高等学校计算机应用规划教材

EDA 技术与 CPLD/FPGA 开发应用简明教程

(第二版)

刘爱荣　王振成　陈杨　叶建森　　编著

清华大学出版社
北　　京

内 容 简 介

在信息技术高速发展的现代社会，电子系统的设计方法和设计手段已有了革命性的变化。可编程逻辑器件和 EDA 技术已广泛应用于通信、工业自动化、智能家电、智能交通、智能仪表、大屏幕、图像处理以及计算机等领域。因此，EDA 技术是电子工程师必须掌握的技术。

全书共分 12 章。本书根据课堂教学和实践的需要，详细介绍了 EDA 技术的基本知识、大规模可编程逻辑器件 CPLD/FPGA 的结构原理、EDA 开发工具的使用方法、VHDL 语言的语法结构和编程技巧、宏功能模块的应用、状态机和 SOPC 设计及应用。为提高读者的工程设计能力， 第 9~11 章分别介绍了 CPLD/FPGA 器件在数字系统、通信工程和计算机等领域的具体应用，并且运用大量综合性实例对各种关键技术进行了深入浅出的分析。此外，基础章节配有思考题，应用章节配有设计题，附录 4 配有实训内容、设计思路和实训步骤，为读者实训提供方便。

本书对应的电子教案、实例源文件和参考答案可以到 http://www.tupwk.com.cn/downpage/index.asp 网站下载。

图书在版编目(CIP)数据

EDA 技术与 CPLD/FPGA 开发应用简明教程/刘爱荣等　编著. —2 版 —北京：清华大学出版社，2013.10

(高等学校计算机应用规划教材)

ISBN 978-7-302-33023-3

Ⅰ. ①E…　Ⅱ. ①刘…　Ⅲ. ①电子电路—电路设计—计算机辅助设计—高等学校—教材　Ⅳ. ①TN702

中国版本图书馆 CIP 数据核字(2013)第 149118 号

责任编辑：胡辰浩　袁建华
装帧设计：牛静敏
责任校对：曹　阳
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62794504
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：27　　**字　　数**：624 千字
版　　次：2013 年 10 月第 1 版　　**印　　次**：2013 年 10 月第 1 次印刷
印　　数：1～3500
定　　价：45.00 元

产品编号：

前　言

由于高密度现场可编程逻辑器件(CPLD/FPGA)和专用集成电路的飞速发展，传统的设计技术已经不适合大规模及超大规模集成电路，以往分立的数字电路已经被可编程逻辑器件所取代。电子设计自动化 EDA(Electronic Design Automation)技术正是为了适应现代电子产品设计的要求，吸收多学科最新成果而形成的一门新技术。

利用 EDA 开发工具进行电子系统的设计，具有以下几个特点：①用软件的方式设计硬件；②由用软件方式设计的系统到由开发软件自动完成硬件系统的转换；③设计过程中可用软件进行各种仿真；④系统可现场编程，在线升级；⑤整个系统可集成在一个芯片上，体积小、功耗低并且可靠性高。目前，可编程逻辑器件在通信系统、智能交通、智能家电、物联网及复杂的现场控制系统中得到广泛的应用。因此，EDA 技术是现代电子设计工程师必须掌握的技术，基于 CPLD/FPGA 技术的开发应用已经成为数字时代的应用技术潮流。

本书的目的是帮助读者尽快掌握 EDA 技术，让读者学会应用硬件描述语言、原理图和状态图的混合设计方法设计数字系统。全书共 12 章：第 1 章主要介绍了 EDA 技术的基本知识和大规模可编程逻辑器件 CPLD/FPGA 的结构原理；第 2 章主要介绍了应用原理图输入法设计逻辑电路的流程；第 3 章介绍了 VHDL 结构和要素；第 4 章介绍了 Quartus II；第 5 章介绍了 VHDL 语言描述语句；第 6 章介绍了基本逻辑电路设计；第 7 章介绍了 CPLD/FPGA 应用系统设计实例；第 8 章介绍了有限状态机的设计；第 9 章介绍了宏功能模块与 IP 应用；第 10 章介绍了 FPGA 在 DSP 领域中的应用；第 11 章介绍了 FPGA 在通信工程中的实践应用；第 12 章介绍了 SOPC 系统开发技术。

本书在选材上注重内容新颖、技术先进并且重点突出。书中给出了经实践验证的大量设计实例，希望能对读者迅速掌握大规模可编程器件设计与应用有所帮助。

全书由刘爱荣、王振成、陈杨、叶建森、李立凯、李红丽、张璐璐、刘宏宇、王欣编写。其中第 1.1~1.10 节和附录 3 由张璐璐编写，第 1.11~1.16 节和第 4 章由王欣编写，第 2 章由王振成编写，第 3 章由刘宏宇编写，第 5.1~5.2 节和第 10 章由刘爱荣编写，第 6、7 和第 12 章由李立凯编写，第 9.1~9.2 节和附录 4 由陈杨编写，第 8 章和 9.3~9.7 节由叶建森编写，第 11 章、5.3~5.4 节及附录 1、附录 2 由李红丽编写，全书由刘爱荣统稿、定稿。另外，解放军信息工程大学王志新教授对此书的编写提出了宝贵意见，在此深表感谢。

在编写本书的过程中参考了相关文献，在此向这些文献的作者深表感谢。由于 EDA 技术是一门发展迅速的新技术，加上作者水平有限，书中难免有疏漏、不妥甚至错误之处，恳请专家和广大读者批评指正。我们的信箱是 huchenhao@263.net，电话是 010-62796045。

编　者

2013 年 3 月

目　录

第1章　EDA概述与可编程逻辑器件

本章主要阐述了 EDA 技术的含义、发展历程、主要内容，面向 CPLD/FPGA 的开发流程；可编程逻辑器件发展进程、种类及分类方法；复杂可编程逻辑器件(CPLD)的基本结构，Altera 公司的主要器件介绍；现场可编程门阵列(FPGA)的结构及配置；FPGA 和 CPLD 的开发应用选择；可数字系统的设计方法及设计工具简介，EDA 技术的优势等内容。通过本章的学习，读者可对 EDA 技术有一个基本的认识。

1.1　EDA 技术

EDA(Electrical Design Automation，电子设计自动化)技术是现代集成电路及电子整机系统设计科技创新和产业发展的关键技术。当前集成电路技术已进入超深亚微米工艺和片上系统(SOC)阶段，集成化、微型化和系统化的趋势使得集成电路设计及以集成电路为核心的电子系统设计成为一个庞大的系统工程，离开 EDA 技术，集成电路及电子系统设计将寸步难行。EDA 技术教学是培养高素质电子设计人才，尤其是 IC 设计人才的重要途径。

EDA 技术是一门发展迅速的新技术，涉及面广，内容丰富，理解各异，目前尚无统一的看法。作者认为：EDA 技术有狭义和广义之分，狭义 EDA 技术就是以大规模可编程逻辑器件为设计载体，以硬件描述语言为系统逻辑描述的主要表达方式，以计算机、大规模可编程逻辑器件的开发软件及实验开发系统为设计工具，通过有关的开发软件，自动完成用软件的方式设计的电子系统到硬件系统的逻辑编译、逻辑化简、逻辑分割、逻辑综合及优化、逻辑布局布线、逻辑仿真，直至完成对特定目标芯片的适配编译、逻辑映射、编程下载等工作，最终形成集成电子系统或专用集成芯片的一门新技术，或称为 IES/ASIC 自动设计技术。本书讨论的对象为狭义的 EDA 技术。广义的 EDA 技术除狭义的 EDA 技术之外，还包括计算机辅助分析 CAA 技术(如 PSPICE、EWB、MATLAB 等)、计算机辅助制造 CAM 和印刷电路板计算机辅助设计 PCB-CAD 技术(如 PROTEL、ORCAD 等)。

1.2　EDA 技术发展历程

EDA 技术伴随着计算机、集成电路、电子系统设计的发展，经历了计算机辅助设计(Computer Assist Design，CAD)、计算机辅助工程设计(Computer Assist Engineering Design，

简称 CAE)和电子设计自动化(Electronic Design Automation，EDA)3 个发展阶段。

1.2.1　20 世纪 70 年代的计算机辅助设计 CAD 阶段

早期的电子系统硬件设计采用的是分立元件，随着集成电路的出现和应用，硬件设计进入到发展的初级阶段。初级阶段的硬件设计大量选用中小规模标准集成电路，人们将这些器件焊接在电路板上，做成初级电子系统，对电子系统的调试是在组装好的 PCB(Printed Circuit Board)板上进行的。

由于设计师对图形符号使用数量有限，传统的手工布线方法无法满足产品复杂性的要求，更不能满足工作效率的要求。这时，人们开始将产品设计过程中高度重复性的繁杂劳动，如布图布线工作，用二维图形编辑与分析的CAD 工具替代，最具代表性的产品就是美国 ACCEL 公司开发的 Tango 布线软件。20 世纪 70 年代，是 EDA 技术发展初期，由于 PCB 布图布线工具受到计算机工作平台的制约，其支持的设计工作有限且性能比较差。

1.2.2　20 世纪 80 年代的计算机辅助工程设计 CAE 阶段

初级阶段的硬件设计是用大量不同型号的标准芯片实现电子系统设计的。随着微电子工艺的发展，相继出现了集成上万只晶体管的微处理器，集成几十万直到上百万储存单元的随机存储器和只读存储器。此外，支持定制单元电路设计的掩膜编程的门阵列，如标准单元的半定制设计方法以及可编程逻辑器件(PAL 和 GAL)等一系列微结构和微电子学的研究成果都为电子系统的设计提供了新天地。因此，可以用少数几种通用的标准芯片实现电子系统的设计。

伴随计算机和集成电路的发展，EDA技术进入到计算机辅助工程设计阶段。20世纪 80 年代初，推出的 EDA 工具则以逻辑模拟、定时分析、故障仿真、自动布局和布线为核心，重点解决电路设计没有完成之前的功能检测等问题。利用这些工具，设计师能在产品制作之前预知产品的功能与性能，能生成产品制造文件，在设计阶段对产品性能的分析前进了一大步。

如果说 20 世纪 70 年代的自动布局布线的 CAD 工具代替了设计工作中绘图的重复劳动，那么，到了 20 世纪 80 年代出现的具有自动综合能力的 CAE 工具则代替了设计师的部分工作，对保证电子系统的设计，制造出最佳的电子产品起着关键的作用。到了 20 世纪 80 年代后期，EDA 工具已经可以进行设计描述、综合与优化和设计结果验证，CAE 阶段的EDA 工具不仅为成功开发电子产品创造了有利条件，而且为高级设计人员的创造性劳动提供了方便。但是，大部分从原理图出发的EDA 工具仍然不能适应复杂电子系统的设计要求，而具体化的元件图形制约着优化设计。

1.2.3　20 世纪 90 年代电子系统设计自动化 EDA 阶段

为了满足千差万别的系统用户提出的设计要求，最好的办法是由用户自己设计芯片，

让用户把想设计的电路直接设计在自己的专用芯片上。微电子技术的发展，特别是可编程逻辑器件的发展，使得微电子厂家可以为用户提供各种规模的可编程逻辑器件，使设计者通过设计芯片实现电子系统功能。EDA 工具的发展，又为设计师提供了全线 EDA 工具。这个阶段发展起来的EDA 工具，目的是在设计前期将设计师从事的许多高层次设计由工具来完成，如可以将用户要求转换为设计技术规范，有效地处理可用的设计资源与理想的设计目标之间的矛盾，按具体的硬件、软件和算法分解设计等。由于电子技术和 EDA 工具的发展，设计师可以在短时间内使用EDA 工具，通过一些简单标准化的设计过程，利用微电子厂家提供的设计库来完成数万门 ASIC 和集成系统的设计与验证。

20 世纪 90 年代，设计师们逐步从使用硬件转向设计硬件，从单个电子产品开发转向系统级电子产品开发(即片上系统集成，System on a chip)。因此，EDA 工具是以系统设计为核心，包括系统行为级描述与结构综合，系统仿真与测试验证，系统划分与指标分配，系统决策与文件生成等一整套的电子系统设计自动化工具。这时的EDA 工具不仅具有电子系统设计的能力，而且能提供独立于工艺和厂家的系统级设计能力，具有高级抽象的设计构思手段。例如，提供方框图、状态图和流程图的编辑能力，具有适合层次描述和混合信号描述的硬件描述语言(如 VHDL、AHDL 或 Verilog-HDL)，同时含有各种工艺的标准元件库。只有具备上述功能的 EDA 工具，才可能使电子设计工程师能够在不熟悉各种半导体工艺的情况下，完成电子系统的设计。

未来的 EDA 技术将向广度和深度两个方向发展，EDA 将会超越电子设计的范畴进入其他领域。随着基于 EDA 的 SOC(单片系统)设计技术的发展，软硬核功能库的建立，以及基于 VHDL 所谓自顶向下设计理念的确立，未来的电子系统的设计与规划将不再是电子工程师们的专利。有专家认为，21 世纪将是 EDA 技术快速发展的时期，并且 EDA 技术将是对 21 世纪产生重大影响的十大技术之一。

1.3　面向 CPLD/FPGA 的 EDA 技术主要内容

EDA 技术涉及面广，内容丰富，从教学和实用的角度看，究竟应掌握些什么内容呢？作者认为，主要应掌握如下 4 个方面的内容：①大规模可编程逻辑器件；②硬件描述语言；③软件开发工具；④实验开发系统。其中，大规模可编程逻辑器件是利用 EDA 技术进行电子系统设计的载体；硬件描述语言是利用 EDA 技术进行电子系统设计的主要表达手段；软件开发工具是利用EDA 技术进行电子系统设计的智能化设计工具；实验开发系统则是利用 EDA 技术进行电子系统设计的硬件验证工具。为了使读者对 EDA 技术有一个总体印象，下面对 EDA 技术的主要内容进行概要介绍。

1.3.1　大规模可编程逻辑器件

可编程逻辑器件(简称 PLD)是一种由用户编程以实现某种逻辑功能的新型逻辑器件。

FPGA 和 CPLD 分别是现场可编程门阵列和复杂可编程逻辑器件的简称。现在，FPGA 和 CPLD 器件的应用已十分广泛，它们将随着 EDA 技术的发展而成为电子设计领域的重要角色。国际上生产 FPGA/CPLD 的主流公司，并且在国内占有市场份额较大的主要是 Xilinx、Altera 和 Lattice 三家公司。Xilinx 公司的 FPGA 器件有 XC2000、XC3000、XC4000、XC4000E、XC4000XLA，XC5200 系列等，可用门数为 1200~18000。Altera 公司的 CPLD 器件有 MAX 系列和 FLEX 系列，FLEX 系列有 FLEX6000、FLEX8000、FLEX10K、FLEX10KE 系列等，提供门数为 5000~25000。Lattice 公司的 ISP-PLD 器件有 ispLSI1000、ispLSI2000、ispLSI3000、ispLSI6000 系列等，集成度可多达 25000 个 PLD 等效门。

FPGA 在结构上主要分为 3 个部分，即可编程逻辑单元、可编程输入/输出单元和可编程连线 3 个部分。CPLD 在结构上主要包括 3 个部分，即可编程逻辑宏单元、可编程输入/输出单元和可编程内部连线。

高集成度、高速度和高可靠性是 FPGA/CPLD 最明显的特点，其时钟延时可小至 ns 级，结合其并行工作方式，在超高速应用领域和实时测控方面有着非常广阔的应用前景。在高可靠应用领域，如果设计得当，将不会存在类似于 MCU 的复位不可靠和 PC 可能跑飞等问题。FPGA/CPLD的高可靠性还表现在几乎可将整个系统下载于同一芯片中，实现所谓片上系统，从而大大缩小了体积，易于管理和屏蔽。由于 FPGA/CPLD 的集成规模非常大，可利用先进的EDA 工具进行电子系统设计和产品开发。由于开发工具的通用性、设计语言的标准化以及设计过程几乎与所用器件的硬件结构没有关系，因而设计开发成功的各类逻辑功能块软件有很好的兼容性和可移植性。它几乎可用于任何型号和规模的 FPGA/CPLD 中，从而使得产品设计效率大幅度提高。可以在很短时间内完成十分复杂的系统设计，这正是产品快速进入市场最宝贵的特征。美国 IT 公司认为，一个 ASIC 的 80%功能可用于 IP 核等现成逻辑合成。而未来大系统的 FPGA/CPLD 设计仅仅是各类再应用逻辑与 IP 核(Core)的拼装，其设计周期将更短。与 ASIC 设计相比，FPGA/CPLD 显著的优势是开发周期短、投资风险小、产品上市速度快、市场适应能力强和硬件升级回旋余地大，而且当产品定型和产量扩大后，可将在生产中达到充分检验的 VHDL 设计迅速实现 ASIC 投产。

对于一个开发项目，究竟是选择 FPGA 还是选择 CPLD 呢？主要看开发项目本身的需要。对于普通规模，且产量不是很大的产品项目，通常使用 CPLD 比较好。对于大规模的逻辑设计、ASIC 设计和单片系统设计，则多采用 FPGA。另外，FPGA 掉电后将丢失原有的逻辑信息，所以在使用中需要为 FPGA 芯片配置一个专用 ROM。

1.3.2　硬件描述语言(HDL)

常用的硬件描述语言有 VHDL、Verilog 和 ABEL。

- VHDL：作为 IEEE 的工业标准硬件描述语言，在电子工程领域，已成为事实上的通用硬件描述语言。

- Verilog：支持的 EDA 工具较多，适用于 RTL 级和门电路级的描述，其综合过程较 VHDL 稍简单，但其在高级描述方面不如 VHDL。
- ABEL：一种支持各种不同输入方式的 HDL，被广泛用于各种可编程逻辑器件的逻辑功能设计，由于其语言描述的独立性，因而适用于各种不同规模的可编程器件的设计。

有专家认为，在新世纪中，VHDL 与 Verilog 语言将承担几乎全部的数字系统设计任务。

1.3.3　软件开发工具

目前比较流行的主流厂家的 EDA 的软件工具有 Altera 公司的 Quartus II、Lattice 公司的 ispEXPERT、Xilinx 公司的 Foundation Series ISE(简称 ISE)等。

- Quartus II：支持原理图、VHDL 和 Verilog 等语言文本文件，以及以波形与 EDIF 等格式的文件作为设计输入，并支持这些文件的任意混合设计。它具有门级仿真器，可以进行功能仿真和时序仿真，能够产生精确的仿真结果。在适配之后，Quartus II 生成供时序仿真用的 EDIF、VHDL 和 Verilog 这 3 种不同格式的网表文件，它界面友好，使用便捷，被誉为业界最易学易用的 EDA 软件，并支持主流的第三方 EDA 工具，支持所有 Altera 公司的 FPGA/CPLD 大规模逻辑器件。
- ispEXPERT：ispEXPERT System 是 ispEXPERT 的主要集成环境。通过它可以进行 VHDL、Verilog 及 ABEL 语言的设计输入、综合、适配、仿真和系统下载。ispEXPERT System 是目前流行的 EDA 软件中最容量掌握的设计工具之一，它界面友好、操作方便、功能强大，并与第三方 EDA 工具兼容良好。
- Foundation Series ISE：Xilinx 公司最新集成开发的 EDA 工具。它采用自动化的、完整的集成设计环境。Foundation 项目管理器集成了 Xilinx 实现工具，并包含了强大的 Synopsys FPGA Express 综合系统，是业界最强大的 EDA 设计工具之一。

这 3 个软件的基本功能相同，主要差别在于：①面向的目标器件不一样；②三者的性能各有优劣。

1.3.4　实验开发系统

实验开发系统提供芯片下载电路及 EDA 实验/开发的外围资源(类似于用于单片机开发的仿真器)，供硬件验证用。一般包括：①实验或开发所需的各类基本信号发生模块，包括时钟、脉冲、高低电平等；②FPGA/CPLD 输出信息显示模块，包括数码显示、发光管显示、声响指示等；③监控程序模块，提供“电路重构软配置”；④目标芯片适配座以及上面的 FPGA/CPLD 目标芯片和编程下载电路。

目前从事 EDA 实验开发系统研究的公司和院校有：杭州康芯电子有限公司、清华大学、北京理工大学、复旦大学、西安电子科技大学、东南大学和浙江天煌科技实业有限公司等。

1.3.5　关于 EDA 技术的学习重点及学习方法

EDA 技术作为一门发展迅速、有着广阔应用前景的新技术，涉及面广，内容丰富。作者结合自己的学习及教学体会，就 EDA 技术学习的有关问题提出一些指导意见，供读者参考。

1. EDA 技术的学习重点

从实用和教学的角度讲，作者认为，EDA 技术的学习主要应掌握 4 个方面的内容：①大规模可编程逻辑器件；②硬件描述语言；③软件开发工具；④实验开发系统。其中，硬件描述语言是重点。

对于大规模可编程逻辑器件，主要是了解其分类、基本结构、工作原理、各厂家产品的系列、性能指标以及如何选用，而对于各个产品的具体结构不必研究过细。

对于硬件描述语言，除了掌握基本语法规定外，更重要的是要理解 VHDL 的 3 个“精髓”：软件的强数据类型与硬件电路的唯一性、硬件行为的并行性决定了 VHDL 语言的并行性、软件仿真的顺序性与实际硬件行为的并行性；要掌握系统的分析与建模方法，能够将各种基本语法规定熟练地运用于自己的设计中。

对于软件开发工具，应熟练掌握从源程序的编辑、逻辑综合、逻辑适配以及各种仿真、硬件验证各步骤的使用。

对于实验开发系统，主要能够根据自己所拥有的设备，熟练地进行硬件验证或变通地进行硬件验证。

2. EDA 技术的学习方法

抓住一个重点：VHDL(或其他语言)的编程；掌握两个工具：FPGA/CPLD 开发软件和 EDA 实验开发系统的使用；运用 3 种手段：案例分析、应用设计、上机实践；采用 4 个结合：边学边用相结合，边用边学相结合，理论与实践相结合，课内与课外相结合。

1.4　EDA 技术应用对象

一般地，利用 EDA 技术进行电子系统设计的最后目标，是完成专用集成电路 ASIC 或印制电路板(PCB)的设计和实现，如图 1-1 所示。其中，PCB 设计指的是电子系统的印制路板设计，从电路原理图到 PCB 上元件的布局、布线、阻抗匹配、信号完整性分析及板级仿真，到最后的电路板机械加工文件生成，这些都需要相应的 EDA 工具软件辅助设计者来完成，这仅是 EDA 技术应用的一个重要方面，但本书限于篇幅不作展开。

ASIC 作为最终的物理平台，集中容纳了用户通过 EDA 技术将电子应用系统的既定功能和技术指标具体实现的硬件实体。

专用集成电路就是具有专门用途和特定功能的独立集成电路器件，根据这个定义，作为 EDA 技术最终实现目标的 ASIC，可以通过 3 种途径来完成。

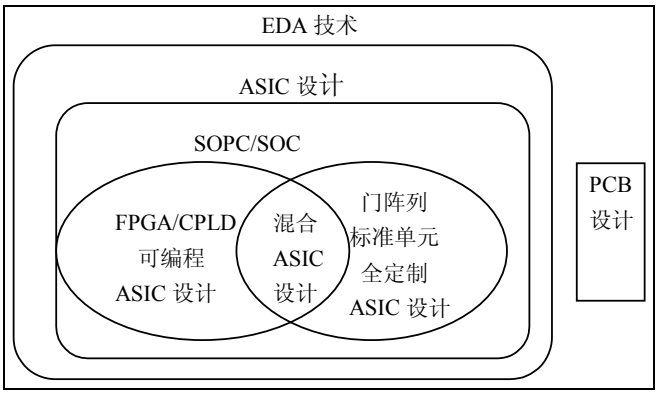


图 1-1　EDA 技术实现目标

1.4.1　可编程逻辑器件

FPGA 和 CPLD 是实现这一途径的主流器件，它们的特点是：直接面向用户，具有极大的灵活性和通用性；使用方便，硬件实现和测试快捷；开发效率高，成本低，上市时间短；技术维护简单，工作可靠性高等。FPGA 和 CPLD 的应用是 EDA 技术有机融合软硬件电子设计技术、SOPC 和 ASIC 设计，以及对自动设计与自动实现最典型的诠释。由于 FPGA 和 CPLD 的开发工具、开发流程和使用方法与 ASIC 有类似之处，因此这类器件通常也被称为可编程专用 IC，或可编程 ASIC。

1.4.2　半定制或全定制 ASIC

基于 EDA 技术的半定制或全定制 ASIC，根据它们的实现工艺，可统称为掩模(Mask) ASIC，或直接称 ASIC。可编程 ASIC 与掩模 ASIC 相比，不同之处在于前者具有面向用户的灵活多样的可编程性，即硬件结构的可重构特性。

1.4.3　混合 ASIC

混合 ASIC(不是指数模混合 ASIC)主要指既具有面向用户的 FPGA 可编程功能和逻辑资源，同时也含有可方便调用和配置的硬件标准单元模块，如 CPU、RAM、硬件加法器、乘法器、锁相环等。

1.5　面向 CPLD/FPGA 的 EDA 开发流程

完整地了解利用 EDA 技术进行设计开发的流程对于正确选择和使用 EDA 软件、优化

设计项目、提高设计效率十分有益。一个完整的、典型的 EDA 设计流程既是自顶向下设计方法的具体实施途径，也是 EDA 工具软件本身的组成结构。

1.5.1　设计输入

对于目标器件为 FPGA 和 CPLD 的 VHDL 设计，其工程设计步骤如何呢？EDA 的工程设计流程与建筑设计流程相似：第一，需要进行“电路系统以原理图或 HDL 文本编辑方式的输入计算机”；第二，要进行“逻辑综合”，即用一定的逻辑表达手段将表达出来的设计经过一系列的操作，分解成一系列的逻辑电路及对应的关系(电路分解)；第三，要进行目标器件的“布线/适配”，即在选用的目标器件中建立这些基本逻辑电路的对应关系(逻辑实现)；第四，目标器件的编程下载，即将前面的软件设计经过编程变成具体的设计系统(物理实现)；最后要进行硬件仿真/硬件测试，即验证所设计的系统是否符合要求。同时，在设计过程中要进行有关“仿真”， 即模拟有关设计结果与设计构想是否相符。综上所述，EDA 的工程设计基本流程如图 1-2 所示，图中的设计流程具有一般性。

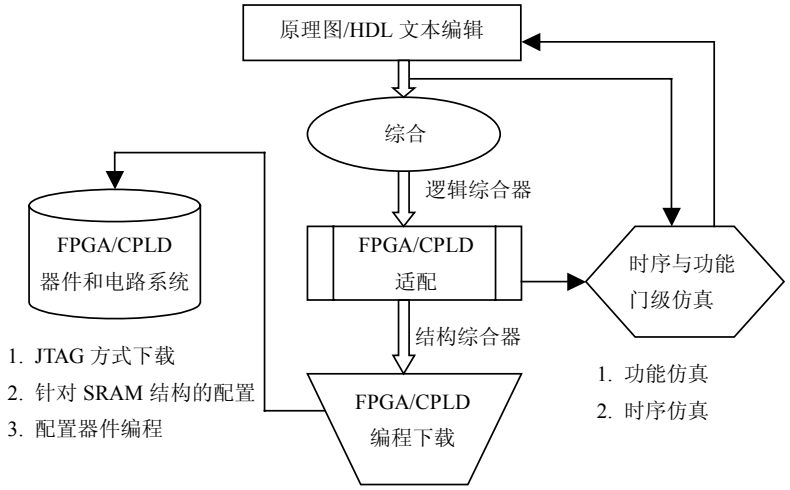


图 1-2　FPGA/CPLD 的 EDA 开发流程

1. 图形输入

图形输入通常包括原理图输入、状态图输入和波形图输入等方法。状态图输入方式就是根据电路的控制条件和不同的转换方式，用绘图的方法，在EDA 软件的状态图编辑器上绘出状态图，然后由 EDA 编辑器和综合器将此状态变化流程图编辑综合成状态网表。

波形图输入方法则是将待设计的电路看成是一个黑盒子，只需告诉EDA 工具该黑盒子电路的输入和输出时序波形图，EDA 工具即能据此完成黑盒子电路的设计。

原理图输入方法是一种类似于传统电子设计方法的原理图编辑输入方式，即在EDA 软件的图形编辑界面上绘制能完成功能的电路原理图。原理图由逻辑器件(符号)和连接线构成，图中的逻辑器件可以是EDA 软件库中预测的功能模块，如与门、非门、或门、触发器以及各种 74 系列器件功能的宏模块，甚至还有一些 IP 核的功能块。

2. 硬件描述语言文本输入

这种方式与传统的计算机软件语言编辑输入基本一致，就是将使用了某种硬件描述语言的电路设计文本，如 VHDL 或 Verilog 的源程序，进行编辑输入。

1.5.2　逻辑综合和优化

综合(Synthesis)，就是把某些东西结合到一起，把设计抽象层次中的一种表述转化成另一种表述的过程。

对于电子设计领域的综合概念可以表示为：将用行为和功能层次表达的电子系统转换为低层次的便于具体实现的模块组合装配而成的过程。

事实上，设计过程的每一步都可称为一个综合环节。设计过程通常从高层次的行为描述开始，以最低层的结构描述结束，每个综合步骤都是上一层次的转换。

- 从自然语言表述转换到 Verilog 语言算法表述，是自然语言综合。
- 从算法表述转换到寄存器传输级(Register Transport Level，RTL)表述，即从行为域到结构域的综合，是行为综合。
- 从 RTL 级转换到逻辑门(包括触发器)的表述，即逻辑综合。
- 从逻辑门表述转换到版图表述(ASIC 设计)，或转换到 FPGA 的配置网表文件，可称为版图综合或结构综合。

一般地，综合是仅对 HDL 而言的。利用 HDL 综合器对设计进行综合是十分重要的一步。因为综合过程将把软件设计的 HDL 描述与硬件结构挂钩，是将软件转化为硬件电路的关键步骤，是文字描述与硬件实现的一座桥梁。综合就是将电路的高级语言——行为描述转换成低级的、可与 FPGA/CPLD 的基本结构相映射的网表文件或程序。

当输入的 HDL 文件在 EDA 工具中检测无误后，首先面临的是逻辑综合，因此要求 HDL 源文件中的语句都是可综合的(即可硬件实现的)。

整个综合过程就是将设计者在 EDA 平台上编辑输入的 HDL 文本、原理图或状态图形描述，依据给定的硬件结构组件和约束控制条件进行编译、优化、转换和综合，最终获得门级电路甚至更底层的电路描述网表文件。由此可见，综合器工作前，必须给定最后实现的硬件结构参数，它的功能就是将软件描述与给定的硬件结构用某种网表文件的方式对应起来，成为相应的映射关系。

如果把综合理解为映射过程，那么显然这种映射不是唯一的，并且综合的优化也不是单纯的或一个方向的。为达到速度、面积(资源)、性能的要求，往往需要对综合加以约束，称为综合约束。

1.5.3　适配(目标器件的布局布线)

适配器也称结构综合器，它的功能是将由综合器产生的网表文件配置于指定的目标器

件中，使之产生最终的下载文件，如 JEDEC、JAM、SOF、POF 文件等。适配所选定的目标器件必须属于原综合器指定的目标器件系列。通常，EDA 软件中的综合器可由专业的第三方 EDA 公司提供，而适配器则需由 FPGA/CPLD 供应商提供。因为适配器的适配对象直接与器件的结构细节相对应。

适配器就是将综合后的网表文件针对某一具体的目标器件进行逻辑映射操作，其中包括底层器件配置、逻辑分割、优化、布局布线操作。适配完成后可以利用适配所产生的仿真文件作精确的时序仿真，同时产生可用于对目标器件进行编程的文件。

1.5.4　仿真

在编程下载前必须利用 EDA 工具对适配生成的结果进行模拟测试，就是所谓的仿真。仿真就是让计算机根据一定的算法和一定的仿真库对 EDA 设计进行模拟，以验证设计的正确性，排除错误。仿真是在 EDA 设计过程中的重要步骤。图 1-2 所示的时序与功能门级仿真通常由 FPGA 公司的 EDA 开发工具直接提供(当然也可以选用第三方的专业仿真工具)，它可以完成两种不同级别的仿真测试。

1. 时序仿真

时序仿真，就是接近真实器件时序性能运行特性的仿真。仿真文件中已包含了器件硬件特性参数，因而，仿真精度高。但时序仿真的仿真文件必须来自针对具体器件的适配器。综合后所得的 EDIF 等网表文件通常作为 FPGA 适配器的输入文件，产生的仿真网表文件中包含了精确的硬件延迟信息。

2. 功能仿真

功能仿真，是直接对 HDL、原理图描述或其他描述形式的逻辑功能进行测试模拟，以了解其实现的功能是否满足原设计要求的过程，仿真过程不涉及任何具体器件的硬件特性。不经历适配阶段，在设计项目编辑编译(或综合)后即可进入门级仿真器进行模拟测试。直接进行功能仿真的好处是设计耗时短，对硬件库、综合器等没有任何要求。

1.5.5　目标器件的编程/下载

如果编译、综合、布线/适配、行为仿真、功能仿真和时序仿真等过程都没有发现问题，即满足原设计的要求，则可以将由 FPGA/CPLD 布线/适配器产生的配置/下载文件通过编程器或下载电缆载入目标芯片 FPGA 或 CPLD 中。

1.6　可编程逻辑器件

可编程逻辑器件 PLD(Programmable Logic Devices)是 20 世纪 70 年代发展起来的一种新型集成器件。PLD 是大规模集成电路技术发展的产物，是一种半定制的集成电路，结合

EDA 技术可以快速、方便地构建数字系统。

数字电路系统都是由基本门来构成的，如与门、或门、非门、传输门等。由基本门可构成两类数字电路，一类是组合电路，另一类是时序电路，它含有存储元件。事实上，不是所有的基本门都是必需的，如用与非门单一基本门就可以构成其他的基本门。任何的组合逻辑函数都可以化为“与-或”表达式，即任何的组合电路可以用“与门-或门”二级电路实现。同样，任何时序电路都可由组合电路加上存储元件，即锁存器、触发器来构成。由此，人们提出了一种可编程电路结构，即可重构的电路结构。

1.6.1　PLD 的分类

可编程逻辑器件的种类很多，几乎每个大的可编程逻辑器件供应商都能提供具有自身结构特点的 PLD 器件。由于历史的原因，可编程逻辑器件的命名各异，在详细介绍可编程逻辑器件之前，有必要介绍几种 PLD 的分类方法。

1. 按集成度划分

第一种分类方法是以集成度划分，一般可分为以下两大类器件：

(1) 低集成度芯片。早期出现的 PROM(Programmable Read OnlyMemory)、PAL(Programmable Array Logic)、可重复编程的 GAL(Generic Array Logic)都属于这类，可重构使用的逻辑门数大约在 500 门以下，称为简单 PLD。

(2) 高集成度芯片。如现在大量使用的 CPLD、FPGA 器件，称为复杂 PLD。

2. 按结构划分

第二种分类是按结构划分，可分为以下两大类器件：

(1) 乘积项结构器件。其基本结构为“与-或”阵列的器件，大部分简单 PLD 和 CPLD 都属于这个范畴。

(2) 查找表结构器件。由简单的查找表组成可编程门，再构成阵列形式。大多数 FPGA 属于此类器件。

3. 按编程工艺划分

第三种分类方法是按编程工艺划分，可分为以下六大类器件：

(1) 熔丝(Fouse)型器件。早期的 PROM 器件采用熔丝结构，编程过程是根据设计的熔丝图文件来烧断对应的熔丝，达到编程和逻辑构建的目的。

(2) 反熔丝(Anti-fuse)型器件。是对熔丝技术的改进，在编程处通过击穿漏层使得两点之间获得导通，这与熔丝烧断获得开路正好相反。

(3) EPROM 型。称为紫外线擦除电可编程逻辑器件，是用较高的编程电压进行编程，当需要再次编程时，用紫外线进行擦除。Atmel 公司曾经有过此类 PID，目前已被淘汰。

(4) EEPROM 型。即电可擦写编程器件，现有部分 CPLD 及 GAL 器件采用此类结构。它是对 EPROM 工艺的改进，不需要紫外线擦除，而是直接用电擦除。

(5) SRAM 型。即 SRAM 查找表结构的器件，大部分 FPGA 器件都采用此种编程工艺，如 Xilinx 和 Altera 的 FPGA 器件。这种方式在编程速度、编程要求上要优于前 4 种器件，不过 SRAM 型器件的编程信息存放在 RAM 中，在断电后就丢失了，再次上电需要再次编程(配置)，因而需要专用器件来完成这类配置操作。

(6) Flash 型。Actel 公司为了解决上述反熔丝型器件的不足，推出了采用 Flash 工艺的 FPGA，可以实现多次可编程，同时做到掉电后不需要重新配置。现在 Xilinx 和 Altera 的多个系列 CPLD 也采用 Flash 型。

在习惯上，还有另外一种分类方法，即掉电后器件是否需要重新配置，CPLD 不需要重新配置，而 FPGA(大多数)需要重新配置。

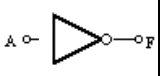
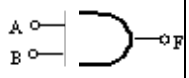
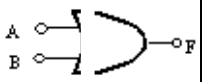
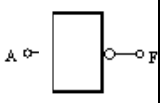
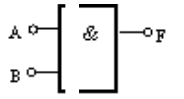
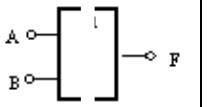
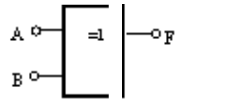
1.6.2　PROM 可编程原理

为了介绍 PLD 器件的可编程原理，在此首先介绍具有典型性的 PROM 结构。但此前有必要熟悉一些常用的逻辑电路符号及常用的描述 PLD 内部结构的专用电路符号。

通常，“国标”是指我国的国家标准，但目前流行于国内高校数字电路教材中的所谓“国标”逻辑符号原本是全盘照搬 ANSI/IEEE-1984 版的 IEC 国际标准符号，且至今没有升级。然而由于此类符号表达形式过于复杂，即用矩形图形逻辑符号(RectangularOutlne Symbols)来标志逻辑功能，故被公认为不适合于表述 PLD 中复杂的逻辑结构。因此数年后，IEEE 又推出了 ANSI/IEEE-1991 标准，于是国际上绝大多数技术资料和相关教材很快就废弃了原标准(1984 版标准)的应用，继而普遍采用了 1991 版本的国际标准逻辑符号。该版本符号的优势和特点是，用图形的不同形状来标志逻辑模块的功能，从而即使图形很小时也能十分容易地辨认出模块的逻辑功能。

本书全部采用 IEEE-1991 标准符号。如表 1-1 为两版符号的比较。

表 1-1　两种不同版本的国际标准逻辑门符号对照表

逻　辑　门	非　　门	与　　门	或　　门	异　或　门
IEE -1991 版标 逻辑符号				
IEE -1984 版标 逻辑符号				
逻 表达式	F= N)T　A	F=A • 3	F=A+B	F=A ⊕ B

在目前流行的 EDA 软件中，原理图中的逻辑符号也都采用了 ANSI/IEEE-1991 标准。由于 PLD 的复杂结构，用 1991 标准的符号的好处是能十分容易地衍生出一套用于描述 PLD 复杂逻辑结构的简化符号。如图 1-3 所示，接入 PLD 内部的与或阵列输入缓冲器电路，一

般采用互补结构，它等效于如图 1-4 所示的逻辑结构，即当信号输入 PLD 后，分别以其同相和反相信号接入。如图 l-5 所示的是 PLD 中与阵列的简化图形，表示可以选择 A、B、C 和 D 4 个信号中的任一组或全部输入与门。在这里用于形象地表示与阵列，这是在原理上的等效。当采用某种硬件实现方法时，如 NMOS 电路时，在图中的与门可能根本不存在。但 NMOS 构成的连接阵列中却含有了与的逻辑。同样，或阵列也用类似的方式表示。如图1-6 所示的是 PLD 中或阵列的简化图形表示。如图 1-7 所示的是在阵列中连接关系的表示。交叉线的交点上打黑点，表示固定连接，即在 PLD 出厂时已连接；交叉线的交点上打叉，表示该点可编程，在 PLD 出厂后通过编程，其连接可随时改变；十字交叉线表示两条线未连接。

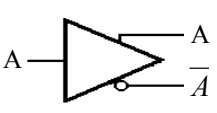


图 1-3　PLD 的互补缓冲器

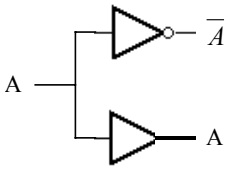


图 1-4　PLD 的互补输入

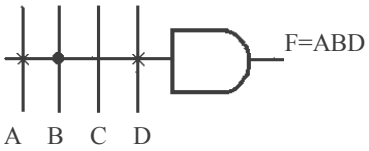


图 1-5　PLD 中与阵列的表示

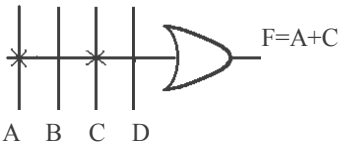


图 1-6　PLD 中或阵列的表示

图 1-7　阵列线连接表示

PROM 作为可编程只读存储器，其 ROM 除了用作只读存储器外，还可作为 PLD 使用。

一个 ROM 器件主要由地址译码部分、ROM 单元阵列和输出缓冲部分构成。对于 PROM，也可以从可编程逻辑器件的角度来分析其基本结构。为了更清晰、直观地表示 PROM 中固定的与阵列和可编程的或阵列，PROM 可以表示为 PLD 阵列图，以 4×2 PROM 为例，如图 l-8 所示。

下式是已知半加器的逻辑表达式，可用 4×2PROM 编程实现。

$S=A_0 \oplus A_1$

$C=A_0 \cdot A_1$

图 1-9 的连接结构表达的是半加器逻辑阵列：

$F_0= A_0 \overline{A}_1 + \overline{A}_0 A_1$

$F_1=A_1A_0$

上述二式是图 1-9 结构的布尔表达式，即所谓的“乘积项”方式。式中的 A_1 和 A_0 分别是加数和被加数：F_0 为和，F_1 为进位。反之，根据半加器的逻辑关系，就可以得到图 1-9 的阵列点连接关系，从而可以形成阵列点文件，这个文件对于一般的 PLD 器件称为熔丝图文件(Fuse Map)。对于 PROM，则为存储单元的编程数据文件。

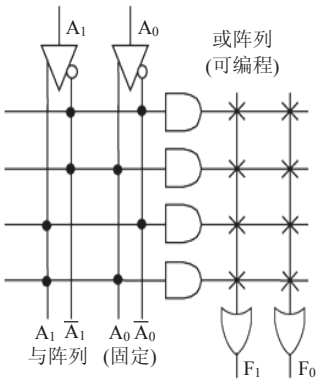


图 1-8　PROM 表达的 PLD 阵列图

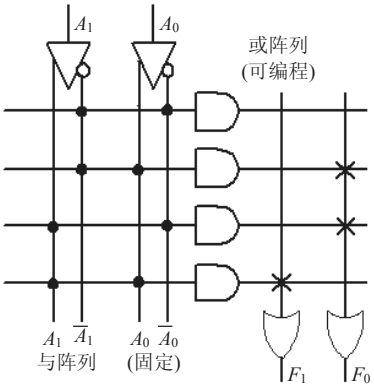


图 1-9　用 PROM 完成半加器逻辑阵列

PROM 只能用于组合电路的可编程用途上。输入变量的增加会引起存储容量的增加，这种增加是按 2 的幂次增加的，所以多输入变量的组合电路函数是不适合用单个 PROM 来编程表达的。

1.6.3　GAL

1985 年，Lattice 公司在 PAL 的基础上设计出了GAL 器件，即通用阵列逻辑器件。GAL 首次在 PLD 上采用 EEPROM 工艺，使得 GAL 具有电可擦除重复编程的特点，彻底解决了熔丝型可编程器件的一次可编程问题。GAL 在与或阵列结构上沿用了 PROM 的与阵列可编程、或阵列固定的结构，如图 1-10 所示，但对 PROM 的 I/O 结构进行了较大的改进，在 GAL 的输出部分增加了输出逻辑宏单元 OLMC(Output Logic Macro Cell)，此结构使得 PLD 器件在组合逻辑和时序逻辑的可编程性或可重构性都成为可能。

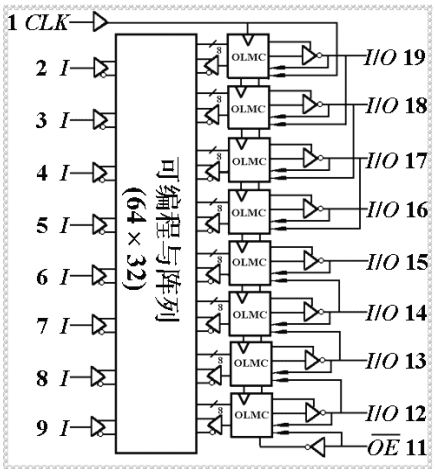


图 1-10　GAL16V8 逻辑图

图 l-10 所示的是 GAL16V8 型号的器件，它包含了 8 个逻辑宏单元 OLMC，每一个 OLMC 可实现时序电路可编程，而其左侧的电路结构是与阵列可编程的组合逻辑可编程结构。专业习惯是将 OLMC 及左侧的可编程与阵列合称为一个逻辑宏单元，即标志 PLD 器

件逻辑资源的最小单元，由此可以认为 GAL16V8 器件的逻辑资源是 8 个逻辑宏单元，而目前最大的 FPGA 的逻辑资源达数十万个逻辑宏单元。也有将逻辑门的数量作为衡量逻辑器件资源的最小单元，如某 CPLD 的资源约 2000 门等，但此类划分方法误差较大。

GAL 的 OLMC 单元设有多种组态，可配置成专用组合输出、专用输入、组合输出双向口、寄存器输出、寄存器输出双向口等，为逻辑电路设计提供了极大的灵活性。由于具有结构重构和输出端的功能均可移到另一输出引脚上的功能，在一定程度上简化了线路板的布局布线，使系统的可靠性进一步提高。

图 l-10 中，GAL 的输出逻辑宏单元 OLMC 中含有 4 个多路选择器，通过不同的选择方式可以产生多种输出结构，分别属于 3 种模式，一旦确定了某种模式，所有的 OLMC 都将工作在同一模式下。

1.7　CPLD 的结构与可编程原理

CPLD 即复杂可编程逻辑器件(Complex Programmable Logic Device)，是伴随着半导体工艺不断完善，用户对集成度要求不断提高的形势下发展起来的。最初是在 EPROM 和GAL 的基础上推出可擦除可编程逻辑器件，也就是 EPLD(Erasable PLD)，其基本结构与 PAL/GAL 相仿，但集成度要高得多。近年来器件的密度越来越高，所以许多公司把原来的 EPLD 产品名称改为 CPLD，但为了与 FPGA、isp-PLD 加以区别，一般把采用 EPROM 结构实现大规模的 PLD 称为 CPLD。

当前 CPLD 的规模已取代 PAL 和 GAL 超过 500 门以下的芯片系列，发展到 500 门以上，现在已有百万门级的 CPLD 芯片系列。随着工艺水平的提高，在增加器件容量的同时，为提高器件的利用率和工作频率，CPLD 从内部结构上作了许多改进，出现了多种不同的形式，其功能更加齐全，应用不断扩展。

1.7.1　CPLD 的基本结构

CPLD 器件种类繁多，特点各异，共同之处包括 3 部分：一个二维的逻辑块阵列，构成 PLD 的逻辑核心；输入/输出块；连接逻辑块的互连资源，用于逻辑块之间、逻辑块与输入/输出块之间的连接。除这 3 个基本部件之外，不同的厂家有各自的特点。

如图 1-11 所示的是 CPLD 的结构原理图，它包括可编程逻辑功能块(FB)；可编程 I/O 单元；可编程内部连线。FB 中包含乘积项、宏单元等。

目前，世界上主要的半导体器件公司，如 Altera、Xilinx 和 Lattice 等，都生产 CPLD 产品。不同的 CPLD 有各自的特点，但总体结构大致相似。本节将以 Altera 公司系列器件为例来介绍 CPLD 的基本原理和结构。

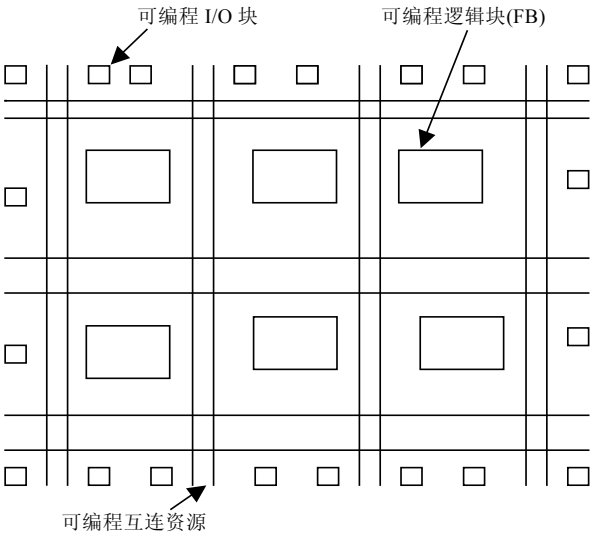


图 1-11　CPLD 的结构原理图

1.7.2　逻辑阵列宏单元

在较早的 CPLD 中，由结构相同的逻辑阵列组成宏单元模块。一个逻辑阵列单元的基本结构如图 1-12 所示。输入项由专用输入端和 I/O 端组成，而来自 I/O 端口的输入项，可通过 I/O 结构控制模块的反馈选择，可以是 I/O 端信号的直接输入，也可以是本单元输出的内部反馈。所有输入项都经过缓冲器驱动，并输出其输入的原码及补码。图 1-12 中所有的竖线为逻辑单元阵列的输入线，每个单元各有 9 条横向线，称为积项线(或称为乘积项)。在每条输入线和积项线的交叉处设有一个 EPROM 单元进行编程，以实现输入项与乘积项的连接关系，这样使得逻辑阵列中的与阵列是可编程的。其中，8 条积项线用作或门的输入，构成一个具有 8 个积项和的组合逻辑输出；另一条积项线(OE 线)连到本单元的三态输出缓冲器的控制端，以 I/O 端作输出、输入或双向输出等工作方式。可以看出，早期 CPLD 中的逻辑阵列结构与 PAL、GAL 中的结构极为类似，只是用 EPROM 单元取代了 PAL 中的熔丝和 GAL 中的 E2PROM 单元。和 GAL 器件一样，可实现擦除和再编程功能。

在基本结构中，每个或门有固定乘积项(8 个)，也就是说，逻辑阵列单元中的或阵列是固定的、不可编程的，因而这种结构的灵活性差。据统计，实际工作中常用到的组合逻辑，约有 70%是只含 3 个乘积项及 3 个以下的积项和。另一方面，对遇到复杂的组合逻辑所需的乘积项可能超过 8 个，这又要用两个或多个逻辑单元来实现，器件的资源利用率不高。为此，目前的 CPLD 在逻辑阵列单元结构方面作了很大改进，下面讨论几种改进的结构形式。

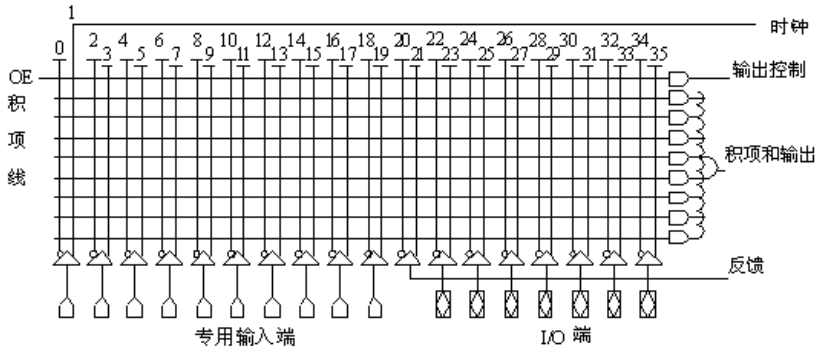


图 1-12　逻辑阵列单元的基本结构

1. 具有两个或项输出的逻辑阵列单元

如图 1-13 所示的是具有两个固定积项和输出的 CPLD 的结构图。

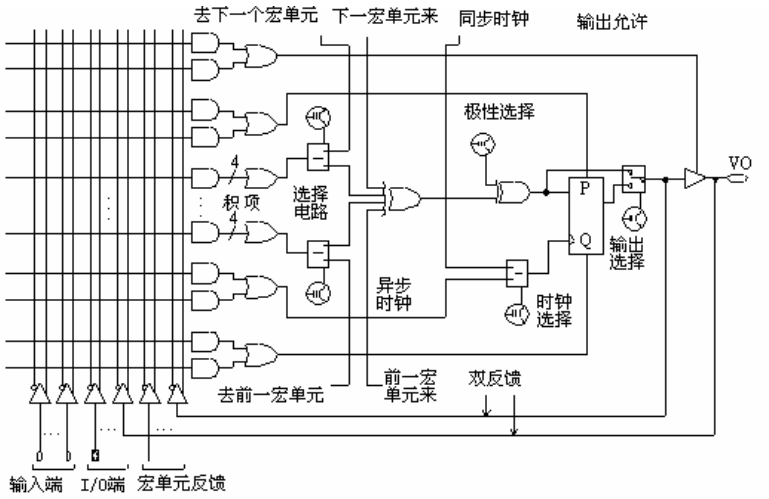


图 1-13　具有两个固定积项和输出的 CPLD 结构图

由图可见，每个单元中含有两个或项输出，而每个或项均有固定的 4 个乘积项输入。为提高内部各或项的利用率，每个或项的输出均先送到一个由 EPROM 单元可编程控制的 1 分 2 选择电路，即阵列单元中上面的或项输出由选择电路控制，既可输送到本单元中第 2 级或门的输入端，也可馈送到相邻的下一个阵列单元第 2 级或门的输入端；同样，阵列单元中下面的或项输出由选择电路控制，可直接送到本单元第 2 级或门的输入端，也可馈送到相邻的前一个阵列单元中的第 2 级或门输入端，使本单元不用的或项放到另一单元中发挥其作用。因而每个逻辑阵列单元又可共享相邻单元中的乘积项，使每个阵列可具有 4、8、12 和 16 四种组合的积项和输出，甚至本单元中的两个或项都可用于相邻的两个单元中。这样，既提高了器件内部各单元的利用率，又可实现更为复杂的逻辑功能。以这种逻辑单元结构实现的代表 EPLD 有 Altera 公司的 EP512 器件等。

在 Atmel 公司的 ATV750 等器件结构调整中，每个逻辑单元中也含有两个或项，但不同单元中构成或项的积项数却不同，它是分别由 4、5、6、7 和 8 个乘积项输入到两个或门

所组成的 5 对阵列单元构成的组合阵列。每个单元中的两个或项输出通过输出逻辑模块中的选择电路控制，可实现各自独立的输出，也可将两个或项再“线或”起来实现功能更为复杂的组合逻辑输出，但各个阵列单元中的或项不能为相邻的阵列单元所共享。

2. 功能更多、结构更复杂的逻辑阵列单元

随着集成规模和工艺水平的提高，出现了大批结构复杂、功能更多的逻辑阵列单元形式，如 Altera 公司的 EP1810 器件采用了全局总线和局部总线相结合的可编程逻辑宏单元结构；采用多阵列矩阵(Multiple Array Matrix，MAX)结构的大规模 CPLD 器件，如 Altera 公司的 EPM 系列和 Atmel 公司的 ATV5000 系列器件；采用通用互连矩阵(Universal Interconnect Matrix，UIM)及双重逻辑功能块结构的逻辑阵列单元，如 Xilinx 公司的 XC7000 和 XC9500 系列产品。

1.7.3　I/O 控制模块

CPLD 中的 I/O 控制模块，根据器件的类型和功能不同，可有各种不同的结构形式，但基本上每个模块都由输出极性转换电路、触发器和输出三态缓冲器 3 部分及与它们相关的选择电路所组成。下面介绍在 CPLD 中广泛采用的几种 I/O 控制模块。

1. 与 PAL 器件相兼容的 I/O 模块

与 PAL 器件相兼容的 I/O 模块如图 1-14 所示。可编程逻辑阵列中每个逻辑阵列逻辑单元的输出都通过一个独立的 I/O 控制模块接到 I/O 端，通过 I/O 控制模块的选择实现不同的输出方式。根据编程选择，各模块可实现组合逻辑输出和寄存器输出方式。

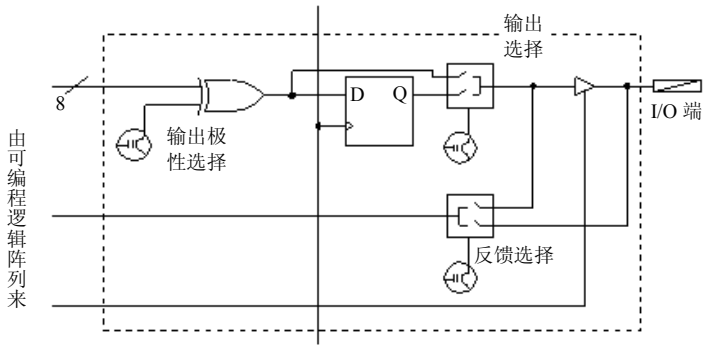


图 1-14　与 PAL 器件相兼容的 I/O 模块

2. 与 GAL 器件相兼容的 I/O 模块——输出宏单元

与 GAL 器件相兼容的 I/O 模块——输出宏单元如图1-15 所示，从逻辑阵列单元输出的积项和首先送到输出宏单元(Output Macro Cell，OMC)的输出极性选择电路，由 EPROM 单元构成的可编程控制位来选择该输出极性(原码或它的补码)。每个OMC 中还有由 EPROM 单元构成的两个结构控制位，根据构形单元表，OMC 可实现 4 种不同的工作方式，即寄存

器输出、双向 I/O 组合方式、固定输出和固定输入方式。

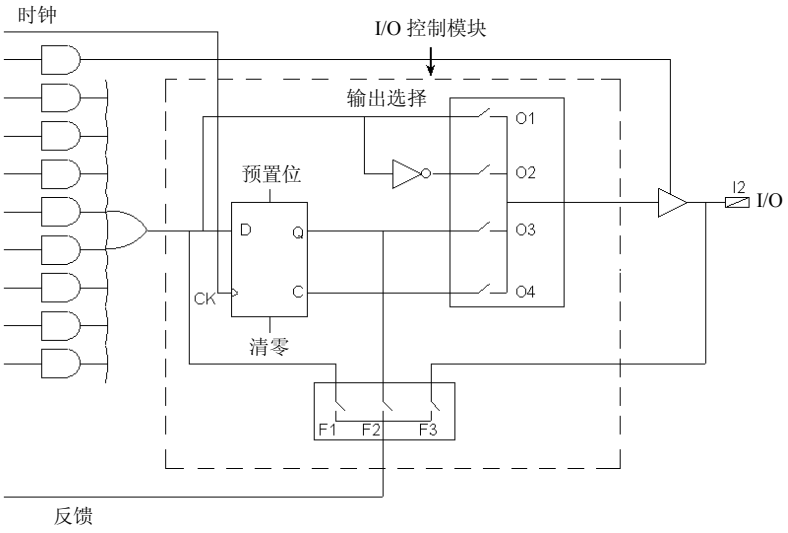


图 1-15　与 GAL 器件相兼容的 I/O 模块——输出宏单元

3. 触发器可编程的 I/O 模块

为了进一步改善 I/O 控制模块的功能，对 I/O 模块中的触发器电路进行改进并由 EPROM 单元进行编程，可实现不同类型的触发器结构，即 D、T、JK、RS 等类型的触发器，如图 1-16 所示。这种改进的 I/O 控制模块，可组合成高达 50 种的电路结构。

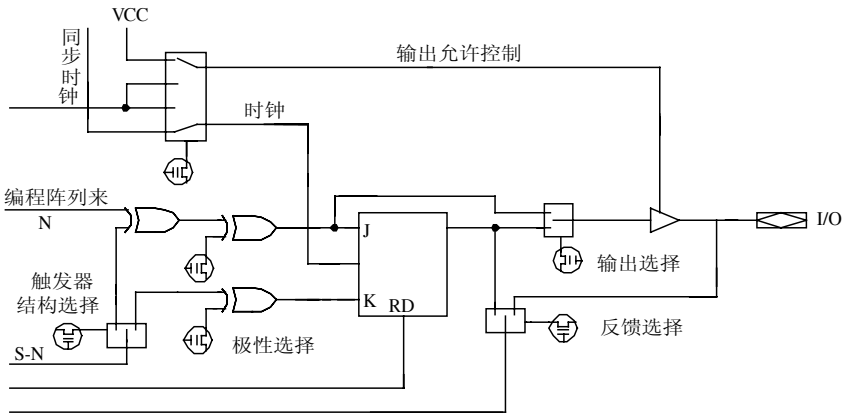


图 1-16　触发器可编程的 I/O 模块

4. 具有三路积项和输入与两个触发器的 I/O 控制模块

具有三路积项和输入与两个触发器的 I/O 控制模块如图 1-17 所示，每个 I/O 模块可接受三路积项和输入，每路各有 4 个乘积项。利用 EPROM 控制单元的编程，可实现下列功能：

- 一路积项和的输出直接馈送到 I/O 端，而另两路积项和的输出则分别馈送到两个触发器的输入端 D1 和 D2，它们的输出均可为“内藏”工作方式，通过编程控制可反馈到逻辑阵列总线中去。

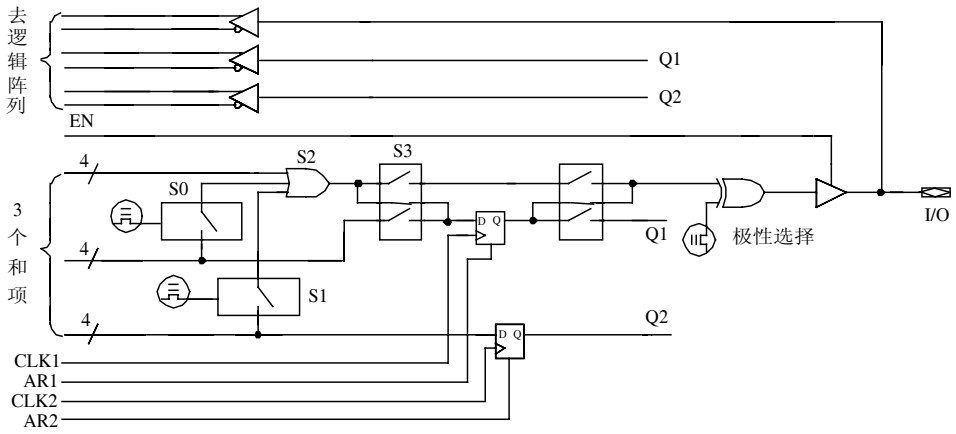


图 1-17　具有三路积项和输入与两个触发器的 I/O 控制模块

- 在实现组合逻辑输出或寄存器方式输出之前，三路和项还可以通过编程组合在一起，以实现高达 12 个积项和的组合逻辑输出或寄存器输出。
- 在组合逻辑输出方式中，通过编程控制可实现 4、8 或 12 个积项和的组合逻辑输出，而模块中的中、下两路和项仍可分别馈送到两个触发器的 D1 和 D2 端，它们的输出 Q1 和 Q2 为“内藏”工作方式，可通过编程反馈到逻辑阵列总线中去。
- 在寄存器输出方式中，上、中两路组合成 8 个积项和自动馈送到触发器 D1 输入端，而下路的和项除馈送到触发器 D2 输入端为“内藏”工作方式外，还可与 D1 共享。
- 两个触发器均可有各自的异步复位和时钟信号，即 AR1、CLK1 和 AR2、CLK2，它们由编程逻辑阵列中的 4 条积项线提供。
- 输出三态缓冲器的控制信号由来自编程逻辑阵列的一条积项线提供。
- 当 I/O 端作输入端使用，或 I/O 模块的输出反馈到逻辑阵列总线中去时，均通过同一个反馈缓冲器输出它们的同相和反相两路信号，馈送到逻辑阵列总线中去，而两个触发器的输出 Q1 和 Q2 则通过各自的反馈缓冲器，将它们的信号(同相及反相信号)馈送到逻辑阵列总线中去。

1.7.4　可编程连线阵列

可编程连线阵列 PIA 的作用是在各逻辑宏单元之间及逻辑宏单元和 I/O 单元之间提供互联网络。各逻辑宏单元通过可编程连线阵列接收来自专用输入/输出端的信号，并将宏单元的信号反馈到其需要到达的 I/O 单元或其他宏单元，这种互连机制有很大的灵活性，它允许在不影响引脚分配的情况下改变内部的设计。

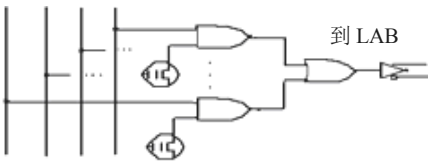


图 1-18　PIA 布线示意图

如图 1-18 所示的是 PIA 布线示意图。CPLD 的 PIA 布线具有可累加的延时，这使得 CPLD 的内部延时是可预测的，从而带来较好的时序性能。

1.8　FPGA 的结构与可编程原理

FPGA(Field Programmable Gate Array)是大规模可编程逻辑器件的另一大类 PLD 器件，而且其逻辑规模比 CPLD 大得多，应用领域也要宽得多。以下介绍最常用的 FPGA 的结构及其工作原理。

1.8.1　FPGA 的结构描述

FPGA 采用类似掩膜可编程门阵列的结构，并结合可编程逻辑器件的特性，即继承了门阵列逻辑器件密度高和通用性强的优点，又具备可编程逻辑器件的可编程特性。它的结构可以分为 3 个部分：可编程逻辑块(CLB)、可编程 I/O 模块和可编程内部连线。

Xilinx 公司的 FPGA 芯片分为 XC2000、XC3000/XC3100、XC4000、XC5000、XC6200、XC8100、Spartan、Virture 等系列。前 3 个系列是三代渐进而兼容的 FPGA 产品，它们包含多种规格，如密度大小、速度高低、温度范围、封装形式等，形成了系列产品。而接下来的 3 个系列是 1995 年推出的产品。其中，XC5000 系列的结构是对 0.6 μm 三层金属丝(TLM)处理工艺优化的第一个 FPGA 系列，其结果是在硅片的利用效率上有惊人的突破，使 XC5000 系列能承受大于 5000 门的各种设计，提供了高密度、低成本的最佳方案。XC6200 系列主要是针对计算机中可反复配置的协处理器而设计的。XC8100 系列是一次可编程的(OTP)，它利用 MicroVia 处理工艺(一种 CMOS，金属到金属的反熔丝和 3 层金属的组合，编程是由 Xilinx 或第三方的编程器来完成，类似于一次编程的 PLD 器件)，提供了低成本和高保密性能的最佳结合，可用于空间通信、工业控制等高保密和高速初始化应用的场合及批量或定型产品中。最后两个系列是 1998 年新推出的大容量、高密度产品，最大门数已达 100 万门量级。每种 FPGA 器件都有专用 LSIC 的功能特征，都是用户可编程和反复可编程的(XC8100 系列除外)。

FPGA 器件的内部结构为逻辑单元阵列(LCA)。LCA 由 3 类可编程单元组成：周边的输入/输出模块(IOB)、核心阵列是可配置逻辑块(CLB)以及各模块间的互连资源。FPGA 的结构原理如图 1-19 所示。

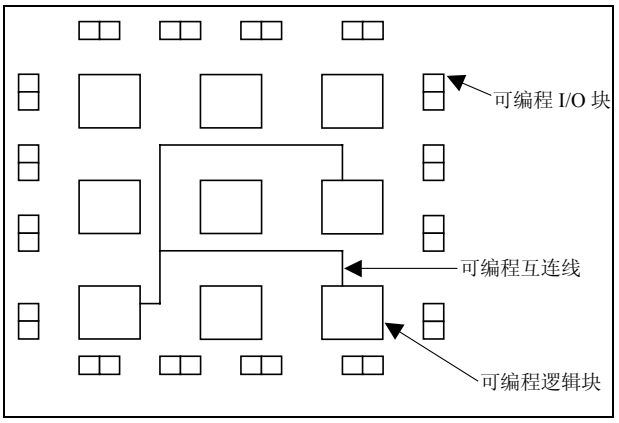


图 1-19　FPGA 的结构原理图

1.8.2　查找表逻辑结构

前面提到的可编程逻辑器件，诸如 GAL/CPLD 之类都是基于乘积项的可编程结构，即由可编程的与阵列和固定的或阵列组成。而在本节中将要介绍的 FPGA，使用了另外可编程逻辑的形成方法，即可编程的查找表(Look Up Table，LUT)结构。LUT 是可编程的最小逻辑构成单元。大部分 FPGA 采用基于 SRAM(静态随机存储器)的查找表逻辑形成结构，就是用 SRAM 来构成逻辑函数发生器。一个 N 输入 LUT 可以实现 N 个输入变量的任何逻辑功能，如 N 输入“与”、N 输入“异或”等。如图 1-20 所示的是 4 输入 LUT，其内部结构如图 1-21 所示。

一个 N 输入的查找表，需要 SRAM 存储 N 个输入构成的真值表，需要用 2 的 N 次幂个位的 SRAM 单元。显然 N 不可能很大，否则 LUT 的利用率很低，输入多于 N 个的逻辑函数，必须用数个查找表分开实现。Xlinx 的 Vinex-6、Spartan-3E、Spartan-6 系列，Altera 的 Cyclone、CycloneⅡ/Ⅲ/Ⅳ、Stratix-3、Stratix-4 等系列都采用 SRAM 查找表构成，是典型的 FPGA 器件。

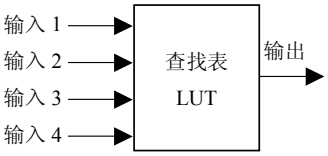


图 1-20　FPGA 查找表单元

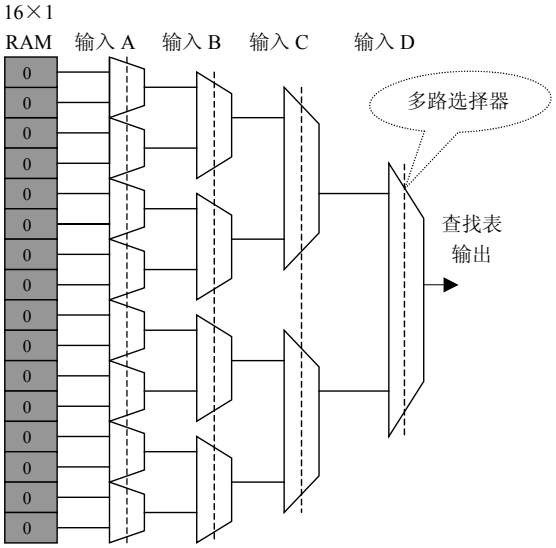


图 1-21　FPGA 查找表单元内部结构

1.8.3　Cyclone III 系列器件与工作原理

考虑到本书之后给出的实训项目中主要以 Cyclone Ⅲ系列 FPGA 为主，且其结构和工作原理也具有典型性，故下面简要介绍此类器件的结构与工作原理。

Cyclone III 系列器件是 Altera 公司近年推出的一款低功耗、高性价比的 FPGA，它主要由逻辑阵列块(LAB)、嵌入式存储器块、嵌入式硬件乘法器、I/O 单元和嵌入式 PLL 等模块构成，在各个模块之间存在着丰富的互连线和时钟网络。Cyclone III 器件的可编程资

源主要来自逻辑阵列块(LAB)，而每个 LAB 都由多个逻辑宏单元 LE(Logic Element)构成。LE 是 Cyclone III FPGA 器件的最基本的对编程单元。如图 1-22 所示的是 Cyclon III FPGA 的 LE 的内部结构。

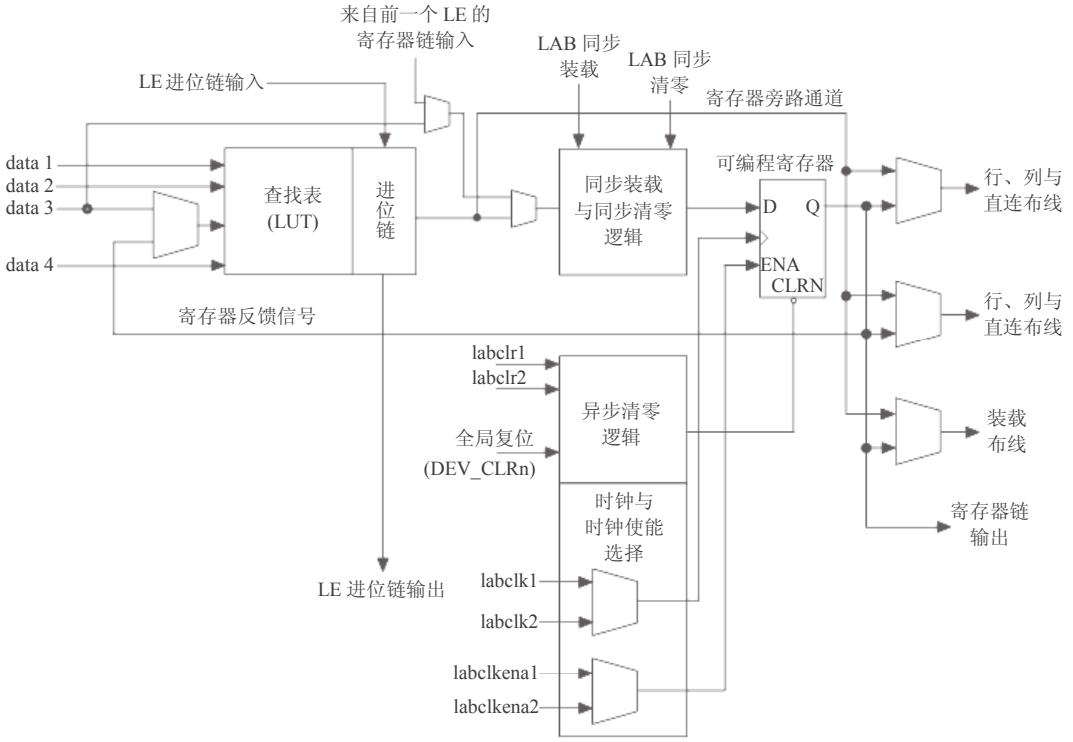


图 1-22　Cyclon Ⅲ FPGA 的 LE 的内部结构图

观察图 1-22 可以发现，LE 主要由一个 4 输入的查找表 LUT、进位链逻辑、寄存器链逻辑和一个可编程的寄存器构成。4 输入的 LUT 可以完成所有的 4 输入 1 输出的组合逻辑功能。每一个 LE 的输出都可以连接到行、列、直连通路、进位链、寄存器链等布线资源。

每个 LE 中的可编程寄存器可以被配置成 D 触发器、T 触发器、J K 触发器和 RS 寄存器模式。每个可编程寄存器具有数据、时钟、时钟使能、清零输入信号、全局时钟网络、通用 I/O 口以及内部逻辑可以灵活配置寄存器的时钟和清零信号。任何一个通用 I/O 和内部逻辑可以驱动时钟使能信号。在一些只需要组合电路的应用中，对于组合逻辑的实现，可将该可配置寄存器旁路，LUT 的输出可作为 LE 的输出。

LE 有 3 个输出驱动内部互连：一个驱动局部互连，另两个驱动行或列的互连资源。LUT 和寄存器的输出可以单独控制，可以实现在一 LE 中，LUT 驱动一个输出，而寄存器驱动另一个输出(这种技术称为寄存器打包)，因而在一个 LE 中的寄存器和 LUT 能够用来完成不相关的功能，能够提高 LE 的资源利用率。

寄存器反馈模式允许在一个 LE 中寄存器的输出作为反馈信号，加到 LUT 的一个输入上，在一个 LE 中就完成反馈。

除上述的 3 个输出外，在一个逻辑阵列块中的 LE 还可以通过寄存器链进行级联。在同一个 LAB 中的 LE 里的寄存器可以通过寄存器链级联在一起，构成一个移位寄存器，那

些 LE 中的 LUT 资源可以单独实现组合逻辑功能，两者互不相关。

Cyclone III 的 LE 可以工作在两种操作模式下，即普通模式和算术模式。

普通模式下的 LE 适合通用逻辑应用和组合逻辑的实现。在该模式下，来自 LAB 局部互连的 4 个输入将作为一个 4 输入 1 输出的 LUT 的输入端口。可以选择进位输入(cin)信号或者 data3 信号作为 LUT 中的一个输入信号。每一个 LE 都可以通过 LUT 链直接连接到(在同一个 LAB 中的)下一个 LE。在普通模式下，LE 的输入信号可以作为 LE 中寄存器的异步装载信号。普通模式下的 LE 也支持寄存器打包与寄存器反馈。

在 Cyclone III 器件中的 LE 还可以工作在算术模式下，在这种模式下，可以更好地实现加法器、计数器、累加器和比较器。在算术模式下的单个 LE 内有两个 3 输入 LUT，可被配置成一位全加器和基本进位链结构。其中一个 3 输入 LUT 用于计算，另外一个 3 输入 LUT 用来生成进位输出信号 cout。在算术模式下，LE 支持寄存器打包与寄存器反馈。逻辑阵列块 LAB 是由一系列相邻的 LE 构成的。每个 Cyclone III 的 LAB 包含 16 个 LE，在 LAB 中、LAB 之间存在着行互连、列互连、直连通路互连、LAB 局部互连、LE 进位链和寄存器链。

在 Cyclone III FPGA 器件中所含的嵌入式存储器(Embedded Memory)，由数十个 M9K 的存储器块构成。每个 M9K 存储器块具有很强的伸缩性，可以实现的功能有：8192 位 RAM(单端口、双端口、带校验、字节使能)、ROM、移位寄存器、FIFO 等。在 Cyclone III FPGA 中的嵌入式存储器可以通过多种连线与可编程资源实现连接，这大大增强了 FPGA 的性能，扩大了 FPGA 的应用范围。

在 Cyclone III 系列器件中还有嵌入式乘法器(Embedded Multiplier)，这种硬件乘法器的存在可以大大提高 FPGA 在处理 DSP(数字信号处理)任务时的能力。嵌入式乘法器可以实现 9×9 乘法器或者 18×18 乘法器，乘法器的输入与输出可以选择是寄存的还是非寄存的(即组合输入输出)。可以与 FPGA 中的其他资源灵活地构成适合 DSP 算法的 MAC(乘加单元)。

在数字逻辑电路的设计中，时钟、复位信号往往需要同步作用于系统中的每个时序逻辑单元，因此在 Cyclone III 器件中设置有全局控制信号。由于系统的时钟延时会严重影响系统的性能，故在 Cyclone III 中设置了复杂的全局时钟网络，以减少时钟信号的传输延迟。另外，在 Cyclone III FPGA 中还含有 2~4 个独立的嵌入式锁相环 PLL，可以用来调整时钟信号的波形、频率和相位。PLL 的使用方法将在后续章节中介绍。

Cyclone III 的 I/O 支持多种 I/O 接口，符合多种 I/O 标准，可以支持差分的 I/O 标准，诸如 LVDS(低压差分串行)和 RSDS(去抖动差分信号)、SSTL-2、SSTL-l8、HSTL-18、HSTL-15、HSTL-12、PPDS、差分 LVPECL， 当然也支持普通单端的 I/O 标准，如 LVTTL、LVCMOS、PCI 和 PCI-X I/O 等，通过这些常用的端口与板上的其他芯片沟通。

Cyclone Ⅲ器件还可以支持多个通道的 LVDS 和 RSDS。Cyclone Ⅲ器件内的 LVDS 缓冲器可以支持最高达 875Mbps 的数据传输速度。与单端的 I/O 标准相比，这些内置于 Cyclone Ⅲ器件内部的 LVDS 缓冲器保持了信号的完整性，并具有更低的电磁干扰、更好的电磁兼容性(EMI)及更低的电源功耗。

如图 1-23 所示的是 Cyclone III 器件内部的 LVDS 接口电路。

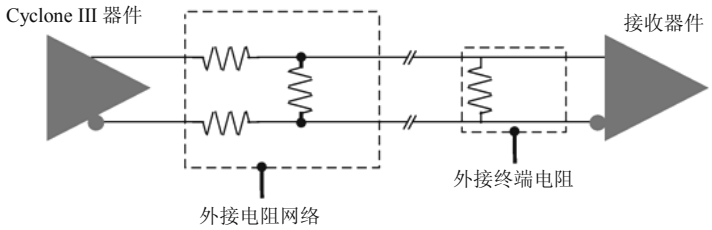


图 1-23　LVDS 连接

Cyclone III 系列器件除了片上的嵌入式存储器资源外，可以外接多种外部存储器，比如 SRAM、NAND、SDRAM、DDR SDRAM、DDR2 SDRAM 等。

Cyclone III 的电源支持采用内核电压和 I/O 电压(3.3V)分开供电的方式，I/O 电压取决于使用时需要的 I/O 标准，而内核电压使用 1.2V 供电，PLL 供电 2.5V。

Cyclone III 系列中有一个子系列是Cyclone III LS 系列，该器件系列可以支持加密功能，使用 AES 加密算法对 FPGA 上的数据进行保护。

1.9　硬件测试技术

进入 21 世纪，集成电路技术飞速发展，推动了半导体存储、微处理器等相关技术的飞速发展，CPLD/FPGA 也在其列。CPLD、FPGA 和 ASIC 的规模和复杂程度同步增加，在 CPLD/FPGA 应用中，测试显得越来越重要。由于其本身技术的复杂性，测试也分多个部分：在“软”的方面，逻辑设计的正确性需要验证，这不仅在功能这一级上，对于具体的 CPLD/FPGA 还要考虑种种内部或 I/O 上的时延特性；在“硬”的方面，首先在 PCB 板级需要测试引脚的连接问题，其次是 I/O 的功能也需要专门的测试。

1.9.1　内部逻辑测试

对于 CPLD/FPGA 的内部逻辑测试是应用设计可靠性的重要保证。由于设计的复杂性，内部逻辑测试面临越来越多的问题。设计者通常不可能考虑周全，这就需要在设计时加入用于测试的部分逻辑，即进行可测性设计(Design For Test，DFT)，在设计完成后用来测试关键逻辑。

在 ASIC 设计中的扫描寄存器，是可测性设计的一种，原理是把 ASIC 中关键逻辑部分的普通寄存器用测试扫描寄存器来代替，在测试中可以动态地测试、分析、设计其中寄存器所处的状态，甚至对某个寄存器加以激励信号，改变该寄存器的状态。

有的 CPLD/FPGA 厂商提供一种技术，在可编程逻辑器件中嵌入某种逻辑功能模块。与 EDA 工具软件相配合提供一种嵌入式逻辑分析仪，帮助测试工程师发现内部逻辑问题，Altera 的 SignalTap II 技术是典型代表之一(将在第 4 章中给予详细介绍)。

在内部逻辑测试时，还会涉及测试的覆盖率问题。对于小型逻辑电路，逻辑测试的覆盖率可以很高，甚至达到100%；可是对于一个复杂数字系统设计，内部逻辑覆盖率不可能达到 100%，这就必须寻求别的更有效的方法来解决。

1.9.2　JTAG 边界扫描测试

在 20 世纪 80 年代，联合测试行动组(Joint Test Action Group，JTAG)开发了 IEEE1149.1-1990 边界扫描测试技术规范。该规范提供了有效的测试引线间隔致密的电路板上集成电路芯片的能力。大多数 CPLD/FPGA 厂家的器件遵守 IEEE 规范，并为输入引脚和输出引脚以及专用配置引脚提供了边界扫描测试(Board Scan Test BST)的功能。

边界扫描测试标准 IEEE1149.1BST 的结构，即当器件工作在 JTAG BST 模式时，使用 4 个 I/O 引脚和一个可选引脚 TRST 作为 JTAG 引脚。4 个 I/O 引脚是 TDI、TDO、TMS 和 TCK。如表 1-2 所示的是这些引脚的功能。

表 1-2　边界扫描 I/O 引脚功能

引　脚	描　述	功　能
TDI	测试数据输入	测试指令和编程数据的串行输入引脚，数据在 TCK 的上升沿输入
TDO	测试数据输出	测试指令和编程数据的串行输出引脚，数据在 TCK 的下升沿移出，如果数据没有被移出，该引脚处于高阻态
TMS	测试模式选择	测控信号输入引脚，负责 TAP 控制器的转换。TMS 必须在 TCK 的上升沿到来之前稳定
TCK	测试时钟输入	时钟输入到 BST 电路，一些操作发生在上升沿，而另一些发生在下降沿
TRST	测试复位输入	低电平有效，异步复位边界扫描电路(在 IEEE 规范中，该引脚可选)

1.10　FPGA/CPLD 产品概述

本节将介绍常用的 FPGA 和 CPLD 器件系列及其基本特性，以及 FPGA 的配置器件。

1.10.1　Lattice 公司的 PLD 器件

Lattice 公司是最早推出 PLD 的公司，其 CPLD 产品主要有 ispLSI、ispMACH 等系列。20 世纪 90 年代以来，Lattice 发明了 ISP(In-System Programmability)下载方式，并将电可擦写存储器技术与 ISP 相结合，使 CPLD 的应用领域有了巨大的扩展。

1. ispLSI 系列器件

ispLSI 系列器件是 Lattice 公司于 20 世纪 90 年代初推出的大规模可编程逻辑器件，集成度在 1000~60000 门之间，Pin-to-Pin(引脚至引脚)延时最小可达 3ns。ispLSI 器件支持在

系统编程和 JTAG 边界扫描测试功能。

2. MACHXO 系列

MACHXO 系列非易失性无限重构可编程逻辑器件，是为传统上用 CPLD 或低密度 FPGA 实现的应用而设计的，可以实现通用 I/O 扩展、接口桥接和电源管理功能，提供嵌入式存储器、内置的 PLL、高性能的 LVDS I/O、远程现场升级(TransFRTM 技术)和一个低功耗的睡眠模式。MACHXO 可编程逻辑器件将所有这些功能都集成在单片器件之中。

MACHXO 系列可编程逻辑器件将 SRAM 和内存配置存储器组合在同一个器件中，SRAM 存储单元控制 MACHXO 可编程逻辑器件的逻辑进行工作，闪存用来存储配置数据。宽的数据连接两个存储器。上电时，通过宽总线从片上闪存将配置载入 SRAM。电源稳定后，不到 1ms 的时间即可使用逻辑。

3. ispMACH 4000 系列

ispMACH 4000 系列 CPLD 器件有 3.3V、2.5V 和 1.8V 3 种供电电压，分别属于 ispMACH-4000V、ispMACH4000B 和 ispMACH4000C 器件系列。ispMACH4000C、ispMACH4000V 和 ispMACH4000B 均支持军用温度范围。ispMACH4000 支持介于 3.3V 和 1.8V 之间的 I/O 标准，既有业界领先的速度性能，又能提供最低的动态功耗。

4. LatticeSC FPGA 系列

LatticeSC/M(系统芯片/MACO)FPGA 系列是 Lattice 半导体的高性能 FPGA 系列，集成了一个高性能的 FPGA 结构，其中包括 3.8Gbps SERDES 和 PCS、2Gbps 并行 I/O、低功耗的 1V Vcc 功能选择、大型的嵌入式 RAM 以及嵌入式 ASIC 块。

5. LatticeECP3 FPGA 系列

ECP 系列器件是 Lattice 公司的 FPGA 系列，使用 0.13μm 工艺制造，提供低成本的 FPGA 解决方案。在 ECP 系列器件中还嵌入了 DSP 模块。

1.10.2　Xilinx 公司的 PLD 器件

Xilinx 在 1985 年首次推出了 FPGA，随后不断推出新的集成度更高、速度更快、价格更低、功耗更小的 FPGA 器件系列。Xilinx 以 CoolRunner、XC9500XL 系列为代表的 CPLD，以及 Spartan、Virtex 系列为代表的 FPGA 器件，如 Sparlan-3A N 和 Spartan-6、Virtex-5、Virtex-6 等系列的性能不断提高。

EasyPath 系列是 Xilinx 的结构化 ASIC 产品，其中的 EasyPath-6 FPGA 为高性能 FPGA，实现了最低的总产品所有成本。在设计用户系统时，可以先用 Virtex-6 FPGA 验证，此后无需其他工程即可在 6 周内为生产提供成本更低的器件。

1. Virtex-6 系列 FPGA

Virtex-6 系列是 Xilinx 的高性能 FPGA 系列，采用 40μm 工艺制造，分为 4 个面向特定应用领域而优化的 FPGA 平台架构，它们分别是：Virtex-6 LXT FPGA，面向具有低功耗串行连接功能的高性能逻辑和 DSP 开发；Virtex-6 SXT FPGA，面向具有低功耗串行连接功能的超高性能 DSP 开发；Virtex-6 HXT FPGA，该系列针对需要带宽最高的串行连接功能的通信、交换和成像系统进行了优化设计；Virtex-6 CXT FPGA，面向那些需要 3.75Gbps 串行连接功能和相应的逻辑性能的应用。Virtex-6 LXT FPGA 系列有 74500~758800 个逻辑宏单元，采用 6 输入查找表(LUT)，其 LUT 可以配置成逻辑单元、分布式 RAM(64 位/LUT 或 256 位/CLB)或移位寄存器。此系列使第二代对角对称互连实现了最短、最快的布线。

2. Spartan-6 器件系列

Spartan-6 FPGA 是 Xilinx 的低成本、低功耗 FPGA。第六代 SParlan 系列基于低功耗 45μm、9 金属铜层、双栅极氧化层工艺技术，以及高级功耗管理技术。此系列含最多 150000 个逻辑单元、集成式 PCI Express 模块、高级存储器支持、250MHz DSP Slice 和 3.125Gbps 低功耗收发器。

3. XC9500/XC9500 XL 系列 CPLD

XC9500 系列被广泛地应用于通信、网络和计算机等产品中。该系列器件采用快闪存储技术(FastFlah)，比 EECMOS 工艺的速度更快，功耗更低。目前，Xilinx 公司 XC9500 系列 CPLD 的 tpd 可达到 4ns，宏单元数达到 288 个，系统时钟可达到 200MHz。XC9500 器件支持 PCI 总线规范和 JTAG 边界扫描测试功能，具有在系统可编程能力。该系列有 XC9500、XC9500XV 和 XC9500XL 3 种类型，内核电压分别为 5V、2.5V 和 3.3V。

4. Xilinx Spartan-3A 系列器件

Spartan-3A 系列 FPGA 是 Xilinx 的低成本 FPGA 系列，具有 50000~3400000 个系统门，有 108~502 个 I/O，可以提供集成式 DSP MAC 在内的大量选项，有双功耗管理模式、Device DNA 安全性、多级存储器架构。

1.10.3　Altera 公司的 PLD 器件

Altera 是著名的 PLD 生产厂商，多年来一直占据着行业领先的地位。Altera 的 PLD 具有高性能、高集成度和高性价比的优点，此外它还提供了功能全面的开发工具和丰富的 IP 核、宏功能库等。因此 Altera 的产品获得了广泛的应用。

Altera 的产品有多个系列，按照推出的先后顺序依次为 Classic、MAX(Multiple Array Matrix)、FLEX(Flexible Logic Element Matrix)、APEX(Advanced Logic Element Matrix)、ACEX、APEX II、Cyclone I/II/III/IV、MAX II 以及 Stratix-1/2/3/4/6 等系列。

1. Stratix-4/6 系列 FPGA

Stratix-4 系列 FPGA 器件是 Altera 的高性能 FPGA 系列，采用 TSMC 40nrn 工艺制造最大的一款具有 8200000 个逻辑宏单元、23.1Mb 嵌入式存储器和 1288 个 18×18 嵌入式硬件乘法器。具有两个速率等级优势，以及先进的逻辑和布线体系结构。具有 8.5Gbps 的 48 个高速收发器，或者达到 24 个为 100G 应用优化的 11.3Gbps 收发器，以及 1067Mbps(533 MHz)DDR3 存储器接口。

在 Stratix-6 中含有 PCI Express 硬核 IP 核：Genl(2.5Gbps)和 Gen2(5.0Gbps)，4 个 x8 模块，实现了全端点或者根端口功能。Stratix-4GX 系列 FPGA 在 PCI Express Genl 和 Gen2 (xl、x4 和 x8)上完全符合 PCI-SIG 要求。

2. Cyclone IV 系列 FPGA

Cyclone IV 系列 FPGA 是 Altera 新近推出的低成本 FPGA 系列，该系列实现了低功耗、高性能和低成本的综合，适用于多种通用逻辑应用，可以应用在广播、消费类、工业、无线通信、固网等领域。最大的一款提供 150000 个逻辑单元。Cyclone IV 系列采用经过优化的 60nm 低功耗工艺，拓展了前一代 Cyclone III FPGA 的低功耗优势，最新一代器件降低了内核电压，与前一代产品相比，总功耗降低了 25%。Cyclone IV 的子系列 Cyclone IV GX FPGA 采用了 Altera 成熟的 GX 收发器技术，具有 8 个集成 3.125Gbps 收发器，可以开发功耗不到 1.5W 的 PCI Express 至千兆以太网桥接应用。另外，Cyclone IV 也支持 Nios II 嵌人式处理器软核，可以实现复杂的多 CPU 嵌入式解决方案。

3. Cyclone 和 Cyclone II 系列 FPGA

Altera 的低成本系列 FPGA，平衡了逻辑、存储器、锁相环和高级 I/O 接口。Cyclone II FPGA 适合于价格敏感的应用。Cyclone II FPGA 具有以下特性：

- 新的可编程构架通过设计实现低成本。
- 嵌入式存储资源支持各种存储器应用和数字信号处理(DSP)实施。
- 支持串行总线和网络接口及各种通信协议。
- 使用 PLL 管理片内和片外系统时序。
- 支持单端 I/O 标准和差分 I/O 技术，支持 LVDS 信号。
- Cyclone II 器件最大的一款具有 68416 个逻辑单元、1.1Mb 可用于嵌入式处理器的通用存储单元、150 个 18×18 位的可用于嵌入式处理器的低成本硬件 DSP 模块。
- 专用外部存储器接口电路，用于连接 DDR2、DDR 和 SDR SDRAM 以及 QDR II SRAM 存储器件。
- 最多 4 个嵌入式 PLL，用于片内和片外系统时钟管理。
- 支持单端 I/O 标准，用于 64 位、66MHz PCI 和 64 位、100MHz PCI-x(模式 1)协议。
- 对安全敏感应用进行自动 CRC 检测。
- 具有支持完全定制 Nios II 嵌入式处理器。
- 采用 EPCS 系列串行配置器件的低成本配置解决方案。

4. Cyclone III 系列 FPGA

在上节已经提到 Cyclone III 系列 FPGA 的内部结构。该系列具有最多 200K 逻辑单元、8MB 存储器，而静态功耗不到 1/4W。采用台积电(TSMC)的低功耗(LP)工艺技术进行制造，可以应用于通信设备、消费类、汽车、显示、工业、视频和图像处理、软件、无线电设备等领域。

Cyclone III 的子系列 Cyclone III LS 系列利用低功耗、高性能 FPGA 平台，在硬件、软件和知识产权(IP)层面上实现了一系列安全特性。可以保护设计者的 IP 不被篡改、异向剖析和克隆。而且，这些器件还能通过设计分离特性，在一个芯片中实现多种功能，从而减小了实际应用的体积、重量和功耗。另外，Cyclone III 也支持 Nios II 嵌入式处理器软核，可以实现复杂的多 CPU 嵌入式解决方案。

Cyclone III 系列 FPGA 器件的其他特性与 Cyclone II 很接近，这里不做赘述。

5. MAX 系列 CPLD

MAX 系列包括 MAX7000AE、MAX7000S、MAX3000A 等器件。这些器件的基本结构单元是乘积项，在工艺上采用 EEPROM 和 EPROM。器件的编程数据可永久保存，可加密。MAX 系列的集成度在数百门到 2 万门之间。所有 MAX 系列的器件都具有 ISP 在系统编程的功能，支持 JTAG 边界扫描测试。

6. MAX II 系列器件

这是一款上电即用、非易失性的 PLD 器件系列，用于通用的低密度逻辑应用环境。除了给予传统 CPLD 设计最低的成本，MAX II 器件还将成本和功耗优势引入了高密度领域。其特点是使用 LUT 结构，内含 Flash，可以实现自动配置；和 3.3V MAX 器件相比，MAX Ⅱ只有十分之一的功耗，1.8 V 内核电压以减小功耗，可靠性高；支持内部时钟频率达 300 MHz，内置用户非易失性 Flash 存储器块；通过取代分立式非易失性存储器件以减少芯片数量；MAXⅡ器件在工作状态时能够下载第一个第二个设计；可降低远程现场升级的成本；有灵活的多电压(MultiVolt)内核；片内由电压调整器支持 3.3V、2.5V 或 1.8V 电源输入；可减少电源电压种类，简化单板设计；可以访问 JTAG 状态机，在逻辑中例化用户功能；可提高单板上不兼容 JTAG 协议的 Flash 器件的配置效率。

1.11　编程与配置

在大规模可编程逻辑器件出现以前，人们在设计数字系统时，把器件焊接在电路板上是设计的最后一个步骤。当设计存在问题并得到解决后，设计者往往不得不重新设计印制电路板。设计周期被无谓地延长了，设计效率也很低。CPLD、FPGA 的出现改变了这一切。现在，人们在未设计具体电路时，就把 CPLD、FPGA 焊接在印制电路板上，然后在设计调试时可以一次又一次随心所欲地改变整个电路的硬件逻辑关系，而不必改变电路板结构。

这一切都有赖于 CPLD、FPGA 的在系统下载或重新配置功能。

目前常见的大规模可编程逻辑器件的编程工艺有以下 3 种：

(1) 基于电可擦除存储单元的 EEPROM 或 Flash 技术。CPLD 一般使用此技术进行编程。CPLD 被编程后改变了电可擦除存储单元中的信息，掉电后可保持。某些 FPGA 也采用 Flash 工艺，比如 Actel 的 ProASIC plus 系列 FPGA、Lattice 的 Lattice XP 系列 FPGA。

(2) 基于 SRAM 查找表的编程单元。对该类器件，编程信息是保持在 SRAM 中的，SRAM 在掉电后编程信息立即丢失，在下次上电后，还需要重新载入编程信息。大部分 FPGA 采用这种编程工艺。所以对于 SRAM 型 FPGA，在使用中必须利用专用配置器件来存储编程信息，以便在上电后，该器件能对 FPGA 自动编程配置。

(3) 基于反熔丝的编程单元。Actel 的 FPGA、Xilinx 部分早期的 FPGA 采用此种结构。相比之下，电可擦除编程工艺的优点是编程后信息不会因掉电而丢失，但编程次数有限，编程的速度不快。对于 SRAM 型 FPGA 来说，配置次数为无限，在加电时可随时更改逻辑，但掉电后芯片中的信息即丢失，每次上电时必须重新载入信息。

Altera 的 FPGA 器件有两类配置下载方式：主动配置方式和被动配置方式。主动配置方式由 FPGA 器件引导配置操作过程，它控制着外部存储器和初始化过程，而被动配置方式则由外部计算机或控制器控制配置过程。FPGA 在正常工作时，它的配置数据(下载进去的逻辑信息)存储在 SRAM 中。由于 SRAM 的易失性，每次加电时，配置数据都必须重新下载。在实验系统中，通常用计算机或控制器进行调试，因此可以使用被动配置方式。而使用系统中，多数情况下必须由 FPGA 主动引导配置操作过程，这时 FPGA 将主动从外围专用存储芯片中获得配置数据，而此芯片中的 FPGA 配置信息是用普通编程器将设计所得的 POC 格式的文件烧录进去的。

EPC 器件中的 EPC2 型号的器件是采用 Flash 存储工艺制作的具有可多次编程特性的配置器件。EPC2 器件通过符合 IEEE 标准的 JTAG 接口可以提供 3.3/5V 的在系统编程能力；具有内置的 JTAG 边界扫描测试(BST)电路，可通过 ByteBlasterMV 下载电缆，使用串行矢量格式文件 POF 或 Jam Byte-Code(.jbc)等文件格式对其进行编程。EPC1/1441 等器件属 OTP 器件。对于 Cyclone、Cyclone II/III 等系列器件，Altera 还提供 AS 方式的配置器件和 EPCS 系列专用配置器件。EPCS 系列(如 EPCS1/4/16 等)配置器件也是串行配置的。

1.12　数字系统的设计方法简介

数字系统设计有多种方法，如模块设计法、自顶向下设计法和自底向上设计法等。

数字系统的设计一般采用自顶向下、由粗到细、逐步求精的方法。自顶向下是指将数字系统的整体逐步分解为各个子系统和模块，若子系统规模较大，则还需将子系统进一步分解为更小的子系统和模块，层层分解，直至整个系统中各子系统关系合理，并便于逻辑电路级的设计和实现为止。采用该方法设计时，高层设计进行功能和接口描述，说明模块的功能和接口，模块功能的更详细的描述在下一设计层次说明，最底层的设计才涉及具体

的寄存器和逻辑门电路等实现方式的描述。采用自顶向下的设计方法有如下优点：

(1) 自顶向下设计方法是一种模块化设计方法。对设计的描述从上到下逐步由粗略到详细，符合常规的逻辑思维习惯。由于高层设计同器件无关，设计易于在各种集成电路工艺或可编程器件之间移植。

(2) 适合多个设计者同时进行设计。随着技术的不断进步，许多设计由一个设计者设计已无法完成，必须经过多个设计者分工协作完成一项设计的情况越来越多。在这种情况下，应用自顶向下的设计方法便于由多个设计者同时进行设计，对设计任务进行合理分配，用系统工程的方法对设计进行管理。

针对具体的设计，实施自顶向下的设计方法的形式会有所不同，但均需遵循以下两条原则：逐层分解功能，分层次进行设计。同时，应在各个设计层次上，考虑相应的仿真验证问题。

1.12.1　数字系统的设计准则

进行数字系统设计时，通常需要考虑多方面的条件和要求，如设计的功能和性能要求、元器件的资源分配和设计工具的可实现性、系统的开发费用和成本等。虽然具体设计的条件和要求千差万别，实现方法也各不相同，但数字系统设计还是具备一些共同的方法和准则的。

1. 分割准则

自顶向下的设计方法或其他层次化的设计方法，需要对系统功能进行分割，然后用逻辑语言进行描述。分割过程中，若分割过粗，则不易用逻辑语言表达；分割过细，则带来不必要的重复和繁琐。因此，分割的粗细需要根据具体的设计和设计工具情况而定。掌握分割程度，可以遵循以下的原则：分割后最底层的逻辑块应适合用逻辑语言进行表达；相似的功能应该设计成共享的基本模块；接口信号尽可能少；同层次的模块之间，在资源和I/O分配上，尽可能平衡，以使结构匀称；模块的划分和设计，尽可能做到通用性好，易于移植。

2. 系统的可观测性

在系统设计中，应该同时考虑功能检查和性能的测试，即系统观测性的问题。一些有经验的设计者会自觉地在设计系统的同时设计观测电路，即观测器，指示系统内部的工作状态。

建立观测器，应遵循以下原则：具有系统的关键点信号，如时钟、同步信号和状态等信号；具有代表性的节点和线路上的信号；具备简单的“系统工作是否正常”的判断能力。

3. 同步和异步电路

异步电路会造成较大延时和逻辑竞争，容易引起系统的不稳定，而同步电路则是按照统一的时钟工作，稳定性好。因此在设计时尽可能采用同步电路进行设计，避免使用异步

电路。在必须使用异步电路时，应采取措施来避免竞争和增加稳定性。

4. 最优化设计

由于可编程器件的逻辑资源、连接资源和 I/O 资源有限，器件的速度和性能也是有限的，用器件设计系统的过程相当于求最优解的过程。因此，需要给定两个约束条件：边界条件和最优化目标。

所谓边界条件，是指器件的资源及性能限制。最优化目标有多种，设计中常见的最优化目标有：器件资源利用率最高；系统工作速度最快，即延时最小；布线最容易，即可实现性最好。具体设计中，各个最优化目标间可能会产生冲突，这时应满足设计的主要要求。

1.12.2　数字系统设计的艺术

一个系统的设计，通常需要经过反复的修改、优化才能达到设计的要求。一个好的设计，应该满足“和谐”的基本特征，对数字系统可以根据以下几点作出判断：设计是否总体上流畅，无拖泥带水的感觉；资源分配、I/O 分配是否合理，是否有设计上和性能上的瓶颈，系统结构是否协调；是否具有良好的可观测性；是否易于修改和移植；器件的特点是否能得到充分的发挥。

1.13　Quartus II

本书给出的实验和设计多是基于 Quartus II 的，其应用方法和设计流程对于其他流行的 EDA 工具而言具有一定的典型性和一般性，所以在此对它做一些介绍。

Quartus II 是 Altera 提供的 FPGA/CPLD 开发集成环境。Quartus II 在 21 世纪初推出，是 Altera 前一代 FPGA/CPLD 集成开发环境 MAX+plus II 的更新换代产品，其界面友好，使用便捷。在 Quartus II 上可以完成 1.5 节所述的整个流程，它提供了一种与结构无关的设计环境，使设计者能方便地进行设计输入、快速处理和器件编程。

Altera 的 Quartus II 提供了完整的多平台设计环境，能满足各种特定设计的需要，是单芯片可编程系统(SOPC)设计的综合性环境和 SOPC 开发的基本设计工具，并为 Altera DSP 开发包进行系统模型设计提供了集成综合环境。

Quartus II 设计工具完全支持 Verilog、VHDL 的设计流程，其内部嵌有 Verilog、VHDL 和 SystemVerilog 逻辑综合器。Quartus II 也可以利用第二方的综合工具，例如 Leonardo Spectrum、SynPlify Pro、DC-FPGA，并能直接调用这些工具。同样，Quartus II 具备仿真功能，同时也支持第三方仿真工具，如 ModelSim。此外 Quartus II 与 MATLAB 和 DSP Builder 结合，可以进行基于 FPGA 的 DSP 系统开发，是 DSP 硬件系统实现的关键 EDA 工具。

Quartus II 包括模块化的编译器。编译器包括的功能模块有分析/综合器(Analysis& Synthesis)、适配器(Fi1ter)、装配器(Assembler)、时序分析器(Timing Analyzer)、设计辅助模块(Design Assistant)、EDA 网表文件生成器(EDA Netlist Writer)、编辑数据接口(ComPiler Database Interface)等。可以通过选择 Start ComPilation 来运行所有的编译器模块，也可以通过选择 Start 单独运行各个模块。还可以通过选择 ComPiler Tool(Tool 菜单)，在 ComPiler Tool 窗口中运行相应的功能模块。在 ComPiler Tool 窗口中，可以打开相应的功能模块所包含的设置文件或报告文件，或打开其他相关窗口。

此外，Quartus II 还包含许多十分有用的 LPM(Lbrary of Parameterized Modules)模块，它们是复杂或高级系统构建的重要组成部分，也可在 Quartus II 中与普通设计文件一起使用(后续章节中将详细介绍这部分内容)。Altera 提供的 LPM 函数均基于 Altera 器件的结构做了优化设计。

如图 1-24 所示的上排是 Quartus II 编译设计主控界面，它显示了 Quartus II 自动设计的各主要处理环节和设计流程，包括设计输入编辑、设计分析与综合、适配、编程文件汇编(装配)、时序参数提取以及编程下载几个步骤。图 1-24 下排的流程框图是与上面的 Quartus II 设计流程相对应的标准的 EDA 开发流程。

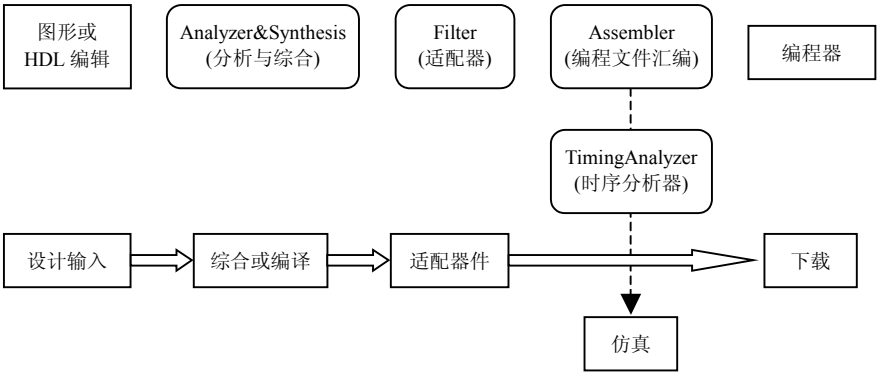


图 l-24　Quartus II 设计流程

Quartus II 编译器支持的硬件描述语言有 VHDL Verilog、System Verilog 及 AHDL。AHDL 是 Altera 公司自己设计、制定的硬件描述语言，是一种以结构描述方式为主的硬件描述语言，只有企业标准。

Quartus II 允许来自第三方的 EDIF、VQM 文件输入，并提供了很多 EDA 软件的接口。Quartus II 支持层次化设计，可以在一个新的编辑输入环境中对使用不同输入设计方式完成的模块(元件)进行调用，从而解决了原理图与 HDL 混合输入设计的问题。在设计输入之后，Quartus II 的编译器将给出设计输入的错误报告。Quartus II 拥有性能良好的设计错误定位器，用于确定文本或图形设计中的错误。对于使用 HDL 的设计，可以使用 Quartus II 带有的 RTL Viewer 观察综合后的 RTL 图。在进行编译后，可对设计进行时序仿真。在仿真前，需要利用波形编辑器编辑一个波形激励文件。编译和仿真经检测无误后，便可以将下载信息通过 Quartus II 提供的编程器下载至目标器件中。

1.14　IP 核

IP 就是知识产权校或知识产权模块的意思，在 EDA 技术开发中具有十分重要的地位。著名的美国 Dataquest 咨询公司将半导体产业的 IP 定义为用于 ASIC 或 FPGA 中的预先设计好的电路功能模块。IP 分软 IP、固 IP 和硬 IP。

软 IP 是用 HDL 等硬件描述语言描述的功能块，但是并不涉及用什么具体电路元件实现这些功能。软 IP 通常是以 HDL 源文件的形式出现，应用开发过程与普通的 HDL 设计也十分相似，只是所需的开发软硬件环境比较昂贵。软 IP 的设计周期短，设计投入少。由于不涉及物理实现，为后续设计留有很大的发挥空间，增大了 IP 的灵活性和适应性。软 IP 的弱点是在一定程度上使后续工序无法适应整体设计，从而需要一定程度的软 IP 修正，在性能上也不可能获得全面的优化。

固 IP 是完成了综合的功能块。它有较大的设计深度，以网表文件的形式提交客户使用。如果客户与固 IP 使用同一个 IP 生产线的单元库，IP 应用的成功率会高得多。

硬 IP 提供设计的最终阶段产品：掩膜。随着设计深度的提高，后续工序所需要做的事情就越少，当然，灵活性也就越小。不同的客户可以根据自己的需要订购不同的 IP 产品。由于通信系统越来越复杂，PLD 的设计也更加庞大，这增加了市场对 IP 核的需求。各大 FPGA 厂家继续开发新的商品 IP，并且开始提供“硬件”IP，即将一些功能在出厂时就固化在芯片中。

1.15　EDA 的发展趋势

随着市场需求的增长，集成工艺水平及计算机自动设计技术的不断提高，促使单片系统，或称系统集成芯片成为 IC 设计的发展方向，这一发展趋势表现在如下几个方面：

- 超大规模集成电路的集成度和工艺水平不断提高，如 40 纳米工艺已经走向成熟，在一个芯片上完成系统级的集成已成为可能。
- 由于工艺线宽的不断减小，在半导体材料上的许多寄生效应已经不能简单地被忽略。这就对 EDA 工具提出了更高的要求，同时也使得 IC 生产线的投资更为巨大。可编程逻辑器件开始进入传统的 ASIC 市场。
- 市场对电子产品提出了更高的要求，如必须降低电子系统的成本，减小系统的体积等，从而对系统的集成度不断提出更高的要求。同时，设计的效率也成了一个产品能否成功的关键因素，促使 EDA 工具和 IP 核应用更为广泛。
- 高性能的 EDA 工具得到长足的发展，其自动化和智能化程度不断提高，为嵌入大系统设计提供了功能强大的开发环境。
- 计算机硬件平台性能大幅度提高，为复杂的 SOC 设计提供了物理基础。

但现有的HDL 只是提供行为级或功能级的描述，尚无法完成对复杂的系统级的抽象

描述。人们正尝试开发一种新的系统级设计语言来完成这一工作，现在已开发出更趋于电路行为级的硬件描述语言，如 SystemC、SystemVerilog 及系统级混合仿真工具，可以在同一个开发平台上完成高级语言(如 C/C++等)与标准 HDL 语言(VerilogHDL VHDL)或其他更低层次描述模块的混合仿真。虽然用户用高级语言编写的模块尚不能自动转化成 HDL 描述，但作为一种针对特定应用领域的开发工具，软件供应商已经为常用的功能模块提供了丰富的宏单元库支持，可以方便地构建应用系统，并通过仿真加以优化，最后自动产生 HDL 代码，进入下一阶段的 ASIC 实现。

此外，随着系统开发对 EDA 技术的目标器件各种性能要求的提高，ASIC 和 FPGA 将更大程度的相互融合。这是因为虽然标准逻辑 ASIC 芯片尺寸小、功能强大、耗电低，但设计复杂，并且有批量生产要求；可编程逻辑器件开发费用低廉，能在现场进行编程，但却体积大、功能有限，而且功耗较大。因此，FPGA 和 ASIC 正在走到一起，互相融合，取长补短。由于一些 ASIC 制造商提供具有可编程逻辑的标准单元，可编程器件制造商重新对标准逻辑单元产生兴趣，而有些公司采取两头并进的方法，从而使市场开始发生变化，在 FPGA 和 ASIC 之间正在诞生一种“杂交”产品，以满足成本和上市速度的要求，例如将可编程逻辑器件嵌入标准单元。

尽管将标准单元核与可编程器件集成在一起并不意味着使 ASIC 更加便宜，或者使 FPGA 更加省电。但是，可使设计人员将两者的优点结合在一起，通过去掉 FPGA 的一些功能，可减少成本和开发时间并增加灵活性。当然现在也在进行将 ASIC 嵌入可编程逻辑单元的工作。目前，许多 PLD 公司开始为 ASIC 提供 FPGA 内核。PLD 厂商与 ASIC 制造商结盟，为 SOC 设计提供嵌入式 FPGA 模块，使未来的 ASIC 供应商有机会更快地进入市场，利用嵌入式内核获得更长的市场生命期。

例如在实际应用中使用所谓可编程系统级集成电路(FPSLIC)，即将嵌入式 FPGA 内核与 RISC 微控制器组合在一起形成新的 IP，广泛用于电信、网络、仪器仪表和汽车中的低功耗应用系统中。当然，也有 PLD 厂商不把 CPU 的硬核直接嵌入在 FPGA 中，而使用了软 IP，并称之为 SOPC(可编程片上系统)，也可以完成复杂电子系统的设计，只是代价将相应提高。

现在 FPGA 和 ASIC 之间的界限正变得模糊。系统级芯片不仅集成 RAM 和微处理器，也集成 FPGA。整个 EDA 和 IC 设计工业都朝这个方向发展，这并非是 FPGA 与 ASIC 制造商竞争的产物，而对于用户来说，意味着有了更多的选择。

1.16　本章小结

本章首先阐述了EDA 技术的含义、发展历程、主要内容；然后详细介绍了EDA 技术的工程设计流程和EDA 软件系统的构成，讨论了数字系统的设计方法；阐述了可编程逻辑器件发展进程、种类及分类方法；然后研究了复杂可编程逻辑器件(CPLD)的基本结构，主要介绍了Altera 公司的Cyclone 系列 FPGA 器件；接着探讨了现场可编程门阵列编程与配置；

最后介绍了数字系统的设计方法。

通过对本章的学习，应该掌握 EDA 技术的主要内容、工程设计流程和 EDA 软件系统的构成。

1.17　习　　题

1-1　EDA 技术与 ASIC 设计和 FPGA 开发有什么关系？FPGA 在 ASIC 设计中有什么用途？

1-2　利用 EDA 技术进行电子系统设计有什么优点？

1-3　什么是综合？有哪些类型？综合在电子设计自动化中的地位是什么？

1-4　IP 在 EDA 技术的应用和发展中的意义是什么？

1-5　叙述 EDA 的 FPGA/CPLD 设计流程，以及涉及的 EDA 工具及其在整个流程中的作用。

1-6　OLMC 有何功能？说明 GAL 是怎样实现可编程组合电路与时序电路的。

1-7　什么是基于乘积项的可编程逻辑结构？什么是基于查找表的可编程逻辑结构？

1-8　就逻辑宏单元而言，GAL 中的 OLMC、CPLD 中的 LC、FPGA 中的 LUT 和 LE 的含义和结构特点是什么？它们都有何异同点？

1-9　为什么说用逻辑门作为衡量逻辑资源大小的最小单元不准确。

l-10　标志 FPGA/CPLD 逻辑资源的逻辑宏单元包含哪些结构？

l-11　解释编程与配置这两个概念。

1-12　请参阅相关资料，并回答问题：按本章给出的归类方式，将基于乘积项的可编程逻辑结构的 PLD 器件归类为 CPLD；将基于查找表的可编程逻辑结构的 PLD 器件归类为 FPGA，那么 MAXⅡ系列应该属于什么类型的 PLD 器件？为什么？

第2章　原理图输入法逻辑电路设计流程

本章拟使用纯原理图输入的设计方式，首先通过一个简单电路模块的功能实现和功能测试，详细介绍 Quartus II 的完整设计流程，使学习者初步掌握利用 Quartus II 完成数字系统设计的基本方法；然后在此基础上通过一个数字频率计的设计，进一步介绍较复杂数字系统的 EDA 技术。这样安排的目的是：无须经历漫长的理论学习阶段，读者一开始就可以仅需借助已有的知识，近距离地接触和体验 EDA 技术中最有特色、最直观的实践内容。这不仅为后续的理论学习提供了重要的感性认识，也有助于在此后的学习过程中，将所学的基础理论更好更快捷地融入工程实践中。

2.1　原理图输入设计方法的特点

利用 EDA 工具进行原理图输入设计的优点是，设计者不必具备许多诸如编程技术、硬件语言等新知识就能迅速入门，完成基于 EDA 的较大规模的电路系统设计。

与 MAX+plus II 相比，Quartus II 提供了更强大、更直观便捷和操作灵活的原理图输入设计功能，同时还配备了更丰富的适用于各种需要的元件库，其中包含基本逻辑元件库(如与非门、反向器、D 触发器等)、宏功能元件库(包含了几乎所有 74 系列的器件功能块)，以及类似于 IP 核的参数可设置的宏功能块 LPM 库、锁相环、嵌入式逻辑分析仪等。Quartus II 同样提供了原理图输入多层次设计功能，使用户能方便地设计更大规模的电路系统以及使用方便、精度良好的时序仿真器。与传统的数字电路设计方法和设计流程相比，Quartus II 提供的原理图输入设计功能具有不可比拟的优势和先进性：

- 能进行几乎任意层次的数字系统设计。传统的数字电路设计只能完成单一层次的设计，使得设计实践总是停留在十分简单且很少能与实际工程设计方法结合的电路设计上。这使设计者无法了解和实现多层次的大系统功能和规范的设计方法。
- 能对系统中的任一层次或任一元件的功能进行精确的时序仿真，精度达 0.1ns。因此能发现对系统可能产生不良影响的竞争冒险现象。
- 通过时序仿真，能迅速定位电路系统的错误所在，并随时纠正。
- 能对设计方案进行随时更改，并储存设计过程中所有的电路和测试文件入档。
- 通过编译和下载，能在 FPGA 上对设计项目随时进行硬件测试验证。

- 如果使用 FPGA 和配置编程方式，通常将不会有器件损坏和损耗的问题。
- 符合现代电子设计技术规范。

传统的数字电路设计和实验利用手工连线的方法完成元件连接，容易对学习者产生误导，以为只要将元件间的引脚用引线按电路图连上即可，而不必顾及引线的长短、粗细、弯曲方式可能产生的分布电感和电容效应，以及电磁兼容性等十分重要的问题。

2.2　数字频率计设计任务导入

如图 2-1 所示的电路是一个数字频率计，是用 EDA 设计软件 Quartus II 的原理图编辑器表达的设计电路，利用 Quartus II 将此电路实现为实际的频率计就是本章的最终任务。

Quartus II 即可将此电路编译成为 FPGA 中的硬件逻辑电路实现文件，而将此文件下载于 FPGA 中后即可完成频率计的设计。图 2-1 中的 TF-CTRL 是此频率计的测频时序控制模块，4 个 CNT10 模块分别是双十进制计数器模块，其内部电路结构如图 2-2 所示；而 4 个 74374 被用作频率计输出数据的锁存器。这些模块的内部结构及其更深层次的结构也是用原理图表述的，它们必须由设计者利用 Quartus II 分别设计好。

以下将详细介绍与此项设计相关的设计技术、测试技术和实现技术。需要注意的是，以下介绍的内容具有一般性，在此后的许多实训设计任务中将反复用到，它也同样适用于其他输入方法的设计，如基于 HDL 的硬件描述语言的输入设计方法或混合输入设计方法等。

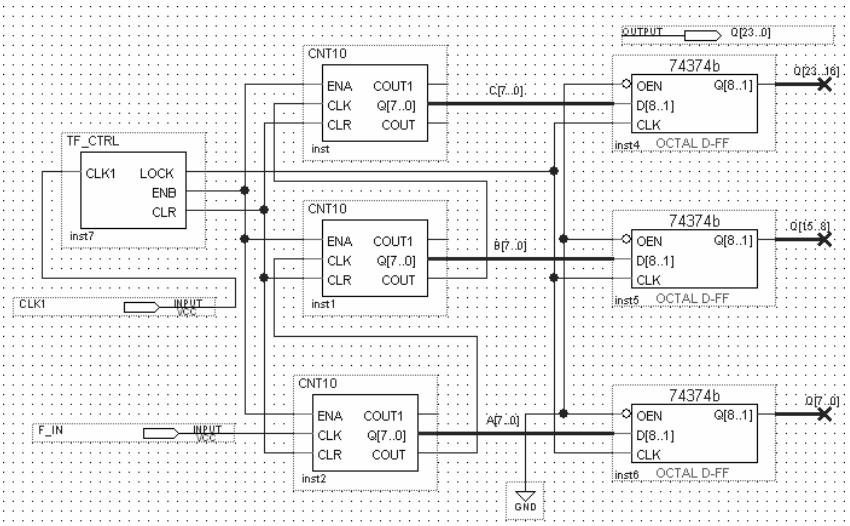


图 2-1　6 位数字频率计顶层电路原理图

2.3　原理图输入方式基本设计流程

本节拟通过一个如图 2-2 所示的 2 位十进制时钟可控计数器介绍基于 Quartus II 的原理

图输入的基本设计流程和方法。下面首先介绍利用 74390 和其他一些电路单元设计可控型的十进制计数单元，然后利用层次化设计方法，完成一个 6 位十进制计数器的设计。

本节介绍的设计流程具有一般性。除了最初的输入方法稍有不同外，与应用 VHDL 的文本输入设计方法的流程是相同的。所以在此后的 VHDL 文本输入设计介绍中，不再重复本章已介绍的相关内容或流程。

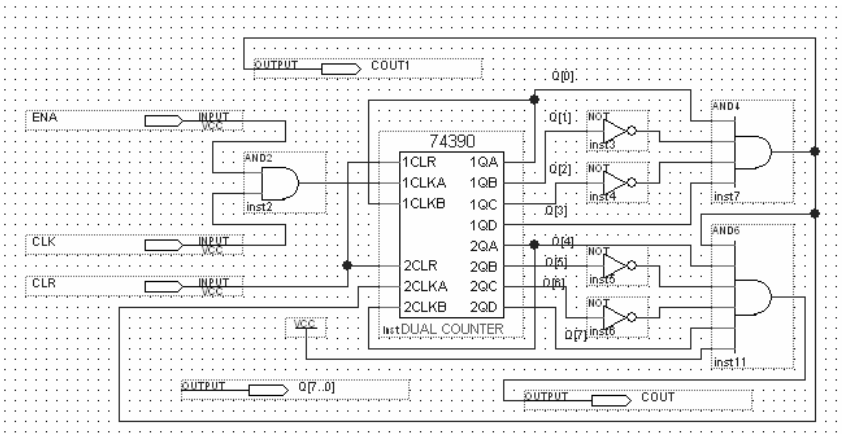


图 2-2　模块 CNT10 的内部结构(2 位十进制计数器电路图)

2.3.1　建立工作库文件夹和存盘原理图空文件

初学者可以按照以下步骤学习和掌握基于原理图输入方式的设计流程。

首先建立工作库目录，以便存储工程项目设计文件。任何一项设计都是一项工程(Project)，都必须先为此工程建立一个放置与此工程相关的所有设计文件的文件夹。此文件夹将被 EDA 软件默认为工作库(Work Library)。一般地，不同的设计项目最好放入不同的文件夹中，而同一工程的所有文件都必须放在同一文件夹中。注意，不要将文件夹设在计算机已有的安装目录中，更不要将工程文件直接放在 Quartus II 的安装目录中。在建立了文件夹后就可以将设计文件通过 Quartus II 的原理图编辑器编辑并存盘，主要步骤如下：

(1) 新建一个文件夹。可以利用 Windows 资源管理器新建一个文件夹。这里假设本项设计的文件夹取名为 MY_PROJECT，在 D 盘中，路径为 D:\MY_PROJECT。注意：文件夹名不能用中文，也最好不要用数字。

(2) 建立原理图源文件编辑窗。打开 Quartus II，选择菜单 File→New。在 New 窗口中的 Design Files 条目中选择原理图文件类型，这里选择 Block Diagram/Schematic File，如图 2-3 所示，以后即可在如图 2-4 所示的原理图编辑窗中加入所需的电路元件。

(3) 空文件存盘。选择 File→Save As 命令，找到已设立的文件夹路径为 D:\MY_PROJECT，存盘文件名可取为 CNT10.bdf。这是一个还没有加入任何电路元件的空原理图文件。加元件和编辑电路的工作要待创建工程后再进行。

存盘时，当出现问句“Do you want to create…”时，单击“是”按钮，则直接进入创建工程流程；单击“否”按钮，可按以下方法进入创建工程流程。这里单击“否”按钮，

利用另一方式进入创建工程流程。

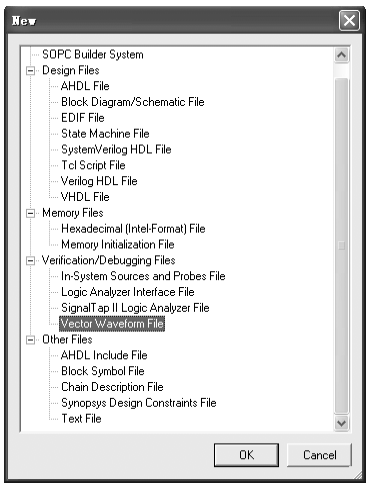


图 2-3　选择原理图编辑文件类型

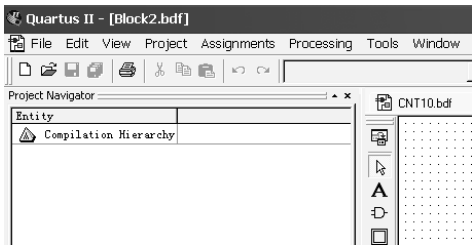


图 2-4　打开原理图编辑窗

2.3.2　创建工程

使用 New Project Wizard 可以为工程指定工作目录、分配工程名称以及指定最高层设计实体的名称；还可以指定要在工程中使用的设计文件、其他源文件、用户库和 EDA 工具，以及目标器件系列和具体器件等。在此要利用 New Project Wizard 创建此设计工程，即令顶层设计文件 CNT10.bdf 为工程文件，并设定此工程的一些相关信息，如工程名、目标器件、综合器、仿真器等。以下设计流程具有一般性。

(1) 打开并建立新工程管理窗。选择 File→New Project Wizard 命令，弹出工程设置对话框，如图 2-5 所示。

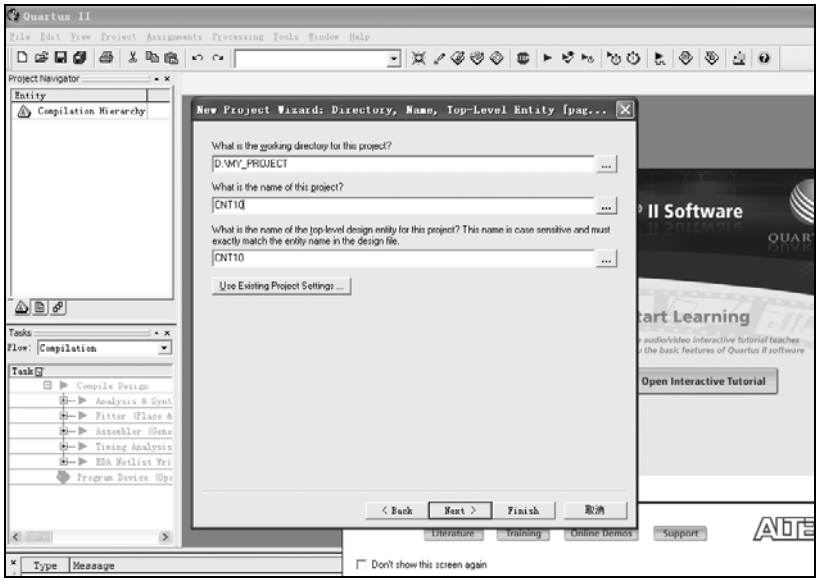


图 2-5　利用 New Project Wizard 创建工程 CNT10

单击此对话框第二栏右侧的“…”按钮，找到文件夹 D:\MY_PROJECT，选中已存盘的文件 CNT10.bdf(此时还是一个空原理图)。再单击“打开”按钮，即出现如图 2-5 所示的设置情况。其中第一栏的 D:\MY_PROJECT 表示工程所在的工作库文件夹；第二栏的 CNT10 表示此项工程的工程名，工程名可以修改，也可直接用顶层文件的实体名作为工程名(如果是 VHDL 文本文件的话)，在此就是按这种方式命名的；第三栏是当前工程顶层文件的实体名，这里为 CNT10。

(2) 将设计文件加入工程中。单击下方的 Next 按钮，在弹出的对话框中单击 File name 栏右侧的按钮“…”，将与工程相关的文件(如CNT10.bdf)加入工程；再单击右侧的Add 按钮，即得到如图 2-6 所示的情况。

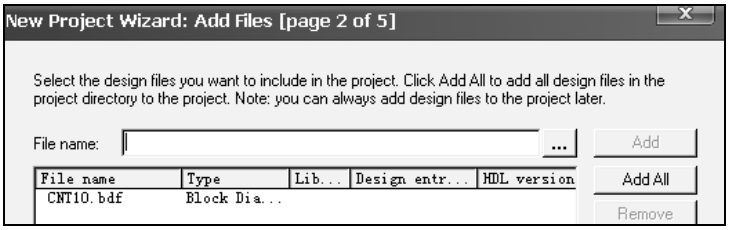


图 2-6　将所有相关的文件都加入此工程

(3) 选择目标芯片。单击 Next 按钮，选择目标芯片。首先在 Family 栏选择芯片系列，在此选择 Cyclone III 系列。这里准备选择的目标器件是 EP3C10E144C8。这里 EP3C10 表示 Cyclone III 系列及此器件的逻辑规模；E 表示带有金属地线底板的 TQFP 封装；C8 表示速度级别。便捷的选择方法是通过如图 2-7 所示的窗口右边的 3 个选项过滤选择，分别选择 Package 为 TQFP、Pin count 为 144、Speed grade 为 8。

(4) 工具设置。单击 Next 按钮后，弹出的下一个窗口是 EDA 工具设置窗。EDA Tool Settings。其中有 3 项选择：EDA design entry/synthesis tool，即选择输入的 HDL 类型和综合工具；EDA simulation tool 用于选择仿真工具；EDA timing amalysis tool 用于选择时序分析工具，这是除 Quartus II 自含的所有设计工具以外的工具。因此，如果都不作选择，即选择默认，表示仅选择 Quartus II 自含的所有设计工具。在此选择默认。

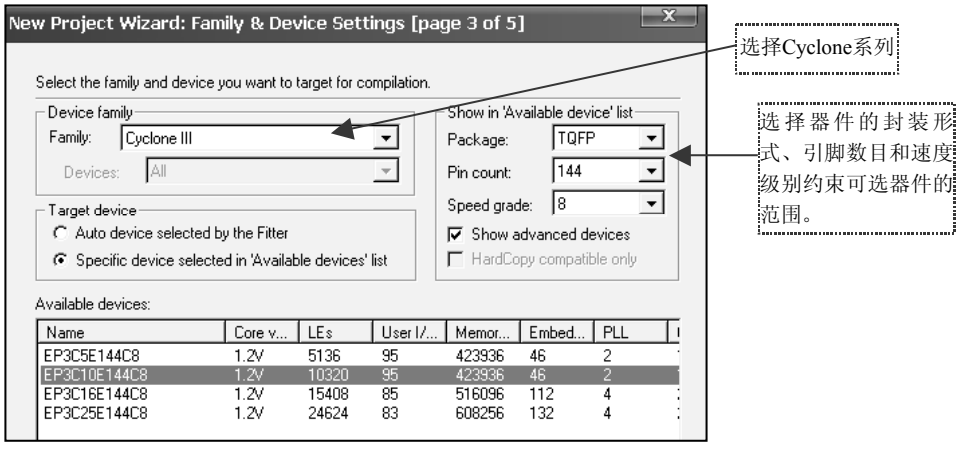


图 2-7　选择目标器件 EP3C10E144C8 型 FPGA

(5) 结束设置。单击 Next 按钮，弹出“工程设置统计”窗口，其中列出了此项工程的相关设置情况。单击 Finish 按钮，确定已设定好此工程，并出现 CNT10 的工程管理窗口，或称 Compilation Hierarchies 窗口，主要显示工程项目的层次结构，如图 2-8 所示。注意，此工程管理窗口左上角所示的是工程路径、工程名 CNT10 和当前已打开的文件名。

Quartus II 将工程信息存储在工程配置文件(Quartus)中。它包含有关 Quartus II 工程的所有信息，包括设计文件、波形文件、SignalTap II 文件、内存初始化文件等，以及为了满足当前工程相关技术指标要求的针对综合、适配和仿真的诸多设置文件。

(6) 编辑构建电路图。现在就可以为空的原理图文件添加电路了。双击图 2-8 左侧的工程名 CNT10，打开其原理图文件窗口，再双击右侧原理图编辑窗口内任意一点，即弹出一个逻辑电路器件输入对话框，如图 2-9 所示。在此对话框的左栏 Name 文本框内输入所需元件的名称，在此为 74390。由于仅考虑器件的逻辑功能，同类功能的器件，如 74ls 390、74HC390、74s390 等，一律命名为 74390。然后单击 OK 按钮，即可将此元件调入编辑窗口中。

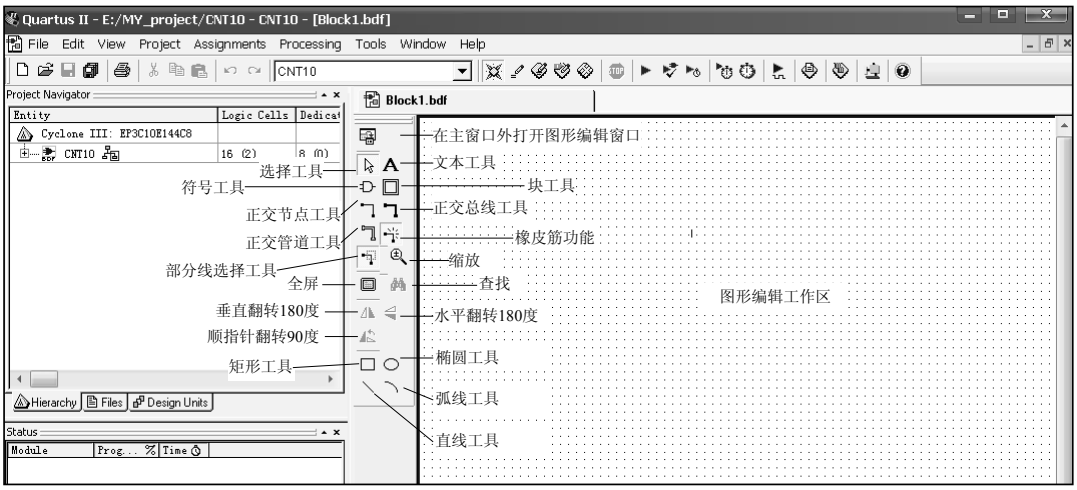


图 2-8　CNT10 工程管理窗口

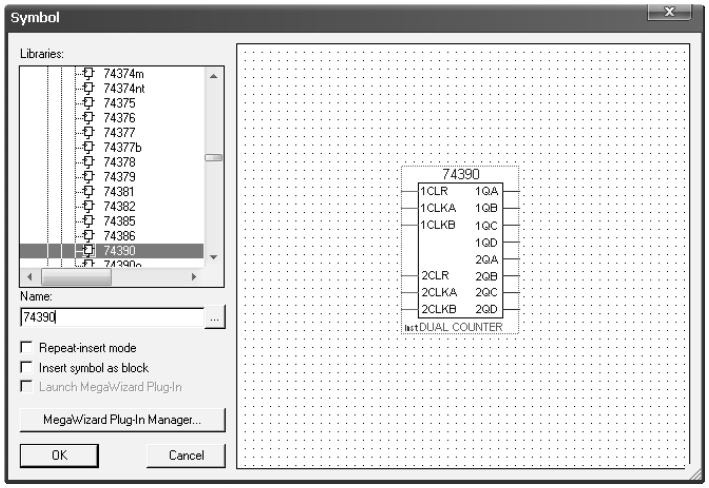


图 2-9　在元件调用对话框调出需要的宏功能元件 74390

再以同样的方法调入一个 2 输入与门，名称是 AND2；一个 4 输入与门，名称是 AND4；一个 6 输入与门，名称是 AND6；4 个反相器，名称是 NOT；以及数个输入输出端口，名称分别是 INPUT 和 OUTPUT。最后直接用鼠标拖出连线，将它们按照所需的逻辑功能连接起来，完成的电路如图 2-2 所示。

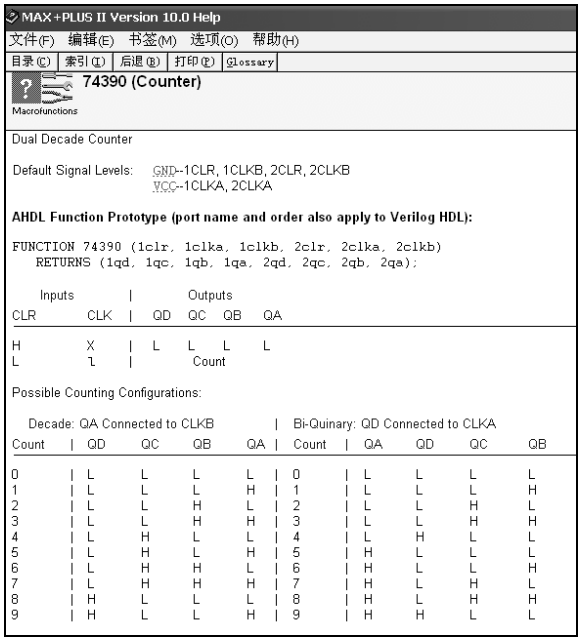
输入输出端口的名称可以通过双击相应端口元件，在弹出的对话框中输入，如 CLR。

在全程编译前，可使用 Settings 对话框(由 Assignments 进入)作一些必要的设置。

2.3.3　功能简要分析

下面初步分析图 2-2 所示电路的基本功能。首先了解元件 74390 的功能。为此可以了解其真值表。打开帮助文件 Macrofunctions(查 www.Kx-soc.com 可获此文件)，然后选择 Messages 选项，继而选择其中的 Macrofunction 选项和 Old_Style Macrofunctions 选项，最后选择 Counters 中的 74390，即可见其真值表，如图 2-10 所示。

74390 是双计数器(Dual Counter)。图 2-2 的电路构成了一个 2 位十进制计数器。输出信号 COUT 是它们的高位计数进位信号，而 COUT1 是低位计数进位信号；Q[7…0]是此计数器的输出总线。注意原理图的总线表达方式，Q[7…0]中 7 与 0 之间的点是两个，而非 3 个。Q[7…0]表示 8 根单线：Q[7]、Q[6]、…、Q[0]。用鼠标双击元件 74390，可以看到 74390 内部的结构。



MAX+PLUS II Version 10.0 Help

文件(F)　编辑(E)　书签(M)　选项(O)　帮助(H)

目录(C)　索引(I)　后退(B)　打印(P)　Glossary

74390 (Counter)

Macrofunctions

Dual Decade Counter

Default Signal Levels:　GND--1CLR, 1CLKB, 2CLR, 2CLKB
VCC--1CLKA, 2CLKA

AHDL Function Prototype (port name and order also apply to Verilog HDL):

```
FUNCTION 74390 (1clr, 1clka, 1clkb, 2clr, 2clka, 2clkb)
   RETURNS (1qd, 1qc, 1qb, 1qa, 2qd, 2qc, 2qb, 2qa);
```

Inputs		Outputs			
CLR	CLK	QD	QC	QB	QA
H	X	L	L	L	L
L	↑	Count			

Possible Counting Configurations:

Decade: QA Connected to CLKB					Bi-Quinary: QD Connected to CLKA				
Count	QD	QC	QB	QA	Count	QA	QD	QC	QB
0	L	L	L	L	0	L	L	L	L
1	L	L	L	H	1	L	L	L	H
2	L	L	H	L	2	L	L	H	L
3	L	L	H	H	3	L	L	H	H
4	L	H	L	L	4	L	H	L	L
5	L	H	L	H	5	H	L	L	L
6	L	H	H	L	6	H	L	L	H
7	L	H	H	H	7	H	L	H	L
8	H	L	L	L	8	H	L	H	H
9	H	L	L	H	9	H	H	L	L

图 2-10　74390 的真值表

图 2-2 中，74390 连接成两个独立的十进制计数器。CLK 通过一个与门进入 74390 的计数器“1”端的时钟输入端 1CLKA。与门的另一端由计数使能信号 EN8 控制：当 ENB=1 时允许计数；当 ENB=0 时禁止计数。计数器 1 的 4 位输出 Q[3]、Q[2]、Q[1]和 Q[0]并成

总线表达方式，即 Q[3…0]同时由一个 4 输入与门和两个反相器构成进位信号，即当计数到 9(1001)时，输出进位信号 COUT1。此进位信号进入第二个计数器的时钟输入端 2CLKA。第二个计数器的 4 位计数输出是 Q[7]、Q[6]、Q[5]和 Q[4]， 总线输出信号是 Q[7…4]。 右图的与门与反相器一同分别构成两个计数器的进位信号。这两个计数器的总的进位信号，可由一个 6 输入与门和两个反相器产生，由 COUT 输出；CLR 是计数器清零信号。

2.3.4　编译前设置

在对当前工程进行编译处理前，必须作好必要的设置，对编译加入一些约束，使编译结果更好地满足设计要求。具体步骤如下：

(1) 选择 FPGA 目标芯片。目标芯片的选择也可以这样来实现；选择 Assignments→Settings 命令，在弹出的如图 2-11 所示的对话框中选择 Category 选项下的 Device。选择需要的 FPGA 目标芯片，如 EP3C10EI144C8(此芯片已在建立工程时选定了)。

(2) 选择配置器件的工作方式。单击图 2-11 中的 Device and Pin Options 按钮，进入 Device and Pin Options 对话框，如图 2-12 所示。在此首先选择 General 选项卡。

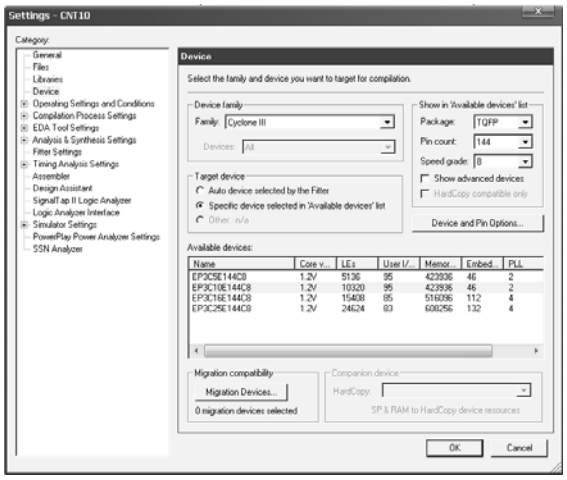


图 2-11　由 Settings 对话框选择目标器件 EP3C10E144C8

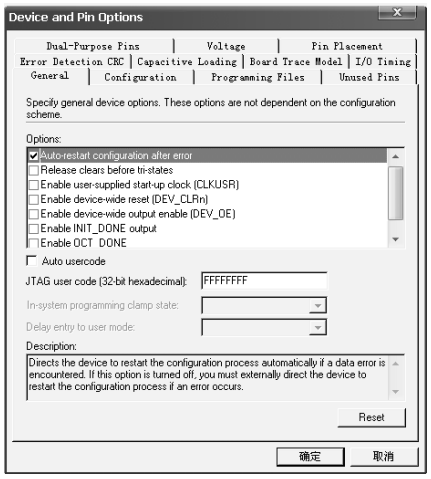


图 2-12　选择配置器件的工作方式

在 Options 列表框内选中 Auto-restart configuration after error，使 FPGA 配置失败后能自动重新配置，并加入JTAG 用户编码(可略)。注意，窗口下方将随项目名的改变而显示对应的帮助说明，用户可随时参考。

(3) 选择配置器件和编程方式。如果希望对编程配置文件能在压缩后通过 AS 模式下载进配置器件中，要在编译前做好设置。在此，选中图 2-12 的 Configuration 选项卡，即出现如图 2-13 所示的对话框。选中 Generate compressed bitstreams 复选框，就能产生用于 EPCS 的 POF 压缩编程配置文件。

在Configuration选项卡中，选择配置器件为 EPCS4，其配置模式可选择Active Serial(默认)。这种方式只对专用的 Flash 技术的配置器件进行编程(专用于 Cyclone/II/III 等系列 FPGA 的 EPCS4、EPCS16 等)。PC 机对 FPGA 的直接配置方式都是 JTAG 方式。

对 FPGA 进行所谓“掉电保护式”编程通常有 3 种模式：主动串行模式(AS Mode)、间接编程模式和被动串行模式(PS Mode)。对 EPCS4/EPCS16 的直接编程必须用主动串行模式，所以在选择了 AS(Active Serial)模式后，必须在 Use configuration device 项中选择配置器为 EPCS4 或 EPCS16。注意根据实验系统上目标器件配置的 EPCS 芯片型号决定。

(4) 双功能输入输出端口设置。选择图 2-13 中的双目标端口选项卡 Dual-Purpose Pins，界面如图 2-14 所示，将 nCE0 原来的 Use as programming pin 改为 Use as regular I/O(nCE0 端口作编程口时，可用于多 FPGA 芯片的配置)，这样可以将此端口也作普通 I/O 口来用。此项设置必须事先完成，否则一旦在编译时报错，软件给出的报告不会明示错误所在，初学者很难判断问题原因。对于已选的 EP3C10 器件，此双功能脚是 Pin 101，需关注此引脚的编译情况。

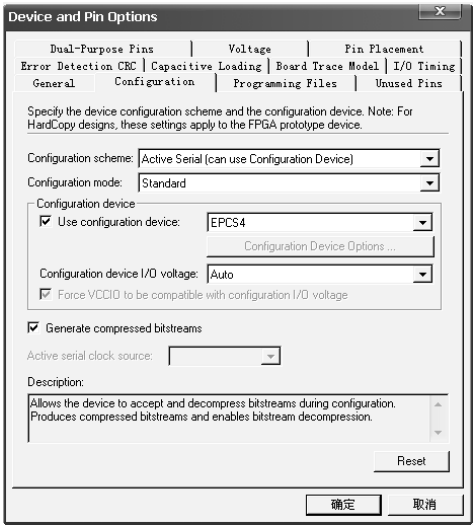


图 2-13　选择配置器件的型号和压缩方式

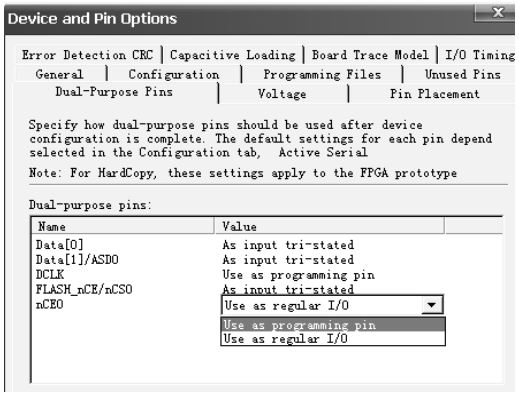


图 2-14　指定 nCE0 端口为普通的 I/O 端口

(5) 选择输出设置(此项操作可保持默认设置。选择 Programming Files 选项卡，选中 Hexadecimal(Intel-Format)Output File，即在生成常规下载文件的同时，产生二进制配置文件*.hexout，并设置起始地址为 0 的递增方式。此文件可用于单片机或 CPLD 与 EPROM 构成的 FFGA 配置电路系统。使用此格式文件的配置模式是被动串行模式。

(6) 选择目标器件闲置引脚的状态。此选项的设置在某些情况下十分重要。选择 Unused Pins 选项卡，可根据实际需要选择目标器件闲置引脚的状态。可选择为输入状态(呈高阻态，推荐此项选择)、输出状态(呈低电平)、输出不定状态，或不作任何选择。

在其他选项卡也可作一些设置，各设置选项的功能可参考对话框下方的说明(Description)。

2.3.5　全程编译

Quartus II 编译器是由一系列处理模块构成的，这些模块负责对设计项目的检错、逻辑综合、结构综合、输出结果的编辑配置，以及时序分析等。在这一过程中，为了可以把设

计项目适配到 FPGA 目标器中，将同时产生多种用途的输出文件，如功能和时序信息文件、器件编程的目标文件等。编译开始后，编译器首先检查出工程设计文件中可能存在的错误信息，供设计者排除，然后产生一个结构化的以网表文件表达的电路原理图文件。

在编译前，设计者可以通过各种不同的设置，指导编译器使用各种不同的综合和适配技术(如增量编译技术、时序驱动技术等)，以便提高设计项目的工作速度，优化器件的资源利用率。而且在编译过程中及编译完成后，可以从编译报告中获得所有相关的详细编译统计数据，以利于设计者及时调整设计方案。

编译前首先选择 Processing→Start Compilation 命令，启动全程编译(Full Compilation)。这里所谓的全程编译包括以上提到的 Quartus II 对设计输入的多项处理操作，其中包括排错、数据网表文件提取、逻辑综合、适配、装配文件生成(仿真文件与编程配置文件)，以及基于目标器件硬件性能的工程时序分析等。

编译过程中要注意工程管理窗口下方的 Processing 栏中的编译信息。如果启动编译后发现有错误，在下方的 Processing 处理栏中会以红色显示出错说明文字，并告知编译不成功，如图 2-15 所示。双击此栏的出错说明，即弹出对应的层次的文件，并用深色标记指出错误所在。改错后再次进行编译直至排除所有错误。

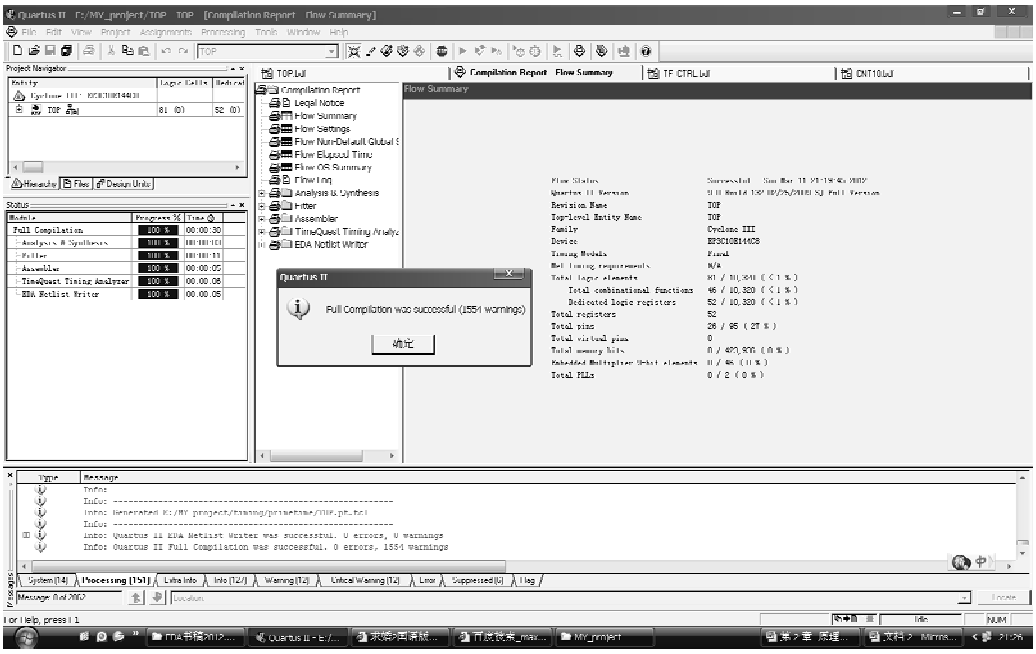


图 2-15　全程编译后出现的报错信息

另外也要注意 Processing 栏中出现的蓝色报警信息文字。有的 Warning 信息并不影响编译结果的正常功能，但有的则不然。如 LPM_ROM 的初始化数据文件未能成功调入等情况，编译器不会报错，只会报 Warning 信息，然而其硬件功能则大受影响。

如果编译成功，可以见到如图 2-15 所示的工程管理窗口左上角显示出工程 CNT10 的层次结构和其中结构模块耗用的逻辑宏单元数；在此栏下是编译处理流程，包括数据网表建立、逻辑综合(Synthesis)、适配(Fittering)、 配置文件装配(Assembling)和时序分析(Classic

Timing Analysizing)等；最下栏是编译处理信息。工程管理窗口的中栏(Compilatioon Report 栏)是编译报告项目选择菜单，单击其中各项可以详细了解编译与分析结果。

例如，单击 Flow Summary，将在右栏显示硬件耗用统计报告，其中报告了当前工程共耗用 16 个逻辑宏单元、8 个 D 触发器、0 个内部 RAM 位等。如果单击 Timing Analyzer 项的加号，则能通过单击列出的各项目，看到当前工程所有相关时序特性报告。如果单击 Fitter 项的加号，则能通过单击列出的各项目查看当前工程所有相关硬件特性适配报告，如其中的 Floorplan View，可观察此项工程在 FPGA 器件中逻辑单元的分布情况和使用情况。

为了更详细地了解相关情况，可以打开 FloorPlan 窗口，选择 View→Fu11 Screen 命令，打开全部界面，再单击此菜单的相关命令，如 Routing→Show Node Fan-In 等。

2.3.6　时序仿真测试电路功能

工程编译通过后，必须对其功能和时序性质进行测试，以了解设计结果是否满足要求。以上工程项目的仿真测试流程的详细步骤如下：

(1) 打开波形编辑器。选择 File→New 命令，在 New 窗口中选择 Verification 项下的 Vector Waveform File，如图 2-3 所示，单击 OK 按钮，即出现空白的仿真波形编辑器，注意将窗口扩大，以便观察。

(2) 设置仿真时间区域。对于时序仿真来说，将仿真时间轴设置在一个合理的时间区域十分重要。通常设置的时间范围在数十微秒间。

选择 Edit→End Time 命令，在弹出的对话框中的 Time 栏处输入 50，单位选 μs，整个仿真域的时间即设定为 50μs，如图 2-16 所示，单击 OK 按钮，结束设置。

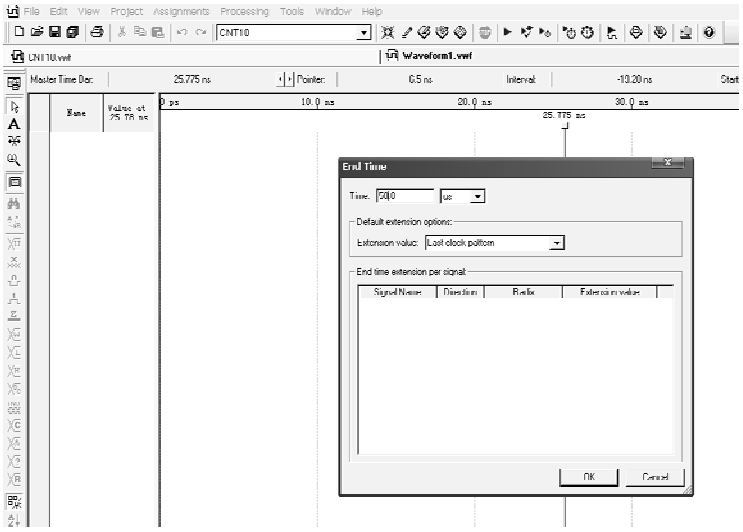


图 2-16　设置仿真时间长度

(3) 波形文件存盘。选择 File→Save As 命令，将以默认名 CNT10.vwf 存入文件夹 D:\MY_PROJECT 中。

(4) 将工程 CNT10 的端口信号名选入波形编辑器中，方法是先选择 View→Utility Windows →Node Finder 命令，弹出的对话框如图 2-17 所示。在 Filter 下拉列表框中选择 Pins:all”选项，通常默认选择此选项，然后单击 List 按钮，下方的 Nodes Found 列表框中出现 CNT10 工程的所有端口名。用户通常希望 Node Finder 对话框是浮动的，可以用右键单击此对话框边框，取消 Enable Docking 选项即可。

注意，如果此对话框不显示 CNT10 工程的端口引脚名，需要重新编译一次，即选择 Processing→Start ComPilation 命令，然后再重复以上操作过程。

最后，用鼠标将重要的端口名 CLK、CLR、ENB、COUT、COUTI 和输出总线信号 Q 分别拖到波形编辑窗口，结束后关闭 Node Finder 对话框。

单击波形窗口左侧的“全屏显示”按钮，使之全屏显示，然后单击“放大缩小”按钮，再在波形编辑区域右击，使仿真坐标处于适当位置，如图 2-17 上方所示，这时仿真时间横坐标设定在数十微秒数量级。

(5) 编辑输入波形(输入激励信号)。单击图 2-17 窗口的时钟信号名 CLK，使之变成蓝色条，再单击左列的时钟设置按钮，如图 2-18 所示，在弹出的如图 2-19 所示的时钟 Clock 设置对话框中设置 CLK 的时钟周期为 1μs；Duty cycle 是占空比，默认为 50，即 50%占空比。然后再分别设置 CLR 和 ENB 的电平。

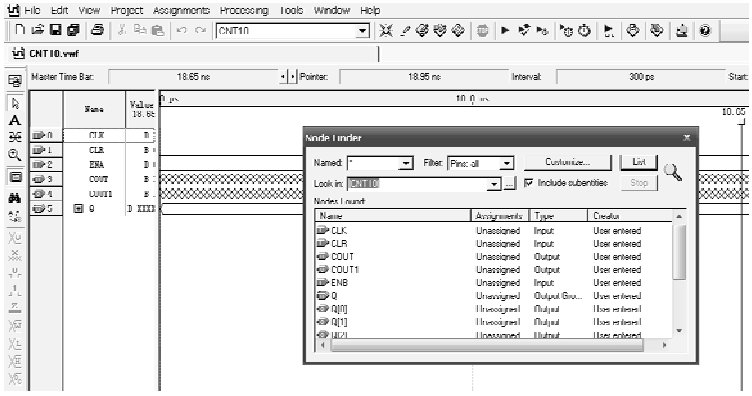


图 2-17　从 Nodes Found 列表框向波形编辑器拖入信号名

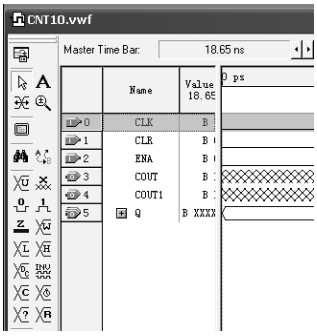


图 2-18　准备给 CLK 设置时钟

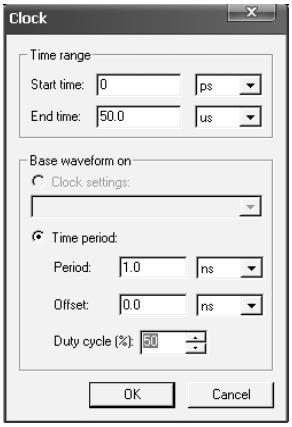


图 2-19　为 CLK 设置周期

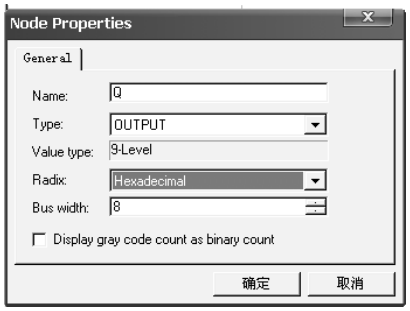


图 2-20　为总线 Q 设置数制 Radix

(6) 总线数据格式设置。单击如图 2-18 所示的输出信号 Q 左边的加号，则能展开此总线中的所有信号；如果双击此加号左边的信号标记，将弹出该信号数据格式的设置对话框，如图 2-20 所示。在 Radix 下拉列有 7 种选择，这里可以选择十六进制整数 Hexadecimal 表达方式。最后设置好的激励信号波形图如图 2-21 所示。

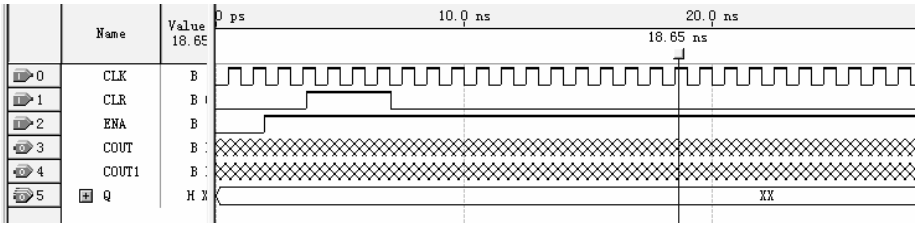


图 2-21　设置好的激励波形图

(7) 仿真器参数设置。选择 Assignment→Settings 命令，在 Settings 对话框中选择 Category →Simulator Settings，如图 2-22 所示，在右侧的 Simulation mode 选项下选择 Timing(通常为默认选项)，即选择时序仿真，并选择仿真激励文件名 CNT10.vwf(通常为默认选项)；选择 Run Simulation until all vector stimuli are used(全程仿真)等。

(8) 启动仿真器。此时所有设置完毕。

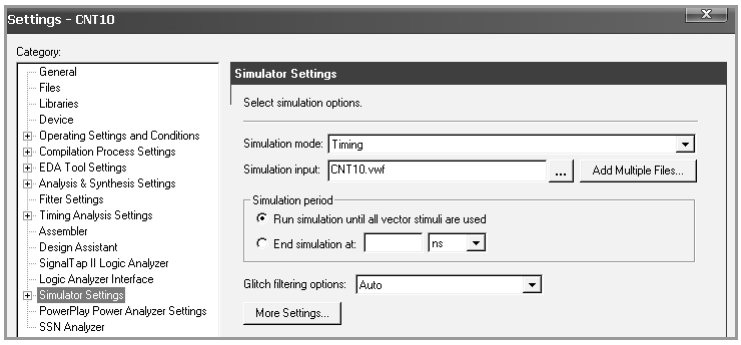


图 2-22　选择仿真约束和控制

选择 Processing→Start Simulation 命令，直到出现 Simulation was successful，仿真结束。

(9) 观察仿真结果。仿真波形文件 Simulation Report 通常会自动弹出，如图 2-23 所示。注意 Quartus II 的仿真波形文件中，波形编辑文件(*.vwf)与波形仿真报告文件(Simulation Report)是分开的，而 MAX+plus II 的激励波形编辑于仿真报告波形文件是合二为一的。完成后需对波形文件再次存盘。

如果在启动仿真运行(Processing→Run Simulation)后，并没有仿真完成后的波形图，而是出现文字(“Can’t open Simulation Report Window”，但报告仿真成功，则可选择 Processing →Simulation Report 命令，自行打开仿真波形报告。

如果无法展开波形显示时间轴上的所有波形图，可以右击波形编辑窗口中的任何位置，这时再选择弹出对话框的 Zoom 选项，在出现的下拉菜单中选择 Fitin Window 命令。

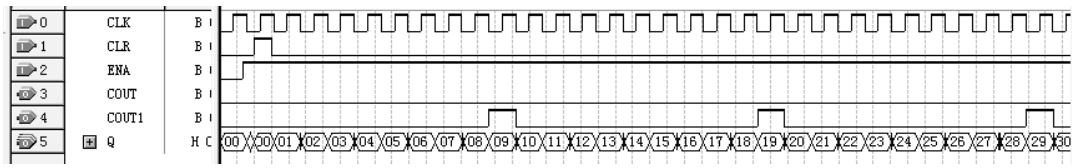


图 2-23　图 2-2 电路的仿真波形输出

由图 2-23 可知，电路功能符合设计要求。CLR 对系统的清 0 是高电平有效；ENB 对 CLK 的计数使能也是高电平有效；低位进位信号 COUT1，逢 9 产生进位脉冲。

2.4　引脚设置和编程下载

为了能对此计数器进行硬件测试，应将其输入输出信号锁定在芯片确定的引脚上，编译后下载，以便对电路设计进行硬件测试。

最后当硬件测试完成后，还必须对配置芯片进行编程，完成 FPGA 的最终开发。

2.4.1　引脚锁定

为了便于说明，在此选择 KX-7C5E+系统(参考附录，以下简称为5E+系统)，确定引脚分别为：主频时钟 CLK 接键 8：K8(对应 Pin 69)； 复位清 0 信号 CLR 则接键 7：K7(对应 Pin68)，注意这些键按下为低电平，松开为高电平。所以，在实验中要始终按住键 K7，才能使计数正常进行，否则需要为电路中的 CLR 输入口加入一个反相器。

计数使能 ENB 接键 6(对应 Pin 67；高位进位 COUT 接发光管 D8(对应 Pin 11，实验板左上角)；低位进位 COUT1 接发光管 D7(对应 Pin 10)；双 4 位二进制数据输出 Q[7…0]可由数码 A 来显示十进制数高位，数码 B 显示十进制数低位；Q[7…0]分别接引脚 43、44、46、32、34、38、39、42。对于其他 FPGA 开发板要作适当改变。确定了锁定引脚编号后就可以完成以下引脚锁定操作了：

(1) 假设现在已打开了 CNT10 工程。如果刚打开 Quartus II，应选择 File→Open Project 项，并单击工程文件 CNT10，打开此前已设计好的工程。

(2) 选择 Assignments→Assignment Editor，即进入如图 2-24 所示的 Assignment Editor 编辑窗口。在 Category 中选择 Locations。

(3) 双击 To 栏的 new，即出现一个按钮，单击此按钮，选择 Node Finder 命令，如图2-24 所示。在弹出的如图 2-25 所示的对话框中选择本工程要锁定的端口信号名，单击 OK 按钮后，所有选中信号名即进入图 2-24 的 To 栏内。然后在每个信号对应的 Location 栏内输入引脚号即可。完成后即如图 2-26 所示。

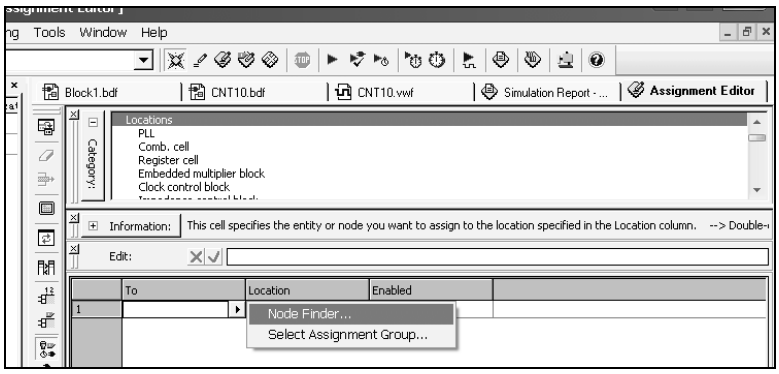


图 2-24　利用 Assignment Editor 编辑器锁定 FPGA 引脚

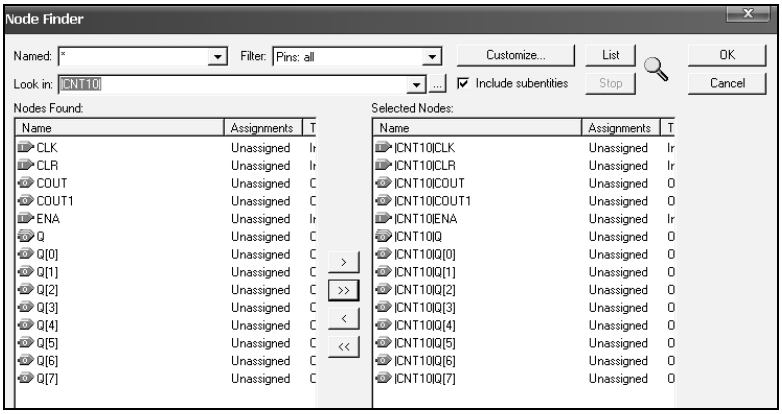
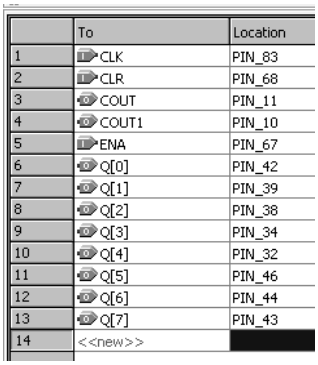


图 2-25　选择需要锁定的引脚信号

在 Assignment Editor 窗口中还能对引脚作进一步的设定。如在 I/O Standard 栏，配合芯片的不同 I/O Bank 上加载的 VCCIO 电压，选择每一信号的 I/O 电压；在 Reserved 栏，可对某些空闲的 I/O 引脚的电气特性作设置。

(4) 存储这些引脚锁定的信息后，必须再编译一次，才能将引脚锁定信息编译进编程下载文件中。此后就可以将编译好的 SOF 文件下载到实验系统的 FPGA 内了。

(5) 引脚锁定还能用更直观的图形方式来完成，即选择 Assignments Pins 项，将弹出目标器件的引脚图编辑窗口，将编辑窗口左侧的信号名逐个拖入右侧器件对应引脚上即可。注意，这种方法适用于引脚数量较少的目标器件。



	To	Location
1	CLK	PIN_83
2	CLR	PIN_68
3	COUT	PIN_11
4	COUT1	PIN_10
5	ENA	PIN_67
6	Q[0]	PIN_42
7	Q[1]	PIN_39
8	Q[2]	PIN_38
9	Q[3]	PIN_34
10	Q[4]	PIN_32
11	Q[5]	PIN_46
12	Q[6]	PIN_44
13	Q[7]	PIN_43
14	<<new>>	

图 2-26　引脚锁定对话框

2.4.2　配置文件下载

引脚锁定并编译完成后，Quartus II 将生成多种形式的针对所选目标 FPGA 的编程文件。其中最主要的是 POF 和 SOF 文件，前者是编程目标文件，用于对配置器件编程；后者是静态 SRAM 目标文件，用于对 FPGA 直接配置，在系统直接测试中使用。这里首先将 SOF 格式配置文件通过 JTAG 口载入(配置进)FPGA 中进行硬件测试。步骤如下：

(1) 打开编程窗和配置文件。首先将实验系统和 USB-Blaster 编程器连接，打开电源。选择 Tools→Programmer，弹出如图 2-27 所示的编程窗口。在Mode 下拉列表中有 4 种编程模式可供选择：JTAG、Passive Serial、Active Serial 和 In-Socket。为了直接对 FPGA 进行配置，在编程窗口的编程模式 Mode 中选择 JTAG(默认)，并选中下载文件右侧的 Program/Configure 复选框。注意，要仔细核对下载文件路径与文件名。如果此文件没有出现或有错，可单击左侧的 Add File 按钮，手动选择配置文件 CNT10.sof。

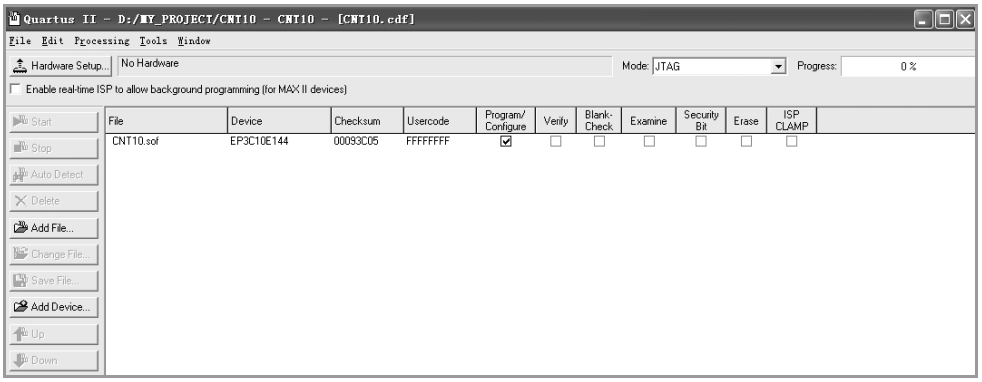


图 2-27　选择编程下载文件和下载模式

(2) 设置编程器。若是初次安装的 Quartus II，在编程前必须进行编程器的选择。若准备选择 USB-Blaster 编程器，可单击 Hardware Setup 按钮(图 2-27)，设置下载接口方式，在弹出的如图 2-28 所示的 Hardware Setup 对话框中，打开 Hardware Settings 选项卡，选择其中的 USB-Blaster 选项之后，单击 Close 按钮，关闭对话框即可。这时会在编程窗右上显示出编程方式：USB-Blaster，如图 2-29 所示。

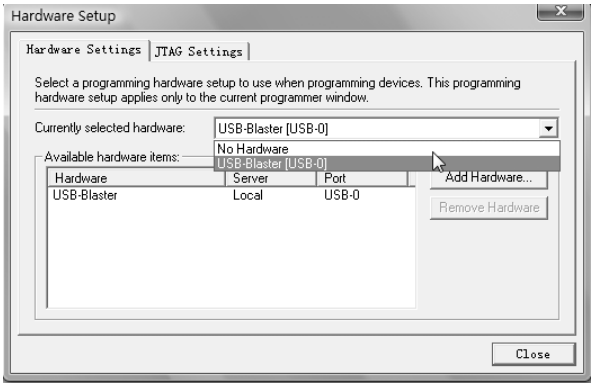


图 2-28　加入编程下载方式

最后单击 Start 按钮，即进入对目标器件 FPGA 的配置下载操作。当 Progress 显示出 100%，以及在底部的处理信息栏中出现“Configuration Succeeded”时，表示编程成功。注意，如果有必要，可再次单击 Start 按钮，直至编程成功。

(3) 测试 JTAG 接口。如果要测试 USB-Blaster 编程器与 FPGA 的 JTAG 口是否连接好，可以单击图 2-27 中的 Auto Detect 按钮，看是否能读出实验系统上 FPGA 目标器件的型号。

(4) 硬件测试。成功下载 CNT10.sof 后，可根据对不同键所定义的功能进行操作测试，结合仿真波形图，观察数码管和发光管的工作情况，通过硬件功能进一步了解此项设计的功能与性能，了解计数器工作情况。

2.4.3　AS 模式直接编程配置器件

为了使 FPGA 在上电启动后仍然保持原有的配置文件，并能正常工作，必须将 POF 配置文件烧写进专用的配置芯片 EPCSx 中。EPCSx 是 Cyclone/Ⅱ/Ⅲ等系列器件的专用配置器件，为 Flash 存储结构，编程周期 10 万次。若选择 Active Serial(AS)编程模式，编程器选择 USB-Blaster。编程流程如下：首先在图 2-27 中的 Mode 下拉列表选择 Active Serial Programming 编程模式，打开编程文件，选中文件 CNT10.pof 后，并选中 3 个编程操作选项，如图 2-29 所示；然后单击 Start 按钮，编程成功后将在下方出现相关的信息。此后每次上电，FPGA 都能被 EPCS4 自动配置，进入正常工作状态。

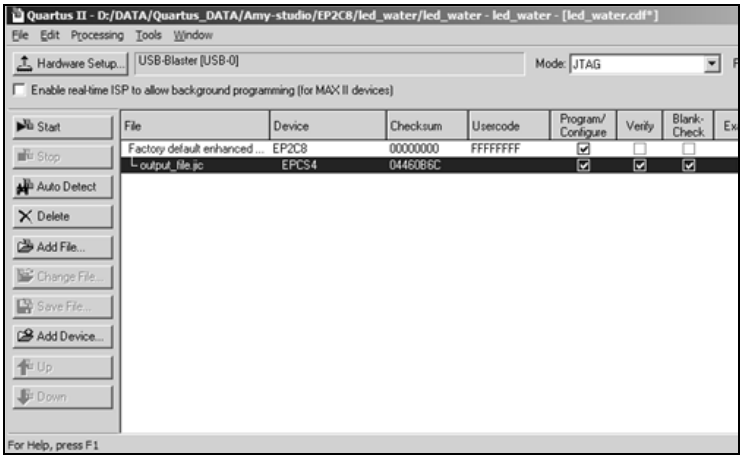


图 2-29　AS 模式编程窗口

2.4.4　JTAG 间接模式编程配置器件

事实上不用 AS 模式，而只用 JTAG 口也能对 EPCS 器件进行配置，即间接配置模式。由于考虑到 AS 直接模式下载涉及复杂的保护电路，为了能更可靠地下载，下面介绍利用 JTAG 口对 EPCS 器件进行间接配置的方法。具体方法是先将 SOF 文件转化为 JTAG 间接配置文件，再通过 FPGA 的 JTAG 口为 EPCS 器件编程。流程如下：

1. 将 SOF 文件转化为 JTAG 间接配置文件

选择 File→Convert Programming Files 命令，在弹出的如图 2-30 所示的对话框中作如下选择：

(1) 首先在 Programming File type 下拉列表框中选择输出文件类型为 JTAG 间接配置文件类型：JTAG Indirect Configuration File，后缀为.jic。

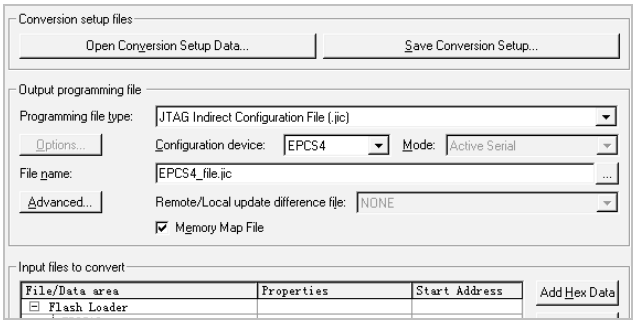


图 2-30　设定 JTAG 间接编程文件

(2) 然后在 Configuration device 下拉列表中选择配置器件型号，这里选择 EPCS4。

(3) 再在 File name 文本框中输入输出文件名，如 EPCS4_file.jic。

(4) 单击最下方 Input files to convert 选项组中的 Flash Loader 项，然后单击右侧的 Add Device 按钮，这时将弹出如图 2-31 所示的 Select Device 器件选择对话框。在此对话框中首先在左栏中选定目标器件的系列，如 Cyclone III；再于右栏中选择具体器件 EP3C10。单击(选中)Input files to convert 选项组中的 SOF Data 项，然后单击右侧的 Add File 按钮，选择 SOF 文件 CNT10.sof，如图 2-32 所示。为了使 EPCS4 能腾出空间以利于今后可能的应用，如 SOPC 设计，需要压缩后进行转换。所以首先单击 Input files to convert 选项中的文件名 CNT10.sof，然后单击右下方的 Properties 按钮，在弹出的对话框中选中 Compression 复选框，如图 2-32 所示。最后单击 Generate 按钮，即可生成所需要的编程配置文件。

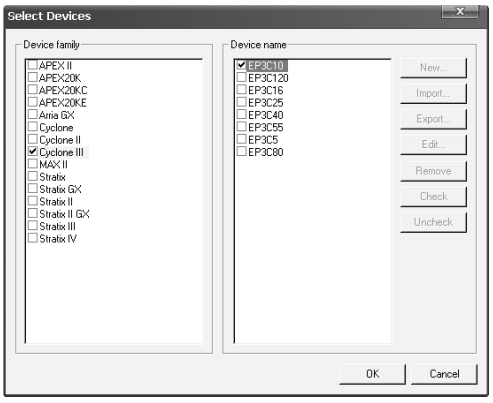


图 2-31　选择目标器件 EP3C10

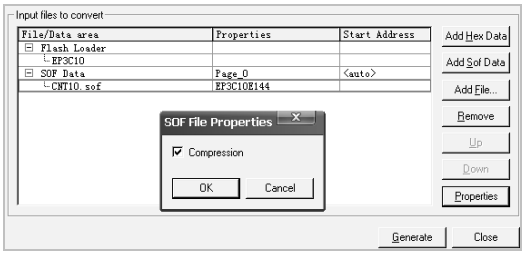


图 2-32　选择文件压缩

2. 下载 JTAG 间接配置文件

选择 Tool→Programmer 命令，选择 JTAG 模式，加入 JTAG 间接配置文件 EPCS4_File.jic，如图 2-33 所示，作必要的选择后，单击 Start 按钮进行编程下载。

为了证实下载后系统是否能正常工作，在下载完成后，必须关闭系统电源，再打开电源，以便启动 EPCS 器件对 FPGA 的配置，然后观察计数器的工作情况。

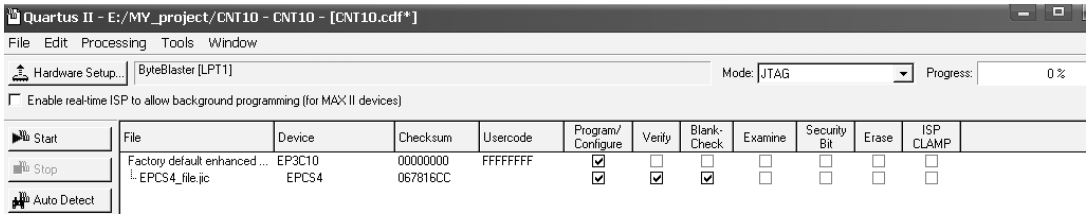


图 2-33　用 JTAG 模式经由 FPGA 对配置器件 EPCS4 进行间接编程

2.4.5　USB-Blaster 编程配置器安装方法

在初次使用 USB-Blaster 编程器前，需首先安装 USB 驱动程序，方法为：将 USB-Blaster 编程器的一端插入 PC 机的 USB 口，这时会弹出一个 USB 驱动程序对话框，根据对话框的引导，选择搜索驱动程序，这里假定 Quartus II 安装在 E 盘，则驱动程序的路径为 E:\altera\Quartus90\drivers\usb-blaster。安装完毕后，打开 Quartus II，选择编程器，单击左上角的 Hardware Setup 按钮，在弹出的对话框中选择 USB-Blaster，双击之，此后就能按照前面介绍的方法使用了。对于 In-System Memory Editor 等功能的使用也一样。

2.5　层次化设计

本节将利用以上设计模块构建一个 6 位十进制计数器，并以此引导读者学习基于原理图编辑器的层次化设计方法，而此计数器将成为数字频率计的一个重要部件。

为了利用以上完成的两位十进制计数器模块CNT10 来构建此计数器，须首先包装这个 CNT10 模块，并在更高的设计层次上作为元件入库。这有必要构建一个新的工程，在此工程中调用已入库的元件 CNT10，以便构成所需要的电路。步骤如下：

1. 构建元件符号

打开 CNT10 工程，然后打开此工程的原理图编辑窗口。选择 File→Create/Update→Create Symbol Files for current File 命令，如图 2-34 所示，即可将当前原理图文件 CNT10.bdf 变成一个包装好的单一元件(Symbol)，并被放置在工程路径指定的目录中以备调用，元件名为 CNT10.bsf。

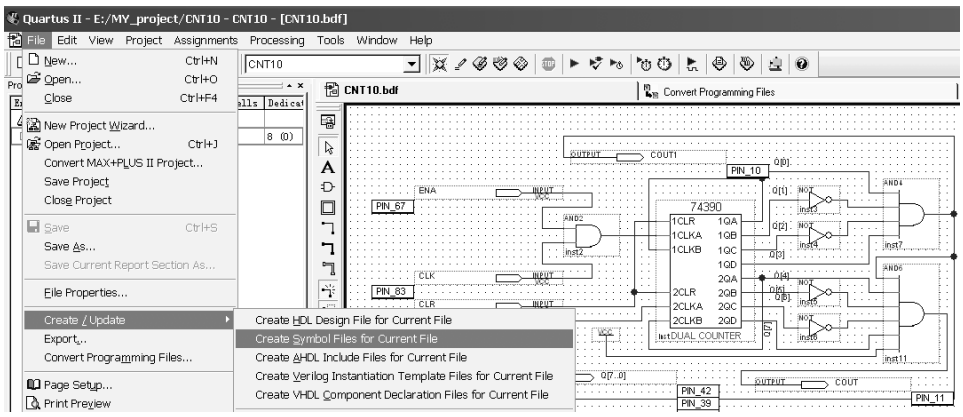


图 2-34　将当前电路原理图设计生成为一个元件(Symbol)模块

2. 构建顶层文件

可以将前面 2.4 节的工作看成是完成了一个底层元件的设计，并被包装入库。现在可以利用这个设计好的元件完成更高层次的项目设计，即设计一个 6 位十进制计数器。首先选择 File→New 命令，在 New 对话框中的 Design Files 中选择原理图文件类型 Block Diagram/Schematic File，打开原理图编辑窗口；再选择 File→Save As 命令，保存这个原理图空文件，可命名为 TOP.bdf；最后选择 File→New Project Wizard，创建一个新工程，工程名为 TOP。目标器件仍然为 EP3 C 10E144。此时 TOP.bdf 是一个没有元件的空原理图文件。然后在此新的原理图编辑窗口打开元件调用对话框，在左上角的 Libraries 栏选中元件名 CNT10，如图 2-35 所示。

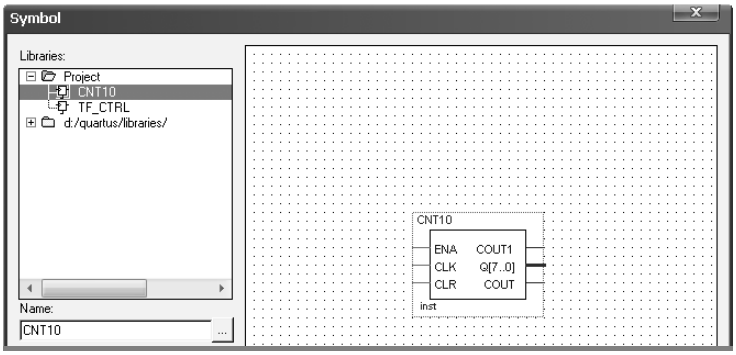


图 2-35　在高一层原理图的当前工程路径中调入元件 CNT10

此元件即为已包装入库的 2 位十进制计数器 CNT10，将它调入原理图编辑窗口中。这时如果对编辑窗口中的元件 CNT10 双击，将弹出此元件内部的原理图。最后根据如图 2-36 所示，再调入 74374b 等元件，在此原理图编辑窗口构建一个 6 位十进制计数器。注意，在元件库中含有 74374 和 74374b 两种具有同类逻辑功能但以不同端口表述的 74 系列器件，74374b 是以总线端口方式表达的器件模块。

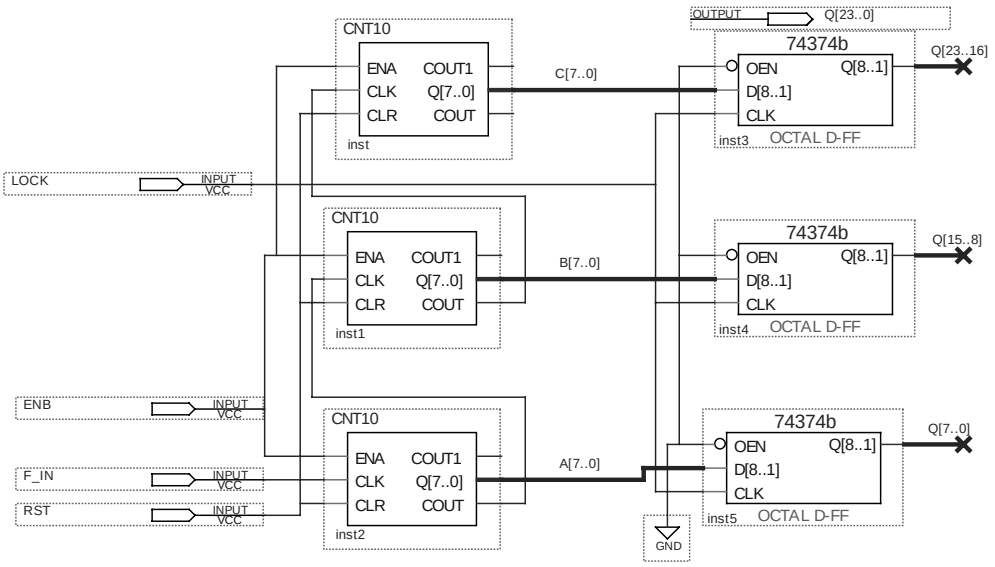
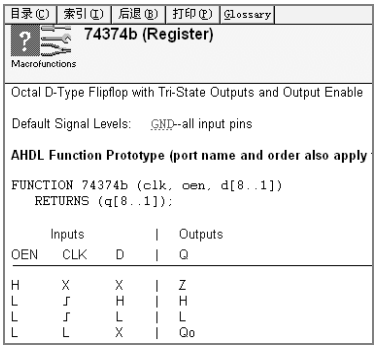


图 2-36　基于元件 CNT10 扩展的 6 位十进制计数器

3. 功能分析和全程编译

双击图 2-36 中的 74374b，可以看到此元件低层的电路结构(如图 2-37 所示是其真值表)，它是由 D 触发器、反相器 NOT 和三态门 TRI 构成的。TRI 的控制电平时，高电平允许输出，低电平时输出口呈高阻态。因此输出使能控制 OEN 统一接地，始终允许输出。输入信号 LOCK 控制 3 个 74374b 的锁存操作，具有稳定输出显示计数值的功能。RST 是全局清 0 控制信号。ENB 是计数使能信号，高电平允许对时钟脉冲计数。为了进一步详细了解 TOP 工程的逻辑功能，首先需要对其进行编译和综合，具体步骤可依照 2.3.5 节进行。


目录(C) | 索引(I) | 后退(B) | 打印(P) | Glossary

74374b (Register)

Macrofunctions

Octal D-Type Flipflop with Tri-State Outputs and Output Enable

Default Signal Levels: GND--all input pins

AHDL Function Prototype (port name and order also apply

```
FUNCTION 74374b (clk, oen, d[8..1])
   RETURNS (q[8..1]);
```

Inputs			Outputs
OEN	CLK	D	Q
H	X	X	Z
L	↑	H	H
L	↑	L	L
L	L	X	Qo

图 2-37　74374b 真值表

4. 时序仿真

仿照以上 2.3.6 节，对 TOP 工程进行仿真测试，详细了解其逻辑功能。如图 2-38 所示的是 VWF 波形文件，即仿真激励波形文件。注意，F_IN、RST、LOCK 和 ENB 这 4 个输入信号必须由设计者根据需要测试的情况加入。

这些信号的设置顺序是：首先设置时钟，F_IN 的周期可以任意设置，只要容易辨认脉冲即可，这里的周期设定为 150ns。然后设定 ENB，它的控制功能是高电平时允许计数。接着设定计数锁存信号 LOCK，为了将 ENB 高电平期间计的脉冲数锁入 74374 中，LOCK 信号必须放在每一 ENB 之后。最后设置 RST 信号(高电平有效)，为了了解每一 ENB 高电平期间所计的脉冲数，必须在把前一次的计数值锁存后，清除计数器中的值。故 RST 信号

须在 LOCK 之后，且在下一次 ENB 高电平出现前。

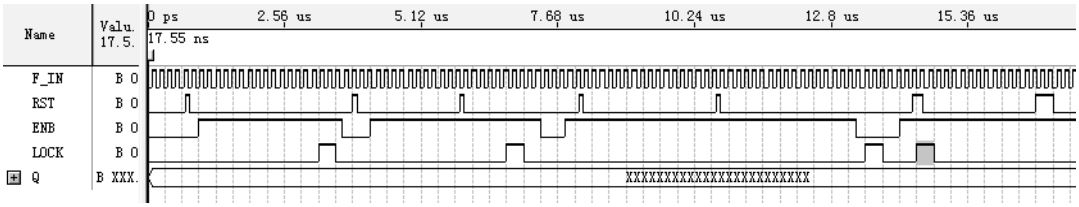


图 2-38　图 2-36 电路的仿真激励波形图或称矢量波形文件

如图 2-39 所示的是编译后的仿真波形结果。可以看到每一 LOCK 脉冲的上升沿后，7434 输出的 Q 将显示此前 ENB 高电平期间所计的数。由于有 RST 清 0 信号，每一次计数都是单独的，没有积累，不同的 ENB 高电平宽度将计不同的数值。显然，若 ENB 高电平的宽度相同，如长 1s，便能容易地获得 F_IN 的频率值。因此这个电路结构很容易构成数字频率计。如图 2-40 所示的 ENB 高电平的长度是相同的，所以当 F_IN 不变时，Q 输出的值也不变，始终等于 41。这是一个典型的 6 位十进制常规数字频率计工作时序。

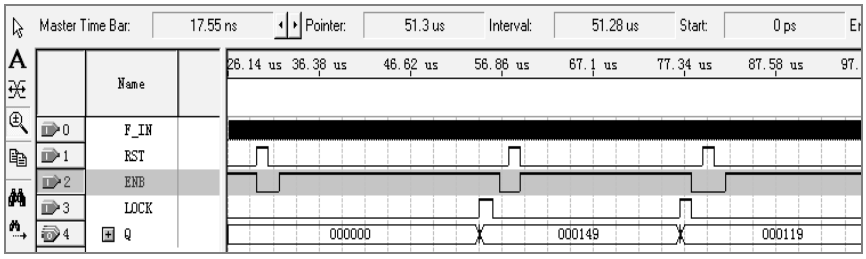


图 2-39　图 2-36 电路的仿真波形图(取 ENB 为不同脉宽)

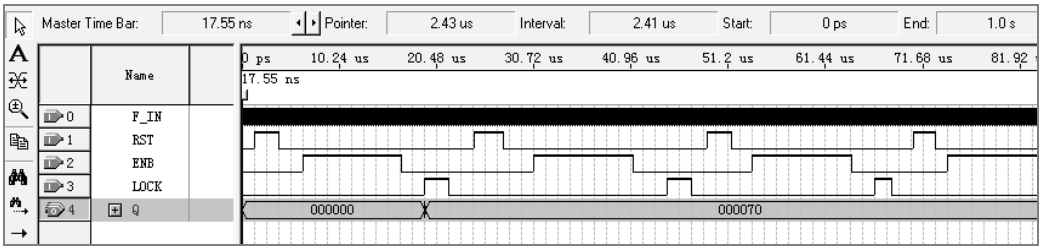


图 2-40　图 2-36 电路的仿真波形图(取 ENB 为相同脉宽)

最后，可以在实验系统上对此项设计进行硬件测试，即编译成功后将生成的 SOF 文件下载于实验系统的 FPGA 中进行测试，此工作留给读者。

2.6　6 位十进制频率计设计

本节首先设计一个控制数字频率计的时序控制器，然后完成频率计的完整设计。

2.6.1　时序控制器设计

前面已经完成了此频率计的部分电路设计，现在只要为图 2-36 的电路配上一个时序控制器就能完成设计了。而该时序控制器的工作时序必须满足图 2-40 的时序，即设计一个能自动测频的时序控制电路。要求它能按照图 2-40 所示的时序关系产生 3 个控制信号：ENB、LOCK 和 CLR (RST)，以便使频率计能顺利完成计数、锁存和清零 3 个重要的功能，其中作为周期信号 ENB 的高电平脉宽必须长 1s。根据控制信号 ENB、LOCK 和 CLR 的时序要求，如图 2-41 所示给出了相应的电路，该电路的文件名取为 tf_ctrl.bdf。

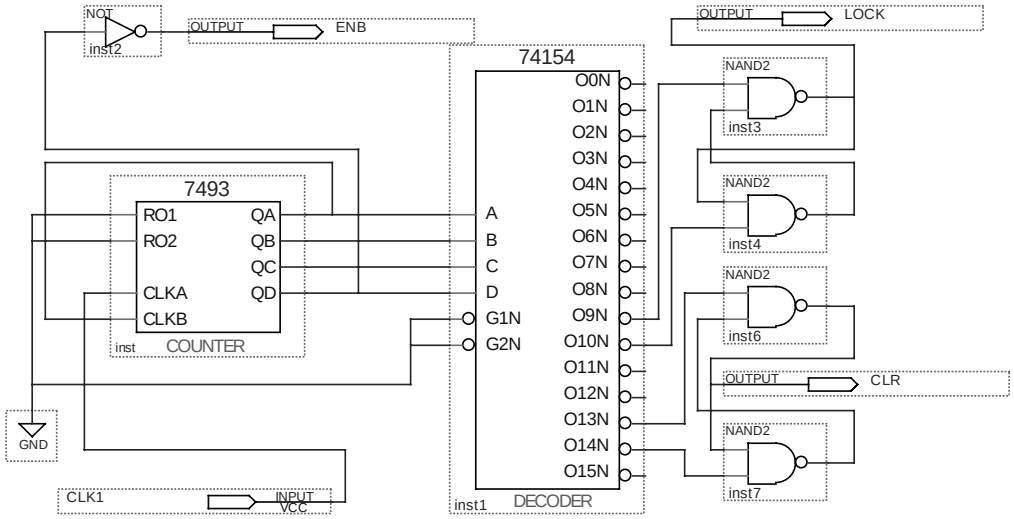


图 2-41　频率计测频时序控制电路

首先建立一个新的工程，工程名为 tf_ctrl。在此原理图编辑窗口中根据图 2-41 完成电路设计。该电路由 3 个部分组成：4 位二进制计数器 7493(根据 2.3.3 节给出的方法查阅其真值表)、4-16 译码器 74154 和两个由双与非门构成的 RS 触发器。其中的 74154 也可以用 3-8 译码器 74138 等代替，读者不妨一试。

根据图 2-41 的电路结构分析，如果 CLK1 的输入频率是 8Hz，则此电路的 ENB 输出信号的频率为 0.5Hz，脉宽为 1s，满足设计要求。

图 2-41 的仿真时序波形如图 2-42 所示，将此波形图与图 2-40 比较可知，通过图 2-41 电路的时序信号，能自动控制图 2-36 的电路，实现频率测试的目的。

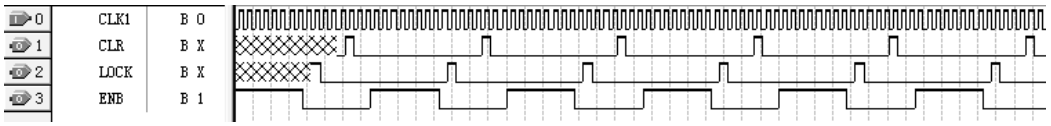


图 2-42　图 2-41 电路的仿真波形

为构成一个完整的频率计，仿照图 2-34 将电路图 2-41 变成一个底层可调用的元件。

事实上，图 2-41 所示的电路还有许多其他用途。例如可构成高速时序发生器，可通过输入不同频率的 clk 信号，或将 RS 触发器接在 74154 的不同输出端，从而产生各种不同脉

宽和频率的脉冲信号。

2.6.2　顶层电路设计与测试

打开图 2-36 的工程 TOP，在此基础上调用时序控制元件 tf_ctrl(电路见图 2-41)，构成一个完整的频率计，其结构如图 2-1 所示。图 2-1 的时序仿真波形如图 2-43 所示。其中被测信号 F_IN 的输入频率对应周期的前半部分是 100ns，后半部分是 50ns；CLK1 周期取 1000ns。但应注意，在下载于 FPGA 中作为频率计实测时，CLK1 的频率必须是 8Hz。

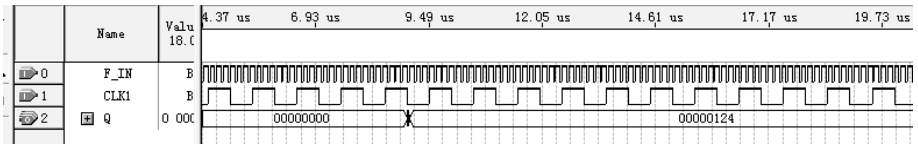


图 2-43　频率计电路图 2-1 的工作时序波形

2.7　本章小结

本章首先通过 Quartus II 工具软件进行原理图设计数字频率计示例，让读者了解 Quartus II 软件的使用和利用原理图设计数字电路的方法和设计流程。通过编辑、综合、仿真、适配、性能测试、引脚锁定和编程/下载和等常规操作技巧，使读者进一步熟悉 EDA 基本设计流程。本章力图使读者迅速掌握 Quartus II 软件的使用；利用 Quartus II 进行原理图输入方法进行逻辑电路设计技术；能够针对编辑好的逻辑电路正确建立时序仿真文件并仿真测试；能够通过时序仿真波形文件分析和判断逻辑电路的问题和功能；能将设计电路转换成电路元件，并在高层次原理图工程文件中调用这些元件，完成顶层设计；能够根据不同的 FPGA 硬件系统正确锁定系统引脚，编程下载和硬件测试；能够将 SOF 文件转换成 JIC 间接编程文件，并完成编程和测试；在 FPGA 上实现硬件测试。

2.8　习　　题

2-1　归纳利用 Quartus Ⅱ进行原理图输入设计的一般流程。

2-2　参考 Quartus II 的帮助，详细说明 Assignments 菜单中 Settings 的功能。

(1) 说明其中的 Timing Requirements&Options 的功能、使用方法和检测途径。

(2) 说明其中的 Compilation Process 的功能和使用方法。

(3) 说明 Analysis & Synthesis Setting 的功能和使用方法，以及其中的 Synthesis Netlist Optimization 的功能和使用方法。

(4) 说明 Fitter Settings 中的 Design Assistant 和 Simulator 的功能，举例说明它们的使用方法。

2-3　传统数字电路实验中，常用插导线的方法连接元件电路。根据已掌握的知识，试说明这种设计方法对系统的正常运行有何不利，为什么？

2-4　时序仿真和功能仿真有何异同点？

2-5　为什么需要FPGA配置器件？对专用配置器件EPCS4有几种编程方法？如何进行？

2-6　在什么情况下必须对设计锁定引脚？锁定引脚有几种方法？如何完成？

2-7　详细说明图 2-40 中各信号波形的功能，并说明如果没有 RST 信号，ENB 第 3 个高电平脉冲后，Q 等于几？说明图 2-1 所示的频率计中的 CLR 控制信号有何作用。

2-8　提出两个新方案，取代图 2-41 电路的功能(注意，输出波形不一定与图 2-42 完全相同，但必须能用于图 2-1 的频率计的正确控制)，用仿真波形图说明其可行性。

2-9　提出一个新方案，取代图 2-2 电路的功能，用仿真波形图说明其可行性。

2-10　基于 Quartus II 设计平台(以下各题相同)， 用 74138 和与非门实现 8421BCD 优先编码器，进行时序仿真。

2-11　用三片 74139 组成一个 5-24 线译码器，给出时序仿真波形。

2-12　用 74283 加法器和逻辑门设计实现一位 8421BCD 码加法器电路，输入输出均是 BCD 码，设 CI 和 CO 分别是低位和高位进位信号，输入为两个 1 位十进制数，输出用 S 表示，给出时序仿真波形。

2-13　设计一个 7 人表决电路，参加表决者 7 人，同意为 1，不同意为 0，同意者过半则表决通过，绿指示灯亮；表决不通过则红指示灯亮，给出时序仿真波形。

2-14　设计一个周期性产生二进制序列 01001011001 的序列发生器，用移位寄存器或用同步时序电路实现，并用时序仿真器验证其功能。

2-15　用 D 触发器构成按循环码(000－001－011－111－101－100－000)规律工作的六进制同步计数器，给出时序仿真波形。

2-16　应用 4 位全加器和 74374 构成 4 位二进制加法计数器。

2-17　用 74194、74273、D 触发器等器件组成 8 位串入并出的转换电路，要求在转换过程中数据不变，只有当 8 位一组数据全部转换结束后，输出才变化一次，进行时序仿真。如果使用 74299、74373、D 触发器和非门来完成上述功能，应该用怎样的电路？

2-18　用一片 74163 和两片 74138 构成具有 12 路脉冲输出的数据分配器。要求在原理图上标明第 1 路到第 12 路输出的位置。若改用 74195 代替 74163，试完成同样的设计，给出时序仿真波形。

2-19　用 7490 设计模为 872 的计数器，且输出的个位、十位、百位都应符合 8421 码权重。

2-20　用 74161 设计一个 97 分频电路，用置 0 和置数两种方法实现。

第3章　VHDL结构和要素

从本章起，开始介绍 VHDL 语言的语法知识及其程序设计方法。本章有两大重点：一、VHDL 程序的基本结构，从整体结构上认识 VHDL 程序，重点介绍 VHDL 程序中各组成部分的语法格式及其在描述数字电路时所起的作用。二、VHDL 语言的基本要素，主要包括词法单元、数据对象、数据类型、运算操作符以及属性。

VHDL 程序主要包括库和程序包调用、实体声明和结构体三部分。其中库和程序包调用使程序能够使用 VHDL 语言体系已经定义好的各种数据类型和函数等；实体声明用于描述模块的对外接口；结构体用于描述模块内部的功能结构。

VHDL 语言的基本要素是 VHDL 语言体系的主要组成部分。VHDL 程序书写过程中应区分各类词法单元，避免出现将关键字用作标识符等的低级错误。深刻理解 VHDL 的各种数据类型，特别是常用数据类型的定义和使用规范，掌握运算符的使用规则，它们是进行 VHDL 程序设计的基础。

VHDL 和其他高级描述语言一样，是一种硬件描述语言。VHDL 语言是硬件电路实现功能的描述，语言结构、要素和描述方法都和硬件密切联系，在学习时应多立足硬件结构考虑，并注意和高级语言程序的区别。在此对初学者提出以下几点建议：

- 注意 VHDL 编程和高级语言编程的区别；
- 注意 VHDL 语言的可综合与可仿真特性；
- 注意基本模块的 VHDL 设计方法；
- 语法学习“贵精不贵多，靠练不靠背”。

3.1　VHDL 程序基本结构

VHDL 语言通常包含实体(Entity)、结构体(Architecture)、配置(Configuration)、库(Library)和程序包(Package)4 个部分。VHDL 程序设计基本结构可以用图3-1 表示，后面将详细介绍此结构。

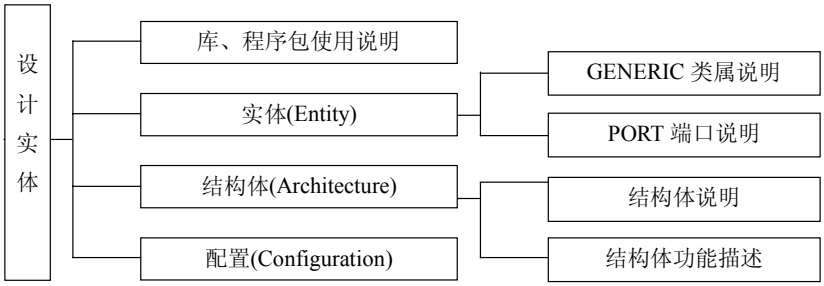


图 3-1　VHDL 程序设计基本结构

其中实体用于描述所设计的模块或系统的外部接口信号，包括端口的数目、方向和类型，其作用相当于传统设计方法中所使用的 IC 元件符号；结构体用于描述系统内部的结构和行为，对应于原理图、逻辑方程；建立输入和输出之间的关系；配置语句安装具体元件到实体—结构体对，可以被看做是设计的零件清单；库和程序包：库是专门存放预编译程序包的地方。包集合存放各个设计模块共享的数据类型、常数和子程序等。

通常将所要设计的逻辑电路看做一个实体。VHDL 从实体与实体外部的接口以及实体内部的功能与结构两个方面来描述该实体，外部(可视部分、端口)，内部(不可视部分、内部功能、算法)，如图 3-2 所示的是 VHDL 对一个实体的描述。

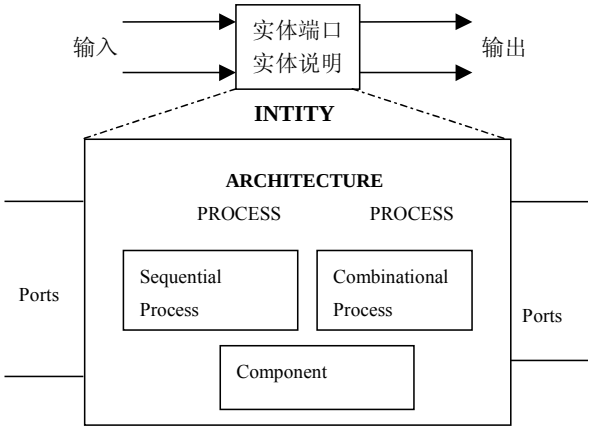


图 3-2　VHDL 基本结构框图

3.1.1　实体(ENTITY)

VHDL 实体作为一个设计实体的组成部分，其功能是对这个设计实体与外部电路的接口描述。实体描述主要用于定义模块的对外输入/输出管脚和类属参数配置。

1. 实体的一般语句格式

实体说明单元的一般语句结构如下：

```
ENTITY  实体名  IS
[   GENERIC(常数名 1：常数数据类型[:=常数值];
```

```
            常数名 2：常数数据类型[:=常数值];
            ……)；]
PORT(
        端口名 1：  端口方向   端口数据类型;
        端口名 2：  端口方向   端口数据类型;
          …….
        端口名 n：  端口方向   端口数据类型;
      );
 END   [实体名];
```

其中，[]中的内容为可选项。

实体说明单元必须以语句“ENTITY 实体名 IS”开始，以语句“END ENTITY 实体名；”结束，其中的实体名是设计者给设计实体的命名，可供其他设计实体对该设计实体进行调用。方括号内的内容表示在特定的情况下该内容可选也可不选。一个设计实体无论多大多复杂，均可在实体中定义实体名，即这个设计的实体名称，或者说是这个设计实体所描述芯片的名称。

2. 类属(GENERIC)说明语句

类属参量是一种端口界面常数，常以一种说明的形式放在实体或块结构体前的说明部分。类属为所说明的环境提供了一种静态信息通道，类属的值可以由设计实体外部提供。因此，设计者可以从外面通过类属参量的重新设定而容易地改变一个设计实体或一个元件的内部电路结构和规模。类属说明的一般格式如下：

```
GENERIC([常数名：数据类型[：设定值]{；常数名：数据类型[：设定值]})；
```

类属参量以关键词 GENERIC 引导一个类属参量表，在表中提供时间参数或总线宽度等静态信息。类属表说明用于确定设计实体和其外部环境通信的参数，传递静态的信息。类属说明在所定义的环境中的地位十分接近常数，但却能从环境(如设计实体)外部动态地接受赋值，其行为又有点类似于端口 PORT。因此，常如以上的实体定义语句那样，将类属说明放在其中，且放在端口说明语句的前面。

在一个实体中定义的可以通过 GENERIC 参数类属的说明，为它创建多个行为不同的逻辑结构。比较常见的情况是选用类属来动态规定一个实体端口的大小，或设计实体的物理特性，或结构体中的总线宽度，或设计实体中、底层中同种元件的例化数量等。一般在结构体中，类属的应用与常数是一样的。【例 3-1】实现并入串出逻辑的 GENERIC 描述。

【例 3-1】实现并入串出逻辑的 GENERIC 描述。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY   PISO   IS
GENERIC ( N: INTEGER);
PORT    ( A: IN    STD_LOGIC_VECTOR(N-1    DOWNTO    0);
          B: OUT STD_LOGIC);
END   PISO;
ARCHITECTURE     BEV    OF PISO    IS
```

```
BEGIN
PROCESS(A)
  BARIABLE  TEMP  :STD_LOGIC;
   BEGIN
       TEMP:='1';
       FOR   I  IN  A'LENGTH – 1  DOWNTO  0   LOOP
       IF  A(I) ='0'   THEN  TEMP :='0';     END IF;
       END LOOP;
   B<= TEMP;
END PROCESS;
END BEV;
```

3. 参数传递映射语句

端口映射语句 PORT MAP()是本结构体对外部元件调用和连接过程中，描述元件间端口的衔接方式；而参数传递映射语句 GENERIC MAP()也具有相似的功能，它描述相应元件类属参数间的衔接和传递方式。参数传递映射语句 GENERIC MAP()可用于从外部端口改变元件内部参数、结构规模或类属元件，其语句格式如下：

```
例化名：  元件名     Generic    MAP    (类属表);
例化名：  元件名     PORT       MAP    (类属表);
```

GENERIC MAP()和 PORT MAP()具有相似的功能和使用方法。例 3-2 给出了参数传递映射语句 GENERIC MAP()配合端口映射语句 PORT MAP()的使用范例。例 3-2 作为顶层实体例化调用例 3-1 的实体，在例 3-2 中类属变量 n 没有明确取值，它的具体取值是在 GENERIC MAP()中指定的，并在两个不同映射语句中做了不同的赋值。

【例 3-2】GENERIC MAPCI 与 PORT MAP()的使用。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  SYP  IS
PORT   (d1,d2,d3,d4,d5,d6,d7：IN STD_LOGIC;
Q1,Q2: OUT STD_LOGIC );
END   SYP;
ARCHITECTURE    BEV1   OF SYP   IS
COMPONENT   PISO    IS
GENERIC( N : INTEGER);
PORT   (a: IN   STD_LOGIC_VECTOR(N-1   DOWNTO   0);
          b: OUT STD_LOGIC );
END COMPONENT;
BEGIN
U1: PISO     GENERIC MAP (N =>2); PORT MAP (A(0)=>D1,A(1)=>D2,B=>Q1);
U2: PISO     GENERIC MAP (N =>5); PORT MAP (A(0)=>D3,A(1)=>D4,A(2)=>D5
A(3)=>D6,A(4)=>D7,C)=>Q2);
END BEV1;
```

4. PORT 端口说明

由 PORT 引导的端口说明语句是对于一个设计实体界面的说明。实体端口说明的一般

格式如下：

```
PORT(端口名：端口模式　数据类型；
　　{端口名：端口模式　数据类型})；
```

其中，端口名是设计者为实体的每一个对外通道所取的名字；端口模式是指这些通道上的数据流动方式，如输入或输出等；数据类型是指端口上流动的数据的表达格式。一个实体通常有一个或多个端口，端口类似于原理图部件符号上的管脚。实体与外界交流的信息必须通过端口通道流入或流出。由于 VHDL 是一种强类型语言，它要求只有相同数据类型的端口信号和操作数才能相互作用。

IEEE 1076 标准包中定义了 4 种常用的端口模式，各端口模式的功能及符号分别如图 3-3 和表 3-1 所示。在实际的数字集成电路中，IN 相当于只可输入的引脚，OUT 相当于只可输出的引脚，BUFFER 相当于带输出缓冲器并可以回读的引脚，而 INOUT 相当于双向引脚。

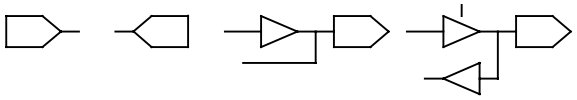


图 3-3　端口模式符号图

表 3-1　端口模式说明

端口模式	端口模式说明(以设计实体为主体)
IN	输入，只读模式，将变量或信号信息通过该端口读入
OUT	输出，单向赋值模式，将信号通过该端口输出
BUFFER	具有读功能的输出模式，可以读或写，只能有一个驱动源
INOUT	双向，可以通过该端口读入或写出信息

3.1.2　结构体(ARCHITECTURE)

结构体用于描述实体内部结构或功能，体现了输入信号和输出信号之间的逻辑关系，而实体体现的是模块的外围输出输入接口情况。结构体描述主要包含结构体声明和结构体功能描述两部分。其中，结构体声明用于定义在结构体内部使用的信号、子程序(函数和过程)和子实体例化等；功能描述部分才是用于描述实体功能和结构的。结构体内部构造的描述层次和描述内容一般可以用如图 3-4 所示的层次来说明。

一般地，一个完整的结构体由以下 3 个基本层次组成：

- 对数据类型、常数、信号、子程序和元件等元素的说明部分。
- 描述实体逻辑行为的，以各种不同的描述风格表达的功能描述语句。
- 以元件例化语句为特征的外部元件端口间的连接。

结构体将具体实现一个实体。每个实体可以有多个结构体，每个结构体对应着实体的不同结构和算法实现方案，其间的各个结构体的地位是同等的，它们完整地实现了实体的

行为，但同一结构体不能为不同的实体所拥有。结构体不能单独存在，它必须有一个界面说明，即一个实体。对于具有多个结构体的实体，必须用 CONFIGURATION 配置语句指明用于综合的结构体和用于仿真的结构体，即在综合后的可映射于硬件电路的设计实体中，一个实体只对应一个结构体。

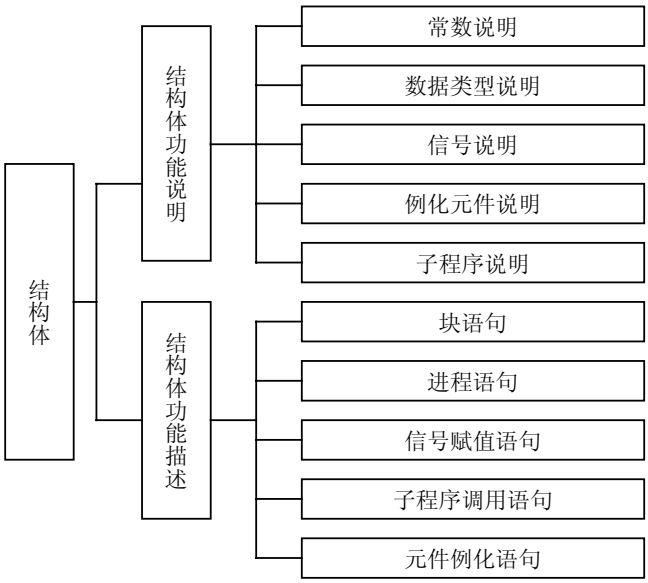


图 3-4　结构体的描述层次和描述内容

1. 结构体的一般语句格式如下：

```
ARCHITECTURE　结构体名 OF 实体名 IS
[说明语句]　　-------内部信号、常数、数据类型、函数等的定义
BEGIN
[功能描述语句]
END 结构体名;
```

其中，结构体名可以由设计者自己选择，但当一个实体具有多个结构体时，结构体的取名不可重复。结构体说明语句部分必须放在关键字 ARCHITECTURE 和 BEGIN 之间。

2. 结构体说明语句

结构体中的说明语句是对结构体的功能描述语句中将要用到的信号(Signal)、数据类型(Type)、常数(Constant)、元件(Component)、函数(Function)和过程(Procedure)等加以说明的语句。但在一个结构体中说明和定义的数据类型、常数、元件、函数和过程只能用在这个结构体中，若希望其能用到其他实体或结构体中，则需要将其作为程序包来处理。

3. 功能描述语句结构

结构体描述设计实体的具体行为，它包含以下两类语句。

- 并行语句：并行语句总是在进程语句(Process)的外部，该语句的执行与书写顺序无关，总是同时被执行。

- 顺序语句：顺序语句总是在进程语句(Process)的内部，从仿真的角度看，该语句是顺序执行的。

如图 3-4 所示的功能描述语句结构可以含有 5 种不同类型的以并行方式工作的语句结构。而在每一语句结构的内部可能含有并行运行的逻辑描述语句或顺序运行的逻辑描述语句。各语句结构的基本组成和功能分别如下：

(1) 块语句是由一系列并行执行的语句构成的组合体，它的功能是将结构体中的并行语句组成一个或多个模块。

(2) 进程语句定义顺序语句模块，用以从外部获得的信号值或内部的运算数据，向其他信号进行赋值。进程语句间是并行执行关系。

(3) 信号赋值语句将设计实体内的处理结果向定义的信号或界面端口进行赋值。

(4) 子程序调用语句用于调用过程和函数，并将获得的结果赋值于信号。

(5) 元件例化语句对其他设计实体作元件调用进行说明，并将此元件的端口与其他元件、信号或高层次实体的界面端口进行连接。

从前面的设计实例可以看出，一个相对完整的 VHDL 程序(或称为设计实体)具有比较固定的结构。至少应包括 3 个基本组成部分：库及程序包的使用说明、实体说明和实体对应的结构体说明。其中，库、程序包的使用说明用于打开(调用)本设计实体将要用到的库、程序包；实体说明用于描述该设计实体与外界的接口信号说明，是可视部分；结构体说明用于描述该设计实体内部工作的逻辑关系，是不可视部分。

3.2　子程序(SUBPROGRAM)

VHDL 子程序(SUBPROGRAM)是 VHDL 的程序模块。这个模块利用顺序语句来定义和完成算法，因此只能使用顺序语句，这一点与进程相似。子程序定义了某种算法或某个功能，用于实现数据类型转换或其他功能行为。VHDL 的子程序函数(FUNCTION)和过程(PROCEDURE)两类。VHDL 子程序常常描述了一个常用的功能模块，主程序通过语句调用其功能。其中函数调用相当于表达式，有一个返回值；过程调用相当于描述语句，没有返回值。相比进程(PROCESS)和元件例化，子程序有以下特点：

- 子程序是设计中的一个独立部分，其内部由顺序语句构成，但它与进程不同。进程可以从结构的并行语句或其他进程结构中直接读取信号值或者向信号赋值；而子程序只能通过子程序调用及与子程序的界面端口进行通信。
- 子程序描述的功能模块能够被结构体调用，但它和元件例化又有所不同。子程序调用只能是对应于当前层次的一部分，而元件例化将产生下一个新的层次。

注意，函数或过程只能定义用于完成组合逻辑的功能模块，不能定义时序逻辑功能模块。

VHDL 子程序与其他软件语言程序中的子程序的应用目的是相似的，即能更有效完成重复性工作。应特别注意，综合后的子程序映射于目标芯片中的一个相应的电路模块，且每一次语句调用都将产生对应于具有相同结构的不同的硬件模块。这与软件语言的子程序有很大不同。

子程序包含声明部分和主体部分。其中，声明部分可以在程序包声明区或结构体声明区声明；主体部分在程序包主体或结构体中描述。函数或过程定义的位置如图 3-5 所示。在结构体中声明和定义的子程序对该结构体来说是局部的，不能被其他设计层调用，如果想被其他设计层调用，则必须将子程序定义到程序包中。

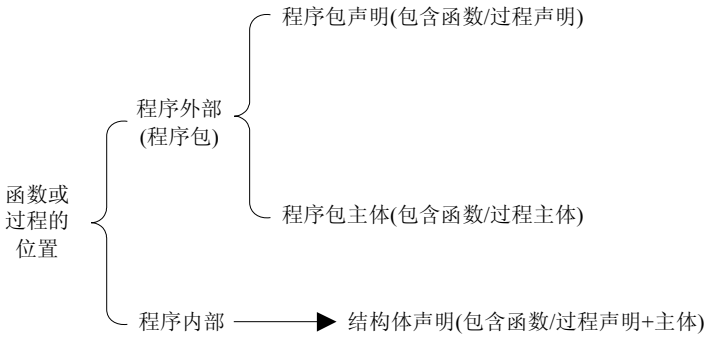


图 3-5　函数或过程在程序中的位置示意图

VHDL 子程序具有可重载的特点，即允许有许多重名的子程序，但这些子程序的参数类型及返回值的数据类型是不同的。

3.2.1　函数(FUNCTION)

函数可以包含多个输入参数，并且只能返回一个值。调用函数前必须先在程序声明部分声明和定义函数。由于函数有返回值，因此函数可以被认为是表达式的延伸和拓展。程序中任何能使用表达式的地方都可以调用函数。

定义和描述函数功能的语法格式如下：

```
FUNCTION  函数名(参数列表) RETURN    返回数据类型;
FUNCTION  函数名(参数列表) RETURN    返回数据类型    IS
函数声明;
        BEGIN
        顺序语句 1;
        顺序语句 2;
        ……
 RETURN (返回值);
END  函数名;
```

其中，各选项的功能分别如下。

- 参数列表：用于声明函数的输入端口，多个参数之间用分号隔开，但最后一个参数不需要带任何符号。函数的端口只能是输入。
- 返回数据类型：可以不附加范围限制，因为从函数的 RETURN 语句可以获得返回数据的实际宽度。
- 函数声明：用于声明在函数内部使用的变量和常量。定义变量时需要指定变量的范围或宽度。函数内部不能定义信号变量，只能定义变量或常量。

- 顺序语句：用于描述函数的功能。函数和进程一样，内部只能使用顺序语句(IF、CASE、LOOP 等)加以描述。不可以使用 WAIT 语句、Component 语句。

【例 3-3】结构体中的函数和函数调用。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  MAX_FUNC  IS
PORT(a,b,c: IN   STD_LOGIC_VECTOR(7  DOWNTO  0 );
       s: OUT  STD_LOGIC_VECTOR(7  DOWNTO  0 ) );
END  MAX_FUNC;
ARCHITECTURE   BEV  OF  MAX_FUNC  IS
FUNCTION  MAX ( X,Y: STD_LOGIC_VECTOR)  RETURN  STD_LOGIC_VECTOR  IS
    VARIABLE  Z: STD_LOGIC_VECTOR(7  DOWNTO  0);
BEGIN
    IF (X>Y)  THEN     Z:=X;
    ELSE               Z:=Y;
    END IF;
RENTURN   Z;
END  MAX;
BEGIN
S<=MAX(MAX(A,B),C);
END BEV;
```

【例 3-4】程序包中的函数和函数的调用。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE packexp IS                ----------在包中函数首的声明部分
  FUNCTION max( a,b: IN STD_LOGIC_VECTOR) RETURN STD_LOGIC_VECTOR;
    --FUNCTION func1( a,b,c: REAL)   RETURN REAL;
    --FUNCTION "*"( a,b: INTEGER)   RETURN INTEGER ;
    --FUNCTION as2( SIGNAL in1,in2: REAL) RETURN REAL;
  END ;
PACKAGE BODY packexp IS       ----------在包中函数体定义部分
  FUNCTION max( a,b:IN STD_LOGIC_VECTOR) RETURN STD_LOGIC_VECTOR IS
    BEGIN
    IF a>b THEN RETURN a;
     ELSE          RETURN b;
    END IF;
  END FUNCTION max;
END;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.packexp.ALL;
ENTITY axamp IS
  PORT(dat1,dat2:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       dat3,dat4:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       out1,out2:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END;
ARCHITECTURE bhv OF axanp IS
  BEGIN
```

```
    out1<= max(dat1,dat2);
  PROCESS(dat3,dat4)
    BEGIN
      out2<=max(dat3.dat4);
    END PROCESS;
END;
```

3.2.2　过程(PROCEDURE)

子程序的另外一种形式是过程(PROCEDURE)。定义和描述过程功能的语法格式如下：

```
PROCEDURE   过程名(参数列表);                    ------过程首
PROCEDURE   过程名(参数列表)IS
过程声明;
        BEGIN
        顺序语句 1;                              ------过程体
        顺序语句 2;
        ……
  RETURN (返回值);
END  过程名;
```

与函数一样，过程也由过程首和过程体构成。过程首也不是必须的，过程体可以独立存在和使用。在进程或结构体中不必定义过程首，而在程序包中必须定义过程首。

过程首由过程名和参数列表组成。参数列表可以对常数、变量和信号 3 类数据对象目标作出说明，并用关键词 IN、OUT、INOUT 定义这些参数的工作模式，即信息流向。如果没有定义模式，则默认为 IN。

过程体是由顺序语句组成的，过程的调用级启动了对过程体顺序语句的执行。与函数一样，过程体的说明部分只是局部的，其中的各种定义只适用于过程体的内部。过程体的顺序语句部分可包含任何顺序执行的语句。

在不同的调用环境中，可以有两种不同的语句方式对过程调用，即顺序语句方式或并行语句方式。对于前者，在一般的顺序语句自然执行的过程中，一个过程被执行，则属于顺序语句方式，因为这时它只相当于一条顺序语句的执行；对于后者，一个过程相当于一个小的进程，当这个语句处于并行语句环境中时，其过程定义的任意目标参数发生改变时，将启动过程调用，这时的调用是属于并行语句的方式。以下是两个过程体的使用实例。

```
PROCEDURE ADDER(SIGNAL A，B: IN STD_LOGIC;     --定义过程名为 ADDER
    SIGNAL SUM: OUT STD_LOGIC);
ADDER(A1，B1，SUM1);    --并行过程调用
…                       --在此，A1、B1、SUM1 即为分别对应于 A、B、SUM 的关联参量名
PROCESS(C1，C2);        --进程语句执行
BEGIN
ADDER(C1，C2，S1);      --顺序(串行)过程调用，在此 C1、C2、S1 即为分别对应于 A、B、SUM
                          的关联参量名
END PROCESS;
```

【例 3-5】结构体中过程的并行调用。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
 ENTITY    MAX_MIN IS
        PORT(
            INP1,INP2: IN INTEGER RANGE 0 TO 255;
       MAX_OUT,MIN_OUT: OUT INTEGER RANGE 0 TO 255
         );
END MAX_MIN;
ARCHITECTURE    BEV OF MAX_MIN IS
PROCEDURE    SORT (IN1, IN2: IN INTEGER;SIGNAL MAX,MIN :OUT INTEGER) IS
       BEGIN
        IF (IN1>IN2)    THEN
        MAX<=IN1;      MIN<=IN2;
    ELSE
            MAX<=IN2;      MIN<=IN1;
        END IF;
END ;
BEGIN
        SORT (INP1,INP2,MAX_OUT,MIN_OUT);
END BEV;
```

【例 3-6】进程中过程的顺序调用。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
 ENTITY    MAX_MIN IS
        PORT(
            INP1,INP2: IN    INTEGER RANGE 0 TO 255;
    MAX_OUT,MIN_OUT: OUT INTEGER RANGE 0 TO 255;
    ENA: IN    STD_LOGIC
         );
END MAX_MIN;
ARCHITECTURE    BEV OF MAX_MIN IS
PROCEDURE    SORT (IN1, IN2: IN INTEGER;SIGNAL MAX,MIN :OUT INTEGER) IS
       BEGIN
        IF (IN1>IN2)    THEN
        MAX<=IN1;      MIN<=IN2;
    ELSE
                MAX<=IN2;      MIN<=IN1;
            END IF;
END;
BEGIN
        PROCESS(ENA,INP1,INP2)
       BEGIN
         IF (ENA='1')    THEN
        SORT (INP1,INP2,MAX_OUT,MIN_OUT);
      END IF;
  END    PROCESS;
END BEV;
```

从以上函数和过程的介绍中可知，函数和过程的区别如下：

- 函数包括 0 个或多个 IN 模式的输入参数，且只能返回一个值。输入参数类型只能是 constant(默认)或 signal，而不能是 variable。实际应用中定义类型为默认的 constant 即可。
- 过程可以包含任意多个 IN、OUT 或 INOUT 类型的参数，参数数据类型可以使 signal、variable 或 constant。输入参数(IN)的默认类型 constant，输出(OUT、INOUT)默认定义为 variable 类型。实际应用中，输入参数类型使用默认 constant，输出参数类型定义为 signal 比较方便。
- 函数调用相当于表达式的作用，而过程调用则为一条语句。过程调用可以当做并行语句，也可以当做顺序语句使用。
- 对于过程和函数，wait 和元件例化语句都不可使用。
- 函数和过程可以在程序的结构体和实体中定义，也可以在程序包中定义。通常，函数和程序作为可重复使用的模块在专门的程序中定义，定义包含声明和主体(BODY)两部分。在编写程序过程中也可以直接在结构体声明中定义函数和过程，而在结构描述中调用它们。两者之中，函数更加常用，因此应多注意常见函数的书写和应用。

3.2.3　重载函数

VHDL 允许以相同的函数名定义函数，即重载函数。但这时要求函数中定义的操作数据必须具有不同的数据类型，以便调用时方便区分不同功能的同名函数。即同样名称的函数可以用不同的数据类型作为此函数的参数定义多次，以此定义的函数称为重载函数(Overload Function)。函数还可以用任意位矢长度来调用。例 3-7 就是一个完整的重载函数 max 的定义和调用实例。

【例 3-7】重载函数的定义和调用实列。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE packexp IS

 FUNCTION max(a,b:IN STD_LOGIC_VECTOR)
  RETURN STD_LOGIC_VECTOR;

 FUNCTION max(a,b:IN BIT_VECTOR)
  RETURN BIT_VECTOR;

 FUNCTION max(a,b:IN INTEGER)
  RETURN INTEGER;
END;
```

```
PACKAGE BODY packexp IS
  FUNCTION max(a,b:IN STD_LOGIC_VECTOR)
  RETURN STD_LOGIC_VECTOR IS
 BEGIN
  IF a>b THEN RETURN a;
   ELSE          RETURN b;
  END IF;
 END FUNCTION max;

 FUNCTION max(a,b:BIT_VECTOR)
  RETURN BIT_VECTOR IS
 BEGIN
  IF a>b THEN RETURN a;
   ELSE          RETURN b;
  END IF;
 END FUNCTION max;
   FUNCTION max(a,b:INTEGER)
  RETURN INTEGER IS
 BEGIN
  IF a>b THEN RETURN a;
   ELSE          RETURN b;
  END IF;
 END FUNCTION max;
END;
LIBRARY IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;
  USE WORK.packexp.ALL;
 ENTITY axamp IS
  PORT(a1,b1:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       a2,b2:IN BIT_VECTOR(4 DOWNTO 0);
       a3,b3:IN INTEGER RANGE 0 TO 15;
           c1:OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
           c2:OUT BIT_VECTOR(4 DOWNTO 0);
           c3:OUT INTEGER RANGE 0 TO 15);
  END;
 ARCHITECTURE bhv OF axamp IS
  BEGIN
  c1<=max(a1,b1);
  c2<=max(a2,b2);
  c3<=max(a3,b3);
  END;
```

在具有不同数据类型操作数构成的同名函数中，以运算符重载式函数最为常用。这种函数为不同数据类型间的运算带来极大的方便，例 3-8 中以加号“+”为函数名的函数即为

运算符重载函数。VHDL 中预定义的操作符如“+”、“-”、 “*”、“=>”、“AND”、“MOD”、“>”等运算符均可以被重载，以赋予新的数据类型操作功能，也就是说，通过重新定义运算符的方式，允许被重载的运算符能够对新的数据类型进行操作，或者允许不同的数据类型之间用此运算符进行运算。例 3-8 给出了一个 Synopsys 公司的程序包 STD_LOGIC_UNSIGNED 中的部分函数结构，示例没有把全部内容列出。在程序包 STD_LOGIC_UNSIGNED 的说明部分只列出了 4 个函数的函数首。在程序包体部分只列出了对应的部分内容，程序包体部分的 UNSIGNED()函数是从 IEEE.STD_LOGIC_ARITH 库中调用的，在程序包体中的最大整型数检出函数 MAXIUM 只有函数体，没有函数首，这是因为它只在程序包体内调用。

【例 3-8】重载函数应用示例。

```
LIBRARY IEEE;                              --程序包首
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE STD_LOGIC_ARITH.ALL;
PACKAGE STD_LOGIC_UNSIGNED IS
FUNCTION" +" (L: STD_LOGIC_VECTOR; R: INTEGER)
                RETURN STD_LOGIC_VECTOR;
FUNCTION "+" (L: INTEGER;    R: STD_LOGIC_VECTOR)
                RETURN STD_LOGIC_VECTOR;
FUNCTION "+" (L: STD_LOGIC_VECTOR; R: STD_LOGIC)
   RETURN STD_LOGIC_VECTOR;
FUNCTION SHR(ARG: STD_LOGIC_VECTOR;
    COUNT: STD_LOGIC_VECTOR) RETURN STD_LOGIC_VECTOR;
END STD_LOGIC_UNSIGNED;
LIBRARY IEEE;                              --程序包体
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
PACKAGE BODY STD_LOGIC_UNSIGNED IS
FUNCTION MAXIMUM(L, R: INTEGER) RETURN INTEGER IS
BEGIN
    IF L>R THEN
      RETURN    L;
ELSE
       RETURN    R;
    END IF;
END;
FUNCTION "+" (L: STD_LOGIC_VECTOR;    R: INTEGER)
RETURN    STD_LOGIC_VECTOR IS
VARIABLE RESULT: STD_LOGIC_VECTOR(L'RANGE);
BEGIN
     RESULT: = UNSIGNED(L)+R;
     RETURN    STD_LOGIC_VECTOR(RESULT);
END;
END STD_LOGIC_UNSIGNED;
```

通过此例，不但可以看到程序包中完整的函数置位形式，而且还能发现函数首的 3 个函数名都是同名的，即都是以加法运算符“+”作为函数名。以这种方式定义函数即为运

算符重载。对运算符重载(即对运算符重新定义)的函数称重载函数。

实际应用中，如果已用 USE 语句打开了程序包 STD_LOGIC_UNSIGNED，这时，如果设计实体中有一个 STD_LOGIC_VECTOR 位矢和一个整数相加，程序就会自动调用第一个函数，并返回位矢类型的值。若是一个位矢与 STD_LOGIC 数据类型的数相加，则调用第三个函数，并以位矢类型的值返回。

两个或两个以上有相同的过程名和互不相同的参数数量及数据类型的过程称为重载过程。对于重载过程，也是靠参量类型来辨别究竟调用哪一个过程的。

【例 3-9】重载过程调用示例。

```
PROCEDURE CAL(V1, V2: IN REAL; SIGNAL OUT1: INOUT    INTEGER);
PROCEDURE CAL(V1, V2: IN IN    TEGER; SIGNAL OUT1: INOUT REAL);
CAL(20.15, 1.42, SIGN1);     --调用第一个重载过程 CAL, SIGN1 为 INOUT 式的整数信号
CAL(23, 320, SIGN2);         --调用第二个重载过程 CAL, SIGN1 为 INOUT 式的实数信号
```

如前所述，在过程结构中的语句是顺序执行的，调用者在调用过程前应先将初始值传递给过程的输入参数。一旦调用，即启动过程语句，按顺序自上而下执行过程中的语句，执行结束后，将输出值返回到调用者的 OUT 和 INOUT 所定义的变量或信号中。

3.2.4　转换函数

VHDL 的转换函数中，数据类型转换函数最为常用，该类函数用于实现 VHDL 中各种数据类型的互相转换。由于 VHDL 数据类型较多，除有多种预定义的数据类型外，还有用户自定义的数据类型。与 Verilog 不同，VHDL 作为一种强类型语言，当数据类型不一致时，需要转换一致后才能给信号赋值或者完成各种运算操作。

VHDL 综合器的 IEEE 标准库里的程序包中定义了许多类型转换函数，如表 3-2 所示，设计者可以直接调用这些函数进行类型转换。另外，也可以自己编写转换函数。

表 3-2　IEEE 库类型转换函数

程　序　包	函 数 名 称	功　　能
STD_LOGIC_1164	TO_BIT	由 STD_LOGIC 转换为 BIT
	TO_BIT VECTOR	由 STD_LOGIC_VECTOR 转换为 BIT_VECTOR
	TO_STD ULOGIC	由 BIT 转换为 STD_LOGIC
	TO_ STD ULOGIC VECTOR	由 BIT_VECTOR 转换为 STD_LOGIC_VECTOR
STD_LOGIC_ARITH	CONV_INTEGER	由 UNSIGNED、SIGNED 转换为 INTEGER
	CONV_UNSIGNED	由 INTEGER、SIGNED 转换为 UNSIGNED
	CONV_ STD_LOGIC_VECTOR	由 INTEGER、UNSIGNED、SIGNED 转换为 STD_LOGIC_VECTOR
STD_LOGIC_UNSIGNED	CONV_INTEGER	由STD_LOGIC_VECTOR转换为INTEGER

3.2.5　决断函数

决断函数不可综合，主要用于 VHDL 仿真中解决信号被多个驱动时驱动信号间的竞争问题。如一个内部总线被多个信号占用时，决断函数将对多个信号占用总线做出裁决，给总线驱动一个适当的信号值。在 VHDL 中，一个信号带多个驱动源时，没有附加决断条件。

3.3　VHDL 库

在进行 VHDL 程序设计时，为了提高设计效率，有必要将一些有用的信息汇集在一个或几个库中供调用。这些信息可以是预先定义好的数据类型、子程序等设计单元的子合体或者程序包，因此可以把库看成一种用来存储预先完成的数据集合体、元件和程序包的仓库。

VHDL 语言库分为两类：一类是设计库，如在具体设计项目中用户设定的文件目录对应的 WORK 库；另一类是资源库，这是常规元件的标准模块存放的库。

3.3.1　库的种类

VHDL 程序设计中常用的库有 IEEE 库、STD 库、WORK 库、VITAL 库、自定义库。

1. IEEE 库

IEEE 库是 VHDL 设计中最为常见的库，它包含有 IEEE 标准的程序包和其他一些支持工业标准的程序包。IEEE 库中的标准程序包主要包括 STD_LOGIC_1164，NUMERIC_BIT 和 NUMERIC_STD 等。其中，STD_LOGIC_1164 是最重要、最常用的程序包，大部分基于数字系统设计的程序包都是以此程序包中设定的标准为基础的。此外，还有一些程序包虽非 IEEE 标准，但由于其已成事实上的工业标准，也都并入了 IEEE 库。这些程序包中，最常用的是 Synopsys 公司的 STD_LOGIC_ARITH(ARITHmetic functions)、STD_LOGIC_SIGNED (SIGNED ARITHmetic functions)和STD_LOGIC_UNSIGNED(UNSIGNED ARITHmetic functions)程序包。目前流行于我国的大多数 EDA 工具都支持 Synopsys 公司的程序包。一般基于大规模可编程逻辑器件的数字系统设计，IEEE 库中的 4 个程序包 STD_LOGIC_1164、STD_LOGIC_ARITH、STD_LOGIC_SIGNED 和 STD_LOGIC_UNSIGNED 已经足够使用。另外需要注意的是，在 IEEE 库中，符合 IEEE 标准的程序包并非符合 VHDL 语言标准，如 STD_LOGIC_1164 程序包，因此在使用 VHDL 设计实体之前必须以显式表达出来。

2. STD 库

VHDL 语言标准定义了两个标准程序包，即 STANDARD 和 TEXTIO 程序包，它们都被收入 STD 库中。只要是在 VHDL 应用环境中，就可随时调用这两个程序包中的所有内容，即在编译和综合过程中，VHDL 的每一设计都自动地将其包含进去了。由于 STD 库符

合 VHDL 语言标准，它定义了最基本的数据类型(Bit,bit_vector,Boolean,Integer，Real and Time)，在应用中不必如 IEEE 库那样显式表达出来。

3. WORK 库

WORK 库是用户的 VHDL 设计的现行工作库，用于存放用户设计和定义的一些设计单元和程序包。是用户自己的仓库，用户设计项目的成品、半成品模块，以及先期已设计好的元件都放在其中。WORK 库自动满足 VHDL 语言标准，在实际调用中，不必以显式预先说明。在计算机上利用 VHDL 语言进行项目设计，不允许在根目录下进行，而是必须为此设计一个目录，用于保存此项目的所有设计文件，VHDL 综合器将此目录默认为 WORK 库，但是必须注意工作库并不是这个目录的目录名，而是一个逻辑名。

4. VITAL 库

使用 VITAL 库，可以提高 VHDL 门级时序模拟的精度，因而只在 VHDL 仿真器中使用。库中包含时序程序包 VITAL_TIMING 和 VITAL_PRIMITIVES。VITAL 程序包已经成为 IEEE 标准，在当前的 VHDL 仿真器的库中，VITAL 库中的程序包都已经并到 IEEE 库中。实际上，由于各 FPGA/CPLD 生产厂商的适配工具(如 ispEXPERT Compiler)都能为各自的芯片生成带时序信息的 VHDL 门级网表，用 VHDL 仿真器仿真该网表可以得到非常精确的时序仿真结果。因此，基于实用的观点，在 FPGA/CPLD 设计开发过程中，一般并不需要 VITAL 库中的程序包。

5. 用户定义库

除了以上提到的库外，EDA 工具开发商为了便于 CPLD/FPGA 开发设计上的方便，都有自己的扩展库和相应的程序包，如 DATAIO 公司的 GENERICS 库、DATAIO 库等，以及上面提到的 Synopsys 公司的一些库。

在 VHDL 设计中，有的 EDA 工具将一些程序包和设计单元放在一个目录下，而将此目录名如 WORK 作为库名，如 Synplicity 公司的 Synplify。有的 EDA 工具是通过配置语句结构来指定库和库中的程序包，这时的配置即成为一个设计实体中最顶层的设计单元。

此外，用户还可以自己定义一些库，将自己的设计内容或通过交流获得的程序包设计实体并入这些库中。

3.3.2　库的用法

在 VHDL 语言中，库的说明语句总是放在实体单元前面，而且库语言一般必须与 USE 语言同用。库语言关键词 LIBRARY 用于指明所使用的库名。USE 语句指明库中的程序包。一旦说明了库和程序包，整个设计实体都可进入访问或调用，但其作用范围仅限于所说明的设计实体。VHDL 要求一项含有多个设计实体的更大的系统，每一个设计实体都必须有自己完整的库说明语句和 USE 语句。USE 语句的使用将使所说明的程序包对本设计实体部分全部开放，即是可视的。USE 语句的使用有以下两种常用格式：

```
USE 库名.程序包名.项目名;
USE 库名.程序包名.ALL;
```

第一种格式的作用是，向本设计实体开放指定库中的特定程序包内所选定的项目；第二种格式的作用是，向本设计实体开放指定库中的特定程序包内所有的内容。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

以上 3 条语句表示打开 IEEE 库，再打开此库中的 STD_LOGIC_1164 程序包和 STD_LOGIC_UNSIGNED.ALL 程序包的所有内容。

【例 3-10】库、程序包和函数的调用示例。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.STD_ULOGIC;
USE IEEE.STD_LOGIC_1164.RISING_EDGE;
```

此例向当前设计实体开放了 STD_LOGIC_1164 程序包中的 RISING_EDGE 函数。但由于此函数需要用到数据类型 STD_ULOGIC，所以在上一条 USE 语句中开放了同一程序包中的这一数据类型。

3.4　VHDL 程序包

为了使已定义的常数、数据类型、元件调用说明以及子程序能被更多的 VHDL 设计实体方便地访问和共享，可以将它们收集在一个 VHDL 程序包中。多个程序包可以并入一个 VHDL 库中，使之适用于更一般的访问和调用范围。这一点对于大系统开发，多个或多组开发人员并行工作显得尤为重要。

3.4.1　程序包定义

程序包就是已定义的常数、数据类型、元件调用说明、子程序的集合。

程序包的内容主要由如下 4 种基本结构组成，因此一个程序包中至少应包含以下结构中的一种。

- 常数说明：主要用于预定义系统的宽度，如数据总线通道的宽度。
- 数据类型说明：主要用于说明在整个设计中通用的数据类型，例如通用的地址总线数据类型定义等。
- 元件定义：主要规定在 VHDL 设计中参与元件例化的文件(已完成的设计实体)对外的接口界面。
- 子程序说明：并入程序包的子程序，有利于在设计中任一处方便地调用。

1. 程序包首

程序包首的说明部分可收集多个不同的 VHDL 设计所需的公共信息，其中包括数据类型说明、信号说明、子程序说明及元件说明等。定义程序包的一般语句结构如下：

```
PACKAGE    程序包名  IS     --程序包首
    {程序包首说明部分}
END  程序包名;
```

2. 程序包体

```
PACKAGE BODY    程序包名    IS   --程序包体
     {程序包体说明部分以及包体内容}
END  程序包名;
```

程序包体用于定义在程序包首中已定义的子程序的主体。程序包体说明部分的组成可以是 USE 语句(允许对其他程序包的调用)、子程序定义、子程序体、数据类型说明、子类型说明和常数说明等。

对于没有子程序说明的程序包体可以省去。一个完整的程序包中，程序包首名和程序包体名是同一个名字。程序包结构中，程序包体并非必须的。程序包首可以独立定义和使用。

【例 3-11】程序包首定义示例。

```
PACKAGE   EXAMPLE   IS                          --程序包首开始
TYPE   BYTE   IS    RANGE   0   TO   255;      --定义数据类型 BYTE
SUBTYPE NIBBLE IS BYTE RANGE 0 TO 15;           --定义子类型 NIBBLE
CONSTANT BYTE_FF: BYTE: =255;                   --定义常数 BYTE_FF
SIGNAL ADDEND: NIBBLE;                          --定义信号 ADDEND
COMPONENT BYTE_ADDER                            --定义元件
PORT(A, B: IN BYTE;
       C: OUT BYTE;
       OVERFLOW: OUT BOOLEAN);
END COMPONENT;                                  --元件定义结束
FUNCTION MY_FUNCTION(A：IN BYTE )               --定义函数
RETURN BYTE;                                    --函数的返回类型为 BYTE
END EXAMPLE;                                    --程序包首结束
```

如果要使用这个程序包中的所有定义，可用 USE 语句访问此程序包，例如：

```
LIBRARY WORK;                                   --此句可省去
USE WORK. EXAMPLE.ALL;
ENTITY…
ARCHITECTURE…
```

程序包首与程序包体的关系：程序包体并非必需，只有在程序包中要说明子程序时，程序包体才是必需的；程序包首可以独立定义和使用。

【例 3-12】在现行 WORK 库中定义程序包并立即使用。

```
PACKAGE   SEVEN   IS          --定义程序包
    SUBTYPE   SEGMENTS IS BIT_VECTOR(0 TO 6);
```

```
    TYPE BCD IS RANGE 0 TO 9;
END SEVEN;
```

上面仅是对程序包首的定义，程序包定义完之后对它进行编译处理，编译后会自动地放入 WORK 库(WORK 库是一个默认库，只要建立一个项目都会自动建立此库)，下面要用这个程序包，必须先调用 WORK 库，例如：

```
LIBRARY WORK;
USE WORK.SEVEN.ALL;          --调用 WORK 库 SEVEN 程序包中的所有数据类型，以便后面使用
ENTITY DECODER IS            --定义一个名为 DECODER(解码器)的实体
    PORT(INPUT：IN STD_LOGIC_VECTOR(3 DOWNTO 0);  --为自定义的 BCD 码数据类型
    DRIVE：OUT SEGMENTS); --为自定义的 SEGMENTS(含有 7 个元素的位矢量)数据类型
END DECODER;
ARCHITECTURE ART OF DECODER IS
BEGIN
WITH INPUT SELECT            --用后面要讲的选择信号赋值语句
DRIVE<=B"0111111" WHEN B"0000";
      B"0000110"WHEN B"0001";
B"1011011" WHEN B"0010";
      B"1001111" WHEN B"0011";
      B"1100110" WHEN B"0100";
      B"1101101" WHEN B"0101";
      B"1111101" WHEN B"0110";
      B"0000111" WHEN B"0111";
      B"1111111" WHEN B"1000";
      B"1101111" WHEN BV1001";
      B"0000000" WHEN   OTHERS;
END ARCHITECTURE ART;
```

此例是一个4位BCD码向七段译码显示码转换的 VHDL 描述。在系统中，常常需要将译码输出显示为十进制数字或其他符号，因此需要译码器能直接驱动数字显示器，或者能与显示器配合起来使用。在程序包 SEVEN 中定义了两个新的数据类型 SEGMENTS 和 BCD，DECODER 的实体描述中使用了这两个数据类型。这种类型的译码器称为显示译码器。七段显示译码器是最为常用的显示译码器，它可用于直接驱动七段数码管。

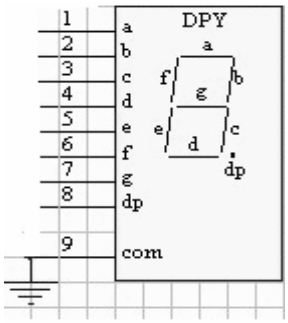


图 3-6　七段数码管显示的原理结构图

七段数码管显示的原理结构如图 3-6 所示。七段数码管有共阴极接地和共阳极接地两种接法。共阴极接地要求译码器输出高电平驱动数码管发亮，共阳极接地要求译码器输出低电平驱动数码管发亮。DRIVE(6)~DRIVE(0)分别对应数码管的 g~a 段。

3.4.2　预定义程序包

VHDL 为设计者提供两种类型的库，分别为设计库和资源库。其中，设计库主要包括 STD

库和 WORK 库；资源库包含 IEEE 库、VITAL 库和用户自定义库。

VHDL 语言标准中有很多预定义的程序包，如广泛使用的 STD 库中的 STANDARD 程序包以及 IEEE 库中的 STD_LOGIC_1164、STD_LOGIC_ARITH、STD_LOGIC_UNSIGNED 等。由于这些程序包广泛用于 VHDL 编程中，因此这里有必要对这些常用的程序包进行简单介绍。

1. STD_LOGIC_1164 程序包

它是 IEEE 库中最常用的程序包，是 IEEE 的标准程序包，预先在 IEEE 库中编译。其中包含了一些数据类型、子类型、函数和逻辑运算符的定义。该程序包中用得最多和最广的是定义了满足工业标准的两个数据类型 STD_LOGIC 和 STD_LOGIC_VECTOR，它们非常适用于 FPGA 器件中的多值逻辑设计结构。

2. STD_LOGIC_ARITH 程序包

它预先编译在 IEEE 库中，是 Synopsys 公司的程序包。此程序包在 STD_LOGIC_1164 程序包的基础上扩展了 3 个数据类型 UNSIGNED、SIGNED 和 SMALL_INT，并为其定义了相关的算术运算符、关系比较运算符和转换函数。

3. STD_LOGIC_UNSIGNED 和 STD_LOGIC_SIGNED 程序包

这两个程序包都是 Synopsys 公司的程序包，都预先编译在 IEEE 库中。这些程序包重载了可用于 INTEGER 型及 STD_LOGIC 和 STD_LOGIC_VECTOR 型混合运算的运算符，并定义了一个由 STD_LOGIC_VECTOR 型到 INTEGER 型的转换函数。这两个程序包的区别是，STD_LOGIC_SIGNED 中定义的运算符考虑到了符号，是有符号数的运算，而 STD_LOGIC_UNSIGNED 则正好相反。

程序包 STD_LOGIC_ARITH、STD_LOGIC_UNSIGNED 和 STD_LOGIC_SIGNED 虽然未成为 IEEE 标准，但已经成为事实上的工业标准，绝大多数的 VHDL 综合器和 VHDL 仿真器都支持它们。

4. STANDARD 和 TEXTIO 程序包

这两个程序包是 STD 库中的预编译程序包。STANDARD 程序包中定义了许多基本的数据类型、子类型和函数。它是 VHDL 标准程序包，实际应用中已隐性地打开了，故不必再用 USE 语句另作声明。TEXTIO 程序包定义了支持文本文件操作的许多类型和子程序。在使用本程序包之前，需加语句 USE STD.TEXTIO.ALL。

TEXTIO 程序包主要供仿真器使用。可以用文本编辑器建立一个数据文件，文件中包含仿真时需要的数据，然后仿真时用 TEXTIO 程序包中的子程序存取这些数据。综合器中此程序包被忽略。

3.5　配置(CONFIGURATION)

配置就是一个设计实体的多种实现方式，即从某一个实体的多种结构描述方式中选择特定的一个。

配置语句描述层与层之间的连接关系以及实体与结构体之间的连接关系。可综合的 VHDL 设计中，有可能使用多个结构体描述一个实体。首先，硬件电路设计中通常存在面积和速度在性能上的互换，因此一个复杂的设计，例如乘法器，可能用不同的结构体加以描述，它们在速度和面积的性能上各有侧重；其次，有时设计者需要调整电路特征以适应特定的应用场景，例如调整计数器的模式(递增或递减)，这也可以使用不同的结构体进行描述完成。在仿真和验证阶段，具有多个结构体形式的程序更加有应用的场合。

配置有两个作用：一是描述元件和设计实体的关系；二是描述设计实体和结构体之间的关系。VHDL 配置的使用十分灵活，而且语法格式和使用方法也相对复杂。所幸配置的作用在大多数可综合的 VHDL 设计中不是必须的。

配置语句的一般格式如下：

```
CONFIGURATION　配置名 OF　实体名 IS
FOR 选配结构体名
END FOR
END 配置名;
```

其中，配置名是该默认配置语句的唯一标志，实体名就是要配置的实体名称，选配结构体名就是用来组成设计实体的结构体名。配置有两种格式，此书只简单讨论其中的一种。

例 3-13 是一个配置的简单应用示例，即在一个描述与非门 nand 的设计实体中会有两个以不同的逻辑描述方式构成的结构体，用配置语句来为特定的结构体需求作配置指定。

【例 3-13】配置语句的应用示例。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY　nand　IS
  PORT(A: IN STD_LOGIC;
       B: IN STD_LOGIC;
       C: OUT STD_LOGIC);
END ENTITY nand;
ARCHITECTURE　art1 OF nand IS
  BEGIN
   C<=NOT (A AND B );
END ARCHITECTURE art1;
ARCHITECTURE art2 OF nand ISBEGIN
      C<= '1' WHEN (A='0') AND(B='0')　ELSE
'1' WHEN (A='0') AND(B='1')　ELSE
'1' WHEN (A='1') AND(B='0')　ELSE
'0' WHEN (A='1') AND(B='1')　ELSE
'0';
END ARCHITECTURE art2;
```

```
CONFIGURATION first OF nand IS
FOR art1
END FOR;
END first;
CONFIGURATION   second   OF nand IS
FOR art2
END FOR;
END second;
```

在本例中若指定配置名为 second，则为实体 nand 配置的结构体为 art2；若指定配置名为 first，则为实体 nand 配置的结构体为 art1。这两种结构的描述方式是不同的，但是有相同的逻辑功能。

3.6　VHDL 文字规则

VHDL 语言作为硬件描述语言，其基本语言要素为：VHDL 文字规则、三类数据对象、数据类型和预算操作符。接下来将详细介绍 VHDL 语言要素。

VHDL 除了具有类似于计算机高级语言所具备的一般文字规则外，还包含许多特有的文字规则和表达方式，在编程中需认真遵循。

3.6.1　关键字

关键字是指 VHDL 预定义的有特殊意义的词语，不可再拿来命名对象或者实体名和结构体名等，因此也叫保留字。例如前面遇到的关键字就有 library、use、entity、architecture、in、out、signal、and 等。完整的关键字表见附录。

关键字不区分大小写，如 STD_LOGIC、Std_Logic 以及 std_logic 等均等价。读者可按照自己的习惯使用大写、小写字母甚至大小写混合字母来书写关键字。

3.6.2　标识符

标识符用来定义常数、变量、信号、端口、子程序或参数的名字。

标识符的命名规则如下。

- 有效字符：26 个大小写英文字母、数字 0~9 以及下划线“_”。
- 标识符第一个字母必须以英文开头，且最后一个字符不能是下划线。
- 标识符中不能包含两个或者两个以上的连续下划线。
- 标识符的字符不区分大小写，但实际使用中仍建议使用完全相同的标识符表示同一个信号，以增强程序的可读性。
- 标识符允许包含图形符号(如回车符、换行符等)，也允许包含空格符。
- 标识符长度不限。

- 保留字不能用于作为标识符使用。

以下是几种标识符的实例。合法标识符如下：

MY_COUNTER, DECODER_1, FFT, Sig_N, NOT_ACK, State0

非法标识符如下：

_DECODER_1, 2FFT, SIG_#N, NO-ACK, ALL_RST_, data_ _BUS, RETURN, ENTITY

3.6.3　数字

VHDL 中的数字有整数类型(无小数点)和实数类型(有小数点)两种，其中，实数类型只能应用仿真，不可综合。

(1) 整数文字

默认格式是十进制，科学计数法表示时使用字母 e 或 E，如：

5，678，0，156E2(=15600)，45_234_287(=45234287)。

其中，数字间的下划线仅仅是为了提高文字的可读性，相当于一个空的间隔符，而没有其他的意义，因而不影响文字本身的数值。

(2) 实数

实数也都是十进制的数，但必须带有小数点，如：

23.34，2.0，44.99E-2(=0.4499)，88_67_551.23_909(=8867551.23909)，1.335，0.0

(3) 非十进制表示数字

应该用这样的格式来表示：进制#数字#E1，第一部分，用十进制数标明数制进位的基数；第二部分，数制隔离符号#；第三部分，表达的文字；第四部分，指数隔离符号#；第五部分，用十进制表示的指数部分，这一部分的数如果是 0 可以省去不写。例如：

10#170#　　--(十进制数表示，等于 170)

2#1111_1110#　--(二进制数表示，等于 254)

16#E#E1　　--(十六进制数表示，等于 2#11100000#，等于 224)或：(=14*16=224)

16#F.01#E+2　--十六进制数表示，(=(15+1/(16*16))*16*16 =3841.00)

(4) 负数

表示负数时直接在数字前加负号，如：-15。

(5) 赋值

同类型数据才可赋值；逻辑矢量赋值时，要求数据类型一致且数据位宽必须相同。

(6) 物理量

物理量文字(VHDL 综合器不接受此类文字)，如：

60s(60 秒)，100m(100 米)，kΩ(千欧姆)，177A(177 安培)

整数可综合实现；实数一般不可综合实现；物理量不可综合实现。

3.6.4　字符和字符串

字符和字符串是 VHDL 词法单元之一。字符是用单引号引起来的 ASCII 字符，可以是数值，也可以是符号或字母，如：‘R’，‘A’，‘*’，‘Z’。而字符串则是一维的字符数组，须放在双引号中。

VHDL 中有两种类型的字符串：文字字符串和数位字符串。

(1) 文字字符串：文字字符串是用双引号引起来的一串文字，如：

“ERROR”，“BOTH S AND Q EQUAL TO L”，“X”，“BB$CC”，“ZZZZZZZZ”，“XXXXXXXX”。

(2) 数位字符串：数位字符串也称位矢量，是预定义的数据类型 BIT 的一位数组，它们所代表的是二进制、八进制或十六进制的数组，其位矢量的长度即为等值的二进制数的位数。数位字符串的表示首先要有计算基数，然后将该基数表示的值放在双引号中，基数符以 B、O 和 X 表示，并放在字符串的前面。格式如下：

基数符号“数值”

其中基数符号有 3 种，它们的含义分别如下。

- B：二进制基数符号，表示二进制数位 0 或 1，在字符串中每一个位表示一个 BIT。
- O：八进制基数符号，在字符串中的第一个数代表一个八进制数，即代表一个 3 位(BIT)的二进制数。
- X：十六进制基数符号(0~F)，代表一个十六进制数，即代表一个 4 位的二进制数。

例如：

```
B"1_1101_1110"   --二进制数数组，长度是 9
O"34”            --八进制数数组，长度是 6
X"1AB"           --十六进制数数组，长度是 12
```

3.6.5　下标名及下标段名

下标名用于指示数组型变量或信号的某一元素，而下标段名则用于指示数组型变量或信号的某一段元素，表达式的数值必须在数组元素下标号范围以内，并且必须是可计算的。TO 表示数组下标序列由低到高，如“2 TO 8”；DOWNTO 表示数组下标序列由高到低，如“8 DOWNTO 2”。

如果表达式是一个可计算的值，则此操作数可很容易地进行综合。如果是不可计算的，则只能在特定的情况下综合，且耗费资源较大。其语句格式如下：

标识符(表达式)

下标段名的格式如下：

标识符 (表达式 to/downto 表达式)

如：

```
a：std_logic_vector(7 downto 0); y : std_logic;
a(7),a(6)…a(0)
a(7 downto o), a(7 downto 4), a(5 downto 3)…
y<=a(m);
```

3.6.6　注释

VHDL 中注释语句以“--”起始，且“--”的有效范围只在本行。因此，连续多行注释需在每行前加“--”符号。

为了程序的可读性，建议多写程序注释，且尽量习惯英文注释，因为有些编译器不兼容中文注释，如 Xilinx 的 ISE 10.1 和 Altera 的 Quartus II 9.0 等。

3.7　数据对象

在 VHDL 中，数据对象(Data Objects)类似于一种容器，它接受不同数据类型的赋值。数据对象有 3 类，即变量(VARIABLE)、常量(CONSTANT)和信号(SIGNAL)，前两种可以从传统的计算机高级语言中找到对应的数据类型，其语言行为与高级语言中的变量和常量十分相似，但信号这一数据对象比较特殊，它具有更多的硬件特征，这是 VHDL 中最有特色的语言要素之一。

从硬件电路系统来看，变量和信号相当于组合电路系统中门与门间的连线及其连线上的信号值，常量相当于电路中的恒定电平如 GND 或 VCC 接口；从行为仿真和 VHDL 语句功能上看，信号与变量具有比较明显的区别，主要表现在接受和保持信号的方式和信息保持与转递的区域大小上，例如，信号可以设置传输延迟量，而变量则不能，变量只能作为局部的信息载体，如只能在所定义的进程中有效，而信号则可作为模块间的信息载体；如在结构体中各进程间传递信息的变量设置，最后的信息传输和界面间的通信都靠信号来完成。综合后的VHDL 文件中信号将对应更多的硬件结构，但需注意的是，对于信号和变量的认识，单从行为仿真和语法要求的角度去认识是不完整的，事实上在许多情况下，综合后所对应的硬件电路结构中信号和变量并没有什么区别，例如在满足一定条件的进程中，综合后它们都能引入寄存器，其关键在于它们都具有能够接受赋值这一重要的共性，而 VHDL 综合器并不理会它们在接受赋值时存在的延时特性(只有 VHDL 行为仿真器才会考虑这一特性差异)。此外还应注意，尽管 VHDL 仿真器允许变量和信号设置初始值，但在实际应用中 VHDL 综合器并不会把这些信息综合进去，这是因为实际的 FPGA/CPLD 芯片在上电后并不能确保其初始状态的取向。因此对于时序仿真来说，设置的初始值在综合时是没有实际意义的。

3.7.1　变量(VARIABLE)

在 VHDL 语法规则中，变量是一个局部量，只能在进程和子程序中使用。变量不能将信息带出对它作出定义的当前设计单元。变量的赋值是一种理想化的数据传输，是立即发生的，不存在任何延时的行为。VHDL 语言规则不支持变量附加延时语句。变量常用在实现某种算法的赋值语句中。

定义变量的语法格式如下：

```
VARIABLE 变量名数据类型: = 初始值
```

例如以下变量定义语句：

```
VARIABLE a : INTEGER;
VARIABLE b, c : INTEGER := 2;
VARIABLE d : STD_LOGIC;
```

分别定义 a 为整数型变量；b 和 c 也为整数型变量，初始值为 2；d 为标准位变量。

3.7.2　信号(SIGNAL)

信号是描述硬件系统的基本数据对象，它类似于连接线信号，可以作为设计实体中并行语句模块间的信息交流通道(交流来自顺序语句结构中的信息)。在 VHDL 中，信号及其相关的信号赋值语句、决断函数、延时语句等很好地描述了硬件系统的许多基本特征，如硬件系统运行的并行性、信号传输过程中的惯性延迟特性、多驱动源的总线行为等。

信号作为一种数值容器，不但可以容纳当前值，也可以保持历史值。这一属性与触发器的记忆功能有很好的对应关系，因此，它又类似于 ABEL 语言中定义了 REG 的节点 NODE 的功能，只是不必注明信号上数据流动的方向。信号定义的语句格式与变量非常相似，信号定义也可以设置初始值。它的定义格式如下：

```
SIGNAL 信号名 数据类型 := 初始值;
```

同样，信号初始值的设置不是必需的，而且初始值仅在 VHDL 的行为仿真中有效。与变量相比，信号的硬件特征更为明显。它具有全局性特征，例如，在程序包中定义的信号，对于所有调用此程序包的设计实体都是可见的。可直接调用的在实体中定义的信号在其对应的结构体中都是可见的。

```
SIGNAL temp: STD_LOGIC: = 0;
SIGNAL flaga flagb: BIT;
SIGNAL data: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL a: INTEGER RANGE 0 TO 15;
```

此例中第一组定义了一个单值信号 temp，数据类型是标准位 STD_LOGIC，信号初始值为低电平；第二组定义了两个数据类型为位 BIT 的信号 flaga 和 flagb；第三组定义了一

个位矢量信号或者说是总线信号或数组信号数据类型，是标准位矢 STD_LOGIC_VECTOR，共有 16 个信号元素；最后一组定义信号 a 的数据类型是整数，变化范围是 0~15。

信号和变量的区别总结如表 3-3 所示。

表 3-3　信号和变量的区别

	信号(Signal)	变量(Variable)
赋值符号	<=	:=
定义位置	结构体中	进程或子程序中
功能	电路内部的连接或端口	内部数据运算
作用范围	全局，进程间通信	进程或子程序的内部
赋值行为	延迟赋值	立即赋值

3.7.3　常量(CONSTANT)

常数的定义和设置主要是为了使设计实体中的常数更容易阅读和修改。例如，将位矢的宽度定义为一个常量，只要修改这个常量就能很容易地改变宽度，从而改变硬件结构。在程序中，常量是一个恒定不变的值，一旦作了数据类型和赋值定义后，在程序中就不能再改变了，因而具有全局性意义。常量的定义形式与变量十分相似，其形式如下：

```
CONSTANT  常数名数据类型   :=   表达式;
```

例如：

```
CONSTANT fbus : BIT_VECTOR := "010115";       --位矢数据类型
CONSTANT Vcc : REAL := 5.0;                   --实数数据类型
CONSTANT dely : TIME := 25ns;                 --时间数据类型
```

VHDL 要求所定义的常量数据类型必须与表达式的数据类型一致。常量的数据类型可以是标量类型或复合类型，但不能是文件类型(file)或存取类型(Access)。常量定义语句所允许的设计单元有实体结构体程序包块进程和子程序。在程序包中定义的常量可以暂不设定具体数值，它可以在程序包体中设定。

综上可知，信号与变量是 VHDL 中常用的数据对象，它们的取值在程序中可以改变；常量的取值在程序中不可改变，常用于提高程序的可读性；文件用于VHDL 仿真程序。VHDL 4 种数据对象的定义和语法总结如表 3-4 所示。

表 3-4　数据对象总结

数 据 对 象	类 型 说 明
信号(signal)	结构体内部的节点，在结构体中声明定义，作用域为整个结构体，赋值具有非“立即性”
变量(variable)	用于算法描述，在进程或子程序定义，作用范围为进程或子程序内，赋值具有“立即性”
常量(constant)	用于定义程序中多次出现的常数数值，由于提高程序的可读性，以及保持数据的一致性
文件(file)	用于程序仿真，在 VHDL 仿真中常用来读取数据流较大的数据源和保存仿真结果

3.8　数据类型

数据类型定义了一个取值集合，以及针对这些取值所允许的操作。这个定义与高级语言中数据类型的定义类似。不同的是，VHDL 是一种强类型语言，主要表现在：

- VHDL 中的每一个信号、变量、常数、文件等数据对象都有一个确定的数据类型与之对应，且只能拥有这个数据类型的取值；
- 对某个数据对象进行的操作类型必须与该对象的数据类型匹配；
- 只有相同数据类型的数据对象才能相互赋值，不同数据类型的数据不能直接代入，即使数据类型相同而位长不同也不能代入。

这种强类型语言的特性限制了 VHDL 语言的灵活性，但在另外一方面使设计者在设计初期就能检查出程序存在的错误，提高了程序设计的效率。为了更好地应用 VHDL 进行硬件描述，必须很好地理解各种数据类型的定义和用法。本节首先介绍 VHDL 中预定义的数据类型，因为理解和掌握这些数据类型的使用方法是理解和书写 VHDL 程序的基础；接着介绍 VHDL 中用户自定义的数据类型；最后介绍数据类型转换。

3.8.1　VHDL 预定义数据类型

预定义的 VHDL 数据类型是 VHDL 最常用、最基本的数据类型。这些数据类型都已在 VHDL 的标准程序包 STANDARD 和 STD_LOGIC_1164 及其他标准程序包中作了定义，并可在设计中随时调用。

- STD 库 STANDARD 程序包：预定了 bit、boolean、integer、real、time 等数据类型；
- IEEE 库 stD_LOGIC_1164 程序包：预定义了 std_logic、STD_logic_vector 等数据类型；
- IEEE 库 stD_LOGIC_arith 程序包：预定义了 signedc、unsigned 数据类型，以及一些数据类型转换函数；
- IEEE 库 stD_LOGIC_signed/stD_LOGIC_unsigned 程序包：预定义了 signedc、unsigned 数据类型，并定义了一些函数，使得 STD_logic_vector 数据类型能进行针对 signedc 或 unsigned 数据类型的操作。

VHDL 中与定义的数据类型很多，但实际编程常用的并不多。按照所在库来划分，可将 VHDL 中预定义的常用的数据类型归纳见附录。其中，STD_LOGIC、STD_LOGIC_VECTOR 最常用，INTEGER 次之。

1. STD 库 STANDARD 包的预定义数据类型

(1) 布尔(BOOLEAN)数据类型

程序包 STANDARD 中定义布尔数据类型的源代码如下：

```
TYPE BOOLEAN IS(FALES, TRUE);
```

布尔数据类型实际上是一个二值枚举型数据类型，它的取值有 FALSE 和 TRUE 两种，常用与逻辑函数。如相等(=)，比较(<)中作逻辑比较；综合器将用一个二进制位表示 BOOLEAN 型变量信号或。

例如，当 A 大于 B 时，在 IF 语句中的关系运算表达式(A>B)的结果是布尔量 TRUE，反之为 FALSE。综合器将其变为 1 或 0 信号值，对应于硬件系统中的一根线。

如，BIT 值转化成 BOOLEAN：

```
BOOLEAN_VAR: =(BIT_VAR= '1 ');
```

(2) 位(BIT)数据类型

位数据类型也属于枚举型，放在单引号中，取值只能是'1'或'0'。位数据类型的数据对象，如变量、信号等，可以参与逻辑运算，运算结果仍是位的数据类型。VHDL 综合器用一个二进制位表示 BIT。在程序包 STANDARD 中定义的源代码如下：

```
TYPE BIT IS ('0', '1');
```

(3) 位矢量(BIT_VECTOR)数据类型

位矢量只是基于 BIT 数据类型的数组，是用双引号括起来的一组位数据。如："001100"，X"00B10B"。在程序包 STANDARD 中定义的源代码如下：

```
TYPE BIT_VECTOR IS ARRAY(NATURAL RANGE<>)OF BIT;
```

使用位矢量必须注明位宽，即数组中的元素个数和排列，例如：

```
SIGNAL A: BIT_VECTOR(7 TO 0);
```

信号 A 被定义为一个具有 8 位位宽的矢量，它的最左位是 A(7)，最右位是 A(0)。

(4) 字符(CHARACTER)数据类型

字符类型通常用单引号引起来，如'A'。字符类型区分大小写，如'B'不同于'b'。字符类型已在 STANDARD 程序包中作了定义。

```
VARIABLE CHARACTER_VAR：CHARACTER;
    ……
CHARACTER_VAR：='A';
```

(5) 整数(INTEGER)数据类型

整数类型的数代表正整数、负整数和零。在 VHDL 中，整数的取值范围是-21 473 647~+21 473 647，即-(2^{31}-1)~+(2^{31}-1)，可用 32 位有符号的二进制数表示。在实际应用中，VHDL 仿真器通常将 INTEGER 类型作为有符号数处理，而 VHDL 综合器则将 INTEGER 作为无符号数处理。在使用整数时，VHDL 综合器要求用 RANGE 子句为所定义的数限定范围，然后根据所限定的范围来决定表示此信号或变量的二进制数的位数，因为 VHDL 综合器无法综合未限定的整数类型的信号或变量。

如语句“SIGNAL S: INTEGER RANGE 0 TO 15; ”规定整数 S 的取值范围是 0~15 共 16 个值，可用 4 位二进制数来表示，因此，S 将被综合成由 4 条信号线构成的信号。

整数常量的书写方式示例如下：

```
2                    --十进制整数
10E4                 --十进制整数
16#D2#               --十六进制整数
2#11011010#          --二进制整数
```

(6) 自然数(NATURAL)和正整数(POSITIVE)数据类型

自然数是整数的一个子类型，非负的整数，即零和正整数；正整数也是整数的一个子类型，它包括整数中非零和非负的数值。它们在 STANDARD 程序包中定义的源代码如下：

```
SUBTYPE NATURAL   IS   INTEGER RANGE 0   TO   INTEGER'HIGH;
SUBTYPE POSITIVE IS   INTEGER RANGE 1   TO   INTEGER'HIGH;
```

(7) 实数(REAL)数据类型

VHDL 的实数类型类似于数学上的实数，或称浮点数。实数的取值范围为-1.0E38~+1.0E38。通常情况下，实数类型仅能在 VHDL 仿真器中使用，VHDL 综合器不支持实数，因为实数类型的实现相当复杂，目前在电路规模上难以承受。

实数常量的书写方式举例如下：

```
65971.333333         --十进制浮点数
8#43.6#E+4           --八进制浮点数
43.6E-4              --十进制浮点数
```

(8) 字符串(STRING)数据类型

字符串数据类型是字符数据类型的一个非约束型数组，或称为字符串数组。字符串必须用双引号括起来。如：

```
VARIABLE STRING_VAR: STRING(1 TO 7);
…STRING_VAR: "Rosebud";
```

(9) 时间(TIME)数据类型

VHDL 中唯一的预定义物理类型是时间。完整的时间类型包括整数和物理量单位两部分，整数和单位之间至少留一个空格，如 55 ms，20 ns。

STANDARD 程序包中也定义了时间。定义如下：

```
TYPE TIME IS RANGE -2147483647 TO 2147483647
units
    fs;                      --飞秒，VHDL 中的最小时间单位
    ps = 1000 fs;            --皮秒
    ns = 1000 ps;            --纳秒
    us = 1000 ns;            --微秒
    ms = 1000 us;            --毫秒
    sec = 1000 ms;           --秒
    min = 60 sec;            --分
    hr = 60 min;             --时
end untis;
```

(10) 错误等级(SEVERITY_LEVEL)

在 VHDL 仿真器中，错误等级用来指示设计系统的工作状态。共有 4 种可能的状态值：NOTE(注意)、WARNING(警告)、ERROR(出错)、FAILURE(失败)。在仿真过程中，可输出这 4 种值来提示被仿真系统当前的工作情况。其定义如下：

```
TYPE SEVERITY_LEVEL IS (NOTE, WARNING, ERROR, FAILURE);
```

2. IEEE 预定义标准逻辑位与矢量

在 IEEE 库的程序包 STD_LOGIC_1164 中，定义了两个非常重要的数据类型，即标准逻辑位 STD_LOGIC 和标准逻辑矢量 STD_LOGIC_VECTOR。

(1) 标准逻辑位 STD_LOGIC_1164 中的数据类型

以下是定义在 IEEE 库程序包 STD_LOGIC_1164 中的数据类型。STD_LOGIC 程序包定义为九值逻辑系统，如下所示：

```
TYPE STD_LOGIC IS ('U', 'X', '0 ', '1', 'Z', 'W', 'L', 'H', '-');
```

各值的含义是：'U'-未初始化的，'X'-强未知的，'0 '-强 0，'1'-强 1，'Z'-高阻态，'W'-弱未知的，'L'-弱 0，'H'-弱 1，'-'-忽略。

在程序中使用此数据类型前，需加入下面的语句：

```
LIBRARY IEEE;
```

USE IEEE.STD_LOGIC_1164.ALL；由定义可见，STD_LOGIC 是标准的 BIT 数据类型的扩展，共定义了 9 种值，这意味着，对于定义为数据类型是标准逻辑位 STD_LOGIC 的数据对象，其可能的取值已非传统的 BIT 那样只有 0 和 1 两种取值，而是如上定义的那样有 9 种可能的取值。目前在设计中一般只使用 IEEE 的 STD_LOGIC 标准逻辑的位数据类型，可以完成电子系统的精确模拟，并可实现常见的三态总线电路，BIT 型则很少使用。

由于标准逻辑位数据类型的多值性，在编程时应当特别注意。因为在条件语句中，如果未考虑到 STD_LOGIC 的所有可能的取值情况，综合器可能会插入不希望的锁存器。

程序包 STD_LOGIC_1164 中还定义了 STD_LOGIC 型逻辑运算符 AND、NAND、OR、NOR、XOR 和 NOT 的重载函数，以及两个转换函数，用于 BIT 与 STD_LOGIC 的相互转换。

在仿真和综合中，STD_LOGIC 值是非常重要的，它可以使设计者精确模拟一些未知和高阻态的线路情况。对于综合器，高阻态和“-”忽略态可用于三态的描述。但就综合而言，STD_LOGIC 型数据能够在数字器件中实现的只有其中的 4 种值，即-、0、1 和 Z。当然，这并不表明其余的 5 种值不存在。这 9 种值对于 VHDL 的行为仿真都有重要意义。

(2) 标准逻辑矢量(STD_LOGIC_VECTOR)数据类型

由 STD_LOGIC_VECTOR 构成的数组，定义如下：

```
TYPE STD_LOGIC_VECTOR IS ARRAY (NATURAL RANGE<>) OF STD_LOGIC;
```

显然，STD_LOGIC_VECTOR 是定义在 STD_LOGIC_1164 程序包中的标准一维数组，

数组中的每一个元素的数据类型都是以上定义的标准逻辑位 STD_LOGIC。

STD_LOGIC_VECTOR 数据类型的数据对象赋值的原则是：相同位宽、相同数据类型的矢量间才能进行赋值。下例描述的是 CPU 中数据总线上位矢赋值的操作示意情况，注意其中信号数据类型定义和赋值操作中信号的数组位宽。

【例 3-14】信号数据类型的定义和赋值操作。

```
TYPE T_DATA IS ARRAY(7 DOWNTO 0) OF STD_LOGIC;                    --自定义数组类型
SIGNAL DATABUS, MEMORY: T_DATA; --定义信号 DATABUS, MEMORY
CPU: PROCESS                                                      --CPU 工作进程开始
VARIABLE REG1: T_DATA                                             --定义寄存器变量 REG1
BEGIN
DATABUS<=REG1;                                                    --向 8 位数据总线赋值
END PROCESS CPU;                                                  --CPU 工作进程结束
MEM: PROCESS                                                      --RAM 工作进程开始
BEGIN
DATABUS<=MEMORY;
END PROCESS MEM;
```

3. 其他预定义标准数据类型

VHDL 综合工具配带的扩展程序包中定义了一些有用的类型，如 Synopsys 公司在 IEEE 库中加入的程序包 STD_LOGIC_ARITH 中定义了如下数据类型：无符号型(UNSIGNED)、有符号型(SIGNED)、小整型(SMALL _INT)。

在程序包 STD_LOGIC_ARITH 中的类型定义如下：

```
TYPE UNSIGNED IS ARRAY (NATURAL RANGE <>) OF STD_LOGIC;
TYPE SIGNED IS ARRAY (NATURAL RANGE<>) OF STD_LOGIC;
SUBTYPE SMALL_INT IS INTEGER RANGE 0 TO 1;
```

如果将信号或变量定义为这几个数据类型，就可以使用本程序包中定义的运算符。在使用之前，必须加入下面的语句：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_ARITH.ALL;
```

UNSIGNED 类型和 SIGNED 类型是用来设计可综合的数学运算程序的重要类型。UNSIGNED 用于无符号数的运算，SIGNED 用于有符号数的运算。在实际应用中，大多数运算都需要用到它们。

在 IEEE 程序包中，UNMERIC_STD 和 NUMERIC_BIT 程序包中也定义了 UNSIGNED 型及 SIGNED 型。NUMERIC_STD 是针对于 STD_LOGIC 型定义的，而 NUMERIC_BIT 是针对于 BIT 型定义的。在程序包中还定义了相应的运算符重载函数。有些综合器没有附带 STD_LOGIC_ARITH 程序包，此时只能使用 NUMBER_STD 和 NUMERIC_BIT 程序包。

在 STANDARD 程序包中没有定义 STD_LOGIC_VECTOR 的运算符，而整数类型一般只在仿真的时候用来描述算法，或作数组下标运算，因此 UNSIGNED 和 SIGNED 的使用率是很高的。

(1) 无符号数据类型(UNSIGNED TYPE)

UNSIGNED 数据类型代表一个无符号的数值，在综合器中，这个数值被解释为一个二进制数，这个二进制数的最左位是其最高位。例如，十进制的 8 可以表示为 UNSIGNED ("1000")，如果要定义一个变量或信号的数据类型为 UNSIGNED，则其位矢长度越长，所能代表的数值就越大。如一个 4 位变量的最大值为 15，一个 8 位变量的最大值则为 255，0 是其最小值，不能用 UNSIGNED 定义负数。

以下是两则无符号数据定义的示例：

```
VARIABLE VAR: UNSIGNED(0 TO 10);          --VAR(0)是最高位(左边元素为最高位)
SIGNAL   SIG: UNSIGNED(5 DOWNTO 0);       --SIG(5)是最高位
```

其中，变量 VAR 有 11 位数值，最高位是 VAR(0)，而非 VAR(10)；信号 SIG 有 6 位数值，最高位是 SIG(5)。

(2) 有符号数据类型(SIGNED TYPE)

SIGNED 数据类型表示一个有符号的数值，综合器将其解释为补码，此数的最高位是符号位，例如：SIGNED("0101") 代表+5，5；SIGNED("1011") 代表-5。

若将上例的 VAR 定义为 SIGNED 数据类型，则数值意义就不同了，如：

```
VARIABLE VAR: SIGNED(0 TO 10);
```

其中，变量 VAR 有 11 位，最左位 VAR(0)是符号位。

3.8.2　用户自定义数据类型

VHDL 允许用户自行定义新的数据类型。可用于自定义的常用数据类型有：枚举类型(ENUMERATION TYPE)、整数类型(INTEGER TYPE)、数组类型(ARRAY TYPE)、记录类型(RECORD TYPE)、子类型、时间类型(TIME TYPE)、实数类型(REAL TYPE)等。

用户自定义数据类型是用类型定义语句 TYPE 和子类型定义语句 SUBTYPE 实现的，以下将介绍这两种语句的使用方法。

(1) TYPE 语句的用法

TYPE 语句的语法格式如下：

```
TYPE  数据类型名   IS   数据类型定义  [OF   基本数据类型];
```

其中，数据类型名由设计者自定义；数据类型定义部分用来描述所定义的数据类型的表达方式和表达内容；关键词 OF 后的基本数据类型是指数据类型定义中所定义的元素的基本数据类型，一般都是取已有的预定义数据类型，如 BIT、STD_LOGIC 或 INTEGER 等。

例如，以下是两种不同的定义方式：

```
TYPE ST1 IS ARRAY(0 TO 15)OF STD_LOGIC;
TYPE WEEK IS (SUN，MON，TUE，WED，THU，FRI，SAT);
```

第一句定义的数据 ST1 是一个具有 16 个元素的数组型数据类型，数组中的每一个元

素的数据类型都是 STD_LOGIC 型；第二句所定义的数据类型是由一组文字表示的，而其中的每一文字都代表一个具体的数值，如可令 SUN=“1010”。

在 VHDL 中，任一数据对象(SIGNAL、VARIABLE、CONSTANT)都必须归属某一数据类型，只有同数据类型的数据对象才能进行相互作用。利用 TYPE 语句可以完成各种形式的自定义数据类型，以供不同类型的数据对象间的相互作用和计算。

(2) SUBTYPE 语句，用法

子类型 SUBTYPE 只是由 TYPE 所定义的原数据类型的一个子集，它满足原数据类型的所有约束条件，原数据类型称为基本数据类型。子类型 SUBTYPE 的语句格式如下：

```
SUBTYPE　子类型名　IS　基本数据类型　RANGE　约束范围;
```

子类型的定义只在基本数据类型上作一些约束，并没有定义新的数据类型。子类型定义中的基本数据类型必须在前面已通过 TYPE 定义。例如：

```
SUBTYPE　DIGITS　INTEGER　RANGE　0　TO　9;
```

其中，INTEGER 是标准程序包中已定义过的数据类型，子类型 DIGITS 只是把 INTEGER 约束到只含 10 个值的数据类型上。

由于子类型与其基本数据类型属同一数据类型，因此属于子类型的和属于基本数据类型的数据对象间的赋值和被赋值可以直接进行，不必进行数据类型的转换。

利用子类型定义数据对象的好处是，除了可使程序提高可读性和易处理性外，其最大的好处在于利于提高综合的优化效率，这是因为综合器可以根据子类型所设的约束范围，有效地推知参与综合的寄存器的最合适的数目。

1. 枚举类型

VHDL 中的枚举数据类型是用文字符号来表示一组实际的二进制数的类型(若直接用数值来定义，则必须使用单引号)。例如，状态机的每一状态在实际电路虽是以一组触发器的当前二进制数位的组合来表示的，但设计者在状态机的设计中，为了更便于阅读和编译，往往将表征每一状态的二进制数组用文字符号来代表。

【例 3-15】枚举数据类型示例。

```
TYPE M_STATE IS( STATE1，STATE2，STATE3，STATE4，STATE5);
SIGNAL CURRENT_STATE，NEXT_STATE：M_STATE;
```

在这里，信号 CURRENT_STATE 和 NEXT_STATE 的数据类型定义为 M_STATE，它们的取值范围是可枚举的，即从 STATE1~STATE5 共 5 种，而这些状态代表 5 组唯一的二进制数值。

在综合过程中，枚举类型文字元素的编码通常是自动的，编码顺序是默认的，一般将第一个枚举量(最左边的量)编码为 0，以后的依次加 1。综合器在编码过程中自动将第一枚举元素转变成位矢量，位矢的长度将取所需表达的所有枚举元素的最小值。如上例中用于表达 5 个状态的位矢长度应该为 3，编码默认值为如下方式：

```
STATE1='000'; STATE2='001'; STATE3='010'; STATE4='011'; STATE5='100';
```

于是它们的数值顺序便成为 STATE1<STATE2<STATE3<STATE4<STATE5。一般而言，编码方法因综合器不同而不同。为了某些特殊的需要，编码顺序也可以人为设置。

2. 数组类型

数组类型属复合类型，是将一组具有相同数据类型的元素集合在一起，作为一个数据对象来处理的数据类型。数组可以是一维(每个元素只有一个下标)数组或多维数组(每个元素有多个下标)。VHDL 仿真器支持多维数组，但 VHDL 综合器只支持一维数组。

数组的元素可以是任何一种数据类型，用以定义数组元素的下标范围子句决定了数组中元素的个数，以及元素的排序方向，即下标数是由低到高，或是由高到低。如子句“0 TO 7”是由低到高排序的 8 个元素；“15 DOWNTO 0”是由高到低排序的 16 个元素。

VHDL 允许定义两种不同类型的数组，即限定性数组和非限定性数组。它们的区别是，限定性数组下标的取值范围在数组定义时就被确定了，而非限定性数组下标的取值范围需留待随后根据具体数据对象再确定。

定义限定性数组的语句格式如下：

```
TYPE  数组名  IS ARRAY (数组范围) OF 数据类型;
```

其中，数组名是新定义的限定性数组类型的名称，可以是任何标识符，其类型与数组元素相同；数组范围明确指出数组元素的定义数量和排序方式，以整数来表示其数组的下标；数据类型即指数组各元素的数据类型。

【例 3-16】限定性数组类型定义示例。

```
TYPE STB IS ARRAY(7 DOWNTO 0) OF STD_LOGIC;
```

这个数组类型的名称是 STB，它有 8 个元素，它的下标排序是 7，6，5，4，3，2，1，0，各元素的排序是 STB(7)，STB(6)，…，STB(1)，STB(0)。每一元素的数据类型是 STD_LOGIC。

定义非限制性数组的语句格式如下：

```
TYPE  数组名  IS ARRAY (数组下标名  RANGE<>) OF  数据类型;
```

其中，数组名是定义的非限制性数组类型的取名；数组下标名是以整数类型设定的一个数组下标名称；符号“< >”是下标范围待定符号，用到该数组类型时，再填入具体的数值范围；数据类型是数组中每一元素的数据类型。

以下三例表达了非限制性数组类型的不同用法。

【例 3-17】非限制性数组的定义示例。

```
TYPE BIT_VECTOR IS ARRAY(NATURAL RANGE<>) OF BIT;
VARABLE VA：BIT_VECTOR(1 TO 6);        --将数组取值范围定在 1~6
```

【例 3-18】非限制性数组的另一种定义示例。

```
TYPE REAL_MATRIX IS ARRAY (POSITIVE RANGE<>) OF REAL;
```

```
VARIABLE REAL_MATRIX_OBJECT: REAL_MATRIX(1 TO 8);     --限定范围
```

【例 3-19】非限制性数组的范围限定设置。

```
TYPE LOGIC_VECTOR IS ARRAY(NATURAL RANGE<>，POSITIVE RANGE<>) OF LOG_4;
VARIABLEL4_OBJECT: LOG_4_VECTOR(0 TO 7，1 TO 2);          --限定范围
```

3. 记录类型

由已定义的、数据类型不同的对象元素构成的数组称为记录类型的对象。定义记录类型的语句格式如下：

```
TYPE  记录类型名   IS   RECORD
元素名:  元素数据类型;
元素名:  元素数据类型;
…
END RECORD [记录类型名];
```

【例 3-20】记录类型定义示例。

```
TYPE RECDATA IS RECORD              --将 RECDATA 定义为 3 个元素的记录类型
ELEMENT1: TIME;                     --将元素 ELEMENT1 定义为时间类型
ELEMENT2: TIME;                     --将元素 ELEMENT2 定义为时间类型
ELEMENT3: STD_LOGIC;                --将元素 ELEMENT3 定义为标准位类型
END RECORD;
```

对于记录类型的数据对象赋值的方式，可以是整体赋值，也可以是对其中的单个元素进行赋值。在使用整体赋值方式时，可以有位置关联方式或名字关联方式两种表达方式。如果使用位置关联，则默认为元素赋值的顺序与记录类型声明时的顺序相同。如果使用了 OTHERS 选项，则至少应有一个元素被赋值，如果有两个或更多的元素由 OTHERS 选项来赋值，则这些元素必须具有相同的类型。此外，如果有两个或两个以上的元素具有相同的子类型，就可以以记录类型的方式放在一起定义。

【例 3-21】利用记录类型定义的一个微处理器命令信息表。

```
TYPE REGNAME IS (AX，BX，CX，DX);
TYPE OPERATION IS RECORD
   OPSTR：STRING(1 TO 10);
   OPCODE：BIT_VECTOR(3 DOWNTO 0);
   OP1，OP2，RES：REGNAME;
END RECORD;
VARIABLE   INSTR1，INSTR2：OPERATION;…
INSTR1: =("ADD AX，BX"，"0001"，AX，BX，AX);
INSTR2: =("ADD AX，BX"，"0010"，OTHERS=>BX);
VARIABLE INSTR3：OPERATION;…
INSTR3.OPSTR: ="MUL AX，BX";
INSTR3.OP1: =AX;
```

本例中，定义的记录 OPERATION 共有 5 个元素，一个是加法指令码的字符串 OPSTR，一个是 4 位操作码 OPCODE，另外 3 个是枚举型数据 OP1、OP2、RES(其中 OP1 和 OP2

是操作数，RES 是目标码)。例中定义的变量 INSTR1 的数据类型是记录型 OPERATION，它的第一个元素是加法指令字符串"ADD AX，BX"；第二个元素是此指令的 4 位命令代码“0001”；第三、第四个元素为 AX 和 BX；AX 和 BX 相加后的结果送入第五个元素 AX，因此这里的 AX 是目标码。语句 INSTR3. OPSTR：="MUL AX，BX"赋给 INSTR3 中的元素 OPSTR。一般地，对于记录类型的数据对象进行单元素赋值时，就在记录类型对象名后加点(.)，再加赋值元素的元素名。

记录类型中的每一个元素仅为标量型数据类型构成称为线性记录类型；否则为非线性记录类型。只有线性记录类型的数据对象才是可综合的。

3.8.3　数据类型转换

由于 VHDL 是一种强类型语言，不允许不同类型的数据相互赋值，而有时又必须在不同的数据间赋值，这时就必须对数据类型进行转换。数据类型的转换有以下两种方法。

(1) 类型标记法

这种转换方法由数据类型名称来实现关系密切的标量类型之间的转换，即直接类型转换。一般语句格式是：

```
数据类型标识符(表达式)
```

一般情况下，直接类型转换仅限于非常关联的数据类型之间(数据类型相互间的关联性非常大)，必须遵循以下规则：

- 所有的抽象数字类型是非常关联的类型，如整型、浮点型，如果浮点数转换为整数，则转换结果是最接近的一个整型数。
- 如果两个数组有相同的维数，两个数组的元素是同一类型，并且在各处的下标范围内索引是同一类型或非常接近的类型，那么这两个数组是非常关联类型。
- 枚举型不能被转换。

【例 3-22】使用类型标记实现类型转换。

```
Variable   x:   integer;
Variable   y:   real;
```

使用类型标记实现类型转换时，可采用以下赋值语句：

```
x := integer(y);
y := real(x);
```

(2) 类型转换函数法

VHDL 语言常用程序包中提供了多种数据类型转换函数，使得某些数据类型间可以相互转换，以实现正确的赋值操作。VHDL 中预定义的数据类型转换函数大多可在 std_logic_arith 程序包中找到。如表 3-5 所示为常用的数据类型转换函数。

表 3-5　常用的数据类型转换函数

函　数　名	所在程序包	函数功能说明
conv_integer(p)	Std_logic_arith	将 unsigned、signed 类型转换为 integer 类型
conv_unsigned(p,b)	Std_logic_arith	将 unsigned、signed、integer 转换为 b，即 bit 的 unsigned 类型
conv_signed(p,b)	Std_logic_arith	将 unsigned、signed、integer 转换为 b，即 bit 的 signed 类型
conv_std_logic_vector(p,b)	Std_logic_arith	将 unsigned、signed、integer 转换为 b，即 bit 的 std_logic_vector 类型
to_bit(p)	Std_logic_1164	将 std_logic 类型转换为 bit 类型
to_bit_vector(p)	Std_logic_1164	将 std_logic_vector 类型转换为 bit_vector 类型
to_std_logic(p)	Std_logic_1164	将 bit 类型转换为 std_logic 类型
to_std_logic_vector(p)	Std_logic_1164	将 bit_vector 类型转换为 std_logic_vector 类型

【例 3-23】数据类型转换示例。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY CNT4 IS
      PORT(CLK: IN STD_LOGIC;
           P: INOUT STD_LOGIC_VECTOR(3 DOWNTO 0);
END CNT4;
LIBRARY DATAIO;
USE DATAIO.STD_LOGIC_OPS.ALL
ARCHITECTURE BEHV OF CNT4 IS
BEGIN
PROCESS(CLK)
BEGIN
IF CLK= '1'AND CLK'EVENT THEN
         P<=TO_VECTOR(2，TO_INTEGER(P)+1);
END IF
END PROCESS;
END BEHV;
```

此例中利用了 DATAIO 库的程序包 STD_LOGIC_OPS 中的两个数据类型转换函数：TO_VECTOR 和 TO_INTEGER，前者将 INTEGER 转换成 STD_LOGIC_VECTOR，后者将 STD_LOGIC_VECTOR 转换成 INTEGER。通过这两个转换函数，就可以使用“+”运算符进行直接加 1 操作了，同时又能保证最后的加法结果是 STD_LOGIC_VECTOR 数据类型。

利用类型转换函数来进行类型的转换需定义一个函数，使其参数类型为被转换的类型，返回值为转换后的类型。这样就可以自由地进行类型转换了。在实际应用中，类型转换函数是很常用的。VHDL 的标准程序包中提供了一些常用的转换函数。

3.9　运算操作符

VHDL 的各种表达式由操作数和操作符组成，其中操作数是各种运算的对象，而操作符则规定运算(算术运算和逻辑运算)的方式。

1. 操作符种类及对应的操作数类型

在 VHDL 中，有四类操作符，即逻辑操作符(Logical Operator)、关系操作符(Relational Operator)和算术操作符(Arithmetic Operator)，此外还有重载操作符(Overloading Operator)。前三类操作符是完成逻辑和算术运算的最基本的操作符的单元，重载操作符是对基本操作符作了重新定义的函数型操作符。各种操作符所要求的操作数类型如表 3-6 所示，操作符之间的优先级别如表 3-7 所示。

表 3-6　VHDL 操作符列表

类　型	操　作　符	功　能	操作数数据类型
算术操作符	+	加	整数
	–	减	整数
	&	并置	一维数组
	*	乘	整数和实数(包括浮点数)
	/	除	整数和实数(包括浮点数)
	MOD	取模	整数
	REM	取余	整数
	SLL	逻辑左移	BIT 或布尔型一维数组
	SRL	逻辑右移	BIT 或布尔型一维数组
	SLA	算术左移	BIT 或布尔型一维数组
	SRA	算术右移	BIT 或布尔型一维数组
	ROL	逻辑循环左移	BIT 或布尔型一维数组
	ROR	逻辑循环右移	BIT 或布尔型一维数组
	**	乘方	整数
	ABS	取绝对值	整数
	+	正	整数
	–	负	整数
关系操作符	=	等于	任何数据类型
	/=	不等于	任何数据类型
	<	小于	枚举与整数类型，及对应的一维数组
	>	大于	枚举与整数类型，及对应的一维数组
	<=	小于等于	枚举与整数类型，及对应的一维数组
	>=	大于等于	枚举与整数类型，及对应的一维数组

(续表)

类　型	操　作　符	功　能	操作数数据类型
逻辑操作符	AND	与	BIT，BOOLEAN，STD_LOGIC
	OR	或	BIT，BOOLEAN，STD_LOGIC
	NAND	与非	BIT，BOOLEAN，STD_LOGIC
	NOR	或非	BIT，BOOLEAN，STD_LOGIC
	XOR	异或	BIT，BOOLEAN，STD_LOGIC
	XNOR	异或非	BIT，BOOLEAN，STD_LOGIC
	NOT	非	BIT，BOOLEAN，STD_LOGIC

表 3-7　VHDL 操作符优先级

运　算　符	优　先　级
NOT， ABS， **	最高优先级
*， /， MOD， REM	
+ (正号)， - (负号)	⇧
+， -， &	
SLL， SLA， SRL， SRA， ROL， ROR	最低优先级
=， /=， <， <=， >， >=	
AND， OR， NAND， NOR， XOR， XNOR	

运算符操作应注意以下几点：

(1) 严格遵循在基本操作符间操作数是同数据类型的规则；严格遵循操作数的数据类型必须与操作符所要求的数据类型完全一致的规则。

(2) 注意操作符之间的优先级别。当一个表达式中有两个以上的操作符时，可使用括号将这些运算分组。

(3) VHDL 共有 7 种基本逻辑操作符，对于数组型如 STD_LOGIC_VECTOR 数据对象的相互作用是按位进行的。一般情况下，经综合器综合后，逻辑操作符将直接生成门电路。信号或变量在这些操作符的直接作用下，可构成组合电路。

(4) 关系操作符的作用是将相同数据类型的数据对象进行数值比较(=、/=)或关系排序判断(<、<=、>、>=)，并将结果以布尔类型(BOOTLEAN)的数据表示出来，即TRUE 或 FALSE 两种。对于数组或记录类型的操作数，VHDL 编译器将逐位比较对应位置各位数值的大小而进行比较或关系排序。

(5) 在表 3-6 中所列出的 17 种算术操作符可以分为求和操作符、求积操作符、符号操作符、混合操作符和移位操作符五类。

- 求和操作符包括加减操作符和并置操作符。加减操作符的运算规则与常规的加减法是一致的，VHDL 规定它们的操作数的数据类型是整数。对于大于位宽为 4 的加法器和减法器，VHDL 综合器将调用库元件进行综合。

并置运算符(&)的操作数的数据类型是一维数组，可以利用并置符将普通操作数或数组组合起来形成各种新的数组。例如“VH”&“DL”的结果为“VHDL”；“0”&“1”的结果为“01”，并置操作常用于字符串。但在实际运算过程中，要注意并置操作前后的

数组长度应一致。

- 求积操作符包括*(乘)、/(除)、MOD(取模)和 REM(取余)4 种。VHDL 规定，乘与除的数据类型是整数和实数(包括浮点数)。在一定条件下，还可对物理类型的数据对象进行运算操作。

操作符 MOD 和 REM 的本质与除法操作符是一样的，因此，可综合的取模和取余的操作数必须是以 2 为底数的幂。MOD 和 REM 的操作数的数据类型只能是整数，运算操作结果也是整数。

- 符号操作符“+”和“-”的操作数只有一个，操作数的数据类型是整数，操作符“+”对操作数不作任何改变，操作符“-”作用于操作数后的返回值是对原操作数取负，在实际使用中，取负操作数需加括号。如 Z: =X*(-Y); 。
- 混合操作符包括乘方“**”操作符和取绝对值“ABS”操作符两种。VHDL 规定，它们的操作数数据类型一般为整数类型。乘方运算的左边可以是整数或浮点数，但右边必须为整数，而且只有在左边为浮点时，其右边才可以为负数。一般地，VHDL 综合器要求乘方操作符作用的操作数的底数必须是 2。
- 6 种移位操作符号(SLL、SRL、SLA、SRA、ROL 和 ROR)都是 VHDL'93 标准新增的运算符，其中 SLL 是将位矢向左移，右边跟进的位补零；SRL 的功能恰好与 SLL 相反；ROL 和 ROR 的移位方式稍有不同，它们移出的位将用于依次填补移空的位，执行的是自循环式移位方式；SLA 和 SRA 是算术移位操作符，其移空位用最初的首位来填补。

3. 重载操作符(Overloading Operator)

VHDL 是强类型语言，相同类型的操作数才能进行操作。为了方便各种不同数据类型间的运算，VHDL 允许用户对原有的基本操作符重新定义，赋予新的含义和功能，从而建立一种新的操作符，这就是重载操作符，定义这种操作符的函数称为重载函数。重载操作符可由原操作符加双引号表示。事实上，在程序包 STD_LOGIC_UNSIGNED 中已定义了多种可供不同数据类型间操作的算符重载函数。

Synopsys 的程序包 STD_LOGIC_ARITH、STD_LOGIC_UNSIGNED 和 STD_LOGIC_SIGNED 中已经为许多类型的运算重载了算术运算符和关系运算符，因此只要引用这些程序包，SINGEND、UNSIGEND、STD_LOGIC 和 INTEGER 之间即可混合运算；INTEGER、STD_LOGIC 和 STD_LOGIC_VECTOR 之间也可以混合运算。

【例 3-24】重载操作符的使用示例。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
    USE IEEE.STD_LOGIC_UNSIGNED.ALL  --在该程序包中对标准逻辑矢量之间相加进行了重载
ENTITY OVERLOAD IS
    PORT(A: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      C: IN INTEGER RANGE 0 TO 15;
      SOM1: OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
      SOM2: OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END OVERLOAD;
```

```
ARCHITECTURE EXAMPLE OF OVERLOAD IS
BEGIN
PROCESS(A,B)
BEGIN
    SOM1<=A+B;      --在调用了 STD_LOGIC_UNSIGNED 程序包后此式才成立
    SOM2<=A+C;      --在调用了 STD_LOGIC_UNSIGNED 程序包后此式才成立
END PROCESS;
END EXAMPLE;
```

3.10　本章小结

本章重点研究了 3 个问题。第一个问题：VHDL 程序的基本结构，主要包含库和程序包、实体、结构体和配置四大部分。其中库和程序包的调用可以使程序能够使用 VHDL 语言体系已经定义好的数据类型和函数等，实体用于描述对外的接口，结构体用于描述模块内部的功能结构，而配置部分不是必须的。第二问题：子程序。子程序包含了函数和进程，其定义和使用有两种方法，重点掌握在程序中定义和调用的函数的使用方法。此外还讲到重载函数、重载符号、过程的顺序调用和并行调用。第三个问题：VHDL 语言要素。主要包含文字规则、数据对象、数据类型和运算操作符四大部分，数据对象是 VHDL 语法的独有特性，应从硬件电路设计的角度深刻理解数据对象的内涵，数据类型分清 VHDL 预定数据类型和自定义数据类型，掌握使用方法，重点掌握可综合的数据类型。其中 VHDL 的关键字(key words)、运算操作符、可综合数据类型、STANDERD 程序包、IEEE.STD_LOGIC_1164 程序包间附录部分。

3.11　习　　题

3-1　正确编写 VHDL 程序需要注意哪几大要素？

3-2　VHDL 程序的基本结构是怎么样的？

3-3　VHDL 中标识符的书写规则有哪些？其功能作用又如何？

3-4　在子程序中包含的函数和过程有什么异同点？

3-5　什么是重载函数？重载运算符有何作用？如何调用重载运算符函数？

3-6　VHDL 数据对象在硬件电路中有什么含义？

3-7　VHDL 中常用的数据类型有哪几种？分别被定义在哪几个程序包中？其赋值语句的一般格式如何？

3-8　端口模式 in、out、inout 和 buffer 有何异同点？

3-9　VHDL 支持哪几种运算操作？哪一种运算优先级最高？

3-10　信号与变量的区别有哪些？信号可以用来描述哪些硬件特性？