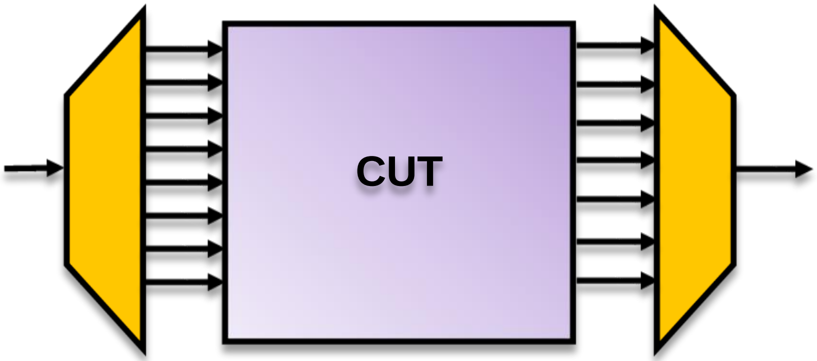
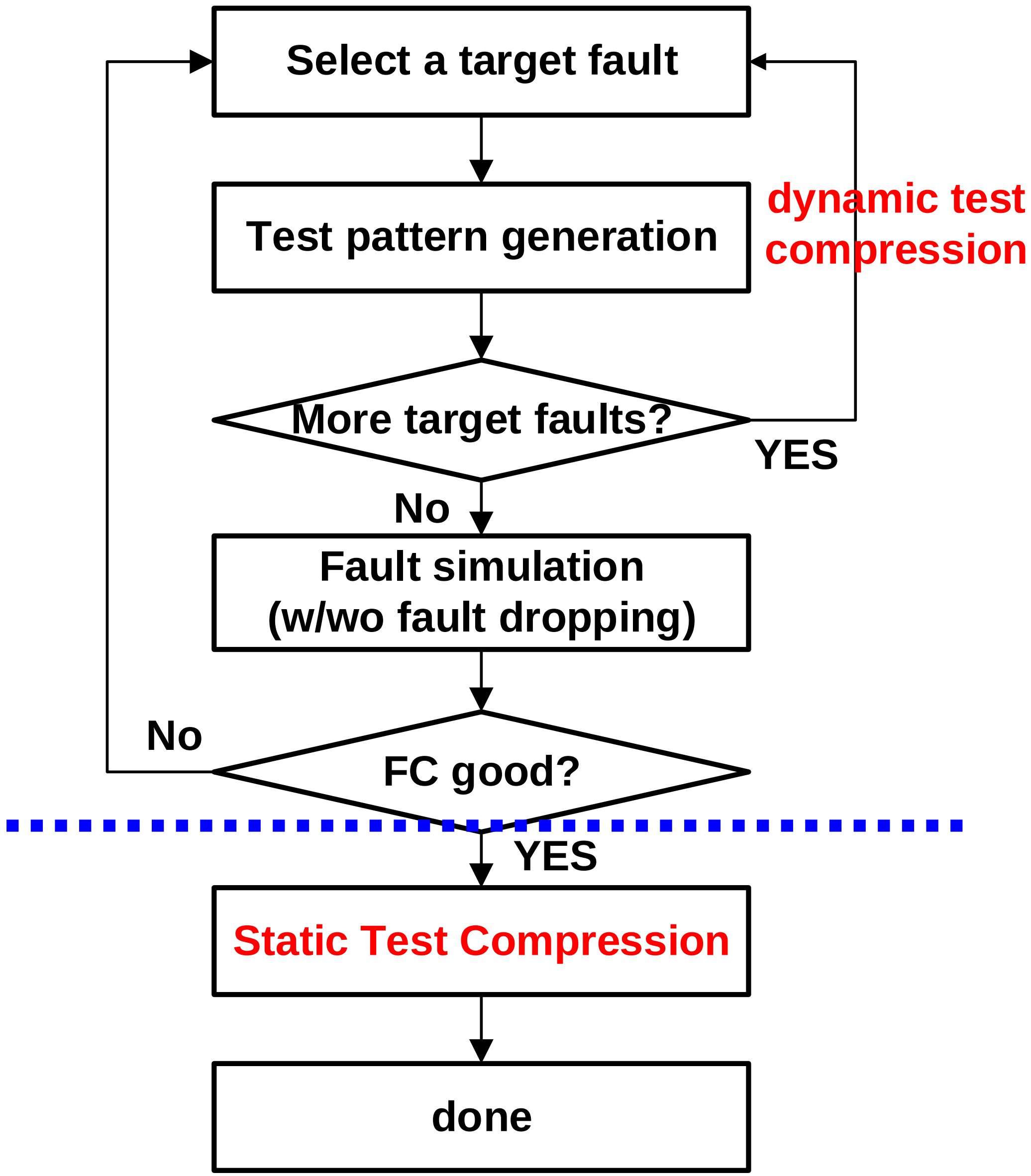
Test Compression

- Introduction
- **Software Techniques**
  - Dynamic Test Compression (DTC)
  - **Static Test Compression (STC)**
- Hardware Techniques
  - Test Stimulus Compression
  - Test Response Compression
  - Industry Practices
- Conclusion



Review: STC vs. DTC

- *Dynamic test compression*
  - performed during TPG
  - more CPU time
  - more effective
- *Static test compression*
  - performed after TPG
  - less CPU time
  - less effective

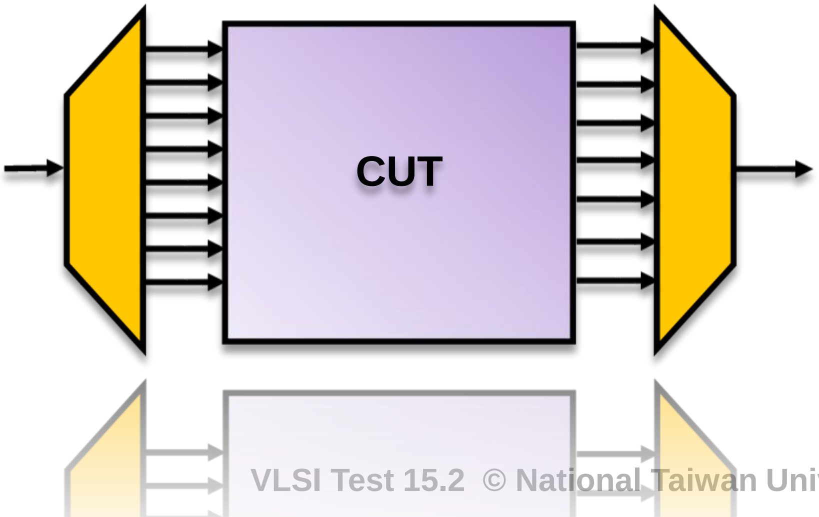


Test Compression

- Introduction
- <u>Software Techniques</u>
    - Dynamic Test Compression
    - <u>Static Test Compression</u>
        - <u>With fault dictionary</u>
        - Without fault dictionary
            - Compatibility graph (X-unfilled)
            - Reverse order fault simulation (X-filled)

3 cases

- Hardware Techniques
    - Test Stimulus Compression
    - Test Response Compression
    - Industry Practices
- Conclusion



STC with Fault Dictionary

- Suppose we have a fault dictionary (without fault dropping)
- *Covering table*
  - Each row is a test pattern, each column is a fault
- Goal: Find minimum test set (select fewest test patterns)
  - Detect all faults
- Finding minimum test set is *minimum set covering problem*
  - NP-hard, but don't give up…

	f_1	f_2	f_3	f_4	f_5
t_1				X	
t_2		X	X		X
t_3		X		X	X
t_4	X			X	

Each row is a pattern
Each column is a fault
X= detection
(not don't care!)

Quine-McCluskey Method [McCluskey 56]

- First EDA algorithm for 2-level logic synthesis
- Fault that is detected only once is *essential fault*
  - Must select *essential patterns* that detect essential faults
- Example:
  - f_1, f_3 are essential faults; t_2, t_4 are essential patterns
  - Test set selected = {t_2, t_3, t_4} or {t_1, t_2, t_4}, minimum test length = 3

		f_1	f_2	f_3	f_4	f_5
	t_1		X		X	
√	t_2			X		X
√	t_3		X		X	X
√	t_4	X			X	

Each row is a pattern
Each column is a fault
X= detection

What if No Essential Faults?

Quine-McCluskey Method (cont'd)

- 1. Remove redundant equivalent row, keep one row is enough
  - Row t_1 is equal to row t_2 because they have X in same columns
- 2. Remove dominated row
  - Row t_3 dominates row t_4 because
  - (1) row t_3 has X in all columns where row t_4 has X, and
  - (2) row t_3 has at least one X where row t_4 does not have X

	f_1	f_2	f_3	f_4	f_5	
t_1	X	X	X		X	
~~t_2~~	~~X~~	~~X~~	~~X~~		~~X~~	equivalent row
t_3		X	X	X	X	
~~t_4~~			~~X~~	~~X~~	~~X~~	dominated row
t_5	X			X	X	

Quine-McCluskey Method (cont'd)

- 3. Remove dominating column
  - Column f_5 dominates column f_4 (f_3, f_2, f_1 also) because
  - (1) column f_5 has X in all rows where column f_4 has X, and
  - (2) column f_5 has at least one X where column f_4 does not have X
- 4. Secondary essential
  - After steps 1~3, t_3 is now secondary essential pattern
- Minimum test set {t_1, t_3} or {t_3, t_5}, minimum test length =2

dominating column

	f_1	f_2	f_3	f_4	f_5
t_1	X	X	X		X
√ t_3		X	X	X	X
t_5	X				X

(cont'd from last page)

Quiz

Q1: Which are essential faults?
Q2: Which are dominated rows? Dominating columns?
Q3: What is minimum test length?

	f_1	f_2	f_3	f_4	f_5
t_1			X	X	X
t_2		X	X		X
t_3			X		
t_4	X				
t_5		X		X	X

QM Solves Many Cases in Polynomial Time

but not all…

FFT

- Mini-set covering is NP-hard
  - Q1: Show an special case where no rule of QM can be applied
  - Q2: What are you going to do with it?

	f_1	f_2	f_3	f_4
t_1	X	X		
t_2		X	X	
t_3			X	X
t_4	X			X

Alternative Solution, 01-ILP

- **Model STC as 01-Integer Linear Programming problem**

$$Objective: \quad \min \sum_i t_i$$

$$s.t. \quad \sum_i d_{i,j} \times t_i \geq 1, \quad foreach \;\; fault \;\; j$$

  - $t_i=1$ **,if test *i* is selected;** $t_i=0$ **otherwise**
  - $d_{i,j}=1$ **if test pattern** t_i **detects fault** f_j
- **Example:** $t_1=0$, $t_2=t_3=t_4=1$

	f_1	f_2	f_3	f_4	f_5
t_1		X			
t_2			X		X
t_3		X		X	X
t_4	X			X	

$$\min \quad t_1 + t_2 + t_3 + t_4$$

$$s.t.$$

$$t_4 \geq 1$$

$$t_1 + t_3 \geq 1$$

$$t_2 \geq 1$$

$$t_3 + t_4 \geq 1$$

$$t_2 + t_3 \geq 1$$

It is Well-solved… What Is Wrong?

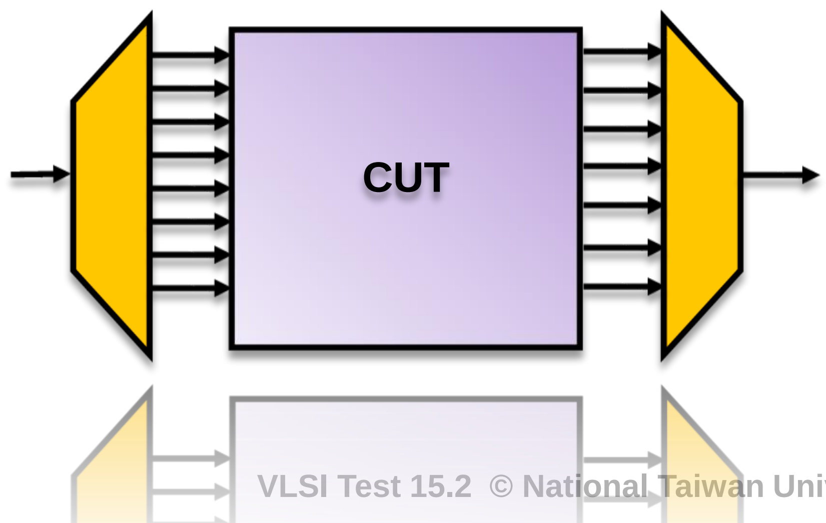
- **Q: What is practical issue of STC with dictionary?**
  - **A: Complete fault dictionary is very large, very slow**

	f_1	f_2	f_3	f_4	f_5
t_1		X	X		
t_2	X	X	X		X
t_3			X		X
t_4	X			X	

Need STC without Dictionary

Test Compression

- Introduction
- **Software Techniques**
  - Dynamic Test Compression
  - **Static Test Compression**
    - * With fault dictionary
    - * **Without fault dictionary**
      - – **Compatibility graph (X-unfilled)**
      - – Reverse order fault simulation (X-filled)
- Hardware Techniques
  - Test Stimulus Compression
  - Test Response Compression
  - Industry Practices
- Conclusion



STC w/o Dictionary (X-unfilled)

- Suppose that we do NOT have dictionary
  - but we have don't care bits in test cubes
- Two test cubes are ***compatible*** iff no conflict in specified bits
- Compatible test cubes can be ***merged*** into one test cube
- Example
  - t_0 and t_1 are compatible, merged to **0xx10**
  - Feasible solution:
    - **4 patterns:** $\{t_0+t_1+t_2, t_3+t_6, t_4+t_5, t_7\}$
    - Any better solution?

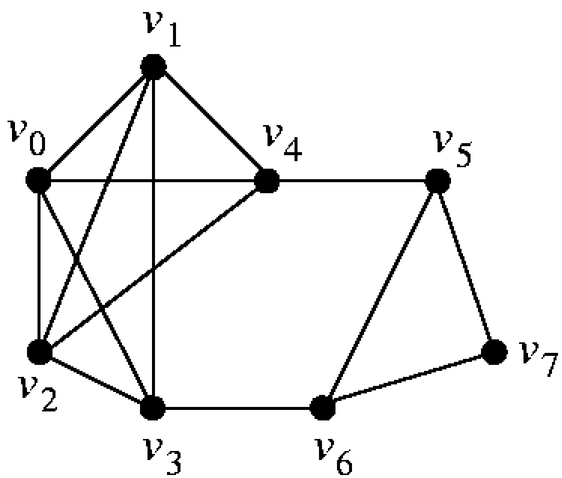
t_0	0xx10
t_1	0xx1x
t_2	0x01x
t_3	01xx0
t_4	x0xx0
t_5	1xxxx
t_6	x1x00
t_7	11xx0

x = don't cares

Any Algorithm to Solve This Problem?

Compatibility Graph

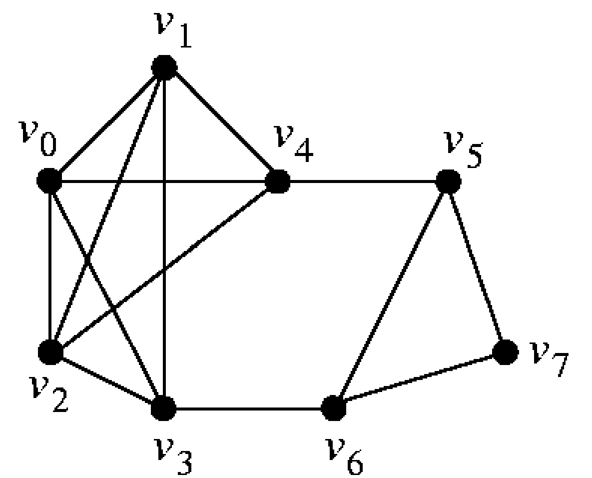
- ***Compatibility graph*** ***G(V, E)***
  - Vertex v_i represents a test cube t_i
  - Edge e_{ij} between v_i and v_j means two test cubes are **compatible**
  - **Adjacent vertices** can be merged into one
- **Example**
  - t_0 and t_1 are adjacent, can be merged



t_0	0xx10
t_1	0xx1x
t_2	0x01x
t_3	01xx0
t_4	x0xx0
t_5	1xxxx
t_6	x1x00
t_7	11xx0

Clique

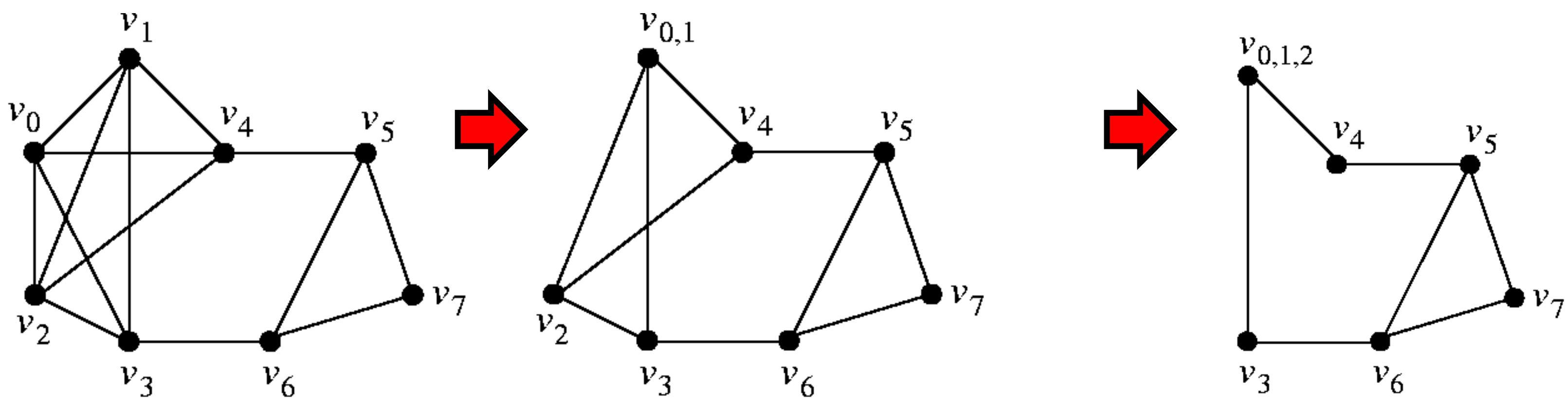
- ***Clique*** **is a subset of vertices such that**
  - **Each pair of vertices are connected**
  - **Clique is a *complete subgraph***
- ***Minimum clique partition problem***
  - **Partition graph into minimum number of cliques**
- **Example:**
  - $\{v_0, v_1, v_2, v_3\}$ $\{v_5, v_6, v_7\}$**are cliques**
  - **minimum clique partition**
    - $\{v_0, v_1, v_2, v_3\}\{v_4\}\{v_5, v_6, v_7\}$
    - **3 partitions**

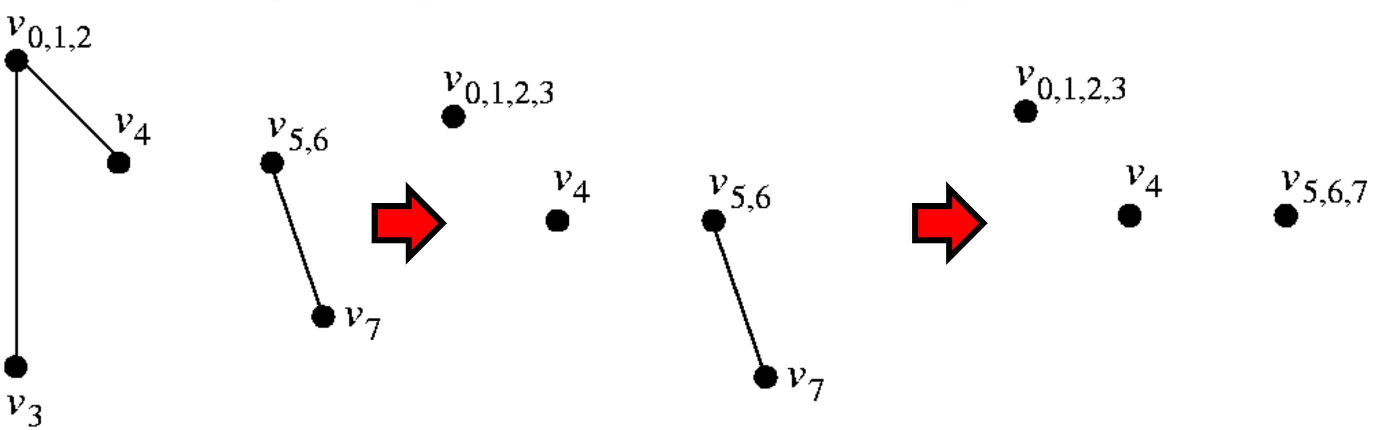


- **MCP is NP-hard problem**
  - **Can be solved by Tseng-Siewiorek**
    - **Greedy algorithm, does NOT guaranteed optimal solution**

Tseng-Siewiorek Idea

- Select two **adjacent vertices** of maximum ***common neighbors***
- **Merge two vertices into a *supervertex***
  - ***e.g.*, merge v_0 v_1 ➔ $v_{0,1}$ supervertices**
- **Iteratively merge vertices until no more edge**





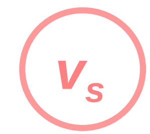
$t_{0,1,2,3}$	01010
t_4	x0xx0
$t_{5, 6, 7}$	11x00

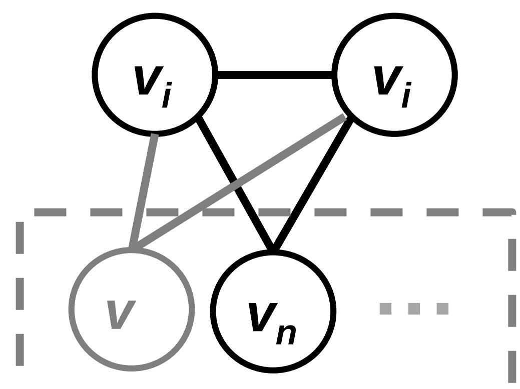
Tseng-Siewiorek Algorithm (1)

G_c^k = compatibility graph in K_{th} iteration
V_c^k = set of vertices in G_c^k
E_c^k = set of edges in G_c^k

$k \leftarrow 0;$
$G_c^k(V_c^k, E_c^k) \leftarrow G_c(V_c, E_c);$
$\boldsymbol{while}(E_c^k \neq \emptyset)\ \{$
    $find\ (v_i, v_j) \in E_c^k\ with$ largest set of common neibors;
    $\mathrm{N} \leftarrow set\ of\ common\ neibors\ of\ v_i\ and\ v_j;$
    $s \leftarrow i \cup j;$
    $V_c^{k+1} \leftarrow V_c^k \cup \{v_s\} \backslash \{v_i, v_j\};$ \ means remove
    $E_c^{k+1} \leftarrow \emptyset;$
    $\boldsymbol{for\ each}\ (v_m, v_n) \in E_c^k$ build new edges
        $\boldsymbol{if}(v_m \neq v_i \wedge v_m \neq v_j \wedge v_n \neq v_i \wedge v_n \neq v_j)$
            $E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_m, v_n)\};$
    $\boldsymbol{for\ each}\ v_n \in N$
        $E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_n, v_s)\};$
    $k \leftarrow k + 1;$
$\}$

v_s=supervertex of v_i and v_j





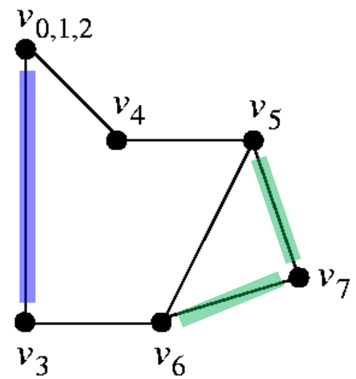
N={common neighbors}

Tseng-Siewiorek Algorithm (2)

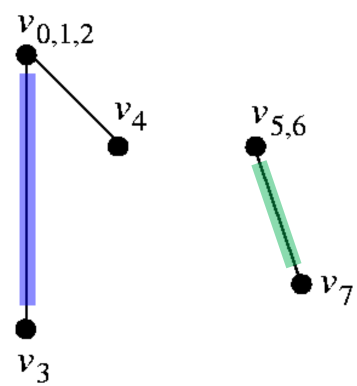
$k \leftarrow 0;$
$G_c^k(V_c^k, E_c^k) \leftarrow G_c(V_c, E_c);$
$\textbf{while}\left(E_c^k \neq \emptyset\right)\{$
$\quad find\ (v_i, v_j) \in E_c^k\ with$ largest set of common neibors;
$\quad \mathrm{N} \leftarrow set\ of\ common\ neibors\ of\ v_i\ and\ v_j;$
$\quad s \leftarrow i \cup j;$
$\quad V_c^{k+1} \leftarrow V_c^k \cup \{v_s\} \backslash \{v_i, v_j\};$
$\quad E_c^{k+1} \leftarrow \emptyset;$
$\quad \boldsymbol{for\ each}\ (v_m, v_n) \in E_c^k$
$\quad\quad \boldsymbol{if}(v_m \neq v_i \wedge v_m \neq v_j \wedge v_n \neq v_i \wedge v_n \neq v_j)$
$\quad\quad\quad E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_m, v_n)\};$
$\quad \boldsymbol{for\ each}\ v_n \in N$
$\quad\quad E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_n, v_s)\};$
$\quad k \leftarrow k + 1;$
$\}$

keep edges touch neither v_i nor v_j
***e.g.* edge (v_{012}, v_3)**

new edge between N and v_s
***e.g.* edge (v_5, v_7)**



v_i, v_j = v_5, v_6



Quiz

Q1: Draw compatibility graph
Q2: What is minimum test length using T-S algorithm?

t_0	0xx10
t_1	x1x10
t_2	0x11x
t_3	00x11

FFT

- Q: What is practical problem with this method?
  - Practically, there are many don’t cares in test cubes
  - X-unfilled test generation is slower and length is longer
    - than X-filled test generation

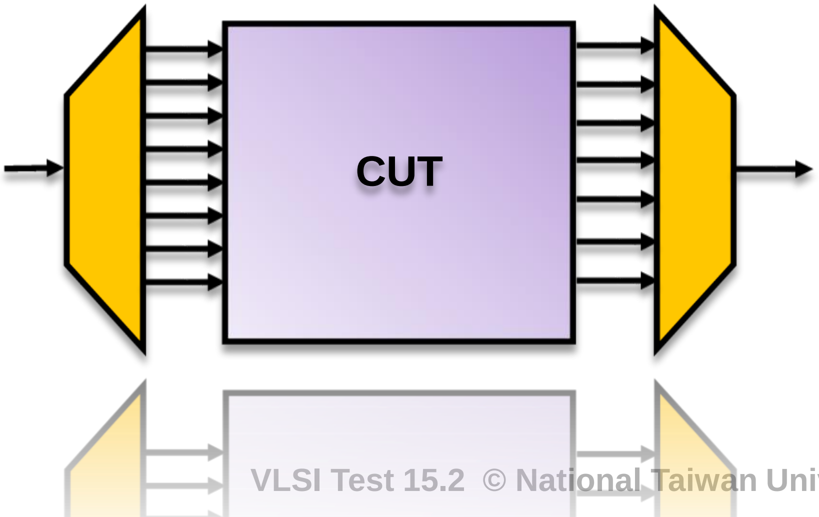
t_0	0xx10xxxxxx1
t_1	0xx1xxxxxxxx
t_2	0x01xxx1xxxx
t_3	01xx0xxxxxxx
t_4	X0xx0xxxxx0x
t_5	1xxxxxxxxxxx
t_6	x1x00xxxx1xx
t_7	11xx0xxxxxx0

t_0	011101100011
t_1	010111000000
t_2	010011010101

Need STC with X-filled

Test Compression

- Introduction
- Software Techniques
  - Dynamic Test Compression
  - Static Test Compression
    - With fault dictionary
    - Without fault dictionary
      - Compatibility graph (X-unfilled)
      - Reverse order fault simulation (X-filled)
- Hardware Techniques
  - Test Stimulus Compression
  - Test Response Compression
  - Industry Practices
- Conclusion



STC with X-filled (1)

- *Reverse-order fault simulation*
  - Fault simulate X-filled patterns in reverse order of ATPG
    - Delete redundant test patterns
  - Example:
    - First simulate t_4, and then t_3, t_2, t_1
    - Delete t_1. Choose test set {t_4, t_3, t_2}
  - Advantage: Simple, no dictionary needed. Most popular STC

ATPG order ↓

	f_1	f_2	f_3	f_4	f_5
t_1		X			
t_2			X		X
t_3		X		X	X
t_4	X			X	

Fault sim. order ↑

- Q: Why ATPG generated redundant t_1 at beginning?

STC with X-filled (2)

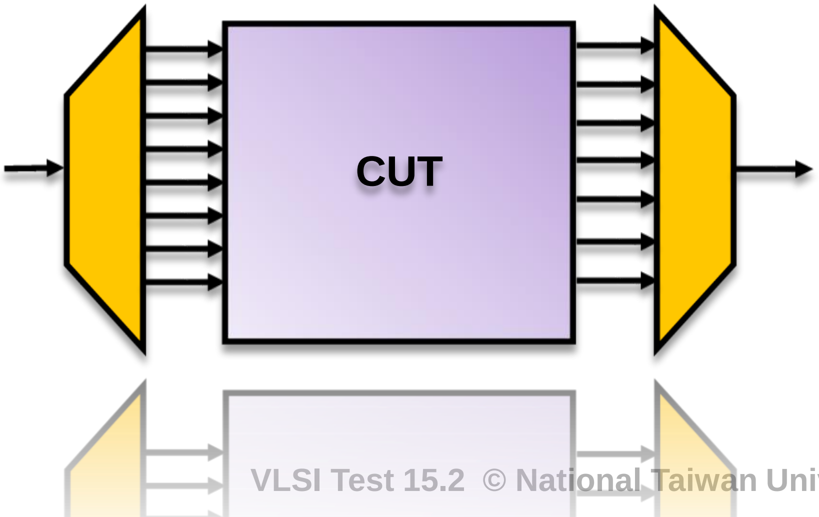
- *Random-order fault simulation*
  - Fault simulate test patterns in random order
  - Example:
    - First simulate t_4, and then t_2 , t_1, t_3
    - Choose test set {t_4, t_1, t_2}

	f_1	f_2	f_3	f_4	f_5
t_1		X			
t_2			X		X
t_3		X		X	X
t_4	X			X	

Too Many Orders to Try

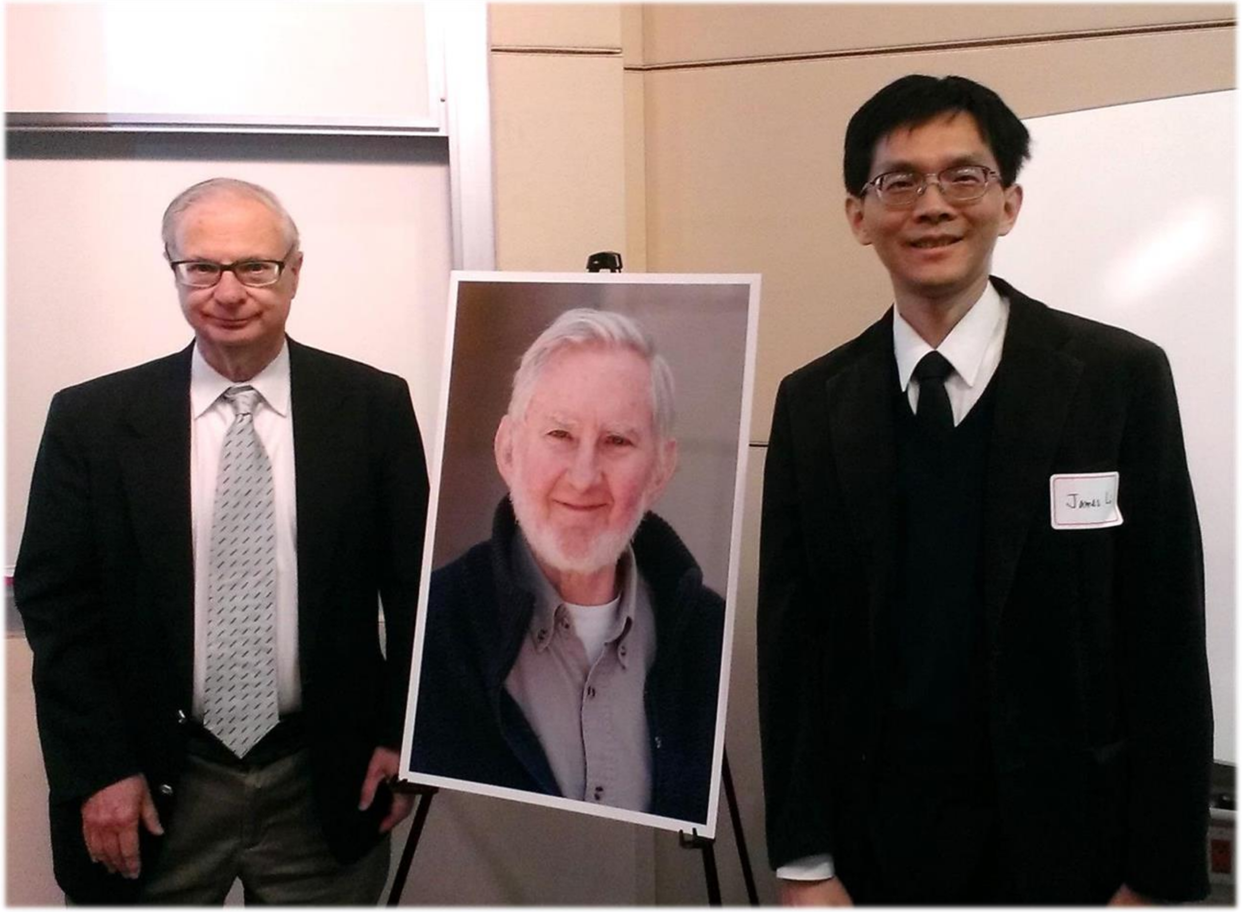
Summary

- **Static test compression**
  - **With fault dictionary**
    - **Minimum set-covering problem**
      - **Quine-McCluskey or 01-ILP**
    - **Too large dictionary**
  - **Without fault dictionary**
    - **Compatibility graph (X-unfilled)**
      - **T-S Algorithm**
    - **Reverse order fault simulation (X-filled)**
      - **Most popular solution**



Three Authors Together

- Prof. Siewiorek, Prof. McCluskey, Prof. James Li
- 2016 Stanford University

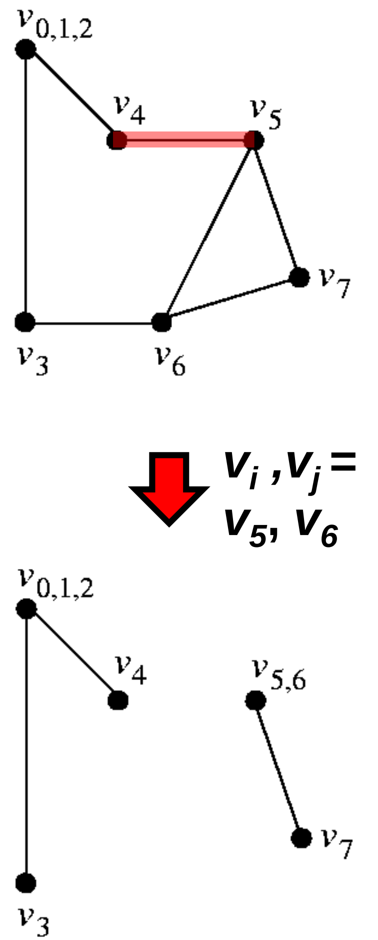


FFT1

$k \leftarrow 0;$
$G_c^k(V_c^k, E_c^k) \leftarrow G_c(V_c, E_c);$
$\boldsymbol{while}(E_c^k \neq \emptyset)\ \{$
$\quad find\ (v_i, v_j) \in E_c^k\ with$ largest set of common neibors;
$\quad \mathbb{N} \leftarrow set\ of\ common\ neibors\ of\ v_i\ and\ v_j;$
$\quad s \leftarrow i \cup j;$
$\quad V_c^{k+1} \leftarrow V_c^k \cup \{v_s\} \backslash \{v_i, v_j\};$
$\quad E_c^{k+1} \leftarrow \emptyset;$
$\quad \boldsymbol{for\ each}\ (v_m, v_n) \in E_c^k$
$\quad\quad \boldsymbol{if}(v_m \neq v_i \wedge v_m \neq v_j \wedge v_n \neq v_i \wedge v_n \neq v_j)$
$\quad\quad\quad E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_m, v_n)\};$
$\quad \boldsymbol{for\ each}\ v_n \in N$
$\quad\quad E_c^{k+1} \leftarrow E_c^{k+1} \cup \{(v_n, v_s)\};$
$\quad k \leftarrow k + 1;$
$\}$

keep edges touch neither v_i nor v_j
e.g. **edge (v_{012} , v_3)**

new edges between N and v_s
e.g. **edge (v_5, v_7)**



Why Edge (v_4,v_5) Removed?

FFT2

- **MCP is NP-hard**
- **Show an example when TS-algorithm fails to find an optimal solution**