國立清華大學電機系

EE-6250
超大型積體電路測試
VLSI Testing

Chapter 11
Logic Diagnosis

Outline

- Introduction
- Combinational Logic Diagnosis
- Scan Chain Diagnosis
- Logic BIST Diagnosis
- Conclusion

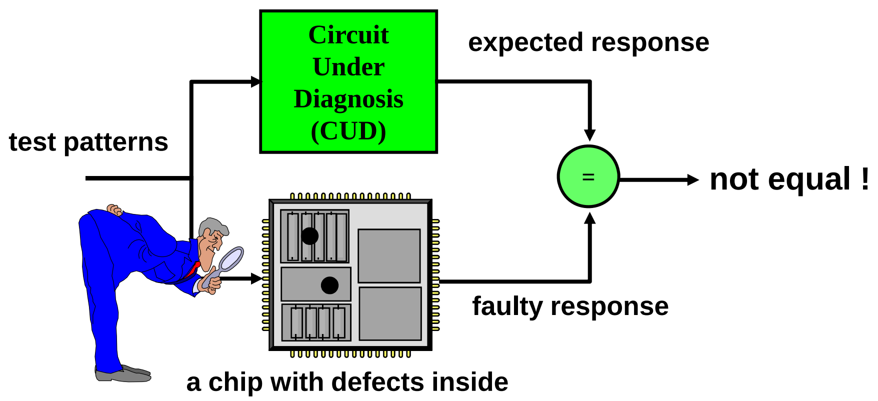
What would you do when chips fail?

- ❑ **Is it due to design bugs?**
  - **If most chip fails with the same syndrome when running an application**
- ❑ **Is it due to parametric yield loss?**
  - **Timing-related failure?**
    - **Insufficient silicon speed?**
  - **Noise-induced failure?**
    - **supply noise, cross-talk, leakage, etc.?**
  - **Lack of manufacturability?**
    - **inappropriate layout?**
- ❑ **Is it due to random defects?**
  - **Via misalignment, Via/Contact void, Mouse bite,**
  - **Unintentional short/open wires, etc.**

Ch11-3

Problem: Fault Diagnosis

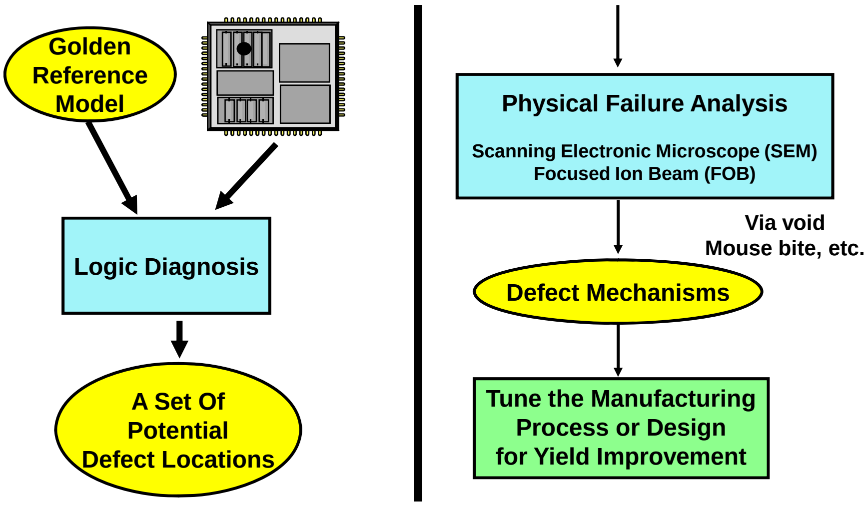
This chapter focuses more on diagnosis of defects or faults, not design bugs



Question: Where are the fault locations ?

Ch11-4

Diagnosis For Yield Improvement



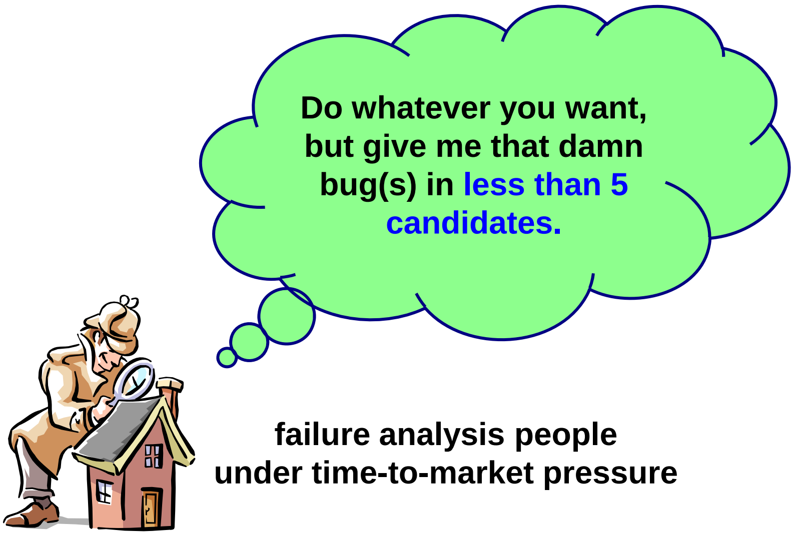
Ch11-5

Quality Metrics of Diagnosis

- **Success rate**
  - The percentage of hitting at least one defect in the physical failure analysis
  - This is the ultimate goal of failure analysis
- **Diagnostic resolution**
  - Total number of fault candidates reported by a tool
  - The perfect diagnostic resolution is 1
  - Though perfect resolution does not necessarily imply high hit rate
- **First-hit index**
  - Used for a tool that reports a ranked list of candidates
  - Refers to the index of the first candidate in the ranked list that turns out to be a true defect site
  - Smaller first-hit index indicates higher accuracy
- **Top-10 hit**
  - Used when there are multiple defects in the failing chip
  - The number of true defects in the top 10 candidates

Ch11-6

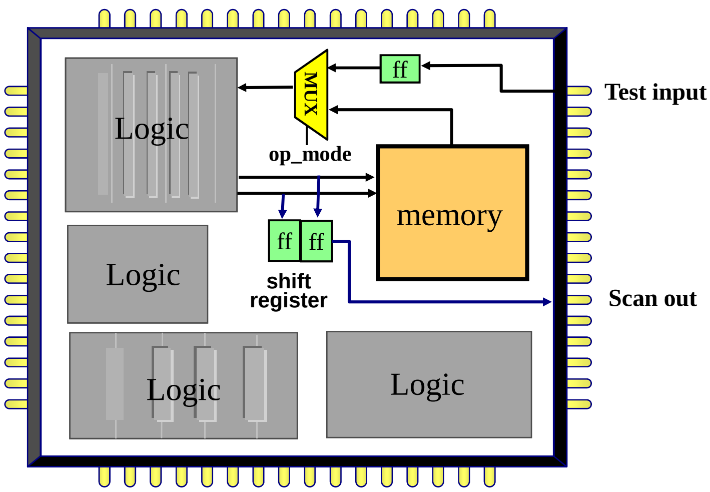
Challenge



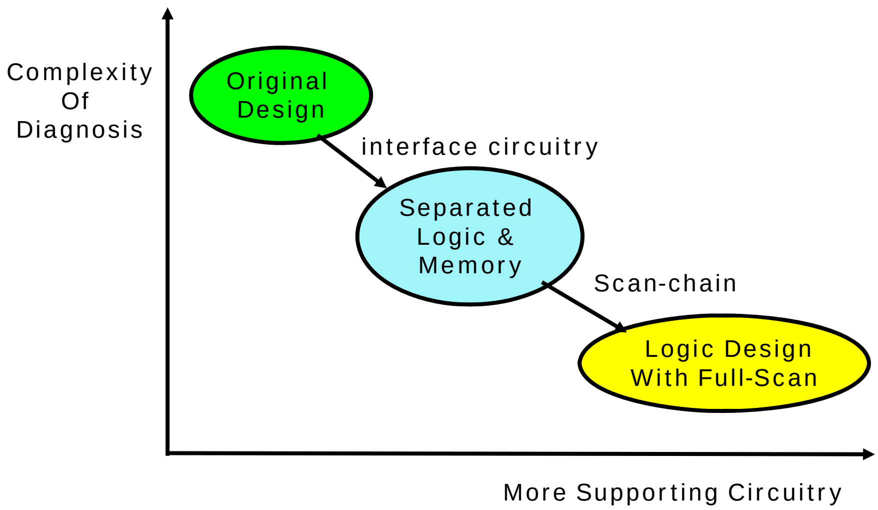
failure analysis people under time-to-market pressure

Supporting Circuitry

Supporting Circuitry:
Makes Logic's <u>inputs controllable</u> and <u>outputs observable</u>



Design For Diagnosis



Ch11-9

Possible Assumptions Used in Diagnosis

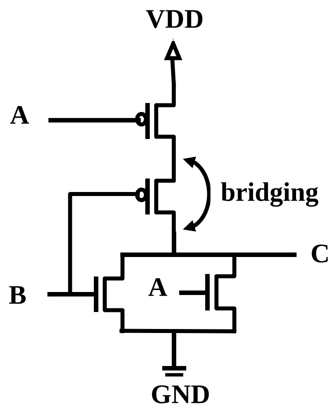
- **Stuck-At Fault Model Assumption**
  - **The defect behaves like a stuck-at fault**
- **Single Fault Assumption**
  - **Only one fault affecting any faulty output**
- **Logical Fault Assumption**
  - **A fault manifests itself as a logical error**
- **Full-Scan Assumption**
  - **The chip under diagnosis has to be full-scanned**

Note: A diagnosis approach less dependent on the fault assumptions is more capable of dealing with practical situations.

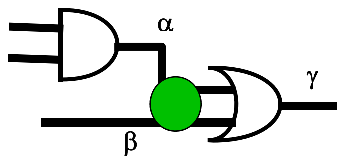
Ch11-10

Examples of Faults

- Node Fault



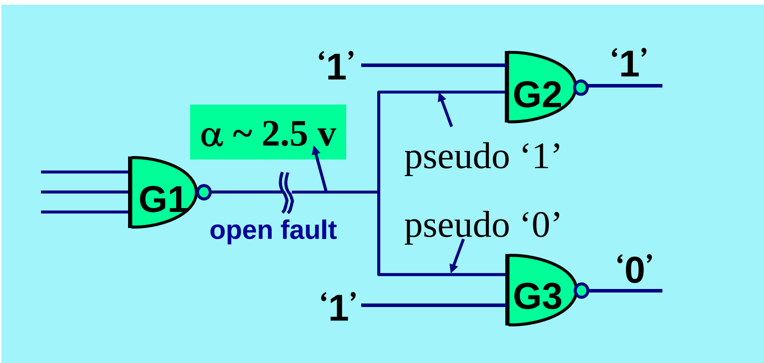
- Short Fault (Bridging)



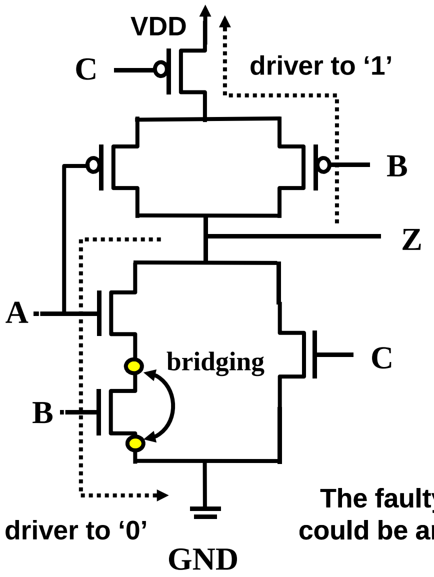
Most diagnosis algorithms performs at the gate level, trying to identify the troubling signals or cells

Byzantine Open Fault

- Definition of Byzantine Fault:
  - A fault that causes an ambiguous voltage level



A Byzantine Node Type



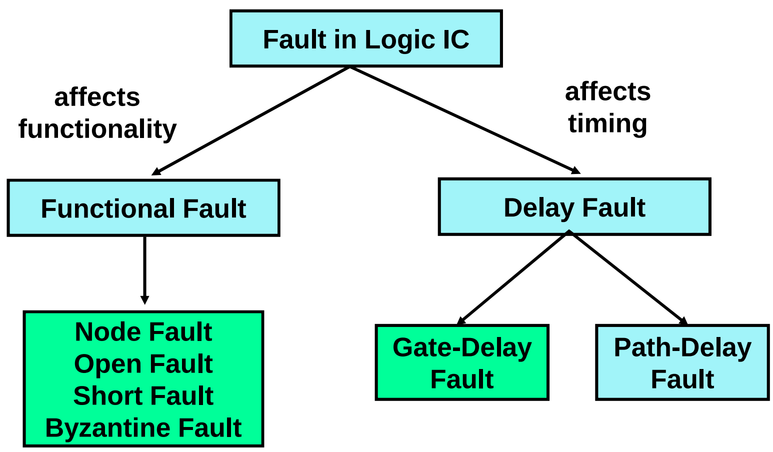
faulty min-term →

The faulty output could be ambiguous

Truth Table

A	B	C	Z	Z^f
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	~0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Fault Classification

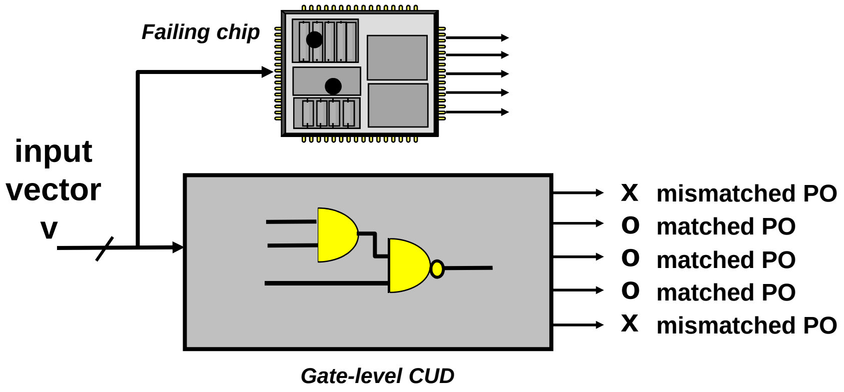


Outline

- Introduction
- **Combinational Logic Diagnosis**
  - Cause-Effect Analysis
  - Effect-Cause Analysis
  - Chip-Level Strategy
  - Diagnostic Test Pattern Generation
- Scan Chain Diagnosis
- Logic BIST Diagnosis
- Conclusion

Terminology

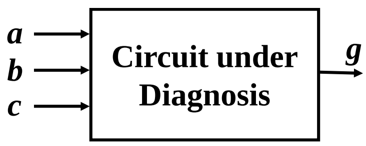
- **Device Under Diagnosis** (DUD): The Failing Chip
- **Circuit Under Diagnosis** (CUD): The Circuit Model
- **Failing Input Vector**: Causes Mismatches



Cause-Effect Analysis

- Fault dictionary (pre-analysis of all causes)
  - **Records test response of every fault under the applied test set**
  - **Built by intensive fault simulation process**
- A chip is diagnosed (effect matching)
  - **By matching up the failing syndromes observed at the tester with the pre-stored fault dictionary**

Fault Dictionary Example

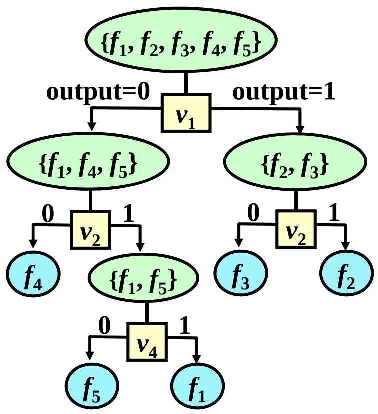


(a) Circuit under diagnosis

Circuits	Test vectors in terms of (*a*, *b*, *c*)				
	v_1	v_2	v_3	v_4	v_5
fault-free	0	0	0	0	1
f_1	0	1	1	1	1
f_2	1	1	1	0	1
f_3	1	0	0	1	1
f_4	0	0	1	0	0
f_5	0	1	1	0	1

(b) Full-response dictionary

A diagnosis session:
traverse from a path from root to a leaf



(c) Diagnostic tree

Fault Dictionary Reduction – P&R

(a) Full-response table

Fault	Output Response (z_1, z_2)			
	t_1	t_2	t_3	t_4
f_1	10	10	11	10
f_2	00	00	11	00
f_3	00	00	00	00
f_4	01	00	00	01
f_5	01	00	01	01
f_6	01	00	01	01
f_7	10	00	10	00
f_8	11	11	11	11

Fault	Pass (0) or Fail (1)			
	t_1	t_2	t_3	t_4
f_1	1	1	0	1
f_2	1	0	0	1
f_3	1	0	1	1
f_4	1	0	1	0
f_5	1	0	1	0
f_6	1	0	1	0
f_7	1	0	1	1
f_8	0	1	0	1

(b) Pass-fail dictionary

(c) P&R compression dictionary

Fault ID	Pass-fail + Extra outputs					
	t_1		t_2	t_3		t_4
f_1	1	1	1	0	1	1
f_2	1	0	0	0	1	1
f_3	1	0	0	1	0	1
f_4	1	0	0	1	0	0
f_5	1	0	0	1	1	0
f_6	1	0	0	1	1	0
f_7	1	1	0	1	0	1
f_8	0	1	1	0	1	1

Response of z_1 Response of z_2

Detection Fault Dictionary

(a) Full-response table

Fault ID	Output Response (z_1, z_2)			
	t_1	t_2	t_3	t_4
f_1	10	10	11	10
f_2	00	00	11	00
f_3	00	00	00	00
f_4	01	00	00	01
f_5	01	00	01	01
f_6	01	00	01	01
f_7	10	00	10	00
f_8	11	11	11	11

failing output vectors

Fault ID	Pass (1) or Fail (0)			
	t_1	t_2	t_3	t_4
f_1	1	1	0	1
f_2	1	0	0	1
f_3	1	0	1	1
f_4	1	0	1	0
f_5	1	0	1	0
f_6	1	0	1	0
f_7	1	0	1	1
f_8	0	1	0	1

(b) Pass-fail dictionary

(c) Detection dictionary

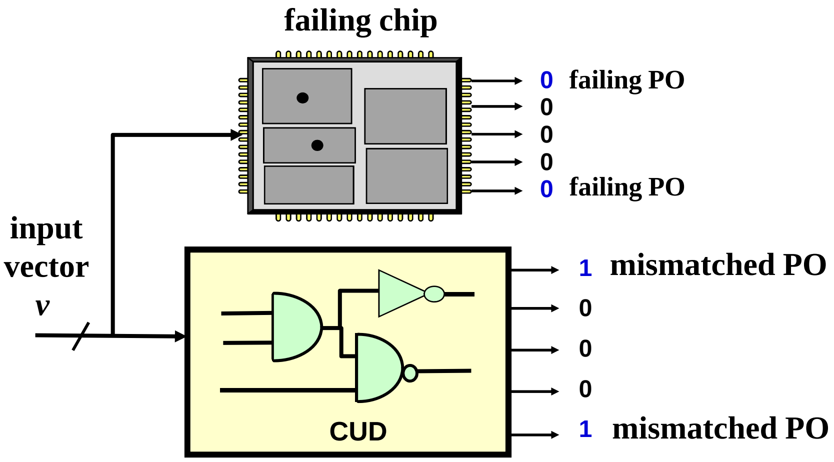
Fault ID	Detection information (Test ID : Output Vector)
f_1	t_1:10 t_2:10 t_4:10;
f_2	t_1:00 t_4:00;
f_3	t_1:00 t_3:00 t_4:00;
f_4	t_1:01 t_3:00;
f_5	t_1:01 t_3:01;
f_6	t_1:01 t_3:01;
f_7	t_1:10 t_3:10 t_4:00;
f_8	t_2:10 t_4:11;

Outline

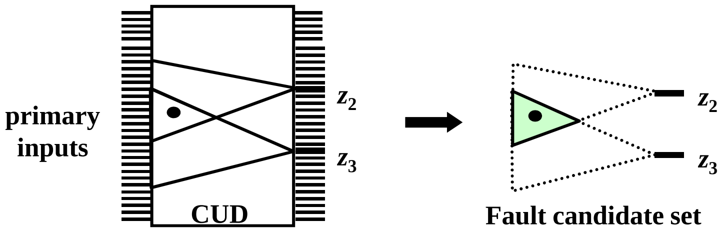
- Introduction
- Combinational Logic Diagnosis
  - Cause-Effect Analysis
  - **Effect-Cause Analysis**
  - Chip-Level Strategy
  - Diagnostic Test Pattern Generation
- Scan Chain Diagnosis
- Logic BIST Diagnosis
- Conclusion

Terminology: Mismatched Output

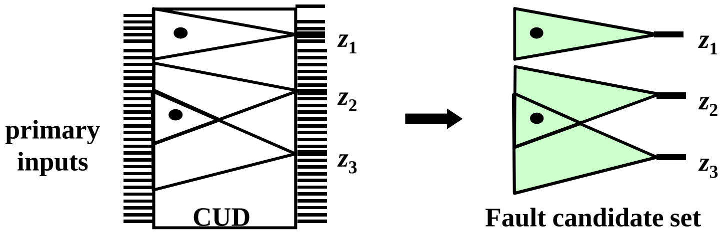
Effect-cause analysis does not build fault dictionary
It predicts fault locations by analyzing CUD from mismatch PO's



Structural Pruning – Intersection or Union?



(a) Cone intersection.



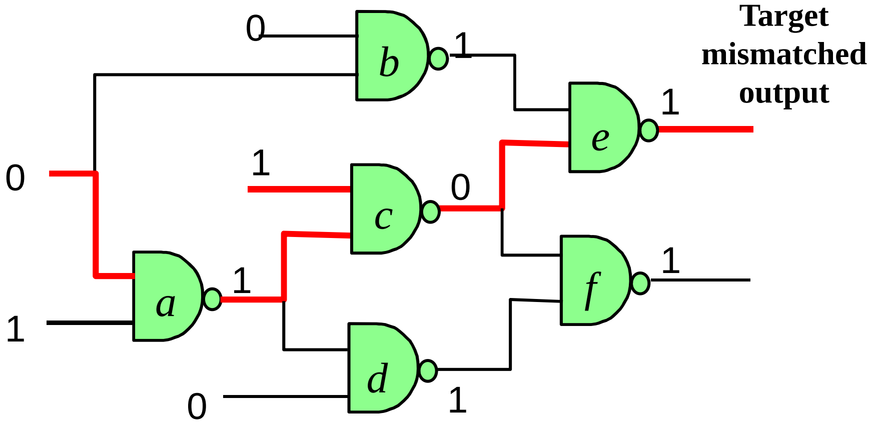
(b) Cone union when there are multiple faults.

Backtrace Algorithm

- **Trace back from each mismatched PO**
  - **To find out suspicious faulty locations**
- **Functional Pruning**
  - **During the traceback, some signals can be disqualified from the fault candidate set based on their signal values.**
- **Rules**
  - **(1) At a controlling case (i.e., 0 for a NAND gate): Its fanin signals with non-controlling values (i.e., 1) are excluded from the candidate set.**
  - **(2) At a non-controlling case (i.e., 1 for a NAND gate): Every fanin signal remains in the candidate set.**

Backtrace Example

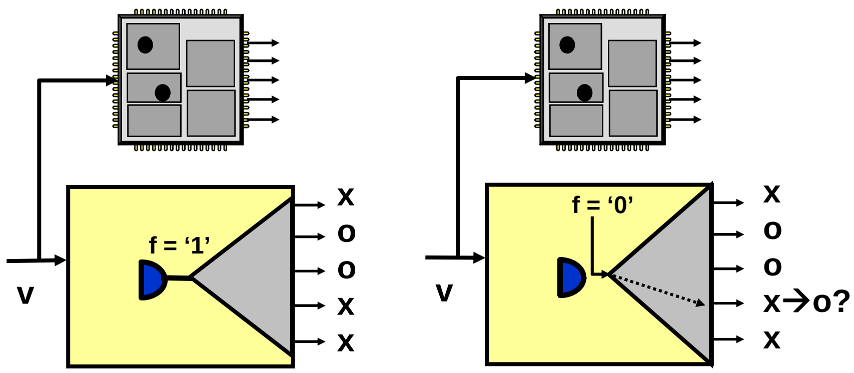
All suspicious fault locations are marked in red.



Ch11-25

Terminology – Injection

***An injection at a signal f** flips its current value*
*which could create **value-change events downstream**.*



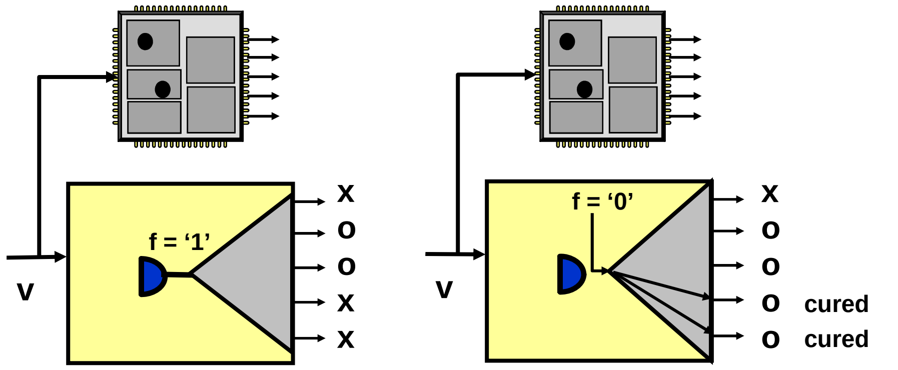
O: correct output
X: failing output

A mismatched output
could be fixed by the injection!

Ch11-26

Terminology – Curable Output

- Diagnosis Criterion
  - A signal is more suspicious if it has more curable outputs



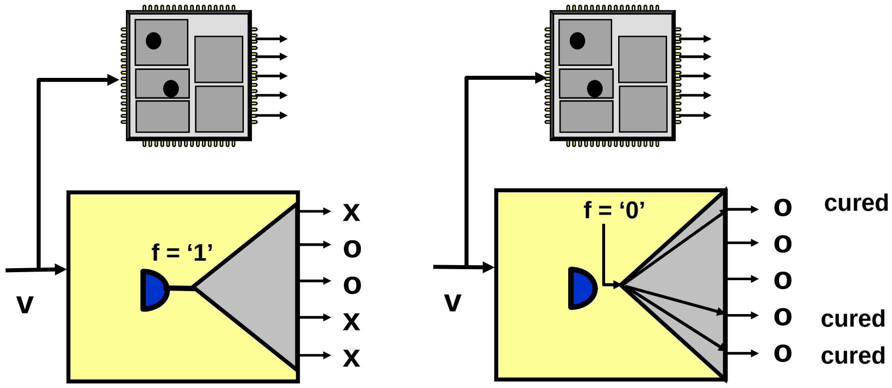
O: correct output
X: failing output

An injection at f fixes two mismatched outputs
➔ Thus, f has two curable outputs !

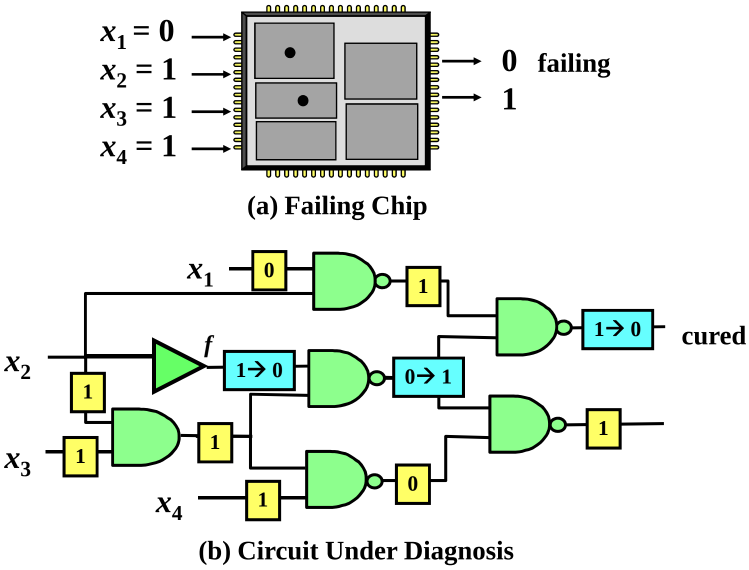
Terminology – Curable Vectors

v is a curable vector by f
➔ because an injection at f exists such that
it cures all mismatches without creating new one

Curable vector is a stronger diagnosis indicator than curable output !



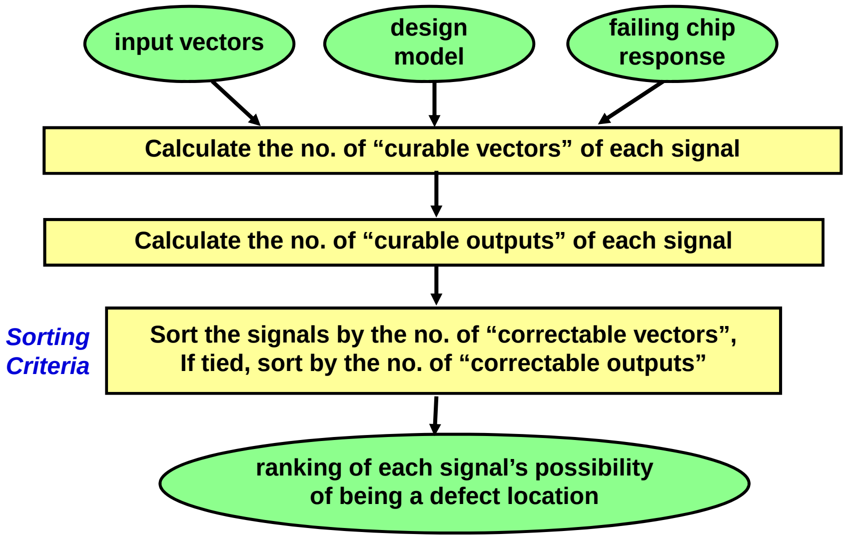
Example of Curable Vector



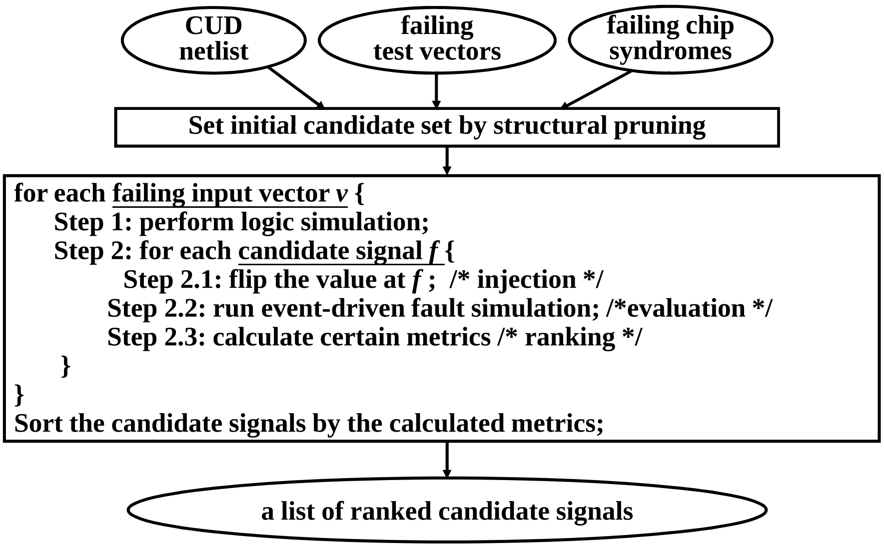
Why Curable Vector ?

- **Information theory**
  - A less probable event contains more information
  - Curable output is an easy-to-satisfy criterion, high aliasing
  - Curable vector is a hard-to-satisfy criterion, low aliasing
- **Not all failing input vectors are equal !**
- **Niche input vector**
  - Is an failing input vector that activates only one fault
  - Likely to be a curable vector of certain signals
  - Few, but tells more about the real fault locations

Inject-and-Evaluate Paradigm

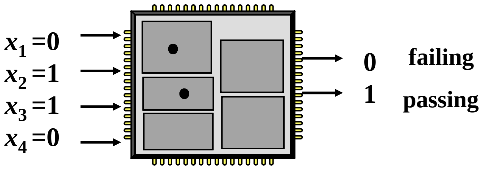


Detailed Computation – Inject-and-Evaluate Paradigm

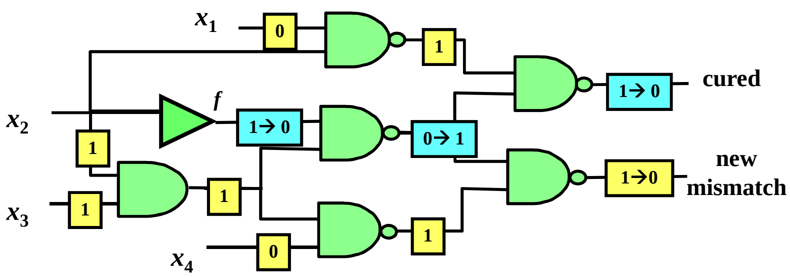


Reward-and-Penalty Heuristic

Rank1: curable vector count
Rank2 = (curable output count – 0.5 * new mismatched output count)



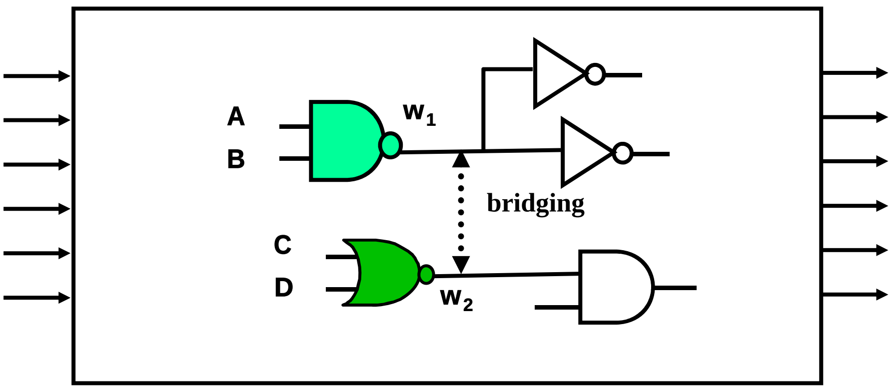
(a) Failing Chip.



(b) Circuit Under Diagnosis.

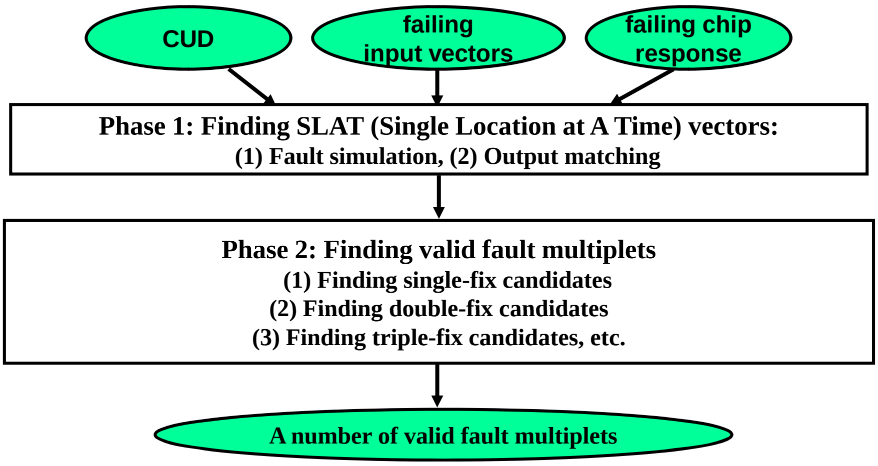
Targeting Bridging Faults

Even in a realistic bridging fault, there is only one victim at any time.
This victim will expose his location by owning some curable vectors.



SLAT Paradigm

Ref: ***SLAT*** **(Single Location At a Time) paradigm [Bartenstein 2001]**
Note: <u>A SLAT vector is a curable vector</u>



Example: SLAT Paradigm

Failing Input Vectors	Signals in the CUD						
	f_1	f_2	f_3	f_4	f_5	f_6	f_7
v_1	*				*		
v_2	*	*	*				
v_3			*	*			*
v_4					*	*	
v_5		*			*		
v_6		*			*		
v_7	*		*				
v_8			*				*
v_9			*		*		
v_{10}					*		*

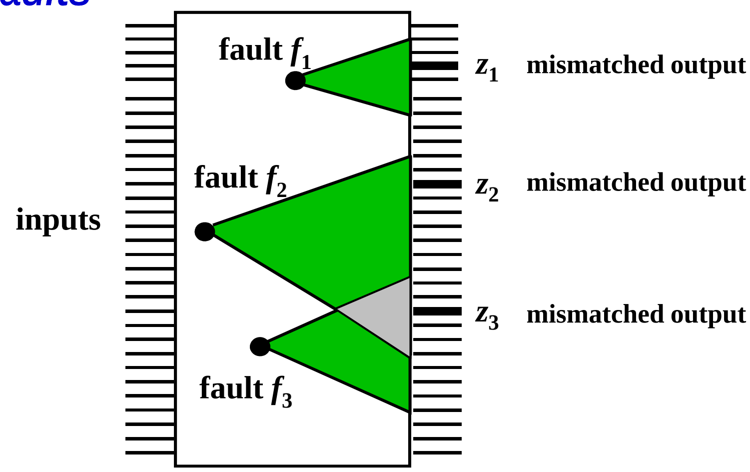
A mark * means the corresponding vector is a SLAT vector of the corresponding signal.

(f_3 and f_5) is a valid fault multiplet

Outline

- Introduction
- Combinational Logic Diagnosis
  - Cause-Effect Analysis
  - Effect-Cause Analysis
  - Chip-Level Strategy
  - Diagnostic Test Pattern Generation
- Scan Chain Diagnosis
- Logic BIST Diagnosis
- Conclusion

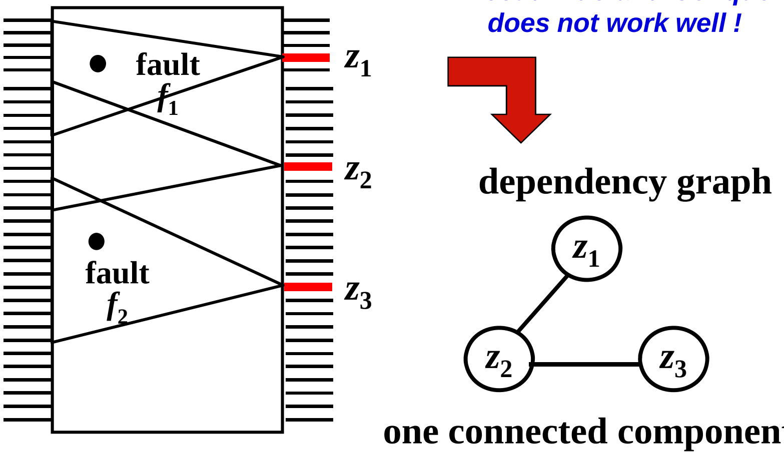
Structurally Dependent and Independent Faults



Fault f_1 is an independent fault.
Faults f_2 and f_3 are dependent faults.

Dependency Graph

Direct divide-and-Conquer does not work well !



one connected component

Two independent faults, f_1 and f_2, lead to one diagnosis block.

Ch11-39

Main Strategy: *Detach-Divide-and-then-Conquer*

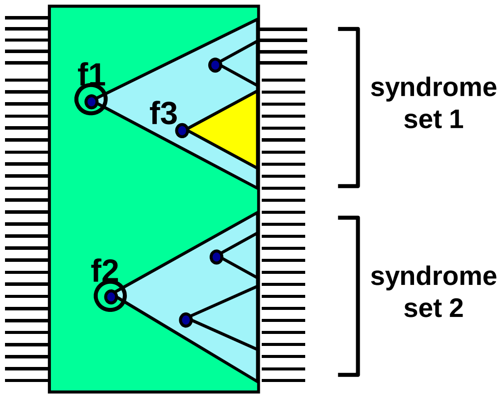
- Phase 1: Isolate Independent Faults
  - Search for prime candidates
  - Use word-level information
- Phase 2: Locate Dependent Faults As Well
  - Perform partitioning
  - Aim at finding one fault in each block

Ch11-40

Prime Candidates

A signal f is a prime candidate if
(1) All failing input vectors are partially curable by f
(2) Curable-Output-Set(f) is not covered by any other's

f1 & f2
are prime !

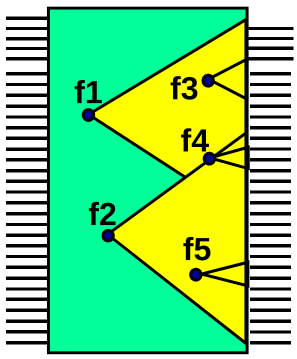


Ch11-41

Fake Prime Candidates

- ❑ **Structurally Independent Faults**
  - **are often prime candidates**
- ❑ **Fake Prime Candidates**
  - **are prime candidates that are NOT really faults - aliasing**

Example: Dependent Double Faults f1 & f2
May create fake prime candidates {f1, f2, f3}.



Ch11-42

Word-Level Registers and Outputs

Signals in a design are often defined in words.
This property can be used to differentiate fake prime candidates from the real ones.

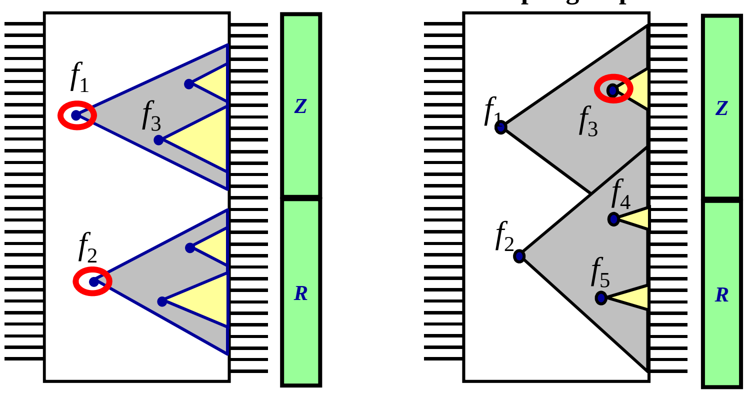
Word-Level Output: O1
Word-Level Registers: R1, R2, State

```
module      design( O1, ...)
   output[31:0]   O1;
   reg[31:0]      R1, R2;
   reg[5:0]       State
   ...
endmodule
```

Ch11-43

Word-Level Prime Candidates

Note: Z and R are two word-level output groups.



Original prime candidates: $\{f_1, f_2\}$
Word-level prime candidates $\{f_1, f_2\}$

Assumed original prime candidates: $\{f_3, f_4, f_5\}$
$\{f_4, f_5\}$ will be identified as fake
➔ *Final Word-level prime candidates $\{f_3\}$*

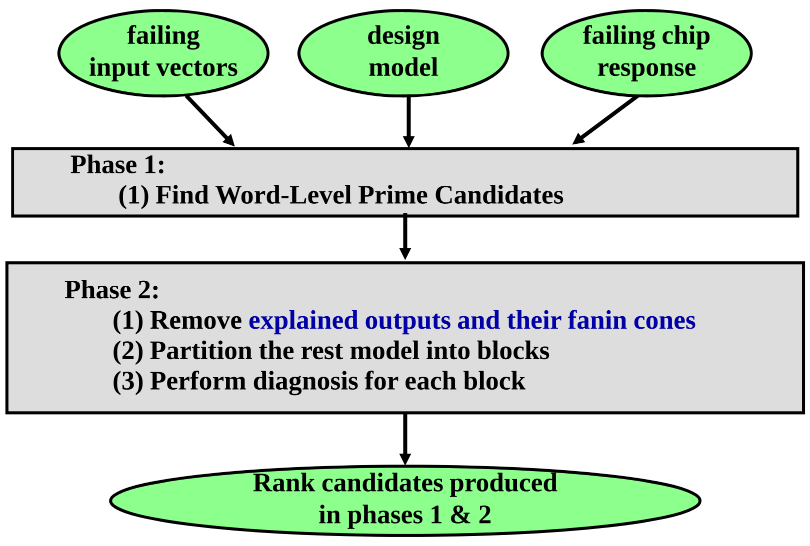
Ch11-44

Efficiency of Using Word-Level Info.

- Without word-level Information
  - 2.4 real faults out of 72.3 candidates
- With word-level Information
  - 1.23 real faults out of 3.65 candidates

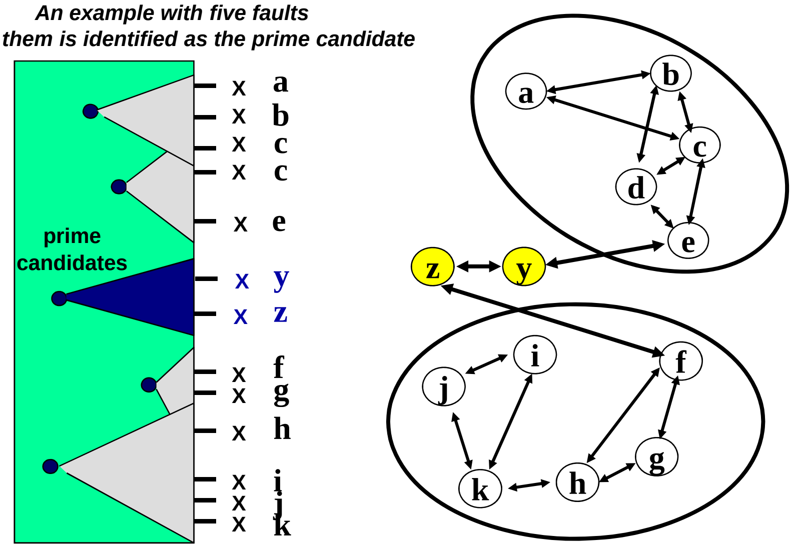
# of candidates	Original	After Filtering	Filtering Ratio
Prime Candidates	2.375	1.23	48.2 %
Fake Prime Candidates	69.96	2.42	96.5 %

Overall Flow



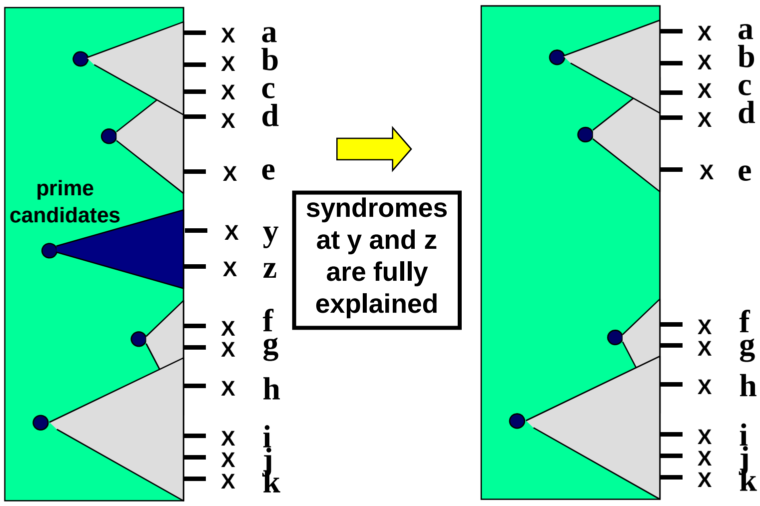
Grouping Using Dependency Graph

An example with five faults
One of them is identified as the prime candidate



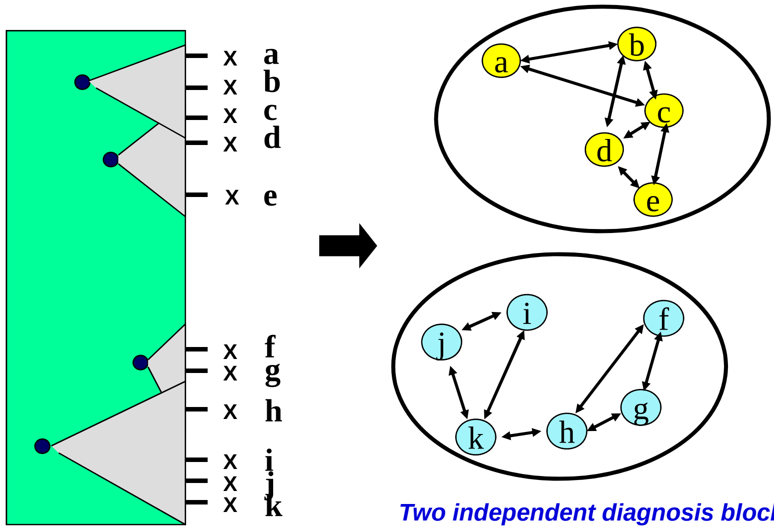
Ch11-47

Removed Explained Faulty Outputs



Ch11-48

Grouping Example



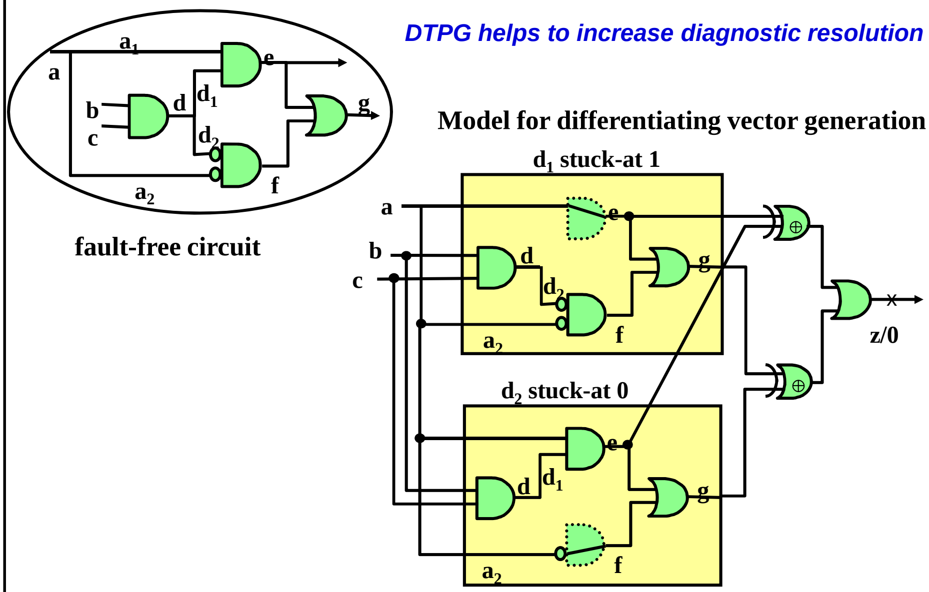
Two independent diagnosis blocks Are successfully derived!

Summary

- Strategy
  - (1) Search For Word-Level Prime Candidates
  - (2) Identify Independent Faults First
  - (3) Locate Dependent Faults As Well
- Effectiveness
  - identify 2.98 faults in 5 signal inspections
  - find 3.8 faults in 10 signal inspections

Diagnostic Test Pattern Generation

DTPG helps to increase diagnostic resolution



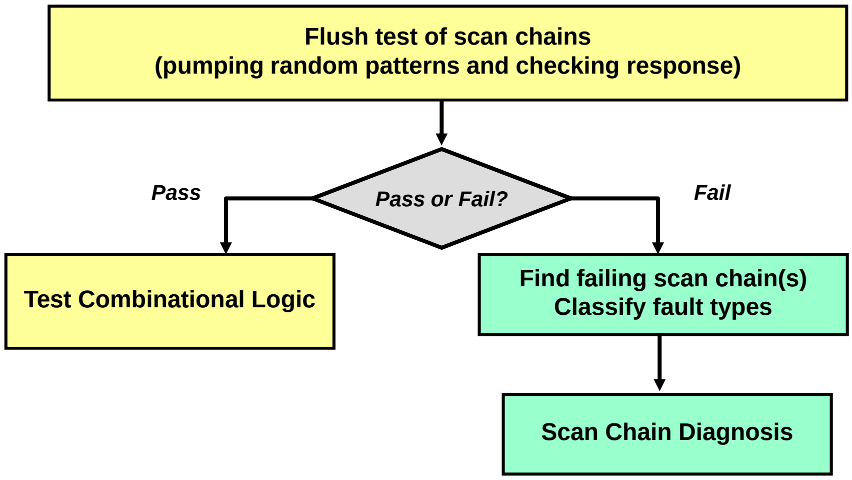
Ch11-51

Outline

- Introduction
- Combinational Logic Diagnosis
- **Scan Chain Diagnosis**
  - Preliminaries
  - Hardware-Assisted Method
  - Signal-Profiling Based Method
- Logic BIST Diagnosis
- Conclusion

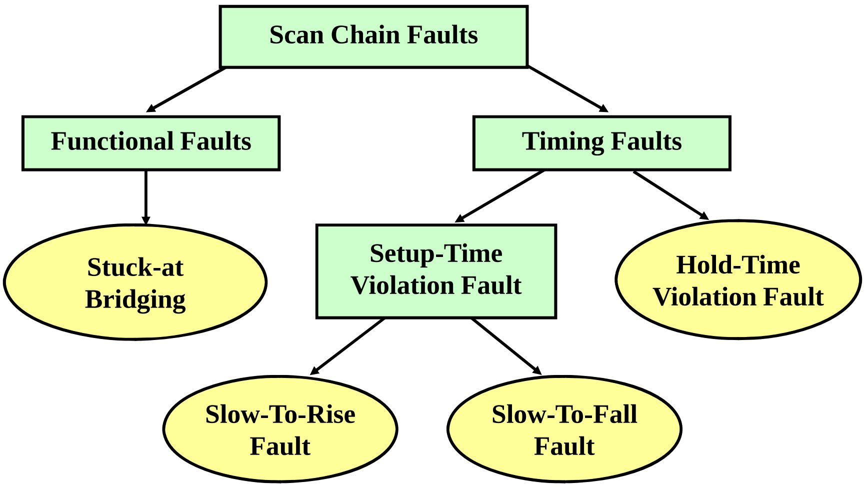
Ch11-52

Scan Test and Diagnosis



Ch11-53

Commonly Used Fault Types in Scan Chains

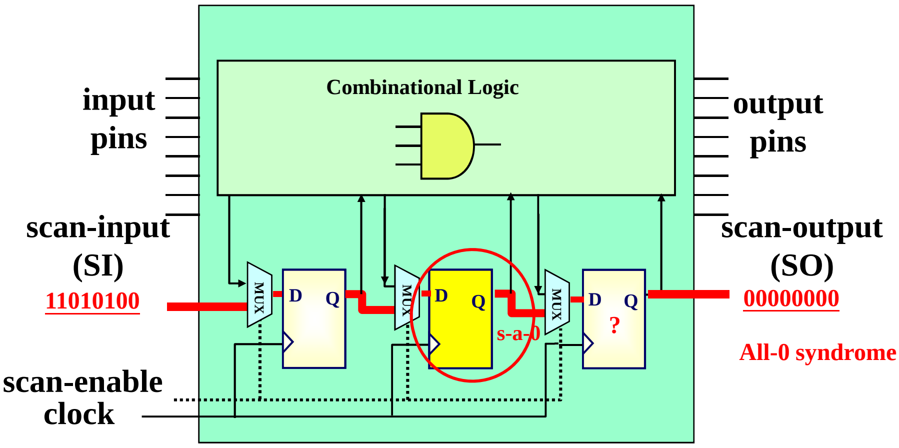


Each fault could be permanent or intermittent.

Ch11-54

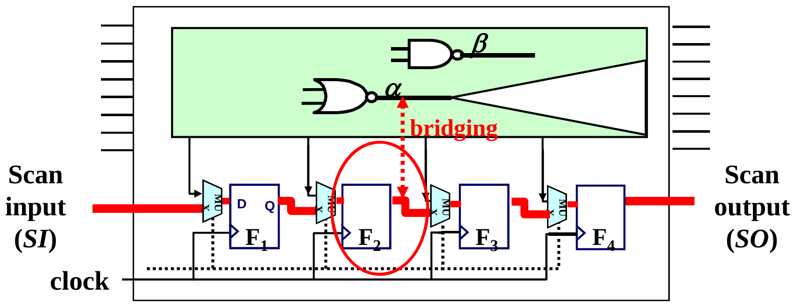
A Stuck-At Fault In the Chain

Effect: A killer of the scan-test sequence

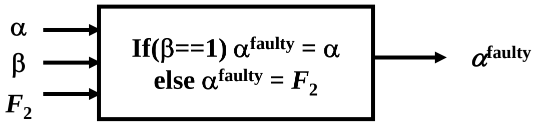


Ch11-55

A Realistic Bridging Fault Model



(a) Bridging between a flip-flop and a logic cell.

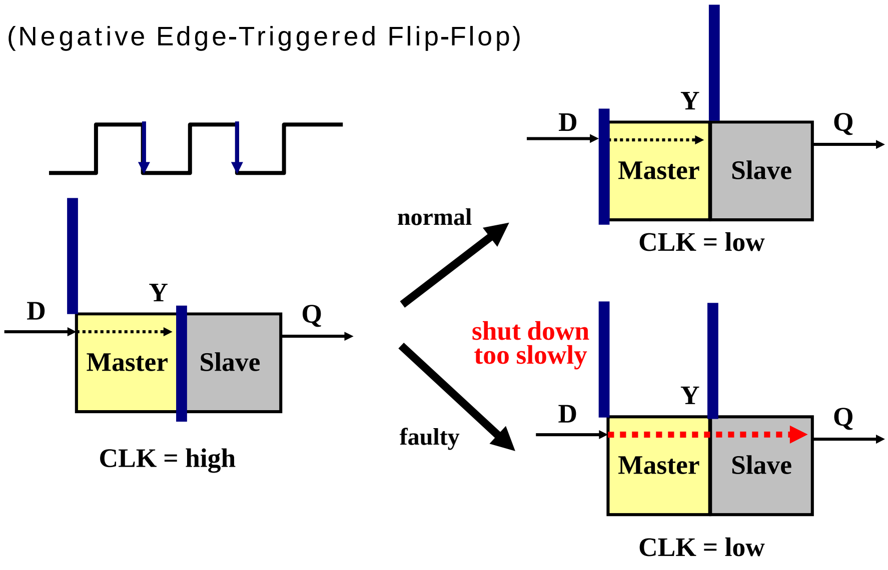


(b) Our bridging fault model.

Ch11-56

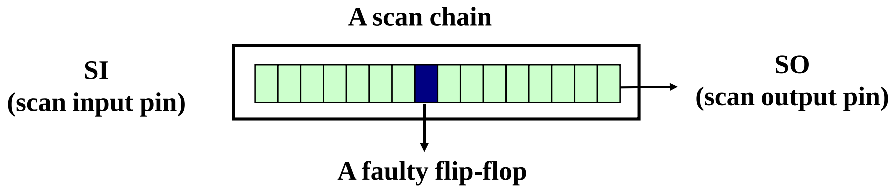
Potential Hold-Time Fault?

(Negative Edge-Triggered Flip-Flop)



Ch11-57

Example: Faulty Syndrome of a Scan Chain



Fault Type	Scan-In Pattern	Observed Syndrome
Stuck-at-0	1100110011001100	0000000000000000
Stuck-at-1	1100110011001100	1111111111111111
Slow-to-Rise	1100110011001100	1000100010001000
Slow-to-Fall	1100110011001100	1101110111011100

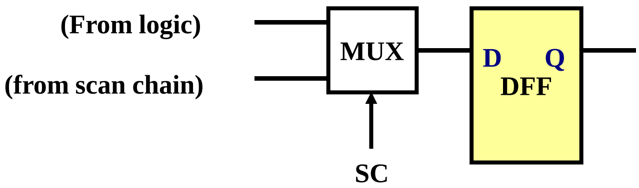
The rightmost bit goes into the scan first

The rightmost bit gets out of the scan first

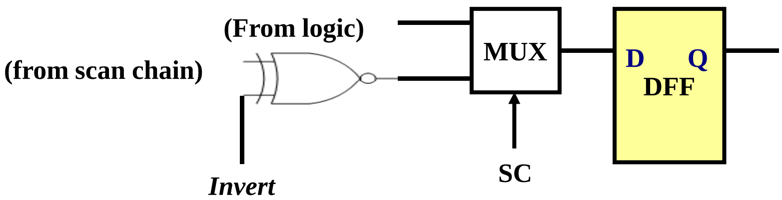
A underlined bit in the observed image is failing.

Ch11-58

Augmentation of a Flip-Flop for Easy Diagnosis

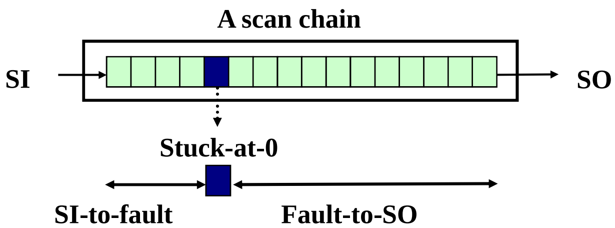


(a) A normal scan flip-flop.



(b) A modified scan flip-flop for easy inversion.

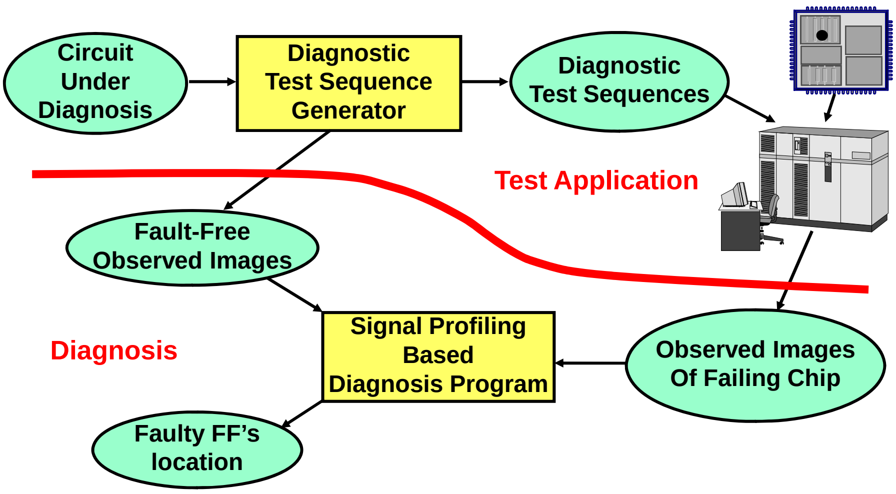
Fault Location via Inversion Operation



(1) **Original bitstream pattern = (1111111111111111)**
(2) **After scan-in: snapshot image = (1111000000000000)**
(3) **After inversion: snapshot image = (0000011111111111)**
(4) **After scan-out: observed image = (0000011111111111)**

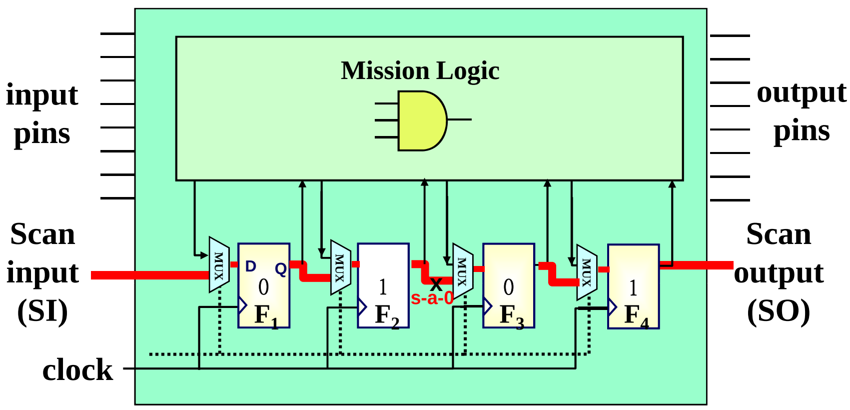
The fault location is at the edge between 0's and 1's

Scan Chain Diagnosis Flow



Definition: Snapshot Image

Def: A snapshot image is the combination of flip-flop values at certain time instance

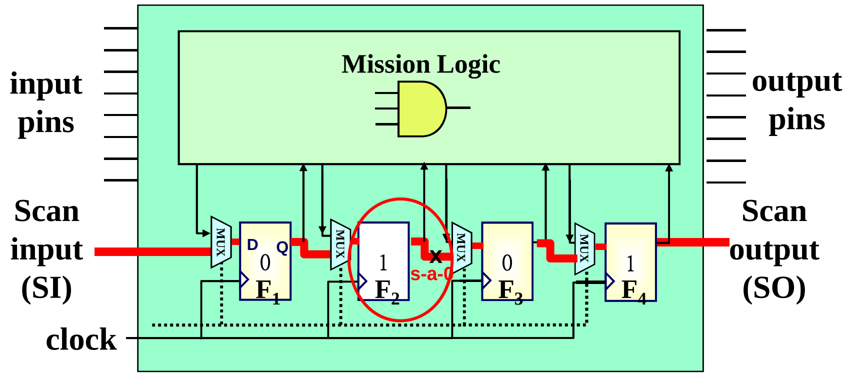


Snapshot image: $\{(F_1, F_2, F_3, F_4) \mid (0, 1, 0, 1)\}$

Ch11-62

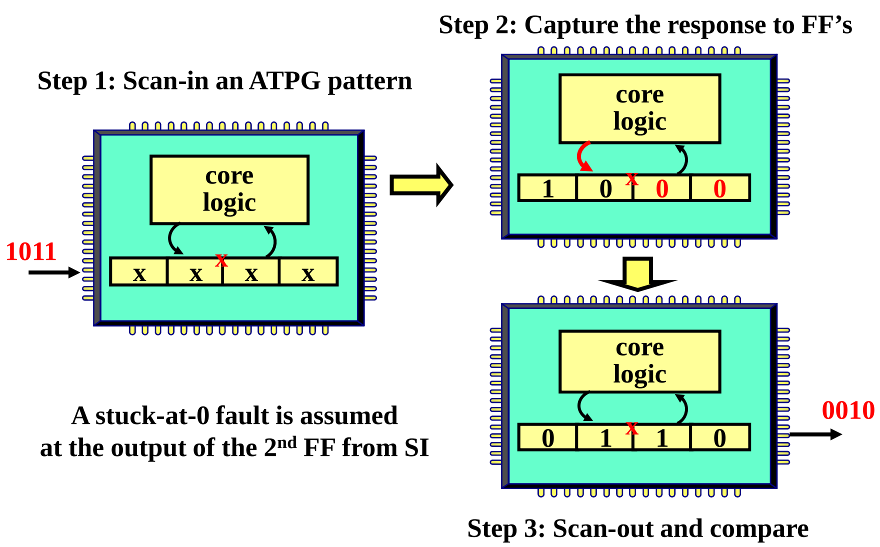
Definition: Observed Image

Def: An observed image is the scanned-out version of a snapshot image.



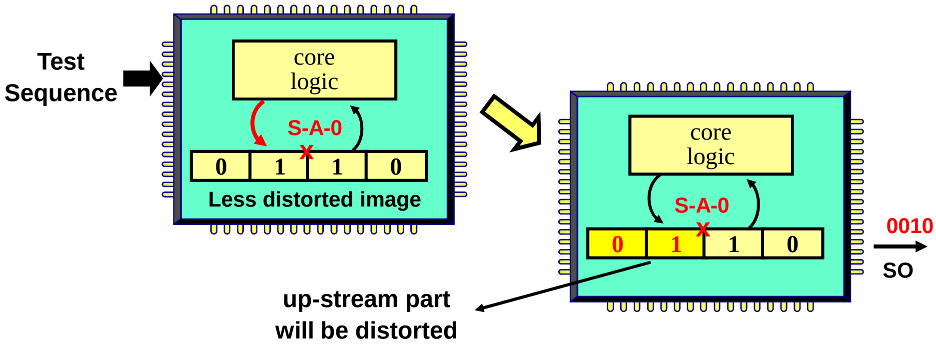
Snapshot image: {(F_1, F_2, F_3, F_4) | (0, 1, 0, 1)}
Observed image: {(F_1, F_2, F_3, F_4) | (0, 0, 0, 1)}

Modified Inject-and-Evaluate Paradigm



Test Application: Run-and-Scan

Step 1: Apply a test sequence from PI's
➔ Setting up a snapshot image at FF's



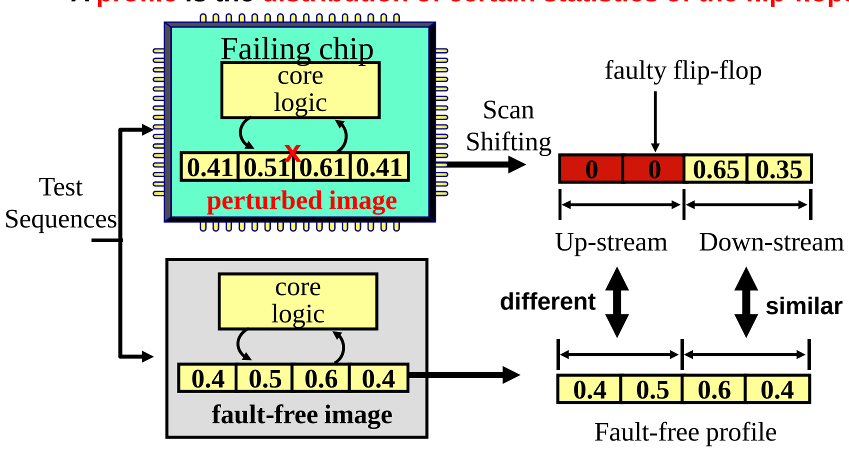
Step 2: Scan-out an observed image

The fault location is embedded in the observed image

Ch11-65

Signal Profiling

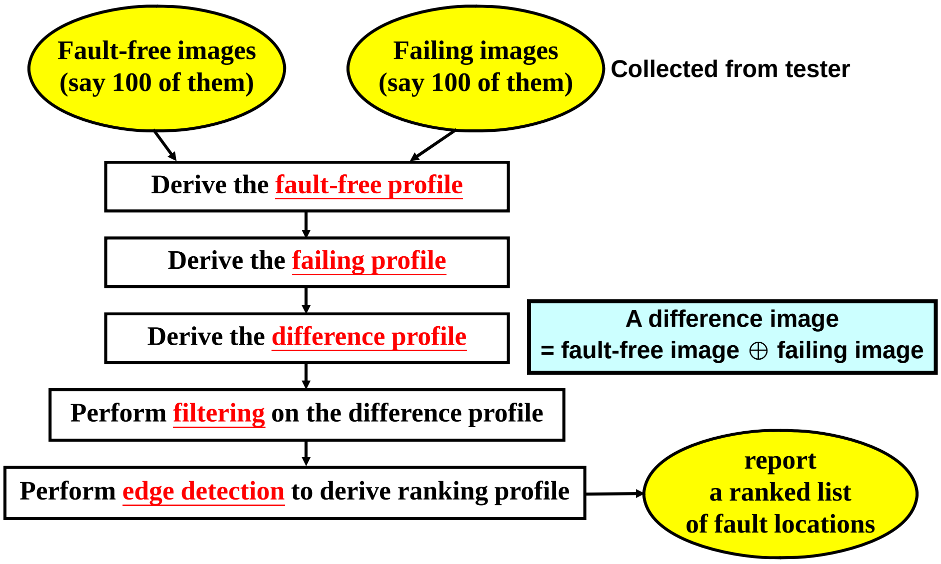
A profile is the distribution of certain statistics of the flip-flops.



Comparing failing profile with the fault-free profile
➔ Could reveal the fault location

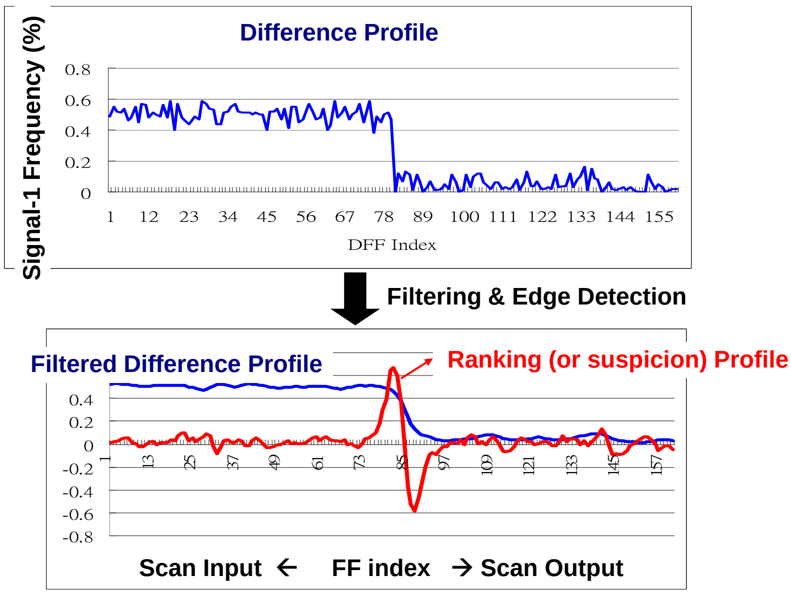
Ch11-66

Profile Analysis



Ch11-67

Example: Filtering & Edge Detection



Computation of Average-Sum Filtering

- **(Average-sum filtering) Assume that the difference profile is given and denoted as *D*[i], where i is the index of a flip-flop. We use the following formula to compute a smoothed difference profile, *SD*[i]:**

$$SD[i] = 0.2*(D[i-2]+ D[i-1]+ D[i]+ D[i+1]+ D[i+2])$$

Computation of Edge Detection

- **The true location of the faulty flip-flop is likely to be the *left-boundary of the transition region in the difference profile*. To detect this boundary, we can use a simply *edge detection formula* defined below.**
- **(Edge detection) On the smoothed difference profile *SD*[i], the following formula can be used to compute the faulty frequency of each flip-flop as a suspicious profile.**

$$suspicion[i] = [-1,-1,-1,1,1,1] \cdot \begin{bmatrix} |SD[i]-SD[i-3]| \\ |SD[i]-SD[i-2]| \\ |SD[i]-SD[i-1]| \\ SD[i]-SD[i+1] \\ SD[i]-SD[i+2] \\ SD[i]-SD[i+3] \end{bmatrix}$$

Summary of Scan Chain Diagnosis

- Hardware Assisted
  - Extra logic on the scan chain
  - Good for stuck-at fault
- Fault Simulation Based
  - To find a faulty circuit matching the syndromes [Kundu 1993] [Cheney 2000] [Stanley 2000]
  - Tightening heuristic → upper & lower bound [Guo 2001][Y. Huang 2005]
  - Use single-excitation pattern for better resolution [Li 2005]
- Profiling-Based Method
  - Locate the fault directly from the difference profiles obtained by run-and-scan test
  - Applicable to bridging faults
  - Use signal processing techniques such as filtering and edge detection

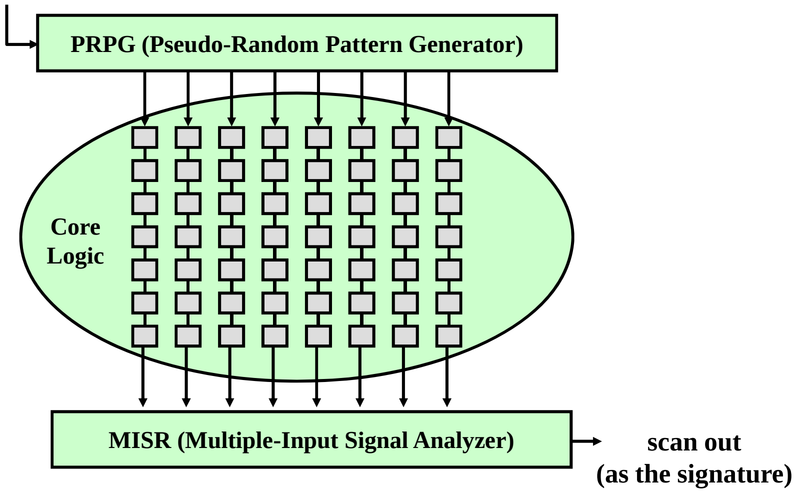
Ch11-71

Outline

- Introduction
- Combinational Logic Diagnosis
- Scan Chain Diagnosis
- ➡ Logic BIST Diagnosis
  - Overview
  - Interval-Based Method
  - Masking-Based Method
- Conclusion

Ch11-72

A Logic BIST Architecture

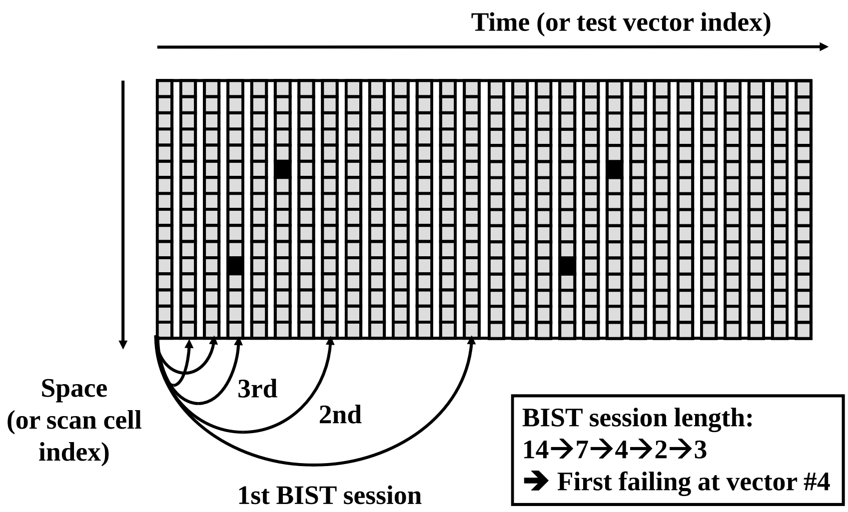


All flip-flops are assumed to be observable through scan chains.

Diagnosis for BISTed Logic

- Diagnosis in a BIST environment requires
  - determining from compacted output responses which test vectors have produced a faulty response (*time information*)
  - determining from compacted output responses which scan cells have captured errors (*space information*)
- The true fault location inside the logic
  - Can then be inferred from the above space and time information using previously discussed combinational logic diagnosis

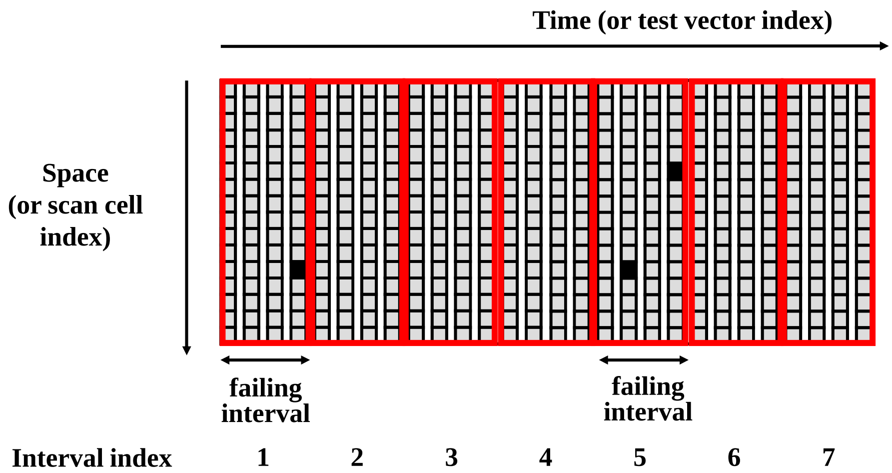
Binary Search To Locate 1st Failing Vector



BIST session length:
14→7→4→2→3
➔ First failing at vector #4

Ch11-75

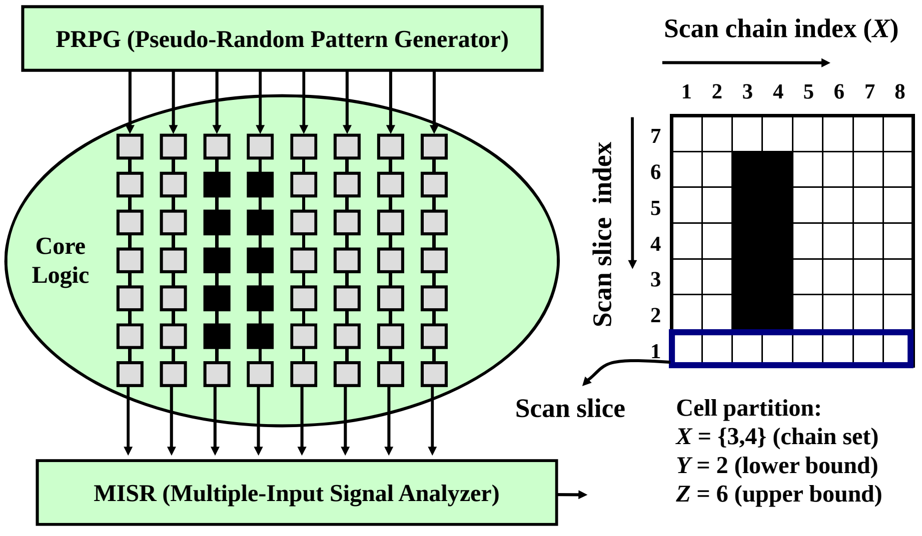
Interval Unloading-Based Diagnosis



A signature is scanned out to the tester
for comparison at the end of each interval

Ch11-76

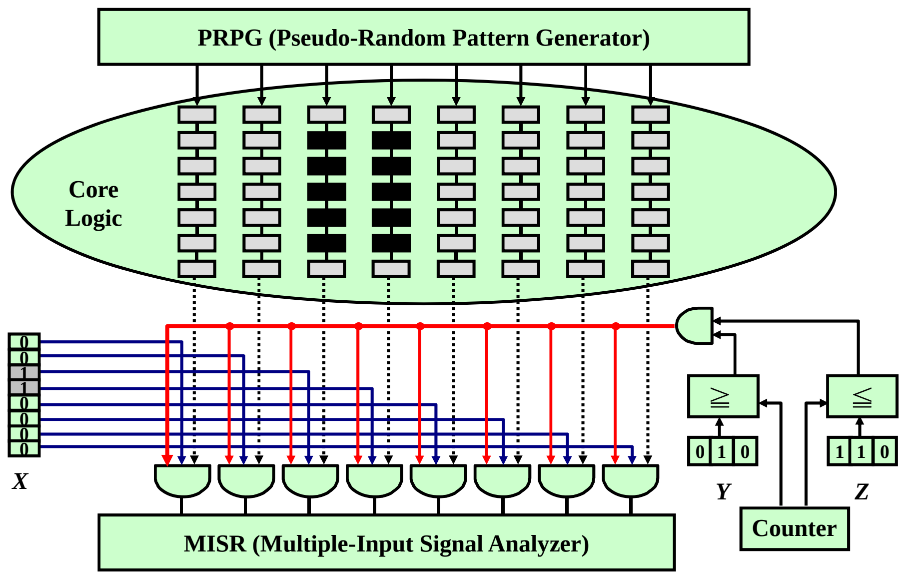
Deterministic Masking-Based Diagnosis



(a) STUMP-based BIST architecture (b) Scan cell matrix

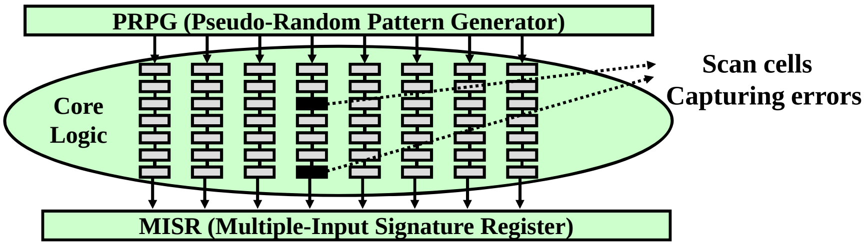
Ch11-77

Circuitry to Support Deterministic Masking

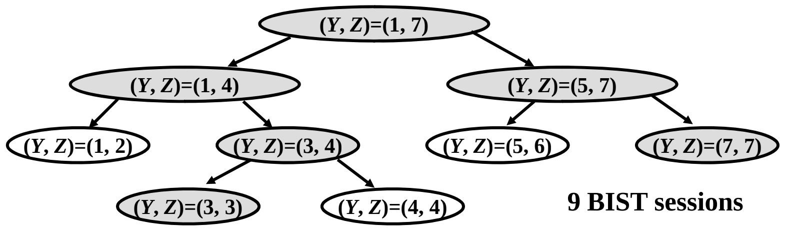


Ch11-78

A Search for Scan Cells Capturing Errors



(a) Scan cells capturing errors in the fourth scan chain



(b) The search tree

Conclusions

- **Logic diagnosis for combinational logic**
  - **Has been mature**
  - **Good for not just stuck-at faults, but also bridging faults**
- **Scan chain diagnosis**
  - **Making good progress …**
  - **Fault-simulation-based, or signal-profiling based**
- **Diagnosis of scan-based logic BIST**
  - **Hardware support is often required**
  - **Interval-unloading, or masking-based**
- **Future challenges**
  - **Performance (speed) debug**
  - **Diagnosis for logic with on-chip test compression and decompression**
  - **Diagnosis for parametric yield loss due to nanometer effects**