第 5 章

逻辑内建自测试

逻辑芯片测试的全过程包括 4 个主要步骤，即针对芯片生成测试激励(第 1 步)，对芯片施加测试向量(第 2 步)，获得芯片实际测试响应(第 3 步)，以及通过与期待测试响应比较得出测试结果(第 4 步)。芯片测试的基本方式为向量储存型测试(stored pattern test)，也就是说，第 1 步通过使用 ATPG 工具完成，而第 2～4 步由 ATE 来完成。向量储存型测试的优点是可以利用的测试向量种类丰富而且测试能力强，且故障诊断较为方便。但是，向量储存型测试的缺点是 ATE 需提供巨大的存储空间以保存全部测试向量，而且还需要具有高速的信号输入/输出和处理能力，导致 ATE 体积大、价格高，使之只能用于芯片的出厂测试而无法应用于芯片被组装入系统之后的测试。

芯片测试的另一个主要方式为内建自测试(Built-In Self-Test，BIST)，它是一种可测试性设计(Design for Test，DFT)技术，其基本思路是对原有电路增加一些特殊设计以使其能够不借助 ATE 就可以对其自身主要部分进行测试。以逻辑电路为对象的内建自测试技术称为逻辑内建自测试(Logic BIST)。逻辑内建自测试的优点是不需要体积大价格高的 ATE，从而可以有效地降低测试开销；比较容易提供高质量测试所需的高速信号输入/输出及处理能力；比较容易产生和利用极为大量的测试激励以增加检测芯片内部各种复杂缺陷的可能性，特别是在芯片被组装入系统之后也可以对芯片反复进行测试。逻辑内建自测试的缺点是需要对被测电路做一些增改，在实际设计上难度较高，而且测试向量种类和测试能力往往受到一定限制，另外故障诊断较为困难。在芯片广泛应用于航空航天、国防、汽车、银行、医疗保健、网络、电信行业等的今天，出厂时的良品芯片由于使用中的老化而产生新的缺陷的问题已成为对系统的整体可靠性的重大威胁。因此，逻辑内建自测试已经超出了单纯的芯片测试的范畴，成为提高和保证整个系统的可靠性的关键技术手段。

本章首先介绍逻辑内建自测试的基本结构，特别是得到工业界广泛应用的 STUMPS (Self-Test Using a MISR and Parallel Shift register sequence generater)结构。随后，对逻辑内建自测试所要求的测试对象电路的基本设计规则和改善故障覆盖率的方法进行介绍。接下来，重点介绍适用于逻辑内建自测试实现的多种测试向量生成和测试响应分析技术，以及逻辑内建自测试常用的时序控制方法。本章末尾以一个设计实践为例，介绍使用商用

Logic BIST 工具进行逻辑内建自测试的所有必要步骤，以使读者对逻辑内建自测试有进一步的感性认识。

5.1　基本结构

逻辑内建自测试的基本结构如图 5-1 所示，它由 4 个主要部分组成，即 BIST 对象电路，测试向量生成器（Test Pattern Generator，TPG），测试响应分析器（Test Response Analyzer，TRA）和 BIST 控制器。

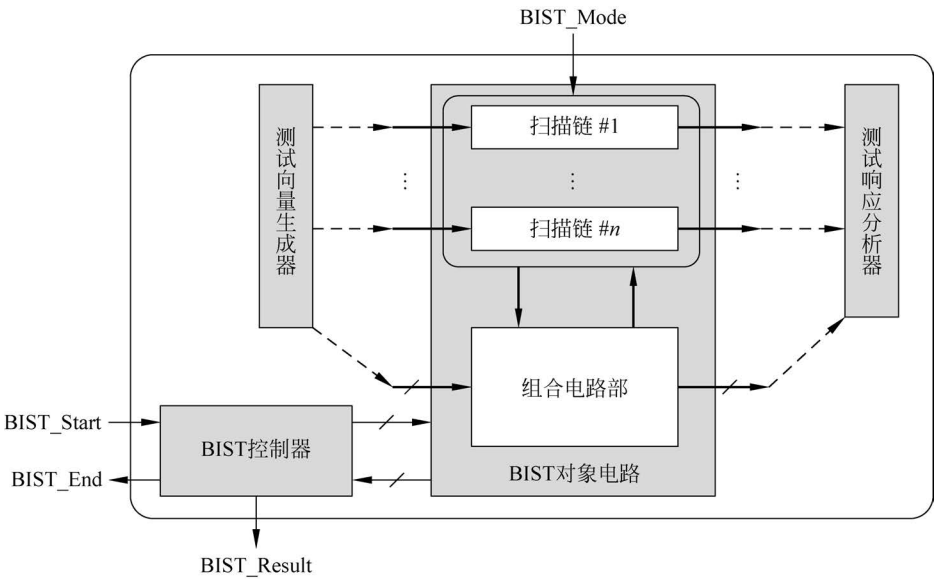


图 5-1　逻辑内建自测试的基本结构

BIST 对象电路的测试由一个 BIST_Mode 信号控制，当 BIST_Mode＝1 时电路进入 BIST 测试模式，启动自测试的运行。BIST 对象电路的核心是具有一条或多条扫描链的全扫描电路，测试向量的输入和测试响应的输出都利用扫描链。因为逻辑内建自测试下的测试向量的产生和测试响应的分析都在芯片内部进行，无须与外界发生数据交换，所以可使 BIST 对象电路拥有很多条扫描链，以缩短扫描链长度，从而减少测试向量的输入和测试响应的输出所需的时间。BIST 对象电路的最基本要求是被测电路当中出现的未确定值（X）不能进入测试响应分析器，以保证可用简单的电路设计来实现测试响应分析。此外，BIST 对象电路往往还需要加入一些叫作测试点的特殊电路，以提高测试向量生成器所产生的伪随机向量的故障覆盖率。

目前在工业界获得广泛应用的测试向量生成器（TPG）和测试响应分析器（TRA）的设计主要基于 20 世纪 80 年代由 IBM 公司提出的 STUMPS 方式。在测试向量生成方面，它利用线性反馈移位寄存器（Linear Feedback Shift Register，LFSR）来产生原始伪随机向量，

再利用移相器(Phase Shifter,PS)增强其随机性后作为测试向量使用。在测试响应分析方面,它利用多输入特征寄存器(Multiple Input Signature Register,MISR)来对测试响应进行压缩以获得最终特征,再把最终特征与期待特征进行比较来决定测试结果。BIST 控制器可由一个输入控制信号(比如 BIST_Start)加以启动,它的作用是生成所需的 BIST 时序控制信号,包括扫描驱动信号和时钟信号,以协调 BIST 对象电路、测试向量生成器和测试响应分析器之间的相关操作。一旦逻辑内建自测试的全部操作完成,BIST 控制器可提供一个输出信号(比如 BIST_End)显示测试完成,并通过另一个输出信号(比如 BIST_Result)来报告测试结果是通过(Pass)还是失败(Fail)。

5.2 BIST 对象电路

逻辑内建自测试的 BIST 对象电路的核心是全扫描电路,与一般的扫描设计相比,它需要进一步遵循许多与未确定值(X)的传播有关的设计限制。这是因为任何直接或间接地传播到测试响应分析器的未确定值(X)都会使特征失去单一性,从而导致无法简单有效地得出测试结果。比如,假设在 BIST 测试过程当中的某个时刻的特征由 8 位逻辑值组成。若其中 1 位变成了未确定值(X),就会导致必须考虑 2 个特征,分别对应 X=0 或 X=1 的情况。这样,若在 BIST 测试过程当中不断有未确定值(X)传播到测试响应分析器,就会导致必须考虑的特征的数量爆发性地增加,致使无法用与一个期待特征进行比较的方式快速有效地确定 BIST 的测试结果。因此,在逻辑内建自测试的设计当中,必须对未确定值进行屏蔽,以防止它们直接或间接地传播到测试响应分析器。

除了需要对未确定值进行屏蔽之外,BIST 对象电路往往还需要加入一些测试点以提高逻辑内建自测试的故障覆盖率。这是因为在电路面积开销的限制下,测试向量生成器往往只能利用构造简单的线性反馈移位寄存器所产生的伪随机向量作为测试向量,而这样的测试向量往往很难满足一些故障检测的条件,造成难以达到较高的故障覆盖率。这就需要加入一些测试点以增加伪随机向量检测故障的可能性。

5.2.1 未确定值屏蔽

一般来说,在进行逻辑内建自测试时,作为测试对象的逻辑电路中有可能会出现一些未确定值发生源(X 源),即在 Logic BIST 模式下(BIST_Mode=1)产生未确定值(X)的信号线。比如,大规模系统芯片往往同时拥有逻辑电路和内置存储器。在 Logic BIST 模式下(BIST_Mode=1),由于内置存储器不是测试对象而不受任何控制,因此其输出值往往为未确定值,从而使其输出线成为 X 源。能够直接或间接地将其未确定值(X)传播到测试响应分析器(TRA)的任何 X 源,都必须使用相应的可测试性设计的手法进行未确定值屏蔽(X 屏蔽)。

图 5-2 是几种基本的 X 屏蔽方法,用于在 Logic BIST 测试模式下(BIST_Mode=1)对 X 源进行屏蔽。图 5-2(a)为 0-控制点,它在 Logic BIST 测试模式下(BIST_Mode=1)通过将 Z 强制设置为 0 来屏蔽 X 源的未确定输出;图 5-2(b)为 1-控制点,它在 Logic BIST 测试

模式下(BIST_Mode=1),通过将 Z 强制设置为 1 来屏蔽 X 源的未确定输出;图 5-2(c)为旁路控制点,它在 Logic BIST 测试模式下(BIST_Mode=1),通过将 Z 切换到某个可变化的外部输入信号来屏蔽 X 源的未确定输出;图 5-2(d)为利用逻辑门的旁路控制点,它在 Logic BIST 测试模式下(BIST_Mode=1),通过将 Z 切换到电路内部的某个逻辑门的输出信号来屏蔽 X 源的未确定输出;图 5-2(e)为利用扫描单元的旁路控制点,它在 Logic BIST 测试模式下(BIST_Mode=1),通过将 Z 切换到某个扫描单元的输出信号来屏蔽 X 源的未确定输出。0-控制点和 1-控制点虽然都可以屏蔽掉未确定值,但是控制点的输出 Z 分别被固定为 0 和 1,对提高故障覆盖率不利。旁路控制点通过用某个逻辑值可变的信号来驱动控制点的输出 Z,使之有助于故障覆盖率的提高;特别是利用扫描单元的旁路控制点的输出 Z 出现各种逻辑值变化的可能性最大,因此也就最有助于故障覆盖率的提高。

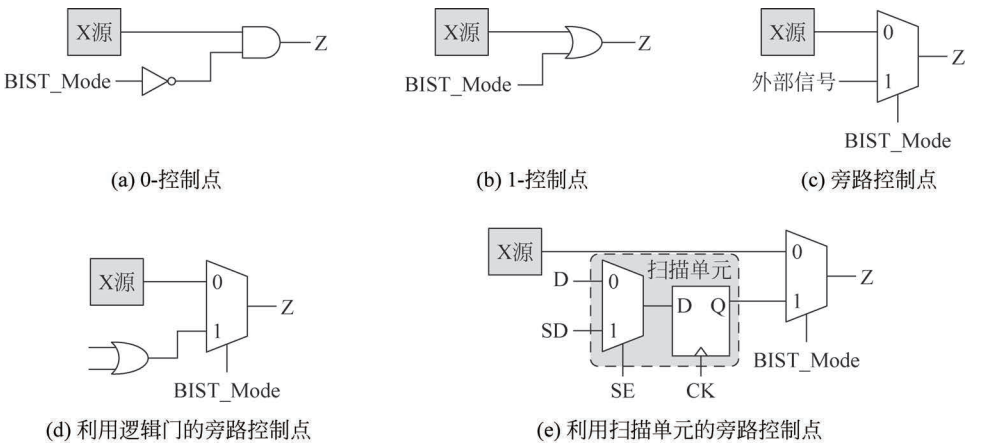


图 5-2　X 屏蔽的基本方法

在准备 BIST 对象电路时,应该根据每个 X 源的特征和性质,选择包括上述基本方法在内的最佳 X 屏蔽方法,以期望在实现 X 屏蔽功能的同时,尽量减少电路面积开销和对电路时序的影响。下边介绍几种典型的 X 源屏蔽方法。

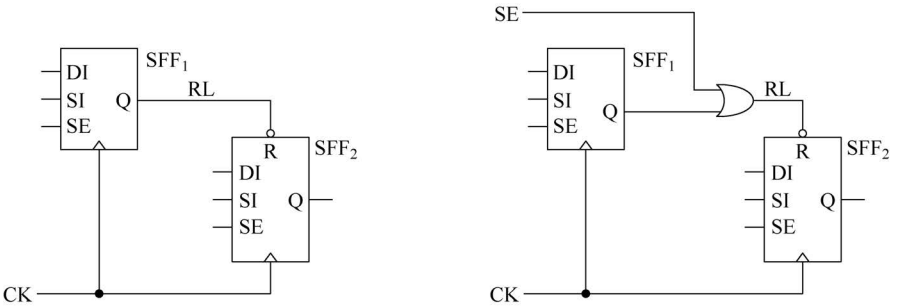
模拟电路模块。芯片当中的模拟电路模块的典型例子是模/数转换器(Analog-to-Digital Converter,ADC)。在 BIST 测试模式下,任何有可能出现未确定值(X)的模拟电路模块的输出都必须强制设置为确定值。这可以通过插入 0-控制点、1-控制点、旁路控制点来实现。

存储器和非扫描存储单元。对于芯片当中的存储器(包括 DRAM、SRAM、闪存等)和非扫描存储单元(包括 D 触发器、D 锁存器等),常用旁路控制点来屏蔽来自它们的未确定值(X)。另一种方法是使用初始化序列将存储器或非扫描存储单元设置为某种确定的状态,这样可以避免增加旁路控制点导致的关键通路的时延增加,但是必须确保在整个 BIST 操作过程中存储状态都不会遭到损坏。

组合反馈环路。应该尽量避免在电路设计当中使用组合反馈环路。如果这样做不可避免的话,则必须在组合反馈环路上插入 0-控制点、1-控制点或旁路控制点,以保证在 BIST

测试模式下切断每个组合反馈环路。

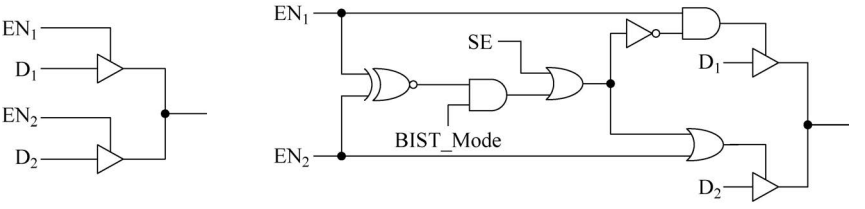
异步置位/复位信号。逻辑内建自测试的基础是逻辑电路的扫描设计，因此如果异步置位/复位信号在扫描移位操作期间变为有效，会导致扫描链中的测试数据遭到破坏。这个问题可以使用如图 5-3 所示的方法解决。为了在扫描移位操作期间(SE＝1)使图 5-3(a)所示异步复位信号 RL 无效(RL＝1)，图 5-3(b)中增加了一个由 SE 控制的 OR 门，使 RL 在扫描移位操作期间(SE＝1)只能取 1，从而防止扫描存储单元 SFF_2 被强制复位。



(a) 异步复位信号　　(b) 在扫描移位操作期间禁止异步复位的电路设计

图 5-3　X 异步复位信号的处理方法

三态总线。当多个驱动单元(通常为三态门)向同一条总线上施加不同的逻辑值时，就会导致总线竞争，从而产生过大电流而对电路造成损坏。如图 5-4(a)所示，在电路正常工作时会通过对 EN_1 和 EN_2 的适当控制来保证任何时刻只有一个三态门打开。但是在 BIST 测试模式下(BIST_Mode＝1)，由于伪随机向量的使用，在移位(SE＝1)和捕获(SE＝0)操作期间都可能导致两个三态门同时打开而产生总线竞争。为了解决这个问题，可利用图 5-4(b)的电路设计来确保在每次移位(SE＝1)或捕获(SE＝0)操作期间仅允许一个三态门来驱动总线。



(a) 三态总线　　(b) 避免总线竞争的电路设计

图 5-4　针对三态总线的 X 屏蔽方法

假通路。假通路(false path)是在电路正常工作状态下不会被激活的通路，因此它们通常不符合时序要求。但是在 BIST 测试模式下，由于伪随机向量的使用，假通路有可能被激活，导致针对时延故障的逻辑内建自测试错误地报告测试失败，从而使良品芯片无法通过测试。为避免这个问题，每个假通路应插入一个 0-控制点或 1-控制点以屏蔽其影响。

关键通路。关键通路(critical path)是对时序敏感的功能路径,它的时延决定整个电路的工作速度,因此应尽可能避免在关键通路上添加其他逻辑门,以防止增加它的时延。为了屏蔽关键通路中的未确定值(X),可在关键通路上向选定一个门(例如 NOT 门、NAND 门或 NOR 门)并为其添加一根输入线,以最大限度地减少在关键通路上的时延增加。如图 5-5 所示,可以选择一个 NOT 门(图 5-5(a)),在 BIST 测试模式下(BIST_Mode=1)把它变成一个 0-控制点(图 5-5(b))或一个 1-控制点(图 5-5(c))。

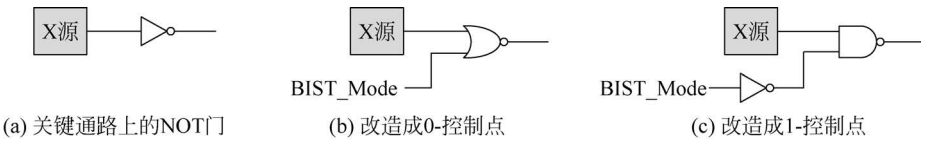


图 5-5　针对关键通路的 X 屏蔽方法

多周期通路。多周期通路(multiple-cycle path)是正常的功能通路,但是信号通过时需要两个或更多时钟周期。与假通路类似,它可能导致针对时延故障的逻辑内建自测试错误地报告测试失败,从而使良品芯片无法通过测试。为避免这个问题,每个多周期通路应插入一个 0-控制点或 1-控制点以屏蔽其影响。

悬空的输入或输出端口。在 BIST 测试模式下,任何外部输入或外部输出端口都不能悬空,这些端口必须正确连接到电源或接地。另外,也必须避免任何内部的模块有悬空的输入,因为这有可能将未确定值(X)传播到测试响应分析器(TRA)。

双向 I/O 端口。双向 I/O 端口在电路设计中很常见。为了正确地进行逻辑内建自测试,应确保每个双向 I/O 端口的方向在 BIST 测试模式下(BIST_Mode=1)固定为输入或输出。图 5-6 是一个将双向 I/O 端口强制设为单向输出的例子。

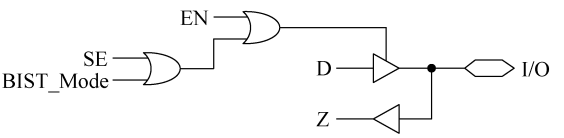


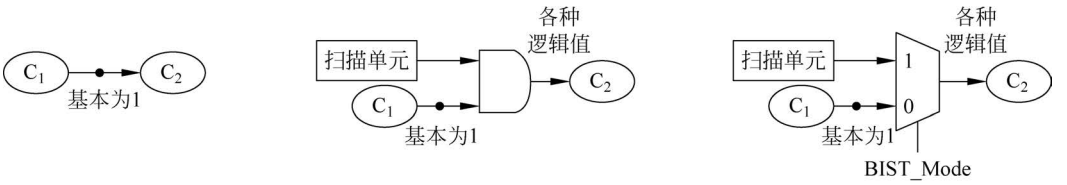
图 5-6　将双向 I/O 端口强制设为单向输出的例子

5.2.2　测试点插入

逻辑内建自测试通常使用伪随机向量作为测试激励,这会造成电路中的某些故障很难被检测。其原因之一是在施加伪随机向量时,被测电路中的某些点常常固定在某个逻辑值(0 或 1),也就是说那些点的可控制性(controllability)极低。另一个原因是在施加伪随机向量时,电路中的某些点的逻辑状态(0 或 1)非常难以反映在测试响应中,也就是说那些点的可观察性(observability)极低。为了提高在施加伪随机向量时的可控制性和可观察性,可以在电路中加入一些测试点(test point),包括用于提高可控制性的控制点(control point)和用于提高可观察性的观察点(observe point)。

图 5-7 为插入控制点的一个例子。图 5-7(a)中的 C_1 和 C_2 为两个逻辑电路,其中 C_1 的

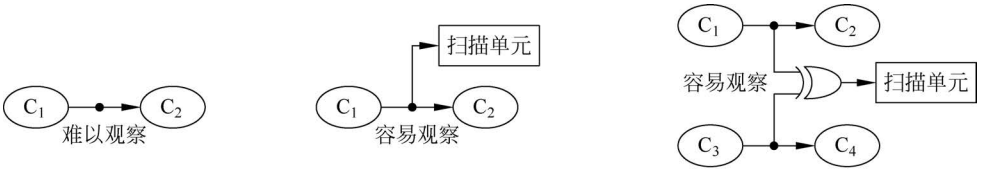
输出值在施加伪随机向量时基本为1，造成 C_2 中的一些故障不能被检测。图 5-7(b)是一个基于 AND 门的控制点，它利用 C_1 的输出值基本为 1 使得扫描单元的输出值进入 C_2。因为扫描单元输出值为 0 和 1 的可能性都很大，使得各种逻辑值得以输入 C_2，这有助于 C_2 中的故障检测。图 5-7(c)是一个基于 MUX 的控制点，它在 BIST 测试模式下(BIST_Mode＝1)把扫描单元的输出值直接送进 C_2，使得各种逻辑值得以输入 C_2，从而有助于 C_2 中的故障检测。



(a) 原电路　(b) 基于AND门的控制点　(c) 基于MUX的控制点

图 5-7　插入控制点的例子

图 5-8 为插入观察点的一个例子。图 5-8(a)中的 C_1 和 C_2 为两个逻辑电路，其中 C_1 的输出值很难被反映在测试响应中，造成 C_1 中的一些故障的效果观察不到而不能被检测。图 5-8(b)是一个专用观察点，它直接把 C_1 的输出接到一个观察用的扫描单元，从而有助于 C_1 中的故障检测。图 5-8(c)是一个共享观察点，C_3 和 C_4 也是两个逻辑电路，而且 C_3 的输出值也很难被反映在测试响应中。如图 5-8(c)所示，C_1 和 C_3 的输出通过 XOR 门接到一个观察用的扫描单元，这样做对 C_1 和 C_3 中的故障检测都有帮助，而且所需的硬件开销也较低。



(a) 原电路　(b) 专用观察点　(c) 共享观察点

图 5-8　插入观察点的例子

5.2.3　Re-Timing

如图 5-1 所示，在进行逻辑内建自测试的设计时，若把测试向量生成器(TPG)和测试响应分析器(TRA)放置在距离 BIST 对象电路较远的地方，在 TPG 和扫描链输入端之间以及在扫描链输出端和 TRA 之间有可能出现时钟偏移(clock skew)，造成测试向量和测试响应不能正确传输。为了避免这种问题并简化设计实现，可以在 TPG 和扫描链输入端之间以及在扫描链输出端和 TRA 之间插入 Re-Timing 电路，它包含一个下降沿触发 D 触发器和一个上升沿触发 D 触发器。图 5-9 的例子显示在扫描链的每一端使用两个触发器的 Re-Timing 逻辑，其中的 3 个时钟(CK_1，CK_2，CK_3)可属于同一个时钟树。

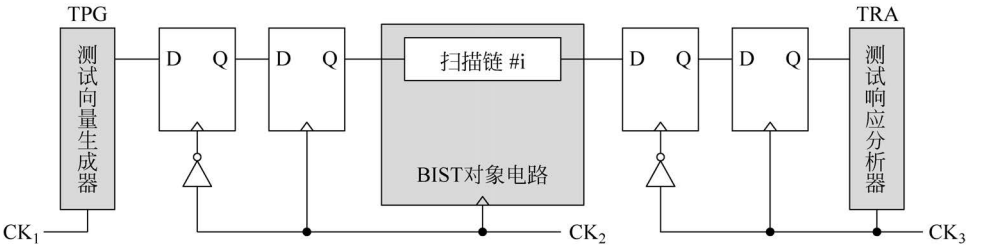
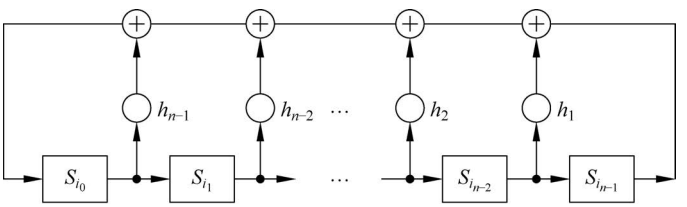


图 5-9　Re-Timing 的例子

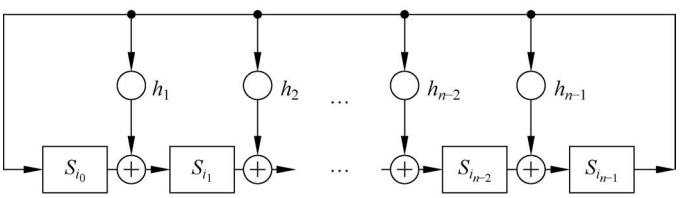
5.3　测试向量生成

逻辑内建自测试往往使用由线性反馈移位寄存器(Linear Feedback Shift Register，LFSR)所构成的测试向量生成器(TPG)，以产生穷举测试、伪穷举测试和伪随机测试所需的测试向量。穷举测试针对具有 n 个输入的被测组合电路生成所有可能的 2^n 个测试向量，这在 n 很大的情况下非常耗时。当具有 n 个输入的被测组合电路的每个输出最多取决于 w 个输入时，可以使用伪穷举测试，它生成 2^w 或 2^k-1 个测试向量($w<k<n$)。目前实际应用的逻辑内建自测试往往使用伪随机测试，它针对具有 n 个输入的被测组合电路生成远小于 2^n 的测试向量。这些测试向量虽然具有一定的随机性，但由于它们是用 LFSR 生成的，可重复再现，所以被称为伪随机测试向量。

图 5-10(a)所示为 n 段标准 LFSR 的构成。它由 n 个 D 触发器和一些 XOR 门组成。由于 XOR 门位于外部反馈路径上，因此标准 LFSR 也称为外部 XOR 型 LFSR。图 5-10(b)所示为 n 段模块化 LFSR，由于每个 XOR 门都位于两个相邻的 D 触发器之间，因此又称为内部 XOR 型 LFSR。模块化 LFSR 的工作速度快于相应的标准 LFSR，因为每级最多引入一个 XOR 门的时延。



(a) 标准LFSR



(b) 模块化LFSR

图 5-10　LFSR 的构成

一个 n 段 LFSR 的内部结构可以通过如下所示的一个 n 阶特征多项式 $f(x)$来描述。

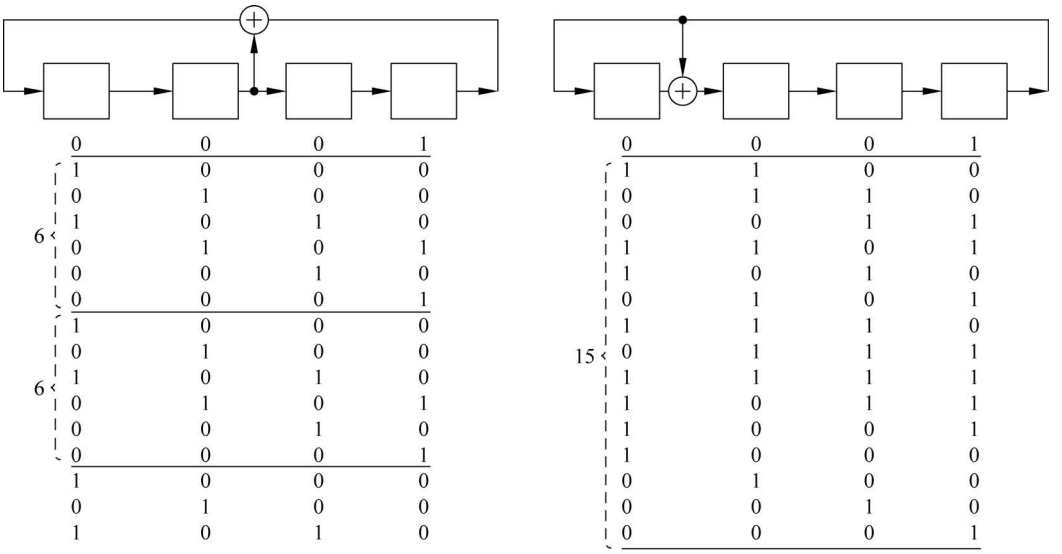
$$f(x)=1+h_1x+h_2x^2+\cdots+h_{n-1}x^{n-1}+x^n$$

其中的 h_i 可以是 1 或 0，取决于反馈路径的存在与否。假设 S_i 代表上述 n 段 LFSR 的初始状态 S_0 的第 i 次移位后的该 n 段 LFSR 的状态，而 $S_i(x)$是 S_i 的多项式表示，那么 $S_i(x)$可表述为如下所示的一个 $n-1$ 阶多项式：

$$S_i(x)=S_{i_0}+S_{i_1}x+S_{i_2}x^2+\cdots+S_{i_{n-2}}x^{n-2}+S_{i_{n-1}}x^{n-1}$$

如果 T 是使得 $f(x)$可以整除 $1+x^T$ 的最小正整数，则 T 称为上述 n 段 LFSR 的周期。如果 $T=2^n-1$，则该 n 段 LFSR 可生成长度最大的序列，因此被称为最大长度 LFSR。

图 5-11(a)和 5-11(b)所示的分别为一个 4 段标准 LFSR 和一个 4 段模块化 LFSR。这两个 LFSR 的特征多项式分别为 $f(x)=1+x^2+x^4$ 和 $f(x)=1+x+x^4$。若每个 LFSR 的初始状态 S_0 设置为{0001}(即 $S_0(x)=x^3$)时，它们将分别产生出如图 5-11(a)和 5-11(b)所示的测试向量序列。从图 5-11(a)和从图 5-11(b)中可以看出，它的每一个测试向量分别在其后第 6 和第 15 个时钟到来时重复出现，因此，图 5-11(a)和 5-11(b)所示的 4 段标准 LFSR 和 4 段模块化 LFSR 的周期分别为 6 和 15。这也意味着 $1+x^6$ 可以被 $1+x^2+x^4$ 整除，而 $1+x^{15}$ 可以被 $1+x+x^4$ 整除。



(a) 4段标准LFSR与其输出序列　　(b) 4段模块化LFSR与其输出序列

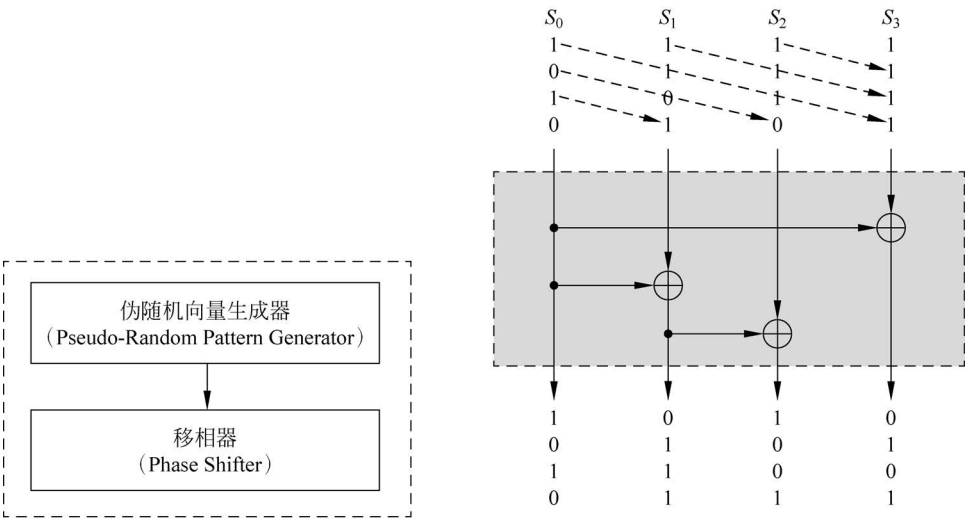
图 5-11　4 段 LFSR 的例子

伽罗瓦场 GF(2)上定义的一个 n 阶原始多项式 $p(x)$是指 $p(x)$可以整除 $1+x^T$ 但不能整除 $1+x^i$(i 为小于 $T(T=2^n-1)$的任何正整数)。原始多项式是不可约的。由于 $T=15=2^4-1$，图 5-11(b)所示的 4 段模块化 LFSR 的特征多项式 $f(x)=1+x+x^4$ 是原始多项式，因此这个 4 段模块化 LFSR 是最大长度 LFSR。

如上所述，一种牺牲电路故障覆盖率但可以减少测试向量数量的方法是使用伪随机向

量生成器(Pseudo-Random Pattern Generator,PRPG)来生成伪随机向量,这种方法既适用于时序电路又适用于组合电路。最大长度 LFSR 通常用于伪随机向量的生成,它的每个输出以 50%的概率产生 1 或 0。最大长度 LFSR 可以是标准、模块化或混合形式,非常容易实现。在实际应用当中,还可以在 PRPG 的输出端增加一些组合电路来改变 1 或 0 的出现概率。这种技术叫作加权向量生成,它有助于提高故障覆盖率。

值得注意的是,伪随机向量生成器的基本结构为移位寄存器,其相邻输出所生成的伪随机序列具有很强的相关性,这会导致故障覆盖率的低下。如图 5-12(a)所示,为了解决这个问题,可在伪随机向量生成器的输出端增加一个移相器(Phase Shifter,PS)。如图 5-12(b)所示,移相器可由若干 XOR 门组成,它的作用是减少由伪随机向量生成器的相邻输出所生成的伪随机序列的相关性,以提高故障覆盖率。



(a) 测试向量生成器的构成　　(b) 增加移相器后的输出序列

图 5-12　移相器的例子

5.4　测试响应分析

在具体实现逻辑内建自测试时,不可能通过将对应全部的伪随机向量的实际测试响应存储在芯片、电路板或系统的内部进行逐位比较以确定测试结果。为了减少测试响应的存储要求,必须先对测试响应进行一定的压缩,再对其进行分析以确定测试结果,这个过程统称为测试响应分析(test response analysis),由测试响应分析器(Test Response Analyzer,TRA)来实现。逻辑内建自测试中应用最广的测试响应分析技术是特征分析(signature analysis),它把全部的伪随机向量对应的实际测试响应压缩为一个特征(signature),再把得到的特征与事先存下的正常特征进行比较以确定测试结果。正常特征是通过在正常的(无故障)测试对象电路上对所有的伪随机向量进行逻辑模拟而获得的。

使用特征分析技术时，要尽量确保测试对象电路在有故障情况下和无故障情况下的特征不同。如果它们相同，则无法检测到故障。这种情况被称为错误屏蔽(error masking 或 aliasing)。另外，确保所有测试响应都不包含未确定值(X)也很重要，因为若未确定值直接或间接地传播到测试响应分析器(TRA)，将使需要考虑的正常特征的数量爆发性地增加，从而导致测试响应分析复杂化，这时就需要使用 5.2.1 节中介绍的技术对未确定值进行屏蔽。

5.4.1 串行特征分析

图 5-13 为 n 段单输入特征寄存器(Single-Input Signature Register，SISR)的一般构成，它在输入处使用附加的 XOR 门，以将 L 位输出序列 M 压缩到一个 n 段模块化 LFSR 中。

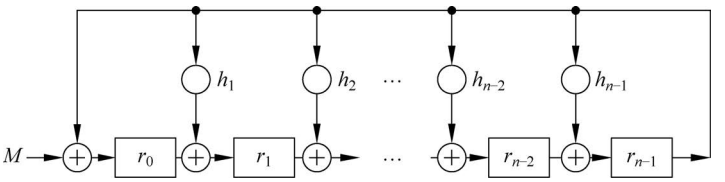


图 5-13 SISR 的构成

令 $M=\{m_0\ m_1\ m_2\cdots\ m_{L-1}\}$并定义 $M(x)$如下：

$$M(x)=m_0+m_1x+m_2x^2+\cdots+m_{L-1}x^{L-1}$$

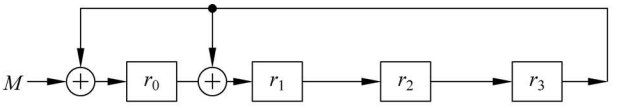
令 n 段模块化 LFSR 的特征多项式为 $f(x)$。SISR 的功能实际上是执行 $M(x)$除以 $f(x)$的多项式除法运算，也就是说：

$$M(x)=q(x)f(x)+r(x)$$

SISR 中的最终状态或特征是上述多项式除法运算的余式 $r(x)$。图 5-14(a)是一个 4 段 SISR，其特征多项式为 $f(x)=1+x+x^4$。假设 $M=\{10011011\}$是一个 8 位输出序列，它可以表示为 $M(x)=1+x^3+x^4+x^6+x^7$。使用多项式除法，可以得到 $q(x)=x^2+x^3$ 和 $r(x)=1+x^2+x^3$ 或 $R=\{1011\}$。如图 5-14(b)所示，$R=\{1011\}$等于 SISR 在初始状态(种子)为{0000}情况下输入 8 位输出序列 M 所得到的特征。若 M 是正常(无故障)电路的输出序列，则称 R 为正常特征。

如图 5-14(c)所示，假设故障 f_1 导致了错误的输出序列 $M'=\{11001011\}$或 $M'(x)=1+x+x^4+x^6+x^7$。使用多项式除法，可以得到 $q'(x)=x^2+x^3$ 和 $r'(x)=1+x+x^2$ 或 $R'=\{1110\}$。由于有故障 f_1 时的特征{1110}与无故障 f_1 时的正常特征{1011}不同，因此可以检测故障 f_1。又如图 5-14(d)所示，假设故障 f_2 导致了错误的输出序列 $M''=\{11001101\}$或 $M''(x)=1+x+x^4+x^5+x^7$。使用多项式除法，可以得到 $q''(x)=x+x^3$ 和 $r''(x)=1+x^2+x^3$ 或 $R''=\{1011\}$。由于有故障 f_2 时的特征{1011}与无故障 f_2 时的正常特征{1011}相同，因此无法检测故障 f_2。这种情况就是错误屏蔽(error masking 或 aliasing)。

通过无故障输出序列 M 和有故障输出序列 M'的错误序列 E 或错误多项式 $E(x)$，可以更好地理解 SISR 的错误屏蔽问题。定义 $E=M+M'$或 $E(x)=M(x)+M'(x)$。如果



(a) 4段SISR

M	r_0	r_1	r_2	r_3
1	0	0	0	0
1	1	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	0	1	1
0	0	0	0	1
0	1	1	0	0
1	0	1	1	0
R	1	0	1	1

(b) 正常特征

M'	r_0	r_1	r_2	r_3
1	0	0	0	0
1	1	0	0	0
0	1	1	0	0
1	0	1	1	0
0	1	0	1	1
0	1	0	0	1
1	1	0	0	0
1	1	1	0	0
R'	1	1	1	0

(c) 发生故障f_1时的特征

M''	r_0	r_1	r_2	r_3
1	0	0	0	0
0	1	0	0	0
1	0	1	0	0
1	1	0	1	0
0	1	1	0	1
0	1	0	1	0
1	0	1	0	1
1	0	1	1	0
R''	1	0	1	1

(d) 发生故障f_2时的特征

图 5-14　4 段 SISR 的例子

$E(x)$不能被 $f(x)$整除，则可以检测导致故障输出序列 M'的所有故障，否则将不会检测这些故障。在故障为 f_1 的情况下，$E=\{01010000\}=M+M'=\{10011011\}+\{11001011\}$或 $E(x)=x+x^3$。由于 $E(x)$不能被 $f(x)=1+x+x^4$ 所整除，因此可以检测故障 f_1。在故障为 f_2 的情况下，$E=\{01010110\}=M+M''=\{10011011\}+\{11001101\}$或 $E(x)=x+x^3+x^5+x^6$。由于 $E(x)$ 可以被 $f(x)=1+x+x^4$ 所整除，即 $E(x)=(x+x^2)f(x)$，因此不能检测故障 f_2。

假设 SISR 由 n 段组成，对于给定的 L 位输出序列 $L>n$，共有 2^{L-n} 种可能的方式来产生 n 位特征，而其中只有一个是正常特征。由于 L 位输出序列中总共有 2^L-1 个错误输出序列，因此使用 n 段 SISR 进行串行特征分析(Serial Signature Analysis，SSA)时发生错误屏蔽的概率为

$$P_{SSA}(n)=(2^{L-n}-1)/(2^L-1)$$

如果 $L\gg n$，则 $P_{SSA}(n)\approx 2^{-n}$ 当 $n=20$ 时，$P_{SSA}(n)<2^{-20}=0.0001\%$。这也就是说，SISR 的段数较多(>20)时，发生错误屏蔽的概率非常低，对测试质量影响不大。

5.4.2　并行特征分析

在逻辑内建自测试当中应用最广的特征分析技术是并行特征分析(Parallel Signature Analysis，PSA)，它利用多输入特征寄存器(Multi-Input Signature Register，MISR)压缩来自具有多个输出的测试对象电路的测试响应。图 5-15 显示 n 段 MISR 的一般构成，它使用 n 个 XOR 门将 n 个 L 位输出序列($M_0\sim M_{n-1}$)同时压缩到一个 n 段模块化 LFSR 中。

n 输入 MISR 可以表述为单输入 SISR，其有效输入序列 $M(x)$和有效错误多项式 $E(x)$可分别表示如下：

$$M(x)=M_0(x)+xM_1(x)+\cdots+x^{n-2}M_{n-2}(x)+x^{n-1}M_{n-1}(x)$$

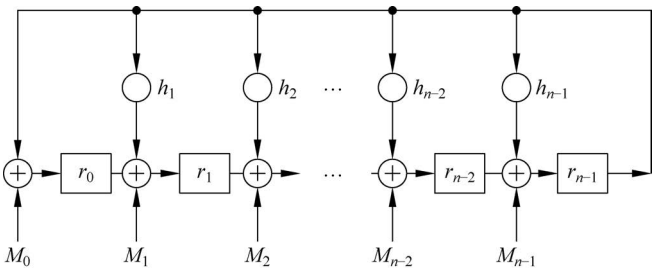
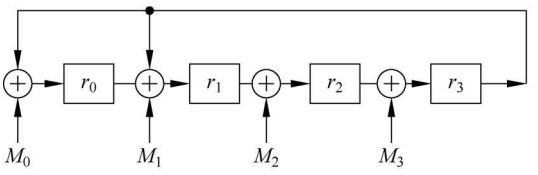


图 5-15 *n* 段 MISR 的一般构成

$$E(x)=E_0(x)+xE_1(x)+\cdots+x^{n-2}E_{n-2}(x)+x^{n-1}E_{n-1}(x)$$

图 5-16(a)是一个 4 段 MISR，它的特征多项式为 $f(x)=1+x+x^4$。假设这个 4 段 MISR 的压缩对象为 4 个 5 位输出序列：$M_0=\{10010\}$，$M_1=\{01010\}$，$M_2=\{11000\}$和 $M_3=\{10011\}$。根据这些信息，MISR 的特征 R 可以计算为{1011}。使用 $M(x)=M_0(x)+xM_1(x)+x^2M_2(x)+x^3M_3(x)$，可以得到 $M(x)=1+x^3+x^4+x^6+x^7$ 或 $M=\{10011011\}$。如图 5-16(b)所示，这与在如图 5-14(b)中的 4 段 SISR 的例子中的数据流 M 相同。



(a) 4段MISR

M_0	1	0	0	1	0			
M_1		0	1	0	1	0		
M_2			1	1	0	0	0	
M_3				1	0	0	1	1
M	1	0	0	1	1	0	1	1

(b) 正常特征

图 5-16 4 段 MISR 的例子

假设在 n 段 MISR 中有 m 个 L 位输出序列要压缩，其中 $L>n\geqslant m\geqslant 2$。并行特征分析的错误屏蔽概率为

$$P_{PSA}(n)=(2^{m(L-n)}-1)/(2^{mL}-1)$$

如果 $L\gg n$，$P_{PSA}(n)\approx 2^{-n}$。当 $n=20$ 时，$P_{PSA}(n)<2^{-20}=0.0001\%$。这表明当 $L\gg n$ 时，$P_{PSA}(n)$主要取决于 n。因此，增加 MISR 段数或使用段数相同但具有不同特征多项式 $f(x)$的 MISR 可以大幅降低错误屏蔽概率。

5.5 测试时序控制

逻辑内建自测试可以通过将 ATE 的大多数功能转移到被测电路上来降低测试成本，其最关键且最困难的部分是如何设计测试时序控制，因为它关系到时钟设计。下面简单介绍低速测试和实速测试所需的测试时序控制方式。

5.5.1　低速测试

低速测试(slow-speed test)是指测试向量的输入结束时刻到测试响应的捕获时刻之间的时间间隔大于被测电路的工作时钟周期,它用来检测时钟域之间和时钟域之内的结构故障(如固定故障和桥接故障)。低速测试的特点是在捕获针对每个测试向量的测试响应时,对每个时钟域仅需要施加一个捕获脉冲。

图 5-17 是低速测试的时序控制的例子,其中 CK 和 SE 分别是测试时钟和扫描使能信号。首先,把 SE 设置为 1 以进入移位窗口(shift window),并施加与扫描链长度(即构成扫描链的扫描单元的总数)相同数目的移位脉冲,完成测试向量的输入。随后,把 SE 设置为 0 以进入捕获窗口(capture window),并施加一个捕获脉冲(C),这样可以检测电路内的结构故障。如图 5-17 所示,在进行低速测试的时序控制设计时应该注意适当调整时延 d_1(最后一个移位脉冲 S_n 与捕获脉冲 C 之间的时间间隔)和 d_2(捕获脉冲 C 与下一个移位窗口的第一个移位脉冲 S_1 之间的时间间隔)以满足各个扫描触发器的时序要求。值得注意的是,d_1 和 d_2 都不需要太短,以便 SE 可以有充足的时间从 1 变化到 0(进入捕获窗口)和从 0 变化到 1(进入移位窗口)。也就是说,用于低速测试的 SE 可以是低速信号,所以比较便于在设计中实现。

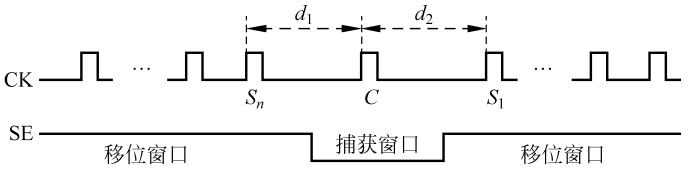


图 5-17　低速测试的时序控制

5.5.2　实速测试

实速测试(at-speed test)是指逻辑跳变产生的时刻到测试响应捕获的时刻之间的时间间隔与被测电路的工作时钟周期相同,它用来检测时延故障(如路径时延故障和跳变时延故障)。实速测试可以通过移位激发(Launch-on-Shift,LoS)或捕获激发(Launch-on-Capture,LoC)的方式来实现。

图 5-18 是基于移位激发的实速测试的时序控制的例子,其中 CK 和 SE 分别是测试时钟和扫描使能信号。首先,把 SE 设置为 1 以进入移位窗口(shift window),并施加移位脉冲。随后,把 SE 设置为 0 以进入捕获窗口(capture window),并在捕获窗口施加一个捕获脉冲 C,而且保证最后一个移位脉冲 S_n 的到达时刻和在捕获窗口施加的一个捕获脉冲 C 的到达时刻之间的时间间隔 d_1 与被测电路的工作时钟周期相同。如图 5-18 所示,若最后一个移位脉冲 S_n 和它之前的移位脉冲 S_{n-1} 使扫描单元 SC_1 输出不同的逻辑值,则扫描单元 SC_1 的输出端会在时刻 T_1(最后一个移位脉冲 S_n 的到达时刻)产生一个逻辑跳变。假设这个逻辑跳变可以沿路径 P(由一个或多个逻辑门构成)传播到另一个扫描单元 SC_2 的

输入端。由于扫描单元 SC_2 在时刻 T_2(捕获脉冲 C 的到达时刻)进行捕获,而且 T_1 和 T_2 的时间间隔 d_1 与被测电路的工作时钟周期相同,所以路径 P 上的时延故障可以被检测。值得注意的是,若被测电路的速度很快,则 d_1 非常短,因此 SE 必须高速地从 1 变化到 0(进入捕获窗口)。也就是说,用于移位激发(LoS)方式的 SE 是与时钟信号同等的高速信号,设计实现比较困难。但是,LoS 方式的故障覆盖率一般比较高。由于 LoS 方式利用移位窗口的最后两个移位脉冲造成同一个扫描单元的输出值的变化来激发逻辑跳变,因此移位激发也被称为偏载(skewed-load)。请注意,为了进入移位窗口,SE 只需低速地从 0 变化到 1。也就是说,尽管 d_1 往往需要很短,但是 d_2 不用太短,因此 SE 可以有充足的时间从 0 变化到 1(进入移位窗口)。

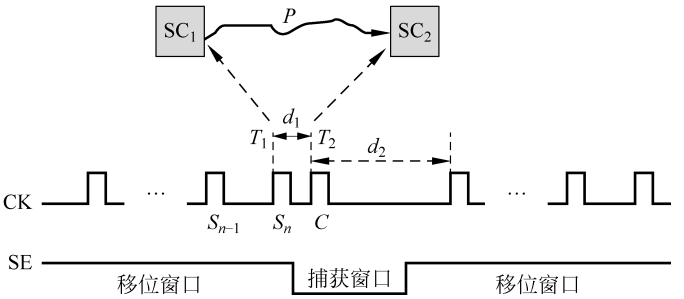


图 5-18 基于 LoS 的实速测试的时序控制

图 5-19 是基于捕获激发(LoC)的实速测试的时序控制的例子,其中 CK 和 SE 分别是测试时钟和扫描使能信号。首先,把 SE 设置为 1 以进入移位窗口(shift window),并施加移位脉冲。随后,把 SE 设置为 0 以进入捕获窗口(capture window),并在捕获窗口施加两个捕获脉冲 C_1 和 C_2,而且保证这两个捕获脉冲之间的时间间隔 d 与被测电路的工作时钟周期相同。如图 5-19 所示,若最后一个移位脉冲 S_n 和第 1 捕获脉冲 C_1 使扫描单元 SC_1 输出不同的逻辑值,则扫描单元 SC_1 的输出端会在时刻 T_1(捕获脉冲 C_1 的到达时刻)产生一个逻辑跳变。假设这个逻辑跳变可以沿路径 P(由一个或多个逻辑门构成)传播到另一个扫描单元 SC_2 的输入端。由于扫描单元 SC_2 在时刻 T_2(捕获脉冲 C_2 的到达时刻)进行捕获,而且 T_1 和 T_2 的时间间隔 d 与被测电路的工作时钟周期相同,所以路径 P 上的时延故

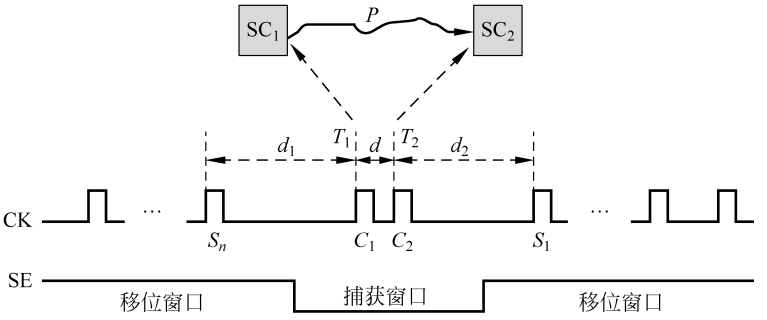


图 5-19 基于 LoC 的实速测试的时序控制

障可以被检测。值得注意的是，d_1 和 d_2 都不需太短，因此 SE 可以有充足的时间从 1 变化到 0(进入捕获窗口)和从 0 变化到 1(进入移位窗口)。也就是说，用于 LoC 方式的 SE 可以是低速信号，比较便于在设计中实现。但是，LoC 方式的故障覆盖率一般不如移位激发(LoS)。因为 LoC 方式在捕获窗口施加两个捕获脉冲，所以它也被称为双重捕获(double capture)。

5.6　实例介绍

本节简单地介绍一个利用商用软件工具(Tessent)进行逻辑内建自测试(Logic BIST)设计的实例，以便读者对本章的内容有一些具体的感知。这个实例是运用 tsdb 流程在 RTL 设计中插入 hybrid tklbist 测试模块。图 5-20 包含 Tessent 工具中插入测试模块的流程，其基本步骤如下。

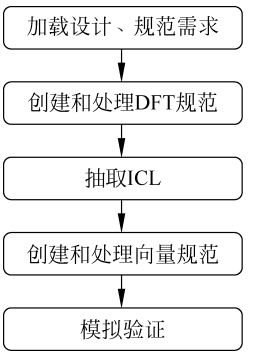


图 5-20　EDT/Tessent LogicBIST 模块插入的流程

A1. **准备好在 rtl1 阶段插入 mbist 电路之后的设计**：这些设计包括 rtl1 阶段生成的 Verilog 设计，icl、pdl、tcd 文件，以及在 rtl2 阶段加载进来的用于插入 hybrid tklbist 的测试模块。

A2. **定义 dft 信号**：该信号用于 edt 和 lbist 电路的控制。

A3. **创建 dft specification**：定义要插入的 occ、edt、lbist 测试模块的配置。

A4. **插入测试电路**：根据 dft specification 的定义插入电路并将修改后的设计信息以及插入模块的信息保存在 tsdb 中，保存下来的文件包括它们的 verilog、icl、pdl、tcd 文件。

A5. **抽取整个设计的连接关系**：保存为 icl 文件。

A6. **创建 pattern specification**：定义测试模块的测试向量的性质。

A7. **生成测试向量**：根据 pattern specification 的定义生成测试向量。

A8. **对测试向量进行仿真**：验证插入的测试电路以确定其功能是否正常。

在设计中插入 hybrid EDT/Tessent LogicBIST 测试模块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 设成在 RTL 级修改设计，插入测试电路的环境。

```
SETUP > set_context dft -rtl -design_id rtl2
```

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 piccpu -design_id rtl1，工具自动在 tsdb 目录下寻找并加载 rtl1 阶段修改后的设计信息。

```
SETUP > read_design piccpu - design_id rtl1
```

B5. 通过 read_cell_library 命令载入标准单元模拟库和存储器模拟库。

```
SETUP > read_cell_library ../lib/tessent/adk.tcelllib ../lib/tessent/picdram.atpglib
```

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

```
SETUP > set_current_design piccpu
```

B7. 通过 add_dft_signal 添加 dft 信号用于 memory bypass 控制和 clock 控制。

```
SETUP > add_dft_signal ltest_en memory_bypass_en tck_occ_en
SETUP > add_dft_signal edt_update - source_node {edt_update }
SETUP > add_dft_signal test_clock - source_node test_clock
SETUP > add_dft_signal edt_clock shift_capture_clock - create_from_other_signals
```

B8. 通过 add_dft_signal 添加 dft 信号用于 HTKLB 电路控制。

```
SETUP > add_dft_signals control_test_point_en observe_test_point_en x_bounding_en
SETUP > add_dft_signals int_ltest_en ext_ltest_en int_mode ext_mode
```

B9. 通过 set_dft_specification_requirements -logic_test on 定义逻辑测试的需求，后续通过 process_dft_specification 插入逻辑测试电路。

```
SETUP > set_dft_specification_requirements - logic_test on
```

B10. 添加 min occ 模块到当前设计中用于 lbist 测试。

```
SETUP > add_core_instances - instances [get_instances * _tessent_sib_sti_inst]
```

B11. 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
```

B12. 通过 set spec [create_dft_specification -sri_sib_list {occ edt lbist }] 创建 dft specification wrapper，并保存在变量中，后续可以修改自定义插入的 occ、edt、lbist 电路的配置。

```
ANALYSIS > set spec [create_dft_specification - sri_sib_list {occ edt lbist } ]
```

B13. 通过 read_config_data -in $spec -from_string 修改 dft specification，定义要插入设计中的 OCC 模块的配置。

```
ANALYSIS > read_config_data - in $ spec - from_string {
Occ {
      ijtag_host_interface: Sib(occ);
      capture_trigger: capture_en;
      static_clock_control: both;
      Controller(clk) {
```

```
            clock_intercept_node: clk;
        }
        Controller(ramclk) {
            clock_intercept_node: ramclk;
        }
    }
}
```

B14. 通过 read_config_data -in $spec -from_string 修改 dft specification，定义要插入设计中的 Hybrid TK/LBIST controller 和 NCPindexDecoder 的配置。

```
ANALYSIS > read_config_data - in $ spec - from_string {
LogicBist {
            ijtag_host_interface: Sib(lbist);
            Controller(c0) {
            burn_in : on ;
            pre_post_shift_dead_cycles : 8 ;
            SingleChainForDiagnosis { Present : on ; }
            ShiftCycles { max: 40; hardware_default: 33 ;}
            CaptureCycles { max: 4; }
            PatternCount { max: 10000; hardware_default : 1024 ; }
            WarmupPatternCount { max : 512;}
            ControllerChain {
                segment_per_instrument : off;
                present : on;
                clock : tck;
            }
            Connections {
               shift_clock_src: clk;
               scan_en_in : scan_en;
               controller_chain_enable:
               piccpu_rtl1_tessent_tdr_sri_ctrl_inst/HTKLB_CCM_EN ;
            }
        }
        NcpIndexDecoder{
            Ncp(clk_occ_ncp) {
                cycle(0): OCC(clk);
                cycle(1): OCC(clk);
            }
            Ncp(ramclk_occ_ncp) {
                cycle(0): OCC(ramclk);
                cycle(1): OCC(ramclk);
            }
            Ncp(ALL_occ_ncp) {
                cycle(0): OCC(clk) ,OCC(ramclk) ,piccpu_rtl1_tessent_sib_1 ;
            }
            Ncp(sti_occ_ncp) {
```

```
                cycle(0):  piccpu_rtl1_tessent_sib_1 ;
                cycle(1):  piccpu_rtl1_tessent_sib_1 ;
            }
        }
    }
}
```

B15. 通过 read_config_data-in $spec -from_string 修改 dft specification，定义要插入设计中的 EDT controller 的配置。

```
ANALYSIS> read_config_data -in $spec -from_string {
EDT {
      ijtag_host_interface : Sib(edt);
      Controller (c0) {
          Compactor { type : basic ; }
          longest_chain_range : 20,60 ;
          scan_chain_count : 10 ;
          input_channel_count : 1;
          output_channel_count : 1;
          ShiftPowerOptions {
              present : on ;
              min_switching_threshold_percentage : 15 ;
          }
          LogicBistOptions {
              misr_input_ratio : 1 ;
              chain_mask_register_ratio : 1 ;
              ShiftPowerOptions {
                  present : on ;
                  default_operation : disabled ;
                  SwitchingThresholdPercentage { min : 25 ; }
              }
          }
      }
    }
}
```

B16. 通过 process_dft_specification 按照定义在 dft specification 中的 occ、edt、lbist 的配置将 occ、edt、lbist 电路插入当前设计中。

```
ANALYSIS> process_dft_specification
```

B17. 通过 extract_icl 抽取 ijtag network，并生成修改设计之后的 icl 文件。

```
INSERTION> extract_icl
```

B18. 通过 create_pattern_specification 生成当前插入的测试模块(ijtag、edt、occ、lbist)的 pattern specification。

```
SETUP> create_pattern_specification
```

B19. 通过 process_pattern_specification 验证 pattern specificaiton 中的定义并生成当前测试模块(ijtag、edt、occ、lbist)的 Verilog 格式测试向量。

```
SETUP > process_pattern_specification
```

B20. 通过 set_simulation_library_sources 定义仿真单元库的存放位置。

```
SETUP > set_simulation_library_sources -v {../lib/verilog/adk.v} -y ../lib/verilog -extension v
```

B21. 通过 run_testbench_simulations 验证测试模块的测试向量是否正确。

```
SETUP > run_testbench_simulations
```

B22. 通过 run_synthesis -generate_script_only 创建综合的脚本用于后续的综合过程。

```
SETUP > run_synthesis -generate_script_only
SETUP > write_design_import_script  ../03.synthesis/piccpu.dc_shell_import_script - replace
```

B23. 退出 Tessent Shell。

```
SETUP > exit
```

5.7　本章小结

逻辑内建自测试通过对原有电路增加一些特殊设计,使其不借助 ATE 就可以自行对其主要部分进行自测试。逻辑内建自测试的优势在于不需要体大价高的 ATE 可有效降低测试开销,比较容易提供高质量测试所需要的高速的信号输入/输出和处理能力,比较容易产生和利用极为大量的测试激励来增加检测芯片内缺陷的可能性,而且在芯片装入系统之后亦可进行反复测试,从而提高系统在使用过程中的可靠性。

本章介绍了逻辑内建自测试的 4 个主要部分,即 BIST 对象电路、测试向量生成器(TPG)、测试响应分析器(TRA)和 BIST 控制器。BIST 对象电路需要对未确定值(X)进行屏蔽,而且还需插入测试点以提高故障覆盖率;测试向量生成器主要由 LFSR 生成的伪随机向量来作为测试向量;测试响应分析器主要利用 MISR 对测试响应序列进行压缩以获得特征;时序控制需根据测试对象要求和设计能力选择最适合的方式。

随着芯片越来越广泛地应用于航空航天、国防、汽车、银行、医疗保健、网络、电信行业等关键领域,出厂时的良品芯片由于老化而产生故障的问题已成为系统整体可靠性的重大威胁。目前,逻辑内建自测试在降低测试功耗,提高测试质量,改进故障诊断能力等方面的研发不断深化。逻辑内建自测试越来越超出单纯的芯片测试的范畴,成为保证系统可靠性的必不可少的关键技术手段。

5.8　习题

5.1　针对由 3 个三态门构成的三态总线设计一个独热(one-hot)解码器,以便在 Logic BIST 模式(BIST_Mode=1)下避免发生总线竞争,从而防止过大电流对电路造成损坏。

5.2 针对双向 I/O 端口，设计一个 X 屏蔽电路，使之在 Logic BIST 模式(BIST_Mode=1)下被强制设置为输入。

5.3 图 5-21 中的逻辑值为该 LSFR 的初始状态，请给出该 LSFR 的输出序列。

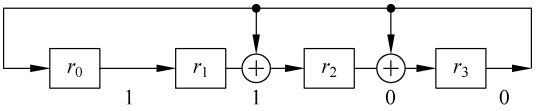


图 5-21 LSFR 的输出序列分析

5.4 图 5-14(a)是一个特征多项式为 $f(x)=1+x+x^4$ 的 4 段单输入特征寄存器(Single-Input Signature Register，SISR)。假设某故障 f 存在于被测电路当中，其输出序列为 $M_f=\{11111111\}$。试问该故障 f 是否可被检测？

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.