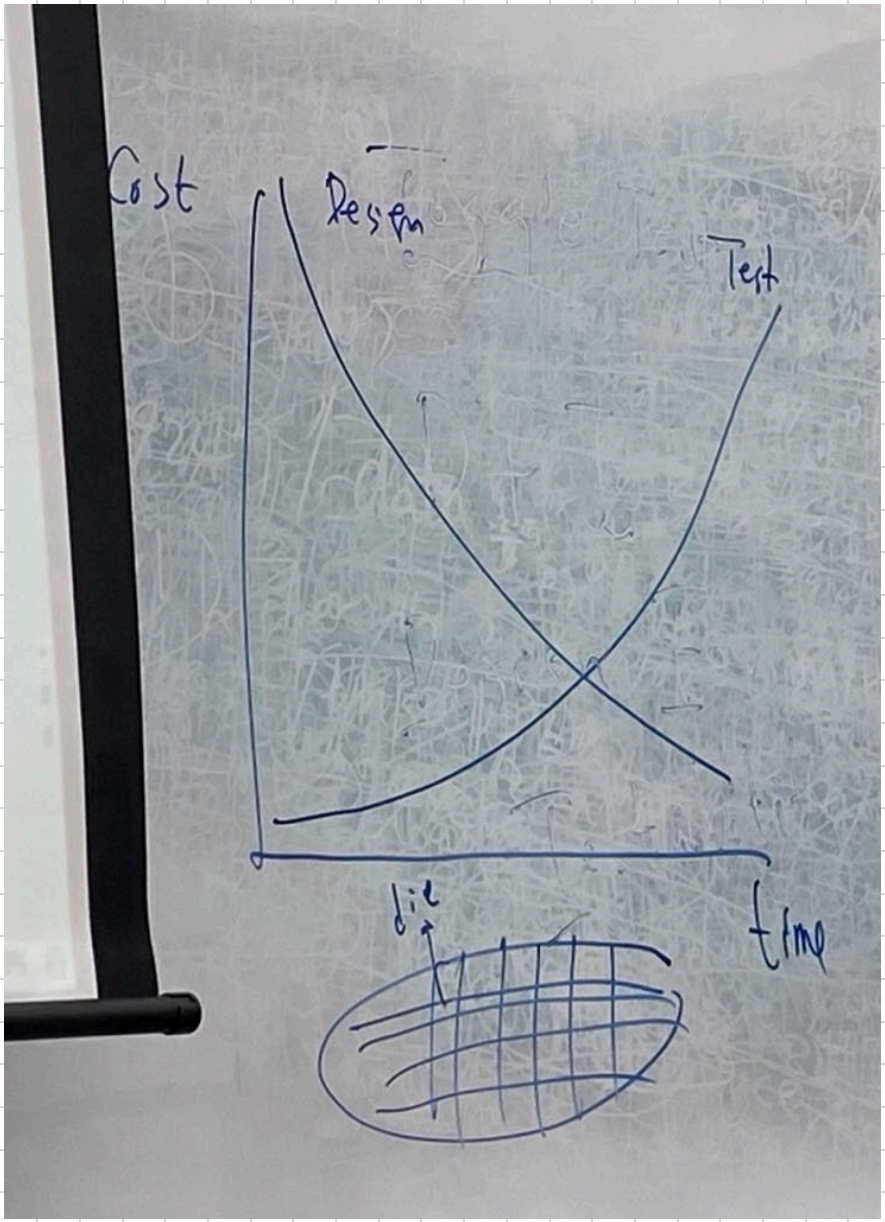
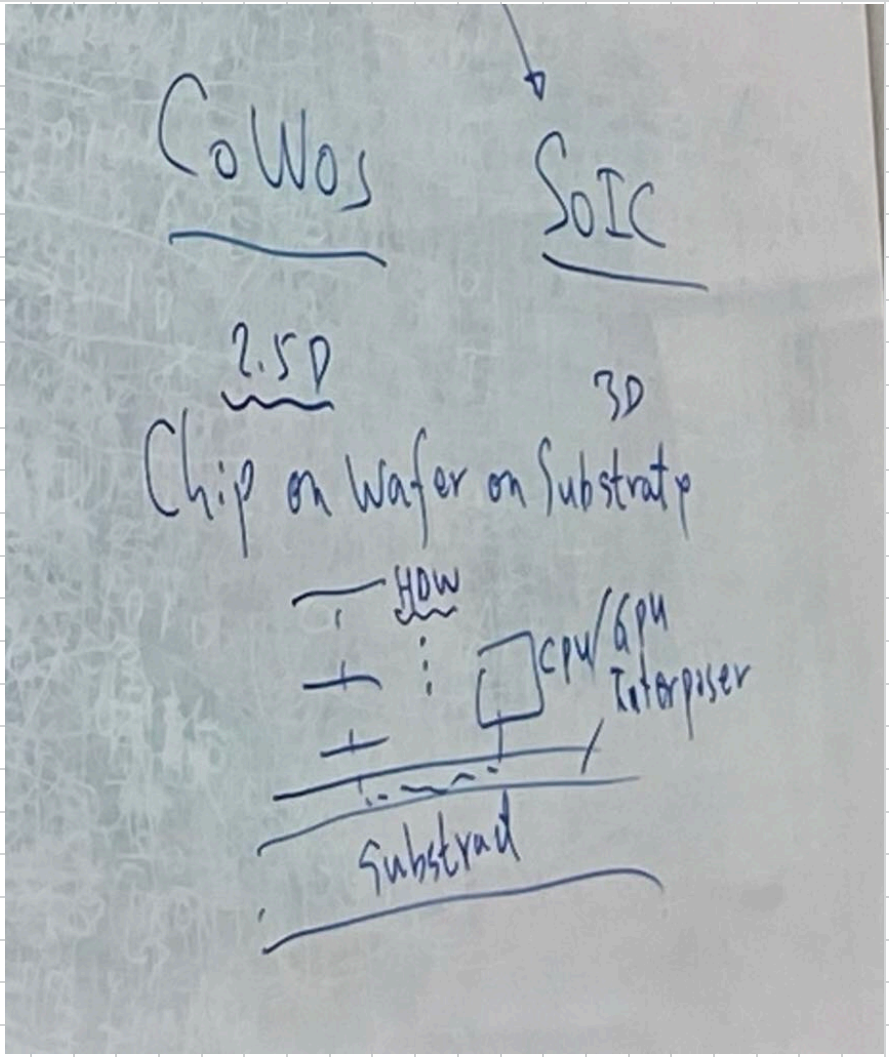
SOC筆記
這堂課目的
希望能夠在很複雜的電路 電版 裡 去測試 零件
因為很難測試
所以測試的零件 在一開始 就要設計進去

隨著時間越來越接近現代
設計 一塊電路 代價越來越低
測試 一塊電路 代價越來越高



這兩個很有名
CoWoS 是用堆疊的電路 2.5D
SoIC 是立體 3D



BIST（built-in self test) 是很重要的單元
Memory Test 也很重要

Verification 判斷有沒有問題
這裡不能用test

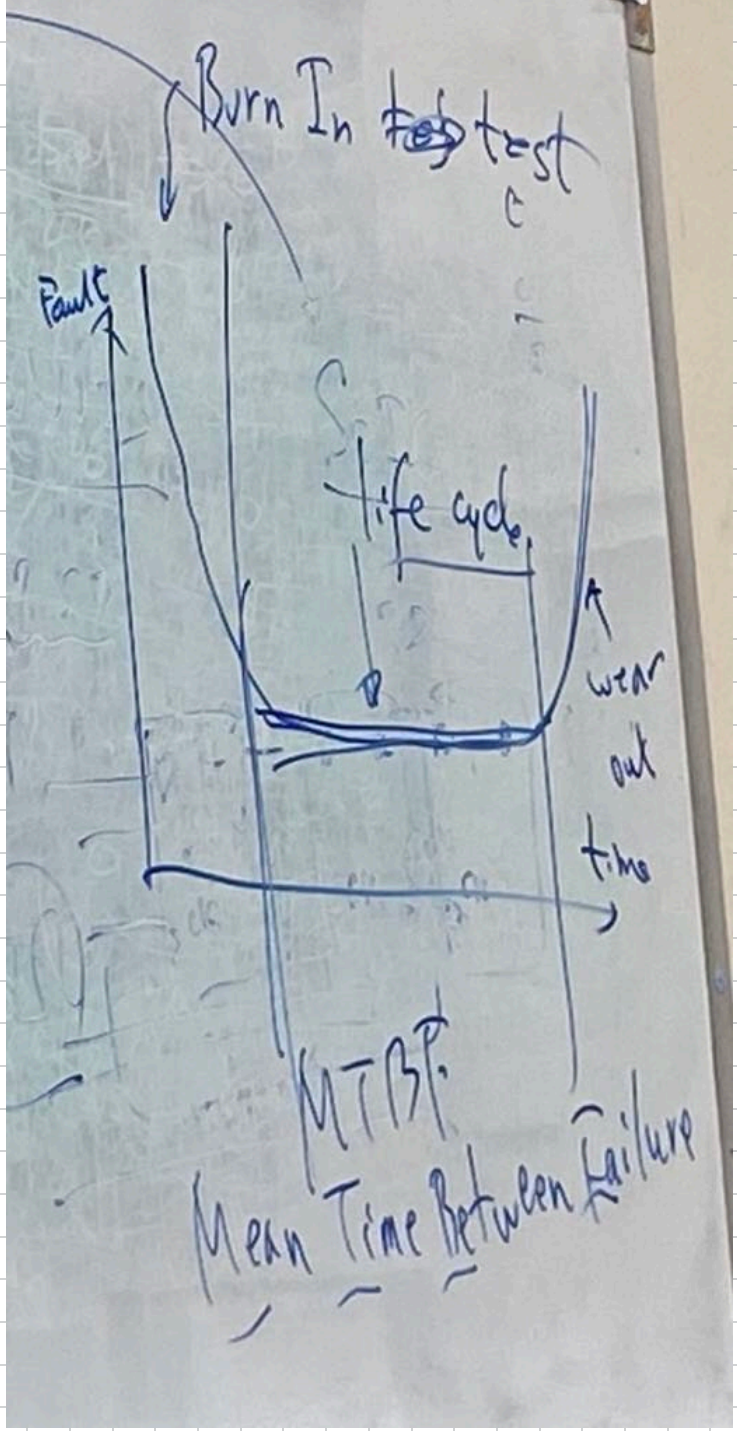
Diagnosis 判斷哪裡有問題
Reliability 可靠度

Verification: To verify the correctness of a design

Diagnosis: To tell the faulty site

Reliability: To tell whether a good system will work after some time.

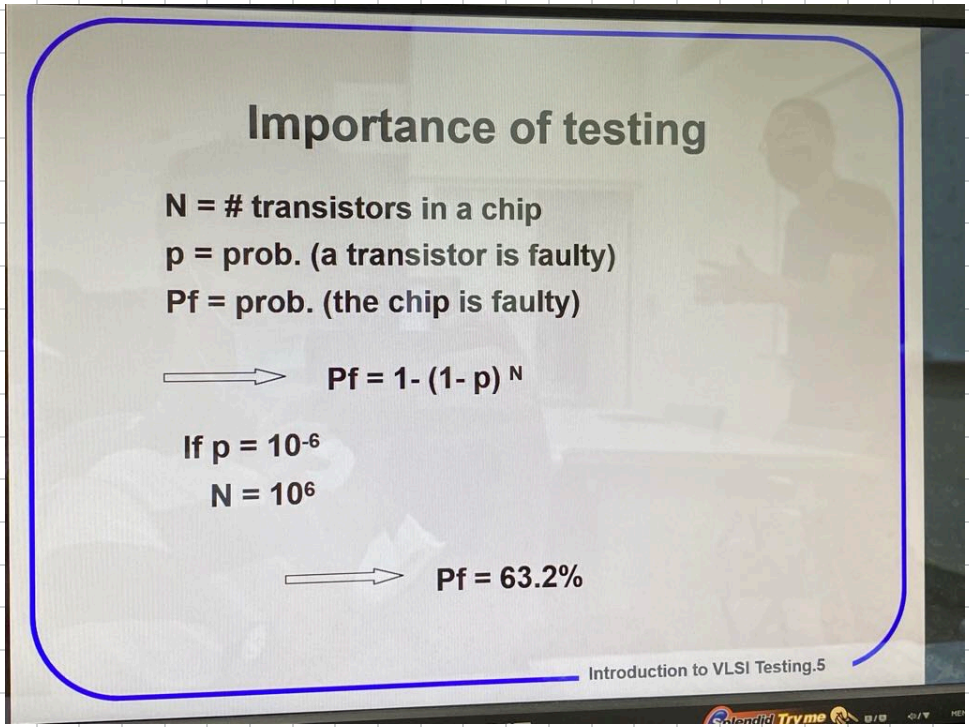
這圖是在說他何時會出錯
Burn in test 熱機測試 一開始就把壞的拿掉一些
Life cycle 生命週期 MTBT
Wear out 老化



N 是 晶片裡面的 電晶體數量
P 是 電晶體 出錯的機率
Pf 是 晶片出錯的機率

1–p 是一個 電晶體 正常的機率

如果不檢查 100個 就有63個出錯
期末盡量報告 這門課相關的 不會上台報告
所以 測試 是很有必要的



Introduction to VLSI Testing

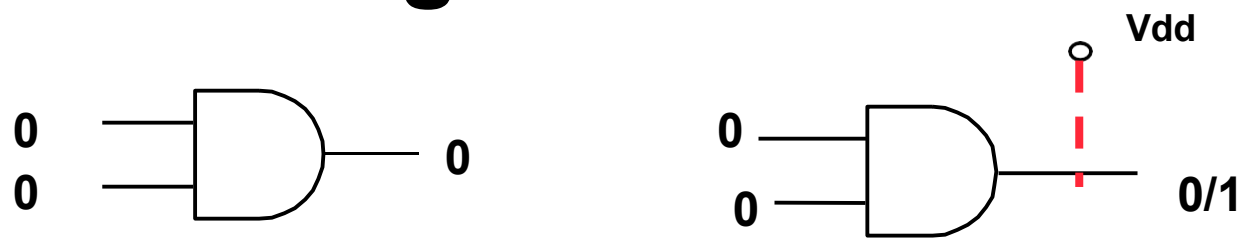
Problems to Think

- A 32 bit adder
- A 32 bit counter with RESET function
- A 1MB cache memory
- A 10^7-transistor CPU

OUTLINE

- Introduction
- Fault modeling
- Fault simulation
- Test generation
- Automatic test pattern generation (ATPG)
- Design for testability (DFT)
- Built-in self test
- Memory Test

Testing: To tell whether a system is good or bad



Related fields

Verification: To verify the correctness of a design

Diagnosis: To tell the faulty site

Reliability: To tell whether a good system will work after some time.

Importance of testing

N = # transistors in a chip

p = prob. (a transistor is faulty)

Pf = prob. (the chip is faulty)

$\Rightarrow$ $Pf = 1 - (1 - p)^N$

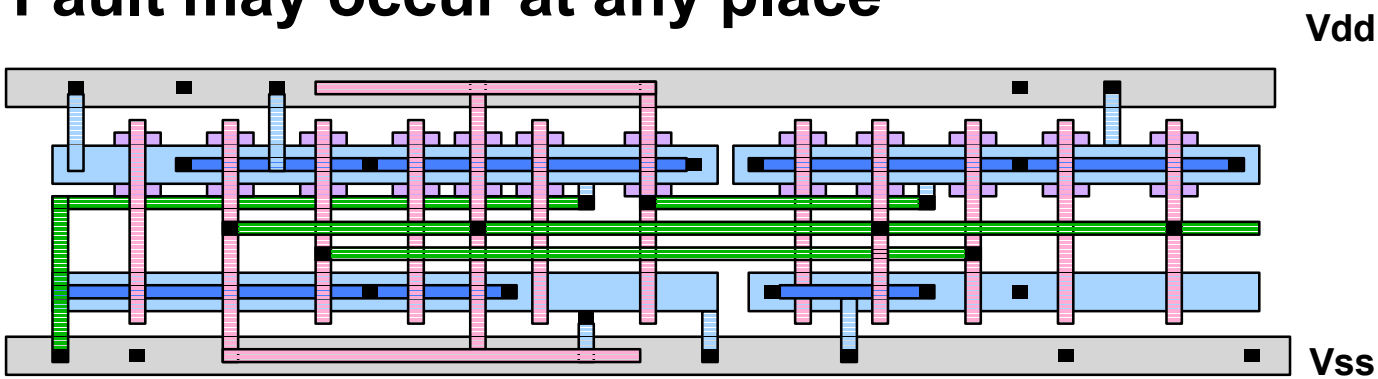
If $p = 10^{-6}$

$N = 10^6$

$\Rightarrow$ $Pf = 63.2\%$

Difficulties in Testing

- **Fault may occur anytime**
  - Design
  - Process
  - Package
  - Field
- **Fault may occur at any place**



- **VLSI circuit are large**
  - Most problems encountered in testing are NP-complete
- **I/O access is limited**

How to do testing from Designer's points of view

- Circuit modeling
- Fault modeling

} Modeling

- Logic simulation
- Fault simulation
- Test generation

這裡是電腦做的 因為太多了

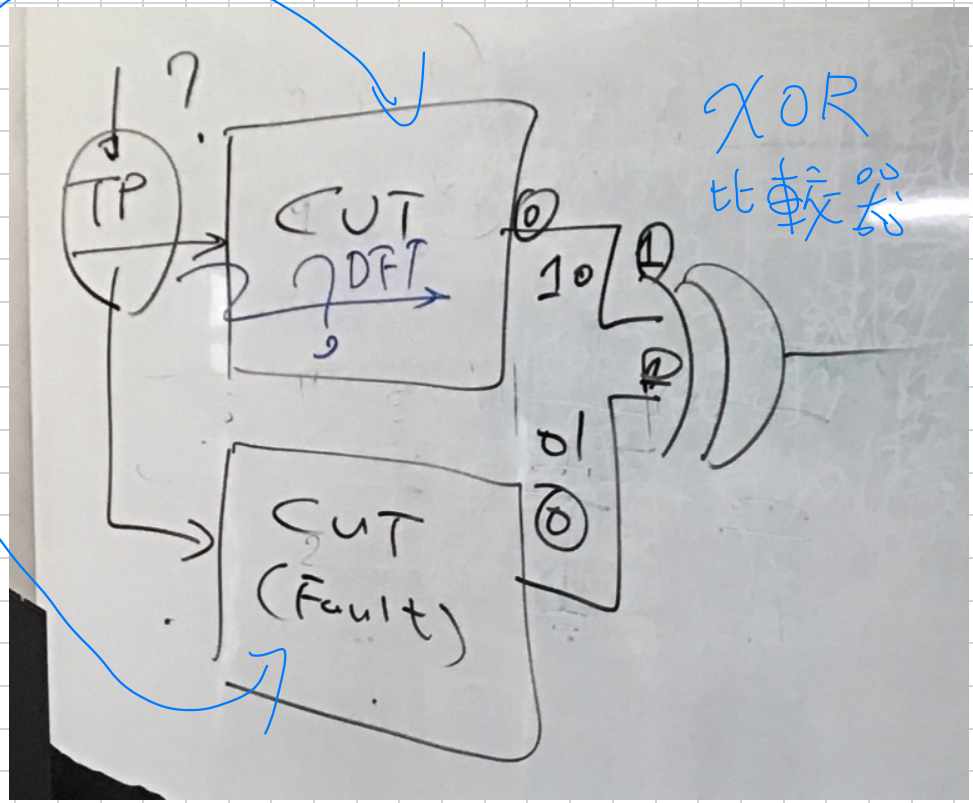
} ATPG

- Design for test DFT
- Built-in self test

} Testable design

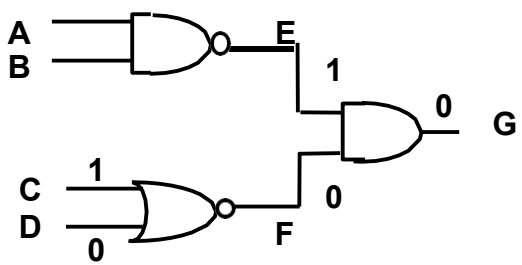
- Synthesis for testability

- **Logic simulation**
- **Fault simulation**
- **Test generation**



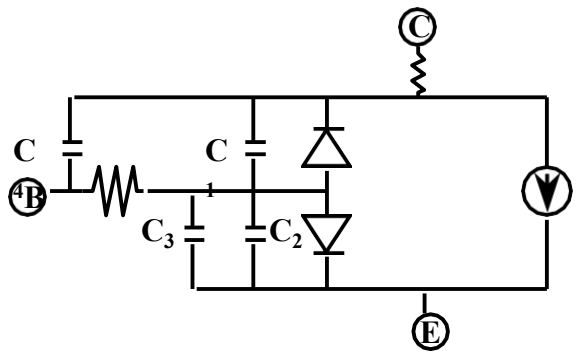
Circuit Modeling

- **Functional model---** logic function
  - f(x1,x2,...)=...
  - Truth table
- **Behavioral model---** functional + timing
  - f(x1,x2,...)=... , Delay = 10
- **Structural model---** collection of interconnected components or elements

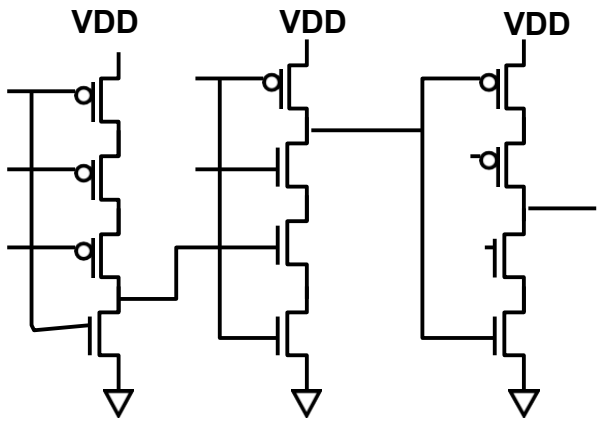


Levels of Description

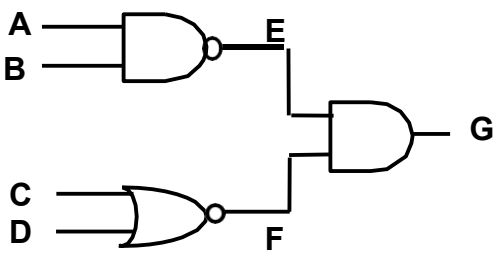
- Circuit level



- Switch level



- Gate level



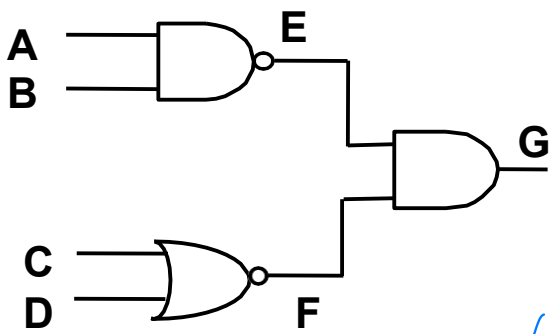
- Higher/ System level

Fault Modeling

任何時間只有一个点錯

- **The effects of physical defects**
- **Most commonly used fault model: Single stuck-at fault**

stuck at 1
卡 在 1



其它 錯

A s-a-1 B s-a-1 C s-a-1 D s-a-1
A s-a-0 B s-a-0 C s-a-0 D s-a-0

E s-a-1 F s-a-1 G s-a-1
E s-a-0 F s-a-0 G s-a-0

(通常只測這14个錯) **14 faults**

- **Other fault models:**
  - Break faults, Bridging faults, Transistor stuck-open faults, Transistor stuck-on faults, Delay faults (為高速电路)

所有錯

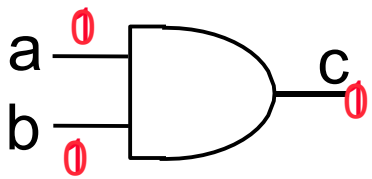
這個可以看 測試的質量 可以測試多少錯誤

Fault Coverage (FC)

$$FC = \frac{\text{\# faults detected}}{\text{\# faults in fault list}}$$

在所有錯誤裡 可以測出多少種

Example:



6 stuck-at faults

(a_0,a_1,b_0,b_1,c_0,c_1)

只用0 0 的輸入 就只能測出 c 卡在1的錯誤

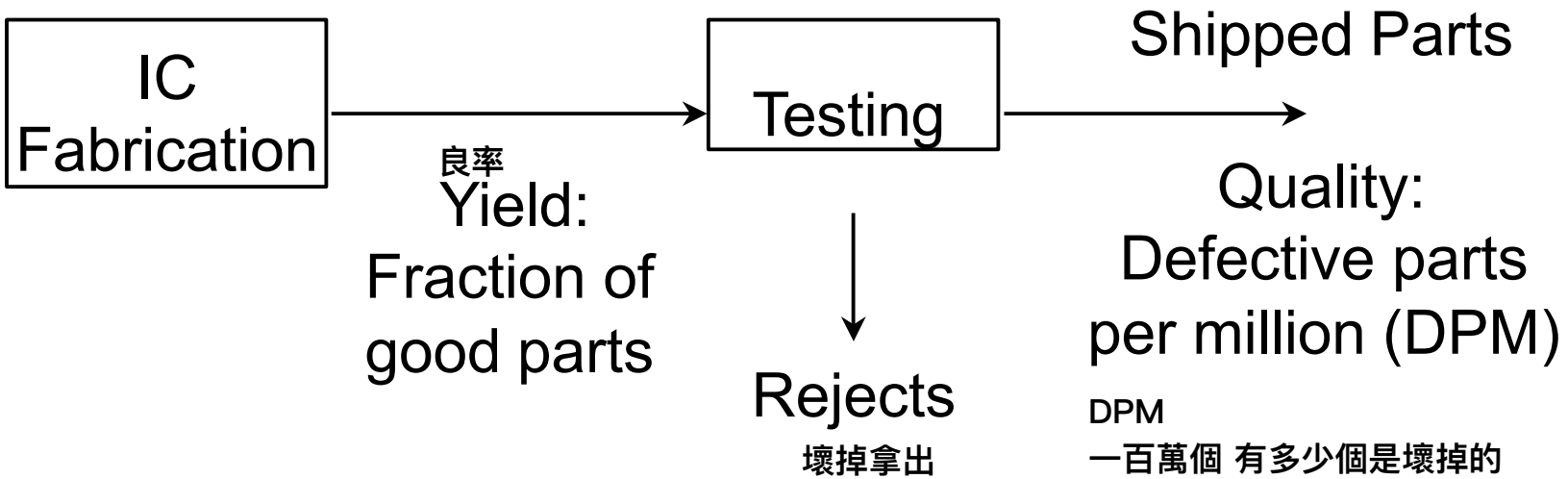
Test	faults detected	FC
{(0,0)}	c_1	16.67%
{(0,1)}	a_1,c_1	33.33%
{(1,1)}	a_0,b_0,c_0	50.00%
{(0,0),(1,1)}	a_0,b_0,c_0,c_1	66.67%
{(1,0),(0,1),(1,1)}	all	100.00%

0 1輸入 可以測出兩種

要測出全部 竟然不用全部狀況

可以省時間 時間就是金錢

Testing and Quality



- **Quality of shipped parts is a function of yield Y and the test (fault) coverage T**
- **Defect level (DL) : fraction of shipped parts that are defective**

瑕疵的比率 程度

Defect Level, Yield and Fault Coverage

測試錯誤的覆蓋率
在IC設計時決定的

$$DL = 1 - Y^{(1-T)}$$

DL: defect level

Y: yield

T: fault coverage

良率

Yield (Y)	Fault Coverage (T)	DPM (DL)
50%	90%	67,000
75%	90%	28,000
90%	90%	10,000
95%	90%	5,000
99%	90%	1,000
90%	90%	10,000
90%	95%	5,000
90%	99%	1,000
90%	99.9%	100

一百萬個
只有1000是壞的
還是挺多

所以盡量提升 測試錯誤的覆蓋率
良率下降一點 沒關係

這樣給別人的 就可以少很多壞掉的

Test Quality Issues

- True pass and true reject are correct decision
- Test escapes = defective chips that pass test
  Also known as under-testing
- Yield loss = good chips that fail the tests
  Also known as overkill, over-testing
- Testing goal reduces both test escape and yield loss
- Quality test reduces test escape but increases yield loss
- Low cost test reduces yield loss but increase test escape

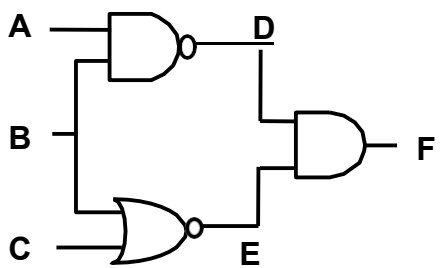
好的IC　　壞掉的ic

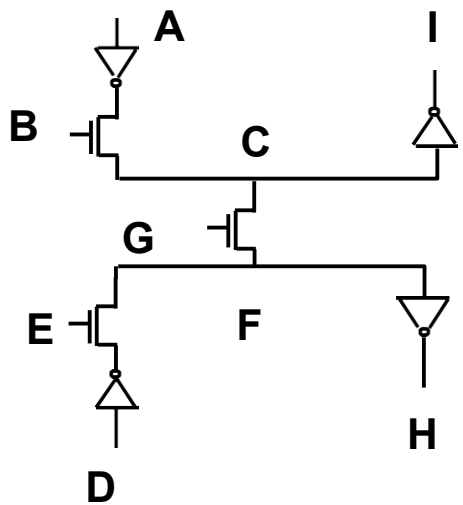
	Good IC	Defective IC
Pass tests	True PASS	Test Escapes
Fail tests	Yield Loss	True Reject

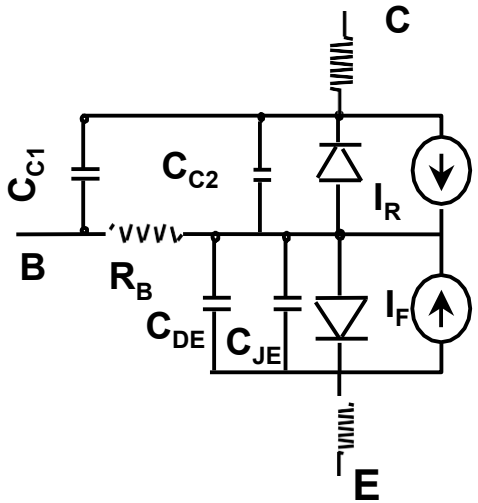
好的被當成壞的
這個業界很在意
好不容易有好的 被丟掉

Logic simulation

- To determine how a good circuit should work
- Given input vectors, determine the normal circuit response





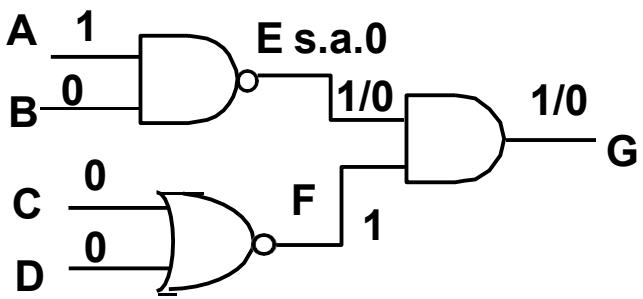


Fault simulation

- **To determine the behavior of faulty circuits**

找到那些能測出錯誤的 pattern
希望有越少的pattern
而 測出更多的問題

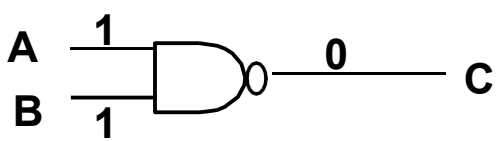
因為要盡量減少儲存pattern
的空間
越少pattern 測越快



- **Given a test vector, determine all faults that are detected by this test vector.**

Example:

像這個1 1就可以測出三個錯誤

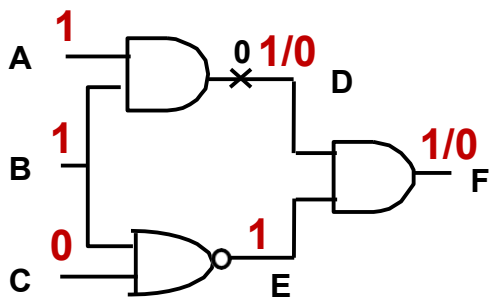


Test vector (1 1) detects
{ a_0, b_0, c_1}

Test generation

- Given a fault, identify a test to detect this fault

Example:



為了要測出 D卡在0
而我們只能看到F的輸出
所以要有110的輸入 才能看出d有沒有問題
但不確定是不是f卡在1

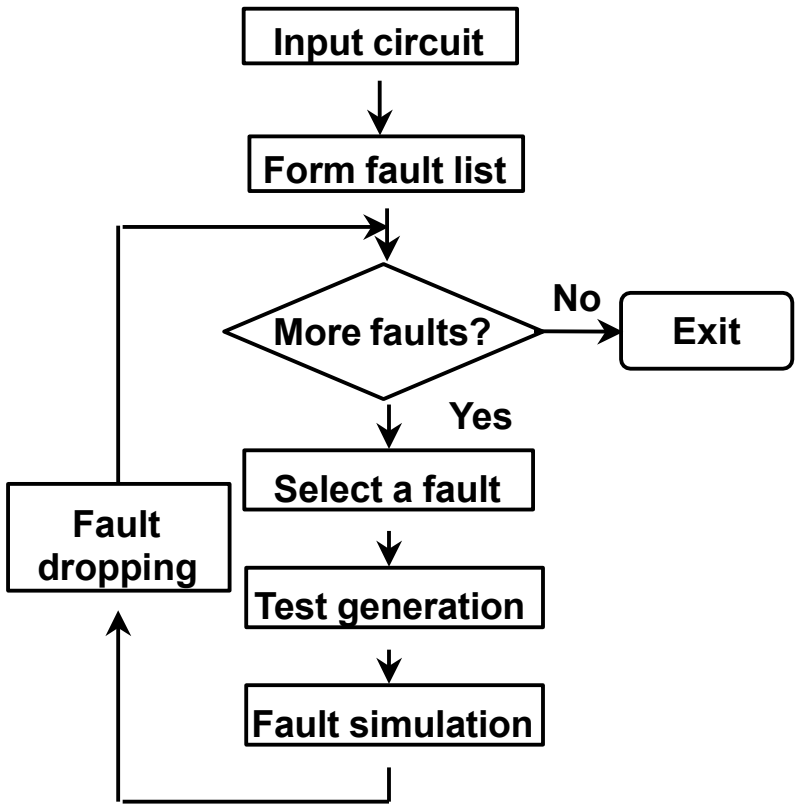
To detect D s-a-0, D must be set to 1.
Thus A=B=1.

To propagate fault effect to the primary output
E must be 1. Thus C must be 0.

Test vector: A=1, B=1, C=0

Automatic Test Pattern Generation (ATPG)

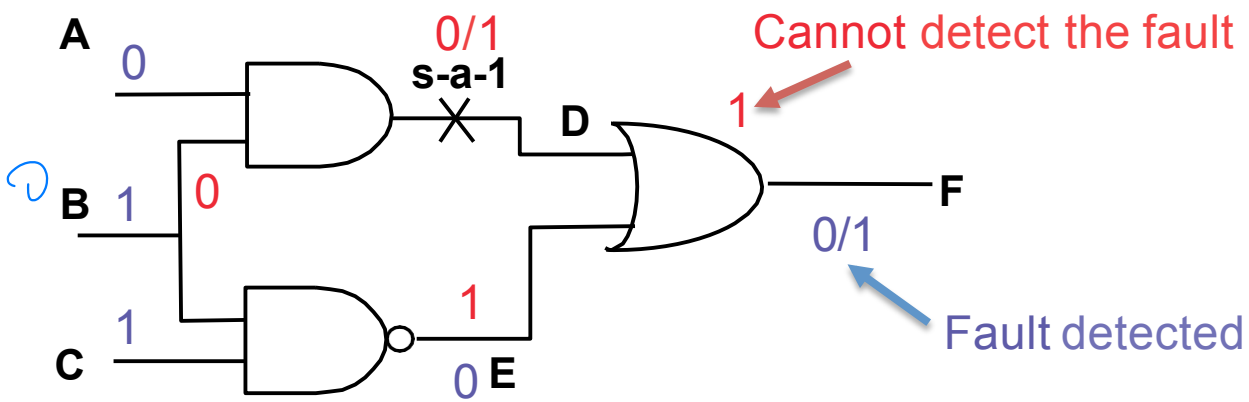
- Given a circuit, identify a set of test vectors to detect all faults under consideration.



Difficulties in test generation

1. Reconvergent fanout

001的輸入 試測不出 D卡在1
所以要重找一個 輸入



↪ 用 1 值測

A SA1 0

SA0 1

B SA1 0

SA0 1

C SA1 0

SA0 1

D SA1 0

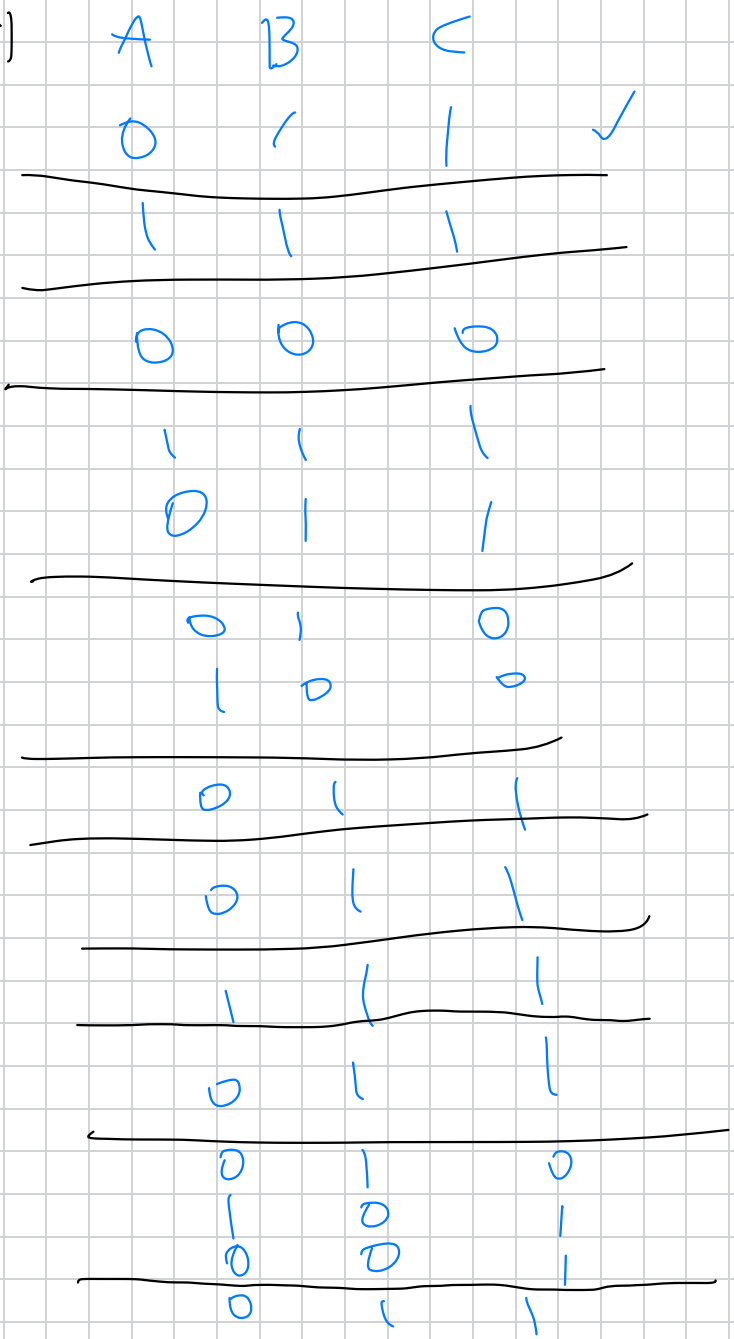
SA0 1

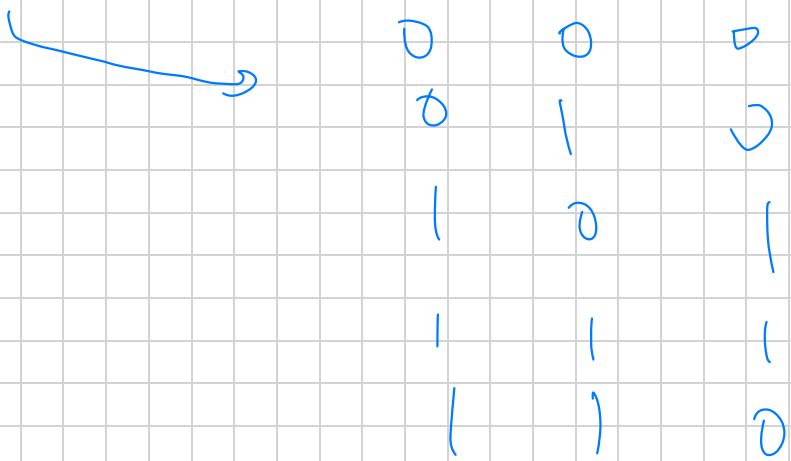
E SA1 0

SA0 1

F SA1 0

SA0 1



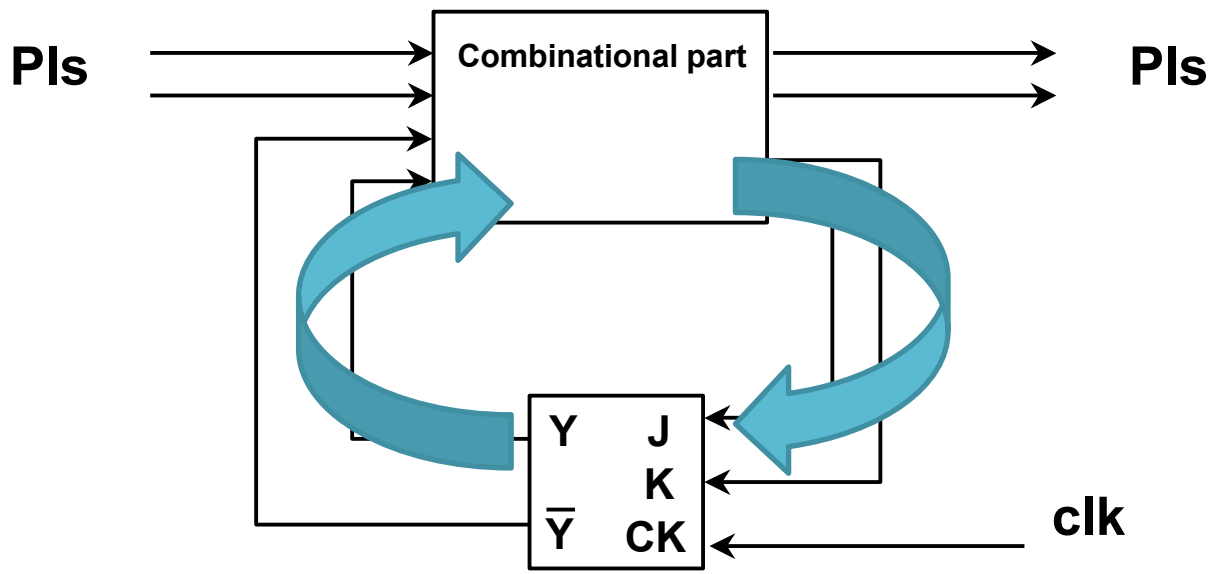


Difficulties in test generation (cont.)

000
001
010
011
.
.
.

續向邏輯 也很難測
要把前面的都先測出來

2. Sequential test generation



Testable Design

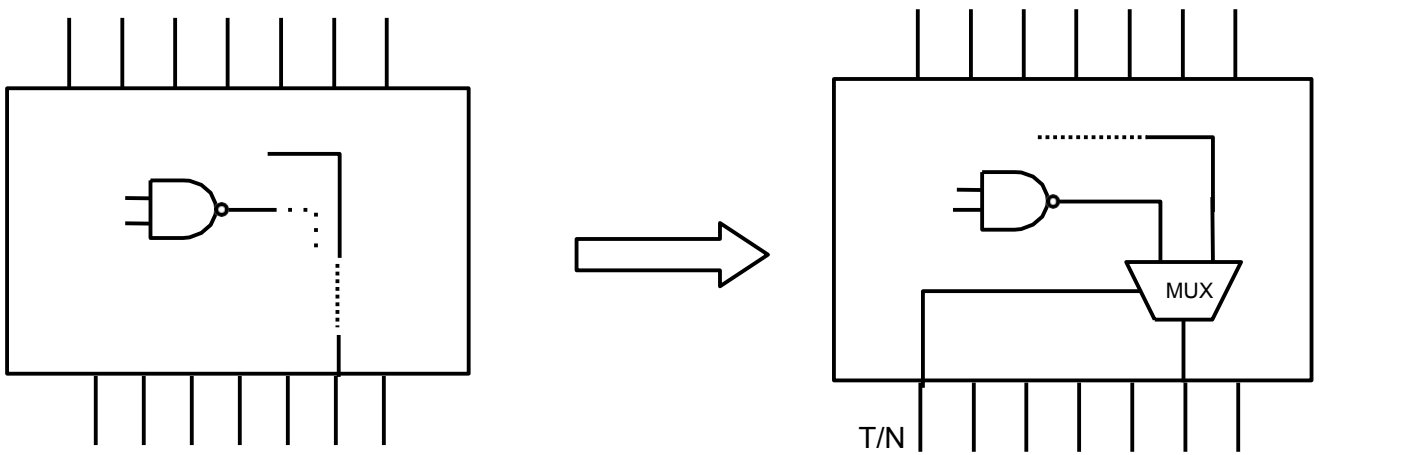
- Design for testability (DFT)
  - *ad hoc* techniques
  - Scan design
  - Boundary Scan 電路板 掃描
    用電路針 測不到
- Built-In Self Test (BIST)
  - Random number generator (RNG)
  - Signature Analyzer (SA)
- Synthesis for Testability

Example of *ad hoc* techniques

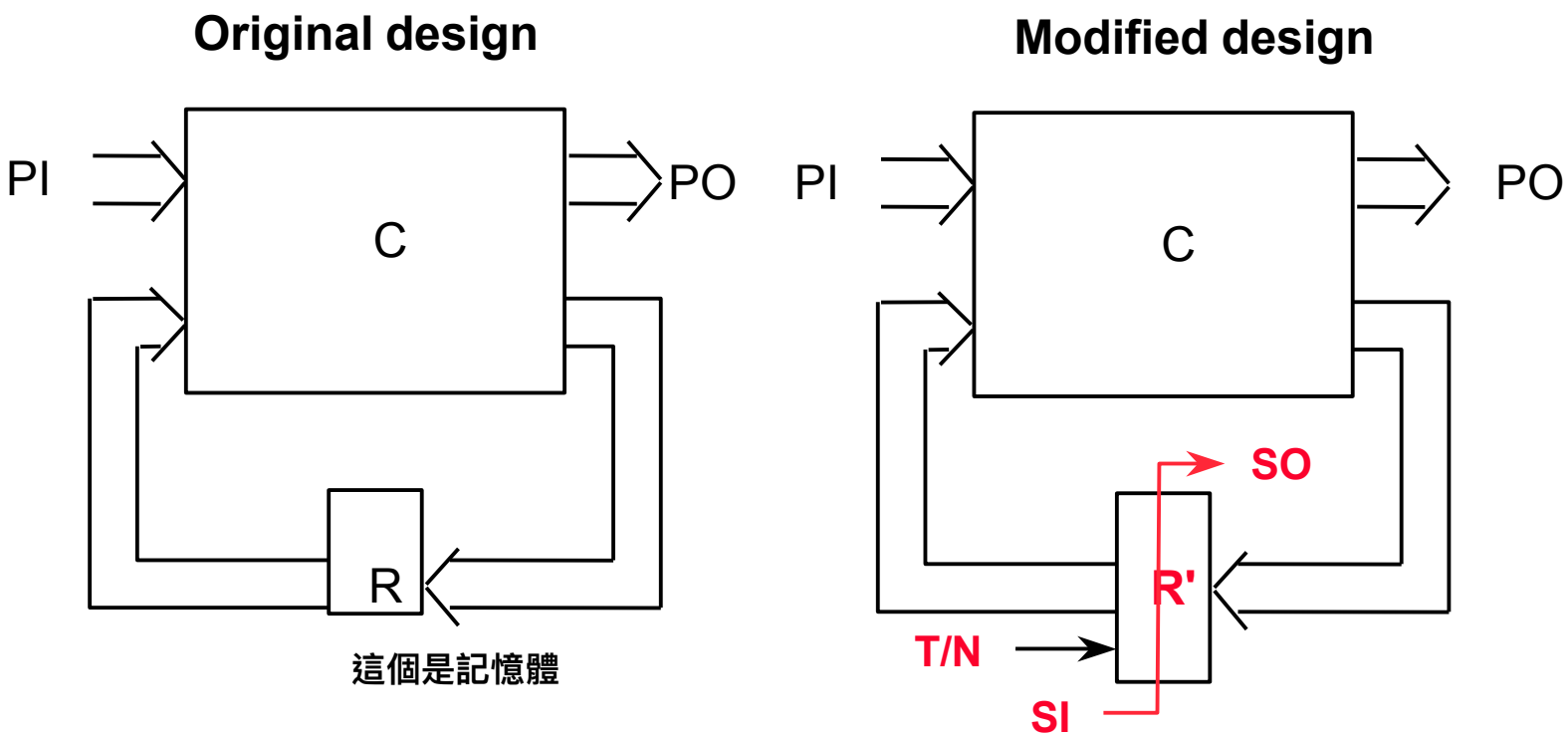
Insert test point

還沒加測試電路長這樣

多工器 這裡可以用來測試虛線
會多一隻腳 用來做多工器的選擇

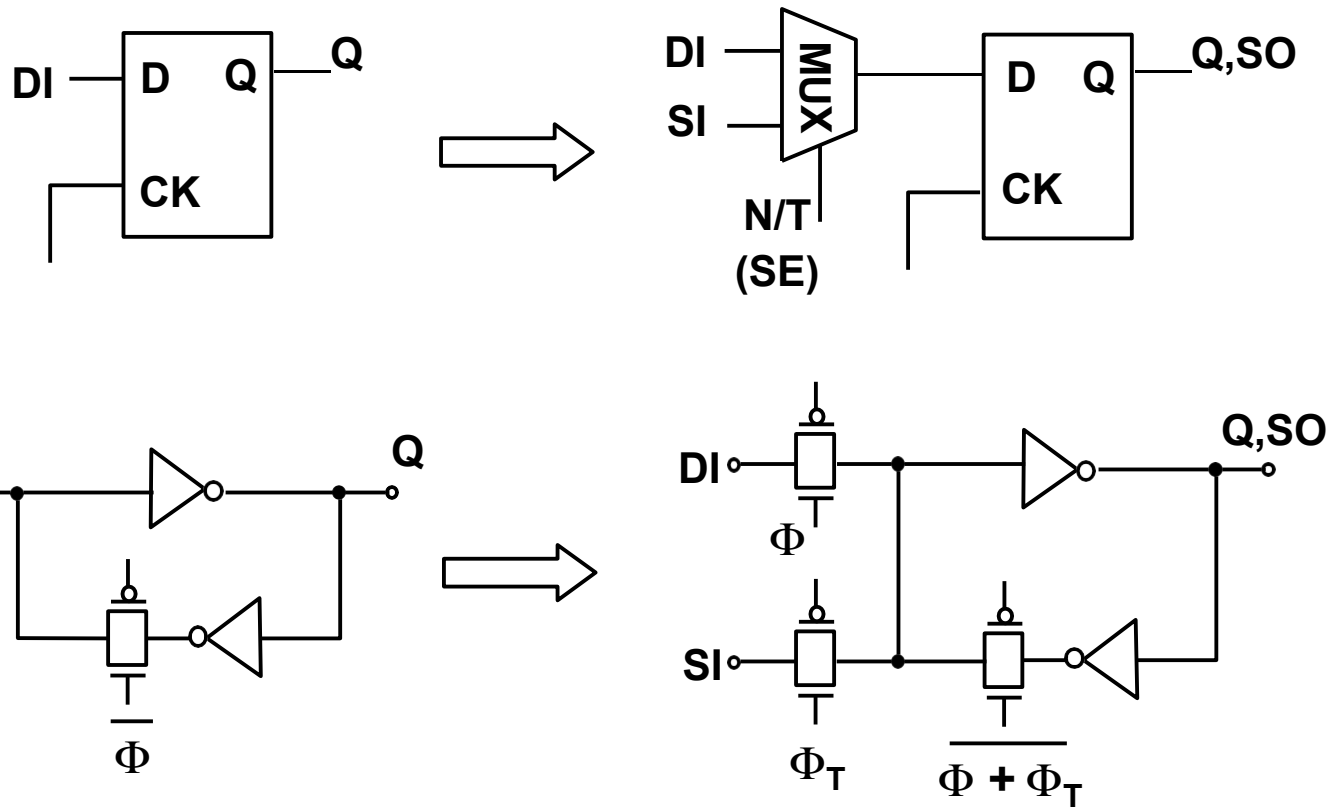


Scan System

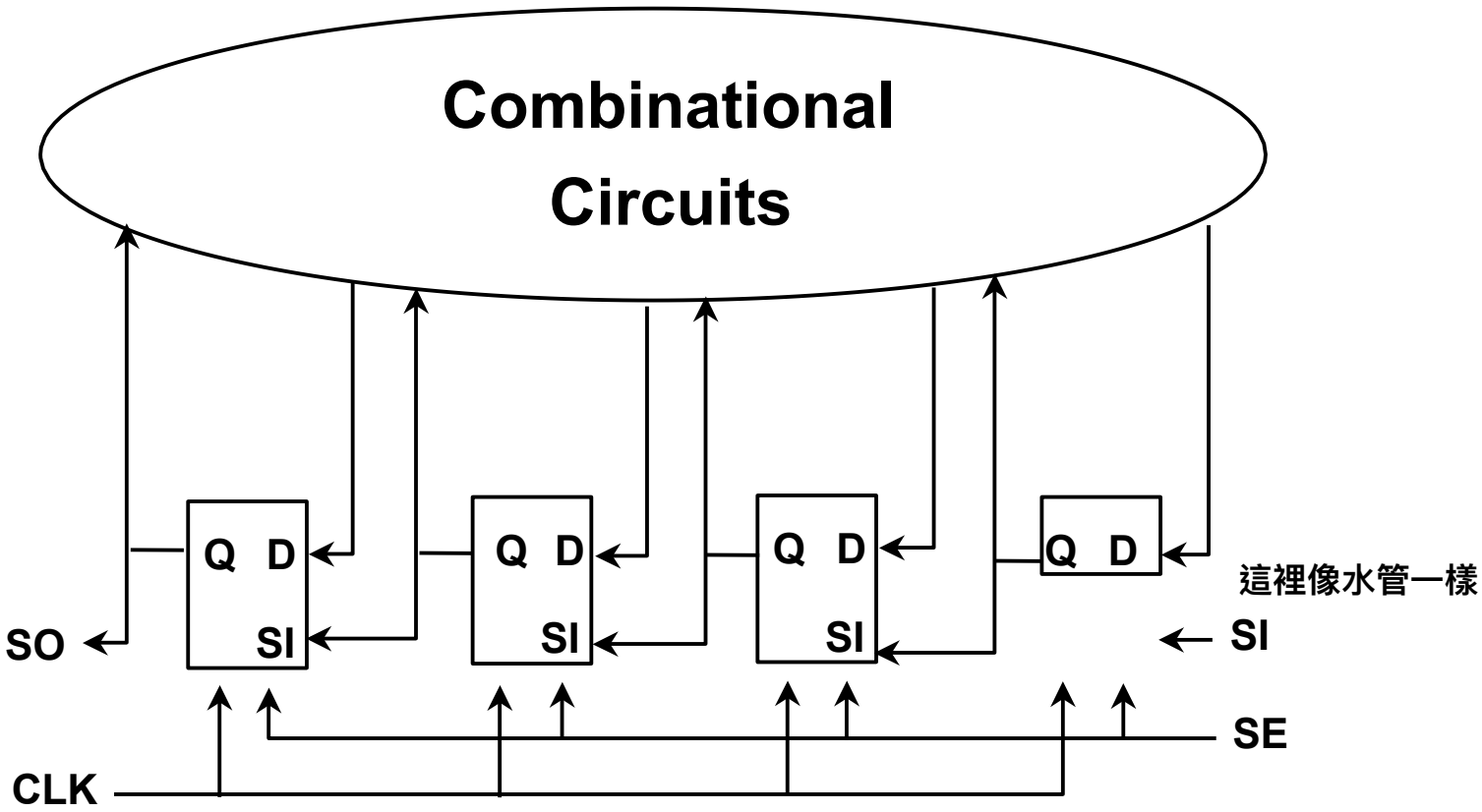

Original design
PI
C
PO
R
這個是記憶體
Modified design
PI
C
PO
SO
R'
T/N
SI
這個正反器比較不一樣

Scan Cell Design

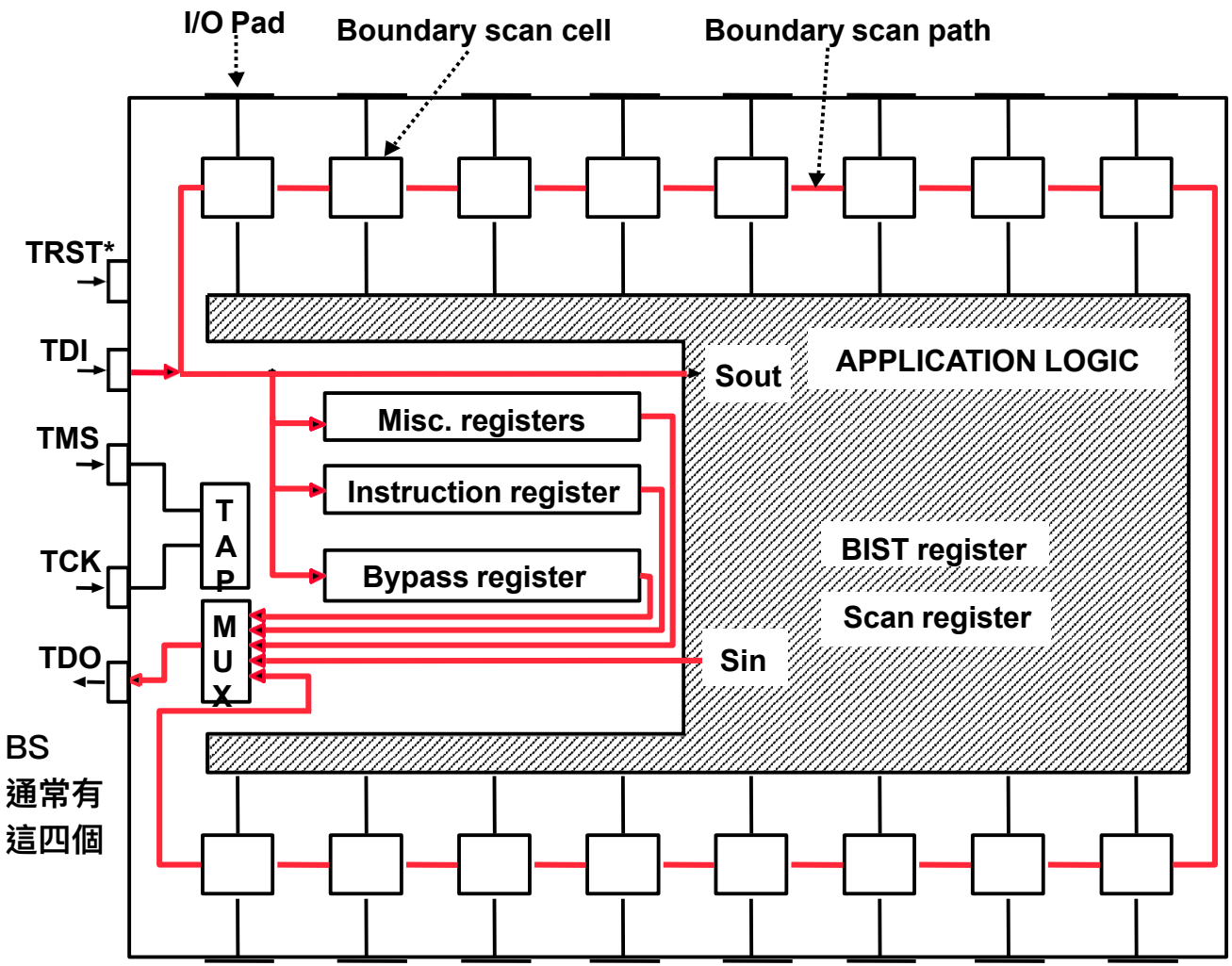
會長這樣



Scan Register

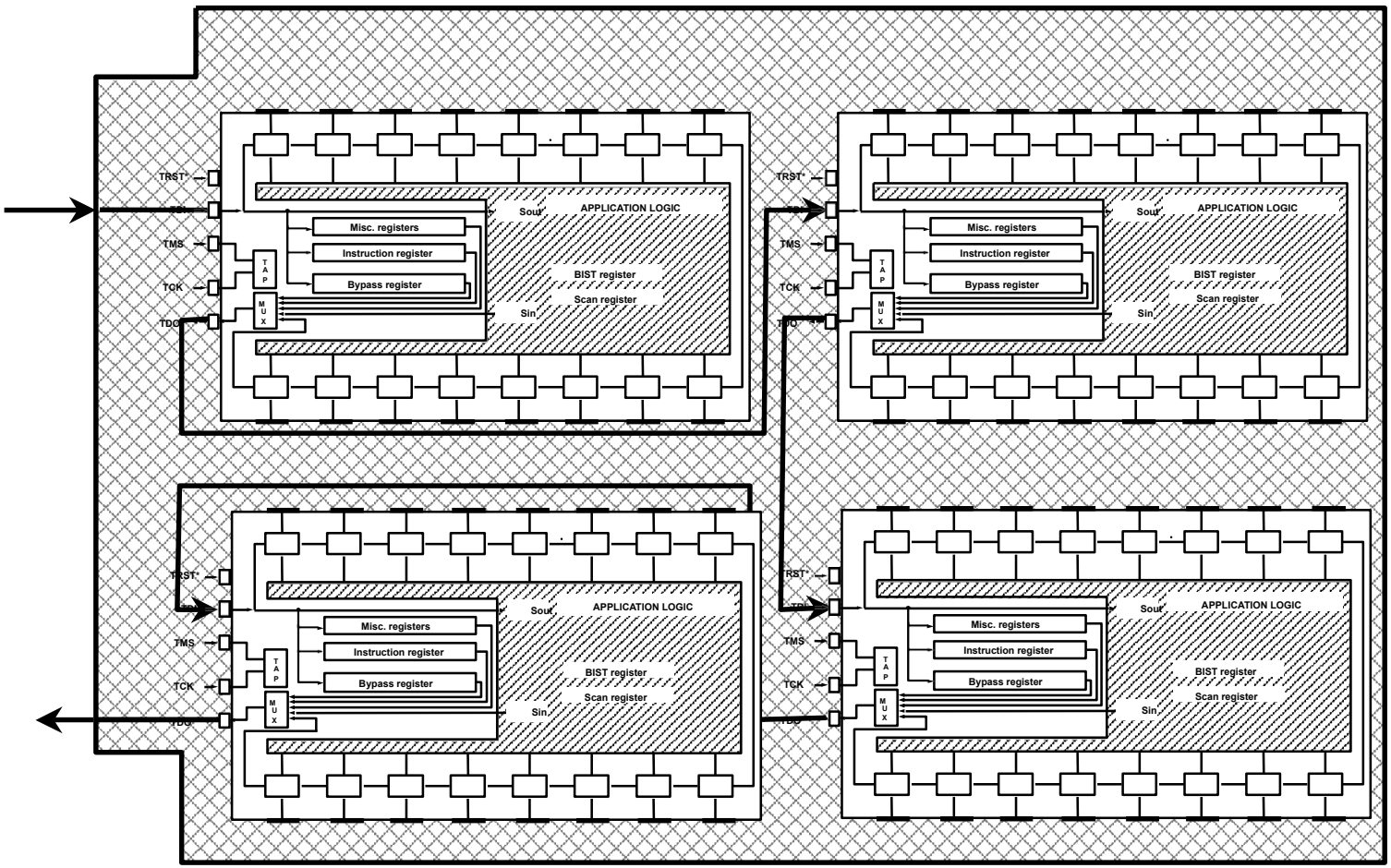


Boundary Scan



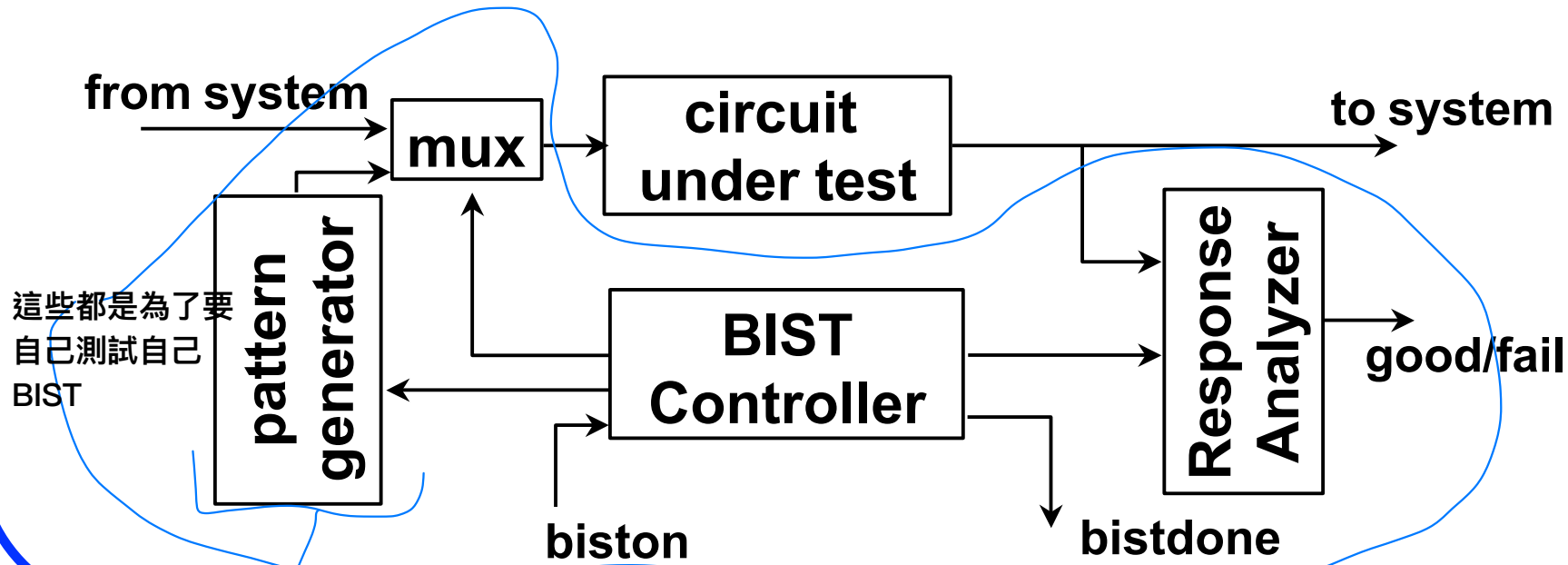
TRST*:Test rest (Optional)
TDI: Test data input
TD0: Test data output
TCK: Test clock
TMS: Test mode select

Boundary Scan (Cont.)



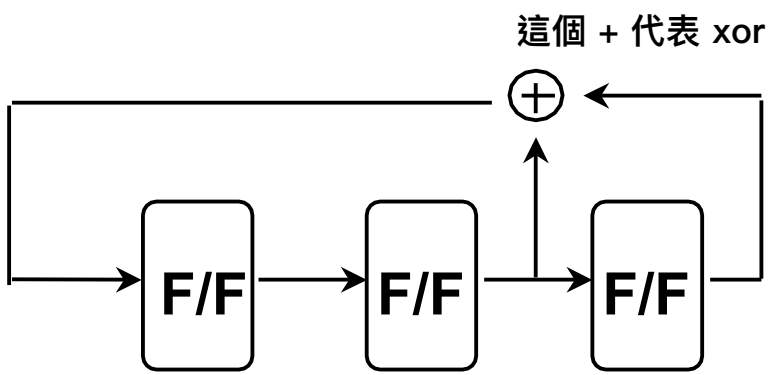
Built-In-Self Test (BIST)

- Places the job of device testing inside the device itself
- Generates its own stimulus and analyzes its own response



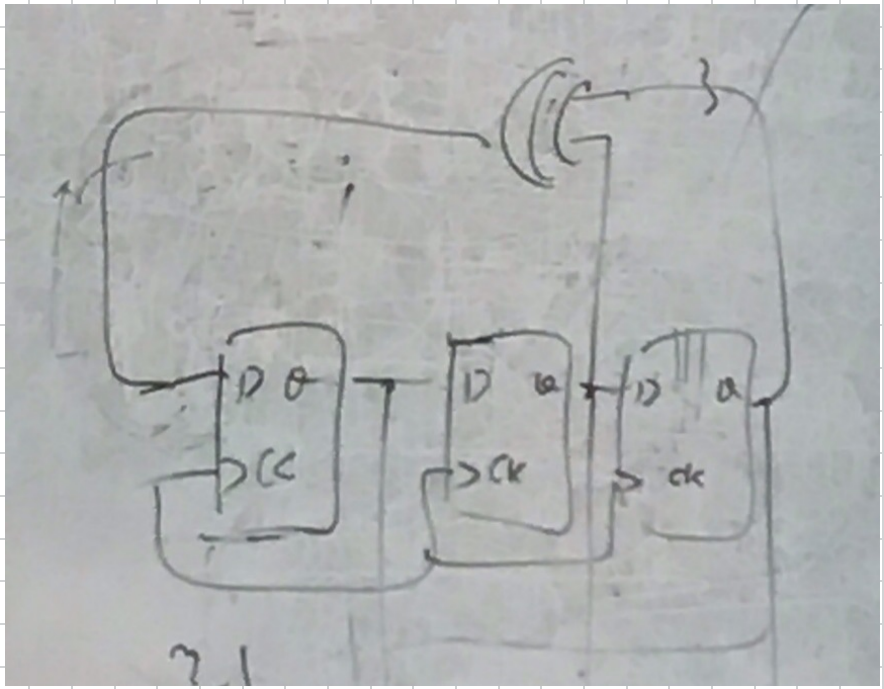
Built-In-Self Test (BIST) (Cont.)

- **Two major tasks**
  - **Test pattern generation**
  - **Test result compaction**
- **Usually implemented by linear feedback shift register**

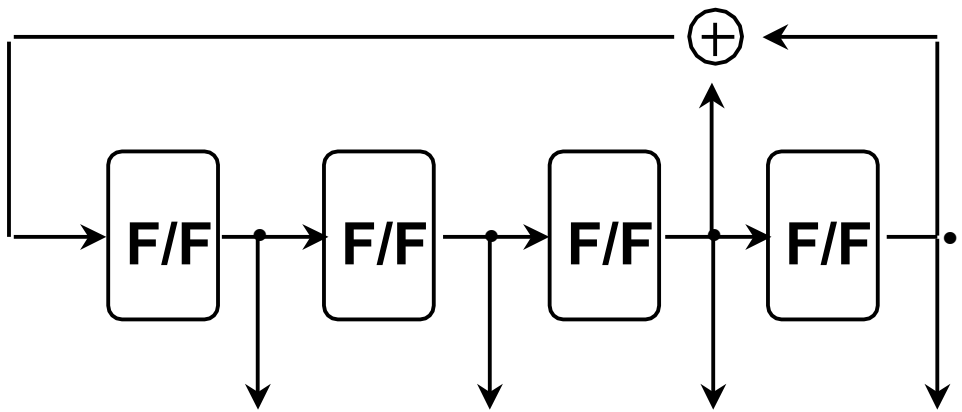


這就是一個
Pattern generator
只需三個 FF

這是random的


D Q
>Clk
D Q
>Clk
D Q
>Clk

Random Number Generator (RNG)



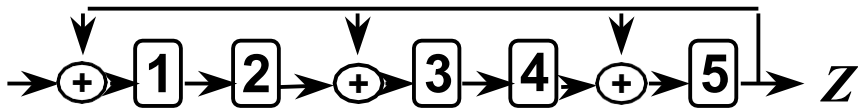
0001	0110	1111
1000	1011	0111
0100	0101	0011
0010	1010	0001
1001	1101	(repeat)
1100	1110	

1. Generate “pseudo” random patterns
2. Period is $2^n - 1$

這個叫做 虛亂數
沒什麼規則
但只要決定好 一開始 後面都有規律的
可以更好的測試出錯誤

Signature Analyzer (SA)

Input sequence 10101111 (8 bits)



$G(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^7$ $\quad P(x) = 1 + x^2 + x^4 + x^5$

可以用多項式 來表達這個變數

Time	Input stream	Register contents	Output stream
0	1 0 1 0 1 1 1 1	0 0 0 0 0 ←	Initial state
1	1 0 1 0 1 1 1	1 0 0 0 0	
.	.	.	
.	.	.	
5	1 0 1	0 1 1 1 1	
6	1 0	0 0 0 1 0	1
7	1	0 0 0 0 1	0 1
8		0 0 1 0 1	1 0 1

Remainder

$R(x) = x^2 + x^4$

Quotient

$1 + x^2$

Signature Analyzer (SA) (Cont.)

$$P(x): x^5 + x^4 + x^2 + 1$$

$$\times\ Q(x): x^2 + 1$$

$$x^7 + x^6 + x^4 + x^2 + x^5 + x^4 + x^2 + 1$$

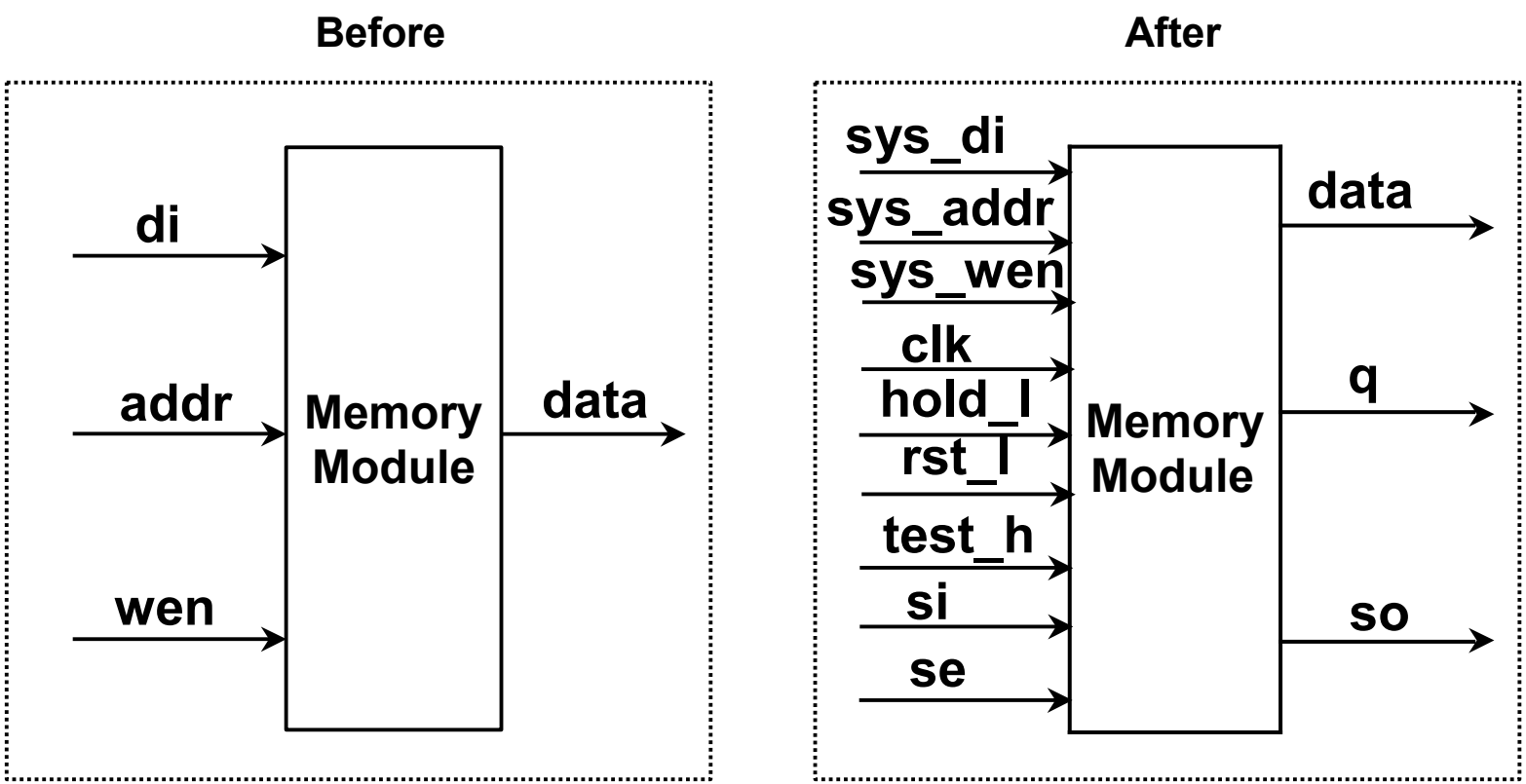
$$= x^7 + x^6 + x^5 + 1$$

$$P(x)Q(x) + R(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1 = G(x)$$

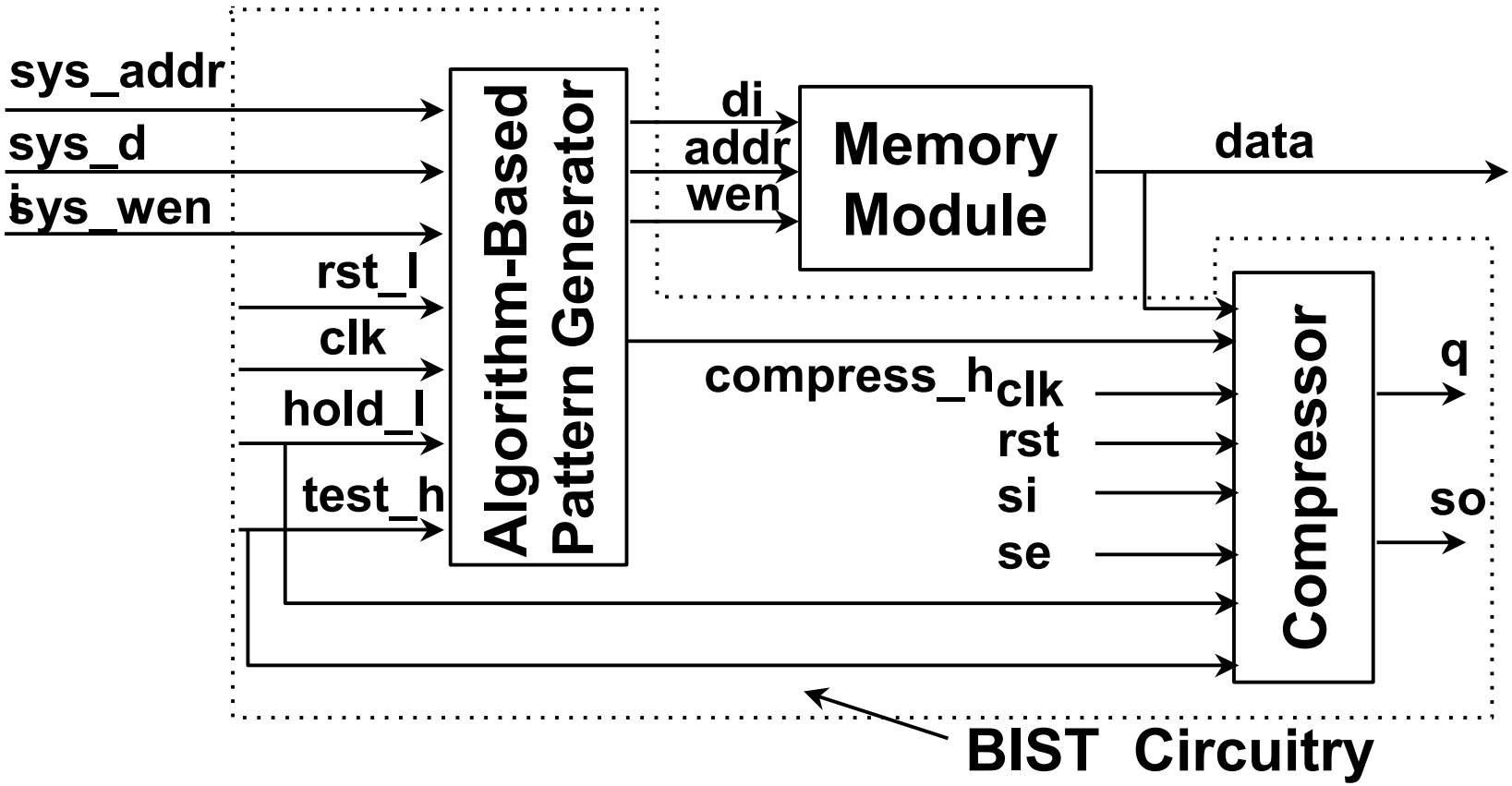
Prob. of aliasing error = $1/2^n$

where n is # of FFs

Memory BIST Architecture with a Compressor



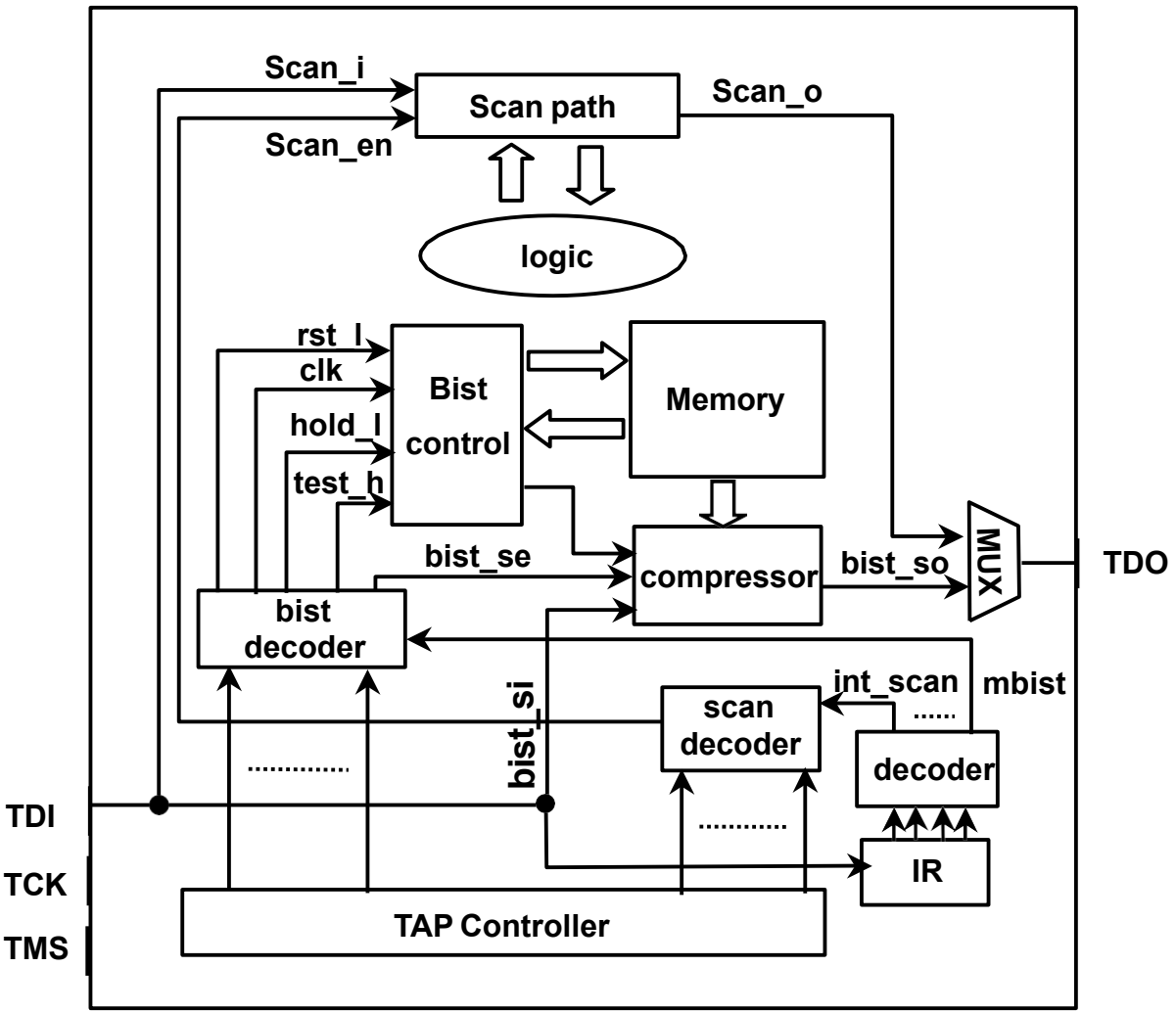
Memory BIST Architecture with a Compressor (Cont.)



Synthesis for Testability

- Automatic v.s Semi-automatic
- Commercial products
  - Testability analysis tools
  - Full / partial scan insertion
  - BIST insertion
  - Boundary scan insertion
- Research
  - RTL synthesis
  - FSM synthesis
  - Gate level synthesis
  - Boolean equation synthesis

CPU Test Control Architecture



Problems re-thinking

- A 32-bit adder --- ATPG
- A 32-bit counter --- Design for testability + ATPG
- A 1MB Cache memory --- BIST
- A 10^7-transistor CPU --- All test techniques