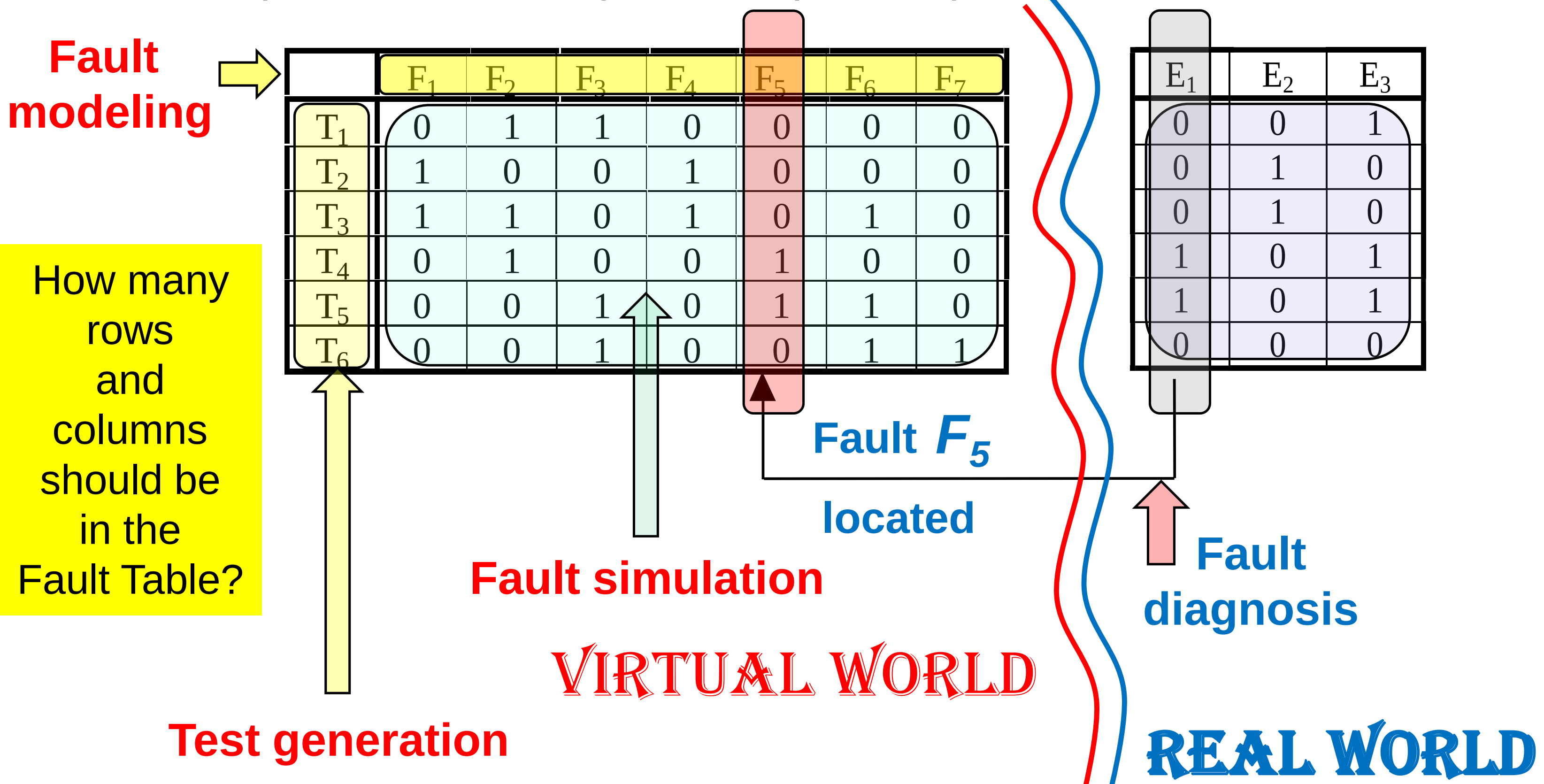
Süsteemide diagnostika

4. Testide süntees digitaalsüsteemidele

- 4.1. Deterministlik testide süntees kombinatsioonskeemidele
- 4.2. Testide genereerimine otsustusdiagrammide abil
- 4.3. Triviaalsete (pseudotäielike) testide süntees
- 4.4. Testide süntees kordsetele riketele (üldjuht)
- 4.5. Testide süntees digitaalsüsteemidele kõrgtasandil

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

Fault modeling

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

Fault simulation

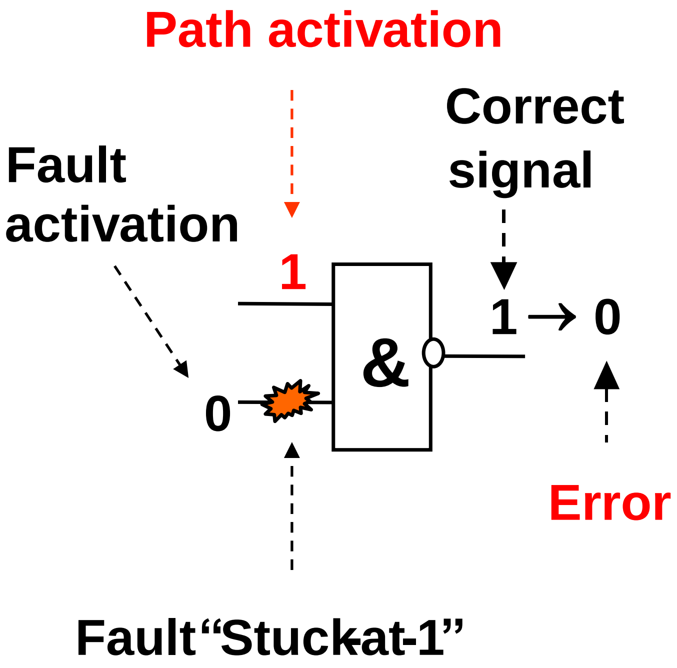
Test generation

Fault F_5 located

Fault diagnosis

Fault Propagation Problem

Logic gate



Path activation

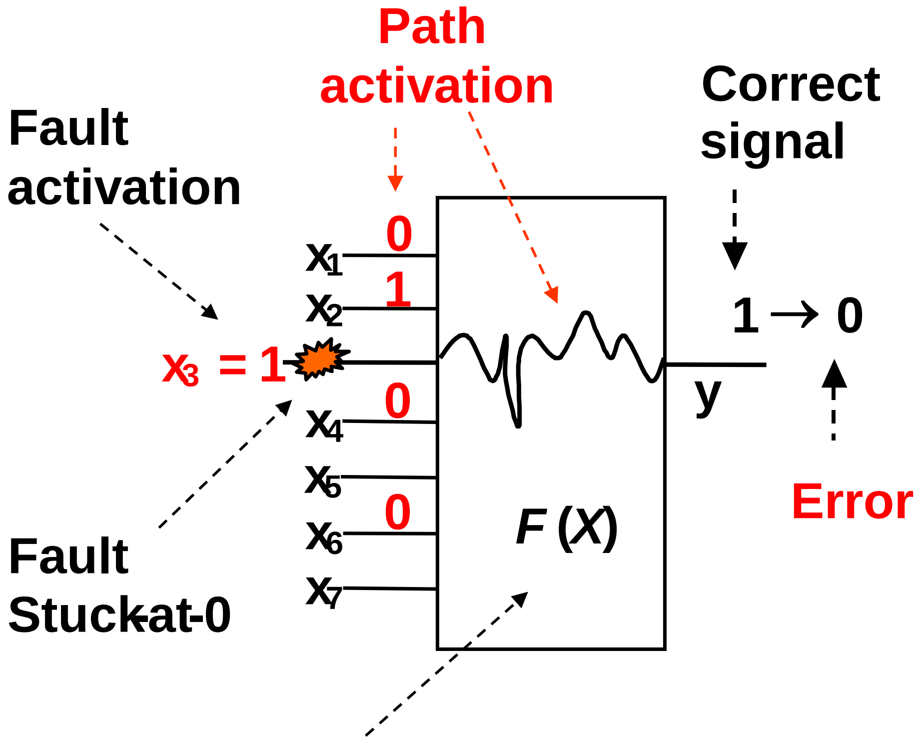
Correct signal

Fault activation

Error

Fault "Stuck-at-1"

Logic circuit



Path activation

Correct signal

Fault activation

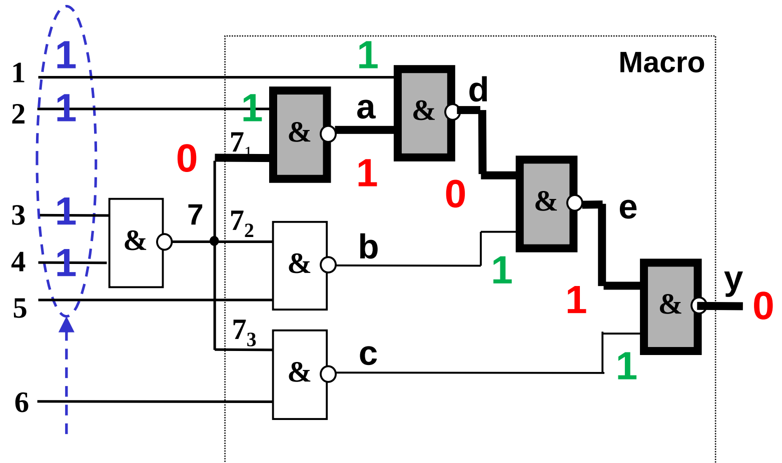
Error

Fault Stuck-at-0

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

Gate-Level Structural Test Generation

Path activation



Test pattern

Gate level test generation:

Fault sensitization (for $x_{7,1} \equiv 1$):

$x_{7,1} = 0$

Fault propagation:

$x_2 = 1$, $x_1 = 1$, $b = 1$, $c = 1$

Line justification:

$x_{7,1} = 0 \rightarrow x_7 = 0 \rightarrow \{x_3 = 1, x_4 = 1\}$

$b = 1 \rightarrow$ (already justified)

$c = 1 \rightarrow$ (already justified)

The expected result:

y = 0 - if fault is missing
y = 1 - if fault is present

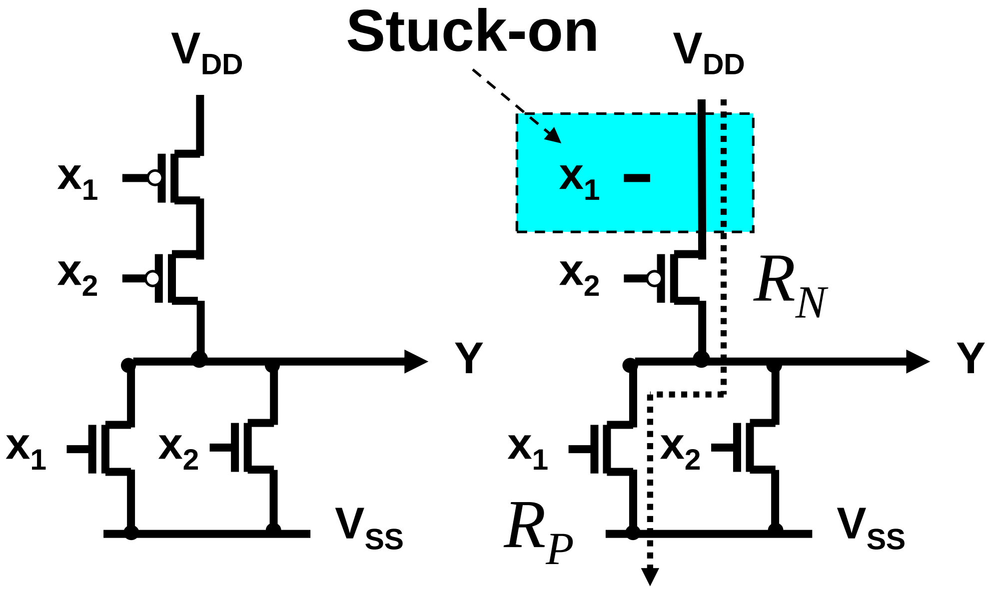
Macro level test generation:

$$y = x_6 x_{7,3} \vee (\overline{x_1} \vee x_2 x_{7,1})(\overline{x_5} \vee \overline{x_{7,2}})$$

$$\frac{\partial y}{\partial x_{7,1}} = (\overline{x_6} \vee \overline{x_{7,3}})(\overline{x_5} \vee \overline{x_{7,2}}) x_1 x_2 = x_1 x_2 \overline{x_7} = 1$$

Transistor Level Stuck-on Faults

NOR gate



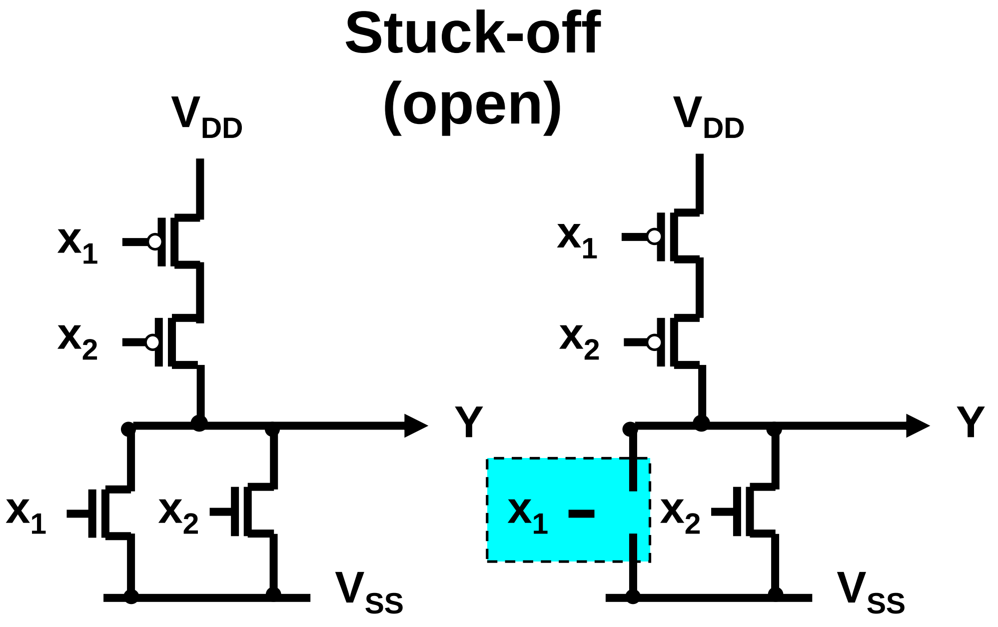
Conducting path for “10”

x_1	x_2	y	y^d
0	0	1	1
0	1	0	0
1	0	0	V_Y/I_{DDQ}
1	1	0	0

$$V_Y = \frac{V_{DD} R_P}{(R_P + R_N)}$$

Transistor Level Stuck-off Faults

NOR gate

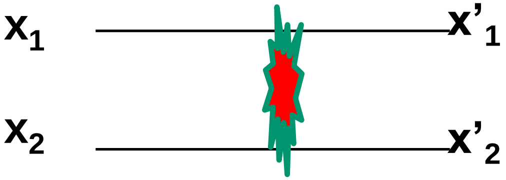


No conducting path from V_{DD} to V_{SS} for “10”

x_1	x_2	y	y^d
0	0	1	1
0	1	0	0
1	0	0	Y’
1	1	0	0

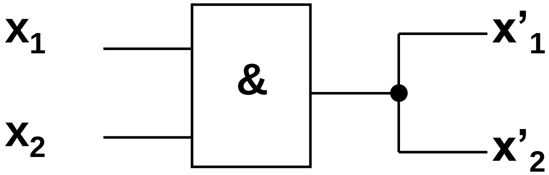
Test sequence is needed: 00,10

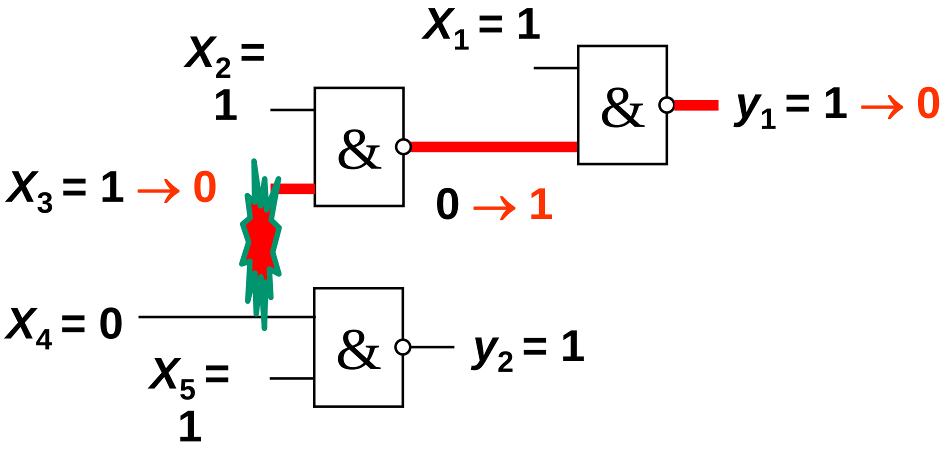
Testing of Bridging Fault Models



Wired AND model

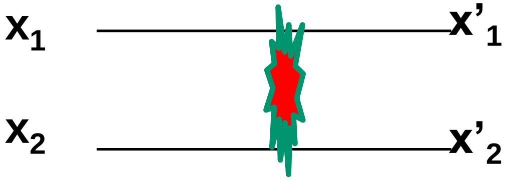
W-AND:



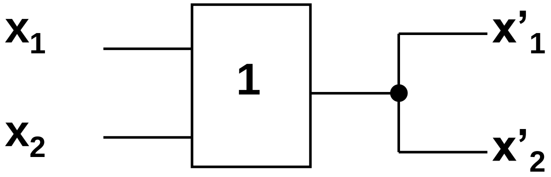


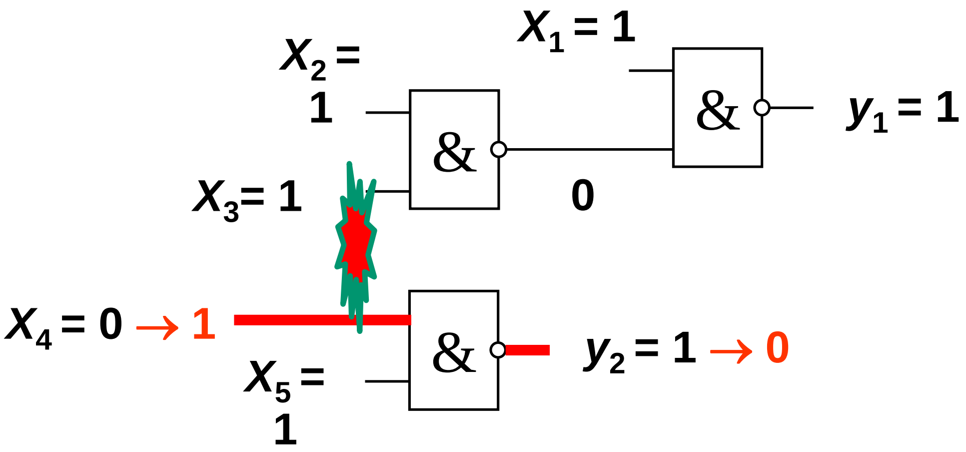
Testing of Bridging Fault Models

Wired OR model

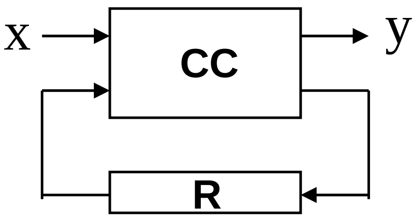


W-OR:





Complexity Problem: Sequential Circuits

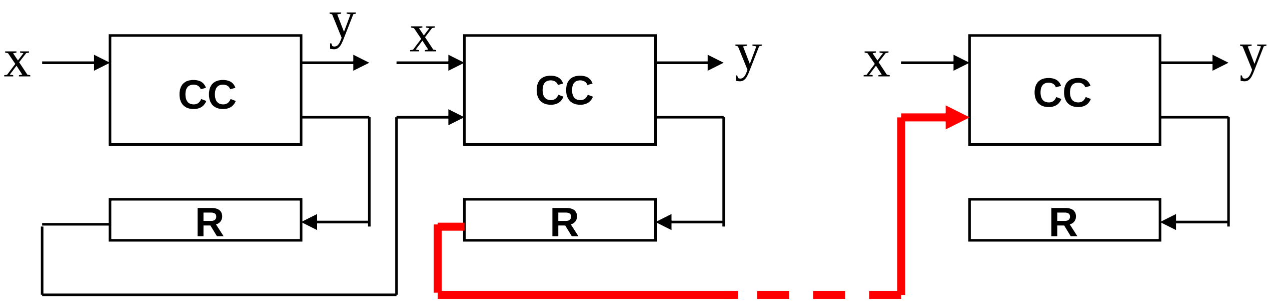


Fault sensitization: Test pattern consists of an input pattern and a state

Fault propagation: To propagate a fault to the output, an input pattern and a state is needed

Line justification: To reach the needed state, an input sequence is needed

Time frame model:

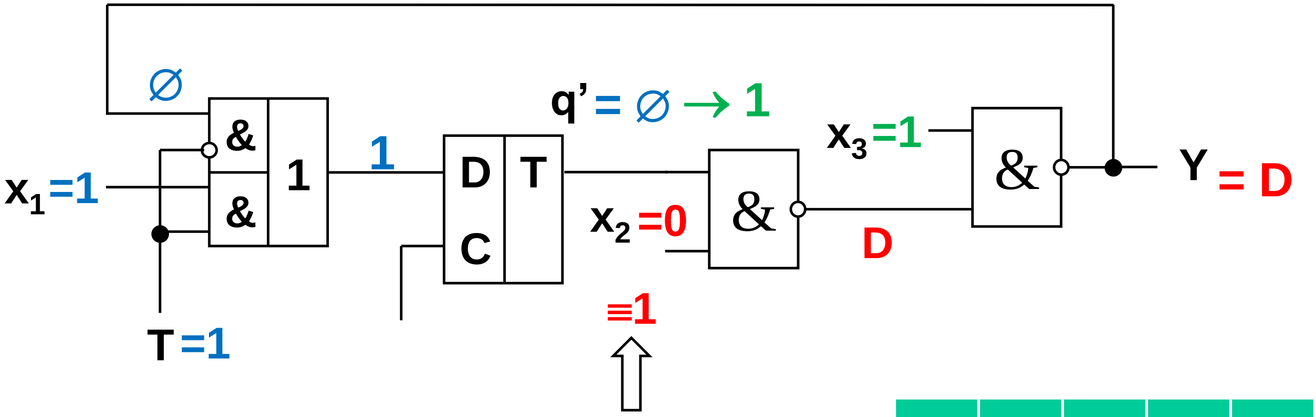


T - 2 **T - 1** **Current time frame T**

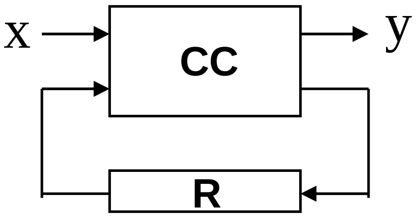
N^3 **N^2** **Search space N**

Complexity Problem: Sequential Circuits

Test generation for a fault in a sequential circuit:



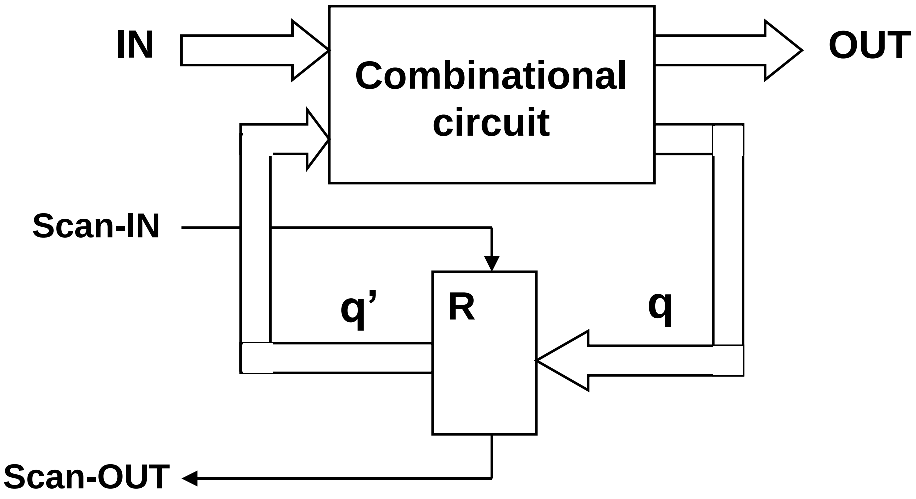
Fault to be tested

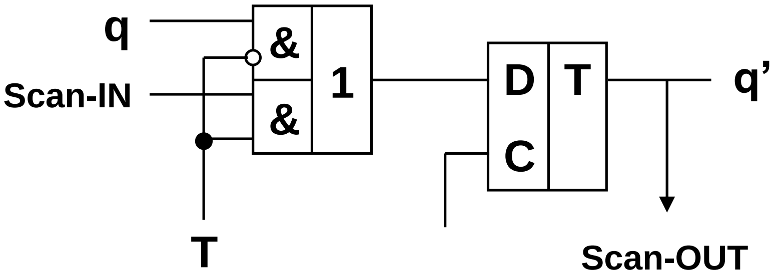


	T_1	q_1	x_1	x_2	x_3	Y
t	1	∅	1	D	∅	∅
t+1		1		D	1	D

Not always it is known how many clock cycles is needed
for propagation the faults through the space and time

Converting Sequentiality to Combinatorics





Scan-Path Design

The complexity of testing is a function of the number of feedback loops and their length

The longer a feedback loop, the more clock cycles are needed to initialize and sensitize patterns

Scan-register is a aregister with both shift and parallel-load capability

T = 0 - normal working mode T = 1 - scan mode

Normal mode : **flip-flops are connected to the combinational circuit**

Test mode: **flip-flops are disconnected from the combinational circuit and connected to each other to form a shift register**

Süsteemide diagnostika

4. Testide süntees digitaalsüsteemidele

4.1. Deterministlik testide süntees kombinatsioonskeemidele

4.2. Testide genereerimine otsustusdiagrammide abil

4.3. Triviaalsete (pseudotäielike) testide süntees

4.4. Testide süntees kordsetele riketele (üldjuht)

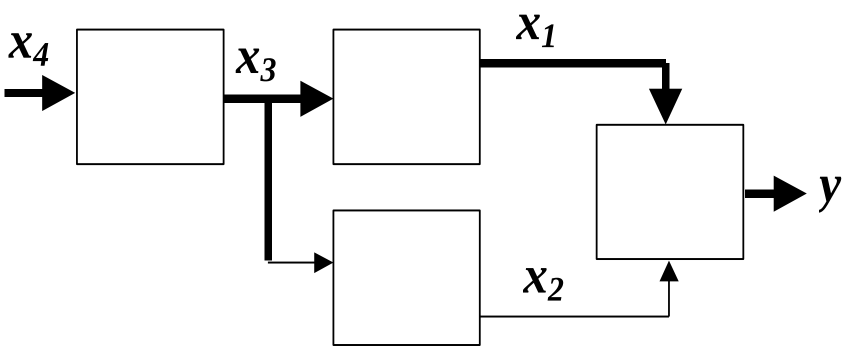
4.5. Testide süntees digitaalsüsteemidele kõrgtasandil

Derivatives for complex functions

Boolean derivative for a complex function:

$$\frac{\partial F_k(F_j(X), X)}{\partial x_i} = \frac{\partial F_k}{\partial F_j} \frac{\partial F_j}{\partial x_i}$$

Example:



$$\frac{\partial y}{\partial x_4} = \frac{\partial y}{\partial x_1} \frac{\partial x_1}{\partial x_3} \frac{\partial x_3}{\partial x_4}$$

Additional condition:

$$\frac{\partial x_2}{\partial x_3} = 0$$

Test Generation with BD and BDD

BD:

$$y = x_1x_2 \vee x_3(\overline{x_2}x_4 \vee \overline{x_1}(x_4 \vee (x_5 \vee \overline{x_2}x_6)) \vee x_1\overline{x_3}$$

$$\frac{\partial y}{\partial x_5} = \overline{(x_1x_2 \vee x_1\overline{\overline{x_3}})}\,x_3\,\overline{(\overline{\overline{x_2}}x_4)}\,\overline{x_1}\,\overline{x_4}\,\overline{(\overline{\overline{x_2}}x_6)}\,\frac{\partial x_5}{\partial x_5} =$$

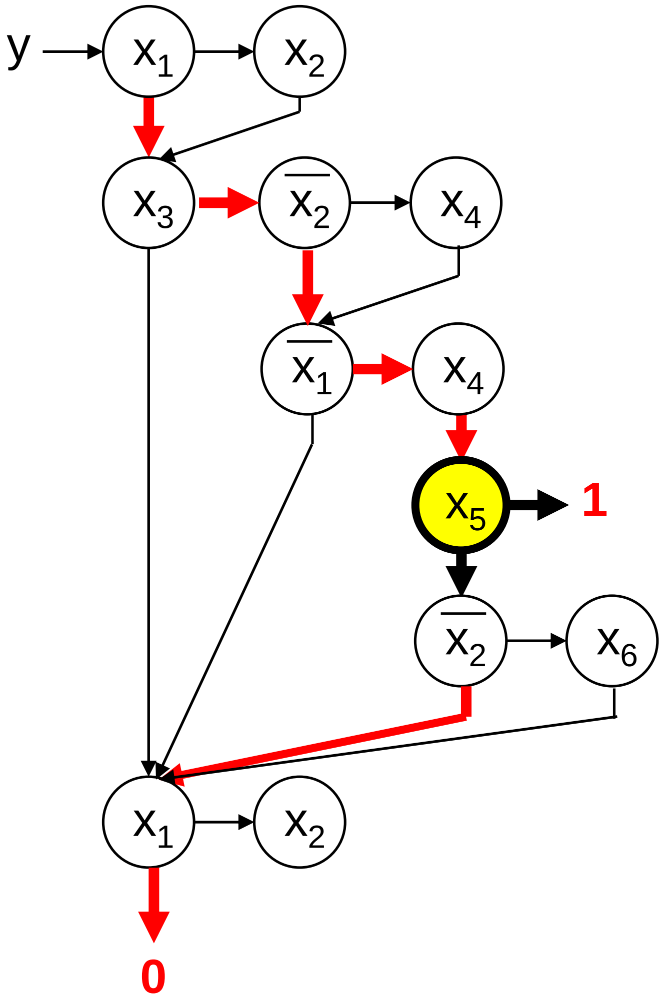
$$= (\overline{x_1} \vee \overline{x_2})(\overline{x_1} \vee x_3)x_3(x_2 \vee \overline{x_4})\overline{x_1}\,\overline{x_4}(x_2 \vee \overline{x_6}) =$$

$$= \overline{x_1}\,\overline{x_4}x_3x_2 \vee \ldots = 1$$

Test pattern:

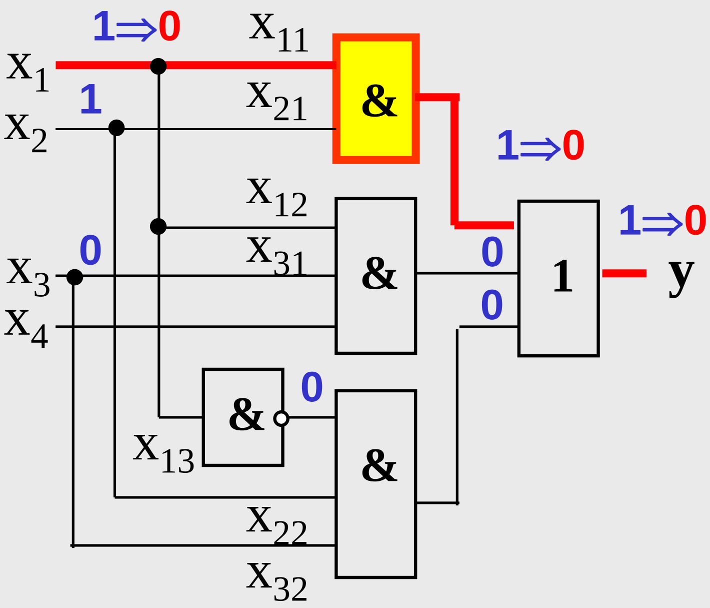
x_1	x_2	x_3	x_4	x_5	x_6	y
0	1	-	0	D	-	D

BDD:

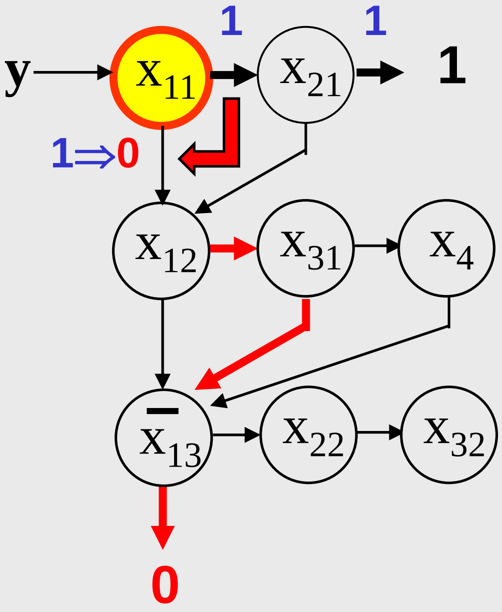


BDDs and Test Generation

Test generation for: $x_{11} \equiv 0$



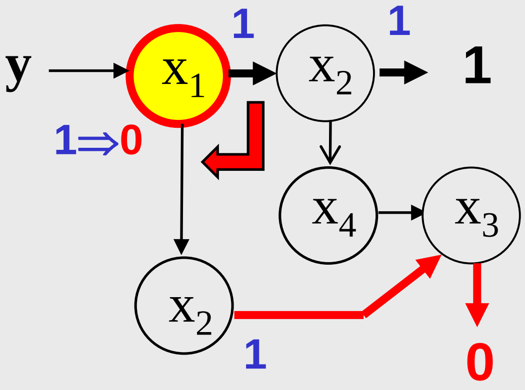
Structural BDD:



Test generation for:

$x_1 \equiv 0$

Functional BDD:



Test pattern:

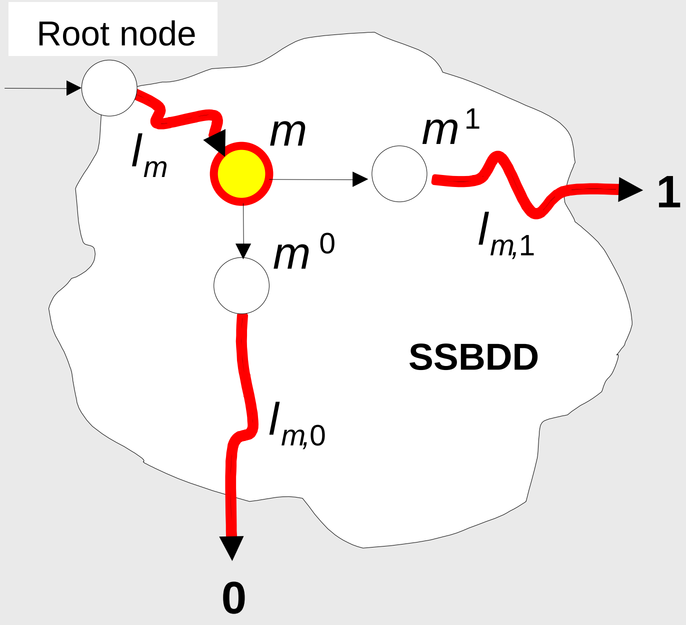
x_1	x_2	x_3	x_4	y
1	1	0	-	1 ⇒0

ALGORITHM:

Begin TG with Functional BDD
Simulate the test on Structural BDD
Update the test on Structural BDD

Topological Idea of Test Generation

BDD (SSBDD) for modeling
a function Y = F(X)



The node **m** is to be tested

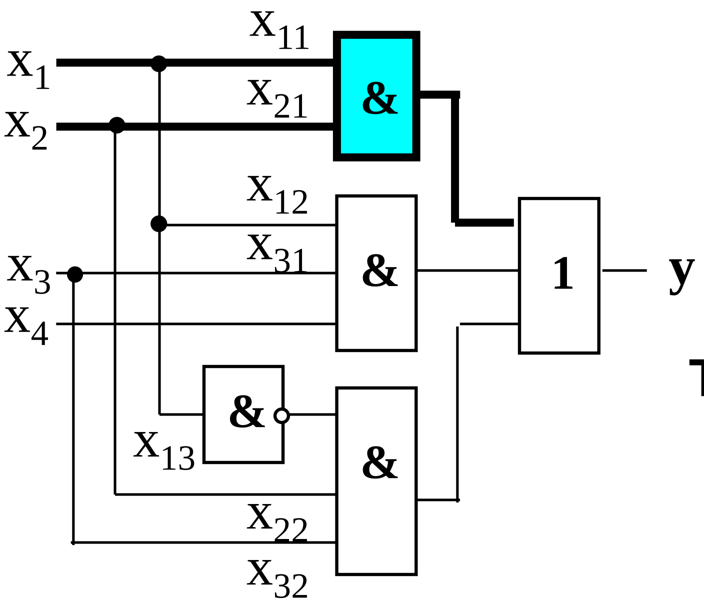
Three paths should be activated:

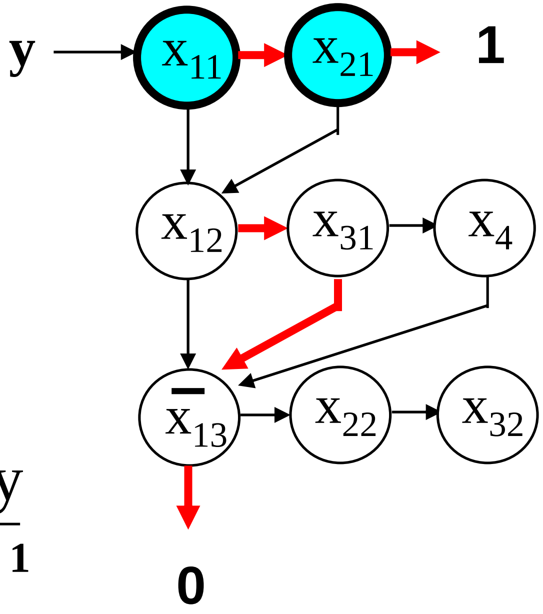
(1) a path **l_m** from **root** to **m**
(2) a path **$l_{m,1}$** from **m^1** to **terminal 1**
(3) a path **$l_{m,0}$** from **m^0** to **terminal 0**

Then
if variable(**m**) = 1 then **Y = 1**
else
if variable(**m**) = 0 then **Y = 0**

Example: Test Generation with SSBDDs

Testing Stuck-at-0 faults on paths:





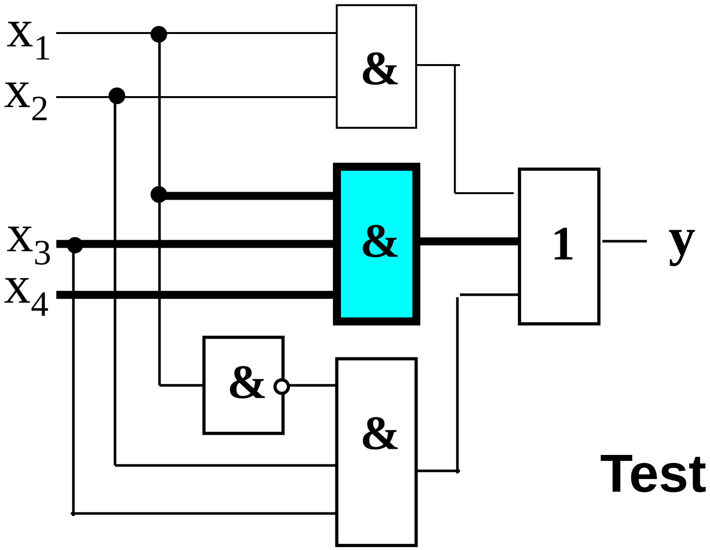
Test pattern:

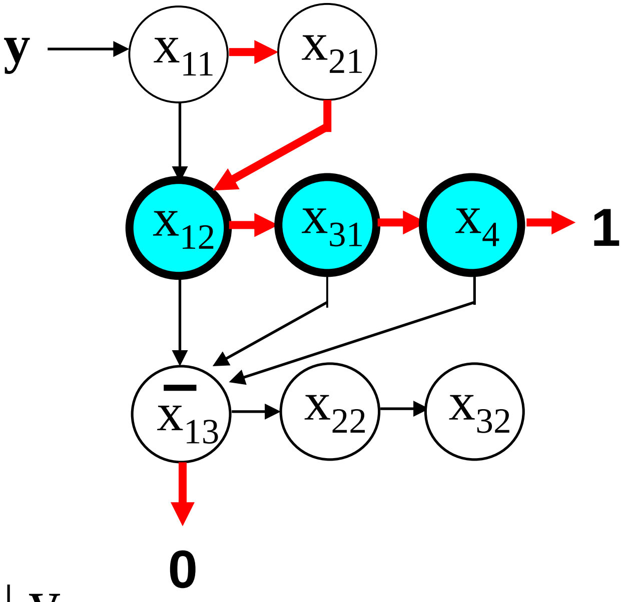
x_1	x_2	x_3	x_4	y
1	1	0	-	1

Tested faults: $x_{11} \equiv 0$, $x_{21} \equiv 0$

Example: Test Generation with SSBDDs

Testing Stuck-at-0 faults on paths:





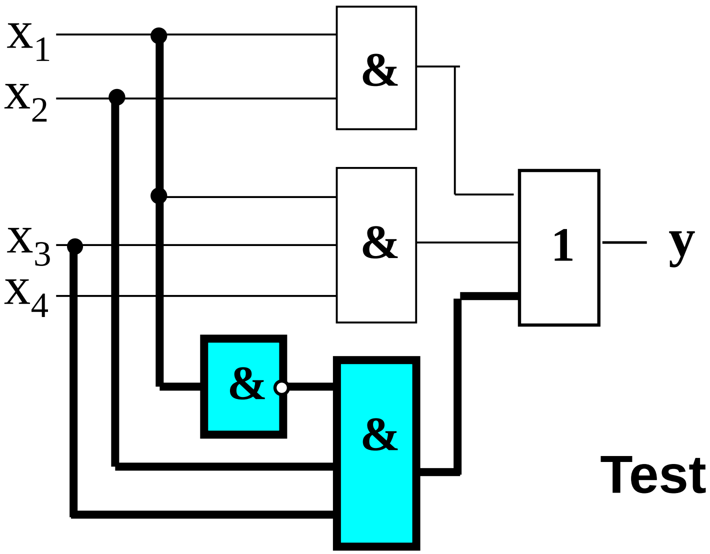
Test pattern:

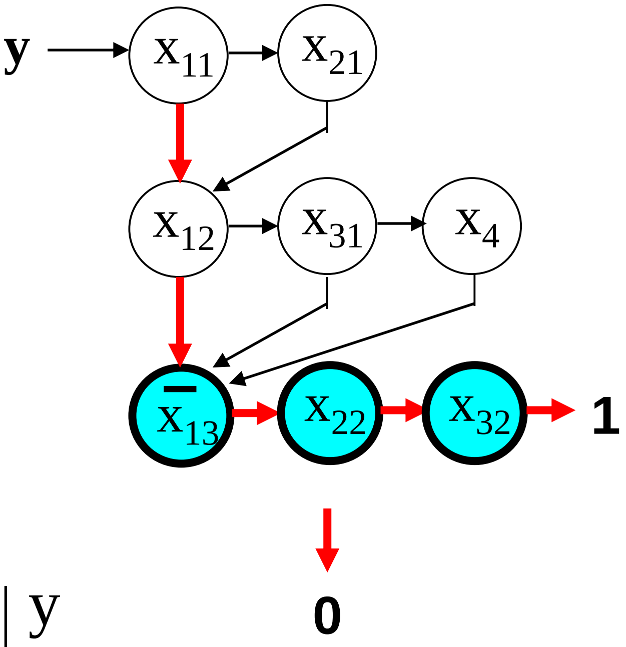
x_1	x_2	x_3	x_4	y
1	0	1	1	1

Tested faults: $x_{12}\equiv0$, $x_{31}\equiv0$, $x_4\equiv0$

Example: Test Generation with SSBDDs

Testing Stuck-at-0 faults on paths:





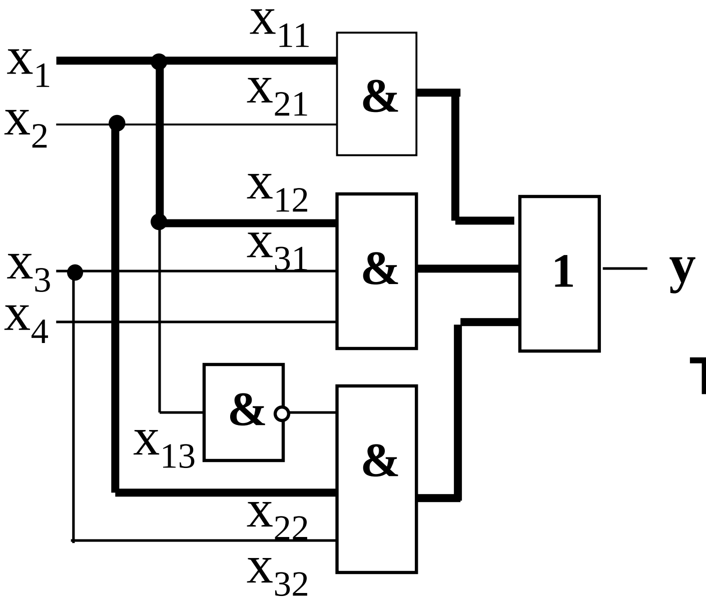
Test pattern:

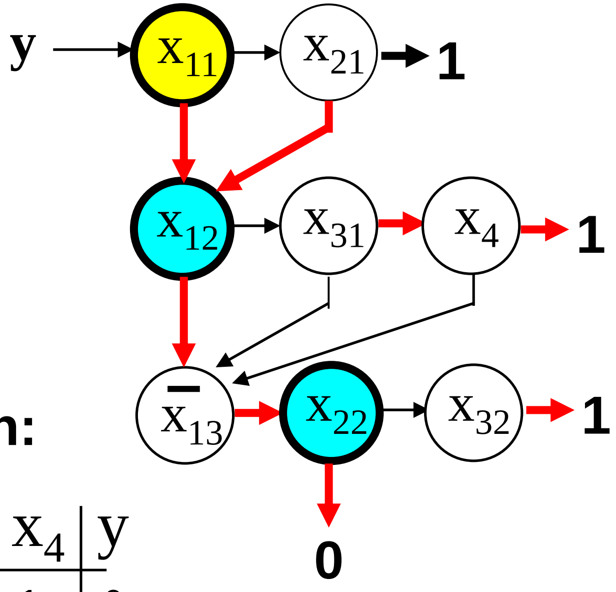
x_1	x_2	x_3	x_4	y
0	1	1	0	1

Tested faults: $x_{13}\equiv1$, $x_{22}\equiv0$, $x_{32}\equiv0$

Example: Test Generation with SSBDDs

Testing Stuck-at-1 faults on paths:





Test pattern:

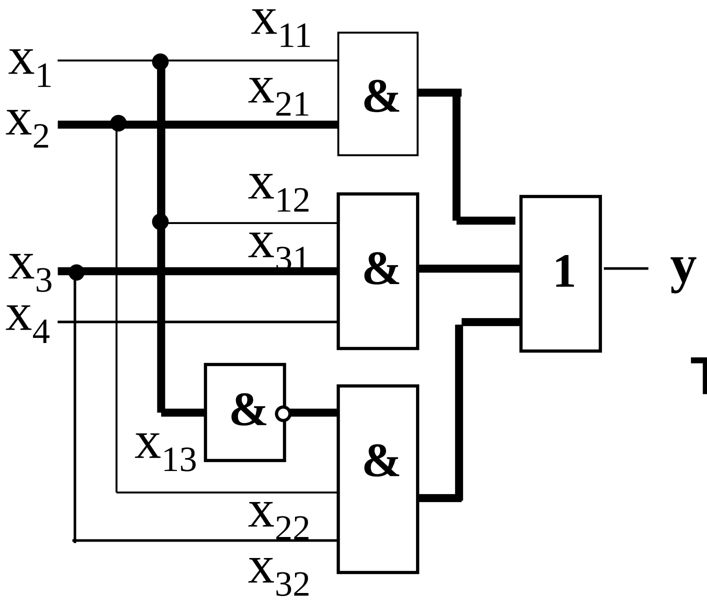
x_1	x_2	x_3	x_4	y
0	0	1	1	0

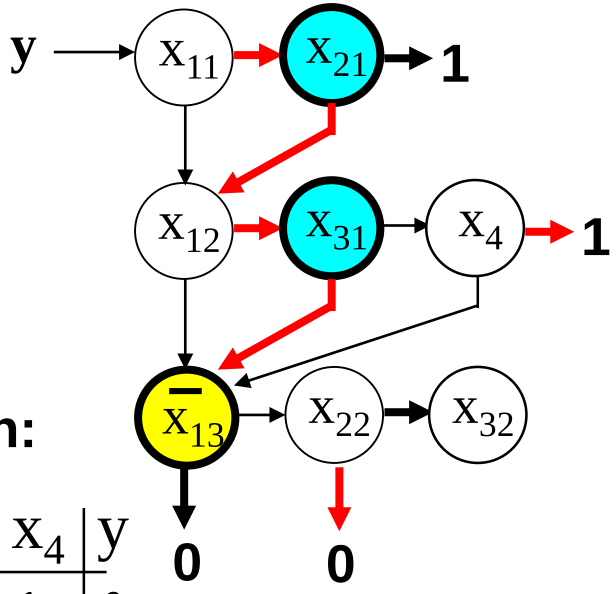
Tested faults: $x_{12} \equiv 1$, $x_{22} \equiv 1$

Not tested: $x_{11} \equiv 1$

Example: Test Generation with SSBDDs

Testing Stuck-at-1 faults on paths:





Test pattern:

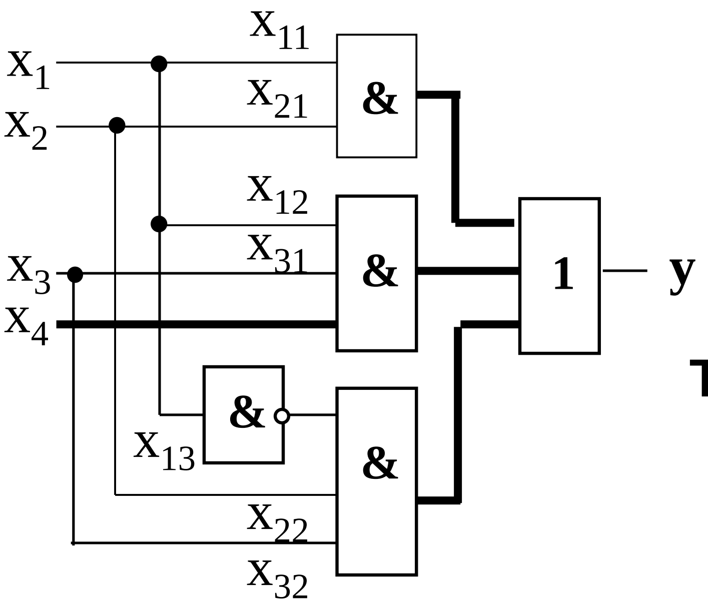
x_1	x_2	x_3	x_4	y
1	0	0	1	0

Tested faults: $x_{21} \equiv 1$, $x_{31} \equiv 1$

Not tested: $x_{13} \equiv 0$

Example: Test Generation with SSBDDs

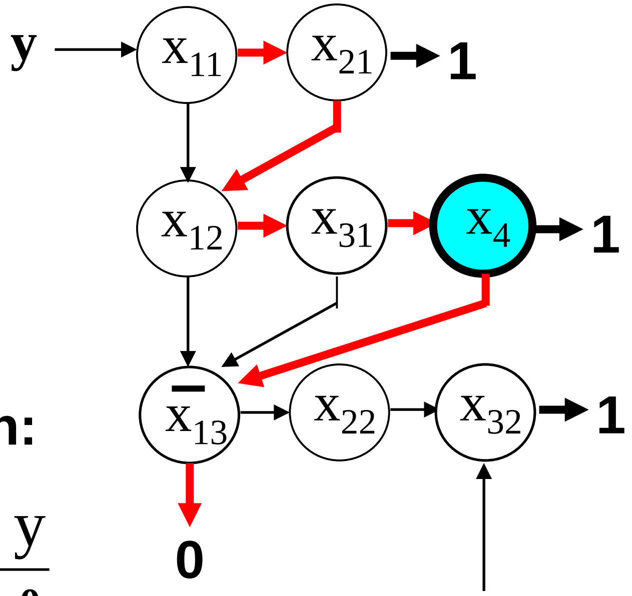
Testing Stuck-at-1 faults on paths:



Test pattern:

x_1	x_2	x_3	x_4	y
1	0	1	0	0

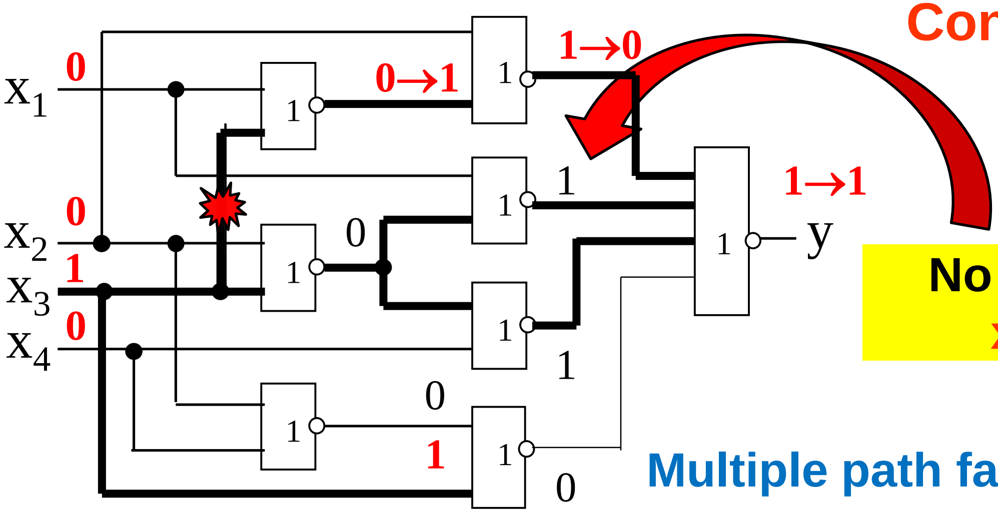
Tested fault: $x_4 \equiv 1$



Not yet tested fault: $x_{32} \equiv 1$

Problems with Test Generation

Test generation for faults at branches



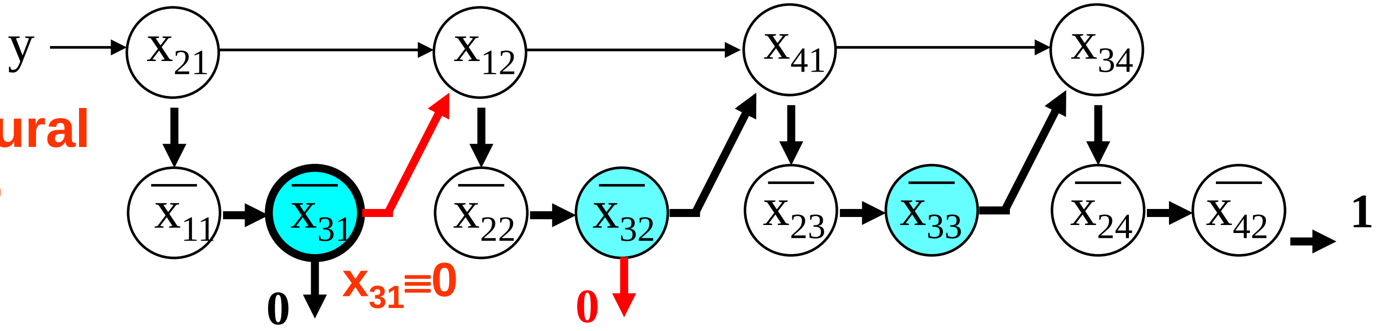
Conflict

No test for $x_{31} \equiv 0$

Multiple path fault propagation by DDs:

Signal paths are directly visible in DDs as nodes

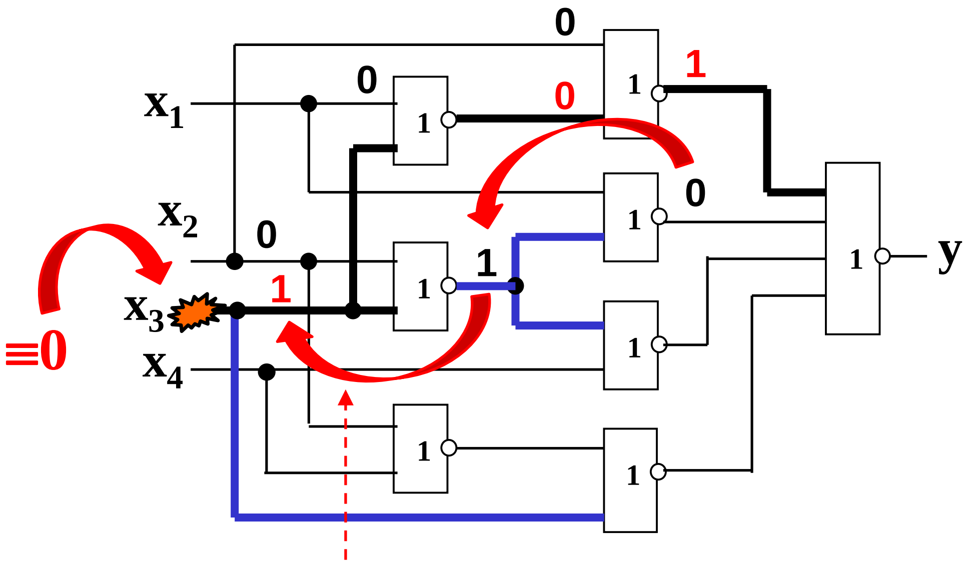
Structural DD



Structural Test Generation: Problems

Again a conflict during test generation:

Single path activation is not possible



Conflict

But there is another possibility

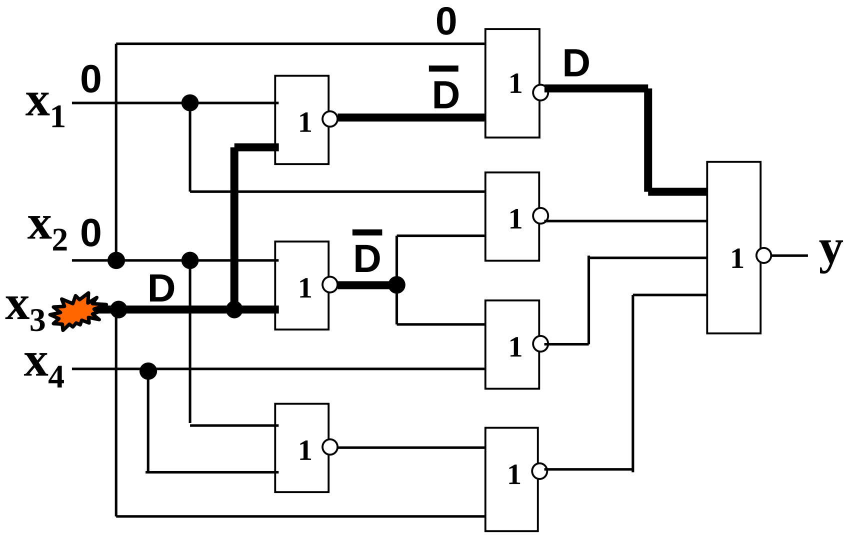
Test Generation: Two Approaches

Symbolic signal propagation

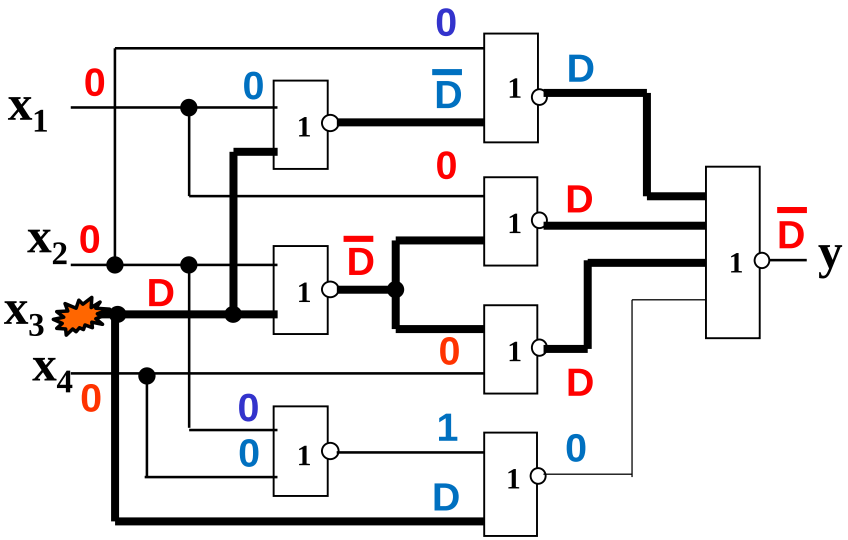
D = 0 – no fault

D = 1 – there is a fault

Single path fault propagation:



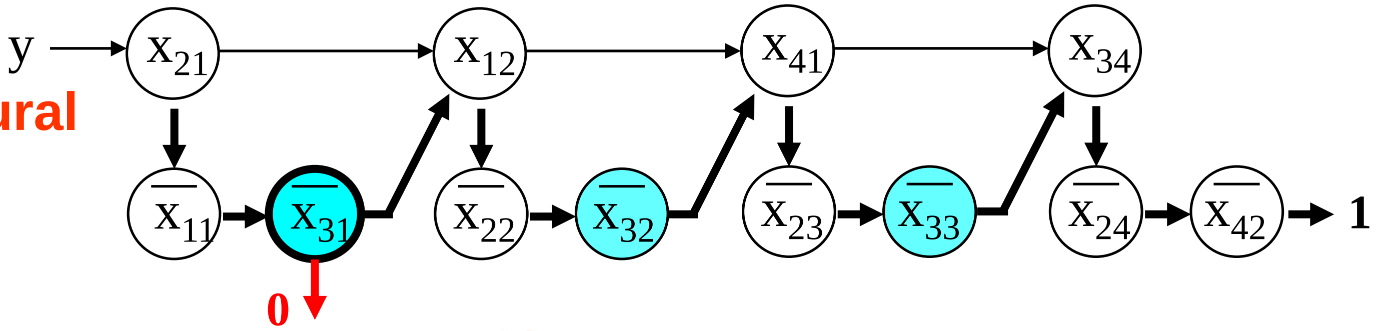
Multiple path fault propagation:



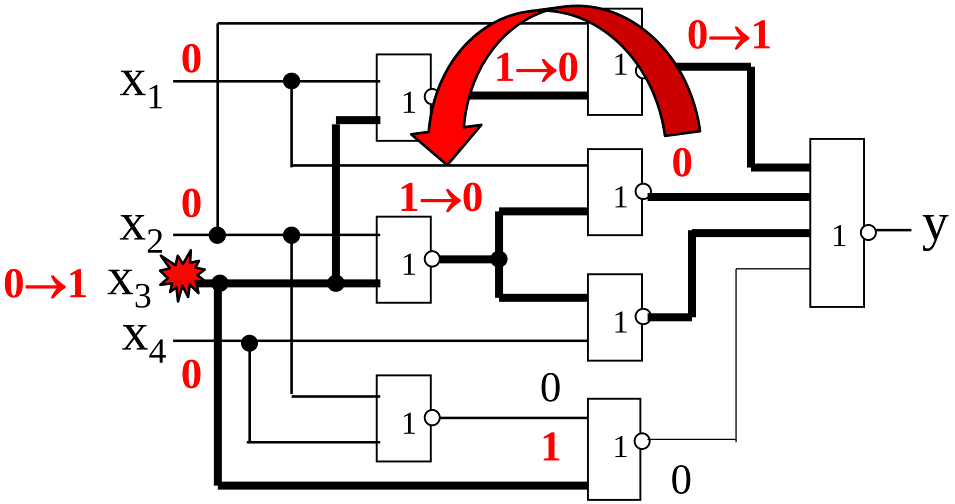
Impact of Redundancies

Multiple path fault propagation by DDs:

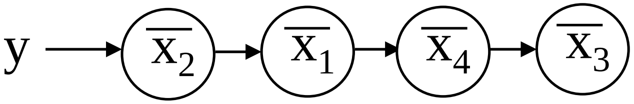
Structural DD



Conflict



Functional DD

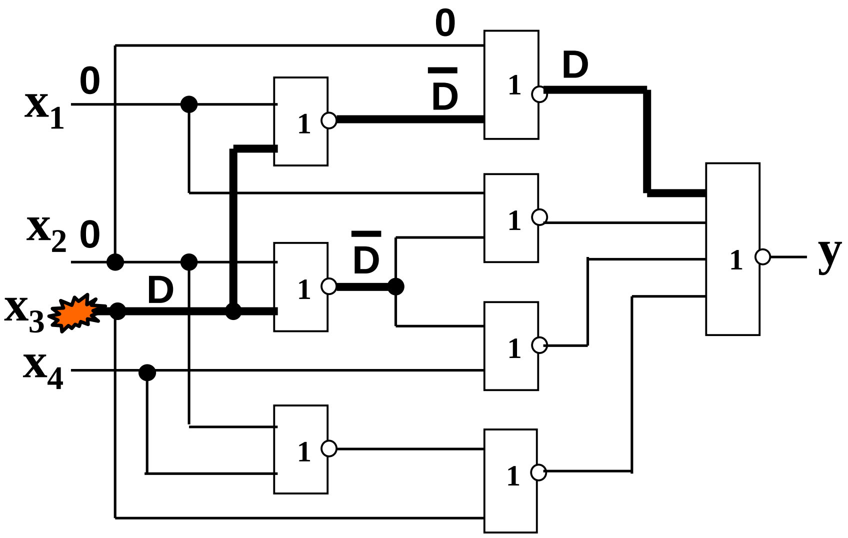


The original circuit of 8 gates has a lot of redundancies, and after optimization has collapsed to a **single AND gate**

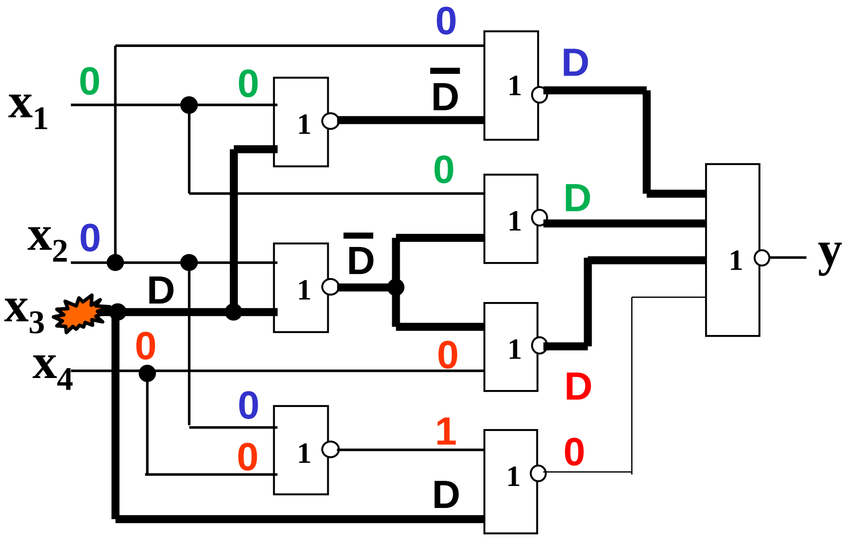
No fan-out any more
The test patterns are needed only for inputs

When Redundancies are Removed

Multiple path fault propagation:

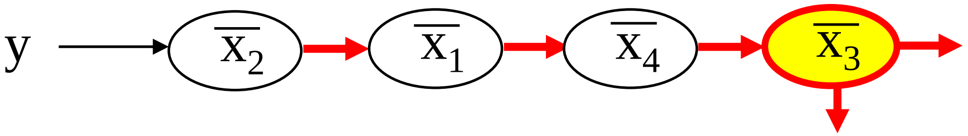


Single path activation is not possible



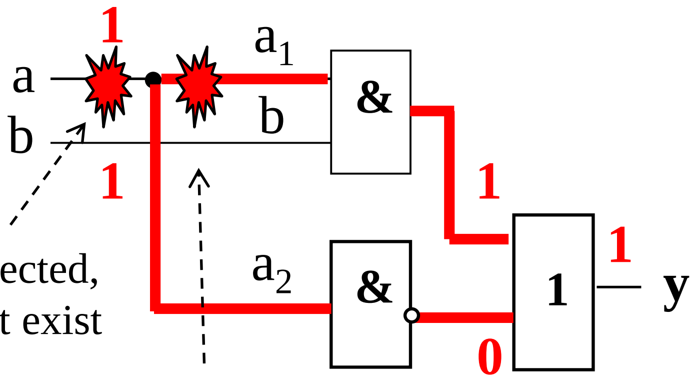
Three paths simultaneously activated

Functional DD



The original circuit has collapsed to a **single AND gate**

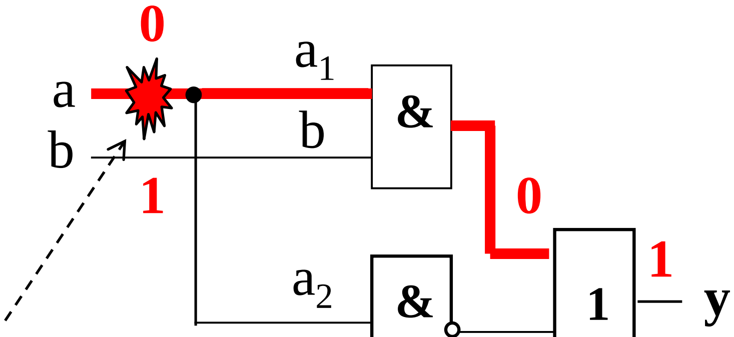
Testing of Redundant Faults



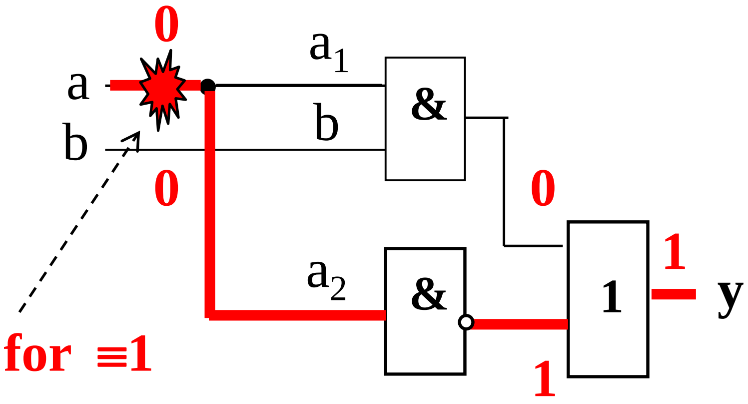
≡0 not detected,
but the test exist

No test for both ≡ 1

$$y = ab \vee \bar{a} = b \vee \bar{a}$$

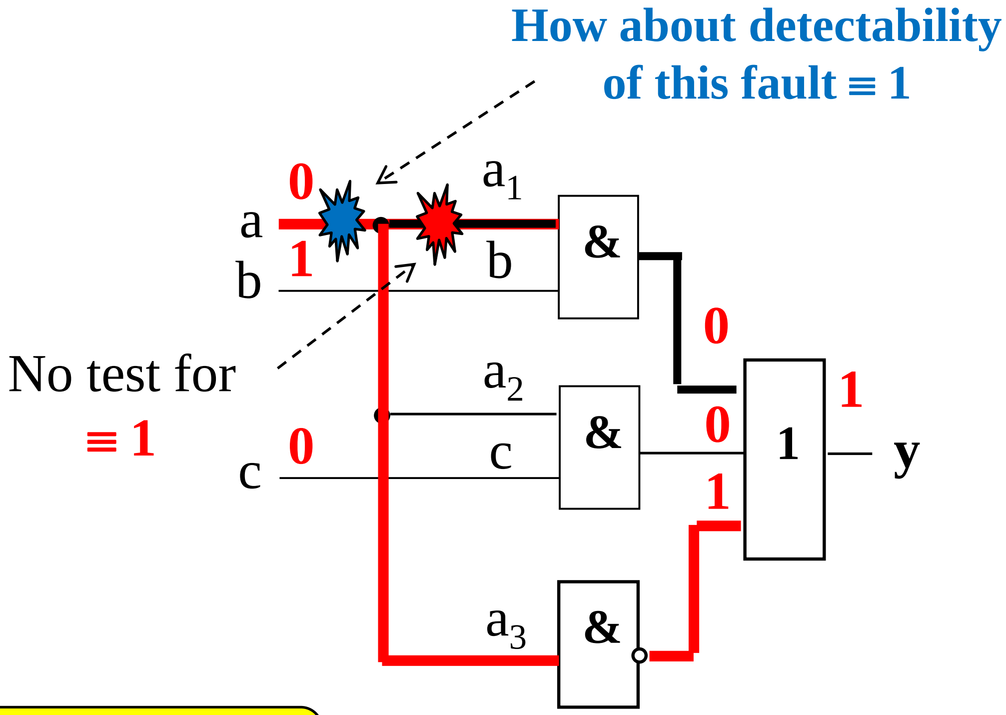


No test for **≡ 1**



Test for ≡1

Testing of Redundant Faults



General test strategy:

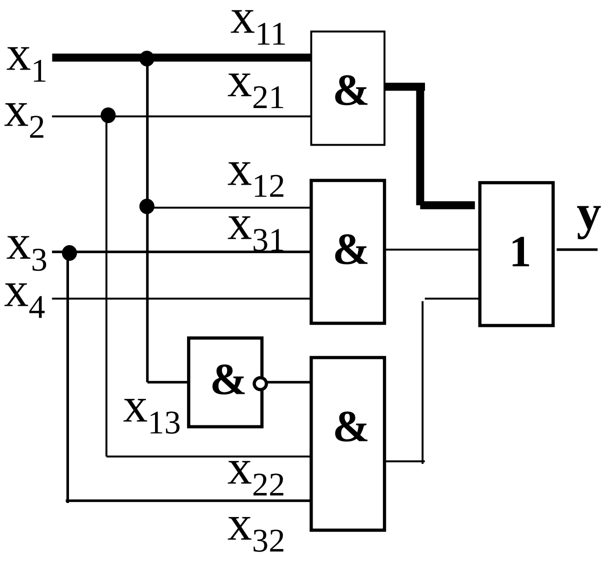
1. Single path activation is preferred
2. If not successful, try another single path
3. If still not successful, try multiple path activation
4. If still not successful, the fault is redundant

Faults $a_1 \equiv 1$ and $a_2 \equiv 1$ are redundant

$$y = ab \vee ac \vee \overline{a} = \overline{a} \vee b \vee c$$

Fast and Simple Test Generation

Test generation by using disjunctive normal forms



$$y = x_1x_2 \vee x_1x_3x_4 \vee \overline{x}_1x_2x_3$$

x_1	x_2		x_1	x_3	x_4		$\overline{x}_1$	x_2	x_3		y		x_1	x_2	x_3	x_4
0	1		0	0			1	1	0		0		0	1	0	
1	0		1	0	1		0	0	1		0		1	0	0	1
0	0		0	1	1		1	0	1		0		0	0	1	1
1	0		1	1	0		0	0	1		0		1	0	1	0
1	1						0	1	1		1		No test			
1	1		1	0			0	1	0		1		1	1	0	
1	0		1	1	1		0	0	1		1		1	0	1	1
0			0				1	1	1		1		0	1	1	

Süsteemide diagnostika

4. Testide süntees digitaalsüsteemidele

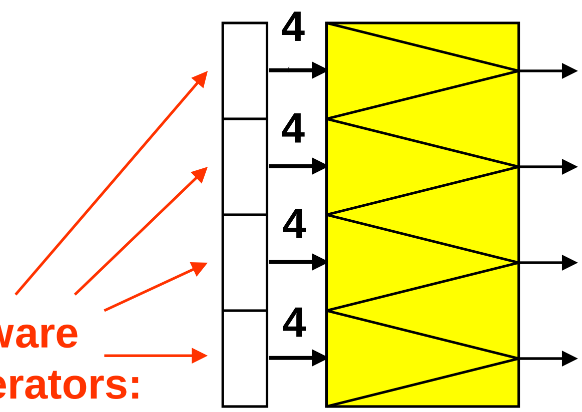
- 4.1. Deterministlik testide süntees kombinatsioonskeemidele
- 4.2. Testide genereerimine otsustusdiagrammide abil
- 4.3. Triviaalsete (pseudotäielike) testide süntees
- 4.4. Testide süntees kordsetele riketele (üldjuht)
- 4.5. Testide süntees digitaalsüsteemidele kõrgtasandil

BIST: Pseudoexhaustive Testing

Pseudo-exhaustive test sets:

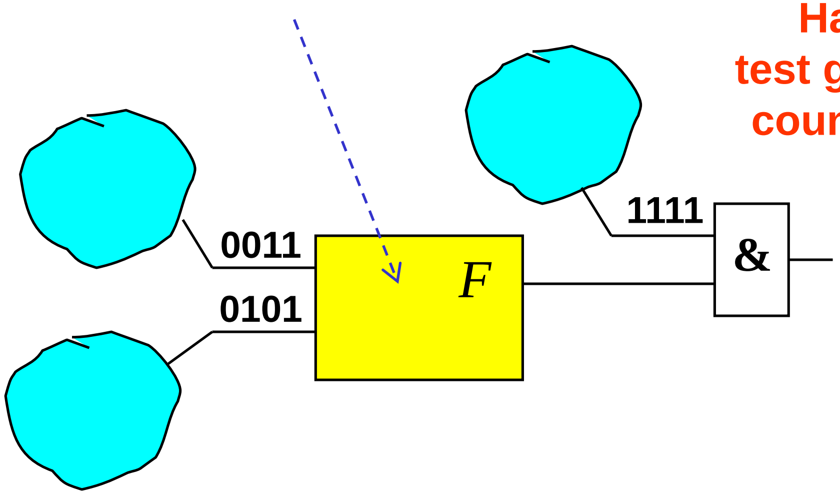
- Output function verification
  - maximal parallel testability
  - partial parallel testability
- Segment function verification

Output function verification



Hardware
test generators:
counter, LFSR

Segment function verification



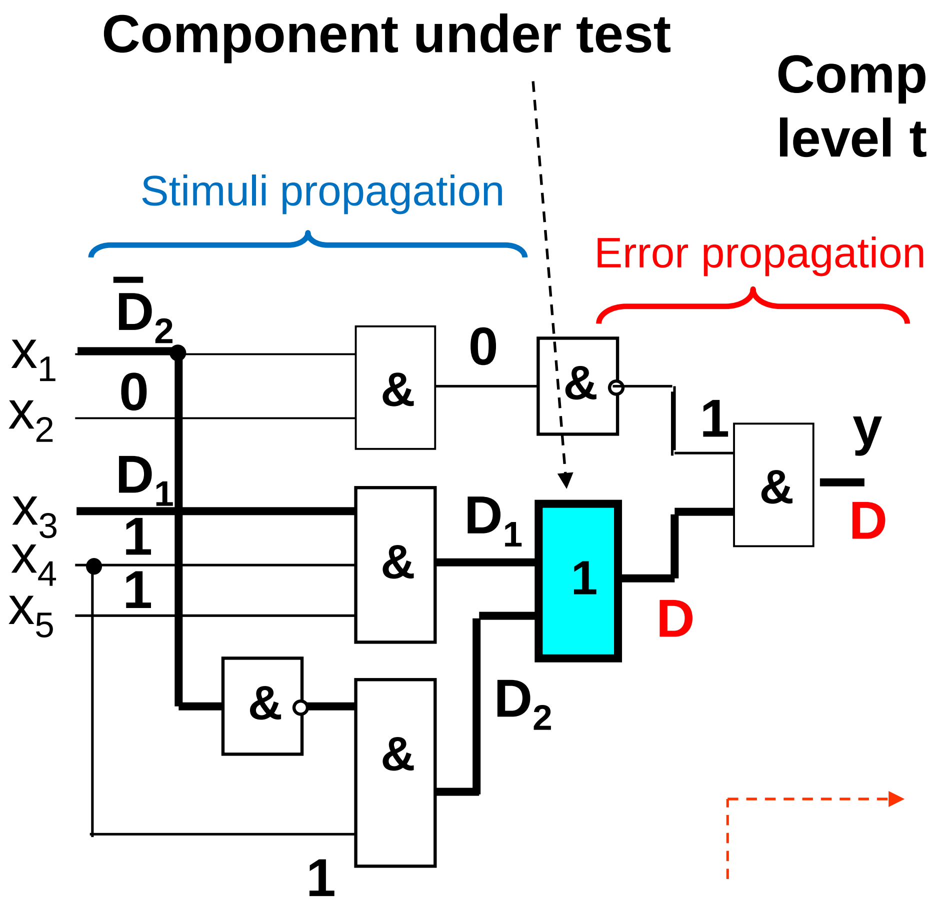
$2^{16} = 65536 >> 4x16 = 64 > 16$

Exhaustive test

Pseudo-exhaustive sequential

Pseudo-exhaustive parallel

Hierarchical Test Generation



Component level test:

D_1	D_2	D
0	0	0
0	1	1
1	0	1

Network level test:

x_1	x_2	x_3	x_4	x_5	y
$\overline{D}_2$	0	D_1	1	1	D

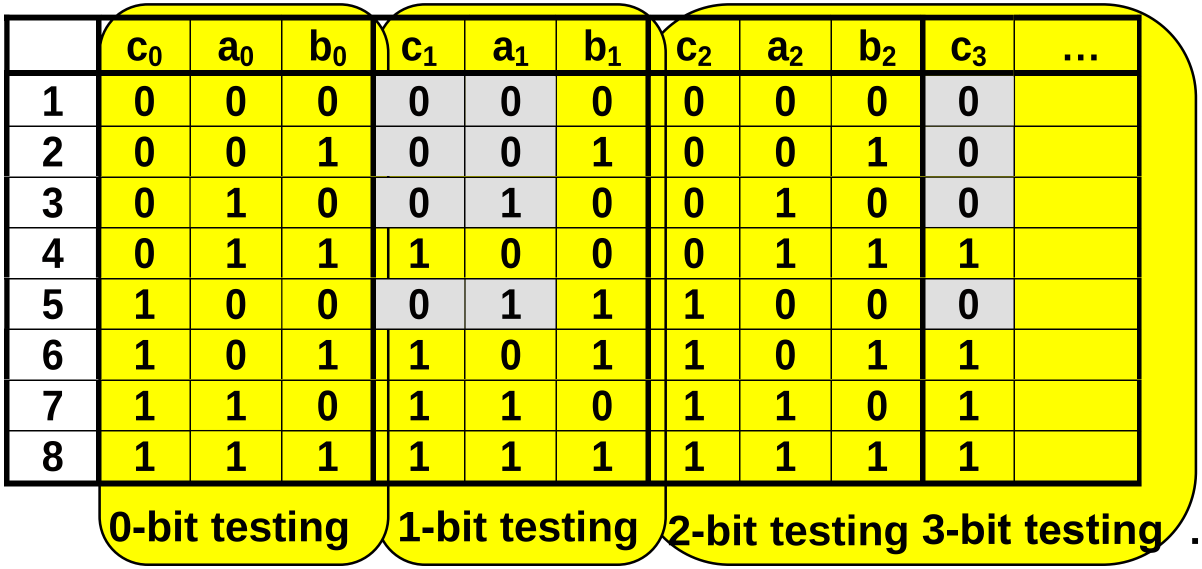
Symbolic test: contains 3 patterns

Testing ripple-carry adder

Output function verification (maximum parallelity)

Pseudo-Exhaustive test generation for n-bit adder:

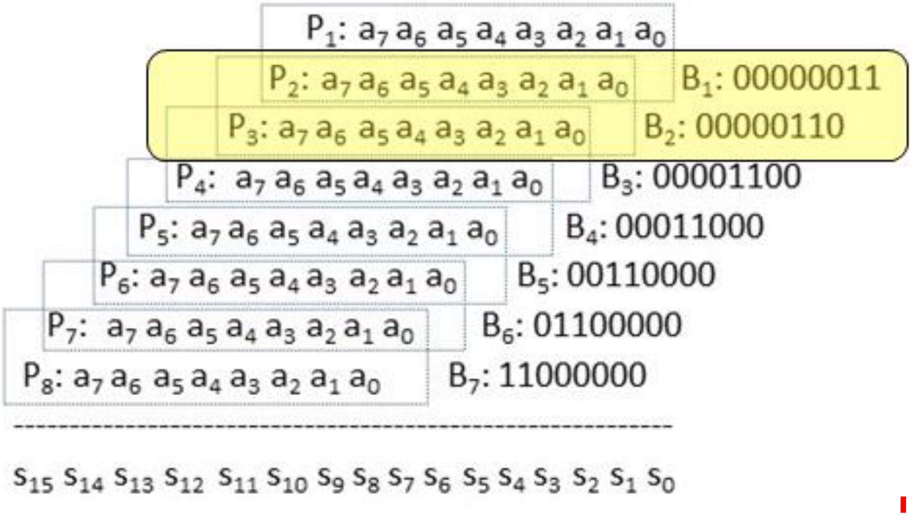
Good news:	*Bad news:*
Bit number n - arbitrary	**The method is correct**
Test length - always 8 (!)	**only for ripple-carry adder**



	c_0	a_0	b_0	c_1	a_1	b_1	c_2	a_2	b_2	c_3	…
1	0	0	0	0	0	0	0	0	0	0	
2	0	0	1	0	0	1	0	0	1	0	
3	0	1	0	0	1	0	0	1	0	0	
4	0	1	1	1	0	0	0	1	1	1	
5	1	0	0	0	1	1	1	0	0	0	
6	1	0	1	1	0	1	1	0	1	1	
7	1	1	0	1	1	0	1	1	0	1	
8	1	1	1	1	1	1	1	1	1	1	

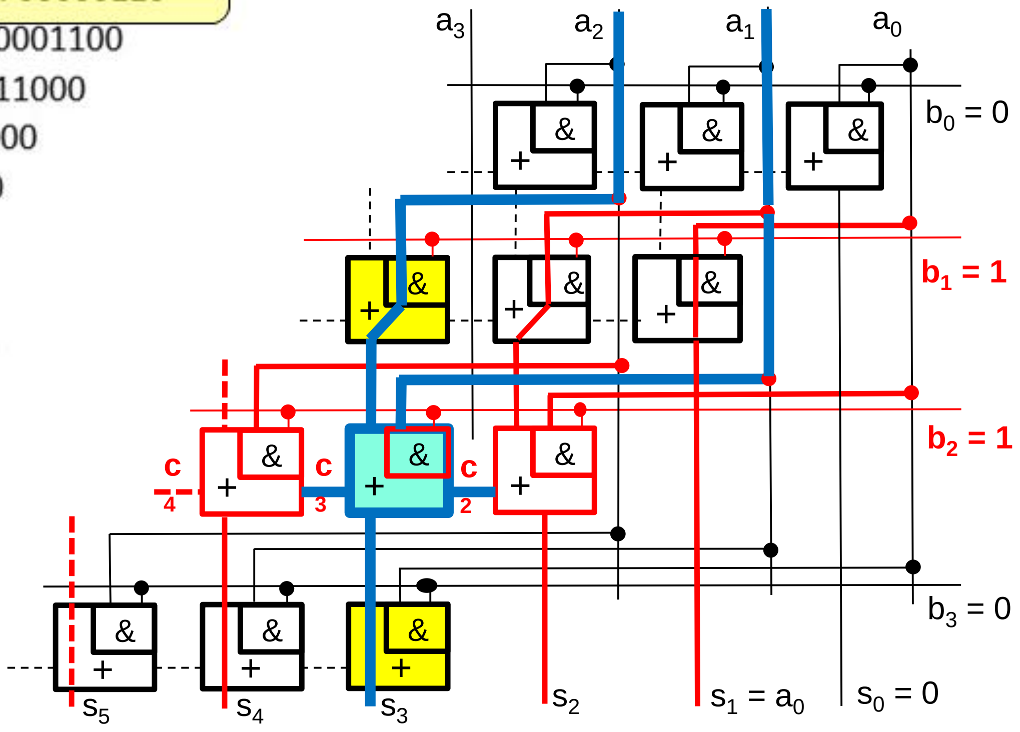
0-bit testing **1-bit testing** **2-bit testing** **3-bit testing** **… etc**

Pseudo-Exhaustive Test for Multiplier



Multiplication with traditional "*paper and pencil*" method

Multiplier array



Pseudo-Exhaustive Test for Multiplier

Replication of columns with pseudo-exhaustive patterns for

Adder

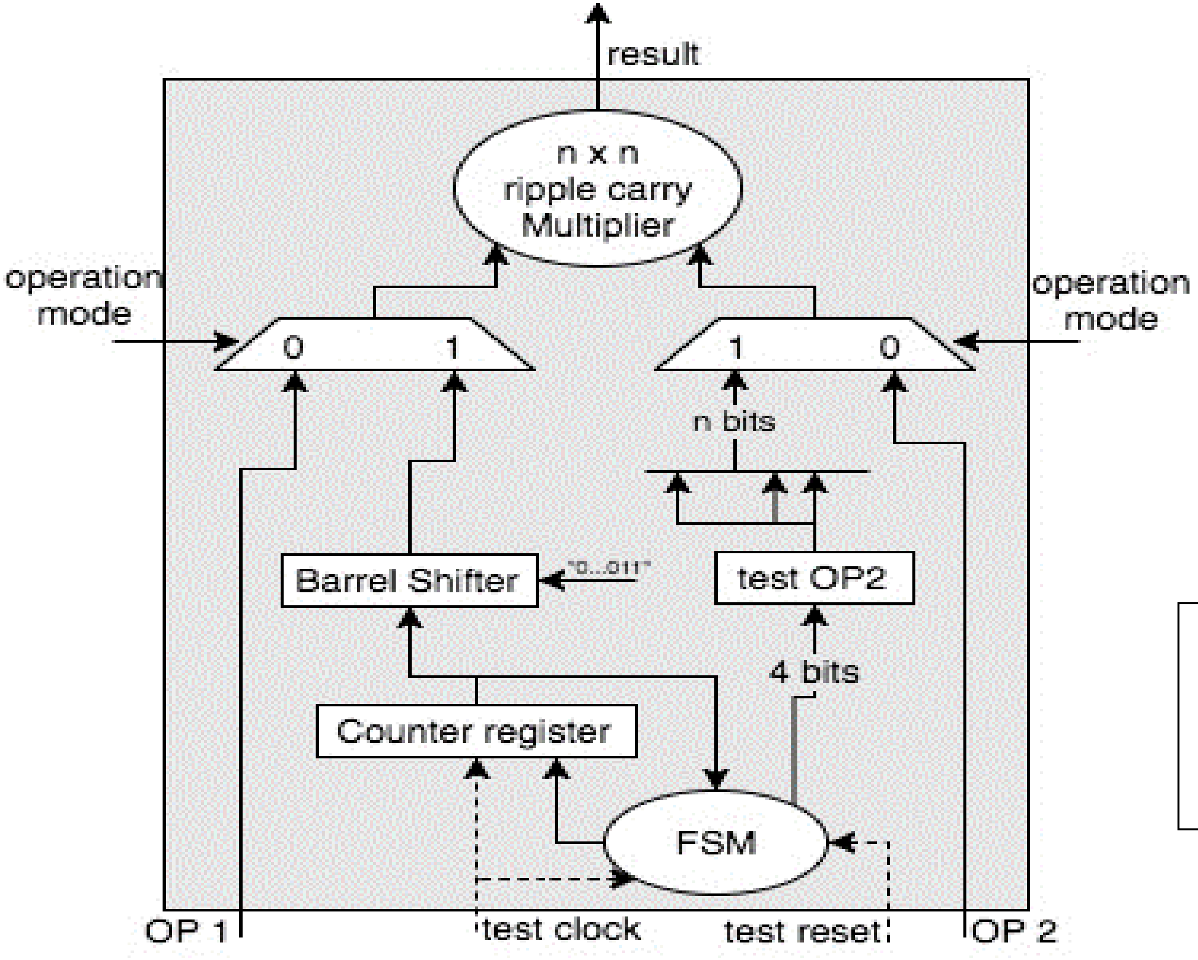
No	…	4-bit	3-bit	2-bit	1-bit	0-bit
		a_4 b_4 c_4	a_3 b_3 c_3	a_2 b_2 c_2	a_1 b_1 c_1	a_0 b_0
1	…	0 0 0	0 0 0	0 0 0	0 0 0	0 0
2	…	0 1 0	0 1 0	0 1 0	0 1 0	0 1
3	…	1 0 0	1 0 0	1 0 0	1 0 0	1 0
4	…	1 1 0	0 0 1	1 1 0	0 0 1	1 1
5	…	0 0 1	1 1 0	0 0 1	1 1 0	0 0
6	…	0 1 1	0 1 1	0 1 1	0 1 1	1 1
7	…	1 0 1	1 0 1	1 0 1	1 0 1	1 1
8	…	1 1 1	1 1 1	1 1 1	1 1 1	1 1

carry multiplier array

Multiplier

N	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit	0-bit
	$c_6a_7a_6$	$c_5a_6a_5$	$c_4a_5a_4$	$c_3a_4a_3$	$c_2a_3a_2$	$c_1a_2a_1$	a_1a_0
1	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0
2	0 1 0	0 0 1	0 1 0	0 0 1	0 1 0	0 0 1	1 0
3	0 0 1	0 1 0	0 0 1	0 1 0	0 0 1	0 1 0	0 1
4	1 0 1	0 1 1	0 1 0	1 0 0	1 0 1	0 1 1	1 0
5	1 1 0	1 0 1	1 1 0	1 0 1	1 1 0	1 0 1	1 1
6	1 0 1	1 1 1	1 1 1	1 1 0	1 0 1	1 1 1	1 1
7	0 1 1	0 1 0	1 0 0	1 0 1	0 1 1	0 1 0	0 0
8	1 0 0	1 0 1	0 1 1	0 1 0	1 0 0	1 0 1	1 1
9	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1
10	0 1 0	1 0 0	1 0 1	0 1 1	0 1 0	0 1 0	1 0
11	1 1 1	1 1 0	1 0 1	1 1 1	1 1 1	1 1 1	1 1

Exhaustively Self-Testing Multiplier



BIST

Built-in Self-Test

Multiplicand operands:

Shifted 11

000000**11**
00000**11**0

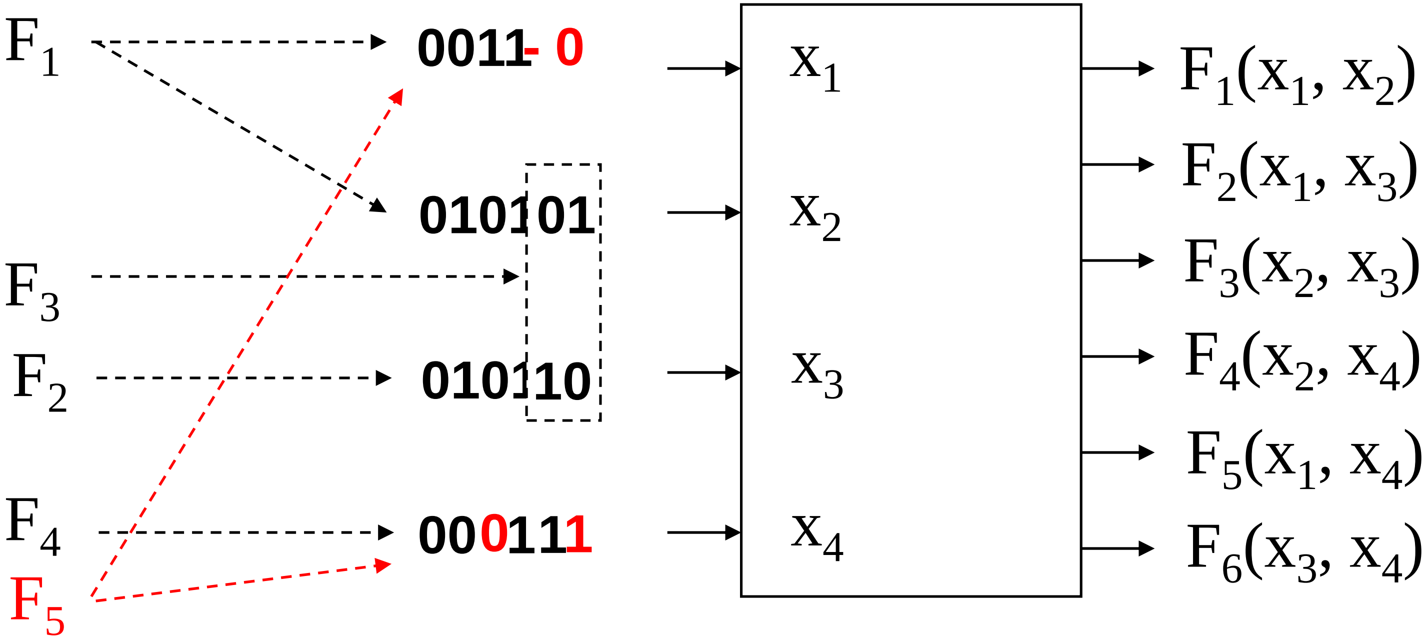
11000000

Multiplier operands:
Generated with FSM
and replicated
5-bit **11 patterns**

Test length: (n-1) × 11

Pseudoexhaustive Test Optimization

Output function verification (partial parallelity)



F_1 F_3 F_2 F_4 F_5

0011- 0

010101

010110

000111

x_1 x_2 x_3 x_4

$F_1(x_1, x_2)$

$F_2(x_1, x_3)$

$F_3(x_2, x_3)$

$F_4(x_2, x_4)$

$F_5(x_1, x_4)$

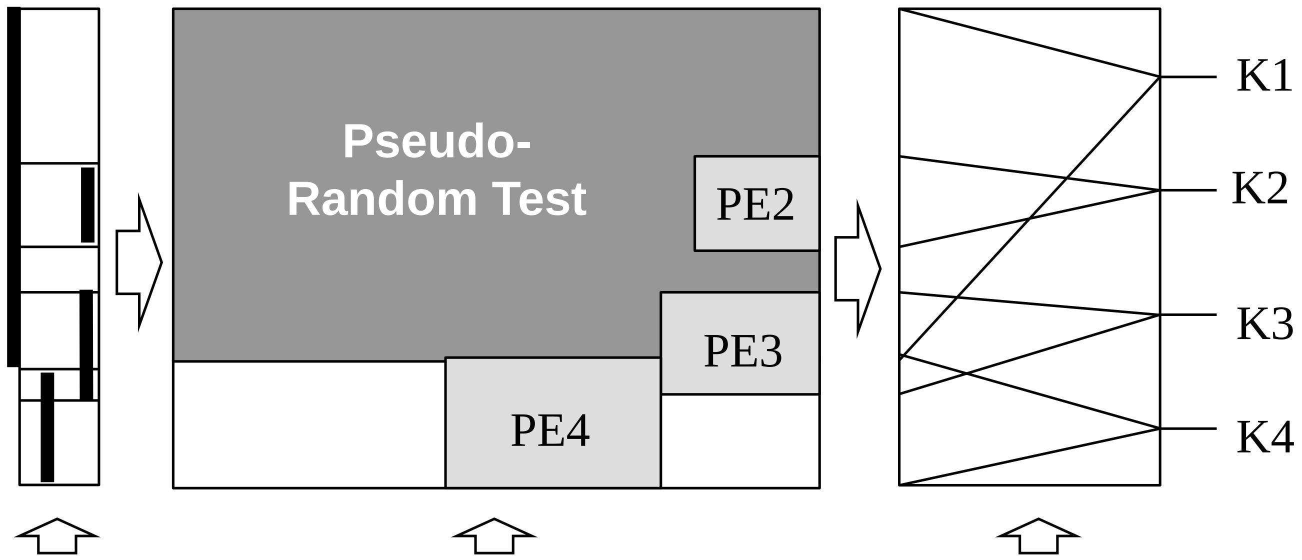
$F_6(x_3, x_4)$

Exhaustive testing - 16

Pseudo-exhaustive, full parallel – 4 (not possible)

Pseudo-exhaustive, partially parallel - 6

Combined Pseudo-Exhaustive-Random Testing



Segmented LFSR

Combined PE-PR Test

Circuit with 4 cones

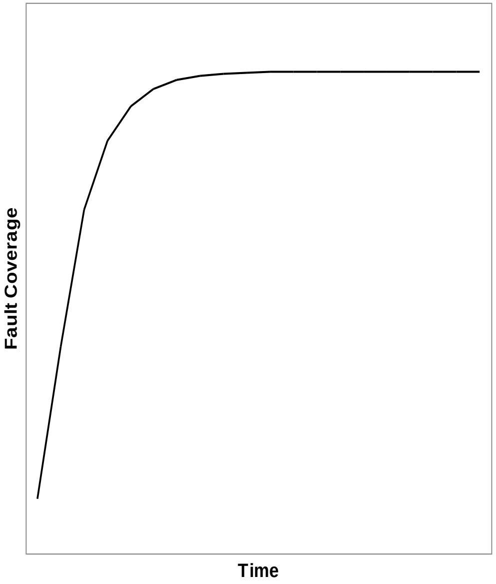
A set of Partial Pseudo-Exhaustive tests can be combined with

1. **(1) Pseudorandom BIST or**
2. **(2) Stored Deterministic test set**

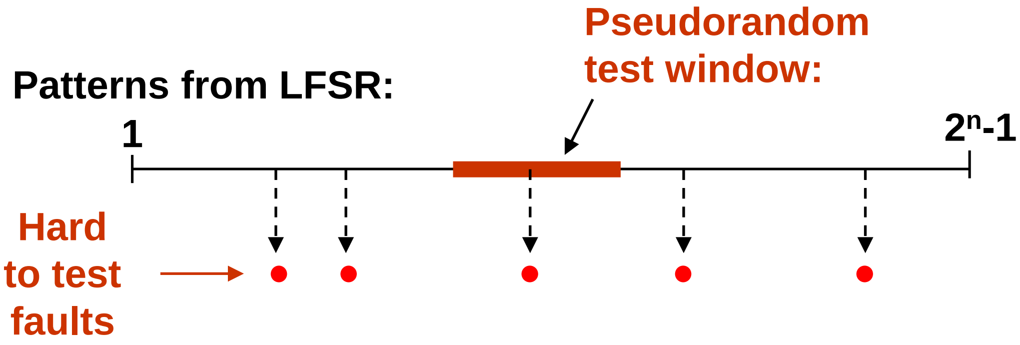
BIST: Hard to Test Faults

The main motivations of using random patterns are:

- low generation cost
- high initial efeciency



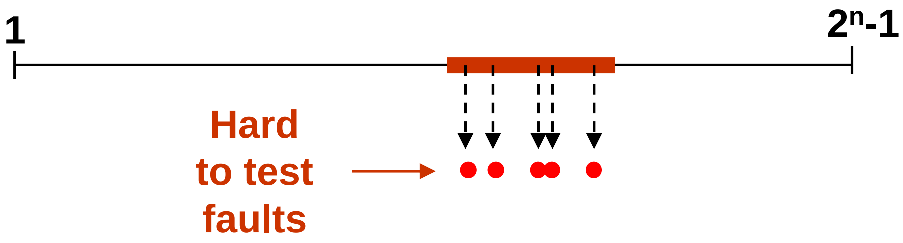
Problem: Low fault coverage


Pseudorandom test window:

Patterns from LFSR:

Hard to test faults

Dream solution: Find LFSR such that:

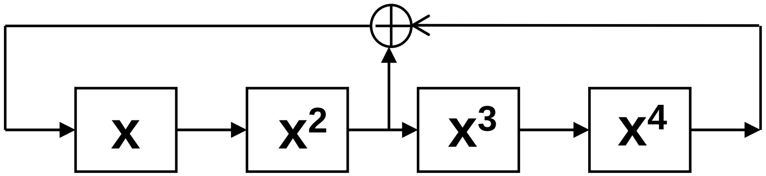


Hard to test faults

Pseudorandom Test with Embedded HW

Non-primitive polynomial

$x^4 + x^2 + 1$



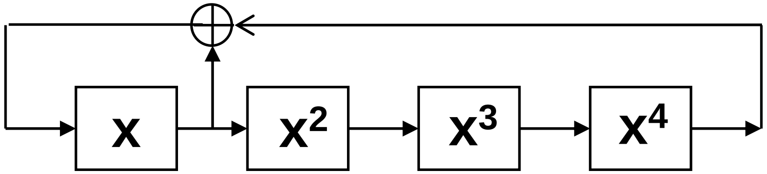
0001
1000
0100
1010
0101
0010
0001

1001
1100
1110
1111
0111
0011
1001

0110
1011
1101
0110

Primitive polynomial

$x^4 + x + 1$



0001
1000
1100
1110
1111
0111

1011
0101
1010
1101
0110
0011

1001
0100
0010
0001

How to Recognize a Primitive Polynomial

Is x^4+x^2+1 **a primitive polynomial?**

Divisibility check:

$$
\begin{array}{r|l}
 & x^{11}+x^{9}+x^{5}+x^{3} \\
\hline
x^{4}+x^{2}+1 & x^{15}+1 \\
 & \underline{x^{15}+x^{13}+x^{11}} \\
 & x^{13}+x^{11}+1 \\
 & \underline{x^{13}+x^{11}+x^{9}} \\
 & x^{9}+1 \\
 & \underline{x^{9}+x^{7}+x^{5}} \\
 & x^{7}+x^{5}+1 \\
 & \underline{x^{7}+x^{5}+x^{3}} \\
 & x^{3}+1
\end{array}
$$

A primitive polynomial of degree *n* is characterized by:

(1) An odd number of terms including 1 term?

Yes, it includes 3 terms

(2) Divisibility into $1+x^k$, where $k=2^n-1$

No, there is remainder

x^4+x^2+1 **is non-primitive?**

Pseudorandom testing

Comparison of test sequences generated:

Primitive polynomials		Non-primitive polynomials	
x^3+x+1	x^3+x^2+1	x^3+1	x^3+x^2+x+1
100	100	100	100
110	010	010	110
111	101	001	011
011	110	100	001
101	111	010	100
010	011	001	110
001	001	100	011
100	100	010	001

Süsteemide diagnostika

4. Testide süntees digitaalsüsteemidele

- 4.1. Deterministlik testide süntees kombinatsioonskeemidele
- 4.2. Testide genereerimine otsustusdiagrammide abil
- 4.3. Triviaalsete (pseudotäielike) testide süntees
- 4.4. Testide süntees kordsetele riketele (üldjuht)
- 4.5. Testide süntees digitaalsüsteemidele kõrgtasandil

Multiple Fault Testing

- **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**

Wire a 0,1,x

Wire b 0,1,x

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

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

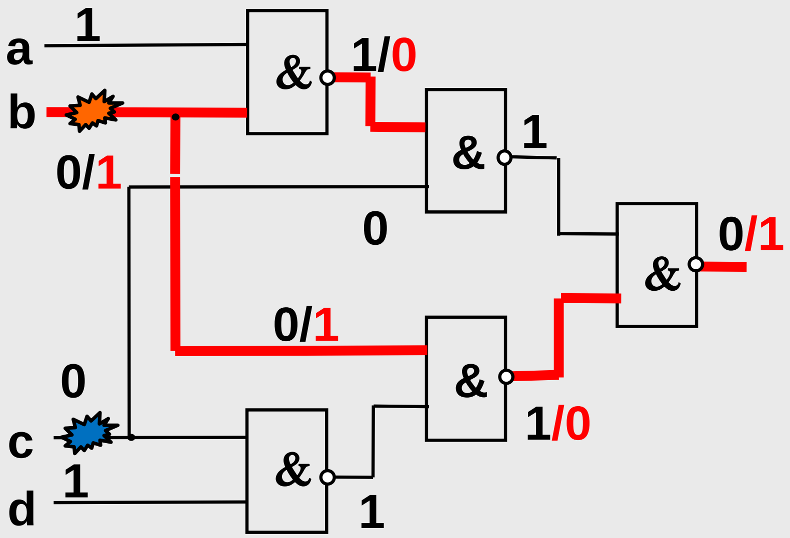
C_n^i **– number of sets of *i* lines,** 2^i **- number of faults on the set**

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

Multiple Fault Testing

No fault masking – No problem for single fault assumption

Multiple fault *F* may be not detected by a complete test *T* for single faults because of circular masking among the faults in *F*



Test pattern set

T = {1111, 0111, 1110, **1001**, 1010, 0101}

detects every single fault

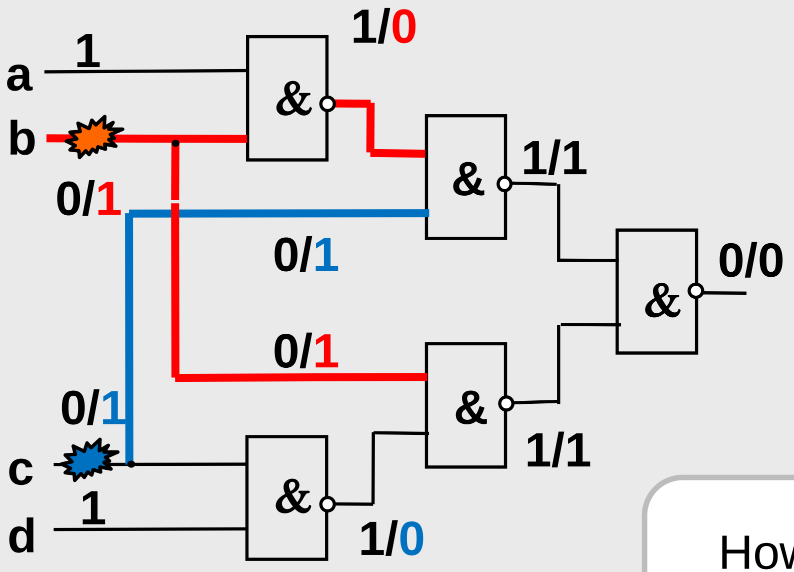
The only test for detecting

$b \equiv 1$ or $c \equiv 1$ is **1001**

Multiple Fault Testing

The problem arised: Fault Masking

Multiple fault *F* may be not detected by a complete test *T* for single faults because of circular masking among the faults in *F*



Test pattern set

T = {1111, 0111, 1110, **1001**, 1010, 0101}

detects every single fault

The only test for detecting $b \equiv 1$ or $c \equiv 1$ is **1001**

However, $b \equiv 1$ masks $c \equiv 1$

and $c \equiv 1$ masks $b \equiv 1$

Multiple Fault Testing

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

Two approaches to testing:

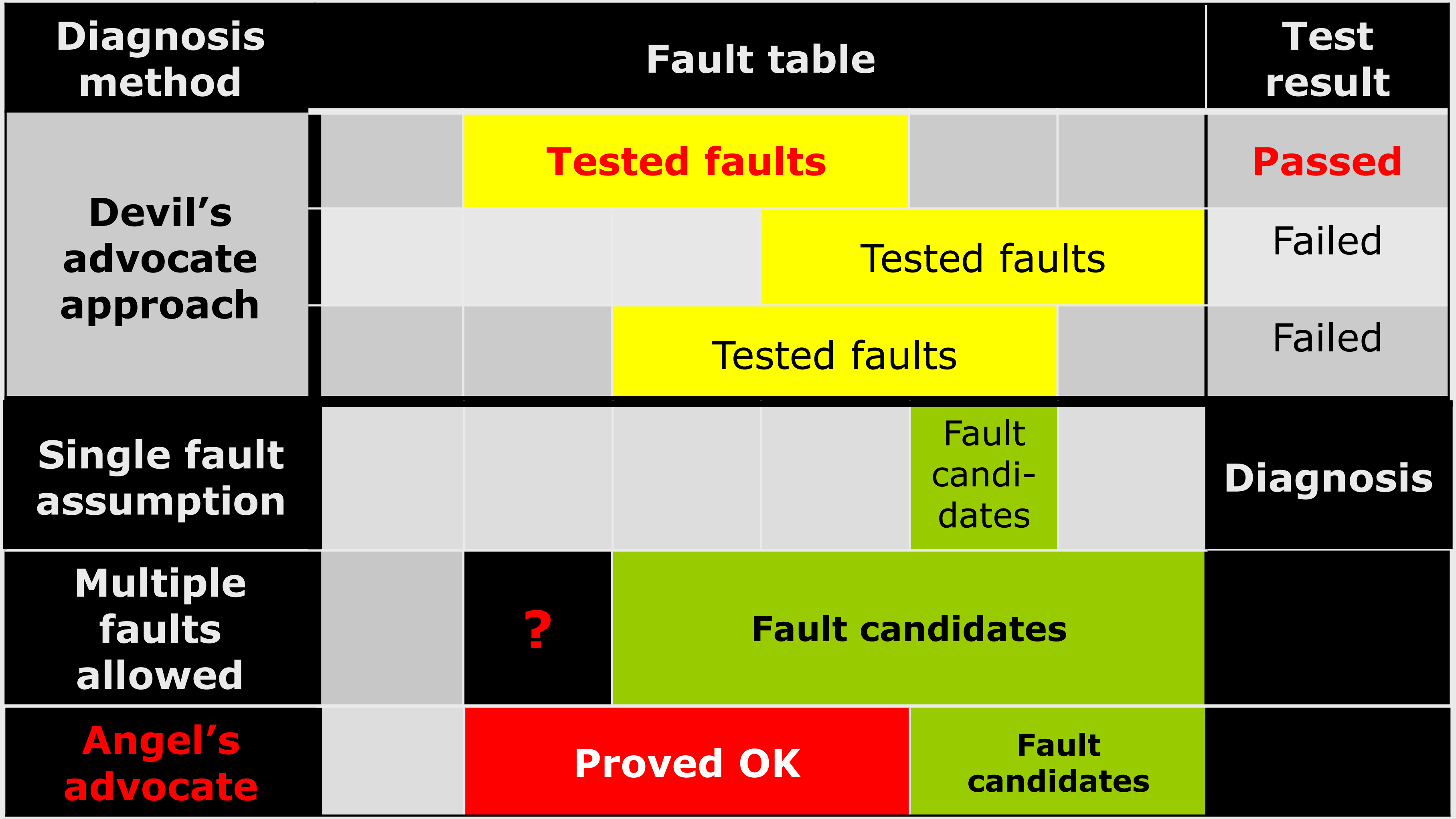
Devil's advocate

- ✓ **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 signal-lines in the circuit**

Fault Diagnosis Dilemmas

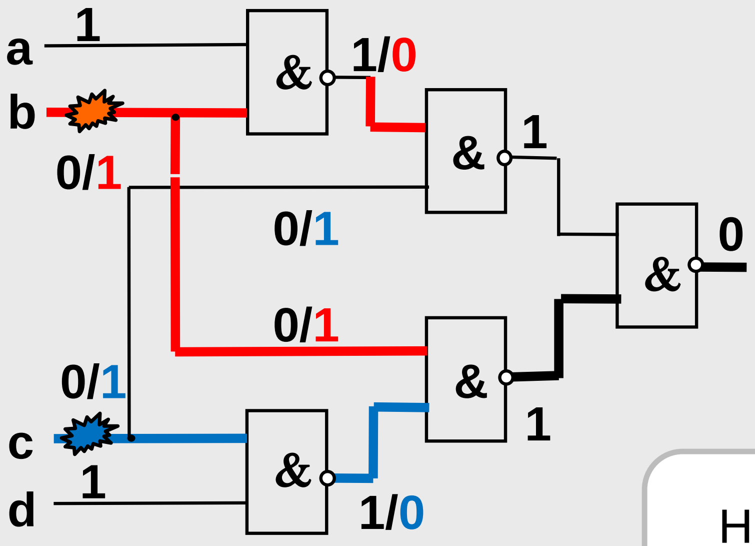


Diagnosis method	**Fault table**						**Test result**
Devil's advocate approach		**Tested faults**					**Passed**
				Tested faults			Failed
			Tested faults				Failed
Single fault assumption					Fault candi-dates		**Diagnosis**
Multiple faults allowed		**?**	**Fault candidates**				
Angel's advocate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

Multiple Fault Testing

The problem: Fault Masking

Multiple fault *F* may be not detected by a complete test *T* for single faults because of circular masking among the faults in *F*



Test pattern set

T = {1111, 0111, 1110, 1001, 1010, 0101}

detects every single fault

The only test for detecting

$b \equiv 1$ or $c \equiv 1$ is 1001

However, $b \equiv 1$ masks $c \equiv 1$

and $c \equiv 1$ masks $b \equiv 1$

Multiple Boolean Derivatives

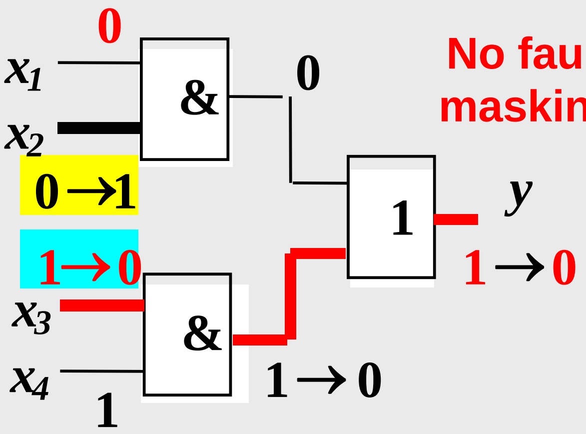
$$y = x_1 x_2 \vee x_3 x_4$$

Test for x_3

$$\frac{\partial y}{\partial x_3} = \bar{x}_1 x_4 \vee \bar{x}_2 x_4 = 1$$

$$\frac{\partial^2 y}{\partial x_2 \partial x_3} = \frac{\partial}{\partial x_2}\left(\frac{\partial y}{\partial x_3}\right) = x_1 x_4 = 0$$

Two faults

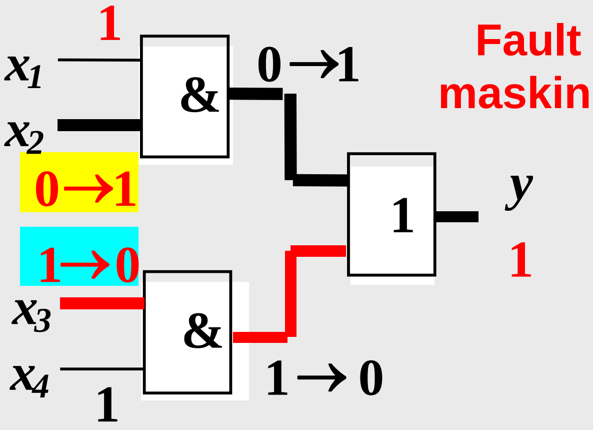


No fault masking

Fault in x_2 cannot mask the fault in x_3

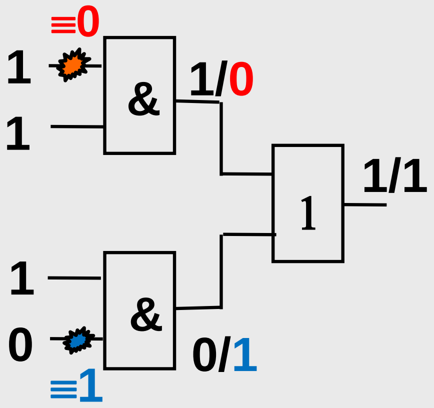
$$x_1 x_4 = 1$$

Faults

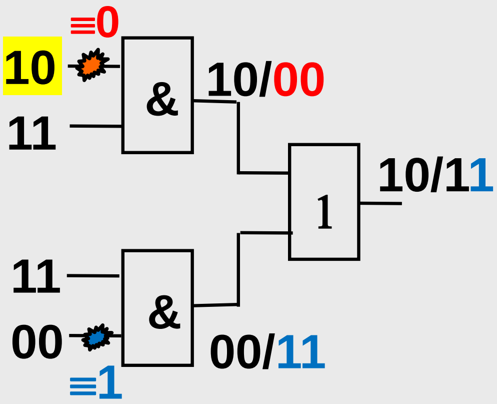


Fault masking

How to Prove that the Fault is Missing?



Fault masking



Test Pair: Fault is detected

But, which fault?

Test Pairs for Multiple Fault Testing

Testing of multiple faults by pairs of patterns

To prove that a path is fault-free
under any multiple faults,
two pattern test is needed

The lower path from *b* is under test

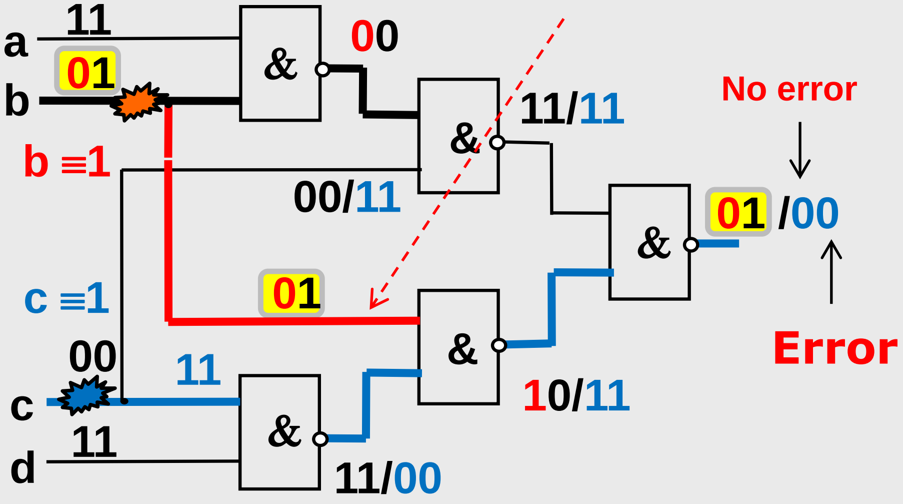
A pair of patterns is applied on *b*

There is a masking fault $c \equiv 1$

1st pattern: fault *b* ≡ 1 is masked

2nd pattern: fault *c* ≡ 1 is detected

Tested path for b ≡ 1/0



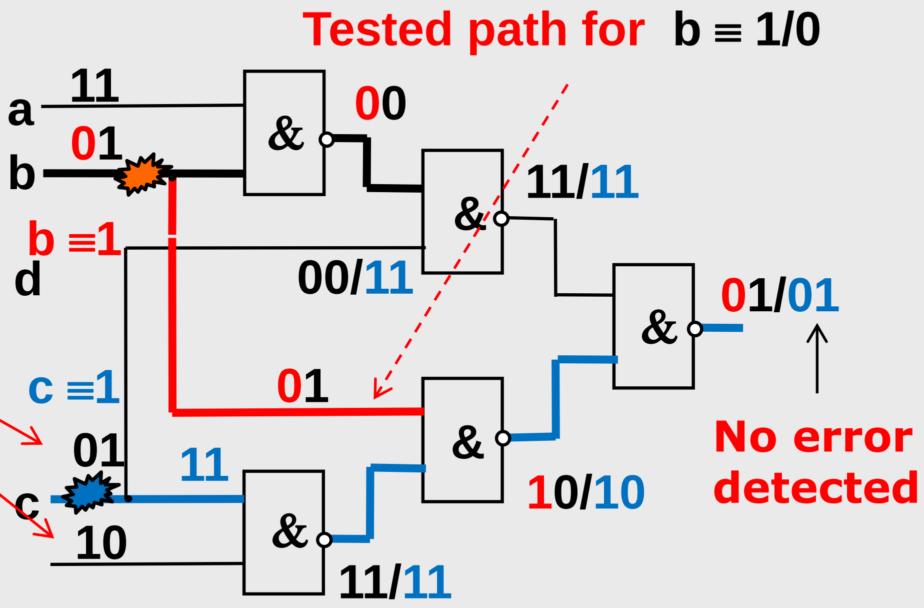
Either
the fault on the path is detected or
the masking fault is detected

The trick: 1st pattern tests b
2nd pattern tests c

Test Pair is not Detecting the Fault(1)

Test pair is not correct

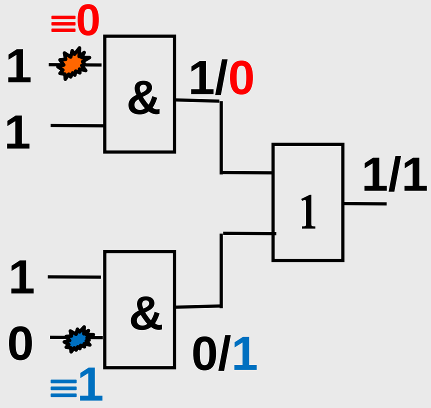
1. Test pair is not systematic – more than one variable is changing the value („Bad“ organizing of the test)



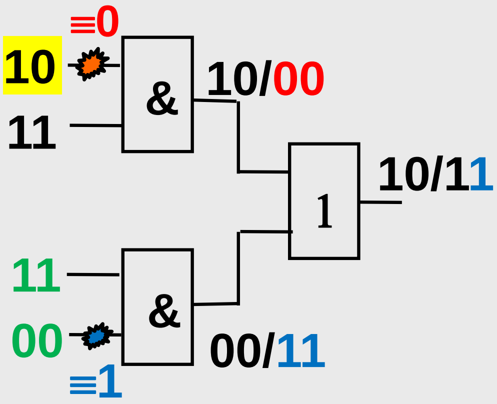
1st pattern: fault $b \equiv 1$ is masked

2nd pattern: fault $c \equiv 1$ is masked

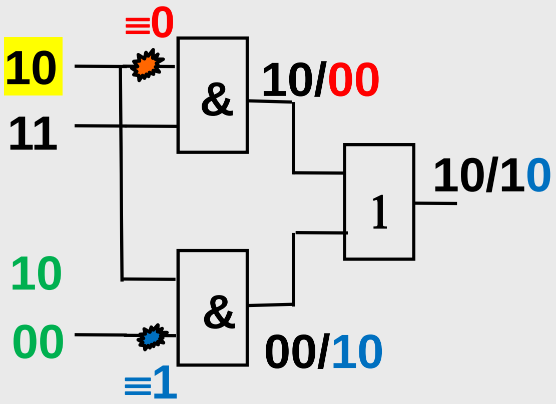
Test Pair is not Detecting the Fault(2)



Fault masking



Test Pair: Fault is detected



2. Test Pair is not working:
Fault is masked by another fault due to corruption of the test pair because of fan-out

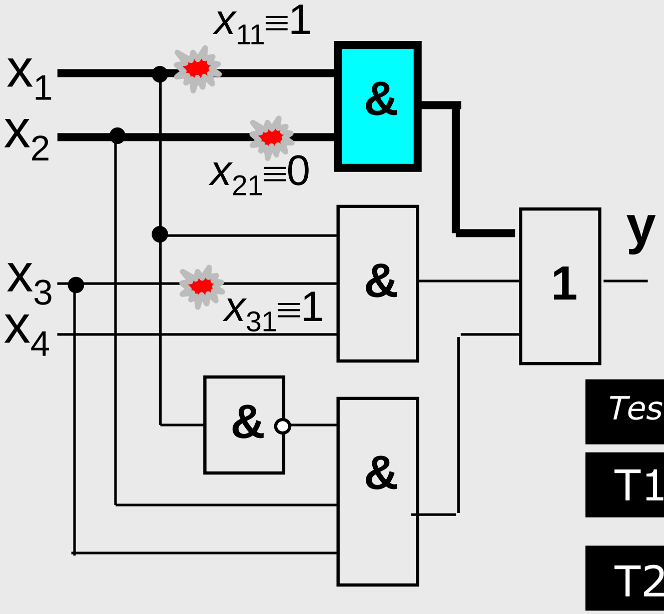
Test Pair Does not Work

How still avoid multiple fault masking

$$y = x_{11}x_{21} \vee x_{12}x_{31}x_4 \vee \overline{x_{13}}x_{22}x_{32}$$

Multiple fault:

$X_{11} \equiv 1, X_{21} \equiv 0, X_{31} \equiv 1$



Fault masking

T1 **T2** **T3**

Fault is detected

$x_{11} \equiv 1$ $x_{21} \equiv 0$ $x_{31} \equiv 1$

Test	x_{11}	x_{21}	x_{12}	x_{31}	x_4	$!x_{13}$	x_{22}	x_{32}	Y	Y^F
T1	0	1	0	0	1	1	1	0	**0**	**0**
T2	**1**	1	**1**	0	1	**0**	1	0	**1**	**1**
T3	1	**0**	1	0	1	0	0	0	**0**	**1**

The concept of **Test Group** →

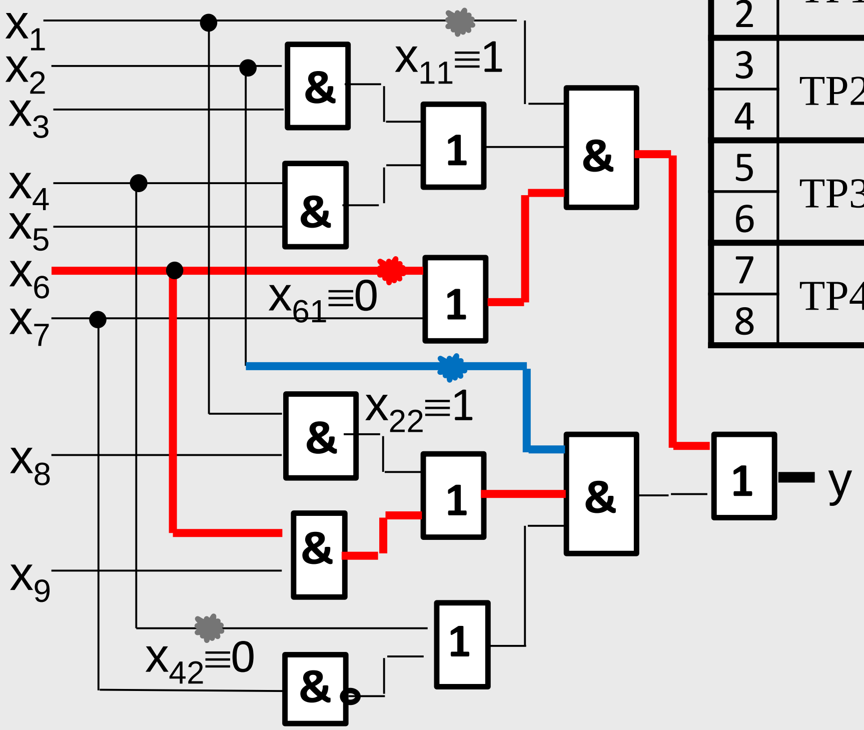
The concept of **Test Pair**

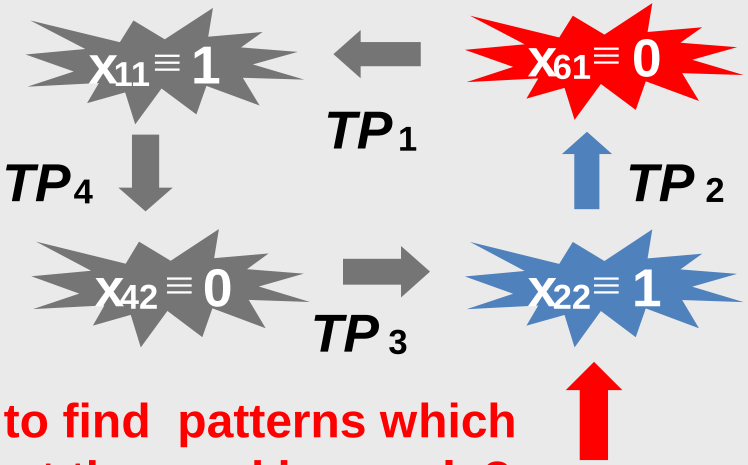
Ring Masking with using Test Pairs

Bad news:

Test pairs don't help always

t	Test type	Test pairs $TP_t = \{T_t, T_{t+1}\}$									Test faults	Mask faults
		x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9		
1	TP1	0	0	-	1	1	1	0	1	0	**$x_{11} \equiv 1$**	$x_{61} \equiv 0$
2		1	0	-	1	1	1	0	1	0	$x_{61} \equiv 0$	$x_{22} \equiv 1$
3	TP2	1	0	-	1	1	1	0	0	1	**$x_{61} \equiv 0$**	$x_{22} \equiv 1$
4		1	0	-	1	1	0	0	0	1	$x_{22} \equiv 1$	$x_{42} \equiv 0$
5	TP3	0	0	1	1	0	1	1	-	1	**$x_{22} \equiv 1$**	$x_{42} \equiv 0$
6		0	1	1	1	0	1	1	-	1	$x_{42} \equiv 0$	$x_{11} \equiv 1$
7	TP4	0	1	0	1	1	1	1	-	1	**$x_{42} \equiv 0$**	$x_{11} \equiv 1$
8		0	1	0	0	1	1	1	-	1	$x_{11} \equiv 1$	$x_{61} \equiv 0$





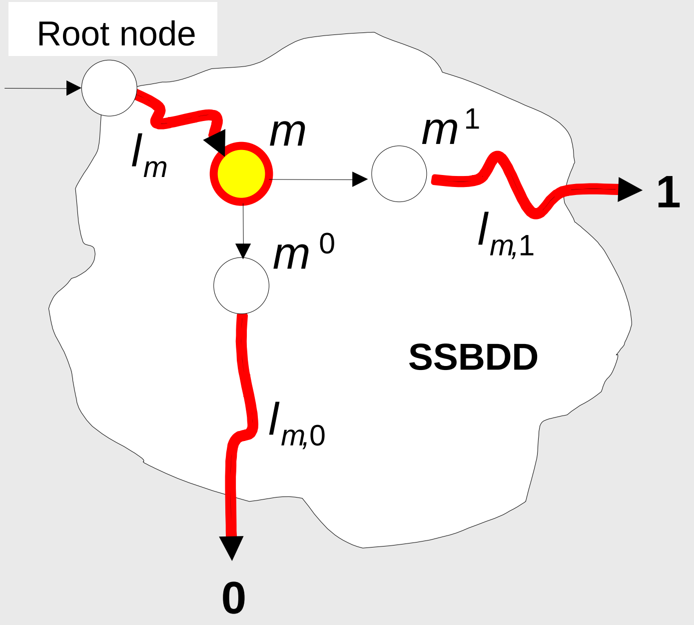
How to find patterns which will cut the masking cycle?

Test Group Conception

- ✓ **The method of test pairs does not work always**
- ✓ We consider now a new method for generating test patterns immune to fault masking
- ✓ Unlike the traditional **devil's advocate** approach, where the **faults** are used as test targets, a novel **angel's advocate** approach is proposed to verify the **correctness of sub-circuits**
- ✓ The proposed method is based on the new concept of **test groups**
- ✓ As the model for solving the task, **Decision Diagrams** are used to allow efficient **topological reasoning** of multiple faults mutual masking

Topological Idea of Test Generation

BDD (SSBDD) for modeling
a function Y = F(X)



The node **m** is to be tested

Three paths should be activated:

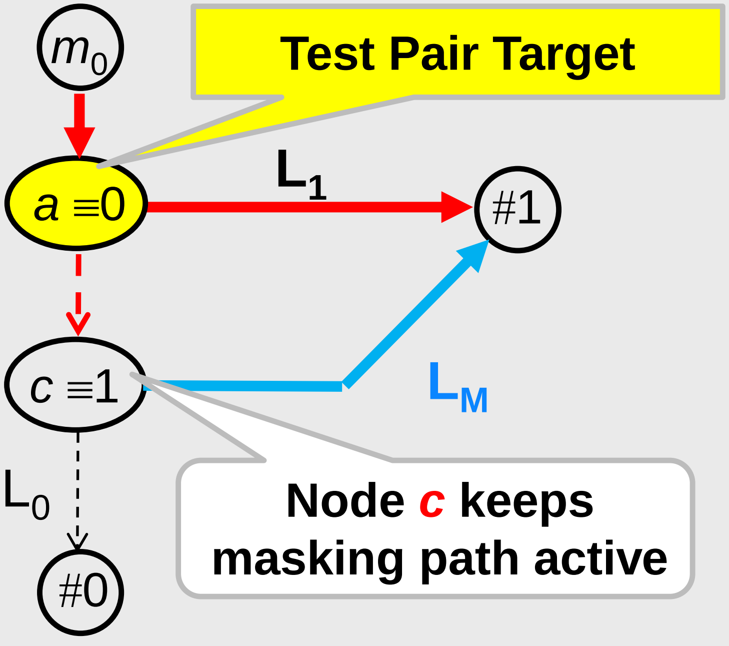
(1) a path l_m from **root** to **m**
(2) a path $l_{m,1}$ from m^1 to **terminal 1**
(3) a path $l_{m,0}$ from m^0 to **terminal 0**

Then
if variable(**m**) = 1 then **Y = 1**
else
if variable(**m**) = 0 then **Y = 0**

Why Test Pairs are not Sufficient?

L_1 - Path under test

L_M - Masking path

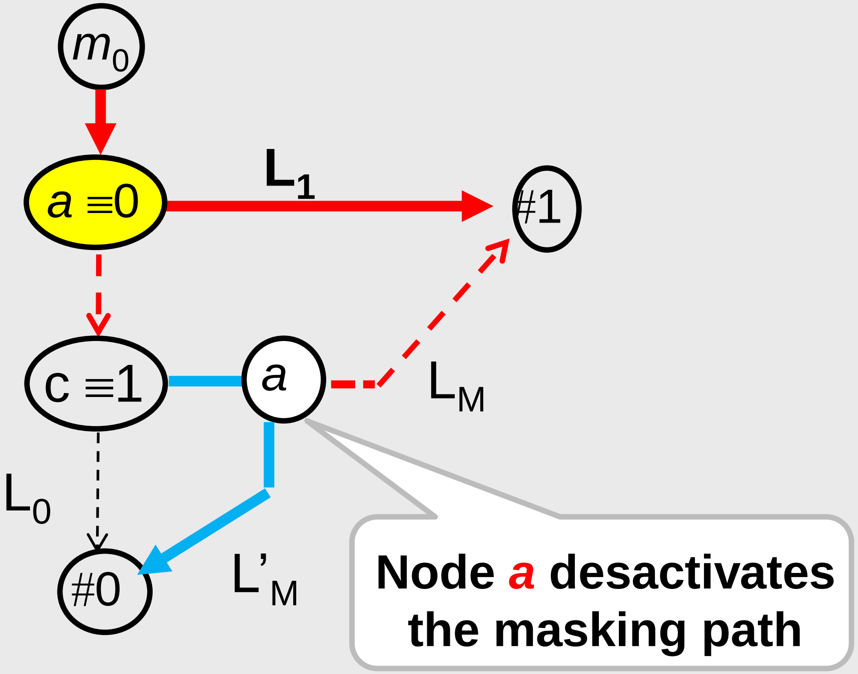


Test Pair Target

Node *c* keeps masking path active

Test pair for {a} works!

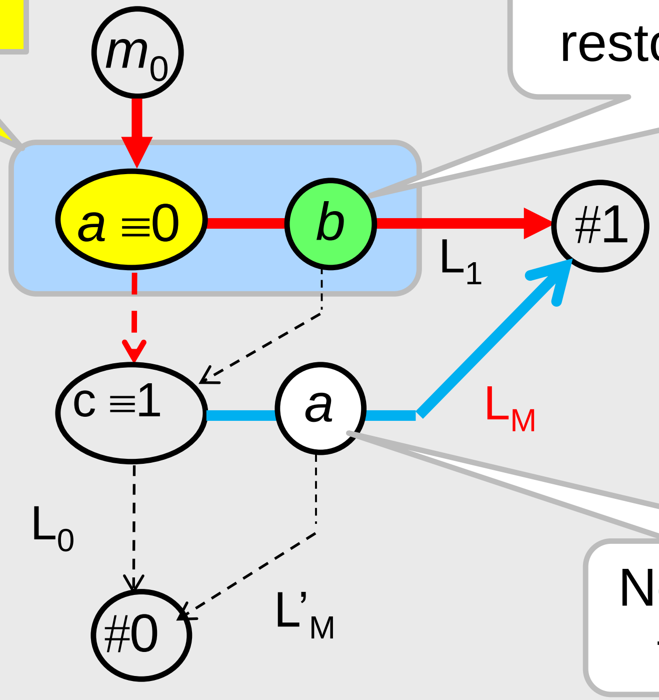
Test pair for {a} does not work!



Node *a* desactivates the masking path

Test Group Concept

Test Group Target



The 3rd test pattern for **b** restores the masking path

a	b	Y
1	1	1
0	1	0
1	0	**1**

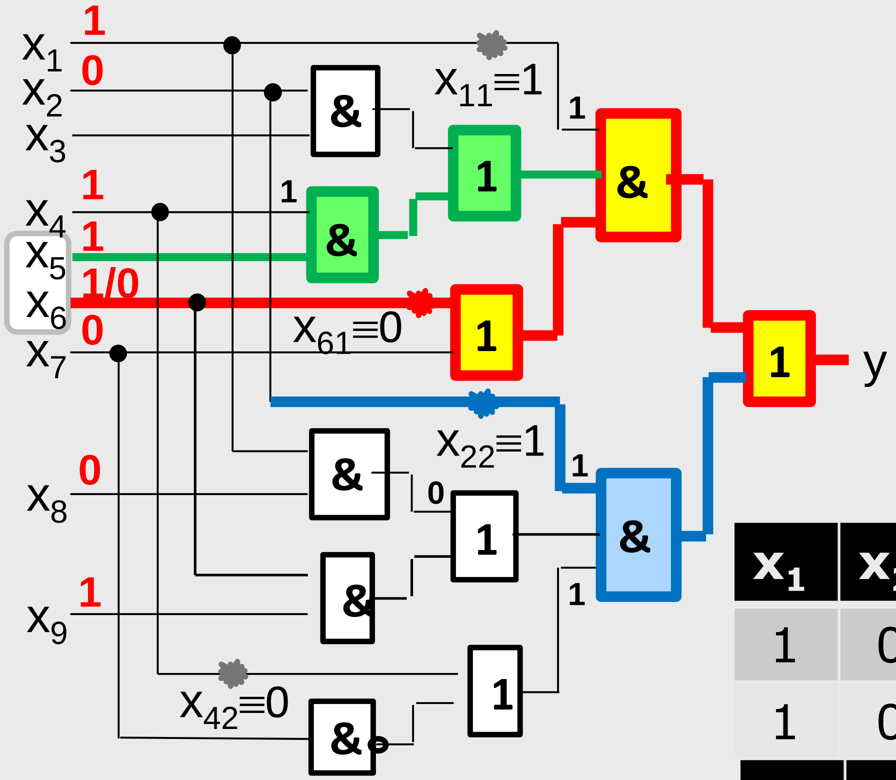
Node ***a*** desactivates the masking path

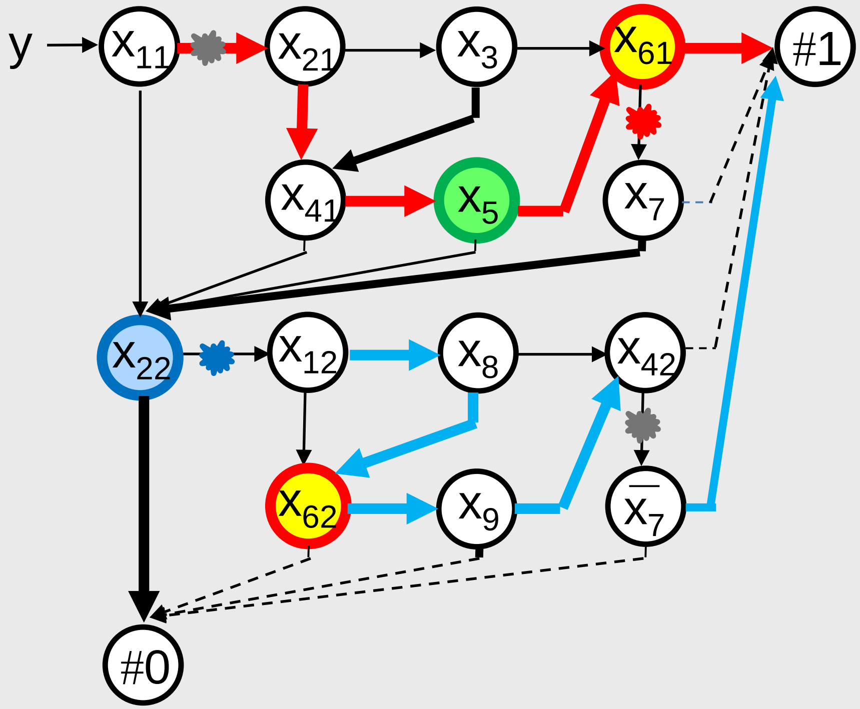
Test group for **{*a*, *b*}** works
Test group joins two test pairs

Test Group as Angel's Advocate Test

1) Fault $x_{61} \equiv 0$ is masked by $x_{22} \equiv 1$

2) Masking fault $x_{22} \equiv 1$ is not detected by the second pattern

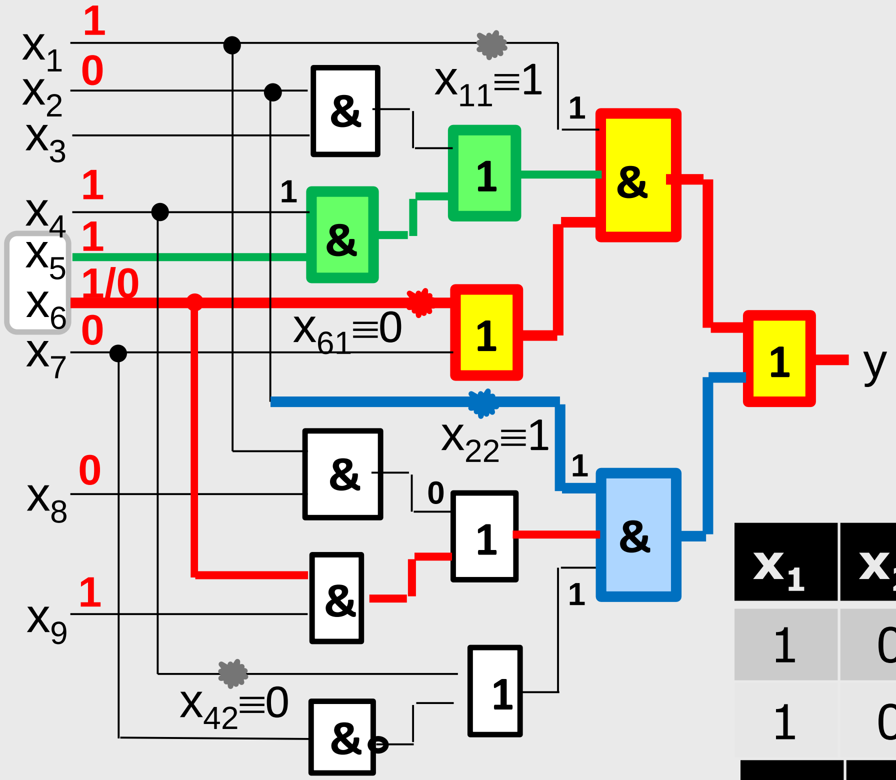


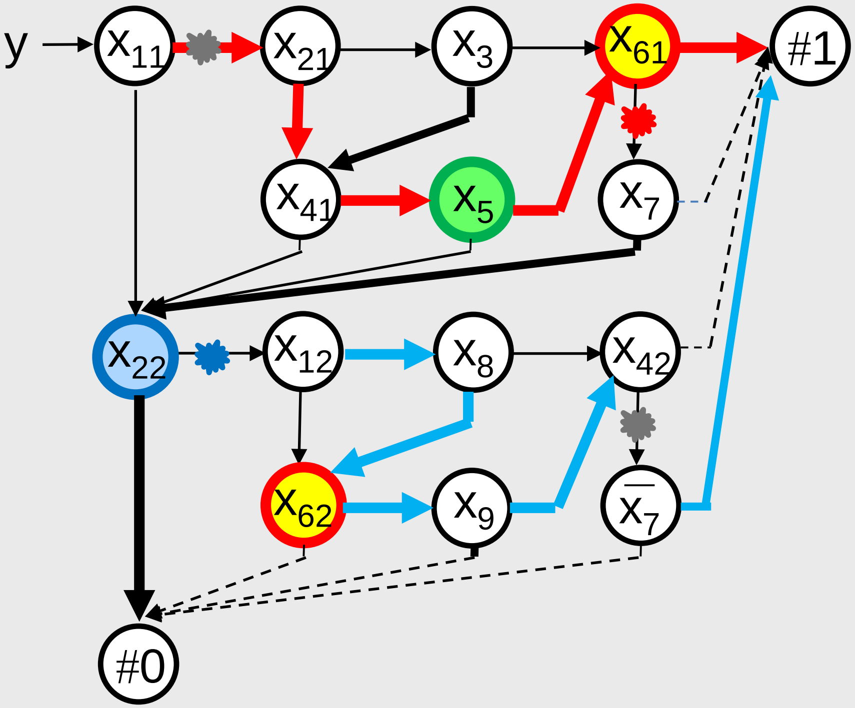


x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y
1	0	0	1	1	1	0	0	1	1/1
1	0	0	1	1	0	0	0	1	0/0
1	0	0	1	0	1	0	0	1	0/1

Test Group as Angel’s Advocate Test

Passed **test group** is a proof **that a sub-circuit is fault-free** at any multiple fault





x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y
1	0	0	1	1	1	0	0	1	1/1
1	0	0	1	1	0	0	0	1	0/0
1	0	0	1	0	1	0	0	1	0/1

Süsteemide diagnostika

4. Testide süntees digitaalsüsteemidele

- 4.1. Deterministlik testide süntees kombinatsioonskeemidele
- 4.2. Testide genereerimine otsustusdiagrammide abil
- 4.3. Triviaalsete (pseudotäielike) testide süntees
- 4.4. Testide süntees kordsetele riketele (üldjuht)
- 4.5. Testide süntees digitaalsüsteemidele kõrgtasandil

Faults and High-Level Decision Diagrams

RTL-statement:

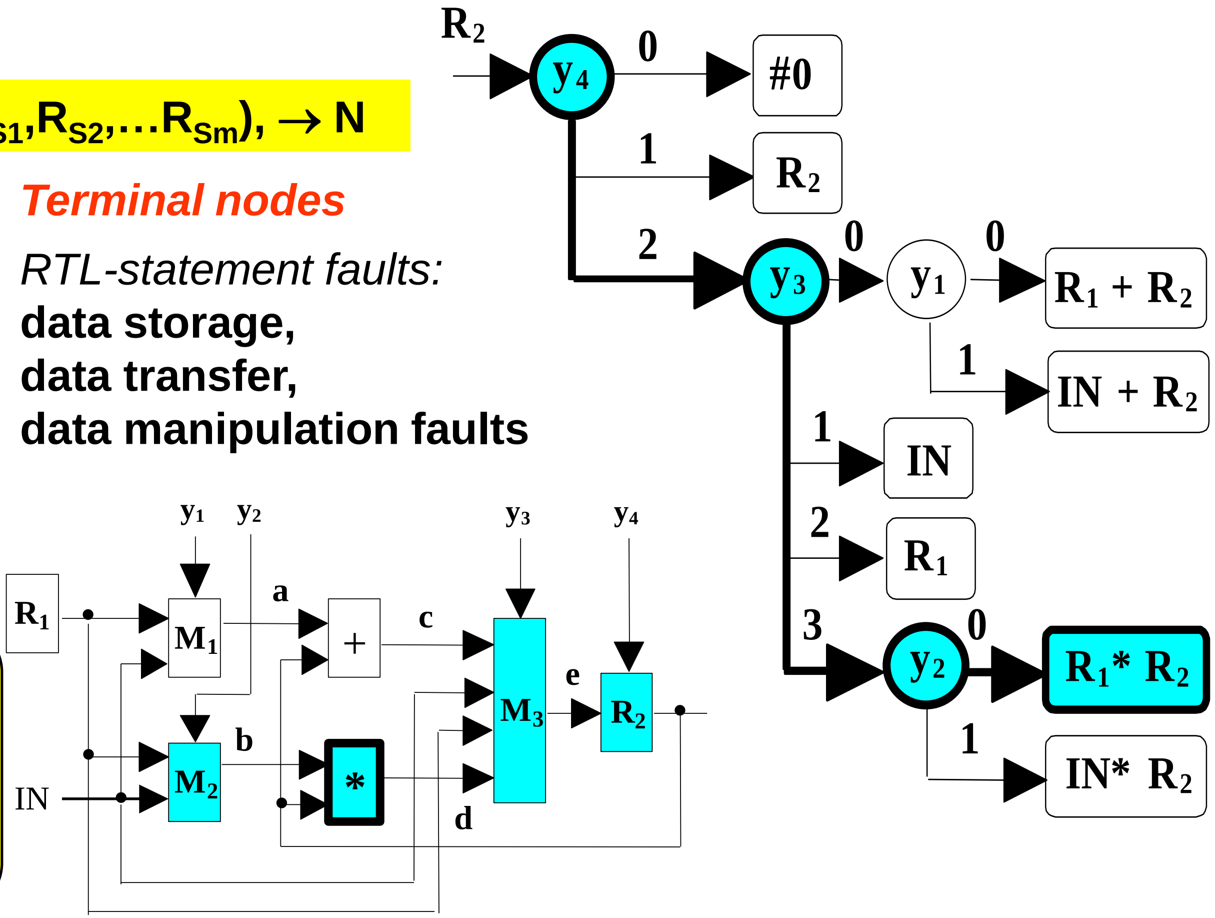
K: (*If* T,C) $R_D \leftarrow F(R_{S1},R_{S2},\ldots R_{Sm}), \rightarrow N$

Nonterminal nodes

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

Terminal nodes

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



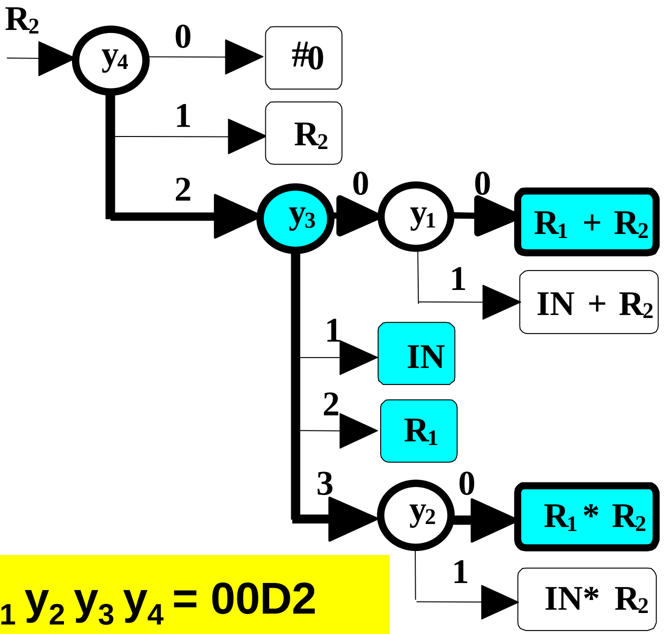
Testing concept
on the DD-model
(uniform for all nodes):
1) Exhaustive testing
2) Optimization

Test Generation for Digital Systems

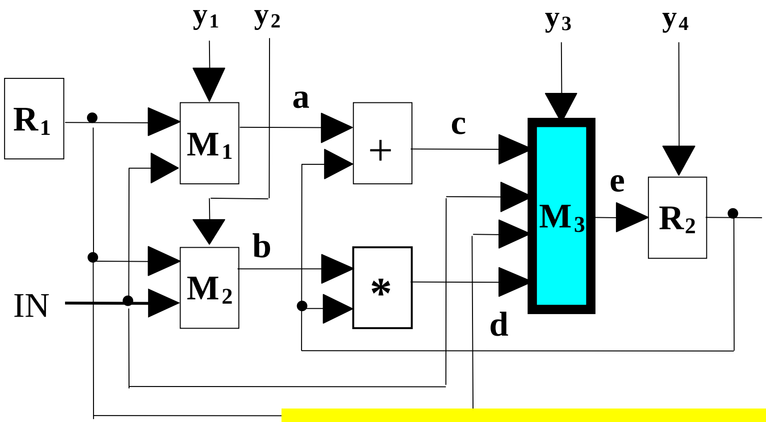
High-level test generation with DDs: Conformity test

Multiple paths activation in a single DD
Control function y_3 is tested

Decision Diagram



Data path



Test program:

Control: *For* **D = 0,1,2,3:** $y_1\, y_2\, y_3\, y_4 = 00D2$

Data: *Solution of* $R_1 + R_2 \neq IN \neq R_1 \neq R_1 * R_2$

Test Program Synthesis with HLDDs

Test algorithm:

Control: *For* **D = 0,1,2,3:** $y_1 y_2 y_3 y_4 = 00D2$

Data: *Solution of* $R_1 + R_2 \neq IN \neq R_1 \neq R_1 * R_2$

Comment:

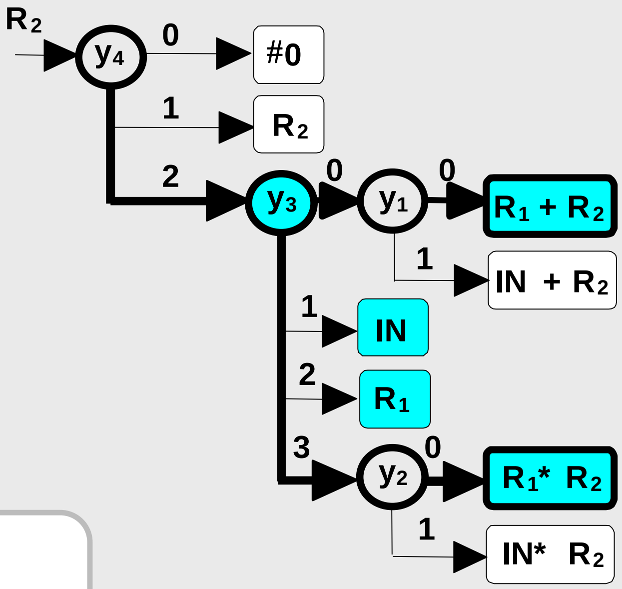
The data rule is simplified

***Test program*:**

```
For D = 0,1,2,3
    Begin
    Load R1 = IN1             } Initialization
    Load R2 = IN2
    Apply                     } Test
        IN = IN3
        $y_1 y_2 y_3 y_4$ = 00D2
    Read R2                   } Observation
    End
```

Data: *(IN1+IN2) ≠ IN3 ≠ IN1 ≠ (IN1*IN2)*

Decision Diagram



Advantages:

Straightforward synthesis procedure

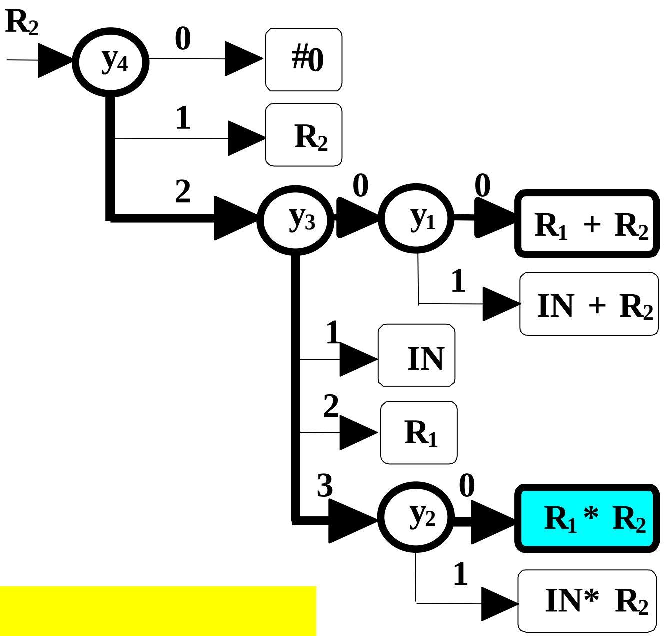
Compactness of the cycle-based test program

Test Generation for Digital Systems

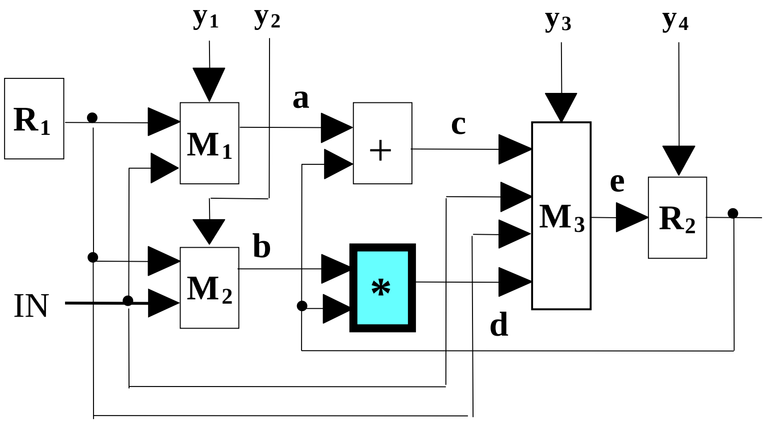
High-level test generation with DDs: Scanning test

Single path activation in a single DD
Data function R_1* R_2 is tested

Decision Diagram



Data path



Test program: **Control:** $y_1 y_2 y_3 y_4 = 0032$

Data: *For all specified pairs of* **(R_1, R_2)**

Test Generation for Digital Systems

High-level test generation with DDs: Scanning test

Test program:

```
For j=1,n
    Begin
        Load R1 = IN($j_1$)
        Load R2 = IN($j_2$)
        $y_1 y_2 y_3 y_4$ = 0032:
        Read R2
    End
```

IN(j_1) IN(j_2)

Test data

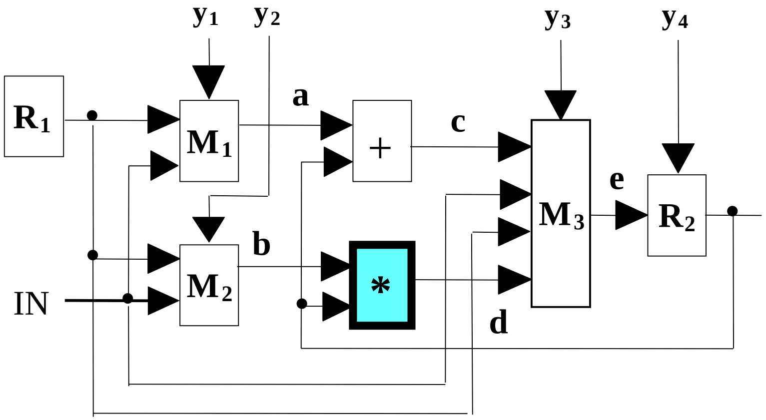
R2(j)

Test results

Test template:

Control: $y_1 y_2 y_3 y_4$ = 0032

Data: *For all specified pairs of* **(R_1, R_2)**



Scan-Path for Making Systems Transparent

Hierarhical test generation with Scan-Path: