从零开始学电子技术丛书

从零开始学

CPLD和Verilog HDL编程技术

刘建清 主编
刘建清 刘汉文 高广海 编著

随书附光盘一张

国防工业出版社
National Defense Industry Press

责任编辑：杨星豪　xhyang@ndip.cn
责任校对：钱辉玲
封面设计：王晓军　xjwang@ndip.cn

从零开始学电子技术丛书

从零开始学电路仿真Multisim与电路设计Protel技术
从零开始学电气控制与PLC技术
从零开始学电子测量技术
从零开始学CPLD和Verilog HDL编程技术
从零开始学单片机C语言
从零开始学单片机技术
从零开始学电路基础
从零开始学元器件识别与检测
从零开始学电动机控制与维修
从零开始学模拟电子技术
从零开始学数字电子技术

◎ 上架建议：电子技术 ◎
http://www.ndip.cn
ISBN 7-118-04609-4
9 787118 046090 >

ISBN 7-118-04609-4/TN · 734
定价：30.00 元（含光盘）

从零开始学电子技术丛书

从零开始学 CPLD 和 Verilog HDL 编 程 技 术

刘建清　主编
刘建清　刘汉文　高广海　编著

国防工业出版社
·北京·

内 容 简 介

CPLD(复杂可编程逻辑器件)在数字电子技术领域中的应用越来越广泛,尤其适合于新产品的开发与小批量生产,因此深受广大工程技术人员喜爱。

本书定位于让初学者从零起步,轻松学会 CPLD 的系统设计技术。本书以 ALTERA 公司的系列芯片为目标载体,简要分析了可编程逻辑器件的结构和特点,以及相应开发软件的使用方法,同时,还用大量篇幅介绍了初学者最容易掌握的 Verilog HDL 硬件描述语言。本书完全以实战为主,通过实践的方法帮助读者加深理解 CPLD 的基本知识。

本书附赠光盘一张,光盘中包含了书中所有实验的源程序。

本书可供从事各类电子系统设计的广大工程技术人员以及电子爱好者阅读,也可作为电子类专业的教材或教学参考书使用。

图书在版编目(CIP)数据

从零开始学 CPLD 和 Verilog HDL 编程技术/刘建清主编;刘建清,刘汉文,高广海编著. —北京:国防工业出版社,2006.8

(从零开始学电子技术丛书)

ISBN 7-118-04609-4

Ⅰ.从... Ⅱ.①刘...②刘...③刘...④高...

Ⅲ.①可编程序逻辑器件②硬件描述语言,VHDL—程序设计 Ⅳ.①TP332.1②TP312

中国版本图书馆 CIP 数据核字(2006)第 071740 号

※

国防工业出版社出版发行

(北京市海淀区紫竹院南路 23 号 邮政编码 100044)

北京四季青印刷厂印刷

新华书店经售

*

开本 787×1092 1/16 **印张** 16½ **字数** 376 千字

2006 年 8 月第 1 版第 1 次印刷 **印数** 1—5000 册 **定价** 30.00 元(含光盘)

(本书如有印装错误,我社负责调换)

国防书店:(010)68428422 发行邮购:(010)68414474

发行传真:(010)68411535 发行业务:(010)68472764

丛书前言

我们所处的时代是一个知识爆炸的新时代。新产品、新技术层出不穷，电子技术的发展更是日新月异。可以毫不夸张地说，电子技术的应用无处不在，电子技术正在不断地改变着我们的生活，改变着我们的世界。

读者朋友：当你对妙趣横生的电子世界发生兴趣时；当你彷徨于就业的关口，想成为电子产业中的一名员工时；当你跃跃欲试，想成为一名工厂的技术革新能手时；当你面对“无所不能”的“单片机”，梦想成为一名自动化高手时；当你的头脑里冒出那么多的奇思妙想，急于把它们应用于或转化为产品时……都是那么急切地想补充自己有关电子技术方面的知识，这时，你首先想到的是找一套适合自己学习的电子技术图书阅读。《从零开始学电子技术丛书》正是为了满足广大读者特别是电子爱好者的实际需要和零起点入门的阅读要求而编著的。

和其他电子技术类图书相比，本丛书具有以下特点：

内容全面，体系完备。本丛书给出了广大电子爱好者学习电子技术的全方位解决方案，既有初学者必须掌握的电路基础、模拟电路和数字电路等基础理论，又有电子元器件检测、电子测量仪器的使用、电路仿真与设计等操作性较强的内容，还有电气控制与PLC、单片机、CPLD等综合应用方面的知识，因此，本丛书内容翔实，覆盖面广。

通俗易懂，重点突出。传统的电子技术图书和教材在介绍电路基础和模拟电子技术等内容时，大都借助高等数学这一工具进行分析，这就给电子爱好者自学电子技术设置了一道门槛，使大多数电子爱好者失去了学习的热情和兴趣。本丛书在编写时，完全考虑到了初学者的需要，不涉及高等数学方面的公式，尽可能地把复杂的理论通俗化和实用化，将烦琐的公式简易化，再辅以简明的分析及典型的实例，从而形成了本丛书通俗易懂的特点。为了满足不同层次读者的需求，本丛书对难点和扩展知识用“*”进行了标注，初学者可跳过此内容。

实例典型，实践性强。本丛书最大程度地强调了实践性，书中给出的例子大都经过了验证，可以实现，并且具有代表性；本丛书中每本书都配有光盘，光盘中收录了书中的实例、常用软件、实验程序和大量珍贵资料，以方便读者学习和使用。

内容新颖，风格活泼。本丛书所介绍的都是电子爱好者最为关心并且在业界获得普遍认同的内容，本丛书的每一分册都各有侧重，又互相补充，论述时疏密结合，重点突出。对于重点、难点和容易混淆的知识，书中还特别进行了标注和提示。

把握新知，结合实际。电子技术发展日新月异，为适应时代的发展，本丛书还对电子技术的新知识做了详细的介绍；本丛书中涉及的应用实例都是编著者开发经验的提炼和总结，相信一定会给读者带来很大的帮助。在讲述电路基础、模拟和数字电子技术时，还

专门安排了计算机辅助软件的仿真实验，实验过程非常接近实际操作的效果，使电子技术的学习变得更为直观，使学习变得更加生动有趣，这可以加深读者对电路理论知识的认识。

总之，对于需要学习电子技术的电子爱好者而言，选择《从零开始学电子技术丛书》不失为一个好的选择。本丛书一定能给你耳目一新的感觉，当你认真阅读之后将会发现，无论是你所读的书，还是读完书的你，都有所不同。

感谢本丛书的策划者——电子科普领域中的知名专家、中国电子学会高级会员刘午平先生，他与我们共同交流，共同探讨，达成了共识，确立了写作方向，并为本丛书的编排、修改和出版做了大量卓有成效的工作，他以丰富的专业知识和认真、敬业的态度为我们所敬佩；感谢山东持恒开关厂总经理陈培军先生和山东金曼克电气集团设计处总工程师高广海先生，他们对本丛书的编写提出了很多建设性的意见和建议，为本丛书的许多实验提供了强有力的支持与帮助，并参与了部分图书的编写工作；感谢网络，本丛书的许多新知识、新内容都是我们通过网络而获得的，我们在写作过程中遇到的许多疑难问题也大都通过网络得以顺利解决，对于这么多乐于助人、无私奉献的站主和作者们，无法在此一一列举，只能道一声“谢谢了！”感谢众多电子报刊、杂志的编辑和作者，他们为本丛书提供了许多有新意、有实用价值的参考文献，使得这套丛书能够别出心裁、与时俱进；感谢国防工业出版社，能与国内一流的出版社合作，我们感到万分的荣幸；感谢其他对本丛书的出版付出过辛勤工作的人士，没有他们的热心与支持，本丛书不知何时才能与读者见面！

最后，祝愿本丛书的每一位读者在学习电子技术的过程中，扬起风帆，乘风破浪！

丛书编者

前　言

CPLD是“复杂可编程逻辑器件”的缩写。CPLD可以完成任何数字器件的功能，上至高性能CPU，下至简单的74系列数字电路，都可以用CPLD来实现。CPLD如同一张白纸或是一堆积木，工程师可以通过传统的原理图输入法或硬件描述语言（如Verilog-HDL）自由地设计一个数字系统。通过软件仿真，我们可以事先验证设计的正确性。在PCB（印制电路板）完成以后，还可以利用CPLD的在线修改能力，随时修改设计而不必改动硬件电路。用CPLD来开发数字电路，使用灵活，设计周期短，费用低，而且可靠性好，承担风险小，因而很快得到普遍应用，发展非常迅速。

硬件描述语言HDL（Hardware Description Language）是一种用形式化方法描述数字电路和系统的语言。其主要目的是用来编写设计文件，建立电子系统仿真模型，利用计算机和EDA工具，对用硬件描述语言建模的数字逻辑进行仿真，然后再自动综合，以生成符合要求且在电路结构上可以实现的数字逻辑网表，根据网表和某种工艺的器件，自动生成具体电路，然后生成该工艺条件下这种具体电路的延时模型，仿真验证无误后，写入CPLD器件中。在我国，比较有影响的主要有3种硬件描述语言：超高速集成电路硬件描述语言VHDL、Verilog HDL语言和AHDL语言。其中，Verilog HDL语言易于上手，因而拥有更广泛的设计群体，应用十分广泛。

本书定位于复杂可编程逻辑器件（CPLD）的系统设计技术，以ALTERA公司的系列芯片为目标载体，简要分析了可编程逻辑器件的结构和特点，以及相应的开发软件的使用方法。同时，本书还用大量篇幅介绍初学者最容易掌握的Verilog HDL硬件描述语言。如果读者具有C语言的基础，那么只要二三个月的时间就能完全学好CPLD的应用设计。而且Verilog HDL语言的最大好处就是不受最终器件的限制，只要写的程序没有问题，就可以完全无障碍地移植到任何其他半导体公司的CPLD上去使用。本书完全以实战为主，通过实践的方法帮助读者加深对CPLD的基本知识的理解，更重要的是通过实战可以帮助读者时刻保持对学习的兴趣。

本书附赠光盘一张，光盘中包含了书中所有实验的源程序。

由于时间仓促，书中错漏之处在所难免，敬请广大读者批评指正。

作　者

2006年6月

目　录

第一章　CPLD 与 FPGA 概述 …… 1
第一节　可编程逻辑器件的发展及特点 …… 1
一、可编程逻辑器件的发展 …… 1
二、CPLD/FGPA 的用途 …… 3
三、CPLD/FPGA 的特点 …… 3
四、CPLD 与 FPGA 的比较 …… 3
五、CPLD/FPGA 和单片机的比较 …… 5
第二节　CPLD/FPGA 的基本工作原理 …… 5
一、基于乘积项的 CPLD 的工作原理 …… 5
二、采用查找表的 FPGA 的工作原理 …… 7
第三节　Altera 系列 CPLD 介绍 …… 9
一、MAX7000 系列器件简介 …… 9
二、MAX7000 系列器件的结构 …… 10
三、MAX7000 系列器件功能描述 …… 13
第四节　Xilinx 系列 CPLD 介绍 …… 15
一、XC9500 系列器件简介 …… 15
二、XC9500 系列器件的结构 …… 16
三、XC9500 系列器件功能描述 …… 20
第五节　可编程逻辑器件的开发 …… 21
一、可编程逻辑器件的设计过程 …… 21
二、可编程逻辑器件设计举例 …… 23
第二章　CPLD 实验仪介绍 …… 25
第一节　DP-MCU/Altera 实验仪 …… 25
一、实验仪主要器件 …… 25
二、应用接口 …… 27
三、跳线接口 …… 29
四、原理简介 …… 31
第二节　DP-MCU/Xilinx 实验仪 …… 36
一、实验仪主要器件 …… 36
二、应用接口 …… 38
三、跳线接口 …… 39
四、原理简介 …… 40
第三节　其他 CPLD 实验仪 …… 40
一、CPLD-MCU 下载仿真实验仪 …… 40

二、Altera CPLD 开发板 …… 42
三、51+CPLD 学习板 …… 44
第三章 CPLD 开发软件和仿真软件的使用 …… 45
第一节 Altera 开发软件 MAX+plusII 的安装和使用 …… 45
一、MAX+plusII 的安装 …… 45
二、MAX+plusII 的使用 …… 48
第二节 Xilinx 开发软件 ISE WebPACK 的安装和使用 …… 61
一、WebPACK 软件的安装 …… 62
二、WebPACK 软件的使用 …… 63
第三节 仿真 Modelsim SE 软件的安装和使用 …… 72
一、Modelsim SE 6.0 软件的安装 …… 73
二、Modelsim SE 6.0 软件的使用 …… 74
第四章 初识 Verilog HDL …… 85
第一节 硬件描述语言概述 …… 85
一、什么是硬件描述语言 …… 85
二、硬件描述语言的发展 …… 85
三、为何使用硬件描述语言 …… 85
第二节 Verilog HDL 基本知识 …… 86
一、什么是 Verilog HDL …… 86
二、Verilog HDL 的发展 …… 86
三、Verilog HDL 与 VHDL 比较 …… 86
四、Verilog HDL 与 C 语言的比较 …… 87
第三节 Verilog HDL 模块介绍 …… 87
一、什么是模块 …… 87
二、模块的结构 …… 87
第五章 Verilog HDL 数据类型与运算符 …… 89
第一节 Verilog HDL 基本词法 …… 89
一、标识符 …… 89
二、关键字 …… 89
三、注释 …… 90
四、空白符 …… 90
第二节 Verilog HDL 常量变量及其数据类型 …… 90
一、常量及其数据类型 …… 90
二、变量及其数据类型 …… 92
第三节 Verilog HDL 运算符 …… 94
一、算术运算符 …… 94
二、逻辑运算符 …… 94
三、位运算符 …… 95
四、关系运算符 …… 96

五、等式运算符 …… 96
六、缩位运算符 …… 97
七、移位运算符 …… 97
八、条件运算符 …… 98
九、位拼接运算符 …… 98
第六章 Verilog HDL 基本语句 …… 100
第一节 赋值语句 …… 100
一、持续赋值语句 …… 100
二、过程赋值语句 …… 100
第二节 块语句 …… 103
一、串行块语句 begin-end …… 103
二、并行块语句 fork-join …… 104
第三节 过程语句 …… 105
一、initial 过程语句 …… 105
二、always 过程语句 …… 106
第四节 条件语句 …… 108
一、if 条件语句 …… 108
二、case 条件语句 …… 110
第五节 循环语句 …… 112
一、forever 语句 …… 112
二、repeat 语句 …… 112
三、while 语句 …… 113
四、for 语句 …… 113
第六节 编译向导语句 …… 115
一、宏替换`define …… 115
二、文件包含`include …… 116
三、条件编译`ifdef、`else、`endif …… 116
四、时间尺度`timescale …… 117
第七节 任务(task)和函数(function)说明语句 …… 118
一、任务(task)说明语句 …… 118
二、函数(function)说明语句 …… 120
第八节 系统任务与系统函数 …… 121
一、$ display 和 $ write 任务 …… 121
二、$ monitor 与 $ strobe …… 122
三、$ time 与 $ realtime …… 122
四、$ finish 与 $ stop …… 124
第七章 Verilog HDL 的描述方式 …… 125
第一节 结构描述方式 …… 125
一、Verilog HDL 内置门元件 …… 125

二、门级结构描述 …… 128
第二节　数据流描述方式 …… 129
第三节　行为描述方式 …… 130
第八章　用 Verilog HDL 描述数字电路 …… 132
第一节　基本门电路的设计 …… 132
一、与门 …… 132
二、或门 …… 132
三、非门 …… 133
四、与非门 …… 134
五、或非门 …… 134
六、异或门 …… 135
七、缓冲门 …… 135
八、三态门 …… 136
第二节　组合逻辑电路的设计 …… 136
一、数据选择器 …… 137
二、编码器 …… 143
三、译码器 …… 148
四、加法器 …… 160
第三节　双稳态触发器的设计 …… 169
一、RS 触发器 …… 169
二、D 触发器 …… 176
三、JK 触发器 …… 180
四、T 触发器 …… 186
第四节　时序逻辑电路的设计 …… 189
一、寄存器 …… 189
二、锁存器 …… 196
三、计数器 …… 199
第九章　CPLD 实验与综合设计实例 …… 215
第一节　CPLD 基本实验 …… 215
一、LED 发光二极管实验 …… 215
二、键盘实验 …… 219
三、数码 LED 显示器实验 …… 223
四、音响实验 …… 231
第二节　CPLD 综合设计实例 …… 235
一、乐曲演奏电路 …… 236
二、数字钟 …… 241
三、频率计 …… 246
四、交通灯 …… 249
参考文献 …… 254

第一章　CPLD 与 FPGA 概述

PLD 是可编程逻辑器件(Programmable Logic Device)的英语缩写，是一种数字集成电路的半成品，在其芯片上按一定排列方式集成了大量的门和触发器等基本逻辑元件，使用者可利用某种开发工具对其进行加工，即按设计要求将这些片内的元件连接起来(此过程称为编程)，使之完成某个逻辑电路或系统的功能，成为一个可在实际电子系统中使用的专用集成电路。现在应用最广泛的 PLD 主要是复杂可编程逻辑器件 CPLD(Complex Programmable Logic Device)和现场可编程门阵列 FPGA(Field Programmable Gate Array)。本章主要介绍 CPLD 与 FPGA 的发展、特点、常用芯片及基本工作原理，并对可编程逻辑器件的开发做一简要说明。

第一节　可编程逻辑器件的发展及特点

一、可编程逻辑器件的发展

自 20 世纪 60 年代以来，数字集成电路已经历了从 SSI(small scale integrated circuit)、MSI(medium scale integrated circuit)、LSI(large scale integrated circuit)到 VLSI (very large scale integrated circuit)的发展过程。数字集成电路按照芯片设计方法的不同大致可以分为 3 类：一类是通用型中、小规模集成电路；第二类是用软件组态的大规模、超大规模集成电路，如微处理器、单片机等；第三类是专用集成电路(ASIC，application-specific integrated circuit)。

ASIC 是一种专门为某一应用领域或为专门用户需要而设计、制造的 LSI 或 VLSI 电路，它可以将某些专用电路或电子系统设计在一个芯片上，构成单片集成系统。

ASIC 分为全定制和半定制两类。全定制 ASIC 的硅片没有经过预加工，其各层掩模都是按特定电路功能专门制造的。半定制 ASIC 是按一定规格预先加工好的半成品芯片，然后再按具体要求进行加工和制造，它包括门阵列、标准单元和可编程逻辑器件三种。门阵列是一种预先制造好的硅阵列，内部包括基本逻辑门、触发器等，芯片中留有一定连线区，用户根据所需要的功能设计电路，确定连线方式，然后交厂家进行最后的布线。标准单元是厂家将预先配置好、经过测试、具有一定功能的逻辑块作为标准单元存在数据库中，设计者根据需要在库中选择单元构成电路，并完成电路到版图的最终设计。这两种半定制 ASIC 都要由用户向生产厂家定做，设计和制造周期较长，开发费用也较高，因此只适用于批量较大的产品生产中。

可编程逻辑器件是 ASIC 的一个重要分支，它是厂家作为一种通用型器件生产的半定制电路，用户可以利用软、硬件开发工具对器件进行设计和编程，使之实现所需要的逻辑功能。由于它是用户可配置的逻辑器件，使用灵活，设计周期短，费用低，而且可靠性

好，承担风险小，因而很快得到普遍应用，发展非常迅速。

可编程逻辑器件(PLD)能够完成各种数字逻辑功能。典型的 PLD 由输入电路、与阵列、或阵列和输出电路组成，如图 1-1 所示，而任意一个组合逻辑都可以用“与—或”表达式来描述，所以，PLD 能以乘积和的形式完成大量的组合逻辑功能。

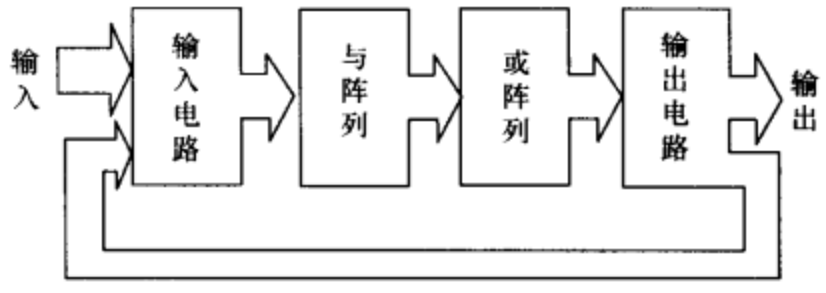


图 1-1　PLD 的基本组成

重点提示　PLD 是这样一种 ASIC(专用集成电路)，内部有大量的门电路，通过软件编程可以实现这些门电路不同的连接关系，从而使整个 PLD 对外就完成了不同的功能，并且这些门电路的连接关系可以不断地用软件来修改。

早期的 PLD 产品主要有可编程阵列逻辑 PAL(programmable array logic)和通用阵列逻辑 GAL(generic array logic)。PAL 由一个可编程的“与”平面和一个固定的“或”平面构成，或门的输出可以通过触发器有选择地被置为寄存状态。PAL 器件是现场可编程的，它的实现工艺有反熔丝技术、EPROM 技术和 EEPROM 技术。在 PAL 的基础上，又发展了一种通用阵列逻辑 GAL，如 GAL16V8，GAL22V10 等。它采用了 EEPROM 工艺，实现了电可擦除、电可改写，其输出结构是可编程的逻辑宏单元，因而它的设计具有很强的灵活性，至今仍有许多人使用。这些早期的 PLD 器件的一个共同特点是可以实现速度特性较好的逻辑功能，但其过于简单的结构也使它们只能实现规模较小的电路。

为了弥补这一缺陷，20 世纪 80 年代中期，Altera 和 Xilinx 公司在 PAL、GAL 等逻辑器件的基础之上，分别推出了复杂可编程逻辑器件 CPLD 和现场可编程门阵列 FPGA，同以往的 PAL、GAL 等相比较，CPLD/FPGA 具有体系结构和逻辑单元灵活、集成度高以及适用范围宽等特点。它们可以替代几十甚至几千块通用 IC 芯片。这样的 CPLD/FPGA 实际上就是一个子系统部件。这种芯片受到世界范围内电子工程设计人员的广泛关注和普遍欢迎。经过了十几年的发展，许多公司都开发出了多种可编程逻辑器件。比较典型的就是 Xilinx 和 Altera 公司的 CPLD/FPGA 系列器件，它们开发较早，占用了较大的 PLD 市场。通常来说，在欧洲用 Xilinx 的人多，在日本和亚太地区用 Altera 的人多，在美国则是平分秋色。全球 CPLD/FPGA 产品 60%以上是由 Altera 和 Xilinx 提供的。可以讲 Altera 和 Xilinx 共同决定了 PLD 技术的发展方向。当然还有许多其他类型器件，如 Lattice、Vantis 等。

重点提示　不同厂家对 CPLD 和 FPGA 的叫法不尽相同，Xilinx 把基于查找表技术，SRAM 工艺，要外挂配置用的 EEPROM 的 PLD 叫 FPGA；把基于乘积项技术，Flash ROM 工艺的 PLD 叫做 CPLD；Altera 把自己的 PLD 产品 MAX 系列(乘积项技术，EEPROM 工艺)和 FLEX 系列(查找表技术，SRAM 工艺)都叫做 CPLD，由于 FLEX 系列也是 SRAM 工艺，基于查找表技术，要外挂配置用的 EPROM，用法和 Xilinx 的 FPGA 一样，所以，很多人把 Altera 的 FELX 系列产品也叫做 FPGA。

二、CPLD/FGPA 的用途

CPLD/FGPA 能做什么呢？可以毫不夸张地讲，CPLD/FGPA 能完成任何数字器件的功能，上至高性能 CPU，下至简单的 74 电路，都可以用 CPLD/FGPA 来实现。CPLD/FGPA 如同一张白纸或是一堆积木，工程师可以通过传统的原理图输入法或硬件描述语言（如 Verilog-HDL、VHDL）自由地设计一个数字系统。通过软件仿真，可以事先验证设计的正确性。在 PCB（电路印制板）完成以后，还可以利用CPLD/FGPA的在线修改能力，随时修改设计而不必改动硬件电路。使用 CPLD/FGPA 来开发数字电路，可以大大缩短设计时间，减少 PCB 面积，提高系统的可靠性。

三、CPLD/FPGA 的特点

CPLD/FPGA 芯片都是特殊的 ASIC 芯片，它们除了具有 ASIC 的特点之外，还具有以下几个优点：

（1）随着超大规模集成电路工艺的不断提高，单一芯片内部可以容纳上百万个晶体管，CPLD/FPGA 芯片的规模也越来越大，其单片逻辑门数已达到上百万门，它所能实现的功能也越来越强，同时也可以实现系统集成。

（2）CPLD/FPGA 芯片在出厂之前都做过百分之百的测试，不需要设计人员承担投片风险和费用，设计人员只需在自己的实验室里就可以通过相关的软硬件环境来完成芯片的最终功能设计。所以，CPLD/FPGA 的资金投入小，节省了许多潜在的花费。

（3）用户可以反复地编程、擦除、使用或者在外围电路不动的情况下用不同软件就可实现不同的功能。所以，用 CPLD/FPGA 试制样片，能以最快的速度占领市场。CPLD/FPGA软件包中有各种输入工具和仿真工具，及版图设计工具和编程器等全线产品，电路设计人员在很短的时间内就可完成电路的输入、编译、优化、仿真，直至最后芯片的制作。当电路有少量改动时，更能显示出 CPLD/FPGA 的优势。电路设计人员使用 CPLD/FPGA 进行电路设计时，不需要具备专门的 IC（集成电路）深层次的知识；CPLD/FPGA 软件易学易用，可以使设计人员更能集中精力进行电路设计，快速将产品推向市场。

CPLD/FGPA 的这些优点使得 CPLD/FGPA 技术在 20 世纪 90 年代以后得到飞速的发展，同时也大大推动了 EDA（电路设计自动化）软件和硬件描述语言的进步。

四、CPLD 与 FPGA 的比较

FPGA 是一种高密度的可编程逻辑器件，自从 Xilinx 公司 1985 年推出第一片 FPGA 以来，FPGA 的集成密度和性能提高很快，其集成密度最高达 500 万门/片以上，系统性能可达 200MHz。由于 FPGA 器件集成度高，方便易用，开发和上市周期短，在数字设计和电子生产中得到迅速普及和应用，并一度在高密度的可编程逻辑器件领域中独占鳌头。

CPLD 是由 GAL 发展起来的，其主体结构仍是与或阵列，自 20 世纪 90 年代初 Lattice 公司开发出在系统可编程 CPLD 以来，CPLD 发展迅速。具有 ISP（在系统编程）功能的 CPLD 器件由于具有同 FPGA 器件相似的集成度和易用性，在速度上还有一定的优势，使其在可编程逻辑器件技术的竞争中与 FPGA 并驾齐驱，成为两支领导可编程器件技术发展的力量之一。

CPLD 与 FPGA 对照情况如下。

1. 结构

FPGA 器件是由逻辑功能块排列为阵列，并由可编程的内部连线连接这些功能块来实现一定的逻辑功能。

CPLD 是由可编程的与/或门阵列以及宏单元构成。与/或阵列是可重新编程的，可以实现多种逻辑功能。宏单元则是可实现组合或时序逻辑的功能模块，同时还提供了真值或补码输出和以不同的路径反馈等额外的灵活性。

2. 集成度

FPGA 可以达到比 CPLD 更高的集成度，同时也具有更复杂的布线结构和逻辑实现。

3. 适合结构

CPLD 组合逻辑的功能很强，一个宏单元就可以分解十几个甚至 20～30 多个组合逻辑输入。而 FPGA 的一个 LUT 只能处理 4 输入的组合逻辑，因此，CPLD 适合用于设计译码等复杂组合逻辑。

FPGA 的制造工艺确定了 FPGA 芯片中包含的 LUT 和触发器的数量非常多，往往都是几千上万，CPLD 一般只能做到 512 个逻辑单元。所以如果设计中使用到大量触发器，例如设计一个复杂的时序逻辑，那么使用 FPGA 就是一个很好的选择。

4. 功率消耗

一般情况下，CPLD 功耗要比 FPGA 大，且集成度越高越明显。

5. 速度

CPLD 优于 FPGA。由于 FPGA 是门级编程，且逻辑块之间是采用分布式互连；而 CPLD 是逻辑块级编程，且其逻辑块互连是集总式的。因此，CPLD 比 FPGA 有较高的速度和较大的时间可预测性。

6. 编程方式

目前的 CPLD 主要是基于 EEPROM 或 FLASH 存储器编程，编程次数达 1 万次。其优点是在系统断电后，编程信息不丢失。CPLD 又可分为在编程器上编程和在系统编程(ISP)两种。ISP 器件的优点是不需要编程器，可先将器件装焊于印制板，再经过编程电缆进行编程，编程、调试和维护都很方便。

FPGA 大部分是基于 SRAM 编程，其缺点是编程数据信息在系统断电时丢失，每次上电时，需从器件的外部存储器或计算机中将编程数据写入 SRAM 中。其优点是可进行任意次数的编程，并可在工作中快速编程，实现板级和系统级的动态配置，因此可称为在线重配置(ICR：In Circuit Reconfigurable)的 PLD 或可重配置硬件(RHP：Reconfigurable Hardware Product)。

7. 使用方便性

在使用方便性上，CPLD 比 FPGA 要好。CPLD 的编程工艺采用 EEPROM 或 FLASH 技术，无需外部存储器芯片，使用简单，保密性好。而基于 SRAM 编程的 FPGA，其编程信息需存放在外部存储器上，需外部存储器芯片，且使用方法复杂，保密性差。

重点提示 对于初学者，一般选择 CPLD，因为 CPLD 价格低，许多 CPLD 为 5V，可以直接和 TTL、CMOS 电路电压兼容，不必考虑电源转换问题，很多低价的 CPLD 是 PLCC 封装，插拔很方便，而 FPGA 一般是 QFP 封装，一旦损坏，很难从电路板上取下。

五、CPLD/FPGA 和单片机的比较

CPLD/FPGA 和单片机比较，CPLD/FPGA 在时序和延迟的实现上不如单片机，但是，CPLD/FPGA 在芯片容量、组合逻辑、工作速度、编程难度以及可以擦写次数上（特别是 FPGA）远优于单片机。

CPLD/FPGA 是电子设计领域中最具活力和发展前途的一项技术，它的影响毫不亚于 20 世纪 70 年代单片机的发明与使用。

第二节　CPLD/FPGA 的基本工作原理

一、基于乘积项的 CPLD 的工作原理

1. 基于乘积项的 CPLD 的结构

基于乘积项的 CPLD 芯片有 Altera 的 MAX7000，MAX3000 系列（EEPROM 工艺）、Xilinx 的 XC9500 系列（Flash 工艺）和 Lattice、Cypress 的大部分产品（EEPROM 工艺），下面以 MAX7000 系列的 EMP7032/EMP7064/EMP7096（如图 1-2 所示）为例，看一下这种 PLD 的总体结构，其他型号的结构与此都非常相似。

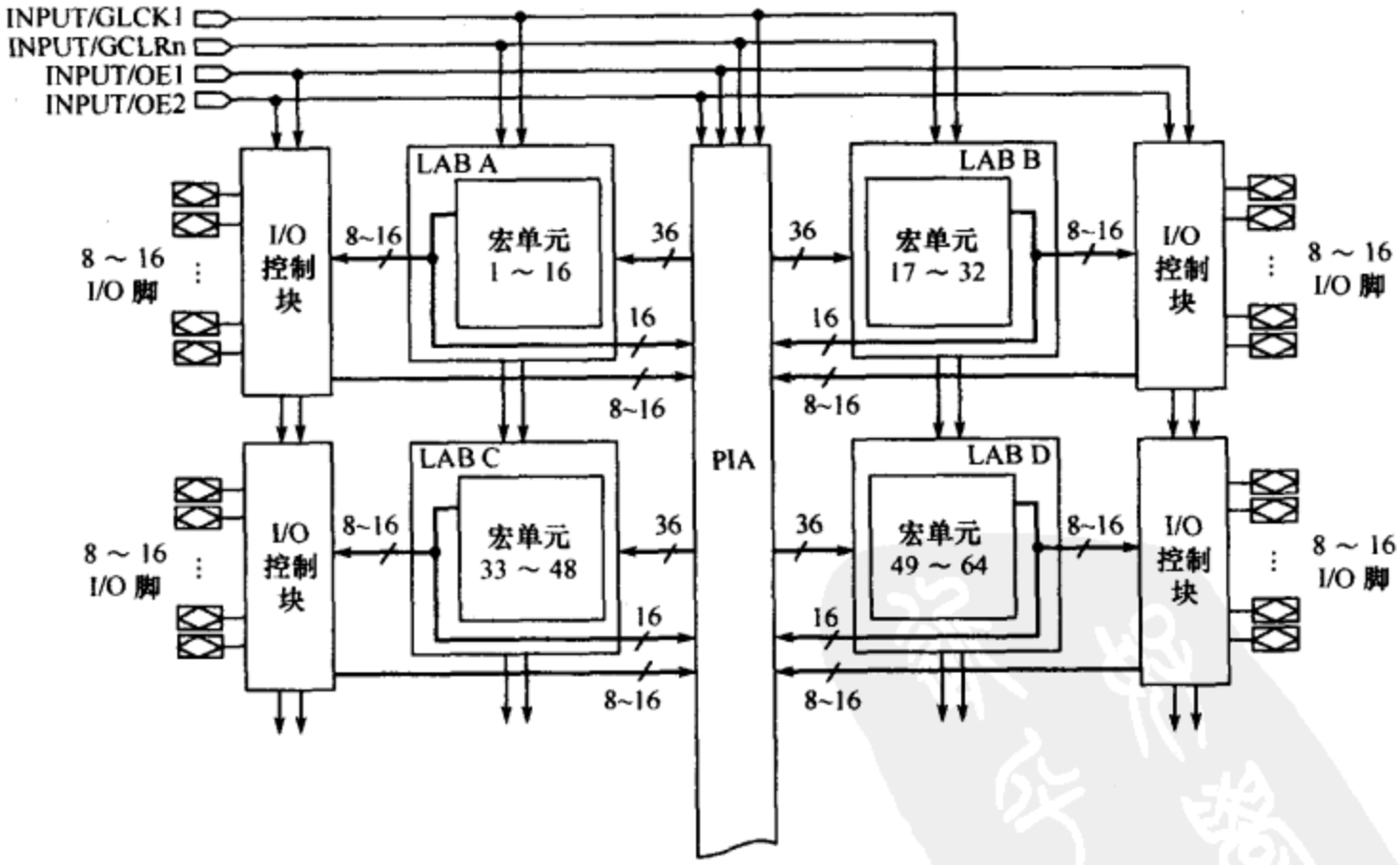


图 1-2　EMP7032/EMP7064/EMP7096 器件的结构

这种 CPLD 的结构主要包括宏单元、可编程连线（PIA）和 I/O 控制块。宏单元是 CPLD 的基本结构，由它来实现基本的逻辑功能。图中灰色部分是多个宏单元的集合（因为宏单元较多，没有一一画出）。可编程连线 PIA 负责信号传递，连接所有的宏单元。I/O控制块负责输入输出的电气特性控制，比如可以设定集电极开路输出、摆率控制、三态输出等。图上部的 INPUT/GCLK1，INPUT/GCLRn，INPUT/OE1，INPUT/OE2 是全

局时钟、清零和输出使能信号，这几个信号有专用连线与 CPLD 中每个宏单元相连，信号到每个宏单元的延时相同并且延时最短。EMP7032/EMP7064/EMP7096 宏单元的具体结构如图 1-3 所示。

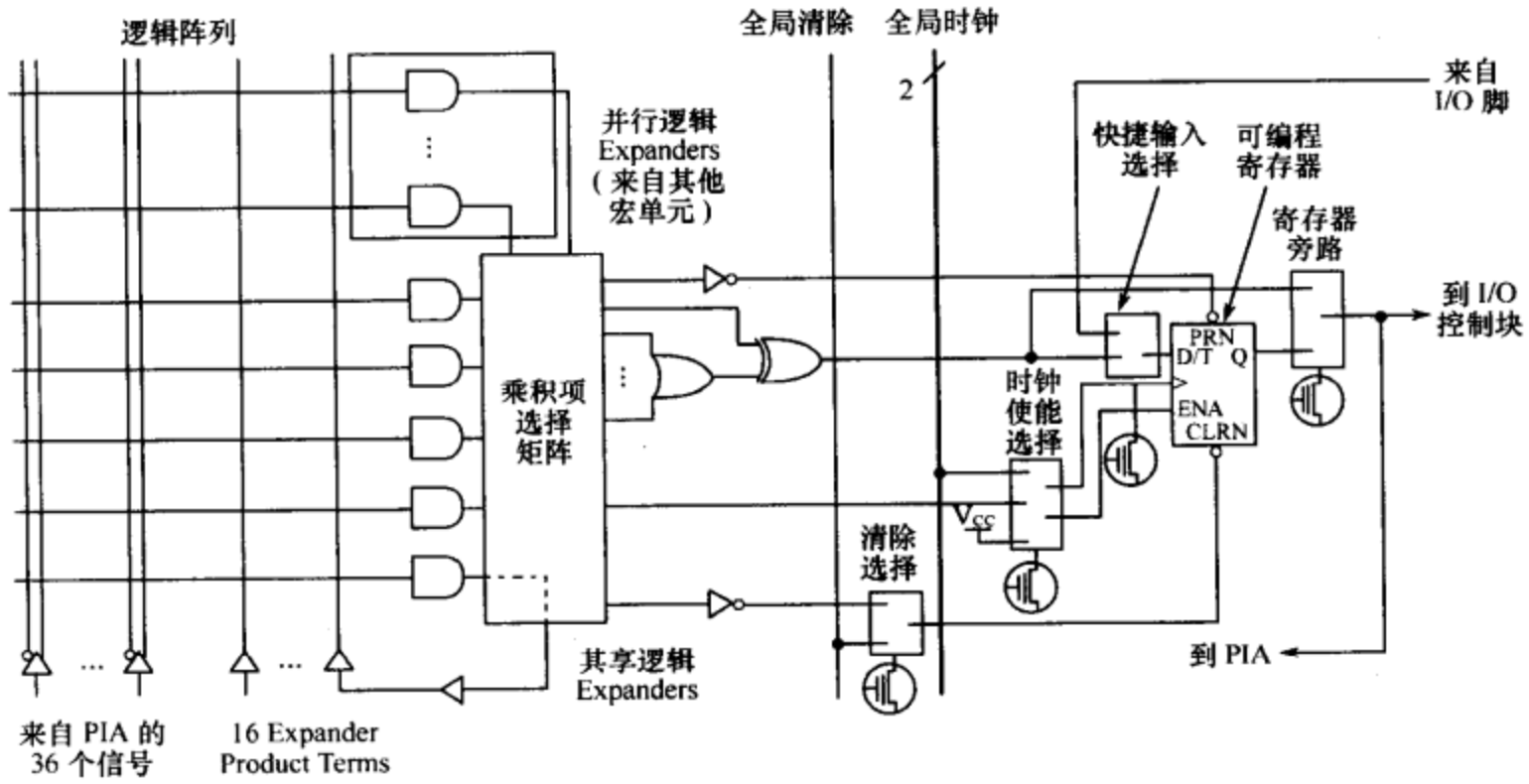


图 1-3　EMP7032/EMP7064/EMP7096 的宏单元

左侧是乘积项阵列，实际上是一个“与或”阵列，每个交叉点都是一个可编程熔丝，如果导通就实现与逻辑，后面的乘积项选择矩阵是一个“或”阵列。两者一起完成组合逻辑。最右侧是一个可编程 D 触发器，它的时钟、清零输入都可以编程选择，可以使用专用的全局清零和全局时钟，也可以使用内部逻辑（乘积项阵列）产生的时钟和清零。如果不需要触发器，也可以将此触发器旁路，信号直接输给 PIA 或输出到 I/O 脚。

2. 基于乘积项结构 CPLD 逻辑实现方式

下面我们以一个简单的电路为例，具体说明 CPLD 是如何利用以上结构实现逻辑的，电路如图 1-4 所示。

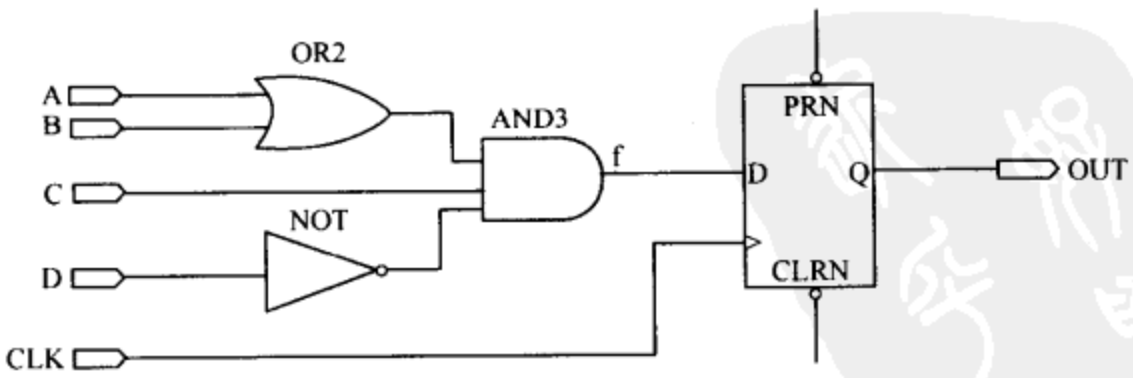


图 1-4　基本逻辑电路

假设组合逻辑的输出（AND3 的输出）为 f，则 f=(A+B)*C*(~D)=A*C*(~D)+B*C*(~D)（“~”表示“逻辑非”）

CPLD 将以如图 1-5 所示的电路来实现组合逻辑 f。

A、B、C、D 由 CPLD 芯片的管脚输入后进入可编程连线阵列（PIA），在内部会产生 A、~A、B、~B、C、~C、D、~D 八个输出。图中每一个“×”表示相连（可编程熔丝导通），

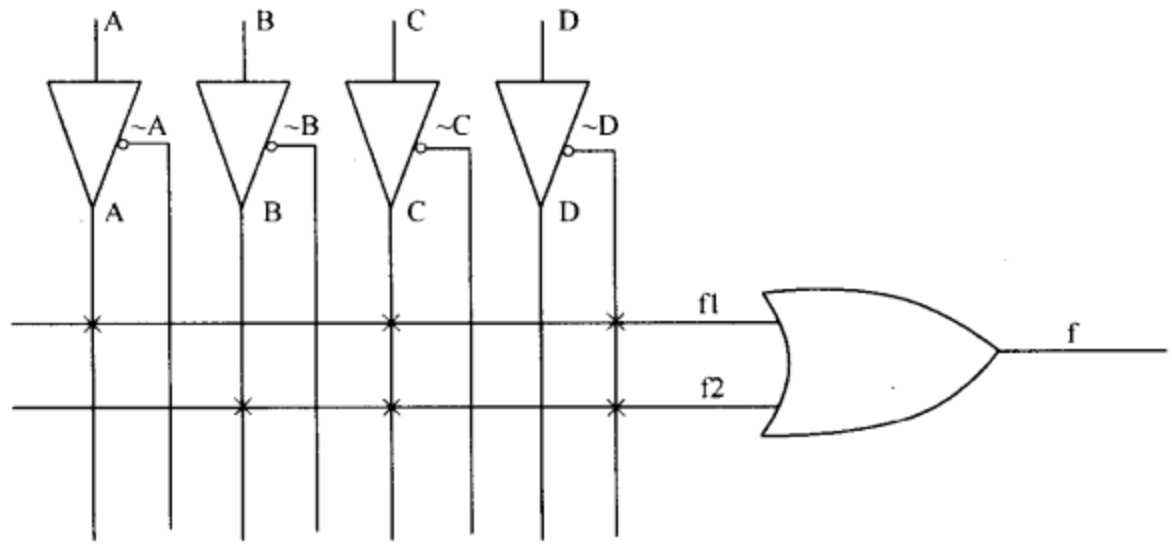


图 1-5　CPLD 的实现方式

所以得到：f＝f1＋f2＝A* C* (～D)＋B* C* (～D)。这样组合逻辑就实现了。

图 1-4 电路中 D 触发器的实现比较简单，直接利用宏单元中的可编程 D 触发器来实现。时钟信号 CLK 由 I/O 脚输入后进入芯片内部的全局时钟专用通道，直接连接到可编程触发器的时钟端。可编程触发器的输出与 I/O 脚相连，把结果输出到芯片管脚。这样 CPLD 就完成了图 1-4 所示电路的功能。以上这些步骤都是由软件自动完成的，不需要人为干预。

图 1-4 的电路是一个很简单的例子，只需要一个宏单元就可以完成。但对于一个复杂的电路，一个宏单元是不能实现的，这时就需要通过并联扩展项和共享扩展项将多个宏单元相连，宏单元的输出也可以连接到可编程连线阵列，再作为另一个宏单元的输入。这样 CPLD 就可以实现更复杂逻辑。

这种基于乘积项的 CPLD 基本都是由 EEPROM 和 Flash 工艺制造的，一上电就可以工作，无需其他芯片配合。

二、采用查找表的 FPGA 的工作原理

1. 查找表的原理与结构

采用查找表结构的 PLD 芯片称之为 FPGA，如 Altera 的 ACEX、APEX 系列，Xilinx 的 Spartan、Virtex 系列等。

查找表(Look-Up-Table)简称为 LUT，LUT 本质上就是一个 RAM。目前 FPGA 中多使用 4 输入的 LUT，所以每一个 LUT 可以看成一个有 4 位地址线的 16×1 的 RAM。当用户通过原理图或硬件描述语言描述了一个逻辑电路以后，FPGA 开发软件会自动计算逻辑电路的所有可能的结果，并把结果事先写入 RAM，这样，每输入一个信号进行逻辑运算就等于输入一个地址进行查表，找出地址对应的内容，然后输出即可。

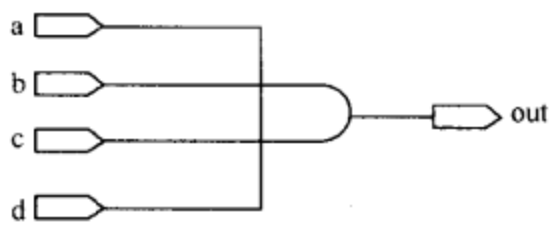
表 1-1 是一个 4 输入与门的例子。

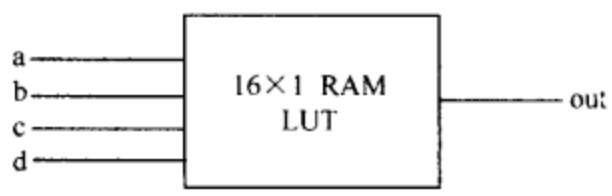
对于 Altera 的 ACEX 系列 FPGA 芯片，其结构主要包括 LAB，I/O 块，RAM 块和可编程行/列连线。在 ACEX 系列中，一个 LAB 包括 8 个逻辑单元(LE)，每个 LE 包括一个 LUT，一个触发器和相关的相关逻辑。LE 是 ACEX 芯片实现逻辑的最基本结构，如图 1-6 所示。Altera 其他系列，如 APEX 的结构与此基本相同。

2. 查找表结构的 FPGA 逻辑实现方式

我们以前面介绍的图1-4所示的逻辑电路为例，说明查找表结构FPGA逻辑实现

表 1-1　4 输入与门实现逻辑电路和 LUT 的实现方式





a、b、c、d 输入	逻辑输出	地址	RAM 存储器的内容
0000	0	0000	0
0001	0	0001	0
……	0	……	0
1111	1	1111	1

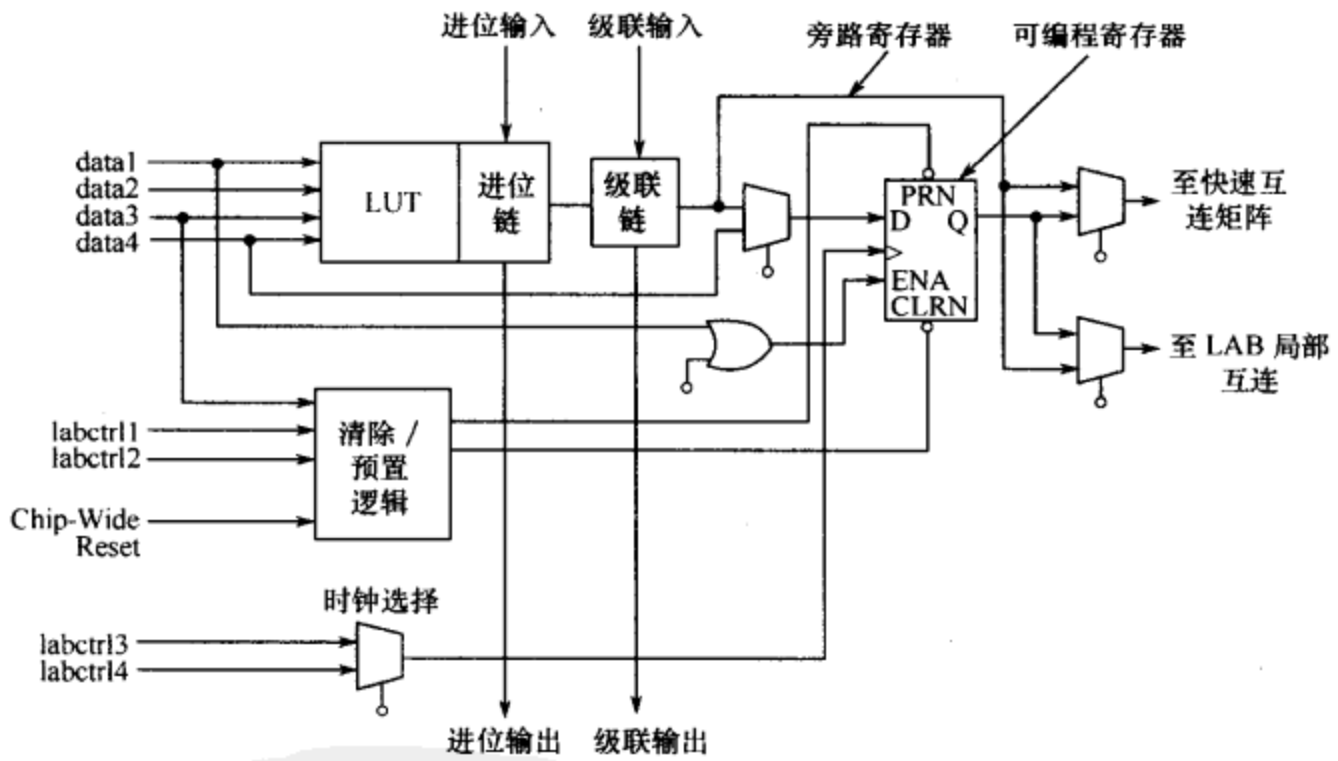


图 1-6　ACEX 系列 FPGA 芯片逻辑单元的结构

的方式。

A、B、C、D 由 FPGA 芯片的管脚输入后进入可编程连线，然后作为地址线连接到 LUT，LUT 中已经事先写入了所有可能的逻辑结果，通过地址查找到相应的数据然后输出，这样组合逻辑就实现了。该电路中 D 触发器是直接利用 LUT 后面 D 触发器来实现。时钟信号 CLK 由 I/O 脚输入后进入芯片内部的时钟专用通道，直接连接到触发器的时钟端。触发器的输出与 I/O 脚相连，把结果输出到芯片管脚。这样 FPGA 就完成了图 1-4所示电路的功能。以上这些步骤都是由软件自动完成的，不需要人为干预。

这个电路是一个很简单的例子，只需要一个 LUT 加上一个触发器就可以完成。对于一个 LUT 无法完成的电路，就需要通过进位逻辑将多个单元相连，这样 FPGA 就可以实现复杂的逻辑。

由于 LUT 主要适合 SRAM 工艺生产，所以目前大部分 FPGA 都是基于 SRAM 工艺的，而 SRAM 工艺的芯片在掉电后信息就会丢失，一定需要外加一片专用配置芯片，在上电的时候，由这个专用配置芯片把数据加载到 FPGA 中，然后 FPGA 就可以正常工作，由于配置时间很短，不会影响系统正常工作。也有少数 FPGA 采用反熔丝或 Flash 工艺，对这种 FPGA，就不需要外加专用的配置芯片。

第三节　Altera 系列CPLD 介绍

在当今迅速变化的市场上，产品的上市周期是取得成功的关键。Altera 公司提供丰富的产品，包括 APEX、FLEX、MAX 和 ACEX 系列。其芯片内部速度可达 200MHz，时钟速度高达 822MHz，并且管脚之间的延迟低于 3.5 ns。另外 Altera 公司系列器件具有系统级的特点，在系统可编程(ISP)，支持 JEDEC 标准 JESD-71。

Altera 公司系列器件针对逐渐改变的设计需求提供了如下的特性：高带宽、低电压 I/O 标准和多重端口电压标准，内部集成有 PLL(锁相环)。

Altera 公司系列器件提供了丰富的芯片封装形式：TQFP、BGA 和 FBGA 等。

Altera 公司的 MAX+PlusII 和最新 QuartusII 开发工具易学易用，功能强大，支持 FLEX、MAX、ACEX 等系列器件。兼容工业标准，提供了 VHDL 和 Verilog-HDL 等文件的直接接口。

下面重点以 Altera 公司的 MAX7000 系列 CPLD 器件为例进行介绍。

一、MAX7000 系列器件简介

MAX 7000 系列以 Alreta 公司的第二代 MAX 结构为基础，以先进的 CMOS 工艺制造。基于 EEPROM 的 MAX 7000 系列可提供 600～5000 个可用的门电路、ISP、引脚间 5ns 的延时以及高达 175.4 MHz 的计数速度。MAX7000 系列器件主要有 MAX7000、MAX7000E 和 MAX7000S 等系列产品。

MAX7000 器件主要包括 EPM7032、EPM7064、EPM7096 等。MAX7000E 器件包括 EPM7128E、EPM7160E、EPM7192E 和 EPM7256E 器件，具有以下增强特性：附加的全局时钟和输出使能控制、增强的互连资源、快速输入寄存器和一个可编程斜率。MAX7000S 是系统可编程器件，包括 EPM7032S、EPM7064S、EPM7128S、EPM7160S、EPM7192S 和 EPM7256S。MAX7000S 包含 MAX7000E 的所有增强特性和带有 128 或更多宏单元的 JTAG BST 电路、ISP 以及开漏输出选择。

MAX7000 系列器件有多种封装形式，如 68 引脚的 PLCC 封装，44 引脚的 PQFP、PLCC、TQFP 封装和 84 引脚的 PLCC 封装等。图 1-7 是 84 引脚的 PLCC 封装芯片的引脚排列图。

MAX7000 系列器件利用 CMOS EEPROM 单元来实现逻辑功能。用户可配置的 MAX7000 结构适用多种独立的组合和时序逻辑功能。在设计开发和调试期间，器件可被快速和高效地反复重新编程，其编程和擦除次数可高达 10 000 次。MAX7000 系列器件包含 32～256 个宏单元，以 16 个宏单元为一组，被称为逻辑阵列块(LAB)。每个宏单元有一个可编程的与门/固定的或门阵列，以及带有独立的可编程时钟、时钟使能、清除和

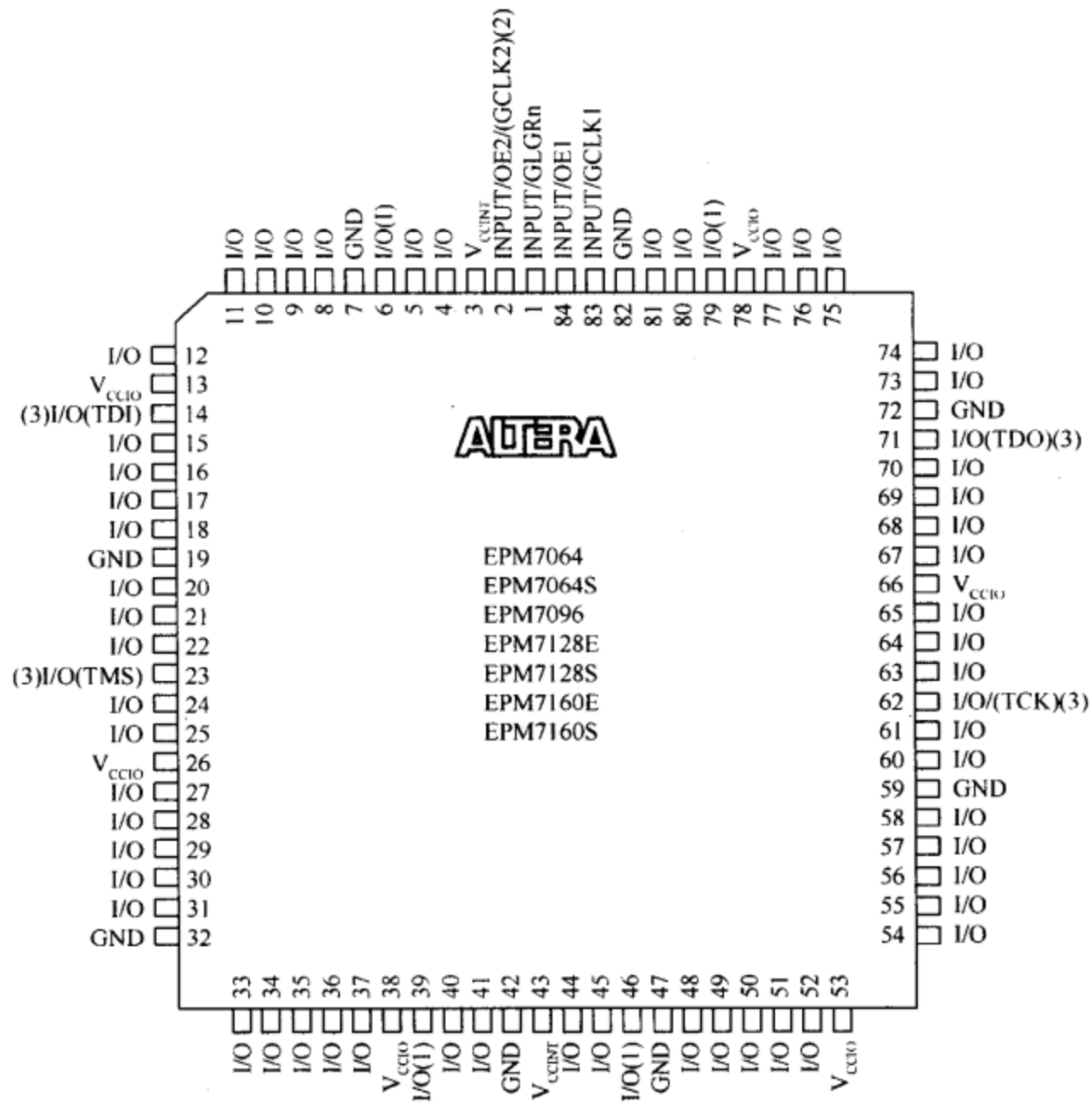


图 1-7　MAX7000 系列 84 引脚的 PLCC 封装芯片的引脚排列

预置功能的一个可配置寄存器。

二、MAX7000 系列器件的结构

1. 逻辑阵列块(LAB)

MAX7000 系列器件是基于高性能并且非常灵活的逻辑阵列块(LAB)连接的体系结构。LAB 包含 16 个宏单元阵列，图 1-8 为 MAX7000E 和 MAX7000S 系列器件结构图。多个 LAB 通过可编程互连阵列(PIA)连接在一起，所有的专用输入端、I/O 脚和宏单元共享一个全局总线。

每个 LAB 输入以下信号：

(1)来自 PIA 的被用做通用逻辑输入的 36 个信号；

(2)全局控制信号，用于寄存器的第二功能；

(3)从 I/O 脚到寄存器的直接输入路径，用于 MAX 7000E 和 MAX 7000S 器件的快速建立时间。

2. 宏单元

MAX 7000 的宏单元可分别设置成时序逻辑或组合逻辑功能。宏单元由 3 个功能模

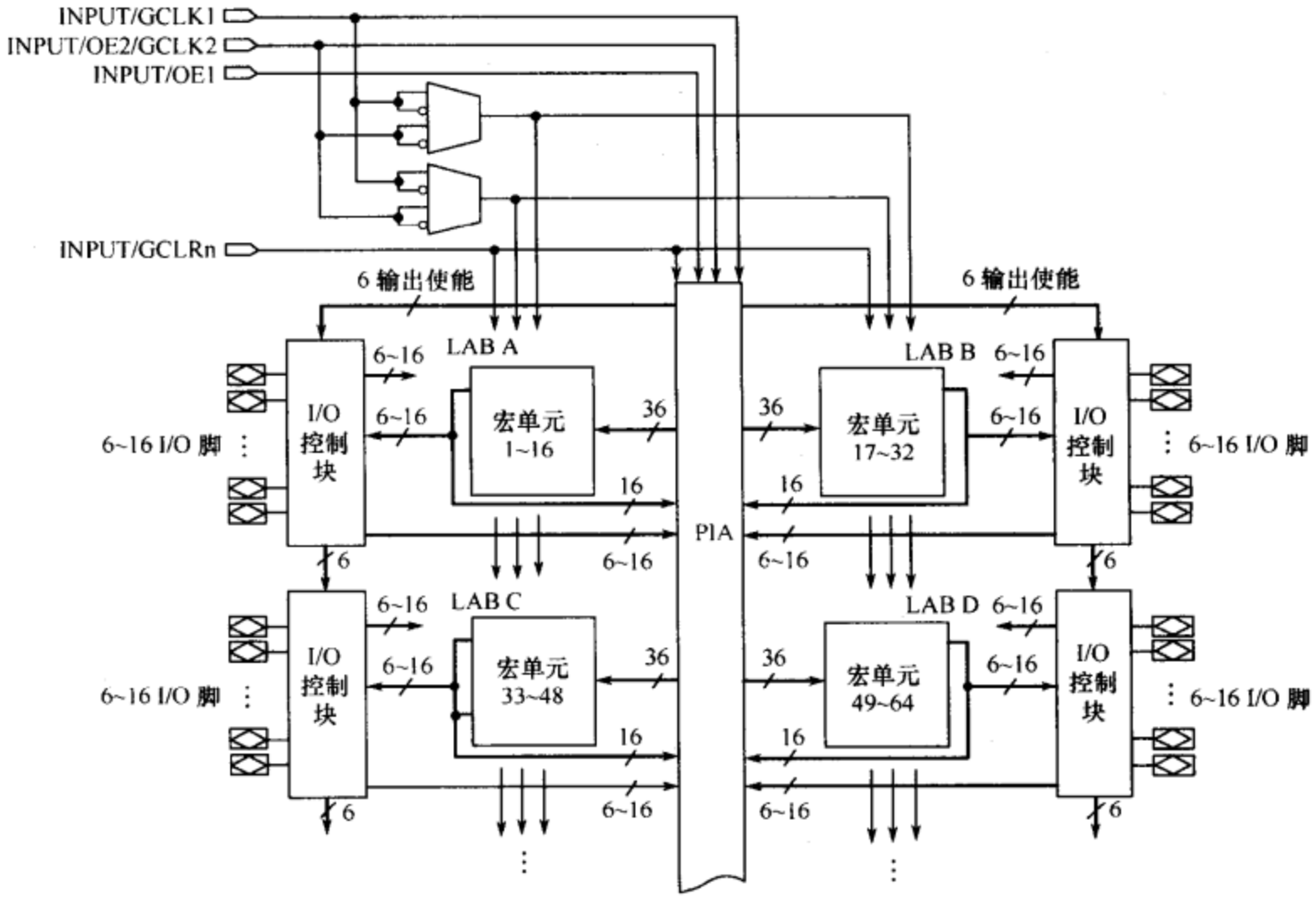


图 1-8　MAX7000E 和 MAX7000S 系列器件结构图

块组成：逻辑阵列、乘积项选择矩阵和可编程寄存器。MAX7000E 和 MAX 7000S 器件的宏单元如图 1-9 所示。

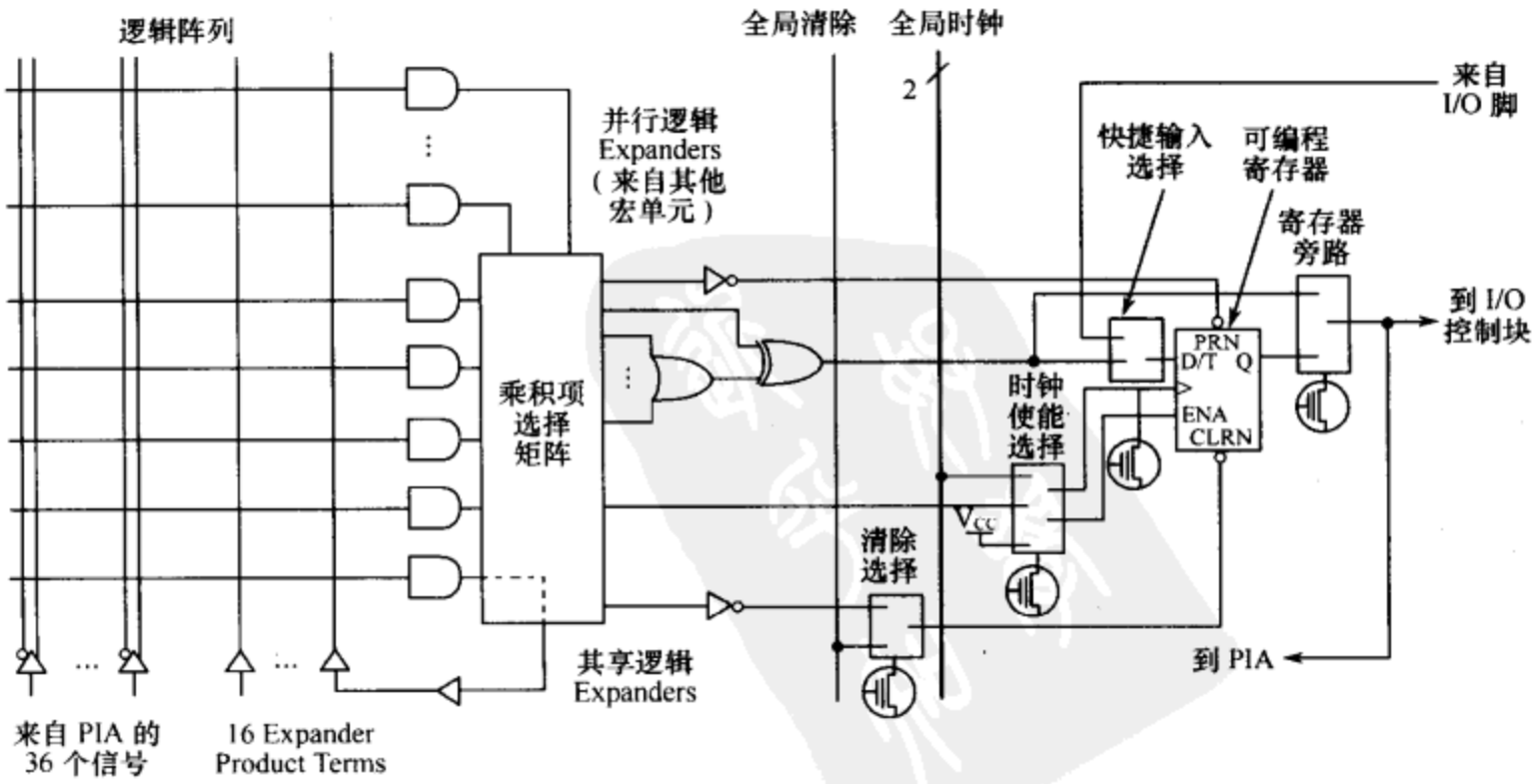


图 1-9　MAX7000E 和 MAX 7000S 器件的宏单元

组合逻辑是在逻辑阵列中实现的。在逻辑阵列中，它为每个宏单元提供 5 个乘积项。乘积项选择矩阵起着分配这些乘积项的作用。

Alreta 开发系统会根据设计的逻辑要求自动地对乘积项分配进行优化。每个宏单元触发器在可编程时钟的控制下可分别编程，来实现 D、T、JK 或 SR 触发器的功能。在组合逻辑操作时，这些触发器就被旁路。在设计入口时，设计者指定所需要的触发器类型；Alreta 开发系统软件再为每个寄存器功能选择最有效的触发器进行工作，以优化资源利用。

每个可编程的寄存器可以在以下 3 种不同的控制时钟下工作：

(1)一个全局时钟信号。这种方式能最快速实现时钟到输出的操作。

(2)一个全局时钟信号由一个高电平有效的时钟使能信号控制。这种方式给每个触发器提供了一个使能信号，但它仍可实现快速的从时钟到输出的操作。

(3)带有一个乘积项的阵列时钟。这种方式下，触发器的时钟信号来自隐藏的宏单元或 I/O 脚。

在 MAX7000E 和 MAX 7000S 器件中，有 2 个全局时钟信号，分别为 GCLKl 和 GCLK2。

每个寄存器也支持异步预置和清除功能。乘积项选择矩阵通过分配乘积项来控制这些功能。尽管由乘积项驱动的对寄存器的预置和清除信号是高电平有效，但通过在逻辑阵列中将此信号反相也可得到低电平。此外，每个寄存器的清除功能可由低电平有效的全局清除引脚(GCLRn)实现。上电时，器件中的每个寄存器的输出都被设置成低电平状态。

所有 MAX7000E 和 MAX 7000S 的 I/O 脚都有一个到达宏单元寄存器的快速输入路径。这个路径允许一个信号绕过 PIA 和组合逻辑，以极短的输入建立时间(2.5 ns)被直接驱动到 D 触发器的输入端。

3. 扩展乘积项

尽管每个宏单元中的 5 个乘积项能实现大部分的逻辑功能，但是，要完成复杂的逻辑功能就需要增加乘积项，所需的逻辑资源由其他宏单元提供。MAX 7000 结构还允许共享和并行扩展乘积项(扩展)，直接为同一个 LAB 中的任意宏单元提供额外的乘积项。这些扩展可以确保以最少的逻辑资源来实现最快速的逻辑合成。

(1) 共享扩展

每个 LAB 含有 16 个共享扩展，可以将其看做不受约束的、带有反馈到逻辑阵列反相输出的单个乘积项的集合。每个共享扩展能被 LAB 中的任意一个或所有宏单元使用和共享，以建立更复杂的逻辑功能。

(2) 并行扩展

并行扩展不使用的乘积项，它被分配给邻近的宏单元来实现快速、复杂的逻辑功能。并行扩展允许多达 20 个乘积项直接输入到宏单元的“或”门，其中的 5 个由宏单元提供，15 个并行扩展由 LAB 中相邻的宏单元提供。

4. 可编程的互连阵列

LAB 之间通过可编程的互连阵列(PIA)形成逻辑通路。这个可编程互连阵列全局总线是一条可编程的路径，它可将任何信号源连接到器件内的任何目的地。所有 MAX7000 系列的专用输入端、I/O 脚和宏单元的输出都输入到 PIA，这就使得 PIA 上包含了贯穿整个器件的所有信号。

5. I/O 控制块

I/O 控制块允许每个 I/O 脚分别设置成输入、输出或双向操作。所有的 I/O 脚都有一个三态缓冲器，它们由一个全局输出使能信号控制，MAX 7000E 和 MAX7000S 器件有 6 个全局输出使能信号，如图 1-10 所示。

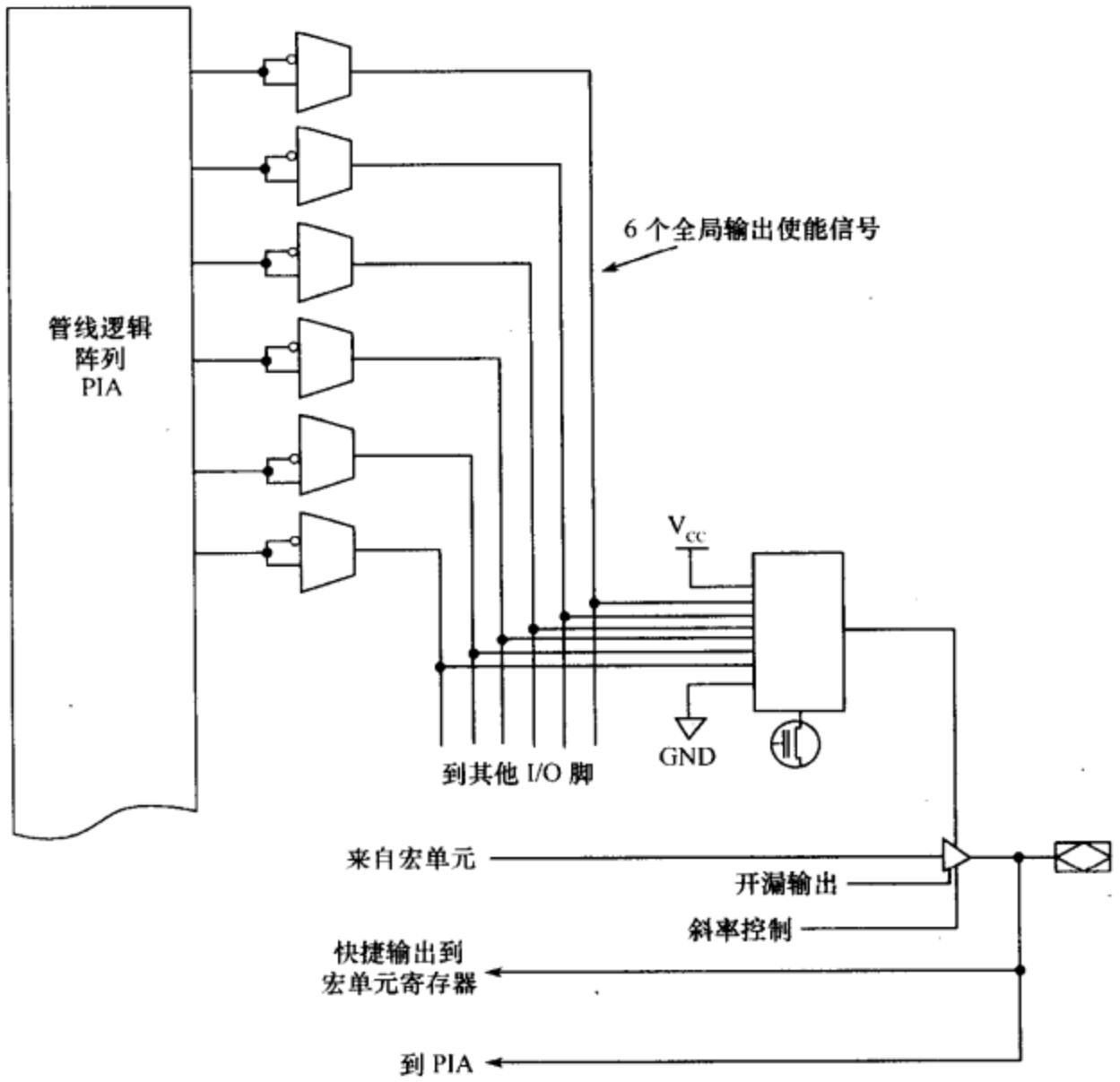


图 1-10　MAX 7000E 和 MAX7000S 器件 I/O 控制块

三、MAX7000 系列器件功能描述

1. 在系统编程 ISP(仅用于 MAX7000S 器件)

ISP(In-System Programming)是“在系统可编程”的意思，指电路板上的空白器件可以编程写入最终用户代码，而不需要从电路板上取下器件，已经编程的器件也可以用 ISP 方式擦除或再编程。

具有 ISP 功能的芯片不但可以防止多引脚封装形式下由于操作器件而出现引脚损坏的情况，而且还可使系统在推向市场后仍能对器件进行重新编程，例如通过软件或调制解调器实现产品的升级等。

MAX7000S 器件通过一个工业标准 4 脚的 JTAG 接口(IEEE Std. 1149. 1-1990)，可以实现在系统编程(ISP)。JTAG 信号线的功能如表 1-2 所示。

表 1-2　JTAG 信号线及功能

信 号 名	方向	功　　能
TCK	输入	同步时钟，上升沿锁存 TMS 和 TD，下降沿更新 TDO
TMS	输入	模式控制信号
TDI	输入	边界扫描的输入数据
TDO	输出	JTAG 命令输入，边界扫描的输出数据

重点提示　(1)对于 EEPROM(或 Flash)工艺的 CPLD(如 Altera 的 MAX7000S，Xilinx 的 XC9500 系列) 厂家提供编程电缆，电缆一端装在计算机上，另一端接在 PCB 板上的插头上，这就是所谓的在线可编程(ISP)，编程时不需要编程器或任何其他编程硬件。早期的可编程逻辑器件是不支持 ISP 的，它们需要用编程器烧写。目前的 CPLD 大都可以用 ISP 在线编程，也可用编程器编程。这种 CPLD 器件可以加密，并且很难解密。

(2)对于基于查找表技术、采用 SRAM 工艺的 FPGA，由于 SRAM 工艺的特点，掉电后数据会消失，因此调试期间可以用下载电缆配置 FOGA 器件，调试完成后，需要将数据固化在一个专用的 EEPROM 中(用通用编程器烧写)，上电时，由这片配置 EEPROM 先对 FPGA 加载数据，十几个毫秒后，FPGA 即可正常工作。但 SRAM 工艺的 FPGA 一般不可以加密。

(3)还有一种反熔丝技术的 FPGA，不能重复擦写，所以初期开发过程比较麻烦，费用也比较高昂。但反熔丝技术也有许多优点：布线能力更强，系统速度更快，功耗更低，同时抗辐射能力强，耐高低温，可以加密，所以在一些有特殊要求的领域中运用较多，如军事及航空航天领域。

2. 可编程的速度/功率控制

MAX7000 系列器件提供了一个节电模式，支持在用户定义的信号通路上或对整个器件进行低功耗操作。因为大部分逻辑应用只需要所有门电路中的一小部分在最高频率下工作，所以这个特性可以使总的功耗减少 50%或者更多。

设计者可以将 MAX 7000 系列器件中每个独立的宏单元编程为高速(打开 Turbo Bit 选项)或低功耗(关闭 Turbo Bit 选项)操作。结果，设计中对速度敏感的路径就能以高速运行，而其余的路径可以工作在低功耗模式下。

3. MultiVolt I/O 界面

MAX7000 系列器件除 44 脚器件外，均支持 MultiVolt I/O 接口特性，允许 MAX7000 系列与不同电压的系统相连。所有封装的 5.0V 器件可设置 3.3V 或 5.0V I/O脚操作。这些器件有一组用于内部操作的 Vcc 引脚和输入缓冲引脚(V_{CCINT})，另外还有一组 I/O 输出的驱动引脚(V_{CCIO})。

V_{CCINT} 引脚必须一直与 5.0 V 电源相连。当 V_{CCINT} 的电平为 5.0V 时，输入电压阈值为 TTL 电平，因此完全兼容 3.3V 和 5.0V 的输入。V_{CCIO}引脚可与 3.3 V 或 5.0 V 的电源相连，取决于输出的需要。当 V_{CCIO}引脚与 5.0V 的电源相连时，输出电平兼容 5.0V 的系统；当它与 3.3 V 相连时，输出的高电平为 3.3V，因此它可兼容 3.3V 或 5.0V 的系统。

4. 开漏输出选项(仅用于 MAX7000S 器件)

MAX7000S 为每个 I/O 脚提供了一个开漏输出的选项，器件可利用这个开漏输出采

提供系统级的控制信号，如中断和写使能信号，可由几个器件进行选择控制。另外，它还提供一个额外的“线或”平面。

通过使用外部 5.0 V 的上拉电阻，MAX7000S 器件的输出引脚可以设置满足 5.0 V CMOS 输入电压要求。若 V_{CCIO} 为 3.3 V，则选择开漏输出将会关闭输出上拉三极管，利用外部上拉电阻将输出拉高以满足 5.0V 的 CMOS 输入电压。若 V_{CCIO} 为 5.0V，因为当引脚输出超过大约 3.8V 时上拉三极管已经关闭，外部上拉电阻可直接将输出拉高来满足 5.0 V CMOS 输入电压的要求，所以不必选择开漏输出。

第四节　Xilinx 系列CPLD 介绍

Xilinx CPLD 包括 CoolRunner-II、CoolRunner XPLA3、XC9500XV、XC9500XL 和 XC9500 系列等多个产品，其中，XC9500 系列 CPLD 应用最为广泛，XC9500 系列提供了范围最广的供电电压、器件密度和温度。与竞争器件相比，整个 XC9500 系列器件以较低的价格提供了较大的逻辑容量。更大的逻辑容量使设计人员可以降低总体成本。

一、XC9500 系列器件简介

XC9500 系列 CPLD 主要器件型号有 XC9536、XC9572、XC95108、XC95144、XC95216、XC95288 等。其主要区别如表 1-3 所示。所有器件都可实现在系统编程，并且编程/擦除周期最少为 10 000 次。该系列的所有器件都支持 IEEE 1149.1(JTAG)边界扫描技术。

表 1-3　XC9500 系列产品型号

器件型号	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
宏单元/个	36	72	108	144	216	288
可用的门/个	800	1600	2400	3200	4800	6400
寄存器/个	36	72	108	144	216	288

重点提示　边界扫描技术

边界扫描测试技术主要解决芯片的测试问题。20 世纪 80 年代后期，对电路板和芯片的测试出现了困难。以往，在生产过程中对电路板的检验是由人工或测试设备进行的，但随着集成电路密度的提高，集成电路的引脚也变得越来越密，测试变得很困难。例如，TQFP 封装器件，管脚的间距仅有 0.6mm，这样小的空间内几乎放不下一根探针。同时，由于国际技术的交流和降低产品成本的需要，也要求为集成电路和电路板的测试制订统一的规范。

边界扫描技术正是在这种背景下产生的。IEEE 1149.1 协议是由 IEEE 组织联合测试行动组在 20 世纪 80 年代提出的边界扫描测试技术标准，用来解决高密度引线器件和高密度电路板上的元件的测试问题。

标准的边界扫描测试只需要四根信号线，能够对电路板上所有支持边界扫描的芯片内部逻辑和边界管脚进行测试。应用边界扫描技术能增强芯片、电路板甚至系统的可测试性。

二、XC9500 系列器件的结构

XC9500 系列器件包括多个功能模块(FB)和 I/O 模块(IOB),并通过 FastCONNECT(快速连接)开关矩阵实现内部连接。IOB 提供器件输入和输出的缓冲。每个 FB 提供 36 个输入和 18 个输出的可编程逻辑。FastCONNECT 开关矩阵将所有 FB 输出和输入信号连接到 FB 输入。对于每一个 FB,12～18 个输出(取决于封装引脚数)和相关的输出使能信号直接驱动 IOB,XC9500 系列 CPLD 结构如图 1-11 所示。

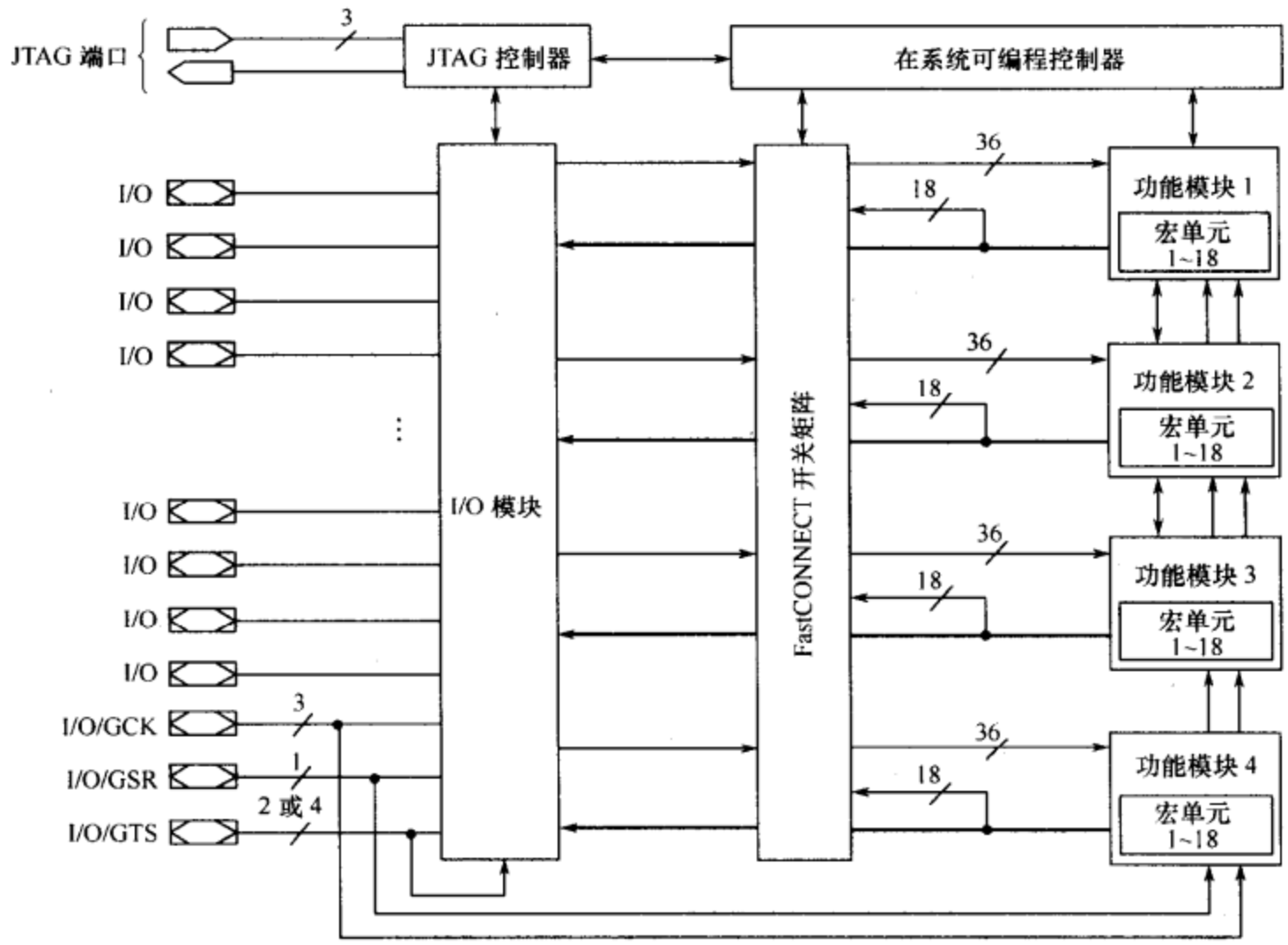


图 1-11　XC9500 系列 CPLD 结构图

1. 功能模块(FB)

功能模块(FB)框图如图 1-12 所示。

每个功能模块都包含 18 个独立的宏单元,都可通过组合或寄存器实现功能。FB 还可接收全局时钟、输出使能和设置/复位信号。FB 产生 18 个输出,用于驱动 FastCONNECT 开关矩阵。这 18 个输出和它们对应的输出使能信号也驱动 IOB。

FB 内的逻辑通过乘积表达式的集合来实现。36 个输入提供 72 个“真”和“补”信号,输入到可编程“与”阵列来构成 90 个乘积项。这些乘积项的任意一个或全部都可通过乘积项定位器定位到每个宏单元。

2. 宏单元

XC9500 系列宏单元可以单独配置为通过组合或寄存的功能。宏单元和相关的 FB 逻辑如图 1-13 所示。

“与”阵列中的 5 个直接乘积项用于主要的数据输入(输入到“或”和“异或”门),以实

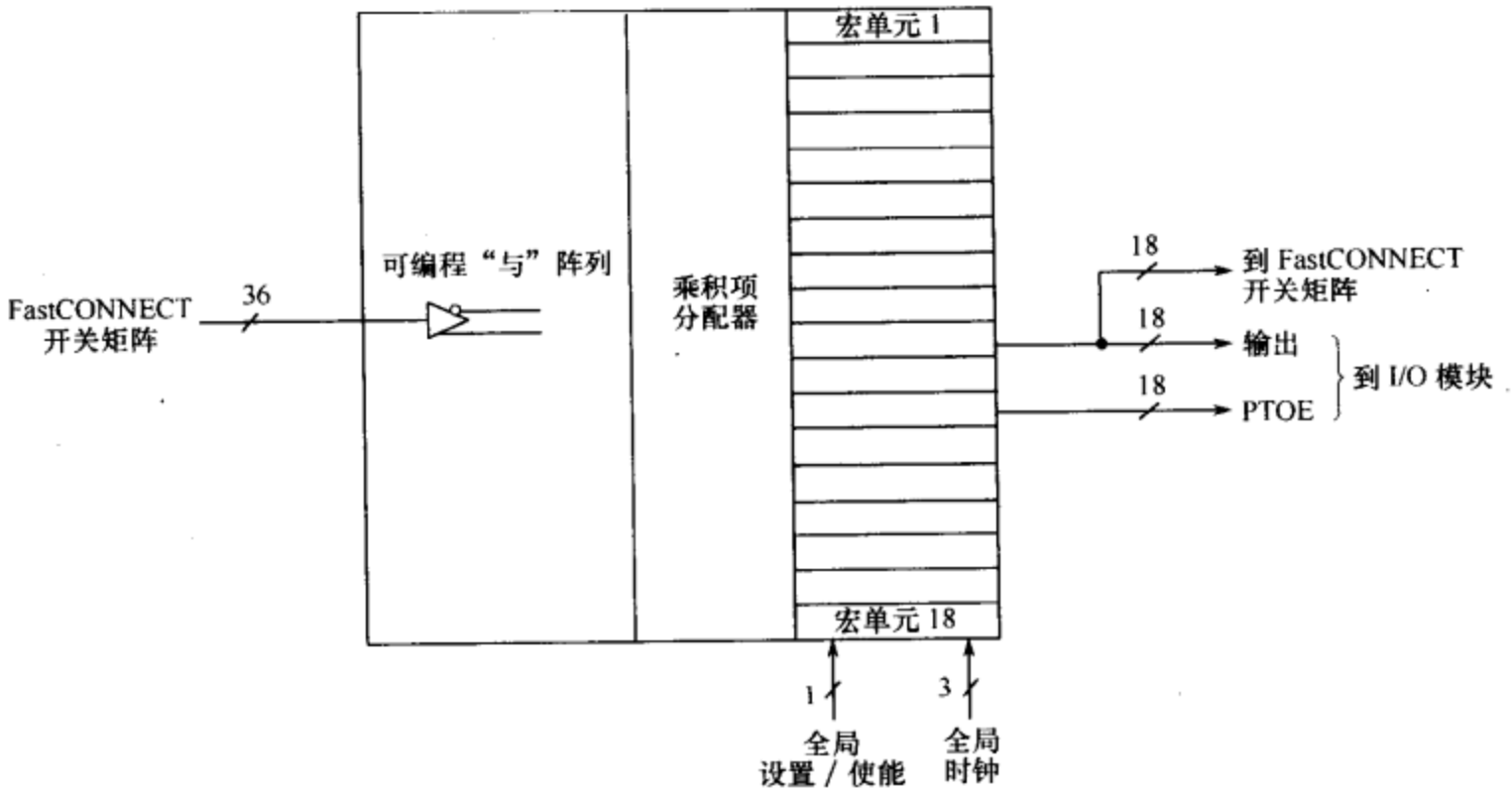


图 1-12　功能模块框图

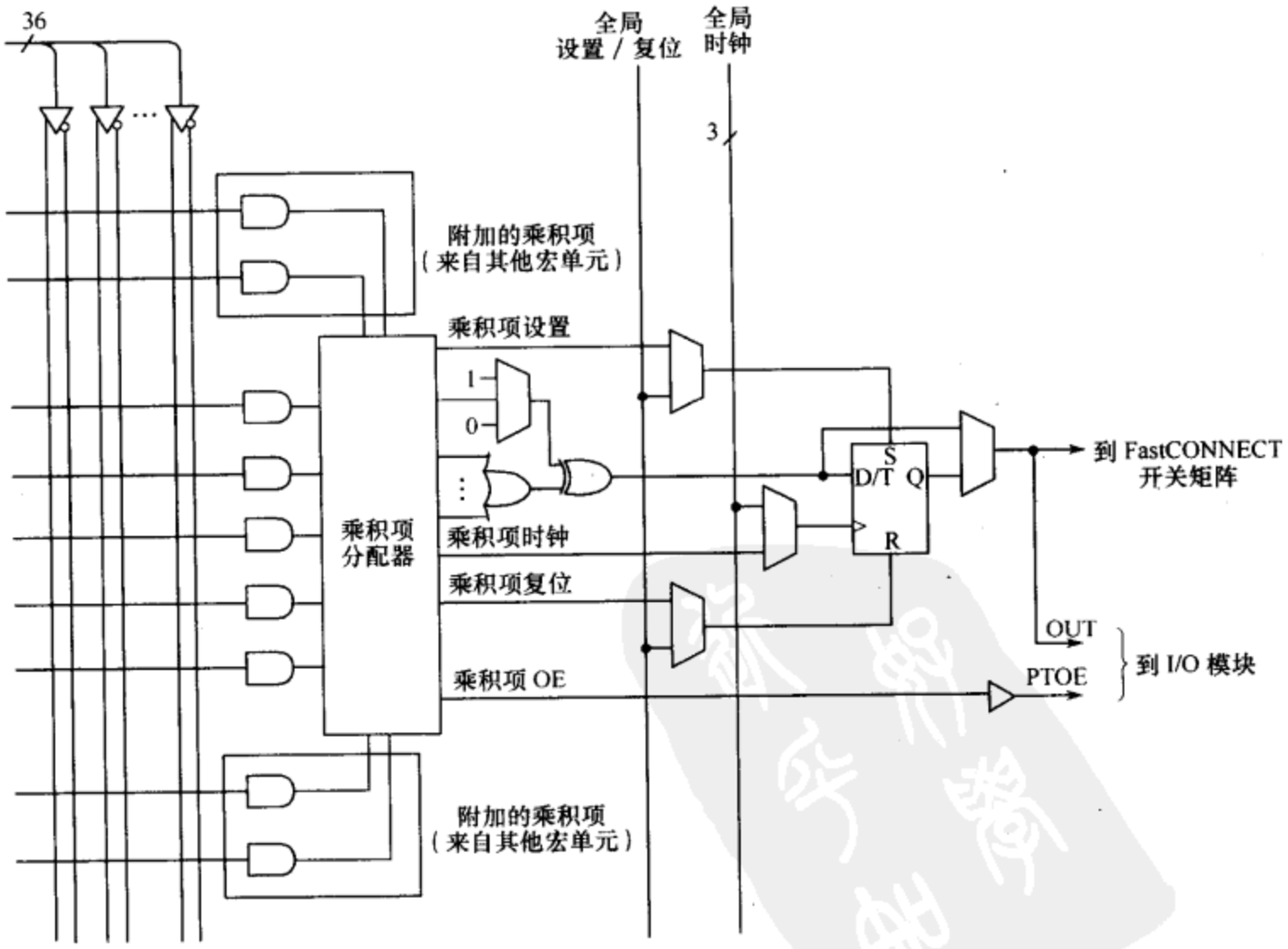


图 1-13　XC9500 功能模块的宏单元

现组合功能，或者作为控制输入信号（包括时钟、设置/复位和输出使能信号）。与每个宏单元相关的乘积项分配器用于选择如何使用这 5 个直接乘积项。

宏单元寄存器可配置为 D 型或 T 型触发器，或者被旁路，以实现组合操作。每个寄

存器都支持异步设置和复位操作。在上电时，所有用户寄存器都初始化为用户预定义的状态(如果未定义，则默认为 0)。

所有的全局控制信号都可用于单个宏单元，包括时钟、设置/复位和输出使能信号。如图 1-14 所示，宏单元寄存器时钟来自 3 个全局时钟之一或乘积项时钟。GCK 引脚的“真”和“补”极性可在器件内部使用。它还提供 GSR 输入，使用户寄存器可以设置成用户定义的状态。

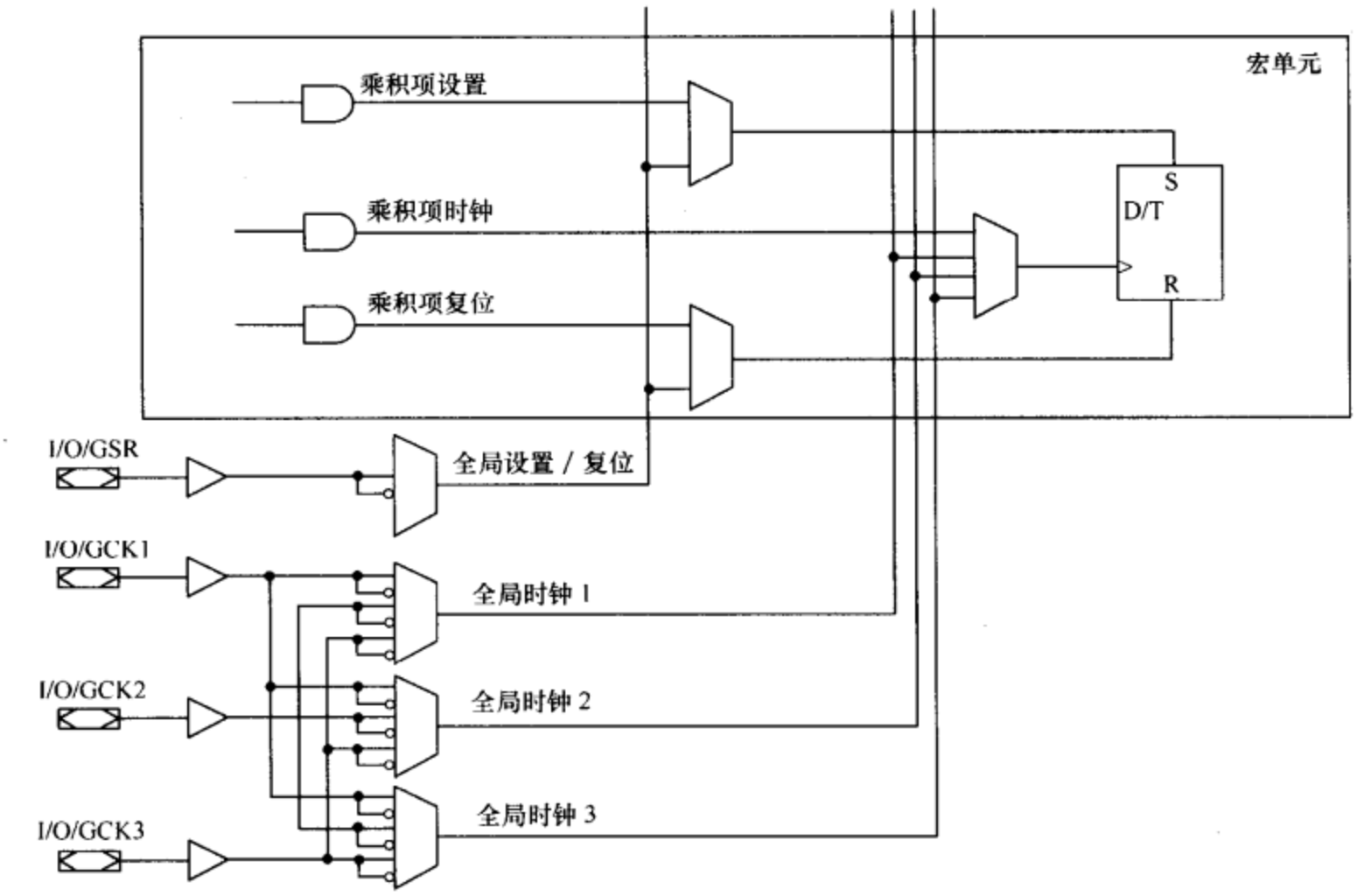


图 1-14　宏单元的时钟和设置复位

3. 乘积项分配器

乘积项分配器控制如何将 5 个直接乘积项分配到每个宏单元。例如，所有 5 个直接乘积项可驱动如图 1-15 所示的“或”功能。

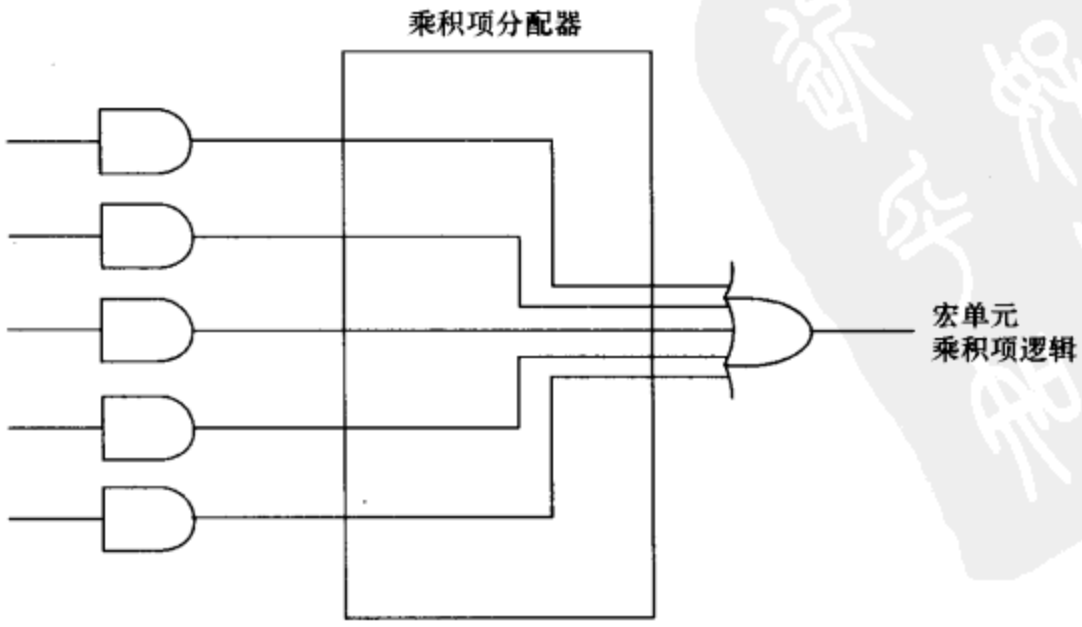


图 1-15　使用直接乘积项的宏单元逻辑

乘积项分配器可在 FB 内重新分配其他乘积项，这样可提高宏单元在 5 个直接乘积项之外的逻辑性能。在 FB 内，任何一个需要额外乘积项的宏单元都可访问其他宏单元中未使用的乘积项。一个宏单元最多可有 15 个乘积项，但单个宏单元的延迟时间增加很少。

4. FastCONNECT 开关矩阵

FastCONNECT(快速连接)开关矩阵将信号连接到 FB 输入端，如图 1-16 所示。所有 IOB 输出(对应于用户引脚输入)和所有 FB 输出驱动 FastCONNECT 矩阵。可选择上述任意输出(一个 FB 的最大扇入限值为 36)通过用户编程以相同的延迟来驱动每个 FB。

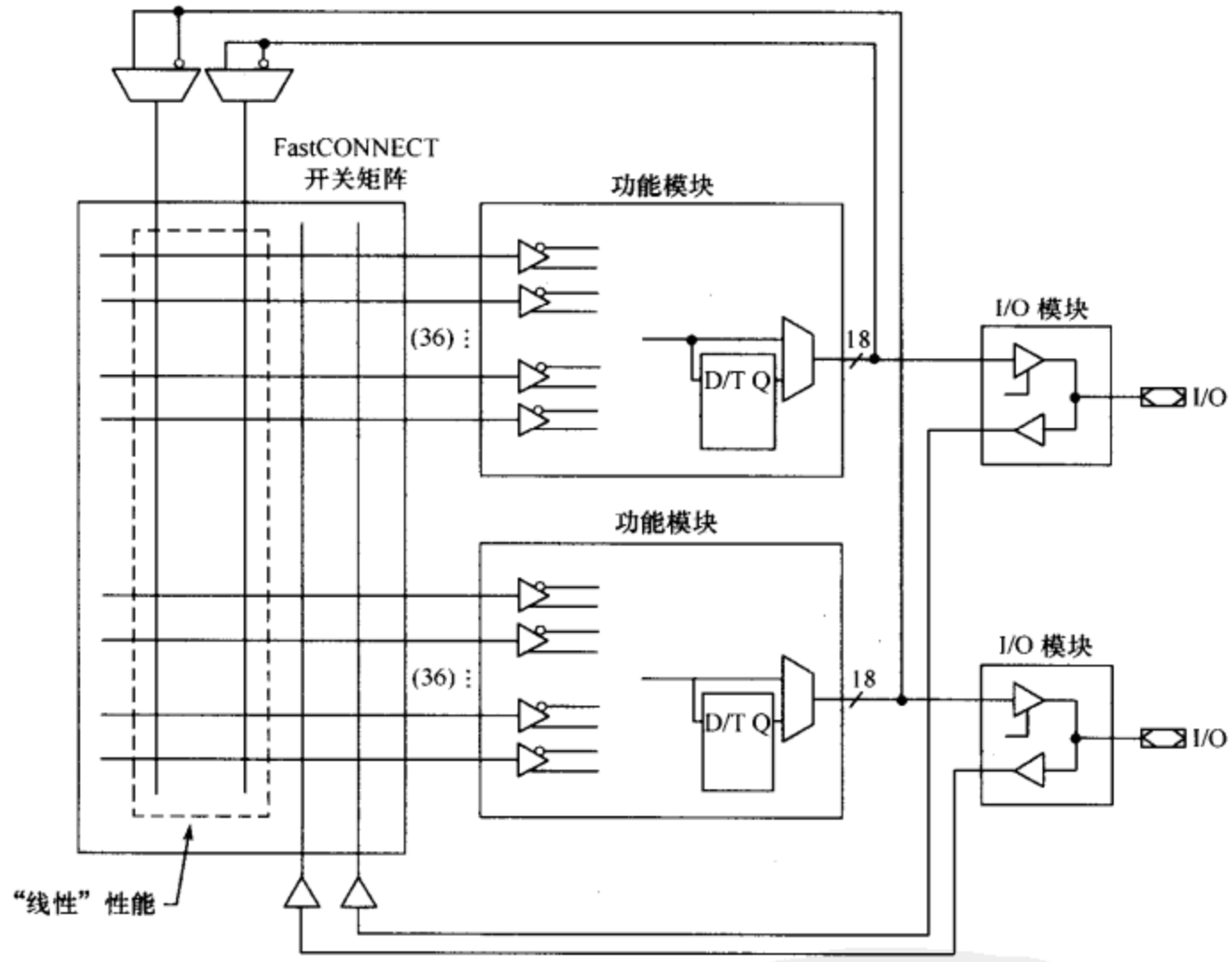


图 1-16　FastCONNECT 开关矩阵

FastCONNECT 开关矩阵可以在驱动目标 FB 之前，将多个内部连接组合成单个"线与"输出。这提供了额外的逻辑能力，并且在没有任何额外时序延迟的情况下，提高了目标 FB 的有效逻辑扇入。

5. I/O 模块

内部逻辑和器件用户 I/O 引脚之间通过 I/O 模块(IOB)接口，每个 IOB 都包括一个输入缓冲器、输出驱动器、输出使能选择和用户可编程接地控制，如图 1-17 所示。

输入缓冲器兼容标准 5 V CMOS、5 V TTL 和 3.3 V 信号电平。输入缓冲器使用内部 5V 电源(V_{CCINT})，以确保内部门坎保持恒定，不随 V_{CCIO} 的电压变化。输出使能可由下列 4 个选项之一产生：一个宏单元的乘积项信号、任意全局输出使能 OE 信号的任意一个(总是为"1"或总是为"0")。输出使能信号对 144 个宏单元的器件有 2 个全局输出使能，对 180 个或更多宏单元的器件有 4 个全局输出使能。任意一个全局 3 态控制(GTS)引脚

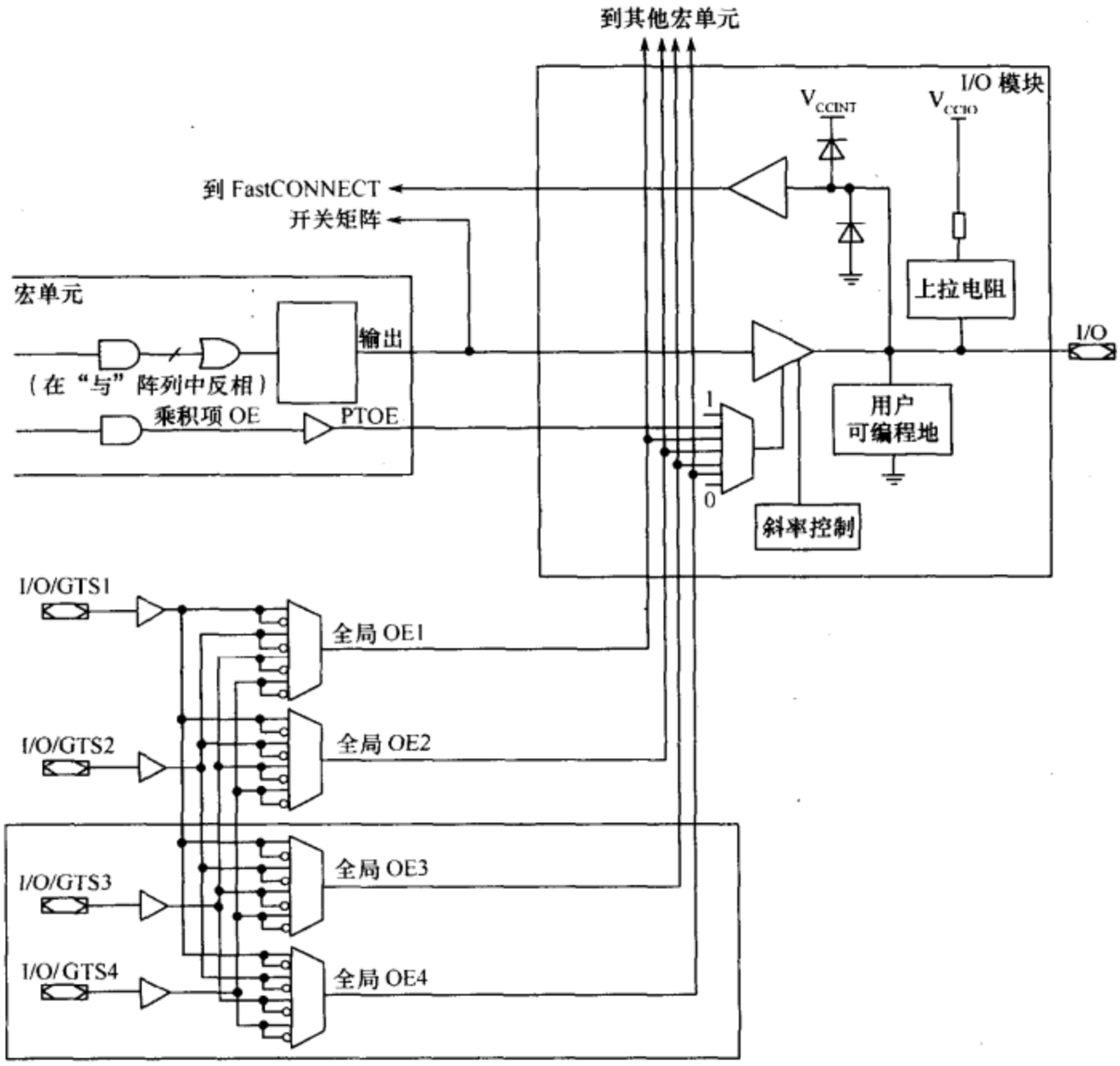


图 1-17　I/O 模块和输出使能

的两个极性可以在全局内被利用。

每个 IOB 都提供用户可编程接地脚性能，这使器件的 I/O 引脚可以配置为额外的地。通过将可编程接地脚和外部地相连接，众多输出同时开/关所产生的系统噪声将因此减小。

当器件处于不正常操作时，连接到每个器件 I/O 引脚的控制上拉电阻可防止引脚处于悬浮状态。该电阻在器件编程模式、上电和擦除芯片时有效；在正常操作时被禁止。

输出驱动器具有 24mA 的驱动能力。通过将器件输出电源(V_{CCIO})连接到 5 V 或 3.3 V，器件内的所有输出驱动器可配置为 5 V TTL 电平或 3.3V 电平。

三、XC9500 系列器件功能描述

1. 在系统编程 ISP

XC9500 器件通过标准 4 脚 JTAG 协议实现在系统编程，在系统编程时，由 IOB 电阻上拉为高电平。如果在该时间内一个特殊信号必须保持低电平，那么可以在该引脚增加一个下拉电阻。所有 XC9500 CPLD 都可承受最少 10 000 次在系统编程/擦除周期。

2. 保密设计

XC9500 器件集成了先进的数据保密特性，可完全防止数据在未经授权的情况下被读出，或因为疏忽对器件进行擦除/重新编程。

3. 低功耗模式

XC9500 器件的每个宏单元都提供低功耗模式。该特性使器件可以显著地降低功耗。用户可将任意一个宏单元编程为低功耗模式。对性能有严格要求的应用部分可保持标准功耗模式，而其他部分可以编程为低功耗模式，以降低整个系统的功率损耗。宏单元编程为低功耗模式时，会增加引脚间组合延迟时间和寄存器建立时间。乘积项时钟到输出和乘积项输出使能的延迟不受功耗模式设定的影响。

4. 上电特性

XC9500 器件在所有操作条件下都可良好工作。在上电时，XC9500 的内部电路使器件保持静止状态，直到 V_{CCINT} 处于安全的电压（约为 3.8 V）为止。在此期间，所有器件引脚和 JTAG 引脚都被禁止。当电源电压到达安全值时，所有用户寄存器开始初始化，器件立即进入可操作状态。

如果器件处于擦除状态，器件输出保持禁止，IOB 上拉电阻使能。JTAG 引脚使能，以允许对器件编程。如果器件已被编程，则器件的输入和输出按照它们的配置状态执行正常操作。JTAG 引脚使能，以允许对器件进行擦除或边界扫描测试。

第五节　可编程逻辑器件的开发

PLD 的开发是指利用开发系统的软件和硬件对 PLD 进行设计和编程的过程。

开发系统的软件是指 PLD 专用的硬件描述语言和相应的开发软件。硬件描述语言最有代表性的是 Verilog HDL、VHDL 和 ABEL HDL 等。VHDL 是最早标准化的 HDL，语法丰富且严谨。Verilog HDL 具有类似于 C 语言的语法体系，库文件丰富，十分便于具有一些 C 语言基础的人学习。本书将采用简单易用的 Verilog HDL 来编写程序和描述数字系统。用 Verilog-HDL 编写的程序是否正确，需要经开发软件进行编译和调试，在开发软件中，应用最多是的 Altera 公司的 MAX＋PLUSⅡ、QuartusII 集成开发软件和 Xilinx 公司的 ISE WebPACK 集成开发软件。

重点提示　早期的 PLD 多使用汇编型软件，如 PALASM、FM 等。这类软件十分小巧，可对 GAL/PAL 可编程逻辑芯片进行方便地编程，使用方法十分简单。但此类软件不具备自动化简功能，只能用化简后的与或逻辑表达式进行设计输入，而且对不同类型的 PLD 兼容性较差。

开发系统的硬件部分包括计算机和编程器。编程器是对 PLD 进行写入和擦除的专用装置，能提供写入或擦除操作所需要的电源电压和控制信号，并通过并行接口从计算机接收编程数据，最终写入 PLD 中。需要说明的是，现在大多数 CPLD 器件采用了 ISP 技术，编程时不使用编程器，只需要通过计算机接口和编程电缆，直接在目标系统或印刷线路板上进行编程。因此，ISP 技术有利于提高系统的可靠性，便于系统板的调试和维修。

一、可编程逻辑器件的设计过程

可编程逻辑器件的设计流程如图 1-18 所示，它主要包括设计准备、设计输入、器件选

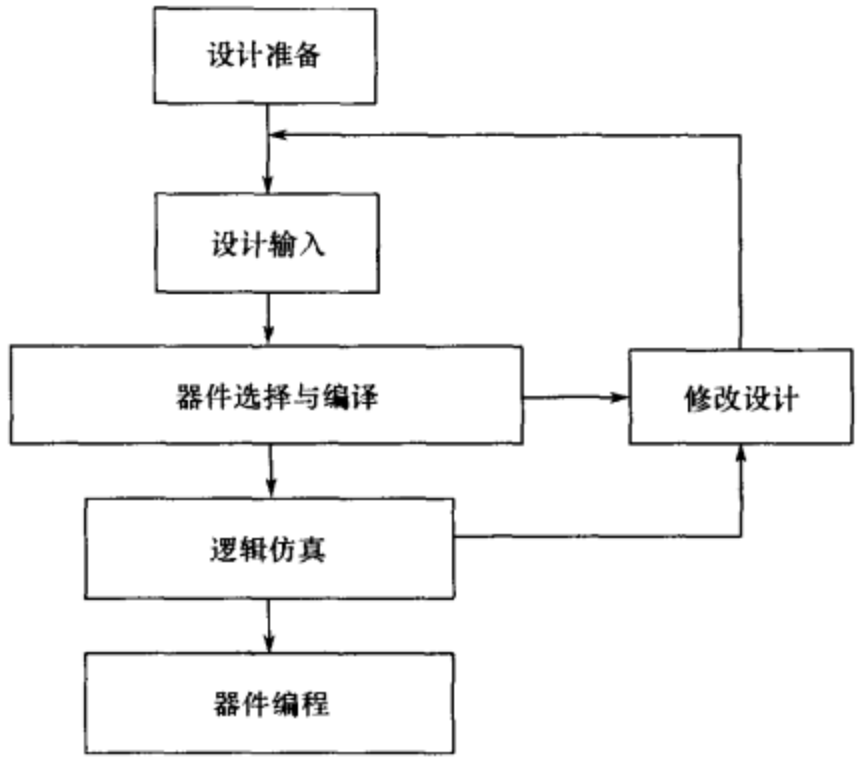


图 1-18　PLD 设计流程

择与编译、逻辑仿真和器件编程等几个步骤。

1. 设计准备

采用有效的设计方案是 PLD 设计成功的关键，因此在设计输入之前首先要考虑两个问题：一是选择系统方案，进行抽象的逻辑设计；二是选择合适的器件，满足设计的要求。

对于 GAL 等低密度 PLD，一般可以进行书面逻辑设计，将电路的逻辑功能直接用逻辑方程、真值表状态图或原理图等方式进行描述，然后根据整个电路输入、输出端数以及所需要的资源（门、触发器数目）选择能满足设计要求的器件系列和型号。器件的选择除了应考虑器件的引脚数、资源外，还要考虑其速度、功耗以及结构特点。

对于 CPLD 等高密度 PLD，系统方案的选择通常采用“自顶向下”的设计方法。首先在顶层进行功能框图的划分和结构设计，然后再逐级设计低层的结构。一般描述系统总功能的模块放在最上层称为顶层设计；描述系统某一部分功能的模块放在下层，称为底层设计。底层模块还可以再向下分层。这种“自顶向下”和分层次的设计方法使整个系统设计变得简洁和方便，并且有利于提高设计的成功率。目前系统方案的设计工作和器件的选择都可以在计算机上完成，设计者可以采用国际标准的两种硬件描述语言 VHDL 或 Verilog HDL 对系统级进行功能描述，并选用各种不同的芯片进行平衡、比较，选择最佳结果。

2. 设计输入

设计者将所设计的系统或电路以开发软件要求的某种形式表示出来，并送入计算机的过程称为设计输入。它通常有原理图输入、硬件描述语言输入和波形输入等多种方式。

原理图输入是一种最直接的输入方式，它大多数用于对系统或电路结构很熟悉的场合，但系统较大时，这种方法输入效率较低。

硬件描述语言是用文本方式描述设计，目前应用最为广泛的硬件描述语言是 VHDL 和 Verilog HDL，它们有许多突出的优点：如输入效率高，便于组织大规模系统的设计，具有很强的逻辑描述和仿真功能，而且，在不同的设计输入库之间转换也非常方便。

3. 选择器件与编译

设计输入完成后，可以选择相应的器件，并对各引脚进行分配，然后，对程序进行编译。在编译过程中，集成开发软件对设计输入文件进行逻辑化简、综合和优化，最后产生编程用的编程文件。

编程文件是可供器件编程使用的数据文件。对于阵列型 PLD 来说，是产生熔丝图文件(简称 JED)，它是电子器件工程联合会制定的标准格式；对于 FPGA 来说，是生成位流数据文件。

在编译过程中出现错误，集成开发软件将给出提示信息，用户应根据提示更正错误，直至编译完全正确为止。

4. 逻辑仿真

为了验证程序的可靠性，用户可以进行逻辑仿真，如果逻辑仿真时达不到预期的目的，需要对程序进行修改，然后再编译、再仿真，直至达到设计的要求。常用的仿真软件有 ModelSim 等。

5. 器件编程

编程是指将编程数据放到具体的 PLD 中去。对阵列型 PLD 来说，是将 JED 文件“下载”到 PLD 中去；对 FPGA 来说，是将位流数据文件“配置”到器件中去。

器件编程需要满足一定的条件，如编程电压、编程时序和编程算法等。普通的 PLD 和一次性编程的 FPGA 需要专用的编程器完成器件的编程工作。基于 SRAM 的 FPGA 可以由 EPROM 或微处理器进行配置。ISP 在系统编程器件(如 MAX7000S 和 XC9500 等)则不需要专门的编程器，只要一根下载编程电缆就可以了。

二、可编程逻辑器件设计举例

现在，让我们通过一个简单的例子来看两种设计过程，见图 1-19 所示的电路。

其逻辑表达式为：

$F=(\sim A)^{*}(B+C)$

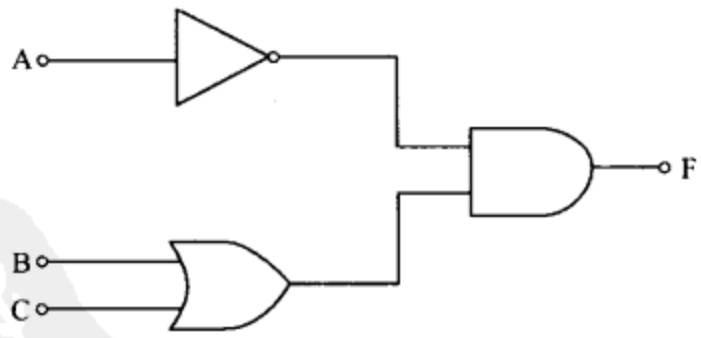


图 1-19　一个简单的逻辑电路

就是这样一个简单的逻辑电路，如果用传统的硬件电路实现方法，要用到 74LS04(反相器)、74LS08(或门)和 74LS32(与门)三个功能集成电路，做如图 1-20 的连接。从图中可以看出，仅仅为了实现一个简单的逻辑关系，就需要有 3 个集成电路，而且还有许多个门白白被浪费掉。

如果用可编程逻辑器件来实现的话，只要用一片集成电路就可以实现了，例如，可以用 Alreta 公司的可编程逻辑器件 MAX7000S 系列中的 EPM7032S。用 EPM7032S 实现该逻辑功能时，只要将逻辑表达式按规定的语法进行描述，经过仿真、编译等过程，最后下载到可编程逻辑器件中，就可以完成所设计的逻辑功能。逻辑电路的设计越复杂，可编程逻辑器件就越能显示出其优越性。不仅如此，有许多逻辑功能用可编程逻辑器件可以很容易地实现，而要用普通的功能特定的集成电路就很难实现。

对于本例的逻辑功能，用 Verilog HDL 可描述如下：

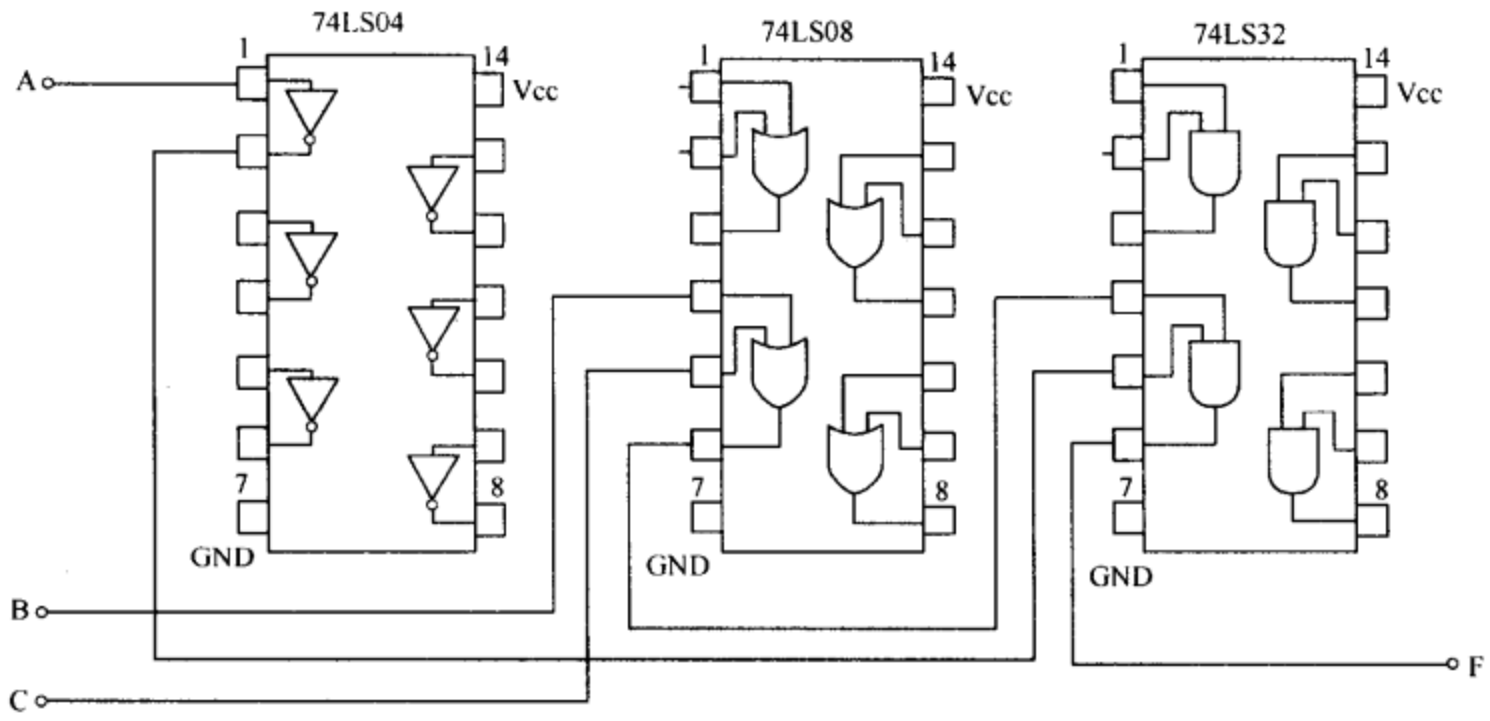


图 1-20　逻辑电路的硬件连接方法

```
module AND_G2 ( A, B, C, F );
input A, B, C;
output F;
assign F = ~A & (B | C);
endmodule
```

将上述的 Verilog HDL 经过仿真确认逻辑关系正确后，就可以用 MAX+plus II 集成开发软件进行编译，然后下载到可编程逻辑器件 EPM7032S 中。这种情况下，一片可编程逻辑器件 EPM7032S 就具有了 74LS04(反相器)、74LS08(或门)和 74LS32 三片集成电路所构建的电路功能，如图 1-21 所示。图中，逻辑门之间的连线是在芯片内部自动完成的。由此可见，它与传统的逻辑电路设计相比，设计过程很简单。

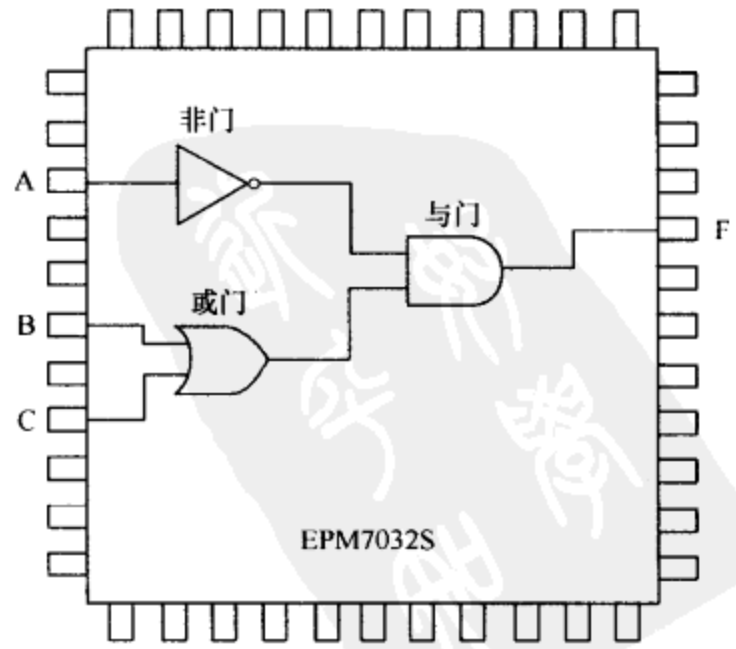


图 1-21　经过编程后的可编程逻辑器件 EPM7032S

第二章　CPLD 实验仪介绍

CPLD 和单片机一样，是一门实践性很强的技术，只有多练习、多实际操作，通过做一系列的实验，才能比较容易地领会 CPLD 中那些枯燥、难懂的专业术语，在这里，主要介绍几种性价比较好的实验仪，以适合初学者选购和使用。

第一节　DP-MCU/Altera 实验仪

DP-MCU/Altera 实验仪是广州致远电子有限公司设计的 DP 系列下载实验仪之一，是一种功能强大的单片机、Altera 系列 CPLD 应用技术学习和调试的开发工具，配合 Keil 软件可进行单片机实验，配合 MAX＋plusII 软件可进行 Altera 公司 CPLD 实验。实验仪电路布局如图 2-1 所示。

一、实验仪主要器件

DP-MCU/Altera 实验仪主要器件如表 2-1 所示。

表 2-1　DP-MCU/Altera 实验仪主要器件

标 号	型 号	功　能	标 号	型 号	功　能
U1	LM7805	5V 三端稳压块	GCLR	KEY	全局置位/复位控制
U2	EPM7128S-84	Altera 的 CPLD 器件	GCLK	KEY	全局时钟输入
U3	P89C61X2	飞利普 89C61 增强型单片机	RST	KEY	单片机复位
U4	74HC00	4 与非门	L1～L8	LED	8 个 LED
U5	LN3461	4 位 8 段共阳数码管	POWER	LED	5V 电源指示
U6	OPTOIS01	红外接收器	ISP	LED	ISP 指示
U7	24WC02	I^2C 接口 EEPROM	RUN	LED	运行指示
U8	MAX232	RS232 串行通信接口器件	PWM LED	LED	PWM 指示
U9	PCF8563T	时钟芯片	Y1	(1～24) MHz 晶振	CPLD 时钟用户自选晶振
B1	BUZZER	蜂鸣器	Y2	11.0592MHz 晶振	单片机时钟
K1～K8	KEY	8 个按键	Y3	32768Hz 晶振	PCF8563T 时钟
GEO	KEY	全局三态控制			

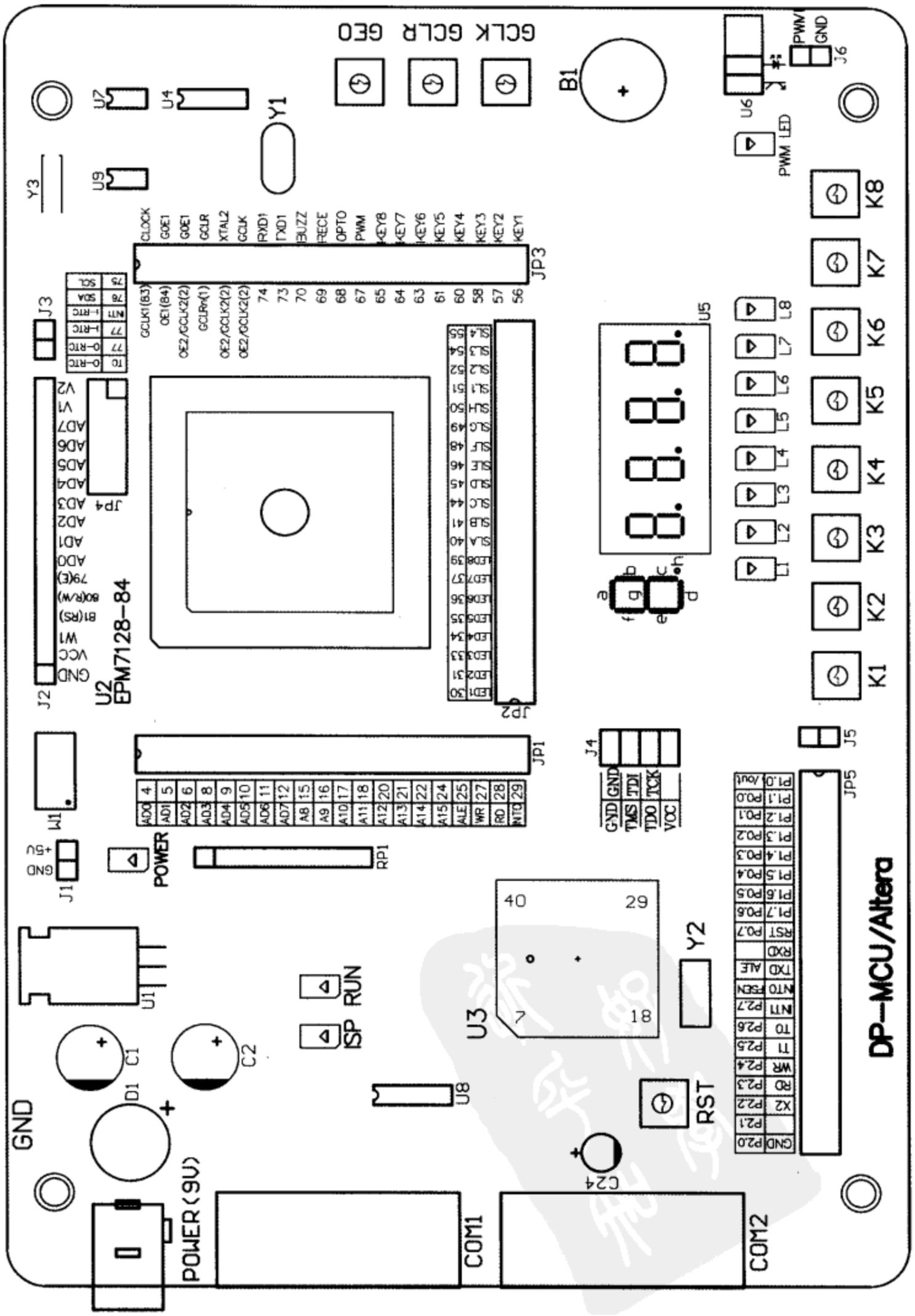


图2-1 DP-MCU/Altera实验仪电路布局

实验仪中 CPLD 采用的是 ALTERA MAX7000 系列中的 EPM7128S，为 PLCC84 封装，EPM7128S 内部有 128 个宏单元、8 个逻辑阵列块和 2500 个门电路。符合 IEEE 1149.1 JTAG 边界扫描标准，具有 5V ISP 功能。具有最小 5ns 引脚到引脚的逻辑时延，具有 175.4MHz 的计数频率，引脚可配置为开漏输出。每个宏单元都有独立的可编程电源控制，最多可以节省 50%的功耗。宏单元内的寄存器具有单独的时钟和复位等信号。支持多种电压接口。

实验仪中的单片机采用的是飞利浦 P89C61X2 Flash 单片机，用户也可以自行换成 P89C51Rx2、P89C66x 等 Flash 单片机，适用范围更广。P89C61X2 器件采用高性能的静态 80C51 设计。以先进的 CMOS 工艺制造并包含非易失性 Flash 程序存储器，可通过并行编程或在系统编程(ISP)的方法进行编程。支持 6 时钟和 12 时钟模式。P89C61X2 包含 64K Flash 程序存储器、1KBRAM、32 个 I/O 口、3 个 16 位定时/计数器、6 中断源-4 中断优先级-嵌套的中断结构、1 个增强型 UART、片内振荡器和时钟电路。此外，器件的静态设计使其具有非常宽的频率范围，甚至可低至零。具有两个软件可选的节电模式一空闲模式和掉电模式。空闲模式冻结 CPU 的运行，但允许 RAM、定时器、串口和中断系统继续保持其功能。掉电模式保持 RAM 的内容，但冻结振荡器，这样使其他片内功能都停止工作。由于是静态设计，时钟停止而不会使用户数据丢失。操作可从时钟停止点恢复运行。

二、应用接口

DP-MCU/Altera 实验仪中，为用户保留了许多应用接口，如表 2-2 所示。

表 2-2　DP-MCU/Altera 实验仪应用接口

标号	标注	功　能	连接目标
POWER	POWER(9V)	+9V 输入电源插座	主电源
COM1	DB9	CPLD 串口	RS-232 通信
COM2	DB9	单片机串口	RS-232 通信
W1		LCD 对比度调节	
J1	+5V	+5V 电源输出接口	
J2	DS1602(SIP16)	字符型液晶显示屏接口	LCD 接口
J3		LCD 背光电源输入接口	
J4	JTAG_7	CPLD 的 JTAG 接口	CPLD 编程
JP5		单片机在线调试器接口	
J6		PWM 输出接口	
U6		红外收发接口	
Y1		CPLD 晶振接口	

1. 电源插座 POWER

电源 POWER 向实验仪提供合适的工作电源，可使用配套的+9V 专用电源，以避免损坏电路元件。当实验仪连接+9V 专用电源后，电源指示 POWER 亮，说明供电正常。

2. RS-232 连接器 COM1 和 COM2

串口 COM1 和 COM2 封装为 DB9 孔式。可以通过串行通信电缆与计算机串口连接，与计算机进行数据通信。COM1 为单片机的 RS-232 输出，COM2 为 CPLD 的 I/O 口构建的 RS-232 输出。RS-232 连接器引脚及其功能如表 2-3 所示。

表 2-3　COM1 与 COM2 连接器

引　脚	名　称	功　能	引　脚	名　称	功　能
2	RXD	PC 接收数据端	5	GND	地
3	TXD	PC 发送数据端	1、4、6、7、8、9	NC	空

3. +5V 电源输出接口 J1

通过 J1 接口，实验仪能够向外部电路输出+5V、200 mA 的直流电源信号，方便用户开发自己的应用电路。需要说明的是，输出电流不可大于 200mA，否则将影响实验仪正常工作。

4. 点阵字符液晶显示屏通用接口 J2

通过 J2 接口，实验仪可以驱动显示一个标准的点阵字符液晶显示屏(16×1 行、16×2 行和 16×4 行等)。图 2-2 是 16 字×2 行(下称 16×2)液晶显示模块的外形，其接口引脚有 16 只，管脚功能如表 2-4 所示。

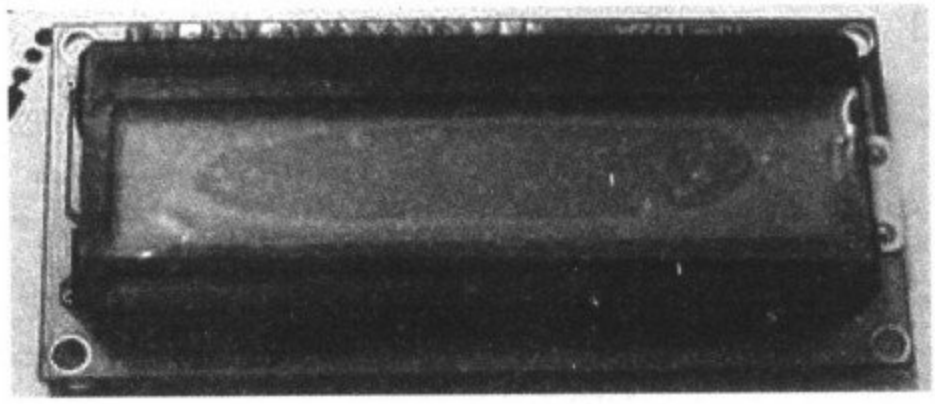


图 2-2　16×2 液晶显示模块外形

表 2-4　字符型液晶显示接口功能

编　号	符　号	功　能	编　号	符　号	功　能
1	VSS	电源地	9	AD2	数据 2
2	VDD	电源正极	10	AD3	数据 3
3	W1(VLCD)	液晶显示偏压信号	11	AD4	数据 4
4	RS	数据/命令选择	12	AD5	数据 5
5	R/W	读/写选择	13	AD6	数据 6
6	E	使能信号	14	AD7	数据 7
7	AD0	数据 0	15	BLA	背光源正极
8	AD1	数据 1	16	BLK	背光源负极

上表中，VSS 为电源地，VDD 接 5V 正电源，W1 为液晶显示器对比度调整端，接正电源时对比度最弱，接地时对比度最高，对比度过高时会产生“鬼影”，使用时可以通过一个 10KΩ 的电位器调整对比度。RS 为寄存器选择，高电平时选择数据寄存器，低电平时选择指令寄存器。R/W 为读写信号线，高电平时进行读操作，低电平时进行写操作。E 端为使能端，当 E 端由高电平跳变成低电平时，液晶模块执行命令。AD0～AD7 为 8 位双向数据线。BLA、BLK 用于带背光的模块，不带背光的模块这两个管脚悬空不接。

5. 背光电源输入接口 J3

通过 J3 接口，可以为带背光照明的 LCD 模块提供背光照明输入电源。需要注意的是，不同型号的 LCD 模块可能有着极性相反的背光电源输入信号。用户应注意阅读相应 LCD 模块的使用说明书。

6. CPLD 的 JTAG 接口 J4

J4 为 CPLD 器件的编程接口。该接口用于给 CPLD 编程。

7. 单片机扩展总线接口 JP5

扩展总线接口 JP5 引出了单片机的所有输出信号，以方便用户将实验仪连接到自己的应用系统，调试具体的功能。例如，可以通过 JPl 连接 A/D、D/A 转换板，步进电机、伺服电机控制板，语音演示板，IC 卡读写板等，或连接用户接口电路。

8. PWM 输出接口 J6

通过 PWM 输出接口，实验仪能够与外部电机相连实现电机驱动。用户可以利用本实验仪来产生 PWM 信号，开发相关的设备。

9. 红外收发接口 U6

通过 U6 红外收发接口，实验仪能够与外部其他红外收发设备进行通信。用户可以利用本实验仪，进行红外收发设备的开发与相关的实验。

10. CPLD 的用户晶振接口 Y1

Y1 为用户提供定义 CPLD 工作频率的晶振接口。用户可接上(1～24) MHz 的无源晶振(如果不接，则工作频率为 50MHz)。

三、跳线接口

DP-MCU/Altera 实验仪提供了 JP1、JP2、JP3、JP4 和 J5 共 5 个跳线接口，用户可以方便地选择 CPLD 各 I/O 口和对本实验仪自带的各种外围器件。下面简要进行说明。

1. 单片机与 CPLD 连接跳线接口 JPl

JPl 接口用于 CPLD 和单片机之间进行通信。CPLD 的 I/O 口可以通过跳线连接到单片机的数据地址总线、读/写信号和中断输入上，使得 CPLD 和单片机浑成一体。CPLD 管脚和单片机连接情况如下：

4	5	6	8	9	10	11	12	15	16
AD0	AD1	AD2	AD3	AD4	AD5	AD6	AD7	A8	A9
17	18	20	21	22	24	25	27	28	29
A10	A11	A12	A13	A14	A15	ALE	WR	RD	INT0

2. CPLD 与 LED 和数码管连接跳线接口 JP2

JP2 接口用于 CPLD 与 LED 和数码管之间的连接。CPLD 的 I/O 口可以通过跳线连接到板上的 LED 和数码管，也可以通过连线引出，供用户使用。CPLD 管脚连接情况如下：

30	31	33	34	35	36	37	39	40	41
LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8	SLA	SLB
44	45	46	48	49	50	51	52	54	55
SLC	SLD	SLE	SLF	SLG	SLH	SL1	SL2	SL3	SL4

3. CPLD 与键盘等连接跳线接口 JP3

JP3 接口用于 CPLD 与键盘、全局变量输入之间的连接。CPLD 的 I/O 口可以通过跳线连接到板上的按键，也可以通过连线引出，供用户使用。CPLD 管脚连接情况如下：

56	57	58	60	61	63	64	65	67	68
KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7	KEY8	PWM	OPTO
69	70	73	74	OE2 GCLK2(2)	OE2 GCLK2(2)	GCLRn (1)	OE2/ GCLK2(2)	OE1 (84)	GCLK1 (83)
RECE	BUZZ	TXD1	RXD1	GCLK	XTAL2	GCLR	GOE1	GOE1	CLOCK

重点提示 CPLD(EPM7128S)的 1、2、83、84 为全局输入引脚，说明如下。

GCLK1(83)：全局时钟脚，这个脚的驱动能力最强，到所有逻辑单元的延时基本相同，所以如系统有外部时钟输入，建议定义此脚为时钟输入脚。如想用其他脚为时钟输入，必须在 MAX+plusII 软件的菜单 Assign→Global project logic synthesis→Automatic global 选项中把 GCLK 前面的勾去掉。这样任意一个 I/O 脚均可做时钟输入脚。在板子上通过 K1 将时钟可以接到 83。

OE1(84)：全局输出使能，如有三态输出，建议由此脚来控制(也可由内部逻辑产生输出使能信号)，用法同上。

OE2/GCLK2(2)：全局输出使能/全局时钟脚，两者皆可。

GCLRn(1)：全局清零，如有寄存器清零，建议由此脚来控制(也可由内部逻辑产生清零信号)，用法同上。分配这些脚和分配普通 I/O 脚是一样的，先在 Assign→Device 中选好器件型号，再在 Assign→Pin 中填入你想分配的管脚号和类型，或直接在原理图中选中 Input 或 Output，点鼠标右键，选 Assign Pin，填入你想分配的管脚号，编译一遍即可。但要注意菜单 Assign→Global project logic synthesis→Automatic global 选项中的设置。

如果上面的管脚不用，最好接地，其他不用管脚一般悬空。如果不用的管脚与外电路相连，为了保证不影响外电路，应该将此电路管脚定义为输入脚，但不接逻辑。

4. CPLD 与 12C 和时钟等连接跳线接口 JP4

JP4 接口用于 CPLD 与 I^2C 和时钟等接口之间的连接。CPLD 的 I/O 口可以通过跳线连接到板上的 EEPROM 和时钟芯片上，也可以通过连线引出，供用户使用。CPLD 管脚连接情况如下：

75	76	INT1	77	77	T0
SCL	SDA	I-RTC	I-RTC	O-RTC	O-RTC

5. 单片机仿真电源选择跳线接口 J5

当用户需要使用仿真器对实验仪进行操作，或利用本实验仪 MCU 对用户的外围电路进行操作时，可选择由本实验仪对仿真头及外围电路供电（短路），或不选择（开路）。这样既方便了用户，又保护了电路。

四、原理简介

1. 时钟电路

实验仪共使用了 3 种不同的时钟源，分别是 CPLD 时钟输入 Y1、单片机时钟输入 Y2 和实时时钟 PCF8563 的时钟输入源 Y3，其电路原理图如图 2-3 所示。其中 Y1 是可插拔、更换的，用户可根据实际需要更换（1～24）MHz 之间的晶振；而 Y2 和 Y3 则是 11.059 26 MHz和 32.768 kHz 的固定晶振。

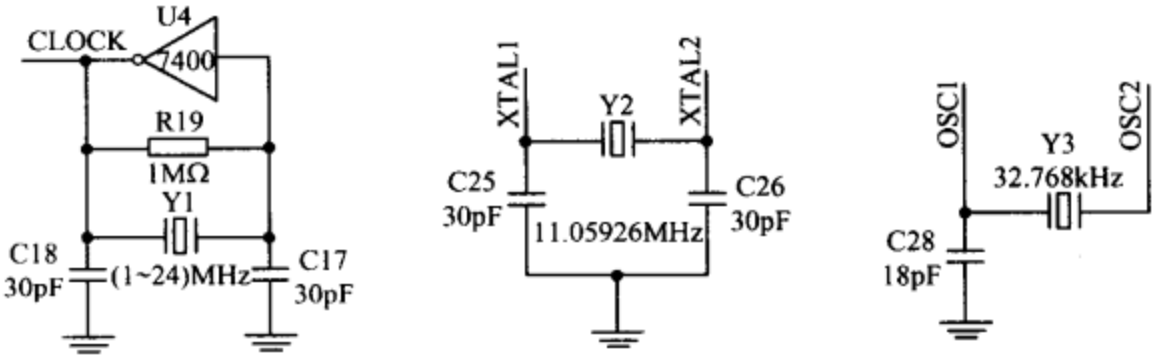


图 2-3　时钟电路

2. 复位电路

复位电路如图 2-4 所示。其中标号 RST 直接与单片机的复位引脚 10 脚相连，而 C0_RST 直接与串口 COM2 的 7 脚相连。由图可以看出，本实验仪不仅具有上点自动复位和手动复位的功能，而且通过对串口的控制（使 7 脚输出一个低电平脉冲）同样也可以完成复位的功能。

3. 工作模式切换电路

实验仪有 2 种工作模式：一种是下载模式，在此模式下通过该公司提供的 ZLGICD 软件，用户可以把目标程序代码经串口下载到单片机 P89C61X2 中；另一种是工作模式，即系统运行单片机内部的用户程序，可完成用户的特定功能。图 2-5 为工作模式切换及指示电路。

其中标号 C0_PSEN 直接与串口 COM2 的 4 引脚相连，而 PSEN 和 RXD0 分别与单片机的 32 引脚 PSEN 和 11 引脚 RXD 相连。这样，在通常状况下，由于 C0_PSEN 悬空，使得 PSEN 为高电平，因此系统复位后该实验仪即处于工作模式。一旦用户需要进入

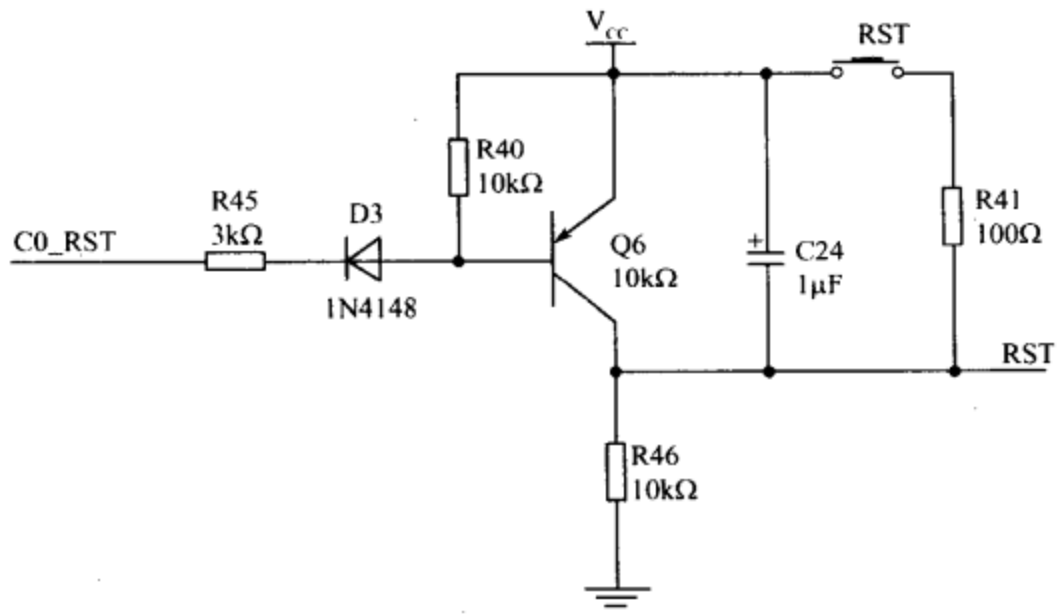


图 2-4　复位电路

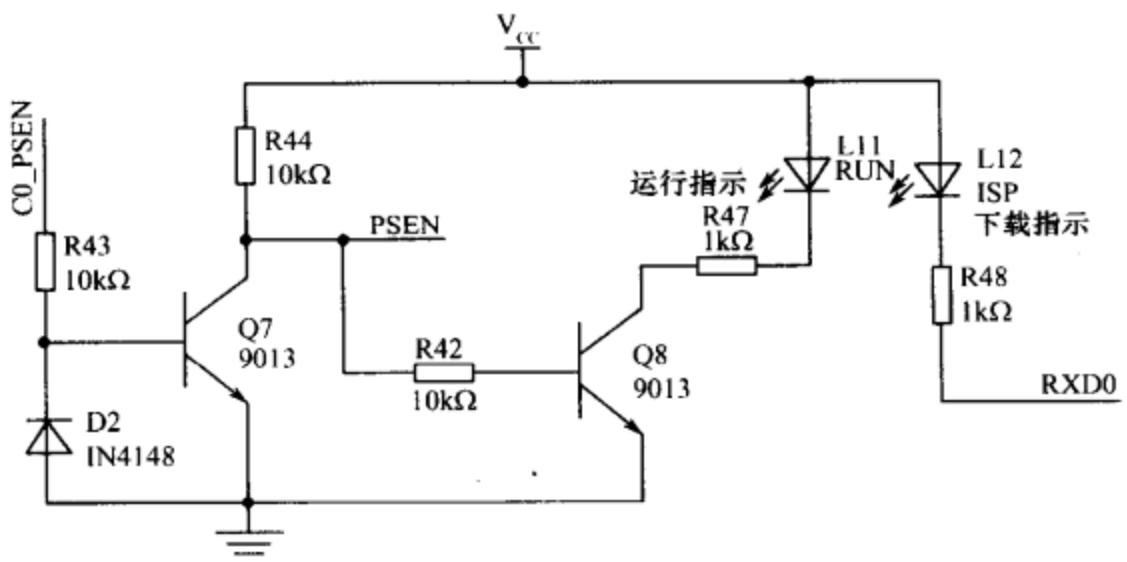


图 2-5　工作模式切换及指示电路

ISP 下载模式，运行 ZLGICD 软件即可。该软件自动把 C0_PSEN 引脚置为高电平“1”，这样由于三极管 Q7 的导通使得 PSEN 为低电平“0”。此后该软件会自动在 COM2 口的 7 脚输出一个低电平的复位脉冲，系统复位后即可进入 ISP 下载模式。

4. 键盘和 LED 发光管

实验仪为用户准备了 8 个通用、独立的键盘 K1～K8 和 8 个独立的 LED 发光管，其电路原理图如图 2-6 所示。通过跳线，它们可以分别与 CPLD 芯片 EPM7128S 的 56～58、60、61、63～65、30、31、33～37 和 39 脚相连，完成基本的输入/输出实验。

5. 全局按键

实验仪为用户设计了 3 个全局按键 GTS(GOE)、GSR(GCLR)和 GCK(GCLK)，如图 2-7 所示。由于通过跳线它们直接与芯片内部的全局模块相连，因此，可以灵活使用这些资源来实现不同的功能，如系统清零、系统使能和系统时钟等。合理使用它们可以大大节省芯片的资源。

6. 蜂鸣器

由于蜂鸣器具有控制简单、声响悦耳动听的特性，在工程项目中常用做人机接口的重要输出设备，用以发出语音提示信息，使系统更加完善、适用。蜂鸣器有交流和直流 2 种。

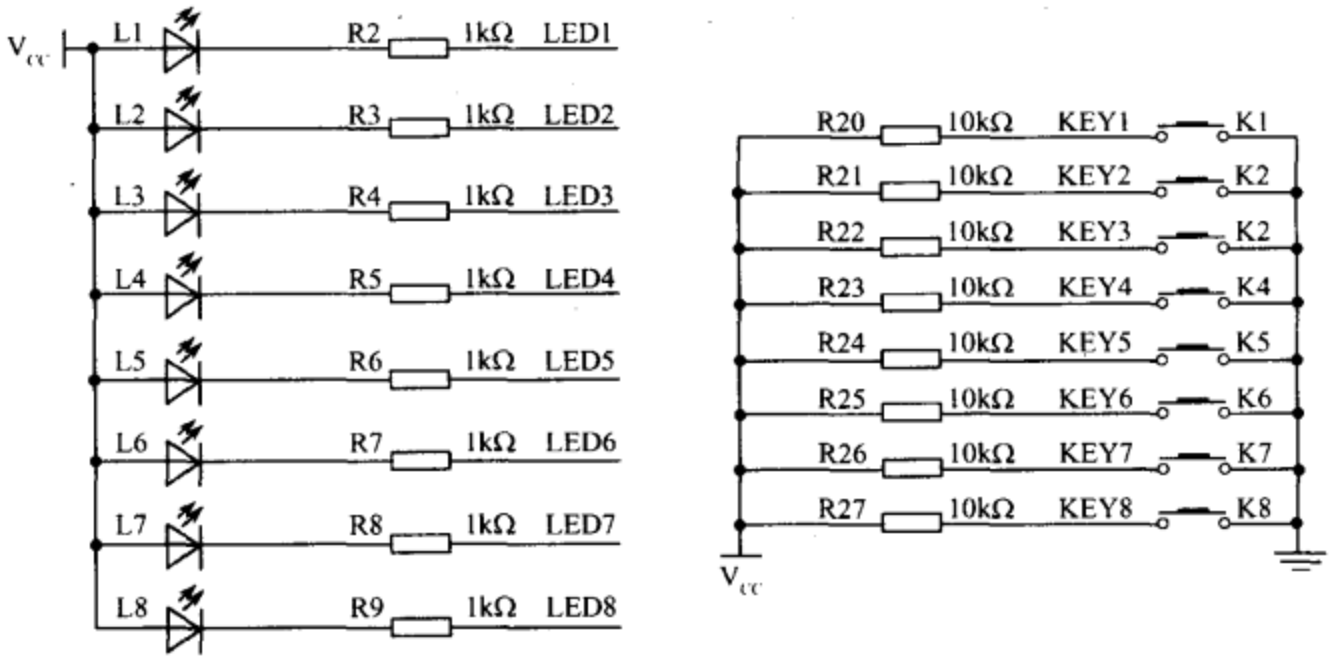


图 2-6　键盘和 LED 发光管电路

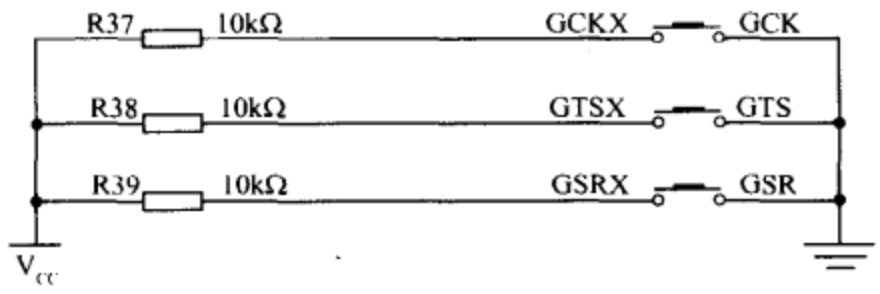


图 2-7　全局按键

直流蜂鸣器驱动简单，一旦在 2 引脚上加入直流电源，就会发出一定频率的声音，此时声音的音调和音量是固定的；而交流蜂鸣器在这方面则显得较灵活，输入的声音信号的频率和音长是用户可控的，因此输出的声音更逼真、更悦耳。实验仪上安放了一台交流蜂鸣器。由于一般 I/O 口的驱动能力有限，在此采用了三极管 Q1 来驱动蜂鸣器，其硬件原理图如图 2-8 所示。BUZZ 通过一个跳线与芯片 EPM7128S 的 70 引脚相连。当 BUZZ 输出高电平时，蜂鸣器不响；而当 BUZZ 输出低电平时，蜂鸣器发出响声。只要控制 BUZZ 输出高低电平的时间，就可以让蜂鸣器发出悦耳的音乐。

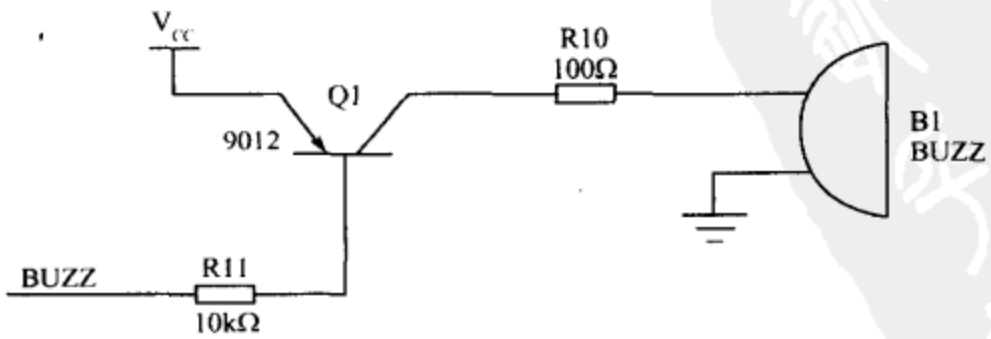


图 2-8　蜂鸣器电路

7. 数码管 LED 显示

实验仪上有 4 位共阳 LED 数码管，其标号分别为 W1～W4，其硬件原理图如图 2-9 所示。

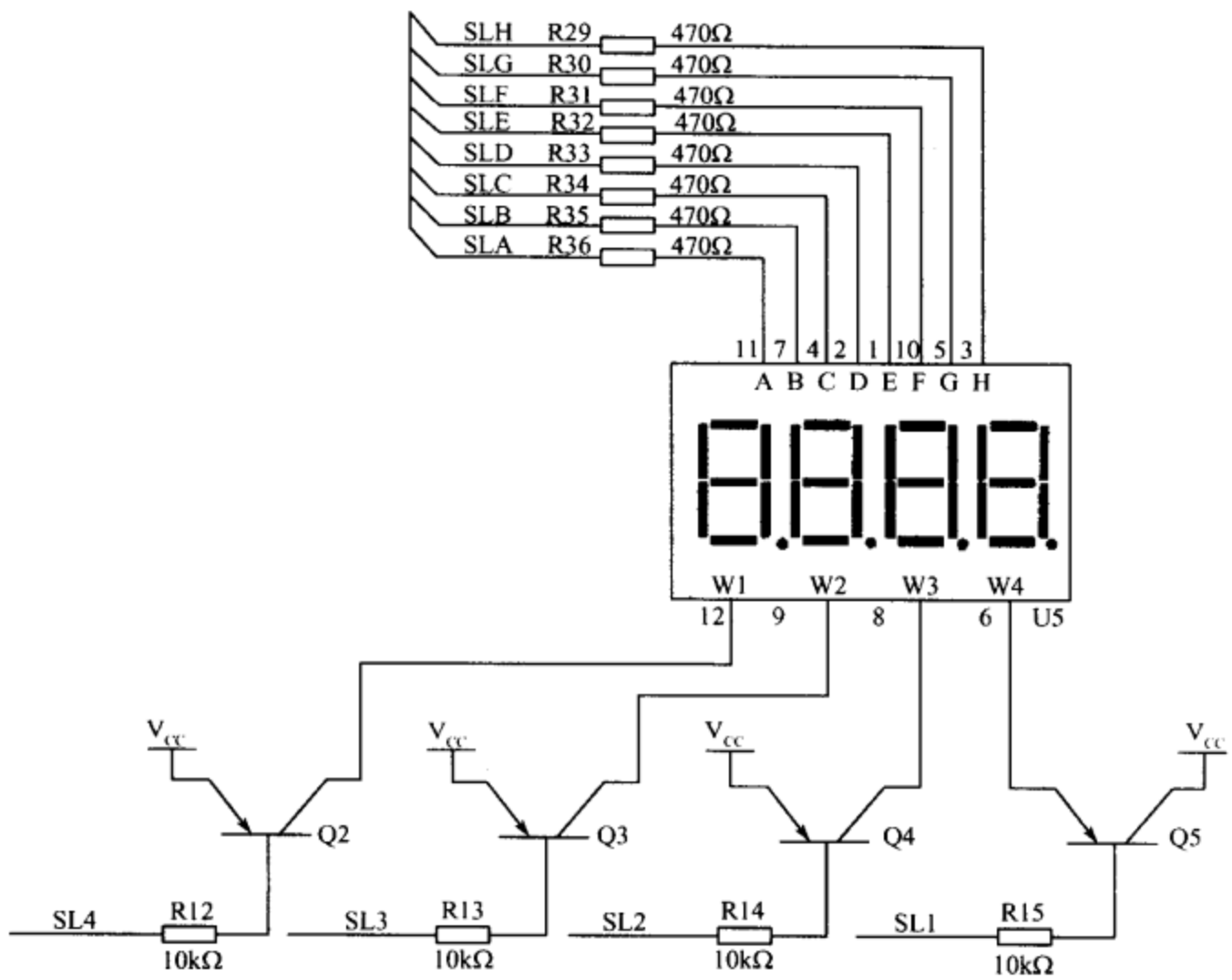


图 2-9　数码管 LED 显示电路

可以看出，段码线 SLA—SLH 通过跳线 JP2 分别与芯片的 40、41、44～46、48～50 引脚相连，而位码线 SL1～SL4 通过跳线 JP3 分别与芯片的 51、52、54、55 引脚相连。这样只需在相应的时刻，在段码和位码线上输出相应的段码和位码，即可完成 LED 数码管显示。

8. LCD 液晶显示接口

实验仪上有一标准的 LCD 液晶显示器接口 J2，如图 2-10 所示。该接口共有 16 个引脚，其中 P15 和 P16 为背光源输入。由于 1602A 液晶块是不带背光的，因此可以不用它。该接口的 7～14 引脚经过上拉电阻直接与单片机的 P0.0～P0.7 相连，而读/写控制信号则直接由 CPLD 给出。

9. 红外接近开关

红外接近开关电路原理图如图 2-11 所示。由图可以看出，该器件由 2 部分组成：一是发射电路，主要由红外发射二极管组成；二是接收电路，主要由红外接收三极管组成。其中标号 OPTO 通过跳线与 CPLD 芯片 EPM7128S 的 68 引脚相连，标号 RECE 通过跳线与芯片的 69 引脚相连。这样一旦 OPTO 为低电平“0”，红外发射二极管发射红外光。此时若发射处的红外光未被反射回来（前方没有检测物体），则红外接收三极管因没有接收到红外光而无法导通，此时 RECE 一直为高电平“1”；若发射处的红外光被反射回来（前方有检测物体），则红外接收三极管因接收到红外光而导通，此时 RECE 被拉为低电平“0”。

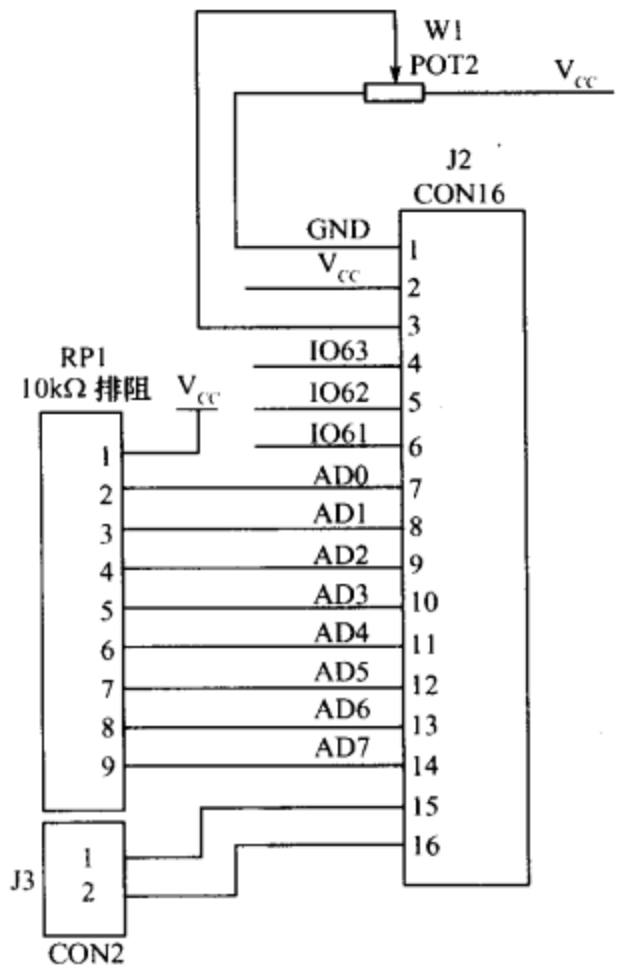


图 2-10　LCD 接口电路

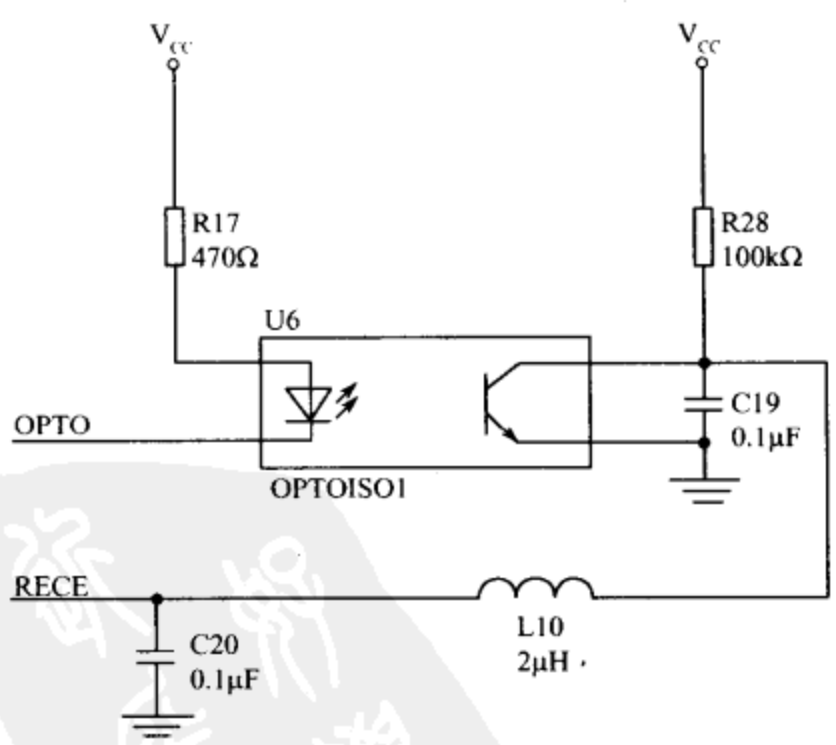


图 2-11　红外接近开关电路

10. 实时时钟 PCF8563

实验仪为用户准备了一个实时时钟电路，其电路原理图如图 2-12 所示。

由图可以看出，它主要由一个精密的时钟占用晶体 32.768 kHz 和时钟芯片 PCF8563T 组成。其中串行 I^2C 总线的时钟信号 SCL 和数据信号 SDA 直接与单片机的 P1.6 和 P1.7 相连，而 I_RTC 和 O_RTC 通过跳线分别与单片机的 P3.2 和 P3.3 或芯片 EPM7128S 的 77 脚相连。这样用户即可以利用单片机对 PCF8563T 进行读/写等操作，

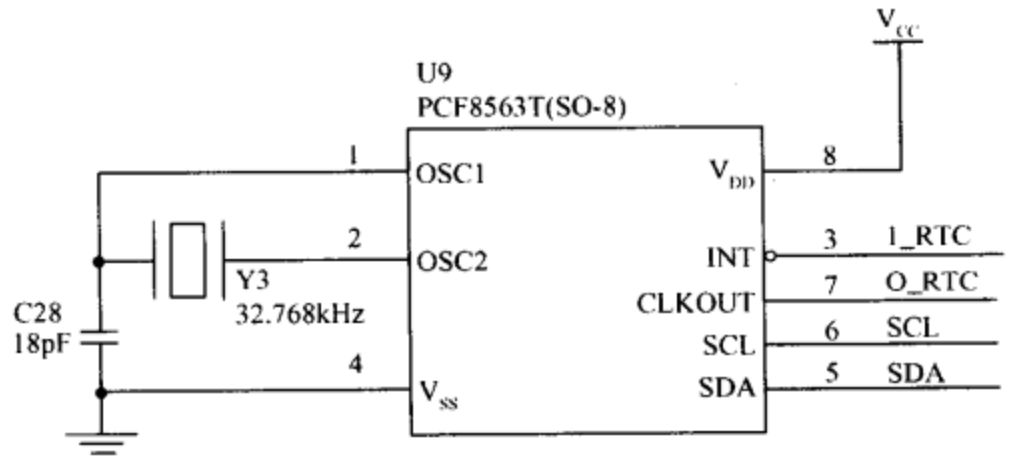


图 2-12　实时时钟电路

也可以利用 CPLD 完成对 PCF8563T 的读/写。

11. EEPROM 存储器

EEPROM 存储器电路原理图如图 2-13 所示。它和 PCF8563 芯片共用 I^2C 总线，用户既可以利用单片机对 PCF8563 进行读/写等操作，也可以利用 CPLD 完成对 PCF8563 的读写。

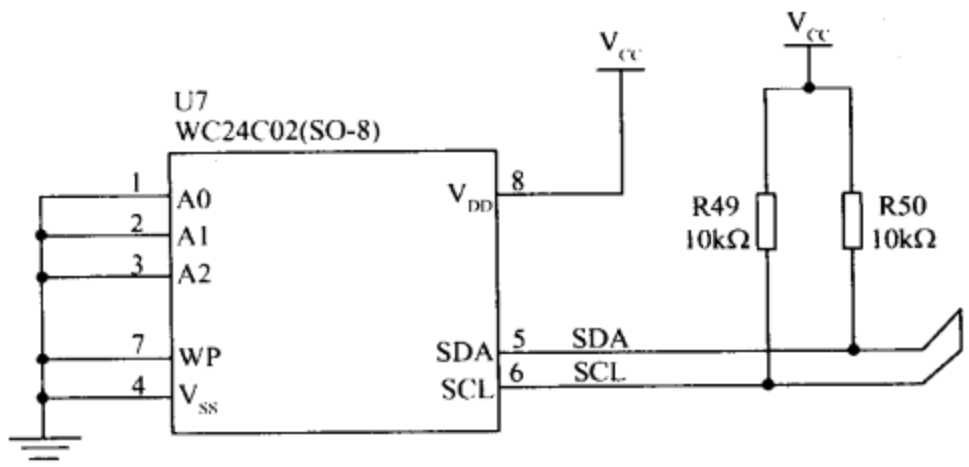


图 2-13　EEPROM 存储器电路

第二节　DP-MCU/Xilinx 实验仪

广州致远电子有限公司设计的 DP 系列下载实验仪共有 2 款，分别是 DP-MCU/Altera 和 DP-MCU/Xilinx 实验仪。它们的原理、电路布局和标号都完全相同，不同之处仅仅在于 DP-MCU/Altera 综合仿真实验仪使用的 CPLD 器件是 EPM7128S，而DP-MCU/Xilinx 实验仪使用的则是 XC95108，由于它们的引脚排列不尽相同，使得某些标号和功能引脚的对应关系也不尽相同，因此在使用时应加以区分。

DP-MCU/Xilinx 实验仪是一种功能强大的单片机、Xilinx 系列 CPLD 应用技术学习和调试的开发工具，配合 keil 软件可进行单片机实验，配合 WebPACK 软件可进行 Xilinx 公司 CPLD 实验。实验仪电路布局如图 2-14 所示。

一、实验仪主要器件

DP-MCU/Xilinx 实验仪主要器件如表 2-5 所示。

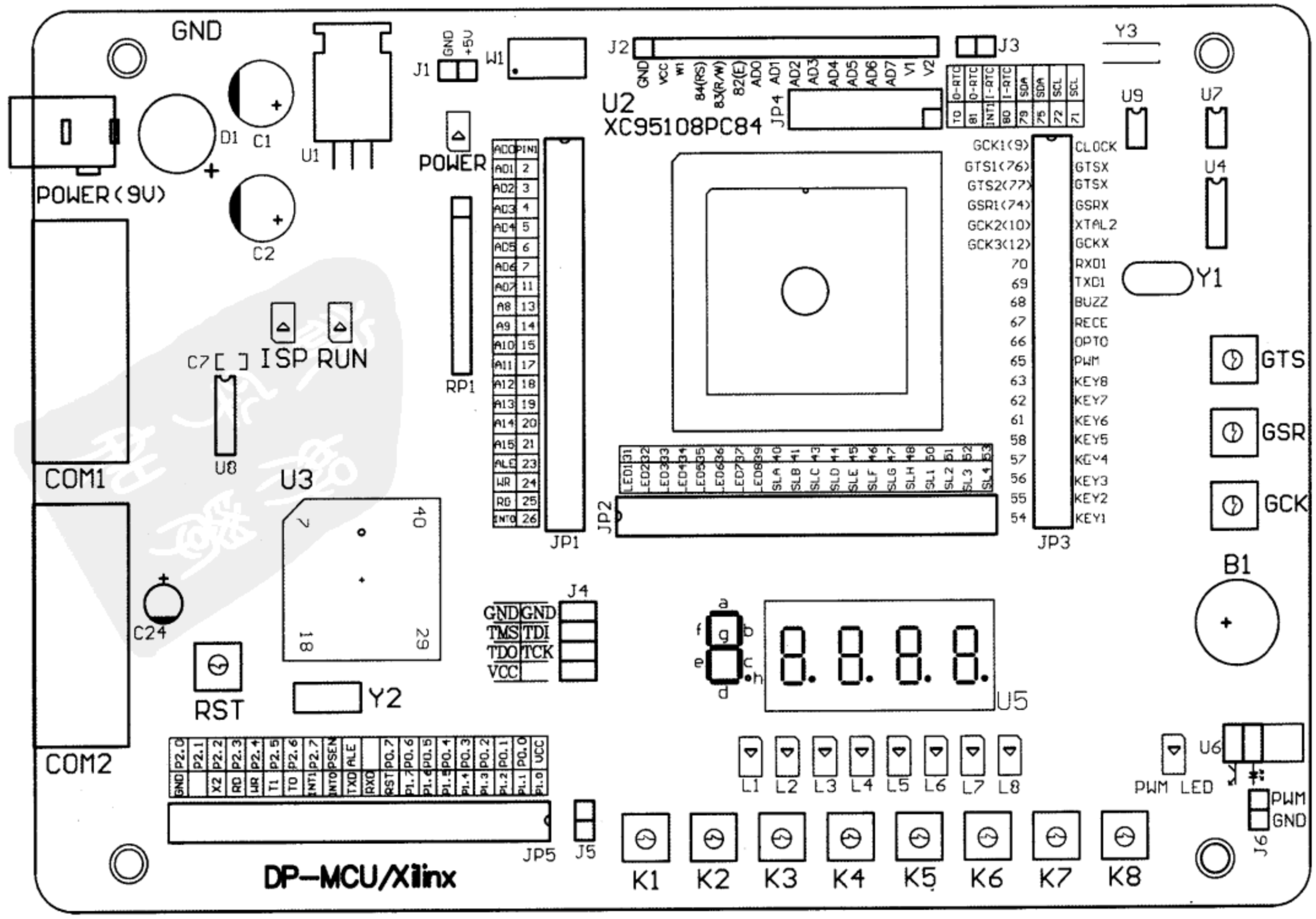


图2-14　DP-MCU / Xilinx实验仪电路布局

表 2-5　DP-MCU/Xilinx 实验仪主要器件

标　号	型　号	功　能
U1	LM7805	5V 三端稳压块
U2	XC95108PC84	Xilinx 的 CPLD 器件
U3	P89C61X2	飞利普 89C61 增强型单片机
U4	74HC00	4 与非门
U5	LN3461	4 位 8 段共阳数码管
U6	OPTOIS01	红外接收器
U7	24WC02	I^2C 接口 EEPROM
U8	MAX232	RS232 串行通信接口器件
U9	PCF8563T	时钟芯片
B1	BUZZER	蜂鸣器
K1～K8	KEY	8 个按键
GTS	KEY	全局三态控制
GSR	KEY	全局置位/复位控制
GCK	KEY	全局时钟输入
RST	KEY	单片机复位
L1～L8	LED	8 个 LED
POWER	LED	5V 电源指示
ISP	LED	ISP 指示
RUN	LED	运行指示
PWM LED	LED	PWM 指示
Y1	1～24MHz 晶振	CPLD 时钟用户自选晶振
Y2	11.0592MHz 晶振	单片机时钟
Y3	32768Hz 晶振	PCF8563T 时钟

实验仪中 CPLD 采用的是 Xilinx 公司的 XC9500 系列 CPLD 中的 XC95108PC84。XC95108 为 PLCC84 封装，内部有 108 个宏单元和 2 400 个门电路。具有 5V ISP 功能，采用先进的 5V FastFlash 工艺，可反复擦除多达 10 000 次，具有增强型引脚锁定功能，支持 JTAG 边界扫描，每个宏单元的功率可以单独配置，个别引脚输出速度可控，可驱动 24mA 输出。

实验仪中的单片机采用的是飞利浦 P89C61X2 Flash 单片机，与 DP-MCU/Altera 实验仪中的单片机相同。

二、应用接口

DP-MCU/Xilinx 实验仪为用户保留了许多应用接口，这些应用接口与DP-MCU/Altera 实验仪完全相同，这里不再重复。

三、跳线接口

DP-MCU/Xilinx 实验仪提供了 JP1、JP2、JP3、JP4 和 J5 共 5 个跳线接口，用户可以方便地选择 CPLD 各 I/O 口和对本实验仪自带的各种外围器件。下面简要进行说明。

1. 单片机与 CPLD 连接跳线接口 JPl

JPl 接口用于 CPLD 和单片机之间进行通信。CPLD 的 I/O 口可以通过跳线连接到单片机的数据地址总线、读/写信号和中断输入上，使得 CPLD 和单片机浑成一体。CPLD 管脚和单片机连接情况如下：

1	2	3	4	5	6	7	11	13	14
AD0	AD1	AD2	AD3	AD4	AD5	AD6	AD7	A8	A9
15	17	18	19	20	21	23	24	25	26
A10	A11	A12	A13	A14	A15	ALE	WR	RD	INT0

2. CPLD 与 LED 和数码管连接跳线接口 JP2

JP2 接口用于 CPLD 与 LED 和数码管之间的连接。CPLD 的 I/O 口可以通过跳线连接到板上的 LED 和数码管，也可以通过连线引出，供用户使用。CPLD 管脚连接情况如下：

31	32	33	34	35	36	37	39	40	41
LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8	SLA	SLB
43	44	45	46	47	48	50	51	52	53
SLC	SLD	SLE	SLF	SLG	SLH	SL1	SL2	SL3	SL4

3. CPLD 与键盘等连接跳线接口 JP3

JP3 接口用于 CPLD 与键盘、全局变量输入之间的连接。CPLD 的 I/O 口可以通过跳线连接到板上的按键，也可以通过连线引出，供用户使用。CPLD 管脚连接情况如下：

54	55	56	57	58	61	62	63	65	66
KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7	KEY8	PWM	OPTO
67	68	69	70	GCK3	GCK2	GSR1	GTS2	GTS1	GCK1
RECE	BUZZ	TXD1	RXD1	GCKX	XTAL2	GSRX	GTSX	GTSX	CLOCK

4. CPLD 与 12C 和时钟等连接跳线接口 JP4

JP4 接口用于 CPLD 与 I^2C 和时钟等接口之间的连接。CPLD 的 I/O 口可以通过跳线连接到板上的 EEPROM 和时钟芯片上，也可以通过连线引出，供用户使用。CPLD 管脚连接情况如下：

71	72	75	79	80	INT1	81	T0
SCL	SCL	SDA	SDA	I-RTC	I-RTC	O-RTC	O-RTC

5. 单片机仿真电源选择跳线接口 J5

当用户需要使用仿真器对实验仪进行操作，或利用本实验仪 MCU 对用户的外围电

路进行操作时，可选择由本实验仪对仿真头及外围电路供电（短路），或不选择（开路）。这样既方便了用户，又保护了电路。

四、原理简介

1. 时钟电路

与 DP-MCU/Altera 实验仪相同。

2. 复位电路

与 DP-MCU/Altera 实验仪相同。

3. 工作模式切换电路

与 DP-MCU/Altera 实验仪相同。

4. 键盘和 LED 发光管

实验仪为用户准备了 8 个通用、独立的键盘 K1～K8 和 8 个独立的 LED 发光管，通过跳线，它们可以分别与 CPLD 芯片 XC95108 的 54～58、61～63、31～37 和 39 脚相连，完成基本的输入/输出实验。

5. 全局按键

实验仪为用户设计了 3 个全局按键 GTS、GSR 和 GCK，由于通过跳线它们直接与芯片内部的全局模块相连，因此，可以灵活使用这些资源来实现不同的功能，如系统清零、系统使能和系统时钟等。

6. 蜂鸣器

BUZZ 通过一个跳线与芯片 XC95108 的 68 脚相连。

7. 数码管 LED 显示

段码线 SLA～SLH 通过跳线 JP2 分别与 XC95108 的 40、41、43～48 引脚相连，而位码线 SL1～SL4 通过跳线 JP3 分别与芯片的 50～53 引脚相连。

8. LCD 液晶显示接口

与 DP-MCU/Altera 实验仪相同。

9. 红外接近开关

标号 OPTO 通过跳线与 CPLD 芯片 XC95108 的 66 引脚相连，标号 RECE 通过跳线与芯片的 67 引脚相连。

10. 实时时钟 PCF8563

与 DP-MCU/Altera 实验仪相同。

11. EEPROM 存储器

与 DP-MCU/Altera 实验仪相同。

第三节　其他CPLD 实验仪

CPLD 实验仪还有很多，下面再简要介绍几种。

一、CPLD-MCU 下载仿真实验仪

CPLD-MCU 下载仿真实验仪由单片机爱好者网站（www. mcufan. com）开发的，外

围器件既可以用于单片机，也可以用于 CPLD，因此，可以同时可以完成单片机和 CPLD 的实验，也可以从中学会怎样去把单片机和 CPLD 联合起来形成一个应用系统。实验仪如图 2-15 所示。

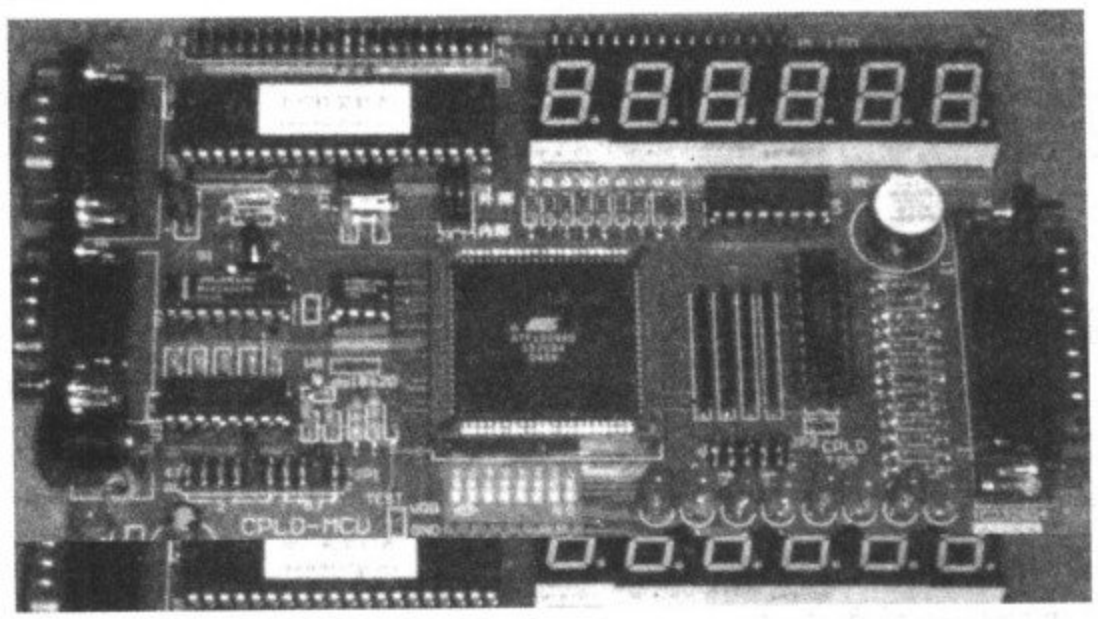


图 2-15　CPLD-MCU 下载仿真实验仪

实验仪主要包含以下器件。

1. CPLD 芯片

实验仪带一片 84 脚 PLCC 封装的 CPLD 芯片 ATF1508AS (U5)。ATF15XX 系列 CPLD 芯片与 Altera 公司 EPM7000 和 EPM3000 系列芯片的引脚一致，可与 Altera 公司芯片进行替换。可重复编程 10 000 次，放心进行实验。并且可以使用 Atmel ISP 软件对 ATF1508AS 进行编程。

2. 仿真模块

实验仪带一片仿真 63K 程序的 Keil 仿真模块(U1)，可以仿真 51 系列的单片机，仿真环境为 Keil C51。J1 插头的 40 根引脚是仿真电缆插脚，插上赠送的仿真电缆，可作为 51 仿真器用，定义和标准的 51 芯片 40 引脚一致。

3. RS232 串行接口

实验仪扩展了 2 个 RS232 接口，分别连接到单片机和 CPLD。RS232A (J2) 连接到单片机，用来给单片机进行 RS232 通信。RS232B (J8) 连接到 CPLD (73、74)，可以用 CPLD 设计一个 UART 来进行通信。

4. 按键

实验仪上有 4 个按键。作为 CPLD 实验仪的输入(CPLD 的 49、50、51、52)。也可连接到 CPU 做中断和按键的输入。

5. 拨动开关

实验仪上有一个 8 位拨动开关，作为 CPLD 实验仪的输入(CPLD—44、45、46、48)。

6. LED 显示

实验仪上有 8 个 LED 灯 L1～L8，分别是连接到 CPLD 接口的 8 个静态输出端(33、34、35、36、37、39、40、41 脚)。

7. 数码管显示

实验仪上有 6 个数码管，可用 CPLD 来进行动态扫描输出，也可用单片机控制数码

管。7 个笔划分别接到 CPLD 的 74、75、76、77、79、80、81 脚；六个段位分别接到 CPLD 的 63、64、65、67、68、69 脚。

8. 可改变频率的信号发生器

实验仪上 JP1 插座用来进行信号发生器信号频率的选择端，分别接 CPLD 的 2、83 脚，用跳线可选择不同的分频，分频信号由 32.768kHz 晶振进行 14-4 级分频后得到。

9. 蜂鸣器

实验仪上的蜂鸣器是一个无源的蜂鸣器，必须加载一个频率才可以发声。可以通过改变输出的频率来产生不同的音乐。CPLD 的 70 脚是蜂鸣器输出端。

10. I^2C 器件

实验仪上装有 I^2C 接口的 E^2PROM 芯片 24C01（U4），可以帮助开发人员快速使用 I^2C 器件，E^2PROM 的总线端接到 CPU 的 P1.7(SDA)和 P1.6(SCL)脚。

11. 温度传感器

实验仪上装有一线式温度传感器 DS1820（U8），连接到单片机的 P1.5 脚，可以做 DALLAS 一线式器件的编程实验。

12. 89S-CPLD 二合一下载电路 J6 JP2

实验仪上设有 CPLD ISP 下载插座(JP2)，可作为 CPLD 下载线，对板上的 CPLD 进行编程；也可编程 Atmel ISP 单片机 89S51、89S52。

13. 其他资源

实验仪上预留了 1 个 LCD 字符液晶显示的标准接口(J5)，可连接各种型号的字符液晶显示屏。

二、Altera CPLD 开发板

Altera CPLD 开发板是北京扬创科技(www.yctek.com)开发的一种性价比较的高 CPLD 实验板，其外形如图 2-16 所示。

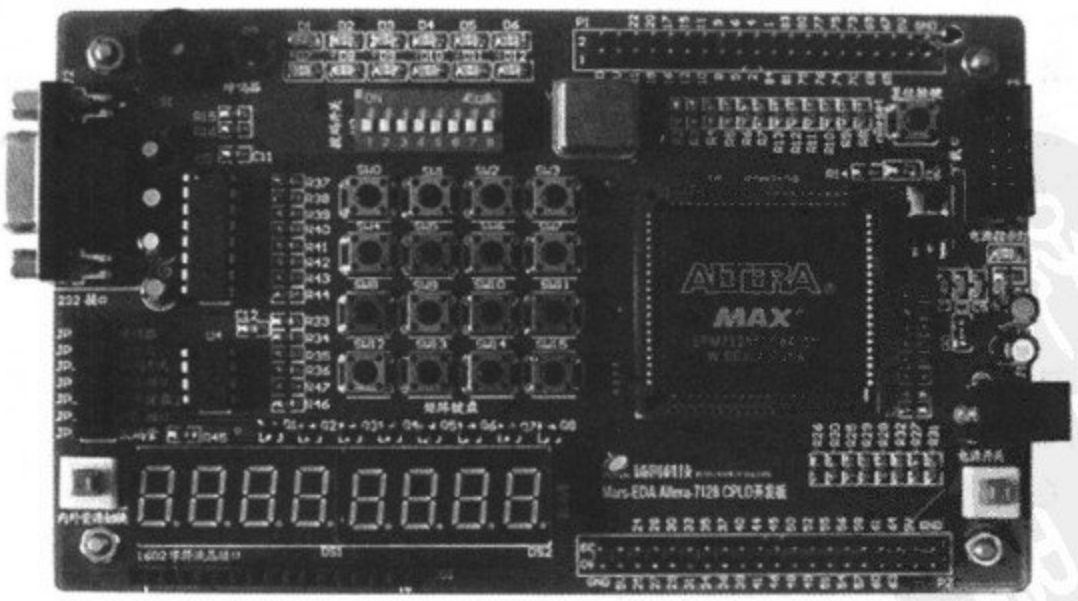


图 2-16　Altera CPLD 开发板外形

Altera CPLD 开发板主要针对 CPLD 初、中级学习者设计，帮助用户降低学习成本和加快用户快速进入可编程逻辑器件设计开发领域，提供一个帮助用户快速开始可编程逻辑器件学习之旅的硬件平台。

该CPLD实验平台提供了大量的实用的实验例程和丰富硬件资源，并介绍了关于如何在本实验平台上完成各个实验过程，实现对板上资源的利用，从而使用户获得对CPLD器件的开发应用流程得到了解。实验例程分为三个部分：基本实验，接口实验和综合实验，由浅入深，一步步引导用户。基本实验包括一些基础的组合与时序电路设计；接口实验包括一些常用的输入、显示电路以及一些较简单的接口如串口、I2C接口的设计；综合实验包括一些针对问题的综合性设计。

板上提供ISP接口并将CPLD的全部I/O引脚引出。同时通过板上的跳线，用户可以将决定是否使用板上的资源或是将开发板作为一个最小系统板以进行自己的设计，方便用户开发自己的产品，最大限度的为用户节约学习成本和加快学习速度。

1. 硬件配置情况

(1) EPM7128S；

(2) MAX232；

(3) AT24C02；

(4) 4×4矩阵按键；

(5) 8段数码管；

(6) 蜂鸣器；

(7) 拨码开关；

(8) LED灯；

(9) 40MHz晶振。

2. 实验例程

(1)基础实验　基础实验主要有加法实验、减法实验、乘法实验、除法实验、四位比较器、多路选择器、优先编码器、二进制到BCD码转换和简单状态机实验。

这几个实验都比较简单，目的是帮助用户熟悉CPLD的基本开发流程和一些常用的、基础的数字电路。

(2)接口实验　接口实验主要有：

① 跑马灯实验：跑马灯实验在CPLD中设计了计数器，利用计数器轮流向LED灯发出低电平，点亮LED灯，实现跑马灯的效果。

② 矩阵键盘实验：包括两个实验。

矩阵键盘实验1：按一个键并在7段数码管上显示相应的键值。这个实现的主要目的是向用户介绍矩阵键盘扫描检测按键的原理，没有考虑去抖动的设计。在矩阵键盘实验2中将介绍一个完整的矩阵键盘实验。

矩阵键盘实验2：按下一个键并在1602液晶显示相应的键值。提供了一个完整的矩阵键盘程序并介绍了操作1602的基本方法和步骤。

③ 7段数码管实验：包括两个实验。

7段数码管测试实验1：以动态扫描方式在8位数码管“同时”显示0～7，帮助用户了解数码管动态显示的方法。

7段数码管实验2：在4位数码管以递增方式向上显示4位数字，是一个利用CPLD设计十进制计数器的实验例子。

④ 蜂鸣器实验：向蜂鸣器发送一定频率的方波可以使蜂鸣器发出相应的音调，该实

验通过设计一个状态机和分频器使蜂鸣器发出“多来咪发梭拉西多”的音调。

⑤ 串口通信实验：从 PC 传送一个 0～F 中其中一位，在 7 段数码管的一位上显示相应的值。按下开发板键盘上某个键回送 Welcome!

⑥ 拨码开关实验：拨码开关 8 位 0、1 状态在 8 位 7 段数码管相应位上显示 0 或 1。

⑦ I^2C 接口 EEPROM 存取实验：按动开发板键盘某个键，CPLD 将拨码开关的数据写入 EEPROM 的某个地址，按动另外一个键，将刚写入的数据读回 CPLD，并在数码管上显示。帮助读者掌握 I^2C 的总线协议和 EEPROM 的读写方法。

⑧ LCD 字符液晶驱动实验：在 1602 上从左到右循环显示“welcome to Mars!”。

(3)综合实验

① 模拟交通灯实验：模拟路口的红黄绿交通灯的变化过程，用 LED 灯表示交通灯，并在数码管上显示当前状态剩余时间。

② 数字时钟实验：利用数码管和 CPLD 设计的计数器实现一个数字时钟，可以显示小时，分钟，秒。程序主要要靠考虑十进制和六十进制计数器的编写。

以上实验例程都有 Verilog、VHDL 两种语言的源代码！所有实验例程都基于 Quartus II 5.0 工程。

三、51＋CPLD 学习板

51＋CPLD 学习板由晓奇工作室网站(www. xiao-qi. com)推出，外观如图 2-17 所示。

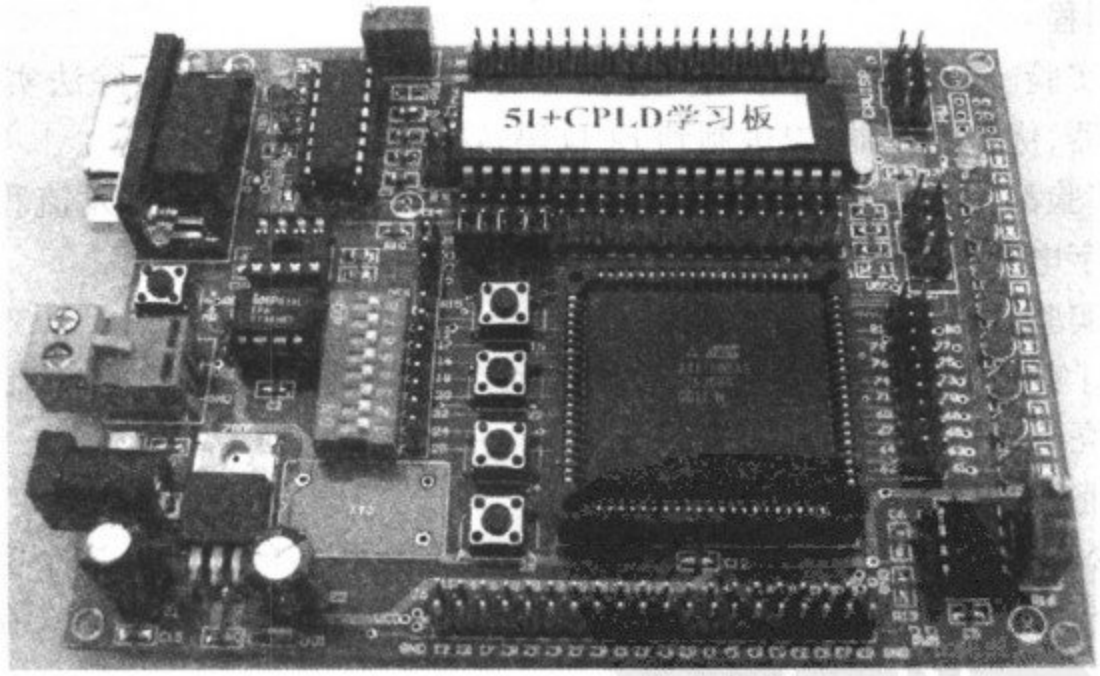


图 2-17　51＋CPLD 学习板

这款实验学习板既可做单片机实验，也可做 CPLD 实验，单片机选用了 Atmel 的 AT89S52 芯片，可以免去编程器，直接支持 ISP 下载编程调试；CPLD 采用带有 128 宏单元的 ATF1508，可以进行上万次的擦写(编程)，特别适合初学者入门学习，反复试验。实验板把所有 I/O 外引，可方便做各种实验；用 555 作的模拟 PWM，有 12 个 I/O 可选择，方便灵活；另有一个运行指示灯，用于调试程序。

第三章　CPLD 开发软件和仿真软件的使用

CPLD 开发软件对于完成硬件描述语言(如 Verilog-HDL)程序设计具有重要的意义,只有在这些开发软件的作用下,才能将 HDL 文本设计转化为具体的电路结构网表,实现设计者的设计思想;同时,只有基于这些工具的支持,才能对设计的程序进行编译和调试,以发现问题,修改设计方案,避免损失。可见,开发工具对于提高设计效率、改善优化质量、增强排错能力等都有着直接的影响。本章主要介绍 Altera 公司的 MAX+plusII 和 Xilinx 公司的 ISE WebPACK 开发软件。此外,本章还重点介绍了 Modelsim 仿真软件的使用方法和技巧。

第一节　Altera 开发软件MAX+plusII 的安装和使用

MAX+plusII(或写成 MAX plus2,或 MP2) 是 Altera 公司推出的的第三代 PLD 开发系统(Altera 第四代 PLD 开发系统被称为 QuartusII,主要用于设计新器件和大规模 CPLD/FPGA)。使用 MAX+plusII 的设计者不需精通器件内部的复杂结构。设计者可以用自己熟悉的设计工具(如原理图输入或硬件描述语言)建立设计,MAX+plusII 把这些设计自动转换成最终所需的格式,其设计速度非常快。对于一般几千门的电路设计,使用 MAX+plusII,从设计输入到器件编程完毕,用户拿到设计好的逻辑电路,大约只需几小时。设计处理一般在数分钟内完成。特别是在原理图输入等方面,MAX+plusII 被公认为是最易使用,人机界面最友善的 PLD 开发软件,特别适合初学者使用。

一、MAX+plusII 的安装

MAX+plusII 软件的试用版(也称基本版)可以从 http://www.altera.com/support/software/sof-download center.html 上下载,该软件最新版本为 MAX+plusII10.2 baseline,支持 30 000 门以下所有设计,是一个功能相当强大的免费软件。下载该软件后,可进行安装,安装过程如下:

(1)双击 baseline10_2 图标,出现如图 3-1 所示安装确认画面。

(2)单击 Next,出现软件许可协议对话框,如图 3-2 所示。

(3)单击 Yes,出现如图 3-3 所示的提示信息窗口。

(4)单击 Next,出现用户信息窗口,输入用户名和公司,如图 3-4 所示。

(5)单击 Next,出现安装组件选择画面,如图 3-5 所示。这里使用默认设置,即将 MAX+plusII baseline、MAX+plusII Help、MAX+plusII Readme 三个选项全部选中。

(6)在接下来的两个对话框中,单击 Next,此时,软件开始安装,安装完成后,将在桌面上生成 MAX+plusII10.2 baseline 快捷画标。

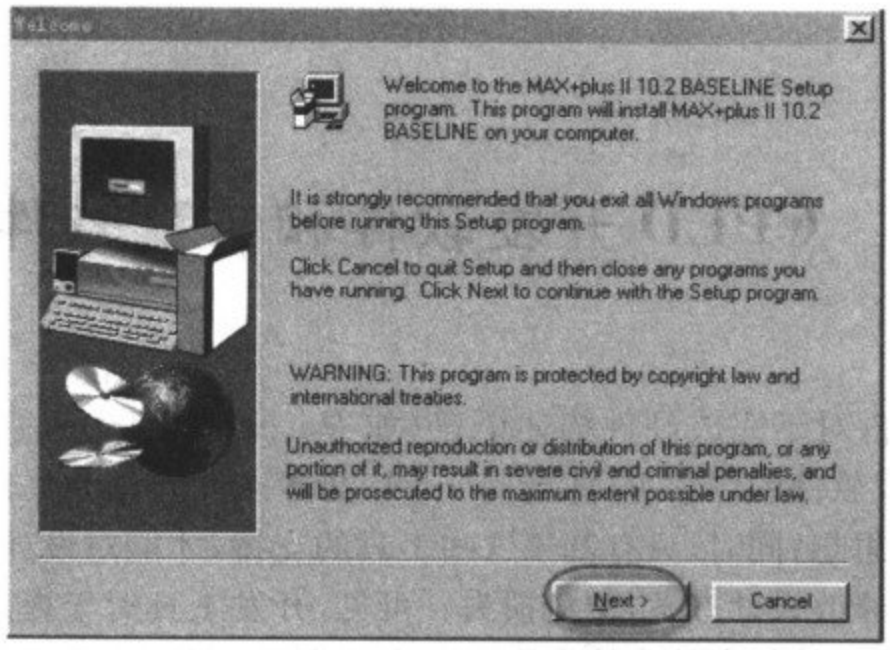


图 3-1　安装确认

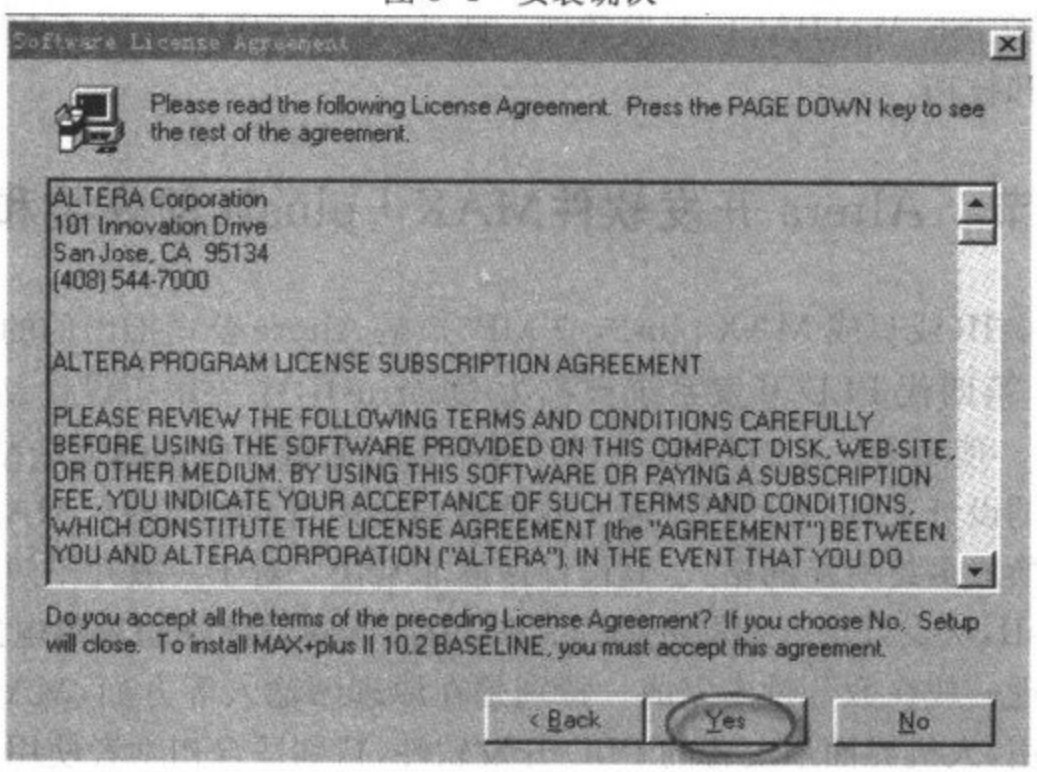


图 3-2　软件许可协议对话框

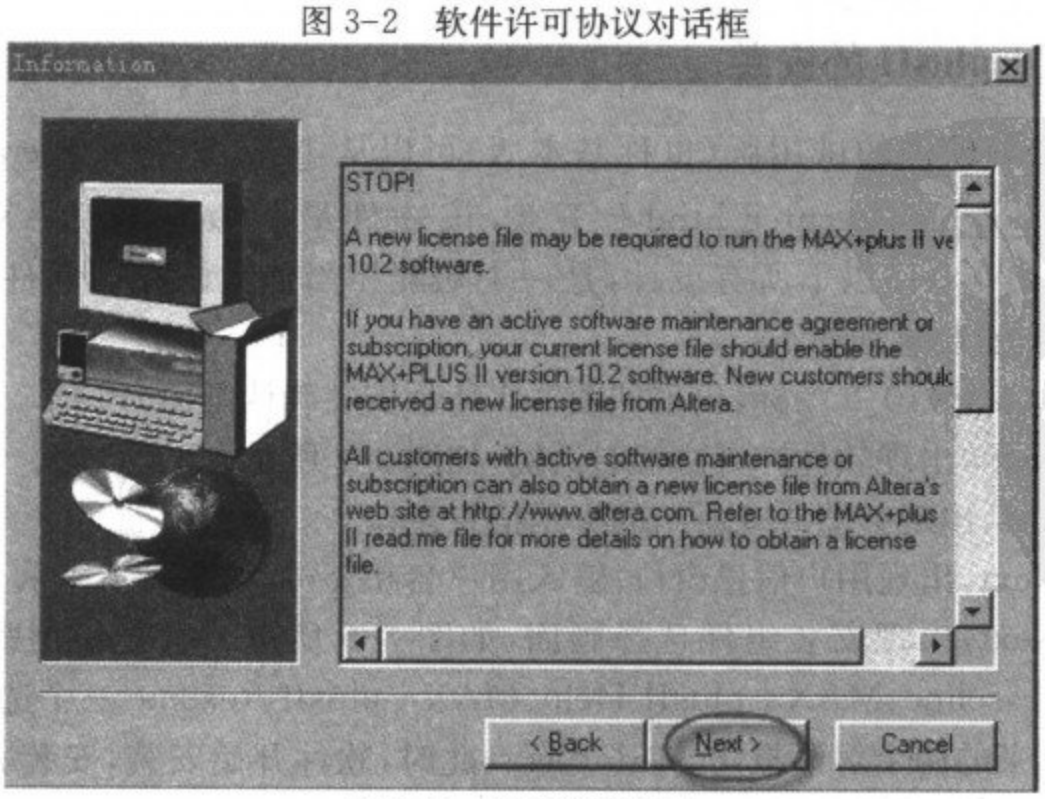


图 3-3　提示信息窗口

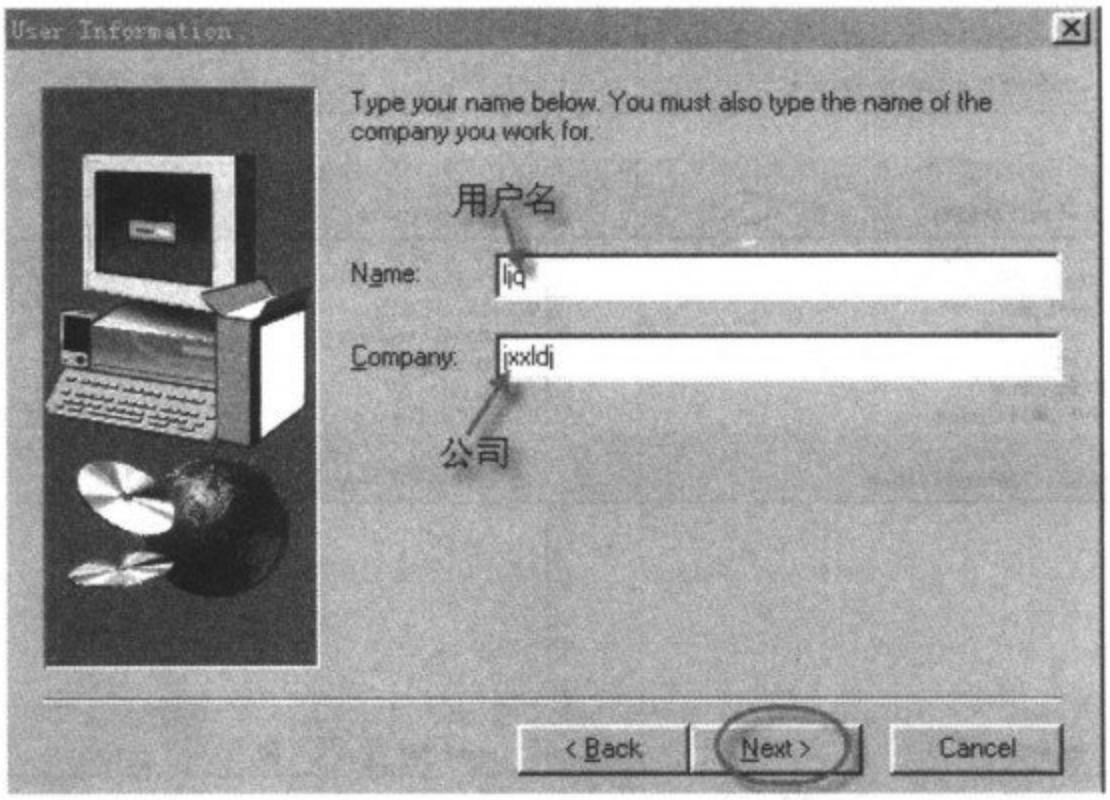


图 3-4　用户信息窗口

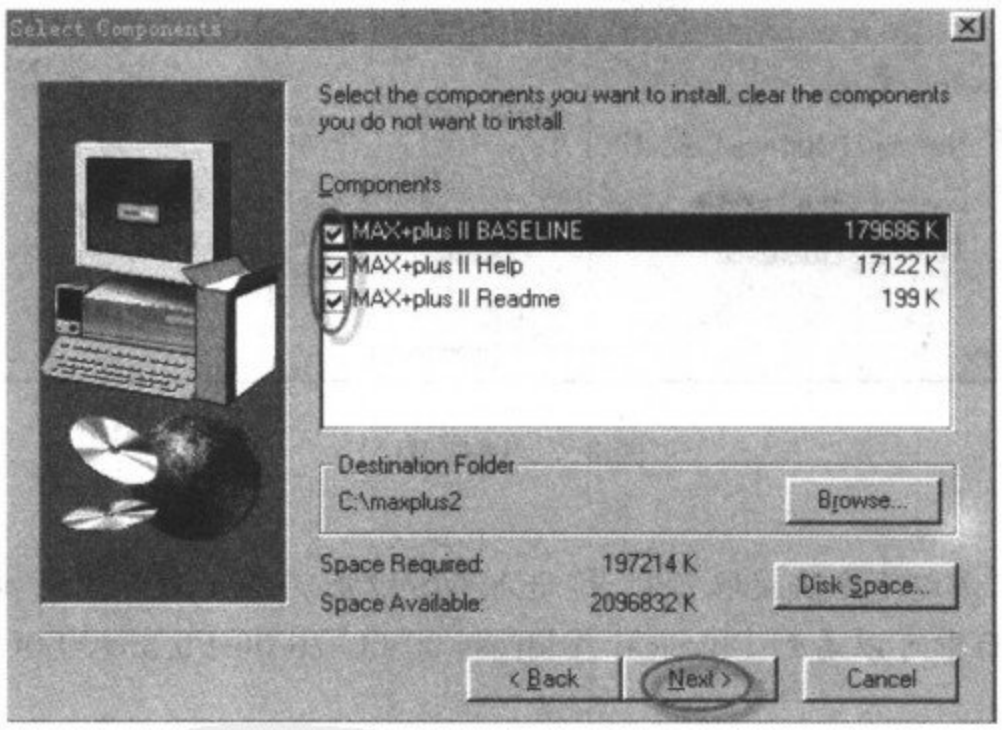


图 3-5　安装组件选择画面

(7)虽然此时已安装完毕,但用户还未获得使用授权,许多功能将受到限制。在获得授权文件 License. dat 后,双击 MAX+plusII 快捷图标进入 MAX+plusII 集成开发环境;单击菜单 Option,在弹出的下拉菜单中选择 LicenseSetup 命令;在出现如图 3-6 所示的授权设置对话框中单击"Browse..."按钮,加载 license. dat 文件;然后单击 OK 按钮,就获得了使用授权,这时许多原来不能使用的功能即可使用。

方法技巧　有 2 种方法在 Altera 公司网站上申请 Linsince. dat 文件:一是硬盘 ID 号;二是网卡号(NIC)。

获取硬盘 ID 号和网卡号的方法是:进入 MS-DOS;在命令提示行下键入 dir cpld 命令;在第 2 提示行可以看到 Volume Serial Number Is 1A63-1D08,1A63-1D08 就是您的硬盘 ID 号。也可在图 3-6 所示的画面中单击"systenm info…",出现如图 3-7 所示的面画,图中的 drive serial number 即为硬盘 ID 号,图中的 Network interface card(NIC) ID

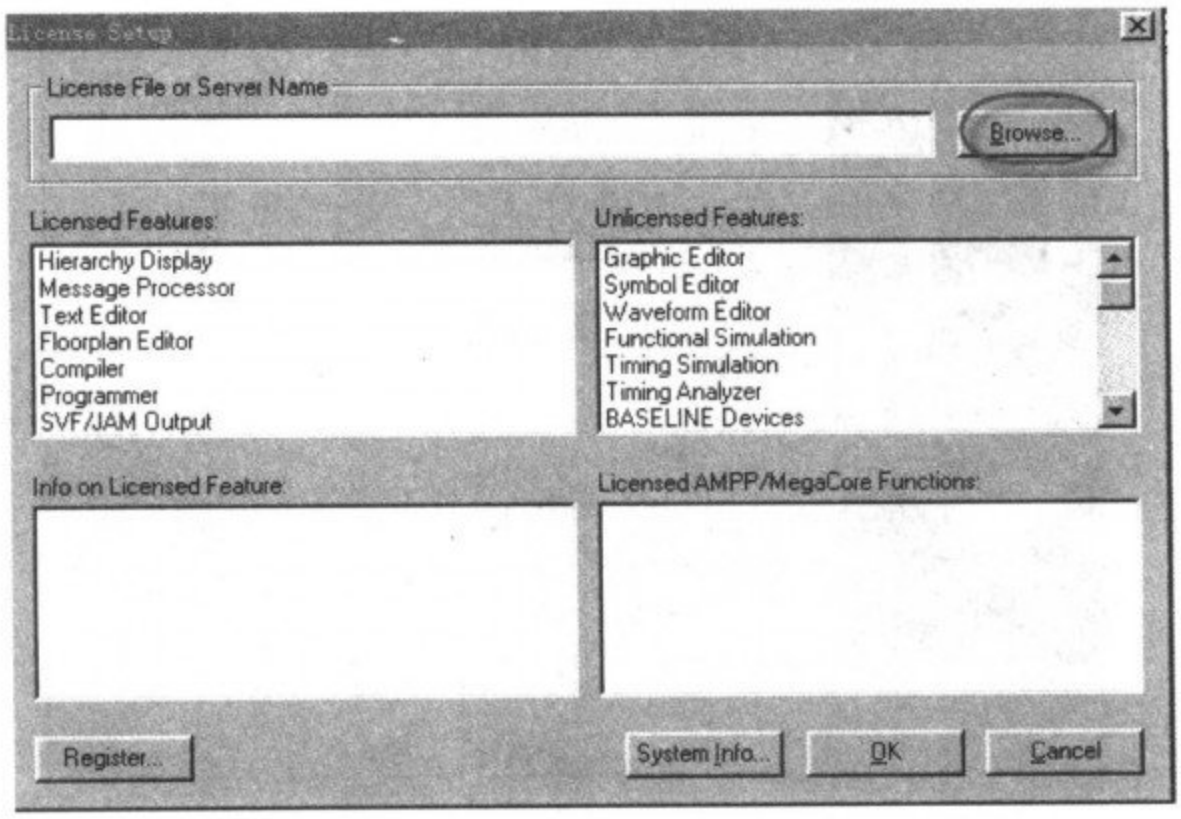


图 3-6　授权设置对话框

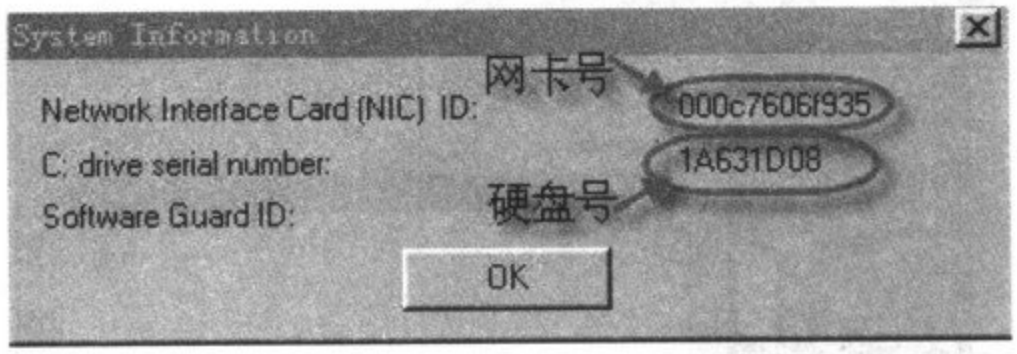


图 3-7　硬盘 ID 号

为网卡号。

另外，也可采用以下方法获取 NIC 号：进入 MS-DOS；在命令提示行下键入ipconfig/all命令；在出现的画面中可以看到 Physical Address 00-0C-76-06-F9-35，00-0C-76-06-F9-35 即为 NIC 号。

二、MAX＋plusII 的使用

下面，我们分别采用原理图输入方式和 Verilog-HDL 语言设计一个简单的 3 人表决器，并下载到 DP-MCU/Altera 实验仪进行实际运行。功能虽然简单，但是大家可以从这个实验中可以学习到 MAX＋plusII 的使用方法和技巧，并从中认识到 CPLD 的设计输入、仿真、下载等一个完整的过程。

3 人表决器的功能描述如下：3 个人分别用手指拨动开关 K1、K2、K3 来表示自己的意愿，如果对某决议同意，各人就把自己的指拨开关按下（此时开关接低电平），不同意则不按（此时开关接高电平）。表决结果用 LED 进行显示；如果对某个决议有任意 2～3 人同意，那么此决议通过，L1 亮；如果对某个决议只有一个人或没人同意，那么此决议不通过，L2 亮。

使用 MAX＋plusII 设计 3 人表决器具体步骤如下：

1. 新建项目

由于 MAX+plusII 是以项目来管理工程文件的，因此，在 MAX+plusII 下的任何开发设计都是以项目开始的。双击桌面 MAX+plusII 软件的快捷图标，进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，出现如图 3-8 所示的新建项目对话框。

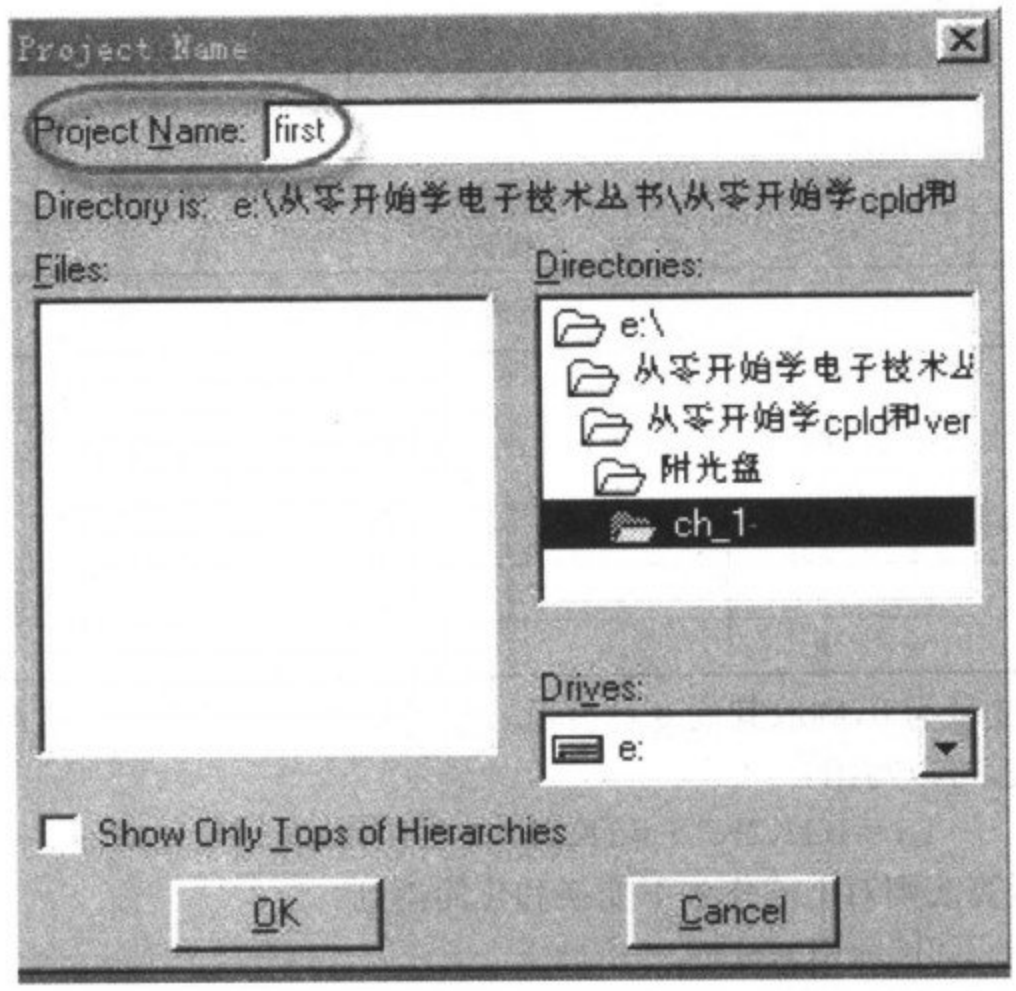


图 3-8　新建项目对话框

在 Project Name 对话框中键入设计项目名，如 first。若需要更改项目所属子目录，可以在 Drives 和 Directories 窗口中修改。单击 OK 按钮，此时 MAX+plusII 项目标题栏中将会出现新的项目名 first 及其所在的路径。

2. 设计输入

MAX+plusII 软件支持多种输入方式，其中以原理图输入方式和硬件描述语言输入方式 2 种较常见，而 HDL 硬件描述语言又包括 AHDL、VHDL 以及 Verilog-HDL 等描述语言。由于 Verilog HDL 语言在编程风格上具有灵活、简单、易学、易用的特点，同时又由于它充分保留了 C 语言简洁、高效的编程风格，因此，如果用户有一定的 C 语言编程经验，那么就可以在较短的时间内学习和掌握 Verilog-HDL 语言，本书中的所有程序都是用 Verilog-HDL 编写的。

依题意列出逻辑状态表，如表 3-1 所示。

对各状态进行赋值。“不按”赋以“1”，“按下”赋以“0”，L1“灭”赋以“1”，L1 亮赋以“0”。得到表 3-2 的真值表。

根据真值表写出逻辑表达式。通常是写出“与”、“或”表达式，即将逻辑真值表中输出 L1＝1 的各个状态，分别表示成全部输入变量的“与”函数(如果输入变量为 1，则用其本

表 3-1　3 人表决器逻辑状态表

按　钮			指示灯 Y	
K1	K2	K3	L1	L2
不按	不按	不按	灭	亮
不按	不按	按下	灭	亮
不按	按下	不按	灭	亮
不按	按下	按下	亮	灭
按下	不按	不按	灭	亮
按下	不按	按下	亮	灭
按下	按下	不按	亮	灭
按下	按下	按下	亮	灭

表 3-2　3 个表决器真值表

输　入			输出 L1	输　入			输出 L1
K1	K2	K3		K1	K2	K3	
1	1	1	1	0	1	1	1
1	1	0	1	0	1	0	0
1	0	1	1	0	0	1	0
1	0	0	0	0	0	0	0

身表示；若输入变量为 0，则取其反变量表示），总的输出 L1 则表示成这些“与”项的“或”函数。对于上表，可以写出：

$$L1=K1K2K3+K1K2\overline{K3}+K1\overline{K2}K3+\overline{K1}K2K3$$

运用逻辑运算法则对上式作如下变换和化简得到：

$L1=K1K2+K2K3+K3K1$

由于 L2 与 L1 相反，因此：

$$L2=\overline{L1}$$

下面以原理图输入和 Verilog-HDL 语言两种方式，分别讲解 MAX+plusII 的设计输入方法。

(1)采用原理图进行输入

① 单击 File→new，新建文件时选择 Graphic Editor file，如图 3-9 所示。

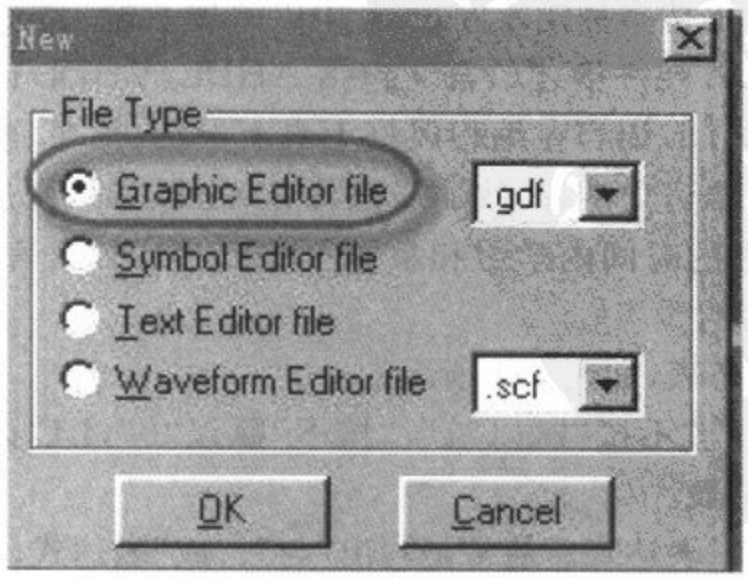


图 3-9　新建图形文件

② 点击 OK，出现如图 3-10 所示原理图编辑窗口。

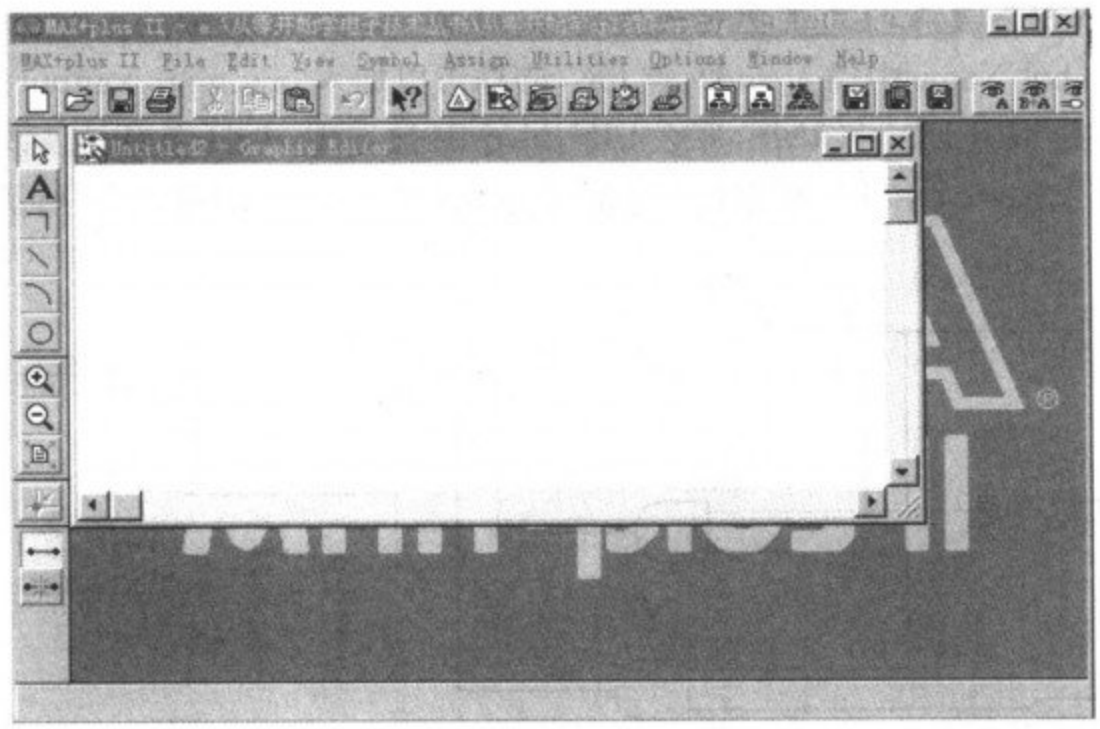


图 3-10　原理图编辑

③ 现在可以在图形文件中输入电路，这个电路需要 and2(2 输入与门)、or3(3 输入或门)、not(非门)3 个逻辑门电路和 input(输入)、output(输出)，点击菜单 Symbol→Enter Symbol 或者双击空白处，出现如图 3-11 所示的符号库对话框。在该对话框中可以选择需要输入的元件/逻辑符号。

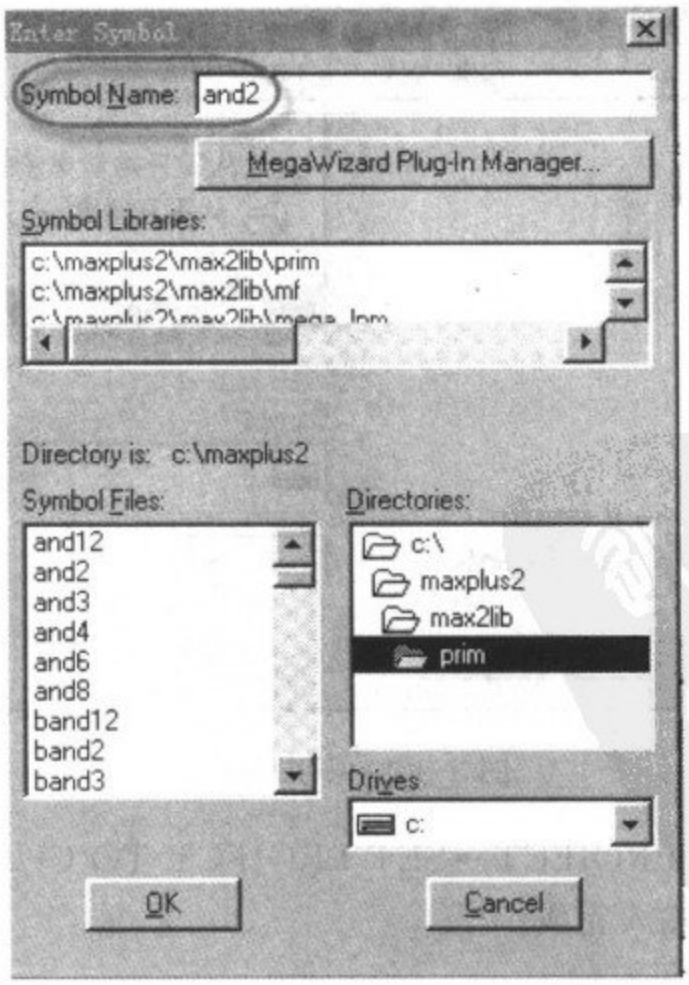


图 3-11　符号库对话框

重点提示　MAX+plusII 为实现不同的逻辑功能，提供了大量的库文件，每个库对应一个目录，这些库根据其功能大小及特点可分为：用户库(用户自建的元件)、prim(基本

库，放入基本逻辑电路，如门、触发器等）、mf（宏功能库，包括所有 74 系列元件）、mega（可调参数库，放入功能复杂的高级功能模块）和 edif 库（和 mf 库类似）。

④ 在 Symbol Name 中输入 and2，点 OK，同样可以加入 or3、input、output、not，和 input、output，鼠标左键双击 PIN_NAME，那么 PIN_NAME 被选中，并且变黑，然后输入你要改的名字，如 K1、K2、K3、L1 和 L2，根据逻辑表达式绘制好的原理图如图 3-12 所示。

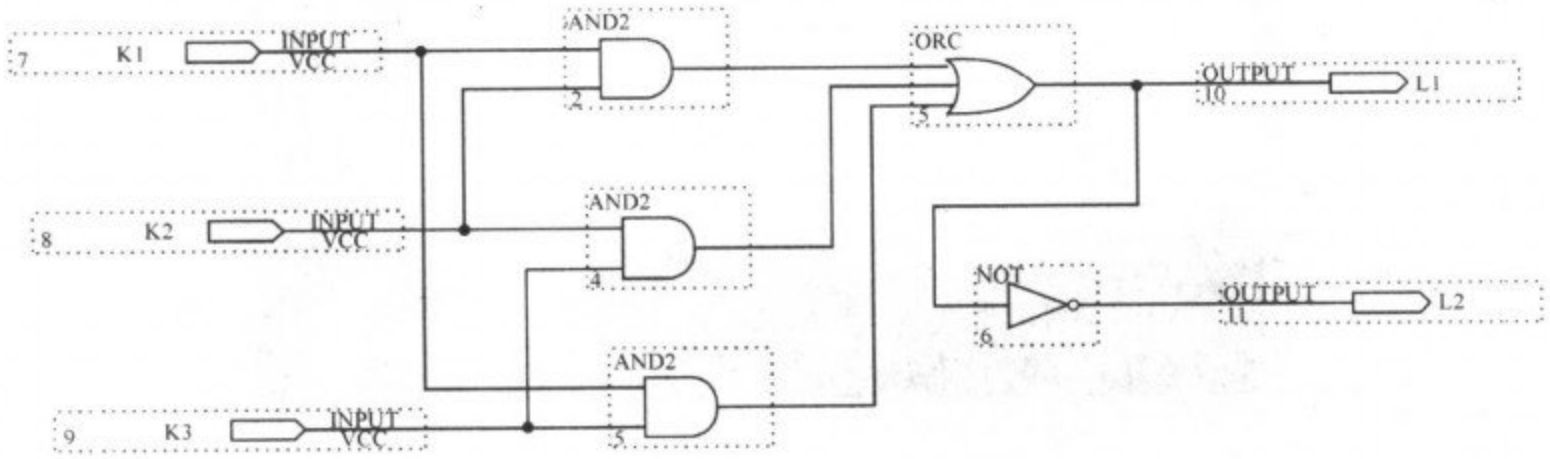


图 3-12　3 人表决器原理图

⑤ 单击工具栏中的保存按钮，将文件保存为 first. gdf 文件，如图 3-13 所示。

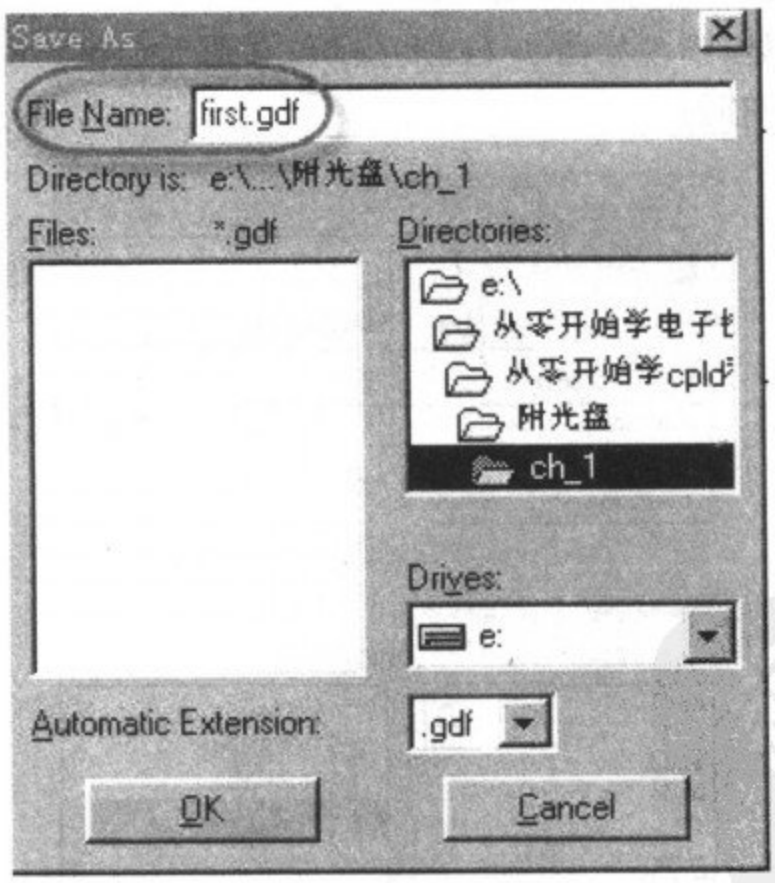


图 3-13　保存原理图

⑥ 单击菜单 FILE→PROJECT→SET PROJECT TO CURRENT FILE，把文件设为当前工程至此，原理图输入完成。

（2）用 Verilog-HDL 语言输入

① 单击单击 File→new，新建文件时选择 Text Editor File 选项，如图 3-14 所示。点击 OK，出现文本编辑窗口。

方法技巧　为了快速和准确地创建 Verilog-HDL 源程序，MAX＋plusII 提供了

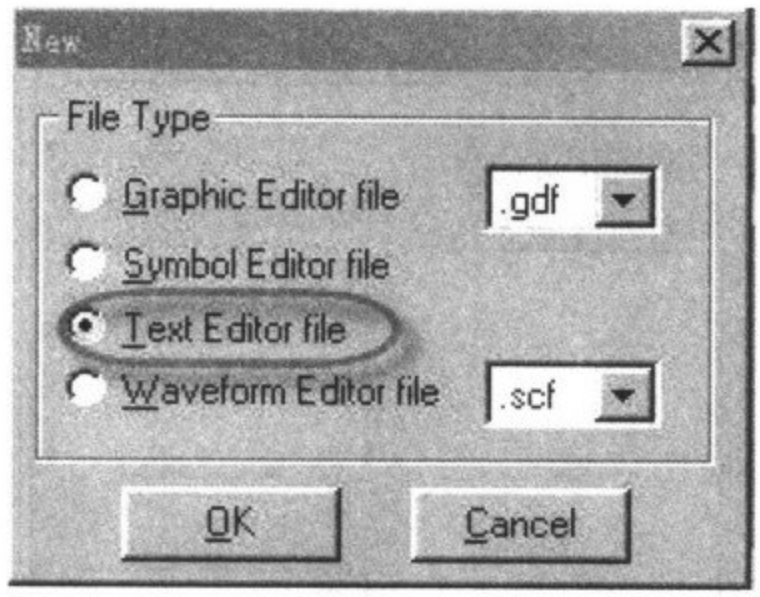


图 3-14　新建文本文件

Verilog 模板。有了模板的提示和帮助，就可以很轻松地完成程序的输入。单击菜单栏中的 Template 选项，在出与现的菜单中选择 Verilog Template 命令，然后在出现的选择窗口中选择 Module Delaration 选项，这时，在 Text Editor 窗口中将出现一个标准的 Verilog 模板，在相应的位置上输入用户的源代码即可。

② 输入以下 Verilog 源程序，其中 K12，K13，K23 为中间变量：

```
module first(K1,K2,K3,L1,L2);
output L1,L2;
input K1,K2,K3;
and(K12,K1,K2);
and(K13,K1,K3);
and(K23,K2,K3);
or(L1,K12,K13,K23);//K12、K23、K13 是中间变量
not(L2,L1);
endmodule
```

③ 单击工具栏中的保存按钮，将文件保存为 first. v 文件，如图 3-15 所示。

④ 单击菜单 File→Project→Set Project To Current File，把文件设为当前工程至此，Verilog 输入完成。

3. 器件选择与引脚锁定

编辑完源程序文件后，可以选择相应的器件，并对各引脚进行分配。

(1)执行菜单栏中的 Assign→Device 命令，出现如图 3-16 所示的器件选择对话框；在 Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。这时用户要做的就是根据自己硬件电路的需要，对各输入/输出引脚进行锁定。

需要说明的是，在对话框中，要把 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

(2)执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出如图 3-17 所示的引脚锁定对话框。在 Node Name 栏内输入信号端口的名称 K1；然后在 Chip Resoure 栏中选择 Pin，根据需要输入要锁定的引脚序号 56。在 Pin type 中输入 input，这时的 Add

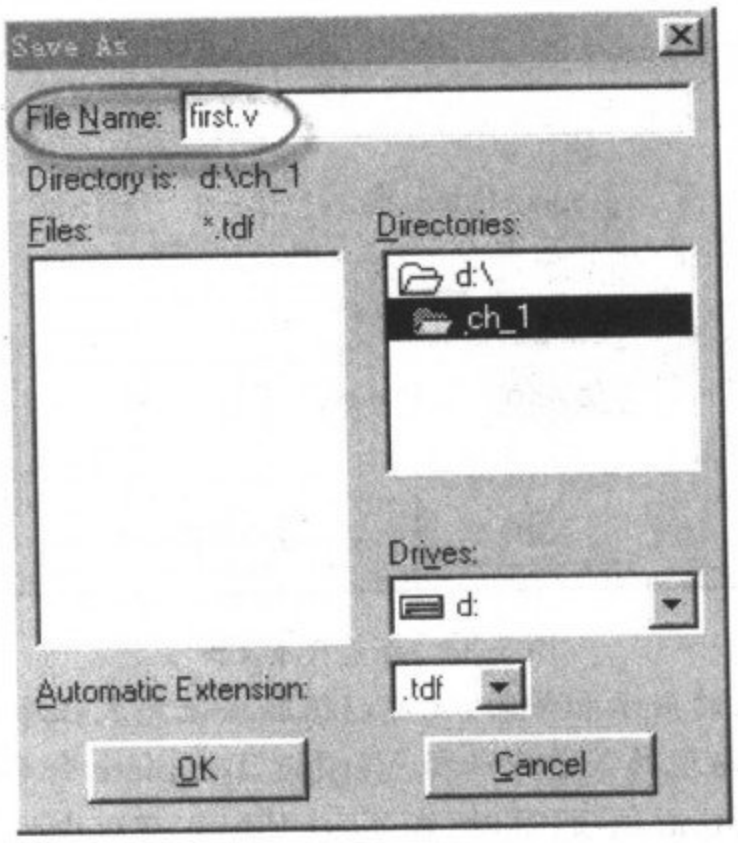


图 3-15　保存文件

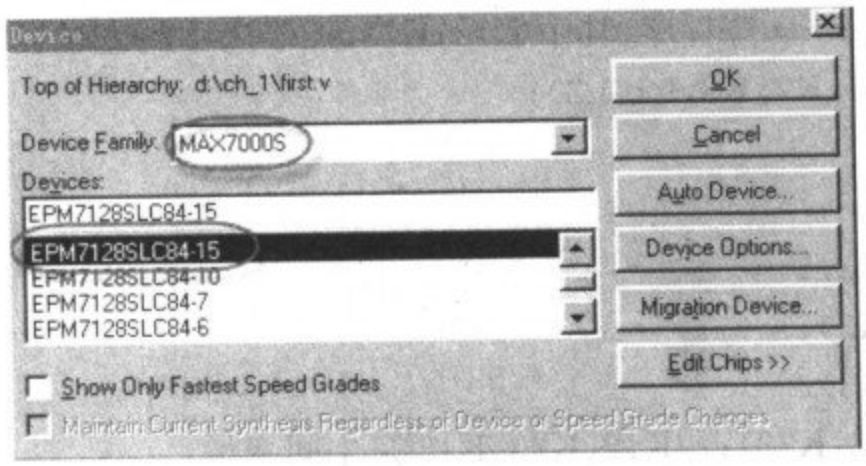


图 3-16　选择器件对话框

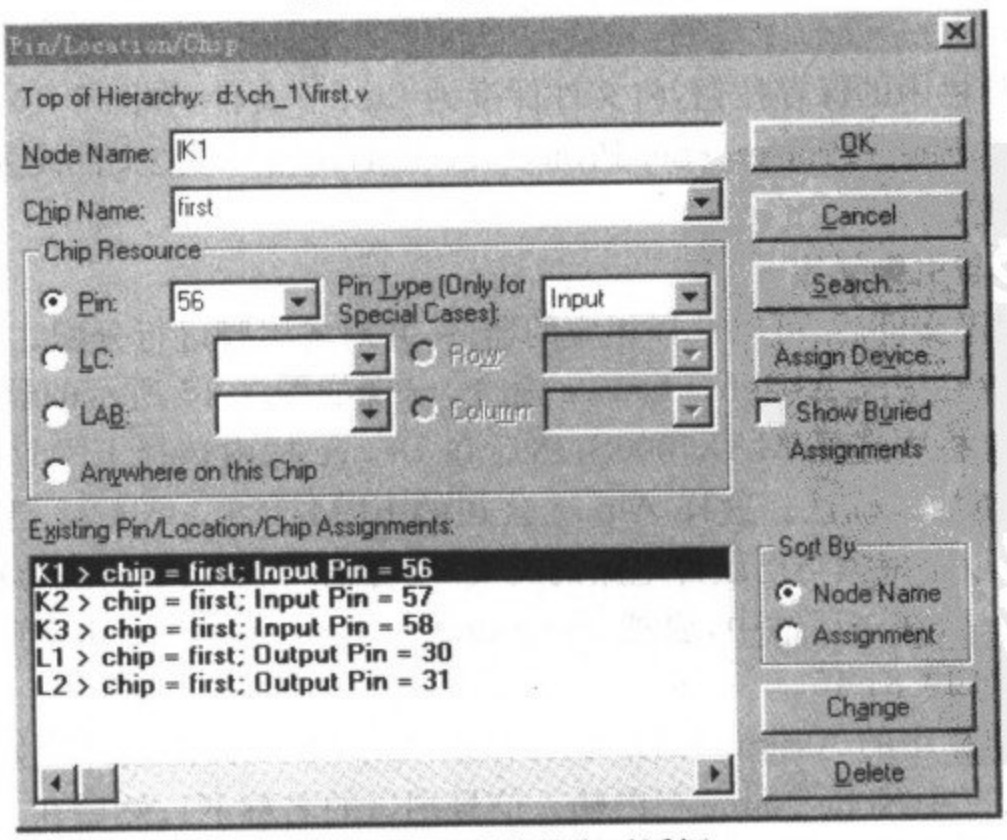


图 3-17　引脚锁定对话框

按钮呈现有效状态，单击它即可把刚才输入的引脚及其信号添加到 Existing Pin/Location/Chip Assignments 窗口中。依此方法锁定其他剩余引脚，锁定的引脚如表3-3所示。锁定完毕单击 OK 即可。

表 3-3 锁定引脚

Node name(脚名)	pin(脚号)	Pin type(输入输出类型)
K1	56	input
K2	57	input
K3	58	input
L1	30	output
L2	31	output

4. 器件的编译

MAX+PLUS II 编译器的作用是检查项目是否有错，并对项目进行逻辑综合，然后对设计进行布局布线，放到一个 Altera 器件中，同时将产生报告文件、编程文件和用于时间仿真用的输出文件。逻辑综合就是把 HDL 语言或原理图翻译成最基本的与、或、非门的连接关系；布局布线就是把这种与、或、非门的连接关系用芯片的内部的可编程结构和连线来实现，编译是整个设计较重要的环节。

(1)执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出如图 3-18 所示的编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。此时可能弹出下图 WARNING 窗口，提示“project has user pin or logic cell assignments, but never been compiled before. For best fitting results, let the compiler choose the first set of assignments instead”，这是因为在管脚指定之前没有 compile，你只要再点 save & compile 就可以了。

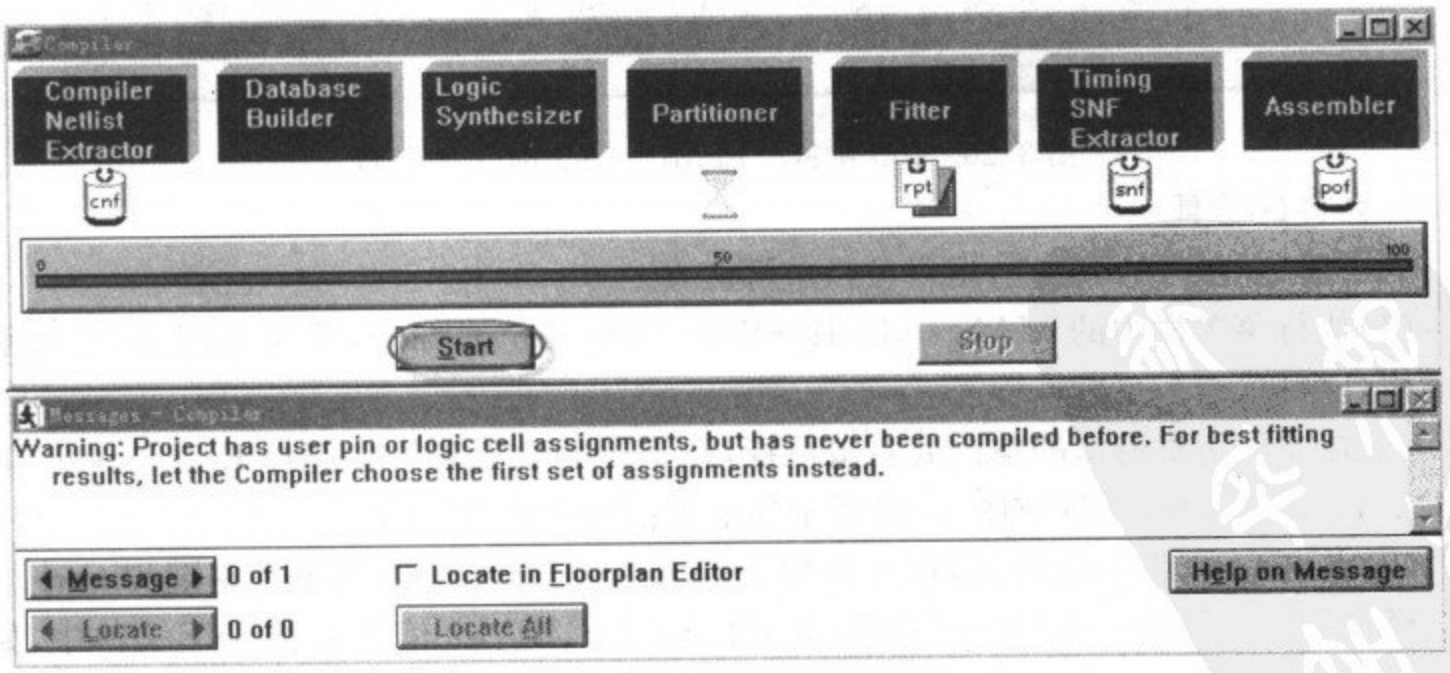


图 3-18 器件的编译

(2)若在编译过程中出现错误，将在提示信息栏内给出错误信息，双击此信息即可返回到程序输入窗口并指向错误处。用户应更正错误直至编译完全正确为止，若编译成功，弹出如图 3-19 所示的编译成功窗口，点击确定按钮加入确认。

图 3-19　编译成功窗口

(3)执行 MAX+plusII→Froorplan Editor 命令,可以看到编译后 I/O 在芯片上的分布,双击空白处,可以看到如图 3-20 所示 I/O 在芯片 EPM7128SLC84 上的布局。

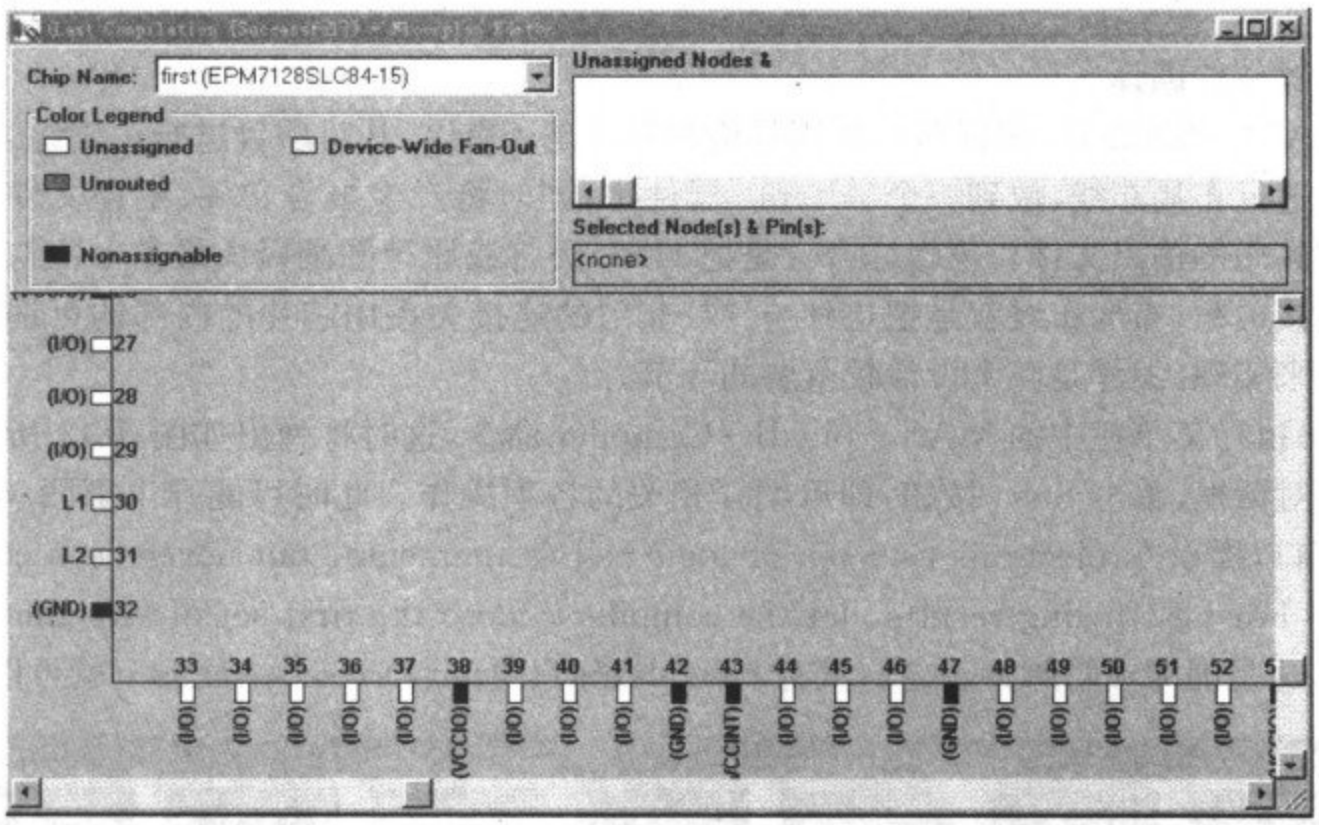


图 3-20　I/O 在芯片 EPM7128SLC84 上的布局

5. 器件的仿真

为了验证程序的可靠性,用户可以进行仿真。

(1)执行菜单栏中的 MAX+plusII→Waveform Editor 命令,弹出如图 3-21 所示的波形仿真窗口。

(2)在波形仿真窗口的空白处右击鼠标,在弹出的快捷菜单中选择 EnterNodes form SNF... 命令,这时将出现如图 3-22 所示的仿真信号选择对话框。

(3)点击 List,将出现端口列表,你默认是选择全部,你也可以通过左键和 Ctrl 组合来选择你想要的信号。点"=>"图标,将 K1、K2、K3、L1、L2 信号加入 SNF 文件中,如图 3-23 所示。

(4)对信号进行赋值,这里,我们对 K1、K2、K3 输入信号赋于周期值。

先选中 K3,点击 XC 按钮,弹出如图 3-24 所示的对话框,在对话框中,将 Starting Value 开始电平设为 1(高电平),Increment By 增加值设为 0(低电平),Multiplied By 设为 200(即半周期为 20μs)。点击 OK 按钮。

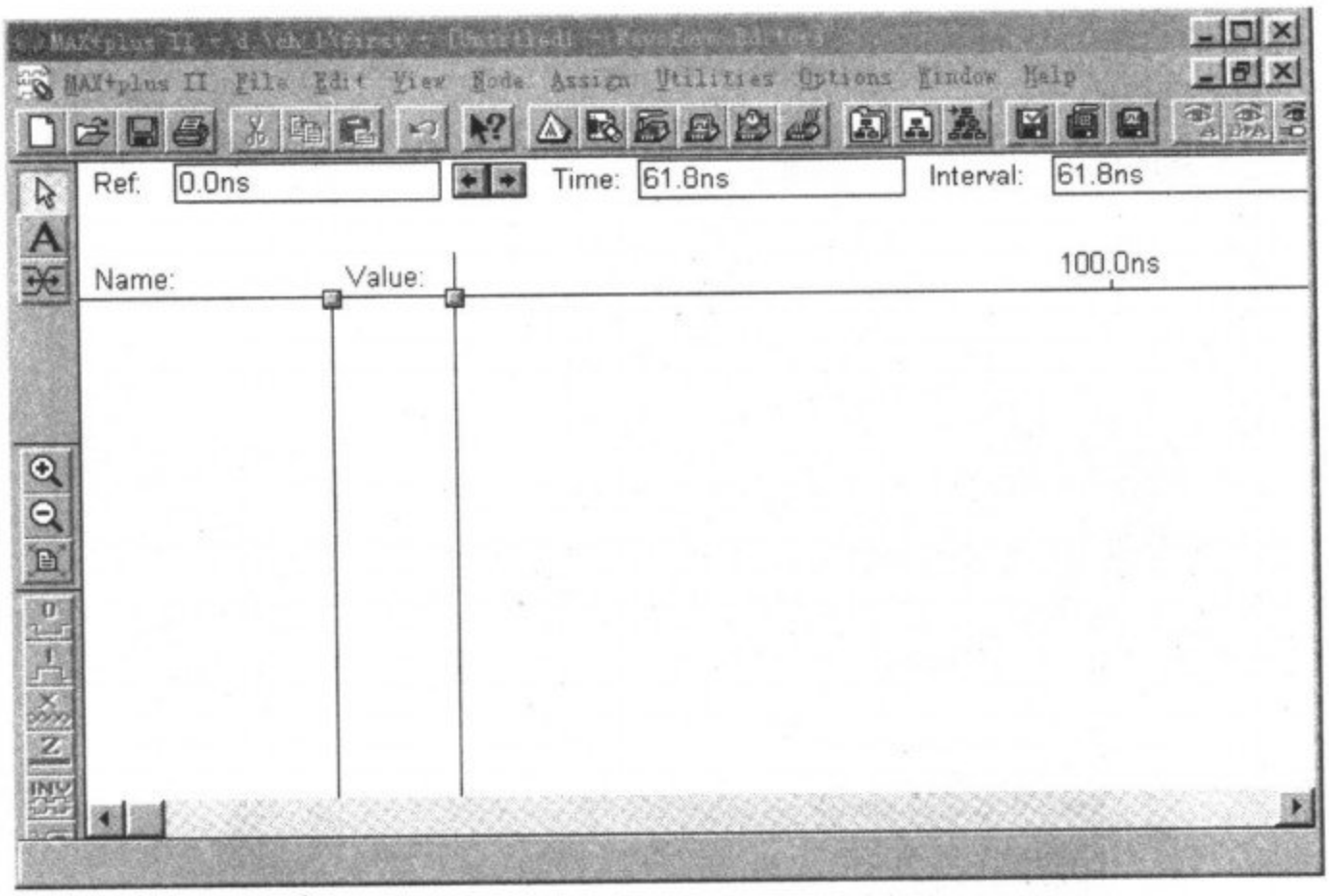


图 3-21　波形仿真窗口

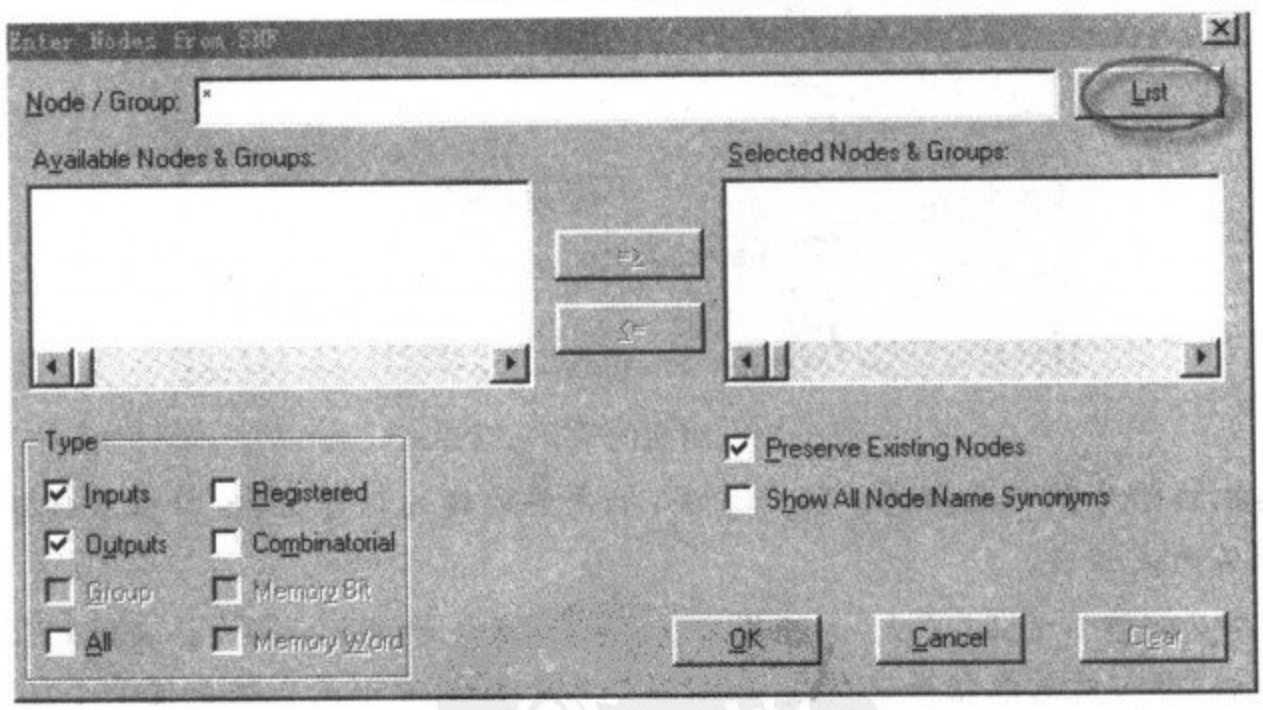


图 3-22　仿真信号选择对话框

在图 3-24 中，To 100μs 是此周期信号的终了时间，终了时间可用菜单命令 File→End Time 进行改变。

方法技巧　各赋值工具功能如下。

：把该信号置为低电平。操作方法：单击要编辑的信号，然后单击该按钮，即可将该信号赋值为低电平。

：把该信号置为高电平。操作方法：单击要编辑的信号，然后单击该按钮，即可将该信号赋值为高电平。

：把该信号置为任意电平，表明与该信号无关。操作方法：单击要编辑的信号，然后单击该按钮，即可将该信号赋值为任意电平。

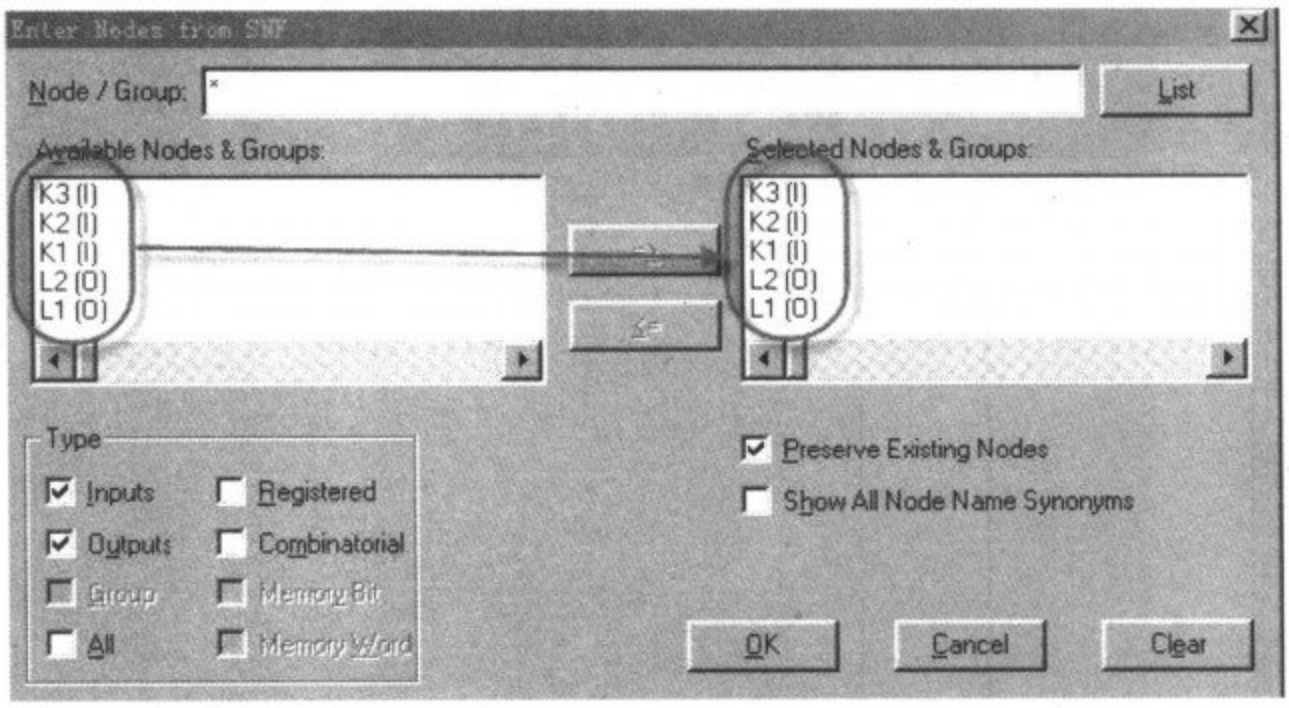


图 3-23　端口列表

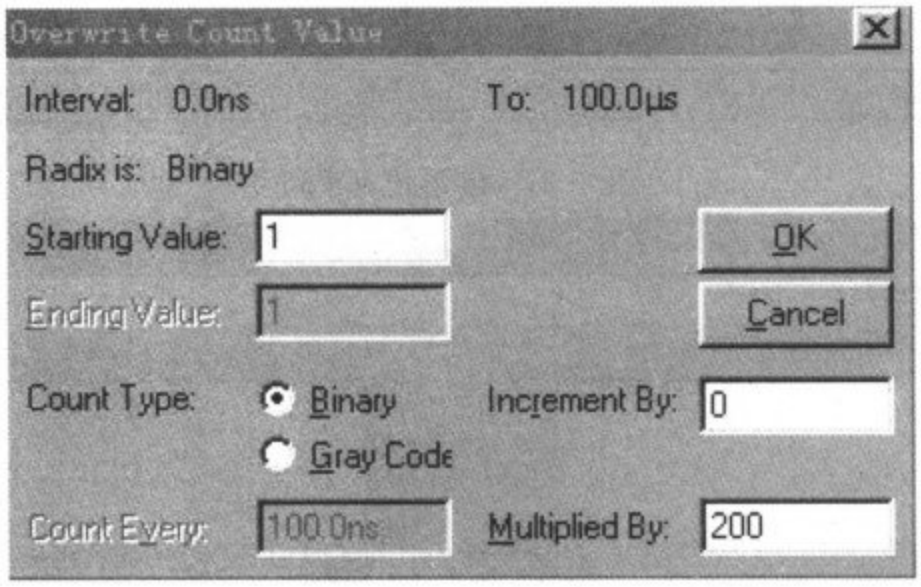


图 3-24　周期信号设置对话框

：把该信号置为高阻态。操作方法：单击要编辑的信号，然后单击该按钮，即可将该信号赋值为高阻态。

：可以把该信号值取“反”。操作方法：单击(或拖动鼠标选择)要编辑的信号，然后单击该按钮，即可对选取信号进行逻辑取“反”，即选择的波形原来若是低电平“0”，则会转变为高电平“1“，反之亦然。

：把该信号赋值为一个时钟信号。操作方法：单击要编辑的信号，然后单击该按钮，即可将该信号赋值为一个时钟信号源。

：同样表示把该信号赋值为一个周期性信号。操作方法：单击要编辑的信号，然后单击该按钮，即可弹出一个设置对话框，可以对选择的波形赋予指定周期的周期信号。

(5)按以上方法将 K2 设置为开始电平设为 1(高电平)，增加值设为 0(低电平)，Multiplied By 设为 100(即半周期为 10μs)。将 K1 设置为开始电平设为 0(低电平)，增加值设为 1(高电平)，Multiplied By 设为 100(即半周期为 10μs)。设置好的波形图如图 3-25 所示。

(6)单击保存按钮，将文件保存为 first. scf 文件，如图 3-26 所示。

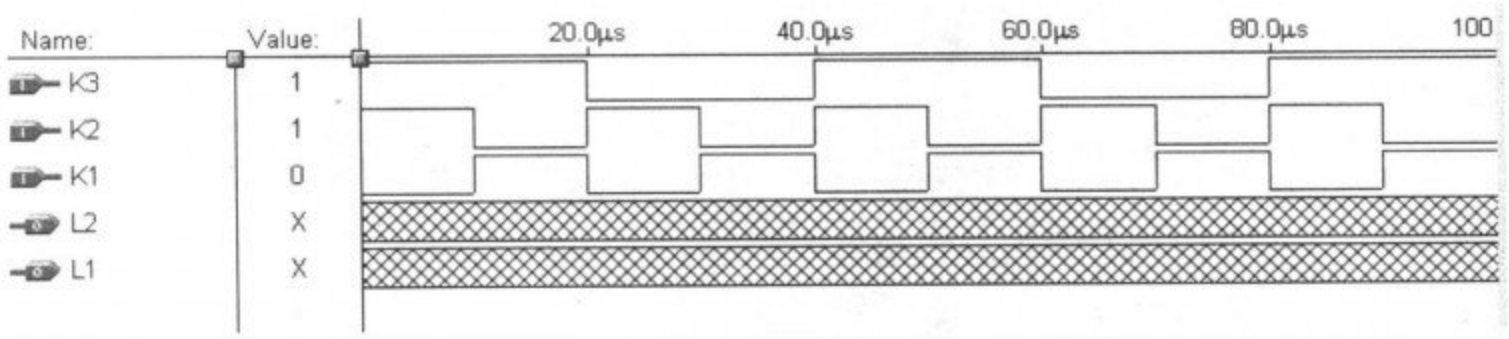


图 3-25　设置好的周期性输入波形

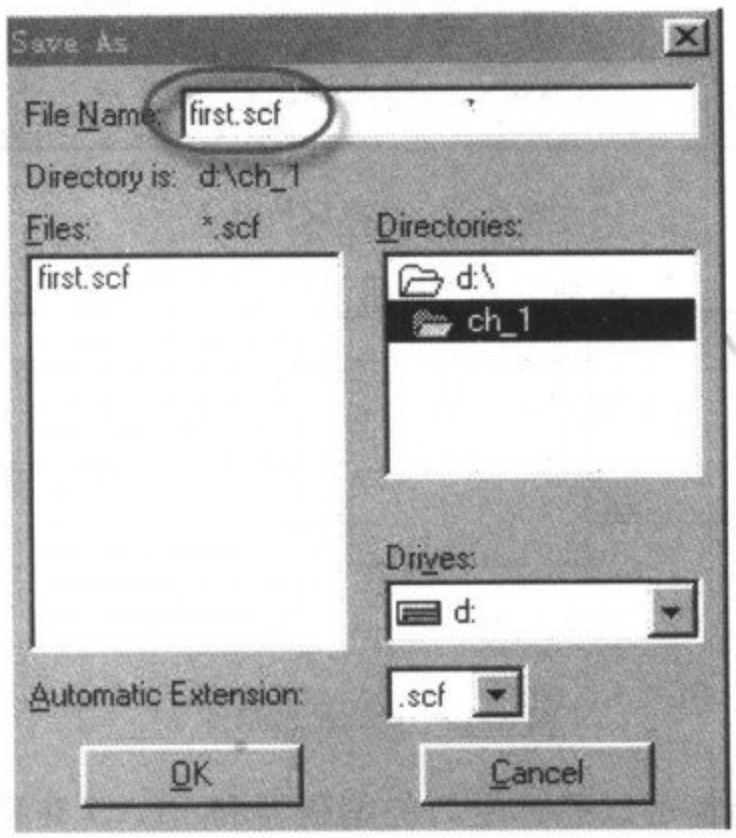


图 3-26　保存波形图文件

(7)执行菜单栏中的 MAX+plusII→Simulator 命令，弹出如图 3-27 所示的对话框。

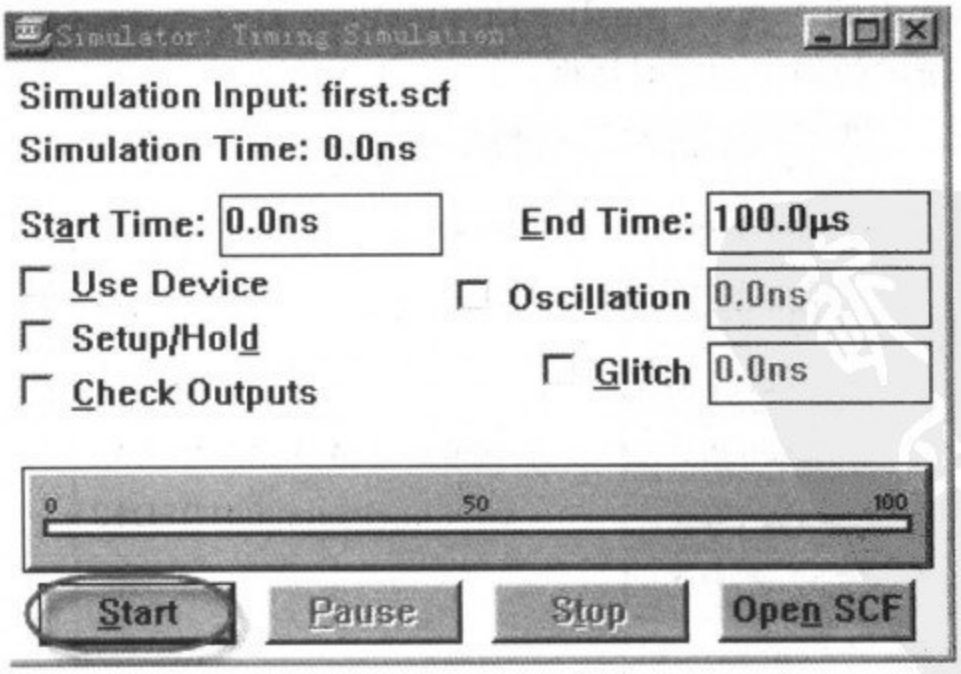


图 3-27　仿真对话框

(8)点击 Start 开始按钮，开始仿真，如果没有错误，将弹出如图 3-28 所示的仿真成功窗口。

(9)仿真的波形图如图 3-29 所示。波形图显示的逻辑功能和设计目的完全一样。

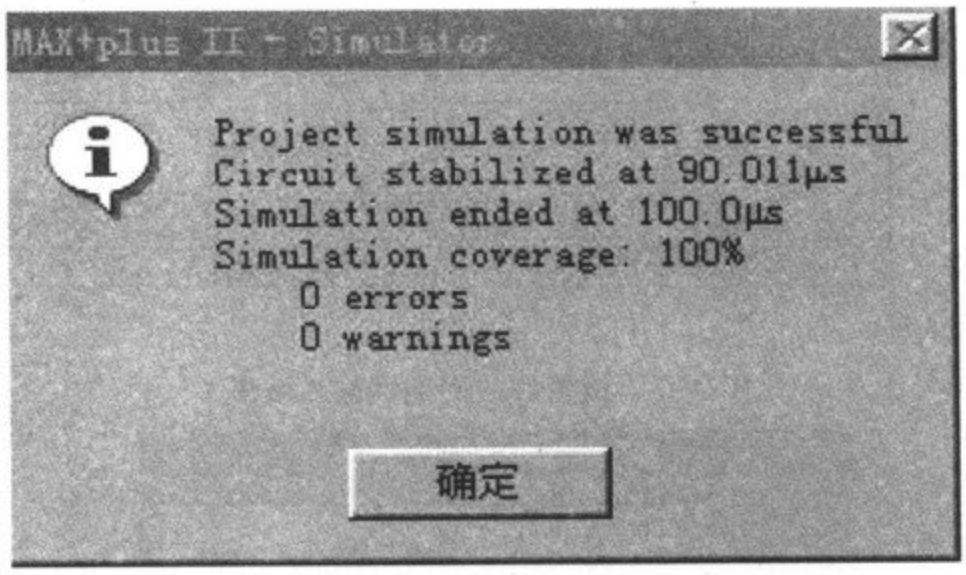


图 3-28　仿真成功窗口

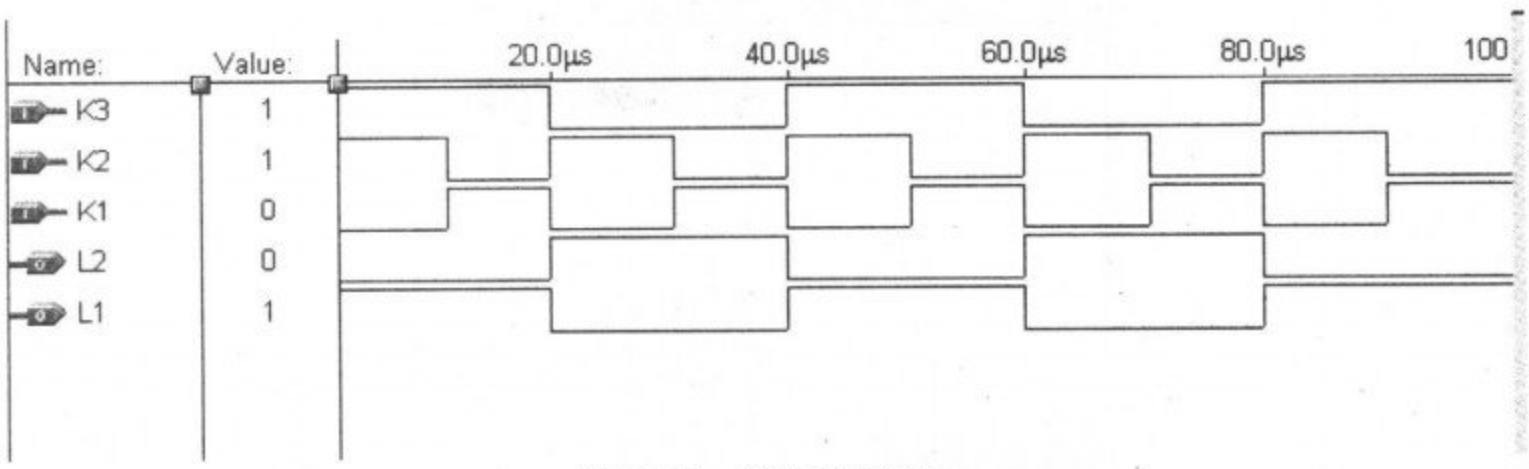


图 3-29　仿真的波形图

6. 器件编程下载

(1)Altera 器件的编程可以通过专门的编程器和 JTAG 在系统编程等多种方法实现。取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。

(2)执行菜单栏中的 MAX＋plusII→Programmer 命令，对器件进行在线编程，这时将弹出如图 3-30 所示的编程命令对话框。

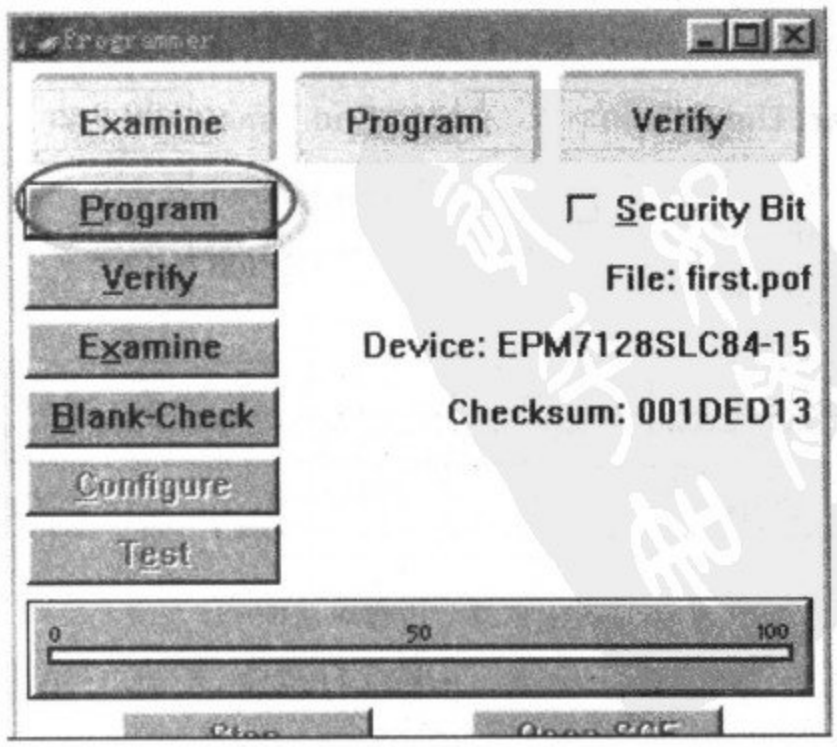


图 3-30　编程对话框

(3)执行菜单栏中的 Option→Hardware Setup,将弹出对下载口进行设置窗口,如图3-31所示;从弹出的窗口中选择 ByteBlaster[MV](JTAG 方式进行编程),单击 OK 按钮确认。

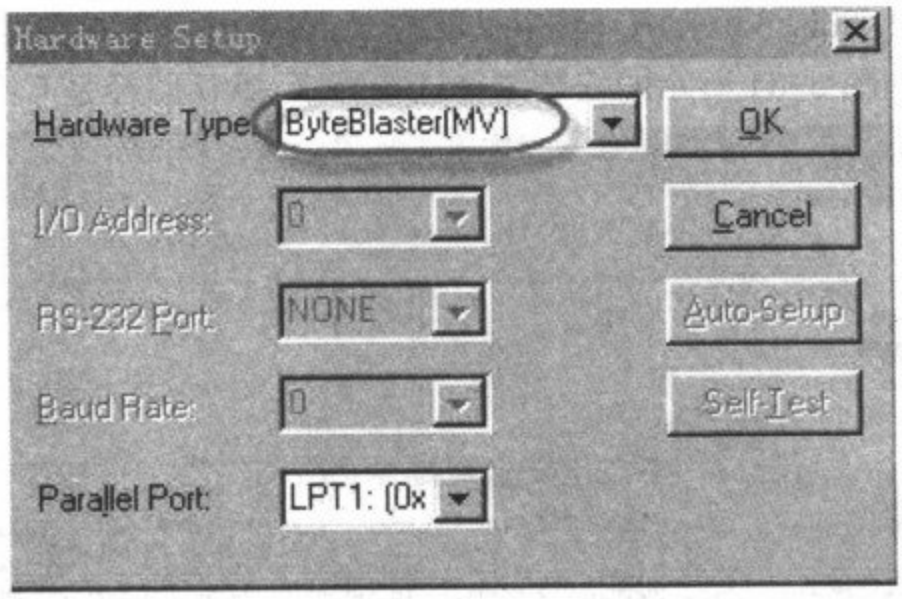


图 3-31　下载口设置窗口

(4)在图 3-30 所示的窗口中,选择 Program 命令就可开始对器件进行编程(若用户需要对程序加密处理,可选择 SecurityBit 选项)。可以看出,下载编程主要包括 Examine、Program 和 Verify 三大步,编程完毕后出现如图 3-32 所示的下载完成信息。

现在你就可以通过实验板上的硬件资源来验证表决器功能是否正确。通过测试,设计的功能完全正确。至此,一个完整的项目开发完成了。

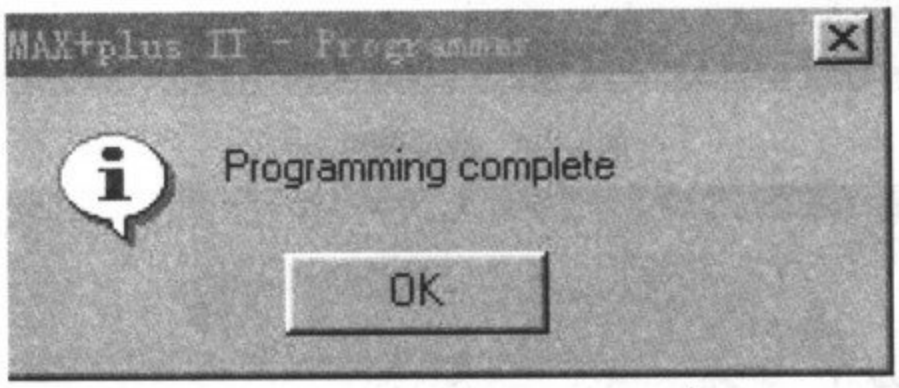


图 3-32　下载完成

重点提示　继 MAX+plusII 之后,Altera 公司又发布了的新一代 PLD 开发系统 QuartusII,QuartusII 比 MAX+plusII 有更多的功能,支持更大规模 PLD 的开发,安装要求也较高,软件可以从 Altera 网站下载。

需要说明的是,MAX+plusII 对 VHDL 和 Verilog 语言的支持有限,有不少语法不能支持,如果遇到这种情况,建议使用 QuartusII 或者专用综合工具进行逻辑综合。关于 QuartusII 的具体使用方法,本书不作介绍,有兴趣的读者可参考相关书籍。

第二节　Xilinx 开发软件ISE WebPACK 的安装和使用

ISE WebPACK 是 Xilinx 公司的开发软件,可以从 Xilinx 公司的网址http://www.xilinx-china.com/进行下载,ISE WebPACK 的版本较多,目前最新版为 7.1i,这里以 6.3 版本为例进行介绍。

一、WebPACK 软件的安装

(1)双击 WebPACK_63_fcfull_i 图标,出现如图 3-33 所示的软件授权画面,将 I Accept The Terms Of This Software License 前打√,点击下一步。

(2)接着出现如图 3-34 所示的安装路径选择界面,默认路径是 C:\Xilinx。当然,

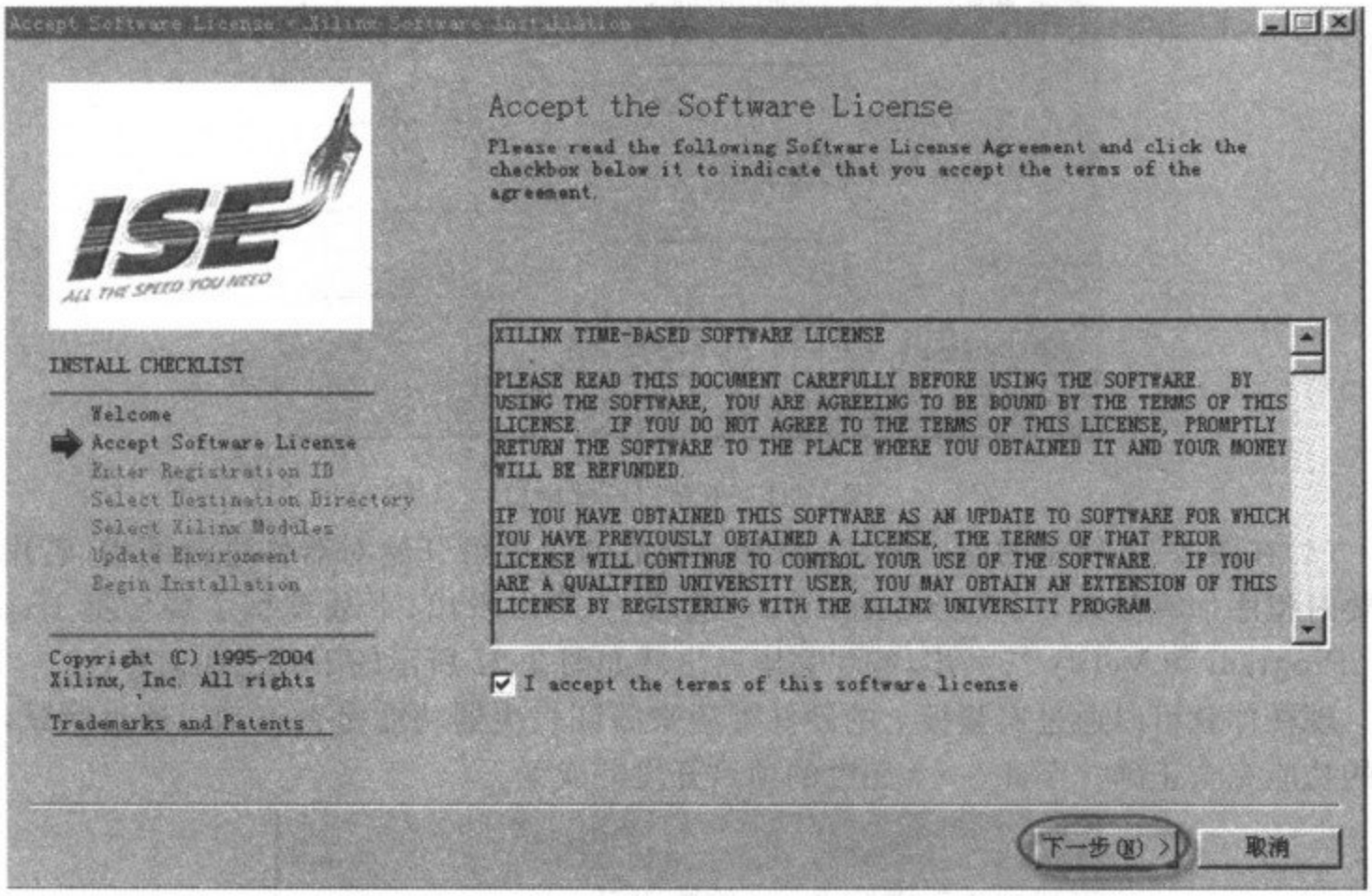


图 3-33　软件授权画面

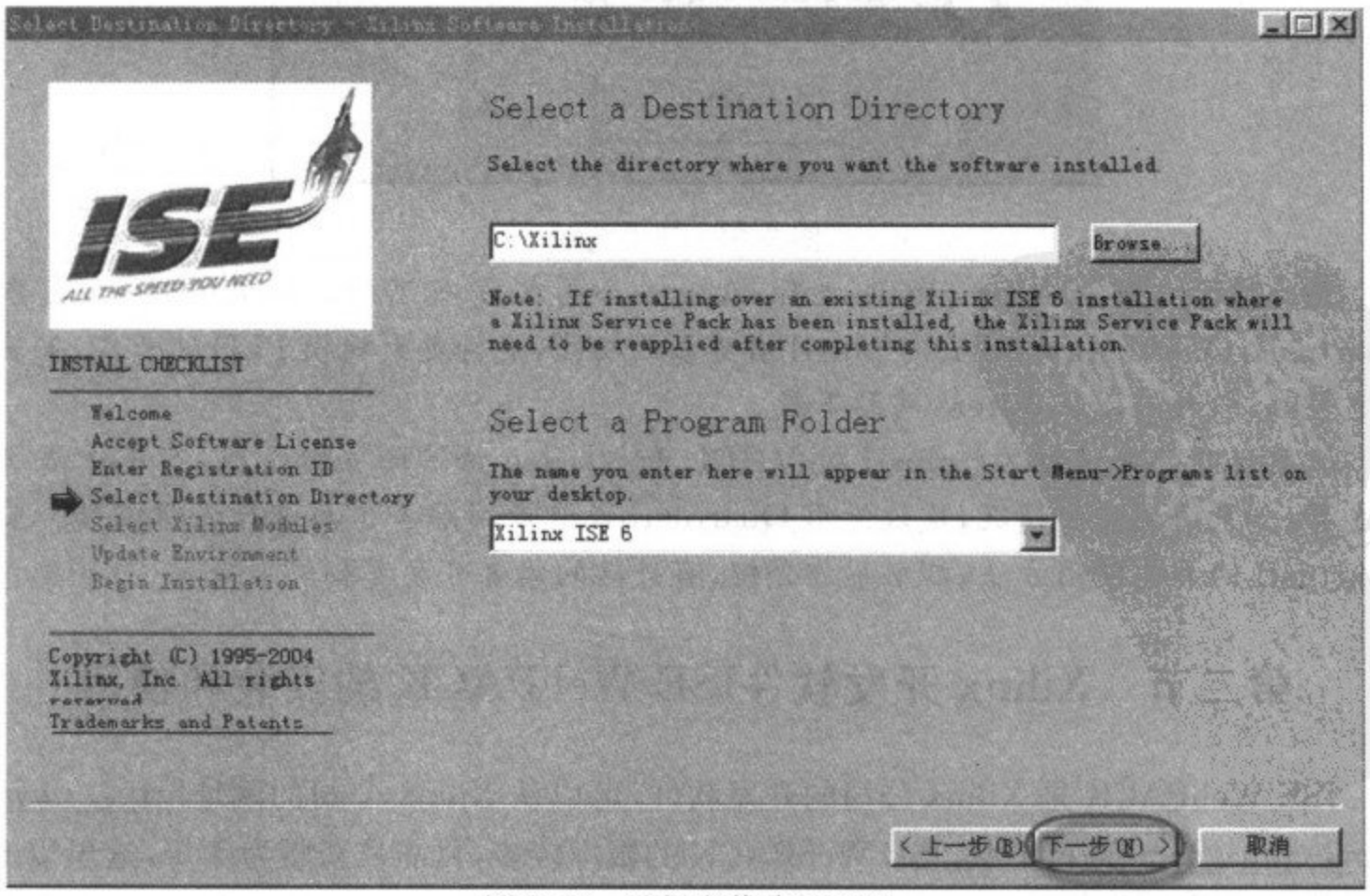


图 3-34　选择安装路径画面

也可以单击“Brower...”按钮来选择适合自己的安装目录，这里选择默认路径。然后进行下一步。

(3)在出现的开始安装画面中，单击 install，出现安装进度指示界面，提示用户该软件的安装进度，等待安装完毕，如图 3-35 所示。

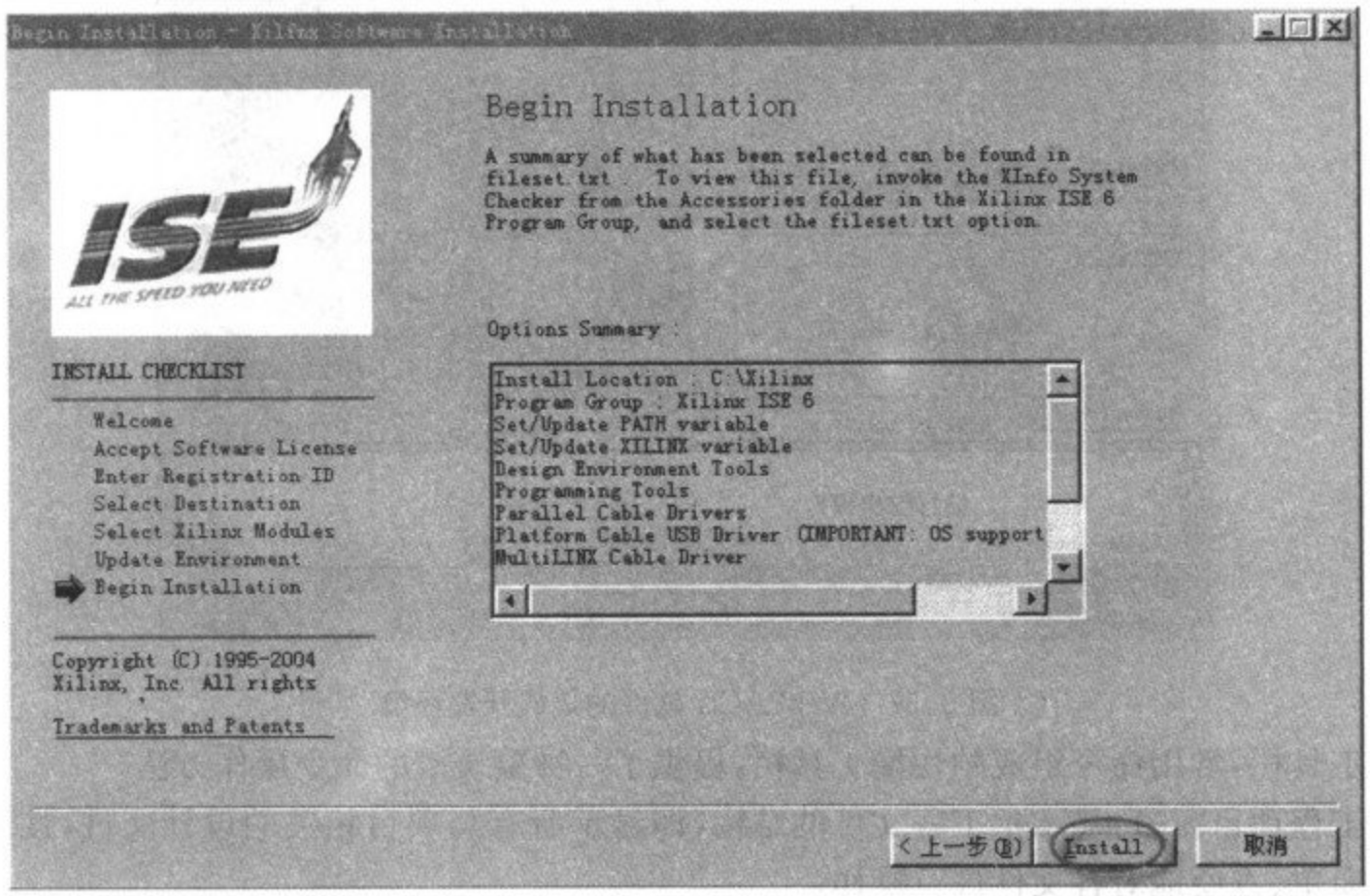


图 3-35　开始安装画面

(4)安装完毕后，在桌面上生成 WebPACK Project Navigator 快捷图标，如图 3-36 所示。

图 3-36　WebPACK Project Navigator 快捷图标

二、WebPACK 软件的使用

双击桌面的快捷图标即可进入如图 3-37 所示的 WebPACK 软件的集成开发环境。从图中可以看出，WebPACK 集成开发环境主要包括以下几部分。

项目标题栏：显示当前正在操作的项目的名称及其所在的路径。

菜单栏：提供了的命令菜单，所有的命令都包括在这些菜单中。

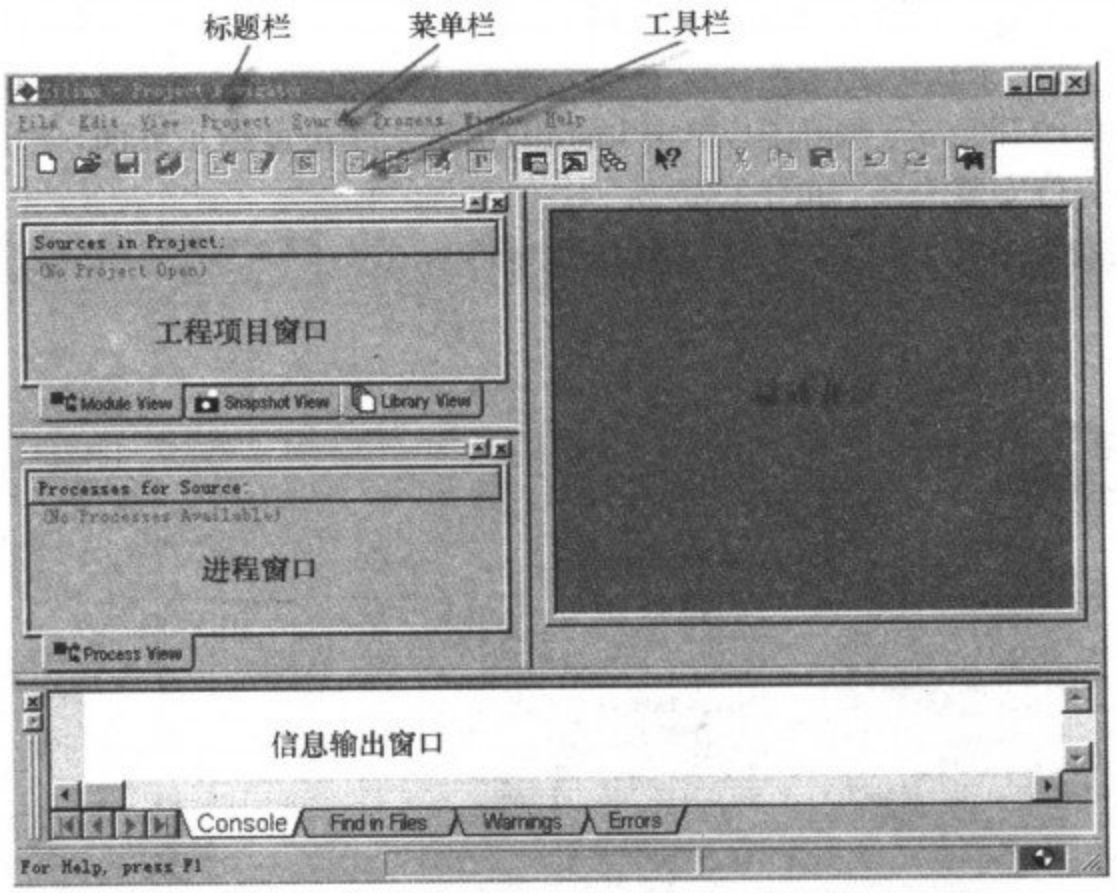


图 3-37　WebPACK 软件的集成开发环境

工具栏：常用命令组成的快捷工具栏，提供了一种更快捷的命令操作方法。

工程项目窗口：显示本工程文件的结构，即显示所有与项目相关的设计文件，使用户能够简单、方便地对各文件进行操作。

进程窗口：显示在工程项目窗口中所选源程序的请求全过程。过程包括综合、测试工作台发生、仿真、实现、器件编程、报告生成或其他逻辑设计步骤。

编辑窗口：源程序文件的编辑和修改。

信息输出窗口：显示处理过程运行结果的全部信息、错误提示信息和警告提示信息等。

下面，仍以前面介绍的三人表决器程序为例，介绍 WebPACK 软件的使用方法和技巧。

1. 新建项目

WebPACK 软件是以工程来管理项目及其项目文件的，因此，在 WebPACK 下的任何设计必须是从工程文件开始的。

(1)单击菜单栏的 File→New Project 命令，这时会出现如图 3-38 所示的新建工程文件对话框。

在 Project Name 栏内输入该工程的文件名，如 second，单击 Project 栏右边的浏览按钮，选择该工程要存放的路径。

(2)单击图 3-38 中的下一步按钮，出现如图 3-39 所示新建工程属性和语言选择对话框。

在 Value 菜单中可对工程文件进行基本的属性设置，其中包括器件的类型、器件的具体型号和所用的编程语言。当然，器件的类型和具体型号必须按照所用的目标器件来选择，这里选择使用 Xilinx 公司的 XC9500 系列芯片 XC95108(PLCC84 型封装)，单击 Val-

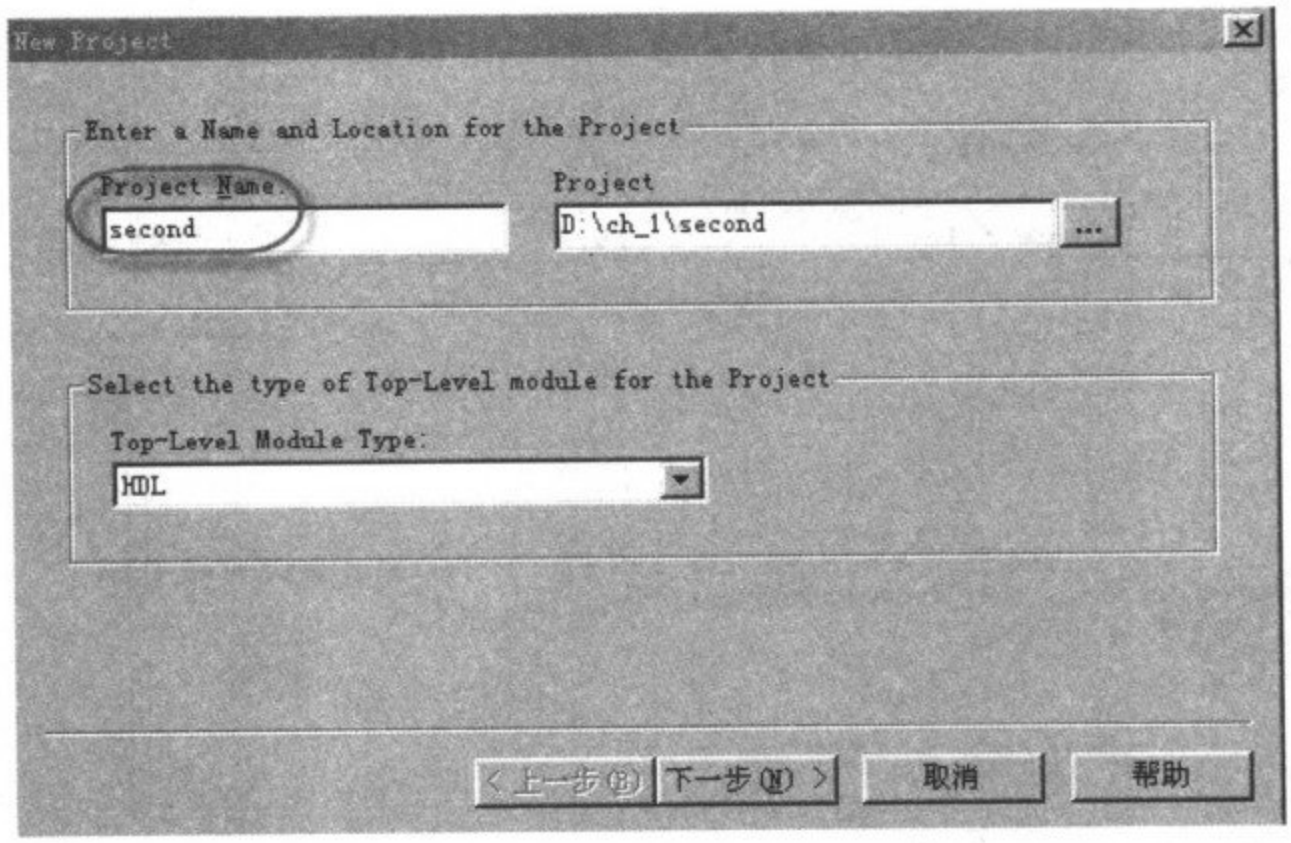


图 3-38　新建工程文件对话框

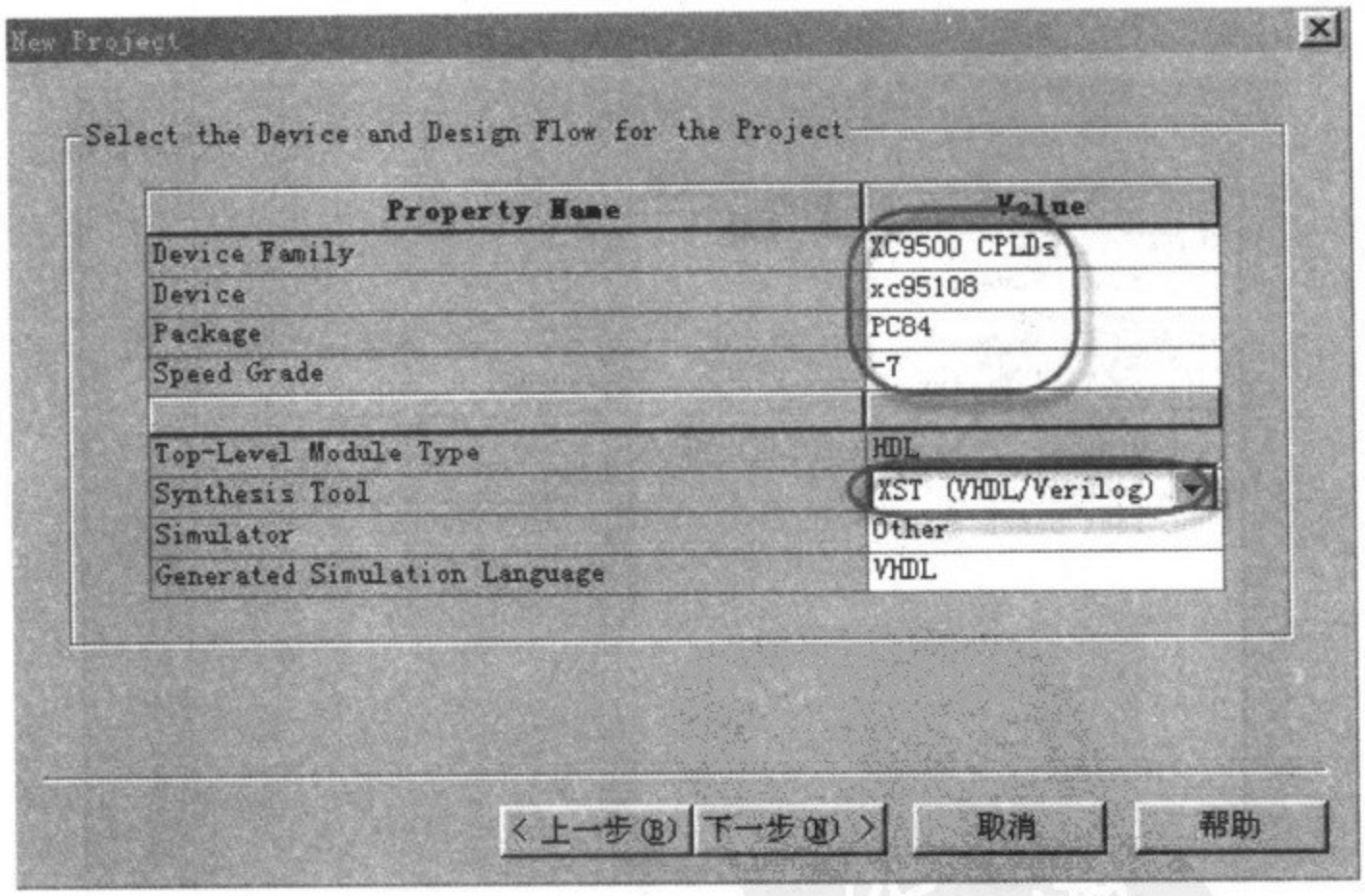


图 3-39　新建工程属性和语言选择对话框

ue 菜单中的相应选项，在弹出的下拉菜单中可进行选择。

在 HDL 语言栏中可对编程语言进行选择，WebPACK 软件支持多种编程语言，如 VHDL 语言和 Verilog HDL 语言等。这里选择 XST VHDL/Verilog 选项。

(3)单击图 3-39 中下一步按钮，出现如图 3-40 所示建立源程序文件对话框。

(4)单击图 3-40 中的 New Source 按钮，出现如图 3-41 所示的新建源程序对话框。

对话框的左边栏内提供了可以选择源程序的输入方法，其中 Verilog Module 选项提

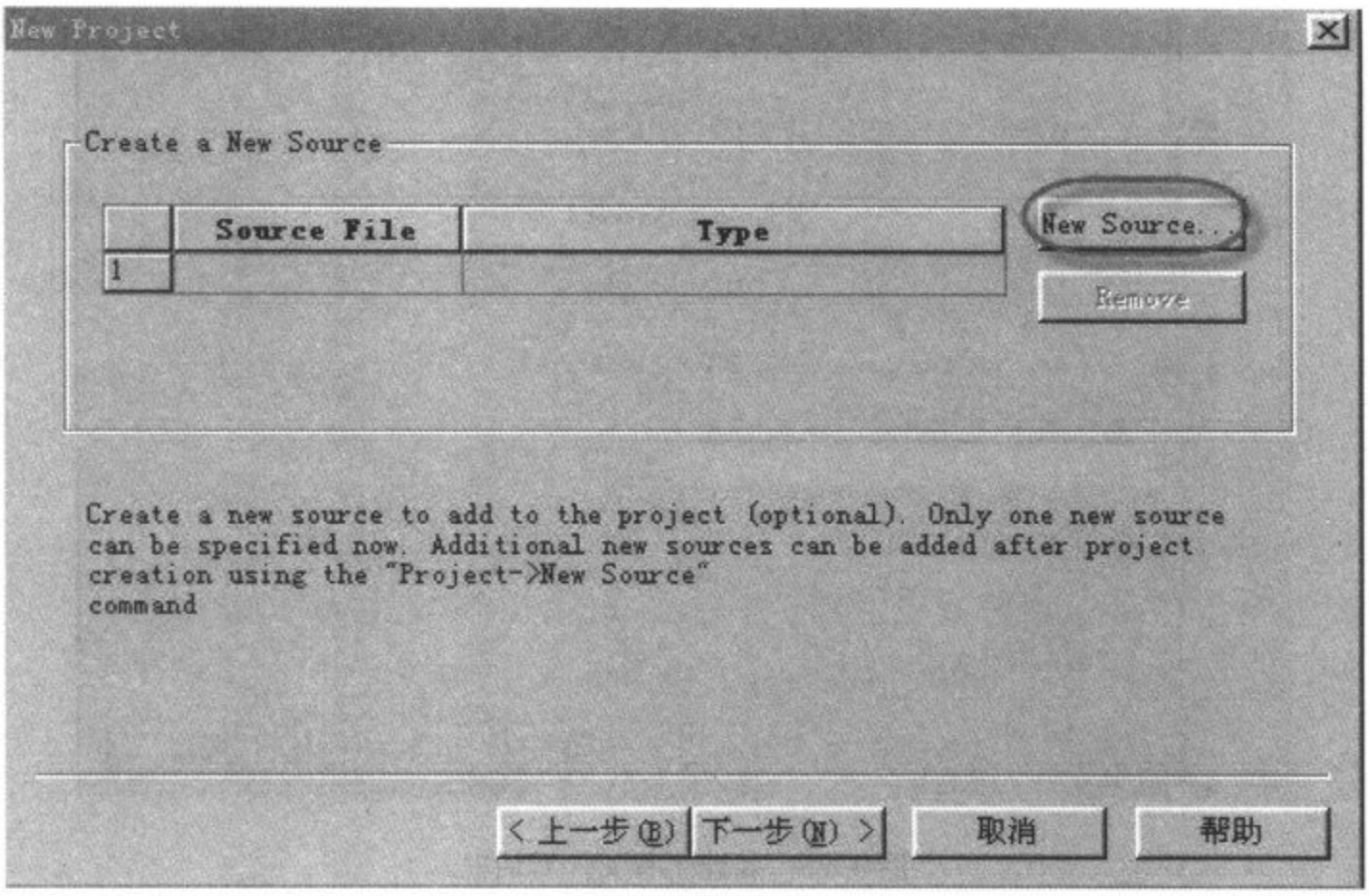


图 3-40　建立源程序对话框

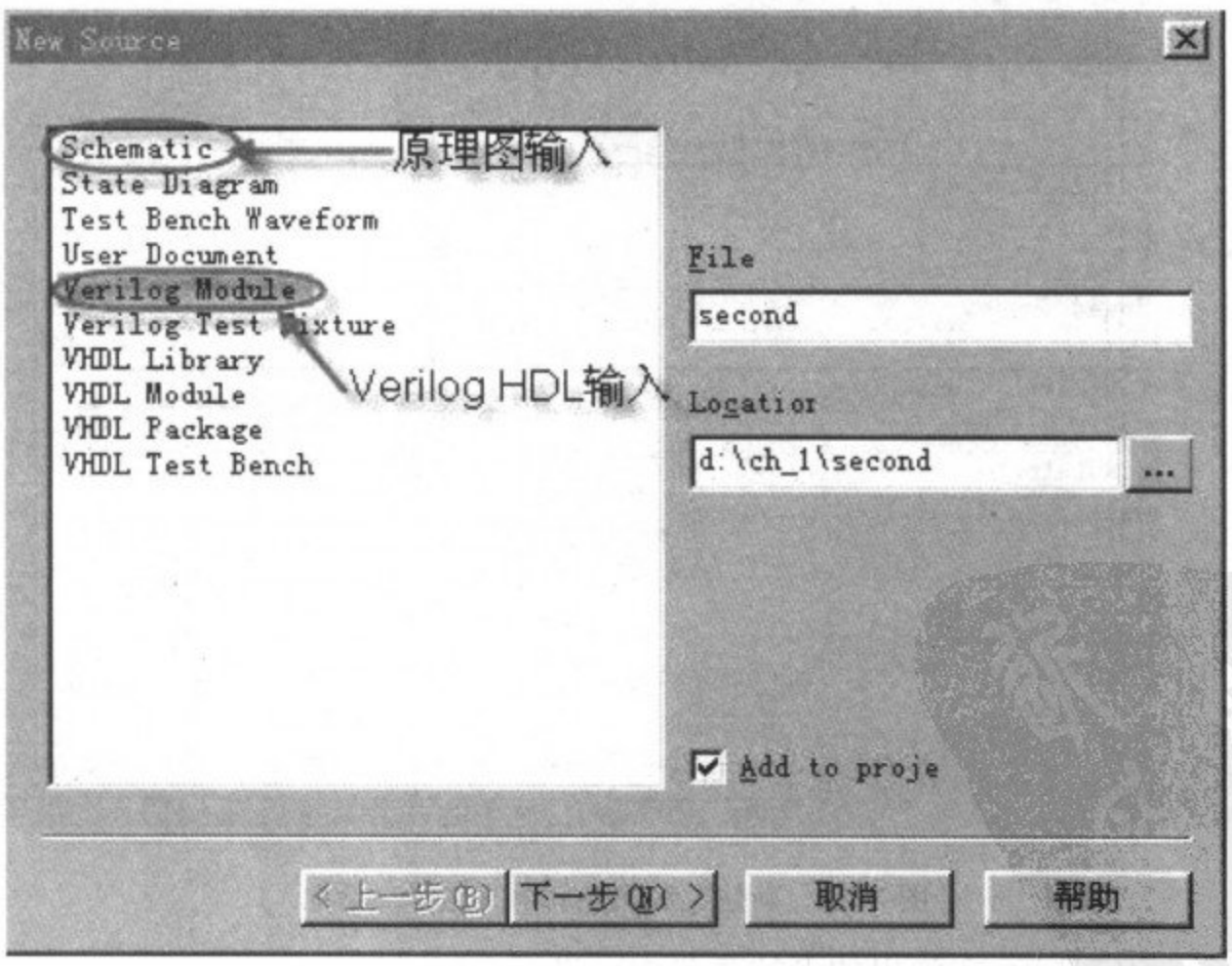


图 3-41　新建源程序对话框

供了一个 Veilog HDL 语言设计方法，而 Schematic 选项则提供了一个原理图的输入设计方法。选择 Verilog Module 选项，这里采用硬件描述语言 Verilog HDL 输入方式。然后在 File 栏内键人该文件的文件名，如 second。对于路径的设置项 Location，可以取其默认值，以保证工程项目和源程序文件在同一个目录中。

(5)单击图 3-41 中的下一步按钮，出现如图 3-42 所示的输入/输出引脚定义对话框，定义所需要的输入/输出引脚。

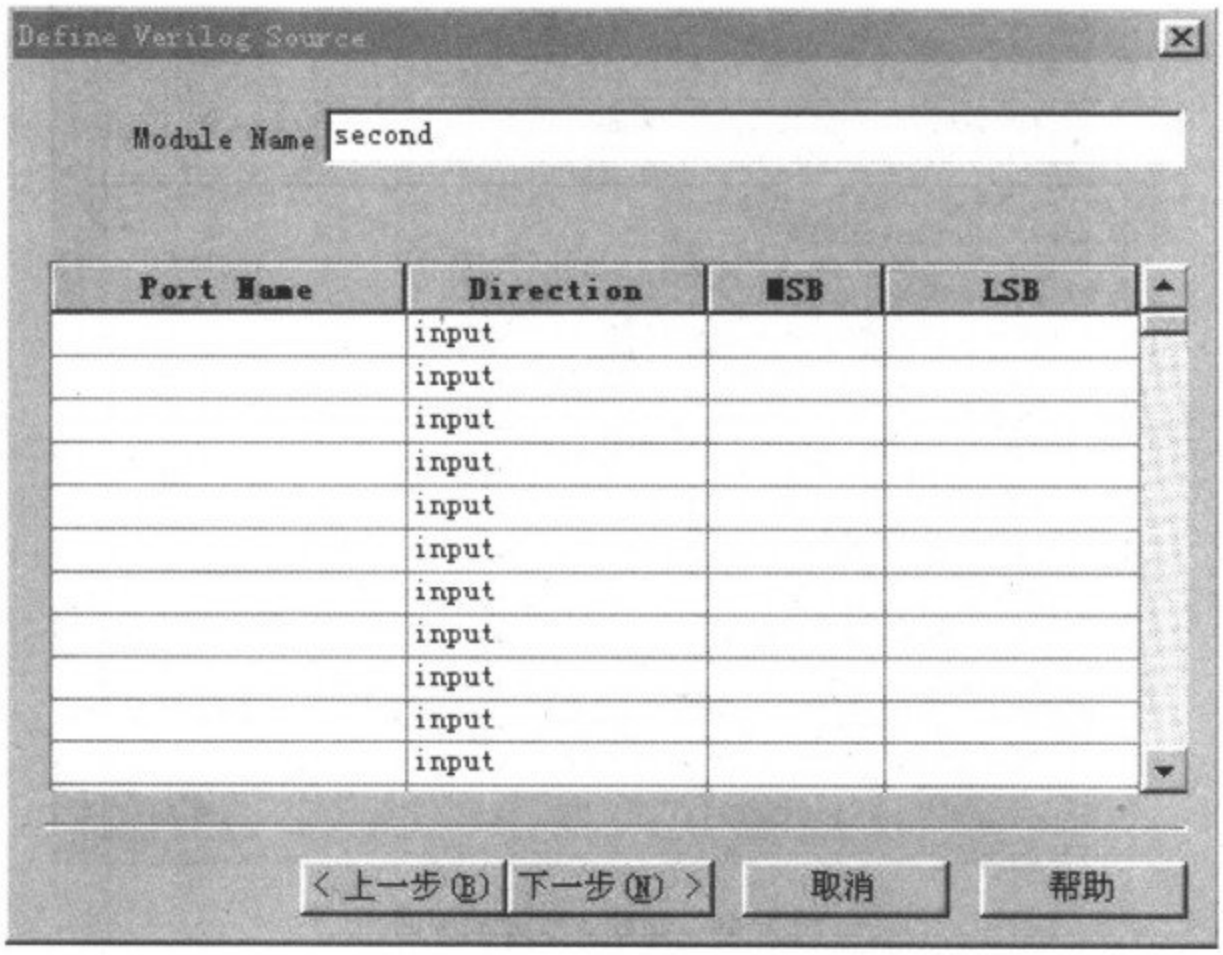


图 3-42　输入/输出引脚定义对话框

(6)在图 3-42 中，首先在 Port Name 栏内键入输入引脚名 K1，设置它的属性为 input；在 MSB 和 LSB 栏内分别输入 0 和 0(这里定义了 1 个输入口，若定义 8 个输入端口，应在 MSB 和 LSB 栏内分别输入 7 和 0，即 [7:0])；用同样的方法输入好 K2、K3、L1 和 L2。输入后的情况如图 3-43 所示。

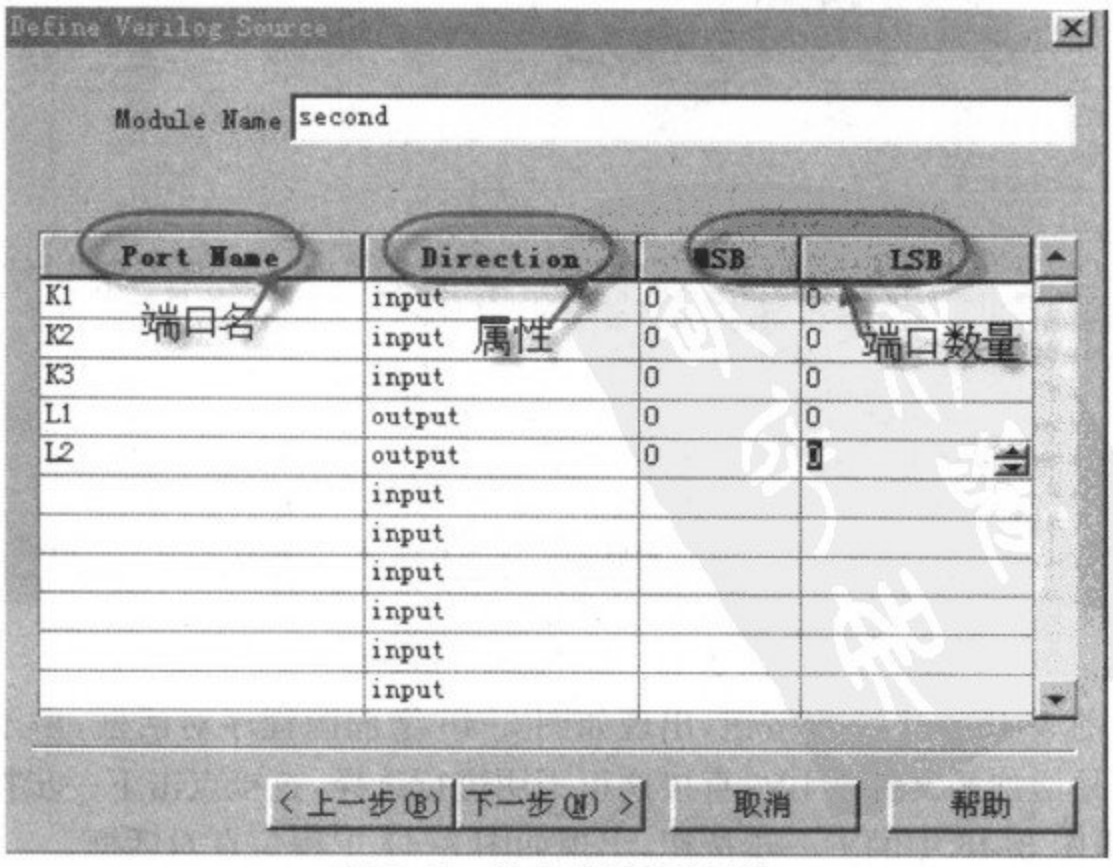


图 3-43　输入引脚名和属性

(7)单击图 3-43 中下一步按钮，出现新建工程文件完成画面，如图 3-44 所示，点击完成按钮，完成操作。

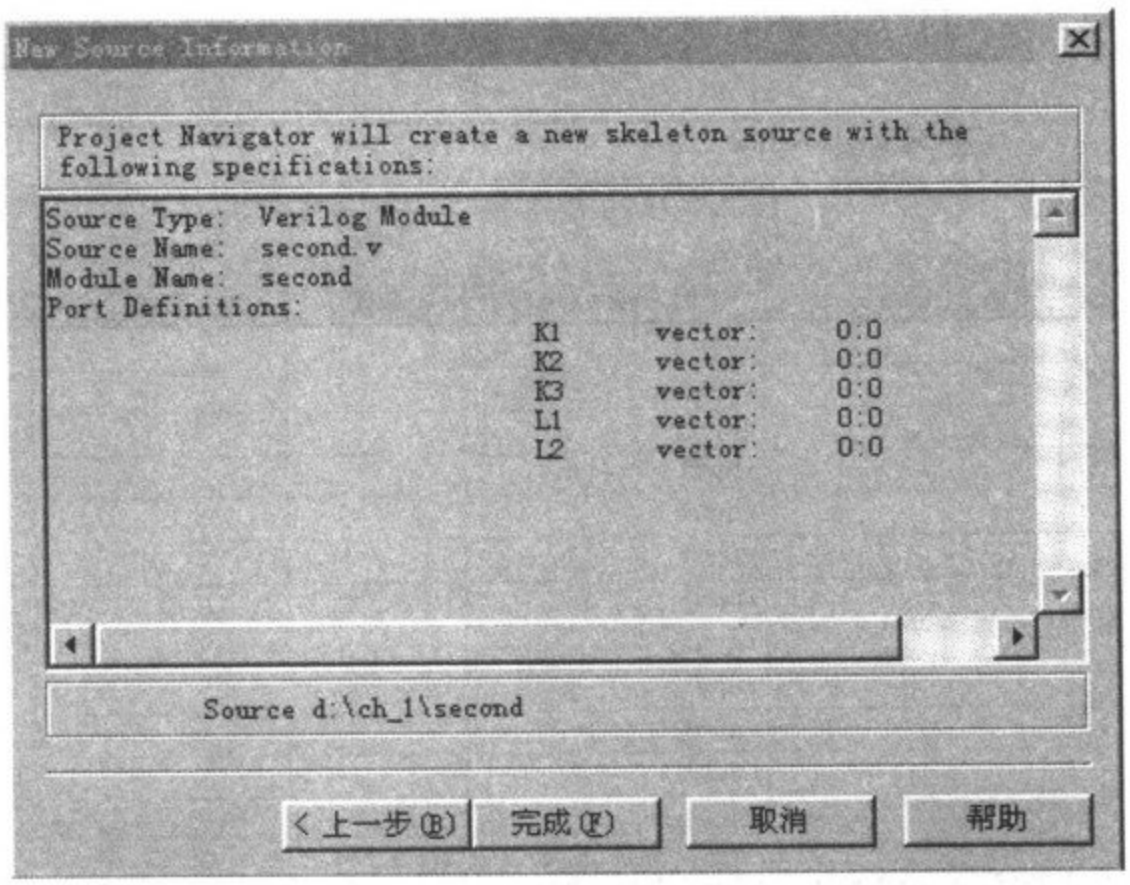


图 3-44　新建工程完成画面

(8)在图 3-44 新建完成画面中单击完成按钮后，出现如图 3-45 所示建立源程序对话框。这时在 Soure File 栏内增加了 second.v 源程序文件。

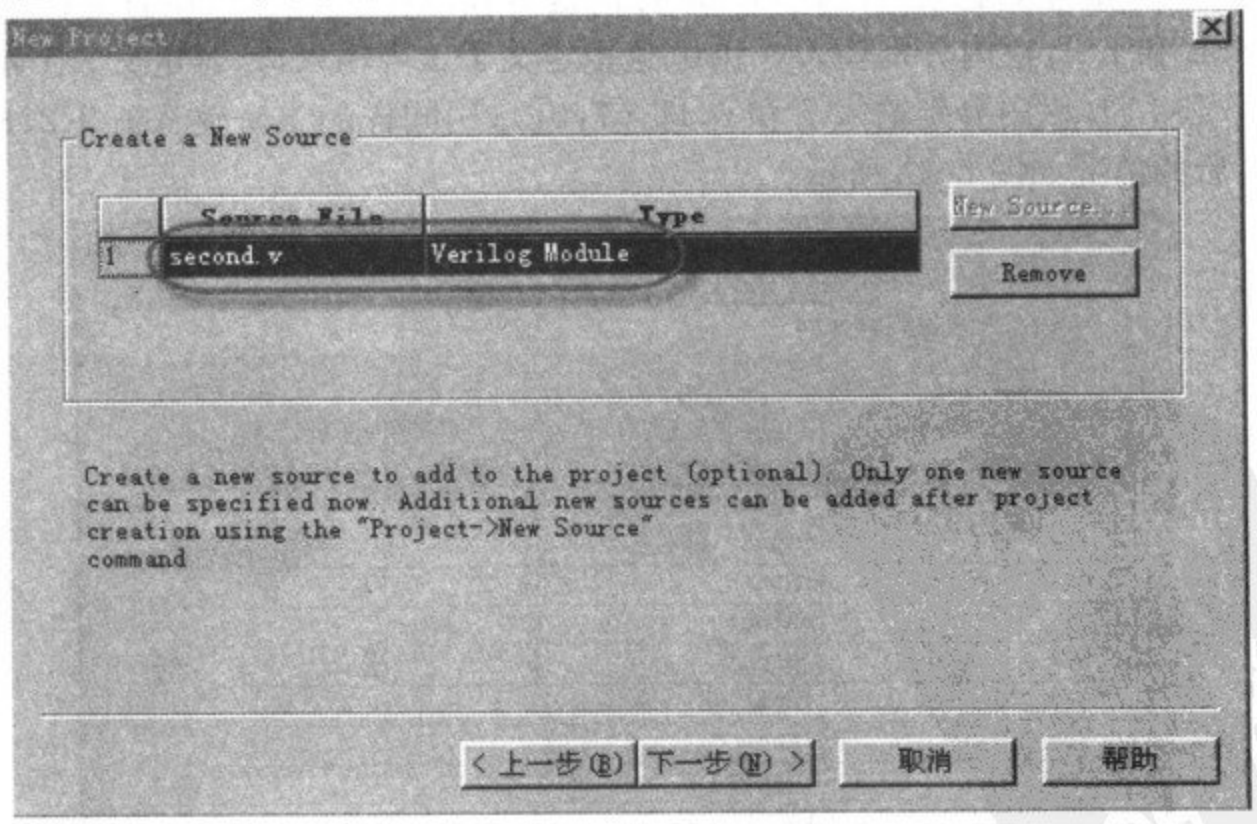


图 3-45　新建源程序对话框

(9)单击图 3-45 中下一步按钮，出现如图 3-46 增加源程序对话框，由于这里是建立源文件，而不是增加源文件，因此，直接这里不用进行选择，直接点击下一步按钮即可。

(10)单击图 3-46 中的下一步按钮，出现如图 3-47 工程信息对话框。

(11)单击完成按钮，返回到 WebPACK 集成开发环境，这时工程项目栏内增加了

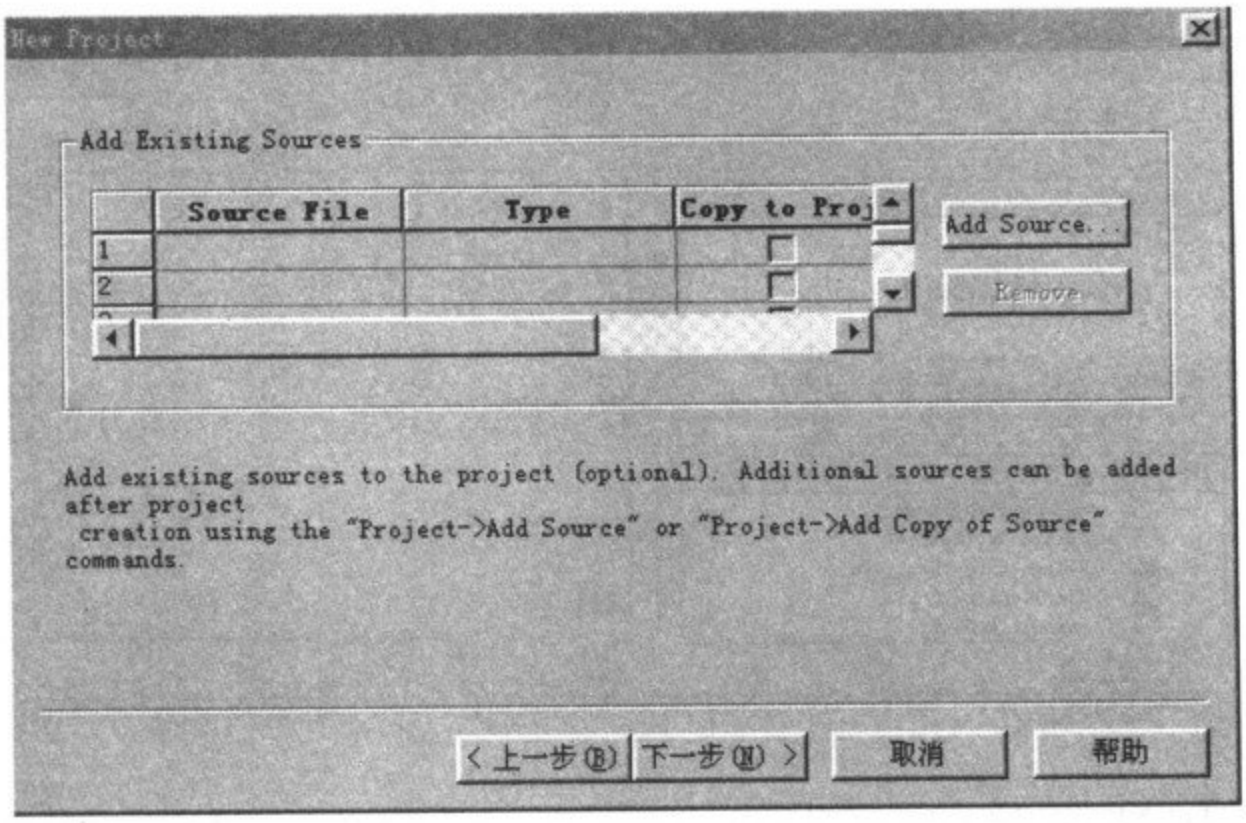


图 3-46　增加源程序对话框

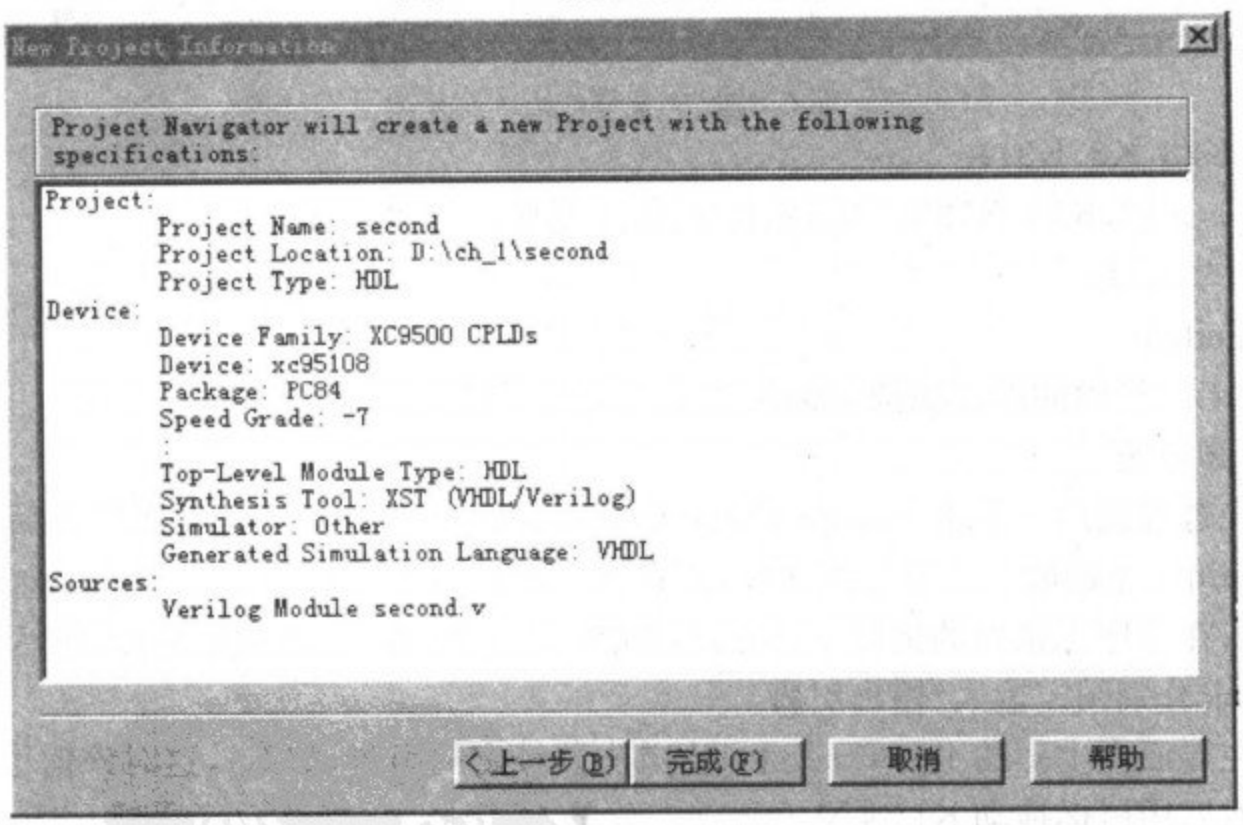


图 3-47　工程信息对话框

second. v 源程序文件，同时在编辑窗口中出现了如图基本框架结构，如图 3-48 所示。

2. 设计输入

在 WebPACK 集成开发环境的编辑窗口中，输入 3 人表决器源 Verilog 源程序，其中 K12，K13，K23 为中间变量：

```
module second(K1,K2,K3,L1,L2);
output L1,L2;
input K1,K2,K3;
and(K12,K1,K2);
and(K13,K1,K3);
```

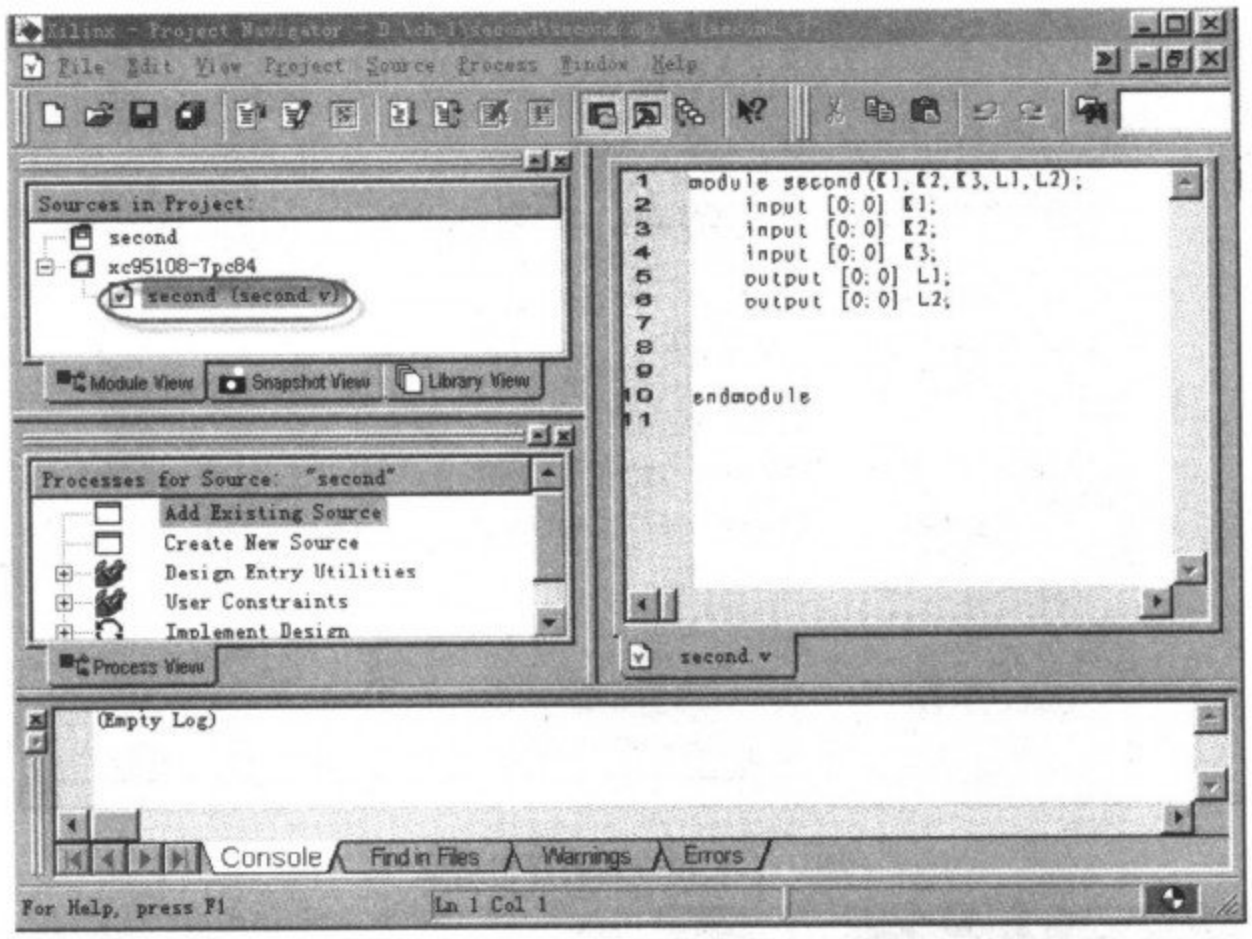


图 3-48　加入工程后的 WebPACK 集成开发环境

and(K23,K2,K3);

or(L1,K12,K13,K23);//K12、K23、K13 是中间变量

not(L2,L1);

endmodule

单击工具栏内的保存快捷图标保存整个工程项目。

3. 引脚锁定

(1)在进程窗口中单击 Design Entry Utilities 左边的“+”符号,层次展开后单击 User Constraints 左边的“+”符号再次展开,在出现的项目中选择 Assign Package Pins 命令并双击它,在程序无错误的情况下,会弹出如图 3-49 所示的引脚锁定界面,然后按照实际的硬件要求分别锁定输入/输出引脚。

(2)在界面左上侧的 I/O Pins 目录内单击要锁定的引脚,如 K1,这时该信号 K1 变成深蓝色;然后用鼠标拖动 K1 到 XC95108 芯片的 54 引脚上(当鼠标移到脚上时,会自动显示脚号)并单击该引脚,这时该引脚变成蓝色,即表示 K1 已经被锁定成功。依此方法将 K2 锁定在 55 脚上,将 K3 锁定在 56 脚上,将 L1 锁定在 31 脚上,将 L2 锁定在 32 脚上,如图 3-50 所示。

锁定完毕后保存、退出,即可重新返回到 WebPACK 集成开发环境。

(3)在进程窗口中双击 Implanment Design 命令,如图 3-51 所示,运行相关进程。其中,打对勾标记的表示该进程已经成功运行;而打感叹号标记的则表示该进程已经运行,但是包含系统给出的警告。

(4)成功运行进程的画面如图 3-52 所示,并生成了 second.jed 程序文件。

4. 器件的下载编程

取出 JTAG 电缆,并将 JTAG 下载电缆的两端分别连接到 PC 机和 DP-MCU/Xilinx

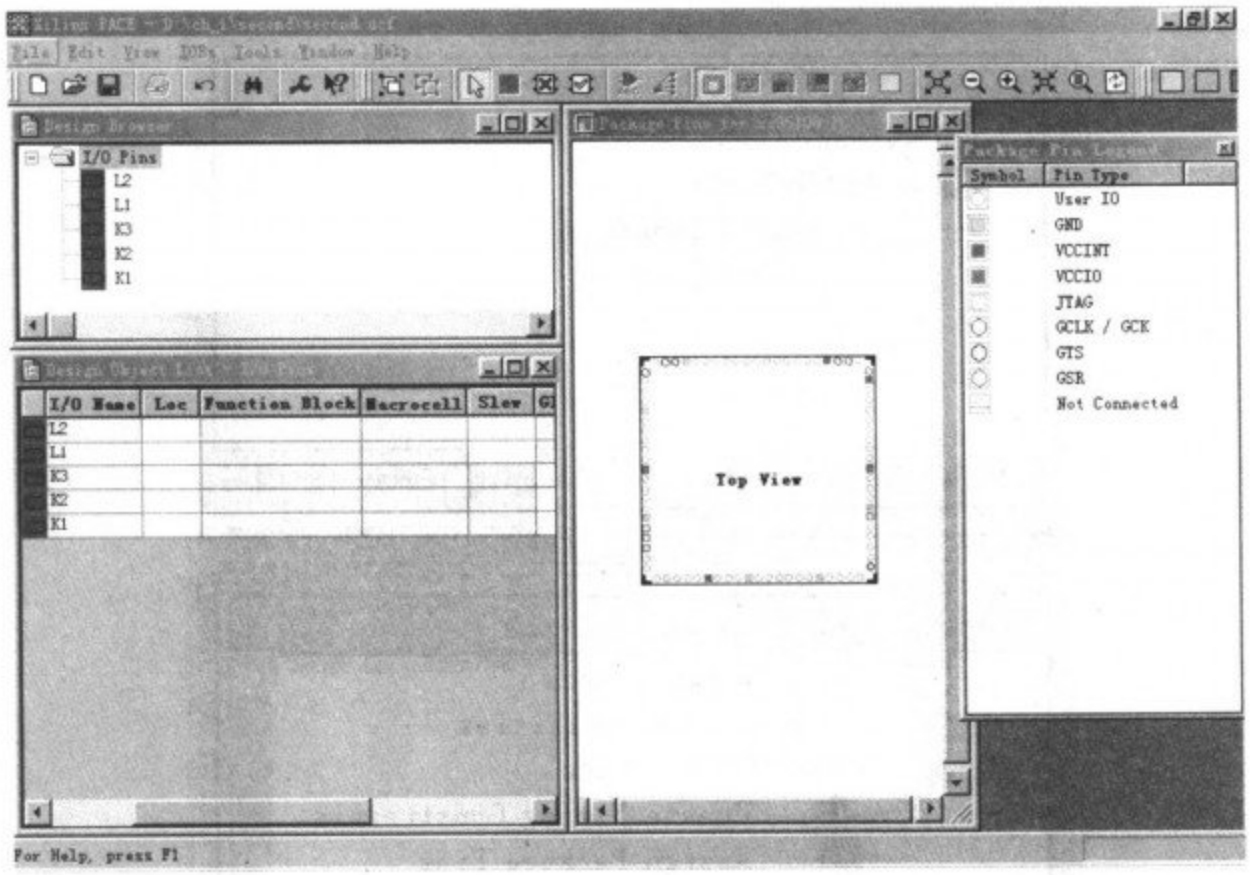


图 3-49　锁定引脚画面

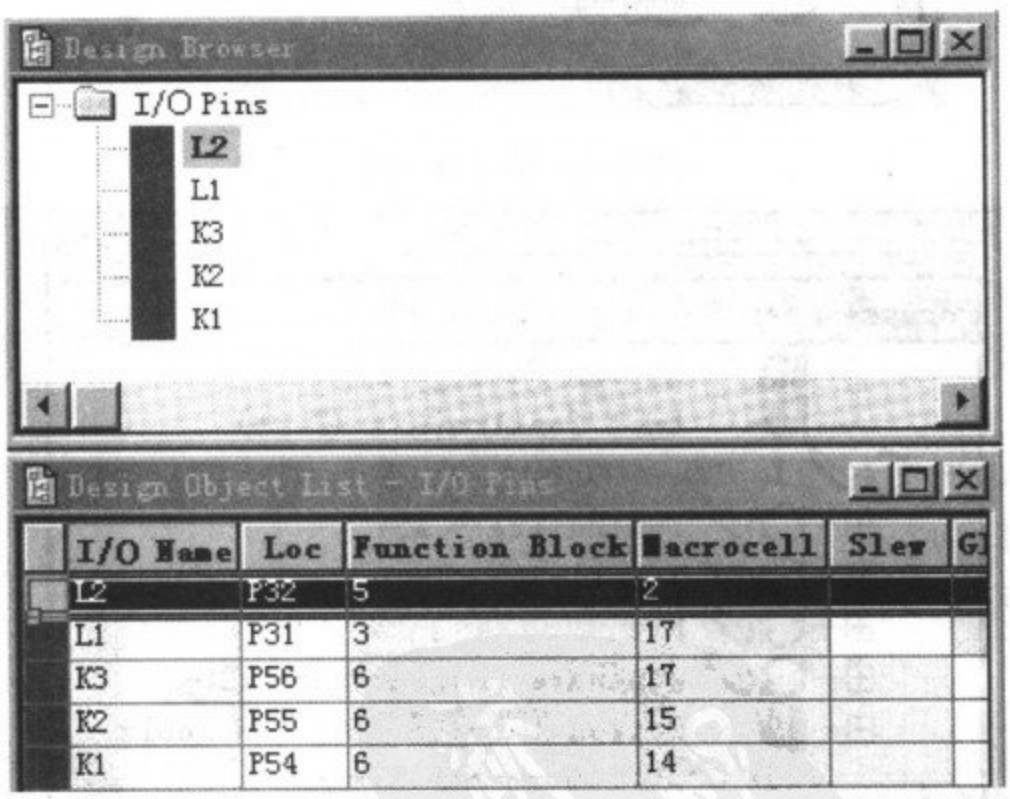


图 3-50　锁定引脚后的情况

实验仪的 JTAG 口 J4 上，打开工作电源；稍后会弹出下载程序窗口，单击芯片 XC95108，此时芯片呈深蓝色的突出标记；然后右击鼠标，从快捷菜单中的选择 Program 下载命令。在出现的下载选项窗口中，根据需要选择相应的选项。其中 Erase Before Programme 选项表示在下载编程前先擦除芯片，Verify 选项表示下载编程完毕后进行下载校验，Read Protect 选项表示读加密保护位，而 Write Protect 选项表示对芯片加密保护。选中第一项和第二项(注意：任何已被加密写过的芯片必须先擦除后下载)，单击 OK 按钮开始下载。下载完毕后，弹出“Programme Success!”表示下载成功。至此，就已经完成一个简单的 CPLD 开发。

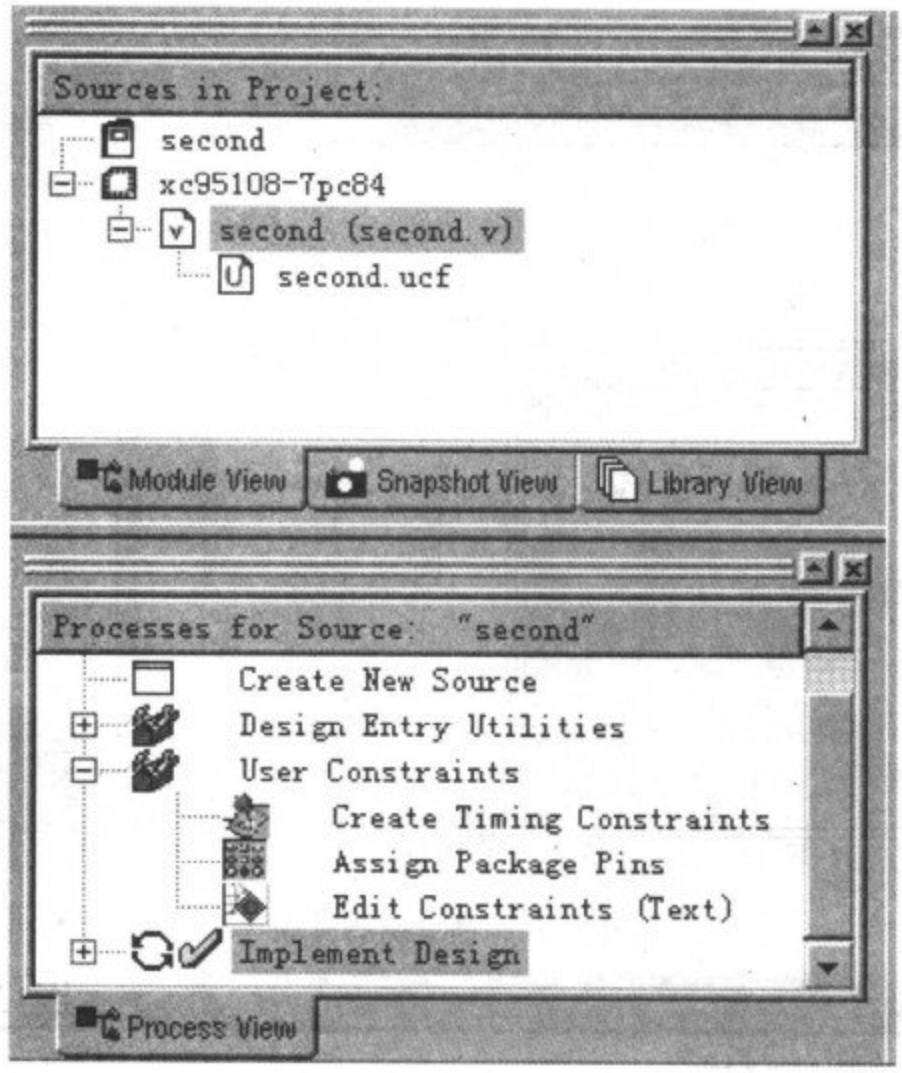


图 3-51　运行相关进程

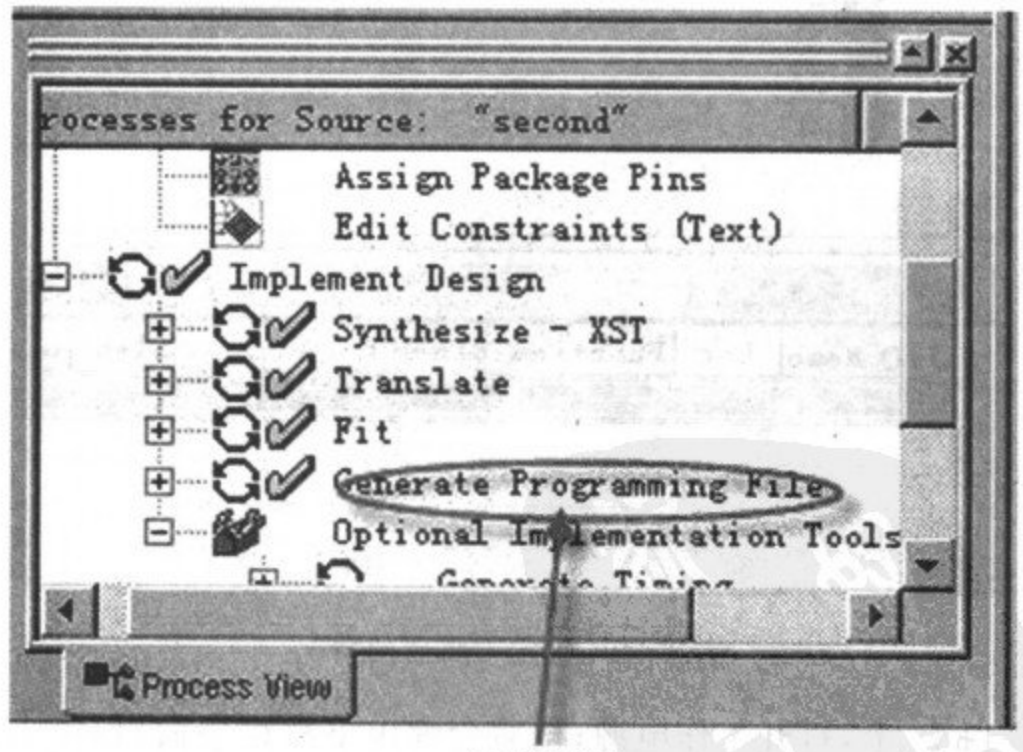


图 3-52　成功运行进程画面

第三节　仿真Modelsim SE 软件的安装和使用

Modelsim SE 是 ModelTechnology 公司的产品，属于编译型 VHDL/Verilog 仿真器。这是一个针对可编程逻辑设计进行优化的完整 HDL 仿真环境。Modelsim SE 版本较多，下面以 Modelsim SE　6.0 版为例进行介绍。

一、Modelsim SE 6.0 软件的安装

(1)双击 Modelsim SE 6.0 安装图标,出现如图 3-53 所示的软件版本选择画面,这里选择 Full Product,点击 Next。

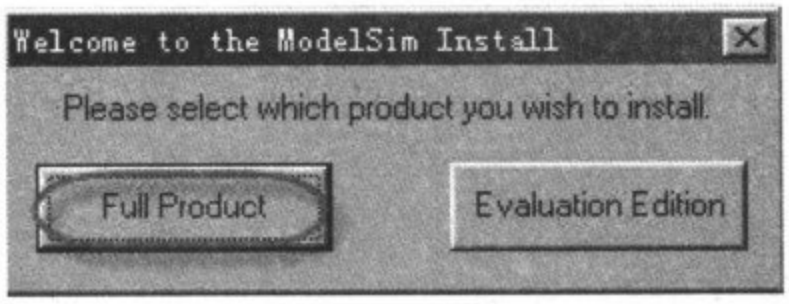


图 3-53　软件版本选择画面

(2)接着出现欢迎画面,如图 3-54 所示,然后单击 Next。

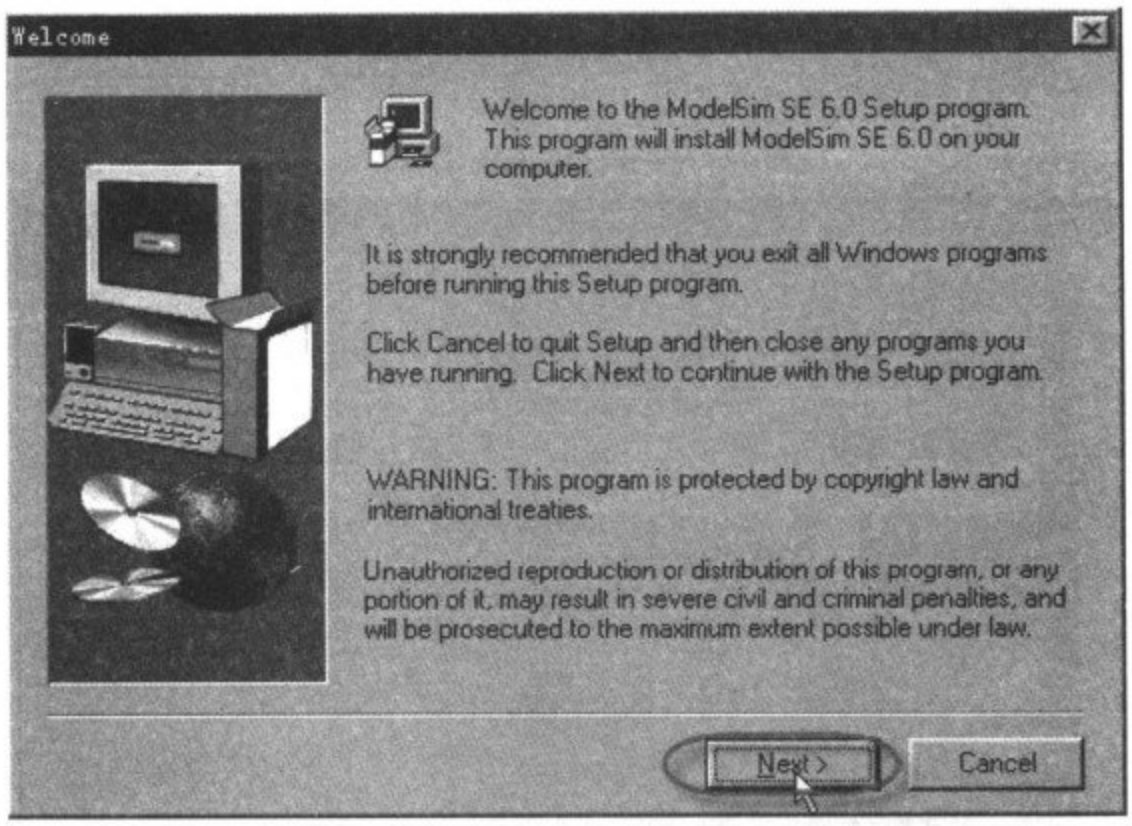


图 3-54　欢迎画面

(3)在出现的软件许可协议对话框中点击 Yes,如图 3-55 所示。

(4)出现的软件安装路径对话框,默认路径是 C:\Modeltech_6.0。当然,也可以单击 Brower...按钮来选择适合自己的安装目录,这里选择默认路径,然后单击 Next,如图 3-56 所示。

(5)出现的程序文件夹对话框,单击 Next,如图 3-57 所示。

(6)随后出现安装进度指示界面,程序开始安装,安装完成后,单击 Finish 按钮,程序安装完成,如图 3-58 所示。

(7)点击“程序→ModelSim SE 6.0→Licensing Wizard”,如图 3-59 所示。

(8)出现授权向导对话框,点击 continue,如图 3-60 所示。

(9)选择许可证存放位置,点击“OK”,如图 3-61 所示,至此,程序安装全部完成。

需要说明的是,许可证需要到软件公司网站上进行申请,申请成功后,公司会通过电子邮件寄到你的信箱中。

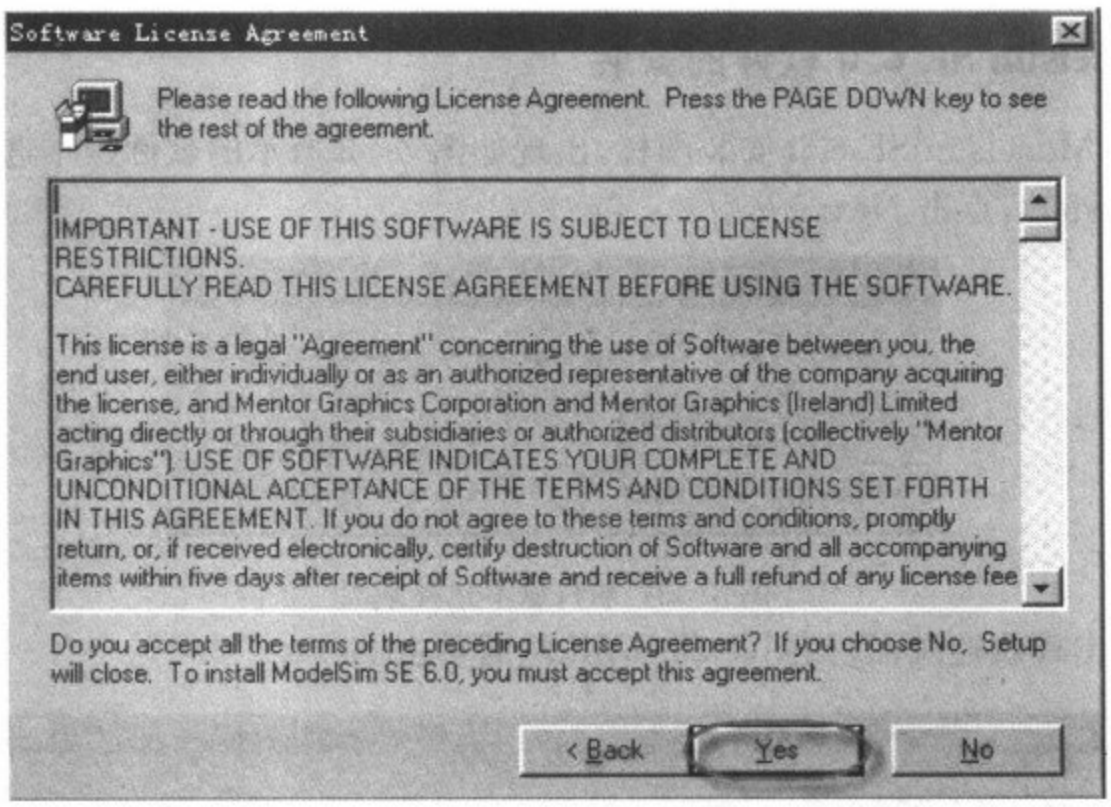


图 3-55　软件许可协议对话框

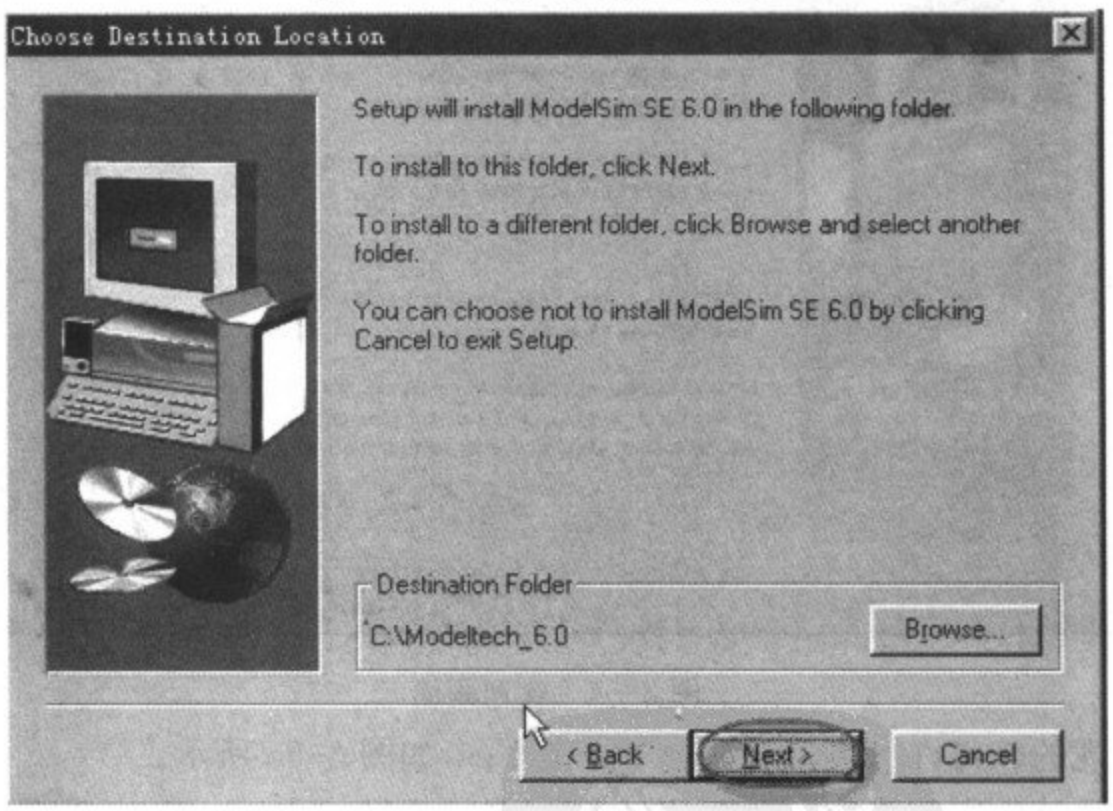


图 3-56　选择安装路径画面

二、Modelsim SE 6.0 软件的使用

下面以二与门为例进行说明，用 Verilog 语言编写的程序如下(文件名为 and2)：

```
module and_2 (A, B, F);
input      A, B;
output F;
and U (F, A, B);
endmodule
```

用 Modelsim XE II 5.8c 进行仿真时，需要编写验证测试程序，用 Verilog 语言编写的测试程序如下(文件名为 and2_test)：

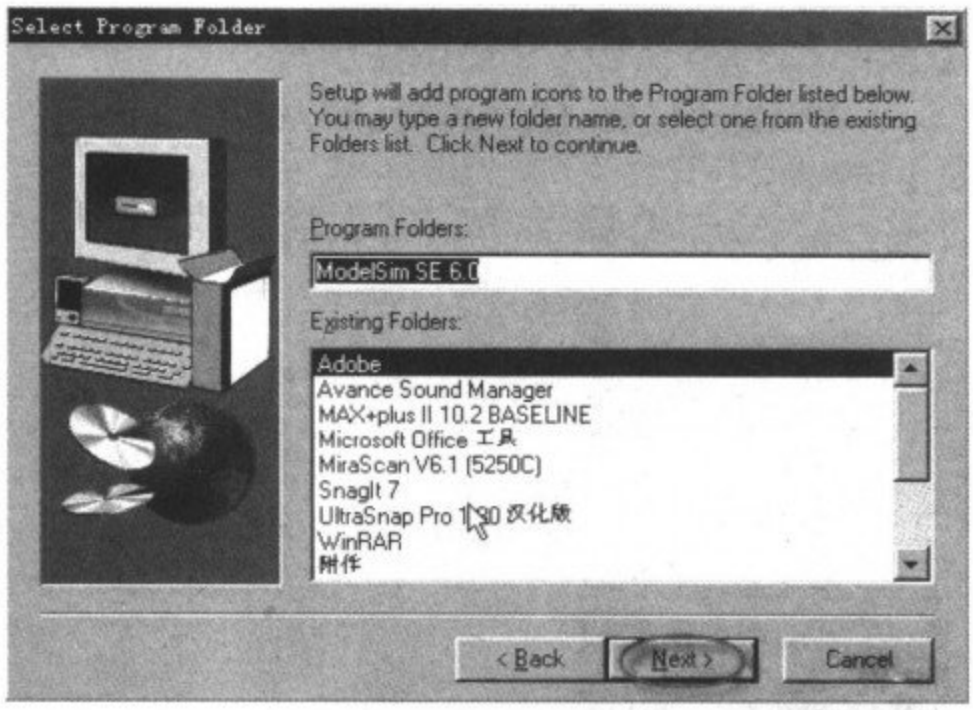


图 3-57　程序文件夹对话框

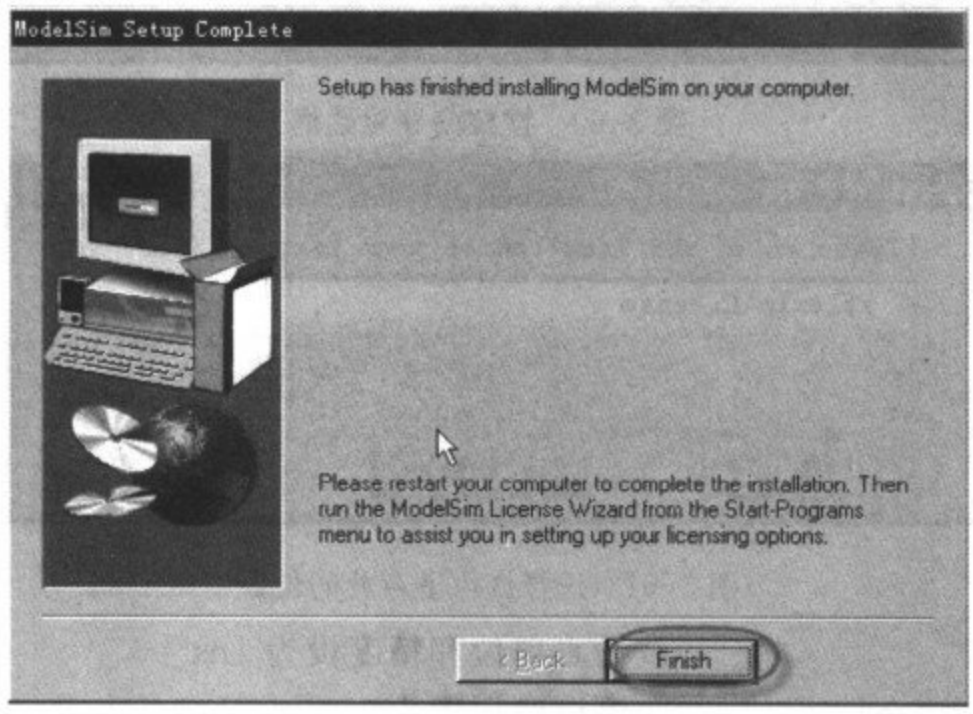


图 3-58　安装完成画面

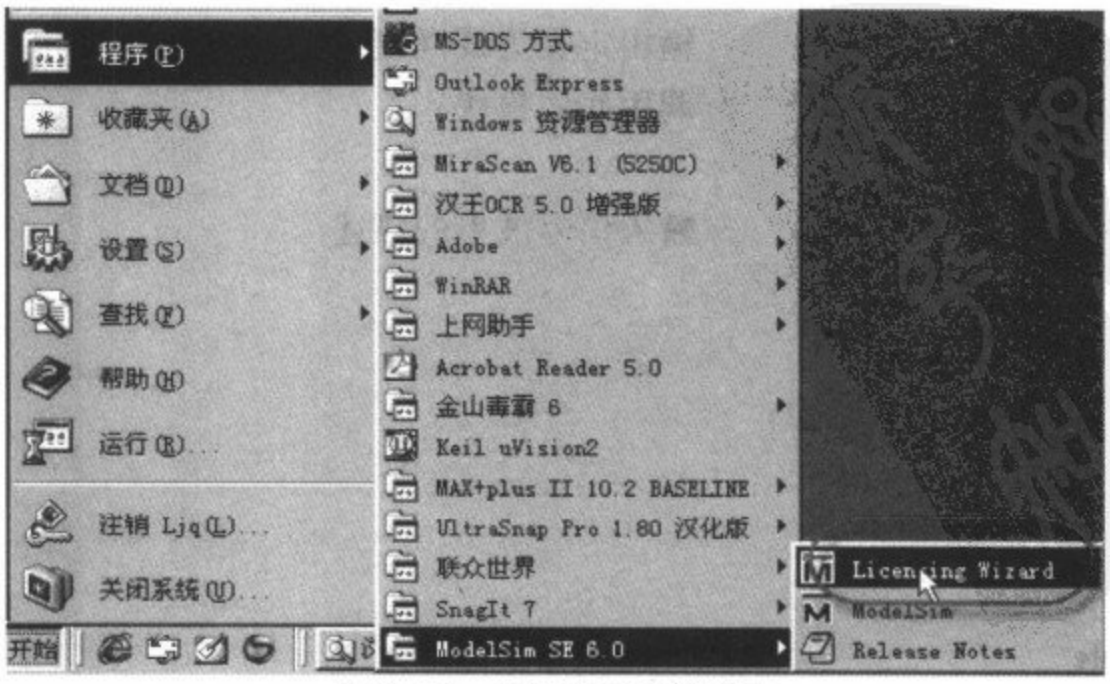


图 3-59　打开授权申请向导

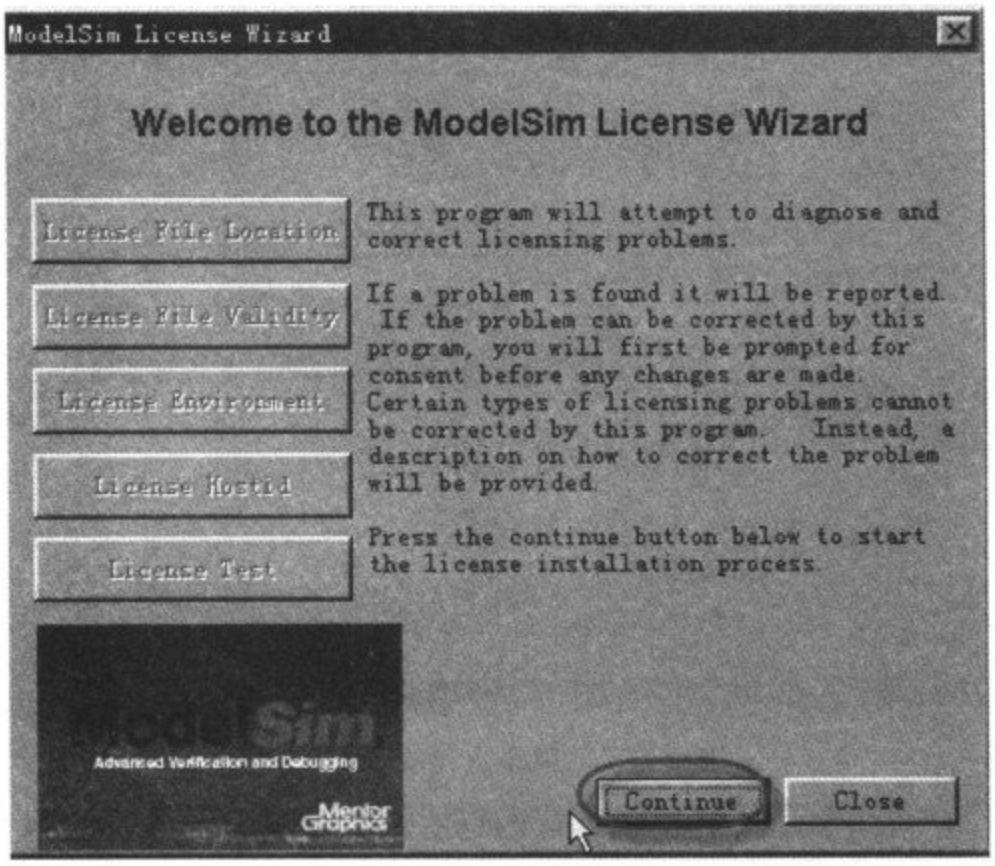


图 3-60　授权向导对话框

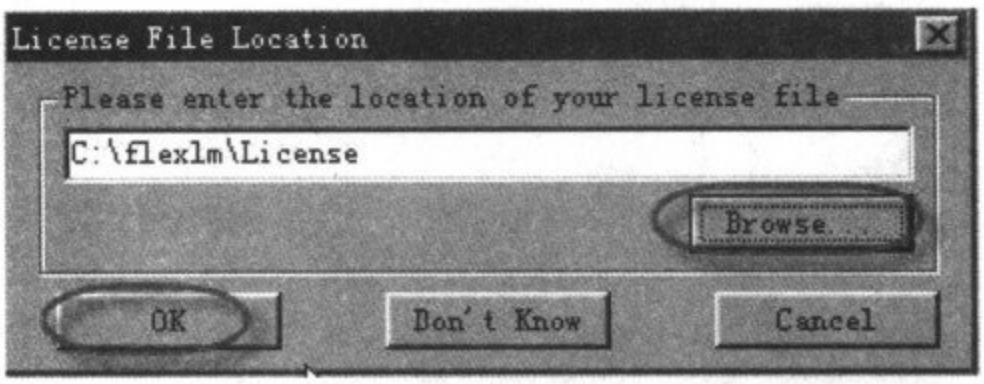


图 3-61　选择许可证存放的位置

```
timescale                    //仿真单位和精度设为 1ns
module and_2_test;           //测试底层模块
reg     A, B;                //输入定义为寄存器型
wire    F;                   //输出定义为线网型
and_2 and_2 (A, B, F);       //调用底层模块
initial
begin                        //输入信号波形的描述
    A=0; B=0;
    #100    A=1;
    #100    A=0; B=1;
    #100    A=1;
    #200    $finish;
end
endmodule
```

将以下两个文件分别保存为 and2. v 和 and2_test. v 文件。

1. Modelsim SE 的启动

(1)双击 ModelSim SE 6.0 图标，进入 Modelsim SE 启动界面，如图 3-62 所示。

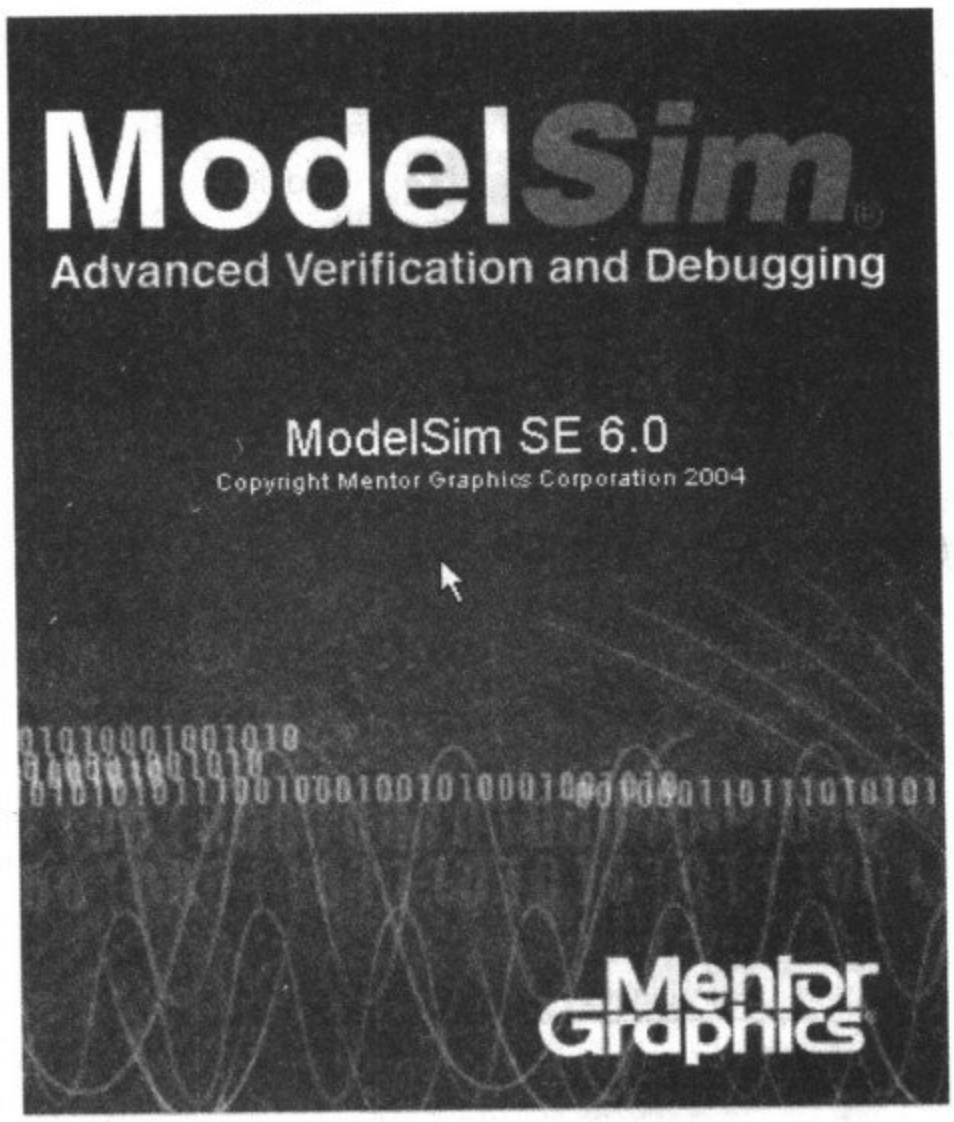


图 3-62　Modelsim SE 启动界面

(2)之后进入 Modelsim SE 主窗口，如图 3-63 所示。

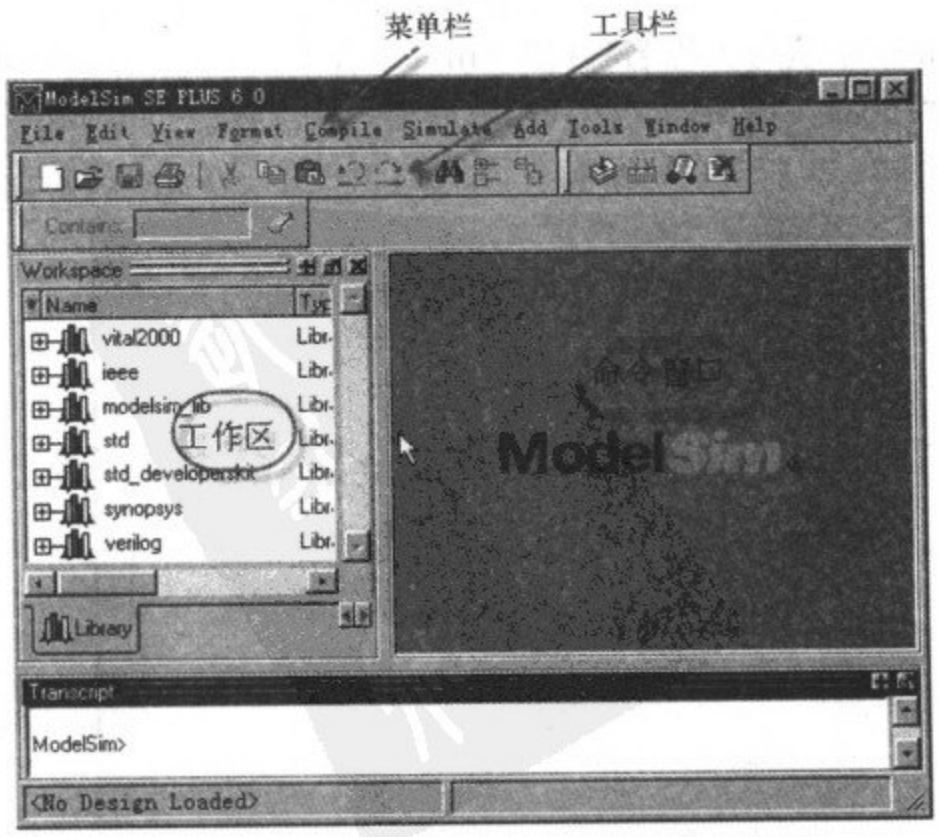


图 3-63　Modelsim SE 主窗口

2. 建立仿真工程项目

(1)在主窗口中,单击菜单栏 File→New→Project...,如图 3-64 所示。

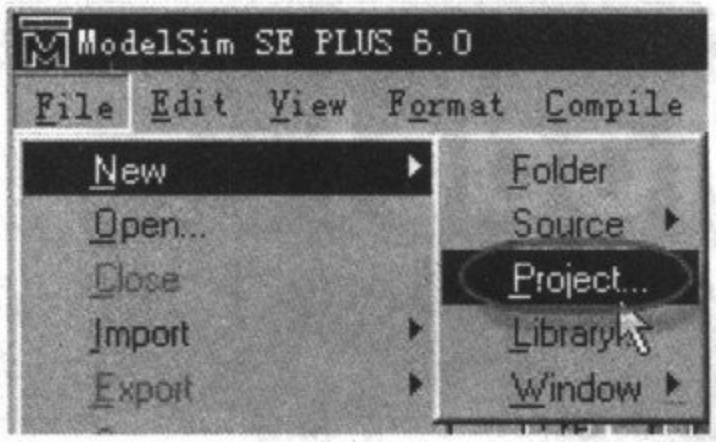


图 3-64　新建工程

(2)之后出现如图 3-65 所示新建工程对话框。

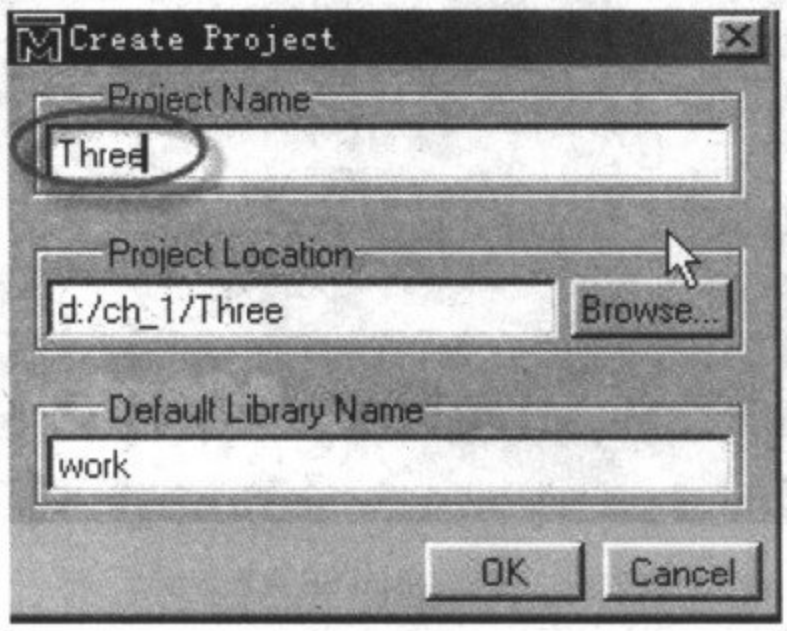


图 3-65　新建工程对话框

在 Project Name 中填写工程名称(如 Three),在 Project Location 填入新建工程文件的位置,可以通过"Browse"选择工程位置,填好后点击 OK。

(3)在出现如图 3-66 所示的添加操作项目对话框中,单击 Add Existing File,如图

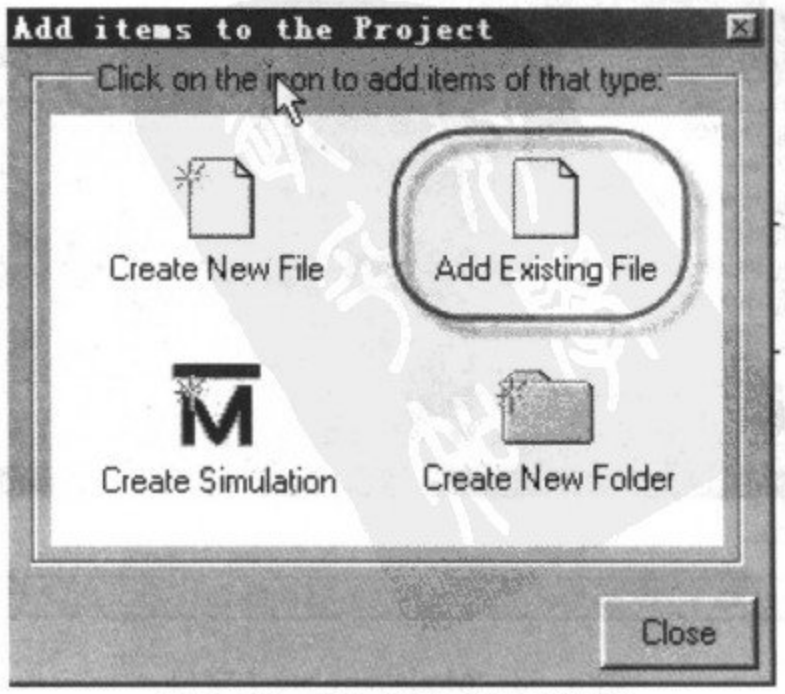


图 3-66　添加操作项目对话框

3-66 所示。

(4)随后出现的添加文件对话框,如图 3-67 所示。

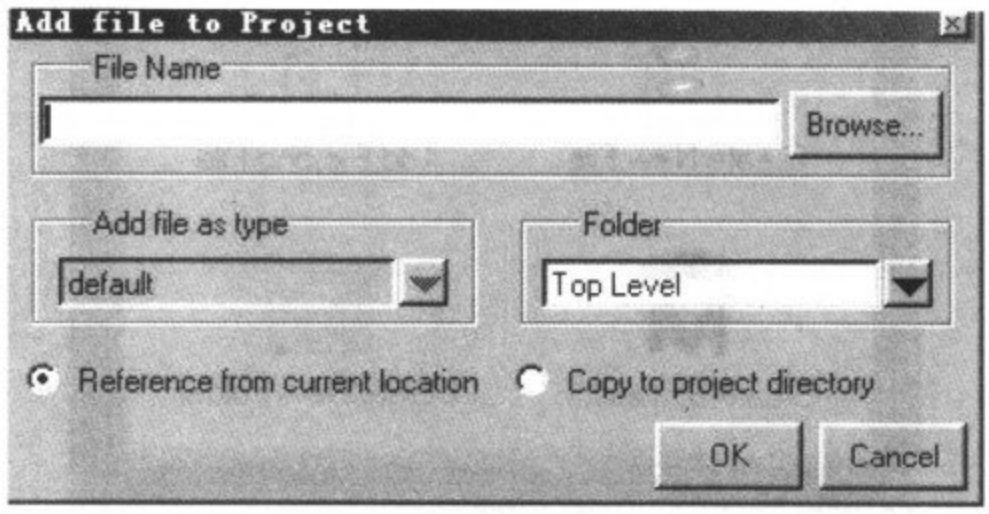


图 3-67　添加文件对话框

(5)单击“Browse...”选择需要添加的文件 and2. v 和 and2_test. v,点击“打开”按钮,如图 3-68 所示。

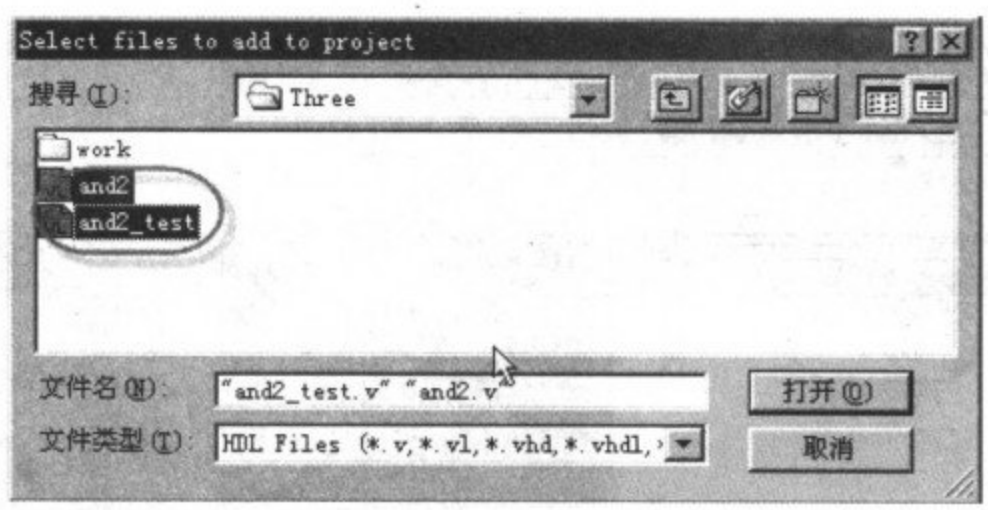


图 3-68　添加文件

(6)添加文件后的对话框如图 3-69 所示,点击 OK 按钮。

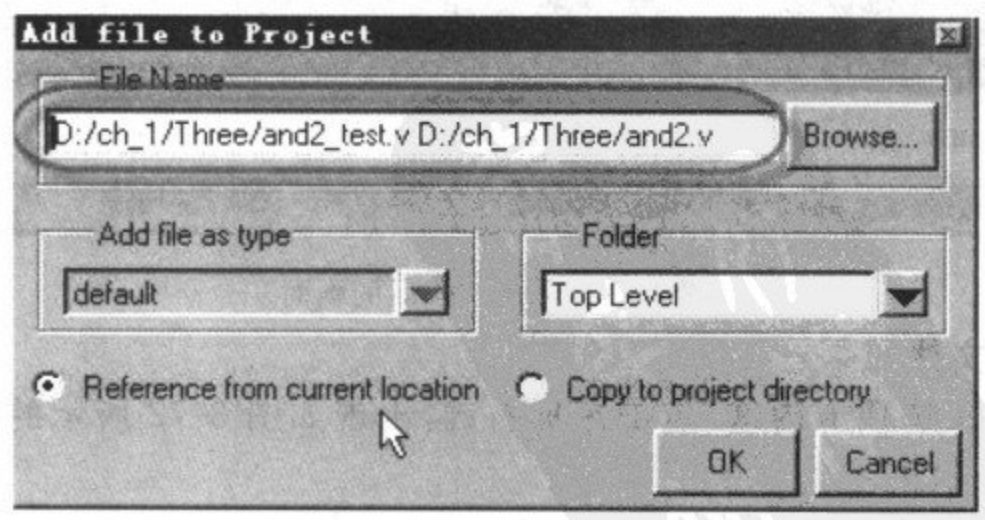


图 3-69　添加文件后的对话框

(7)点击“Close”,关闭添加操作项目对话框,如图 3-70 所示。

(8)添加文件后的画面如图 3-71 所示,从图中可以看出,在 Project 选项卡中出现源文件和测试文件,即建立了一个新工程项目。

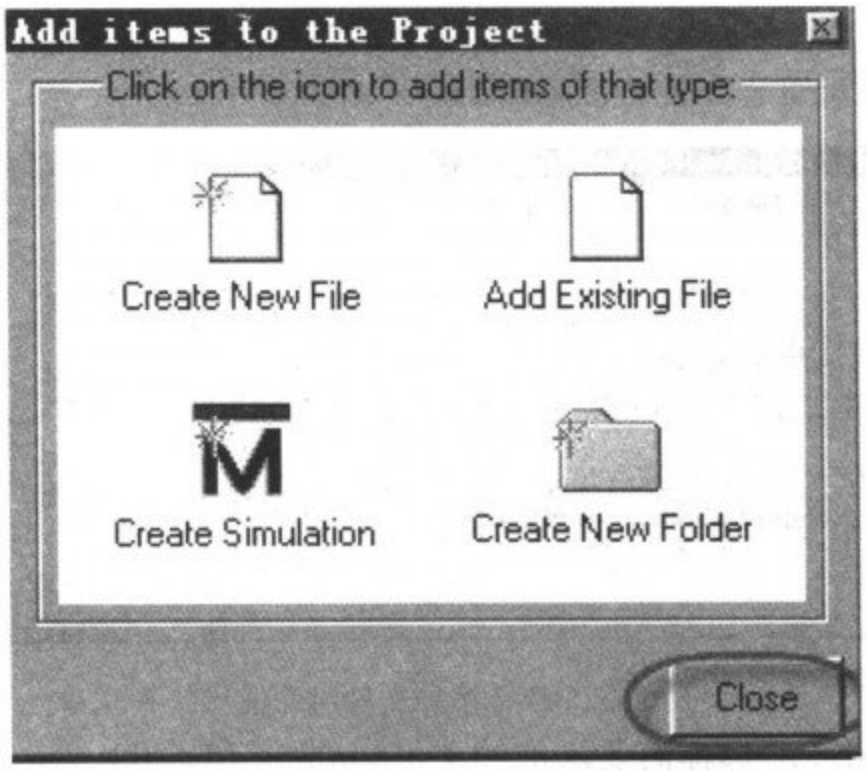


图 3-70　关闭添加操作项目对话框

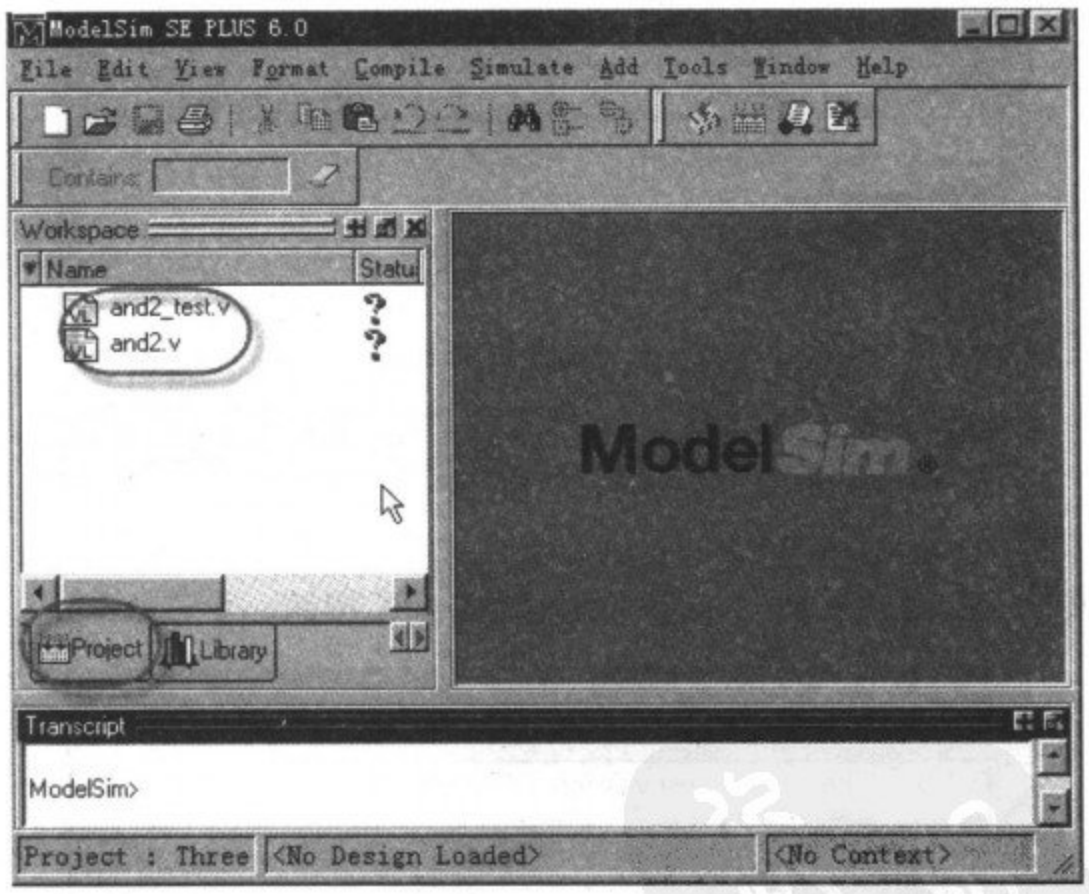


图 3-71　添加文件后的画面

3. 编译仿真文件

(1)在 Project 选项卡内部，点击鼠标右键，出现如图 3-72 所示的快捷菜单，选择 Compile/Compile All。

(2)编译后的画面如图 3-73 所示，从图中可以看出，在 Status 栏中出现对号，表示编译成功。

4. 装载仿真库

(1)打开 Library 选项卡，点击 work 旁边的加号，如图 3-74 所示。

(2)双击 and2_test，加载测试文件，如图 3-75 所示。从图中可以看出，图中出现了

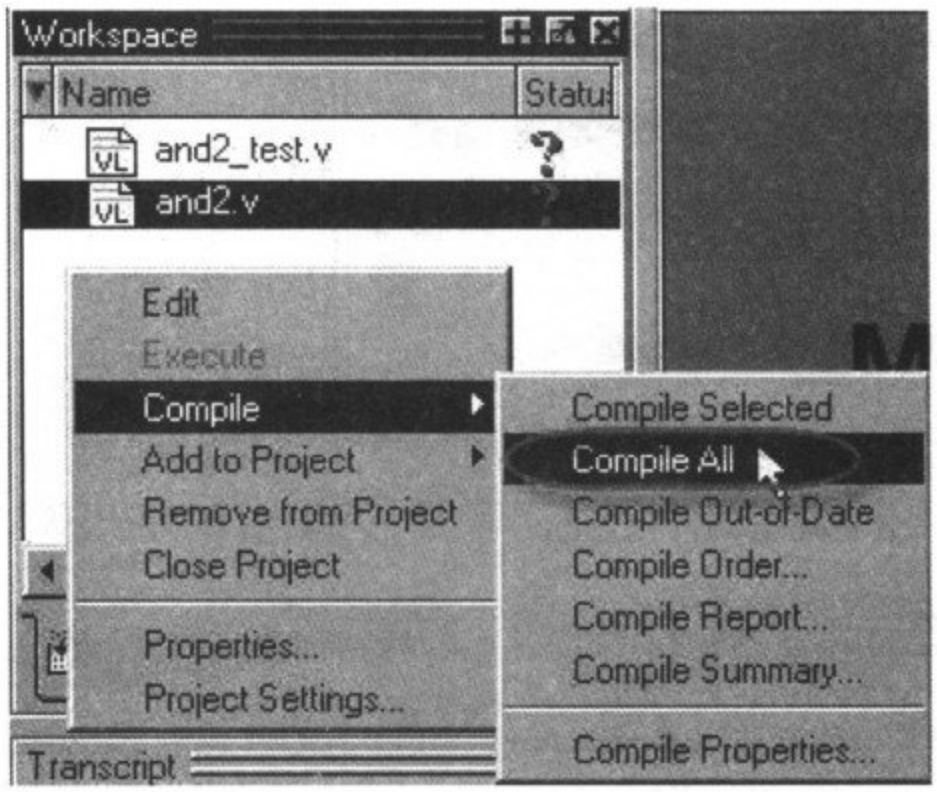


图 3-72　编译快捷菜单

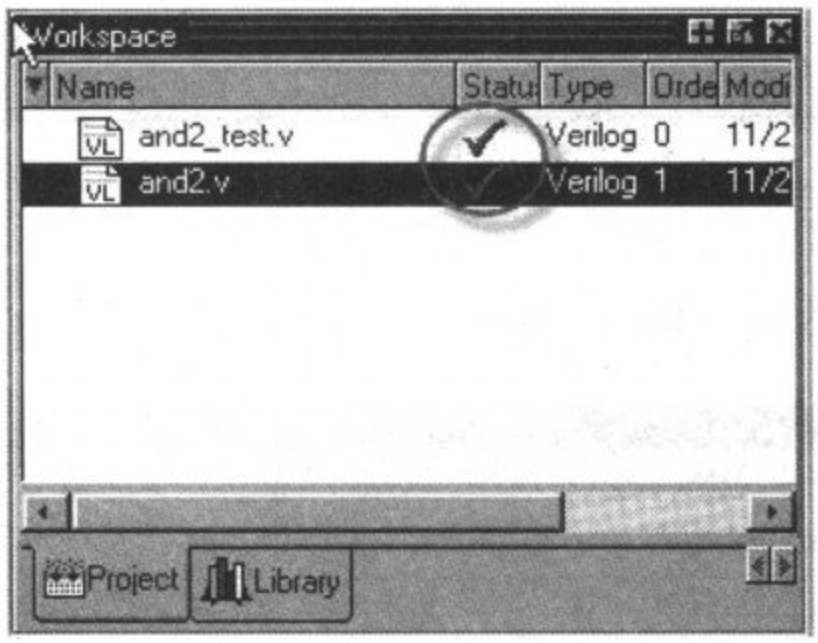


图 3-73　编译成功画面

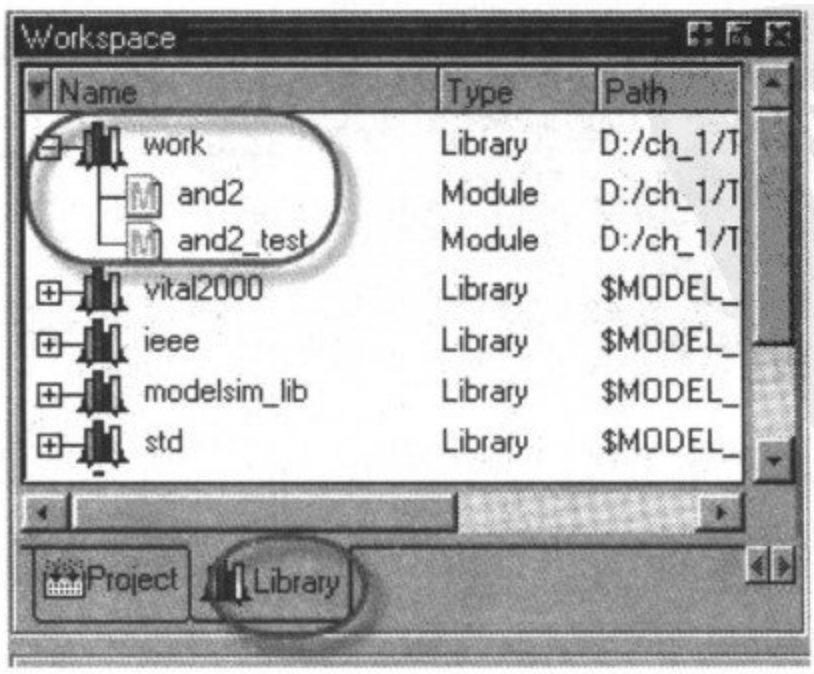


图 3-74　打开 Library 选项卡

sim 选项卡和 Object 窗口。

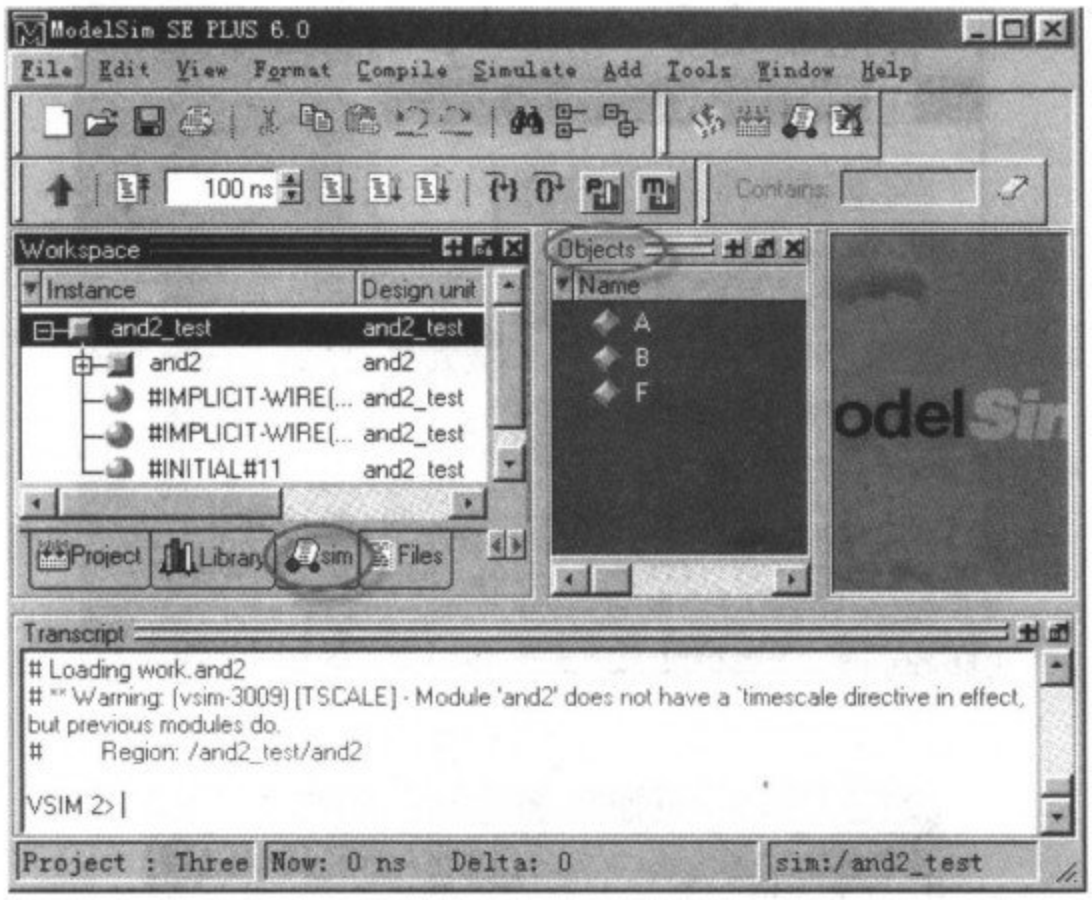


图 3-75　双击 and2_test

5. 执行仿真

(1)单击菜单栏 View→Debug Windows→Wave，打开波形窗口，如图 3-76 所示。

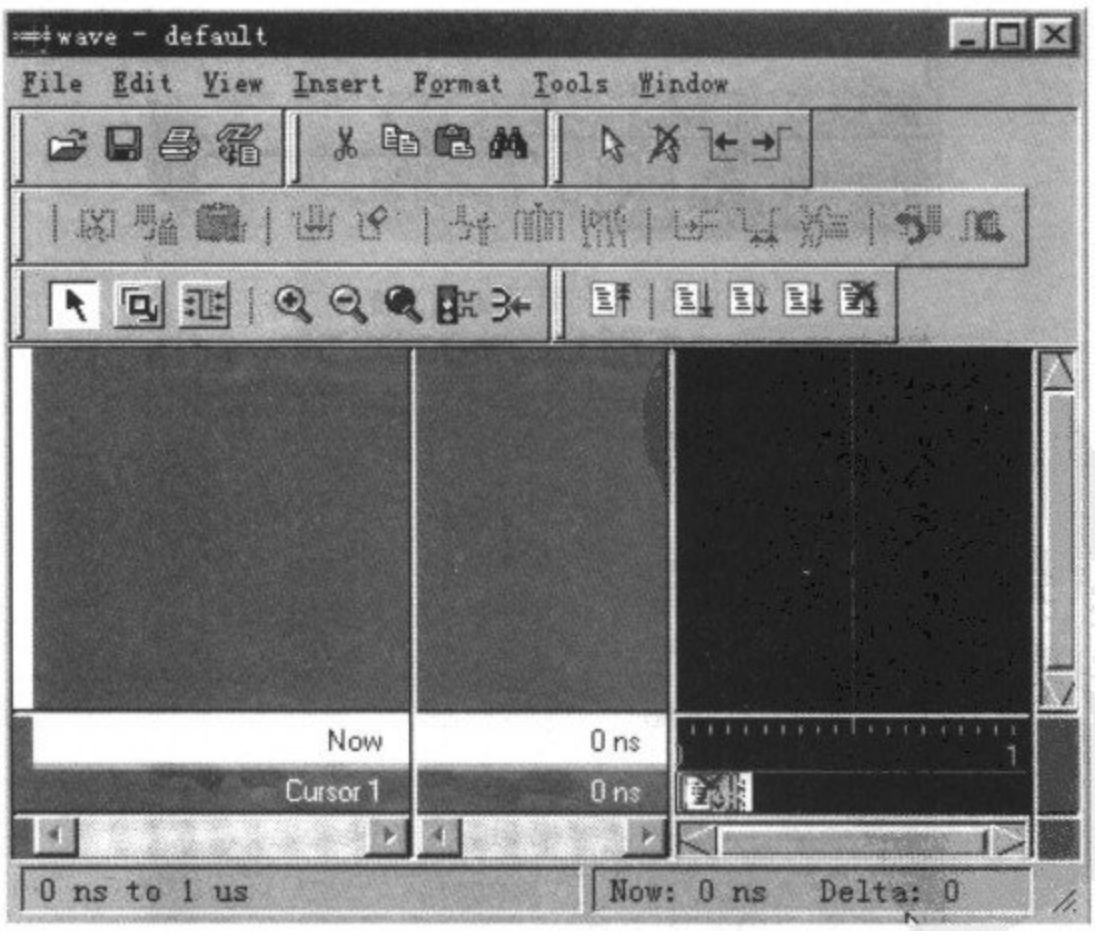


图 3-76　波形窗口

(2)在 Object 窗口中单击鼠标右键,在出现的快捷菜单中选择 Add to Wave→Signals in Region,将信号增加到波形窗口中,如图 3-77 所示。

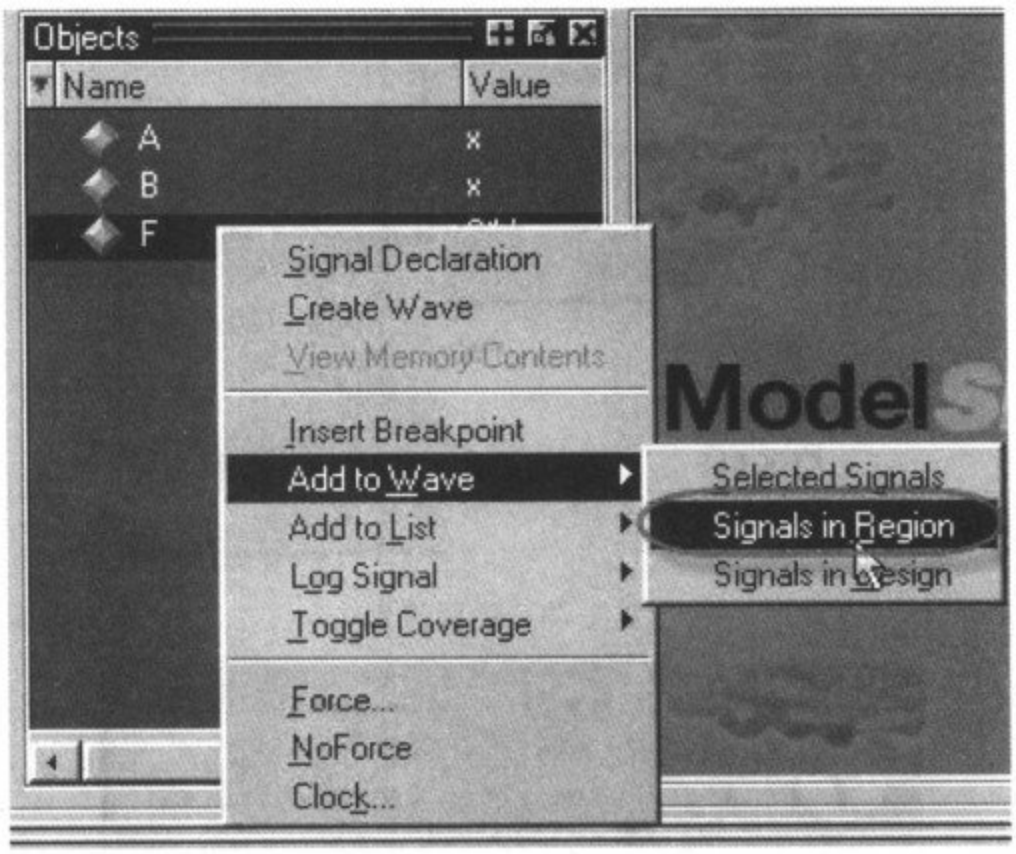


图 3-77　将信号增加到波形窗口中

(3)增加信号后的波形窗口如图 3-78 所示。

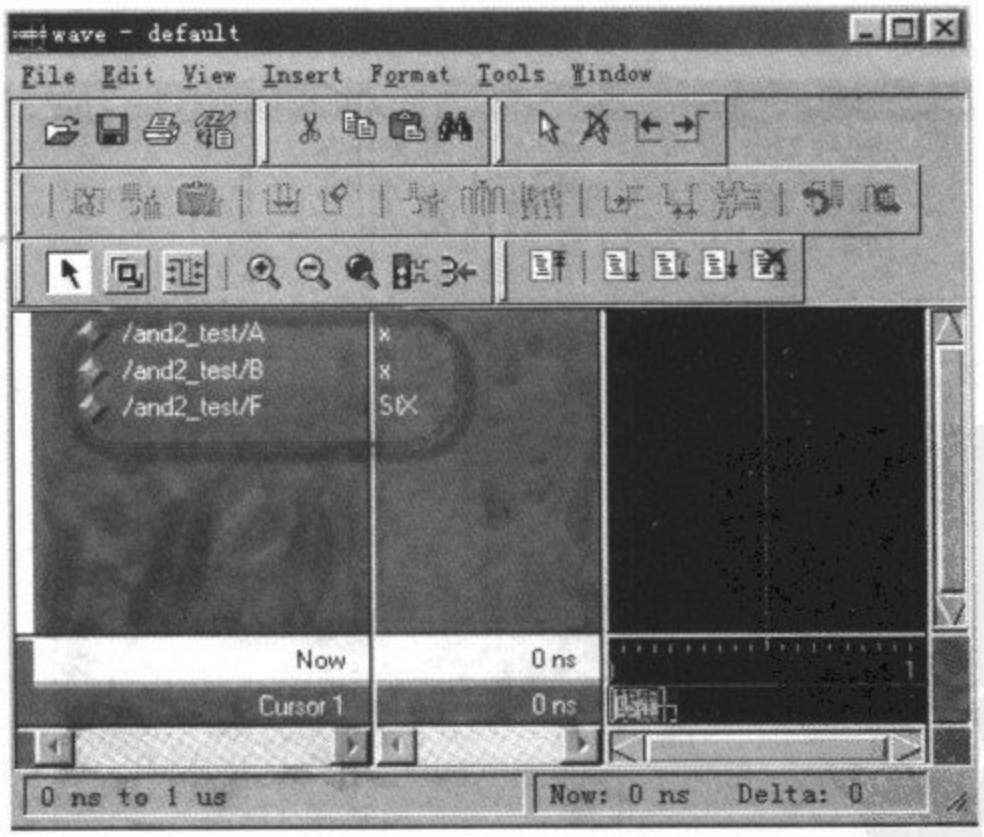


图 3-78　增加信号后的波形窗口

(4)单击 Modelsim SE 6.0 菜单栏 Simulate→Run→Run All,如图 3-79 所示。

(5)在出现的选择是否结束的对话框,点击“否”,如图 3-80 所示。

(6)在波形窗口中,可以观察到仿真波形,如图 3-81 所示。

从图中可以看出,输出与输入波形符合与门的关系。

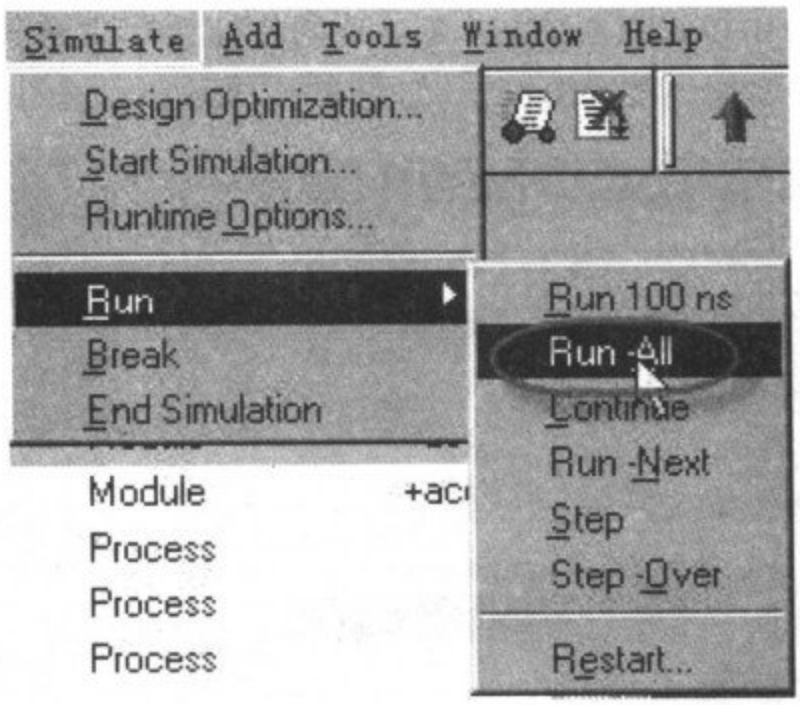


图 3-79　运行仿真

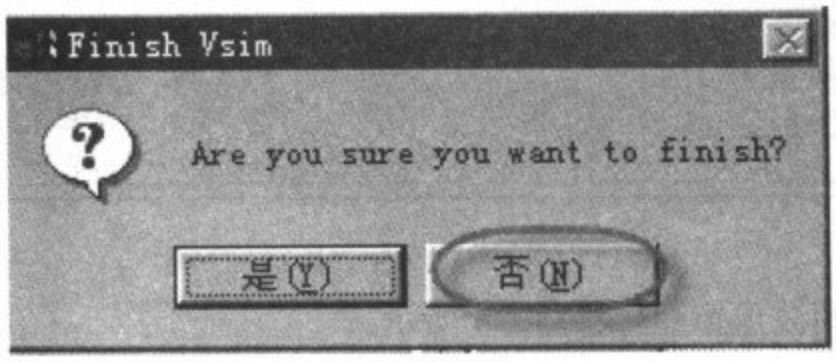


图 3-80　选择是否结束对话框

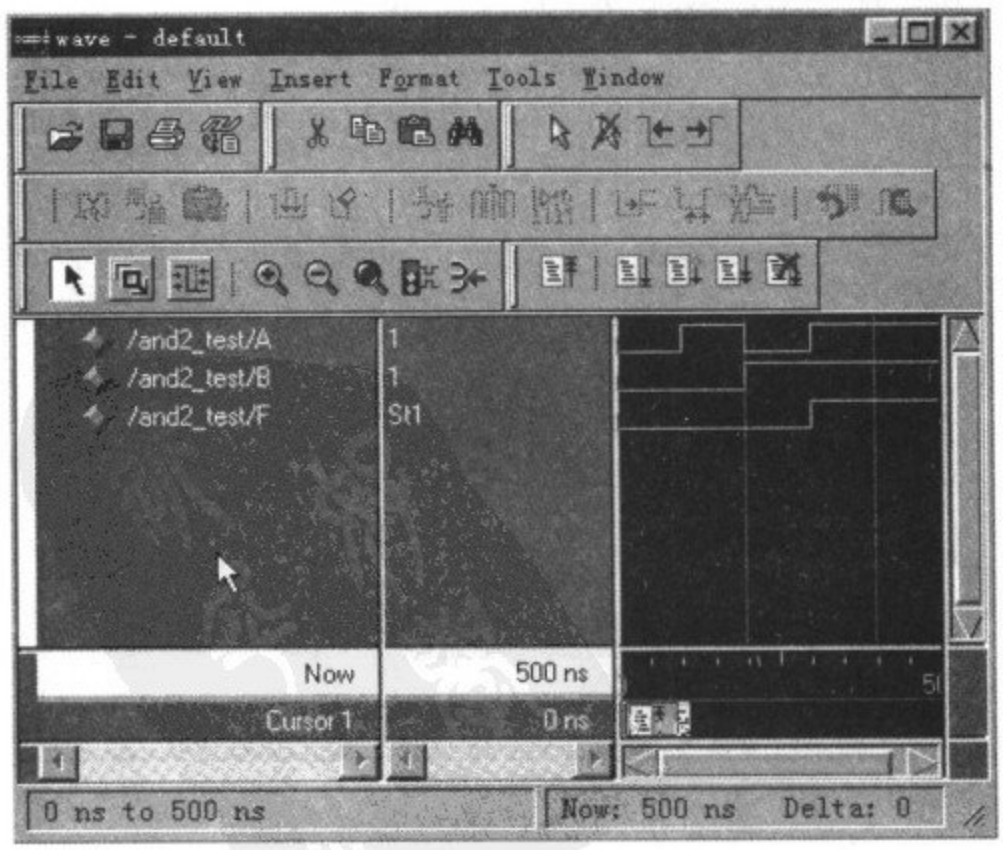


图 3-81　仿真波形

第四章　初识 Verilog HDL

随着电子系统设计自动化(electronic design automation,EDA)和超大规模可编程逻辑器件的快速发展,一类新型电子系统开发工具正在迅速普及,其中就有很重要的一类开发软件工具——硬件描述语言(Hardware Description Language,HDL)。目前 HDL 已经成为从事 EDA 的电子工程师必须掌握的工具。本章主要介绍硬件描述语言 Verilog HDL 的历史、功能、特点以及常用基本概念。

第一节　硬件描述语言概述

一、什么是硬件描述语言

硬件描述语言 HDL 是一种用形式化方法描述数字电路和系统的语言。其主要目的是用来编写设计文件,建立电子系统仿真模型,利用计算机和 EDA 工具,对用硬件描述语言建模的数字逻辑进行仿真,然后再自动综合,以生成符合要求且在电路结构上可以实现的数字逻辑网表,根据网表和某种工艺的器件,自动生成具体电路,然后生成该工艺条件下这种具体电路的延时模型,仿真验证无误后,写入 CPLD 和 FPGA 器件中。

二、硬件描述语言的发展

硬件描述语言发展至今约有 20 多年的历史,它成功地应用于系统开发的设计、综合、仿真和验证等各个阶段。到目前为止,在国外已有上百种硬件描述语言,PLD 厂商、高等学校、科研单位、EDA 公司都有自己的 HDL 语言,它们有些从 C 语言发展而来,有些则从 Pascal 语言发展而来。有些 HDL 语言已成为国际电气与电子工程师协会 IEEE(International Electrical&Electronic Engineer)标准,但大部分是企业标准。

在我国,比较有影响的主要有三种硬件描述语言:超高速集成电路硬件描述语言 VHDL、Verilog HDL 语言和 AHDL 语言。其中,前两种语言已成为 IEEE 标准,应用也最为广泛。

三、为何使用硬件描述语言

早期的数字逻辑电路设计规模比较小,也比较简单,设计时只能采用厂家提供的电路图输入工具进行。为了满足设计性能指标,往往需要花很长的时间进行手工布线,设计周期较长。

近年来,CPLD 和 FPGA 的设计规模越来越复杂,如果仍采用原理图输入的方法,将无法完成设计任务,必须采用 HDL 来进行设计。采用 HDL 进行设计时,由于 HDL 已标准化,可以很容易地把完成的设计移植到不同厂家的不同芯片中去,并在不同规模应用时

可以较容易地进行修改，在仿真验证时，测试向量也可以用该语言来描述，此外，采用HDL进行设计还具有与工艺无关性，这使得设计人员在功能设计、逻辑验证阶段可以不必过多考虑门级及工艺实现的具体细节，只需根据系统设计的要求，施加不同的约束条件，即可设计出实际电路。

第二节　Verilog HDL 基本知识

一、什么是 Verilog HDL

Verilog HDL 是目前应用最为广泛的硬件描述语言之一，它允许设计者用其来进行各种级别的逻辑设计，以及数字逻辑系统的仿真验证、时序分析和逻辑综合。

二、Verilog HDL 的发展

Verilog HDL 语言最初是于 1983 年由 Gateway Design Automation 公司为其模拟器产品开发的硬件建模语言。那时它只是一种专用语言。由于他们的模拟仿真器产品的广泛使用，Verilog HDL 作为一种便于使用且实用的语言逐渐为众多设计者所接受，在一次努力增加语言普及性的活动中，Verilog HDL 语言于 1990 年被推向公众领域，OVI 组织是促进 Verilog 发展的国际性组织，1992 年，OVI 决定致力于推广 Verilog OVI 标准成为 IEEE 标准，这一努力最后获得成功，Verilog 语言于 1995 年成为 IEEE 标准，称为 IEEE 1364-1995。2001 年，在原标准的基础上经过改进和补充，发布了 Verilog HDL IEEE 1364-2001 新标准。

三、Verilog HDL 与 VHDL 比较

Verilog HDL 和 VHDL 都是用于逻辑设计的硬件描述语言，并且都已成为 IEEE 标准。VHDL 是在 1987 年成为 IEEE 标准的，Verilog HDL 则在 1995 年才正式成为 IEEE 标准。

之所以 VHDL 比 VerilogHDL 早成为 IEEE 标准，是因为 VHDL 是美国军方组织开发的，而 Verilog HDL 则是从一个普通的民间公司的私有财产转化而来，基于 Verilog HDL 的优越性才成为 IEEE 标准，因而有更强的生命力。

Verilog HDL 和 VHDL 的共同的特点是：能抽象表示电路的行为和结构，支持逻辑设计中层次与范围的描述，可借用高级语言的精巧结构来简化电路行为的描述，具有电路仿真与验证机制以保证设计的正确性，支持电路描述由高层到低层的综合转换，硬件描述与实现工艺无关（有关工艺参数可通过语言提供的属性包括进去），便于文档管理，易于理解和移植等。

Verilog HDL 和 VHDL 又各有自己的优势和特点。Verilog HDL 早在 1983 年就已推出，至今已有近 20 多年的历史，因而 Verilog HDL 拥有更广泛的设计群体，其设计资源也比 VHDL 丰富。与 VHDL 相比，Verilog HDL 的最大优点是：它是一种非常容易掌握的硬件描述语言，只要有 C 语言的编程基础，通过一段时间的学习和实际操作，可以在较短的时间内掌握这种设计技术；而掌握 VHDL 设计技术就比较困难。这是因为

VHDL 不很直观，需要有 Ada 编程基础，一般需要至少半年以上的专业培训，才能掌握 VHDL 的基本设计技术。因此，Verilog HDL 作为学习 HDL 设计方法的入门和基础是比较合适的。

四、Verilog HDL 与 C 语言的比较

VerilogHDL 是在 C 语言的基础上发展而来的。从语法结构上看，Verilog HDL 与 C 语言有许多相似之处，继承和借鉴了 C 语言的很多语法结构，Verilog HDL 与 C 语言的运算符几乎完全相同。当然，Verilog HDL 作为一种硬件描述语言，与 C 语言还是有着本质的区别的。

第三节　Verilog HDL 模块介绍

一、什么是模块

简单地说，模块(**module**)是 Verilog 程序的基本设计单元。例如，图 4-1 就是一个二与门模块，设其模块名为 and_2，输入为 A、B，输出为 F。

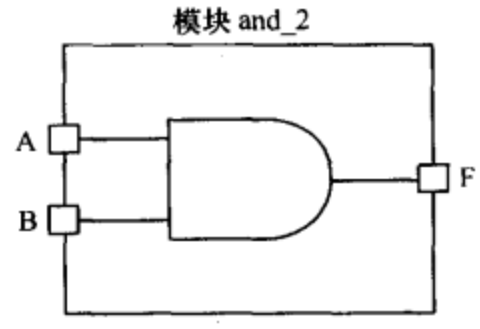


图 4-1　二与门模块

二、模块的结构

上面介绍的二与门电路用 Verilog 语言描述如下：

```
module and_2(A,B,F);    //模块名为 and_2，端口列表 A,B,F
input A,B;              //模块的输入端口为 A,B
output F;               //模块的输出端口为 F
wire A,B,F;             //定义信号的数据类型
assign F=A&B;           //逻辑功能描述
endmodule
```

该程序的第 1 行为模块的名字、模块的端口列表；第 2、3 行为输入、输出端口声明；第 4 行定义了端口的数据类型；在第 5 行中对输入、输出信号间的逻辑关系进行了描述。程序中“//”后面的内容为注释，注释语句要写在“/*”和“*/”之间(主要用于多行注释)或写在“//”后(主要用于单行注释)。注释行不被编译，仅起注释作用。

从这个简单的程序可以看出，Verilog 模块结构嵌在 **module** 和 **endmodule** 关键字之间，每个 Verilog 程序包括 4 个主要部分，即模块声明、端口定义、信号类型说明和逻辑功能描述。

1. 模块声明

模块声明包括模块名字和模块输入、输出端口列表。其格式如下：

module 模块名(端口 1，端口 2，……)；

模块结束的标志为关键字：**endmodule**

2. 端口定义

端口是模块与外界或其他模块连接和通信的信号线，端口有 3 种类型，分别是输入端

口(**input**)、输出端口(**output**)和输入输出端口(**inout**)。对模块的端口要明确说明,其格式为:

input　　端口名 1,端口名 2,……;　　//输入端口
output　　端口名 1,端口名 2,……;　　//输出端口
inout　　端口名 1,端口名 2,……;　　//输入/输出端口

注意事项　对于端口应注意以下 3 点:

(1)每个端口除了要声明是输入、输出还是双向端口外,还要声明其数据类型,是连线型(**wire**)还是寄存器型(**reg**),如果没有声明,则综合器将其默认为是 **wire** 型。

(2)输入和双向端口不能声明为寄存器型。

(3)在测试模块中不需要定义端口。

3. 信号类型声明

对模块中所用到的所有信号(包括端口信号、节点信号等)都必须进行数据类型的定义。Verilog 语言提供了各种信号类型,分别模拟实际电路中的各种物理连接和物理实体。常用的数据类型有 **wire**(连线型)和 **reg**(寄存器型)。如果信号的数据类型没有定义,则综合器将其默认为是 **wire** 型。

4. 逻辑功能定义

有多种方法可在模块中描述和定义逻辑功能,还可以调用函数(**function**)和任务(**task**)来描述逻辑功能。常用的有以下几种。

(1)用 **assign** 持续赋值语句定义

例如:**assign**　F=A&B;

assign 语句一般用于组合逻辑的赋值,称为持续赋值方式。这种方法简单,只需将逻辑表达式放在关键字 **assign** 后即可。

(2)调用元件(元件例化)

调用元件的方法类似于在电路图输入方式下调入图形符号来完成设计,这种方法侧重于电路的结构描述。

例如:

and U1(F,A,B);

该语句表示二与门 U1,输入端口为 A,B,输出端口为 F,在括号中的顺序必须是(输出,输入,输入)的形式,所以必须写成(F,A,B);U1 为实例名,实例名在具有行为功能的描述行里也可以省略。

另外,在 Verilog 语言中,还可以调用开关级元件和用户定义元件 UDP 来描述电路的结构。

(3)用 **always** 过程块赋值

always 过程块赋值主要用于时序逻辑电路,例如:对于 D 触发器,可采用以下语句进行描述。

```
always@(posedge clk) //每当 clk 上升沿到来时执行一遍 begin 和 end 块内的语句
begin
Q<=D;
end
```

第五章　Verilog HDL 数据类型与运算符

本章主要介绍 Verilog HDL 的基本数据类型和运算符，这些语法现象和 C 语言十分相似，甚至有些地方完全相同，但也有许多地方则完全不同。比如：Verilog HDL 的运算符与 C 语言的运算符很相像，而数据类型（**wire**、**reg** 等）则是 Verilog HDL 所特有的。必须深入理解硬件描述语言与软件编程语言的本质区别。

第一节　Verilog HDL 基本词法

Verilog HDL 程序是由各种符号流构成的，这些符号包括标识符、关键字、注释、空白符、数据类型和运算符等，本节先介绍标识符、关键字、注释、空白符，后面二节再介绍数据类型和运算符。

一、标识符

标识符是用来标识源程序中某个对象的名字的，这些对象可以是变量、语句、数据类型和函数等等。

Verilog HDL 中的标识符（identifier）可以是任意一组字母、数字、$ 符号和_（下划线）符号的组合，但标识符的第一个字符必须是字母或者下划线。另外，标识符是区分大小写的。

标识符在命名时，应当简单，含义清晰，尽量为每个标识符取一个有意义的名字，这样有助于阅读理解程序。

二、关键字

关键字则是编程语言保留的特殊标识符，它们具有固定名称和含义，在程序编写中不允许标识符与关键字相同。

需要注意的是，只有小写的关键词才是保留字。例如，标识符 **always**（这是个关键词）与标识符 ALWAYS（非关键词）是不同的。下面是 Verilog HDL 常使用的关键字：

always, and, assign, begin, buf, bufif0, bufif1, case, casex, casez, cmos, deassign, default, defparam, disable, edge, else, end, endcase, endmodule, endfunction, endprimitive, endspecify, endtable, endtask, event, for, force, forever, fork, function, highz0, highz1, if, initial, inout, input, integer, join, large, macromoduk, medium, module, nand, negedge, nmos, nor, not, notif0, notif1, or, output, parameter, pmos, posedge, primitive, pull0, pull1, pullup, pulldown, rcmos, reg, releses, repeat, mmos, rpmos, rtran, rtranif0, rtranif1, scalared, small, specity, specparam, strength, Strong0, strong1, supply0, supply1, table, task, time, tran,

tranif0, **tranif1**, **tri**, **tri0**, **tri1**, **triand**, **trior**, **trireg**, **vectored**, **wait**, **wand**, **weak0**, **weak1**, **while**, **wire**, **wor**, **xnor**, **xor**。

为便于读者阅读方便，书中的关键字全部采用了加粗字体。

三、注释

在 Verilog HDL 中，有两种形式的注释。一是单行注释，以“//”开始到本行结束，不允许续行。二是多行注释，以“/ * ”开始，到“ * /”结束。这两种注释方法和 C 语言是相同的。

四、空白符

在 Verilog HDL 代码中，空白符包括空格、tab、换行和换页。空白符使代码错落有致，阅读起来更方便。在综合时，空白符被忽略。

例如：**initial begin** Top = 3 'b001；#2 Top = 3 'b011； **end** 和下面的指令一样：

```
initial
  begin
  Top=3 'b001;
  #2 Top= 3 'b011;
  end
```

第二节　Verilog HDL 常量变量及其数据类型

任何程序设计都离不开对于数据的处理，一个程序如果没有数据，它就无法工作。Verilog HDL 共有近 20 种数据类型，包括 **wire** 型、**reg** 型、**integer** 型、**parameter** 型、**large** 型、**medium** 型、**scalared** 型、**time** 型、**small** 型、**tri** 型、**tri0** 型、**tril** 型、**triand** 型、**trior** 型、**trireg** 型、**vectored** 型、**wand** 型和 **wor** 型等。

在上面的数据类型中，最基本的是两类：一类称为连线型，另一类称为寄存器型。在这两类中最常用的是：**wire** 型、**reg** 型、**integer** 型和 **parameter** 型 4 种数据类型。

Verilog HDL 也有常量和变量之分，它们分别属于以上这些数据类型。

一、常量及其数据类型

常量是在程序运行过程中不能改变值的量，常见的常量有数字和字符串。

Verilog HDL 有下面 4 种基本的逻辑状态：

0：低电平、逻辑 0 或“非”；

1：高电平、逻辑 1 或“真”；

x 或 X：不定值或未知的逻辑状态；

z 或 Z：高阻态。

Verilog HDL 中的常量是由以上这 4 类基本值组成的。其中，x 和 z 都不区分大小写，另外，z 还有一种表示形式是可以写作“?”。

1. 数字

数字主要有以下几种形式。

(1)整数(integer)

整数按如下方式书写：+/－<位宽>'<进制><数字>

其中，进制有如下 4 种表示形式：

二进制(b 或 B)；

十进制(d 或 D 或缺省)；

十六进制(h 或 H)；

八进制(o 或 O)。

在书写中，十六进制中的 a 到 f 值不区分大小写。

另外，整数可以带符号(正、负号)，并且正、负号应写在最左边。负数通常表示为二进制补码的形式。

下面是一些合法的整数的例子；

```
8 'bll000001      //位宽为 8 位的二进制数 11000001
8 'hf4            //位宽为 8 位的十六进制数 f4
5 'O32            //位宽为 5 位的八进制数 32
4 'D10            //位宽为 4 位的十进制数 10
4 'hz             //4 位十六进制数 z，即 zzzz
4 'Blx_11         //允许使用下划线，下划线本身没有意义，但可提高程序的可读性
8 'h   3 e        //在位宽和字符之间，以及进制和数值之间允许出现空格
```

下面是一些不正确的书写整数的例子：

```
8 'd－5           //非法，数值不能为负，有负号应放最左边
3 'b001           //非法，'和进制 b 之间不允许出现空格
(5＋2)'b2         //非法，位宽不能为表达式
```

注意事项 在书写和使用数字中需注意下面一些问题：

①当数字不说明位宽时，默认值为 32 位。

②x 或 z 在二进制中代表 1 位 x 或 z，在八进制中代表 3 位 x 或 z，在十六进制值中代表 4 位 x 或 z，其代表的宽度取决于所用的进制。

例如：

```
8 'b111lxxxx      //等价于 8 'hfx
8 'b1010zzzz      //等价于 8 'haz
```

——如果没有定义一个整数的位宽，其宽度为相应值中定义的位数。

例如：

```
'h1a              //8 位十六进制数
```

——如果定义的位宽比实际的位数长，通常在左边填“0”补位。但如果数最左边一位为 x 或 z，就相应地用 x 或 z 在左边补位。

例如：

```
4 'b10              //左边补 0,0010
4 'bz0              //左边补 z,zzz0
```

如果定义的位宽比实际的位数小，那么最左边的位相应地被截断。

例如：

```
4 'b1001_1000    //与 4 'b1000 相等
```

(2)实数(Real)

实数有下面两种表示法：

①十进制表示法

例如：2.1，3.0 等。

②科学计数法

例如：2.2e2　　//与 220 相同

2. 字符串

字符串是双引号内的字符序列。字符串不能分成多行书写。

例如："Verilog HDL"

3. 参数型(parameter)

在 Vefilog HDL 中，用 parameter 来定义符号常量，即用 **parameter** 来定义一个标志符代表一个常量，称为符号常量。其定义格式如下：

parameter 参数名 1＝表达式 1，参数名 2＝表达式 2……；

例如：

```
parameter msb=8;          //定义参数 msb 代表常数 8(十进制)
parameter sizee=8 'h0f;   //定义参数 size 代表常量 0f(16 进制)
```

二、变量及其数据类型

在程序运行过程中，其值可以被改变的量称为变量，在 Verilog HDL 中，变量的数据类型有很多种，下面简要进行介绍。

1. 连线型

连线型数据相当于硬件电路中的各种物理连接，其特点是输出的值紧跟输入值的变化而变化。Verilog HDL 提供了多种连线型变量，如 **wire**，**tri**，**wor**，**wand**，**triand**，**tri0**，**tri1**，**supply0**，**supply1** 等，其中，**wire** 是最常用的连线型变量，下面以 **wire** 型数据为例进行重点介绍。

wire 型数据常用来表示以 **assign** 语句赋值的组合逻辑信号。Vedlog HDL 模块中的输入输出信号类型缺省时自动定义为 **wire** 型。**wire** 型信号可以用作任何表达式的输入，也可以用作"**assign**"语句和实例元件的输出。对于综合器而言，其取值可为 0、1、x，z。

wire 型变量的定义格式如下：

wire　数据名 1，数据名 2……；

例如：**wire** A，B，F；　　//定义了 3 个 **wire** 型变量 A、B 和 F

上面 3 个变量 A、B、F 的宽度都是 1 位，若定义一个多位的 **wire** 型数据，则定义格式如下：

wire[n－1:0]数据名 1，数据名 2……；　　//数据的宽度为 n 位

或：

wire[n:1]　数据名 1，数据名 2……；　　//数据的宽度为 n 位

例如：

wire[7:0]　data；　　//data 的宽度是 8 位

wire[15:0]　addr；　　//addr 的宽度是 16 位

2. 寄存器型

寄存器型变量对应的是具有状态保持作用的电路元件，如触发器、寄存器等。寄存器型变量与连线型变量的根本区别在于：寄存器型变量需要被明确地赋值，并且寄存器型变量在被重新赋值前一直保持原值，默认初始值为不定值 x。在设计中必须将寄存器型变量放在过程语句(如 initial、always)中，通过过程赋值语句赋值。另外，在 always、initial 等过程块内，被赋值的每一个信号都必须定义成寄存器型。

Verilog HDL 有 4 种寄存器型变量，分别是 **reg**、**integer**、**real** 和 **time**，其中，**reg** 型变量是最常用的一种寄存器型变量，下面着重对其进行介绍。

reg 型变量的定义格式类似于 **wire** 型，如下所示：

reg　数据名 1，数据名 2……；

例如：**reg** Q；　　//定义了一个 **reg** 型变量 Q，即定义了一个 1 位寄存器

上面变量 Q 的宽度是一位，若定义一个多位的 **reg** 型数据，则定义格式如下：

reg[n－1:0]数据名 1，数据名 2……；　　//数据的宽度为 *n* 位

或：

reg[n:1]　数据名 1，数据名 2……；　　//数据的宽度为 *n* 位

例如：

reg[7:0]　QOUT；　　//QOUT 的宽度是 8 位，即定义了一个 8 位寄存器

3. memory 型

Verilog HDL 通过对 **reg** 型变量建立数组来对存储器建模，可以描述 RAM 型存储器、ROM 存储器和 **reg** 文件。数组中的每一个单元通过一个数组索引进行寻址。在 Verilog HDL 语言中没有多维数组存在。memory 型数据是通过扩展 **reg** 型数据的地址范围来生成的。其格式如下：

reg[n－1:0]存储器名[m－1:0]；

或：

reg[n－1:0]存储器名[m:1]；

这里，**reg**[n－1:0]定义了存储器中每一个存储单元的大小，即该存储单元是一个 n 位的寄存器。存储器名后的[m－1:0]或[m:1]则定义了该存储器中有多少个这样的寄存器。

例如：

reg[7:0] mema[255:0];

这个例子定义了一个名为 mema 的存储器，该存储器有 256 个 8 位存储器。该存储器的地址为 0～255。

注意事项 尽管 memory 型数据与 reg 型数据的定义格式很相似，但要注意其不同之处。如 1 个由 n 个 1 位寄存器构成的存储器组是不同于 1 个 n 位的寄存器的。

reg[n－1:0] QOUT; //一个 n 位的寄存器
reg mema[n－1:0] //一个由 n 个 1 位寄存器构成的存储器

一个 n 位的寄存器可以在一条赋值语句里进行赋值，而一个完整的存储器则不行。例如：

QOUT＝1; //合法赋值语句
mema＝1; //非法赋值语句

如果想对 memory 中的存储单元进行读/写操作，必须指定该单元在存储器中的地址。下面的写法是正确的。

mema[3]＝1; //给 mema 中的第 3 个存储单元赋值为 1

第三节 Verilog HDL 运算符

运算符就是完成某种特定运算的符号。Verilog HDL 提供了丰富的运算符，运算符按其表达式中与运算符的关系可分为单目运算符、双目运算符和三目运算符。单目就是指需要有一个运算对象，双目就要求有两个运算对象，三目则要 3 个运算对象。表达式则是由运算及运算对象所组成的具有特定含义的式子。表达式后面加“;”号就构成了一个表达式语句。运算符按功能分的话，包括算术运算符、逻辑运算符、关系运算符、等式运算符、缩减运算符、条件运算符、位运算符、移位运算符和拼接运算符等，下面对 Verilog HDL 语言中的运算符分别进行介绍。

一、算术运算符

Verilog HDL 有以下几种算术运算符，如表 5-1 所示。

表 5-1 算术运算术的功能

算术运算符号	功 能	算术运算符号	功 能
＋	加法	/	除法
－	减法	%	求余
*	乘法		

算术运算符都是双目运算符，在进行整数除法运算时，结果值要略去小数部分。

另外，求余运算符的对象只能是整型，在%运算符左侧的运算数为被除数，右侧的运算数为除数，运算结果是两数相除后所得的余数。

二、逻辑运算符

逻辑运算符用于求条件式的逻辑值，Verilog HDL 有 3 种逻辑运算符，如表 5-2 所示。

用逻辑运算符将关系表达式或逻辑量连接起来就是逻辑表达式。逻辑表达式的一般形式如下。

逻辑与:操作数 1 && 操作数 2

逻辑或:操作数 1 ||操作数 2

逻辑非:！ 操作数

表 5-2 逻辑运算符

逻辑运算符	功 能
&&	逻辑与
||	逻辑或
!	逻辑非

逻辑与,是指 2 个操作数都为真时,结果才为真,如果任何一个操作数为假,则结果为假。

逻辑或,是指只要 2 个操作数中有一个为真时,运算结果就为真,只有当操作数都不为真时,逻辑运算结果才为假。

逻辑非则是把逻辑运算结果值取反,如果操作数的值为真,进行逻辑非运算后则结果变为假,操作数运算值为假时,逻辑结果为真。

重点提示 需要说明的是,如果操作数不只 1 位的话,则应将操作数作为一个整体来对待,即如果操作数全“0”,则相当于逻辑 0,但只要某 1 位是“1”,则操作数就应该整体看作逻辑 1。逻辑运算符的操作结果是 1 位的。

三、位运算符

位运算符的作用是将操作数按对应位分别进行逻辑运算,Verilog HDL 中共有 5 种位运算符,如表 5-3 所示。

按位取反的真值表如表 5-4 所示。

表 5-3 位运算符

位运算符	功能
&	按位与
|	按位或
^	按位异或
~	按位取反
^~或~^	按位同或

表 5-4 按位取反真值表

~	结果
1	0
0	1
x	x

按位与的真值表如表 5-5 所示。

按位或的真值表如表 5-6 所示。

表 5-5 按位与的真值表

&	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

表 5-6 按位或的真值表

|	0	1	x
0	0	1	x
1	1	1	1
x	x	1	x

按位异步的真值表如表 5-7 所示。

按位同或的真值表如表 5-8 所示。

表 5-7 按位异或的真值表

^	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

表 5-8 按位同或的真值表

^~	0	1	x
0	1	0	x
1	0	1	x
x	x	x	x

例如：若 a=4 'bll00；b=4 'b1010，则：

~a=4 'b0011

a&b=4 'b1000

a|b=4 'blll0

a^b=4 'b0110

a^~b=4 'b1001

需要注意的是：两个不同长度的数据进行位运算时，会自动地将两个操作数按右端对齐，位数少的操作数会在高位用“0”补齐。

四、关系运算符

关系运算符用来比较变量的值或常数的值。Verilog HDL 有 6 种关系运算符，如表 5-9 所示。

表 5-9　关系运算符

关系运算符	功能	关系运算符	功能
>	大于	<	小于
>=	大于等于	<=	小于等于

当两个表达式用关系运算符连接起来时，这时就是关系表达式。关系表达式通常是用来判别某个条件是否满足。在进行关系运算时，如果声明的关系是假，则返回值是 0；如果声明的关系是真，则返回值是 1，如果某个操作数的值不定，则关系的结果是模糊的，返回值是不定值。

另外要注意的是：“<=”操作符也用于表示信号的一种赋值操作。

五、等式运算符

等式运算符有以下 4 种，如表 5-10 所示。

表 5-10　等式运算符

等式运算符	功能	等式运算符	功能
==	等于	===	全等
!=	不等于	!==	不全等

这 4 种运算符都是双目运算符，得到的结果是 1 位的逻辑值，如果得到 1，说明声明的关系为真；如得到 0，说明声明的关系为假。

相等运算符(==)和全等运算符(===)的区别是：对于“==”，参与比较的两个操作数必须逐位相等，其相等比较的结果才为 1，如果某些位是不定态或高阻值，其相等比较得到的结果是不定值；而全等比较(===)则是对这些不定态或高阻值的位也进行比较，两个操作数必须完全一致，其结果才是 1，否则结果是 0。

相等运算符(==)和全等运算符(===)的真值表分别见表 5-11 和表 5-12 所示。

表 5-11　相等运算真值表

==	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

表 5-12　全等运算真值表

===	0	1	x	z
0	1	0	0	0
1	0	1	0	0
x	0	0	1	0
z	0	0	0	1

比如:若变量 a＝4 'blx01,b＝4 'blx01,则“a＝＝b”得到的结果为不定值 x,而“a＝＝＝b”得到的结果为 1。

六、缩位运算符

缩减运算符是单目运算符,共有 6 种,如表 5-13 所示。

表 5-13　缩位运算符

缩位运算符	功能	缩位运算符	功能
&	与	~|	或非
~&	与非	^	异或
|	或	^~或~^	同或

缩位运算符与位运算符的逻辑运算法则一样,但其运算过程不同。位运算是对操作数的相应位进行逻辑运算,操作数是几位数,则运算结果也是几位数;而缩减运算则不同,缩减运算是对单个操作数进行逻辑递推运算,最后的运算结果是 1 位的二进制数。

缩减运算的具体运算过程是这样的:第一步先将操作数的第 1 位与第 2 位进行逻辑运算;第二步将运算结果与第 3 位进行逻辑运算;依次类推,直至最后 1 位。

例如:

```
reg[3:0]  a;
b=&a;      //等效于 b=((a[0]&a[1])&a[2])&a[3];
```

再比如:

若 a=4 'bll00,则有:

```
&a=0;      //只有 a 的各位都为 1 时,其与缩减运算的值才为 l
|a=1;      //只有 a 的各位都为 0 时,其或缩减运算的值才为 0
```

七、移位运算符

Verilog HDL 的移位运算符只有两个:“>>”(右移)和“<<”(左移)

移位运算符的用法为:

```
a>>n
```

或

```
a<<n
```

其中 a 表示要移位的操作数,n 表示要移多少位。这种两种移位运算都是用 0 来填补移出的空位。

例如；

若 a=4 'b1001，则：

a>>2 的值为 4 'b0010；　　//将 a 右移 2 位，用 0 添补移出的位

a<<2 的值为 4 'b0100；　　//将 a 左移 2 位，用 0 添补移出的位

八、条件运算符

和 C 语言一样，Verilog HDL 中也有一个三目运算符，它就是“?:”条件运算符，它要求有 3 个运算对象。它可以把 3 个表达式连接构成一个条件表达式。条件表达式的一般形式如下：

逻辑表达式?　表达式 1 ：表达式 2

条件运算符的作用简单来说就是根据逻辑表达式的值选择使用表达式的值。当逻辑表达式的值为真时(非 0 值)时，整个表达式的值为表达式 1 的值；当逻辑表达式的值为假(值为 0)时，整个表达式的值为表达式 2 的值。要注意的是，条件表达式中逻辑表达式的类型可以与表达式 1 和表达式 2 的类型不一样。

例如，对如图 5-1 所示的二选一数据选择器可用条件运算符描述为：

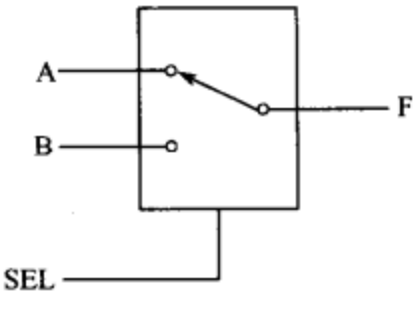


图 5-1　二选一数据选择器

F=SEL? B:A;　　//SEL 为 1 时 F=B;SEL 为 0 时 F=A

或

F=(SEL==0)? A:B; //SEL 为 1 时 F=B;SEL 为 0 时 F=A

九、位拼接运算符

在 Verilog HDL 中，有一个特殊的运算符，即位拼接运算符{ }，该运算符将两个或多个信号的某些位拼接起来，使用如下：

{信号 1 的某几位，信号 2 的某几位，……，信号 n 的某几位}

例如，在进行加法运算时，可将和与进位输出拼接在一起使用：

```
output[3:0]  sum;                    //sum 代表和
output  cout;                        //cout 为进位输出
input[3:0]  ina,inb;                 //ina,inb 为两个加数
input  cin;                          //cin 为进位输入
assign  {cout,sum}=ina+inb+cin;      //进位与和拼接在一起
```

位拼接可以嵌套使用，还可以用重复法来简化书写，如：

{3{a,b}}等同于{{a,b},{a,b},{a,b}}，也等同于{a,b,a,b,a,b}。

重点提示　运算符的优先级

以上运算符的优先级见表 5-14 所示。在书写程序时，建议用括号( )来控制运算的优先级，这样不但能有效地避免错误，而且也增加了程序的可读性。

表 5-14　运算符的优先级

运算符	优先级别
! ~	高优先级 ↓ 低优先级
* / %	
+ -	
<< >>	
< <= > >=	
== ! = === ! ==	
&	
^ ~	
|	
&&	
||	
?:	

第六章　Verilog HDL 基本语句

Verilog HDL 提供了相当丰富的程序控制语句。这些语句主要包括过程语句、块语句、赋值语句、条件语句、循环语句、编译向导语句、任务(**task**)、函数(**function**)、系统任务与系统函数等等，语句可以说是组成程序的灵魂。学习和掌握语句的用法是 Verilog HDL 学习中的重点，Verilog HDL 的语句中，既有和 C 语言相似或相同的地方，也有 Verilog HDL 特有的语句，阅读本章时应注意对照和理解。

第一节　赋值语句

在 Verilog HDL 中，赋值语句主要有两种，一种是持续赋值语句，另一种是行为赋值语句，下面分别进行介绍。

一、持续赋值语句

持续赋值语句用来驱动连线型变量，这一连线型变量必须已经事先定义过，只要输入端操作数的值发生变化，该语句就重新计算并刷新赋值结果，持续赋值语句一般用来描述组合逻辑电路。

例 1：用持续赋值语句描述二与门电路。

```
module and_2(A,B,F);    //模块名为 and_2，端口列表 A,B,F
input  A,B;             //模块的输入端口为 A,B
output F;               //模块的输出端口为 F
assign F=A&B;           //数据流描述
endmodule
```

在上面的赋值中，输出变量 F 和输入变量 A、B 都为 **wire** 型变量(默认为 **wire** 类型)，A 和 B 信号的任何变化都将随时反映到 F 上来。

二、过程赋值语句

上面介绍的持续赋值语句多用于 **wire** 类型变量，下面介绍的过程赋值语句则多用于对 **reg** 型变量进行赋值。过程赋值有非阻塞赋值和阻塞赋值两种方式。

1. 非阻塞赋值方式

非阻塞赋值方式的符号为“<=”，例如：b<=a；意义是，在整个过程块结束时，将 a 的值赋于 b。

非阻塞赋值方式符号和小于等于号是一样的，但意义完全不同，小于等于号是关系运算符，用于比较大小，而非阻塞赋值语句则用于赋值操作。

2. 阻塞赋值方式

阻塞赋值的符号为“=”，例如：b=a；意义是，将 a 的值立即赋于 b，也就是说，阻塞赋值在该语句结束时就立即完成赋值操作，即 b 的值在该条语句结束后立刻改变。

如果在一个块语句中有多条阻塞赋值语句，那么在前面的赋值语句没有完成之前，后面的语句就不能被执行，仿佛被阻塞了一样，因此称为阻塞赋值方式。

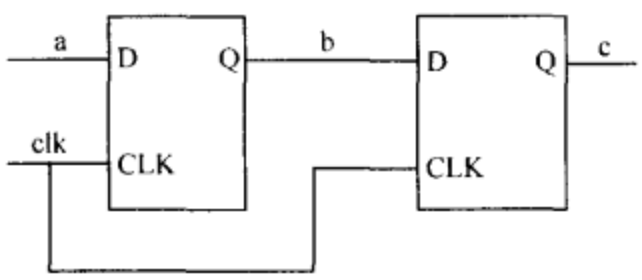
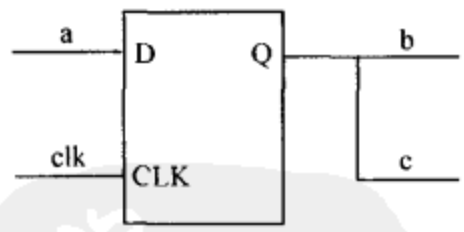
为便于读者对非阻塞赋值和阻塞赋值有一个清楚的认识，下面举例进行说明，如表 6-1所示。

例 2：非阻塞赋值和阻塞赋值对照说明。

表 6-1　非阻塞赋值和阻塞赋值对照说明

赋值方式	非阻塞赋值	阻塞赋值

程序	**module** non_block(c,b,a,clk);
	output c,b;
	input clk,a;
	reg c,b;
	always@(posedge clk)
	　　begin
	　　b<=a;
	　　c<=b;
	　　end
	endmodule
		module block(c,b,a,clk);
		output c,b;
		input clk,a;
		reg c,b;
		always@(posedge clk)
		　　begin
		　　b=a;
		　　c=b;
		　　end
		endmodule

说明	**always** 块中用了非阻塞赋值方式，**always** 块中定义了二个 **reg** 型输出信号 c、b，当 clk 上升沿到来时，b 就等于 a，c 就等于 b，但赋值是在 **always** 块结束后进行的	**always** 块中用了阻塞赋值方式，**always** 块中定义了二个 **reg** 型输出信号 c、b，当 clk 上升沿到来时，b 马上取 a 的值，c 马上取 b 的值，即 c=a
描述的电路		

将上面两段代码用 MAX+plusII 软件进行仿真，可分别得到如图 6-1(非阻塞赋值波形)和图 6-2(阻塞赋值波形)所示的波形图。

具体仿真方法可根据第三章介绍的步骤进行，本实例附在光盘的 ch_6/non_block 和 ch_6/block 文件中。

从波形图中可以看出：对于非阻塞赋值，c 的值落后 b 的值一个时钟周期，这是因为非阻塞赋值语句 b<=a 和 c<=b 是同时执行的。因此，每次执行完后，b 的值得到更新，而 c 的值仍是上一时钟周期的 b 值。对于阻塞赋值，阻塞赋值 b=a 和 c=b 两条语句是顺序执行的。c 的值和 b 的值一样。

为了用阻塞赋值方式完成与上述非阻塞赋值同样的功能，可采用两个“**always**”块来

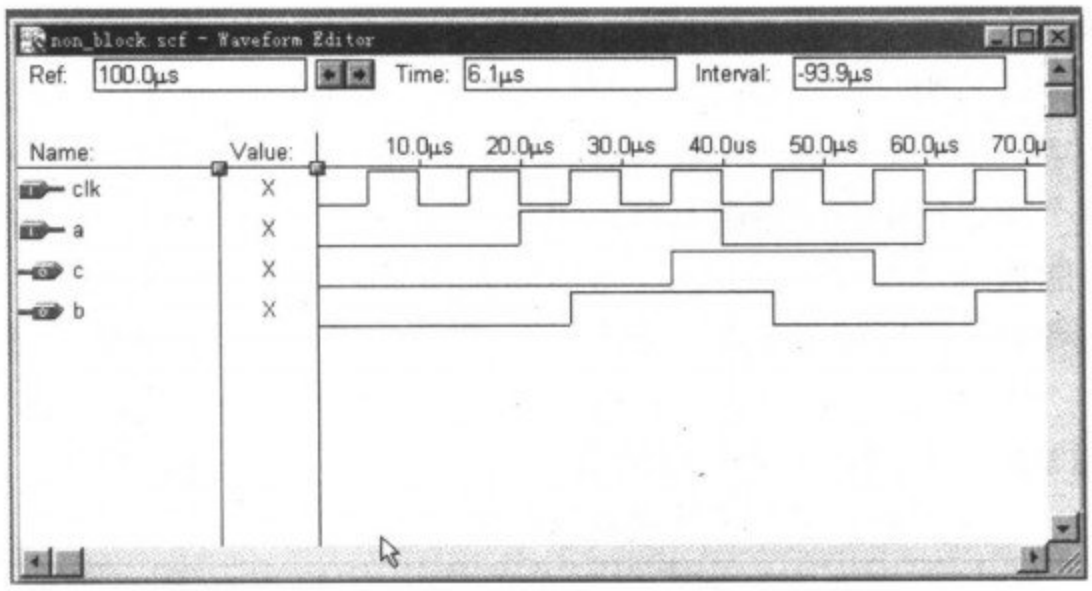


图 6-1　非阻塞赋值仿真波形图

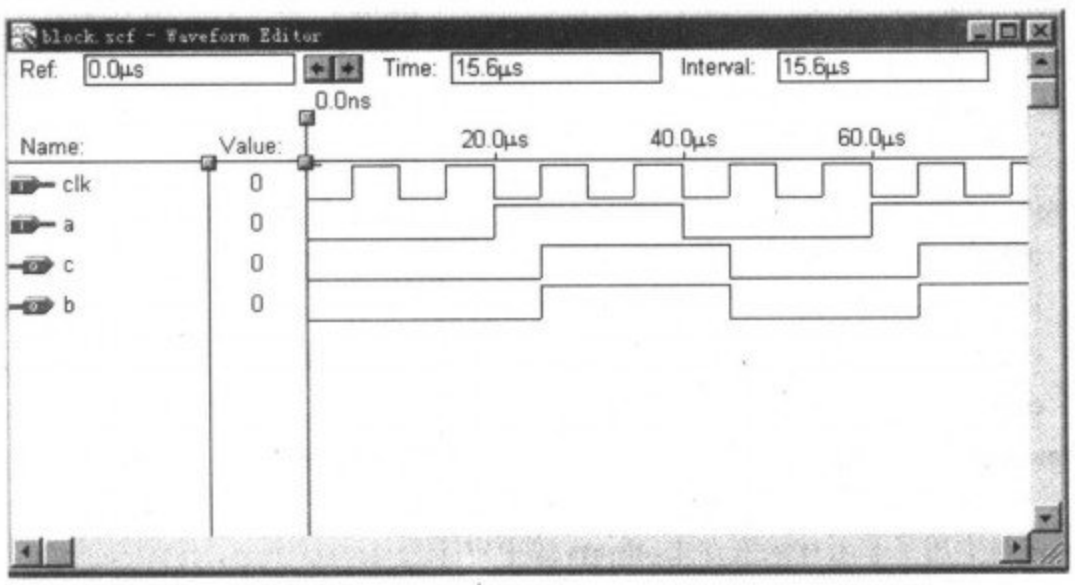


图 6-2　阻塞赋值仿真波形图

实现，两个 **always** 过程块是并发执行的。程序如下：

```
module  non_block(c,b,a,clk);
output  c,b;
input  clk,a;
reg  c,b;
always@(posedge clk)
    begin
    b=a;
    end
always@(posedge clk)
    begin
    c=b;
    end
endmodule
```

方法技巧　非阻塞赋值“<=”与阻塞赋值“=”与的区别在于：非阻塞赋值“<=”右端表达式计算完后并不立即赋给左端，而是同时启动下一条语句继续执行，我们可以将其

理解为所有右端表达式在进程开始时同时计算，是并行的，计算完后，等进程结束时同时分别赋给左端变量。而阻塞赋值语句在每个“=”右端表达式计算完后立即赋给左端变量，即赋值语句 b=a，执行完后 b 是立即更新的，同时，只有 b=a 执行完后才可执行语句 c=b，前一条语句的执行结果直接影响到后面语句的执行结果，即这种赋值方式是串行的。

非阻塞赋值“<=”不能用于“**assign**”持续赋值中，而只能用于对寄存器型变量进行赋值，即用在“**initial**”和“**always**”等过程块中；阻塞赋值则既能用于“**assign**”持续赋值，也能用于“**initial**”和“**always**”等过程赋值中。

对于时序逻辑描述和建模，应尽量使用非阻塞赋值方式。以免发生不必要的错误。

第二节　块语句

块语句通常由两条语句或多条语句组合在一起，使其在格式上看起来像是一条语句。块语句有两种：一种是串行块语句 **begin-end**，另一种是并行块语句 **fork-join**，下面分别进行介绍。

一、串行块语句 begin-end

串行块语句具有以下特点：

begin-end 块内的语句是按顺序执行的，即只有上面一条语句执行完后，下面的语句才能执行；**begin-end** 块内的每条语句的延迟时间是相对于前一条语句的仿真时间而言的；直到最后一条语句执行完，程序流程控制才跳出该语句块。

begin-end 串行块的一般格式如下：

```
begin
  语句 1
  语句 2
  ⋮
  语句 n
end
```

例如：

```
begin
  b=a;
  c=b;
end
```

由于 **begin-end** 块内的语句顺序执行，在最后，将 b、c 的值都更新为 a 的值。该 **begin-end** 块执行完后，b、c 的值是相同的。

例 3：用 **begin-end** 块语句产生一段周期为一个时间单位的信号波形。

```
`timescale 1ns/100ps    //时间单位为 1ns，时间精度为 100ps
module sign_wave;
```

```
reg wave;
parameter d=10;          //声明 d 是一个参数
initial
    begin
        wave=0;
      #(d/2)  wave=1;
      #(d/2)  wave=0;
      #(d/2)  wave=1;
      #(d/2)  $finish;    //系统任务,表示仿真结束
    end
initial $monitor( $time,,, "wave=%b",wave);//系统任务,输出结果
endmodule
```

上面的程序用 ModelSim 编译仿真后,可得到一段周期为一个时间单位的信号波形。

二、并行块语句 fork-join

并行块语句具有以下特点:

fork-join 块内的语句是同时执行的,即程序一进入该并行块,块内的语句就开始同时执行;在进行仿真时,fork-join 并行块中的每条语句前面的延时都是相对于该并行块的起始执行时间的;当按时间时序在最后的语句执行完后,程序流程控制才跳出该语句块。

fork-join 并行块的一般格式如下:

```
fork
  语句 1
  语句 2
  ⋮
  语句 n
join
```

例如:

```
fork
  b=a;
  c=b;
join
```

由于 **fork-join** 并行块中的语句是同时执行的,在上面的块语句执行完后,b 更新为 a 的值,而 c 的值更新为没有改变前的 b 的值,故执行完后,b 与 c 的值是不同的。

例 4:用 **fork-join** 块语句产生一段周期为一个时间单位的信号波形。

```
`timescale 1ns/100ps    //时间单位为 1ns,时间精度为 100ps
module sign_wave;
reg wave;
```

```
parameter d=5;          //声明 d 是一个参数
initial
    fork
        wave=0;
    #(d)   wave=1;
    #(2*d)   wave=0;
    #(3*d)   wave=1;
    #(4*d)    $finish;     //系统任务,表示仿真结束
    join
initial  $monitor( $time,,, "wave=%b",wave);//系统任务,输出结果
endmodule
```

上面的程序用 ModelSim 编译仿真后,可得到一段周期为一个时间单位的信号波形。

第三节　过程语句

Verilog HDL 有两个过程语句:**initial** 和 **always** 过程块。一个程序模块可以有多个 **initial** 和 **always** 过程块。每个 **initial** 和 **always** 说明语句在仿真时同时执行。**initial** 语句只执行一次,而 **always** 语句则是不断地重复活动着,直到仿真过程结束;但 **always** 语句后跟着的过程块是否运行,则要看它的触发条件是否满足。如果满足则运行过程块一次,若再次满足则再运行一次,直至仿真过程结束。在一个模块中,使用 **initial** 和 **always** 语句的次数是不受限制的,它们都是同时开始运行的。

一、initial 过程语句

initial 语句不带触发条件,**initial** 过程中的块语句沿时间轴只执行一次。**initial** 语句通常用于仿真模块中对激励向量的描述,或用于给寄存器变量赋初值,它是面向模拟仿真的过程语句,通常不能被逻辑综合工具所支持。

initial 语句的使用格式如下:

```
initial
  begin
   语句 1;
   语句 2;
   :
   语句 n
  end
```

例 5:用 initial 语句对存储器进行初始化。

程序如下:

```
initial
  begin
```

```
    for(index=0;index<size;index=index+1)
    memory[index]=0;                          //初始化赋值
  end
```

这个例子使用 **initial** 语句对 memory 存储器进行初始化，将其所有的存储单元的初始值都置为“0”。

例 6：编写二与门逻辑电路的顶层模块。

用 Modelsim 对 Verilog 进行仿真时，需要有一个测试程序，测试程序也称为验证程序、顶层模块。下面就是一个二与门逻辑电路的仿真测试程序。

```
`timescale                          //仿真单位和精度设为 1ns
module and_2_test;                  //测试底层模块
reg     A, B;                       //输入定义为寄存器型
wire    F;                          //输出定义为线网型
and_2   and_2 (A, B, F);            //调用底层模块
initial                             //输入信号波形的描述
  begin
        A=0; B=0;
  #100  A=1;
  #100  A=0; B=1;
  #100  A=1;
  #200  $ finish;
  end
endmodule
```

在顶层模块第 5 行 and_2 and_2 (A, B, F)语句中，第一个 and_2 为底层模块名，第二个 and_2 为顶层模块实例名，而(A, B, F)为参数定义。即在 Verilog-HDL 中，调用底层模块的语法结构为：

底层模块名　实例名　参数定义

需要注意，在顶层模块中，所调用的底层模块的模块名要与原程序的模块名一致。顶层模块实例名可以自由起名，在此使用了相同的名字 and_2。

顶层模块的“**initial**”以下顺序描述了波形的变化情况，所描述的内容在“**begin**”和“end”之间。

二、always 过程语句

always 过程语句通常是带有触发条件的，触发条件写在敏感信号表达式中，只有当触发条件满足时，则执行过程块一次；如果不断满足则不断地循环执行。

always 语句的格式如下：

always@(敏感信号表达式)

格式中的敏感信号表达式又称事件表达式，即当该表达式中变量的值改变时，就会引发块内语句的执行。因此，敏感信号表达式中应列出影响块内取值的所有信号。若有两

个或两个以上信号，它们之间用“**or**”连接。

敏感信号可以分为两种类型：一种为边沿敏感型（关键字 posedge 表示上升沿，negedge 表示下降沿），一种为电平敏感型。每一个 always 过程最好只由一种类型的敏感信号来触发，而不要将边沿敏感型和电平敏感型信号列在一起。

例如：

```
@(posedge clock)                    //当 clock 的上升沿到来时
@(negedge clock)                    //当 clock 的下降沿到来时
@(posedge  clk or negedge clr)   //当 clk 上升沿或 clr 下降沿到来时
```

例 7：编写同步 **D** 触发器程序。

同步 D 触发器的逻辑符号如图 6-3 所示。图中 D 为数据输入端，clk 为时钟端，Q 为输出端。这种触发器的逻辑功能是：不论触发器原来的状态如何，输入端的数据 D（无论 D=0还是 D=1）将由时钟 clk 的上升沿将其送入触发器，使得 Q=D。用 Verilog HDL 描述如下：

```
module     D_FF( D,clk,Q,QB);
input      D, clk;
output     Q, QB;        //Q 为输出端，QB 为反相输出端
reg        Q;
assign     QB=~Q;
always     @(posedge clk)
           Q<=D;
endmodule
```

例 8：编写带复位功能的同步 **D** 触发器程序。

带复位功能的同步 D 触发器的逻辑符号如图 6-4 所示。在初始状态，电路的逻辑处于不定状态，当复位脉冲（低电平复位）到来后，将电路初始化为 Q=0，随后，在 D 的控制下，电路的状态作相应的翻转。用 Verilog HDL 描述如下：

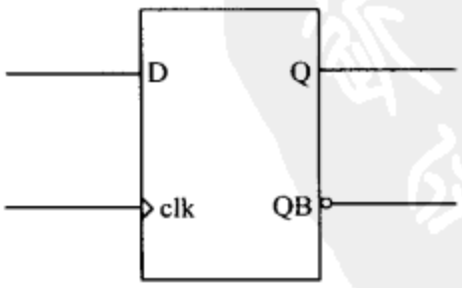


图 6-3　同步 D 触发器逻辑符号

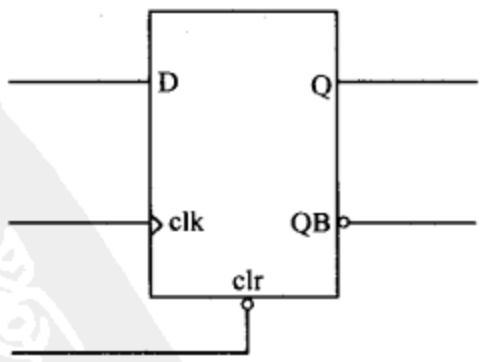


图 6-4　带复位功能的同步 D 触发器逻辑符号

```
module     D_FF_R( clr, D, clk, Q, QB );
input      clr, D, clk;
output     Q, QB ;
reg        Q;
assign     QB=~Q;
```

```
always      @(posedge clk or negedge clr)
    Q <= (! clr )? 0: D;          //clr 为低电平,Q 为 0,clr 为高电平,Q 为 D
endmodule
```

第四节　条件语句

条件语句有 **if** 条件语句和 **case** 条件语句两种,它们都是顺序语句,下面对这两种语句分别进行介绍。

一、if 条件语句

1. if 条件语句的格式

if 条件语句又被称为分支语句,其关键字是由 **if** 构成。Verilog HDL 提供了 3 种形式的 **if** 条件语句。

(1)**if-else** 语句

if-else 语句格式如下:

```
if (表达式)语句 1;
else 语句 2;
```

“语句 1”和“语句 2”表示当条件成立时和不成立分别要执行语句。用一句表达,就是:如果条件成立(也称条件为真),那么程序执行语句 1,否则(条件为假),程序执行语句 2。这就是 **if...else** 语句。

(2)**if** 语句

if-else 中的 **else**(否则)并不是必须的,因为有时候,当指定条件成立时,我们执行某些动作,否则,不执行那些动作。

if 语句格式如下:

```
if (表达式) 语句;
```

该语句的执行过程是:如果条件成立(条件为真),那么程序执行其后的 **if** 语句,然后执行 **if** 语句的下一条语句,如果条件不成立(条件为假),则跳过 **if** 语句,直接执行 **if** 语句的下一条语句。

(3)嵌套的 **if-else** 语句

嵌套的 **if-else** 语句格式如下:

```
if(表达式 1)语句 1;
else if(表达式 2)语句 2;
else if(表达式 3)语句 3;
……
else if(表达式 n)语句 n;
else  语句 n+1;
```

以上形式的嵌套 **if** 语句执行过程可以这样理解:从上向下逐一对 **if** 后的表达式进行

检测，当检测某一表达式的值为真时，就执行与此有关的语句。如果所有表达式的值均为零，则执行最后的 **else** 语句。

2. if 条件语句使用说明

(1)上述 3 种 **if** 条件语句中，“表达式”一般为逻辑表达式或关系表达式，也可能是 1 位的变量。系统对表达式的值进行判断，若为 0、x、z，按“假”处理；若为 1，按“真”处理，执行指定语句。

(2)表达式允许用简写方式。例如：

if(a)等同于 **if**(a==1)

if(! a)等同于 **if**(a! =1)

(3)语句可以是单句，也可以是多句，多句时用“**begin-end** 块语句括起来。对于嵌套的 **if** 语句，若不清楚 **if** 和 **else** 的匹配，最好用“**begin-end**”语句括起来，在这里，**begin-end** 相当于 C 语言中的“{}”。

例如：

```
if(a>b)
  begin
  outl<=intl;
  out2<=in2;
  end
else
  begin
   outl<=int2;
   out2<=intl;
  end
```

例 9：编写同步 T 触发器程序。

同步 T 触发器的逻辑符号如图 6-5 所示，与异步 T 触发器相比，多了一个时钟端。其逻辑功能为：当时钟 clk 到来时，如果 T=1，则触发器翻转；如果 T=0，则触发器的状态保持不变。clr 为复位端(高电平复位)，当 clr 为高电平时，输出与输入的时钟无关，即：Q=0。

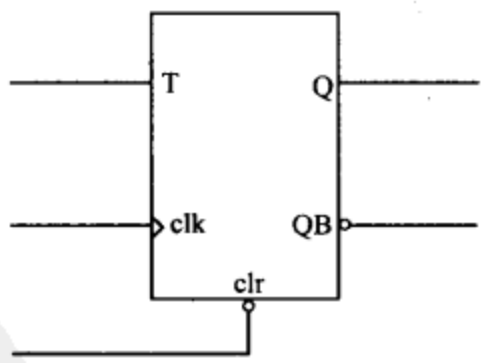


图 6-5　同步 T 触发器逻辑符号

根据同步 T 触发器的逻辑功能，编写的程序如下：

```
module  T_FF(clr, T, clk, Q, QB);
input   clr, T, clk;
output  Q, QB;
reg   Q;
assign  QB=~Q;
always@(posedge clk or posedge clr)
        if  (clr)
```

```
        Q <= 0;
    else if （T）
        Q <= ~Q;
endmodule
```

二、case 条件语句

前面我们学习了 **if** 语句，用多个 **if** 语句可以实现多方向条件分支，但是可以发现，使用过多的 **if** 语句实现多方向分支会使条件语句嵌套过多，程序冗长，不便于阅读。这时，使用 **case** 语句，不但可以达到处理多分支选择的目的，而且又可以使程序结构清晰。

1. case 语句的格式

case 语句有 **case**、**casez**、**casex** 三种表示方式，下面分别进行介绍。

(1)**case** 语句

case 语句的使用格式如下：

```
case(控制表达式)
  值 1:  语句 1;          //case 分支项
  值 2:  语句 2;
  ……
  值 n:  语句 n;          //值 1、值 2……也称为分支表过式
  default:语句 n + 1;
endcase
```

当控制表达式的值为值 1 时，执行语句 l；为值 2 时，执行语句 2；依此类推。如果表达式的值与上面列出的值都不相同的话，则执行 **default** 后面的语句。

(2)**casez** 与 **casex** 语句

在 **case** 语句中，表达式与值 1～值 n 间的比较是一种全等比较，必须保证两者的对应位全等。**casez** 与 **casex** 语句是 **case** 语句的两种变体。

case、**casez** 和 **casex** 语句的真值表如表 6-1 所示。

表 6-1　**case**、**casez** 和 **casex** 真值表

case	0	1	x	z	casez	0	1	x	z	casex	0	1	x	z
0	1	0	0	0	0	1	0	0	1	0	1	0	1	1
1	0	1	0	0	1	0	1	0	1	1	0	1	1	1
x	0	0	1	0	x	0	0	1	1	x	1	1	1	1
z	0	0	0	1	z	1	1	1	1	z	1	1	1	1

例如：

```
case(a)
2 'blx:out=1;          //只有 a=1x，才有 out=1
casex(a)
2 'blx:out=1;          //如果 a=10、l1、1x、1z 等，都有 out=1
```

```
casez(a)
3 'b1??:out=1;          //如果 a=100、101、110、111 或 1xx、1zz 等,都有 out=1
```

2. case 语句使用说明

(1)**default** 项可有可无。一个 **case** 语句里只准有一个 **default** 项。如果已列出了表达式所有可能的取值,则 **default** 语句可以省略。

(2)每一个 **case** 分支项的分支表达式的值必须互不相同;否则,就会出现矛盾现象(对表达式的同一个值,有多种执行方案)。

(3)执行完 **case** 分支项后的语句,则跳出该 **case** 语句结构,终止 **case** 语句的执行。

(4)在用 **case** 语句表达式进行比较的过程中,只有当信号对应位的值能明确进行比较时,比较才能成功,因此要注意详细说明 **case** 分项的分支表达式的值。

(5)**case** 语句的所有表达式的值的位宽必须相等,只有这样,控制表达式和分支值 1、值 2……才能进行对应位的比较。一个经常犯的错误是,用 bx、bz 来替代 n 'bx、n 'bz,因为信号 x 和 z 的默认宽度是机器的字节宽度,通常是 32 位(此处 n 是 **case** 控制表达式的位宽),所以这样写是不对的。

(6)**case** 语句与 **if** 语句相比,主要优点有两点,一是与 **case** 语句处理多分支选择语句时程序结构清晰。二是对于那些分支表达式中存在不定值 x 和高阻值 z 位的情况,**case** 语句提供了处理这种情况的手段。

注意事项 在使用条件语句时,应注意列出所有条件分支,否则,编译器认为条件不满足时,会引进一个触发器保持原值。这一点可用于设计时序电路,例如在计数器的设计中,条件满足则加 1,否则保持不变;而在组合电路设计中,应避免这种隐含触发器的存在。当然,一般不可能列出所有分支,因为每一变量至少有 4 种取值 0、1、z、x。为包含所有分支,可在 **if** 语句最后加上 **else**;在 **case** 语句的最后加上 **default** 语句。

下面是一个隐含锁存器的例子。

```
module sou(c,b,a);
output c;
input b,a;
reg c;
always@(a or b)
  begin
    if((b==1)&&(a==1)
    c=a&b;
  end
endmodule
```

设计者原意是设计一个二输入与门,但因 **if** 语句中无 **else** 语句,在对此语句逻辑综合时会认为 **else** 语句为:"c=c;",即保持不变,所以形成了一个隐含锁存器。

在对上例进行仿真时,只要出现 a=1 及 b=1,则 c=1,然后,c 的值一直都维持为 1。为改正此错误,可将"**always**"过程块变为:

```
always@(a Or b)
```

```
begin
 if((b==1)&&(a==1) c=a&b;
 else c=0
end
```

第五节　循环语句

在许多实际问题中，需要进行具有规律性的重复操作，此时需要采用循环语句。在Verilog HDL中，用来实现循环的语句有以下4种。

forever：连续地执行语句；多用在“**initial**”块中，以生成时钟等周期性波形；

repeat：连续执行一条语句n次。

while：执行循环体语句，直到某个条件不满足。

for：有条件的循环语句。

一、forever语句

forever语句的格式如下：

```
forever      语句；
```

或

```
forever
  begin
  多条语句；
  end
```

forever循环语句连续不断地执行后面的语句或语句块，常用来产生周期性的波形，作为仿真测试信号。**forever**语句与**always**语句的不同之处在于，不能独立写在程序中，而必须写在**initial**过程语句中。

二、repeat语句

repeat语句用来连续执行一条语句n次，循环次数由循环表达式进行控制。**repeat**语句的格式如下：

```
repeat(循环次数表达式)  语句；
```

或

```
repeat(循环次数表达式)
  begin
  多条语句；
  end
```

三、while 语句

while 语句用来执行循环体语句，直到某个条件不满足。**while** 语句的格式如下：

while(条件表达式) 语句；

或

```
while(条件表达式)
  begin
  多条语句;
  end
```

while 语句后一对圆括号中的表达式用来控制循环体是否执行。在语法上，要求循环体可以是一条简单可执行语句；若循环体内需要多个语句，应该用 **begin-end** 括起来。

例如：

```
initial
    begin
      i=0;
      while(i<5)
        begin
          i=i+1;
          out=out+1;
        end
    end
```

while 语句的执行过程如下：

(1)计算 **while** 后一对圆括号中条件表达式的值。当值为非零时，执行步骤(2)；当值为零时，执行步骤(4)。

(2)执行循环体中语句。

(3)转去执行步骤(1)。

(4)退出 **while** 循环。

由以上叙述可知，**while** 后一对圆括号中表达式的值决定了循环体是否执行，因此，进入 **while** 循环后，一定要有能使此表达式的值变为 0 的操作，否则，循环将会无限制地进行下去。

注意事项 不要把由 **if** 语句构成的选择结构与由 **while** 语句构成的循环结构混同起来。若 **if** 后条件表达式的值为非零时，其后的 **if** 子句只执行一次；而 **while** 后条件表达式的值为非零时，其后的循环体中的语句将重复执行，而且在设计循环时，通常应在循环体内改变条件表达式中有关变量的值，使条件表达式的值最终变成 0，以便能及时退出循环。

四、for 语句

for 语句构成的循环结构通常称为 **for** 循环。**for** 循环的一般形式为：

for(表达式 1;表达式 2;表达式 3) 语句;

for 语句后的一对圆括号中通常含有 3 个表达式,各表达式之间用“;”隔开。这 3 个表达式可以是任意形式的表达式,通常主要用于 **for** 循环的控制,紧跟在 **for**(…)之后的循环体。

for 循环的执行过程如下:

(1)计算“表达式 1”(“表达式 1”通常称为“初值设定表达式”)。

(2)计算“表达式 2”(“表达式 2”通常称为“终值条件表达式”);若其值为非零,转步骤(3);若其值为零,转步骤(5)。

(3)执行一次 **for** 循环体。

(4)计算“表达式 3”(“表达式 3”通常称为“更新表达式”);转向步骤(2)。

(5)结束循环,执行 **for** 循环之后的语句。

for 循环语句实际上相当于采用 **while** 循环语句建立以下的循环结构:

```
begin
  循环变量赋初值;
  while(循环终止条件)
  begin
   执行语句
   循环变量更新;
  end
end
```

这样,对于需要 8 条语句才能完成的一个循环控制,而 **for** 循环语句只需 2 条即可。

例 10:设计一个 7 人表决器,若超过四人赞成则通过。

用 key[6:0]表示 7 人的投票情况,1 代表赞成,即 key[i]=1 代表第 i 个人赞成,pass=1表示表决通过。设计的程序如下:

```
module key_7(pass,key);
output pass;
input[6:0]  key;
reg[2:0]  sum;                    //统计赞成的人数
integer i;
reg pass;
always@(key)
   begin
   sum=0;
   for(i=0;i<=6;i=i+1)
   if(key[i])      sum=sum+1;
   if(sum[2]) pass=1       //若超过 4 人赞成,则 pass=1
   else      pass=0;
   end
```

endmodule

重点提示 这里 sum 位宽为 3 位，第 1 位为 sum[0]、第 2 位为 sum[1]，第 3 位为 sum[2]。若赞成人数为 4 人或 4 人以上，则可能是 100、101、110、111 等几种情况，这几种情况中，sum[2]均为 1(最高位表示 sum[2])。因此，用 sum[2]=1 可作为是否超过 4 人的判断条件。

第六节 编译向导语句

Vedlog HDL 和 C 语言一样也提供了编译向导功能。Vefilog HDL 允许在程序中使用特殊的编译向导语句。编译时，通常先对这些向导语句进行预处理，然后再将预处理的结果和源程序一起进行编译。

向导语句以符号“`”开头，以区别于其他语句。Venlog HDL 提供了十几条编译向导语句，比较常用的有**`define**、**`include** 和**`ifdef**、**`else**、**`endif**、**`timescale** 等，下面就这些常用语句进行说明。

一、宏替换`define

`define 语句用于将一个简单的名字或标志符(或称为宏名)来代替一个复杂的名字或字符串，其使用格式为：

`define 宏名(标志符) 字符串

例如：

`define sum ina+inb+inc+ind

在上面的语句中，用简单的宏名 sum 来代替了一个复杂的表达式 ina+inb+inc+ind，采用了这样的定义形式后，在后面的程序中就可以直接用 sum 来代表表达式 ina+inb+inc+ind 了。

比如程序中若出现：

assign out=`sum+ine;

它实际上与 **assign** out=ina+inb+inc+ind+ine;是完全等价的。

再比如：

`define size 7
reg[`size:0] addr;

它与 **reg**[7:0] addr;是完全等价的。

注意事项

(1) **`define** 用于将一个简单的宏名来代替一个字符串或一个复杂的表达式。宏定义语句行末不加分号。

(2)在引用已定义的宏名时，不要忘了在宏名的前面加上符号“`”，以表示该名字是一个宏定义的名字。

(3)采用宏定义，可以简化程序的书写，也便于修改，若需要改变某个变量，只需改变`define定义行，一改全改。

二、文件包含`include

`include是文件包含语句，它可将一个文件全部包含到另一个文件中。其格式为：

`include　“文件名”

文件包含命令是很有用的，它可以节省程序设计人员的重复劳动，可以将一些常用的宏定义命令或任务组成一个文件，然后用`include命令将这些宏定义包含到自己所写的源文件中，相当于工业上的标准元件拿来使用。另外在编写 Verilog HDL 源文件时，一个源文件可能经常要用到另外几个源文件中的模块，遇到这种情况即可用`include命令将所需模块的源文件包含进来。

下面是一个 8 位加法器的例子。在该例中，adder8 模块使用`include语句调用了一个通用的加法器 adder 模块。或者说，adder8 模块包含了 adder 模块。

例 11：使用`include语句的 8 位加法器。

```
`include "adder. v"
module adder8(cout,sum,a,b,cin);
output cout;
output[7:0]  sum;
input[7:0]  a,b;
input cin;
adder my_adder(cout,sum,a,b,cin);      //调用 adder 模块
endmodule
//下面是 adder 模块代码
module  adder(cout,sum,a,b,cinl);
output  cout;
output[7:0]  sum;
input  cin;
input[7:0]  a,b;
assign  {cout,sum}=a+b+cin;
endmodule
```

注意事项　一个`include语句只能指定一个被包含的文件。`include语句可以出现在源程序的任何地方。被包含的文件若与包含文件不在同一个子目录下，必须指明其路径名。文件包含允许多重包含，比如文件 1 包含文件 2，文件 2 又包含文件 3 等。

三、条件编译`ifdef、`else、`endif

一般情况下，Verilog HDL 源程序中所有的行都将参加编译，但有时希望对其中的一部分内容只有在满足条件才进行编译，也就是对一部分内容指定编译的条件，这就是条件编译。有时，希望当满足条件时对一组语句进行编译，而当条件不满足时则编译另一部

分。条件编译命令有以下 2 种形式。

第一种格式：

`ifdef 宏名(标识符)
程序段 1
`else
程序段 2
`endif

它的作用是若宏名已经被定义过(用`define 命令定义)，则对程序段 1 进行编译，程序段 2 将被忽略；否则编译程序段 2，程序段 1 被忽略。

形式 2 没有`else 部分，其形式如下所示：

`ifdef 宏名(标识符)
程序段 1
`endif

这里的宏名是一个 Verilog HDL 的标识符，程序段可以是 Verilog HDL 语句组，也可以是命令行，这些命令可以出现的程序的任何地方。

四、时间尺度`timescale

`timescale 命令用来说明跟在该命令后的模块的时间单位和时间精度。

`timescale 命令的格式如下：

`timescale<时间单位>/<时间精度>

在这条命令中，时间单位参量是用来定义模块中仿真时间和延迟时间的基准单位的。时间精度参量是用来声明该模块的仿真时间的精确程度的，该参量被用来对延迟时间值进行取整操作(仿真前)，因此该参量又可以被称为取整精度。如果在同一个程序设计里，存在多个`timescale 命令，则用最小的时间精度值来决定仿真的时间单位。另外时间精度至少要和时间单位一样精确，时间精度值不能大于时间单位值。

在`timescale 命令中，用于说明时间单位和时间精度参量值的数字必须是整数. 其有最有效数字为 1，10，100，单位为 s，ms，μs，ns，ps，fs，分别表示 1s，10^{-3} s，10^{-6} s，10^{-9} s，10^{-12} s，10^{-15} s。

例如：`timescale1ns/1ps 表示的含义如下：这个命令之后，模块中所有的时间值都表示是 1ns 的整数倍。这是因为在`timescale 命令中，定义了时间单位为 1ns。模块中的时间精度为 1ps。

例 12：说明以下程序的含义。

```
`timescale 10ns/1ns
module  test;
reg  set
parameter  d=1.55;
initial
```

```
    begin
        #d set=0;
        #d set=1;
    end
endmodule
```

在这个例子中，**timescale** 命令定义了模块 test 的时间单位为 10ns、时间精度为 1ns。因此，在模块 test 中，所有的时间值应为 10ns 的整数倍，且以 1ns 为时间精度。这样经过取整操作，存在参数 d 中的延迟时间实际是 16ns(即 1.6×10ns)，这意味着在仿真时刻为 16ns 时寄存器 set 被赋值为 0，在仿真时刻为 32ns 时寄存器 set 被赋值为 1。仿真时刻值是按照以下的步骤来计算的。

(1)根据时间精度，参数 d 值被从 1.55 取整为 1.6。

(2)因为时间单位是 10ns，时间精度是 1ns，所以延迟时间 #d 为时间单位的整数倍，即为 16ns。

(3)仿真工具在仿真时刻为 16ns 的时候给寄存器 set 赋值为 0，在仿真时刻为 32ns 的时候给寄存器 set 赋值为 1。

第七节　任务(task)和函数(function)说明语句

task 和 **function** 说明语句分别用来定义任务和函数。利用任务和函数可以把一个很大的程序模块分解成许多较小的任务和函数，这样便于理解和调试。输入、输出和总线信号的值可以传入、传出任务和函数。任务和函数往往还是大的程序模块中在不同地点多次用到的相同的程序段。学会使用 **task** 和 **function** 语句可以简化程序的结构，使程序明白易懂，是编写较大模块的基本功。

一、任务(task)说明语句

任务(task)的定义格式如下。

定义：**task**<任务名>；

端口及数据类型声明语句；

其他语句；

endtask

任务调用的格式如下：

<任务名>(端口 1，端口 2，…，端口 n)；

需要说明的是，任务调用变量和定义时说明的 I/O 变量是一一对应的。下面举例说明。

例如：以下定义了一个任务名为 my_task 的任务。

```
task   my_task;
input a,b;
output F1,F2;
```

```
F1=a&b;
F2=a|b;
endtask
```

当调用该任务时，可使用如下语句：

```
my_task(u,v,x,y);
```

调用任务 test 时，由 u、v 传入的变量赋给 a 和 b，而任务执行完后，F1 和 F2 的值则赋给了 x 和 y。

例 13：编写交通灯控制程序。

程序如下：

```
module traffic_lights;
reg clock,red,amber,green;
parameter  on=1,off=0,red_tics=350,amber_tics=30,green_tics=200
initial    red=off;
initial    amber=off;
initial    green=off;                    //交通灯控制时序
  always
  begin
      red=on;                           //开红灯
      light(red,red_tics);              //调用等待任务
      green=on;                         //开绿灯
      light(green,green_ties)           //等待
      amber=on;                         //开黄灯
      light(amber,amber_tics)           //等待
    end                                 //定义交通灯开启时间的任务
task  light
    output  color;
    input[31:0] tics;
    begin
      repeat(tics)
      @(posedge clock);                 //等待 tics 个时钟的上升沿
      color=off;                        //关灯
    end
endtask                                 //产生时钟脉冲的 always 块
  always
    begin
      #100 clock=0;
      #100 clock=1;
    end
```

```
endmodule
```

这个例子描述了一个简单的交通灯的时序控制，并且该交通灯有它自己的时钟发生器。

二、函数(function)说明语句

1. 函数的定义与调用

函数的定义格式如下：

```
function   <返回值位宽或类型说明>，函数名；
端口声明；
局部变量定义；
其他语句；
endfunction
```

<返回值位宽或类型说明>是一个可选项，如果默认，则返回值为一位寄存器类型的数据。

在函数定义时，将函数返回值所使用的寄存器名称设为与函数同名的内部变量，因此函数名被赋予的值就是函数的返回值。

函数的调用是通过将函数作为表达式中的操作数来实现的，调用格式如下：

```
函数名(<表达式> <表达式>)；
```

例 14：编写二选一数据选择器程序。

二选一数据选择器有两个输入端 A、B，一个输出端 F，一个选择信号 SEL，其逻辑功能是：若 SEL 为高电平，则输出端 F 输出 B 信号，若 SEL 为低电平，则输出端 F 输出来自 A 的信号。用 Verilog 编写的程序如下：

```
module     SEL2_1( A, B, SEL, F );
input      A, B, SEL;
output     F;
function     SEL2_1_FUN;
  input      A, B, SEL;
  if ( SEL == 0 )  SEL2_1_FUN= A;
  else  SEL2_1_FUN= B;
endfunction
assign     F= SEL2_1_FUN( A, B, SEL );
endmodule
```

注意事项

(1)函数的定义与调用须在一个 **module** 模块内。

(2)函数只允许有输入变量，且必须至少有一个输入变量，输出变量由由函数名本身担任，比如，在上面的例子中，函数名 SEL2_1_FUN 即是输出变量。

(3)定义函数时，没有端口名列表，如上例中定义的函数为 SEL2_1_FUN 没有端口列表，但调用函数时，需列出端口名列表，如 **assign** F= SEL2_1_FUN(A, B, SEL)；端口

名的排序和类型必须与定义时的相一致，这一点与任务相同。

(4)函数可以出现在持续赋值 **assign** 的右端表达式中。

2. 函数与任务的区别

函数与任务不同之处主要有以下几点：

(1)函数只能与主模块共用同一个仿真时间单位，而任务可以定义自己的仿真时间单位。

(2)函数不能调用任务，而任务可以调用别的任务和函数，且调用任务和函数个数不受限制。

(3)函数至少要有一个输入变量，而任务可以没有或有多个任何类型的变量。

(4)函数返回一个值，而任务则不返回值。函数的目的是通过返回一个值来响应输入信号的值。任务却能支持多种目的，能计算多个结果值，这些结果值只能通过被调用的任务的输出或总线端口送出。Verilog HDL 模块使用函数时是把它当作表达式中的操作符，这个操作的结果值就是这个函数的返回值。

第八节　系统任务与系统函数

Verilog HDL 的系统任务和系统函数主要用于仿真，比如实时显示当前仿真时间(**$ time**)、显示信号的值(**$ display**、**$ monitor**)、暂停仿真(**$ stop**)、结束仿真(**$ finish**)。使用系统任务和系统函数，可以显示模拟结果，对文件进行操作，以及控制模拟的执行过程等。

系统任务和系统函数一般以符号“$”开头，使用不同的 Verilog HDL 仿真工具(如 VCS、Verilog-XL、ModelSim 等)进行仿真时，这些系统任务和系统函数在使用方法上可能存在差异，应根据使用手册来使用。一般在 **initial** 或 **always** 过程块中调用系统任务和系统函数。用户可以通过编程语言接口将自己定义的系统任务和系统函数加到语言中，以方便仿真和调试。

由于系统任务和系统函数比较多，有的开发工具还有自己定义的系统任务和函数，因此，下面只介绍常用的系统任务和系统函数，这些任务和函数大多数仿真工具都支持，且基本能够满足一般的仿真测试的需要。

一、$ display 和 $ write 任务

$ display 和 **$ write** 是两个系统任务，两者的功能相同，都用于显示模拟结果，其区别是 **$ display** 在输出结束后能自动换行，而 **$ write** 不能。

$ display 和 **$ write** 的使用格式为：

$ display("格式控制符",输出变量名列表);

$ write("格式控制符",输出变量名列表);

格式控制符如表 6-2 所示。

例如：

```
$ display  ( $ time, , ,  "a=%h b=%h c=%h",a,b,c);
```

表 6-2　格式控制符

输出格式	说　明	输出格式	说　明
%h 或%H	以十六进制输出	%m 或%M	输出等级层次的名称
%d 或%D	以十进制输出	%s 或%S	以字符串的形式输出
%o 或%O	以八进制输出	%t 或%T	以当前的时间格式输出
%b 或%B	以二进制输出	%e 或%E	以指数的形式输出实型数
%c 或%C	以 ASCII 码字符输出	%f 或%F	以十进制的形式输出实型数
%v 或%V	输出网络型信号强度		

上面的语句定义了信号显示的格式，即以十六进制格式显示信号 a、b、c 的值，两个相邻的逗号“,,”表示加入一个空格。

也可以用 **$ display** 显示字符串，比如：

$ display("Verilog\n");

上面的语句将直接输出引号中的字符串，而“\n”是转义字符，表示换行。表 6-3 中列举了常用的转义字符，这些转义字符也用于输出格式的定义。

表 6-3　常用转义字符

转义字符	说　明	转义字符	说　明
\n	换行	\"	双引号字符
\t	跳到下一输出区	\%	百分符号
\\	反斜杠字符	\o	(1～3)位八进制代表的字符

二、$ monitor 与 $ strobe

$ monitor、**$ strobe** 与 **$ display**、**$ write** 一样，也属于输出控制类的系统任务，**$ monitor** 与 $ **strobe** 都提供了监控和输出参数列表中字符或变量的值的功能，其使用格式为：

$ monitor("格式控制符",输出变量名列表);
$ strobe("格式控制符",输出变量名列表);

这里的格式控制符、输出变量名列表等与 **$ display**、**$ write** 中定义的完全相同。

例如：**$ monitor**(**$ time**,,,"a=%d b=%d c=%d"a,b,c);

每次 a、b、c 信号的值发生变化都会激活上面的语句，并显示当前仿真时间十进制格式的 a、b、c 信号。

$ strobe 相当于是选通监控器，只有在模拟时间发生改变时，并且所有的件都已处理完毕后，**$ strobe** 才将结果输出。 **$ strobe** 多用来显示用非阻塞方式赋值的变量。

三、$ time 与 $ realtime

$ time、**$ realtime** 是属于显示仿真时间标度的系统函数。这两个函数被调用时，都返回前时刻距离仿真开始时刻的时间量值，所不同的是，**$ time** 函数以 64 位整数值的形式返回模拟时间，**$ realtime** 函数则以实数型数据返回模拟时间。

下面举例说明。

例 15：$ time 函数应用

```
`timescale   10 ns/1ns
module   test;
reg   set;
parameter   P=1.6;
initial
 begin
     $ monitor( $ time,,"set=",set);
     #p set=0;
     #p set=1;
 end
endmodule
```

输出结果为：

```
0 set=x
2 set=0
3 set=1
```

在这个例子中，模块 test 想在时刻为 16ns 时将寄存器 set 设置为 0，在时刻为 32ns 时将寄存器设置 set 为 1。但是由 **$ time** 记录的 set 变化时刻却和预想的不一样。这是由下面两个原因引起的：

(1) **$ time** 显示时刻受时间尺度比例的影响。在该例中，时间尺度是 10ns，因为 **$ time** 输出的时刻总是时间尺度的倍数，这样将 16ns 和 32ns 输出为 1.6 和 3.2。

(2)因为 **$ time** 总是输出整数，所以在输出经过尺度比例变换的数字输出时，要先进行取整。在该例中，1.6 和 3.2 经取整后为 2 和 3 输出。在这里，时间的精确度并不影响数字的取整。

例 16：$ realtime 函数应用

```
`timescale   10 ns/1ns
module   test;
reg   set;
parameter   P=1.55;
initial
  begin
     $ monitor( $ realtime,,"set=",set);
     #p set=0;
     #p set=1;
  end
endmodule
```

输出结果为：

```
0      set＝x
1.6    set＝0
3.2    set＝1
```

从这个例子中可以看出，$ realtime 将仿真时刻经过尺度变换以后即输出，不需进行取整操作，所以，$ realtime 返回的时刻是实型数。

四、$ finish 与 $ stop

系统任务 **$ finish** 与 **$ stop** 用于对仿真过程进行控制，分别表示结束仿真和中断仿真。

$ finish 与 **$ stop** 的使用格式如下：

```
$ stop;
$ stop(n);
$ finish;
$ finish(n);
```

n 是 **$ finish** 和 **$ stop** 的参数，n 可以是 0、1、2 等值，分别表示如下含义。

0：不输出任何信息；

1：给出仿真时间和位置；

2：给出仿真时间和位置，还有其他一些运行统计数据。

如果不带参数，则默认的参数值是 1。

当仿真程序执行到 **$ stop** 语句时，将暂时停止仿真，此时设计者可以输入命令，对仿真器进行交互控制。而当仿真程序执行到 **$ finish** 语句时，则终止仿真，结束整个的仿真过程，返回主操作系统。

第七章　Verilog HDL 的描述方式

Verilog HDL 是一种用于数字逻辑电路设计的语言。用 Verilog HDL 描述的电路就是该电路的 Verilog HDL 模型。Verilog HDL 既可以用电路的功能建模，也可以用元器件和它们之间的连接来建模。Verilog HDL 程序主要有结构描述、行为描述和数据流描述三种描述方式。另外，还可以采用上述方式的混合形式来描述设计，本章将对这三种常用的描述方式进行介绍。

第一节　结构描述方式

在 Verilog HDL 程序中可通过如下方式来描述电路的结构：

(1)调用 Verilog 内置门元件(门级结构描述)；

(2)调用开关级元件(开关级结构描述)；

(3)用户自定义元件 UDP(也在门级)。

此外，在多层次结构电路的设计中，不同模块间的调用也可以认为是结构描述。

一、Verilog HDL 内置门元件

Verilog HDL 内置 26 个基本元件，其中 14 个是门级元件，12 个为开关级元件，如表 7-1 所示。

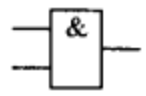
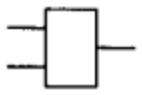
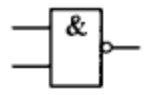
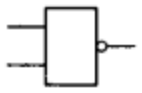
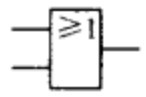
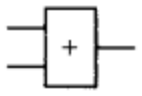
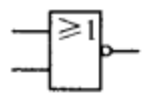
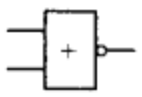
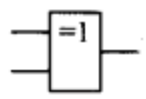
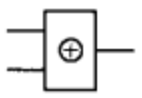
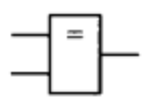
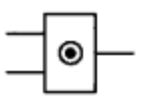
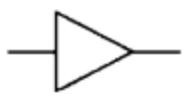
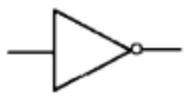
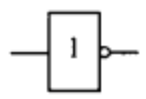
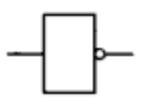
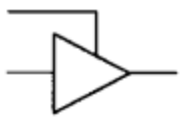
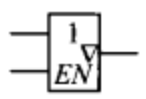
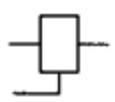
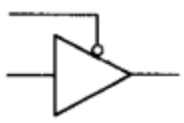
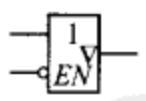
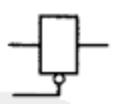
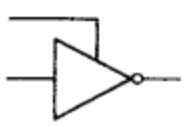
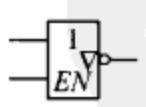
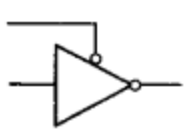
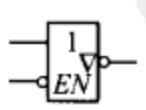
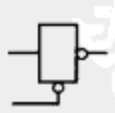
表 7-1　Verilog HDL 内置的 26 个基本元件

类　型		元　件
基本门	多输入门	and nand or nor xor nxor
	多输出门	buf not
三态门	允许定义驱动强度	bufif0 bufif1 notif0 notif1
MOS 开关	无驱动强度	nmos pmos cmos rnmos rpmos rcmos
双向开关	无驱动强度	tran tranif0 tranif1 rtran rtranif0 rtranif1
上拉电阻、下拉电阻	允许定义驱动强度	pullup pulldown

下面重点介绍门级元件和门级结构描述。

Verilog HDL 内置的门元件如表 7-2 所示。

表 7-2　**Verilog HDL** 常用的内置门元件

类　别	关 键 字	国外流行符号	国际符号	曾用符号	门 名 称
多输入门	and				与门
	nand				与非门
	or				或门
	nor				或非门
	xor				异或门
	xnor				异或非门
多输出门	buf				缓冲器
	not				非门
三态门	bufif1				4 种三态门
	buif0				
	notif1				
	notif0				

1. 逻辑门真值表

(1)与门真值表

与门真值表如表 7-3 所示。

(2)与非门真值表

与非门真值表如表 7-4 所示。

表 7-3　与门真值表

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

表 7-4　与非门真值表

nand	0	1	x	z
0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

(3)或门真值表

或门真值表如表 7-5 所示。

(4)或非门真值表

或非门真值表如表 7 6 所示。

表 7-5　或门真值表

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

表 7-6　或非门真值表

nor	0	1	x	z
0	1	0	x	x
1	0	0	0	0
x	x	0	x	x
z	x	0	x	x

(5)异或门真值表

异或门真值表如表 7-7 所示。

(6)异或非门真值表

异或非门真值表如表 7-8 所示。

表 7-7　异或门真值表

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

表 7-8　异或非门真值表

xnor	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

(7)三态门 **bufif1** 真值表

三态门 **bufif1** 为高电平使使能缓冲门，真值表如表 7-9 所示。

(8)三态门 **bufif0** 真值表

三态门 **bufif0** 为低电平使能缓冲门，其真值表如表 7-10 所示。

表 7-9　三态门 bufif1 真值表

bufif1		使能端			
		0	1	x	z
输入端	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	x	x	x

表 7-10　三态门 bufif0 真值表

bufif0		使能端			
		0	1	x	z
输入端	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	x	z	x	x

(9)三态门 **notif1** 真值表

三态门 **notif1** 为高电平使能反相缓冲门,其真值表如表 7-11 所示。

(10)三态门 **notif0** 真值表

三态门 **notif0** 为低电平使能反相缓冲门,真值表如表 7-12 所示。

表 7-11　三态门 **notif1** 真值表

notif1		使能端			
		0	1	x	z
输入端	0	z	1	H	H
	1	z	0	L	L
	x	z	x	x	x
	z	z	x	x	x

表 7-12　三态门 **notif0** 真值表

notif0		使能端			
		0	1	x	z
输入端	0	1	z	H	H
	1	0	z	L	L
	x	x	z	x	x
	z	x	z	x	x

2. 门元件的调用

调用门元件的格式为:

门元件名字<例化的门名字>(<端口列表>)

其中,普通门的端口列表按下面的顺序列出:

(输出,输入 1,输入 2,输入 3,……);

例如:

```
and U1 (out,inl,in2);          //二输入与门,其名字为 U1
```

对于三态门,则按如下顺序列出输入、输出端口:

(输出,输入,使能控制端);

例如:

```
bufif1 U2(out,in,enable);      //高电平使能的三态门
bufif0 U3(out,in,enable);      //低电平使能的三态门
```

对于 **buf** 和 **not** 两种元件的调用,需要注意的是,它们允许有多个输出,但只能有一个输入。

例如:

```
not U4(out1,out2,in);          //1 个输入 in,2 个输出 outl,out2
buf U5(out,in);                //1 个输入 in,1 个输出 out
```

二、门级结构描述

门级描述是指采用 Verilog HDL 所具有的内置门实例语句的描述方法。

例 1:用门级结构描述方式描述二与门电路。

对于二与门电路,若采用门级描述方法,则描述如下:

```
module and_2(A,B,F);           //模块名为 and_2,端口列表 A,B,F
input A,B;                     //模块的输入端口为 A,B
output F;                      //模块的输出端口为 F
```

```
and U1 (F,A,B);                    //门级描述
endmodule
```

例 2:用门级结构描述方式描述二选一数据选择器。

二选一数据选择器有两个输入端 A、B,一个输出端 F,一个选择信号 SEL,其逻辑功能是:若 SEL 为高电平,则输出端 F 输出 B 信号,若 SEL 为低电平,则输出端 F 输出来自 A 的信号。二选一数据选择器的逻辑函数为:F=/SEL·A+SEL·B,其内部电路如图 7-1所示。

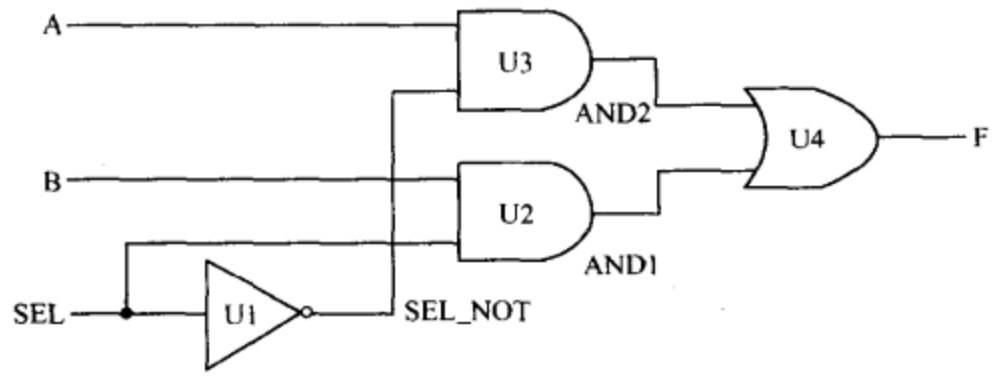


图 7-1　二选一数据选择器内部逻辑组成

根据其内部逻辑组成,用 Verilog 的编写的门级描述方式如下:

```
module  SEL  ( A, B, SEL, F );
input   A, B, SEL;
output  F;
wire    SEL_NOT, AND1, AND2;
not     U1   ( SEL_NOT, SEL );
and     U2   ( AND1, B, SEL ),
         U3     ( AND2, A, SEL_NOT );
or      U4 ( F , AND1, AND2 );
endmodule
```

第二节　数据流描述方式

数据流描述方法是使用持续赋值语句描述数据流程的运动路径、运动方向和运动结果。

例 3:采用数据流描述方式描述二与门电路。

采用数据流描述方式描述的二与门电路如下:

```
module and_2(A,B,F);          //模块名为 and_2,端口列表 A,B,F
input  A,B;                   //模块的输入端口为 A,B
output F;                     //模块的输出端口为 F
assign F=A&B;                 //数据流描述
endmodule
```

例 4:采用数据流描述方式描述二选一数据选择器。

根据二选一数据选择器的逻辑函数 F=/SEL·A+SEL·B,用数据流描述方式描述如下:

```
module  SEL(A,B,SEL,F);
input   A, B, SEL;
output  F ;
assign  F=~SEL&A|SEL&B;
endmodule
```

第三节　行为描述方式

前面介绍的硬件电路的结构描述侧重于表示一个电路由哪些基本元件组成,以及这些基本元件的相互连接关系。而硬件电路的行为描述则主要反映该电路输入、输出信号间的相互关系。行为描述方式一般采用 initial 语句(此语句只执行一次)或 always 语句(此语句总是循环执行,或者说此语句重复执行)来描述逻辑功能。

行为描述相当于软件设计过程中的流程图描述和算法描述,它着重要表达的是工作的抽象或者说是软件功能的行为表现,而不是具体的实现手段和方法。在行为描述的基础上,经过论证是可行的话,才考虑用哪种语言来实现。

对于行为描述方式的数据选择器来说,在描述其输出时只需描述与输入的关系,而无需费力地像门级描述方式那样说明究竟是怎样实现的。

行为描述既可以描述组合逻辑电路,也可以描述时序逻辑电路,但多用于描述时序逻辑。

例 5:采用行为描述方式,描述 **D** 触发器。

对于 D 触发器,用行为描述方式描述如下:

```
module D_FF(Q,D,clk);    //模块名为 D_FF,端口列表 Q,D,clk
input D,clk;              //模块的输入端口为 D,clk
output Q;                 //模块的输出端口为 Q
reg Q;                    //定义 Q 为寄存类型
always@(posedge clk)      //每当 clk 上升沿到来时执行一遍 begin 和 end 块内的
                            语句
begin
Q<=D;
end
endmodule
```

例 6:采用行为描述方式,描述二选一数据选择器。

对于二选一数据选择器,采用行为描述方式描述如下:

```
module  SEL2_1(A,B,SEL,F);
input   A, B, SEL;
```

```
output    F;
function SEL2_1_FUN;
    input   A, B, SEL;
    if  ( SEL == 0)   SEL2_1_FUN= A;
    else   SEL2_1_FUN= B;
endfunction
assign   F= SEL2_1_FUN( A, B, SEL );
endmodule
```

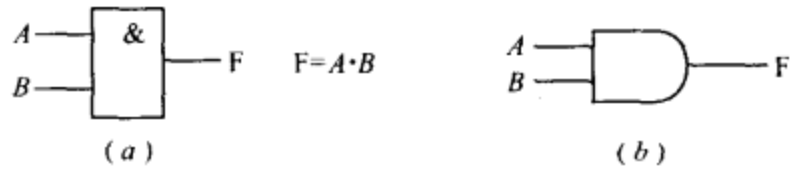
第八章　用 Verilog HDL 描述数字电路

本章介绍基本数字电路的 Verilog HDL 描述，主要包括常用基本门电路、组合逻辑电路、触发器和时序逻辑电路等，这些基本的数字模块通常可以用多种方式来设计，如门级描述、数据流描述或行为描述等，掌握基本数字电路的设计方法可以为更复杂的电路设计提供方便。

第一节　基本门电路的设计

一、与门

以二与门为例，其电路符号如图 8-1 所示。



(a)　　(b)

图 8-1　与门符号

(a) 中国和国际标准符号；(b) 美国符号。

用 Verilog HDL 描述的二与门如下。

1. 门级描述方式

```
module and_gat(A,B,F);          //模块名为 and_gat,端口列表 A,B,F
input  A,B;                     //模块的输入端口为 A,B
output F;                       //模块的输出端口为 F
and U1 (F,A,B);                 //门级描述
endmodule
```

2. 数据流描述方式

```
module and_gat(A,B,F);          //模块名为 and_gat,端口列表 A,B,F
input  A,B;                     //模块的输入端口为 A,B
output F;                       //模块的输出端口为 F
assign F=A&B;                   //数据流描述
endmodule
```

二、或门

以二或门为例，其电路符号如图 8-2 所示。

用 Verilog HDL 描述的二或门如下。

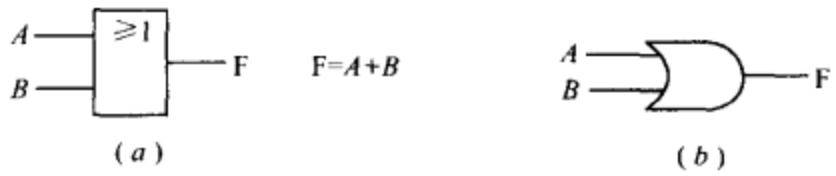


图 8-2　或门符号

(a) 中国和国际标准符号；(b) 美国符号。

1. 门级描述方式

```
module or_gat(A,B,F);          //模块名为 or_gat,端口列表 A,B,F
input  A,B;                    //模块的输入端口为 A,B
output F;                      //模块的输出端口为 F
or U1 (F,A,B);                 //门级描述
endmodule
```

2. 数据流描述方式

```
module or_gat(A,B,F);          //模块名为 or_gat,端口列表 A,B,F
input  A,B;                    //模块的输入端口为 A,B
output F;                      //模块的输出端口为 F
assign F=A|B;                  //数据流描述
endmodule
```

三、非门

非门电路符号如图 8-3 所示。

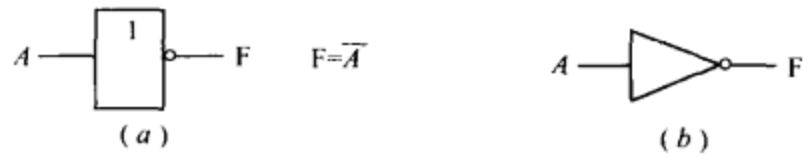


图 8-3　非门符号

(a) 中国和国际标准符号；(b) 美国符号。

用 Verilog HDL 描述的非门如下。

1. 门级描述方式

```
module not_gat(A,F);           //模块名为 not_gat,端口列表 A,F
input  A;                      //模块的输入端口为 A
output F;                      //模块的输出端口为 F
not U1 (F,A);                  //门级描述
endmodule
```

2. 数据流描述方式

```
module not_gat(A,F);           //模块名为 not_gat,端口列表 A,F
input  A;                      //模块的输入端口为 A
output F;                      //模块的输出端口为 F
assign F=~A;                   //数据流描述
endmodule
```

四、与非门

以二与非门为例，其电路符号如图 8-4 所示。

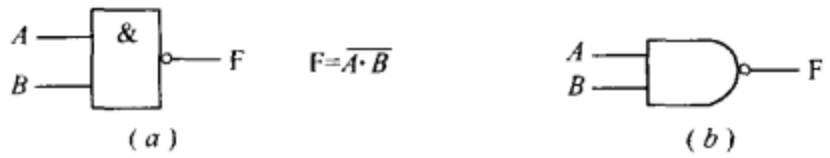


图 8-4　与非门符号

(a) 中国和国际标准符号；(b) 美国符号。

用 Verilog HDL 描述的二与非门如下。

1. 门级描述方式

```
module nand_gat(A,B,F);          //模块名为 nand_gat,端口列表 A,B,F
input A,B;                       //模块的输入端口为 A,B
output F;                        //模块的输出端口为 F
and U1 (F,A,B);                  //门级描述
endmodule
```

2. 数据流描述方式

```
module nand_gat(A,B,F);          //模块名为 nand_gat,端口列表 A,B,F
input A,B;                       //模块的输入端口为 A,B
output F;                        //模块的输出端口为 F
assign F=~(A&B);                 //数据流描述
endmodule
```

五、或非门

以二或非门为例，其电路符号如图 8-5 所示。

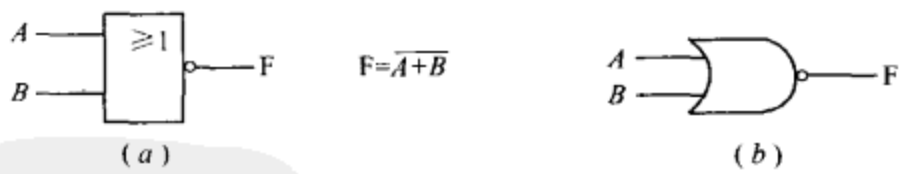


图 8-5　或非门符号

(a) 中国和国际标准符号；(b) 美国符号。

用 Verilog HDL 描述的二或非门如下。

1. 门级描述方式

```
module nor_gat(A,B,F);           //模块名为 nor_gat,端口列表 A,B,F
input A,B;                       //模块的输入端口为 A,B
output F;                        //模块的输出端口为 F
nor U1 (F,A,B);                  //门级描述
endmodule
```

2. 数据流描述方式

```
module nor_gat(A,B,F);           //模块名为 nor_gat,端口列表 A,B,F
```

```
input A,B;                    //模块的输入端口为 A,B
output F;                     //模块的输出端口为 F
assign F=~(A|B);              //数据流描述
endmodule
```

六、异或门

异或门电路符号如图 8-6 所示。

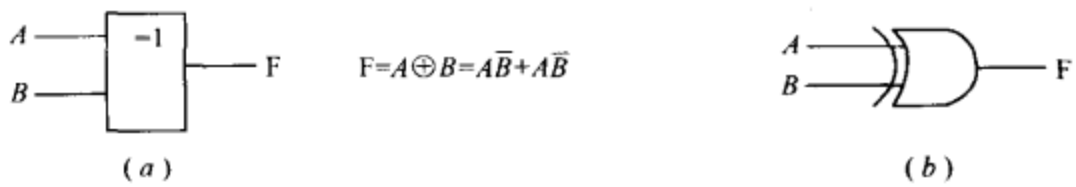


图 8-6　异或门符号

(a) 中国和国际标准符号；(b) 美国符号。

用 Verilog HDL 描述的异或门如下。

1. 门级描述方式

```
module xor_gat(A,B,F);        //模块名为 xor_gat,端口列表 A,B,F
input A,B;                    //模块的输入端口为 A,B
output F;                     //模块的输出端口为 F
xor U1 (F,A,B);               //门级描述
endmodule
```

2. 数据流描述方式

```
module xor_gat(A,B,F);        //模块名为 xor_gat,端口列表 A,B,F
input A,B;                    //模块的输入端口为 A,B
output F;                     //模块的输出端口为 F
assign F=A^B;                 //数据流描述
endmodule
```

七、缓冲门

缓冲门电路符号如图 8-7 所示。

图 8-7　缓冲门符号

用 Verilog HDL 描述的缓冲门如下。

1. 门级描述方式

```
module buf_gat(A,F);          //模块名为 buf_gat,端口列表 A,F
input A;                      //模块的输入端口为 A
output F;                     //模块的输出端口为 F
```

```
buf U1 (F,A);                    //门级描述
endmodule
```

2. 数据流描述方式

```
module buf_gat(A,F);             //模块名为 buf_gat,端口列表 A,F
input  A;                        //模块的输入端口为 A
output F;                        //模块的输出端口为 F
assign F=A;                      //数据流描述
endmodule
```

八、三态门

三态门主要有 **buif0**、**buif1**、**notif0**、**notif1** 等 4 种，其电路符号如图 8-8 所示。下面以 **buif1** 三态门为例进行说明。**buif1** 的真值表如表 8-1 所示。

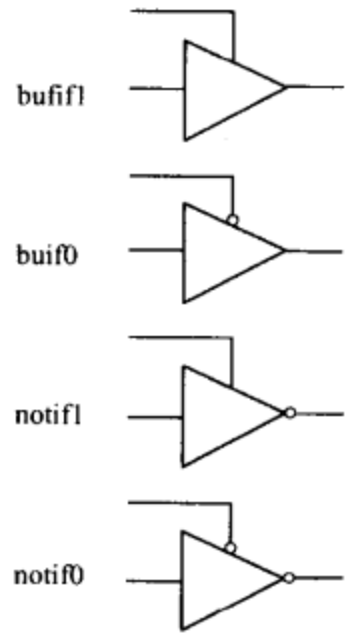


图 8-8　三态门符号

表 8-1　**buif1** 真值表

bufif1		使能端(EN)			
		0	1	x	z
输入端	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	x	x	x

1. 门级描述方式

```
module buf_en(A,EN,F);           //模块名为 buf_en,端口列表 A,EN,F
input  A,EN;                     //模块的输入端口为 A,EN
output F;                        //模块的输出端口为 F
bufif1 U1 (F,A,EN);              //门级描述,注意端口排列顺序
endmodule
```

2. 数据流描述方式

```
module buf_en(A,EN,F);           //模块名为 buf_en,端口列表 A,EN,F
input  A,EN;                     //模块的输入端口为 A,EN
output F;                        //模块的输出端口为 F
assign F=EN? A:'bz;              //若 EN=1 则,F=A,若 EN=0,则 F 为高阻
endmodule
```

第二节　组合逻辑电路的设计

组合逻辑电路的特点是：在任意时刻，电路的输出信号仅仅取决于当时的输入信号，

与电路原来所处的状态无关。组合逻辑电路通常可由若干个基本的逻辑门组成。常用的组合逻辑电路有数据选择器、编码器、译码器、加法器等。

一、数据选择器

数据选择器又称为多路开关，简称 MUX。它的逻辑功能是在地址选择信号 SEL 的控制下，从多路输入(A、B...)数据中选择某一路数据作为输出。

1. 二选一数据选择器

(1)二选一数据选择器分析

二选一数据选择器电路框图如图 8-9 所示，输入端为 A、B，输出端为 F，SEL 为控制端。

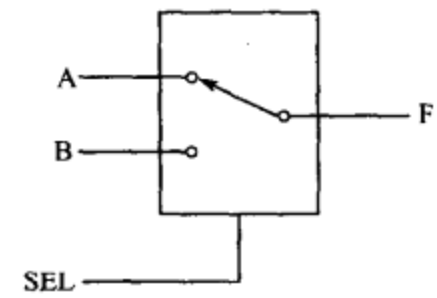


图 8-9　二选一数据选择器框图

二选一数据选择器真值表如表 8-2 所示。

表 8-2　二选一数据选择器真值表

控制信号	输入信号		输出信号	控制信号	输入信号		输出信号
SEL	A	B	F	0	0	0	0
1	0	0	0	0	0	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	1	1
1	1	1	1				

从真值表中可以看出，当 SEL 为 1 时，输出端 F 输出 B 信号，当 SEL 为 0 时，输出端 F 输出 A 信号。

二选一数据选择器的逻辑表达式为：

$$F=/SEL \cdot A+SEL \cdot B$$

(2)用 Verilog HDL 描述二选一数据选择器

采用数据流描述方式，描述如下：

```
module    SEL2_1(A,B,SEL,F);
input     A,B,SEL;
output    F;
assign    F=~SEL&A|SEL&B;
endmodule
```

采用行为描述方式，描述如下：

```
module          SEL2_1(A,B,SEL,F);
input           A, B, SEL;
output          F;
function        SEL2_1_FUN;
  input         A, B, SEL;
  if(SEL==0)    SEL2_1_FUN=A;
```

```
    else          SEL2_1_FUN= B;
endfunction
assign            F=SEL2_1_FUN(A,B,SEL );//注意此行不可放在函数定义之前
endmodule
```

(3)二选一数据选择器的仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 SEL2_1。

②设计输入

单击 File/new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 SEL2_1. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin→Location→Chip 命令,此时会弹出引脚锁定对话框。在 Node Name 栏内输入信号端口的名称 SEL;然后在 Chip Resoure 栏中选择 Pin,根据需要输入要锁定的引脚序号 1。在 Pin type 中输入 **input**,这时的 Add 按钮呈现有效状态,单击它即可把刚才输入的引脚及其信号添加到 Existing Pin/Location/Chip Assignments 窗口中。依此方法锁定 A、B、F 引脚,锁定完毕单击 OK 即可,如图 8-10 所示。

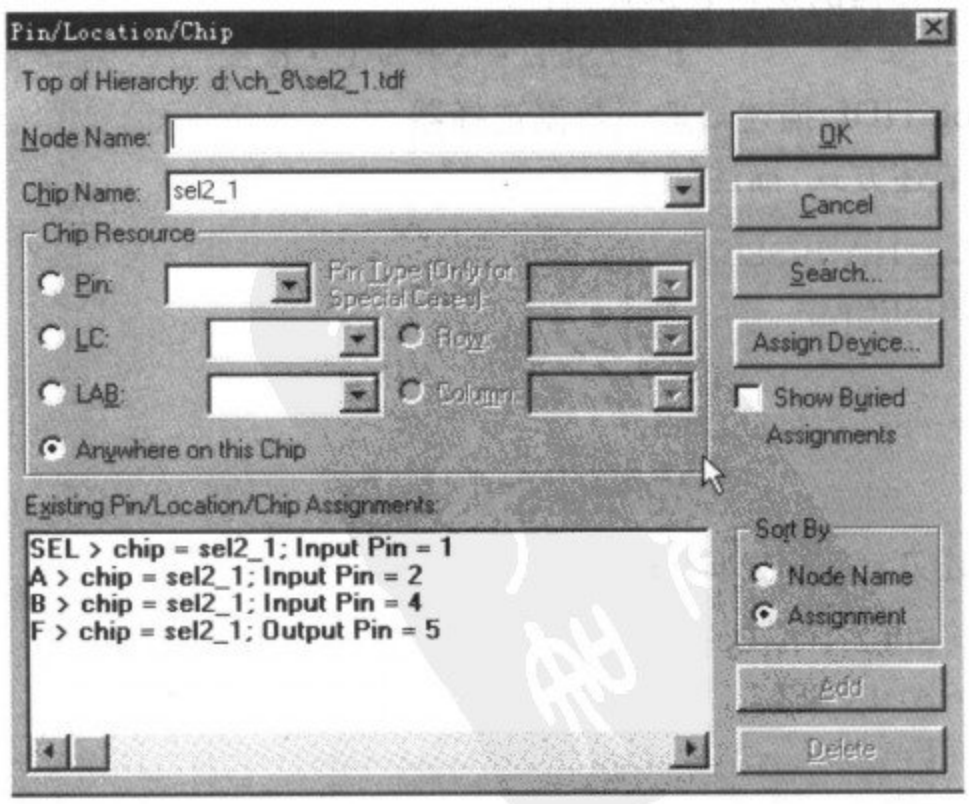


图 8-10　二选一数据选择器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“⇒”图标，将 A、B、SEL、F 信号加入 SNF 文件中，如图 8-11 所示。

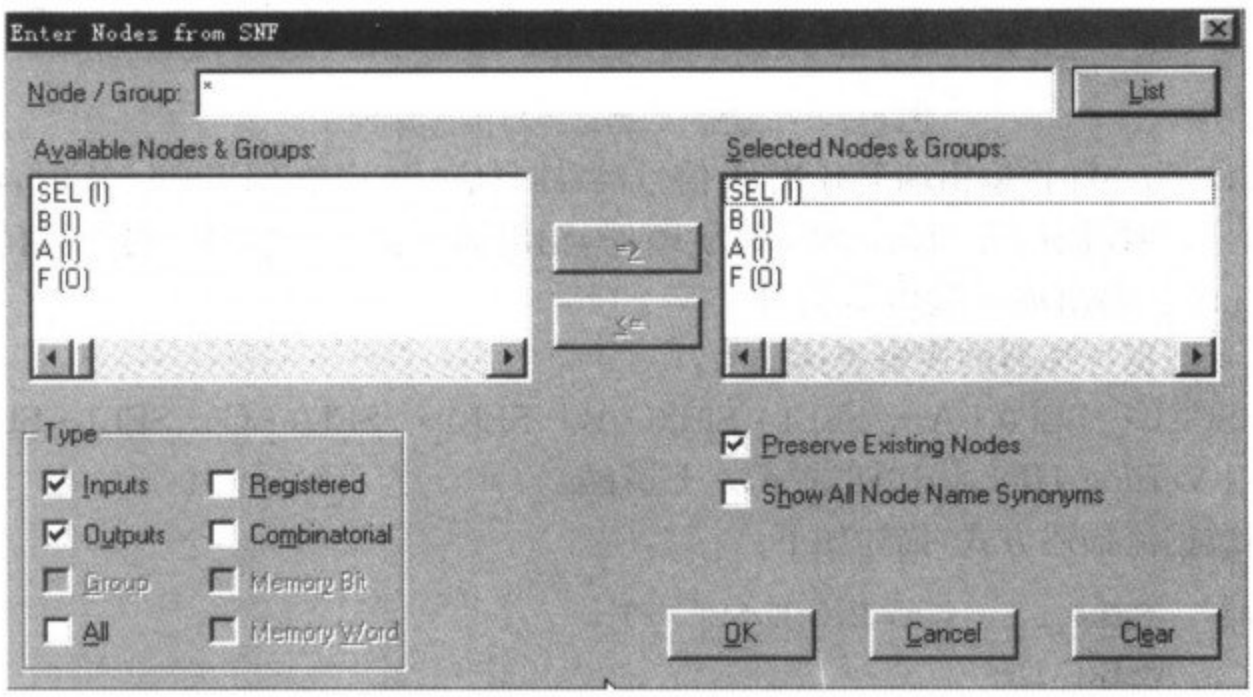


图 8-11　二选一数据选择器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

下面，就可以对 SEL、A、B 输入信号赋值。

先选中 SEL，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1(高电平)，Increment By 增加值设为 0(低电平)，Multiplied By 设为 200(即半周期为 20μs)。点击 OK 按钮。

按以上方法将 A 设置为开始电平设为 1(高电平)，增加值设为 0(低电平)，Multiplied By 设为 50(即半周期为 5μs)。将 B 设置为开始电平设为 0(低电平)，增加值设为 1(高电平)，Multiplied By 设为 50(即半周期为 5μs)。单击保存按钮，将文件保存为 SEL2_1. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-12 所示。

从波形图中可以看出，当 SEL 为 1 时，输出端 F 输出 B 信号，当 SEL 为 0 时，输出端 F 输出 A 信号。波形图显示的逻辑功能和设计目的完全一样。

2. 四选一数据选择器

(1)四选一数据选择器分析

四选一数据选择器有 4 个输入端，这里定义为 A、B、C、D；2 个控制端，这里定义为 SEL0、SEL1；1 个输出端，这里定义为 F。四选一数据选择器的逻辑功能是：当控制信号

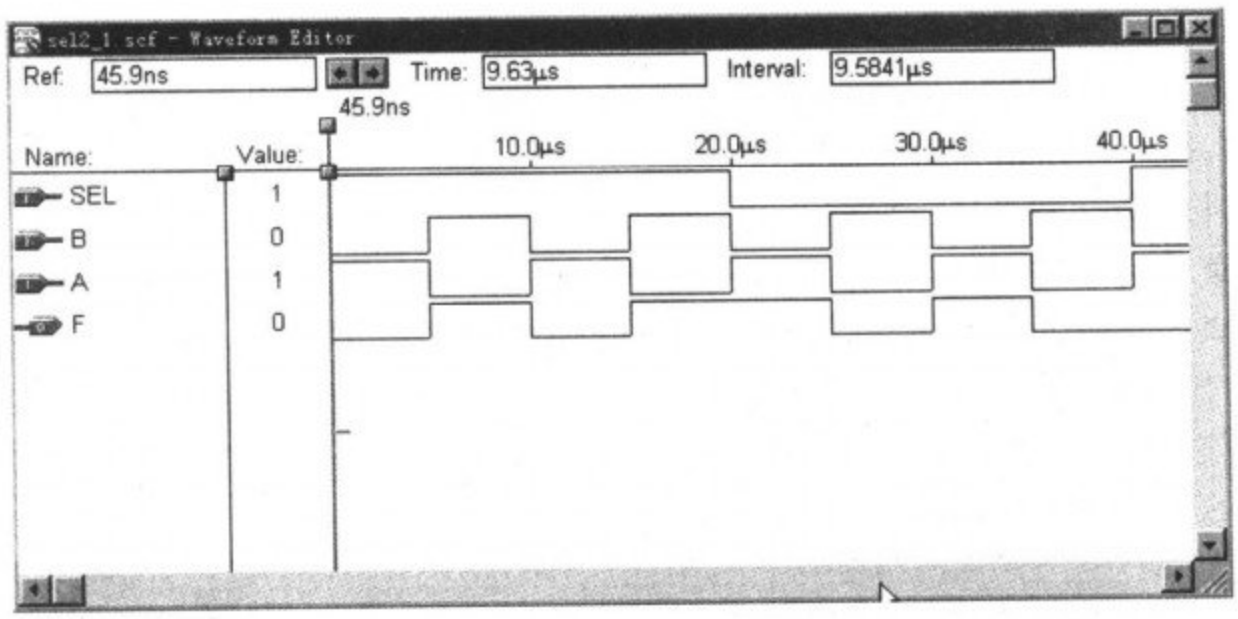


图 8-12　二选一数据选择器仿真波形图

SEL1、SEL0 为 00 时，输出端 F 输出 A 信号；当控制信号 SEL1、SEL0 为 01 时，输出端 F 输出 B 信号；当控制信号 SEL1、SEL0 为 10 时，输出端 F 输出 C 信号；当控制信号 SEL1、SEL0 为 11 时，输出端 F 输出 D 信号。

四选一数据选择器的逻辑表达式为：

F=/SEL1・/SEL0・A+/SEL1・SEL0・B+ SEL1・/SEL0・C+ SEL1・SEL0・D

(2)用 Verilog HDL 描述四选一数据选择器

采用数据流描述方式，描述如下：

```
module    SEL4_1    ( A, B,C,D,SEL, F);
input     A, B,C,D;
input  [1:0]  SEL;
output    F;
assign    F = ( ~ SEL1& ~ SEL0&A) | ( ~ SEL1&SEL0&B) | ( SEL1& ~ SEL0&C) |
(SEL1&SEL0&D)
endmodule
```

采用行为描述方式，描述如下：

```
module    SEL4_1( A, B, C, D, SEL, F );
input     A, B, C, D;
input     [1:0] SEL;
output    F;
function     SEL4_1_FUNC;
    input     A, B, C, D;
    input     [1:0] SEL;
     case (SEL)
      2'b00:SEL4_1_FUNC = A;
      2'b01:SEL4_1_FUNC = B;
      2'b10:SEL4_1_FUNC = C;
      2'b11:SEL4_1_FUNC = D;
          endcase
```

```
endfunction
assign     F=SEL4_1_FUNC(A, B, C, D, SEL);
endmodule
```

(3)四选一数据选择器的仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 SEL4_1。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 SEL4_1. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。在 Node Name 栏内输入信号端口的名称 SEL1;然后在 Chip Resoure 栏中选择 Pin,根据需要输入要锁定的引脚序号 1。在 Pin type 中输入 **input**,这时的 Add 按钮呈现有效状态,单击它即可把刚才输入的引脚及其信号添加到 Existing Pin/Location/Chip Assignments 窗口中。依此方法锁定 SEL0、A、B、C、D、F 引脚,锁定完毕单击 OK 即可,如图 8-13 所示。

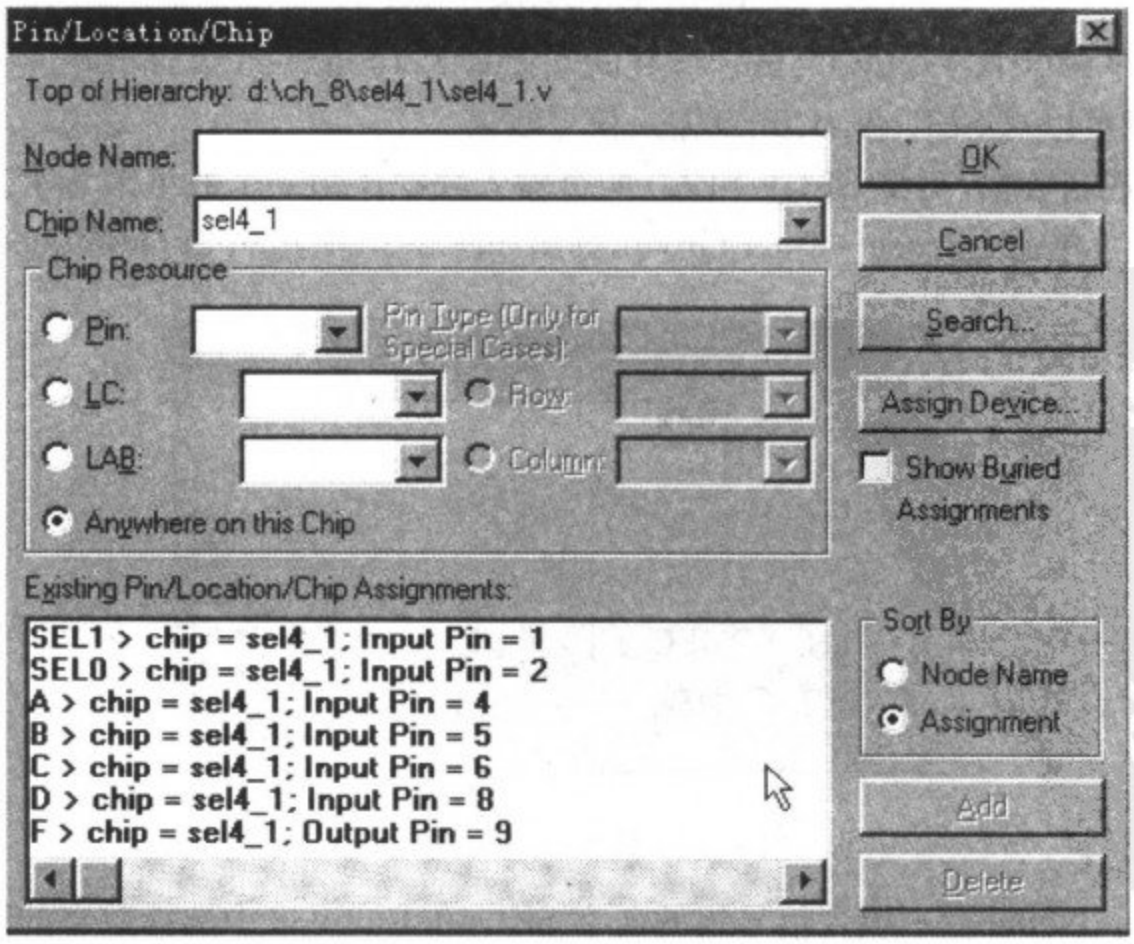


图 8-13　四选一数据选择器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 A、B、C、D、SEL1、SEL0、F 信号加入 SNF 文件中，如图 8-14 所示。

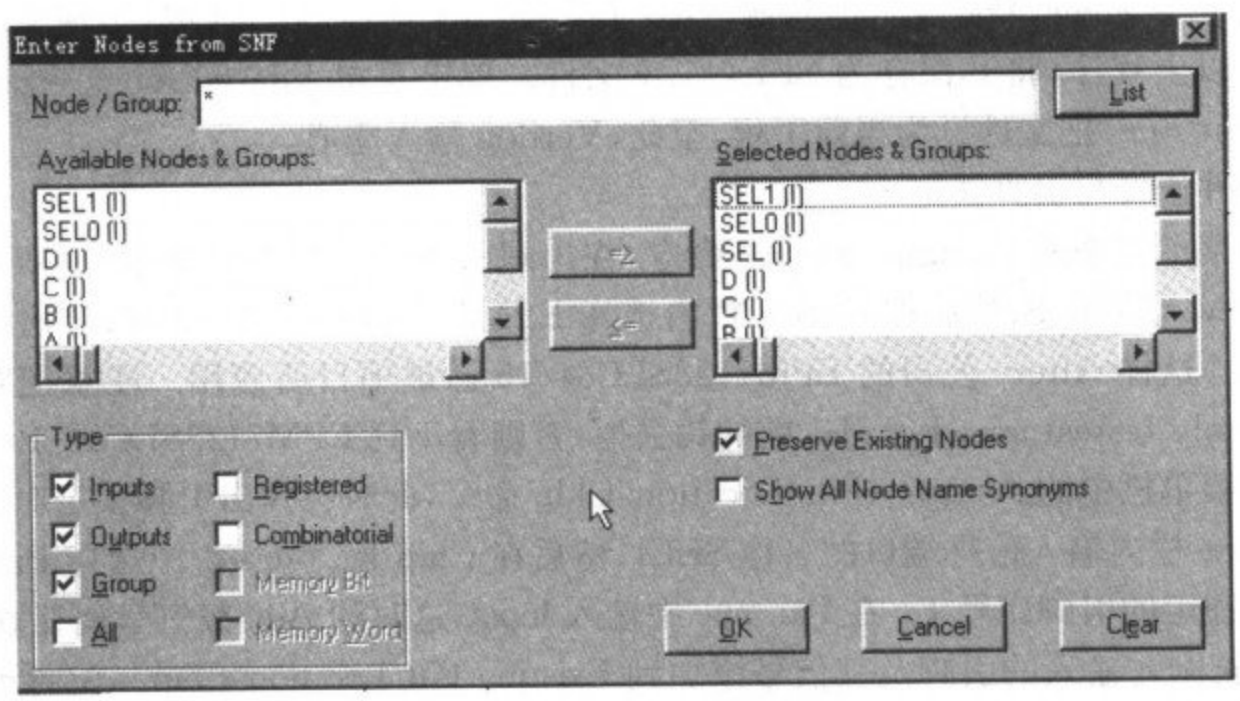


图 8-14　仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

下面，就可以对 SEL、A、B、C、D 输入信号赋值。

为便于观测数据，可将 SEL1、SEL0 两个输入数据作为一个数组来显示。方法是：选中 SEL，执行菜单命令 Node→Enter Group，弹出如图 8-15 所示的 Enter Group 对话框。

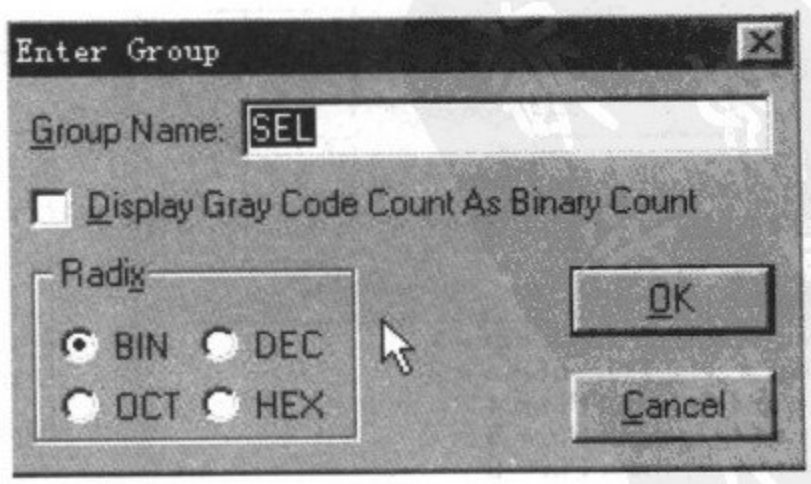


图 8-15　Enter Group 对话框

在 Group Name 栏中填入信号组的名字 SEL；在 Radix 区中选择数字的进位制；选项 Display Gray Code Count As Binary Count 决定在以二进制计数时是否以格雷码显示。

然后，再点击XC按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1(高电平)，Increment By 增加值设为 0(低电平)，Multiplied By 设为 200(即半周期为 20μs)。点击 OK 按钮。

按以上方法将 A 设置为开始电平设为 1(高电平)，增加值设为 0(低电平)，Multiplied By 设为 50(即半周期为 5μs)。将 B 设置为开始电平设为 0(低电平)，增加值设为 1(高电平)，Multiplied By 设为 50(即半周期为 5μs)。将 C 设置为开始电平设为 1(高电平)，增加值设为 0(低电平)，Multiplied By 设为 100(即半周期为 10μs)。将 D 设置为开始电平设为 0(低电平)，增加值设为 1(高电平)，Multiplied By 设为 100(即半周期为 10μs)单击保存按钮，将文件保存为 SEL4_1. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-16 所示。

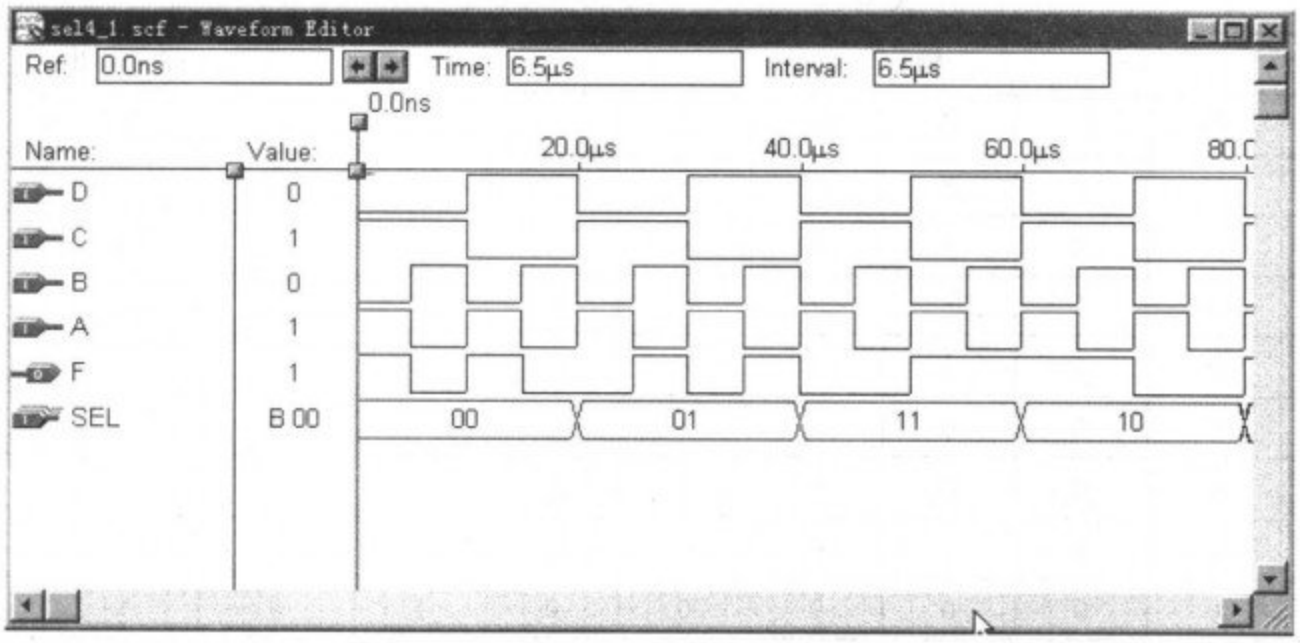


图 8-16　四选一数据选择器仿真波形图

从波形图中可以看出，波形图显示的逻辑功能和设计目的完全一样。

二、编码器

1. 编码器分析

在数字系统里，常常需要将某一信息(输入)变换为某一特定的代码(输出)。把二进制码按一定的规律编排，例如 8421 码、格雷码等，使每组代码具有一特定的含义(代表某个数字或控制信号)称为编码，具有编码功能的逻辑电路称为编码器。编码器有若干个输入，但在某一时刻只有一个输入信号被转换成为二进制码。常用的编码器有二进制编码器和二-十进制编码器等，下面以 3 位二进制优先编码器(也称为 8-3 优先编码器)为例进行介绍。

如图 8-17 所示是 8-3 优先编码器的示意框图，I0～I7 是要进行优先编码的 8 个输入信号，Y0～Y3 是用来进行优先编码的 3 位二进制代码。

I0～I7 八个输入信号中，假定 I7 优先级别最高，I6 次之，依此类推，I0 最低，并分别用 Y2Y1Y0 取值为 000、001、…、111 表示 I0、I1、…、I7。根据优先级别高的输入信号排斥低的特点，即可列出优先编码器的简化真值表，如表 8-3 所示。

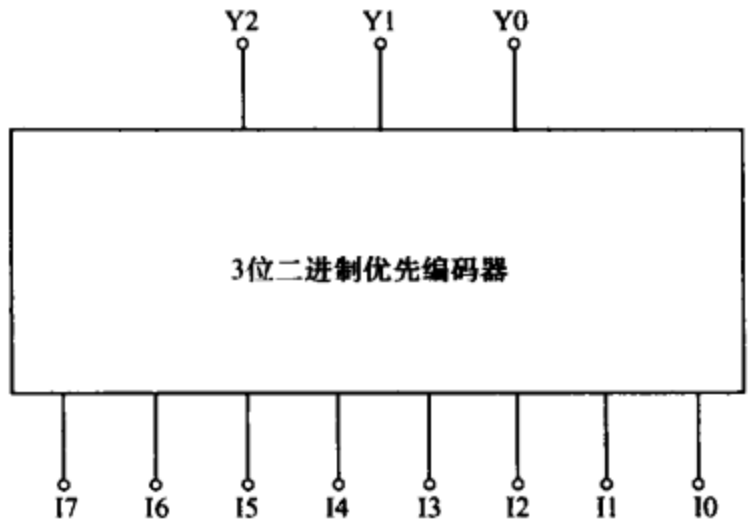


图 8-17　优先编码器框图

表 8-3　3 位二进制优先编码表

输　入								输　出		
I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
1	×	×	×	×	×	×	×	1	1	1
0	1	×	×	×	×	×	×	1	1	0
0	0	1	×	×	×	×	×	1	0	1
0	0	0	1	×	×	×	×	1	0	0
0	0	0	0	1	×	×	×	0	1	1
0	0	0	0	0	1	×	×	0	1	0
0	0	0	0	0	0	1	×	0	0	1
0	0	0	0	0	0	0	1	0	0	0

优先编码表表中“×”号的意思是被排斥，也就是说有优先级别高的信号存在时，级别低的输入信号取值无论是 0 还是 1 都无所谓，对电路输出均无影响。

2. 用 Verilog 描述 8-3 优先编码器

```
module code8_3(I,Y);
input[7:0] I;
output[2:0] Y;
function[2:0] code;                          //函数定义
  input[7:0] I;                              //函数只有输入端口，输出为函数名本身
  if(I[7])          code = 3'b111;
  else if(I[6])     code = 3'b110;
  else if(I[5]) code = 3'b101;
  else if(I[4]) code = 3'b100;
  else if(I[3]) code = 3'b011;
  else if(I[2]) code = 3'b010;
  else if(I[1]) code = 3'b001;
  else              code = 3'd000;
endfunction
assign Y=code(I);                            //函数调用
```

```
endmodule
```

3.8-3 优先编码器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 code8_3。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 code8_3 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 I0、I1、I2、I3、I4、I5、I6、I7 和 Y0、Y1、Y2，锁定完毕单击 OK 即可，如图 8-18 所示。

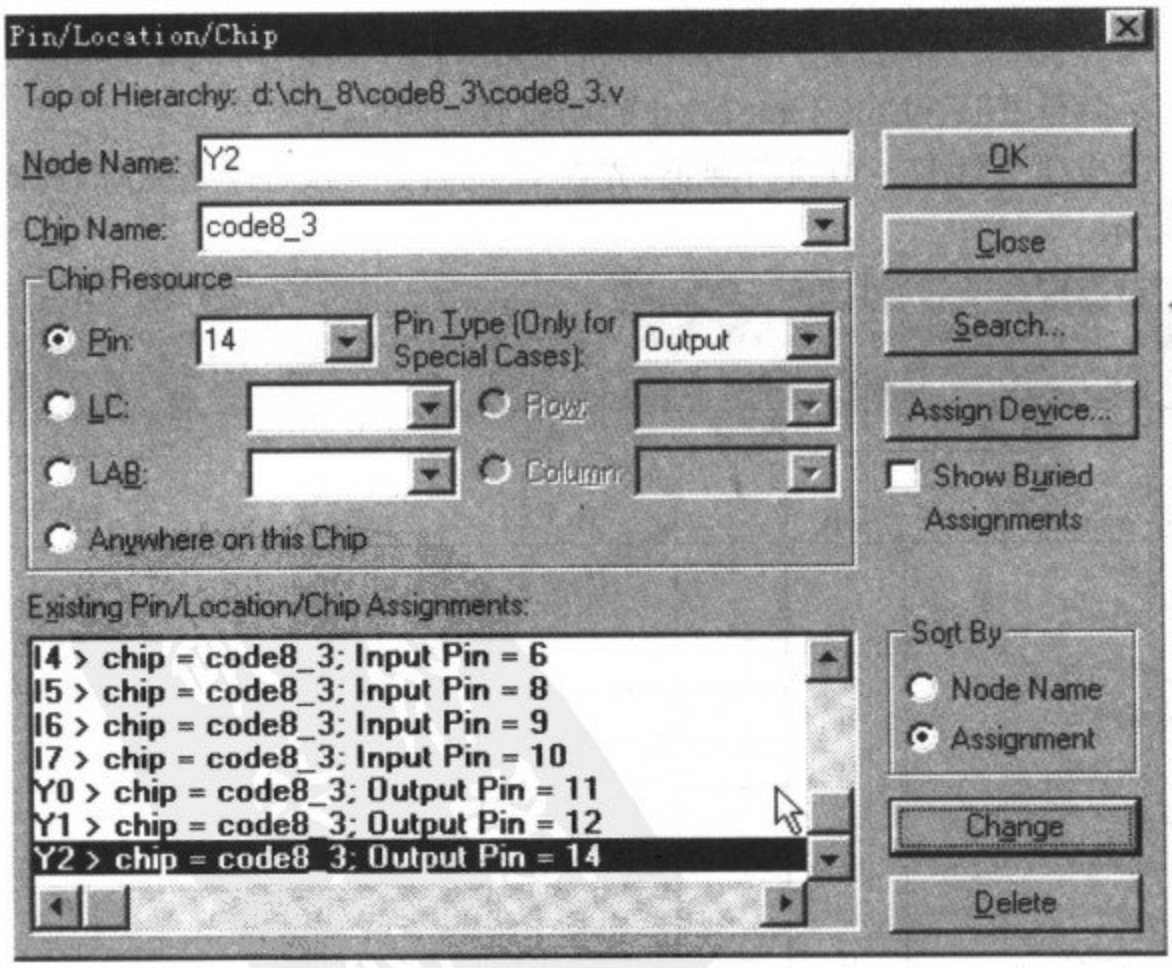


图 8-18　8-3 优先编码器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 Enter Nodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 I0～I7、Y0～Y2 信号加入 SNF 文件中，如图 8-19 所示。

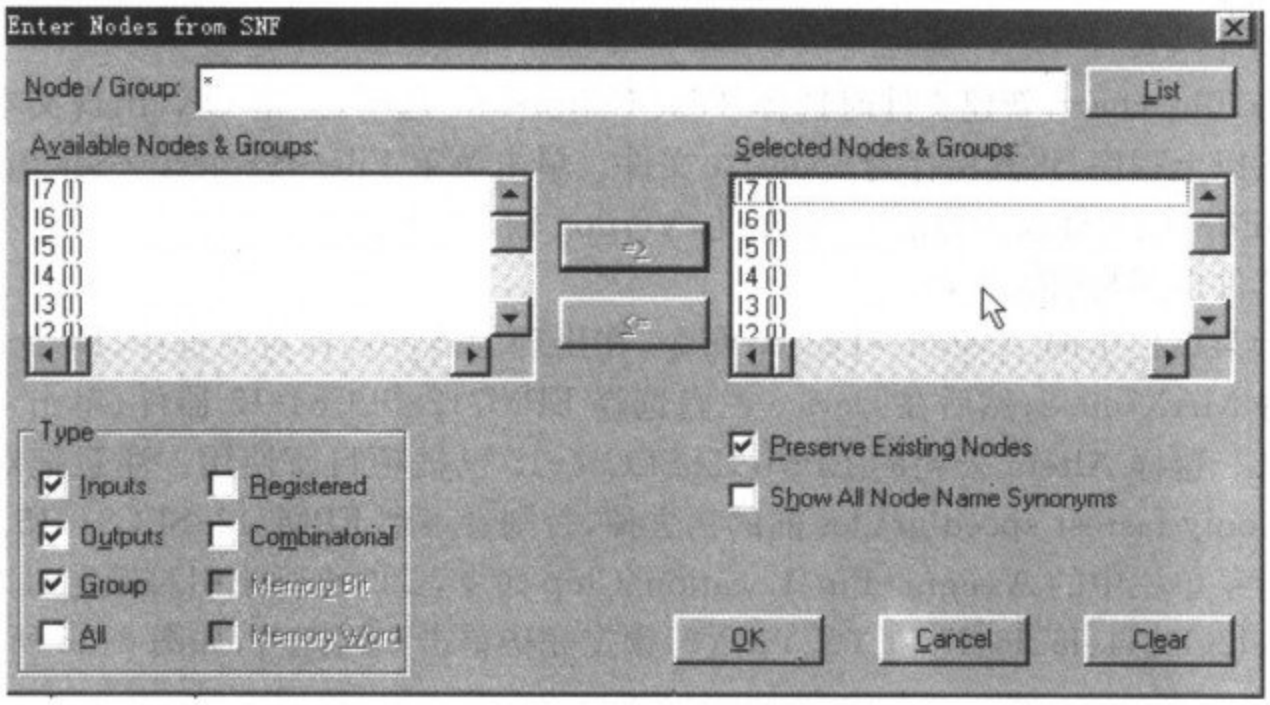


图 8-19　仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

下面采用拖动法对 I7～I0 进行赋值：先在 I7 的右方 5μs 处单击鼠标，此时出现光标，如图 8-20 所示。

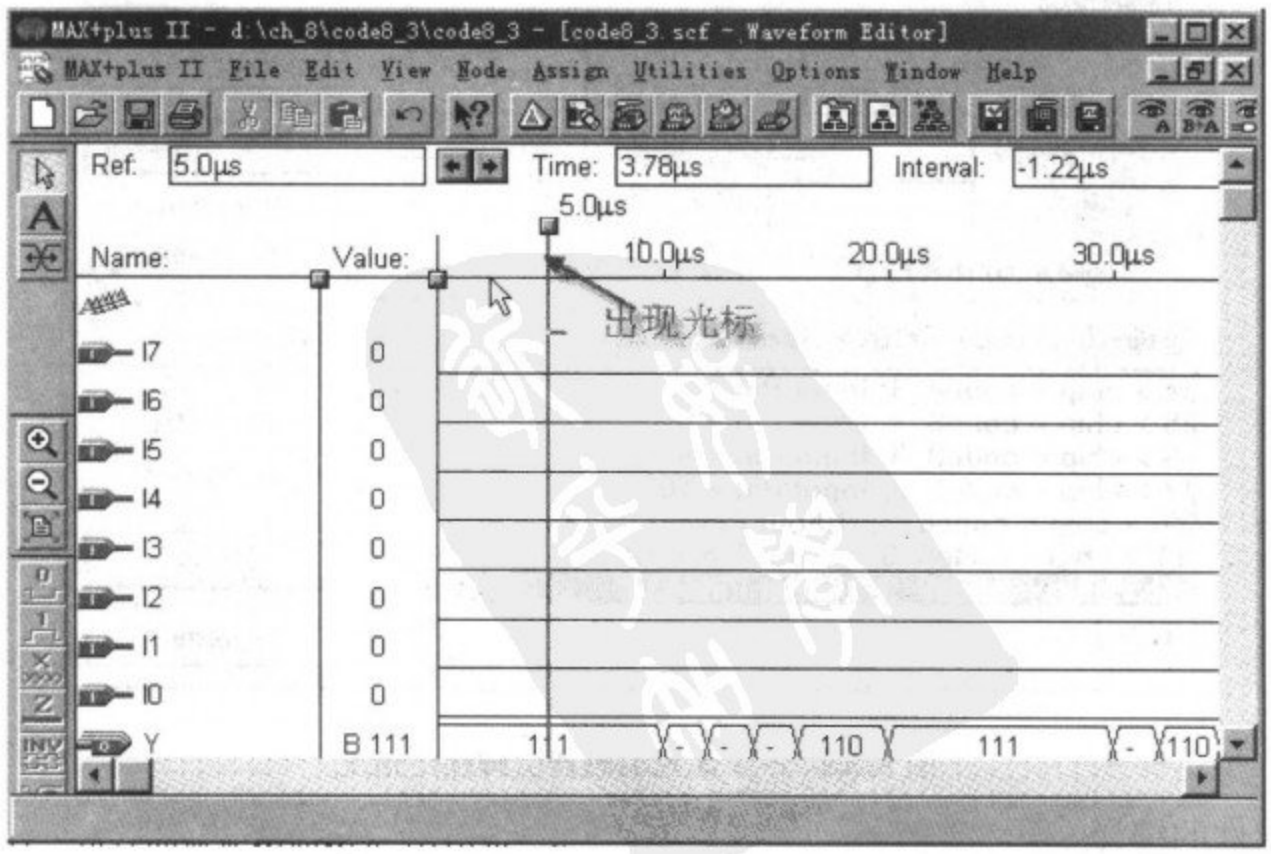


图 8-20　出现光标

从右向左拖动鼠标到 0μs 处，此时在(0～5)μs 间会出现一个拖动的黑块，如图 8-21

所示。

点击按钮，将(0～5)μs 处的小黑块部分设置为高电平。如图 8-22 所示。

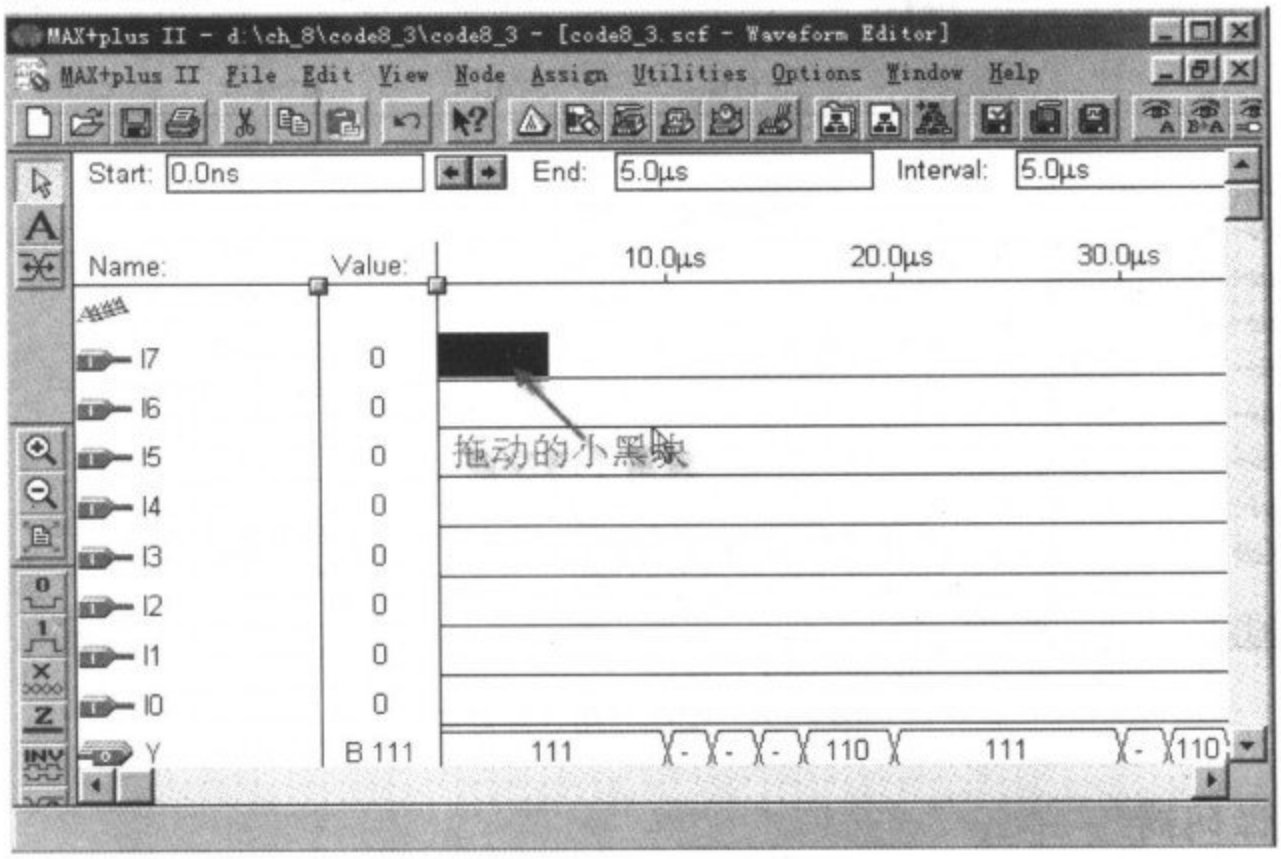


图 8-21　拖动鼠标

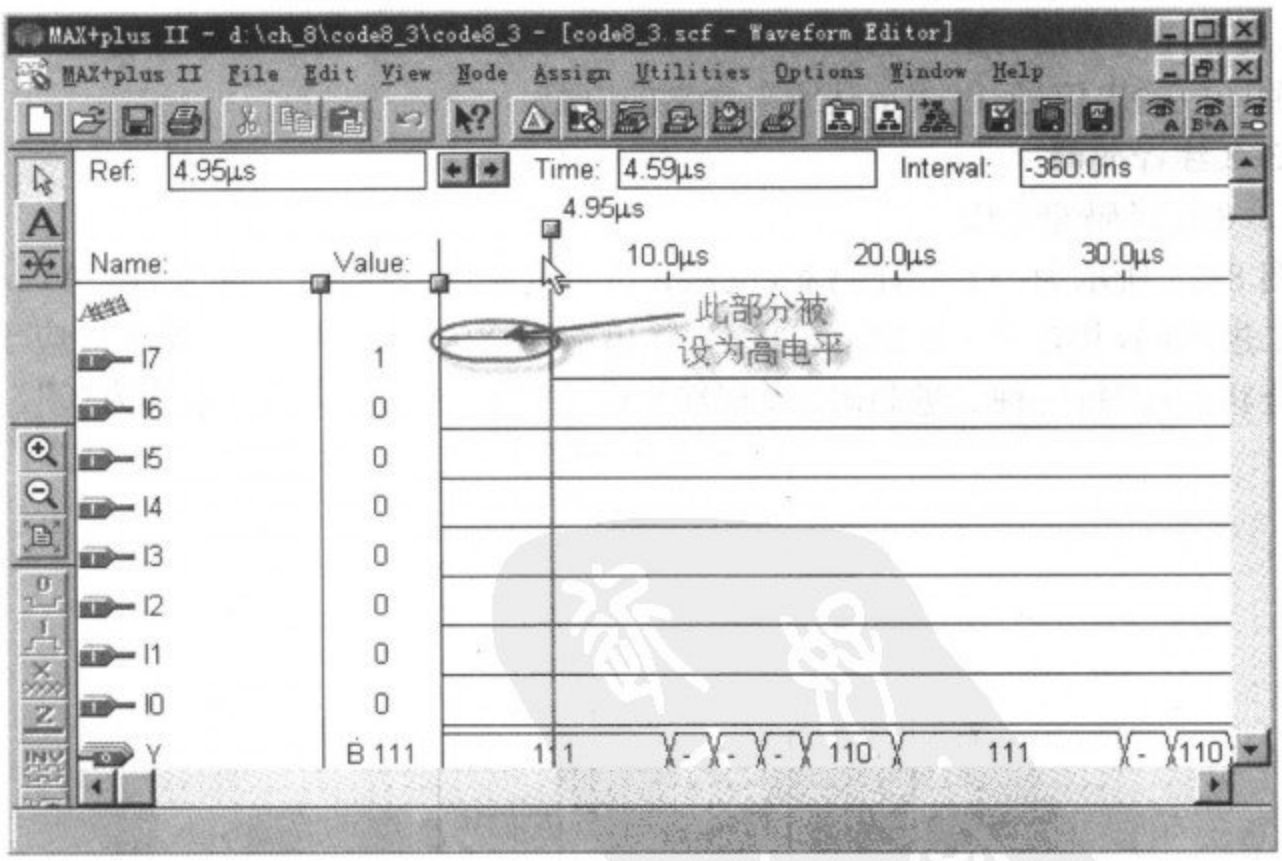


图 8-22　将(0～5)μs 部分设置为高电平

采用上面相同的方法将 I6 的(5～10)μs 部分设为高电平，将 I5 的(10～15)μs 部分设为高电平，将 I4 的(15～20)μs 部分设为高电平，将 I3 的(20～25)μs 部分设为高电平，将 I2 的(25～30)μs 部分设为高电平，将 I1 的(30～35)μs 部分设为高电平，将 I0 的(35～40)μs 部分设为高电平。单击保存按钮，将文件保存为 cide8_3. scf 文件。

执行菜单栏中的 MAX＋plusII→Simulator 命令，在弹出的对话框中，点击 Start 开

始按钮，开始仿真，仿真的波形图如图 8-23 所示。

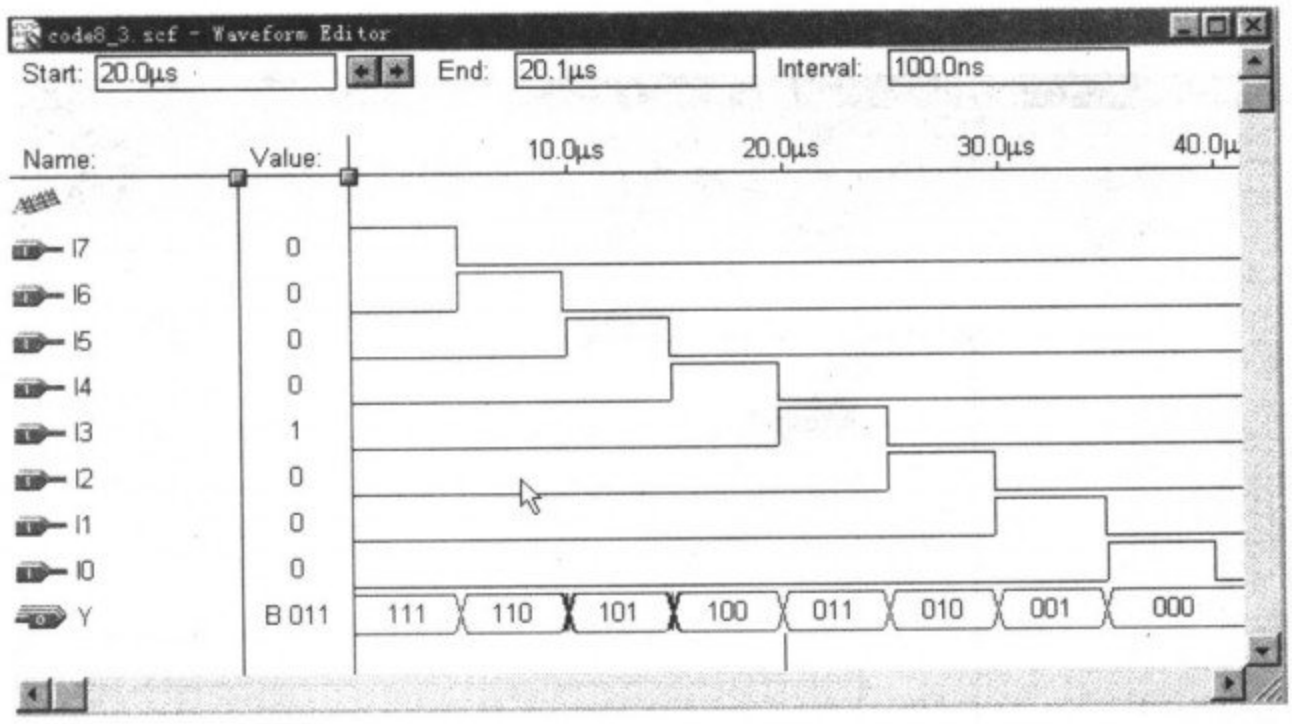


图 8-23　编码器仿真波形图

三、译码器

译码是编码的逆过程，它的功能是将具有特定含义的二进制码进行辨别，并转换成控制信号，具有译码功能的逻辑电路称为译码器。也就是说，译码器可以将输入二进制代码的状态翻译成输出信号，以表示其原来含义的电路。

1. 3-8 线译码器

(1)3-8 线译码器分析

如图 8-24 所示为 3-8 线译码器，也称 3 位二进制译码器，输入端为 A0、A1、A2，可输入 3 位二进制码，共有 $2^3=8$ 种组合状态，输出端为 Y0～Y7 共 8 线，要求对应于输入的每种组合状态(即每一种二进制码)，8 根输出线中只有一根为“1”，其余均为“0”。

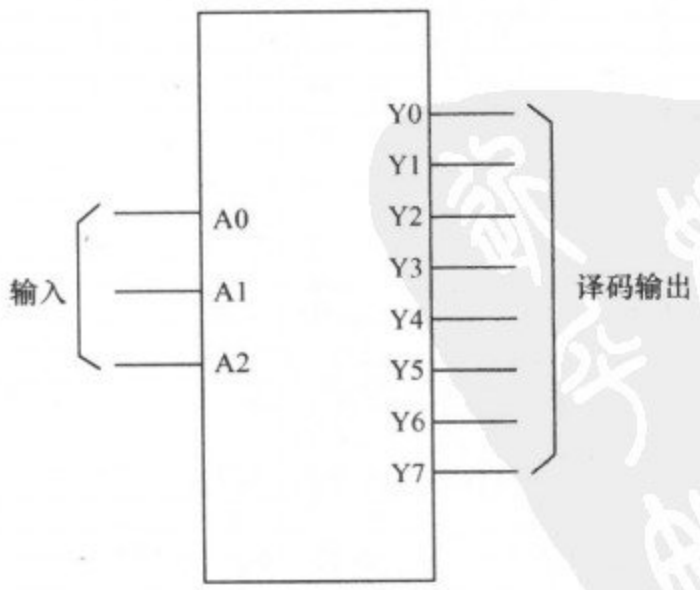


图 8-24　3-8 线译码器

设输入 A0、A1、A2 的 8 种组合状态分别对应输出信号 Y0～Y7 的 8 种情况，在这 8 种情况中，每种都只有一个输出端为“1”，译码真值表如表 8-4 所示。

表 8-4　3-8 线译码器的真值表

输　入			输　出							
A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

(2)用 Verilog 描述 3-8 线译码器

用 Verilog 描述 3-8 线译码器如下：

```
module decode3_8(Y,A);
output[7:0] Y;
input[2:0] A;
reg[7:0] Y;
always @(A)
  begin
    case(A)
      3'b000: Y=8'b00000001;
      3'b001: Y=8'b00000010;
      3'b010: Y=8'b00000100;
      3'b011: Y=8'b00001000;
      3'b100: Y=8'b00010000;
      3'b101: Y=8'b00100000;
      3'b110: Y=8'b01000000;
      3'b111: Y=8'b10000000;
      endcase
  end
endmodule
```

(3)3-8 线译码器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 decode3_8。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 decode3_8. v 文件。单击菜单 File→Project→Set Project

To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 Y0～Y7 和 A0～A2，锁定完毕单击 OK 即可，如图 8-25 所示。

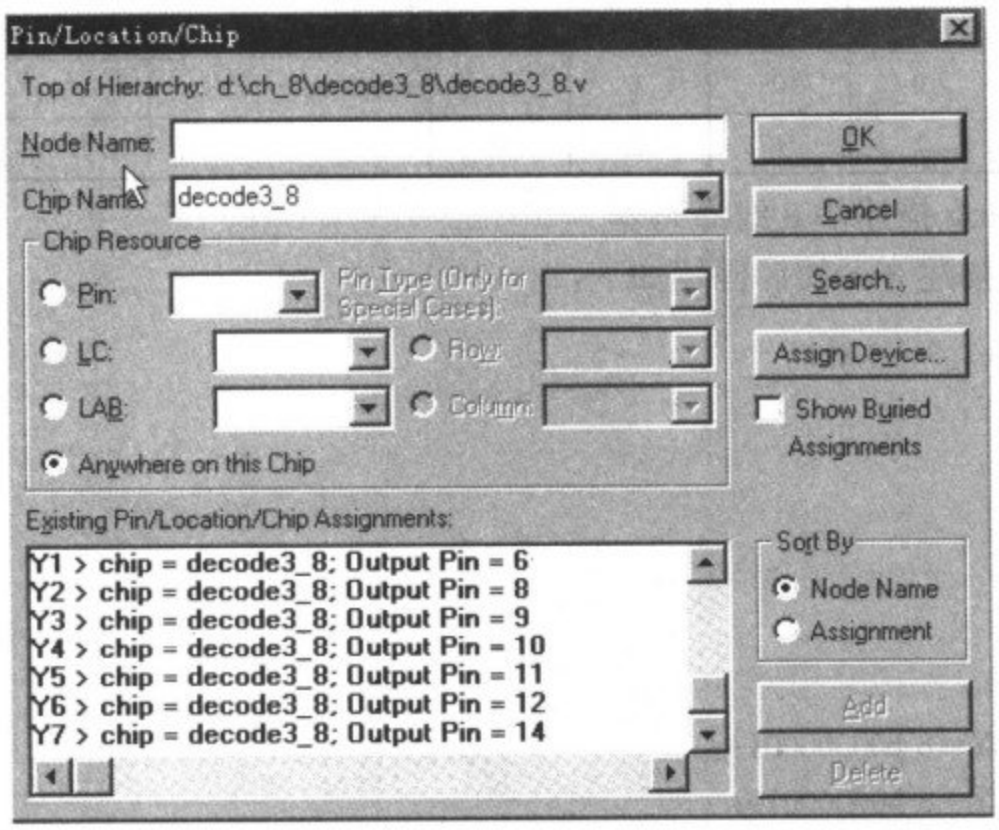


图 8-25　3-8 线译码器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 I0～I7、Y0～Y2 信号加入 SNF 文件中，如图 8-26 所示。

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 A[2:0]，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 000，Increment By 增加值设为 001，Multiplied By 设为 50（即半周期为 5μs）。点击 OK 按钮，如图 8-27 所示。单击保存按钮，将文件保存为 decode3_8. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-28 所示。

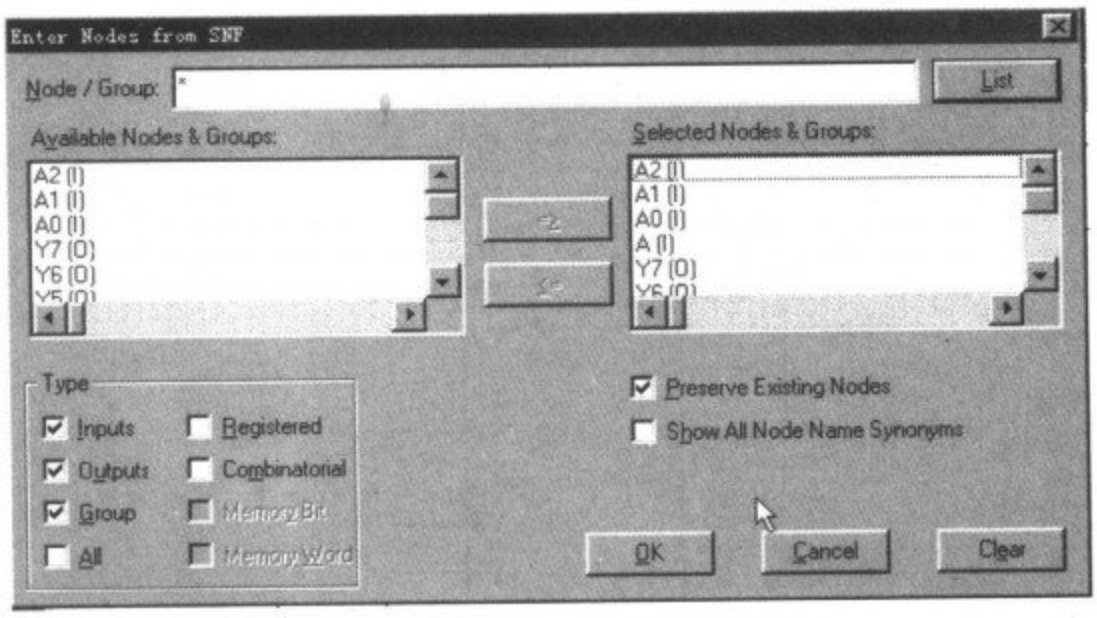


图 8-26　3-8 线译码器仿真端口列表

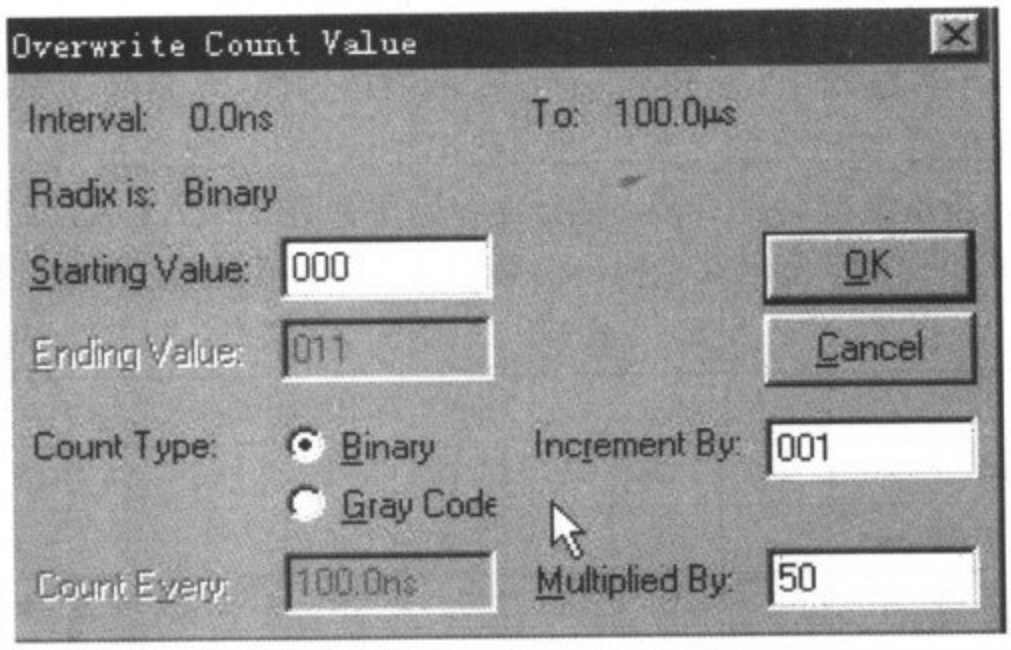


图 8-27　设置 A 输入状态

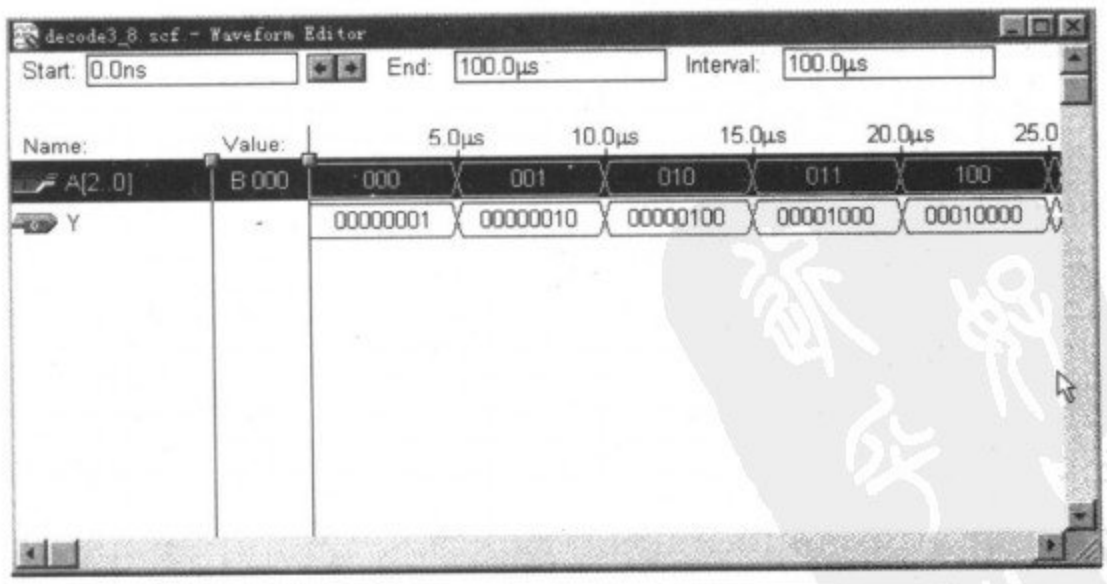


图 8-28　3-8 线译码器仿真波形图

2. 4-10 线译码器

(1)4-10 线译码器分析

将二-十进制代码翻译 10 个十进制数字信号的电路叫做二-十进制译码器。这种译码器的输入是十进制数的二进制编码(BCD)码,输出的 10 个信号分别与十进制数的 10

个数字相对应，其示意框图如图 8-29 所示。这种译码器也常称为 4 线-10 线译码器。

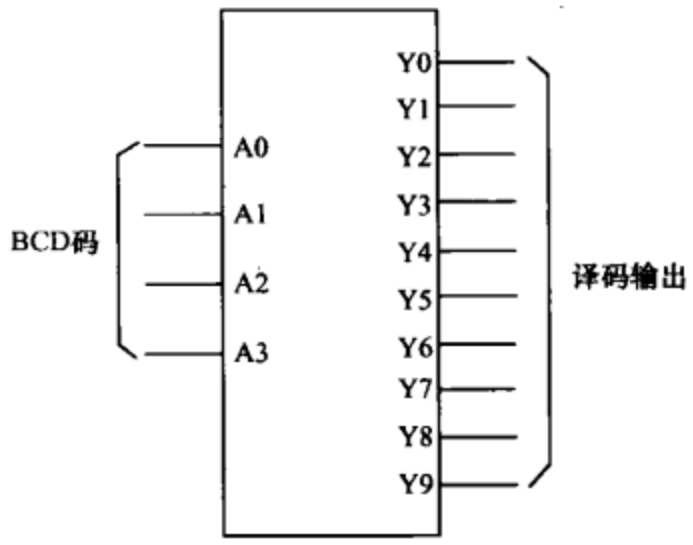


图 8-29　4 线-10 线译码器

在 4 线-10 线译码器中，比较常用的是 8421BCD 码输入的 4 线-10 线译码器。8421BCD 码告诉我们，在输入的 4 位二进制代码中，0000 的含义是 0，也即表示的是 0，0001 表示的是 1，依次类推，……1001 表示的是 9，如果用 A3、A2、A1、A0 表示输入的二进制代码，用 Y0～Y9 表示范区 0 个输出信号 0～9，据此可列出真值表如表 8-5 所示。

表 8-5　4 线-10 线译码器真值表

输入				输出									
A3	A2	A1	A0	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	0	×	×	×	×	×	×	×	×	×	×
1	0	1	1	×	×	×	×	×	×	×	×	×	×
1	1	0	0	×	×	×	×	×	×	×	×	×	×
1	1	0	1	×	×	×	×	×	×	×	×	×	×
1	1	1	0	×	×	×	×	×	×	×	×	×	×
1	1	1	1	×	×	×	×	×	×	×	×	×	×

在 8421BCD 码中，代码 1010～1111 六种取值没有用，在正常的情况下不会在译码器

的输入端出现，因此称之为伪码，相应地，在译码器的各个输出信号处均记上“×”号。

(2)用 Verilog HDL 描述 4-10 线译码器

用 Verilog HDL 描述 4-10 线译码器如下：

```
module decode4_10(A,Y);
output[9:0] Y;
input[3:0] A;
reg[9:0] Y;
always @(A)
  begin
    case(A)
    4'b0000:Y=10'b0000000001;
    4'b0001:Y=10'b0000000010;
    4'b0010:Y=10'b0000000100;
    4'b0011:Y=10'b0000001000;
    4'b0100:Y=10'b0000010000;
    4'b0101:Y=10'b0000100000;
    4'b0110:Y=10'b0001000000;
    4'b0111:Y=10'b0010000000;
    4'b1000:Y=10'b0100000000;
    4'b1001:Y=10'b1000000000;
    default:Y=10'bx;
    endcase
  end
endmodule
```

(3)4-10 线译码器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 decode4_10。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 decode4_10. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 Y0～Y9 和 A0～A3，锁定完毕单击 OK 即可，如图 8-30 所示。

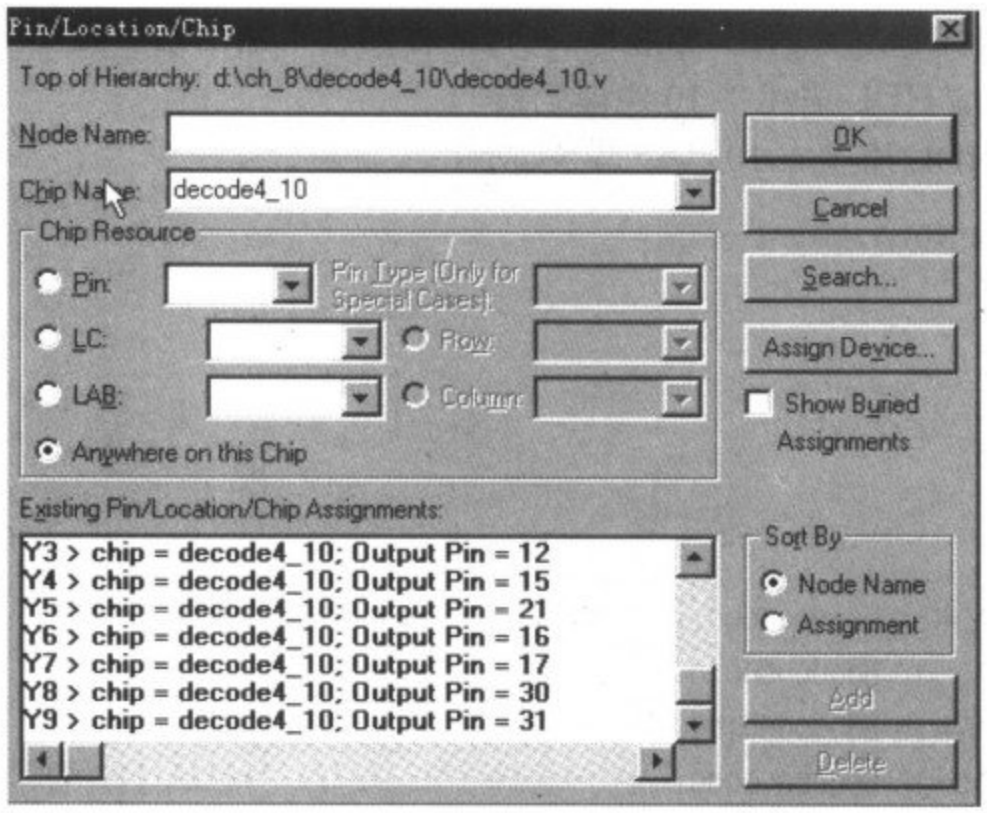


图 8-30　4-10 线译码器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 Y0～Y9、A0～A3 信号加入 SNF 文件中，如图 8-31 所示。

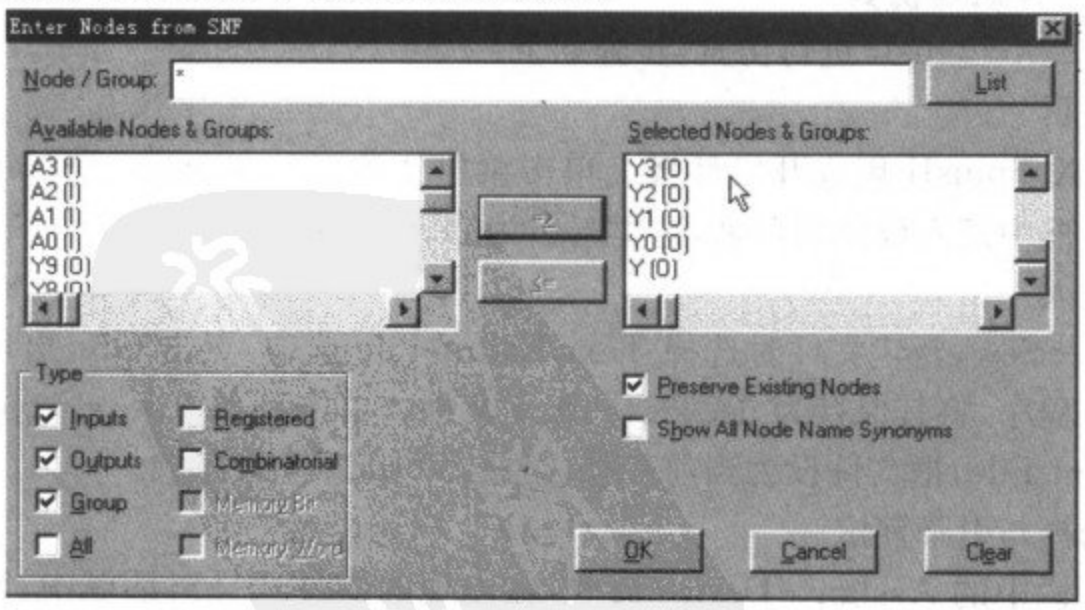


图 8-31　4-10 线译码器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 A[3:0]，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0000，Increment By 增加值设为 0001，Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮，如图 8-32 所示。单击保存按钮，将文件保存为 decode4_10. scf 文件。

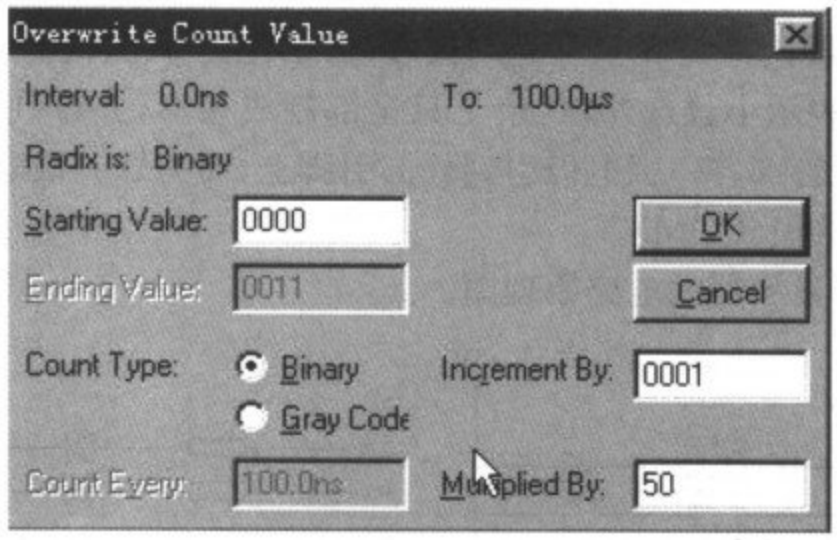


图 8-32　设置 A 输入状态

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-33 所示。

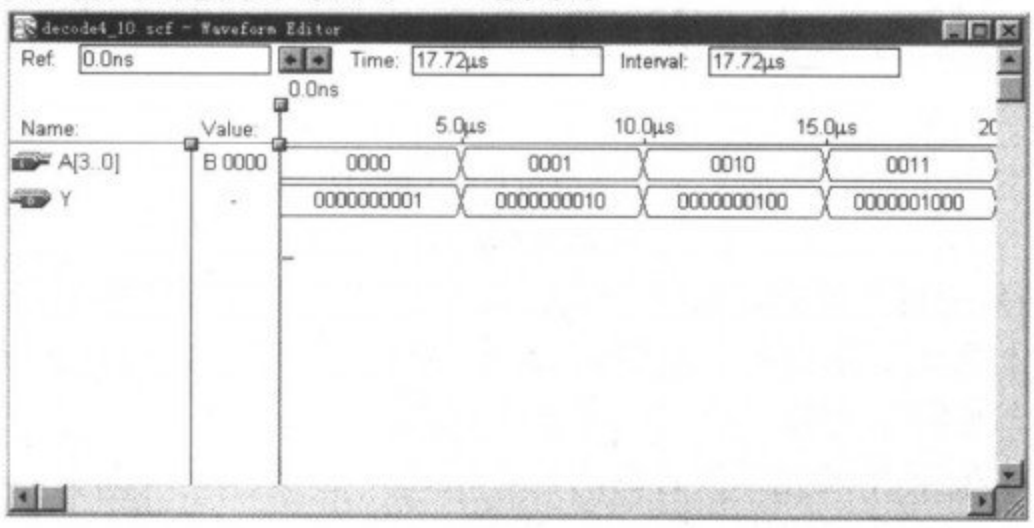


图 8-33　4-10 线译码器仿真波形图

3. LED 显示译码器

(1)LED 显示器的结构

LED 是发光二极管的简称，其 PN 结是用某些特殊的半导体材料(如磷砷化镓)做成的，当外加正向电压时，可以将电能转换成光能，从而发出清晰悦目的光线。如果将多个 LED 管排列好并封装在一起，就成为 LED 显示器。LED 发光管和常用的 8 段 LED 显示器(也称 LED 数码管)的结构如图 8-34 所示。

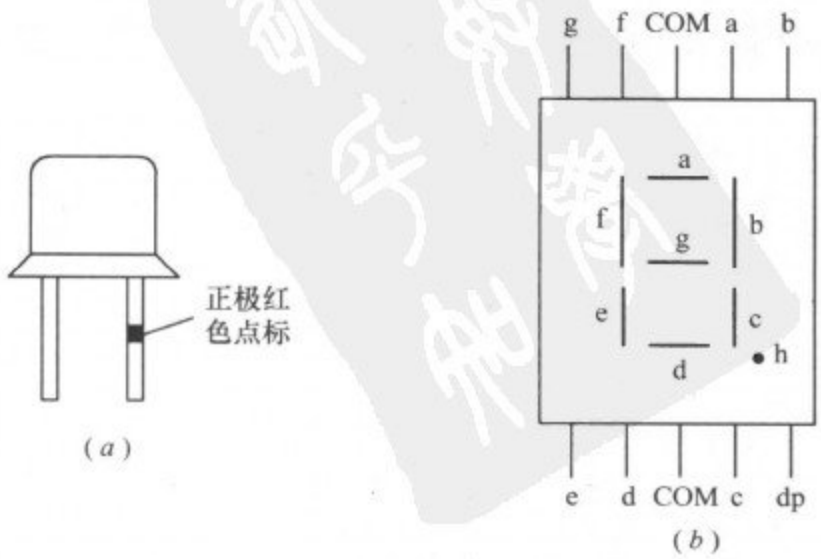


图 8-34　LED 发光二极管和 LED 显示器

(a) 发光二极管；(b) 数码管。

图中，LED 显示器内部是 8 只发光二极管，分别记为 a、b、c、d、e、f、g、dp(h)，其中除 dp(h)制成圆形用以表示小数点外，其余 7 只全部制成条形，并排列成如图所示的“8”字形状。每只发光二极管都有一根电极引到外部引脚上，而另外一根电极全部连接在一起，引到外引脚，称为公共极(COM)。

图 8-35 是 LED 显示器的电路原理图。

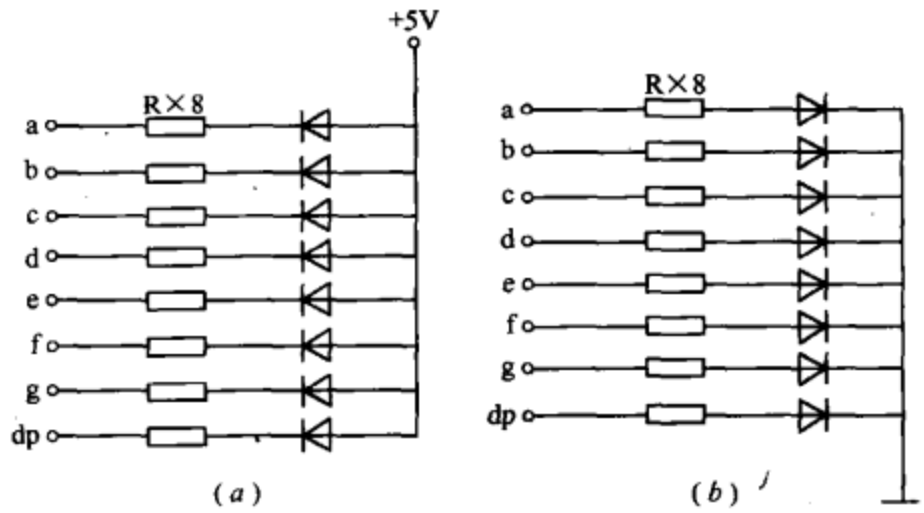


图 8-35　共阳和共阴 LED 显示器

(a) 共阳极；(b) 共阴极。

图 8-35(a)是把各个发光二极管的阳极都连在一起，从 COM 端引出，阴极分别从其他 8 根引脚引出，称为共阳结构。使用时，公共阳极接＋5V，这样，阴极端输入低电平的发光二极管就导通点亮，而输入高电平的段则不能点亮。

图 8-35(b)是把各个发光二极管的阴极都接在一起，从 COM 端引出，阳极分别从其他 8 根引脚引出，称为共阴结构。使用时，公共阴极接地，这样，阳极端输入高电平的发光二极管就导通点亮，而输入低电平的段则不能点亮。

在购买和使用 LED 显示器时，必须说明是共阴还是共阳结构。

(2)LED 显示器的原理

根据 LED 显示器结构可知，如果希望显示“8”字，那么除了“dp”管不要点亮以外，其余管全部点亮，其余均不必点亮。同理，如果要显示“1”，那么，只需 b、c 两个发光二极管点亮。对于共阳结构，就是要把公构端 COM 接到电源正极，而 b、c 两个负极分别经过一个限流电阻后接低电平；对于共阴结构，就是要把公共端 COM 接低电平(电源负极)，而 b、c 两个正极分别经一个限流电阻后接到高电平。接照同样的方法分析其他显示数和字型码，如表 8-6 所示。

表 8-6　8 段 LED 数码管段位与显示字型码的关系

显示	共阳									共阴								
	dp	g	f	e	d	c	b	a	十六进制数	dp	g	f	e	d	c	b	a	十六进制数
0	1	1	0	0	0	0	0	0	0C0H	0	0	1	1	1	1	1	1	3FH
1	1	1	1	1	1	0	0	1	0F9H	0	0	0	0	0	1	1	0	06H
2	1	0	1	0	0	1	0	0	0A4H	0	1	0	1	1	0	1	1	5BH
3	1	0	1	1	0	0	0	0	0B0H	0	1	0	0	1	1	1	1	4FH
4	1	0	0	1	1	0	0	1	99H	0	1	1	0	0	1	1	0	66H

（续）

显示	共阳									共阴								
	dp	g	f	e	d	c	b	a	十六进制数	dp	g	f	e	d	c	b	a	十六进制数
5	1	0	0	1	0	0	1	0	92H	0	1	1	0	1	1	0	1	6DH
6	1	0	0	0	0	0	1	0	82H	0	1	1	1	1	1	0	1	7DH
7	1	1	1	1	1	0	0	0	0F8H	0	0	0	0	0	1	1	1	07H
8	1	0	0	0	0	0	0	0	80H	0	1	1	1	1	1	1	1	7FH
9	1	0	0	1	0	0	0	0	90H	0	1	1	0	1	1	1	1	6FH
A	1	0	0	0	1	0	0	0	88H	0	1	1	1	0	1	1	1	77H
B	1	0	0	0	0	0	1	1	83H	0	1	1	1	1	1	0	0	7CH
C	1	1	0	0	0	1	1	0	0C6H	0	0	1	1	1	0	0	1	39H
D	1	0	1	0	0	0	0	1	0A1H	0	1	0	1	1	1	1	0	5EH
E	1	0	0	0	0	1	1	0	86H	0	1	1	1	1	0	0	1	79H
F	1	0	0	0	1	1	1	0	8EH	0	1	1	1	0	0	0	1	71H
H	1	0	0	0	1	0	0	1	89H	0	1	1	1	0	1	1	0	76H
L	1	1	0	0	0	1	1	1	0C7H	0	0	1	1	1	0	0	0	38H
P	1	0	0	0	1	1	0	0	8CH	0	1	1	1	0	0	1	1	73H
U	1	1	0	0	0	0	0	1	0C1H	0	0	1	1	1	1	1	0	3EH
Y	1	0	0	1	0	0	0	1	91H	0	1	1	0	1	1	1	0	6EH
灭	1	1	1	1	1	1	1	1	0FFH	0	0	0	0	0	0	0	0	00H

重点提示 这种规定和定义并非是一成不变的，在实际应用中，为了减少走线交叉便于电路板布线，设计者可自行定义8段LED显示器的引脚符号，并根据其工作原理编制相应的“段码表”以及设计与之相匹配的联接电路。

(3)用Verilog描述显示译码器

用Verilog描述的共阳LED显示译码器如下：

```
module decode4_8(seg_out,D);
output[7:0] seg_out;          //段码输出
input[3:0] D;                 //输入的4位BCD码
reg[7:0] seg_out;
always @(D)
  begin
    case(D)
    4'd0: seg_out =8'hc0;
    4'd1: seg_out =8'hf9;
    4'd2: seg_out =8'ha4;
    4'd3: seg_out =8'hb0;
    4'd4: seg_out =8'h99;
    4'd5: seg_out =8'h92;
```

```
        4'd6: seg_out =8'h82;
        4'd7: seg_out =8'hf8;
        4'd8: seg_out =8'h80;
        4'd9: seg_out =8'h90;
        default: seg_out =8'hx;
        endcase
    end
endmodule
```

(4)显示译码器仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 decode4_8。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 decode4_8. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 seg_out0～seg_out7 和 D0～D3,锁定完毕单击 OK 即可,如图 8-36 所示。

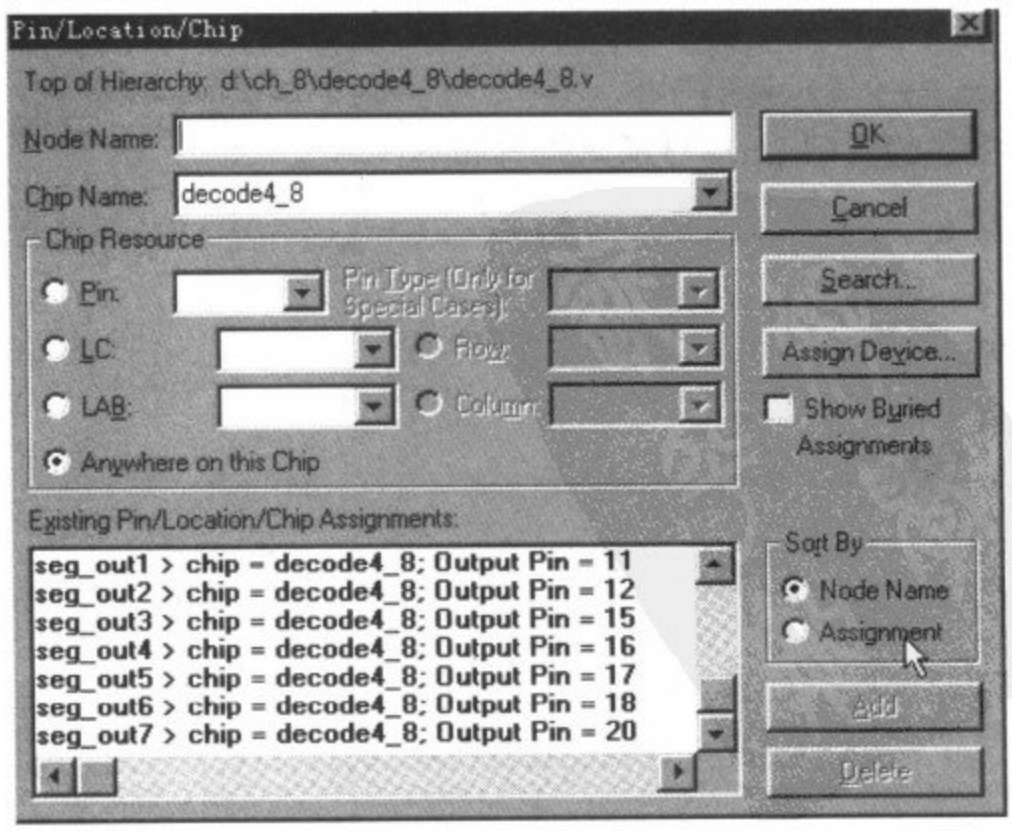


图 8-36　显示译码器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 seg_out0～seg_out9、D0～D3 信号加入 SNF 文件中，如图 8-37 所示。

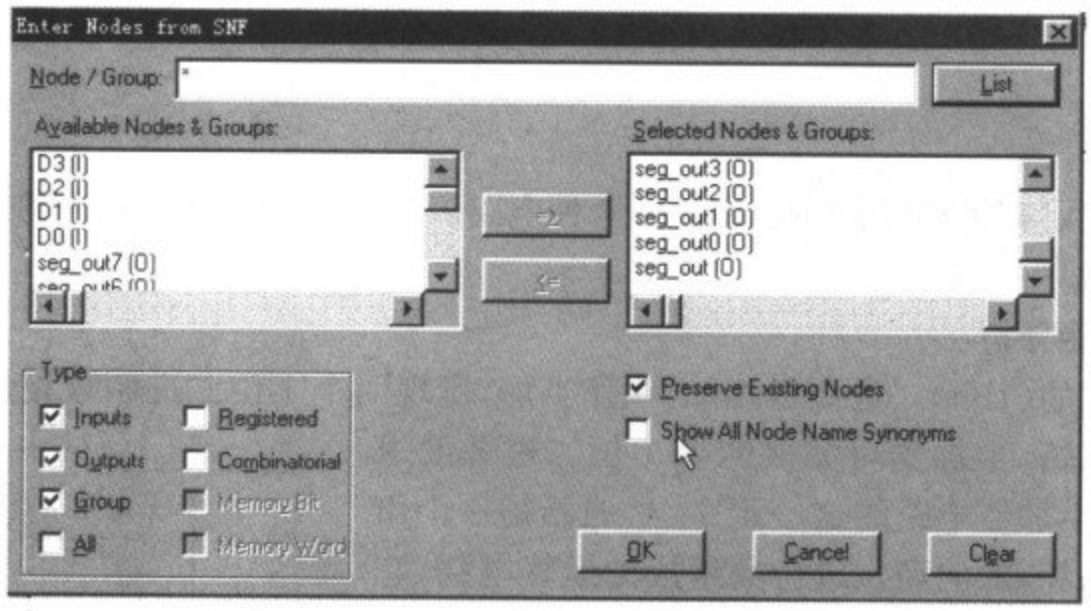


图 8-37　显示译码器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 D[3:0]，点击 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0000，Increment By 增加值设为 0001，Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮，如图 8-38 所示。单击保存按钮，将文件保存为 decode3_8. scf 文件。

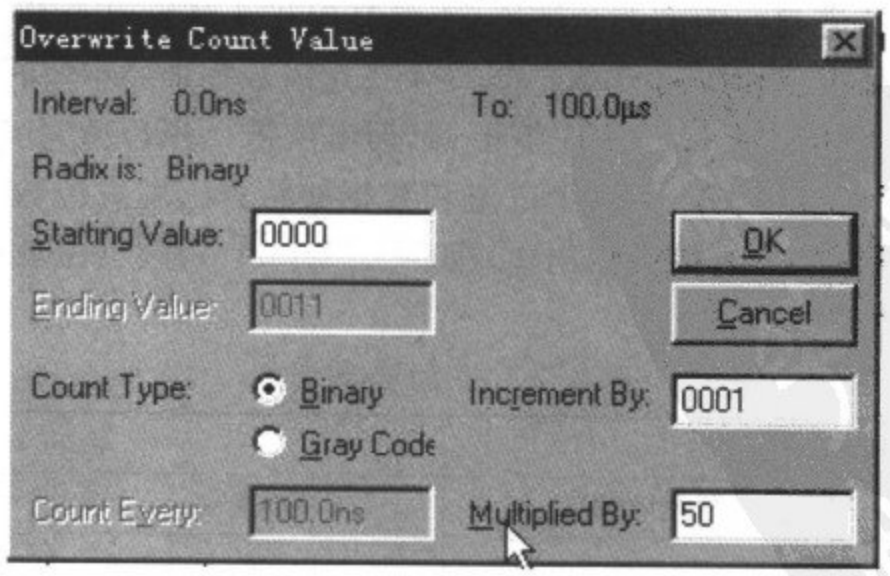


图 8-38　设置 D 输入状态

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-39 所示。

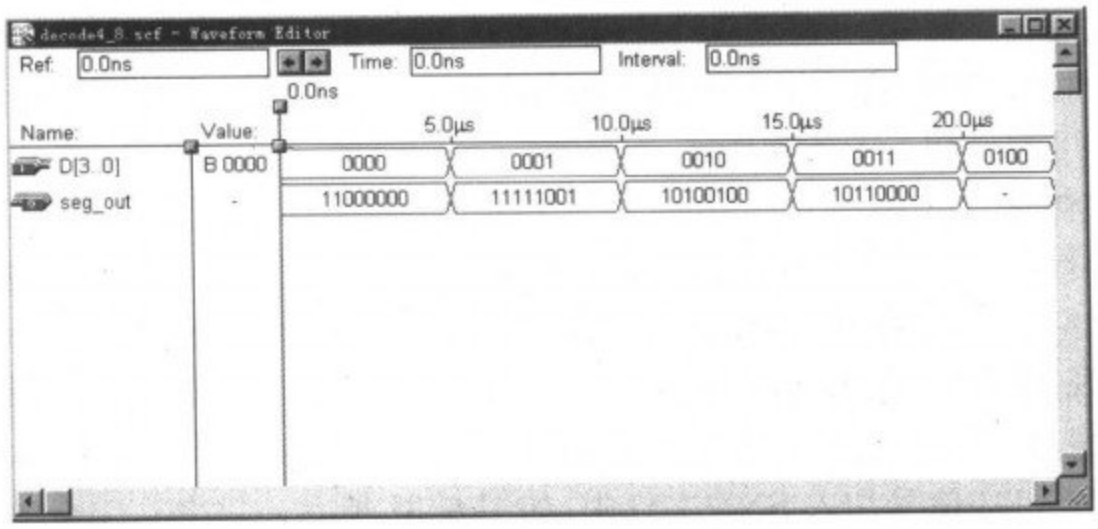


图 8-39　显示译码器仿真波形图

四、加法器

加法器主要有半加法器和全加法器两种电路。它们都是二进制数中的加法运算。

1. 半加器

(1)半加器分析

两个 1 位的(1bit)二进制数相加,叫作半加,实现两个 1 位二进制数相加运算的电路叫做半加器电路,半加器可完成两个 1 位二进数的求和运算,根据半加器电路的这一定义,半加器是一个由加数、被加数、和数、向高位进位数组成的运算电路,它仅考虑本位数相加,而不考虑低位来的进位数。

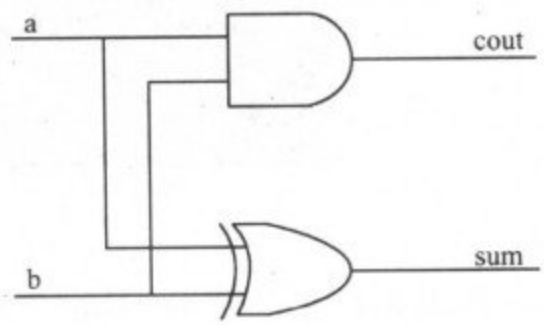


图 8-40　半加器内部逻辑电路

图 8-40 是半加器内部逻辑电路,半加器具有以下特点:

①这种电路共有 4 个端子,其中包括 2 个输入端子和 2 个输出端子。

②输入端 a 和 b 分别是加数输入端和被加数输入端,a 和 b 只有 1 或 0 两个变量,注意这里的 1 和 0 是二进制数中的两个数码,不是高电平 1 和低电平 0。

③输出端 sum 是本位和数输出端子,即 2 个二进制数相加后本位的结果输出,如果是 0+0,本位则是 0;1+0 本位是 1;1+1 应等于 10,但本位是 0,所以此时 sum 端仍然输出 0。

④输出端 cout 是进位数端子,2 个二进制数相加后若出现进位数,如 1+1=10,此时 1 就是进位数,所以此时 cout 端会输出 1。如果是 1+0=1,则进位数是 0。

半加器真值表如表 8-7 所示。

表 8-7　半加器真值表

输 入 端		输 出 端	
加数端 a	被加数端 b	和数端 sum	进位数端 cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(2)用 Verilog 描述半加器

采用门级描述方式描述如下：

```
module half_add(a,b,sum,cout);
input a,b;
output sum,cout;
and (cout,a,b);
xor (sum,a,b);
endmodule
```

采用数据流描述方式描述如下：

```
module half_add(a,b,sum,cout);
input a,b;
output sum,cout;
assign sum=a^b;
assign cout=a&b;
endmodule
```

采用行为描述方式描述如下：

```
module half_add(a,b,sum,cout);
input a,b;
output sum,cout;
reg sum,cout;
always @(a or b)
  begin
    case ({a,b})
    2'b00:begin sum=0; cout=0; end
    2'b01:begin sum=1; cout=0; end
    2'b10:begin sum=1; cout=0; end
    2'b11:begin sum=0; cout=1; end
    endcase
  end
endmodule
```

(3)半加器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 half_add。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 half_add. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 a、b、sum、cout，锁定完毕单击 OK 即可，如图 8-41 所示。

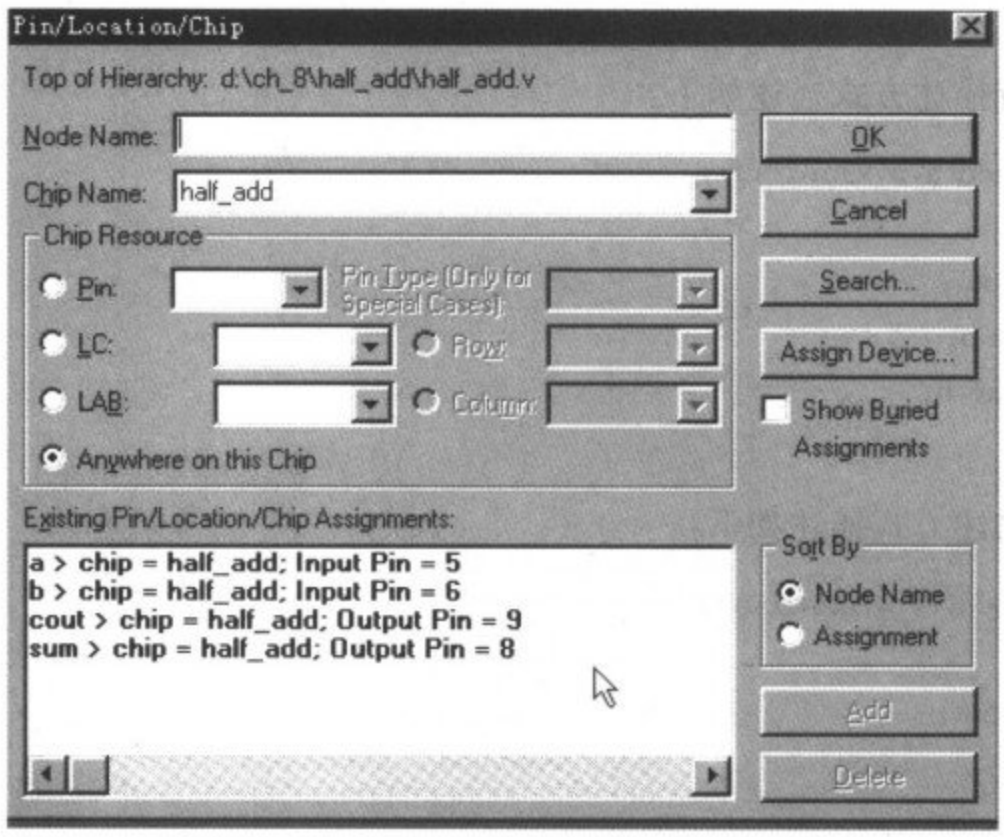


图 8-41　半加器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 a、b、sum、cout 信号加入 SNF 文件中，如图 8-42 所示。

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 a，点击XC按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 50（即半周期为 5μs）。点击 OK 按钮。选中 b，点击XC按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 100（即半周期为 10μs）。点击 OK 按钮。单击保存按钮，将文件保存为 half_add. scf 文件。

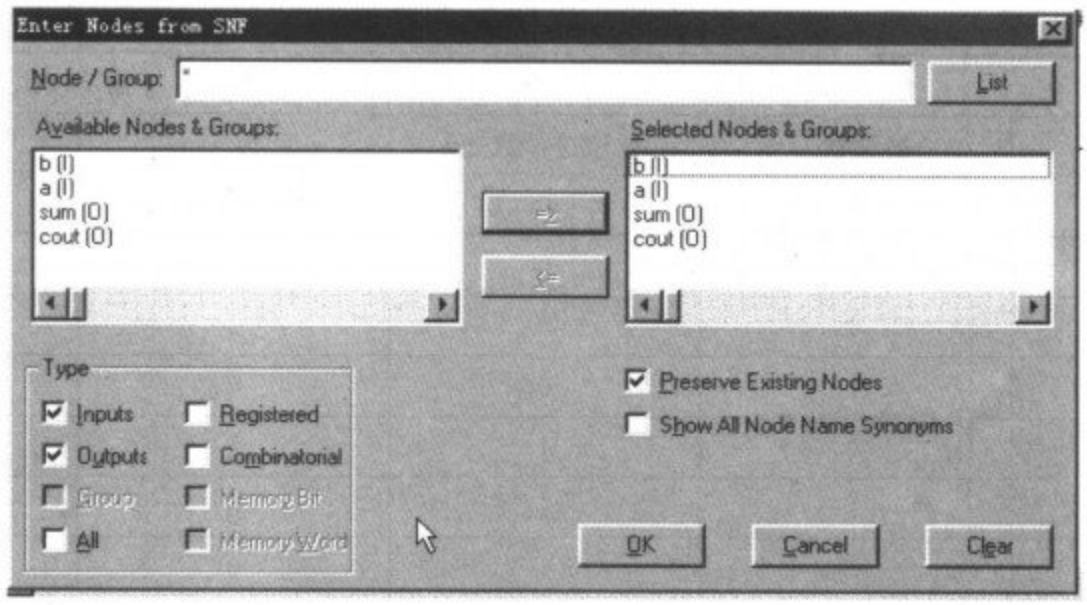


图 8-42　半加器仿真端口列表

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-43 所示。

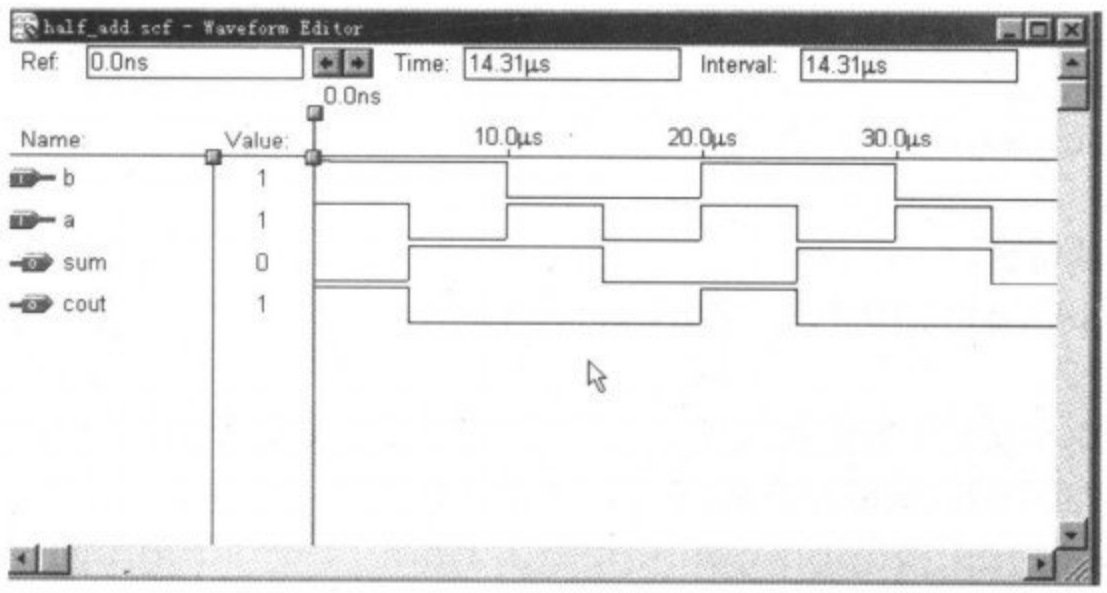


图 8-43　半加器仿真波形图

2. 全加器

(1)全加器分析

前面介绍的半加器只有两个输入端，不能处理由低位送来的进位数，全加器则能够实现二进制全加运算。全加器在对两个二进制数进行加法运算时，除了能将本位的两个数 a、b 相加外，还要加上低位送来的进位数 cin。所以，全加器比半加器电路多一个输入端，共有三个输入端。全加器仍然是一个 1 位加法器电路，与半加器相比只是多了一个低位进位数端。

图 8-44 是全加器电路符号，cin 称为低位进位数端。a 是加数输入端，b 是被加数输入端，sum 是和数输出端，cout 是向高位进位数输出端。

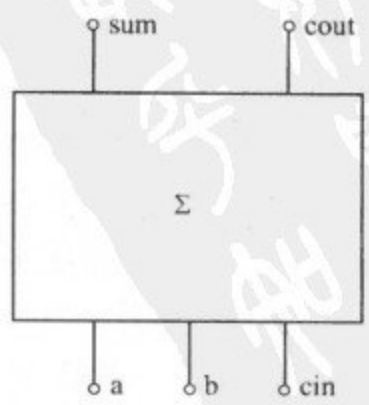


图 8-44　全加器的符号

根据全加器的定义，可得出全加器的真值表，如表 8-8 所示。

表 8-8　全加器真值表

输入端			输出端	
加数 a	被加数 b	低位进位数 cin	和数 sum	进位数 cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

(2)用 Verilog HDL 描述全加器

采用数据流描述方式,用 Verilog HDL 描述全加器如下:

```
module full_add(a,b,cin,sum,cout);
input a,b,cin;
output sum,cout;
assign {cout,sum}=a+b+cin;
endmodule
```

(3)全加器仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 full_add。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 full_add. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 a、b、cin、sum、cout,锁定完毕单击 OK 即可,如图 8-45 所示。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令,弹出波形仿真窗口。在波

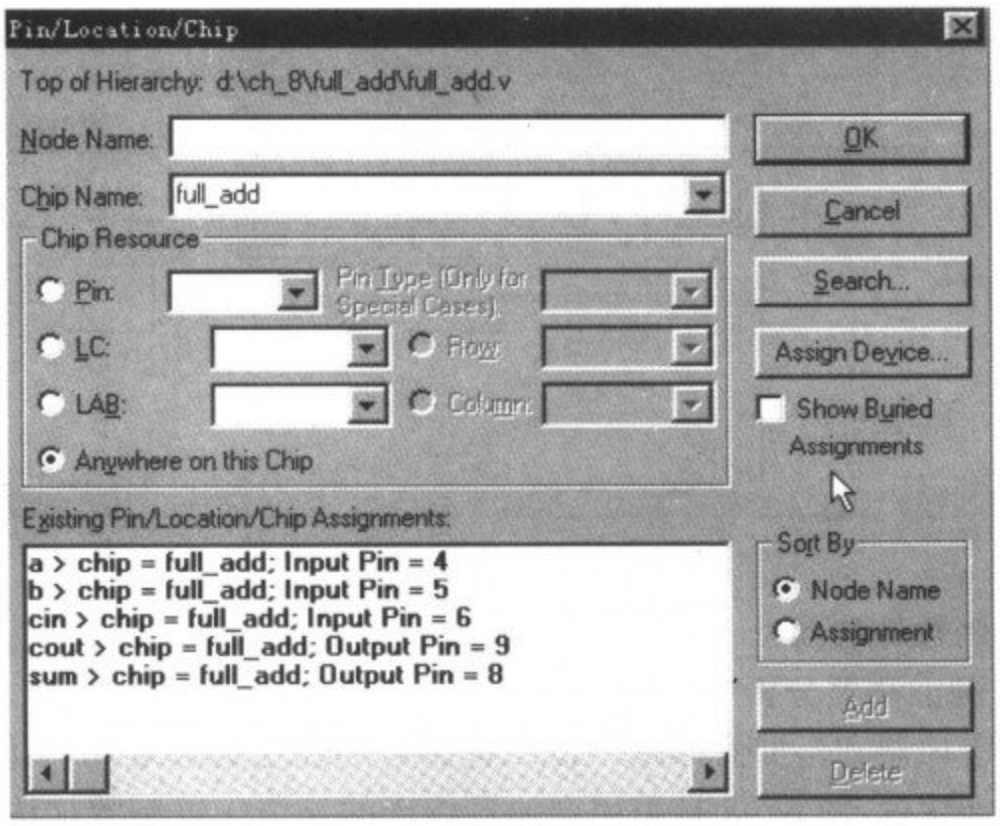


图 8-45　全加器引脚锁定情况

形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 a、b、cin、sum、cout 信号加入 SNF 文件中，如图 8-46 所示。

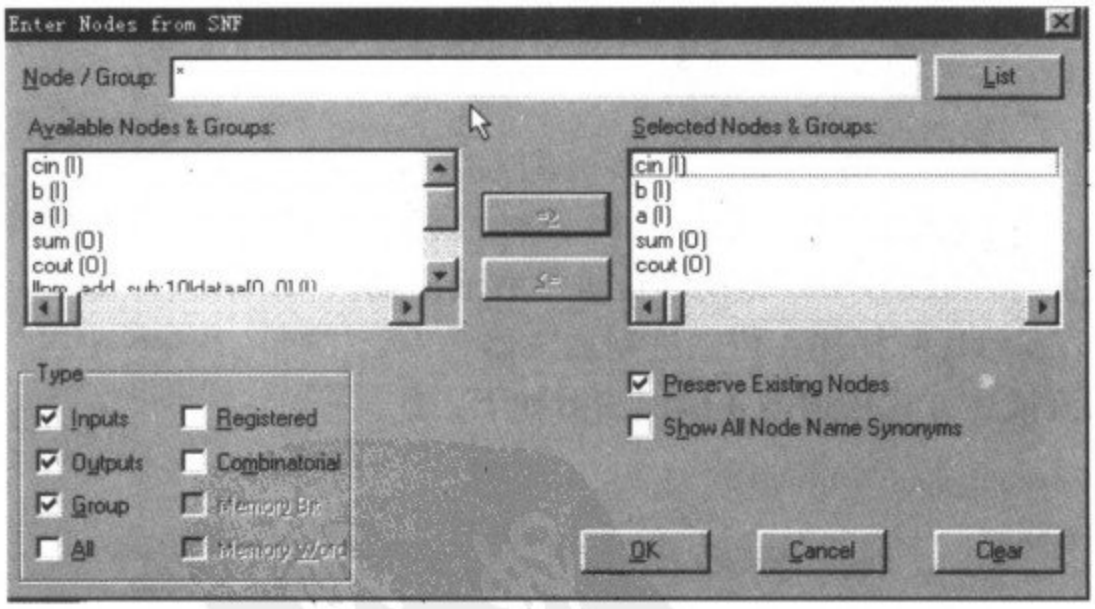


图 8-46　全加器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 a，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 50（即半周期为 5μs）。点击 OK 按钮。选中 b，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 100（即半周期为 10μs）。点击 OK 按钮。点击 cin，将其设为高电平“1”。单击保存按钮，将文件保存为 full_add. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-47 所示。

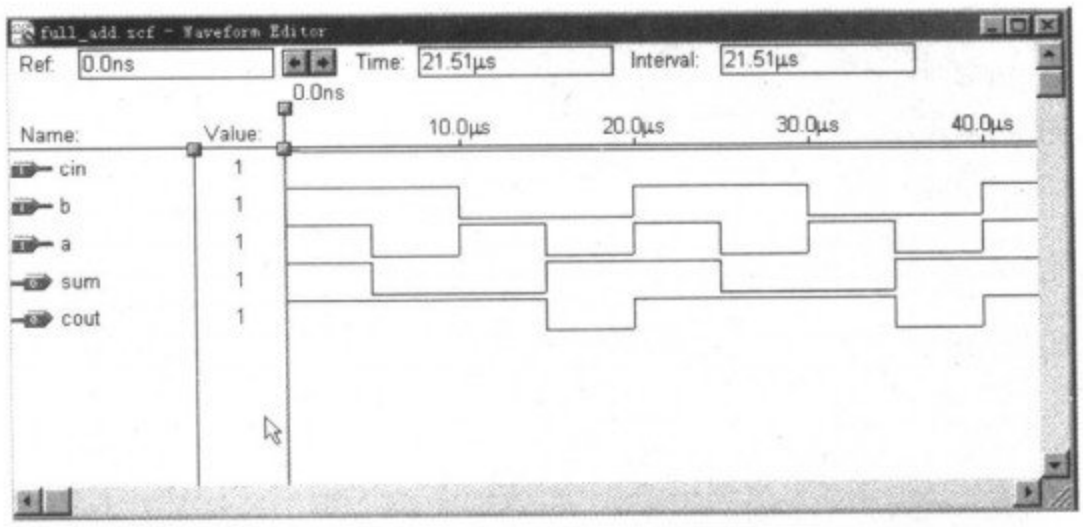


图 8-47　全加器仿真波形图

3. 4 位二进制数加法器

(1)4 位二进制加法器分析

前面介绍的半加器和全加器都是一位二进制数的加法运算电路，4 位二进制数的加法运算要用 4 位加法器，图 8-48 是采用全加器构成的 4 位二进制数加法器，这是一种结构最简单的电路，也称为 4 位串行进位加法器。

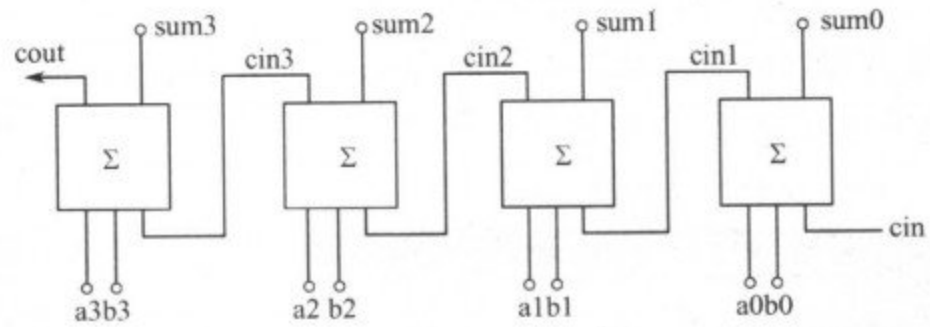


图 8-48　4 位串行进位加法器

电路中，a0～a3 是加数输入端，b0～b3 是被加数输入端，sum1～sum3 是本位和数输出端，cin 是低位进位数端，cout 是向高位进位数输出端。

(2)用 Verilog HDL 描述 4 位二进制加法器

采用数据流描述方式描述的 **4** 位加法器如下：

```
module add_4(cout,sum,a,b,cin);
output[3:0] sum;
output cout;
input[3:0] a,b;
input cin;
assign {cout,sum}=a+b+cin;
endmodule
```

采用行为描述方式，描述的 **4** 位加法器如下：

```
module add_4(cout,sum,a,b,cin);
output[3:0] sum;
output cout;
input[3:0] a,b;
input cin;
```

```
reg[3:0] sum;
reg cout;
always @(a or b or cin)
  begin
  {cout,sum}=a+b+cin;
  end
endmodule
```

(3)4 位二进制加法器仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 add_4。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 add_4. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 a0,a1,a2,a3,b0,b1,b2,b3,cin、sum0,sum1,sum2,sum3,cout,锁定完毕单击 OK 即可,如图 8-49 所示。

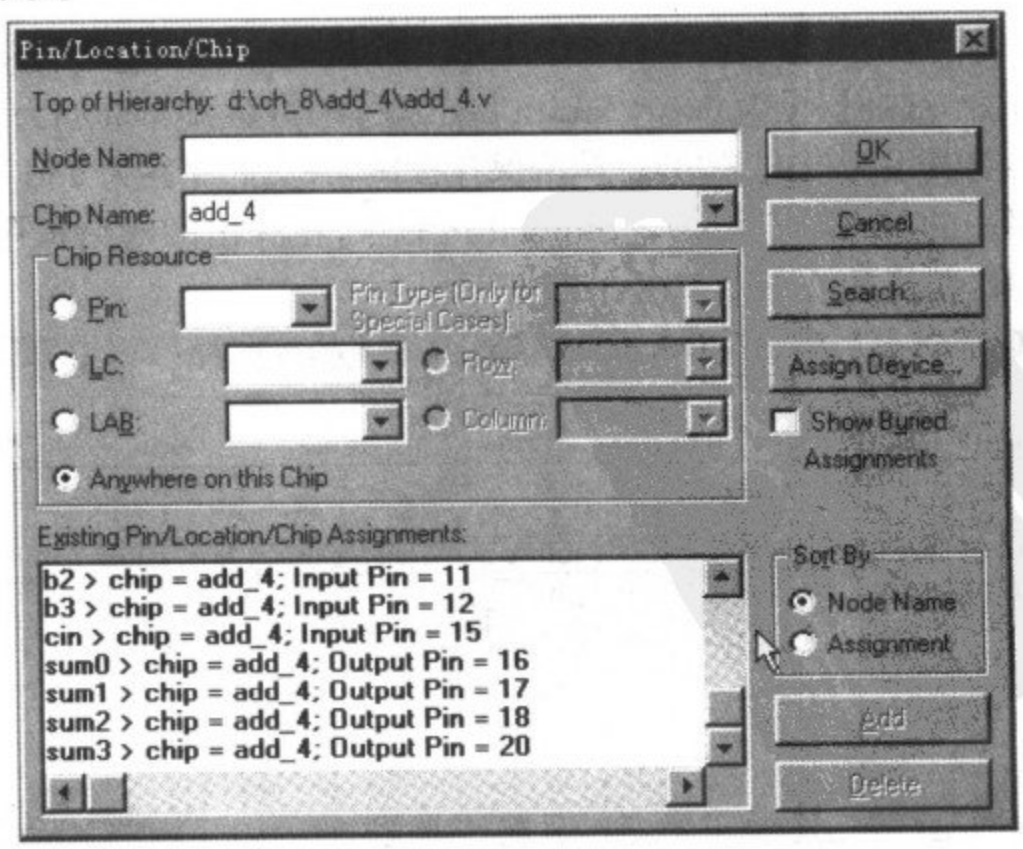


图 8-49　4 位加法器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 a0，a1，a2，a3，b0，b1，b2，b3，cin、sum0，sum1，sum2，sum3，cout 信号加入 SNF 文件中，如图 8-50 所示。

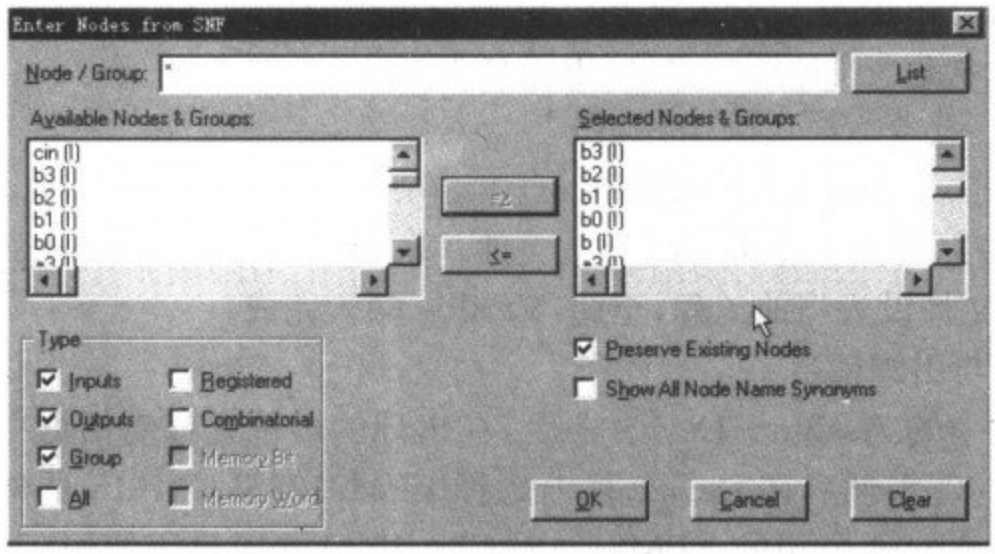


图 8-50　4 位二进制加法器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 a[3:0]，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0000，Increment By 增加值设为 0001，Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮。选中 b[3:0]，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0000，Increment By 增加值设为 0001，Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮。点击 cin，将其设为低电平“0”。单击保存按钮，将文件保存为 add_4. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-51 所示。

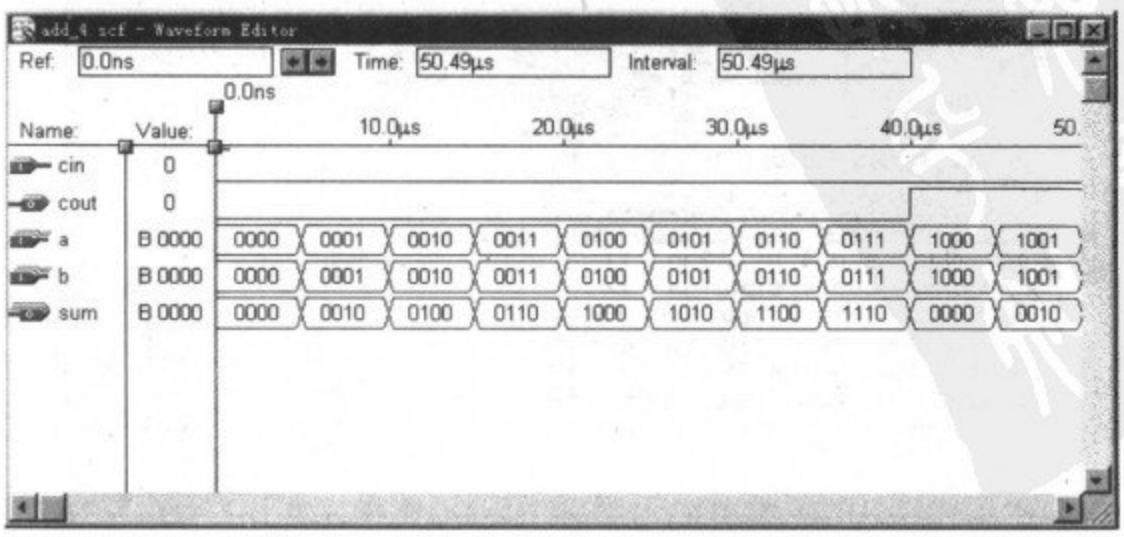


图 8-51　4 位加法器仿真波形图

第三节　双稳态触发器的设计

前面介绍了组合逻辑电路，组合电路的特点是，在任何时刻，电路的输出仅仅取决于当时的输入，它没有记忆功能。双稳态触发器(Flip-Flop)是一种具有记忆功能，可以存储二进制信息的双稳态电路，它是组成时序逻辑电路的基本单元。

双稳态触发器具以以下几个特点：

(1)触发器有两个输出端和多个输入端。触发器的两个输出端，这里用 Q 和 QB(即 $\overline{Q}$)来分别表示，在没有具体说明是哪一个输出端时，通常是指输出端 Q 而不是反相输出端 QB。触发器的输入端对不同类型的触发器其数目不等。

(2)触发器具有记忆功能。各种触发器的基本功能是能够存储二进制码，具有记忆二进制数码的能力。由于触发器具有记忆功能，所以触发器在受输入信号触发后进行工作时，不仅受到输入信号的影响，还要受到触发器本身所记忆数码(即前次触发结果)的影响，这一点与逻辑门电路完全不同。

触发器在输入端触发作用过去后，触发器会保持稳定状态，这说明触发器能够将输入信号保存下来，一个触发器能够保存一位二进制数码信息。

(3)触发器具有两个互为相反的输出端。触发器的两个输出端 Q 和 QB 其输出状态始终相反，当 Q=1 时 QB=0，当 Q=0 时 QB=1。规定触发器的输出端状态是指输出端 Q 的输出状态，即 Q=1 时说成触发器输出高电平 1，Q=0 时就是触发器输出低电平 0。

(4)触发器的两个输出端输出状态必须相反。如果两个输出端的输出状态相同或不是一个输出 1 一个输出 0，都说明触发器已不能进行正常的工作。

(5)触发器的两个稳定状态是可翻转的。触发器有两个稳定状态，即 1 态和 0 态，在没有输入信号作用时，一直稳定在其中的一个状态。触发器的 1、0 两个稳定状态就是表示了二进制码中的 1 与 0。

触发器两个稳定状态是可以发生翻转的，如果触发器稳定在 1 态(Q=1，QB=0)，在输入端有效触发信号的作用下，触发器可以翻转到输出 0 态；同理，如果触发器的输出状态稳定在 0 态，在输入端有效信号的触发下即翻转到 1 态。无论触发器如何翻转，其输出状态都在 1 和 0 之间变化。

下面分别介绍几种常用的触发器及其用 Verilog HDL 描述的方法。

一、RS 触发器

1. 基本 RS 触发器(异步 RS 触发器)

(1)基本 RS 触发器分析

基本 RS 触发器既可以由与门电路组成，也可以由或门电路组成，图 8-52 所示是用两个或非门交叉连接起来构成的基本 RS 触发器。

基本 RS 触发器的逻辑符号如图 8-53 所示。

R、S 是信号输入端，Q、QB 既表示触发器的状态，又是两个互补的信号输出端。

①现态

把触发器接收输入信号之前所处的状态称为现态，并用 Q^n 表示。触发器有两个稳

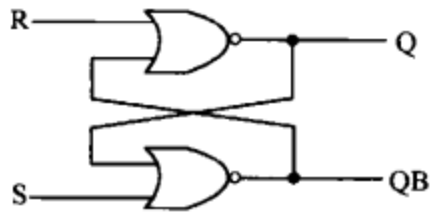


图 8-52　由或非门构成的基本 RS 触发器

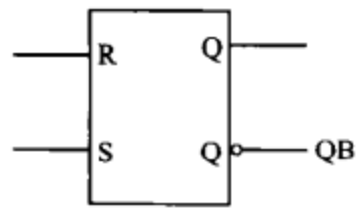


图 8-53　基本 RS 触发器的逻辑符号

定状态，在未接收输入信号或输入信号未到来之前，它总是处在某一个稳态，不是 0 就是 1，也即 Q^n 不是 0 就是 1。

②次态

把触发器接收输入信号之后所处的新的状态叫做次态，并用 Q^{n+1} 表示。大家知道，当输入信号到来时，触发器会根据输入信号的取值更新状态，显然，Q^{n+1} 的值不仅和输入信号有关，而且还取决于现态 Q^n。

③特性表

反映触发器次态 Q^{n+1} 和现态 Q^n 和输入信号之间对应关系的表格叫做特性表。根据基本 RS 触发器工作原理，可得出表 8-9 所示的特性表。

由上表可明显地看出：当 R＝S＝0 时，触发器保持原来状态不变，也即 $Q^{n+1}=Q^n$；当 R＝0、S＝1 时，触发器置 1，也即 $Q^{n+1}=1$；当 R＝1、S＝0 时，触发器置 0，也即 $Q^{n+1}=0$；而 R＝S＝1 是不允许的，属于不用情况。表 8-10 所示是一种常用的特性表的简化形式。

表 8-9　基本 RS 触发器的特性表

R	S	Q^n	Q^{n+1}
0	0	0	0
		1	1
0	1	0	1
		1	1
1	0	0	0
		1	0
1	1	0	不用
		1	

表 8-10　基本 RS 触发器的简化特性表

R	S	Q^{n+1}	说明
0	0	Q^n	保持
0	1	1	置 1
1	0	0	置 0
1	1	不用	不允许

(2)用 Verilog HDL 描述基本 RS 触发器

采用门级描述方式描述如下：

```
module    RS_FF (R,S,Q,QB);
input     R, S;
output    Q, QB;
nor       (Q,R,QB);
nor       (QB,S,Q);
endmodule
```

采用行为描述方式描述如下：

```
module    RS_FF (R,S,Q,QB);
```

```
input     R, S;
output    Q, QB;
reg       Q;
assign    QB = ~Q;
always  @(R or S)
    case ({ R ,S })
    2'b01:Q<=1;
    2'b10:Q<=0;
    2'b11:Q<=1'bx;
    endcase
endmodule
```

(3)基本 RS 触发器的仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 RS_FF。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 RS_FF. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 R,S,Q,QB,锁定完毕单击 OK 即可,如图 8-54 所示。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令,弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标,在弹出的快捷菜单中选择 EnterNodes form SNF... 命令,这时将出现如图仿真信号选择对话框。点击 List,将出现端口列表,你默认是选择全部,你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标,将 R,S,Q,QB 信号加入 SNF 文件中,如图 8-55 所示。

点击菜单命令 File→End Time,终止时间设置为 100μs。

选中 R,点击[redacted]按钮,在弹出的对话框中,将 Starting Value 开始电平设为 1,Increment By 增加值设为 0,Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮。选中

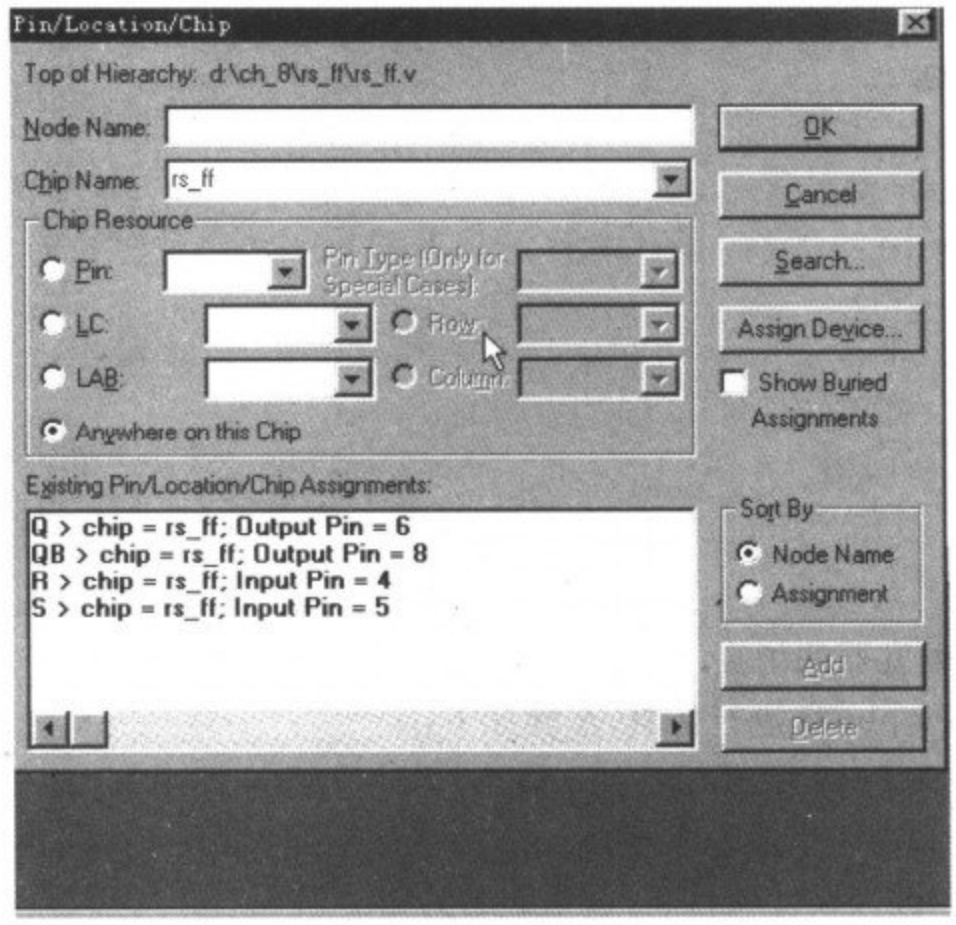


图 8-54　RS 基本触发器引脚锁定情况

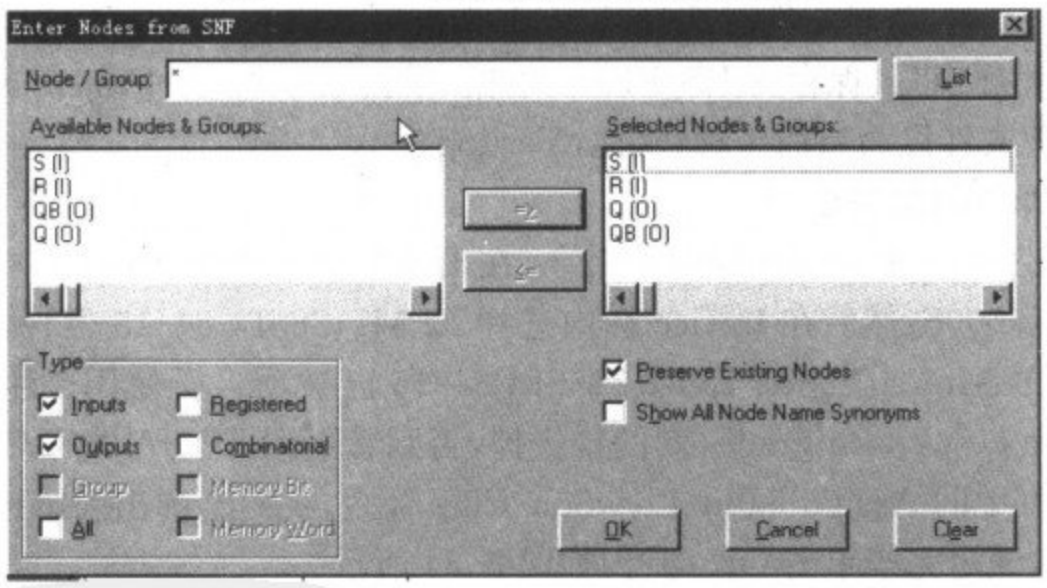


图 8-55　RS 基本触发器仿真端口列表

b，点击XC按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0，Increment By 增加值设为 1，Multiplied By 设为 100（即半周期为 5μs）。点击 OK 按钮。单击保存按钮，将文件保存为 RS_FF. scf 文件。

执行菜单栏中的 MAX＋plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-56 所示。

2. 同步 RS 触发器

(1)同步 RS 触发器分析

同步 RS 触发器中，输入信号是经过控制门输入的，而管理控制门的则是叫做时钟脉冲(CLK 信号)，只有在 CLK 信号到来时，输入信号才能进入触发器，否则就会被拒之门外，对电路不起作用。同步 RS 触发器分为正边沿(上升沿)和负边沿(下降沿)两种触发形式，其逻辑符号如图 8-57 所示。

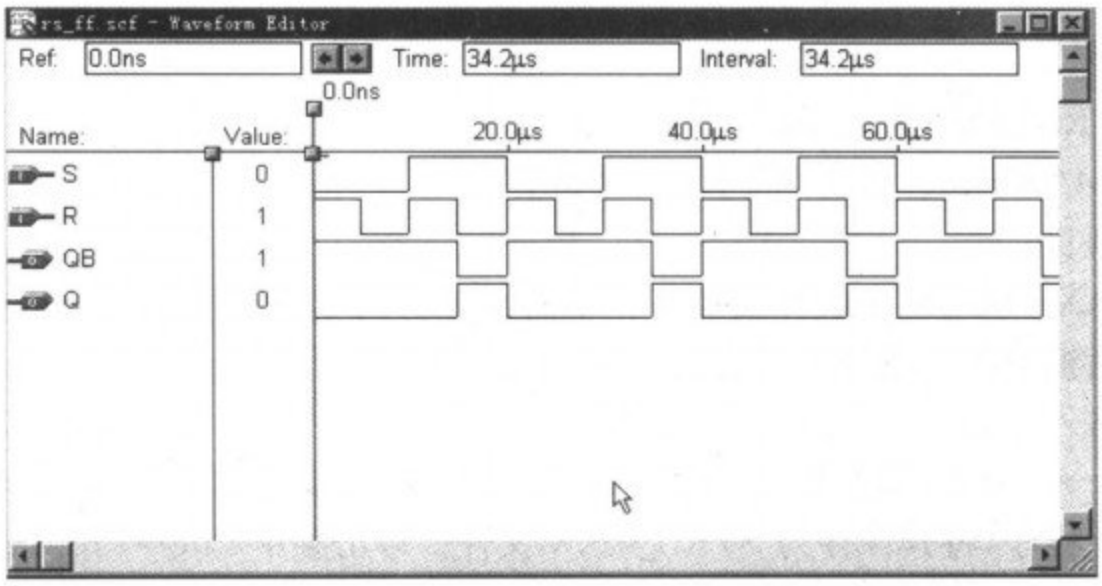


图 8-56　RS 基本触发器仿真波形图

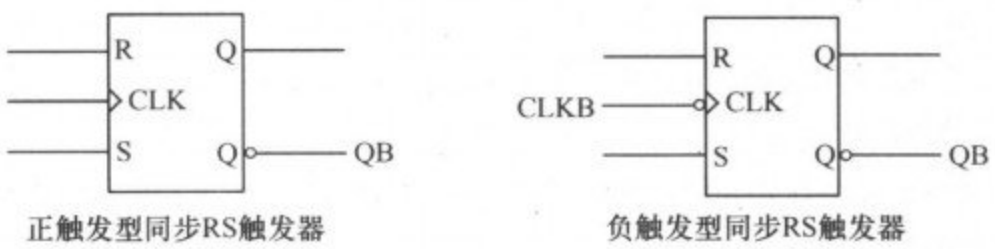


图 8-57　同步 RS 触发器逻辑符号

图中，R 和 S 为输入端，CLK 为时钟端，上升沿或下降沿触发有效，Q 和 QB 是互为反相的输出端。

对于上升沿触发型同步 RS 触发器，只有当时钟脉冲 CLK 的上升沿到来时，电路的输出状态才有可能发生变化，而变化与否又取决于 R 端和 S 端。R 端和 S 端均为低电平时，输出保持电路原有状态；R 端和 S 端分别为高电平和低电平时，输出 Q 为低电平；R 端和 S 端分别为低电平和高电平时，输出 Q 为高电平；R 端和 S 端均为高电平时，输出为“不定”。在实际应用中应避免这种 R 端和 S 端均为高电平的情况出现。

对于下降沿触发型同步 RS 触发器，只有当时钟脉冲 CLK 的下降沿到来时，电路的输出状态才有可能发生变化，变化情况同正触发型。

(2)用 Verilog HDL 描述同步 RS 触发器

用 Verilog HDL 描述同步 RS 触发器(上升沿触发型)如下：

```
module SYRS_FF (R,S,CLK,Q,QB);
input R, S, CLK;
output Q, QB;
reg Q;
assign QB=~Q;
always @(posedge CLK)
    case({R,S})
    2'b01:Q<=1;
    2'b10:Q<=0;
    2'b11:Q<=1'bx;
```

```
    endcase
endmodule
```

(3)同步 RS 触发器的仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 SYRS_FF。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 SYRS_FF. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 R,S,CLK,Q,QB,锁定完毕单击 OK 即可,如图 8-58 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

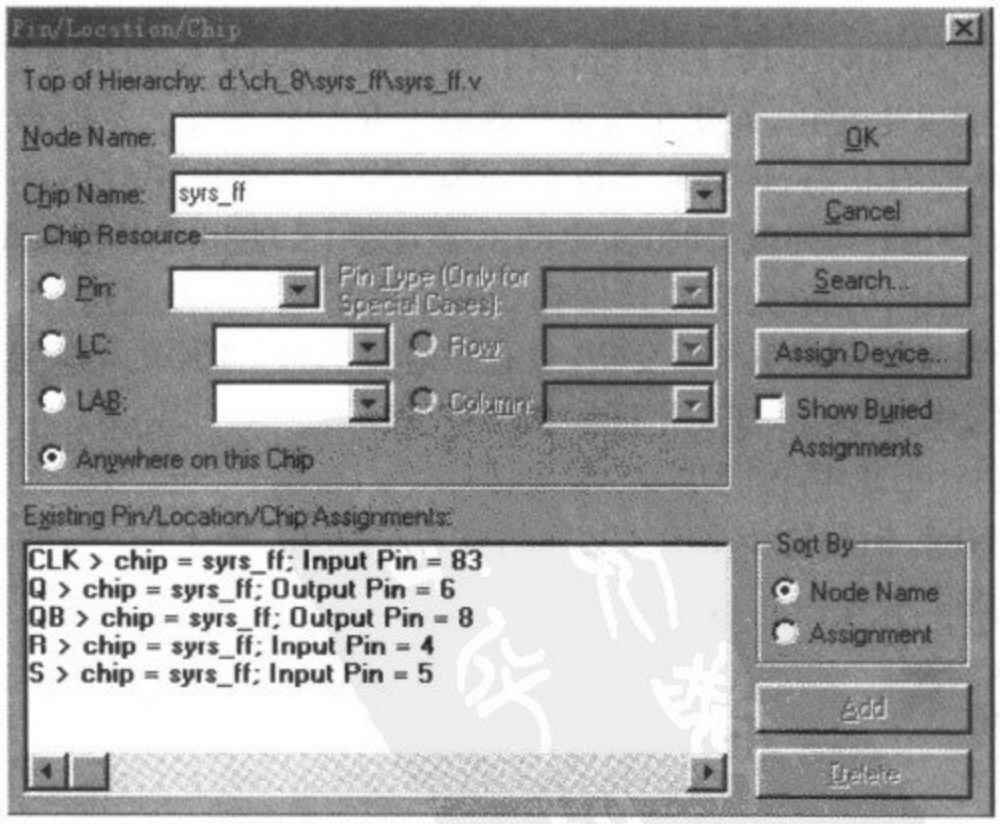


图 8-58　同步 RS 触发器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 R，S，CLK，Q，QB 信号加入 SNF 文件中，如图 8-59 所示。

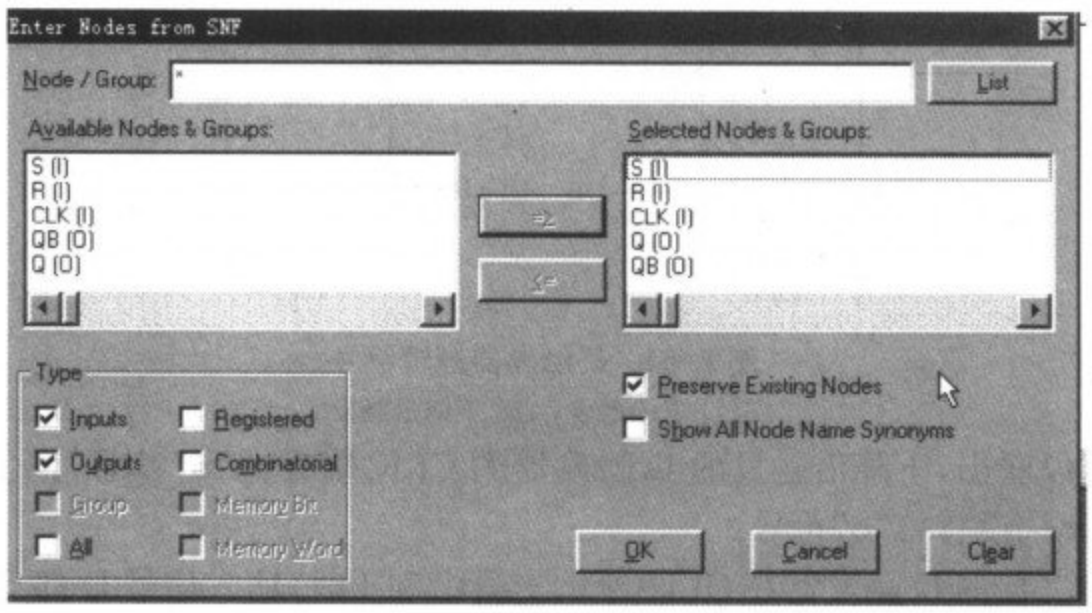


图 8-59　同步 RS 触发器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 R，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮。选中 b，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0，Increment By 增加值设为 1，Multiplied By 设为 100(即半周期为 5μs)。点击 OK 按钮。选中 CLK，点击 XC 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 1，Multiplied By 设为 20(即时钟半周期为 2μs)。单击保存按钮，将文件保存为 SYRS_FF. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-60 所示。

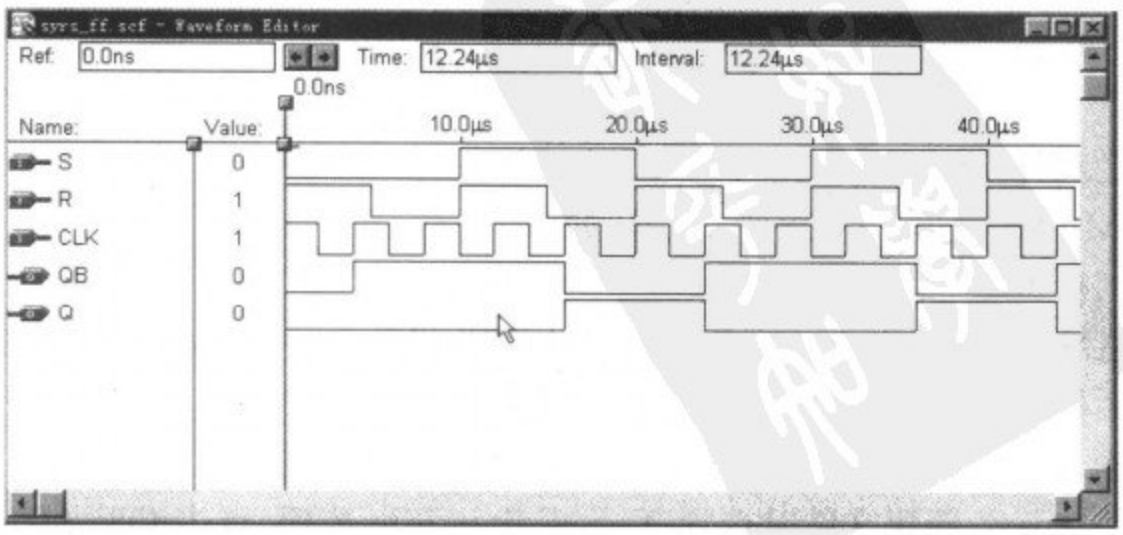


图 8-60　同步 RS 触发器仿真波形图

二、D 触发器

1. D 触发器

(1)D 触发器分析

D 触发器主要有正边沿(上升沿)和负边沿(下降沿)两种触发方式。其逻辑符号如图 8-61 所示。

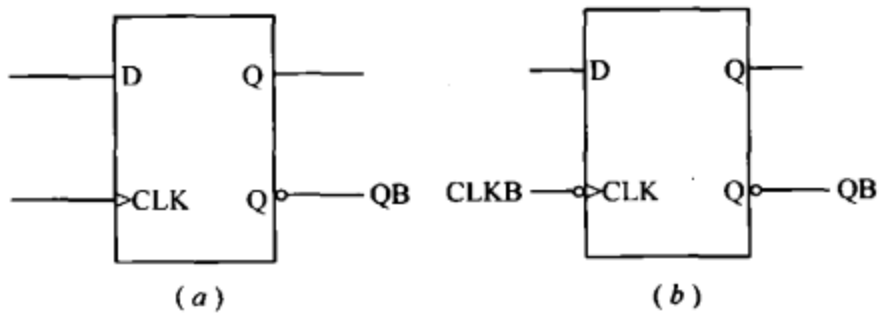


图 8-61　D 触发器逻辑符号

(a) 上升沿触发；(b) 下降沿触发。

从图中可以看出，下降沿触发的 D 触发器的 CLK 端有打“o”符号，而上升沿触发的 D 触发器 CLK 端没有打“o”符号。

下降沿触发器的 D 触发器在 CLK 为 0、上升沿和 1 时，输入信号 D 都不起作用，只有在 CLK 为下降沿时，触发器才按照特性方程 $Q^{n+1}=D$ 更新状态。而上升沿触发器 D 触发器则相反，在 CLK 为 0、下降沿和 1 时，输入信号 D 都不起作用，只有在 CLK 为上升沿时，触发器才按照特性方程 Q^{n+1}=D 更新状态。

(2)用 Verilog HDL 描述 D 触发器

```
module D_FF(D,CLK,Q,QB);
output Q,QB;
input D,CLK;
reg Q;
assign QB=~Q;
always @(posedge CLK)
  begin
  Q<= D;
  end
endmodule
```

(3)D 触发器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 D_FF。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 D_FF. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 D，CLK，Q，QB，锁定完毕单击 OK 即可，如图 8-62 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

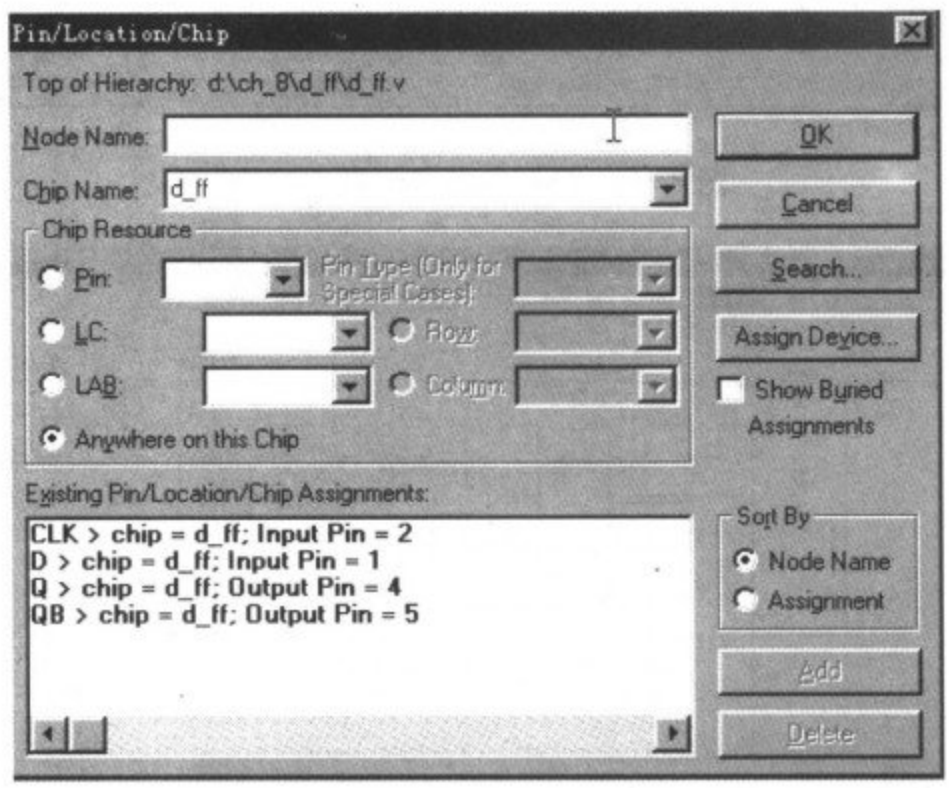


图 8-62　D 触发器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 D，CLK，Q，QB 信号加入 SNF 文件中，如图 8-63 所示。

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 D，点击 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮。选中 CLK，点击 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 1，Multiplied By 设为 20(即时钟半周期为 2μs)。单击保存按钮，将文件保存为 D_FF. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-64 所示。

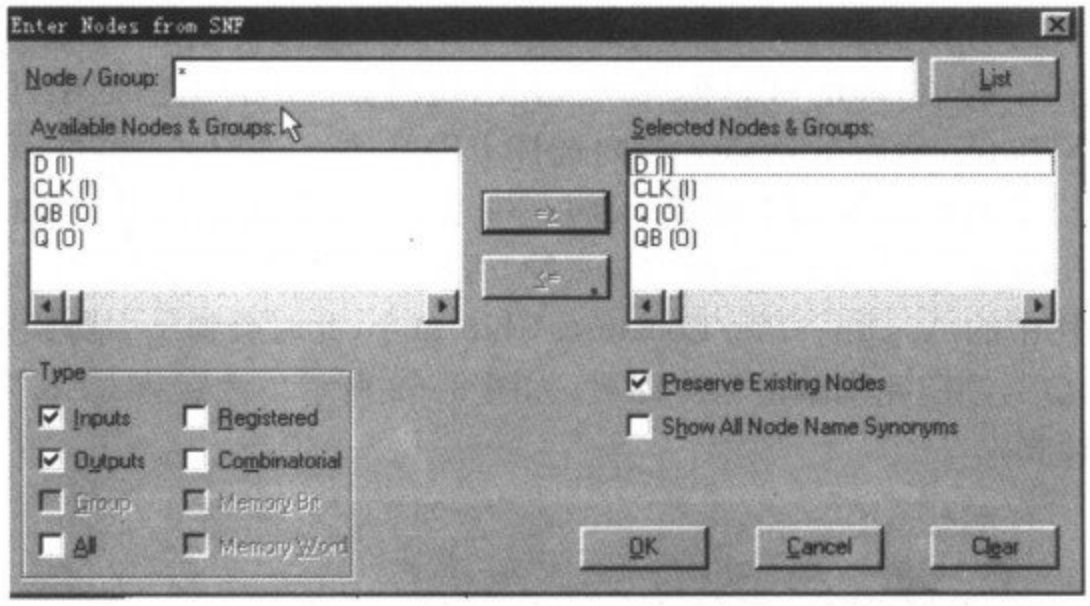


图 8-63　D触发器仿真端口列表

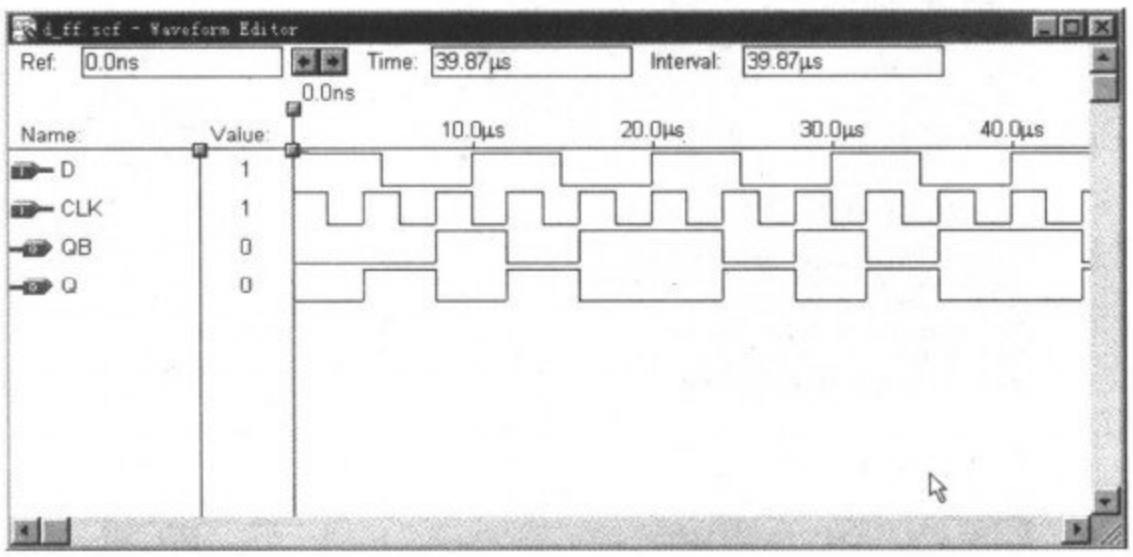


图 8-64　D触发器仿真波形图

2. 带复位功能的 D 触发器

(1)带复位功能的 D 触发器分析

带复位功能的 D 触发器如图 8-65 所示。这里,复位信号 CLR 采用低电平复位,即 CLR 的脚上有一个小圆圈。时钟信号为上升沿触发。

带复位功能的 D 触发器其逻辑功能是:在复位脉冲到来后,将电路初始化为 Q=0 状态。然后,在 D 端的控制下,电路作相应的翻转。

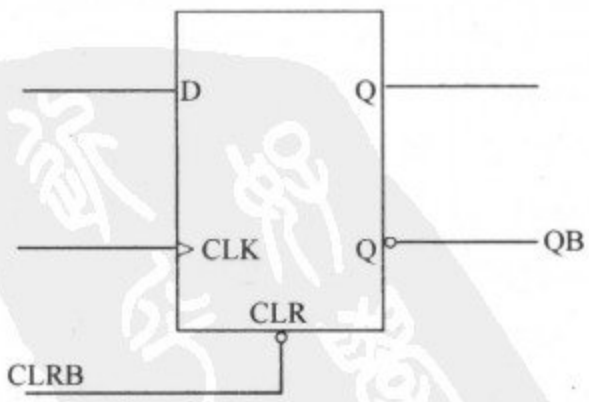


图 8-65　带复位功能的 D 触发器

(2)用 Verilog HDL 描述带复位功能 D 触发器

```
module CLRD_FF(D,CLK,CLRB,Q,QB);
output Q,QB;
input D,CLK,CLRB;
reg Q,QB;
always @(posedge CLK or negedge CLRB)
    begin
    if (! CLRB)
```

```
        begin
        Q<=0;
        QB<=1;
        end
      else
        begin
        Q<=D;
        QB<=~D;
        end
      end
endmodule
```

(3)带复位功能 D 触发器仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 CLRD_FF。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 CLRD_FF 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中:Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 D,CLK,,CLRB,Q,QB,锁定完毕单击 OK 即可,如图 8-66 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令,弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标,在弹出的快捷菜单中选择 EnterNodes form SNF... 命令,这时将出现如图仿真信号选择对话框。点击 List,将出现端口列表,你默认是选择全部,你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标,将 D,CLK,CLRB,Q,QB 信号加入 SNF 文件中,如图 8-67 所示。

点击菜单命令 File→End Time,终止时间设置为 100μs。

选中 D,点击[redacted]按钮,在弹出的对话框中,将 Starting Value 开始电平设为 1,Incre-

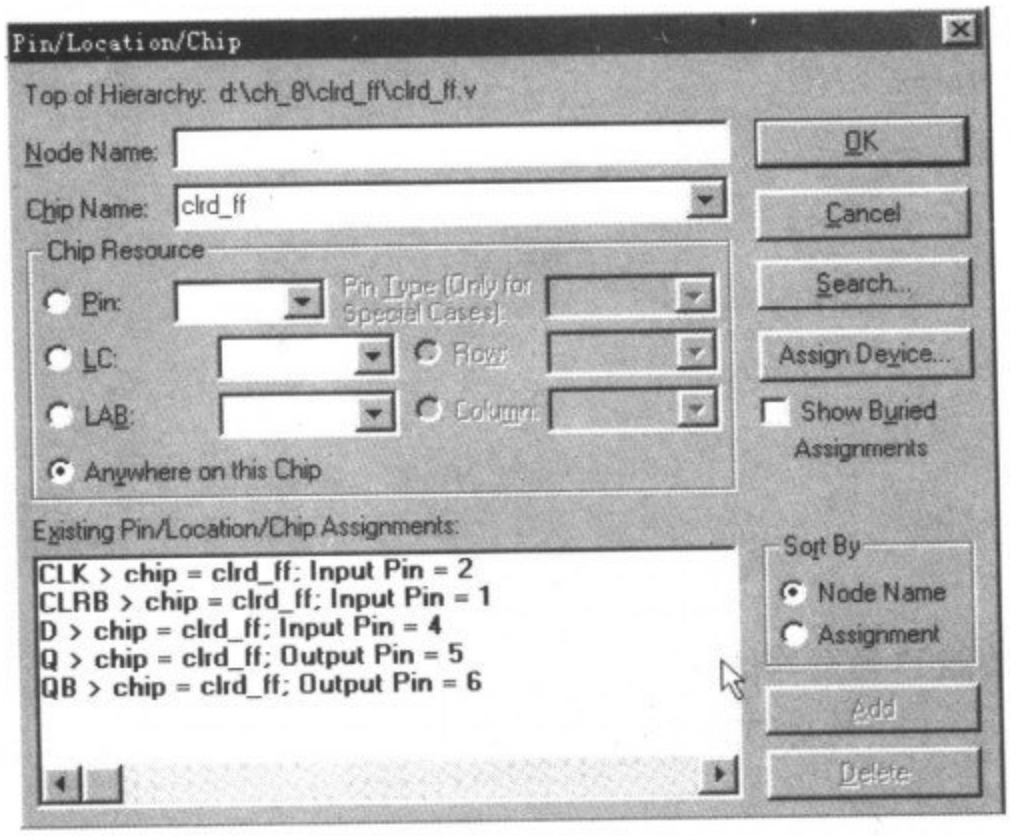


图 8-66　带复位功能 D 触发器引脚锁定情况

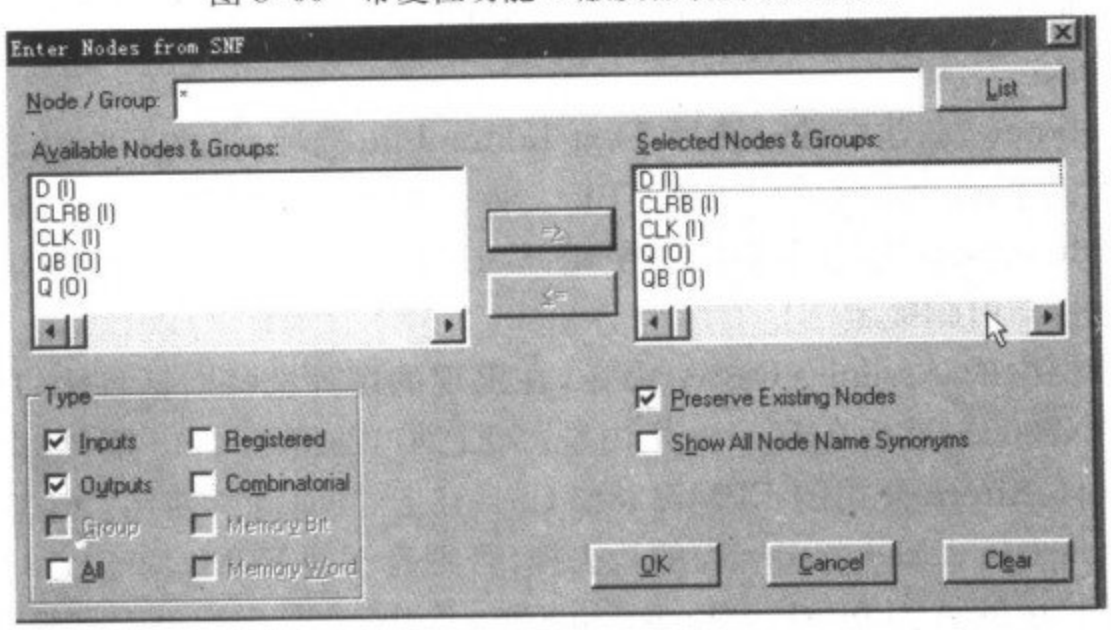


图 8-67　带复位功能 D 触发器仿真端口列表

ment By 增加值设为 0，Multiplied By 设为 50(即半周期为 5μs)。点击 OK 按钮。选中 CLK，点击 XC 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 1，Multiplied By 设为 20(即时钟半周期为 2μs)。选中 CLRB，用拖动的方法将开始的 10μs 设置为低电平，其他时段设置为高电平，单击保存按钮，将文件保存为 CLRD_FF. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-68 所示。

三、JK 触发器

1. 基本 JK 触发器

(1)基本 JK 触发器分析

基本 JK 触发器有负边沿(下降沿)触发和正边沿(上升沿)触发两种触发方式。其逻辑符号如图 8-69 所示。

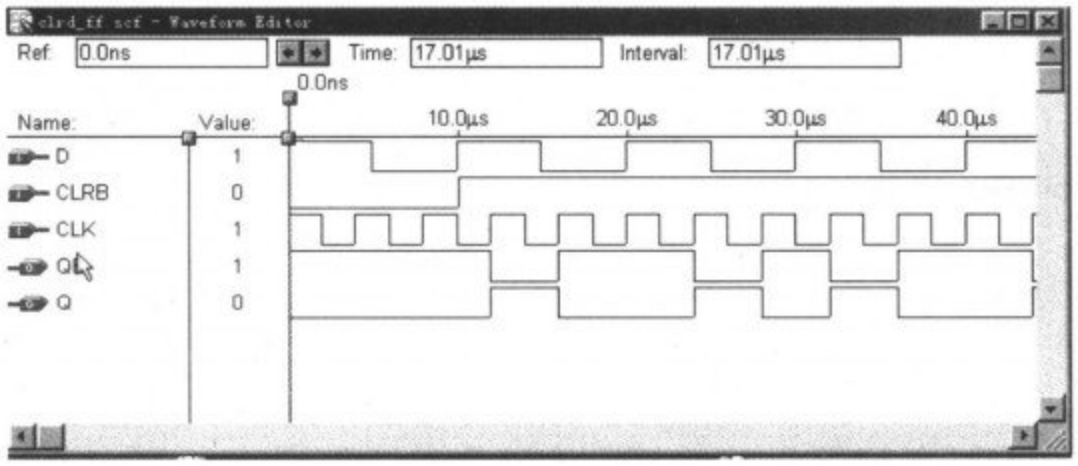


图 8-68　带复位功能 D 触发器仿真波形图

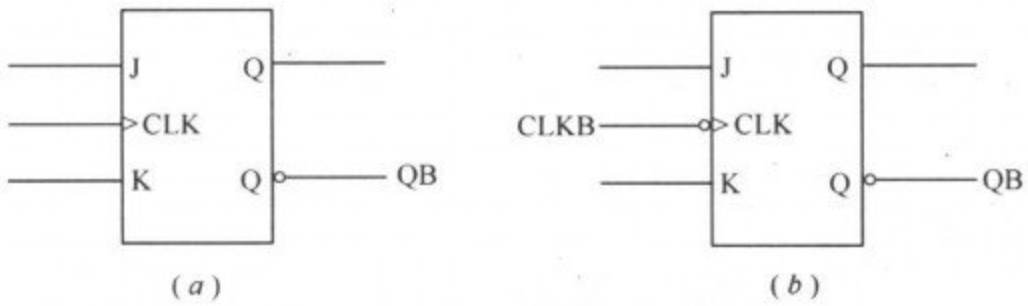


图 8-69　JK 触发器的逻辑符号

(*a*) 正触发 JK 触发器逻辑符号；(*b*) 负触发 JK 触发器逻辑符号。

从图中可以看出，负边沿触发的 JK 触发器的 CLK 端有打“o”符号，而正边沿触发的 JK 触发器 CLK 端没有打“o”符号。

基本 JK 触发器的特性方程为 $Q^{n+1}=J\overline{Q^n}+\overline{K}Q^n$，JK 触发器的特性表如表 8-11 所示。

表 8-11　JK 触发器特性表

J	K	Q^n	Q^{n+1}	说明
0	0	0	0	保持
0	0	1	1	
0	1	0	0	同步置 0
0	1	1	0	
1	0	0	1	同步置 1
1	0	1	1	
1	1	0	1	翻转
1	1	1	0	

负边沿 JK 触发器在 CLK 为 0、上升沿和 1 时，输入信号 J、K 都不起作用，只有在 CLK 为下降沿时，触发器才按照特性方程 $Q^{n+1}=J\overline{Q^n}+\overline{K}Q^n$ 更新状态。而正边沿 JK 触发器则相反，在 CLK 为 0、下降沿和 1 时，输入信号 J、K 都不起作用，只有在 CLK 为上升沿时，触发器才按照特性方程 $Q^{n+1}=J\overline{Q^n}+\overline{K}Q^n$ 更新状态。

(2)用 Verlilog HDL 描述基本 JK 触发器

用 Verlilog HDL 描述基本 JK 触发器(上升沿触发)如下：

```
module JK_FF(CLK,J,K,Q,QB);
input CLK,J,K;
```

```
output Q,QB;
reg Q;
assign QB=~Q;
always @(posedge CLK)
    begin
      case({J,K})
      2'b00 : Q <= Q;
      2'b01 : Q <= 1'b0;
      2'b10 : Q <= 1'b1;
      2'b11 : Q <= ~Q;
      default: Q<= 1'bx;
      endcase
    end
endmodule
```

(3)基本 JK 触发器仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 JK_FF。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 JK_FF. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 J,K,CLK,Q,QB,锁定完毕单击 OK 即可,如图 8-70 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令,弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标,在弹出的快捷菜单中选择 EnterNodes form SNF... 命令,这时将出现如图仿真信号选择对话框。点击 List,将出现端口列表,你默认是选择全部,你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标,将 J,K,CLK,Q,QB 信号加入 SNF 文件中,如图 8-71 所示。

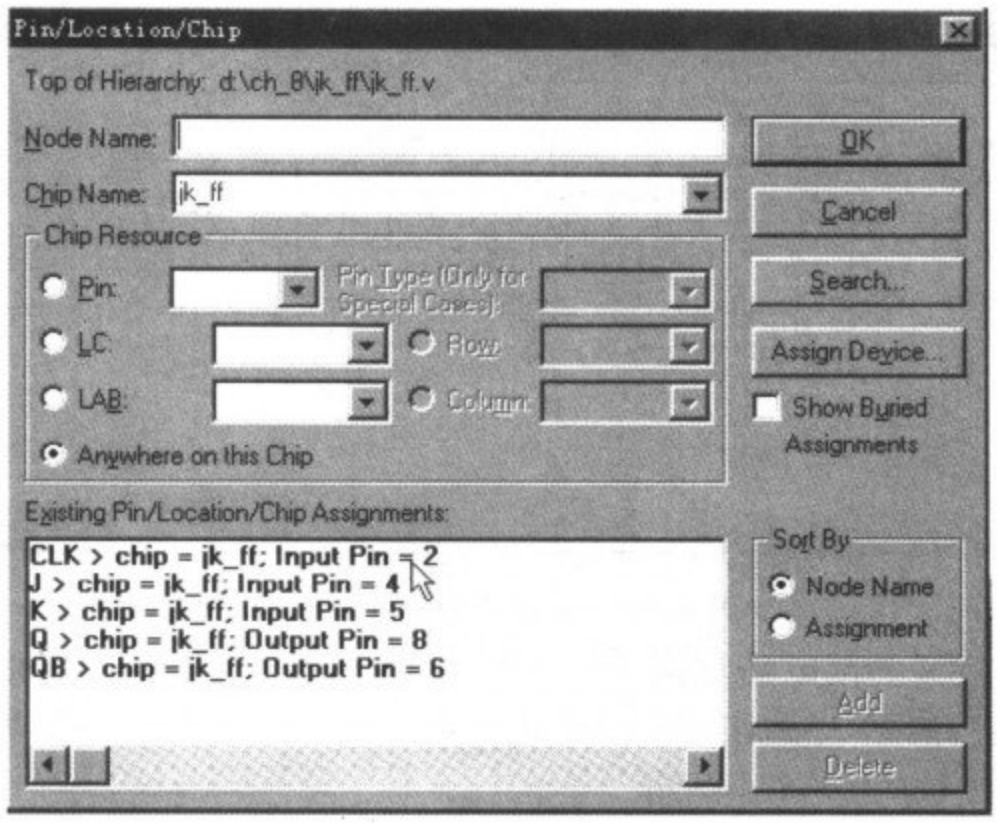


图 8-70　JK 触发器引脚锁定情况

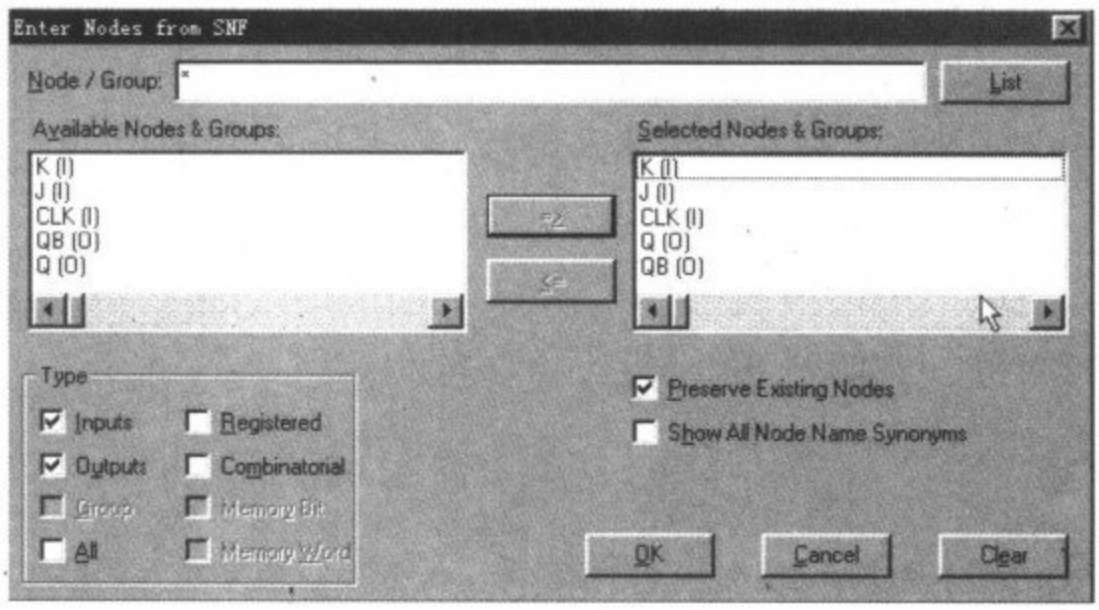


图 8-71　JK 触发器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 J，点击 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 50（即半周期为 5μs）。点击 OK 按钮。选中 K，点击 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 100（即半周期为 10μs）。点击 OK 按钮。选中 CLK，点击 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 1，Multiplied By 设为 20（即时钟半周期为 2μs）。单击保存按钮，将文件保存为 JK_FF. scf 文件。

执行菜单栏中的 MAX＋plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-72 所示。

2. 带复位功能的 JK 触发器

(1)带复位功能的 JK 触发器分析

带复位功能的 JK 触发器逻辑符号如图 8-73 所示。这里，将时钟 CLK 定义为上升

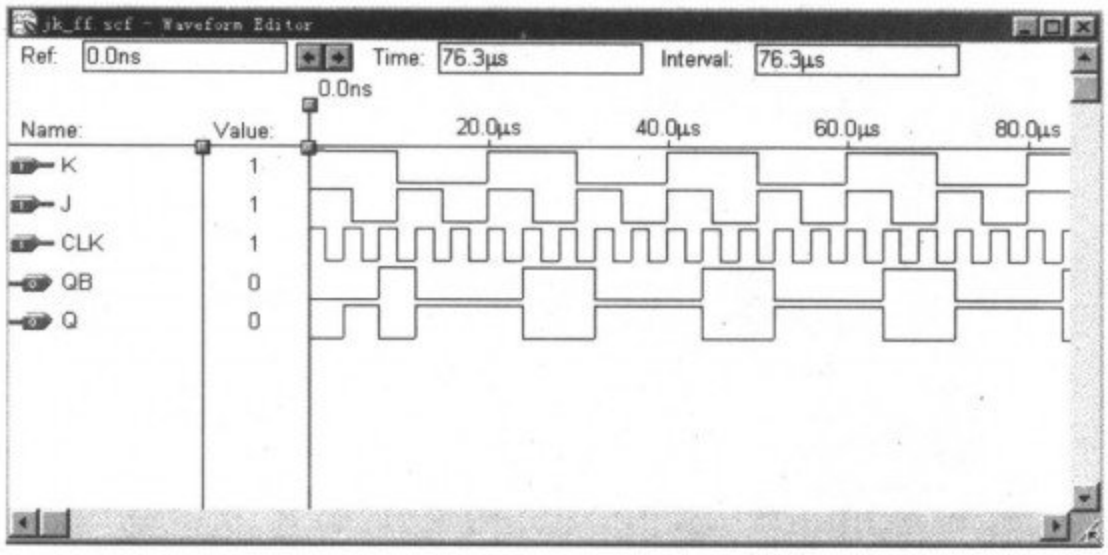


图 8-72　JK 触发器仿真波形图

沿触发;将 CLR 定义为高电平复位,即 CLR 为高电平时,将电路初始化为 Q=0 状态。然后,在 JK 端的控制下,电路作相应的翻转。当然,也可以将 CLR 定义为低电平复位,此时,需在 CLR 的脚上加上一个小圆圈。

(2)用 Verlilog HDL 描述带复位功能 JK 触发器

用 Verlilog HDL 描述带复位功能 JK 触发器如下:

```
module CLRJK_FF(CLK,CLR,J,K,Q,QB);
input CLK,CLR,J,K;
output Q,QB;
reg Q;
assign QB=~Q;
always @(posedge CLK or posedge CLR)
  begin
      if(CLR)
          begin
          Q<=0;
          end
      else
          case({J,K})
          2'b00:Q<=Q;
          2'b01:Q<=1'b0;
          2'b10:Q<=1'b1;
          2'b11:Q<=~Q;
          default: Q<=1'bx;
          endcase
      end
endmodule
```

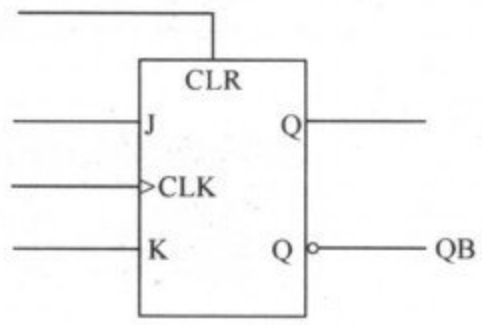


图 8-73　带复位功能的 JK 触发器逻辑符号

(3)带复位功能 JK 触发器仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 CLRJK_FF。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 CLRJK_FF. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 J，K，，CLR，CLK，Q，QB，锁定完毕单击 OK 即可，如图 8-74 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

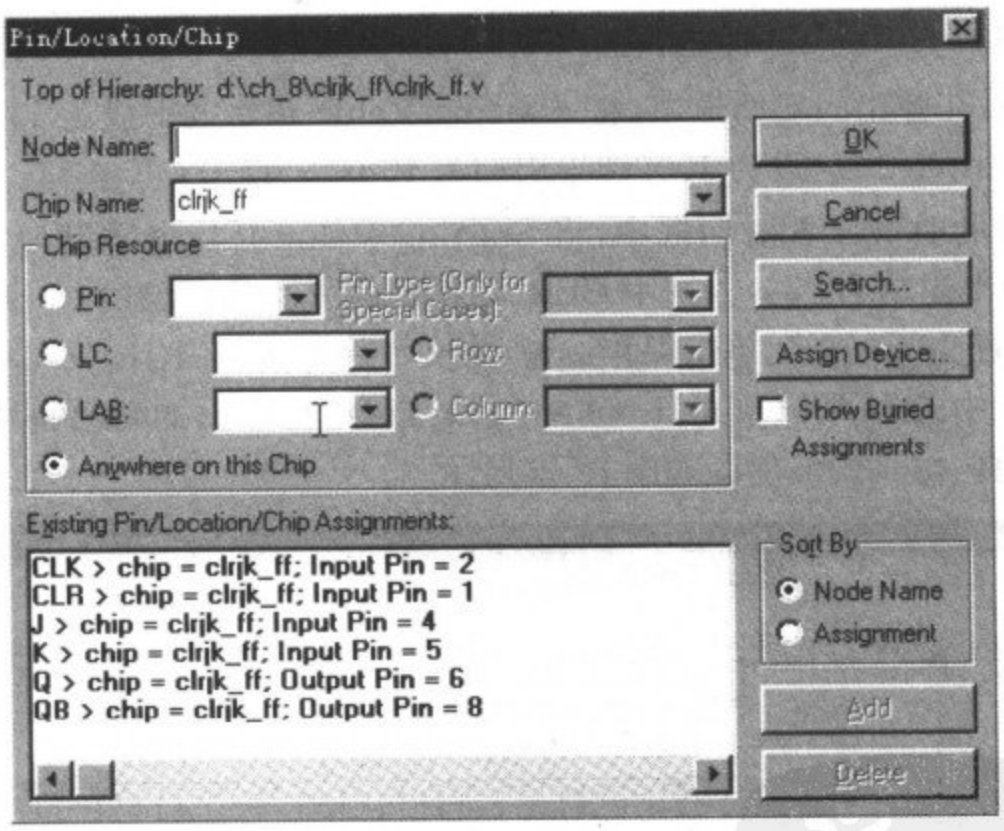


图 8-74　带复位功能 JK 触发器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 J，K，CLR，CLK，Q，QB 信号加入 SNF 文件中，如图 8-75 所示。

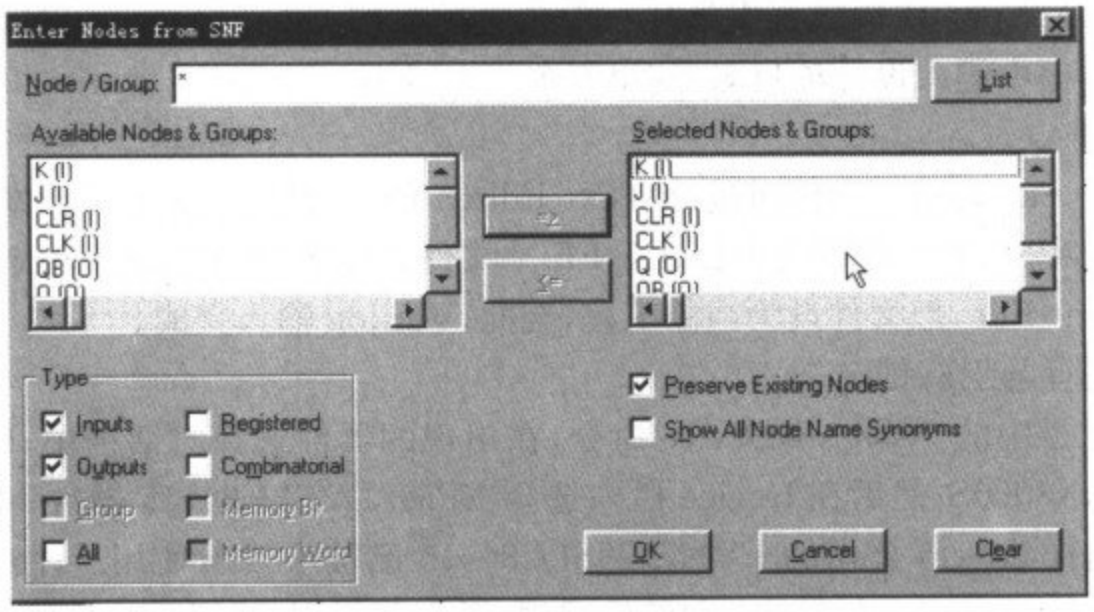


图 8-75　带复位功能 JK 触发器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 J，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 50（即半周期为 5μs）。点击 OK 按钮。选中 K，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 100（即半周期为 10μs）。点击 OK 按钮。选中 CLK，点击 XC 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 1，Multiplied By 设为 20（即时钟半周期为 2μs）。选中 CLR，用拖动的方法将开始的 0～10μs 设置为高电平，其余时段设置为低电平。单击保存按钮，将文件保存为 CLRJK_FF. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-76 所示。

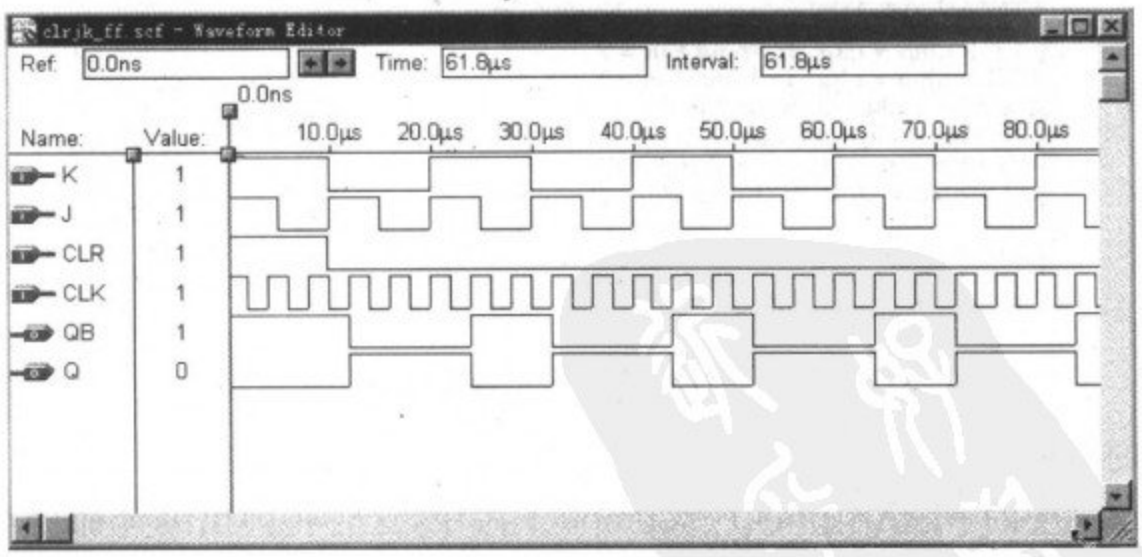


图 8-76　带复位功能 JK 触发器仿真波形图

四、T 触发器

1. T 触发器分析

在时钟脉冲操作下，根据输入信号 T 取值的不同，凡是具有保持和翻转功能的电路，即当 T=0 时能保持状态不变，T=1 时翻转的电路，都称之为 T 型时钟触发器，简称为 T 型触发器或 T 触发器。

图 8-77 所示是带有时钟的同步 T 触发器的逻辑符号(不带时钟信号称为异步 T 触发器),T 是信号输入端,CLK 是时钟脉冲端,T 触发器的特性方程为 $Q^{n+1}=\overline{T}Q^n+T\overline{Q^n}$,特性表如表 8-12 所示。

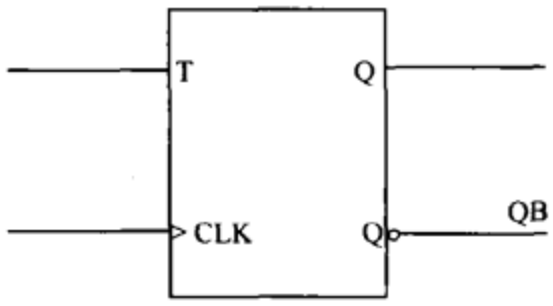


图 8-77 T 触发器逻辑符号

表 8-12 T 型触发器特性表

T	Q^n	Q^{n+1}	说明
0	0	0	保持
0	1	1	
1	0	1	翻转
1	1	0	

2. 用 Verilog HDL 描述 T 触发器

用 Verilog HDL 描述 T 触发器如下:

```
module    T_FF (T,CLK,Q,QB);
input     T,CLK;
output    Q, QB;
reg       Q;
assign    QB=~Q;
always    @(posedge CLK)
  if(T) Q<=~Q;
endmodule
```

3. T 触发器仿真

下面用 MAX+plusII 进行仿真,步骤如下:

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 T_FF。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 T_FF. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 T,CLK,Q,QB,锁定完毕单击 OK 即可,如图 8-78 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击

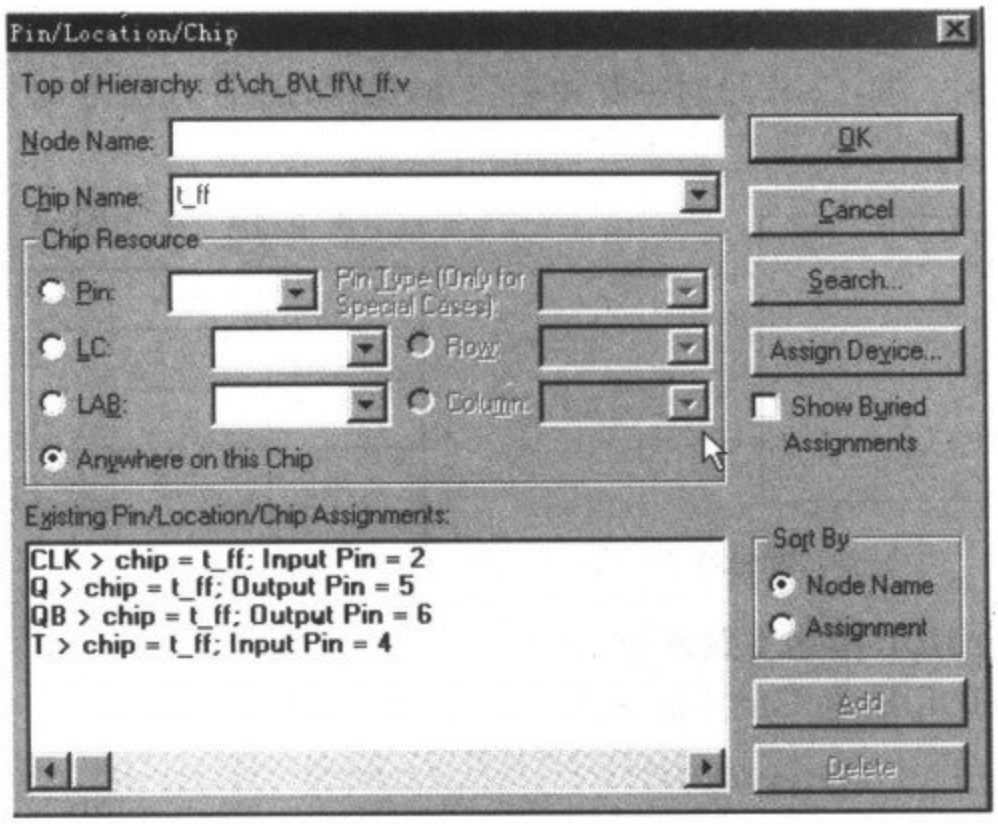


图 8-78　T 触发器引脚锁定情况

Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 T，CLK，Q，QB 信号加入 SNF 文件中，如图 8-79 所示。

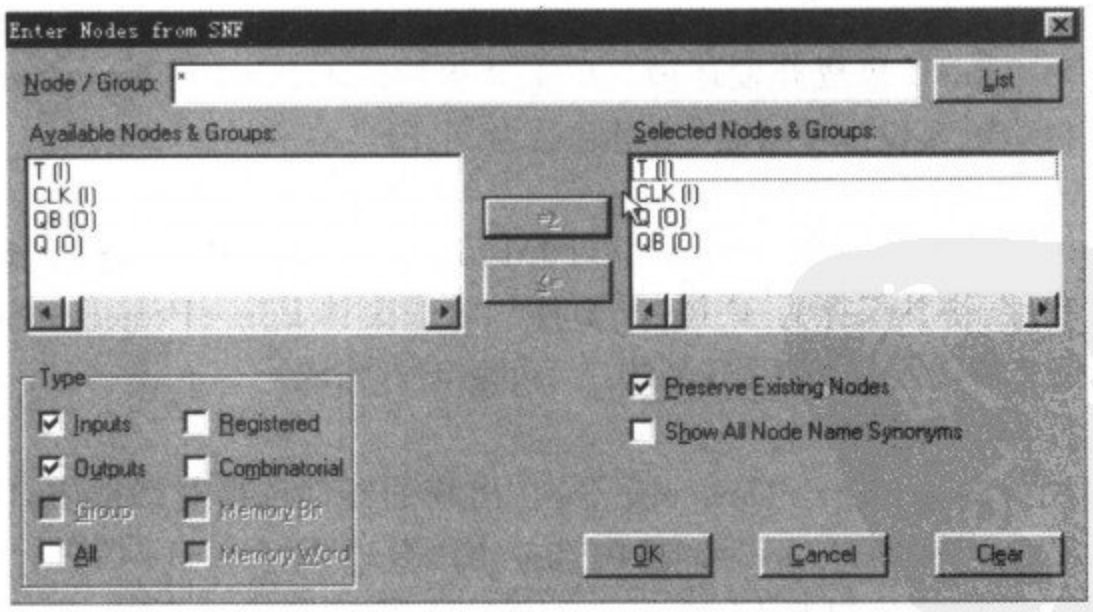


图 8-79　T 触发器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 T，点击 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 1，Increment By 增加值设为 0，Multiplied By 设为 100(即半周期为 10μs)。点击 OK 按钮。选中 CLK，点击 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 1，Multiplied By 设为 30(即时钟半周期为 3μs)。单击保存按钮，将文件保存为 T_FF. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-80 所示。

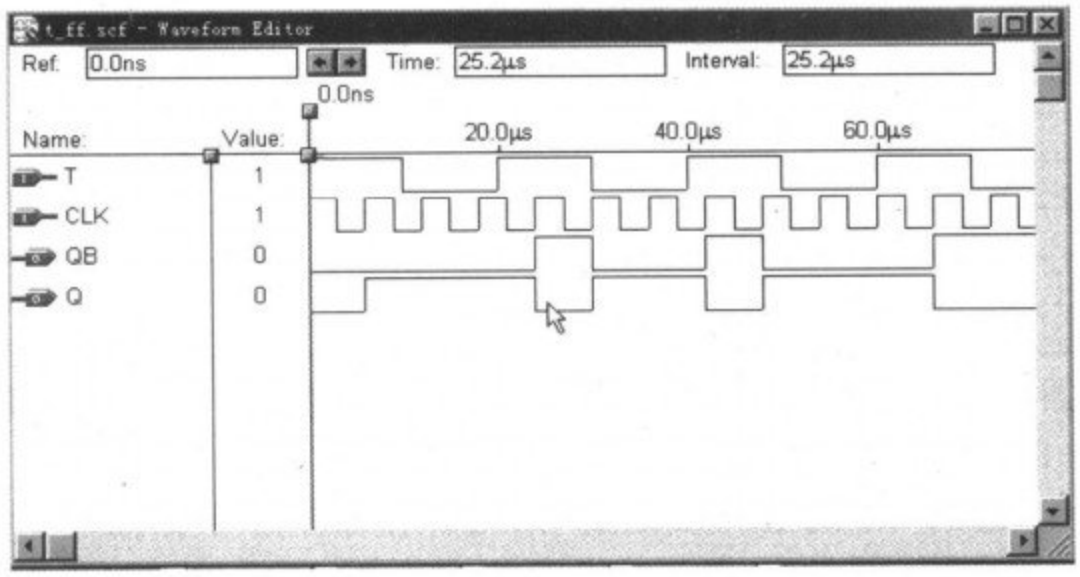


图 8-80　T 触发器仿真波形图

第四节　时序逻辑电路的设计

在数字电路中，凡是任一时刻的稳定输出不仅取决于该时刻的输入，而且还和电路原来的状态有关的电路称为时序逻辑电路，简称时序电路。前面介绍的触发器就是最简单的时序逻辑电路。

一、寄存器

把二进制数据或代码暂时存储起来的操作叫做寄存。具有寄存功能的电路称为寄存器。寄存器是一种基本时序电路，在各种数字系统中，几乎是无所不在。因为任何现代数字系统，都必须把需要处理的数据、代码先寄存起来，以便随时取用。

寄存器具有以下特点：从电路组成看，寄存器是由具有存储功能的触发器组合起来构成的，使用的可以是基本触发器、同步触发器、主从触发器或边沿触发器，电路结构比较简单。从基本功能看，寄存器的任务主要是暂时存储二进制数据或者代码，一般情况下，不对存储内容进行处理，逻辑功能比较单一。

寄存器一般可分成两大类：一类是基本寄存器，一类是移位寄存器。

1. 基本寄存器

(1)基本寄存器分析

对于基本寄存器，数据或代码只能并行送入寄存器中，需要时也只能并行输出。如图 8-81 所示是 4 位寄存器的内部逻辑电路。

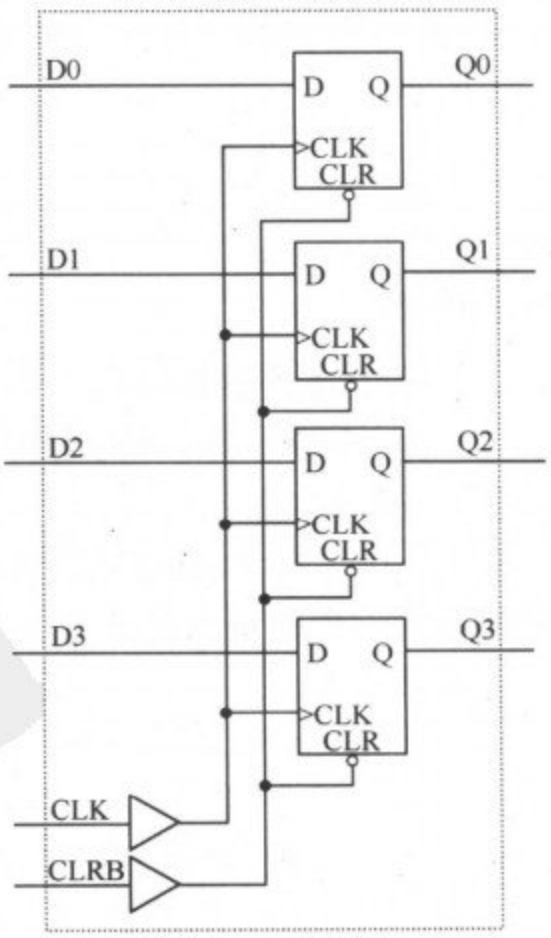


图 8-81　4 位寄存器内部逻辑电路

图中，D0～D3 是并行数据输入端，Q0～Q3 是并

行数据输出端，CLK 是时钟控制端，上升沿触发，CLRB 是复位清零端，低电平复位。4 位寄存器的状态表如表 8-13 所示。

表 8-13　4 位寄存器状态表

输入						输出				说明
CLRB	CLK	D0	D1	D2	D3	Q0	Q1	Q2	Q3	
0	×	×	×	×	×	0	0	0	0	清零
1	↑	D0	D1	D2	D3	D0	D1	D2	D3	送数
1	1	×	×	×	×					保持
1	0	×	×	×	×					

在往寄存器中寄存数据或代码之前，必须先将寄存器清零，否则有可能出错。只要 CLRB=0，4 个 D 触发器都复位到零状态。

当 CLRB=1 时，CLK 的上升沿送数，无论寄存器中原来存储的数据是什么，在 CLRB=1 时，只要送数时钟 CLK 的上升沿到来，加在并行输入端 D0～D3 的数码马上就被送入寄存器中。输出数据可以并行从 Q0～Q3 端引出。

当 CLRB=1、CLK 上升沿以外的时间，寄存器保持内容不变，即各个输出端的状态与输入的数据无关。

(2)用 Verilog HDL 描述 4 位基本寄存器

用 Verilog HDL 描述 4 位基本寄存器如下：

```
module REG_4(CLRB,CLK,D,Q);
output[3:0] Q;
input[3:0] D;
input CLK,CLRB;
reg[3:0] Q;
always  @(posedge CLK or negedge CLRB)
  begin
    if(! CLRB) Q<=0;
    else       Q<=D;
  end
endmodule
```

(3)4 位基本寄存器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 REG_4。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 REG_4. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family

栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 D0,D1,D2,D3,CLK,CLRB,Q0,Q1,Q2,Q3,锁定完毕单击 OK 即可,如图 8-82 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

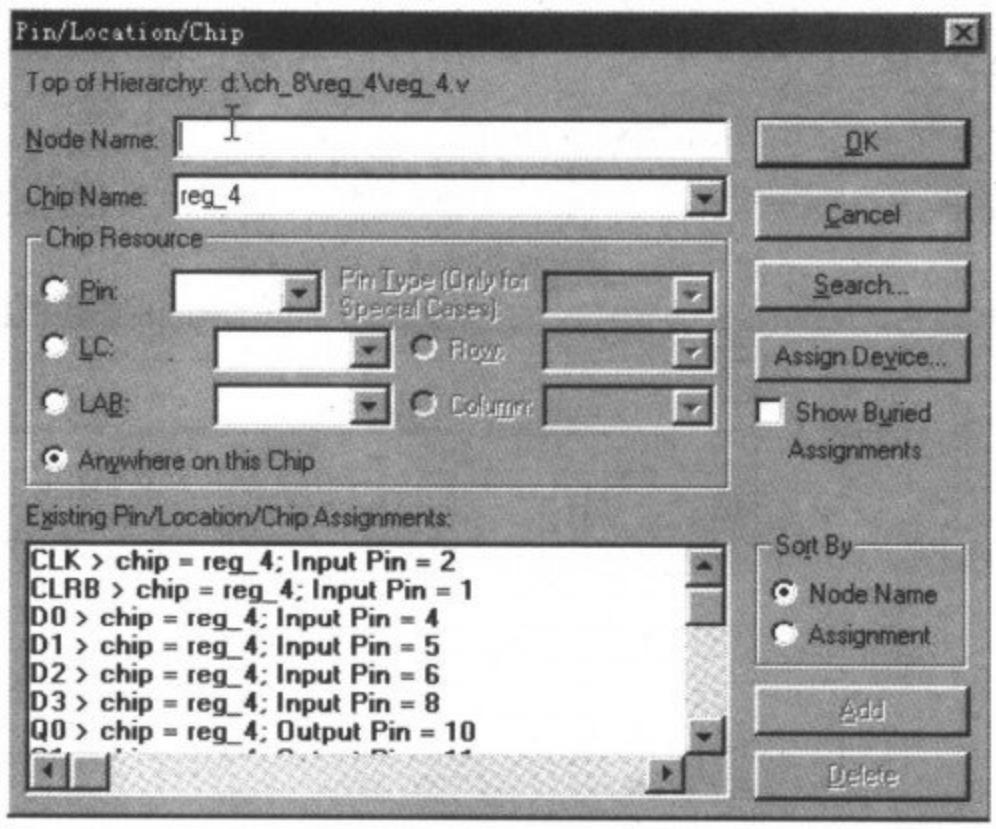


图 8-82　基本寄存器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令,弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标,在弹出的快捷菜单中选择 EnterNodes form SNF... 命令,这时将出现如图仿真信号选择对话框。点击 List,将出现端口列表,你默认是选择全部,你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标,将 D0,D1,D2,D3,CLK,CLRB,Q0,Q1,Q2,Q3 信号加入 SNF 文件中,如图 8-83 所示。

点击菜单命令 File→End Time,终止时间设置为 100μs。

选中 D[3:0],点击 XC 按钮,在弹出的对话框中,将 Starting Value 开始电平设为 0000,Increment By 增加值设为 0001,Multiplied By 设为 40。点击 OK 按钮。选中 CLK,点击 XC 按钮,在弹出的时钟设置对话框中,将 Starting Value 开始电平设为 1,Multiplied By 设为 20。选中 CLRB,用拖动的方法将开始的 5μs 设置为低电平,其他时段设置为高电平。单击保存按钮,将文件保存为 REG_4.scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令,在弹出的对话框中,点击 Start 开始按钮,开始仿真,仿真的波形图如图 8-84 所示。

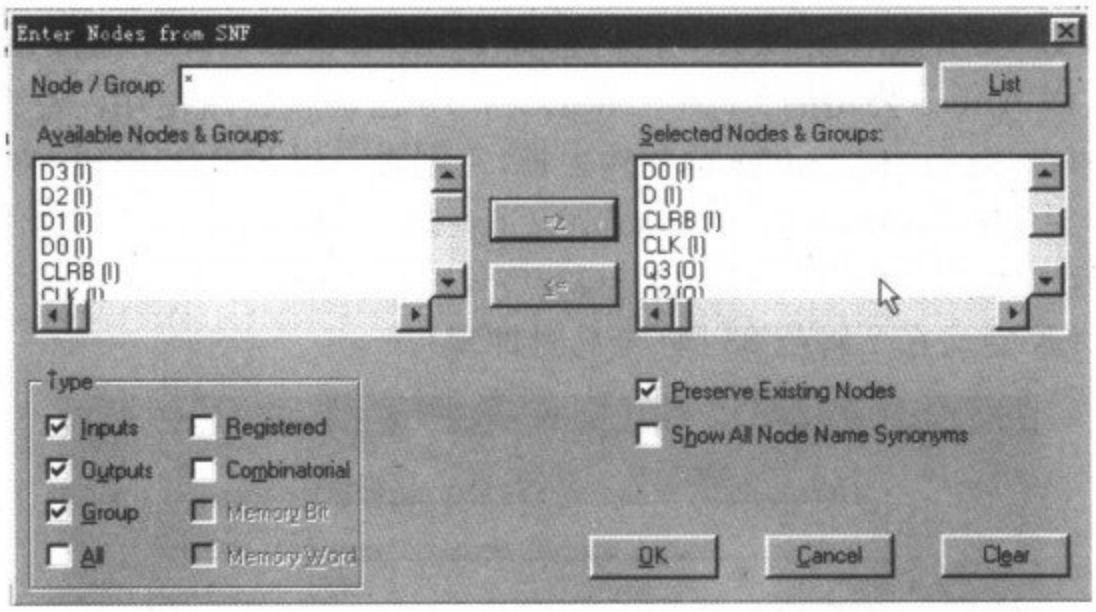


图 8-83　基本寄存器仿真端口列表

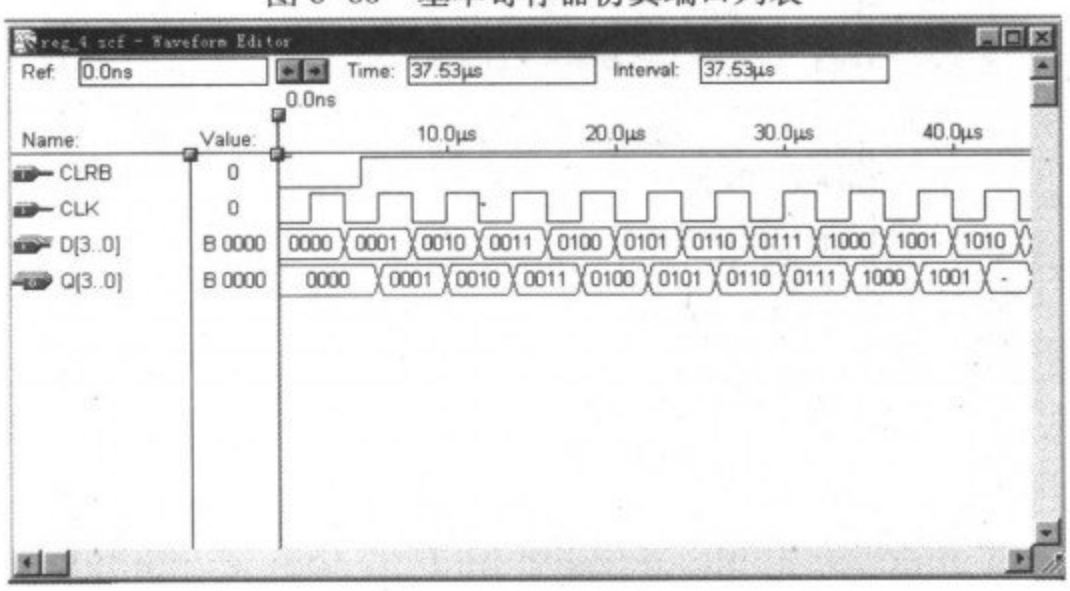


图 8-84　基本寄存器仿真波形图

右击 D[3:0]，在出现的快捷菜单中选择 upgroup，将 D 分解为 D0，D1，D2，D3；右击 Q[3:0]，在出现的快捷菜单中选择 upgroup，将 Q 分解为 Q0，Q1，Q2，Q3；则展开后的波形如图 8-85 所示。

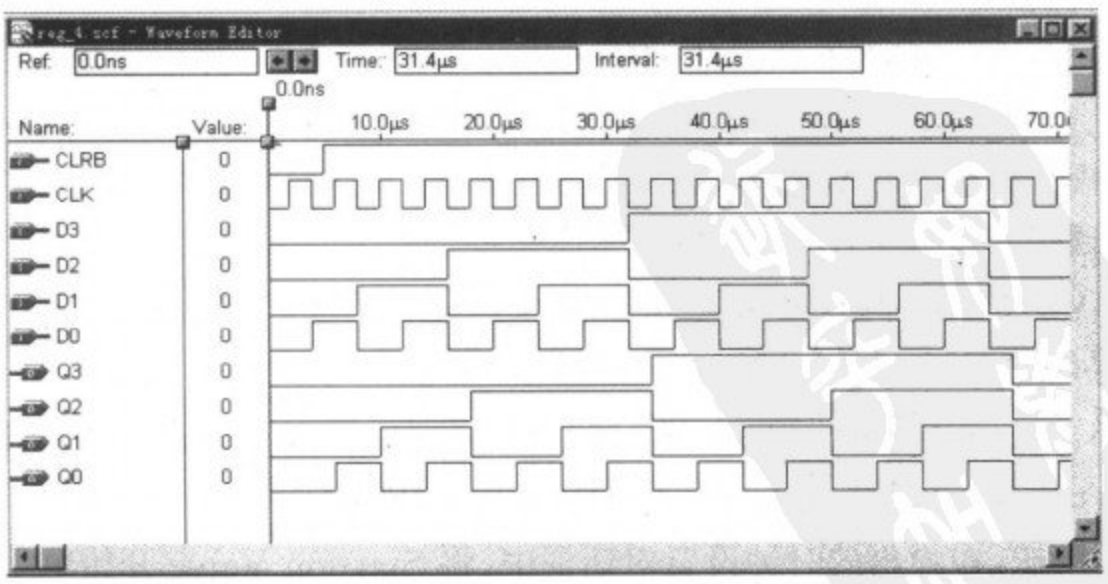


图 8-85　展开后的基本寄存器波形

2. 移位寄存器

(1)移位寄存器分析

存储在寄存器中的数据或代码，在移位脉冲的操作下，可以依次逐位右移或左移，而

数据或代码，既可以并行输入、并行输出，也可以串行输入、串行输出，还可以并行输入、串行输出，串行输入、并行输出，十分灵活，用途也很广。下面主要以 4 位串行输入并行输出移位寄存器为例进行说明。

4 位串行输入并行输出移位寄存器逻辑电路如图 8-86 所示。

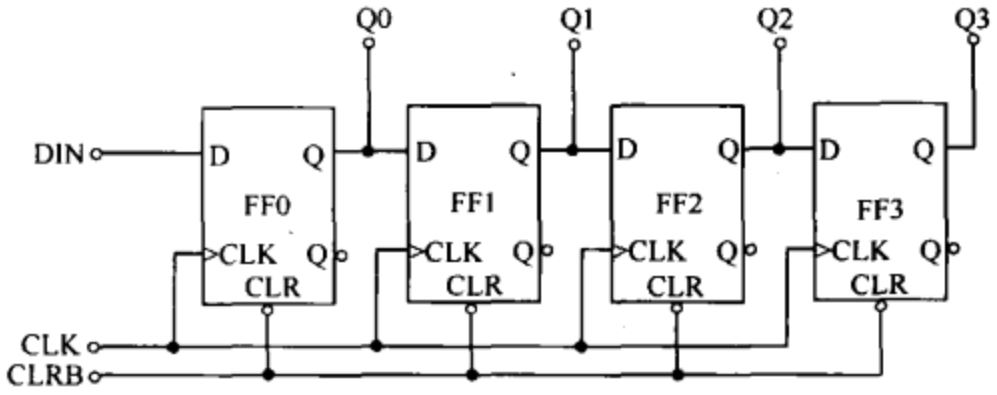


图 8-86　4 位串行输入并行输出移位寄存器逻辑电路

图中，4 个 D 触发器串联运用，每一个 D 触发器的 CLRB 端相连接，用于寄存器置 0。CLK 端是移位脉冲输入端，Q3～Q0 是寄存器的并行输出端，另外，Q3 还作为这一寄存器的串行输出端。

使用前，先给 CLRB 端输入负脉冲，使寄存器置 0，即 Q3Q2Q1Q0＝0000。

设输入数码是 D3D2D1D0＝1101，D3 为最高位，D0 为最低位，；输入的数码从高位(D3)至低位(D0)依次送到触发器 FF0 的输入端。

第一个 CLK 脉冲到来后，在 CLK 脉冲的上升沿，从输入端输入最高位 D3 的数码 1，存入 FF0 触发器中，即 Q0＝1。由于第一个 CLK 作用前，Q3～Q0 都是 0，所以，在第一个 CLK 脉冲作用后，寄存器输出状态为 Q3Q2Q1Q0＝0001。

第二个 CLK 脉冲到来后，在 CP 脉冲的上升沿触发下，电路发生了两种变化：一是由于 Q0＝1，它加到 FF1 触发器的输入端，所以 Q1＝1。二是 D2 输入数码 1 从 FF0 触发器的输入端输入，使 Q0＝1。这样，在第二个 CLK 脉冲作用后，寄存器输出状态为 Q3Q2Q1Q0＝0011。

同理可分析出，在第三个 CLK 脉冲作用后，寄存器的输出状态为 Q3Q2Q1Q0＝0110；在第四个 CLK 脉冲作用后，寄存器的输出状态为 Q3Q2Q1Q0＝1101

由以上分析可知，从输入端输入的 D3D2D1D0＝1101 经过 4 个 CLK 脉冲作用后，已将这四位数移存于这一寄存器电路中。为了便于理解，表 8-14 给出了 4 位移位寄存器输出状态表。

表 8-14　4 位移位寄存器输出状态表

CLK 作用的次数	输出端 Q3	输出端 Q2	输出端 Q1	输出端 Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	0
4	1	1	0	1

(2)用 Verilog HDL 描述 4 位串行输入并行输出移位寄存器

用 Verilog HDL 描述 4 位串行输入并行输出移位寄存器如下：

```
module SHIFT_4(DIN,CLK,CLRB,Q);
input DIN,CLK,CLRB;
output[3:0] Q;
reg[3:0] Q;
always @(posedge CLK or negedge CLRB)
    begin
      if (! CLRB) Q<= 4'b0000; //同步清 0,低电平有效
      else
        begin
        Q<= Q << 1;              //输出信号左移一位
        Q[0] <= DIN;             //输入信号补充到输出信号的最低位
        end
    end
endmodule
```

(3)4 位串行输入并行输出移位寄存器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 SHIFT_4。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 SHIFT_4. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 DIN，CLK，CLRB，Q0，Q1，Q2，Q3，锁定完毕单击 OK 即可，如图 8-87 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命

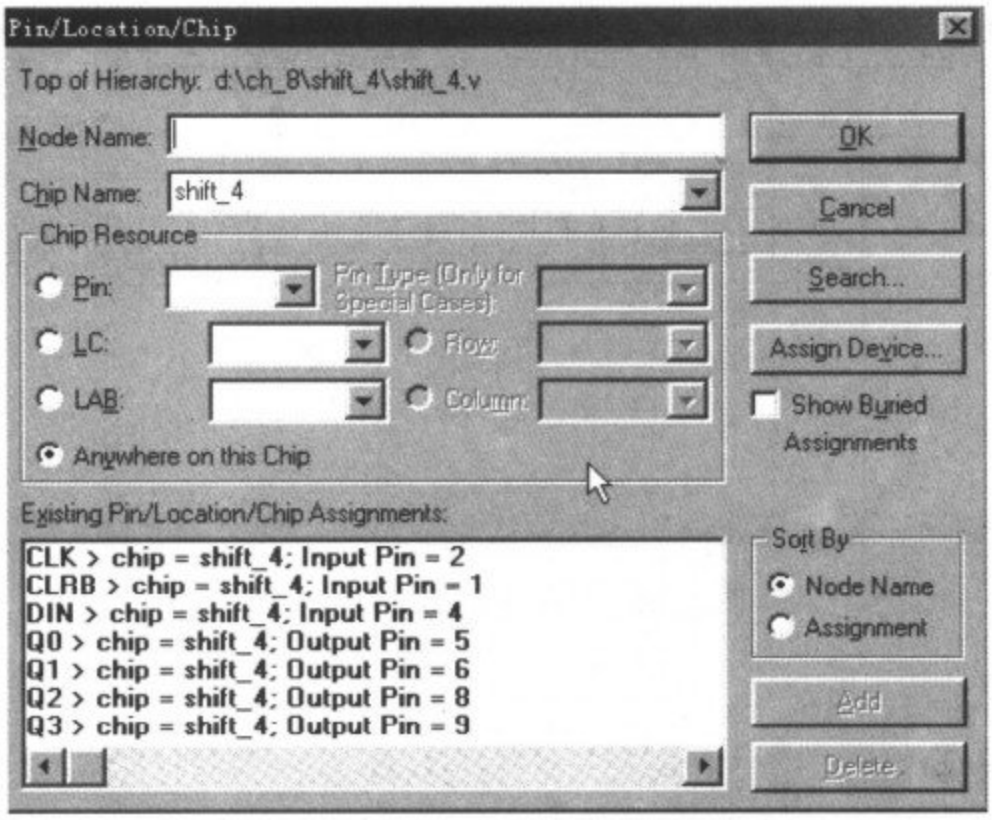


图 8-87　移位寄存器引脚锁定情况

令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 DIN，CLK，CLRB，Q0，Q1，Q2，Q3 信号加入 SNF 文件中，如图 8-88 所示。

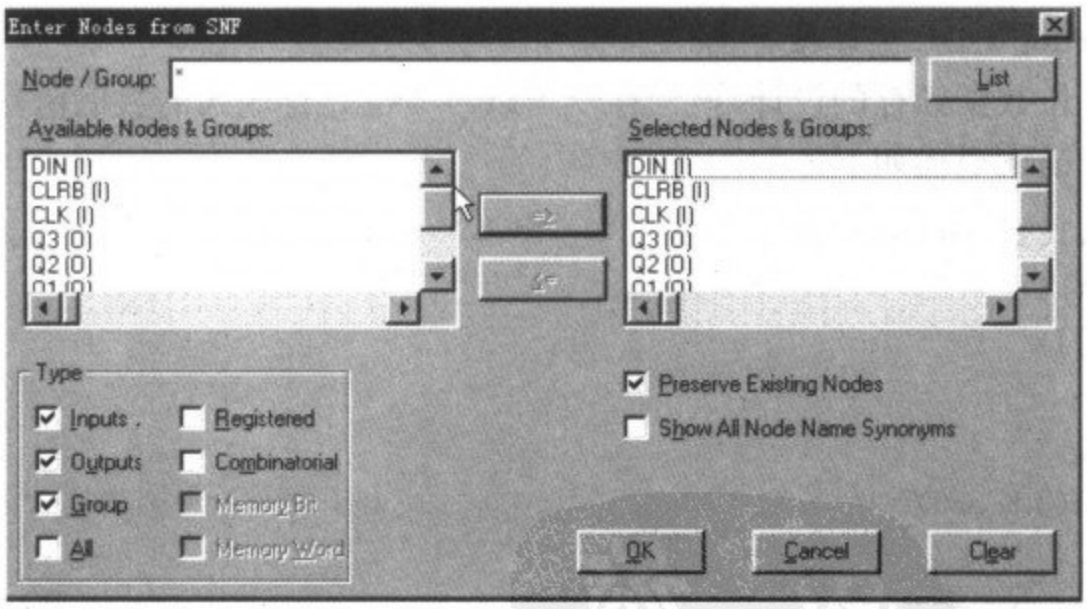


图 8-88　移位寄存器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 DIN，点击 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0，Increment By 增加值设为 1，Multiplied By 设为 80。点击 OK 按钮。选中 CLK，点击 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 0，Multiplied By 设为 20。选中 CLRB，用拖动的方法将开始的 4μs 设置为低电平，其他时段设置为高电平。单击保存按钮，将文件保存为 SHIFT_4. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-89 所示。

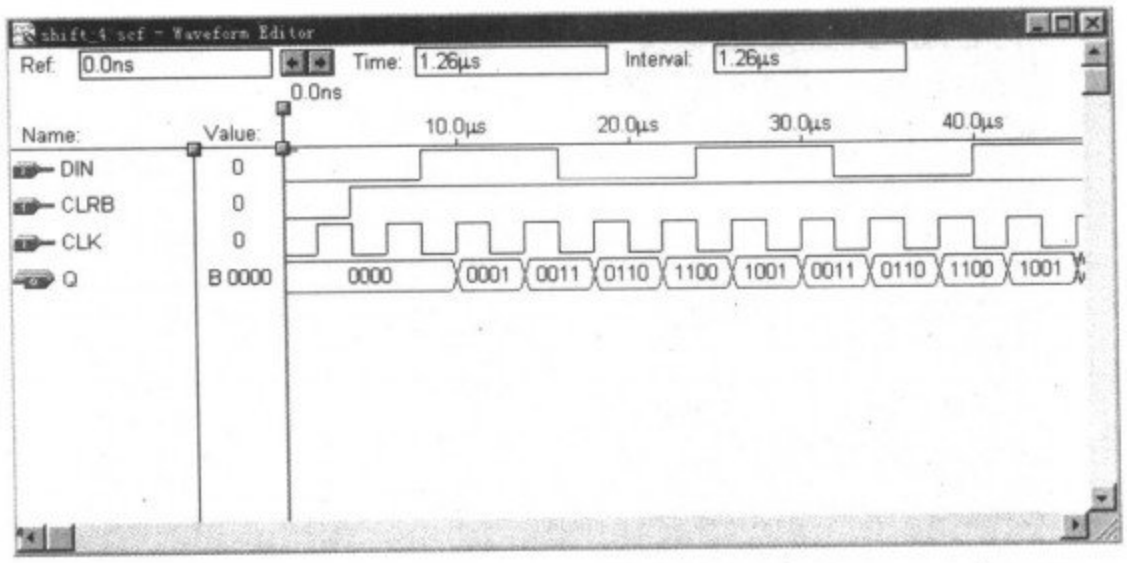


图 8-89　移位寄存器仿真波形图

二、锁存器

1. 数据锁存器分析

先来看一下数据锁存器和数据寄存器的区别。从寄存数据的角度来看，锁存器和寄存的功能是相同的，两者的区别在于：锁存器一般由电平信号来控制，属于电平敏感型，而寄存器一般由同步时钟信号控制。两者有不同的使用场合，主要取决于控制方式及控制信号和数据信号之间的时序关系：若数据有效滞后于控制信号有效，则只能使用锁存器；若数据提前于控制信号，并要求同步操作，则可以选择寄存器来存放数据。

2. 用 Verilog HDL 描述锁存器

下面是 4 位数据锁存的设计，该锁存器可以对 4 位并行输入的数据信号进行锁存，用 Verilog HDL 描述描述如下：

```
module LATCH_4(Q,D,CLK);
output[3:0] Q;
input[3:0] D;
input CLK;
reg[3:0] Q;
always  @(CLK or D)
  begin
  if  (CLK) Q<=D;
  end
endmodule
```

3. 锁存器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 LATCH_4。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 LATCH_4. v 文件。单击菜单 File→Project→Set Pro-

ject To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 D0，D1，D2，D3，CLK，Q0，Q1，Q2，Q3，锁定完毕单击 OK 即可，如图 8-90 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

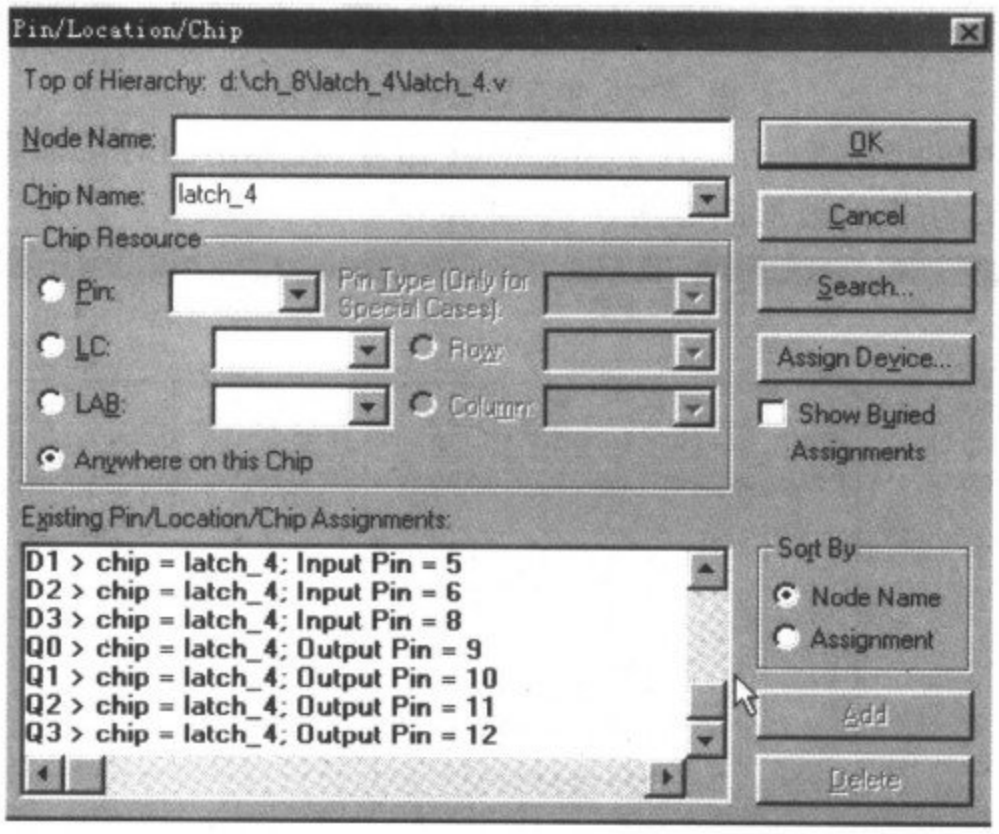


图 8-90　锁存器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 D0，D1，D2，D3，CLK，Q0，Q1，Q2，Q3 信号加入 SNF 文件中，如图 8-91 所示。

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 D[3:0]，点击 XC 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0000，Increment By 增加值设为 0001，Multiplied By 设为 40。点击 OK 按钮。选中 CLK，点击 XC 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 1，Multiplied By 设为 20。单击保存按钮，将文件保存为 LATCH_4. scf 文件。

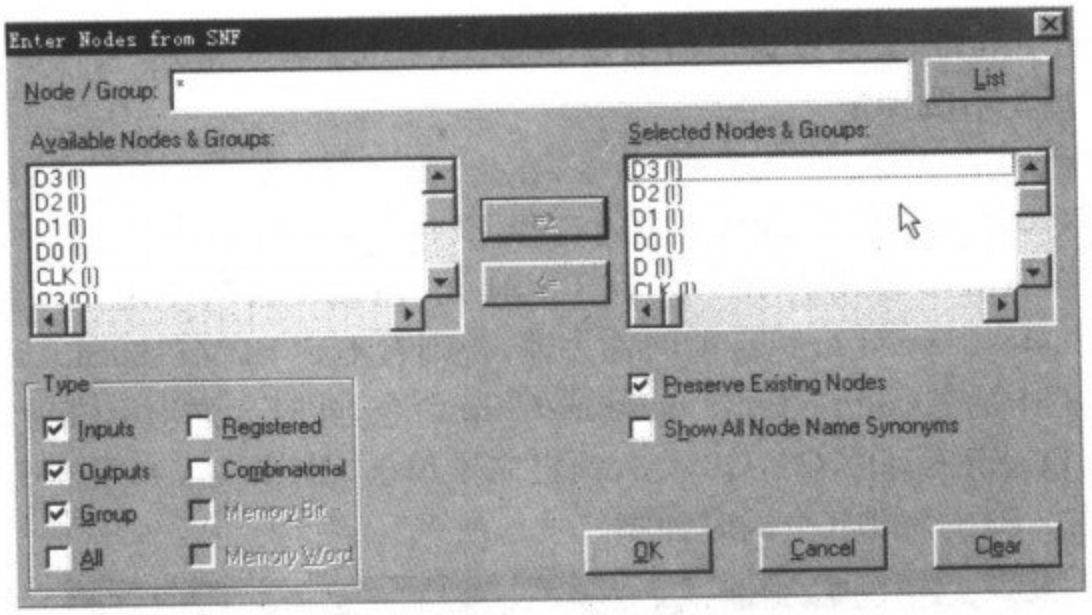


图 8-91　锁存器仿真端口列表

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-92 所示。

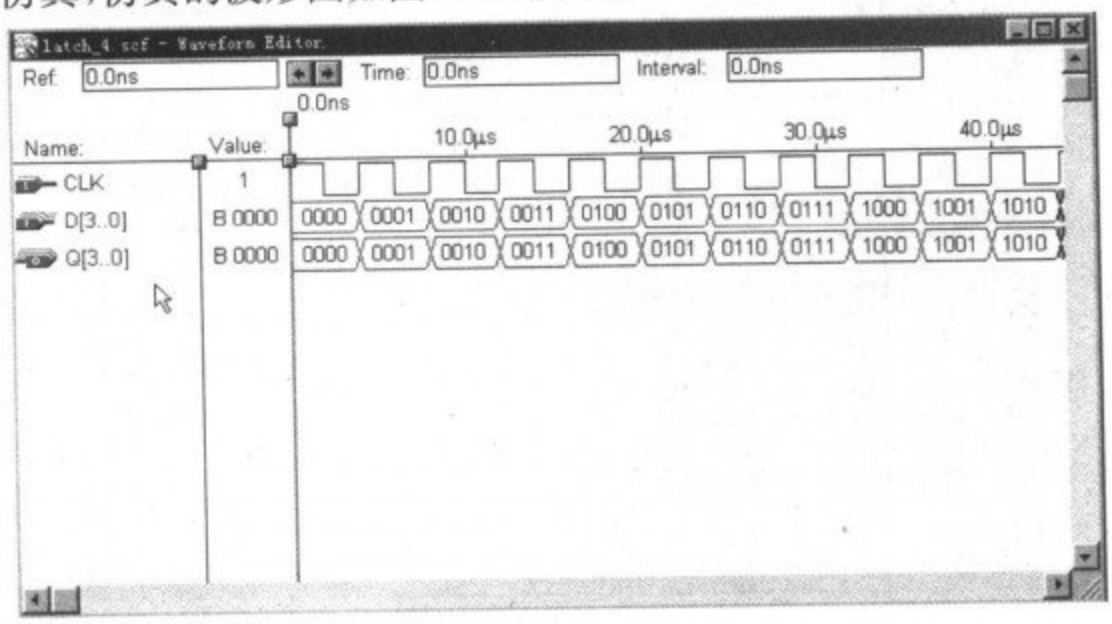


图 8-92　锁存器仿真波形图

右击 D[3:0]，在出现的快捷菜单中选择 upgroup，将 D 分解为 D0，D1，D2，D3；右击 Q[3:0]，在出现的快捷菜单中选择 upgroup，将 Q 分解为 Q0，Q1，Q2，Q3；则展开后的波形如图 8-93 所示。

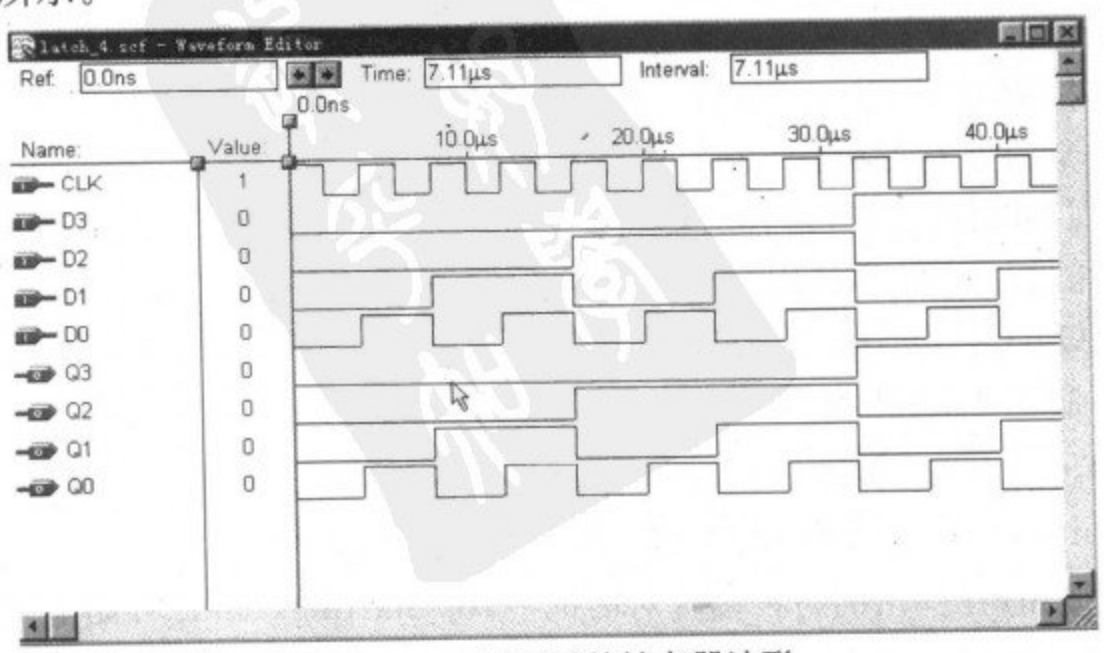


图 8-93　展开后的锁存器波形

三、计数器

在数字系统中，把记忆输入 CLK 脉冲个数的操作叫做计数，能实现计数操作的电路称为计数器。计数器是数字系统中用得较多的基本逻辑器件。它不仅能记录输入时钟脉冲的个数，还可以实现分频、定时、产生节拍脉冲和脉冲序列等。例如，计算机中的时序发生器、分频器、指令计数器等都要使用计数器。

1. 计数器的分类

(1)按数的进制分

按数的进制分，计数器可分为以下 3 类：

①二进制计数器

当输入计数脉冲到来时，按二进制数规律进行计数的电路都叫做二进制计数器。

②十进制计数器

按十进制数规律进行计数的电路称为十进制计数器。

③N 进制计数器

除了二进制和十进制计数器之外的其他进制的计数器，都叫做 N 进制计数器，例如，$N=8$ 时的八进制计数器，$N=16$ 时的十六进制计数器等。

(2)按计数时是递增还是递减分

按计数时是递增还是递减分，计数器可分为以下 3 类：

①加法计数器

当输入计数脉冲到来时，按递增规律进行计数的电路叫做加法计数器。

②减法计数器

当输入计数脉冲到来时，进行递减计数的电路称为减法计数器。

③可逆计数器

在加减信号的控制下，既可进行递增计数，也可进行递减计数的电路叫做可逆计数器。

(3)按计数器中触发器翻转是否同步分

按计数器中触发器翻转是否同步分，计数器可分为以下 2 类：

①同步计数器

当输入计数脉冲到来时，要更新状态的触发器都是同时翻转的计数器，叫做同步计数器。从电路结构上看，计数器中各个时钟触发器的时钟信号都是输入计数脉冲。

②异步计数器

当输入计数脉冲到来时，要更新状态的触发器，有的先翻转有的后翻转，是异步进行的，这种计数器称为异步计数器。从电路结构上看，计数器中各个时钟触发器，有的触发器其时钟信号是输入计数脉冲，有的触发器其时钟信号却是其他触发器的输出。

2. 二进制异步计数器

(1)二进制异步加法计数器

图 8-94 是一个由 3 个 D 触发器构成的 3 位二进制异步加法计数器电路。图中，3 个 D 触发器的清零端相连后作为复位端$\overline{\text{CLR}}$(用 Verilog HDL 描述时记为 CLRB)，CLK 端是计数脉冲输入端，Q 端为触发器输出端，本位的 $\overline{Q}$ 端(用 Verilog HDL 描述时记为 QB)与高一位触发器的计数输入端 CLK 和本位触发器的 D 端相连。

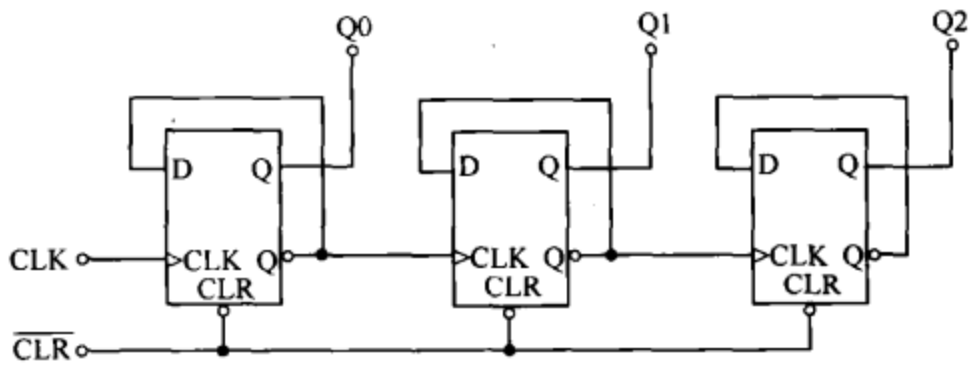


图 8-94　异步二进制加法计数器逻辑电路

计数前，一般需要在CLR端加上负脉冲，使各 D 触发器清零，即 Q2＝0，Q1＝0，Q0＝0。

从电路中可以看出，各触发器的 $\overline{Q}$ 端与 D 端相连，此时 D 触发器为计数状态，有一个 CLK 脉冲的有效触发，D 触发器输出端 Q 翻转一次。

第一个计数脉冲 CLK 到来后，在 CLK 脉冲从 0 变成 1 的上升沿，触发器 FF0 触发，其输出端 Q0 由原来的 0 变成 1（$\overline{Q0}$由 1 变为 0），即第一个 CLK 脉冲过后 Q0＝1。由于 $\overline{Q0}$与下一位触发器 FF1 的 CLK 端相连，$\overline{Q0}$从 1 变成 0（即下降沿）不能对 FF1 构成有效触发（因为触发器是上升沿触发），所以 FF1 保持原输出状态，即 Q1＝0。同理，第一个 CLK 脉冲作用时，Q2＝0。可见，在第一个 CLK 脉冲作用后，计数器的输出状态为 Q2＝0，Q1＝0，Q0＝1。

第二个计数脉冲 CLK 到来后，CLK 脉冲上升沿对触发器 FF0 再次有效触发，其输出端 Q0 由原来的 1 变成 0，$\overline{Q0}$由 0 变成 1。$\overline{Q0}$对 FF1 进行触发，所以 FF1 翻转一次，其输出端 Q1 从 0 变成 1（$\overline{Q1}$从 1 变成 0）。由于$\overline{Q1}$从 1 变成 0（下降沿）不能对 FF2 形成有效触发，于是 FF2 保持原状态。可见，在第二个 CLK 脉冲作用之后，计数器的输出状态为 Q2＝0，Q1＝1，Q0＝0。

同样的道理，在 CLK 脉冲的不断输入触发下，电路中的各触发器作相应的翻转变化，完成二进制加法计数。3 位二进制加法计数器电路的状态表如表 8-15 所示。

表 8-15　3 位二进制加法计数器状态表

输入脉冲个数	各触发器状态			表示的十进制数
	Q2	Q1	Q0	
0	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7
8	0	0	0	0

从状态表可知：当 CLK 脉冲完成第 7 个触发后，计数器的输出端为 111，即 Q2＝Q1＝Q0＝1，此时第 8 个 CLK 脉冲出现后，计数器状态为 000，第 9 个 CLK 脉冲出现后，计

数器状态为 001，开始从头计数，即 3 位的二进制加法计数器最多只能计 8 个数，因此，3 位二进制计数器是一种八进制计数器。

由以上分析可知，来一个 CLK 脉冲，最低位的触发器 FF0 就翻转一次，而触发器 FF1 是在 FF0 翻转 2 次后才翻转 1 次，显然，FF1 的翻转频率就比 FF0 降低了一倍。当 FF1 翻转两次时，FF2 才翻转一次，也就是当 FF0 翻转四次时，FF2 才翻转一次。如图 8-95 所示是 3 位二进制加法计数器的工作波形示意图，从该工作波形图中可清楚地看出上述关系。

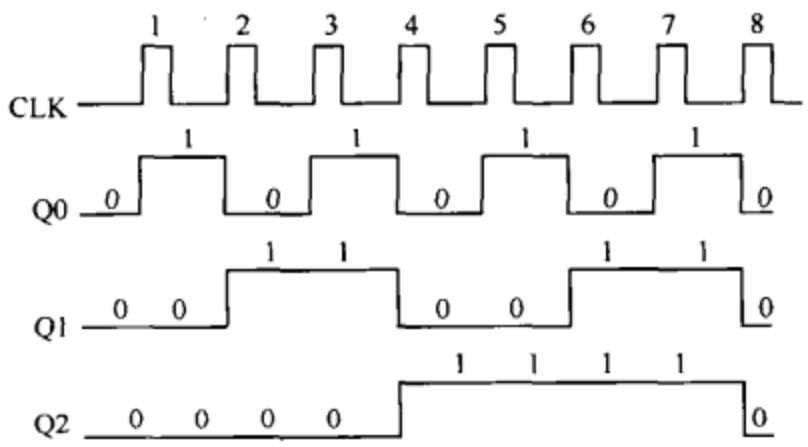


图 8-95　3 位异步二进制加法计数器工作波形

(2)二进制异步减法计数器

在二进制异步加法计数器电路中，若后一级的 CLK 接至前一级的 Q 端而不是 $\overline{Q}$ 端，如图 8-96 所示，则该电路就成为异步减法计数器。其对应的状态表如表 8-16 所示。

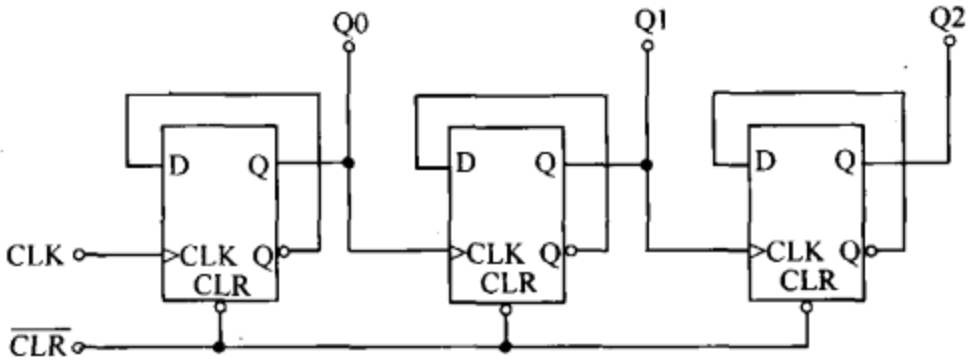


图 8-96　3 位二进制异步减法计数器逻辑电路

表 8-16　3 位二进制异步减法计数器状态表

输入脉冲个数	各触发器状态			表示的十进制数
	Q2	Q1	Q0	
0	0	0	0	0
1	1	1	1	7
2	1	1	0	6
3	1	0	1	5
4	1	0	0	4
5	0	1	1	3
6	0	1	0	2
7	0	0	1	1
8	0	0	0	0

在复位后，各输出端为 0000，即 Q2＝0，Q1＝0，Q0＝0。

当第一个计数脉冲 CLK 到来后，在 CLK 脉冲从 0 变成 1 的上升沿，触发器 FF0 触发，其输出端 Q0 由原来的 0 变成 1，即第一个 CLK 脉冲过后 Q0＝1。由于 Q0 从 0 变成 1，而 Q0 端与 FF1 的 CLK 端相连，对触发器 FF1 是有效触发，使 Q1 从 0 变成 1。同样的道理，Q1 从 0 变成 1，使 Q2＝1。这样，在第一个计数脉冲 CLK 作用后，减法计数器输出状态变成 Q2＝1，Q1＝1，Q0＝1。

用同样的方法进行分析，就可以得到与上面状态表相同的结果，读者可自己分析。

(3)用 Verilog HDL 描述 3 位二进制(八进制)异步加法计数器

用 Verilog HDL 描述 3 位二进制异步加法计数器如下：

```
/ * 以下是主模块 * /
`include "D_FF. v"                    //包含 D_FF 子模块
module     CNT_3(CLRB, CLK, Q );
input      CLRB, CLK;
output[2:0] Q;
wire[2:0] QB;
  D_FF     D_FF0  ( CLRB, QB[0], CLK, Q[0], QB[0] );
  D_FF     D_FF1  ( CLRB, QB[1], QB[0], Q[1], QB[1] );
  D_FF     D_FF2  ( CLRB, QB[2], QB[1], Q[2], QB[2] );
endmodule
/ * 以下是 D 触发器 D_FF 子模块 * /
module     D_FF     ( CLRB, D, CLK, Q, QB );
input      CLRB, D, CLK;
output     Q, QB;
reg     Q;
assign     QB = ~Q;
always     @(posedge CLK or negedge CLRB )
    begin
      if(! CLRB)   Q<=0;
      else         Q<= D;
    end
endmodule
```

(4)二进制异步计数器的仿真

下面用 MAX＋plusII 进行仿真，步骤如下：

①新建项目

进入 MAX＋plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 CNT_3。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，在出现文本编辑窗口。输入以上主模块程序，将其保存为 CNT_3. v 文件。再单击 File→new，新建文件时，选择 Text Editor File 选项，在出现文本编辑窗口中，输入以上 D_FF 子模块程序，将其保存为 D_

FF. v,注意,要将 CNT_3. v 和 D_FF. v 都保存在 CNT_3 文件夹中,然后将 D_FF. v 文本编辑窗口关闭。单击菜单 File→Project→Set Project To Current File,把 CNT_3. v 文件设为当前工程,至此,Verilog 输入完成。

重点提示 这是一个具有多层次结构的 Verlog HDL 设计。在这个例子中,主模块 CNT_3. v 中调用了 D 触发器 D_FF. v,而 D_FF. v 是作为一个单独子模块独立于主模块的。在做设计时,可以先设计好若干个具有特定功能的子模块,然后再设计主模快。在模块中只要将子模块用`include 包含进来,整个设计可顺利地完成,整体调试的周期也会缩短。调用子模块的好处在于:具有较好的资源重复利用性。例如在本例中,D_FF. v 这个子模块就被调用了 3 次。

另外,MAX+plusII 还提供了库管理功能,假设文件 CNT_3. v 和 D_FF. v 都在目录 d:\ch_8\CNT_3 目录下,只在要综合 CNT_3. v 项目文件时选择菜单 Options→User libraries 打开如图 8-97 所示的对话框,将目录 d:\ch_8\CNT_3 加到 Existing Directories 中即可,这样,在综合 CNT_3. v 时会自动调用 CNT_3. v 和 D_FF. v 文件。

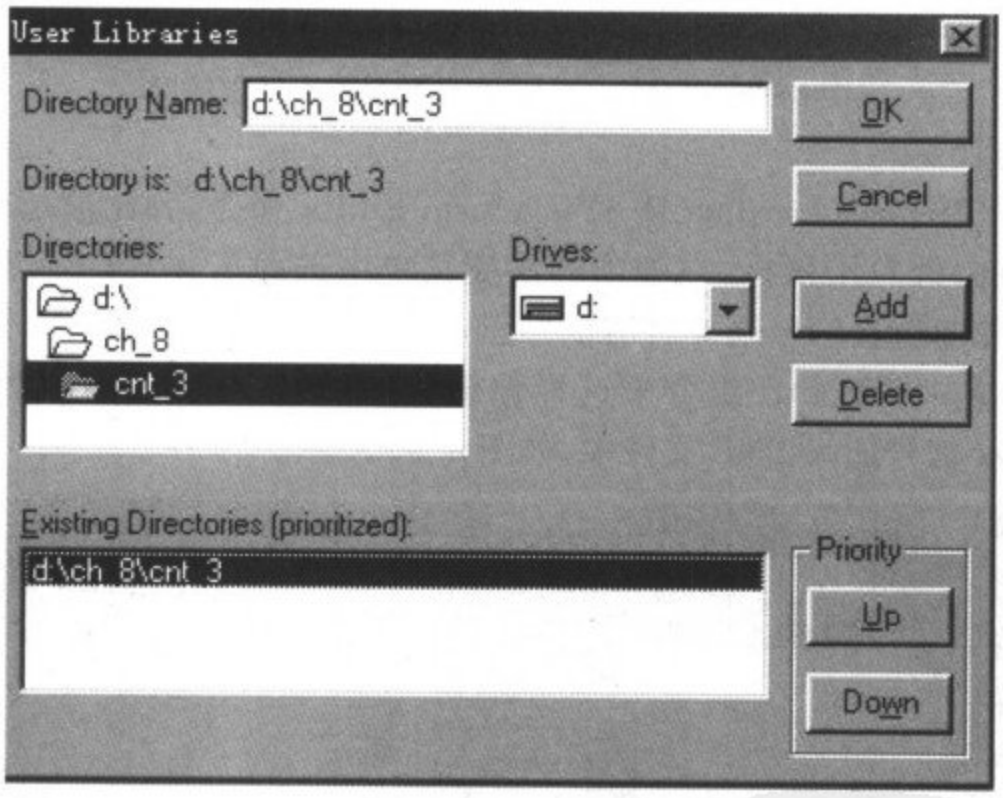


图 8-97 库管理对话框

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中:Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 CLRB,CLK,Q0,Q1,Q2,锁定完毕单击 OK 即可,如图 8-98 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

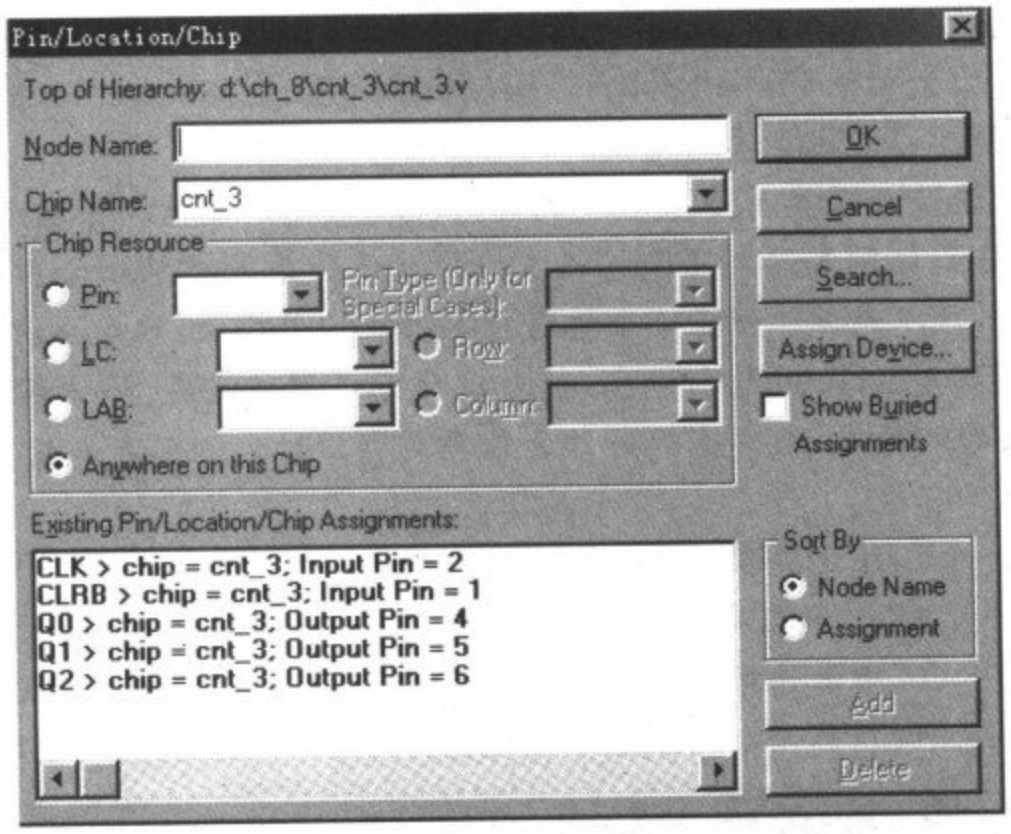


图 8-98　3 位二进制异步计数器引脚锁定情况

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 CLRB，CLK，Q0，Q1，Q2 信号加入 SNF 文件中，如图 8-99 所示。

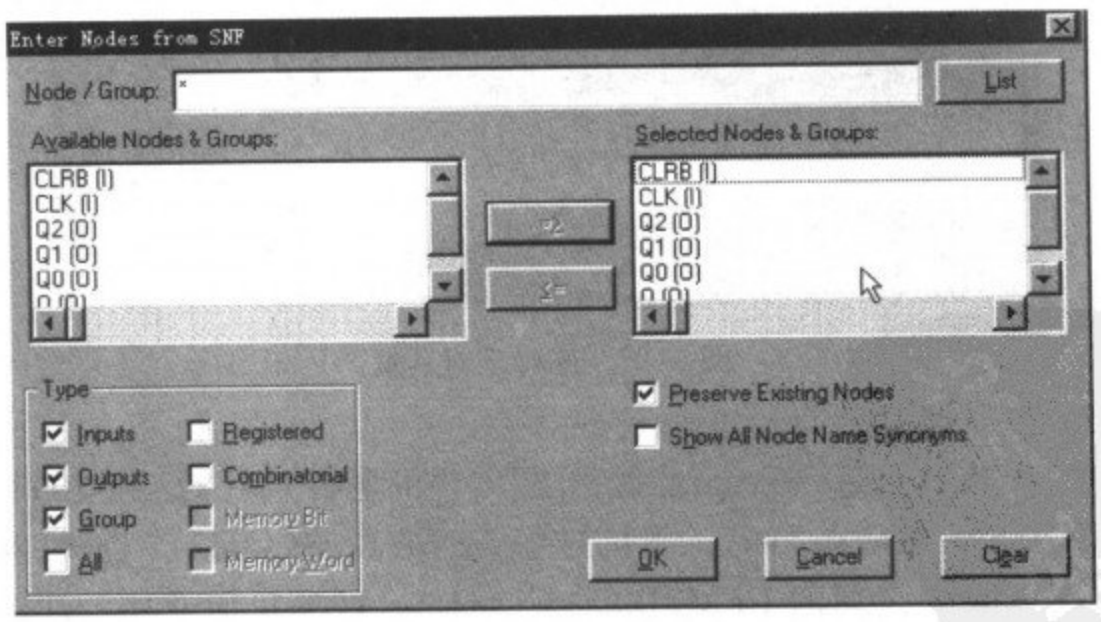


图 8-99　3 位二进制异步计数器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 CLK，点击按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 0，Multiplied By 设为 20。选中 CLRB，用拖动的方法将开始的(0～1)μs 设置为低电平，其余时段设置为高电平。单击保存按钮，将文件保存为 CNT_3. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-100 所示。

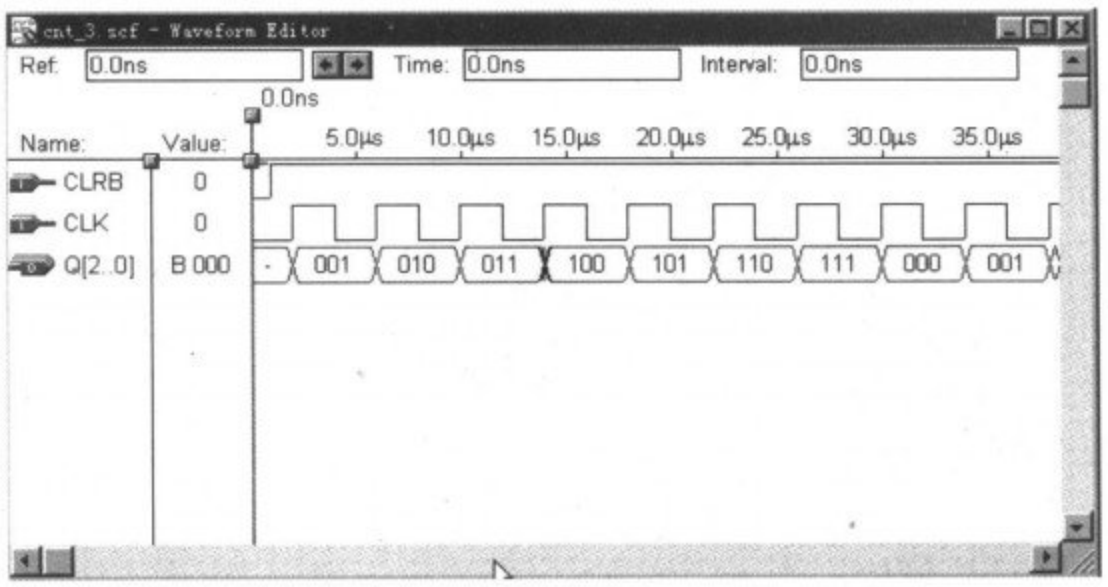


图 8-100　3 位二进制异步计数器仿真波形图

右击 Q[2:0],在出现的快捷菜单中选择 upgroup,将 Q 分解为 Q0,Q1,Q2;则展开后的波形如图 8-101 所示。

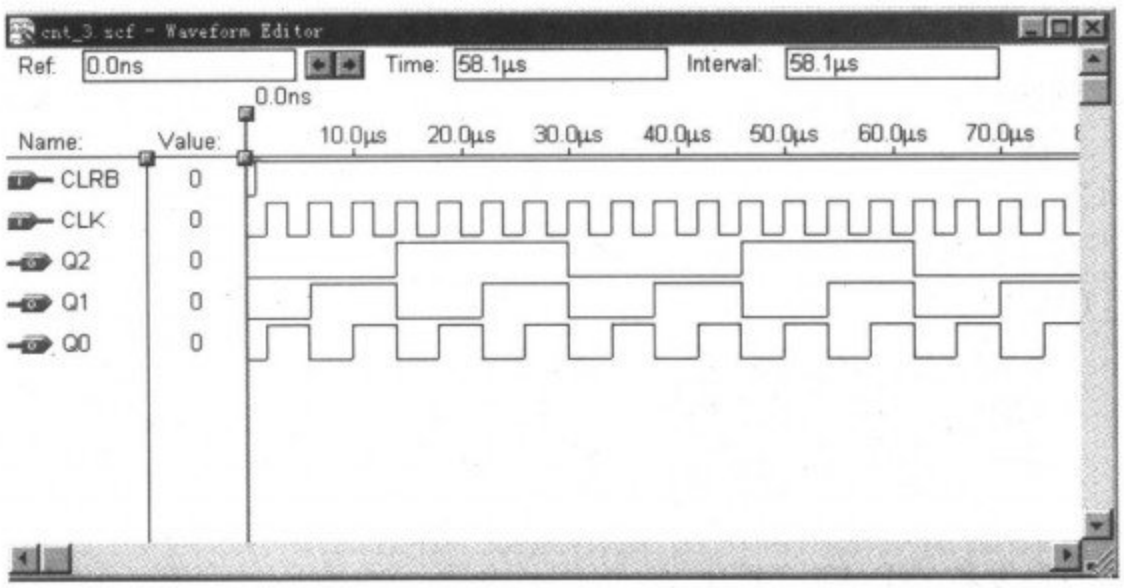


图 8-101　展开后的 3 位二进制异步计数器波形

3. 二进制同步计数器

(1)二进制同步计数器分析

上面所介绍的计数器的连接特点之一是外部计数脉冲只作用于首级;特点之二是各级触发器的动作时间是有先后次序的,先是首级,接着是次级……故称为异步计数器。这种计数器的电路结构简单,但当级联数目较多时,会影响总体的工作速度。而同步计数器可以解决这一矛盾。

同上计数器是将外部计数脉冲同时作用于所有的触发器,所以各触发器(如果要翻转的话)是在同一时刻翻转的,但各触发器的翻转情况必须符合加法器或减法器的变化规律。

如图 8-102 是 3 位二进制(八进制)同步加法计数器逻辑电路图。具体工作情况比较复杂,这里不再分析。

从图中可以看出,同步加法计数器具有以下连接特点:

①各触发器的 CLK 端同时连接到了外部计数脉冲上;

②FF0 的 $D=\overline{Q0}$;

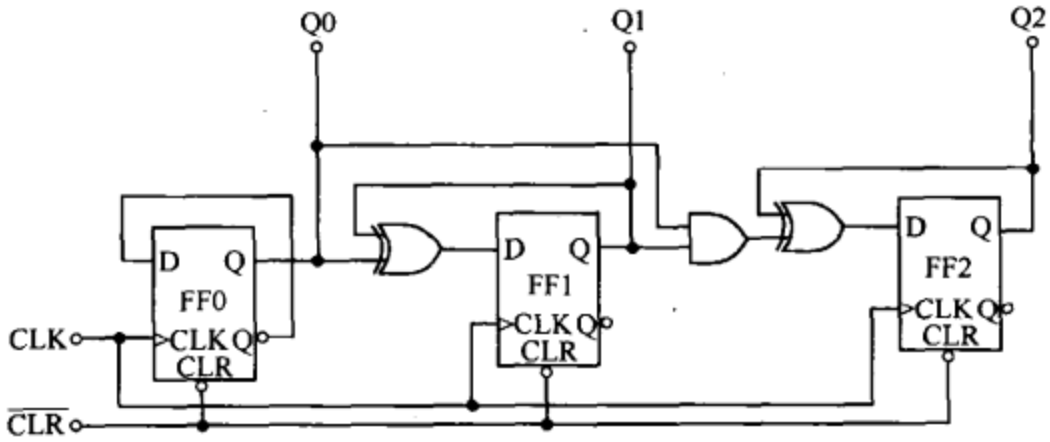


图 8-102　3 位二进制同步加法计数器

③FF1 的 D=Q1^Q0

④FF2 的 D=Q2^Q1&Q0。

(2)用 Verilog HDL 描述 3 位二进制(八进制)同步加法计数器

用 Verilog HDL 描述 3 位二进制同步加法计数器如下：

/ * 以下是主模块 * /

```
`include  "SYD_FF. v"
module    SYCNT_3(CLRB, CLK, Q );
input     CLRB, CLK;
output    [2:0] Q;
wire[2:0] QB;
SYD_FF    SYD_FF0  (CLRB, QB[0] , CLK, Q[0], QB[0] );
SYD_FF    SYD_FF1  ( CLRB, Q[1] ^ Q[0], CLK, Q[1], QB[1] );
SYD_FF    SYD_FF2  ( CLRB, Q[2] ^ Q[1]&Q[0], CLK, Q[2], QB[2] );
endmodule
/ * 以下是 D 触发器 SYD_FF 子模块 * /
module    SYD_FF    (CLRB, D, CLK, Q, QB );
input     CLRB, D, CLK;
output    Q, QB;
reg Q;
assign    QB = ~Q;
always    @(posedge CLK or negedge CLRB )
    begin
      if(! CLRB)  Q<=0;
      else        Q<= D;
    end
endmodule
```

(3)二进制同步计数器的仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 SYCNT_3。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，在出现文本编辑窗口。输入以上主模块程序，将其保存为 SYCNT_3. v 文件。再单击 File→new，新建文件时，选择 Text Editor File 选项，在出现文本编辑窗口中，输入以上 SYD_FF 子模块程序，将其保存为 SYD_FF. v，注意，要将 SYCNT_3. v 和 SYD_FF. v 都保存在 SYCNT_3 文件夹中，然后将 SYD_FF. v 文本编辑窗口关闭。单击菜单 File→Project→Set Project To Current File，把 SYCNT_3. v 文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 CLRB，CLK，Q0，Q1，Q2，锁定完毕单击 OK 即可，如图 8-103 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

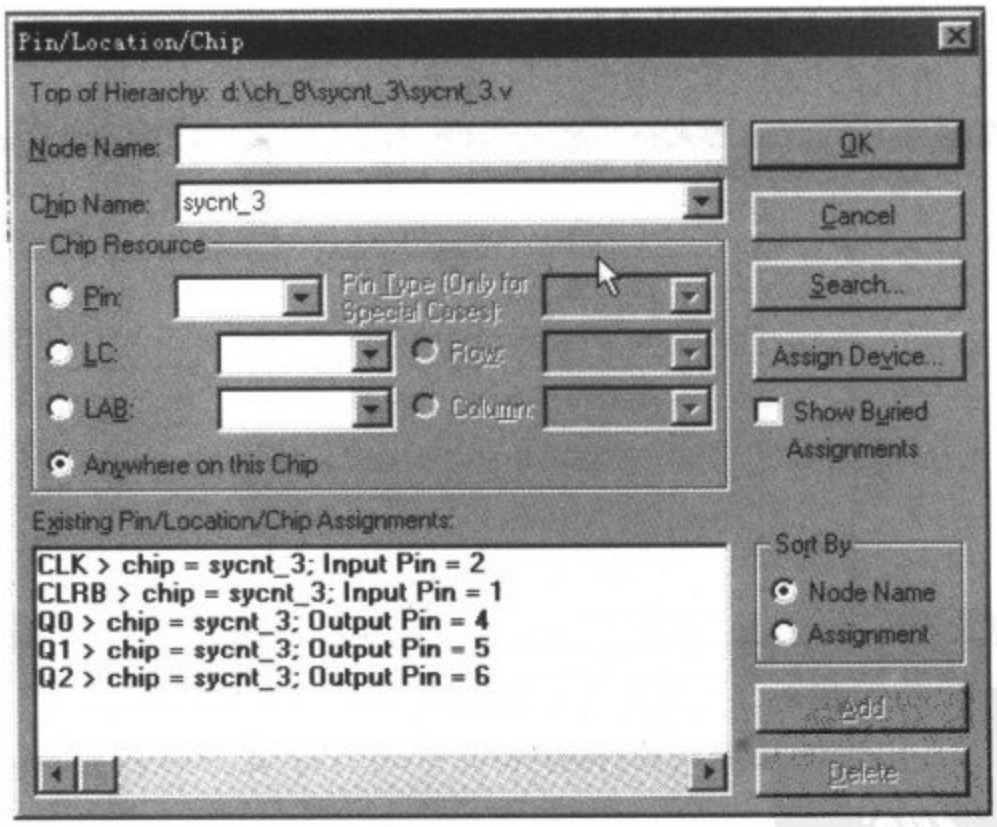


图 8-103　3 位二进制同步计数器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 CLRB，CLK，Q0，Q1，Q2 信号加入 SNF 文件中，如图 8-104 所示。

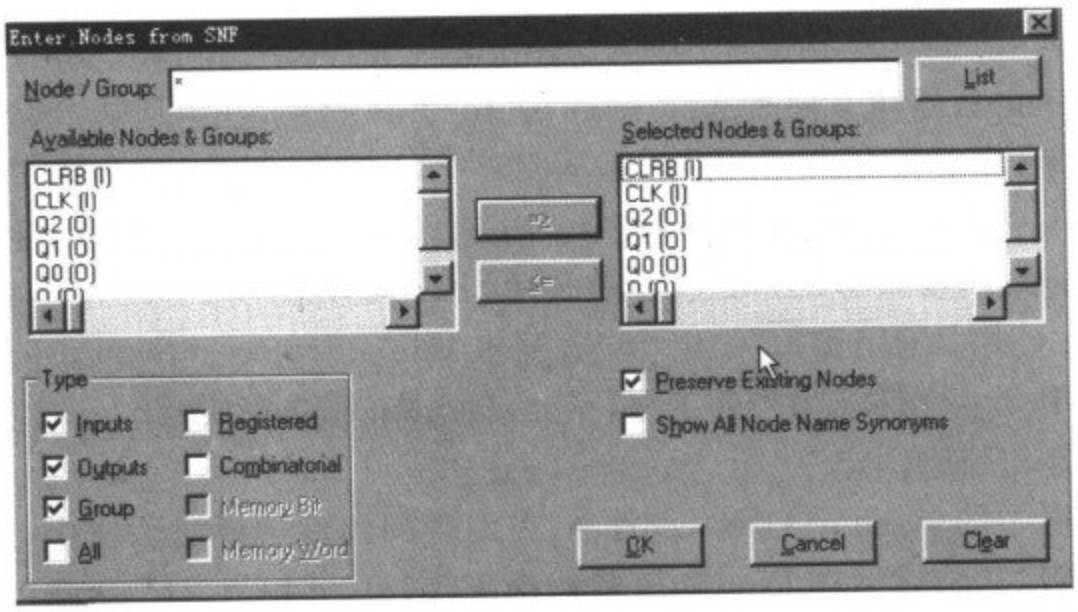


图 8-104　3 位二进制同步计数器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 CLK，点击按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 0，Multiplied By 设为 20。选中 CLRB，用拖动的方法将开始的 0～1μs 设置为低电平，其余时段设置为高电平。单击保存按钮，将文件保存为 SYCNT_3. scf 文件。

执行菜单栏中的 MAX＋plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-105 所示。

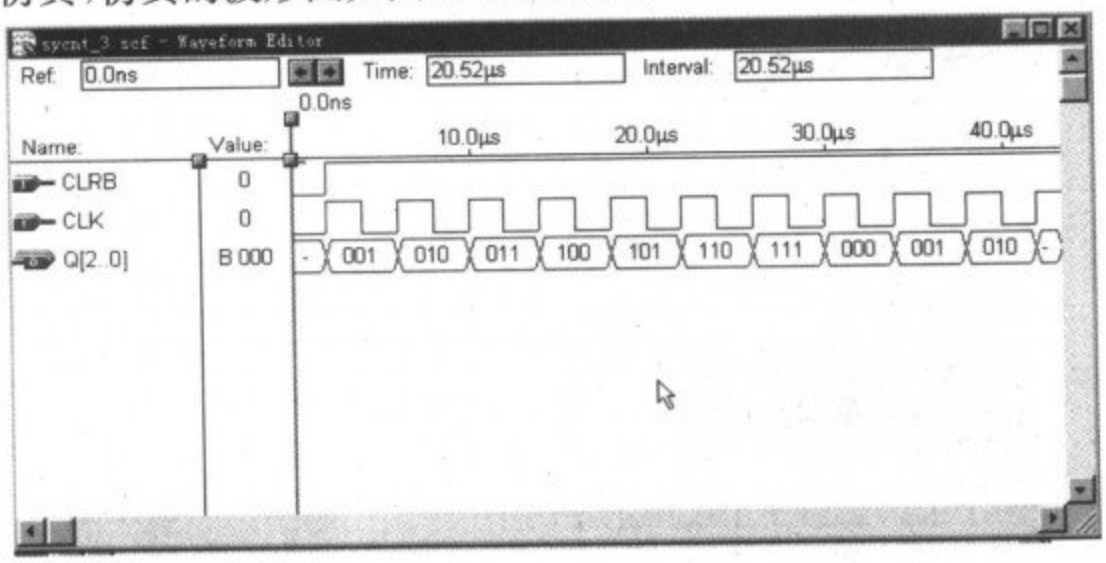


图 8-105　3 位二进制同步计数器仿真波形图

右击 Q[2:0]，在出现的快捷菜单中选择 upgroup，将 Q 分解为 Q0，Q1，Q2；则展开后的波形如图 8-106 所示。

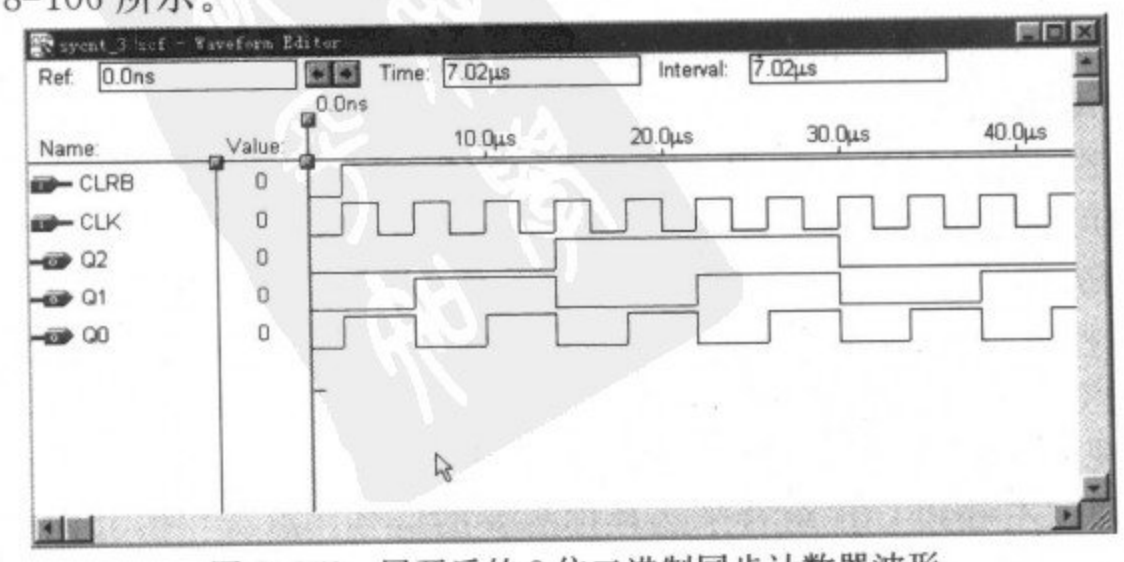


图 8-106　展开后的 3 位二进制同步计数器波形

4. 任意进制计数器

n 位二进制计数器可以组成 2^n 进制的计数器，例如四进制、八进制、十六进制等，但在实际应用中，需要的往往不是 2^n 进制的计数器，例如五进制、七进制、十进制等。这类不为 2^n 进制的计数的组成方法可用触发器和门电路进行设计，但比较复杂，本书不作介绍，下面以设计一个十进制计数器为例，说明用反馈归零法进行设计的任意进制计数器的方法。

(1)十进制计数器分析

十进制计数器是在二进制计数器的基础上演变而来的，它用 4 位二进制数来代表 1 位十进制数，即其输出结果是 BCD 码。

十进制的编码方式有多种，最常用的 8421 编码方式是取 4 位二进制编码中 16 个状态的前 10 个状态 0000～1001 来表示十进制数的 0～9 这 10 个数码的。也就是当计数器计数到第 9 个脉冲后，若再来一个脉冲，计数器的状态必须由 1001 变到 0000，完成一个循环的变化。十进制加法计数器的状态表如表 8-17 所示。

表 8-17　十进制加法计数器的状态表

计数脉冲	Q3	Q2	Q1	Q0	十进制数
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	0(1)	0	0(1)	0	10

根据状态表，可采用反馈归零法来设计十进制计数器，其基本思想是：截取计数过程中的某一个中间状态来控制清零端，使计数器从该状态返回到零而重新开始计数，这样就弃掉了后面的一些状态，把模较大的计数器改成了模较小的计数器(所谓模，是指计数器中循环状态的个数)。对于十进制计数器，需要截取 0～9 前几位数，从第十个脉冲开始，要返回到零状态。

(2)用 Verilog HDL 描述十进制加法计数器

用 Verilog HDL 描述十进制计数器如下：

```
module    CNT_10 (CLRB, CLK, Q);
input     CLRB, CLK;
output    [3:0] Q;
```

```
reg[3:0] Q;
always      @(posedge CLK or negedge CLRB )
    if (! CLRB)      Q <= 0;
    else if(Q==9) Q <= 0;
    else              Q <= Q + 1;
endmodule
```

重点提示 以上程序十进制同步加法计数器，具有一定的通用性，并且通过适当修改程序部分语句，还可以设计出任意进制的计数器。例如，要设计前面介绍的八进制同步加法计数器，则修改如下：

```
module     CNT_8     (CLRB, CLK, Q );
input      CLRB, CLK;
output     [2:0] Q;
reg   [2:0] Q;
always     @(posedge CLK or negedge CLRB)
    if (! CLRB)      Q<=0;
    else if(Q==7) Q<= 0;
    else              Q<=Q+1;
endmodule
```

(3)十进制加法计数器仿真

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 CNT_10。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，在出现文本编辑窗口。输入以上程序，将其保存为 CNT_10. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 CLRB，CLK，Q0，Q1，Q2，Q3，锁定完毕单击 OK 即可，如图 8-107 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

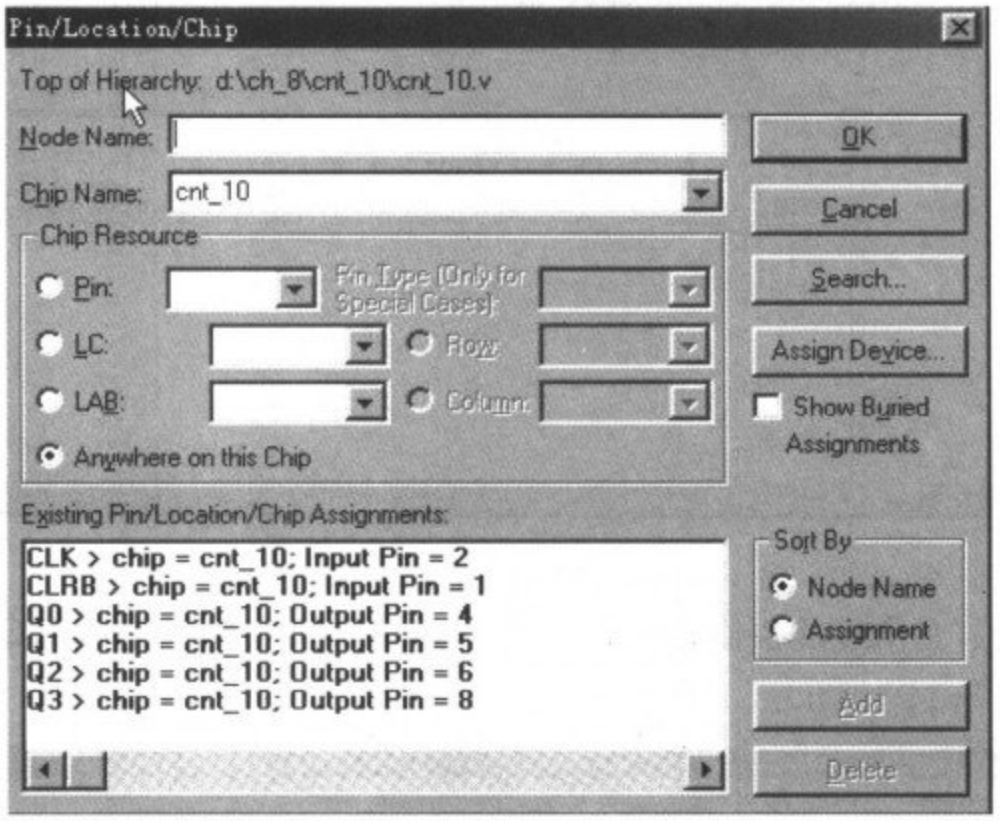


图 8-107　十进制计数器引脚锁定情况

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 CLRB，CLK，Q0，Q1，Q2，Q3 信号加入 SNF 文件中，如图 8-108 所示。

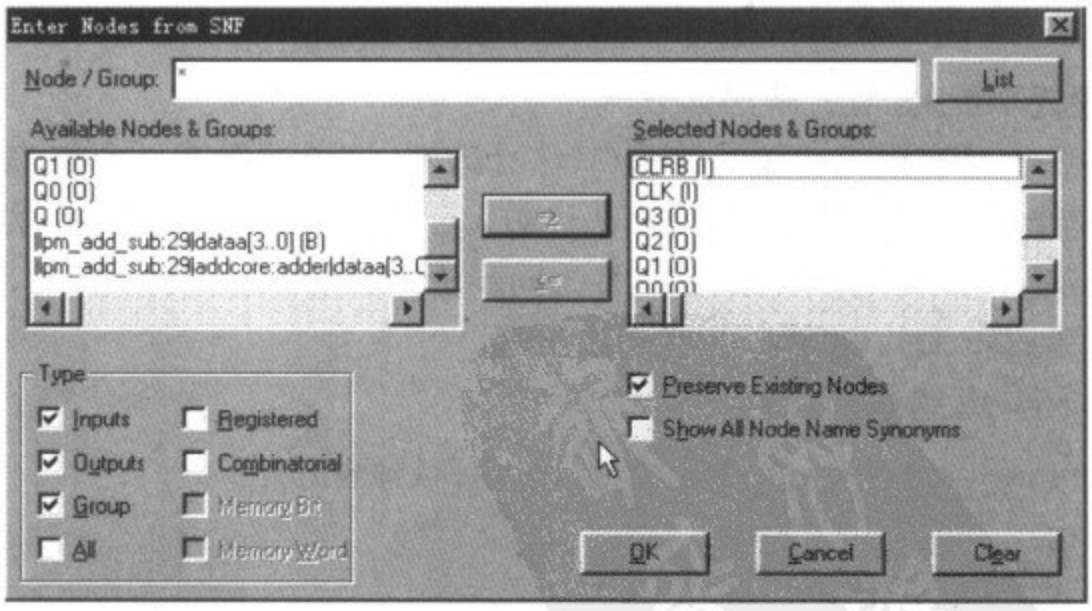


图 8-108　十进制计数器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 CLK，点击 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 0，Multiplied By 设为 20。选中 CLRB，用拖动的方法将开始的 0～1μs 设置为低电平，其余时段设置为高电平。单击保存按钮，将文件保存为 CNT_10. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-109 所示。

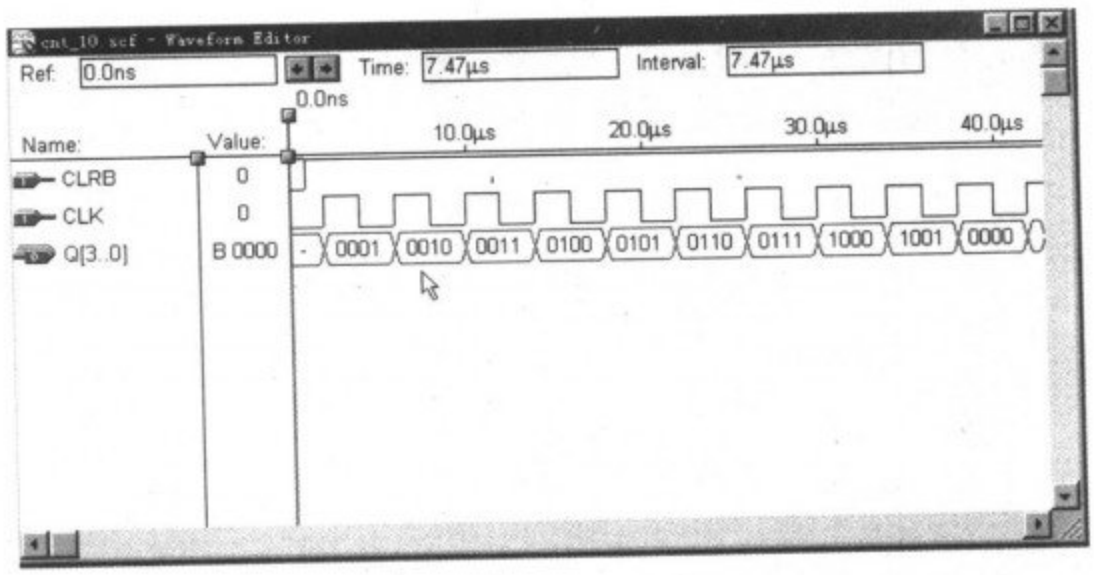


图 8-109　十进制计数器仿真波形图

右击 Q[3:0],在出现的快捷菜单中选择 upgroup,将 Q 分解为 Q0,Q1,Q2,Q3;则展开后的波形如图 8-110 所示。

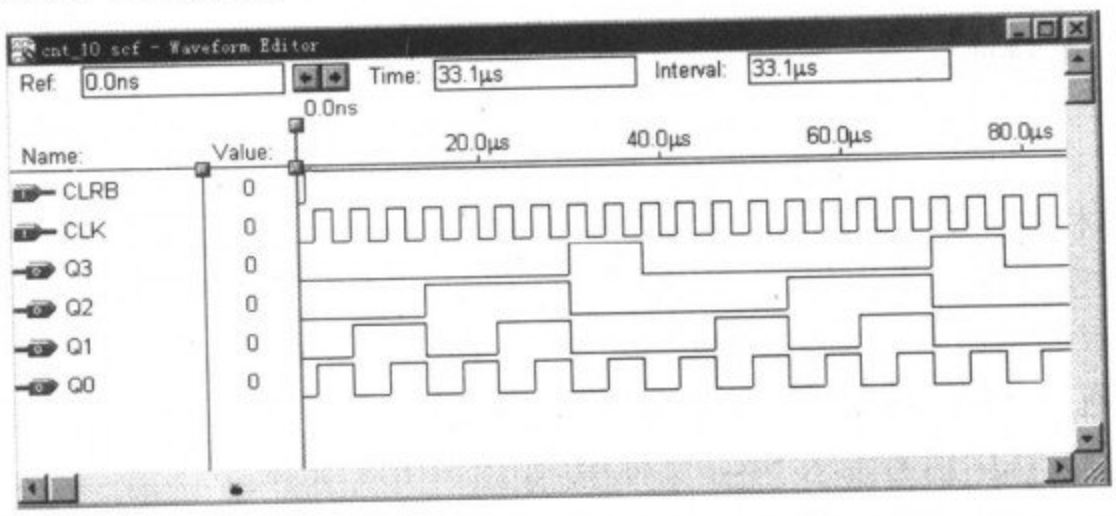


图 8-110　展开后的十进制计数器波形

5. 具有预置功能的加法/减法计数器

下面描述一个具有预置功能的加法/减法计数器,该计数器有一个加/减控制端 UP_DOWN,当该控制端为高电平时,实现加法计数,当该控制端为低电平时,实现减法计数;LOAD 为同步预置端,高电平置数;CLRB 为同步清零端(低电平清零);D0、D1、D2、D3 为并行数据输入端;Q0、Q1、Q2、Q3 为状态输出端。

用 Verilog HDL 描述如下:

```
module UPDOWN_CNT(D,CLK,CLRB,LOAD,UP_DOWN,Q);
input[3:0] D;
input CLK,CLRB,LOAD;
input UP_DOWN;
output[3:0] Q;
reg[3:0] COUNT;
assign Q=COUNT;
always @(posedge CLK)
    begin
      if(! CLRB)          COUNT<=0;          //清零
      else if(LOAD)       COUNT<=D;          //置数
```

```
        else if(UP_DOWN)   COUNT<=COUNT+1;   //加法计数
        else               COUNT<=COUNT-1;   //减法计数
    end
endmodule
```

下面用 MAX+plusII 进行仿真，步骤如下：

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 UPDOWN_CNT。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，在出现文本编辑窗口。输入以上程序，将其保存为 UPDOWN_CNT. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 CLRB，CLK，LOAD，UP_DOWN，D0，D1，D2，D3，Q0，Q1，Q2，Q3，锁定完毕单击 OK 即可，如图 8-111 所示。注意将 CLK 定义在 2 脚或 83 脚(全局时钟脚)。

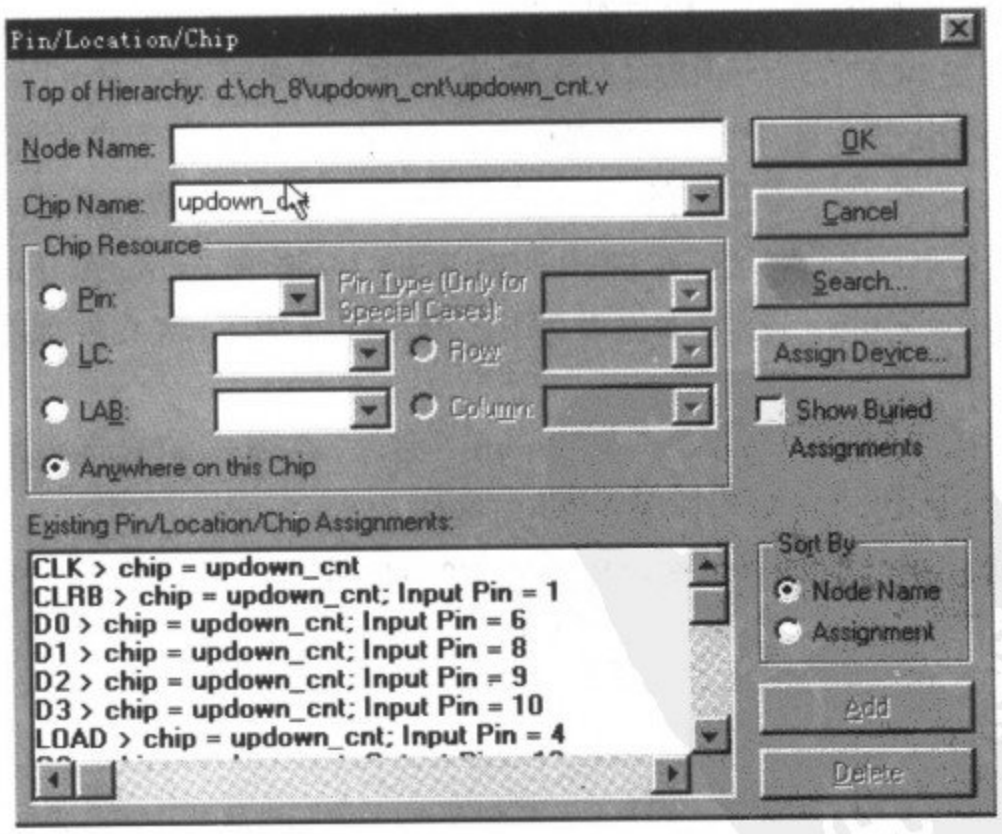


图 8-111　加法减法计数器引脚锁定情况

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 CLRB，CLK，LOAD，UP_DOWN，D0，D1，D2，D3，Q0，Q1，Q2，Q3 信号加入 SNF 文件中，如图 112 所示。

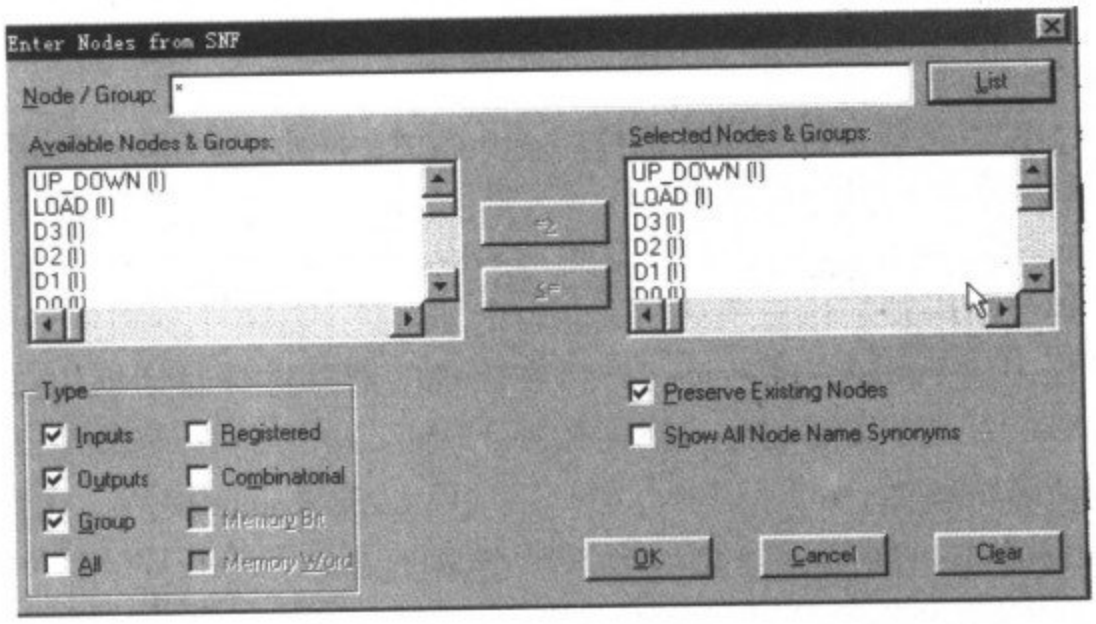


图 8-112　加法减法计数器仿真端口列表

点击菜单命令 File→End Time，终止时间设置为 100μs。

选中 CLK，点击 按钮，在弹出的时钟设置对话框中，将 Starting Value 开始电平设为 0，Multiplied By 设为 20。选中 CLRB，用拖动的方法将开始的 0～1μs 设置为低电平，其余时段设置为高电平。选中 UP_DOWN，点击 ，将其设置为高电平（加法计数器），单击 LOAD，用拖动的方法将开始的 0～5μs 设置为高电平（置数），其余时段设置为低电平。选中 D[3:0]，点击 按钮，在弹出的对话框中，将 Starting Value 开始电平设为 0111（置入数 7），Increment By 增加值设为 0。单击保存按钮，将文件保存为 UPDOWN_CNT. scf 文件。

执行菜单栏中的 MAX+plusII→Simulator 命令，在弹出的对话框中，点击 Start 开始按钮，开始仿真，仿真的波形图如图 8-113 所示。

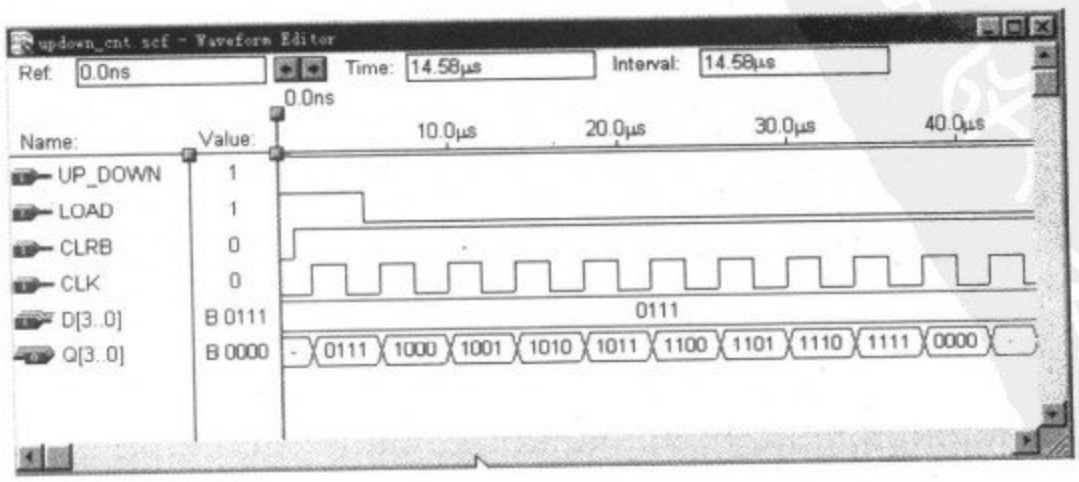


图 8-113　加法减法计数器仿真波形图

第九章　CPLD 实验与综合设计实例

通过前面几章的学习，我们已经对 CPLD 器件及相应的开发工具有了一定的了解。本章将结合 DP-MCU/Altera 综合仿真实验仪，进行实际演示和实验，以帮助读者尽快掌握 CPLD 设计开发的方法和步骤。紧接着安排了一些 CPLD 综合设计实例，完成这些设计后，读者将会感受到，开发 CPLD 不但十分简单，而且非常有趣。

第一节　CPLD 基本实验

一、LED 发光二极管实验

1. 闪烁灯

(1)简要分析

在 DP-MCU/Altera 综合仿真实验仪上，设有 8 个发光二极管 L0～L7，其原理图如图 9-1 所示，标号 LED0～LED7 通过跳线 JP2 分别与芯片的 30、31、33～37 和 39 引脚相连，从图中可以看出，8 个发光二极管采用共阳接法，即 8 个发光二极管的正极接电源 Vcc，负极分别接 CPLD(EPM7128S)I/O 脚相连，只要 CPLD 的相应的 I/O 脚上输出低电平 0，就可以点亮相应的发光二极管，例如，要点亮 L0，则需要 CPLD 的 30 脚输出低电平。

如果想要灯进行闪烁，只需在 CPLD 的相应引脚上周期性地输出高电平和低电平即可。为便于观察，闪烁频率可定在 1Hz 左右(即 1s 闪烁一次)。为了产生 1Hz 的时钟脉冲，可以使用计数器对时钟脉冲 CLK 进行计数，每来一个时钟脉冲 CLK，计数器就加 1。而每当判断出计数器达到某个数值时，就使得灯 LED0-LED7 的亮灭反转一次，即周期性地输出高电平“1”和低电平“0”。

在 DP-MCU/Altera 综合仿真实验仪上有一个有源脉冲发生电路，如图 9-2 所示。时钟脉冲 CLK 经过跳线与 CPLD 的 83 脚相连。图中的 Y1 是可插拔更换的，用户可根据实际需要更换(1～24)MHz 之间的晶振，这里选用 11.05926MHz 晶振。

如果把 11.592MHz 的时钟进行 5529630 分频(可用计数器统计分频次数)，变为 2Hz(即周期为 0.5s)，然后，使 LED0～LED7 输出端每 0.5s 取反一次，则 LED0～LED7 可输出 1Hz 的信号(即周期为 1s)。

(2)用 Verilog HDL 描述

```
module LED_LIGHT (CLK,LED);
input CLK;
output[7:0 ]LED;
reg[7:0 ]LED;
```

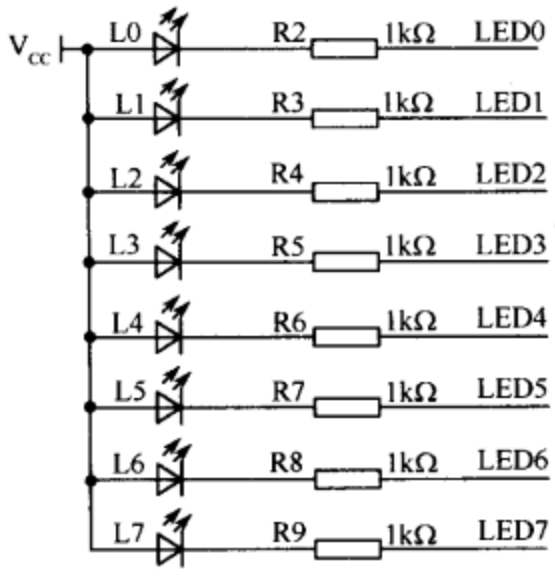


图 9-1　发光二极管电路

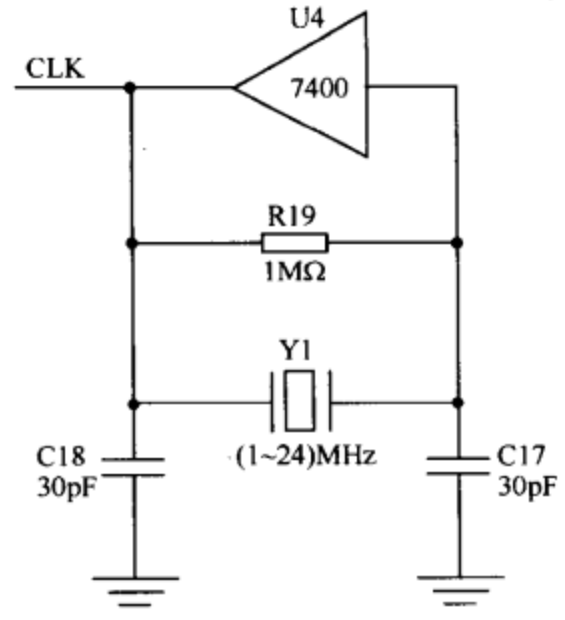


图 9-2　脉冲产生电路

```
reg[22:0 ]buffer;
always  @ (posedge CLK)
   begin
      buffer=buffer+1;
      if(buffer==23'd5529630)//11.05926MHz 进行 5529630 次分频后为 2Hz(0.5s)
      begin
         LED=~LED; //LED0~LED7 每 0.5s 反转一次,则周期为 1s,即频率为 1Hz
      end
   end
endmodule
```

(3)仿真与实验

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 LED_LIGHT。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 LED_LIGHT. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 CLK,LED0,LED1,LED2,LED3,LED4,LED5,LED6,LED7 进行引脚锁定,锁定情况如表 9-1 所示。

表 9-1　闪烁灯引脚锁定情况

Node name（脚名）	pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	pin(脚号)	Pin type(输入输出类型)
LED0	30	**output**	LED5	36	**output**
LED1	31	**output**	LED6	37	**output**
LED2	33	**output**	LED7	39	**output**
LED3	34	**output**	CLK	83	**input**
LED4	35	**output**			

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 CLK，LED0～LED7 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

⑥器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能完全正确。

2. 流水灯

(1)简要分析

下面用 DP-MCU/Altera 综合仿真实验仪做一个流水灯的设计。如果使得最右边的灯先亮，然后，通过移位，在其左侧的灯，由右向左依次点亮，而已经亮的灯又不灭，这样就形成了向左的流水灯，同理，如果由左向右依次点亮，则形成向右的流水灯。

初始状态时，8 个灯都不亮。每来一个时钟脉冲 CLK，计数器就加 1。每当判断出计数器中的数值达到 5529630 时，就会点亮一个灯，并进行移位。这样，依次点亮所有的灯，就形成了流水灯。而当 8 个灯都点亮时，需要一个操作使得所有的灯恢复为初始状态，即：灯都不亮。然后，再一次流水即可。如果是左移位，就出现向左流水的现象；反之，向右流水。

(2)用 Verilog HDL 描述

根据上面分析，可以得到下面的 Verilog-HDL 描述。

```
module LED_WATER (CLK, LED);
```

```
input CLK;
output [7:0] LED;
reg [8:0] LED;
reg [22:0] buffer;
initial      // 对 8 个灯进行初始化,使得 8 个灯都不亮
LED=9'b111111111;
always @(posedge CLK)
  begin
  buffer=buffer+1;
  if(buffer==23'd5529630)
    begin
    LED=LED<<1;// LED 向左移位,空闲位自动添 0 补位
    if (LED==9'b000000000)
    LED=9'b111111111; //如果 8 个灯全亮,恢复为初始状态,即 8 个灯都不亮
    end
  end
endmodule
```

重点提示 在程序中,如果将 LED 赋值为 8 位位宽值,则流水灯流到 7 个全部点亮时,会全部熄灭,然后再重新开始,达不到设计的目的;因此,在程序中要将 LED 赋为 9 位位宽值。

(3)仿真与实验

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name,在出现的新建项目对话框中键入设计项目名,这里起名为 LED_WATER。

②设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 LED_WATER. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 CLK,LED0,LED1,LED2,LED3,LED4,LED5,LED6,LED7 进行引脚锁定,锁定情况如表 9-2 所示。

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

表 9-2　闪烁灯引脚锁定情况

Node name（脚名）	pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	pin(脚号)	Pin type(输入输出类型)
LED0	30	**output**	LED5	36	**output**
LED1	31	**output**	LED6	37	**output**
LED2	33	**output**	LED7	39	**output**
LED3	34	**output**	**CLK**	83	**input**
LED4	35	**output**			

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 CLK，LED0～LED7 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

⑥器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能完全正确。

二、键盘实验

DP-MCU/Altera 综合仿真实验仪为用户准备了 8 个通用、独立的键盘 K0～K7，其电路原理图如图 9-3 所示。通过跳线，它们可以分别与 CPLD 芯片 EPM7128S 的 56～58、60、61、63～65 脚相连。一旦 K0～K7 中有键按入，则该节点由高电平变为低电平。

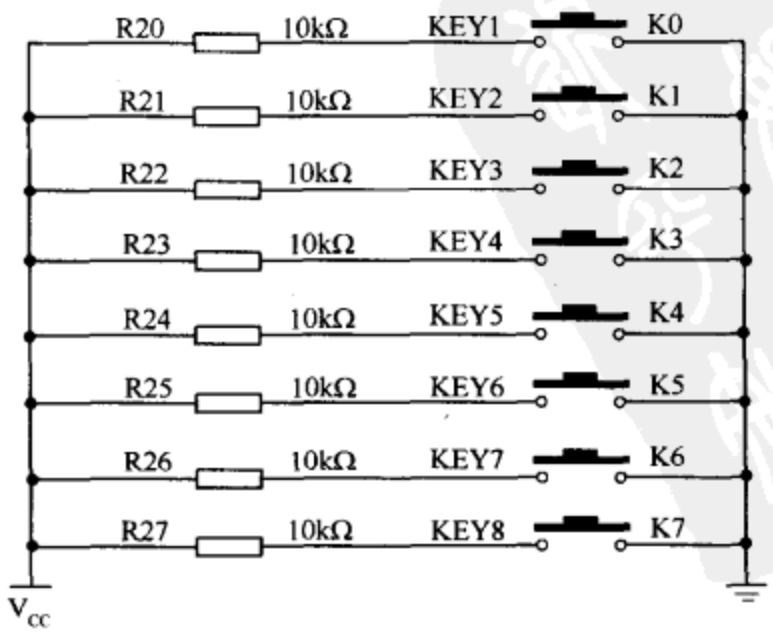


图 9-3　键盘电路

1. 键盘实验 1

实验要求：设计一个 8 人表决器，若超过四人赞成则通过。

(1)简要分析

用 key[7:0]表示 8 人的投票情况，用 pass 表示投票结果。8 人投票情况可用 DP-MCU/Altera 实验仪上的 8 个按键进行表示，若有 4 个以上键被同时按下，则表示通过。由于按键被按下时为 0，因此，0 代表赞成，即 key[i]=0 代表第 i 个人赞成，pass=0 表示表决通过，此时，LED0 发光二极管亮。

(2)用 Verilog HDL 描述

设计的程序如下：

```
module key_8(led0,key);
output[7:0] led;
input[7:0]  key;
reg[2:0]  sum;                    //统计赞成的人数
integer i;
reg pass;                         //定义寄存器
always@(key)
  begin
  sum=0;
  for(i=0;i<=7;i=i+1)
  if(! key[i])      sum=sum+1;
  if(sum[2]) pass=0;   //若超过 4 人赞成，则 pass=0
  else      pass=1;      //投票未通过
  end
assign led0=pass;          //输出寄存器内容，led0 亮，投票通过，灭，投票未通过
endmodule
```

重点提示 这里 sum 位宽为 3 位，第 1 位为 sum[0]、第 2 位为 sum[1]，第 3 位为 sum[2]。若赞成人数为 4 人或 4 人以上，则可能是 100、101、110、111 等几种情况，这几种情况中，sum[2]均为 1(最高位表示 sum[2])。因此，用 sum[2]=1 可作为是否超过 4 人的判断条件。

(3)仿真与实验

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 key_8。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 key_8. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family

栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 led0,key0, key1,key2, key3, key4, key5, key6, key7 进行引脚锁定,锁定情况如表 9-3 所示。

表 9-3 8 人表决器引脚锁定情况

Node name (脚名)	pin(脚号)	Pin type(输入输出类型)	Node name (脚名)	pin(脚号)	Pin type(输入输出类型)
led00	30	output	key4	61	input
key0	56	input	key5	63	input
key1	57	input	key6	64	input
key2	58	input	key7	65	input
key3	60	input			

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令,弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标,在弹出的快捷菜单中选择 EnterNodes form SNF... 命令,这时将出现如图仿真信号选择对话框。点击 List,将出现端口列表,你默认是选择全部,你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标,将 led0, key0, key1,key2, key3, key4, key5, key6, key7 信号加入 SNF 文件中进行仿真,看是否符合设计要求,若不符合,则修改程序的相关部分。

⑥器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆,并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上,打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令,这时将弹出编程命令对话框,选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在,就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试,设计程序的功能完全正确。

2. 键盘实验 2

实验要求:在 **DP-MCU/Altera** 实验仪完成对 **K0～K7** 的 8 个按键的识别,一旦有键被按下,则判断出键号,并用相应的发光二极管进行指示,例如,若 **K2** 被按下,则 **LED2** 点亮。

(1)简要分析

在 DP-MCU/Altera 实验仪上,8 个按键分别与 CPLD 芯片 EPM7128S 的 56～58、60、61、63～65 脚相连。一旦 K0～K7 中有键按入,则该节点由高电平变为低电平。可根据按键的高低电平变化进行设计。

(2)用 Verilog HDL 描述

用 Verilog HDL 描述如下：

```
module key_led(key,led);
input  [7:0] key;
output [7:0] led;
reg [7:0] led_reg;//定义寄存器,显示输出中间变量
reg [7:0] buffer; //定义寄存器,按键输入中间变量
always @ (key)
    begin
    buffer=key;
    case(buffer)
        8'b11111110:led_reg=8'b11111110; //K0 按下,LED0 亮
        8'b11111101:led_reg=8'b11111101; //K1 按下,LED1 亮
        8'b11111011:led_reg=8'b11111011; //K2 按下,LED2 亮
        8'b11110111:led_reg=8'b11110111; //K3 按下,LED3 亮
        8'b11101111:led_reg=8'b11101111; //K4 按下,LED4 亮
        8'b11011111:led_reg=8'b11011111; //K5 按下,LED5 亮
        8'b10111111:led_reg=8'b10111111; //K6 按下,LED6 亮
        8'b01111111:led_reg=8'b01111111; //K7 按下,LED7 亮
        default:led_reg=8'b11111111;      //否则,LED 全部熄灭
        endcase
    end
assign led=led_reg;
endmodule
```

(3)仿真与实验

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 key_led。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 key_led. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 led0，led1，led2，led3，led4，led5，led6，led7，key0，key1，key2，key3，key4，key5，key6，key7 进行引脚锁定，锁定情况如表 9-4 所示。

表 9-4　按键识别电路引脚锁定情况

Node name（脚名）	pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	pin(脚号)	Pin type(输入输出类型)
led0	30	**output**	key0	56	**input**
led1	31	**output**	key1	57	**input**
led2	33	**output**	key2	58	**input**
led3	34	**output**	key3	60	**input**
led4	35	**output**	key4	61	**input**
led5	36	**output**	key5	63	**inputZ**
led6	37	**output**	key6	64	**input**
led7	39	**output**	key7	65	**input**

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 led0，led1，led2，led3，led4，led5，led6，led7，key0，key1，key2，key3，key4，key5，key6，key7 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

⑥器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能完全正确。

三、数码 LED 显示器实验

数码 LED 显示器是应用较广的一种显示器件，内部是 8 只发光二极管，分别记为 a、b、c、d、e、f、g、dp(h)，其中除 dp(h)制成圆形用以表示小数点外，其余 7 只全部制成条形，并排列成“8”字形状。每只发光二极管都有一根电极引到外部引脚上，而另外一根电极全部连接在一起，引到外引脚，称为公共极(COM)。

LED 显示器有共阳和共阴两种类型。把各个发光二极管的阳极都连在一起，从 COM 端引出，阴极分别从其他 8 根引脚引出，为共阳结构。使用时，公共阳极接+5V，这样，阴极端输入低电平的发光二极管就导通点亮，而输入高电平的段则不能点亮。把各个

发光二极管的阴极都接在一起，从 COM 端引出 ，阳极分别从其他 8 根引脚引出，为共阴结构。使用时，公共阴极接地，这样，阳极端输入高电平的发光二极管就导通点亮，而输入低电平的段则不能点亮。

DP-MCU/Altera 实验仪上有 4 位共阳 LED 数码管，其标号分别为 W1～W4，其硬件原理图如图 9-4 所示。

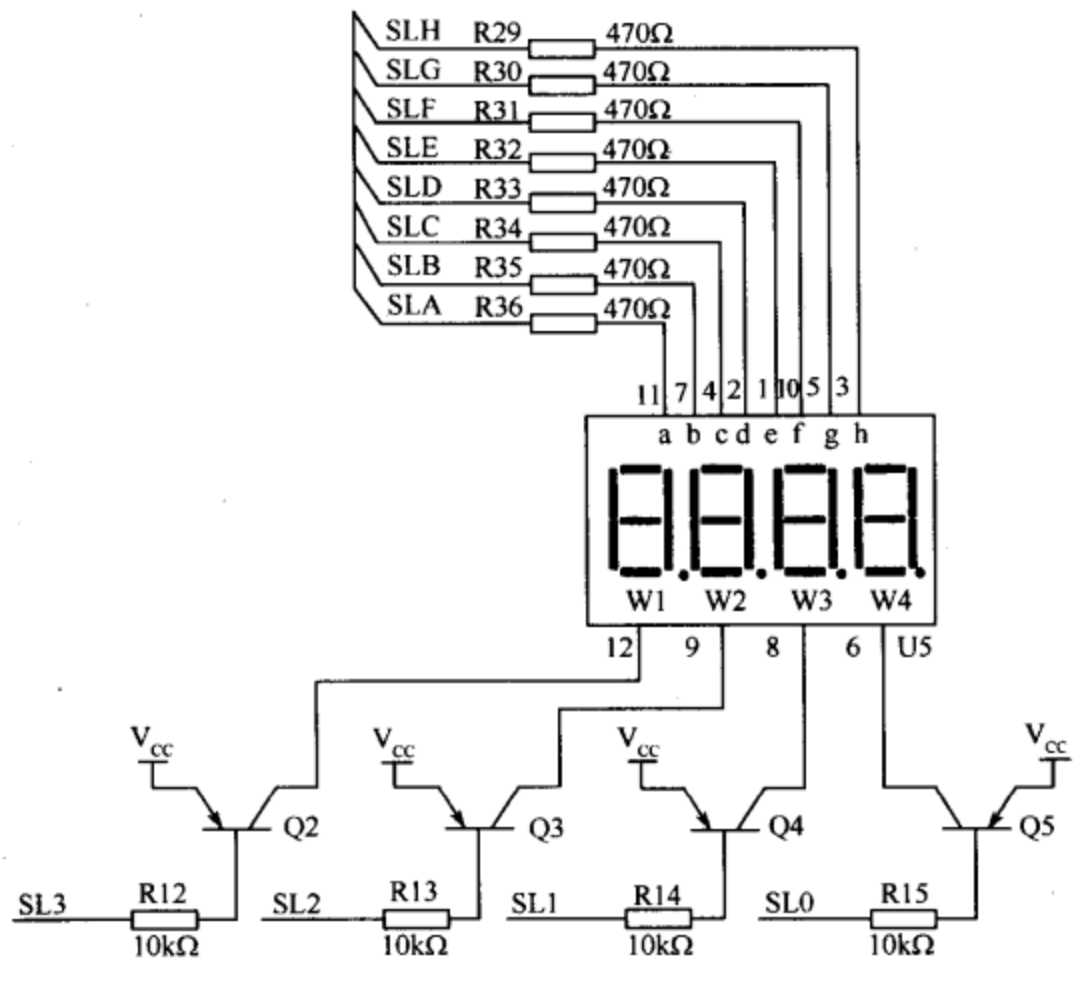


图 9-4　数码管 LED 显示电路

可以看出，段码线 SLA—SLH 通过跳线 JP2 分别与芯片的 40、41、44～46、48～50 引脚相连，而位码线 SL3～SL0 通过跳线 JP2 分别与芯片的 51、52、54、55 引脚相连。这样只需在相应的时刻，在段码和位码线上输出相应的段码和位码，即可完成 LED 数码管显示。

1. 静态显示实验

实验要求：采用静态显示方法，在 **DP-MCU/Altera** 实验仪上使 **4** 只数码 **LED** 管循环显示“**0000～9999**”10 个数。

(1)简要分析

所谓静态显示，就是当显示某一个数字时，代表相应笔划的发光二极管恒定发光，例如数码管的 a、b、c、d、e、f 笔段亮时，显示数字“0”；b、c 亮时显示“1”；a、b、d、e、g 亮时显示“2”等等。

对于 DP-MCU/Altera 实验仪，静态显示时。每位共阳数码管的公共端 COM 接在一起接正电压，即 CPLD 的位选线 SL3～SL0(51、52、54、55 脚)要输出低电平，使三极管 Q2～Q5导通。段选线 a，b，c，d，e，f，g，h 分别通过限流电阻与 CPLD 的段驱动端 40、41、44～46、48～50 引脚相连，限流电阻的阻值根据驱动电压和 LED 的额定电流确定。

静态显示的优点是显示稳定，在驱动电流一定的情况下显示的亮度高，缺点是使用元器件较多(每一位都需要一个驱动器，每一段都需要一个限流电阻)，功耗大，且占用较多的 I/O 口。

(2)用 Verilog HDL 描述

```
module led_jing (seg , sl , clk);                    //定义模块结构
output [7:0] seg;                                    //定义数码管段输出引脚
output [3:0] sl;                                     //定义数码管选择输出引脚
input clk;                                           //定义输入时钟引脚
reg [7:0] seg_reg;                                   //定义数码管段输出寄存器
reg [3:0] sl_reg;                                    //定义数码管选择输出寄存器
reg [3:0] disp_dat;                                  //定义显示数据寄存器
reg [27:0] count;                                    //定义计数器寄存器
always @(posedge clk)                                //定义 clk 信号下降沿触发
  begin
    count=count+1;                                   //计数器值加 1
  end
always
  begin
    sl_reg=4'b0000;
  end
always
  begin
    disp_dat = count[27:24];                         //显示数值
    if(disp_dat>9) disp_dat=4'b0000;                 //只显示 0～9 这 10 个数
  end
always @(disp_dat)                                   //显示译码输出
  begin
    case (disp_dat)                                  //选择输出数据
        4'h0: seg_reg = 8'hc0;                       //显示 0
        4'h1: seg_reg = 8'hf9;                       //显示 1
        4'h2: seg_reg = 8'ha4;                       //显示 2
        4'h3: seg_reg = 8'hb0;                       //显示 3
        4'h4: seg_reg = 8'h99;                       //显示 4
        4'h5: seg_reg = 8'h92;                       //显示 5
        4'h6: seg_reg = 8'h82;                       //显示 6
        4'h7: seg_reg = 8'hf8;                       //显示 7
        4'h8: seg_reg = 8'h80;                       //显示 8
        4'h9: seg_reg = 8'h90;                       //显示 9
        endcase
```

```
    end
  assign seg=seg_reg;                    //输出数码管译码结果
  assign sl=sl_reg;                      //输出数码管选择
  endmodule
```

重点提示 上面程序中，定义 count[27:24]为待显示的数值，刚开始计数时，计数的值为 0，因此，count[27:24]中的值为 4′b0000，下面简要分析当计数器 count[24]为 1 以后的变化情况：

当 count[27:24]=4b′0001 时

count[27:0]=0001,0000,0000,0000,0000,0000,0000=2^{24}=16777216

当 count[27:24]=4b′0010 时

count[27:0]=0010,0000,0000,0000,0000,0000,0000=2^{25}=33554432

也就是说，当 count[27:24]从 4b′0001(十进制数 1)变化到 4b′0010(十进制数 2)时，计数器 count 需要计 33554432-16777216=16777216 个数。

由于 CPLD 采用的时钟 clk 为 11.05926MHz，其周期约为 9×10^{-8} s，因此，计 16777216 个数需要的时间为：

$16777216\times9\times10^{-8}\approx1.5$s

当 count[27:24]=4b′0011 时

count[27:0]=0011,0000,0000,0000,0000,0000,0000=$2^{25}+2^{24}$

也就是说，当 count[27:24]从 4b′0010(十进制数 2)变化到 4b′0011(十进制数 3)时，计数器 count 需要计 $2^{25}+2^{24}-2^{25}=2^{24}$=16777216 个数。时间间隔也约为 1.5s。

同理可计算出从 3 变化到 4，从 4 变化到 5……时间间隔均为 1.5s 左右。即 4 个 LED 数码管按 1.5s 的时间间隔轮流显示 0000～9999 十个数。

(3)仿真与实验

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 led_jing。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 led_jing.v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 seg0，seg1，seg2，seg3，seg4，seg5，seg6，seg7，sl0，sl1，sl2，sl3，clk 进行引脚锁定，锁定情况如表 9-5 所示。

表 9-5　数码 LED 静态显示引脚锁定情况

Node name（脚名）	pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	pin(脚号)	Pin type(输入输出类型)
seg0	40	output	seg7	50	output
seg1	41	output	sl0	55	output
seg2	44	output	sl1	54	output
seg3	45	output	sl2	52	output
seg4	46	output	sl3	51	output
seg5	48	output	clk	83	input
seg6	49	output			

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 seg0，seg1，seg2，seg3，seg4，seg5，seg6，seg7，sl0，sl1，sl2，sl3，clk 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

⑥器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能完全正确。

2. 动态显示实验

实验要求：采用动态显示方法，在 **DP-MCU/Altera** 实验仪上使 **4** 只数码 **LED** 管显示 **“7128”**。

(1)简要分析

上面介绍的静态显示方法的最大缺点是使用元件多、引线多、电路复杂，而动态显示使用的元件少、引线少、电路简单。仅从引线角度考，静态显示从显示器到控制电路的基本引线数为“段数×位数”，而动态显示从显示器到控制电路的基本引线数为“段数+位数”。以 6 位显示为例，动态显示时的基本引线数为 7+6=13(无小数点)或 8+6=14(有小数点)，而静态显示的基本引线数为 7×6=42(无小数点)或 8×6=48(有小数点)。在实际应用中，显示器与主电路往往不会安装在同一块电路板上，有时甚至有一段距离，因此静态显示的引线数大多会给实际安装、加工工艺带来困难。

动态显示所显示的若干位数是逐位轮流显示的，周而复始不断循环，只要轮流的速度足够快(每秒轮流 50 次以上)，由于人眼“视觉暂留”的特性，感觉不到显示器的闪动，所看到的是连续显示一组数字。对于 DP-MCU/Altera 实验仪上 4 只共阳 LED 数码管，若采用动态显示方式，则位选端 SL0、SL1、SL2、SL3 的时序关系波形图如图 9-5 所示。

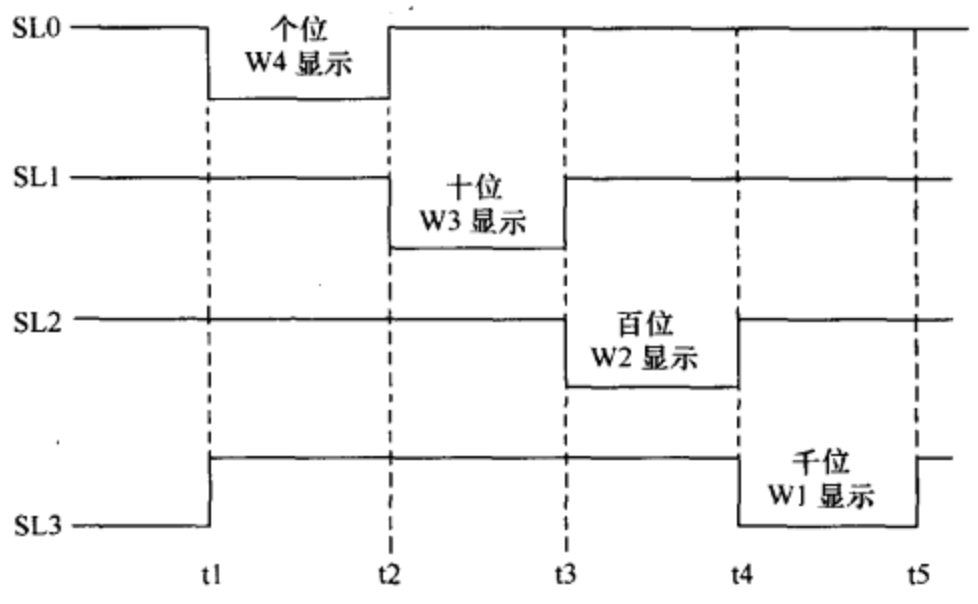


图 9-5　动态显示时序关系图

在 t1～t2 期间，CPLD 输出的个位段码(40、41、44～46、48～50 脚)加到 4 只数码 LED 的 a,b,c,d,e,f,g,h 端；在这期间，CPLD 的位选端 SL3～SL0(接 CPLD 的 51、52、54、55 脚)输出数据 4b′1110，个位控制信号 SL0 低电平，三极管 Q5 饱和导通，个位 W4 数码 LED 阳极接 Vcc，使个位数码 LED 显示器相应的字段点亮，显示个位数；而控制信号 SL1、SL2、SL3 为高电平，故三极管 Q2、Q3、Q4 截止，使 W1、W2、W3 数码 LED 的阳极不能接向 Vcc，因此 W1、W2、W3 位的 LED 显示器不亮。

在 t2～t3 期间，CPLD 输出十位的段码加到 4 只 LED 数码管的阳极 a,b,c,d,e,f,g,h 端；相应地，位选端输出数据 4b′1101，使十位控制信号 SL1 为低电平，而 SL0、SL2、SL3 为高电平，故只有三极管 Q4 饱和导通，而 Q2、Q3、Q5 截止，即只有十位的数码 LED 点亮。

同理在 t3～t4 期间，只有百位 LED 点亮，在 t4～t5 期间，只有千位 LED 点亮。此规律周而复始不断循环，就使得个位、十位、百位、千位 4 个 LED 依次轮流循环显示，这就是动态扫描显示的原理，控制信号 SL0、SL1、SL2、SL3 称为“位扫描”信号。

从以上分析可以看到，动态扫描显示方式可以十分有效地减少元件、减少引线，尤其是在显示位数较多时，其优越性就更为突出，因此在 CPLD 显示电路中，大多采用这种方式。

(2)用 Verilog HDL 描述

```
module led_dong(seg , sl , clk);                    //定义模块结构
output  [7:0] seg;                                  //定义数码管段输出引脚
output  [3:0] sl;                                   //定义数码管选择输出引脚
input clk;                                          //定义输入时钟引脚
reg  [7:0] seg_reg;                                 //定义数码管段输出寄存器
reg  [3:0] sl_reg;                                  //定义数码管选择输出寄存器
reg  [3:0] disp_dat;                                //定义显示数据寄存器
reg  [27:0] count;                                  //定义计数器寄存器
always  @(posedge clk)                              //定义 clock 信号下降沿触发
```

```
    begin
      count=count+1;                    //计数器值加 1
    end
  always  @(count[16:15])               //定义显示数据触发事件
    begin
      case  (count[16:15])              //选择扫描显示数据
        2'b00: disp_dat =4'b1000;       //显示个位数值 8
        2'b01: disp_dat =4'b0010;       //显示十位数值 2
        2'b10: disp_dat =4'b0001;       //显示百位数值 1
        2'b11: disp_dat =4'b0111;       //显示千位数据 7
      endcase
      case  (count[16:15])              //选择数码管显示位
        2'b00: sl_reg = 4'b1110;        //选择个位数码管
        2'b01: sl_reg = 4'b1101;        //选择十位数码管
        2'b10: sl_reg = 4'b1011;        //选择百位数码管
        2'b11: sl_reg = 4'b0111;        //选择千位数码管
      endcase
    end
  always  @(disp_dat)                   //显示译码输出
    begin
      case  (disp_dat)                  //选择输出数据
        4'h0: seg_reg = 8'hc0;          //显示 0
        4'h1: seg_reg = 8'hf9;          //显示 1
        4'h2: seg_reg = 8'ha4;          //显示 2
        4'h3: seg_reg = 8'hb0;          //显示 3
        4'h4: seg_reg = 8'h99;          //显示 4
        4'h5: seg_reg = 8'h92;          //显示 5
        4'h6: seg_reg = 8'h82;          //显示 6
        4'h7: seg_reg = 8'hf8;          //显示 7
        4'h8: seg_reg = 8'h80;          //显示 8
        4'h9: seg_reg = 8'h90;          //显示 9
        4'ha: seg_reg = 8'h88;          //显示 a
        4'hb: seg_reg = 8'h83;          //显示 b
        4'hc: seg_reg = 8'hc6;          //显示 c
        4'hd: seg_reg = 8'ha1;          //显示 d
        4'he: seg_reg = 8'h86;          //显示 e
        4'hf: seg_reg = 8'h8e;          //显示 f
      endcase
  end
```

```
assign seg=seg_reg;                    //输出数码管译码结果
assign sl=sl_reg;                      //输出数码管选择
endmodule
```

重点提示 下面对上面程序中的以下语句进行简要分析。

```
case (count[16:15])                    //选择数码管显示位
    2'b00: sl_reg = 4'b1110;           //选择个位数码管
    2'b01: sl_reg = 4'b1101;           //选择十位数码管
    2'b10: sl_reg = 4'b1011;           //选择百位数码管
    2'b11: sl_reg = 4'b0111;           //选择千位数码管
endcase
```

count[16:15]具有二个方面的作用：第一个作用是，在 **case** 语句中用来选择数码 LED，为 2b′00 时，选择个位 LED，为 2b′01 时选择十位 LED，为 2b′10 时选择百位 LED，为 2b′11 时选择千位 LED。第二个作用是用来提供动态扫描轮流扫描时间间隔。

当计数器 count[16:15]为 2b′01 时

count[27:0]=0000,0000,0000,1000,0000,0000,0000=2^{15}=32768

当 count[16:15]=2b′10 时

count[27:0]=0000,0000,0001,0000,0000,0000,0000=2^{16}=65536

也就是说，当 count[16:15]从 2b′01 变化到 2b′10 时，计数器 count 需要计 65536－32768=32768 个数。

由于 CPLD 采用的时钟 clk 为 11.05926MHz，其周期约为 9×10^{-5} ms，因此，计 32768 个数需要的时间为：

$32768\times9\times10^{-5}\approx3$ms

也就是说，扫描一位 LED 显示器需要 3ms，扫描四个 LED 显示器则需要 4×3=12(ms)，即动态扫描时间为 12ms。由于人眼的视觉暂留现象，这个扫描速度看不出数码 LED 的闪烁现象。

请读者实验一下，如果将程序中的 count[16:15]改为 count[18:17]，会发现有明显的闪烁现象。这是因为动态扫描周期太少，频率太低。

(3)仿真与实验

①新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 led_dong。

②设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 led_dong. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

③器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的

show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 seg0，seg1，seg2，seg3，seg4，seg5，seg6，seg7，sl0，sl1，sl2，sl3，clk 进行引脚锁定，锁定情况如表 9-6 所示。

表 9-6　数码 LED 静态显示引脚锁定情况

Node name（脚名）	pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	pin(脚号)	Pin type(输入输出类型)
seg0	40	**output**	seg7	50	**output**
seg1	41	**output**	sl0	55	**output**
seg2	44	**output**	sl1	54	**output**
seg3	45	**output**	sl2	52	**output**
seg4	46	**output**	sl3	51	**output**
seg5	48	**output**	clk	83	***input***
seg6	49	**output**			

④器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

⑤器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 seg0，seg1，seg2，seg3，seg4，seg5，seg6，seg7，sl0，sl1，sl2，sl3，clk 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

⑥器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能完全正确。

四、音响实验

由于蜂鸣器具有控制简单、声响悦耳动听的特性，在工程项目中常用做人机接口的重要输出设备，用以发出语音提示信息，使系统更加完善。蜂鸣器有交流和直流二种。直流蜂鸣器驱动简单，一旦在引脚上加入直流电源，就会发出一定频率的声音，此时声音的音调和音量是固定的；而交流蜂鸣器在这方面则显得较灵活，输入的声音信号的频率和音长可由用户进行控制，因此输出的声音更逼真、更悦耳。DP-MCU/Altera 实验仪上安放了一台交流蜂鸣器。由于一般 I/O 口的驱动能力有限，在此采用了三极管 Q1 来驱动蜂鸣

器，其硬件原理图如图 9-6 所示。BUZZ 通过一个跳线与芯片 EPM7128S 的 70 引脚相连。当 BUZZ 输出高电平时，蜂鸣器不响；而当 BUZZ 输出低电平时，蜂鸣器发出响声。只要控制 BUZZ 输出高低电平的时间，就可以让蜂鸣器发出悦耳的音乐。

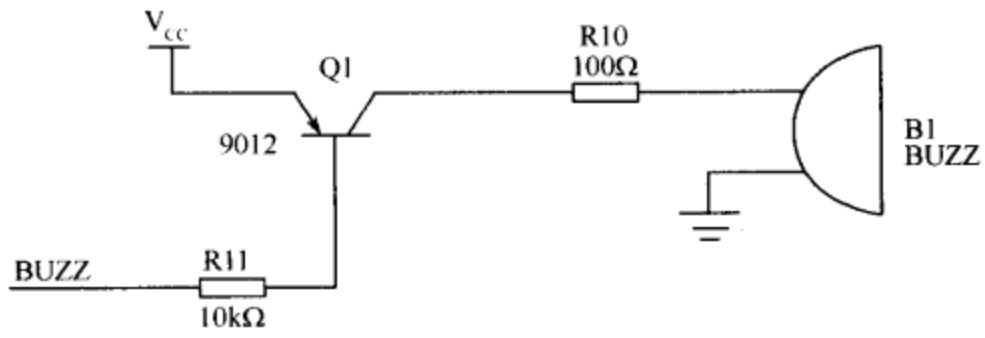


图 9-6　蜂鸣器原理图

实验要求：在 **DP-MCU/Altera** 实验仪上做以下实验：按下 **K0～K7** 的 8 个键，能分别发出“中音 **1**、中音 **2**、中音 **3**、中音 **4**、中音 **5**、中音 **6**、中音 **7**、高音 **1**”这 **8** 个音。

1. 简要分析

每个音名，都有一个固定的振动频率，频率的高低决定了音调的高低。音乐的十二平均率规定：每两个八度音（如简谱中的中音 1 与高音 1）之间的频率相差一倍。在两个八度音之间又可分为十二个半音，每两个半音的频率比为$\sqrt[12]{2}$。另外，音名 A（简谱中的低音 6）的频率为 440Hz，音名 B（简谱中的音 7）到 C（简谱中的音 1）之间、E（简谱中的音 3）到 F（简谱中的音 4）之间为半音，其余为全音。由此可以计算出简谱中从低音 1 至高音 1 之间每个音名对应的频率，见表 9-7 所示。

表 9-7　简谱中的音名与频率的关系

音名	频率/Hz	音名	频率/Hz	音名	频率/Hz
低音 1	261.6	中音 1	523.3	高音 1	1046.5
低音 2	293.7	中音 2	587.3	高音 2	1174.7
低音 3	329.6	中音 3	659.3	高音 3	1318.5
低音 4	349.2	中音 4	698.5	高音 4	1396.9
低音 5	392	中音 5	784	高音 5	1568
低音 6	440	中音 6	880	高音 6	1760
低音 7	493.9	中音 7	987.8	高音 7	1975.5

在 DP-MCU/Altera 实验仪上，CPLD 的 70 脚经过三极管驱动一个无源蜂鸣器，构成一个简单的音响电路，因此，只要有了某个音的频率数，就能产生出这个音来。

所有不同频率的信号都是从同一个基准频率分频得到的。由于音阶频率多为非整数，而分频系数又不能为小数，因此必须将计算得到的分频系数四舍五入取整。DP-MCU/Altera 实验仪上基准频率为 11.05926MHz，为了减小输出的偶次谐波分量，最后输出到扬声器的波形应为对称方波，因此在到达扬声器之前有一个二分频的分频器。据此，可求出不同音的分频系数，例如，中音 1 的频率为 523Hz，则其分频系数为：

11059260÷523.3÷2＝10566

采用同样的方法可以求出不同音名的分频系数，如表 9-8 所示。

表 9-8　不同音名的分频系数和预置数

音名	分频系数	音名	分频系数	音名	分频系数
低音 1	21137	中音 1	10566	高音 1	5283
低音 2	18827	中音 2	9415	高音 2	4707
低音 3	16776	中音 3	8339	高音 3	4193
低音 4	15835	中音 4	7916	高音 4	3958
低音 5	14106	中音 5	7053	高音 5	3526
低音 6	12567	中音 6	6283	高音 6	3141
低音 7	11196	中音 7	5597	高音 7	2792

2. 用 Verilog HDL 描述

```
module song_1( key,clk,song_out,led);
input clk;
input [7:0] key;
output song_out;
output [7:0] led;
reg song_reg;
reg [13:0] count, delay;
reg [7:0] key_reg;
always @ (posedge clk)
  begin
    count=count+1;
    if((count==delay)&(! (delay==22'd16383)))
      begin
        count=22'd0;
        song_reg=! song_reg;
      end
  end
always @ (key)
  begin
    key_reg=key;
    case(key_reg)
    8'b11111110: delay=20'd10566;     //中音 1
    8'b11111101: delay=20'd9415;      //中音 2
    8'b11111011: delay=20'd8339;      //中音 3
    8'b11110111: delay=20'd7916;      //中音 4
    8'b11101111: delay=20'd7053;      //中音 5
    8'b11011111: delay=20'd6283;      //中音 6
    8'b10111111: delay=20'd5597;      //中音 7
```

```
        8'b01111111: delay=20'd5283;        //高音 1
        default: delay=20'd16383;
        endcase
      end
    assign song_out=song_reg;
    assign led=key_reg;
    endmodule
```

重点提示 “中音 1、中音 2、中音 3、中音 4、中音 5、中音 6、中音 7、高音 1”8 个音中，最大的分频系数为是“中音 1”，为 10566，故可采用 14 位二进制计数器分频可满足需要（最大计数 $2^{14}-1=16383$）。

在程序中，有以下语句：

```
if((count==delay)&(!(delay==22'd16383)))
  begin
  count=22'd0;
  song_reg=! song_reg;
  end
```

以上语句的意义是：如果计数值 count 等于 delay（在 **case** 语句中代表各音的分频系数），并且 delay 不等于 16383 最大计数值，则计数器 count 清零，二分频输出，产生方波信号，使蜂鸣器发声。

需要说明的是，计数器的模不要选得太大，如果将程序中第 7 行语句

```
    reg [13:0] count, delay;
```

改为

```
    reg [27:0] count, delay;
```

将程序中的第 12 行语句

```
    if((count==delay)&(!(delay==22'd16383)))
```

改为

```
    if((count==delay)&(!(delay==22'hfffff)))
```

实验会发现，按键按下 1s 后，蜂鸣器才发声，这是因为计数器的模设置过大的缘故。

3. 仿真与实验

(1)新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 song_1。

(2)设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 song_1. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

(3)器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中：Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮

加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 led0，led1，led2，led3，led4，led5，led6，led7，key0，key1， key2， key3，key4，key5，key6，key7，clk 进行引脚锁定，锁定情况如表 9-9 所示。

表 9-9　音响电路引脚锁定情况

Node name（脚名）	Pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	pin(脚号)	Pin type(输入输出类型)
led0	30	output	key1	57	input
led1	31	output	key2	58	input
led2	33	output	key3	60	input
led3	34	output	key4	61	input
led4	35	output	key5	63	input
led5	36	output	key6	64	input
led6	37	output	key7	65	input
led7	39	output	clk	83	input
key0	56	input			

(4)器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

(5)器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 led0，led1，led2，led3，led4，led5，led6，led7，key0，key1，key2，key3，key4，key5，key6，key7，clk 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

(6)器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能完全正确。

第二节　CPLD 综合设计实例

下面介绍一些多功能数字电路的综合设计实例，进一步讨论 Verilog HDL 的开发实践，这些设计实例中的文件，可以下载到 DP-MCU/Altera 实验仪上，进行实际效果的验证。

一、乐曲演奏电路

要求：设计一个乐曲演奏电路，演奏《两只老虎》的片段。其曲谱如图 9-7 所示。

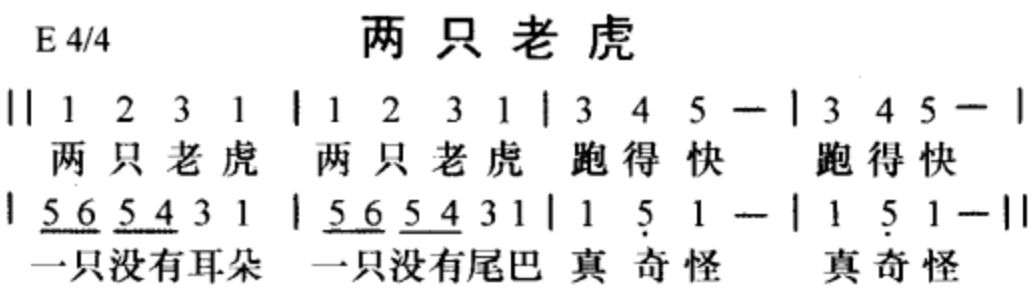


图 9-7　两只老虎曲谱

1. 设计原理

乐曲演奏的原理是这样的：组成乐曲的每个音符的频率值（音调）及其持续的时间（音长）是乐曲能连续演奏所需的两个基本数据，因此只要控制输出到扬声器的激励信号的频率的高低和持续的时间，就可以使扬声器发出连续的乐曲声。

（1）音调的控制

乐曲是由不同音符编制而成的，音符中有 7 个音名：C、D、E、F、G、A、B。它们分别唱做哆、唻、咪、法、嗦、啦、唏。每个音名，都有一个固定的振动频率，频率的高低决定了音调的高低。各音名的频率参见前面的表 9-7 所示。

所有不同频率的信号都是从同一个基准频率分频得到的。这里采用基准频率为 11.05926MHz的晶振，为了减小输出的偶次谐波分量，最后输出到扬声器的波形应为对称方波，因此在到达扬声器之前有一个二分频的分频器。据此，可求出不同音的分频系数，例如，中音 1 的频率为 523Hz，则其分频系数为：

11059260÷523.3÷2=10566

采用同样的方法可以求出《两只老虎》片段中各音名的分频系数，如表 9-10 所示。

表 9-10　《两只老虎》片段中各音名的分频系数和预置数

音名	分频系数	预置数	音名	分频系数	预置数
低音 5	14106	2277	中音 4	7916	8467
中音 1	10566	5817	中音 5	7053	9330
中音 2	9415	6968	中音 6	6283	10100
中音 3	8339	8044			

从表中可以看出，最大的分频系数为 14106，故采用 14 位二进制计数器（count）分频可满足需要（最大计数 $2^{14}-1=16383$）。在表中，除给出了分频比以外，还给出了对应于各个音阶频率时计数器不同的预置数。预置数等于 16383 减去相应的分频系数，对于不同的分频系数，只要加载不同的预置数即可。

此外，对于乐曲中的休止符，可将分频系数设为 0，即加载预置数 16383，此时，蜂鸣器不会发声。

（2）音长的控制

乐曲中的音符不单有音调的高低，还要有音的长短，如有的音要唱四分之一拍，有的音要唱二拍等。在节拍符号中，如用×代表某个音的唱名，×下面无短线为四分音符，有

一条短横线代表八分音符，有二条横线代表十六分音符，×右边有一条短横线代表二分音符，×右边有 3 条短横线代表全音符，有“.”的音符为符点音符。

如果提供一个 4Hz(0.25s)的时钟频率来产生四分音符的时长，则各节拍的音长如表 9-11 所示。

表 9-11　节拍的音长

节拍符号	×=	×-	×-·	×	×·	×-	×---
名称	十六分音符	八分音符	八分符点音符	四分音符	四分符点音符	二分音符	全音符
拍数	1/4 拍	半拍	3/4 拍	1 拍	1 又 1/2 拍	2 拍	4 拍
音长	0.0625s	0.125s	0.1875s	0.25s	0.375s	0.5s	1s

为了产生 4Hz 脉冲，可以把 11.592MHz 的时钟进行 1382407 分频(用计数器 CNT 统计分频次数)，变为 8Hz(即周期为 0.125s)，然后，使 4Hz 信号每 0.125s 取反一次，则变为 4Hz 的信号(即周期为 0.25s)。

控制音长是通过计数器预置数的停留时间来实现的，预置数停留的时间越长，则该音符演奏的时间也越长，本例演示的是《两只老虎》片段，最短的音符为四分音符(0.25s)，由于没有符点音符，因此，每个音符演奏的时间都是 0.25s 的整数倍。

另外，为了能使演奏循环进行，需另设一个时长计数器 cnt，当乐曲演奏完成时，以便能自动从头开始演奏。

2. 用 Verilog HDL 描述

```
module song_2(clk,song_out);
input clk;                          //clk 为 11.05926MHz 时钟
output song_out;                    //蜂鸣器输出端
reg[3:0] high,med,low;              //高、中、低音寄存器
reg[13:0] count,start;              //count 为 14 位计数器,start 为预置数
reg[7:0] cnt;                       //cnt 为 8 位计数器,用于循环演奏
reg song_out;
wire carry;                         //进位信号
reg clk_4;
reg[22:0]  CNT;                     //CNT 为 23 位计数器,用于产生 4Hz 信号
always @ (posedge clk)              //4Hz 信号产生电路
  begin
  CNT=CNT+1;
  if(CNT==23'd1382407)              //分频为 8Hz 信号
    begin
      CNT=23'h000000;
      clk_4=~clk_4;                 //产生 4Hz 信号
    end
  end
assign carry=(count==16383);
```

```
always @(posedge clk)
  begin
    if(carry) count<=start;              //计数计满时,重装预置数
    else count<=count+1;
  end
always @(posedge carry)
  begin
    song_out=~song_out;                  //2 分频产生方波信号
  end
always @(posedge clk_4)
  begin
    case({high,med,low})                          //分频比预置
    12'b000000000101: start=2277;                 //低音 5 预置数
    12'b000000010000: start=5817;                 //中音 1 预置数
    12'b000000100000: start=6968;                 //中音 2 预置数
    12'b000000110000: start=8044;                 //中音 3 预置数
    12'b000001000000: start=8467;                 //中音 4 预置数
    12'b000001010000: start=9330;                 //中音 5 预置数
    12'b000001100000: start=10100;                //中音 6 预置数
    endcase
  end
always @(posedge clk_4)
  begin
    if(cnt==63) cnt<=0;                           //计时,以实现循环演奏
else cnt<=cnt+1;
case(cnt)
      0: {high,med,low}=12'b000000010000;         //中音"1",2 个时钟节拍
      1: {high,med,low}=12'b000000010000;
      2: {high,med,low}=12'b000000100000;         //中音"2",2 个时钟节拍
      3: {high,med,low}=12'b000000100000;
      4: {high,med,low}=12'b000000110000;         //中音"3",2 个时钟节拍
      5: {high,med,low}=12'b000000110000;
      6: {high,med,low}=12'b000000010000;         //中音"1",2 个时钟节拍
      7: {high,med,low}=12'b000000010000;
      8: {high,med,low}=12'b000000010000;         //中音"1",2 个时钟节拍
      9: {high,med,low}=12'b000000010000;
      10:{high,med,low}=12'b000000100000;         //中音"2",2 个时钟节拍
      11:{high,med,low}=12'b000000100000;
      12:{high,med,low}=12'b000000110000;         //中音"3",2 个时钟节拍
```

```
13:{high,med,low}=12'b000000110000;
14:{high,med,low}=12'b000000010000;    //中音“1”,2 个时钟节拍
15:{high,med,low}=12'b000000010000;
16:{high,med,low}=12'b000000110000;    //中音“3”,2 个时钟节拍
17:{high,med,low}=12'b000000110000;
18:{high,med,low}=12'b000001000000;    //中音“4”,2 个时钟节拍
19:{high,med,low}=12'b000001000000;
20:{high,med,low}=12'b000001010000;    //中音“5”,4 个时钟节拍
21:{high,med,low}=12'b000001010000;
22:{high,med,low}=12'b000001010000;
23:{high,med,low}=12'b000001010000;
24:{high,med,low}=12'b000000110000;    //中音“3”,2 个时钟节拍
25:{high,med,low}=12'b000000110000;
26:{high,med,low}=12'b000001000000;    //中音“4”,2 个时钟节拍
27:{high,med,low}=12'b000001000000;
28:{high,med,low}=12'b000001010000;    //中音“5”,4 个时钟节拍
29:{high,med,low}=12'b000001010000;
30:{high,med,low}=12'b000001010000;
31:{high,med,low}=12'b000001010000;
32:{high,med,low}=12'b000001010000;    //中音“5”,1 个时钟节拍
33:{high,med,low}=12'b000001100000;    //中音“6”,1 个时钟节拍
34:{high,med,low}=12'b000001010000;    //中音“5”,1 个时钟节拍
35:{high,med,low}=12'b000001000000;    //中音“4”,1 个时钟节拍
36:{high,med,low}=12'b000000110000;    //中音“3”,2 个时钟节拍
37:{high,med,low}=12'b000000110000;
38:{high,med,low}=12'b000000010000;    //中音“1”,2 个时钟节拍
39:{high,med,low}=12'b000000010000;
40:{high,med,low}=12'b000001010000;    //中音“5”,1 个时钟节拍
41:{high,med,low}=12'b000001100000;    //中音“6”,1 个时钟节拍
42:{high,med,low}=12'b000001010000;    //中音“5”,1 个时钟节拍
43:{high,med,low}=12'b000001000000;    //中音“4”,1 个时钟节拍
44:{high,med,low}=12'b000000110000;    //中音“3”,2 个时钟节拍
45:{high,med,low}=12'b000000110000;
46:{high,med,low}=12'b000000010000;    //中音“1”,2 个时钟节拍
47:{high,med,low}=12'b000000010000;
48:{high,med,low}=12'b000000010000;    //中音“1”,2 个时钟节拍
49:{high,med,low}=12'b000000010000;
50:{high,med,low}=12'b000000000101;    //低音“5”,2 个时钟节拍
51:{high,med,low}=12'b000000000101;
```

```
        52:{high,med,low}=12'b000000010000;     //中音"1",4 个时钟节拍
        53:{high,med,low}=12'b000000010000;
        54:{high,med,low}=12'b000000010000;
        55:{high,med,low}=12'b000000010000;
        56:{high,med,low}=12'b000000010000;     //中音"1",2 个时钟节拍
        57:{high,med,low}=12'b000000010000;
        58:{high,med,low}=12'b000000000101;     //低音"5",2 个时钟节拍
        59:{high,med,low}=12'b000000000101;
        60:{high,med,low}=12'b000000010000;     //中音"1",4 个时钟节拍
        61:{high,med,low}=12'b000000010000;
        62:{high,med,low}=12'b000000010000;
        63:{high,med,low}=12'b000000010000;
        endcase
    end
endmodule
```

3. 实验验证

以上乐曲演奏程序可以在 DP-MCU/Altera 实验仪上进行验证，方法如下。

(1)新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 song_2。

(2)设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 song_2. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

(3)器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 clk，song_out 进行引脚锁定，锁定情况如表 9-12 所示。

表 9-12　数字钟引脚锁定情况

Node name（脚名）	Pin(脚号)	Pin type(输入输出类型)
clk	83	input
song_out	70	output

(4)器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

(5)器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标，将 clk，song_out 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

(6)器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能完全正确。

二、数字钟

要求：设计一个简单的数字钟，显示时间范围为 **00:00～59:59**，且具有清零功能，即按下清零键时，时钟回到 **00:00**。

1. 设计原理

数字钟主要由三部分组成：秒脉冲产生电路、显示部分和计时处理部分。

为了产生秒脉冲，可以把 11.592MHz 的时钟进行 5529630 分频(可用计数器统计分频次数)，变为 2Hz(即周期为 0.5s)，然后，使秒信号 second 每 0.5s 取反一次，则秒信号 second 为 1Hz 的信号(即周期为 1s)。

显示部分与前面介绍的数码 LED 显示原理相同。

在计时处理电路中，设置了一个秒处理寄存器 sec[7:0]和分处理寄存器 min[7:0]，它们都是 8 位寄存器，分别存放秒的低位和高位，以及分的低位和高位。其中，sec[3:0]存放秒信号的个位，sec[7:4]存放秒信号的十位，min[3:0]存放分信号的个位，min[7:4]存放分信号的十位。

秒信号处理的过程是：在秒信号(second)的上升沿，如果秒信号个位 sec[3:0]满 10，清 0 秒个位 sec[3:0]，同时秒的十位 sec[7:4]加 1；如果秒信号十位 sec[7:4]满 6，清 0 秒十位 sec[7:4]，同时向分产生进位信号 cn，即 cn 置 1。

分信号的处理过程是：在进位信号 cn 的上升沿，分的个位 min[3:0]加 1；如果分信号个位 min[3:0]满 10，清 0 分个位 min[3:0]，同时分的十位信号 min[7:4]加 1。如果分信号十位 min[7:4]满 6，清 0 分十位 min[7:4]。这里由于未设小时显示部分，所以，分信号不必考虑其进位情况。

这个数字钟功能比较简单，程序也不复杂，读者可以在此基础上进行修改，增加数字钟的功能，如暂停功能、校时功能和定时功能等。

2. 用 Verilog HDL 描述

```
module clock (clr_key,clk,segdat,sl);
```

```
input clr_key,clk;                          //clr_key 为清 0 键
output [7:0] segdat;                        //段码输出
output [3:0] sl;                            //位码输出
reg [22:0] count;                           //计数器
reg [7:0] sec,min;                          //秒和分处理寄存器
reg [7:0] segdat_reg;                       //段码寄存器
reg [3:0] sl_reg;                           //位码寄存器
reg [3:0] disp_dat;                         //显示数据寄存器
reg second;                                 //秒信号
reg cn;                                     //秒向分进位信号
always @ (posedge clk)                      //秒信号产生电路
  begin
  count=count+1;
  if(count==23'd5529630)                    //分频为 2Hz 信号
    begin
      count=23'h000000;
      second =~second;                      //产生 1Hz 信号
    end
  end
always @ (count[11:10])
    begin
      case(count[11:10])
      2'b00: disp_dat =sec[3:0];            //取秒的个位数据
      2'b01: disp_dat =sec[7:4];            //取秒的十位数据
      2'b10: disp_dat =min[3:0];            //取分的个位数据
      2'b11: disp_dat =min[7:4];            //取分的十位数据
      endcase
    end
always @ (disp_dat)
    begin
      case(disp_dat)
      4'h0: segdat_reg = 8'hc0;             //显示 0
      4'h1: segdat_reg = 8'hf9;             //显示 1
      4'h2: segdat_reg = 8'ha4;             //显示 2
      4'h3: segdat_reg = 8'hb0;             //显示 3
      4'h4: segdat_reg = 8'h99;             //显示 4
      4'h5: segdat_reg = 8'h92;             //显示 5
      4'h6: segdat_reg = 8'h82;             //显示 6
      4'h7: segdat_reg = 8'hf8;             //显示 7
```

```
        4'h8: segdat_reg = 8'h80;          //显示 8
        4'h9: segdat_reg = 8'h90;          //显示 9
        endcase
        if ((count[11:10]==2'b10)&second)
        segdat_reg=segdat_reg&8'b01111111;//小数点闪烁
      end
always @(count[11:10])
      begin
        case(count[11:10])
        2'b00:sl_reg=4'b1110;              //扫描最低位
        2'b01:sl_reg=4'b1101;              //扫描次低位
        2'b10:sl_reg=4'b1011;              //扫描次高位
        2'b11:sl_reg=4'b0111;              //扫描最高位
        endcase
   end
always @(posedge second)                   //秒处理
   begin
     if(!clr_key)                          //若按下清 0 键
        begin
          sec[7:0]=8'h0;                   //则秒寄存器清 0
          cn=0;                            //进位清 0
        end
      else
        begin
          cn=0;                            //因为秒满 60 后 cn 置 1,所以此处要清 0
          sec[3:0]=sec[3:0]+1;             //秒加 1
          if(sec[3:0]==4'd10)              //若秒个位为 10(加 1 后的值)
            begin
              sec[3:0]=4'd0;               //秒个位清 0
              sec[7:4]=sec[7:4]+1;         //秒十位加 1
              if(sec[7:4]==4'd6)           //若秒十位为 6(加 1 后的值)
                begin
                  sec[7:4]=4'd0;           //秒十位清 0
                cn=1;                      //秒向分的进位 cn 置 1
                end
            end
        end
      end
always @(posedge cn)                       //分处理
```

```
    begin
      if(!clr_key)                          //若按下清 0 键
        begin
          min[7:0]=8'h0;                    //则分寄存器清 0
        end
      else
        begin
          min[3:0]=min[3:0]+1;              //分加 1
          if(min[3:0]==4'd10)               //若分个位为 10(加 1 后的值)
            begin
              min[3:0]=4'd0;                //分个位清 0
              min[7:4]=min[7:4]+1;          //分十位加 1
              if(min[7:4]==4'd6)            //若分十位为 6(加 1 后的值)
                begin
                  min[7:4]=4'd0;            //分十位清 0
                end
            end
        end
      end
  assign segdat=segdat_reg;                 //将段码寄存器的值送段码输出
  assign sl=sl_reg;                         //将位码寄存器的值送位码输出
  endmodule
```

重点提示 程序中有以下语句：

```
if((count[11:10]==2'b10) &second)
segdat_reg=segdat_reg&8'b01111111;
```

其意义说明如下：如果 count[11:10]==2'b10(即分的个位)为 1，且 second 为 1 时，段码寄存器 segdat_reg 等于 segdat_reg 与 01111111 相与，即段码寄存器 segdat_reg 最高位(小数点位)为 0。对于共阳显示器，小数点每秒闪烁 1 次。

如果将以上语句改为：

```
if(count[11:10]==2'b10)
segdat_reg=segdat_reg&8'b01111111;
```

实验时会发现小数点始终显示，而不是每秒闪烁一次。请读者想一想，这是为什么？(提示：因为 second 在 0 和 1 之间每秒变化 1 次)。

3. 实验验证

以上数字钟程序可以在 DP-MCU/Altera 实验仪上进行验证，方法如下：

(1)新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 colck。

(2)设计输入

单击 File→new,新建文件时选择 Text Editor File 选项,点击 OK,出现文本编辑窗口。输入以上程序,将其保存为 colck. v 文件。单击菜单 File→Project→Set Project To Current File,把文件设为当前工程,至此,Verilog 输入完成。

(3)器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令,在出现的器件选择对话框中;Device Family 栏中选择 MAX7000S;然后在 Device 栏内选择 EPM7128SLC84-15 器件;单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉,否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令,此时会弹出引脚锁定对话框。输入 segdat0,segdat1,segdat2,segdat3,segdat4,segdat5,segdat6,segdat7,sl0,sl1,sl2,sl3,clr_key,clk 进行引脚锁定,锁定情况如表 9-13 所示。

表 9-13　数字钟引脚锁定情况

Node name (脚名)	Pin(脚号)	Pin type(输入输出类型)	Node name (脚名)	Pin(脚号)	Pin type(输入输出类型)
segdat0	40	**output**	segdat7	50	**output**
segdat1	41	**output**	sl0	55	**output**
segdat2	44	**output**	sl1	54	**output**
segdat3	45	**output**	sl2	52	**output**
segdat4	46	**output**	sl3	51	**output**
segdat5	48	**output**	clk	83	**input**
segdat6	49	**output**	clr_key	56	**input**

(4)器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令,这时将弹出编译命令对话框,单击 Start 按钮,即可编译相关的各项操作。

(5)器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令,弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标,在弹出的快捷菜单中选择 EnterNodes form SNF... 命令,这时将出现如图仿真信号选择对话框。点击 List,将出现端口列表,你默认是选择全部,你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“=>”图标,将 lsegdat0,segdat1,segdat2,segdat3,segdat4,segdat5,segdat6,segdat7,sl0,sl1,sl2,sl3,clr_key,clk 信号加入 SNF 文件中进行仿真,看是否符合设计要求,若不符合,则修改程序的相关部分。

(6)器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆,并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上,打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令,这时将弹出编程命令对话框,选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在,就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试,设计程序的功能完全正确。

三、频率计

要求：设计一个 4 频率计，可测量(**1～9999**)**Hz** 的频率信号，并将被测量信号的频率通过数码 **LED** 显示出来。

1. 设计原理

测频率的方法是首先得到一个标准的基准时钟，在单位时间内(常取 1s)对被测信号约脉冲进行计数，所计脉冲的个数即为该信号的频率。本频率计由秒基准时钟发生模块、控制计数模块和输出显示模块三大部分组成。

秒脉冲发生电路主要完成对系统时钟 11.0592MHz 进行 11059260 次分频，得到标准的 1 Hz 的秒脉冲。

在控制计数模块中，设置了一个频率计数寄存器 fosc[15:0]和一个频率更新寄存器 fosc_flash[15:0]，用于存放被测信号的频率值和更新值。

显示模块主要完成把测量的数据输出到数码 LED 上显示出来，与前面介绍的数码 LED 显示原理相同。

2. 用 Verilog HDL 描述

```
module frequency (clk_x,clk,segdat,sl);
input clk_x,clk;                          //clk_x 为被测量信号
output [7:0] segdat;                      //段码输出
output [3:0] sl;                          //位码输出
reg [23:0] count;                         //分频计数器
reg [15:0] fosc;                          //计数频率寄存器
reg [15:0] fosc_flash;                    //计数频率更新寄存器
reg [7:0] segdat_reg;                     //段码寄存器
reg [3:0] sl_reg;                         //位码寄存器
reg[3:0] disp_dat;                        //显示数据寄存器
reg second;                               //秒信号
reg flag_flash;                           //计数更新标志
always @ (posedge clk)
begin
    count=count+1;
    if(count==24'd11059260)               //产生 1Hz 信号
        begin
            second=~second;
            count=24'd0;
        end
    end
always @ (count[11:10])
    begin
```

```
            case(count[11:10])
            2'b00:disp_dat=fosc[3:0];                    //输出个位数值
            2'b01:disp_dat=fosc[7:4];                    //输出十位数值
            2'b10:disp_dat=fosc[11:8];                   //输出百位数值
            2'b11:disp_dat=fosc[15:12];                  //输出千位数值
            endcase
        end
    always  @ (disp_dat)
        begin
            case(disp_dat)
            4'h0: segdat_reg = 8'hc0;                    //显示 0
            4'h1: segdat_reg = 8'hf9;                    //显示 1
            4'h2: segdat_reg = 8'ha4;                    //显示 2
            4'h3: segdat_reg = 8'hb0;                    //显示 3
            4'h4: segdat_reg = 8'h99;                    //显示 4
            4'h5: segdat_reg = 8'h92;                    //显示 5
            4'h6: segdat_reg = 8'h82;                    //显示 6
            4'h7: segdat_reg = 8'hf8;                    //显示 7
            4'h8: segdat_reg = 8'h80;                    //显示 8
            4'h9: segdat_reg = 8'h90;                    //显示 9
            endcase
        end
    always  @ (count[11:10])
        begin
        case(count[11:10])
        2'b00:sl_reg=4'b1110;                            //扫描个位
        2'b01:sl_reg=4'b1101;                            //扫描十位
        2'b10:sl_reg=4'b1011;                            //扫描百位
        2'b11:sl_reg=4'b0111;                            //扫描千位
        endcase
        end
    always  @ (posedge clk_x)
      begin
        if(second)      //second 为 1 开始计数,为 0 则停止计数,计数周期为 1s
          begin
          flag_flash=1;                                  //计数更新标志置 1
          fosc_flash[3:0]=fosc_flash[3:0]+1;             //个位加 1
          if(fosc_flash[3:0]>4'h9)                       //若个位大于 9
            begin
```

```
            fosc_flash[3:0]=4'h0;                          //个位清 0
            fosc_flash[7:4]=fosc_flash[7:4]+1;             //十位加 1
            if(fosc_flash[7:4]>4'h9)                       //若十位大于 9
              begin
              fosc_flash[7:4]=4'h0;                        //十位清 0
              fosc_flash[11:8]=fosc_flash[11:8]+1;         //百位加 1
              if(fosc_flash[11:8]>4'h9)                    //若百位大于 9
                begin
                fosc_flash[11:8]=4'h0;                     //百位清 0
                fosc_flash[15:12]=fosc_flash[15:12]+1;     //千位加 1
                if(fosc_flash[15:12]>4'h9)                 //若千位大于 9
                fosc_flash[15:12]=4'h0;                    //千位清 0
                end
              end
            end
        end
    else if(flag_flash)                                    //若计数更新标志为 1
        begin
        flag_flash=0;                                      //计数更新标志清 0
        fosc[15:0]=fosc_flash[15:0];                       //输出计数更新寄存器内容
        fosc_flash=16'h0;                                  //计数更新寄存器清 0
        end
    end
assign segdat=segdat_reg;                                  //输出段码数据
assign sl=sl_reg;                                          //输出位码数据
endmodule
```

3. 实验验证

以上频率计程序可以在 DP-MCU/Altera 实验仪上进行验证，方法如下：

(1)新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 frequency。

(2)设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 frequency. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

(3)器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的

show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 segdat0，segdat1，segdat2，segdat3，segdat4，segdat5，segdat6，segdat7，sl0，sl1，sl2，sl3，clk，clk_x 进行引脚锁定，锁定情况如表 9-14 所示。

表 9-14　频率计引脚锁定情况

Node name（脚名）	Pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	Pin(脚号)	Pin type(输入输出类型)
segdat0	40	**output**	segdat7	50	**output**
segdat1	41	**output**	sl0	55	**output**
segdat2	44	**output**	sl1	54	**output**
segdat3	45	**output**	sl2	52	**output**
segdat4	46	**output**	sl3	51	**output**
segdat5	48	**output**	clk	83	**input**
segdat6	49	**output**	clk_x	2	**input**

(4)器件的编译

执行菜单栏中的 MAX+plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

(5)器件的仿真

执行菜单栏中的 MAX+plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“＝＞”图标，将 lsegdat0，segdat1，segdat2，segdat3，segdat4，segdat5，segdat6，segdat7，sl0，sl1，sl2，sl3，clk，clk_x 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

(6)器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX+plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能符合要求。

四、交通灯

要求：设计一个交通灯控制电路，要求绿灯亮 **30s**，接着黄灯亮 **5s**，随后，红灯亮 **20s**。

1. 设计原理

交通灯控制的基本设计原理是：产生一个标准的基准时钟，然后对绿、黄和红灯亮的时间 RGY_time 进行减 1 计数。当减 1 计数到 0 时，应进行相应灯的切换，切换时，可以设置一个切换标志寄存器 RGY_time，来存放“绿→黄、黄→红，红→绿”3 种不同的状态。

由于 DP-MCU/Altera 实验仪只有一种颜色的灯，因此，在程序设计中，可用 led0 表

示黄灯，用 led1 表示红灯，用 led2 表示绿灯。

红黄绿灯的显示时间为 2 位数，在程序设计时可采用数码 LED 的低 2 位，将高 2 位设置为 0 或熄灭。

2. 用 Verilog HDL 描述

```
module traffic(led,clk,segdat,sl);                    //模块及其结构
output [7:0] segdat;                                  //定义 LED 段码数据线
output [3:0] sl;                                      //定义 LED 位码线
output [7:0] led;                                     //定义交通灯输出口
input clk;                                            //定义时钟输入
reg [23:0] count;                                     //分频系数寄存器
reg [7:0] seg_reg;                                    //段码寄存器
reg [3:0] sl_reg;                                     //位码寄存器
reg [3:0] disp_dat;                                   //若显示数据寄存器
reg [1:0] RGY_status;                                 //定义显示红绿黄灯状态寄存器
reg sec;                                              //秒信号寄存器
reg [7:0] RGY_time;                                   //定义红绿黄灯显示时间寄存器
reg [7:0] led_reg;                                    //定义交通灯输出寄存器
initial                                               //初始化
    begin
      RGY_status=3'h0;
      RGY_time=8'h30;
    end
always @(posedge clk)
  begin
  count=count+1;
  if (count==24'd5529630)                             //2Hz 信号
      begin
        sec=~sec;                                     //产生 1Hz 信号
        count=0;                                      //清零计数寄存器
      end
  end
always @(posedge sec)                                 //计时模块
    begin
    if (RGY_status == 2'h0)                           //绿灯状态
        begin
        if (RGY_time == 8'h0)                         //绿灯该灭了吗?
          begin
            RGY_status=2'h1;                          //是,则切换到黄灯
            RGY_time=8'h5;                            //赋值为 5
            led_reg=8'b11111110;                      //L0(L0 表示黄灯)亮
          end
```

```
        else
            begin
                RGY_time=RGY_time-1;          //否则,绿灯亮的时间减 1
                if (RGY_time[3:0]>9)          //将十六进制转为十进制
                RGY_time[3:0]=9;
            end
    end
  else if (RGY_status == 2'h1)                //黄灯状态(从绿灯变为黄灯)
    begin
    if (RGY_time == 8'h0)                     //黄灯该灭了吗
        begin
            RGY_status=2'h2;                  //是,点亮红灯
            RGY_time=8'h20;                   //赋计时初值 20
            led_reg=8'b11111101;              //点亮 L1(L2 表示红灯)
        end
      else
            begin
                RGY_time=RGY_time-1;          //否则,黄灯的计时时间减 1
        end
      end
    else if (RGY_status == 2'h2)              //红灯状态
      begin
        if (RGY_time == 8'h0)                 //红灯该灭了吗
            begin
                RGY_status=2'h0;              //是,点亮绿亮
                RGY_time=8'h30;               //赋初值为 30
                led_reg=8'b11111011;          //点亮 L2(L2 表示绿灯)
        end
      else
        begin
                RGY_time=RGY_time-1;          //否则,红灯计数值减 1
                if (RGY_time[3:0]>9)          //将十六进制转为十进制
                RGY_time[3:0]=9;
          end
        end
    end
always @(count[12:11])
begin
    case (count[12:11])
        2'h0: disp_dat = RGY_time[3:0];       //显示个位数据
        2'h1: disp_dat = RGY_time[7:4];       //显示十位数据
        2'h2: disp_dat = 4'h0;                //百位数据为 0
```

```
        2'h3: disp_dat = 4'h0;                    //千位数据为 0
    endcase

    case (count[12:11])
        2'h0: sl_reg = 4'b1110;                   //扫描个位
        2'h1: sl_reg = 4'b1101;                   //扫描十位
        2'h2: sl_reg = 4'b1011;                   //扫描百位
        2'h3: sl_reg = 4'b0111;                   //扫描千位
    endcase
end
always @(disp_dat)
begin
case (disp_dat)
    4'h0: seg_reg = 8'hc0;                        //显示 0
    4'h1: seg_reg = 8'hf9;                        //显示 1
    4'h2: seg_reg = 8'ha4;                        //显示 2
    4'h3: seg_reg = 8'hb0;                        //显示 3
    4'h4: seg_reg = 8'h99;                        //显示 4
    4'h5: seg_reg = 8'h92;                        //显示 5
    4'h6: seg_reg = 8'h82;                        //显示 6
    4'h7: seg_reg = 8'hf8;                        //显示 7
    4'h8: seg_reg = 8'h80;                        //显示 8
    4'h9: seg_reg = 8'h90;                        //显示 9
    endcase
end
assign led=led_reg;                               //输出 LED 状态
assign segdat=seg_reg;                            //输出段码
assign sl=sl_reg;                                 //输出位码
endmodule
```

3. 实验验证

以上交通灯程序可以在 DP-MCU/Altera 实验仪上进行验证，方法如下。

(1)新建项目

进入 MAX+plusII 集成开发环境。单击菜单栏的 File→Project→Name，在出现的新建项目对话框中键入设计项目名，这里起名为 traffic。

(2)设计输入

单击 File→new，新建文件时选择 Text Editor File 选项，点击 OK，出现文本编辑窗口。输入以上程序，将其保存为 traffic. v 文件。单击菜单 File→Project→Set Project To Current File，把文件设为当前工程，至此，Verilog 输入完成。

(3)器件选择与引脚锁定

执行菜单栏中的 Assign→Device 命令，在出现的器件选择对话框中；Device Family 栏中选择 MAX7000S；然后在 Device 栏内选择 EPM7128SLC84-15 器件；单击 OK 按钮

加以确认。这样 Altera 公司的 EPM7128SLC84-15 已被项目所使用。注意把对话框中的 show only fastest speed grades 前的钩去掉，否则看不到 EPM7128SLC84-15。

执行菜单栏中的 Assign→Pin/Location/Chip 命令，此时会弹出引脚锁定对话框。输入 segdat0，segdat1，segdat2，segdat3，segdat4，segdat5，segdat6，segdat7，sl0，sl1，sl2，sl3，clk，led0，led1，led2，led3，led4，led5，led6，led7 进行引脚锁定，锁定情况如表 9-15 所示。

表 9-15　交通灯引脚锁定情况

Node name（脚名）	Pin(脚号)	Pin type(输入输出类型)	Node name（脚名）	Pin(脚号)	Pin type(输入输出类型)
led0	30	**output**	segdat3	45	**output**
led1	31	**output**	segdat4	46	**output**
led2	33	**output**	segdat5	48	**output**
led3	34	**output**	segdat6	49	**output**
led4	35	**output**	segdat7	50	**output**
led5	36	**output**	sl0	55	**output**
led6	37	**output**	sl1	54	**output**
led7	39	**output**	sl2	52	**output**
segdat0	40	**output**	sl3	51	**output**
segdat1	41	**output**	clk	83	**input**
segdat2	44	**output**			

(4)器件的编译

执行菜单栏中的 MAX＋plusII→Compiler 命令，这时将弹出编译命令对话框，单击 Start 按钮，即可编译相关的各项操作。

(5)器件的仿真

执行菜单栏中的 MAX＋plus II→Waveform Editor 命令，弹出波形仿真窗口。在波形仿真窗口的空白处右击鼠标，在弹出的快捷菜单中选择 EnterNodes form SNF... 命令，这时将出现如图仿真信号选择对话框。点击 List，将出现端口列表，你默认是选择全部，你也可以通过左键和 Ctrl 组合来选择你想要的信号。点击“＝＞”图标，将lsegdat0～segdat7，sl0～sl3，clk，led0～led7 信号加入 SNF 文件中进行仿真，看是否符合设计要求，若不符合，则修改程序的相关部分。

(6)器件编程下载

取出 DP-MCU/Altera 实验仪随机附送的 JTAG 电缆，并将 JTAG 下载电缆的两端分别接到 PC 机和 DP-MCU/Altera 实验仪的 JTAG 口 J4 上，打开工作电源。执行菜单栏中的 MAX＋plusII→Programmer 命令，这时将弹出编程命令对话框，选择对话框中的 Program 命令就可开始对器件进行编程。编程完毕后出现下载完成信息。现在，就可以通过实验板上的硬件资源来验证程序是否符合设计要求。通过测试，设计程序的功能符合要求。

参 考 文 献

[1] 周立功，夏宇闻，等. 单片机与 CPLD 应用技术. 北京：北京航空航天大学出版社，2003.

[2] 王金明，等. Verilog HDL 程序设计教程. 北京：人民邮电出版社，2004.

[3] 常晓明. Verilog-HDL 实践与应用系统设计. 北京：北京航空航天大学出版社，2003.

[4] 黄正谨. CPLD 系统设计技术入门与应用. 北京：电子工业出版社，2002.

[5] 夏宇闻. Verilog HDL 数字系统设计教程. 北京：北京航空航天大学出版社，2004.