold consists of a capacitor-edge between $100nF$ and $1\mu F$ connected to S, and a resistor-edge between 100Ω and 1kΩ connected to M. Structure 4 does not comply with this mold since it contains an inductor. Structure 5 does match since the combined value of the capacitors is within the specified range and the orientation does not matter.



(a) Matching of an isolated mold. (b) matching of a compound mold.

Figure 3: Mold matching.

4.2.2 Generic Test Method

All structures that could not be covered by a test defined in the test library are handed to the generic test method (GTM). The generic test method is a method that tries to map a test as effective as possible, onto the structure with as few test steps as possible regardless of what is inside that structure. The generic test method can only perform impedance measurements and is therefore more limited than the tests in the library.

To determine the frequencies to test a structure at, the generic test method has to determine at which frequencies the components in the structure are *visible*. Just like the components inside the structure, the impedance of the structure itself has a tolerance. Using modified nodal analysis [10] (an algorithm using the Kirchoff voltage and current laws) the currents through and voltages over each component can be determined. If the $\frac{i_c}{i_s}$ ratio, where i_c is the current through a component and i_s the current though the complete structure, is smaller than the tolerance of the structure t_s, the component is defined to be current-*dominated*. We can therefore not determine its presence. If a component is voltage-dominated ($\frac{v_c}{v_s} < t_s$), we neither determine if the component to be correct nor live.

Using this principle, the PCOLA-coverage can be determined for each of the frequencies by scanning over the frequency domain of the board tester and determining if components are dominated or visible. By selecting as few as possible frequencies that result in maximal coverage, the efficiency can be optimized. In order to determine the guard ratio, the impedance of the false path is determined next. Finally a suitable voltage is selected that is lower than the determined maximal voltage, but high enough for the board tester to perform a reliable measurement. From all these potential tests, all tests that are superfluous are removed

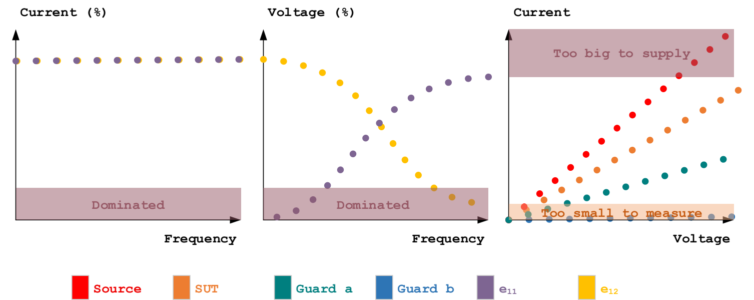


Figure 4: Example of the generic test method.

Figure 4 shows an example of the generic test method applied on a structure consisting of a resistor can capacitor in series. Since the components are in series the current through both components is always 100% of the total current. The voltage over each of the components does depend on the frequency and therefore the resistor is voltage-dominated for low frequencies, while visible for higher frequencies. If the guard ratio for a given frequency is within the range specified by the board tester configuration file, a voltage can be selected. In this case the voltage is not limited due to active components, but by the board tester. If a single frequency somewhere in the middle of the frequency range is chose, both components have optimal coverage and the hardware of the tester is able to supply the current.

5 Experimental Results

5.1 Setup

We selected six boards for consumer and industrial applications from the Prodrive Technologies product catalog for our experiments. We used our tool to automatically generate tests. The board characteristics are shown in Table 1. The QA board is specially designed for qualification of the ICT tester hardware. The EQDM board is designed towards maximum ICT access. The test generation software ran on a PC with a Core i7-6820HQ CPU at 2.7 GHz and 8 GB of DDR4 RAM at 2.133 GHz. Reported compute times are measured using the C# System.Diagnostics.Stopwatch class.

Prodrive Technologies' ICT board tester is capable of generating and measuring signals in a range of [0Hz, 150kHz]. It can measure in the following ranges: resistance: [10Ω, 150kΩ]; capacitance [200pF, 1mF]; inductance [10μH, 1H].

To take the generic test method into account, the library of tests is deliberately kept small. It contains a test for parallel capacitors (C+), large capacitors (C_{big}), small capacitors (C_{small}), resistors (R), and

Table 1: Selected boards.

ID	Type	#Components Active	Passive	#Nets Total	Access
QA	Qualification	17	201	268	267
ETMA	Consumer	42	129	57	26
MIUC	Control	338	1006	847	630
EQDM	ICT enabled	229	2799	1977	1470
PACB	Control	189	1045	695	564
ECU	Automotive	343	407	750	744

Table 2: Pre-processing algorithms.

	$\lvert\mathbb{B}\rvert$	Biggest	$\lvert\mathbb{C}\rvert$	Isolated *FP*	Isolated $\neg FP$	Compound *FP*	Compound $\neg FP$	Time
QA	147	10	39	118	13	27	4	1s
ETMA	55	12	51	43	13	8	13	1s
MIUC	370	63	295	288	262	43	109	14s
EQDM	663	183	742	765	767	100	276	626s
PACB	312	112	318	230	347	48	114	133s
ECU	99	122	111	134	78	56	12	129s

both a test for a capacitor and resistors in series (R-C) and in parallel (R||C).

5.2 Pre-Processing

Table 2 shows the results of the pre-processing algorithms. Its subsequent columns show the number of bi-connected components $\mathbb{B}$, the size of the largest biconnected component, the number of collapsed vertices $\mathbb{C}$, and the number of structures. These structures are both categorized by isolated or compound and if there is a false paths. Finally the computation time is shown as well.

The pre-processing of the EQDM board takes about 10 minutes, which is almost completely spend on the minimal guard set algorithm. This reduces the board the complexity of the graph by factor of 16 (from 3028 components to 183).

5.3 Test Generation

Table 3 shows the results of the test generation procedure. For each of the six boards the number of components covered by a test is shown. The GTM column shows the number of tests that are handled by the generic test method. The NA column shows the number of component for which currently no test could be generated.

Note that using this limited library of tests a substantial part of components is covered by a test from the library. When that is not the case (e.g. EQDM) the generic test method takes over. This results in few components that are not covered.

5.4 Test Execution

The QA and ECU boards are at the time of writing the only boards which can be connected to the ICT tester, for the other boards no fixture is manufactured yet. Therefore the test are only executed these boards.

The tests generated for the ECU board are executed using set of 15 boards. 252 of the 299 tests (84,28%) proved to be stable with a $C_{pk} \geq 2$ [11], 47 tests required a minor change to e.g., limits or frequency in order to get a stable test. The generated test for the QA board ran successfully, none of the tests steps proved to be unstable.

Table 3: Test generation results.

	C+	C_{big}	C_{small}	R	R||C	R-C	GTM	NA	Time
QA	27	6	27	60	30	-	65	-	45ms
ETMA	36	-	1	22	12	-	55	36	28ms
MIUC	122	6	23	205	37	-	573	31	3406ms
EQDM	579	12	579	394	166	30	1144	285	8606ms
PACB	154	4	16	124	58	12	578	111	3226ms
ECU	3	-	46	139	60	-	36	15	567ms

6 Conclusion

The graph representation of a board design has proven to be a versatile tool that allows for relative easy computation of the the false paths and the impedances between certain nets. It allows unreachable structures to be collapsed into reachable compound structures and decomposing the graph into its biconnected components the. These sub-graphs are the basis of the minimal guard set algorithm, which finds all possible combinations of vertices to guard. If these guard sets are computed, it can then be determined what the maximal voltage is that can be applied to a net, using the original graph.

Tests in a pre-defined library of tests can be mapped onto the components in the DUT. These tests specify a so called mold, which determines what kind of structures can be covered by that test. If the mold matches the mapping procedure can continue. For all cases that are not covered by tests in the test library a generic test method is developed. Even though this method can only perform impedance measurements it proved to be a good tool to reliably increase the test coverage.

By utilizing the PCOLA-SOQ properties the effectiveness of a proposed test can be optimize and guaranteed. Test of which the effectiveness is dominated by the rest of the tests are automatically removed. In addition the PCOLA-SOQ score is also a well defined metric that allows the effectiveness to be compared to and even combined with the effectiveness of other test equipment.

Future work entails extending the current functionality. Currently discrete active components are not covered by the tool. Furthermore, the tool currently only can test for opens (presence implies no open contacts), but not shorts. another area for improvement would be to distinguish which of two tests with equal coverage is likely the most stable in production environments.

References

[1] K. Hird, K. P. Parker, and B. Follis. Test coverage: what does it mean when a board test passes? In *Proceedings IEEE International Test Conference (ITC)*, pages 1066–1074, 2002. doi: 10.1109/TEST.2002.1041863.

[2] A. W. Ley. Defect coverage of non-intrusive board tests (nbt): What does it mean when a non-intrusive board test passes? In *2009 International Test Conference*, pages 1–1, Nov 2009. doi: 10.1109/TEST.2009.5355828.

[3] Ieee standard for test access port and boundary-scan architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pages 1–444, May 2013. doi: 10.1109/IEEESTD.2013.6515989.

[4] R. Matheson. Second-generation pcb self-learn. *Computer-Aided Engineering Journal*, 4(5):209–212, October 1987. doi: 10.1049/cae:19870051.

[5] IEEE Standard American National Standard Canadian Standard Graphic Symbols for Electrical and Electronics Diagrams (Including Reference Designation Letters). *IEEE Std 315-1975 (Reaffirmed 1993)*, pages 241–244, 1993. doi: 10.1109/IEEESTD.1993.93397.

[6] EDIF Steering Committee et al. Edif electronic design interchange format version 2 0 0. *Electronic Industries Association*, 1987.

[7] F. Harary. *Graph Theory*. Addison-Wesley series in mathematics. Addison-Wesley Publishing Company, 1969.

[8] John Hopcroft and Robert Tarjan. Efficient Algorithms for Graph Manipulation. *Communications of the ACM*, 16(6):372–378, June 1973. doi: 10.1109/TEST.1990.114074.

[9] Wenxue Du. On graph isomorphism problem. *arXiv preprint arXiv:1710.09526*, 2017.

[10] Chung-Wen Ho, A. Ruehli, and P. Brennan. The Modified Nodal Approach to Network Analysis. *IEEE Transactions on Circuits and Systems*, 22(6):504–509, Jun 1975. doi: 10.1109/TCS.1975.1084079.

[11] Victor E Kane. Process capability indices. *Journal of quality technology*, 18(1):41–52, 1986.