Defects, Errors and Faults

Models bridge the gap between *physical reality* and a *mathematical abstraction* and allow for development of analytical tools.

Definitions:

- **Defect**: An untended difference between the implemented hardware and its intended function.

- **Error**: A wrong *output signal* produced by a defective system.

- **Fault** is a *logic level abstraction* of a **physical defect**.

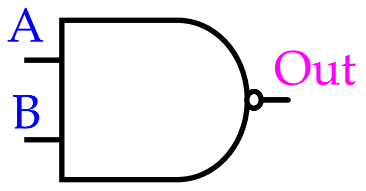
  Used to describe the change in the logic function caused by the defect.

  Fault abstractions reduce the number of conditions that must be considered in deriving tests.

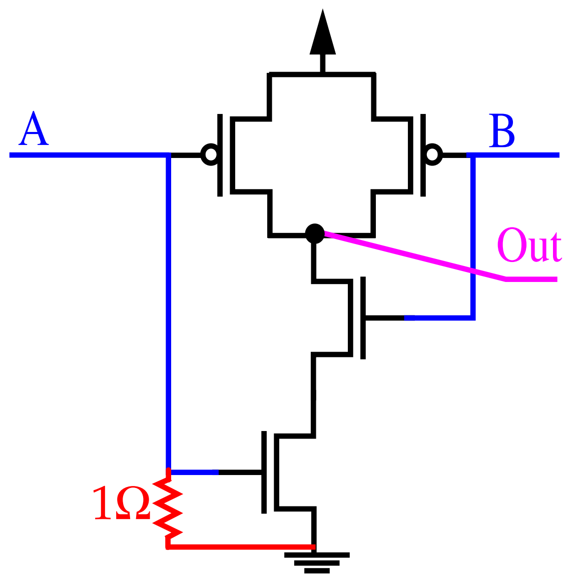
A *collection of faults*, all of which are based on the same set of assumptions concerning the nature of defects, is called a **fault model**.

Defects, Errors and Faults

For example:



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



- **Defect**: *A* shorted to GND.
- **Fault**: *A* Stuck-at logic 0.
- **Error**: *A=1*, *B=1* => *C=1*

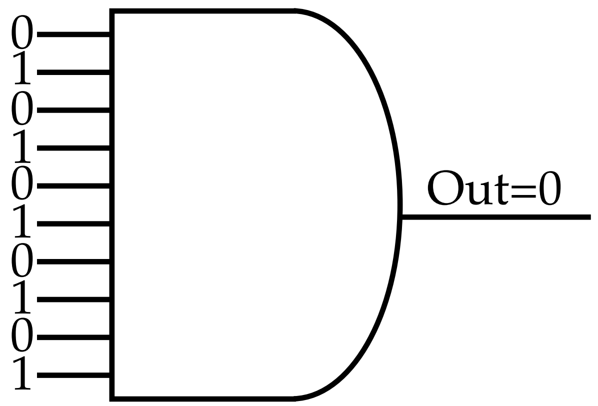
  Correct output is *C=0*.

Note that the **error** is *not* permanent

For example, no error occurs if at least one input is 0.

Functional Vs. Structural Testing

For a 10-input AND gate, we can apply the test *0101010101*.



If *Out* = *0*, what can we conclude?

- (A) it's an AND gate.
- (B) it's not a NAND gate.
- (C) it's a NOR gate.
- (D) it's not an OR gate.

There are $2^{2^{10}}$ Boolean functions possible.

To be absolutely sure, we need to apply all 2^{10} patterns.

Clearly, functional test is not feasible.

Functional Vs. Structural Testing

Design verification may need to use exhaustive functional tests.

Fortunately, for hardware testing, we **can assume** the function is correct.

The focus on **structure** makes it possible to develop algorithms that are independent of the design.

The algorithms are based on **fault models**.

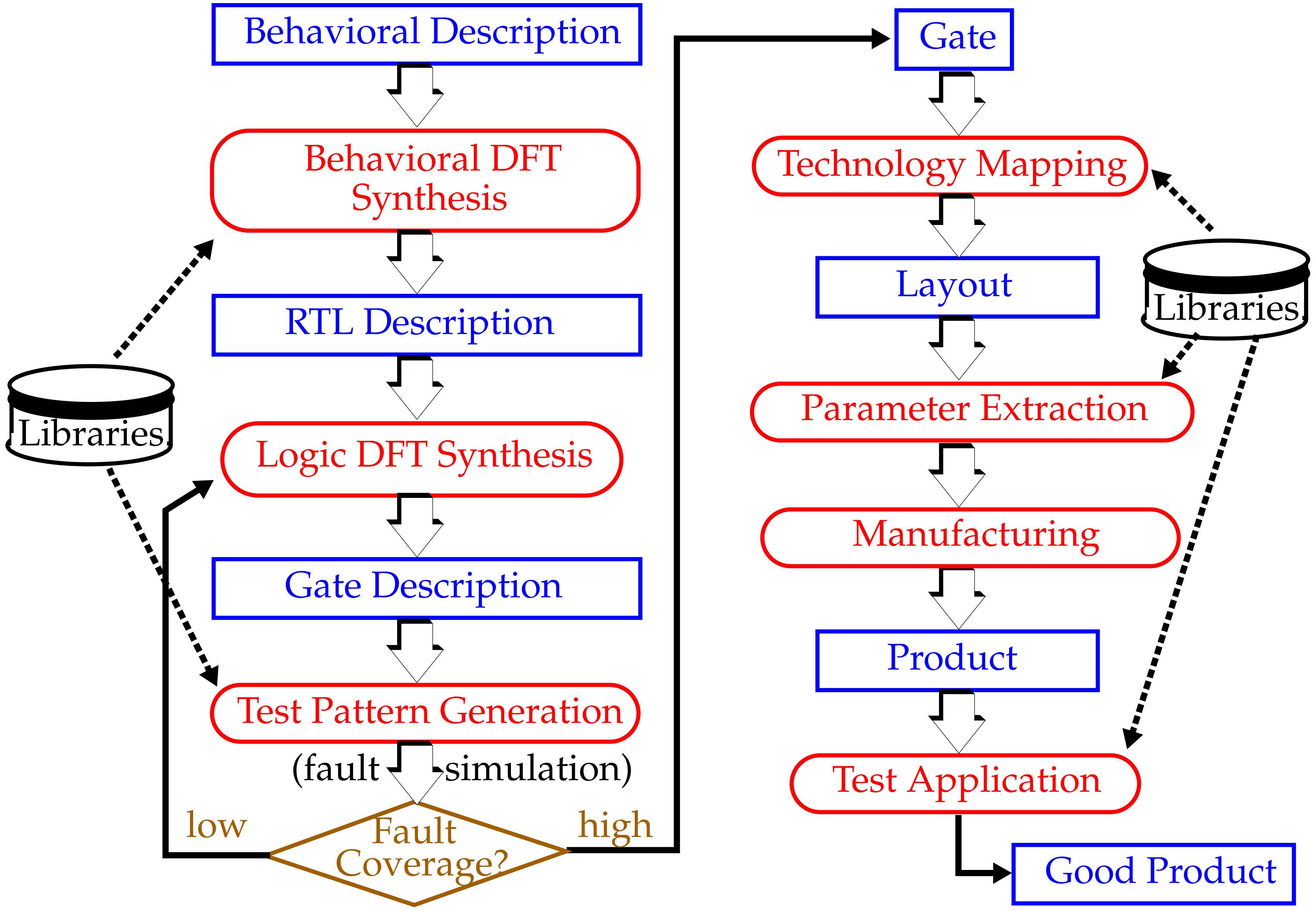
Fault models can be formulated at the various levels of design abstraction.

- *Behavioral level*: Faults may not have any obvious correlation to defects.
- *RTL and Logic level*: Stuck-at faults most popular, followed by bridging and delay fault models.
- *Transistor (switch) level*: Technology-dependent faults.

I_{DDX}: **Defect-based methods**

Adv: can represent defects not represented by other fault models.

Design Levels from Testability Perspective:



Fault Models

The text describes these and other fault models:

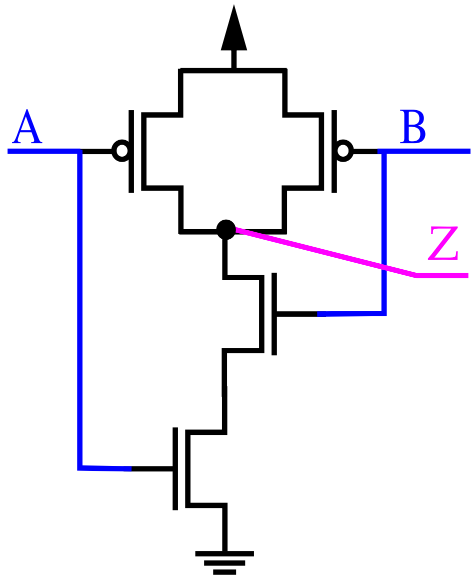
- Bridging fault
- Defect-oriented fault (e.g. bridging, stuck-open, I_{DDQ}).
- Delay fault (transition, gate-delay, line-delay, segment-delay, path-delay).
- Intermittent fault
- Logical fault (often stuck-at)
- Memory fault (single cell SA0/1, pattern sensitive, cell coupling faults).
- Non-classical fault (stuck-open or stuck-on, transistor faults for CMOS).
- Pattern sensitive fault
- Pin fault (SA faults on the signal pins of all modules in the circuit).
- Redundant fault
- Stuck-at fault

Single stuck-at faults (SSF)

Assumes defects cause the signal net or line to remain at a fixed voltage level.

Model includes **stuck-at-0** (SA0) or **stuck-at-1** (SA1) faults and assumes only one fault exists.

For example, how many *SSF* faults can occur on an *n-input* NAND gate?



Inputs	Fault-Free	Faulty Response					
AB	Response	A/0	B/0	Z/0	A/1	B/1	Z/1
00	1	1	1	**0**	1	1	1
01	1	1	1	**0**	**0**	1	1
10	1	1	1	**0**	1	**0**	1
11	0	**1**	**1**	0	0	0	**1**

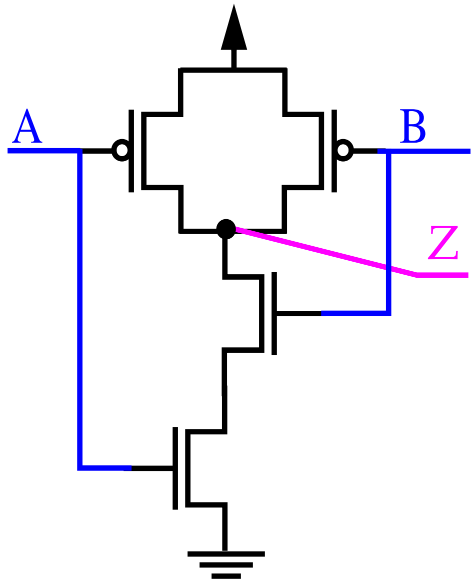
What fault(s) does the pattern *AB* = *01* detect?

What is the *minimum* number of tests needed to "detect" all of them?

What are the tests?

SSF

A **dominant** input value is defined as the value that *determines* the state of the output independent of the other values of the inputs.



Inputs	Fault-Free	Faulty Response					
AB	Response	A/0	B/0	Z/0	A/1	B/1	Z/1
00	1	1	1	**0**	1	1	1
01	1	1	1	**0**	**0**	1	1
10	1	1	1	**0**	1	**0**	1
11	0	**1**	**1**	0	0	0	**1**

What is the dominant input value for the NAND?

How many tests do you need to diagnosis the fault?

Can you distinguish between all of the faults?

SSF

An *n*-line circuit can have at most *2n* SSF faults.

This number can be further reduced through **fault collapsing**.

Fault detection requires:

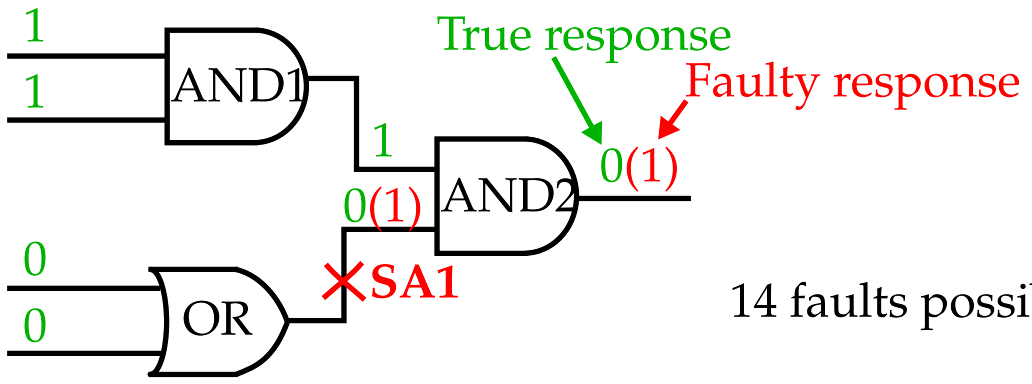
- A test *t* activates or provokes the fault *f*.

- *t* propagates the error to *observation point* (primary output (PO)/scan latch).

  A line that changes with *f* is said to be **sensitized** to the fault site.

Fault propagation requires *off-path* inputs be set to *non-dominant* values.

01, *10*, and *11* do **not** provoke the fault

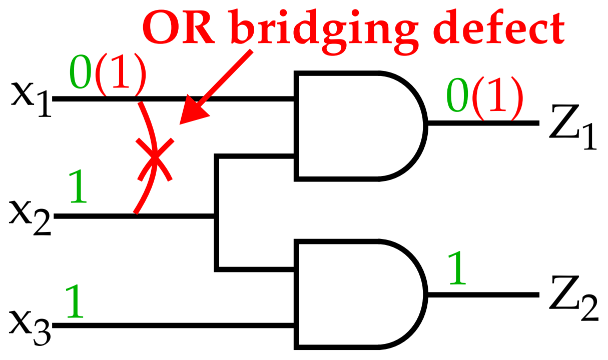


14 faults possible here.

SSF

Let *Z(t)* represent the response of a circuit *N* under input vector *t*.

Fault *f* transforms the circuit to N_f and its response to $Z_f(t)$.



$Z_1 = x_1x_2 \longrightarrow Z_{1f} = x_1 + x_2$

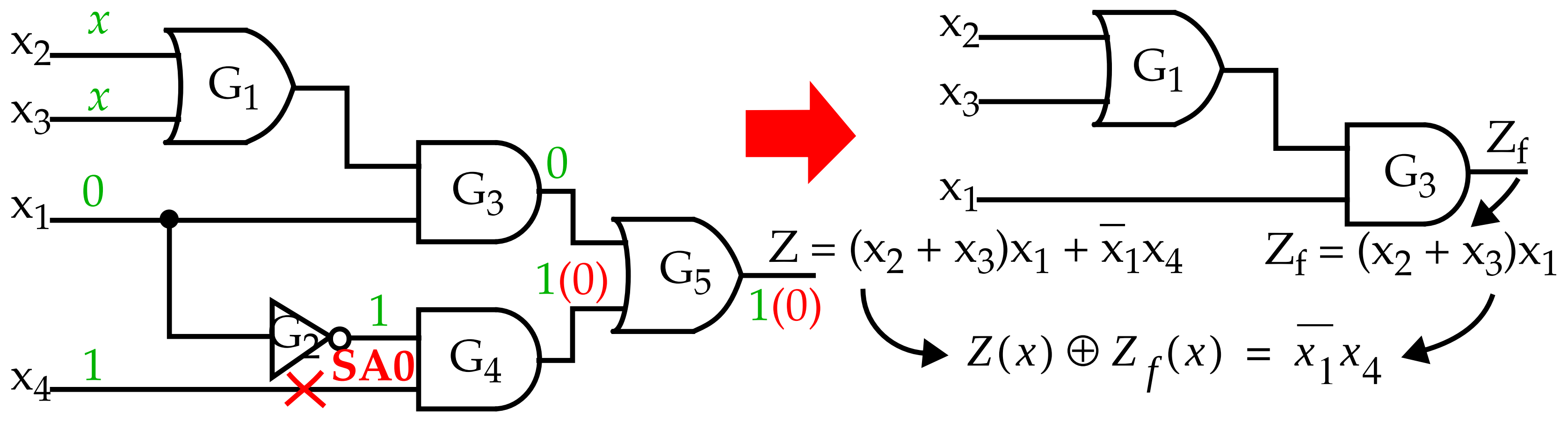
$Z_2 = x_2x_3 \longrightarrow Z_{2f} = (x_1 + x_2)x_3$

011 defects *f* because Z(*011*) = *01* and Z_f(*011*) = *11*.

The set of all tests *x* that detect *f* is given by

$$Z(x) \oplus Z_f(x) = 1$$

In this example, any test in which $x_1 = 0$ and $x_4 = 1$ is a test for *f*.



$Z = (x_2 + x_3)x_1 + \overline{x}_1x_4$ $Z_f = (x_2 + x_3)x_1$

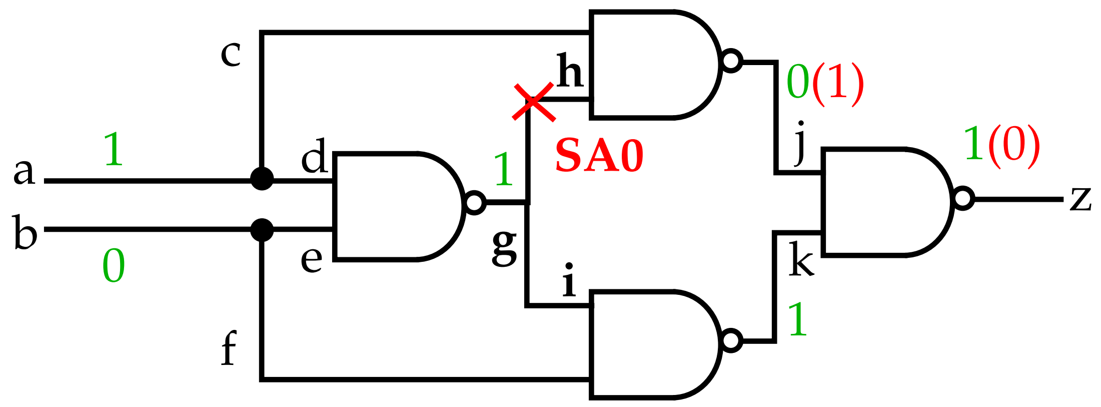
$$Z(x) \oplus Z_f(x) = \overline{x}_1x_4$$

The expression $\overline{x}_1x_4$ represents 4 tests (*0001*, *0011*, *0101*, *0111*).

SSF

Note that faults on *fanout stems* and *fanout branches* are **not** the same.

Signal line *g*, *h* and *i* carry the same signal value.



Input *ab* = (*10*) activates *h SA0* and is detectable at *z*.

This input also activates *SA0* faults on *g* and *i*

However, *i SA0* is **not** detectable since it is blocked by *f* = 0.

Stem *g* is the *output* of the NAND and lines *h* and *i* are *inputs* to downstream NANDs.

To consider all possible faults, we model faults on all nodes *g*, *h* and *i*.

Fault Equivalence

Two faults *f* and *g* are considered **functionally equivalent** iff $Z_f(x) = Z_g(x)$.

There is no test that can distinguish between *f* and *g*. i.e., *all tests* that detect *f* also detect *g*.

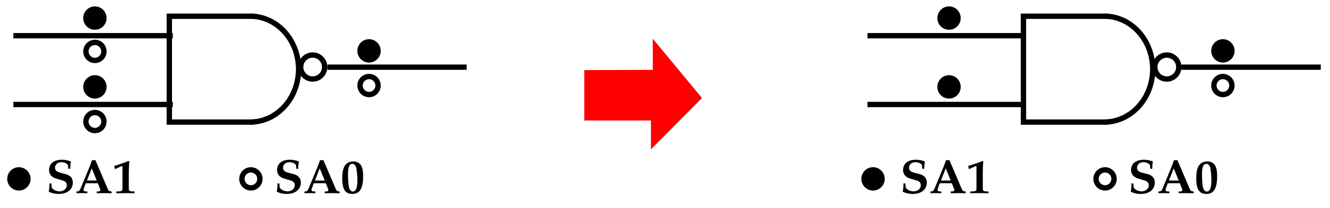
For large circuits, determining this is computationally intensive.

However, equivalence can be determined for simple gates and applied to large circuits.

Any *n*-input gate has 2(n+1) SA faults.

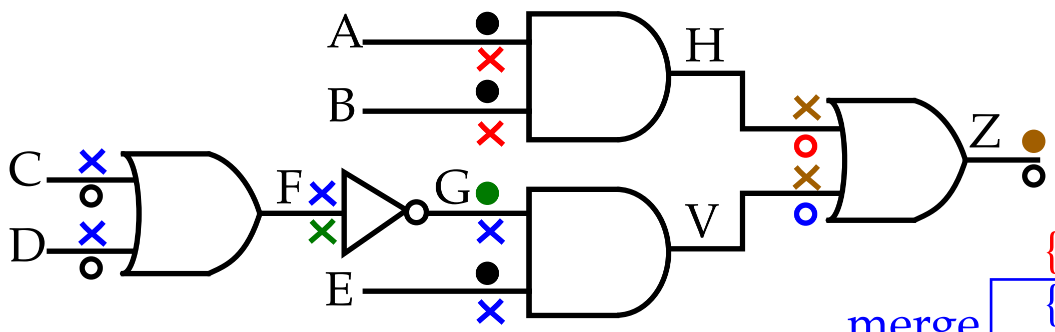
For the NAND gate, the *SA0* on the inputs are equivalent to *SA1* on the output and all three are detected by the same test pattern *AB*=(*11*).

For any *n*-input primitive gate with *n*>1, only ***n*+2** single SA faults need to be considered.



Fault Equivalence

Equivalence fault collapsing is performed from inputs to output.



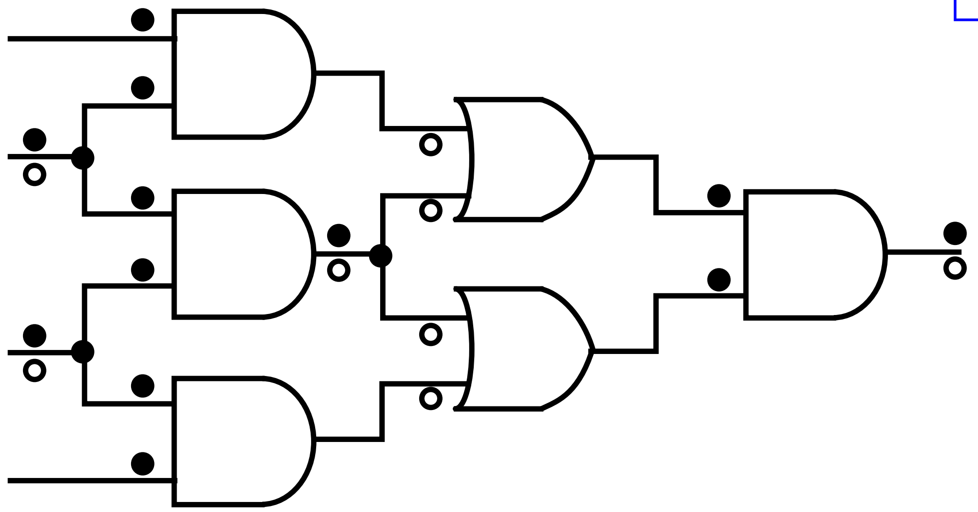
10/20 = 0.5 remain

{A/0, B/0, H/0}

merge $\boxed{\begin{array}{l}\{C/1, D/1, F/1, G/0\}\\ \{E/0, G/0, V/0\}\end{array}}$

{H/1, V/1, Z/1}

{F/0, G/1}



20/32 = 0.625 remain

Reduction is between 50-60% and is larger, in general, for fanout free circuits.

Fault Equivalence

Functional equivalence partitions the set of all faults into *functional equivalence classes*, from each of which only one fault needs to be considered.

This property is useful for test generation programs.

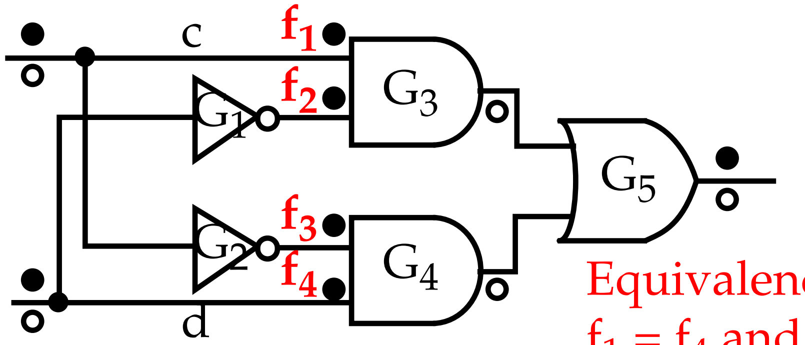
Fault equivalence reduces the size of the fault list.

Fault equivalence is important for fault location analysis as well.

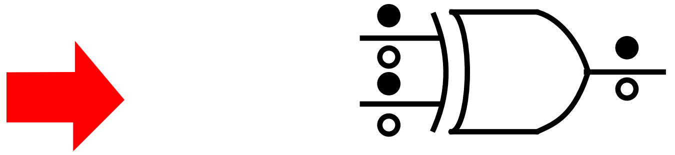
A *complete location test* set can diagnose a fault to within a functional equivalence class.

This represents the *maximal diagnostic* resolution achievable.

Fault collapsing using this simple method cannot find all equivalencies.



Equivalence of SA1 faults:
$f_1 = f_4$ and $f_2 = f_3$

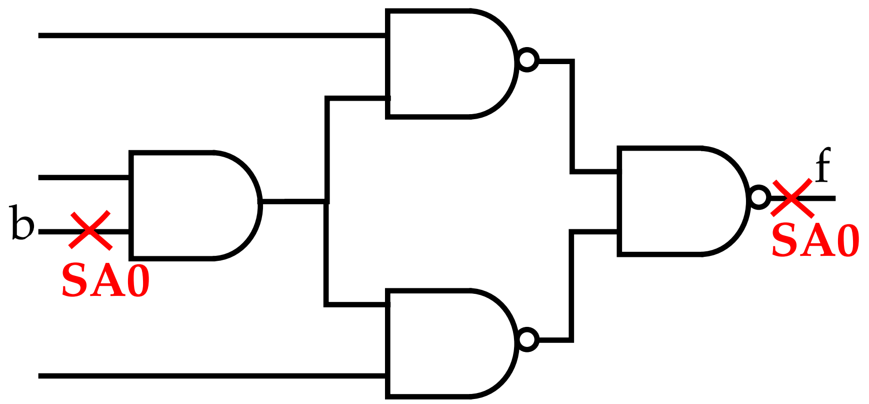


Method indicates need to model all 6 faults

Fault Equivalence

Therefore, the equivalence classes derived are not maximal.

Our method will not identify faults *b* SA0 and *f* SA0 as equivalent



Algorithms are available to solve the more general case of **functional equivalence** based on:

$$Z_f(x) \oplus Z_g(x) = 0$$

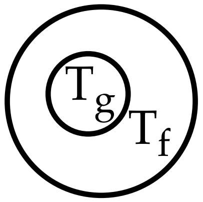
The benefit of identifying these additional fault equivalences is usually not worth the effort, however.

See text for ISCAS'85 benchmark circuit results.

Fault Dominance

If fault detection is the objective (not diagnosis), then **fault dominance** can be used to further reduce the fault list.

A fault *f* dominates another fault *g* if the set of all tests that detect *g*, T_g, is a subset of the test set of T_f.



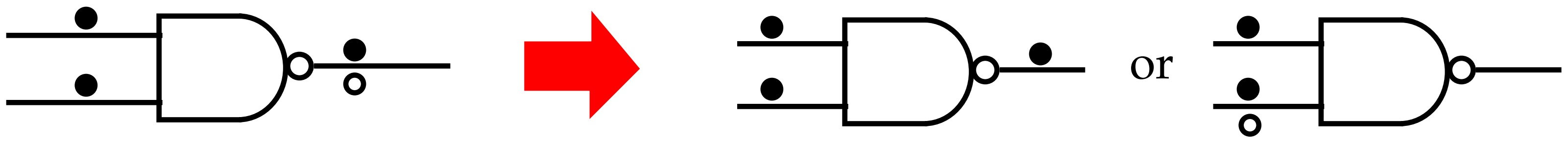
$T_g \subseteq T_f$

Def: *f* dominates *g* iff
f and *g* are functionally equivalent under T_g.

Therefore, any test that detects *g* will also detect *f*.
Since *g* implies *f*, it is sufficient to include *g* in the fault list.

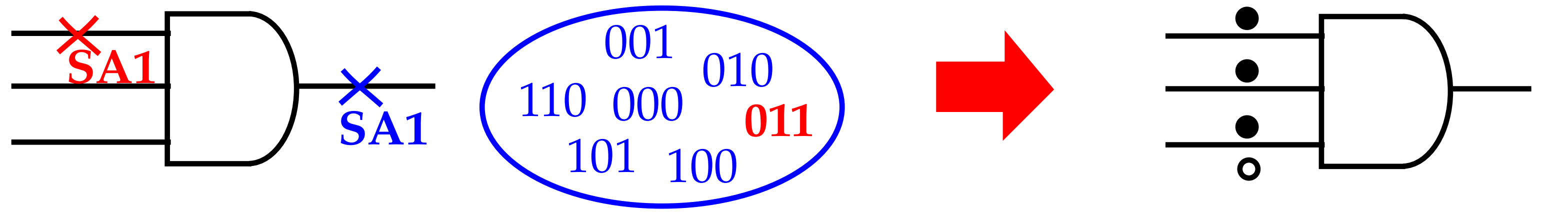
For example, the *SA1* test (*g*) for input *A* of the NAND gate also detects *SA0* (*f*) on the output (the same is true for input *B*.)

Therefore, *Z-SA0* can be dropped from the list.

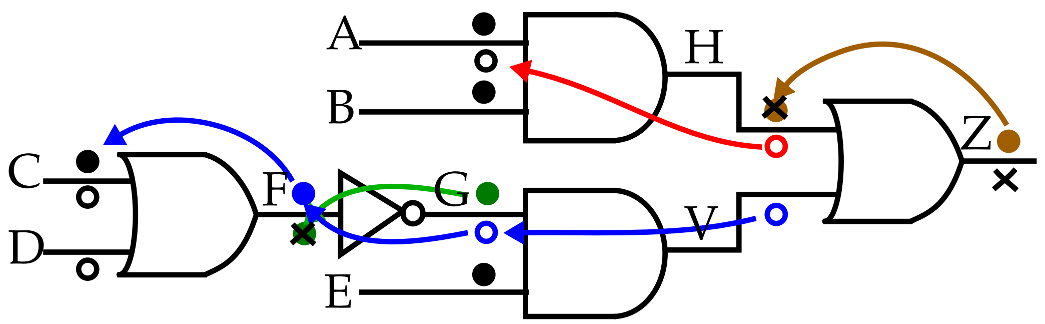


Fault Dominance

For larger input gates.



Dominance fault collapsing is performed from outputs to inputs.



7/20 = 0.3 remain

Here, the *x* indicates the collapsing of the fault via dominance.

We can also, optionally, choose to move the output fault preserved in the equivalence fault collapsing to an arbitrary input.

One such fault list may be: {*A/0*, *A/1*, *B/1*, *C/0*, *C/1*, *D/0*, *E/1*}

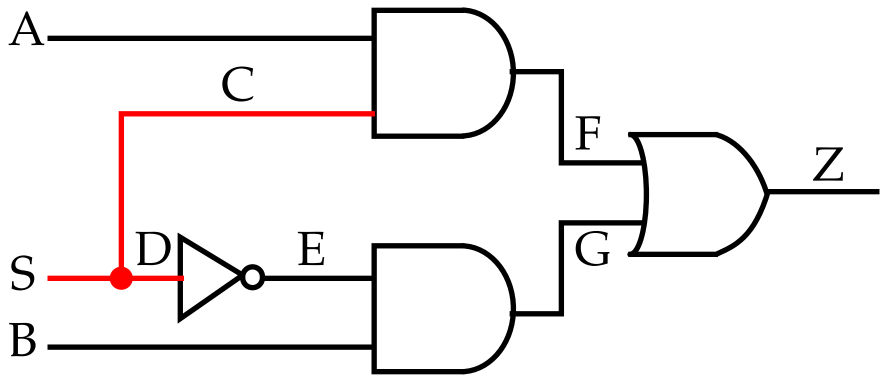
Or another may be: {*B/0*, *A/1*, *B/1*, *C/0*, *D/1*, *D/0*, *E/1*}

Fault Dominance

Note that these lists contains **only** faults on the PIs.

Any test set that detects all SSFs on the PIs of a **fanout free** combinational circuit C detects all SSFs in C.

What happens in the presence of fanout?



Eliminating S, reduces the circuit to a fanout free circuit.

SAB = (*010*) detects C SA1, *SAB* = (*001*) detects D SA1 => both detect S SA1.

SAB = (*110*) detects C SA0, *SAB* = (*101*) detects D SA0 => both detect S SA0.

Therefore, SA faults on *stem* **dominate** SA faults on the *fanout branches*.

More generally, any test set that detects all SSF on the PIs and fanout branches of C detects all SSF in C.

The PIs and fanout branches are called **checkpoints**.

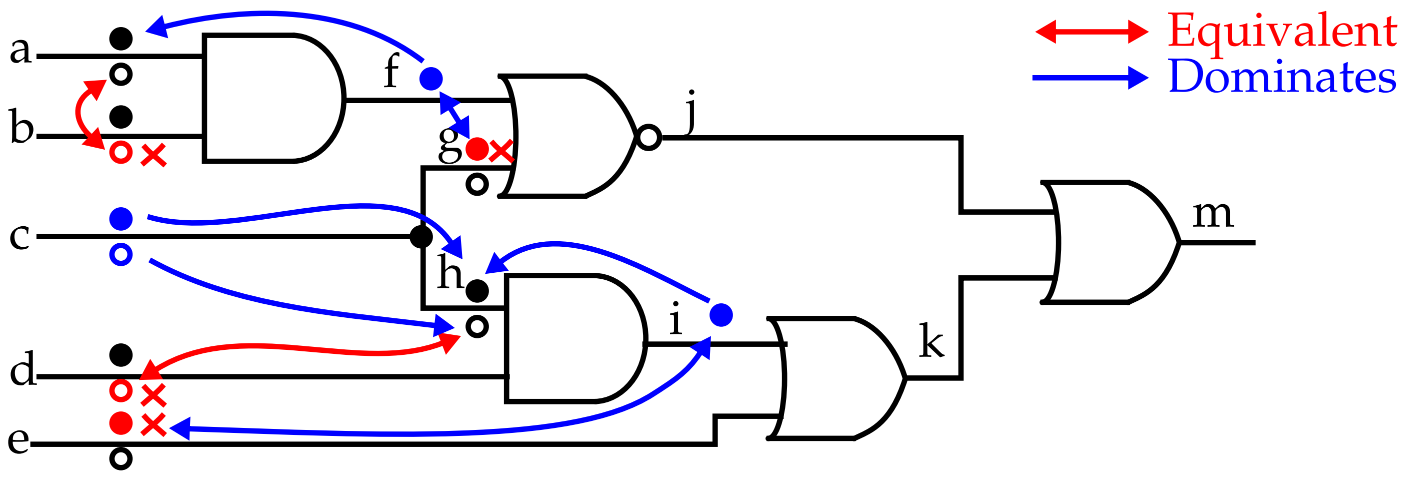
Fault Dominance

Therefore, it is sufficient to target faults only at the *checkpoints*.

Equivalence and dominance relations can then be used to further collapse the list of faults.

For example, this circuit has 24 SSFs.

But it only has 14 checkpoint faults (the 5 PIs) + *g* and *h*.



This leaves 8 faults from the original list of 14 checkpoint faults.

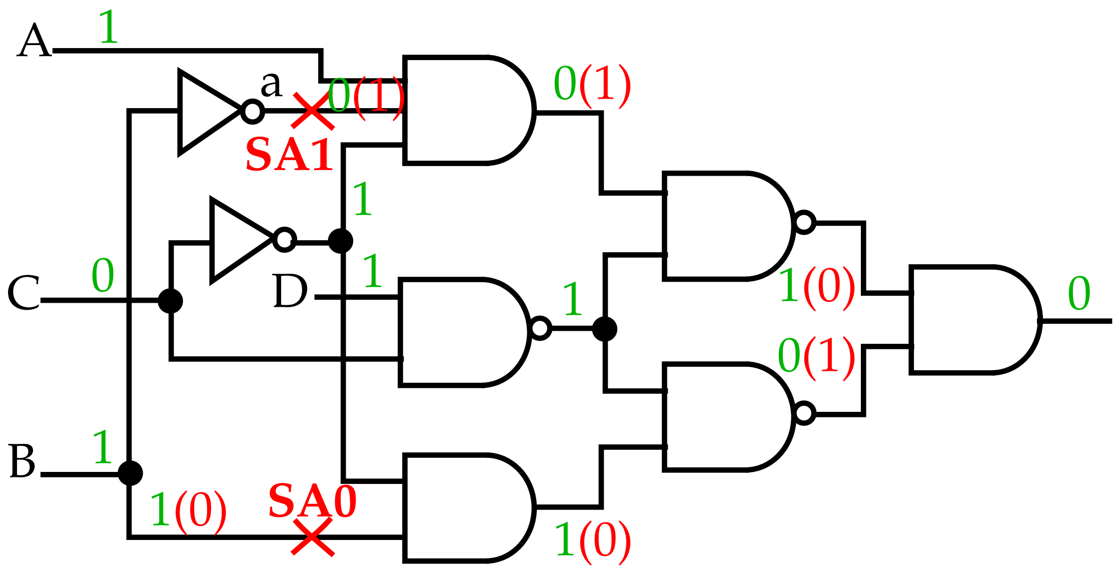
Undetectable Faults

A fault is **detectable** if there exists a test *t* that defects *f*. It is **undetectable**, if **no** test simultaneously activates *f* and creates a sensitized path to a PO.

Undetectable faults may appear to be harmless.

However, a *complete* test set may not be sufficient if one is present in a chip with *multiple faults*.

The fault *b* SA0 is **no longer detectable** by the test *t* = (*1101*) if the **undetectable fault** *a* SA1 is present.



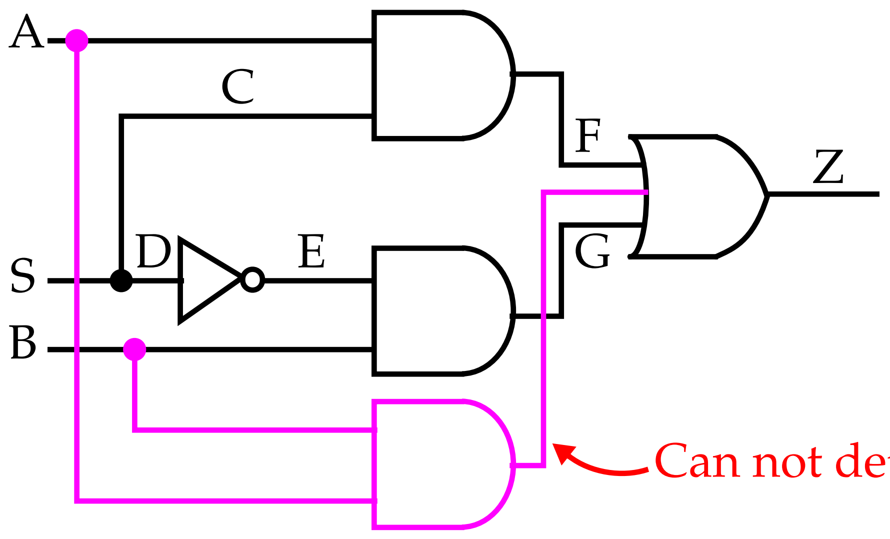
Undetectable Faults

If test *t* is the only test in the complete test set *T* that detects *b* SA0, then *T* is no longer complete in the presence of *a* SA1.

Redundancy: A combinational circuit that contains an *undetectable fault* is said to be redundant.

Redundant faults cause ATPG algorithms to exhibit worst-case behavior.

Redundancy is not always **undesirable**.



Can not detect SA0 -- why?

Describe the transition that causes a static hazard without the 'shaded' AND.

$F(A,B,S) = AS + B\vec{S}$

$F(A,B,S) = AS + B\vec{S} + AB$

redundant product term.