VLSI Testing
第6章 时序电路的测试

西安电子科技大学微电子学院

6.1 时序电路测试概念

- 大多数电路为时序电路
- 时序电路的输出响应与当前输入和内部状态有关
- 测试时需要引入到已知状态
- 时序测试的困难
- 时序电路测试生成方法
  - 功能测试法、时序转换组合电路、可测性设计

6.2 时序电路的功能测试

- 时序电路功能测试应用在以下情况：不知道电路的逻辑图；知道状态转换图或流程表；电路太过复杂；故障模式不明
- 检测系统是否正常，可以通过核实它的序列偶**(I,O)**正常与否完成
- **T=(I,O)**称之为系统的功能核实序列，包括几种序列：
  - 同步**(Synchronizing)**序列**Xs**；
  - 引导**(Homing)**序列X_H；
  - 区分**(Distinguishing)**序列X_D等

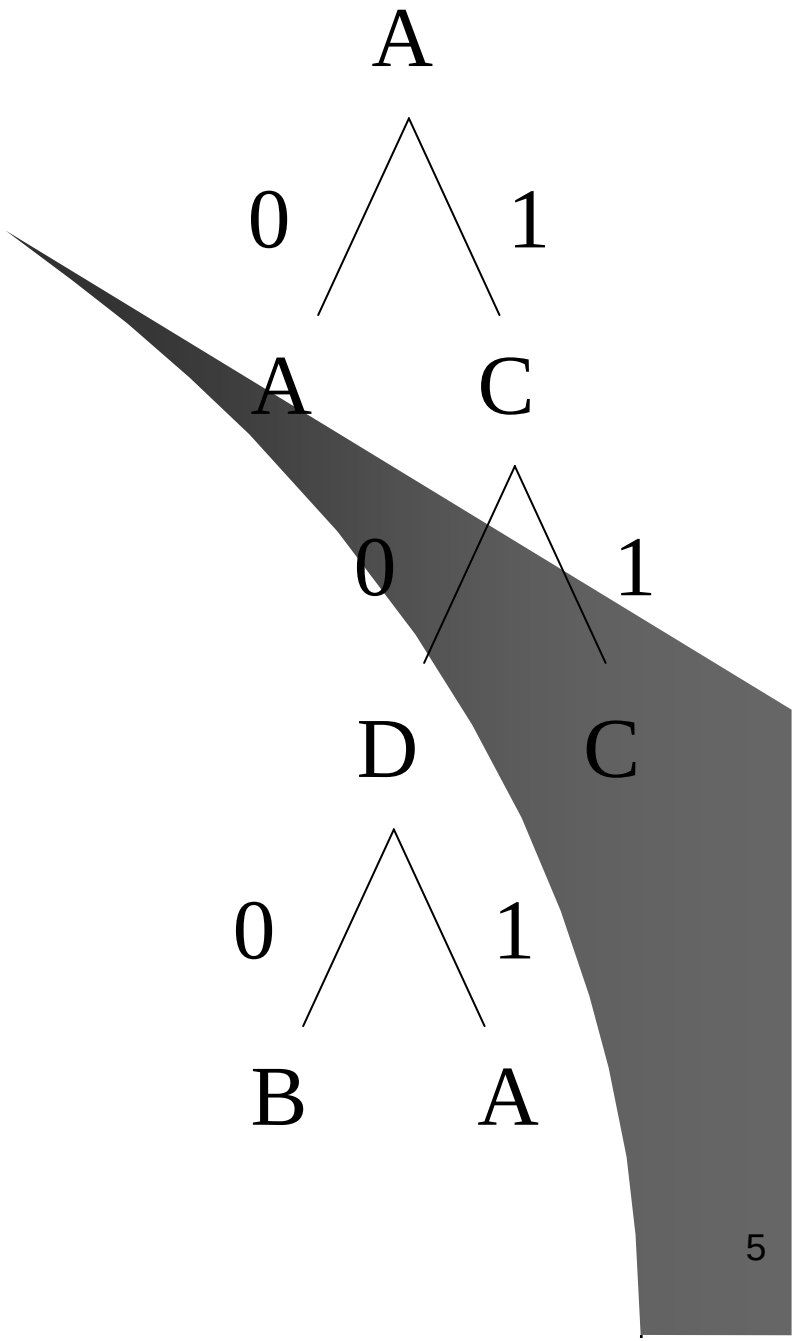
6.2.1 时序电路的检查序列

- 变迁（**transition**）序列：引起时序电路状态变化的输入序列
- 同步（**synchronizing**）序列：将系统从任意状态转移到*同一个*已知末态的序列，不一定都存在。
- 引导（**homing**）序列：使系统从未知状态转移到已知的末态的输入序列。总是存在。
- 区分（**distinguishing**）序列：能够根据不同响应序列区分系统的初态和末态的输入序列。不一定存在。

1. 变迁序列的求法

- 通过状态变迁树求得

	x	
	0	1
A	A/0	C/1
B	B/1	C/0
C	D/1	C/0
D	B/0	A/1

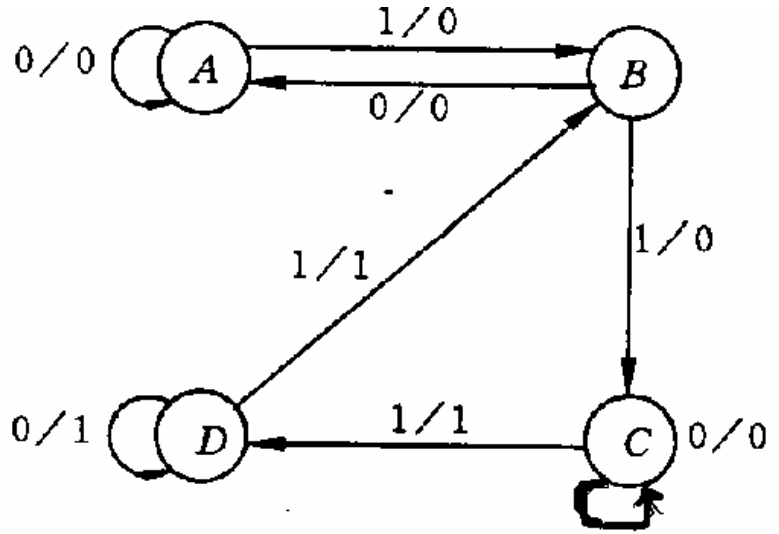


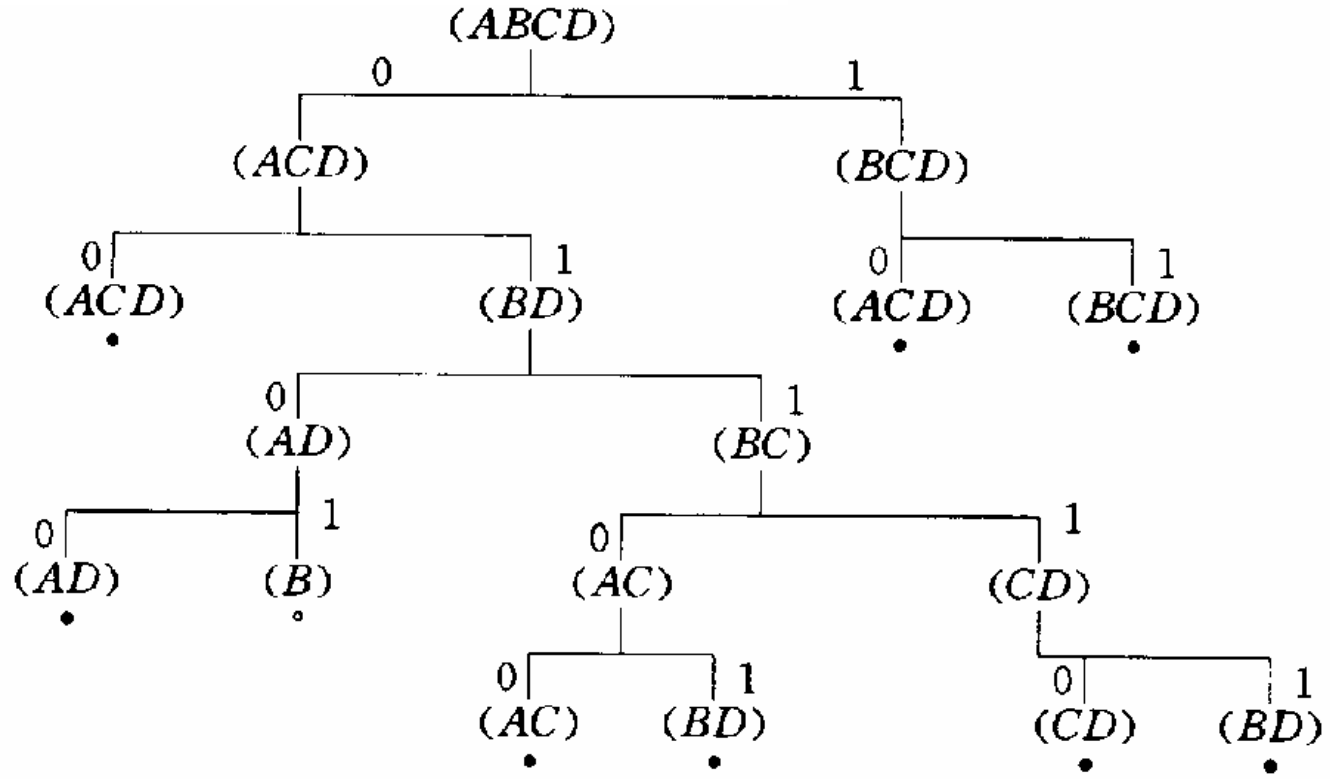
2. 同步序列求法

- 以系统的状态为树根，根据不同输入激励向下分支，得到响应状态的集合
  1）合并相同的状态
  2）新状态集合与以前的集合相同，则停止分支，对该状态标记“.”
  3）新状态集合只有一个元素，则停止分支，对该状态标记“。”
  4）其它情况继续分支
- 最后从根记到“。”的序列Xs

同步序列求法

- 同步序列是**0101**

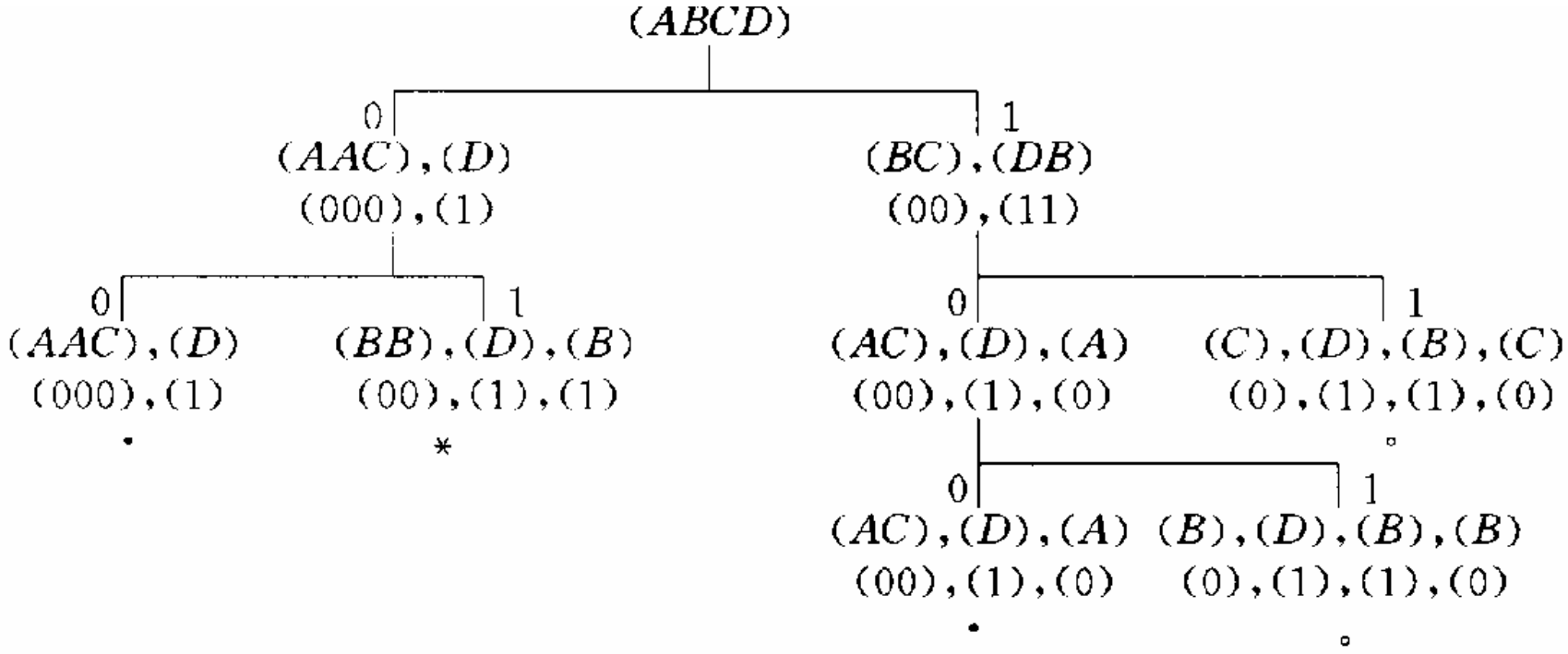




3. 引导序列的求法

- 以系统的状态为树根，根据不同输入激励向下分支，得到次态集合和输出
  1) 根据不同输入响应，将次态分割次态子集
  2) 若次态子集重复，停止分支记“.”
  3) 若每个次态元素都相同，停止分支记“*”
  4) 若每个次态子集都只有一个元素，停止分支，记“。”；其它情况继续分支
- 最后从根记到“。”或“*”的序列Xh

引导序列的求法举例



- 引导序列为**01**，输出**01**时末态**D,**其它**B**
- 引导序列为**11**，输出**00/10**时末态**C**，输出**01**时末态**D**，输出**11**时末态**B**
- 引导序列为**101**，输出**001**时末态**D,**其它**B**

4. 区分序列的求法

- 过程与求引导序列基本相同
- 每个区分序列必是引导序列
- 不同处在于：施加引导序列时，可以根据不同的响应序列来确定系统引导态；而区分序列不仅确定它的末态，而且可以确定不同的初态
- 区分序列是从树根到‘。’的输入序列
- 不一定每个电路都存在

区分序列举例

- 输入11

输出	初态	末态
00	A	C
01	B	D
10	D	C
11	C	B

- 输入101

输出	初态	末态
000	A	B
001	B	D
111	C	B
100	D	B

6.2.2 时序电路的功能测试

同步时序电路的功能测试

- 利用同步序列将系统从未知的初态同步到一个唯一的同步状态；如果没有同步序列，则先施加引导序列，再用变迁序列使系统进入唯一态
- 利用区分序列核实系统状态集中的各种状态；若无区分序列，用引导序列实施
- 核实状态转换功能的核实序列，用到区分序列以及变迁序列

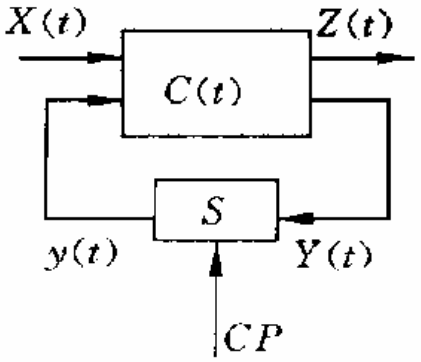
异步时序电路的功能测试

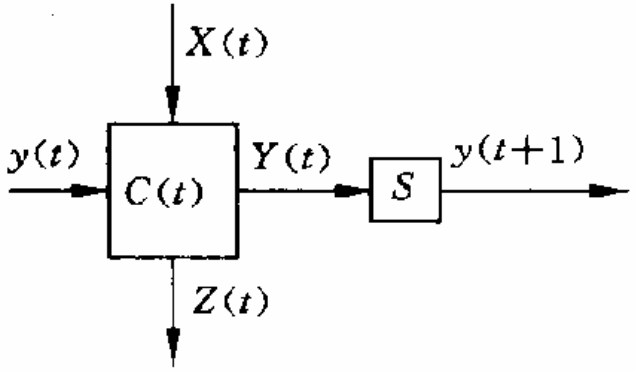
- 序列产生与同步电路相似
- 注意两点
  - 尽量避免竞争险态现象，通常使输入矢量每次只变化一个变量
  - 在任何测试序列的下激励都要避免电路产生振荡

6.3 时序电路的确定性测试生成

6.3.1时序电路组合化模型：

- 基本思路：将时序电路先转换为组合电路，再进行组合电路的测试生成
- 时间→空间分割 **Z(t)=Z(y(t),X(t))**
  Y(t)=Y(y(t),X(t))　y(t+1)=y(Y(t))





6.3.2 测试序列的生成

- 与组合电路同样需要故障激活和故障驱赶到输出端
- 由于有存储器件的存在，必须增加一组测试矢量，构成一个测试矢量序列
- 组合化时序电路测试可以利用D算法产生测试序列

1. 时序电路的初态

- 时序电路的初态往往是未知的
- 找一个测试序列，将电路引入确定状态，或者直接寻找一个与电路初态无关的故障测试序列
- 并非所有电路都适用，尽量用同步元件以及总复位线

2. 故障D立方的处理

- 时序元件的故障D立方从特征方程的差异产生，组合元件用组合电路的方法
- 可以从真值表，也可以由特征方程得出
- 以JK触发器为例

$y(t+1)=J(t)\overline{y(t)}+\overline{K(t)}y(t)$

y(t)	J(t)	K(t)	y(t+1)
0	0	×	$\overline{D}$
1	×	1	$\overline{D}$
0	1	×	D
1	×	0	D

3. 传播D立方的处理

- 时序元件的传播**D**立方

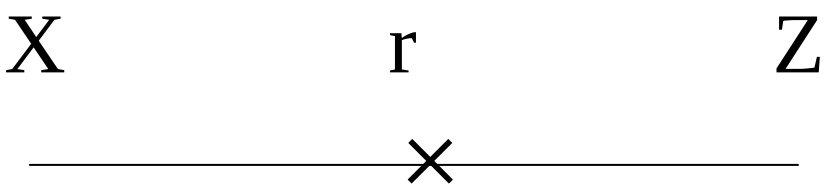
  通过特征方程推出，首先假设某个输入或输出的信号为**D($\overline{D}$),**再确定其它端条件，使输出端有故障出现

- 故障线的传播**D**立方

  在重复阵列中**D**驱赶与组合电路一样

  故障点的故障信号处理用传播**D**表计算

X	r	Z
0	0	0
1	0	D
D	0	D
$\overline{D}$	0	0
0	1	$\overline{D}$
1	1	1
D	1	1
$\overline{D}$	1	$\overline{D}$



4. 结束标志的处理

- 把故障信号D($\overline{D}$)驱赶到时段K的输出Z(K)=D($\overline{D}$),相应的输入序列X(0),X(1),...,X(K)即是所求的测试序列
- 如果有n个存储元件，每个的输出信号只有4种取值(0,1,D,$\overline{D}$)，读取了4^n个可能性仍不能产生测试向量，可以中止测试生成，该故障不可测

三 测试时序电路D算法的过程

1) 构造同步时序的时段化组合化模型
2) K=0，置初值y(0)
3) 仅对输出线Z(K)，执行D算法，若Z(K)=D($\overline{D}$),则找到测试序列；若无D($\overline{D}$)，继续扩展重复阵列做D驱赶
4) K=K+1，重复3)，若直到K＝4^n仍未找到测试序列，故障不可测
5) 结束

6.3.3 扩展的向后驱赶算法

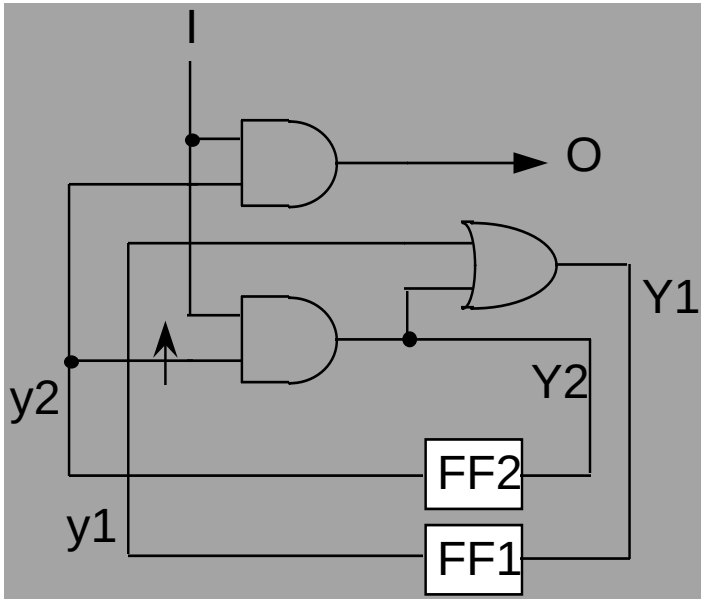
- Extended back-trace EBT
- 测试生成的时间反向处理法
- CV -V(n) & PV -V(n-1)
- 时间与空间的逆向处理

Extended D-Algorithm [1]

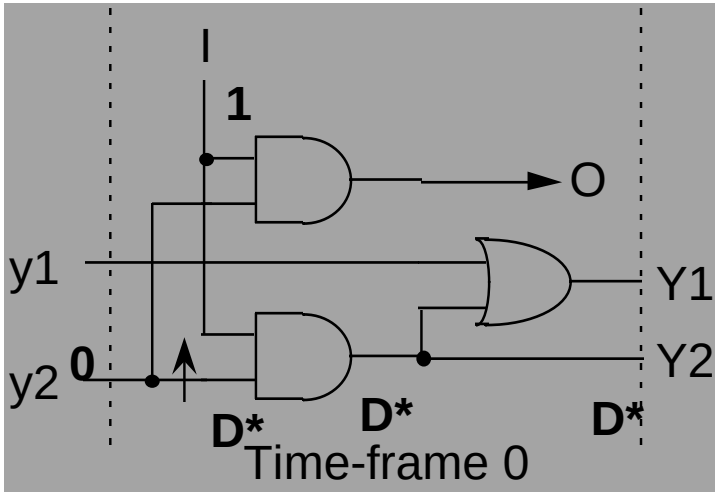
(Kubo, NEC Research & Development, Oct. 1968)
(Putzolu and Roth, IEEE TC, June 1971)

1. **Pick up a target fault f.**
2. **Create a copy of a combinational logic, set it time-frame 0.**
3. **Generate a test for f using D-algorithm for time-frame 0.**
4. **When the fault effect is propagate to the DFFs, continue fault propagation in the next time-frame.**
5. **When there are values required in the DFFs, continue the justification in the previous time-frame.**

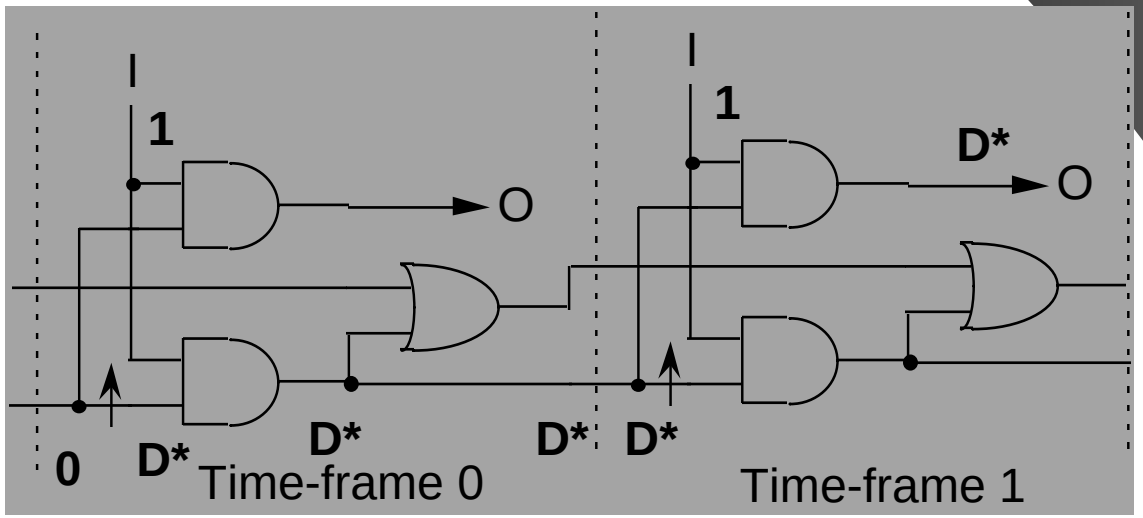
Example for Extended D-Algorith



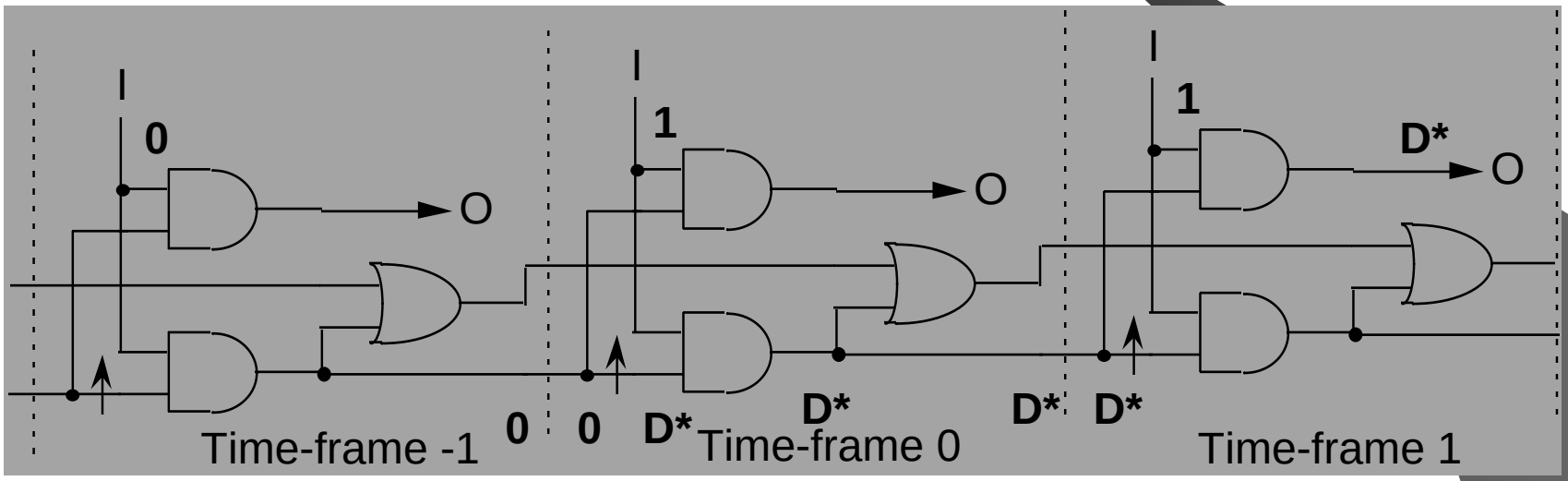
Example: Step 1



Example: Step 2



Example: Step 3



6.4 九值算法

- 本质上仍是**D**算法
- 主要解决异步时序电路的测试问题
- 做**D**驱赶时比一般**D**算法减少很多无用计算
- 充分考虑了故障在重复阵列模型中的重复影响作用，减少计算量
- 对一般**D**算法无法测试的故障有时可以产生测试矢量

一、九值模拟

1. 九值算法

- 3种逻辑值0、1、x的9种状态

  0=(0, 0) ， S_0=(0, 1) ， G_0=(0, x) ，

  1=(1, 1) ， S_1=(1, 0) ， G_1=(1, x) ，

  U=(x, x) ， F_0=(x, 0) ， F_1=(x, 1) ，

- 其中0, 1, S_0($\overline{D}$), S_1(D), U是5值D算法

- 基本运算

$$X_1 X_2 = (g_1, f_1)(g_2, f_2) = (g_1 g_2, f_1 f_2)$$

$$X_1 + X_2 = (g_1, f_1) + (g_2, f_2) = (g_1 + g_2, f_1 + f_2)$$

$$\overline{X} = \overline{(g, f)} = (\overline{g}, \overline{f})$$

2. 基本运算规则

- 交换律 $A+B=B+A$ $AB=BA$
- 分配率 $(A+B)C=AC+BC$
  $A+BC=(A+B)(A+C)$
- 反演律 $\overline{(A+B)}=\overline{A}+\overline{B}$
- 结合律 $(A+B)+C=A+(B+C)$
  $A(BC)=(AB)C$
- 公式$A(\overline{A})=0$和$A+(\overline{A})=1$不成立

3. 故障的激活立方

- 元件的原始立方集B=(Bf, Bg)，
  - Bf是有故障时的原始立方集，
  - Bg是故障求反的原始立方集，
  - B’f是用x代替故障端的立方集，
- 将B’f中输出值为1和0的子集分别与Bg的输出值为0和1的子集相交，然后合并成一个故障激活立方

4. 故障驱赶立方

- 若元件的输入端v有故障信号S0(S1)，原始立方集B=(Bf, Bg)
  - 当v=S0时，Bg={B中v＝0的子集}
  - 当v=S0时，Bf={B中v＝1的子集}
  - 当v=S1时，Bg={B中v＝1的子集}
  - 当v=S1时，Bf={B中v＝0的子集}

  然后把Bf和Bg中元素一一组合，构成元件E对输入端v的驱赶立方集

5. 反向计算

- 已知元件输出端为确定值，根据它确定输入端值的过程
- 计算方法：

  若输出为0(1)，则用C0(C1)；1(0)，则用C1(C0)

  若输出为S0(S1)，则用C0/C1(C1/C0)

  若输出为G0(G1)，则用Bg0/Bf1(Bg1/Bf0)

  若输出为F0(F1)，则用Bf1/Bg0(Bf0/Bg1)

  其中C0/C1表示元件输出为0/1的奇异立方集

6. 相容性检查

- 将状态信号做求交运算

$(g1, f1) \cap (g2, f2) = (g1 \cap g2, f1 \cap f2)$

- 主要是检查信号要求有无矛盾

7. 输入信号确认

- 为了确认输入信号，把G0,F0和S0用1代替，把G1,F1,S1用0代替，用x代替U，就可以得到集合(1,0,x)中的输入矢量

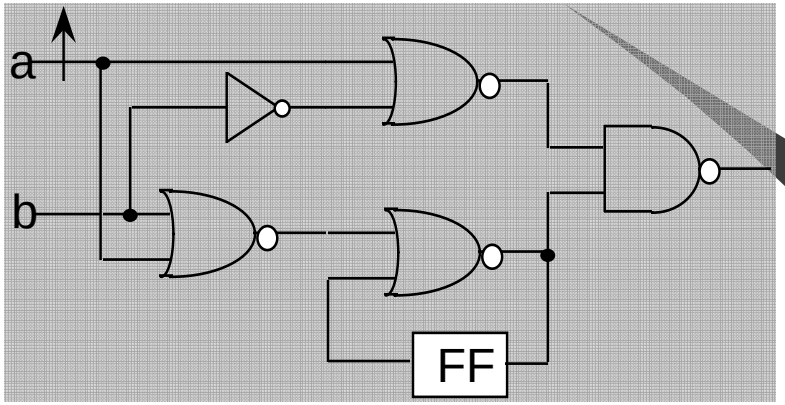
8. 九值算法流程

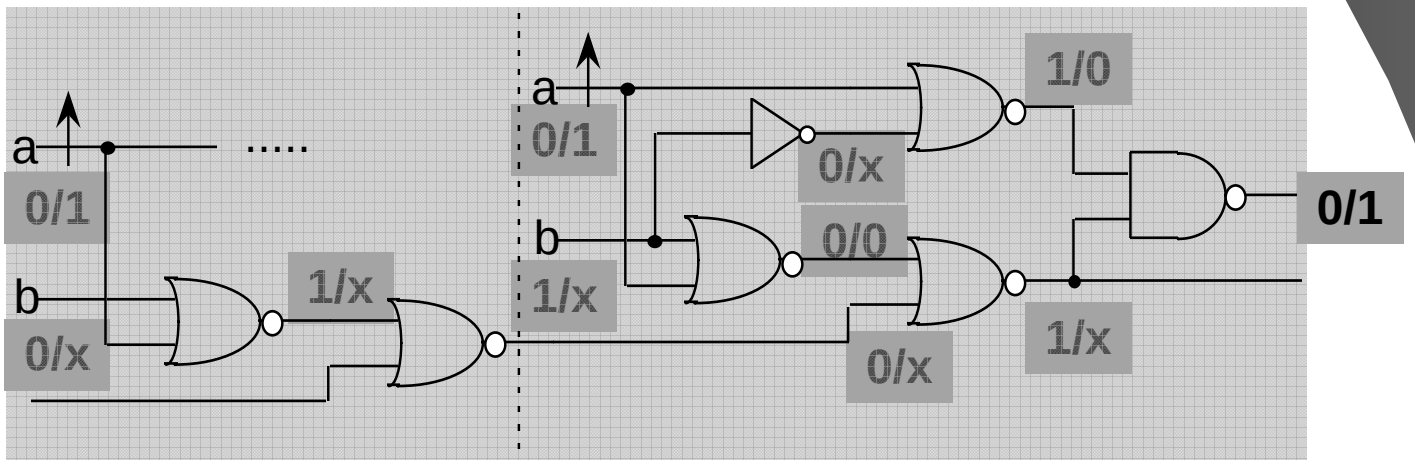
1. 异步时序电路的转换
2. 故障敏化
3. 故障驱赶
4. 反向计算
5. 在2－4各项中，都需做相容性检查，检查不通过及时返回

9. 九值算法改进

- 删去无效的故障敏化
- 减少反向计算过程中的反向寻踪的次数

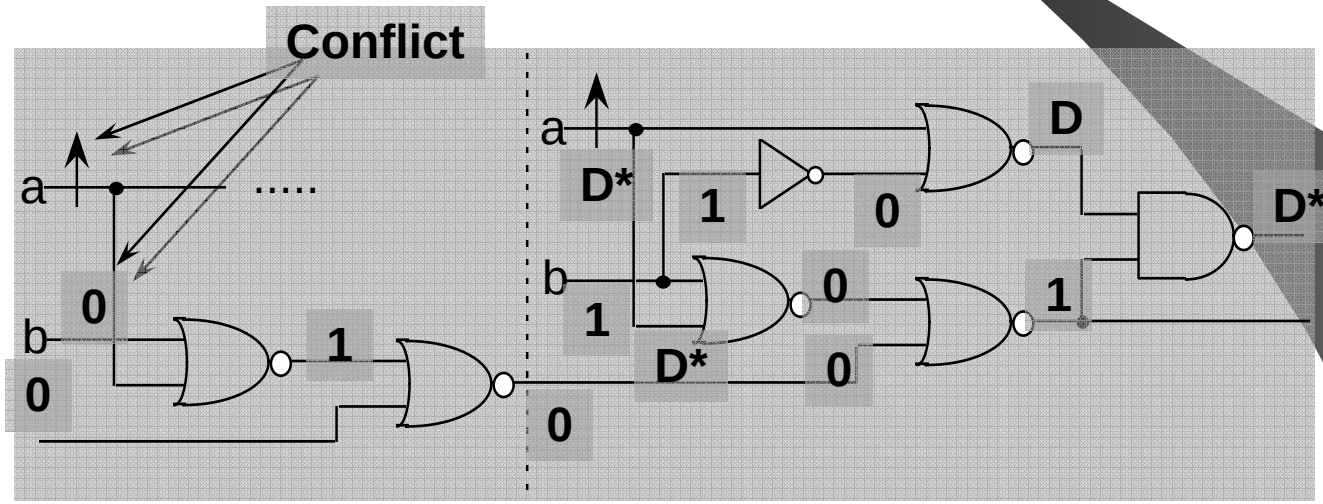
Example: Nine-Valued TG





9-V Algorithm

If Five-Valued TG Is Used



The test can not be generated by five-value TG.

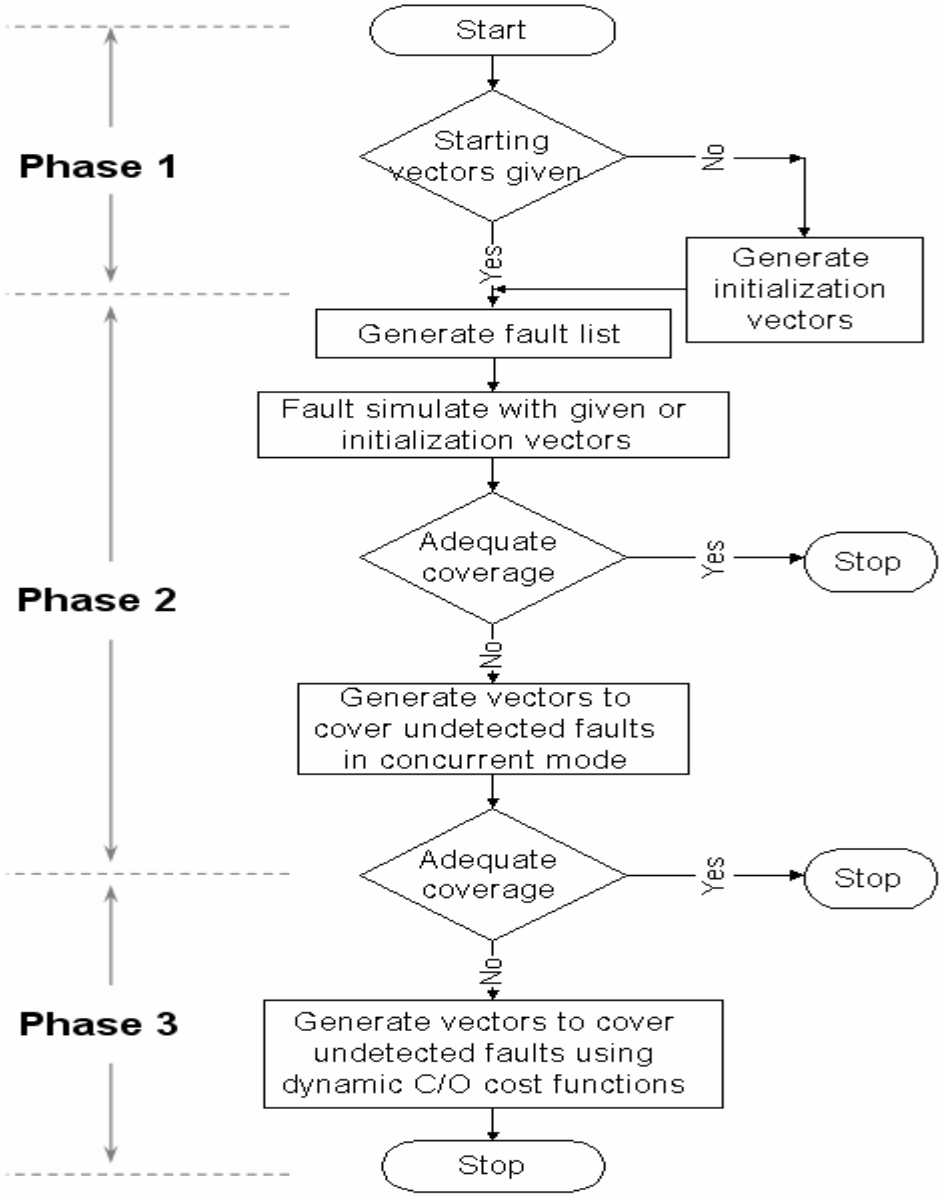
FASTEST算法

- 基于**PODEM**算法
- 九值电路模型
- 时间向前处理**TG**
- 状态初始化

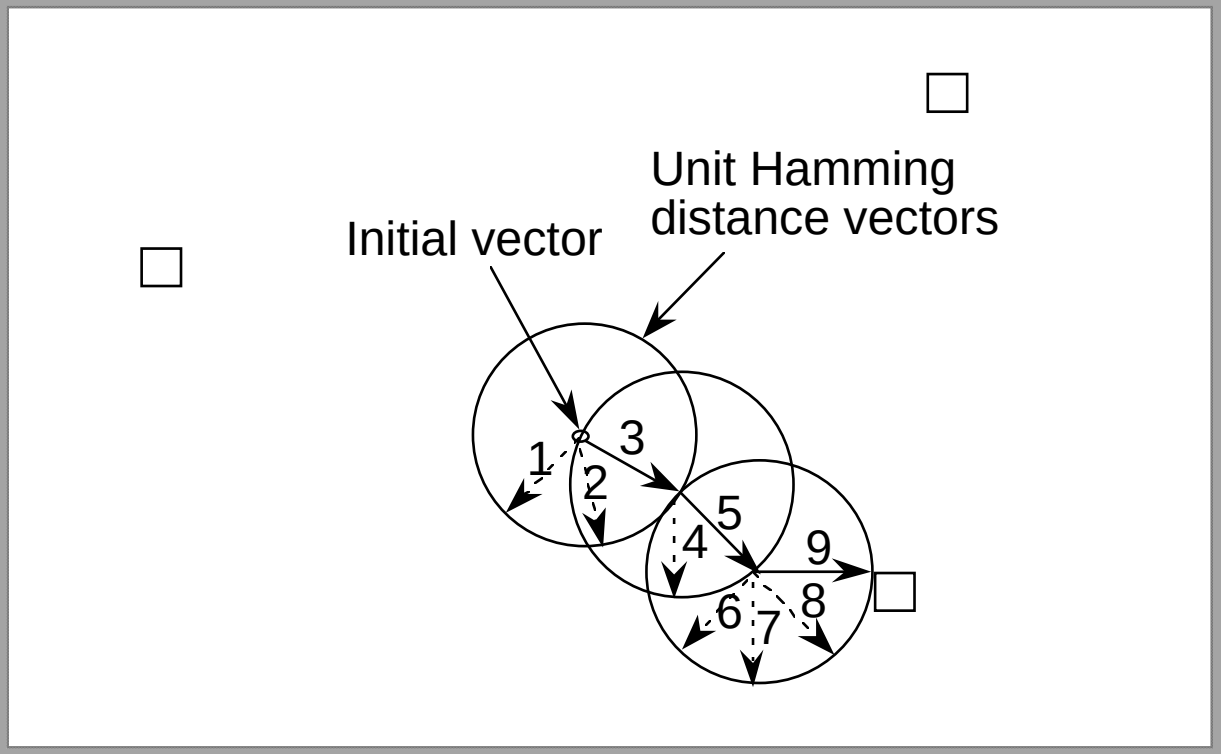
CONTEST算法

- 基于模拟的测试生成算法
- 思路
  - 选定矢量模拟结果并计算成本函数（如未知输出状态的个数）
- 对一组故障的测试生成
  - 成本函数（敏化路径上门的个数）
- 单故障的测试生成
  - 故障激活成本函数、故障效应成本函数

Flow Chart of CONTEST



Directed-Search in Input Vector Space



the possible tests for the given target fault

the trial vectors that are simulated but not accepted because of their higher costs

the accepted moves

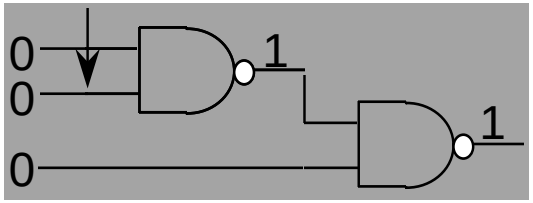
Phase 1 : Initialization

- **Start with arbitrary vector.**
- **Start with all flip-flops in unknown state.**
- **Use only true-value simulation**
  - **- Cost = number of flip-flops in unknown state.**
  - **- Generate new vectors to reduce cost by one-bit changes in the present vector.**
- **Stop when cost drops below desired value.**

Phase 2 : Concurrent Fault Detection

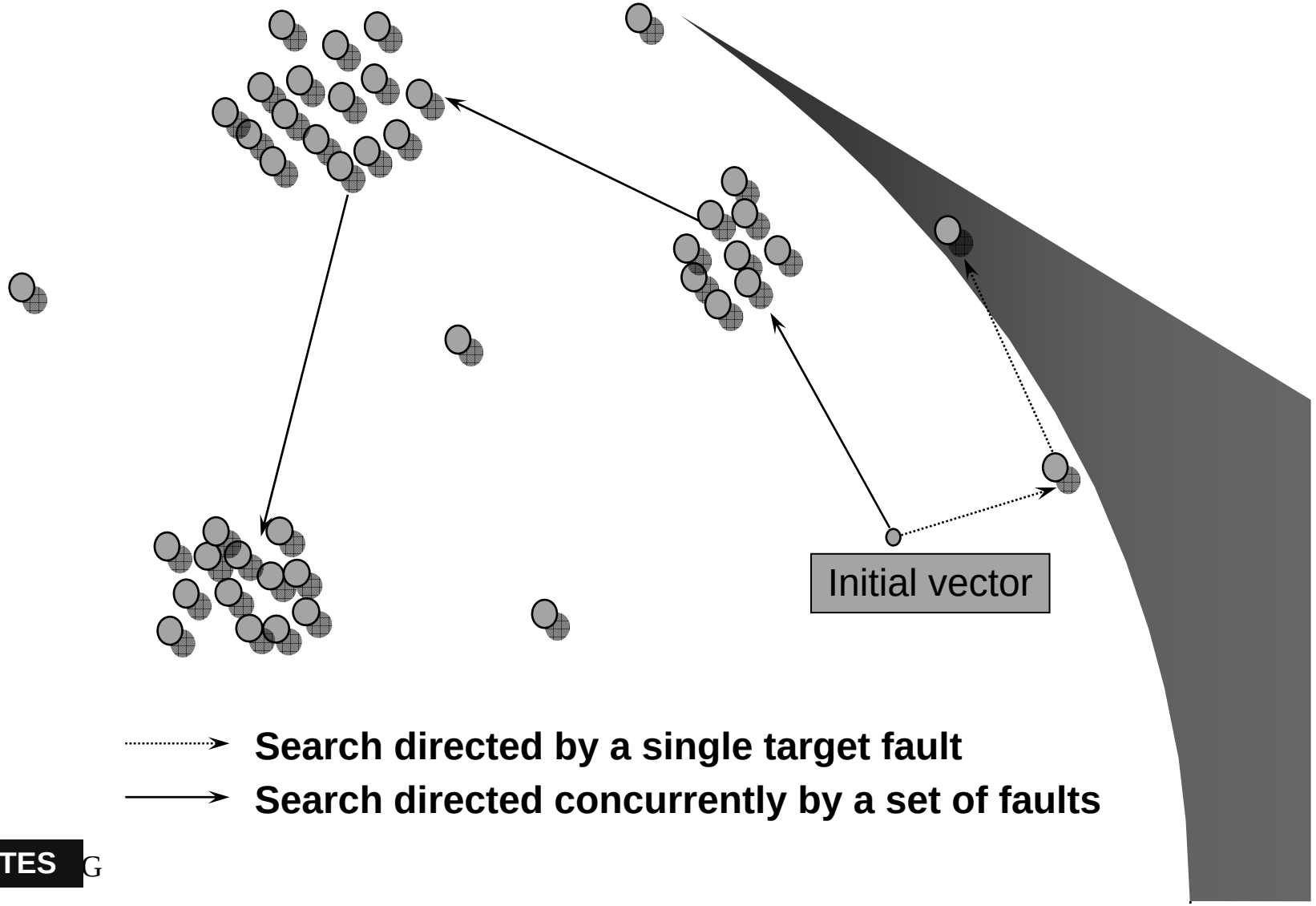
- **Start with initialization vectors.**
- **Fault simulate the vectors and remove detected faults.**
- **Compute cost function for the last vector.**
  - **Cost of an undetected fault = minimum distance of its effect from an PO.**
  - **Cost of the vector = Sum of costs of all undetected faults.**
- **Generate new vectors by accepting only those one-bit changes that reduce vector cost.**

Example : Distance Cost Functio



Trial vectors	**0**	**1**	**0**	**1**	**0**	**1**	**1**
	0	**0**	**0**	**1**	**1**	**0**	**1**
	0	**0**	**0**	**0**	**0**	**0**	**1**
Distance		**2**		**1**		**2**	**0**

Multi-Target Guidance



Phase 3 : Single Fault Detection

- **Start with the last vector obtained from Phase 2.**
- **Cost = k x Activation cost + Propagation cost.**
  - **Activation cost is the dynamic controllability of faulty line.**
  - **Propagation cost is the minimum (over all paths) dynamic observability of the faulty line.**
  - **K is a large weighting factor.**
- **Generate new vectors by accepting only those one-bit changes that reduce vector cost until the fault is detected or the search is abandoned due to a local cost minimum.**

Simulation-Based Approaches

Advantages:

- **Timing is considered.**
- **Asynchronous circuits can be handled.**
- **Can be easily implemented by modifying a fault simulator**

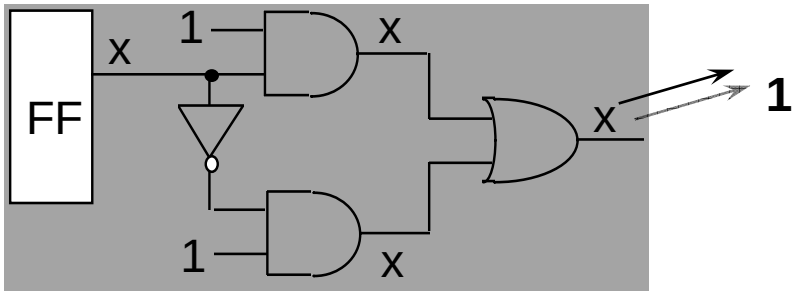
Disadvantages:

- **Can not identify undetectable faults.**
- **Hard-to-activate faults may not be detected.**

Difficulties of Sequential Test Generation

- **Initialization is difficult.**
  - **Justifying invalid states**
  - **Long initialization sequence**
  - **Simulator limitations**

ex:



- **Timing can not be considered by time-frame expansion.**
  - **Generated tests may cause races and hazards.**
  - **Asynchronous circuits can not be handled.**