# VLSI Testing
積體電路測試

Fault Collapsing

Professor James Chien-Mo James Li 李建模

Lab. of Dependable Systems (LaDS)

Graduate Institute of Electronics Engineering
National Taiwan University

Course Roadmap (EDA Topics)

Logic Sim.

Fault Modeling / Collapsing

Fault Simulation

Testability

Comb. ATPG

Diagnosis

Seq. ATPG

Delay Test

2

Test 4.1 © National Taiwan University

Why Am I Learning This?

- **Fault collapsing reduce number of faults**
  - ♦ **Speed up fault simulation and ATPG**
  - ♦ **Reduce Test Patterns**

"Simplicity is the ultimate sophistication."
*(Leonardo da Vinci)*

3
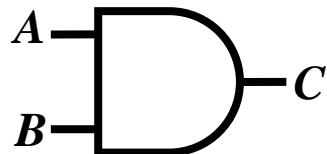
Fault Collapsing

- **Introduction**
- **Equivalence Fault Collapsing**
  - ◆ **Fanout-free circuits**
  - ◆ **Circuits with fanout**
- **Dominance Fault Collapsing**
  - ◆ **Fanout-free circuits**
  - ◆ **Circuits with fanout**
- **Discussions**
- **Conclusion**

4

Introduction

- **Why Fault Collapsing?**
  - **Reduce number of faults so that**
    - **Speed up ATPG**
    - **Shorten test set**
- **How?**
  - **Quickly find *representative faults***
  - **Use only those representative faults for ATPG**

- **Requirement of fault collapsing algorithm**
  - **Must be fast; otherwise, it is not worth doing**

- **There are many different ways to do fault collapsing**
  - **Obtaining the *minimum* solution is not so important**
  - **As long as we do not miss any fault**

5
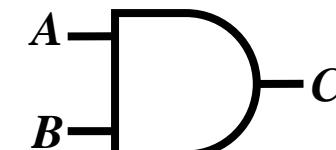
Equivalent Faults

- **Faults *f* and *g* are *functionally equivalent* (or simply *equivalent*)**
  - ♦ **if faulty outputs of them are identical for *all* test patterns**
- **Example: *A/0 B/0 and C/0* are all *equivalent faults***
  - ♦ **They belong to the same *equivalent fault class***

Input		Output						
A	*B*	good	*A/0*	*C/0*	*B/0*	*A/1*	*C/1*	*B/1*
0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0
1	0	0	0	0	0	0	1	1
1	1	1	0	0	0	1	1	1

**6**

Properties of Equivalence
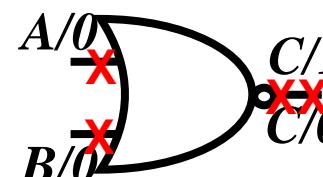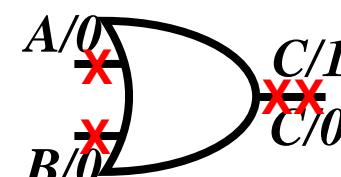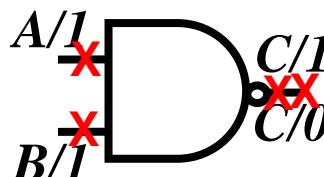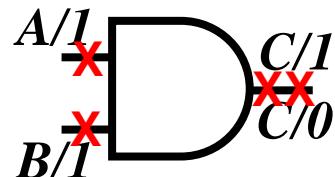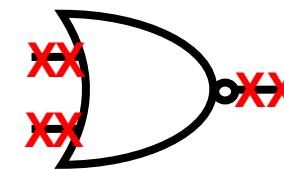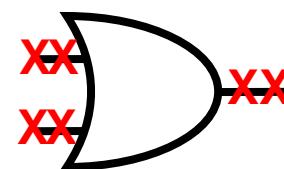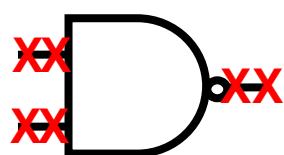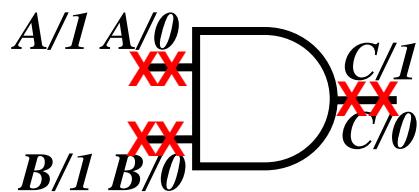
- **Equivalence relationship is *symmetric***
  - ◆ **if fault *f* is equivalent to fault *g*, then fault *g* is equivalent to fault *f***
- **Equivalence relationship is also *transitive***
  - ◆ **if fault *f* is equivalent to fault *g* and fault *g* is equivalent to fault *h*, then fault *f* is equivalent to fault *h***
- **Example**
  - ◆ **A/0 fault is equivalent to B/0 fault, which is also equivalent to C/0 fault. These three faults belong to the same *equivalence class***

Input		Output						
A	*B*	good	*A/0*	*C/0*	*B/0*	*A/1*	*C/1*	*B/1*
0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0
1	0	0	0	0	0	0	1	1
1	1	1	0	0	0	1	1	1

Equivalence Fault Collapsing, EFC

- **EFC reduces fault list using fault equivalence relationship**
  - ♦ **Select one representative fault from every equivalent class**
- **Example:**
  - ♦ **Originally six faults per 2-input gate**
  - ♦ **After EFC, only four faults per gate**
  - ♦ **6→4**

- **NOTE: EFC is not unique**
 - ♦ **{A/0, A/1, B/1, C/1} is also EFC for AND gate**

8
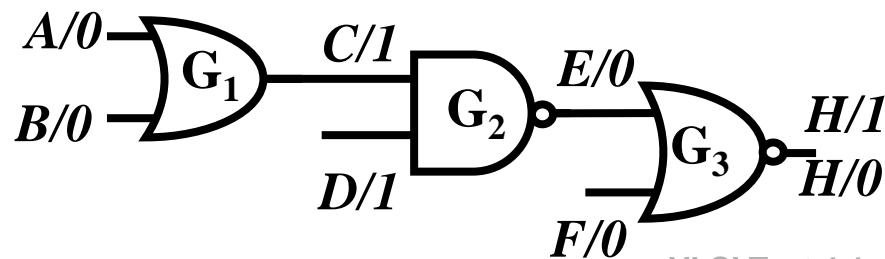
How to Perform EFC?

- **EFC can be performed by *functional analysis* or *structure analysis***
- ***Functional analysis***
  - ◆ **Apply *all possible* test patterns and fault simulate two faults**
    - ∗ **If same faulty outputs , two faults are equivalent**
  - ◆ **Exhaustive functional time consuming**
    - ∗ **2^n test patterns may be needed for an *n*-input circuit**
- ***Structure analysis***
  - ◆ **Identify equivalent faults by circuit structure**
  - ◆ **No test patterns applied**
- **Many different structure analysis approaches**
  - ◆ **In this lecture, we show a linear time structure analysis example**
  - ◆ **Resulting equivalence collapsed fault list may not be minimal**
    - ∗ **but good enough for most applications**

Structure Analysis Is Enough for Most Applications

9

EFC on Fanout-free Circuits

- **Our EFC Rules**
  - ♦ **(1) both stuck-at one and zero faults** for every primary output
  - ♦ **(2) one collapsed fault** for each gate input
    - ∗ **stuck-at** *non-controlling* **values (see slide 8)**
- **Example**
  - ♦ **keep both H/0 and H/1 faults for primary output H**
  - ♦ **keep one fault for each gate input,**
    - ∗ **A/0 and B/0 for OR gate G1 and etc**
  - ♦ **Original 14 faults → 8 faults after EFC**
  - ♦ **Why faults on the gate outputs are removed?**
    - ∗ **gate G1 output stuck-at zero fault is equivalent to C/0 fault, which is equivalent to E/1, which is equivalent to H/0.**
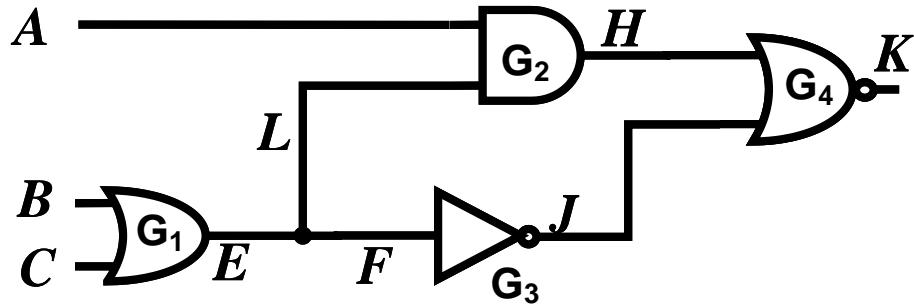
EFC on Circuits with Fanouts

- **Fanout stem faults are NOT always equivalent to fanout branch faults**
- **Example:**
  - ♦ **E/0 is equivalent to F/0**
    - ∗ **but not equivalent to L/0**
  - ♦ **The other faults are NOT equivalent**

This functional analysis is slow

| Input			Output						
A	*B*	*C*	good	*E/0*	*F/0*	*L/0*	*E/1*	*F/1*	*L/1*
0	0	0	0	0	0	0	1	1	0
0	0	1	1	0	0	1	1	1	1
0	1	0	1	0	0	1	1	1	1
0	1	1	1	0	0	1	1	1	1
1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	1	0	0	0
1	1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	1	0	0	0

11
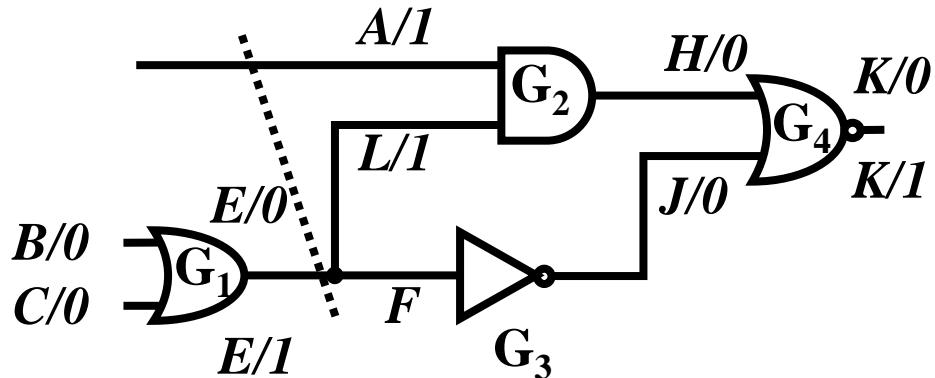
How to Handle Circuits w/ Fanouts?

- ***Stem analysis*** determines equivalent faults on fanout stem and its branches
    - ♦ However, stem analysis is time consuming
        - ∗ not worth the time
        - ∗ details skipped in this lecture

- An approximate solution
    - ♦ Partition circuit into independent *fanout-free subcircuits*
    - ♦ Every fanout stem treated as a primary output
        - ∗ both s@1 s@0 faults are included on branches
        - ∗ They are *NOT* collapsed

Treat each FFS partition independently

12

Example

- **2 partitions**
- **Originally 18 faults, → after EFC 10 faults**

- **NOTE:**
  - **Inverter G$_3$ ignored**
  - **because its input fault s@0 is always equivalent to its output s@1 fault**

Simple_EFC Algorithm
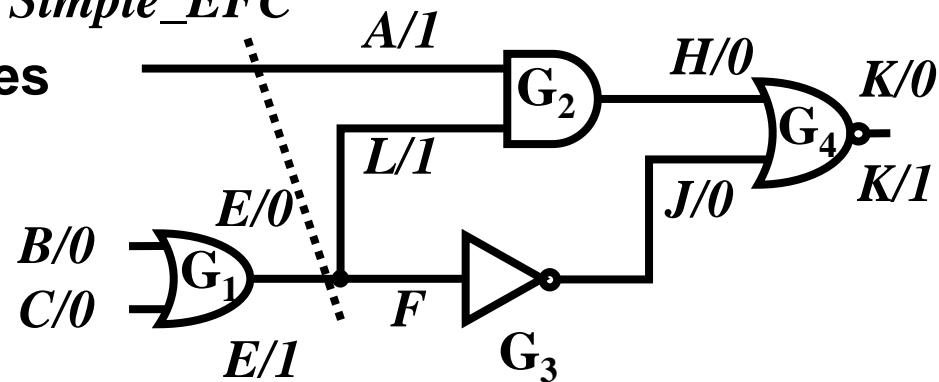
- **Linear time (NOT unique; other EFC algorithm is possible)**

Simple_EFC (*N*)   /*N is a netlist*/
0. fault_list = { };
1. **foreach** gate or PO or PI *g* in *N*
2.   **if** ((*g* is PO) || (g is PI and fanout stem)) **then**
3.   fault_list = fault_list ∪ *g* stuck-at 0 and stuck-at 1;
4.   **else if** (output of gate *g* is fanout stem) **then**
5.   fault_list = fault_list ∪ *g* output stuck-at 0 and 1;
6.   **end if**
7.   **if** (gate *g* is AND) || (gate *g* is NAND) **then**
8.   fault_list = fault_list ∪ *g* input stuck-at 1;
9.   **else if** (gate *g* is OR) || (gate *g* is NOR) **then**
10.   fault_list = fault_list ∪ *g* input stuck-at 0;
11.   **end if**
12. **end foreach**
13. **return** (fault_list);

*This structural analysis is fast.

14

# Problems of *Simple_EFC* (1)
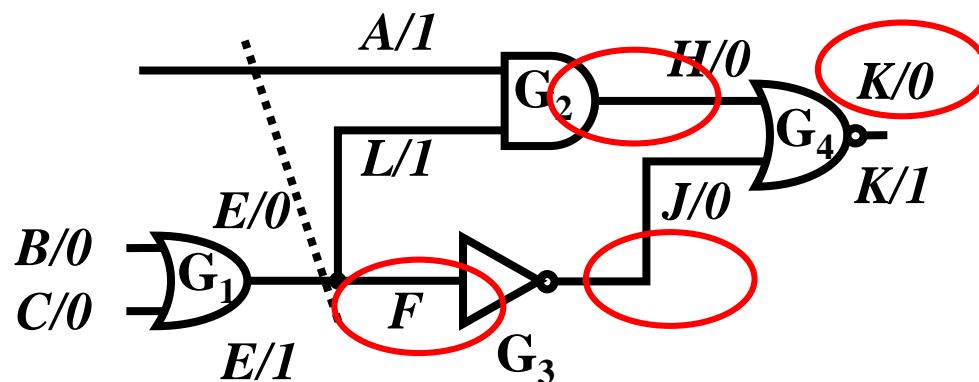
- **Does *NOT* guarantee optimal result because it lacks stem analysis**
  - ♦ **For example, E/0 fault actually equivalent to K/0 fault**
    - ∗ **but they both appear after *Simple_EFC***
  - ♦ **Often acceptable in most cases**

Input			Output						
A	*B*	*C*	good	*E/0*	*F/0*	*L/0*	*E/1*	*F/1*	*L/1*
0	0	0	0	0	0	0	1	1	0
0	0	1	1	0	0	1	1	1	1
0	1	0	1	0	0	1	1	1	1
0	1	1	1	0	0	1	1	1	1
1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	1	0	0	0
1	1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	1	0	0	0

15
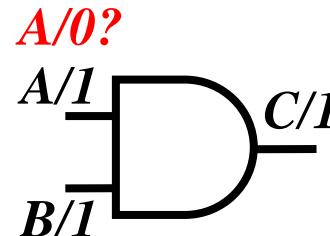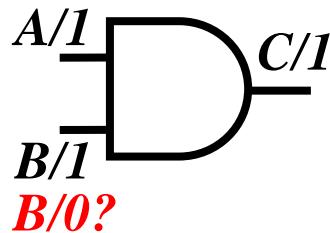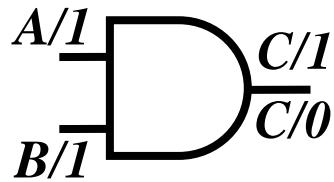
Problems of *Simple_EFC* (2)

- **Link** between original (uncollapsed) faults and their corresponding collapsed faults is **lost**
- **Example:**
  - ♦ **No link between four faults {F/0, J/1, H/1, K/0} in same equivalence class and their collapsed fault K/0**
- **Links between uncollapsed faults and collapsed faults are needed when calculating the *uncollapsed fault coverage***
  - ♦ **Example: suppose we detect A/1 K/0 and B/0 faults**
    - ∗ **Collapsed fault coverage is 3/10 = 30%**
   - ∗ **But uncollapsed fault coverage is ?/18 = ?**

FFT

- **Q1: Why did we choose C/0?**
  - ◆ **why not A/0 ? B/0?**

- **Q2: How do we modify the *Simple_EFC* algorithm to calculate number of faults in each equivalence class?**
  - ◆ *i.e.* **how do we know there are 4 faults in K/0 equivalence class?**