学　士　学　位　论　文

数字系统可测性设计方法的探讨

系（所）：　电子工程系

专　　业：　电子学与信息系统

姓　　名：　祝永明

学　　号：　9721004

指导教师：　唐长文

完成日期：　2001 年 6 月 8 日

摘要

本文主要探讨了用全扫描结构(FULL SCAN METHOD)来实现数字电路可测性设计(DESIGN FOR TEST)的原理与方法。其中涉及到扫描结构(SCAN)的算法依据、电路的基本结构、测试矢量的生成(ATPG)以及测试的时序等诸多问题。并结合最常用的综合工具 SYNOPSYS 中的 DFT COMPILER 部分，深入描述了为一数字电路芯片加入扫描部分和产生测试矢量集的具体流程。扫描结构对数字电路的结构有一些限制，为了避免违反这些限制，文中罗列了所谓的设计规范，并详细介绍了如何对违反这些设计规范的电路进行修改和处理的方法。

目 录

前言

随着微电子学的迅速发展,集成电路规模迅速膨胀,电路结构也越来越复杂,同时又受到芯片管脚的限制,大量故障变的不可测,因此过去由设计人员根据所完成的功能来设计电路，而测试人员根据已经设计或研制完成的系统和电路来制定测试的方案的传统做法已不适应实际生产的要求。功能设计人员在设计系统和电路的同时，必须考虑到测试的要求，即衡量一个系统和电路的标准不仅有实现功能的优劣，所用器件的多少，而且还要看所设计的电路是否可测，测试是否方便，测试码生成是否容易等问题。这就是所谓的可测性设计[5]。

可测性设计的优势	可测性设计的劣势
变不可测故障为可测故障	增加电路复杂度 I/O 管脚数量
测试矢量集生成时间少	影响面积和电路延迟
测试矢量少	引起功耗上升和成品率下降

表 1　可测性设计优、劣势的比较

实现可测性设计有许多种方法,本文讨论是最流行的有扫描方式电路设计(SCAN)。与其它方法相比，扫描结构的优势在于结构相对简单，而且实用效果相当不错，SYNOPSYS 中测试部分实现的便是扫描结构。扫描方式可将测试矢量集从输入端□移入□电路内部,同时可将电路内部的信号值□移出□来观察,从本质上提高了电路的可观察性和可控制性。当然扫描结构对电路的结构有一定的要求，所以在设计电路时必须遵循一定的设计规范。而对已有电路，如果结构上有问题，则必须针对问题对电路进行必要的修改。总之，可测性设计的要领便是在设计一个电路或系统的一开始便把测试问题考虑在内，依据一定的设计规范进行设计，这样最后在加入测试部分时才会水到渠成，避免不必要的返工。

下面第一、二章先介绍一下可测性设计的几个必要概念和算法基础，其后几章都是围绕 SYNOPSYS 软件中的全扫描结构部分具体展开。第三章介绍全扫描结构的具体电路和测试时序。第四章说明了加入扫描电路的流程和相关的命令。最后两章罗列了相应的设计规范，以及如何对违反这些规范的电路进行修改的具体方法。

第一章 可测性设计的概念及发展状况

1.1 可测性设计的度量

总的来说，一个电路是可测的，则意味着在预定的经费开支和一定的时间内可以产生一个测试矢量集，且可以时间予以评估和计算已经实际施加这些测试矢量，以便完成预定故障的检测和定位。

由此可见，一个电路的可测性问题应该包括两个方面[8]：

- 要容易由外部输入信号来控制电路中的各个节点的电平值，以便能够敏化故障和控制敏化通路上的各控制信号。这种特性反应用测试矢量来改变一个节点逻辑的难易程度，称为节点的**可控性**(CONTROLLABILITY)。
- 要容易建立故障的敏化通路，使内部故障能传播到外部输出端，以便能够从外部输出端口观察内部故障是否存在。这种特性反映从外部输出端观察内部故障的难易程度，称为**可观性**(OBSERVABILITY)。

一个电路中各个节点的可控性和可观性是不同的，为了科学的评估他们，必须对他们进行数量化，这就是所谓可控性和可观性的量度。目前出现的分析可控性和可观性的量度基准和方法各不相同。

电路节点的可测性应该同这个节点的可控性和可观性均有关系，即它应是可控性和可观性的函数。在各个不同的可测性量度方法中，这个函数关系也是不同的。常用的可测性量度有史蒂文森可测性分析、高尔德斯泰可测性分析以及基于电路结构的可测性分析(SCOAP)等[1]。

1.2 可测性设计的分类

可测性设计的方法主要可以分成两大类：一类是专项设计(Ad Hoc Design),即按功能基本要求设计电路，采取一些比较简单易行的措施,使他们的可测性得到提高；另一类是结构设计(Structured Design),它是根据可测性设计的一般原则和基本模式来进行电路的功能设计。显然，前者不能根本解决测试问题的，尤其对于时序电路，本来测试就十分困难，稍做改进后仍会相当困难。所以时序电路的设计中，一般都使用结构设计,扫描结构便是一种最常见的结构设计。

无论是专项设计还是结构设计，他们的基本设计思想是一致的。首先，把电路分块，因为据统计，电路的测试矢量数与电路的输入端数的三次方成正比[8]。另一个设计思想是要提高电路的可观性和可控性，因为他们和电路的可测性是直接相关的。

1.3 测试矢量与测试码自动生成(ATPG)

在对一个电路产品进行测试时，要用特定的一系列输入信号以一定的顺序加在被测电路的输入端，观察其输出端的输出结果。如果输出结果和正常电路的输出结果相同，认为被测电路是合格的；反之，则不合格。

测试码生成的方法有许多种，如穷举测试码(EXHAUSTIVE TEST PATTERN)、伪随机数测试码(PSEUDO-RANDOM PATTERN)、测试生成算法(TEST GENRATION ALGORITHM)和故障模拟(FAULT SIMULATION)等[1]。其中，根据逻辑电本身的结构用算法自动生成测试码，称为测试码自动生成(AUTOMATIC TEST GENRATION ALGORITHM，简称 ATPG)。在 SYNOPSYS 软件中，有专门的测试编译器(TEST COMPILER)，可以针对电路的扫描结构(SCAN)，自动产生测试矢量，并且对测试矢量集进行、压缩及优化。

1.4 故障覆盖率

所谓故障覆盖率是指一个测试集已测故障数占所有可测故障数的百分比。

$$\text{故障覆盖率} = \frac{\text{已测故障数}}{\text{故障总数} - \text{不可测故障数}} \times 100\%$$

对于一个复杂电路，要找到其完备的测试集是很不容易的。一般说来，要确定一个故障覆盖率指标，比如 95%，达到这个指标就认为是完成了测试码的生成。随着数字系统规模的迅速扩大，测试生成变的越来越困难。因此人们逐渐把注意力转向电路设计方面。在设计过程中充分考虑可测性，使得电路的测试码容易形成，或者在电路内部增加测试功能。

第二章 D 算法的原理及实现方法

2.1 故障模型

在讨论具体的算法之前，我们先来定义一下故障的类型。一个电路的物理故障是各式各样的，为了便于研究，按照其特点和影响将其分类，称为故障模型(FAULT MODEL)。

电路故障分为参数故障和逻辑故障。参数故障指电路参数变化引起的故障，不属于本篇的讨论范围。逻辑故障又可分为“永久故障”、“间歇故障”和“瞬态故障”。我们讨论的是永久性故障中的“固定型故障”(STUCK FAULT)。所谓固定型故障，是指某个信号线的值固定为某一电平值(0 或 1)。值为 1 的故障称为“固定型 1 故障”(STUCK AT 1 FAULT)；值为 0 的故障称为“固定型 0 故障”(STUCK AT 0 FAULT)。图 2.1 是一个输出端有固定型 0 故障(STUCK AT 0 FAULT)的与门模型。

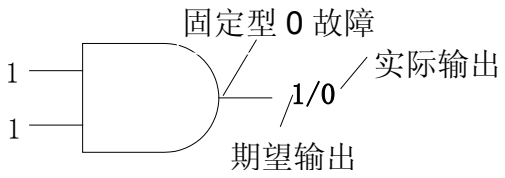


图 2.1 输出端有固定型 0 故障

固定型故障模型在实际应用中用的最普遍，因为电路中的元件损坏、连线的开路和相当一部分的短路故障都可以用固定型故障模型比较准确的描述出来。而且它的描述比较简单，因此处理故障也比较方便。如果一个电路中只存在一个固定型故障，称之为单固定型故障；如果有两个或两个以上，则称为多固定型故障。一般，只要不是毁灭性的事故所导致的故障，都认为是单固定型故障。

2.2 敏化路径法

对指定故障点的测试码的生成算法的基本思想是通过输入端测试矢量把故障传播到输出端，使得正常电路的输出与故障输出结果不同。这种基于故障传播路径生成测试码的方法，称为敏化路径法。

为了把故障传播到外部输出，要有两个条件：

- 输入测试矢量能够使得故障点在正常情况下与故障情况下的状态值不同；
- 有至少一个外部输出端的正常值与有故障是的值不同。为了能做到这一点，要求从故障点出发能找到一条或几条路径到达输出端，使得该路径上每个节点的正常值与有故障时的值不同，这条路径称为敏化路径。

下面通过一个例子，说明如何通过敏化路径法求得测试矢量。

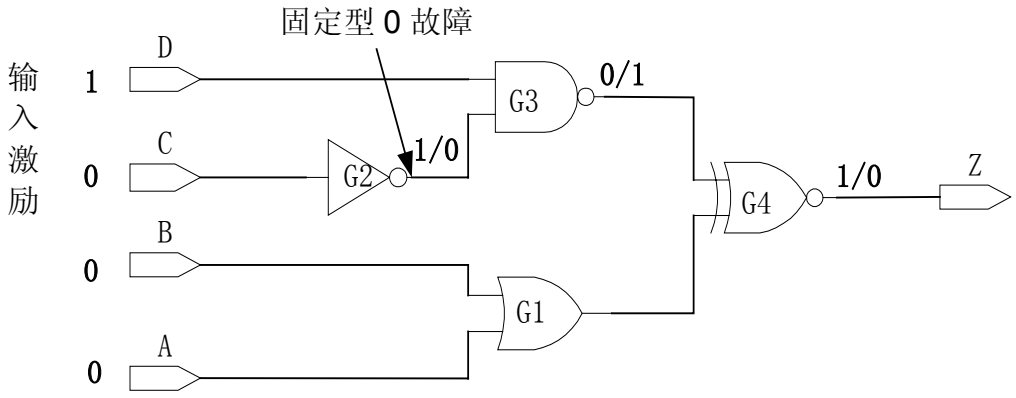


图 2.2　G2 输出端有固定型 0 故障

图 2.1 中，非门 G2 的输出端有固定 0 故障。为了检测出这处故障，必须控制激励信号使得 G2 的输出为 1(与故障值相反)。所以，需在 C 端输入 0。

为了保证故障的影响能传播到输出端 Z 口，必须控制电路其它输入的值，使得最后的输出 Z 只决定于 G2 的输出值。分两步做：

- 使得输入端 D 为 1，则 G3 的输出之决定于 G2 的输出；
- 使得输入端 A 和 B 为 0，则 G4 的输出也只决定于 G2 的输出。

于是，得到了针对 G2 输出端固定型 0 故障的测试矢量 T={(0，0，0，1)；(1)}，其故障输出为 0。

2.3 D 算法简介

单路敏化法方法简单，缺点是不能保证对任一非冗余故障都能找到测试矢量。典型的情况是故障处于再会聚路径(RECONVERGENT PATH)。而 D 算法则可以做到。

D 算法是一种多路径敏化法，是在立方体理论的基础上实现路径敏化的。它

的基本思路与单路径敏化法相同。也是把故障传播到输出端去，同时确定输入矢量和其它信号值，以确保个信号的一致性。其区别在于 D 算法采用立方体运算，并考虑到多路径的情况。D 算法的具体介绍，可参照附录的书目[5]。

D 算法规律性很强，而且对固定性错误的检测十分有效，所以它在计算机可操作性和解决问题的通用性上都得到了广泛的应用。但是在具体应用中，由于其进行敏化通路的选择时随意性太大，需要作大量的返回操作，导致计算工作量太大，尤其是对大型的组合电路计算时间很长，以至于很难付诸实际应用。

D 算法是针对组合电路而言的，对于时序电路，可以用类似于 D 算法的方法实现。应用这种类似 D 算法来检测时序电路中的故障，通常需要以下两个步骤：

- 一个或一个以上周期来激励这个故障；
- 一个或一个以上周期来传播这个故障，直到能从外部探测到。

也就是说，为了探测一个固定性故障，我们需要一个序列的测试矢量和一个以上的时钟周期才能做到。显然，这对大型复杂的时序电路来说，无论在测试矢量集的大小和测试时间上都是难以忍受的。

注意：

在 SYNOPSYS 中，采用扫描结构(SCAN DESIGN)，等于把大型的时序电路分割为小型的组合电路来测试，避免了时序电路测试的测试序列问题，也不需要返回操作。从而使测试码自动生成(ATPG)限制在一个可以接受的计算量和时间之内。这也是扫描结构可以大大缩短测试时间的一个重要原因。

第三章 SYNOPSYS 中的测试部分及扫描结构

3.1 SYNOPSYS 中扫描电路的基本结构

DFT COMPILER 支持以下四种扫描结构[4]：

- 多路选择型(MULTIPLEXED FLIP FLOP);
- 时钟型(CLOCKED SCAN)；
- 电平敏感型(LSSD)；
- 辅助时钟电平敏感型(AUXILIARY CLOCK LSSD)。

后三种结构都是在第一种结构的基础上进行的改进，目的是减少由于扫描电路的误触发导致的竞争和冒险的问题。比如第三种 LSSD 结构便是在触发时，使用时钟电平而不是常用的时钟边沿，从而有效的避免了上述问题。但这种改进也会大大的增加电路的复杂度，所以最常用的还是第一种多路选择型的扫描结构。

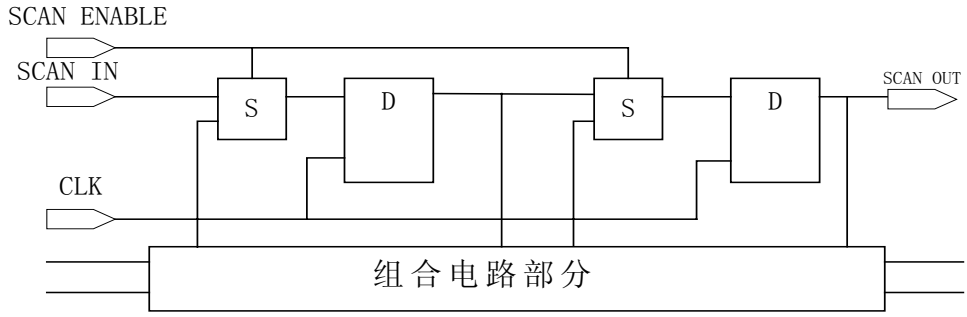


图 3.1 典型的扫描方式电路

所谓多路选择结构，是使用一个多路信号输入来实现串行移位功能。在正常工作模式下，扫描使能端(也就是输入信号多路选择器的控制端)选择系统输入信号。当扫描工作模式时，扫描使能端选择扫描输入信号。扫描输入信号来自扫描输入端口，或者上一个扫描器件的扫描输出端口。图 3.1 是典型扫描方式电路的大致框图。图中电路组合部分被单列开来，而由 D 触发器组成的时序电路部分成为了扫描链。由图中可以看出，当控制输入 SCAN ENABLE=0 时，电路动作按通常方式进行，执行电路的功能。当控制信号 SCAN ENABLE=1 时，各触发器形成一个移位寄存器，为扫描方式。在扫描方式下，各触发器可设定任意值，也可方便的从扫描输出端(scan output)观察其输出值。

对于多路选择结构，需要增加以下端口：

- 扫描输入(SCAN IN)
- 扫描使能(SCAN ENABLE)
- 扫描输出(SCAN OUTPUT)

其中扫描输入端必须单独列出，而扫描输入和扫描输出都可以和其它管脚公用，所以扫描结构所增加的管脚数也十分有限。

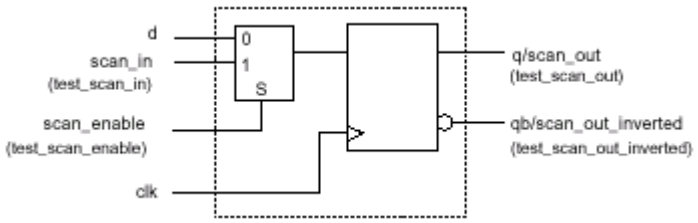


图 3.2　SYNOPSYS 中 D 触发器的扫描结构

大多数器件库都支持多路选择结构的 D 触发器，但大都不支持锁存器模型(LATCH)。图 3.2 为转换后默认的多路选择结构 D 触发器结构，框图中的 S 是二路选择器。下面是其真值表。

d	scan_in	scan_enable	clk	q	qb	Mode
0	X	0	↑	0	1	Functional
1	X	0	↑	1	0	Functional
X	0	1	↑	0	1	Scan
X	1	1	↑	1	0	Scan
X	X	X	0/1	q	qb	Either

↑ = 时钟上升沿
X = 无关

表 3.1　二路选择器的真值表

多路选择结构的特征：

- 由于在电路中增加了多路选择器，会导致电路正常工作时，额外延时的增加。
- 对于芯片面积的增加较少。多路选择型的 D 触发器比一般的 D 触发器增加 15 到 30%的芯片面积。
- 最少只需要增加一个 SCAN ENABLE 管脚。扫描输入、输出管脚可以与

原有功能管脚共用。

- 一般适用于时钟边沿触发的电路。

3.2 扫描测试的协议及时序

将测试矢量集应用到具有扫描电路结构的电路时，需要使用自动测试设备(ATE)。一般每一个测试矢量的应用需要以下五个阶段：

- 扫描输入阶段(SCAN IN PHASE);
- 并行测量阶段(PARALLEL MEASURE)；
- 并行取值阶段(PARALLEL CAPTURE)；
- 链首输出阶段(FIRST SCAN OUT)；
- 扫描输出阶段(SCAN OUT PHASE)。

其中第一和最后两个阶段为串行工作方式，当中三个为平行工作方式。

五个阶段的时钟构成如下图：

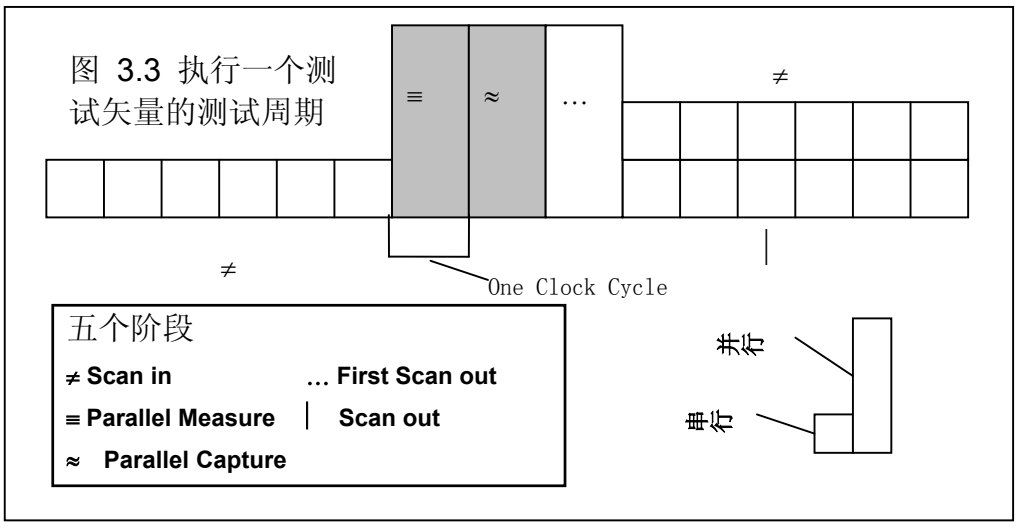


图 3.3 执行一个测试矢量的测试周期

3.3 各阶段各管脚具体的时序图[3](SE为1时有效)：

看以下时序图，必须注意以下三点：1)**SE是否有效**。当SE有效时电路中的时序部分成为扫描链，组合部分不起作用；2)**是否有时钟信号**(CLK)。在并行测量阶段没有时钟信号，因为在此阶段需要取得组合逻辑的输出，而如果有时钟信号便会破坏信号的稳定；3)**是否有探测脉冲**(STROBE)。不该有测试脉冲的阶段千万不能误加，否则所得的测试结果序列便会出错。

- 扫描输入与扫描输出阶段(SCAN IN & SCAN OUT PHASE)

有上面的时钟构成图可以看出，其实扫描输入和输出两个阶段是同时进行的。如图3.4，在这个阶段中，扫描使能信号(SE)一直有效。扫描电路将测试矢量移入电路内部，同时将上一个测试周期的测试结果移出，在扫描输出端(SCAN OUT)进行探测。

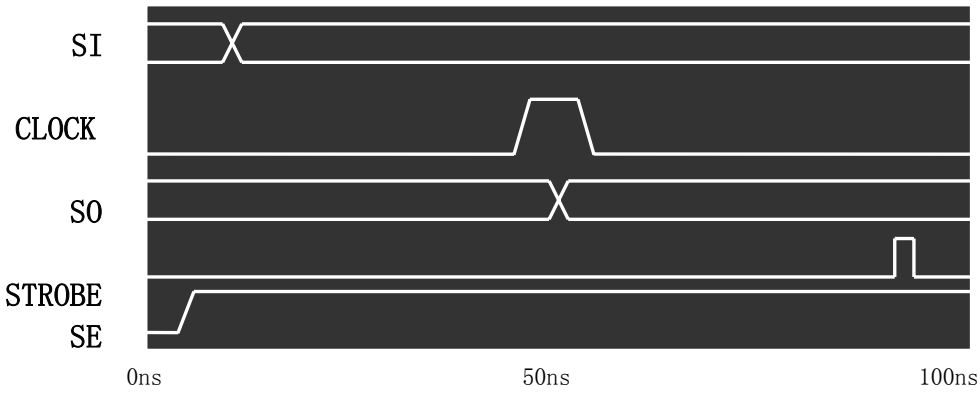


图 3.4 扫描输入与扫描输出阶段时序

- 并行测量阶段(PARALLEL MEASURE)

  如图3.5，在这一阶段时，被测器件(CUT)处于正常工作状态，扫描使能信号(SE)无效。此阶段没有时钟信号，测试矢量已经被移入芯片内部，CUT处于一个已知状态。当状态稳定后，会对并行输出进行检测。

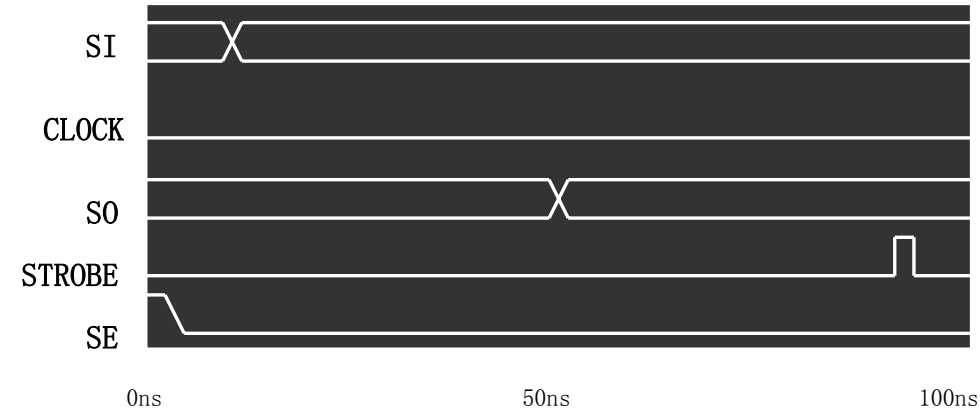


图 3.5　并行测量阶段时序

- 并行取值阶段(PARALLEL CAPTURE)

如图3.6，在这一阶段，被测电路仍处于正常工作状态，扫描使能信号(SE)仍然无效。此阶段中，时钟信号被激活一次。这样，扫描链中的虚拟输出端(Virtual PO)便可以得到测试的结果。这些数据准备被下一个扫描输出阶段移位输出到扫描输出端，提供给ATE检测。

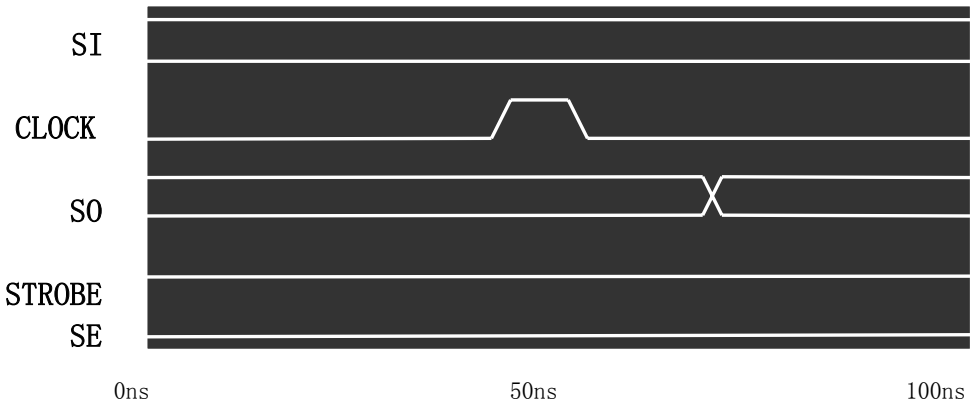


图 3.6　并行取值阶段时序

- 链首输出阶段(FIRST SCAN OUT)

  如图3.4，但这个周期中没有时钟，ATE在扫描输出端进行一次检测。增加这个周期主要是防止扫描链第一位结果的丢失。接下来便是扫描输出阶段，内部的测试结果移出，同时将下一个测试矢量移入芯片内。

在以上的时序图中，我们把测试的时钟周期默认为 100ns。其实，各个厂商生产的 ATE 所用的测试时钟周期并不相同，他们共同的特性是必须留出足够的时间让测试结果稳定，并被检测到。所以，我们在设置测试的“时间包”(TIME PACKAGE)时，必须注意到这一点。

了解测试的时序十分重要，因为SYNOPSYS软件在检测电路是否符合扫描结构的设计规范时，会进行检测的仿真，而其所用的时序与以上的真实时序是相同的。设计者可以跟据时序仿真时产生的错误，结合时序图进行分析，从而找出错误的根源。

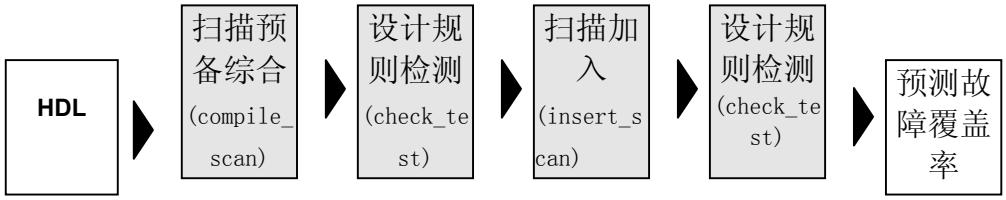
第四章 扫描电路的加入及 ATPG 的产生过程

任何成功的扫描测试应用必须包括以下两个步骤：

- 在电路中插入适当的扫描链(SCAN CHAINS);
- 产生正确的测试矢量集(ATPG)并达到满意的故障覆盖率。

用DFT COMPILER在电路中加入扫描电路的大致流程[2]如下：

图 4.1 模块扫描加入的典型流程



注：括号内为所使用的命令

下面，以对未布线源文件加入扫描电路为标准流程作详细的说明，而后对已布线和已有扫描路径的电路进行说明，最后说明对 CHECK_TEST 命令的应用和 ATPG 的原理。

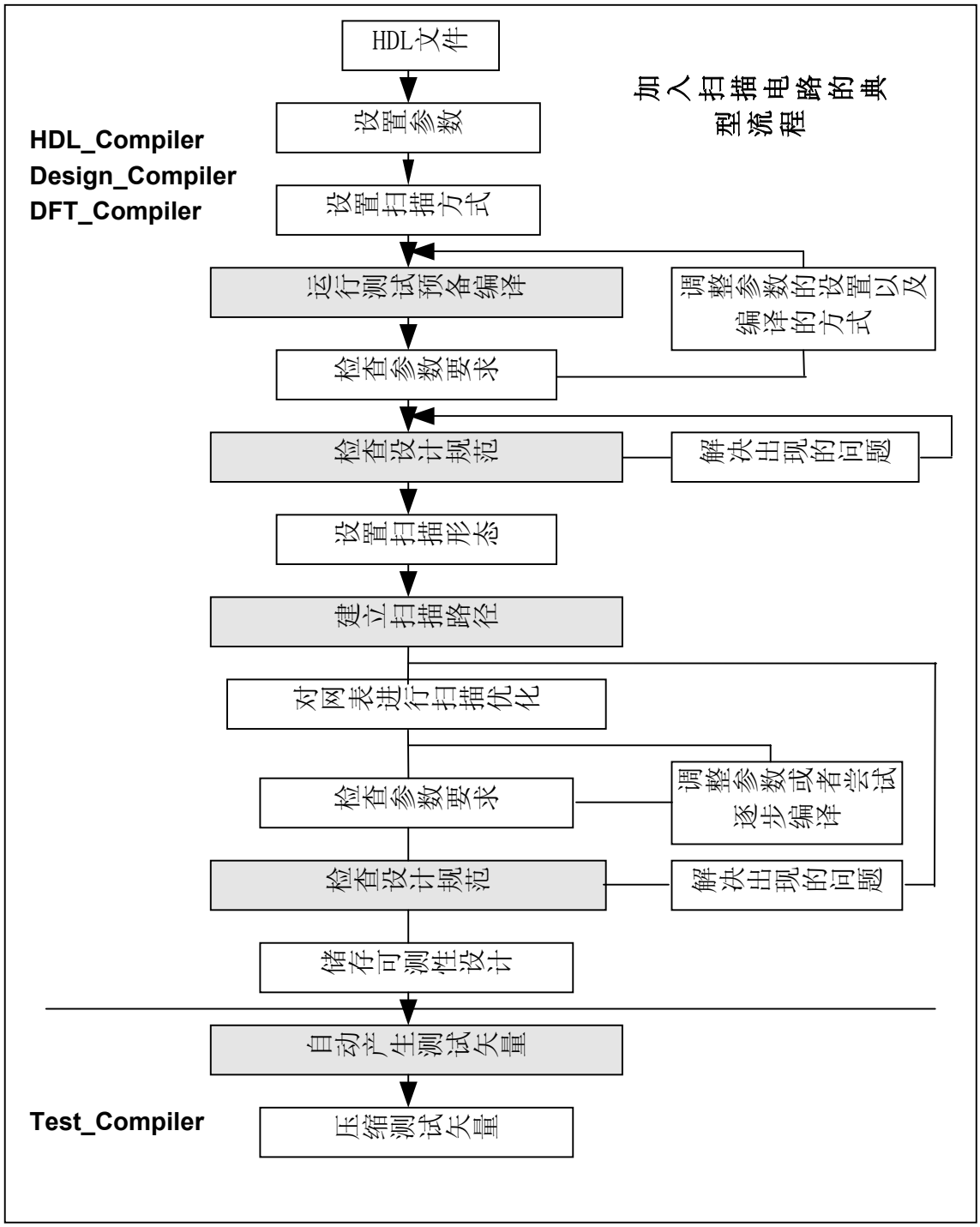
4.1 对未布线源文件(Unmapped Design)加入扫描电路的流程

以下说明的是使用 Test Compiler 以及 DFT Compiler 和 HDL Compiler 来进行扫描可测性设计的典型流程。在设计流程的最后，测试编译器会一组故障覆盖率很高的测试矢量集，并可运用到实际的测试仪器中去。根据这个流程，可以使一个硬件描述级的电路(HDL)成为一个对于内部扫描结构完整优化的电路。

以下流程可分成三大部分：

- 对源文件进行综合及测试预备编译(TEST READY COMPILE)；
- 建立扫描通路(SCAN CHAINS)；
- 产生测试矢量集。

下面是流程图，流程图的后面对流程中所用的命令及需要注意的地方进行逐步说明。



4.2 对源文件进行综合及测试预备编译(TEST READY COMPILE)。

需顺序执行以下步骤：

1) 使用下列命令读入一个源程序：

```
dc_shell> read -format verilog design_name.v
dc_shell> read -format vhdl design_name.vhdl
```

如果你想在扫描使能端(SCAN ENABLE)加缓冲门，则需在你的源文件中加入一个未连接的扫描使能端。虽然 DFT Compiler 会给没有扫描使能端的电路自动加上一个，但不会给其加缓冲门。

1) 使用下列命令设置目标库(TARGET LIBRARY)以及相关库(LINK LIBRARY)。

```
dc_shell> target_library = asic_vendor.db
dc_shell> link_library = { * asic_vendor.db }
```

2) 将设计中的各个模块联系起来：

```
dc_shell> link
```

SYNOPSYS 支持单模块的重复调用，但必须在编译前执行 **uniquify** 命令。

3) 设置电路参数。比如，要设计一个面积在 1000 门以内，时钟周期为 20ns 的电路，则需进行以下设置：

```
dc_shell> max_area 1000
dc_shell> create_clock clock_port -period 20 \
                -waveform {10,15}
```

其中，第二条设置时钟的命令中，大括号内的之分别指时钟上升沿和下降沿的时间点。

4) 设置扫描模式(SCAN METHODOLOGY)。全扫描模式(FULL SCAN)是DFT COMPILER的默认扫描模式，如果要设置为部分扫描方式(PARTIAL SCAN)，用以下命令：

```
dc_shell> set_scan_configuration -methodology \
```

partial_scan

5) 设置扫描方式(SCAN STYLE)。如果你没有在.synopsys_dc.setup中设置default_scan_style参数，你可以用以下命令将扫描方式设为multiplexed flip flop型。

dc_shell> **test_default_scan_style = **

multiplexed_flip_flop

你可以用set_scan_configuration-style命令达到同样的效果。

6) 设置测试参数。这些是针对测试的时钟参数，可以包括在.synopsys_dc.setup文件中。另外，还需设置你的测试时钟周期。

dc_shell> **create_test_clock –p 100 –w {45 55} CLK**

dc_shell> **test_default_delay = *nn***

dc_shell> **test_default_bidir_delay = *nn***

dc_shell> **test_default_strobe = 95**

dc_shell> **test_default_strobe_width = 0**

这些就是所谓的时钟包，可以包含在一个SCRIPT当中。

7) 根据你设置的参数，综合你的设计，并由你的目标库(TARGET LIBRARY)产生网表。

dc_shell> **compile –scan**

这条命令可以在综合时，对于面积和速度进行优化，而且去除多余的逻辑(REDUNDANT LOGIC)。使用scan选项，综合时电路中的触发器都会被具有扫描结构的等效触发器所代替。

8) 存储你的设计。

dc_shell> **write -format db -out design_test_ready.db**

注意：

不要将你的设计存储为ASCII格式的文档。因为，DFT COMPILER对做过测试预备综合的电路做有特殊的标记，它们会在ASCII格式下被丢失。

9) 根据你的设计形式，检查你的设计规范(DRC)。使用命令：

dc_shell> **check_test**

通过这条命令，DFT COMPILER会检查并描述你设计中关于测试方面的各种问题。当你解决了所有的错误和警告后，你才能进行下一步操作。不能解决的警告问题，往往会导致故障覆盖率的急剧下降。

CHECK_TEST命令的详解请看本章的第四节。如何避免产生错误和警告，可参见下一章"扫描结构对电路的限制"。

4.3 建立扫描通路(SCAN CHAINS)

真正在电路中插入扫描通路。

1) 确认扫描的使能端。

dc_shell> **set_scan_signal test_scan_enable **
-port scan_enable_port

2) 用insert_scan命令建立扫描通路，DFT COMPILER会认为需要对电路中的触发器进行扫描结构的改造。而你已经使用了测试预备综合(COMPILE SCAN)，电路内部已存在扫描结构，所以不需要在建立扫描通路时重复加入扫描结构，用下面的命令防止DFT COMPILER报错：

dc_shell> **set_scan_configuration -replace false**

3) 在电路中插入扫描通路结构，并将电路中的时序器件用具有扫描功能的器件代替。

dc_shell> **insert_scan**

4) 检查电路是否和你所做的参数要求相符，可用以下命令产生相关的报告：

dc_shell> **report_constraint -all_violators**

5) 再次检查设计规范(DRC)。

dc_shell> **check_test**

因为插入扫描通路本身对电路结构产生了影响，可能会产生新的错误和警告，所以必须再次运行CHECK_TEST命令。解决了所用的错误和警告后，才能进行下一步的操作。

6) 现在可以用report_test命令，来观察扫描路径的各种特性。比如，你要知道扫描路径的结构和具体构成，可用以下命令：

dc_shell> **report_test -scan_path**

7) 存储你完整的设计，并进行下一步测试矢量集的产生：

dc_shell> **write -format db -hierarchy -out design.db**

4.4 产生测试矢量集

1) 产生一系列完整的测试矢量集，用以下命令：

dc_shell> **create_test_patterns **

-compaction_effort high

一般来说，如果在先前执行CHECK_TEST命令时，将所有的错误及警告全部都改正过来，那么所产生的测试矢量集一般都能达到97%以上的故障覆盖率。选用compaction_effort_high选项，使产生的测试矢量集的测试矢量数目最少，同时也可减少测试时间。

2) 测试矢量集可被存为你需要的格式。下面的命令，将其存为Waveform Generation Language(WGL)格式：

dc_shell> **write_test -out test_vectors -format WGL**

至此为止，我们已经给电路加上了完整的扫描测试电路，并且产生了完备的测试矢量集。当然，这是比较理想情况，而实际情况往往糟糕的多。纠正各种错误和警告，达到满意的覆盖率，往往要进行许多的返工，花费设计者许多的时间。最好的方法，是在设计时就遵照一定的设计规范，不要违反比如第五章

中提到的限制，这样产生扫描电路的时间将大大减少。这也是所谓可测性设计的初衷。

DFT_COMPILER支持从上至下和从下至上两种加入扫描结构的方法。上面的流程是指从上至下的方法。从下至上的流程，则是对每个模块进行与上面相似的流程，然后在顶层再执行一遍。当然，在顶层要对底层原有的扫描路径进行辨识及重组。但是，测试问题从根本上说是一个系统整体的问题。所以，一些模块在单独编译时没有问题，但加入整个系统后，便会出现可测性方面的问题。因此，如果可能的话，尽量使用从上至下的方法，可以避免不必要的返工。

4.5 对已编译电路及已有扫描路径的处理

对已编译电路及已有扫描路径的电路加入扫描结构的流程与上面的流程大部分也都相同，只有几处不同需要注意。

- 对已编译电路。

  不需要执行 COMPILE_SCAN 命令，因为对已编译电路不可能在编译时将触发器模型替代掉，这个过程只有在执行 INSERT_SCAN 命令时进行。因为在编译时没有考虑到扫描型触发器所要占用的面积和时延问题，所以最后产生的电路可能在时序上产生问题，需进行进一步的仿真。

- 对已有扫描路径的电路。

  在建立扫描结构前，要使用以下命令：

  dc_shell> **set_scan_configuration -existing_scan true**

  这条命令可以使 DFT COMPILER 在加入扫描结构时，意识到已有扫描路径存在。同时，在下一步还需用以下命令标识出扫描路径特有的端口，才能使已存在的扫描电路正常工作。

  dc_shell> **set_signal_type test_scan_in test_scan_in_port**
  dc_shell> **set_signal_type test_scan_out **
  test_scan_out_port
  dc_shell> **set_signal_type test_scan_enable **

test_scan_enable_port

当然，如果读入的是.db 文件，这些信息有可能已经包含在源文件中了，可以用 report_test_configuration 命令查询。

就如前一节所说的，测试总体来说是一个系统问题，而且测试电路本身也会对整个电路的面积和时序产生影响，所以最好是在设计的一开始就考虑测试的问题，并在最后顶层再加入扫描结构。

4.6 CHECK_TEST 命令的应用

CHECK_TEST 是 DFT COMPILER 中最关键的命令之一，它的主要功能就是检查电路的设计规范。因为扫描结构有其严谨的设计规范，只有完全符合了这些规范才能使最后产生的电路达到满意的故障覆盖率。所以，怎样用好 CHECK_TEST 命令来检测设计的错误，并通过修改设计来消灭错误，便成了加入扫描结构流程中的关键。下面就介绍一下 CHECK_TEST 的执行过程。

- 执行 CHECK_TEST 所要做的准备。

  在执行 CHECK_TEST 命令前，必须先定义标准的时间包，即标准流程中第一部分的第 7 步。还需引用测试时钟(TEST CLOCK)。如果是已存在扫描路径的电路，必须把扫描路径标识出来。

- CHECK_TEST 所执行的功能。

  CHECK_TEST 主要分以下 4 个阶段执行其功能：

  - ✓ 模型检测(MODELING CHECK)
  - ✓ 结构检测(TOPOLOGICAL CHECK)
  - ✓ 引用协议(PROTOCOL INFERENCE)
  - ✓ 协议仿真(PROTOCOL SIMULATION)

- 模型检测(MODELING CHECK)。

  执行模型检测时，DFT COMPILER 对电路中的模块和器件逐一检测是否符合设计规范，如果不满足，会产生以下几类错误信息：

- ✓ 黑匣子(BLACK BOX)。指库中没有的器件。
- ✓ 不支持器件(UNSUPPORTED CELLS)。虽然库中有这个器件，没有它的扫描结构的等效器件。
- ✓ 锁存器(LATCH)。一般不支持锁存器的扫描结构，所以会被认为是不支持器件。

- 结构检测(TOPOLOGICAL CHECK)。

  结构检测主要检测以下两方面的错误。
  - ✓ 管脚和缓冲门。如果发现诸如三态门等无法控制的管脚，便会出现警告。
  - ✓ 组合电路的反馈。软件会自动打破其回路。

- 引用协议(PROTOCOL INFERENCE)。

  所谓协议(PROTOCOL)，其实是一个.tdf 文件，在其中定义了一些有关测试环境和电路特性的参数。在执行 CHECK_TEST 命令时，DFT COMPILER 必须引用协议，并进行协议仿真。在协议仿真阶段，DFT COMPILER 会模仿真正检测的五个阶段，即扫描输入阶段(SCAN IN PHASE)、并行测量阶段(PARALLEL MEASURE)、并行取值阶段(PARALLEL CAPTURE)、链首输出阶段(FIRST SCAN OUT)和扫描输出阶段(SCAN OUT PHASE)。在这个过程中，检测电路中不符合设计规范的部分。它所能检测错误有：
  - ✓ 不可控的时钟端和复位端。由于在这些端口前有逻辑门或者异步电路，使其在扫描输入阶段无法同步控制，导致错误。
  - ✓ 管脚值被锁定。该管脚无法控制。
  - ✓ 时钟信号被用作输入信号。会在并行取值阶段阶段出现时序混乱，从而导致错误。
  - ✓ 扫描链中触发器使用的触发沿不同。在并行取值阶段时，前后触发器取值时间不同，导致所得数据错误。

最后，CHECK_TEST 命令会列出设计中违反设计规范的总数以及可用作扫描元件的时序元件数。要进行下一步的操作，设计者必须将这些错误一一改正，否则会导致故障覆盖率的大幅下降。下面是一个 DRC 报告的例子。

例 4.1：

```
*******************************************************
Test Design Rule Violation Summary
Total violations: 3
*******************************************************
TOPOLOGY VIOLATIONS
     1 Combinational feedback loop violation (TEST-117)
     1 Unreachable circuitry violation (TEST-118)
CAPTURE VIOLATIONS
     1 Illegal path violation (TEST-478)
     1 Cell cannot capture violation (TEST-310)
*******************************************************
Sequential Cell Summary
     1 out of 3 sequential cells have violations
*******************************************************
SEQUENTIAL CELLS WITH VIOLATIONS
     * 1 cell has parallel capture violations
       o1_reg
SEQUENTIAL CELLS WITHOUT VIOLATIONS
     * 2 cells are valid scan
```

在上例中，结构检测发现了电路中的组合电路回路和不可控点的错误，在协议仿真的取值阶段发现了非法路径错误，最后在元件总结中，说明了 3 个元件中有一个有错误，另两个可以作为有效的扫描路径。可以通过错误后面的编号(如 TEST-117)和 man 命令，查找有关错误的信息。关于如何改正这些错误，可参见第六章。

4.7 ATPG 的产生及原理

当一个设计被成功的插入了扫描结构并有了固定的 I/O 管脚，便可对其运行 create_test_patterns 命令，自动产生测试矢量集(ATPGs)。下面是执行 create_test_patterns 产生的结果。

例 4.2：

```
                    Non-collapsed     Collapsed

No. of detected faults      388           245
```

```
No. of abandoned faults     0          0
No. of tied faults          0          0
No. of redundant faults     0          0
No. of untested faults      106           68
Total no. of faults         494           313
Fault coverage              78.54         78.27
No. of test patterns        27
Test Generation Time (CPU)  5.17 sec
Start compaction...
...End compaction
No. of compacted patterns   21
```

在 DFT COMPILER 中，定义了以下五种故障:

- 可测故障(DETECTED)。可以被一个测试矢量监控的故障。
- 摈弃故障(ABANDONED)。TEST COMPILER 达到其 CPU 极限仍无法找到一个合适的测试矢量来检测的故障。
- 固定故障(TIED)。端口被锁定为固定值，无法检测。
- 沉余故障(REDUNDANT)。沉余型逻辑，无法检测。
- 不可测故障(UNTESTABLE)。找不到有效的测试矢量来检测此故障。

这里所指的故障覆盖率由以下公式决定：

$$\frac{\text{Detected Faults}}{\text{Detected Faults + Abandoned Faults + Untestable Faults}} \times 100\%$$

自动产生测试矢量集需要进行下列三个步骤。

- 产生随机测试矢量，测试那些比较容易被检测到的故障。
- 产生固定测试矢量，测试某些特有的故障。
- 对测试矢量集进行压缩，达到满意的故障覆盖率。

在上例中，故障覆盖率为 78%，压缩前的测试矢量数为 27 个，压缩后为 21 个。所谓的 collapsed，指的是某些故障可以用同一个测试矢量来检测，也就是所有的等效故障的集合。在 TEST COMPILER 中，产生 ATPG 时使用的是零沿时模型。所以，ATPG 必须是在门级电路，也就是在部局布线之前产生并仿真。

注意：

在 TEST COMPILER 中的 ATPG 命令只能支持 50，000 门以下电路的测试矢量集自动生成，如果电路规模更大则需求助于 TetraMAX 等专业的 ATPG 软件。

第五章 扫描结构对电路的限制

扫描结构的测试是一种结构性测试(Structured Design)，它是通过扫描链(SCAN CHAIN)来增加内部各个节点的可控性(CONTROLLABILITY)和可观性(OBSERVABILITY)，所以对电路的时钟、复位端以及影响到扫描链移位的结构

有十分严格的限制。只有当电路的结构符合这些限制时，在加入扫描结构后，才能达到满意的故障覆盖率，否则故障覆盖率会大大降低。由此可见，设计电路时遵循一定的设计规范是十分必要的。下面，便将所需的设计规范逐条列出并解释其原理。

本篇主要讨论的是多路选择(Multiplexed Flip Flop)的全扫描结构。在下面的讨论中，提到的测试状态逻辑(TEST MODE LOGIC)是指任何为了使电路能够适应扫描测试结构(SCAN TEST)而对其做的修改部分。

5.1 电路中需增加的管脚

对每一个设计，扫描结构需要一个测试使能端(TEST ENABLE)和一个测试时钟输入(一般与系统时钟共用)。另外，对于大多数实际电路，都需要一个测试状态输入端(TEST MODE)，用来控制测试状态逻辑。对于电路内每一个扫描链，都需要有一个扫描输入端(SDI)和一个扫描输出端(SDO)。这两类端口通常可以与其他功能的输入、输出端口共用。

5.2 避免使用 LATCH

锁存器(LATCH)是时序器件之一，但在多路选择扫描模式下没有它的等效扫描模型。这是因为锁存器的扫描模型无法在实际电路中工作。如果不做任何修改，Test Compiler 会把锁存器归类为黑匣子(BLACK BOX)。这也就是说 Test Compiler 不知如何处理这类器件，而且带有锁存器的通路都将变得不可测。

5.3 避免产生组合电路的反馈

组合电路的回路会导致无法同步控制的内部状态。为了使 ATPG 算法正常的工作，必须把回路打破。软件可以自动把回路打破，但这不是最佳的方法，因为你不知道软件会在何处将回路割断。最好的方法是使用“set test isolate”命令，人工观察所有的回路，并在设计者人为地最佳点将回路打破。在回路中的双向端口会被认为是输出端口。

5.4 对时钟信号及复位信号的限制

Test Compiler 需要对电路内部的状态有完全的控制。所以，它必须对电路内

部所有的时钟和使能端都有完整的控制。当任何器件的使能端或时钟端无法被控制时，它便会被排除在扫描链之外，从而使故障覆盖率大大降低。以下是几种无法受控的时钟端和使能端：

- 多重时钟沿。是指在一个电路中既有上升沿触发的触发器，又有下降沿触发的触发器。这两种触发器如果出现在同一条扫描链中，会导致数据的误传。最简单的解决方法是避免在同一条扫描路径中同时使用两种触发器。
- 时钟端或使能端被用作数据输入端。这会导致在扫描过程中，无效数据的传播。可以通过测试状态逻辑将其改正。
- 非同步复位端(Asynchronous Reset)。如果在复位端前有逻辑门(GATED RESET)，那么必须通过使用测试状态逻辑将它们屏蔽。这样也会导致故障覆盖率的下降，因为复位端将无法检测到。有时，可以通过使用手工定制的起始协议(INITIALIZATION TEST PROTOCOL)来控制复位端。
- 时钟前有逻辑门(GATED CLOCK)。这种情况往往不会对故障覆盖率产生大的影响，但有可能产生竞争与冒险的问题。为了避免逻辑门的影响，可以用“set signal type test scan enable CLK ENABLE”命令。其中，CLK ENABLE 指的是控制逻辑门的信号。这会产生很高的故障覆盖率，但需要人工找出所有的有问题的时钟端。
- 时钟前有时序逻辑。必须用测试状态逻辑来旁路这些时钟端。这会使故障覆盖率下降。
- 多重时钟。Test Compiler 有时可以支持多重时钟。在使用“check test”命令时，Test Compiler 会试图将不同时钟区域的触发器归入不同的扫描链中去。但这样产生的扫描链并不是最优的。一个更好的方法是使用测试状态逻辑，使整个电路在扫描测试时使用同一个时钟。

5.5 对三态总线的限制

内部三态总线往往会导致内部总线冲突，从而使扫描结构变的不可能实施。TEST COMPILER 不会自动的对三态总线进行处理，需要人工确定在扫描时不会出现冲突。这所要花费很多的人力，有时甚至是不可能的。你可以插入电路把三态总线屏蔽掉，但这会导致故障覆盖率的下降，有时也会使 ATPG 出现错误。

所以，如果要应用扫描型测试结构，最好不要在设计中加入内部三态总线。

对于双向门(BIDIRECTONAL PORTS)，TEST COMPILER 也认为其是三态门的一种。在测试时，你必须通过命令或增加电路，将其设为一个方向的输入或输出门。另外，为了在并行取值阶段(PARALLEL CAPTURE)能够正确的取得数据，设计者需对 test_default_bidir_delay 参数进行合适的设置。一般来说，这个参数应该比测试时钟的有效边沿大，才能防止由于双向门方向的改变引起的数据冲突。

以上就是所有的设计规范。如果能严格按照以上的规范设计一个数字系统，那么最后加入扫描结构后一定能达到一个满意的故障覆盖率。但是有时由于电路的要求，不得不违反以上的规范，而且这在实际的电路中往往是不可避免的。如果出现了这种情况，则必须通过对电路的修改，绕过这些限制，从而到达满意的覆盖率。在上面的论述中也提到一些修改的方法，下一章将对如何修改有问题的电路进行详细的论述。

第六章 对有问题电路进行的修改

如果已有电路不符和设计规范就必须对其进行修改。跟据对 check_test 命令产生的 DRC 报告的分析，可以找出错误的出处，并跟据错误的缘由对电路进行相应的修改。对错误的修改有手动修改和自动修改两种。下面就分别介绍一下。

6.1 手动修改

手动修改适用于任何一种错误，当然这比较耗费时间于精力。手动修改一般共有的有以下几个步骤：

- 在电路中增加一个 TEST MODE 管脚；
- 用 set 命令将寄存器和双向门设为可测试状态；
- 如果需要的话，修改 HDL 源文件加入测试状态逻辑；
- 运行 CHECK_TEST 前，将 TEST MODE 管脚设为有效值(set_test_hold)。

下面跟据上一章罗列的设计规范，说明具体的修改方法[4]。

- 寄存器(LATCHES)。

  如果电路中存在寄存器，可以用以下命令将其设为透明工作模式，这样测试时便不会影响到数据的移位。

  dc_shell> **set_scan_FALSE {cell_list} transparent**

  其中 cell_list 指的是所需修改属性的寄存器的器件列表。即使锁存器已被设为透明状态，使用寄存器仍会使整个的故障覆盖率下降。这是因为锁存器使能端口被设定位恒定值，所以无法在扫描过程中被测试。

- 双向门(BIDIRECTIONALS)。

  用“set_scan_configuration bidi mode -output/input”命令将双向门设为单向输出或输入门。

- 组合电路反馈(Combinational Feedback Loops)

  图 6.1 中有一反馈回路，它会导致电路内部的状态不稳定，而使测试覆盖率降低。可以用以下几种方法将其改正。

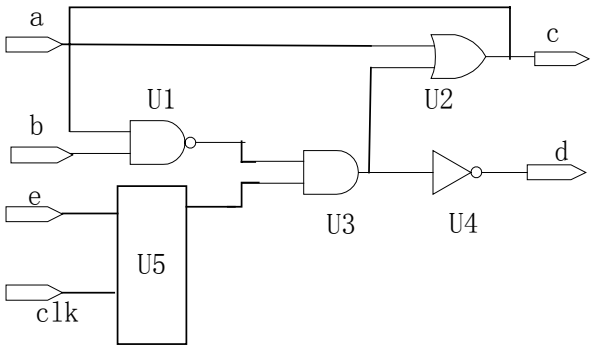


图 6.1 带有反馈回路的组合电路

1.用 set_hold 命令将 b 端设为固定值 0，则回路被切断。

2.可以通过修改协议(PROTOCOL)，用初始向量将时序部分 U5 的输出固定为 0，这样通过 U3 将反馈回路阻断。(具体做法参见后面关于协议的应用)

3.可以用set_test_isolate命令将反馈回路中的任意一点阻断。

set_test_isolate的作用是将电路中某点的值在测试时设为未知，这样在测试阶段便不会向该点取值。该命令会对故障覆盖率产生影响，因为它并没有解决问题，只是将问题隐瞒了起来。

以上三种方法都会对故障覆盖率产生影响，可以跟据客户的要求及结果的对比进行选择使用[3]。

- 不可控的时钟端和复位端(Uncontrollable Clock And Reset)。

这其中包括非同步复位端(Asynchronous Reset)、时钟前有逻辑门(GATED CLOCK)、时钟前有时序逻辑等所有不可控的时钟端和复位端。基本思路是增加一部分逻辑电路和一个扫描工作模式使能端(TM)，当扫描工作模式使能端值为真时，电路进入扫描工作模式。这种模式下，可以将有问题的部分或端口旁路掉，使扫描正常进行。这种方法可以用在几乎所有的问题上，而且也是在实际设计中用的最广泛的方法。但这需要比较多的人工干预，所以需要较多的时间。图6.2和6.3分别对不可控时钟和复位端进行了修改。

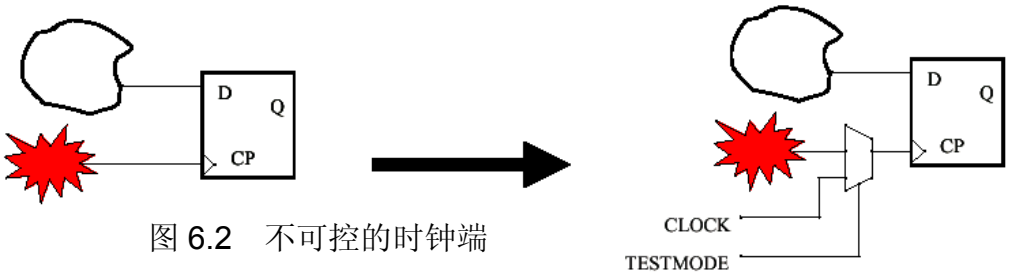


图 6.2　不可控的时钟端

上图中在不可控时钟前加了一个二路选择器，所以当TESTMODE有效时，便可将原时钟旁路掉，改由系统的测试时钟CLOCK来控制。

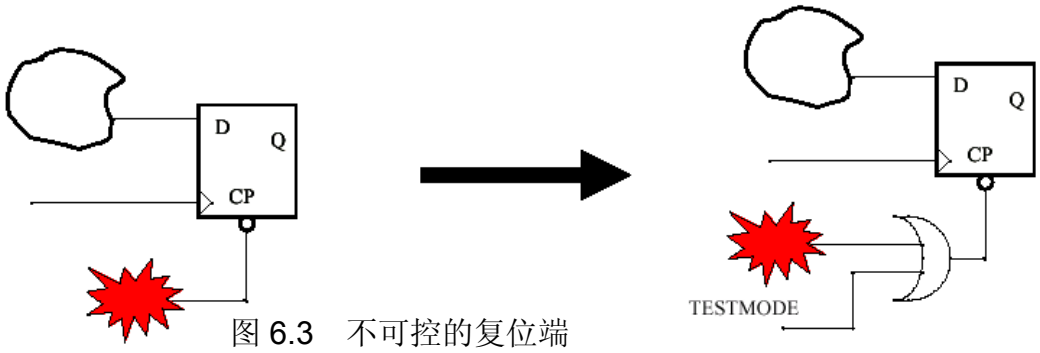


图 6.3　不可控的复位端

上图中在不可控的复位端前加了一个或门，当TESTMODE有效时，原来的复位信号被或门屏蔽掉，不会再对时序器件产生影响。

注意：

进行以上修改只能在HDL源文件中进行[6]。实行了上述修改后，在实行测试前，必须执行**set_test_hold 1 TESTMODE**命令才能使所加入的部分产生效果。

- 多重时钟沿

如果在同一电路中，既用到时钟的上升沿又用到下降沿，那么在CHECK_TEST时便会产生警告。解决的方法是应用多重通过ATPG(Multi Pass)。在多重通过的ATPG过程中，必须设置两次时钟波形，并进行两次测试矢量集的自动生成(ATPG)，DFT COMPILER会自动把两次的结果结合起来，达到满意的故障覆盖率。下面时进行多重通过ATPG的SCRIPT。

例6.1

```
……
multi_pass_test_generation = true          // 设置多重通过ATPG属性 //
create_test_clock –period 100   \                    // 设置第一个时钟 //
 -waveform {45,55} clk                    // 上升沿为45，下降沿为55 //
 check_test
create_test_patterns –out pass1             // 第一次产生测试矢量集 //

create_test_clock –period 100 \                    // 设置第二个时钟 //
 -waveform {55,45} clk                    // 上升沿为55，下降沿为45 //
check_test
create_test_patterns –out pass2             // 第二次产生测试矢量集 //
multi_pass_test_generation = false          // 取消多重通过ATPG属性 //
……
```

在第二次测试矢量集时，由于有了第一次的基础，产生的故障覆盖率可以达到令人满意的程度。

- 避免竞争与冒险

如果一个电路既用到外部时钟，又有内部时钟时，便有可能在扫描测试时产生竞争与冒险。解决方法是手工将不同的时钟区域(CLOCK DOMAIN)划分到不同的扫描链中。下面是一个例子。

例6.2

```
……
create_clock –name clock1 –p 100 \          //定义第一个时钟区域//
 find{port,"clk"}
create_clock –name clock2 –p 100 \          //定义第二个时钟区域//
 find{port,"clk_inst/DIV_CLK"}

set_scan_path path1 all_registers \   //设定第一个时钟区域的扫描链//
 {-edge_triggered –clock clock1}   \
        –complete true
```

set_scan_path　path2　　**//**设定剩下时钟区域的扫描链**//**

……

另外，可以用set_scan_path命令将一条较长的扫描链划分开来，因为扫描链太长会使扫描输入、输出的周期增多，从而增加了测试的时间。

- 三态门和三态总线

对于三态输出门，DFT COMPILER规定必须在输出管脚加一个上拉电阻，才能使三态门成为可测。对于三态总线，只能将其屏蔽掉，这样会导致故障覆盖率大幅下降。所以，对于三态总线最好采用别的测试方法。

- 用初始向量对电路进行初始化。

假设下图中的输出端RESET输出到其它时序元件的复位端，因为其是异步复位端，所以必须在测试时将其设为固定值。这可以通过对POR端设为固定值1来实现，也可用初始向量，即通过对协议(PROTOCOL)的修改来达到目的。

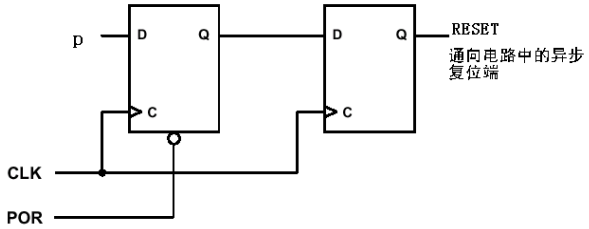


图 6.4　带有异步复位端的电路

正如前面所说的，协议(PROTOCOL)其实是一个.tdf文件，在其中定义了有关测试环境和电路特性的参数，还有测试的初始向量。在执行CHECK_TEST命令时，DFT COMPILER必须引用协议，并进行协议仿真。初始向量定义在PROTOCOL的FOREACH_PROGRAM()函数中，只需在这个函数中加入下面两条SET命令，便可将RESET初始为1。

例6.3

```
.......
protocol_start() {
```

```
foreach_program() {
        vector() {
                set(P' CLK' "1' C");
        }
        vector() {
                set(P' CLK' "1' C");
        }
.....
```

两个初始向量

其中，C 指的是时钟信号。在进行下一次 CHECK_TEST 时，DFT COMPILER 便会引用新的协议，自动将 RESET 置为 1。利用初始向量可以对其它需要进行预置数才能进入扫描测试状态的电路进行预置操作。

6.2 自动修改(AUTOFIX)

DFT COMPILER 独有其自动修复功能(AUTOFIX)。它的适用范围为：

- 不可控的时钟端；
- 异步的复位端。

AUTOFIX 会自动对你指定的管脚进行修改，所做的修改与上一节中，对不可控的时钟端和复位所做的修改相同。AUTOFIX 功能在默认时是关闭着的，要使用它必须用相应的命令将其打开。用其进行设计的流程，与前面所说的基本相似，只是许多具体命令都有不同。一个较典型的 SCRIPT 见下。

例 6.4¨

```
check_dft                          //检测设计规范相当于原来的 check_test//
set_dft_configuration -order {autofix}      //指定使用 AUTOFIX 功能//
set_autofix_clock CLK U1   //指定修改的管脚¨模块 U1 中的 CLK 管脚//
preview_dft/               //预览修改的结果相当于原来的 preview_scan//
insert_dft                 //插入修改的部分相当与原来的 insert_scan//
check_dft                                  //再次检测设计规范//
```

AUTOFIX 可以被设定对某一模块、某一器件或某一些器件的管脚进行修改，也可只修改时钟端或只修改复位端。它同时支持自上而下和自下而上两种设计流程。总得来说，对于时钟及复位端的错误，AUTOFIX 的功能还是比较强的。

而对于其他的错误则还是需要用手动的修改。

6.3 对于修改方法的小结

对于一般电路，用check_test检查设计规范，如果只是时钟与复位端有问题，则以AUTOFIX修复，如果还有其它错误，就必须用手动修改，直到所有问题解决后，才可做下一步编译。

以上所说的几种，都是亡羊补牢的做法。对于可测性设计来说，最好是在电路一开始的设计时，便考虑到测试结构的问题，尽量避免违反设计规范，这样在最后加入测试结构电路时便可以减少不必要的返工。当然，这需要设计者有相当多的设计经验以及对可测性设计的充分理解。

总结

对于可测性设计，只要能解决好电路的结构规范，扫描型结构总是能使电路达到满意的故障覆盖率。当然，文中谈到的都是最简单的情况，实际的情况往往要比这复杂的多。对一个复杂的大系统来说，将会碰到的问题要多的多，要达到一个满意的总故障覆盖率仍需要很多时间和精力，有时甚至不可能。总得来说，测试是一个系统的问题。光光一个芯片可测是远远不够的，必须做到芯片之间也可测，这就要用到 IEEE Std 1149.1 边界扫描标准，来达到 PCB 板级的可测。总之，测试日益成为电路设计师们要解决的首要问题，随着人力和财力的投入，相信总有一天能够找到一个完满的解决方法。

致谢

在本论文的写作过程中，许多同学和老师都给了很多的指导、帮助和建议。尤其要感谢唐长文老师，在软、硬件的使用上的对我不厌其烦的指导和帮助，以及在我论文的写作过程中，及时指出我论文中的错误和不足，给予了我无私的帮助。

参考文献

[1]. Jon Turino, "Design for Test considerations for ATPG", IEEE Journal of IC Design, vol 30, No. 3, March 1999

[2]. Cliff Warren, Engineer of American Microsystems, Inc. "The advantage of using logic BIST for ASIC designs", IEEE Journal of IC Design, vol 33, No. 1, March 2000

[3]. SZ Franz Rottner, Automatic Test equipment links design and production, Electronics Engineer, September, 2000

[4]. David Johns and Ken Martin, Scan Insertion and ATPG Development via Synopsys™ Test Compiler, Electronic Engineer, March 2001

[5]. 杨士元，数字系统的故障诊断与可靠性设计(第二版)，1999 年 10 月，清华大学出版社

[6]. 曾繁泰，VHDL 程序设计，2000 年 8 月，清华大学出版社

[7]. 孙艺，MCU 可测性设计的实现，微处理器 1999 年 第 3 期

[8]. 薛宏熙，苏明，数字系统设计自动化，1996 年 10 月，清华大学出版社