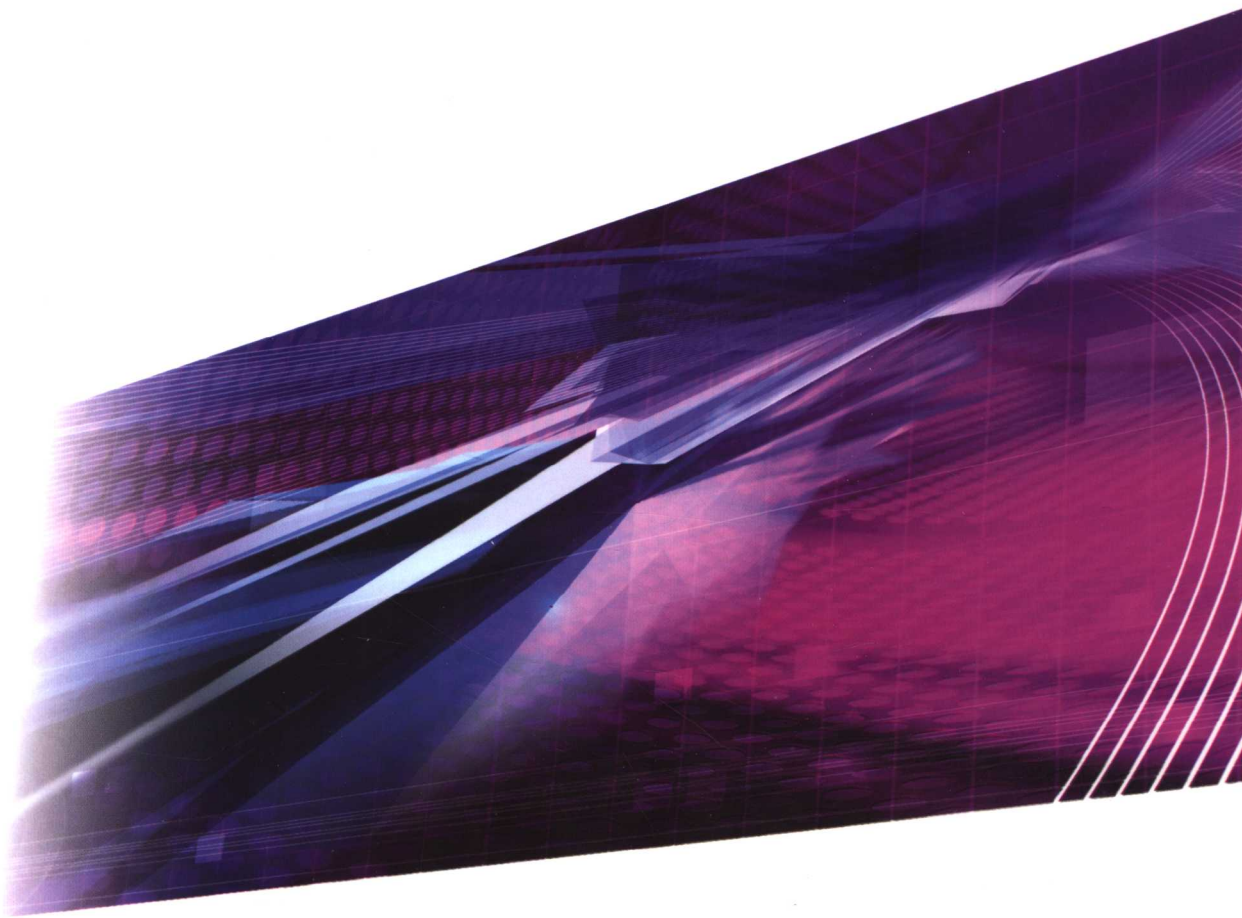
可编程逻辑器件快速进阶丛书

基于 Verilog HDL 的数字系统应用设计

王钿 卓兴旺 编著



国防工业出版社
National Defense Industry Press

可编程逻辑器件快速进阶丛书

基于 Verilog HDL 的数字系统应用设计

王 钿　卓兴旺　编著

国防工业出版社
·北京·

图书在版编目(CIP)数据

基于 Verilog HDL 的数字系统应用技术/王钿，卓兴旺编著. —北京：国防工业出版社，2006.1
(可编程逻辑器件快速进阶丛书)
ISBN 7-118-04281-1

Ⅰ. 基… Ⅱ. ①王…②卓… Ⅲ. ①硬件描述语言，VHDL—程序设计②数字系统—系统设计 Ⅳ. ①TP312 ②TP271

中国版本图书馆 CIP 数据核字（2005）第 150616 号

国防工业出版社出版发行
(北京市海淀区紫竹院南路 23 号)
(邮政编码 100044)
北京市李史山胶印厂
新华书店经售
*
开本 787×1092 1/16 印张 18¾ 428 千字
2006 年 1 月第 1 版 2006 年 1 月北京第 1 次印刷
印数：1—3000 册 定价：33.00 元

(本书如有印装错误，我社负责调换)
国防书店：**(010)68428422** 发行邮购：**(010)68414474**
发行传真：**(010)68411535** 发行业务：**(010)68472764**

内 容 简 介

本书结合实践系统地介绍了基于 Verilog 数字逻辑设计相关的内容，包括工具使用、RTL 设计及 Testbench 的设计。

本书共分为 6 章。第 1 章对数字逻辑设计进行了概述；第 2 章介绍了常用 EDA 工具的使用；第 3 章介绍了 RTL 设计的相关内容；第 4 章介绍了功能验证及 Testbench 相关的内容；第 5 章结合一个串口配置寄存器的电路对第 3 章和第 4 章的内容进行了实践；第 6 章对数字信号处理中的常用电路进行了讲解。

本书适合对 Verilog 语法已略有了解的读者阅读，也适于在数字逻辑设计方面摸索多年的工程师参考。

前　言

回顾自己以前在学校学习 VHDL/Verilog 的情形，当时不可谓不刻苦，每天起早摸黑，但是苦于市面上的书籍多以介绍语法为主，缺乏实践的内容，网上的资料又鱼龙混杂、真假难辨，所以水平提高很慢。后来因为一次机遇认识了我的老师及一些国外的朋友，他们给我讲了很多逻辑设计的基本思想，这才让我真正找到了逻辑设计的门路，从此也开始走上了这条逻辑设计之路。

我和我的同伴卓兴旺很愿意将这其中的一些学习经历和大家分享，希望可以让读者少走些弯路，也希望读者通过这些内容可以快速地入门，在前人的基础上推动中国 EDA 设计的发展。

学习建议

任何一种语言的语法学习从来都不是一件很难的事，难的是对这门语言背后思想的掌握。我们希望读者在学习时不要过分拘泥于具体的语法，而要真正地去理解语法背后的思想；我们并不希望读者对我们的内容全盘接收，而希望读者能够去思考这部分内容的本质，真正知道它们为什么是这样而不是那样。

我们期望读者在学习完这本书后能有如下收获或者体会：

(1) 能熟练掌握 FPGA 设计中常用的开发工具。

(2) RTL 设计的基础在于硬件意识，提高在于理解设计时序的思想。RTL 级设计只是一场时序“游戏”。

(3) 写好 Testbench 的基础在于熟练掌握 Verilog 行为级语法，提高在于对 Testbench 的层次有较好的把握。

组织结构

本书由 6 章组成：

第 1 章　逻辑设计发展现状及开发流程

本章主要是对逻辑设计的概况做了些介绍。

第 2 章　常用 FPGA 开发工具的使用

本章介绍了 FPGA 的常用开发工具的使用，包括：仿真软件 Modelsim6.0、综合工具 Synplify Pro8.0、Altera 公司的开发软件 Quartus II。

第 3 章　RTL 级建模

本章介绍了以下内容：硬件意识、基本语法、目标器件结构、约束原理及如何添加约束及设计时序的思想。

第 4 章　Testbench

本章介绍了功能验证涉及到的一些概念、Verilog 行为级的语法及 Testbench 的结构。

第 5 章 RS232 通信程序的设计

本章结合一个串口通信程序设计的例子，对第 3 章和第 4 章的内容进行了实践。

第 6 章 数字信号处理的 Verilog 设计

数字信号处理是逻辑设计中的另一重要内容，本章对数字信号处理的一些常用电路的原理进行了详细介绍。

其中，第 1 章、第 2 章和第 6 章由卓兴旺执笔，第 3 章、第 4 章和第 5 章由王钿完成。

限于我们的水平，错误和不当之处难以避免，欢迎大家发电子邮件与我们讨论相关内容，我的电子邮箱是：wangdian@tom.com。

编著者

2005 年 10 月

目　录

第 1 章　逻辑设计发展现状及开发流程 1
1.1　硬件描述语言 HDL(Hardware Description Language) 1
1.1.1　硬件描述语言简介 1
1.1.2　Verilog 语言简介 2
1.2　可编程逻辑器件 3
1.2.1　专用 ASIC 芯片 VS.可编程逻辑器件 3
1.2.2　FPGA VS.CPLD 4
1.2.3　主流 FPGA 厂商介绍 4
1.2.4　在选择 FPGA 器件时需要考虑的问题 5
1.3　基于 Verilog 的 FPGA 设计方法及流程 6
1.3.1　设计方法 6
1.3.2　典型的 FPGA 设计流程 6
1.4　SOC 与 IP 复用 8
1.4.1　SOC 简介 8
1.4.2　IP CORE 简介 9
1.4.3　设计方法学的进展 9

第 2 章　常用 FPGA 开发工具的使用 11
2.1　仿真工具 Modelsim 11
2.1.1　Modelsim 简介 11
2.1.2　用 Modelsim6.0 做功能仿真 12
2.1.3　用 Modelsim 做时序仿真 16
2.1.4　Modelsim 其他一些应用技巧 18
2.2　综合工具 Synplify Pro 21
2.2.1　Synplify Pro 简介 22
2.2.2　用 Synplify Pro 进行设计综合流程 23
2.3　集成开发环境 Quartus II 27
2.3.1　Quartus II 简介 27
2.3.2　设计输入 29
2.3.3　约束输入 34
2.3.4　综合 37
2.3.5　布局布线 42
2.3.6　仿真 45

2.3.7 时序分析……50
2.3.8 编程和配置……54

第 3 章 RTL 级建模……56
3.1 硬件意识……56
3.2 RTL 级语法……58
3.2.1 Verilog 模块基本结构……58
3.2.2 端口定义……59
3.2.3 对带三态输出端口的建模……59
3.2.4 对双向口的建模……60
3.2.5 数据类型……60
3.2.6 连续赋值语句……62
3.2.7 敏感信号列表……62
3.2.8 always 块……62
3.2.9 条件语句……63
3.2.10 多路分支语句……65
3.2.11 if …else 语句与 case 语句综合结果的比较……71
3.2.12 再谈锁存器……72
3.2.13 循环语句……73
3.2.14 阻塞与非阻塞赋值……74
3.2.15 模块例化……74
3.3 常用电路的设计……76
3.3.1 D 触发器……76
3.3.2 多路复用器……76
3.3.3 多路解复用器……77
3.3.4 计数器与分频器……78
3.3.5 移位寄存器……84
3.3.6 时钟使能电路……85
3.3.7 边沿检测电路……87
3.4 有限状态机的设计……89
3.4.1 概述……89
3.4.2 moore 型状态机……90
3.4.3 mealy 型状态机……91
3.4.4 moore 型状态机与 mealy 型状态机的选用……93
3.4.5 状态机的代码风格……94
3.4.6 状态编码……97
3.5 FPGA 结构……105
3.5.1 FPGA 的整体结构……106
3.5.2 IO 管脚……107

3.5.3 LE ····· 107
3.5.4 LAB ····· 109
3.5.5 片内存储单元 ····· 110
3.5.6 锁相环与全局时钟网络 ····· 111
3.5.7 DSP 模块 ····· 112
3.6 时序分析的基本概念 ····· 113
3.6.1 t_{SU} 与 t_H ····· 113
3.6.2 亚稳态 ····· 113
3.6.3 t_{CO} ····· 114
3.6.4 F_{MAX} ····· 114
3.6.5 Clock skew ····· 115
3.6.6 Multicycle path ····· 116
3.7 同步设计 ····· 117
3.7.1 什么是同步设计 ····· 117
3.7.2 同步设计的优点 ····· 117
3.7.3 同步设计准则 ····· 118
3.8 约束 ····· 120
3.8.1 约束对综合工具/布局布线工具的影响 ····· 120
3.8.2 在 synplify 中添加约束 ····· 120
3.8.3 在 Quartus 中添加约束 ····· 127
3.8.4 静态时序分析报告 ····· 133
3.9 如何提高电路的工作频率 ····· 135
3.9.1 影响电路工作频率的因素 ····· 135
3.9.2 减少走线时延 ····· 135
3.9.3 减少组合逻辑的时延 ····· 137
3.10 多时钟域处理 ····· 138
3.10.1 单个信号跨时钟域 ····· 139
3.10.2 一组信号跨时钟域 ····· 140
3.11 设计时序 ····· 142
3.12 RTL 级设计的其他注意事项 ····· 143
3.12.1 命名规范 ····· 143
3.12.2 保持良好的代码风格 ····· 143
3.12.3 参数化设计 ····· 144
3.12.4 输出应尽可能采用寄存器输出 ····· 144
3.12.5 将相关逻辑放在同一模块 ····· 144
3.12.6 尽量在“叶子”中做逻辑，顶层只做例化 ····· 144

第 4 章 Testbench ····· 145
4.1 功能验证 ····· 145

4.1.1 收敛模型 ····· 145
4.1.2 验证方法 ····· 146
4.1.3 覆盖率检查 ····· 147
4.2 Testbench 概述 ····· 147
4.2.1 什么是 Testbench ····· 147
4.2.2 为什么要写 Testbench ····· 148
4.2.3 Testbench 模型 ····· 148
4.2.4 一个简单的 Testbench ····· 149
4.3 行为级的 Verilog 语言 ····· 152
4.3.1 RTL 建模 VS.行为级建模 ····· 152
4.3.2 行为级的 Verilog 语法 ····· 154
4.3.3 再谈阻塞与非阻塞赋值 ····· 163
4.3.4 信号竞争问题 ····· 166
4.4 激励和响应 ····· 167
4.4.1 激励 ····· 167
4.4.2 响应 ····· 172
4.4.3 自动比较响应 ····· 173
4.5 总线功能模型 ····· 175
4.5.1 总线功能模型的地位 ····· 175
4.5.2 总线功能模型的要求 ····· 175
4.5.3 总线功能模型的设计 ····· 179
4.6 Testbench 的结构 ····· 182
4.6.1 Testbench 的层次 ····· 182
4.6.2 Testbench 的重用性 ····· 182

第 5 章 RS232 通信程序的设计 ····· 189
5.1 RS232 基础 ····· 189
5.2 设计需求 ····· 190
5.3 模块划分 ····· 191
5.3.1 RTL 级划分 ····· 191
5.3.2 Testbench 的结构划分 ····· 191
5.4 RTL 级代码 ····· 192
5.4.1 top_module 模块 ····· 192
5.4.2 config_registers 模块 ····· 194
5.4.3 frame_deal 模块 ····· 198
5.4.4 tx_frame 模块 ····· 200
5.4.5 rx_frame 模块 ····· 204
5.4.6 rs232 模块 ····· 209
5.4.7 txmit 模块 ····· 211

5.4.8 rxvr 模块 ····213
5.4.9 clken_gen 模块 ····216
5.5 Testbench ····218
5.5.1 testcase 模块 ····218
5.5.2 bm_frame_deal 模块 ····221
5.5.3 harness 模块 ····222
5.5.4 bfm_uart 模块 ····224
5.5.5 osc_rst 模块 ····226
5.6 仿真结果 ····227

第 6 章 数字信号处理的 Verilog 设计 ····228
6.1 数字信号处理 FPGA 实现简介 ····228
6.2 数字信号处理基本模块的实现 ····230
6.2.1 加法器 ····230
6.2.2 乘法器 ····239
6.2.3 积分器 ····245
6.2.4 微分器 ····246
6.2.5 抽取和内插 ····246
6.2.6 用 CORDIC 算法实现信号处理的常用模块 ····249
6.3 FIR 滤波器的实现 ····261
6.3.1 FIR 滤波器简介 ····261
6.3.2 FIR 滤波器的串行实现 ····262
6.3.3 FIR 滤波器的并行实现 ····266
6.3.4 FIR 滤波器的分布式实现 ····272
6.3.5 三种滤波方案的比较和选用 ····284
6.4 数字信号处理程序的仿真验证 ····285
附录 A 相关资源介绍 ····288
参考文献 ····289

第 1 章　逻辑设计发展现状及开发流程

目前数字电路设计的发展日新月异，从传统的 SSI(Small Scale Integration)、MSI(Medium Scale Integration)到现在的 LSI(Large Scale Integration)和 VLSI(Very Large Scale Integration),现在的发展趋势是把整个系统集成在一个芯片内，即所谓的 SOC(System On Chip)。随着设计复杂性的提高，传统的基于电路图的设计方法越来越不能满足设计需求，这在大规模集成电路时代就体现无遗，试想如何处理一个上千页的电路原理图？另外，原理图设计也不利于设计的重用性，难以实现跨平台设计，无法使用更加强大的第三方仿真工具及第三方综合工具。因此，硬件描述语言 HDL(Hardware Description Language)应运而生，同时传统的设计方法和流程也不能跟上器件的发展，新的设计方法和流程不断涌现，而 FPGA 器件以其灵活的可重配置和可重编程特性，逐渐成为一种流行的设计载体，本章将主要介绍这方面的相关内容。

1.1　硬件描述语言 HDL(Hardware Description Language)

1.1.1　硬件描述语言简介

硬件描述语言是一种用形式化方法来描述数字电路和设计数字逻辑系统的语言。它可以使数字逻辑电路设计者利用这种语言来描述自己的设计思想，然后利用电子设计自动化(EDA)工具进行仿真，自动综合到门级电路，再利用 ASIC 或 FPGA 实现其具体功能。在硬件描述语言出现之前，已经有很多成功的软件设计语言，比如 Fortran、Pascal 和 C 等，为什么不用这些语言描述硬件？答案是这些软件设计语言只能描述顺序执行的程序，不适合描述硬件的并行；软件设计语言中没有时序的概念，难以描述信号间的时序关系。

硬件描述语言自出现起，发展非常迅速，已经成功应用在数字逻辑设计的各个阶段，包括设计、仿真、验证、综合等，它们对设计自动化起到了极大的推动作用。到了 20 世纪 80 年代，已经出现了上百种硬件描述语言，这些语言一般面向特定的应用领域，彼此不兼容，这给设计的兼容和重用造成很大麻烦，设计工程师急需一种通用、标准的硬件描述语言。从 80 年代后期开始，硬件描述语言逐步向着标准化的方向发展，最终 VHDL 和 Verilog 适应了标准化需求，先后成为 IEEE 标准。此后用这两种标准的硬件描述语言进行数字逻辑设计的方法逐渐流行。与传统的电路图设计方法相比，用硬件描述语言进行电路设计有如下好处：

(1) 使用硬件描述语言，可以在较高的抽象层次描述设计。也就是说这样的设

计方法不仅提高了设计人员的效率，而且设计与特定的工艺无关。逻辑综合工具能自动地把设计转换成针对某种工艺的门级网表，如果出现了新的工艺，设计者不必重新设计电路，而只要针对新工艺，重新综合即可。

(2) 使用硬件描述语言进行设计，可以在设计阶段进行功能验证，这样设计者可以不断地修改和优化 RTL 代码描述，直到满足设计需求。大部分的错误和缺陷在这时被排除，防止错误被带到门级或者物理设计阶段。实践证明，错误发现越早，排除错误所花费的代价就越小。由于用硬件描述语言进行的设计可以在早期进行验证，从而极大地缩短设计周期、节约设计成本。

(3) 设计用文本的方式表示，简单高效，可以对设计添加注释，易于开发、调试和维护。

目前 VHDL 和 Verilog 两种硬件描述语言都非常流行，它们各有特点，它们孰优孰劣不是我们讨论的重点。本书主要介绍用 Verilog 语言进行数字逻辑与系统设计。

1.1.2 Verilog 语言简介

Verilog HDL 是硬件描述语言的一种，它是在 1983 年由 GateWay Design Automation 公司的 Phil Moorby 首创。在 1984 年—1985 年，Moorby 设计出第一个关于 Verilog-XL 的仿真器，1986 年，他对 Verilog HDL 的发展又做出了另一个巨大贡献，即提出了用于快速门级仿真的 XL 算法，使仿真速度有了很大提高。随着这种仿真器的流行，Verilog HDL 语言得到迅速发展。1989 年，Cadence 公司收购了 GateWay 公司，Verilog HDL 语言成为 Cadence 公司的私有财产。由于 Verilog 的私有性，妨碍了使用者之间的交流与共享，为与 VHDL 语言竞争，1990 年，Cadence 公司决定公开 Verilog HDL 语言，于是成立了 OVI(Open Verilog International)组织来负责 Verilog HDL 语言的发展。基于 Verilog HDL 的优越性，IEEE 于 1995 年制定了 Verilog HDL 的 IEEE 标准，即 Verilog HDL1364-1995。

采用 Verilog 语言设计的优点有以下几点：

(1) 作为一种通用的硬件描述语言，Verilog 易学易用，因为在语法上它与 C 语言非常类似，有 C 语言编程经验的人很容易发现这一点。

(2) 同一个设计，Verilog 语言允许设计者在不同层次上进行抽象。Verilog 语言中提供开关级、门级、RTL 级和行为级的支持，一个设计可以先用行为级语法描述它的算法，仿真通过后，再用 RTL 级描述，得到可综合的代码。

(3) Verilog 语言支持广泛，基本上所有流行的综合器、仿真器都支持 Verilog。

(4) 所有的后端生产厂商都提供 Verilog 的库支持，这样在制造芯片时，可以有更多选择。

(5) 能够描述层次设计，可使用模块实例结构描述任何层次，模块的规模可以是任意的，语言对此没有任何限制。

(6) Verilog HDL 语言的描述能力可以通过使用编程语言接口(PLI)机制进一步扩展。PLI 允许外部函数访问 Verilog 模块内部信息、允许设计者通过软件程序与仿真器进行交互。

(7) Verilog 语言对仿真提供强度的支持，虽然现在出现了专门的用于验证的语言，

但用 Verilog 语言直接对设计进行测试仍然是大部分工程师的首选。

1.2　可编程逻辑器件

1.2.1　专用 ASIC 芯片 VS. 可编程逻辑器件

逻辑器件按功能是否固定可分为两大类，即专用 ASIC(Application Specific Integrated Circuit，专用集成电路)芯片和可编程逻辑器件。

专用 ASIC 芯片与可编程逻辑器件的电路都是固定的，一经流片，所有电路在芯片中的位置、电路的结构及电路之间的连线都不可再改变。它们之间的区别在于两者的功能是否可改变：专用 ASIC 芯片的电路功能不可改变，例如一块南桥芯片就只能完成南桥的功能，而无法实现北桥的功能；可编程逻辑器件则不然，我们只需要改变配置文件，就能实现我们的功能，我们可以在一片可编程逻辑器件上实现南桥的功能，也可以让它实现北桥的功能，夸张一点说，我们希望它是什么它就能是什么。

专用 ASIC 芯片与可编程逻辑器件(下面将以 FPGA 为例进行比较)各有优缺点：

1. 开发周期

从开发周期这一角度来看，专用 ASIC 芯片的开发周期远远长于 FPGA。对于专用 ASIC 芯片的开发，它的开发周期包括前端设计、功能验证、时序验证、后端设计、生产及测试的时间，根据设计复杂程度需要数月至两年时间不等；如果芯片流片失败，还需要进行设计的反复，将耗费更长的时间。而对于 FPGA 设计来说，由于它是现成的芯片，芯片的性能已由厂商给予了保证，不涉及到后端的流程，并且它可以反复烧写，不用担心因一次设计失败而导致重新流片的危险，因此，使用 FPGA 的开发周期远短于专用 ASIC。

2. 性能

从性能这一角度来看，专用 ASIC 的表现要强于 FPGA。FPGA 的电路基于单元是相同的，电路布局也是固定的，片上 RAM、DSP 模块、逻辑单元的位置不可改变，为了保证资源之间的互联，FPGA 大量的电路面积都耗费在片上总线互联方面(甚至可达到 90%)，而专用 ASIC 则要灵活许多，电路可以选择全定制或者是基于标准单元库，电路之间的连线也可由后端设计指定。因此，FPGA 中的线时延通常要远大于专用 ASIC 芯片，从而电路工作频率上要低于专用 ASIC 芯片；在甚高速设计中，只能采用专用 ASIC 芯片设计。

3. 成本

专用 ASIC 的成本主要包括两部分：NRE 费用及每片的生产费用。NRE 费用是固定费用，指的是芯片最终从芯片制造厂制造出来以前客户需要投入的所有成本，这些成本包括工程资源、昂贵的软件设计工具、用来制造芯片不同金属层的昂贵光刻掩模组，以及初始原型器件的生产成本，这些 NRE 费用可能从数十万美元至数百万美元不等。生产费用是可变费用，指生产每片芯片所需要的费用。因此，芯片的产量越高，平摊到每片芯片的 NRE 费用也就越低，芯片的平均成本也就越低。

FPGA 由于大部分芯片面积都消耗在片内互连总线，在完成相同的功能的条件下

其芯片规模必要比专用 ASIC 大，因此每片的生产费用也比 ASIC 高不少。但对于用户来说，它不涉及到 NRE 的费用，在量少的时候平均到每片的成本还是要比 ASIC 低不少。

因此，业界一般只对市场需求量巨大的芯片才会采用专用 ASIC 设计的方式，对于需求不大的芯片则较多地使用 FPGA 或者 CPLD。

4. 风险

在开发风险方面，由于专用 ASIC 芯片的设计流程复杂，开发周期长，其风险要远高于 FPGA。

1.2.2 FPGA VS. CPLD

可编程逻辑器件的两种主要类型是 FPGA(Field Programmable Gate Array，现场可编程门阵列)和 CPLD(Complex Programmable Logic Device，复杂可编程逻辑器件)。虽然两者都是可编程的逻辑器件，但是两者之间仍有较大的区别。

(1) FPGA 是触发器密集型的器件，具有大量的触发器资源；而 CPLD 是组合逻辑密集型的器件，触发器资源较少。

(2) FPGA 的集成度远高于 CPLD。

(3) FPGA 的分段式布线结构决定了其延迟的不可预测性，而 CPLD 的连续式布线结构决定了它的时延是均匀的和可预测的。

因此，FPGA 的应用范围远较 CPLD 广泛，FPGA 可以用工业控制、通信、消费类电子等各种领域中较为复杂的设计，而 CPLD 一般只用于实现简单的控制，如地址译码等。

1.2.3 主流 FPGA 厂商介绍

目前主要的 FPGA 生产厂家包括 Xilinx、Altera、Actel、Lattice。其中，Xilinx 和 Altera 两大公司的产品大约占有 FPGA 市场 80%的份额。但是这四家的 FPGA 却各有特色。

Xilinx 公司是 FPGA 领域的老牌厂商，曾在 FPGA 领域占有绝对的领先优势。目前，它在低端市场推出了 Spartan-2 和 Spartan-3 系列产品，在高端市场则推出了 Virtex-2、Virtex-4 系列产品。

Altera 公司是目前在 FPGA 领域惟一可以和 Xilinx 平起平坐的公司。它在低端市场的产品有 Cyclone 和 CycloneII 系列产品，在高端市场则有 Stratix、Stratix GX 及 Stratix II 系列产品。

Actel 公司的 FPGA 虽然在 FPGA 市场的占有率不高，但是该公司基于反熔丝(Anit-fuse)工艺的 FPGA 却很有特色。反熔丝工艺最大的特点是芯片只能编程一次，不能反复烧写。虽然这种 FPGA 的灵活性很差，但是它的特点也使得它不易受外太空射线的影响而产生电子误翻转的情况(基于 SRAM 工艺的 FPGA 会有这种情况发生)，因此 Actel 的器件在军事领域使用得较多。

Lattice 公司的 CPLD 做得很不错，而 FPGA 领域则是新进入的。该公司拥有先进的高速收发器的技术，除此之外，它针对原有 FPGA 器件保密性差(可通过一定电路将配置

芯片的内容读取出来获得 FPGA 的烧写文件)的弱点，将配置芯片集成到 FPGA 内部，在增强保密性的同时减少了单板芯片的数量。

1.2.4　在选择 FPGA 器件时需要考虑的问题

怎样选择一款适合自己需求的 FPGA 芯片是每个设计者都会遇到的问题，对 FPGA 的选择通常要先预估设计的逻辑规模，然后考虑如下一些问题：

1. 器件的资源

这方面设计者主要会考虑器件中包含多少查找表和触发器？

器件能提供多大容量的 RAM 资源？是双口还是单口的？是分布式的还是块 RAM？它们能以何种方式进行组合使用？

器件具有多少可用的 IO 引脚？

器件可以提供多少可用的全局时钟/置位/复位网络？

器件中是否具有硬件乘法器或 DSP 模块？有多少？支持几种配置方式？

器件中是否带有高速串行收发器？

器件中是否有时钟管理单元或者锁相环？有多少？

器件是否支持丰富的 IP 核？

在同一系列中，是否提供引脚兼容且密度更高的器件？

2. 速度

也就是对性能的需求，每一种器件都有自己极限的运行频率(当然实际的运行频率还和设计本身有关)，我们选择的器件是否可以满足系统的工作频率要求？

3. 价格

我们选择的器件是否是在满足设计要求的条件下，价格最低的器件？

4. 服务

当设计过程中遇到与器件相关的问题时，是否能够得到快速高效的技术支持？

5. 环境适应性

若器件需要工作在一个比较苛刻的环境时，比如低温环境或高温环境，厂商是否能提供工业档或者军档的器件？

6. 工艺

从工艺上分，目前有两大类 FPGA，一是基于 SRAM 结构的；一是基于 Flash 结构的。SRAM 结构需要专门的配置芯片，掉电后需要重新配置。Flash 结构的不需要配置芯片，掉电后程序仍旧保存。

7. 软件支持

器件公司是否提供相应的设计软件？这些软件便宜吗？第三方软件对其支持的程度等。

8. 现有基础

包括设计人员对器件的熟悉程度，公司具备的条件等，因为这些制约产品的设计周期。

9. 供货周期

厂商的供货周期有多长，货源是否稳定？

1.3 基于 Verilog 的 FPGA 设计方法及流程

1.3.1 设计方法

有两种典型的系统设计方法，及自顶向下(top-down)和自低向上(bottom-up)。在top-down 的设计方法中，我们首先定义顶层(top-level)模块，然后定义顶层模块需要的子模块(sub-blocks)，然后继续向下分，一直到不能再分为止，我们称这样的模块为叶子模块(leaf cell)。top-down 设计方法的示意图如图 1-1 所示。

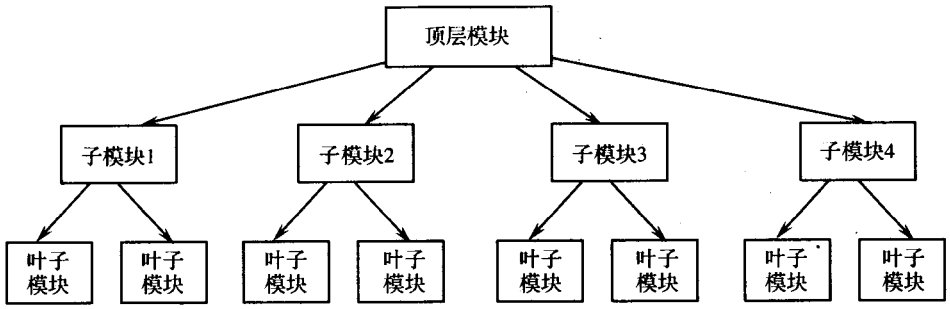


图 1-1 top-down 设计方法的示意图

在 bottom-up 设计方法中，我们首先定义和建立我们需要的叶子模块，然后利用这些叶子模块构建一些更大的模块，我们利用这些更大的模块构建高一级的功能块，直到整个系统。bottom-up 设计方法示意图如图 1-2 所示。

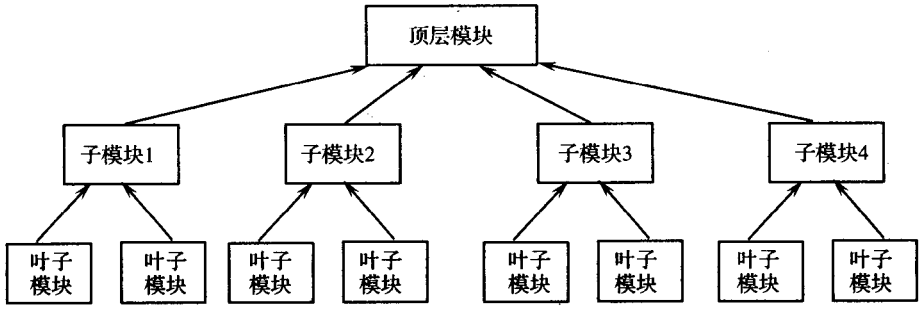


图 1-2 bottom-up 设计方法的示意图

在设计中我们会经常组合使用这两种设计方法，通常在定义系统结构和顶层系统模块时采用 top-down 设法方法，定义好需要的各个模块。在设计实现的时候，通常会采用先实现小的叶子模块，再实现大一点的模块，最后构建成整个系统，也就是采用的 bottom-up 的方法。一般在设计验证阶段也是采用 bottom-up 验证方法，先验证底层模块，保证每个底层模块的正确性后，再验证顶层模块。

1.3.2 典型的 FPGA 设计流程

图 1-3 给出了典型的 FPGA 设计流程。对于简单的只有逻辑设计的系统，可以不需