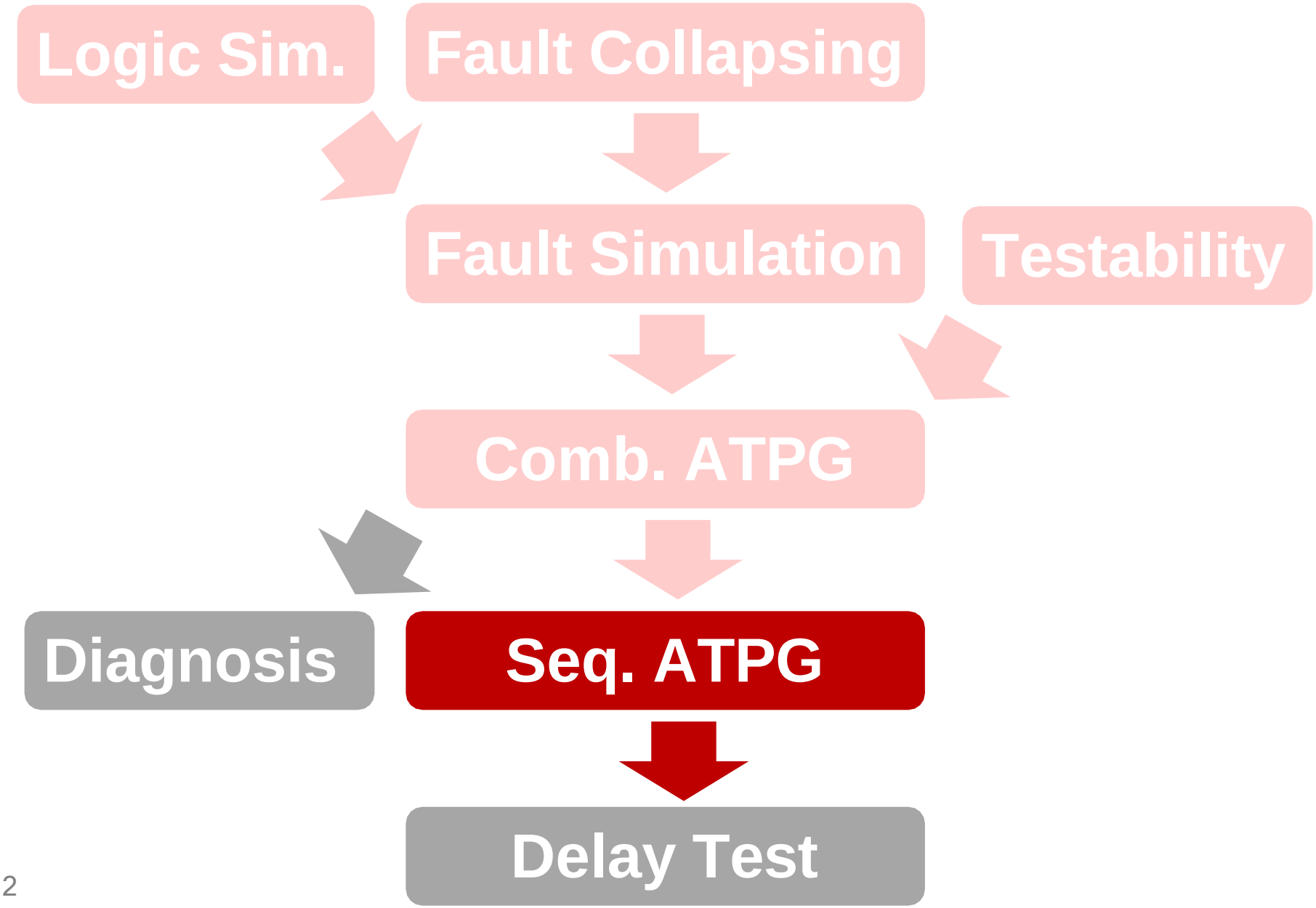
Sequential ATPG

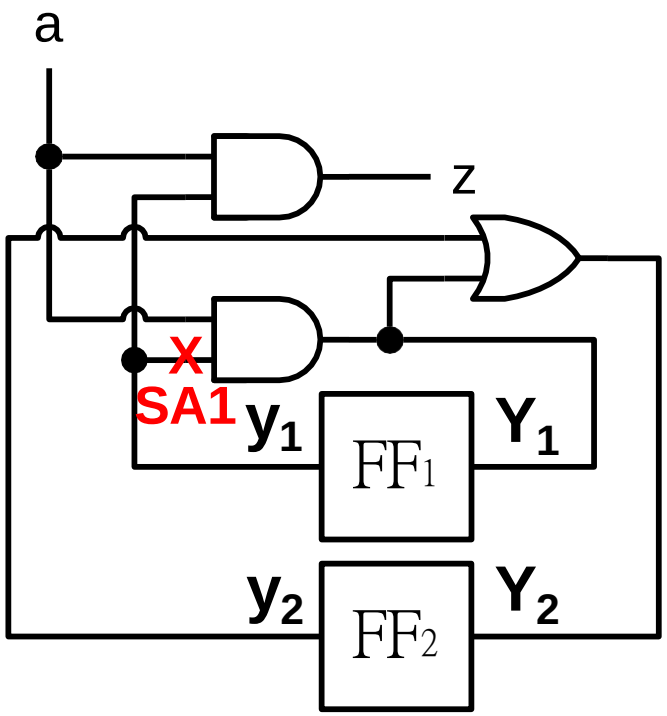
Dr. Bharat Garg
Assistant Professor,
DECE, TIET, Patiala

Course Roadmap (EDA Topics)



Motivating Problem

- You already know ATPG for combinational circuits
- But if is required to generate a test for sequential circuits
  - No scan allowed in flip-flops (FF)



Why Am I Learning This?

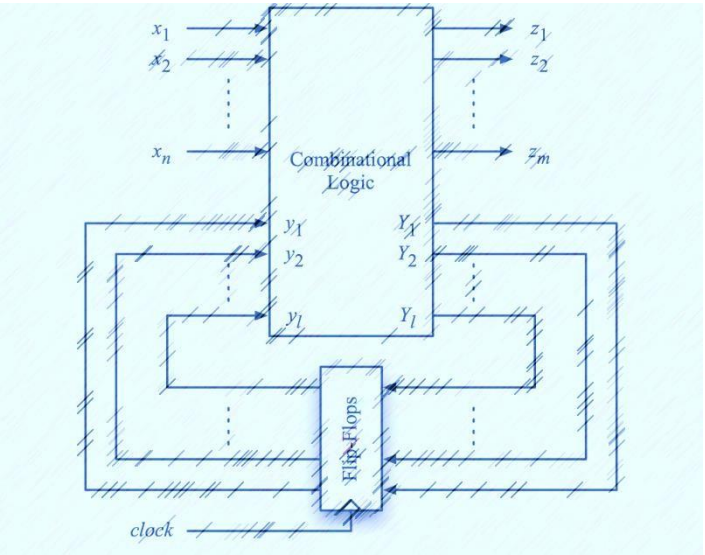
- **Sequential ATPG**
  - **Generate test patterns for sequential circuits**
  - **Without DFT or scan**

Test Generation

Fault Models	Combinational Circuits (or Sequential ckt. with scan)	Sequential Circuits
No fault model	PET	Checking experiment
Single Stuck-at Fault Model	D PODEM FAN	Extended D 9-valued
Delay Fault Model	Path Delay Transition Delay Fault	Launch on Capture Launch on Shift

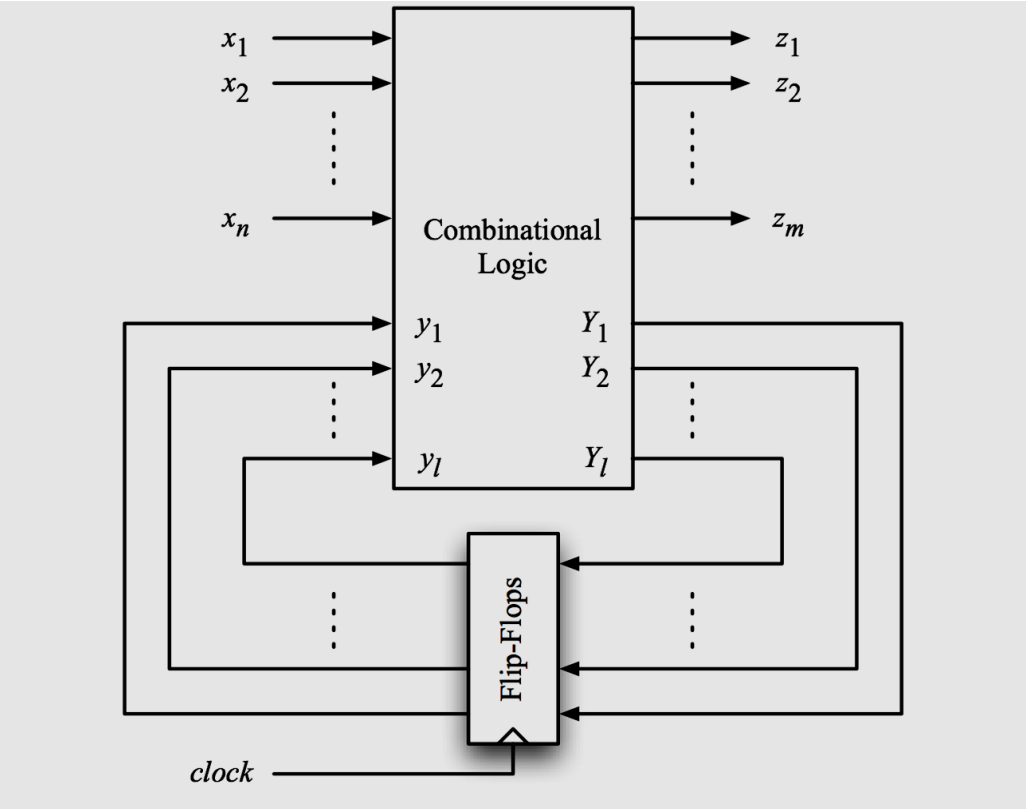
Sequential ATPG

- **Introduction**
- **Time-frame expansion methods**
- **Simulation-based methods**
- **Issues of Sequential ATPG**
- **Conclusions**



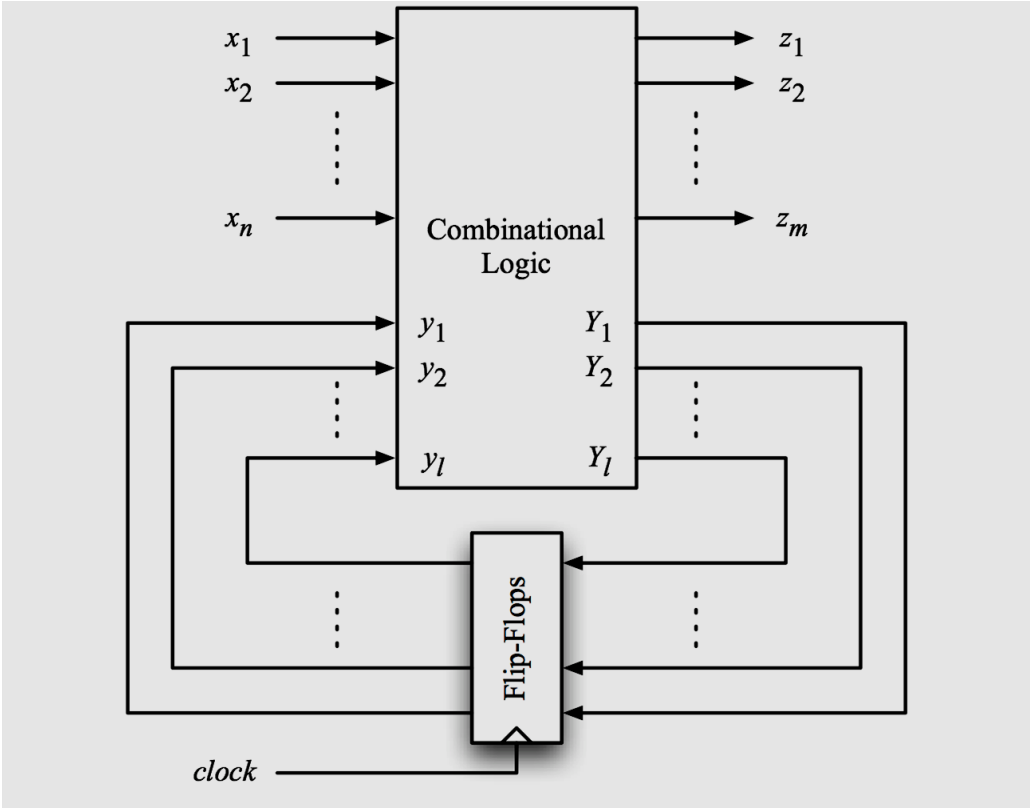
Huffman Model for Sequential Ckt. [Huffman 53]

- x_i = primary inputs (PI)
- z_i = primary outputs (PO)
- y_i = FF (or latch) current states
- Y_i = FF (or latch) next states



Sequential ATPG Assumptions

- NO SCAN ALLOWED!
  - Control only PI
    - FF not controllable
  - Observe only PO
    - FF not observable
- Faults in CL only
  - NO fault in FF/latches



Challenges of Sequential ATPG

- 1. FF/latches states uncontrollable and unobservable
  - FF/latch unknown initial states
- 2. Long run time
  - Comb. ATPG complexity (for a given fault)
    - $O(2^{number\_of\_PI})$
  - Seq. ATPG complexity (for a given fault)
    - $O(2^{number\_of\_PI} \times 9^{number\_of\_FF})$

*9-vlued logic will be covered soon

- 3. Large memory space required
  - Time frame expansion
- 4. Low fault coverage
  - Much worse than Comb. ATPG

Seq. ATPG is More Difficult Than Comb. ATPG

Quiz

Q: For a sequential circuit with 100 flip-flops, what is worst case sequential ATPG complexity?

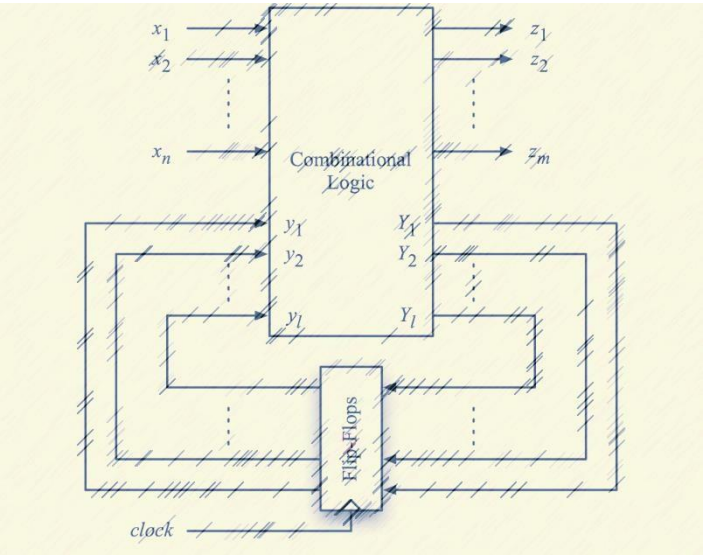
A: 9^{100}

B: 2^{100}

C: 100^2

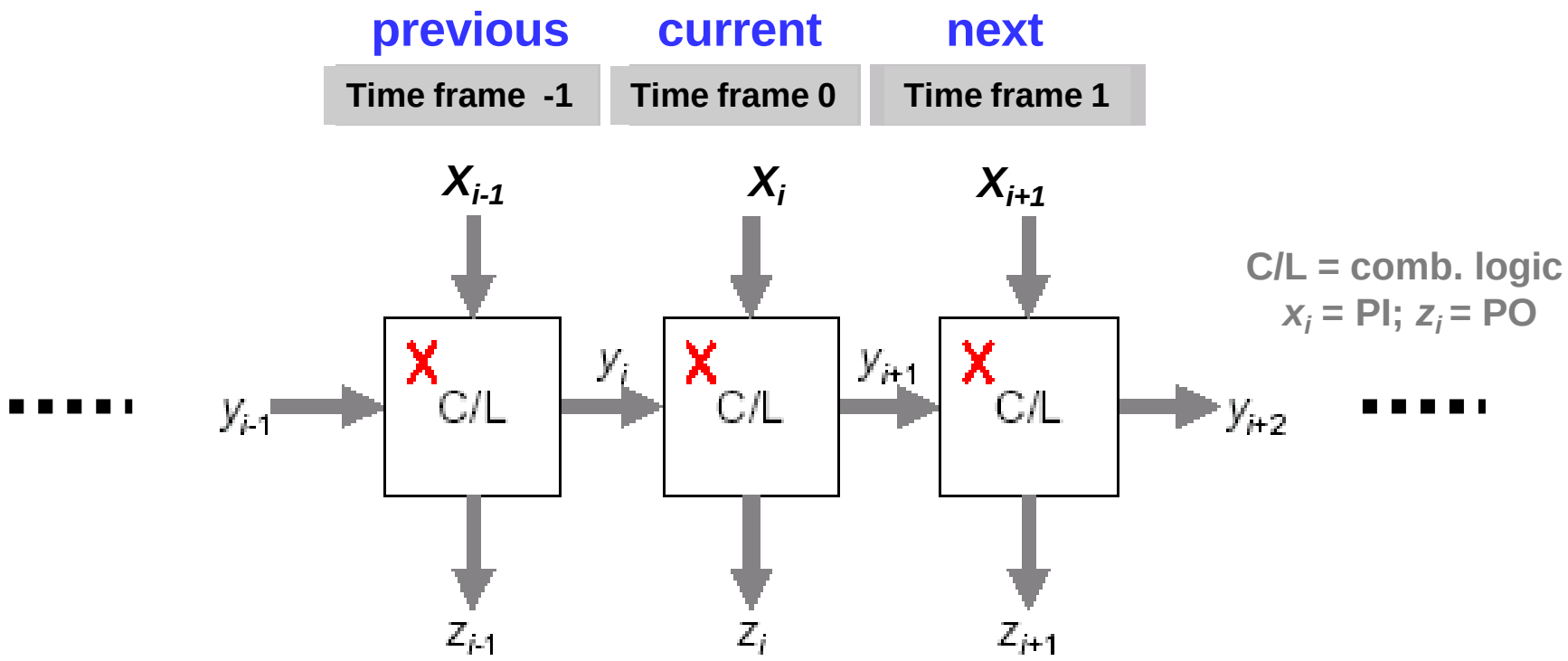
Sequential ATPG

- Introduction
- Time-frame expansion methods
  - The extended D-algorithm [Kubo 68]
  - 9-valued D algorithm [Muth 76]
  - EBT [Marlett 78], BACK [Cheng 88] *
  - Summary
- Simulation-based methods*
- Issues of Sequential ATPG*
- Conclusions



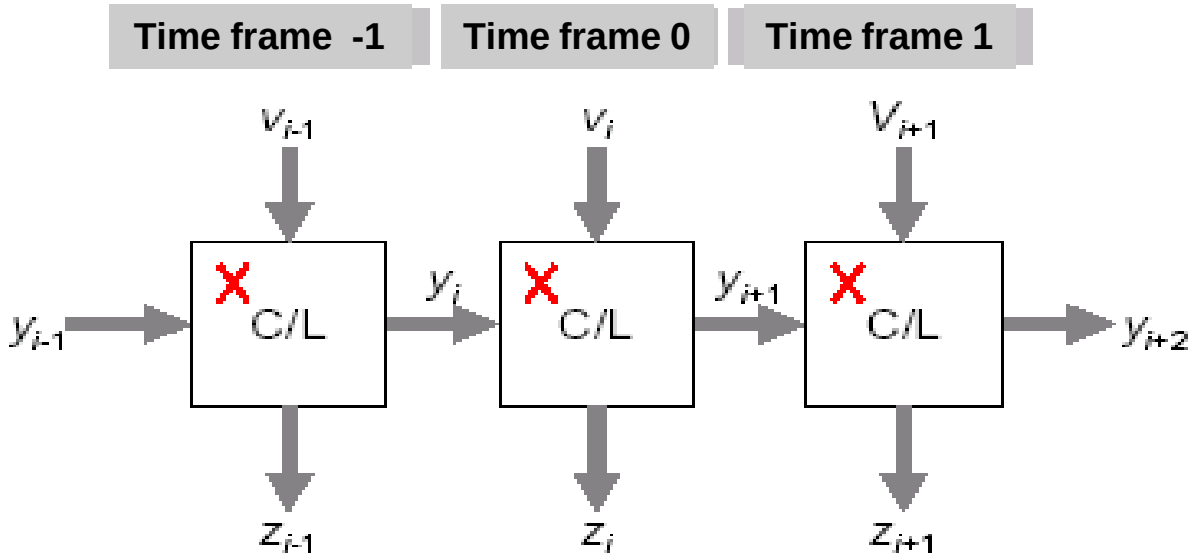
Time Frame Expansion

- IDEA: Replicate circuits and connect time frames by wires
  - y_i = "states"; No FF!
  - Replace clock cycles by space
- Becomes combinational ATPG problem
  - NOTE: Target fault appears in every time frame



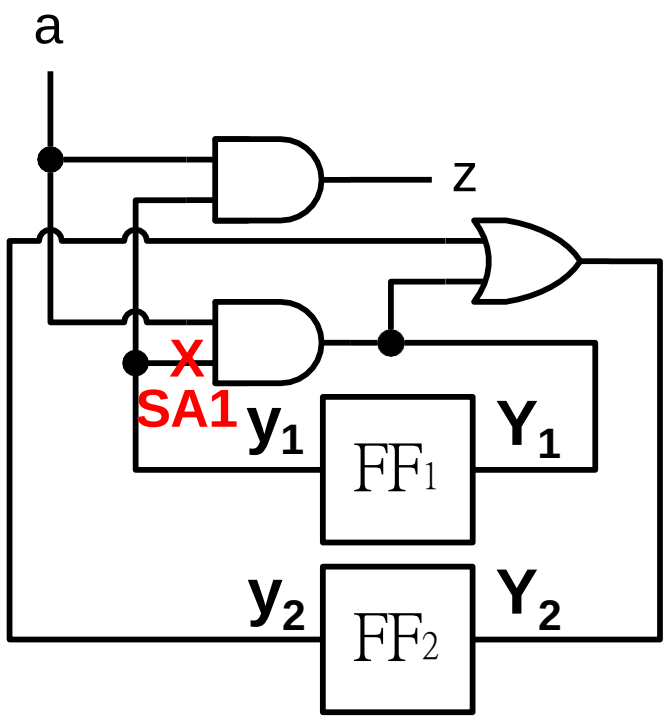
Extended D-Algorithm [Kubo 68]

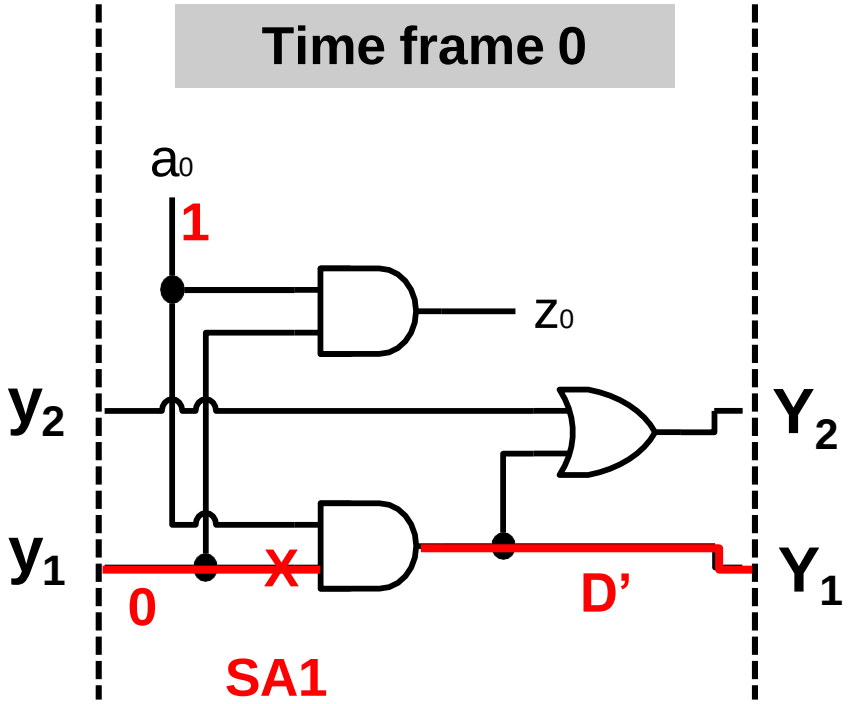
1. Select a target fault *f*
2. Create a copy of the combinational logic, set it to time frame 0
3. Generate a test for *f* for time frame 0 using D-algorithm
4. If the fault effect is propagated to the FF's, continue fault effect *propagation* in the *next* time frame
5. If there are values required in the FF outputs, continue the *justification* in the *previous* time frame



Example (1)

- **STEP 2: create Time frame 0**
- **STEP 3: generate a test**
  - **a_0=1; y_1=0 ; Y_1=D'**

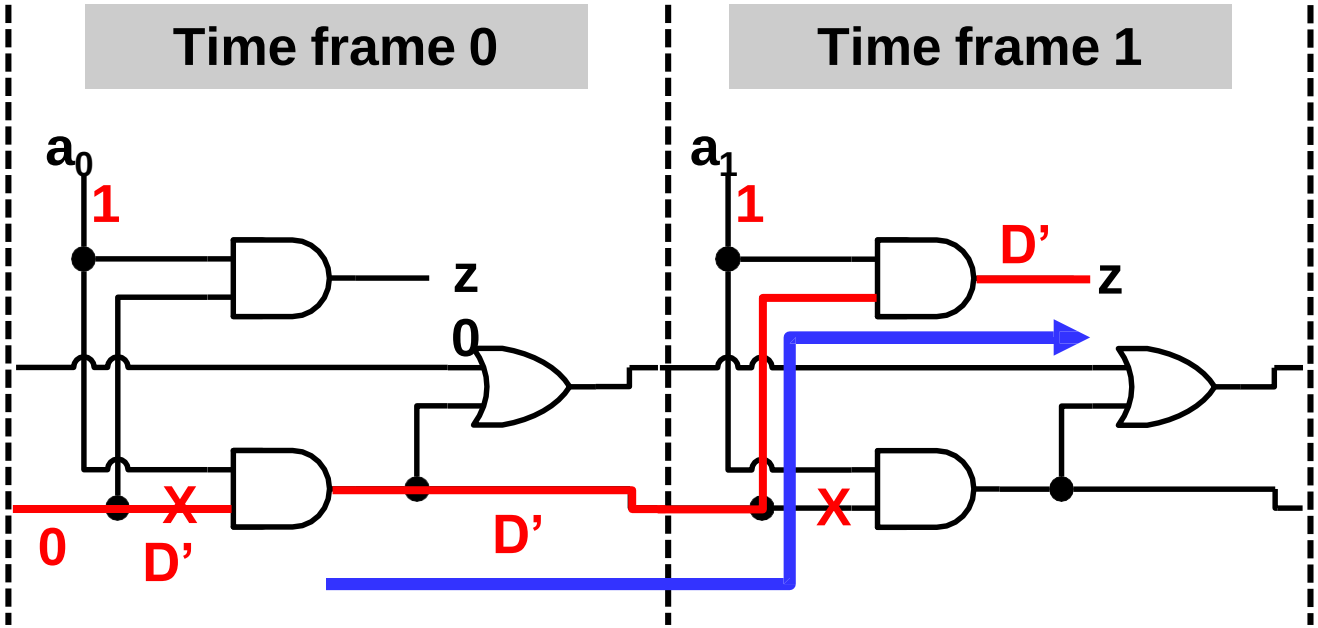




Still Need Propagation

Example (2)

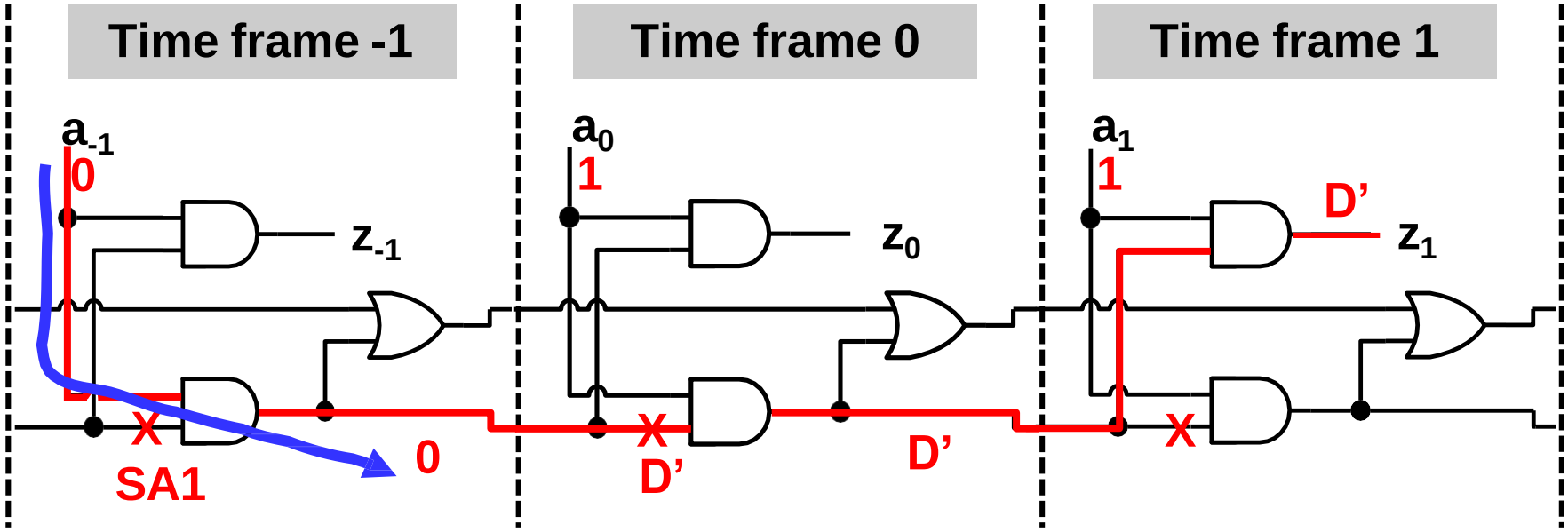
- **STEP4: Fault effect propagation to time frame 1**
  - **a_1=1**



Still Need Activation

Example (3)

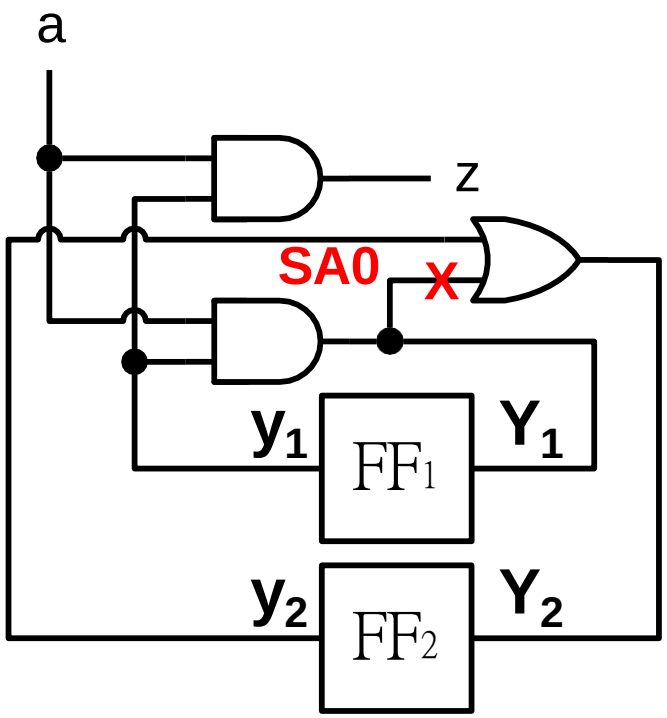
- **STEP 5: Fault activation back to time frame -1**
  - **a_{-1}=0**



Test Generated = 0, 1, 1

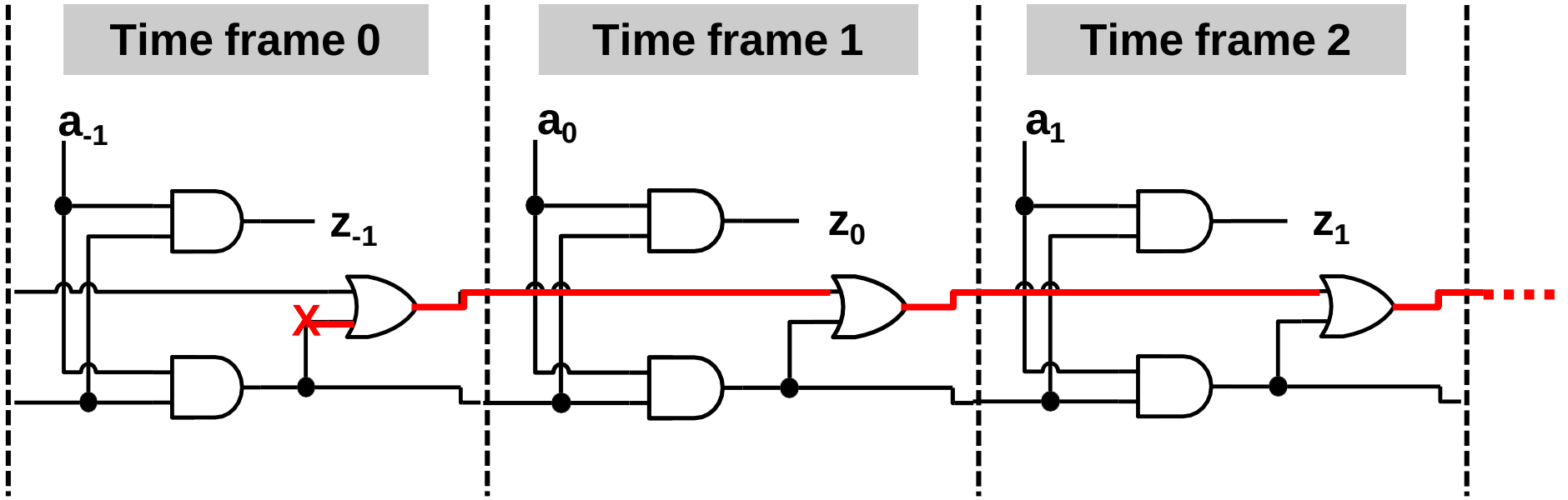
Quiz

Q: Generate a test for stuck-at zero fault in sequential circuit.



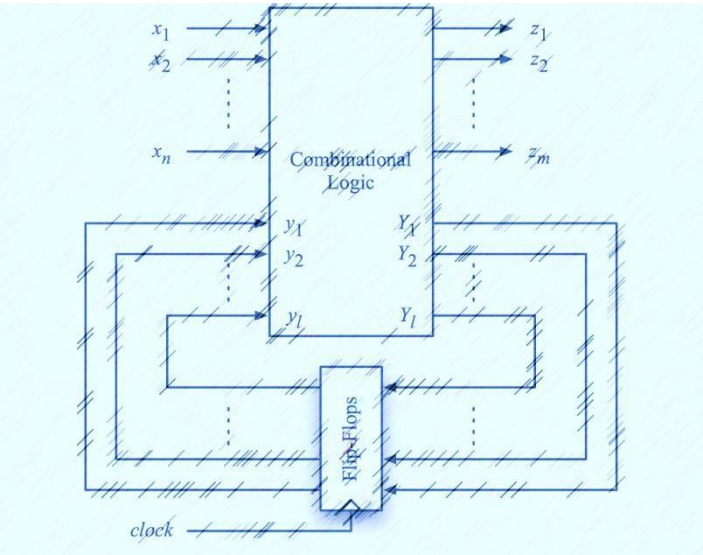
ANS

- No way to propagate
  - This fault is ***untestable*** by sequential ATPG
- Endless timeframe expansion …
  - Memory explosion!



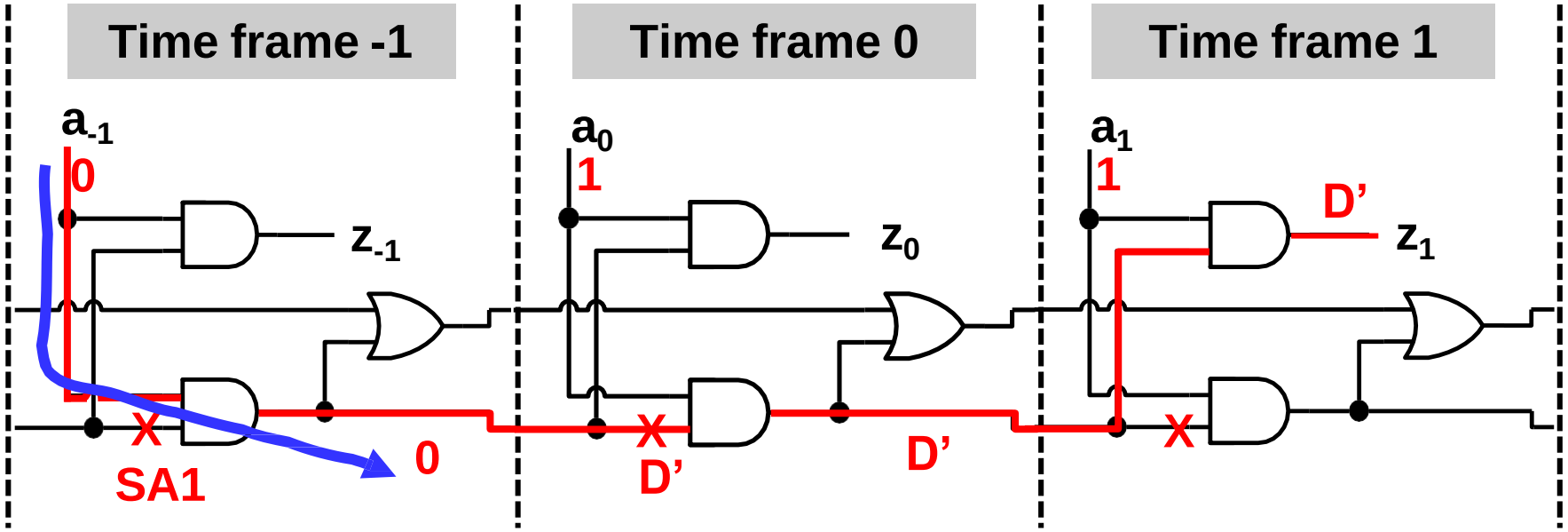
Summary

- **Sequential ATPG**
  - **Time and space consuming**
  - **Low fault coverage**
- **Time-frame expansion methods - extended D-algorithm**
  - **Replicate circuits into many time frames**
  - **Propagate forward, the then activated backward**
  - **create new time frame if needed**



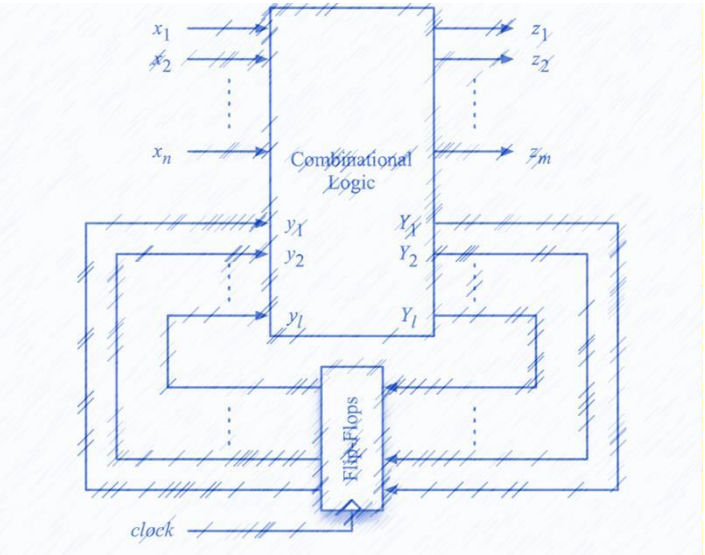
FFT

- In comb. ATPG, first fault activation, then propagation
- In seq. ATPG, first propagation, then activation
  - why?



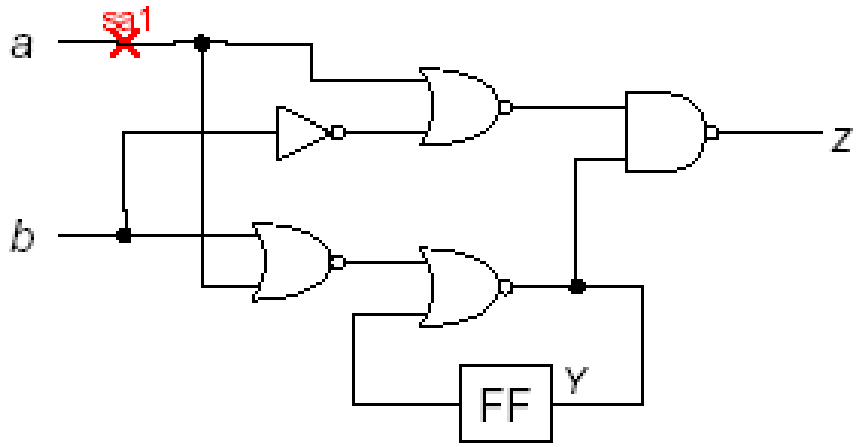
Sequential ATPG

- Introduction
- **Time-frame expansion methods**
  - The Extended D-algorithm [Kubo 68]
  - **9-valued D algorithm [Muth 76]**
  - EBT [Marlett 78], BACK [Cheng 88] *
  - Summary
- Simulation-based methods*
- Issues of Sequential ATPG*
- Conclusions



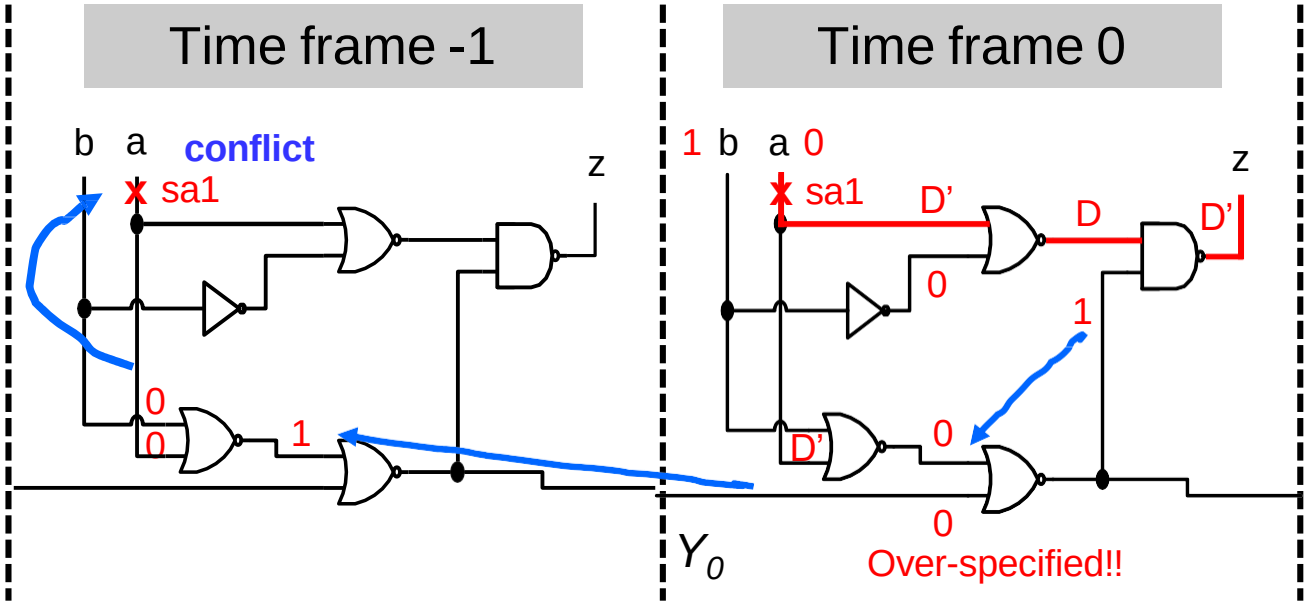
Quiz

Q: Given this test, can we detect the fault?
ANS:



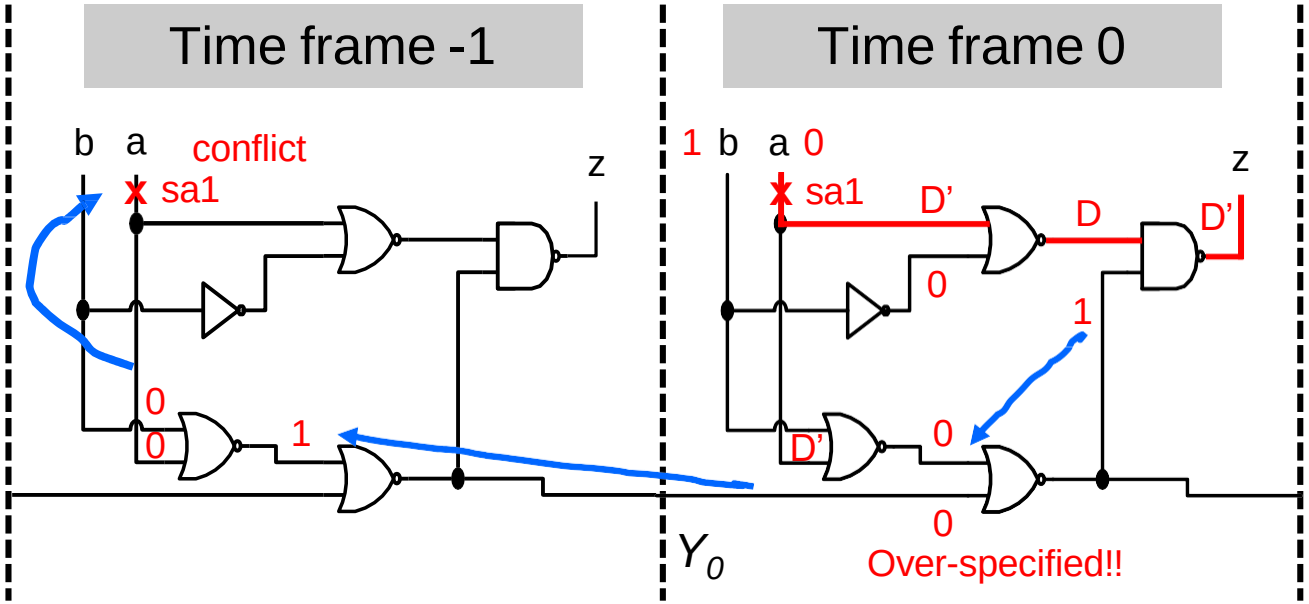
	a	*b*
v_1	0	0
v_2	0	1

Extended D-algorithm Fails!



- **Extended- D algorithm fails due to a conflict**
  - **Requires *a*=0 in time frame -1, but SA1**
  - **Actually, Y_0 is *over-specified* in 5-valued logic**

Why Fails?



- **Traditional 5-valued logic (0/0, 1/1, x/x, 0/1, 1/0) is NOT sufficient**
  - **cannot express 1/x, 0/x, x/0, x/1**

Q: How many total cases do we need?
ANS:

Nine-valued D-algorithm [Muth 76]

- **Solution: use *9-valued logic*, instead of 5-valued logic**

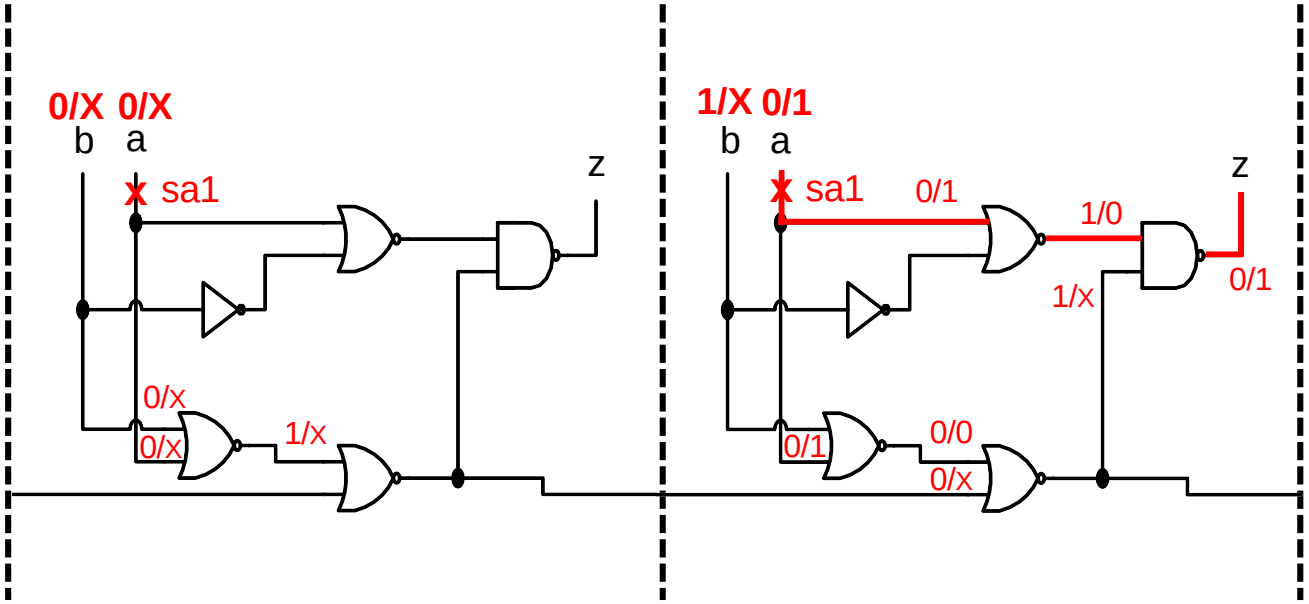
Symbol	Meaning	Roth's 5-valued logic		Muth's 9 valued logic	
		Fault-free	faulty	Fault-free	faulty
D	(1/0)	1	0	1	0
D'	(0/1)	0	1	0	1
0	(0/0)	0	0	0	0
1	(1/1)	1	1	1	1
X	(x/x)	X	X	X	X
G0	(0/x)	-	-	0	X
G1	(1/x)	-	-	1	X
F0	(x/0)	-	-	X	0
F1	(x/1)	-	-	X	1

Nine-valued Truth Table

- **Example of AND gate**

AND	0	0/x	D’	x/0	x/x	x/1	D	1/x	1
0	0	0	0	0	0	0	0	0	0
0/x	0	0/x	0/x	0	0/x	0/x	0	0/x	0/x
D’	0	0/x	D’	0	0/x	D’	0	0/x	D’
x/0	0	0	0	x/0	x/0	x/0	x/0	x/0	x/0
x/x	0	0/x	0/x	x/0	x/x	x/x	x/0	x/x	x/x
x/1	0	0/x	D’	x/0	x/x	x/1	x/0	x/x	x/1
D	0	0	0	x/0	x/0	x/0	D	D	D
1/x	0	0/x	0/x	x/0	x/x	x/x	D	1/x	1/x
1	0	0/x	D’	x/0	x/x	x/1	D	1/x	1

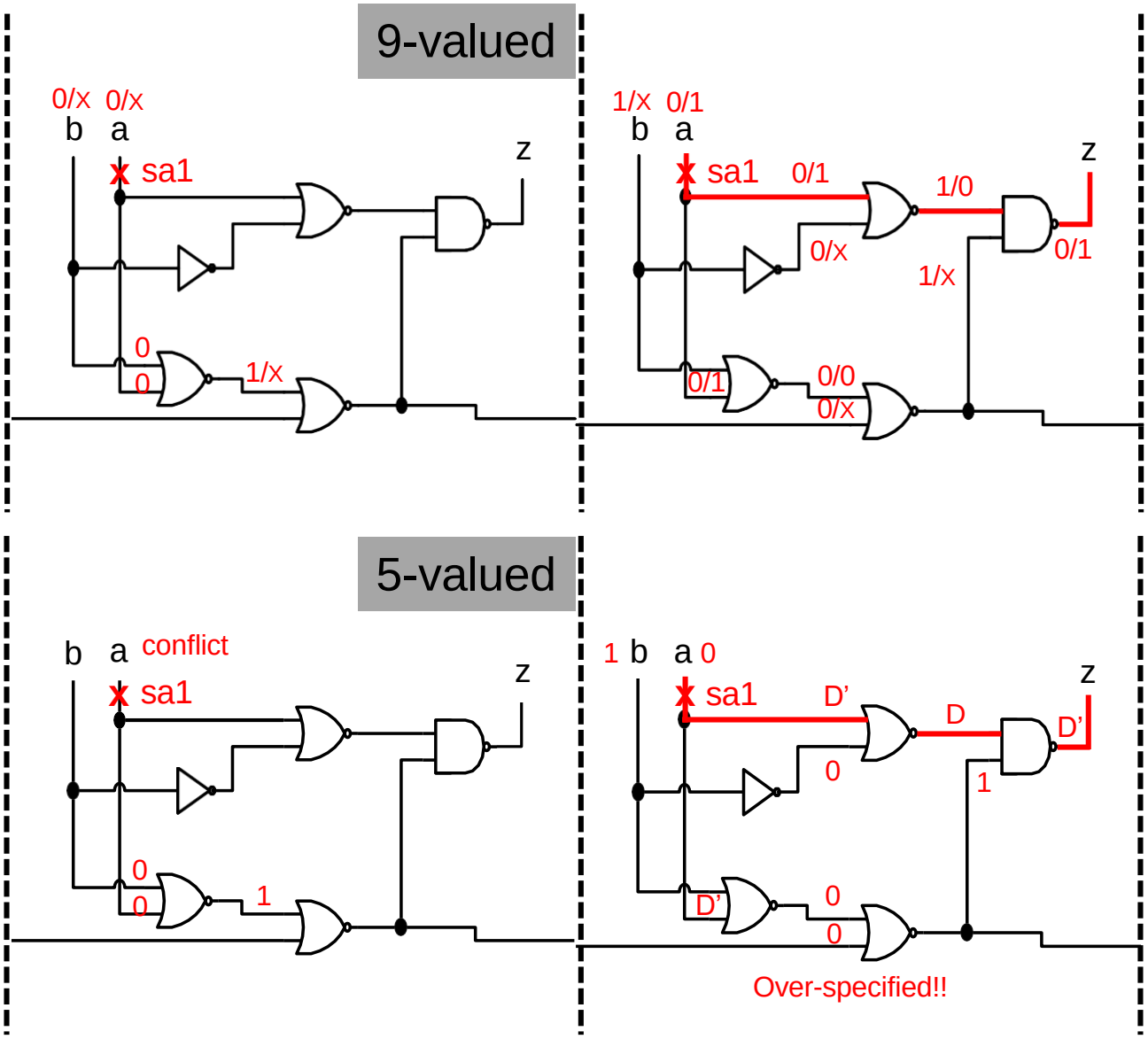
Nine-Valued Test Generation



Test pattern successfully generated

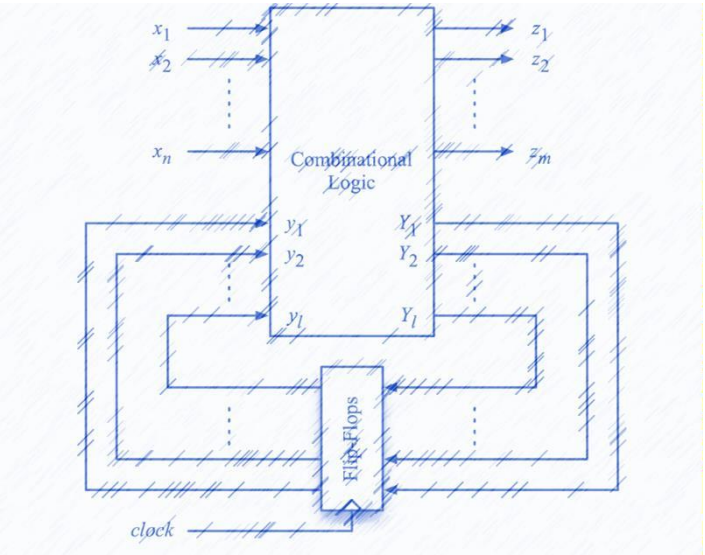
	a	b
v_1	0	0
v_2	0	1

Comparison: 9 v.s. 5 valued



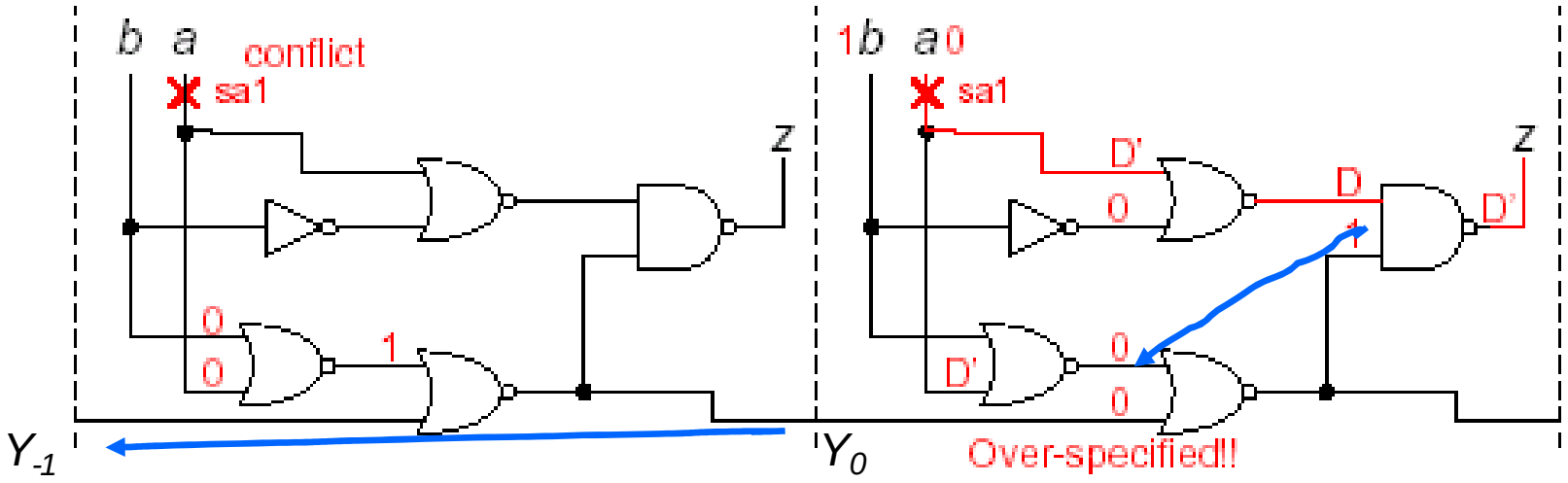
Sequential ATPG

- Introduction
- **Time-frame expansion methods**
  - The Extended D-algorithm [Kubo 68]
  - **9-valued D algorithm [Muth 76]**
    - express all nine possible logic states
    - avoid over-specification
  - EBT [Marlett 78], BACK [Cheng 8
  - Summary
- Simulation-based methods*
- Issues of Sequential ATPG*
- Conclusions



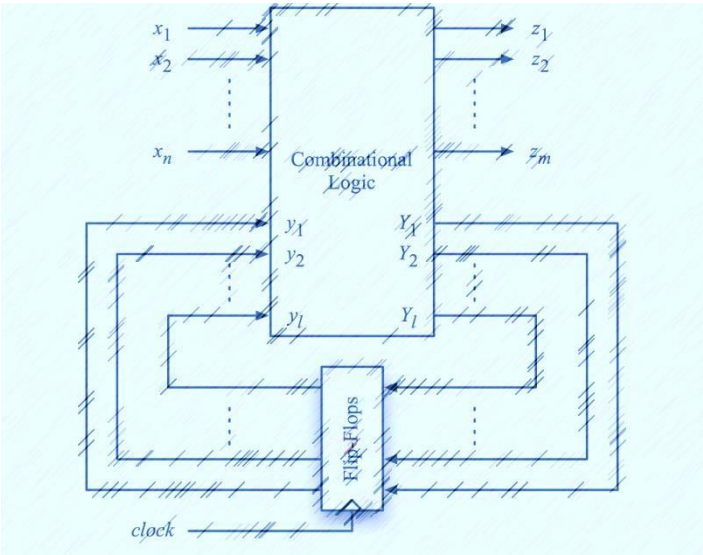
FFT

- Q1: Why NOT consider 1/x, 0/x, x/0, x/1 in combinational ATPG?
- Q2: Why NOT backtrace Y_{-1} one more time frame to the left?



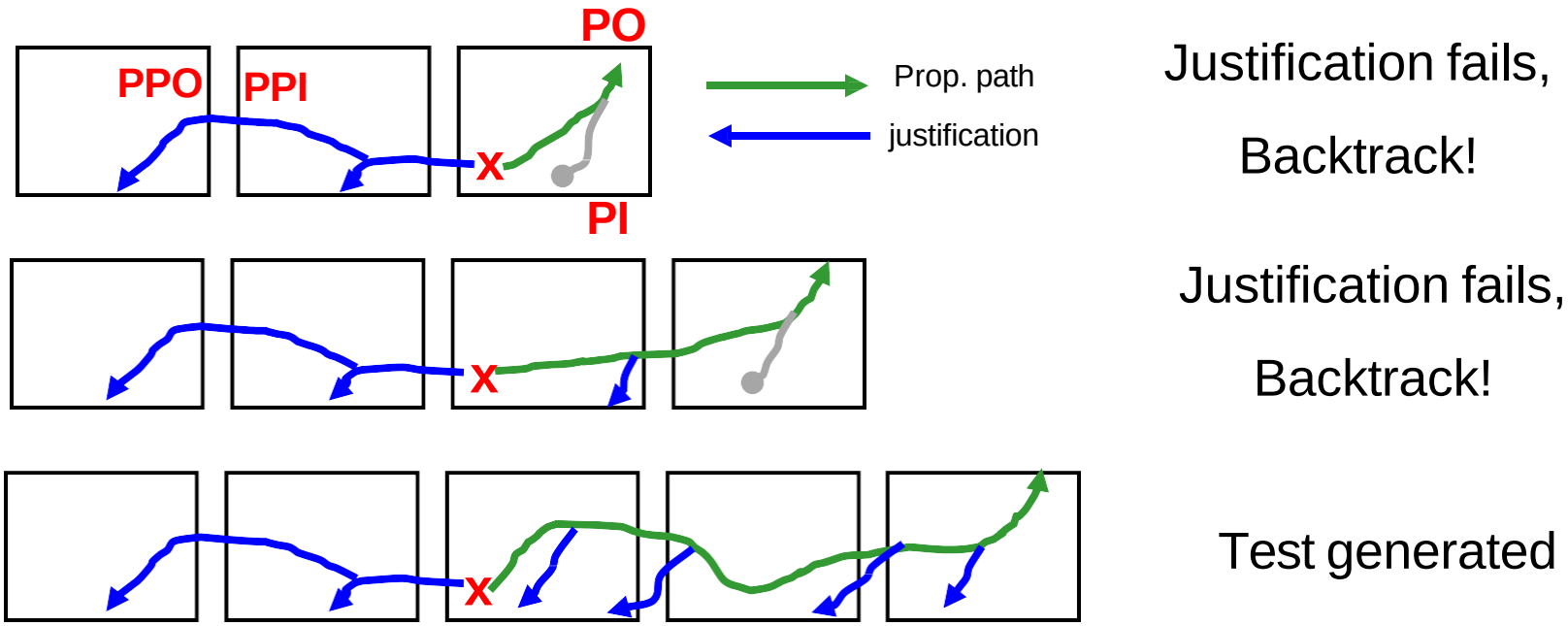
Sequential ATPG

- Introduction
- **Time-frame expansion methods**
  - The Extended D-algorithm [Kubo 68]
  - 9-valued D algorithm [Muth 76]
  - **Backward Time Frame Processing* (not in exam)**
    - **EBT [Marlett 78]**
    - **BACK [Cheng 88]**
  - Simulation-based methods*
- Issues of Sequential ATPG*
- Conclusions



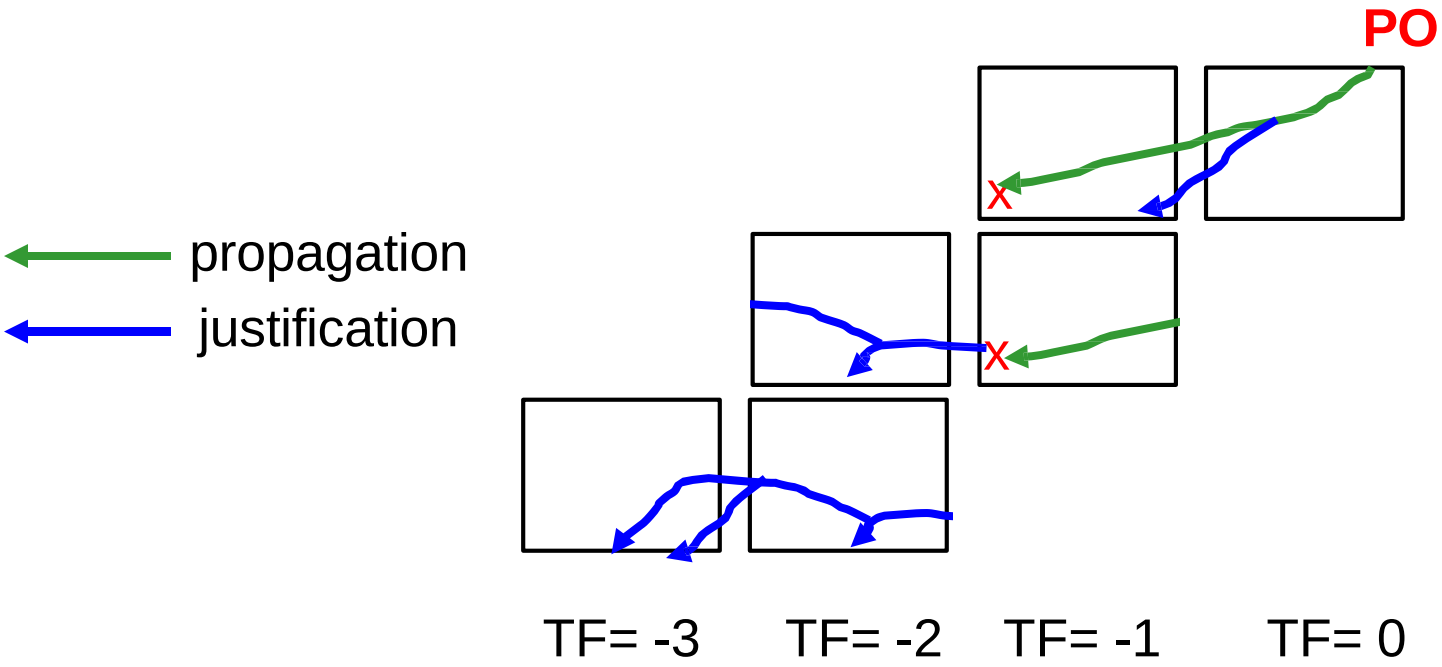
Problems of Ext. D-Algorithm

- *Mixed Forward and Backward Time Frame Processing*
  - **Reuse existing D-algorithm**
    - **both forward fault propagation and backward justification**
  - **How many time frames ?**
    - **Memory requirement hard to predict**



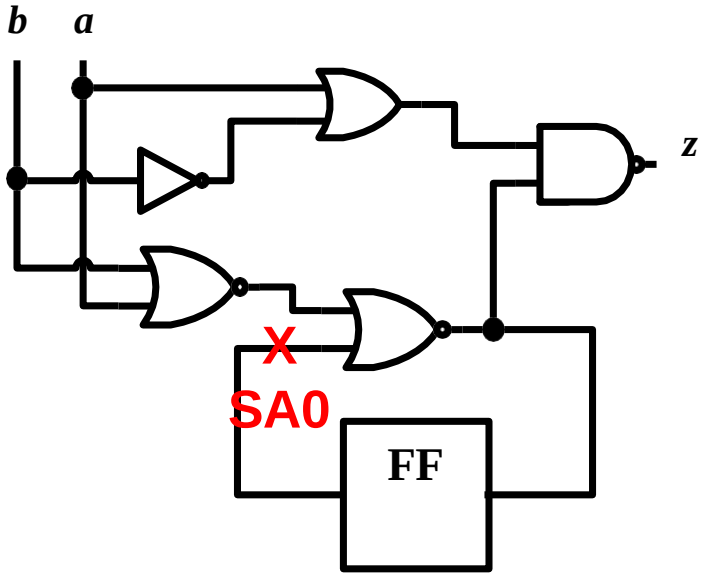
Extended Backtrace (EBT) [Marlett 78]

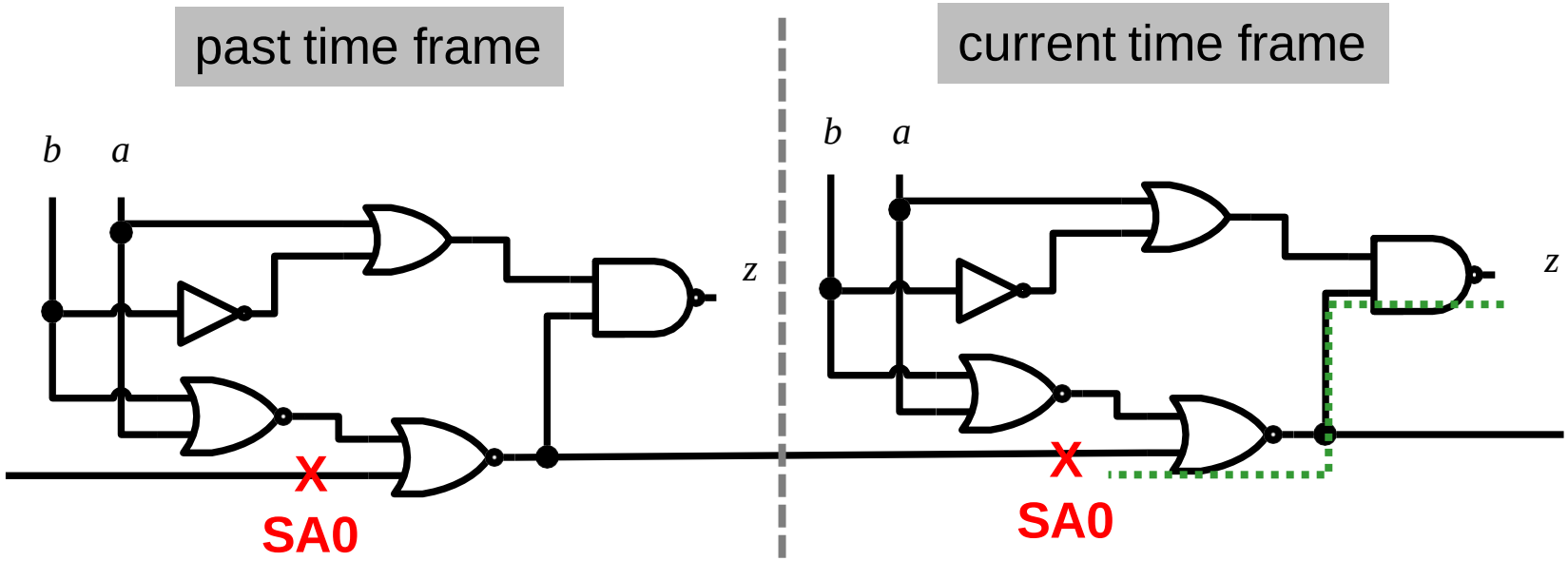
- *Backward Time Frame Processing* Only
  - 1. Select a path from fault site to PO
  - 2. Sensitize path backwards from the PO
  - 3. Justify required values backward
    - If justification fails, choose another path
- Advantage: only two time frames needed

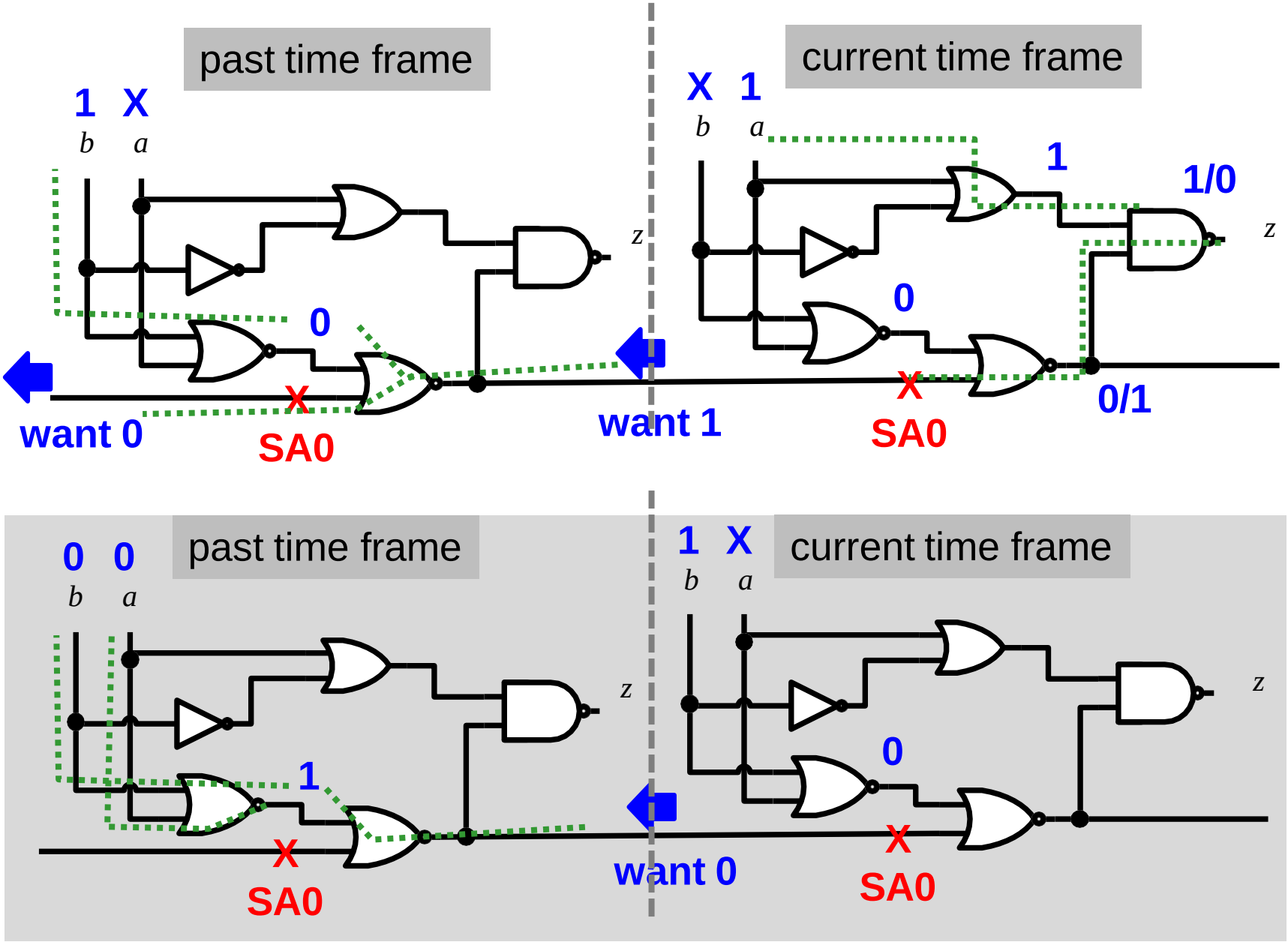


EBT Example

- 1. Create two time frames
- 2. Choose a path

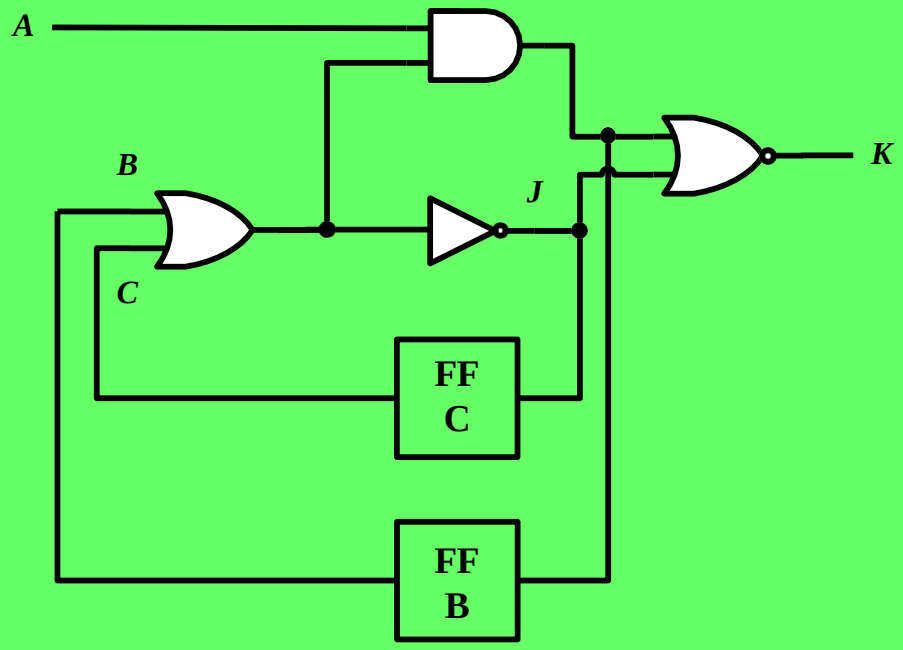





past time frame
current time frame
1 X
b a
X 1
b a
0
X
SA0
want 0
want 1
1
1/0
0/1
z
past time frame
current time frame
0 0
1 X
want 0
ab = 00, 1x, x1

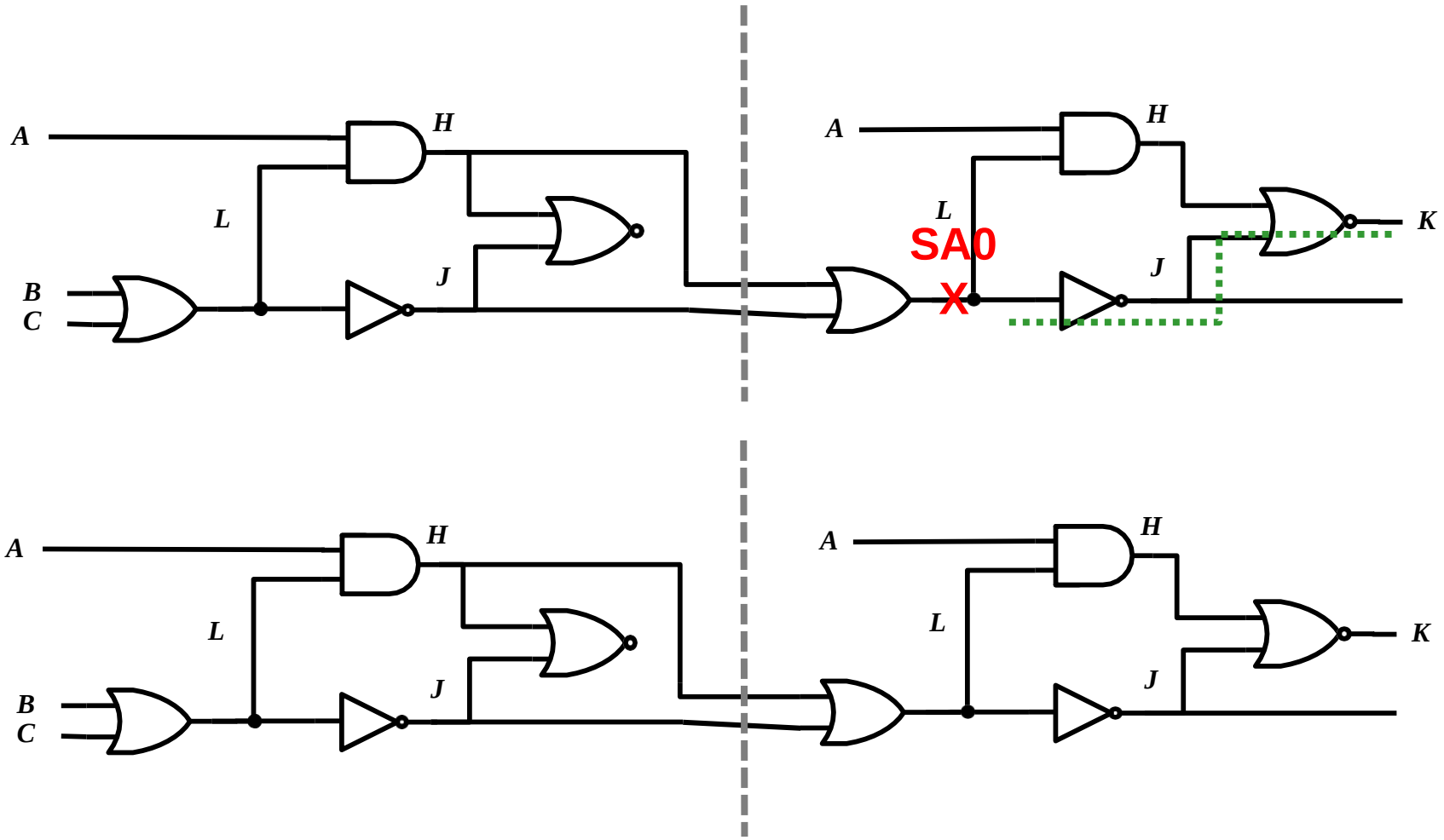
Quiz

Q: Please redraw this into two time frames

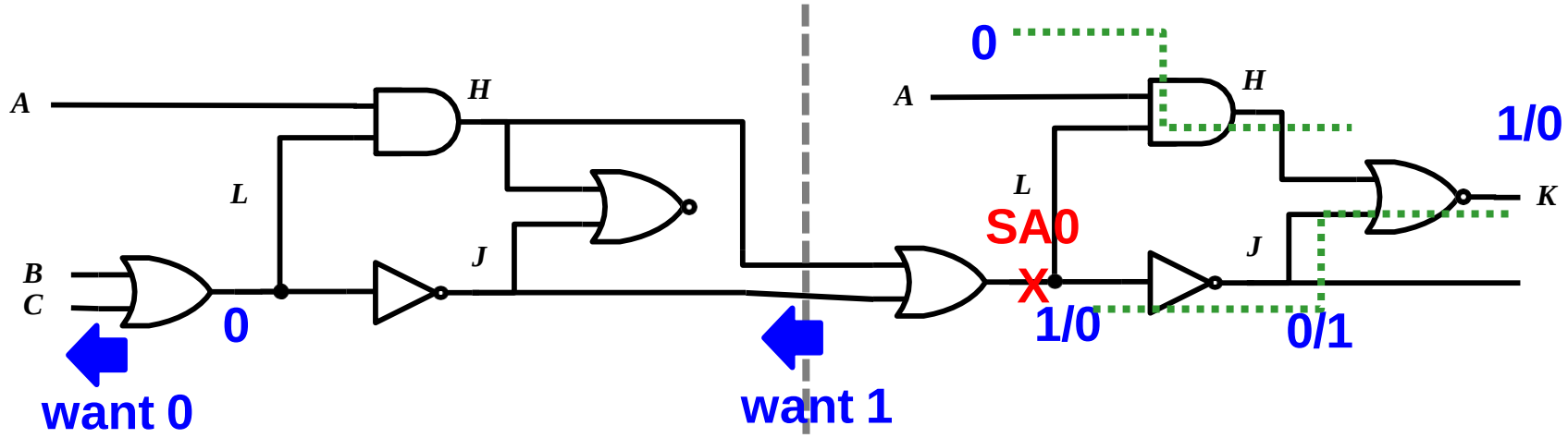


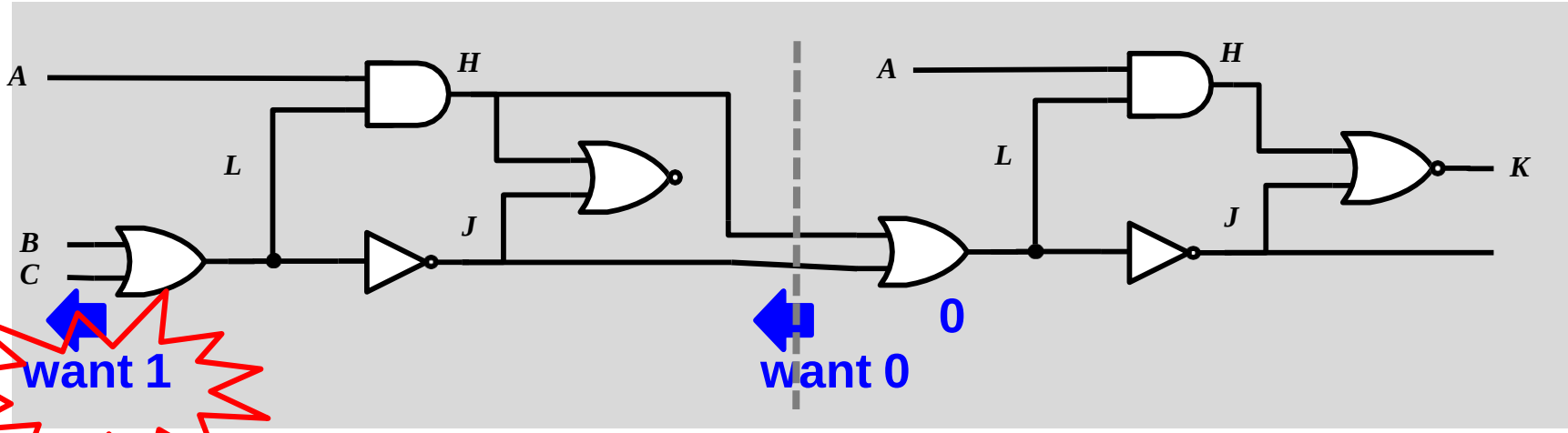
Quiz (cont'd)

Q: Use EBT to generate test patterns for SA0 fault



State Repetition





Justification Fails Due to State Repetition

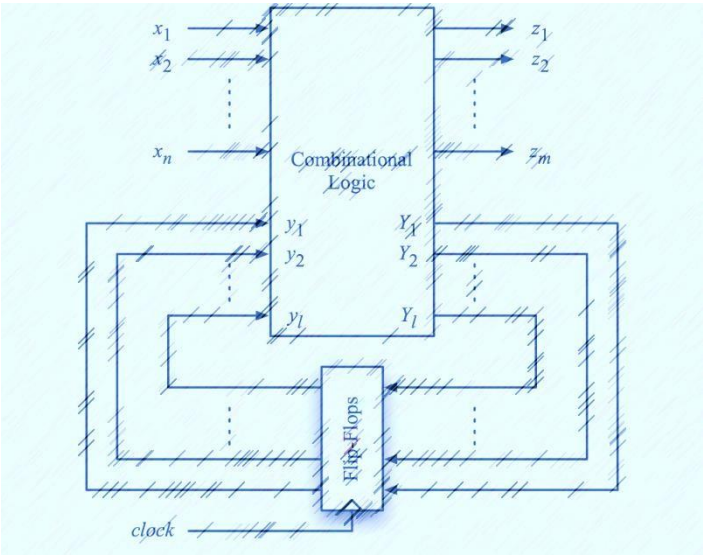
Comparison

	Advantages	Disadvantages	Example
Mixed F/B Time Frame	☺Reuse existing D algorithm	☹Too many time frames	Extended D
Backward TF Only	☺Fixed time frames ☺State repetition recognized	☹ Too many paths! How to choose path?	EBT

Need Help to Make Smart Decision

Sequential ATPG

- Introduction
- **Time-frame expansion methods**
  - The Extended D-algorithm [Kubo 68]
  - 9-valued D algorithm [Muth 76]
  - **Backward Time Frame Processing* (not in exam)**
    - **EBT [Marlett 78]**
    - **BACK [Cheng 88]**
  - Simulation-based methods*
- Issues of Sequential ATPG*
- Conclusions



BACK Algorithm [Cheng 88]

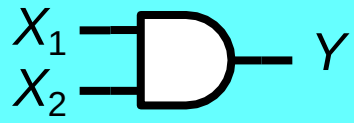
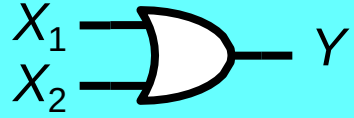
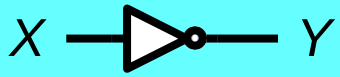
- BACK selects a ***PO*** for fault detection,
  - Do not explicitly select a *path*
- PO selection based on a testability measures
  - ***Drivability***

- Sensitized path will be created implicitly when drivability calculation

BACK Chooses PO Instead of Path

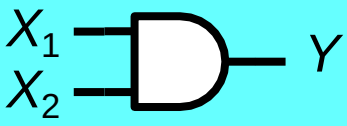
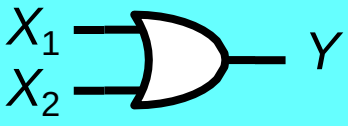
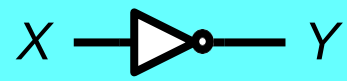
Review: SCOAP (CH 6)

- *$CC^0(N)$, $CC^1(N)$*
  - **Minimum number of combinational PI assignments and logic levels required to control a 0 or a 1 on node *N***
  - **Estimates effort to control signal to zero or one**
    - ***Smaller* number, *easier* to control**

	$CC^0(Y)$	$CC^1(Y)$
	$\min[CC^0(X_1),CC^0(X_2)] + 1$	$CC^1(X_1) + CC^1(X_2) + 1$
	$CC^0(X_1) + CC^0(X_2) + 1$	$\min[CC^1(X_1),CC^1(X_2)] + 1$
	$CC^1(X) + 1$	$CC^0(X) + 1$
Primary inputs	1	1

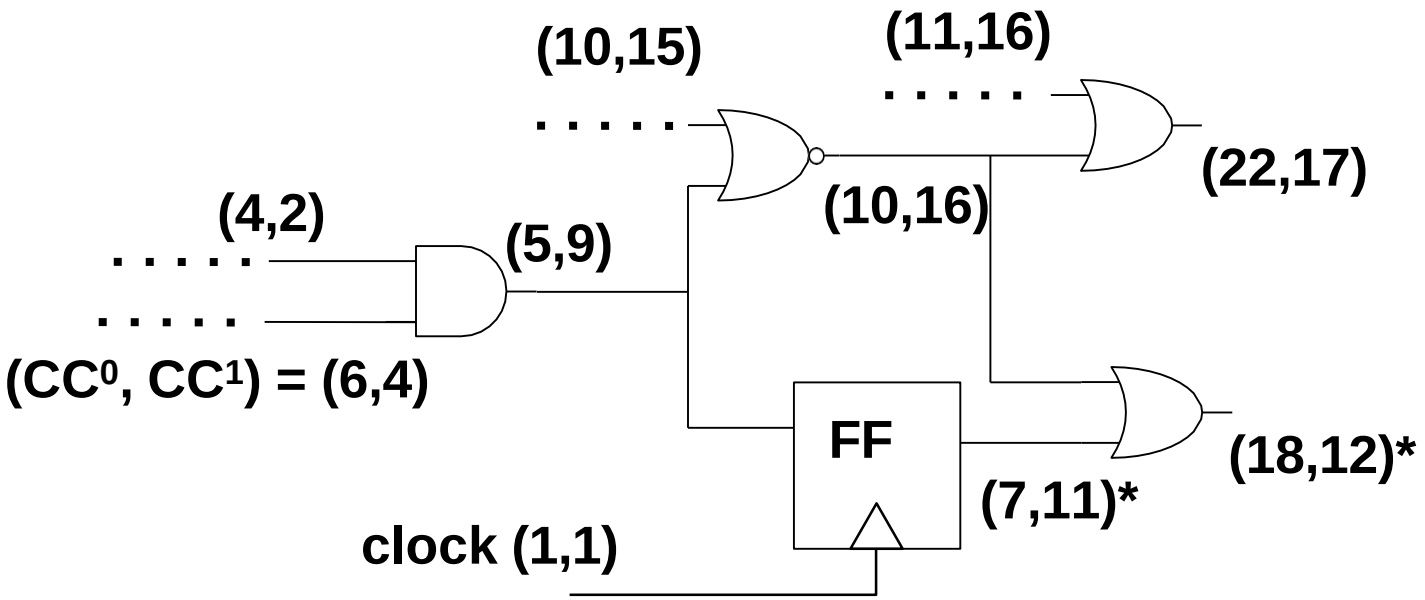
Drivability: $d^{0/1}$, $d^{1/0}$

- Estimates effort to propagate a D or D' from fault site to Y
  - $d^{0/1}(Y)$ = *drivability of D'* to node Y
  - $d^{1/0}(Y)$ = *drivability of D* to node Y

	$d^{0/1}$ (*Y*)	$d^{1/0}$ (Y)
	min {$CC^1(X_1) + d^{0/1}(X_2)$, $d^{0/1}(X_1)+CC^1(X_2)$, $d^{0/1}(X_1)+d^{0/1}(X_2)$}+1	min {$CC^1(X_1)+d^{1/0}(X_2)$, $d^{1/0}(X_1)+CC^1(X_2)$, $d^{1/0}(X_1)+d^{1/0}(X_2)$}+1
	min {$CC^0(X_1) + d^{0/1}(X_2)$, $d^{0/1}(X_1)+CC^0(X_2)$, $d^{0/1}(X_1)+d^{0/1}(X_2)$}+1	min {$CC^0(X_1)+d^{1/0}(X_2)$, $d^{1/0}(X_1)+CC^0(X_2)$, $d^{1/0}(X_1)+d^{1/0}(X_2)$}+1
	$d^{1/0}(X) + 1$	$d^{0/1}(X) + 1$
Y = FF out X = FF in	$d^{0/1}(X)$ + K (constant)	$d^{1/0}(X)$ + K (constant)

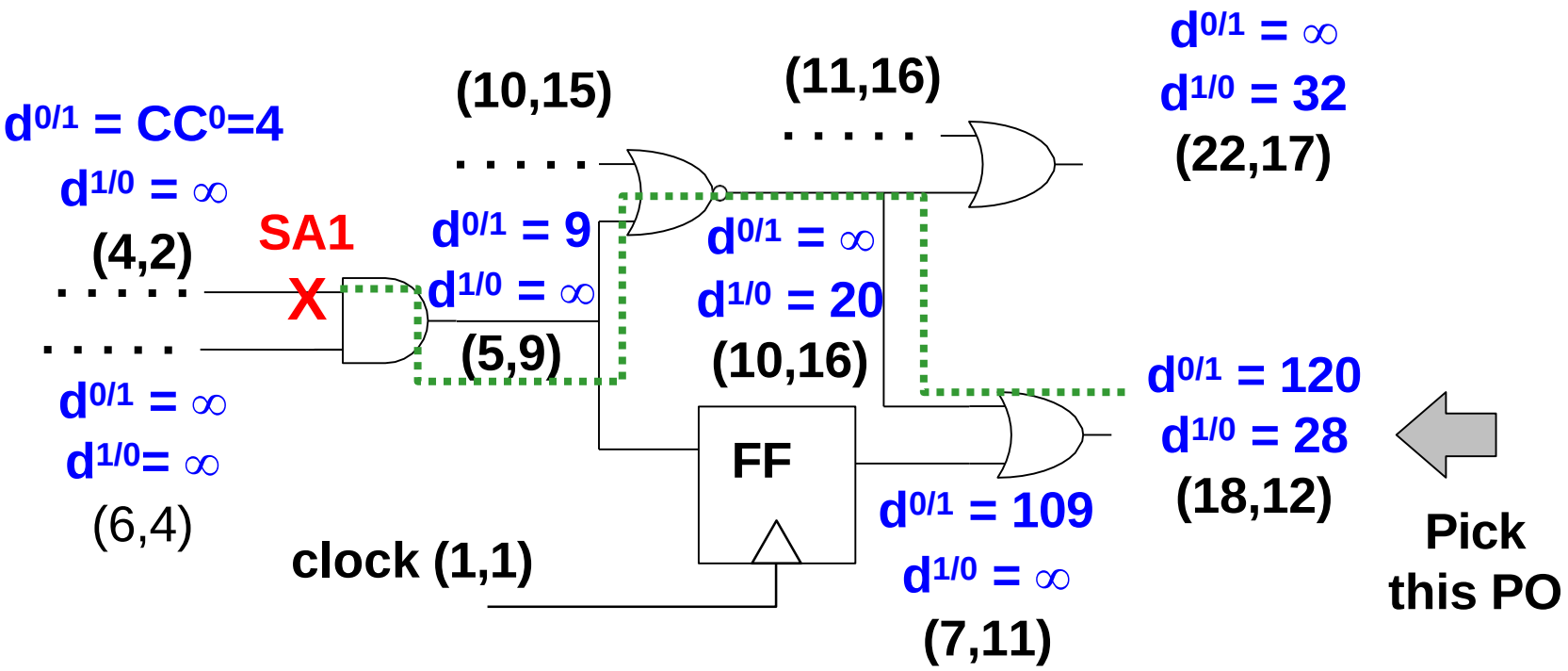
Example 1/2: SCOAP (BA Fig. 8.7)

- Fault-free circuit



* different from textbook

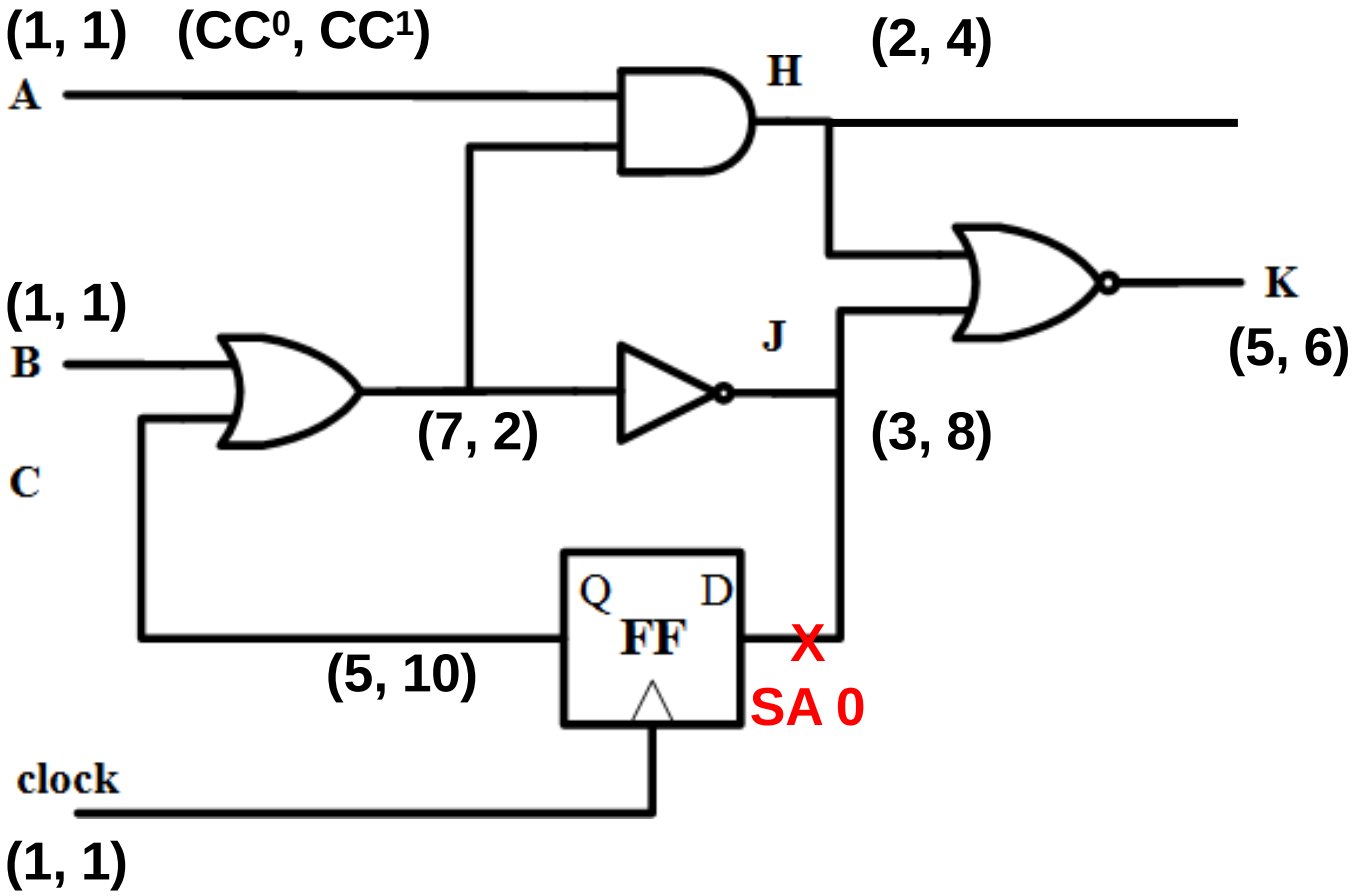
Example 2/2: Drivability (BA Fig. 8.7)



- K=100
- When D impossible, $d^{1/0} = \infty$

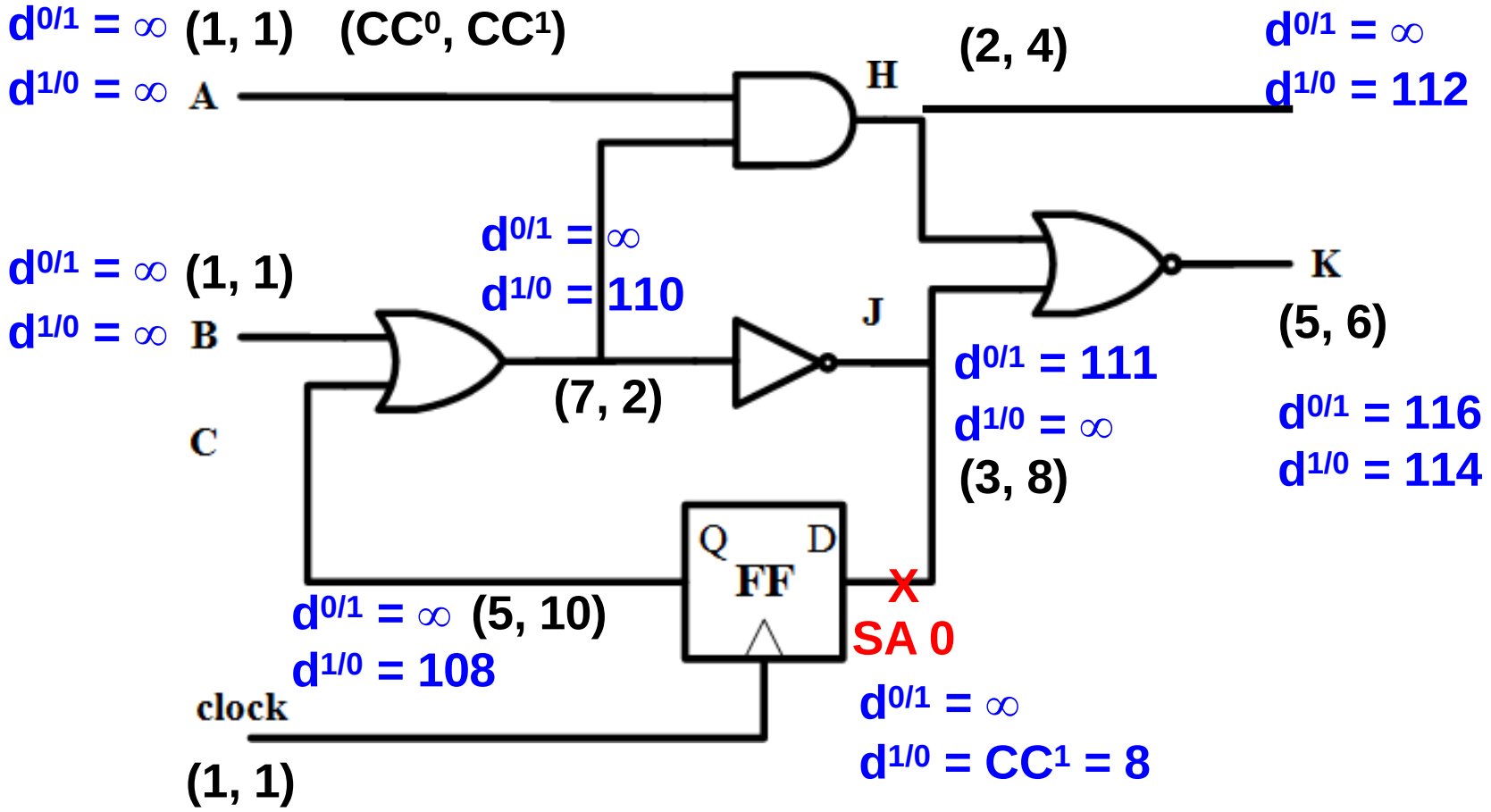
Quiz

Q: Based on SCOAP, please calculate drivability of SA0 fault. (Assume K=100)



Quiz

Q: Based on SCOAP, please calculate drivability of SA0 fault. (Assume K=100)



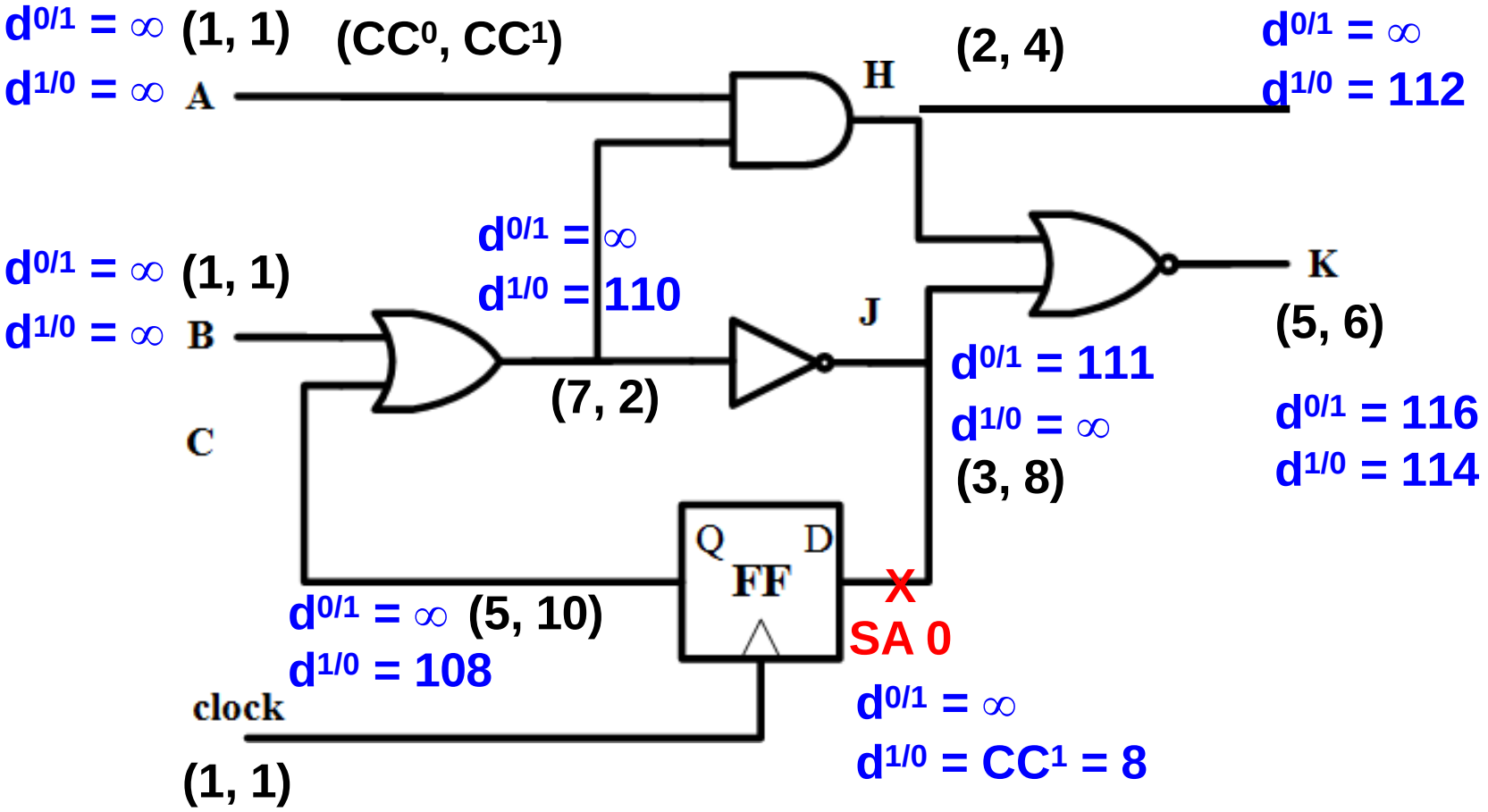
Summary

- ***Backward time frame processing*** **has advantages:**
  - **Fixed time frame**
  - **State repetition recognized**
- **EBT selects one path at a time**
- **BACK proposes drivability to select best PO**

	Advantages	Disadvantages	Example
Mixed F/B Time Frame	☺Reuse existing alg.	☹ Too many time frames	**Extended D**
Backward TF Only	☺Fixed time frames ☺State repetition recognized	☹ Too many paths!	**EBT** **BACK**
Forward TF Only	☺ No need to justify	☹ Hard to decide time frames	**FASTEST** (PODEM-like)

FFT1

- Do we need feedback to iteratively calculate Drivability?



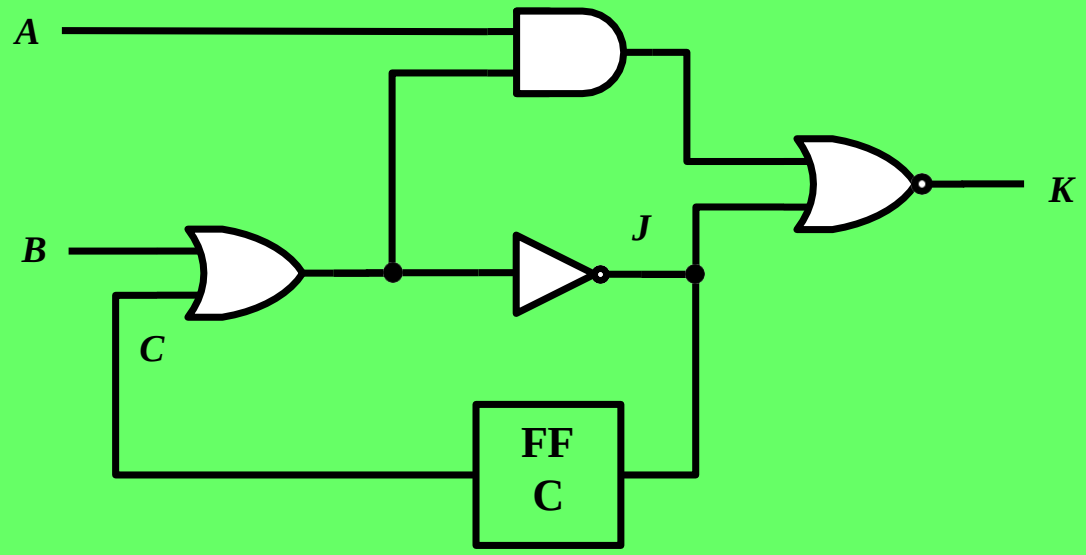
FFT2

- Q: For forward time frame processing, can we fix time frames, like EBT?

	Advantages	Disadvantages	Example
Mixed F/B Time Frame	☺Reuse existing alg.	☹ Too many time frames	Extended D
Backward TF Only	☺Fixed time frames ☺State repetition recognized	☹ Too many paths!	EBT BACK
Forward TF Only	☺ No need to justify	☹ Hard to decide time frames	PODEM-like

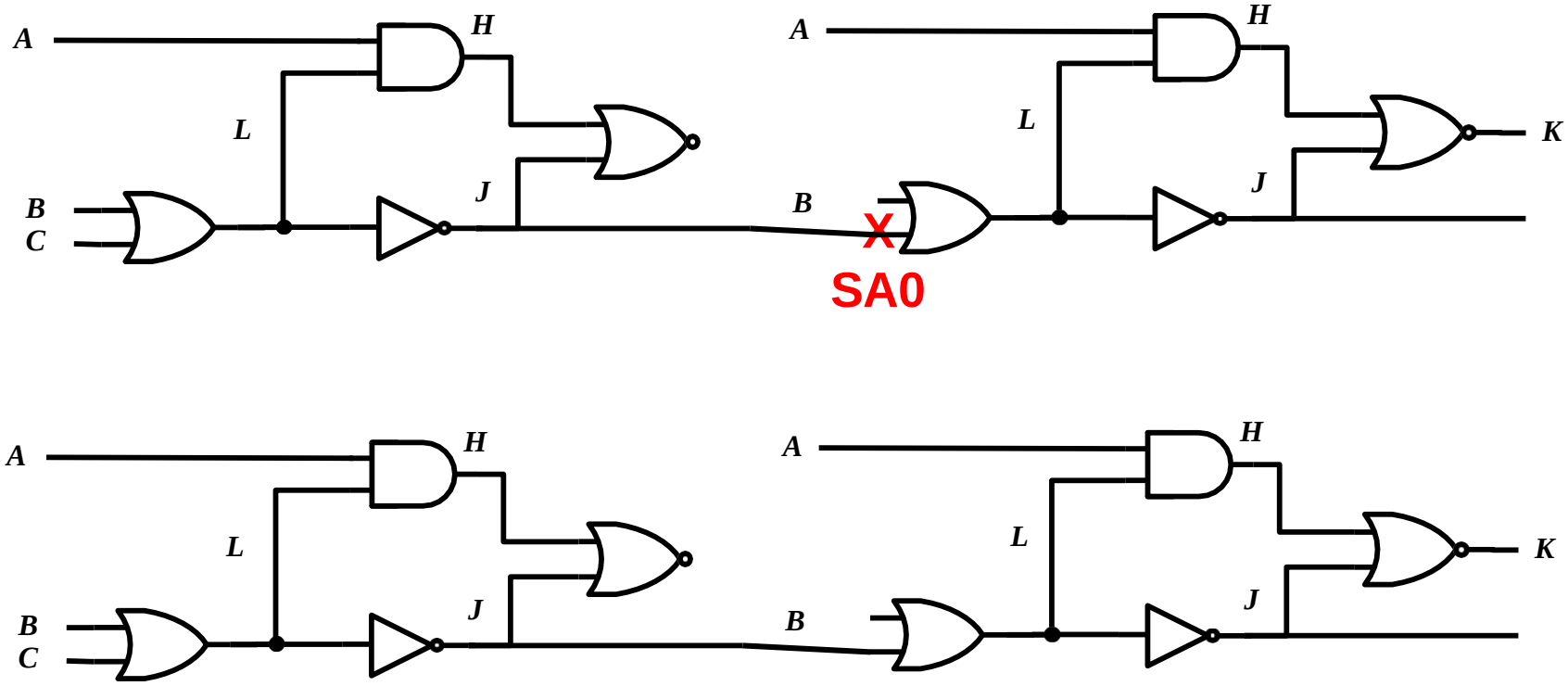
Quiz

Q: Please redraw this into two time frames



Quiz (cont'd)

Q: Use EBT to generate test patterns for SA0 fault



Drivability

- ***Drivability*** **estimates effort of propagating a D or D' from fault site to that signal**
  - **Similar to *SCOAP***
- **Review of SCOAP**

	$CC^0(Y)$	$CC^1(Y)$
$Y = X_1$ AND X_2	$\min[CC^0(X_1), CC^0(X_2)] + 1$	$CC^1(X_1) + CC^1(X_2) + 1$
$Y = X_1$ OR X_2	$CC^0(X_1) + CC^0(X_2) + 1$	$\min[CC^1(X_1), CC^1(X_2)] + 1$
$Y = X'$	$CC^1(X) + 1$	$CC^0(X) + 1$
Primary inputs	1	1

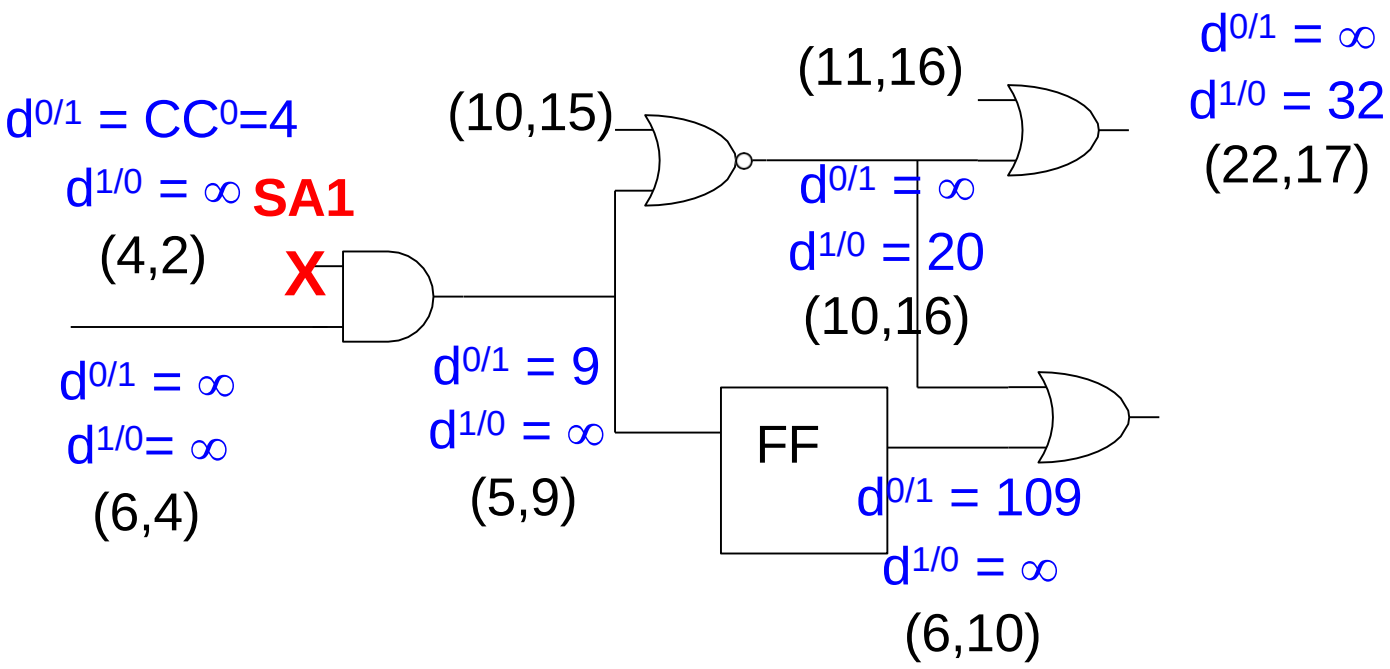
Drivability is like SCOAP for a Specific Fault

Drivability: d(0/1) & d(1/0)

- $d^{0/1}(Y)$ = *drivability of D'*
  - Effort to bring node Y to D' when the fault presents
- $d^{1/0}(Y)$ = *drivability of D*

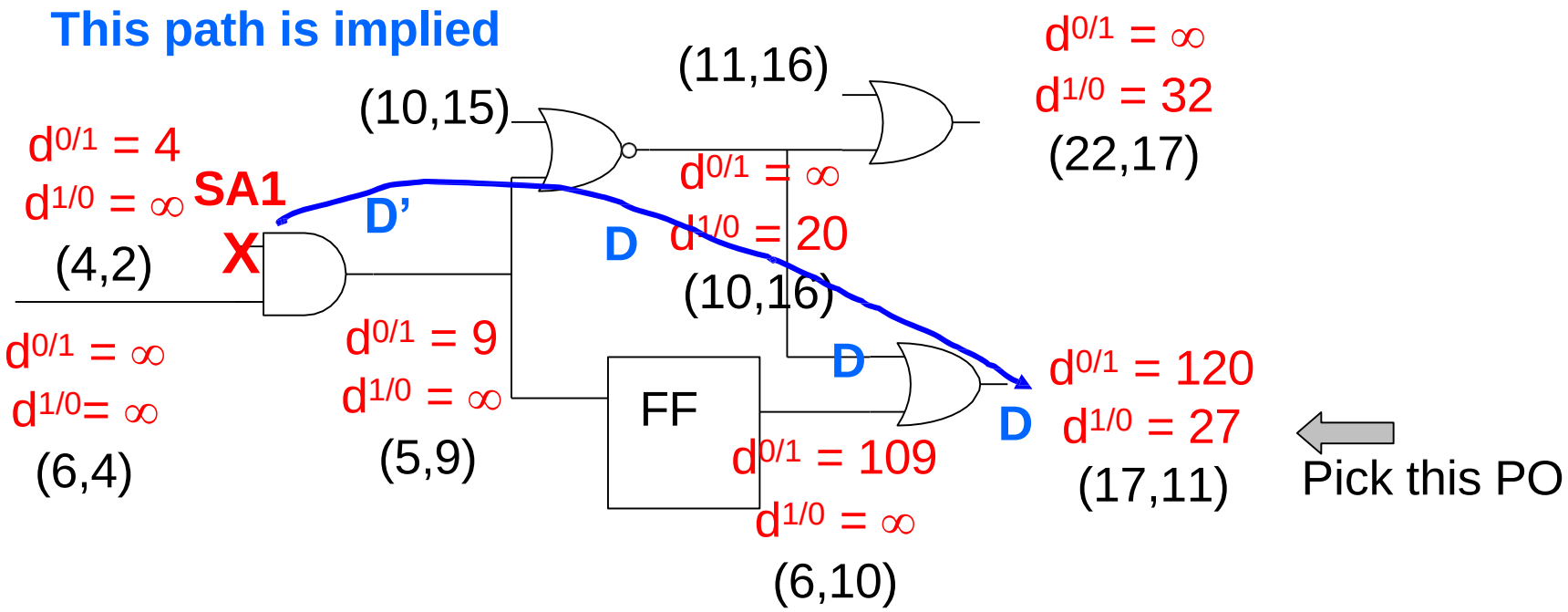
	$d^{0/1}(Y)$	$d^{1/0}(Y)$
Y = X1 AND X2	$\min\{CC^1(X_1) + d^{0/1}(X_2),$ $d^{0/1}(X_1)+CC^1(X_2),$ $d^{0/1}(X_1)+d^{0/1}(X_2)\}+1$	$\min\{CC^1(X_1)+d^{1/0}(X_2),$ $d^{1/0}(X_1)+CC^1(X_2),$ $d^{1/0}(X_1)+d^{1/0}(X_2)\}+1$
Y = X1 OR X2	$\min\{CC^0(X_1) + d^{0/1}(X_2),$ $d^{0/1}(X_1)+CC^0(X_2),$ $d^{0/1}(X_1)+d^{0/1}(X_2)\}+1$	$\min\{CC^0(X_1)+d^{1/0}(X_2),$ $d^{1/0}(X_1)+CC^0(X_2),$ $d^{1/0}(X_1)+d^{1/0}(X_2)\}+1$
Y = X'	$d^{1/0}(X) + 1$	$d^{0/1}(X) + 1$
Y = FF out X = FF in	$d^{0/1}(X)$ + K (constant)	$d^{1/0}(X)$ + K (constant)

Drivability Example 1/2 (BA Fig. 8.7)



- **K=100**
- **When D impossible, $d^{1/0} = \infty$**

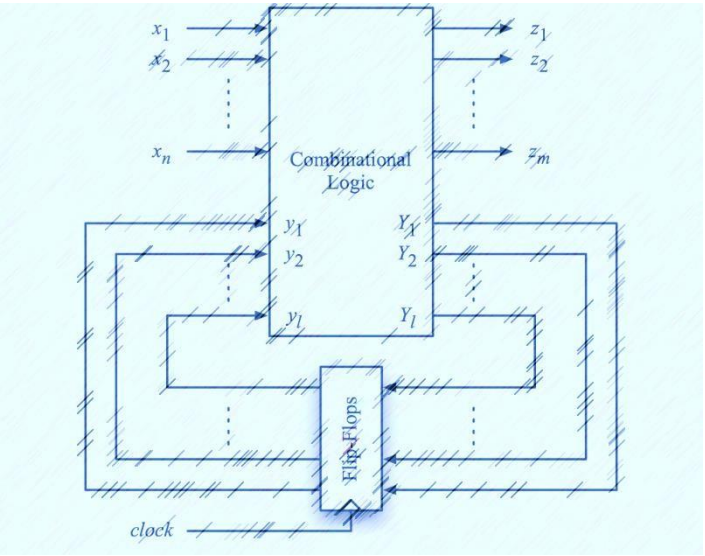
Drivability Example 2/2 (BA Fig. 8.7)



- **To justify a D or D' at gate output, select gate input with smallest drivability as D or D' input**

Sequential ATPG

- Introduction
- Time-frame expansion methods
- **Simulation-based methods (* not in exam)**
  - **CONTEST [Agrawal & Cheng 88]**
  - **Genetic Algorithm**
- Issues of Sequential ATPG
- Conclusions



Simulation-Based Methods

- **Idea: use logic/fault simulators to guide ATPG [Seshu 62]**
  - **Simulation is faster than ATPG**
- **Approach**
  - **Generate candidate test vectors**
  - ***Fitness**** **of candidates evaluated by logic or fault simulation**
  - **Select best candidate based on a certain *cost function***
- **Advantage:**
  - **No time frame expansion. Easy memory management**

Simulate Many Vectors and Choose Best

CONTEST– Concurrent Test Generator for Sequential Circuits [Agrawal & Cheng 89]

- Based on event-driven concurrent fault simulator
- Search for test vectors guided by cost functions
- Three phases
  - ① Initialization
  - ② Concurrent fault detection
  - ③ Single fault detection

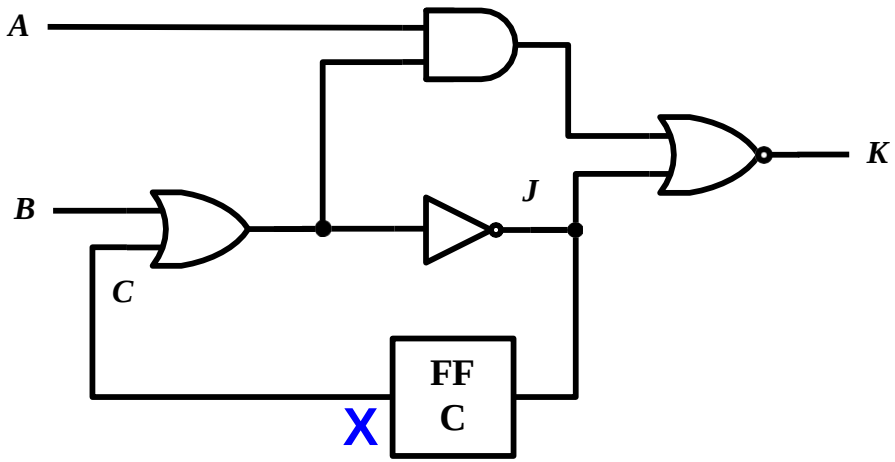
1. Initialization Phase

- Start with arbitrary test vector
  - Start with FFs in unknown states
- Use logic simulation (not fault simulation)
  - Cost = number of FFs in unknown state
  - *Trial vectors* are generated by single-bit change of the current vector. A trial vector is accepted and becomes the current vector if it lowers the cost
- Stop this phase when cost drops below a desired value

QUIZ

Q1: Initially the FF is unknown. Given three trial vectors, please simulate the circuit and decide their costs, where cost = number of unknown FF.

Q2: Which trial vector would you pick?

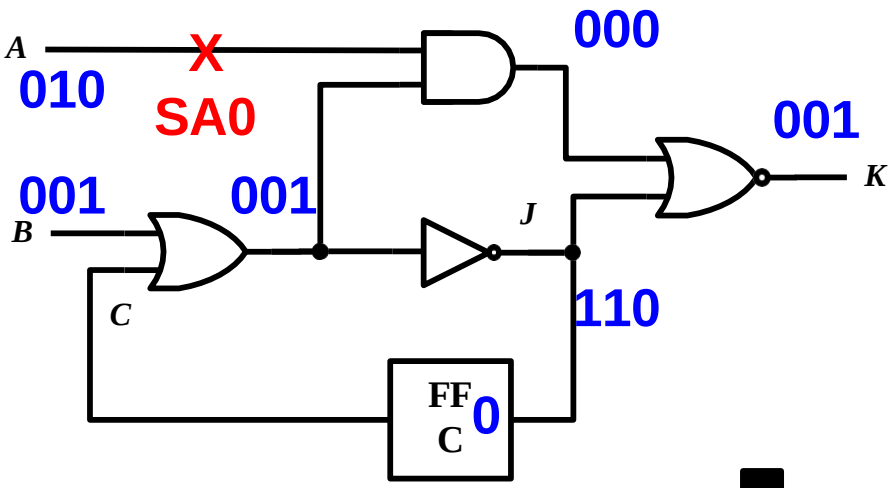


AB	cost = number of unknown FF
00	1
01	
10	

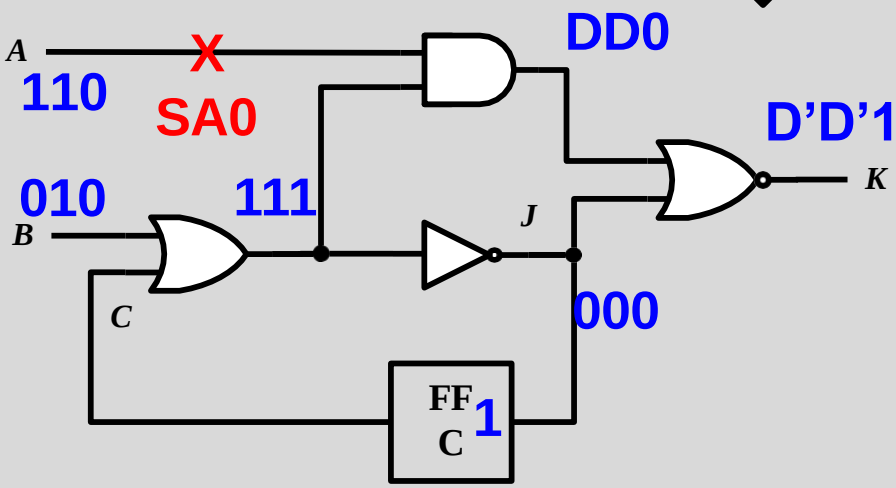
2. Concurrent Fault Detection Phase

- Start with fault simulation of generated initialization sequence
  - Detected faults are dropped from the fault list
- Compute the cost of the last vector
  - Cost of an undetected fault *f*
    - COST(*f*) = minimum distance of its fault effect to a PO
    - distance = level of logic gates
  - Cost of a vector
    - Sum of costs of all undetected faults
- Trial vectors are generated by single-bit change
  - Only accept the vectors that reduce the cost

Example



AB	cost = distance of D or D' to PO
00	∞
10	2
01	∞



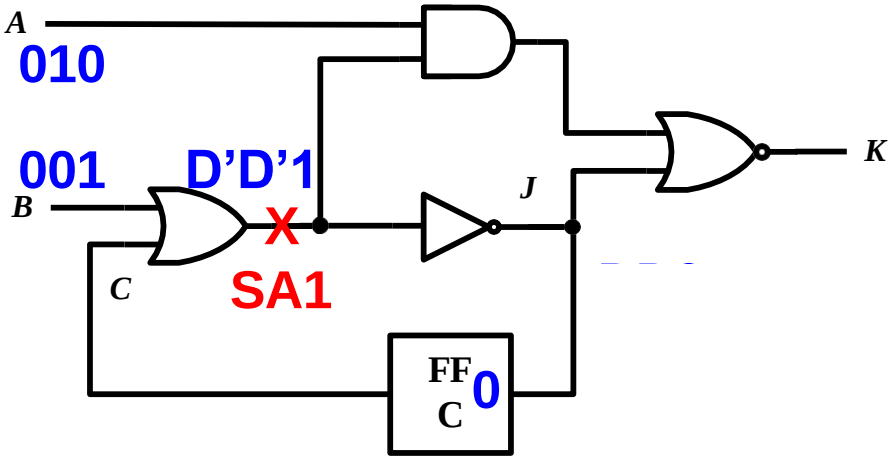
AB	cost = distance of D or D' to PO
10	0
11	0
00	∞

Test Vector Sequence: AB = (01), 10, 11

QUIZ

Q1: Given FF initial state is zero, please evaluate the cost of three trial vectors: 00, 10, 01

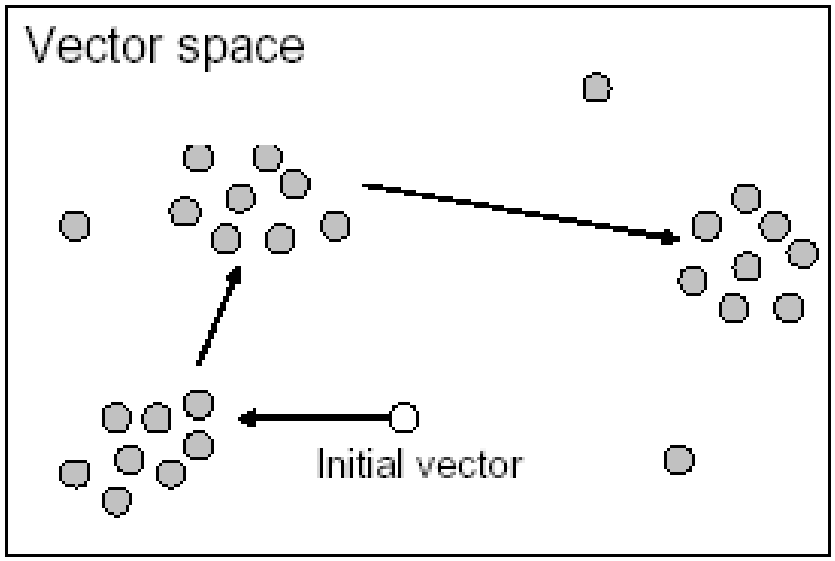
Q2: Which test vector would you pick?



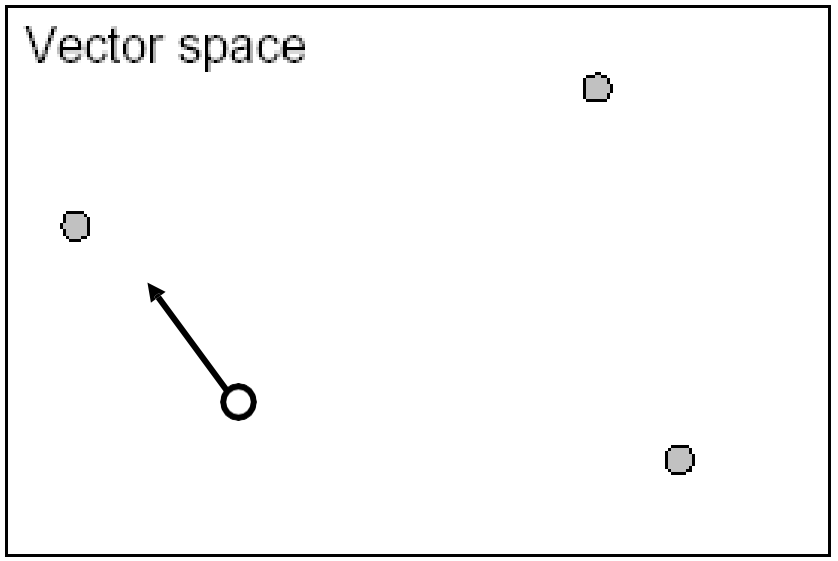
AB	cost = distance of D or D' to PO
00	
10	
01	

Need Phase 3

- **Experience shows test patterns for all stuck-at faults are usually clustered instead of being evenly distributed**
- **When only a few faults are left, their tests will be isolated vectors and we need a different test generation strategy**



Phase 2: Concurrent Fault Detection



Phase 3: Single Fault Detection

3. Single Fault Detection Phase

- Start with any vector
- Generate new vectors by single-bit change to reduce cost of the selected fault until it is detected
  - The lowest cost fault is picked first

- Cost of a fault f at signal line g is
  - If not activated yet:

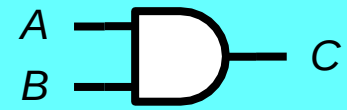
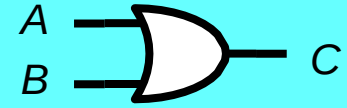
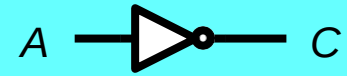
    $KC_A(f) + C_P(f)$

    K = constant ; C_A=activation ;C_p=propagation cost
  - If activated:

    $Min(C_P(i))$, $i \in$ the set of inputs to signal g

C_A and DC

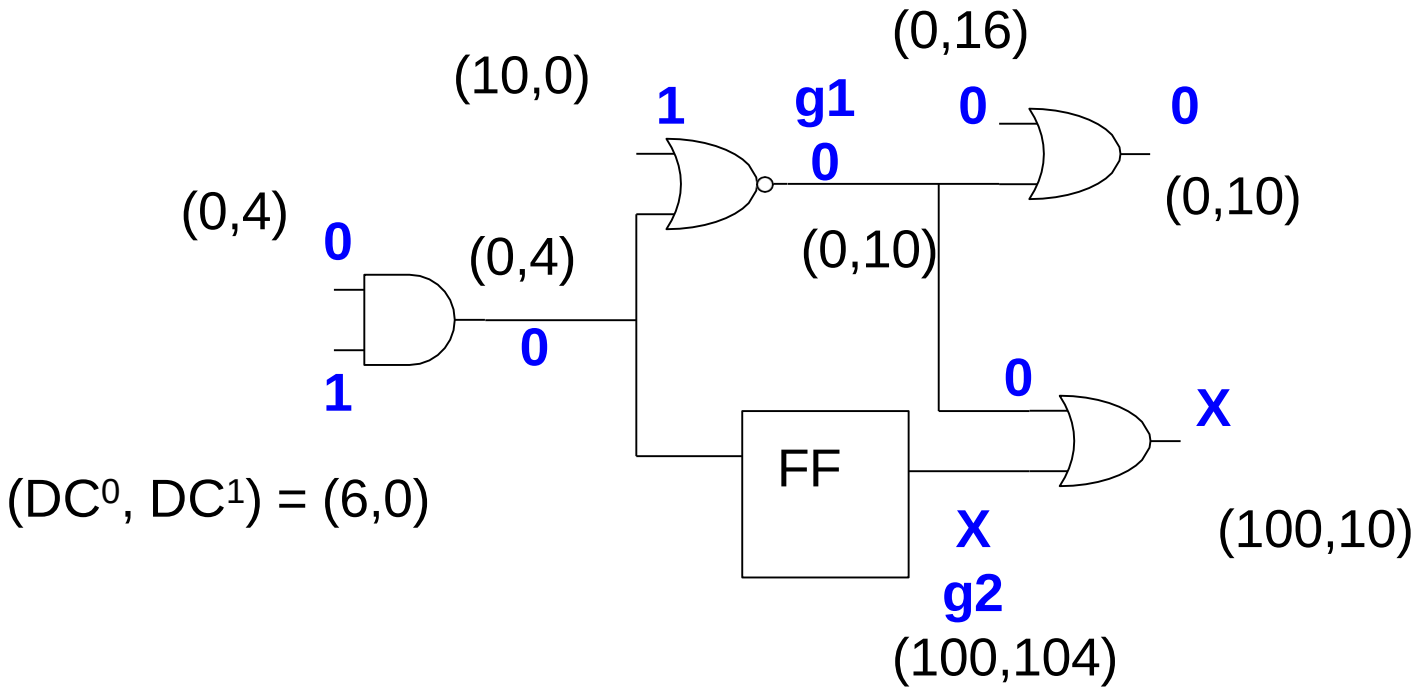
- *Activation Cost, C_A*
  - C_A(*g* stuck-at-*v*) = $DC_{v'}$(*g*) = dynamic controllability of line *g* at *v'*
- *Dynamic Controllability, DC*
  - Similar to sequential controllability in SCOAP except logic values known

	$DC^0(C)$	$DC^1(C)$
	$\min[DC^0(A),DC^0(B)]$, if C=1 or x 0 , if C=0	$DC^1(A) + DC^1(B)$, if C=0 or x 0 , if C=1
	$DC^0(A) + DC^0(B)$, if C=1 or x 0 , if C=0	$\min[DC^1(A),DC^1(B)]$, if C=0 or x 0 , if A=1
	$DC^1(A)$, if C=1 or x 0 , if C=0	$DC^0(A)$, if C=0 or x 0 , if C=1
Primary inputs	1 , if C=1 or x 0 , if C=0	1 , if C=0 or x 0 , if C=1
C = FF(A)	$DC^0(A)+K$, if C=1 or x 0 , if C=0	$DC^1(A)+K^*$, if C=0 or x 0 , if C=1

* K is a chosen constant

C_A and DC Example

- $C_A(g_1$ stuck-at 0) = $DC_1(g_1)$ =10 ← easier
- $C_A(g_2$ stuck-at 1) = $DC_0(g_2)$ =100

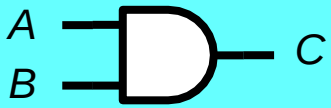
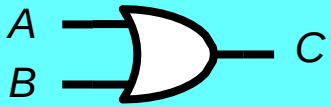
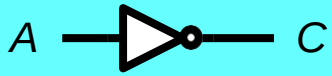
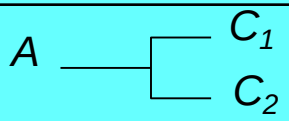


Propagation Cost, C_P

- $C_p(g)$ = Dynamic Observability of node *g*
- Dynamic observability (DO)
  - Similar to combinational observability in SCOAP
  - Measure the effort to observe the fault on a given node
    - the number of gates between *N* and PO's, and
    - the minimum number of PI assignments required to propagate the logical value on node *N* to a primary output.

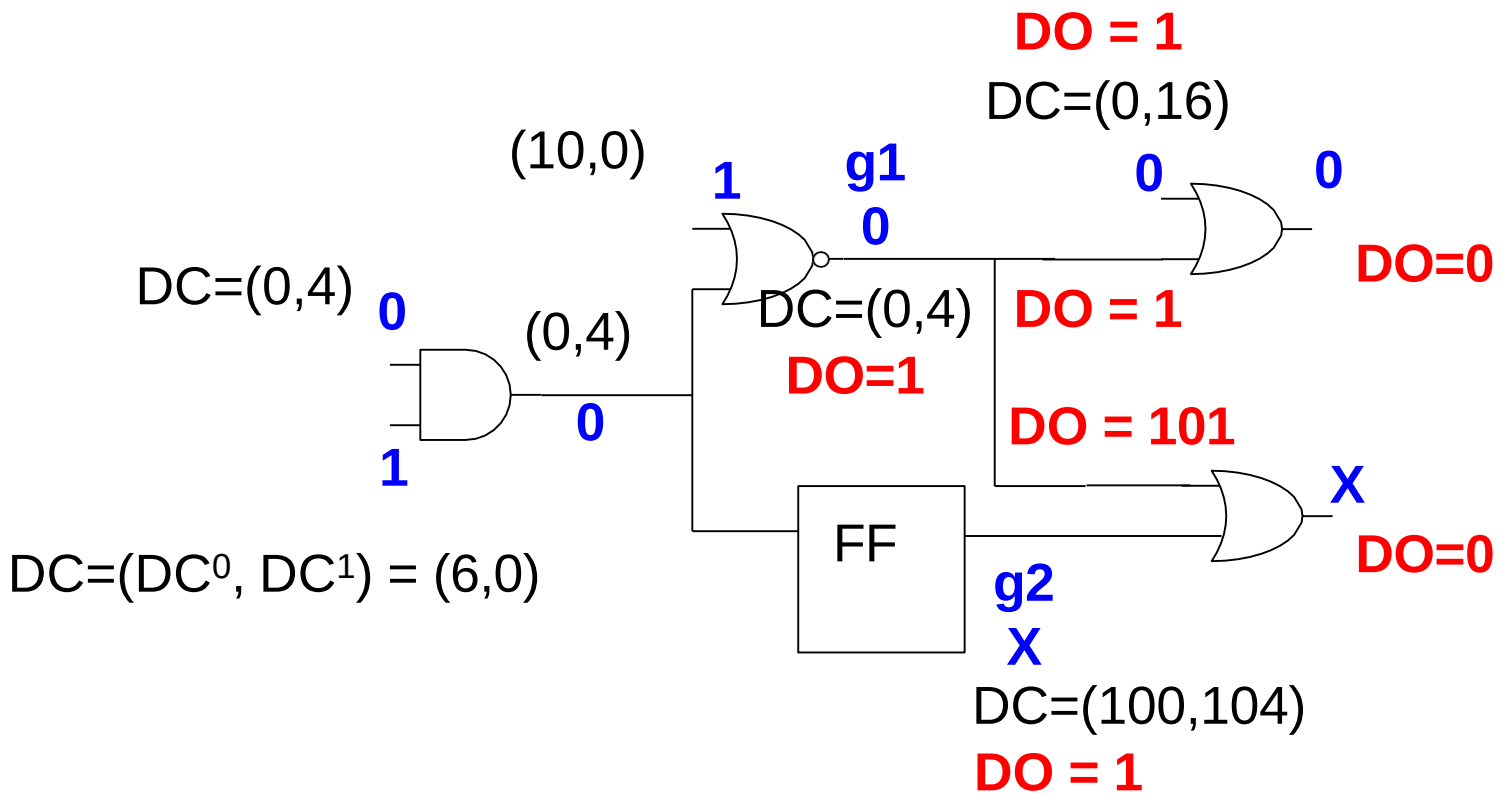
Dynamic Observability (DO)

- **Similar to combinational observability in SCOAP**

	DO(*A*)
	DO(C) + DC^1(*B*) + 1
	DO(*C*) + DC^0(*B*) + 1
	DO(*C*) + 1
	min[DO(C_1),DO(C_2)]
Primary outputs	**0**

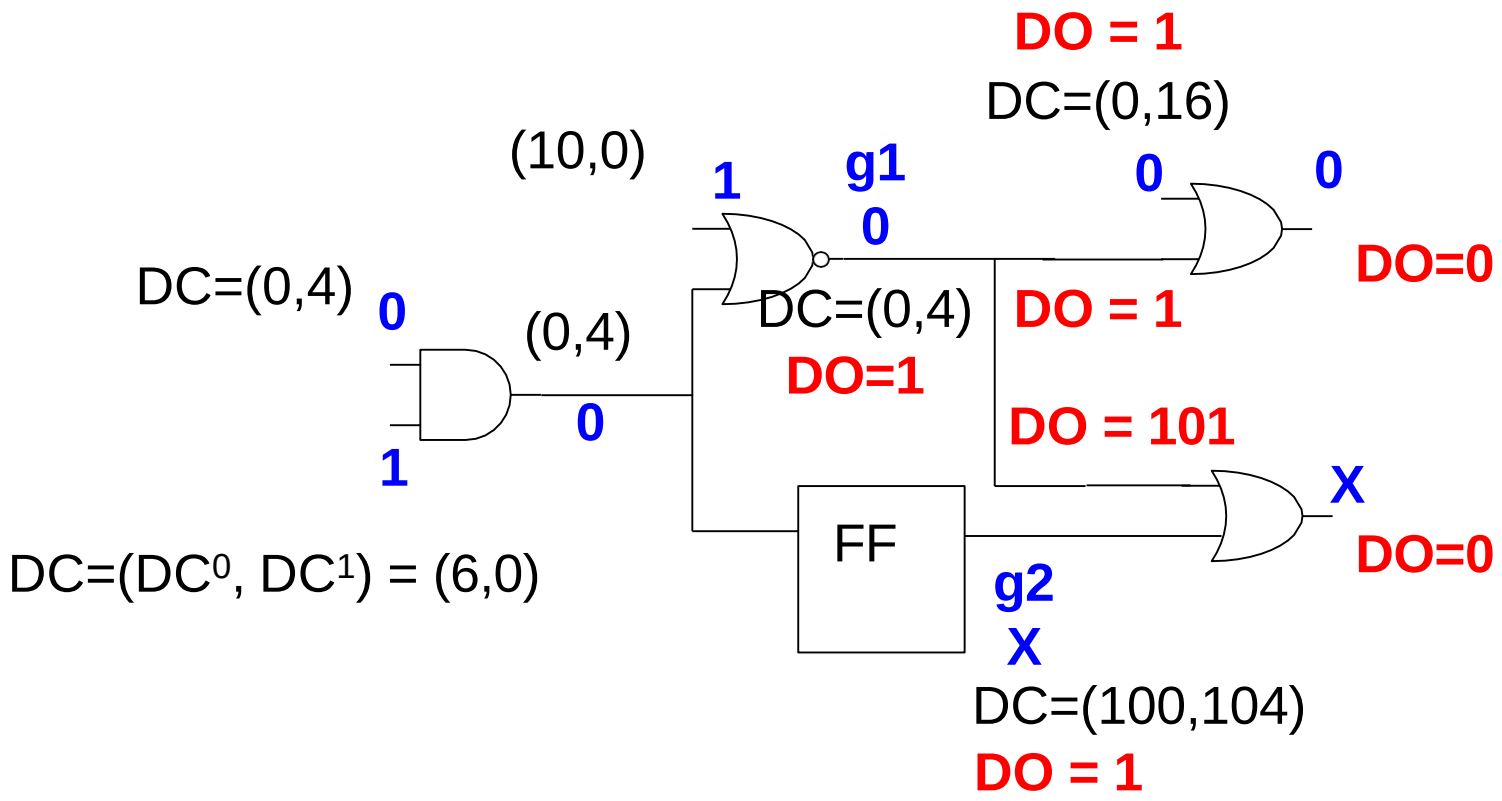
C_p and DO Example

- $C_p(g_1) = DO(g_1) = 1$
- $C_p(g_2) = DO(g_2) = 1$



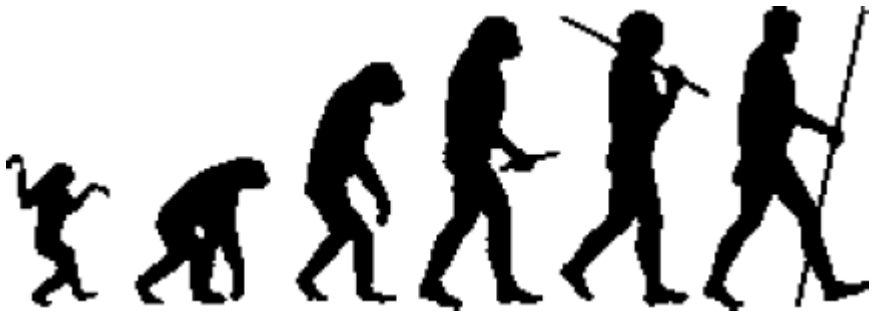
Total Cost

- **Fault g_1: $C_A = 10$, $C_p = 1$**
- **Fault g_2: $C_A = 100$, $C_p = 1$**
- **Choose g1 SA0 as target fault to generate test vector**



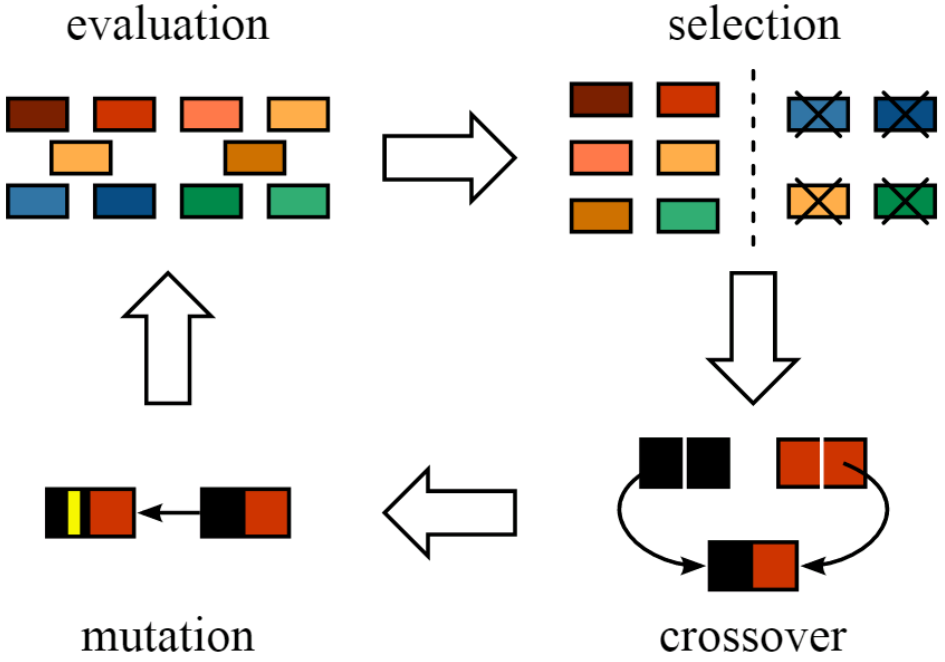
Sequential ATPG

- Introduction
- Time-frame expansion methods
- **Simulation-based methods**
  - **CONTEST**
  - **Genetic Algorithm**
- Issues of Sequential ATPG*
- Conclusions



Genetic Algorithms (GA) [Holland 1975]

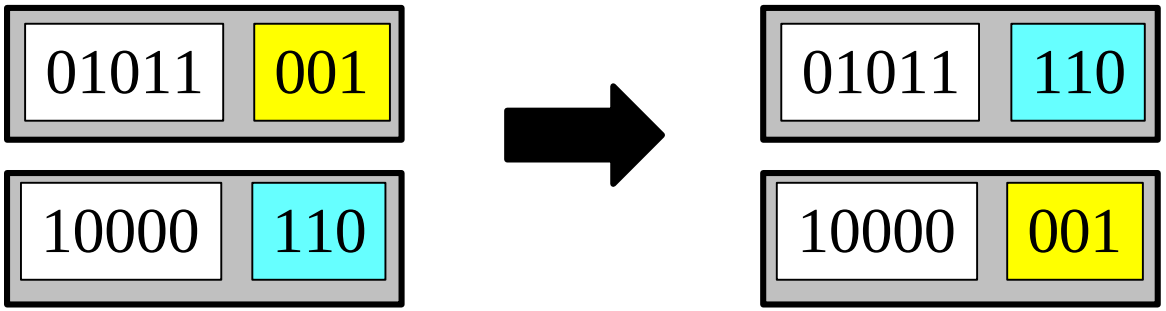
- General Principle: Survival of fittest(s)
  - Keep a *population* of feasible solutions, not just one
  - Parent population generates child population
    - by gene *crossover, mutation* etc
  - Select only best children, remove weak children
  - Repeat the above for many generations



[www.jade-cheng.com]

Crossover and Mutation

- Test vectors are represented by bit-stream "gene"
- *Crossover*: Two feasible solutions generate child by switching gene

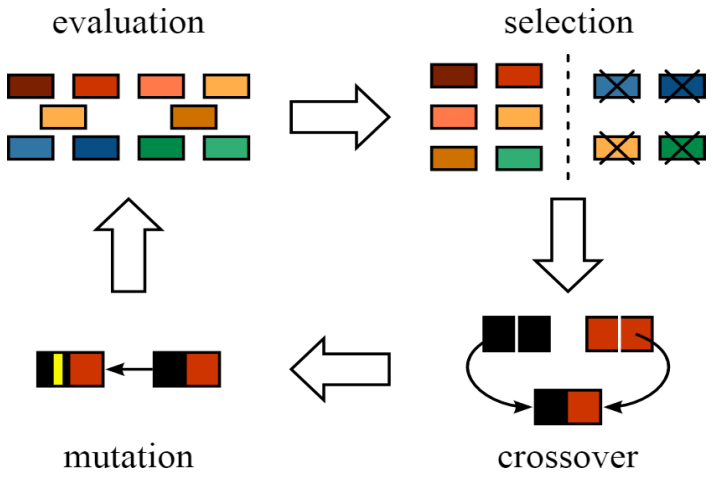


- *Mutation*: some gene can change by a random probability

Pseudo Code of GA

```
GENETICALGORITHM
1   pop = set of initial solutions
2   do
3     childpop = ∅
4     for (i = 1 to (n x pop.size))  // n times size
5       crossover = random 0 or 1
6       if (crossover)
7         parent1 = random_choose(pop)
8         parent2 = random_choose(pop)
9         child = crossover(parent1, parent2)
10      else   // mutate
11        parent = random_choose(pop)
12        child = mutate(parent)
13      childpop = childpop ∪{child}
14    pop = evaluate&select(childpop)
15  while (!stop)
16  return (best solution)
```

- **Need to decide**
- **1. initial solution**
- **2. corssover/ mutation**
- **3. evaluate & select**
- **4. stop criterion**

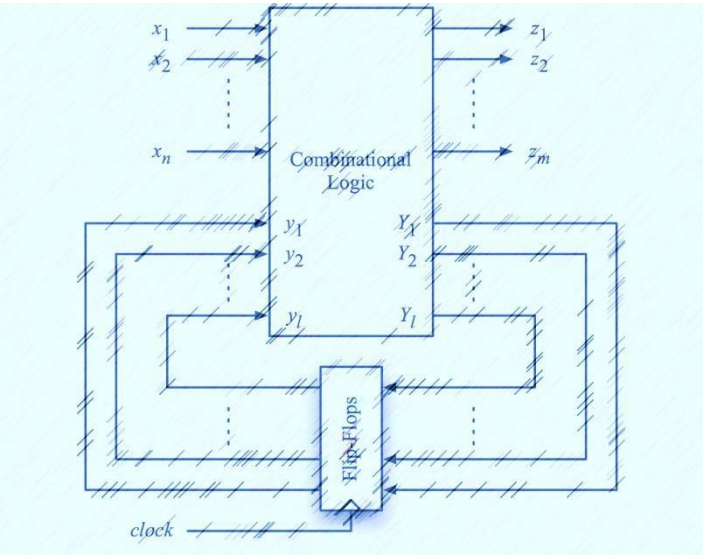


Summary

- Simulation-based methods
  - Randomly generate many trial test vectors
  - Evaluate test vectors by simulation and pick the best
  - Need many testability measure to help smart decision
- Advantages
  - Better memory management than time frame expansion
  - Timing can be considered
  - Use genetic algorithm to optimize
- Disadvantages
  - Cannot identify untestable faults
  - Test length can be longer than time frame expansion

Sequential ATPG

- Introduction
- Time-frame expansion methods
- Simulation-based methods*
- Issues of Sequential ATPG* (not in exam)
- Conclusions



Issues of Sequential ATPG

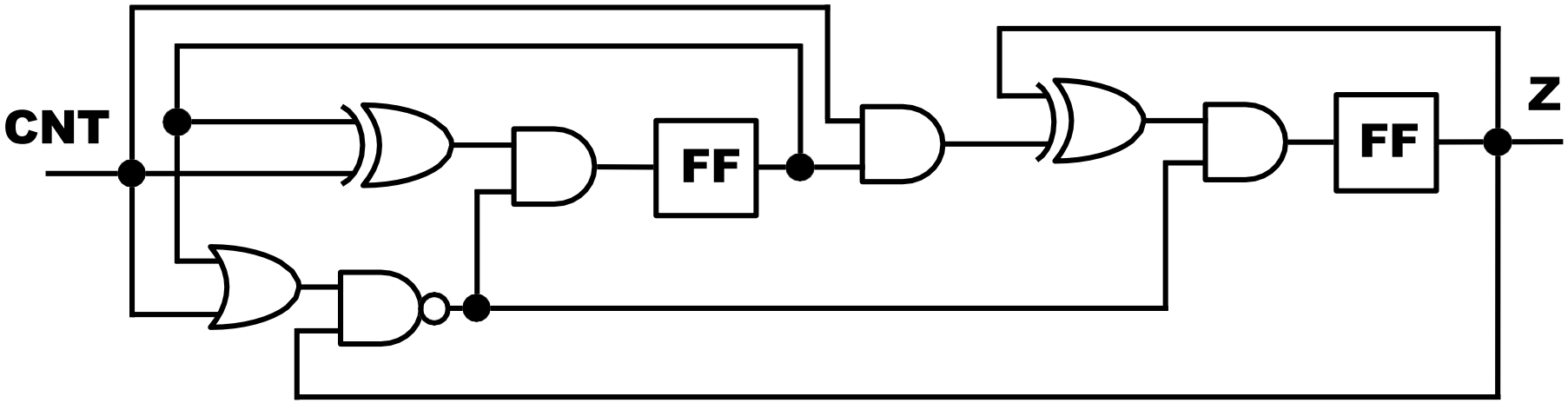
① Ckts. without initialization input

② Potentially detected faults can escape test

③ Asynchronous ckt. requires special attention

① Circuit w/o Initialization Input

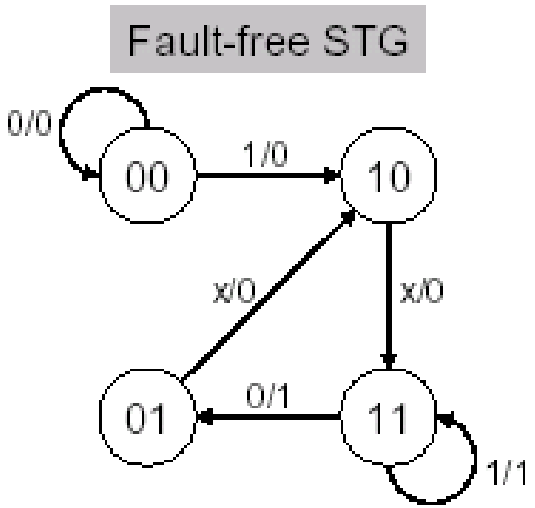
- **Example**
  - **Mod 3 counter (BA Fig. 8.12)**
    - **10→ 00→ 01→ 10 → …..**

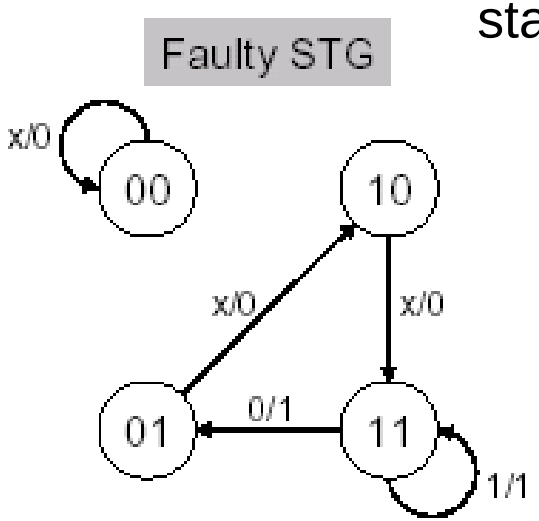


Cannot Find Initialization Sequence

② Potentially Detected Faults

- **DEF: faults that may or may not be detected**
  - Also see Video 5.7
- **Example: detection of fault depends on the power-up states**
  - $Y_1 = xy_1y_2 + y_1y_2' + y_1'y_2$ (+ $xy_1'y_2'$) faulty STG missing term
  - $Y_2 = y_1y_2 + y_1y_2'$
  - $Z = y_1y_2$

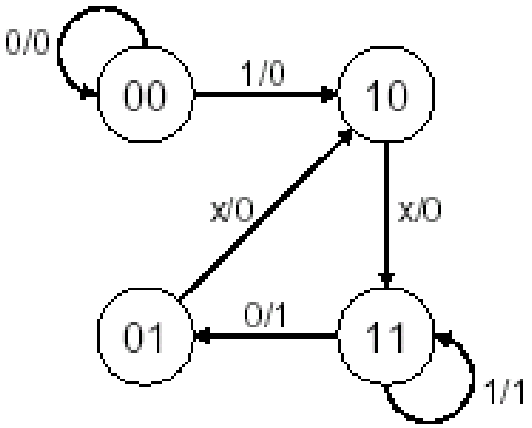




state transition graph

X/Z
(Input/output)
y1y2
(states)

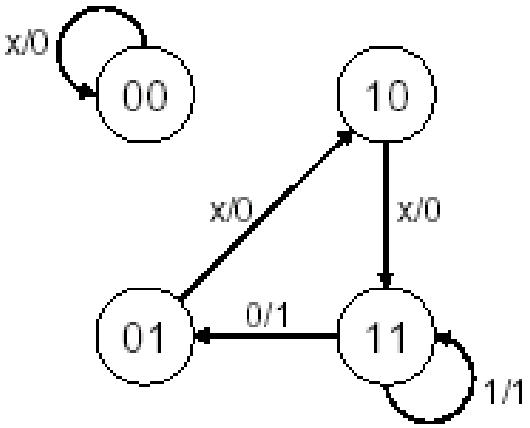
Fault-free STG



X/Z

- {0 1 1} initializes the circuit to state 11
- Good response to input {0110} is
  - z = 1001

Faulty STG



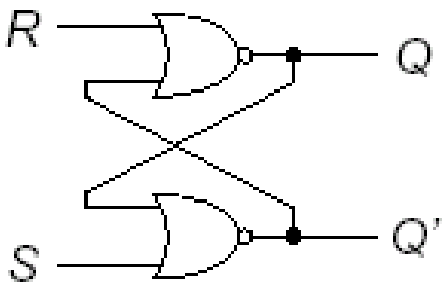
- {0 1 1} cannot initialize circuit
- Response at z to {0110}:
  - Power-up state 00,
    - z = 0000
    - fault detected
  - Power-up state 01,10, or 11
    - z = 1001
    - fault NOT detected

Fault Detection is Uncertain

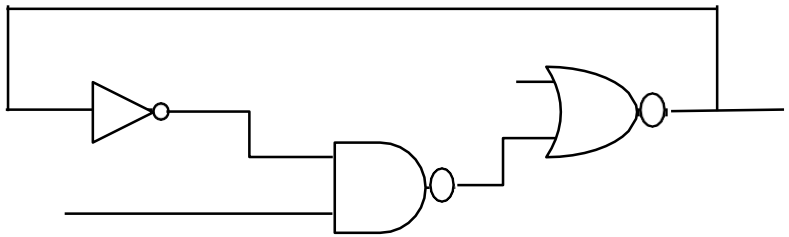
③ Asynchronous Circuits

- **No explicit clock. Signals can change asynchronously**
- **Timing can be difficult to model**
  - **Test patterns generated may cause *races* and *hazards***
  - **Need to verify test sequence with a fault simulator**
- **Example: Circuits with combinational loop**

local feedback



global feedback



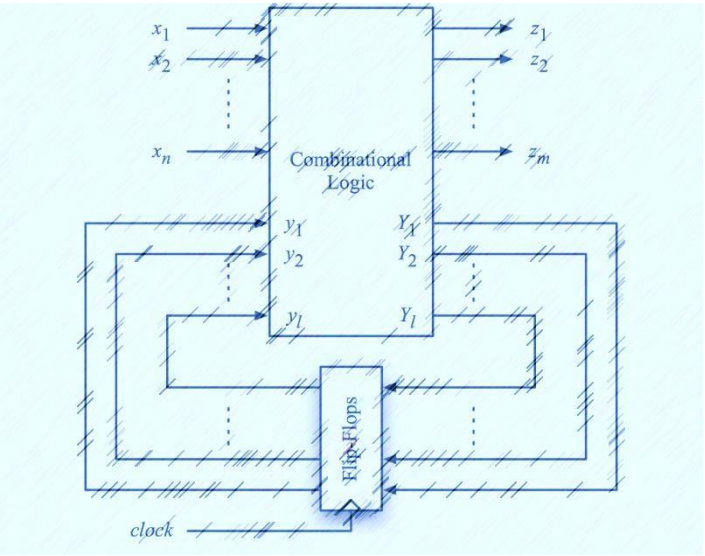
QUIZ

Q: Suppose you are a designer. Which of the following circuits you should avoid in order to avoid sequential ATPG?

A) Combinational circuits with feedback loops
B) Non-scan flip-flops without reset pins (video 11.8)
C) Circuits with many different types of flip-flops
D) Circuits with SRAM memories

Sequential ATPG

- Introduction
- Time-frame expansion methods
- Simulation-based methods*
- **Issues of Sequential ATPG***
- **Conclusions**



Concluding Remarks

- Sequential ATPG
  - Generate PI patterns, observe PO. No scan allowed
- Benefits
  - Enable at-speed testing
  - Handles partial scan or non-scan circuits
- Problems
  - Low fault coverage, long run time, large memory
- Techniques
  - Time frame expansion (EBT, BACK, Extended-D …)
  - Simulation (CONTEST)
- Current DFT & ATPG Practice
  - Use as much combinational ATPG as possible
  - Use sequential ATPG only when necessary

Sequential ATPG is Difficult. Need DfT! (Ch11)

References

- [Agrawal 88] Agrawal, Vishwani D., K-T. Cheng, and Prathima Agrawal. "CONTEST: A concurrent test generator for sequential circuits," Design Automation Conference, 1988. Proceedings., 25th ACM/IEEE. IEEE, 1988.
- [Cheng 88] Cheng, W-T. "The BACK algorithm for sequential test generation," Computer Design: VLSI in Computers and Processors, 1988. ICCD'88., Proceedings of the 1988 IEEE International Conference on. IEEE, 1988.
- [Holland 1975] Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA.
- [Kubo 68] H. KUBO "A procedure for generating test sequences to detect sequential circuit failures," NEC Res. & Dev., (Oct. 1968), 69 –78.
- [Marlett 78] Marlett, Ralph A. "EBT: A comprehensive test generation technique for highly sequential circuits," Proceedings of the 15th Design Automation Conference. IEEE Press, 1978.
- [Muth 76] P. Muth"A nine-valued circuit model for test generation," IEEE Trans. Comput., 25 (6), pp. 630–636, 1976.
- [Huffman 53] D. A. Huffman, "The Synthesis of Sequential Switching Circuits," MIT Thesis, 1953.

Commercial Tools

- Mentor Graphics
  - Flextest
- Synoposys
  - Tetramax
- Syntest
  - turboscan

Academic Tools

- Time-Frame Expansion
  - ESSENTIAL [89]
  - FASTEST[89]
  - HITEC [Niermanh & Patel 91]
  - Lee-Reddy [91]
- Genetic Algorithm
  - CRIS [Saab & Abraham 96]
  - GATEST [Rudnick 97]
  - GATTO [96]
  - STRATEGATE [97]