EDA322
Digital Design

Lecture 15:

Faults, Verification and Testing

Ioannis Sourdis

Outline of Lecture 15

- Verification vs. Testing
- Faults & Fault Tolerance
- Defects and Yield
- Fault models
  - Stuck-at Faults
  - Fault equivalence
- Testing
  - Combinational and Sequential circuits
  - ATPG
- Design for testing
  - Scan-chains– JTAG
  - Built-in Self-Test (BIST)
  - Boundary-scan

What can go wrong on digital circuits?

- **Design bugs**:
  - What to do: **Verification**
- **Faults**:
  - Permanent (implementation) faults
    - What to do: (Design for) **Testing**
  - Other (Transient) Faults
    - What to do: (Design for) **Fault Tolerance**

Verification vs Testing

Verification vs. Testing

- Verification is the task of ensuring that a design meets its specification
  - Looking for design mistakes

- Testing is performed to ensure that a particular instantiation of a design functions properly
  - Looking for implementation faults

Motivation for verification

- On a typical digital system project more effort is expended on verification than on the design itself.
- Intel Pentium float point Division bug (1994)
- Recall all chips
- Estimated cost $500M

Motivation for testing

Faulty circuits/boards/chips

Faulty boards	Faulty systems	Cost
5%	5000	$1million
1%	1000	$200,000
0.1%	100	$20,000
0.01%	10	$2,000

The error is detected during testing of the circuit board

Faulty boards	Faulty systems	Cost
5%	5000	$5million
1%	1000	$1million
0.1%	100	$100,000
0.01%	10	$10,000

The fault is detected at customer

Verification coverage

- The set of test patterns (test suit) written to verify a design should be complete.
- Specification coverage
  - All the features of the design have to be specified
- Implementation coverage
  - Every line of the VHDL code should be checked.
  - Examples
    - Every case of a *case* statement, should be activated.
    - For every state machine in our design, every edge between states should be traversed.

Types of verification

- Static timing analysis
  - Setup and hold time margins.
  - Synthesis Tool or separate tool.
- Formal verification
  - Proof techniques to verify the functional correctness without the need for simulation (equivalence checking).
  - Often used to verify protocols.
- Bug tracking system
  - Keep track of all bugs that are identified by tests.

Faults and Fault tolerance

Types of Faults

- Transient faults
  - Faults that happen only once
    - and it’s VERY unlikely to happen again
  - Causes:
    - Electromagnetic Interference
      - Neighbors mobile phone
      - Static electricity
    - Various particles hitting the silicon surface
      - Heavy ions such as iron, α-particles, neutrons.
    - Internal effects
      - Crosstalk, metastability, power supply disturbances
- Permanent faults
  - Faults that are always there
  - Causes:
    - Design defects
    - Manufacturing defects
    - Transistor aging
- Intermittent faults
  - Faults that come and go (probably periodically)
  - Causes: Variations
    - **Static**: transistors on a chip may not be exactly the same although they were supposed to be
    - **Dynamic**: temperature changes

Faults and CMOS Technology

As Technology Scales chips are becoming less reliable

- Harder to keep # of permanent faults low
  - Transistors become smaller
- Variations become more severe
  - Static: Transistors are different
  - Dynamic: Temperature,
- Transistor aging is accelerated
  - The more they are used the easier they "break"
- Soft-error rate grows exponentially
  - More transistors on-chip, higher probability for a bit-flip

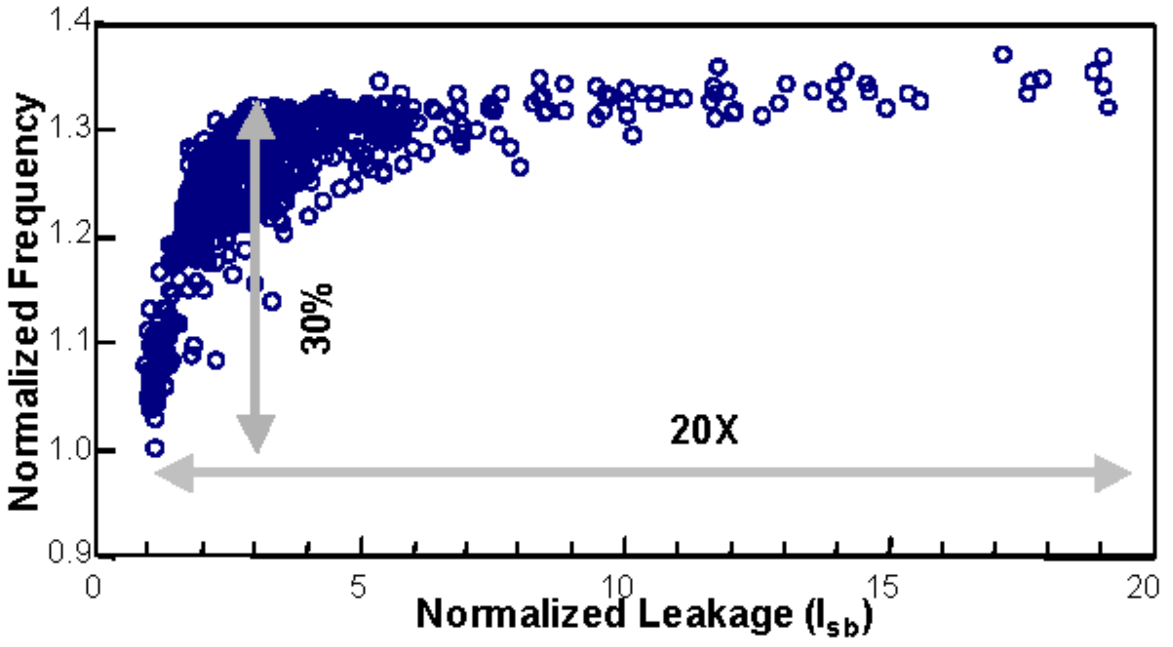
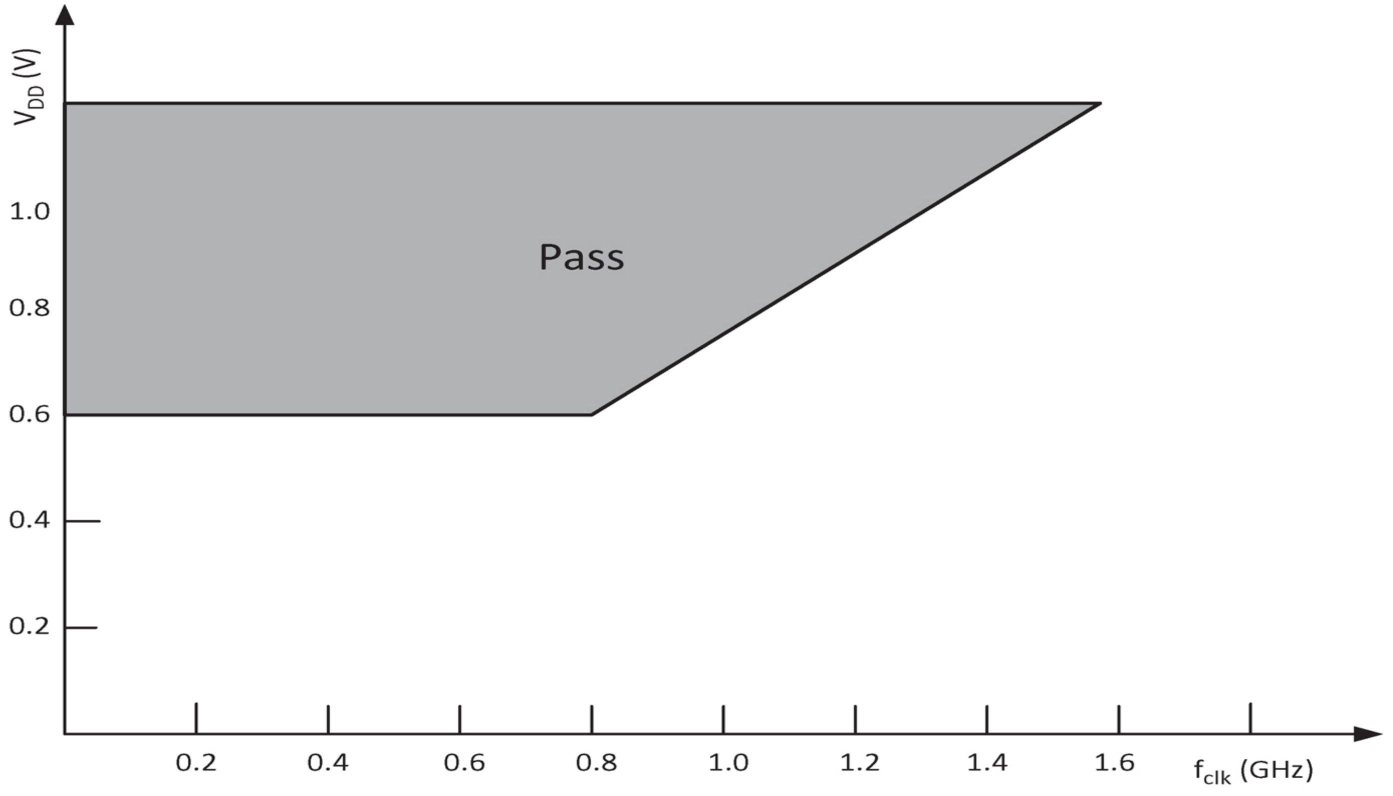


Figure 5. Soft-error failure-in-time of a chip (logic and memory).

Transistor's Saturation current

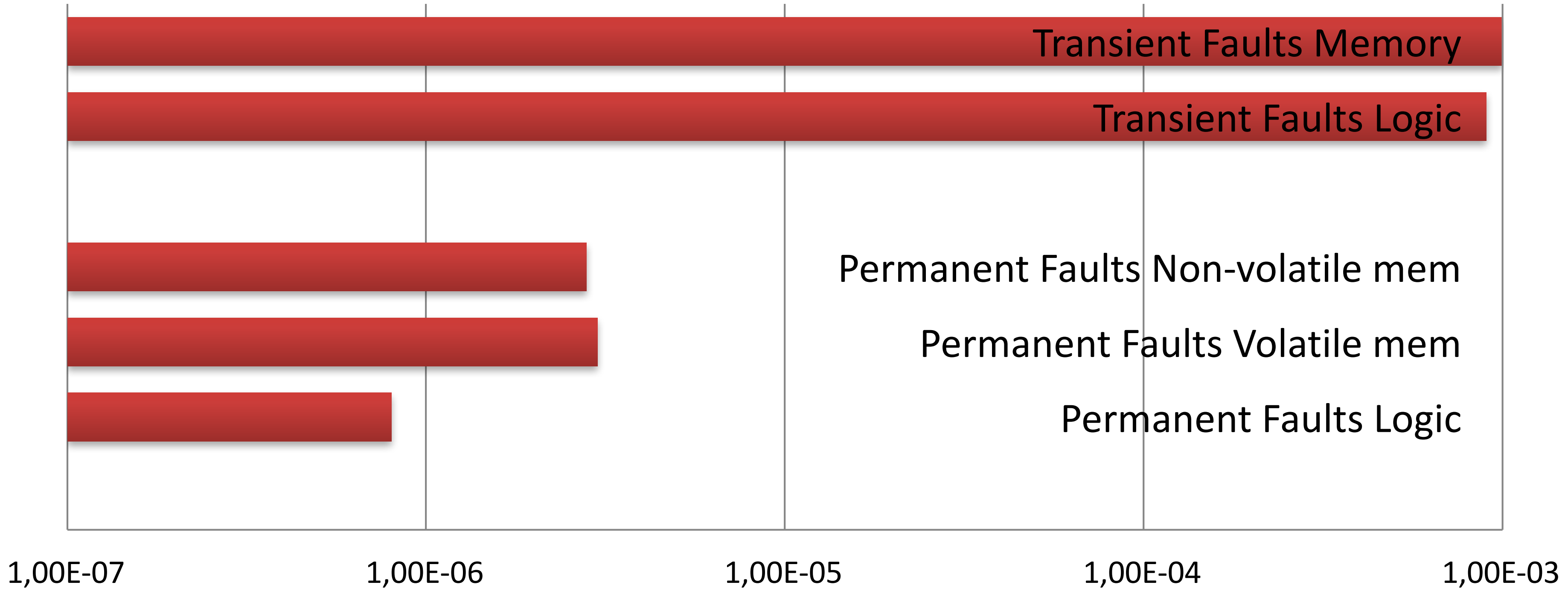
Characterization

- The operating envelop of a chip is the range of supply voltage and clock frequency over which a chip operates properly.



Fault Rate (FIT)

- Failure in Time (FIT)
  - 1 FIT = 1 failure per billion (10^9) hours (114,155 years)



Detecting Transient faults

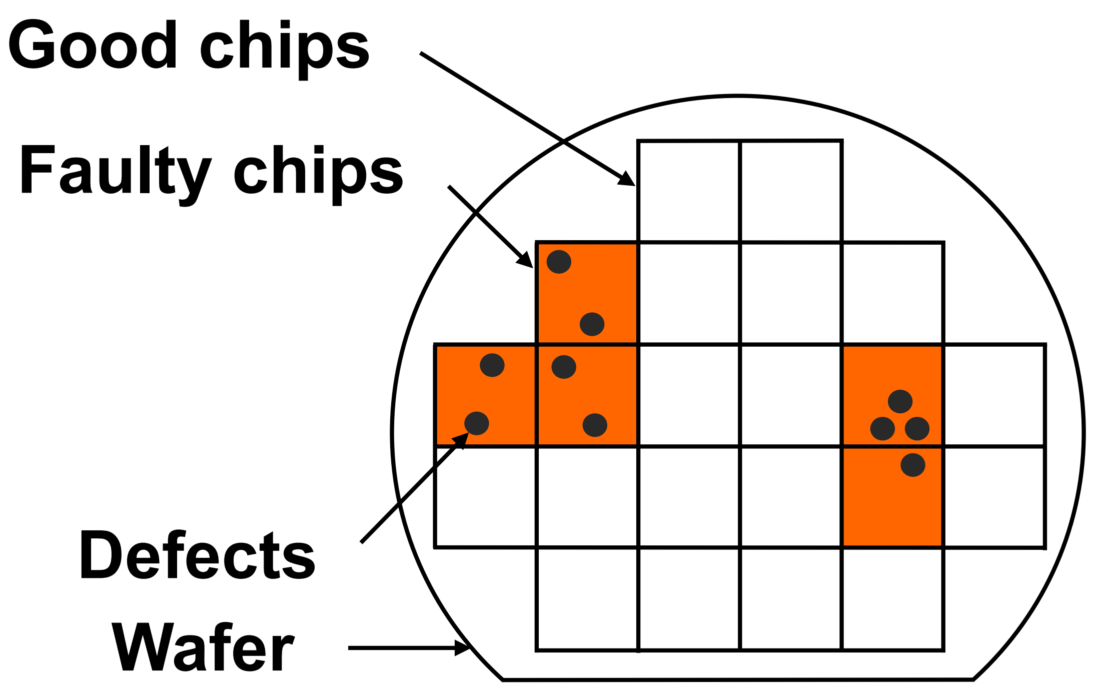
- Error correcting Codes (ECC)
  - Parity, checksums
- Time or Space Redundancy
  - Code executed at least twice
- Watchdog-mechanisms
  - A timer which is reset as long as the system is "alive"
  - It resets the system when reaches a "timeout" state (if the main program does not respond due to some fault condition such as a hang.
  - The intention is to bring the system back from the hung state into normal operation.

Detecting Permanent faults

- Production testing (off-line testing)
  - Process testing
    - Testing the parameters and the precision in manufacturing
  - Reliability testing
    - Varying temperature, supply voltage changes
  - Functional testing
  - Electrical testing
    - "At-speed"-testing to determine the maximum speed
- On-line testing
  - Small lightweight tests to detect whether a fault is permanent, while the chip is working

Defects and Yield

Circuit Fabrication and Defects



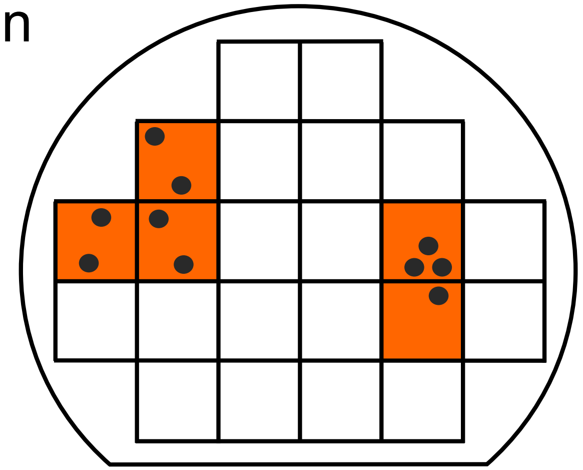
Defects are permanent faults caused during the Manufacturing process

$$\text{yield} = \frac{\text{number of working chips produced}}{\text{Total number of chips produced}}$$

Wafer yield = 17/22 = 0.77

VLSI Chip Yield

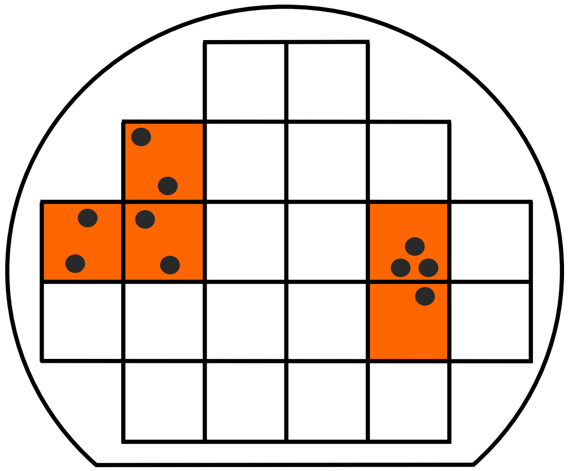
- A **chip** is an area on a Silicon wafer that contains millions, or billions of transistors and wires/interconnects.
- A **manufacturing defect** is a finite chip area with electrically malfunctioning circuitry caused by errors in the fabrication process.
- A chip with no manufacturing defect is called a **good chip**.
- Fraction (or percentage) of good chips produced in a manufacturing process is called the ***yield***. Yield is denoted by symbol ***Y***.
- Cost of a (working) chip:



$$\frac{\text{Cost of fabricating and testing a wafer}}{\text{Yield x Number of chip sites on the wafer}}$$

Yield

- Manufacturing yield is a function of three parameters:
    - Chip area, A
    - Defect density (defects per unit area), d
    - Fault clustering parameter, α (*alpha*)
- $Y = (1 + Ad/\alpha)^{-\alpha}$
- Typical values:
    - $A = 1$ cm^2 for a processor chip
    - $d = 1$ defect/cm^2 for a matured process
    - $\alpha = 0.2 - 0.5$



Yield Equation

Y = Prob (zero defect on a chip) = p (0)

$$\boxed{Y = (1 + Ad / \alpha)^{-\alpha}}$$

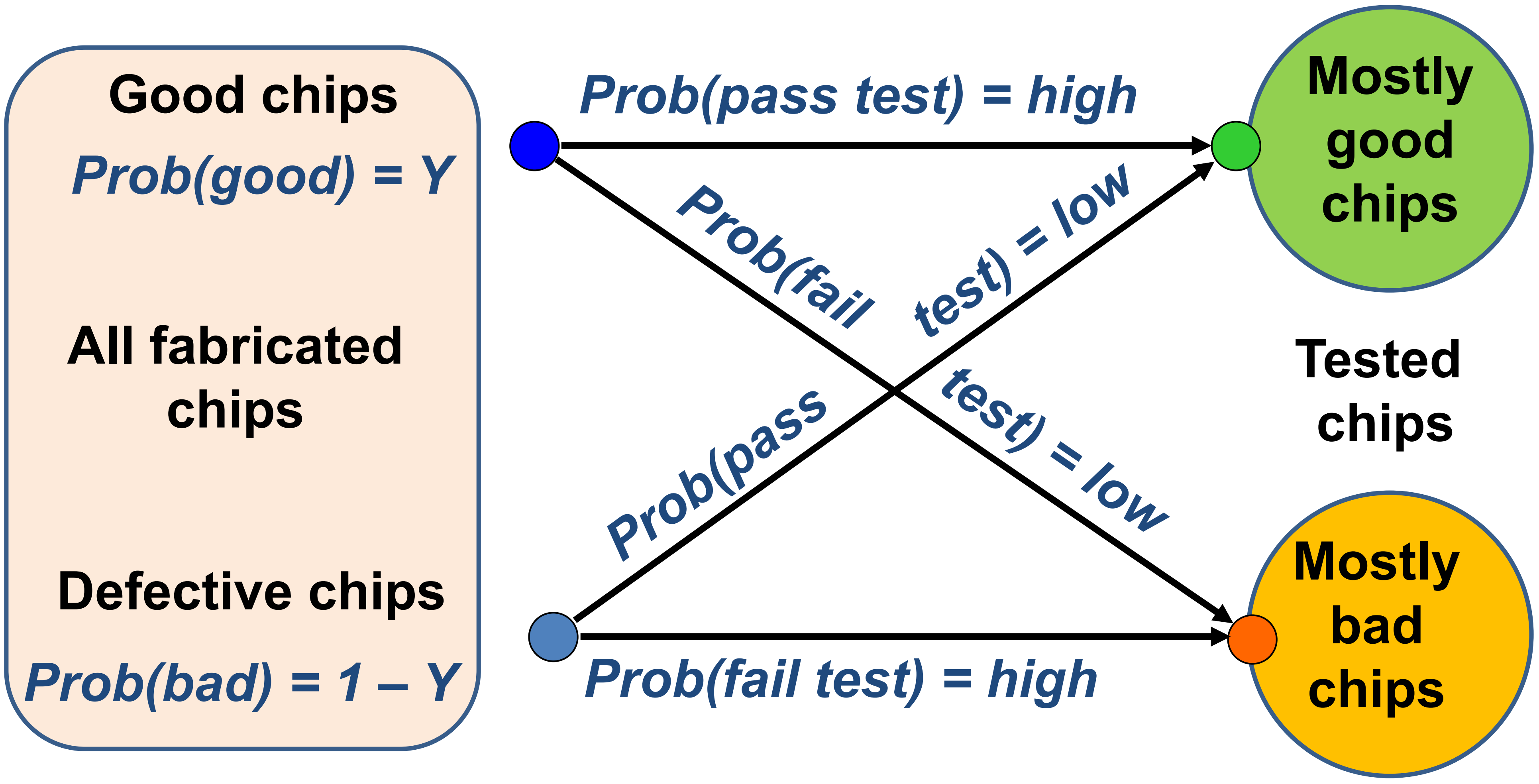
Example: $Ad = 0.5$, $\alpha = 0.5$, $Y = 0.71$

Unclustered defects: $\alpha = \infty$, $Y = e^{-Ad}$

Example: $Ad = 0.5$, $\alpha = \infty$, $Y = 0.61$

too pessimistic !

Testing Separates Good from Bad

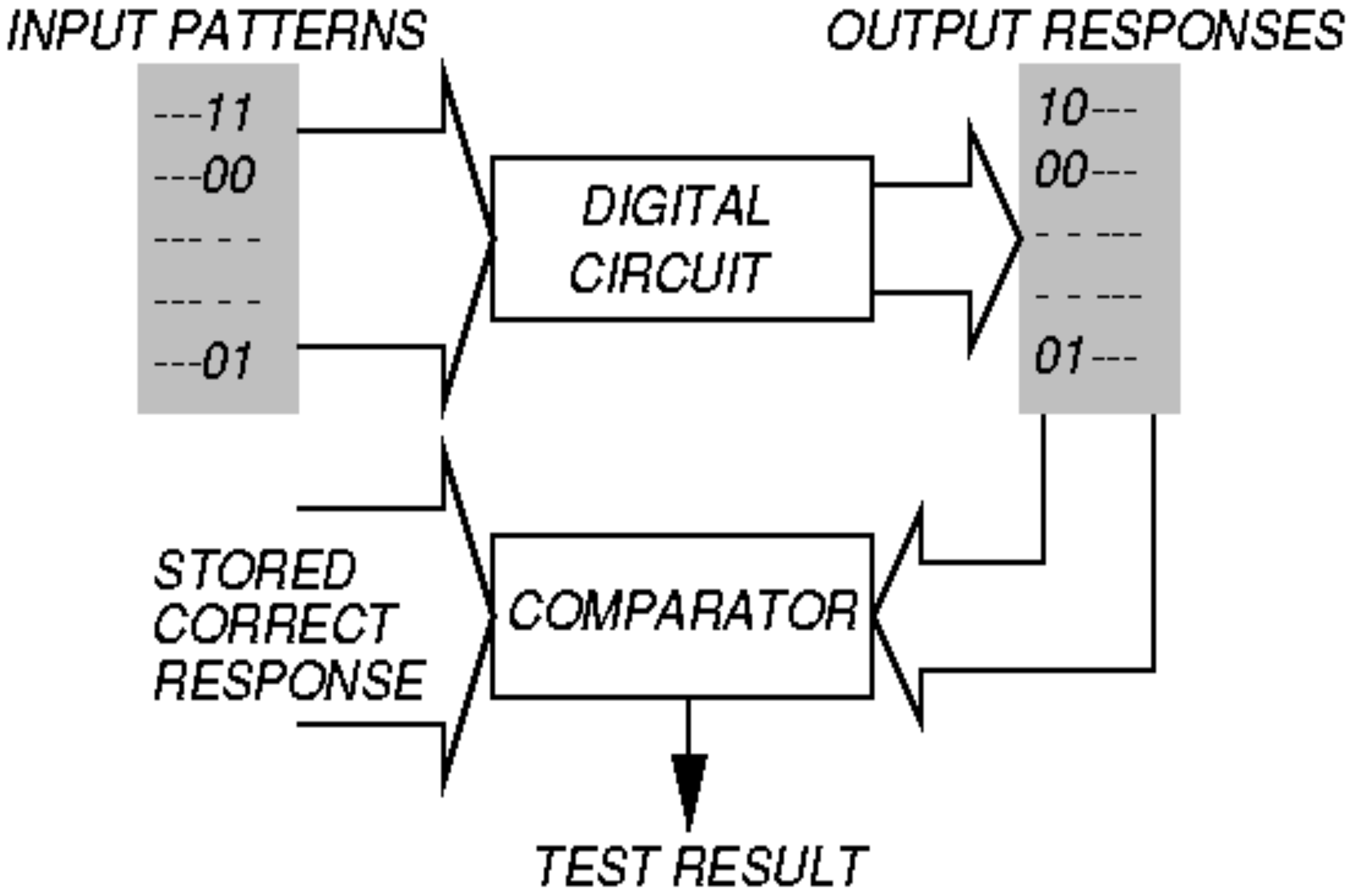


Defect Level or Reject Ratio

- *Defect level* (DL) is the ratio of faulty chips among the chips that pass tests.
- DL is measured as *parts per million* (ppm).
- DL is a measure of the effectiveness of tests.
- DL is a quantitative measure of the manufactured product quality. For commercial VLSI chips a DL greater than 500 ppm is considered unacceptable.

Testing

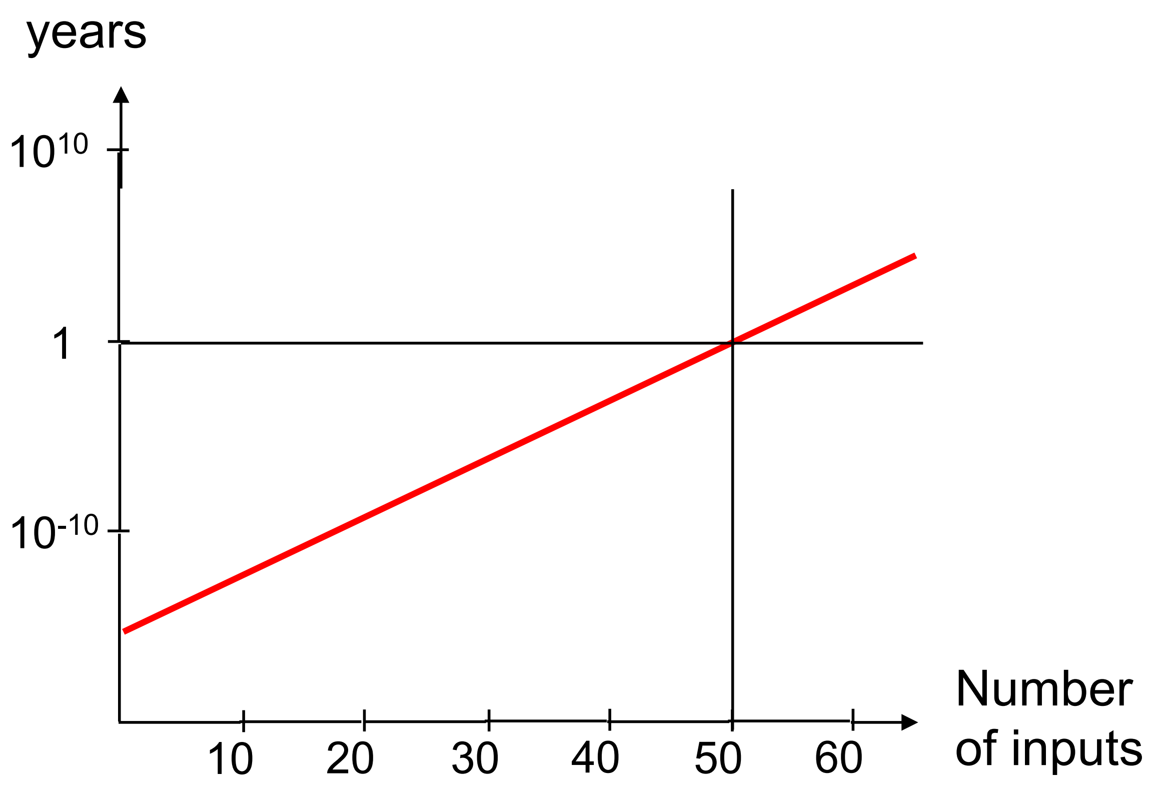
How to Test?


INPUT PATTERNS
---11
---00
--- - -
--- - -
---01
DIGITAL CIRCUIT
OUTPUT RESPONSES
10---
00---
- - ---
- - ---
01---
STORED CORRECT RESPONSE
COMPARATOR
TEST RESULT

Types of tests

- Exhaustive
  - How many tests do we need to check for a 64-bit adder ?
- Directed tests and Random tests
  - Check an add that produces the largest positive (negative number) and an add one more than this to check for overflow.
  - Add two random input operands
- Non-uniform random tests around area of interest
- Auttomatic Test Pattern Generation (ATPG)

(Exhaustive) Test time as a function of number inputs to a circuit



30 inputs take 2s
40 inputs take1h
50 inputs take1year

(Test speed 500MHz)

Why Compromise on Testing?

- Complexity and cost.
- Example: A digital circuit with 64 inputs (such as a 32-bit adder).
- There are 2^{64} possible inputs
  - 2^{64} = **18,446,744,073,709,551,616** possible input patterns or vectors.
- If we can apply one pattern in 1ns, then time to test the circuit with all patterns is
    - **$2^{64} \times 10^{-9}$ /3600 = 5124095.6 hours**
    - **Or 213504 days**
    - **Or 585 years**

More realistic methods

- Exhaustively testing all inputs is too expensive/slow
- A subset of input values is used
- This subset of inputs is determined, as follows:
  - **Random test vectors**
  - **Based on the logical function (functional testing)**
    - E.g. check boundary conditions which make your circuit operate differently
  - **Based on the design structure (structural testing)**
    - E.g. partition your hardware in smaller blocks (exploit the hierarchy of your hardware design)
  - **Based on the fault model**
    - Group a huge number of things that can go wrong in a much smaller finite set of fault models

Fault Models

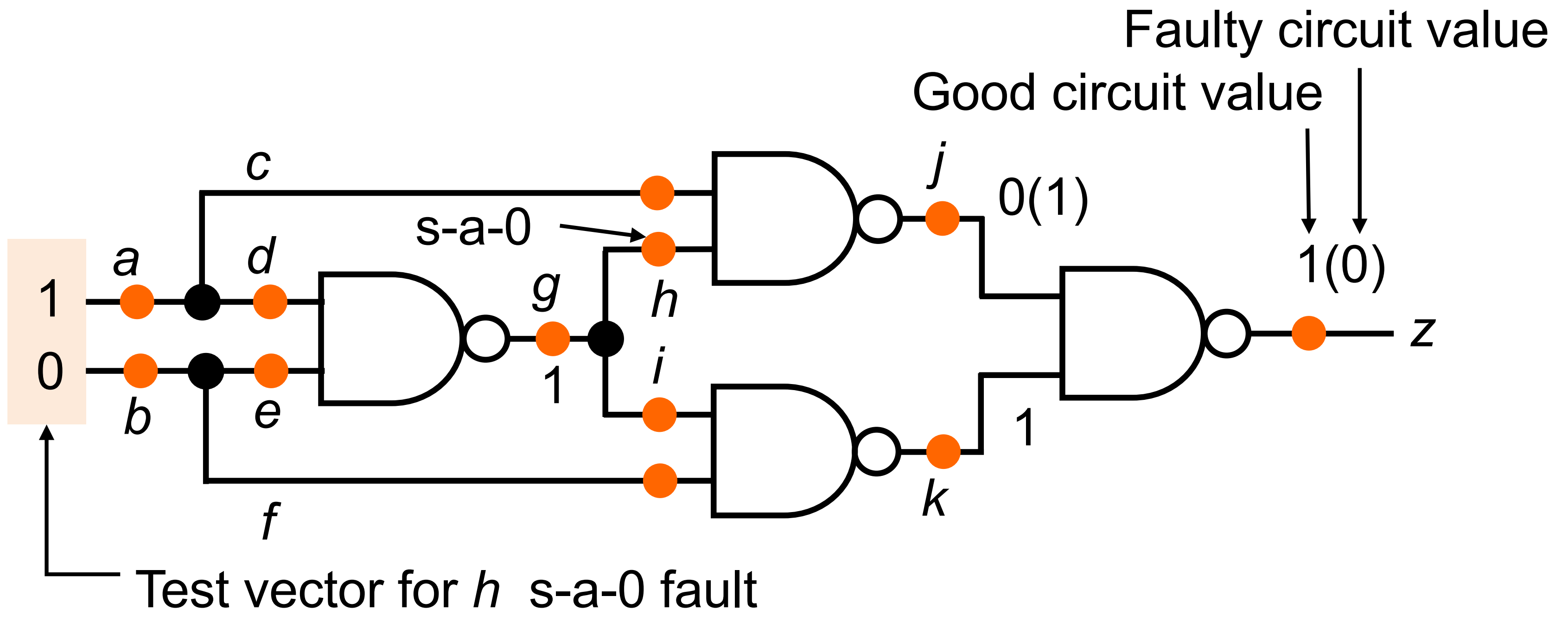
Fault models

- A fault model is an abstract model of physical faults that could cause a chip not to work.

- Stuck-at fault model: This model abstracts all potential faults as resulting in a logical node of the circuit being either stuck at logic “0” or stuck at logic “1”.

Single Stuck-at Fault

- Three properties define a single stuck-at fault
  - **Only one line is faulty**
  - **The faulty line is permanently set to 0 or 1**
  - **The fault can be at an input or output of a gate**
- Example: XOR circuit has 12 fault sites (●) and 24 single stuck-at faults

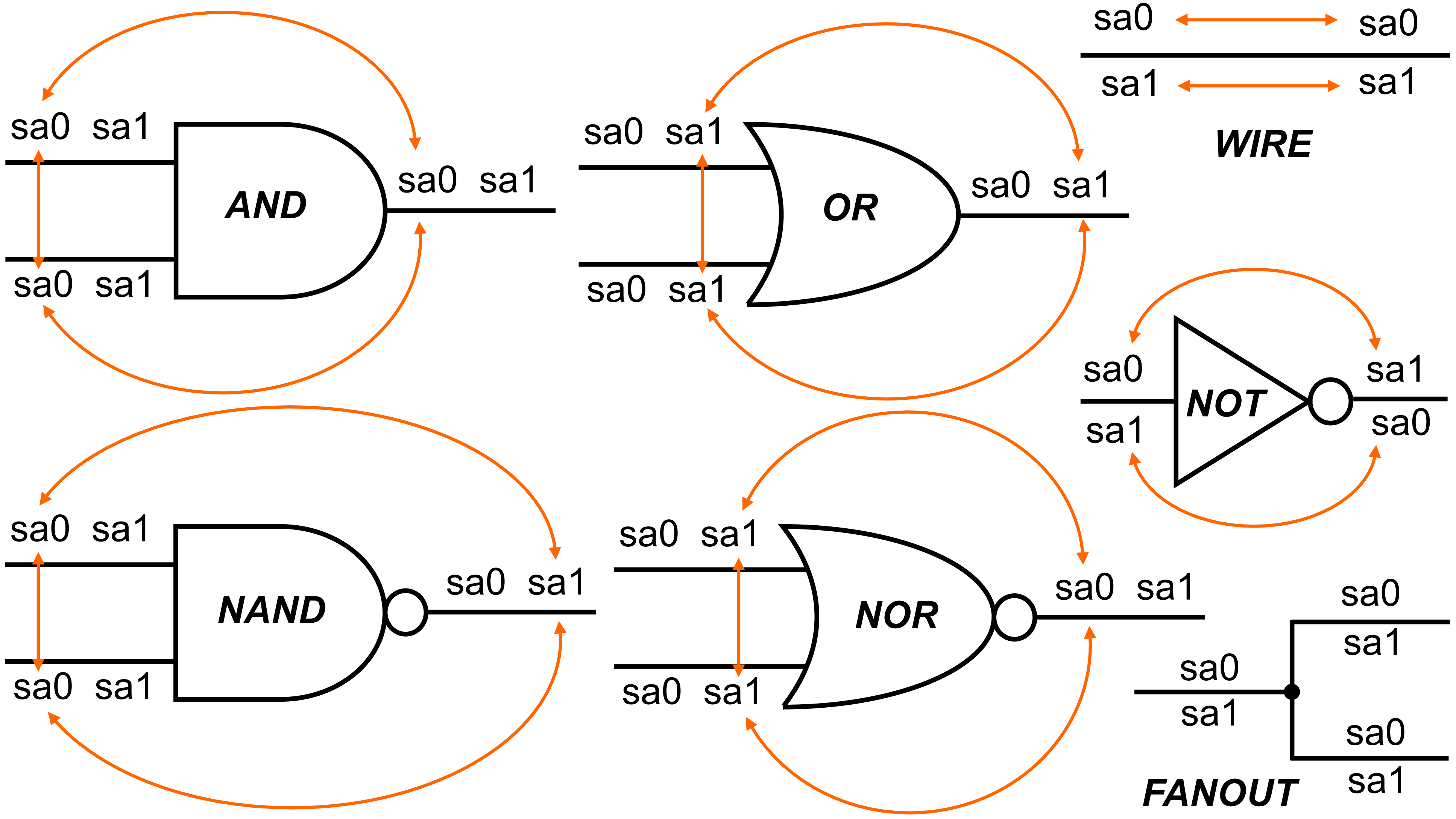


Fault Equivalence

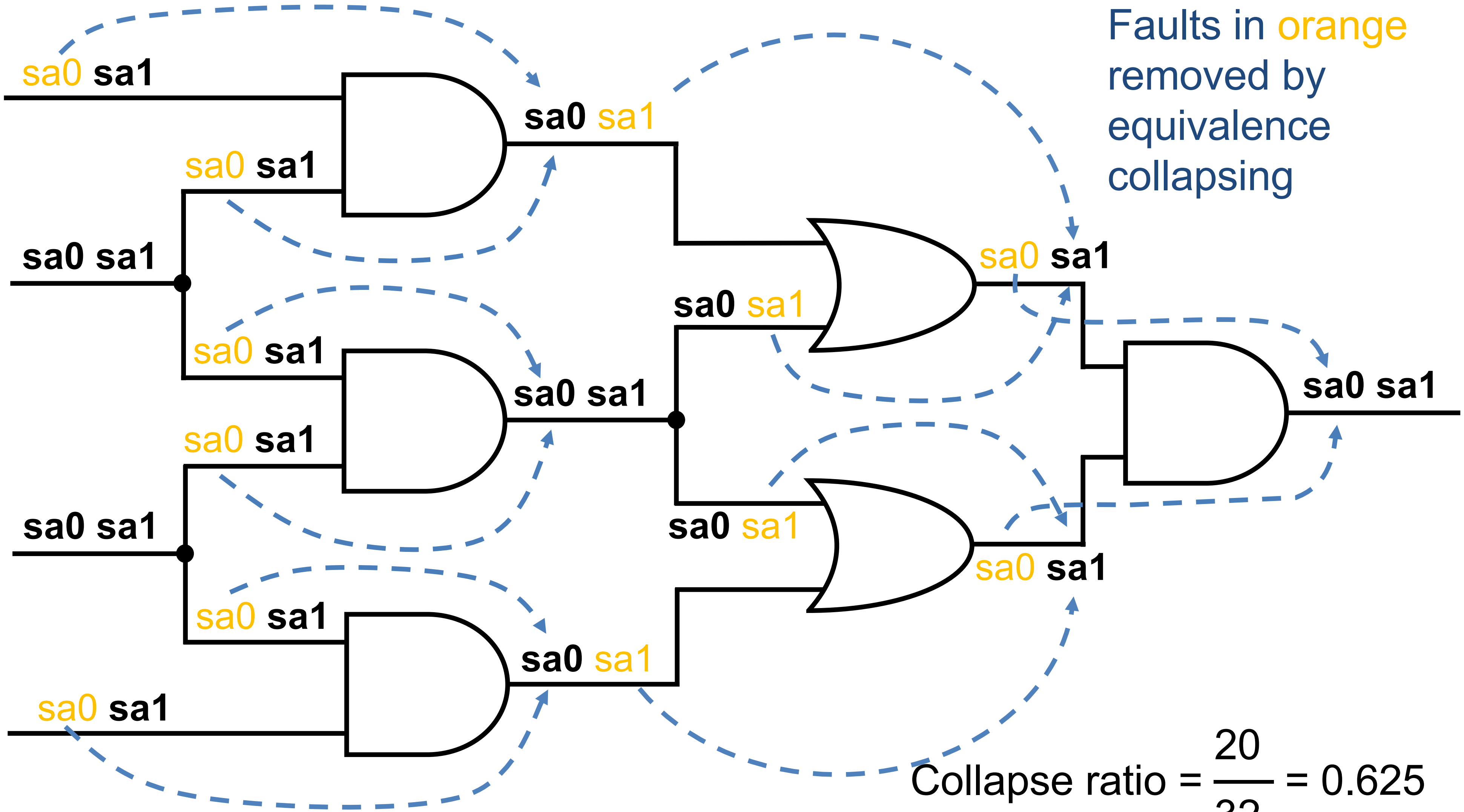
Fault Equivalence

- Number of fault sites in a Boolean gate circuit is
  = #Primary-Inputs + #gates + # (fanout branches)
- **Fault equivalence:** Two faults f1 and f2 are equivalent if the corresponding faulty functions are identical.
- If faults f1 and f2 are equivalent then any test that detects f1 also detects f2, and vice-versa.
- Fault equivalence checking rule: Faults f1 and f2 are equivalent iff the corresponding faulty functions are identical.
- **Fault collapsing:** All single faults of a logic circuit can be divided into disjoint equivalence subsets, where all faults in a subset are mutually equivalent. A collapsed fault set contains one fault from each equivalence subset.

Fault Equivalences Around Gates



Equivalence Collapsing Example



Faults in orange removed by equivalence collapsing

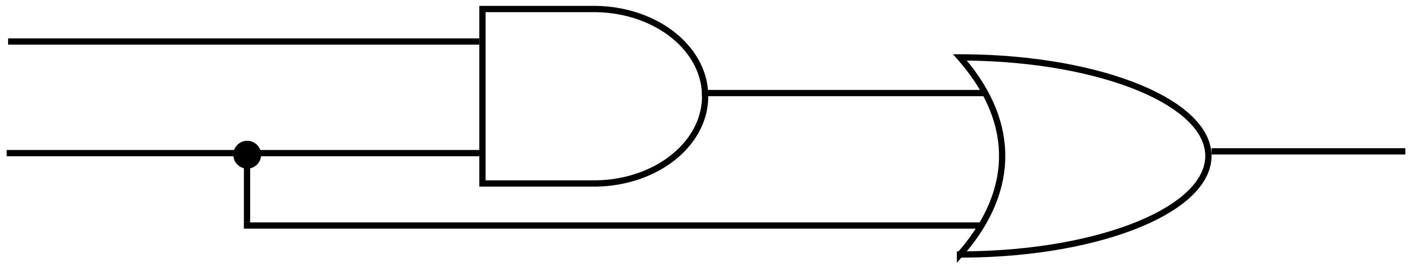
$$\text{Collapse ratio} = \frac{20}{32} = 0.625$$

Quiz 13-1

http://m.socrative.com/student/#joinRoom

room number: 713113

- Q1: What is the yield of a wafer that has 3 out of 10 chips defective:
  a) 30%
  b) 70%
  c) 130%
- The cost of a chip is $1.00 when its yield is 50%. What will be its cost if you increased the yield to 80%.
- What is the total number of single stuck-at faults, counting both stuck-at-0 and stuck-at-1, in the following circuit?
- Which faults are left after equivalence fault collapsing? What is the collapse ratio?



Answers to the Exercise

- The cost of a chip is US$1.00 when its yield is 50%. What will be its cost if you increased the yield to 80%.

Assume a wafer has *n* chips, then

$$\text{Chip cost} = \frac{\text{wafer cost}}{0.5 \times n} = \$1.00$$

$$\text{Wafer cost} = 0.5n \times \$1.00 = 50n \text{ cents}$$

For yield = 0.8, chip cost = wafer cost/(0.8*n*) = 50*n*/(0.8*n*) = 62.5 cents

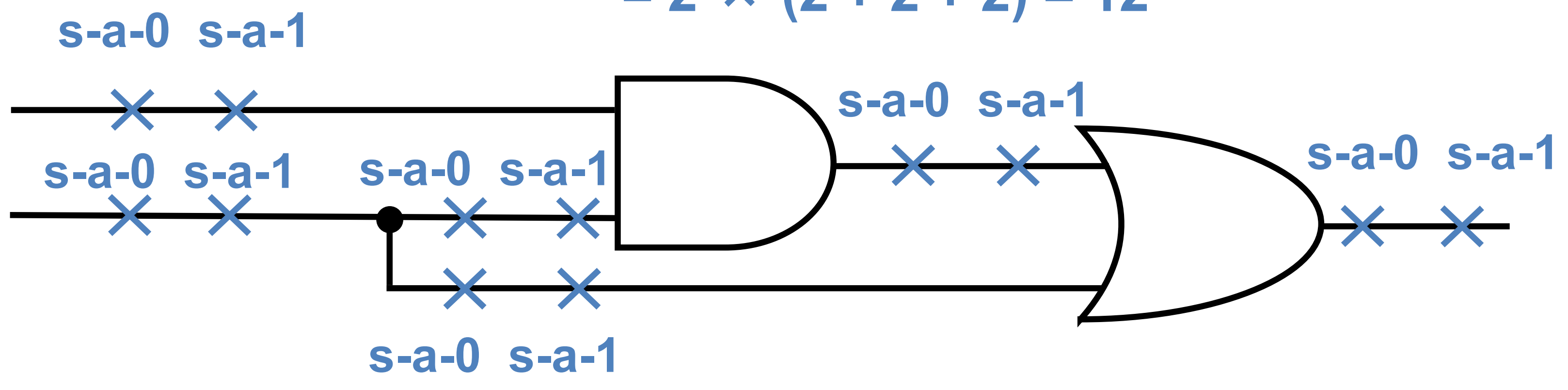
Answers Continued

- What is the total number of single stuck-at faults, counting both stuck-at-0 and stuck-at-1, in the following circuit?

Counting two faults on each line,

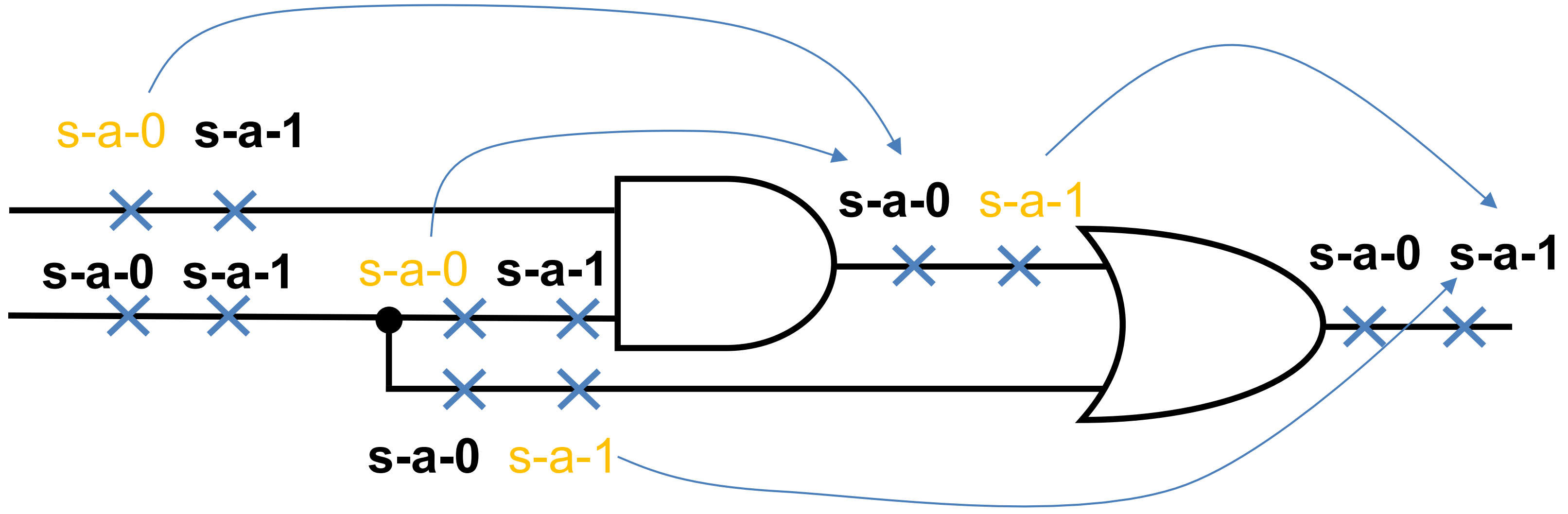
Total number of faults = 2 × (#PI + #gates + #fanout branches)

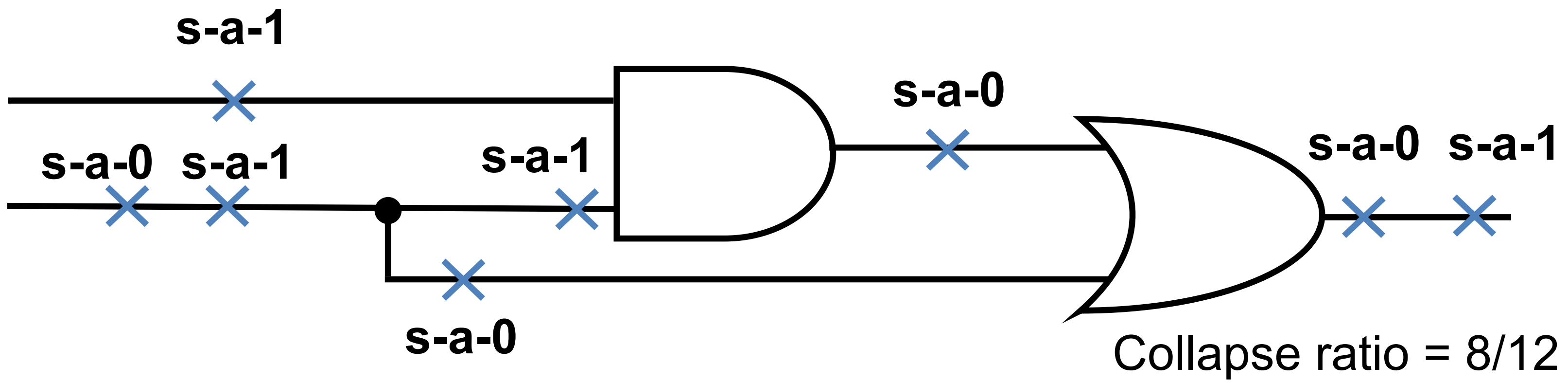
= 2 × (2 + 2 + 2) = 12



Answers Continued

- How many faults are left after equivalence fault collapsing?



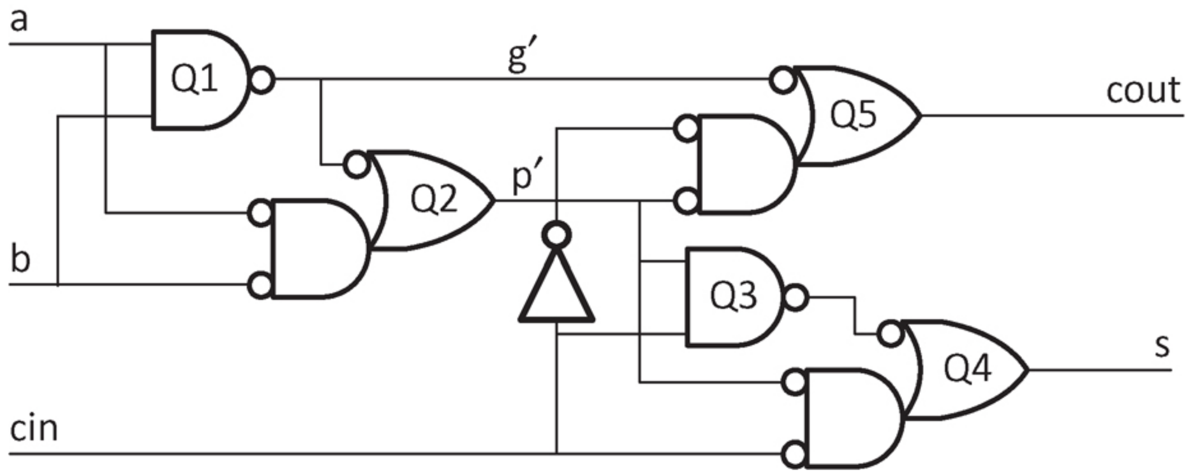


Collapse ratio = 8/12 = 0.67

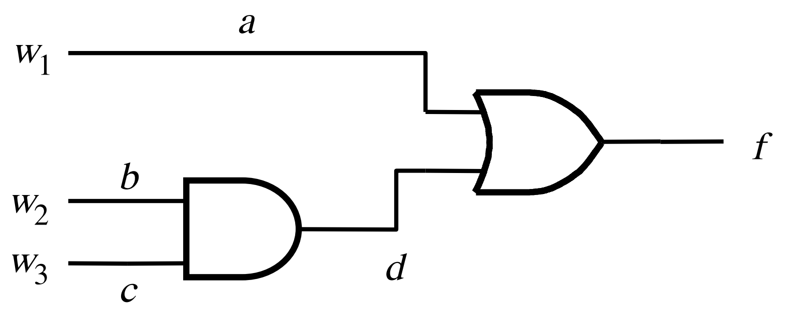
Testing of combinational circuits

Combinational testing

- How can we detect if a node in the design has stuck to 1 or 0 ?
- To test a block of combinational logic, we need a set of test patterns (test vectors).
- To detect a particular fault, we need to drive the node in question to the opposite state while the output is sensitive to that node



Testset



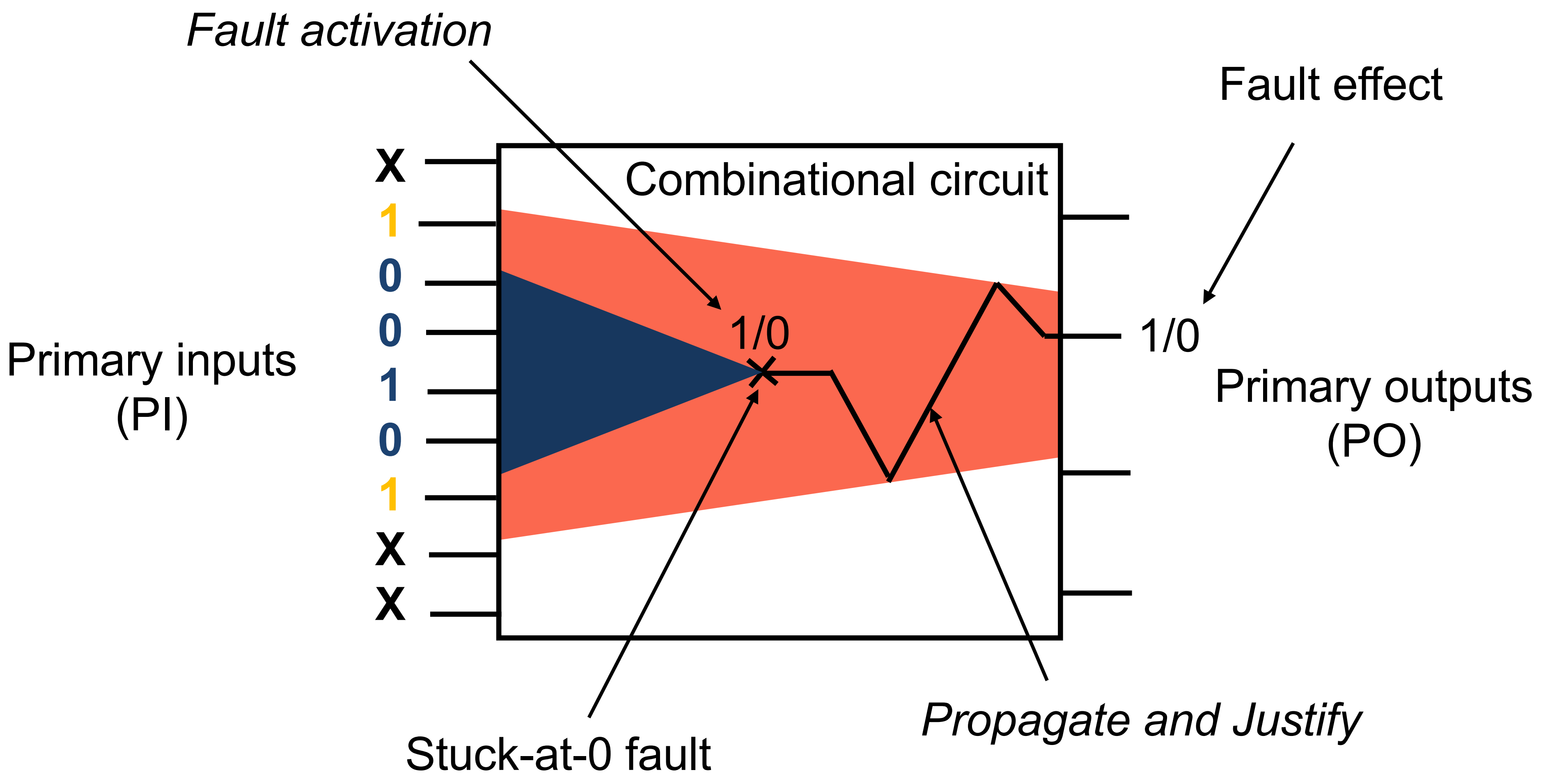
One possible approach but unrealistic for big circuits!

Test	Fault detected									
$w_1 w_2 w_3$	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		√						√		√
001		√		√				√		√
010		√				√		√		√
011			√		√		√		√	
100	√								√	
101	√								√	
110	√								√	
111									√	

Automatic Test Pattern Generation

- ATPG: Automatic test pattern generation
  - *Given*
    - A circuit (usually at gate-level)
    - A fault model (usually stuck-at type)
  - *Find*
    - A set of input vectors to detect all modeled faults.
- Core solution: Find a test vector for each given fault.
- Combine the “core solution” with a fault simulator into an ATPG system.

Path sensitization



Algorithm for “path sensitization”

1. ***Activate****: Specify inputs so as to generate the* appropriate (opposite than the fault) value (0 for SA-1, 1 for SA-0) at the site of the fault.

2. ***Propagate****: Select a path from the site of the* fault to an output and specify additional signal values to propagate the fault signal along this path to the output (error propagation).

3. ***Justify****: Specify input values so as to produce* the signal values specified in (2) (line justification).

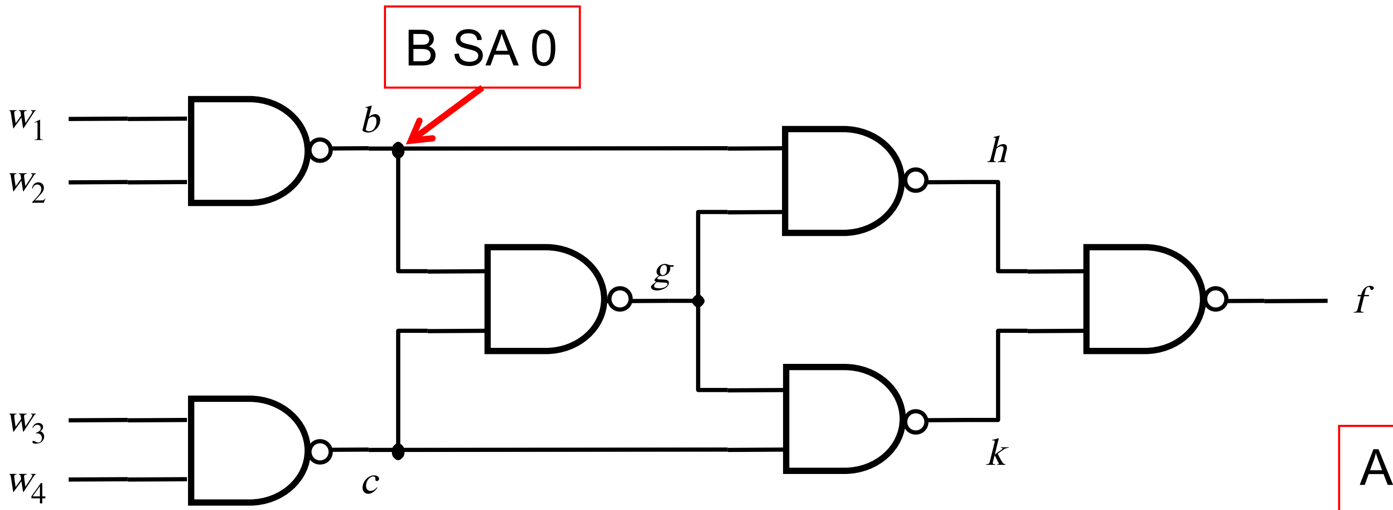
Algorithm for “path sensitization”

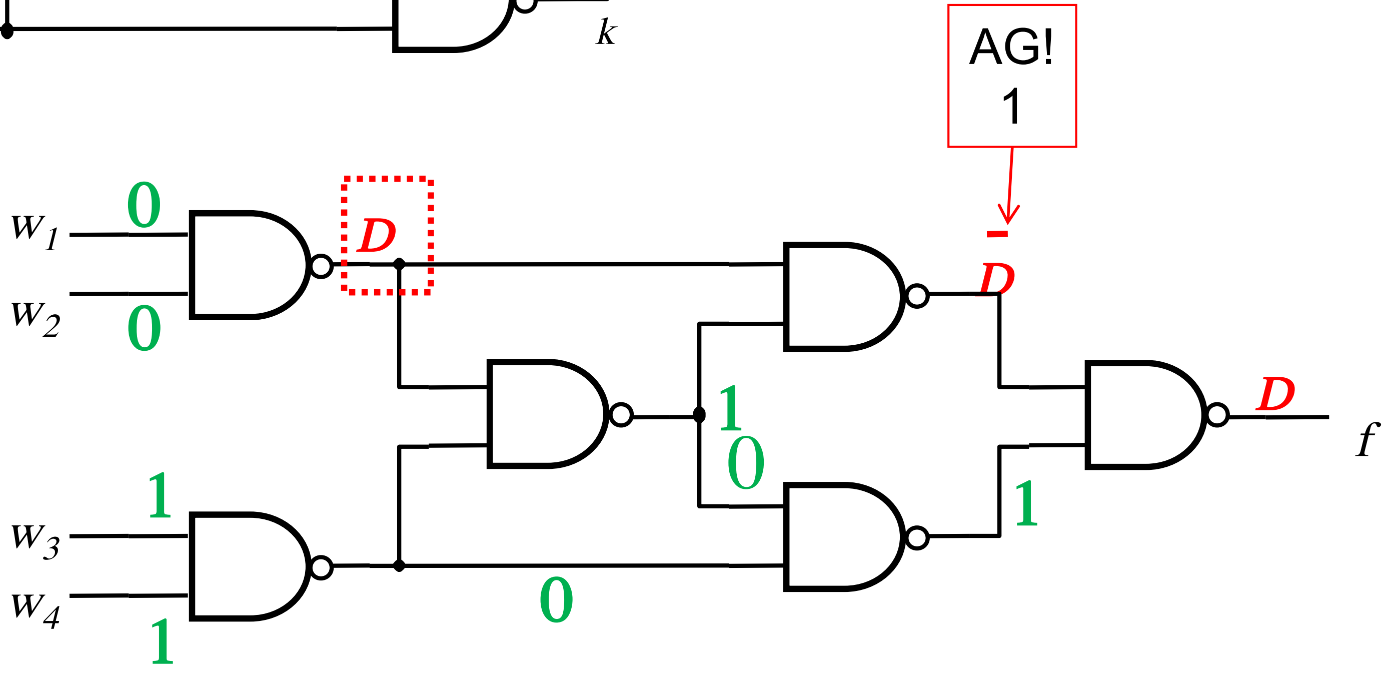
- The process is finished when all the assigned values have been solved by "justify" and there are no conflicts

- A conflict has occurred on two different Justifications require different values assigned to a node.
  - An input value required to activate the fault is different than the input values required to propagate the fault.

- If a conflict arises, we look for another way for justification.

D•D’ =0, D+D’=1, D’ •1 = D’



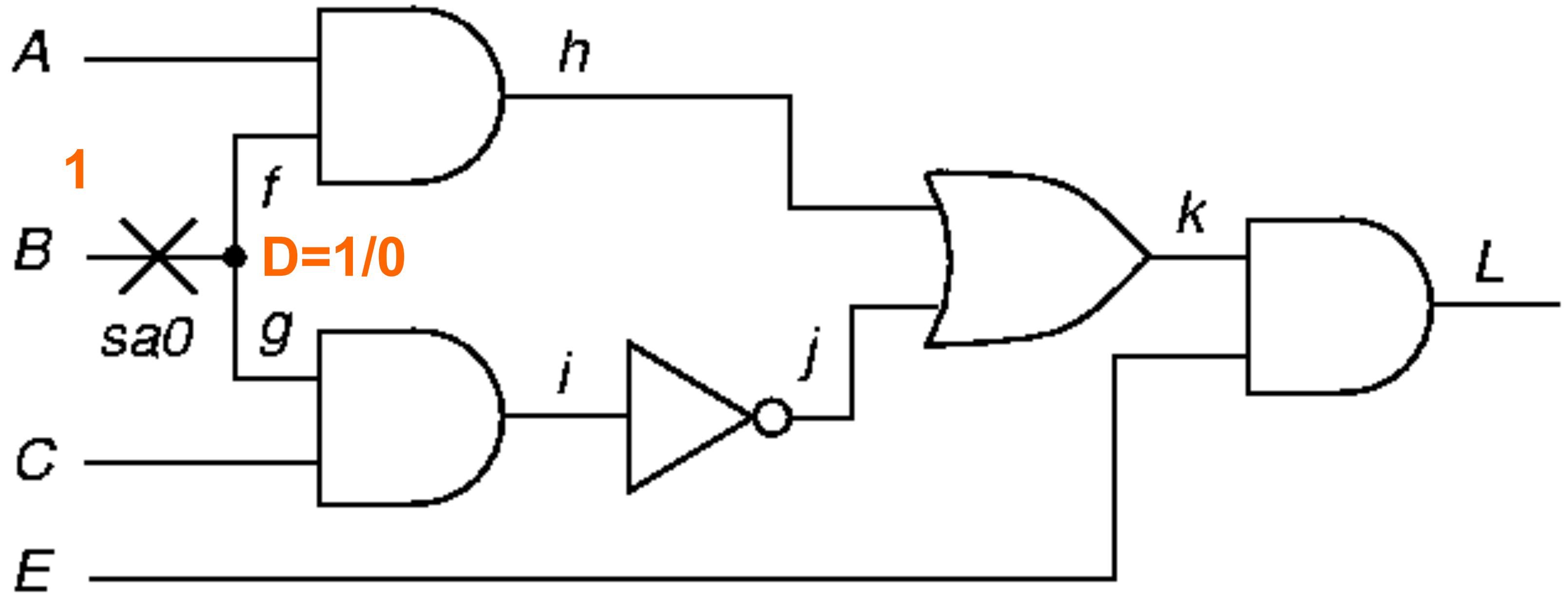


An ATPG Example

1 Fault activation

2 Propagation

3 Line justification

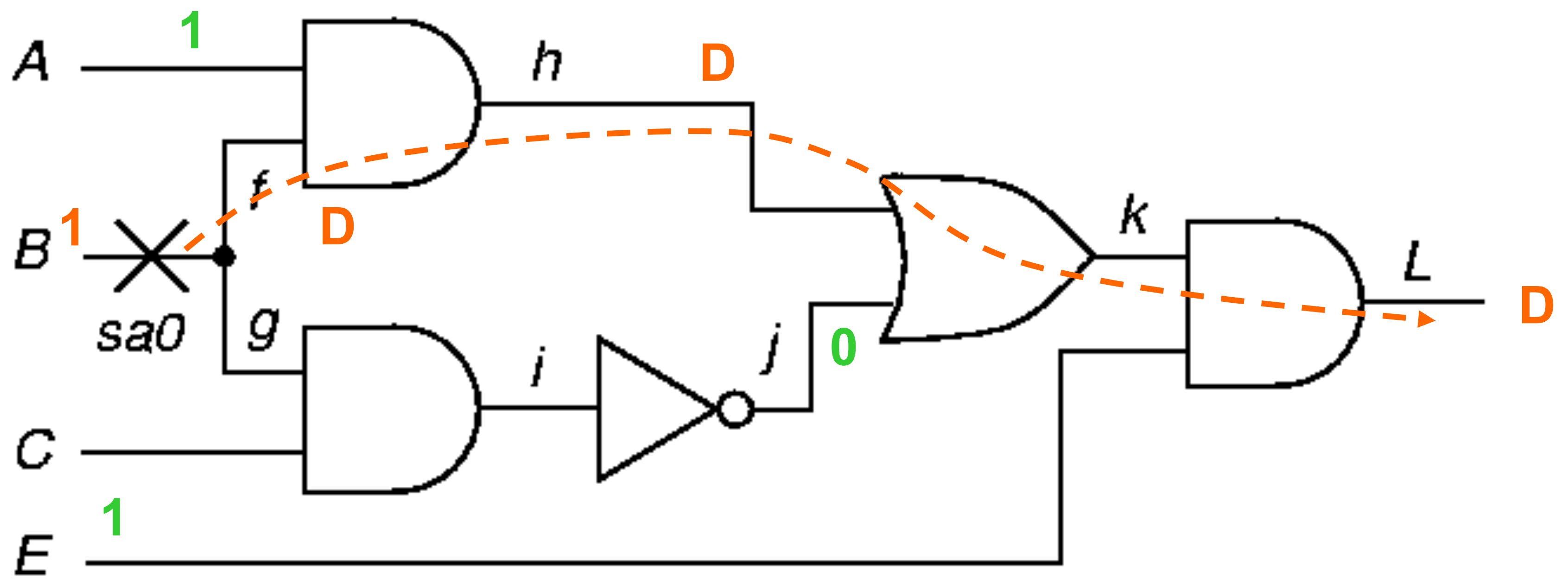


ATPG Example (Cont.)

1 Fault activation

2 Propagation

3 Line justification

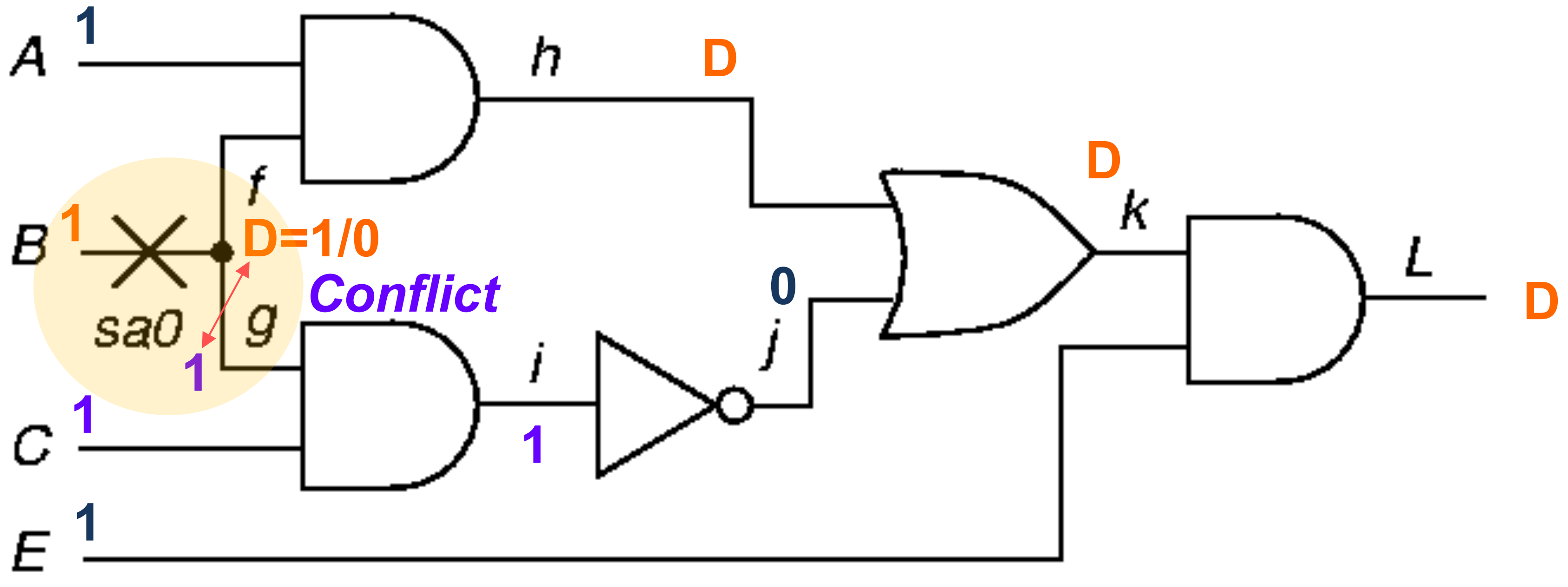


ATPG Example (Cont.)

1 Fault activation

2 Propagation

3 Line justification



ATPG Example (Cont.)

1 Fault activation

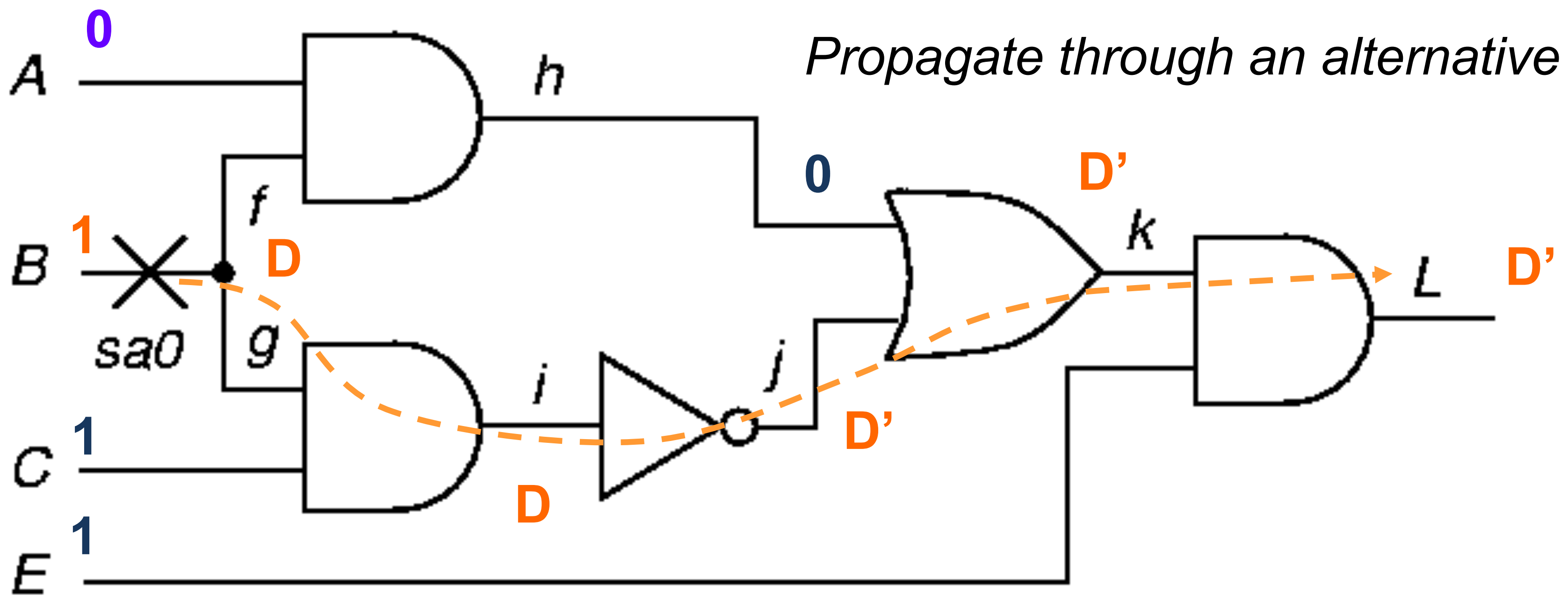
2 Propagation

3 Line justification

Backtrack

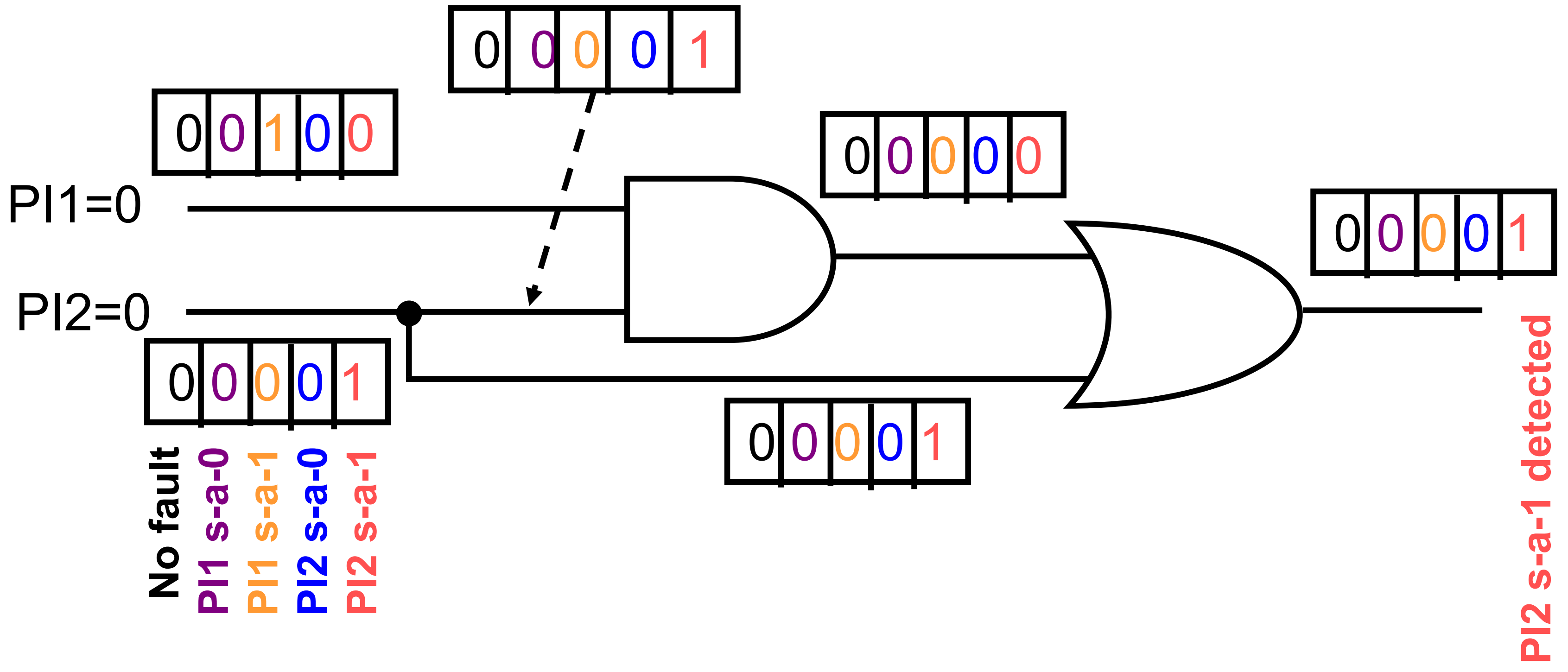
Test found

Propagate through an alternative path



Fault Simulation: Find Faults Detected by a Vector

- Fault simulation of four PI faults is illustrated below.
  Fault PI2 s-a-1 is detected by the 00 test input.

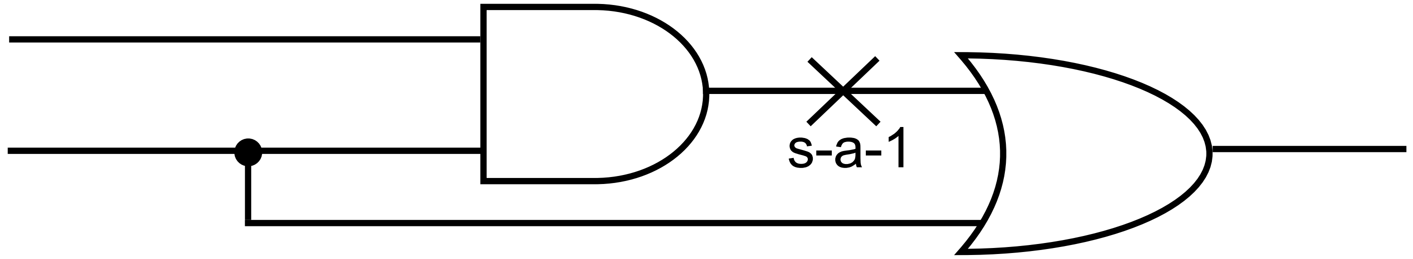


Quiz 14-1

http://m.socrative.com/student/#joinRoom

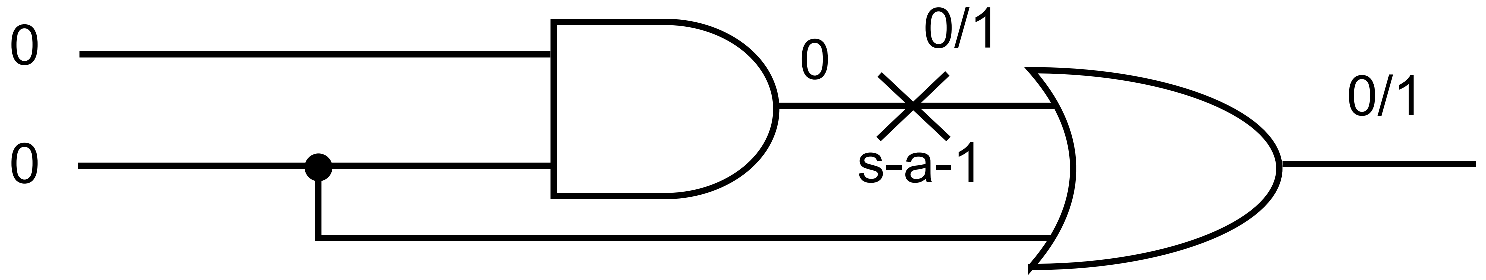
room number: 713113

- Q1: find a test vector to detect the following stuck at fault in the circuit:

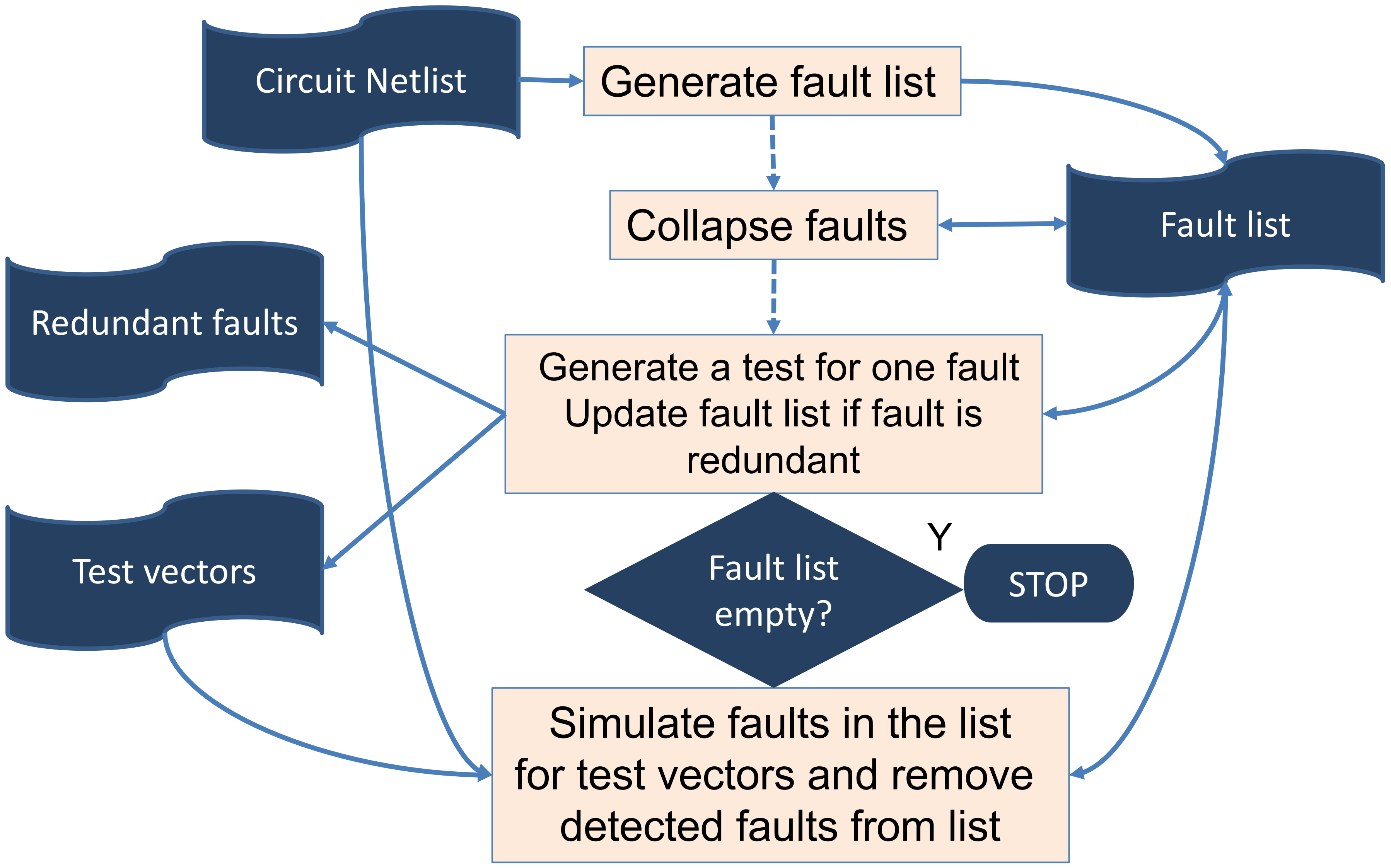


Q1: Find a Test for a Fault

- A test for the stuck-at-1 fault shown in the diagram is 00.



A Test Generation System

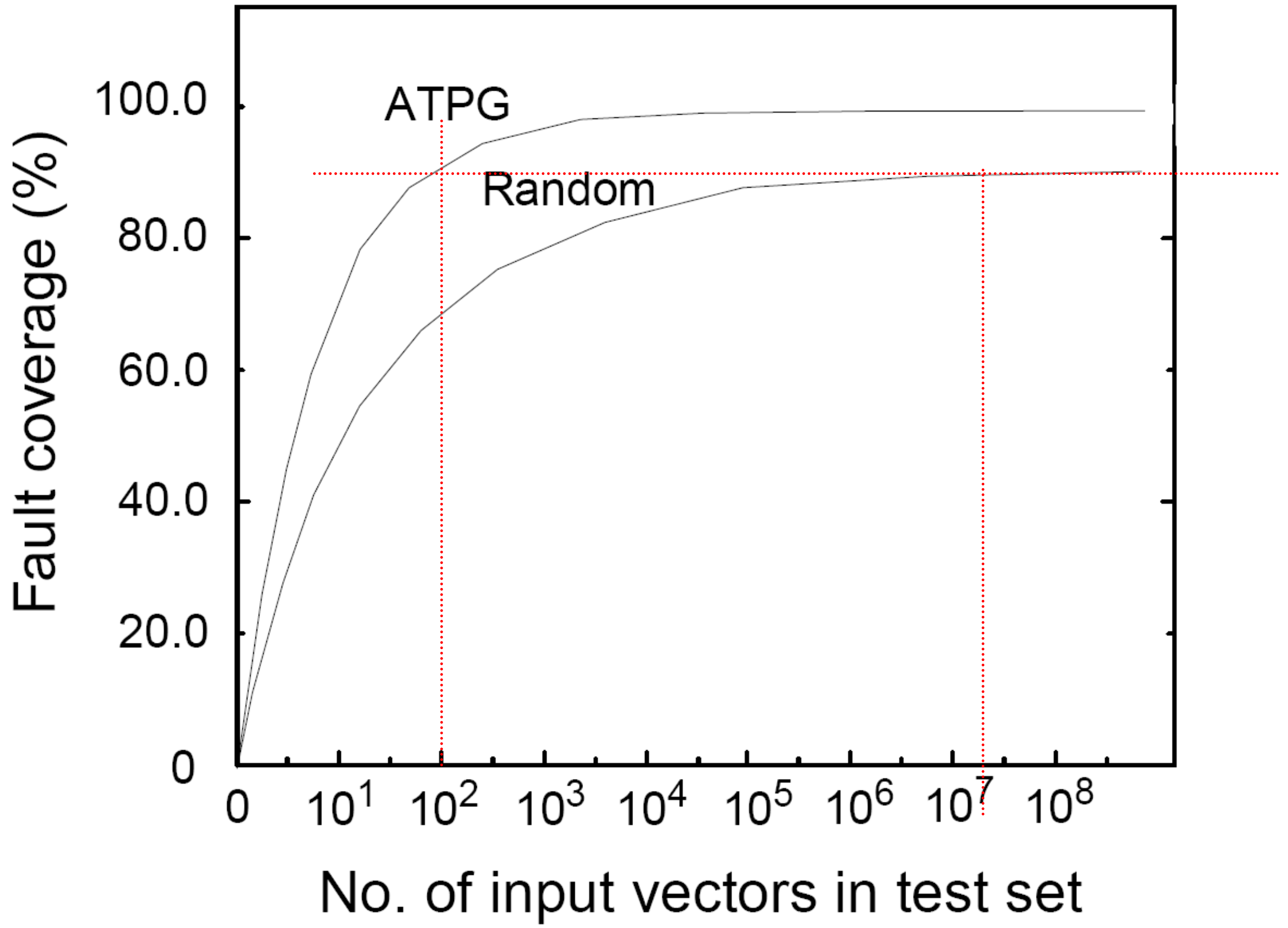


Circuit Netlist
Generate fault list
Collapse faults
Fault list
Redundant faults
Generate a test for one fault
Update fault list if fault is redundant
Test vectors
Fault list empty?
Y
STOP
Simulate faults in the list for test vectors and remove detected faults from list

Evaluating Test Fault Coverage

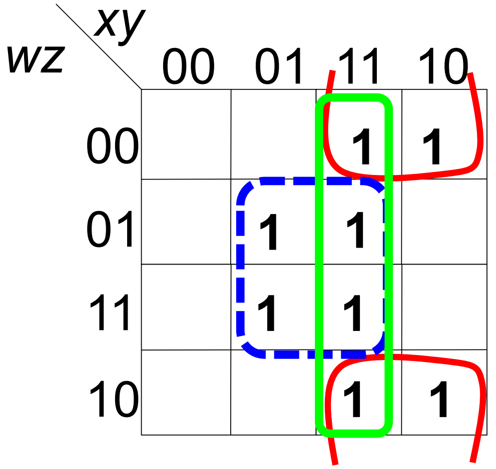
- The percentage of faults in an total set of faults that is detectable by our tests.
- Fault coverage is used both as:
  - as a measure of efficiency for a given amount of test vectors, and
  - as quality measures for test-generation algorithms.

Fault coverage for ATPG and random vectors



Why we cannot have 100% coverage?

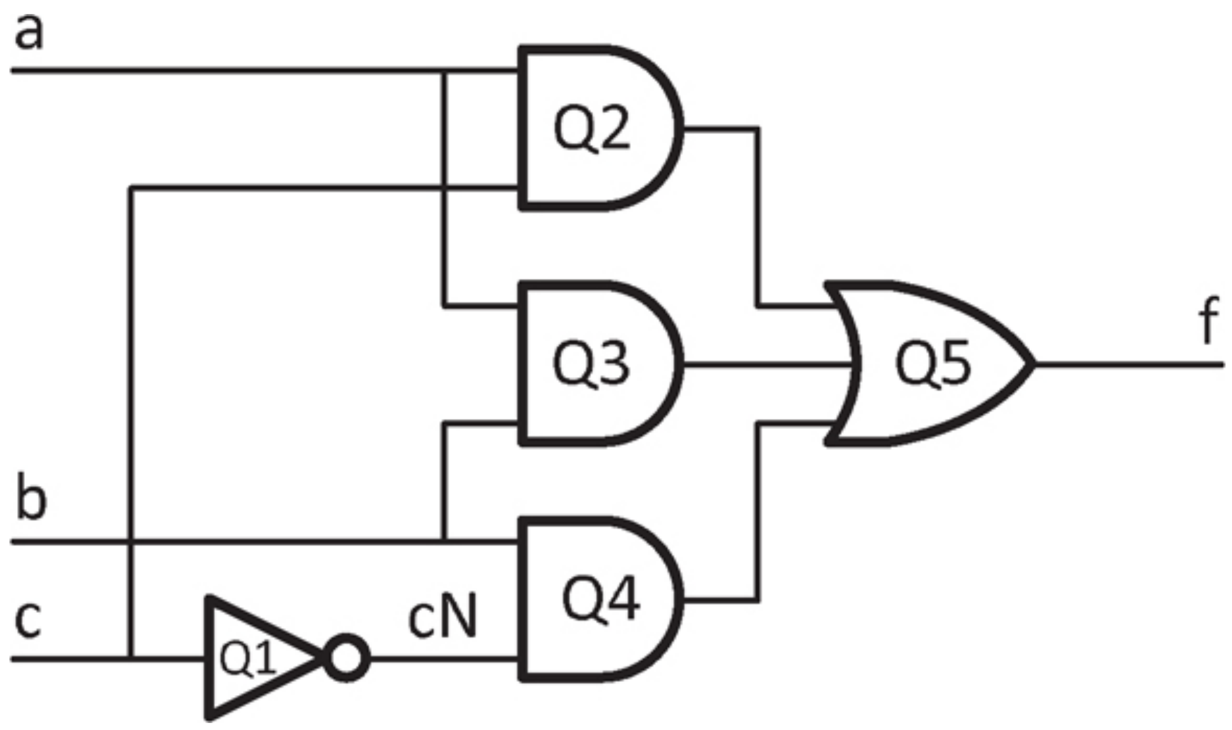
- ATPG Algorithms can sometimes be inefficient
- Redundancy in circuits creates untestable SA-fault.
  - Accidental redundancy can be obtained by insufficient minimization of functions
  - Intentional redundancy is given by the hazard elimination or fault-tolerant circuits
- We define a combinatorial circuit as redundant when it contains untestable SA-faults.



$$U = x\bar{z} + xy + yz$$

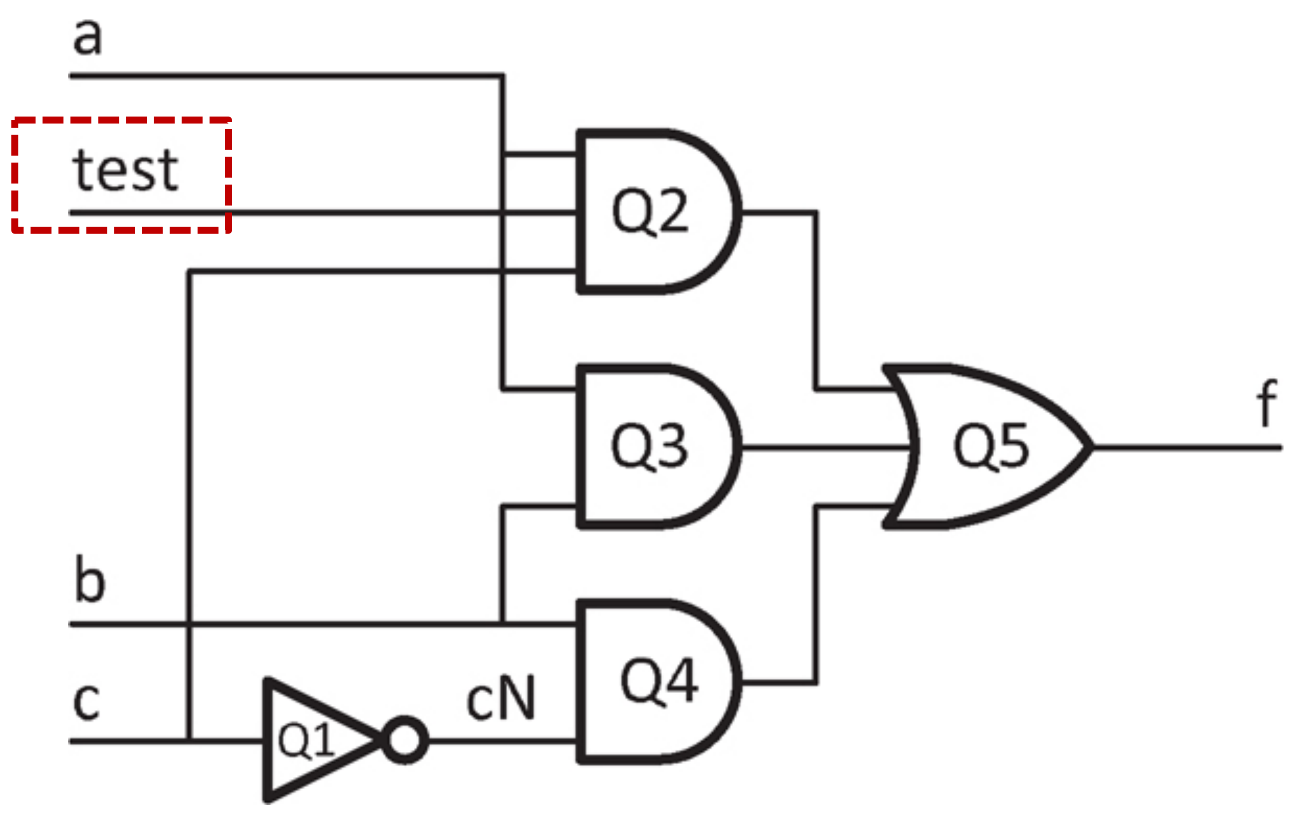
Testing redundant logic 1/2

- Is it possible to test if Q3 is working with stuck-at fault model ?



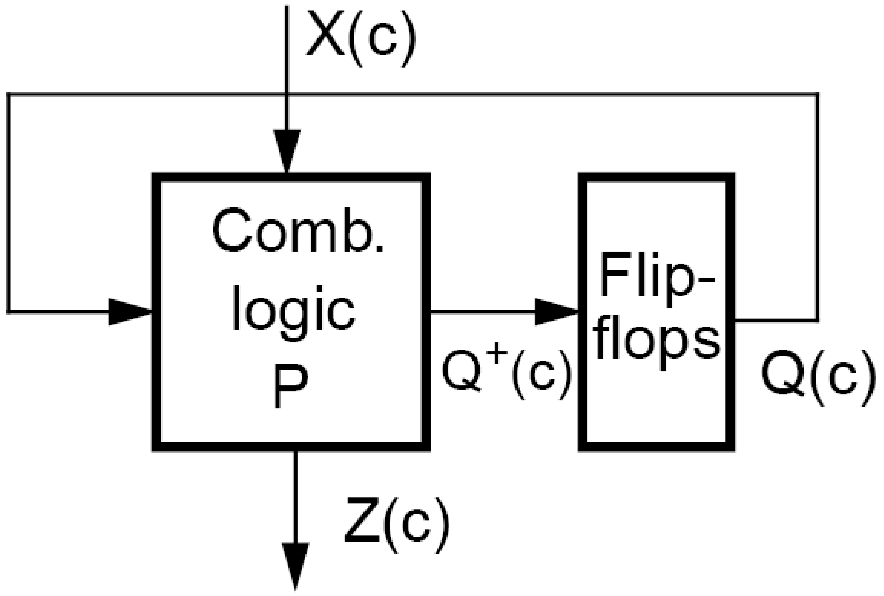
Testing redundant logic 2/2

- To test this circuit we have to introduce an auxiliary input to one of the other two gates.



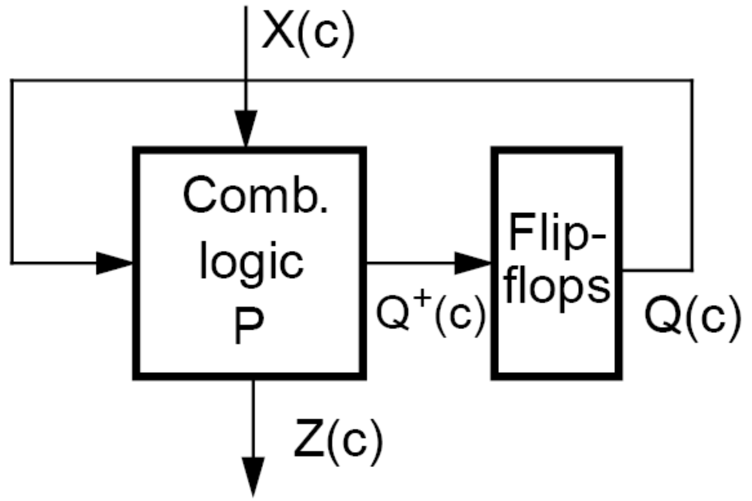
Testing of sequential circuits

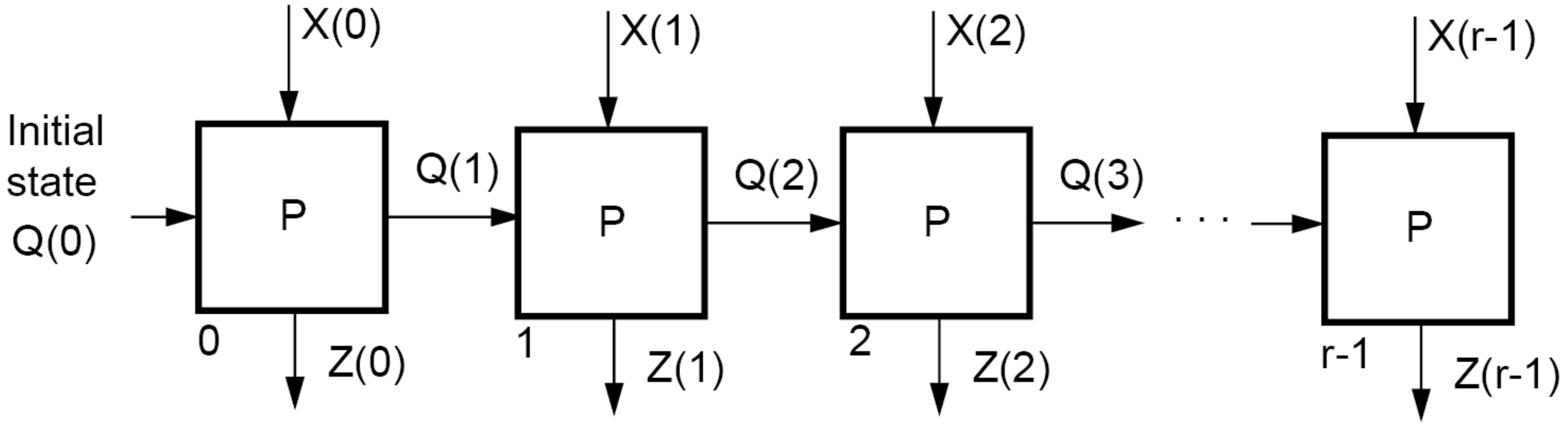
- More difficult than combinational circuits
  - Outputs not defined only by inputs
    - There exist internal memory elements (which maintain some state)
- Aspects to be considered
  - Memory elements to be set in a
    Known state
  - Both input vectors and state
    need to be considered in the
    Test vector
  - More than one input vector may
    be needed to activate a fault



Generating tests for sequencial circuits

- Unroll the sequential circuit in time
- Then: Initialize to a known state Q(0)
  - Use the method of combinatorial circuits to generate vectors for cell 0
  - The input vector from the test generation can be used for r cycles
  - If the fault is not detected after r cycles, increase r by 1





Design for Testing (DFT)

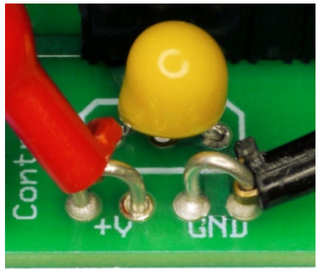
Design for testability

- A major problem when testing is often the ability to **control** and **observe** the values of signals. This makes difficult to **activate** and **propagate** faults.
  - To improve testability we use various design-for-test (DFT) methods.
  - The disadvantage of DFT is the increased complexity, more silicon is required, and some performance degradation.

Design for testability

- Two possible approaches
  - Ad-hoc methods
    - can be applied **after** a circuit is designed
  - Structured design methods
    - Testability is involved from the beginning

Ad-hoc methods for testability

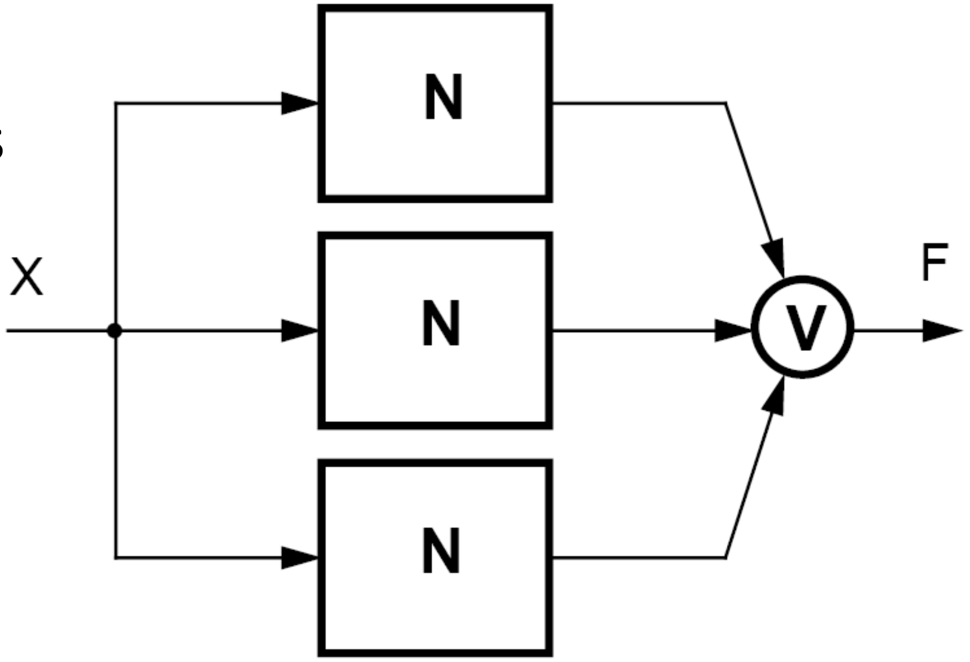


- Well-known techniques based on experience to increase ability to control and observe
  - **Add test points.**
    - By adding additional pins to/from the circuit to increase ability to control and observe
    - **Initialization – reset**
      - Get the circuit to a known state

  - **Control the clock**
    - Stop it, single step, different frequency than the normal
  - **Partition large circuits/systems**
    - Divide and conquer!
    - Much easier to generate the test, and test small circuits than large ones.
    - Careful partitioning often makes it possible to test parts of the circuit in parallel, which improves speed.

Ad-hoc methods

- Redundant logic
  - AVOID!
    - No TMR (Triple Modular Redundancy)
- Global feedbacks
  - Breaking the feedback with logic
- Asynchronous sequential circuits
  - **AVOID**
- Divide and conquer!
  - Datapath – control logic
  - Analog – digital



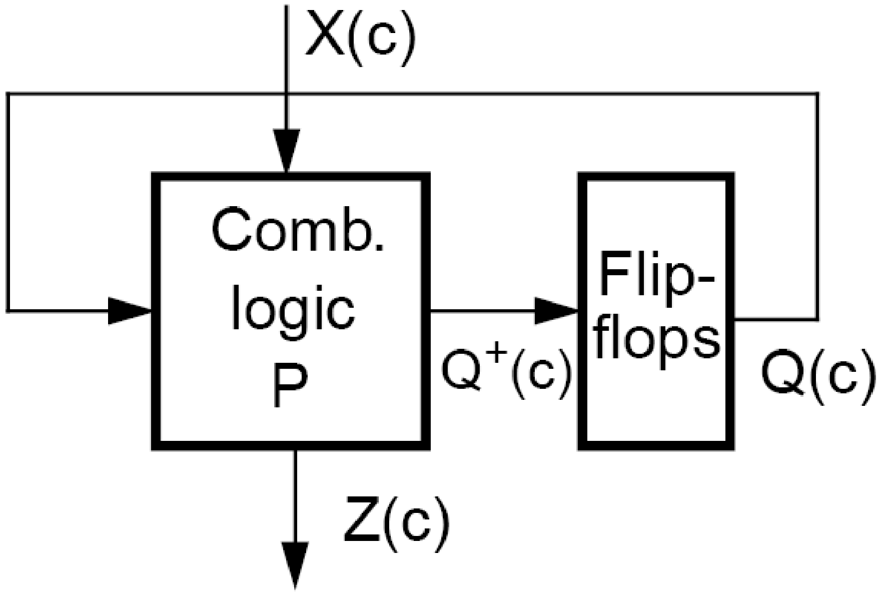
(a) TMR design

Structred design for test

- Split sequential circuits to:
  - Combinatorial part
  - Registers (memory elements)

- Using algorithms for combinatorial circuits in the combinatorial part

- Manage registers separately



Scan-chains

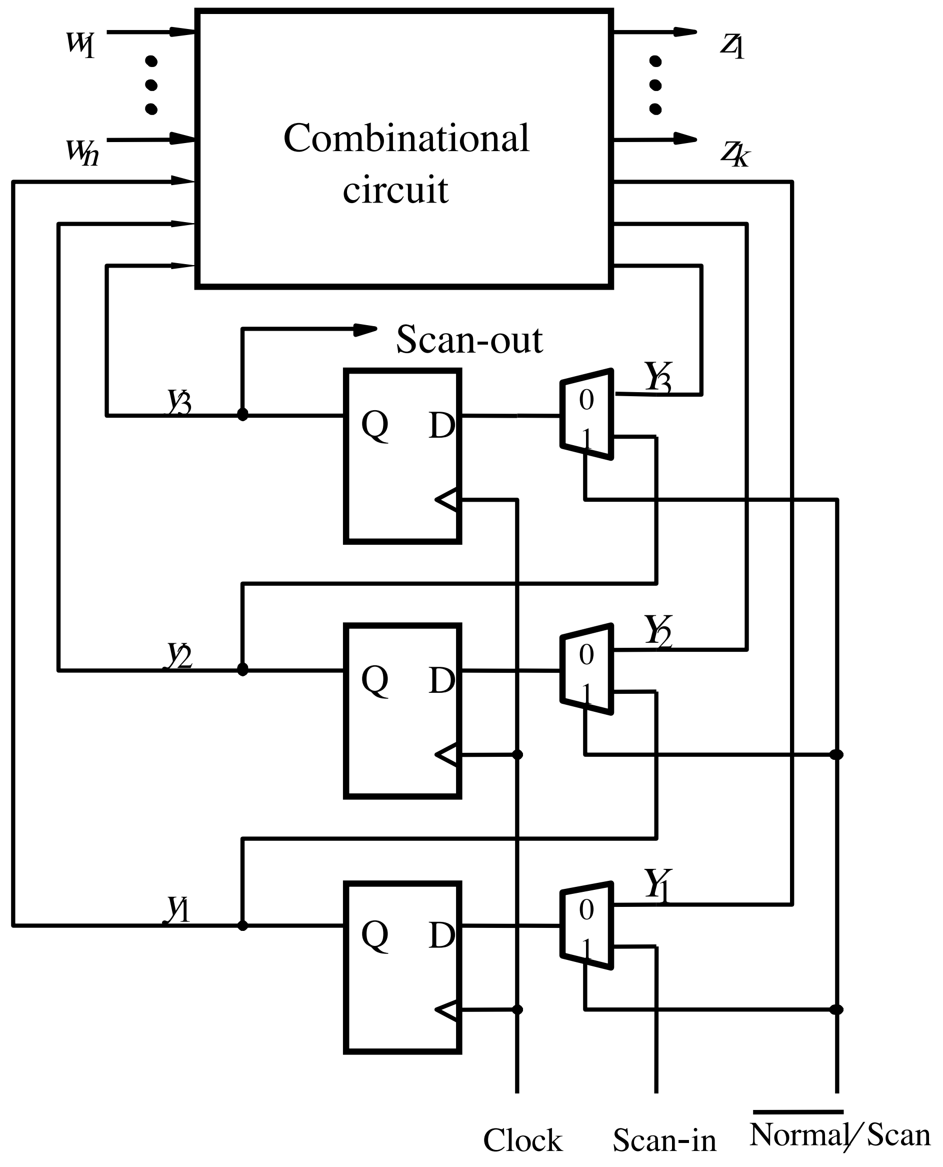
- Scan-chains are the most common DFT method.
- All the memory elements in the circuit are chained together to form a shift register so that you can easily shift in/out data in the circuit.
- Costs about 30% more compared to standard circuits

Pros:

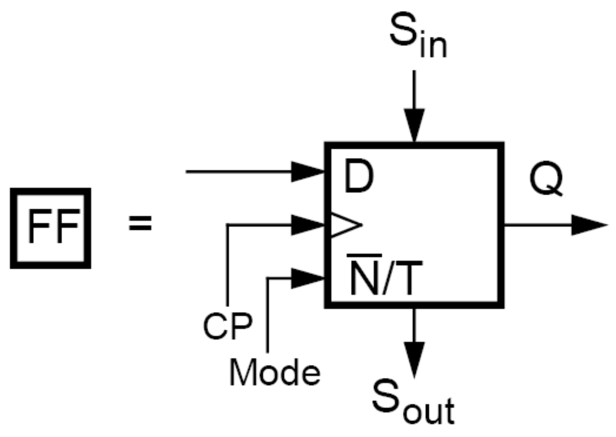
- Easier test vector generation
- Requires only little extra chip area
- Construction materials are cheap

Cons:

- Signal delay increases
- More memory required in the test equipment



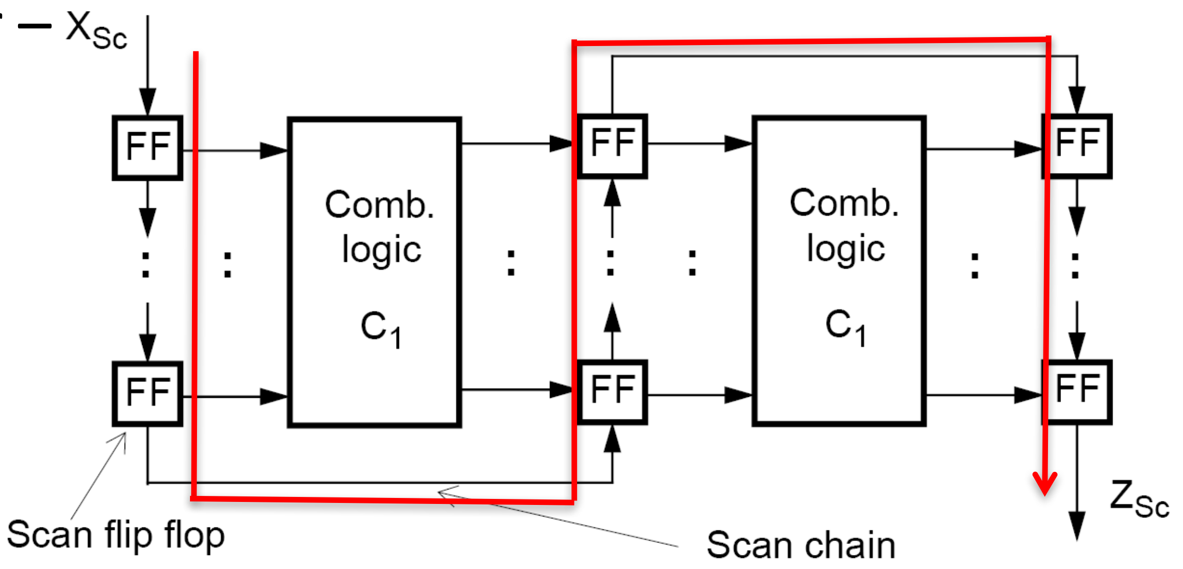
Scan-chains



Mode=0: Normal mode
Input: D, Output: Q

Mode=1: Test mode
Input: S_{in}
Output: S_{out}

- While a vector is shifted in the circuit, we read out the result.
  1. Test mode
  2. shift in the test vector – check the output
  3. Normal mode
  4. Clock-pulse
  5. Start with the next vector (1)



Build-in Self Test (BIST)

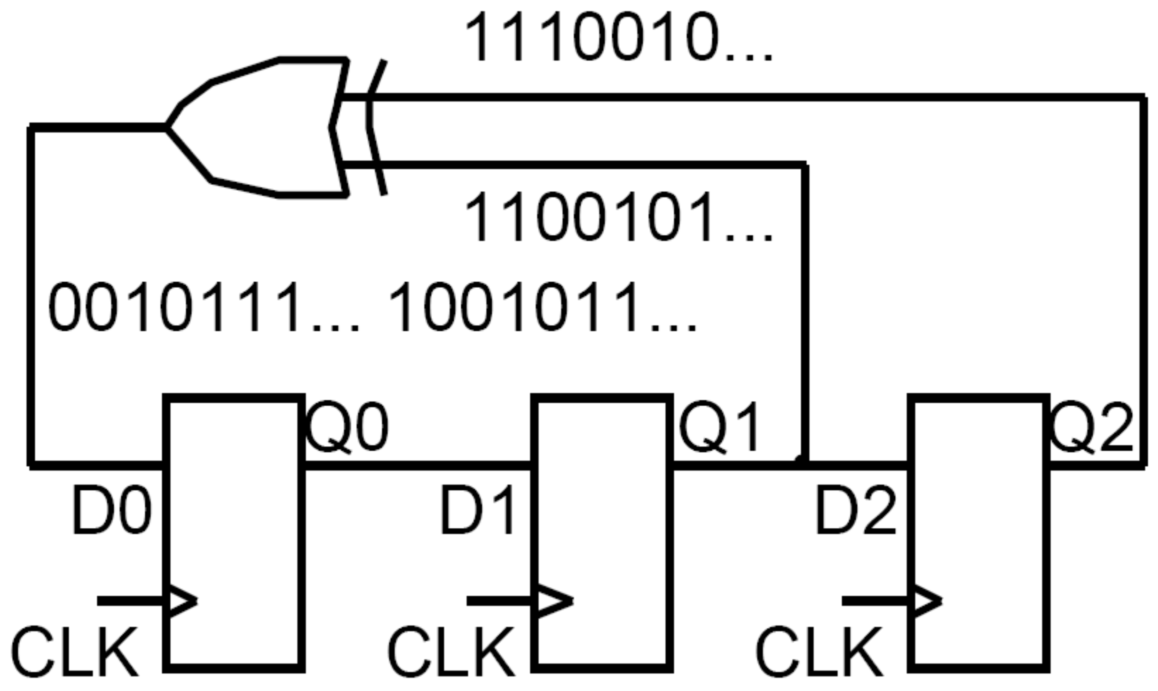
- Is a technique to allow the circuit to test itself
  - In production - no test equipment needed
  - In box - easy to test the function
- The testing is done when the system is not operating
  - Cannot be performed during normal operation

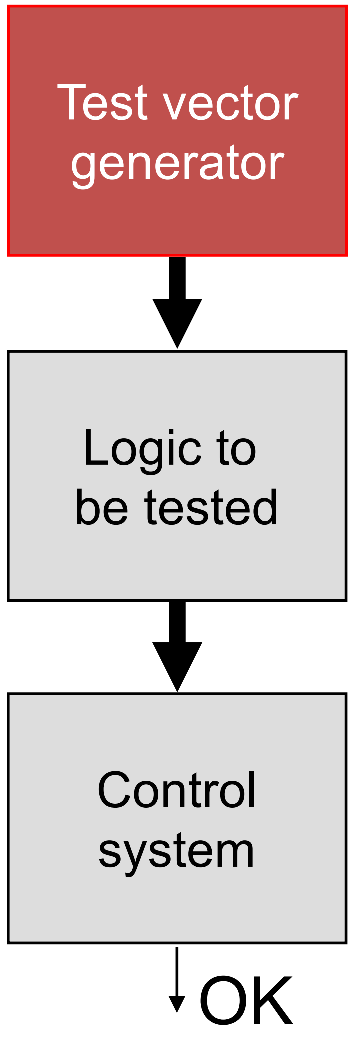
- Pros
  - Simpler system test
  - Simpler test planning
  - Testing at normal frequency
  - Circuit Libraries
  - Can easily apply many test vectors

- Cons
  - Not supported by CAD/EDA-tools
  - Overhead in design
  - Additional components are needed
    - More silicon area
    - More things can go wrong

Build-in Self Test (BIST)

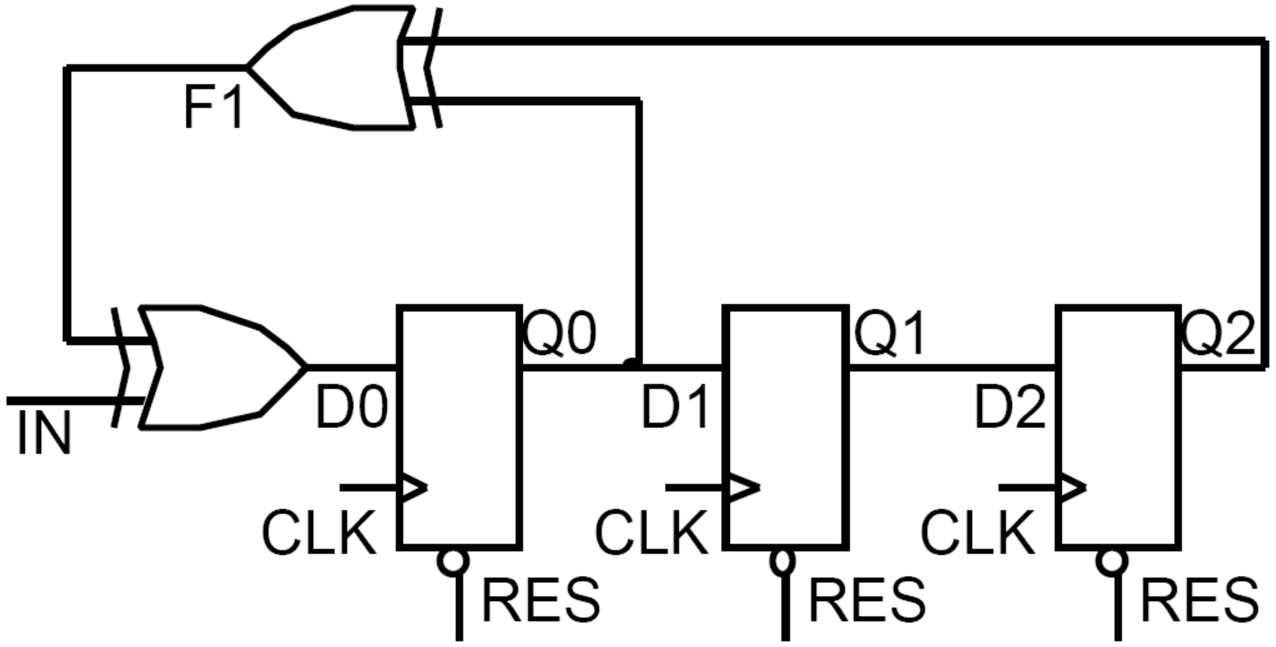
- Principle
- How?
  - ”Random number generator ”
  - ROM
  - Counters, LFSR (Linear Feedback Shift Register) etc.

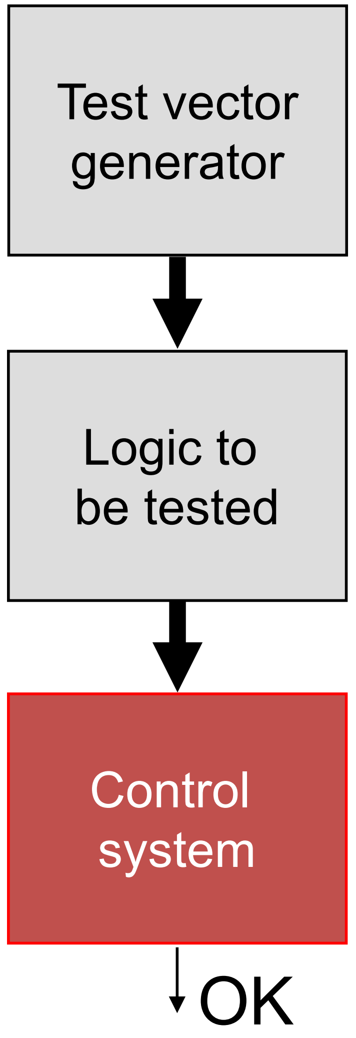




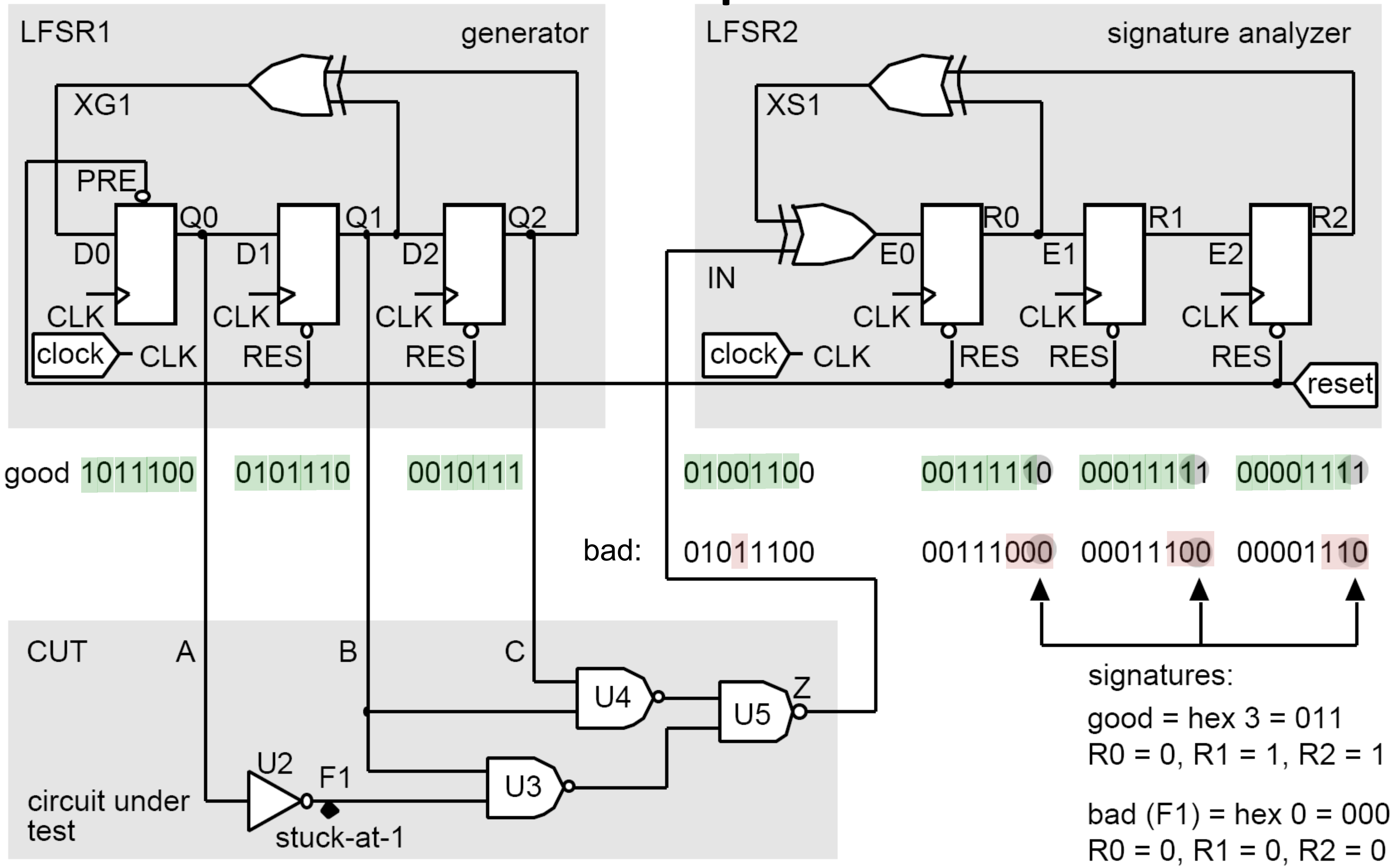
Build-in Self Test (BIST)

- Compression
  - Get a signature





Example



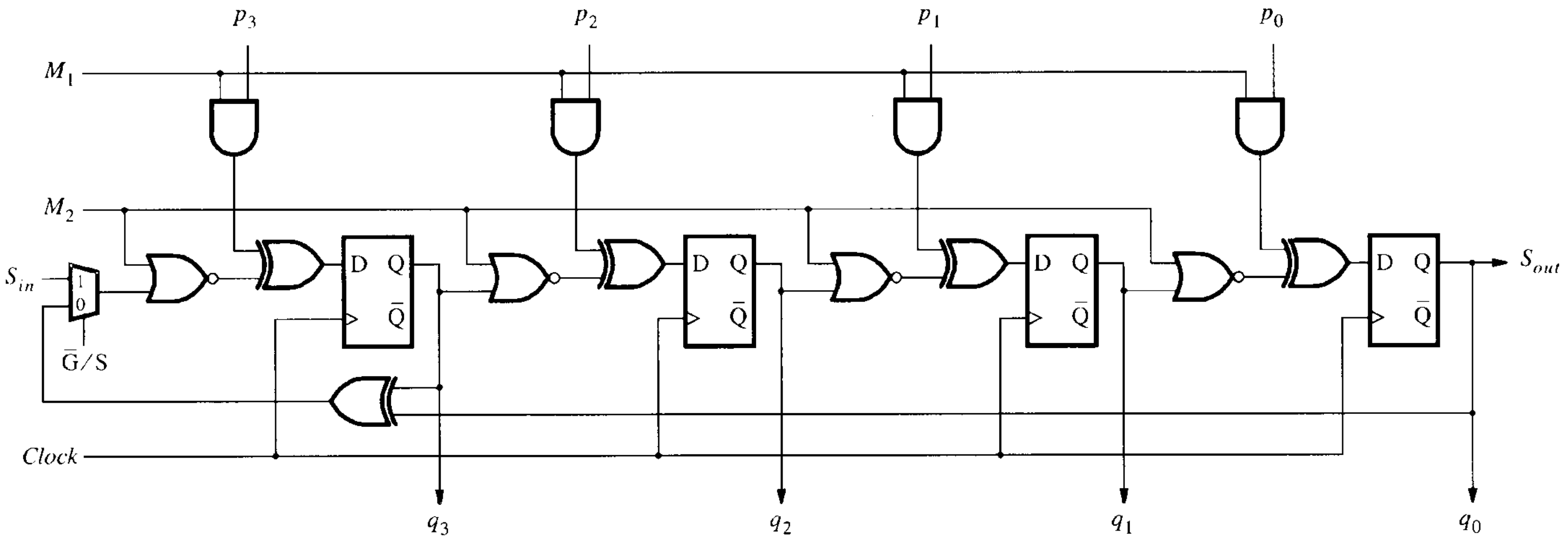
signatures:

good = hex 3 = 011
R0 = 0, R1 = 1, R2 = 1

bad (F1) = hex 0 = 000
R0 = 0, R1 = 0, R2 = 0

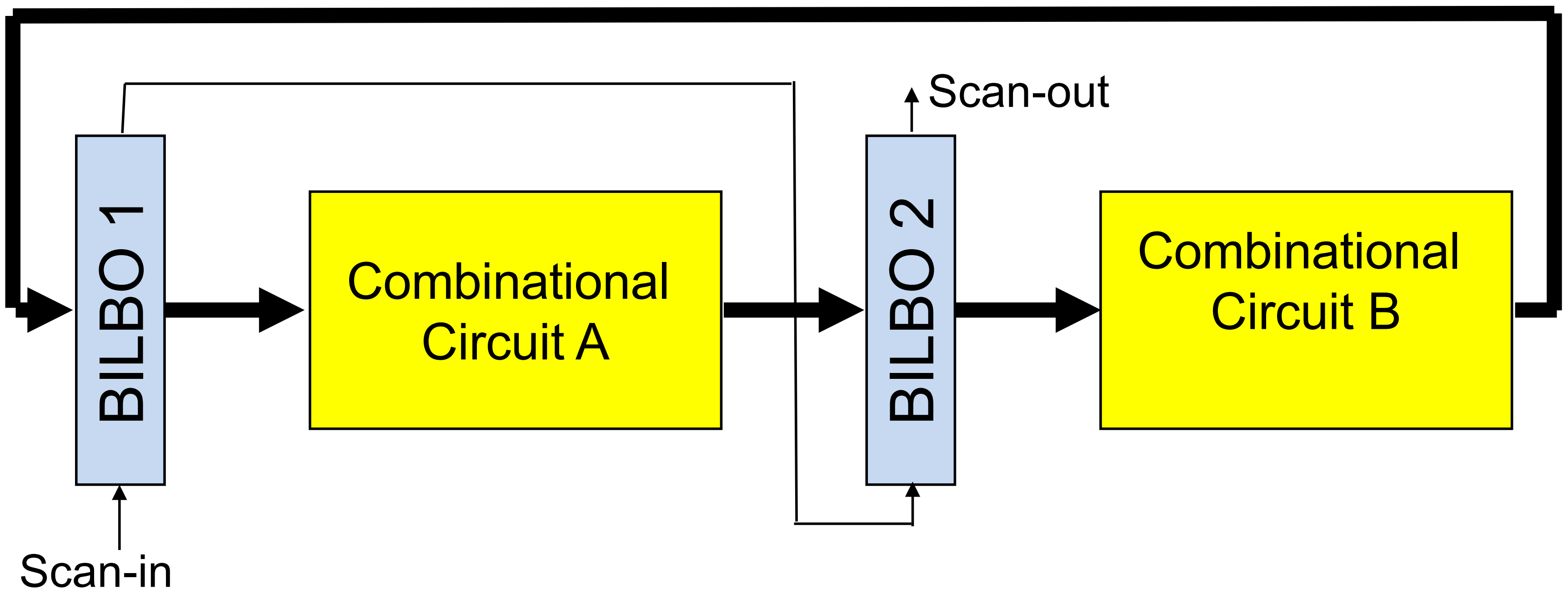
BILBO (***B**uilt-**i**n **l**ogic **b**lock **o**bserver)*

- The built-in self-tester requires extra logic and flip-flops.
- A better approach is to use existing flip-flops in the design like in the scan chains.



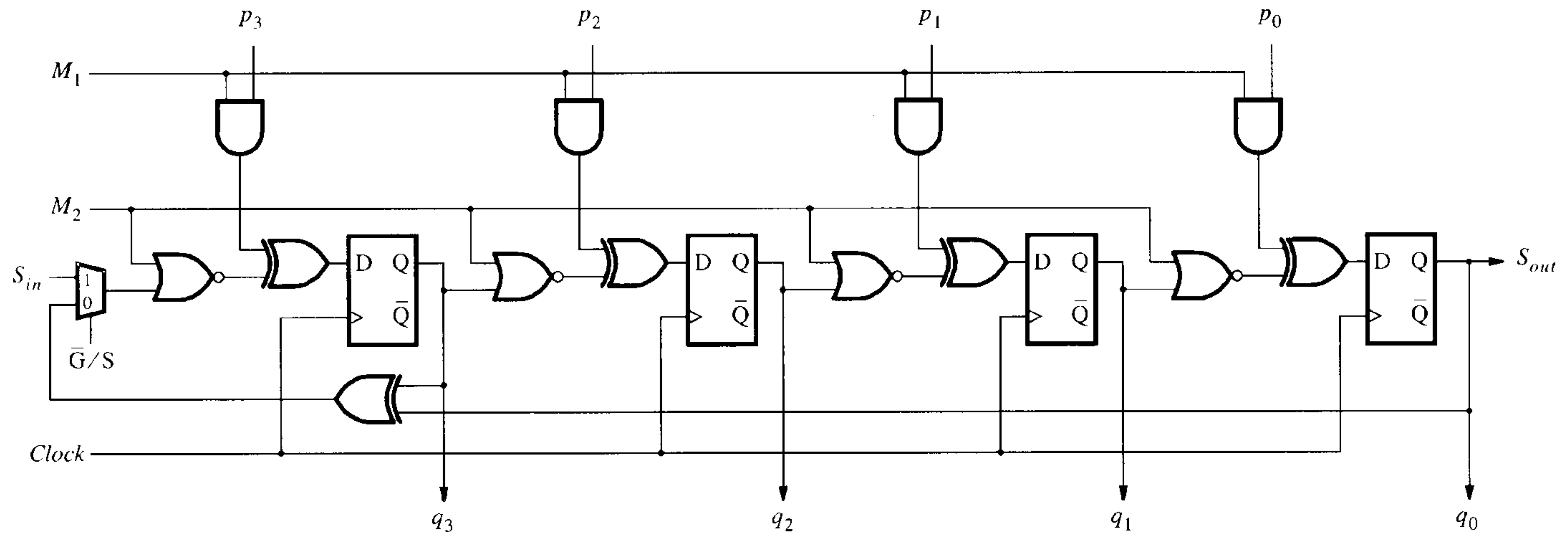
Signature analysis - BILBO

- Before test, BILBO units are tested

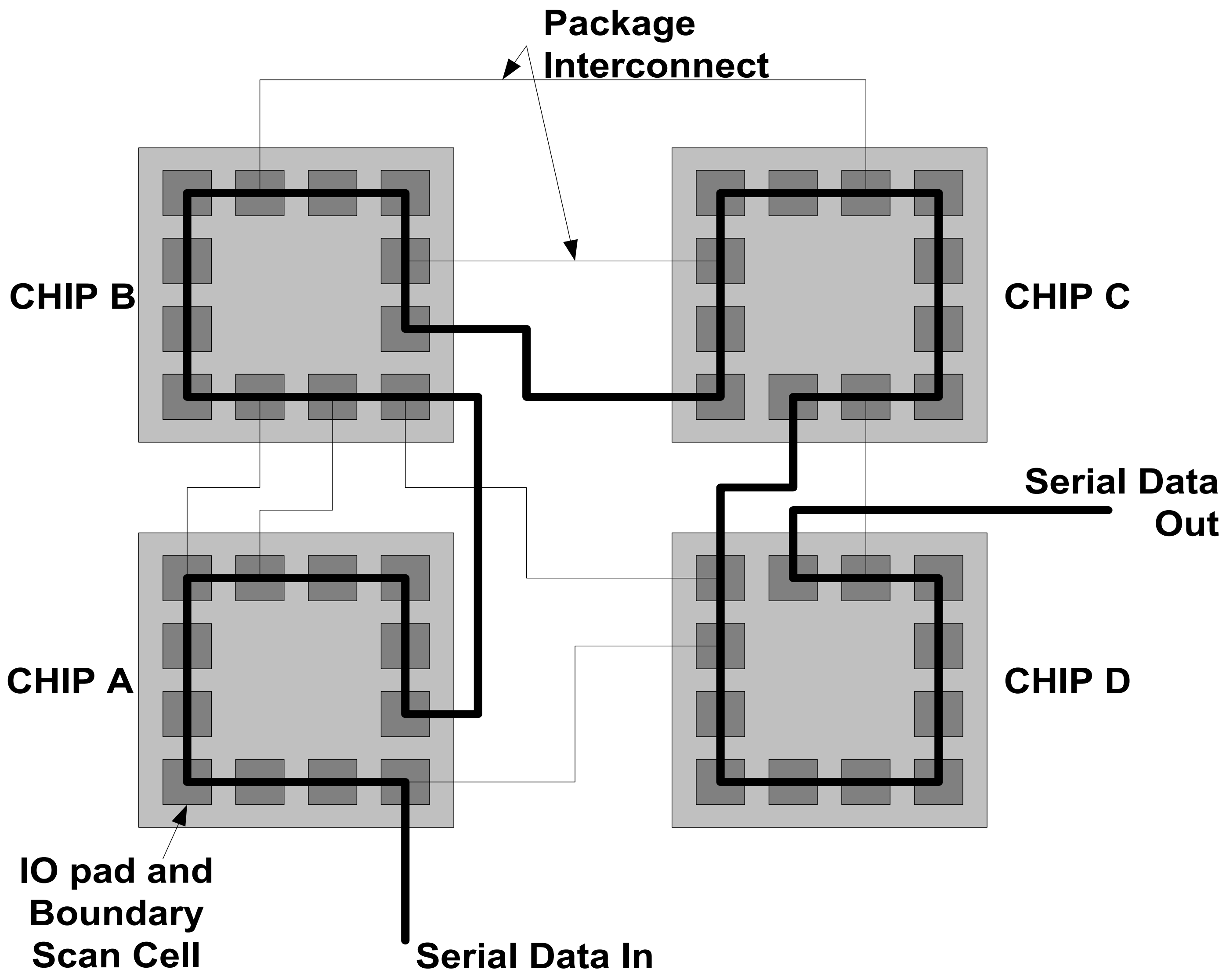


BILBO *(cont'd)*

- 4 different modes
  1. M_1M_2 = 11 Normal mode: flip-flops functioning normally
  2. M_1M_2 = 00 Shift register mode: test-vectors→ in, result→ out
     - ❑ If G/S=0 test vectors are generated
  3. M_1M_2 = 10 Signature mode: inputs→ signature
  4. M_1M_2 = 01 Reset mode: 0→ all flip-flops



Boundary-Scan (IEEE 1149.1)



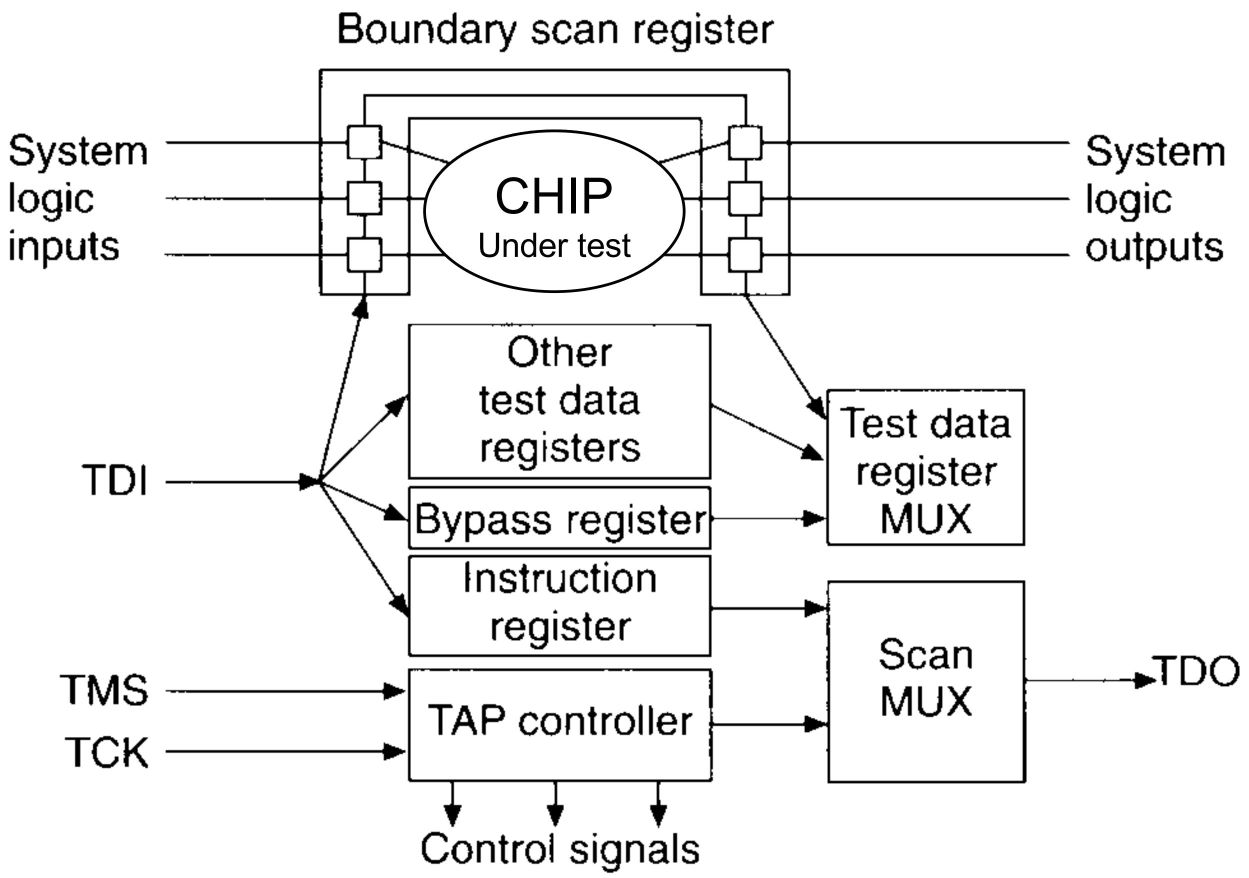
Boundary-Scan (IEEE 1149.1)

- Also known as JTAG
  - Joint Test Action Group
  - **We program our Xilinx FPGAs in the lab with JTAG**
- JTAG consists of:

Test access port (TAP):

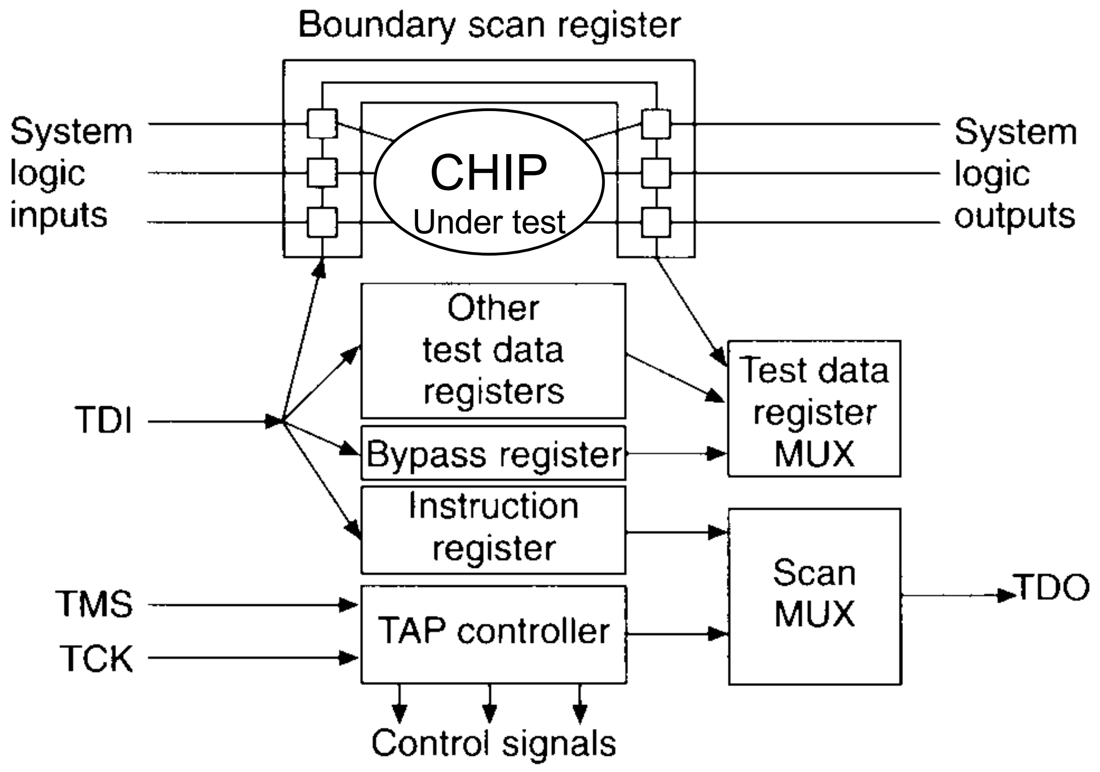
❑ Consists of 5 pins

1. TDI (Test data in)
2. TDO (Test Data Out)
3. TMS (Test Mode Select) control FSM
4. TCK – Test clock
5. TRST (Test Reset) - optional

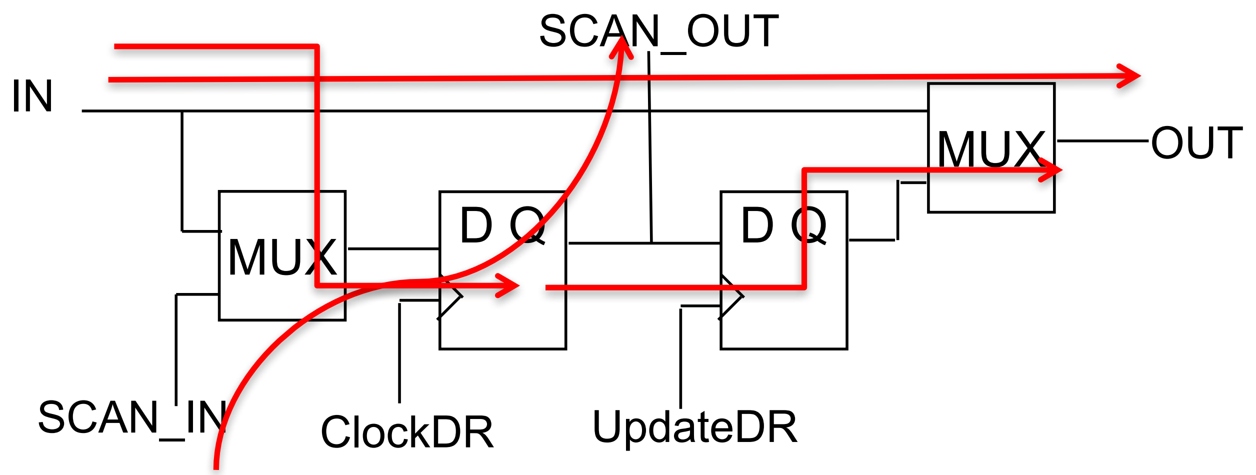


JTAG

1. **Test data registers**
   - All components of the scan-chain must be connected with a particular test-data-register. It moves data to and from the I/O pins of a device
   - Allows bypass of a component.
   - Test-data-register may contain an identifier
2. **Instruction register:**
   - at least two bits. Defines what to do with signals that are received
3. **TAP controller**
   - a state machine that controls the behavior of the JTAG system



Boundary-scan Cell



- Typical **Boundary scan cell**
  - For an input of the component: IN →pin, CORE→OUT,
  - For an output pin: CORE→IN, OUT→pin
- The cell has 4 modes
  1. **Normal mode:** the cell is dicsonnected (IN→OUT)
  2. **Scan mode:** test vectors shifted in, result is checked at the output (SCAN_IN→Left_FF .. also Left_FF to SCAN_OUT)
  3. **Capture mode:** the cell captures the simulation results (IN→Left_FF)
  4. **Update mode:** data are moved from first flip-flop to the next (Left_FF→OUT)

General DFT rules

1. Synchronous design
2. Additional test points (spare pins or multiplexers)
3. Partitioning
4. Initialization
5. Sequential circuits (split them in combinatorial logic and memory elements)
6. Redundancy (must be broken)
7. Analog / digital design (split)
8. Select the test method for the system

Quiz 15-2

http://m.socrative.com/student/#joinRoom

room number: 713113

- Q1: Connect the terms (1) Scan chain, (2) BIST, (3)BILBO, (4) Boundary scan, with the descriptions:
  a. A design that (when in test mode) uses its existing flipflops as test pattern generator and test analyzer
  b. Connects multiple chips that have JTAG on a board
  c. A circuit that has it's own test pattern generator and signature analyzer
  d. The flip-flops of a circuit can be connected with each other (when in test mode) to allow test patterns to be inserted and test results to be sent out.

Note: put a, b, c, d in the correct order to match the order of the terms 1,2,3,4

e.g. dcba

Outline

- Verification vs. Testing
- Faults & Fault Tolerance
- Defects and Yield
- Fault models
  - Stuck-at Faults
  - Fault equivalence
- Testing
  - Combinational & Sequential circuits
  - ATPG
- Design for testing
  - Scan-chains– JTAG
  - Built-in Self-Test (BIST)
  - Boundary-scan

- Book (complimentary to the slides):
  - Chapter 20

- Next Lecture:
  - Timing, Delay, Power