FPGA 设计流程指南

前言

本部门所承担的 FPGA 设计任务主要是两方面的作用：系统的原型实现和 ASIC 的原型验证。编写本流程的目的是：

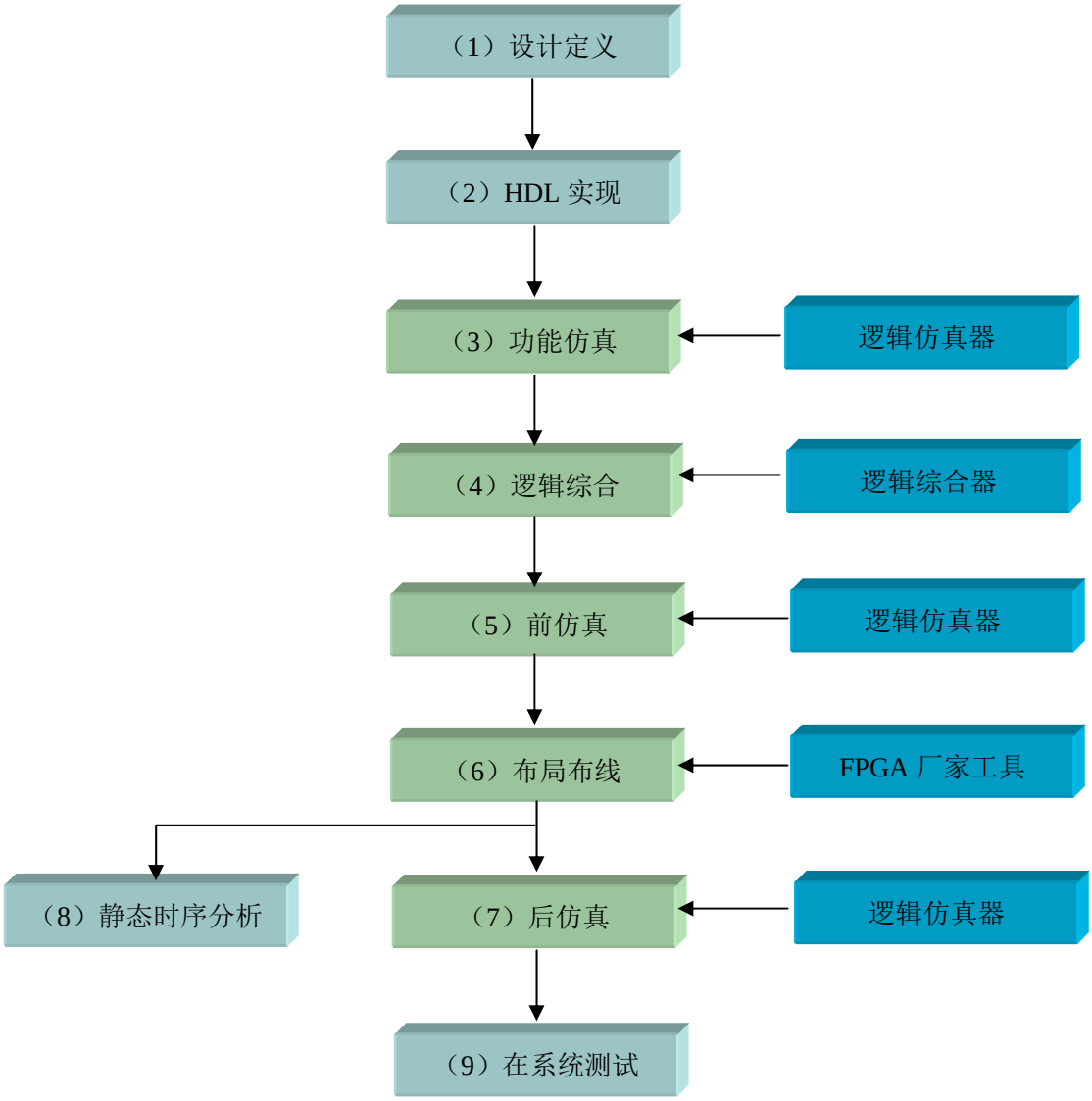
- 在于规范整个设计流程，实现开发的合理性、一致性、高效性。
- 形成风格良好和完整的文档。
- 实现在 FPGA 不同厂家之间以及从 FPGA 到 ASIC 的顺利移植。
- 便于新员工快速掌握本部门 FPGA 的设计流程。

由于目前所用到的 FPGA 器件以 Altera 的为主，所以下面的例子也以 Altera 为例，工具组合为 modelsim + LeonardoSpectrum/FPGACompilerII + Quartus，但原则和方法对于其他厂家和工具也是基本适用的。

目 录

1. 基于HDL的FPGA设计流程概述

1.1 设计流程图

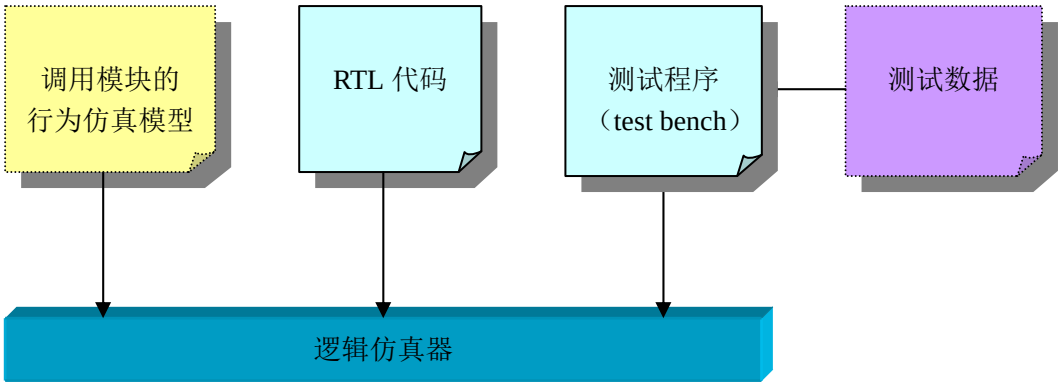


说明：

- 逻辑仿真器主要指 modelsim，Verilog-XL 等。
- 逻辑综合器主要指 LeonardoSpectrum、Synplify、FPGA Express/FPGA Compiler 等。
- FPGA 厂家工具指的是如 Altera 的 Max+PlusII、QuartusII，Xilinx 的 Foundation、Alliance、ISE4.1 等。

1.2 关键步骤的实现

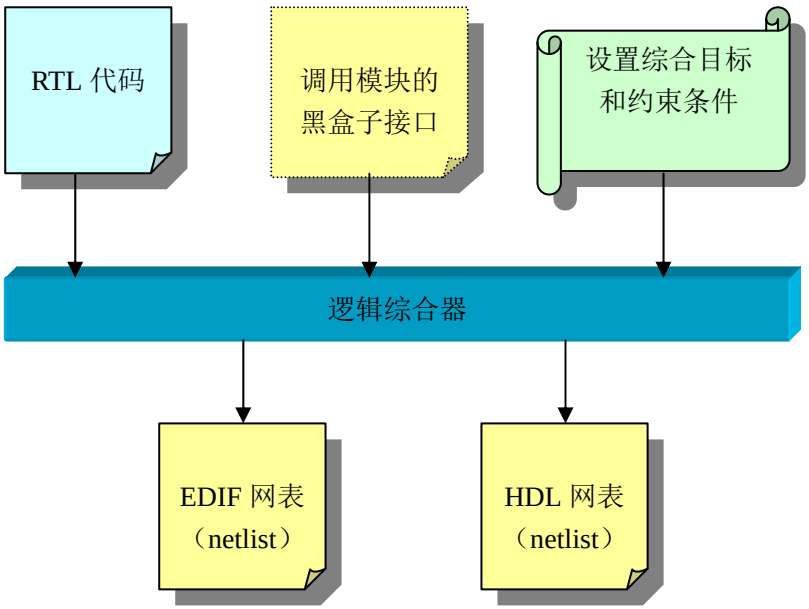
1.2.1 功能仿真



说明：

“调用模块的行为仿真模型”指的是 RTL 代码中引用的由厂家提供的宏模块/IP，如 Altera 提供的 LPM 库中的乘法器、存储器等部件的行为模型。

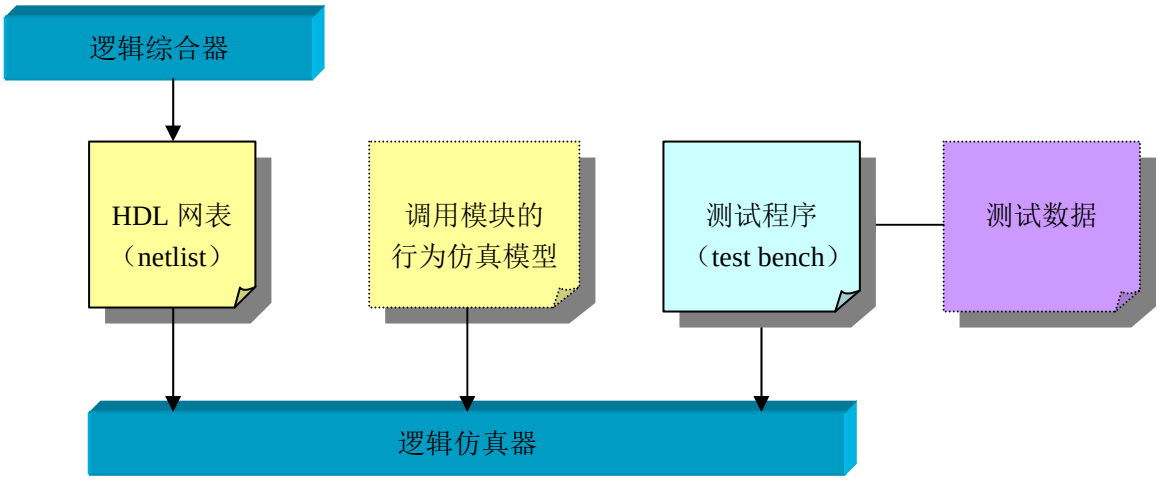
1.2.2 逻辑综合



说明：

“调用模块的黑盒子接口”的导入，是由于 RTL 代码调用了一些外部模块，而这些外部模块不能被综合或无需综合，但逻辑综合器需要其接口的定义来检查逻辑并保留这些模块的接口。

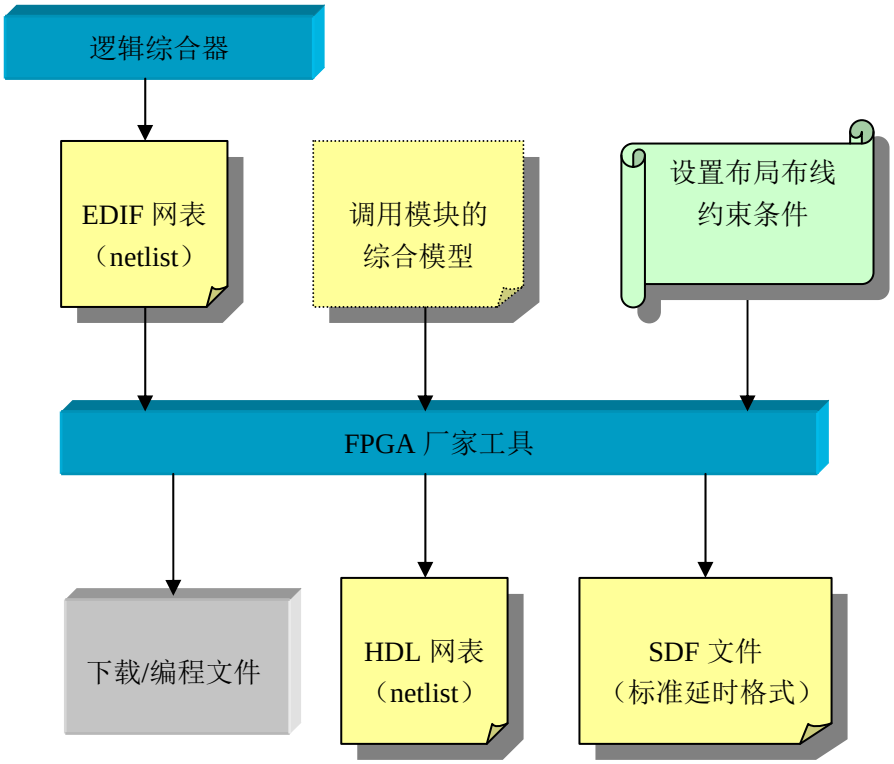
1.2.3 前仿真



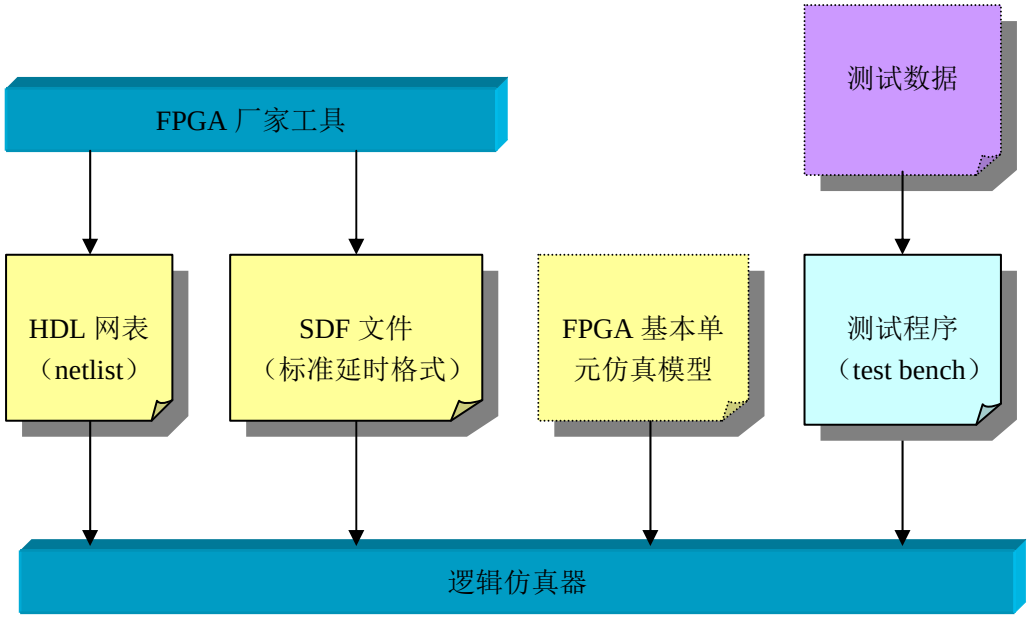
说明：

一般来说，对 FPGA 设计这一步可以跳过不做，但可用于 debug 综合有无问题。

1.2.4 布局布线



1.2.5 后仿真（时序仿真）



2. Verilog HDL设计

基于将来设计转向 ASIC 的方便，本部门的设计统一采用 **Verilog HDL**，但针对混合设计和混合仿真的趋势，所有开发人员也应能读懂 VHDL。

Verilog HDL 的学习可参考[1][2]。

2.1 编程风格（Coding Style）要求

2.1.1 文件

（1） 每个模块（module）一般应存在于单独的源文件中，通常源文件名与所包含模块名相同。

（2） 每个设计文件开头应包含如下注释内容：

- 年份及公司名称。
- 作者。
- 文件名。
- 所属项目。
- 顶层模块。
- 模块名称及其描述。
- 修改纪录。

请参考标准示例程序[3]。

2.1.2 大小写

（1） 如无特别需要，模块名和信号名一律采用小写字母。

（2） 为醒目起见，常数（`define 定义）/参数（parameter 定义）采用大写字母。

2.1.3 标识符

（1） 标识符采用传统 C 语言的命名方法，即在单词之间以“_”分开，如：max_delay、data_size 等等。

（2） 采用有意义的、能反映对象特征、作用和性质的单词命名标识符，以增强程序的可读性。

（3） 为避免标识符过于冗长，对较长单词的应当采用适当的缩写形式，如用‘buff’代替‘buffer’，‘ena’代替‘enable’，‘addr’代替‘address’等。

2.1.4 参数化设计

为了源代码的可读性和可移植性起见，不要在程序中直接写特定数值，尽可能采用`define 语句或 paramater 语句定义常数或参数。

2.1.5 空行和空格

（1） 适当地在代码的不同部分中插入空行，避免因程序拥挤不利阅读。

（2） 在表达式中插入空格，避免代码拥挤，包括：

赋值符号两边要有空格；

双目运算符两边要有空格；

单目运算符和操作数之间可没有空格，

示例如下：

```
a  <=  b;
c  <=  a  +  b;
if (a  ==  b) then ...
a  <=  ~a  &  c;
```

2.1.6 对齐和缩进

（1） 不要使用连续的空格来进行语句的对齐。

（2） 采用制表符 Tab 对语句对齐和缩进，**Tab 键采用 4 个字符宽度**，可在编辑器中设置。

（3） 各种嵌套语句尤其是 if...else 语句，必须严格的逐层缩进对齐。

2.1.7 注释

必须加入详细、清晰的注释行以增强代码的可读性和可移植性，注释内容占代码篇幅不应少于 30%。

2.1.8 参考C语言的资料

要形成良好的编程风格，有许多细节需要注意，可以参考资料[4]，虽然它是针对 C 语言的讨论，但由于 Verilog HDL 和 C 语言的形式非常近似，所以里面提到的很多原则都是可以借鉴的。

2.1.9 可视化设计方法

为提高设计效率和适应协同设计的方式，可采用可视化的设计方法，Mentor Grahpics 的 Renoir 软件提供了非常好的设计模式。

2.2 可综合设计

用 HDL 实现电路，设计人员对可综合风格的 RTL 描述的掌握不仅会影响到仿真和综合的一致性，也是逻辑综合后电路可靠性和质量好坏最主要的因素，对此应当予以充分的重视。

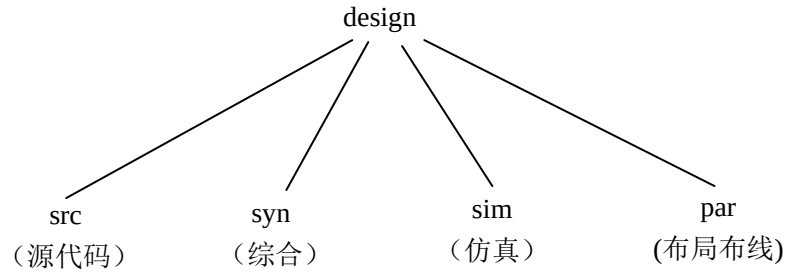
学习可综合的 HDL 请参考 [5][6] [7]。

学习设计的模块划分请参考[8]。

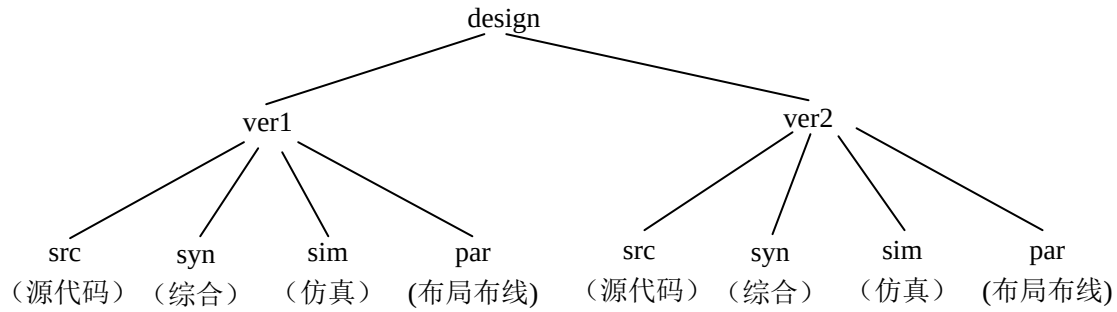
2.3 设计目录

采用合理、条理清晰的设计目录结构有助于提高设计的效率、可维护性。建议采用类似下面的目录结构：

（1）



（2）



3. 逻辑仿真

考虑到性能和易用性，首选的逻辑仿真器是 Mentor Graphics 的 modelsim。

3.1 测试程序（test bench）

测试程序对于设计功能和时序的验证有着举足轻重的影响，测试激励的完备性和真实性是关键所在，有以下原则须遵循：

（1） 测试激励输入和响应输出采集的时序应当兼顾功能仿真（无延时）和时序仿真（有延时）的情况。

（2） 对于周期较多的测试，为提高效率，尽可能采用程序语句来判断响应与标准结果是否一致，给出成功或出错标志，而不是通过观察波形来判断。

（3） 采用基于文件的测试是很好的办法，即由 matlab 或 spw 等系统工具产生测试数据，测试程序将其读入产生激励，再把响应结果写入到文件，再交给上述工具进行处理或分析。

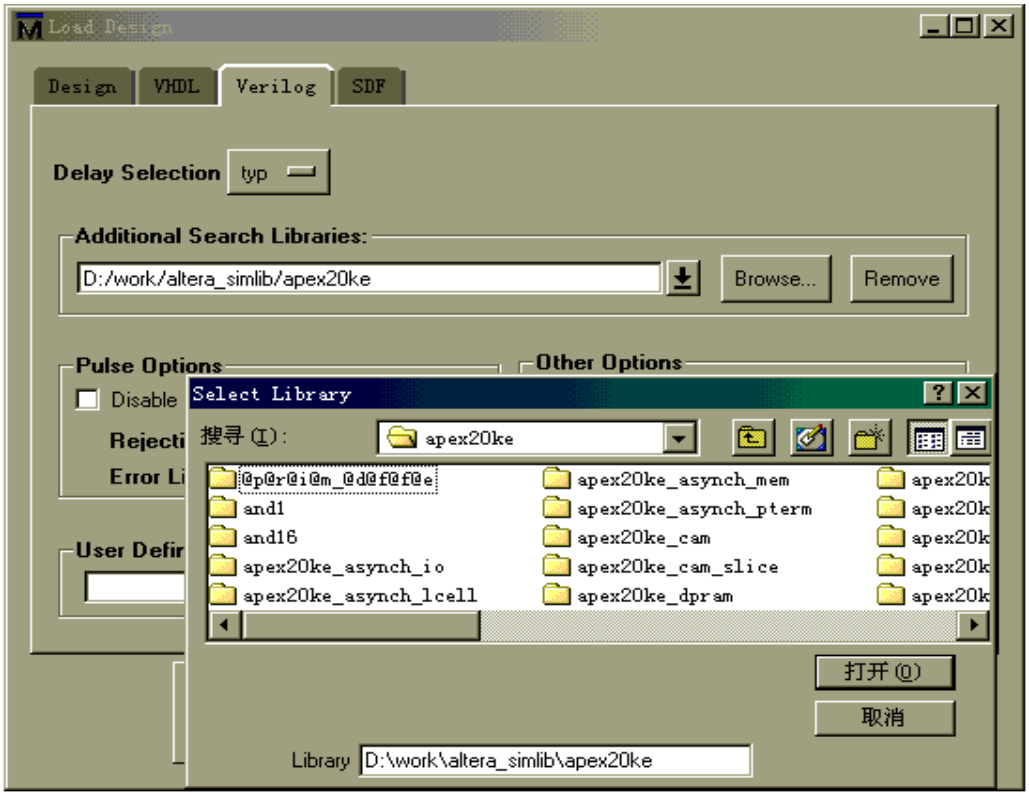
（4） 仿真器支持几乎所有的 Verilog HDL 语法，而不仅仅是常用的 RTL 的描述，应当利用这一点使测试程序尽可能简洁、清楚，篇幅长的要尽量采用 task 来描述。

3.2 使用预编译库

在进行功能仿真和后仿真时都需要某些模块的行为仿真模型和门级仿真模型，如 Altera Quartus 里的 220model.v（LPM 模块行为仿真模型）和 apex20ke_atoms.v（20KE 系列门级仿真模型），为避免在不同的设计目录中多次编译这些模型，应当采用一次编译，多次使用的方法。具体做法如下（以 20KE 门级库为例）：

1：在某个工作目录下新建一库名 apex20ke，将 apex20ke_atoms.v 编译到其中。

2：在图形界面中的 Load Design 对话框中装入仿真设计时，在 Verilog 标签下指定预编译库的完整路径。（见下图）



4. 逻辑综合

目前可用的 FPGA 综合工具有 Mentor Graphics 的 LeonardoSpectrum，Synplicity 的 Synplify 和 Synopsys 的 FPGA CompilerII/FPGA Express，LeonardoSpectrum 由于性能和速度最好，成为我们首选的综合器，FPGA CompilerII/FPGA Express 由于可以和 Design Compiler 代码兼容也可用。见参考[9]

4.1 逻辑综合的一些原则

HDL 代码综合后电路质量的好坏主要取决于三个方面：RTL 实现是否合理、对厂家器件特点的理解和对综合器掌握的程度。参考[10]中有比较全面的讨论。

4.1.1 关于LeonardoSpectrum

LeonardoSpectrum 对综合的控制能力比较强，但使用也略为复杂，故需要在使用前尽量熟悉其功能，才能取得较好的综合结果。

当出现综合结果不能满足约束条件时，不要急于修改设计源文件，应当通过综合器提供的时序和面积分析命令找出关键所在，然后更改综合控制或修改代码。

在 LeonardoSpectrum 2000.1b 以前的版本输出的 .v 网表都不能用于仿真。

4.1.1 大规模设计的综合

- 分块综合

当设计规模很大时，综合也会耗费很多时间。如果设计只更改某个模块时，可以分块综合。如有设计 top.v 包含 a.v 和 b.v 两个模块，当只修改 a.v 的话，可以先单独综合 b.v，输出其网表 b.edf，编写一个 b 模块的黑盒子接口 b_syn.v，每次修改 a.v 后只综合 top.v、a.v、b_syn.v，将综合后的网表和 b.edf 送去布线，可以节约综合 b 模块的时间。

- 采用脚本命令

当设计规模比较大时，综合控制也许会比较复杂，可以考虑采用脚本控制文件的方式进行综合控制，modelsim、LeonardoSpectrum 和 Quartus 都支持 TCL（Tool Command Language）语言，采用脚本控制可以提供比图形界面更灵活和更方便的控制手段。

4.1.3 必须重视工具产生的警告信息

综合工具对设计进行处理可能会产生各种警告信息，有些是可以忽略的，但设计者应该尽量去除，不去除必须确认每条警告的含义，避免因此使设计的实现产生隐患。

这个原则对仿真和布局布线同样适用。

4.2 调用模块的黑盒子（Black box）方法

使用黑盒子方法的原因主要有两点：

一是 HDL 代码中调用了一些 FPGA 厂家提供的模块（如 Altera 的 LPM 模块）或第三方提供的 IP，这些模块不需要综合，而且有些综合器也不能综合（如 FPGA CompilerII/FPGA Express 可以综合包含 LPM 的代码而 LeonardoSpectrum 不能）。因此须提供一个黑盒子接口给综合器，所调用的模块到布局布线时才进行连接。

二是方便代码的移植，由于厂家提供的模块或第三方提供的 IP 通常都是与工艺有关的，直接在代码中调用的话将不利于修改，影响代码移植。

下面以调用 Altera 的 LPM 库中的乘法器为例来说明。调用这样一个模块需要这样一个文件：mult8x8.v（可由 Quartus 的 MegaWizer Plug-in Manager 产生），代码如下：

```
// mult8x8.v
module mult8x8 (dataa, datab, result);
input [7:0] dataa;
input [7:0] datab;
output [15:0] result;

// exemplar translate_off
// synopsys translate_off
lpm_mult      lpm_mult_component(
            .dataa    (dataa),
            .datab    (datab),
            .aclr     (1'b0),
            .clock    (1'b0),
            .clken    (1'b0),
            .sum      (1'b0),
            .result   (result)
            );
      defparam
            lpm_mult_component.lpm_widtha              = 8,
            lpm_mult_component.lpm_widthb              = 8,
            lpm_mult_component.lpm_widths              = 16,
            lpm_mult_component.lpm_widthp              = 16,
            lpm_mult_component.lpm_representation      = "SIGNED",
// exemplar translate_on
// synopsys translate_on

endmodule
```

注意上述的代码有两对编译指示：

// exemplar translate_off 和 // exemplar translate_on （LeonardoSpectrum 支持）

// synopsys translate_off 和 // synopsys translate_on（LeonardoSpectrum 和 FPGA CompilerII 都支持）

对于相应的综合器，在这些编译指示中间的语句将会被忽略，那我们可以看到在综合过程中模块 mult8x8 实际变成了一个只有 I/O 定义的空盒子（即 black box），所以该部分的代码没有连接，在 Quartus 布局布线的时候，lpm 模块的代码才连接到整个设计，在仿真的时候，编译指示不影响模块的完整性。

参考

[1]：台湾清华 Verilog HDL 教程
[2]：Verilog HDL 硬件描述语言
[3]：文件头注释块示例
[4]：C 语言的风格
[5]：Verilog HDL Reference manual
[6]：Actel HDL coding style guide
[7]：LeonardoSpectrum HDL Synthesis
[8]：ASIC Design Partitioning
[9]：三种 FPGA 综合工具的比较
[10]: FPGA Synthesis Training Course

修订纪录

V2.0　　（何辉，2001-8-1）
　　修改了 4.2 节（黑盒子方法）的描述

V1.0　　（何辉，2001-3）
　　第一个版本