第 3 章

测 试 生 成

本章首先介绍与测试生成有关的基本概念，包括测试的定义和相关术语，以及对于测试生成的基本要求。测试生成方法根据是否以故障为对象可分为非面向故障型和面向故障型两类，本章的侧重点在于说明面向故障型的测试生成，特别是基于通路敏化法的测试生成。本章通过固定型故障的测试生成来阐明通路敏化法的基本原理并介绍作为其代表性算法的PODEM。本章还会对减少测试数据量所需的测试精简技术进行简述，并介绍时延故障测试生成的基本原理。最后，本章介绍一个基于商用测试生成软件工具的测试生成流程的实例。

3.1 基本概念

芯片制造是一个非常复杂的过程，涉及大量的工艺、技术、设备、材料等。只要有一个环节出现哪怕是极其微小的异常，制造出来的芯片就会有缺陷，成为不良品芯片。芯片测试的根本目的就是把不良品芯片与良品芯片区分开来。如图 3-1 所示，为了进行芯片测试，首先需要进行测试生成(test generation)，其目的是产生测试激励(test stimulus)，即芯片测试所需的电路逻辑输入值，以及期待响应(expected response)，即正常电路应有的逻辑输出值。一个测试激励往往由逻辑 0 和逻辑 1 的组合来定义，故又常被称为测试码(test pattern)或测试向量(test vector)。测试一个电路(芯片)所需的全部的测试激励与期待响应的集合，就是测试数据(test data)。以考试来做比喻，测试激励相当于考试题目，期待响应相当于标准答案，一个被测电路(芯片)相当于一位考生。在实施芯片测试时，通过自动测试设备(Automatic Test Equipment，ATE)把测试激励逐个施加于芯片，并取得该芯片的实际响应(actual response)，然后把它与相应的期待响应相比较。若对于所有的测试激励，芯片的实际响应与其期待响应完全一致，则该芯片测试的结果为合格(被测芯片被认为是良品芯片，可以出厂)；若在某个测试激励下，芯片的实际响应与其期待响应不一致，则该芯片测试的结果为不合格(被测芯片被认为是不良品芯片，必须废弃)。

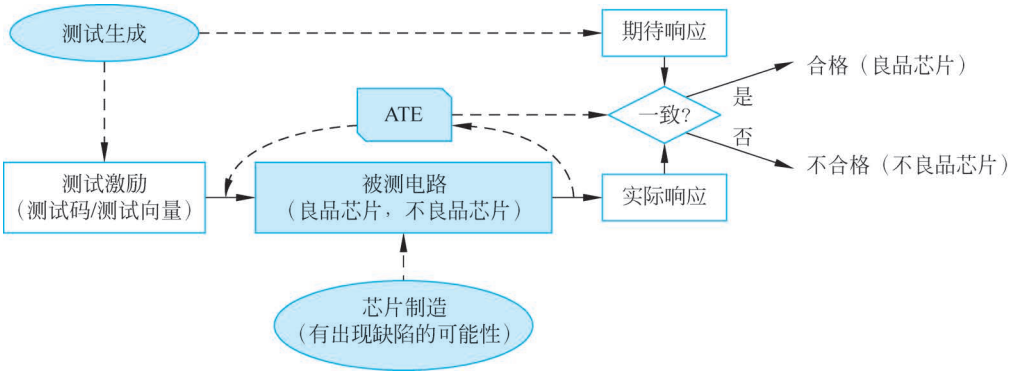


图 3-1 芯片测试的基本流程

测试生成所产生的测试数据对测试结果具有决定性的影响。如图 3-2 所示，若测试激励不充分，就有可能产生故障电路（不良品芯片）被误测为合格的现象（伪合格），造成测试不足（under-test）；若测试数据不正确，就有可能产生无故障电路（良品芯片）被误测为不合格的现象（伪不合格），造成过度测试（over testing）。这两种现象都会降低测试质量（test quality），从而有可能给芯片的使用方（因为伪合格）和生产方（因为伪不合格）造成重大的经济损失。因此，测试生成的第一要求是尽量提高测试质量，即尽可能降低不良品芯片被误测为合格（伪合格）的概率以及良品芯片被误测为不合格（伪不合格）的概率。另外，芯片测试具有对象数量庞大（有时每月需测试数十万到数百万的芯片），可用时间极短（一个数字芯片的测试时间往往不能超过几秒到几十秒），所需设备昂贵（一台高端 ATE 往往价值数百万人民币）的特点，很容易导致测试开销（test cost）过高。因此，测试生成的第二要求是尽量降低测试开销。

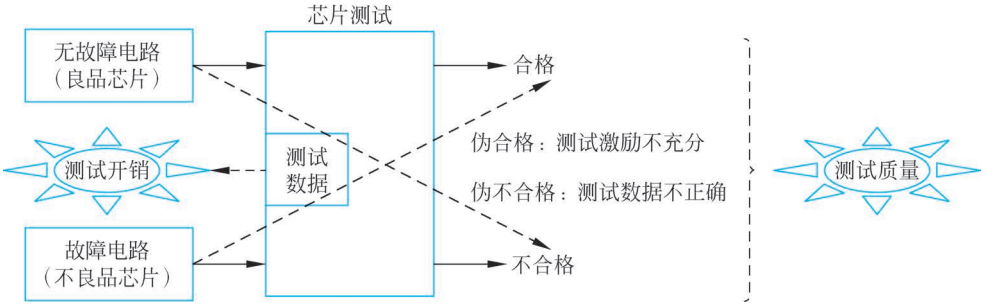


图 3-2 测试数据的重要性

多年以来，测试生成的第一要求（提高测试质量）主要是依靠尽量提高所生成的测试激励的芯片缺陷检测能力，以减少伪合格的概率来达到的。这涉及选择更好的测试生成策略，使用更好的故障模型（即用于反映芯片缺陷影响的各种假设），使用更好的测试生成算法等。近年来，通过减少伪不合格的概率来提高测试质量也受到了重视，其代表性的例子是低功耗测试生成，因为过高的测试功耗往往使良品芯片无法通过测试。另外，

测试生成的第二要求(降低测试开销)主要是靠尽量减少测试数据量来达到的。这涉及测试生成的过程当中利用各种测试精简技巧,以及在可测试性设计的过程当中利用各种测试压缩技术。

3.2　测试生成的分类

在作为测试对象的电路(芯片)的规模较小的年代,测试数据往往由电路设计者手工生成。但随着电路规模的不断增大,自动测试向量生成(Automatic Test Pattern Generation,ATPG)已占据主导地位,并由自动测试生成程序(Automatic Test Generation Program,ATGP)依据特定的测试生成方法来实现。测试生成方法可分为两大类,即非面向故障的测试生成和面向故障的测试生成。在电路规模不断增大的现代,面向故障的测试生成(又称为确定性测试生成或结构测试生成)已成为主流。

3.2.1　非面向故障的测试生成

非面向故障的测试生成的特点是它只需考虑被测电路的输入,而不用直接针对存在于其复杂的内部结构当中的物理缺陷或作为物理缺陷模型的逻辑故障,因此实现起来比较简便。但是,这种测试生成方法的缺点是为了达到一定的测试质量往往需要非常多的测试向量。此外,虽然非面向故障的测试生成不需考虑被测电路的内部结构及其物理缺陷或逻辑故障,但为了评估所生成测试向量的质量,往往还是需要通过故障模拟来计算故障覆盖率,这依然需要考虑被测电路的内部结构和逻辑故障。因为需要处理的测试向量非常多,这种故障模拟一般非常耗时。以下介绍三种典型的非面向故障的测试生成方法。

1. 功能测试生成

功能测试生成(functional test generation)可分为两种类型,即完全型功能测试生成和选择型功能测试生成。完全型功能测试生成是指产生一组测试向量来全面地验证被测电路是否能实现所有功能。比如,16 位加法器有 33 根输入线(包括 1 根进位输入线),17 根输出线(包括 1 根进位输出线)。要全面地验证这个加法器的所有功能,需要 2^{33} 个 50 位测试向量。尽管这些测试向量和期待响应非常容易生成,但因数量巨大,导致无法使用其全部来进行测试。在现实当中,人们往往使用选择型功能测试生成,即挑选一些用来验证被测电路关键功能的输入值作为功能测试向量,以便能在较短时间内完成测试。但是在这种情况下,需要通过故障模拟来计算故障覆盖率以评估测试质量。因此,当被测电路规模较大或功能测试向量较多时,故障模拟的时间开销往往非常大。

2. 穷举测试生成

穷举测试生成(exhaustive test generation)是指对于具有 n 根输入线的电路,产生 2^n 个测试向量。与完全型功能测试生成相似,当 $n>15$ 时,穷举测试生成的测试向量因所需测试时间太长而难以实际应用。为了解决这个问题,可以利用伪穷举测试生成(pseudo-exhaustive test generation),即在被测电路的输入线数 n 较大的情况下,把被测电路分割成

若干逻辑锥，以使每个逻辑锥的输入线数较小（比如不大于 15），再针对每个逻辑锥的输入组进行穷举测试生成。如图 3-3 所示，伪穷举测试生成的优点是，即使被测电路的输入线总数较大（本例当中为 26），也可使用较少的测试向量对各个输入组（本例当中为输入组＃1 和输入组＃2）对应的非重叠部分（本例当中的 A 部分和 B 部分）进行充分的测试。但是，伪穷举测试生成的缺点是不同输入组对应的重叠部分（本例当中的 C 部分）中的某些故障的检测可能需要从多个输入组同时输入某个特定的逻辑值组合，从而导致其不能得到充分的测试。

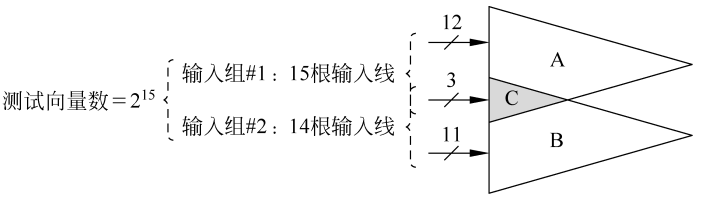


图 3-3 伪穷举测试生成

3. 随机测试生成

随机测试生成（random test generation）是指从所有可能的输入向量当中随机地选择一部分作为测试向量。若选择的随机测试向量的数目不大，则可实际应用于大规模电路的测试。但由于随机测试向量的不完全性，需要进行故障模拟来计算故障覆盖率，以便对测试质量进行评估。一般来说，随机测试向量可以比较容易地达到一定的故障覆盖率（比如 70％～80％）。但是，电路当中往往会存在一些很难用随机测试向量检测的故障。比如，被测电路内部的一个 5 输入 AND 门的输出线上的固定为 0 故障（s-a-0）的检测至少需要该门的所有输入值都为逻辑值 1，这个要求很难通过对被测电路的输入线施加随机测试向量来达到。这样的故障被称为随机测试向量抵抗性故障（random pattern resistant fault）。这个问题可以通过分析被测电路的结构并改变其输入线的逻辑值 1（或 0）的出现概率得到一定的缓解，但其最终解决还是需要利用面向故障的测试生成。

3.2.2 面向故障的测试生成

非面向故障的测试生成由于不考虑被测电路的内部结构以及存在于其中的物理缺陷或逻辑故障，因此实现起来非常简单，测试生成时间也很短。但是，非面向故障的测试生成的最大的缺点是为了达到一定的测试质量往往需要非常多的测试向量，从而造成测试时间及测试开销的剧增。为了解决这个问题，面向故障的测试生成，也称为确定性测试生成（deterministic test generation），采用了根据被测电路的内部结构而直接针对其中的逻辑故障（即用来反映物理缺陷的影响的某种模型，往往简称为故障）来产生测试向量的思路。一般情况下，检测一个故障需要一个测试向量。由于被测电路内部可能存在的故障的总数远远少于其输入线的全部逻辑值组合数，因此面向故障的测试生成的测试向量数通常远远少于非面向故障的测试生成的测试向量数。面向故障的测试生成因为需要考虑被测电路的内部结构，故又常被称为结构测试生成（structural test generation）。

目前，结构测试生成是测试生成的主要实际应用方式。如图 3-4 所示，16 位加法器有 33 根输入线（包括 1 根进位输入线 C_{in}），要完全验证这个加法器的所有功能，功能测试生成需要产生 2^{33} 个测试向量。从结构上看，16 位加法器由 16 个全加器组成。在门级实现当中，全加器有 16 个位置有可能发生单固定型故障，可能是固定为 0 故障（s-a-0）也可能是固定为 1 故障（s-a-1），所以全加器共有 32 个可能发生的单固定型故障。这样，由 16 个全加器所构成的 16 位加法器就有 512 个可能发生的单固定型故障。因此，结构测试生成最多只需产生 512 个测试向量（由于 1 个测试向量通常可以检测多个故障，实际所需的测试向量的数目会少于 512），远远少于功能测试向量的数目（2^{33}），非常适合应用于实际的芯片测试。

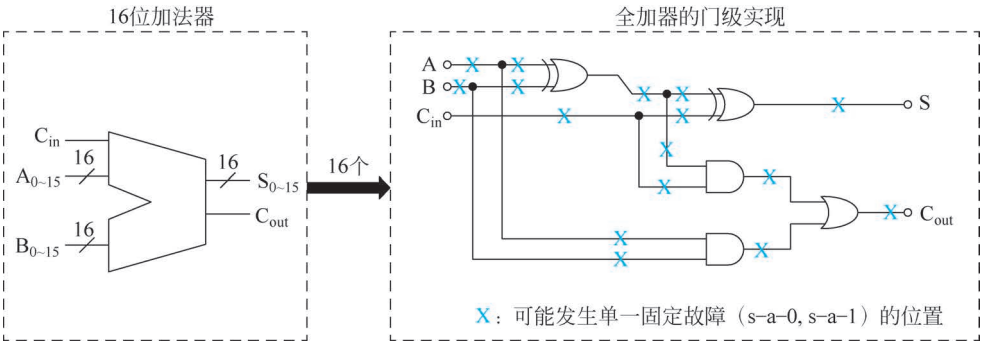


图 3-4　面向故障的测试生成（结构测试生成）

结构测试生成具有两个明显的特点。第一，结构测试生成需要用到被测电路的结构信息，可以是门级的，也可以是晶体管级的，还可以是 RTL 级的。在实际当中，使用门级电路结构信息的结构测试生成最为常见。第二，结构测试生成需要选定作为测试生成对象的故障模型，如固定型故障、桥接故障、时延故障等。要达到较高的测试质量，往往需要针对包括固定型故障模型在内的多个故障模型进行结构测试生成。在实际的结构测试生成中，往往首先生成针对固定型故障的测试向量，并利用故障模拟计算故障覆盖率来评估测试质量。若需要进一步提高测试质量，则可追加使用其他的故障模型（比如时延故障）进行结构测试生成，产生更多的测试向量。

另外，作为测试对象的芯片的内部电路往往是时序电路，其特征是具有内部状态，致使其输出值不仅取决于现在的输入值，还取决于过去的输入值，从而导致即使规模很小的时序电路的测试生成时间也非常长。因此，在实际当中，往往需要对时序电路进行可测试性设计（第 4 章介绍），建立完整的扫描链。这样做的好处是可以把时序电路的测试生成问题转换成为组合电路（其输出值完全取决于现在的输入值）的测试生成问题。尽管组合电路的测试生成问题依然是 NP 完全问题，但是通常都可以在比较现实的时间内产生所需的测试向量。下面介绍的通路敏化法和 PODEM（Path Oriented Decision Making）算法，就是用来解决组合电路的测试生成问题的常用策略和代表性算法。

3.3 通路敏化法

结构测试生成可以利用包括布尔差分在内的多种方式来实现，但其最主要的实现方式是通路敏化(path sensitization)法。这种方法原理简单，适用于大规模电路，因此在实际当中得到了广泛的应用。下面以门级逻辑电路的固定型故障的测试生成为例，来介绍通路敏化法的基本原理及作为其代表性算法的 PODEM。

3.3.1 基本原理

通路敏化法的基本原理可用如图 3-5 所示的门级电路作为例子来说明。这个电路具有 6 个门(G_1～G_6)，4 根外部输入线(a，b，c，d)和 2 根外部输出线(x 和 y)，另外还有 10 根内部信号线(L_1～L_{10})，测试生成的对象故障为内部信号线 L_7 上的固定为 0 故障(s-a-0)。很明显，若该对象故障存在，则不论外部输出值如何，L_7 的值都将固定为 0，即故障值，在图 3-5 中用“0”来表示。

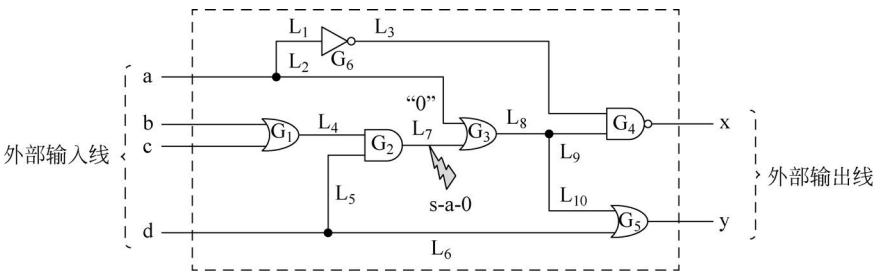


图 3-5 被测电路与对象故障

通路敏化法通过以下 3 个主要步骤来实现。

步骤 1：故障激活。故障激活(fault sensitization)是指通过把与固定型故障的故障值相反的逻辑值赋予故障线，以暴露该故障的存在。图 3-5 所示电路的故障线是 L_7，故障值是 0，因此激活这个故障需要把 1 赋予 L_7(表示为$<L_7,1>$)。如图 3-6 所示，这个赋值的结果是使故障效果(fault effect)出现在 L_7 上。也就是说，若 s-a-0 故障存在于 L_7，则故障值 0 会出现在 L_7 上；若 s-a-0 故障不存在于 L_7，则正常值 1 会出现在 L_7 上。如图 3-6 所示，这个故障效果常用记号 D(正常值为 1/故障值为 0)来表示。若故障效果是正常值为 0 而故障值为 1，则可用记号 $\bar{D}$(正常值为 0/故障值为 1)来表示。此外，至少有一个输入值为 D 或 $\bar{D}$ 而输出值尚未确定的所有逻辑门组成集合称为 D 前沿(D-frontier)。在图 3-6 所示的例子当中，D 前沿为$\{G_3\}$。

步骤 2：故障传播。故障传播(fault propagation)是指通过对包括内部信号线在内的一些信号线的赋值而使故障效果沿着一条或多条通路向外部输出线传播，使其最终到达至少一根外部输出线，从而达到检测该故障的目的。如图 3-7(a)所示，由于此时的 D 前沿为$\{G_3\}$，可以通过把 0 赋予 G_3 的输入线 L_2(表示为$<L_2,0>$)以使 L_7 上的故障效果 D 传播到

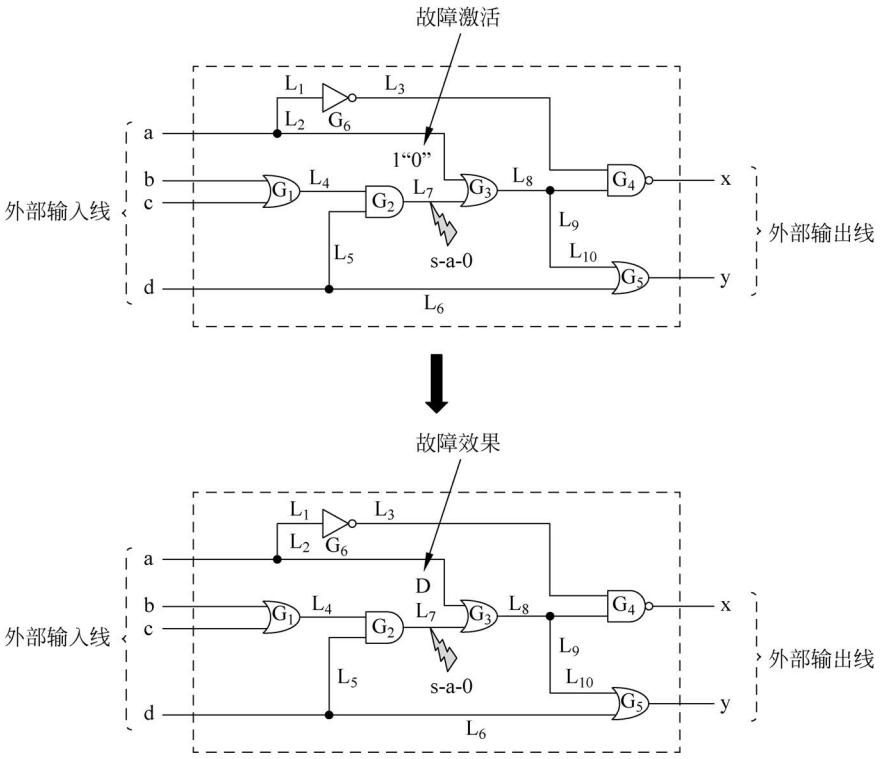
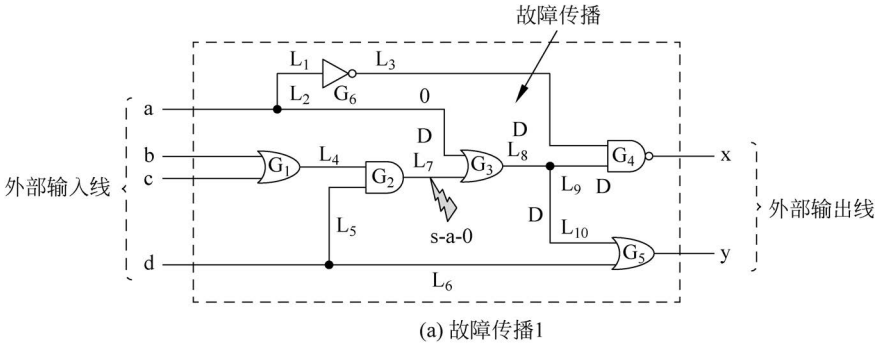


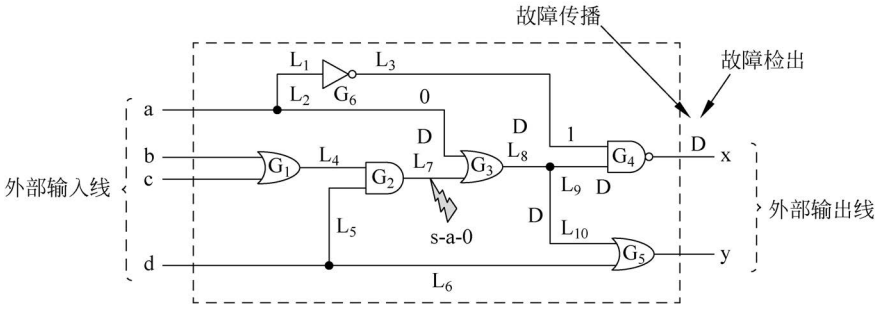
图 3-6　故障激活

L_8、L_9、L_{10} 上，这导致 D 前沿由{G_3}变为{G_4，G_5}。如图 3-7(b)所示，可以通过把 1 赋予 G_4 的输入线 L_3(表示为<L_3，1>)使得 L_9 上的故障效果 D 进一步传播到外部输出线 x 上，从而使 L_7 的 s-a-0 故障得到检测。在图 3-7 所示的例子当中，故障传播(共计两次)所需的赋值为<L_2，0>和<L_3，1>。

步骤 3：线确认。线确认(line justification)是指通过确定一些外部输入线的逻辑值而使故障激活和故障传播所需的所有信号线赋值得以实现。若线确认成功，则对外部输入线所赋的逻辑值组合就是可检测对象故障的测试向量。在图 3-8 所示的例子当中，故障激活所需的赋值为<L_7，1>，故障传播所需的赋值为<L_2，0>和<L_3，1>。如图 3-8 所示，这些赋值要求可以成功地通过线确认来实现，其结果是<a＝0，b＝1，c＝X，d＝1>，这就是 L_7 的 s-a-0 故障的测试向量。在这里，X 表示其逻辑值尚未确定，即可为 0 亦可为 1，不会影响测试结果。这样的含有未确定值 X 的外部输入线逻辑值组合也称为测试立方(test cube)。应该注意的是，测试立方是测试生成过程中的中间产物，其中的 X 最终必须以某种方式设置为 0 或 1 之后才能用于实际测试，这个操作叫作 X 填充(X-filling)。为了区别起见，测试向量往往指不含有未确定值 X 的外部输入线逻辑值组合。测试立方和测试向量的这种区别如图 3-8 所示。



(a) 故障传播1



(b) 故障传播2

图 3-7 故障传播

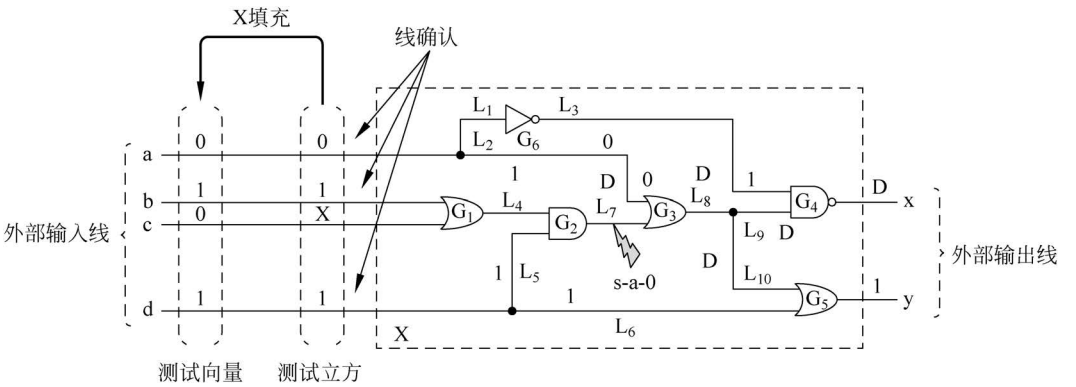


图 3-8 线确认

3.3.2 PODEM

通路敏化法催生了众多的测试生成算法。下面要介绍的 PODEM 算法以简洁明了著称，它的许多基本技法被广泛应用于其他的基于通路敏化法的测试生成算法当中。图 3-9 为 PODEM 的概念性流程，其中的测试向量是广义的，既可指不含有未确定值 X 的输入线逻辑值组合（即狭义的测试向量），也可指含有未确定值 X 的输入线逻辑值组合（即测试立方）。

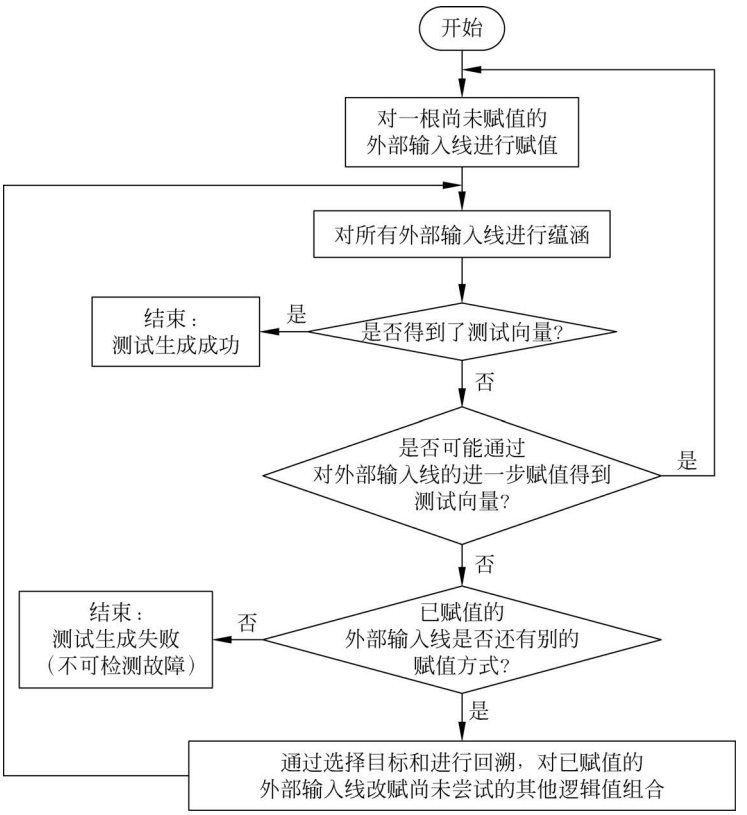


图 3-9　PODEM 的概念性流程

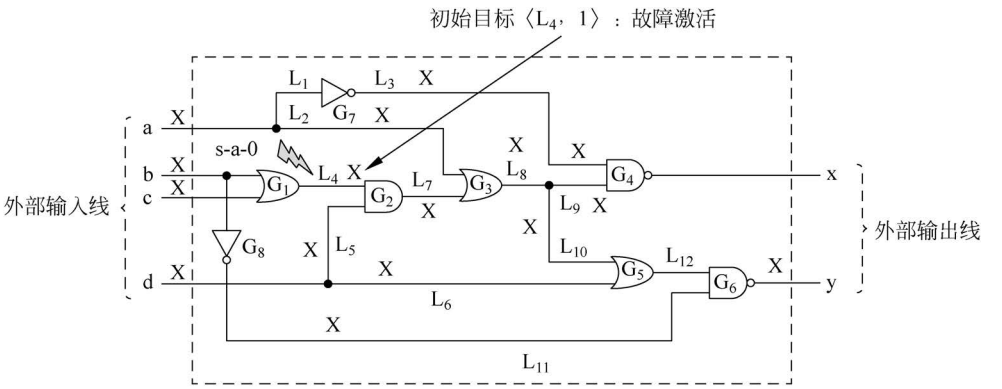
PODEM 的特点是把测试向量生成作为一个探索问题来解决，其探索空间由全部的外部输入线构成。假设被测电路共有 n 根外部输入线，因每根输入线都有 0 和 1 两种选择，因此探索空间的大小为 2^n。针对一个故障的测试生成，就是试图在该探索空间中找到一个可以检测该故障的测试向量（即对被测电路外部输入线的一组赋值）的过程。PODEM 利用下面介绍的几个技法使这个探索尽可能快速有效地完成。

如图 3-9 所示，PODEM 每次选一根外部输入线进行赋值，并通过蕴涵（implication）操作来确定其他一些信号线的逻辑值。随后，PODEM 确认一个测试向量是否已被生成。倘若测试向量尚未被生成但是继续探索有望成功，PODEM 会选择下一根外部输入线并重复以上的操作。若继续探索不可能成功，PODEM 会进行回溯（backtrack）操作，以尝试外部输入线的其他的赋值方式是否可使测试生成成功。以下列出 PODEM 需要面对的 5 个关键问题并介绍其解决思路和技巧。

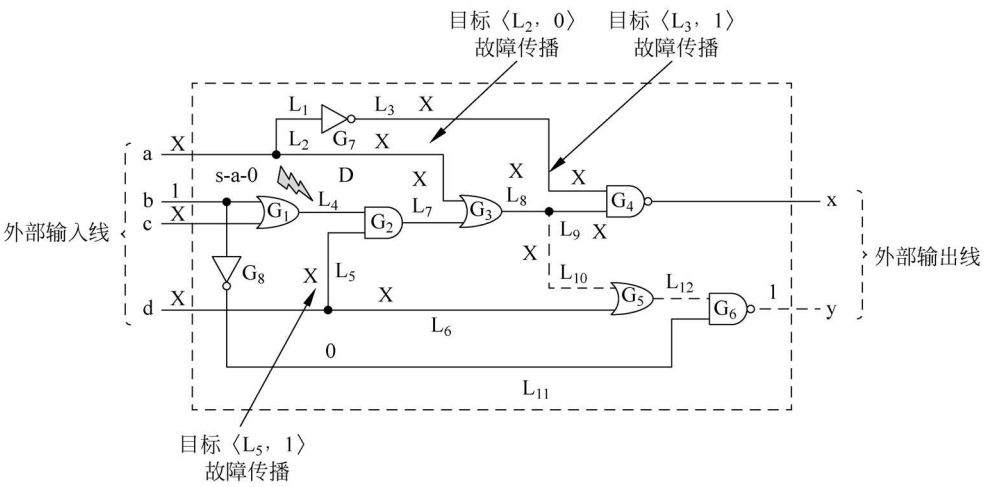
【问题 1】　每次选择哪一根外部输入线并赋予其什么逻辑值（0 还是 1）？

PODEM 利用建立目标（objective）和进行回退（backtrace）操作来解决这个问题。一个目标由一根信号线 g 和对它的赋值 v 所组成，可表示为<g，v>。初始目标（initial objective）为故障激活。如图 3-10(a)所示，若对象故障为 L_4 上的固定为 0 故障（s-a-0）时，初始目标

为<L_4,1>,以使得与故障线 L_4 的 s-a-0 故障的故障值 0 相反的逻辑值 1 出现在 L_4 上。通过对外部输入线 b 赋逻辑值 1 来实现初始目标,结果是使故障效果 D 出现在 L_4 上,这时的 D 前沿为{G_2}。随后,PODEM 利用 X-PATH-CHECK 操作对 D 前沿进行检查,看其中是否有逻辑门具有从其输出线到达至少一根外部输出线的 X 路径(X-PATH)。X 路径是指构成该路径的所有逻辑门的输出皆为未确定值(X)。如图 3-10(b)所示,L_7-L_8-L_9-x 为 X 路径,但 L_7-L_8-L_{10}-L_{12}-y 不是 X 路径,因为 y 的值是 1 而不是未确定值(X)。X 路径 L_7-L_8-L_9-x 意味着故障效果 D 有可能经过它从 L_4 传播到外部输出线 x,从此可以导出若干为实现故障传播而建立的目标。如图 3-10(b)所示,为故障传播建立的目标依次为<L_5,1>,<L_2,0>,<L_3,1>。选择<L_5,1>是因为 D 前沿为{G_2},L_5 为 G_2 的一根输入线,1 是其非控制值,这个目标可以使故障效果 D 经过 G_2 传播到 L_7 上,这也使得 D 前沿由{G_2}变为{G_3}。选择<L_2,0>是因为 D 前沿为{G_3},L_2 为 G_3 的一根输入线,0 是其非控制值,这个



(a) 为故障激活而确定的目标



(b) 为故障传播而确定的目标

图 3-10　确定目标的例子

目标可以使故障效果 D 经过 G_3 传播到 L_8 和 L_9 上，这也使得 D 前沿由{G_3}变为{G_4}。选择<L_3，1>是因为这时的 D 前沿为{G_4}，L_3 为 G_4 的一根输入线，1 是其非控制值，这个目标可以使故障效果 D 经过 G_4 传播到外部输出线 x 上，从而使 L_4 的 s-a-0 故障得到检测。

确定目标的一般算法 objective 的概要如图 3-11 所示。

```
objective（g，V）
{
 /* 对象故障为门g的输出线的固定v故障（s-a-v）*/
    if（门g的输出线尚未赋值）
        /* 目标为故障激活 */
        return（g，v̄）；
    /* 目标为故障传播 */
    从D前沿中选取一个门p；
    g=门p的尚未赋值的输入中选取一个输入；
    v=门p的非控制值；
    returnn（g，v）
}
```

图 3-11　确定目标的算法 objective 的概要

把目标转换成为对外部输入线的赋值是通过回退(backtrace)操作来实现的。针对由一根信号线 g 及对其赋值 v 所组成的目标<g，v>，回退操作需要从信号线 g 逆向退回到一根外部输入线，并确定针对该外部输入线的赋值。在图 3-12 的例子当中，目标是<x，1>，即希望将 1 赋值于外部输出线 x。针对这个目标的回退操作由三个步骤组成。第一个步骤的目的是选择逻辑门 G_6 的一根输入线并决定它应有的逻辑值，以使其有助于目标<x，1>的实现。G_6 是 NAND 门，欲使其输出线 x 为 1，只需要它的任何一根输入线为 0 即可。为使测试生成尽快完成，可先选 G_6 的最容易设置为 0 的输入线 L_8，并确定应出现在 L_8 的逻辑值为 0。第二个步骤是选择逻辑门 G_3 的一根输入线并决定它应有的逻辑值，以使其有助于使 0 出现在 L_8 上。G_3 是 OR 门，欲使其输出为 0，需要它的所有输入为 0。为使测试生成不成功时能被尽早知道，可先选 G_3 的最难设置为 0 的输入线 L_1，并确定应出现在 L_1 的逻辑值为 0。第三个步骤的目的是选择逻辑门 G_1 的一根输入线并决定它应有的逻辑值，以使

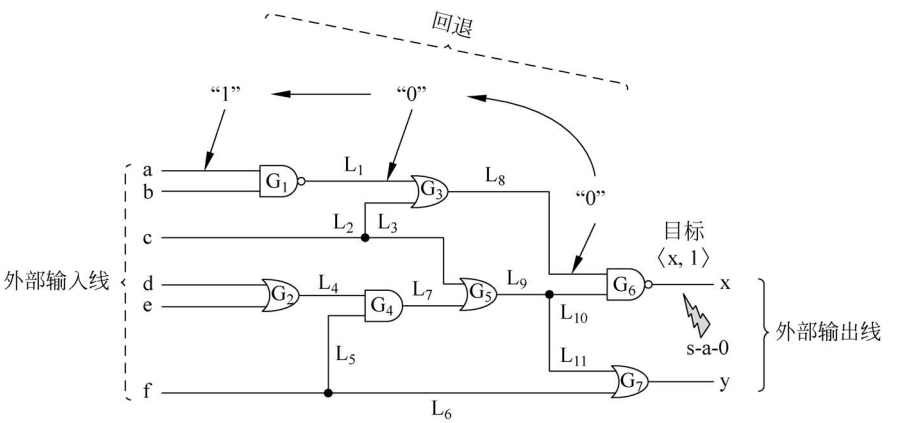


图 3-12　回退操作的例子

其有助于使 0 出现在 L_1 上。G_1 是 NAND 门，欲使其输出为 0，需要它的所有输入线为 1。为使测试生成不成功时能被尽早知道，可先选 G_1 的最难设置为 1 的输入线 a（在这个例子当中亦可选择 b），并确定应出现在 a 的逻辑值为 1。

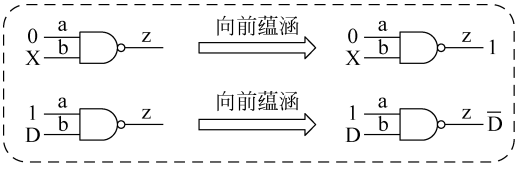
回退操作的一般算法 backtrace 的概要如图 3-13 所示。进行回退往往需要选择逻辑门的某根输入线，它或是最容易设置为某个逻辑值，或是最难设置为某个逻辑值。这种设置为某个逻辑值时的难易度可以用信号线的可控制性来衡量。

```
backtrace（s, v_s）/* 把目标转换成为对外部输入线的赋值 */
{
    v=v_s ;
    while（s为门的输出线）
    {
        if（s对应的门为NAND门，NOR门，NOT门）v=v̄ ;
        if（目标要求对该门的所有输入进行赋值）
            选择s的尚未赋值而且最难控制成逻辑值v的输入线a ;
        else
            选择s的尚未赋值而且最容易控制成逻辑值v的输入线a ;
        s=a ;
    }
    /* 这时的s为一个外部输入线 */
    return（s，）/* 指示把v赋予s*/
}
```

图 3-13　回退算法 backtrace 的概要

【问题 2】　如何进行蕴涵？

蕴涵是指根据某些信号线的现有的逻辑值而自动确定其他一些信号线的应有的逻辑值的操作。在 PODEM 中用到的蕴涵是由对外部输入线的赋值而引起的，是所谓前向蕴涵（forward implication），即根据逻辑门的某些输入信号线的现有逻辑值来确定该门应有的输出值，尽管该逻辑门的其他输入信号线的逻辑值尚未确定。如图 3-14 所示，蕴涵可利用逻辑门的扩大真值表来进行。扩大真值表是针对逻辑值（0 和 1）、未定值（X）以及故障效果（D 或 $\bar{D}$）的逻辑运算，可以通过门的逻辑功能直接建立。



NAND门的扩大真值表

b \ a	0	1	X	D	$\bar{D}$
0	1	1	1	1	1
1	1	0	X	$\bar{D}$	D
X	1	X	X	X	X
D	1	$\bar{D}$	X	$\bar{D}$	1
$\bar{D}$	1	D	X	1	D

图 3-14　向前蕴涵的例子

【问题 3】　如何确认一个测试向量或测试立方是否已被生成？

这可通过故障效果(D 或 $\bar{D}$)是否已经出现在至少一根外部输出线上来判断。若 D 或 $\bar{D}$ 已经出现在至少一根外部输出线上，则该对象故障可以被检测。若此时所有的外部输入线都已被赋予了逻辑值(0 或 1)，则外部输入值的组合为该对象故障的测试向量。若此时至少有一根外部输入线的逻辑值尚未确定(X)，则外部输入值的组合为该对象故障的测试立方。

【问题 4】　如何判断是否可能通过对外部输入线的进一步赋值得到测试向量？

在通过故障激活产生故障效果(D 或 $\bar{D}$)之后，会出现由至少一个输入值为 D 或 $\bar{D}$ 而输出值尚未确定(即为 X)的所有逻辑门组成的集合，即 D 前沿。若对外部输入线的赋值造成 D 前沿变成空集合(即 D 前沿消失)，则意味着不可能通过对外部输入线的继续赋值得到测试向量。在这种情况下，就需要进行回溯等操作。

【问题 5】　如何进行回溯？

如前所述，PODEM 的本质是把测试向量生成当成一个探索问题来解决，其探索空间由全部外部输入线构成。探索过程的管理是通过蕴涵栈(implication stack)来实现的。蕴涵栈的每一个单元包含信号线的名称，该信号线当前所赋的逻辑值(v)，以及是否已经对该信号线尝试过赋相反的逻辑值($\bar{v}$)的信息。图 3-15 所示的蕴涵栈表示如下的探索过程：首先是对 a 已赋值 0，其相反值 1 已被尝试赋过；其次是对 c 已赋值 1，其相反值 0 尚未被尝试赋过；之后是对 d 已赋值 0，其相反值 1 已被尝试赋过。如图 3-15 所示，这个探索过程也可用二叉树来表示。

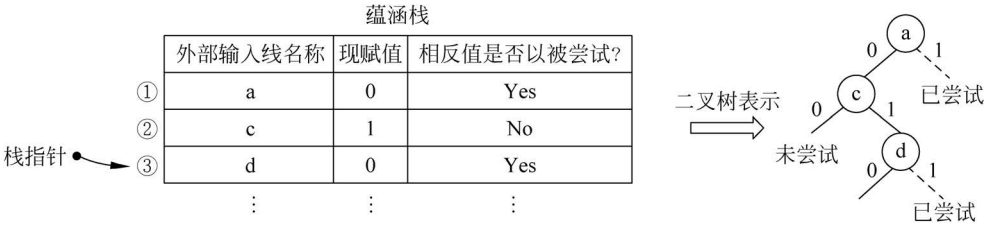


图 3-15　利用蕴涵栈的探索过程管理

在图 3-15 的例子当中，若对外部输入线的现有赋值<a=0，c=1，d=0>导致了 D 前沿消失，则在外部输入线的现有赋值的基础上继续对外部输入线赋值就不可能最终产生测试向量。在这种情况下，就应该进行回溯操作。在图 3-15 的例子当中，会从栈指针指向的单元③(<d，0，Yes>)开始回溯操作。这个单元表示外部输入线 d 的现有赋值为 0 而且其相反值 1 已被尝试赋过，也就是说，关于 d 已无任何选择余地。因此，PODEM 会删除单元③，并取消对外部输入线 d 的赋值(恢复为未确定值 X)，栈指针指向单元②。单元②(<c，1，No>)表示外部输入线 c 的现在赋值为 1 但其相反值 0 尚未被尝试赋过。在这种情况下，PODEM 会把对外部输入线 c 的赋值从 1 改为 0，并判断对外部输入线的赋值组合<a=0，c=0>是否可能最终产生测试向量，回溯操作就是这样进行的。

参照以上的 PODEM 的概念性流程的说明，就比较容易理解 PODEM 算法的基本原理和思路。

3.4 测试精简

在测试生成过程中一般需要进行测试精简，其目的是减少生成的最终测试向量的数量，从而降低所需的测试时间以降低测试开销。测试精简有两种类型，即在测试生成进行当中实施的动态压缩（dynamic compaction）和在测试生成完成之后实施的静态压缩（static compaction），它们有时也被统称为测试精简。

针对一个故障所进行的测试生成的结果，是对电路外部输入线的一个赋值组合。在一般情况下，这个组合除了确定了的逻辑值（0，1）之外，往往还含有未确定值（X）。如 3.3.1 节所述，这样的由 0、1、X 组成的赋值组合称为测试立方。为了区别起见，仅由 0 和 1 组成的赋值组合往往被称为测试向量。

在图 3-8 所示的例子当中，信号线 L_7 的 s-a-0 故障的测试生成的直接结果为测试立方 <a=0，b=1，c=X，d=1>。对拥有许多外部输入线的大型电路来说，其测试立方往往含有非常多的未确定值（X）。很明显，在现有的测试立方的基础之上可以继续进行测试生成，以通过对其中的未确定值（X）进一步赋值来检测其他的故障。这个操作就是动态压缩，它可以使一个测试立方或测试向量同时检测更多的故障，从而减少整个电路所需的测试向量的总数。动态压缩的一般过程如下：在每次开始测试生成时，先从尚未检测的故障当中选择一个主故障（primary fault）并生成其测试立方；之后，从尚未检测的故障中选择一个副故障（secondary fault）（步骤 1），并在主故障的测试立方的基础之上生成其测试立方（步骤 2）；重复步骤 1 和步骤 2，直到现有的测试立方中的未确定值（X）已变得很少或已经很难生成可以检测副故障的测试立方为止。一般情况下，通过动态压缩可以检测不止一个副故障。

动态压缩结束后的测试立方称为最终测试立方，它一般仍含有一些未确定值 X。图 3-16 为实施了动态压缩的测试生成的全过程当中所产生的所有最终测试立方中的未确定值 X 的含有率的示例。从图中可见，在测试生成的初始阶段，实施动态压缩以利用测试立方中的未确定值 X 来检测副故障比较容易成功，导致最终测试立方中的残余未确定值 X 的含有率

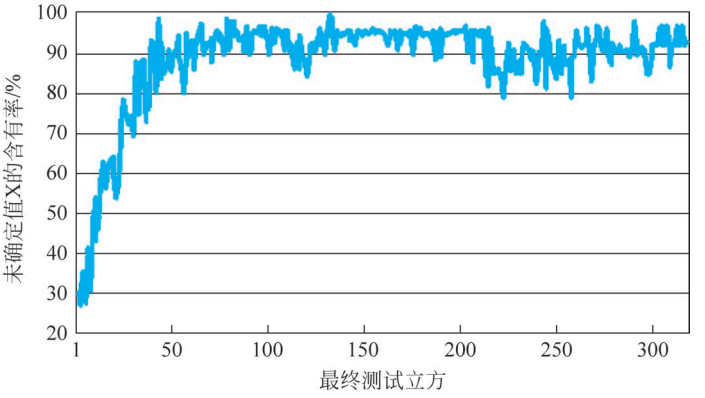


图 3-16 测试生成过程中的最终测试立方的未确定值 X 的含有率的示例

较低。但是，随着测试生成过程的进行，针对副故障的测试生成的成功率下降，造成最终测试立方中的未确定值 X 的含有率非常高。

由上述可见，即使在测试生成过程中实施了动态压缩，其结果仍会是含有大量未确定值 X 的测试立方。对于这些测试立方，可以利用静态压缩来进一步减少测试所需的最终测试立方的数量。如图 3-17 所示，静态压缩是指把两个相互兼容的测试立方合并为一个测试立方的操作。在图 3-17 的例子当中，有两个测试立方，$\boldsymbol{V}_x=<0,X,1,X,X>$与 $\boldsymbol{V}_y=<0,1,X,1,X>$，它们相互兼容，因此可以把它们合并为一个新的测试立方 $\boldsymbol{V}_z=<0,1,1,1,X>$。显然，这个新的测试立方($\boldsymbol{V}_z$)可以检测两个旧的测试立方($\boldsymbol{V}_x$ 与 $\boldsymbol{V}_y$)所能检测的所有故障。由此可见，通过实施静态压缩可以有效地减少测试立方的数量而不影响故障检测能力。

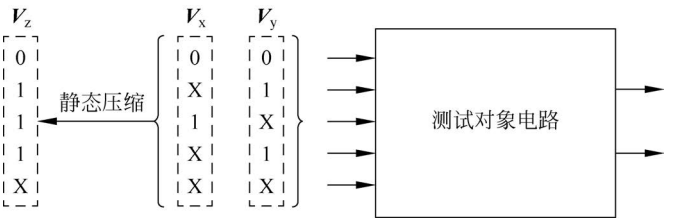


图 3-17　静态压缩的例子

一般情况下，在实施了动态压缩和静态压缩之后，测试生成的结果依然可能还是测试立方，即仍然含有一些未确定值(X)。因为实际测试中使用的 ATE 只能接受处理逻辑值(0，1)，测试立方中的未确定值 X 必须依靠某种方法设置为 0 或 1 之后才能用于实际测试，这个操作称为 X 填充(X-filling)。最为直接的和常用的 X 填充方法是随机填充(random-fill)，即随机地选择 0 或 1 来对测试立方中的每一个 X 进行赋值。随机填充的结果是只由 0 和 1 组成的测试向量，可以应用于实际测试。值得注意的是，一个测试立方经过随机填充变成一个测试向量后，往往可以检测出一些新的故障。这个现象称为偶然检测(fortuitous detection)，它有助于减少测试向量数。但是，随机填充也有一些缺点，比如容易造成测试功耗的上升等。因此，如果需要降低测试向量所产生的测试功耗，就需要使用随机填充以外的 X 填充方法，比如 0 填充(0-Fill)，1 填充(1-Fill)等。

3.5　时延故障的测试生成

对芯片来说，不仅要保证其输出值的正确性，还要保证其具有设计要求的性能。也就是说，芯片电路的外部输入线的逻辑值跳变(0→1 或 1→0)的影响必须能够在设计要求的时间内反映在该电路的外部输出线。然而，在芯片制造过程当中，有许多因素(比如，晶体管或信号线的缺陷)会使其工作速度达不到设计要求。时延故障(delay fault)就是用来对造成逻辑电路的外部输入线的跳变不能在设计要求的时间内反映在外部输出线的现象进行建模的故障模型。代表性的时延故障包括跳变时延故障(transition delay fault)和通路时延故障(path delay fault)。在实际当中经常需要对跳变时延故障生成测试向量，以下简单地介绍

跳变时延故障的测试生成的基本原理。

跳变时延故障是指逻辑电路的某根信号线的时延过长，导致该信号线上的逻辑跳变(0→1 或 1→0)造成的影响不能在设计要求的时间内到达逻辑电路的外部输出线。跳变时延故障有两种类型，即上升缓慢(slow-to-rise)型和下降缓慢(slow-to-fall)型。前者是指对象故障信号线的逻辑跳变为 0→1，后者是指对象故障信号线的逻辑跳变为 1→0。与固定型故障相似，逻辑电路的每一根信号线对应两个跳变时延故障，一个为上升缓慢型，另一个为下降缓慢型。

图 3-18 为上升缓慢型跳变时延故障的例子，其中，对象信号线为 L_7，其时延过长，从而导致 L_7 上的上升型逻辑跳变 0→1 造成的影响(即外部输出端 x 上的逻辑跳变 1→0)不能在设计要求的时间内(T_2 与 $T_2+\Delta$ 之间)到达逻辑电路的外部输出线 x。在这个例子当中，L_7 上的上升型逻辑跳变 0→1 是由发生在 T_2 时刻对原有值为 0 的外部输入线 b 赋值 1 造成的。设计上要求这个发生在外部输入线 b 上的逻辑跳变 0→1 的影响能在时间间隔 Δ 之内反映到外部输出线上。因此，若信号线 L_7 的时延过长，造成外部输出线 x 上的逻辑跳变 1→0 不能在 Δ 之内出现，则 L_7 的上升缓慢型跳变时延故障被检测。具体来说，若在观测时间点 $T_2+\Delta$ 实际观测到的外部输出线 x 的逻辑值为 0，则 L_7 的上升缓慢型跳变时延故障不存在或未被检测。若在观测时间点 $T_2+\Delta$ 实际观测到的外部输出端 x 的逻辑值为 1，则 L_7 的上升缓慢型跳变时延故障被检测。

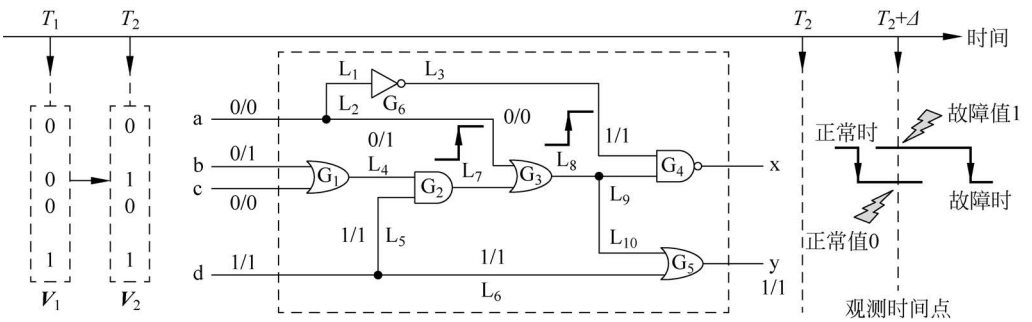


图 3-18　跳变时延故障及其测试向量

由图 3-18 的例子可以看出，信号线 L_7 的上升(0→1)缓慢型跳变时延故障的检测需要一对输入向量$\{\boldsymbol{V}_1,\boldsymbol{V}_2\}$，其中 $\boldsymbol{V}_1$(称为初始向量，在 T_1 时刻施加于逻辑电路)将 L_7 设置为 0，而 $\boldsymbol{V}_2$(称为测试向量，在 T_2 时刻施加于逻辑电路)将 L_7 设置为 1 并且将 L_7 上的故障效果传播到至少一个外部输出线上。因此，$\boldsymbol{V}_2$ 实际上是一个针对 L_7 的固定型故障 s-a-0 的测试向量，可以利用诸如 PODEM 之类的测试生成算法来生成。

在图 3-18 的例子中，做如下假设：

d_1 = b-G_1-L_4-G_2 的传输时延

d_f = L_7 的上升缓慢型跳变时延故障的时延

d_2 = G_3-L_8-L_9-G_4-x 的传输时延

在这种情况下，L_7 的上升缓慢型跳变时延故障被检测出的条件为：

$$d_1+d_f+d_2>\Delta$$

其中，d_1+d_2 是为检测 L_7 的跳变时延故障而被敏化的通路（b-G_1-L_4-G_2-L_7-G_3-L_8-L_9-G_4-x）的正常传输时延。当故障时延 d_f 较大时，即便这个敏化通路的时延比较小，L_7 的跳变时延故障亦可被检测出；但是，当 d_f 较小（微小时延故障）时，若这个敏化通路的时延比较小，L_7 的跳变时延故障即使存在也有可能不被检测出。在实际当中，通路的时延大小往往与通路的长短相关。而且，d_f 较小的时延故障，即微小时延故障，随着集成电路微细加工技术的发展呈现多发的趋势。因此，较短的敏化通路无法检测出微小时延故障的问题越来越严重。

如图 3-19(a)所示，通常的自动测试向量生成（ATPG）工具不考虑敏化通路的长短，在这个例子当中，敏化通路为 P_{12}-L-P_{22}，其长度较短，因此即使成功地生成了跳变时延故障的测试向量，也可能无法检测出 L 上的微小时延故障。为了解决这个问题，可以利用如图 3-19(b)所示的考虑时序的 ATPG 工具。这种 ATPG 会试图敏化尽可能长的通路，在这个例子当中为 P_{11}-L-P_{23}，其长度较长，因此有较大的可能利用所生成的测试向量成功地检测出 L 上的微小时延故障。一般来讲，考虑时序的 ATPG 生成的测试向量更有可能检测出微小时延故障，但相对于通常的不考虑时序的 ATPG，考虑时序的 ATPG 的测试生成时间较长，生成的测试向量的数量也较多。

(a) 敏化通路较短的例子　　(b) 敏化通路较长的例子

图 3-19　不同的 ATPG 的敏化通路的差异

3.6　实例介绍

测试生成是芯片测试设计实现当中的一个重要步骤。本节简单地介绍一下利用商用软件工具（Tessent）生成测试向量的基本流程，以便读者对测试生成有一些具体的感知。基于 Tessent 的测试生成建立在原有电路设计中插入扫描链的基础上。图 3-20 包含了完整的插入扫描链电路（第 4 章介绍）和利用 ATPG 工具生成测试向量的流程，其基本步骤如下。

（A1）**获得原始设计信息**：准备和综合 RTL 级的芯片设计。

（A2）**插入扫描链电路**：运行 Tessent Shell，以综合后的门级网表和 DFT 库文件作为输入，自行指定扫描链的长度和数量以及其他插入的要求，使用 Tessent Scan 在门级网表中插入扫描链。

（A3）**获得测试设计信息**：生成并保存带扫描链电路的门级网表以及 TCD（Tessent Core Description）文件。TCD 文件用于 ATPG，包含了扫描链的定义和操作过程。

（A4）**生成测试向量**：运行 Tessent Shell，以 A2 中插完扫描链之后的门级网表和 TCD

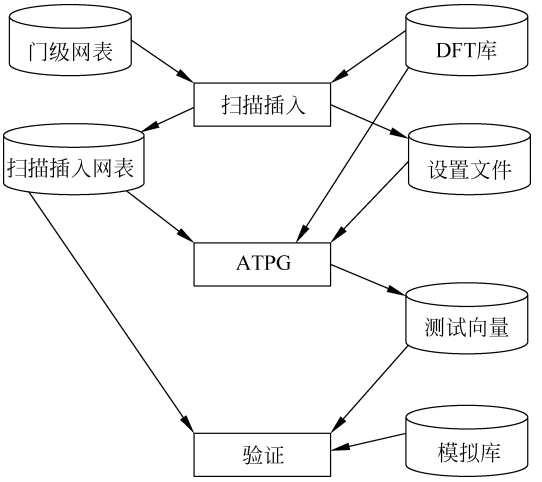


图 3-20　测试向量生成的流程

文件作为输入，以 Tessent FastScan 作为 ATPG 工具来生成测试向量。测试向量可以根据需要保存成不同格式，如 STIL 和 Verilog 等。

（A5）**功能仿真**：用逻辑模拟器模拟（A4）中产生的测试向量来完成功能仿真，以发现可能存在的问题。通常的要求是模拟所有的并行测试向量和选出的若干串行测试向量。

利用商用软件工具（Tessent）生成测试向量时用到的主要操作命令如下。

（B1）通过命令行启动 Tessent Shell，其默认模式是配置（Setup）模式。

```
$ tessent -shell
```

（B2）通过 set_context 命令，将工具的 context 设成使用 ATPG 生成测试向量的环境。

```
SETUP > set_context patterns - scan
```

（B3）通过 set_tsdb_output_directory 定义保存修改后的设计以及其他输出文件的 tsdb 目录位置。

```
SETUP > set_tsdb_output_directory ../tsdb_outdir
```

（B4）通过 read_design 工具自动在 tsdb 目录下寻找并加载插完扫描链的门级网表以及相关设计信息。

```
SETUP > read_design cpu_top -design_id gate -verbose
```

（B5）通过 read_cell_library 命令载入单元模拟库。

```
SETUP > read_cell_library ../library/standard_cells/tessent/adk.tcelllib
SETUP > read_cell_library ../library/memories/picdram.tcelllib
```

（B6）通过 set_current_design 命令设置工具处理的当前设计。

```
SETUP > set_current_design
```

（B7）通过 set_current_mode 定义当前的测试模式的名字为 scan_stuck，默认类型为

unwrappered。

```
SETUP > set_current_mode scan_stuck
```

（B8）通过 import_scan_mode 导入 TCD 文件，即 scan_mode 的配置，包括插入的 scan chain 组成信息。

```
SETUP > import_scan_mode scan_mode
```

（B9）通过 set_system_mode 命令，将系统模式切换到 Analysis 模式，该模式进行电路扁平化、电路学习和设计规则验证。

```
SETUP > set_system_mode analysis
```

（B10）通过 create_patterns 命令生成针对单固定型故障的测试向量。

```
ANALYSIS > create_patterns
```

（B11）通过 write_tsdb_data 保存 flat model、TCD、patdb 格式的测试向量以及故障列表到 tsdb 中。

```
ANALYSIS > write_tsdb_data - replace
```

（B12）通过 write_flat_model 生成 flat model 用于后续的诊断过程。

```
ANALYSIS > write_flat_model results/pipe.flat - replace
```

（B13）通过 write_patterns 命令保存不同格式的测试向量。STIL 格式用于 ATE 测试，Verilog 格式用于功能验证。

```
ANALYSIS > write_patterns results/cpu_top_stuck.stil  - stil - parallel  - replace
ANALYSIS > write_patterns results/cpu_top_stuck_par.v - verilog - parallel - replace
ANALYSIS > set_pattern_filtering - sample 2
ANALYSIS > write_patterns results/cpu_top_stuck_ser.v - verilog - serial - replace
```

（B14）退出 Tessent Shell。

```
ANALYSIS > exit
```

3.7　本章小结

测试生成是集成电路测试的核心技术之一。本章在简述了测试生成的基本概念和基本要求之后，对非面向故障型及面向故障型的测试生成方法的基本原理做了介绍。本章的重点在于通过固定型故障的测试生成来说明通路敏化法的基本原理并介绍其代表性算法 PODEM。本章还介绍了测试生成当中常用的两种测试精简手法（动态压缩和静态压缩），以及跳变时延故障测试生成的基本原理。目前，已有一些开源的 ATPG 项目可供参考[①]。

随着集成电路设计和制造技术的不断发展，电路规模不断增大，制造缺陷的类型不断增

① https://www.gitlink.org.cn/opendacs/oATPG

多，车载电子等高可靠性应用对测试质量的要求不断提高，低功耗高性能电路对降低测试功耗的要求更加紧迫，使测试生成不断面对许多新的挑战。利用分布式并行算法及高性能通用图形处理器(GPGPU)来提高测试生成系统的性能，利用新的故障模型或检测条件(例如，Cell-Aware ATPG、Timing-Aware ATPG 等)来提高测试向量的质量，利用人工智能来提高测试生成算法的效率，利用 X 填充来降低测试功耗等新的测试生成技术正在不断涌现，值得持续关注。

3.8 习题

3.1 某电路由 7 根外部输入线(a,b,c,d,e,f,g)和 3 根外部输出线(x,y,z)组成；从电路结构上看，x 只可能受到 a、b、c 的影响，y 只可能受到 c、d、e、f 的影响，z 只可能受到 e、f、g 的影响。为这个电路设计一个尽量小的伪穷举测试集合。

3.2 简要说明回溯(backtrack)和回退(backtrace)的区别。

3.3 给出 OR 门的扩大真值表。

3.4 图 3-21 所示电路的单固定型故障的总数是多少？测试向量 $\boldsymbol{V}$(<a =0,b=1,c=1,d=0,e=1>)一共可检测几个故障？

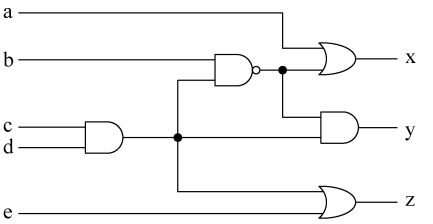


图 3-21 单固定型故障测试生成的练习电路

3.5 利用 PODEM 分别为图 3-22 所示电路中的以下两个故障生成测试立方或测试向量。

(1) 信号线 L_5 的固定为 0 故障(s-a-0)；

(2) 信号线 L_4 的固定为 1 故障(s-a-1)。

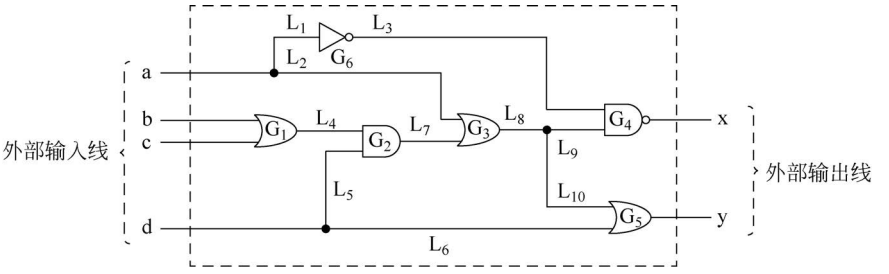


图 3-22 用 PODEM 算法进行测试生成的练习电路

参 考 文 献

[1] FUJIWARA H. Logic testing and design for testability[M]. Cambridge, MA: The MIT Press, 1985.

[2] ABRAMOVICI M, BREUER M A, FRIEDMAN A D. Digital systems testing and testable design [M]. New York: Computer Science Press, 1990.

[3] CROUCH A L. Design-for-test for digital IC's and embedded core systems[M]. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

[4] BUSHNELL M L, AGRAWAL V D. Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits[M]. Boston: Kluwer Academic Publishers, 2000.

[5] WANG L T, WU C W, WEN X. VLSI test principles and architectures: Design for testability[M]. San Francisco: Elsevier, 2006.