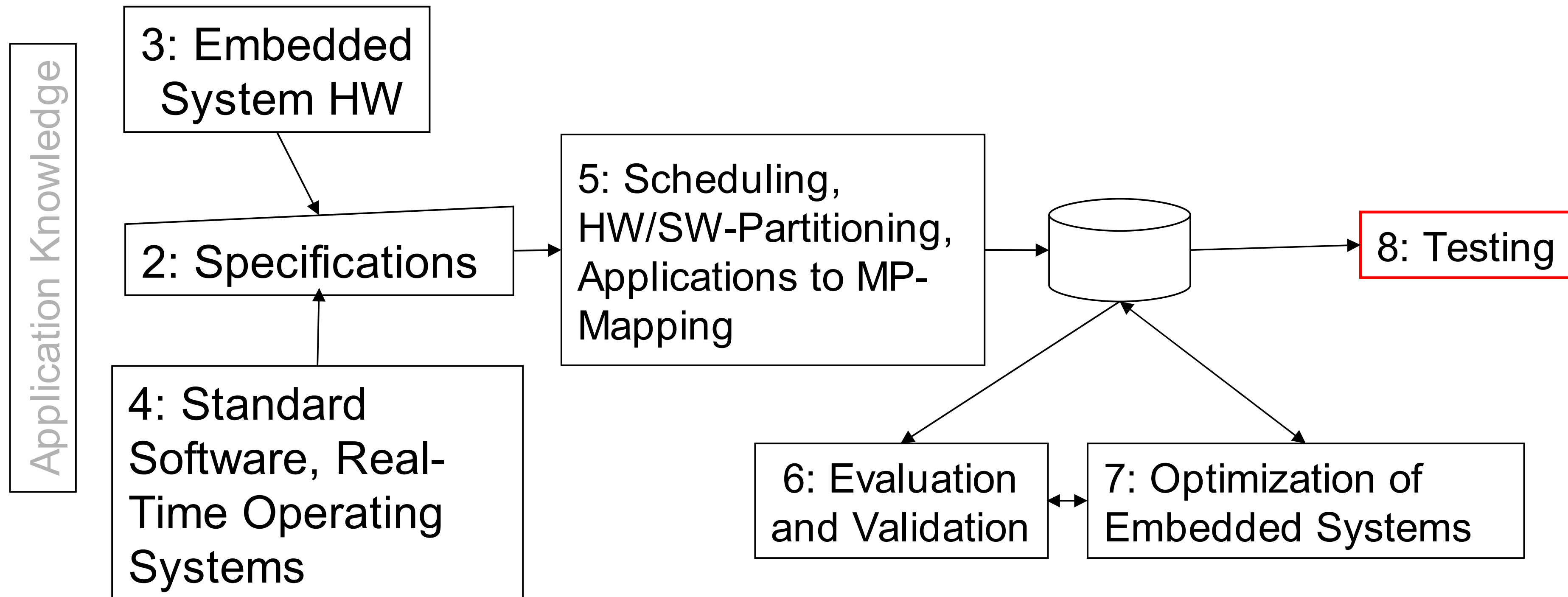
Test

Peter Marwedel
TU Dortmund
Informatik 12
Germany

2009/01/17

Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

Structure of this course



Test: Goals

1. **Production test**
2. Is there any way of using test patterns for production test already during the design*?
3. **Test for failures after delivery to customer**

* Workshop focusing on the integration of production testing and design validation: HLDVT IEEE International High Level Design Validation and Test Workshop

© p. marwedel, informatik 12, 2009

Test: Scope

Testing includes

- the application of test patterns to the inputs of the device under test (DUT) and
- the observation of the results.

More precisely, testing requires the following steps:

6. test pattern generation,
7. test pattern application,
8. response observation, and
9. result comparison.

Test pattern generation

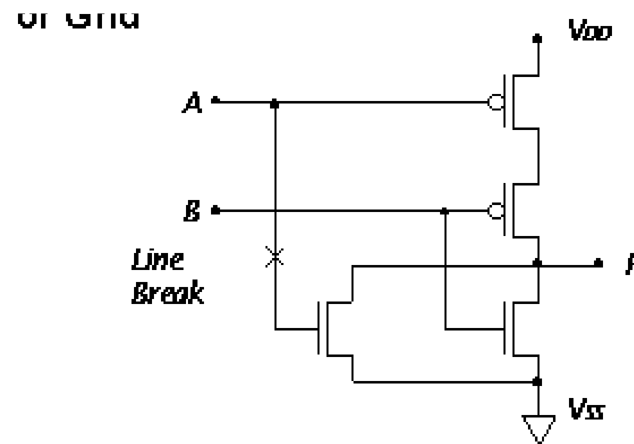
Test pattern generation typically

- considers certain fault models and
- generates patterns that enable a distinction between the faulty and the fault-free case.
- Examples:
  - Boolean differences
  - *D*-Algorithm

Fault models

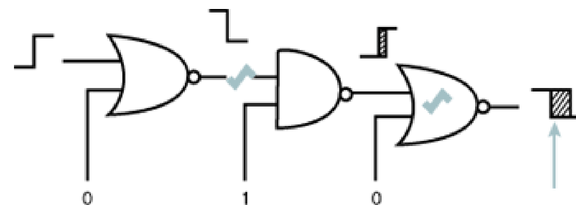
Hardware fault models include:

- stuck-at fault model (net permanently connected to ground or V_{dd})
- stuck-open faults: for CMOS, open transistors can behave like memories
- delay faults: circuit is functionally correct, but the delay is not.



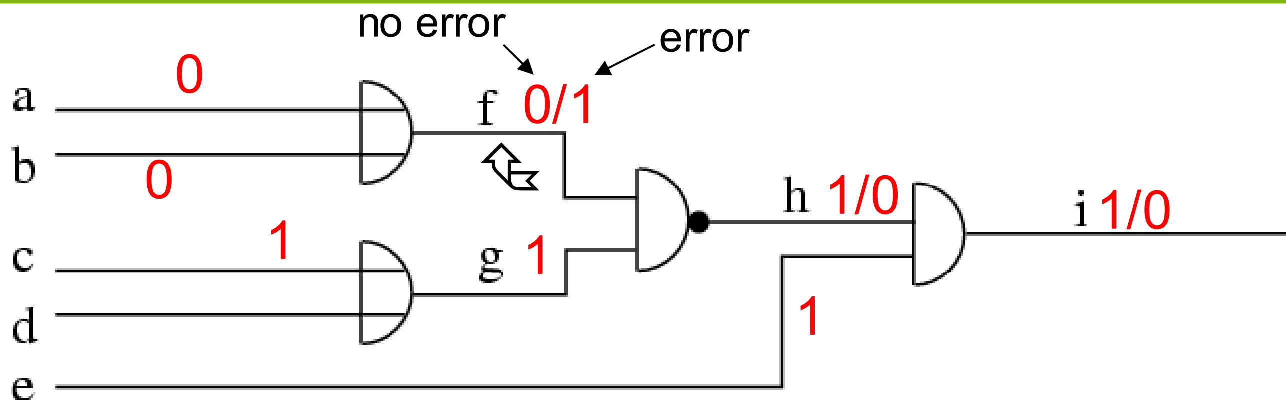
www.cedcc.psu.edu/ee497f /rassp_43/sld022.htm

- *Break above results in a "memory-effect" in the behavior of the circuit*
- *With AB=10, there is not path from either VDD or VSS to the output*
  - *F retains the previous value for some undetermined discharge time*



www.synopsys.com/products/test/tetramax_ds.html

Simple example



Could we check for a stuck at one error at a (s-a-1(a)) ?

Solution (just guessing):

- f='1' if there is an error
- ☞ a='0', b='0' in order to have f='0' if there is no error
- g='1' in order to propagate error
- c='1' in order to have g='1' (or set d='1')
- e='1' in order to propagate error
- i='1' if there is no error & i='0' if there is

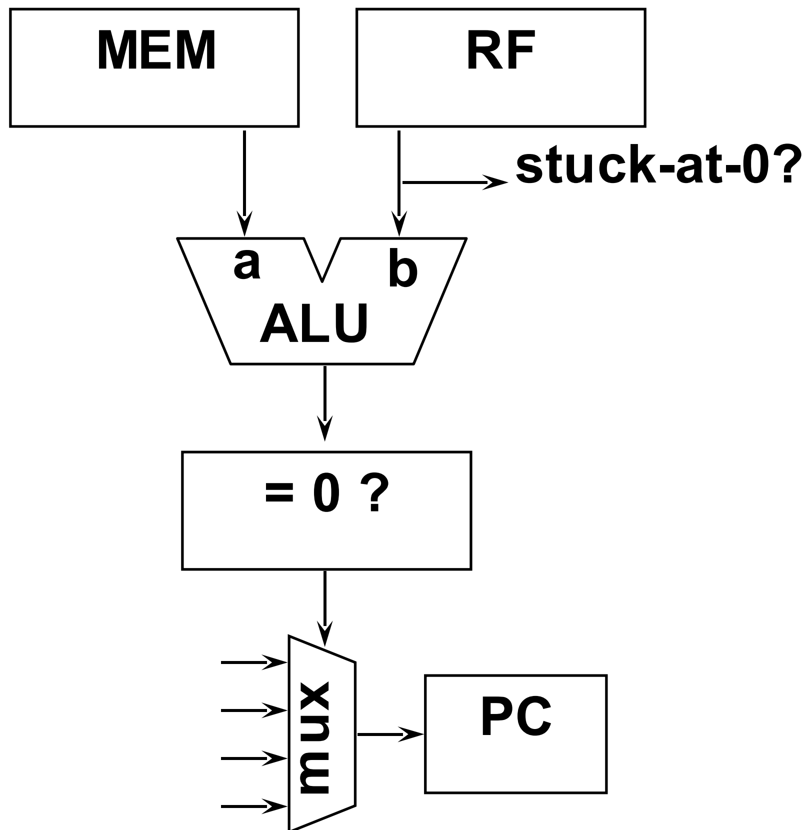
Fault coverage

A certain set of test patterns will not always detect all faults that are possible within a fault model

$$coverage = \frac{\text{Number of detectable faults for a given test pattern set}}{\text{Number of faults possible due to the fault model}}$$

For actual designs, the coverage should be at least in the order of 98 to 99%

Generation of Self-Test Program Generation - Key concept -



RF(0) := "11...1";

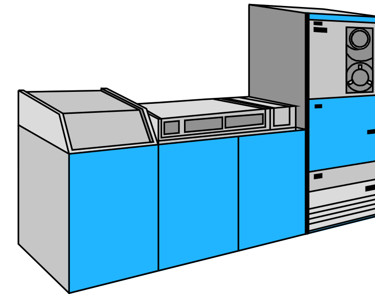
MEM(0) := "11...1";

IF MEM(0) - R(0) <>"00...0"

THEN Error;

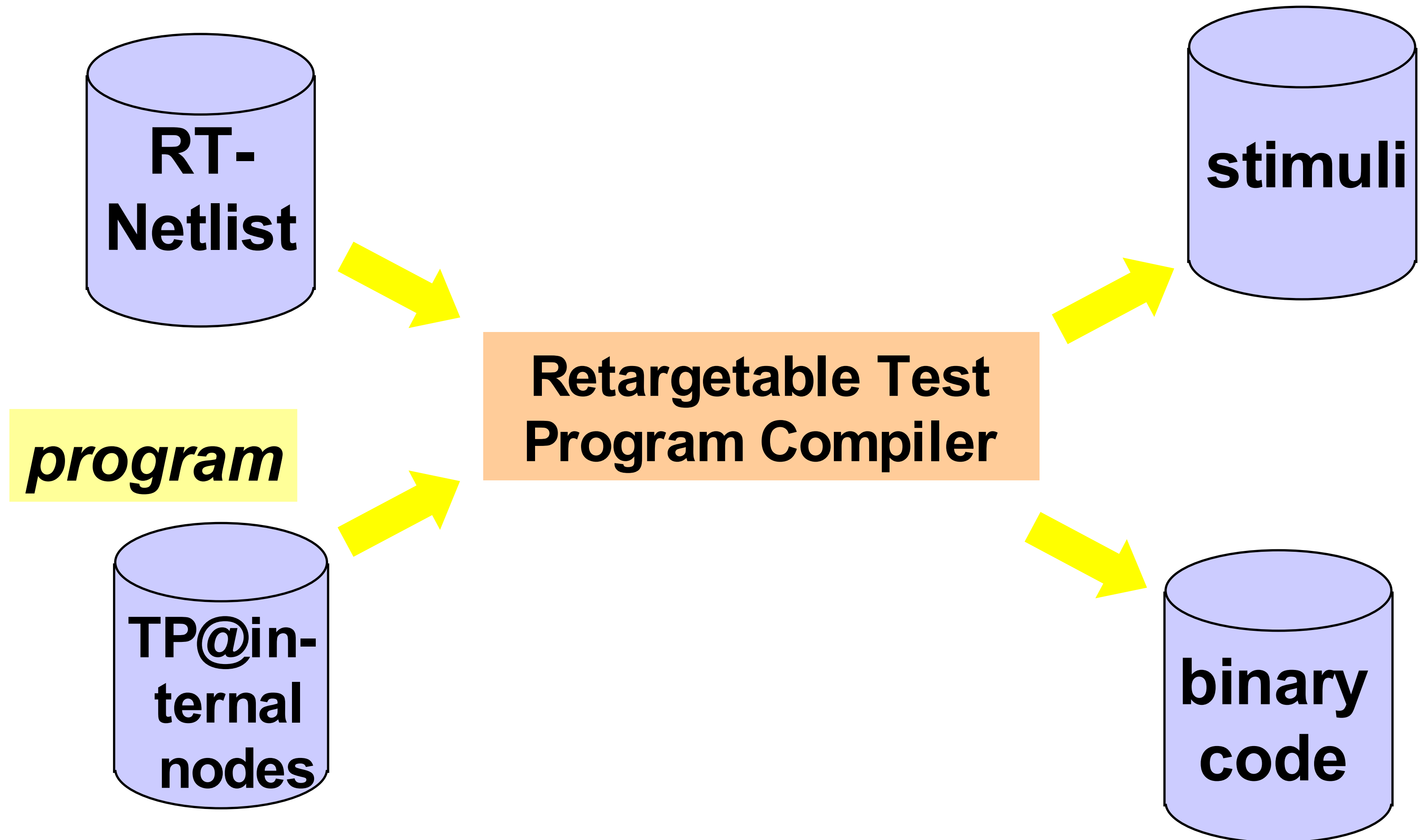
Test Program Generation (2)

- Programs running on the processors to be tested
- Well-known concept (diagnostics @ mainframes)



- Very poor tool support
- Mostly ad-hoc process:
  Initial ad-hoc program;
  Extended until sufficient coverage achieved;
  Extended if undetected errors are reported by field service

Self-Test Programs generated by Retargetable Test Compiler



[Bieker, 1995]

Fault simulation (1)

Coverage can be computed with **fault simulation**:

- ∀ faults ∈ fault model: check if distinction between faulty and the fault-free case can be made:

  Simulate fault-free system;

  ∀ faults ∈ fault model **DO**

    ∀ test patterns DO

      Simulate faulty system;

      Can the fault be observed for ≥1 pattern?

  Faults are called **redundant** if they do not affect the observable behavior of the system,

- Fault simulation checks whether mechanisms for improving fault tolerance actually help.

Fault simulation (2)

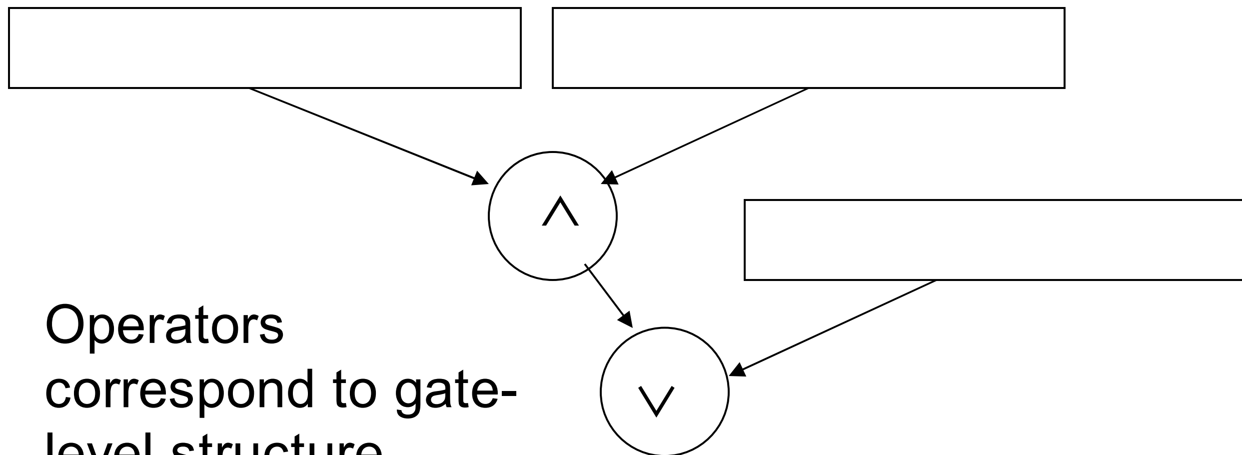
High computational requirements.

Parallel fault-simulation at the gate level:

Each bit in a word represents a different input pattern.

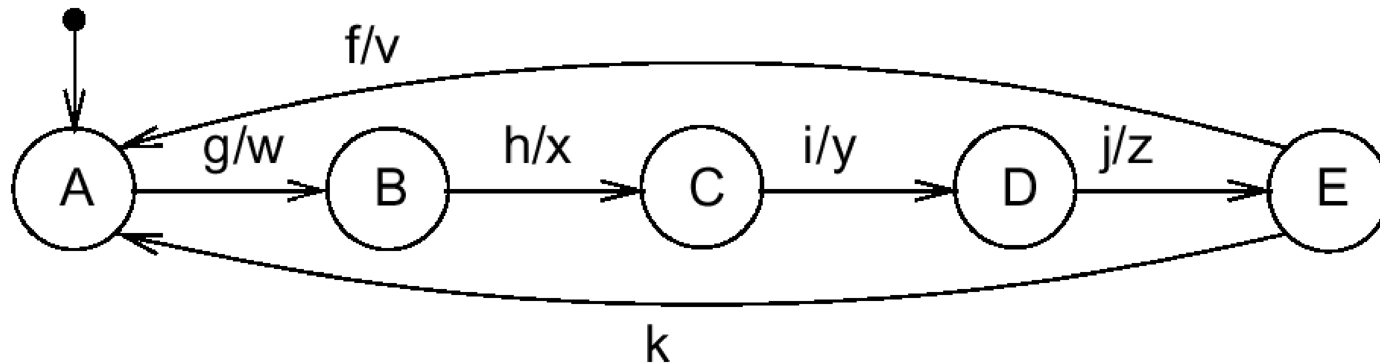
E.g.: 32 input patterns simulated at the same time.

Each bit corresponds to one test pattern



Testing finite state machines

Difficult to check states and transitions.

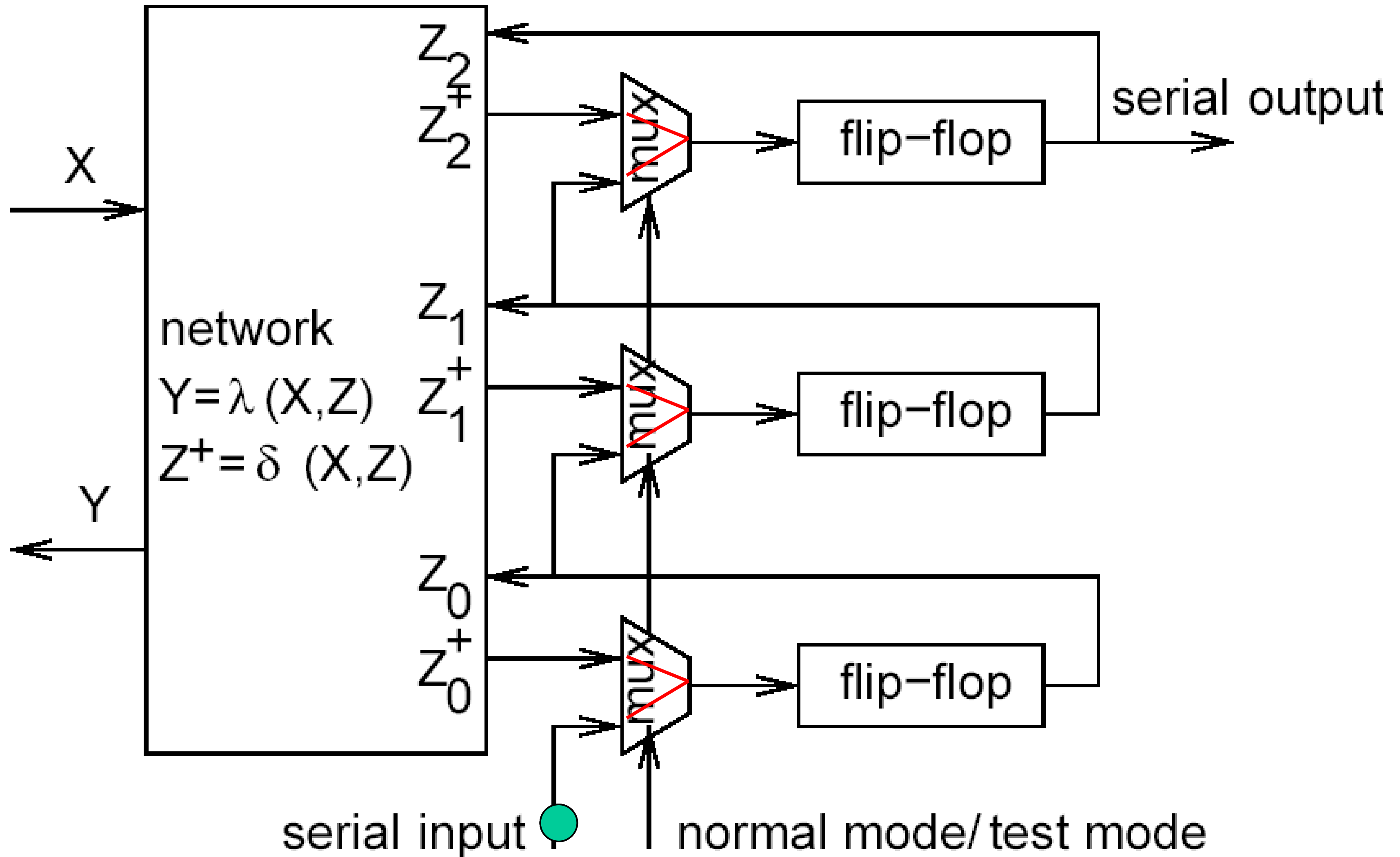


For example, verifying the transition from C to D requires

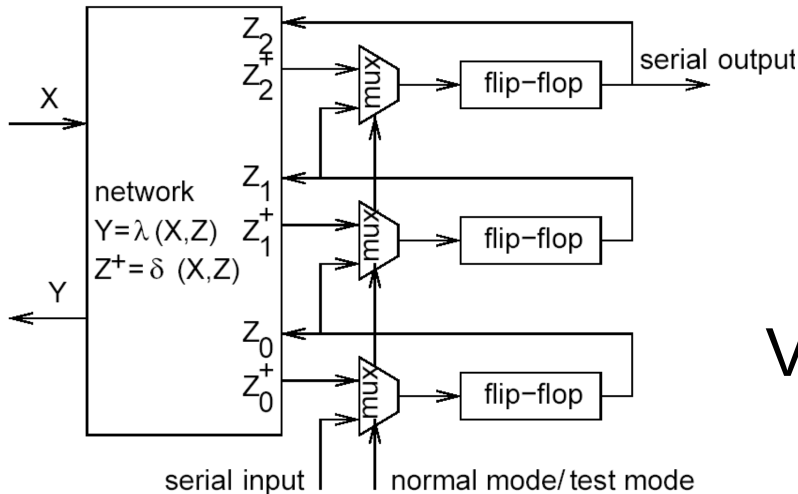
- Getting into state C
- Application of i
- Check if output is y
- Check if we have actually reached D

Simplified with scan design:

Scan design



Scan design: usage



Verifying a transition requires

- Shifting-in the „old state“
- Application of the input pattern
- Checking if output is correct
- Shifting-out the new state and comparing it.

Essentially reduced to testing combinatorial logic

JTAG (Boundary scan) (1)

JTAG defines a 4..5-wire serial interface to access complex ICs .. Any compatible IC contains shift registers & FSM to execute the JTAG functions.
TDI: test data in; stored in instruction register or in one of the data registers.
TDO: test data out
TCK: clock
TMS: controls the state of the test access port (TAP).
Optional **TRST*** is reset signal.

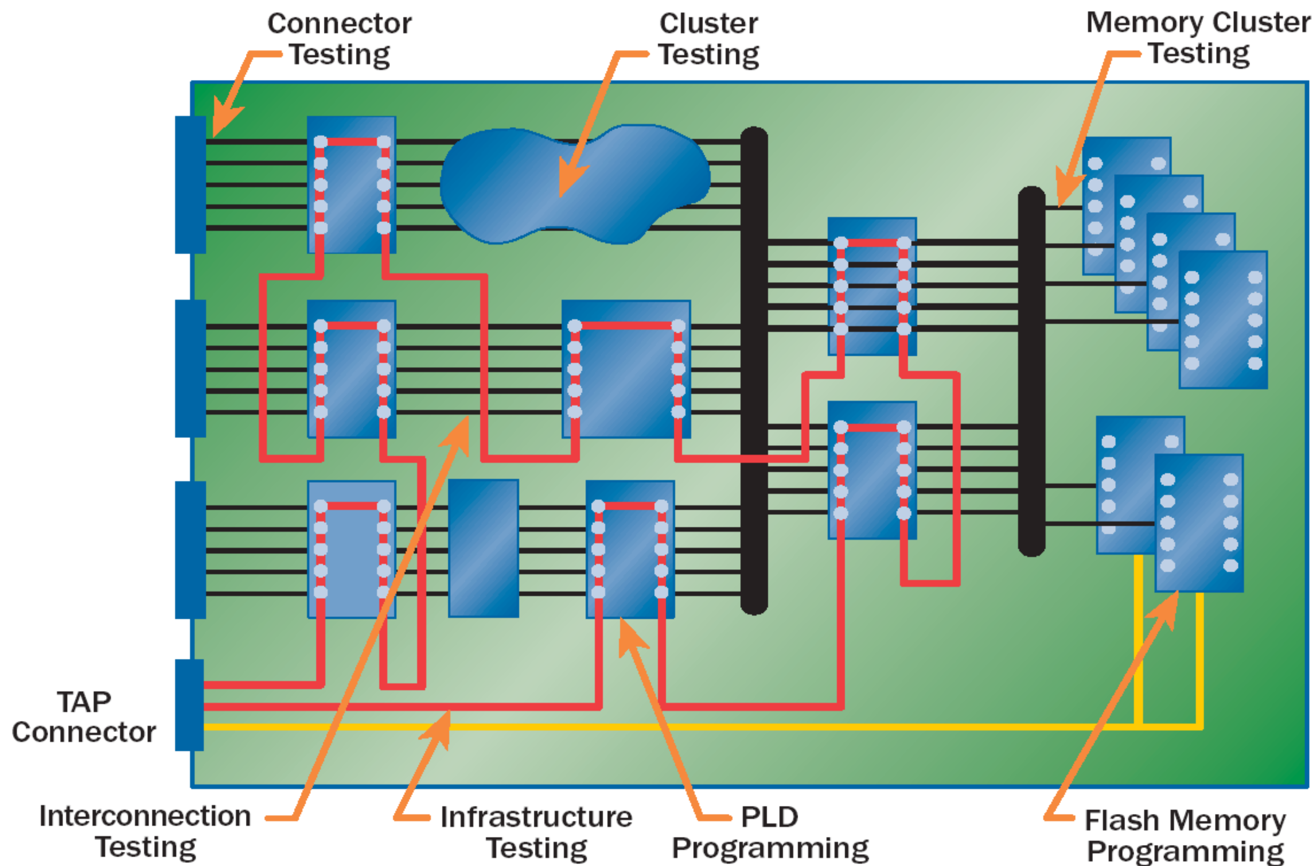
IEEE 1149.1 Device Architecture

Source: http://www.jtag.com/brochure.php

© p. marwedel, informatik 12, 2009

JTAG (Boundary scan) (2)

Defines method for setting up a scan chain on a PCB



Applications of Boundary-Scan

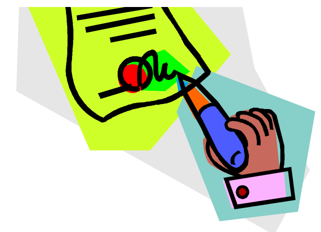
Source: http://www.jtag.com/brochure.php

Limitations of a single serial scan chain

For chips with a large number of flop-flops, serial shifts can take a quite long time.

Hence, it becomes necessary to provide several scan chains.

- Trying to avoid serial shifts by generating test patterns internally and by also storing the results internally.
- Compaction of circuit response in a **signature**.
  Shifting the entire result out becomes obsolete, we just shift out the signature.



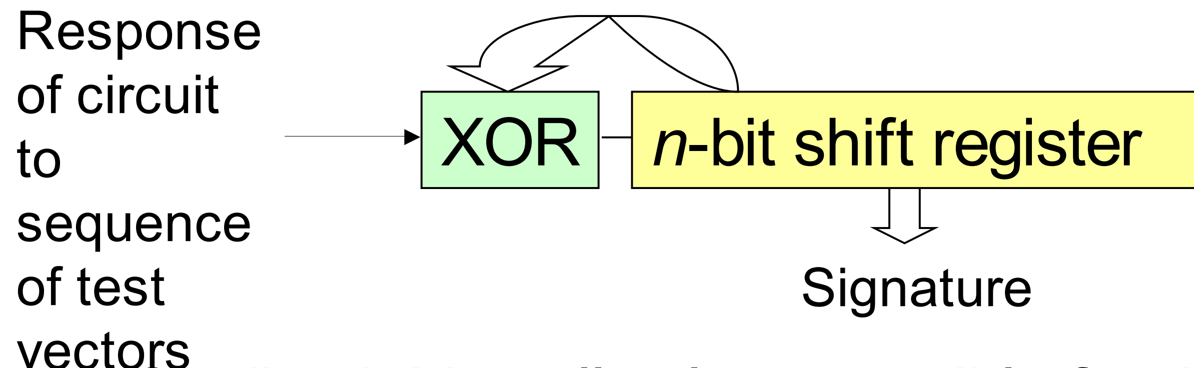
Signature analysis

Response of circuit to sequence of test patterns compacted in a signature. Only this signature is compared to the golden reference.

Exploit an *n*-bit signature register as well as possible:

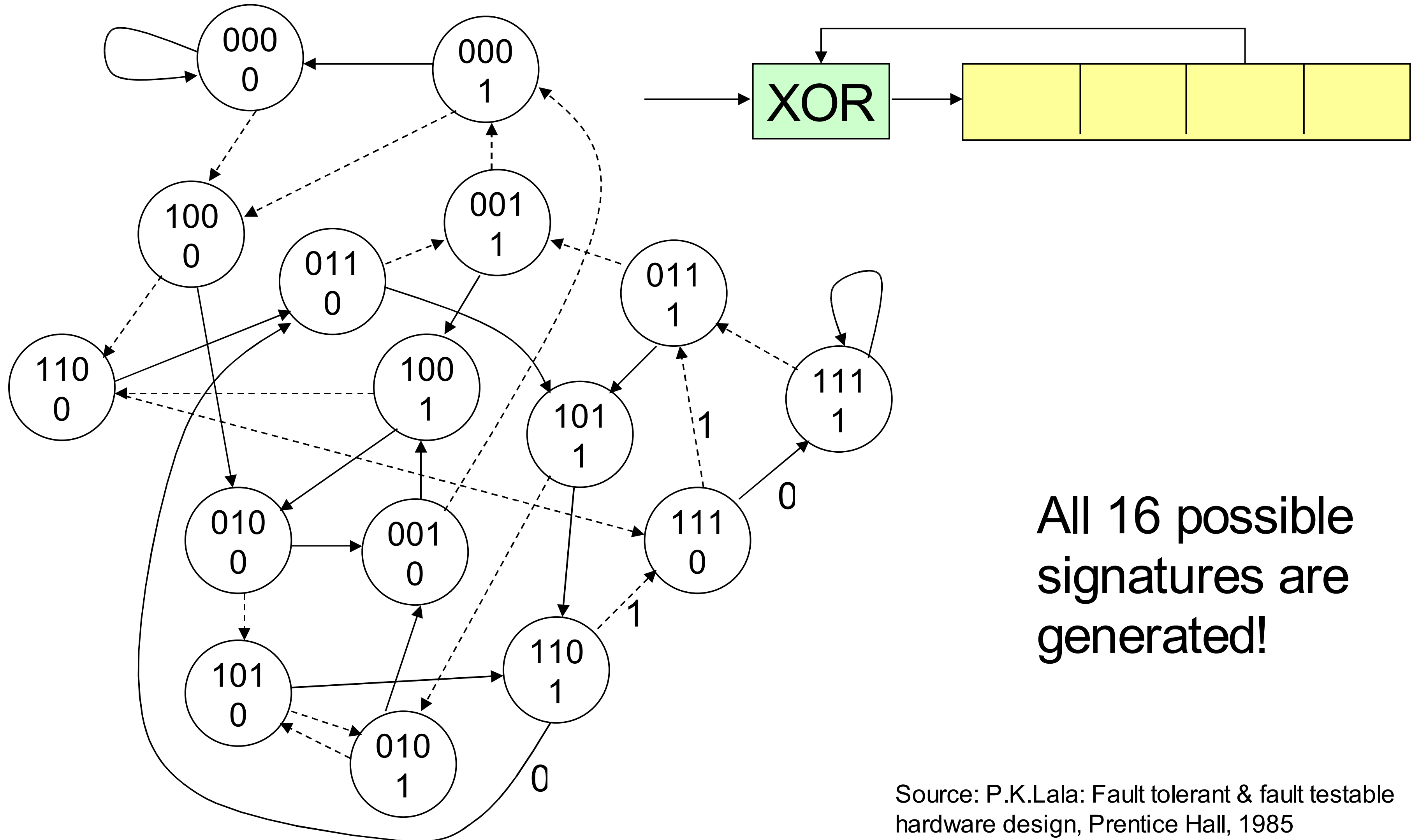
☞ try to use all values possible for that registers!

In practice, we use shift-registers with linear feedback:



Using proper feedback bits, all values possible for the register can be generated.

Example: 4-bit signature generator



Aliasing for signatures

Consider aliasing for some current pattern

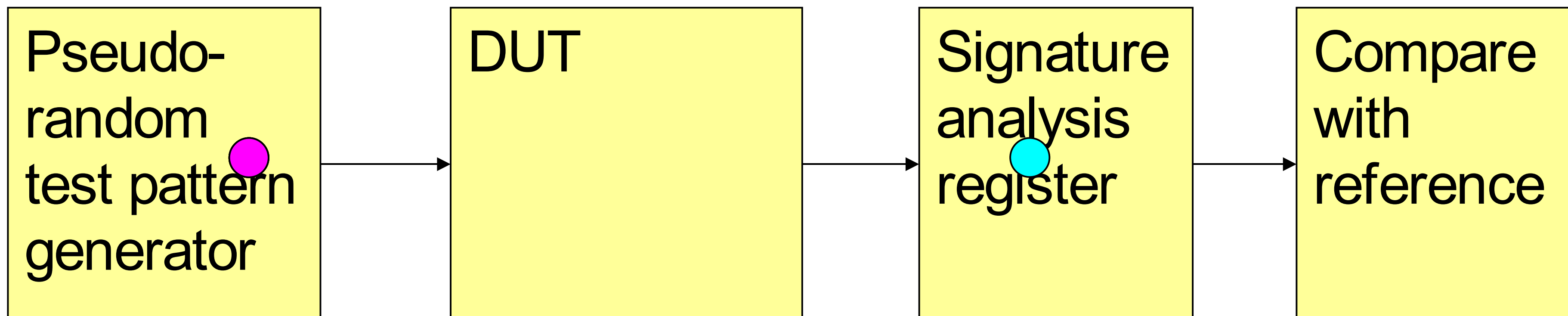
- An n-bit signature generator can generate 2^n signatures.
- For an m-bit input sequence, the best that we can get is to evenly map $2^{(m-n)}$ patterns to the same signature.
- Hence, there are $2^{(m-n)}-1$ sequences that map to the same signature as the pattern currently considered.
- In total, there are 2^m-1 sequences different from the current one.

☞ $$P = \text{Probability}\left(\frac{\text{other patterns map to same signature}}{\text{total number of other patterns}}\right) = \frac{2^{(m-n)}-1}{2^m-1}$$

$$P = \frac{1}{2^n} \text{ for } m >> n \text{ provided that we evenly map patterns to signatures}$$

Replacing serially shifted test pattern by pseudo random test patterns

Shifting in test patterns can be avoided if we generate (more or less) all possible test patterns internally with a pseudo-random test pattern generator.



Effect of pseudo random numbers on coverage to be analyzed.
Signature analysis register shifted-out at the end of the test.

Pseudo random test pattern generation

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

XOR

Linear feedback shift register (LFSR)

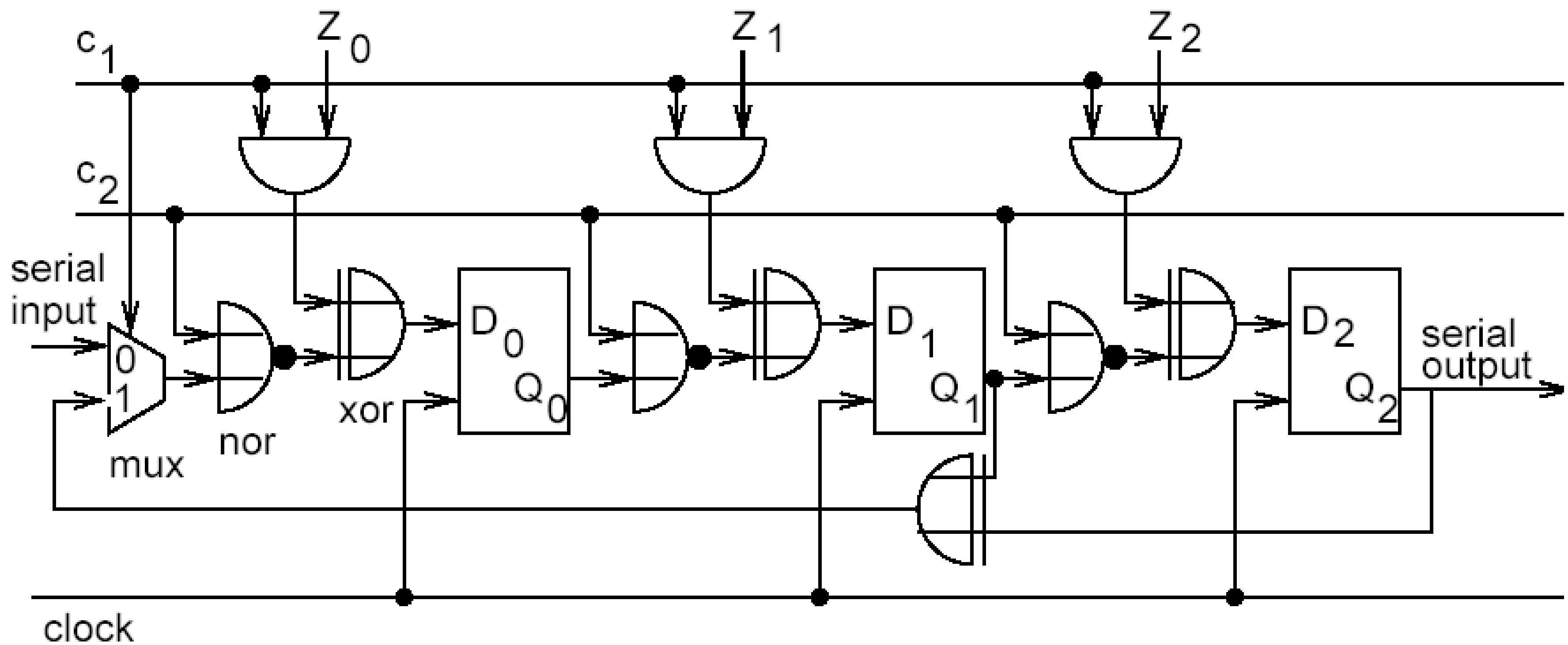
2^n-1 patterns generated!

Superior to counters if not all patterns are used.

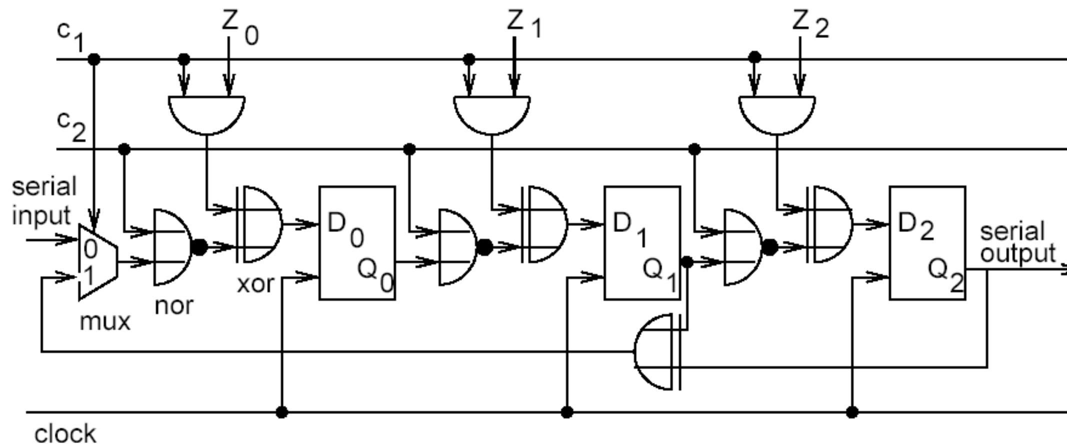
Combining signature analysis with pseudo-random test patterns: Built-in logic block observer (BILBO)

Könemann & Mucha

Uses *parallel* inputs to compress circuit response

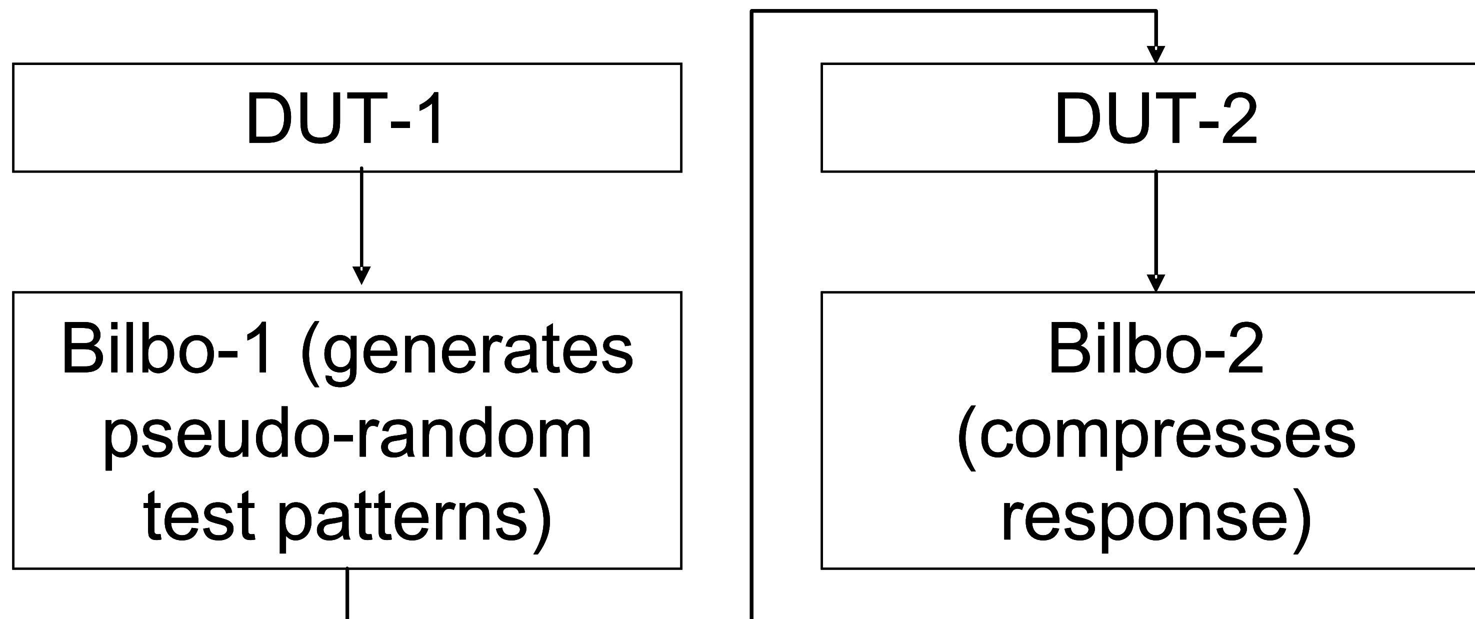


Built-in logic block observer (BILBO)
Modes of operation



c_1	c_2	D_i	
'0'	'0'	$'0' \oplus \overline{Q_{i-1}} = \overline{Q_{i-1}}$	scan path mode
'0'	'1'	$'0' \oplus \overline{'1'} = '0'$	reset
'1'	'0'	$Z_i \oplus \overline{Q_{i-1}}$	LFSR mode
'1'	'1'	$Z_i \oplus \overline{'1'} = Z_i$	normal mode

Typical application



Compressed response shifted out of Bilbo-2 & compared to known „golden“ reference response.

Roles of Bilbo-1 and 2 swapped for testing DUT-1

Summary

- Test
  - Fault model
  - TPG (…, generation of assembly progrograms, ..)
  - Application of test patterns
  - Checking the results
  - Fault coverage
  - Fault simulation for computing coverage
- Design for Test (DFT)
  - Scan path, Boundary scan
  - Signature analysis,
  - Pseudo random patterns, BILBO

The end