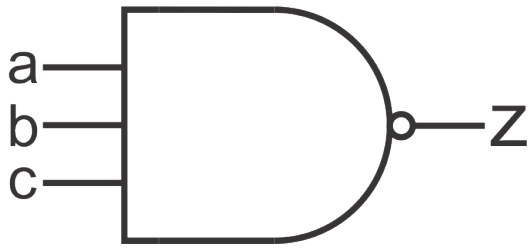
Fault Collapsing and the Minimal Test Set

The circuit



is given.

Determine the minimal test set for the singular stuck-at faults in this circuit.

"Singular stuck-at" Fault Model

"Stuck-at" refers to a short-circuited line having the value logical one or logical zero, without considering intermediary or unstable values.

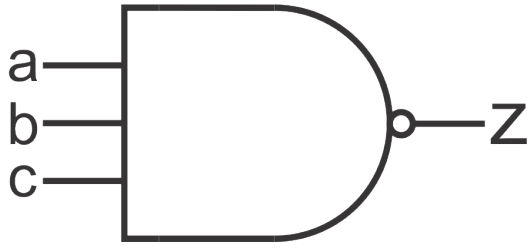
"Singular" refers to the fact that the circuit is either correctly functioning, without any faults, either it contains a single faulty line. For example, the line a could be stuck at 0, while the other lines and the rest of the circuit are correctly functioning. If it is supposed that, for example, the line b is stuck at 1, then it is considered that the other lines, including a, are correctly functioning.

It can be seen that the diagnosis is not performed for identifying the fault, first being assumed that the fault exists and then confirming or invalidating its existence.

Certainly, the "singular stuck-at" fault is not realistic because, on the one side, "unstable" faults can occur, which means a short circuit at 0, 1 or intermediary values, and on the other side, in an integrated circuit, usually the failure does not occur by oneself, but in group with others. Anyway, this failure model is a classical one, is well implemented by the testing software and the testing equipment and it represents a good base for the first phase of testing a circuit and also for modelling other fault types.

Tests

The circuit (3-input NAND gate) in the next figure is given.



The truth table for this circuit is given in the following table.

abc	d
000	1
001	1
010	1
011	1
100	1
101	1
110	1
111	0

The output of the circuit can be easily computed: being a NAND gate, the output can be computed as that of an AND gate which is then inverted. For an AND gate, for the majority of the input combinations, the output will have the value 0. The "special" situation, or the "exception", takes place when all the inputs have the value 1 and when the output is 1. Therefore, for an NAND gate, like the one in our case, the exception takes place when all the inputs are 1 and the output is 0. For the rest of the input combinations the output will be 1. Certainly, this rapid method is not compulsory, the table can be computed rigorously, this meaning that all the input combinations are analysed and the output is computed for each one of them. This is the safer way.

Usually, during the circuit design, the methods for testing the manufactured circuit are developed too based on the project. The reason why this is done is that when the first circuit is produced, it has to be tested, this implying that the testing method is already documented and tuned up.

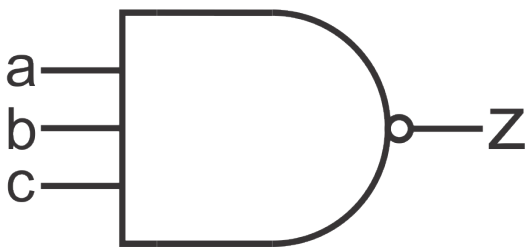
Here, we try to adjust the method with which to test different singular stuck-at fault types that could occur in our circuit previously depicted and, more precisely, in circuits based on the scheme presented here.

The problem is that although the circuit structure is clear on the display, we do not have access inside the physical set-up which will be manufactured because it consists of an integrated circuit. For example, if we assume that the line a is blocked on 0 and we wish to verify if this assumption is true or not (we hope it is not), we cannot measure voltages inside the circuit. We can only measure the voltage at the IC pins, which are a, b, c and d, the primary inputs and outputs, but not in the immediate vicinity of the gate, where the failure occurs.

The voltages (the logical values) on the primary inputs do not have to be measured because they are controllability points, being applied to the inputs and being previously known. The single observability point, where we can measure the voltage for understanding what happens in the real manufactured circuit, is the output, d.

That is why, if we suppose a to be b-l-0 (stuck-at-0), the only way to verify if this assumption is true is to apply a well chosen binary combination on the inputs a, b and c, measure the logical value on the d output and from this value to validate the presence or the absence of the a_0 (a b-l-0) fault. The trick lies in the correct choice of the input combination, which will be named *test*, from now on.

Again, the circuit:



We fill in the truth table:

abc	d	a_0
000	1	1
001	1	1
010	1	1
011	1	1
100	1	1
101	1	1
110	1	1
111	0	1

The new a_0 column contains values of the d output also, but these are obtained in the case in which a is stuck at 0. If a is stuck at 0, a will be 0 no matter what combination we have on the input lines. As it can be seen on the last row of the table, the problem occurs when using the test 111. When we put 111 on the inputs, because a is stuck at 0, the gate will have 011 on its inputs, producing 1 on the output (line d).

Actually, in our circuit's case, it is very simple to compute the output when a is b-l-0: we have 1 on the entire column, because for every AND or NAND gate, if at least one of their input lines is 0, then the output will be 0 or 1 respectively, no matter the values the other inputs have.

In the same way, for every OR or NOR gate, if at least one input is 1, then the output will be 1 or 0, respectively.

The following table will be filled in for the other possible singular stuck-at type faults:

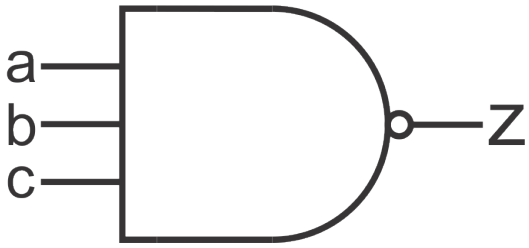
abc	d	a_0	a_1	b_0	b_1	c_0	c_1	d_0	d_1
000	1	1	1	1	1	1	1	0	1
001	1	1	1	1	1	1	1	0	1
010	1	1	1	1	1	1	1	0	1
011	1	1	0	1	1	1	1	0	1
100	1	1	1	1	1	1	1	0	1
101	1	1	1	1	0	1	1	0	1
110	1	1	1	1	1	1	0	0	1
111	0	1	0	1	0	1	0	0	1

Fast computation:

- a_0 – is already computed. It can be computed in the same way as "blue" below;
- in our circuit, if an input is stuck at 0, then the output will be 1 (blue);
- a_1 – in case of the test where $a = 1$, the output with the fault a_1 has the same value as the output without the fault, d (red);
- b_1 – in the rows where the value of b is 1, the output d can be copied (green);
- c_1 – where there is $c = 1$ in the test, we can copy the output without fault (orange);
- d_0 and d_1 – if the output d is stuck at 0, the circuit output will be 0. The same for d_1;
- the rest of the output values, for different faults and tests, must be computed (violet).

It should not be forgotten that the values in the above table, excepting the trio abc, are values of the d output for different combinations of the abc inputs and singular stuck-at faults.

Again, the circuit:



The last table, again:

abc	d	a_0	a_1	b_0	b_1	c_0	c_1	d_0	d_1
000	1	1	1	1	1	1	1	(0)	1
001	1	1	1	1	1	1	1	(0)	1
010	1	1	1	1	1	1	1	(0)	1
011	1	1	(0)	1	1	1	1	(0)	1
100	1	1	1	1	1	1	1	(0)	1
101	1	1	1	1	(0)	1	1	(0)	1
110	1	1	1	1	1	1	(0)	(0)	1
111	0	(1)	0	(1)	0	(1)	0	0	(1)

It can be observed that some of the values are circled with a (red) line. These values are important because they point the test that detects certain faults.

For example, the test 000 does not detect the fault a_0 because if it is supposed that a b-l-0 and the combination 000 is applied to the abc inputs, the d output will have the value 1. Unfortunately, this happens when the circuit is functioning correctly, too, without the presence of the a b-l-0 fault. It cannot be known whether the output is 1 because of the a_0 fault, or because we injected **a=0**, $b = 0$, $c = 0$ (000) and the circuit is functioning correctly. Because for the test 000 the output has the same value for the correctly functioning circuit and for the fault a_0, it does not detect the a_0 fault.

On the other hand, the test 111 detects a_0 because in this case, the correctly functioning circuit outputs 0 while the faulty circuit outputs 1. A difference between the two cases can be observed.

It should not be forgotten that it is not necessary to detect which fault occurs in the circuit, but only to assume that the fault occurs, and then to verify if it is there or not. Therefore, the value of the output is circled where it differs from the circuit that runs correctly (second column).

abc	a_0	a_1	b_0	b_1	c_0	c_1	d_0	d_1
000							x	
001							x	
010							x	
011		x					x	
100							x	
101				x			x	
110						x	x	
111	x		x		x			x

The table can be redrawn only with the important positions filled in, but this is not compulsory (done in the neighbouring table).

The column for d was omitted because it is no longer necessary.

This way we obtained a test set $T = \{000, 001, 010, 011, 100, 101, 110, 111\}$. For example, if it is supposed that d is stuck at 0, this fault can be detected with 000 (or 001, 010, 011, 100, 101, 110, too). d_1 is detectable with 111.

Minimal Test Set

The problem of the above test set is that it is exhaustive, that meaning that it contains all the possible combinations for the inputs. This is not a big concern where there are only three inputs because there are only $2^3 = 8$ tests. It is not the same for a circuit which has 1000 inputs, which requires $2^{1000} = 1.07 \cdot 10^{301}$ tests. So, it is not feasible to try all the possible combinations, because the testing of an integrated circuit would take too much time.

The reduction of the number of tests, even to a minimum set, is possible. This way, we obtain "The Minimal Test Set". This is possible because of the fault model, which is "singular". For example, in the preceding table, the test 011 detects a_1, but also detects d_0. It should not be forgotten that it is assumed that two faults do not appear in the same time, and, even more, there is no difference between them – we know before applying the test for which fault we apply it.

For determining the minimal test set, the following method is used.

Essential Tests

The same table:

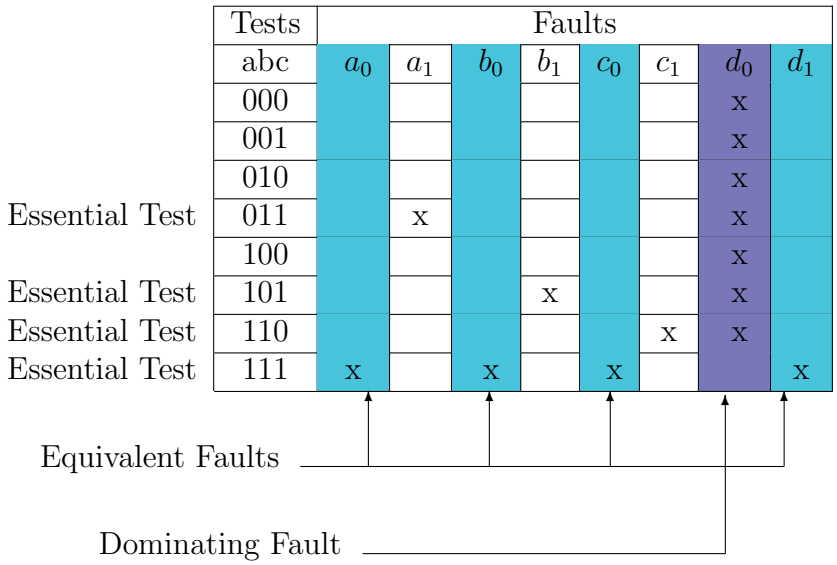
	abc	a_0	a_1	b_0	b_1	c_0	c_1	d_0	d_1
	000							x	
	001							x	
	010							x	
es.	011		x					x	
	100							x	
es.	101				x			x	
es.	110						x	x	
es.	111	x		x		x			x

The aim is the shortening, even the minimization of the number of tests, but only till the limit at which for every possible singular stuck-at fault in the circuit there still remains at least one test that is capable of detecting it.

For the a_0 fault, the test 111 is very important, because it is the only one that detects it. If this test is not included in the minimal set, then it will be incomplete, it will not contain any test that detects a b-l-0. In the same manner, 011 is essential because of a_1, 111 because of b_0, 101 because of b_1, 111 because of c_0, 110 because of c_1, 111 because of d_1. It can be observed that d_0 does not posses essential tests because it has more than one test capable of detecting it.

Fault Collapsing

Again, the previous table:



Equivalences

It can be observed that the **test sets** for the a_0, b_0, c_0 and d_1 faults **coincide**. Certainly, in the case of the current circuit, these sets consist of a single test, 111. Therefore, the faults a_0, b_0, c_0 and d_1 are considered to be equivalent and they form an **equivalence class**. They were marked with the light blue colour.

On the other hand, for example the fault c_1 is not equivalent to d_0, because d_0 possesses tests which do not detect c_1, and so their sets do not coincide. This circuit has a single equivalence class for the singular stuck-at faults.

For the minimal test set, a representative fault should be chosen from each equivalence class and a test for each fault should be found. The other faults in that equivalence class can be ignored in the test search because the tests that are found for the representative fault will detect its equivalent faults too.

Dominances

It can be observed that the test set for the fault d_0 includes the test set for a_1, the test set for b_1 and the test set for c_1. This is why, the fault d_0 is considered to be **dominant** for a_1, b_1 and c_1, or "the fault d_1 dominates the faults a_1, b_1 and c_1" or "the faults a_1, b_1 and c_1 are dominated by the fault d_0". It is marked in the table with the light violet colour .

On the other hand, for example, the failure d_1 does not dominate c_0 because the test set for c_0 is not a subset of the test set for d_1. Anyway, these two faults are already equivalent.

For the minimal test set, it is sufficient to find tests for the dominated faults in each dominance class and the dominant fault can be ignored because the tests which detect the dominated faults will detect it, too.