Chapter 3 Basics of VLSI Testing (2)

Jin-Fu Li

Advanced Reliable Systems (ARES) Laboratory

Department of Electrical Engineering

National Central University

Jhongli, Taiwan

Outline

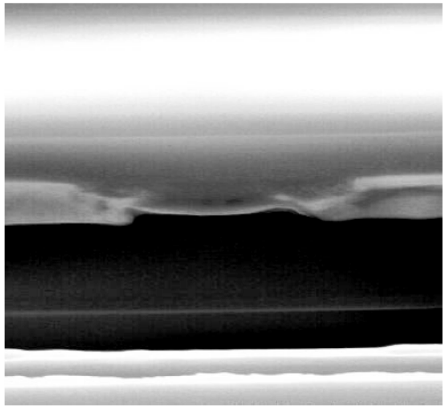
- □ Testing Process
- □ Fault Modeling
- □ Test Pattern Generation
- □ Fault Simulation
- □ Design-for-Testability

Test Process

- The testing problem
  - *Given a set of faults in the circuit under test (or device under test), how do we obtain a certain (small) number of test patterns which guarantees a certain (high) fault coverage?*
- Test process
  - What faults to test? (***fault modeling***)
  - How are test pattern obtained? (***test pattern generation***)
  - How is test quality (fault coverage) measured? (***fault simulation***)?
  - How are test vectors applied and results evaluated? (***ATE/BIST***)

Defect Categories

- Defect categories
  - Random defects, which are independent of designs and processes
  - Systematic defects, which depend on designs and processes used for manufacturing
- For example, random defects might be caused by random particles scattered on a wafer during manufacturing



A resistive open defect [Source: Cadence]

Logical Fault Models

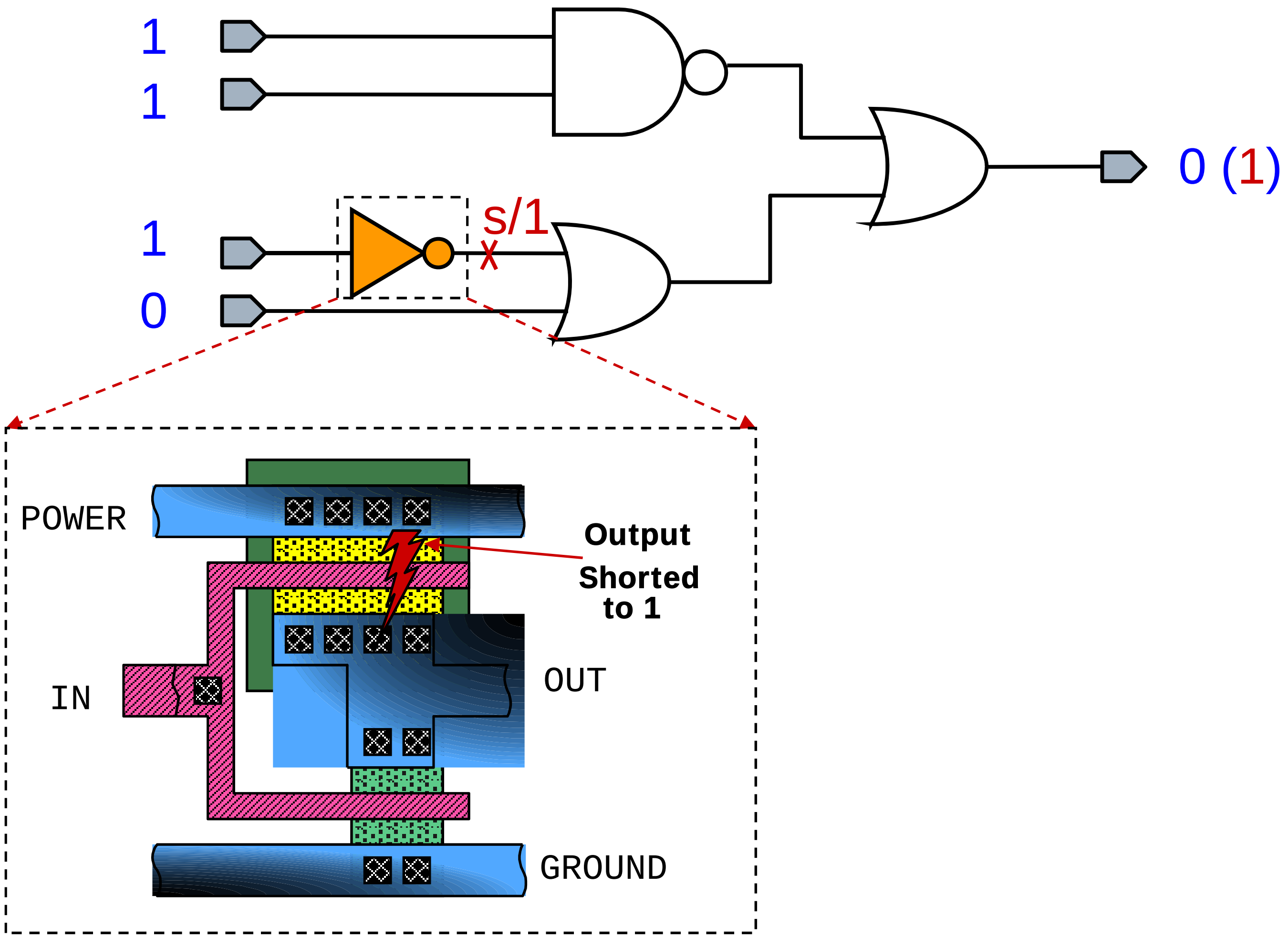
- Systematic defects might be caused by process variations, signal integrity, and design integrity issues.
- It is possible both random and systematic defects could happen on a single die
- With the continuous shrinking of feature sizes, somewhere below the 180nm technology node, system defects have a larger impact on yield than random defects
- Logical faults
  - *Logical faults represent the physical defects on the behaviors of the systems*

Single Stuck-At Fault

- *Single (line) stuck-at fault*
  - The given line has a constant value (0/1) independent of other signal values in the circuit
- Properties
  - 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
  - Simple logical model is independent of technology details
  - It reduces the complexity of fault-detection algorithms
- One stuck-at fault can model more than one kind of defect

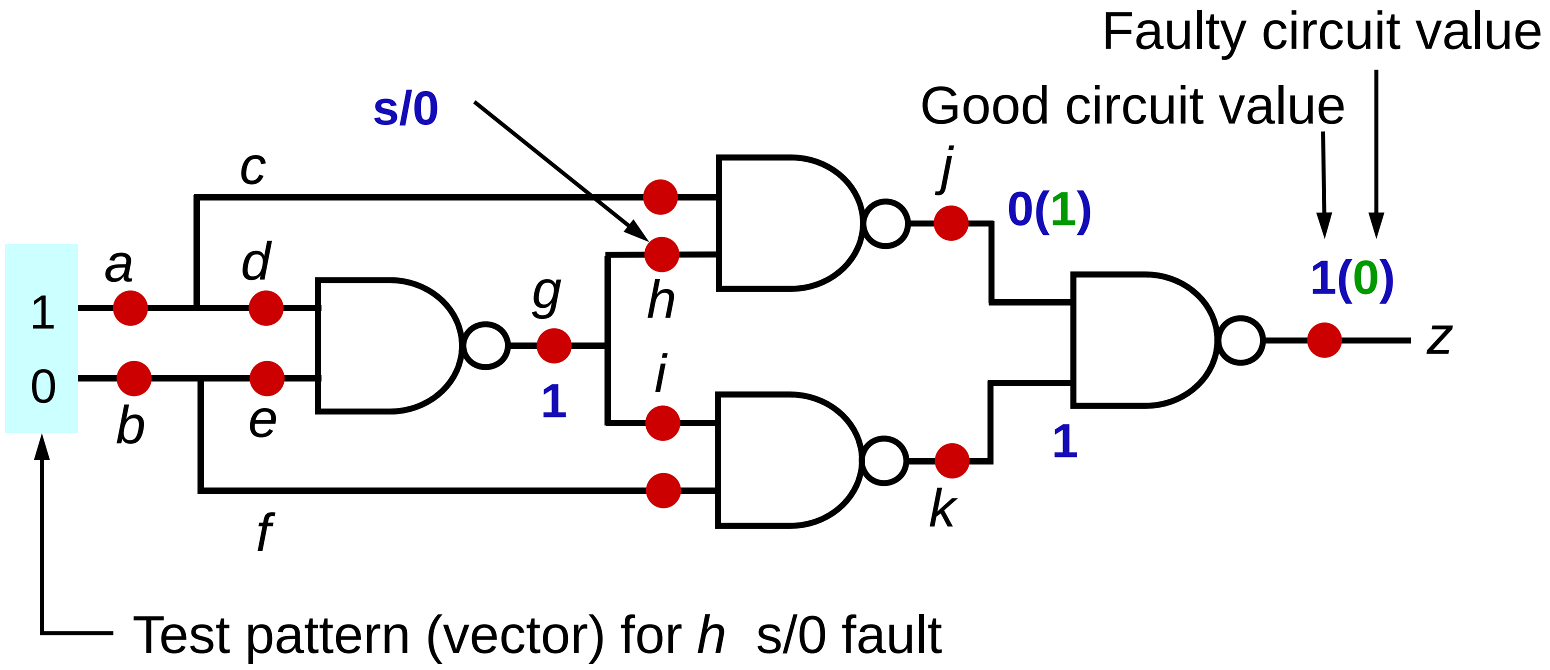
Single Stuck-At Fault Example

- □ A circuit with single stuck-at fault



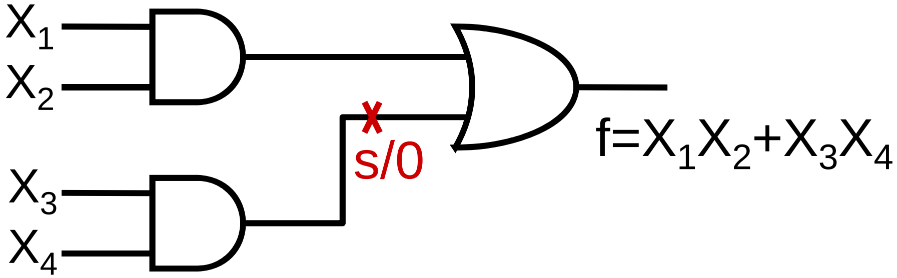
Number of Single Stuck-At Faults

- □ Number of fault sites in a Boolean gate circuit
  - ■ #PI + #gates + #(fanout branches)
- □ Example: XOR circuit has 12 fault sites (●) and 24 single stuck-at faults



Test & Test Set

- A ***test*** for a fault α in a circuit *C* is an input combination for which the output(s) of *C* is different when α is present than when it is not.
  - A.k.a. test pattern or test vector
  - X detect α then $f(X) \oplus f_\alpha(X) = 1$
- A ***test set*** for a class of faults *A* is a set of tests *T* such that $\forall \alpha \in A, \exists t \in T$ and t detects α
  - The test set for a fault α is $T_\alpha = f \oplus f_\alpha$
  - For example,



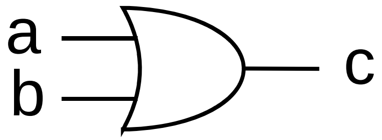
$$
\begin{aligned}
T_\alpha &= f \oplus f_\alpha \\
&= (X_1X_2 + X_3X_4) \oplus X_1X_2 \\
&= \overline{X_1}X_3X_4 + \overline{X_2}X_3X_4 \\
&= \{0011, 0111, 1011\}
\end{aligned}
$$

Testing & Diagnosis

- □ ***Testing*** is a process which includes test pattern generation, test pattern application, and output evaluation.
- □ ***Fault detection*** tells whether a circuit is fault-free or not
- □ ***Fault location*** provides the location of the detected fault
- □ ***Fault diagnosis*** provides the location and the type of the detected fault
  - ■ The input X distinguishes a fault α from another fault β iff $f_\alpha(X) \neq f_\beta(X)$, i.e., $f_\alpha(X) \oplus f_\beta(X) = 1$

Testing & Diagnosis

- □ Example:

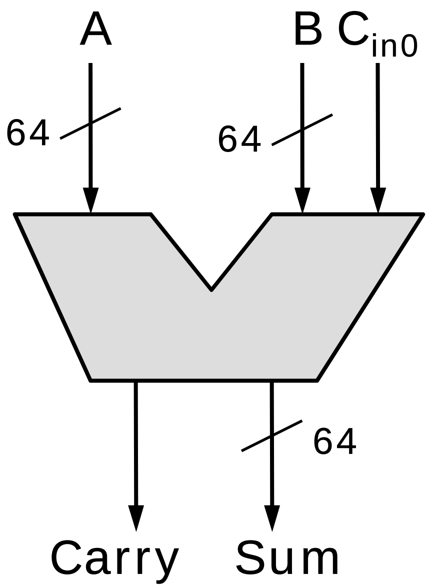


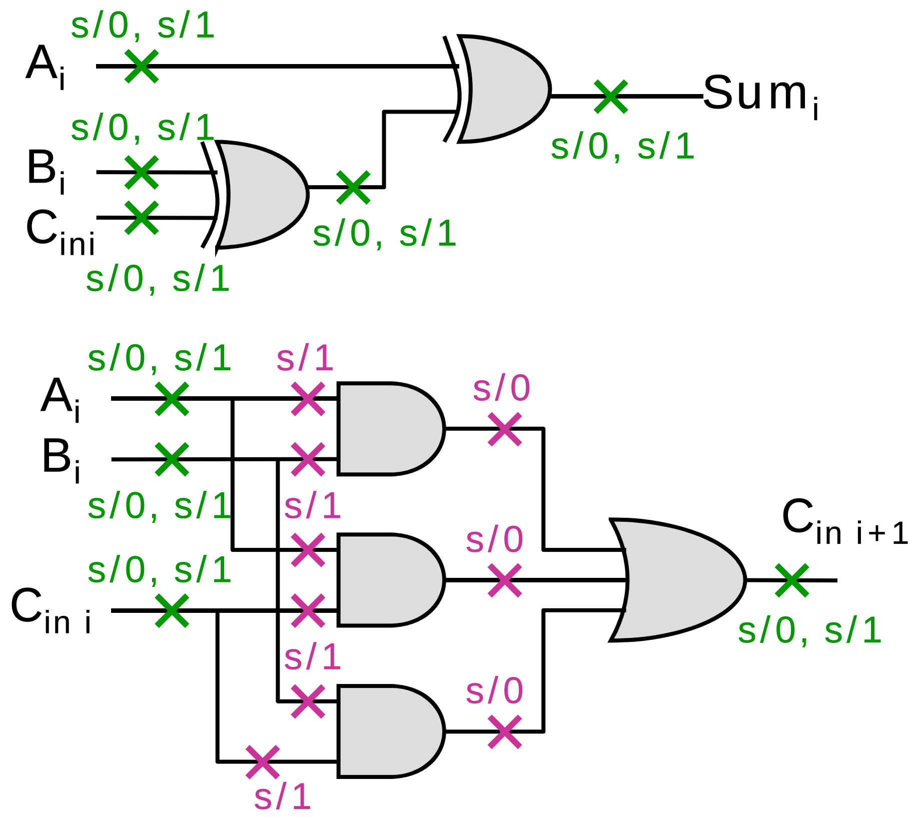
a	b	c	$c_{a/0}$	$c_{a/1}$	$c_{b/0}$	$c_{b/1}$	$c_{c/0}$	$c_{c/1}$
0	0	0	0	**1**	0	**1**	0	**1**
0	1	1	1	1	**0**	1	**0**	1
1	0	1	**0**	1	1	1	**0**	1
1	1	1	1	1	1	1	**0**	1

- □ $C_{a/0}$ and $C_{c/0}$ are detected by the test pattern (1,0)
- □ If we apply two test patterns: (1,0) & (0, 1)
  - Two corresponding outputs are faulty→$C_{c/0}$
  - Only the output with respect to the input (1,0) is faulty→$C_{a/0}$

Functional vs. Structural Test

- Consider a 64-bit adder as shown below





Functional Test

Structural Test

Functional vs. Structural Test

- Functional test
  - Generate complete set of tests for circuit input-output combinations
  - 129 inputs & 65 outputs
  - 2^{129}=680,564,733,841,876,926,926,749,214, 863,536,422,912 test patterns are required
  - Using 1 GHz ATE, would take 2.15 x 10^{22} years
- Structural test
  - 64 bit slices and each slice has 27 faults (using fault collapsing)
  - At most 64x27=1728 faults, thus only 1728 test patterns are required
  - Takes 0.000001728 seconds on 1 GHz ATE

Algorithm Types of Test Pattern Generation

- □ Exhaustive test generation
  - Completely exercise the fault-free behavior
  - Appropriate only when the number of PIs is small
  - Detects all the universal faults (i.e., all combinational faults)
- □ Pseudoexhaustive test generation
  - Test most of universal faults by applying exhaustive test on subsets of PIs
- □ Pseudorandom test generation
  - Generate test pattern deterministically
  - Patterns have many characteristics of random patterns but are repeatable

Algorithm Types of Test Pattern Generation

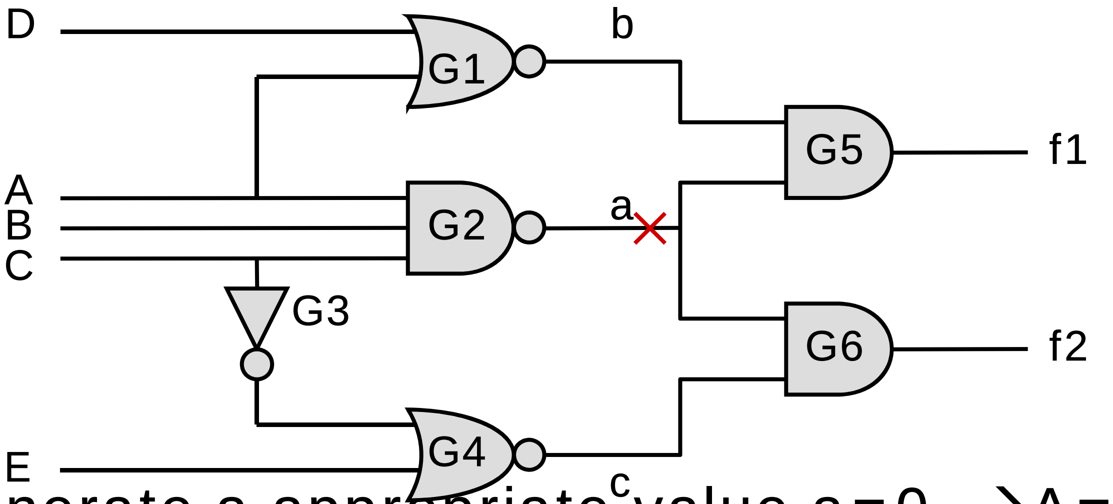
- Algorithmic (deterministic) test generation
  - Algebraic (symbolic) techniques
    - SPOOFs
    - Line condition equations
    - Boolean difference
  - Path-oriented techniques
    - Single-path sensitization
    - D-algorithm
    - PODEM
    - FAN
  - Produces higher-efficiency test patterns, but its cost is more expensive

Test Pattern Generation Procedure

- □ *Fault activation or excitation*
  - Specify inputs so as to generate the appropriate value at the fault site, i.e., 0 for s/1 and 1 for s/0
- □ *Fault propagation*
  - Select a path from the fault site to an output and specify other signal values to propagate the fault (error signal) along the path to the output
- □ *Line justification*
  - Specify input values so as to produce the signal values specified in fault activation and fault propagation, i.e., perform *consistency* check

An Example

- □ Use single-path sensitization to derive a test set for $\alpha \equiv a/1$ in the following circuit



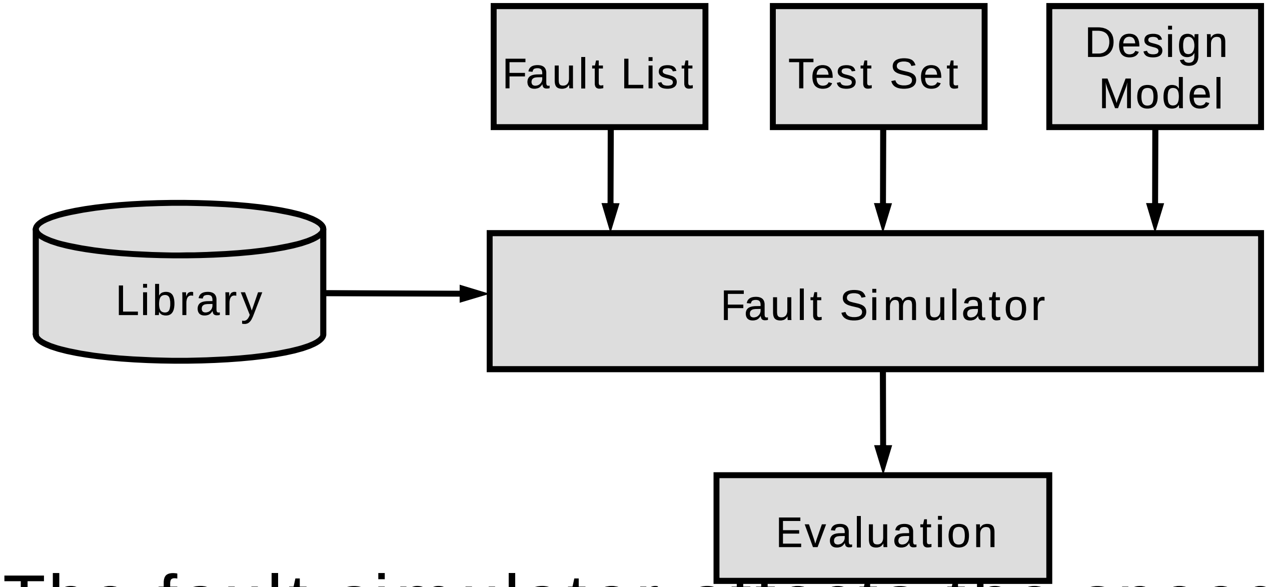
- Generate a appropriate value a=0. →A=B=C=1
- Choose a path via G5→b=1→A=D=0. Contradiction!
- Try another path via G6→c=1→C=1 and E=0. OK! Therefore, T=ABCE'

Fault Simulation

- Fault simulation
  - In general, simulating a circuit in the presence of faults is known as fault simulation
- The main goals of fault simulation
  - Measuring the effectiveness of the test patterns
  - Guiding the test pattern generator program
  - Generating fault dictionaries
- Outputs of fault simulation
  - Fault coverage - fraction (or percentage) of modeled faults detected by test vectors
  - Set of undetected faults

Elements of Fault Simulation

- The fault simulation process is illustrated as below



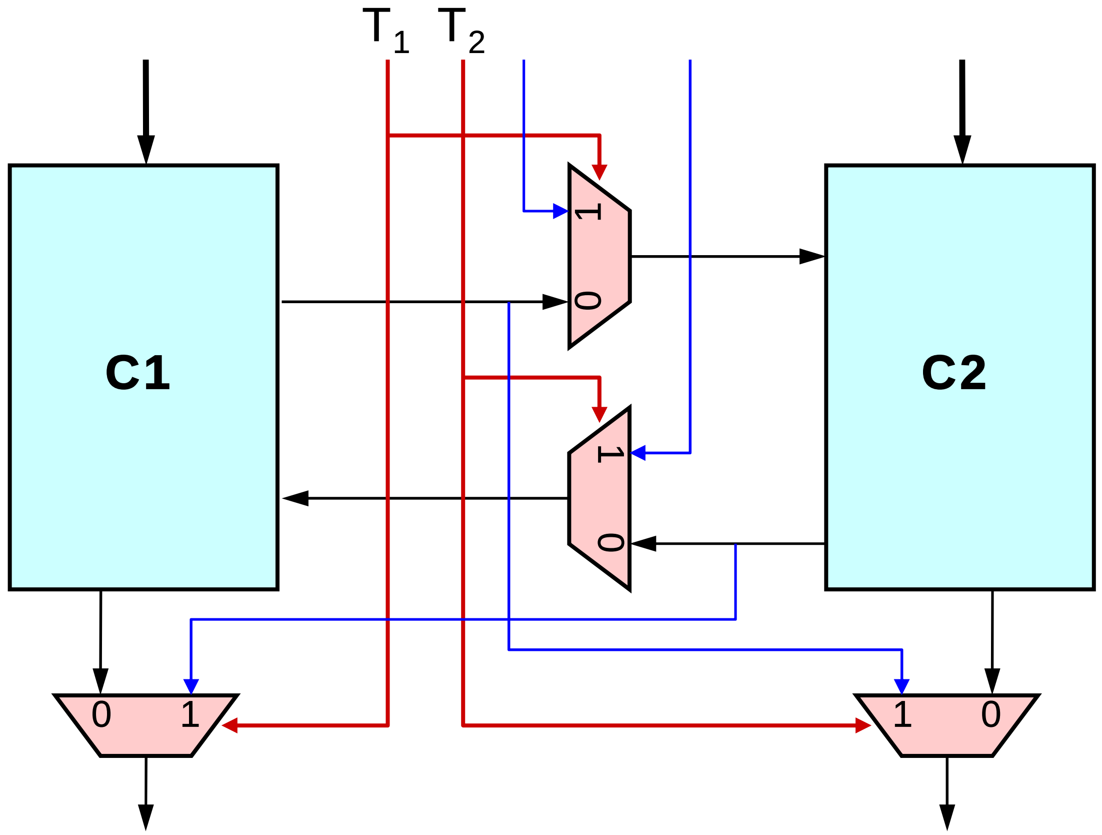
- The fault simulator affects the speed of overall fault simulation

Design for Testability

- Definition
  - A fault is ***testable*** if there exists a well-specified procedure to expose it, which is implementable with a reasonable cost using current technologies. A circuit is ***testable with respect to a fault set*** when each and every fault in this set is testable
- Definition
  - ***Design for testability*** **(DFT)** refers to those design techniques that make test generation and test application cost-effective
- Electronic systems contain three types of components: (a) digital logic, (b) memory blocks, and (c) analog or mixed-signal circuits
- Here, we only discuss DFT techniques for digital logic

Ad Hoc DFT Guidelines

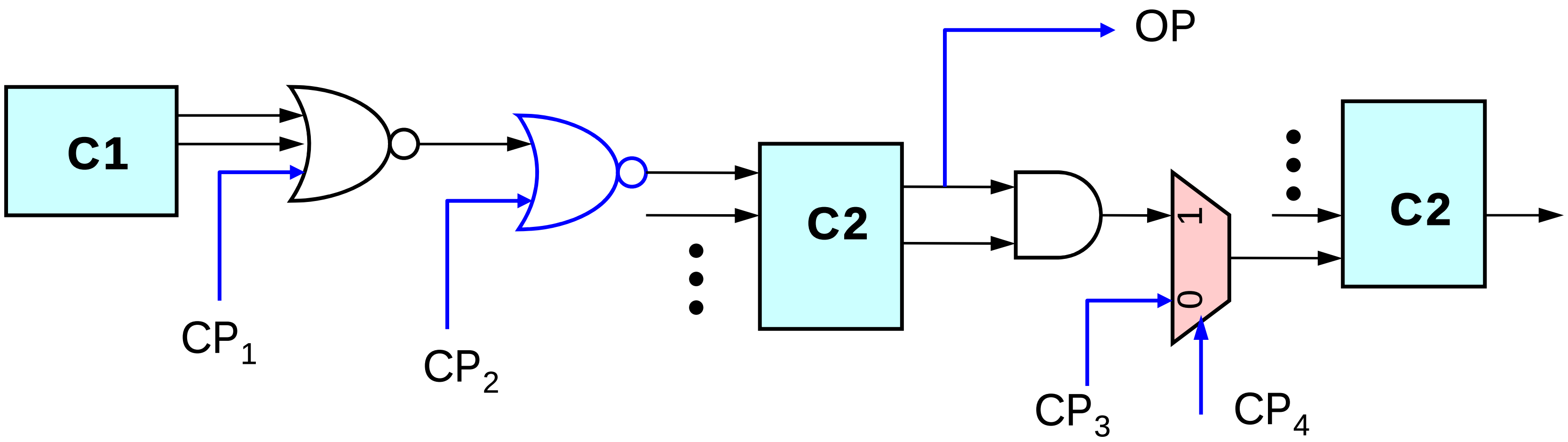
□ Partition large circuits into smaller subcircuits to reduce test generation cost (using MUXed and/or scan chains)



Mode	T_1	T_2
Normal	0	0
Test C1	0	1
Test C2	1	0

Ad Hoc DFT Guidelines

- Insert test points to enhance controllability & observability
  - Test points: control points & observation points

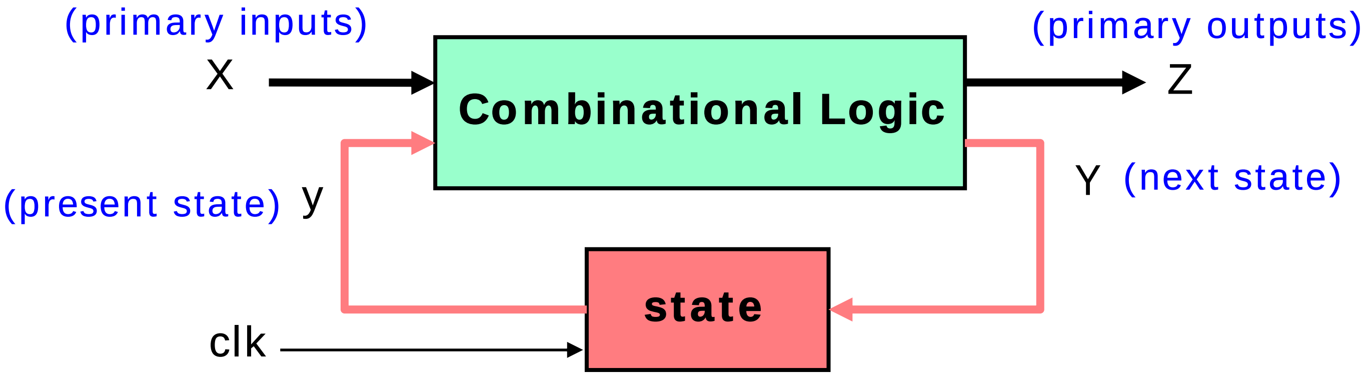


Ad Hoc DFT Guidelines

- Design circuits to be easily initializable
- Provide logic to break global feedback paths
- Partition large counter into smaller ones
- Avoid the use of redundant logic
- Keep analog and digital circuits physically apart
- Avoid the use of asynchronous logic
- Consider tester requirements (pin limitation, etc)
- Etc

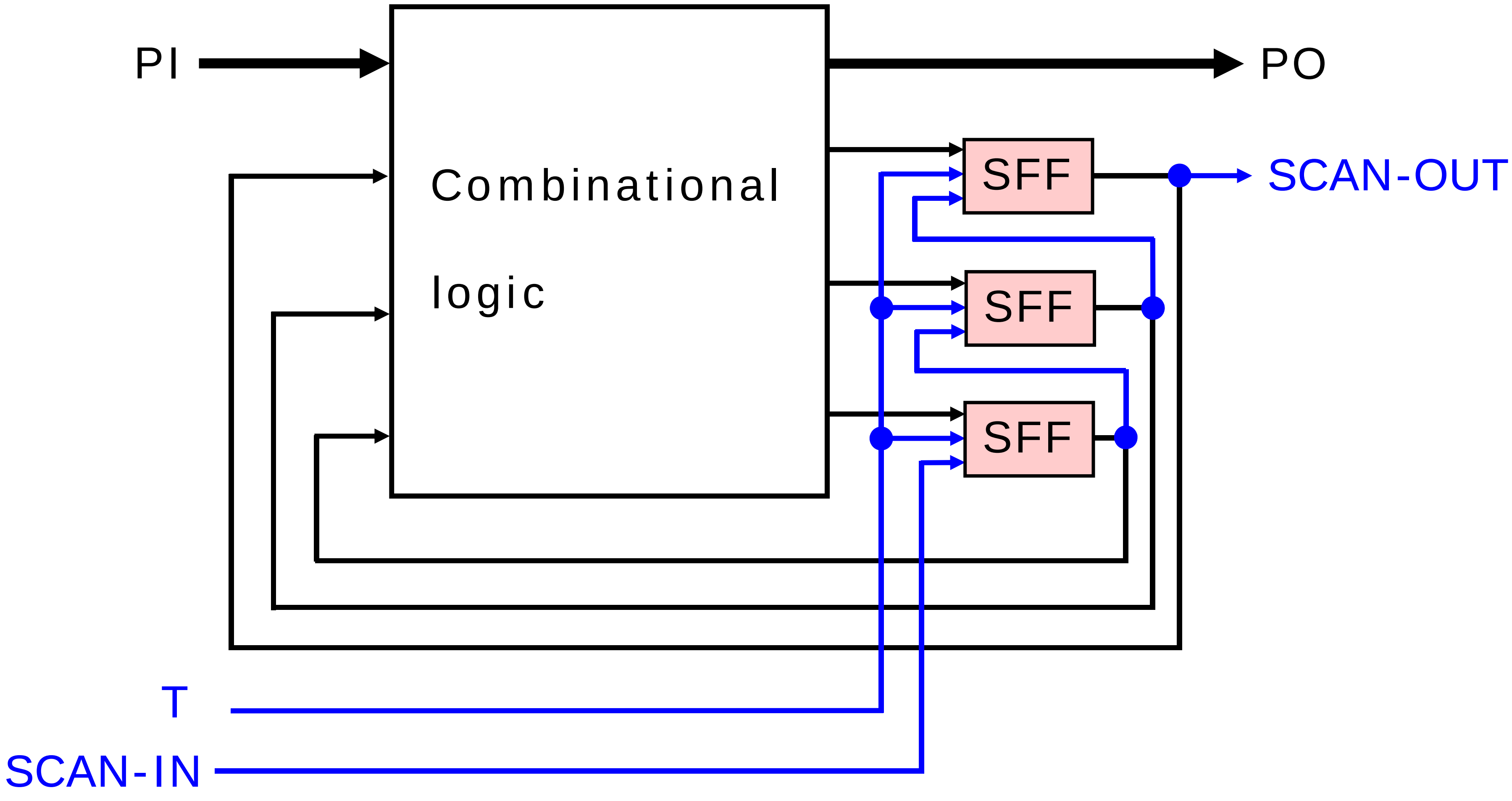
Scan Design Approaches

- Consider a representation of sequential circuits



- To make elements of state vector controllable and observable, we add
  - A TEST mode pin (T)
  - A SCAN-IN pin (SI)
  - A SCAN-OUT pin (SO)
  - A MUX (switch) in front of each FF (M)

Adding Scan Structure



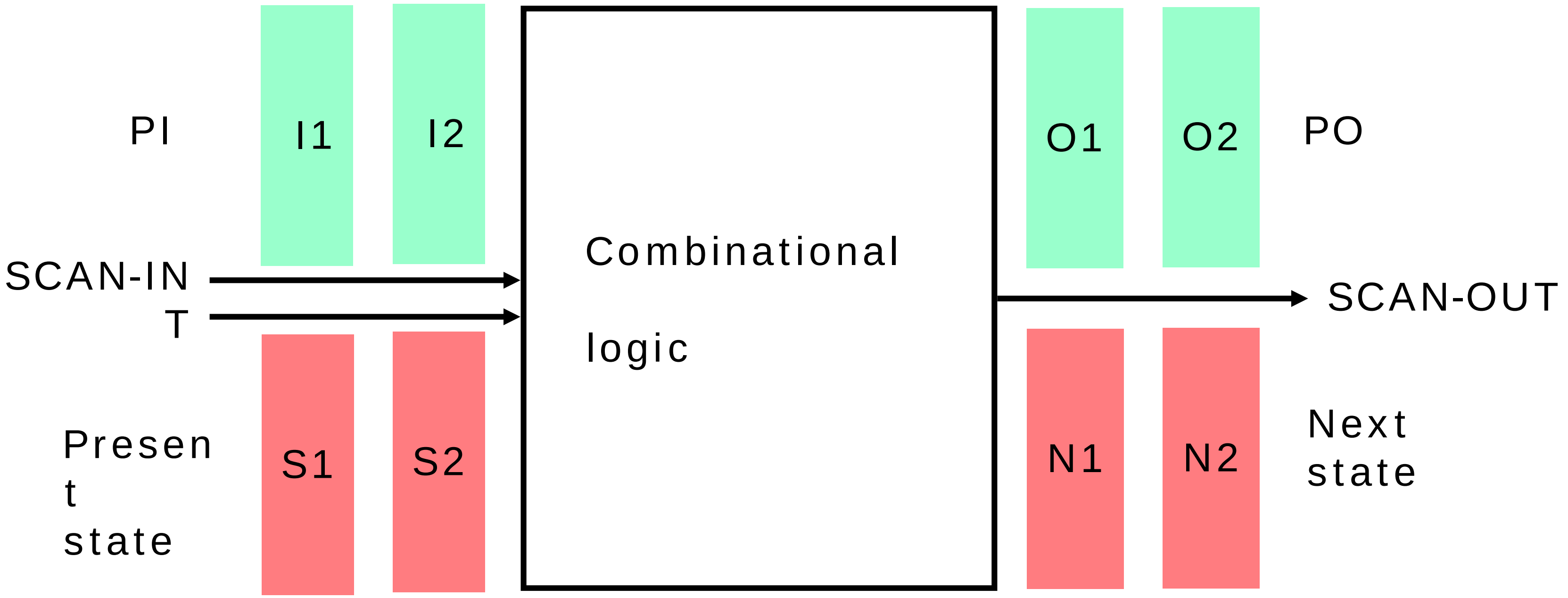
Scan Test Generation & Design Rules

- Test pattern generation
  - Use combinational ATPG to obtain tests for all testable faults in the combinational logic
  - Add shift register tests and convert ATPG tests into scan sequences for use in manufacturing test
- Scan design rules
  - Use only clocked D-type of flip-flops for all state variables
  - At least one PI pin must be available for test; more pins, if available, can be used
  - All clocks must be controlled from PIs
  - Clocks must not feed data inputs of flip-flops

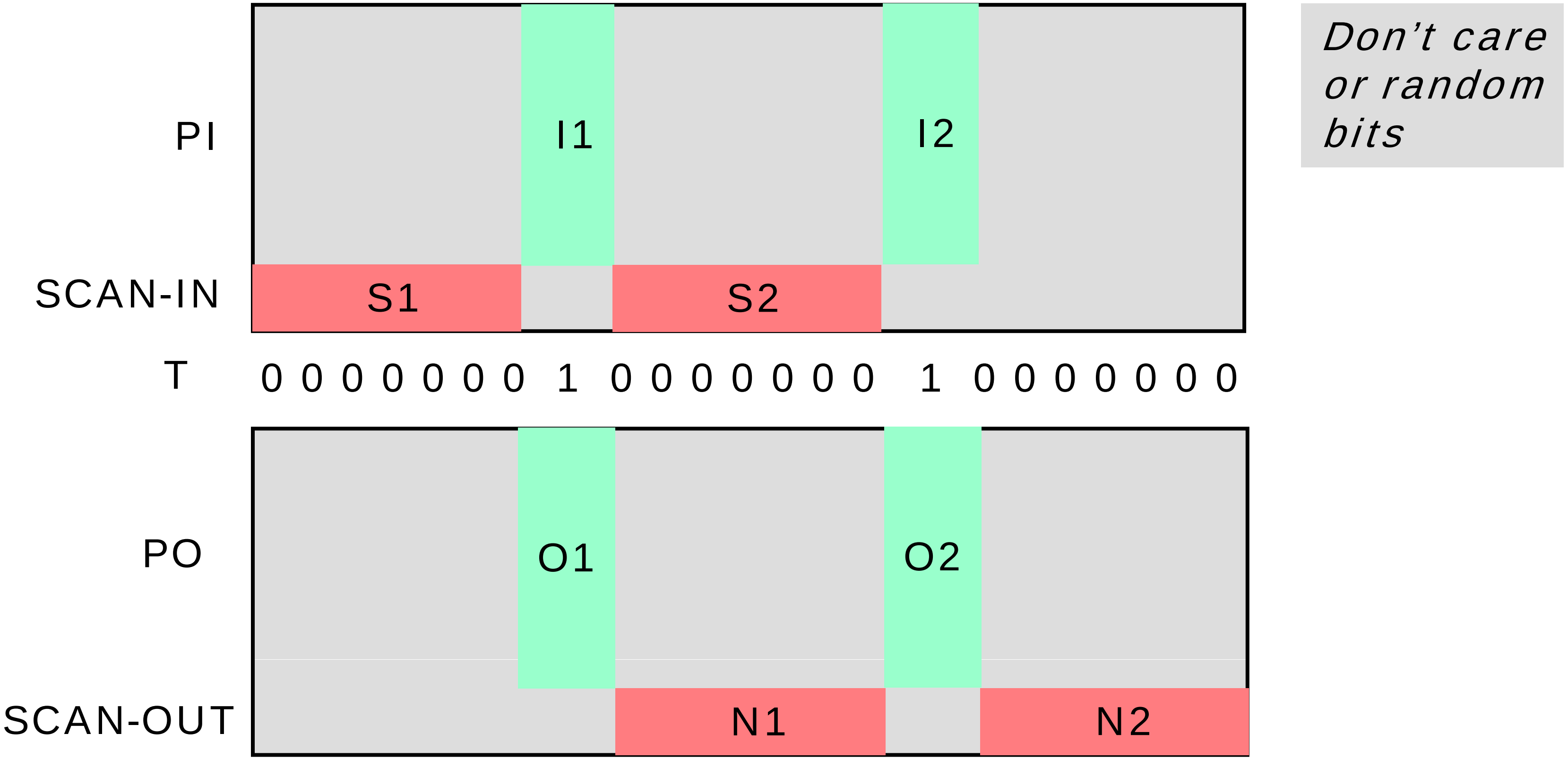
Scan Test Procedure

- □ **Step 1**: Switch to the shift-register mode and check the SR operation by shifting in an alternating sequence of 1s and 0s, e.g., 00110 (functional test)
- □ **Step 2**: Initialize the SR---load the first pattern
- □ **Step 3**: Return to the normal mode and apply the test pattern
- □ **Step 4**: Switch to the SR mode and shift out the final state while setting the starting state for the next test. Go to **Step 3**

Combining Test Vectors



Combining Test Vectors



Sequence length = $(n_{comb} + 1)\ n_{sff} + n_{comb}$ *clock periods*

n_{comb} = number of combinational vectors

n_{sff} = number of scan flip-flops

Testing Scan Register

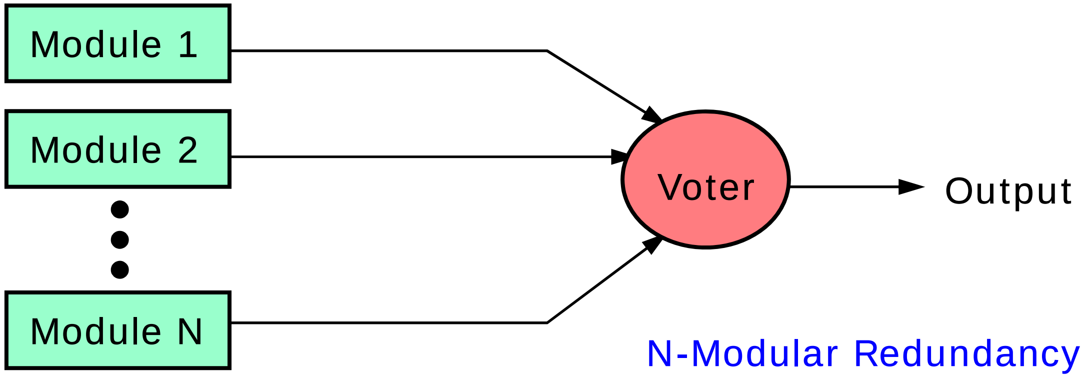
- □ Scan register must be tested prior to application of scan test sequences
- □ A shift sequence 00110011 . . . of length $n_{\text{sff}}+4$ in scan mode (TC=0) produces 00, 01, 11 and 10 transitions in all flip-flops and observes the result at SCAN-OUT output
- □ Total scan test length: $(n_{\text{comb}}+2)n_{\text{sff}}+n_{\text{comb}}+4$ *clock periods*
- □ Example: 2,000 scan flip-flops, 500 comb. vectors, total scan test length ~ 10^6 clocks
- □ Multiple scan registers reduce test length

Introduction to Built-In Self-Test

- Built-in self-test (BIST):
  - The capability of a circuit (chip/board/system) to test itself
- Advantages of BIST
  - Test patterns generated on-chip →controllability increased
  - (Compressed) response evaluated on-chip →observability increased
  - Test can be on-line (concurrent) or off-line
  - Test can run at circuit speed →more realistic; shorter test time; easier delay testing
  - External test equipment greatly simplified, or even totally eliminated
  - Easily adopting to engineering changes

Introduction to Built-In Self-Test

- On-line BIST
  - Concurrent (EDAC, NMR, totally self-checking checkers, etc.):
    - Coding or modular redundancy techniques (fault tolerance)

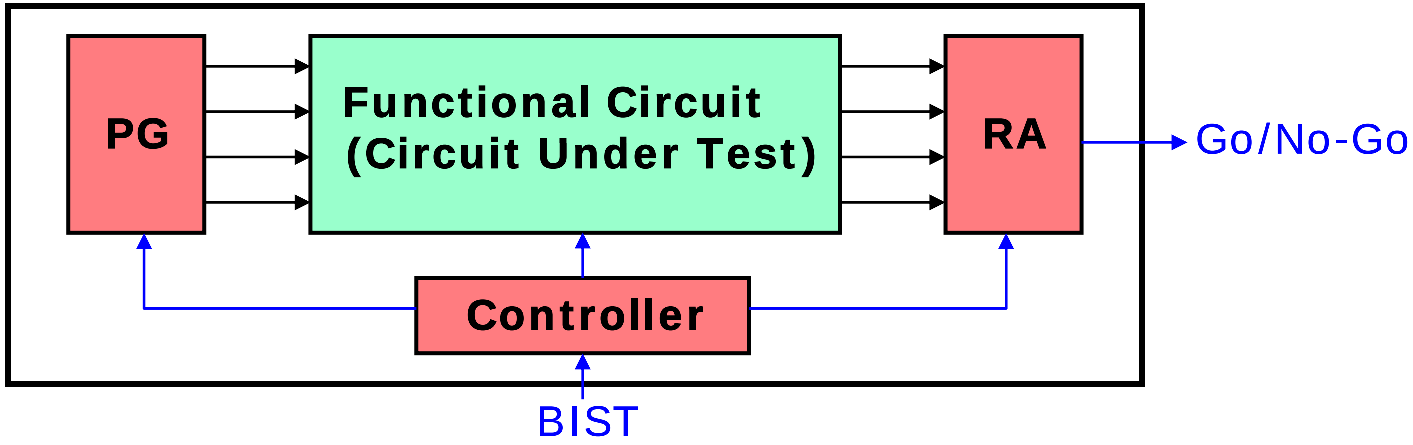


N-Modular Redundancy (NMR)

    - Instantaneous correction of errors caused by temporary or permanent faults
  - Nonconcurrent (diagnostic routines):
    - Carried out while a system is in an idle state

Introduction to Built-In Self-Test

- Off-line BIST
  - A typical BIST architecture



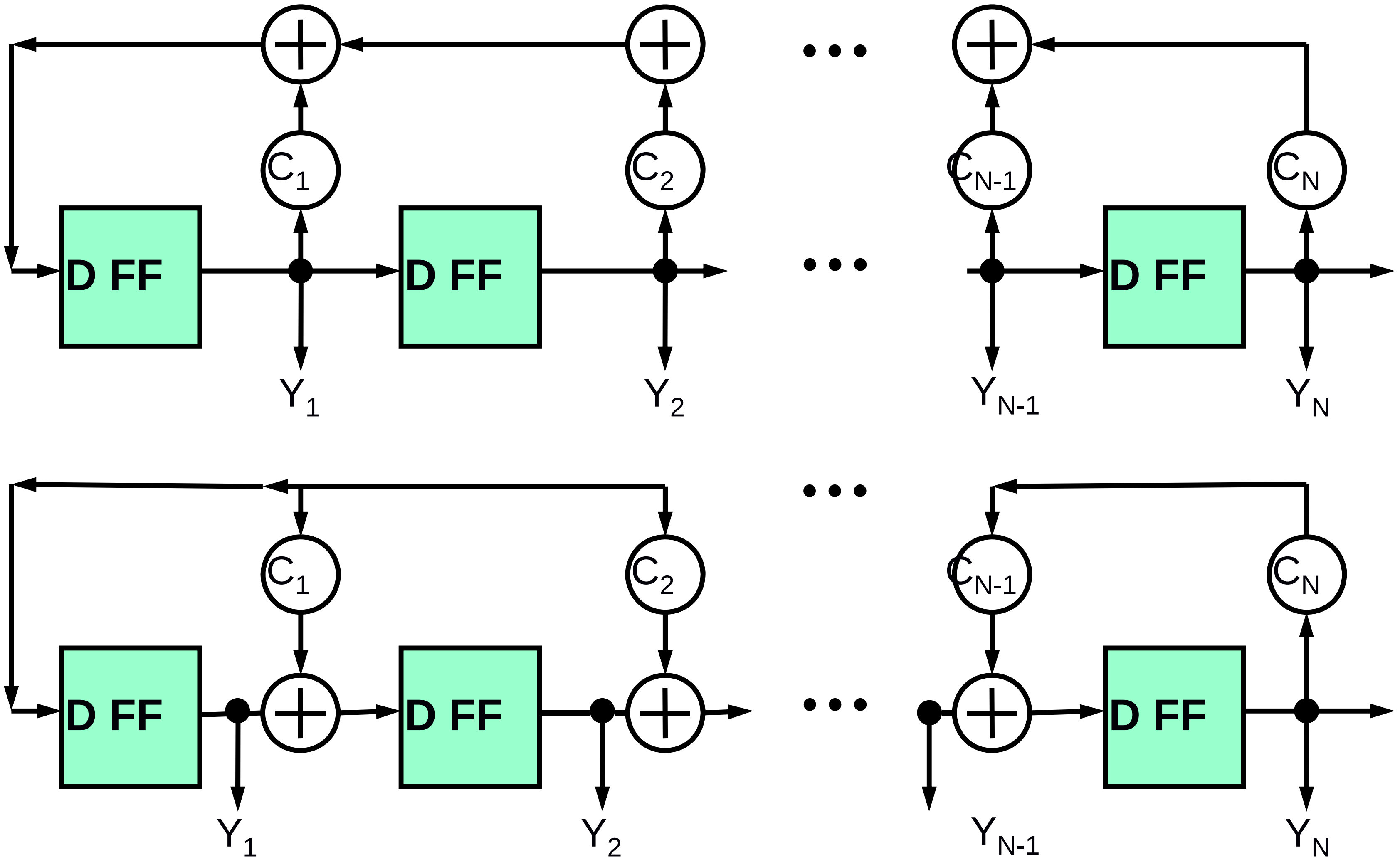
  - Test generation
    - Prestored TPG, e.g., ROM or shift register
    - Exhaustive TPG, e.g., binary counter
    - Pseudo-exhaustive TPG, e.g., constant-weight counter, combined LFSR and SR
    - Pseudo-random pattern generator, e.g., LFSR

Introduction to Built-In Self-Test

- Response analysis
  - Check-sum
  - Ones counting
  - Transition counting
  - Parity checking
  - Syndrome analysis
  - Etc.
- Linear feedback shift register (LFSR) can be both the test generator and response analyzer
- We need a gold unit to generate the good signature or a simulator

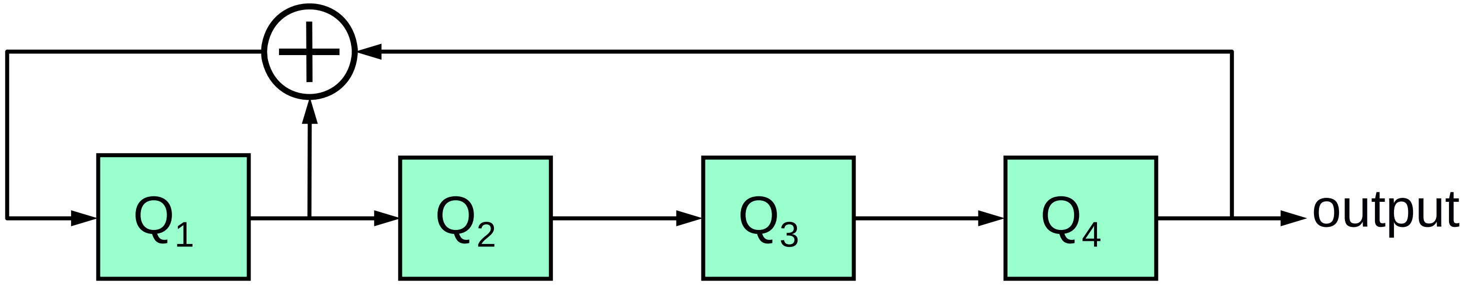
Structures of LFSR

- □ Two types of generic standard LFSRs



Pseudorandom Pattern Generator (PRPG)

- Example: the following ALFSR generates the pseudorandom sequence shown in the table below

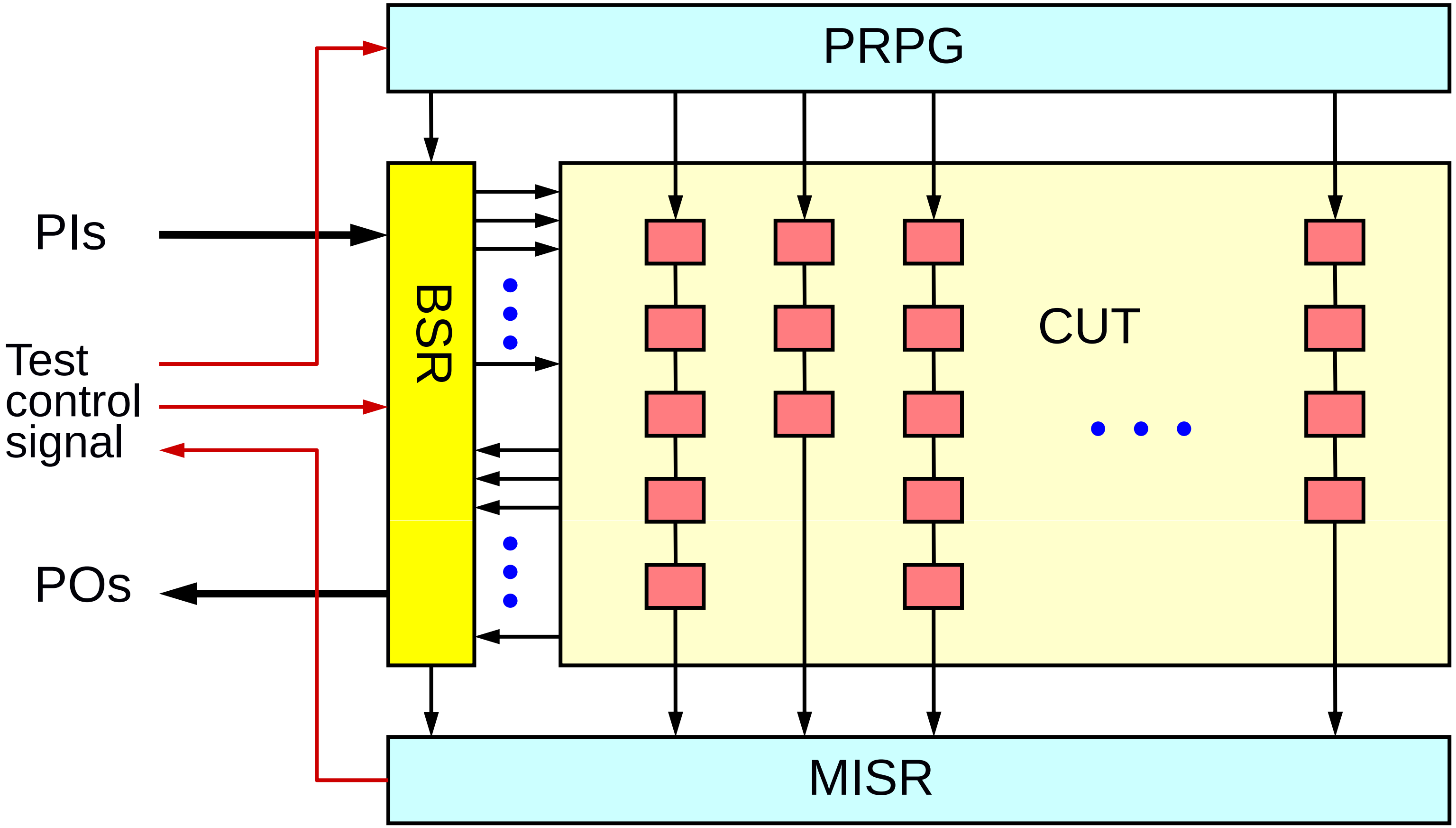


State	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15=0
Q_1	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
Q_2	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0
Q_3	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0
Q_4	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0

  - The output sequence is 000111101011001, which repeats after 15(2^n-1) clocks
  - Max period for an n-stage ALFSR=2^n-1
  - All-0 state of the register cannot occur in the max-length cycle

STUMPS Architecture

- Logic BIST with STUMPS architecture


PRPG
PIs
BSR
CUT
Test
control
signal
POs
MISR