ATPG（自动测试模式生成，Automatic Test Pattern Generation）算法是一种用于数字集成电路（IC）故障检测的方法。它可以自动生成用于检测电路中潜在缺陷的测试模式。ATPG算法的主要目标是通过最小化所需测试模式数量来实现高覆盖率和低测试成本。

ATPG算法的工作原理：

* 建立一个电路模型：使用标准硬件描述语言(HDL)或其他方式对设备进行建模。
* 定义故障列表：基于设备模型，列出可能导致电路失效的所有单点故障。常见的故障类型包括连接断开、短路和程控故障。
* 生成测试模式：使用算法来自动生成检测故障列表中每个故障的输入向量。这些输入向量将被应用到设备上，以观察其输出是否与预期一致。
* 评估和优化：分析生成的测试模式的覆盖率和效率，并根据需要优化算法以改进结果。

这种方法的主要优势是能提高故障检测的速度和效率，从而有助于生产高质量的产品并降低制造成本。ATPG算法已广泛应用于半导体行业，特别是在制造数字IC和微处理器等设备方面。

以下是一些常见的ATPG（自动测试模式生成，Automatic Test Pattern Generation）算法：

故障模拟法（Fault Simulation）：通过对部分输入向量进行假设，使用模拟器计算预期输出。同时，比较正常电路和故障电路的输出值以确定当前输入向量是否能够检测到故障。

D算法（D-algorithm）：利用错误传播机制来识别故障的深度优先搜索（DFS）策略。它通过控制和感知路径来判断是否可以区分正常和故障状态。

PODEM算法（Path-Oriented Decision Making）：这是一种启发式搜索算法，通过递归回溯和决策树生成输入向量。算法尝试通过调整输入值在控制和感知路径上传播故障。

FAN算法（Fanout-Oriented Cone）：FAN算法是对D算法的改进，将电路分解成多个锥形子结构以减少搜索空间。它能更快地找到区分正常和错误状态的测试向量。

SAT-Based ATPG：该方法将故障检测问题转换为可满足性问题（SAT），并利用现有的SAT求解器来识别故障。这种方法通常具有较高的内存效率和计算速度。

Genetic Algorithm-based ATPG：基于遗传算法的ATPG方法使用类似自然进化的策略找到最佳测试模式。这种方法可以有效地发现全局最优解并降低搜索空间。

动态编译技术（Dynamic Compilation Techniques）：将故障模拟器从软件转移到硬件（如FPGA），以加速模拟过程，进而提高ATPG的效率。

D算法是第一个完备的ATPG算法,也是当下主流的ATPG算法。它的主要思想是逐级敏化从故障源到电路所有输出的全部可能的通路。在D算法之后，又衍生出一些优化算法，如PODEM，FAN，CPT等等，不同算法的思想、方法各有不同，但目的都是相同的。作为ATPG算法的鼻祖，学习D算法的思想的流程对我们将来从事DFT行业是非常有益的。

1. 简介

D算法是一种有效的故障检测方法，用于发现集成电路和电子系统中的潜在问题。它采用布尔逻辑方程来表示目标系统，并根据生成的测试向量对其进行故障注入以及传播分析。通过对比正常和有故障的输出结果，可以确定故障位置和类型。

2. D算法原理与步骤

D算法的基本步骤如下：

初始化： 确定输入向量、故障发现概率阈值，将所有节点标记为未访问。

故障模拟： 模拟故障电路，计算假设故障所导致的输出差异。

故障传播： 从输出端开始，沿着故障可能传播的路径回溯，直至找到一个合适的测试点或达到预定的搜索深度。

D-前驱计算： 根据故障传播路径，计算D-前驱集，即故障可能影响的上游节点。

故障定位： 通过对比正常电路和含故障电路的输出，找到可能的故障位置。

生成测试向量： 基于D-前驱集，生成可区分故障与正常电路的输入测试向量。

迭代过程: 重复以上步骤，直至找到所有潜在的故障位置和类型。

3. 示例实现

给定一个简单的数字系统，其中Y = A ∧ (B ∨ C)，故障仅发生在与非门上。假设存在两种可能的故障：输出永远保持高电平（stuck-at-1，SA1）或低电平（stuck-at-0，SA0）。接下来使用D-算法进行分析。

SA1 故障分析:

初始化： 设定输入向量为(A, B, C)，所有节点标记为未访问。

故障模拟： 当故障发生时，Y始终保持为1。即使输入满足正常逻辑，Y也不会变成0。

故障传播： 要使该故障在输出端可观察到，我们需要找到一组输入，使得在正常情况下Y=0，而在故障情况下Y=1。

D-前驱计算： 为了使Y=0，我们可以将A设置为0，这样无论(B ∨ C)的值为多少，Y都会是0。在故障情况下，Y将保持为1。

故障定位： 对比正常和故障电路的输出，发现当A=0时，Y的状态不同，因此故障位于与非门上。

生成测试向量： 任何具有A=0的输入向量都可作为测试向量，例如(0, 0, 0)、(0, 1, 0)、(0, 0, 1)、(0, 1, 1)。

SA0 故障分析:

初始化： 设定输入向量为(A, B, C)，所有节点标记为未访问。

故障模拟： 当故障发生时，Y始终保持为0。即使输入满足正常逻辑，Y也不会变成1。

故障传播： 要使该故障在输出端可观察到，我们需要找到一组输入，使得在正常情况下Y=1，而在故障情况下Y=0。

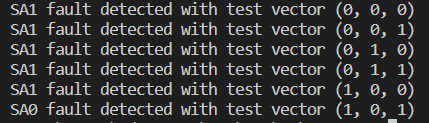
D-前驱计算： 为了使Y=1，我们需要A=1且(B ∨ C)=1。在故障情况下，Y将保持为0。

故障定位： 对比正常和故障电路的输出，发现当A=1且(B ∨ C)=1时，Y的状态不同，因此故障位于与非门上。

生成测试向量： 任何具有A=1且(B ∨ C)=1的输入向量都可作为测试向量，例如(1, 1, 0)、(1, 0, 1)、(1, 1, 1)。

示例代码，首先定义了正常数字电路和两种故障电路（SA1和SA0）。然后，函数generate\_test\_vectors()生成所有可能的输入向量。接下来，diagnostic\_algorithm()函数使用这些输入向量对正常和故障电路进行测试，以检测SA1和SA0故障。

最后得到的结果，即根据不同故障得到的测试向量如下图所示：



优缺点：

优点：

高故障覆盖率：D-算法能有效发现和定位数字电路中的故障。

适用范围广：适用于不同类型和规模的数字电路。

计算效率较高：通过聚焦故障传播路径，减少了搜索空间。

缺点：

对于某些复杂电路，D-算法可能面临搜索空间过大的问题。

某些情况下可能难以生成测试向量，需要额外的优化手段。

PODEM

PODEM（Path-Oriented Decision Making）算法是一种用于自动测试模式生成（ATPG, Automatic Test Pattern Generation）的故障检测方法。它主要应用于数字电路，以诊断单一固定故障（stuck-at faults）。与D-算法类似，PODEM算法也使用逻辑回溯和隐式测试分析来生成能区分正常和故障电路的输入测试向量。

主要步骤：

初始化： 确定目标故障、初始决策节点、所有节点的值。设所有节点未访问。

故障模拟： 模拟故障电路，计算故障所导致的输出差异。

隐式测试： 使用隐式测试策略选择一个观察点，尝试在该点上检测故障。

传播约束： 从观察点开始，沿着故障可能传播的路径回溯，找到一个合适的决策节点或达到预定的搜索深度。

决策： 如果当前决策满足故障检测条件，则移向下一个决策节点；否则回溯至上一个决策节点并尝试其他选项。

测试向量生成： 当所有决策均满足故障检测条件时，基于这些决策生成输入测试向量。

优缺点：

优点：

高故障覆盖率：PODEM算法能有效发现和定位数字电路中的故障。

良好的计算效率：聚焦于具有潜在故障影响的路径，减少搜索空间。

适用范围广：适用于不同类型和规模的数字电路。

缺点：

当多个故障同时出现时，算法可能需要额外的优化手段。

假设电路N中有节点g，其原始输入和输出分别为n个和m个，则有

要诊断故障，需要解方程

并且至少存在一个j，使得

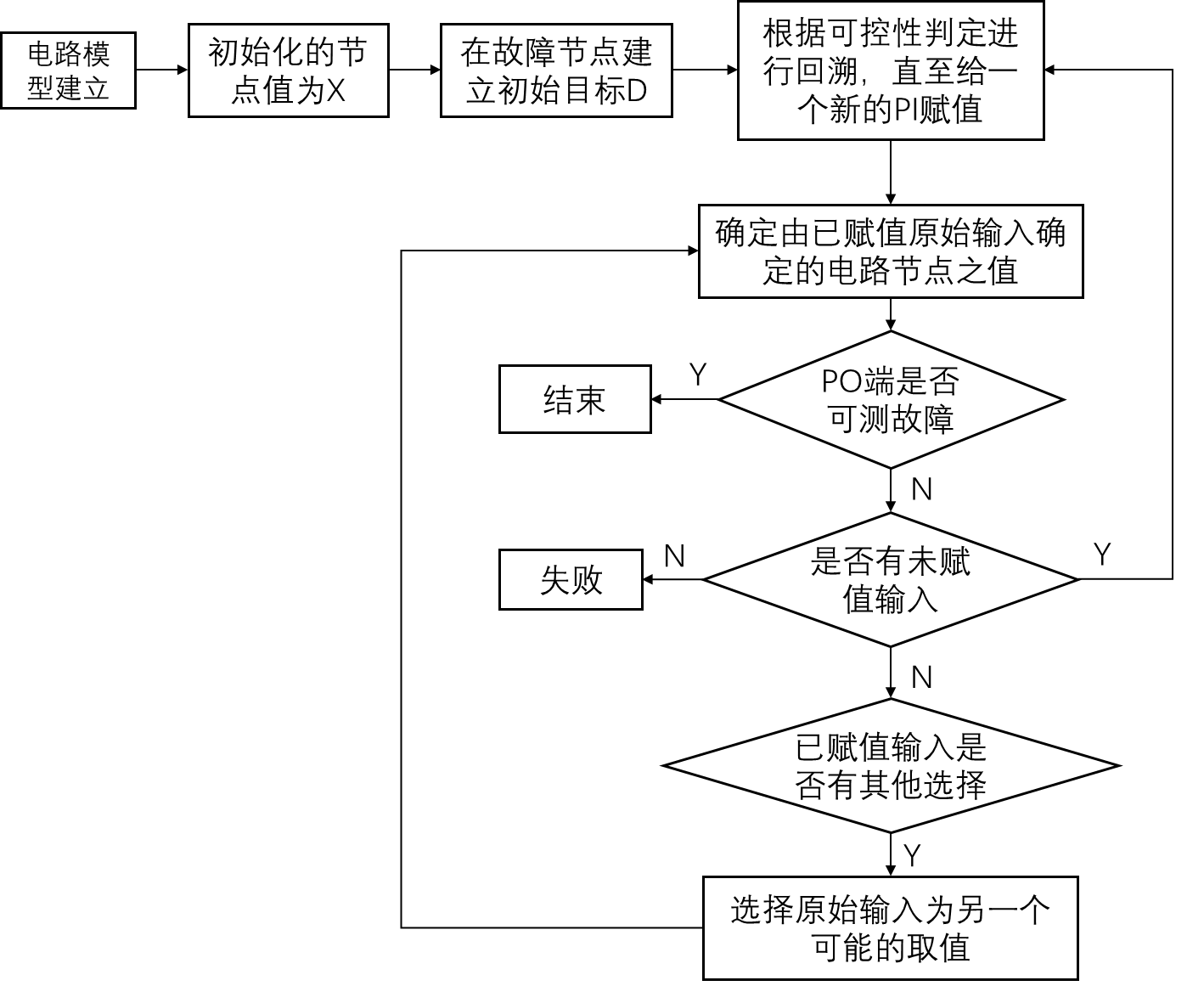
PODEM算法将解方程的n为空间搜索问题转化为分支判决树问题。

为便于程序的实现，节点用逻辑门序号标记。采用5值逻辑：。根据节点的可控性进行目标传递。在CAMELOT法中，可控性*CY*定义在0至1的范围内。可以置为“0”，“1”的*CY*具有最大值1，无法置为“0”，“1”的*CY*值最小为0。对于其他节点，采用下式计算：

*CTF*是针对某输出端的“器件可控性传递因子”。函数计算了对该输出端有控制作用的所有输入端的*CY*值。

*CTF*只取决于器件的逻辑功能，而与器件在电路中的位置无关。

*N(0)*和*N(1)*为器件输入端+所有不同的输入图形时在输出端出现“0”和“1”的总数。

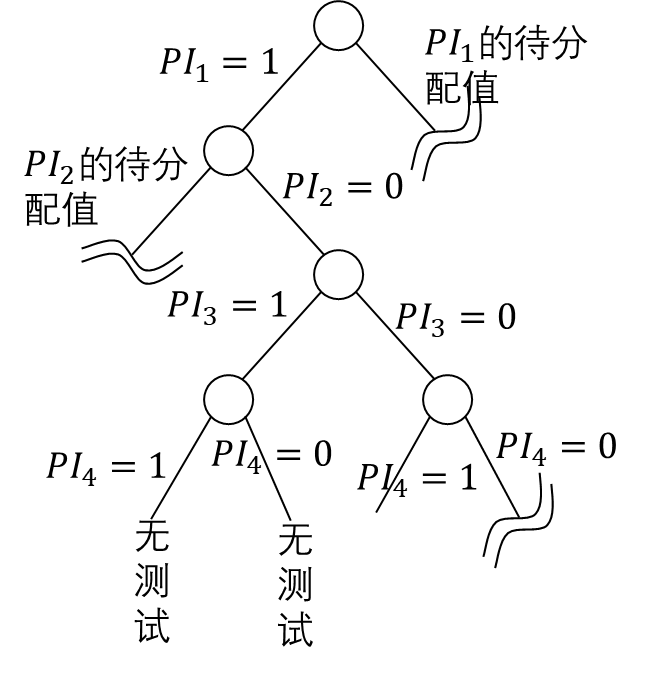


首先建立电路模型，初始化所有节点值为X，输入一个指定故障，对应初始化目标（）。根据最难/最易原则，把器件输出端的目标传递到每个未赋值的输入端进行回溯。若该输入为初级输入（PI），则赋予初级输入合适的值以满足目标。若该组新初级输入不满足目标或无测试则退出。当满足D目标、但未把D目标驱赶到初级输出（PO）时，选择一条从D目标到初级输出的最短通路，沿着该通路，进行目标前驱，重复此过程，直到把目标驱赶到输出或证明测试不存在为止。

下图所示就是判决书的流程。由于每个PI（原始输入端）只有两种可能的输入值，因此可以依次对每个输入进行两种试探，以致最后成下图所示的二分叉树。其中的所谓“无测试”是指下述两种情况：一是已确定的PI所蕴含的故障点的值同故障值相同，即故障没有得到激活；二是已确定的PI已经切断了故障点的故障信号想PO（原始输出端）传输的通路，即故障的通路没有得到敏化。在这两种情况中，在PO均不能观察到的故障信号，因此对应的矢量都不是故障的测试矢量。

在下图中，如果PIi的两种值均已做实验，其结果都是“无测试”，那么就把这个节点山区，同时把PIi-1的当前选择也删去，而且也不必再做PIi+1以后的各种选择。为了区别PIi的选择状态，我们给已做一种赋值试验而不成功，现在正在做新的尝试的PIi赋一个标志，以示区别。

显然为了加速搜索过程，各个PI变量的选择次序以及各个PI先赋什么值是一个关键问题。在PODEM算法中使用了初始目标的概念，而选择PI的策略就是要最大可能地达到这个目标。



示例分析：

在这个问题中，我们有一个电路包含一个AND门（类型：1）和一个OR门（类型：2）。输入线有2条（编号0和1），输出线有1条（编号3）。我们将使用PODEM算法（Path-Oriented Decision Making）来求解测试矢量。

首先，我们需要理解基本的PODEM算法步骤：

初始化电路

检查故障是否被检测到

通过后向传播选择目标，并为目标赋值

使用前向模拟进行模拟

如果成功找到测试矢量，则结束；否则回溯

现在，按照这些步骤对给定示例进行分析：

初始化电路

gates = [

{"type": 1, "input1": 0, "input2": 1, "output": 2},

{"type": 2, "input1": 0, "input2": 2, "output": 3}

]

输入线编号：0, 1

输出线编号：3

检查故障是否被检测到

由于我们没有明确指定要检测的故障类型，因此我们可以假设我们要检测 stuck-at-0 或 stuck-at-1 故障。在这种情况下，我们可以检查输出线3上的信号值。

通过后向传播选择目标，并为目标赋值

我们需要找到将错误效果传播到输出的路径。在这个例子中，我们可以通过控制输入线0和1来实现这一点。根据故障类型（stuck-at-0或stuck-at-1），分别为输入线0和1设置不同的值。

例如，如果我们检测 stuck-at-1 故障，我们可以尝试将输入线0设置为1，输入线1设置为0。

使用前向模拟进行模拟

有了输入值后，我们需要对电路进行模拟。以下是每个门的输出计算：

AND门：output 2 = input 0 AND input 1

OR门：output 3 = input 0 OR output 2

以我们设定的输入值为例：

输入0 = 1

输入1 = 0

模拟结果：

输出2 = 1 AND 0 = 0

输出3 = 1 OR 0 = 1

检查测试矢量是否成功

在这个例子中，我们设定的输入值确实产生了一个与设定的故障类型相反的输出（stuck-at-1）。因此，我们找到了一个有效的测试矢量：输入0 = 1, 输入1 = 0。

在PODEM算法中，对故障电路和无故障电路做逻辑分析计算的过程对应于D算法中的D驱赶，而D算法中的反向蕴含是包含在初始目标的确定和PI的确定过程中。二者不同之处是：在D算法中D驱赶的方向选择是随意的，而在PODEM算法中一定要寻找故障信号D到PO端路径最近的通路做驱赶；在做反向蕴含时，PODEM算法要选择“最容易”和“最困难”的路径，这样目标比较明确，成功的概率比较大，从而大大减少D算法中的返回次数，提高了运算速度。