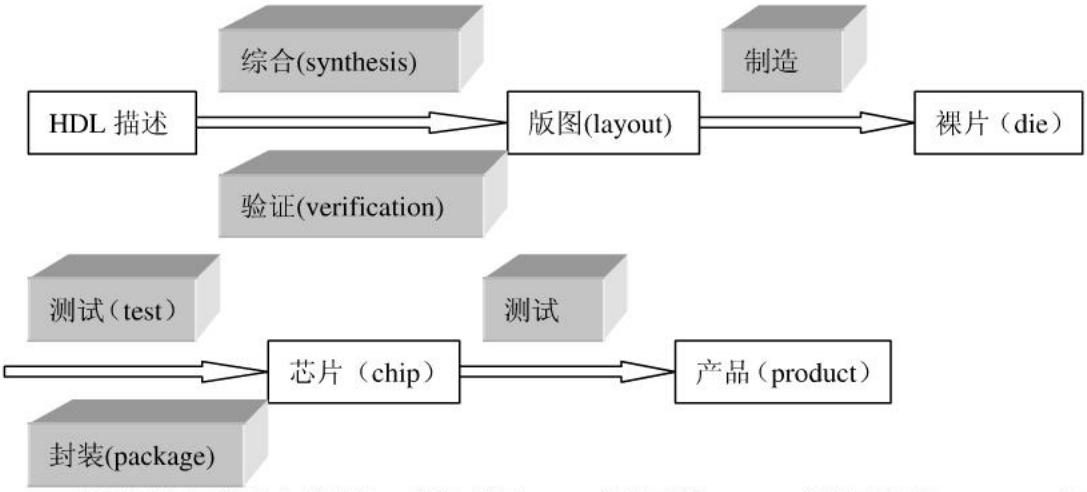
第四章 DFT 基础

4．1 测试在半导体产品实现过程中的意义

一 半导体产品的实现过程

集成电路从设计到产品一般要经历以下几个步骤才能成为产品（如下图所示）：

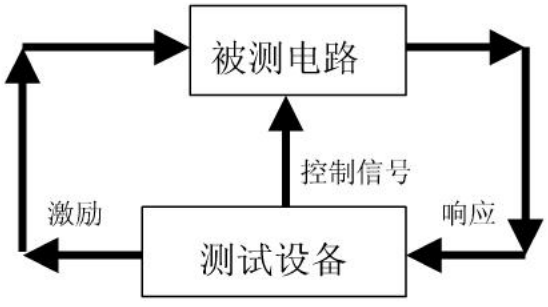


1. 设计过程：集成电路设计一般从编写 HDL 代码开始，HDL 代码可能用 verilog，也可能用 VHDL 语言写成，可能是 RTL 级，也可能是门级。如果是 RTL 级，首先进行逻辑综合、验证将设计转变成门级网表，然后进行布局布线变成最终的版图。
2. 制造过程：代工厂接受来自设计者的版图数据（GDSII）将其制成掩膜版（mask），然后通过复杂的制造过程将期望的电路做在晶园片上，这时晶园片上已经包含了若干个芯片的原型－－裸片（die）。
3. 晶园片测试：制造好的晶园片需要进行严格的测试然后划片、封装，实际上只有那些通过测试的裸片才会进行封装，而未通过测试的裸片被淘汰。经过封装的裸片就变成芯片。
4. 芯片测试：通过晶园片测试和封装的芯片还不能算真正的产品，它仍然要进一步进行测试确认没有故障才能成为真正的半导体产品。

从这个过程可以看出，测试是半导体产品实现过程中一个必不可少的环节。

二 测试定义

测试实际上是指将一定的激励信号加载到需要检测的半导体产品的输入引脚，然后在它的输出引脚检测电路的响应，并将它与期望的响应相比较以判断电路是否有故障的过程（如图所示）。



由图可以看出，要实现测试，首先要有激励信号，这个激励信号就是所谓的测试向量（test vector）。激励信号由测试设备产生；然后要判断电路是否有故障，就必须检测响应，并将实际检测的响应与期望的响应相比较，如果两者不一致，我们就认为电路中有故障。当然在这个过程中，测试设备要发出适当的控制信号，以使得整个测试过程得以顺利进行。

在测试领域，与测试向量相对应还有测试模式（test pattern），这两者的主要区别在于测试向量仅仅包含激励信号，而测试模式不仅包含激励信号，而且还包含期望的响应。

由上面的分析可以看出，测试问题在测试前就是测试模式生成和测试模式验证（时序验证）问题；而在测试时就是测试向量施加和测试响应检测及结果判断问题。

测试为最终的半导体产品的质量和可靠性提供一种度量。

测试是设计过程中验证（Verification)工作的继续，它实际上是对实际芯片的验证过程。

即使是经过严格验证的设计也很难保证设计是 100％正确，一个有经验的测试工程师可以从不同角度找出设计中可能存在的错误。

三 测试的分类

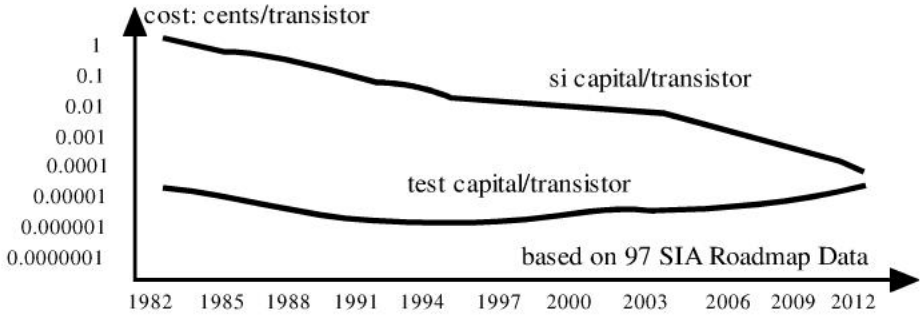
测试分功能测试和制造测试（结构测试）。

功能测试主要寻找设计上可能存在的错误，他是用来验证电路中的逻辑行为，是验证过程的延续，如果存在错误，需要进行故障诊断。

制造测试用于寻找在制造过程中可能存在的制造缺陷（开路、短路等）。

从产品实现过程看，测试分完全测试和制造测试。

完全测试一般在产品大批量投片以前进行，首先进行制造测试，然后进行详细的功能测试。这个功能测试用于检测那些具有确定性的故障；而在产品大批量投片时一般只进行制造测试，制造测试要检测的故障是随机的。随着集成电路制造技术的不断发展，集成电路已经进入深亚微米时代，集成电路的测试变得更加复杂，测试的费用不断增高，如下图所示。



图中数据代表的是每个晶体管的费用，上面一条线代表单个晶体管的硅片费用，这条线基本按摩尔定律变化；而下面一条线代表的是单个晶体管的测试费用，这条线在 1982～1991 年呈下降趋势，1991～1997 间基本不变，而在 1997～2012 确成上升趋势，按这个趋势发展，到 2012 年以后，测试费用几乎可以与硅片费用相比拟，这是令人无法容忍的。因此，测试问题就变得更为迫切。为了解决日益突出的测试问题，人们不得不从设计的一开始就开始考虑测试问题，这就是所谓的可测性设计问题。

4.2 可测性设计

可测性设计 DFT（design for test）就是指为了使测试（制造测试）尽可能简单而有意识地在设计中加入一定附加逻辑地设计方法。

可测性设计的意义主要有三点：

1 缩短产品进入市场时间，即 TTM（time to market）：由于在设计时考虑了可测性设计，这就使得 ATPG（automatic test pattern generation）可以进行，这样测试模式的生成时间大大减少，从而使得整个的设计周期大大缩短，从而大大加快产品进入市场的时间。

DFT＝>>ATPG＝>>测试向量生成时间减　少＝>>设计周期缩短＝>>TTM 缩短

2．降低测试费用即 COT（cost of test）：由于在设计时考虑了可测性设计，这就使得 ATPG 可以进行，而 ATPG 压缩测试模式，从而减少测试模式的数目，使得测试过程简单，从而降低测试费用。

DFT＝>>测试向量数目减少＝>>测试过程简单＝>>测试费用降低

3．提高产品质量：可测性设计可以使设计获得较高的故障覆盖率 FC(Fault Coverage)。

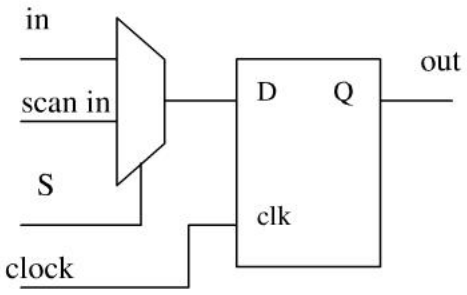
DFT=>>高故障覆盖率

需要指出的是测试向量不仅决定了测试时间，测试向量越多，测试时间越长；而且决定需要高档还是低档的测试设备，高档测试设备价格高，测试费用上升，低档测试设备价格低，测试费用也就低。

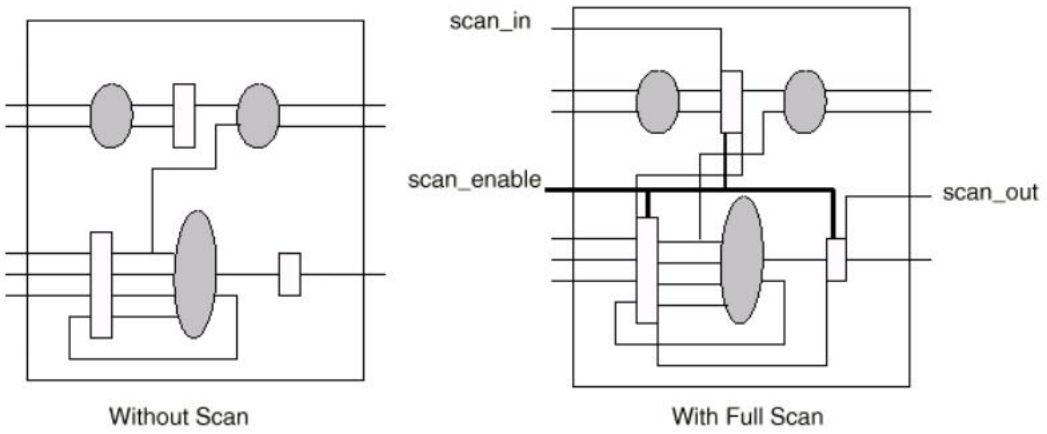
4．3 常用的可测性设计技术

1.扫描（scan）设计

基于扫描设计是可测性设计中最常用的一种方法。它是指将电路中的普通触发器（flip-flops）替换为具有扫描能力的扫描触发器。扫描触发器最常用的结构是多路器扫描触发器，即它在普通触发器的输入端口加上一个多路器如图所示。当 S=0 时，触发器为正常的功能输入，而当 S=1 时，触发器为扫描输入。



基于扫描设计分为两种：全扫描以及部分扫描。全扫描设计是指将电路中所有的触发器替换为扫描触发器并将它们连在一起构成扫描链，如图所示。图中，椭圆代表组合电路，长方形代表时序单元。左图为没有扫描链，右图为带有一条扫描链的全扫描设计。



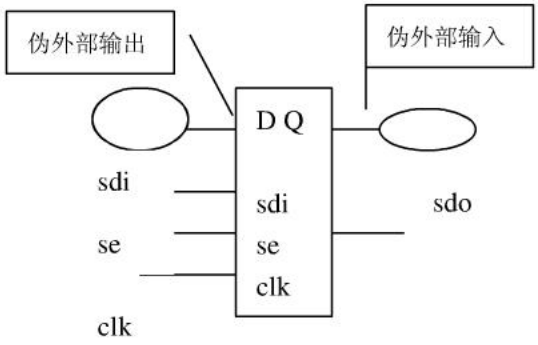
部分扫描设计则是将电路中的部分触发器替换为扫描触发器并将它们连在一起构成扫描链。部分扫描一般不用在那些关键的数据通路上，因为这些数据通路速度快，对延时的要求很高，而扫描替换增加了延时，故这些路径上的时序单元不进行扫描替换。

对于全扫描设计可以采用组合 ATPG 工具生成测试模式，而部分扫描设计只能采用时序 ATPG 工具生成测试模式，而时序 ATPG 工具的运行时间要比组合 ATPG 工具运行的时间长几倍，且时序 ATPG 工具对内存的要求要大得多，一般是组合 ATPG 的三倍以上。

扫描设计大大增强了电路的可测性，因为当电路中生成扫描链后，扫描链上的所有时序单元的 D 端都是扫描可观的，与 D 相连的节点就是可观性节点，而数据输出端 Q 端是扫描可控的，与 Q 相连的节点就是可控性节点。

扫描设计最初是为了解决时序电路的测试而提出的一种可测性设计方法，它是将电路中的时序单元替换成具有扫描能力的时序单元并将它们连接起来形成链的过程。

扫描设计的优越性不仅因为它解决了时序电路的测试问题，而且扫描链为电路中的组合逻辑部分提供了激励的加载通路和响应的观测通路。在基于扫描设计的芯片进行ATPG（automatic test pattern generation）时，扫描链上的每一个时序单元的数据输入端口D端作为一个外部输出端口以观测组合电路的响应，我们称之为伪外部输出端口[3]（pseudo-PO）；而扫描链上的各个时序单元的数据输出端口Q端都作为一个外部输入端口用来加载激励向量，我们称之为伪外部输入端口[2](pseudo-PI)，如图 1 所示。其中sdi为扫描数据输入端口，sdo为扫描数据输出端口，se为扫描允许端口，clk为时钟端口，椭圆代表组合逻辑电路，D为正常的数据输入端口，Q为正常的数据输出端口。



基于扫描设计可以显著地降低测试的复杂性，但是其不足在于使芯片面积略微增大，这不仅是因为扫描设计要将普通的触发器转换为扫描触发器的缘故，而且扫描设计大大增加了布线复杂性。

扫描测试的时序

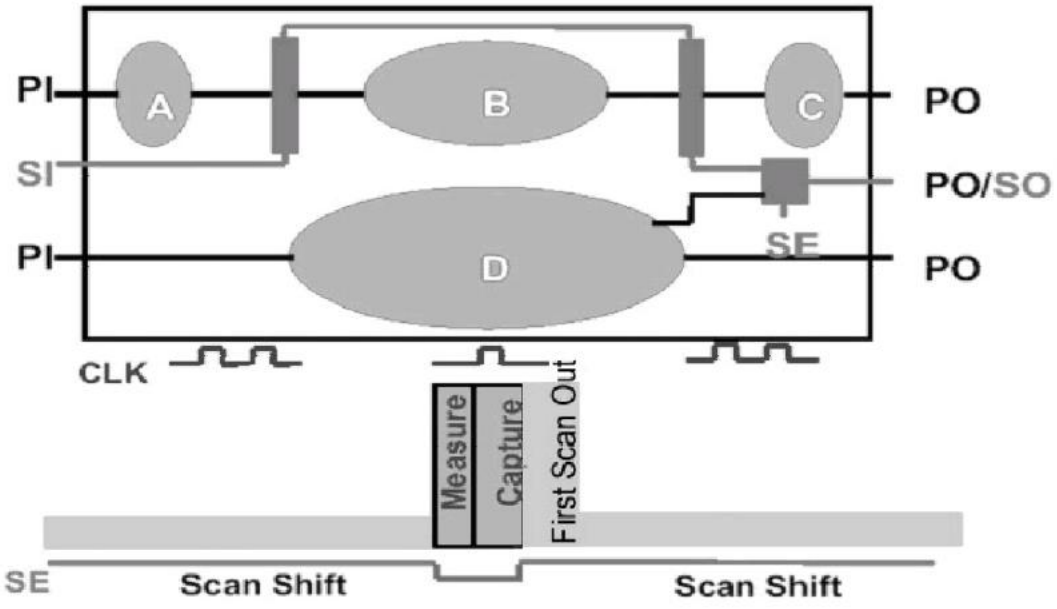
扫描测试的时序如下图所示。SI 为扫描输入，SO 为扫描输出，SE 为扫描允许。

扫描测试的第一步是通过扫描移位（shift）操作将设计中的时序单元设置为期望的值，这个过程所需的时钟数就是内部最长的扫描链长度。这个过程 SE＝1，电路工作在扫描移动测试状态。

第二步是施加激励并抓取响应（capture）。这个过程 SE=0，电路工作在正常的功能状态。

第三步是移出抓取的电路组合部分的响应。这个过程 SE＝1，电路工作在扫描移动测试状态。

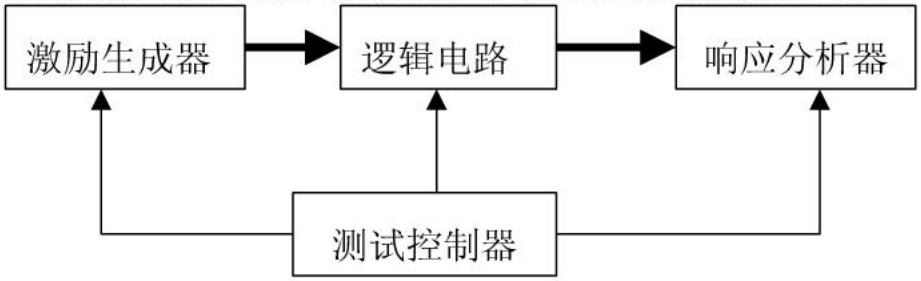
值得注意的是当需要施加多个测试模式时，在前一个测试响应移出电路的同时就移入当前电路的测试激励。



2. 内建自测试（BIST）

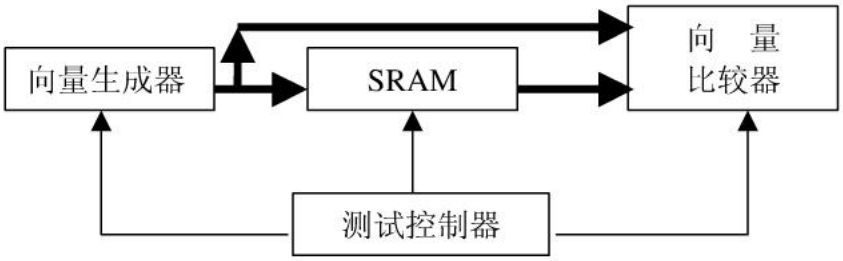
内建自测试是可测性设计的另一种重要的方法。这种方法的基本思想是由电路自己生成测试向量，而不是要求外部施加测试向量，它依靠自身逻辑来判断所得到的测试结果是否是正确的，这样就大大降低了对测试设备的要求，而且它所要求“借用”的芯片封装引脚的数目要少得多。内建自测试的基本原理如图所示。

BIST 又分为逻辑内建自测试(LBIST)和存储器内建自测试（MBIST），MBIST 又分为 RAMBIST 以及 ROMBIST。由于 BIST 要求电路自身生成测试向量，而随机逻辑的测试向量生成是非常复杂，故逻辑内建自测试在实际中应用有限，最常用的是 MBIST。



2.1 RAMBIST

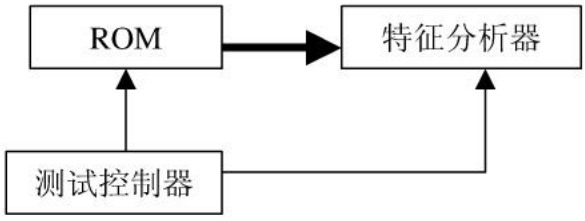
由于 RAM 可读可写，因此我们要从读和写两个方面对它进行测试，其原理如图所示；又由于 RAM 结构规整致密，故其测试向量不像普通电路测试向量那样复杂，RAM 测试的关键在于施加测试向量的时序上，最常用的是 March 算法。



2.2 ROMBIST

ROM 与 RAM 最大的不同之处是 RAM 可读可写，而 ROM 只读不可写，ROM 中的信息是由制造厂家做好了的，因此 ROMBIST 与 RAMBIST 的最大不同就是前者没有向量生成电路，但由于 ROM 中的信息是多种多样，故其响应分析是非常复杂的，通常要用特征分析器首先对其响应进行压缩得出响应特征，然后针对特征进行比较，其结构原理图如下图所示。

ROM BIST的作用有两个：一方面是要验证存储在ROM中的数据是否正确；另一方面是确保能够准确读出存储器中的信息，而不出现破坏性的读操作，即在进行读操作时不会改变或毁坏数据。ROM测试也可以采用类似RAM测试的March算法[1]，



3. 边缘扫描（Boundary Scan）

边缘扫描是欧美一些大公司联合成立的一个组织——联合测试行动小组（JTAG）为了解决 PCB 板上芯片与芯片之间互连测试而提出的一种解决方案。由于该方案的合理性，它于 1990 年被 IEEE 采纳而成为一个标准，即 IEEE1149.1。边缘扫描是在芯片的每一个输入

输出引脚上增加一个存储单元，然后再将这些存储单元连成一个扫描通路，从而构成一条扫描链。由于这条扫描链分布在芯片的边缘，故称为边缘扫描。边缘扫描单元的原理如图所示。

边缘扫描单元有以下四个数据端口：di 代表功能数据输入端口，do 代表功能数据输出端口，sdi 代表扫描数据输入端口，sdo 代表扫描数据输出端口。

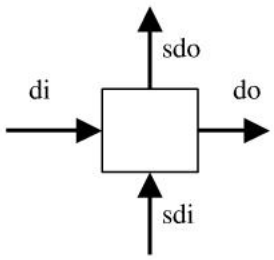
边缘扫描单元有四种操作：

1　di－>do 这是正常的功能操作。

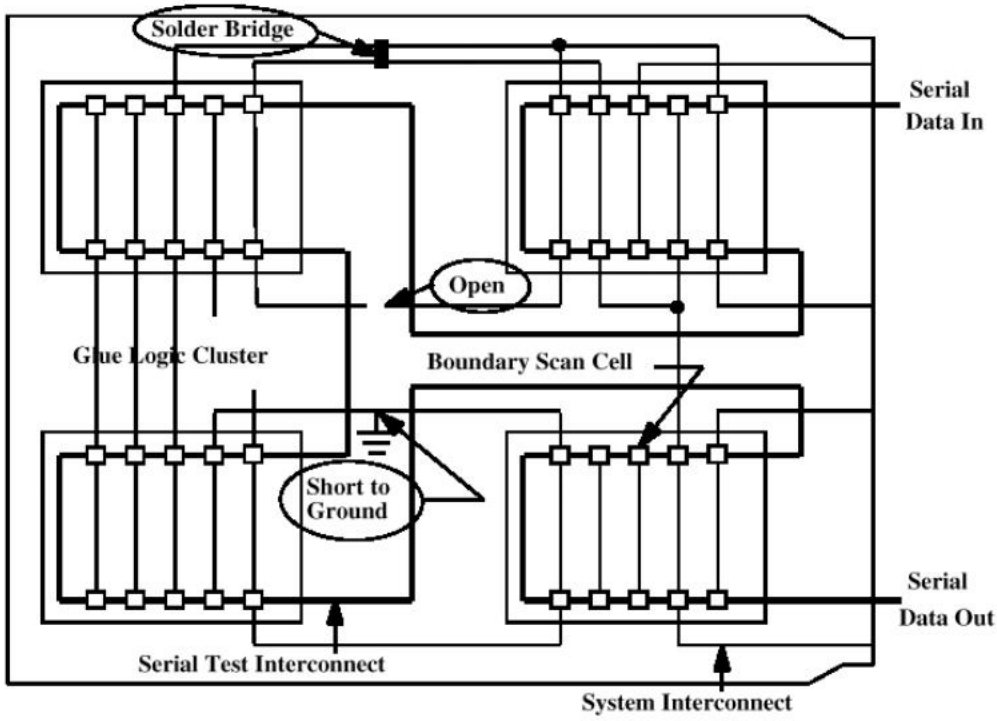
2　sdi－>sdo 这是测试时扫描移位操作。

3　sdi－>do 这是测试时将扫描移动数据施加到功能端口。

4　di－>sdo 这是测试时抓取功能端口的数据进入边缘扫描单元。



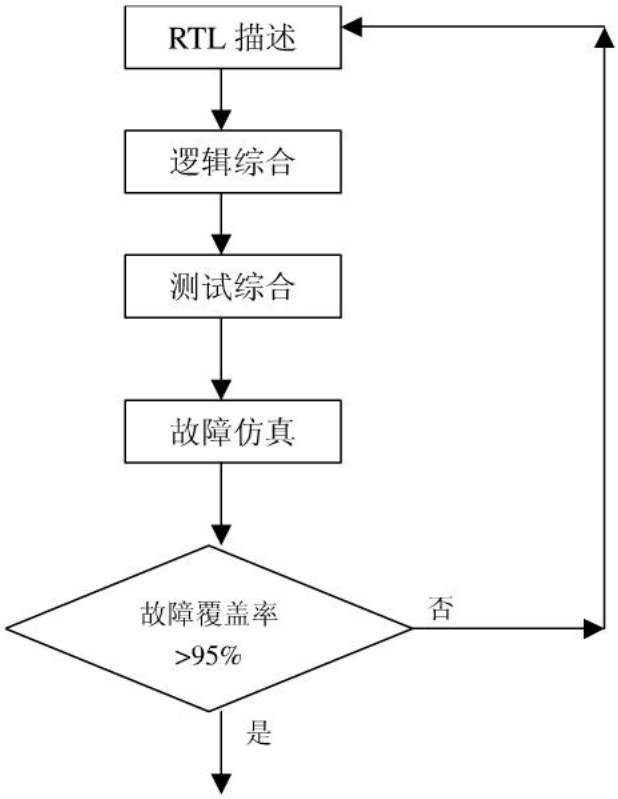
边缘扫描占用四个芯片引脚，即测试数据输入（TDI）、测试数据输出（TDO）、测试模式选择（TMS）以及测试时钟（TCK）。在正常的工作模式下，边缘扫描单元作为通常的输入输出器件；在测试模式下，测试向量将被施加到扫描输入、输出芯片的引脚。



边缘扫描与基于扫描设计有着明显的区别，前者是在电路的输入输出端口增加扫描单元，并将这些扫描单元连成扫描通路；而后者是将电路中普通时序单元替换为具有扫描能力的时序单元，再将它们连成扫描通路。

4 DFT 设计范畴的任务

概括地说，DFT 可分为两个范畴，一个是设计范畴，另一个就是测试模式生成范畴。设计范畴的任务可以用下图表示。



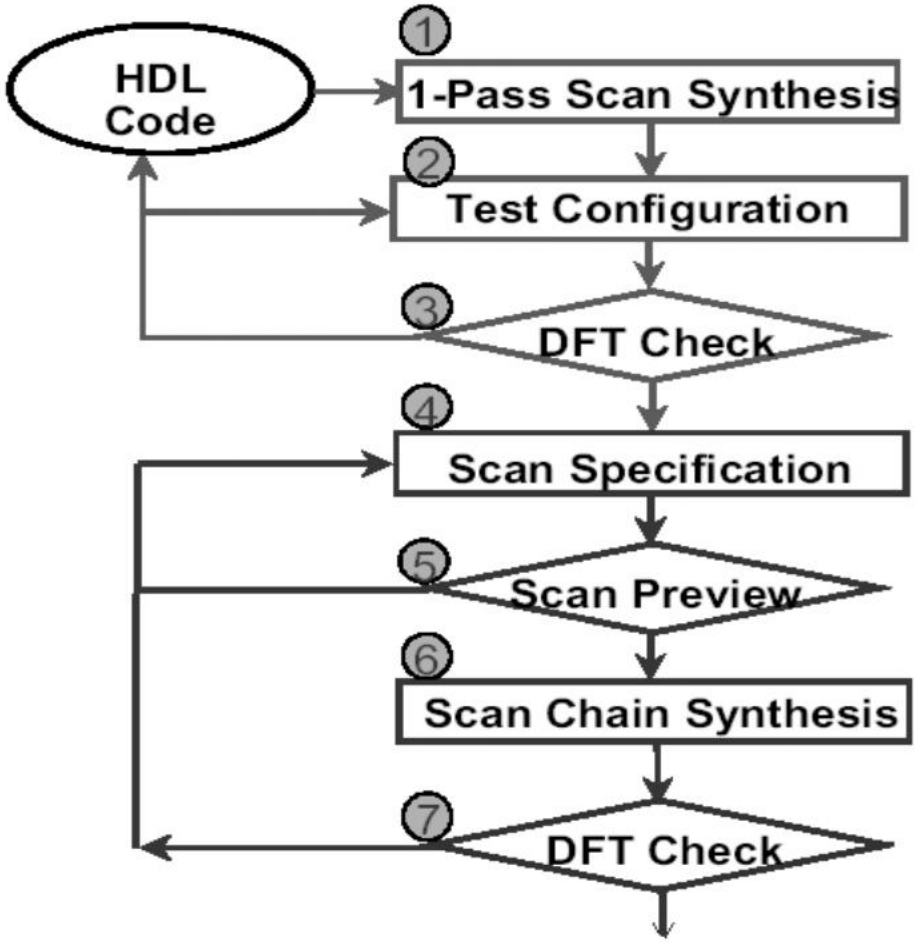
其中测试综合包括扫描综合、BIST 综合，如果需要的话 JTAG 电路的综合。

扫描综合的目标是故障覆盖率达到 95％以上，扫描综合的工具是用 SYNOPSYS 的 Test Compiler。

扫描综合－－Test Compiler 的使用

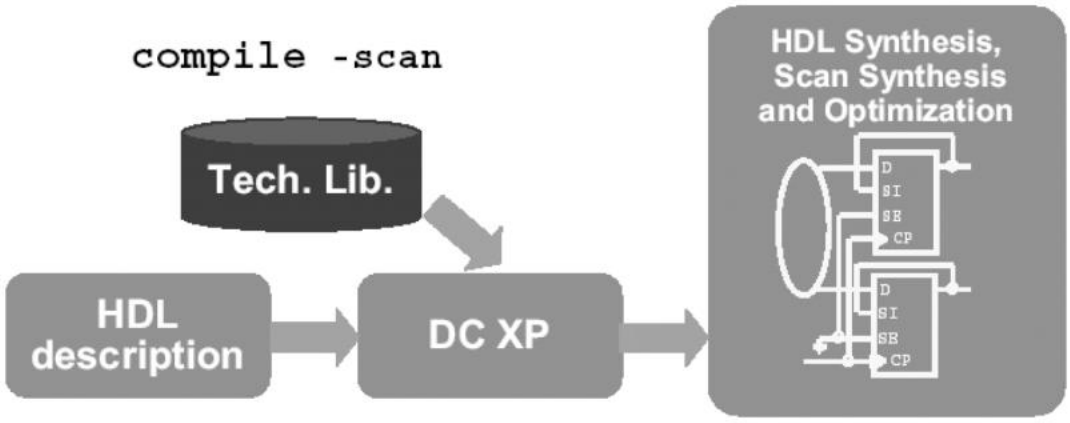
扫描综合主要分为两步：扫描单元替换（scan cells replacement）和扫描链连接（scan chains assembling）。扫描单元替换就是将电路中的普通时序单元替换为具有扫描能力的扫描时序单元。扫描链连接就是将经过扫描替换的扫描时序单元连接起来形成一条链的过程。

扫描综合的完整步骤如下图所示。共分 7 步。



第一步：一次通过的扫描综合（one-pass scan synthesis）。这一步将逻辑综合与扫描综合整合在一起一步完成，即在逻辑综合的同时就考虑到扫描结构对电路性能的影响。

compile -scan



第二步：测试结构配置：

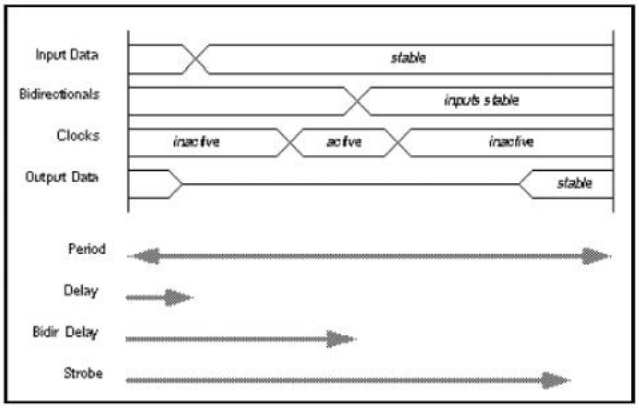
Test Configuration

ATE Cycle Variables

```
test_default_period          =100
test_default_delay           =5
test_default_bidir_delay     =55
test_default_strobe          =95
test_default_strobe_width    =0
```

Default Test Clock

```
period test_default_period
1st edge 0.45*test_default_period
2nd edge 0.55*test_default_period
```



第三步：DFT 检测：

在扫描链连接之前检测门级网表中可能存在的违反扫描设计规则问题，一方面通知设计者扫描结构可能存在的问题，另一方面检测替换后的扫描触发器是否违反扫描设计规则。

主要命令如下：

set_test_hold

check_test

read_init_protocol

read_test_protocol

第四步：扫描设计规范：

常用命令：

set_scan_configuration –methodology [style, chain_count, clock_mixing, bidi_mode, add_lockup, replace, disable]

set_scan_path

set_scan_signal

set_scan_element

set_scan_segment

set_scan_transparent

第五步：扫描结构回顾（preview）：

报告你设定的扫描结构，检查是否完全，是否有遗漏等。

Preview_scan

第六步：扫描链连接：

insert_scan

注意：对于非一次通过综合，insert_scan 同时完成扫描单元替换和扫描链连接。

第七步：DFT 检测：

完成扫描链后进行 DFT 检测，进一步检测是否存在扫描测试规则违反情况，以确认电路中不存在 DFT 问题。

Check_test

Report_test –scan_path

一个完整的 check_test 报告主要包括以下几个方面：

Error：严重的测试问题，这种问题必须解决

Warning：可测性问题，不解决会降低故障覆盖率

Information：提供更多的信息

Test design rule violation summary：可能存在的测试规则违反如下：

Total Violations: 12

Topology Violation

1 combinational feedback loop violation (TEST-117)

Scan In Violations

2 cells constant during scan violations (TEST-142)

Capture Violations

1 clock used a data violation (TEST-131)
4 illegal path to data pin violations (TEST-478)
4 cell does not capture violations (TEST-310)

Sequential cell summary：时序单元综述：

Sequential Cell Summary

11 out of 45 sequential cells have violations

Sequential cells with violations

4 Cells have parallel capture violations
2 Cells have constant values
2 Cells have scan shift violations
3 Cells are black box

Sequential cells without violations

30 cells are valid scan cells
4 cells are transparent latches

如果设计中出现测试规则违反时，我们应该处理如下：

Specify proper test configuration

Constant values to be held at primary inputs
Initialization sequences to enter a test mode

Workaround the problem

Insert DFT logic to bypass the problem

Redesign the module

Avoid uncontrollable clocking
Avoid uncontrollable asyn. set/reset
Avoid uncontrollable three-state/bidir enables

Ignore the problem

Only if the reduction in FC is allowable

下面我们给出一个完整的测试综合的脚本：

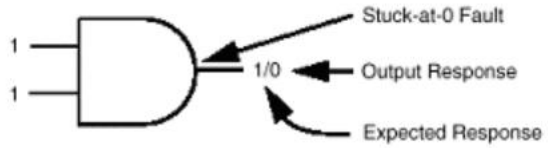
Test Synthesis Script

```
current_design TOP
compile –scan                                   /* test-ready compile */
set_test_hold 1 find (port, TM)                 /* set TM=1           */
check_test                                      /* pre-scan check     */
set_scan_configuration –chain_count 2           /* 2 scan chains      */
set_scan_configuration –bidi_mode input         /* bidir_mode=input   */
set_scan_configuration –clock_mixing mix_clocks /* multi-clocks/chain */
set_scan_signal test_scan_in –port {si1, si2}   /* assign scan in port*/
set_scan_signal test_scan_enable –port SE       /* assign scan enable*/
set_scan_signal test_scan_out –port {so1, so2}  /* assign scan out    */
preview_scan                                    /* scan preview       */
insert_scan                                     /* scan synthesis     */
check_test                                      /* post-scan check    */
report_test –scan_path                          /* scan chain report  */
```

4．4　故障模型及相应制造测试技术

1 固定故障模型（stuck-at faults）

固定故障模型主要反映的是电路中某个节点上的信号不可控性，也就是说在电路正常工作过程中该节点电平始终固定在某一个值。如果固定在高电平上就称之为固定 1 故障（stuck-at-1），如果固定在低电平上就称之为固定 0 故障（stuck-at-0）。



2 固定故障测试（stuck-at testing）

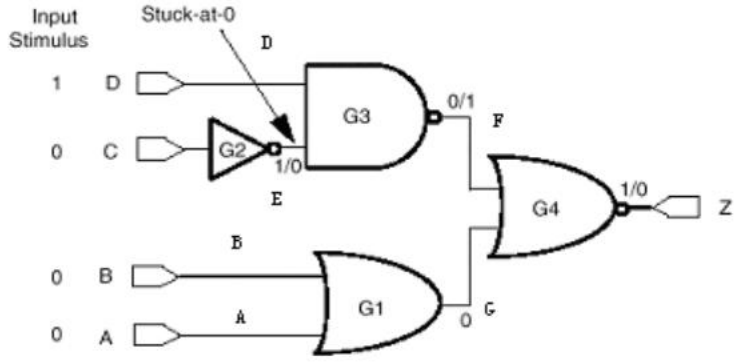
基于固定故障模型的测试就是固定故障测试。固定故障测试有一个重要的假设，那就是单故障假设，即在分析、评估和诊断的任何时间都只存在一个故障；这一假定降低了分析的复杂性。固定故障测试的原理如图所示。

固定故障测试主要分两步：

1 敏化通路（sensitive path）：在输入端口上加上适当的激励信号使得故障点的信号可以在没有阻碍地到达输出端口。

3　激活故障，并传送故障至输出端口。

这个过程如下图所示。



假定E点有一个固定0故障，要辨别E点是否有固定0故障，必须要使E点电平为1，并将E点为1的电路操作传送到Z。

敏化通路：为了使E点故障被传送， 必须D点为1，这样F＝0，要使F＝0传送到Z，则G＝0，要使G＝0，则A=0,B=0,于是A=0,B=0,D=1就是通路敏化的条件。

故障激活并传送：为了激活故障，则C=0。这样在E点无固定0故障时Z＝1，如果有固定0故障时Z＝0。

这样，测试E点的固定0故障的测试向量就为：ABCD=0001。

4 跳变故障模型（transition fault）

跳变故障模型主要验证电路的时序关系，即电路中的门延时。也就是说在电路正常工作过程中该节点电平变化太缓慢以至于电路工作不正常。与固定故障模型相似，它也有两个故障模型即慢下降模型（slow-to-fall）对应于固定1故障；慢上升模型（slow-to-rise）对应于固定0故障。

5 跳变测试（transition testing）

基于跳变故障模型的测试就是跳变测试。与固定故障测试类似，跳变测试也有所谓的单故障假设。

与固定故障测试不同的是跳变测试在测试一个故障时需要两个向量，一个是初始向量（initialization vector），一个是跳变向量（transition vector）即所谓的向量对（vector pairs）：初始向量的作用是建立故障传送的通路（敏化通路），并设置故障点的初始值；跳变向量设置故障点期望的跳变值。对于上图的电路，初始向量为：ABCD=0011，跳变向量为：ABCD=0001。

为了检测故障，在施加跳变向量后的适当时间进行响应检测。相应的跳变测试需要以下四部：

1 施加初始向量

2 施加跳变向量，启动一次跳变

3 等待适当时间

4 在输出端口检测响应，抓取跳变的结果

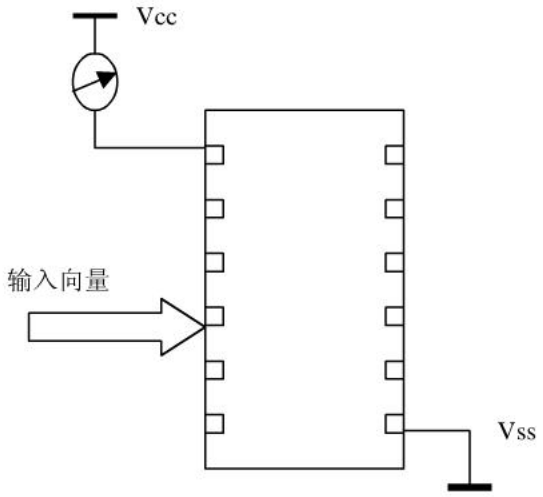
6 基于电流测试－－IddQ测试

CMOS电路一般可分为三种类型，即完全静止型、阻抗型（registive）比如上拉/下拉电路等以及动态逻辑电路。完全静止CMOS电路在正常工作时其静态电流IddQ接近为0，而IddQ测试就是基于这一原理来进行的。也就是说对于一个无故障器件而言其静态电流接近于0，而吸收过量静态电流的电路其内部可能存在某种制造缺陷。其原理如图所示。

IddQ测试分为两步：

(1) 测试设备施加测试向量使CMOS电路稳定

(2) 测量电源的静态电流



需要指出的是 IddQ 测试与固定故障测试以及跳变测试不是矛盾的，而是互补的。理想情况下任何测试向量都可监测其 IddQ，但实际测试过程中一般只选择其中一部分（一般 20 个测试模式即可）。

4．5 测试模式生成——ATPG

自动测试模式生成(ATPG)的任务是根据所采用的故障模型确定一最小的激励向量集以使得设计的故障覆盖率达到期望值。从这个定义可以看出，ATPG 的前提是确定故障模型，不同的故障模型有不同 ATPG，也就意味着有不同结果；ATPG 的目标是获得一个最小的激励向量集——测试模式集，不同的故障模型有不同的测试模式集。

最初测试模式通过两个渠道获得：

1 由芯片级仿真时的仿真向量转换而成

2 由测试工程师手工开发而成,很明显，这样的测试模式生成过程是非常费时费力的，一个稍微复杂一点芯片这个过程可能要好几个月；而 ATPG 使得这一过程自动化，大大缩短了测试模式生成的时间，而且节省宝贵的人力资源。

一个完整的 ATPG 过程由以下三个子过程组成：

预 ATPG 过程

ATPG 过程

后 ATPG 过程

1 预 ATPG 过程

创造 ATPG 库：ATPG 工具能够识别的标准单元库。

设计描述格式化：将设计描述的格式转化为 ATPG 工具能够接受的格式。

建立 ATPG 约束：为了使 ATPG 能够顺利进行，必须对那些测试控制信号以及相关的信号建立适当的约束。

2 ATPG 过程

随机性模式生成：首先随机的给出一些测试激励，通过故障仿真，获得相应的期望响应，从而组成测试模式，同时通过故障仿真分析得出那些故障已经被现有的测试模式覆盖，并将其从故障列表中删除。它实际上是一个由测试模式到故障的过程。

决定性模式生成：当随机增加测试向量对故障覆盖率的提高贡献很小时，我们就应该转到决定性模式生成上来。它实际上是先从故障列表中挑出一个故障，然后根据通路敏化技术，确定相应的测试模式。它实际上是一个由故障到测试模式的过程。

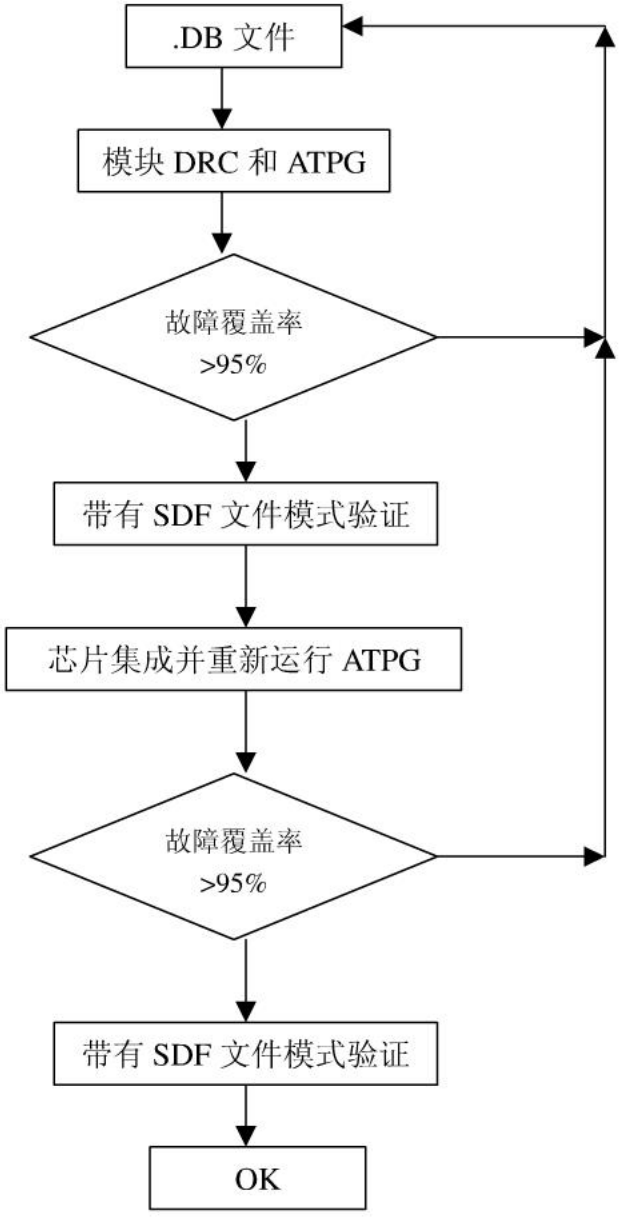
3 后 ATPG 过程

测试模式压缩：由上述的 ATPG 过程生成的测试模式还可进行合并，使得测试模式的数目进一步减少。

测试模式格式化：ATPG 工具生成的测试模式格式可能并不一定能被相应测试设备所接受，因此还需要将测试模式转化为测试设备能接受的测试模式格式。

测试模式验证：ATPG 工具在进 ATPG 时一般不考虑电路延时，因此生成的测试模式是否可行，还必须进行时序验证，这个验证采用仿真工具比如 Verilog-XL 或 VCS 等，但需要将布局布线后生成的 sdf 文件并将它反标到门级网表（netlist）中。

前面已经提到 DFT 设计范畴的任务，现在我们来总结一下测试模式生成范畴的任务。这个任务如图所示，它主要包括 ATPG 和测试模式验证。



ATPG 工具主要有 SYNOPSYS 的 TetraMAX 以及 MENTOR 的 FastScan。

ATPG－－TetraMAX 的使用

1. TetraMAX 的特点

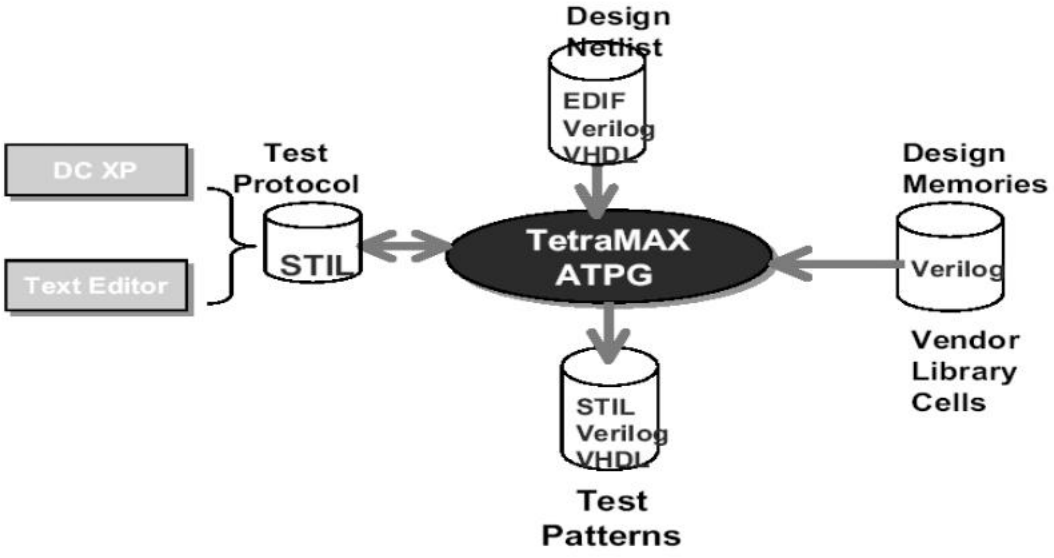
- 使用现存的供应商提供的功能仿真库，可以是 Verilog 格式，也可以是 VHDL 格式。
- TetraMAX 内部集成了故障仿真器，可以直接进行功能模式仿真，获得相应的故障覆盖率和测试覆盖率。
- 支持 RAM 和 ROM
- 支持 IDDQ 测试模式生成
- 提供交互式 DFT 规则分析和调试能力
- 在调试信息和在线手册之间提供超文本连接
- 支持 Verilog、EDIF 和 VHDL 网表输入
- 支持 Verilog、WGL 和 VHDL 模式输入
- 支持 Verilog、WGL 和 VHDL 模式输出
- 支持 STIL 输入输出
- 提供静态和动态的测试模式压缩技术
- 提供内嵌的 GZIP 压缩功能，它可以直接读入被压缩的设计、库、协议、测试模式和故障列表，也可以直接保存设计、库、协议、测试模式和故障列表为压缩格式。

2. TetraMAX 输入输出

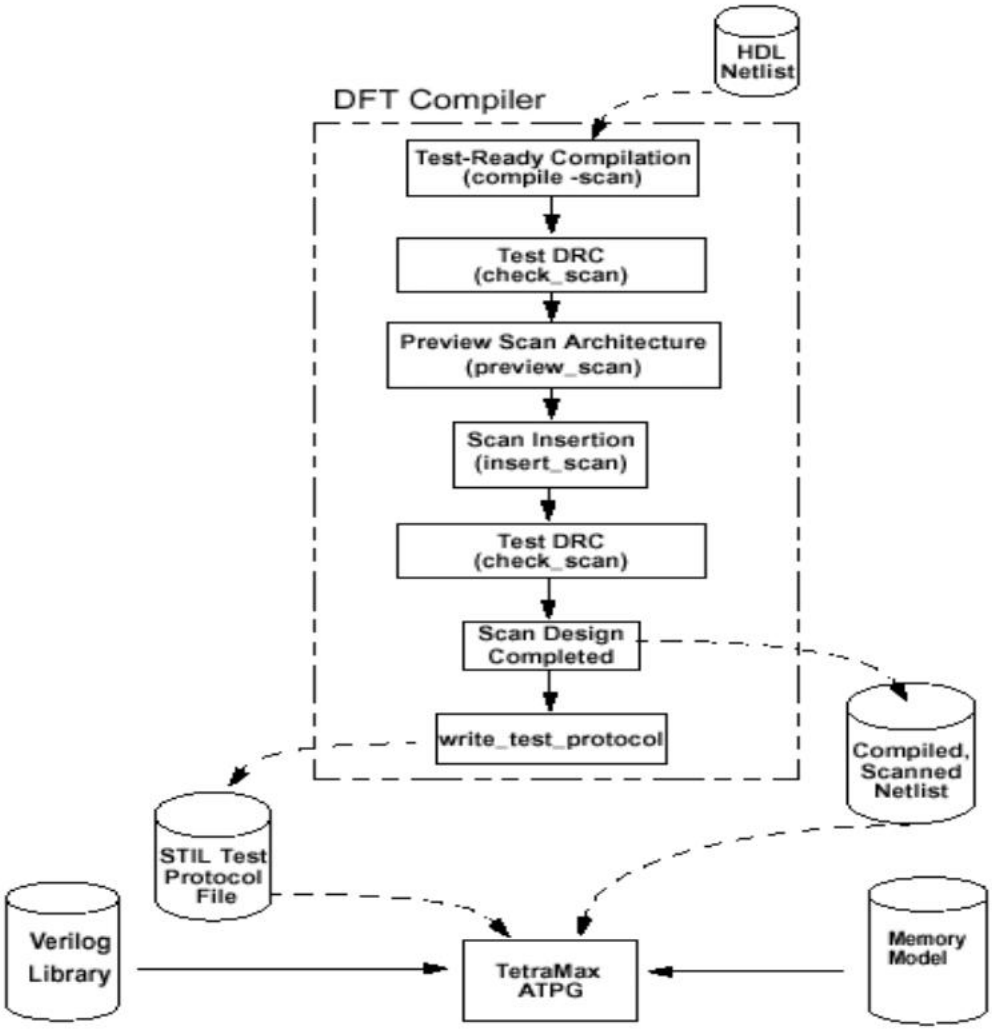
TetraMAX 输入输出如图所示。

TetraMAX 的输入包括三个部分：设计网表、ATPG 库和测试协议。

- 设计网表：设计网表可以是 Verilog 格式，可以是 VHDL 格式，也可以是 Edif 格式，或者是上述三种格式的混合。
  设计描述可以是分层次的，也可以是平坦的。
  设计可以是一个文件，也可以是多个文件。
- ATPG 库：ATPG 库就是功能仿真使用的仿真库，可以是 Verilog 格式或 VHDL 格式。
- 测试协议文件 SPF：这个文件可以手工生成，也可以是 DC 自动生成，一般 DC 自动生成的文件可能需要进行部分修改。

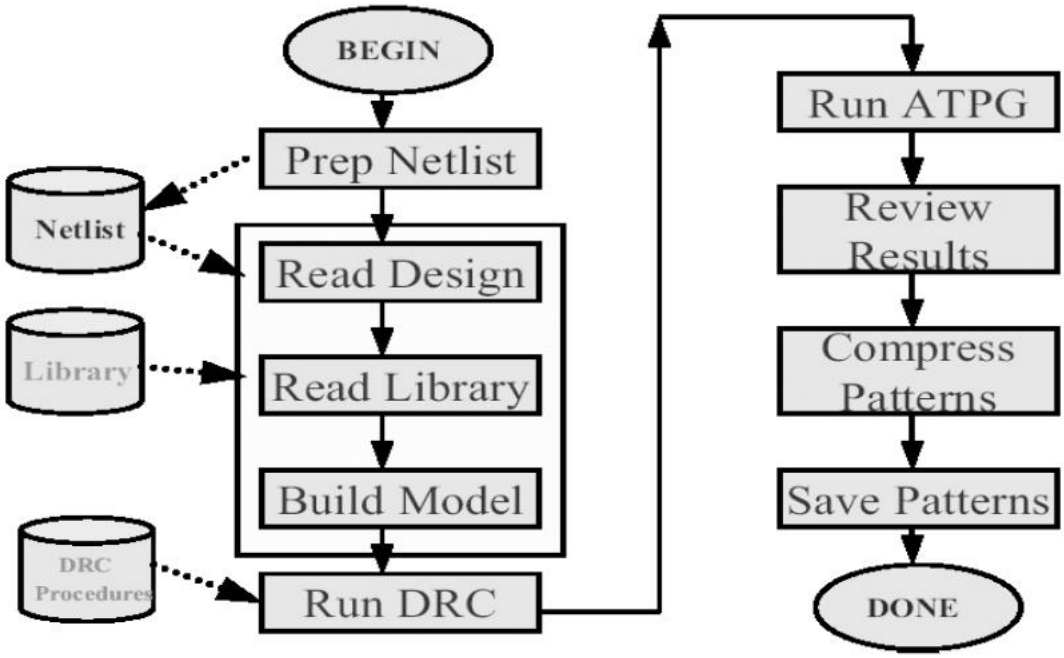


DC 与 TetraMAX 之间的联系如图所示。



3. TetraMAX 流程

TetraMAX 流程如图所示主要分 7 步，叙述如下：



- 读入设计和库。

read netlist filename

允许使用*以及？等符号。

read netlist /design/*/?design??.v

- 创造 ATPG 模型

run build_model topmodule

- 读入测试协议文件 SPF_file，执行 ATPG 设计规则检查 DRC

run drc SPF_file

所检查的内容涉及到：

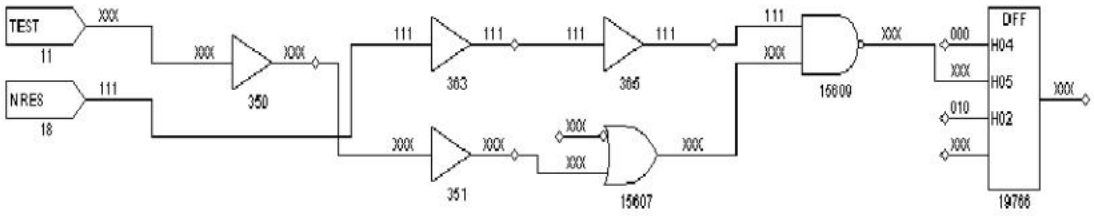
检查扫描链是否正确连接，在扫描移动过程中扫描链是否正确移动

检查时钟和异步置位、复位信号是否仅仅由芯片外部输入引脚控制

检查多驱动网络点是否存在冲突

如果出现违反情况，我们就要分析出现违反的原因，具体步骤如下：

- 使用 ANALYZE 按钮，选择一个违反，则 GSV 窗口将会显示出现问题的地方
- 跟踪 X 态到 TEST 引脚（如图所示）
- 原因：TEST 引脚未被约束
- 编辑 SPF 文件添加 TEST 引脚约束，重新运行 DRC



- 运行 ATPG

在运行 ATPG 之前要建立故障列表

add_faults -all

如果故障列表已存在，则可以直接读入：

read fault fault_list_name

然后运行 ATPG，执行命令

run _atpg

- 报告生成结果

report summaries

```
TEST> report summaries
Uncollapsed Fault Summary Report
-----------------------------------------------
fault class                     code    #faults
------------------------------  ----  ----------
Detected                         DT       83348
Possibly detected                PT         324
Undetectable                     UD        1071
ATPG untestable                  AU        3453
Not detected                     ND         212
-----------------------------------------------
total faults                              88408
test coverage                             95.62%
-----------------------------------------------
            Pattern Summary Report
-----------------------------------------------
#internal patterns                         1636
-----------------------------------------------
```

- 测试模式压缩

run pattern_compression N

N 为运行压缩的次数。最大为 99。

- 保存测试模式

write patterns filename –internal –format <format>

<format>代表：verilog_single_file or VHDL or stil or wgl

下面给出一个典型的 TetraMAX 的脚本文件如图所示。

```
/* read design and library netlists */
read netlist *.v -d
/* set build option and run build ATPG model */
set build -black_box module
run build topmodule
run drc filename.spf
/* set fault lists, ATPG options then run ATPG */
add fault -all
run atpg -auto
/* write out test patterns */
write patterns file.v -internal -format
  verilog_single_file
write patterns file.wgl -internal -format wgl
```

在 ATPG 过程中可以生成故障列表，以备下次运行时使用。

write faults filename -all –replace

TetraMAX 可以读入上次运行生成的故障列表或其它工具生成的故障列表。

read faults fault_list_name

4．SPF 文件的特点以及生成方法

SPF 代表 stil procedure file，它可以提供以下信息：

- 扫描输入输出的名
- 时钟端口
- 引脚、时钟以及信号测量的时序定义
- 需要进行约束的端口定义
- 使用 test_setup 宏建立测试过程控制信号序列
- 使用 load_unload 建立扫描允许控制信号序列
- 使用 shift 过程建立扫描移位序列
- 可能还会包括其它的设计依赖过程：master_observe，shallow_observe
- SPF 可能还包括其它的模式数据

SPF 文件生成：

我们可以使用命令生成 SPF 模板。

write drc filename.spf -replace

然后在生成的 SPF 文件模板中增加相关信息。

添加时钟：add clock <off_state> <port_name>

添加端口约束：add pi constraint <hold_value> <port_name>

添加等效端口：add pi equivalence <port_name> [-invert] <port_name>

需要特别指出的是，这里的时钟 clock 是指任何影响时序单元值的信号，包括通常意义上的时钟以及置位复位信号等。

一个典型的 SPF 文件如图所示。

SPF consists of

- **STIL header**
- **ScanStructures block**
- **Procedures block**
  - **load_unload procedure**
  - **Shift statement**
- **test_setup macro is optional, but may be needed to initialize a particular design for test mode**

```
STIL;
ScanStructures {…}
Procedures {
  load_unload {
    V {…}
    Shift {…}
  }
}

MacroDefs {
  test_setup {…}
}
```

扫描链定义：

扫描链是由 ScanStructures 模块定义的；例如

```
ScanStructures{
        ScanChain “c1” {ScanIn SI1; ScanOut SO1;}
        ScanChain “c2” { ScanIn SI2; ScanOut SO2}
          }
```

数据 load_unload 定义：

```
Procedures {
  "load_unload" {
     V { CLOCK=0; RESETB=1; SCAN_ENABLE=1; }
  }
```

扫描移位定义：

```
Procedures {
  "load_unload" {
     V { CLOCK=0; RESETB=1; SCAN_ENABLE = 1; }
     Shift {
        V { _si=####; _so=####; CLOCK=P;}
     }
     V { CLOCK=0; RESETB=1; SCAN_ENABLE = 0;}
  }
}
```

shift 语句定义了扫描链如何一位一位进行移动。Shift 过程语句由一个测试周期语句 V 组成。而 V 由三部分组成：_si 扫描数据输入，_so 扫描数据输出，CLOCK 端口加载脉冲 P。＃是一种特殊的占位符，四个#意味着有四个扫描链。在 shift 语句中可以使用重复因子 r，其用法为：\r<重复数>。例如上面的语句可以改写成为：

V {_si=\r4　#；_so=\r4　#；CLOCK=P；}

当 load_unload 过程语句施加时，shift 过程语句被重复施加，重复的次数等于最长扫描链的位数。

在 shift 语句后增加了一个测试周期语句 V，这是一个可选的语句，它确保在移位过程中时钟和异步置位复位信号处于非激活状态。

测试初始化：test_setup

test_setup 宏通过保持某一顶层端口为一恒定值使得设计进入测试模式。如图所示，test_setup 宏由两个测试周期组成。

```
MacroDefs {
  "test_setup" {
            V  { TEST_MODE=1; RESETB=0;}
            V  { RESETB=1;}
  }
}
```

5．TetraMAX 的故障类型及测试覆盖率

TetraMAX 将故障分为 5 大类 11 小类，5 大类为：

- DT: Detected
- PT: Possibly Detected
- UD: Undetected
- AU: ATPG Untestable
- ND: Not Detected

11 小类为：

- 11 Fault Classes
  - DT: detected
    - DS: detected by simulation
    - DI: detected by implication
  - PT: possibly detected
    - AP: ATPG untestable, possibly detected
    - NP: not analyzed, possibly detected
  - UD: undetectable
    - UU: undetectable unused
    - UT: undetectable tied
    - UB: undetectable blocked
    - UR: undetectable redundant
  - AU: ATPG untestable
    - AN: ATPG untestable, not detected
  - ND: not detected
    - NC: not controlled
    - NO: not observed

测试覆盖率定义为：

Test Coverage is defined as

$$\text{Test Coverage} = \frac{DT + (PT * posdet\_credit)}{all\ faults - (UD + (AN * au\_credit))}$$

posdet_credit = 50% (default)

au_credit = 0% (default)