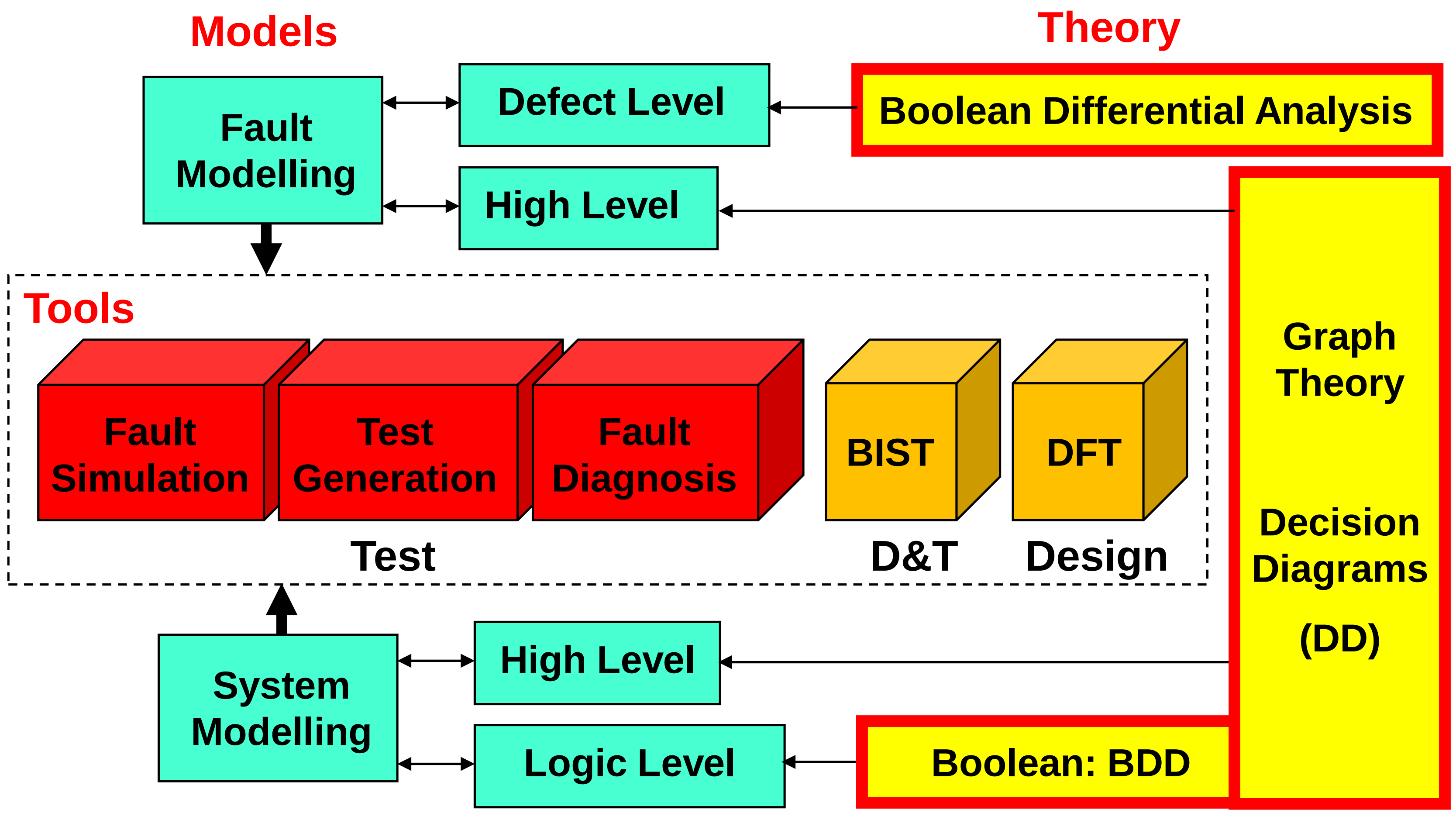
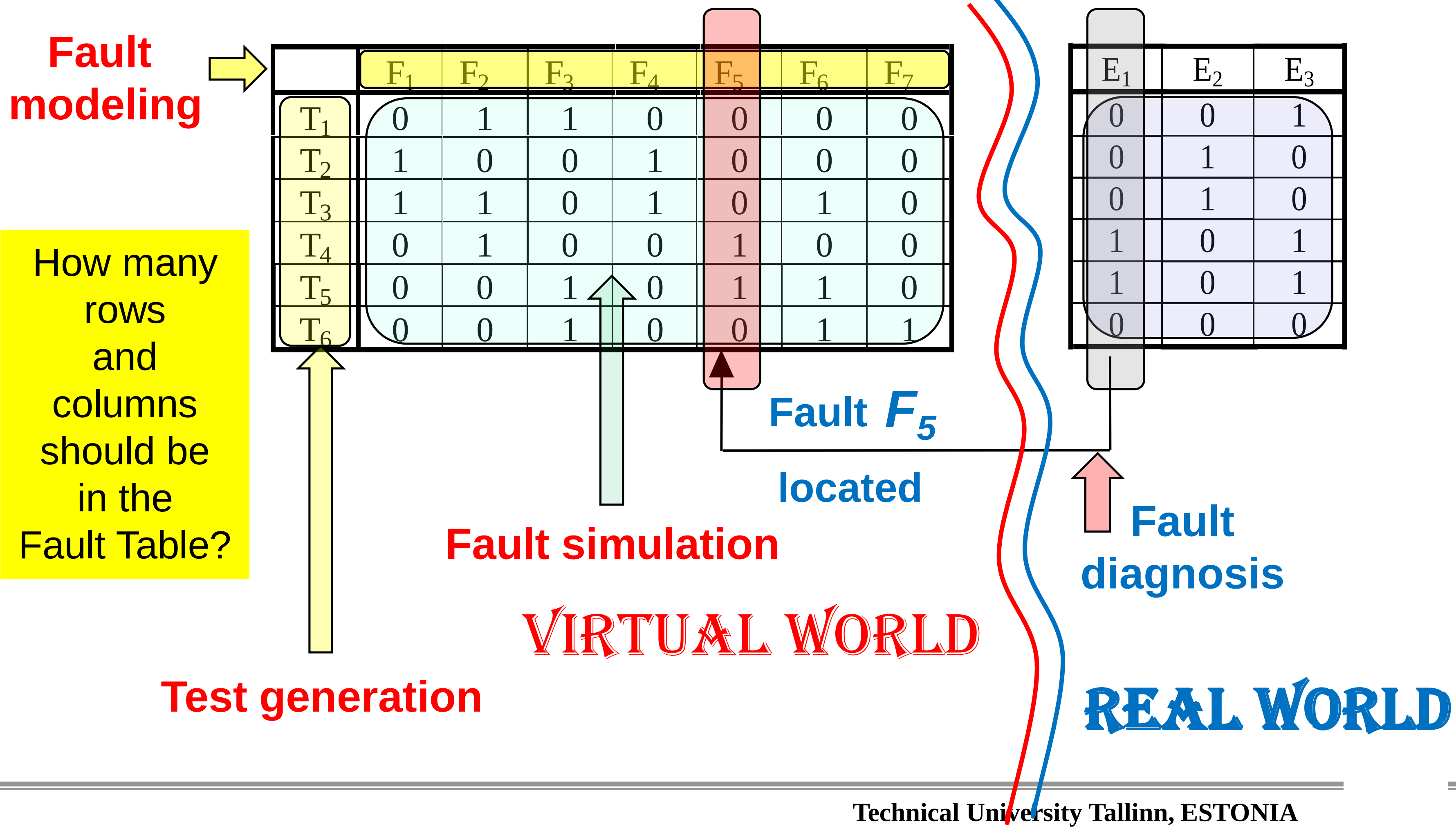
Süsteemide diagnostika

3. Rikete modelleerimine

Introduction to Theories: The Course Map



Test Related Basic Problems



Fault table (Solutions of Diagnostic equations)

	F_1	F_2	F_3	F_4	F_5	F_6	F_7
T_1	0	1	1	0	0	0	0
T_2	1	0	0	1	0	0	0
T_3	1	1	0	1	0	1	0
T_4	0	1	0	0	1	0	0
T_5	0	0	1	0	1	1	0
T_6	0	0	1	0	0	1	1

Test experiment data

E_1	E_2	E_3
0	0	1
0	1	0
0	1	0
1	0	1
1	0	1
0	0	0

How many rows and columns should be in the Fault Table?

Why We Need Fault Models?

- Fault models are needed for
  - test generation,
  - test quality evaluation and
  - fault diagnosis
- To handle real physical defects is too difficult
- The fault model should
  - reflect accurately the behaviour of defects, and
  - be computationably efficient
- Usually combination of different fault models is used
- Fault model free approaches (!)

Classification of Fault Models

- Fault modeling levels
  - Transistor level defects
    - stuck-open, stuck-off
  - Logic level faults
    - stuck-at fault model
    - bridging fault model
    - delay fault model
  - Register transfer level faults
  - ISA level faults (MP faults)
  - SW level faults
- Hierarchical fault modeling and mapping
- Functional fault modeling

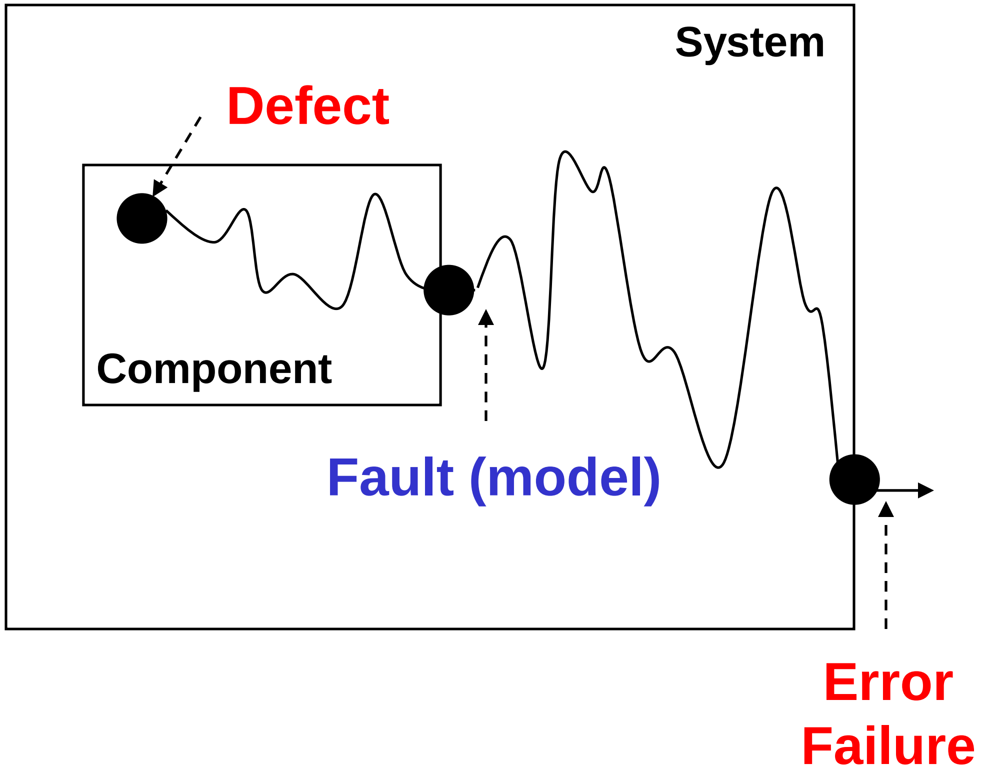
Low-Level models

High-Level models

Fault modeling terminology

- **Defect:** a **physical imperfection**, which can manifest itself as an erroneous logic signal
- Defect does not allow easy and direct mathematical treatment for diagnostic purposes
- **Fault:** a **logic fault model** as a manifestation of an error in a logic signal
- **Error:** an instance of an **incorrect operation** of the system being tested
- The causes of the observed errors may be **design errors** or **physical faults (defects)**
- **Failure:** an error which causes a **system failing** to perform in a required manner

Defects, faults and errors



Physical Defects as Fault Causes

Physical defects may occur:

- **Manufacturing process:** **missing contacts , parasitic transistors, gate oxide shorts, oxide break-down, metal-to silicon shorts, missing or wrong components, broken or shorted tracks (board design), etc.**
- **Process fabrication marginalities:** **line width variation, etc.**
- **Material and age defects:** **bulk defects (cracks, crystal imperfections), surface impurities, dielectric breakdown, electromigration, etc.**
- **Packaging:** **contact degradation, seal leaks, etc.**
- **Enviromental infuence:** **temperature related defects, high humidity, vibration, electrical stress, crosstalk, radiation, etc.**

Soft and Hard Defects

Defects can be divided roughly into two basic groups :

- **Soft defects**
  - defects which cause speed fault
  - show up at high speed or produce some temperature
  - they need two or more test patterns for their activation and error observation (require carefully constructed transitions for defect activation);
  - require tests to be applied at speed.
  - examples: “high resistance” bridges, x-coupling, “tunneling break”
- **Hard defects**
  - defects observated at all frequencies
  - a test can be applied at slow speed
  - they need only one-pattern test set
  - example: “low resistance” bridge

Defect Manifestation and Test Methods

Defects have to be measured and modeled into the faults

They are manifested in different measurable manners:

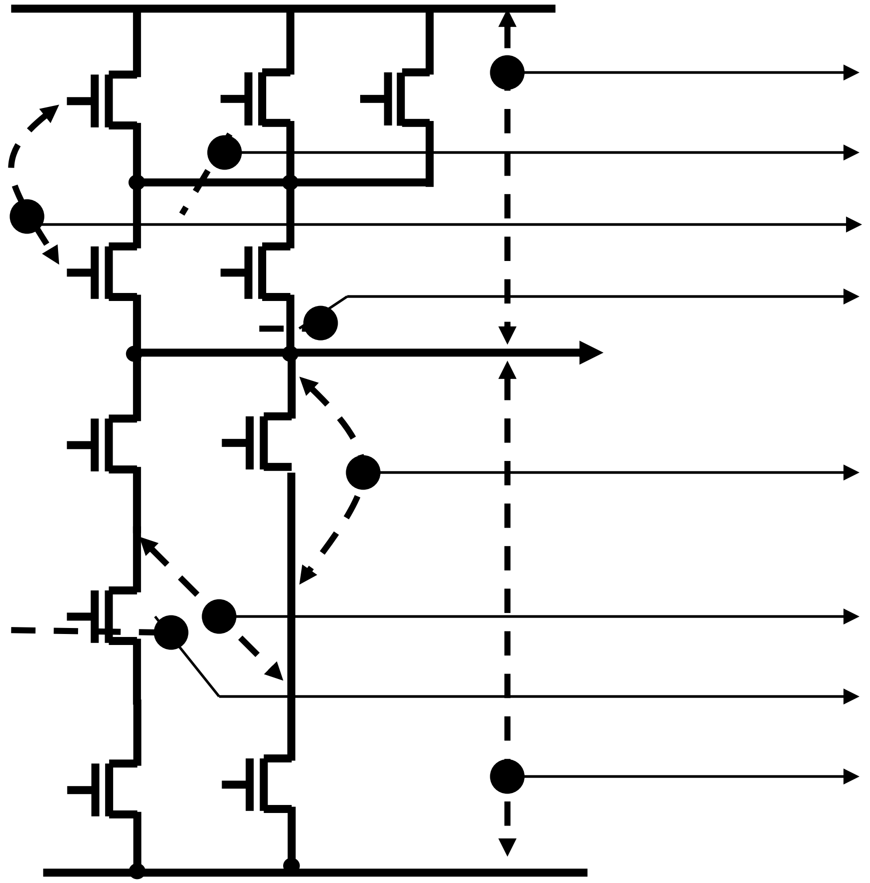
- by changing a logical value on a circuit node (Boolean testing, or testing at the logical level)
- by changing time specifications (At-speed testing)
- by increasing the steady state supply current (IDDQ testing)
- by variation in one or a set of parameters such that their specific distribution in a circuit makes it fall out of specifications

The test methods listed are not replacable

They all have to be used for achieving high quality of testing

Transistor Level Faults

Logic level interpretations:



Stuck-at-1

Broken (change of the function)

Bridging

Stuck-open

(change of the number of **states)**

Stuck-on (change of the function)

Short (change of the function)

Stuck-off (change of the function)

Stuck-at-0

Süsteemide diagnostika

3. Rikete modelleerimine

- 3.1. Rikete klassifikatsioon
- 3.2. Loogikatasandi konstantrikked
- 3.3. Tingimuslikud rikked
- 3.4. Kõrgtasandi rikked

Structural and Functional Fault Modeling

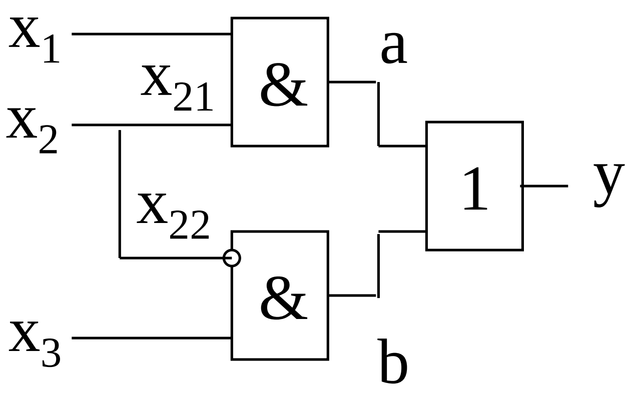
Classification of fault models

Fault models are: **explicit** and **implicit**

- **explicit** faults may be enumerated
- **implicit** faults are given by some characterizing properties

Fault models are: **structural** and **functional:**

- **structural** faults are related to structural models, they modify interconnections between components
- **functional** faults are related to functional models, they modify functions of components



Structural faults:

- line *a* is broken

- short between x_2 and x_3

Functional fault:

Instead of $y = x_1 x_2 \vee \overline{x_2} x_3$

$y = \overline{x_2} x_3$

Fault and defect modeling

Structural faults

- Structural fault models assume that components are fault-free and only their interconnections are affected:
    - a **short** is formed by connecting points not intended to be connected
    - an **open** results from the breaking of a connection
- Structural fault models are:
    - a line is **stuck at** *a fixed logic value v* (*v*$\in$ {0,1}), examples:
        - a short between ground or power and a signal line
        - an open on a unidirectional signal line
        - any internal fault in the component driving its output that it keeps a constant value
    - **bridging faults** (shorts between signal lines) with two types: AND and OR bridging faults (depending on the technology).

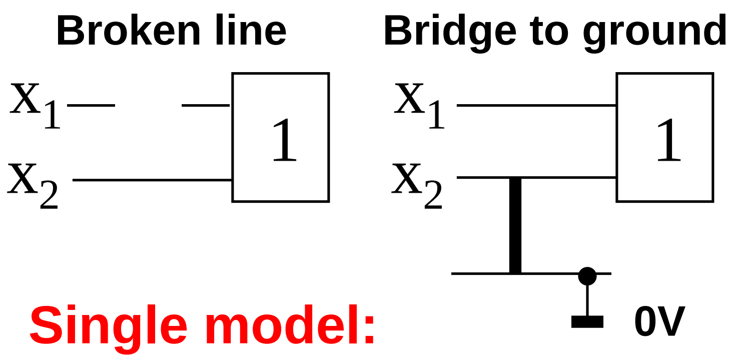
Structural Logic Level Fault Modeling

Why logic fault models?

- complexity of simulation reduces (many physical faults may be modeled by the same logic fault)
- one logic fault model is applicable to many technologies
- logic fault tests may be used for physical faults whose effect is not completely understood
- **they give a possibility to move from the lower physical level to the higher logic level**

Stuck-at fault model:

Two defects:



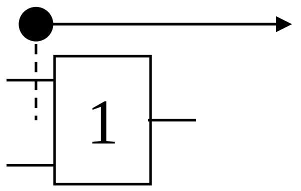
Single model:
Stuck-at-0

Gate-Level Faults: SAF Model

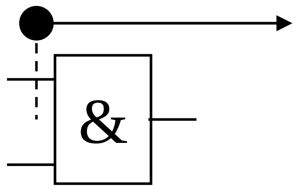
- SAF is modeled by assigning a fixed (0,1) value to a signal line: stuck_at 0 (SAF0) or stuck_at 1 (SAF1)
- SAF model is the industrial standard since 1959
- The death of the SAF model has been predicted, but several reasons and SAF properties have been persuaded that the SAF model continues living:
  - simplicity: SAF is easy to apply to a CUT
  - tractability: can be applied to millions of gates at once
  - logic behavior: fault behavior can be determined logically, so simulation is straightforward and deterministic
  - measurability: detection/non detection are easy
  - adaptability: can apply on gates, systems, transistors, RTL, etc.

Gate-Level Faults: SAF Model

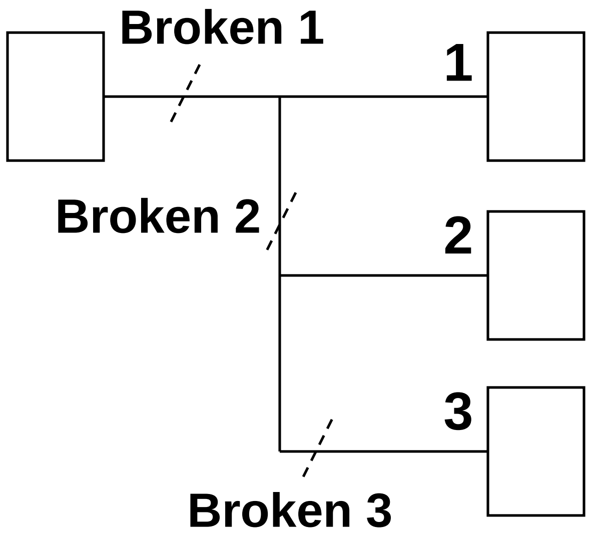
Stuck-At Fault Model



Broken (stuck-at-0)



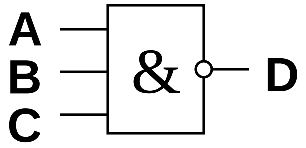
Broken (stuck-at-1)



Broken 1 $\rightarrow$ stuck branches: 1,2,3 (or stuck stem)
Broken 2 $\rightarrow$ stuck branches: 2,3
Broken 3 $\rightarrow$ stuck branches: 3

Stuck-at Fault Properties

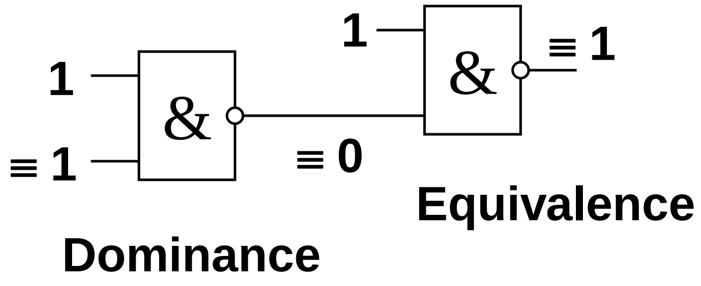
Fault equivalence and fault dominance:

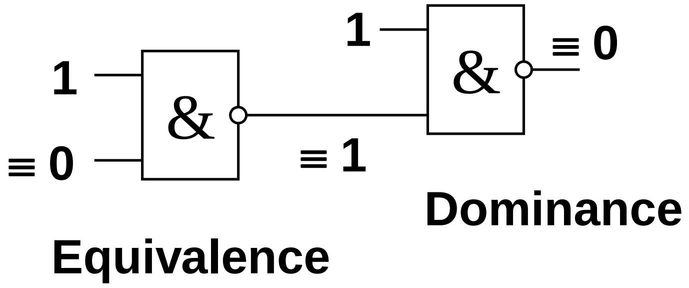


A	B	C	D	Fault class
1	1	1	0	A/0, B/0, C/0, D/1 → Equivalence class
0	1	1	1	A/1, D/0
1	0	1	1	B/1, D/0
1	1	0	1	C/1, D/0

→ Dominance classes

Fault collapsing:





Rikete dominants

Kuidas seletada dominantsi suhet:

A	B	C	D	Rikete vahelised suhted
1	1	1	0	A/0, B/0, C/0, D/1 Ekvivalents
0	1	1	1	A/1, D/0
1	0	1	1	B/1, D/0
1	1	0	1	C/1, D/0

$\rightarrow$ D/0 domineerib

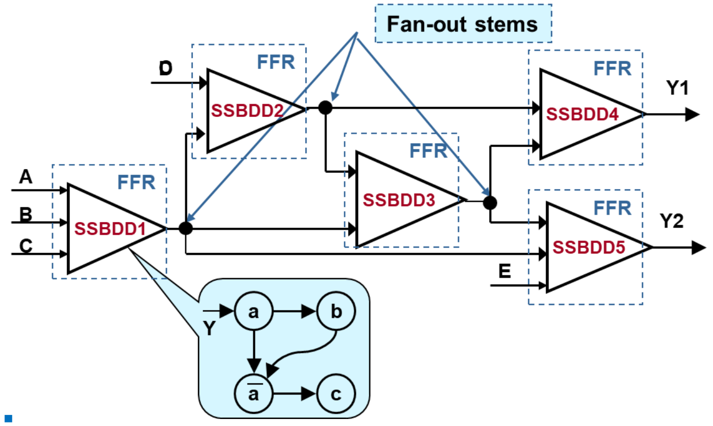
- Viin ja jää hävitavad su neerud
- Rumm ja jää hävitavad su maksa
- Viski ja jää hävitavad su südame
- Džinn ja jää hävitavad su aju
- Pepsi ja jää hävitavad su hambad

Ilmselt domineerib kõikjal jää ja on seega surmav

Impact of Fault Collapsing

Theorem 1:

A test that detects all single SAF on all inputs of tree like circuit detects all single SAF in that circuit



FFR – Fan-out-free circuit

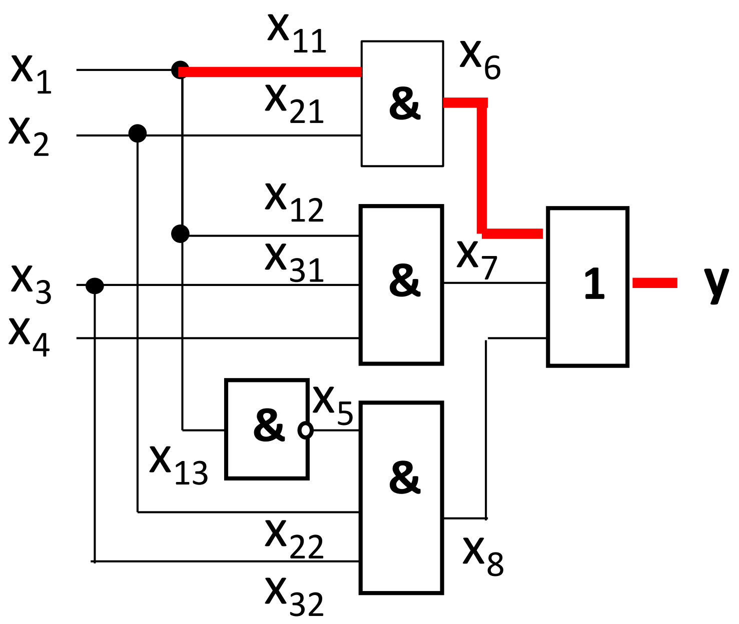
Theorem 2:

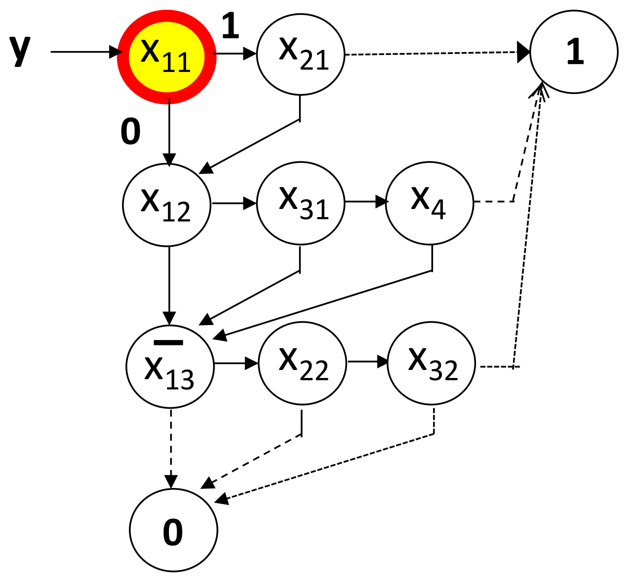
A test that detects all single SAF on all inputs and all fan-out branches of a circuit will detect all single SAFs in that circuit

The idea of **N-detect** single SAF test vectors was proposed to detect more defects not covered by the SAF model

Fault Collapsing with SSBDDs

Each node in SSBDD represents a signal path:



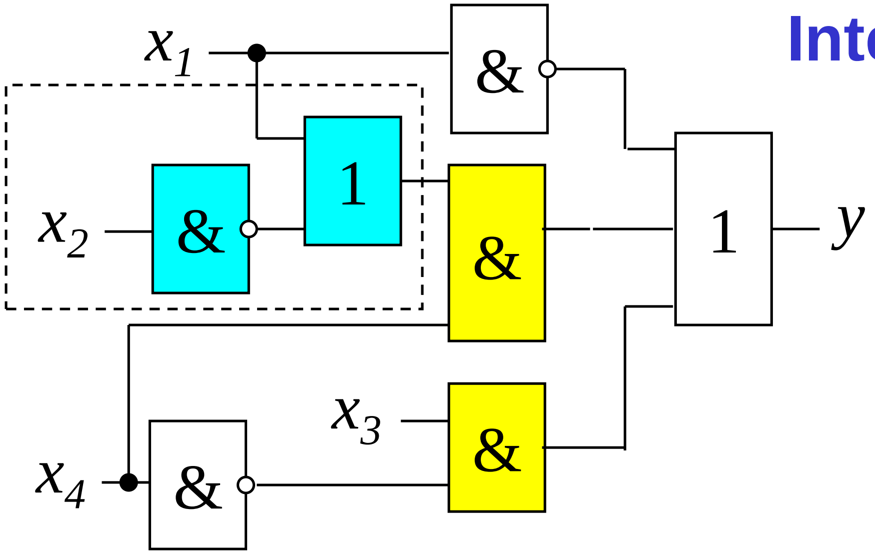


Theorem 2:

A set of test vectors that detects all single SAFs on all primary inputs and all fanout branches of a combinational logic circuit will detect all single SAFs in that circuit

Fault Redundancy

Redundant gates (bad design):



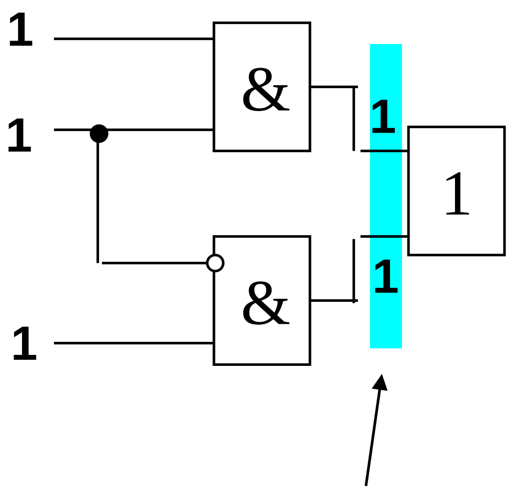
$$y = \overline{x_1} \vee (x_1 \vee \overline{x_2}) x_4 \vee x_3 \overline{x_4}$$

$$\frac{\partial y}{\partial x_2} \equiv 0$$

Faults at x_2 not testable

Optimized function: $y = \overline{x_1} \vee x_4 \vee x_3$

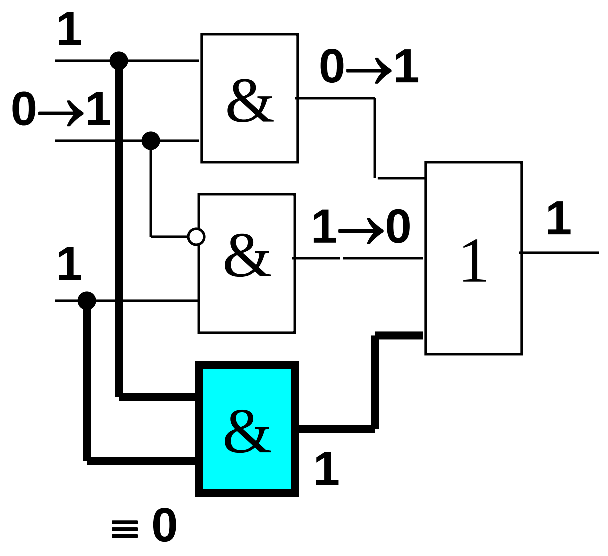
Internal signal dependencies:



Impossible pattern,
OR → XOR
not testable

Fault Redundancy

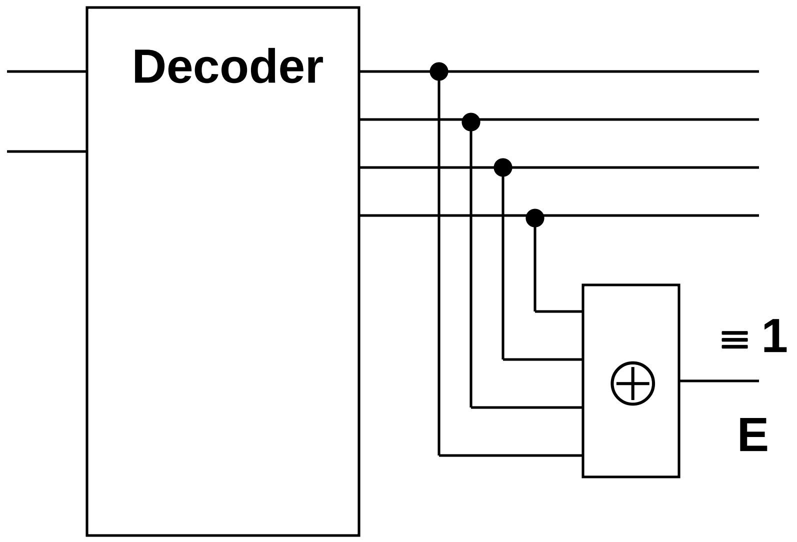
Hazard control circuitry:



Redundant AND-gate

Fault ≡ 0 is not testable

Error control circuitry:



E ≡ 1 if decoder is fault-free

Fault ≡ 0 is not testable

Fault Redundancy

- **Why this phenomenon is important and troublesame**
  - **It makes test generation (search for a proper test pattern for the given fault extremely time consuming)**
    - ***n*** – number of inputs of the circuit
    - If fault is redundant, **2^n backtracks** in search are needed
    - If 64 inputs, then **$2^{64} = 10^{19}$ backtracts**
  - **It does not allow evaluate the test quality trustworthy – the problem of test efficiency and fault coverage**
    - ***F*** – number of all faults
    - ***F_R*** – number of redundant faults
    - ***F_D*** – number of detected faults
    - ***FC*** – fault coverage
    - ***TE*** – test efficiency
  - **Fault coverage: $FC = F_D / F$**
  - **Test efficiency: $TE = F_D / (F - F_R)$**

Example:

Faults: $F = 1000$

Redundant faults: $FR = 100$

Detected faults: $FD = 880$

Fault coverage: $FC = 880/1000 = 88\%$

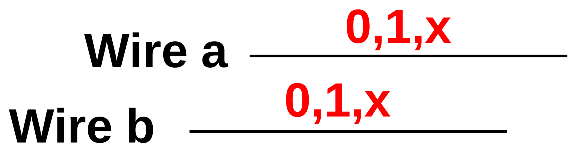
Test efficiency: $TE = 880/900 = 98\%$

Contradiction: between fault tolerance and fault coverage

Problems with Testing: Multiple Faults

- **Multiple stuck-fault (MSF) model is an extension of the single stuck-fault (SSF) where several lines can be simultaneously stuck**
- **If *n* - is the number of possible SSF sites, there are $2n$ possible SSFs, but there are**

$3^n - 1$ possible MSFs



- **If we assume that the multiplicity of faults is no greater than *k* , then the number of possible MSFs is**

$$N = \sum_{i=1}^{k} \{C_n^i\} 2^i \quad << \quad 3^n - 1 \qquad\qquad C_n^i = \frac{n!}{i!(n-i)!}$$

- **C_n^i - number of sets of i lines, 2^i – number of faults on the set**

Multiple Fault Problem

We have three wires, each of them may be in three states:
0, 1, OK

$$N = \sum_{i=1}^{k} \{C_n^i\} 2^i \quad << \quad 3^n - 1$$

$$C_n^i = \frac{n!}{i!(n-i)!}$$

$N = 3^n - 1 = 26$

Number of assumed faults	Number of combinations of faulty wires C^i_n	Number of faults on this combination of wires 2^i	Number of faults for each case *i*	Total number of multiple faults *N*
Single fault	1, 2, 3 $\rightarrow$ 3	$2^1 = 2$	6	**6**
2 faults	1,2; 1,3; 2,3 $\rightarrow$ 3	$2^2 = 4$	12	**18**
3 faults	1,2,3 $\rightarrow$ 1	$2^3 = 8$	8	**26**

The number of multiple faults is very big. However, their consideration is needed because of possible

fault masking

Multiple Fault Testing

- ✓ **2n** single faults (SSAF) vs. **3^n – 1** multiple faults (MSAF)

Two approaches to testing:

Devil's advocate

- ✓ **Goal:** to test and identify **faults**
- ✓ Does not work because of huge number of multiple fault combinations

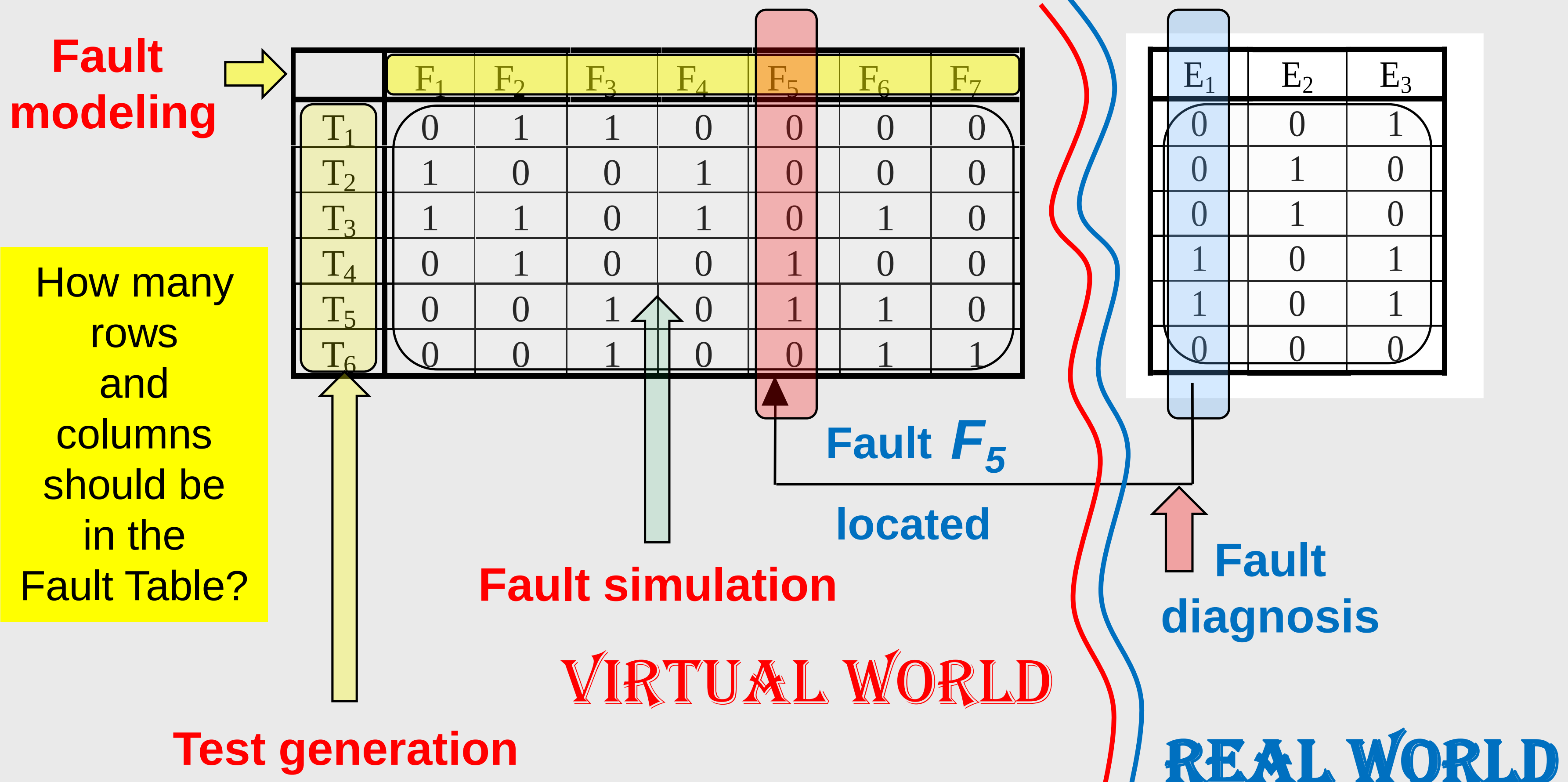
Angel's advocate

- ✓ **Goal:** to identify **fault-free lines**
- ✓ **State of the Art**: Test generation using **test pairs**

Test Related Basic Problems

Fault table (Solutions of Diagnostic equations)

Test experiment data



	F_1	F_2	F_3	F_4	F_5	F_6	F_7
T_1	0	1	1	0	0	0	0
T_2	1	0	0	1	0	0	0
T_3	1	1	0	1	0	1	0
T_4	0	1	0	0	1	0	0
T_5	0	0	1	0	1	1	0
T_6	0	0	1	0	0	1	1

E_1	E_2	E_3
0	0	1
0	1	0
0	1	0
1	0	1
1	0	1
0	0	0

How many rows and columns should be in the Fault Table?

Fault Diagnosis Dilemmas

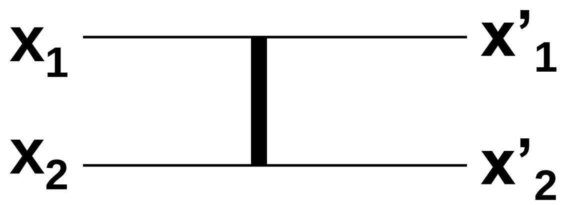
Diagnosis method	Fault table						Test result
Devil's advocate approach		**Tested faults**					**Passed**
				Tested faults			Failed
			Tested faults				Failed
Single fault assumption					Fault candi-dates		**Diagnosis**
Multiple faults allowed		**?**	**Fault candidates**				
Angel's advocate		**Proved OK**			**Fault candidates**		

Süsteemide diagnostika

Bridging Faults

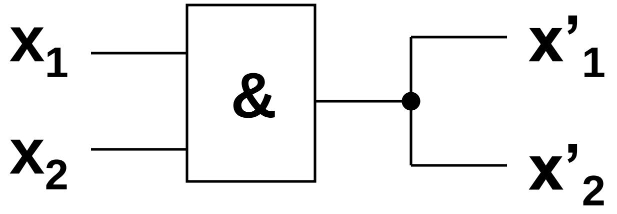
- Bridging faults model all defects that cause unintended electrical connections across two or more circuit nodes
- Physical causes of the shorts:
  - extra conducting material: e.g. photolitographic printing error, conductive particle contamination, etc.
  - missing insulating material: printing error, gate-oxide defect causing pinhole, insulating particle contamination, etc.
- Bridges have non-linear or linear properties with resistance from zero to > 1 MΩ. The typical values for resistance:
  - logical critical resistance is 100 Ω to 2 kΩ
  - timing critical resistance is 5 kΩ to 10 kΩ
- Bridging faults can be classified:
  - inter-gate shorts (can produce sequential behavior if short creates feedback)
  - intra-gate shorts

Bridging Faults

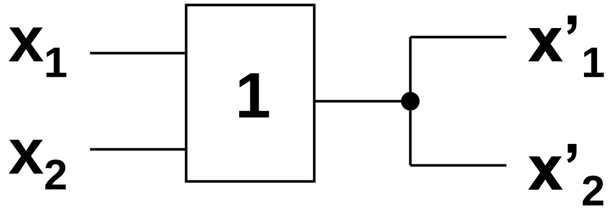


Wired AND/OR model

W-AND:



W-OR:



Fault-free		W-AND		W-OR	
x_1	x_2	x'_1	x'_2	x'_1	x'_2
0	0	0	0	0	0
0	1	0	▨ 0	▨ 1	1
1	0	▨ 0	0	1	▨ 1
1	1	1	1	1	1

Simulating of Bridging Faults

- In absence of any physical layout information, a fault list may be created by exhaustively enumerating every two nets in the design
- This method, however, is only feasible for very small circuits, because the number of all net pairs in the design grows exponentially
- For larger circuits, fault sampling may be used, where a set of net pairs is chosen randomly
- An alternative method of creating a bridging fault list without layout information is to enumerate all possible input-to-input and input-to-output shorts for each gate (or cell) in the design
- This method would require physical layout information

Advanced Bridging Fault Models

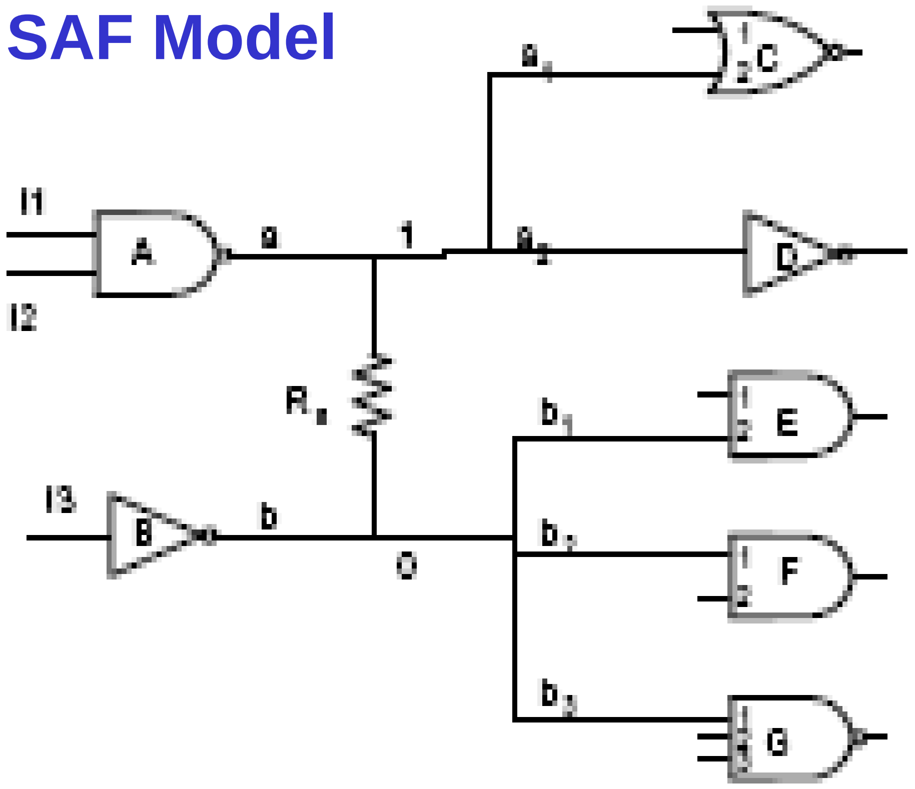
Constrained Multiple Line SAF Model

Bridge between a and b

The two branches of a and three branches of b could be interpreted by the driven gates to be any one of the 32 combinations

One corresponds to fault free situation, 31 correspond to faulty situations – 31 MLSFs

Method of implicit fault simulation: assign one branch with faulty value, and let other branches with unknown values



Copyright © G.Chen, S.Reddy, I.Pomeranz, J.Rajski, P.Engelke, B.Becker 2005

Advanced Bridging Fault Models

Constrained Multiple Line SAF Model

Advantages:

- **Method is uniform to consider opens and bridges**
- **Method does not need circuit level information such as relative strengths and threshold voltages of transistors associated with bridge**
- **Method allows different levels of model complexity and accuracy (e.g. using implicit simulation with different number of unknown values)**
- **Method is based on constrained SAF model, hence, traditional gate level tools can be used**

Delay Faults

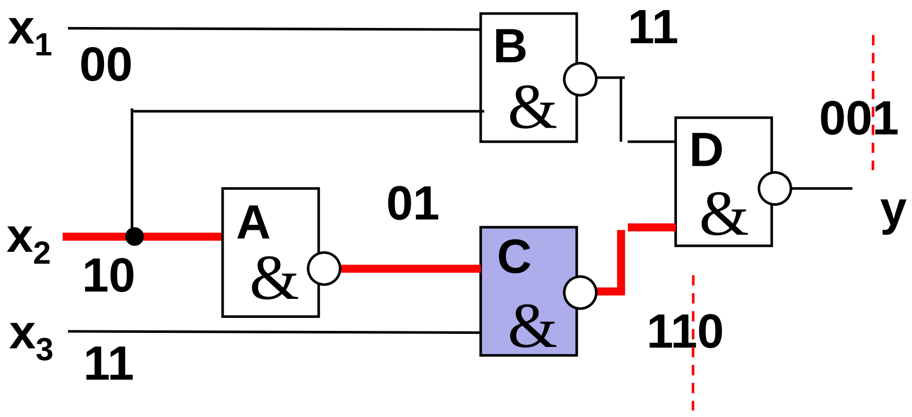
- Studies of the electrical properties of defects have shown that most of the random CMOS defects cause a timing (delay) effect rather than a other catastrophic defects (e.g. resistive bridges above a critical resistance cause delay)
- Delay fault means that a good CUT may perform correctly its function in a system, but it fails in designed timing specifications
- Delay faults could be caused by:
  - subtle manufacturing process defects,
  - transistor threshold voltage shifts,
  - increased parasitic capacitance,
  - improper timing design, etc.

Delay Fault Models

Delay faults are tested by test pattern pairs:

- the first test pattern initializes the circuit, and

- the second pattern sensitizes the fault



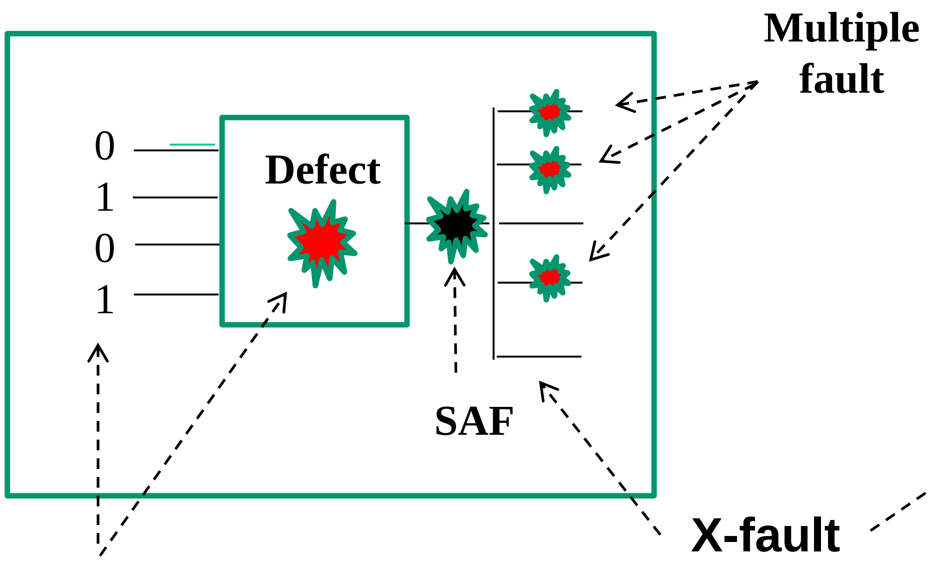
Delay fault models:

- **Gate delay fault** (delay fault is lumped at a single gate, quantitative model)
- **Transition fault** (qualitative model, gross delay fault model, independent of the activated path)
- **Path delay fault** (sum of the delays of gates along a given path)
- **Line delay fault** (is propagated through the longest senzitizable path)
- **Segment delay fault** (tradeoff between the transition and the path delay fault models)

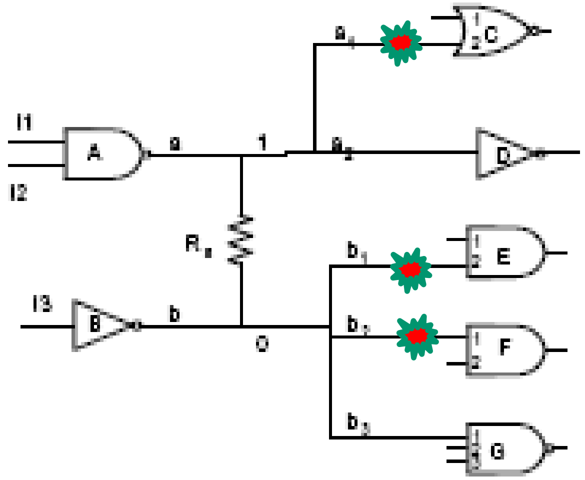
Comparison of Delay Faults

Fault models	Advantages	Limitations
Gate delay	All gates can be modeled	• Distributed failures not considered • Exact defect size not possible
Transition fault	Easy to model all gates	Distributed failures not considered
Path delay	Distributed failures considered	Impossible to enumerate all paths
Line delay	• All gates are modeled • Distributed failures considered • Better coverage metric • Additional fault coverage by using multi-path technique	• Existence of nonrobust test • May fail for some shorter paths
Segment delay	Considers general delay defect from spot to distributed failures	Longest delay path may not be tested

Extended Fault Models



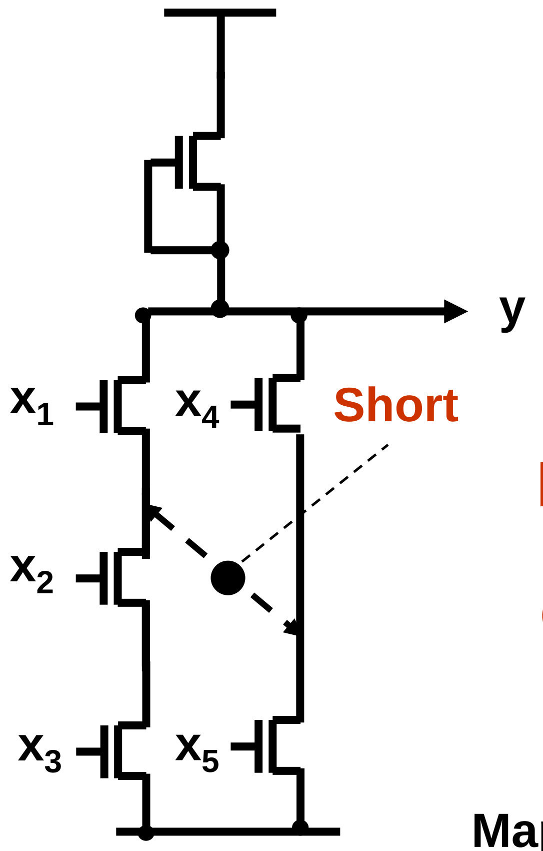
Resistive bridge fault



Conditional fault
Pattern fault
Constrained SAF
Single faulty signal

X-fault
Byzantine fault
Bridges
Stuck-opens
Multiple faulty signal

Mapping Transistor Faults to Logic Level



A transistor fault causes a change in a logic function not representable by SAF model

Function: $y = x_1 x_2 x_3 \vee x_4 x_5$

Faulty function: $y^d = (x_1 \vee x_4)(x_2 x_3 \vee x_5)$

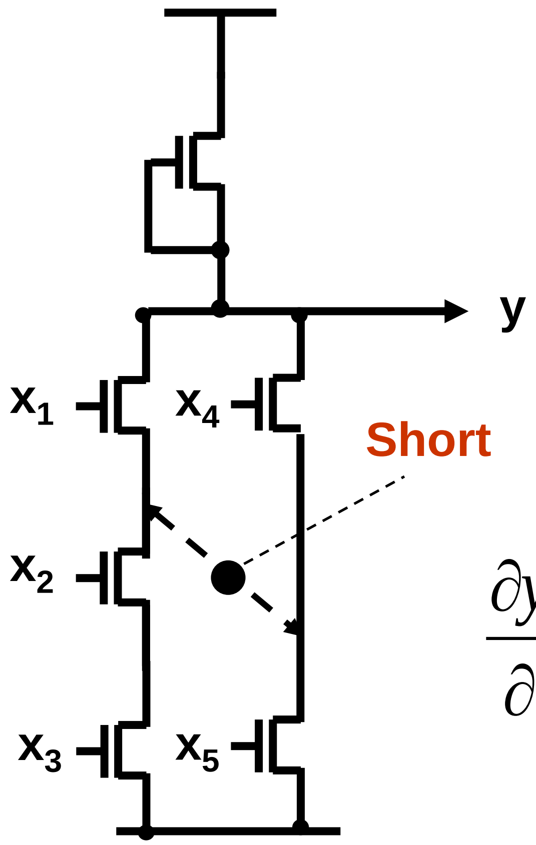
Defect variable: $d = \begin{cases} 0 - \text{defect } d \text{ is missing} \\ 1 - \text{defect } d \text{ is present} \end{cases}$

Generic function with defect:

$$y^* = (y \wedge \overline{d}) \vee (y^d \wedge d)$$

Mapping the physical defect onto the logic level by solving the equation: $\dfrac{\partial y^*}{\partial d} = 1$

Mapping Transistor Faults to Logic Level



Function: $y = x_1x_2x_3 \vee x_4x_5$

Faulty function: $y^d = (x_1 \vee x_4)(x_2x_3 \vee x_5)$

Generic function with defect:

$$y^* = (y \wedge \overline{d}) \vee (y^d \wedge d)$$

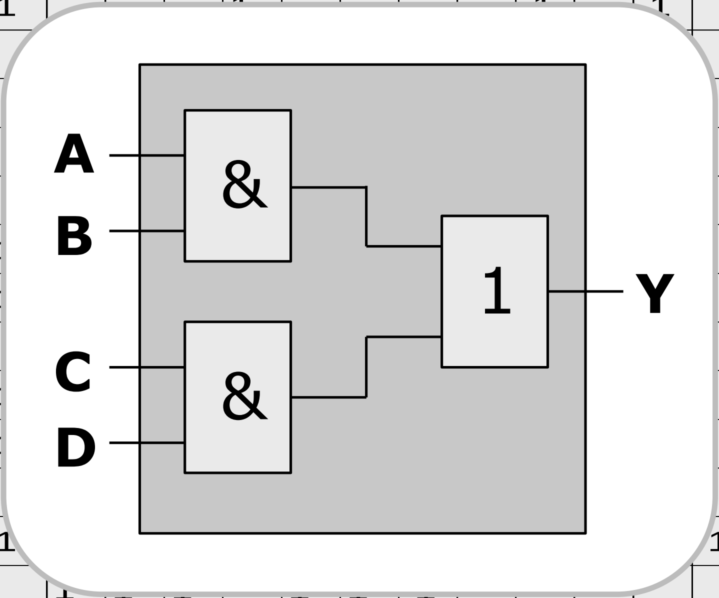
Test calculation by Boolean derivative:

$$\frac{\partial y^*}{\partial d} = \frac{\partial\left((x_1x_2x_3 \vee x_4x_5)\overline{d} \vee (x_1 \vee x_4)(x_2x_3 \vee x_5)d\right)}{\partial d} =$$

$$= x_1\overline{x_2}\,\overline{x_4}x_5 \vee x_1\overline{x_3}\,\overline{x_4}x_5 \vee \overline{x_1}x_2x_3x_4\overline{x_5} = 1$$

Fault Table: Mapping Defects to Faults

i	Fault d_i	Erroneous function f^{di}	p_i	Input patterns t_j															
				0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	B/C	not((B*C)*(A+D))	0.010307065				1								1	1	1		
2	B/D	not((B*D)*(A+C))	0.000858922				1								1	1		1	
3	B/N9	B*(not(A))	0.043375564	1	1	1					1	1	1	1					
4	B/Q	B*(not(C*D))	0.007515568	1	1	1						1	1	1		1	1	1	
5	B/VDD	not(A+(C*D))	0.001717844									1	1	1					
6	B/VSS	not(C*D)	0.035645265													1	1	1	
7	A/C	not((A*C)*(B+D))	0.098990767				1				1					1	1		
8	A/D	not((A*D)*(B+C))	0.013098561				1				1					1		1	
9	A/N9	A*(not(B))	0.038651492	1	1	1													
10	A/Q	A*(not(C*D))	0.025982392	1	1	1													
11	A/VDD	not(B+(C*D))	0.000214731																
12	C/N9	not(A+B+D)+(C*(not((A*B)+D)))	0.020399399		1														
13	C/Q	C*(not(A*B))	0.033927421	1	1		1												
14	C/VSS	not(A*B)	0.005153532				1												
15	D/N9	not(A+B+C)+(D*(not((A*B)+C)))	0.007730298			1													
16	D/Q	D*(not(A*B))	0.149452437	1		1	1												
17	N9/Q	not((A*B)+(B*C*D)+(A*C*D))	0.143654713				1												
18	N9/VDD	not((C*D)+(A*B*D)+(A*B*C))	0.253382006																
19	Q/VDD	SA1 at Q	0.014386944				1												1
20	Q/VSS	SA0 at Q	0.095555078	1	1	1		1	1	1		1	1	1					



Generalization: Functional Fault Model

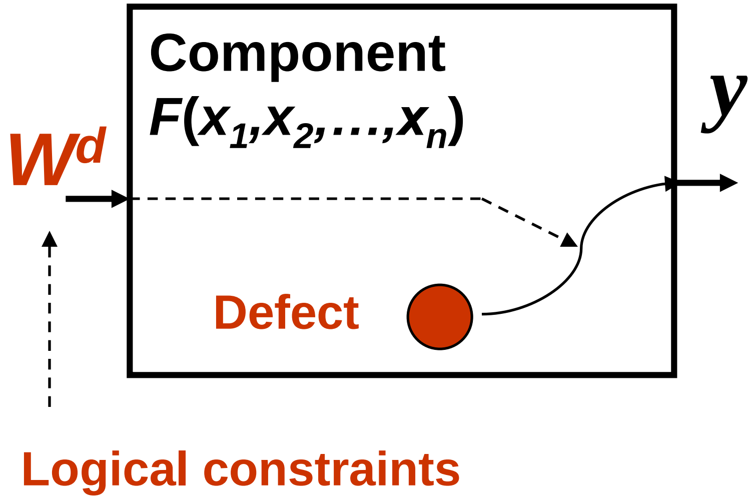
Constraints calculation:

Fault-free Faulty

$$y^* = F^*(x_1, x_2, \dots, x_n, d) = \bar{d}F \vee dF^d$$

$d = 1$, if the **defect** is present

Component with defect:



Constraints:

$$\boxed{W^d = \frac{\partial y^*}{\partial d} = 1}$$

Fault model:

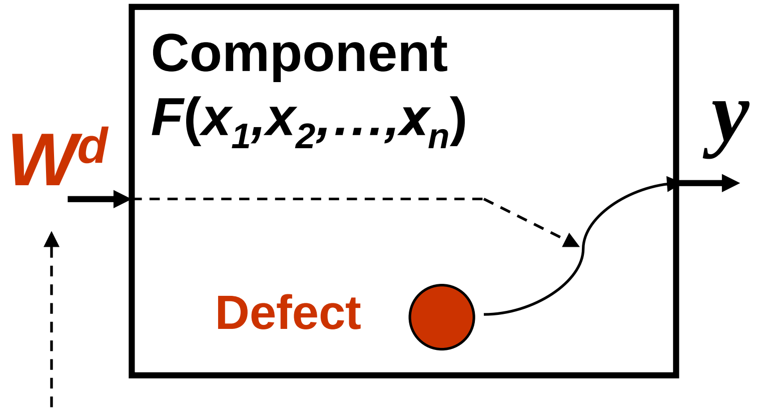
(dy, W^d), $(dy, \{W_k^d\})$

Functional Fault Model Examples

Constraints:

$$W^d = \frac{\partial y^*}{\partial d} = 1$$

Component with defect:



Logical constraints

Constraints examples:

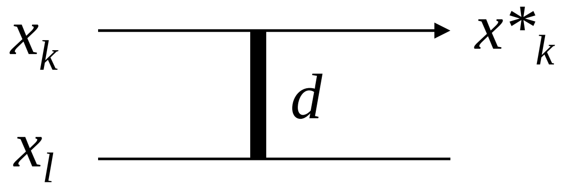
N	Fault (defect)	Constraints
1	SAF $x \equiv 0$	x = 1
2	SAF $x \equiv 1$	x = 0
3	Short between x and z	x = 1, z = 0
4	Exchange of x and z	x = 1, z = 0
5	Delay fault on x	x = 1, x' = 0

FF model:

(dy, W^d), $(dy, \{W_k^d\})$

Synthesis of a Functional Fault Model

Example:



Bridging fault **between leads x_k and x_l**

$$x_k^* = \bar{d}x_k \vee dx_k^{\,d} = \bar{d}x_k \vee dx_k x_l = x_k(\bar{d} \vee x_l)$$

$x_k^* = f(x_k, x_l, \mathbf{d})$

$$W^d = \frac{\partial x_k^*}{\partial d} = x_k \overline{x_l}$$

Wired-AND model

The condition $W^d = x_k \overline{x_l} = 1$ **means that**

in order to detect the short between leads x_k and x_l
on the lead x_k
we have to assign to x_k the value 1 and to x_l the value 0.

Synthesis of a Functional Fault Model

Example:

A short between leads x_k and x_l changes the combinational circuit into sequential one

$$y^* = \overline{d}(x_1x_2 \vee \overline{x_3}) \vee d(x_1x_2y \vee \overline{x_3}) =$$

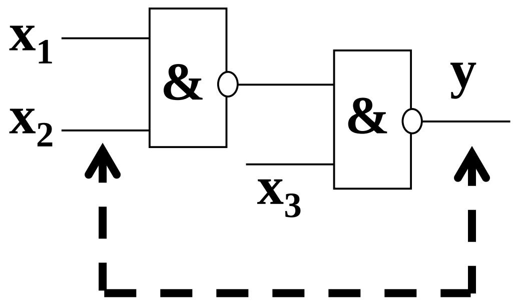
$$x_1x_2(\overline{d} \vee y')\overline{x_3}$$

$$W^d = \partial y^* / \partial d = x_1x_2x_3\overline{y'} = 1$$

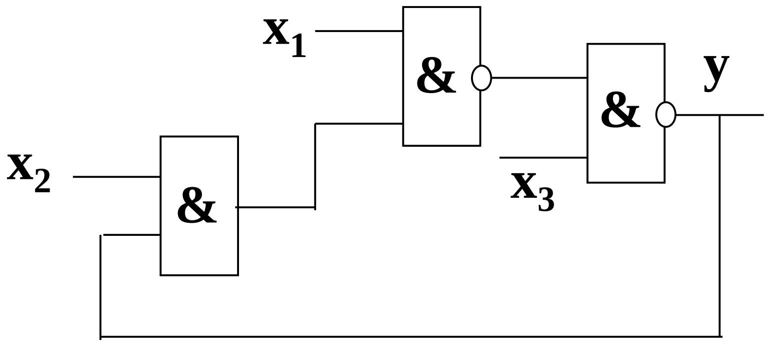
Sequential constraints:

t	x_1	x_2	x_3	y
1	**0**		**1**	**0**
2	**1**	**1**	**1**	**1**

Bridging fault causes a feedback loop:



Equivalent faulty circuit:

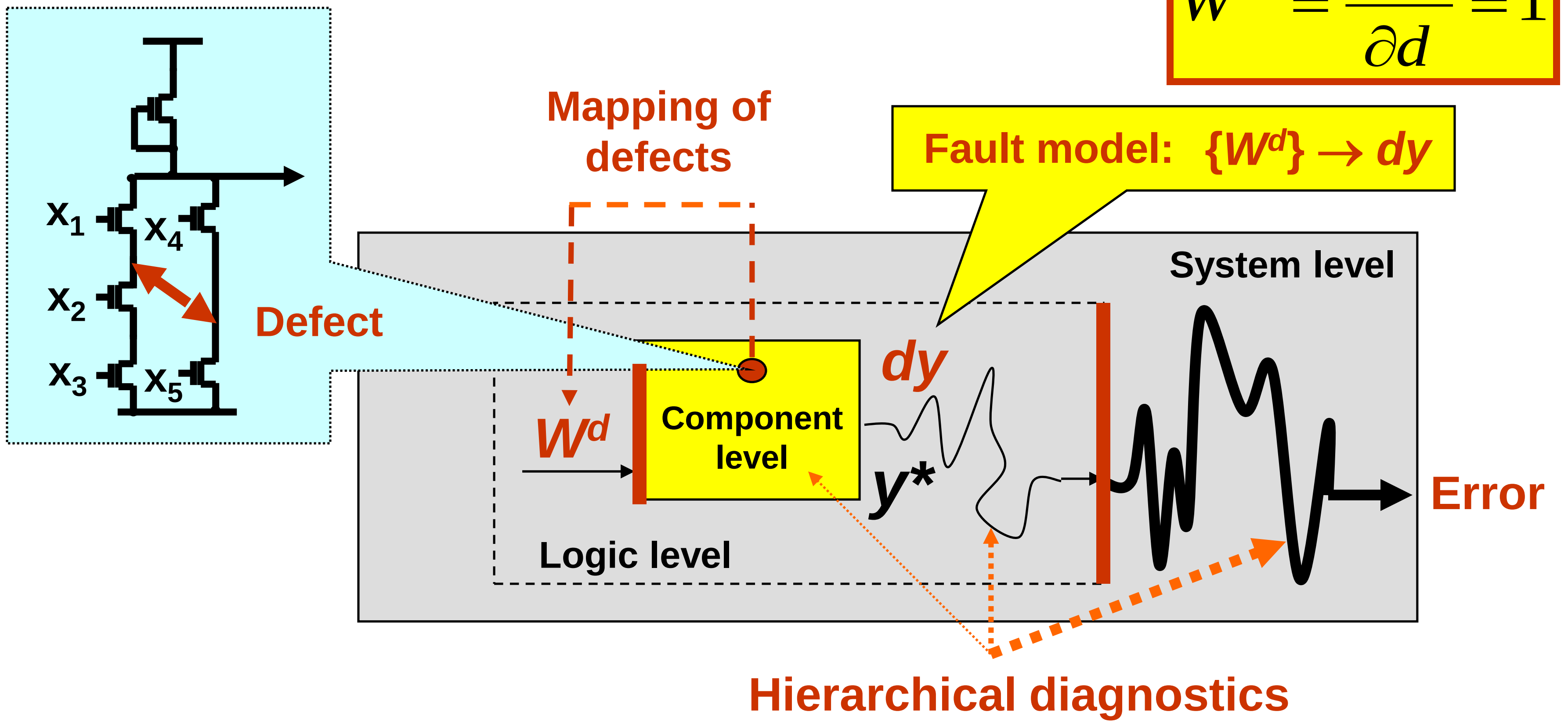


Süsteemide diagnostika

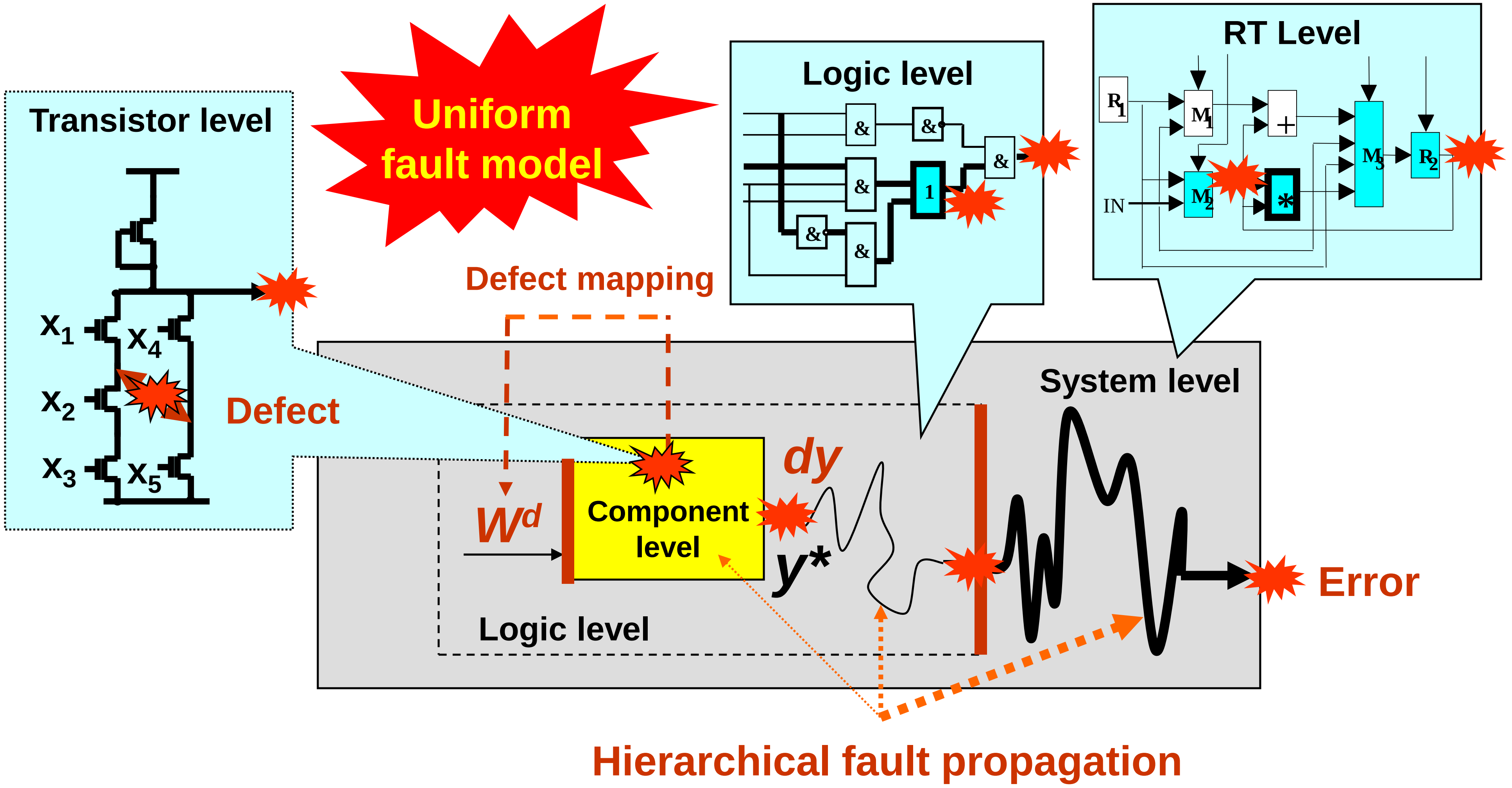
Hierarchical Fault Modeling

$$y^* = F^*(x_1,\ldots,x_n,d) = \bar{d}F \vee dF^d$$

$$\boxed{W^d = \frac{\partial y^*}{\partial d} = 1}$$



Hierarchical Diagnostic Modeling of Systems



Hierarchical fault propagation

Motivations for High-Level Fault Models

Current situation:

- The efficiency of test generation (quality, speed) is highly depending on
  - the description method (level, language), and
  - fault models
- Because of the growing complexity of systems, gate level methods have become obsolete
- High-Level methods for diagnostic modeling are today emerging, however they are not still mature

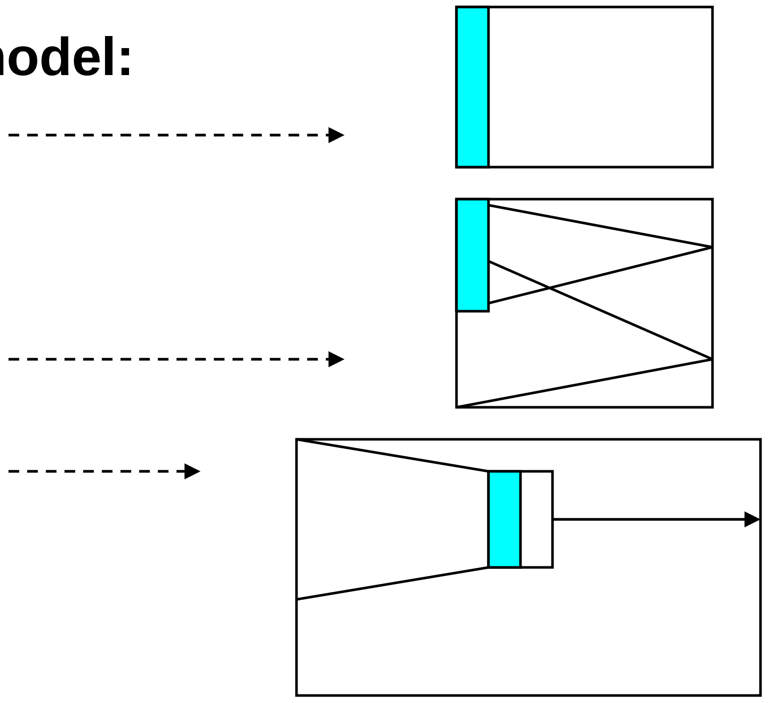
Main disadvantages:

- The known methods for fault modeling are
  - dedicated to special classes (i.e. for microprocessors, for RTL, VHDL etc. languages...), not general
  - not well defined and formalized

Fault Models for Combinational Circuits

Exhaustive combinational fault model:

- exhaustive test patterns
- pseudoexhaustive test patterns
  - exhaustive output line oriented test patterns
  - exhaustive module oriented test patterns



Fault Models for High-Level Components

Decoder:

- instead of correct line, incorrect is activated
- in addition to correct line, additional line is activated
- no lines are activated

Multiplexer (n inputs $\log_2 n$ control lines):

- stuck-at - 0 (1) on inputs
- another input (instead of, additional)
- value, followed by its complement
- value, followed by its complement on a line whose address differs in 1 bit

Memory fault models:

- one or more cells stuck-at - 0 (1)
- two or more cells coupled

Fault models and Tests

Dedicated functional fault model for multiplexer:

- **stuck-at-0 (1) on inputs,**
- **another input (instead of, additional)**
- **value, followed by its complement**
- **value, followed by its complement on a line whose address differs in one bit**

Functional fault model

Test description

Register Level Fault Models

RTL statement:

$$K: (\mathit{If}\ T,C)\ \ R_D \leftarrow F(R_{S1}, R_{S2}, \ldots R_{Sm}), \rightarrow N$$

Components (variables) of the statement:

K	- label
T	- timing condition
C	- logical condition
R_D	- destination register
R_S	- source register
F	- operation (microoperation)
$\leftarrow$	- data transfer
$\rightarrow$ N	- jump to the next statement

RT level faults:

$K \rightarrow K'$	- label faults
$T \rightarrow T'$	- timing faults
$C \rightarrow C'$	- logical condition faults
$R_D \rightarrow R_D$	- register decoding faults
$R_S \rightarrow R_S$	- data storage faults
$F \rightarrow F'$	- operation decoding faults
$\leftarrow$	- data transfer faults
$\rightarrow$ N	- control faults
$(F) \rightarrow (F)'$	- data manipulation faults

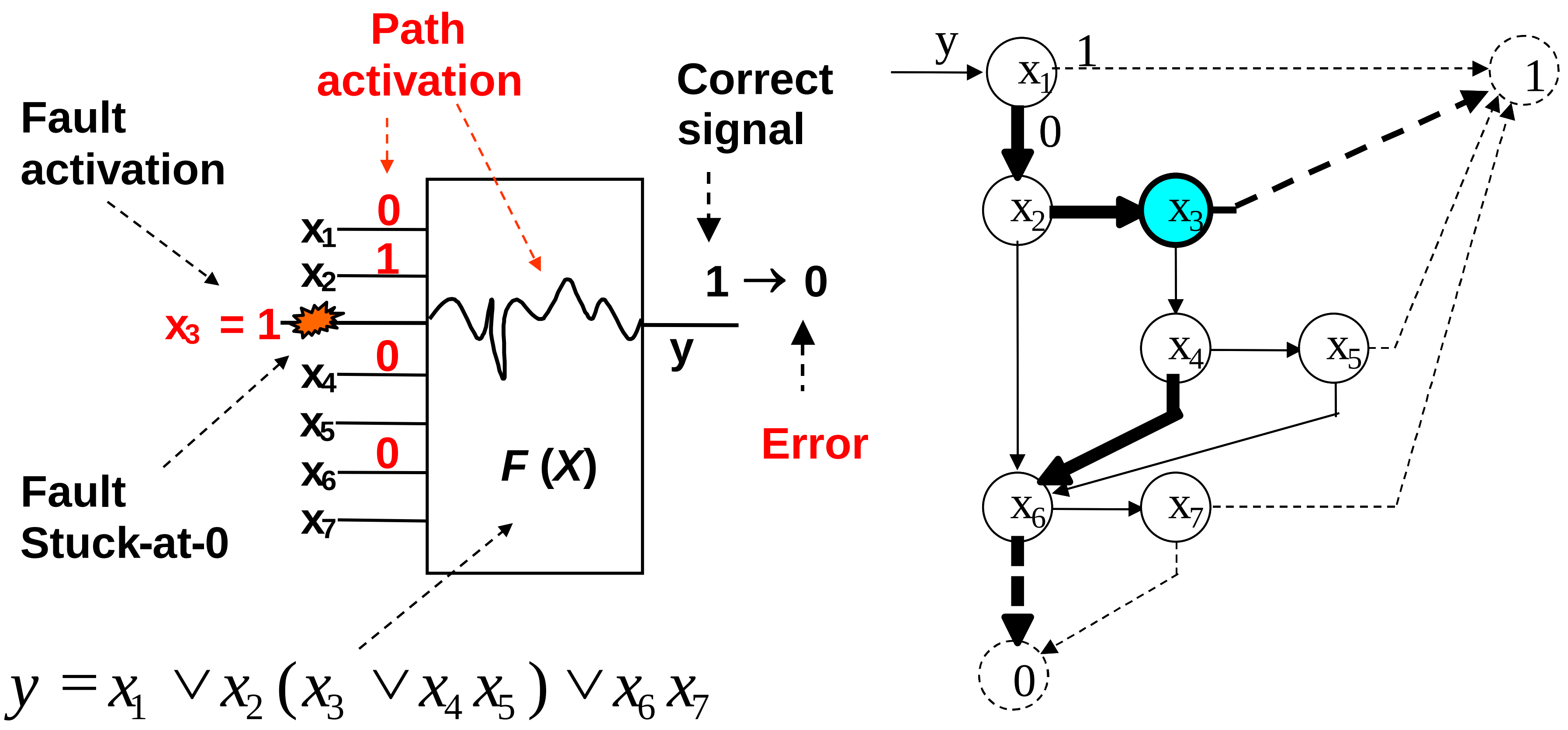
Microprocessor Fault Model

Faults affecting the operation of microprocessor can be divided into the following classes:

- addressing faults affecting register decoding
- addressing faults affecting the instruction decoding and – sequencing functions;
- faults in the data-storage function;
- faults in the data-transfer function;
- faults in the data-manipulation function.

Binary Decision Diagrams and Faults

Fault modeling on Structurally Synthesized BDDs:



$$y = x_1 \vee x_2 (x_3 \vee x_4 x_5) \vee x_6 x_7$$

High-Level Decision Diagrams and Faults

RTL-statement:

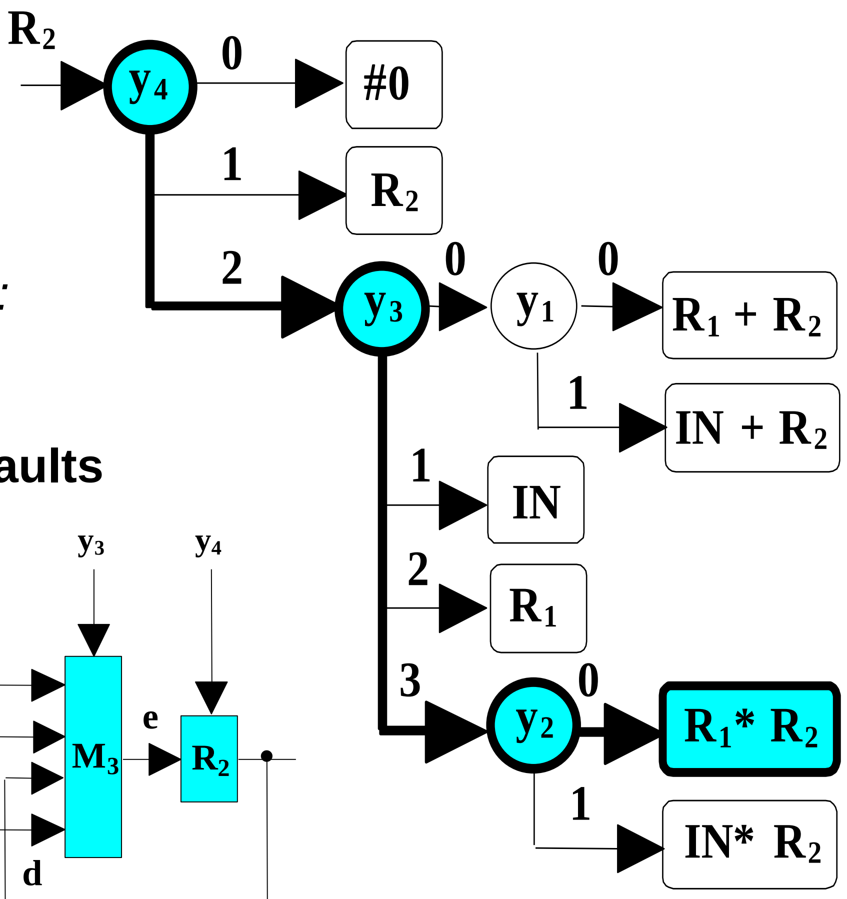
K: (*If* T,C) $R_D \leftarrow F(R_{S1},R_{S2},\ldots R_{Sm}), \rightarrow N$

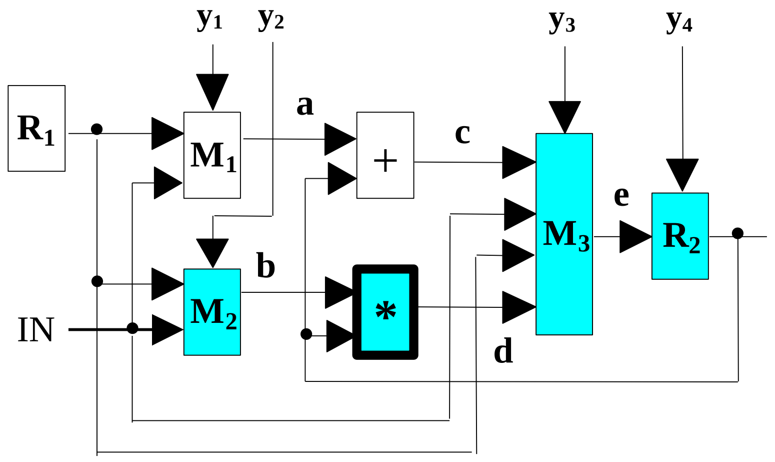
Terminal nodes

RTL-statement faults:
data storage,
data transfer,
data manipulation faults

Nonterminal nodes

RTL-statement faults:
label,
timing condition,
logical condition,
register decoding,
operation decoding,
control faults

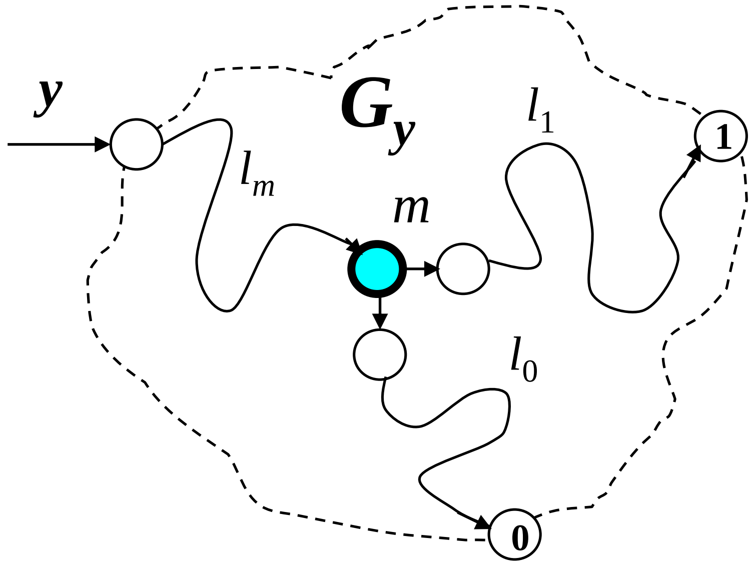




Fault Modeling on DDs

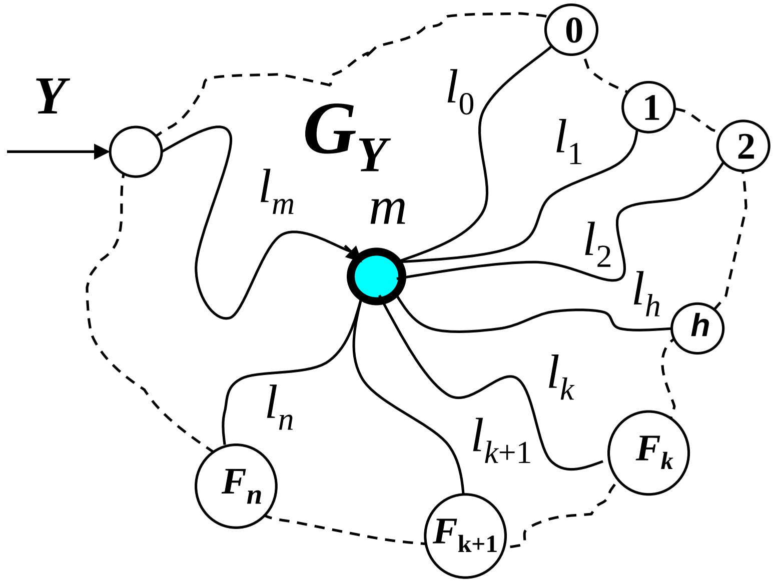
Binary DD

with 2 terminal nodes and
2 outputs
from each node



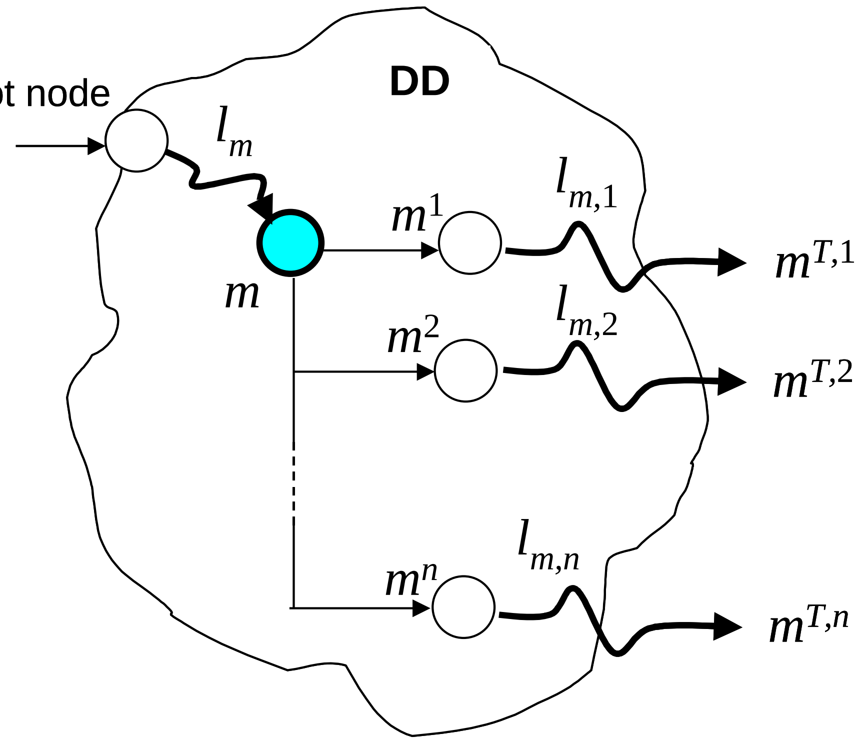
General case of DD

with n ≥ 2 terminal nodes and
n ≥ 2 outputs
from each node



Fault Modeling on DDs

- **Each path in a DD describes the behavior of the system in a specific mode of operation**
- **The faults having effect on the behaviour can be associated with nodes along the path**
- **A fault causes incorrect leaving the path activated by a test**

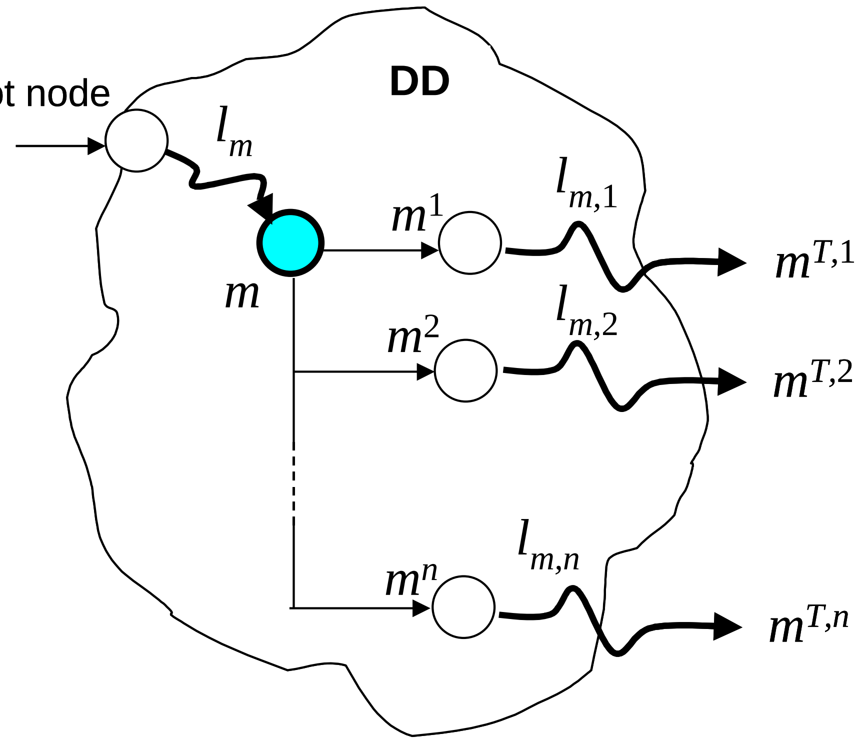


Uniform Formal Fault Model on DDs

D1: **the output edge**
for $x(m) = i$ of a node m
is always activated

D2: **the output edge**
for $x(m) = i$ of a node m
is broken

D3: **instead of the given edge,**
another edge or
a set of edges
is activated



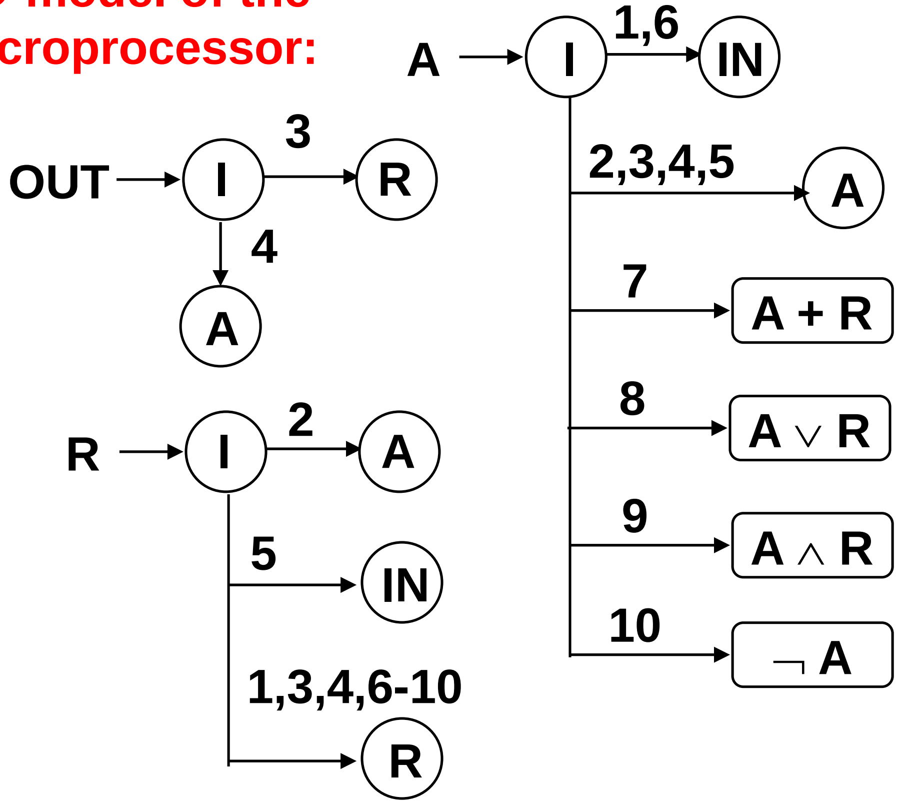
Modeling Microprocessors with DDs

High-Level DDs for a microprocessor (example):

Instruction set:

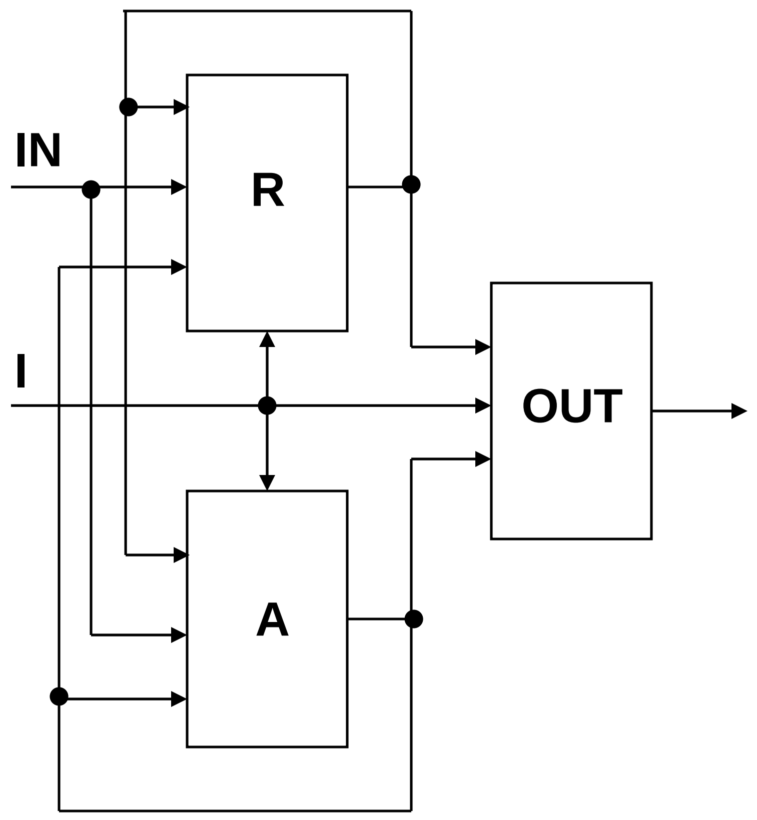
I_1:	MVI A,D	$A \leftarrow IN$
I_2:	MOV R,A	$R \leftarrow A$
I_3:	MOV M,R	$OUT \leftarrow R$
I_4:	MOV M,A	$OUT \leftarrow A$
I_5:	MOV R,M	$R \leftarrow IN$
I_6:	MOV A,M	$A \leftarrow IN$
I_7:	ADD R	$A \leftarrow A + R$
I_8:	ORA R	$A \leftarrow A \vee R$
I_9:	ANA R	$A \leftarrow A \wedge R$
I_{10}:	CMA A,D	$A \leftarrow \neg A$

DD-model of the microprocessor:

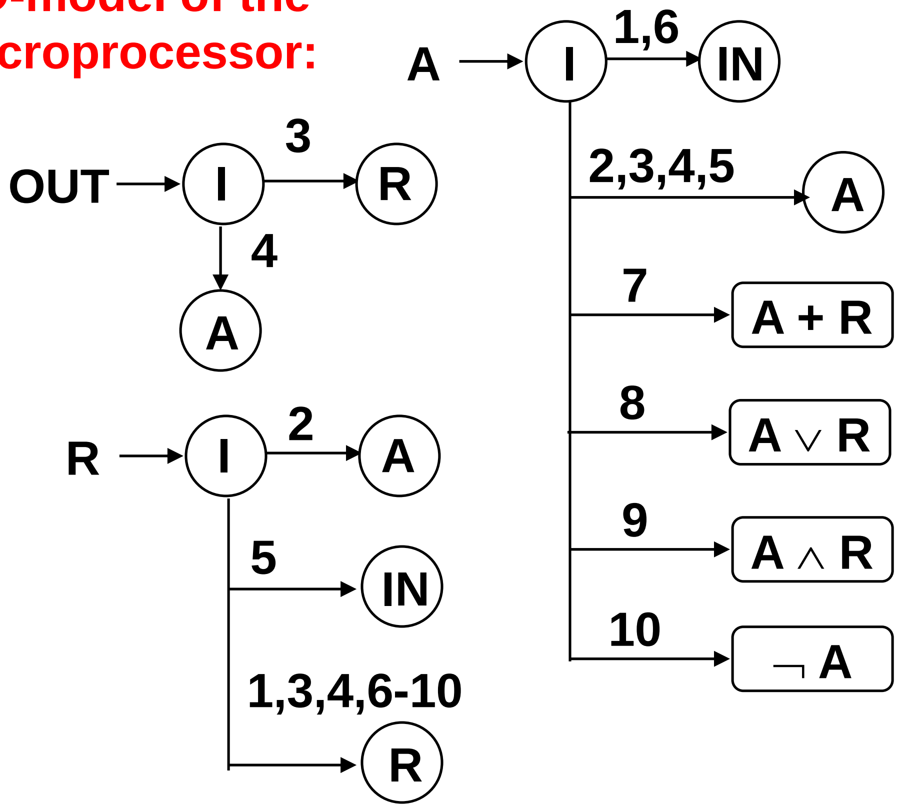


Decision Diagrams for Microprocessors

High-Level DD-based structure of the microprocessor (example):



DD-model of the microprocessor:



From MP Instruction Set to HLDDs

OP	B	Semantic	RT level operations	
0	0	READ memory	*R*(*A*1) = *M*(*A*)	*PC* = *PC* + 2
	1	WRITE memory	*M*(*A*) = *R*(*A*2)	*PC* = *PC* + 2
1	0	Transfer	*R*(*A*1) = *R*(*A*2)	*PC* = *PC* + 1
	1	Complement	*R*(*A*1) = ¬ *R*(*A*2)	*PC* = *PC* + 1
2	0	Addition	*R*(*A*1) = *R*(*A*1)+ *R*(*A*2)	*PC* = *PC* + 1
	1	Subtraction	*R*(*A*1) = *R*(*A*1)- *R*(*A*2)	*PC* = *PC* + 1
3	0	Jump	*PC* = *A*	
	1	Conditional jump	IF C=1, THEN PC = A,ELSE PC = PC + 2	

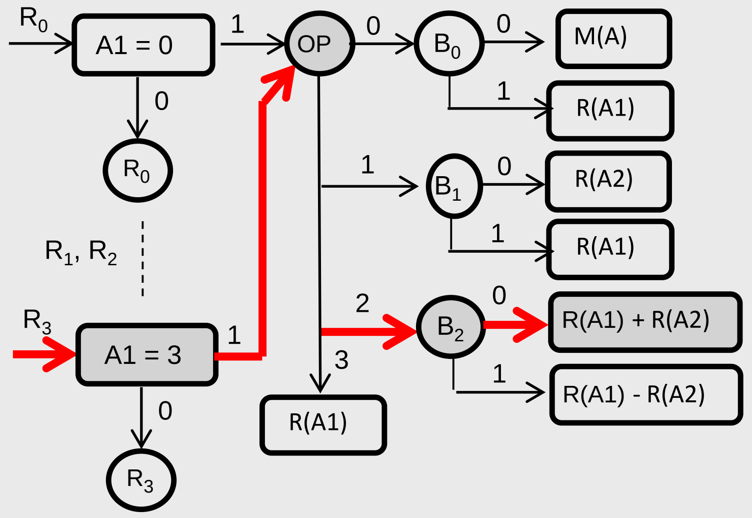
Instruction code:

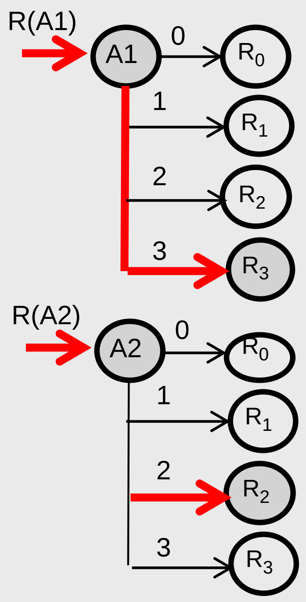
ADD A1 A2

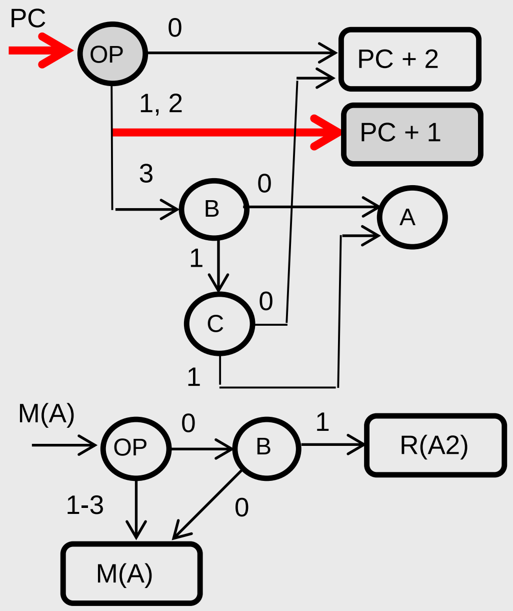
OP=2. B=0. A1=3. A2=2

$$\mathbf{R_3 = R_3 + R_2}$$

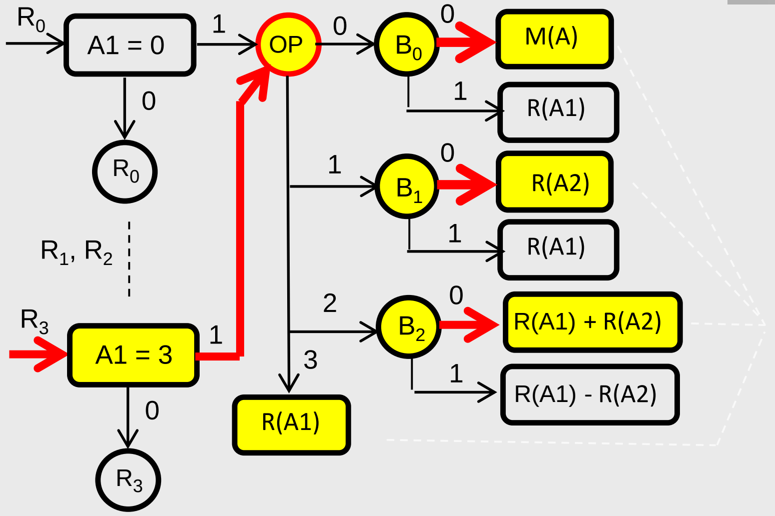
$$\mathbf{PC = PC+1}$$







Uniform Conditional Node Fault Model

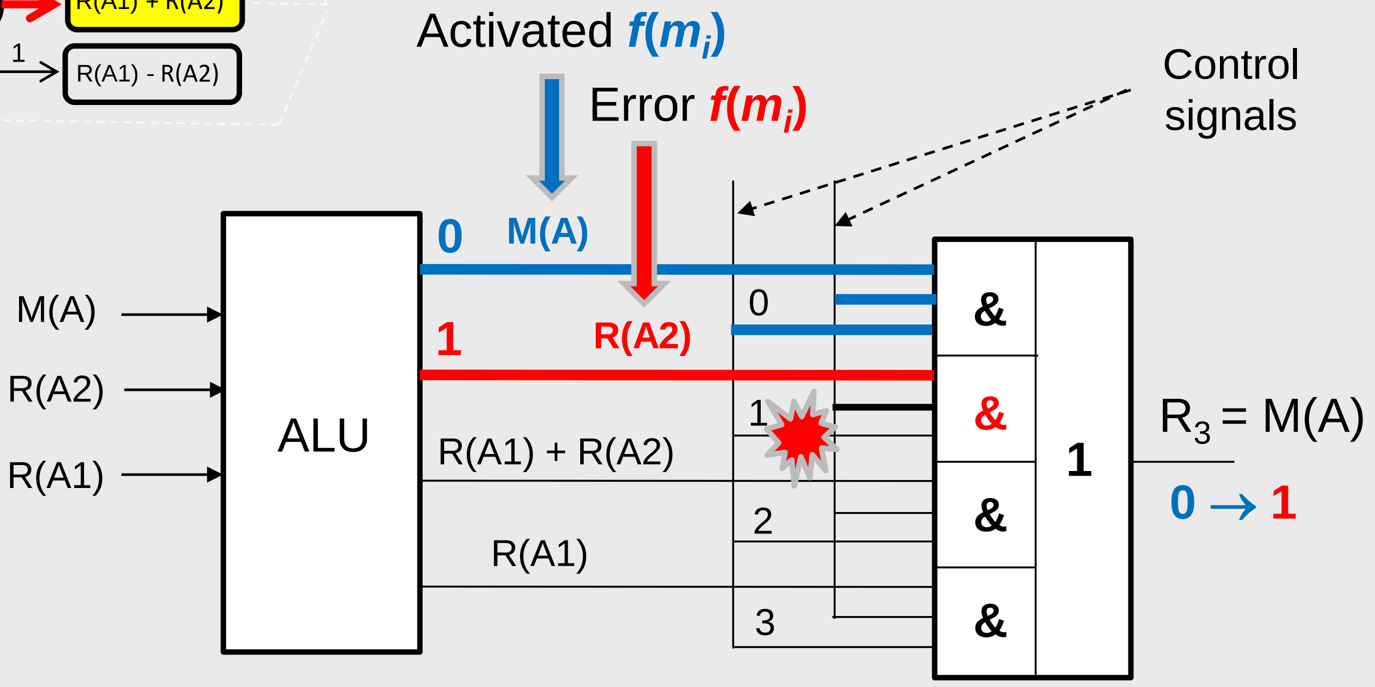


Test data calculation rules:

$\forall m^T \in M^T(OP)$: [$f(m^T) \neq$ ZERO]

$\forall m_i, m_j \in M^T(OP)$: [$(f(m_i) < f(m_j)$]

Explanation of the meaning of constraints (rules):



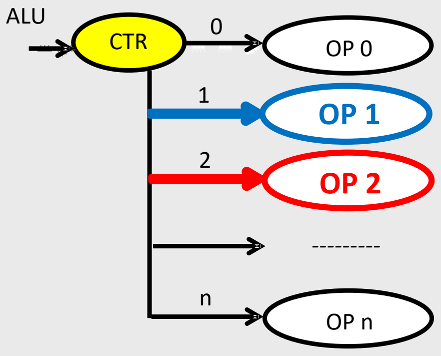
To detect the fault: R3 = M(A) ∨ R(A2) ⟶ M(A) < R(A2) is needed

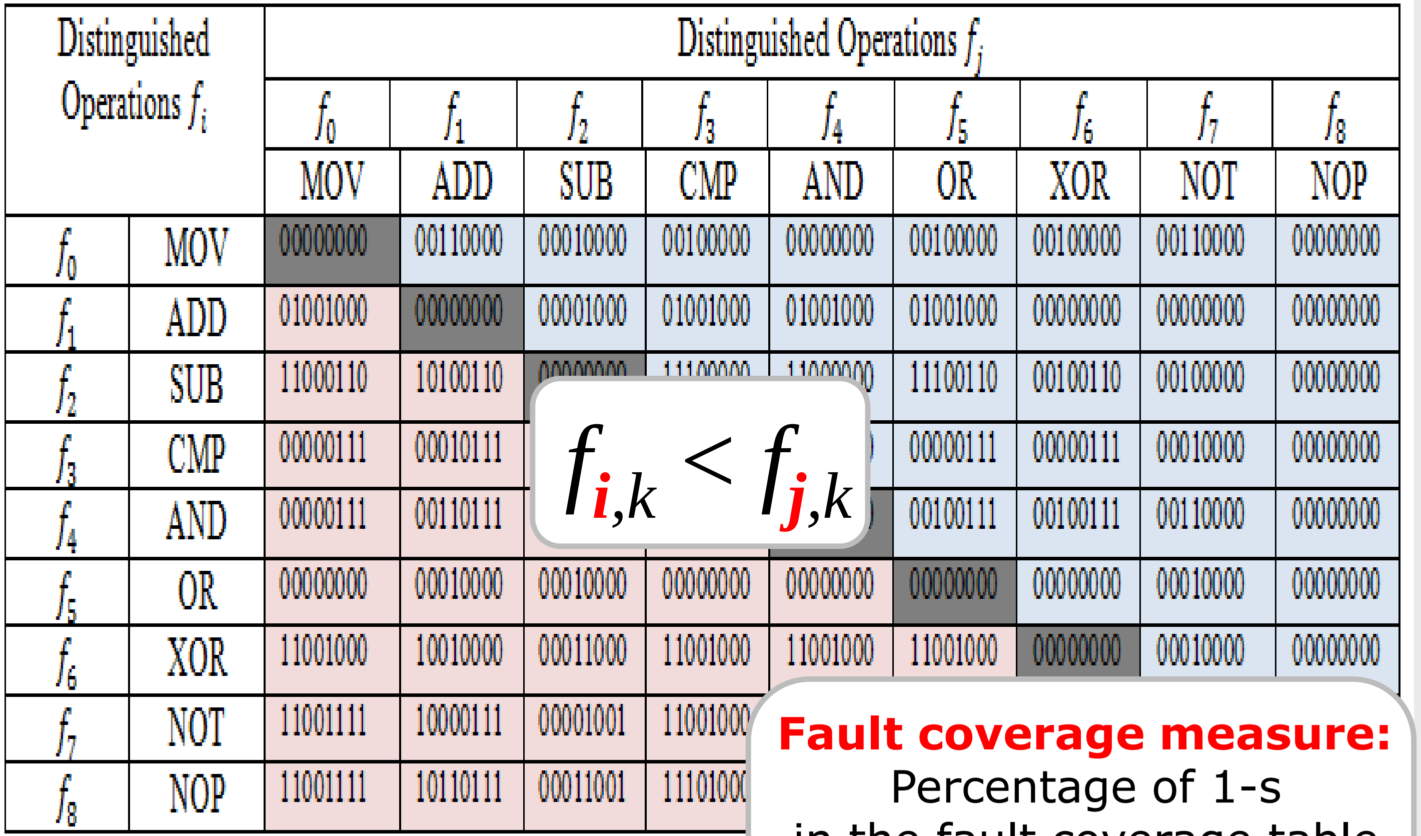
High-Level Control Fault Coverage Table

Functional fault model (for control nodes):

$\forall j\ [f_j \neq \text{ZERO})]$

$\forall i,j\text{: } \forall k\ [(f_{i,k} < f_{j,k}]$





Distinguished Operations f_i		f_0 MOV	f_1 ADD	f_2 SUB	f_3 CMP	f_4 AND	f_5 OR	f_6 XOR	f_7 NOT	f_8 NOP
f_0	MOV	00000000	00110000	00010000	00100000	00000000	00100000	00100000	00110000	00000000
f_1	ADD	01001000	00000000	00001000	01001000	01001000	01001000	00000000	00000000	00000000
f_2	SUB	11000110	10100110	00000000	11100000	11000000	11100110	00100110	00100000	00000000
f_3	CMP	00000111	00010111	[illegible]	[illegible]	[illegible]	00000111	00000111	00010000	00000000
f_4	AND	00000111	00110111	[illegible]	[illegible]	[illegible]	00100111	00100111	00110000	00000000
f_5	OR	00000000	00010000	00010000	00000000	00000000	00000000	00000000	00010000	00000000
f_6	XOR	11001000	10010000	00011000	11001000	11001000	11001000	00000000	00010000	00000000
f_7	NOT	11001111	10000111	00001001	11001000	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
f_8	NOP	11001111	10110111	00011001	1110100	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

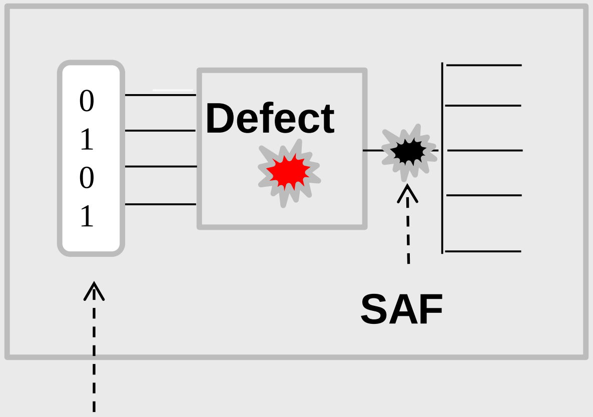
$$f_{i,k} < f_{j,k}$$

Fault coverage measure:
Percentage of 1-s
in the fault coverage table

1 – means that the constraint is satisfied by at least one pair of data operands

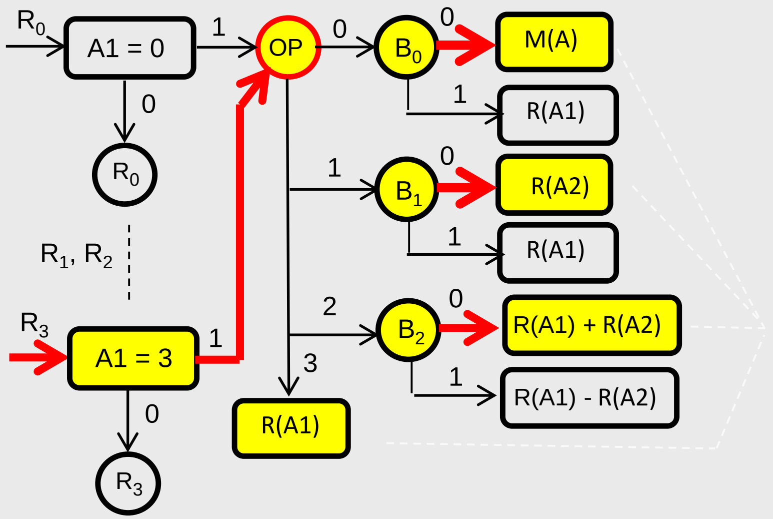
Uniform Conditional Node Fault Model

Logic level analog:
Conditional SAF model



Condition (constraint)

High level fault model: **Constraints for testing a node *OP* in HLDD:**



High-level fault model:

Test data calculation rules:

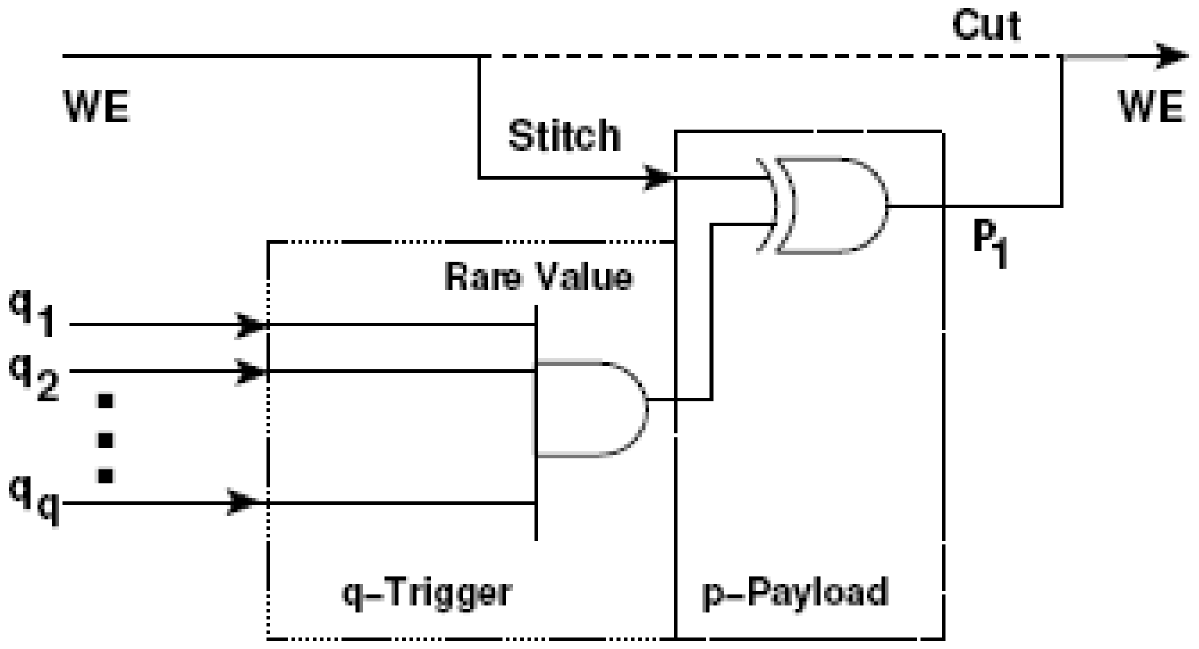
$\forall m^T \in M^T(OP)$: $[f(m^T) \neq \text{ZERO})]$

$\forall m_i, m_j \in M^T(OP)$: $[(f(m_i) < f(m_j)]$

Functional Fault Modeling: Trojans

A trojan is inserted into a main circuit at manufacturing and is mostly inactive unless it is triggered by a rare value or time event

Then it produces a payload error in the circuit, potentially catastrophic



Conclusions

- **Different fault models for different representation levels of digital systems can be replaced on DDs by the uniform node fault model**
- **It allows to represent groups of structural faults through groups of functional faults**
- **As the result, the complexity of fault representation can be reduced, and the simulation speed can be raised**
- **The fault model on DDs can be regarded as a generalization**
  - **of the classical gate-level stuck-at fault model, and**
  - **of the known higher level fault models**