第 1 章
ASIC 设计方法学

由于深亚微米半导体几何尺寸的缩小，使用传统的芯片设计方法进行设计已经变得越来越困难。此外要在相同的芯片面积下将晶体管数目增加，会使设计工作变得更加艰巨。而且，在“上市时间”这一理念的压力下，芯片设计周期需要保持原有的水平甚至要不断地缩短。为了克服这些问题，就需要发展新的方法和工具来完善 ASIC 设计方法学。

本章主要讲述在亚微米领域芯片设计各个阶段的最新技术，同时对设计流程的各种改进技术也进行了讨论。

在本书的上一版之后，Synopsys 引进了一个称为 Physical Compiler 的工具。在这个工具中，综合和布局联系得更加紧密，从而使传统的设计流程有了重大的改变。本章强调新技术的重要性，并且解释在设计流程中使用这些新技术缩短整个设计周期以取得最大效益的必要性。考虑到这个工具对 IC 设计界还相当新，而且还未 100%地被 ASIC 设计界所接受，因此传统的设计流程和新的设计流程都在本书中有所讨论。

本章重点论述基于 ASIC 设计流程方法学，从 RTL 编码到最终定案下单（tape-out）的全部综合过程，对于传统和基于 Physical Compiler 的流程也都进行了讨论。

1.1 传统的设计流程

传统的 ASIC 设计流程（如图 1-1 所示）大致包含如下步骤，后面章节将详细地叙述与综合相关的主题。

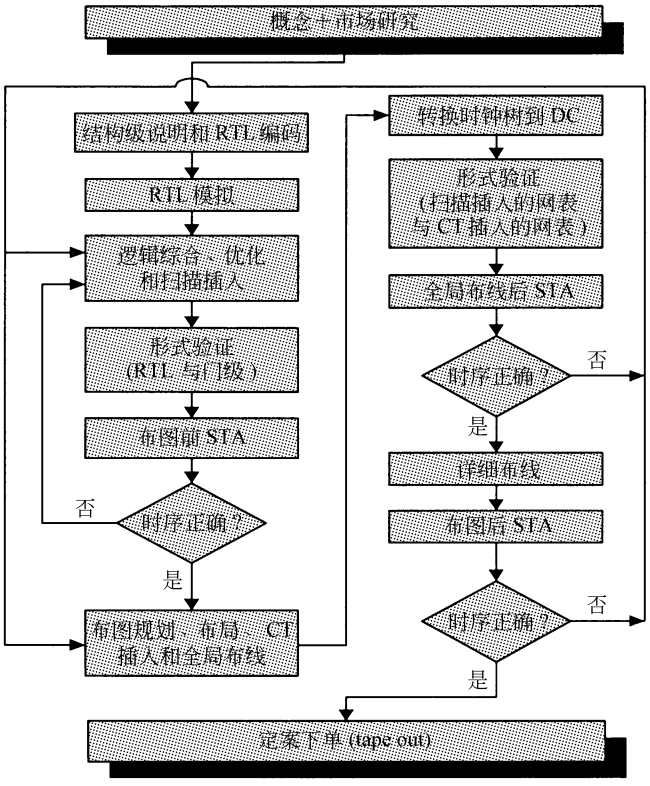

图 1-1 传统的 ASIC 设计流程

1. 结构及电学特性规范。
2. HDL 中的 RTL 编码。
3. 为包含存储单元的设计插入 DFT memory BIST。
4. 为验证设计功能，进行详尽的动态仿真。
5. 设计环境设置，包括将使用的工艺库及其他环境属性。
6. 使用 Design Compiler 对具有扫描插入（和可选择的 JTAG）的设计进行约束和综合设计。
7. 使用 Design Compiler 的内建静态时序分析机进行模块级静态时序分析。
8. 设计的形式验证，使用 Formality 将 RTL 和综合后的网表进行对比。
9. 使用 PrimeTime 进行整个设计布图前的静态时序分析。
10. 对布图工具进行时序约束的前标注。

11. 具有时序驱动单元布局、时钟树插入和全局布线的初始布局划分。

12. 将时钟树转换到驻留在 Design Compiler 中的原始设计（网表）。

13. 在 Design Compiler 中进行设计的布局优化。

14. 使用 Formality 在综合网表和时钟树插入的网表之间进行形式验证。

15. 在全局布线后（第 11 步）从版图提取估计的延时。

16. 从全局布线得到的估计时间数据反标注到 PrimeTime。

17. 使用在全局布线后提取的估计延时数据在 PrimeTime 中进行静态时序分析。

18. 设计的详细布局。

19. 提取来自详细布局设计的实际时间延迟。

20. 实际提取时间数据反标注到 PrimeTime。

21. 使用 PrimeTime 进行布图后的静态时序分析。

22. 布图后的门级功能仿真（如果需要）。

23. 在 LVS 和 DRC 验证之后定案下单（tape out）。

图 1-1 说明了上面讨论的典型的 ASIC 设计流程。缩略词 STA 和 CT 分别表示静态时序分析和时钟树，DC 表示 Design Compiler。

1.1.1　规范和 RTL 编码

芯片设计起始于由市场得出的想法概念，由这些想法进而得出结构和电学特性规范。结构规范定义了芯片的功能并划分为一些能够处理的模块，而电学特性规范通过时序信息定义模块之间的关系。

下一阶段涉及这些规范的实现。过去是利用来自单元库的元件，由人工来完成电路图，这一过程耗时并且不利于设计复用。为了克服这个问题，开发了硬件描述语言 （HDL）。正如它的名称，使用 HDL 可以对设计的功能进行编码。目前有两种常用的硬件描述语言：Verilog 和 VHDL，两种语言具有相同的功能，并都有各自的优缺点。

设计可以用三个抽象层次来表示：行为级、寄存器传输级（RTL）和结构级。行为级编码是在一个较高层次上的抽象，它主要用来将结构规范转换为一个可以仿真的代码，行为编码是完成设计目标的最基本方法。而 RTL 编码描述并推断结构元件和它们之间的连接，这类

编码用来描述设计的功能并且可综合成一个结构网表，这个网表由目标库中的元件和它们相应的连接组成，非常类似于电路图的实现方法。

不论是用 Verilog 还是 VHDL 或是两者混用，设计都是使用 RTL 编码。如果需要，它也能被划分为许多小的模块来形成一个层次，由一个顶层模块连接所有的下层模块。

注：Synopsys 最近引入了 Behavior Compiler，因此有能力进行综合行为级编码。由于这是与本书无关的主题，所以本书只讨论与 RTL 有关的综合。

1.1.2 动态仿真

下一步是通过仿真 RTL 代码以检查设计的功能。所有目前可得到的仿真器都能够仿真行为级及 RTL 级编码。此外，它们也用来仿真映射后的门级设计。

图 1-2 描述一个准备仿真的由测试基环绕的分块设计。这一个测试基通常用行为级 HDL 描述而实际的设计使用 RTL 编码。

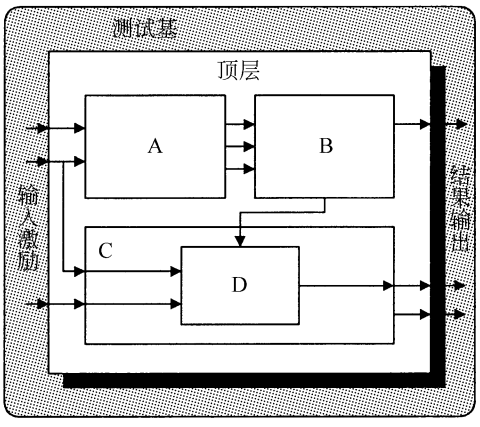

图 1-2 层次设计范例

尽管市场上有一些仿真器有能力仿真一个混合 HDL 设计，但通常仿真器还是只能识别一种语言（Verilog 或 VHDL）。

测试基的目的是为设计提供必需的激励。测试执行的次数和测试基的性能对完成整个设计测试是重要的，这就是为什么完善的测试基对设计极其重要。在 RTL 的仿真期间，不考虑元件（或门）时序。

因此，为了最小化 RTL 仿真和综合后门级仿真之间的差异，通常时序单元的 RTL 源码包含延迟。

1.1.3　约束、综合和扫描插入

在过去很长的一段时间里，硬件描述语言用来进行逻辑验证，设计者需要手工将 HDL 转换为电路图并描述元件间互连来产生一个门级网表。由于综合工具的出现，已经不用手工完成这项工作，工具已经取代了它并且可以完成 RTL 级到门级网表的转换，这一过程被称为综合。

Synopsys 的 DC 是目前在 ASIC 工业中最流行的综合工具和实际标准。

综合设计是一个迭代过程，从为设计中的每个模块定义时序约束开始，这些时序约束定义了每个信号与某个特定模块的时钟输入的相互联系。除了约束外，还需要定义综合环境的文件，这个环境文件详细说明了工艺单元库和 DC 在综合过程中使用的其他相关信息。

DC 读取设计的 RTL 代码并且使用时序约束，综合 RTL 代码到结构级，从而产生一个映射后的门级网表，这个过程如图 1-3 所示。

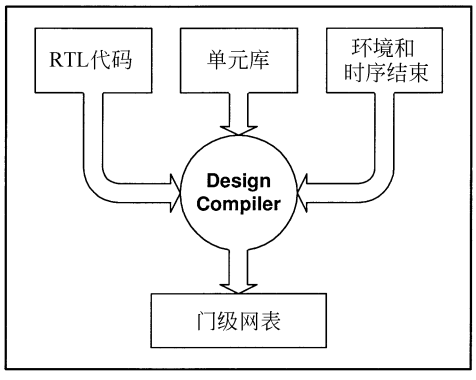

图 1-3　Design Compiler 的输入和输出

通常对于小模块的设计，DC 用内置的静态时序分析来报告综合后设计的时序信息。DC 尽可能地优化设计以满足指定的时序约束。如果时序需求不满足，就必须做进一步的工作。

目前大多数的设计都结合了可测试性设计（DFT）逻辑以便在芯片制造后用来测试它们的功能。DFT 包括逻辑和存储 BIST（内置自

测试)、扫描逻辑和边界扫描逻辑（JTAG）等。

基于控制逻辑并且在综合前合并到设计中的可综合 RTL 构成逻辑存储 BIST。目前在市场上有一些可以用来产生 BIST 控制和周边逻辑的工具，遗憾的是，Synopsys 没有提供这方面的工具。

使用 DC 的测试预编译特性可以执行扫描插入，在扫描链链接它们之前，这一个过程可直接将 RTL 映射到扫描触发器（scan-flop）。使用这一特性的优点是它可以使DC在综合时考虑扫描触发器的时序能力。与对应的非扫描触发器（或正常触发器）相比，扫描触发器可产生不同的延时，因此这项技术是重要的。

JTAG 或边界扫描主要用于测试带有芯片的板连接，JTAG 控制器和周边的逻辑也可以直接由 DC 产生。

1.1.4 形式验证

形式验证的概念对 ASIC 设计领域是相当新的。形式验证技术使用数学方法来确认一个设计，无需考虑工艺因素，如时序和物理效应的影响，它们通过与参考设计对比来检查一个设计的逻辑功能。

许多 EDA 工具厂商都开发了形式验证工具，然而直到最近 Synopsys 才向市场推出形式验证工具，称为 Formality。

形式验证方法和动态仿真之间的主要不同是，形式验证技术通过证明两个设计的结构和功能是逻辑等价的来验证设计；动态仿真方法只能检查那些敏感路径，所以不可能找出其他出现的问题。此外，与动态仿真相比，形式验证方法所需的运行时间是可以忽略不计的。

在设计流程中，形式验证的目标是要验证 RTL 与 RTL、门级网表与 RTL 代码或两个门级网表之间的对应关系是否正确。

RTL 对 RTL 的验证是用来确认新的 RTL 与原来的 RTL 在功能上是否一致。这通常是为了适应因增加的附加性能而需经常修改设计的情况。当这些特性增加到 RTL 源的时候，总是有改变原有正确功能的危险。为了避免这种情况，可以在原来的 RTL 和新的 RTL 之间执行形式验证，以检查原有功能的有效性。

RTL 对门级的验证是用来确定 DC 综合的逻辑是正确的。由于通过动态仿真来验证 RTL 功能正确，所以在 RTL 和有扫描插入的门级网表之间做形式验证，能保证门级也有相同的功能。在这种情况下，如果我们要使用动态仿真方法验证门级，就会花费较长的时间 （数

天和数星期，取决于设计的大小）来验证设计。与动态仿真比较，形式 RTL 方法只用几个小时就可完成一个类似的验证。

最后是门级网表对门级网表的验证。这也是验证过程的一个重要的步骤，因为它主要用来确认版图输入信息与版图输出信息。显然从版图得出的是时钟树插入的网表（平面或层次化的）。这意味着进入布图工具的原来的网表已被修改了。形式验证技术用来确认修改后的网表与原来的网表是逻辑等价的。

1.1.5　使用 PrimeTime 进行静态时序分析

正如前文提到的，DC 可以作模块级静态时序分析。虽然使用上述方法可以完成芯片级静态时序分析，但是这里推荐使用 PrimeTime。PrimeTime 是由 Synopsys 提供的、可独立运行的、能够提高品质的静态时序分析工具，它能够非常快地完成整个芯片级设计的静态时序分析。它提供一个 Tcl 接口，该接口为设计分析和调试提供了一个强大的环境。

从某种程度上说，在整个 ASIC 设计过程中静态时序分析是最重要的步骤。静态时序分析允许用户详细分析设计的所有关键路径并且给出一个有条理的报告。此外，该报告也可包含其他调试信息，如扇出能力或每个线网的容性负载。

对布图（layout）前后的门级网表进行静态时序分析。在布图前，PrimeTime 使用由库指定的线载模型估计线网延时，在这一过程中，先前输入到 DC 的时序约束同样也输入到 PrimeTime 中并详细说明主要的输入输出信号和时钟的关系。如果对于所有关键路径的时序是可接受的，则由 PrimeTime 或 DC 可以得到一个约束文件，目的是为了预标注到布图工具。这个约束文件以 SDF 格式详细描述在布图工具中使用的每个逻辑组之间的时序，以便完成单元的时序驱动布局。

在布图后，实际提取的延迟被反标注到 PrimeTime 以提供真实的延迟计算，这些延迟由连线电容和互连 RC 延迟所组成。

与综合类似，静态时序分析也是一个迭代过程，它与芯片布局布线（placement and routing）的联系非常紧密，这个操作通常需要执行许多次才能满足时序需求。

1.1.6 布局、布线和验证

正如本节标题所示，布图工具完成布局和布线。完成这一步骤有许多方法，本节只讨论与综合相关的内容。

布图规划（floorplan）和布局的质量比实际的布线更重要。最佳的单元布局，不但加速最终的布线，而且在满足时序约束和减少阻塞方面也产生非常好的效果。如前所述，约束文件用来进行时序驱动布局。时序驱动布局方法能够使布图工具根据单元之间的时序关键程度放置单元。

在单元布局后，时钟树通过布图工具插入设计。时钟树插入是可选择的并且只依赖设计需求和用户的偏爱。用户可选择使用较传统的方法对时钟网络进行布线，例如，为了要减少总时间延迟和时钟的倾斜使用 fishbone/spine 结构的时钟网络。当工艺尺寸缩小，由于互连线电阻的增加（从而 RC 延迟），spine 方法实现变得更困难。因此本节（和本书）重点放在时钟树综合方法方面。

在这个阶段，一个附加的步骤对完成时钟树插入是必需的。如上所述，在单元布局后，布图工具将时钟树插入设计，因此，从 DC 产生的最初网表（并且输入到布图工具）缺少时钟树信息（整个时钟树网络，包括缓冲器和线网），所以，时钟树一定要插入到原有的网表中并进行形式验证。一些布图工具通过提供直接的接口给 DC 来完成这一步骤。第 9 章将介绍一些方法，包括传统的和非传统的实现方法。为了简化，假设已经将时钟树插入到原有的网表。

布图工具完成布线通常由两步组成：全局布线和详细布线。在布局后，设计进行全局布线以确定布局的质量和提供估计延迟与在详细布线后实际延时值的近似程度。如果单元布局不是最佳的，与单元布局相比，完成全局布线将花费比较长的时间。不好的布局也会影响整个设计的时序。因此，为使布图综合迭代的次数最小并且改进布局质量，在全局布线后，从版图提取时序信息。尽管这些延迟数据不如在详细布线后提取的数据准确，但它们确实提供了布线后的时序信息。这些估计的延迟被反标注到 PrimeTime 进行分析，并且只有当时序关系满足后，余下的进程才被允许执行。

详细布线是布图工具进行的最后步骤，在详细布线完成后，提取

芯片的实际时间延迟并且输入到 PrimeTime 进行分析。

这些步骤是反复的并且依赖于该设计的时序余量。如果设计不能满足时序需求，在布图后的优化要在进行布图另一次迭代之前运行。如果设计通过静态时序分析，在定案下单（tape-out）前它还要进行 LVS（版图对原理图）和 DRC（设计规则检查）。

值得注意的是，上面讨论的所有步骤也能应用于层次化的布局布线，换言之，在对子模块布局以形成最终版图之前，可以对每个子模块重复上述步骤。

1.1.7　工程改变命令

这个步骤是正常设计流程的一个例外，不要与通常的设计周期混淆。因此，这一个步骤在后面章节将不再加以介绍。

许多设计者认为，工程改变命令（ECO）是在 ASIC 设计流程的每个最后阶段对网表进行必需的改变。例如，当在最后阶段（比如定案下单）后，在设计中遇到的硬件隐藏错误并且需要通过对设计的一个小部分进行再布线来完成一个金属掩模改变时，就运行 ECO。

由于 ECO 的执行结果只对芯片的小部分有作用，为了避免干扰芯片其余部分的布局和布线，因此保留芯片其余部分的时序，只有受影响的部分被修改。可以通过只影响芯片包含的备用的门电路或只对一金属层布线来完成此项操作。这一过程称为金属掩模改变。

通常，这个过程只在对整个芯片（或一个模块，如果做层次化的布局和布线）的修改少于 10%时才执行。如果修改需要超过 10%，那么最好重复整个的过程和重新对整个芯片（或模块）布线。

最新版的 DC 包含 ECO 编译器。它利用数学算法（也应用于形式验证技术），自动地实现需求的变化。利用 ECO 编译器提供设计者手动修改插入网表的方法，可将芯片的完成时间减到最小。

一些布图工具已经将 ECO 算法合并到它们的工具里。布图工具的优点就是不受设计的层次化边界限制。同样，布图工具也受益于知道备用的单元（通常由设计者引入到设计中）放置位置，因此能以最接近备用单元的位置为目标以完成所需的 ECO 改变并获得最小化的布线。

1.2 Physical Compiler 流程

由于半导体几何尺寸的缩小，基于线载模型的综合结果变得很不准确和不可预知。由 Synopsys 开发的一个新工具 Physical Compiler，通过在一个公共引擎中集成综合和布局解决了上述问题，从而可以避免基于线载模型的延迟计算。

Physical Compiler 基本的设计流程一般包含下面列出的一些步骤。图 1-4 所示为与下面描述的设计流程相关的流程图。由于一些步骤在传统流程和基于 Physical Compiler 的流程中都存在，所以这里只说明与 Physical Compiler 流程相关的一些步骤。

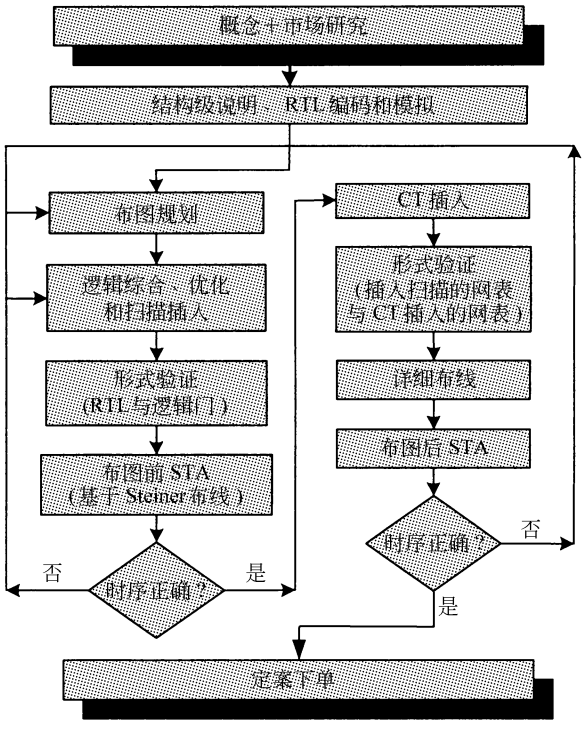

图 1-4 Physical Compiler 流程

1. 设计环境设置，包括要使用的工艺库和物理库以及其他的环境属性。

2. 布图规划设计。

3. 使用 Physical Compiler 进行约束、综合（使用扫描插入）及设计布局。

4. 使用 PrimeTime 进行布图前静态时序分析（基于布局而不是线载模型的时间延迟数据）。

5. 使用 Formality 对设计的 RTL 与综合后的网表进行形式验证。

6. 将网表和布局信息移植到布图工具上。

7. 使用布图工具在设计中插入时钟树。

8. 在插入时钟树的网表与原有的扫描插入的网表之间进行形式验证。

9. 使用布图工具进行详细布线。

10. 从详细布线设计提取实际时间延迟。

11. 将实际提取的数据反标注到 PrimeTime。

12. 使用 PrimeTime 进行布图后的静态时序分析。

13. 利用布图后的时序进行设计的门级功能仿真（如果想要）。

14. 在 LVS 和 DRC 验证后定案下单（tape out）。

1.2.1　物理综合

传统的综合方法是以线载模型为基础的，而线载模型的基本性质是以它们的扇出为基础建立的。换言之，单元的延迟计算是基于单元驱动的扇出数目进行的，这种方法适用于较大的几何尺寸（大于 0.35μm），而不适合于较小的几何尺寸，基于扇出进行延时计算时，线的电阻决定了单元延迟时间，因而是不可靠的并且完全不可预知。

为了解决上述问题，Synopsys 近来以 Physical Compiler（因而称为 PhyC）的形式引入了物理综合概念。PhyC 除保留了原有 DC 的功能外还增加了一些功能，所以 PhyC 是 DC 的一个超集。

PhyC 不使用线载模型，并将以扇出为基础的延迟计算改为以布局为基础的延迟计算。换言之，综合和优化是以单元布局为基础的，并且将扫描链重排功能包含在目前的版本 PhyC（2000.11）中，它确实是极其强大的有用工具。

图 1-4 用一种非常一般的形式来描述这种实现方法。PhyC 有两种模式：RTL 到门级布局（rtl2pg）或门到门级布局（g2pg）。对于前一种模式，PhyC 的输入是 RTL、布图规划信息及包含逻辑库和物理

库的必要设置。PhyC 的输出是 PDEF3.0 格式的结构网表和布局后的门信息。第二种模式（g2pg）在基于布图规划信息的条件下，对一个已存在的门级网表进行优化。在这种情况下，PhyC 的输入替代 RTL 的门级网表，而其余的设置和输入输出文件保持一致。

值得注意的是，目前的 PhyC（2000.11 版）版本没有综合时钟树的能力。Synopsys 最近发布了一个加入到 PhyC 中的有效的时钟树编译工具。没有这个工具的用户只能使用他们的布图工具将时钟树插入到设计数据库中。

进行有效的综合需要完成许多步骤。这些将在后面的章节讨论。然而目前出于解释设计流程的目的，上述论述已足够了。

1.3 小结

本章对包括由超深亚微米（VDSM）技术组成的最新工具和工艺的 ASIC 设计流程进行了介绍。设计流程从定义规范开始，以物理版图结束。重点放在逻辑和物理综合等相关主题。

物理综合是本章引入的一个新概念，它可应用于设计流程中以缩短芯片的设计周期。物理综合的重要性是它能够得到一个较好的对延迟的估计并且能缩短上市时间。

第 2 章
入门指南 静态时序分析与综合

本章针对 Synopsys 工具的初级用户和高级用户。建议以前没有使用 Synopsys 工具的初学者先跳过这一章，在阅读了本书其他部分后再返回这一章。有少量综合经验的初级用户可把本章当作学习使用 Synopsys 工具进行 ASIC 设计流程的起点，高级用户可将本章作为参考。

本章很少甚至没有解释 Synopsys 命令（将在以后章节中进行介绍）。其重点是介绍第 1 章中所描述的以 Synopsys 综合为核心的 ASIC 设计流程的实际应用。这有助于读者将理论概念同实际应用联系起来。

为了描述基于传统和 Physical Compiler 的流程，保留了同前者相关的所有的脚本（将命令改变为 Tcl 格式）。此外还有一个小节介绍了 Physical compiler 流程，用户可挑选最适合自己设计需求的流程。

虽然第 1 章为了强调形式验证方法而跳过门级仿真，但是许多设计人员不愿采用前一种方法。因此，本章也介绍了从 DC 生成用于仿真的 SDF，除了使用 Formality 的形式验证的应用外，本章还介绍了使用 Primetime（PT）的静态时序分析。

可用任何方法进行综合和优化，这只取决于你最喜欢用的和最方便使用的方法。本章采用了最为 Synopsys 用户们广泛使用的一种方法，你可以相当容易地掌握这种方法以满足自己的要求。

为了清晰容易地进行解释，所有的示例和脚本都采用自底向上的编译方法（稍后介绍），这同本章中的综合过程相关。还必须注意的

是，整个 ASIC 流程是反复迭代的过程，用户不应当认为本章介绍的过程满足所有设计。后续章节将详细讨论每个论题，用户可加以调整以适应自己的设计和方法。

2.1 设计示例

学习这个主题的最好方法就是用一个设计示例走一遍整个设计流程。下面所示的是用 Verilog HDL 包含一个层次的 tap 控制器的设计。

tap_controller.v
- *tap_bypass.v*
- *tap_instruction.v*
- *tap_state.v*

顶层设计为 *tap_controller*，其中例化了三个模块，分别为 *tap_bypass*、*tap_instruction* 和 *tap_state*。这一设计包括一个 30MHz 的时钟“tck”和一个复位“trst”。设计时序规范指定所有与“tck”相关的输入信号所需的建立时间为 10ns，而保持时间为 0ns。此外，所有输出信号必须比时钟延迟 10ns。

用于这一设计的工艺为 0.25 微米。由于工艺偏差，为了得到更加精确的结果，使用了两个分别对应最差情况和最佳情况参数的 Synopsys 标准单元工艺库。这两个库分别为 *ex25_worst.db* 和 *ex25_best.db*，还有一个相应的包含电路图表示的符号库 *ex25.sdb*。在 *ex25_worst.db* 库中定义工作条件名为 WORST，而在 *ex25_best.db* 库中定义工作条件名为 BEST。

假设设计的功能已经通过了 RTL 级动态仿真验证。

2.2 初始设置

下一步是综合设计，也就是说要把设计映射到工艺库中的门。在我们开始进行综合前，必须生成如下设置文件：

a）.synopsys_dc.setup 文件，用于 DC 和 PhyC。

b）.synopsys_pt.setup 文件，用于 PT。

第一个文件是用于逻辑综合和物理综合的 DC 和 PhyC 设置文件，而第二个文件同 PT 相关并且定义了静态时序分析所要求的设置。

假设库被保存在目录—/usr/golden/library/std_cells/中，创建具有如下内容的两个文件：

DC & PhyC .synopsys_dc.setup 文件

```
set search_path          [list . /usr/golden/library/std_cells]
set target_library       [list ex25_worst.db]
set link_library         [list {*} ex25_worst.db ex25_best.db]
set symbol_library       [list ex25.sdb]
set physical_library     [list ex25_worst.pdb]

define_name_rules BORG -allowed {A-Za-z0-9_}          \
      -first_restricted "_" -last_restricted "_"      \
      -max_length 30                                  \
      -map {{"*cell*","mycell"},{"*-return","myreturn"}}

set bus_naming_style                    %s[%d]
set verilogout_no_tri                   true
set verilogout_show_unconnected_pins    true
set test_default_scan_style             multiplexed_flip_flop
```

PT .synopsys_pt.setup 文件

```
set search_path          [list . /usr/golden/library/std_cells]
set link_library         [list {*} ex25_worst.db ex25_best.db]
```

2.3　传统流程

下面的步骤描述了传统流程。这里 DC 用于逻辑综合，而布图工具处理包括布局和布线等其他后端问题。

2.3.1　布图前的步骤

以下几个小节介绍在布图前的阶段涉及到的步骤，包括带扫描插

入的逻辑综合、静态时序分析、用于执行门级功能仿真的 SDF 生成以及最终的 RTL 源码与综合后网表之间形式验证的全过程。

2.3.1.1 综合

布图前逻辑综合包括对最大的建立时间优化设计、采用统计的线载模型及 *ex25_worst.db* 工艺库中的最差情况工作条件。为了最大化建立时间，通过定义建立时间的时钟不确定性来约束设计。一般说来，为了最小化综合与布图之间的迭代次数，10％过度约束通常是足够的。

在最初的综合后，如果检测到保持时间（hold-time）违例，它们应该在布图前加以改正。这也有助于减少综合与布图之间的反复。然而，可取的方法是在布图后用反标注的真实延迟来修正次要的保持时间违例。

在本指南中，我们假定存在次要的保持时间违例，因而这些违例可在布图后优化中加以改正。修正保持时间违例涉及将从版图提取的延迟反标注到 DC 中。此外，保持时间修正需要利用 *ex25_best.db* 库中的最佳情况工作条件的用法。

子模块的通用综合脚本

```
set active_design tap_bypass

analyze –format verilog $active_design.v
elaborate $active_design

current_design $active_design
link

uniquify

set_wire_load_model –name SMALL
set_wire_load_mode top
set_operating_conditions WORST

create_clock –period 33 –waveform [list 0 16.5] tck
set_clock_latency 2.0 [get_clocks tck]
```

```
set_clock_uncertainty –setup 3.0 [get_clocks tck]
set_clock_transition 0.1 [get_clocks tck]
set_dont_touch_network [list tck trst]

set_driving_cell –cell BUFF1X –pin Z [all_inputs]
set_drive 0 [list tck trst]

set_input_delay 20.0 –clock tck –max [all_inputs]
set_output_delay 10.0 –clock tck –max [all_outputs]

set_max_area 0

set_fix_multiple_port_nets –buffer_constants –all
compile –scan

check_test

remove_unconnected_ports [find –hierarchy cell {"*"}]

change_names –h –rules BORG

set_dont_touch current_design

write –hierarchy –output $active_design.db
write –format verilog –hierarchy        \
                        –output $active_design.sv
```

上述脚本包括用户定义的变量 *active_design*，它对要综合的模块进行命名。这一变量应用于整个脚本，从而使得脚本的其余部分通用。通过将 *active_design* 重新定义到其他子模块（*tap_instruction* 和 *tap_state*），可使同样的脚本用于综合这些子模块。用户可将相同的概念应用到时钟名、时钟周期等以参数化脚本。

假设已经成功综合了三个子模块，命名为 *tap_bypass*,、*tap_instruction* 和 *tap_state*。这一综合过程没有将扫描链串起来，只是将触发器直接映射到扫描触发器。除了在读取 *tap_controller.v* 文件之前必须包括子模块的已映射的“db”文件，我们可以将相同的综合脚本应用于顶层综合，此外，还需要进行扫描插入来把扫描链串起来。为更好地建模互连线，线载模型需要改变为 enclosed。由于子模块

包括 dont_touch 属性，顶层综合不会跨越层次边界进行优化，并可能会违反设计规则约束。为了去除这些违例，必须删除子模块的 dont_touch 属性以重新进行综合/优化。

顶层 DFT 扫描插入是从子模块中除去 dont_touch 属性的另一个原因。这是因为如果子模块包含 dont_touch 属性，就不能在顶层进行 DFT 扫描插入。以下脚本通过执行带有扫描使能的初始综合来说明此过程，在再次编译（compile -only_design_rule）设计前，去除了所有子模块的 dont_touch 属性。

顶层的综合脚本

```
set active_design tap_controller

set sub_modules {tap_bypass tap_instruction tap_state}

foreach module $sub_modules{
        set syn_db $module.db
        read_db syn_db
}

analyze –format verilog $active_design.v
elaborate $active_design

current_design $active_design
link

uniquify

set_wire_load_model –name LARGE
set_wire_load_mode enclosed
set_operating_conditions WORST

create_clock –period 33 –waveform [list 0 16.5] tck
set_clock_latency 2.0 [get_clocks tck]
set_clock_uncertainty –setup 3.0 [get_clocks tck]
set_clock_transition 0.1 [get_clocks tck]
set_dont_touch_network [list tck trst]

set_driving_cell –cell BUFF1X –pin Z [all_inputs]
```

```
set_drive 0 [list tck trst]

set_input_delay 20.0 –clock tck –max [all_inputs]
set_output_delay 10.0 –clock tck –max [all_outputs]

set_max_area 0

set_fix_multiple_port_nets –all –buffer_constants
compile –scan

remove_attribute [find –hierarchy design {"*"}] dont_touch

current_design $active_design
uniquify

check_test
create_test_patterns –sample 10
preview_scan
insert_scan
check_test

compile –noly_design_rule

remove_unconnected_ports [find –hierarchy cell {"*"}]
change_names –hierarchy –rules BORG

set_dont_touch current_design

write –hierarchy –output $active_design.db
write –format verilog –hierarchy        \
                        –output $active_design.sv
```

2.3.1.2　使用 PrimeTime 进行静态时序分析

在综合成功后，对得到的网表必须进行分析以检查时序违例。时序违例可包括建立和/或保持时间违例。

设计在综合过程中以最大化建立时间（setup-time）为重点，因而即使存在建立时间违例，也很少遇到。然而，保持时间违例通常会出现在这一阶段，这是由于数据到达时序单元输入的速度太快（在时

序单元锁存数据前，其值改变），从而违反了保持时间的要求。

如果设计不满足建立时间要求，你别无选择，只能重新综合设计，并以违例路径为目标进行进一步地优化。这涉及到组合违例路径或对具有违例的整个子模块加紧约束。然而如果设计不满足保持时间要求，可在布图阶段前也可推迟到布图后再修正这些违例。许多设计人员对次要的保持时间违例倾向于采用后一种方法（这里也采用此种方法），因为布图前综合和时序分析采用的是统计线载模型，并且在布图前修正保持时间违例可导致同一路径在布图后建立时间违例。必须注意的是，总的保持时间违例应在布图前阶段加以修正，以减少布图后的保持时间修正的数目。

布图前建立时间分析的 PT 脚本

```
set active_design tap_controller

read_db –netlist_only $active_design.db

current_design $active_design

set_wire_load_model –name large
set_wire_load_mode top

set_operating_conditions WORST

set_load 50.0 [all_outputs]
set_driving_cell –cell BUFF1X –pin Z [all_inputs]

create_clock –period 33 –waveform [list 0 16.5] tck
set_clock_latency 2.0 [get_clocks tck]
set_clock_transition 0.2 [get_clocks tck]
set_clock_uncertainty 3.0 –setup [get_clocks tck]

set_input_delay 20.0 –clock tck [all_inputs]
set_output_delay 10.0 –clock tck [all_outputs]

report_constraint –all_violators

report_timing –to [all_registers –data_pins]
report_timing –to [all_outputs]
```