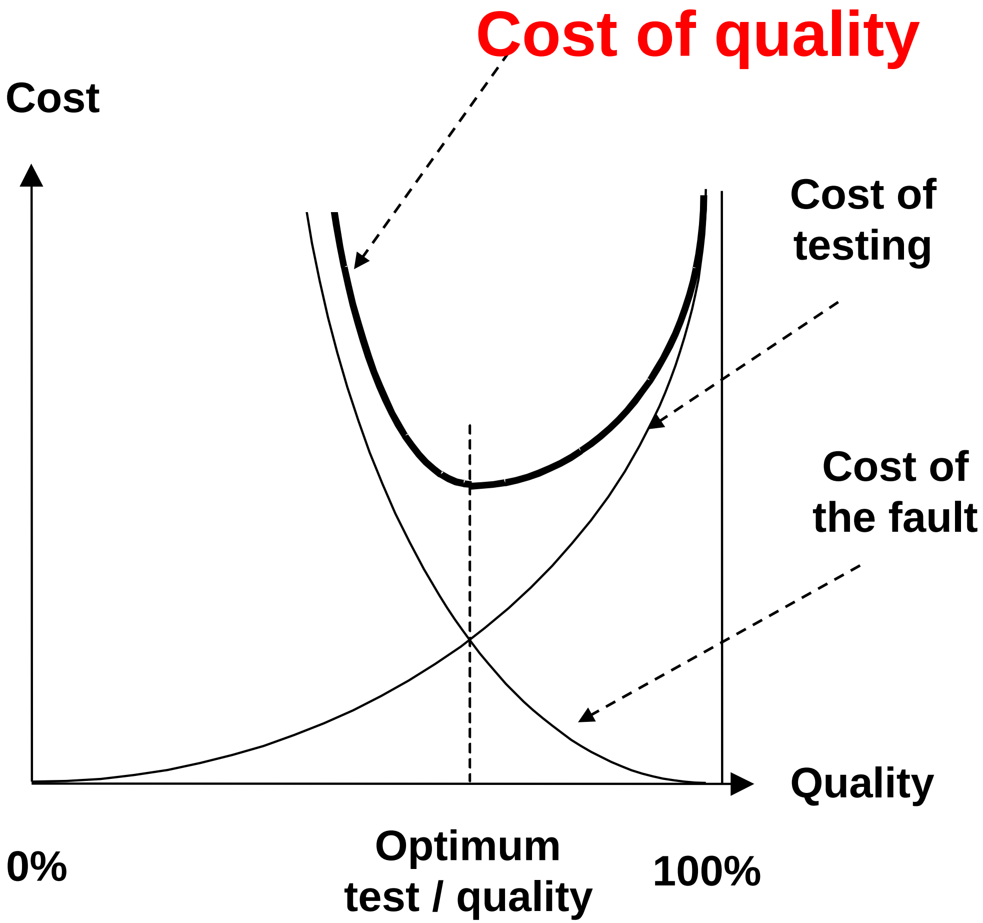
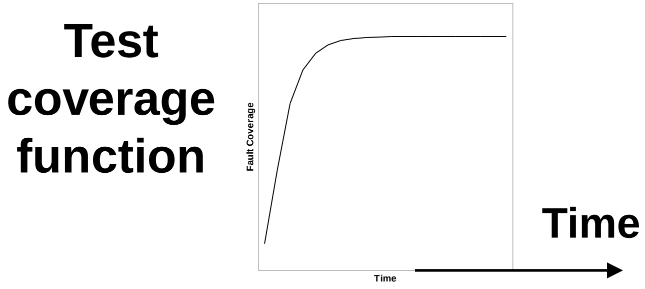
Introduction: The Problem is Money?



How to succeed?
Try too hard!
How to fail?
Try too hard!
(From American Wisdom)



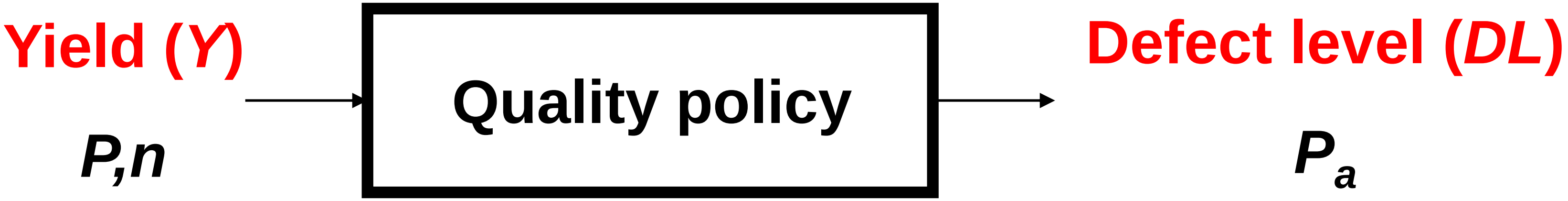
Conclusion:

“The problem of testing can only be contained not solved”

T.Williams

Design for Testability

The problem is - QUALITY:



Yield (Y)

P, n

Quality policy

Defect level (DL)

P_a

$$DL = \frac{P_a}{(1-P)^n + P_a} = 1-(1-P)^{n-m} = 1-Y^{\frac{n-m}{n}} = 1-Y^{(1-\frac{m}{n})} = 1-Y^{(1-T)}$$

n **- number of defects**
m **- number of faults tested**
P **- probability of a defect**
P

$$P_a = (1-P)^m - (1-P)^n$$

$$Y = (1-P)^n$$

Testability of Design Types

General important relationships:

- **T (Sequential logic) < T (Combinational logic)**

  Solutions: **Scan-Path design strategy**

- **T (Control logic) < T (Data path)**

  Solutions: **Data-Flow design, Scan-Path design strategies**

- **T (Random logic) < T (Structured logic)**

  Solutions: **Bus-oriented design, Core-oriented design**

- **T (Asynchronous design) < T (Synchronous design)**

Testability Estimation Rules of Thumb

Circuits less controllable

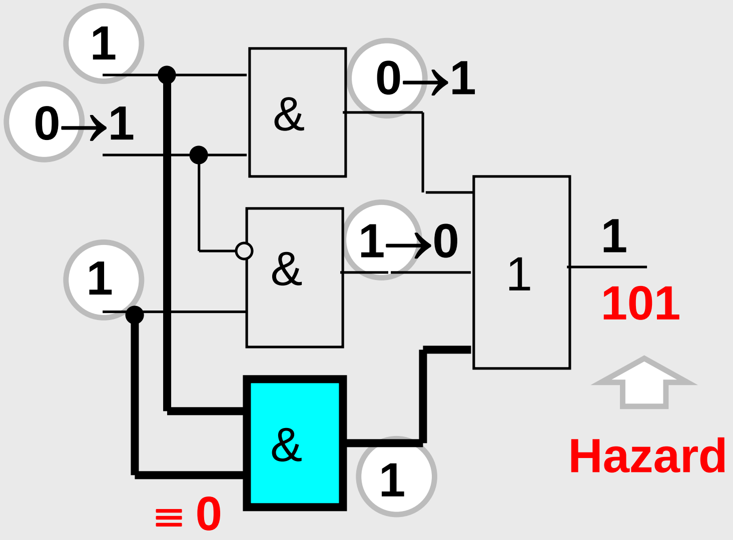
- Decoders
- Circuits with feedback
- Counters
- Clock generators
- Oscillators
- Self-timing circuits
- Self-resetting circuits

Circuits less observable

- Circuits with feedback
- Embedded
  - RAMs
  - ROMs
  - PLAs
- Error-checking circuits
- Circuits with redundant nodes

Fault Redundancy

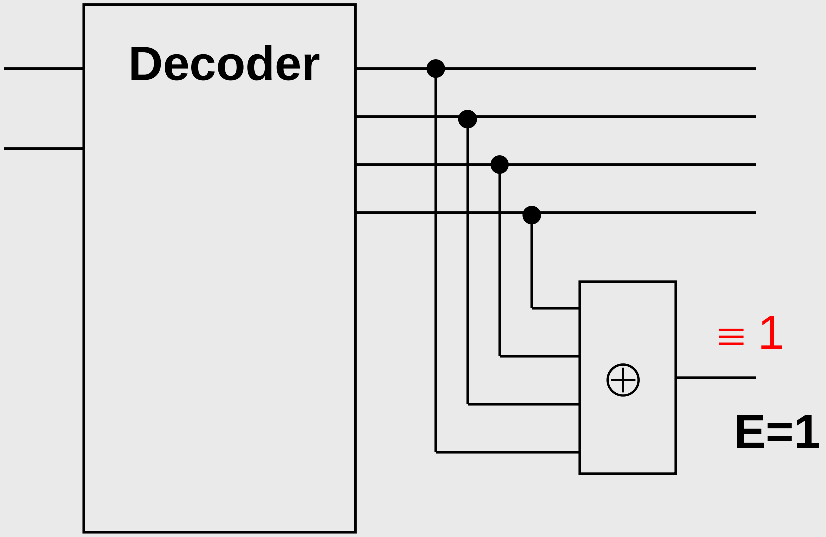
Hazard control circuit:



Redundant AND-gate

Fault ≡ 0 is not testable

Error control circuitry:



E = 1 if decoder is fault-free

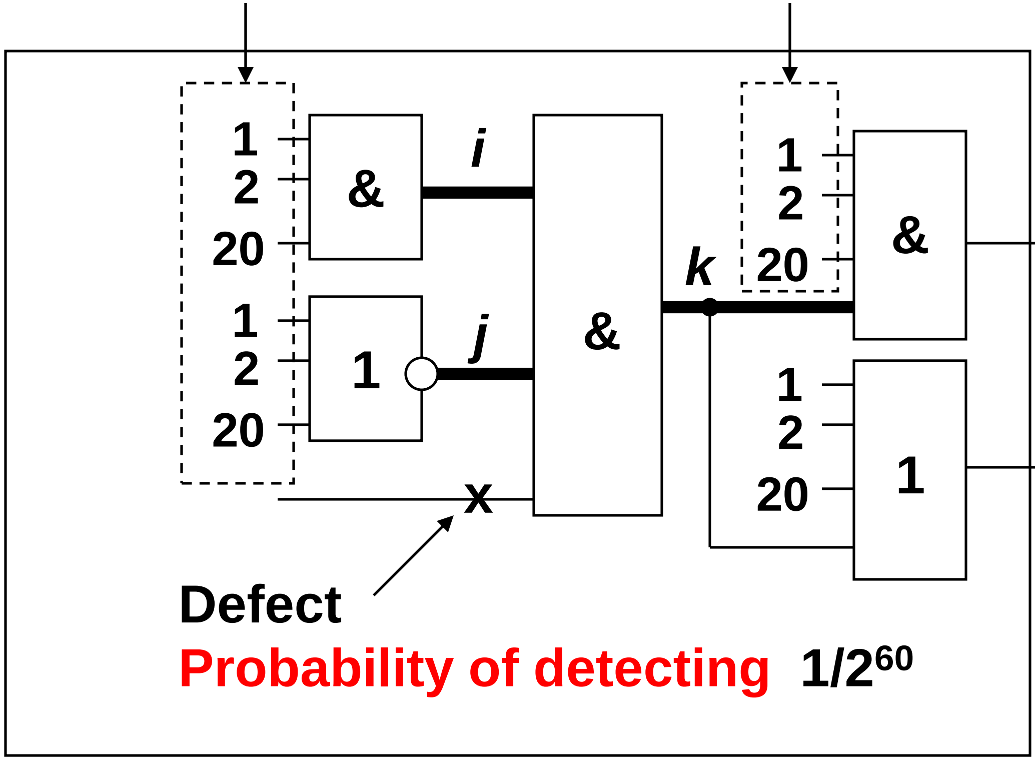
Fault ≡ 1 is not testable

Hard to Test Faults

Evaluation of testability:

- Controllability
  - C_0 (*i*)
  - C_1 (*j*)
- Observability
  - O_Y (*k*)
  - O_Z (*k*)
- Testability

Controllability for 1 needed



Defect

Probability of detecting $1/2^{60}$

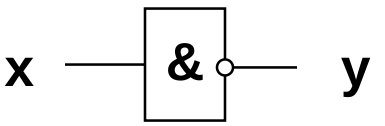
Probabilistic Testability Measures

Controllability calculation:

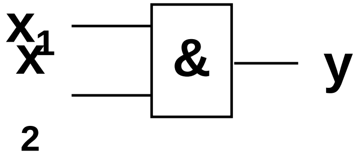
Value: minimum number of nodes that must be set in order to produce 0 or 1

For inputs: $C_0(i) = p(x_i=0)$ $C_1(i) = p(x_i=1) = 1 - p(x_i=0)$

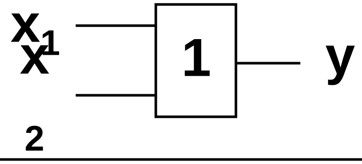
For other signals: recursive calculation rules:



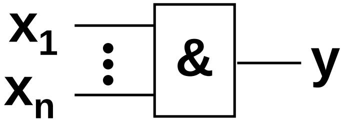
$p_y = 1 - p_x$



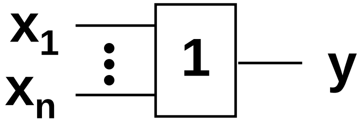
$p_y = p_{x1}\, p_{x2}$



$p_y = 1 - (1 - p_{x1})(1 - p_{x2})$



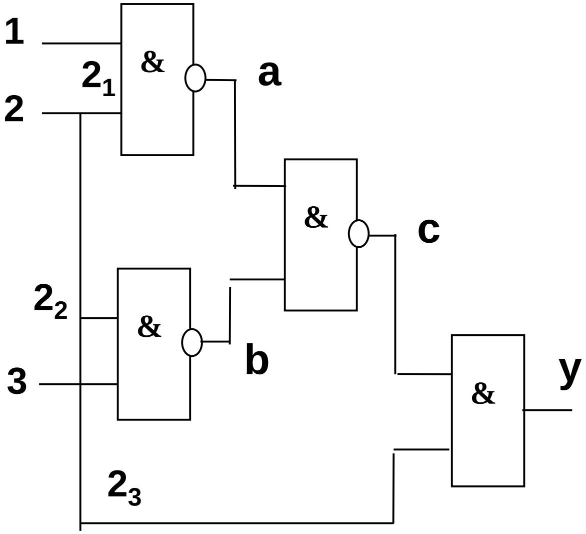
$$p_y = \prod_{i=1}^{n} p_{xi}$$



$$p_y = 1 - \prod_{i=1}^{n} (1 - p_{xi})$$

Calculation of Signal Probabilities

Straightforward methods:



For all inputs: $p_k = 1/2$

Calculation gate by gate:

$$p_a = 1 – p_1p_2 = 0{,}75,$$

$$p_b = 0{,}75,\ p_c = 0{,}4375,\ p_y = 0{,}22$$

Parker - McCluskey algorithm:

$$p_y = p_cp_2 = (1- p_ap_b)\ p_2 =$$

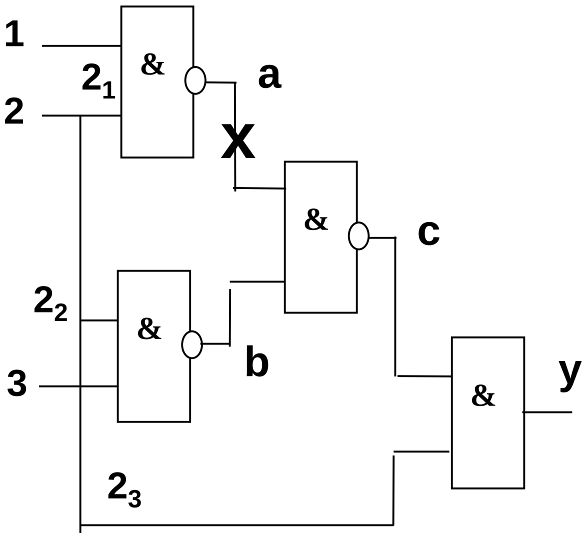
$$= (1 – (1- p_1p_2)\ (1- p_2p_3))\ p_2 =$$

$$= p_1p_2{}^2 + p_2{}^2p_3 - p_1p_2{}^3p_3 =$$

$$= p_1p_2 + p_2\,p_3 - p_1p_2p_3 = 0{,}38$$

Probabilistic Testability Measures

Parker-McCluskey:



For all inputs: $p_k = 1/2$

Observability:

$$p(\partial y/\partial a = 1) = p_b\, p_2 =$$

$$= (1 - p_2p_3)\, p_2 = p_2 - p_2^2p_3$$

$$= p_2 - p_2p_3 = 0{,}25$$

Testability:

$$p(a \equiv 1) = p(\partial y/\partial a = 1)\,(1 - p_a) =$$

$$= (p_2 - p_2p_3)(p_1p_2) =$$

$$= p_1p_2^2 - p_1p_2^2p_3 =$$

$$= p_1p_2 - p_1p_2p_3 = 0{,}125$$

Calculation of Signal Probabilities

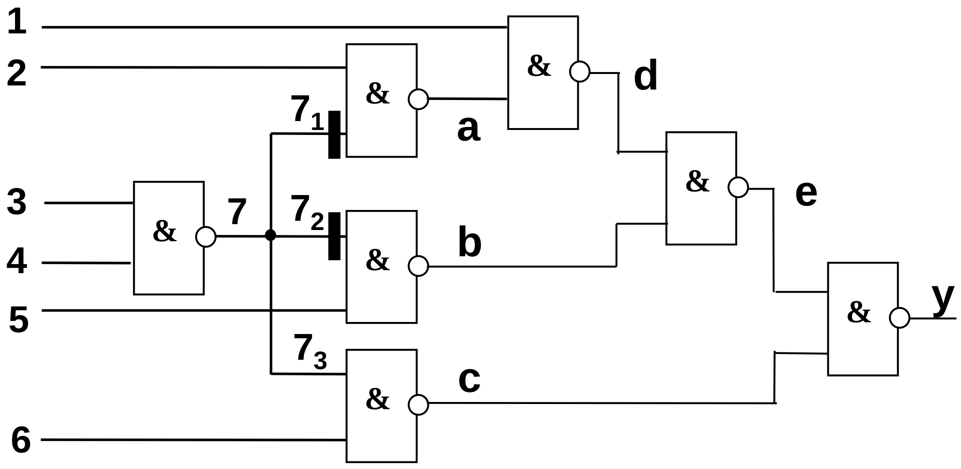
Cutting method

Idea:

- Complexity of exact calculation is reduced by using lower and higher bounds of probabilities

Technique:

- Reconvergent fan-outs are cut except of one
- Probability range of [0,1] is assigned to all the cut lines
- The bounds are propagated by straightforward calculation

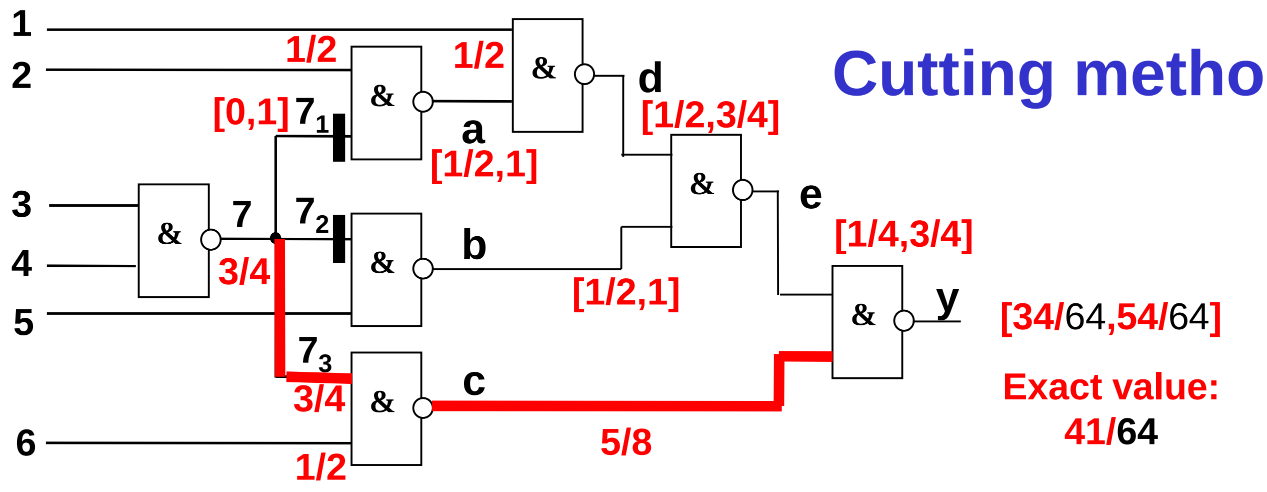


Lower and higher bounds for the probabilities of the cut lines:

p_{71} := (0;1), p_{72} := (0;1), p_{73} := 0,75

Calculation of Signal Probabilities

- For all inputs: p_k = 0,5
- Reconvergent fan-outs are cut except of one – 7_1 and 7_2
- Probability range of [0,1] is assigned to all the cut lines - 7_1 and 7_2
- The bounds are propagated by straightforward calculation



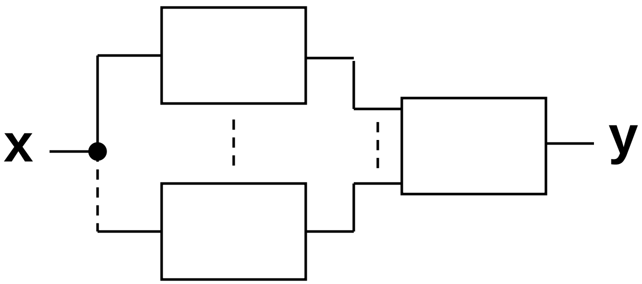
Cutting method

Calculation steps:

p_k	$[p_{LB}, p_{HB})$	Exact p_k	p_k	$[p_{LB}, p_{HB})$	Exact p_k
p_7	3/4	3/4	p_b	[1/2, 1]	5/8
p_{71}	[0, 1]	3/4	p_c	5/8	5/8
p_{72}	[0, 1]	3/4	p_d	[1/2, 3/4]	11/16
p_{73}	3/4	3/4	p_e	[1/4, 3/4]	19/32
p_a	[1/2, 1]	5/8	p_y	[34/64, 54/64]	41/64

Calculation of Signal Probabilities

Method of conditional probabilities



P(y) = p(y/x=0) p(x=0) + p(y/x=1) p(x=1)

Probabilitiy for – *y*

Conditions – *x* ∊ set of conditions

$$p(y) = \sum_{i \in (0,1)} p(y/(x=i)\, p(x=i)$$

Conditional probabilitiy

Idea of the method:

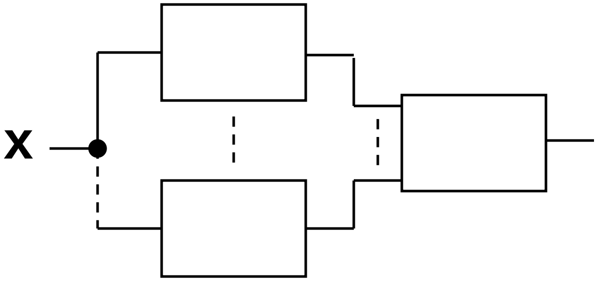
Two conditional probabilities are calculated along the paths (NB! not bounds as in the case of the cutting method)

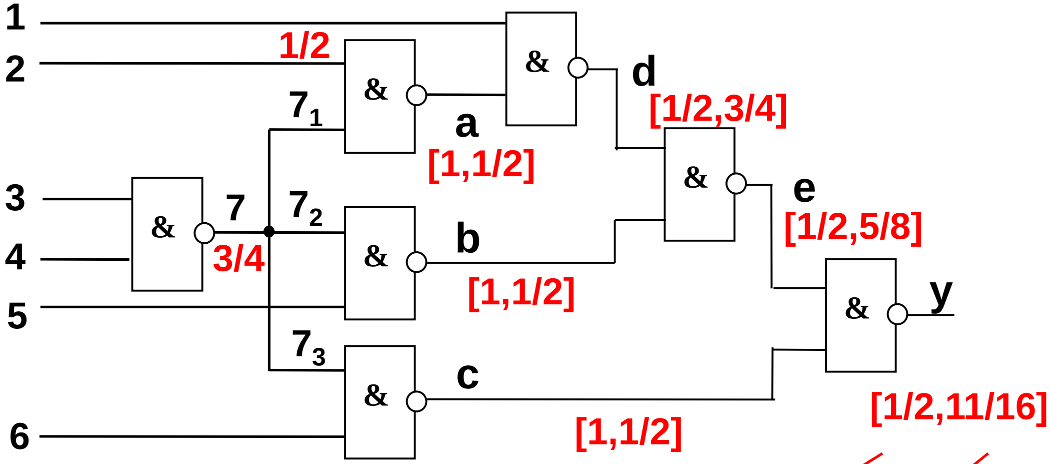
Since no reconvergent fanouts are on the paths, no danger for signal correlations

Calculation of Signal Probabilities

Method of conditional probabilities

$$p(y) = \sum_{i\in(0,1)} p(y/(x=i)\,p(x=i)$$





P_k	$[P_k^*, P_k^{**}]$	P_k	$[P_k^*, P_k^{**}]$
P_7		P_b	[1, 1/2]
P_{71}		P_c	[1, 1/2]
P_{72}		P_d	[1/2, 3/4]
P_{73}		P_e	[1/2, 5/8]
P_a	[1, 1/2]	P_y	[1/2, 11/16]

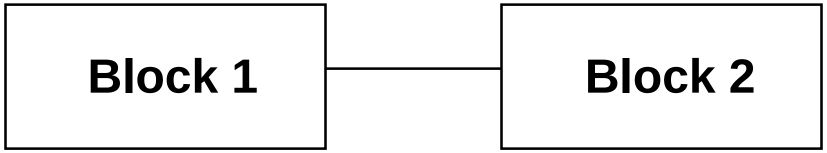
NB! Probabilities

$P_k = [P_k^* = p(x_k/x_7=0), P_k^{**} = p(x_k/x_7=1)]$
are propagated, not bounds
as in the cutting method.
For all inputs: $p_k = 1/2$

$p_y = p(y/x_7=0)(1 - p_7) + p(y/x_7=1)p_7 = (1/2 \times 1/4) + (11/16 \times 3/4) = 41/64$

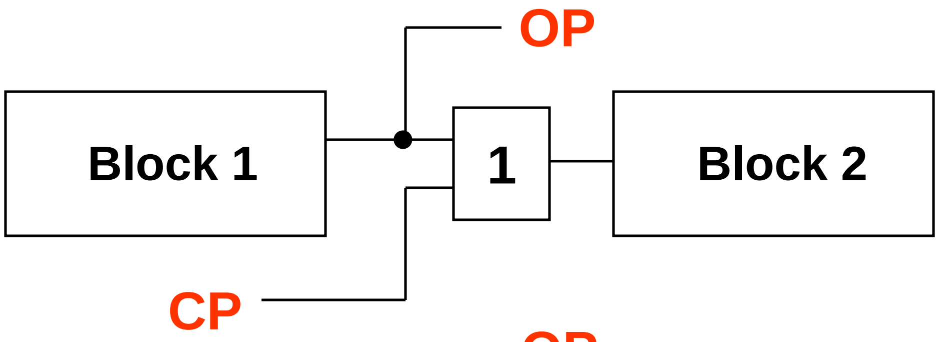
Ad Hoc Design for Testability Techniques

Method of Test Points:

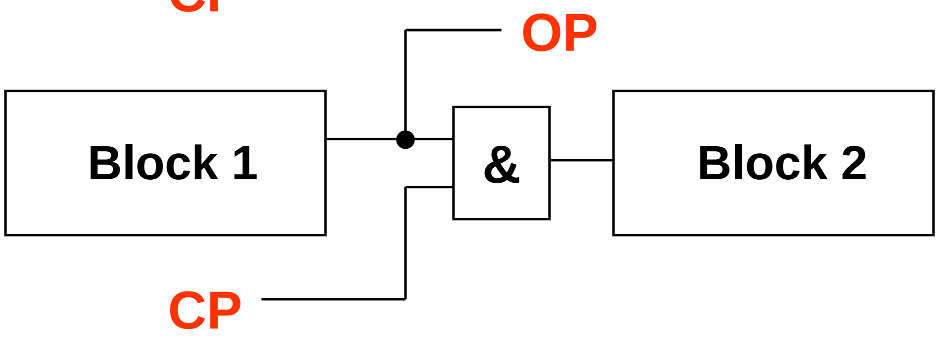


Block 1 is not observable,
Block 2 is not controllable

Improving controllability and observability:



1- controllability:
CP = 0 - normal working mode
CP = 1 - controlling Block 2 with signal 1



0- controllability:
CP = 1 - normal working mode
CP = 0 - controlling Block 2 with signal 0

Ad Hoc Design for Testability Techniques

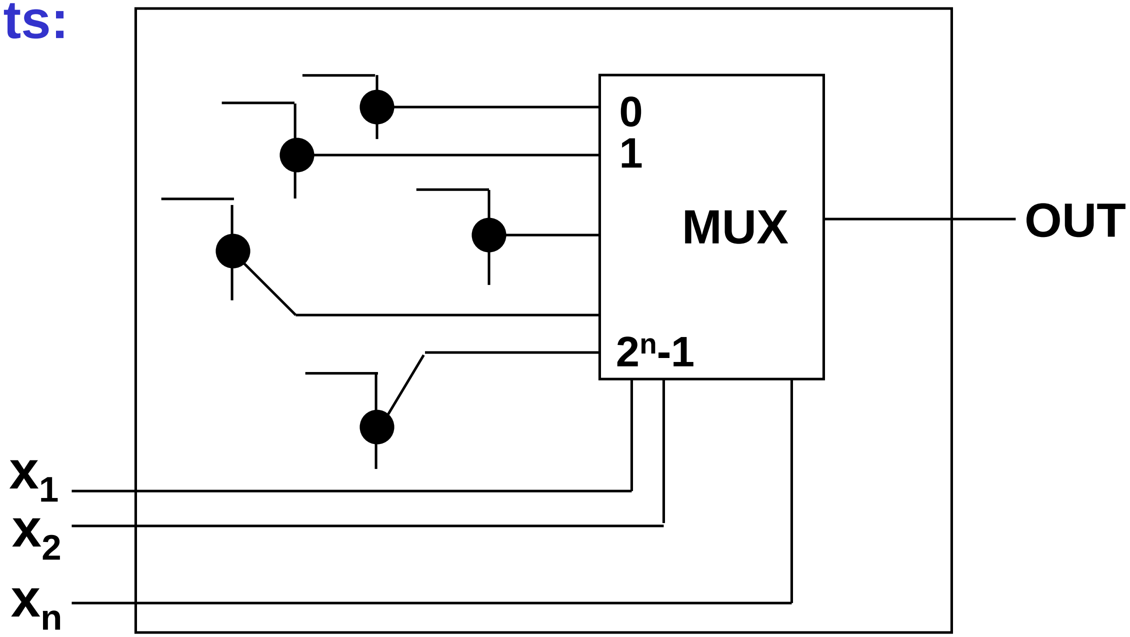
Multiplexing monitor points:

To reduce the number of output pins for observing monitor points, multiplexer can be used:

2^n observation points are replaced by a single output and n inputs to address a selected observation point

Disadvantage:

Only one observation point can be observed at a time



Number of additional pins: (n + 1)
Number of observable points: [2^n]

Advantage: $(n + 1) << 2^n$

Ad Hoc Design for Testability Techniques

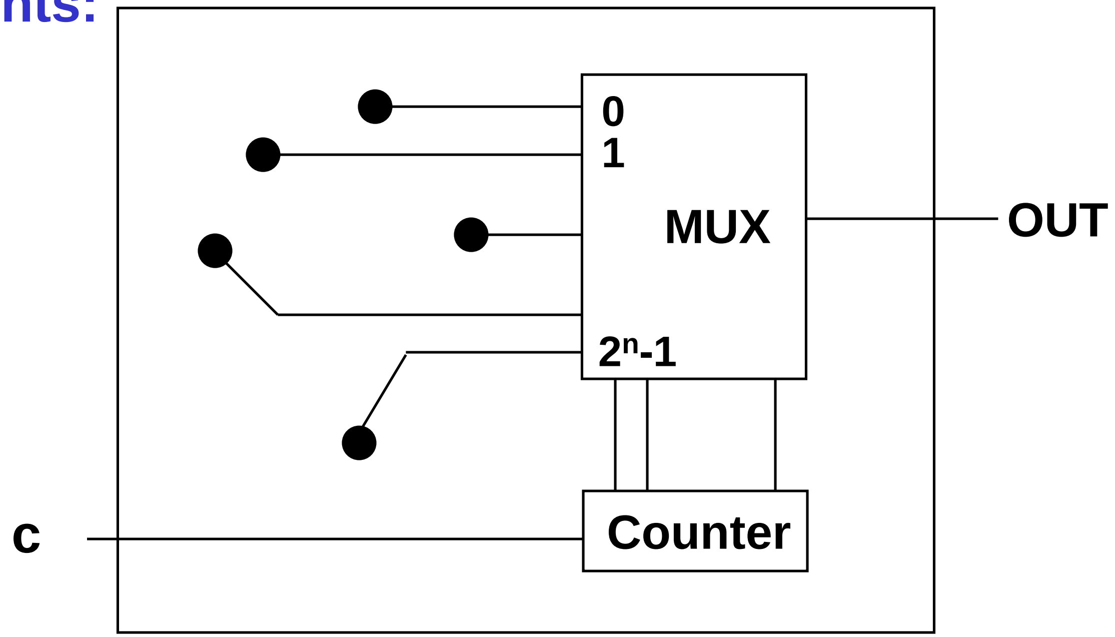
Multiplexing monitor points:

To reduce the number of output pins for observing monitor points, multiplexer can be used:

To reduce the number of inputs, a counter (or a shift register) can be used to drive the address lines of the multiplexer

Disadvantage:

Only one observation point can be observed at a time

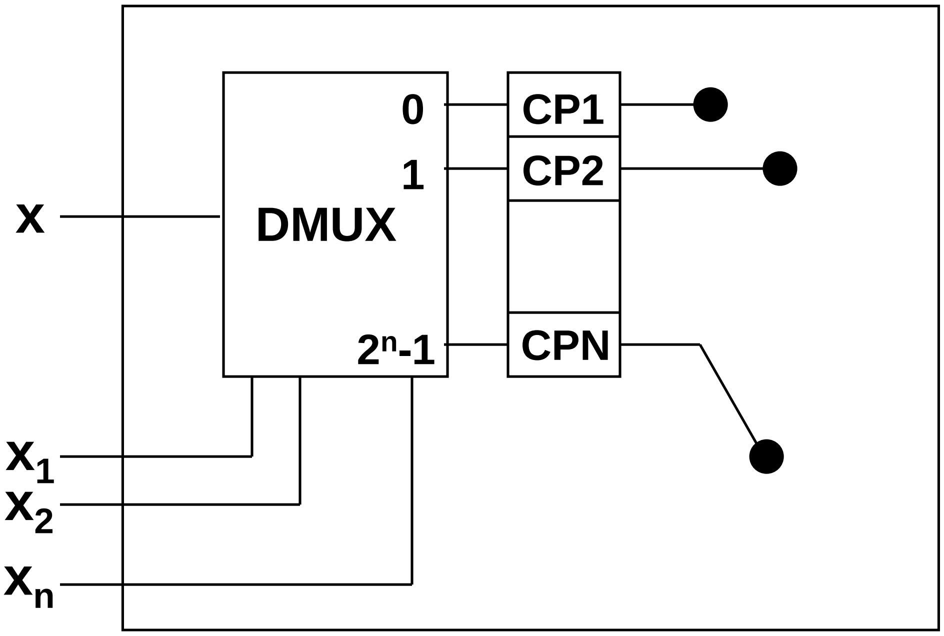


Number of additional pins: 2
Nmber of observable points: [2^n]

Advantage: $2 << 2^n$

Ad Hoc Design for Testability Techniques

Demultiplexer for implementing control points:



To reduce the number of input pins for controlling testpoints, demultiplexer and a latch register can be used.

Disadvantage:

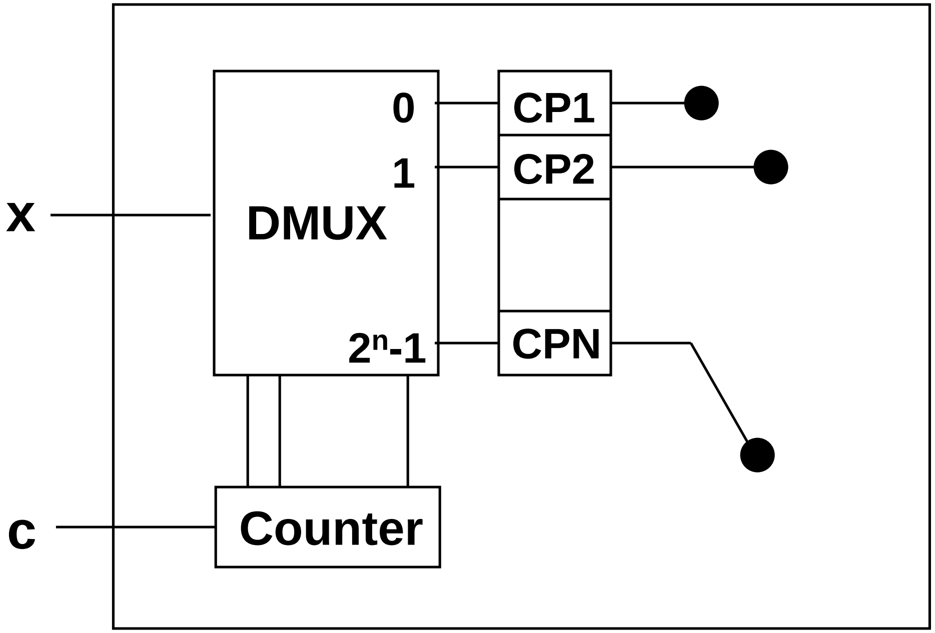
N clock times are required between test vectors to set up the proper control values

Number of additional pins: (n + 1)
Number of control points: $2^{n-1} < N \leq 2^n$

Advantage: (n + 1) << N

Ad Hoc Design for Testability Techniques

Demultiplexer for implementing control points:



Number of additional pins: 2
Number of control points: N

Advantage: 2 << N

To reduce the number of input pins for controlling testpoints, demultiplexer and a latch register can be used.

To reduce the number of inputs for addressing, a counter (or a shift register) can be used to drive the address lines of the demultiplexer

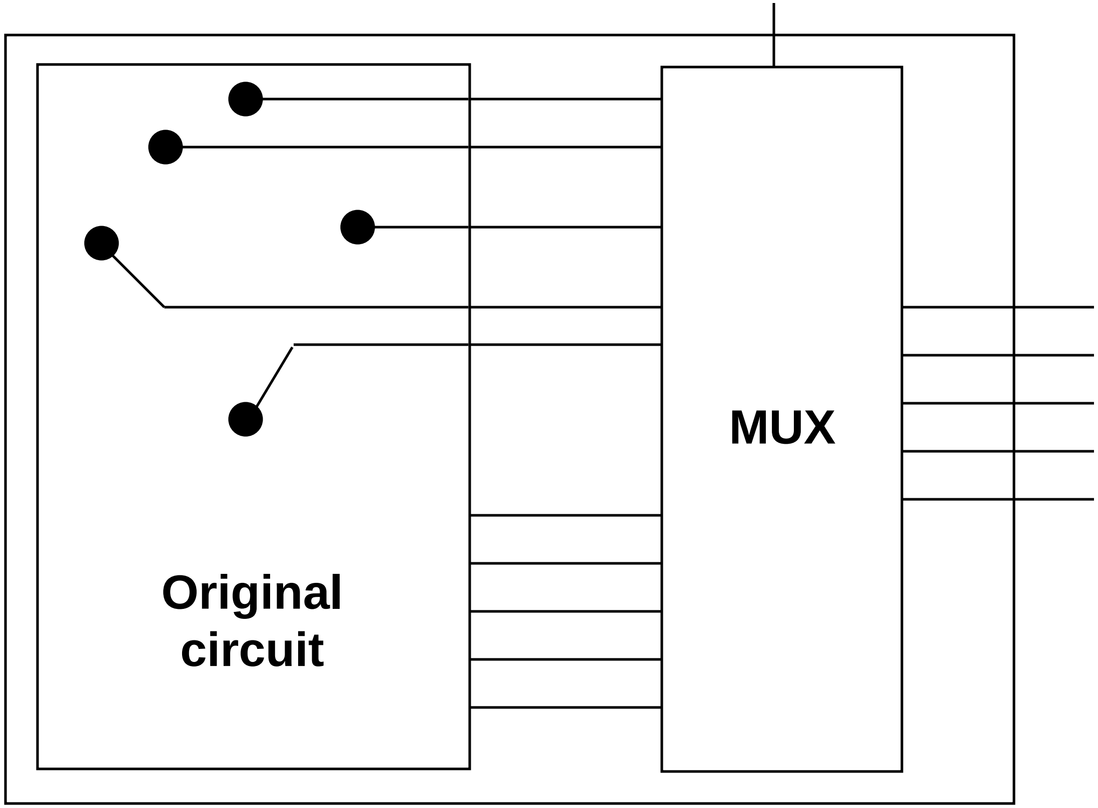
Disadvantage:

N clock times are required between test vectors to set up the proper control values

Time-sharing of outputs for monitoring

To reduce the number of output pins for observing monitor points, time-sharing of working outputs can be introduced: no additional outputs are needed

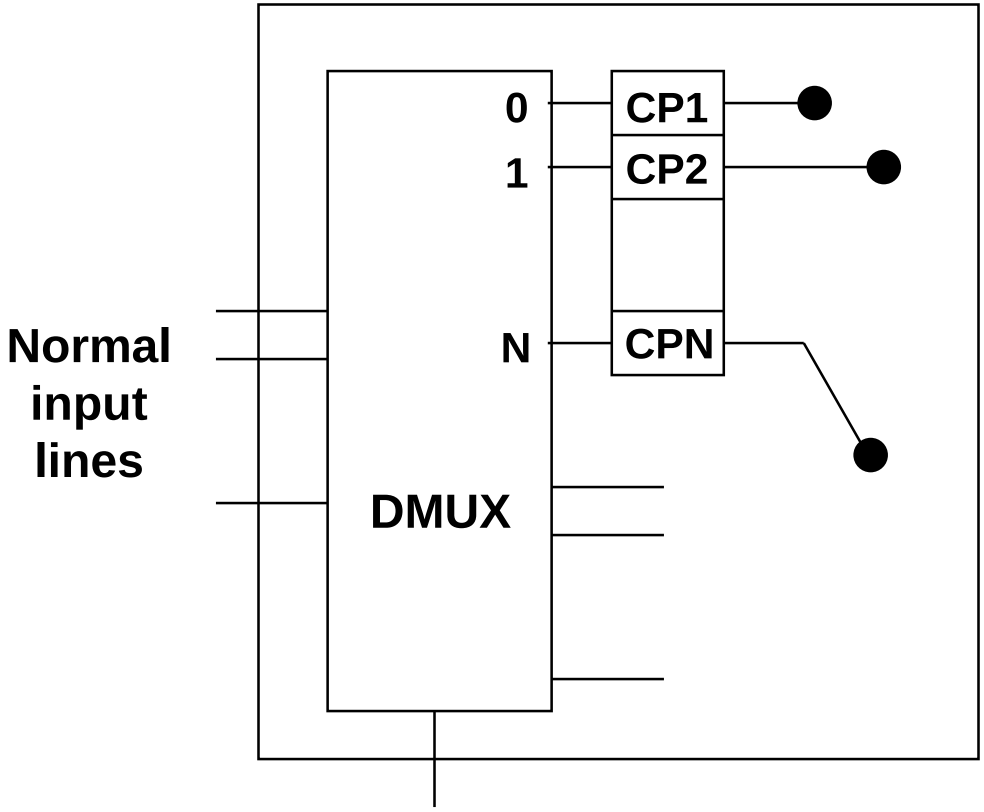
To reduce the number of inputs, again counter or shift register can be used if needed



Number of additional pins: 1
Number of control points: N

Advantage: 1 << N

Time-sharing of inputs for controlling



To reduce the number of input pins for controlling test points, time-sharing of working inputs can be introduced.

To reduce the number of inputs for driving the address lines of demultiplexer, counter or shift register can be used if needed

Number of additional pins: 1
Number of control points: N

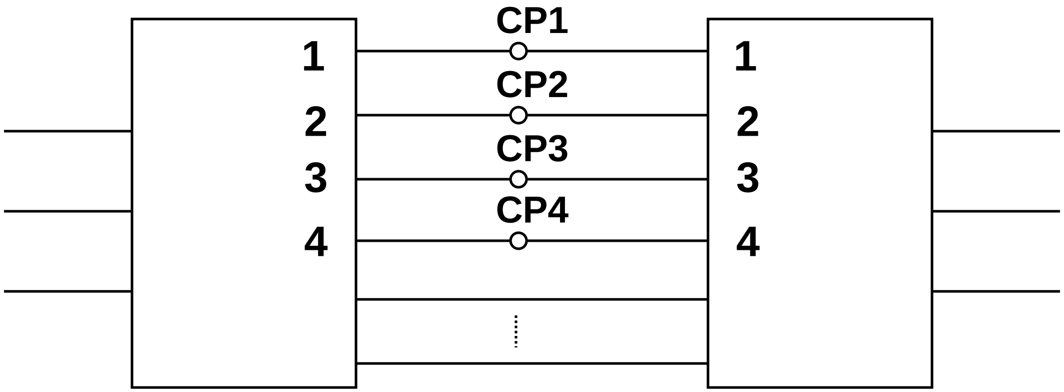
Advantage: 1 << N

Example: DFT with MUX-s and DMUX-s

Given a circuit:

- CP1 and CP2 are not controllable
- CP3 and CP4 are not observable

DFT task: Improve the testability by using a single control input, no additional inputs/outputs allowed



Example: DFT with MUX-s and DMUX-s

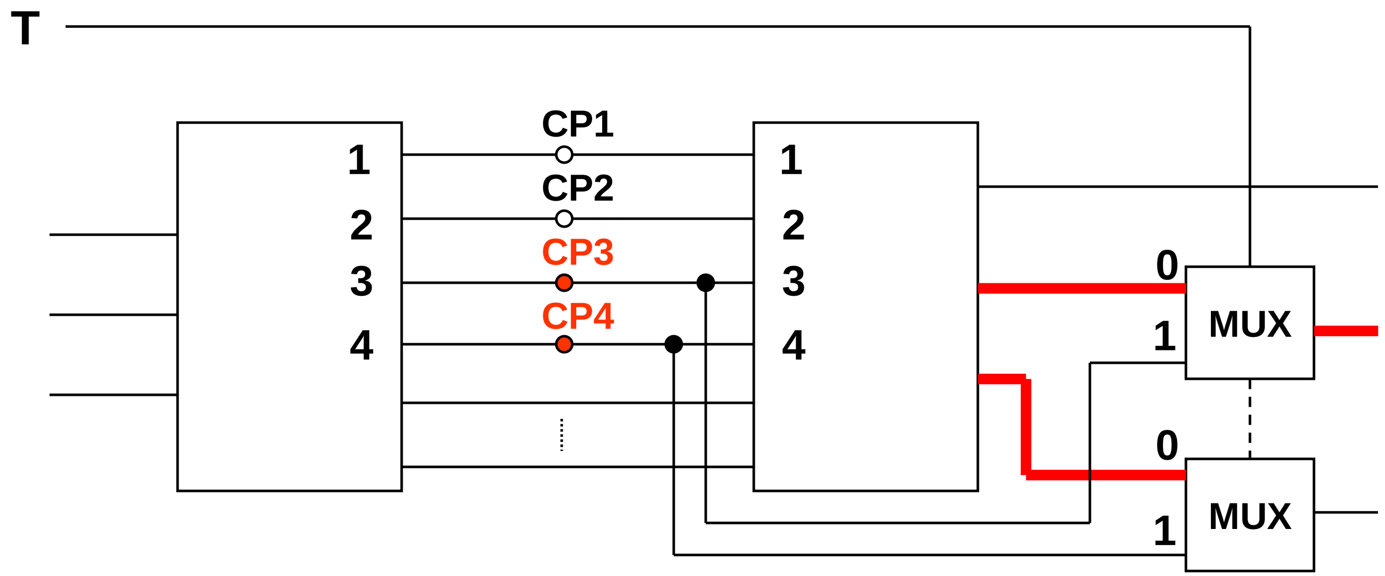
Given a circuit:

CP3 and CP4 are not observable

→ Improving the observability

Coding:

T	Mode	MUX
0	Norm.	0
1	Test	1



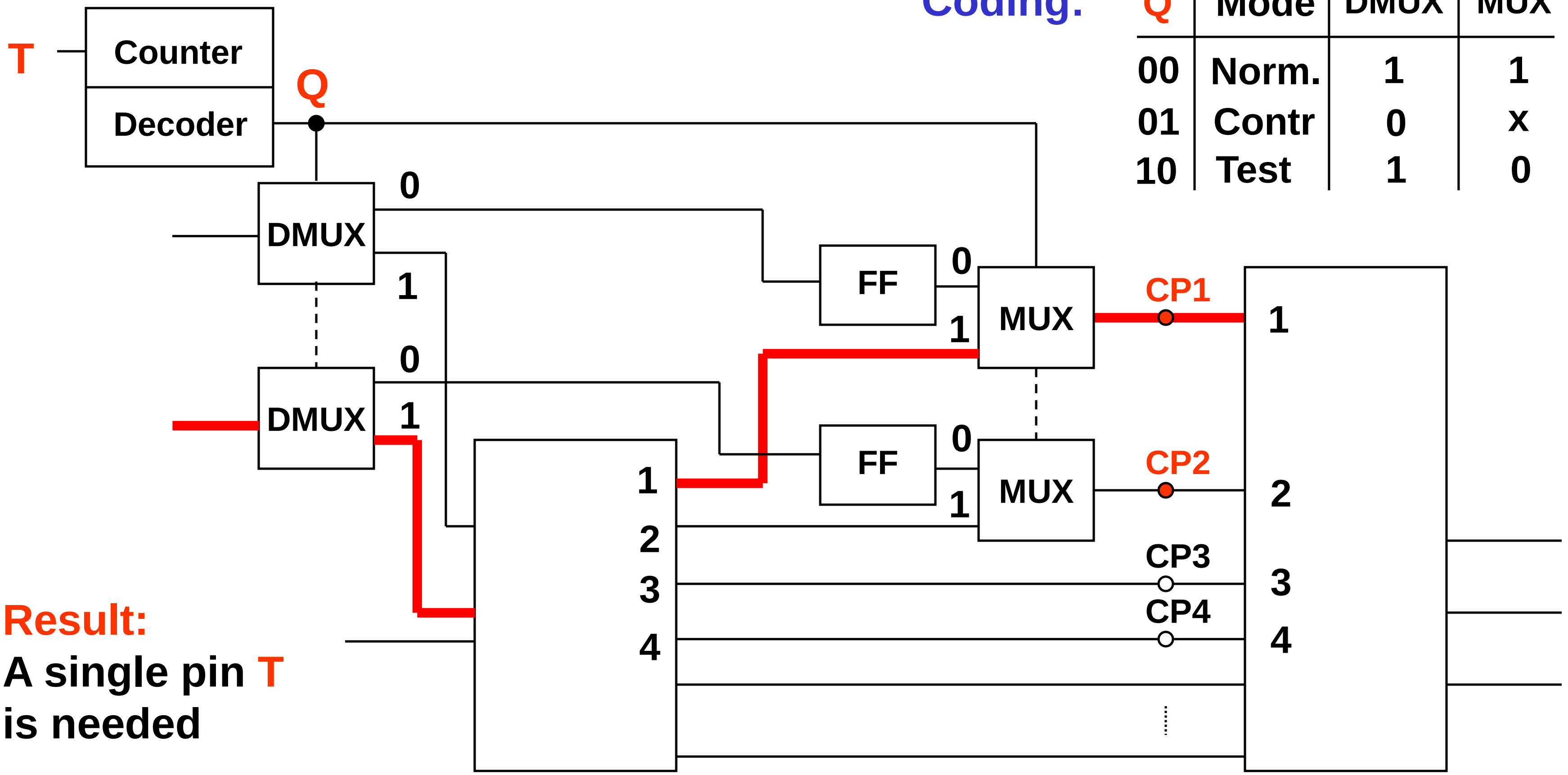
Result: **A single pin T is needed**

Example: DFT with MUX-s and DMUX-s

Given a circuit: CP1 and CP2 are not controllable → Improving the controllability

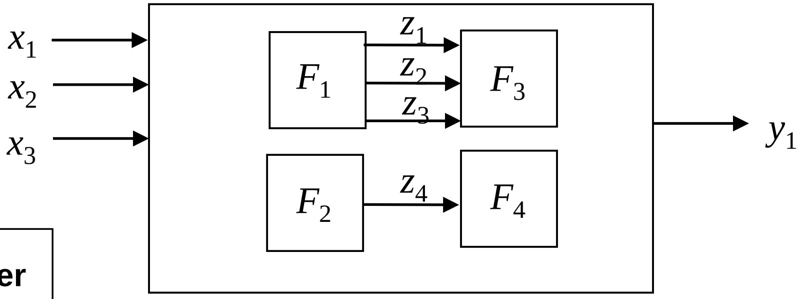
Coding:

Q	Mode	DMUX	MUX
00	Norm.	1	1
01	Contr	0	x
10	Test	1	0

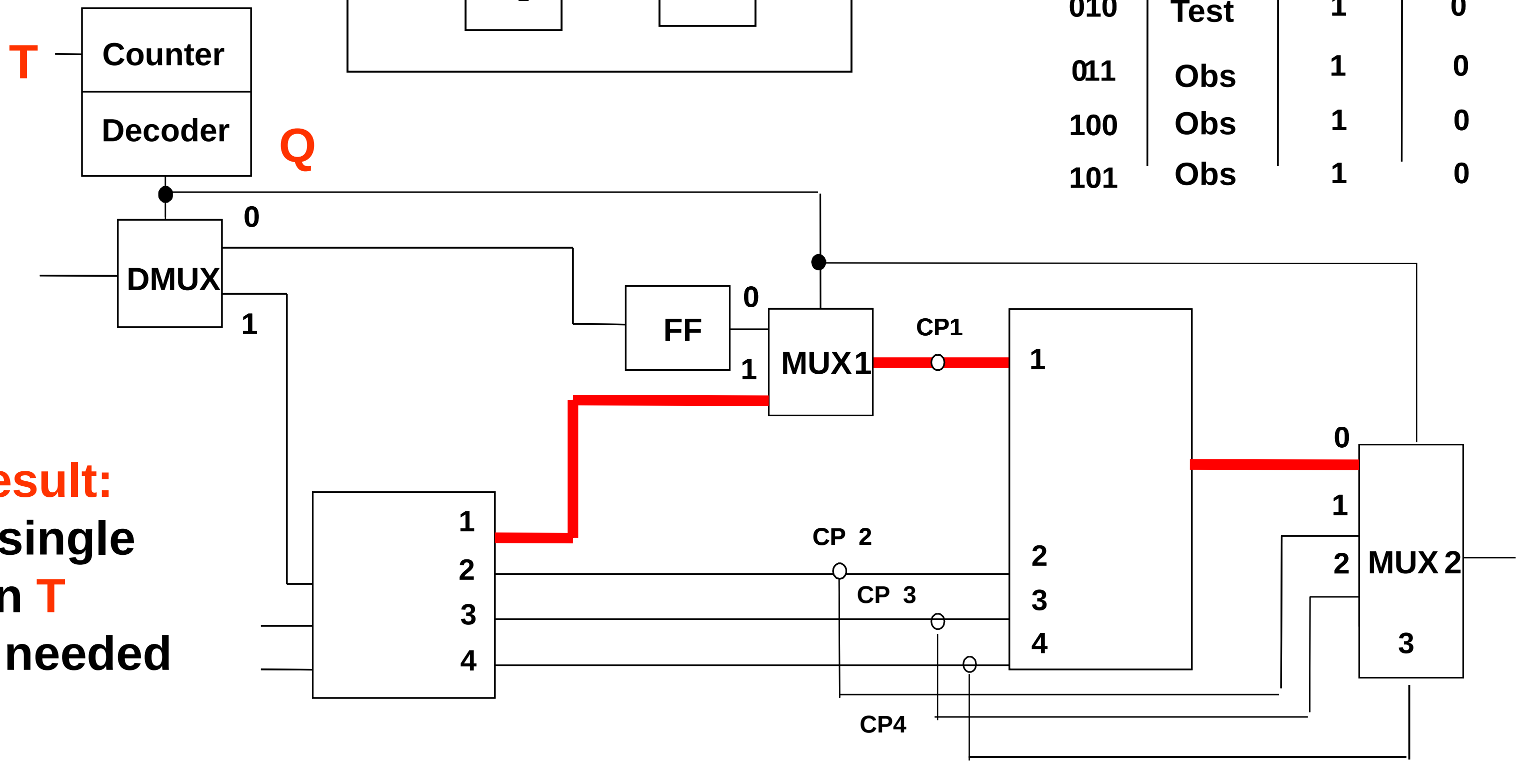


Result:
A single pin T
is needed

Example: DFT with MUX-s and DMUX-s



Q	Mode	DMUX	MUX 1	MUX 2
000	Norm	1	1	0
001	Contr	0	x	x
010	Test	1	0	0
011	Obs	1	0	1
100	Obs	1	0	2
101	Obs	1	0	3



Result:
A single pin T is needed

Ad Hoc Design for Testability Techniques

Logical redundancy:

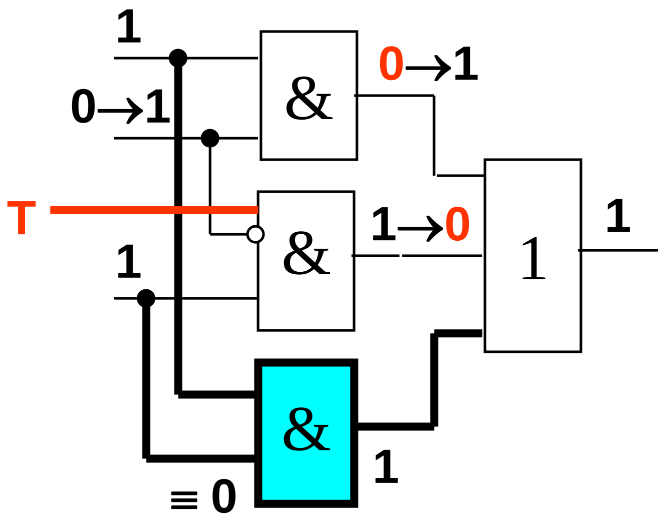
Redundancy should be avoided:

- If a redundant fault occurs, it may invalidate some test for nonredundant faults
- Redundant faults cause difficulty in calculating fault coverage
- Much test generation time can be spent in trying to generate a test for a redundant fault

Redundancy intentionally added:

- To eliminate hazards in combinational circuits
- To achieve high reliability (using error detecting circuits)

Hazard control circuitry:



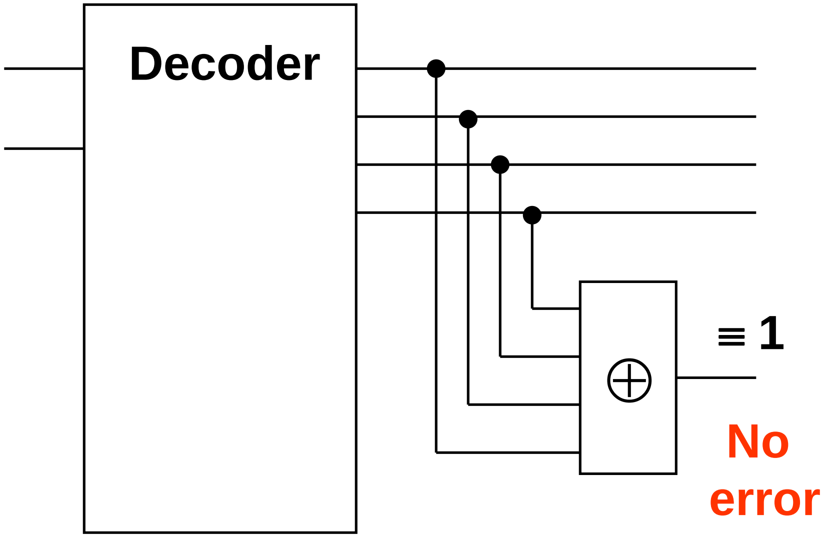
Redundant AND-gate
Fault ≡ 0 not testable

Additional control input added:
T = 1 - normal working mode
T = 0 - testing mode

Ad Hoc Design for Testability Techniques

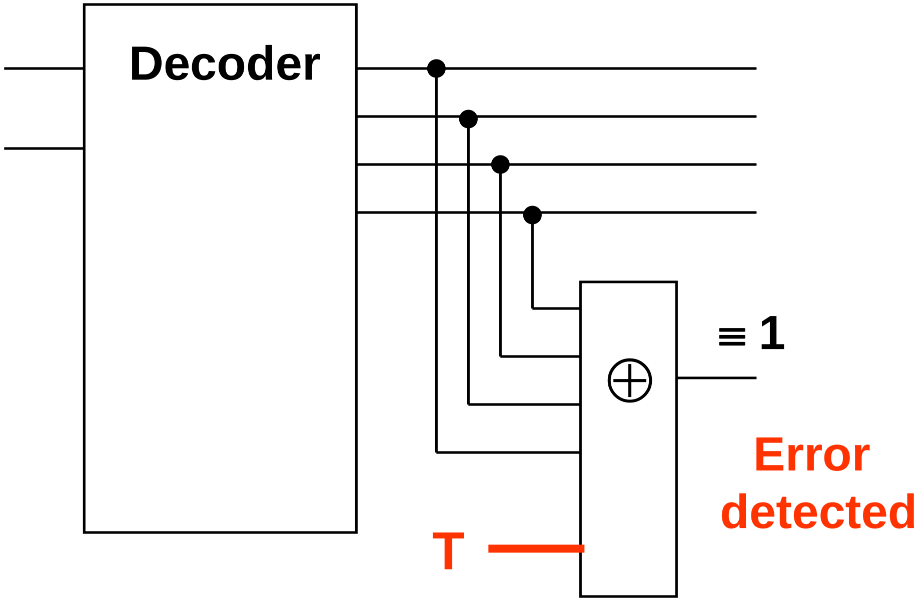
Fault redundancy:

Error control circuitry:



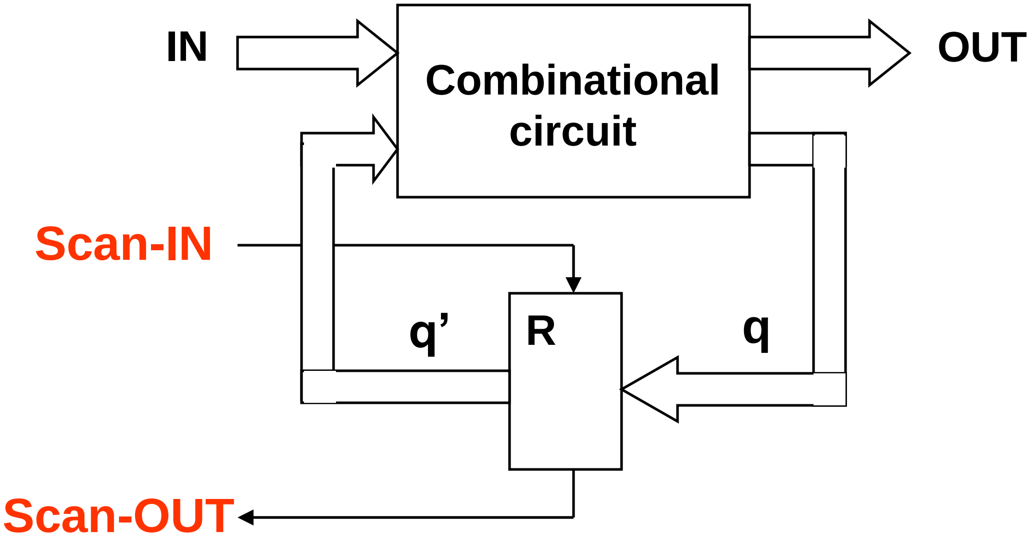
E = 1 if decoder is fault-free
Fault ≡ 1 not testable

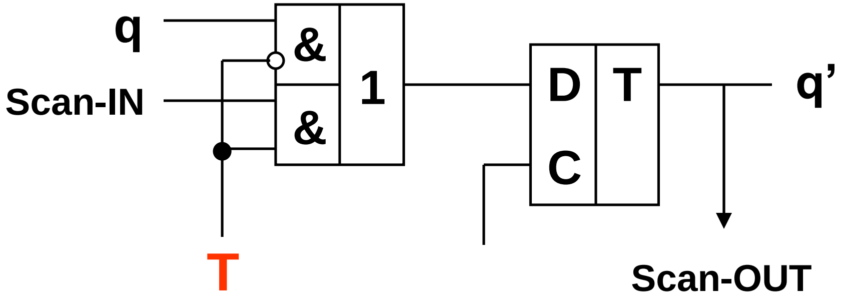
Testable error control circuitry:



Additional control input added:
T ≡ 0 - normal working mode
T = 1 - testing mode

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

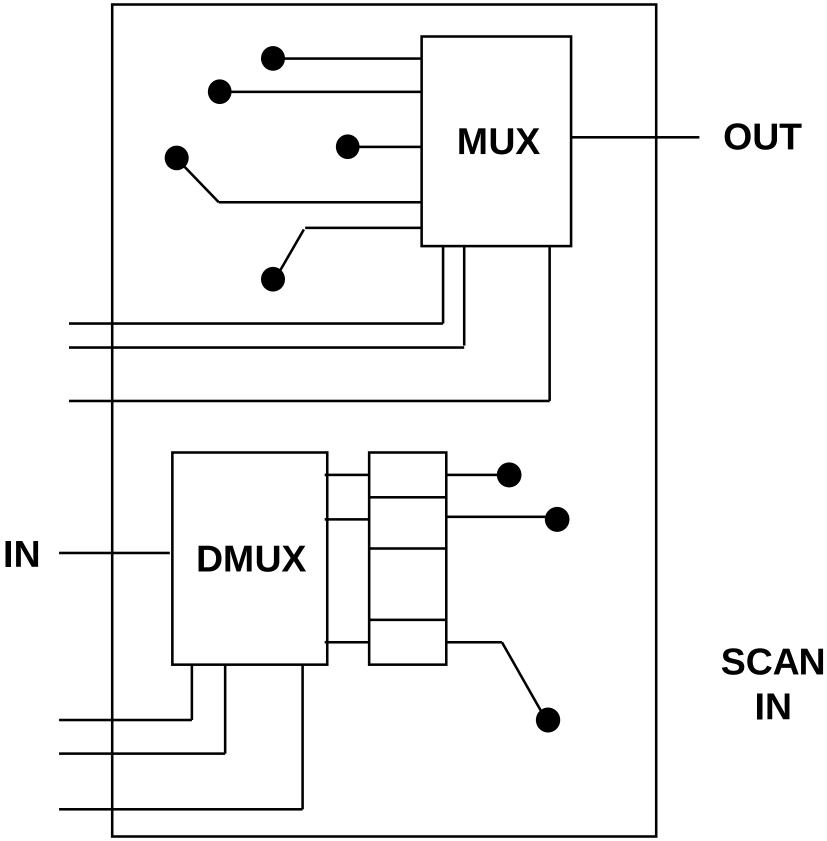
T = 1 - scan mode

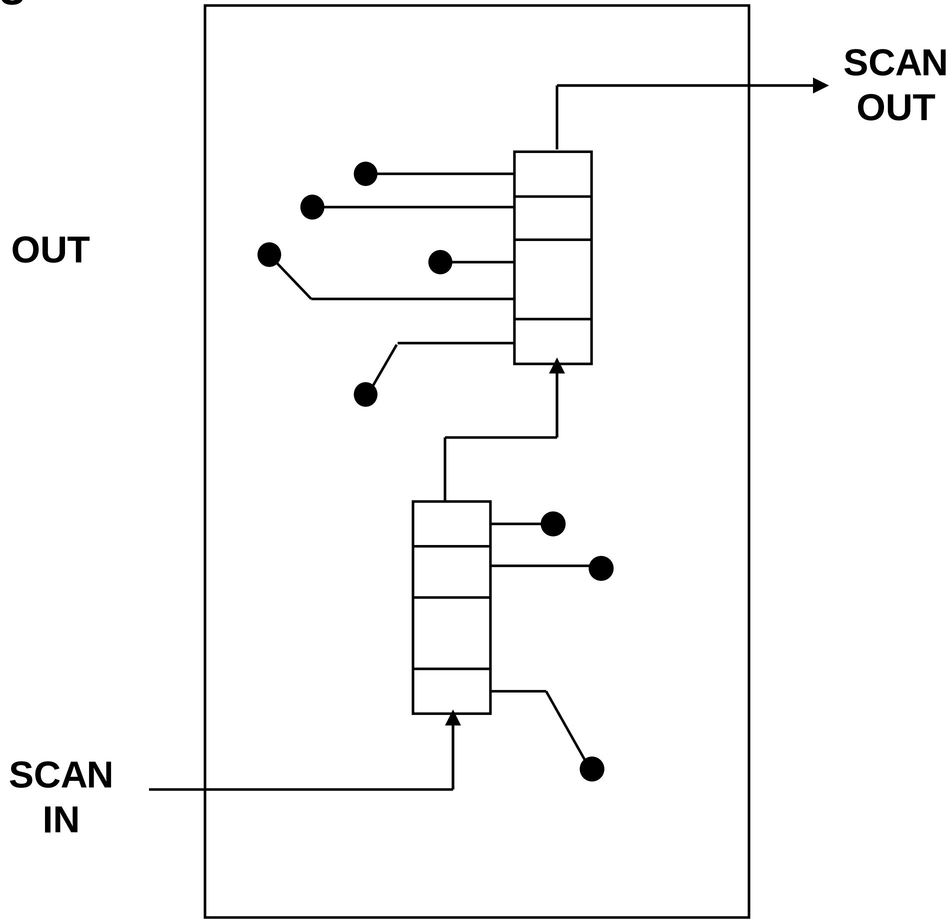
Normal mode : flip-flops are connected to the combinational circuit

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

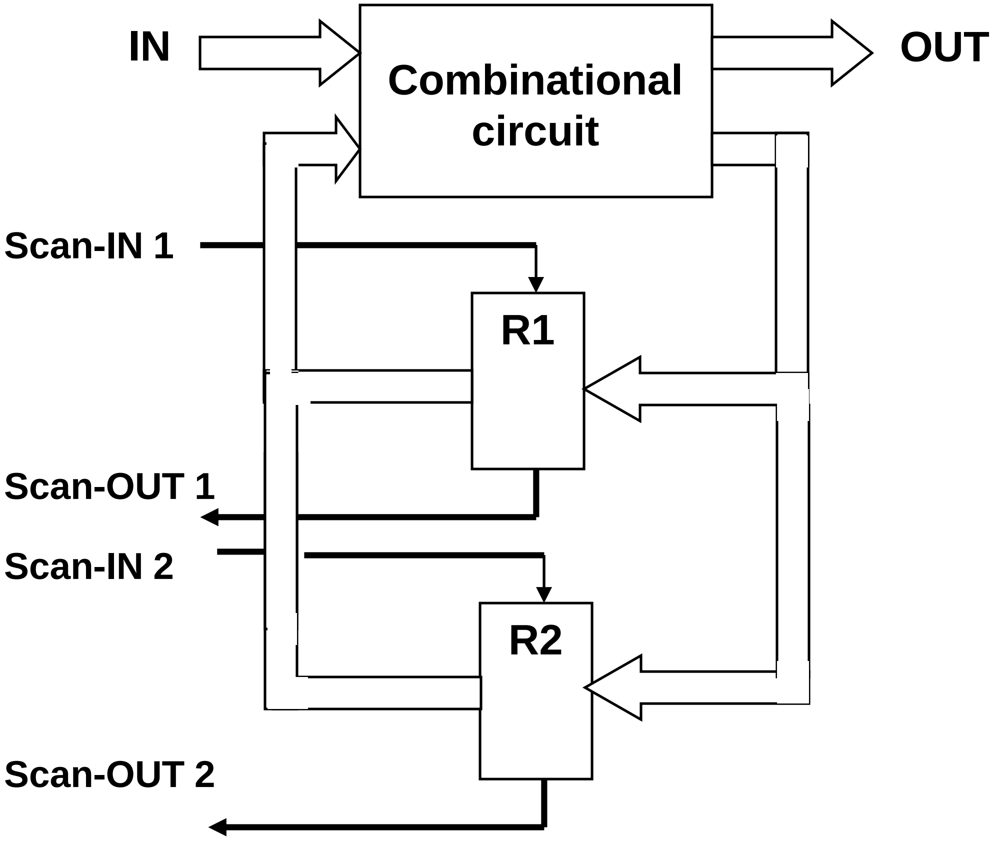
Scan-Path Design and Testability

Two possibilities for improving controllability/observability





Parallel Scan-Path

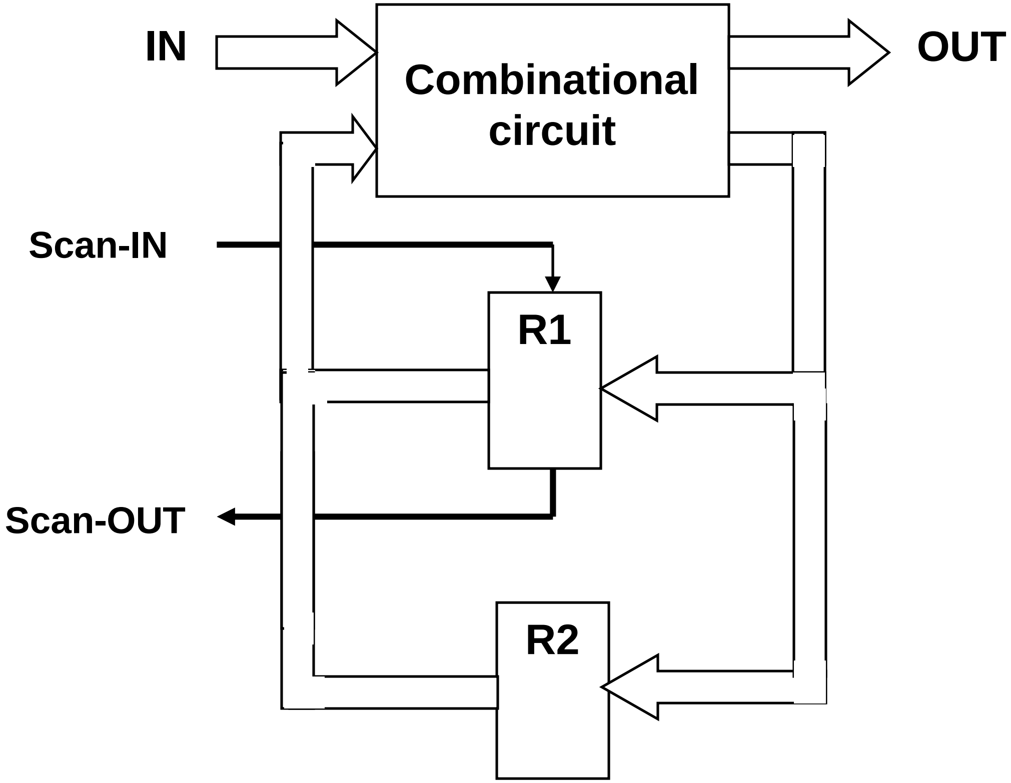


In **parallel scan path** flip-flops can be organized in more than one scan chain

Advantage: **time ↓**

Disadvantage: **# pins ↑**

Partial Scan-Path

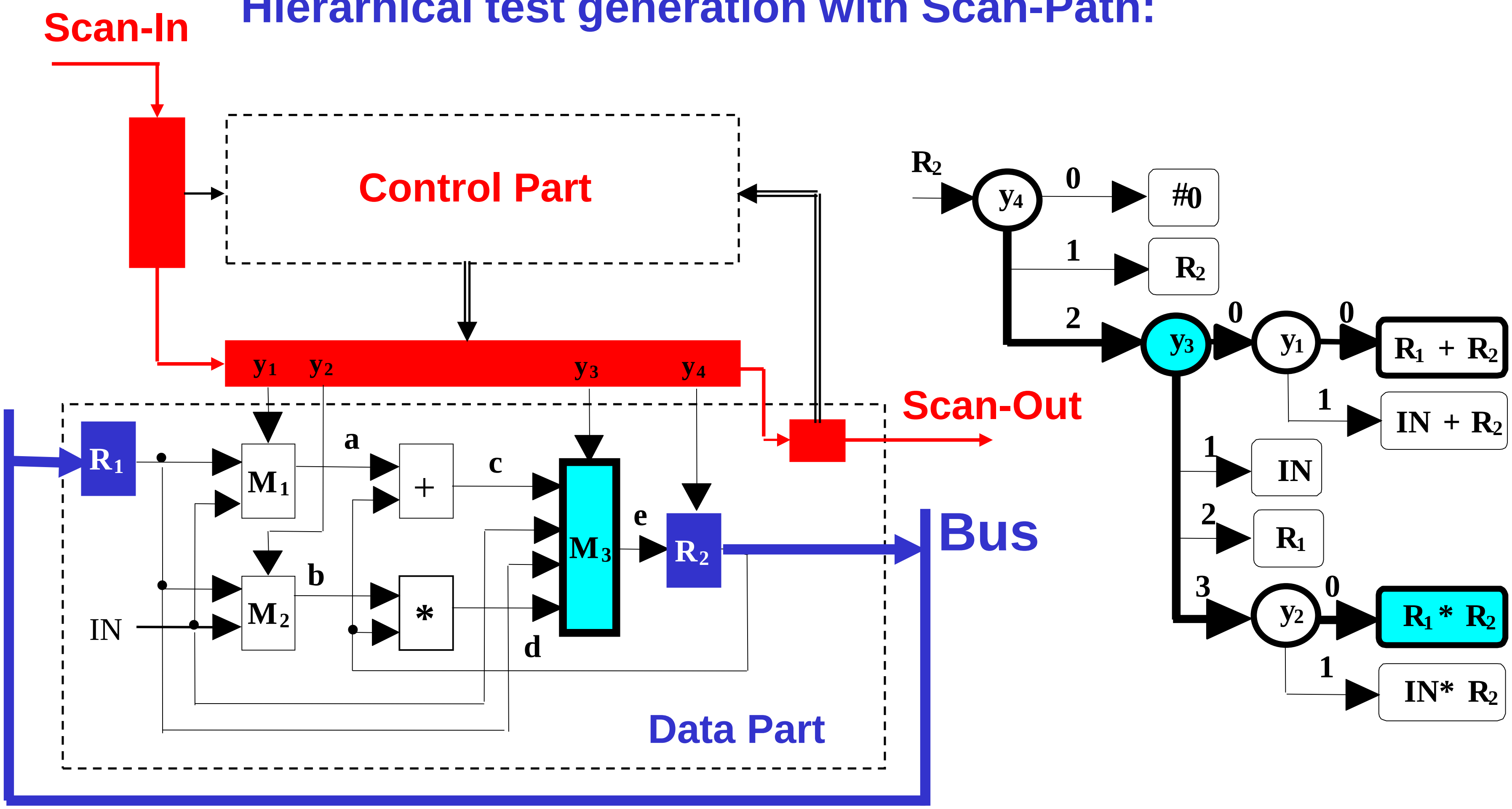


In **partial scan** instead of full–scan, it may be advantageous to scan **only some** of the flip–flops

Example: **counter** – even bits joined in the scan–register

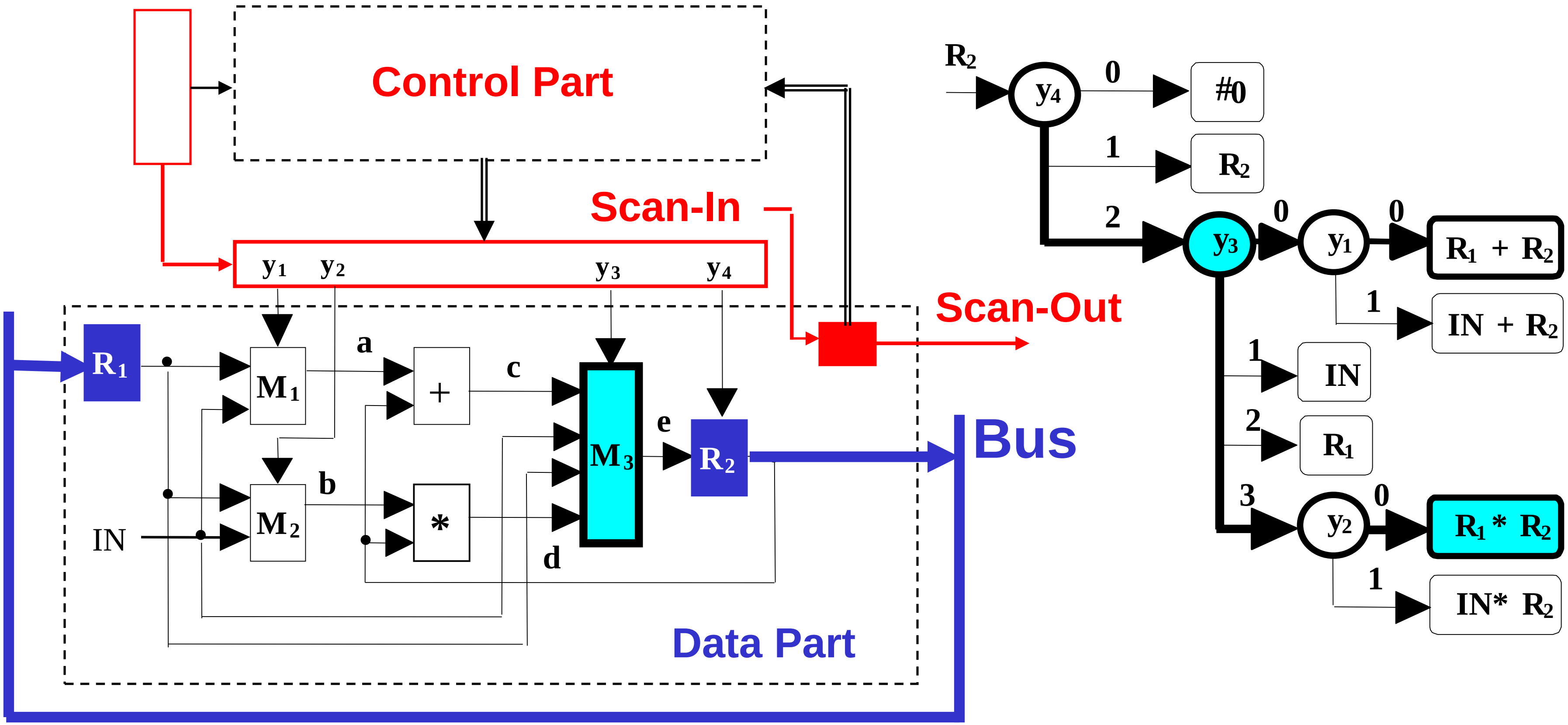
Partial Scan Path

Hierarhical test generation with Scan-Path:

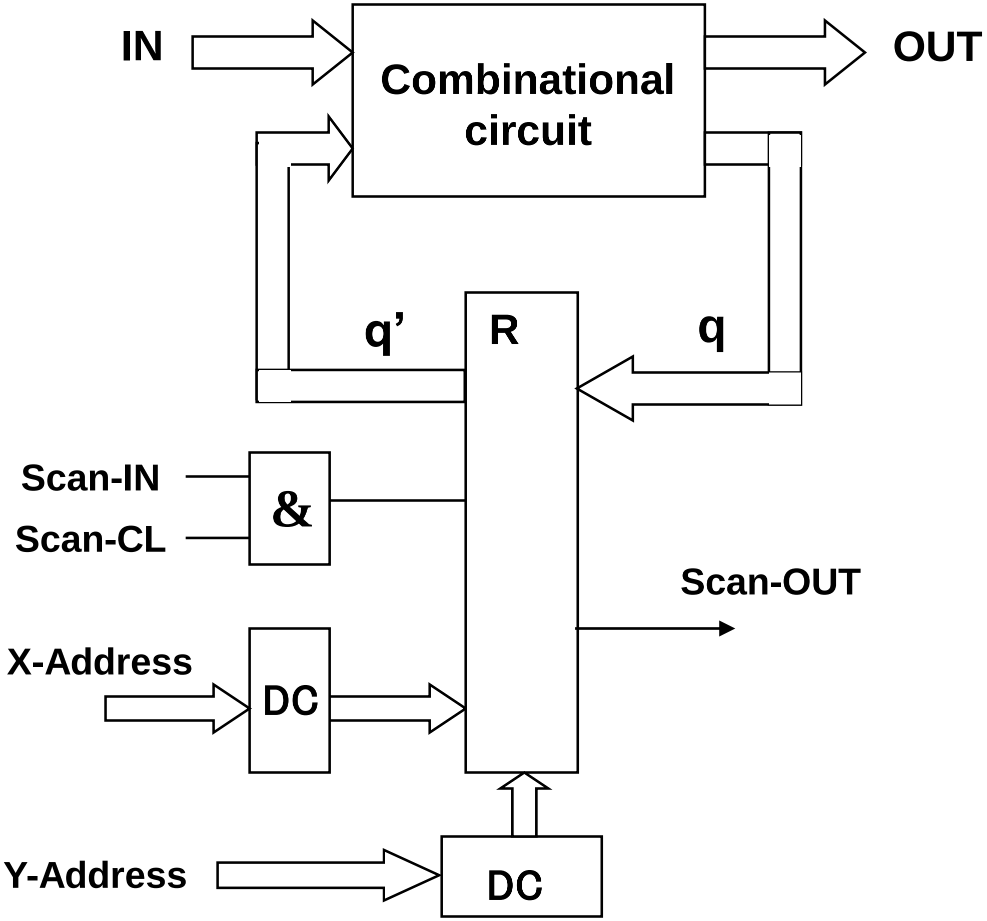


Testing with Minimal DFT

Hierarhical test generation with Scan-Path:



Random Access Scan



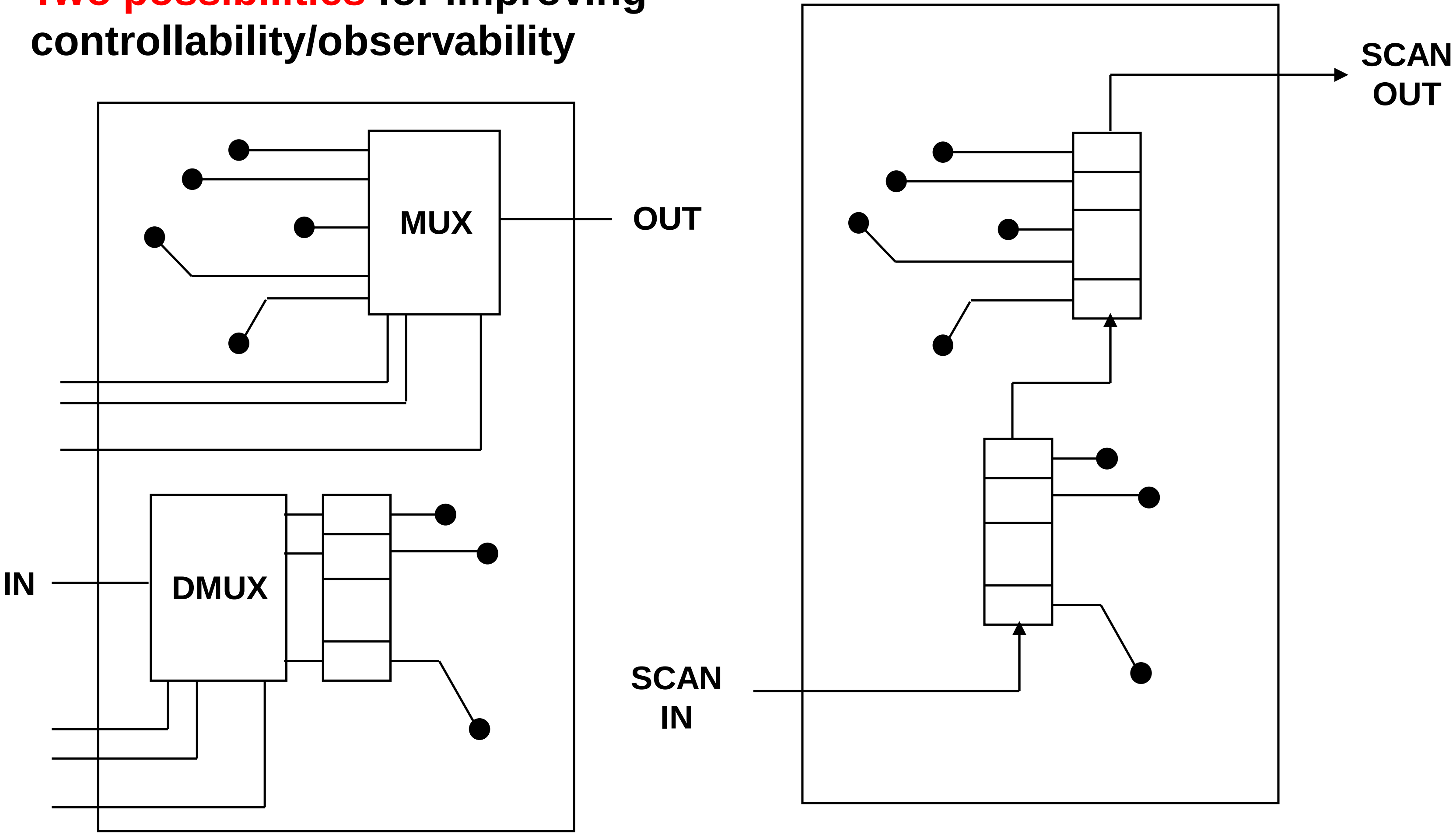
In random access scan each flip-flop in a logic network is selected individually by an address for control and observation of its state

Example:

Delay fault testing

Improving Testability by Inserting CPs

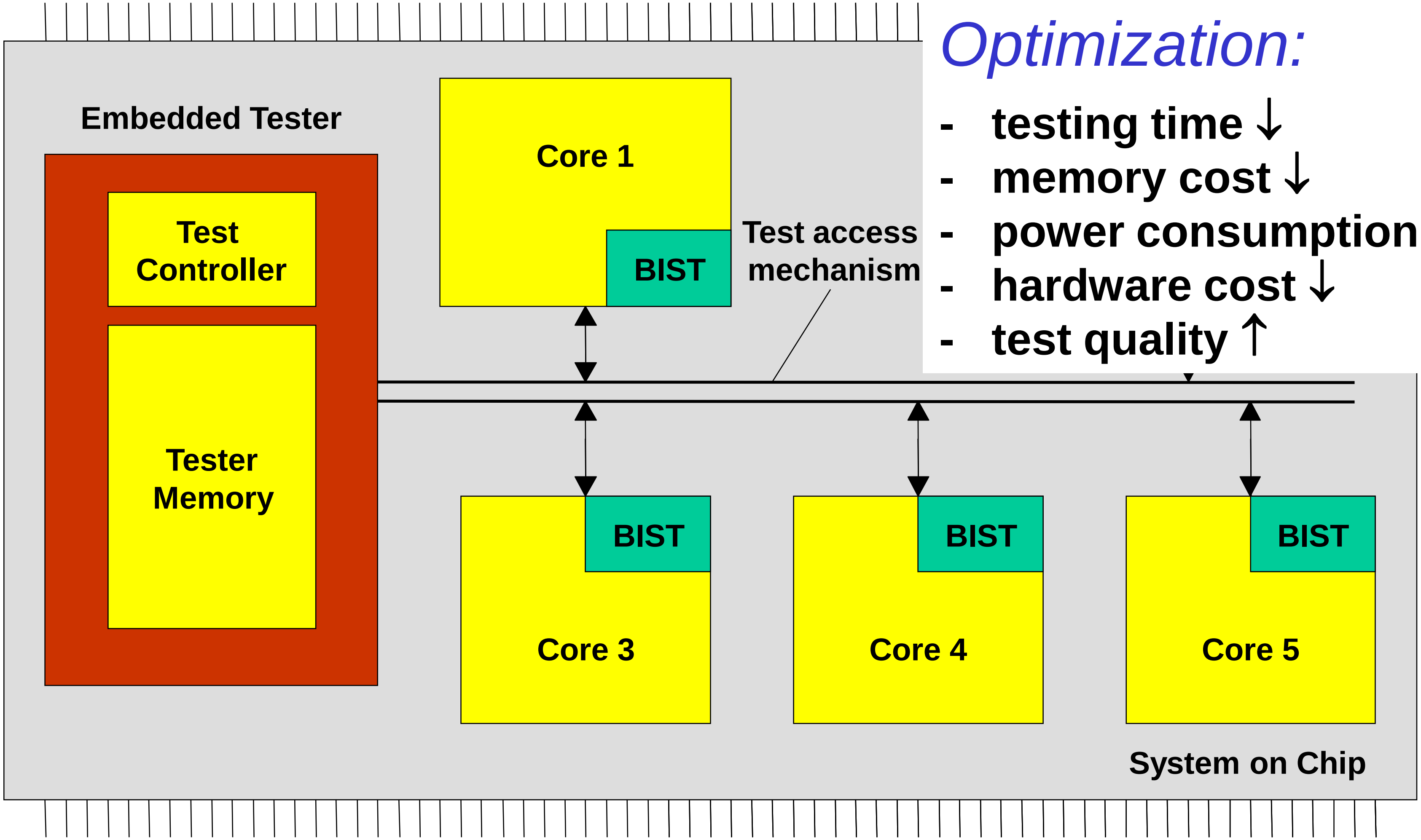
Two possibilities for improving controllability/observability



Built-In Self-Test

- **Motivations for BIST:**
  - **Need for a cost-efficient testing (general motivation)**
  - **Doubts about the stuck-at fault model**
  - **Increasing difficulties with TPG (Test Pattern Generation)**
  - **Growing volume of test pattern data**
  - **Cost of ATE (Automatic Test Equipment)**
  - **Test application time**
  - **Gap between tester and UUT (Unit Under Test) speeds**
- **Drawbacks of BIST:**
  - **Additional pins and silicon area needed**
  - **Decreased reliability due to increased silicon area**
  - **Performance impact due to additional circuitry**
  - **Additional design time and cost**

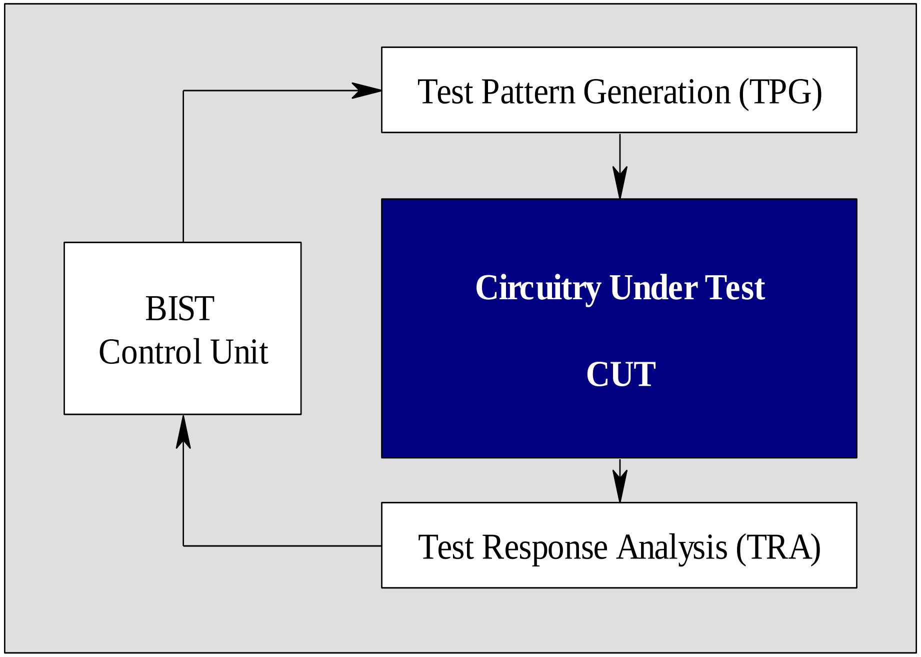
SoC BIST



Optimization:

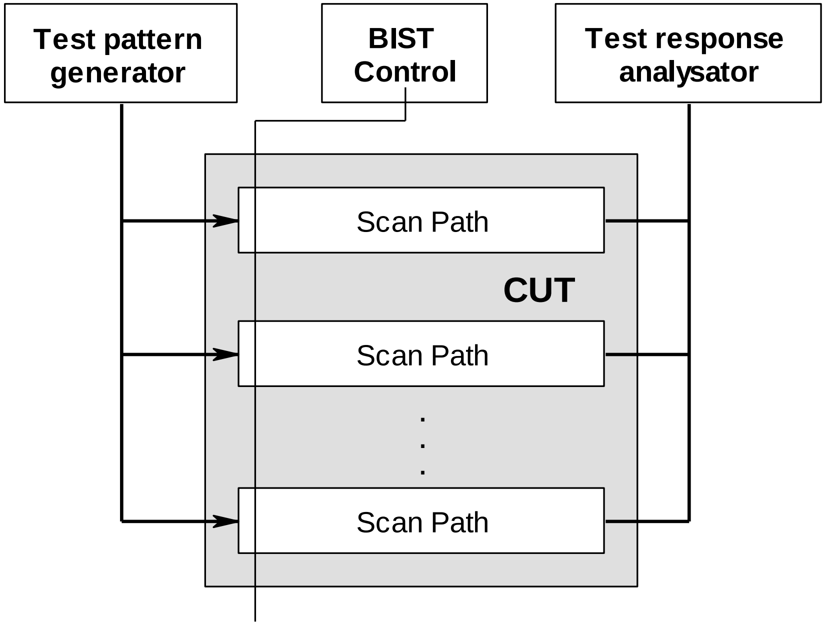
- testing time ↓
- memory cost ↓
- power consumption ↓
- hardware cost ↓
- test quality ↑

General Architecture of BIST



- BIST components:
  - Test pattern generator (TPG)
  - Test response analyzer (TRA)
- TPG & TRA are usually implemented as linear feedback shift registers (LFSR)
- Two widespread schemes:
  - test-per-scan
  - test-per-clock

Built-In Self-Test



- **Assumes existing scan architecture**
- **Drawback:**
  - **Long test application time**

Test per Scan:

Initial test set:

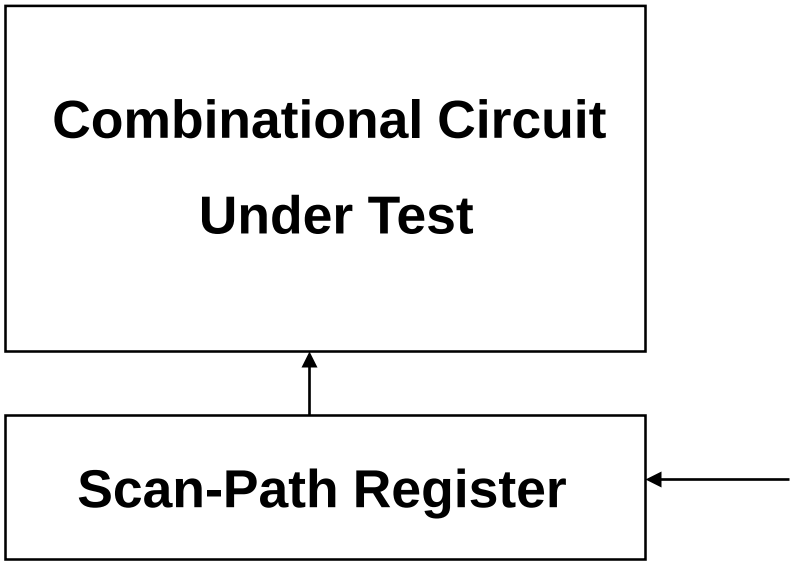
T1: 1100
T2: 1010
T3: 0101
T4: 1001

Test application:

1100 T 1010 T 0101T 1001 T
Number of clocks = (4 x 4) + 4 = 20

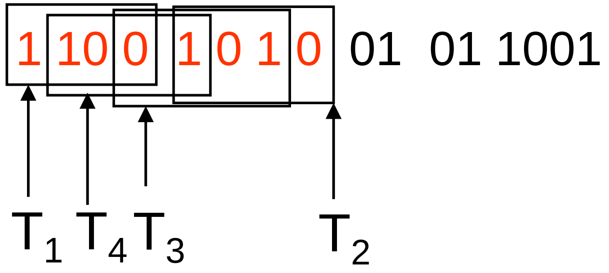
Built-In Self-Test

Test per Clock:



- **Initial test set:**
- T1: 1100
- T2: 1010
- T3: 0101
- T4: 1001

- **Test application:**

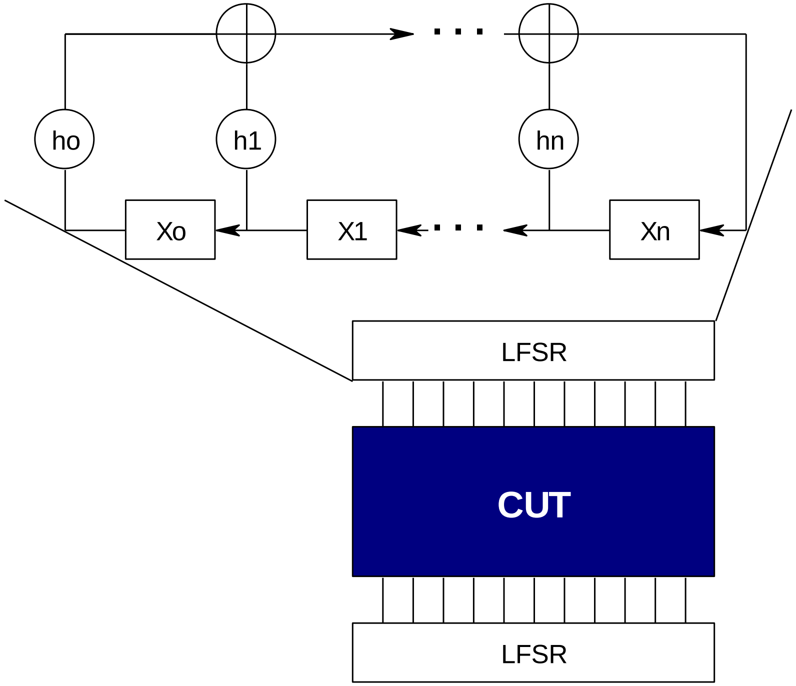


- 1 10 0 1 0 1 0 01 01 1001

- Number of clocks = 8 < 20

Pattern Generation

Pseudorandom test generation by LFSR:

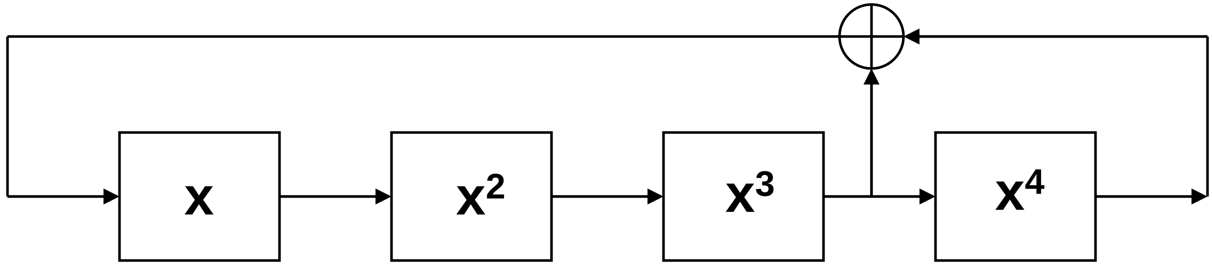


- **Using special LFSR registers**
  - Test pattern generator
  - Signature analyzer
- **Several proposals:**
  - BILBO
  - CSTP
- **Main characteristics of LFSR:**
  - polynomial
  - initial state
  - test length

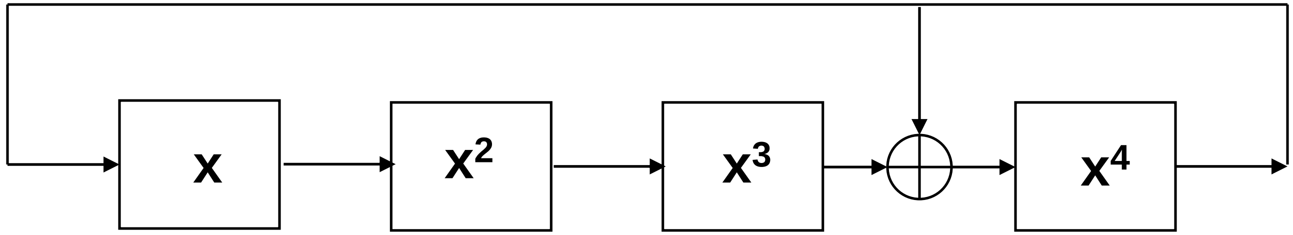
Pseudorandom Test Generation

LFSR – Linear Feedback Shift Register:

Standard LFSR



Modular LFSR

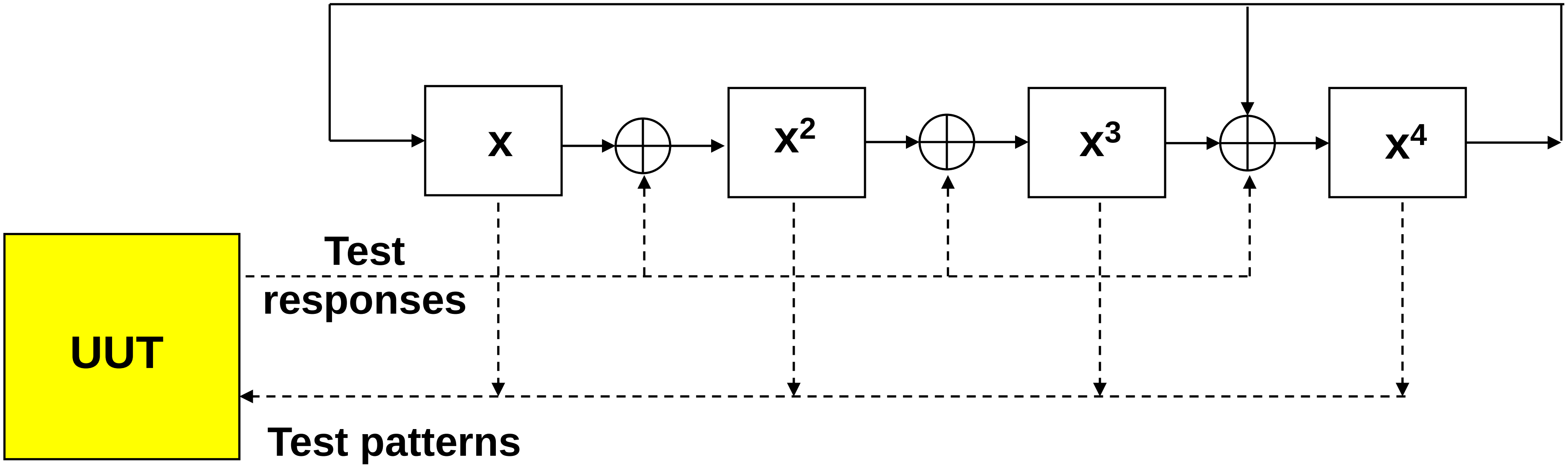


Polynomial: $P(x) = x^4 + x^3 + 1$

Pseudorandom Test Generation

LFSR – Linear Feedback Shift Register:

Why modular LFSR is useful for BIST?



Polynomial: $P(x) = x^4 + x^3 + 1$

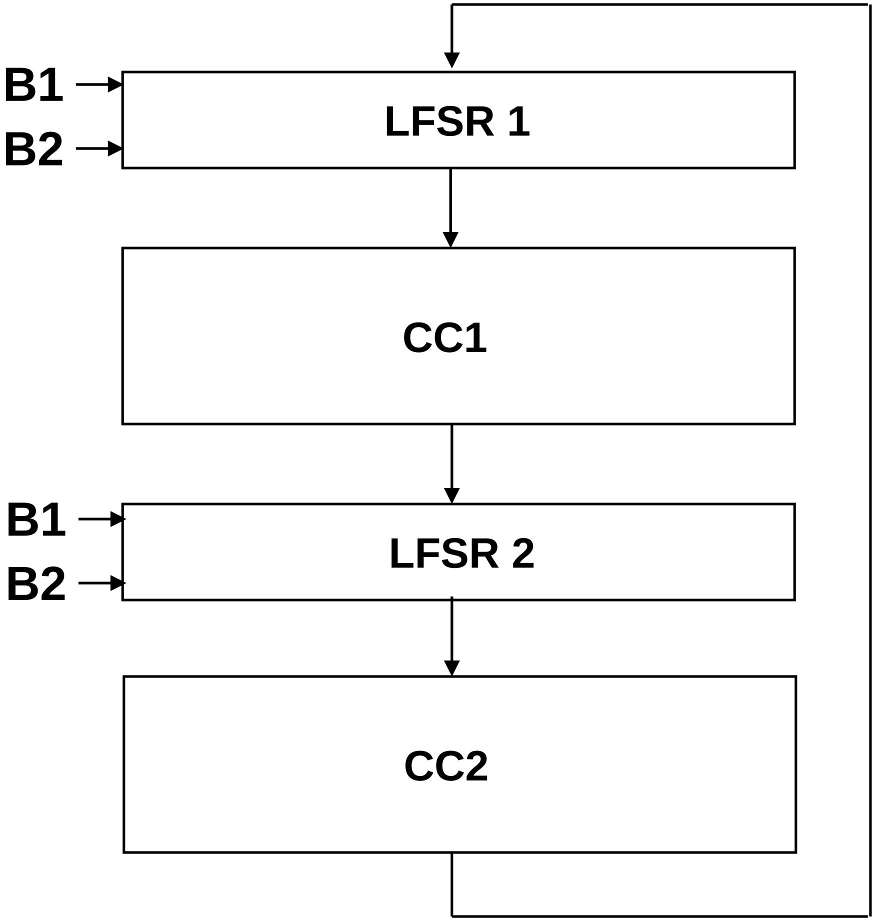
BILBO BIST Architecture

Working modes:

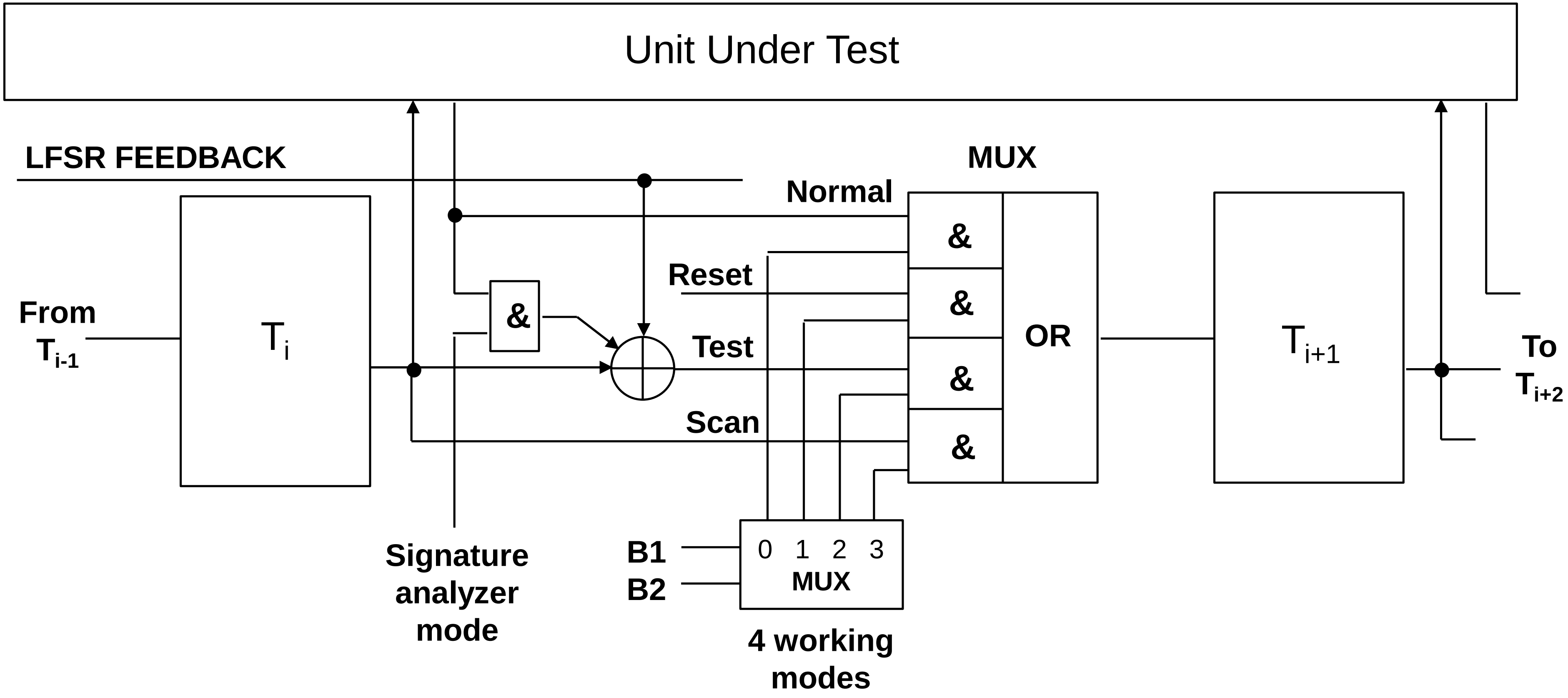
B1	B2	
0	0	Normal mode
0	1	Reset
1	0	Test mode
1	1	Scan mode

Testing modes:

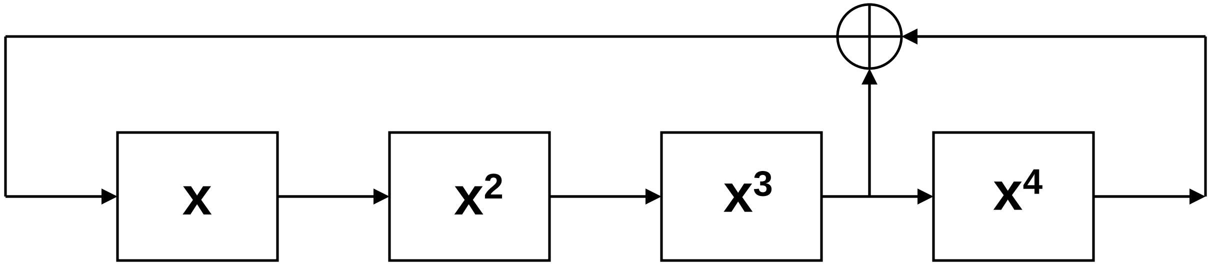
CC1:	LFSR 1	-	TPG
	LFSR 2	-	SA
CC2:	LFSR 2	-	TPG
	LFSR 1	-	SA



Reconfiguration of the LFSR



Pseudorandom Test Generation



Polynomial: $P(x) = x^4 + x^3 + 1$

$$\begin{pmatrix} X_4\,(t+1) \\ X_3\,(t+1) \\ X_2\,(t+1) \\ X_1\,(t+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & \underset{\underset{1}{\uparrow}}{h_3} & \underset{\underset{0}{\uparrow}}{h_2} & \underset{\underset{0}{\uparrow}}{h_1} \end{pmatrix} \begin{pmatrix} X_4\,(t) \\ X_3\,(t) \\ X_2\,(t) \\ X_1\,(t) \end{pmatrix}$$

t	x	x^2	x^3	x^4	t	x	x^2	x^3	x^4
1	0	0	0	1	9	0	1	0	1
2	1	0	0	0	10	1	0	1	0
3	0	1	0	0	11	1	1	0	1
4	0	0	1	0	12	1	1	1	0
5	1	0	0	1	13	1	1	1	1
6	1	1	0	0	14	0	1	1	1
7	0	1	1	0	15	0	0	1	1
8	1	0	1	1	16	0	0	0	1

Theory of LFSR: Primitive Polynomials

Properties of Polynomials:

- ***Irreducible polynomial*** **–** cannot be factored, is divisible only by itself
- Irreducible polynomial of degree ***n*** is characterized by:
  - An odd number of terms including 1 term
  - Divisibility into $1 + x^k$, where $\boldsymbol{k} = 2^n – 1$
- Any polynomial with all even exponents can be factored and hence is ***reducible***
- An irreducible polynomial of degree ***n*** is ***primitive*** if it divides the polynomial $\mathbf{1+x^k}$ for $\mathbf{k = 2^n – 1}$, **but not for any smaller** positive integer k

Theory of LFSR: Examples

Polynomials of degree n=3 (examples):

$k = 2^n - 1 = 2^3 - 1 = 7$

Primitive polynomials:

$$x^3 + x^2 + 1$$

$$x^3 + x + 1$$

The polynomials will divide evenly the polynomial ***x^7 + 1*** but not any one of ***k<7***, hence, they are primitive

They are also reciprocal: coefficients are 1011 and 1101

Reducible polynomials (non-primitive):

Primitive polynomial

$$x^3 + 1 = (x+1)\boxed{(x^2 + x + 1)}$$

$$x^3 + x^2 + x + 1 = (x+1)(x^2+1)$$

The polynomials don't divide evenly the polynomial $x^7 + 1$

Theory of LFSR: Examples

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

Irreducible polynomial of degree *n* is characterized by:

- An odd number of terms including 1 term?

Yes, it includes 3 terms

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

No, there is remainder

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

Divisibility check:

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

Theory of LFSR: Examples

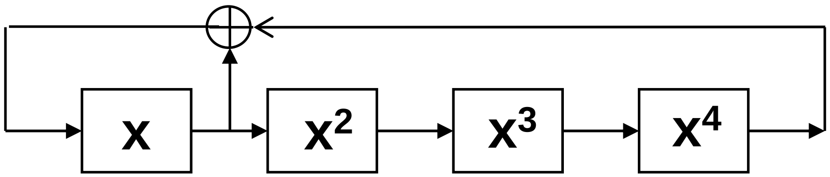
Comparison of test sequences generated:

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

Theory of LFSR: Examples

Primitive polynomial

$x^4 + x + 1$



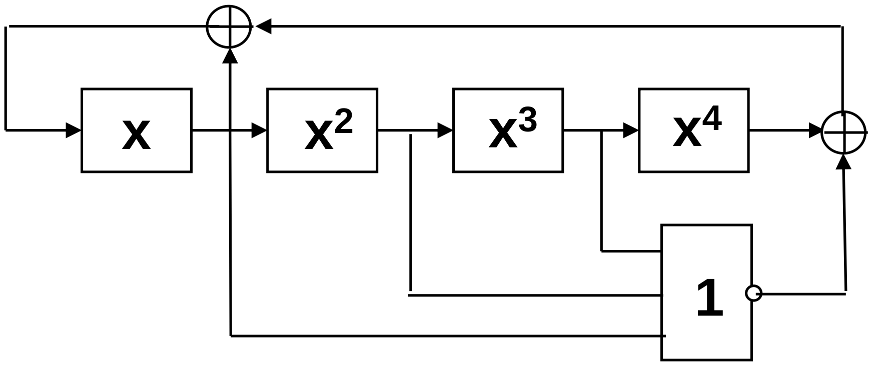
0001
1000
1100
1110
1111
0111

1011
0101
1010
1101
0110
0011

1001
0100
0010
0001

The code 0000 is missing

Zero generation:



0000
1000
1100
1110
1111
0111

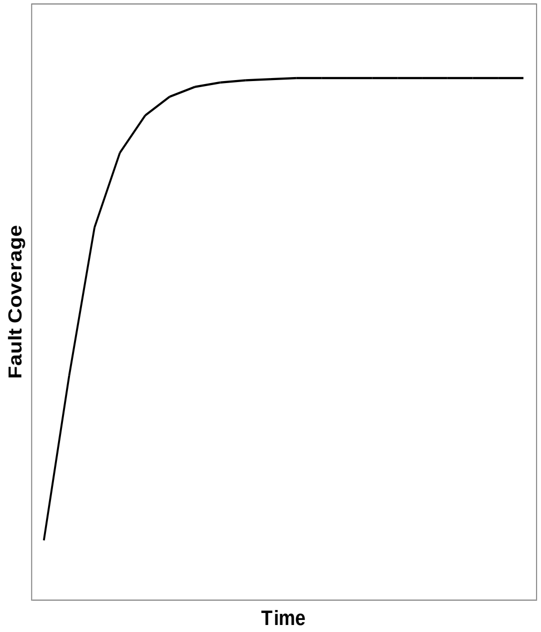
1011
0101
1010
1101
0110
0011

1001
0100
0010
0001
0000

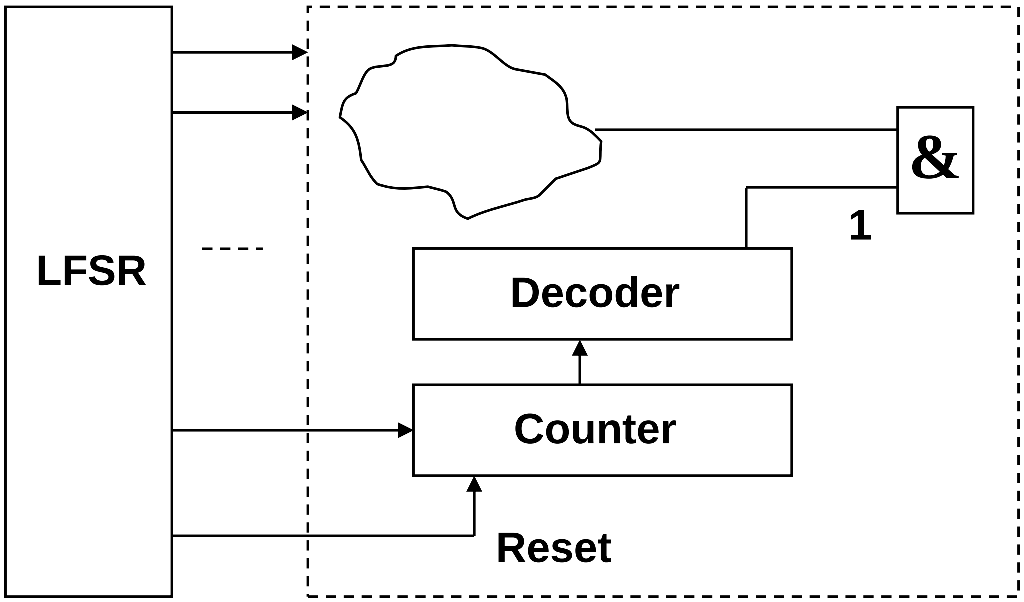
Other Problems with Pseudorandom Test

The main motivations of using random patterns are:

- low generation cost
- high initial efeciency



Problem: low fault coverage

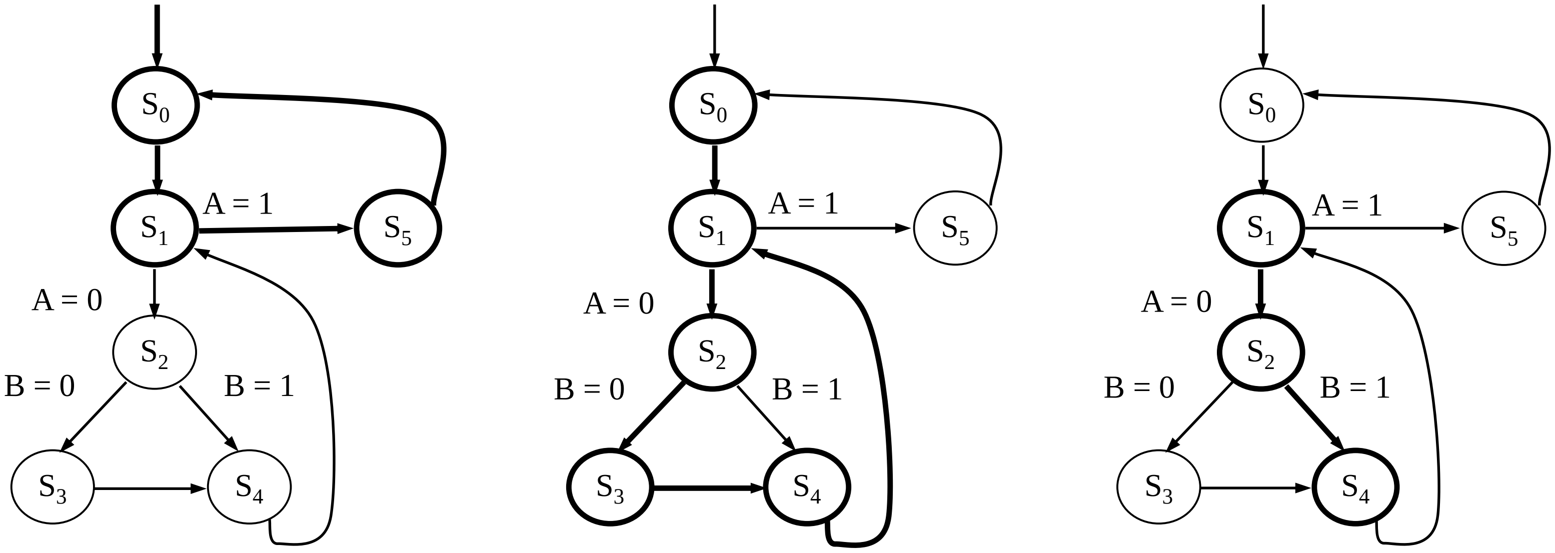


If Reset = 1 signal has probability 0,5 then counter will not work and
1 for AND gate may never be produced

Sequential BIST

A DFT technique of BIST for sequential circuits is proposed

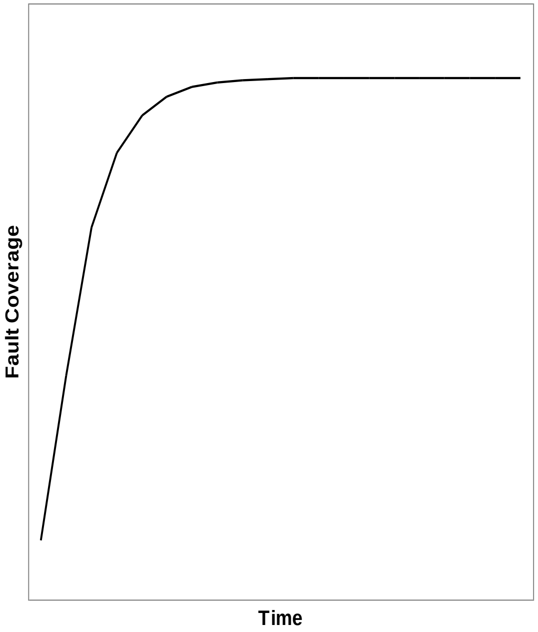
The approach proposed is based on all-branches coverage metrics which is known to be more powerful than all-statement coverage



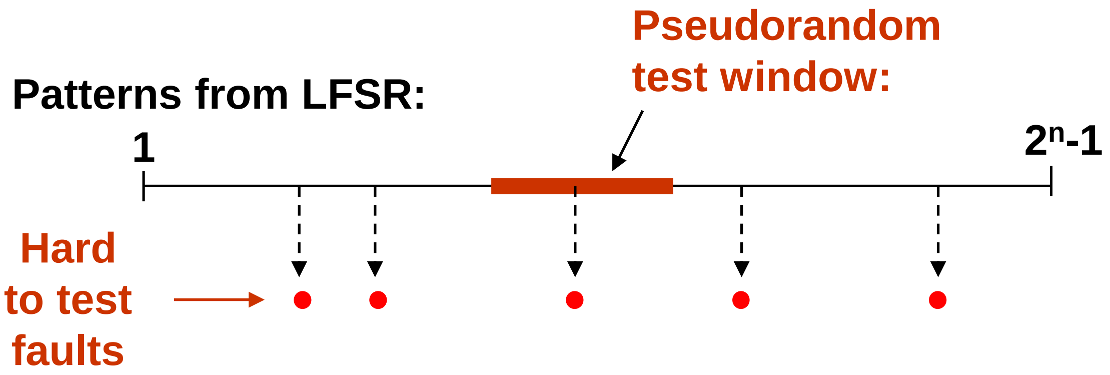
Problems with BIST: Hard to Test Faults

The main motivations of using random patterns are:

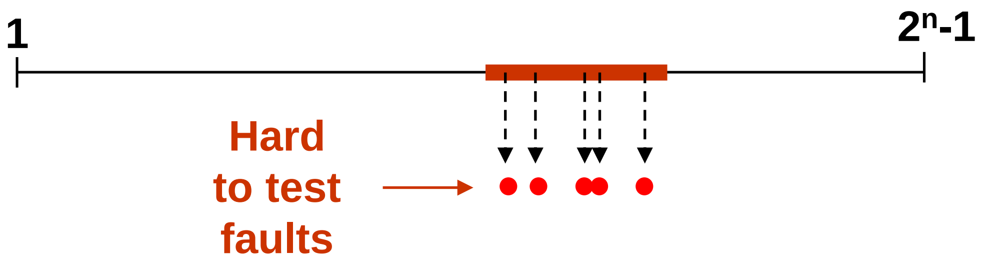
- low generation cost
- high initial efeciency



Problem: Low fault coverage

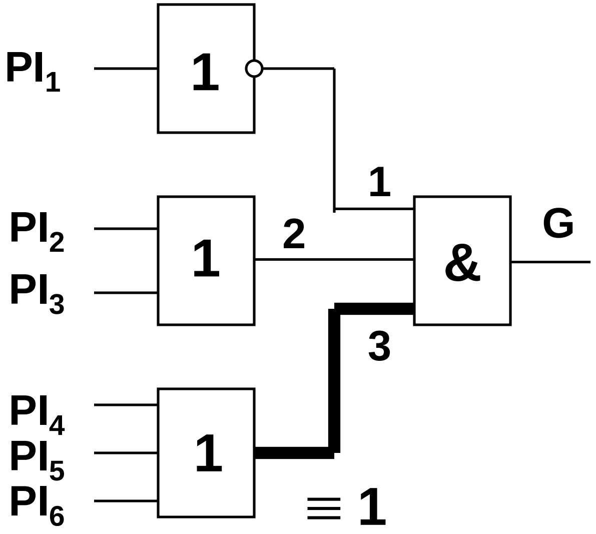


Dream solution: Find LFSR such that:



BIST: Weighted pseudorandom test

Calculation of signal probabilities:



For PI_1 :	**P = 0.15**
For PI_2 and PI_3 :	**P = 0.6**
For PI_4 - PI_6 :	**P = 0.4**

Probability of detecting the fault ≡1 at the input 3 of the gate G:

1) equal probabilities (p = 0.5):

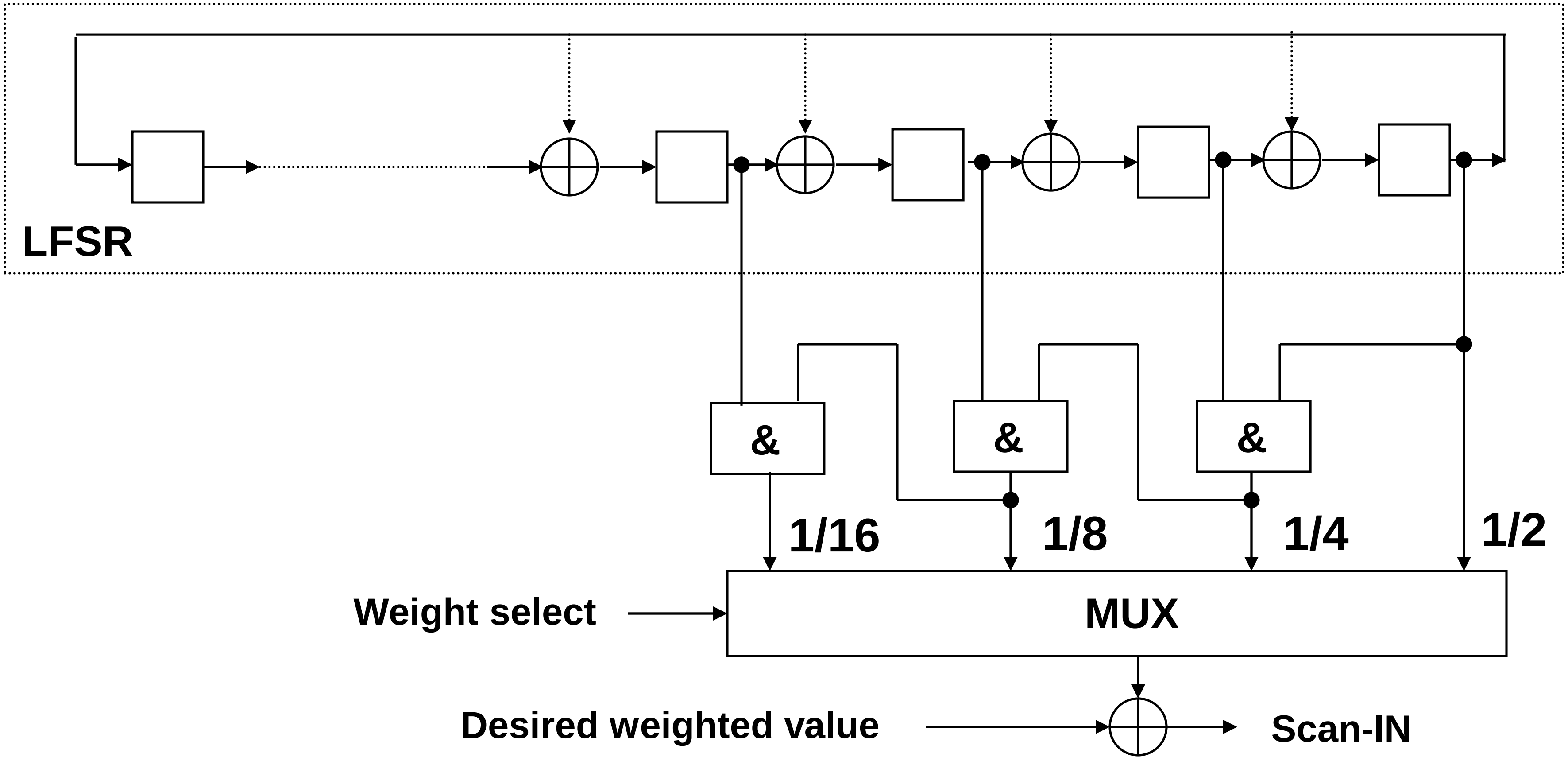
$$P = 0.5 * (0.25 + 0.25 + 0.25) * 0.5^3 =$$
$$= 0.5 * 0.75 * 0.125 =$$
$$= 0.046$$

2) weighted probabilities:

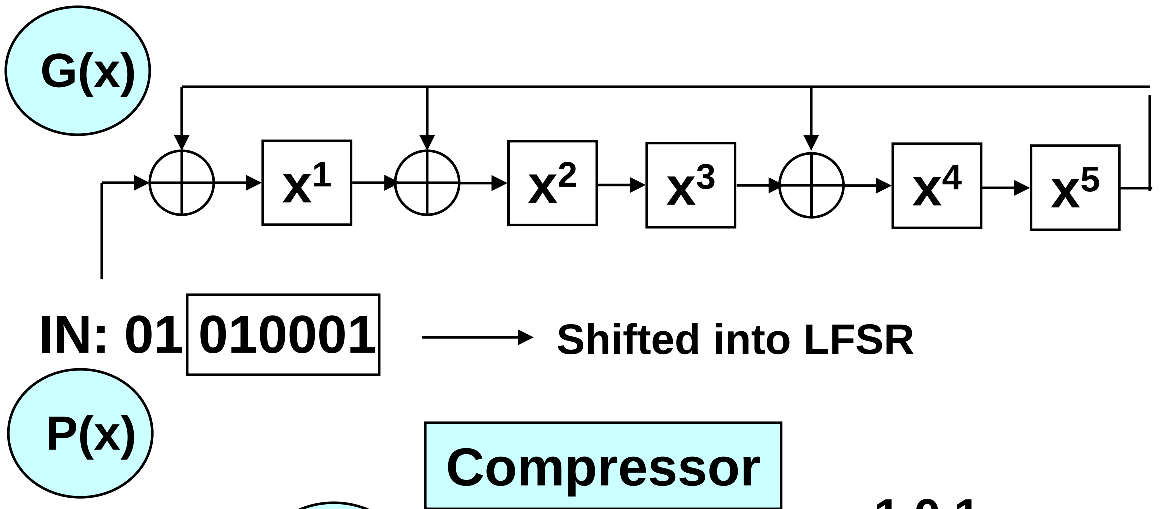
$$P = 0.85 *$$
$$* (0.6 * 0.4 + 0.4 * 0.6 + 0.6^2) *$$
$$* \ 0.6^3 =$$
$$= 0.85 * 0.84 * 0.22 =$$
$$= 0.16$$

BIST: Weighted pseudorandom test

Hardware implementation of weight generator



BIST: Signature Analysis



The division process can be mechanized using LFSR

The divisor polynomial G(x) is defined by the feedback connections

Shift creates x^5 which is replaced by $x^5 = x^3 + x + 1$

Compressor

G(x)

Response

P(x)

```
                    1 0 1
             -----------------
1 0 1 0 1 1 | 1 0 0 0 1 0 1 0
              1 0 1 0 1 1
             -----------------
              0 0 1 0 0 1 1 0
                  1 0 1 0 1 1
                 -------------
                  0 0 1 1 0 1  = R(x) = x³ + x² + 1
```

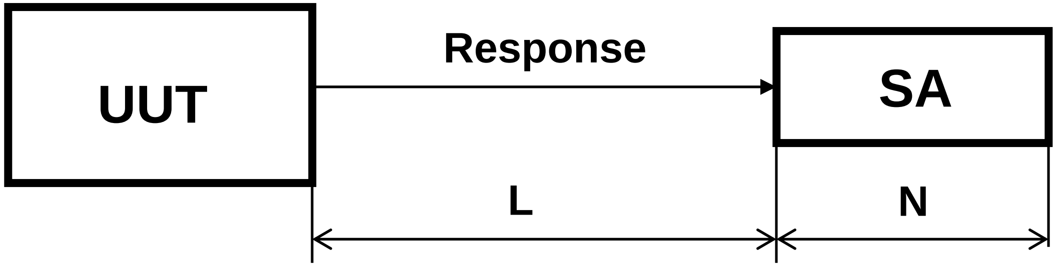
x^5

$$\frac{P(x)}{G(x)} = \frac{x^7 + x^3 + x}{x^5 + x^3 + x + 1}$$

Signature

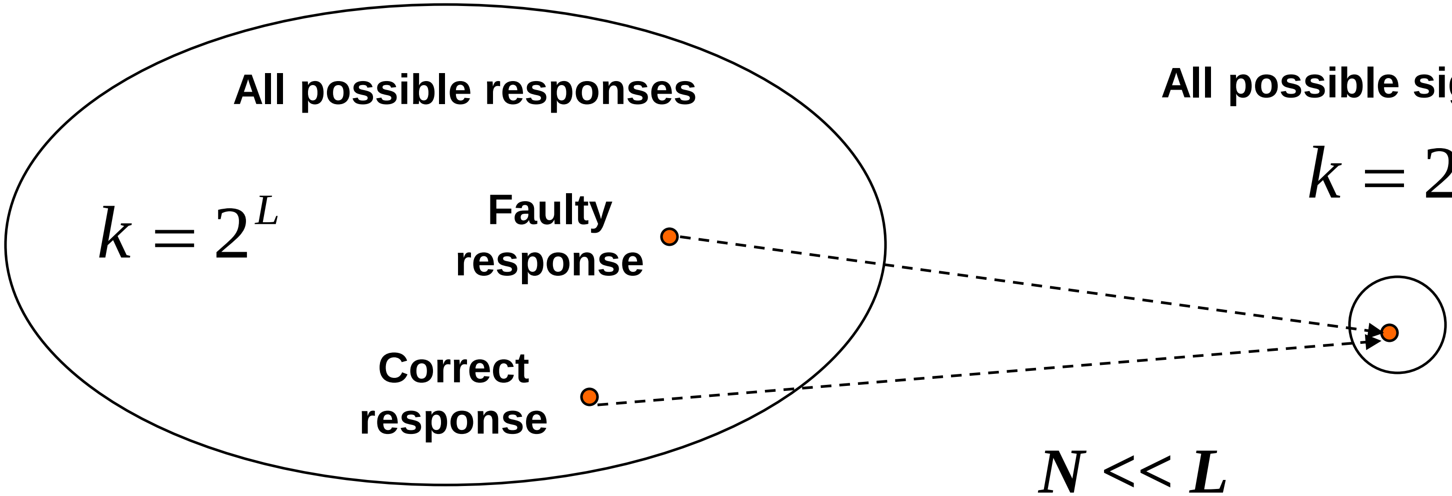
BIST: Signature Analysis

Aliasing:



L - test length

N - number of stages in Signature Analyzer



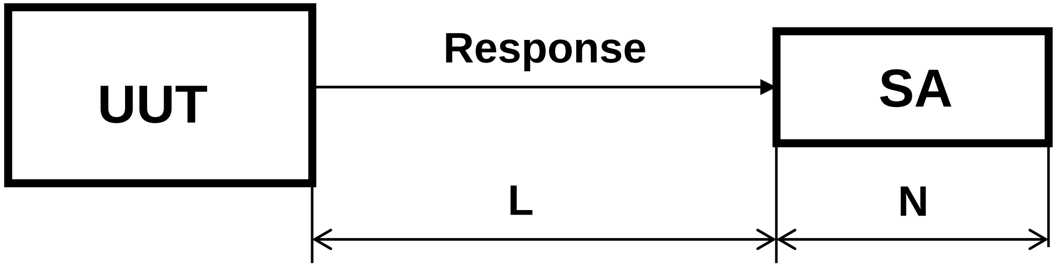
All possible signatures

$$k = 2^N$$

$$N << L$$

BIST: Signature Analysis

Aliasing:



L - test length

N - number of stages in Signature Analyzer

$k = 2^L$ - number of different possible responses

No aliasing is possible for those strings with *L - N* leading zeros since they are represented by polynomials of degree *N - 1* that are not divisible by characteristic polynomial of LFSR

$$\underbrace{000000000000000 \ldots 00000\ \underbrace{XXXXX}_{N}}_{L}$$

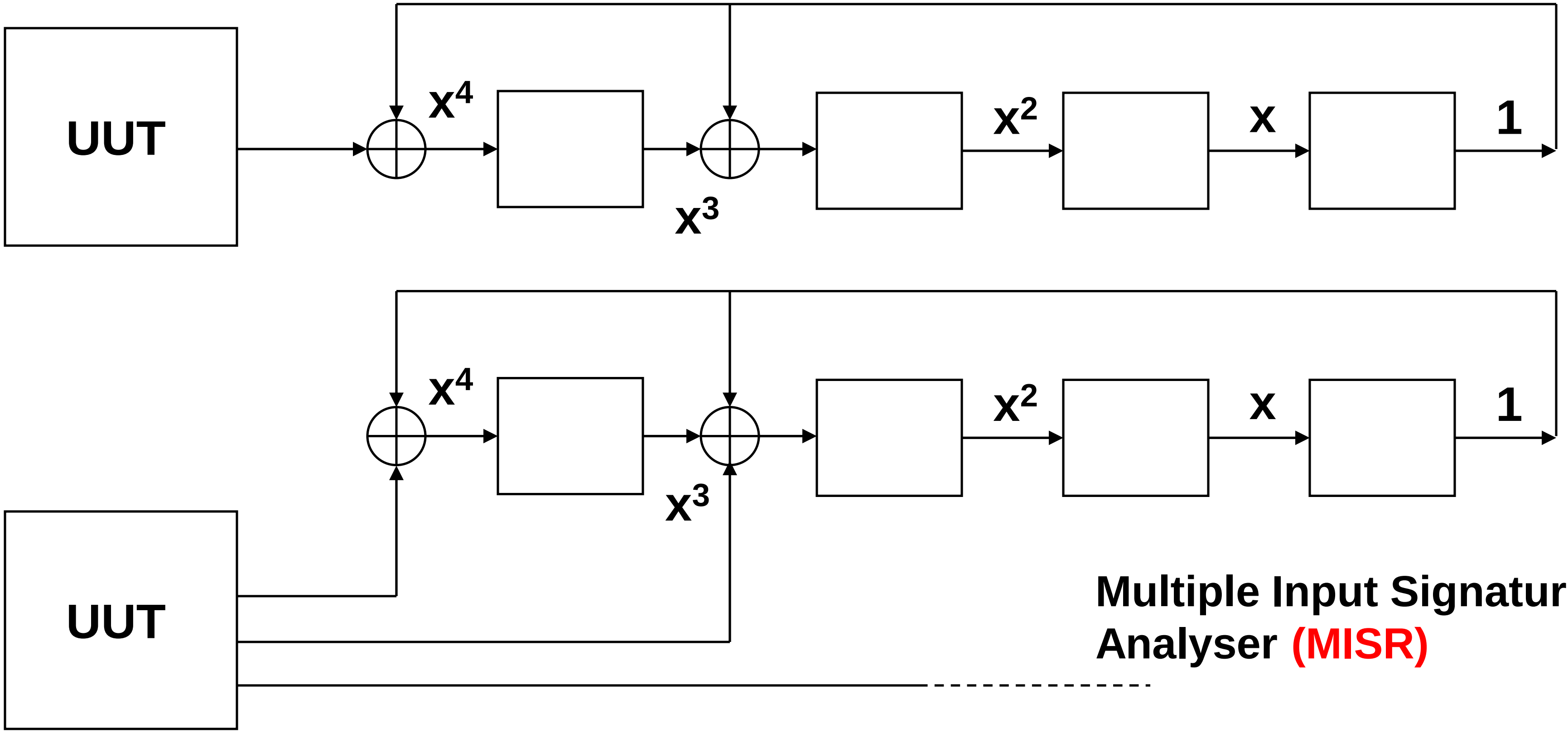
$2^{L-N} - 1$ - aliasing is possible

Probability of aliasing: $P = \dfrac{2^{L-N} - 1}{2^L - 1} \xrightarrow{L >> 1} \boxed{P = \dfrac{1}{2^N}}$

BIST: Signature Analysis

Parallel Signature Analyzer:

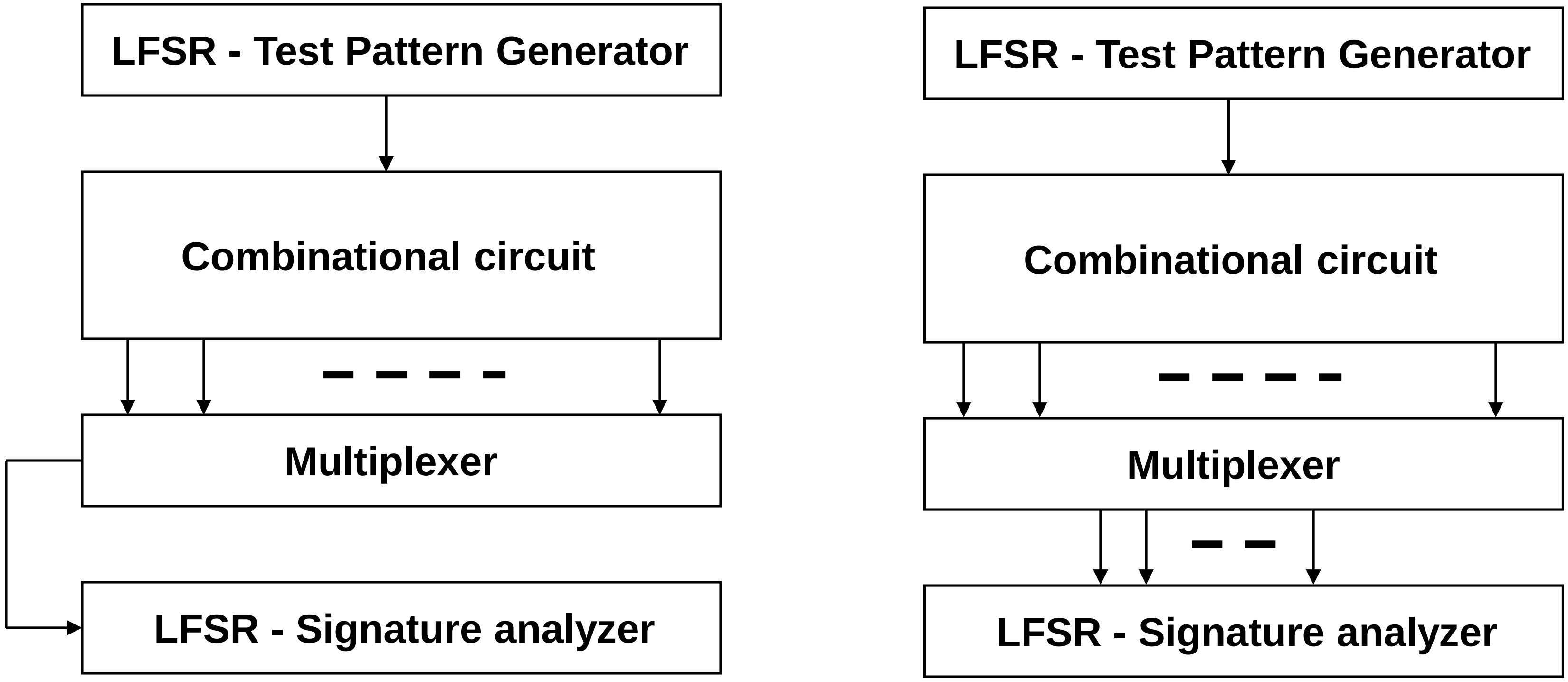
Single Input Signature Analyser



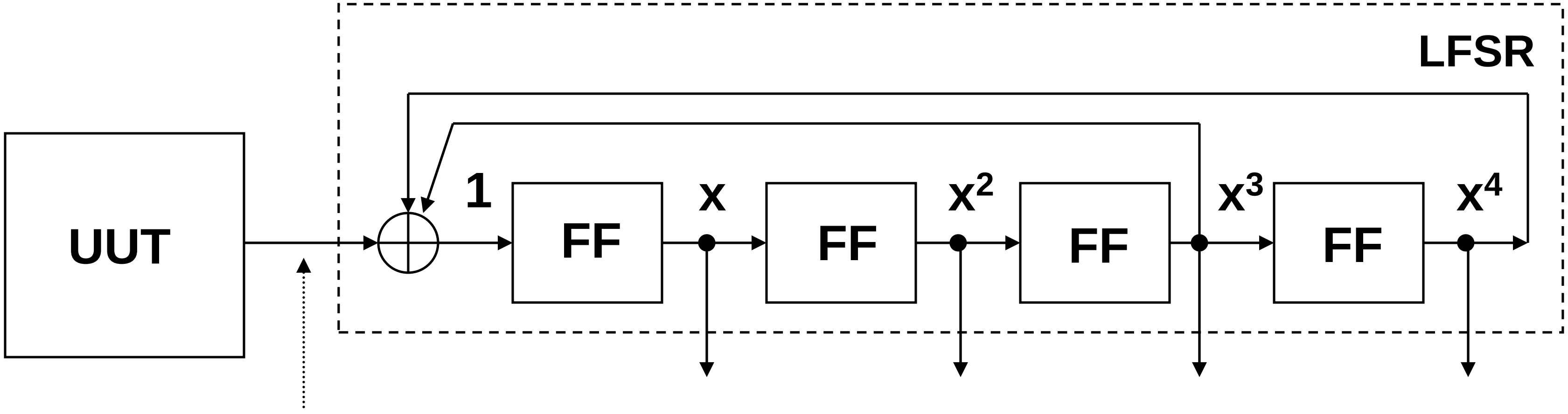
Multiple Input Signature Analyser (MISR)

BIST: Signature Analysis

Signature calculating for multiple outputs:


LFSR - Test Pattern Generator
Combinational circuit
Multiplexer
LFSR - Signature analyzer
LFSR - Test Pattern Generator
Combinational circuit
Multiplexer
LFSR - Signature analyzer

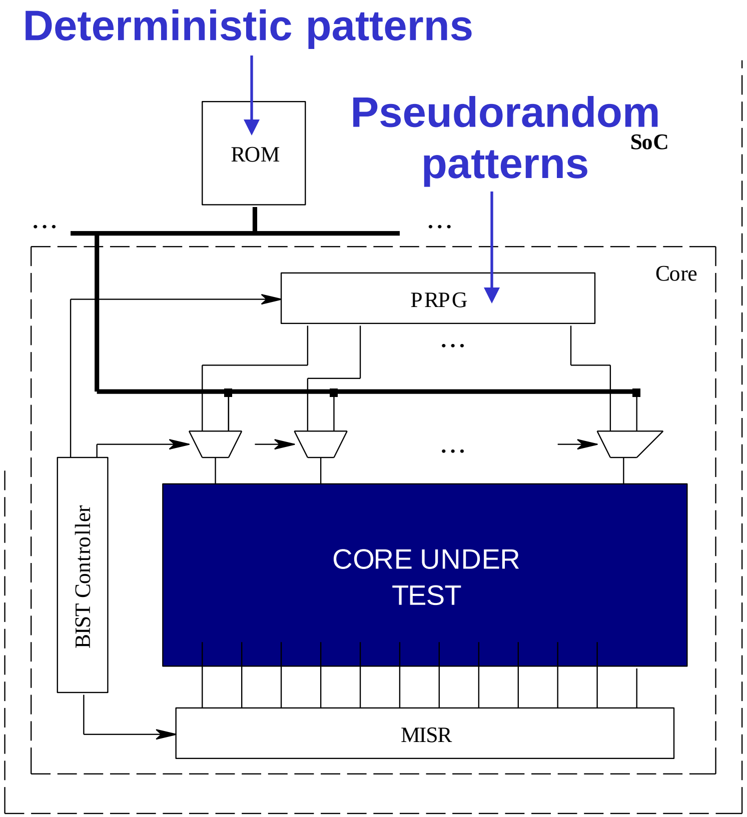
BIST: Joining TPG and SA



Response string for
Signature Analysis

Test Pattern (when generating tests)
Signature (when analyzing test responses)

Hybrid Built-In Self-Test


Deterministic patterns

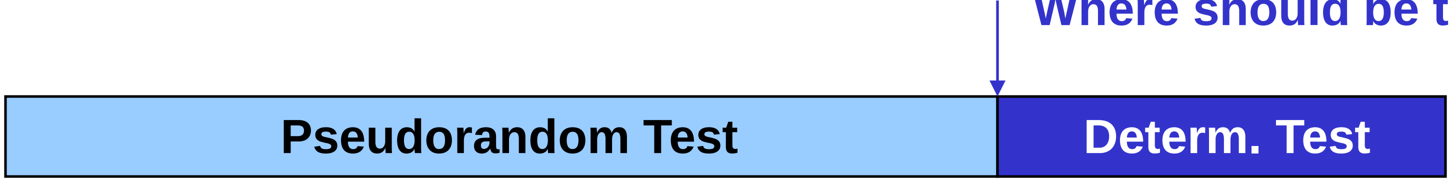
Pseudorandom patterns

Hybrid test set contains pseudorandom and deterministic vectors

Pseudorandom test is improved by a stored test set which is specially generated to target the random resistant faults

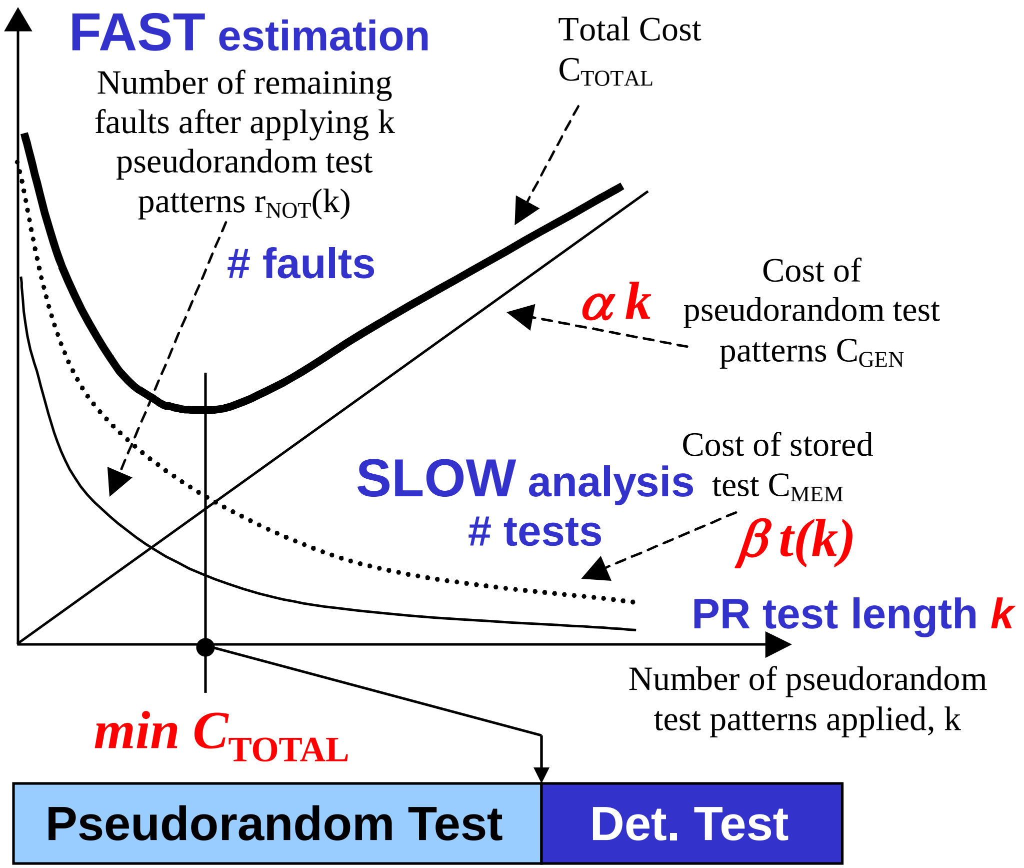
Optimization problem:

Where should be this breakpoint?



Optimization of Hybrid BIST

Cost of BIST: $C_{TOTAL} = \alpha k + \beta t(k)$



FAST estimation

Number of remaining faults after applying k pseudorandom test patterns $r_{NOT}(k)$

Total Cost C_{TOTAL}

Cost of pseudorandom test patterns C_{GEN}

SLOW analysis

Cost of stored test C_{MEM}

Number of pseudorandom test patterns applied, k

PR test length → k

# faults not detected (fast analysis) → $r_{NOT}(k)$

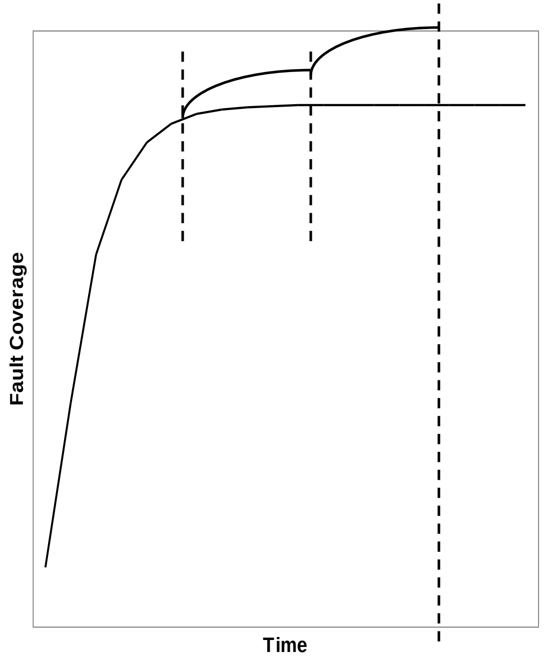
# tests needed (slow analysis) → $t(k)$

k	$r_{DET}(k)$	$r_{NOT}(k)$	$FC(k)$	$t(k)$
1	155	839	15.6%	104
2	76	763	23.2%	104
3	65	698	29.8%	100
4	90	608	38.8%	101
5	44	564	43.3%	99
10	104	421	57.6%	95
20	44	311	68.7%	87
50	51	218	78.1%	74
100	16	145	85.4%	52
200	18	114	88.5%	41
411	31	70	93.0%	26
954	18	28	97.2%	12
1560	8	16	98.4%	7
2153	11	5	99.5%	3
3449	2	3	99.7%	2
4519	2	1	99.9%	1
4520	1	0	100.0%	0

Hybrid BIST with Reseeding

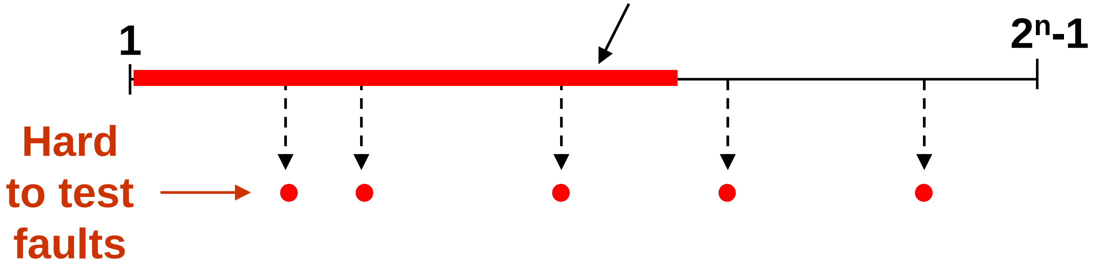
The motivation of using random patterns is:

- low generation cost
- high initial efeciency



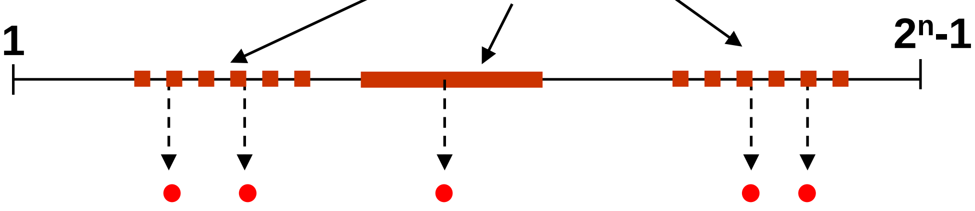
Problem: low fault coverage → long PR test

Pseudorandom test:

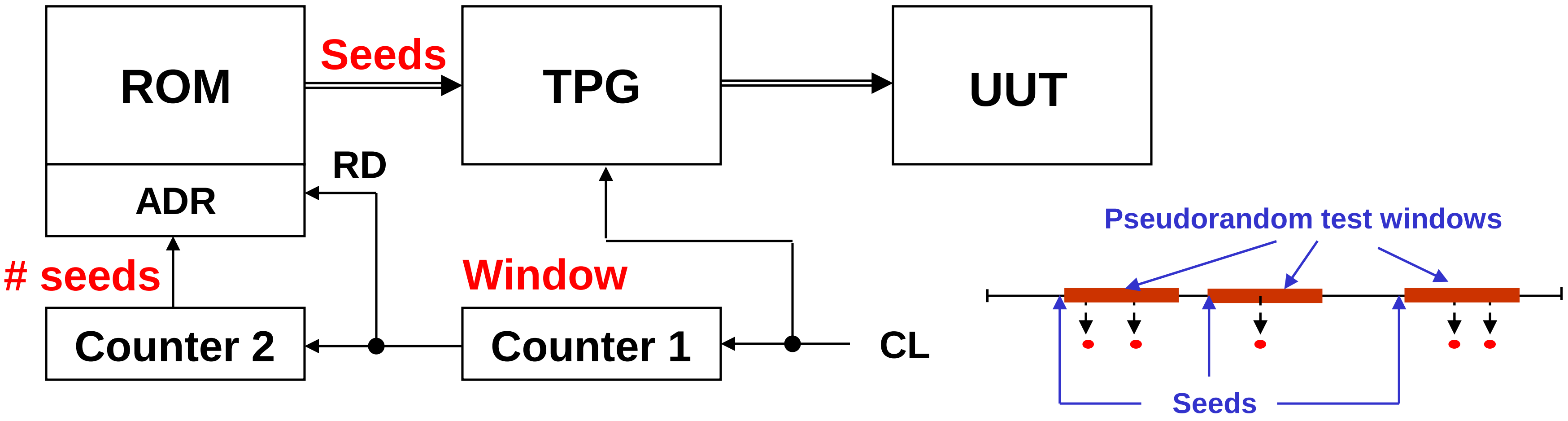


Solution: many seeds:

Pseudorandom test:

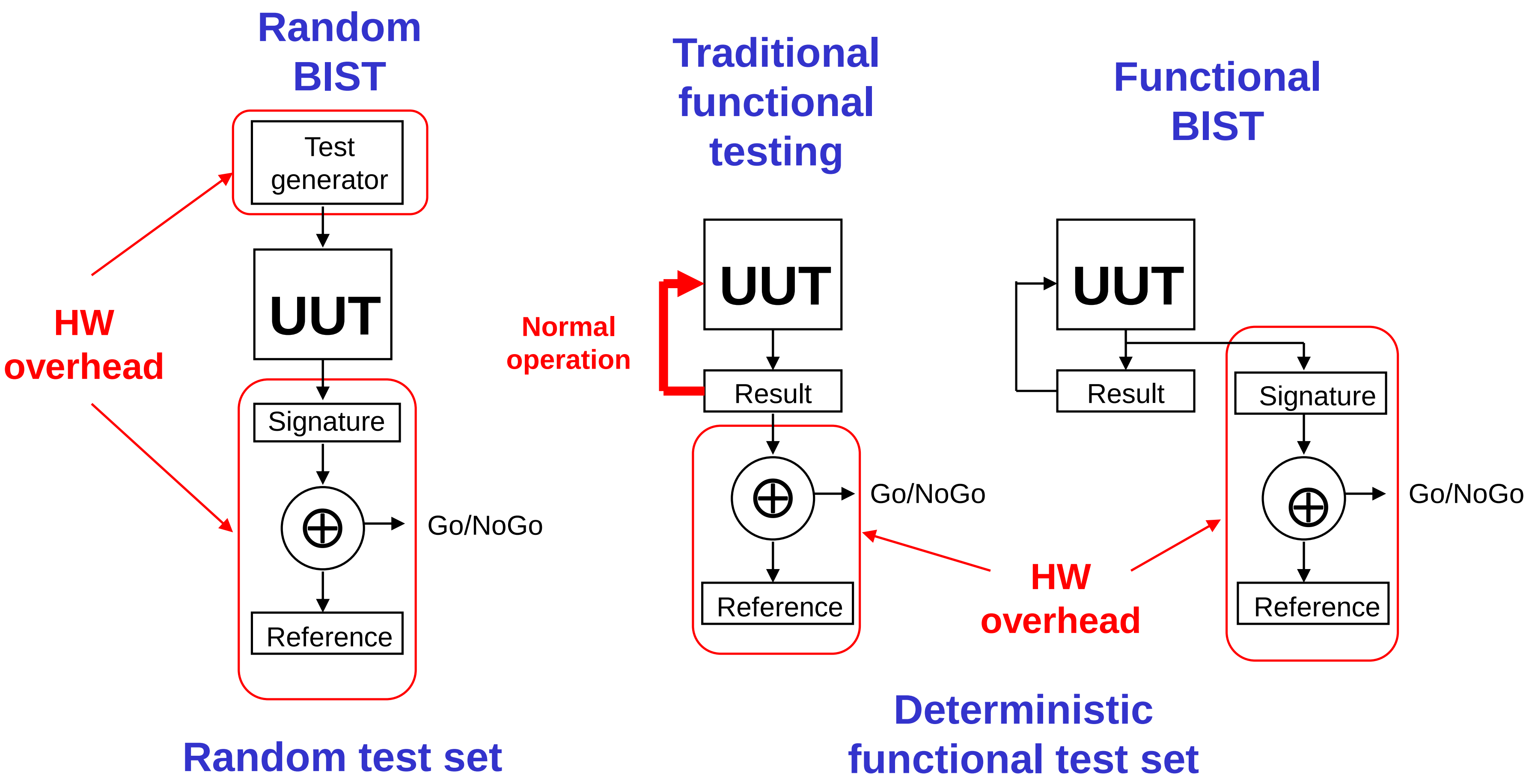


Store-and-Generate Test Architecture



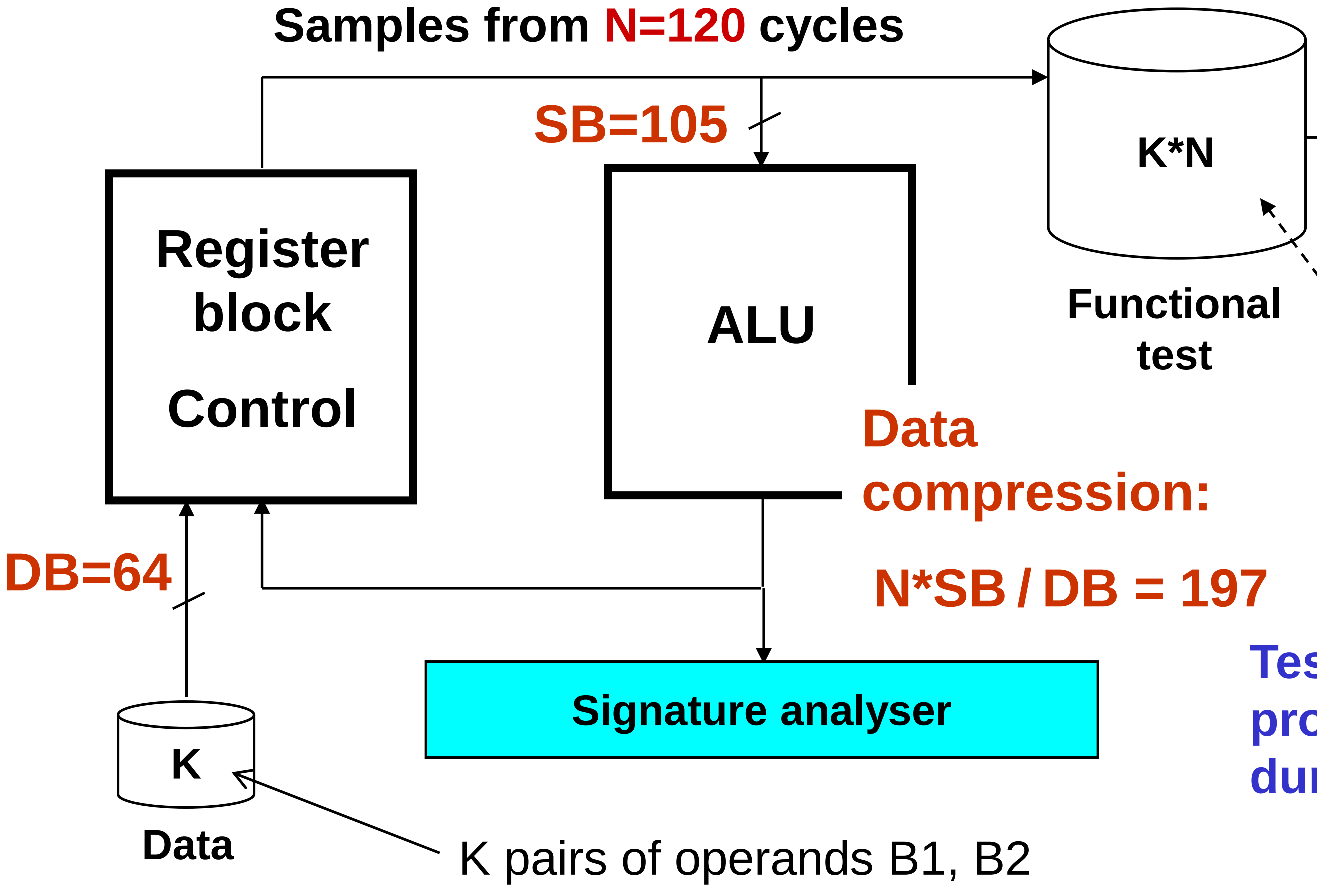
- ROM contains test patterns for hard-to-test faults
- Each pattern P_k in ROM serves as an initial state of the LFSR for test pattern generation (TPG) - **seeds**
- Counter 1 counts the number of pseudorandom patterns generated starting from P_k - **width of the windows**
- After finishing the cycle for Counter 2 is incremented for reading the next pattern P_{k+1} – **beginning of the new window**

Random BIST vs Functional BIST



Example: Functional BIST for Divider

Functional BIST quality analysis for



Data compression:

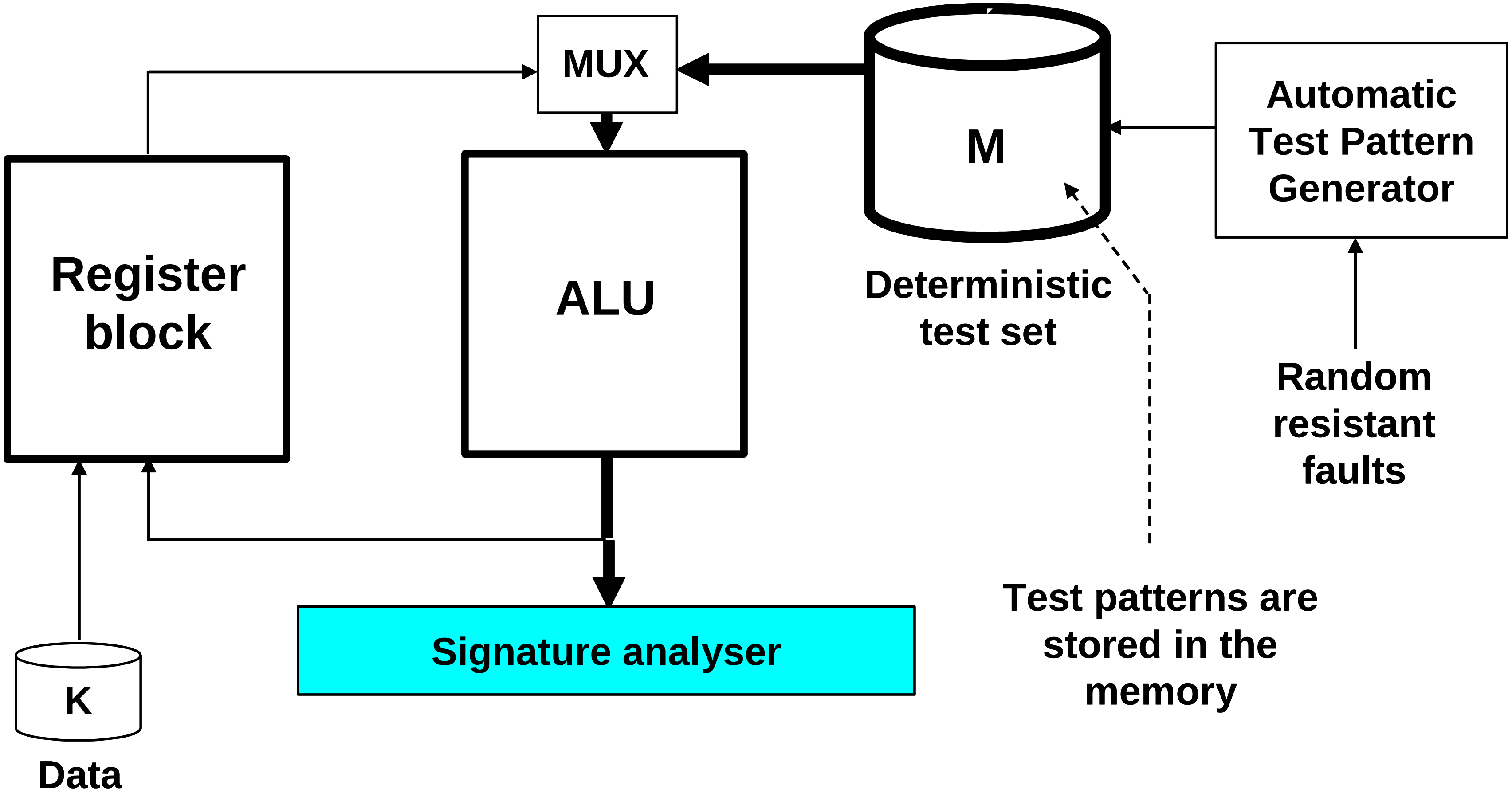
N*SB / DB = 197

Test patterns (samples) are produced on-line during the working mode

K pairs of operands B1, B2

Hybrid Functional BIST for Divider

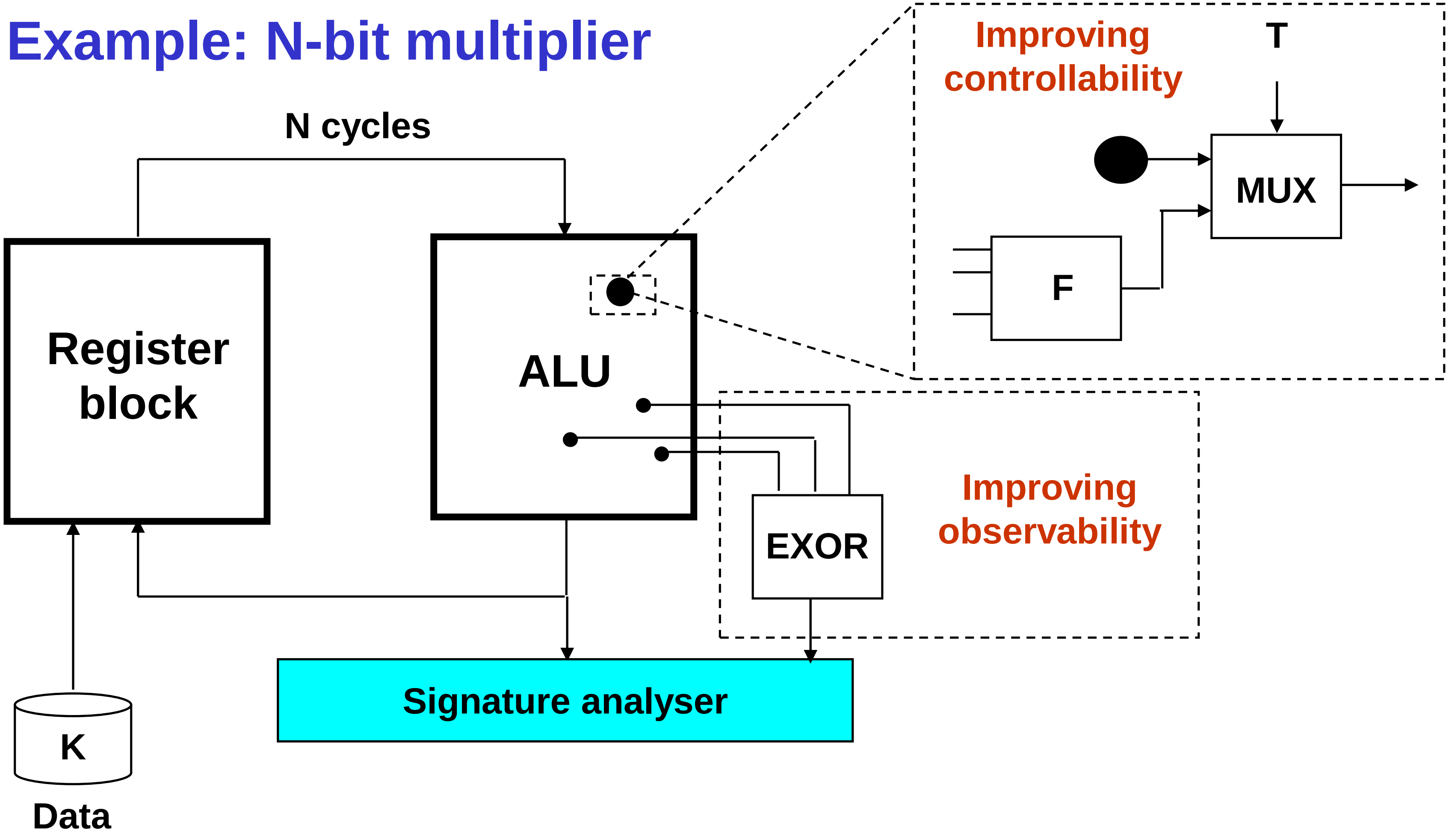
Functional BIST implementation



Test patterns are stored in the memory

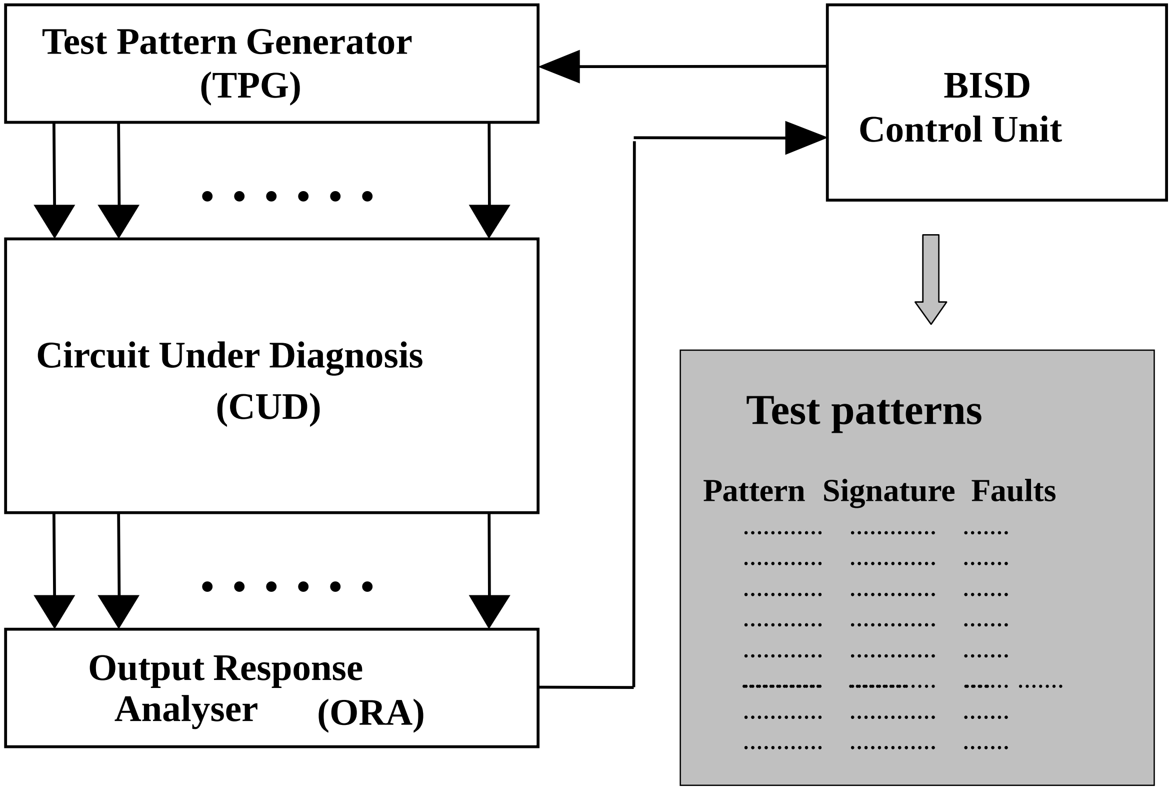
Functional Self-Test with DFT

Example: N-bit multiplier



Embedded BIST Based Fault Diagnosis

BISD scheme:



Test Pattern Generator (TPG)

BISD Control Unit

Circuit Under Diagnosis (CUD)

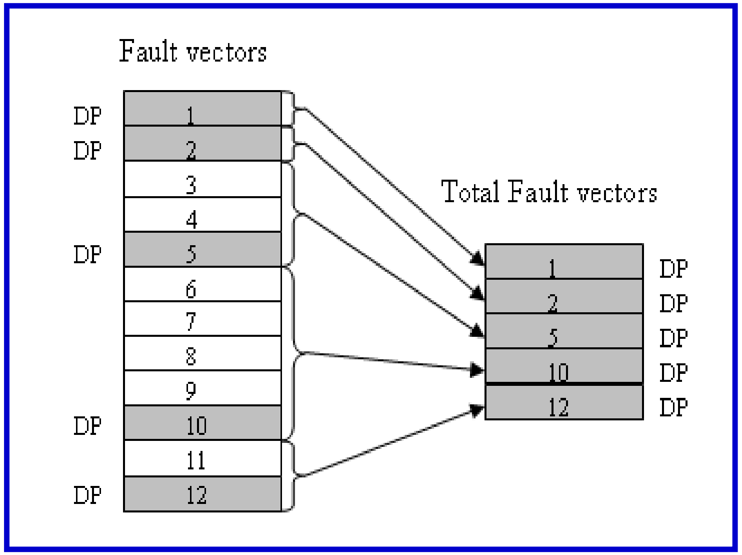
Output Response Analyser (ORA)

Test patterns

Pattern Signature Faults

```
............  .............  .......
............  .............  .......
............  .............  .......
............  .............  .......
............  .............  .......
............  .............  ....... .......
............  .............  .......
............  .............  .......
```

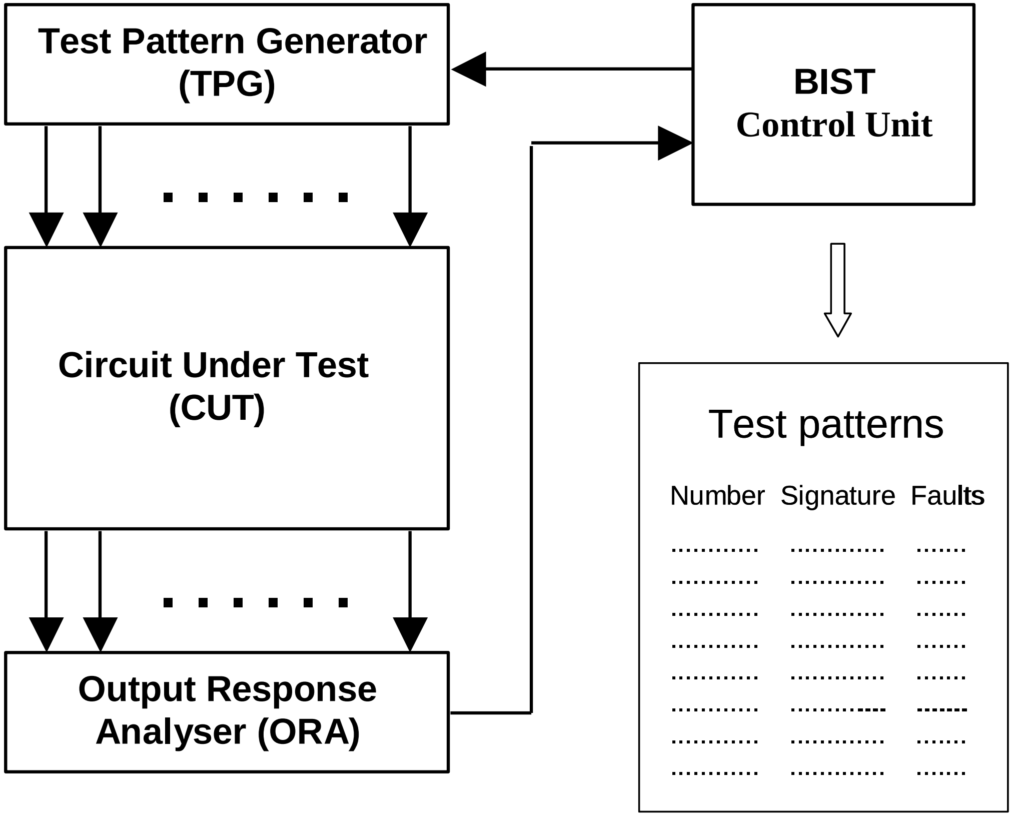
Pseudorandom test sequence:



Diagnostic Points (DPs) – patterns that detect new faults

Further minimization of DPs – as a tradeoff with diagnostic resolution

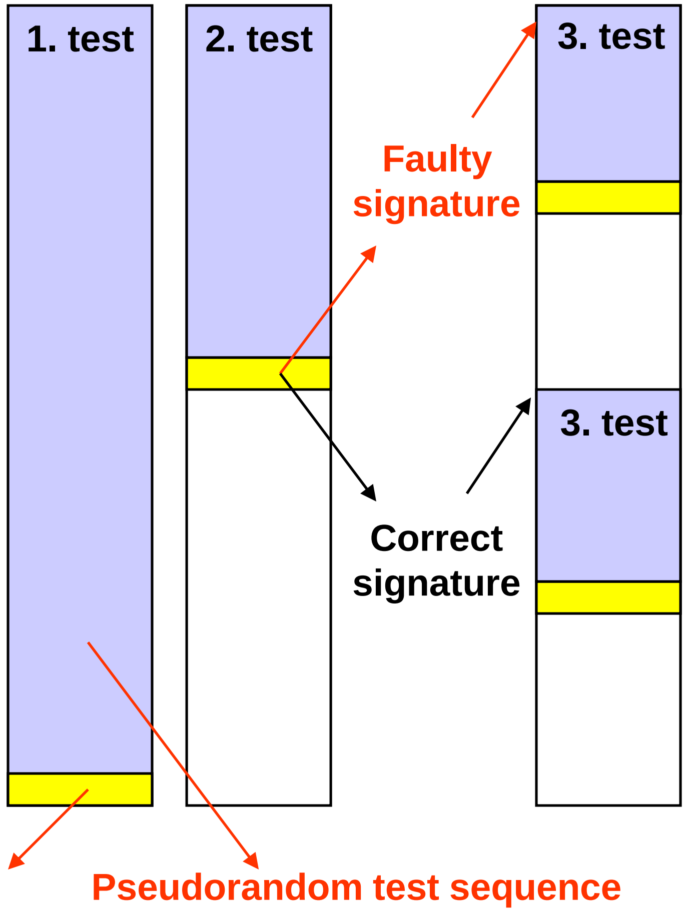
Built-In Fault Diagnosis



Test patterns

Number	Signature	Faults
....	....	....
....	....	....
....	....	....
....	....	....
....	....	....
....	....	....
....	....	....
....	....	....

Diagnosis procedure:



Built-In Fault Diagnosis

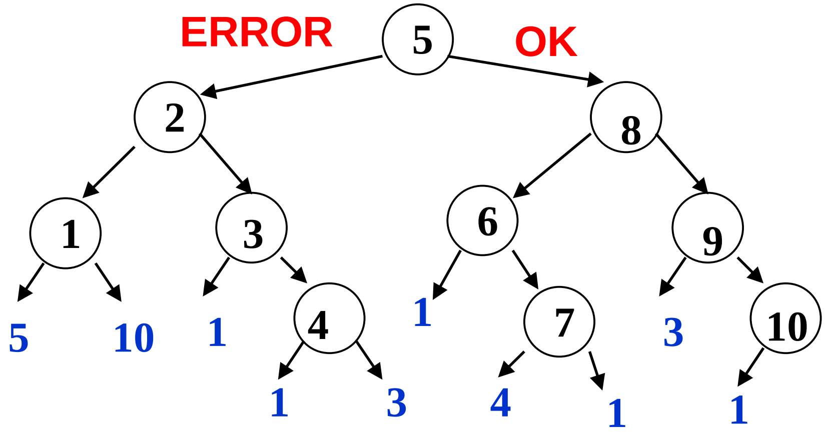
Measuring of information we get from the test:

$$I = -\, p\, log_2\, p - (1\text{-}p)\, log_2\, (1\text{-}p)$$

Pseudorandom test fault simulation (detected faults)

№	All faults	New faults	Coverage
1	5	5	16.67%
2	15	10	50.00%
3	16	1	53.33%
4	17	1	56.67%
5	20	3	66.67%
6	21	1	70.00%
7	25	4	83.33%
8	26	1	86.67%
9	29	3	96.67%
10	30	1	100.00%

Binary search with bisectioning of test patterns



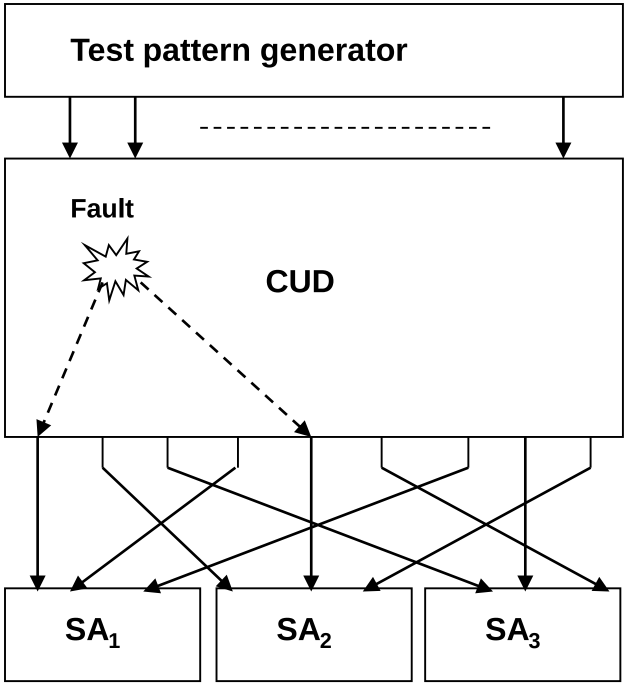
Average number of test sessions: 3,3

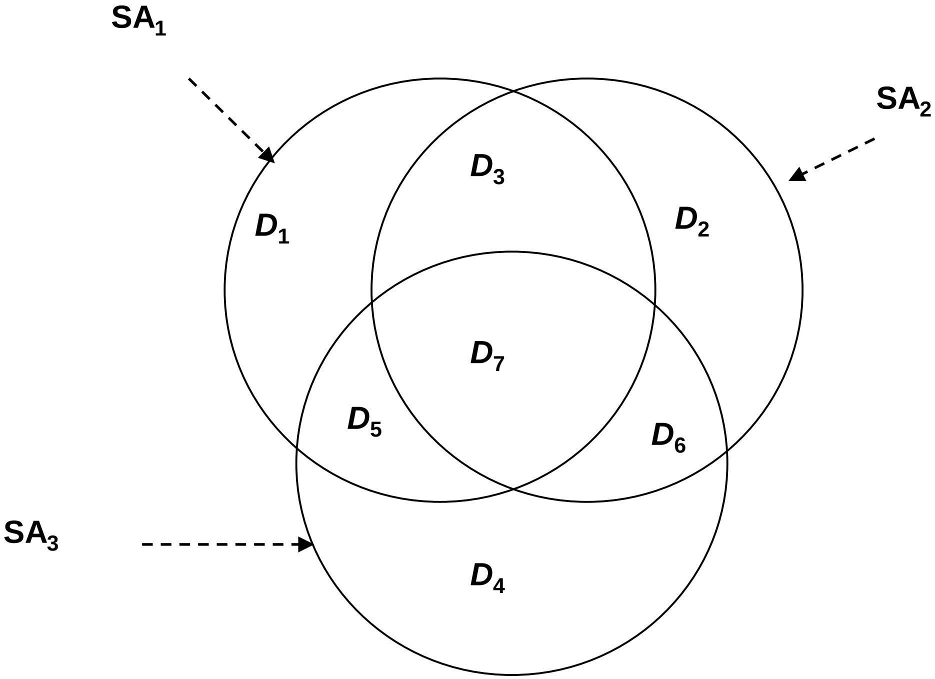
Average number of clocks: 8,67

Built-In Fault Diagnosis

Diagnosis with multiple signatures

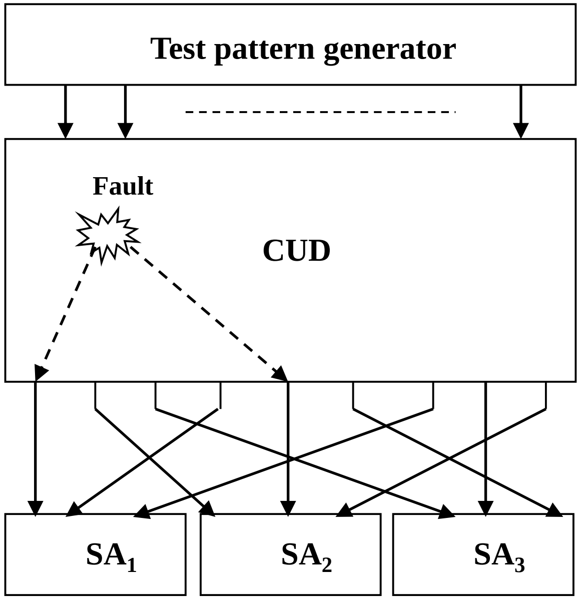
(based on reasoning of spacial information):



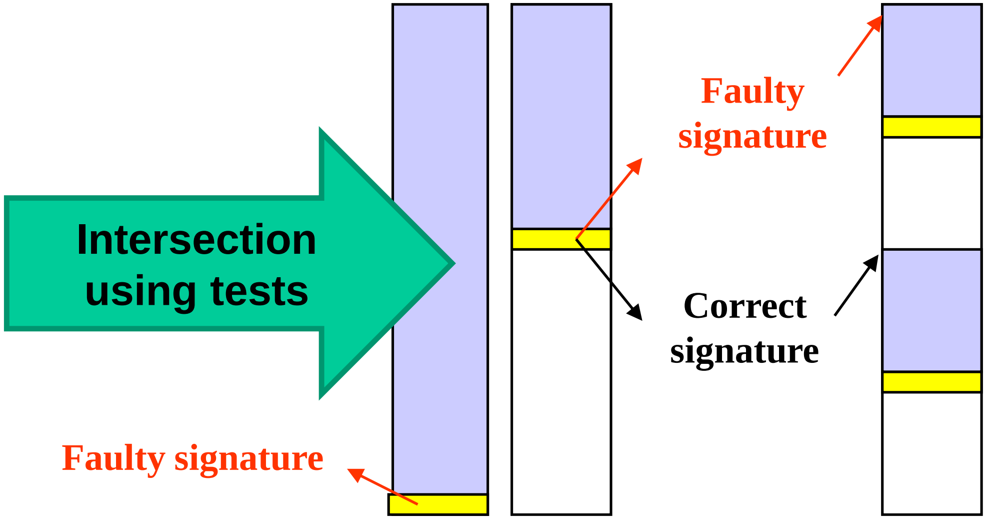


Built-In Fault Diagnosis

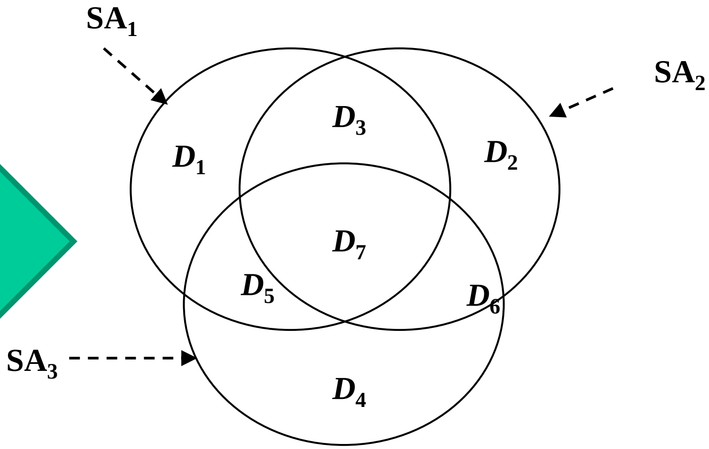
BIST with multiple signature analyzers



Intersection using tests



Intersection using SA-s



Optimization of the interface between CUD and SA-s

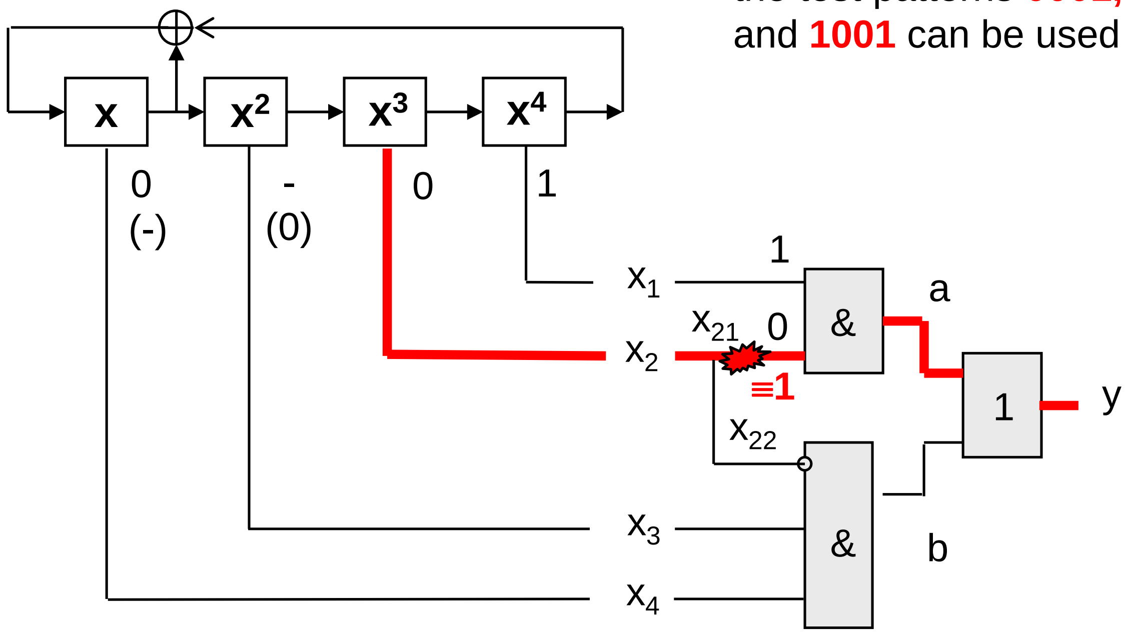
Pseudorandom Testing with LFSR

Primitive polynomial

$x^4 + x + 1$

0001
1000
1100
1110
1111
0111
1011
0101
1010
1101
0110
0011
1001
0100
0010
0001

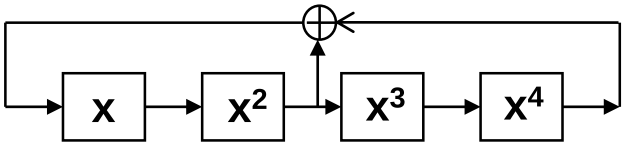
For testing the fault $x_{21} \equiv 1$ the test patterns **0001, 0101** and **1001** can be used



Pseudorandom Testing with LFSR

Non-primitive polynomial

$x^4 + x^2 + 1$

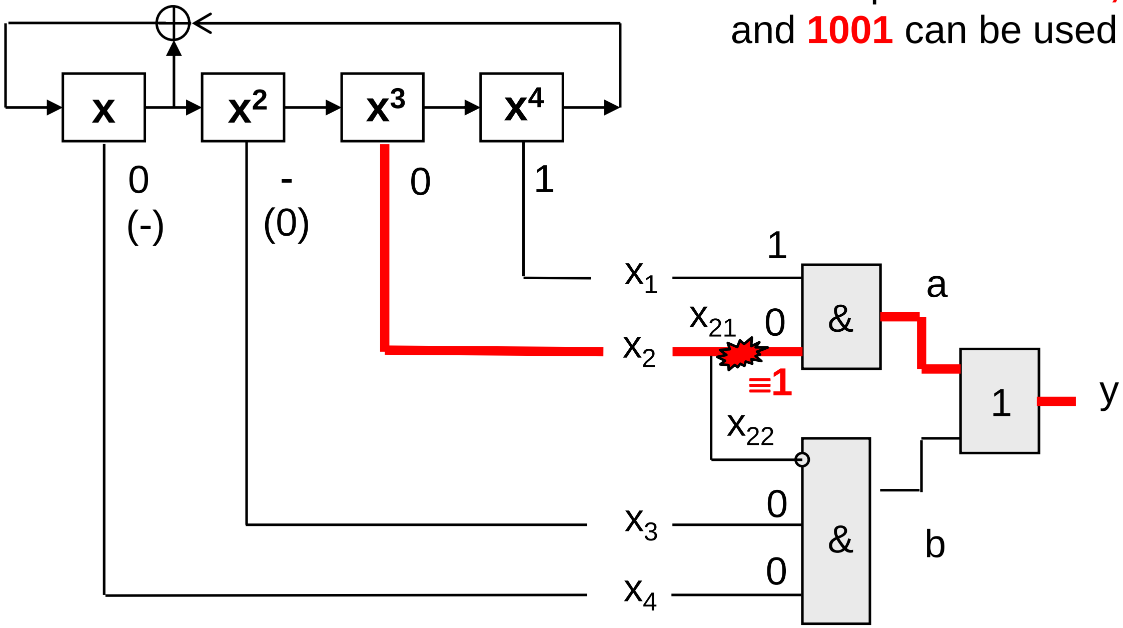


0001
1000
0100
1010
0101
0010
0001

1001
1100
1110
1111
0111
0011
1001

0110
1011
1101
0110

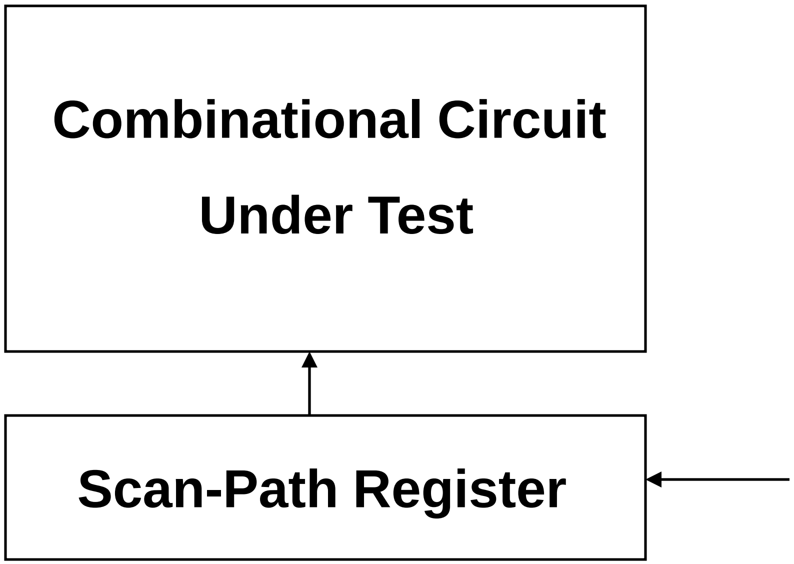
For testing the fault $x_{21} \equiv 1$ the test patterns **0001, 0101** and **1001** can be used



Be careful: no proper patterns can be generated using the seed 0110

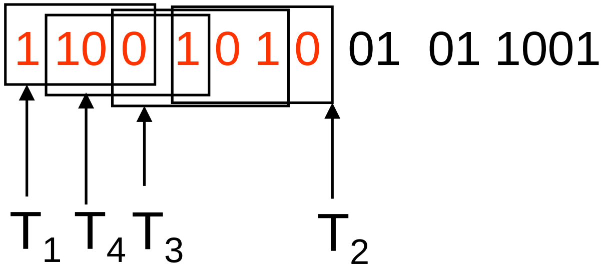
Deterministic Scan-Path Test

Test per Clock:



- **Initial test set:**
- T1: 1100
- T2: 1010
- T3: 0101
- T4: 1001

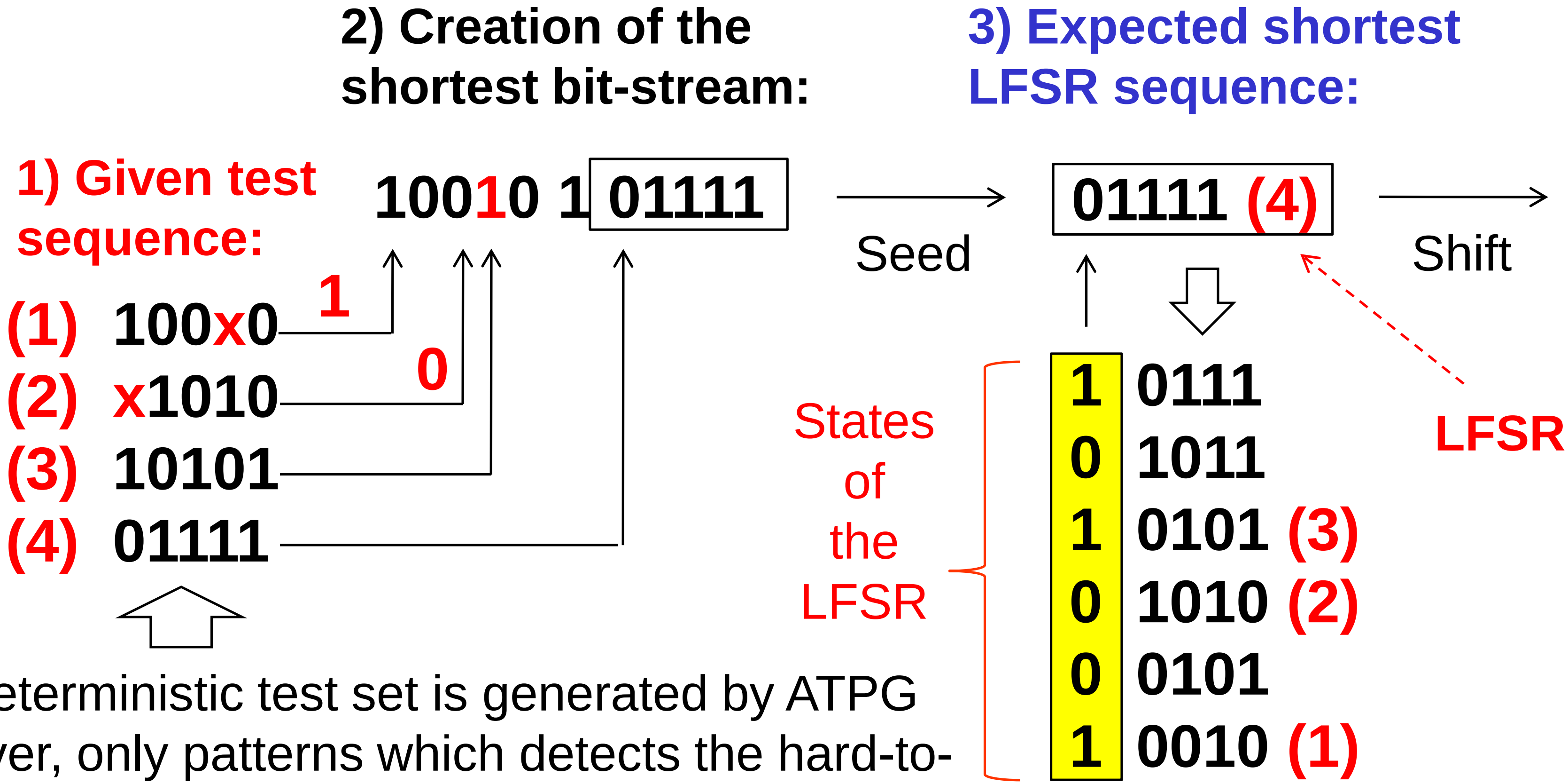
- **Test application:**
- 1 10 0 1 0 1 0 01 01 1001



- Number of clocks = 8 < 20

Deterministic Synthesis of LFSR

Generation of the polynomial and seed for the given test sequence



1) Given test sequence:

(1) 100x0

(2) x1010

(3) 10101

(4) 01111

2) Creation of the shortest bit-stream:

10010 1 01111

1

0

3) Expected shortest LFSR sequence:

Seed

01111 (4)

Shift

LFSR

States of the LFSR

1	0111	
0	1011	
1	0101	(3)
0	1010	(2)
0	0101	
1	0010	(1)

This deterministic test set is generated by ATPG However, only patterns which detects the hard-to-test faults can be chosen

Deterministic Synthesis of LFSR

Generation of the polynomial and seed for the given test sequence

System of linear equations: $a_k x_1 \oplus b_k x_2 \oplus c_k x_3 \oplus d_k x_4 \oplus e_k x_5 = f_k$

Expected shortest LFSR sequence:

01111 (4)

1 0111
0 1011
1 0101 (3)
0 1010 (2)
0 0101
1 0010 (1)

Currrent state of the LFSR

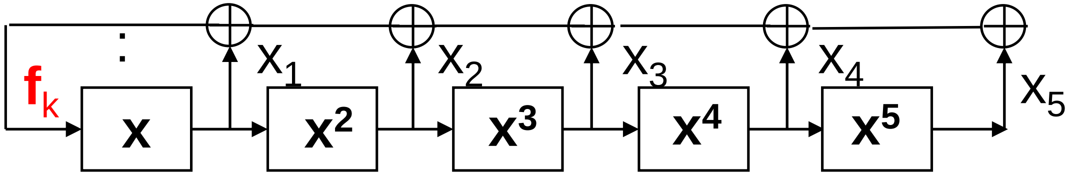
k=1,2…6
j=1,2,,,5

$$\begin{array}{ccccc} a_k b_k c_k d_k e_k & & x_j & & f_k \\ \left|\begin{array}{c} 01111\\ 10111\\ 01011\\ 10101\\ 01010\\ 00101 \end{array}\right| & \times & \left|\begin{array}{c} x_1\\ x_2\\ x_3\\ x_4\\ x_5 \end{array}\right| & = & \left|\begin{array}{c} 1\\ 0\\ 1\\ 0\\ 0\\ 1 \end{array}\right| \end{array}$$

k *j* *k*

Next input signal into LFSR

We are looking for the values of x_i



Deterministic Synthesis of LFSR

Generation of the polynomial and seed for the given test sequence

System of linear equations:

$$\begin{array}{c|c|}1 & 01111\\ 2 & 10111\\ 3 & 01011\\ 4 & 10101\\ 5 & 01010\\ 6 & 00101\end{array} \times \begin{vmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{vmatrix} = \begin{vmatrix}1\\ 0\\ 1\\ 0\\ 0\\ 1\end{vmatrix}$$

Solving the equation by Gaussian elimination with swapping of rows

Rows:

1,2,4,6

4,6

1,3

2,4

1,3,6

Results:

$$\begin{vmatrix}01000\\ 10000\\ 00100\\ 00010\\ 00001\\ 00001\end{vmatrix} \times \begin{vmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{vmatrix} = \begin{array}{|c|c}\multicolumn{2}{r}{f_k}\\ 0 & \\ 1 & f_2\\ 0 & f_3\\ 0 & \\ 1 & \\ 1 & \end{array}$$

$a_kx_1 \oplus b_kx_2 \oplus c_kx_3 \oplus d_kx_4 \oplus e_kx_5 = f_k$, $k = 1,2,..,6$

k=2 (4) 10101 0

(6) 00101 1

$(4 \oplus 6)$ **10000 1**

k=3 (1) 01111 1

(3) 01011 1

$(1 \oplus 3)$ **00100 0**

4) Solution:

x_1	x_2	x_3	x_4	x_5
1	0	0	0	1

Deterministic Synthesis of LFSR

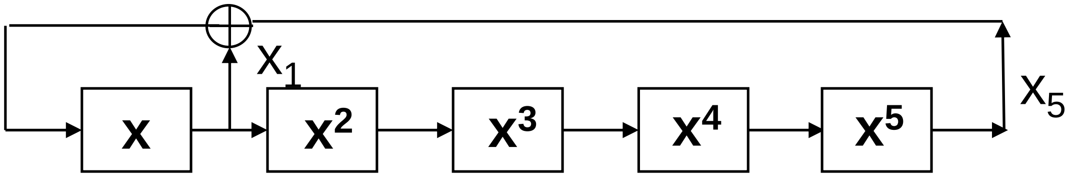
Embedding deterministic test patterns into LFSR sequence:

4) Solution: $x_1\ x_2\ x_3\ x_4\ x_5$

$1\ \ 0\ \ 0\ \ 0\ \ 1$

⇩

5) Polynomial: $x^5 + x + 1$ Seed: **01111**



Given deterministic test sequence:

(1) 100x0
(2) x1010
(3) 10101
(4) 01111

LFSR sequence:

(1) 01111 (4)
(2) 10111
(3) 01011
(4) 10101 (3)
(5) 01010 (2)
(6) 00101
(7) 10010 (1)

Exam Tasks - 1

Testability measures: probability calculation

1.Calculation of the probability of a signal (7,8)

2.Comparison of probability calculation with Parker McCluskey and linear methods (7,8)

3.Calculation of the probabilistic testability of a fault (7,9)

4.Calculation of the length of pseudorandom test for detecting a fault (7,9)

5.Calculating of signal probabilities with Cutting Method (10,11)

6.Calculating of signal probabilities with the method of Conditional Probabilities (12,13)

Exam Tasks - 2

Design for testability:

1.Comparison of test lengths for detecting a fault with and without of DFT (test point insertion) (7,9,14,25)

2.Calculation of test lengths (number of LFSR clocks) for different ad hoc designs: multiplexing of observers, de-multiplexing of control, time sharing (15-20)

3.Comparison of test lengths (number of LFSR clocks) for ad hoc and scan-based DFT solutions (15-20, 28)

Exam Tasks - 3

Built-in Self-Test:

1.Calculation of the test sequence for a given LFSR polynomials (45,49)

2.Design of LFSR reconfiguration logic for given functions (43,44)

3.Determination if the LFSR polynomial is primitive or not (46,47,48)

4.Design a LFSR for a weighted pseudorandom testing with given probabilities (54,55)

5.Synthesis of an LFSR which is able to cover a given test pattern set (77-81)

6.Synthesis of an LFSR which is able to detect a given set of faults; generate the test sequence (75, 76)