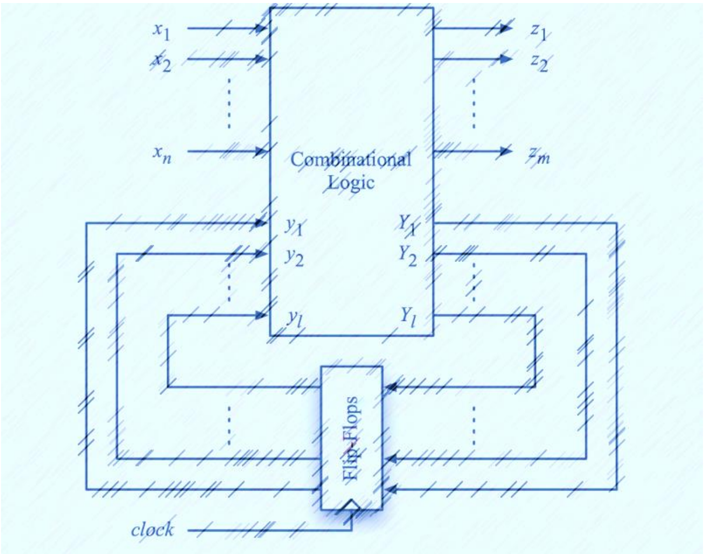
Sequential ATPG

- 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

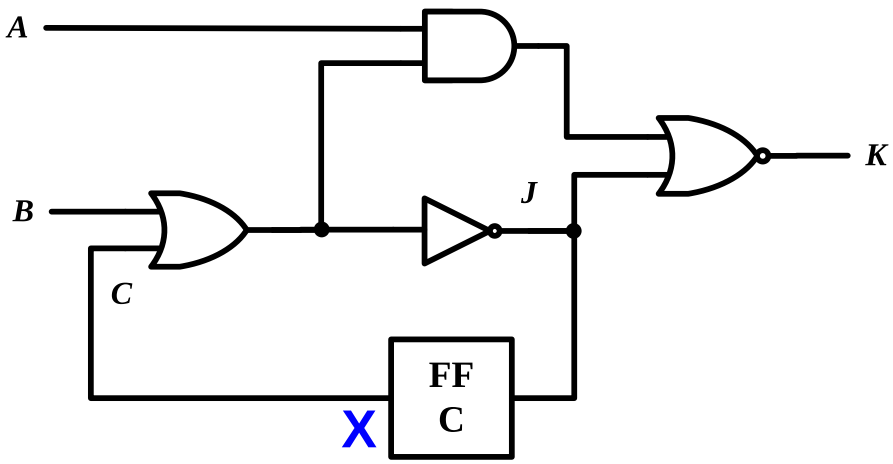
1. Initialization Phase

- 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?

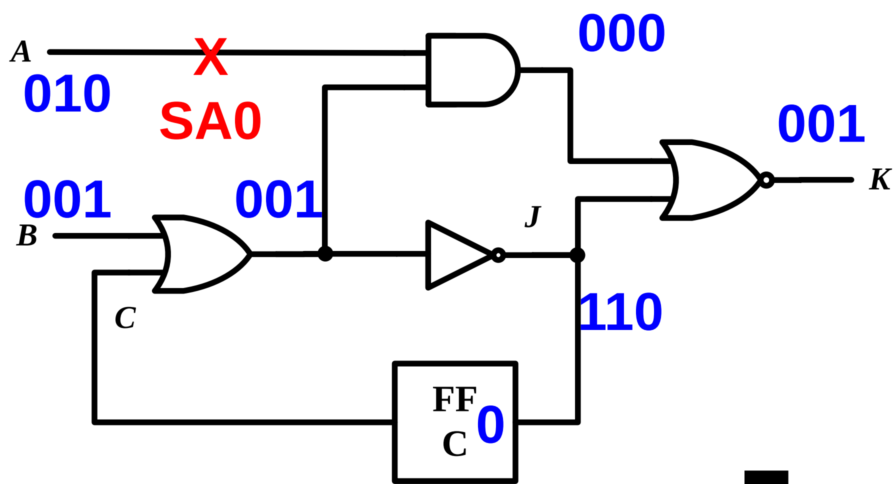


AB	cost = number of unknown FF
00	1
01	
10	

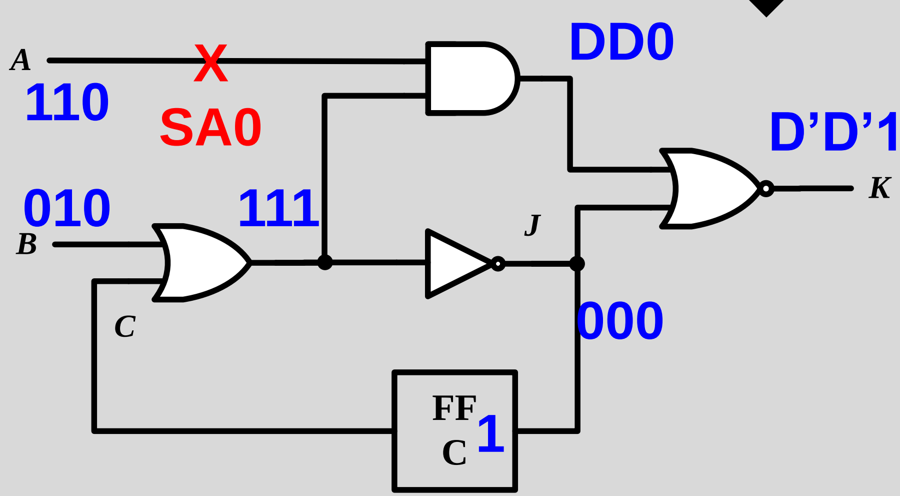
2. Concurrent Fault Detection Phase

- 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	∞



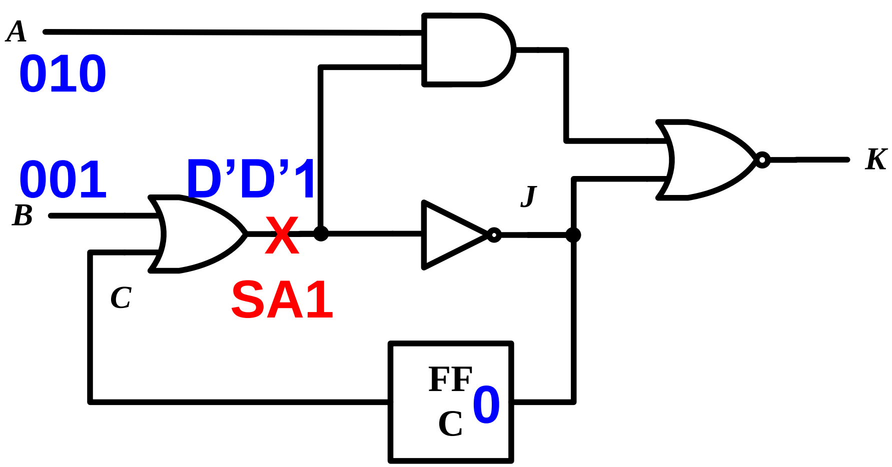
AB	cost = distance of D or D’ to PO
10	0
11	0
00	∞

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

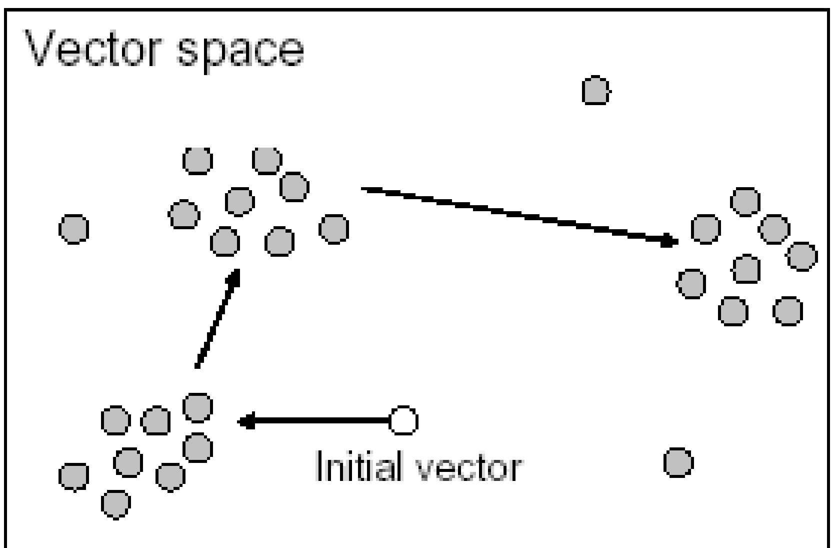
Q2: Which test vector would you pick?



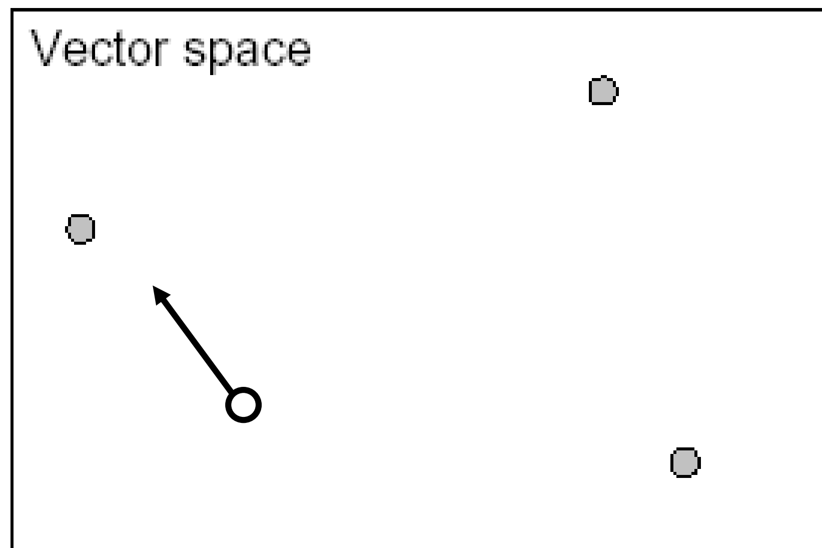
AB	cost = distance of D or D' to PO
00	
10	
01	

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:

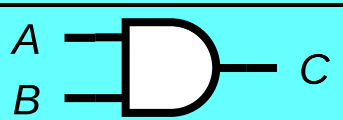
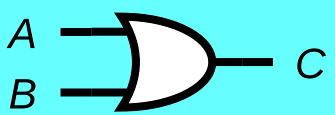
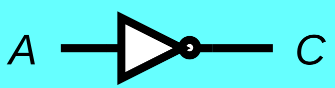
    $KC_A(f) + C_P(f)$

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

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

C_A and DC

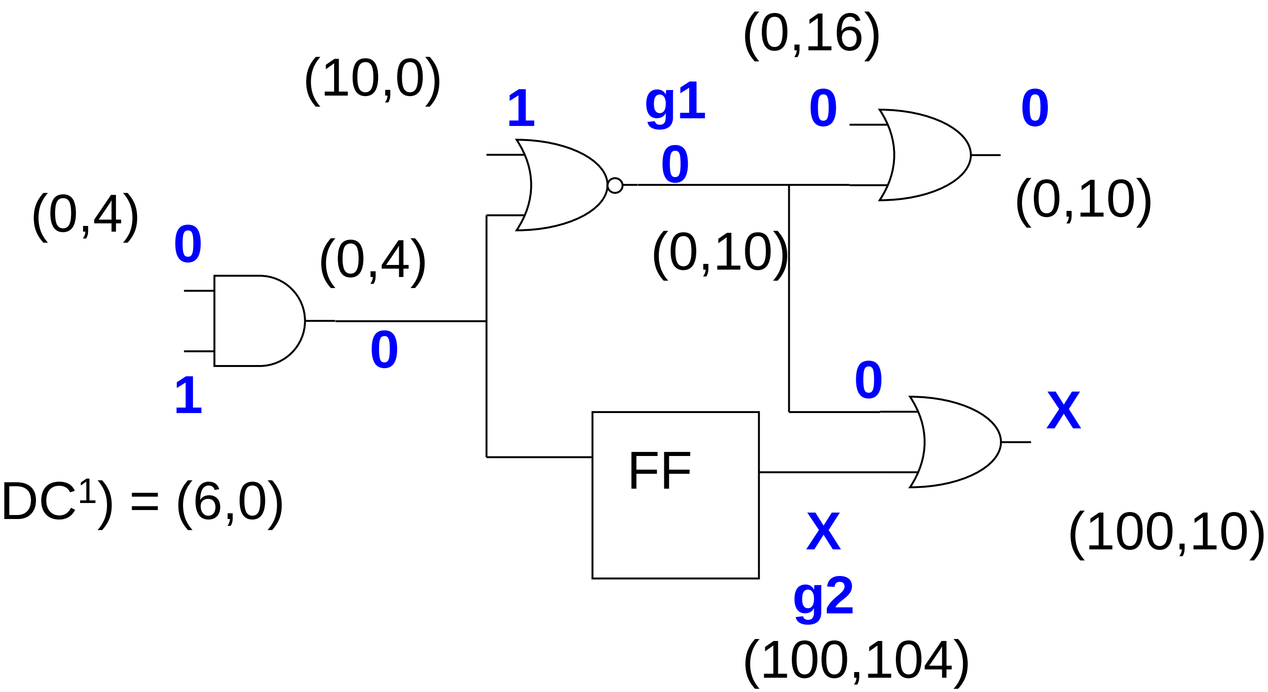
- ***Activation Cost, C_A***
  - C_A(*g* stuck-at-*v*) = $DC_{v'}$(*g*) = dynamic controllability of line *g* at *v'*
- ***Dynamic Controllability, DC***
  - Similar to sequential controllability in SCOAP except logic values known

	$DC^0(C)$	$DC^1(C)$
	$\min[DC^0(A), DC^0(B)]$, 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**



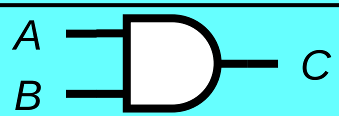
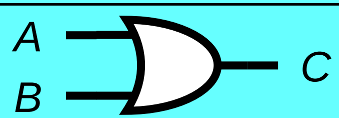
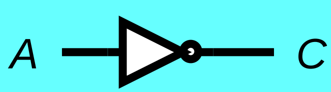
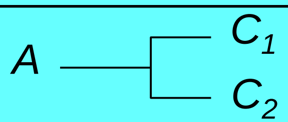
$(DC^0, DC^1) = (6,0)$

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.

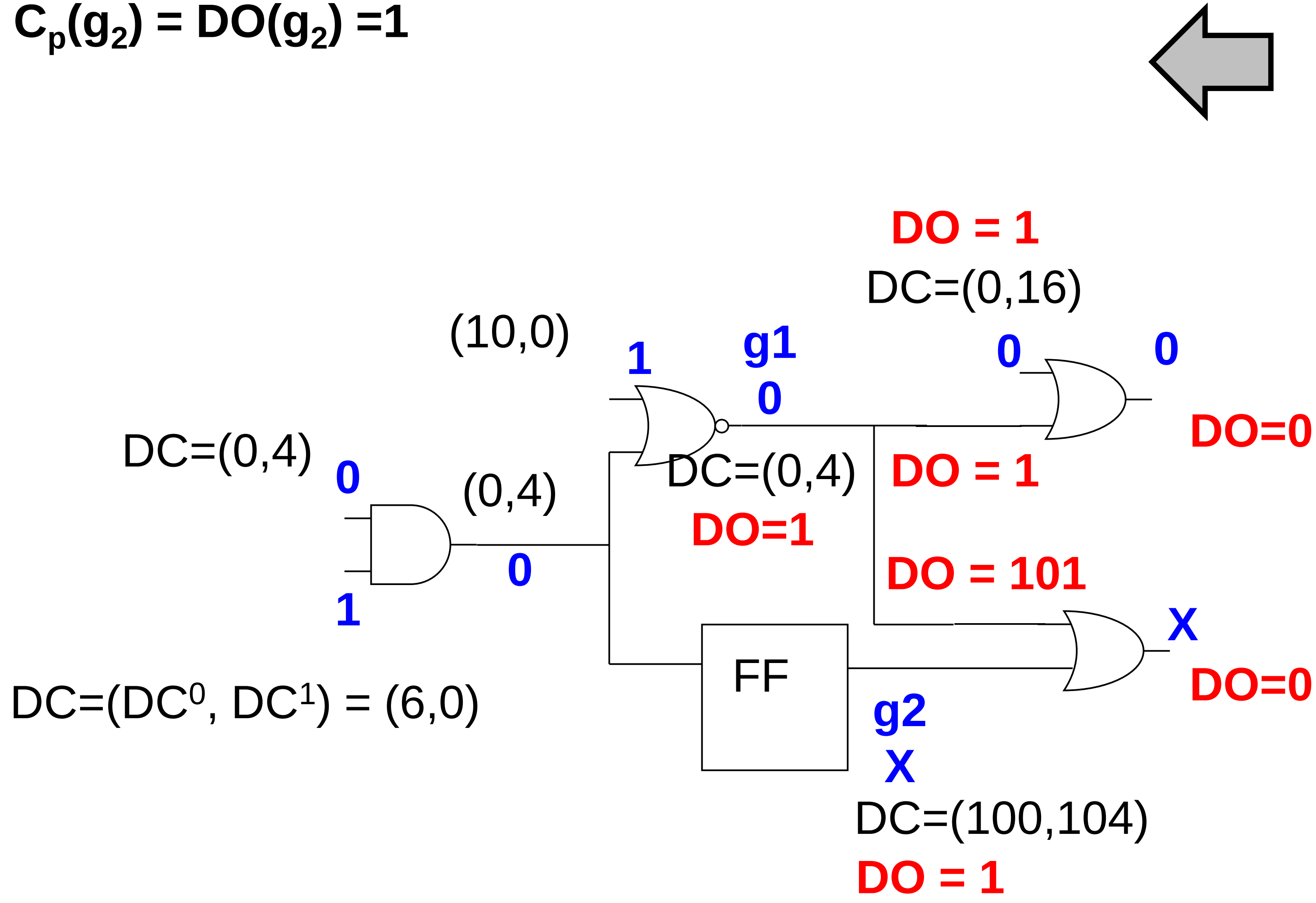
Dynamic Observability (DO)

- **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**

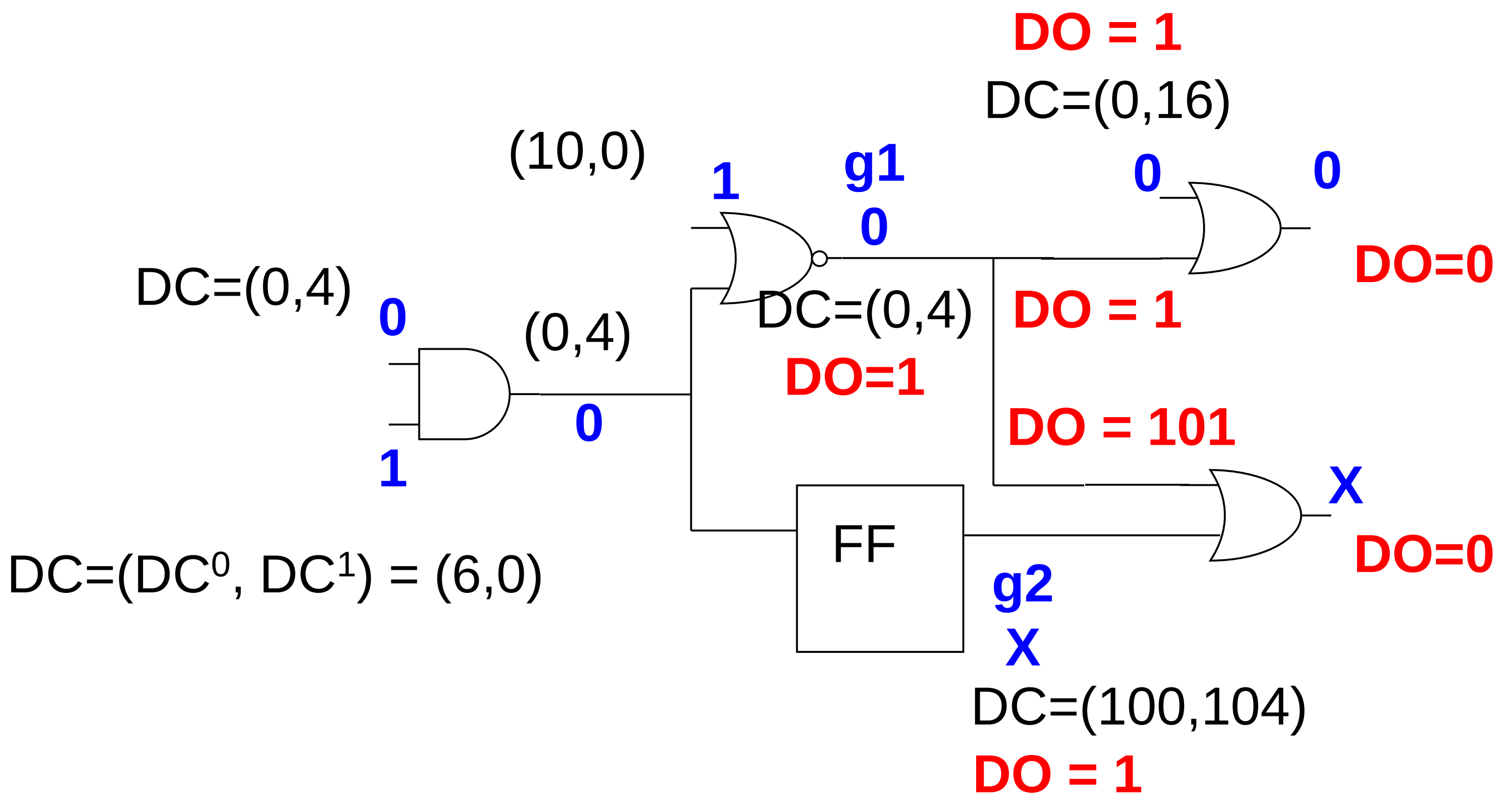
C_p and DO Example

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



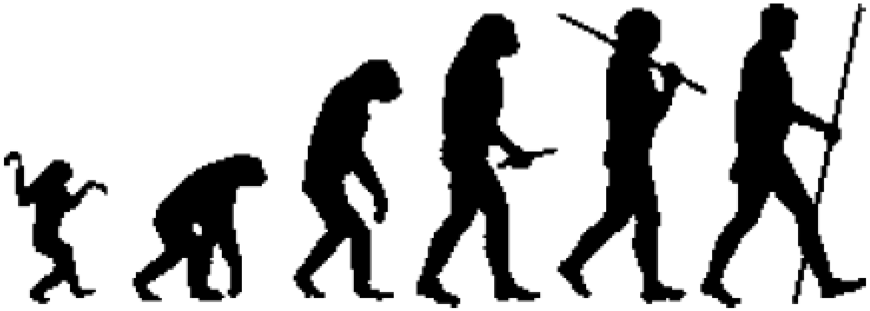
Total Cost

- **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**



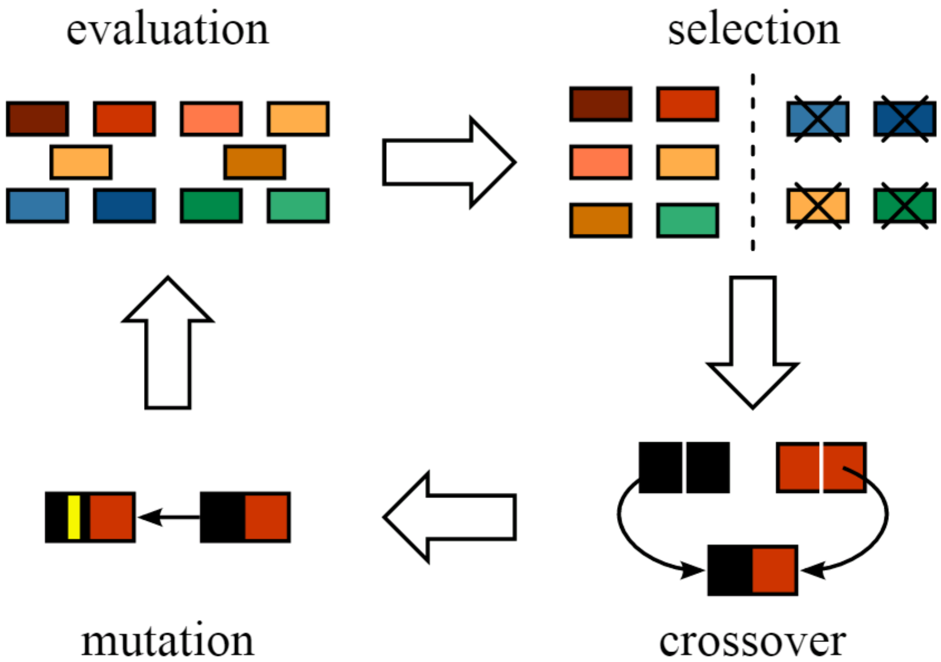
Sequential ATPG

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



Genetic Algorithms (GA) [Holland 1975]

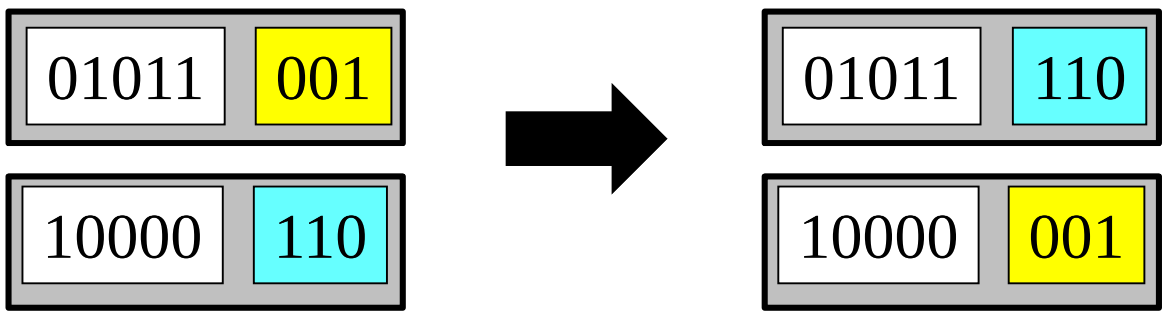
- **General Principle: Survival of fittest(s)**
  - **Keep a *population* of feasible solutions, not just one**
  - **Parent population generates child population**
    - **by gene *crossover, mutation* etc**
  - **Select only best children, remove weak children**
  - **Repeat the above for many generations**



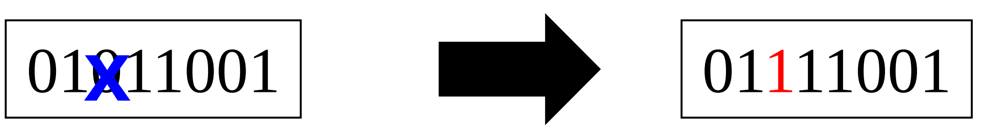
[www.jade-cheng.com]

Crossover and Mutation

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



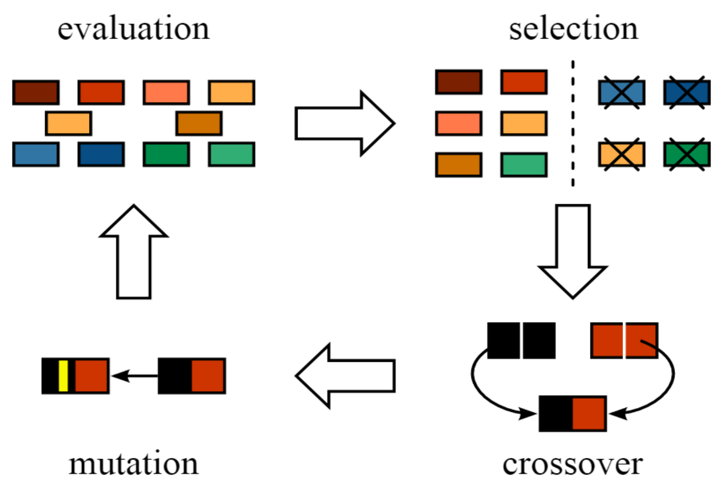
- ***Mutation*: some gene can change by a random probability**



Pseudo Code of GA

```
GENETICALGORITHM
pop = set of initial solutions
do
   childpop = ∅
   for (i = 1 to (n x pop.size))  // n times size
       crossover = random 0 or 1
       if (crossover)
          parent1 = random_choose(pop)
          parent2 = random_choose(pop)
          child = crossover(parent1, parent2)
       else   // mutate
          parent = random_choose(pop)
          child = mutate(parent)
       childpop = childpop ∪{child}
   pop = evaluate&select(childpop)
while (!stop)
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