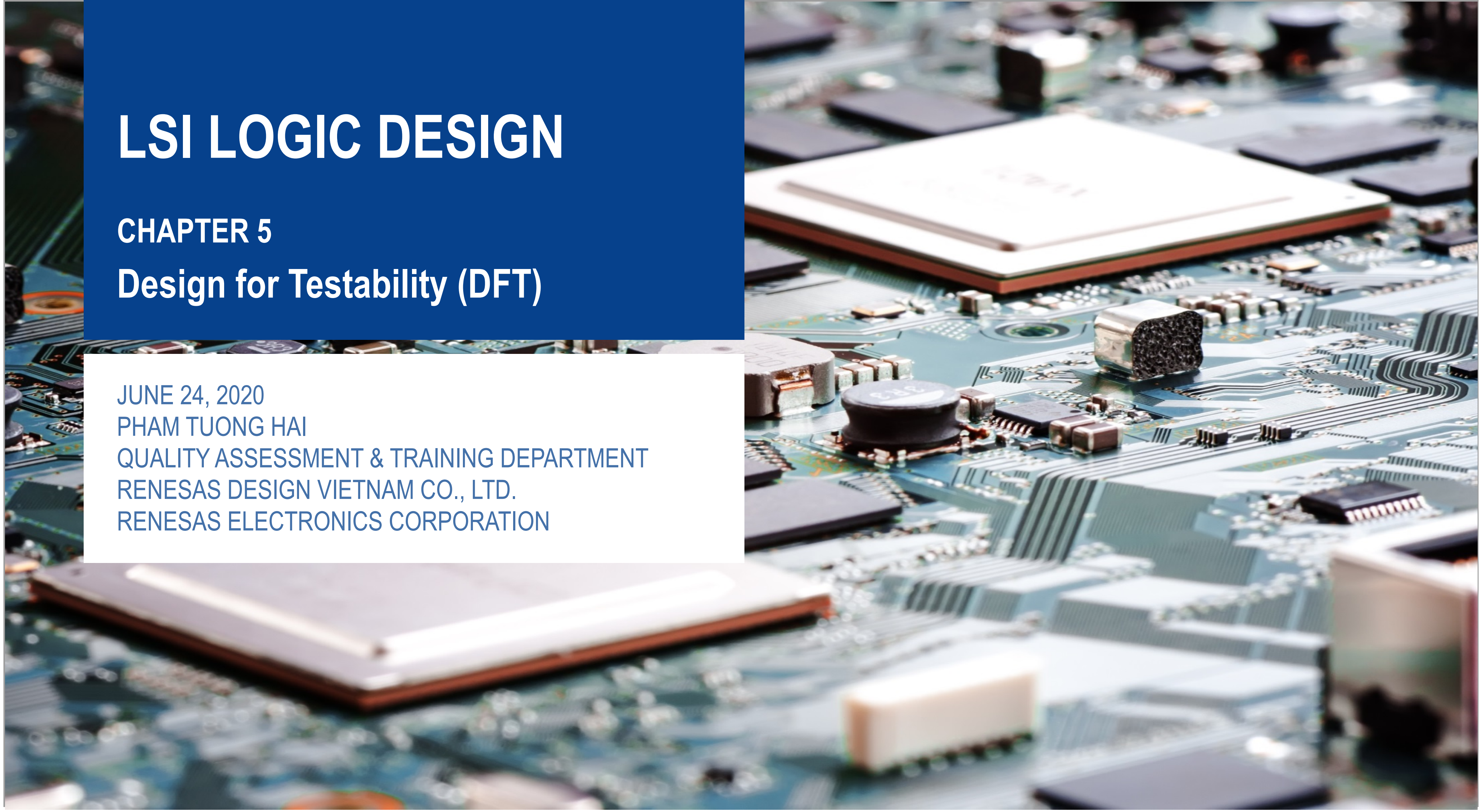


LSI LOGIC DESIGN

CHAPTER 5

Design for Testability (DFT)

JUNE 24, 2020

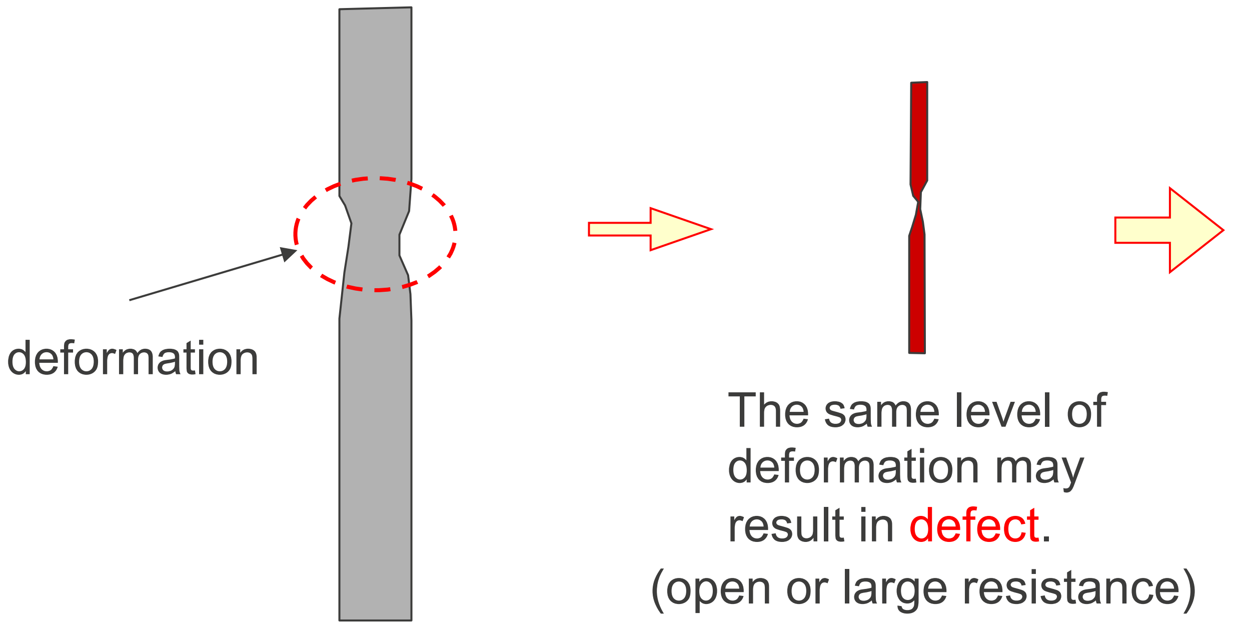
PHAM TUONG HAI
QUALITY ASSESSMENT & TRAINING DEPARTMENT
RENESAS DESIGN VIETNAM CO., LTD.
RENESAS ELECTRONICS CORPORATION

CHAPTER 5. Design for Testability (DFT)

5.1 Manufacturing Defects

Manufacturing Defects

Manufacturing defects are physical defects happening during the manufacturing time.



The same level of deformation may result in defect.
(open or large resistance)

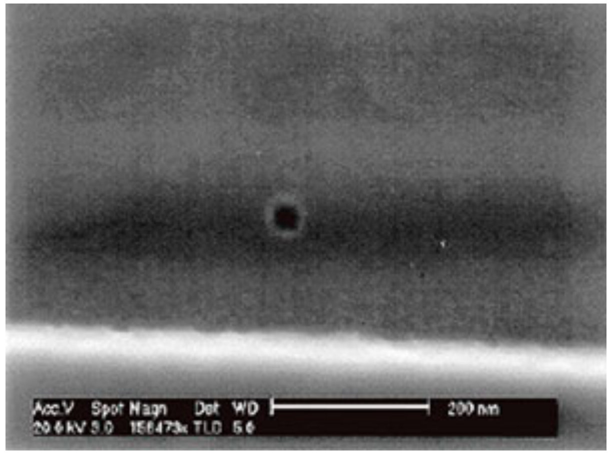
In Deep submicron era, we have to be sensitive to this kind of manufacturing defects because they result in malfunction of the products.

In former process generation, the deformation of the wiring shown above did not cause any problem.

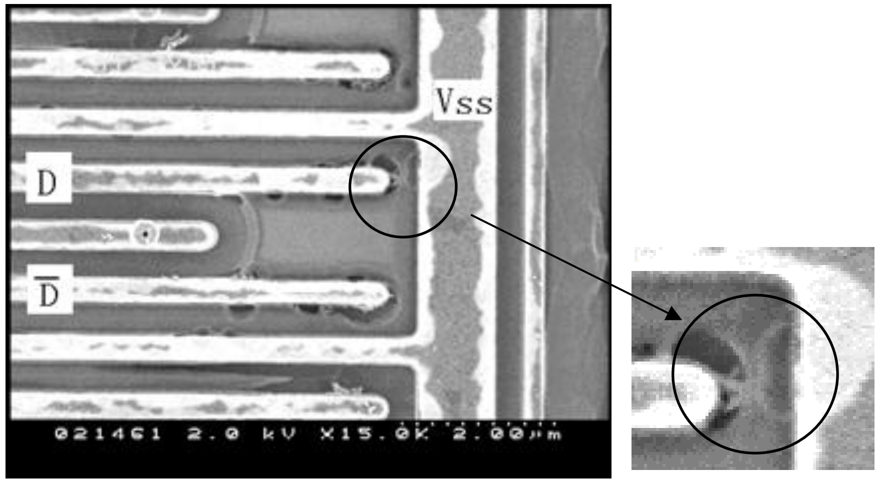
We have to apply some methods to check if the products are manufactured without defects or not before shipping them to the market.

Physical Defects

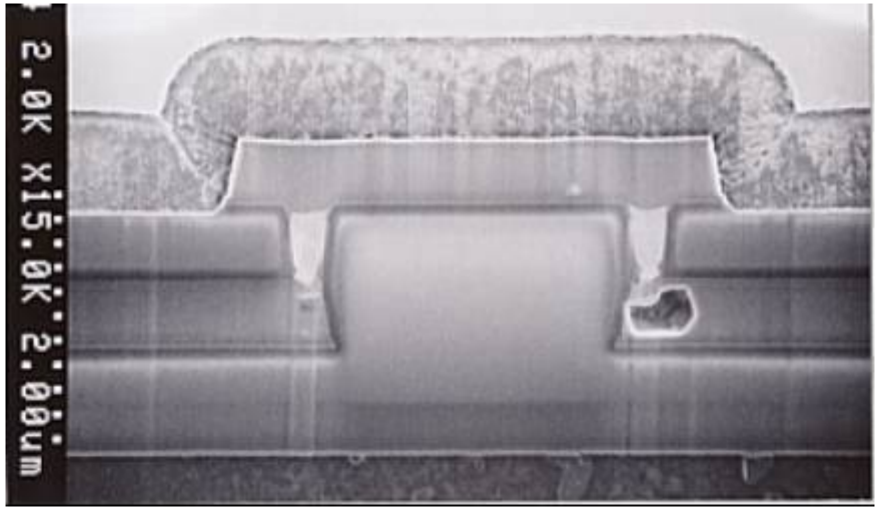
- Abnormality in MOS Transistor
  - ☞ Abnormality in Gate Oxide
  - ☞ Shortage in MOS Transistor
  - ☞ Abnormality in wire width, space and thickness
  - ☞ Contamination due to foreign particle
- Shortage in wiring
  - ☞ Etching residuum
  - ☞ Bridge due to foreign particle
  - ☞ Abnormal wider width and thicker thickness
- Disconnection in wiring
  - ☞ Non-conductive via and higher resistance of via
  - ☞ Abnormal narrower width and thinner thickness
  - ☞ Half disconnection due to foreign particle and void



Gate oxide pinhole



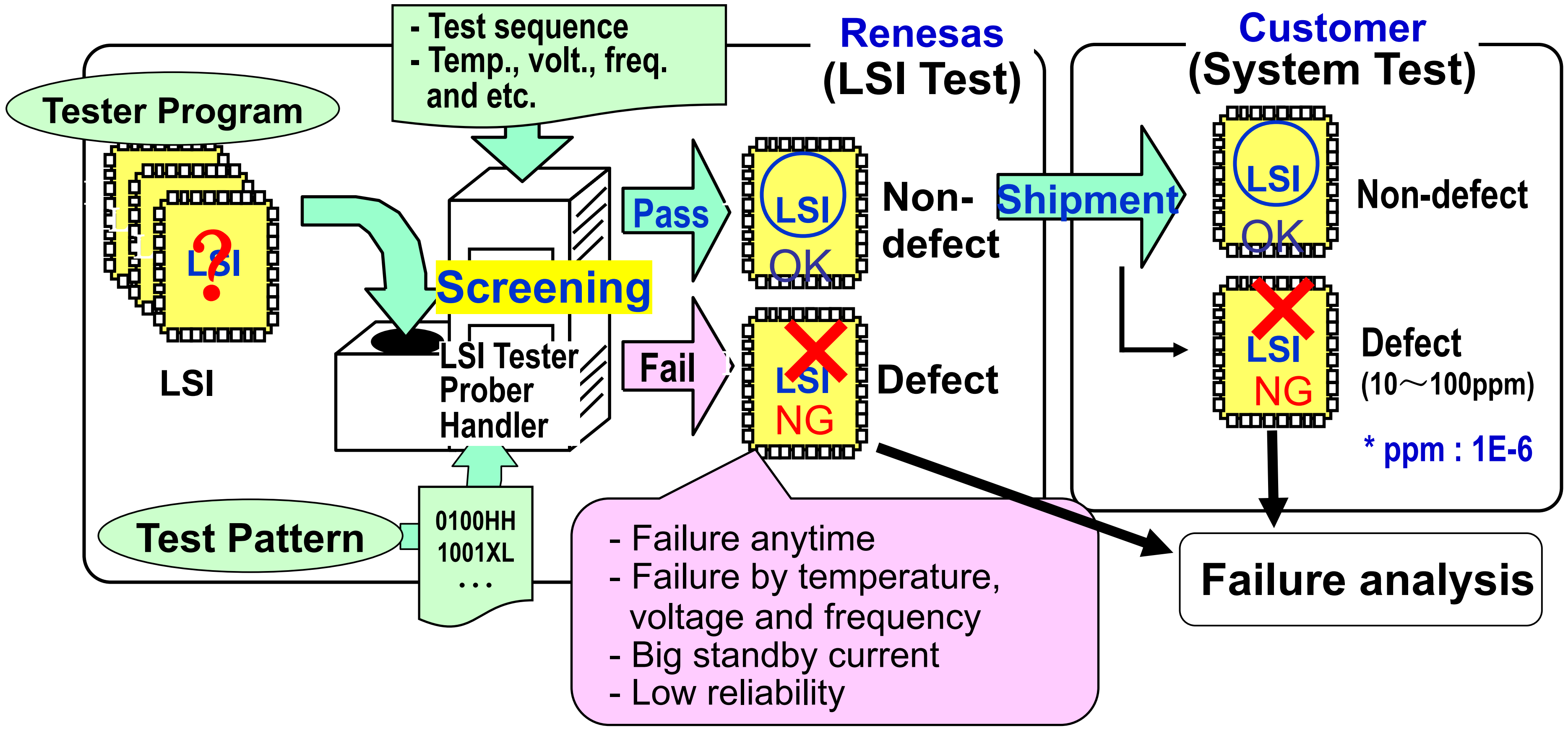
Shortage due to foreign particle



Disconnection due to electron migration

Purpose of Test

- Screening of manufacturing defects
- i.e. "*last defense for LSI quality assurance*"



Test cost is increasing year by year

Purpose of test is as follows

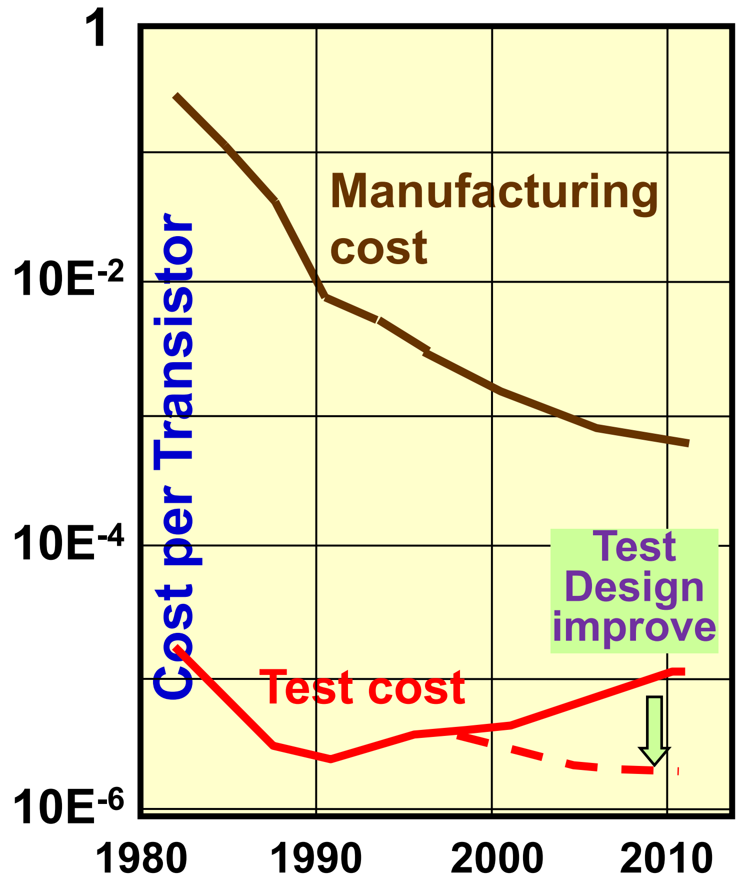
- Maximum quality assurance of LSI
- Adequate test cost

Test Cost (*tester-related cost*)

- Equipment (*tester, prober, handler*)
- Operation (*labor cost*)
  ☞ increase of ratio to manufacturing cost

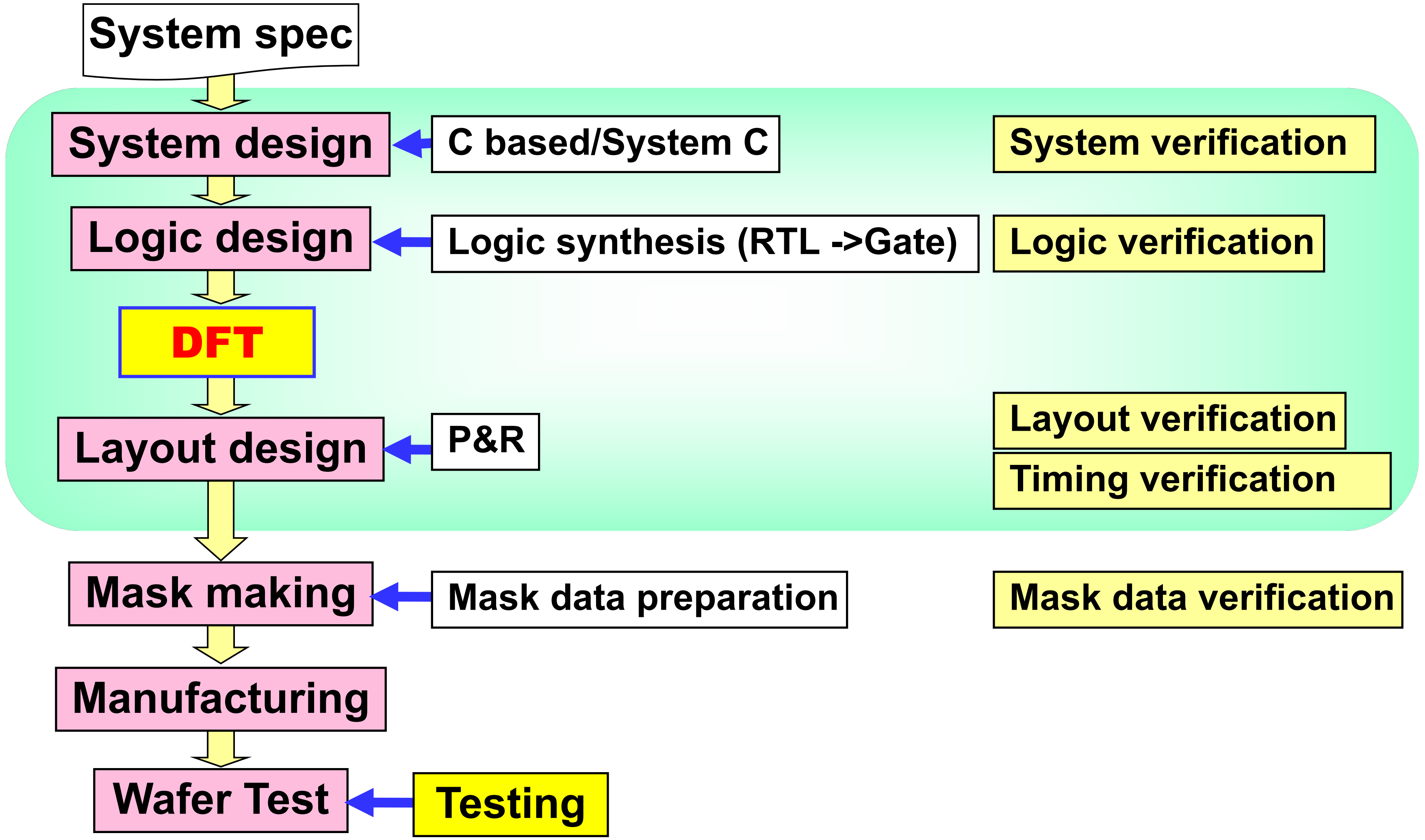
Test Cost (*cost other than tester*)

- Test design cost (*labor cost*)
- Test circuit cost (*chip area*)
- Failure analysis cost (*labor test*)



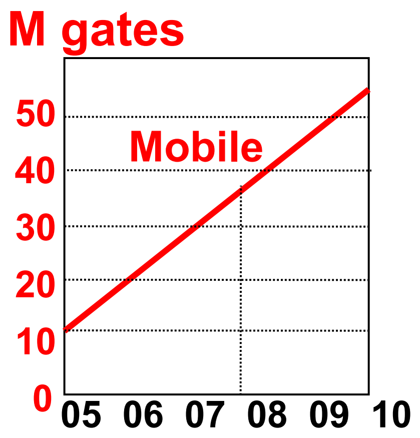
(*Source: ITRS2002 Public Conference*)

LSI Design Flow



Impact of SoC Design Trend

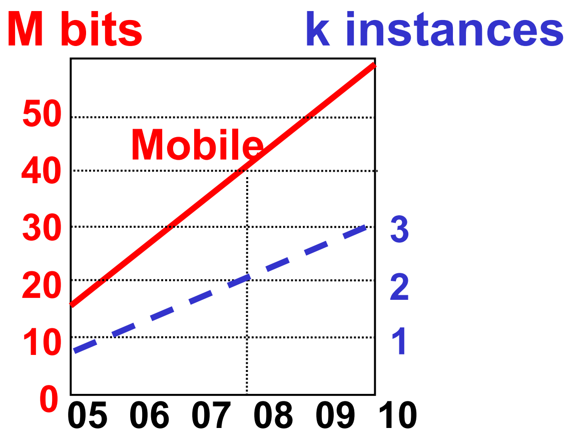
(1) Logic size



> 30% increase a year
-> test time increases
(three-halves power of size)

LBIST or Compression Test

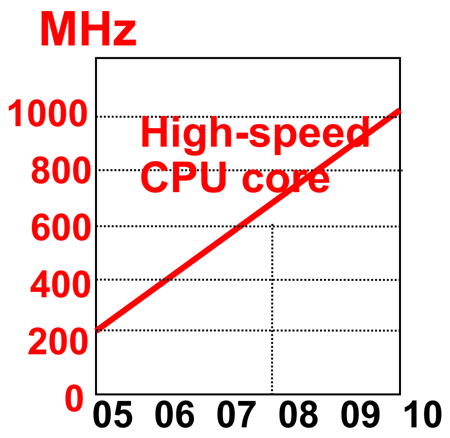
(2) Memory



> 30% increase a year
-> test time increases
(proportion to size)

Concurrent measurement using MBIST

(3) Frequency



> 30% increase a year
-> High-speed tester

Enable to lower external clock freq. using PLL (around 30 MHz)

DFT reduces the increased test time and needs no tester of high specification

Test Methodology

	Functional Test	Structural Test
Concept	Actual LSI operation = Pay attention to Function Check whether LSI functions in the same conditions to the actual condition in use	Pay attention to defects during manufacturing and resulting faults Cover all faults by **modeling faults** completely
Pass/Fail Decision	Decide by operating functionally or not	Decide the existing of defects during manufacturing or not
Coverage Measure	Stuck-at fault + critical path check by manual	Evaluating coverage quantitatively for each fault models by tools
Total coverage capability	Difficult to evaluate quantitatively (no fault modeling => no definition of coverage)	Easy to evaluate quantitatively (Summary for each fault models)
Test quality	There is no scientific improvement because there is no quantification	Evaluate scientifically by quantification (*important to choose models*)

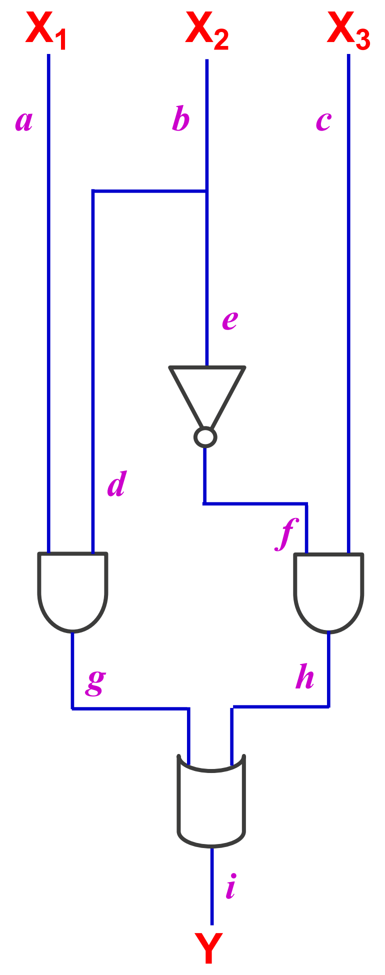
Fault Coverage

Fault Coverage: the percentage of total faults for which test patterns have been generated

$$\boxed{Fault\ Coverage = \frac{Number\ of\ Detected\ Faults}{Total\ Number\ of\ Faults\ in\ the\ Circuit} \times 100}$$

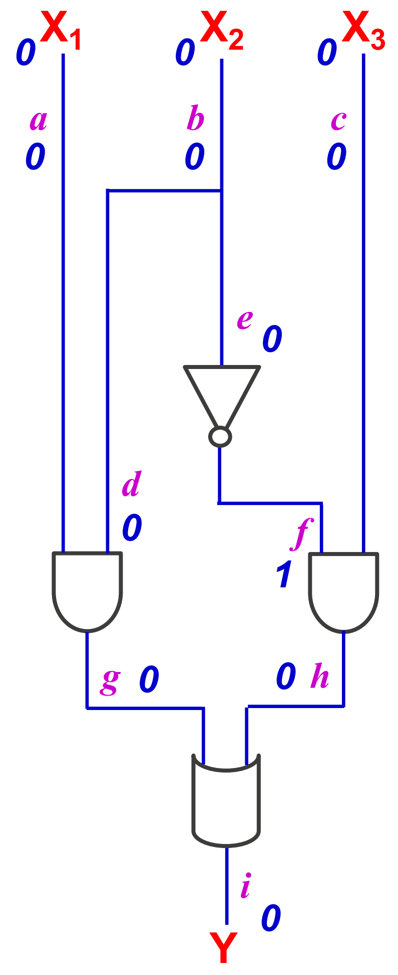
Fault Coverage is also called **Failure Detection Rate** and can be estimated by using **Fault Models**

Modeling Faults Example: *Stack-at-Fault Model*



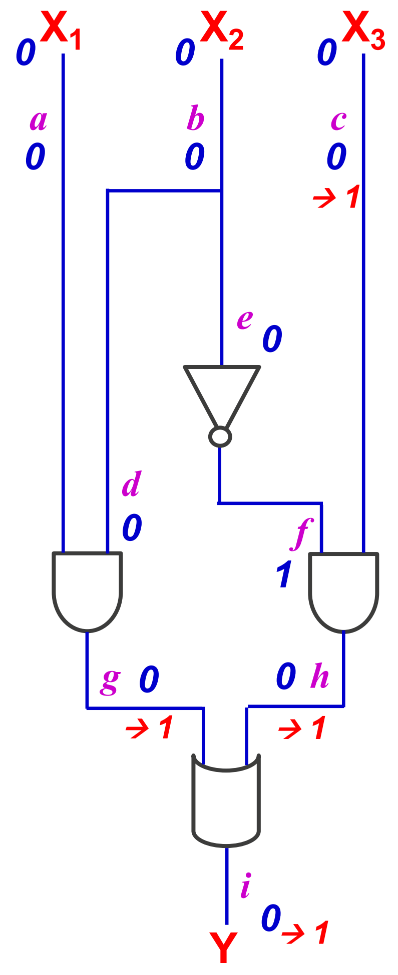
$X_1\ X_2\ X_3$		000	001	010	011	100	101	110	111
Y		0	1	0	0	0	1	1	1
a	SA0								
	SA1								
b	SA0								
	SA1								
c	SA0								
	SA1								
d	SA0								
	SA1								
e	SA0								
	SA1								
f	SA0								
	SA1								
g	SA0								
	SA1								
h	SA0								
	SA1								
i	SA0	0	0	0	0	0	0	0	0
	SA1	1	1	1	1	1	1	1	1

Modeling Faults Example: *Stack-at-Fault Model*



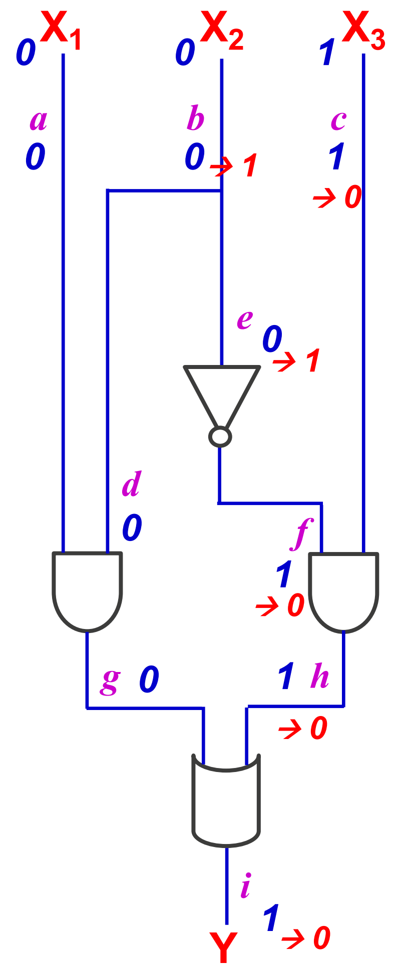
$X_1\ X_2\ X_3$		000	001	010	011	100	101	110	111
Y		0	1	0	0	0	1	1	1
a	SA0	0							
	SA1								
b	SA0	0							
	SA1								
c	SA0	0							
	SA1								
d	SA0	0							
	SA1								
e	SA0	0							
	SA1								
f	SA0								
	SA1	1							
g	SA0	0							
	SA1								
h	SA0	0							
	SA1								
i	SA0	0							
	SA1								

Modeling Faults Example: *Stack-at-Fault Model*



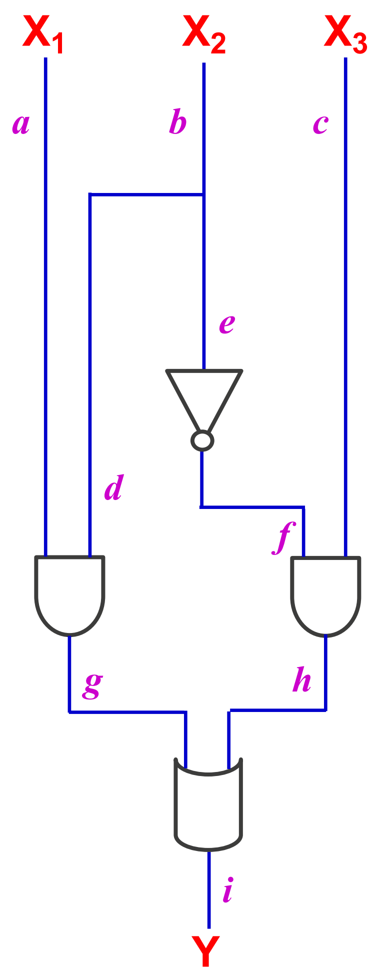
X_1 X_2 X_3		000	001	010	011	100	101	110	111
Y		0	1	0	0	0	1	1	1
a	SA0	0							
	SA1								
b	SA0	0							
	SA1								
c	SA0	0							
	SA1	1							
d	SA0	0							
	SA1								
e	SA0	0							
	SA1								
f	SA0								
	SA1	1							
g	SA0	0							
	SA1	1							
h	SA0	0							
	SA1	1							
i	SA0	0							
	SA1	1							

Modeling Faults Example: *Stack-at-Fault Model*



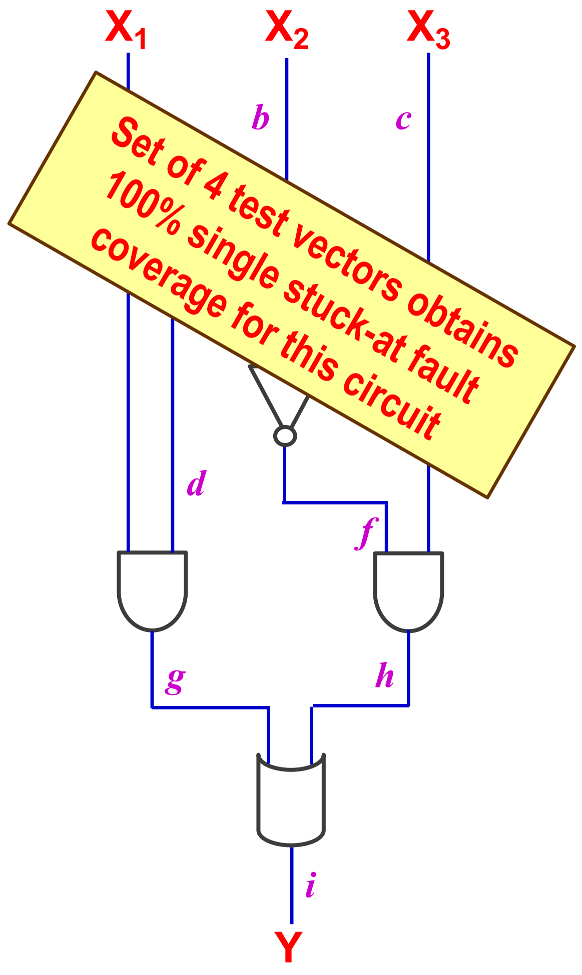
X_1 X_2 X_3		000	001	010	011	100	101	110	111
Y		0	1	0	0	0	1	1	1
a	SA0	0	0						
	SA1								
b	SA0	0	0						
	SA1		1						
c	SA0	0	0						
	SA1	1	1						
d	SA0	0	0						
	SA1								
e	SA0	0	0						
	SA1		1						
f	SA0		0						
	SA1	1	1						
g	SA0	0	0						
	SA1	1							
h	SA0	0	0						
	SA1	1	1						
i	SA0	0	0						
	SA1	1	1						

Modeling Faults Example: *Stack-at-Fault Model*



X_1 X_2 X_3		000	001	010	011	100	101	110	111
Y		0	1	0	0	0	1	1	1
a	SA0	0	0	0	0			**0**	**0**
	SA1			**1**	**1**	1	1	1	1
b	SA0	0	0		**0**	0	0	**0**	**0**
	SA1		**1**	1	1	**1**	**1**	1	1
c	SA0	0	**0**	0		0	**0**	0	
	SA1	**1**	1		1	**1**	1		1
d	SA0	0	0			0	0	**0**	**0**
	SA1			1	1	**1**		1	1
e	SA0	0	0		**0**	0	0		
	SA1		**1**	1	1		**1**	1	1
f	SA0		**0**	0	0		**0**	0	0
	SA1	1	1		**1**	1	1		
g	SA0	0	0	0	0	0	0	**0**	**0**
	SA1	**1**		**1**	**1**	**1**		1	1
h	SA0	0	**0**	0	0	0	**0**	0	0
	SA1	**1**	1	**1**	**1**	**1**	1		
i	SA0	0	**0**	0	0	0	**0**	**0**	**0**
	SA1	**1**	1	**1**	**1**	**1**	1	1	1

Modeling Faults Example: *Stack-at-Fault Model*



Set of 4 test vectors obtains 100% single stuck-at fault coverage for this circuit

$X_1\ X_2\ X_3$		000	001	010	011	100	101	110	111
Y		0	1	0	0	0	1	1	1
a	SA0	0	0	0	0			**0**	**0**
	SA1			**1**	**1**	1	1	1	1
b	SA0	0	0		**0**	0	0	**0**	**0**
	SA1		**1**	1	1	**1**	**1**	1	1
c	SA0	0	**0**	0		0	**0**	0	
	SA1	**1**	1		1	**1**	1		1
d	SA0	0	0			0	0	**0**	**0**
	SA1			1	1	**1**		1	1
e	SA0	0	0		**0**	0	0		
	SA1		**1**	1	1		**1**	1	1
f	SA0		**0**	0	0		**0**	0	0
	SA1	1	1		**1**	1	1		
g	SA0	0	0	0	0	0	0	**0**	**0**
	SA1	**1**		**1**	**1**	**1**		1	1
h	SA0	0	**0**	0	0	0	**0**	0	0
	SA1	**1**	1	**1**	**1**	**1**	1		
i	SA0	0	**0**	0	0	0	**0**	**0**	**0**
	SA1	**1**	1	**1**	**1**	**1**	1	1	1

5.2 DFT and Scan Method

What is DFT?

DFT is a design technique that adds test circuits inside the LSI hardware. These added circuits permits to easily apply tests on the designed LSI after manufacturing

DFT consists of

- Creating test circuits
- Inserting test circuits into LSI hardware
- Preparing test data for physical failure analysis

Purpose of DFT

- Guarantee the LSI product quality
- Improve manufacture process
- Optimize test cost

Target of DFT

- Minimize area overhead
- Increase test/fault coverage
- Minimize the test time
- Minimize the test cost

Required Quality and Test Level

Required quality (quality level of products)

- Zero defect
- Guarantee of operation under warranty condition

Test level

- Test items
- Test conditions
  (*temperature, voltage, frequency/timing, etc.*)
- Test model (*measure representing test sufficiency*)
  and target fault coverage (*quantitative evaluation of test sufficiency*)

Target Quality of Renesas Products

Standardization of common quality level requested by Consumer

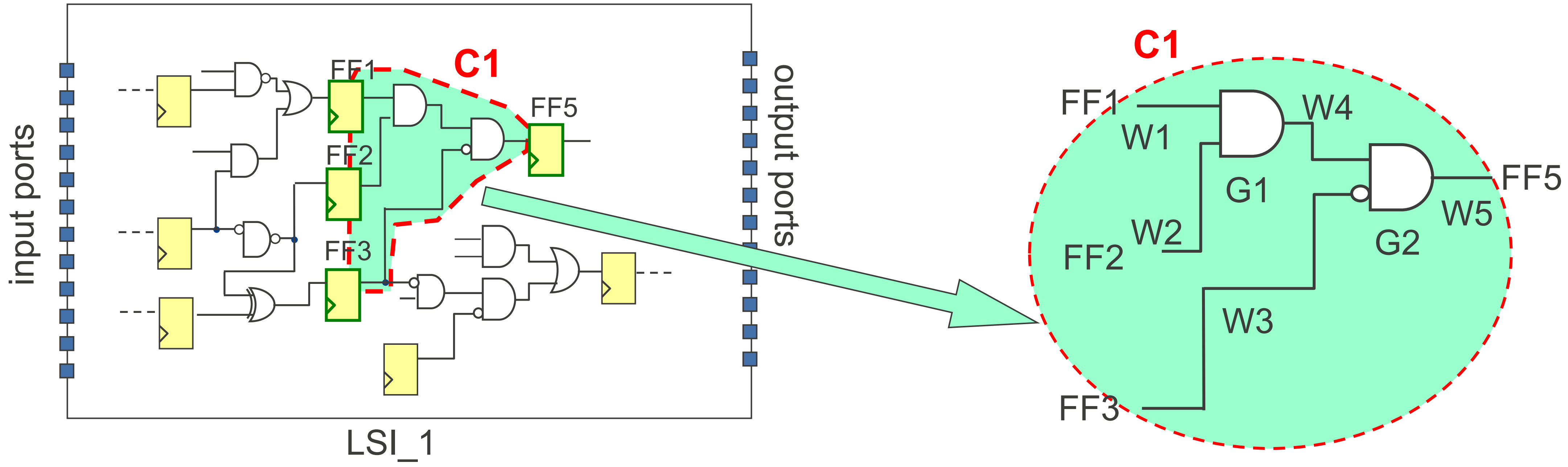
Quality Classification		Guaranteed Lifetime	Main products
High Reliability	Q1A	20 years	Automobile parts (*Engine controller*, *etc.*) General traffic equipment
	Q1B	10 years	Car electronics (*accessories*: *genuine goods*), etc.
Industry	Q2	10 years	Car electronics, FA equipment, etc.
Consumer	Q3	10 years	PC, Consumer electronics, Mobile equipment
Custom	QX	Determine individually for each product	Game instruments, Ultra high reliable equipment

Structural Test

To check if the C1 part of an LSI shown below is manufactured without defect, we have to check wires W1, W2, W3, W4 and W5, and gates G1 and G2 are manufactured without defect.

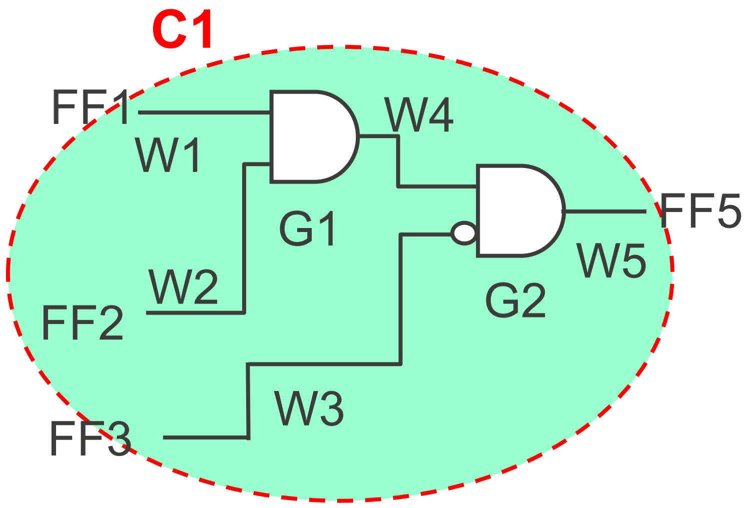
Usually logic designers do not know what kind of gates are created on which wire connections. Therefore, it is almost impossible to set a specific value at the sending end of a wire such as W4 and checking the voltage at the other end of the wire by applying test data input at the input ports of the LSI.

This means functional test is not suitable for detecting manufacturing defects.



However, knowing the gate net list, we can tell what value must come out at W5 output when we apply the values listed on the table below to FF1, FF2, and FF3.

By checking whether W5 output is equal to the value shown in the table, we can detect manufacturing defects in C1.



Applicable input pattern and expected output

pattern	FF1	FF2	FF3	W5 output
A	1	1	1	0
B			0	1
C		0	1	0
D			0	0
E	0	1	1	0
F			0	0
G		0	1	0
H			0	0

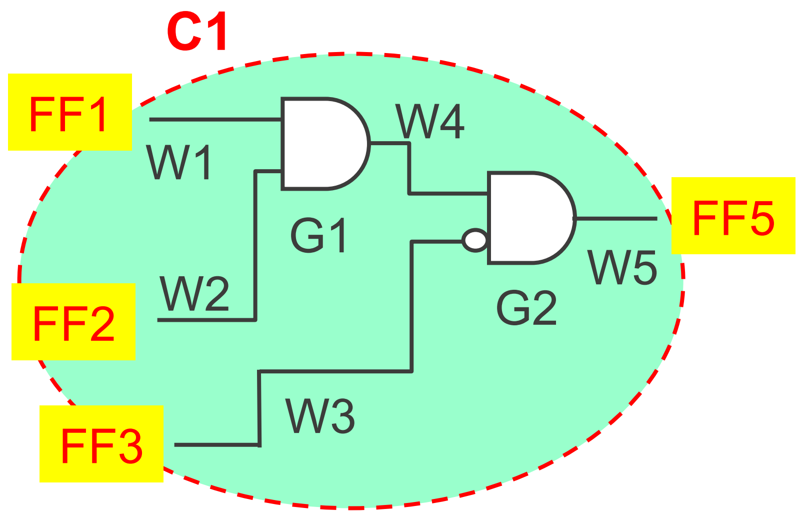
Patterns E and G are not needed because they can be covered by pattern C.

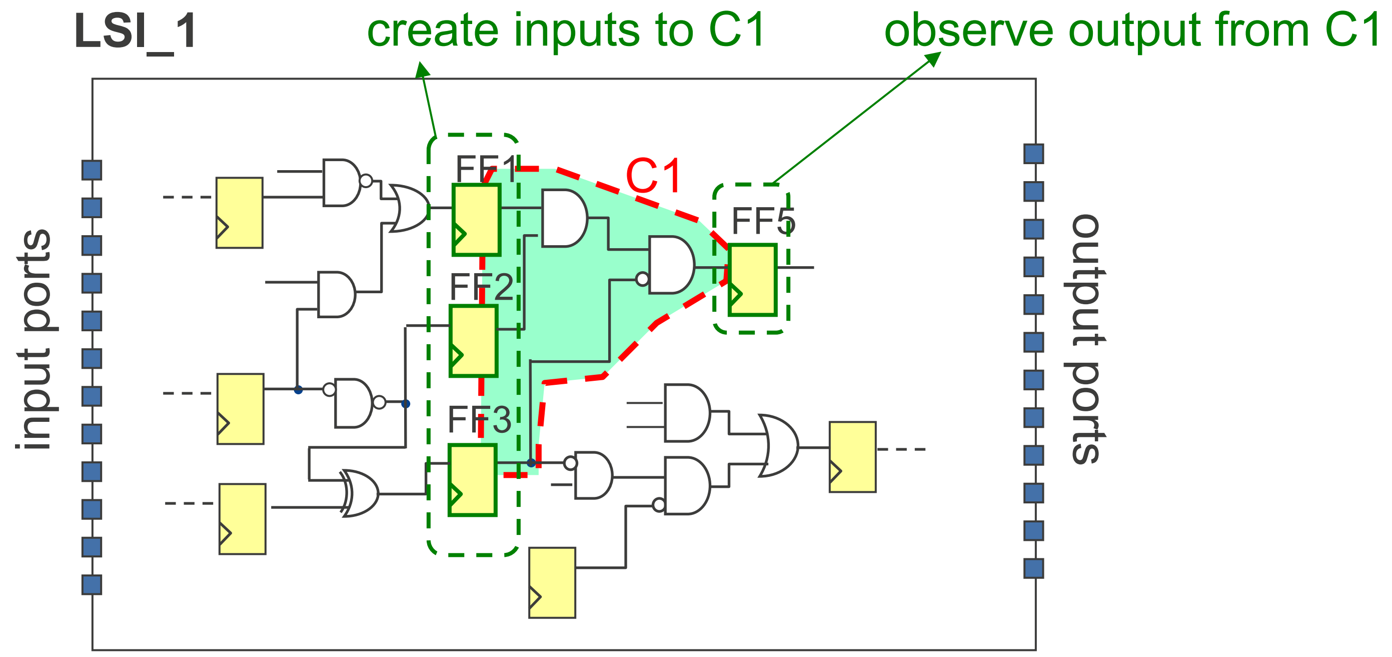
A, B, C, D, F, and H are enough to check all the logic elements.

Scan Method

For the logic block shown on the right, we can calculate what must come out at W5 output when we apply some input patterns on W1, W2, and W3, based on the net list information.

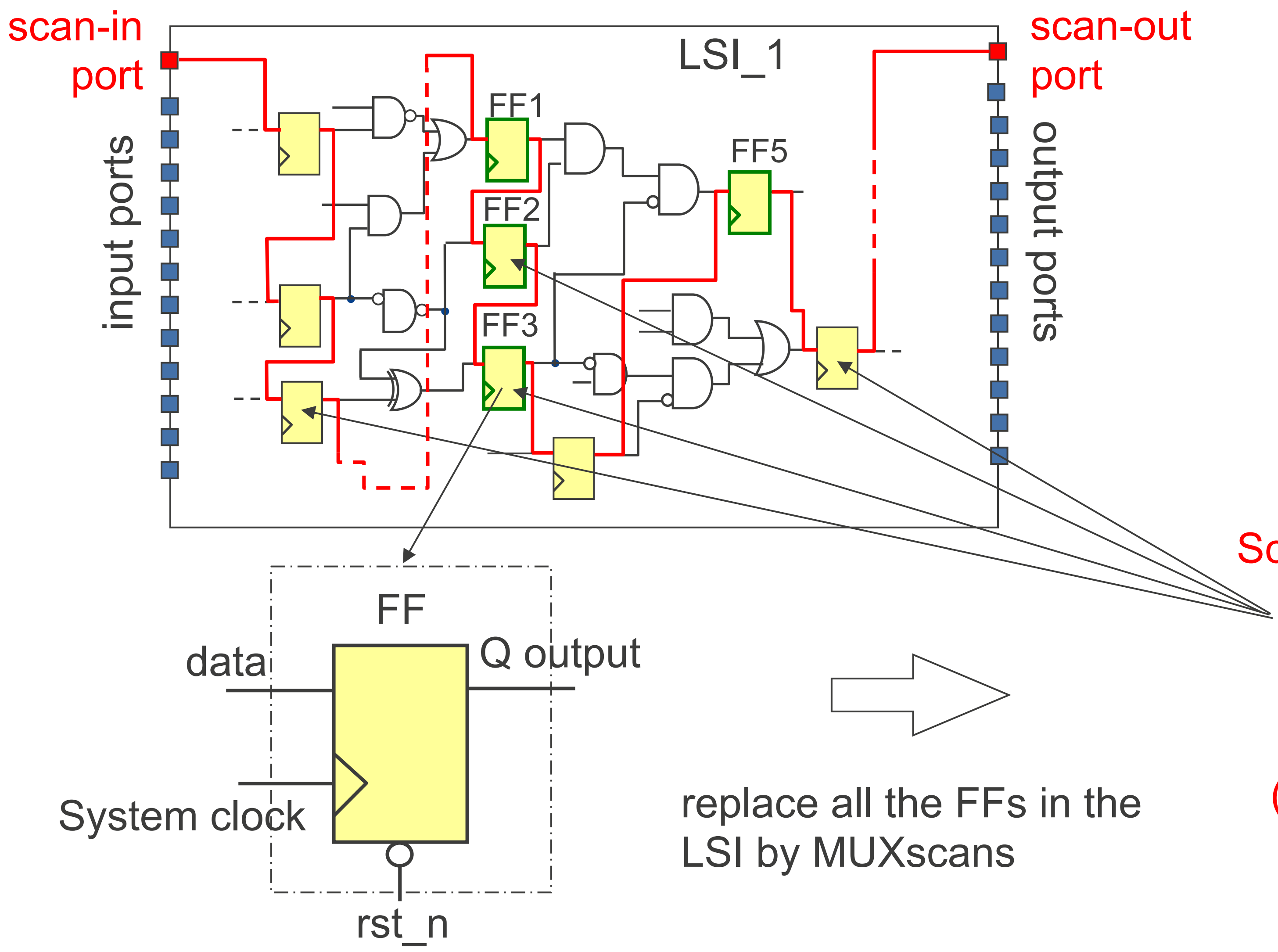
Scan is a method to use FF or latch to set input patterns on W1, W2, and W3 and observe (memorize) the output on W5.





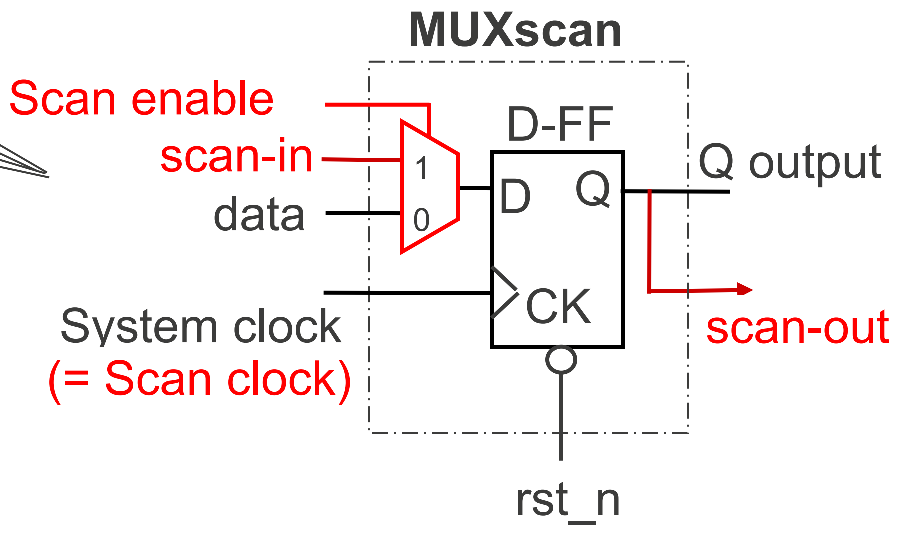
MUXscan is an idea to use FF1, FF2, and FF3, created from the RTL code by a synthesis tool, to set values on W1, W2, and W3 respectively and use FF5 to observe output W5.

To apply MUXscan we have to replace all FFs in the LSI by "DFT ready FF", specially designed FF suitable for MUXscan.



DFT tools automatically connects all the replaced FFs by a scan chain.

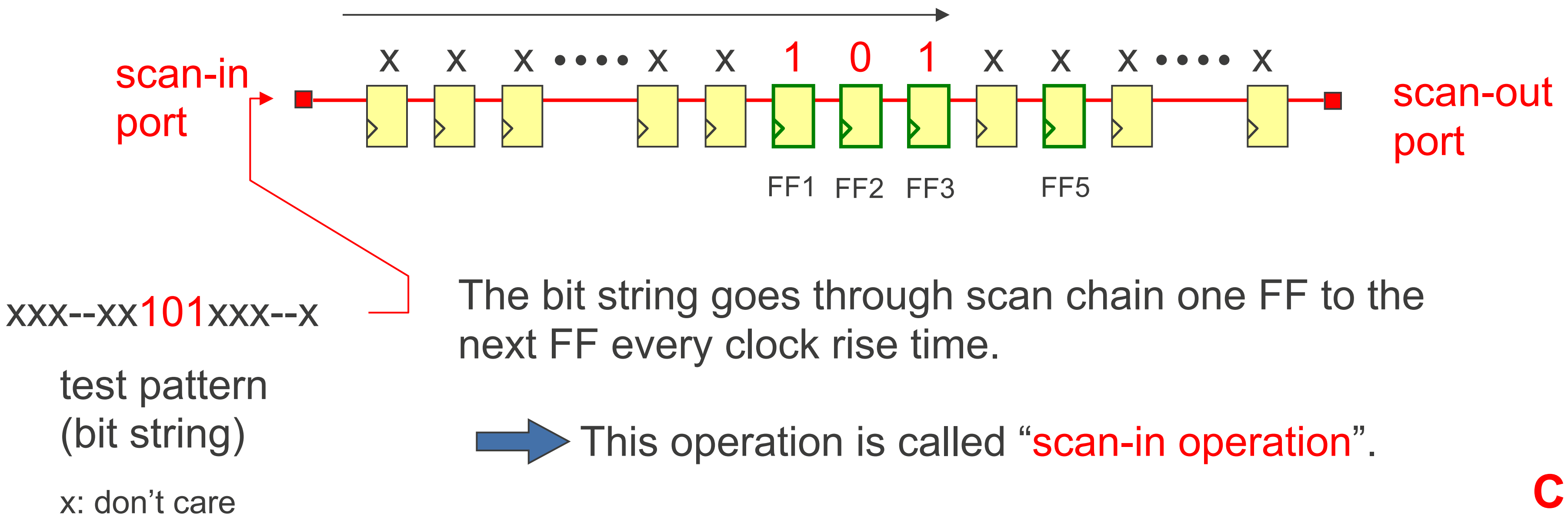
Replacement is done by DFT tools automatically.



replace all the FFs in the LSI by MUXscans

Scan chain created by a DFT tool works as shown below:

(1) If we prepare bit string like xxx--x101xx--xxxx at scan in port and apply clock signal to FFs, then the bit string goes into FFs through the scan chain and as a result 1, 0, and 1 are set to FF1, FF2, and FF3 respectively.

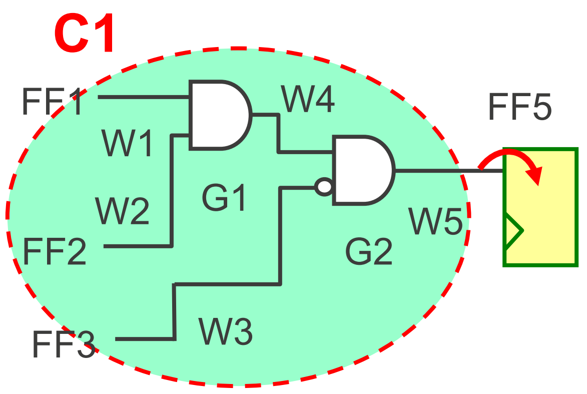


The bit string goes through scan chain one FF to the next FF every clock rise time.

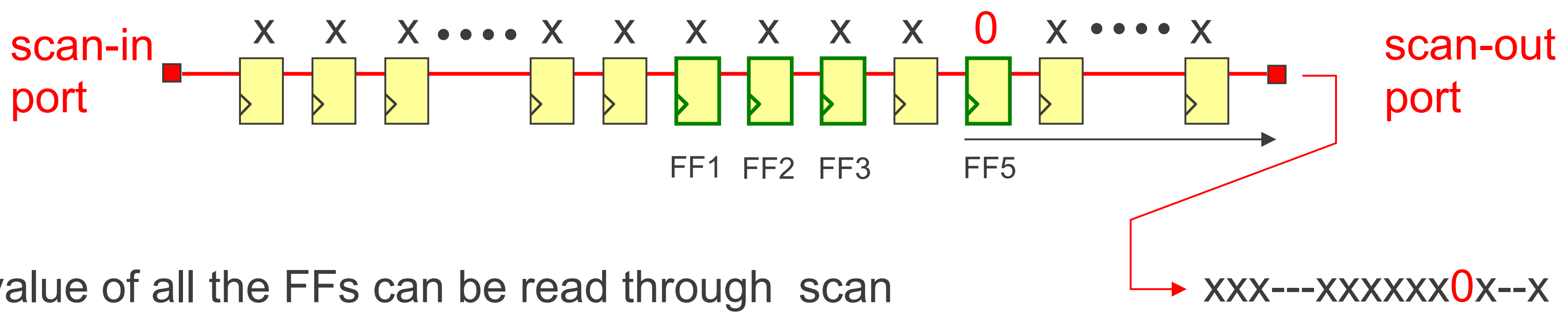
➡ This operation is called "scan-in operation".

(2) After setting 1, 0, and 1 on FF1, FF2, and FF3 respectively, operate the LSI for 1 clock cycle to get output W5 into FF5.

➡ This operation is called "capture operation".



(3) After capture operation, the value of FF5 can be got from the scan out port through scan chain as shown below.



x: don’t care

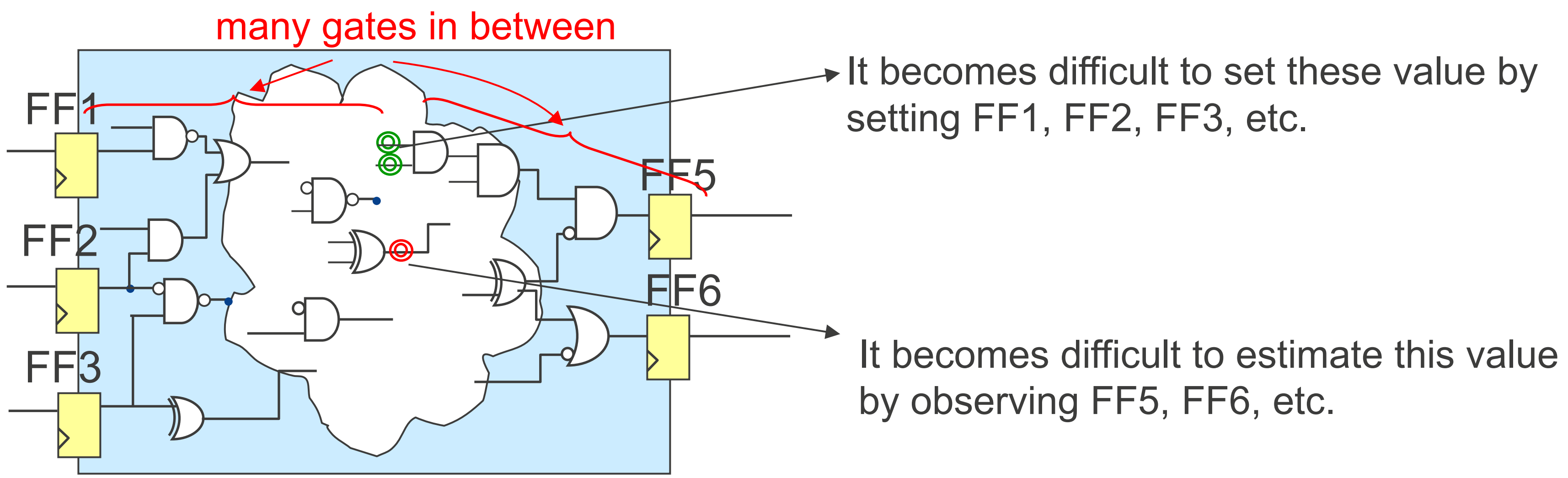
The value of all the FFs can be read through scan chain. They will come out every clock rise time.

This operation is called “scan out operation”.

(4) The value of FF5 came through the scan chain is compared to the expected result of C1. If it matches with the expected results for all the test pattern, then C1 is manufactured without defects.

Test Point Insertion (TPI)

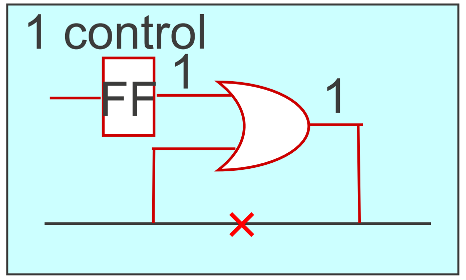
Because a DFT tool knows the gate net list, it can create data pattern which can test logic blocks in the LSI. However, if the logic scale between FFs is very large, then it is difficult to create patterns to check-out all the defects in that logic block.

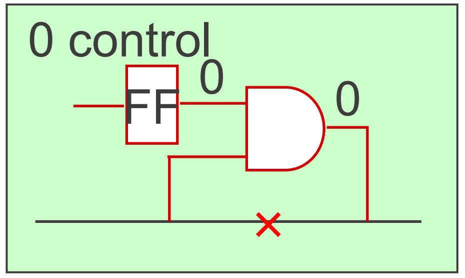


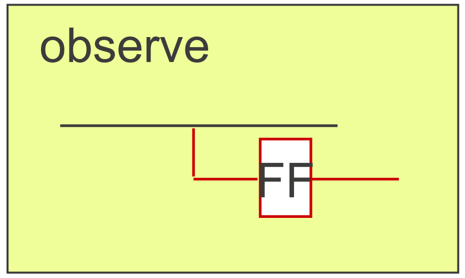
It becomes difficult to set these value by setting FF1, FF2, FF3, etc.

It becomes difficult to estimate this value by observing FF5, FF6, etc.

Inserting the following logics in proper signal line, setting and observing signals become very easy. This test point insertion improves coverage of test pattern generated by random pattern generator.

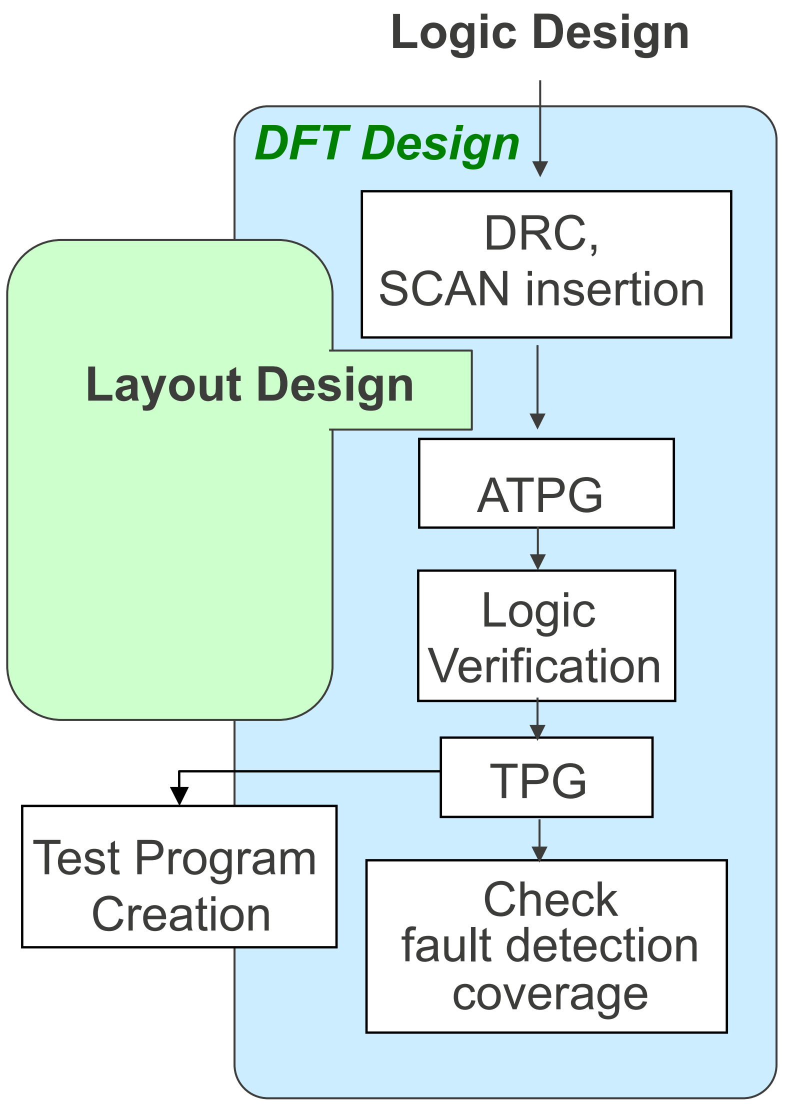






5.3 DFT Design Flow

DFT Design Flow



Insertion of Scan and BIST logic/circuits
(DFTAdviser/DFTCompiler, "Shingen", MINORI)

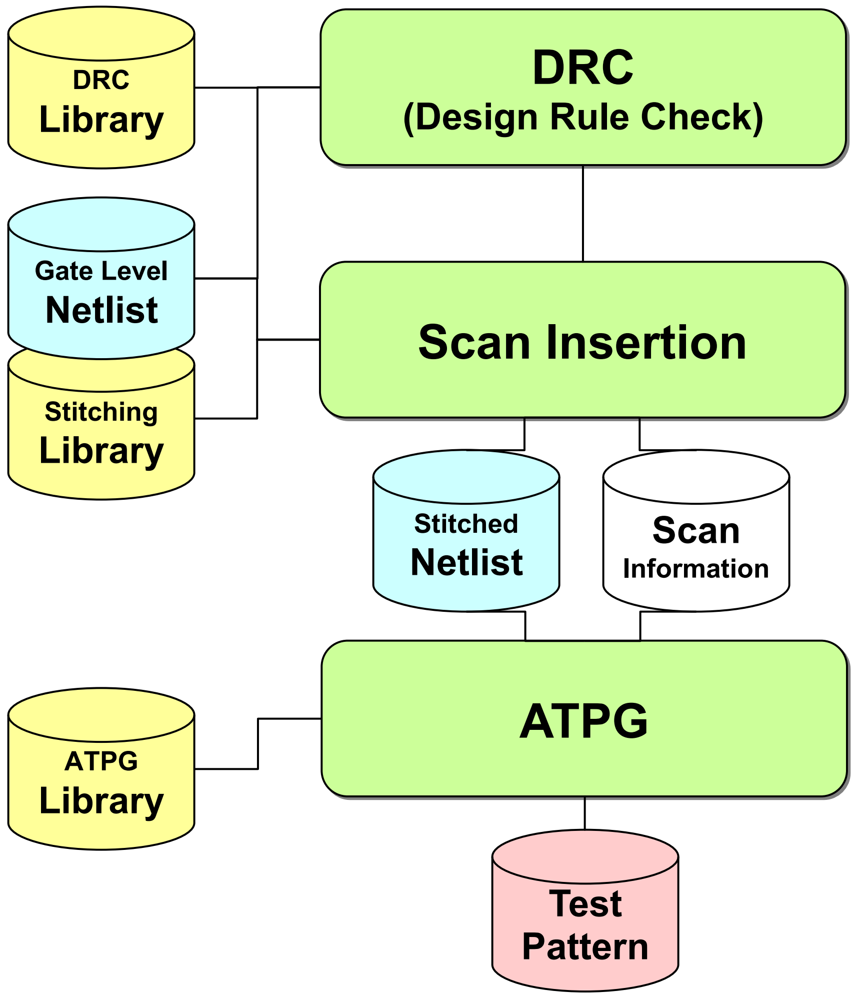
Automatic Test Pattern Generation
(TetraMAX, FastScan, "Shingen", MINORI)

Verification including inserted test circuits
(Verilog_XL/NC-Verilog/Conformal-LEC)

Test Program (for tester) Generation

Check fault detection rate of the test program in a tester
(Fault simulator: Z01X!, Verifault-XL)

SCAN Design



DRC (Design Rule Check)

Verify a design satisfies design constraints for MUX scan method

Scan Insertion

Construct scan chain

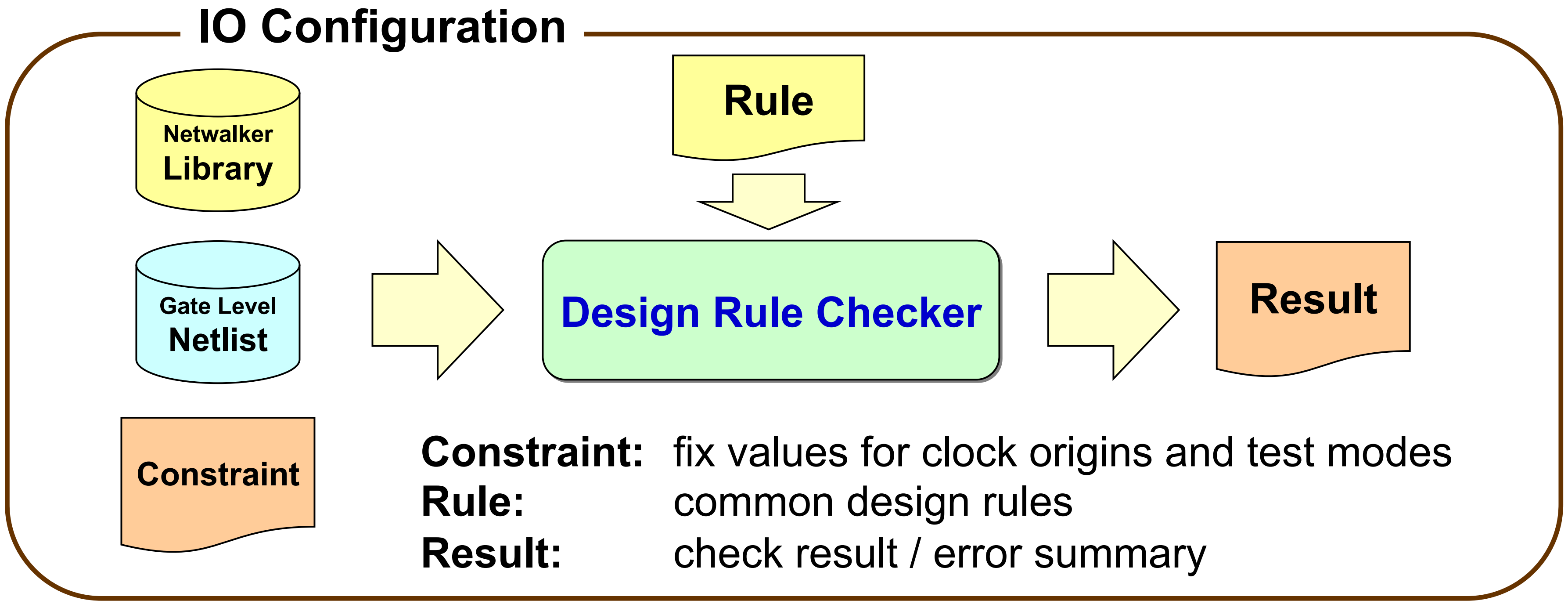
Compression and decompression circuits are also inserted in the use of scan compression

ATPG (Test pattern generation)

Generate test pattern

DRC (Design Rule Check)

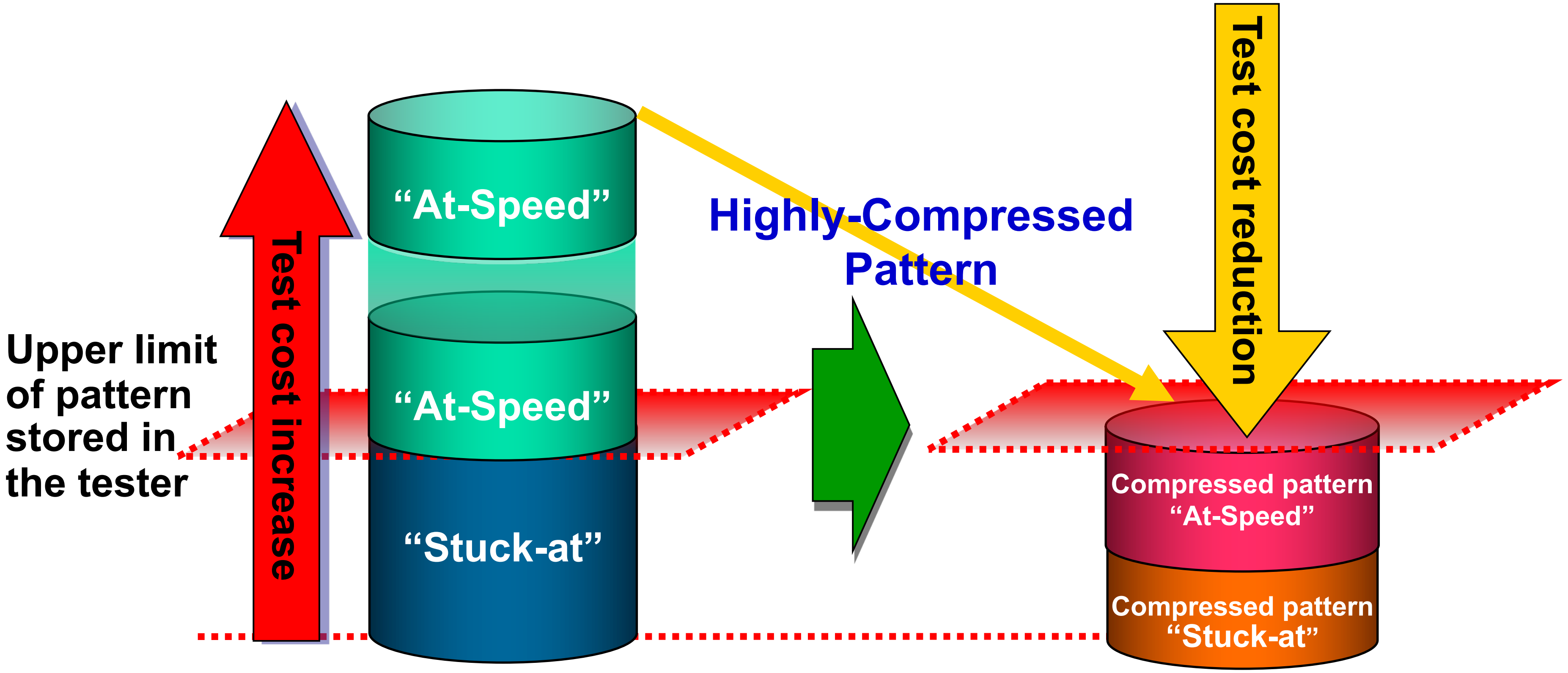
- A tool which can verify design constraints for MUX scan method is satisfied
- Necessary constraints for MUX scan are covered
- Easy error analysis by detail error report and debugging functionality



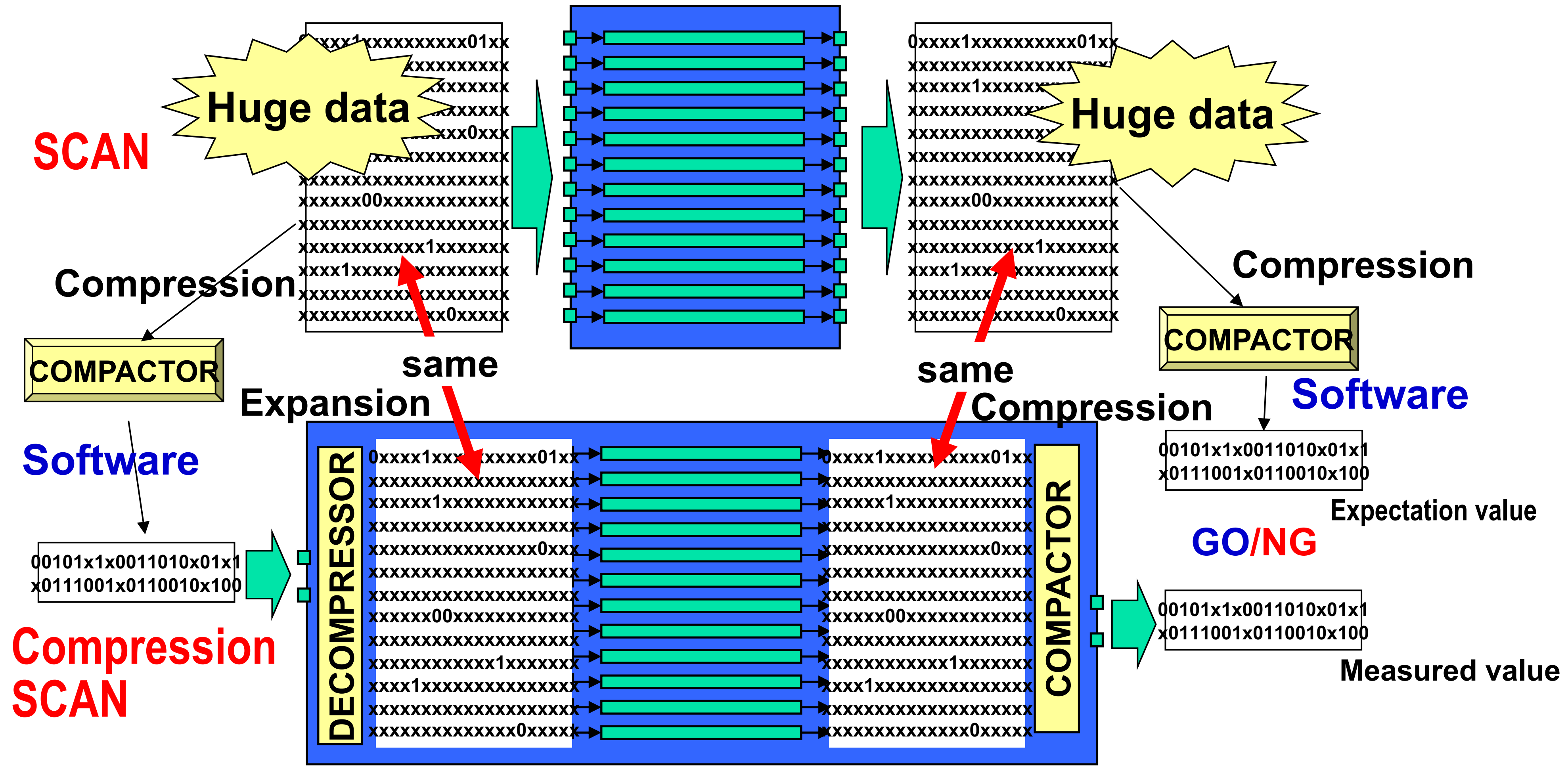
Constraint:	fix values for clock origins and test modes
Rule:	common design rules
Result:	check result / error summary

Necessity for Highly-Compressed Pattern

- Need more pattern generated for AC SCAN test (3 to 5 times)
- Drastically Highly-Compressed Pattern is necessary to cope with large scale while maintaining the high test quality


Upper limit of pattern stored in the tester
Test cost increase
"At-Speed"
"At-Speed"
"Stuck-at"
Highly-Compressed Pattern
Test cost reduction
Compressed pattern "At-Speed"
Compressed pattern "Stuck-at"

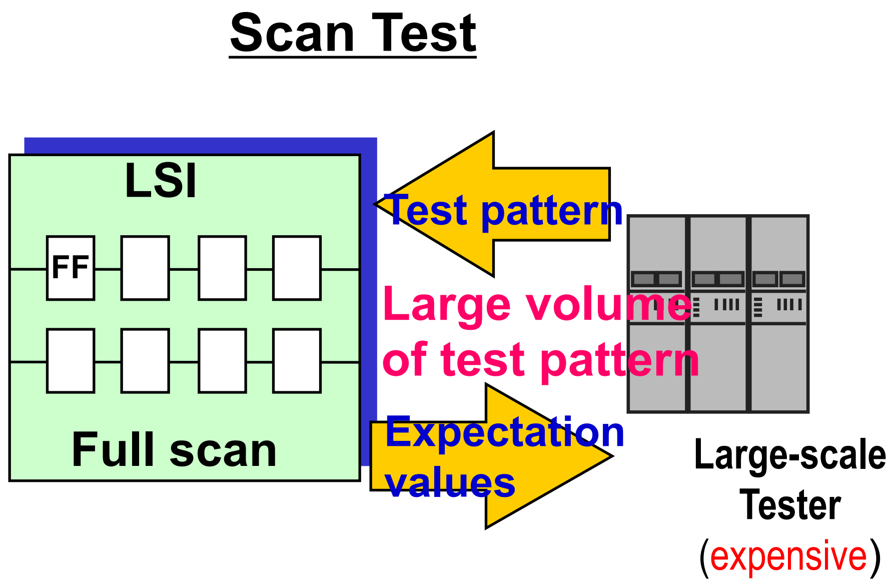
Basic Concept of Compression SCAN

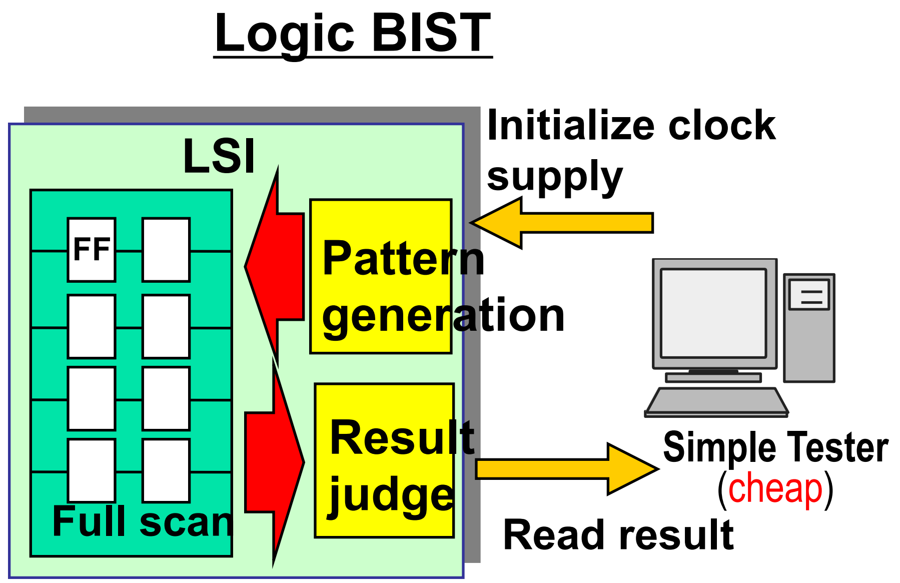


Data compression/expansion circuits (**DECOMPRESSOR/COMPACTOR**) in a LSI for SCAN input/output can reduce pattern data size

Built-In Self-Test (BIST)

- Pattern generation circuits and pass/fail judge circuits are inserted into an LSI
- Can test an LSI using pseudo random number patterns
- Can judge pass/fail after a test completed





Renesas.com