Xilinx FPGA
开发实用教程

田耘 徐文波 编著

清华大学出版社

Xilinx公司是最早也是最大的FPGA生产商，其芯片设计技术、开发软件和相关解决方案在业界属于顶级水平，拥有广泛的客户群。本书主要讲述了Xilinx FPGA的开发知识，包括FPGA基础知识、VerilogHDL语言基础、基于Xilinx芯片的HDL语言高级进阶、ISE开发环境使用指南、FPGA配置电路及软件操作、在线逻辑分析仪ChipScope的使用、基于FPGA的数字信号处理技术、基于System Generator的DSP系统开发技术、基于FPGA的可编程嵌入式开发技术、基于FPGA的高速数据连接技术以及时序分析原理和时序分析器的使用等11章内容，涵盖了FPGA开发的主要方面。期望本书能够提高读者的工程开发能力。

建议上架类别：电子技术/可编程逻辑

XILINX®

EEFOCUS
Click for Answer!

ISBN 978-7-302-18425-6

定价：59.00元

Xilinx FPGA
开发实用教程

田耘 徐文波 编著

清华大学出版社
北京

内 容 简 介

本书系统讲述了 Xilinx FPGA 的开发知识，包括 FPGA 开发简介、Verilog HDL 语言基础、基于 Xilinx 芯片的 HDL 语言高级进阶、ISE 开发环境使用指南、FPGA 配置电路及软件操作、在线逻辑分析仪 ChipScope 的使用、基于 FPGA 的数字信号处理技术、基于 System Generator 的 DSP 系统开发技术、基于 FPGA 的可编程嵌入式开发技术、基于 FPGA 的高速数据连接技术和时序分析原理以及时序分析器的使用 11 章内容，各章均以实例为基础，涵盖了 FPGA 开发的主要方面。

本书适合从事 Xilinx 系列 FPGA 设计和开发的工程师，以及相关专业的研究生和高年级本科生使用。

图书在版编目(CIP)数据

Xilinx FPGA 开发实用教程/田耘，徐文波编著. —北京：清华大学出版社，2008.11
ISBN 978-7-302-18425-6

Ⅰ. X…　Ⅱ. ①田… ②徐…　Ⅲ. 可编程逻辑器件—系统开发—教材　Ⅳ. TP332.1

中国版本图书馆 CIP 数据核字(2008)第 129399 号

责任编辑：王一玲
责任校对：白　蕾
责任印制：何　芊
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京密云胶印厂
装 订 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：185×260　**印　张**：39.25　**字　数**：946 千字
版　　次：2008 年 11 月第 1 版　　**印　　次**：2008 年 11 月第1 次印刷
印　　数：1～3000
定　　价：59.00 元

序

PREFACE

赛灵思(Xilinx)公司作为可编程器件(PLD)的领导厂商，占有超过50%的市场份额，为客户提供可编程逻辑芯片(CPLD、FPGA 和 PROM)、软件设计工具、不同等级的知识产权核(IP Core)以及系统级的完整解决方案。

随着 FPGA 工艺和设计水平的不断提高，其在数字系统中所扮演的角色也从逻辑胶合者提升到处理核心。目前，赛灵思公司的 FPGA 涵盖了逻辑应用、数字信号处理以及嵌入式三大应用领域，例如 Spartan-3A/AN/E 系列 FPGA 采用 90nm 工艺，广泛应用在中低规模系统中，包括机器视觉、机顶盒、DCD 播放器以及广泛的多媒体处理等；Virtex-4/5 系列 FPGA 分别采用 90nm、65nm 工艺，主要面向高端应用，如高速互联网络、无线通信、宽带接入以及汽车工业。此外，赛灵思公司的 Virtex-5 系列 FPGA 是目前业界主要的 65nm 工艺可编程器件的提供商，占据了超过 90%的市场份额。

基于 Xilinx 公司的领先技术，更多的工程师和研究人员加入到赛灵思 FPGA 的开发队伍中来。在过去一年中，Xilinx 公司通过大学和开源社区 Openhard，举办了 Xilinx 杯开源硬件创新大赛以及多个网络研讨会，帮助广大技术人员、在校的研究生和高年级本科生尽快掌握 Xilinx FPGA 的开发流程，但切入点都比较零散，对于大多数开发人员来讲缺少一本合适的系统级书籍。《Xilinx FPGA 开发实用教程》一书弥补了上述不足。

整体而言，本书具有以下 3 个特色：首先，从逻辑设计、数字信号处理、嵌入式系统设计和高速连接功能等 4 个方面系统地介绍了赛灵思 FPGA 开发应用，条理清晰、思路明确，符合 FPGA 目前和未来的发展趋势；其次，较为详细地介绍了 Verilog HDL 语言和 Xilinx FPGA 的开发技巧，融入了作者的工程开发经验，对于初学者和工程开发人员来讲都具有较强的可读性；第三，全面介绍了赛灵思公司的 ISE、System Generator 以及 EDK 开发软件，内容完整性高。

所以本书是一本较为理想的工程工具书和大学的教辅书籍，我郑重地将其推荐给大家，希望通过本书的出版，使更多的读者掌握赛灵思 FPGA 的开发技能，更好地促进 FPGA 开发技术的普及和推广。

赛灵思公司(Xilinx, Inc.)中国区大学计划经理

谢凯年博士

2008 年 9 月

前言

FOREWORD

2007 年 10 月份，作者有幸聆听了 Xilinx 公司全球 CTO Ivo Bolsens 先生在清华大学题为“FPGA：The future platform for transforming，transporting and computing”的演讲，感触颇深。Ivo 先生指出了 FPGA 的三大应用领域：数字处理中的信号变换、高速交换中的数据收发以及求解中的复杂计算。作者本人虽然已有多年的 FPGA 开发经验，但还是第一次听到如此精辟的总结，随即想到深入了解并推广这种实用且精辟的 FPGA 开发理念。考察了许久，我们发现市场上没有此类相关书籍，且已有书籍比较偏重于单一软件的操作或 HDL 语言的讲解，因此就萌生了编写一本书，从系统开发的角度，以软、硬件结合的方式来阐述先进的 FPGA 开发理念。于是经过半年的思索、查阅资料、和相关专家大量讨论以及反复修改，便有了这本书的诞生，以期起到抛砖引玉的作用。

Xilinx 公司是最早也是最大的 FPGA 生产商，其芯片设计技术、开发软件和相关解决方案在业界属于顶级水平，拥有广泛的客户群。本书主要讲述了 Xilinx FPGA 的开发知识，包括 FPGA 开发简介、Verilog HDL 语言基础、基于 Xilinx 芯片的 HDL 语言高级进阶、ISE 开发环境使用指南、FPGA 配置电路及软件操作、在线逻辑分析仪 ChipScope 的使用、基于 FPGA 的数字信号处理技术、基于 System Generator 的 DSP 系统开发技术、基于 FPGA 的可编程嵌入式开发技术、基于 FPGA 的高速数据连接技术和时序分析原理以及时序分析器的使用共 11 章内容，各章均以实例为基础，涵盖了 FPGA 开发的主要方面。由于篇幅所限，我们没有在本书中给出一个完整的工程实例。为了弥补这一缺陷，我们 Xilinx FPGA 开源社区 Openhard 网站中附带了本书所有的实例，期望本书能够帮助提高读者的工程开发能力。

全书各章由田耘、徐文波完成，孙霏菲参与了第 7 章的编写工作。此外，在成文过程中，我们参考了较多的书籍、论文和网络文献，向其作者表示深深的谢意。Xilinx 公司中国区大学计划经理谢凯年博士在百忙之中为本书作序，并提供了硬件实验环境；Xilinx 公司亚太区公共关系经理张俊伟女士一直关心、鼓励作者，并最终促成本书成稿。与非网科技的贺潇荃先生、陶丹博士等在成书过程中给予了我们诸多良好的建议和帮助；清华大学出版社的王一玲编辑为本书的修改付出了许多劳动，并给出许多中肯的修改意见，感谢他们为本书所做的贡献。

本书适合从事 Xilinx 系列 FPGA 设计和开发的工程师，以及相关专业的研究生和高年级本科生使用。毫无疑问，市场上已经有很多关于 FPGA 设计的书籍，我们也不认为本书是其中最重要的一本，但我们意识到，FPGA 开发一定要结合芯片特点以及提供商的诸多建议和协议，只有这样才能真正掌握其开发之道。

书中的全部内容都是实际项目硬件和 Xilinx 公司各类文档、书籍的结合体，全部信息

几乎都可以从 Xilinx 网站以及 Google 上找到渊源，不过我们仍然向您推荐本书，因为网络的信息是分散的、杂乱的，且正确性不是 100%的，本书各章内容的安排是从大量的实践中总结出来的，循序渐进，条理清楚，且都经过作者验证。我们的目的就是从 Ivo Bolsens 先生的观点出发，结合项目开发，将网络上尽可能多的相关信息以相对较高的质量组合起来。

FPGA 技术博大精深且发展迅猛，不可能通过一本书进行全方位的详细介绍，更多的还需要读者自己动手实践。由于作者水平有限，加上时间比较仓促，书中不妥之处，敬请指正。在本书出版后，作者将继续在 Openhard 社区中维护书籍内容，进行修正和补充，详细网址为：http://www.openhw.org/html/08-05/415531070314nup7.html.

作　者

2008 年 5 月

注：限于篇幅，本书部分程序的 RTL 级综合结果示意图不能全幅度显示，因而不清楚。读者可在本书配套网站上查看。

目录

CONTENTS

第1章

FPGA开发简介

CHAPTER 1

FPGA(Field Programmable Gate Array)即现场可编程门阵列,属于可编程逻辑器件的一种,在20世纪90年代获得突飞猛进的发展。经过近20年的发展,到目前它已成为实现数字系统的主流平台之一。本章主要介绍FPGA的起源、发展历史、芯片结构、工作原理、开发流程以及Xilinx公司的主要可编程芯片,为读者提供FPGA系统设计的基础知识。

1.1 可编程逻辑器件基础

1.1.1 可编程逻辑器件概述

可编程逻辑器件(Programmable Logic Device,PLD)起源于20世纪70年代,是在专用集成电路(ASIC)的基础上发展起来的一种新型逻辑器件,是当今数字系统设计的主要硬件平台,其主要特点就是完全由用户通过软件进行配置和编程,从而完成某种特定的功能,且可以反复擦写。在修改和升级PLD时,不需额外地改变PCB电路板,只是在计算机上修改和更新程序,使硬件设计工作成为软件开发工作,缩短了系统设计的周期,提高了实现的灵活性并降低了成本,因此获得了广大硬件工程师的青睐,形成了巨大的PLD产业规模。

目前常见的PLD产品有编程只读存储器(Programmable Read Only Memory,PROM)、现场可编程逻辑阵列(Field Programmable Logic Array,FPLA)、可编程阵列逻辑(Programmable Array Logic,PAL)、通用阵列逻辑(Generic Array Logic,GAL)、可擦除的可编程逻辑阵列(Erasable Programmable Logic Array,EPLA)、复杂可编程逻辑器件(Complex Programmable Logic Device,CPLD)和现场可编程门阵列等类型。PLD器件从规模上又可以细分为简单PLD(SPLD)、复杂PLD(CPLD)以及FPGA。它们内部结构的实现方法各不相同。

可编程逻辑器件按照颗粒度可以分为3类:①小颗粒度(如"门海(sea of gates)"架构);②中等颗粒度(如FPGA);③大颗粒度(如CPLD)。按照编程工艺可以分为4类:①熔丝(Fuse)和反熔丝(Antifuse)编程器件;②可擦除的可编程只读存储器(UEPROM)编程器件;③电信号可擦除的可编程只读存储器(EEPROM)编程器件(如CPLD);④SRAM编程器件(如FPGA)。在工艺分类中,前3类为非易失性器件,编程后,配置数据保留在器件上;第4类为易失性器件,掉电后,配置数据会丢失,因此在每次上电后需要重新进行数据配置。

1.1.2 可编程逻辑器件的发展历史

可编程逻辑器件的发展可以划分为4个阶段，即从20世纪70年代初到70年代中为第1阶段，20世纪70年代中到80年代中为第2阶段，20世纪80年代中到90年代末为第3阶段，20世纪90年代末到目前为第4阶段。

第1阶段的可编程器件只有简单的可编程只读存储器(PROM)、紫外线可擦除只读存储器(EPROM)和电可擦只读存储器(EEPROM)3种。由于结构的限制，它们只能完成简单的数字逻辑功能。

第2阶段出现了结构上稍微复杂的可编程阵列逻辑(PAL)和通用阵列逻辑(GAL)器件，正式被称为PLD，能够完成各种逻辑运算功能。典型的PLD由“与”、“非”阵列组成，用“与或”表达式来实现任意组合逻辑，所以PLD能以乘积和形式完成大量的逻辑组合。

在第3阶段，Xilinx和Altera公司分别推出了与标准门阵列类似的FPGA以及类似于PAL结构的扩展性CPLD。它们提高了逻辑运算的速度，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，兼容了PLD和通用门阵列的优点，能够实现超大规模的电路，编程方式也很灵活，成为产品原型设计和中小规模(一般小于10 000)产品生产的首选。在这一阶段，CPLD、FPGA器件在制造工艺和产品性能方面都获得长足的发展，达到了0.18μm工艺和系统门数百万门的规模。

第4阶段出现了SOPC(System On Programmable Chip，可编程的片上系统)和SOC(System On Chip，片上系统)技术。它们是PLD和ASIC技术融合的结果，涵盖了实时化数字信号处理技术、高速数据收发器、复杂计算以及嵌入式系统设计技术的全部内容。Xilinx和Altera公司也推出了相应的SOC FPGA产品，制造工艺已达到65μm，系统门数也超过百万门。并且，这一阶段的逻辑器件内嵌了硬核高速乘法器、吉比特差分串行接口、时钟频率高达500MHz的PowerPC微处理器、软核MicroBlaze、PicoBlaze、Nios以及Nios Ⅱ，这不仅实现了软件需求和硬件设计的完美结合，还实现了高速与灵活性的完美结合，使其已超越了ASIC器件的性能和规模，也超越了传统意义上FPGA的概念，使PLD的应用范围从单片扩展到系统级。目前，基于PLD片上可编程的概念仍在不断地向前发展。

1.1.3 PLD开发工具

基于高复杂度PLD器件的开发，在很大程度上要依靠电子设计自动化(EDA)来完成。PLD的EDA工具以计算机软件为主，将典型的单元电路封装起来形成固定模块并形成标准的硬件开发语言(如HDL语言)供设计人员使用。设计人员考虑如何将可组装的软件库和软件包搭建出满足需求的功能模块甚至完整的系统。PLD开发软件需要自动地完成逻辑编译、化简、分割、综合及优化、布局布线、仿真以及对于特定目标芯片的适配编译和编程下载等工作。典型的EDA工具中必须包含两个特殊的软件包，即综合器和适配器。综合器的功能就是将设计者在EDA平台上完成的针对某个系统项目的HDL、原理图或状态图形描述，针对给定的硬件系统组件，进行编译、优化、转换和综合。适配器一般不用于FPGA设计。

随着开发规模的级数性增长，必须减短 PLD 开发软件的编译时间，提高其编译性能，并提供丰富的知识产权(IP)核资源供设计人员调用。此外，PLD 开发界面的友好性以及操作的复杂程度也是评价其性能的重要因素。目前在 PLD 产业领域中，各个芯片提供商的 PLD 开发工具已成为影响其成败的核心成分。只有全面做到芯片技术领先、文档完整和 PLD 开发软件优秀，芯片提供商才能获得客户的认可。一个完美的 PLD 开发软件应当具备下面 5 点：

- 准确地将用户设计转换为电路模块；
- 能够高效地利用器件资源；
- 能够快速地完成编译和综合；
- 提供丰富的 IP 核资源；
- 用户界面友好，操作简单。

Xilinx 公司的 ISE、Altera 公司的 Quartus Ⅱ 和 Maxplus Ⅱ 是业界公认的优秀集成 PLD 开发软件。此外，综合软件 Synplify 和仿真软件 ModelSim 等诸多第三方 EDK 开发软件也满足上述要求。

1.2 FPGA 芯片结构

1.2.1 FPGA 工作原理与简介

如前所述，FPGA 是在 PAL、GAL、EPLD、CPLD 等可编程器件的基础上进一步发展的产物。它是作为 ASIC 领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路有限的缺点。

由于 FPGA 需要被反复烧写，它实现组合逻辑的基本结构不可能像 ASIC 那样通过固定的与非门来完成，而只能采用一种易于反复配置的结构，查找表可以很好地满足这一要求。目前，主流 FPGA 都采用了基于 SRAM 工艺的查找表结构，也有一些军品和宇航级 FPGA 采用 Flash 或者熔丝与反熔丝工艺的查找表结构。可通过烧写文件改变查找表内容的方法来实现对 FPGA 的重复配置。

根据数字电路的基本知识可以知道，对于一个 n 输入的逻辑运算，不管是与或非运算还是异或运算，最多只可能存在 $2n$ 种结果。所以，如果事先将相应的结果存放于一个存储单元中，就相当于实现了与非门电路的功能。FPGA 的原理也是如此，它通过烧写文件去配置查找表的内容，从而在相同的电路情况下实现了不同的逻辑功能。

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 HDL 语言描述了一个逻辑电路以后，PLD/FPGA 开发软件会自动计算逻辑电路的所有可能结果，并把真值表(即结果)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后输出即可。

下面给出一个 4 输入与门电路的例子来说明 LUT 实现逻辑功能的原理。

例 1-1 表 1-1 给出一个使用 LUT 实现 4 输入与门电路的真值表。

表 1-1 4 输入与门的真值表

实际逻辑电路		LUT 的实现方式	
a,b,c,d 输入	逻辑输出	RAM 地址	RAM 中存储的内容
0 0 0 0	0	0 0 0 0	0
0 0 0 1	0	0 0 0 1	0
…	…	…	…
1 1 1 1	1	1 1 1 1	1

从中可以看到，LUT 具有和逻辑电路相同的功能。实际上，LUT 具有更快的执行速度和更大的规模。

由于基于 LUT 的 FPGA 具有很高的集成度，其器件密度从数万门到数千万门不等，可以完成极其复杂的时序逻辑电路与组合逻辑电路，所以它适用于高速、高密度的高端数字逻辑电路设计领域。其组成部分主要有可编程输入/输出单元、基本可编程逻辑单元、内嵌 SRAM、丰富的布线资源、底层嵌入功能单元和内嵌专用单元等，主要设计和生产厂家有 Xilinx、Altera、Lattice、Actel、Atmel 和 QuickLogic 等公司，其中最大的是 Xilinx、Altera、Lattice 三家。

如前所述，FPGA 是由存放在片内的 RAM 来设置其工作状态的，因此工作时需要对片内 RAM 进行编程。用户可根据不同的配置模式，采用不同的编程方式。FPGA 有如下几种配置模式：

- 并行模式：并行 PROM、Flash 配置 FPGA；
- 主从模式：一片 PROM 配置多片 FPGA；
- 串行模式：串行 PROM 配置 FPGA；
- 外设模式：将 FPGA 作为微处理器的外设，由微处理器对其编程。

目前，FPGA 市场占有率最高的两大公司 Xilinx 和 Altera 生产的 FPGA 都是基于 SRAM 工艺的，需要在使用时外接一个片外存储器以保存程序。上电时，FPGA 将外部存储器中的数据读入片内 RAM，完成配置后，进入工作状态；掉电后，FPGA 恢复为白片，内部逻辑消失。这样，FPGA 不仅能反复使用，还无需专门的 FPGA 编程器，只需通用的 EPROM、PROM 编程器即可。Actel、QuickLogic 等公司还提供反熔丝技术的 FPGA。它只能下载一次，具有抗辐射、耐高低温、低功耗和速度快等优点，在军品和航空航天领域中应用较多，但这种 FPGA 不能重复擦写，开发初期比较麻烦，费用也比较昂贵。Lattice 是 ISP (In-System Programmable，在系统可编程)技术的发明者，它在小规模 PLD 应用上有一定的特色。早期的 Xilinx 产品一般不涉及军品和宇航级市场，但目前已经有 Q Pro-R 等多款产品进入该类领域。

1.2.2 FPGA 芯片结构

目前，FPGA 芯片仍是基于查找表技术的，但其概念和性能已经远远超出查找表技术的限制，并且整合了常用功能的硬核模块(如块 RAM、时钟管理和 DSP)。图 1-1 所示为 Xilinx 公司 Spartan-2 系列 FPGA 的内部结构图(注意，图 1-1 所示只是一个示意图。实际

上，每一个系列的 FPGA 都有其相应的内部结构，由于应用场合不同，所以内部结构会有局部不同），从中可以看出 FPGA 芯片主要由 6 个部分组成：可编程输入/输出单元、基本可编程逻辑单元、完整的时钟管理、嵌入块式 RAM、丰富的布线资源、内嵌的底层功能单元和内嵌专用硬件模块。

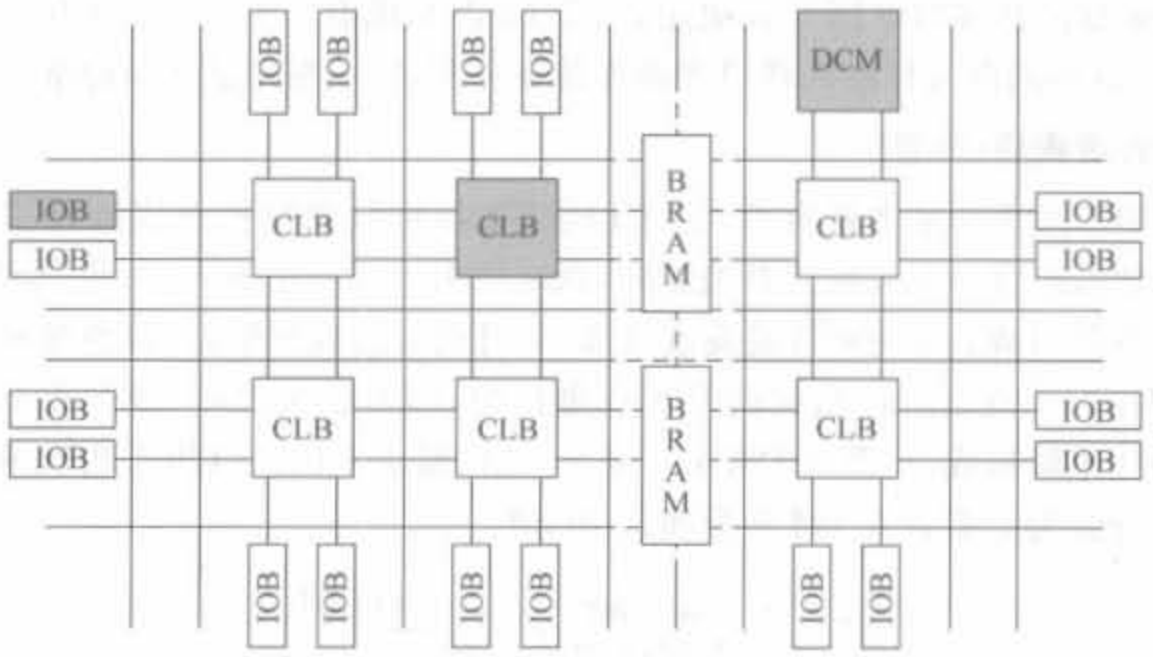


图 1-1 FPGA 芯片的内部结构

每个模块的功能简述如下。

1. 可编程输入/输出单元(IOB)

可编程输入/输出单元简称 I/O 单元，是芯片与外界电路的接口部分，用于完成不同电气特性下对输入/输出信号的驱动与匹配要求，其示意结构如图 1-2 所示。FPGA 内的 I/O 按组分类，每组都能够独立地支持不同的 I/O 标准。通过软件的灵活配置，可适配不同的电气标准与 I/O 物理特性，可以调整驱动电流的大小，可以改变上、下拉电阻。目前，I/O 口的频率越来越高，一些高端的 FPGA 通过 DDR 寄存器技术可以支持高达 2Gb/s 的数据速率。

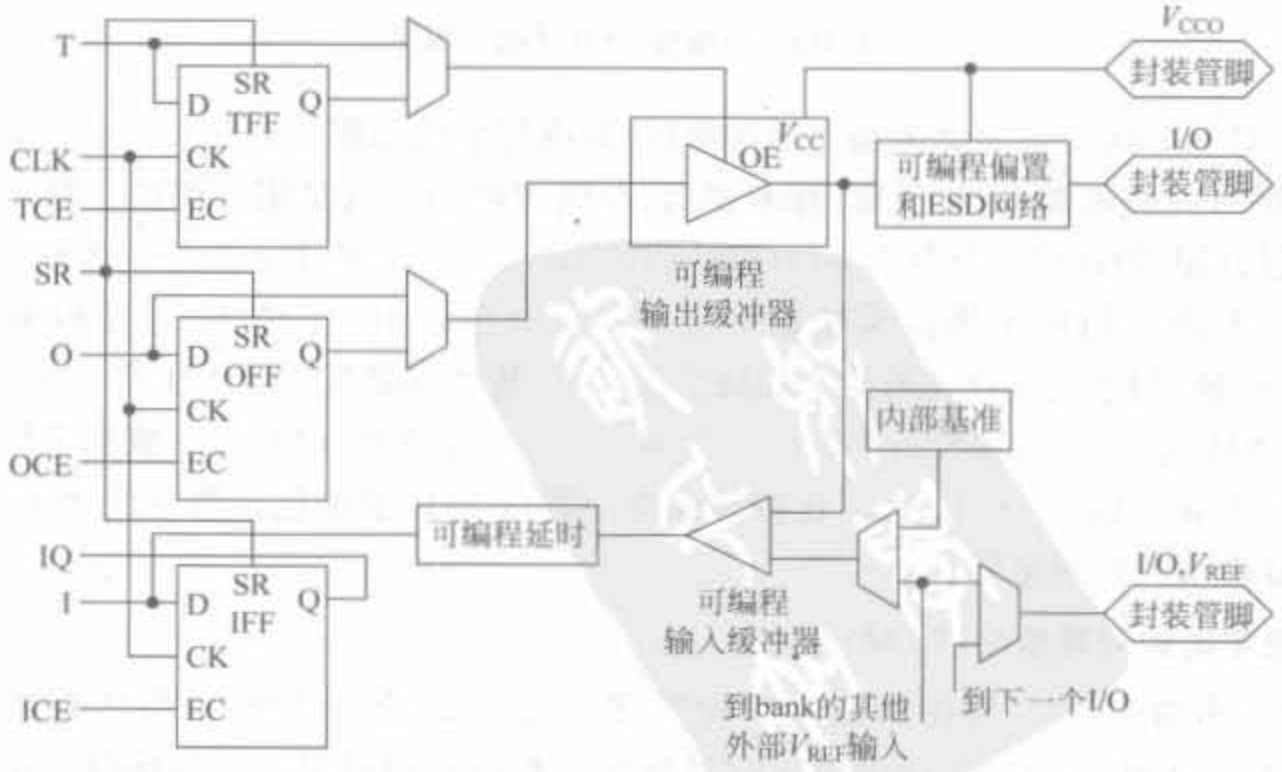


图 1-2 典型的 IOB 内部结构示意图

外部输入信号可以通过 IOB 模块的存储单元输入到 FPGA 的内部，也可以直接输入 FPGA 内部。当外部输入信号经过 IOB 模块的存储单元输入到 FPGA 内部时，其保持时间(Hold Time)的要求可以降低，通常默认为 0。

为了便于管理和适应多种电气标准，FPGA 的 IOB 被划分为若干个组(bank)，每个 bank 的接口标准由其接口电压 V_{cco} 决定，一个 bank 只能有一种 V_{cco}，但不同 bank 的 V_{cco} 可以不同。只有相同电气标准的端口才能连接在一起，V_{cco} 相同是接口标准的基本条件。

2. 可配置逻辑块(CLB)

CLB 是 FPGA 内的基本逻辑单元。CLB 的实际数量和特性会依器件的不同而不同，但是每个 CLB 都包含一个可配置开关矩阵，此矩阵由 4 或 6 个输入、一些选型电路(多路复用器等)和触发器组成。开关矩阵是高度灵活的，可以对其进行配置，以便处理组合逻辑、移位寄存器或 RAM。在 Xilinx 公司的 FPGA 器件中，CLB 由多个(一般为 4 个或 2 个)相同的 Slice 和附加逻辑构成，如图 1-3 所示。每个 CLB 模块不仅可以用于实现组合逻辑、时序逻辑，还可以配置为分布式 RAM 和分布式 ROM。

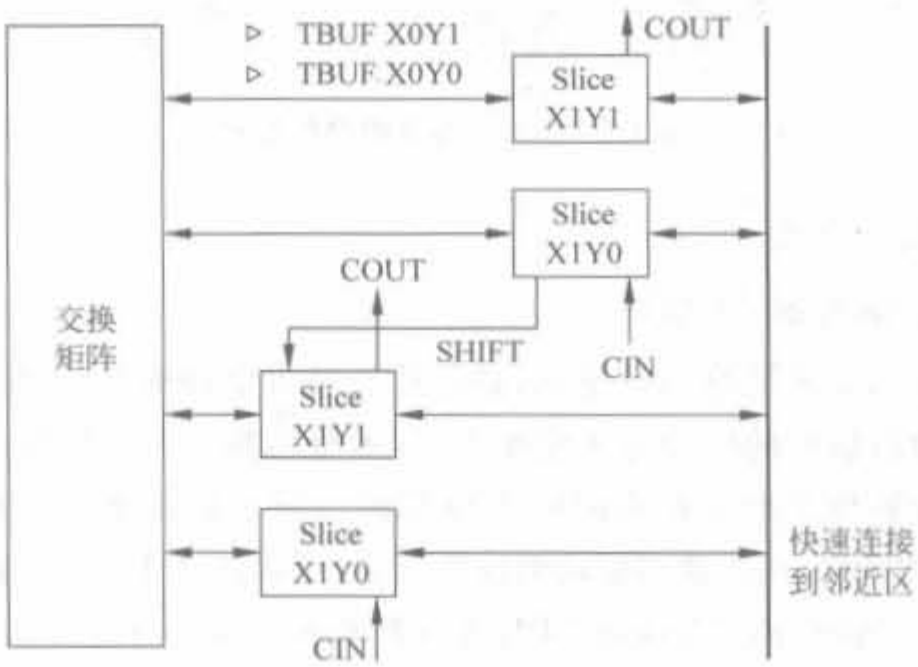


图 1-3　典型的 CLB 结构示意图

Slice 是 Xilinx 公司定义的基本逻辑单位，其内部结构如图 1-4 所示，一个 Slice 由两个 4/6 输入的查找表函数、进位逻辑、算术逻辑、存储逻辑和函数复用器组成。算术逻辑包括一个异或门(XORG)和一个专用与门(MULTAND)，一个异或门可以使一个 Slice 实现 2bit 全加操作，专用与门用于提高乘法器的效率；进位逻辑由专用进位信号和函数复用器(MUXC)组成，用于实现快速的算术加减法操作；4 输入函数发生器用于实现 4 输入 LUT、分布式 RAM 或 16 比特移位寄存器(目前，基于 65nm 工艺的 FPGA 一般都采用 6 输入查找表，可以实现 6 输入 LUT 或 64 比特移位寄存器)；进位逻辑包括两条快速进位链，用于提高 CLB 模块的处理速度。

3. 数字时钟管理模块(DCM)

业内大多数 FPGA 均提供数字时钟管理(Xilinx 的全部 FPGA 均具有这种特性)。Xilinx 推出最先进的 FPGA 提供数字时钟管理和相位环路锁定。相位环路锁定能够提供精确的时钟综合，且能够降低抖动，并实现过滤功能。

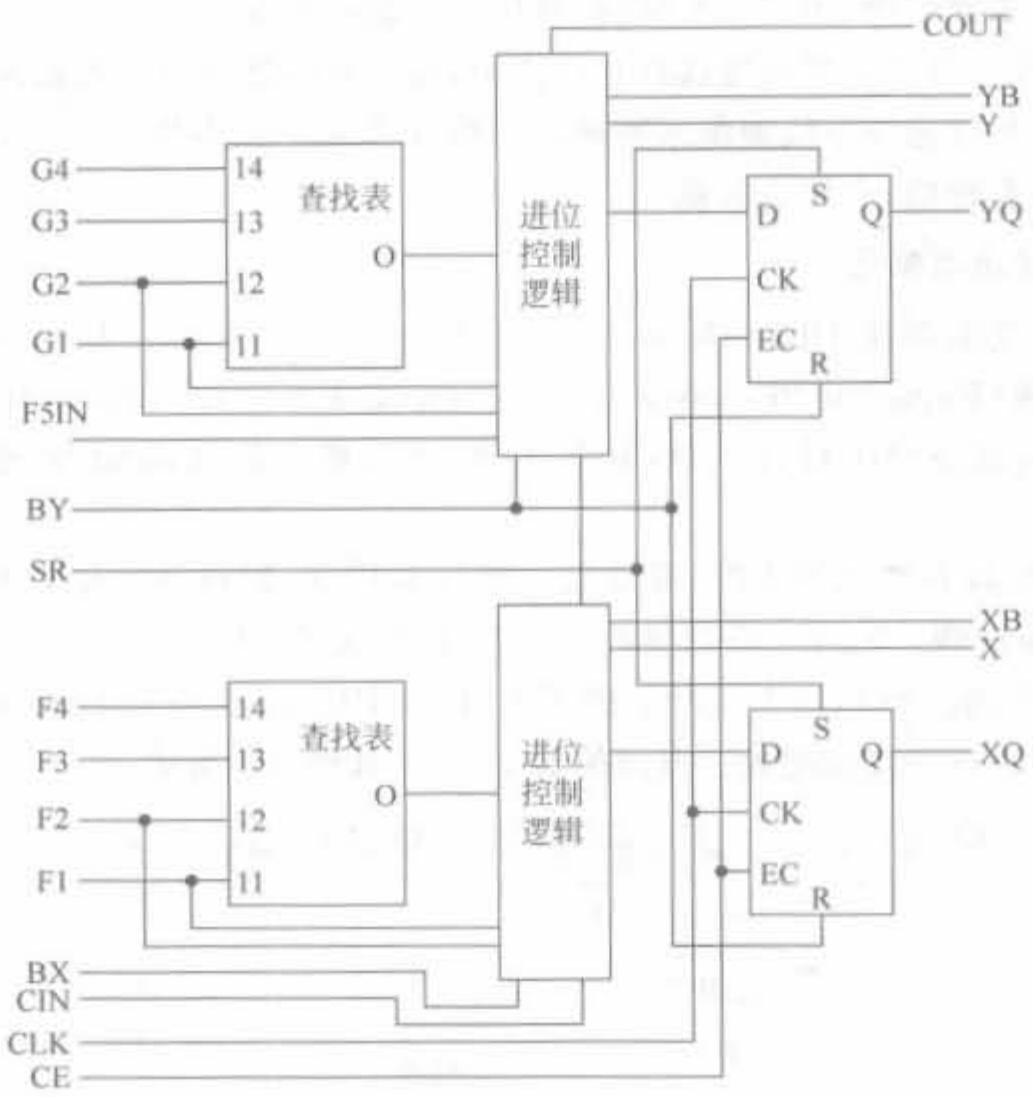


图 1-4 典型的 4 输入 Slice 结构示意图

4. 嵌入式块 RAM(BRAM)

大多数 FPGA 都具有内嵌的块 RAM,这大大拓展了 FPGA 的应用范围和灵活性。块 RAM 可被配置为单端口 RAM、双端口 RAM、内容地址存储器(CAM)以及 FIFO 等常用存储结构。RAM、FIFO 是比较普及的概念,在此就不赘述。CAM 存储器在其内部的每个存储单元中都有一个比较逻辑,写入 CAM 中的数据会和内部的每一个数据进行比较,并返回与端口数据相同的所有数据的地址,因而在路由的地址交换器中有广泛的应用。除了块 RAM,还可以将 FPGA 中的 LUT 灵活地配置成 RAM、ROM 和 FIFO 等结构。在实际应用中,芯片内部块 RAM 的数量也是选择芯片的一个重要因素。

单片块 RAM 的容量为 18Kbit,即位宽为 18bit、深度为 1024bit,可以根据需要改变其位宽和深度,但要满足两个原则:首先,修改后的容量(位宽×深度)不能大于 18Kbit;其次,位宽最大不能超过 36bit。当然,可以将多片块 RAM 级联起来形成更大的 RAM,此时只受限于芯片内块 RAM 的数量,而不再受上面两条原则约束。

5. 丰富的布线资源

布线资源连通 FPGA 内部的所有单元,而且连线的长度和工艺决定着信号在连线上的驱动能力和传输速度。FPGA 芯片内部有着丰富的布线资源,根据工艺、长度、宽度和分布位置的不同而划分为 4 个不同的类别。第一类是全局布线资源,用于芯片内部全局时钟和全局复位/置位的布线;第二类是长线资源,用于完成芯片 bank 间的高速信号和第二全局时钟信号的布线;第三类是短线资源,用于完成基本逻辑单元之间的逻辑互连和布线;第

四类是分布式的布线资源，用于专有时钟、复位等控制信号线。

在实际中，设计者不需要直接选择布线资源，布局布线器可自动地根据输入逻辑网表的拓扑结构和约束条件选择布线资源来连通各个模块单元。从本质上讲，布线资源的使用方法和设计的结果有密切、直接的关系。

6. 底层内嵌功能单元

内嵌功能模块主要指 DLL(Delay Locked Loop)、PLL(Phase Locked Loop)、DSP 和 CPU 等软处理核(Embeded Processor)。正是由于集成了丰富的内嵌功能单元，从而使得单片 FPGA 成为系统级的设计工具，具备了软、硬件联合设计的能力，逐步向 SOC 平台过渡。

DLL 和 PLL 具有类似的功能，可以完成时钟高精度、低抖动的倍频和分频，以及占空比调整和移相等功能。Xilinx 公司生产的芯片上集成了 DLL，Altera 公司的芯片集成了 PLL，Lattice 公司的新型芯片上同时集成了 PLL 和 DLL。PLL 和 DLL 可以通过 IP 核生成的工具方便地进行管理和配置。典型的 DLL 结构如图 1-5 所示。

图 1-5 典型的 DLL 模块示意图

7. 内嵌专用硬核

内嵌专用硬核是相对底层嵌入的软核而言的，指 FPGA 处理能力强大的硬核(Hard Core)，等效于 ASIC 电路。为了提高 FPGA 性能，芯片生产商在芯片内部集成了一些专用的硬核。例如，为了提高 FPGA 的乘法速度，主流的 FPGA 中都集成了专用乘法器；为了适用通信总线与接口标准，很多高端的 FPGA 内部都集成了串并收发器(SERDES)，可以达到数十吉比特/秒的收发速度。

例如，Xilinx 公司的高端产品不仅集成了 PowerPC 系列 CPU，还内嵌了 DSP Core 模块，其相应的系统级设计工具是 EDK 和 Platform Studio，并依此提出了片上系统 SOC 的概念。通过 PowerPC、MicroBlaze、PicoBlaze 等处理器平台，能够开发标准的 DSP 处理器及其相关应用，达到 SOC 的开发目的。

1.2.3 软核、硬核以及固核的概念

IP(Intellectual Property)核是具有知识产权核的集成电路芯核总称，是经过反复验证的、具有特定功能的宏模块，它与芯片制造工艺无关，可以移植到不同的半导体工艺中。到了 SOC 阶段，IP 核设计已成为 ASIC 电路设计公司和 FPGA 提供商的重要任务，也是其实力的体现。对于 FPGA 开发软件，它提供的 IP 核越丰富，用户的设计就越方便，其市场占有率就越高。目前，IP 核已经变成系统设计的基本单元，并作为独立设计成果被交换、转让

和销售。

从IP核的提供方式上，通常将其分为软核、硬核和固核这3类。从完成IP核所花费的成本来讲，硬核代价最大；从使用灵活性来讲，软核的可复用性最高。

1. 软核

软核在EDA设计领域指的是综合之前的寄存器传输级(RTL)模型。具体在FPGA设计中，指的是对电路的硬件语言描述，包括逻辑描述、网表和帮助文档等。软核只经过功能仿真，需要经过综合以及布局布线才能使用。其优点是灵活性高、可移植性强，允许用户自配置；缺点是对模块的预测性较低，在后续设计中存在发生错误的可能性，有一定的设计风险。软核是IP核应用最广泛的形式。

2. 固核

固核在EDA设计领域指的是带有平面规划信息的网表。具体在FPGA设计中，可以看作带有布局规划的软核，通常以RTL代码和对应具体工艺网表的混合形式提供。将RTL描述结合具体标准单元库进行综合优化设计，形成门级网表，再通过布局布线工具即可使用。和软核相比，固核的设计灵活性稍差，但在可靠性上有较大提高。目前，固核也是IP核的主流形式之一。

3. 硬核

硬核在EDA设计领域指经过验证的设计版图。具体在FPGA设计中，指布局和工艺固定、经过前端和后端验证的设计，设计人员不能对其修改。不能修改的原因有两个：首先是系统设计对各个模块的时序要求很严格，不允许打乱已有的物理版图；其次是保护知识产权的要求，不允许设计人员对其有任何改动。IP硬核的不许修改特点使其复用有一定的困难，因此只能用于某些特定应用，使用范围较窄。

1.3 基于FPGA的开发流程

1.3.1 FPGA设计方法概论

FPGA是可编程芯片，因此FPGA的设计方法包括硬件设计和软件设计两部分。硬件包括FPGA芯片电路、存储器、输入/输出接口电路以及其他设备，软件即是相应的HDL程序以及最新才流行的嵌入式C程序。硬件设计是基础，但其方法比较固定，本书将在第4章详细介绍。本节主要介绍软件的设计方法。

目前微电子技术已经发展到SOC阶段，即集成系统(Integrated System)阶段，相对于集成电路(IC)的设计思想有着革命性的变化。SOC是一个复杂的系统，它将一个完整产品的功能集成在一个芯片上，包括核心处理器、存储单元、硬件加速单元以及众多的外部设备接口等，它具有设计周期长、实现成本高等特点，因此其设计方法必然是自顶向下的从系统级到功能模块的软、硬件协同设计，达到软、硬件的无缝结合。

这么庞大的工作量显然超出了单个工程师的能力，因此需要按照层次化、结构化的设计方法来实施。首先由总设计师将整个软件开发任务划分为若干个可操作的模块，并对其接

口和资源进行评估，编制出相应的行为或结构模型，再将其分配给下一层的设计师。这就允许多个设计者同时设计一个硬件系统中的不同模块，并为自己所设计的模块负责；然后由上层设计师对下层模块进行功能验证。

自顶向下的设计流程从系统级设计开始，划分为若干个二级单元，再把各个二级单元划分为下一层次的基本单元，一直下去，直到能够使用基本模块或者 IP 核直接实现为止，如图 1-6 所示。流行的 FPGA 开发工具都提供了层次化管理，可以有效地梳理错综复杂的层次，使得用户能够方便地查看某一层次模块的源代码，以便修改错误。

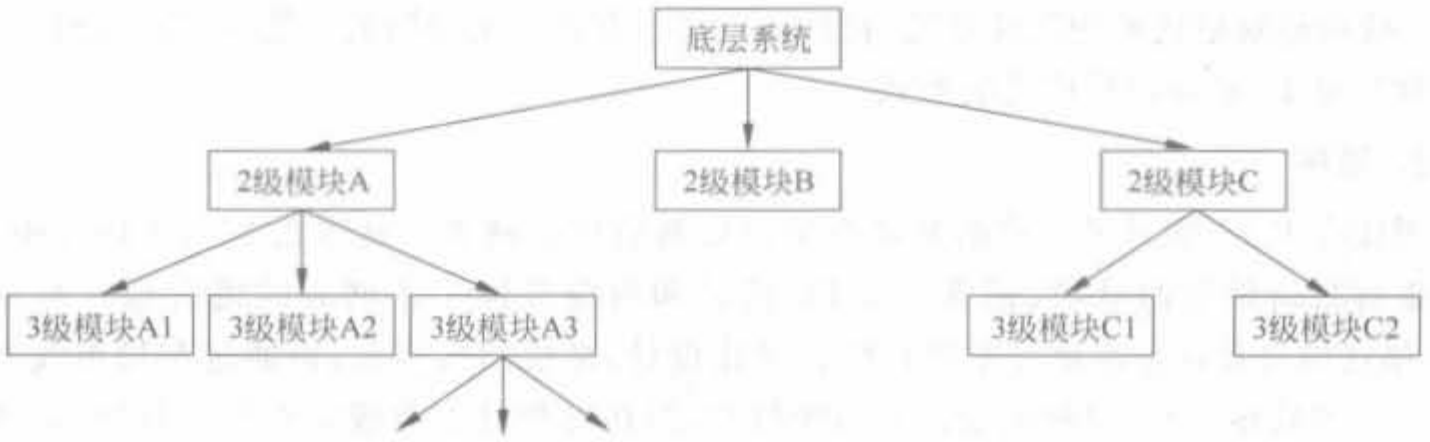


图 1-6　自顶向下的 FPGA 设计开发流程

在工程实践中，还存在软件编译时长的问题。由于大型设计包含多个复杂的功能模块，其时序收敛与仿真验证的复杂度很高，为了满足时序指标的要求，往往需要反复修改源文件，再对所修改的新版本进行重新编译，直到满足要求为止。这里面存在两个问题：首先，软件编译一次需要长达数小时甚至数周的时间，这是开发所不能容忍的；其次，重新编译和布局布线后，结果差异很大，会将已满足时序的电路破坏。因此必须提出一种能有效提高设计性能，继承已有结果，便于团队化设计的软件工具。FPGA 厂商意识到这类需求，开发出了相应的逻辑锁定和增量设计的软件工具。例如，Xilinx 公司的解决方案就是 PlanAhead。

PlanAhead 允许高层设计者为不同的模块划分相应的 FPGA 芯片区域，并允许底层设计者在所给定的区域内独立地进行设计、实现和优化，等各个模块都正确后，再进行设计整合。如果在设计整合中出现错误，单独修改即可，不会影响到其他模块。PlanAhead 将结构化设计方法、团队化合作设计方法以及重用继承设计方法三者完美地结合在一起，有效地提高了设计效率，缩短了设计周期。

不过从其描述可以看出，新型的设计方法对系统顶层设计师有很高的要求。在设计初期，他们不仅要评估每个子模块所消耗的资源，还需要给出相应的时序关系；在设计后期，需要根据底层模块的实现情况完成相应的修订。

1.3.2　典型 FPGA 开发流程

FPGA 的设计流程就是利用 EDA 开发软件和编程工具对 FPGA 芯片进行开发的过程。FPGA 的开发流程一般如图 1-7 所示，包括电路功能设计、设计输入、功能仿真、综合、综合后仿真、实现与布局布线、时序仿真与验证、板级仿真与验证以及芯片编程与调试等主要步骤。

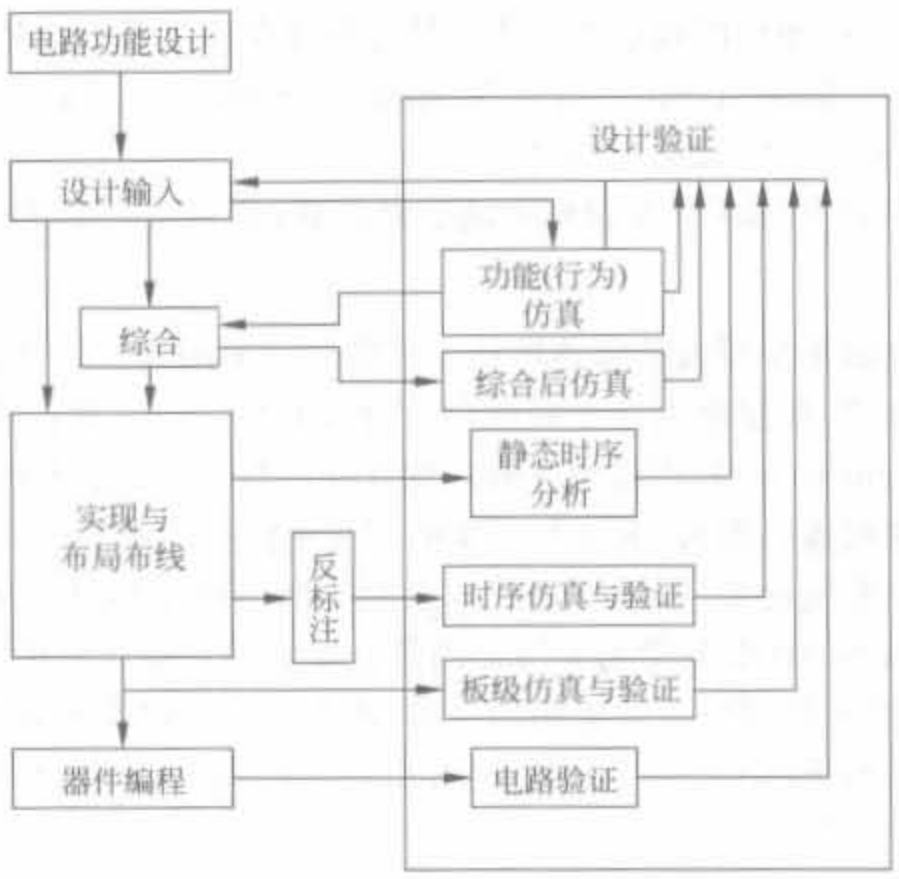


图 1-7 FPGA 开发的一般流程

1. 电路功能设计

在系统设计之前，首先要进行的是方案论证、系统设计和 FPGA 芯片选择等准备工作。系统工程师根据任务要求，如系统的指标和复杂度，对工作速度和芯片本身的资源、成本等方面进行权衡，选择合理的设计方案和合适的器件类型。一般都采用自顶向下的设计方法，把系统分成若干个基本单元，然后把每个基本单元划分为下一层次的基本单元，一直这样做下去，直到可以直接使用 EDA 元件库为止。

2. 设计输入

设计输入是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，并输入给 EDA 工具的过程。常用的方法有硬件描述语言(HDL)和原理图输入方式等。原理图输入方式是一种最直接的描述方式，在可编程芯片发展的早期应用比较广泛，它将所需的器件从元件库中调出来，画出原理图。这种方法虽然直观并易于仿真，但效率很低，且不易维护，不利于模块构造和重用，更主要的缺点是可移植性差，当芯片升级后，所有的原理图都需要作一定的改动。目前，在实际开发中应用最广的就是 HDL 语言输入法，利用文本描述设计，可以分为普通 HDL 和行为 HDL。普通 HDL 有 ABEL、CUR 等，支持逻辑方程、真值表和状态机等表达方式，主要用于简单的小型设计。而在中、大型工程中，主要使用行为 HDL，其主流语言是 Verilog HDL 和 VHDL。这两种语言都是美国电气与电子工程师协会(IEEE)的标准，其共同的突出特点有语言与芯片工艺无关，利于自顶向下设计，便于模块的划分与移植，可移植性好，具有很强的逻辑描述和仿真功能，而且输入效率很高。本书第 2 章将会给出 Verilog HDL 设计基础。

3. 功能仿真

功能仿真也称为前仿真，是在编译之前对用户所设计的电路进行逻辑功能验证。此时的仿真没有延迟信息，仅对初步的功能进行检测。仿真前，要先利用波形编辑器和 HDL 等建立波形文件和测试向量(即将所关心的输入信号组合成序列)。仿真结果将会生成报告文

件和输出信号波形，从中便可以观察各个节点信号的变化。如果发现错误，则返回设计进行修改。常用的工具有 Model Tech 公司的 ModelSim、Synopsys 公司的 VCS 和 Cadence 公司的 NC-Verilog 以及 NC-VHDL 等软件。

虽然功能仿真不是 FPGA 开发过程中的必需步骤，但却是系统设计中最关键的一步。

4. 综合

综合就是将较高级抽象层次的描述转化成较低层次的描述。综合优化根据目标与要求优化所生成的逻辑连接，使层次设计平面化，供 FPGA 布局布线软件来实现。就目前的层次来看，综合优化(Synthesis)是指将设计输入编译成由与门、或门、非门、RAM、触发器等基本逻辑单元组成的逻辑连接网表，而并非真实的门级电路。真实、具体的门级电路需要利用 FPGA 制造商的布局布线功能，根据综合后生成的标准门级结构网表来产生。为了能转换成标准的门级结构网表，HDL 程序的编写必须符合特定综合器所要求的风格。由于门级结构、RTL 级的 HDL 程序的综合是很成熟的技术，所有的综合器都可以支持到这一级别的综合。常用的综合工具有 Synplicity 公司的 Synplify/Synplify Pro 软件以及各个 FPGA 厂家自己推出的综合开发工具。

5. 综合后仿真

综合后仿真检查综合结果是否和原设计一致。在仿真时，把综合生成的标准延时文件反标注到综合仿真模型中去，可估计门延时带来的影响。但这一步骤不能估计线延时，因此和布线后的实际情况还有一定的差距，并不十分准确。目前的综合工具较为成熟，对于一般的设计可以省略这一步，但如果在布局布线后发现电路结构和设计意图不符，则需要回溯到综合后仿真来确认问题之所在。在功能仿真中介绍的软件工具一般都支持综合后仿真。

6. 实现与布局布线

实现是将综合生成的逻辑网表配置到具体的 FPGA 芯片上，布局布线是其中最重要的过程。布局将逻辑网表中的硬件原语和底层单元合理地配置到芯片内部的固有硬件结构上，并且往往需要在速度最优和面积最优之间作出选择。布线根据布局的拓扑结构，利用芯片内部的各种连线资源，合理、正确地连接各个元件。目前，FPGA 的结构非常复杂，特别是在有时序约束条件时，需要利用时序驱动的引擎进行布局布线。布线结束后，软件工具会自动生成报告，提供有关设计中各部分资源的使用情况。由于只有 FPGA 芯片生产商对芯片结构最为了解，所以布局布线必须选择芯片开发商提供的工具。

7. 时序仿真与验证

时序仿真也称为后仿真，是指将布局布线的延时信息反标注到设计网表中来检测有无时序违规(即不满足时序约束条件或器件固有的时序规则，如建立时间、保持时间等)现象。时序仿真包含的延迟信息最全，也最精确，能较好地反映芯片的实际工作情况。由于不同芯片的内部延时不一样，不同的布局布线方案也给延时带来不同的影响。因此在布局布线后，通过对系统和各个模块进行时序仿真，分析其时序关系，估计系统性能，以及检查和消除竞争冒险是非常有必要的。在功能仿真中介绍的软件工具一般都支持综合后仿真。

8. 板级仿真与验证

板级仿真主要应用于高速电路设计中，对高速系统的信号完整性、电磁干扰等特征进行分析，一般都以第三方工具进行仿真和验证。

9. 芯片编程与调试

设计的最后一步就是芯片编程与调试。芯片编程是指产生使用的数据文件(位数据流文件,Bitstream Generation),然后将编程数据下载到 FPGA 芯片中。其中,芯片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方面。逻辑分析仪(Logic Analyzer,LA)是 FPGA 设计的主要调试工具,但需要引出大量的测试管脚,且 LA 价格昂贵。目前,主流的 FPGA 芯片生产商都提供了内嵌的在线逻辑分析仪(如 Xilinx ISE 中的 ChipScope、Altera Quartus Ⅱ 中的 SignalTap Ⅱ 以及 SignalProb)来解决上述矛盾,它们只需要占用芯片少量的逻辑资源,具有很高的实用价值。

1.3.3 基于 FPGA 的 SOC 设计方法

基于 FPGA 的 SOC 设计理念将 FPGA 可编程的优点带到了 SOC 领域,其系统由嵌入式处理器内核、DSP 单元、大容量处理器、吉比特收发器、混合逻辑、IP 以及原有的设计部分组成。相应的 FPGA 规模大都在百万门以上,适合于许多领域,如电信、计算机等行业。

系统设计方法是 SOC 常用的方法,其优势在于可进行反复修改并对系统架构实现进行验证,还包括 SOC 集成硬件和软件组件之间的接口模块。不过,目前仍存在很多问题,最大的问题就是没有通用的系统描述语言和系统级综合工具。随着 FPGA 平台的融入,将 SOC 逐步地推向了实用。SOC 平台的核心部分是内嵌的处理内核,其硬件是固定的,软件则是可编程的;外围电路由 FPGA 的逻辑资源组成,大都以 IP 的形式提供,例如存储器接口、USB 接口以及以太网 MAC 层接口等,用户根据自己的需要在内核总线上添加,并能自己订制相应的接口 IP 和外围设备。

基于 FPGA 的典型 SOC 开发流程为:

1. 芯片内设计的考虑

从设计生成开始,设计人员需要从硬件/软件协同验证的思路入手,以找出只能在系统集成阶段才会被发现的软、硬件缺陷;然后选择合适的芯片以及开发工具,在综合过程中得到优化;随后进行精确的实现,以满足实际需求。由于设计规模越来越大,工作频率也到了数百兆赫兹,布局布线的延迟将变得非常重要。为了确保满足时序,需要在布局布线后进行静态时序分析,对设计进行验证。

2. 板级验证

在芯片设计完毕后,需要再进行板级验证,以便在印刷电路板(PCB)上保证与最初的设计功能一致。因此,PCB 布局以及信号完整性测试应被纳入设计流程。由于芯片内设计所做的任何改变都将反映在下游的设计流程中,各个过程之间的数据接口和管理也必须是无误的。预计 SOC 系统以及所必需的额外过程将使数据的大小呈指数增长,因此,管理各种数据集本身是极具挑战性的任务。

1.4 Xilinx 公司主流可编程逻辑器件简介

世界上第一款 FPGA 芯片由 Xilinx 公司于 1984 年推出。之后,从 XC330、XC4000 到 Spartan-3/3A /3E 系列和 Virtex-4 系列,Xilinx 公司一直保持着 FPGA 领域的全球领先地

位。目前，XC3000、XC4000、Spartan、Spartan-XL、Virtex、Virtex-E 系列已经基本被淘汰或者正逐渐退出市场。Spartan-2 和 Virtex-2 以后的 FPGA 产品为目前的主流产品。此外，Xilinx 公司的 PROM 配置器件也具有相当广泛的应用范围。

1.4.1 Xilinx FPGA 芯片介绍

Xilinx 公司目前有两大类 FPGA 产品：Spartan 类和 Virtex 类，前者主要面向低成本的中低端应用，是目前业界成本最低的一类 FPGA；后者主要面向高端应用，属于业界的顶级产品。这两个系列的差异仅限于芯片的规模和专用模块上，都采用了先进的 0.13μm、90nm，甚至 65nm 制造工艺，具有相同的卓越品质。

1. Spartan 类

Spartan 系列适用于普通的工业、商业等领域，目前主流的芯片包括 Spartan-2、Spartan-2E、Spartan-3、Spartan-3A 以及 Spartan-3E 等种类。其中，Spartan-2 最高可达 20 万系统门，Spartan-2E 最高可达 60 万系统门，Spartan-3 最高可达 500 万门，Spartan-3A 和 Spartan-3E 不仅系统门数更大，还增强了大量的内嵌专用乘法器和专用块 RAM 资源，具备实现复杂数字信号处理和片上可编程系统的能力。

(1) Spartan-2 系列

Spartan-2 在 Spartan 系列的基础上继承了更多的逻辑资源，可达到更高的性能，芯片密度高达 20 万系统门。由于采用了成熟的 FPGA 结构，它支持流行的接口标准，具有适量的逻辑资源和片内 RAM，并提供灵活的时钟处理，可以运行 8 位 PicoBlaze 软核，主要应用于各类低端产品中。其主要特点如下所述：

- 采用 0.18μm 工艺，密度达到 5292 逻辑单元；
- 系统时钟可以达到 200MHz；
- 系统最大门数为 20 万门，具有延迟数字锁相环；
- 具有可编程用户 I/O；
- 具有片上块 RAM 存储资源。

Spartan-2 系列产品的主要技术特征如表 1-2 所示。

表 1-2 Spartan-2 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC2S15	15k	216	6144b	16Kb	0	0	86	0
XC2S30	30k	486	13 824b	24Kb	0	0	132	0
XC2S50	50k	864	24 567b	32Kb	0	0	176	0
XC2S100	100k	1350	38 400b	40Kb	0	0	196	0
XC2S150	150k	1944	55 296b	48Kb	0	0	260	0
XC2S200	200k	2646	75 264b	56Kb	0	0	284	0

(2) Spartan-2E 系列

Spartan-2E 基于 Virtex-E 架构，具有比 Spartan-2 更多的逻辑门、用户 I/O 和更高的性能。Xilinx 还为其提供了包括存储器控制器、系统接口、DSP、通信以及网络等 IP 核，并可以运行 CPU 软核，对 DSP 有一定的支持。其主要特点如下所述：

- 采用 0.15μm 工艺，密度达到 15 552 逻辑单元；
- 最高系统时钟可达 200MHz；
- 最大门数为 60 万门，最多具有 4 个延时锁相环；
- 核电压为 1.2V，I/Q 电压可为 1.2V、2.5V、3.3V，支持 19 个可选的 I/O 标准；
- 最大可达 288Kb 的块 RAM 和 221Kb 的分布式 RAM。

Spartan-2E 系列产品的主要技术特征如表 1-3 所示。

表 1-3 Spartan-2E 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC2S50E	50k	964	24 576b	32Kb	0	0	182	83
XC2S100E	110k	1350	38 400b	40Kb	0	0	202	86
XC2S150E	150k	1944	55 296b	48Kb	0	0	265	114
XC2S200E	200k	2646	752 645b	56Kb	0	0	289	120
XC2S300E	300k	3456	98 304b	64Kb	0	0	329	120
XC2S400E	400k	5400	153 600b	160Kb	0	0	410	172
XC2S600E	600k	7776	221 184b	288Kb	0	0	514	205

(3) Spartan-3 系列

Spartan-3 基于 Virtex-Ⅱ FPGA 架构，采用 90nm 技术，8 层金属工艺，系统门数超过 500 万，内嵌了硬核乘法器和数字时钟管理模块。从结构上看，Spartan-3 将逻辑、存储器、数学运算、数字处理器、I/O 以及系统管理资源完美地结合在一起，使之有更高层次、更广泛的应用。它获得了商业上的成功，占据了较大份额的中低端市场。其主要特性如下：

- 采用 90nm 工艺，密度高达 74 880 逻辑单元；
- 最高系统时钟为 340MHz；
- 具有 18×18 的专用乘法器；
- 核电压为 1.2V，端口电压为 3.3V、2.5V、1.2V，支持 24 种 I/O 标准；
- 高达 520Kb 的分布式 RAM 和 1872Kb 的块 RAM；
- 具有片上时钟管理模块(DCM)；
- 具有嵌入式 XtremeDSP 功能，每秒可执行 3300 亿次乘加。

Spartan-3 系列产品的主要技术特征如表 1-4 所示。

表 1-4 Spartan-3 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC3S50	50k	864	12Kb	72Kb	4	2	124	56
XC3S200	200k	2260	30Kb	216Kb	12	4	173	76
XC3S400	400k	4032	56Kb	288Kb	16	4	264	116
XC3S1000	1000k	8640	120Kb	432Kb	24	4	391	175
XC3S1500	1500k	14 976	208Kb	576Kb	32	4	487	221
XC3S2000	2000k	23 040	320Kb	720Kb	40	4	565	270
XC3S4000	4000k	31 104	432Kb	1728Kb	96	4	712	312
XC3S5000	5000k	37 440	520Kb	1872Kb	104	4	784	344

(4) Spartan-3A/3A DSP/3AN 系列

Spartan-3A 在 Spartan-3 和 Spartan-3E 平台的基础上整合了各种创新特性，帮助客户极大地削减了系统总成本。它利用独特的器件 DNA ID 技术，实现业内首款 FPGA 电子序列号；提供了经济、功能强大的机制来防止发生篡改、克隆和过度设计的现象；具有集成式看门狗监控功能的增强型多重启动特性。它支持商用 Flash 存储器，有助于削减系统总成本。其主要特性为：

- 采用 90nm 工艺，密度高达 74 880 逻辑单元；
- 工作时钟范围为 5MHz～320MHz；
- 领先的连接功能平台，具有最广泛的 I/O 标准(26 种，包括新的 TMDS 和 PPDS)支持；
- 利用独特的 Device DNA 序列号，实现业内首个功能强大的防克隆安全特性；
- 5 个器件，具有高达 1.4M 的系统门数和 502 个 I/O；
- 灵活的功耗管理。

Spartan-3A 系列产品的主要技术特征如表 1-5 所示。

表 1-5 Spartan-3A 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC3S50A	50k	864	11Kb	54Kb	3	2	144	64
XC3S200A	200k	2016	28Kb	288Kb	16	4	248	112
XC3S400A	400k	4032	56Kb	360Kb	20	4	311	142
XC3S700A	700k	6624	92Kb	360Kb	20	8	372	165
XC3S1400A	1400k	12 672	176Kb	576Kb	32	8	502	227

Spartan-3A DSP 平台提供了最具成本效益的 DSP 器件，其架构的核心就是 Xtreme DSP DSP48A Slice，还提供了性能超过 30GMAC/s、存储器带宽高达 2196Mb/s 的新型 XC3SD3400A 和 XC3SD1800A 器件。新型 Spartan-3A DSP 平台是成本敏感型 DSP 算法和需要极高 DSP 性能的协处理应用的理想之选。其主要特征如下所述：

- 采用 90nm 工艺，密度高达 74 880 逻辑单元；
- 内嵌的 DSP48A 可以工作到 250MHz；
- 采用结构化的 Select RAM 架构，提供了大量的片上存储单元；
- V_{CCAUX} 支持 2.5V 和 3.3V，对于 3.3V 的应用简化了设计；
- 低功耗效率，Spartan-3A DSP 器件具有很高的信号处理能力，最高可达 4.06 GMACs/mW。

Spartan-3A DSP 系列产品的主要技术特征如表 1-6 所示。

表 1-6 Spartan-3A DSP 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC3S1800A	1800k	18 770	260Kb	1512Kb	84	8	519	227
XC3S3400A	3400k	26 856	373Kb	2268Kb	126	8	469	213

Spartan-3AN 芯片为最高级别系统集成的非易失性安全 FPGA，它提供下列两个独特的性能：先进 SRAM FPGA 的大量特性和高性能，以及非易失性 FPGA 的安全、节省板空

间和易于配置的特性。Spartan-3AN 平台是对空间要求严苛和/或安全应用及低成本嵌入式控制器的理想选择。Spartan-3AN 平台的关键特性包括：

- 业界首款 90nm 非易失性 FPGA。它可以实现灵活的、低成本安全性能的 Device DNA 电子序列号；
- 业内最大的片上用户 Flash，容量高达 11Mb；
- 提供最广泛的 I/O 标准支持，包括 26 种单端与差分信号标准；
- 灵活的电源管理模式，休眠模式下可节省超过 40%的功耗；
- 5 个器件，具有高达 1.4M 的系统门和 502 个 I/O。

Spartan-3AN 系列产品的主要技术特征如表 1-7 所示。

表 1-7 Spartan-3AN 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC3S50AN	50	792	11Kb	54Kb	3	2	108	50
XC3S200AN	200	2016	28Kb	288Kb	16	4	195	90
XC3S400AN	400	4032	56Kb	360Kb	20	4	311	142
XC3S700AN	700	6624	92Kb	360Kb	20	8	372	165
XC3S1400AN	1400	12 672	176Kb	576Kb	32	8	502	227

(5) Spartan-3E 系列

Spartan-3E 是目前 Spartan 系列最新的产品，具有系统门数从 10 万到 160 万的多款芯片，是在 Spartan-3 成功的基础上改进的产品。它提供了比 Spartan-3 更多的 I/O 端口和更低的单位成本，是 Xilinx 公司性价比最高的 FPGA 芯片。由于它更好地利用了 90nm 技术，在单位成本上实现了更多的功能和处理带宽，所以 Spartan-3E 是 Xilinx 公司新的低成本产品代表，是 ASIC 的有效替代品。它主要面向消费电子应用，如宽带无线接入、家庭网络接入以及数字电视设备等，其主要特点如下：

- 采用 90nm 工艺；
- 大量用户 I/O 端口，最多可支持 376 个 I/O 端口或者 156 对差分端口；
- 端口电压为 3.3V、2.5V、1.8V、1.5V、1.2V；
- 单端端口的传输速率可以达到 622Mb/s，支持 DDR 接口；
- 最多可达 36 个 18×18 的专用乘法器、648Kb 块 RAM、231Kb 分布式 RAM；
- 宽的时钟频率 5MHz～300MHz 以及多个专用片上数字时钟管理(DCM)模块。

Spartan-3E 系列产品的主要技术特征如表 1-8 所示。

表 1-8 Spartan-3E 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC3S100E	100k	960	15Kb	72Kb	4	2	108	40
XC3S250E	250k	2448	38Kb	216Kb	12	4	172	68
XC3S500E	500k	4656	73Kb	360Kb	20	4	232	92
XC3S1200E	1200k	8672	136Kb	504Kb	28	8	304	124
XC3S1600E	1600k	14 752	231Kb	648Kb	36	8	376	156

2. Virtex 类

Virtex 系列是 Xilinx 的高端产品，也是业界的顶级产品。Xilinx 公司正是凭借 Virtex 系列产品赢得市场，从而获得 FPGA 供应商领头羊的地位。可以说，Xilinx 公司以其 Virtex-5、Virtex-4、Virtex-Ⅱ Pro 和 Virtex-Ⅱ 系列 FPGA 产品引领现场可编程门阵列行业，主要面向电信基础设施、汽车工业、高端消费电子等应用。目前的主流芯片包括 Virtex-2、Virtex-2 Pro、Virtex-4 和 Virtex-5 等种类。

(1) Virtex-2 系列

Virtex-2 系列具有优秀的平台解决方案，这进一步提升了其性能；且内置 IP 核的硬核技术可以将硬 IP 核分配到芯片的任何地方，具有比 Virtex 系列更多的资源和更高的性能。其主要特征如下所述：

- 采用 0.15/0.12μm 工艺；
- 核电压为 1.5V，工作时钟可以达到 420MHz；
- 支持 20 多种 I/O 接口标准；
- 内嵌了多个 18×18 硬核乘法器，提高了 DSP 处理能力；
- 具有完全的系统时钟管理功能，多达 12 个 DCM 模块。

Virtex-2 系列产品的主要技术特征如表 1-9 所示。

表 1-9 Virtex-2 系列 FPGA 主要技术特征

型号	系统门数	Slice 数目	分布式 RAM 容量	块 RAM 容量	专用乘法器数	DCM 数目	最大可用 I/O 数	最大差分 I/O 对数
XC2V40	40k	256	8Kb	72Kb	4	4	88	0
XC2V80	80k	512	16Kb	144Kb	8	4	120	0
XC2V250	250k	1536	48Kb	432Kb	24	8	200	0
XC2V500	500k	3072	96Kb	576Kb	32	8	264	0
XC2V1000	1000k	5120	160Kb	720Kb	40	8	432	0
XC2V1500	1500k	7680	240Kb	864Kb	48	8	528	0
XC2V2000	2000k	10 752	336Kb	1008Kb	56	8	624	0
XC2V3000	3000k	14 336	448Kb	1728Kb	96	12	720	0
XC2V4000	4000k	23 040	720Kb	2160Kb	120	12	912	0
XC2V6000	6000k	33 792	1056Kb	2592Kb	144	12	1104	0
XC2V8000	8000k	46 592	1456Kb	3024Kb	168	12	1108	0

(2) Virtex-2 Pro 系列

Virtex-2 Pro 系列在 Virtex-2 的基础上增强了嵌入式处理功能，内嵌了 PowerPC405 内核，还包括了先进的主动互连(Active Interconnect)技术，以解决高性能系统所面临的挑战。此外，它还增加了高速串行收发器，提供了千兆以太网的解决方案。其主要特征如下所述：

- 采用 0.13μm 工艺；
- 核电压为 1.5V，工作时钟可以达到 420MHz；
- 支持 20 多种 I/O 接口标准；

- 增加了 2 个采用高性能 RISC 技术、频率高达 400MHz 的 PowerPC 处理器；
- 增加多个 3.125Gb/s 速率的 Rocket 串行收发器；
- 内嵌了多个 18×18 硬核乘法器，提高了 DSP 处理能力；
- 具有完全的系统时钟管理功能，多达 12 个 DCM 模块。

Virtex-2 Pro 系列产品的主要技术特征如表 1-10 所示。

表 1-10 Virtex-2 Pro 系列 FPGA 主要技术特征

型号	Slice 数目	分布式 RAM 容量	块 RAM 容量	PowerPC	专用乘法器数	DCM 数目	Rocket I/O	最大可用 I/O 数
XC2VP2	1408	44Kb	216Kb	0	12	4	4	204
XC2VP4	3008	94Kb	504Kb	1	28	4	4	348
XC2VP7	4928	154Kb	792Kb	1	44	8	8	396
XC2VP20	9280	290Kb	1584Kb	2	88	8	8	564
XC2VP30	13 696	428Kb	2448Kb	2	136	8	8	644
XC2VP40	19 392	606Kb	3456Kb	2	192	8	12	804
XC2VP50	23 616	738Kb	4176Kb	2	232	8	12	852
XC2VP70	33 088	1034Kb	5904Kb	2	328	8	16 或 20	996
XC2VP100	44 096	1378Kb	7992Kb	2	444	12	20	1164
XC2VP125	55 616	1738Kb	10 008Kb	4	556	12	20 或 24	1200

(3) Virtex-4 系列

Virtex-4 器件整合了 20 万个逻辑单元，具有 500MHz 的性能和无可比拟的系统特性。Virtex-4 产品基于新的高级硅片组合模块(ASMBL)架构，提供了一个多平台方式(LX、SX、FX)，使设计者可以根据需求选用不同的开发平台。其主要特点如下所述：

- 采用了 90nm 工艺，集成了 20 万个逻辑单元；
- 系统时钟 500MHz；
- 采用了集成 FIFO 控制逻辑的 500MHz Smart RAM 技术；
- 具有 DCM 模块、PMCD 相位匹配时钟分频器和片上差分时钟网络；
- 每个 I/O 都集成了 ChipSync 源同步技术的 1Gb/s I/O；
- 具有超强的信号处理能力，集成了数以百计的 XtremeDSP Slice，单片最大的处理速率为 256GMAC/s。

Virtex-4 LX 平台 FPGA 的特点是密度高达 20 万逻辑单元，是全球逻辑密度最高的 FPGA 系列之一，适合对逻辑门需求高的设计应用。

Virtex-4 SX 平台提高了 DSP、RAM 单元与逻辑单元的比例，最多可以提供 512 个 XtremeDSP 硬核，可以工作在 500MHz，其最大的处理速率为 256GMAC/s，可以以其创建 40 多种不同功能，并能多个组合实现更大规模的 DSP 模块。与 Virtex-2 Pro 系列相比，Virtex-4 SX 平台大大降低了成本和功耗，具有极低的 DSP 成本。SX 平台的 FPGA 非常适合应用于高速、实时的数字信号处理领域。

Virtex-4 FX 平台内嵌了 1～2 个 32 位 RISC PowerPC 处理器，提供了 4 个 1300 Dhrystone MIPS、10/100/1000 自适应的以太网 MAC 内核，协处理器控制器单元(APU)允许处理器在 FPGA 中构造专用指令，使 FX 器件的性能达到固定指令方式的 20 倍。此外，FX 平台还

包含 24 个 Rocket I/O 串行高速收发器，支持常用的 0.6Gb/s、1.25Gb/s、2.5Gb/s、3.125Gb/s、4Gb/s、6.25Gb/s、10Gb/s 等高速传输速率。FX 平台适用于复杂计算和嵌入式处理应用。

Virtex-4 系列产品的主要技术特征如表 1-11 所示。

表 1-11 Virtex-4 系列 FPGA 主要技术特征

型号	Slice 数目	分布式 RAM 容量	块 RAM 容量	PowerPC	XtremeDSP Slice	DCM 数目	Rocket I/O	以太网 MAC
XC4VLX15	6144	96Kb	864Kb	0	32	4	0	0
XC4VLX25	10 572	168Kb	1296Kb	0	48	8	0	0
XC4VLX40	18 432	288Kb	1728Kb	0	64	8	0	0
XC4VLX60	26 624	416Kb	2880Kb	0	64	8	0	0
XC4VLX80	35 840	560Kb	3600Kb	0	80	12	0	0
XC4VLX100	49 152	768Kb	4320Kb	0	96	12	0	0
XC4VLX160	67 584	1056Kb	5184Kb	0	96	12	0	0
XC4VLX200	89 088	1392Kb	6048Kb	0	96	12	0	0
XC4VSX25	10 240	160Kb	2304Kb	0	128	4	0	0
XC4VSX35	15 360	240Kb	3456Kb	0	192	8	0	0
XC4VSX55	24 576	384Kb	5760Kb	0	320	8	0	0
XC4VFX12	5472	86Kb	648Kb	1	32	4	0	2
XC4VFX20	8544	134Kb	1224Kb	1	32	4	8	2
XC4VFX40	18 624	291Kb	2592Kb	2	48	8	12	4
XC4VFX60	25 280	395Kb	4176Kb	2	128	12	16	4
XC4VFX100	42 176	659Kb	6768Kb	2	160	12	20	4
XC4VFX140	63 168	987Kb	9936Kb	2	192	20	24	4

(4) Virtex-5 系列

Virtex-5 系列是 Xilinx 公司最新一代的 FPGA 产品，计划提供 4 种新型平台，每种平台都在高性能逻辑、串行连接功能、信号处理和嵌入式处理性能方面实现了最佳平衡。现有的 3 款平台为 LX、LXT 以及 SXT。LX 针对高性能逻辑进行了优化，LXT 针对具有低功耗串行连接功能的高性能逻辑进行了优化，SXT 针对具有低功耗串行连接功能的 DSP 和存储器密集型应用进行了优化。其主要特点如下：

- 采用了最新的 65nm 工艺，结合低功耗 IP 块将动态功耗降低了 35%；此外，还利用 65nm 三栅极氧化层技术保持低静态功耗；
- 利用 65nm ExpressFabric 技术，实现了真正的 6 输入 LUT，并将性能提高了 2 个速度级别；
- 内置有用于构建更大型阵列的 FIFO 逻辑和 ECC 的增强型 36Kbit 块 RAM，带有低功耗电路，可以关闭未使用的存储器；
- 逻辑单元 33 万个，可以实现无与伦比的高性能；
- I/O 管脚 1200 个，可以实现高带宽存储器/网络接口，1.25Gb/s LVDS；
- 低功耗收发器 24 个，可以实现 100Mb/s～3.75Gb/s 高速串行接口；
- 核电压为 1V，系统时钟 550MHz；

- 550MHz DSP48E Slice 内置有 25×18 MAC，提供 352 GMAC 的性能，能够在将资源使用率降低 50%的情况下，实现单精度浮点运算；
- 利用内置式 PCI Express 端点和以太网 MAC 模块提高面积效率；
- 更加灵活的时钟管理管道(Clock Management Tile)结合了用于进行精确时钟相位控制与抖动滤除的新型 PLL 和用于各种时钟综合的数字时钟管理器(DCM)；
- 采用了第二代稀疏锯齿型(sparse chevron)封装，改善了信号完整性，并降低了系统成本；
- 增强了器件配置，支持商用 Flash 存储器，从而降低了成本。

现有 Virtex-5 系列产品的主要技术特征如表 1-12 所示。

表 1-12 Virtex-5 系列 FPGA 主要技术特征

型号	Virtex-5 Slice	分布式 RAM 容量	块 RAM 容量	以太网 MAC	DSP48E Slice	Rocket I/O	I/O bank 数目	最大可用 I/O 数
XC5VLX30	4800	320Kb	1152Kb	0	32	0	13	400
XC5VLX50	7200	480Kb	1728Kb	0	48	0	17	560
XC5VLX85	12 950	840Kb	3456Kb	0	48	0	17	560
XC5VLX110	17 280	1120Kb	4608Kb	0	64	0	23	800
XC5VLX220	34 560	2280Kb	6912Kb	0	128	0	23	800
XC5VLX330	51 840	3520Kb	10 368Kb	0	192	0	23	1200
XC5VLX30T	4800	320Kb	1296Kb	4	32	8	12	360
XC5VLX50T	7200	480Kb	2160Kb	4	48	12	15	450
XC5VLX85T	12 960	840Kb	3888Kb	4	48	12	15	450
XC5VLX110T	17 280	1120Kb	5328Kb	4	64	16	20	680
XC5VLX220T	34 560	2280Kb	7632Kb	4	128	16	20	680
XC5VLX330T	51 840	3420Kb	11 664Kb	4	192	24	27	980
XC5VSX35T	5440	520Kb	3024Kb	4	192	8	12	360
XC5VSX50T	8160	760Kb	4750Kb	4	288	12	15	480
XC5VSX95T	14 720	1520Kb	8784Kb	4	640	16	18	640

注：一个 Virtex-5 Slice 具有 4 个 LUT 和 4 个触发器，而前文所提及的常规 Slice 只包含 2 个 LUT 和 2 个触发器。每个 DSP48E 包含一个 25×18 位的硬核乘法器、一个加法器和一个累加器。

1.4.2 Xilinx PROM 芯片介绍

Xilinx 公司的 Platform Flash PROM 能为所有型号的 Xilinx FPGA 提供非易失性存储。全系列 PROM 的容量范围为 1Mbit～32Mbit，兼容任何一款 Xilinx 的 FPGA 芯片，具备完整的工业温度特性(－40℃～85℃)，支持 IEEE 1149.1 所定义的 JTAG 边界扫描协议。

PROM 芯片可以分成 3.3V 核电压的 XCF××S 系列和 1.8V 核电压的 XCF××P 系列两大类，后者主要面向高端的 FPGA 芯片，支持并行配置、设计修订(Designing Revisioning)和数据压缩(Compression)等高级功能，以容量大、速度快著称。PROM 芯片的详细参数如表 1-13 所示。

表 1-13 Xilinx 公司 PROM 芯片总结

型号	容量	V_{CCINT}	封　装	JTAG ISP 配置	串行配置	并行配置	设计修订	数据压缩
XCF01S	1Mbit	3.3V	VO20/VOG20	√	√			
XCF02S	2Mbit	3.3V	VO20/VOG20	√	√			
XCF04S	4Mbit	3.3V	VO20/VOG20	√	√			
XCF08P	8Mbit	1.8V	VO48/VOG48/FS48/FSG48	√	√	√	√	√
XCF16P	16Mbit	1.8V	VO48/VOG48/FS48/FSG48	√	√	√	√	√
XCF32P	32Mbit	1.8V	VO48/VOG48/FS48/FSG48	√	√	√	√	√

XCF××S 系列包含 XCF01S、XCF02S 和 XCF04S(容量分别为 1Mbit、2Mbit 和 4Mbit)，其共同特征是有 3.3V 核电压、串行配置接口以及 VO20 封装。XCF××S 内部控制信号、数据信号、时钟信号和 JTAG 信号的整体结构如图 1-8 所示。

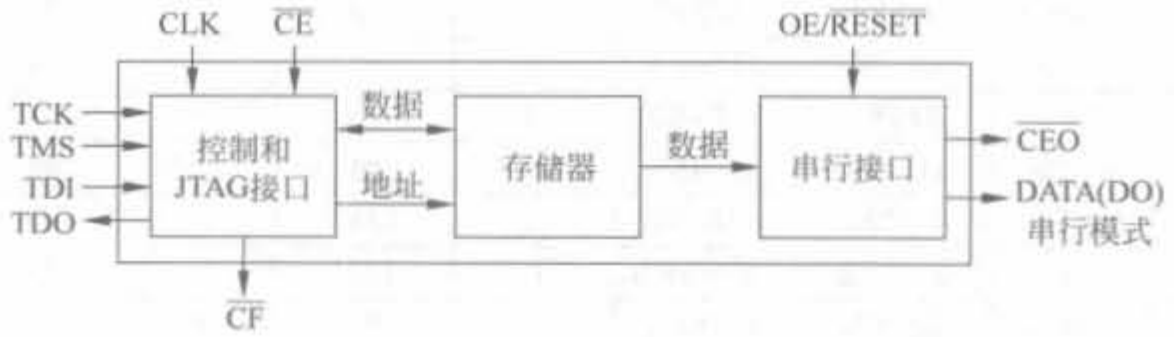


图 1-8 XCF01S/XCF02S/XCF04S PROM 结构组成框图

XCF××P 系列有 XCF08P、XCF16P 和 XCF32P(容量分别为 8Mbit、16Mbit 和 32Mbit)，其共同特征是有 1.8V 核电压、串行或并行配置接口、设计修订、内嵌的数据压缩器、FS48 封装或 VQ48 封装和内嵌振荡器。XCF××P 内部控制信号、数据信号、时钟信号和 JTAG 信号的整体结构如图 1-9 所示，其先进的结构和更高的集成度在使用中带来了极大的灵活性。

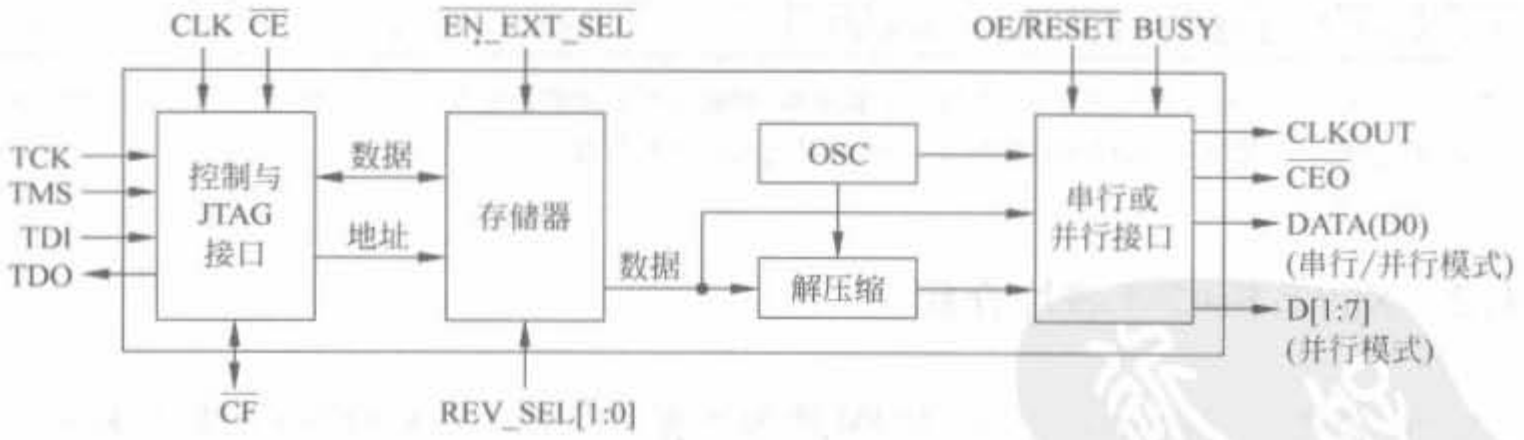


图 1-9 XCF08P/XCF16P/XCF32P PROM 结构组成框图

值得一提的是 XCF××P 系列的设计修正和数据压缩这两个功能。设计修订功能在 FPGA 加电启动时改变其配置数据，根据所需来改变 FPGA 的功能，允许用户在单个 PROM 中将多种配置存储为不同的修订版本，从而简化 FPGA 配置更改。在 FPGA 内部加入少量的逻辑，用户就能在 PROM 中存储多达 4 个不同修订版本之间的动态切换。数据压缩功能可以节省 PROM 的空间，最高可节约 50%的存储空间，从而降低成本，是一项非

常实用的技术。当然，如果编程时在软件端采用了压缩模式，则需要一定的硬件配置来完成相应的解压缩。

1.5 本章小结

本章首先介绍了 FPGA 器件的基本概念和发展历史；然后详细介绍了 FPGA 器件的工作原理、芯片结构以及组成部件，并在此基础上讨论了 FPGA 的开发流程，给出传统的开发方法以及 SOC 阶段的设计方法；最后列举了 Xilinx 公司的主流 FPGA 芯片和 PROM 芯片，这些芯片被广泛地应用在数字系统设计中，熟悉并了解其性能指标和参数是 Xilinx 芯片开发人员所必需的。

第2章

Verilog HDL语言基础

CHAPTER 2

随着人们对数十万门、数百万门乃至数千万门电路设计需求的增加，依靠传统的原理图输入已经不能满足设计人员的要求，因此基于硬件描述语言（HDL）的设计方式应运而生。HDL是以文本形式来描述数字系统硬件结构和行为的语言，用它可以表示逻辑电路图、逻辑表达式，还可以表示数字逻辑系统所完成的逻辑功能。本书的实例代码开发都是基于Verilog语言的，因此本章首先对Verilog作简单的介绍，然后详细地解释其语法标准，并针对几个专题深入地讨论，最后给出常用程序实例。关于VHDL和System C的使用，可参考相关文献，限于篇幅，本书不对它们展开分析。

2.1 Verilog HDL语言简介

Verilog HDL和VHDL是目前世界上最流行的两种硬件描述语言（Hardware Description Language，HDL），均为IEEE标准，被广泛地应用于基于可编程逻辑器件的项目开发。二者都是在20世纪80年代中期开发出来的，前者由Gateway Design Automation公司（该公司于1989年被Cadence公司收购）开发，后者由美国军方研发。

HDL语言以文本形式来描述数字系统硬件结构和行为，是一种用形式化方法来描述数字电路和系统的语言。利用它，设计人员可以从上层到下层来逐层描述自己的设计思想，即用一系列分层次的模块来表示复杂的数字系统，并逐层进行验证、仿真，再把具体的模块组合由综合工具转化成门级网表，然后利用布局布线工具把网表转化为具体电路结构的实现。目前，这种自顶向下的方法已被广泛使用。概括地讲，HDL语言有以下主要特征：

- HDL语言既包含一些高级程序设计语言的结构形式，同时兼顾描述硬件线路连接的具体结构。
- 通过使用结构级行为描述，可以在不同的抽象层次描述设计。HDL语言采用自顶向下的数字电路设计方法，主要包括3个领域的5个抽象层次。
- HDL语言是并行处理的，具有同一时刻执行多任务的能力。这和一般高级设计语言（例如C语言等）串行执行的特征是不同的。
- HDL语言具有时序的概念。一般的高级编程语言是没有时序概念的。但在硬件电路中，从输入到输出总是有延时存在，为了描述这一特征，需要引入延时的概念。HDL语言不仅可以描述硬件电路的功能，还可以描述电路的时序。

2.1.1 Verilog HDL 语言的历史

1983 年,Gateway Design Automation(GDA)硬件描述语言公司的 Philip Moorby 首创了 Verilog HDL。后来,Moorby 成为 Verilog HDL-XL 的主要设计者和 Cadence 公司的第一合伙人。1984—1986 年,Moorby 设计出第一个关于 Verilog HDL 的仿真器,并提出了用于快速门级仿真的 XL 算法,使 Verilog HDL 语言得到迅速发展。1987 年,Synonsys 公司开始使用 Verilog HDL 行为语言作为综合工具的输入。1989 年,Cadence 公司收购了 Gateway 公司,Verilog HDL 成为 Cadence 公司的私有财产。1990 年初,Cadence 公司把 Verilog HDL 和 Verilog HDL-XL 分开,并公开发布了 Verilog HDL。随后成立的 OVI (Open Verilog HDL International)组织负责 Verilog HDL 的发展并制定有关标准,OVI 由 Verilog HDL 的使用者和 CAE 供应商组成。1993 年,几乎所有 ASIC 厂商都开始支持 Verilog HDL,并且认为 Verilog HDL-XL 是最好的仿真器。同时,OVI 推出 2.0 版本的 Verilog HDL 规范,IEEE 则将 OVI 的 Verilog HDL 2.0 作为 IEEE 标准的提案。1995 年 12 月,IEEE 制定了 Verilog HDL 的标准 IEEE 1364-1995。目前,最新的 Verilog 语言版本是 2000 年 IEEE 公布的 Verilog 2001 标准,它大幅度地提高了系统级和可综合性能。

2.1.2 Verilog HDL 的主要能力

Verilog HDL 既是一种行为描述语言,也是一种结构描述语言。如果按照一定的规则和风格编写代码,就可以将功能行为模块通过工具自动转化为门级互连的结构模块。这意味着,利用 Verilog 语言所提供的功能,就可以构造一个模块间的清晰结构来描述复杂的大型设计,并对所需的逻辑电路进行严格的设计。

下面列出 Verilog 语言的主要功能:

- 可描述顺序执行或并行执行的程序结构;
- 用延迟表示式或事件表达式来明确地控制过程的启动时间;
- 通过命名的事件来触发其他过程里的激活行为或停止行为;
- 提供了条件和循环等程序结构;
- 提供了可带参数且非零延续时间的任务程序结构;
- 提供了可定义新的操作符的函数结构;
- 提供了用于建立表达式的算术运算符、逻辑运算符和位运算符;
- 提供了一套完整的表示组合逻辑基本元件的原语;
- 提供了双向通路和电阻器件的描述;
- 可建立 MOS 器件的电荷分享和衰减模型;
- 可以通过构造性语句精确地建立信号模型。

表 2-1 给出 Verilog HDL 的表述能力。

表 2-1 Verilog HDL 语言能力总结

描述级别	抽象级别	功能描述	物理模型
行为级	系统级	用语言提供的高级结构能够实现所设计模块外部性能的模型	芯片、电路板和物理划分的子模块
	算法级	用语言提供的高级功能能够实现算法运行的模型	部件之间的物理连接，电路板
	RTL 级	描述数据如何在寄存器之间流动和如何处理、控制这些数据流动的模型	芯片、宏单元
逻辑级	门级	描述逻辑门和逻辑门之间连接的模型	标准单元布图
电路级	开关级	描述器件中三极管和存储节点以及它们之间连接的模型	晶体管布图

此外，Verilog HDL 语言还有一个重要特征，就是和 C 语言的风格有很多相似之处，学习起来比较容易。

2.1.3 Verilog HDL 和 VHDL 的区别

Verilog HDL 和 VHDL 都是用于逻辑设计的硬件描述语言。VHDL 在 1987 年成为 IEEE 标准，Verilog HDL 则在 1995 年才成为 IEEE 标准，这是因为前者是美国军方组织开发的，后者则是从民间公司转化而来，要成为国际标准，必须放弃专利。相比而言，Verilog HDL 具有更强的生命力。

Verilog HDL 和 VHDL 的相同点在于：都能形式化地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；可以简化电路行为的描述；具有电路仿真和验证机制；支持电路描述由高层到低层的综合转换；与实现工艺无关；便于管理和设计重用。

Verilog HDL 和 VHDL 又有各自的特点。由于 Verilog HDL 推出较早，因而拥有更广泛的客户群体、更丰富的资源。Verilog HDL 还有一个优点就是容易掌握，如果具有 C 语言学习的基础，很快就能够掌握它。而 VHDL 需要 Ada 编程语言基础，一般需要半年以上的专业培训才能够掌握。传统观点认为 Verilog HDL 在系统级抽象方面较弱，不太适合特大型的系统。但经过 Verilog 2001 标准的补充之后，系统级表述性能和可综合性能有了大幅度提高。当然，这两种语言仍处于不断完善的过程中，都在朝着更高级描述语言的方向发展。

2.1.4 Verilog HDL 设计方法

1. 自下而上的设计方法

自下而上的设计是传统的设计方法，是从基本单元出发，对设计进行逐层划分的过程。这种设计方法与用电子元件在模拟实现板上建立一个系统的步骤有密切的关系，其优、缺点分别如下：

1）优点

设计人员对这种设计方法比较熟悉；实现各个子模块所需的时间较短。

2）缺点

对系统的整体功能把握不足；由于必须先对多个子模块进行设计，因此实现整个系统的功能所需的时间长。另外，对设计人员之间的相互协作有较高的要求。

2. 自上而下的设计方法

自上而下的设计是从系统级开始，把系统划分为基本单元，再把基本单元划分为下一层次的基本单元，直到可用 EDA 元件实现为止。这种方法的优、缺点分述如下。

1）优点

在设计周期开始就做好了系统分析；由于设计的主要仿真和调试过程是在高层完成的，所以能够在早期发现结构设计上的错误，避免了设计工作的浪费，方便了系统的划分和整个项目的管理，可减少设计人员的劳动，避免了重复设计。

2）缺点

得到的最小单元不标准，且制造成本高。

3. 混合的设计方法

复杂数字逻辑电路和系统设计过程通常是以上两种设计方法的结合，设计时需要考虑多个目标的综合平衡。高层系统用自上而下的设计方法实现，而使用自下而上的方法从库元件或以往的设计库中调用已有的设计单元。混合设计方法兼有以上两种方法的优点，并且可使用先进的矢量测试方法。

在实际工程开发中，往往采取混合设计方法。首先从高层开始按照功能划分相对独立、规模较大的功能子模块；然后在每个模块内部根据各个底层模块的实现难度以及时序要求等方面进行综合考虑，在高层需求的前提下，从底层模块一级一级向上开发，最后连接成整个功能子模块；最后，将子模块有效地组合成整个设计。

2.2 Verilog HDL 基本程序结构

用 Verilog HDL 描述的电路设计就是该电路的 Verilog HDL 模型，也称为模块，是 Verilog 的基本描述单位。模块描述某个设计的功能或结构，以及与其他模块通信的外部接口。一般来说，一个文件就是一个模块，但并不绝对如此。模块是并行运行的，通常需要一个高层模块通过调用其他模块的实例来定义一个封闭的系统，包括测试数据和硬件描述。一个模块的基本架构如下：

```
module module_name (port_list)
    //声明各种变量、信号
    reg //寄存器
    wire//线网
    parameter//参数
    input//输入信号
    output//输出信号
    inout//输入/输出信号
    function//函数
    task//任务
    ...
```

```
        //程序代码
        initial assignment
        always assignment
        module assignment
        gate assignment
        UDP assignment
        continous assignment
endmodule
```

说明部分用于定义不同的项，例如模块描述中使用的寄存器和参数。语句用于定义设计的功能和结构。说明部分可以分散于模块的任何地方，但是变量、寄存器、线网和参数等的说明必须在使用前出现。一般的模块结构如下：

```
module <模块名>(<端口列表>)
<定义>
<模块条目>
endmodule
```

其中，<定义>用来指定数据对象为寄存器型、存储器型、线型以及过程块。<模块条目>可以是 initial 结构、always 结构、连续赋值或模块实例。

下面给出一个简单的 Verilog 模块，实现一个二选一选择器。

例 2-1 二选一选择器(见图 2-1)的 Verilog 实现。

```
module muxtwo(out,a,b,sl);
    input a,b,sl;
    output out;
    reg out;
    always @ (sl or a or b)
        if (!sl) out = a;
        else out = b;
endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-1 所示。

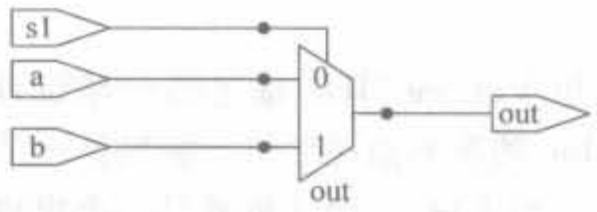


图 2-1 二选一选择器的 RTL 级结构示意图

模块的名字是 muxtwo。模块有 4 个端口：3 个输入端口 a、b 和 sl，一个输出端口 out。由于没有定义端口的位数，所有端口大小都默认为 1 位；由于没有定义端口 a、b 和 sl 的数据类型，这 3 个端口都默认为线网型数据类型。输出端口 out 定义为 reg 类型。如果没有明确的说明，则端口都是线网型的，且输入端口只能是线网型的。

2.3 Verilog HDL 语言的数据类型和运算符

本节主要介绍 Verilog HDL 语言的基本要素，包括标志符、数据类型、模块端口、值集合以及表达式等。

2.3.1 标志符

标志符可以是一组字母、数字、下划线和$符号的组合，且标志符的第一个字符必须是字母或者下划线。另外，标志符是区别大小写的。下面给出标志符的几个例子：

```
Clk_100MHz
diag_state
_ce
P_o1_02
```

需要注意的是，Verilog HDL定义了一系列保留字，叫做关键字，具体资料可查阅相关标准。只有小写的关键字才是保留字。因此在实际开发中，建议将不确定是否是保留字的标志符首字母大写。例如，标志符if(关键字)与标志符IF是不同的。

2.3.2 数据类型

数据类型用来表示数字电路硬件中的数据存储和传送元素。Verilog HDL中总共有19种数据类型，本章只介绍4个常用的数据类型：wire型、reg型、memory型和parameter型，其他类型将在后续章节中逐步介绍。

1. wire型

wire型数据常用来表示以assign关键字指定的组合逻辑信号。Verilog程序模块中的输入、输出信号类型默认为wire型。wire型信号可以用作方程式的输入，也可以用作“assign”语句或者实例元件的输出。

wire型信号的定义格式如下：

```
wire[n-1:0] 数据名1,数据名2,…,数据名N;
```

这里，总共定义了N条线，每条线的位宽为n。下面给出几个例子：

```
wire[9:0] a,b,c; // a、b、c都是位宽为10的wire型信号
wire d;
```

2. reg型

reg是寄存器数据类型的关键字。寄存器是数据存储单元的抽象，通过赋值语句可以改变寄存器存储的值，其作用相当于改变触发器存储器的值。reg型数据常用来表示always模块内的指定信号，代表触发器。通常在设计中要由always模块通过使用行为描述语句来表达逻辑关系。在always块内被赋值的每一个信号都必须定义为reg型，即赋值操作符的右端变量必须是reg型。

reg型信号的定义格式如下：

```
reg[n-1:0] 数据名1,数据名2,…,数据名N;
```

这里，总共定义了N个寄存器变量，每条线的位宽为n。下面给出几个例子：

```
reg [9:0] a,b,c; // a、b、c 都是位宽为 10 的寄存器
reg d;
```

reg 型数据的默认值是未知的。reg 型数据可以为正值或负值。但当 reg 型数据是一个表达式中的操作数时，它的值被当作无符号值，即正值。如果一个 4 位的 reg 型数据被写入－1，在表达式中运算时，其值被认为是＋15。

reg 型和 wire 型的区别在于：reg 型保持最后一次的赋值，而 wire 型需要持续地驱动。

3. memory 型

Verilog 通过对 reg 型变量建立数组来对存储器建模，可以描述 RAM、ROM 存储器和寄存器数组。数组中的每一个单元通过一个整数索引进行寻址。memory 型通过扩展 reg 型数据的地址范围来达到二维数组的效果，其定义的格式如下：

```
reg [n-1:0] 存储器名 [m-1:0];
```

其中，reg [n－1:0]定义了存储器中每一个存储单元的大小，即该存储器单元是一个 n 位位宽的寄存器；存储器后面的[m－1:0]定义了存储器的大小，即该存储器中有多少个这样的寄存器。例如：

```
reg [15:0] ROMA [7:0];
```

这个例子定义了一个存储位宽为 16 位，存储深度为 8 的存储器。该存储器的地址范围是 0～8。

需要注意的是，对存储器进行地址索引的表达式必须是常数表达式。

尽管 memory 型和 reg 型数据的定义比较接近，但二者有很大的区别。例如，一个由 n 个 1 位寄存器构成的存储器是不同于一个 n 位寄存器的。

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

一个 n 位的寄存器可以在一条赋值语句中直接赋值，一个完整的存储器则不行。

```
rega = 0;                            // 合法赋值
memb = 0;                            // 非法赋值
```

如果要对 memory 型存储单元进行读写，必须要指定地址。例如：

```
memb[0] = 1;                         // 将 memb 中的第 0 个单元赋值为 1
reg [3:0] Xrom [4:1];
Xrom[1] = 4'h0;
Xrom[2] = 4'ha;
Xrom[3] = 4'h9;
Xrom[4] = 4'hf;
```

4. parameter 型

在 Verilog HDL 中用 parameter 来定义常量，即用 parameter 来定义一个标志符表示一个常数。采用该类型可以提高程序的可读性和可维护性。

parameter 型信号的定义格式如下：

```
parameter 参数名 1 = 数据名 1;
```

下面给出几个例子：

```
parameter s1 = 1;
parameter [3:0] S0 = 4'h0,
                S1 = 4'h1,
                S2 = 4'h2,
                S3 = 4'h3,
                S4 = 4'h4;
```

2.3.3 模块端口

模块端口是指模块与外界交互信息的接口，包括 3 种类型：

(1) input：模块从外界读取数据的接口，在模块内不可写。

(2) output：模块往外界送出数据的接口，在模块内不可读。

(3) inout：可读取数据，也可以送出数据。数据可双向流动。

2.3.4 常量集合

Verilog HDL 有下列 4 种基本的数值：

(1) 0：逻辑 0 或“假”；

(2) 1：逻辑 1 或“真”；

(3) x：未知；

(4) z：高阻。

其中，x、z 是不区分大小写的。Verilog HDL 中的数字由这 4 类基本数值表示。

Verilog HDL 中的常量分为 3 类：整数型、实数型以及字符串型。下划线符号“_”可以随意用在整数和实数中，没有实际意义，只是为了提高可读性。例如，56 等效于 5_6。

1. 整数

整数型可以按如下两种方式书写：简单的十进制数格式以及基数格式。

1) 简单的十进制数格式

简单的十进制数格式的整数定义为带有一个“+”或“-”操作符的数字序列。下面是这种简易十进制形式整数的例子：

```
45    十进制数 45
-46   十进制数 -46
```

简单的十进制数格式的整数值代表一个有符号的数，其中负数可使用两种补码形式表示。例如，32 在 6 位二进制形式中表示为 100000；在 7 位二进制形式中为 0100000，这里最高位 0 表示符号位。-15 在 5 位二进制中的形式为 10001，最高位 1 表示符号位；在 6 位二进制中为 110001，最高位 1 为符号扩展位。

2) 基数表示格式

基数格式的整数格式为：

[长度]'基数 数值

长度是常量的位长，基数可以是二进制、十进制、十六进制之一。数值是基于基数的数字序列，且数值不能为负数。下面是一些具体实例：

```
6'b001001   6 位二进制数
5'o7        5 位八进制数
9'd6        9 位十进制数
```

2. 实数

实数可以用下列两种形式定义：

1）十进制计数法

```
2.0
16539.236
```

2）科学计数法

这种形式的实数举例如下，其中 e 与 E 相同：

```
235.12e2    其值为 23512
5e-4        其值为 0.0005
```

根据 Verilog 语言的定义，实数通过四舍五入隐式地转换为最相近的整数。

3. 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：

```
"counter"
```

用 8 位 ASCII 值表示的字符可看作是无符号整数，因此字符串是 8 位 ASCII 值的序列。为存储字符串“counter”，变量需要 8×7 位。

```
reg [1:8 * 7] Char;
Char = "counter";
```

2.3.5 运算符和表达式

在 Verilog HDL 语言中，运算符所带的操作数是不同的。按其所带操作数的个数可以分为 3 种：

- 单目运算符：带一个操作数，且放在运算符的右边。
- 双目运算符：带两个操作数，且放在运算符的两边。
- 三目运算符：带三个操作数，且被运算符间隔开。

Verilog HDL 语言参考了 C 语言中大多数算符的语法和句义，运算范围很广，其运算符按其功能分为下列 9 类。

1. 基本算术运算符

在 Verilog HDL 中，算术运算符又称为二进制运算符，有下列 5 种：

- ＋　加法运算符或正值运算符，如 s1＋s2；＋5；

- — 减法运算符或负值运算符，如 s1－s2；－5；
- * 乘法运算符，如 s1 * 5；
- / 除法运算符，如 s1/5；
- % 模运算符，如 s1%2。

在进行整数除法时，结果值要略去小数部分。在取模运算时，结果的符号位和模运算第一个操作数的符号位保持一致。例如：

运算表达式	结果	说明
12.5/3	4	结果为 4，小数部分省去
12%4	0	整除，余数为 0
－15%2	－1	结果取第一个数的符号，所以余数为－1
13/－3	1	结果取第一个数的符号，所以余数为 1

注意：在进行基本算术运算时，如果某一操作数有不确定的值 X，则运算结果也是不确定值 X。

2. 赋值运算符

赋值运算分为连续赋值和过程赋值两种。

1）连续赋值

连续赋值语句和过程块一样也是一种行为描述语句，有的文献中将其称为数据流描述形式，但本书将其视为一种行为描述语句。

连续赋值语句只能用来对线网型变量进行赋值，而不能对寄存器变量进行赋值，其基本的语法格式为：

```
线网型变量类型［线网型变量位宽］线网型变量名；
assign #（延时量）线网型变量名 = 赋值表达式；
```

例如：

```
wire a;
assign a = 1'b1;
```

一个线网型变量一旦被连续赋值语句赋值之后，赋值语句右端赋值表达式的值将持续对被赋值变量产生连续驱动。只要右端表达式任一个操作数的值发生变化，就会立即触发对被赋值变量的更新操作。

在实际使用中，连续赋值语句有下列几种应用：

- 对标量线网型赋值

```
wire a,b;
assign a = b;
```

- 对矢量线网型赋值

```
wire [7:0] a,b;
assign a = b;
```

- 对矢量线网型中的某一位赋值

```
wire [7:0] a,b;
```

```
assign a[3] = b[1];
```

- 对矢量线网型中的某几位赋值

```
wire [7:0] a,b;
assign a[3:0] = b[3:0];
```

- 对任意拼接的线网型赋值

```
wire a,b;
wire [1:0] c;
assign c = {a,b};
```

2）过程赋值

过程赋值主要用于两种结构化模块(initial 模块和 always 模块)中的赋值语句。在过程块中只能使用过程赋值语句(不能在过程块中出现连续赋值语句)，同时过程赋值语句也只能用在过程赋值模块中。

过程赋值语句的基本格式为：

<被赋值变量><赋值操作符><赋值表达式>

其中，<赋值操作符>是“=”或“<=”，它分别代表了阻塞赋值和非阻塞赋值类型。本书在3.3.2节将对阻塞赋值和非阻塞赋值操作进行详细解释。

过程赋值语句只能对寄存器类型的变量(reg、integer、real 和 time)进行操作。经过赋值后，上面这些变量的取值将保持不变，直到另一条赋值语句对变量重新赋值为止。过程赋值操作的具体目标可以是：

- reg、integer、real 和 time 型变量(矢量和标量)；
- 上述变量的一位或几位；
- 上述变量用{}操作符所组成的矢量；
- 存储器类型，只能对指定地址单元的整个字进行赋值，不能对其中某些位单独赋值。

例 2-2 给出一个过程赋值的例子。

```
reg c;
always @(a)
begin
  c = 1'b0;
end
```

3. 关系运算符

关系运算符总共有以下 8 种：

- > 大于
- >= 大于等于
- < 小于
- <= 小于等于
- == 逻辑相等
- != 逻辑不相等

- ===　实例相等
- !==　实例不相等

在进行关系运算时，如果操作数之间的关系成立，返回值为1；关系不成立，则返回值为0；若某一个操作数的值不定，则关系是模糊的，返回的是不定值X。

实例算子"==="和"!=="可以比较含有X和Z的操作数，在模块的功能仿真中有着广泛的应用。所有的关系运算符有着相同的优先级，但低于算术运算符的优先级。

4. 逻辑运算符

Verilog HDL中有3类逻辑运算符：

- &&　逻辑与
- ||　逻辑或
- !　逻辑非

其中，"&&"和"||"是双目运算符，要求有两个操作数；而"!"是单目运算符，只要求一个操作数。"&&"和"||"的优先级高于算术运算符。逻辑运算符的真值表如表2-2所示。

表2-2　逻辑运算符的真值表

a	b	!a	!b	a&&b	a||b
1	1	0	0	1	1
1	0	0	1	0	1
0	1	1	0	0	1
0	0	1	1	0	0

5. 条件运算符

条件运算符的格式如下：

```
y = x? a: b;
```

条件运算符有3个操作数，若第一个操作数y=x是True，算子返回第二个操作数a，否则返回第三个操作数b。例如：

```
wire y;
assign y = (s1 == 1)? a: b;
```

嵌套的条件运算符可以实现多路选择，例如：

```
wire[1:0]s;
assign s = (a>=2)? 1: (a<0)? 2: 0;
//当a>=2时，s=1；当a<0时，s=2；在其余情况下，s=0。
```

6. 位运算符

作为一种针对数字电路的硬件描述语言，Verilog HDL用位运算来描述电路信号中的与、或以及非操作，总共有7种位逻辑运算符：

- ~　非
- &　与
- |　或

- ^　　异或
- ^～　　同或
- ～&　　与非
- ～|　　或非

位运算符中除了“～”,都是双目运算符。位运算对其自变量的每一位进行操作。例如,s1&s2 的含义就是 s1 和 s2 的对应位相与。如果两个操作数的长度不相等的话,将会对较短的数高位补零,然后进行对应位运算,使输出结果的长度与位宽较长的操作数长度保持一致。例如:

```
s1 = ~s1;
var = ce1 & ce2;
```

7. 移位运算符

移位运算符只有两种:“<<”(左移)和“>>”(右移),左移 1 位相当于乘 2,右移 1 位相当于除 2。其使用格式为:

```
s1<<N; 或 s1>>N;
```

其含义是将第一个操作数 s1 向左(右)移位,所移动的位数由第二个操作数 N 来决定,且都用 0 来填补移出的空位。

在实际运算中,经常通过不同移位数的组合来计算简单的乘法和除法。例如 s1 * 20,因为 20=16+4,所以可以通过 s1<<4+s1<<2 来实现。

8. 拼接运算符

拼接运算符可以将两个或更多个信号的某些位并接起来进行运算操作。其使用格式为:

```
{s1,s2,…,sn}
```

将某些信号的某些位详细地列出来,中间用逗号隔开,最后用一个大括号表示一个整体信号。

在工程实际中,拼接运算受到了广泛使用,特别是在描述移位寄存器时。

例 2-3　给出拼接符的 Verilog 实例。

```
reg [15:0] shiftreg;
always @( posedge clk)
    shiftreg [15:0]<= {shiftreg [14:0],data_in};
```

9. 一元约简运算符

一元约简运算符是单目运算符,其运算规则类似于位运算符中的与、或、非,但其运算过程不同。约简运算符对单个操作数进行运算,最后返回一位数,其运算过程为:首先将操作数的第一位和第二位进行与、或、非运算;然后将运算结果和第三位进行与、或、非运算;依次类推,直至最后一位。

常用的约简运算符的关键字和位操作符关键字一样,仅仅有单目运算和双目运算的区别。

例 2-4 给出一元简约运算符的 Verilog 实例。

```
reg [3:0] s1;
reg s2;
s2 = &s1; //& 即为一元约简运算符"与"
```

2.4 Verilog HDL 语言的描述语句

Verilog HDL 可以完成实际电路不同抽象级别的建模，具体而言，有 3 种描述形式：如果从电路结构的角度来描述电路模块，则称为结构描述形式；如果对线型变量进行操作，就是数据流描述形式；如果只从功能和行为的角度来描述一个实际电路，称为行为级描述形式。如前所述，电路具有 5 种不同模型(系统级、算法级、RTL 级、门级和开关级)。其中，系统级、算法级、RTL 级属于行为描述；门级属于结构描述；开关级涉及模拟电路，在数字电路中一般不考虑。

2.4.1 结构描述形式

通过实例进行描述的方法，将 Verilog HDL 预先定义的基本单元实例嵌入到代码中，监控实例的输入。Verilog HDL 中定义了 26 个有关门级的关键字，比较常用的有 8 个。在实际工程中，简单的逻辑电路由逻辑门和开关组成，通过门原语可以直观地描述其结构。

基本的门类型关键字有 And、nand、nor、or、xor、xnor、buf 和 not。

Verilog HDL 支持的基本逻辑部件是由该基本逻辑器件的原语提供的，其调用格式为：

门类型 <实例名> (输出，输入 1，输入 2，…，输入 N)

例如，

```
nand na01(na_out,a,b,c );
```

表示一个名字为 na01 的与非门，输出为 na_out，输入为 a、b、c。

例 2-5 一个简单的全加器例子。

```
module ADD(A,B,Cin,Sum,Cout);
    input A,B,Cin;
    output Sum,Cout;
    // 声明变量
    wire S1,T1,T2,T3;

    xor X1 (S1,A,B),
        X2 (Sum,S1,Cin);

    and A1 (T3,A,B),
        A2 (T2,B,Cin),
      A3 (T1,A,Cin);

    or O1 (Cout,T1,T2,T3);
endmodule
```

在这一实例中，模块包含门的实例语句，也就是包含内置门 xor、and 和 or 的实例语句。门实例由线网型变量 S1、T1、T2 和 T3 互连。由于未指定顺序，门实例语句可以以任何顺序出现。

门级描述本质上也是一种结构网表。在实际中的使用方式为：先使用门逻辑构成常用的触发器、选择器、加法器等模块，再利用已经设计的模块构成更高一层的模块，依次重复几次，便可以构成一些结构复杂的电路。其缺点是不易管理，难度较大，且需要一定的资源积累。

2.4.2 数据流描述形式

数据流型描述一般都采用 assign 连续赋值语句来实现，主要用于实现组合功能。连续赋值语句右边所有的变量受持续监控，只要这些变量有一个发生变化，整个表达式被重新赋值给左端。这种方法只能用于实现组合逻辑电路，其格式如下：

```
assign L_s = R_s;
```

例 2-6 一个利用数据流描述的移位器。

```
module mlshift2(a,b);
    input a;
    output b;

    assign b = a<<2;
endmodule
```

在上述模块中，只要 a 的值发生变化，b 就会被重新赋值，所赋值为 a 左移两位后的值。

2.4.3 行为描述形式

行为型描述主要包括过程结构、语句块、时序控制、流控制 4 个方面，主要用于时序逻辑功能的实现。

1. 过程结构

过程结构采用下面 4 种过程模块来实现，具有强的通用型和有效性：

- initial 模块；
- always 模块；
- 任务(task)模块；
- 函数(function)模块。

一个程序可以有多个 initial 模块、always 模块、task 模块和 function 模块。initial 模块和 always 模块都是同时并行执行的，区别在于 initial 模块只执行一次，而 always 模块不断重复地运行。另外，task 模块和 function 模块能被多次调用，其具体使用方法可参见 3.3 节。

1) initial 模块

在进行仿真时，一个 initial 模块从模拟 0 时刻开始执行，且在仿真过程中只执行一次。在执行完一次后。该 initial 模块就被挂起，不再执行。如果仿真中有两个 initial 模块，则同

时从 0 时刻开始并行执行。

initial 模块是面向仿真的，是不可综合的，通常被用来描述测试模块的初始化、监视、波形生成等功能。其格式为：

```
initial begin/fork
    块内变量说明
    时序控制 1  行为语句 1;
    …
    时序控制 n  行为语句 n;
end/join
```

其中，begin…end 块定义语句中的语句是串行执行的，而 fork…join 块语句中的语句定义是并行执行的。当块内只有一条语句且不需要定义局部变量时，可以省略 begin…end/fork…join。

例 2-7 给出一个 initial 模块的实例。

```
initial begin
// 初始化输入向量
    clk = 0;
    ar = 0;
    ai = 0;
    br = 0;
    bi = 0;
    // 等待 100ns,全局 reset 信号有效
    #100;
    ar = 20;
    ai = 10;
    br = 10;
    bi = 10;
end
```

2）always 模块

和 initial 模块不同，always 模块是一直重复执行的，并且可被综合。always 过程块由 always 过程语句和语句块组成的，其格式为：

```
always @(敏感事件列表) begin/fork
    块内变量说明
    时序控制 1  行为语句 1;
    …
    时序控制 n  行为语句 n;
end/join
```

其中，begin…end/fork…join 的使用方法和 initial 模块中的一样。敏感事件列表是可选项，但在实际工程中很常用，而且是比较容易出错的地方。敏感事件表的目的就是触发 always 模块的运行，而 initial 模块后面是不允许有敏感事件表的。

敏感事件表由一个或多个事件表达式构成，事件表达式就是模块启动的条件。当存在多个事件表达式时，要使用关键词 or 将多个触发条件结合起来。Verilog HDL 的语法规定：对于这些表达式所代表的多个触发条件，只要有一个成立，就可以启动块内语句的执行。例如，在语句

```
always@ (a or b or c) begin
    ...
end
```

中，always 过程块的多个事件表达式所代表的触发条件是，只要 a、b、c 信号的电平有任意一个发生变化，begin…end 语句就会被触发。

always 模块主要是对硬件功能的行为进行描述，可以实现锁存器和触发器，也可以用来实现组合逻辑。利用 always 模块实现组合逻辑时，要将所有的信号放进敏感列表；实现时序逻辑时，不一定将所有的结果放进敏感信号列表。敏感信号列表未包含所有输入的情况称为不完整事件说明，有时可能会引起综合器的误解，产生许多意想不到的结果。

例 2-8 给出敏感事件未包含所有输入信号的情况。

```
module and3(f,a,b,c);
    input a,b,c;
    output f;
    reg f;

    always @(a or b )begin
        f = a & b & c;
    end
endmodule
```

```
module and3(f,a,b,c);
    input a,b,c;
    output f;
    reg f;

    always @(a or b or c )begin
        f = a & b & c;
    end
endmodule
```

2. 语句块

语句块就是在 initial 模块或 always 模块中位于 begin…end/fork…join 块定义语句之间的一组行为语句。语句块可以有个名字，写在块定义语句的第一个关键字之后，即 begin 或 fork 之后，可以唯一地标识出某一语句块。如果有了块名字，则该语句块被称为一个有名块。在有名块内部可以定义内部寄存器变量，且可以使用“disable”中断语句中断。块名提供了唯一标识寄存器的方法。

例 2-9 语句块使用实例。

```
always @ (a or b )
begin: adder1
    c = a + b;
end
```

该实例定义了一个名为 adder1 的语句块，实现输入数据的相加。

按照界定不同，语句块分为两种。

1）begin…end

用来组合需要顺序执行的语句，称为串行块。例如：

```
parameter d = 50;
reg[7:0] r;
begin                          //由一系列延迟产生的波形
   # d r = 'h35;               //语句 1
   # d r = 'hE2;               //语句 2
   # d r = 'h00;               //语句 3
   # d r = 'hF7;               //语句 4
   # d -> end_wave;            //语句 5，触发事件 end_wave
end
```

串行块的执行特点如下：

- 串行块内的各条语句是按它们在块内的语句逐次逐条顺序执行的，当前一条执行完之后，才能执行下一条。上例中，语句 1 至语句 5 是顺序执行的。
- 块内每一条语句中的延时控制都是相对于前一条语句结束时刻的延时控制。上例中，语句 2 的延时为 2d。
- 在进行仿真时，整个语句块总的执行时间等于所有语句执行时间之和。上例中，语句块中总的执行时间为 5d。

2）fork…join

用来组合需要并行执行的语句，称为并行块。例如：

```
parameter d = 50;
reg[7:0] r;
fork                           //由一系列延迟产生的波形
   # d r = 'h35;               //语句 1
   # 2d r = 'hE2;              //语句 2
   # 3d r = 'h00;              //语句 3
   # 4d r = 'hF7;              //语句 4
   # 5d -> end_wave;           //语句 5，触发事件 end_wave
join
```

并行块的执行特点为：

- 并行语句块内的各条语句是各自独立地同时开始执行的，各条语句的起始执行时间都等于程序流程进入该语句块的时间。上例中，语句 2 并不需要等语句 1 执行完才开始执行，它与语句 1 是同时开始的。
- 块内每一条语句中的延时控制都是相对于程序流程进入该语句块的时间而言的。上例中，语句 2 的延时为 2d。
- 在进行仿真时，整个语句块总的执行时间等于执行时间最长的那条语句所需要的执行时间。上例中，整个语句块的执行时间为 5d。

3）混合使用

在分别介绍了串行块和并行块之后，还需要讨论二者的混合使用。混合使用可以分为下面两种情况。

(1) 串行块和并行块分别属于不同的过程块时，串行块和并行块是并行执行的。例如，一个串行块和一个并行块分别存在于两个 initial 过程块中，由于这两个过程块是并行执行的，所以其中所包含的串行语句和并行语句也是同时并行执行的。在串行块内部，其语句是串行执行的；在并行块内部，其语句是并行执行的。

(2) 当串行块和并行块嵌套在同一过程块中时，内层语句可以看作是外层语句块中的一条普通语句，内层语句块什么时候得到执行，是由外层语句块的规则决定的；而在内层语句块开始执行时，其内部语句怎么执行，要遵守内层语句块的规则。

3. 时序控制

Verilog HDL 提供了两种类型的显示时序控制。一种是延迟控制，在这种类型的时序控制中，通过表达式定义开始遇到这一语句和真正执行这一语句之间的延迟时间。另外一种是事件控制，这种时序控制是通过表达式来完成的，只有当某一事件发生时，才允许语句继续向下执行。

1) 延时控制

延时控制的语法如下：

```
# 延时数 表达式;
```

延时控制表示在语句执行前的“等待延时”，下面给出一个例子：

```
initial
begin
   #5 clk = ~clk;
end
```

延时控制只能在仿真中使用，是不可综合的。在综合时，所有的延时控制都会被忽略。

2) 事件控制

事件控制分为两种：边沿触发事件控制和电平触发事件控制。

(1) 边沿触发事件是指指定信号的边沿信号跳变时发生指定的行为，分为信号的上升沿和下降沿控制。上升沿用 posedge 关键字来描述，下降沿用 negedge 关键字描述。边沿触发事件控制的语法格式为：

- 第一种：@(<边沿触发事件>) 行为语句；
- 第二种：@(<边沿触发事件 1> or <边沿触发事件 2> or…or <边沿触发事件 n>) 行为语句；

例 2-10 边沿触发事件计数器。

```
reg [4:0] cnt;
always @(posedge clk) begin
   if (reset)
      cnt<=0;
   else
      cnt<=cnt+1;
end
```

上面这个例子表明：只要 clk 信号有上升沿，那么 cnt 信号就会加 1，完成计数的功能。这种边沿计数器在同步分频电路中有着广泛的应用。

(2) 电平敏感事件是指指定信号的电平发生变化时发生指定的行为。下面是电平触发事件控制的语法和实例。

- 第一种：@(<电平触发事件>) 行为语句；
- 第二种：@(<电平触发事件 1> or <电平触发事件 2> or…or <电平触发事件 n>) 行为语句；

例 2-11 电平沿触发计数器。

```
reg [4:0] cnt;
always @(a or b or c) begin
    if (reset)
      cnt<=0;
    else
      cnt<=cnt+1;
end
```

其中，只要 a、b、c 信号的电平有变化，信号 cnt 的值就会加 1，这可以用于记录 a、b、c 变化的次数。

4. 流控制

流控制语句包括 3 类，即跳转、分支和循环语句。

1) if 语句

if 语句的语法如下：

```
if (条件 1)
    语句块 1
else if (条件 2)
    语句块 2
…
else
    语句块 n
```

如果条件 1 的表达式为真(或非 0 值)，那么语句块 1 被执行，否则语句块不被执行；然后依次判断条件 2 至条件 n 是否满足，如果满足就执行相应的语句块；最后跳出 if 语句，整个模块结束。如果所有的条件都不满足，则执行最后一个 else 分支。在应用中，else if 分支的语句数目由实际情况决定；else 分支也可以默认，但会产生一些不可预料的结果，生成本不期望的锁存器。

例 2-12 给出一个 if 语句的例子，并说明省略 else 分支所产生的一些结果。

```
always @(a1 or b1)
begin
    if (a1) q<=d;
end
```

if 语句只能保证当 a1=1 时，q 才取 d 的值，但程序没有给出 a1=0 时的结果。因此在缺少 else 语句的情况下，即使 a1=0 时，q 的值会保持 a1=1 的原值，这就综合成了一个锁存器。

如果希望 a1=0 时，q 的值为 0 或者其他值，那么 else 分支是必不可少的。下面给出 a1=0，q=0 的设计：

```
always @(a1 or b1)
begin
    if (a1) q<=d;
    else q<=0;
end
```

2）case 语句

case 语句是一个多路条件分支形式，其用法和 C 语言的 case 语句是一样的。

下面给出一个 case 语句的例子：

```
reg [2:0] cnt;
case (cnt)
    3'b000: q = q + 1;
    3'b001: q = q + 2;
    3'b010: q = q + 3;
    3'b011: q = q + 4;
    3'b100: q = q + 5;
    3'b101: q = q + 6;
    3'b110: q = q + 7;
    3'b111: q = q + 8;
    default: q <= q + 1;
endcase
```

需要指出的是，case 语句的 default 分支虽然可以默认，但是一般不要默认，否则会和 if 语句中缺少 else 分支一样，生成锁存器。

例 2-13 给出 case 语句的 Verilog 实例。

```
always @(a1[1:0] or b1)
begin
    case (a1)
    2'b00: q <= b'1;
    2'b01: q <= b'1 + 1;
end
```

这会生成锁存器。一般地，为了使 case 语句可控，都需要加上 default 选项，则

```
always @(a1[1:0] or b1)
begin
    case (a1)
    2'b00: q <= b1;
    2'b01: q <= b1 + 1;
    default: q <= b1 + 2;
end
```

在实际开发中，要避免生成锁存器的错误。如果用 if 语句，最好写上 else 选项；如果用 case 语句，最好写上 default 选项。遵循上面两条原则，就可以避免发生错误，使设计者更加明确设计目标，同时增加了 Verilog 程序的可读性。

此外，还需要解释在硬件语言中使用 if 语句和 case 语句的区别。在实际中，如果有分支情况，尽量选择 case 语句。这是因为 case 语句的分支是并行执行的，各个分支没有优先级的区别。而 if 语句的选择分支是串行执行的，是按照书写的顺序逐次判断的。如果设计者没有这种优先级的考虑，if 语句和 case 语句相比，需要占用额外的硬件资源。

3）循环语句

Verilog HDL 中提供了 4 种循环语句：for 循环、while 循环、forever 循环和 repeat 循环。其语法和用途与 C 语言很类似。

(1) for 循环按照指定的次数重复执行过程赋值语句。for 循环的语法为：

```
for(表达式 1；表达式 2；表达式 3) 语句
```

for 循环语句最简单的应用形式是很容易理解的，其形式为：

```
for(循环变量赋初值；循环结束条件；循环变量增值)
```

例如：

```
for(bindex = 1; bindex< = size; bindex = bindex + 1)
    result = result + (a<<(bindex - 1));
```

(2) while 循环执行过程赋值语句，直到指定的条件为假。如果表达式条件在开始时不为真(包括假、x 以及 z)，那么过程语句将永远不会被执行。while 循环的语法为：

```
while (表达式) begin
    …
end
```

例如：

```
while (temp) begin
  count = count + 1;
end
```

(3) forever 循环语句连续执行过程语句。为跳出这样的循环，中止语句可以与过程语句共同使用。同时，在过程语句中必须使用某种形式的时序控制，否则 forever 循环将永远执行下去。forever 语句必须写在 initial 模块中，用于产生周期性波形。forever 循环的语法为：

```
forever begin
    …
end
```

例如：

```
initial
forever begin
    if(d) a = b + c;
    else a = 0;
end
```

(4) repeat 循环语句执行指定循环数，如果循环计数表达式的值不确定，即为 x 或 z 时，那么循环次数按 0 处理。repeat 循环语句的语法为：

```
repeat(表达式) begin
    …
end
```

例如：

```
repeat (size) begin
  c = b<<1;
end
```

2.4.4 混合设计模式

在模型中，结构描述、数据流描述和行为描述可以自由混合。也就是说，模块描述中可以包括实例化的门、模块实例化语句、连续赋值语句以及行为描述语句的混合，它们之间可以相互包含。使用 always 语句和 initial 语句（切记只有寄存器类型的数据才可以在模块中赋值）来驱动门和开关，而来自于门或连续赋值语句（只能驱动线网型）的输出能够反过来用于触发 always 语句和 initial 语句。

下面给出一个混合设计方式的实例。

例 2-14 用结构和行为实体描述一个 4 位全加器。

```
module adder4(in1,in2,sum,flag);
    input [3:0] in1;
    input [3:0] in2;
    output [4:0] sum;
    output flag;

    wire c0,c1,c2;

    fulladd u1 (in1 [0],in2 [0],0,sum[0],c0);
    fulladd u2 (in1 [1],in2 [1],c0,sum[1],c1);
    fulladd u3 (in1 [2],in2 [2],c1,sum[2],c2);
    fulladd u4 (in1 [3],in2 [3],c2,sum[3],sum[4]);

    assign flag = sum? 0: 1;
endmodule
```

在这个例子中，用结构化模块计数 sum 输出，用行为级模块输出标志位。

2.5 Verilog 代码书写规范

代码书写规范覆盖面很广，还涉及很多细节问题，这些都需要在实际编写过程中加以考虑。虽然代码规范不是绝对的，需要用户灵活处理，但是在一个项目组内部、一个项目的进程中，应该有一套完备的代码书写规范来作为约束，使代码整洁且具备良好的可读性。下面给出部分代码书写规范的基本要点。

2.5.1 信号命名规则

信号命名规则在团队开发中占据着重要地位，统一、有序的命名能大幅减少设计人员之间的冗余工作，还可便于团队成员对所写代码的查错和验证。比较著名的信号命名规则当推 Microsoft 公司的“匈牙利”法。该命名规则的主要思想是“在变量和函数名中加入前缀，以增进人们对程序的理解”。例如，所有的字符变量均以 ch 为前缀；若是常数变量，则追加前缀 c。信号命名的整体要求为：命名字符具有一定的意义，直白易懂，且项目命名规则唯

一。对于 HDL 设计，设计人员还需要注意以下命名规则。

1. 系统级信号的命名

系统级信号指复位信号、置位信号、时钟信号等需要输送到各个模块的全局信号。系统信号以字符串 sys 或 syn 开头；时钟信号以 clk 开头，并在后面添加相应的频率值；复位信号一般以 rst 或 reset 开头；置位信号以 st 或 set 开头。典型的信号命名方式如下所示：

```
wire [7:0] sys_dout,sys_din;
wire clk_32p768MHz;
wire reset;
wire st_counter;
```

2. 低电平有效的信号命名

低电平有效的信号后一律加下划线和字母 n。例如：

```
wire SysRst_n;
wire FifoFull_n;
```

3. 经过锁存器锁存后的信号

经过锁存器锁存后的信号后加下划线和字母 r，与锁存前的信号相区别。例如，CpuRamRd 信号经锁存后应命名为 CpuRamRd_r。

低电平有效的信号经过锁存器锁存后，其命名应在_n 后加 r。例如，CpuRamRd_n 信号经锁存后应命名为 CpuRamRd_nr。

对于多级锁存的信号，可多加 r 以标明。例如，CpuRamRd 信号经两级触发器锁存后，应命名为 CpuRamRd_rr。

2.5.2 模块命名规则

HDL 语言的模块类似于 C 语言中的函数，可采用 C 语言函数的大多数规则。模块的命名应该尽量用英文表达出其完成的功能。遵循动宾结构的命名法则，函数名中动词在前，并在命名前加入函数的前缀，函数名的长度一般不少于 2 个字母。HDL 模块的命名还需要考虑以下情况：

1. 模块的命名规则

在系统设计阶段，应该为每个模块命名。命名的方法是将模块英文名称的各个单词的首字母组合起来，形成 3～5 个字符的缩写。若模块的英文名只有一个单词，可取该单词的前 3 个字母。各模块的命名以 3 个字母为宜。例如，Arithmatic Logical Unit 模块命名为 ALU，Data Memory Interface 模块命名为 DMI，Decoder 模块命名为 DEC。

2. 模块之间接口信号的命名

所有的变量命名分为两个部分：第一部分表明数据方向，其中数据发出方在前，数据接收方在后；第二部分为数据名称。两个部分之间用下划线隔离开。第一部分全部大写；第二部分中所有具有明确意义的英文名全部拼写或缩写的第一个字母大写，其余部分小写。例如：

```
wire CPUMMU_WrReq;
```

下划线左边是第一部分，代表数据方向是从 CPU 模块发向存储器管理单元模块(MMU)。下划线右边的 Wr 为 Write 的缩写，Req 是 Request 的缩写，两个缩写的第一个字母都大写，便于理解。整个变量连起来的意思就是："CPU 发送给 MMU 的写请求信号"。模块上、下层次间信号的命名也遵循本规定。若某个信号从一个模块传递到多个模块，其命名应视信号的主要路径而定。

3. 模块内部信号

模块内部的信号由几个单词连接而成，缩写要求能基本表明本单词的含义；单词除常用的缩写方法外(如 Clock→Clk、Write→Wr、Read→Rd 等)，一律取该单词的前几个字母(如 Frequency→Freq、Variable→Var 等)；每个缩写单词的第一个字母大写；若遇两个大写字母相邻，中间添加一个下划线(如 DivN_Cntr)。例如：

```
SdramWrEn_n;
FlashAddrLatchEn;
```

2.5.3 代码格式规范

1. 分节书写格式

各节之间加 1 行到多行空格。如每个 always、initial 语句都是一节，每节基本上完成一个特定的功能，即用于描述某几个信号的产生。在每节之前有几行注释对该节代码加以描述，至少列出本节中所描述信号的含义。

行首不要使用空格来对齐，而是用 Tab 键，Tab 键的宽度设为 4 个字符宽度。行尾不要有多余的空格。

2. 注释的规范

使用"//"开始的注释以行为单位；使用"/* */"的注释，"/*"和"*/"各占用一行，并且顶头。例如：

```
// Edge detector used to synchronize the input signal;
```

对于函数，应该从"功能"、"参数"、"返回值"、"主要思路"、"调用方法"、"日期"6 个方面用如下格式注释：

```
// 程序说明开始
// ================================================================== //
// 功能：完成两个输入数的相加
// 参数：strByDelete,strToDelete
// 输入参数
// 输出参数
// 主要思路：本算法主要采用 2 级流水线完成相加
// 日期：起始日期，如 2007/8/21.9：40—2007/8/23.21：45
// 版本：
// 程序编写人员：
// 程序调试记录：
```

```
// ================================================================== //
// 模块说明结束
```

此外，在注释说明中，需要注意以下细节：

- 在注释中，应该详细说明模块的主要实现思路，特别要注明自己的一些想法。如果有必要，应该写明想法产生的来由。
- 在注释中详细注明函数的使用方法、对于输入参数的要求以及输出数据的格式。
- 在注释中要强调调用时的危险方面、可能出错的地方。
- 对日期的注释要求记录从开始编写模块到模块测试结束之间的日期。
- 对模块注释开始到模块命名之间应该有一组用来标识的特殊字符串。如果算法比较复杂，或算法中的变量定义与位置有关，则要求对变量的定义进行图解。对难以理解的算法，能图解尽量图解。

3. 空格的使用

不同变量，以及变量与符号、变量与括号之间都应当保留一个空格。Verilog 关键字与其他任何字符串之间都应当保留一个空格。例如：

```
always @ (…)
```

使用大括号和小括号时，前括号的后边和后括号的前边应当留有一个空格。逻辑运算符、算术运算符、比较运算符等运算符的两侧各留一个空格，与变量分隔开来；单操作数运算符例外，直接位于操作数前，不使用空格。使用“//”进行注释时，在“//”后应当有一个空格；注释行的末尾不要有多余的空格。例如：

```
assign SramAddrBus = { AddrBus[31:24],AddrBus[7:0] };
assign DivCntr[3:0] = DivCntr[3:0] + 4'b0001;
assign Result = ~Operand;
```

4. begin…end 的书写规范

同一个层次的所有语句左端对齐；initial、always 等语句块的 begin 关键词跟在本行的末尾，相应的 end 关键词与 initial、always 对齐，并且在 end 后面添加注释标明结束。这样做的好处是避免因 begin 独占一行而造成行数太多。例如：

```
always@(posedge SysClk or negedge SysRst) begin
    if(!SysRst)DataOut<=4'b0000;
    else if(LdEn)begin
        DataOut<=DataIn;
        End
else
        DataOut<=DataOut+4'b0001;
end   //end always 模块
```

不同层次之间的语句使用 Tab 键进行缩进，每加深一层，缩进一个 Tab；在 endmodule、endtask、endcase 等标记一个代码块结束的关键词后面要加上一行注释，说明这个代码块的名称。

2.5.4 模块调用规范

在 Verilog 中有两种模块调用的方法。一种是位置映射法，指严格按照模块定义的端口顺序来连接，不用注明原模块定义时规定的端口名，其语法为：

```
模块名(连接端口 1 信号名,连接端口 2 信号名,连接端口 3 信号名,…);
```

另一种为信号映射法，即利用“.”符号，表明原模块定义时的端口名，其语法为：

```
模块名 (.端口 1 信号名(连接端口 1 信号名),
        .端口 2 信号名(连接端口 2 信号名),
        .端口 3 信号名(连接端口 3 信号名),…);
```

显然，信号映射法同时将信号名和被引用端口名列出来，不必严格遵守端口顺序，这不仅降低了代码易错性，还提高了程序的可读性和可移植性。因此，在良好的代码中，严禁使用位置调用法，全部采用信号映射法。

2.6 Verilog 常用程序示例

Verilog 程序已经流行了数十年，积累了大量的经典设计。这些基本模块和优秀设计并不会随着芯片制造工艺的进步而消失，它们是开发人员的宝贵资源。本节将介绍一些基本模块的 Verilog 程序。

2.6.1 Verilog 基本模块

1. 触发器的 Verilog 实现

时序电路是高速电路的主要应用类型，其特点是在任意时刻，电路产生的稳定输出不仅与当前的输入有关，还与电路过去时刻的输入有关。时序电路的基本单元就是触发器。下面介绍几种常见同步触发器的 Verilog 实现。

1）同步 RS 触发器

RS 触发器分为同步触发器和异步触发器，二者的区别在于同步触发器有一个时钟端 clk，只有在时钟端的信号上升(正触发)或下降(负触发)时，触发器的输出才会发生变化。下面以正触发为例，给出其 Verilog 代码实现。

例 2-15 正触发型同步 RS 触发器的 Verilog 实现。

```
module sy_rs_ff (clk,r,s,q,qb);
    input clk,r,s;
    output q,qb;
    reg q;

    assign qb = ~q;
    always @(posedge clk) begin
      case({r,s})
```

```
            2'b00: q<=q;
            2'b01: q<=1;
            2'b10: q<=0;
            2'b11: q<=1'bx;
        endcase
    end
endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-2 所示。

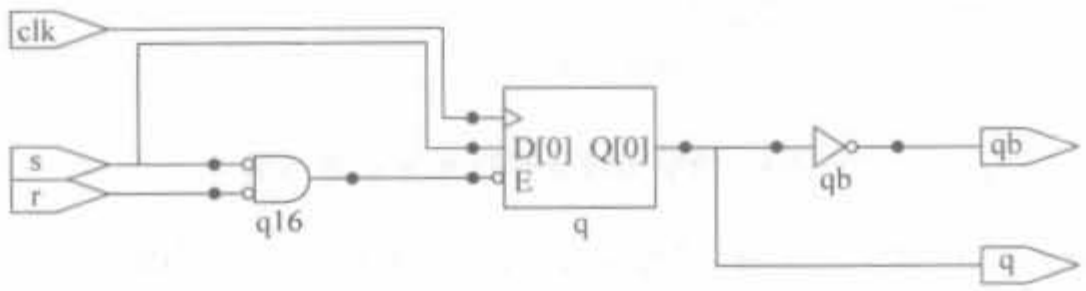


图 2-2 同步 RS 触发器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 2-3 所示。从中可以看出，当输入 r=0、s=0 时，输出 q 保持不变；当 r=0、s=1 时，输出 q 为 1；当 r=1、s=0 时，输出 q 为 0；当 r=1、s=1 时，输出为禁止态。

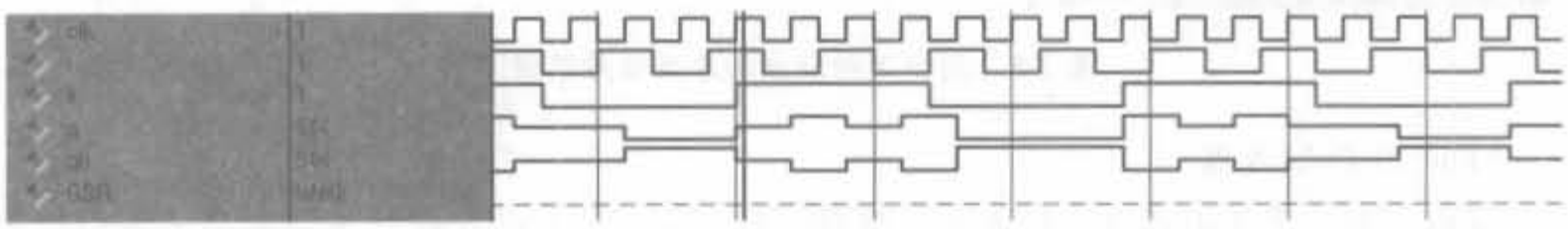


图 2-3 同步 RS 触发器的仿真结果示意图

2）同步 T 触发器

T 触发器也分为同步触发器和异步触发器，二者的区别在于同步 T 触发器多了一个时钟端。同步 T 触发器的逻辑功能为：当时钟 clk 沿到来时，如果 T=1，则触发器状态保持不变；否则，触发器输出端反转。R 为复位端，当其为高电平时，输出 Q 与时钟无关，Q=0。

例 2-16 同步 T 触发器的 Verilog 实现。

```
module sy_t_ff(clk,r,q,qb);
    input clk,r;
    output q,qb;
    reg q;

    assign qb = ~q;
    always @(posedge clk) begin
        if(r)
          q<=0;
        else
        if(t)
          q<=q;
        else
          q<=~q;
```

```
    end
endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-4 所示。

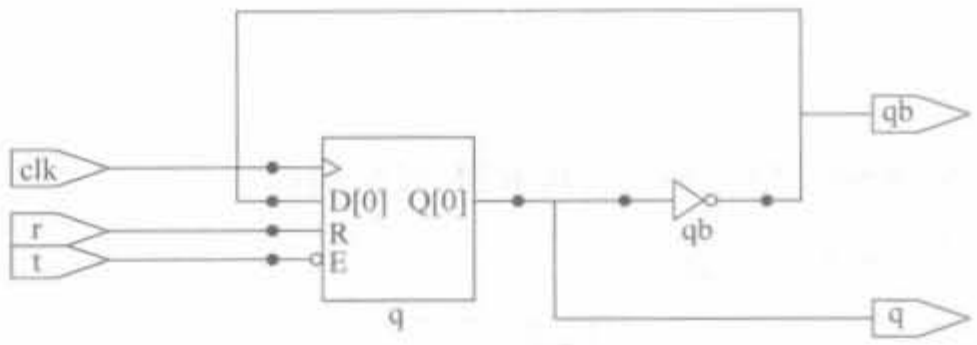


图 2-4　同步 T 触发器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 2-5 所示。从中可以看出，当 r=1 时，输出 q 为 0；当 r=0、t=1 时，输出 q 保持不变；当 r=0、t=0 时，输出 q 反转。

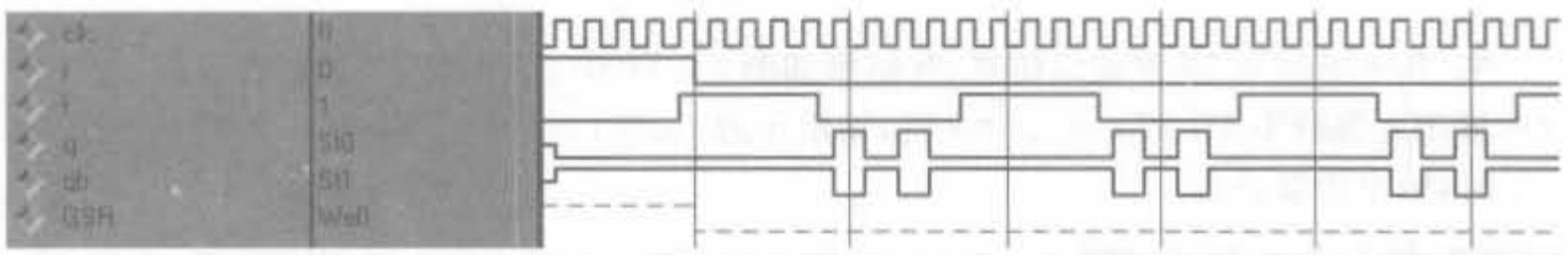


图 2-5　同步 T 触发器的仿真结果示意图

3）同步 D 触发器

同步 D 触发器的功能为：D 输入只能在时序信号 clk 的沿变化时才能被写入到存储器中，替换以前的值。它常用于数据延迟以及数据存储模块中。由于 D 触发器只有一个输入端，在许多情况下，可使触发器之间的连接变得非常简单，因此应用十分广泛。

例 2-17　同步 D 触发器的 Verilog 实现。

```
module sy_d_ff(clk,d,q,qb);
    input clk,d;
    output q,qb;
    reg q;

    assign qb = ~q;
    always@(posedge clk)begin
        q<=d;
    end
endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-6 所示。

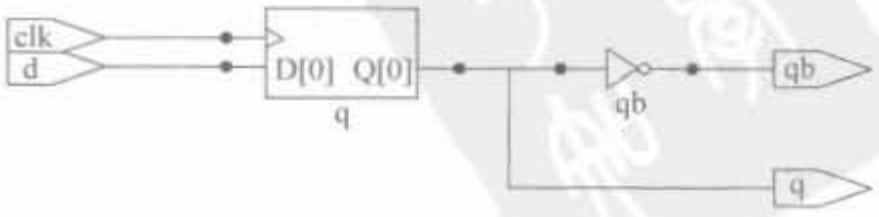


图 2-6　同步 D 触发器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 2-7 所示。从中可以看出，在时钟上升沿，D 触发器都将输入数据接收并寄存。

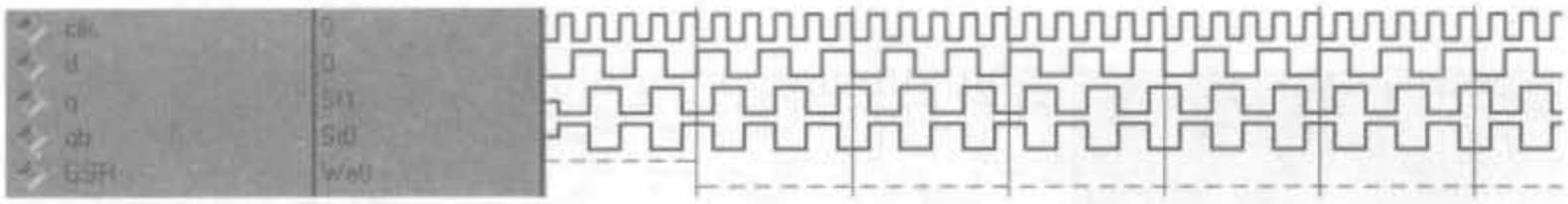


图 2-7 同步 D 触发器的仿真结果示意图

4）同步 JK 触发器

JK 触发器是在 RS 触发器的基础上发展而来的，常用于实现计数器。当 clk=0 时，触发器不工作，处于保持状态。当时钟 clk=1 时，触发器的功能如下：当 JK 为 00、01 以及 10 时，实现 RS 触发器的功能；当 JK 为 11 时，实现 T 触发器的功能。JK 触发器应用比较广泛，对输入值不加限制，可任意取值。

例 2-18 同步 JK 触发器的 Verilog 实现。

```
module sy_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<=0;
            2'b10: q<=1;
            2'b11: q<=~q;
        endcase
    end
endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-8 所示。从 RTL 级结构图上可以看出，JK 触发器是由 RS 触发器加上两条反馈线而构成的，即从“qb”端反馈到原输入信号“r”端，并把“s”输入端改为“j”，“r”改为“k”。

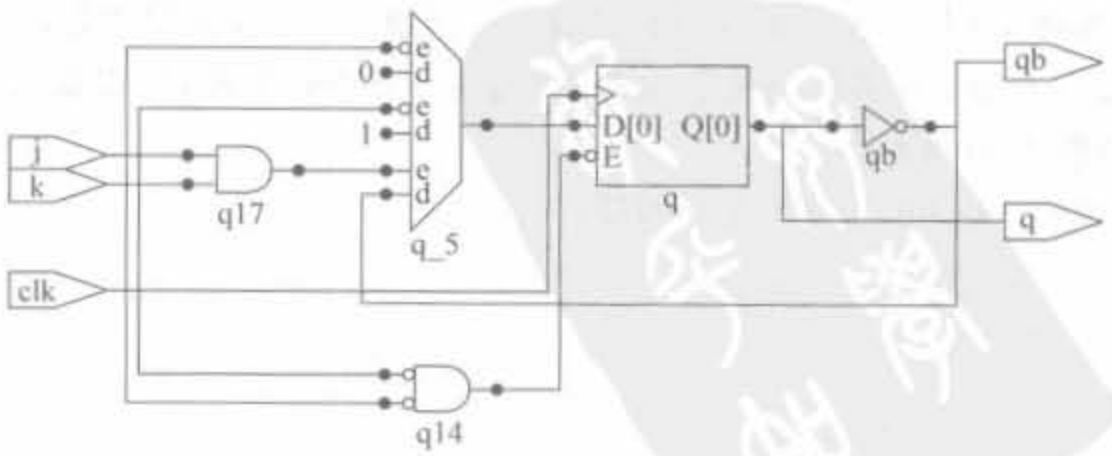


图 2-8 同步 JK 触发器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 2-9 所示。当输入 j=0、k=0 时，输出 q 保持不变；当 j=0、k=1 时，q 为 0；当 j=10、k=0 时，q 为 1；当输入 j=10、k=1 时，输出 q 反转。

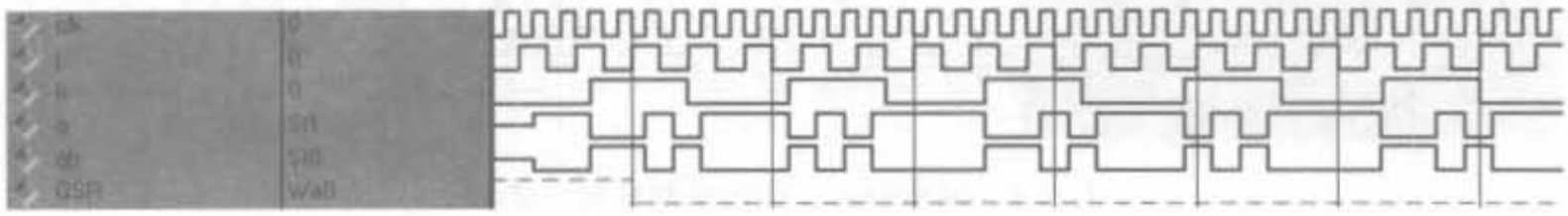


图 2-9　同步 JK 触发器的仿真结果示意图

2. 三态缓冲器的 Verilog 实现

三态缓冲器也称三态门，其典型应用是双向端口，常用于双向数据总线的构建。在数字电路中，逻辑输出有两个正常态：低电平状态（对应逻辑 0）和高电平状态（对应逻辑 1）。此外，电路还有不属于 0 和 1 状态的高组态（对应于逻辑 Z）。高阻即输出端处于浮空状态，只有很小的漏电流流动，其电平随外部电平的高低而定，门电平放弃对输出电路的控制；或者可以理解为，输出与电路是断开的。最基本的三态缓冲器的逻辑符号如图 2-10 所示。

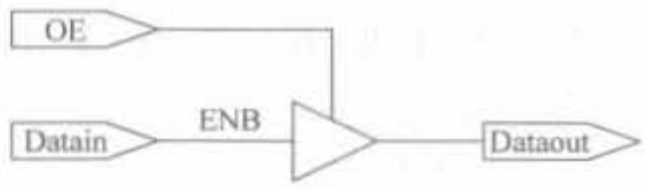


图 2-10　三态缓冲器的逻辑符号图

当 OE 为高电平时，Dataout 与 Datain 相连；OE 为低时，Dataout 为高阻态，相当于和 Datain 之间的连线断开。

例 2-19　使用 Verilog 实现三态缓冲器。

```
inout a;
wire z,b;
//当控制信号 z 为 1 时，开通三态门，b 为输入端口；当 z 为 0 时，三态门为高阻，
//a 为输出端口
assign a = (z)? b: 8'bz;
```

3. 3-8 译码器的 Verilog 实现

3-8 译码器是通过 3 条线来达到控制 8 条线的状态，就是通过 3 条控制线不同的高低电平组合，一共可以组合出 $2^3=8$ 种状态。它在电路中主要起到扩展 I/O 资源的作用。当然，可根据实际需求将 3-8 译码器扩展到更高级数上。传统的 3-8 译码器都是以 74 系列的小集成电路块实现的，占用了过多的电路板面积，而通过 Verilog 实现，只需要极少的逻辑资源就可以实现，如例 2-20 所示。

例 2-20　使用 Verilog 实现 3-8 译码器。

```
module decoder3to8(din,reset,dout);
 input [2:0]  din;
 input        reset;
 output [7:0] dout;

 reg  [7:0]   dout;
 always@(din or reset) begin
```

```
    if(!reset)
    dout = 8'b0000_0000;
else
    case(din)
          3'b000: dout = 8'b0000_0001;
            3'b001: dout = 8'b0000_0010;
            3'b010: dout = 8'b0000_0100;
            3'b011: dout = 8'b0000_1000;
            3'b100: dout = 8'b0001_0000;
            3'b101: dout = 8'b0010_0000;
            3'b110: dout = 8'b0100_0000;
            3'b111: dout = 8'b1000_0000;
    endcase
end

endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-11 所示。

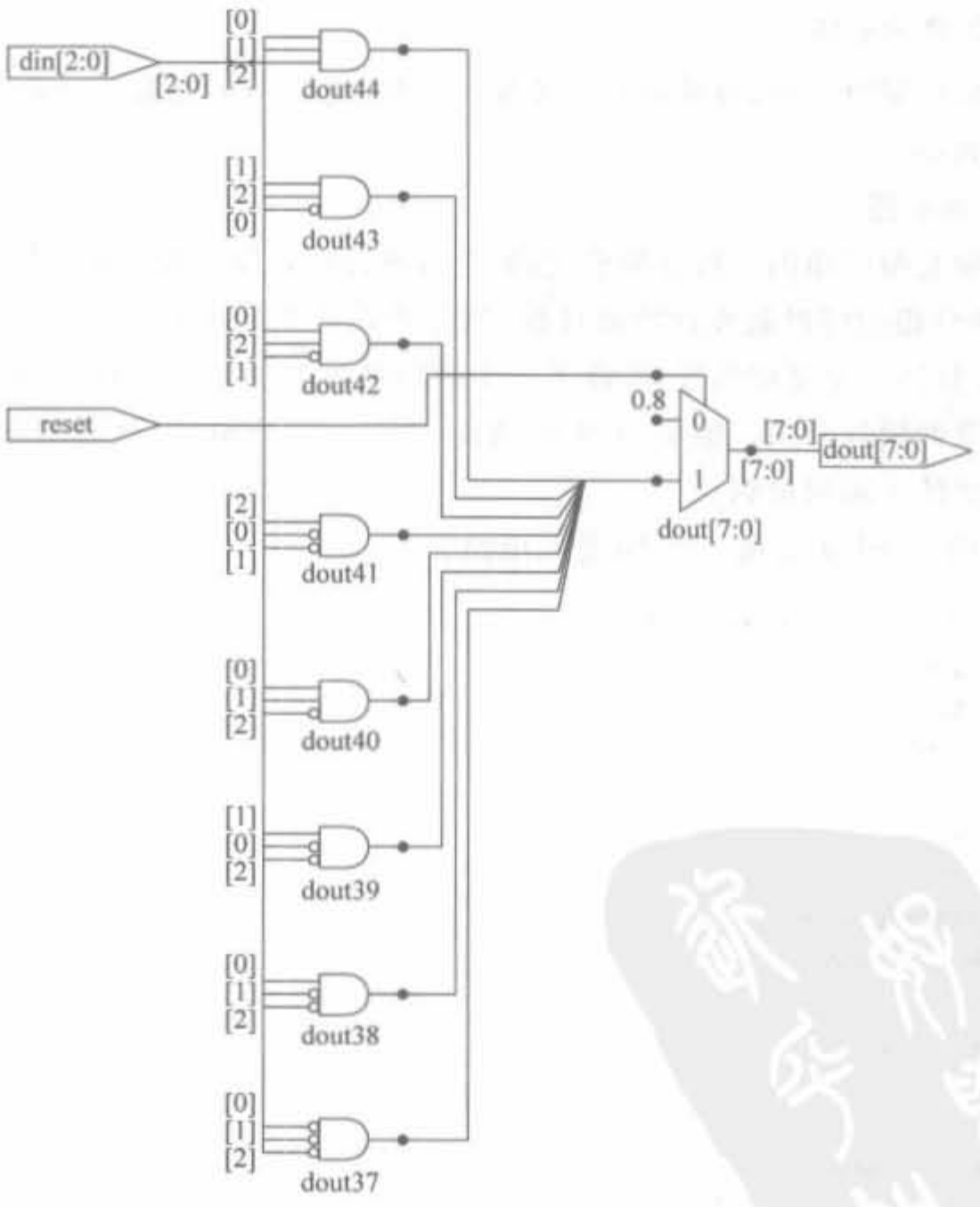


图 2-11 3-8 译码器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 2-12 所示。从图中可以看出，当 reset 信号变高后，依次输入 1～7 以及 0 这 8 个数，8bit 输出信号对应下标的信号值为高，其余值为 0，正确实现了 3-8 译码器的功能。

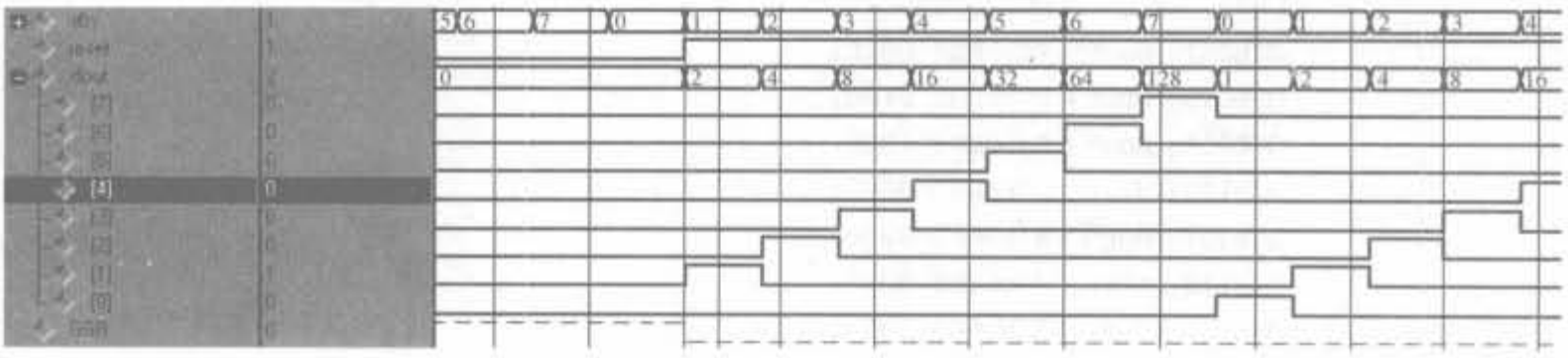


图 2-12　3-8 译码器的仿真结果示意图

2.6.2　基本时序处理模块

1. 奇、偶数分频电路

在数字逻辑电路设计中，分频器是一种基本电路，通常用来对某个给定频率进行分频，以得到所需的频率。

1）偶数分频电路

偶数倍分频是最简单的一种分频模式，完全可通过计数器计数实现。如要进行 N 倍偶数分频，可由待分频的时钟触发计数器计数，当计数器从 0 计数到 $N/2-1$ 时，输出时钟进行翻转，并给计数器一个复位信号，使得下一个时钟从零开始计数，以此循环下去。这种方法可以实现任意的偶数分频。例 2-21 给出的是一个 16 分频电路，其他倍数的分频电路可通过修改计数器的上限值得到。

例 2-21　用 Verilog 实现一个 16 分频电路。

```
module clk_div16(clk_in,reset,clk_out);
 input clk_in;
 input reset;
 output clk_out;

 reg clk_out;
 reg [2:0] cnt;
 always @(posedge clk_in) begin
    if(!reset) begin
   cnt<=0;
    clk_out<=0;
 end
 else
    if(cnt==7) begin
         cnt<=0;
          clk_out<=!clk_out;
    end
    else begin
```

```
        cnt<=cnt+1;
          clk_out<=clk_out;
      end
  end

endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-13 所示。

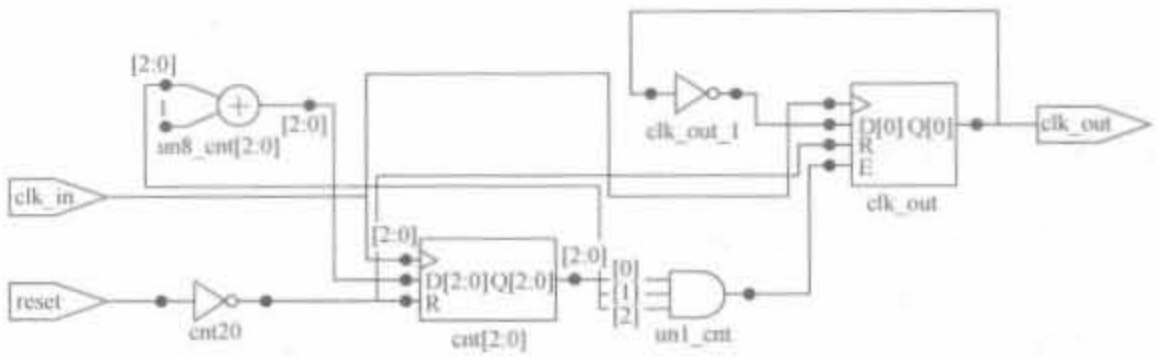


图 2-13　16 分频电路的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 2-14 所示。从图中可以看出，例 2-21 成功实现了输入时钟的 16 分频。

图 2-14　16 分频电路的仿真结果示意图

2）奇数分频电路

奇数倍分频有多种实现方法，下面介绍常用的错位“异或”法的原理。如要进行 3 分频，通过待分频时钟上升沿触发计数器进行模 3 计数，当计数器计数到邻近值时进行两次翻转。比如在计数器计数到 1 时，输出时钟进行翻转；计数到 2 时，再次翻转，即在邻近的 1 和 2 时刻进行两次翻转。这样实现的 3 分频占空比为 1/3 或者 2/3。如果要实现占空比为 50% 的 3 分频时钟，可以通过待分频时钟下降沿触发计数，和上升沿同样的方法计数进行 3 分频，然后将下降沿产生的 3 分频时钟和上升沿产生的时钟进行相或运算，即可得到占空比为 50%的 3 分频时钟。

这种错位“异或”法可以推广实现任意的奇数分频：对于实现占空比为 50%的 N 倍奇数分频，首先进行上升沿触发的模 N 计数，计数到某一选定值时进行输出时钟翻转，然后经过$(N-1)/2$ 再次翻转，得到一个占空比非 50%的奇数 N 分频时钟。再者，同时进行下降沿触发的模 N 计数，到和上升沿触发输出时钟翻转选定值相同值时，进行输出时钟翻转，同样经过$(N-1)/2$，输出时钟再次翻转，生成占空比非 50%的奇数 N 分频时钟。两个占空比非 50%的 N 分频时钟相或运算，得到占空比为 50%的奇数 N 分频时钟。

例 2-22　使用 Verilog 实现 3 分频电路。

```
module clk_div3(clk_in,reset,clk_out);
 input clk_in;
```

```
input reset;
output clk_out;

reg [1:0] cnt,cnt1;
reg       clk_1to3p,clk_1to3n;
always @(posedge clk_in) begin
   if(!reset) begin
         cnt<=0;
          clk_1to3p<=0;
   end
   else begin
       if(cnt == 2'b10) begin
          cnt<=0;
           clk_1to3p<=clk_1to3p;
       end
       else begin
          cnt<=cnt+1;
          clk_1to3p<=!clk_1to3p;
       end
   end
end

always @(negedge clk_in) begin
   if(!reset) begin
       cnt1<=0;
        clk_1to3n<=0;
   end
   else begin
       if(cnt1 == 2'b10) begin
        cnt1<=0;
         clk_1to3n<=clk_1to3n;
       end
       else begin
          cnt1<=cnt1+1;
       clk_1to3n<=!clk_1to3n;
      end
   end
end

assign clk_out=clk_1to3p|clk_1to3n;

endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-15 所示。

在 ModelSim 6.2b 中完成仿真，其结果如图 2-16 所示。可以看到，输出时钟为占空比为 50%的 3 分频时钟。

2. 同步采样模块

在实际应用中，外部输入的异步信号需要经过系统时钟的同步化，且将输入的异步信号整形成一个时钟长的脉冲信号，如图 2-17 所示。这里以例 2-23 来说明实现的方法。

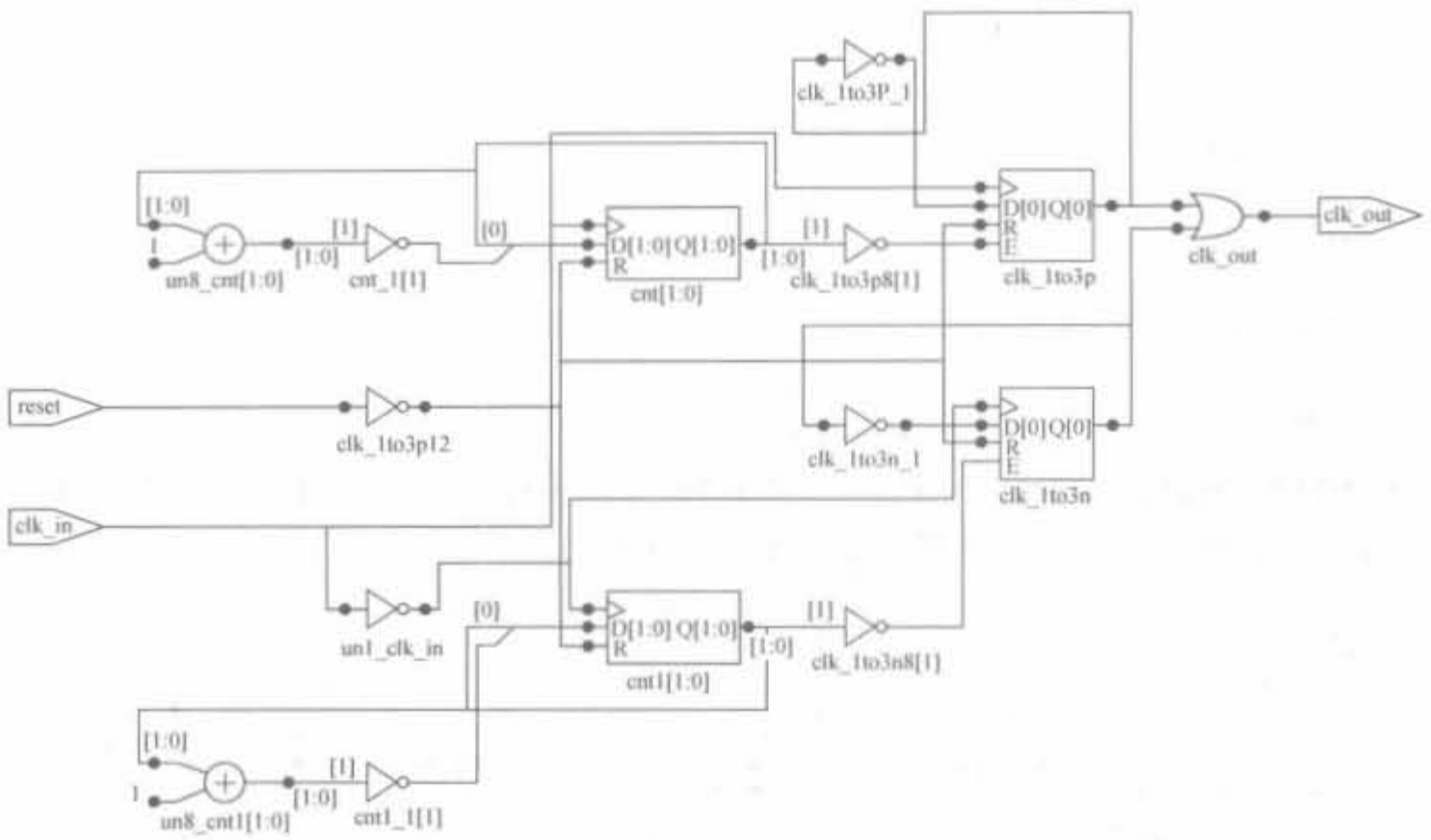


图 2-15 3 分频电路的 RTL 级结构图

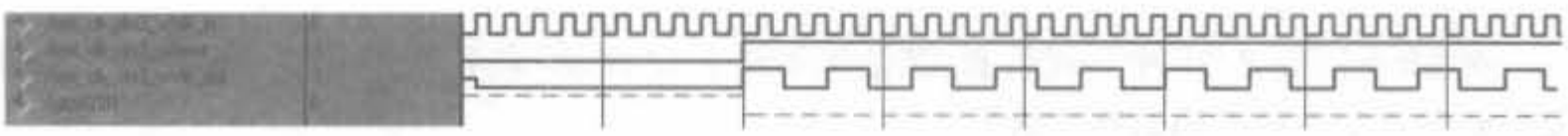


图 2-16 3 分频电路的仿真结果

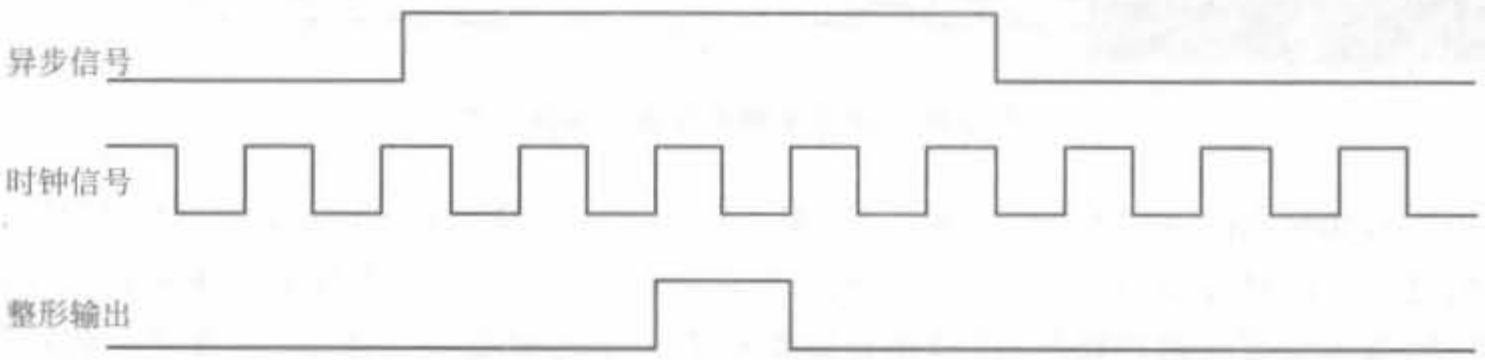


图 2-17 异步信号的同步采样示意图

例 2-23 使用 Verilog 将外部异步信号进行同步整形。

```
module clk_syn(clk,reset,s_in,s_out);
  input clk;
  input reset;
  input s_in;
  output s_out;

  reg s_t1,s_t2;
  always @(posedge clk) begin
     if(!reset) begin
          s_t1<=0;
           s_t2<=0;
```

```
        end
        else begin
            s_t1<=s_in;
            s_t2<=s_t1;
        end
    end

    assign s_out=s_t1 & (!s_t2);

endmodule
```

上述程序经过综合 Synplify Pro 后，其 RTL 级结构如图 2-18 所示。从结果上看，该电路非常简单，但需要一定的时序逻辑能力才能真正理解该段程序。

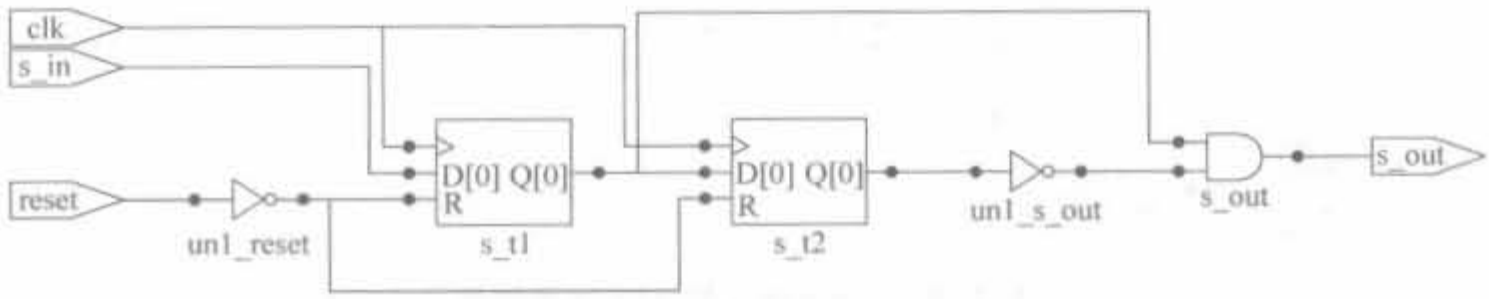


图 2-18　同步电路的 RTL 级结构示意图

在 ModelSim 6.2b 中完成仿真，其结果如图 2-19 所示，即将不等长的高电平信号整形成宽度为一个时钟周期的脉冲信号。

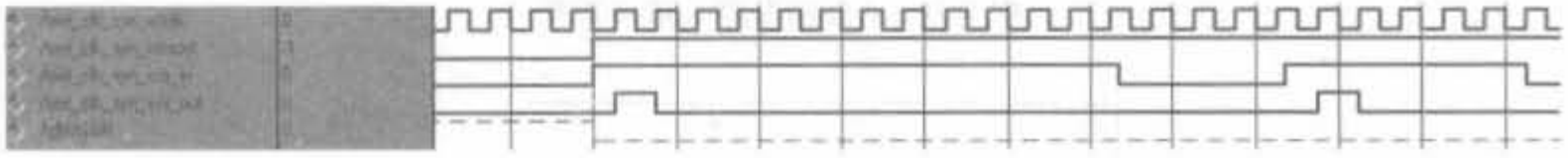


图 2-19　同步电路的仿真结果示意图

其中，如果在时钟的上升沿 din="1"，则 x=1，y=0，dout=x and (not y)=1；如果 din="1"超过一个时钟宽度，则 x=1，y=1，dout=x and (not y)=0。即使 din 在时钟周期内出现抖动，也不会影响输出结果，信号还是被整形成一个时钟宽度。所以，不管是长周期信号还是短周期信号，经过同步采样后，有效高电平宽度都等于时钟周期。

3. 同步状态机的 Verilog 实现

状态机一般包括组合逻辑和寄存器逻辑两部分。组合电路用于状态译码和产生输出信号，寄存器用于存储状态。状态机的下一个状态及输出不仅与输入信号有关，还与寄存器的当前状态有关。根据输出信号产生方法的不同，状态机可分为米里(Mealy)型和摩尔(Moore)型。前者的输出是当前状态和输入信号的函数，后者的输出仅是当前状态的函数。在硬件设计时，根据需要决定采用哪种状态机。

1) 状态编码

状态编码又称状态分配。通常有多种编码方法，编码方案选择得当，设计的电路可以简单；反之，电路会占用过多的逻辑或速度降低。设计时，需要综合考虑电路复杂度和电路性能这两个因素。下面主要介绍二进制编码、格雷编码和独热码。

二进制编码和格雷码都是压缩状态编码。二进制编码的优点是使用的状态向量最少，但从一个状态转换到相邻状态时，可能有多个比特位发生变化，瞬变次数多，易产生毛刺。格雷编码在相邻状态的转换中，每次只有1个比特位发生变化，虽减少了产生毛刺和一些暂态的可能，但不适用于有很多状态跳转的情况。

独热码是指对任意给定的状态，状态向量中只有1位为1，其余位都为0。n状态的状态机需要n个触发器。这种状态机的速度与状态的数量无关，仅取决于到某特定状态的转移数量，速度很快。当状态机的状态增加时，如果使用二进制编码，状态机速度会明显下降。而采用独热码，虽然多用了触发器，但由于状态译码简单，节省和简化了组合逻辑电路。独热编码还具有设计简单、修改灵活、易于综合和调试等优点。对于寄存器数量多，而门逻辑相对缺乏的FPGA器件，采用独热编码可以有效提高电路的速度和可靠性，也有利于提高器件资源的利用率。独热编码有很多无效状态，应该确保状态机一旦进入无效状态时，可以立即跳转到确定的已知状态。

2）有限状态机的Verilog实现

用Verilog语言描述有限状态机可使用多种风格，不同的风格会极大地影响电路性能。通常有3种描述方式：单always块、双always块和三always块。

单always块把组合逻辑和时序逻辑用同一个时序always块描述，其输出是寄存器输出，无毛刺。但是这种方式会产生多余的触发器，代码难于修改和调试，应该尽量避免使用。

双always块大多用于描述Mealy状态机和组合输出的Moore状态机，时序always块描述当前状态逻辑，组合逻辑always块描述次态逻辑并给输出赋值。这种方式结构清晰，综合后的面积和时间性能好。但组合逻辑输出往往会有毛刺，当输出向量作为时钟信号时，这些毛刺会对电路产生致命的影响。

三always块大多用于同步Mealy状态机，两个时序always块分别用来描述现态逻辑和对输出赋值，组合always块用于产生下一状态。这种方式的状态机也是寄存器输出，输出无毛刺，并且代码比单always块清晰易读，但是面积大于双always块。随着芯片资源和速度的提高，目前这种方式得到了广泛应用。

下面以三always块模块给出状态机的Verilog模板。

```
// 构成状态跳转环
always @(posedge clk or negedge rst_n)
    current_state <= next_state;

// 完成状态机的内部逻辑
always @(current_state or ) begin
    case(current_state)
        S1: next_state = S2;
        S2: next_state = S1;
    default: next_state = S2;
    endcase
end

// 完成状态机的外部逻辑
always @(current_state or ) begin
```

```
        case(current_state)
        S1:
        S2:
        default:
        endcase
    end
```

3）综合状态机的一般原则

在硬件描述语言中，许多基于仿真的语句虽然符合语法规则，但是不能映射到硬件逻辑电路单元。如果要最终实现硬件设计，必须写出可以综合的程序。通常，综合的原则为：

（1）综合之前一定要进行仿真，仿真会暴露逻辑错误。如果不做仿真，没有发现的逻辑错误会进入综合器，使综合的结果产生同样的逻辑错误。

（2）每一次布线之后都要进行仿真，在器件编程或流片之前一定要进行最后的仿真。

（3）用 Verilog HDL 描述的异步状态机是不能综合的，应该避免用综合器来设计。在必须设计异步状态机时，建议用电路图输入的方法。

（4）状态机应该有一个异步或同步复位端，以便在通电时将硬件电路复位到有效状态。建议使用异步复位，以简化硬件开销。

（5）时序逻辑电路建模时，用非阻塞赋值。用 always 块写组合逻辑时，采用阻塞赋值。不要在多个 always 块中为同一个变量赋值。

（6）always 块中应该避免组合反馈回路。在赋值表达式右端参与赋值的信号都必须出现在敏感信号列表中；否则在综合时，会为没有列出的信号隐含地产生一个透明锁存器。

2.6.3 常用数字处理算法的 Verilog 实现

1. 加法器的 Verilog 实现

1）串行加法器

组合逻辑的加法器可以利用真值表，通过与门和非门简单地实现。假设 x_i 和 y_i 表示两个加数，S_i 表示和，C_{i-1} 表示来自低位的进位，C_i 表示向高位的进位。每个全加器都执行如下所示的逻辑表达式：

$$S_i = P_i \oplus C_i \qquad \text{其中 } P_i = x_i \oplus y_i \tag{2-1}$$

$$C_i = P_i \cdot C_{i-1} + G_i \qquad \text{其中 } G_i = x_i \cdot y_i \tag{2-2}$$

这样可以得到加法器的一种串行结构。因此，式(2-1)所示的加法器也被称为串行加法器。图 2-20 给出了一个4 位串行加法器的结构示意图。

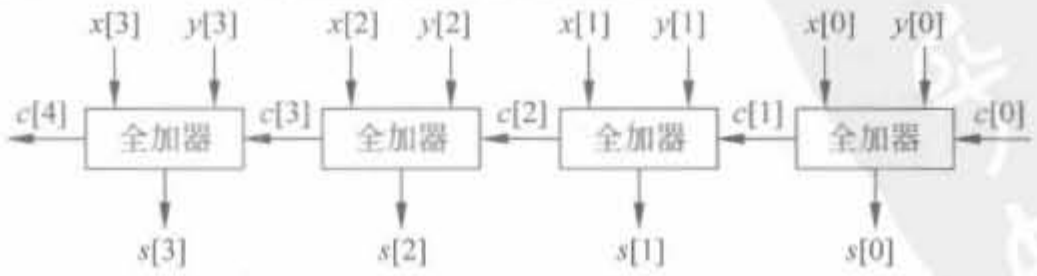


图 2-20 串行加法器的结构示意图

在数字信号处理中，常常要用到多位数字量的加法运算。如果用串行加法器实现，最节省资源，但速度较慢。而并行加法器能满足高速处理的时序要求，并且结构不复杂。现在普遍使用的并行加法器是超前进位加法器，只是在几个全加器的基础上增加了一个超前进位形成逻辑，以减少由于逐步进位信号的传递所造成的延时。图 2-21 给出了一个 4 位并行加法器的结构示意图。在 4 位并行加法器的基础上，可以递推出 16 位、32 位和 64 位的快速并行加法器。

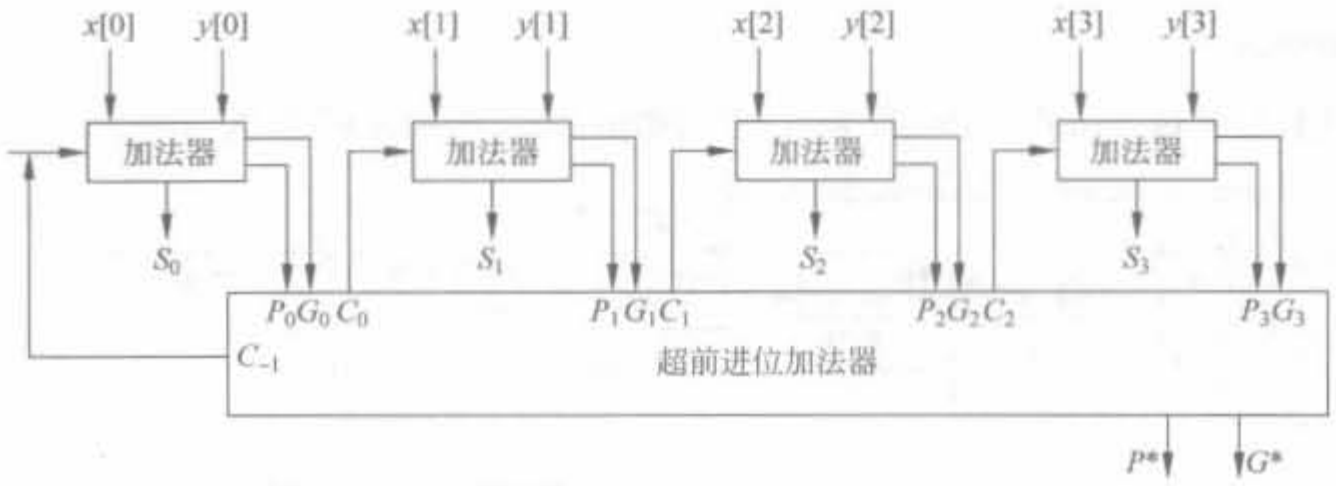


图 2-21 并行加法器的示意图

2）流水线加法器

在使用了并行加法器后，仍旧只有在输出稳定后才能输入新的数进行下一次计算，即计算的节拍必须大于运算电路的延迟。此外，许多门级电路和布线的延迟会随着位数的增加而累加，因此加法器的频率还是受到了限制。但如果采用流水线，就有可能将一个算术操作分解为一些小规模的基本操作，将进位和中间值存储在寄存器中，并在下一个时钟周期内继续运算，这样就可以提高电路的利用效率。将流水线规则应用于 FPGA 中，只需要很少或根本不需要额外的成本。这是因为每个逻辑单元都包含两个触发器，大多数情况下，这两个触发器或者没有用到，或者用于存储布线资源，那么就可以利用它们来实现流水线结构。如果采用了流水线后，加法器的速度仍然不能满足需要，可以采用第 3 章中将提到的串并转换来进一步提高计算的并行度。

由于一个 Slice 中有两个触发器，还需要有 1 个触发器来作为进位输出，那么采用 N 级流水线，就可以构造一个最大位数为 $2N-1$ 的加法器。

下面给出一个 16 位流水线加法器的代码。

例 2-24 16 位 2 级流水线加法器的 Verilog 设计。

```
module adder16_2(cout,sum,clk,cina,cinb,cin);
    input [15:0]cina,cinb;
    input clk,cin;
    output [15:0] sum;
    output cout;
    reg cout;
    reg cout1;
    reg[7:0] sum1;
    reg[15:0] sum;
```

```
always @(posedge clk) begin // 低 8 位相加;
  {cout1,sum1} = {cina[7],cina[7:0]} + {cinb[7],cinb[7:0]} + cin;
end

always @(posedge clk) begin // 高 8 位相加,并连成 16 位
  {cout,sum} = {{cina[15],cina[15:8]}} + {cinb[15],cinb[15:8]} + cout1,sum1};
end

endmodule
```

上述程序经过 Synplify Pro 综合后,得到如图 2-22 所示的 RTL 级结构图。

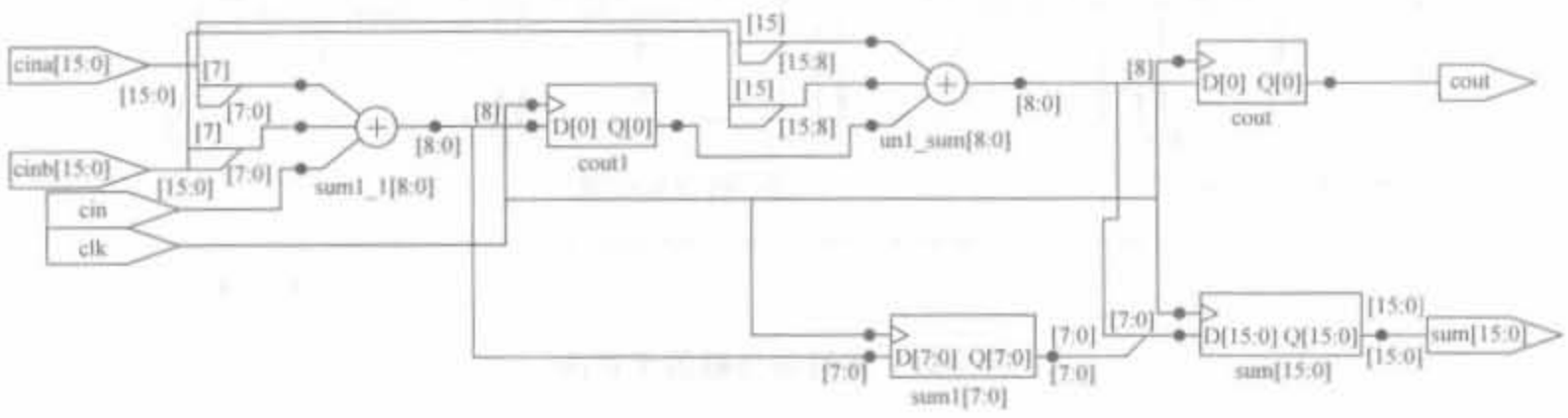


图 2-22　16 位加法器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真,其结果如图 2-23 所示,正确地实现了 16 比特加法。

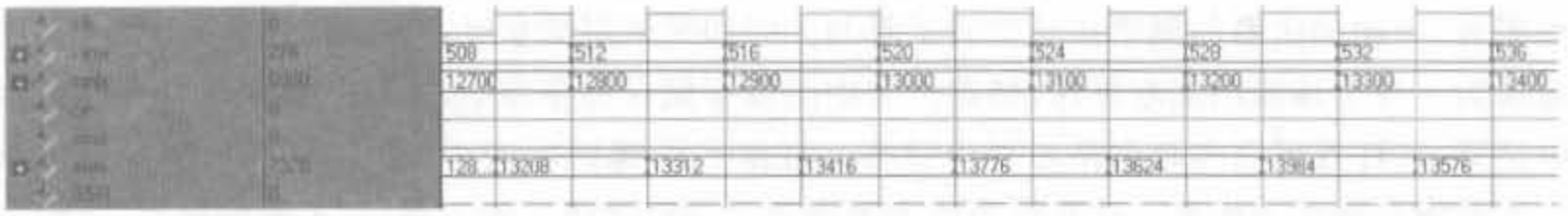


图 2-23　16 位加法器的仿真结果示意图

2. 乘法器的 Verilog 实现

1) 串行乘法器

对于两个 N 位二进制数 x、y 的乘积,最简单的方法就是利用移位操作来实现,用公式可以表示为

$$P = xy = \sum_{k=0}^{N-1} x_k 2^k y \tag{2-3}$$

这样,输入量随着 k 的位置连续地变化,然后累加 $2^k y$。

例 2-25　用 Verilog 实现一个 8 位串行乘法器。

```
module ade (clk,x,y,p);
input    clk;
input    [7:0] x,y;
output   [15:0] p;
reg      [15:0] p;

parameter s0 = 0,s1 = 1,s2 = 2;
```

```
reg [2:0] count;
reg [1:0] state;
reg   [15:0] p1,t;              // 比特位加倍
    reg [7:0] y_reg;

always @(posedge clk) begin
          case (state)
           s0: begin            // 初始化
              y_reg<=y;
              state<=s1;
              count=0;
              p1<=0;
              t<={{8{x[7]}},x};
           end
           s1: begin            // 处理步骤
            if (count==7)     //判断是否处理结束
               state<=s2;
            else begin
               if (y_reg[0]==1)
                  p1<=p1+t;
                  y_reg<=y_reg>>1; //移位
                  t<=t<<1;
                  count<=count+1;
                  state<=s1;
               end
           end
           s2: begin
              p<=p1;
              state<=s0;
           end
          endcase
      end
endmodule
```

上述程序在 Synplify Pro 中综合后，得到如图 2-24 所示的 RTL 级结构示意图。

图 2-25 给出了串行乘法器模块在 ModelSim 中的仿真结果，验证了功能的正确性。

从仿真结果可以看出，上述串行乘法器的速度比较慢，延时很大。但这种乘法器的优点是所占用的资源是所有类型乘法器中最少的，在低速信号处理中有着广泛的应用。

2）流水线乘法器

一般的快速乘法器通常采用逐位并行的迭代阵列结构，将每个操作数的 N 位都并行地提交给乘法器。但是一般对于 FPGA 来讲，进位的速度快于加法的速度，因此这种阵列结构并不是最优的。所以可以采用多级流水线的形式，将相邻的两个部分的乘积结果再加到最终的输出乘积上，即排成一个二叉树形式的结构。这样，对于 N 位乘法器，需要 $\log_2(N)$ 级来实现。一个 8 位乘法器如图 2-26 所示。

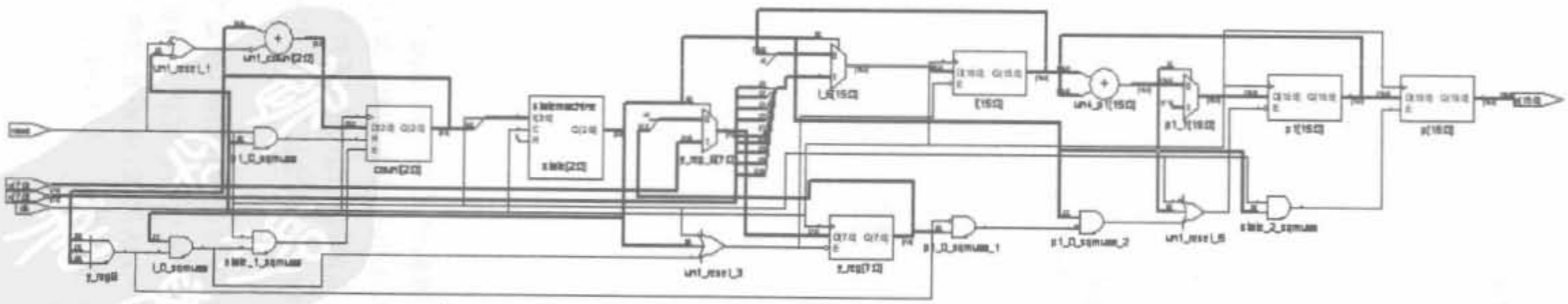


图 2-24　串行乘法器的 RTL 级结构图

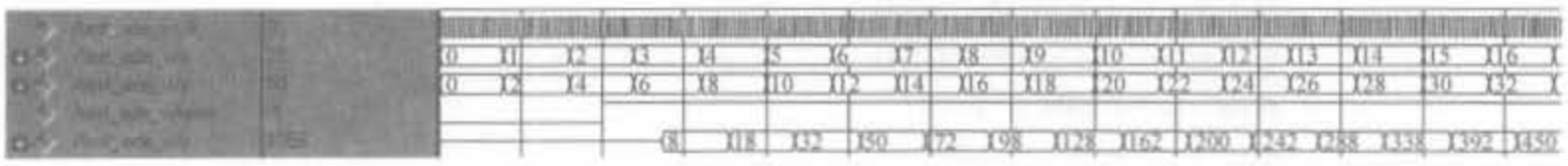


图 2-25 串行乘法器的局部仿真结果示意图

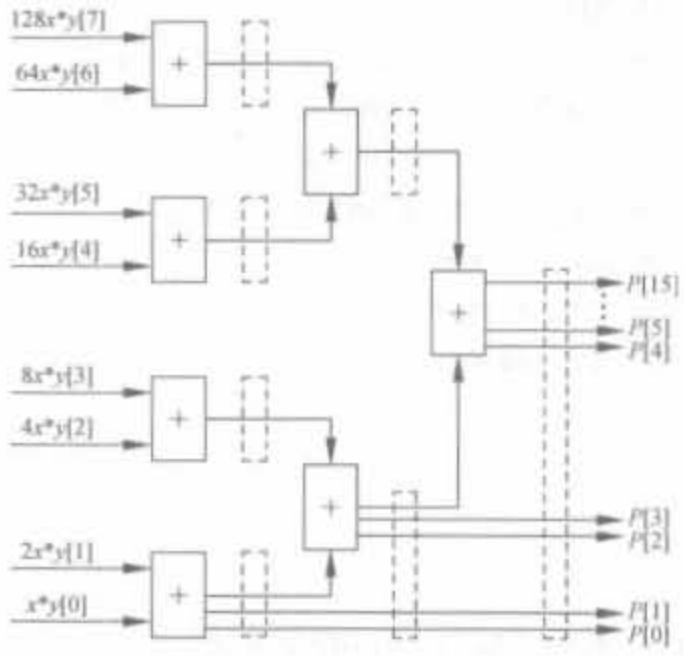


图 2-26 流水线乘法器结构图

例 2-26 用 Verilog HDL 实现一个 4 位的流水线乘法器。

```
module mul_addtree(mul_a,mul_b,mul_out,clk,rst_n);
parameter MUL_WIDTH = 4;
parameter MUL_RESULT = 8;

input [MUL_WIDTH-1:0] mul_a;
input [MUL_WIDTH-1:0] mul_b;
input clk;
input rst_n;
output [MUL_RESULT-1:0] mul_out;
reg [MUL_RESULT-1:0] mul_out;
reg [MUL_RESULT-1:0] stored0;
reg [MUL_RESULT-1:0] stored1;
reg [MUL_RESULT-1:0] stored2;
reg [MUL_RESULT-1:0] stored3;
reg [MUL_RESULT-1:0] add01;
reg [MUL_RESULT-1:0] add23;

always @ (posedge clk or negedge rst_n)
begin
   if(!rst_n)
      begin //初始化寄存器变量
         mul_out<=8'b0000_0000;
         stored0<=8'b0000_0000;
         stored1<=8'b0000_0000;
```

```
                stored2<=8'b0000_0000;
                stored3<=8'b0000_0000;
                add01<=8'b0000_0000;
                add23<=8'b0000_0000;
            end
        else
            begin //实现移位相加
                stored3<=mul_b[3]?{1'b0,mul_a,3'b0}:8'b0;
                stored2<=mul_b[2]?{2'b0,mul_a,2'b0}:8'b0;
                stored1<=mul_b[1]?{3'b0,mul_a,1'b0}:8'b0;
                stored0<=mul_b[0]?{4'b0,mul_a}:8'b0;
                add01<=stored1+stored0;
                add23<=stored3+stored2;
                mul_out<=add01+add23;
            end
    end

endmodule
```

上述程序在 Synplify Pro 软件中综合后，得到如图 2-27 所示的 RTL 级结构示意图。

图 2-28 给出了流水线乘法器模块在 ModelSim 中的仿真结果，验证了功能的正确性。

从仿真结果可以看出，上述流水线乘法器比串行加法器的速度快很多，因此它在非高速的信号处理中有着广泛的应用。至于高速信号的乘法，一般需要利用 FPGA 芯片中内嵌的硬核 DSP 单元来实现。

3. 无符号除法器的 Verilog 实现

两个无符号二进制数(如正整数)相除的时序算法是通过“减并移位”的思想来实现的，即从被除数中重复地减去除数，直到已检测到余数小于除数。这样，可以通过累计减法运算的次数而得到商；而余数是在减法运算结束时被除数中的剩余值。当除数较小时，这种基本电路必须进行多次减法，因此效率都不高。图 2-29 给出了一种更加高效的除法器基本结构[x]。

在进行两个数相除的运算时，通常采用的步骤是调整除数与被除数，使其最高位对齐；然后反复地从被除数中减去除数，并将除数向被除数的最低位移动，且在高效除法器结构中并行执行这些操作步骤。在具体的硬件实现中，是通过将被除数寄存器的内容不断地向除数的最高位移动来完成除法运算的。

在设计除法器结构的过程中应特别小心。在图 2-29 所示的减法运算步骤中，必须要将除数和被除数对齐，这取决于它们的相应大小和每个字的最高一位 1 的相对位置。同样，被除数寄存器也应向左扩展 1 位，以适应可能出现的已调整的除法寄存器内容初始值超过被除数寄存器中相应 4 位值的情况。在这种情况下，在进行减法操作之前，应从被除数的最高位移出一个 1。例如，$(1100)_2$ 与 $(0111)_2$ 相除，应首先将被除数向左移，为下一步减法运算调整好被除数。因此，该机器的控制器会变得更加复杂，而且要包括能使除数和被除数移动的控制信号，如图 2-29 所示。

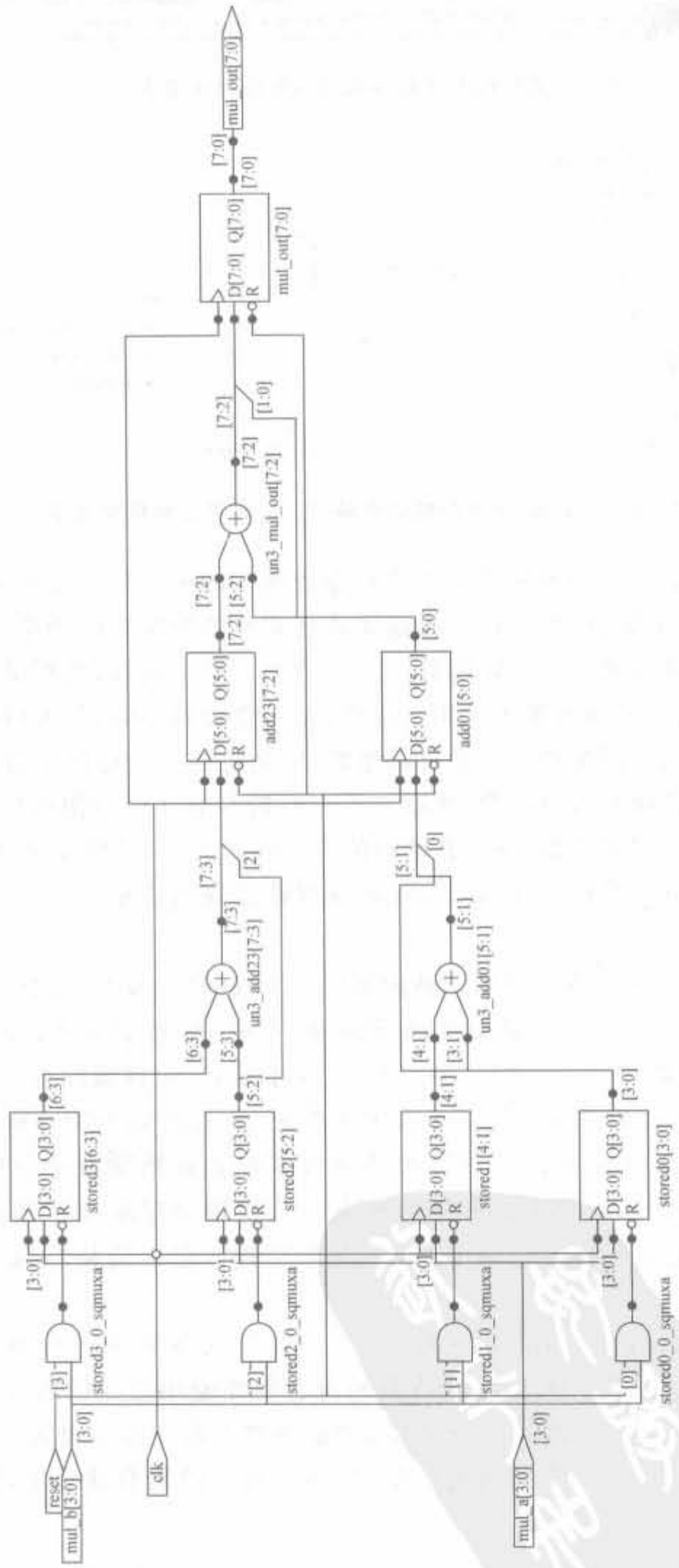


图 2-27 流水线乘法器的 RTL 级结构示意图

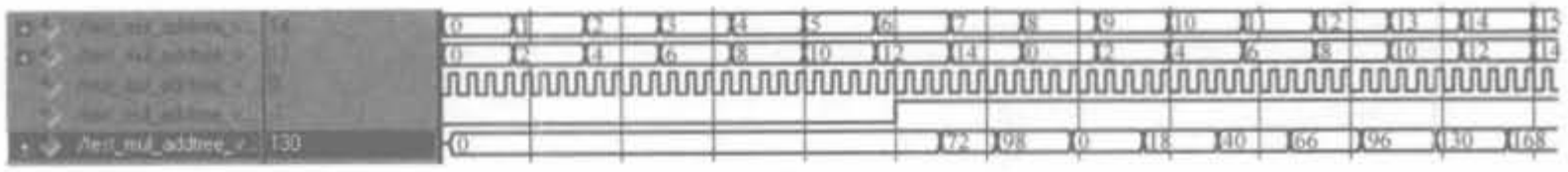


图 2-28　流水线乘法器的局部仿真结果示意图

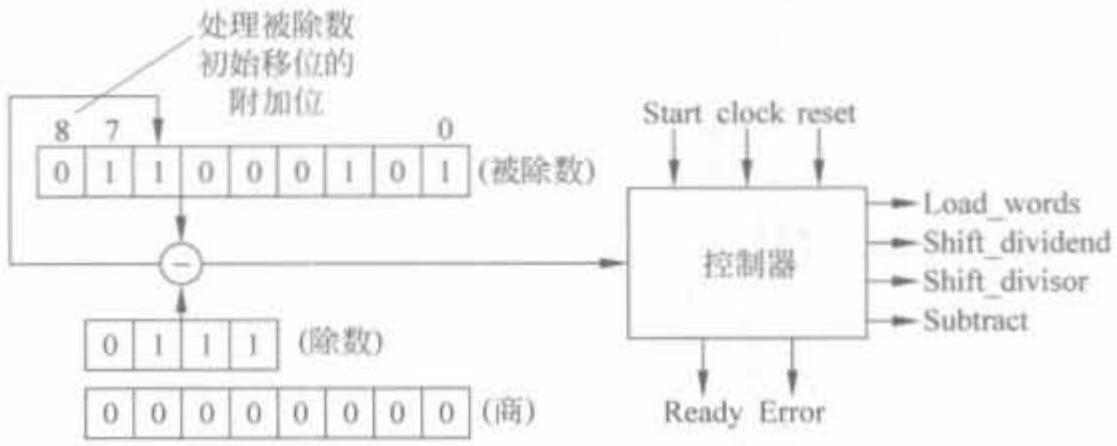


图 2-29　8 位被除数,4 位除数的无符号二进制字自调整除法器

该物理结构将调整除数字与被除数的 8 位数据通道中的最左边 4 位对齐。在操作中,被除数字不断地从右向左移动,而且每一步都要从已调整的被除数的相应位中减去除数。这种操作取决于除数是否比被除数选定部分的对应值小。调整机器,使得从被除数中减去的不是除数,而是除数与 2 的幂的最大乘积。这样,在当除数较小时,就可以消去一些需要重复进行的减法运算。这种调整被称为是自调整的,因为它在一个除法运算序列的开始就能自动判断是否需要调整除数或被除数,这取决于它们最左非 0 位的相对位置。在除法运算中经常性地启动对两个字的调整,使得其最高位为 1 的方法效率较低,因为这可能会需要更多的移位。所采用的方法是在一开始就将除数移到被除数的最左非 0 位(而不是被除数的最低位)。

有两种需要对数据通路字进行初始调整的情况:(1)被除数最左 4 位的值小于除数的值(例如$(1100)_2$除以$(1110)_2$);(2)除数的最低位为 0,同时除数字节可以向左移动,而且仍然可以去除被除数(例如$(1100)_2$除以$(0101)_2$)。对于前者,应将被除数依次向左移动 1 位,直到扩展 1 位的被除数的最左 5 位等于或大于除数,或者直到不能再移位为止。而对于后者,应将除数向左移,直到再移动所得到的字节不能去除被除数字节的最左 4 位为止(不包括扩展位)。余数位在除法运算序列结束时的物理位置取决于被除数是否进行了移位调整。因此,可将调整移位记录下来,并用来控制状态机调整在执行序列结束后的余数值。

图 2-30 给出了自调整除法器的状态转移图[3]。在给定状态下,从一个状态节点出发的支路中所使用的控制标记仅适于该支路,而在其他没有明确使用该标记离开当前状态的支路中被视为是不成立的。在任何离开一个状态节点的支路中都没有出现的标记被认为是无关紧要的。只有 S_idle 状态下才会给出复位信号,而在其他的所有状态上,复位信号均被视为是异步动作。

该机器的状态与它的动作有关。在 S_Adivr 状态下,Shift_divisor 的动作是将除法调整到被除数的最高非 0 位;在 S_Adivn 状态下,Shift_dividend 的动作是调整被除数寄存

器，以进行减法运算；在 S_div 状态下，同时进行实际的减法运算和许多移位操作。在状态 S_Adivn 和 S_Adivr 下，变量 Max 将检测所允许的最大移位何时发生。

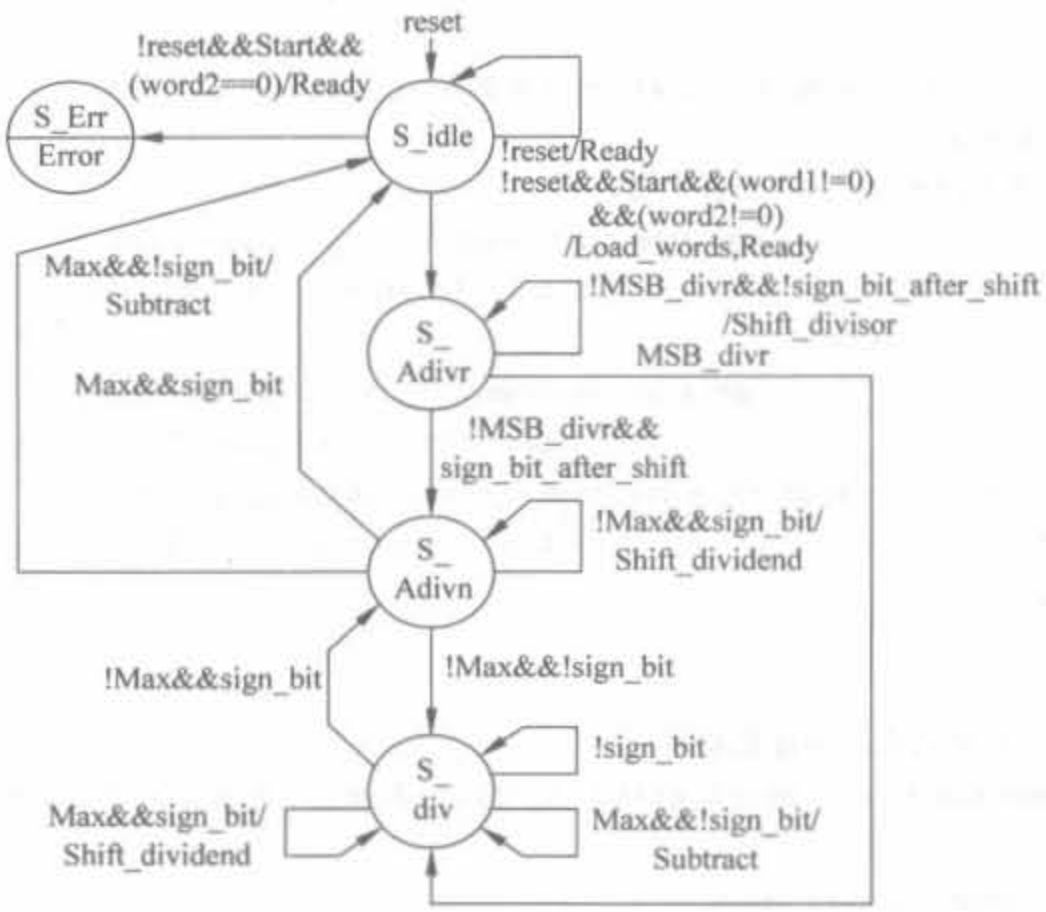


图 2-30 自调整除法器的状态转移图

例 2-27 用 Verilog 实现一个被除数为 8 位，除数为 4 位的高效除法器。

```
module divider(clock,reset,word1,word2,Start,quotient,remainder,Ready,Error);

    parameter L_divn = 8;
    parameter L_divr = 4;
    parameter S_idle = 0,S_Adivr = 1,S_Adivn = 2,S_div = 3,S_Err = 4;
    parameter L_state = 3,L_cnt = 4,Max_cnt = L_divn-L_divr;
    input [L_divn-1:0] word1;          //被除数数据通道
    input [L_divr-1:0] word2;          //除数数据通道
    input Start,clock,reset;
    output [L_divn-1:0] quotient;      //商
    output [L_divn-1:0] remainder;     //余数
    output Ready,Error;

    reg [L_state-1:0] state,next_state;
    reg Load_words,Subtract,Shift_dividend,Shift_divisor;
    reg [L_divn-1:0] quotient;
    reg [L_divn:0] dividend;           //扩展的被除数
    reg [L_divr-1:0] divisor;
    reg [L_cnt-1:0] num_shift_dividend,num_shift_divisor;
    reg [L_divr:0] comparison;
    wire MSB_divr = divisor[L_divr-1];
    wire Ready = ((state == S_idle) &&!reset);
```

```
wire Error = (state == S_Err);
wire Max = (num_shift_dividend == Max_cnt + num_shift_divisor);
wire sign_bit = comparison[L_divr];

always@(state or dividend or divisor or MSB_divr) begin //从被除数中减去除数
    case(state)
        S_Adivr: if(MSB_divr == 0)
                    comparison = dividend[L_divn: L_divn-L_divr] +
                                 {1'b1,~(divisor<<1)} + 1'b1;
                 else
                    comparison = dividend[L_divn: L_divn-L_divr] +
                                 {1'b1,~divisor[L_divr-1:0]} + 1'b1;
        default: comparison = dividend[L_divn: L_divn-L_divr] +
                                 {1'b1,~divisor[L_divr-1:0]} + 1'b1;
    endcase
end

//将余数移位来对应于整体的移位
assign remainder = (dividend[L_divn-1: L_divn-L_divr])-num_shift_divisor;

always@(posedge clock) begin
    if(reset)
        state<=S_idle;
    else
        state<=next_state;
end

//次态与控制逻辑
always@(state or word1 or word2 or state or comparison or sign_bit or Max) begin
    Load_words = 0;
    Shift_dividend = 0;
    Shift_divisor = 0;
    Subtract = 0;
    case(state)
        S_idle: case(Start)
            0: next_state = S_idle;
            1: if(word2 == 0)
                    next_state = S_Err;
                else if(word1) begin
                    next_state = S_Adivr;
                    Load_words = 1;
                end
                else
                    next_state = S_idle;
            endcase
        S_Adivr: case(MSB_divr)
            0: if(sign_bit == 0) begin
```

```
                    next_state = S_Adivr;
                    Shift_divisor = 1;          //可移动除数
                end
                else if(sign_bit == 1) begin
                    next_state = S_Adivn;       //不可移动除数
                end
            1: next_state = S_div;
        endcase
    S_Adivn: case({Max,sign_bit})
            2'b00: next_state = S_div;
            2'b01: begin
                    next_state = S_Adivn;
                    Shift_dividend = 1;
                end
            2'b10: begin
                    next_state = S_idle;
                    Subtract = 1;
                end
            2'b11: next_state = S_idle;
        endcase
    S_div: case({Max,sign_bit})
            2'b00: begin
                    next_state = S_div;
                    Subtract = 1;
                    end
            2'b01: next_state = S_Adivn;
                    2'b10: begin
                            next_state = S_div;
                            Subtract = 1;
                        end
                    2'b11: begin
                            next_state = S_div;
                            Shift_dividend = 1;
                        end
                    endcase
                default: next_state = S_Err;
        endcase
    end

    always@(posedge clock)begin                 //寄存器,数据通道操作
     if(reset)begin
         divisor<=0;
         dividend<=0;
         quotient<=0;
         num_shift_dividend<=0;
         num_shift_divisor<=0;
     end
     else if(Load_words == 1) begin
         dividend<=word1;
         divisor<=word2;
         quotient<=0;
```

```
            num_shift_dividend<=0;
            num_shift_divisor<=0;
        end
        else if(Shift_divisor) begin
            divisor<=divisor<<1;
            num_shift_divisor<=num_shift_divisor+1;
        end
        else if(Shift_dividend) begin
            dividend<=dividend<<1;
            quotient<=quotient<<1;
            num_shift_dividend<=num_shift_dividend+1;
        end
        else if(Subtract) begin
            dividend[L_divn:L_divn-L_divr]<=comparison;
            quotient[0]<=1;
        end
    end

endmodule
```

上述程序经过 Synplify 综合后，得到如图 2-31 所示的 RTL 级结构。

在 ModelSim 中经过仿真，结果如图 2-32 所示。仿真中输入了两组数据，且在输出信号 Ready 为高时输出相应的商和余数。当被除数为 57，除数为 6 时，可以看到，商和余数分别为 9 和 3；当被除数为 98，除数为 9 时，可以看到，商和余数分别为 10 和 8。这表明了上述程序的正确性。

4. CORDIC 算法的 Verilog 实现

1) CORDIC 算法的原理

CORDIC 算法可以将多种难以用硬件电路直接实现的复杂运算分解为统一的简单的移位、加迭代运算，而且结构规则，运算周期可以预测，适合于 VLSI 实现。许多数字信号处理算法，如 DXT、FFT、复数滤波器、格形滤波器、基于 Givens 旋转的 QR 分解、奇异值/特征值分解、最小二乘求解以及线性系统求解等，都很容易用圆周旋转或双曲旋转来描述其基本的操作，因此都可以用 CORDIC 算法得到很好的实现。所以，以 CORDIC 为核心的 FPGA 应用日益受到人们的重视。

CORDIC 是用于计算广义矢量旋转的一种迭代方法。由 J. D. Volder 于 1959 年提出[4]，主要用于三角函数、双曲函数、指数和对数的运算。该算法使得矢量的旋转和定向运算不需要三角函数表以及乘法、开方、反三角函数等复杂的运算，仅需要进行加、减和移位即可。1971 年，Walther 提出了统一的 CORDIC 算法，引入了参数 m，将 CORDIC 实现的 3 种迭代模式：三角运算、双曲运算和线性运算统一于一个表达式下，形成目前所用到的 CORDIC 算法最基本的数学基础。该算法的基本思想是通过一系列固定的、与运算基数相关的角度不断偏摆，以逼近所需的旋转角度，可由下列等式描述：

$$\begin{bmatrix} X(n) \\ Y(n) \end{bmatrix} = \prod_{i=1}^{n} \cos(m^{1/2}a_i) \begin{bmatrix} 1 & -m^{1/2}\sigma_i \tan(m^{1/2}\sigma_i) \\ m^{1/2}\sigma_i \tan(m^{1/2}\sigma_i) & 1 \end{bmatrix} \begin{bmatrix} X(0) \\ Y(0) \end{bmatrix} \tag{2-4}$$

$$Z(n) = Z(0) + \sum_{i=1}^{n} \sigma_i a_i \tag{2-5}$$

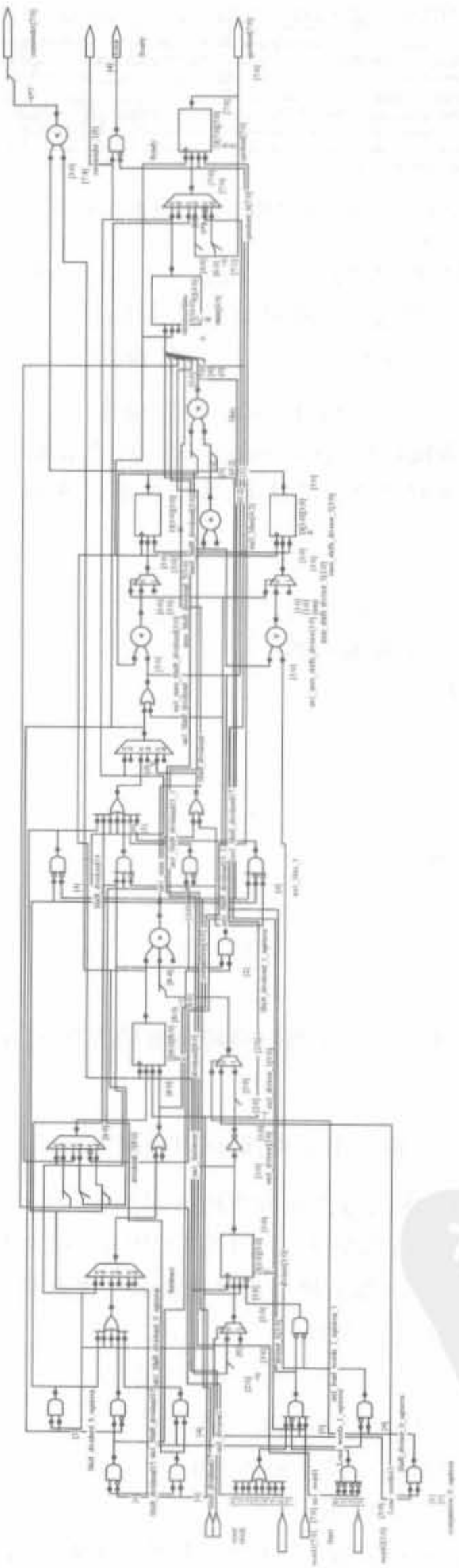


图 2-31 无符号高效除法器的 RTL 级结构图

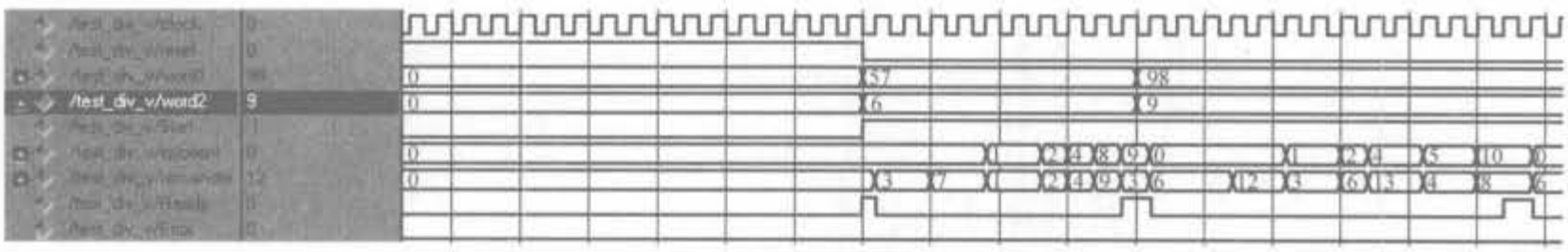


图 2-32　无符号高效除法器的仿真波形

其中，$X(n)$、$Y(n)$和 $Z(n)$为所期望得到的函数。根据 $m=1$、-1 或 0，可以将上面的运算分别称为圆周旋转运算、双曲旋转运算或线形旋转运算。其中，

$$\sigma_i = \begin{cases} \text{sign}(Z(i)) & \text{旋转模式} \\ -\text{sign}(Y(i)) & \text{向量模式} \end{cases} \tag{2-6}$$

使结果 $Z(n)=0$ 的旋转称为旋转模式(rotation mode)，使结果 $Y(n)=0$ 的旋转称为向量模式(vector mode)。为了能达到所要求的结果，各旋转角 a_i 要满足下列条件：

$$\frac{1}{2}a_{i-1} < a_i < a_{i-1} \tag{2-7}$$

$$\tan(m^{1/2}a_i) = 2^{-F_m(i)} - \delta_i 2^{-G_m(i)} \tag{2-8}$$

其中，$\delta_i=0$ 或 1，$F_m(i)$和 $G_m(i)$是非负整数值。

最通常的微转角选择方法为：

当 $m=1$ 时，$\tan(\sigma_i)=2^{-i}$；

当 $m=-1$ 时，$\tanh(\sigma_i)=-2^{-i}$；

当 $m=0$ 时，$\sigma_i=2^{-i}$。

此时，每一级迭代运算可以简化为

$$X_{i+1} = X_i - m \times \sigma_i \times Y_i \times 2^{-i} \tag{2-9}$$

$$Y_{i+1} = Y_i + \sigma_i \times X_i \times 2^{-i} \tag{2-10}$$

$$Z_{i+1} = Z_i - \sigma_i \times a_i \tag{2-11}$$

可以仅由加法、减法和移位来实现。本级的微转角旋转方向 σ_i 由上一级运算结果和所处的旋转模式决定。

在所有级旋转之后需执行一次模校正运算，即乘以模校正因子 $k_{m,n} = \prod_{i=1}^{n} \cos(m^{1/2}a_i)$。一旦如上旋转一系列微转角之后，无论每个微转角的方向如何，对于确定的 m 值，当 n 趋向无穷大时，模校正因子趋近于一个极限值 K_m。因为 CORDIC 算法本身是一种逐位逼近算法，所以一般不论旋转级数 n 是多少，都直接应用其极限的二进制码作为模校正因子。

总结起来，CORDIC 算法的迭代公式为

$$\begin{cases} X_{i+1} = X_i - \sigma_i \times Y_i \times 2^{-i} \\ Y_{i+1} = Y_i + \sigma_i \times X_i \times 2^{-i} \\ Z_{i+1} = Z_i - \sigma_i \times \tan^{-1}(2^{-i}) \end{cases} \tag{2-12}$$

其中，i 为迭代次数，σ_i 的定义如式(2-6)所示。

对于不同的 m 值、工作模式和初始值，可以产生不同的结果，如表 2-3 所示。

表 2-3 CORDIC 算法在不同情况下的输出

m	模式	初始化	输出
1	旋转	$X_0=x$	$X_n=A(x\cos\theta-y\sin\theta)$
		$Y_0=y$	$Y_n=A(y\cos\theta+x\sin\theta)$
		$Z_n=\theta$	$Z_n=0$
1	旋转	$X_0=K$	$X_n=\cos\theta$
		$Y_0=0$	$Y_n=\sin\theta$
		$Z_n=\theta$	$Z_n=0$
1	向量	$X_0=x$	$X_n=A\text{sign}(X_0)\text{sqrt}(x^2+y^2)$
		$Y_0=y$	$Y_n=0$
		$Z_n=\theta$	$Z_n=\theta+\tanh^{-1}(y/x)$
0	旋转	$X_0=x$	$X_n=x$
		$Y_0=y$	$Y_n=y+xz$
		$Z_n=z$	$Z_n=0$
0	向量	$X_0=x$	$X_n=x$
		$Y_0=y$	$Y_n=0$
		$Z_n=z$	$Z_n=z+y/x$
−1	旋转	$X_0=x$	$X_n=A(x\cosh\theta-y\sinh\theta)$
		$Y_0=y$	$Y_n=A(y\cosh\theta+x\sinh\theta)$
		$Z_n=\theta$	$Z_n=0$
−1	旋转	$X_0=K$	$X_n=\cosh\theta$
		$Y_0=0$	$Y_n=\sinh\theta$
		$Z_n=\theta$	$Z_n=0$
−1	向量	$X_0=x$	$X_n=A\text{sign}(X_0)\text{sqrt}(x^2-y^2)$
		$Y_0=y$	$Y_n=0$
		$Z_n=\theta$	$Z_n=\theta+\tanh^{-1}(y/x)$

2) CORDIC 算法的 Verilog 实现

CORDIC 算法的实现方式有两种：简单状态机法和高速全流水线处理器。前者主要采用折叠/迭代方式，后者采用展开/流水线方式。

(1) 简单状态机结构

如果计算时间不严格的话，可以采用如图 2-33 所示的状态机。在每个周期内都将精确地计算一次式(2-9)～式(2-11)所示的迭代，其中最复杂的就是两个简形移位器。

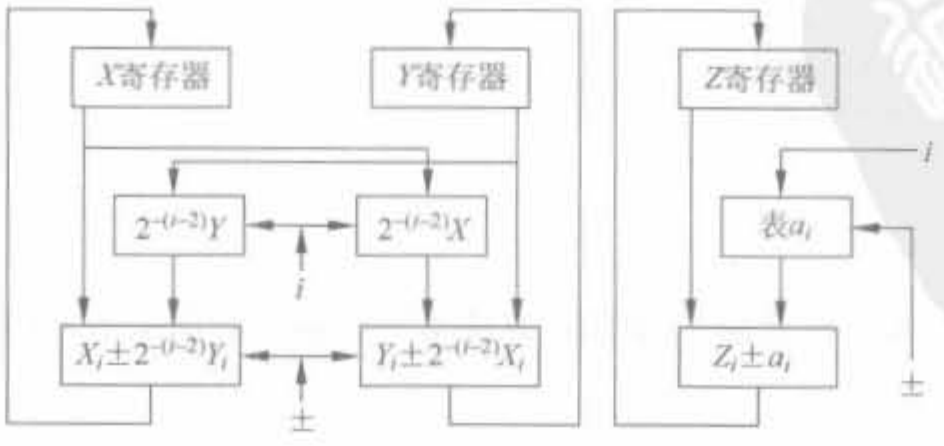


图 2-33 CORDIC 算法状态机

(2) 流水线 CORDIC 结构

流水线 CORDIC 结构虽然占用的硬件资源较多,但是流水线结构可以提高数据的吞吐率(Throughput)。对于大多数 DSP 算法来说,存在很多同一条指令连续处理很长一段数据的情况,此时高吞吐率更有意义。从当前 VLSI 的发展趋势上来看,芯片内的门资源相对富裕,对流水线 CORDIC 的实现规模约束很小。此外,流水线 CORDIC 不存在迭代式 CORDIC 的反馈回路,使得单元结构更加规则,有利于 VLSI 实现。

图 2-34 给出了 CORDIC 算法的一般流水线结构。

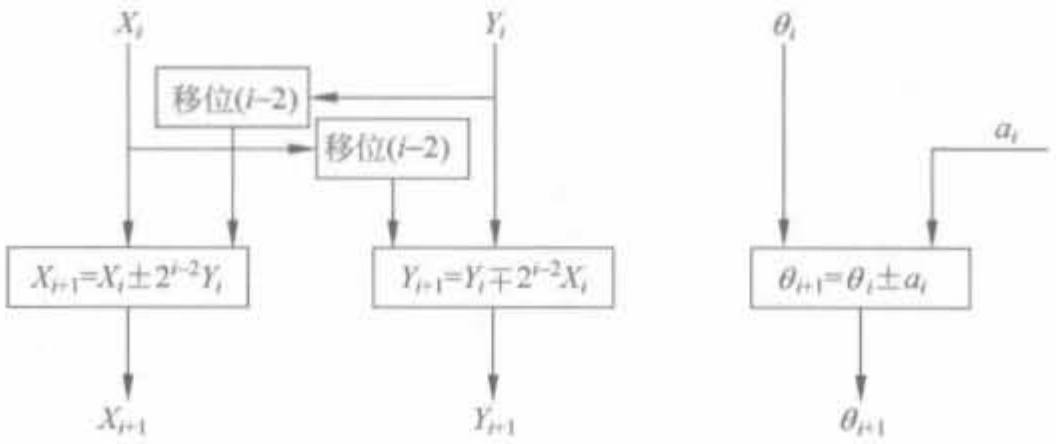


图 2-34 CORDIC 算法的一般流水线结构

例 2-28 基于 CORDIC 算法的平方根计算两个数平方和根的 Verilog 实现。

① MATLAB 代码

为了便于读者理解,首先用 MATLAB 实现计算 sqrt(x^2+y^2)的 CORDIC 算法,代码如下:

```
function Xout = mysqrt (x,y);
K = 8;
An = 1;
for i = 1: K
      a(i) = 1/(2^(i));
      An = An * sqrt(1 + 1/(4^i));
end
% anger = atan(a); % 计算出的角度
theta = 0;

for i = 1: K
      sigma = - sign(y);
      x = x - sigma * y/(2^i);
      y = y + sigma * x/(2^i);
   %  z = z - sigma * a(i);
end

Xout = x/An;
```

经过测试,上述程序可利用 CORDIC 算法来计算输入数据平方和的根。

② Verilog 代码

```
module mysqrt(clk,x,y,fout,fy);
    input          clk;
```

```
input  [15:0]  x;
input  [15:0]  y;
output [15:0]  fout;
output [15:0]  fy;
reg    [15:0]  fout;
reg    [15:0]  fy;

//采用 8 级流水线来实现
wire   [15:0] x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;
wire   [15:0] x6,y6,x7,y7,x8,y8;

//
reg    [15:0] xtemp,ytemp;
reg    addx1,addx2,addx3,addx4,addx5,addx6,addx7,addx8;
reg    addy1,addy2,addy3,addy4,addy5,addy6,addy7,addy8;

always @(posedge clk) begin
   xtemp<=x;
    ytemp<=y;

    fout<=x8;
    fy <=y8;

    if(ytemp[15]==0) begin
        addx1<=1;
        addy1<=0;
    end
    else begin
        addx1<=0;
        addy1<=1;
    end
  if(y1[15]==0) begin
        addx2<=1;
        addy2<=0;
    end
    else begin
        addx2<=0;
        addy2<=1;
    end
  if(y2[15]==0) begin
        addx3<=1;
        addy3<=0;
    end
    else begin
        addx3<=0;
        addy3<=1;
    end
  if(y3[15]==0) begin
        addx4<=1;
        addy4<=0;
    end
```

```
else begin
    addx4<=0;
    addy4<=1;
end
  if(y4[15]==0) begin
    addx5<=1;
    addy5<=0;
end
else begin
    addx5<=0;
    addy5<=1;
end
  if(y5[15]==0) begin
    addx6<=1;
    addy6<=0;
end
else begin
    addx6<=0;
    addy6<=1;
end
  if(y6[15]==0) begin
    addx7<=1;
    addy7<=0;
end
  else begin
    addx7<=0;
    addy7<=1;
  end
    if(y7[15]==0) begin
      addx8<=1;
      addy8<=0;
  end
  else begin
      addx8<=0;
      addy8<=1;
  end
end

//第1次迭代模块
addandsub addandsub1x(
.A(xtemp),
.B({ytemp[15],ytemp[15:1]}),
.ADD(addx1),
.Q(x1),
.CLK(clk));

addandsub addandsub1y(
.A(ytemp),
.B({xtemp[15],xtemp[15:1]}),
.ADD(addy1),
.Q(y1),
```

```
.CLK(clk));

//第 2 次迭代模块
addandsub addandsub2x(
.A(x1),
.B({{2{y1[15]}},y1[15:2]}),
.ADD(addx2),
.Q(x2),
.CLK(clk));

addandsub addandsub2y(
.A(y1),
.B({{2{x1[15]}},x1[15:2]}),
.ADD(addy2),
.Q(y2),
.CLK(clk));

//第 3 次迭代模块
addandsub addandsub3x(
.A(x2),
.B({{3{y2[15]}},y2[15: 3]}),
.ADD(addx3),
.Q(x3),
.CLK(clk));

addandsub addandsub3y(
.A(y2),
.B({{3{x2[15]}},x2[15: 3]}),
.ADD(addy3),
.Q(y3),
.CLK(clk));

//第 4 次迭代模块
addandsub addandsub4x(
.A(x3),
.B({{4{y3[15]}},y3[15: 4]}),
.ADD(addx4),
.Q(x4),
.CLK(clk));

addandsub addandsub4y(
.A(y3),
.B({{4{x3[15]}},x3[15: 4]}),
.ADD(addy4),
.Q(y4),
.CLK(clk));

//第 5 次迭代模块
addandsub addandsub5x(
.A(x4),
.B({{5{y4[15]}},y4[15: 5]}),
```

```
    .ADD(addx5),
    .Q(x5),
    .CLK(clk));

    addandsub addandsub5y(
    .A(y4),
    .B({{5{x4[15]}},x4[15: 5]}),
    .ADD(addy5),
    .Q(y5),
    .CLK(clk));

    //第 6 次迭代模块
    addandsub addandsub6x(
    .A(x5),
    .B({{6{y5[15]}},y5[15:6]}),
    .ADD(addx6),
    .Q(x6),
    .CLK(clk));

    addandsub addandsub6y(
    .A(y5),
    .B({{6{x5[15]}},x5[15:6]}),
    .ADD(addy6),
    .Q(y6),
    .CLK(clk));

    //第 7 次迭代模块
    addandsub addandsub7x(
    .A(x6),
    .B({{7{y6[15]}},y6[15: 7]}),
    .ADD(addx7),
    .Q(x7),
    .CLK(clk));

    addandsub addandsub7y(
    .A(y6),
    .B({{7{x6[15]}},x6[15: 7]}),
    .ADD(addy7),
    .Q(y7),
    .CLK(clk));

    //第 8 次迭代模块
    addandsub addandsub8x(
    .A(x7),
    .B({{8{y7[15]}},y7[15:8]}),
    .ADD(addx8),
    .Q(x8),
    .CLK(clk));

    addandsub addandsub8y(
    .A(y7),
```

```
.B({{8{x7[15]}},x7[15:8]}),
.ADD(addy8),
.Q(y8),
.CLK(clk));

endmodule
```

上述程序的 RTL 级结构图比较复杂，这里就不再给出。在 ModelSim 6.2b 中完成仿真，其结果如图 2-35 所示，如竖线所标，输入为 170、170 时，经过 8 个时钟周期，输出 248，近似等于 $\sqrt{170^2+170^2}$。如果对输出计算精度有更高的要求，可加大迭代次数。

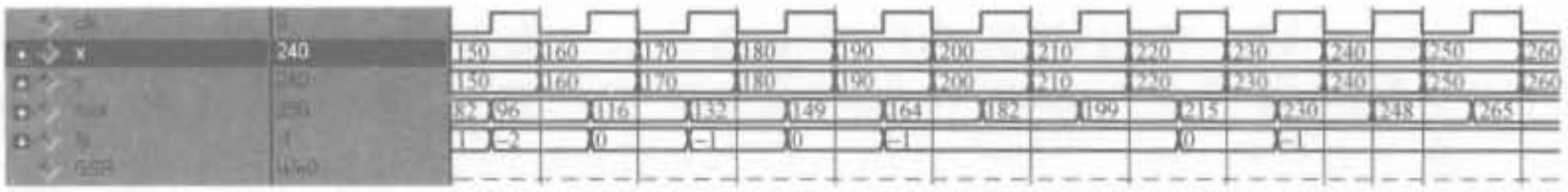


图 2-35 CORDIC 算法的仿真结果示意图

2.7 本章小结

本章简要介绍了 Verilog 硬件描述语言的发展历史和技术特点；然后介绍了相关的基本语法，这是进行 FPGA 设计所必须掌握的基础知识。读者如果想要了解 Verilog 语言更多的细节，可参考关于 Verilog 语言的专著。本章最后给出了典型的 Verilog 实例，包括基本的触发器、译码器设计，常用的算术运算模块以及高级的 CORDIC 算法模块，希望读者从中体会到 Verilog 的开发技巧，初步具备 Verilog 设计、开发的能力。

第3章

基于Xilinx芯片的HDL语言高级进阶

CHAPTER 3

Verilog 语言是一种国际标准，具有广泛的通用性。但是由于 FPGA 生产厂商和编译器提供商的多家性，决定了单纯的 Verilog 语言在某些特定场合下必然存在效率低下的情况。因此，在使用某一特定厂家的 FPGA 时，不仅需要熟练掌握 Verilog 语言，还需要了解其芯片结构、专用模块及其底层内嵌单元的使用方法。目前实用的 FPGA 设计方法是把 Verilog 看成一种胶合物，将芯片特有的组件融入其中，只有这样才能进行高效的开发，达到事半功倍的效果。本章主要介绍 Verilog 语言在 Xilinx FPGA 芯片开发中的高级应用。

3.1 面向硬件电路的设计思维

3.1.1 面向硬件的程序设计思维

面向硬件的设计思维，指的是如何将具体功能形成硬件的 RTL 级模型，然后选择具体的物理电路来实现，最后用合适的语言去实现，而不是凭空写 Verilog 代码。可以说，Verilog 是电路的一种适合管理的高级记录形式，而不是传统意义上的编程语言。

1. 程序的并行执行特点

和 C/C++等顺序编程语言不同，HDL 语言用于电路描述，代表着门电路和触发器电路。在任何时刻，只要 FPGA 上电，芯片内部各个模块将同时工作，不会因为各个模块带电的先后不同而存在执行顺序的差异。这需要设计人员以并行思维来考虑算法结构。

模块内部的执行顺序比较复杂，在优秀的设计中，模块内部是并行执行的，即使在“begin…end”语句内部。这里，不少读者可能存在疑问，“begin…end”语句不是串行执行的模块吗？下面对此进行详细解释。

“begin…end”之间存在阻塞赋值和非阻塞赋值两种赋值方式，如果使用不当，会有冒险和竞争现象，必须遵照下面两条准则[5]：①在描述组合逻辑的 always 块中使用阻塞赋值，则综合组合逻辑的电路结构；②在描述时序逻辑的 always 块中使用非阻塞赋值，则综合时序逻辑的电路结构。在组合逻辑中，阻塞赋值只与电平有关，往往和触发沿没有关系，可以将其看成是并行执行的；在时序逻辑中，非阻塞赋值是并行执行的。因此，对于优秀的 HDL 设计，其内部语句也是并行执行的。为了便于读者理解，给出例 3-1 予以说明。

例 3-1 给出 Verilog HDL 语句并行执行特点的实例说明。

```
module feizusefuzhi(clk,reset,a,b);
```

```
    input         clk;
    input         reset;
    input  [3:0] a;
    output [3:0] b;

    reg   [3:0] tempa1,tempa2,b;

  always @(posedge clk) begin
     if(! reset) begin
              tempa1<=0;
              tempa2<=0;
              b<=0;
     end
     else begin
           tempa1<=a+1'b1;
           tempa2<=tempa1+1'b1;
           b<=tempa2+1'b1;
     end
  end

endmodule
```

上述程序经过 Synplify 综合后，得到的 RTL 级结构图如图 3-1 所示。其中，对于信号 tempa2 和 b 的处理结构是一样的，是同时执行的，只是存在数据流到达先后的差异。

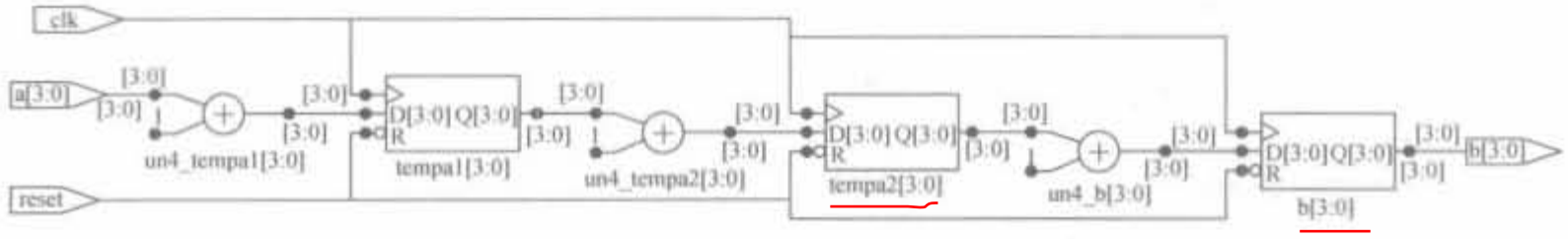


图 3-1　非阻塞赋值实例的 RTL 级结构示意图

在 ModelSim 6.2b 中完成仿真，其结果如图 3-2 所示。信号 tempa2 并不是等到本次“a+1”的执行结果赋给 tempa1 后才执行“tempa1+1”，而是在执行“a+1”的同时执行“tempa1+1”，其中的 tempa1 是上一次“a+1”的结果。同样的过程还可以在信号 b 的赋值中看到。因此，当 a 的值为 9 时，b 的值为 10，而不是 12。

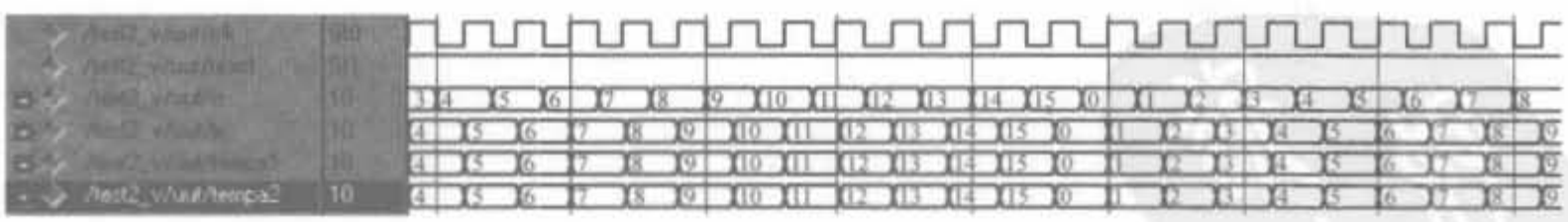


图 3-2　非阻塞赋值实例的仿真结果

2. 时钟是程序的执行控制器

上文解释了 HDL 程序的并行性，但在设计中需要像 C/C++语言的串行控制功能，如先接收外部配置指定，然后接收数据并完成模块内部配置，再将配置结果反馈到外部，这需要通过时间的精确定位来获取严格的先后关系。那么，怎么来实现呢？

其实很简单，假设全部事件需要 5 个时钟周期，那么利用一个周期为 5 的循环计数器来实现。在计数器为 1 的时候，完成事件 1；在计数器为 2 的时候，完成事件 2；……如此循环即可。总结起来，就是按照时钟节拍来完成串行控制。当然，这样的电路在 FPGA 资源的利用上是存在浪费的，因为在执行事件 1 时，用于执行事件 2、3、4、5 的逻辑处于等待状态，但它们始终占用着逻辑资源。

例 3-2 时钟执行控制器实例。

```
module clk_model(clk,reset,a1,y1,y2);
  input clk;
  input reset;
  input [7:0] a1;
  output [7:0] y1,y2;

  reg [7:0] y1,y2;
  reg [2:0] cnt;
  always @(posedge clk) begin
     if(!reset) begin
      cnt<=0;
      y1<=0;
      y2<=0;
     end
     else begin
     if(cnt==2'b11)
       cnt<=2'b00;
     else
       cnt<=cnt+1'b1;
    case(cnt)
     2'b00: begin
          y1<=a1+1'b1;
          y2<=y2;
     end
     2'b01: begin
          y1<=y1;
          y2<=y1+1'b1;
     end
     default: begin
          y1<=y1;
          y2<=y2;
     end
    endcase
  end
end

endmodule
```

上述程序经过 Synplify 综合后，得到的 RTL 级结构图如图 3-3 所示。可以看到，生成了 3 个加法器，即为变量 y1、y2 以及 cnt 的计算都生成了加法器。但 y1 与 y2 的加法器在时钟信号 clk 的控制下工作，其工作时间是不连续的，每 4 个周期计算 1 次，且 y1 的计算永远在 y2 的计算之前。

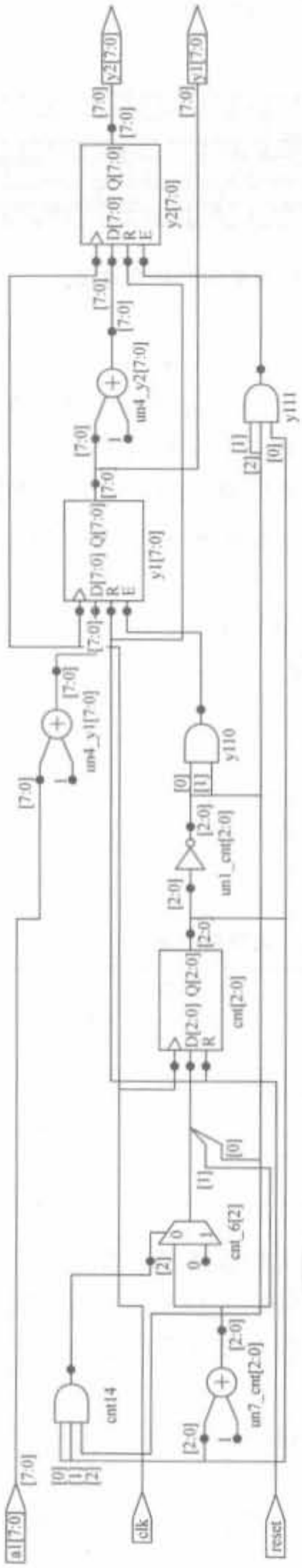


图 3-3 时钟控制实例的 RTL 级结构示意图

在 ModelSim 6.2b 中完成仿真，其结果如图 3-4 所示。从中可以看出，y1 和 y2 的周期是时钟的 4 倍。y1 的值等于 cnt 为 0 时所采样的输入信号 a1 的值加 1；y2 的值等于 y1 的值加 1，且比 y1 的值延迟了一个时钟周期。

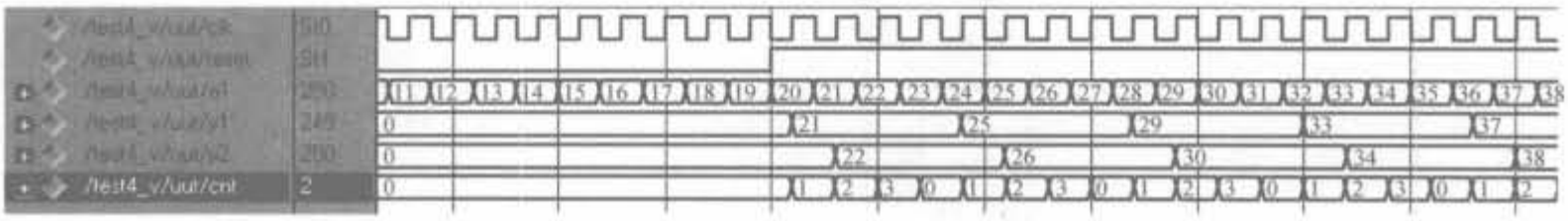


图 3-4　时钟控制实例的仿真结果

3. 程序的可综合性

IEEE 的 Verilog HDL 标准只是定义了 Verilog HDL 语言本身的规范，其本身是面向 HDL 仿真的，并不是所有用 Verilog HDL 语言写的程序都是硬件可综合的。Verilog 语言中可以被综合的语言成为可综合子集。每个厂商的综合工具所支持的可综合子集可能有所不同。下面以常用的综合工具 Synplify Pro 为例来说明可综合的 Verilog HDL 风格。

1）Synplify Pro 支持的 Verilog 结构

（1）线网类型：wire、tri、supply1、supplu0 等。

（2）寄存器类型：reg、interger、time 等。

（3）连续赋值。

（4）门原型和模块例化。

（5）always 模块、用户定义的 task 和 function。

（6）input、output 和 inout 端口。

（7）所有运算符，如表 3-1 所示。

表 3-1　运算符列表

运算符(说明)	运算符	运算符	运算符	运算符	运算符
+	<	!=	!	~|	>>
-	>	===	~	^~	?
(空格)	<=	!==	&	~^	{}
/	>=	&&	~&	^	{{}}
%	==	!!	|	<<	

其中，“/”和“%”运算符只适合于能得到整数结果的常量。由于运算符的优先级对于各个综合工具都不一样，为了解决这个问题，Silicon Integration 组织(www.si2.org)正在努力促进可综合风格的标准化。作者建议在有多个运算符的表达式中使用括号“()”来指定运算顺序。

（8）行为语句：如 if-else-if、case、casex、casez、for、repeat、while、forever、begin、end、fork、join。

（9）过程赋值(procedural assignments)：阻塞赋值“=”、非阻塞赋值“<=”(同一个寄存器不能同时使用“<=”和“=”)。

（10）编译器 directive：'define、'ifdef、'else、'endif、'include、'undef。

(11) 其他：在本地“begin…end”模块中，在赋值语句的左端和右端可以对向量进行可变参数的索引。

2) Synplify Pro 不支持的 Verilog 结构

(1) Net 类型：trireg、wor、trior、wand、triand、tri0、tri1、strength。

(2) register 类型：real。

(3) 单向和双向开关、pull-up、pull-down。

(4) 行为语句：deassign、wait。

(5) 命令的事件(Event)和事件触发。

(6) UDP 和 specify 模块。

(7) force、release 和层次化的 Net 名称(仅用于仿真)。

3) Synplify Pro 忽略的 Verilog 结构

Synplify Pro 遇到下面这些语言结构时不会中断综合进程，而是继续进行。

(1) delay，delay 控制，驱动强度。

(2) 标量化、向量化(scalared、vectored)。

(3) initial 模块。

(4) 除了 'define、'ifdef、'else、'endif、'include、'undef 之外的 directive。

(5) 对系统 task 和系统 function 的调用(仅用于仿真)。

3.1.2 “面积”和“速度”的转换原则

这里的“面积”主要是指设计所占用的 FPGA 逻辑资源数目，即利用所消耗的触发器(FF)和查找表(LUT)来衡量。“速度”是指在芯片上稳定运行时所能够达到的最高频率。面积和速度这两个指标始终贯穿着 FPGA 的设计，是设计质量评价的最终标准。本节主要讨论一个基本原则：面积和速度的互换。

面积和速度是一对对立和统一的矛盾体。一方面，要提高速度，需要消耗更多的资源，即需要更大的面积；另一方面，为了减少面积，需要降低处理速度。所以，既要提高速度，又要减少面积，是不可能同时实现的。但在实际中，总是存在二者之间的平衡，意味着二者可以互换。

面积和速度互换的具体操作很多，比如模块复用、乒乓操作、串/并变换以及流水线操作。本节对面积和速度互换只作简略的介绍，具体应用及方法将在后面的章节中逐步提及。其中，流水线的操作比较重要，将在下一节详细介绍。这里主要说明在串/并转换中利用这一互换原理进行设计。

例 3-3 给出一个实际开发中串/并转换的示例。

如图 3-5 所示，假设数据速率是乘法器模块处理速度的 3 倍，而乘法器模块的数据吞吐量满足不了要求。在这种情况下，利用面积换速度的思想，复制 3 个乘法器模块。首先将输入数据进行串/并转换，然后利用这 3 个模块并行处理所分配到的数据，最后将处理结果并/串转换，达到数据速率的要求。这个例子只是对面积换速度思想的一个简单举例，在具体操作过程中还涉及很多方法和技巧，需要读者从大量实例中自己体会。在后续章节中，我们还会逐步地应用这一互换思想。

图 3-5　串/并转换的示意图

3.1.3　同步电路的设计原则

1. 同步电路的定义

同步电路和异步电路的区别在于电路触发是否与驱动时钟同步。从行为上讲，就是所有电路是否在同一时钟沿的触发下同步地处理数据。常用于区分二者的典型电路就是同步复位和异步复位电路，其示例代码如表 3-2 所示。

表 3-2　同步复位电路和异步复位电路的代码比较

同步复位代码	异步复位代码

module test(clk,reset,s);
input clk;
input reset;
output s;
reg s;
 	
always@(posedge clk) begin
if(!reset)
s<=0;
else
s<=s+1;
end
 	
endmodule
	module test(clk,reset,s);
	input clk;
	input reset;
	output s;
	reg s;
	
	always@(posedge clk or posedge reset) begin
	if(!reset)
	s<=0;
	else
	s<=s+1;
	end
	
	endmodule

在同步复位的代码中，always 语句只有 posedge clk 一个触发条件；但是异步复位代码中的 always 语句有 posedge clk 和 posedge reset 两个触发条件。如果外部来了低电平的复位信号 reset，同步电路必须要等到 clk 的上升沿才能接收外部信号，而异步电路能立刻触发。

在实际系统中，常存在多时钟的情况，这时应该延伸同步电路的理念，使设计做到局部同步，即同一时钟驱动的电路要同步于其同一时钟沿(上升沿或下降沿)。

2. 同步电路的优点

同步设计主要有以下 3 个优点：

(1) 可以有效避免毛刺的影响，提高设计可靠性。毛刺是数字电路的天敌，只要有逻辑

电路，就会有毛刺发生，是永远存在的。因此，优秀的设计都必须从如何避免毛刺对设计的不良影响入手，提高设计的稳定性。同步设计是避免毛刺影响的最简单方法，其原理如例 3-4 所示。

例 3-4　异步电路缺点实例。

```
module as_model(clk,a1,a2,y,yout);
    input  clk;
    input  a1,a2;
    output y,yout;
    reg    yout;

    assign y = a1 & a2;

    always @(posedge clk) begin
      yout<= y;
    end

endmodule
```

上述程序经过 Synplify 综合后，得到的 RTL 级结构图如图 3-6 所示。yout 信号为 y 信号级联 D 触发器的输出结果，意味着对 y 信号进行了同步采样。

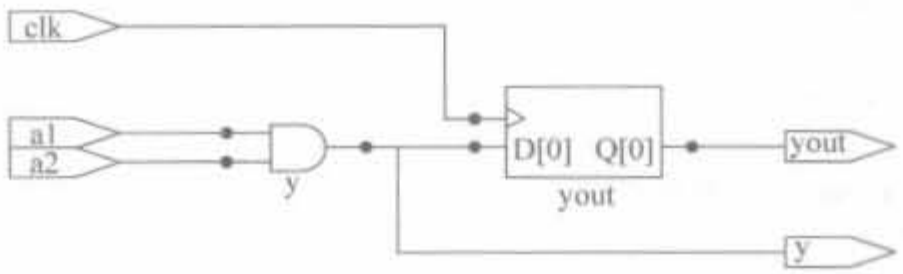


图 3-6　同步、异步比较实例的 RTL 级结构示意图

在 ModelSim 6.2b 中完成仿真，其结果如图 3-7 所示。由于 a1、a2 信号到达与门时间的不一致性，导致信号 y 产生了毛刺。经过同步处理后，有效地消除了毛刺，这正是同步电路的优点。

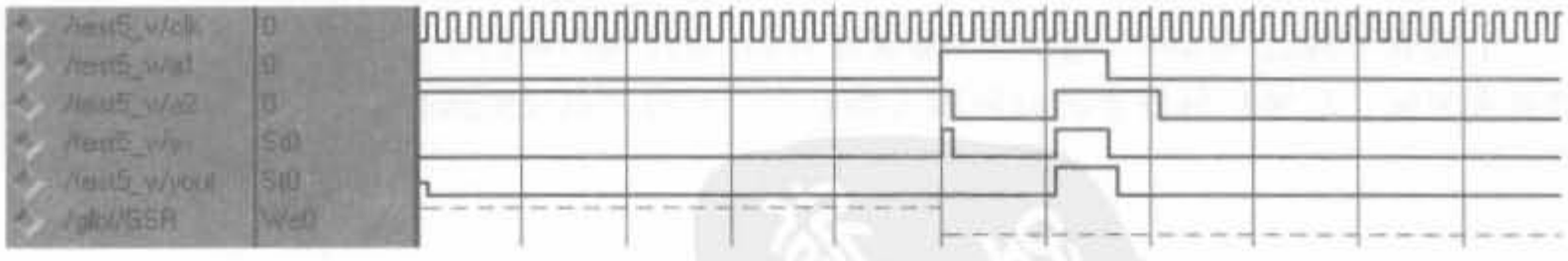


图 3-7　同步、异步比较实例的仿真结果

（2）可以简化时序分析过程。时序分析是高速数字设计的重要话题，本书第 11 章将对其进行详细讨论。

（3）可以减少工作环境对设计的影响。异步电路受工作温度、电压等影响，器件延时变化较大，异步电路时序将变得更加苛刻，导致芯片无法正常工作。同步电路只要求时钟和数据沿相对稳定，时序要求较为宽松，因此对环境的依赖性较小。

3. 同步电路的准则

1）单时钟策略

尽量在设计中使用单时钟，且走全局时钟网络。在单时钟设计中，很容易将整个设计同步于驱动时钟，使设计得到简化。全局时钟网络的时钟是性能最优，最便于预测的时钟，它具有最强的驱动能力，不仅能保证驱动每个寄存器，而且时钟漂移可以忽略。在多时钟应用中，要做到局部时钟同步。

在实际工程中，应将时钟信号和复位信号通过 FPGA 芯片的专用全局时钟管脚送入，以获得更高质量的时钟信号。全部时钟的调用方法将在第 11 章详细说明。

2）单时钟沿策略

尽量避免使用混合时钟沿来采样数据或驱动电路。使用混合时钟沿将会使静态时序分析复杂，并导致电路工作频率降低。下面给出利用混合时钟沿采样数据而降低系统工作时钟的实例。

例 3-5 混合时钟沿采样的分析实例。

```
module hunhe(clk,din,d1,dout);
    input  clk;
    input  [7:0] din;
    output [7:0] d1;
    output [7:0] dout;

    reg [7:0] d1,dout;

    always @(negedge clk) begin
        d1<=din;
    end

    always @(posedge clk) begin
        dout<=d1;
    end

endmodule
```

上述程序经过 Synplify 综合后，得到的 RTL 级结构图如图 3-8 所示。比较两个 D 触发器就会发现：左端 D 触发器的时钟输入端有一个对时钟取反的操作。

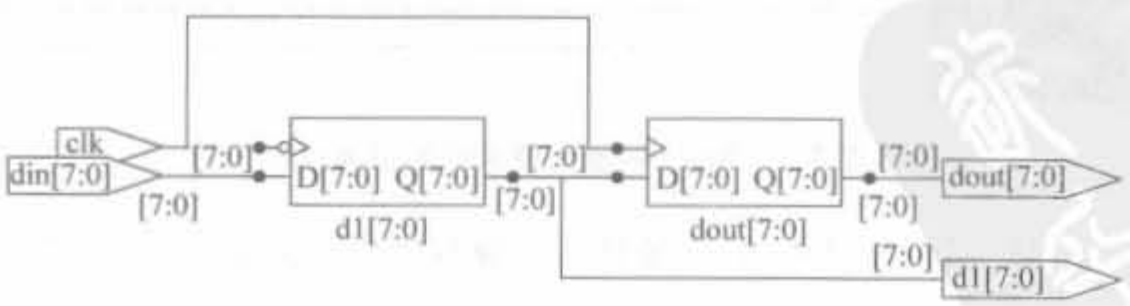


图 3-8 混合时钟沿采样实例的 RTL 级结构示意图

在 ModelSim 6.2b 中完成仿真，其结果如图 3-9 所示。从中可以看出，信号 d1 对数据采样错误，存在顺序颠倒的现象。这是由采样建立、保持时间不够而引起的，意味着在给定

的时钟频率下，电路无法正常工作。

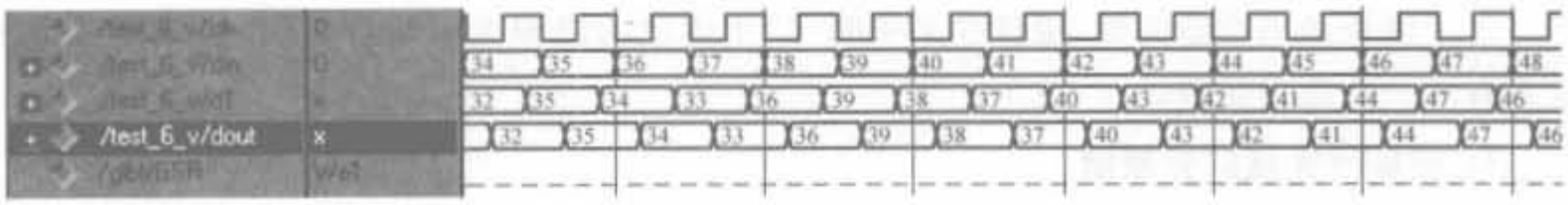


图 3-9　混合时钟沿采样实例的仿真结果

3）避免使用门控时钟

如果一个时钟节点由组合逻辑驱动，那么就形成了门控时钟，如图 3-10 所示。门控时钟常用来减少功耗，但其相关的逻辑不是同步电路，即可能带有毛刺，而任何的一点点小毛刺都可以造成 D 触发器误翻转。此外，门控逻辑会污染时钟质量，产生毛刺，并恶化偏移和抖动等指标。门控时钟对设计的可靠性有很大影响，应尽可能避免。不要为了节省功耗去使用门控时钟。最近发展起来的用于减少功耗的方法是低核电压 FPGA、FPGA 休眠技术以及动态部分重构技术等，有兴趣的读者可以深入阅读文献[6]。

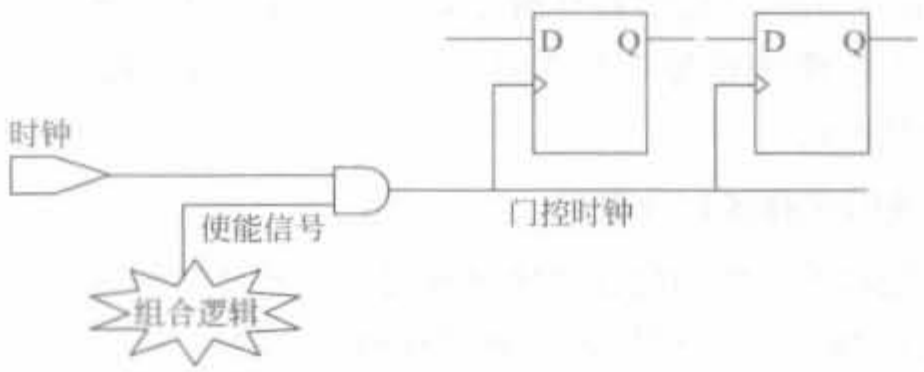


图 3-10　门控时钟示意图

4）禁止在模块内部使用计数器分频产生所需时钟

各个模块内部各自分频会导致时钟管理混乱，不仅使得时序分析变得复杂，产生较大的时钟漂移，浪费了宝贵的时序裕量，降低了设计可靠性，而且其功能可以通过时钟使能电路实现。

3.1.4　模块划分的设计原则

自顶向下的层次化设计方法中最关键的工作就是模块划分，即将一个很大的工程合理地划分为一系列功能独立的部分，且具备良好的协同设计能力，以便快速地实现整个设计。此外，模块划分直接影响到所需的逻辑资源、时序要求以及实现效率。模块划分的基本原则如下所述。

1. 信息隐蔽、抽象原则

上一层模块只负责为下一层模块的工作提供原则和依据，并不规定下层模块的具体行为，以保证各个模块的相对独立性和内部结构的合理性，使得模块之间层次分明，易于理解、实施和维护。

2. 明确性原则

每个模块必须功能明确，接口含义明确，禁止多重功能和无用接口，在整个设计过程中应具有统一的命名规范。

3. 模块时钟域区分原则

在设计中，经常采用多时钟设计，必然存在亚稳态，如果处理不当，将会给设计的可靠性带来极大的隐患。这里需要通过异步 FIFO 以及双口 RAM 来建立接口，尽量避免让信号直接跨越不同的时钟域。此外，由于时钟频率不同，其时序约束需求也不同，可以将低频率时钟域划分到同一模块，如多时钟路径等，让综合器尽量节约面积。

4. 资源复用原则

在 HDL 设计中，要将可以复用的逻辑或者相关逻辑尽量放在同一模块中，这不仅节省硬件资源，还有利于优化关键路径。但在实际中，不能为了资源复用而将存储器逻辑混用。因为 FPGA 芯片生产商提供了各类存储器的硬件原语，尽量使用原语，而不是使用查找表和寄存器来实现原语的功能，才能将设计所需资源最小化。从概念上讲，模块越大越利于资源共享和复用，但庞大的模块在仿真验证时需要较长的时间和较高的 PC 配置，不利于修改，无法使用增量设计模式。

5. 同步时序模块的寄存器划分原则

在设计时，应尽量将模块中的同步时序逻辑输出信号以寄存器的形式送出，以便于综合工具区分时序和组合逻辑；并且时序输出的寄存器应符合流水线设计思想，能工作在更高的频率，以极大地提高模块吞吐量。

3.2 优秀的 HDL 代码风格

目前，FPGA 的规模越来越大，HDL 代码的功能越来越复杂，规模也越来越大，代码的可移植性以及时序、资源等指标的要求也越来越高，并且设计的稳定性越来越被关注。与此同时，EDA 越来越智能化，同一个程序经过不同的工具分析后可能会产生不同的结果。因此，代码的书写风格极大地影响着设计。优秀的代码风格可以减少错误，提高电路性能，达到事半功倍的效果。如果使用不当，会有南辕北辙的后果。

3.2.1 代码风格的含义

代码风格有两层含义：其一是 Verilog 的代码书写习惯；另一个则是对于一个特定电路，用哪一种形式的语言描述，才能将电路描述得更准确，使得综合以后产生的电路更为合理。前者在 2.5 节已有涉及，本节主要介绍后者。

代码风格有通用风格和专用风格两大类。前者指不依赖于 FPGA 开发的 EDA 软件工具和 FPGA 芯片类型，它仅仅是从 HDL 语言出发的代码风格；后者指和开发软件以及硬件芯片密切相关的代码风格。此时不仅需要关注 EDA 软件在语法细节上的差异，还要紧

密依赖于固有的硬件结构。显然，前者具有较好的通用性，但性能未必最优。在使用时，如果有后续的进一步开发，建议使用通用风格；否则采用后者，以便极大地挖掘芯片潜力。

3.2.2　通用代码风格的介绍

1. 逻辑复用与逻辑复制

在 3.1.2 节介绍过 FPGA 设计中面积和速度的转化关系，该理念始终贯穿 HDL 代码设计，二者典型的转换方式有逻辑复用和逻辑复制。前者通过速度换面积，后者通过面积换取速度，两者各有相应的应用范围。

1）逻辑复用

逻辑复用是通过提高工作频率来节省面积的优化方法，经常用于存在多个资源可共享单元的设计中，是大规模 FPGA 设计的核心思想。为了便于理解，首先给出一个例子。

例 3-6　幅度到功率转化模块中的逻辑复用实例。假设系统输入信号为 I、Q 两路，每路速率为 5Mb/s、位宽为 16bit，分别给出普通实现和使用逻辑复用的实现代码。

功能分析：计算功率，需要先将 I、Q 输入信号经过平方计算模块，再将 2 个平方和相加输出。

普通实现方式如图 3-11 所示。

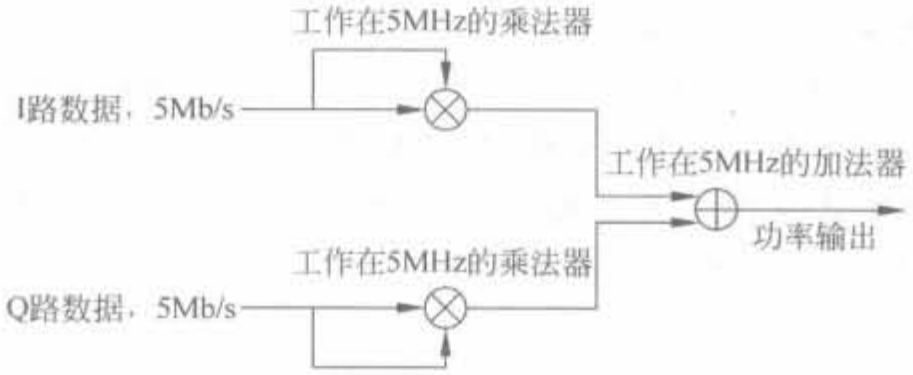


图 3-11　功率统计模块的一般实现方式

逻辑复用方式如图 3-12 所示。

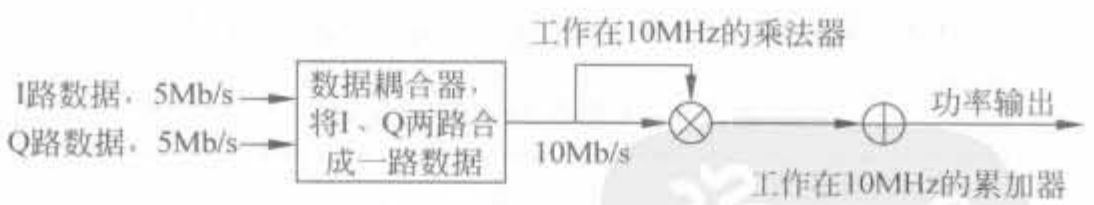
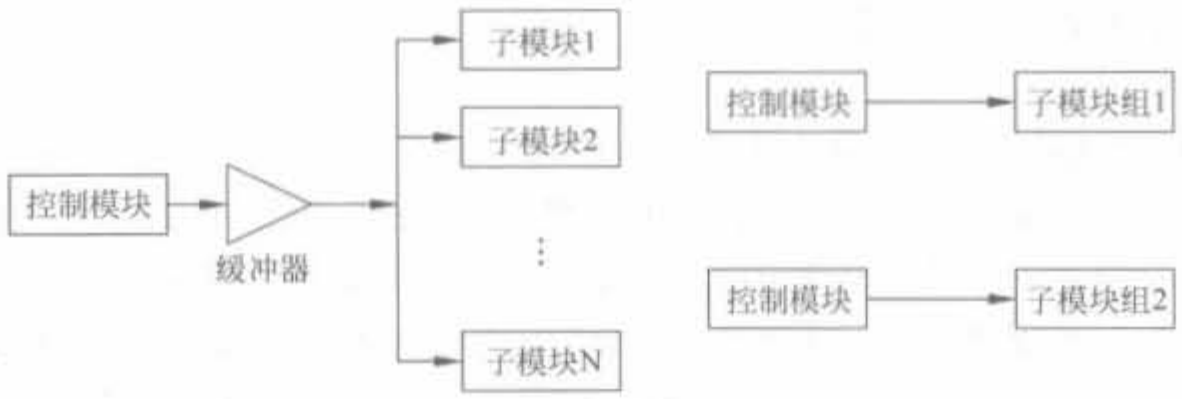


图 3-12　采用逻辑复用方式实现功率统计模块

经过比较可以发现，第 1 种实现方式需要两个 16bit 乘法器；第 2 种实现方式只需要 1 个乘法器，基本上将资源缩减到第 1 种方式的一半。目前，虽然有很多综合工具可以提供逻辑复用的选项，但不能因为此而放松对代码编程的要求，主要原因有两点：首先，EDA 工具的能力毕竟有限，在很多情况下不能智能发现可以复用的逻辑；其次，不同综合工具的优化参数以及能力并不相同，所以综合结果对于设计者是不确定的。因此，逻辑复用主要还是通过代码体现。

2）逻辑复制

逻辑复制是通过增加面积而改善设计时序的优化方法，经常用于调整信号的扇出。如果信号具有高的扇出(如时钟和复位信号等)，即要驱动很多后续电路，则要添加缓存器来增强驱动能力，但这会增大信号的延时。通过逻辑复制，使用多个相同的信号来分担驱动任务，每路信号的扇出就会变低，就不需要额外的缓冲器来增强驱动，可减少信号的路径延迟。例如，用于产生控制信号的监控模块一般都有高的扇出，这时就需要考虑逻辑复制这一功能。图 3-13(a)给出了未采用逻辑复制的设计模式，其占用资源较少，但延迟大，容易出错；而采用逻辑复制的设计如图 3-13(b)所示，它延迟小，但占用的资源多。



(a) 未采用逻辑复制的设计模式　　(b) 采用逻辑复制的设计模式

图 3-13　逻辑复制示例

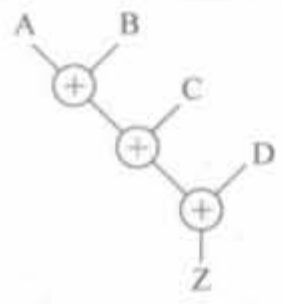
逻辑复用和逻辑复制是资源与速度的对立统一，目的都是提高设计性能、达到设计目标，但一个是为了节省面积来提高速度，另一个却是为了提高速度来占用额外的面积，两者之间存在转换和平衡的关系。在实际工程中，经常可以看到这两种方法的应用。在 XST 以及 Synplify Pro 综合工具中，用户可以设定最大扇出数。当某信号的扇出超过最大扇出值时，该信号会自动被综合工具复制，以降低扇出。

2. 逻辑结构

逻辑结构主要分为链状结构(Chain Architecture)和树状结构(Tree Architecture)。一般来讲，链状结构具有较大的延时，后者具有较小的延时。链状结构主要指程序是串行执行的，树状结构是串并结合的模式，具体如例 3-7 所示。

例 3-7　表 3-3 给出了具有链状结构和树状结构的 4 输入加法器的实现。

表 3-3　4 输入加法器的实现

	链状结构	树状结构
程序描述	Z<=A+B+C+D;	Z<=(A+B)+(C+D);
RTL 综合图		
性能	具有 3 个延迟	在占用同等数量的资源下，具有 2 个延迟

从例 3-7 可以明显看出树状结构的优势，它能够在同等资源的情况下缩减运算延时，从而提高电路吞吐量，以节省面积。

3. if 和 case 语句的使用原则

1）if 和 case 语句的区别

if 语句指定了一个有优先级的编码逻辑，而 case 语句生成的逻辑是并行的，不具有优先级。if 语句可以包含一系列不同的表达式，而 case 语句比较的是一个公共的控制表达式。通常 if-else 结构速度较慢，但占用的面积小，如果对速度没有特殊要求而对面积有较高要求，可用 if-else 语句完成编解码。case 结构速度较快，但占用面积较大，所以用 case 语句实现对速度要求较高的编解码电路。嵌套的 if 语句如果使用不当，就会导致设计的更长延时。为了避免较大的路径延时，最好不要使用特别长的嵌套 if 结构。如想利用 if 语句来实现那些对延时要求苛刻的路径，应将最高优先级给最迟到达的关键信号。有时为了兼顾面积和速度，可以将 if 和 case 语句合用。

2）if 和 case 实例

下面分别给出两个使用 if 和 case 语句的实例，希望读者从中体会到二者的不同。

例 3-8　使用 if 语句实现一个 4 选 1 的数据通路选择器。

```
module sdata_if(clk,reset,x,s,y);
  input clk;
  input reset;
  input [3:0] x;
  input [1:0] s;
  output y;

  reg y;
  always @(posedge clk) begin
    if(!reset) begin
      y<=0;
    end
    else begin
      if(s == 2'b00)
            y<=x[0];
        else if (s == 2'b01)
            y<=x[1];
        else if (s == 2'b10)
            y<=x[2];
        else
            y<=x[3];
    end
end

endmodule
```

上述程序经过综合后，其中数据选择部分的 RTL 级结构如图 3-14 所示。从中可以看出，状态变量 s[1:0]通过 y13、y14、y15 以及 y16 是串行输入到复用器中的，且具有严格的逻辑顺序，其逻辑级数为 4。

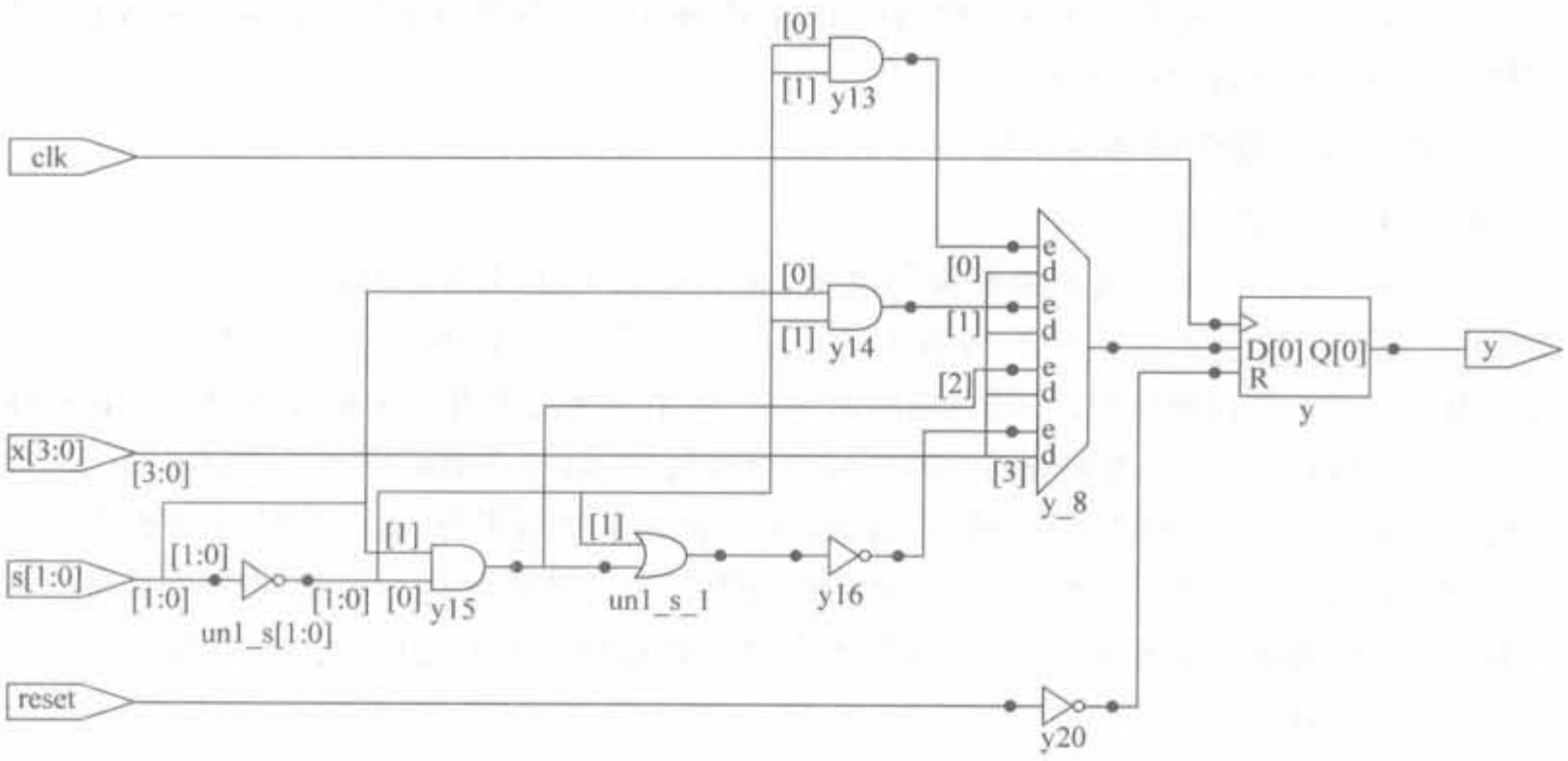


图 3-14　用 if 语句实现的 4 选 1 选择器 RTL 级结构图

例 3-9　使用 case 语句实现一个 4 选 1 的 8 位数据选择器。

```
module sdata_case(clk,reset,x,s,y);
  input clk;
  input reset;
  input [3:0] x;
  input [1:0] s;
  output y;

  reg y;
  always @(posedge clk) begin
    if(!reset) begin
        y<=0;
    end
    else begin
        case(s)
            2'b00: y<=x[0];
            2'b01: y<=x[1];
            2'b10: y<=x[2];
            2'b11: y<=x[3];
          endcase
      end
  end

endmodule
```

上述程序经过综合后，其中数据选择部分的 RTL 级结构如图 3-15 所示。从中可以看出，状态变量 s[1:0]通过 y13、y14、y15 以及 y16 是并行输入到复用器中的，因此逻辑级数只有 1 级。

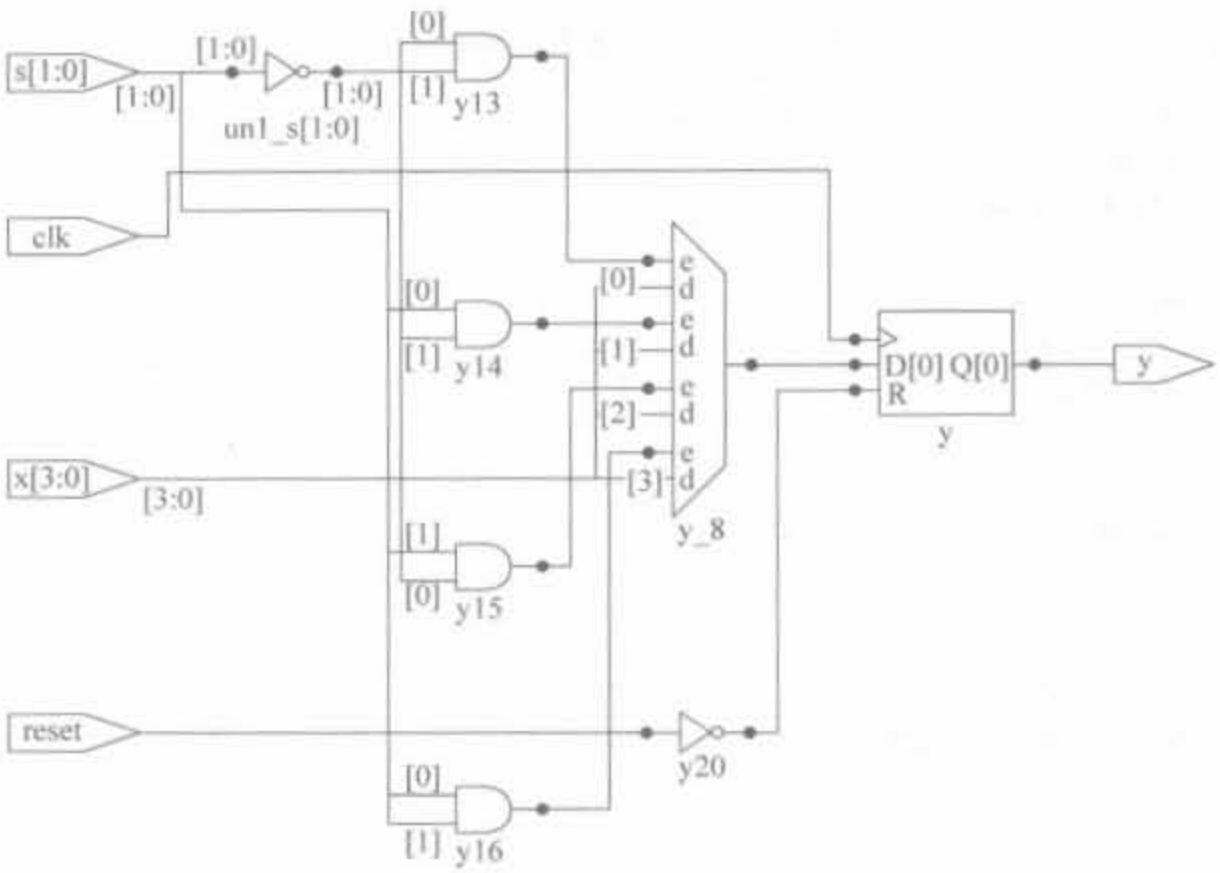


图 3-15　用 case 语句实现的 4 选 1 选择器 RTL 级结构图

4. 关键路径信号处理原则

在 Verilog 设计中，经常会遇到由于信号路径过长或信号来得比较晚，从而造成建立时间不够的情况。这种引起电路建立时间不够的信号路径就称为关键路径。在复杂电路设计中必须有效地处理关键信号，尽量减少其延时，提高电路的工作频率。

1）简单组合电路关键路径的提取方法

简单组合电路关键路径的提取方法就是拆分逻辑，将复杂逻辑变成多个简单组合电路的进一步组合，缩减关键信号的逻辑级数，如例 3-10 所示。

例 3-10　简单关键电路的提取实例。

对于语句

```
assign y = a & b & c|d & e & b;
```

信号 b 为关键信号。先计算其简单路径，再经过关键路径逻辑。

```
assign temp = a & b & c & d;
    assign y = b & temp;
```

通过关键路径提取，将信号 b 的路径由 2 级变成 1 级。拆分逻辑的方法就是布尔逻辑扩展，也被称为香农扩展，其原理如下式所示：

$$F(x,y,z) = xF(1,y,z) + \bar{x}F(0,y,z)$$

可以看出，拆分逻辑可通过复制逻辑，缩短那些组合路径长的关键信号的路径延时，从而提高工作频率。

2）复杂 always 块中关键路径的提取方法

对于 always 模块中时间要求非常紧的信号，需要通过分步提取方法，让关键路径先行，保证改写后的描述与原 always 块逻辑等效。例 3-11 给出了提取并改善 always 模块中关键信号的实例。

例 3-11 always 块中关键路径的提取和优化实例。

```
always@(w or x or y or z or in1 or in2) begin
    if(!w) begin
        if(x&&(!(y&&z)))
            out = in1;
        else
            out = in2;
    else if(y&&z)
        out = in1;
    end
    else begin
        out = out;
    end
end
```

其中，若 z=0，则原代码等效于

```
if (!w) out = in1;
else out = out;
```

若 z=1，则源代码等效于

```
if (!w && x &&!y ) out = in1;
else if(!w && x && y) out = in2;
else out = out;
```

对于信号 y 也有类似的分析结果，因此 y、z 都是关键信号，可通过首先计算关键路径进行优化。上述指令代码改写为：

```
always@(w or x or y or z or in1 or in2) begin
    temp = y && z;
    if(!temp) begin
        if(x&&!w)
            out = in1;
        else
            out = in2;
    else if(temp)
        out = in1;
    end
    else begin
        out = out;
    end
end
```

5. 避免出现意外锁存器

锁存器是电平触发的存储器，触发器是边沿触发的存储器。在同步电路设计中要尽量避免出现锁存器。在 Verilog HDL 设计中，很容易由于条件判断语句表述得不完整，而造成设计中出现意想不到的锁存器。本质上，锁存器和 D 触发器的逻辑功能是基本相同的，都可存储数据，且锁存器的资源更少，具备更高的集成度。但锁存器对毛刺敏感，不能异步复位，因此在上电后处于不确定的状态。此外，锁存器还会使静态时序分析变得非常复杂，不具备可重用性。在 FPGA 芯片中，基本的单元是由查找表和触发器组成的，若生成锁存

器，反而需要更多的资源。因此，在设计中需要避免产生锁存器。这一事件具体可分为两种情况：其一是在 if 语句中，另一种情况是在 case 语句中。下面将对 if 和 case 语句造成的锁存器分别进行分析。

1）由 if 语句造成的锁存器

例 3-12 给出了在 always 块中使用 if 语句，但缺乏 else 分支而造成锁存器的情况。

例 3-12　表 3-4 给出了由于 if 语句不完整而生成意外的锁存器的示例。

表 3-4　if 语句不完整的情况

生成意想不到的锁存器	无锁存器

always@(a or data_in) begin
  if(a) begin
  data_out = data_in;
  end
end
	always@(a or data_in) begin
	  if(a) begin
	  data_out = data_in;
	  end
	  else begin
	  data_out = 0;
	  end
	end

经过 Synplify Pro 综合后，得到的 RTL 级结构图分别如图 3-16(a)以及(b)所示。

(a) 存在锁存器的RTL级结构　　(b) 不存在锁存器的RTL级结构

图 3-16　if 语句生成锁存器的 RTL 级结构示意图

从表 3-4 可以看出，左边的语句块只有在 a 的值为 1 的情况下，data_in 的值才能传递给 data_out，但没有指定 a 值为 0 的情况下 data_out 的取值。这样，在 always 语句块中，如果没有改变变量的赋值，变量值将保持不变，生成锁存器。如果希望 a＝0 时，data_out 值为 0，那么程序应如右例所示。本例说明，利用 else 分支不会生成锁存器。

2）由 case 语句造成的锁存器

例 3-13 给出了在 always 块中使用 case 语句，由于缺乏 default 分支而造成锁存器的情况。

例 3-13　表 3-5 给出了由于 case 语句不完整而生成意外的锁存器的示例。

表 3-5　case 语句不完整的情况

生成意想不到的锁存器	无锁存器

always@(a[1:0] or data_in1 or data_in2) begin
  case(a)
  2'00: data_out = data_in1;
  2'01: data_out = data_in2;
  endcase
end
	always@(a[1:0] or data_in1 or data_in2) begin
	  case(a)
	  2'00: data_out = data_in1;
	  2'01: data_out = data_in2;
	  default: data_out = 0;
	  endcase
	end

在表 3-5 中，左边的例子中，当 a[1:0]的值为 2'00、2'01 时，分别将 data_in1 或 data_in2 赋给 data_out，在 a 为其余值的时候就生成了锁存器，data_out 保持上一次的赋值不变；右边的例子比较明确，在 a 的值不等于 2'00、2'01 时，data_out 的值为 0，不会生成锁存器。

以上两个例子表明了如何避免生成意外的锁存器。即如果用到 if 语句，最好有 else 分支；如果用到 case 语句，最好有 default 语句。即使需要锁存器，也通过 else 分支或 default 分支来显式说明。按照上面的建议，可以避免意想不到的错误，提高程序的稳健性和可读性。

6. 流水线技术的使用

1）流水线技术的工作原理

流水线能动态地提升器件性能，它的基本思想是对经过多级逻辑的长数据通路进行重新构造，把原来必须在一个时钟周期内完成的操作分成在多个周期内完成。这种方法允许更高的工作频率，因而提高了数据吞吐量。因为 FPGA 的寄存器资源非常丰富，所以对 FPGA 设计而言，流水线是一种先进的而又不耗费过多器件资源的结构。但是采用流水线后，数据通道将会变成多时钟周期，所以要特别考虑设计的其余部分，解决增加通路带来的延迟。

流水线设计的结构示意图如图 3-17 所示。

(a) 没有流水线的结构示意图　　(b) 流水线的结构示意图

图 3-17　流水线结构示意图

流水线的基本结构是将适当划分的 N 个操作步骤串联起来。流水线操作的最大特点是数据流在各个步骤的处理，从时间上看是连续的；其操作的关键在于时序设计的合理安排，前、后级接口间数据的匹配。如果前级操作的时间等于后级操作的时间，直接输入即可；如果前级操作时间小于后级操作时间，则需要对前级数据进行缓存，然后才能输入到后级；如果前级操作时间大于后者，则需要用串/并转换等方法进行数据分流，然后再输入到下一级。

2）流水线技术实例

例 3-14　使用 Verilog 实现具有 4 级流水线结构的 8 位加法器。

```
module adder8_4(cout,sum,clk,cina,cinb,cin);
    input [7:0]cina,cinb;
    input clk,cin;
    output [7:0] sum;
    output cout;
    reg cout;
    reg cout1,cout2,cout3;
    reg [2:0]sum1;
    reg [4:0] sum2;
    reg [6:0] sum3;
    reg [7:0]sum;

    always @(posedge clk) begin // 低 2 位相加:
```

```
    {cout1,sum1} = cina [ 1: 0 ] + cinb [ 1: 0 ] + cin;
    end

    always @(posedge clk) begin  //相加,并且将低 4 位拼接起来;
        {cout2,sum2} = {{cina[3],cina[3: 2 ]} + {cinb[3],cinb[3: 2 ]} + cout1,sum1};
    end

    always @(posedge clk) begin //相加,并且将低 6 位拼接起来;
        {cout3,sum3} = {{cina[5],cina [ 5: 4 ]} + {cinb[5],cinb [ 5: 4 ]} + cout2,sum2};
    end

    always @(posedge clk) begin //高 2 位相加,并且将 8 位拼接起来;
        {cout,sum} = {{cina[7],cina [7: 6 ]} + {cinb[7],cinb[7: 6 ]} + cout3,sum3};
    end

endmodule
```

图 3-18 给出了 Synplify 综合后的 RTL 级结构图,其中有 4 个 2 位加法器,添加了较多的辅助逻辑,使加法的运算延时降低到对 2 个比特的处理,使之可以应用在高速设计中。

3.2.3 专用代码风格的简要说明

专用代码风格是指从 FPGA 器件特征的角度考虑,尽可能利用芯片结构以及内嵌的底层宏单元,以取得最佳的综合和实现效果。对于同一个设计,使用适合于 FPGA 体系结构特点的优化设计方法,可以大大提高芯片利用率和设计实现速度。

1. Xilinx FPGA 的体系结构特点

Xilinx FPGA 芯片的 3 种可构造单元:①可编程输入、输出块 IOB,主要为逻辑阵列与外部芯片管脚之间提供一个可编程接口;②可编程逻辑块 CLB,主要由一个组合逻辑、几个触发器、若干个多选一电路和控制单元组成;若干个 CLB 有规则地组成 FPGA 逻辑单元阵列结构,以完成用户指定的逻辑功能;③各种连线资源,包括可编程的开关矩阵,内部连接点和金属线。它们位于芯片内部的逻辑块之间,经编程后形成连线网络,以连接芯片内的逻辑块及传递逻辑信息。优秀的设计应在芯片架构的基础上,合理使用组合逻辑、触发器以及块 RAM 等内迁硬核单元。

组合逻辑功能通过用户可编程的查找表实现。查找表是由静态存储器构成函数发生器,在此基础上增加触发器形成的,它是既可实现组合逻辑功能,又可实现时序逻辑功能的基本逻辑单元电路。SRL16 是 Xilinx 器件中独有的一种移位寄存器查找表,有 4 个输入用来选择输出序列的长度,能够以极少的硬件资源实现数据缓存和组合逻辑。

每个 CLB 中包含两个触发器,CLB 的组合逻辑功能较少,触发器资源十分丰富。每个 CLB 中包含一个高速进位逻辑。专用进位电路速度远远大于采用传统的加速方法所能增加的速度。算术进位逻辑为有关算术运算中许多新的应用问题提供了有效的解决途径。

同时,Xilinx 提供了片上 RAM,特别是大量块 RAM,可以配置成双口 RAM 或 ROM,它们存储量大、速度快,且不占用逻辑资源,在设计实现中有着广泛的应用。

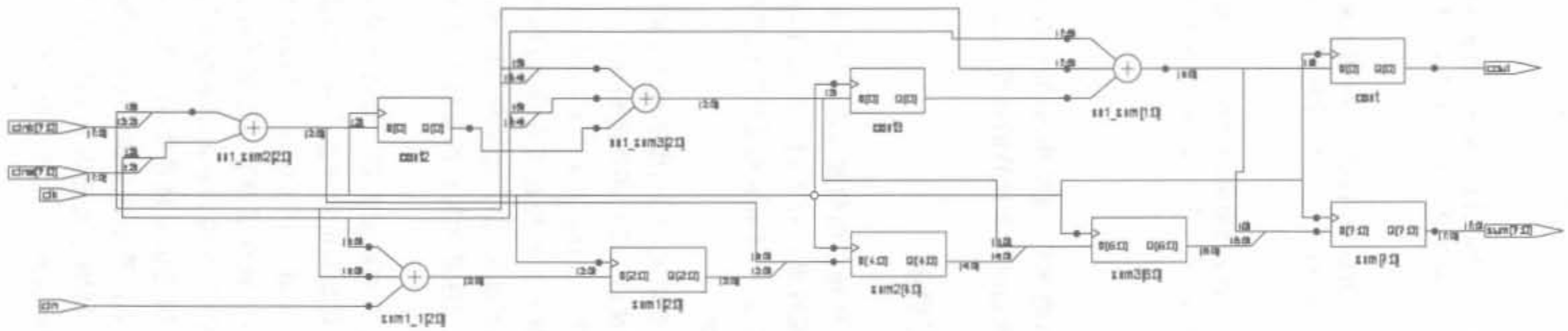


图 3-18　具有 4 级流水线结构加法器的 RTL 级结构图

内嵌的宏单元包括硬核乘加器、硬核处理器、数字时钟处理模块以及高速串行接口，其处理能力强，为片上最高，且不存在时序问题。如果合理地利用宏单元，可达到事半功倍的效果。

2. Xilinx FPGA 芯片专用代码风格

1）时钟信号的分配策略

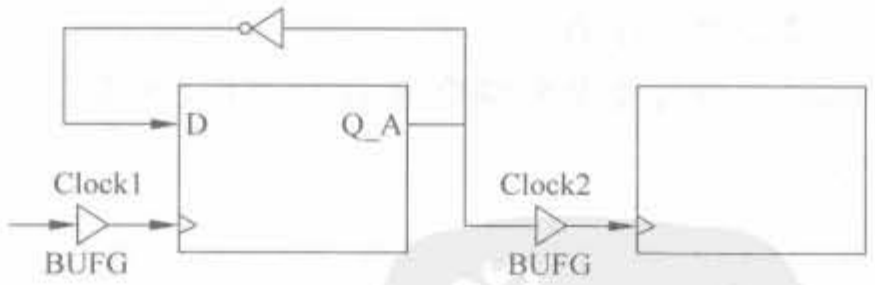
时钟分配网络是 FPGA 芯片中的特殊布线资源，由特定的管脚和特定的驱动器驱动，只能驱动芯片上触发器的时钟输入端或除了时钟输入端外有限的一些负载，其目的是为设计提供小延时偏差和扭曲可忽略的时钟信号。

首先，使用全局时钟，可为信号提供最短的延时和可忽略的扭曲。全局网线由全局缓冲器 BUFG 来驱动。使用 BUFG 时，时钟信号经 BUFG 驱动后，通过长线同时接到每个触发器的时钟端，减少传输延迟。如不使用 BUFG，时钟信号按一般布线连接到不同 CLB。时钟信号到达各触发器的延迟不一致，使同步时序电路出现不同步的现象。

其次，FPGA 特别适合于同步电路设计，尽可能减少使用的时钟信号种类。如在 TTL 电路设计中，经常采用的由组合逻辑生成多个时钟分别驱动多个触发器的设计方法对 FPGA 的设计不适用。因为这样做使得时钟种类很多，不能利用专用的时钟驱动器和专用的时钟布线资源，时钟信号只能由通用的布线资源拼凑而成，各个负载点上的时钟延迟偏差很大，会引起数据保持时间问题，降低了工作速度。

第三，减小时钟摆率的另一种更有效的方法是使用一个时钟信号，生成多个时钟使能信号，分别驱动触发器的时钟使能端。所有触发器的数据装入都由同一个时钟控制，但只有时钟使能信号有效的触发器才会装入数据，时钟使能信号无效的触发器则保持数据。这种方法充分发挥了 FPGA 器件体系结构的优势。

图 3-19 所示电路为分频器的一般设计和优化设计的对比。两种设计方法相比较，图(a)所示电路使用两个全局缓冲，实现两个触发器的异步控制。图(b)所示电路将异步控制转化为同步控制，只占用一个全局缓冲，利用时钟使能信号 CE 控制触发器的动作。因此，图(b)所示电路占用更少的全局时钟资源，而且使用一种时钟信号更利于同步控制和减小时钟摆率。

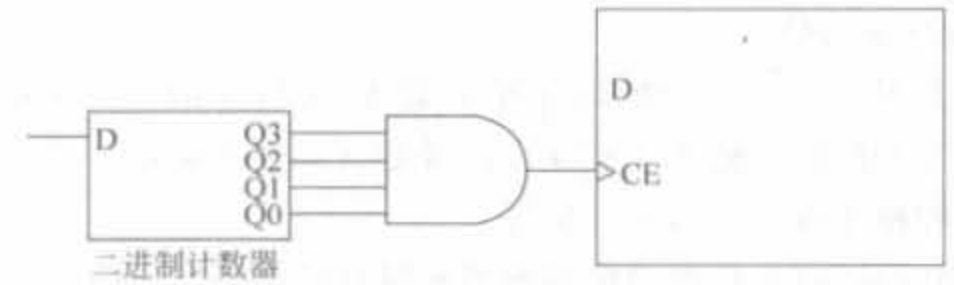


(a) 分频器的普通设计

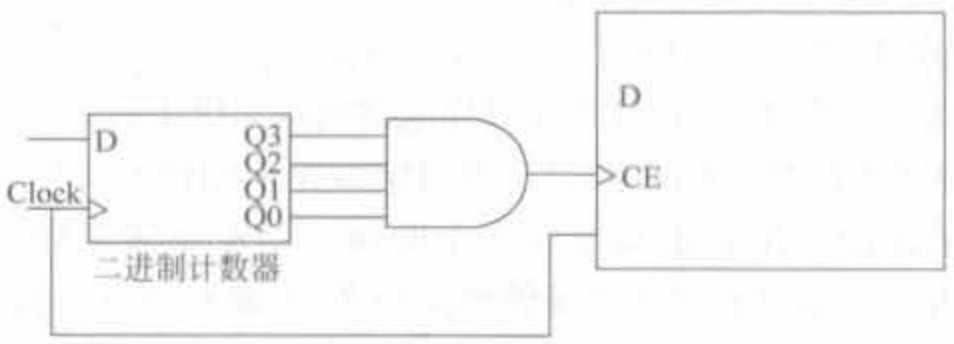
(b) 分频器的优化设计

图 3-19　分频器两种设计电路的比较

第四，要避免时钟信号毛刺。由图 3-20 所示电路可知，当二进制计数器从 0111 向 1000 变化时，必然会出现一个 1111 的过渡过程，D 触发器的时钟就会产生毛刺。毛刺出现的时间很短，但对于高速处理来讲，足以使触发器误动作。图(b)所示是使用同步时钟，并使用时钟使能 CE 端避免时钟信号毛刺的设计电路。



(a) 会产生毛刺的传统设计



(b) 能消除毛刺的优化电路

图 3-20　两种设计电路的比较

2) SRL16 的使用

SRL16 是一种基于查找表(LUT)的移位寄存器，可用于构建高密度 DSP 结构(如滤波器)，能够大幅削减硬件资源。在 Virtex-5 芯片中为 6 输入的查找表，在其余系列芯片中都为 4 输入查找表。其位宽(B)和深度(D)可以任意配置，最大的特点就是占用 Slice 资源特别少。占用 Slice 资源 M 的计算公式为

$$M = B(R[D/16]+1)$$

其中，$R[]$函数的意义是取整。从上式可以看出，深度为 1 的移位寄存器和深度为 16 的移位寄存器所占用的 Slice 资源是一样的。

例 3-15　使用 SRL16 生成位宽为 8，深度为 10 的移位寄存器。

```
module lut_ram(clk,d,q);
  input clk;
  input [7:0] d;
  output [7:0] q;

  srl16_based_ram srl16_based_ram(
  .clk(clk),
  .d(d),
  .q(q)
  );
endmodule
```

其中，srl16_based_ram 例化了 Xilinx 提供的 RAM_Based_Shiftreg 的 IP Core，使用

SRL16 结构完成了移位寄存器。上述程序经过 Synplify Pro 综合后，得到的 RTL 结构如图 3-21 所示。

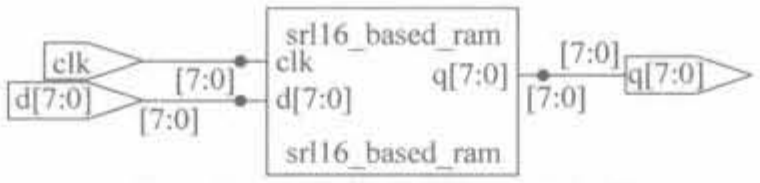


图 3-21　SRL16 结构移位寄存器的 RTL 结构示意图

在 ModelSim 6.2b 中完成仿真，其结果如图 3-22 所示

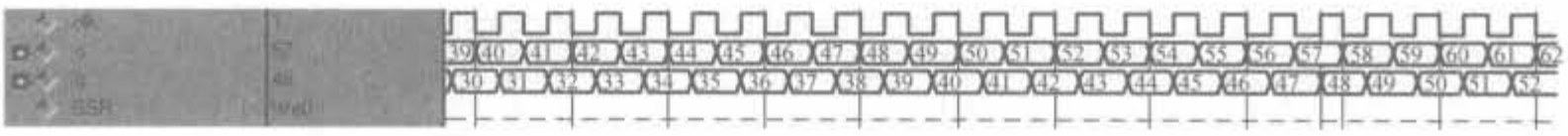


图 3-22　SRL16 结构移位寄存器的仿真结果示意图

3）触发器资源的分配技术

由于 FPGA 是一种触发器密集型可编程器件，因此系统的逻辑设计就应该充分利用触发器资源，尽可能降低每个组合逻辑操作的复杂度。

首先，应尽量使用库中的触发器资源。因为 FPGA 触发器资源丰富，而且开发系统在划分逻辑块时，对 D 触发器等元件直接利用 CLB 中的触发器，而对自建触发器认为是组合电路，需要使用 CLB 中的组合逻辑电路构成，这样既占用更多的 CLB，又浪费了 CLB 的触发器资源。将两种方法进行比较，每使用一个自建 D 触发器比使用库中 D 触发器的电路多占用 2～3 个 CLB。

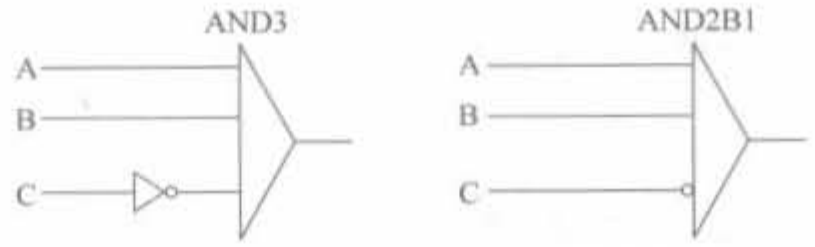
其次，在设计状态机时，应该尽量使用 ONE-HOT 状态编码方案，不用二进制状态编码方案。ONE-HOT 状态编码方案表示每个状态由 1 位触发器来表示，而二进制状态编码方案是用 $\lg N/\lg 2$ 位触发器来表示 N 个状态。由于二进制状态编码的稳定度较低，ONE-HOT 状态编码方案对于触发器资源丰富的 FPGA 芯片十分适用。

4）信号反相的处理策略

在处理反相信号时，设计时应尽可能地遵从分散反相原则。即应使用多个反相器分别反相，每个反相器驱动一个负载，这个原则无论对时钟信号还是对其他信号都是适用的。因为在 FPGA 设计中，反相是被吸收到 CLB 或 IOB 中的，使用多个反相器并不占用更多的资源，而使用一个反相器将信号反相后驱动多个负载会多占资源，而且延迟也增加了。

首先，如果输入信号需要反相，应尽可能地调用输入带反相功能的符号，而不是用分离的反相器对输入信号进行反相。例如，如图 3-23 所示电路是对逻辑 $Y=AB\overline{C}$ 的两种设计电路。两者相对比，最优的方案为采用如图(b)所示的电路，即直接调用 AND2B1，而不要用图(a)所示的电路，用分离的非门对输入信号 C 反相后，再连接到 AND3 的输入。因为在前一种做法中，由于函数发生器用查表方式实现逻辑，C 的反相操作是不占资源的，也没有额外延迟；而在后一种做法中，C 的反相操作与 AND3 操作可能会被分割到不同的逻辑单元中实现，从而消耗额外的资源，增加额外的延迟。

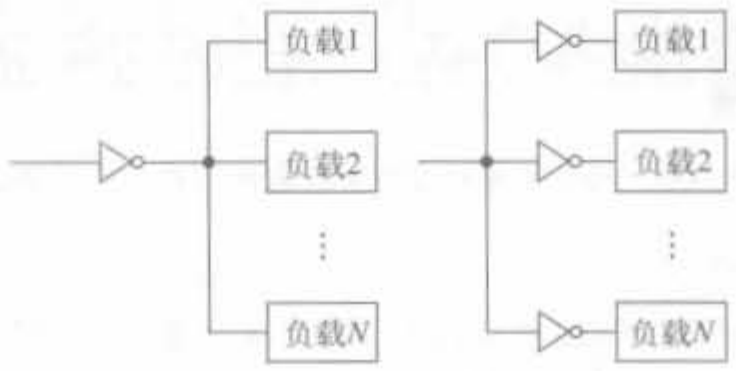
其次，如果一个信号反相后驱动了多个负载，应将反相功能分散到各个负载中实现，如图 3-24(b)所示；而不能采用传统 TTL 电路设计，采用集中反相驱动多个负载来减少所用的器件的数量，如图 3-24(a)所示。因为在 FPGA 设计中，集中反相驱动多个负载往往会多



(a) 非优化反相设计　　(b) 优化反相设计

图 3-23　两种反相设计电路比较

占一个逻辑块或半个逻辑块，而且延迟也增加了。分散信号的反相可以与其他逻辑在同一单元内完成，而不消耗额外的逻辑资源。



(a) 非优化集中反相　　(b) 优化分散反相

图 3-24　两种反相驱动电路比较

3.3　Verilog 建模与调试技巧

学习 Verilog 语言不仅在于了解语法和语句，更多的是积累实践经验。在使用 Verilog 语言时，一定要注意在调试过程中对相关问题和结论进行积累，这样才会逐步形成优秀的代码设计风格。本节将结合作者多年的工程经验给出一些常用的建模和调试技巧，以期对读者有所帮助。

3.3.1　双向端口的使用和仿真

顾名思义，双向端口既可以作为输入端口接收数据，也可以作为输出端口发出数据，对数据的操作是双向的。比如，某个设计需要一个 16 位的数据输入口和一个 16 位的数据输出口，并且数据输入和输出不会同时发生。如果分别设计数据输入口和输出口，需要 32 根数据线；而用双向端口来设计，只需要 16 根数据线，这样就节省了 16 根数据线管脚。本节给出 FPGA 中双向端口的设计原理和方法，以及仿真和初始化双向端口的方法。

1. 双向端口的实现原理

双向端口是通过控制三态门来实现的，其典型结构如图 3-25 所示。当 z=0 时，上面输出的管子开通，此时数据可以从上面的管子中输出，这样双向端口就作为输出口；当 z=1 时，上面的管子被置为高阻态，数据不能从上面的管子输出，此时数据只可以从下面的管子由外向内输入，这样的双向端口是输入口。

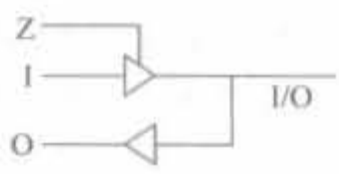


图 3-25　双向端口的硬件结构

2. 双向端口的 Verilog 实现

根据 Verilog HDL 语法，I/O 端口可以分成三类：输入端口 input、输出端口 output 以及双向端口 inout。input 端口不能被定义成寄存器变量型，只能是线网型；output 端口信号可定义成寄存器型变量，并在 always 块内可以被赋值使用；而 inout 型双向端口信号不能被定义成 reg 型变量，因此只能采用 assign 赋值语句，不能在 always 块内使用，这一点与 VHDL 中双向端口的使用方法不同。

双向端口的语法为：

```
inout a; wire z,b;
//当控制信号 z 为 1 时，开通三态门，a 为输入端口；当 z 为 0 时，三态门为高阻，a 为输出端口
```

例如，

```
assign a = (z)? b: 8'bz;
```

例 3-16　使用 Verilog 实现一个位宽为 16bit 的数据选择器，其结构如图 3-26 所示，当控制信号 z=1 时，将输入数据从双向端口输出；当控制信号 z=0 时，将双向端口数据从输出端口输出。

图 3-26　16bit 数据选择器的示意图

```
module bidirec_data(clk,z,din,dinout,dout);
    input clk;                  //时钟
    input z;                    //控制信号
    input [15:0] din;           //输入时钟
    inout [15:0] dinout;        //双向端口
    output [15:0] dout;         //输出时钟

    reg [15:0] dout;
    reg [15:0] din_t;

    assign dinout = (!z)? din_t: 16'bz;    //完成双向赋值

    always@(posedge clk) begin
    if (!z)
        din_t<=din;
    else
        dout<=dinout;
    end

endmodule
```

3. 双向端口的仿真

当双向端口作为输出口时，我们不需要对它进行初始化，而只需要开通三态门。当双向端口作为输入口时，需要对它进行初始化赋值，并关闭三态门。对双向端口的初始化赋值需要使用 wire 型的数据。此外，可以通过 force 命令来对双向端口输入赋值。

例 3-17 用 Verilog 完成例 3-16 的测试，并给出相应的测试结果。

```
module test_bidata;
    // The input signals
    reg clk;
    reg z;
    reg [15:0]din;
    // The output signals
    wire [15:0] dout;
    wire [15:0] dinout;
    integer i;

    bidirec_data uut(
        .din(din),
        .z(z),
        .clk(clk),
        .dout(dout),
        .dinout(dinout));

    always #10 clk = ~clk;

    initial begin
        z = 1;
        clk = 0;
        din = 0;
        force dinout = 20;
        #200 for (i = 0; i<10; i = i + 1)
        #20 force dinout = dinout - 1;
    end
    always #20 din =   din + 1;

endmodule
```

上述程序经过 ModelSim SE 6.2b 仿真，得到如图 3-27 所示的仿真结果。从中可以看到，该数据选择器的功能是正确的。

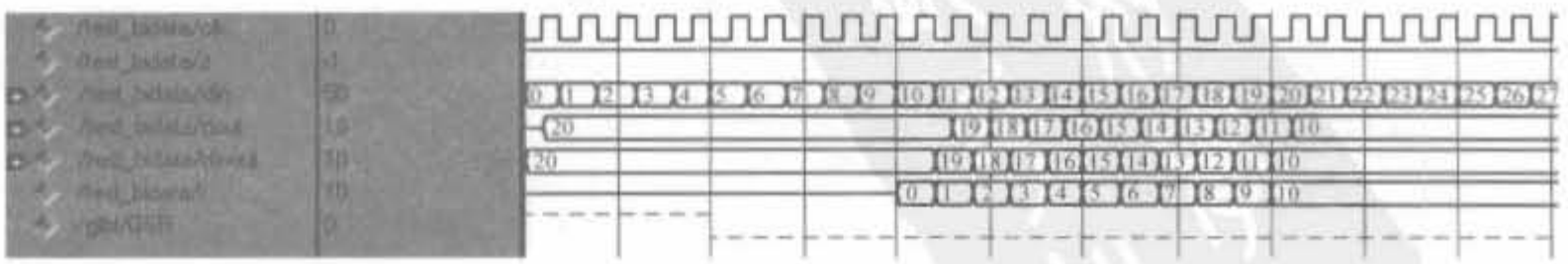


图 3-27 数据选择器的局部仿真结果示意图

3.3.2　阻塞赋值与非阻塞赋值

在对 Verilog HDL 程序进行仿真和综合的过程中，经常会遇到由于“=”和“<=”的使用不当而产生的不易察觉的问题，从而使仿真和综合的结果不一致，这是由于设计者对阻塞与非阻塞过程赋值的功能和执行过程没有深刻理解所造成的。本节将详细地阐述阻塞与非阻塞过程赋值的功能和执行过程，并通过一些具体的例子来分析它们之间的差异。

1. 功能定义

在硬件中，过程赋值语句表示用赋值语句右端表达式所推导出的逻辑来驱动该赋值语句左边表达式的变量。过程赋值语句只能出现在 always 语句和 initial 语句中。有两种过程赋值语句：

1）阻塞赋值(blocking assignments)

阻塞赋值由符号“=”来完成。“阻塞赋值”由其赋值操作行为而得名，“阻塞”即在当前的赋值完成前阻塞其他类型的赋值任务，但是如果右端表达式中含有延时语句，则在延时没结束前不会阻塞其他赋值任务。

2）非阻塞赋值(nonblocking assignments)

非阻塞赋值由符号“<=”来完成。“非阻塞赋值”也由其赋值操作行为而得名：在一个时间步(time step)的开始估计右端表达式的值，并在这个时间步结束时用等式右边的值取代左端表达式。在估算右端表达式和更新左端表达式的中间时间段，其他对左端表达式的非阻塞赋值可以被执行。即“非阻塞赋值”从估计右端开始并不阻碍执行其他的赋值任务。

2. 组合逻辑中的阻塞与非阻塞

首先，给出一个例子来说明两种赋值语句的不同。

例 3-18　使用 Verilog 给出阻塞赋值的实例。

```
module ex1 ( out,a,b,c,d);
    input a,b,c,d;
    output out;
    reg t1,t2,out;
    always @ (a or b or c or d) begin
        t1 = a & b;      //t1 <= a & b;
        t2 = c & d;      //t2 <= c & d;
        out = t1|t2;     //out <= t1|t2;
    end
endmodule
```

经过 ModelSim 6.2b 仿真后，得到的结果如图 3-28 所示。

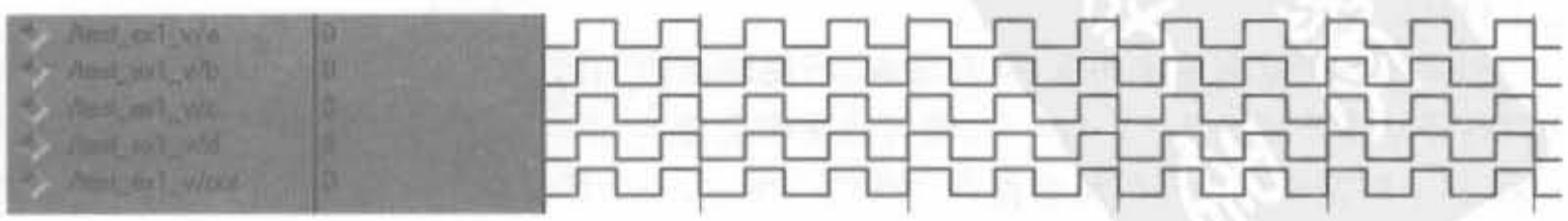


图 3-28　阻塞赋值的仿真结果

从例子中可以看出，如果 a、b、c、d 的值发生如下变化：a、b、c、d 都从 0→1，则采用阻塞赋值语句(always 语句中采用“//”前的语句)所得的结果是：t1 变为 1，t2 变为 1，out 变为 1；而采用非阻塞赋值语句(always 语句中采用“//”后的语句)所得的结果是：t1 变为 1，t2 变为 1，out 仍为 0。从阻塞与非阻塞赋值语句的执行过程来看就是：阻塞赋值是一步完成的，而且一条语句执行的同时会阻止其他阻塞赋值语句的执行，所以执行“out＝t1 | t2;”语句时所用的 t1、t2 的值是更新过的值；非阻塞赋值是两步完成的，而且一条语句的执行不会阻止其他非阻塞赋值语句同时执行其右端的估值，此时 t1、t2 都没有被更新，使用的都是未更新的旧值，然后才执行左端的更新，所以 out 的值不变。

由例 3-18 可见“＝”和“＜＝”对逻辑的不同影响，很显然，此例要实现的是两个与门然后或门的一个组合逻辑电路，而使用“＜＝”的电路并不能满足要求。如果读者仍希望利用非阻塞赋值，那么通过在触发事件表中增加变量，可以满足功能。即对于上例，如果仍利用“＜＝”，可以把@(a or b or c or d) 改为@ (a or b or c or d or t1 or t2)，这样，每当 t1 和 t2 发生变化时，always 语句就会被重新计算，并最终得到正确的 out 值，但是这样会降低仿真器性能。由本例可以发现非阻塞赋值存在以下两个问题：

- 非阻塞赋值不反映逻辑流；
- 需要将所有赋值对象都列入事件表中。

如果用阻塞式赋值，就很容易避免这两个问题。所以，一个好的编程风格就是：对组合逻辑建模采用阻塞式赋值。

3. 时序逻辑中的阻塞与非阻塞

首先来看一个用阻塞赋值语句实现的简单时序逻辑——D 触发器。

例 3-19 使用 Verilog 实现 D 触发器。

```
module ex2 (clk,d,q1,q2)
    input clk,d;
    output q1,q2;
    reg q1,q2;
    always @ (posedge clk) begin
        q1 = d;
        q2 = q1;
    end
endmodule
```

例 3-19 综合后的结果并不能得到所期望的逻辑电路。根据阻塞赋值语句的执行过程可以得到执行后的结果是 q1＝d，q2＝d，即实际只会综合出一个寄存器，而不是所期望的两个。那如何才能得到所需要的电路呢？如果把 always 块中的两个赋值语句的次序颠倒后再进行分析：先把 q1 的值赋予 q2，再把 d 赋予 q1，这样，q1 和 q2 的值就不再都是 d 了，满足了设计的要求。

如果用非阻塞赋值来实现，就会发现，不论两条语句的次序如何，都能满足要求。如果把寄存器从 2 个变为 3，4，…，n 个，语句的次序会更多，不同的次序对阻塞赋值会有不同的结果，但非阻塞赋值语句的结果都是一样的。所以，一个好的编程风格就是：对时序逻辑建模采用非阻塞式赋值。

通过实践表明，遵循以下好的编码风格可以大大减少设计中的错误和提高设计效率：

- 对组合逻辑建模采用阻塞式赋值。
- 对时序逻辑建模采用非阻塞式赋值。
- 用多个 always 块分别对组合和时序逻辑建模。
- 尽量不要在同一个 always 块里面混合使用“阻塞赋值”和“非阻塞赋值”。如果在同一个 always 块里面既为组合逻辑又为时序逻辑建模，应使用“非阻塞赋值”。

3.3.3　输入值不确定的组合逻辑电路

有些电路，尽管它的某些输入是不确定值，但其输出是个确定值。这种情况的最简单例子是，与门的一个输入是不确定值 x，另一个输入却是 0。在这种情况下，Verilog HDL 可以识别出门的输出一定是 0。此外，还有更复杂的例子，如实际中的 2 选 1 选择器，如果选择器的两个输入都是 0，而输入的控制信号是 x，那么不管控制信号是什么，输出确定是 0。但在这种情况下，Verilog HDL 不能认识到这种情况，相反地会把 x 值传播到输出口。所以，这需要设计者自行设计一个电路，使其在所有条件下都能展示出期望行为的选择器。

在 Verilog HDL 中，有两种不同的原因可能导致信号值为 x。第一种原因是有两个不同的信号源用相同的强度驱使同一个节点，并试图驱动成不同的逻辑值，这一般是由设计错误造成的。第二种原因是信号值没有初始化。所以在设计组合逻辑时，需要将不确定的输入转化成确定输入，然后再完成组合逻辑。这种结构如图 3-29 所示。

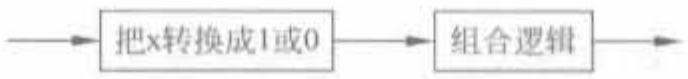


图 3-29　将输入的不确定值转换成确定值

例 3-20　一个转换不确定输入的模块。

```
module x2one (in,out);
 input in;
 output out;
     assign out = (in == 1)? 1: 0;
endmodule
```

3.3.4　数学运算中的扩位与截位操作

1. 扩位操作

在定点计算中，经过加法和乘法运算后，输出结果的位宽会增加。但如果继续使用和输入操作数同等位宽的数来表示结果，就会丢失有用的比特信息，造成输出结果错误。例如，在有限字长的情况下，若两个 M 位的数相加，其结果最高可能为 $M+1$ 位；若两个 M 位的数相乘，其结果最多可为 $2M$ 位。在例 3-21 中给出了扩位现象的例子。

例 3-21　展示 4bit 加法运算中的扩位现象。

下面分有符号数和无符号数两类情况来说明：

1）无符号数的加法：4'b1111＋4'b1111＝5'11110，但是由于加数是 4 位，在 Verilog 语言中只保留低 4 位，就会得到 4'b1111＋4'b1111＝4'1110 的结果，造成计算错误。

2）有符号数的加法：4'b0101 和 4'b0111 分别对应着+5 和+7，二者相加后本应为+12，即 5'b01100。但由于位宽限制，如不扩位，只能保留低 4 位，即 4'b1100，对应着-4，造成严重的计算错误。类似的错误还会造成负数相加变成正数。

从上面可以看出，对于数学运算，需要考虑位宽效率，否则会造成严重的计算错误。

2. 截位操作

在有限字长的情况下，若两个 M 位的数相加，其结果就是 $M+1$ 位；若两个 M 位的数相乘，其结果就是 $2M$ 位。但在实际的操作过程中，考虑到资源的问题，不能任由相加、相乘操作来增加操作数的位宽，必须进行截断。例如，两个 16 位数相乘后，其结果为 32 位，如再和一个 16 位数相乘，结果就变为 48 位，这样下去，用不了几个乘法操作，就会使操作数的位宽剧增，所占用的硬件资源也会很多。因此，需要将乘积结果进行截位，寄存在 M 位的寄存器中。

扩位和截取都是按照定点仿真的结果来定的。下面依次给出加法扩位操作、加保护截取操作和移位操作的书写规范。

1）加法实现规范，扩展符号位后相加。

```
reg[12:0] Adder_Out;
reg[11:0] Adder_In1,Adder_In2;

Adder_Out<={Adder_In1[11],Adder_In1}+{Adder_In2[11],Adder_In2};
```

2）对于截取乘法的结果，需要加溢出保护的截取规范。例如，要截取 12bit 输出的第 6 位到第 2 位，其实现代码为：

```
if((addRakeOut[11:6]==0)||(addRakeOut[11:6]==63))
     tmptraffic<=addRakeOut[6:2];
else
     tmptraffic<=(addRakeOut[11]==1)? 16:15;
```

或者：

```
if((addRakeOut[11:6]==6'b000000)||(addRakeOut[11:6]==6'b111111))
     tmptraffic<=addRakeOut[6:2];
else
     tmptraffic<=(addRakeOut[11]==1)? 5'b10000:5'b01111;
```

3）常用的移位操作范例：

```
reg[15:0] Data;

//左移 1 位
if((Data[15:14]==2'b00)||(Data[15:14]==2'b11))
    Data<={Data[14:0],1'b0};
else
     Data<=(Data[15])? 16'b1000_0000_0000_0000:16'b0111_1111_1111_1111;
//右移 1 位
Data<={Data[15],Data[15:1]};
```

3.3.5　利用块 RAM 来实现数据延迟

块 RAM 是 Xilinx FPGA 的一类核心资源，不占用任何逻辑资源。在设计中，合理利用块 RAM 能节省大量的逻辑资源，并且能保证时序，简化时序设计。块 RAM 作为一种存储单元，可以将其封装为 FIFO、移位寄存器以及延迟器。ISE 中提供了基于块 RAM 以及分布式 RAM 的 FIFO、移位寄存器等模块的 IP Core，可满足用户参数化配置。因此，本节主要讲述基于 RAM 的数据延迟器。

利用块 RAM 实现数据延迟器的思路就是：按照一定规律排列写地址，将数据依次写入。在读出时，将读地址比写地址延时 N 个空间，即可实现数据延迟。由于块 RAM 可以工作在芯片所支持的最高频率下，其工作时钟一般可达到数据流的几倍甚至几十倍，因此只要块 RAM 的容量满足，就可用于多路数据的延时。例 3-22 给出了块 RAM 实现两路数据延迟的实现。

例 3-22　利用块 RAM 实现 a、b 两路数据的延时，其中 a、b 两路数据的位宽都为 32bit，速率都为 61.44Mb/s。要求 a 路延迟 16 个数据时钟周期，b 路延迟 8 个数据时钟周期。

```
module bram_delay(clk_122p88MHz,a,b,a_delay,b_delay);
    input           clk_122p88MHz;
    input  [31:0]   a;
    input  [31:0]   b;
    output [31:0]   a_delay;
    output [31:0]   b_delay;

    reg    [31:0]   a_delay;
    reg    [31:0]   b_delay;
    wire   [5:0]    addra,addrb;
    wire   [31:0]   douta,doutb;
    reg    [5:0]    addra1 = 0;
    reg    [5:0]    addra2 = 0;
    reg    [5:0]    addrb1 = 32;
    reg    [5:0]    addrb2 = 32;
    reg             wea = 0;
    reg             web = 0;
    reg             flag = 0;

    always @(posedge clk_122p88MHz) begin
        flag <= !flag;
        if(flag == 1'b1) begin
            a_delay<= a_delay;
            b_delay<= b_delay;
            wea<= 1'b1;
            web<= 1'b1;
            addra2<= addra2;
            addrb2<= addrb2;
            if(addra1 == 31)
```

```
                addra1 <= 0;
            else
                addra1 <= addra1 + 1'b1;
            if(addrb1 == 63)
                addrb1 <= 32;
            else
                addrb1 <= addrb1 + 1'b1;
        end
        else begin
            wea <= 1'b0;
            web <= 1'b0;
            a_delay <= douta;
            b_delay <= doutb;
            addra1 <= addra1;
            addrb1 <= addrb1;
            if(addra1 <= 15)
                addra2 <= addra1 + 16;
            else
                addra2 <= addra1 - 16;
            if(addrb1 <= 39)
                addrb2 <= addrb1 + 24;
            else
                addrb2 <= addrb1 - 8;
        end
    end

    assign addra = !flag? addra1: addra2;
    assign addrb = !flag? addrb1: addrb2;

    bram_16 bram_16(
        .clka(clk_122p88MHz),
        .dina(a),
        .addra(addra),
        .wea(wea),
        .douta(douta),
        .clkb(clk_122p88MHz),
        .dinb(b),
        .addrb(addrb),
        .web(wea),
        .doutb(doutb)
    );

endmodule
```

上述程序经过 Synplify Pro 综合后，其 RTL 级结构如图 3-30 所示。其中，块 RAM 的 IP Core 作为黑盒子被综合。

在 ModelSim 6.2b 中完成仿真，其结果如图 3-31 所示。从中可以看出，上例成功利用块 RAM 完成了两路数据的延迟。

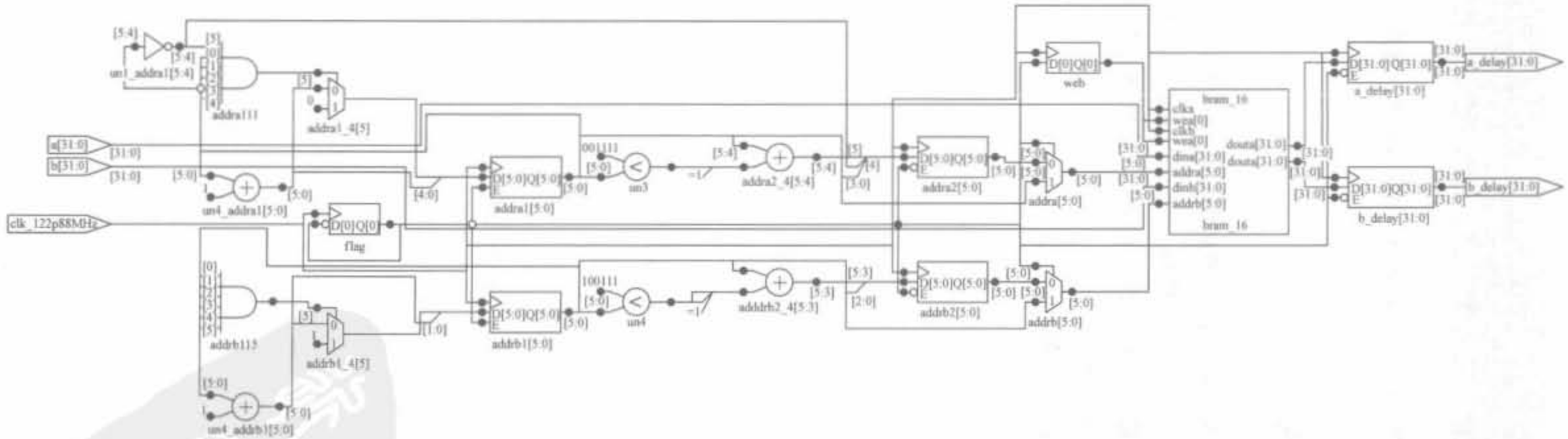


图 3-30　数据延迟电路的 RTL 级结构图

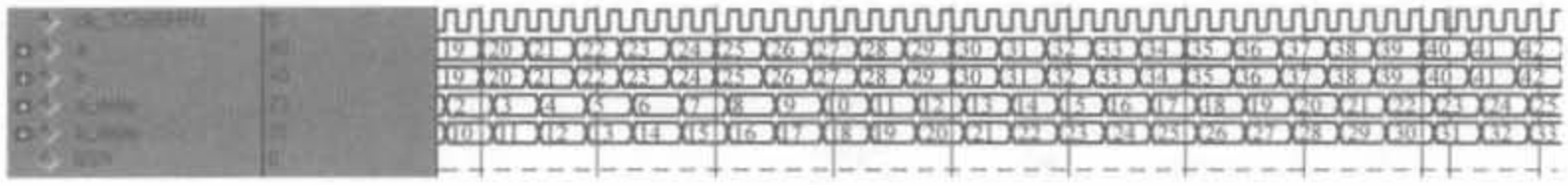


图 3-31　数据延迟电路的仿真结果示意图

3.3.6　测试向量的生成

Verilog HDL 还可以用来描述变化的测试信号。描述测试信号的变化和测试过程的模块叫做测试平台(Testbench),它可以对任何一个 Verilog/VHDL 模块进行动态的全面测试。通过测试被测试模块的输出信号是否符合要求,可以测试和验证逻辑系统的设计和结构正确与否,发现问题并及时修改。

下面给出测试模块的代码编写风格。

```
timescale 1ns / 1ps
module cmult_v;
    // 输入信号向量
    reg clk;
    reg [15:0] ar,ai,br,bi;

    //输出信号向量
    wire [31:0] qr,qi;

    // 实例化待测的模块单元 (UUT)
    cmultip uut (
        .clk(clk),.ar(ar),.ai(ai),.qr(qr),.br(br),.bi(bi),.qi(qi)
    );

    initial begin
        // 初始化输入向量
        clk = 0;   ar = 0;   ai = 0;   br = 0; bi = 0;

        #100;   //等待 100ns 后,全局 reset 信号有效
    ar = 20;   ai = 10;   br = 10; bi = 10;
    end

    always #5 clk = ~clk;
    always # 10 ar = ar + 1;
    always # 10 ai = ai + 1;
    always # 10 br = br + 1;
    always # 10 bi = bi + 1;

endmodule
```

在测试模块中,测试向量的产生是测试问题中的一个重要部分,只有测试向量产生得完备,分析测试结果才有意义。如果有方法产生出期望的结果,可以用 Verilog 或者其他工具

自动地比较期望值和实际值。如果没有简易的方法产生期望的结果，那么明智地选择测试向量，可以简化仿真的结果。当然，测试向量的产生是个在烦琐中追求特殊的情况，所以需要根据实际情况来选择测试向量。

在本书中我们引入先进的 Verilog 验证方法，应用 MATLAB 软件辅助电路设计，并进行电路功能验证。即用 MATLAB 首先完成浮点运算的设计，由输入得到相应的输出并将其量化，将量化后的定点输入导入仿真软件；完成仿真后，将仿真软件的输出和 MATLAB 的量化输出作对比，并统计结果。这样，只需要几个操作，便可以完成模块的功能仿真。

3.4 Xilinx 公司原语的使用方法

原语，其英文名字为 Primitive，是 Xilinx 针对其器件特征开发的一系列常用模块的名字，用户可以将其看成 Xilinx 公司为用户提供的库函数，类似于 C++ 中的“cout”等关键字。它们是芯片中的基本元件，代表 FPGA 中实际拥有的硬件逻辑单元，如 LUT、D 触发器和 RAM 等，相当于软件中的机器语言。在实现过程中的翻译步骤时，要将所有的设计单元都转译为目标器件中的基本元件，否则就是不可实现的。原语在设计中可以直接例化使用，是最直接的代码输入方式，它和 HDL 语言的关系类似于汇编语言和 C 语言的关系。

Xilinx 公司提供的原语涵盖了 FPGA 开发的常用领域，但只有相应配置的硬件才能执行相应的原语，并不是所有的原语都可以在任何一款芯片上运行。在 Verilog 中使用原语非常简单，将其作为模块名直接例化即可。本节以 Virtex-4 平台介绍各类原语，因为该系列的原语类型是较为全面的，其他系列芯片原语的使用方法是类似的。

Xilinx 公司的原语按照功能分为 10 类，包括计算组件、I/O 端口组件、寄存器和锁存器、时钟组件、处理器组件、移位寄存器、配置和检测组件、RAM/ROM 组件、Slice/CLB 组件以及吉比特收发器组件。下面将分别详细介绍。

3.4.1 计算组件

计算组件指的就是 DSP48 核，也有人将其称为硬件乘法器，功能描述如表 3-6 所示。

表 3-6 计算组件清单

原语名	描　述
DSP48	其结构为一个 18×18bit 的有符号乘法器，且在后面级联了一个带有可配置流水线的 3 输入加法器

DSP48 核由一个 18bit 的乘法器后面级联一个 48bit 的加法器构成，乘法器和加法器的应用位宽分别可以在 18bit 和 48bit 内任意调整。它在乘加模块中有广泛应用，特别是在各类滤波器系统中，不仅可以提高系统稳定性，还能够节省逻辑资源且工作在高速模式下。它在 Verilog 中的例化模板为：

```
module fpga_v4_dsp48(
    BCOUT,P,PCOUT,A,B,BCIN,C,CARRYIN,CARRYINSEL,CEA,CEB,
    CEC,CECARRYIN,CECINSUB,CECTRL,CEM,CEP,CLK,OPMODE,
```

```
PCIN,RSTA,RSTB,RSTC,RSTCARRYIN,RSTM,RSTP,SUBTRACT);
output [17:0] BCOUT;
output [47:0] P,PCOUT;     //乘加器的输出结果
input [17:0] A,B;          //乘加器输入端口
input [47:0] C,PCIN;
input [1:0] CARRYINSEL;
input [6:0] OPMODE;
input BCIN,CARRYIN,CEA,CEB,CEC,CECARRYIN,CECINSUB,CECTRL,CEM,
CEP,CLK,RSTA,RSTB,RSTC,RSTCARRYIN,RSTM,RSTP,SUBTRACT;

//对 DSP48 原语的功能进行配置
DSP48   #(
        .AREG(1),          //输入 A 口的流水线寄存器长度,可为 0、1、2
        .BREG(1),          //输入 A 口的流水线寄存器长度,可为 0、1、2
        .B_INPUT("DIRECT"),
        //设定 B 端口直接从构造或级联的其余 DSP48 单元输入
        .CARRYINREG(1),
        //进位寄存器 CARRYINREG 的流水线寄存器长度,可为 0 或 1
        .CARRYINSELREG(1),
        //进位设置寄存器 CARRYINSELREG 的流水线寄存器长度,可为 0 或 1
        .CREG(1),
        //C 输入端口的流水线寄存器数量,可为 0 或 1
        .LEGACY_MODE("MULT18X18S"),
        //向后兼容特性设置,可为 NONE、MULT18X18 或者 MULT18X18S
        .MREG(1),
        //乘法器流水线寄存器的数量,可为 0 或 1
        .OPMODEREG(1),
        //OPMODE 输入端的流水线寄存器,可为 0 或 1
        .PREG(1),
        //P 输出端的流水线寄存器,可为 0 或 1
        .SUBTRACTREG(1)
        // SUBTRACT 输入端的流水线数量,可为 0 或 1
)  fpga_v4_dsp48 (
        .BCOUT(BCOUT),
        //18bit 的级联输出
        .P(P),
        //48bit 的乘积输出 t
        .PCOUT(PCOUT),
        //48bit 的级联输出
        .A(A),
        //18bit 的 A 数据输入端口
        .B(B),
        //18bit 的 B 数据输入端口
        .BCIN(BCIN),
        //18bit 的 B 端口级联输入
        .C(C),
        //48bit 级联输入
        .CARRYIN(CARRYIN),
        //进位输入信号 CARRYIN
        .CARRYINSEL(CARRYINSEL),
        //2bit 的进位输入选择信号 CARRYINSEL
```

```
.CEA(CEA),
// A端口数据时钟使能信号 CEA,输入端口
.CEB(CEB),
// B端口数据时钟使能信号 CEB,输入端口
.CEC(CEC),
// C端口数据时钟使能信号 CEC,输入端口
.CECARRYIN(CECARRYIN),
// 进位输入端口数据时钟使能信号 CECARRYIN,输入端口
.CECINSUB(CECINSUB),
// CINSUB 时钟使能信号 CECINSUB
.CECTRL(CECTRL),
// CTRL 寄存器的时钟使能信号 CECTRL
.CEM(CEM),
// 乘法器寄存器的时钟使能输入信号
.CEP(CEP),
// P 寄存器的时钟使能输入信号
.CLK(CLK),
// 时钟输入信号 CLK
.OPMODE(OPMODE),
// 7bit 的操作模式输入信号 OPMODE
.PCIN(PCIN),
// 48bit 的 PCIN 输入信号 PCIN
.RSTA(RSTA),
// A 端口流水线寄存器的复位输入信号 RSTA
.RSTB(RSTB),
// B 端口流水线寄存器的复位输入信号 RSTB
.RSTC(RSTC),
// C 端口流水线寄存器的复位输入信号 RSTC
.RSTCARRYIN(RSTCARRYIN),
// 进位寄存器的复位输入信号 RSTCARRYIN
.RSTCTRL(RSTCTRL),
// CTRL 寄存器的复位输入信号 RSTCTRL
.RSTM(RSTM),
// 硬核乘法器的复位输入信号 RSTM
.RSTP(RSTP),
// P 流水线寄存器的复位信号 RSTP
.SUBTRACT(SUBTRACT)
// 减法操作输入控制信号
);

endmodule
```

3.4.2　时钟组件

时钟组件包括各种全局时钟缓冲器、全局时钟复用器、普通 I/O 本地的时钟缓冲器以及高级数字时钟管理模块，如表 3-7 所示。

表 3-7　时钟组件的清单

原　语　名	描　　述
BUFG	全局时钟缓冲器
BUFGCE	全局时钟复用器，附带时钟使能信号和 0 状态输出
BUFGCE_1	全局时钟复用缓冲器，附带时钟使能信号和 1 状态输出
BUFGCTRL	全局时钟复用缓冲器
BUFGMUX	全局时钟复用缓冲器，附带时钟使能信号和 0 状态输出
BUFMUX_1	全局时钟复用器，附带 0 状态输出
BUFGMUX_VIRTEX4	Virtex-4 器件特有的全局时钟复用缓冲器
BUFIO	I/O 端口本地时钟缓冲器
BUFR	I/O 端口和 CLB 的本地时钟缓冲器
DCM_ADV	带有高级特征的数字时钟管理模块
DCM_BASE	带有基本特征的数字时钟管理模块
DCM_PS	带有基本特征和移相特征的数字时钟管理模块
PMCD	匹配相位时钟分频器

下面对几个常用时钟组件进行简单介绍，其余组件的使用方法是类似的。

1. BUFG

BUFG 是具有高扇出的全局时钟缓冲器，一般由综合器自动推断并使用。全局时钟是具有高扇出驱动能力的缓冲器，可以将信号连到时钟抖动可以忽略不计的全局时钟网络。BUFG 组件还可应用于典型的高扇出信号和网络，如复位信号和时钟使能信号。如果要对全局时钟实现 PLL 或 DCM 等时钟管理，需要手动例化该缓冲器，其例化的代码模板如下所示：

```
// BUFG: 全局时钟缓存(Global Clock Buffer),只能以内部信号驱动
// Xilinx HDL 库向导版本,ISE 9.1
BUFG BUFG_inst (
.O(O),//时钟缓存输出信号
.I(I)  //时钟缓存输入信号
);
// 结束 BUFG_ins 模块的例化过程
```

在综合结果分析中，它和同类原语的 RTL 级结构如图 3-32 所示。

图 3-32　全局时钟原语的 RTL 级结构示意图

2. BUFMUX

BUFMUX 是全局时钟复用器，选择两个输入时钟 I0 或 I1 中的一个作为全局时钟。当选择信号 S 为低时，选择 I0；否则输出 I1。BUFMUX 原语和 BUFMUX1 原语的功能一样，只是选择逻辑不同。对于 BUFMUX1，当选择信号 S 为低时，选择 I1；否则输出 I0。

BUFMUX 原语的例化代码模板如下所示：

```
// BUFGMUX: 全局时钟的 2 到 1 复用器(Global Clock Buffer 2 - to - 1 MUX)
```

```
// 适用芯片：Virtex-Ⅱ/Ⅱ-Pro/4/5,Spartan-3/3E/3A
// Xilinx HDL 库向导版本,ISE 9.1
BUFGMUX BUFGMUX_inst (
.O(O),  //时钟复用器的输出信号
.I0(I0),  // 0 时钟输入信号
.I1(I1),  //1 时钟输入信号
.S(S)   // 时钟选择信号
);
// 结束 BUFGMUX_inst 模块的例化过程
```

需要注意的是，该原语只用于全局时钟处理，不能作为接口使用。在综合结果分析时，它和同类原语 BUFMUX1 的 RTL 级结构如图 3-33 所示。

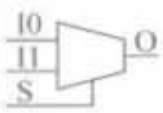


图 3-33 全局时钟复用器的 RTL 级结构示意图

3. BUFIO

BUFIO 是本地 I/O 时钟缓冲器，只有一个输入与输出，非常简单。BUFIO 使用独立于全局时钟网络的专用时钟网络来驱动纵向 I/O 管脚，所以非常适合同步数据采集。BUFIO 要求时钟和相应的 I/O 必须在同一时钟区域，而不同时钟网络的驱动需要 BUFR 原语来实现。需要注意的是，由于 BUFIO 引出的时钟只到达了 I/O 列，所以不能用来驱动逻辑资源，如 CLB 和块 RAM。

BUFIO 的例化代码模板如下：

```
// BUFIO：本地 I/O 时钟缓冲器( Local Clock Buffer)
// 适用芯片：Virtex-4/5
// Xilinx HDL 库向导版本,ISE 9.1
BUFIO BUFIO_inst (
.O(O),//本地 I/O 时钟缓冲器的输出信号
.I(I)  //本地 I/O 时钟缓冲器的输入信号
);
// 结束 BUFIO 模块的例化过程
```

在综合结果分析时，其 RTL 级结构如图 3-34 所示。

图 3-34 本地 I/O 时钟缓冲器的 RTL 级结构示意图

4. BUFR

BUFR 是本地 I/O 时钟、逻辑缓冲器。BUFR 和 BUFIO 都是将驱动时钟引入某一时钟区域的专用时钟网络，而独立于全局时钟网络；不同的是，BUFR 不仅可以跨越不同的时钟区域(最多 3 个)，还能够驱动 I/O 逻辑以及自身或邻近时钟区域的逻辑资源。BUFIO 的输出和本地内部互连都能驱动 BUFR 组件。此外，BUFR 能完成输入时钟 1～8 的整数分频。因此，BUFR 是同步设计中实现跨时钟域以及串/并转换的最佳方式。

BUFIO 的例化代码模板如下：

```
// BUFR：本地 I/O 时钟、逻辑缓冲器(Regional Clock Buffer)
// 适用芯片：Virtex-4/5
// Xilinx HDL 库向导版本，ISE 9.1
BUFR #(
.BUFR_DIVIDE("BYPASS"),
//分频比，可选择 "BYPASS"、"1"、"2"、"3"、"4"、"5"、"6"、"7"和"8"
.SIM_DEVICE("VIRTEX4")
// 指定目标芯片，"VIRTEX4" 或者 "VIRTEX5"
) BUFR_inst (
.O(O),        //时钟缓存输出信号
.CE(CE),      //时钟使能信号，输入信号
.CLR(CLR),    //时钟缓存清空信号
.I(I)         // 时钟缓存输入信号
);
// 结束 BUFR 模块的例化过程
```

需要注意的是，BUFIO 和 BUFR 只能在 Virtex-4 系列以及更高系列芯片中使用。在综合结果分析时，其 RTL 级结构如图 3-35 所示。

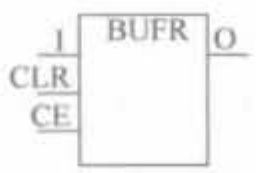


图 3-35　本地 I/O 时钟、逻辑缓冲器的 RTL 级结构示意图

5. DCM_BASE

DCM_BASE 是基本数字时钟管理模块的缩写，是相位和频率可配置的数字锁相环电路，常用于 FPGA 系统中复杂的时钟管理。如果需要频率和相位动态重配置，可以选用 DCM_ADV 原语；如果需要相位动态偏移，可使用 DCM_PS 原语。模块接口信号的说明如表 3-8 所示。

表 3-8　DCM_BASE 原语的信号描述列表

管脚名	方向	位宽	简要描述
相关的时钟输入、输出管脚			
CLK0	输出信号	1	输出和 DCM 模块输入管脚 CLKIN 有效频率相同的时钟。如果不选中 CLKIN_DIVIDE_BY_2 选项，在默认情况下，其大小等于 CLKIN。如果连接了 CLKFB，那么 CLK0 与 CLKIN 的相位也是对准的
CLK90	输出信号	1	CLK90 输出时钟的频率和 CLK0 大小一致，只是在相位上偏移了 90°
CLK180	输出信号	1	CLK180 输出时钟的频率和 CLK0 大小一致，只是在相位上偏移了 180°，也就是 CLK0 的反相信号
CLK270	输出信号	1	CLK270 输出时钟的频率和 CLK0 大小一致，只是在相位上偏移了 270°

续表

管脚名	方向	位宽	简要描述
相关的时钟输入、输出管脚			
CLK2X	输出信号	1	CLK2X 输出时钟的频率为 CLK0 的两倍，相位亦和 CLK0 时对齐，且占空比为 50%。在 DCM 模块未达到锁相的稳定状态前，其占空比为 1/3，用来调整 DCM 模块锁相到正确的变化沿上
CLK2X180	输出信号	1	CLK2X180 输出时钟的频率和 CLK2X 大小一致，只是在相位上有 180°的偏移，即 CLK2X 的反相信号
CLKDV	输出信号	1	CLKDV 输出时钟的频率为 CLK0 频率的分数，分频比由 CLKDV_DIVIDE 参数决定
CLKFX	输出信号	1	CLKFX 输出时钟的频率由下式决定： CLKFX 频率＝CLKIN 有效频率×(*M*/*D*) 其中，*M* 是倍频比，*D* 是分频比。当 CLKFB 管脚连接时，其相位和 CLK0、CLK2X 以及 CLKDV 是一致的
CLKFX180	输出信号	1	CLKFX180 的时钟频率和 CLKFX 频率大小一致，只是移相了 180°
CLKIN	输入信号	1	源时钟输入信号，其大小必须满足一定的范围，具体数值需要参考不同芯片的数据手册。CLKIN 的输入时钟必须来自下列缓存器： 1. 全局时钟输入缓存(IBUFG) 2. 内部全局时钟缓冲器(BUFG/BUFGCTRL) 3. 输入缓存器(IBUF)
相关的控制输入以及状态输出管脚			
LOCKED	输出信号	1	PLL 的同步输出，用于指示锁相环是否已完成锁相，可以完成用户操作
RST	输入信号	1	DCM 模块的复位信号，高电平有效。在复位期间，所有的输出管脚都为低电平

DCM_BASE 组件可以通过 Xilinx 的 IP Wizard 向导产生，也可以直接通过例化代码直接使用。其 Verilog 的例化代码模板为：

```
// DCM_BASE：基本数字时钟管理电路(Base Digital Clock Manager Circuit)
// 适用芯片：Virtex-4/5
// Xilinx HDL 库向导版本，ISE 9.1
DCM_BASE #(
.CLKDV_DIVIDE(2.0),
// CLKDV 分频比可以设置为 1.5、2.0、2.5、3.0、3.5、4.0、4.5、5.0、5.5、6.0、6.5
// 7.0、7.5、8.0、9.0、10.0、11.0、12.0、13.0、14.0、15.0 或 16.0
.CLKFX_DIVIDE(1),
// CLKFX 信号的分频比，可为 1～32 之间的任意整数
.CLKFX_MULTIPLY(4),
// CLKFX 信号的倍频比，可为 2～32 之间的任意整数
.CLKIN_DIVIDE_BY_2("FALSE"),
// 输入信号 2 分频的使能信号，可设置为 TRUE/FALSE
.CLKIN_PERIOD(10.0),
// 指定输入时钟的周期，单位为 ns，数值范围为 1.25～1000.00
.CLKOUT_PHASE_SHIFT("NONE"),
```

```
// 指定移相模式,可设置为 NONE 或 FIXED
.CLK_FEEDBACK("1X"),
// 指定反馈时钟的频率,可设置为 NONE、1X 或 2X。相应的频率关系都是针对 CLK0 而言的
.DCM_PERFORMANCE_MODE("MAX_SPEED"),
// DCM 模块性能模式,可设置为 MAX_SPEED 或 MAX_RANGE
.DESKEW_ADJUST("SYSTEM_SYNCHRONOUS"),
// 抖动调整,可设置为源同步、系统同步或 0～15 之间的任意整数
.DFS_FREQUENCY_MODE("LOW"),
// 数字频率合成模式,可设置为 LOW 或 HIGH 两种频率模式
.DLL_FREQUENCY_MODE("LOW"),
// DLL 的频率模式,可设置为 LOW、HIGH 或 HIGH_SER
.DUTY_CYCLE_CORRECTION("TRUE"),
// 设置是否采用双周期校正,可设为 TRUE 或 FALSE
.FACTORY_JF(16'hf0f0),
// 16bit 的 JF 因子参数
.PHASE_SHIFT(0),
// 固定相移的数值,可设置为 - 255～1023 之间的任意整数
.STARTUP_WAIT("FALSE")
// 等 DCM 锁相后再延迟配置 DONE 管脚,可设置为 TRUE/FALSE
) DCM_BASE_inst (
.CLK0(CLK0),           // 0°移相的 DCM 时钟输出
.CLK180(CLK180),       // 180°移相的 DCM 时钟输出
.CLK270(CLK270),       // 270°移相的 DCM 时钟输出
.CLK2X(CLK2X),         // DCM 模块的 2 倍频输出
.CLK2X180(CLK2X180),   // 经过 180°移相的 DCM 模块 2 倍频输出
.CLK90(CLK90),         // 90°移相的 DCM 时钟输出
.CLKDV(CLKDV),         // DCM 模块的分频输出,分频比为 CLKDV_DIVIDE
.CLKFX(CLKFX),         // DCM 合成时钟输出,分频比为(M/D)
.CLKFX180(CLKFX180),   // 180°移相的 DCM 合成时钟输出
.LOCKED(LOCKED),       // DCM 锁相状态输出信号
.CLKFB(CLKFB),         // DCM 模块的反馈时钟信号
.CLKIN(CLKIN),         // DCM 模块的时钟输入信号
.RST(RST)              // DCM 模块的异步复位信号
);
// 结束 DCM_BASE 模块的例化过程
```

在综合结果分析时,DCM 系列原语的 RTL 级结构如图 3-36 所示。

图 3-36 DCM 模块的 RTL 级结构示意图

3.4.3 配置和检测组件

配置和检测组件提供了 FPGA 内部逻辑和 JTAG 扫描电路之间的数据交换以及控制功能,主要由 6 个原语组成,如表 3-9 所示。

表 3-9 配置和检测原语列表

原 语	描 述
BSCAN_VIRTEX4	提供到 Virtex-4 边界扫描点的接入
CAPTURE_VIRTEX4	Virtex-4 边界扫描控制逻辑电路
FRAME_ECC_VIRTEX4	读入一帧 Virtex-4 配置数据，并能完成汉明、单错误纠正和双错误检测
ICAP_VIRTEX4	Virtex-4 内部配置接入端口
STARTUP_VIRTEX4	Virtex-4 配置时钟、全局复位、全局三态控制和其他配置信号的用户接口
USR_ACCESS_VIRTEX4	带有 32 位数据总线和有效数据指示端口的 32bit 寄存器

下面对 BSCAN_VIRTEX4 组件进行简单介绍，其余组件的使用方法是类似的。

当 JTAG USER1/2/3/4 指令被加载后，BSCAN_VIRTEX4 允许设计人员检测 TCK、TMS 以及 TDI 等专用 JTAG 管脚的数据，并且可以将用户数据写入到 TDO 管脚上。这样，可以在 PC 上通过 JTAG 链读取芯片内部的用户数据。

BSCAN_VIRTEX4 的管脚信号说明如下：

(1) CAPTURE：位宽为 1 的输出信号，用于指示是否加载了用户指令。当 JTAG 接口处于 CAPTURE-DR 状态时，输出为高电平。

(2) DRCK：位宽为 1 的输出信号，用于监测 JTAG 电路的 TCK 信号。当 JTAG 链路处于用户指令模式，或者 JTAG 接口为 SHIFT-DR 状态时，才有信号输出。

(3) RESET：位宽为 1 的输出信号，在加载用户指令时有效。当 JTAG 接口控制器处于 TEST-LOGIC-RESET 状态时，置高电平。

(4) SEL：位宽为 1 的输出信号，高电平有效。用于指示 USER1 数据是否加载到 JTAG 指令寄存器中。SEL 在 UPDATE-IR 状态时有效，直到加载新的指令之前，一直保持有效电平。

(5) SHIFT：位宽为 1 的输出信号，加载用户指令时有效。当 JTAG 接口控制器处于 SHIFT-DR 状态时，置高电平。

(6) TDI：位宽为 1 的输出信号，用于检测 JTAG 链的 TDI 信号。

(7) UPDATE：位宽为 1 的输出信号，加载 USER1 指令和 USER2 指令时有效。当 JTAG 接口控制器处于 UPDATE-DR 状态时，置高电平。

(8) TDO：位宽为 1 的输入信号，可以将外部 JTAG 链的 TDO 信号直接连到该管脚上。

在 Virtex-4 芯片中，有 4 个 BSCAN_VIRTEX4 硬件原语可用。因此，其属性 JTAG_CHAIN 的有效值为 1～4，默认值为 1。

BSCAN_VIRTEX4 原语的例化代码模板如下所示：

```
// BSCAN_VIRTEX4：完成内部逻辑和 JTAG 接口连接的边界扫描原语(Boundary Scan primitive for
connecting internal logic to JTAG interface)
// 适用芯片：Virtex-4/5
// Xilinx HDL 库向导版本，ISE 9.1
BSCAN_VIRTEX4 #(
.JTAG_CHAIN(1)
// 指定 JTAG 链用户指令，必须为 1、2、3 或 4 中的任何一个正整数
```

```
) BSCAN_VIRTEX4_inst (
.CAPTURE(CAPTURE),          // 捕获到的从 TAP 控制器的输出
.DRCK(DRCK),                // 用户函数服务的数据寄存器输出
.RESET(RESET),              // TAP 控制器输出的复位信号
.SEL(SEL),                  // 用户激活输出
.SHIFT(SHIFT),              // TAP 控制器的移位输出
.TDI(TDI),                  // TAP 控制器的 TDI 输出信号
.UPDATE(UPDATE),            // TAP 控制器的 UPDATE 输出信号
.TDO(TDO)                   // 用户函数的数据输入信号
);
// 结束 BSCAN_VIRTEX4 模块的例化过程
```

在综合结果分析时，BSCAN_VIRTEX4 的 RTL 级结构图如图 3-37 所示。

图 3-37 BSCAN_VIRTEX4 原语的 RTL 级结构图

3.4.4 吉比特收发器组件

吉比特收发器组件用于调用 Rocket I/O 吉比特级收发器，支持 622Mb/s～6.5Gb/s 的多速率应用。它符合最广泛的芯片、背板和光学器件的标准及协议，具有高级的 Tx/Rx 均衡技术。其中，收发器最多可达 24 个，提供了完整的串行 I/O 解决方案，具体包括 4 个组件，如表 3-10 所示。

表 3-10 吉比特收发器组件

原　语	描　述
GT11_CUSTOM	速度范围可达 622Mb/s～11.1Gb/s 的 Rocket I/O MGT，单片 FPGA 可提供 8～24 个高速收发器，VCO 的频率高达 2.5GHz～5.55GHz，且抖动小于 1ns
GT11_DUAL	Rocket I/O 专用俗语，包含两个 GT11_CUSTOM
GT11CLK	能够完成差分包输入时钟、来自于 Fabric 的参考时钟以及驱动 MGT 列两个垂直参考时钟总线的 rxbclk 三者时分复用的复用器
GT11CLK_MGT	允许差分包输入来驱动 MGT 列的两个垂直参考时钟总线

由于吉比特收发器操作复杂，使用原语很容易出错，不易配置，因此需要在 ISE 中通过结构向导完成。有关吉比特收发器的原理和使用方法，将在第 10 章详细介绍。

3.4.5 I/O 端口组件

I/O 组件提供了本地时钟缓存、标准单端 I/O 缓存、差分 I/O 信号缓存、DDR 专用 I/O 信号缓存、可变抽头延迟链、上拉、下拉以及单端信号和差分信号之间的相互转换，具体包括了 21 个原语，如表 3-11 所示。

表 3-11　I/O 端口组件

原　语	描　述
BUFIO	I/O 的本地时钟缓存
DCIRESER	FPGA 配置成功后，DCI 状态机的复位信号
IBUF	标准和容量可选择 I/O 单端口输入缓存
IBUFDS	带可选择端口的差分信号输入缓存
IBUFG	带可选择端口的专用输入缓存
IBUFGDS	带可选择端口的专用差分信号输入缓存
IDDR	用于接收外部 DDR 输入信号的专用输入寄存器
IDELAY	专用的可变抽头输入延迟链
IDELAYCTRL	IDELAY 抽头数的控制模块
IOBUF	带可选择端口的双向缓存
IOBUFDS	低有效输出的三态差分信号 I/O 缓存
ISERDES	专用 I/O 缓存的输入分解器
KEEPER	KEEPER 符号
OBUF	单端输出端口缓存
OBUFT	带可选择端口的低有效输出的三态输出缓冲
OBUFDS	带可选择端口的差分信号输出缓冲
OBUFTDS	带可选择端口的低有效输出的三态差分输出缓冲
ODDR	用于向外部 DDR 发送信号的专用输出寄存器
OSERDES	用于快速实现输入源同步接口
PULLDOWN	输入端寄存器下拉至 0
PULLUP	输入端寄存器、开路以及三态输出端口上拉至 V_{CC}

下面对几个常用 I/O 组件进行简单介绍，其余组件的使用方法是类似的。

1. BUFIO

BUFIO 是 FPGA 芯片内部简单的时钟输入、输出缓存器。BUFIO 使用独立于全局时钟网络的专用时钟网络来驱动 I/O 列，因此非常适合用于源同步的数据采集。但要注意的是，BUFIO 只能在单一的时钟区域内使用，不能跨时钟域操作。此外，BUFIO 不能用于驱动逻辑资源(CLB、块 RAM 等)，因此它只能到达 I/O 列。

BUFIO 原语的例化代码模板如下所示：

```
// BUFIO: 本地时钟缓存(Local Clock Buffer)
// 适用芯片: Virtex-4/5
// Xilinx HDL 库向导版本,ISE 9.1
BUFIO BUFIO_inst (
.O(O),     // 时钟缓冲器输出
.I(I)      // 时钟缓冲器输入
);
// 结束 BUFIO 模块的例化过程
```

在综合结果分析时，BUFIO 原语的 RTL 结构如图 3-38 所示。

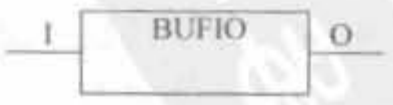


图 3-38　BUFIO 的 RTL 结构图

2. IBUFDS

IBUFDS原语用于将差分输入信号转化成标准单端信号，且可加入可选延时。在IBUFDS原语中，输入信号为I和IB，一个为主，一个为从，二者相位相反。

IBUFDS的逻辑真值表如表3-12所示，其中“－＊”表示输出维持上一次的输出值，保持不变。

表3-12 IBUFDS原语的输入、输出真值表

输　入		输　出
I	IB	O
0	0	-*
0	1	0
1	0	1
1	1	-*

BUFDS原语的例化代码模板如下所示：

```
// IBUFDS:  差分输入缓冲器(Differential Input Buffer)
// 适用芯片：Virtex-Ⅱ/Ⅱ-Pro/4,Spartan-3/3E
// Xilinx HDL库向导版本,ISE 9.1
IBUFDS #(
.DIFF_TERM("FALSE"),
// 差分终端,只有 Virtex-4 系列芯片才有,可设置为 TRUE/FLASE
.IOSTANDARD("DEFAULT")
// 指定输入端口的电平标准,如果不确定,可设为 DEFAULT
) IBUFDS_inst (
.O(O),     // 时钟缓冲输出
.I(I),     // 差分时钟的正端输入,需要和顶层模块的端口直接连接
.IB(IB)    // 差分时钟的负端输入,需要和顶层模块的端口直接连接
);
// 结束 IBUFDS 模块的例化过程
```

在综合结果分析时，IBUFDS的RTL级结构如图3-39所示。

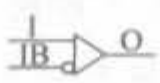


图3-39 IBUFDS原语的RTL级结构图

3. IDELAY

在Virtex-4系列芯片中，每个用户I/O管脚的输入通路都有一个IDELAY模块，可用于数据信号或时钟信号，以使二者同步，准确采集输入数据。IDELAY具有可控的64抽头延迟线，每个抽头的延迟都是经过精密校准的78ps，且与进程、电压和温度特性无关，其内部结构如图3-40所示。

IDELAY原语的信号说明如下：

(1) I：单比特输入信号，来自于IOB的串行输入数据。

(2) C：单比特输入信号，时钟输入信号。

(3) INC：单比特输入信号，用于增加或减少延迟抽头数。

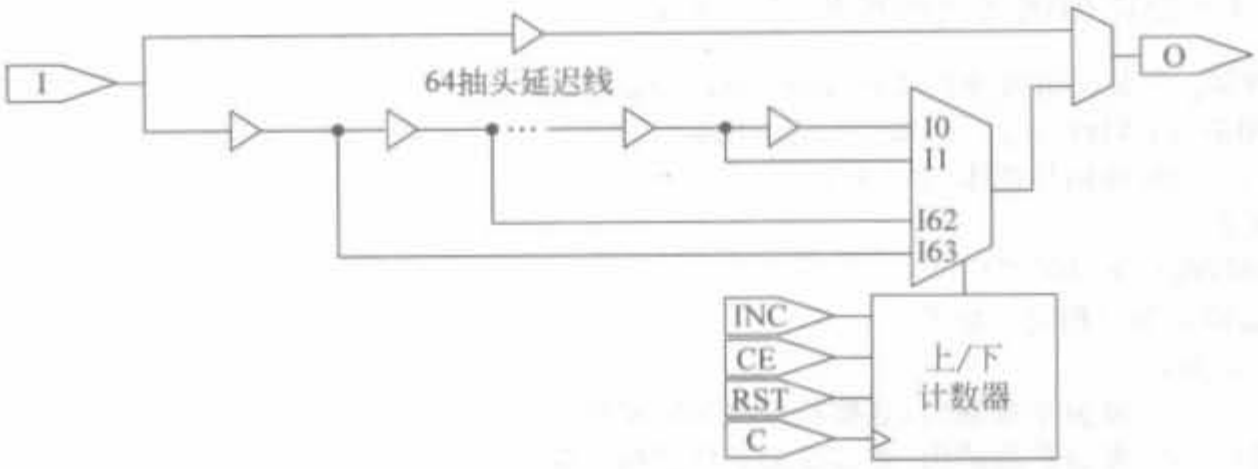


图 3-40 IDELAY 模块的 RTL 级结构图

(4) CE：单比特输入信号，使能延迟抽头数增加或减少的功能。

(5) RST：单比特输入信号，复位延迟链的延迟抽头数。如果没有编程输入，则为 0。

(6) O：单比特输出信号。

IDELAY 原语的例化代码模板如下所示：

```
// IDELAY: 输入延迟单元(Input Delay Element)
// 适用芯片：Virtex-Ⅱ/Ⅱ-Pro/4,Spartan-3/3E
// Xilinx HDL 库向导版本,ISE 9.1
IDELAY #(
.IOBDELAY_TYPE("DEFAULT"),
// 输入延迟类型,可设置为"DEFAULT"、"FIXED"或者"VARIABLE"
.IOBDELAY_VALUE(0)
// 输入延迟周期数,可设置为 0～63 之间的任意整数
) IDELAY_inst (
.O(O),     //1bit 输出信号
.C(C),     // 1bit 时钟输入信号
.CE(CE),   // 1bit 时钟使能信号
.I(I),     // 1bit 数据输入信号
.INC(INC),// 1bit 增量输入信号
.RST(RST) //1bit 复位输入信号
);
// 结束 IDELAY 模块的例化过程
```

在综合结果分析时，IDELAY 原语的 RTL 级结构如图 3-41 所示。

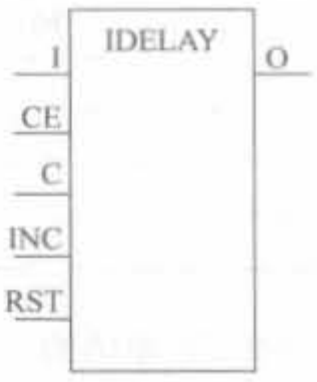


图 3-41 IDELAY 原语的 RTL 级结构图

4. OBUFDS 原语

OBUFDS 将标准单端信号转换成差分信号，输出端口需要直接对应到顶层模块的输出信号，和 IBUFDS 为一对互逆操作。OBUFDS 原语的真值表如表 3-13 所示。

表 3-13 OBUFDS 原语的真值表

输　入	输　出	
I	O	OB
0	0	1
1	1	0

OBUFDS 原语的例化代码模板如下所示：

```
// OBUFDS：差分输出缓冲器(Differential Output Buffer)
// 适用芯片：Virtex-Ⅱ/Ⅱ-Pro/4,Spartan-3/3E
// Xilinx HDL 库向导版本,ISE 9.1
OBUFDS #(
.IOSTANDARD("DEFAULT")
// 指定输出端口的电平标准
) OBUFDS_inst (
.O(O),     // 差分正端输出,直接连接到顶层模块端口
.OB(OB),   // 差分负端输出,直接连接到顶层模块端口
.I(I)      // 缓冲器输入
);
// 结束 OBUFDS 模块的例化过程
```

在综合结果分析时，OBUFDS 原语的 RTL 级结构如图 3-42 所示。

图 3-42　OBUFDS 的 RTL 级结构图

5. IOBUF 原语

IOBUF 原语是单端双向缓冲器，其 I/O 接口必须和指定的电平标准相对应，支持 LVTTL、LVCMOS15、LVCMOS18、LVCMOS25 以及 LVCMOS33 等信号标准，同时可通过 DRIVE、FAST 以及 SLOW 等约束来满足不同驱动和抖动速率的需求。默认的驱动能力为 12mA，低抖动。IOBUF 由 IBUF 和 OBUFT 两个基本组件构成，当 I/O 端口为高阻时，其输出端口 O 为不定态。IOBUF 原语的功能也可以通过其组成组件的互连来实现。

IOBUF 原语的输入、输出真值表如表 3-14 所示。

表 3-14　IOBUF 原语的真值表

输　入		双　向	输　出
T	I	I/O	O
1	X	Z	X
0	1	1	1
0	0	0	0

IOBUF 原语的例化代码模板如下所示：

```
// IOBUF：单端双向缓冲器(Single-ended Bi-directional Buffer)
// 适用芯片：所有芯片
// Xilinx HDL 库向导版本,ISE 9.1
IOBUF #(
.DRIVE(12),
// 指定输出驱动的强度
.IOSTANDARD("DEFAULT"),
// 指定 I/O 电平的标准,不同的芯片支持的接口电平可能不同
.SLEW("SLOW")
// 制定输出抖动速率
```

```
) IOBUF_inst (
.O(O),        // 缓冲器的单向输出
.I/O(I/O),    // 缓冲器的双向输出
.I(I),        // 缓冲器的输入
.T(T)         // 三态使能输入信号
);
// 结束 IOBUF 模块的例化过程
```

在综合结果分析时，IOBUF 原语的 RTL 级结构如图 3-43 所示。

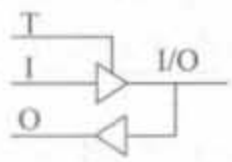


图 3-43　IOBUF 的 RTL 级结构图

6. PULLDOWN 和 PULLUP 原语

数字电路有 3 种状态：高电平、低电平和高阻状态。有些应用场合不希望出现高阻状态，可以通过上拉电阻或下拉电阻的方式使其处于稳定状态，如图 3-44 所示。FPGA 的 I/O 端口可以通过外接电阻上、下拉；也可以在芯片内部，通过配置完成上、下拉。上拉电阻用来解决总线驱动能力不足时提供电流的，而下拉电阻用来吸收电流。通过 FPGA 内部配置完成上、下拉，能有效节约电路板面积，是设计的首选方案。

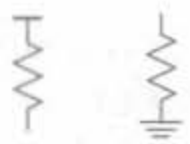


图 3-44　上、下拉电路示意图

上、下拉的原语分别为 PULLUP 和 PULLDOWN。

1) PULLUP 原语的例化代码

```
// PULLUP：上拉原语(I/O Buffer Weak Pull-up)
// 适用芯片：所有芯片
// Xilinx HDL 库向导版本，ISE 9.1
PULLUP PULLUP_inst (
.O(O),
//上拉输出，需要直接连接到设计的顶层模块端口上
// 结束 PULLUP 模块的例化过程
```

2) PULLDOWN 原语的例化代码

```
// PULLDOWN：下拉原语( I/O Buffer Weak Pull-down)
// 适用芯片：所有芯片
// Xilinx HDL 库向导版本，ISE 9.1
PULLDOWN PULLDOWN_inst (
.O(O),
// 下拉输出，需要直接连接到设计的顶层模块端口上
);
// 结束 PULLDOWN 模块的例化过程
```

3.4.6 处理器组件

处理器组件主要包括高速以太网 MAC 控制器和 PowerPC 硬核，如表 3-15 所示。

表 3-15 处理器组件列表

原　语	描　述
EMAC	集成的 10/100/1000Mb/s 以太网 MAC 层控制器
PPC405_ADV	PowerPC 核的原语

处理器组件是 Xilinx 的嵌入式解决方案，涉及软、硬件调试以及系统设计，将在第 9 章详细介绍。

3.4.7 RAM/ROM 组件

RAM/ROM 组件可用于例化 FIFO、分布式 RAM、块 RAM、分布式 ROM 以及块 ROM，具体包括 12 个组件，如表 3-16 所示。

表 3-16 RAM/ROM 原语列表

原　语	描　述
FIFO16	基于 Virtex-4 块 RAM 的内嵌 FIFO
RAM16×1D	深度为 16、位宽为 1 的静态同步双口 RAM
RAM16×1S	深度为 16、位宽为 1 的静态同步 RAM
RAM32×1S	深度为 32、位宽为 1 的静态同步 RAM
RAM64×1S	深度为 64、位宽为 1 的静态同步 RAM
RAMB16	单口块 RAM，位宽可配置成 1、2、4、9、18 或 36，其大小可配置成 16 384bit 的数据存储器，或者 2048bit 的奇偶存储器
RAMB32_S64_ECC	带有差错处理的深度为 64、位宽为 64 的同步双口块 RAM
ROM16×1	深度为 16、位宽为 1 的 ROM
ROM32×1	深度为 32、位宽为 1 的 ROM
ROM64×1	深度为 64、位宽为 1 的 ROM
ROM128×1	深度为 128、位宽为 1 的 ROM
ROM256×1	深度为 256、位宽为 1 的 ROM

下面主要介绍 FIFO、分布式双口 RAM 以及块双口 RAM 原语的使用。单口 RAM 和 ROM 原语的用法类似，限于篇幅，就不再介绍。

1. RAM16×1S

RAM16×1S 是深度为 16bit，位宽为 1 的同步 RAM。当写使能信号 WE 为低电平时，写端口的数据操作无效，RAM 内部的数据不会改变；当 WE 为高电平时，可以在任意地址中写入比特。为了保证数据的稳定性，地址和数据应该在 WCLK 的上升沿前保持稳定。输出信号 O 为 RAM 中由读地址信号所确定的地址中所存数据的值。此外，还可通过属性指定 RAM 的初始值。

RAM16×1S 原语的例化代码模板如下所示：

```
// RAM16×1S：16bit 1 深度同步 RAM
// 适用芯片：所有芯片
// Xilinx HDL 库向导版本，ISE 9.1
RAM16×1S #(
.INIT(16'h0000)
//对 RAM 的内容进行初始化。这里初始化为全 1
) RAM16×1S_inst (
.O(O),// RAM output
.A0(A0),     // RAM address[0] input
.A1(A1),     // RAM address[1] input
.A2(A2),     // RAM address[2] input
.A3(A3),     // RAM address[3] input
.D(D),       // RAM data input
.WCLK(WCLK),// Write clock input
.WE(WE)      // Write enable input
);
// 结束 RAM16×1S 模块的例化过程
```

需要注意的是，RAM16×1S 原语是 Xilinx 独有的一类结构，在小数据量存储方面非常节省资源。在综合结果分析时，RAM16×1S 原语的 RTL 级结构如图 3-45 所示。

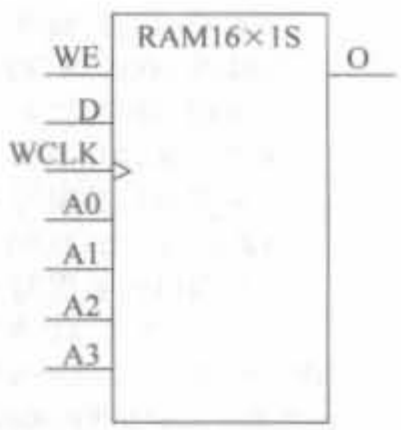


图 3-45 原语 RAM16×1S 的 RTL 级结构图

2. RAMB16

RAMB16 是 FPGA 芯片中内嵌的双口块 RAM，数据位宽可配置成 1、2、4、9、18 以及 36bit，每个块 RAM 的大小为 18×1024bit，所以位宽越大，深度越小。块 RAM 在 FPGA 中按照矩阵的方式排列，其数量完全取决于芯片容量的大小。在使用中，可以添加坐标来约束块 RAM 的位置。例如：

```
LOC = RAMB16_X#Y#;
```

同样，也可以对块 RAM 完成初始化。块 RAM 是以硬核的方式内嵌到 FPGA 芯片中的，不占用芯片的逻辑资源，是 FPGA 芯片内部非常宝贵的一种资源。在工作时，要尽量使用芯片的块 RAM 资源，这不仅能保证较高的工作频率，还具有很低的动态功耗。

RAMB16 的 Verilog 例化代码如下所示：

```
// RAMB16：块 RAM(Virtex-4 系列 18k 块 RAM
// 适用芯片：Virtex-4 芯片
// Xilinx HDL 库向导版本，ISE 9.1
RAMB16 #(
.DOA_REG(0),
// A 端口可选的输出寄存器，可设置为 0 或 1，分别表示输出不寄存或寄存
.DOB_REG(0),
// B 端口可选的输出寄存器，可设置为 0 或 1，分别表示输出不寄存或寄存
.INIT_A(36'h000000000),
// 初始化 A 端口的输出初始值
.INIT_B(36'h000000000),
// 初始化 A 端口的输出初始值
```

```
.INVERT_CLK_DOA_REG("FALSE"),
// 反转 A 端口寄存器的时钟,可设置为"TRUE"或"FALSE",分别表示不反转或反转
.INVERT_CLK_DOB_REG("FALSE"),
// 反转 B 端口寄存器的时钟,可设置为"TRUE"或"FALSE",分别表示不反转或反转
.RAM_EXTENSION_A("NONE"),
// RAM 的 A 端口扩展选项,可设置为"UPPER"、"LOWER"或"NONE",分别表示作为
// 高字节、低字节级联,或者不扩展
.RAM_EXTENSION_B("NONE"),
// RAM 的 B 端口扩展选项.READ_WIDTH_A(0),同 A 端口
.READ_WIDTH_A(0),
// A 端口的数据读取宽度,有效值为 1、2、4、9、18 或 36
.READ_WIDTH_B(0),
// B 端口的数据读取宽度,有效值为 1、2、4、9、18 或 36
.SIM_COLLISION_CHECK("ALL"),
// 碰撞检测使能信号,可设置为"ALL"、"WARNING_ONLY"、"GENERATE_X_ONLY"
// 或者"NONE"
.SRVAL_A(36'h000000000),
// 复位后 A 端口的输出值
.SRVAL_B(36'h000000000),
//复位后 A 端口的输出值
.WRITE_MODE_A("WRITE_FIRST"),
// A 端口的写模式,可设置为"WRITE_FIRST"、"READ_FIRST"或"NO_CHANGE"
.WRITE_MODE_B("WRITE_FIRST"),
// B 端口的写模式,可设置为"WRITE_FIRST"、"READ_FIRST"或"NO_CHANGE"
.WRITE_WIDTH_A(2),
// A 端口的数据写入宽度,有效值为 1、2、4、9、18 或 36
.WRITE_WIDTH_B(0),
// B 端口的数据写入宽度,有效值为 1、2、4、9、18 或 36
    // 下列 INIT_xx 声明用于指定 RAM 的初始化内容
.INIT_00(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_01(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_02(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_04(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_05(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_08(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_09(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_0A(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_0B(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_0C(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_0D(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_0E(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_0F(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_10(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_11(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_12(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_13(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_14(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_15(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_16(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_17(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_18(256'h0000000000000000000000000000000000000000000000000000000000000000),
```

```
.INIT_19(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_1A(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_1B(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_1C(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_1D(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_1E(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_1F(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_20(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_21(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_22(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_23(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_24(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_25(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_26(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_27(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_28(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_29(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2A(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2B(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2C(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2D(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2E(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2F(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_30(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_31(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_32(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_33(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_34(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_35(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_36(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_37(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_38(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_39(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_3A(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_3B(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_3C(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_3D(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_3E(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_3F(256'h0000000000000000000000000000000000000000000000000000000000000000),
    // 下面的 INITP_xx 用于初始化校验比特
.INITP_00(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INITP_01(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INITP_02(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INITP_03(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INITP_04(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INITP_05(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INITP_06(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INITP_07(256'h0000000000000000000000000000000000000000000000000000000000000000)
) RAMB16_inst (
    .CASCADEOUTA(CASCADEOUTA),     // A 端口的 1bit 级联输出
    .CASCADEOUTB(CASCADEOUTB),     // B 端口的 1bit 级联输出
    .DOA(DOA),                     // A 端口的 32bit 数据输出
```

```
    .DOB(DOB),                      // B端口的32bit数据输出
    .DOPA(DOPA),                    // A端口4bit校验数据输出
    .DOPB(DOPB),                    // B端口4bit校验数据输出
    .ADDRA(ADDRA),                  // A端口15bit地址输入
    .ADDRB(ADDRB),                  // B端口15bit地址输入
    .CASCADEINA(CASCADEINA),        // A端口1bit级联地址输入
    .CASCADEINB(CASCADEINB),        // B端口1bit级联地址输入
    .CLKA(CLKA),                    //A端口的工作时钟输入
    .CLKB(CLKB),                    // B端口的工作时钟输入
    .DIA(DIA),                      // A端口的32bit写数据
    .DIB(DIB),                      // B端口的32bit写数据
    .DIPA(DIPA),                    //A端口4bit校验输入数据
    .DIPB(DIPB),                    //B端口4bit校验输入数据
    .ENA(ENA),                      //A端口的1bit使能信号
    .ENB(ENB),                      //B端口的1bit使能信号
    .REGCEA(REGCEA),                //A端口1bit寄存器使能信号
    .REGCEB(REGCEB),                //B端口1bit寄存器使能信号
    .SSRA(SSRA),                    // A端口1bit复位输入信号
    .SSRB(SSRB),                    // B端口1bit复位输入信号
    .WEA(WEA),                      // A端口4bit写使能输入信号
    .WEB(WEB)                       // B端口4bit写使能输入信号
);
// 结束RAMB16模块的例化过程
```

在综合结果分析时，RAMB16的RTL级结构如图3-46所示。

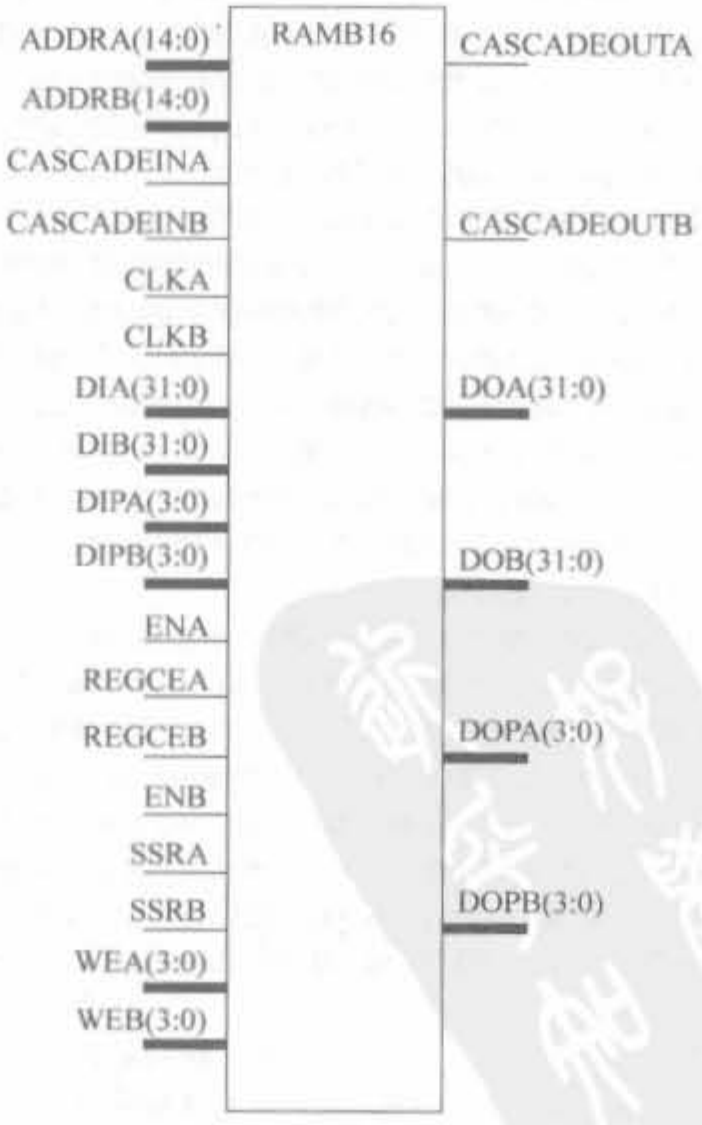


图3-46 RAMB16的RTL级结构示意图

3.4.8 寄存器和锁存器

寄存器和锁存器是时序电路中常用的基本元件，其原语如表 3-17 所示。

表 3-17 寄存器和锁存器原语列表

原语	描 述
FDCPE	带有时钟使能、异步预配置和清空信号的 D 触发器
FDRSE	带有同步时钟使能、同步预配置和清空信号的 RS 触发器
LCDPE	带有时钟使能、异步预配置和清空信号的透明数据锁存器

下面只介绍 FDCPE 原语的使用，其余两个原语的使用方法类似。

1. FDCPE 原语

FDCPE 指带单数据输入信号 D、单输出 O、时钟使能信号 CE、异步复位 PRE 和异步清空信号 CLR 的 D 触发器。当 PRE 信号为高电平时，输出端 O 为高电平；当 CLR 为高电平时，输出端 O 为低电平；当 PRE 和 CLR 都为低电平，CE 信号为高电平时，输入信号 D 在时钟上升沿被加载到触发器中，并被送到输出端；当 PRE 和 CLR 都为低电平，CE 信号为低电平时，输出端保持不变。FDCPE 的真值表如表 3-18 所示。

表 3-18 FDCPE 原语的真值表

输入信号					输出信号
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	保持不变
0	0	1	0	上升沿	0
0	0	1	1	上升沿	1

FDCPE 原语的 Verilog 实例化代码如下所示：

```
// FDCPE: D 触发器(Single Data Rate D Flip-Flop )
// 适用芯片：所有 FPGA 芯片
// Xilinx HDL 库向导版本,ISE 9.1
FDCPE #(
.INIT(1'b0)
//初始化寄存器的值,可设置为 1'b0 或 1'b1
) FDCPE_inst (
.Q(Q),          // 数据输出端口
.C(C),          // 时钟输入端口
.CE(CE),        // 时钟使能输入信号
.CLR(CLR),      // 异步清空输入信号
.D(D),          // 数据输入信号
.PRE(PRE)       // 异步复位输入信号
);
// 结束 FDCPE 模块的例化过程
```

在综合结果分析时，FDCPE 原语的 RTL 级结构如图 3-47 所示。

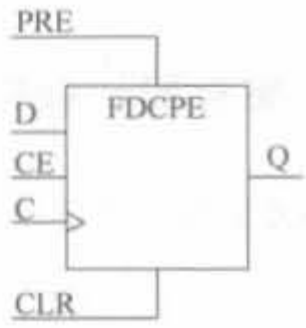


图 3-47　FDCPE 的 RTL 级结构图

3.4.9　移位寄存器组件

移位寄存器组件为 Xilinx 芯片所独有，充分利用了查找表的结构特点。由于属性不同，具体有 8 个移位寄存器组件，如表 3-19 所示。

表 3-19　移位寄存器组件列表

原语	描　述
SRL16	16bit 移位寄存器查找表
SRL16_1	在时钟下降沿作用的 16bit 移位寄存器查找表
SRL16E	带有时钟使能信号的 16bit 移位寄存器查找表
SRL16E_1	带有时钟使能信号，且在时钟下降沿作用的 16bit 移位寄存器查找表
SRLC16	带有进位的 16bit 移位寄存器查找表
SRLC16_1	带有进位，且在时钟下降沿作用的 16bit 移位寄存器查找表
SRLC6E	带有时钟使能和进位信号的 16bit 移位寄存器查找表
SRLC6E_1	带有时钟使能和进位信号，且在时钟下降沿作用的 16bit 移位寄存器查找表

各个移位寄存器原语都是在 SRL16 的基础上发展起来的，因此，本节主要介绍基本的 SRL16 原语。

1. SRL16 原语

SRL16 是基于查找表(LUT)的移位寄存器，在实际应用中既能节省资源，还能保证时序。其输入信号 A3、A2、A1 以及 A0 选择移位寄存器的读取地址，当写使能信号高电平有效时，输入信号将被加载到移位寄存器中。单个移位寄存器的深度可以是固定的，也可以动态调整，最大不能超过 16。需要更大深度的移位寄存器时，要将多个 SRL16 拼接起来。单个 SRL16 的地址信号线与输出的真值表如表 3-20 所示。

表 3-20　SRL16 的功能说明表

输　入			输　出
Am	CLK	D	Q
Am	X	X	Q(Am)
Am	↑(上升沿)	D	Q(Am−1)

注：m=0,1,2,3。

SRL16 本质上是一种基于查找表(LUT)的移位寄存器，因此其位宽(B)和深度(D)需要满足查找表的结构特点，所占用 Slice 资源 M 的计算公式为

$$M = B(R[D/16]+1)$$

其中，R[]函数的意义是取整。从上式可以看出，深度为 1 的移位寄存器和深度为 16 的移位寄存器所占用的 Slice 资源是一样的。和触发器相比，SRL16 最大的特点就是占用 Slice 资源特别少。

只要固定 A3～A0 的信号电平，即可获得一个固定长度的移位寄存器，其长度范围为

1～16，可由公式(M)计算得到。当 A3～A0 全为 0 时，其寄存深度为 1；当 A3～A0 全为 1 时，其寄存深度为 16。

$$\text{长度}=(8\times A3)+(4\times A2)+(2\times A1)+A0+1 \qquad (M)$$

SRL16 原语的 Verilog 实例化代码如下所示：

```
// SRL16：16 位查找表移位寄存器(16-bit shift register LUT operating on posedge of clock)
// 适用芯片：所有 FPGA 芯片
// Xilinx HDL 库向导版本，ISE 9.1
SRL16 #(
.INIT(16'h0000)
// 初始化移位寄存器的值，可以为 16bit 任意整数
) SRL16_inst (
.Q(Q),       // SRL16 的数据输出信号
.A0(A0),     // 选择[0]输入
.A1(A1),     // 选择[1]输入
.A2(A2),     // 选择[2]输入
.A3(A3),     // 选择[3]输入
.CLK(CLK),   // 时钟输入信号
.D(D)        // SRL16 的数据输入信号
);
// 结束 SRL16 模块的例化过程
```

在综合结果分析时，SRL16 原语的 RTL 级结构如图 3-48 所示。

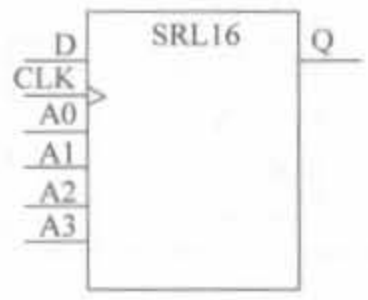


图 3-48　SRL16 的 RTL 级结构示意图

3.4.10　Slice/CLB 组件

Slice/CLB 组件涵盖了 Xilinx FPGA 中所有的逻辑单元，包括各种查找表、复用器以及逻辑操作等 21 个原语，如表 3-21 所示。

表 3-21　Slice/CLB 组件列表

原　语	描　述
BUFCF	快速连接缓冲
LUT1	带通用输出的 1bit 查找表
LUT2	带通用输出的 2bit 查找表
LUT3	带通用输出的 3bit 查找表
LUT4	带通用输出的 4bit 查找表
LUT1_D	带两个输出的 1bit 查找表
LUT2_D	带两个输出的 2bit 查找表
LUT3_D	带两个输出的 3bit 查找表
LUT4_D	带两个输出的 4bit 查找表
LUT1_L	带本地输出的 1bit 查找表
LUT2_L	带本地输出的 2bit 查找表
LUT3_L	带本地输出的 3bit 查找表
LUT4_L	带本地输出的 4bit 查找表

续表

原　语	描　述
MULT_AND	用于乘法的快速与门
MUXCY	带有进位和通用输出的 2 到 1 复用器
MUXCY_D	带有进位和双输出的 2 到 1 复用器
MUXCY_L	带有进位和本地输出的 2 到 1 复用器
MUXF5	基于查找表的 2 到 1 复用器,带通用输出
MUXF5_D	基于查找表的 2 到 1 复用器,带双输出
MUXF5_L	基于查找表的 2 到 1 复用器,带本地输出
MUXF6	基于查找表的 2 到 1 复用器,带通用输出
MUXF6_D	基于查找表的 2 到 1 复用器,带双输出
MUXF6_L	基于查找表的 2 到 1 复用器,带本地输出
MUXF7	基于查找表的 2 到 1 复用器,带通用输出
MUXF7_D	基于查找表的 2 到 1 复用器,带双输出
MUXF7_L	基于查找表的 2 到 1 复用器,带本地输出
MUXF8	基于查找表的 2 到 1 复用器,带通用输出
MUXF8_D	基于查找表的 2 到 1 复用器,带双输出
MUXF8_L	基于查找表的 2 到 1 复用器,带本地输出
XORCY	带通用输出进位逻辑的 XOR
XORCY_D	带两个输出进位逻辑的 XOR
XORCY_L	带本地输出进位逻辑的 XOR

本节主要介绍基本的单输出 1bit 查找表原语和通用输出 2 到 1 复用器原语的使用方法。

1. LUT1 原语

FPGA 内部的组合逻辑都可以通过 LUT 结构实现。常用的 LUT 结构有 LUT1、LUT2、LUT3 以及 LUT4,其区别在于查找表输入比特宽度的不同。LUT1 是最简单的一种,常用于实现缓冲器和反相器,是带通用输出的 1bit 查找表。

LUT1 原语的 Verilog 实例化代码如下所示:

```
// LUT1: 1bit 输入的通用查找表(1-input Look-Up Table with general output)
// 适用芯片: 所有 FPGA 芯片
// Xilinx HDL 库向导版本,ISE 9.1
LUT1 #(
.INIT(2'b00)
// 指定 LUT 的初始化内容
) LUT1_inst (
.O(O),     // LUT 的通用输出
.I0(I0)    // LUT 输入信号
);
// 结束 LUT1 模块的例化过程
```

在综合结构分析时,LUT1 原语的 RTL 级结构如图 3-49 所示。

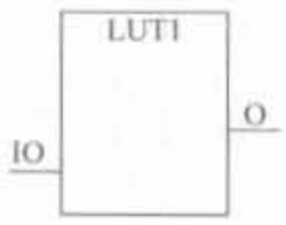


图 3-49　LUT1 的 RTL 级结构示意图

2. MUXF7 原语

MUXF7 是带通用输出的 2 到 1 查找表复用器，其控制信号 S 可以是任何内部信号。当其为低电平时，选择 I0；当其为高电平时，选择 I1。MUXF7 需要占用一个完整的 CLB 资源，和其相关的查找表可组成 7 功能的查找表和 16 到 1 的选择器，可将 MUXF6 的输出连接到 MUXF7 的输入端。MUXF7 的真值表如表 3-22 所示。

表 3-22　MUXF7 的真值表

输入			输出
S	I0	I1	O
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

MUXF7 原语的实例化代码如下所示：

```
// MUXF7：2 到 1LUT 复用器(CLB MUX to tie two MUXF6's together with general output)
// 适用芯片：Virtex-Ⅱ/Ⅱ-Pro 以及 Spartan-3/3E
// Xilinx HDL 库向导版本，ISE 9.1
MUXF7 MUXF7_inst (
.O(O),        // MUX 的通用输出信号
.I0(I0),
.I1(I1),      // 复用器的输入信号，连接到 MUXF6 LO
.S(S)         // 复用器的输入选择信号
);
// End of MUXF7_inst instantiation
```

图 3-50　MUXF7 的 RTL 级结构示意图

在综合结构分析时，MUXF7 原语的 RTL 级结构如图 3-50 所示。

本节给出了典型的 Xilinx 原语的使用，更多的原语需要读者在实践中大量应用才能熟练掌握。只有全面掌握了原语，才能对 Xilinx FPGA 的资源有一个理性认识，从而将工程设计提升到艺术雕刻的高度，这是每一个 FPGA 开发人员的目标。详细的原语资料可在 ISE 安装目录下的“doc\usenglish\ books\docs”文件夹中找到，其中 Virtex-4 系列芯片的原语资料文件“v4ldl. pdf”位于“v4ldl”文件中，Virtex-5 系列芯片的原语资料文件“v5ldl. pdf”位于“v5ldl”文件中，Spartan-3A 系列芯片的原语资料文件“s3adl. pdf”位于“s3adl”文件中，Spartan-3E 系列芯片的原语资料文件“s3edl. pdf”位于“s3edl”文件中。

3.5　本章小结

本章主要介绍基于 Xilinx 芯片的 Verilog 开发技术。首先介绍了典型的硬件设计思维；然后讨论了优秀的通用代码风格和 Xilinx 专用的代码风格，无论哪种风格，都是稳定性、速度以及硬件资源三者的优化，只不过前者从 FPGA 器件共有的特性出发，后者面向 Xilinx 公司的 FPGA 芯片特点；最后，介绍了 Xilinx 公司 FPGA 器件的硬件原语，以及典型原语的使用方法。特别需要强调的一点是，对原语的全面掌握是对高级 Xilinx FPGA 开发人员的一项基本要求。

第4章

ISE开发环境使用指南

CHAPTER 4

ISE Foundation 软件是 Xilinx 公司推出的 FPGA/CPLD 集成开发环境，不仅包括逻辑设计所需的一切，还具有大量简便易用的内置式工具和向导，使得 I/O 分配、功耗分析、时序驱动设计收敛、HDL 仿真等关键步骤变得容易而直观。全面掌握 ISE，是使用 Xilinx 公司 FPGA 芯片完成逻辑设计的人员所必备一项基本技能。本章简要介绍 ISE 的基本操作和开发流程，目的在于介绍一些入门知识，更多的技巧和经验需要读者在大量实践中逐步掌握。

4.1 ISE 套件的介绍与安装

4.1.1 ISE 简要介绍

Xilinx 是全球领先的可编程逻辑完整解决方案的供应商，它研发、制造并销售应用范围广泛的高级集成电路、软件设计工具以及定义系统级功能的 IP(Intellectual Property)核，长期以来，它一直推动着 FPGA 技术的发展。Xilinx 的开发工具不断升级，由早期的 Foundation 系列逐步发展到目前的 ISE 9.1i 系列，集成了 FPGA 开发需要的所有功能，其主要特点有：

- 包含了 Xilinx 新型 SmartCompile 技术，可以将实现时间缩减 2.5 倍，能在最短的时间内提供最高的性能，提供了一个功能强大的设计收敛环境；
- 全面支持 Virtex-5 系列器件(业界首款 65nm FPGA)；
- 集成式的时序收敛环境有助于快速、轻松地识别 FPGA 设计的瓶颈；
- 可以节省一个或多个速度等级的成本，并可在逻辑设计中实现最低的总成本。

Foundation Series ISE 具有界面友好、操作简单的特点，再加上 Xilinx 的 FPGA 芯片占有很大的市场，使其成为非常通用的 FPGA 工具软件。ISE 作为高效的 EDA 设计工具集合，与第三方软件扬长补短，使软件功能越来越强大，为用户提供了更加丰富的 Xilinx 平台。

4.1.2 ISE 功能简介

ISE 的主要功能包括设计输入、综合、仿真、实现和下载，涵盖了 FPGA 开发的全过程。从功能上讲，其工作流程无需借助任何第三方 EDA 软件。

1）设计输入：ISE 提供的设计输入工具包括用于 HDL 代码输入和查看报告的 ISE 文本编辑器(The ISE Text Editor)，用于原理图编辑的工具 ECS(The Engineering Capture System)，用于生成 IP Core 的 Core Generator，用于状态机设计的 StateCAD 以及用于约束文件编辑的 Constraint Editor 等。

2）综合：ISE 的综合工具不但包含了 Xilinx 自身提供的综合工具 XST，还可以内嵌 Mentor Graphic 公司的 LeonardoSpectrum 和 Synplicity 公司的 Synplify，实现无缝链接。

3）仿真：ISE 本身自带了一个具有图形化波形编辑功能的仿真工具 HDL Bencher，同时提供使用 Mentor Graphic 公司的 ModelSim 进行仿真的接口。

4）实现：此功能包括翻译、映射、布局布线等，还具备时序分析、管脚指定以及增量设计等高级功能。

5）下载：下载功能包括 BitGen，用于将布局布线后的设计文件转换为位流文件；还包括 IMPACT，功能是进行设备配置和通信，控制将程序烧写到 FPGA 芯片中去。

使用 ISE 进行 FPGA 设计的各个过程可能涉及的设计工具如表 4-1 所示。

表 4-1 ISE 设计工具表

设计输入	综合	仿真	实现	下载
HDL 文本编辑器 ECS 原理图编辑器 StateCAD 状态机编辑器 Core Generator Constraint Editor	XST FPGA Express (Synplify 和 LeonardoSpectrum)	HDL Bencher (ModelSim)	Translate MAP Place and Route Xpower	BitGen IMPACT

4.1.3 ISE 软件的安装

ISE 9.1 软件安装的基本硬件要求如下：CPU 在 P Ⅲ以上，内存大于 256MB，硬盘大于 4GB 的硬件环境安装。为了更好地使用软件，至少需要 512MB 内存，CPU 的主频在 2GHz 以上。本书使用的集成开发环境是 ISE 9.1，仿真工具是 ModelSim 6.2b，综合工具为 Synplify Pro 8.8。其中 ISE、ModelSim 软件和 Synplify 软件的不同版本之间的差异不是很大，所以操作和设计结果的差别是很小的。具体安装过程如下：

1）将光盘放进 DVD 光驱，等待其自动运行(如果没有自动运行，直接执行光盘目录下的 Setup.exe 文件程序即可)，将弹出如图 4-1 所示的欢迎界面，单击 Next 进入下一页。

2）进入注册码获取、输入对话框，如图 4-2 所示。注册码可以通过网站、邮件和传真方式申请。如果已有注册码，输入后单击 Next 按钮继续。

购买了正版软件后，最常用的方法就是通过网站注册获取安装所需的注册码。首先在 Xilinx 的官方主页 www.xilinx.com 上建立自己的账号，然后单击图 4-2 中的“Website”按钮，登录账号，输入 CD 盒上的产品序列号(序号的格式为 3 个字符＋9 个数字)，会自动生成 16 位注册码，直接记录下来即可，同时 Xilinx 网站会将注册码的详细信息发送到账号所对应的邮箱中。如果安装试用版，直接安装即可，无需任何操作，但是只能使用该软件 60 天。

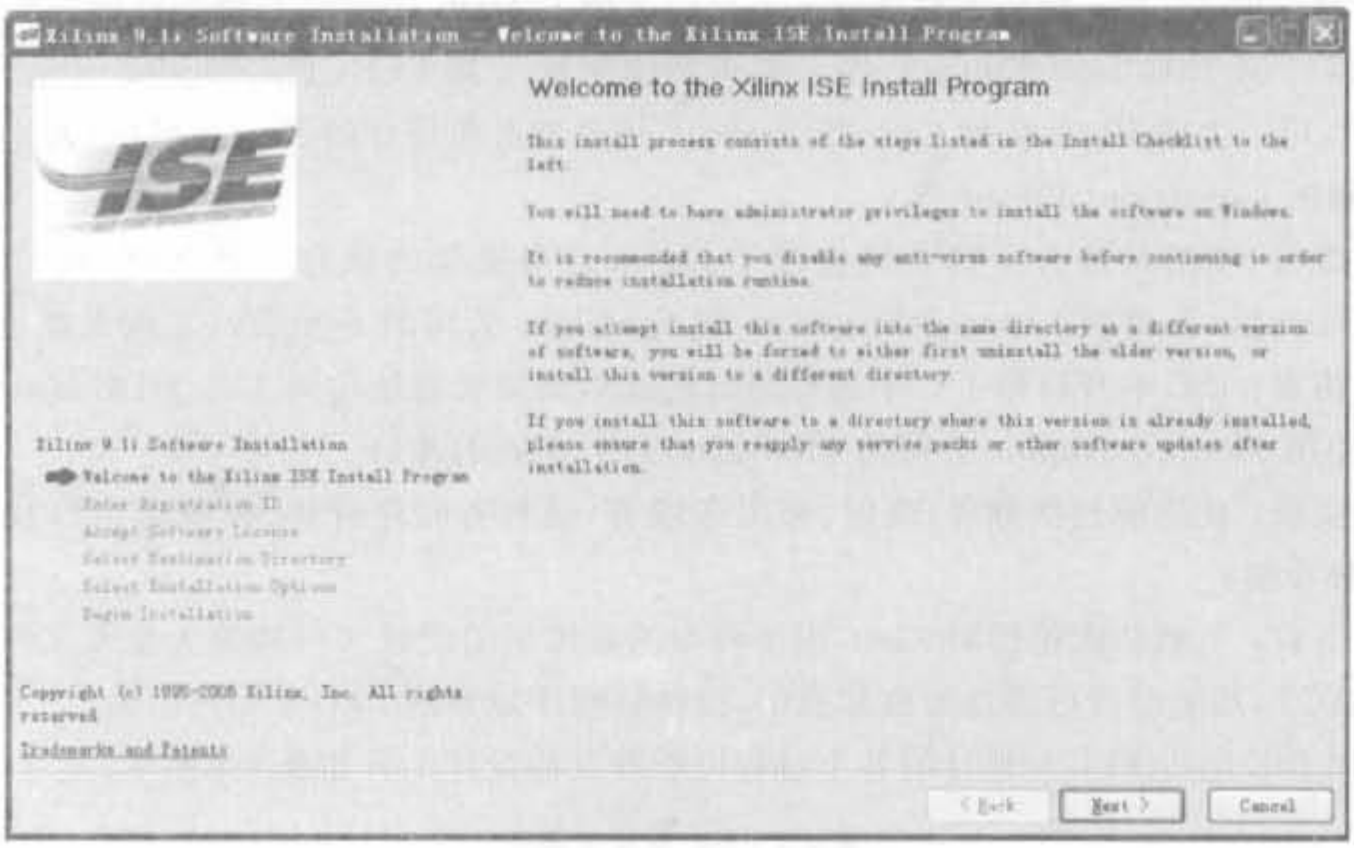


图 4-1　ISE 安装过程的欢迎界面

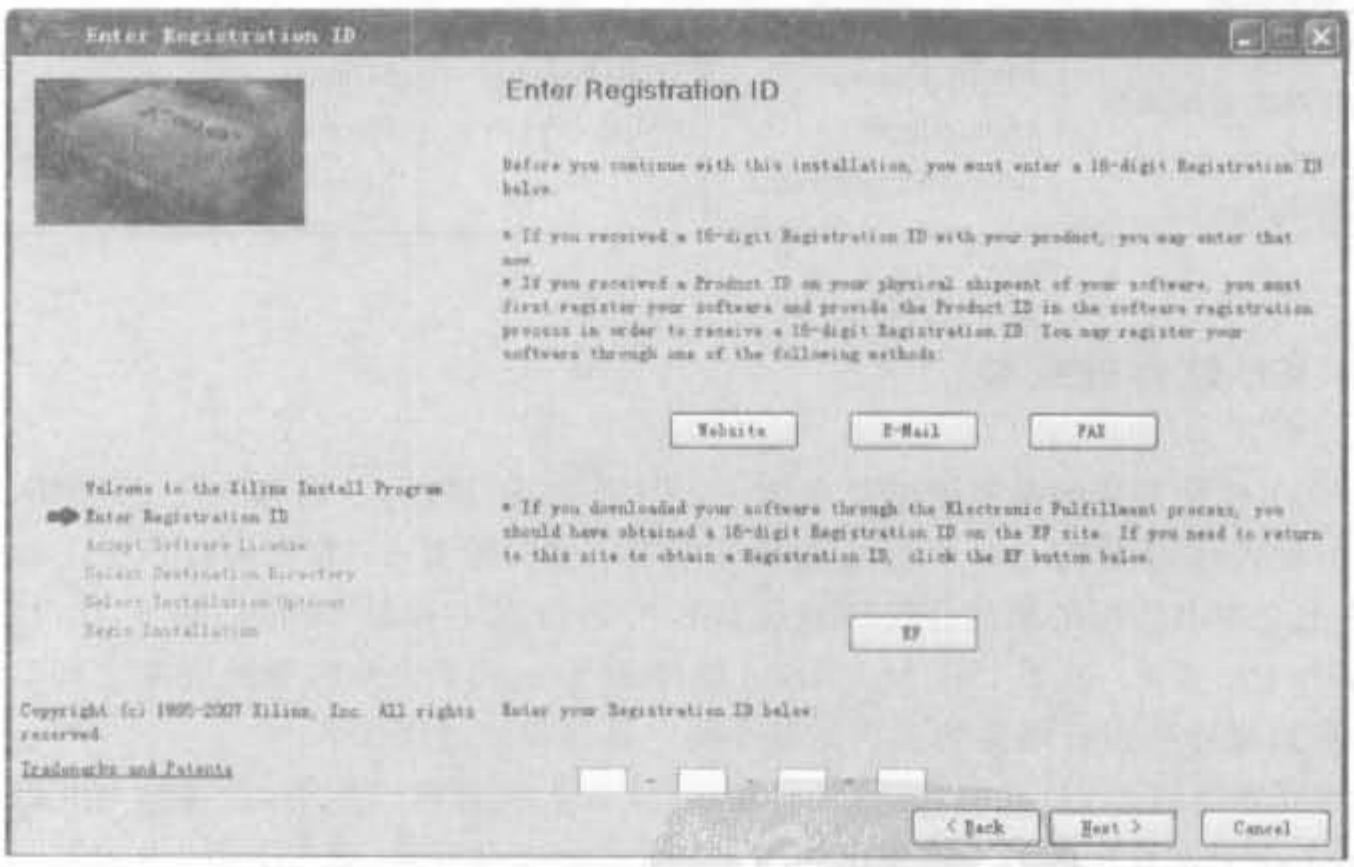


图 4-2　ISE 9.1 安装程序的注册码输入界面

3）下一个对话框是 Xilinx 软件的授权声明对话框，选中 I accept the terms of this software license，单击 Next 后进入安装路径选择界面，如图 4-3 所示。单击 Browse 按钮后选择自定义安装路径，单击 Next 按钮继续。

4）接下来的几个对话框分别是安装组件选择，如图 4-4 所示，用户需要选择自己使用的芯片所对应的模块，这样才能在开发中使用这些模块。在计算机硬盘资源不紧张的情况下，通常选择 Select All。

图 4-3 ISE 软件安装路径选择对话框

图 4-4 ISE 安装组件选择界面

5）进入设置环境变量页面，保持默认即可。如果环境变量设置错误，则安装后不能正常启动 ISE。选择默认选项，安装完成后，在“我的电脑”上单击右键，选择“属性”→“环境变量”，可看到名为 Xilinx 的环境变量，其值为安装路径。最后进入安装确认对话框，单击 Install 按钮，即可按照用户的设置自动安装 ISE，如图 4-5 所示。

6）安装完成后，会在桌面以及程序菜单中添加 Project Navigator 的快捷方式，双击即可进入 ISE 集成开发环境。

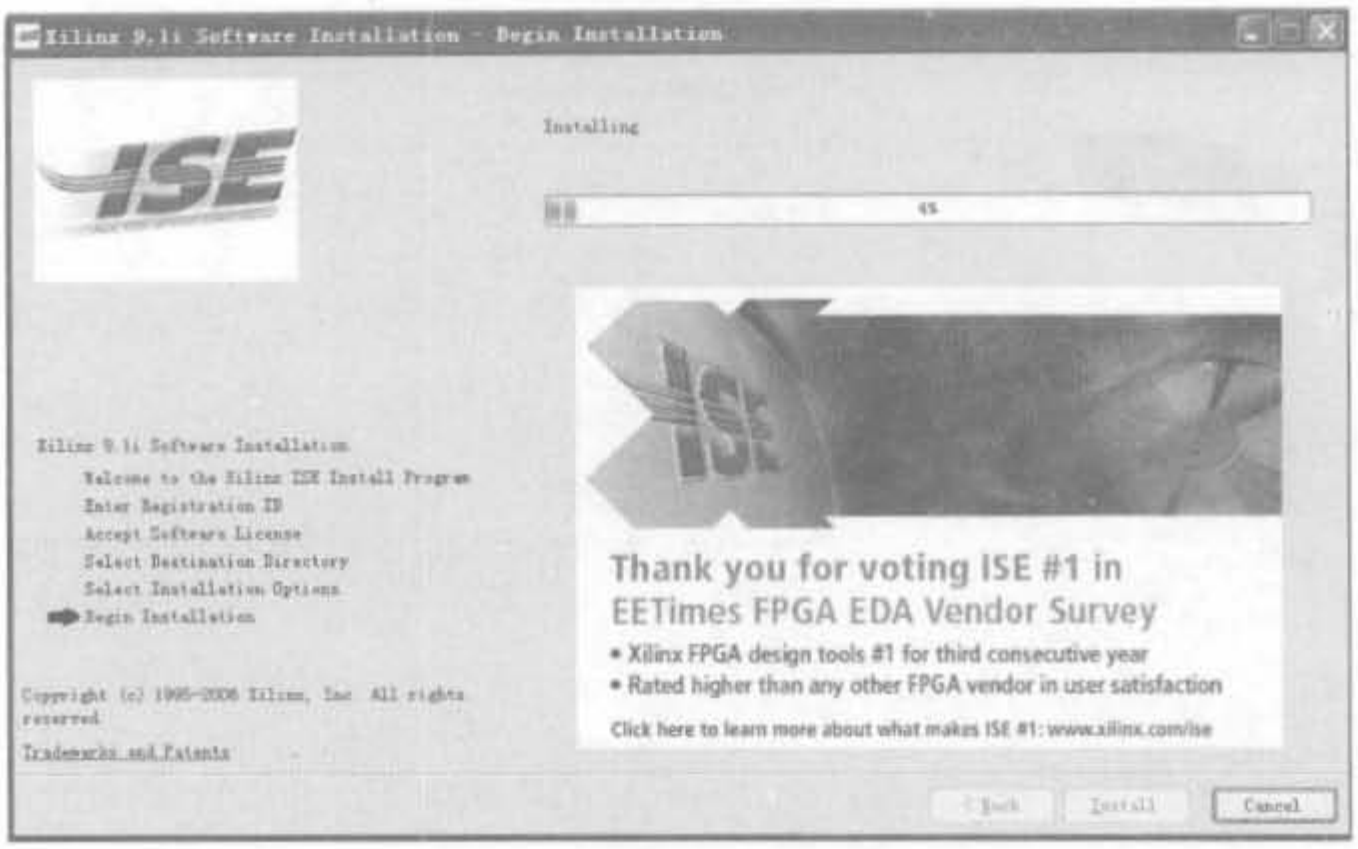


图 4-5　ISE 安装进程示意图

4.1.4　ISE 软件的基本操作

1. ISE 用户界面

ISE 9.1i 的界面如图 4-6 所示，由上到下主要分为标题栏、菜单栏、工具栏、工程管理区、源文件编辑区、过程管理区、信息显示区、状态栏 8 个部分。

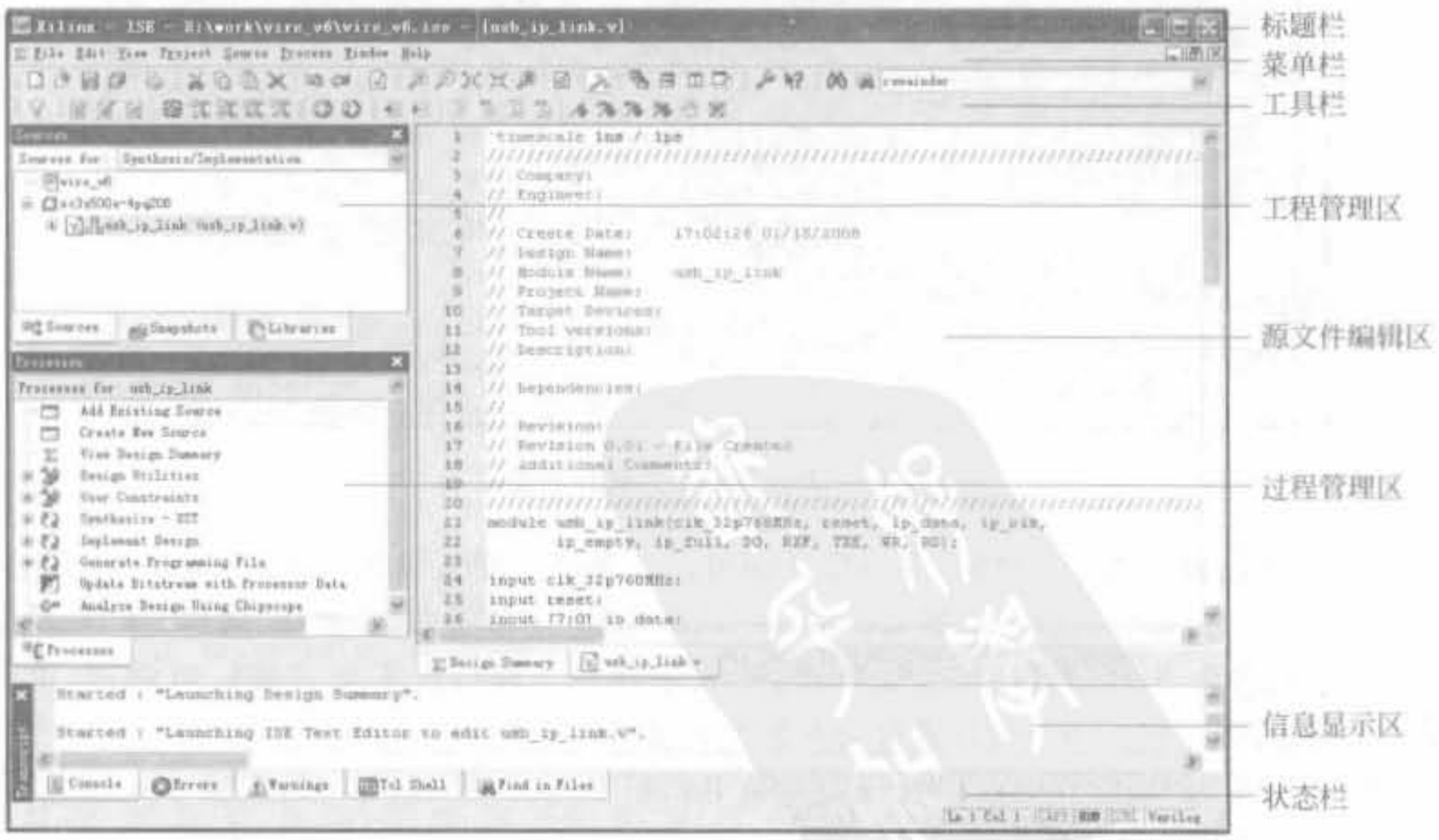


图 4-6　ISE 的主界面

(1) 标题栏：主要显示当前工程的名称和当前打开的文件名称。

(2) 菜单栏：主要包括“文件(File)”、“编辑(Edit)”、“视图(View)”、“工程(Project)”、“源文件(Source)”、“操作(Process)”、“窗口(Window)”和“帮助(Help)”8 个下拉菜单。其使用方法和常用的 Windows 软件类似。

(3) 工具栏：主要包含常用命令的快捷按钮。灵活运用工具栏可以极大地方便用户在 ISE 中的操作。在工程管理中，此工具栏的运用极为频繁。

(4) 工程管理区：提供了工程及其相关文件的显示和管理功能，主要包括源文件视图(Source View)、快照视图(Snapshot View)和库视图(Library View)。其中，源文件视图比较常用，它显示了源文件的层次关系。快照是当前工程的备份，设计人员可以随时备份，也可以将当前工程随时恢复到某个备份状态。快照视图用于查看当前工程的快照。执行快照功能的方法是选择菜单项 Project | Take Snapshot。库视图则显示了在工程中用户产生的库内容。

(5) 源文件编辑区：源文件编辑区提供了源代码的编辑功能。

(6) 过程管理区：本窗口显示的内容取决于工程管理区中所选定的文件。相关操作和 FPGA 设计流程紧密相关，包括设计输入、综合、仿真、实现和生成配置文件等。对某个文件进行了相应的处理后，在处理步骤的前面会出现一个图标来表示该步骤的状态。

(7) 信息显示区：显示 ISE 中的处理信息，如操作步骤信息、警告信息和错误信息等。信息显示区的下脚有两个标签，分别对应控制台信息区(Console)和文件查找区(Find in Files)。如果设计出现了警告和错误，双击信息显示区的警告和错误标志，就能自动切换到源代码出错的地方。

(8) 状态栏：显示相关命令和操作的信息。

2. ISE 菜单的基本操作

ISE 所有的操作都可通过菜单完成，下面简要介绍 ISE 的菜单命令以及功能。

1) File 菜单

File 菜单的命令包括 New Project、Open Project、Open Examples、Close Project、Save Project As、New、Open、Save、Save As、Save All、Print Preview、Print、Recent Files、Recent Projects 以及 Exit 等。

(1) New Project 命令：用于新建工程，是开始设计的第一步。ISE 会为新建的工程创建一个和工程同名的文件夹，专门用于存放工程的所有文件。

(2) Open Project 命令：用于打开已有的 ISE 工程。高版本的 ISE 可以打开低版本的工程，但需要版本转换，该转换是单向的、不可逆的，因此需要做好版本备份。低版本的 ISE 不能打开高版本的 ISE 工程。

(3) Open Examples 命令：用于打开 ISE 提供的各种类型的示例。

(4) Close Project 命令：关闭当前工程。如果关闭前未保存文件，ISE 会提示用户保存后再退出。

(5) Save Project As 命令：可将整个工程另存为其他名字的工程，在大型开发中，常使用该命令来完成版本备份。

(6) New 命令：用于新建源文件，可生成原理图、符号以及文本文件。文本文件另存时可修改其后缀名，以生成. v 或. vhd 的源文件。

(7) Open 命令：用于打开所有 Xilinx 所支持的文件格式，便于用户查看各类文件资源。

(8) Save、Save As 以及 Save All 命令：分别用于保存当前源文件、另存为当前源文件以及保存所有源文件。用户要在开发当中养成及时保存文件的习惯，避免代码丢失。

(9) Print Preview 命令：用于打印预览当前文件，Print 用于打印当前文件。

(10) Recent Files 命令：用于查看最近打开的文件。

(11) Recent Projects 命令：用于查看最近打开的工程。

(12) Exit 命令：用于退出 ISE 软件。

2) Edit 菜单

Edit 菜单的命令包括 Undo、Redo、Cut、Copy、Paste、Delete、Find、Find Next、Find in Files、Language Templates、Select All、Unselect All、Message Filters、Object Properties 以及 Preference 等。大多数命令用于源代码开发。只介绍了大多数，因此，Object Properties 不用介绍。

(1) Undo 命令：用于撤销当前操作，返回到前一状态。

(2) Redo 命令：是“Undo”命令的逆操作，用于恢复被撤销的操作。

(3) Cut 命令：剪贴选中的代码，快捷键为“Crtl+X”。

(4) Copy 命令：复制选中的代码，快捷键为“Crtl+C”。

(5) Paste 命令：粘贴剪贴和复制的代码，快捷键为“Crtl+V”。

(6) Delete 命令：删除选中的代码。

(7) Find 命令：查找选中的文字，或寻找在其输入框中输入的内容，快捷键为“Crtl+F”。

(8) Find Next 命令：寻找下一个要查找的内容，并跳至相应的位置，快捷键为“F3”。

(9) Language Templates 命令：可打开语言模板，里面有丰富的学习资料，是非常完整的 HDL 语言帮助手册，其地位类似于 Visual C++的 MSDN。

(10) Select All 命令：选中所有的代码，其快捷键为“Crtl+A”。

(11) Unselect All 命令：撤销已选中的全部代码，是 Select All 的逆操作。

(12) Message Filters 命令：过滤消息，只显示用户期望的消息。

(13) Preference 命令：用于设定 ISE 的启动参数以及运行参数，有着众多的设置项，最常用的就是第三方 EDA 软件的关联设置，将在第 4.5 节详细介绍。

3) View 菜单

View 菜单主要管理 ISE 软件的视图，不涉及 FPGA 开发中的任何环节，其中常用的命令有 Layout Horizontally、Layout Vertically 以及 Restore Default Layout。

(1) Layout Horizontally 命令：将水平地排列 ISE 主界面中的过程管理区、过程管理区以及代码编辑区等主要栏目。

(2) Layout Vertically 命令：将垂直地排列 ISE 主界面中的过程管理区、过程管理区以及代码编辑区等主要栏目。

(3) Restore Default Layout 命令：将恢复 ISE 默认的主界面布局。

4) Project 菜单

Project 菜单包含了对工程的各个操作，是设计中最常用的菜单之一，包括 New

Source、Add Source、Add Copy of Source、Cleanup Project Files、Toggle Paths、Archive、Take Snapshot、Make Snapshot Current、Apply Project Properties 以及 Source Control 命令。

(1) New Source 命令：用于向工程中添加源代码，可以添加 HDL 源文件、IP Core 以及管脚和时序约束文件。

(2) Add Source 命令：将已有的各类源代码文件加入到工程中，Verilog 模块的后缀为. v，VHDL 模块的后缀为. vhd，IP Core 源文件为. xco 文件或. xaw 文件，约束文件的后缀为. ucf。

(3) Add Copy of Source 命令：将目标文件复制一份添加到工程中。

(4) Cleanup Project Files 命令：用于清空综合和实现过程所产生的文件和目录。如果在 EDIF 设计模式中，只清空实现过程所产生的文件。

(5) Toggle Paths 命令：用于显示或隐藏非工程文件夹中的远端源文件的路径。

(6) Archive 命令：用于压缩当前工程，包括所有的文件。默认压缩类型为. zip 格式。

(7) Take Snapshot 命令：用于产生一个工程快照，即当前目录和远程资源的一个只读记录，常用于版本控制。

(8) Make Snapshot Current 命令：用户恢复快照覆盖当前工程。由于该命令会将当前工程删除，所以使用前一定要做好数据备份工作。

(9) Apply Project Properties 命令：应用工程属性，会提示用于选择相应的工程。

(10) Source Control 命令：常用于代码的导入和导出，有“Export”和“Import”两个子命令。

5) Source 菜单

Source 菜单主要面向工程管理区，包含了对资源文件的各个操作，每个命令的操作也都可以在工程管理区单击鼠标右键弹出的对话框中单击实现，包括 Open、Set as Top Module、Use SmartGuide、New Partition、Delete Partition、Partition Properties、Partition Force、Remove、Move to Library 以及 Properties 等命令。

(1) Open 命令：可打开所有类型的源文件，包括. v、. vhd、. xco、. xaw 以及. ucf 等格式。

(2) Set as Top Module 命令：用于将选中的文件设置成顶层模块。只有设置成顶层模块，才能对其综合、实现以及生成相应的二进制比特流文件。

(3) Use SmartGuide 命令：允许用户在本次实现时利用上一次实现的结果，包括时序约束以及布局布线结果，可节省实现的时间，但前提是工程改动不大。

(4) New Partition 命令：新建分区，常用于区域约束。

(5) Delete Partition 命令：删除区域约束的分区。

(6) Partition Properties 命令：可设置分区属性，详细说明可参考 4.4.4 节。

(7) Partition Force 命令：包含“Force Synthesis Out-of-data”和“Force Implement Design Out-of-data”两个指令，分别用于分区综合和增量设计。

(8) Remove 命令：把选中的文件从工程中删除，但仍保留在计算机硬盘上。

(9) Move to Library 命令：将选中的源文件移动到相应的库中，以便建立用户文件库。

(10) Properties 命令：查看源文件属性，有 Synthesis/Implementation Only、Simulation

Only 以及 Synthesis/Imp+Simulation 3 种类型。其中，Simulation Only 类文件只能仿真，不能被综合。

6) Process 菜单

Process 菜单包含了工程管理区的所有操作，每个命令的操作也都可以在过程管理区单击相应的图标实现，包括 Inmolement Top Module、Run、Rerun、Rerun All、Stop、Open Without Updating 以及 Properties 等命令。

(1) Inmolement Top Module 命令：完成顶层模块的实现过程。

(2) Run 命令：在工程过程栏选中不同的操作，单击该命令，可分别启动综合、转换、映射、布局布线等过程。

(3) Rerun 命令：重新运行“Run”指令执行的内容。

(4) Rerun All 命令：重新运行所有“Run”指令执行的内容。

(5) Stop 命令：停止当前操作，可中止当前操作，包括综合和实现的任一步骤。

(6) Open Without Updating 命令：该指令用于打开相应上一次完成的综合或实现过程所产生的文件。

(7) Properties 命令：在工程过程栏选中不同的操作，单击该命令，可设置不同阶段的详细参数。

7) Window 菜单

Window 菜单的主要功能是排列所有窗口，使其易看、易管理。通过本菜单可以看到当前打开的所有窗口，并能直接切换到某个打开的窗口。由于各命令操作简单，不再介绍。

8) Help 菜单

Help 菜单主要提供 ISE 所有帮助以及软件管理操作，包括 Help Topics、Software Manuals、Xilinx on the Web、Tutorials、Update Software Product Configuration、Tip of the Day、WebUpdata 以及 About 命令。

(1) Help Topics 命令：单击后，将自动调用 IE 浏览器打开 ISE 的帮助文档。

(2) Software Manuals 命令：单击后，将自动打开 PDF 文件，通过超链接到用户感兴趣的软件使用文档，其内容比网页形式的帮助文档要丰富。

(3) Xilinx on the Web 命令：包括完整的 Xilinx 网络资源，可根据需要单击查看链接。

(4) Tutorials 命令：包括本地快速入门 ISE 的说明文档和 Xilinx 网站的入门教学内容，可单击查看。

(5) Update Software Product Configuration 命令：用于更新 ISE 软件的注册 ID，如果试用版用户在试用期间购买了正版软件，不用卸载再重新安装，只需要通过该命令更换 ID 即可。

(6) Tip of the Day 命令：每天提示，可设置或关闭在每次启动 ISE 时弹出对话框，列出 ISE 的最新功能和一个应用技巧。

(7) WebUpdata 命令：单击该命令，可自动连接到 Xilinx 的官方网站，下载最近的软件包并提示用户安装。

(8) About 命令：单击该命令将弹出 ISE 的版本信息，包括主版本和升级号以及注册 ID。

4.2 基于 ISE 的代码输入

4.2.1 新建工程

首先打开 ISE，每次启动时，ISE 都会默认恢复到最近使用过的工程界面。当第一次使用时，由于此时还没有过去的工程记录，所以工程管理区显示空白。选择“File | New Project”选项，在弹出的新建工程对话框的工程名称中输入“one2two”。在工程路径中单击“Browse”按钮，将工程放到指定目录，如图 4-7 所示。

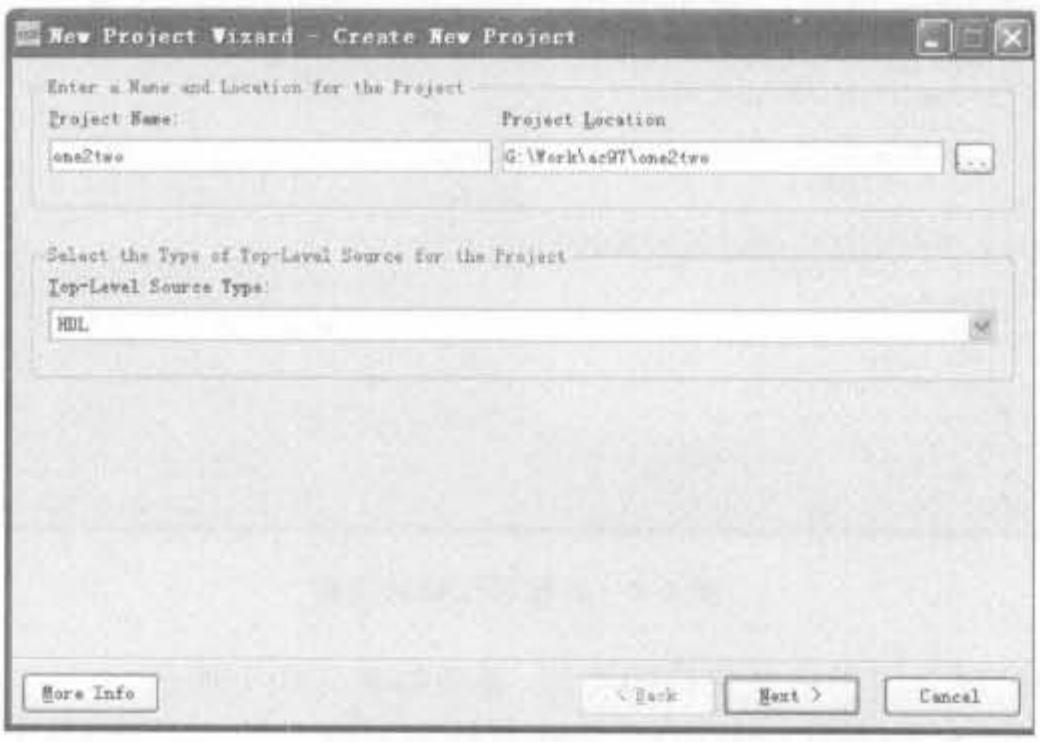


图 4-7 利用 ISE 新建工程的示意图

单击 Next 按钮进入下一页，选择所使用的芯片类型以及综合、仿真工具。计算机上所安装的所有用于仿真和综合的第三方 EDA 工具都可以在下拉菜单中找到，如图 4-8 所示。在图中，我们选用了 Virtex4-10 芯片，并且指定综合工具为 Synplify(Verilog)，仿真工具选为 Modelsim-SE Verilog。

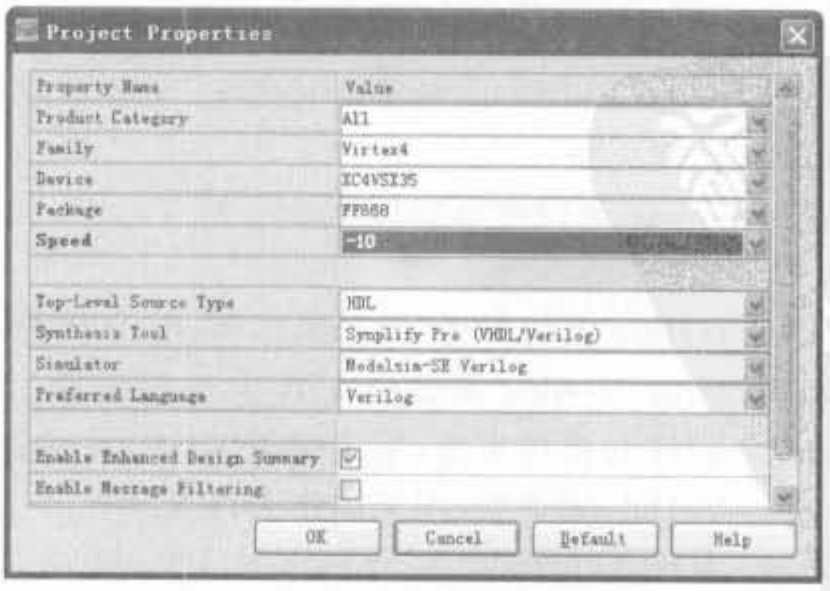


图 4-8 新建工程器件属性配置表

再单击 Next 按钮进入下一页，可以选择新建源代码文件，也可以直接跳过，进入下一页。第 4 页用于添加已有的代码，如果没有源代码，单击“Next”按钮进入最后一页，单击“OK”按钮后，就可以建立一个完整的工程。

4.2.2 代码输入

在工程管理区的任意位置单击鼠标右键，在弹出的菜单中选择“New Source”命令，会弹出如图 4-9 所示的“New Source Wizard”对话框。

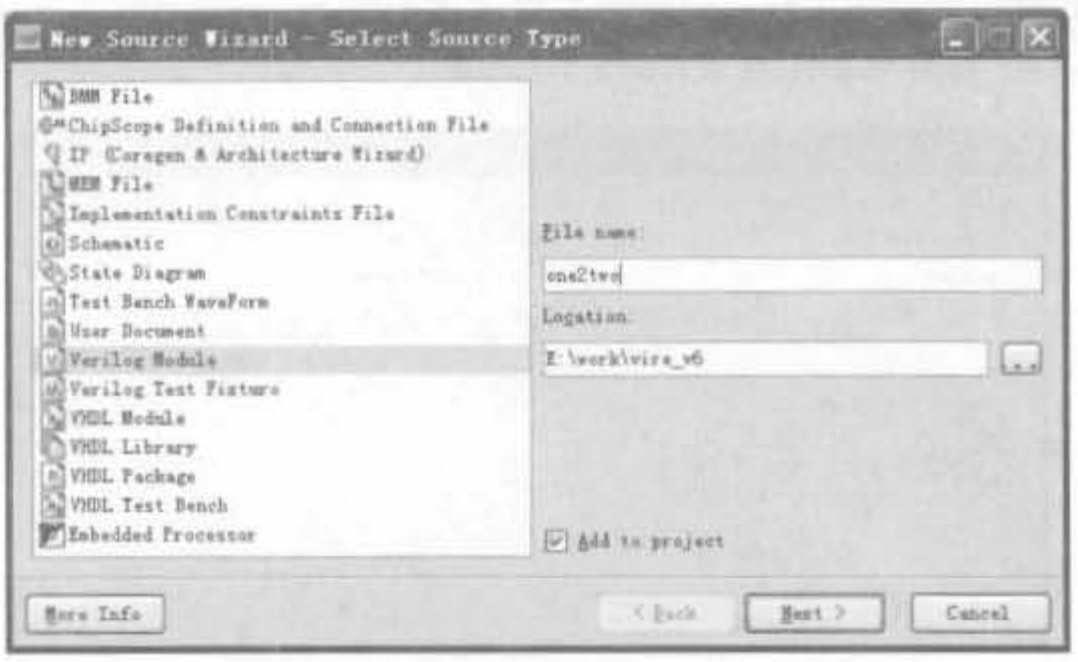


图 4-9 新建源代码对话框

对话框左侧的列表用于选择代码的类型，各项的意义如下所示。

(1) BMM(Block Memory Map)File：块存储器映射文件，用于将单个的块 RAM 连成一个更大容量的存储逻辑单元。

(2) ChipScope Definition and Connection File：在线逻辑分析仪 ChipScope 文件类型，具有独特的优势和强大的功能，将在第 6 章讨论。

(3) IP(Coregen & Architecture Wizard)：由 ISE 的 IP Core 生成工具快速生成可靠的源代码，这是目前最流行、最快速的一种设计方法，将在 4.2.3 节讨论，并贯穿于全书。

(4) MEM(Memory Define)File：存储器定义文件，用于定义 RAMB4 和 RAMB16 存储单元的内容。注意，一个工程只能包含一个 MEM 文件。

(5) Implementation Constraints File：约束文件类型，可添加时序和位置约束。

(6) State Diagram：状态图类型。

(7) Test Bench WaveForm：测试波形类型。

(8) User Document：用户文档类型。

(9) Verilog Module：Verilog 模块类型，用于编写 Verilog 代码。

(10) Verilog Test Fixture：Verilog 测试模块类型，专门编写 Verilog 测试代码。

(11) VHDL Module：VHDL 模块类型，用于编写 VHDL 代码。

(12) VHDL Library：VHDL 库类型，用于制作 VHDL 库。

(13) VHDL Package：VHDL 包类型，用于制作 VHDL 包。

(14) VHDL Test Bench：VHDL 测试模块类型，专门编写 VHDL 测试代码。

在“代码类型”中选择“Verilog Module”选项，在“File Name”文本框中输入“one2two”，单击“Next”按钮进入端口定义对话框，如图 4-10 所示。

图 4-10 Verilog 模块端口定义对话框

其中，“Module Name”就是输入的“one2two”，下面的列表框用于对端口的定义。“Port Name”表示端口名称，“Direction”表示端口方向（可以选择为 input、output 或 inout），“MSB”表示信号的最高位，“LSB”表示信号的最低位。单位信号的 MSB 和 LSB 不用填写。

定义了模块端口后，单击“Next”按钮进入下一步，单击“Finish”按钮完成创建。这样，ISE 会自动创建一个 Verilog 模块的例子，并且在源代码编辑区内打开。简单的注释、模块和端口定义已经自动生成，所剩余的工作就是在模块中实现代码。填入的代码如下：

```
module one2two(x_in,flag,y1_out,y2_out);
    input [7:0] x_in;
    input flag;
    output [7:0] y1_out;
    output [7:0] y2_out;

    // 以下为手工添加的代码
    assign y1_out = flag? x_in: 8'b0000_0000;
    assign y2_out = flag? 8'b0000_0000: x_in;

endmodule
```

4.2.3 代码模板的使用

ISE 中内嵌的语言模块包括了大量的开发实例和所有 FPGA 语法的介绍和举例，包括 Verilog HDL/HDL 的常用模块、FPGA 原语使用实例、约束文件的语法规则以及各类指令和符号的说明。语言模板不仅可在设计中直接使用，还是 FPGA 开发最好的工具手册。在 ISE 工具栏中单击图标，或选择菜单“Edit|Language Templates”，都可以打开语言模板，其界面如图 4-11 所示。

界面左边有 4 项：ABEL、UCF、Verilog 以及 VHDL，分别对应各自的参考资料。其中，ABEL 语言主要用于 GAL 和 ISP 等器件的编程，不用于 FPGA 开发。

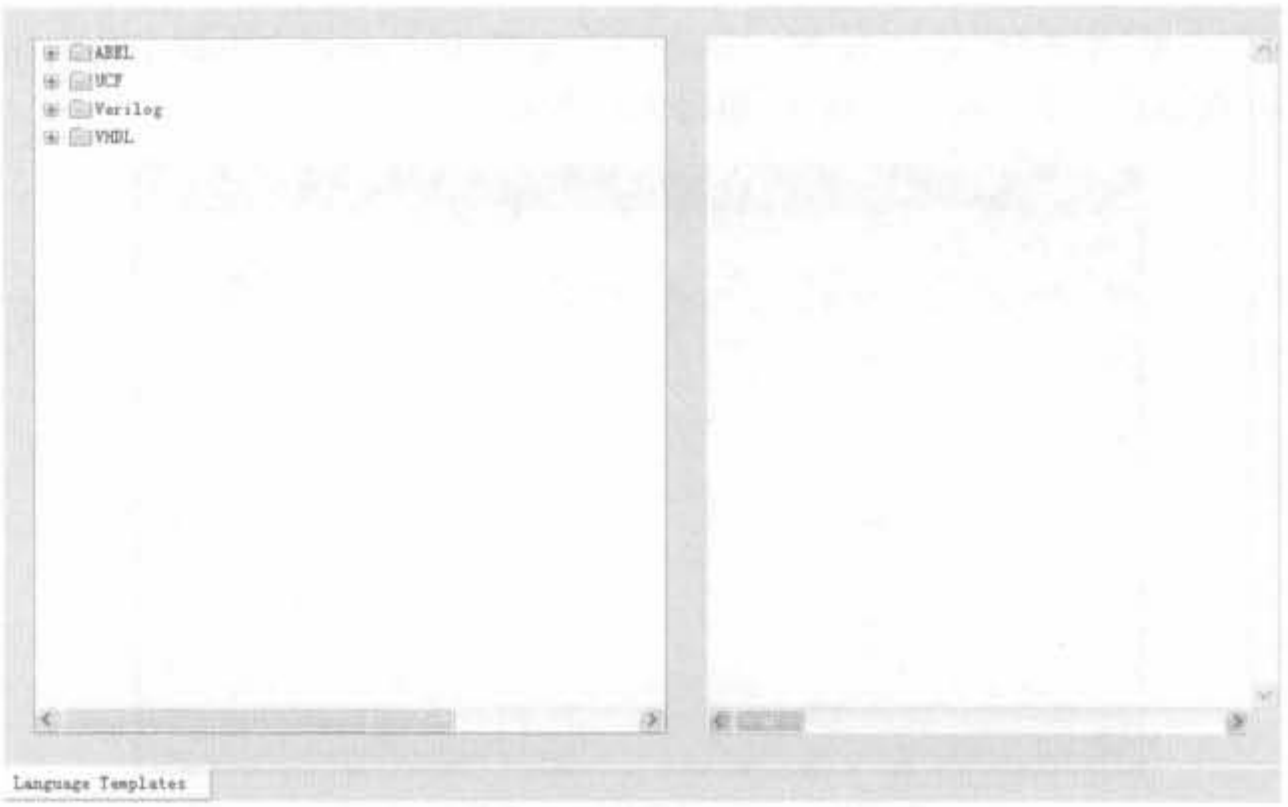


图 4-11　ISE 语言模板用户界面

以 Verilog 为例，单击其前面的“+”号，会出现 Common Constructs、Device Primitive Instantiation、Simulation Constructs、Synthesis Constructs 以及 User Templates 5 个子项。其中，第 1 项主要介绍 Verilog 开发中所用的各种符号的说明，包括注释符以及运算符等；第 2 项主要介绍 Xilinx 原语的使用，可以最大限度地利用 FPGA 的硬件资源；第 3 项给出了程序仿真的所有指令和语句的说明及示例；第 4 项给出了实际开发中可综合的 Verilog 语句，并给出了大量可靠、实用的应用实例，FPGA 开发人员应熟练掌握该部分内容。User Templates 项是设计人员自己添加的，常用于在实际开发中统一代码风格。

下面以调用全局时钟缓冲器模板为例，给出语言模板的使用方法。在语言模板中，选择“Device Primitive Instantiation”→“FPGA Clock Components”→“Clock Buffers”→“Global Clock Buffer(BUFG)”，即可看到调用全局时钟缓冲的示例代码，如图 4-12 所示。

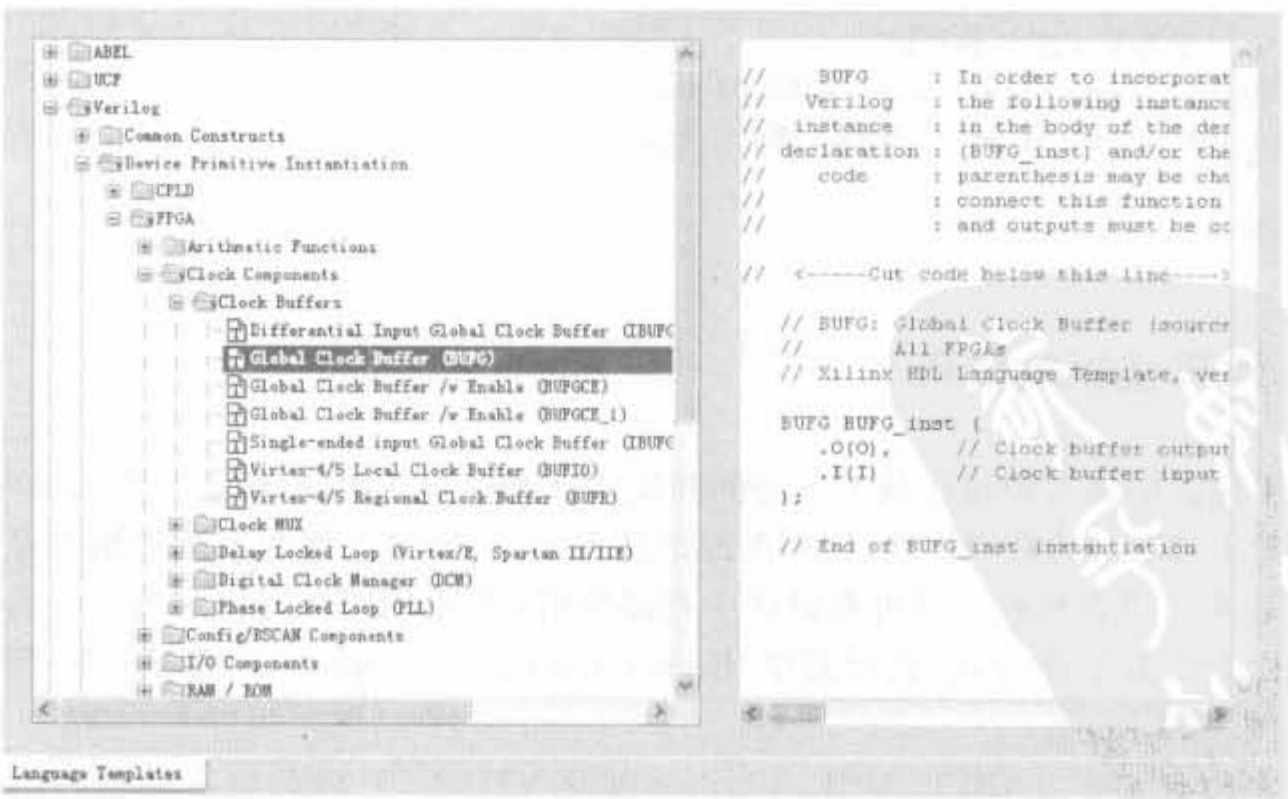


图 4-12　全局时钟缓冲器的语言模板

4.2.4 Xilinx IP Core 的使用

1. Xilinx IP Core 基本操作

IP Core 就是预先设计好、经过严格测试和优化过的电路功能模块，如乘法器、FIR 滤波器、PCI 接口等，并且一般采用参数可配置的结构，方便用户根据实际情况来调用这些模块。随着 FPGA 规模的增加，使用 IP Core 完成设计成为发展趋势。

IP Core 生成器(Core Generator)是 Xilinx FPGA 设计中的一个重要工具，提供了大量成熟的、高效的 IP Core 供用户选择，涵盖了汽车工业、基本单元、通信和网络、数字信号处理、FPGA 特点和设计、数学函数、记忆和存储单元、标准总线接口等 8 大类，从简单的基本设计模块到复杂的处理器，一应俱全。配合 Xilinx 网站的 IP 中心使用，能够大幅度减轻设计人员的工作量，提高设计可靠性。

Core Generator 最重要的配置文件的后缀是. xco，它既可以是输出文件，又可以是输入文件，包含了当前工程的属性和 IP Core 的参数信息。

启动 Core Generator 有两种方法，一种是在 ISE 中新建 IP 类型的源文件，另一种是双击运行“开始”→“程序”“Xilinx ISE 9. 1i”→“Accessories”→“Core Generator”。限于篇幅，本节只以调用加法器 IP Core 为例来介绍第一种方法。

在工程管理区单击鼠标右键，在弹出的菜单中选择“New Source”，选中 IP 类型，在“File Name”文本框中输入“adder”(注意，该名字不能出现英文的大写字母)，然后单击“Next”按钮，进入 IP Core 目录分类页面，如图 4-13 所示。

图 4-13 IP Core 目录分类页面

下面以加法器模块为例介绍详细操作。首先选中“Math Functions”→“Adders & Subtracters”→“Adder Subtracter v7. 0”，单击“Next”进入下一页，选择“Finish”完成配置。这时在信息显示区会出现“Customizing IP. . .”的提示信息，并弹出一个“Adder Subtracter”配置对话框，如图 4-14 所示。

选中“adder”，设置位宽为 16，然后单击“Generate”，信息显示区显示“Generating IP. . .”，直到出现“Successfully generated adder”的提示信息。此时在工程管理区出现一个

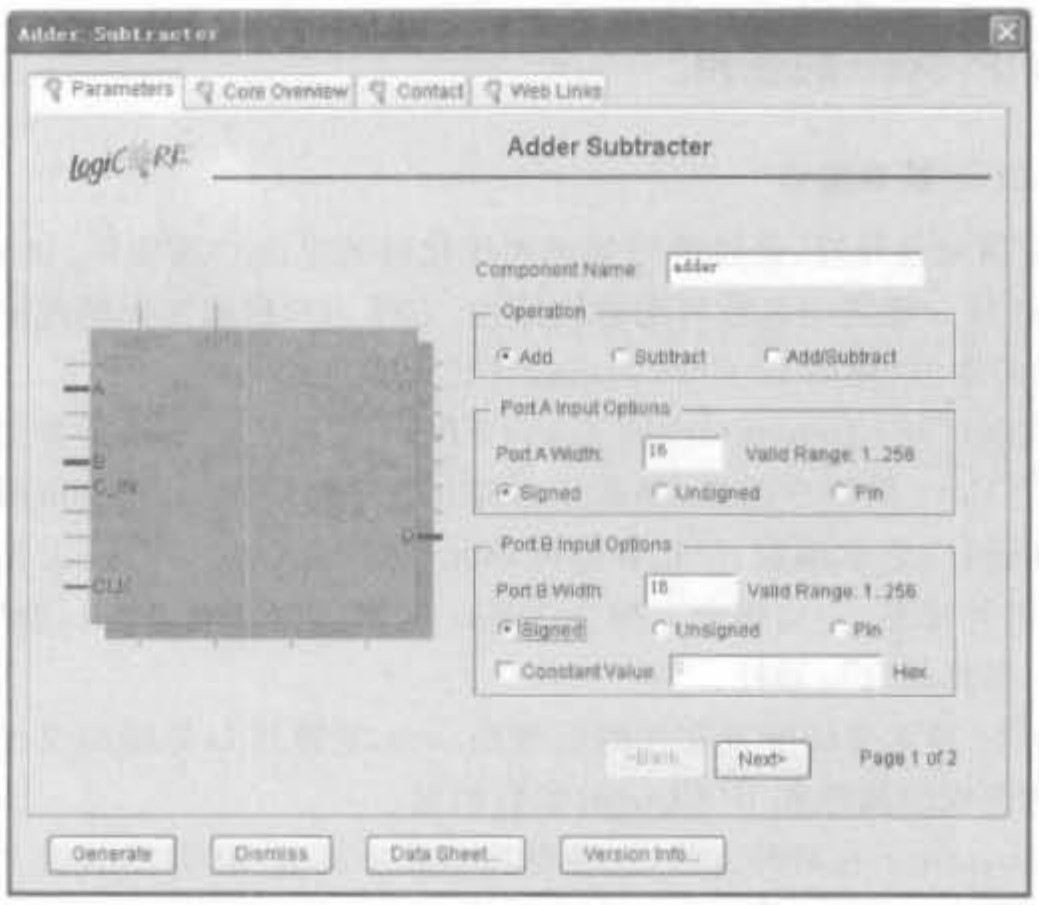


图 4-14　加法器 IP Core 配置对话框

名为 adder. xco 的文件。这样，加法器的 IP Core 已经生成并成功调用。

IP Core 在综合时被认为是黑盒子，综合器不对 IP Core 做任何编译。IP Core 的仿真主要是运用 Core Generator 的仿真模型来完成的，会自动生成扩展名为. v 的源代码文件。设计人员只需要从该源文件中查看其端口声明，将其作为一个普通的子程序进行调用即可。下面给出加法器的应用实例。

例 4-1　调用加法器的 IP Core，并用其实现图 4-15 所示的 2 级加法树。

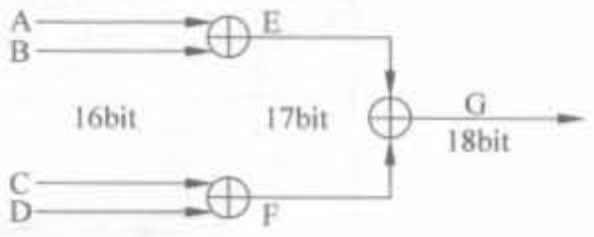


图 4-15　2 级加法器示意图

按照本节介绍的步骤生成 2 个加法器的 IP Core Add16 和 Add17，前者用于实现第 1 级加法，后者用于实现第 2 级加法，对应的代码为：

```
module addertree(clk,a1,a2,b1,b2,c);
    input            clk;
    input  [15:0]  a1;
    input  [15:0]  a2;
    input  [15:0]  b1;
    input  [15:0]  b2;
    output [17:0]  c;

    wire   [16:0]  ab1,ab2;

    adder16 adder16_1(
        .A(a1),
        .B(a2),
        .Q(ab1),
        .CLK(clk)
```

```
    );

    adder16 adder16_2(
        .A(b1),
        .B(b2),
        .Q(ab2),
        .CLK(clk)
    );

    adder17 adder17(
         .A(ab1),
        .B(ab2),
        .Q(c),
        .CLK(clk)
    );

endmodule
```

上述程序经过综合后，得到如图 4-16 所示的 RTL 级结构图。

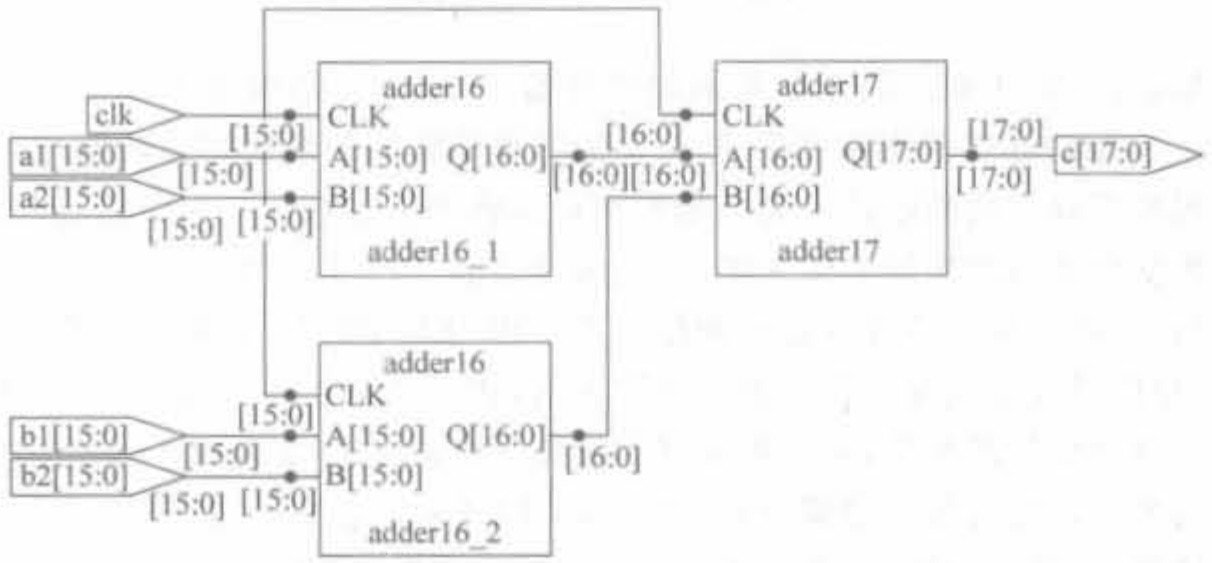


图 4-16 2 级加法树的 RTL 级结构图

经过 ModelSim 6.2b 仿真测试，得到的功能波形图如图 4-17 所示。由于每一级加法器会引入一个时钟周期的延迟，因此，两级加法器引入 2 个时钟的周期。可以看出，仿真结果和设计分析的结果是一样的。

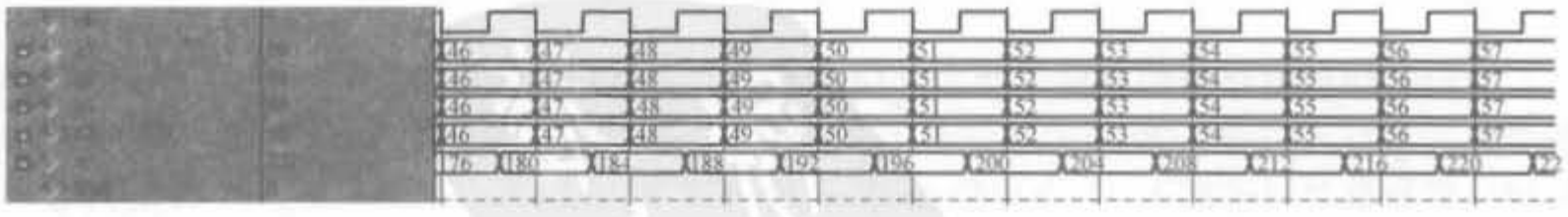


图 4-17 2 级加法树仿真结果示意图

Xilinx 公司提供了大量的、丰富的 IP Core 资源，究其本质，可以分为两类：一是面向应用的，和芯片无关；还有一种用于调用 FPGA 底层的宏单元，和芯片型号密切相关。下面针对这两类，分别给出数字频率合成器模块 DDS 的调用实例。

2. DDS 模块 IP Core 的调用实例

1) DDS 算法原理

DDS 技术是一种新的频率合成方法，是频率合成技术的一次革命，最早由 Joseph Tierney 等 3 人于 1971 年提出，但由于受当时微电子技术和数字信号处理技术的限制，DDS 技术没有受到足够重视。随着数字集成电路和微电子技术的发展，DDS 技术日益显露出它的优越性。

DDS 的工作原理为：在参考时钟的驱动下，相位累加器对频率控制字进行线性累加，得到的相位码对波形存储器寻址，使之输出相应的幅度码，经过模/数转换器得到相应的阶梯波，最后使用低通滤波器对其进行平滑，得到所需频率的平滑、连续的波形，其结构如图 4-18 所示。

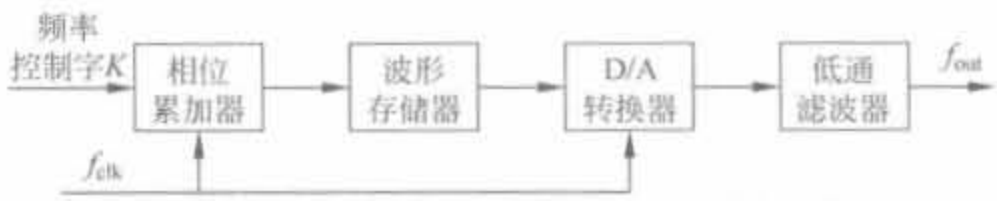


图 4-18　DDS 的结构框图

相位累加器由 N 位加法器与 N 位累加寄存器级联构成，结构如图 4-19 所示。每来一个时钟脉冲 f_{clk}，加法器将频率控制字 K 与累加寄存器输出的累加相位数据相加，把相加后的结果送至累加寄存器的数据输入端。累加寄存器将加法器在上一个时钟脉冲作用后所产生的新相位数据反馈到加法器的输入端，以使加法器在下一个时钟脉冲的作用下继续与频率控制字相加。这样，相位累加器在时钟作用下不断对频率控制字进行线性相位累加。由此可以看出，相位累加器在每一个时钟脉冲输入时，把频率控制字累加一次，相位累加器输出的数据就是合成信号的相位，相位累加器的溢出频率就是 DDS 输出的信号频率。用相位累加器输出的数据作为波形存储器(ROM)的相位取样地址，这样就可把存储在波形存储器内的波形抽样值(二进制编码)经查找表查出，完成相位到幅值的转换。

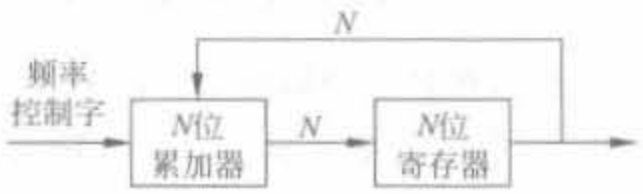


图 4-19　DDS 相位累加器

波形存储器所存储的幅度值与余弦信号有关。余弦信号波形在一个周期内的相位幅度的变化关系可以用图 4-20 中的相位圆表示，每一个点对应一个特定的幅度值。一个 N 位的相位累加器对应着圆上 2^N 个相位点，其相位分辨率为 $\Delta\phi=2\pi/2^N$。若 $N=4$，则共有 16 种相位值与 16 种幅度值相对应，并将相应的幅度值存储于波形存储器中。存储器的字节数决定了相位量化误差。在实际的 DDS 中，可利用正弦波的对称性，将 2π 范围内的幅、相点减小到 $\pi/2$ 内，以降低所需的存储量。量化的比特数决定了幅度量化误差。

波形存储器的输出送到 D/A 转换器，D/A 转换器将数字量形式的波形幅值转换成所要求合成频率的模拟量形式信号。低通滤波器用于滤除不需要的取样分量，以便输出频谱纯净的正弦波信号。DDS 在相对带宽、频率转换时间、高分辨率、相位连续性、正交输出以

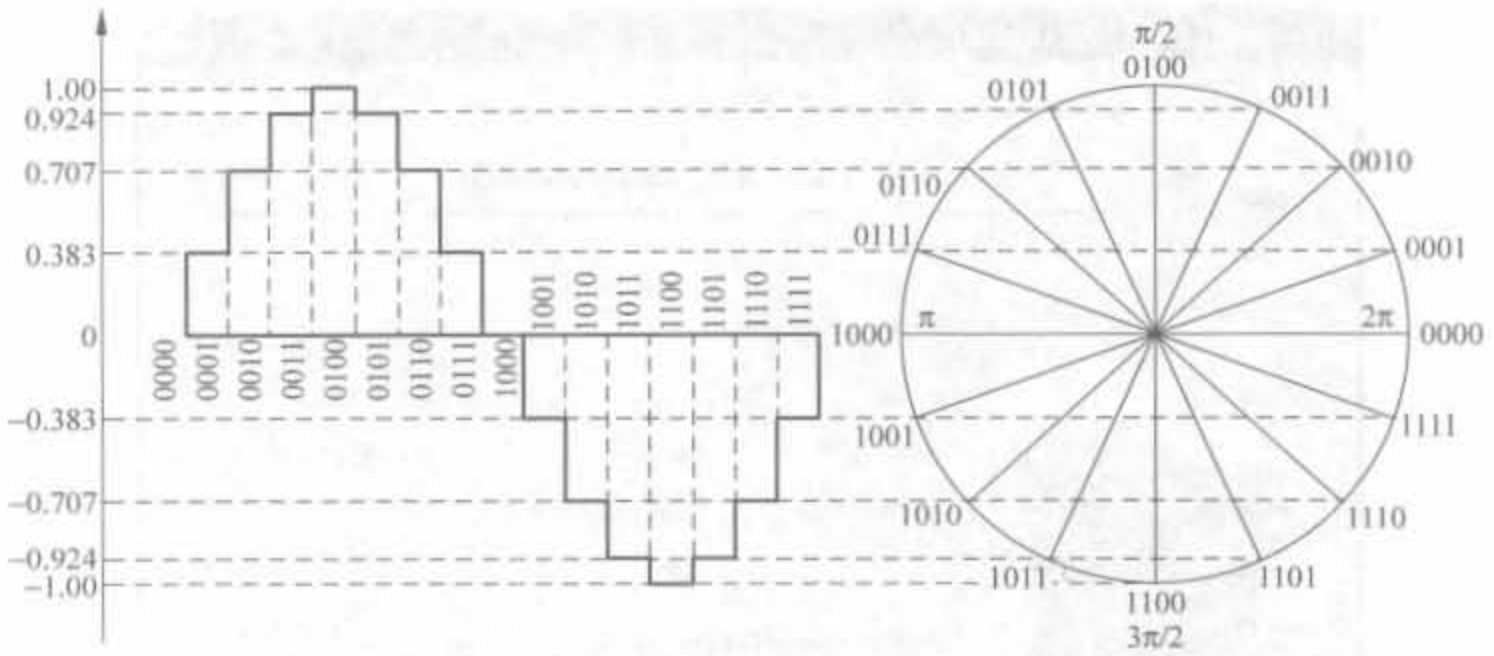


图 4-20 三角函数相位与幅度的对应关系

及集成化等一系列性能指标方面远远超过了传统频率合成技术所能达到的水平，为系统提供的信号源优于模拟信号源。

DDS 模块的输出频率 f_{out} 是系统工作频率 f_{clk}、相位累加器比特数 N 以及频率控制字 K 三者的一个函数，其数学关系由式(4-1)给出。

$$f_{out} = \frac{f_{clk}}{2^N} \tag{4-1}$$

它的频率分辨率，即频率的变化间隔为

$$\Delta f = \frac{f_{clk}}{2^N} \tag{4-2}$$

2) DDS IP Core 的调用

DDS 模块 IP Core 的用户界面如图 4-21 所示。该 IP Core 支持余弦、正弦以及正交函数的输出，旁瓣抑制比的范围为 18dB～115dB，最小频率分辨率为 0.02Hz，可同时独立支持 16 个通道。其中的查找表既可以利用分布式 RAM，也可利用块 RAM。

DDS 模块的信号端口说明如下：

(1) CLK：输入信号，DDS 模块的工作时钟，对 DDS 输出信号的频率和频率分辨率有很大的影响，即式(4-1)中的 f_{clk}。

(2) A：输入信号，由于 DDS 模块的相位增量存储器和相位偏置寄存器共用一个数据通道，A 端口信号用于片选相位增量寄存器和偏置寄存器。当 A 端口的最高位为 1 时，相位偏置寄存器被选中；当其为 0 时，则选中相位增量寄存器。其余的低 4 位用于片选 DDS 的输出通道，最多可以输出 16 路信号。

(3) WE：输入信号，写有效控制信号，高电平有效。只有当 WE 为高电平时，DATA 端口的数值才能被写入相应的寄存器。

(4) DATA：输入信号，时分复用的数据总线，用于配置相位增量寄存器和相位偏置寄存器。

(5) ACLR：输入信号，异步的清空信号，高电平有效。当 ACLR 等于 1 时，DDS 模块

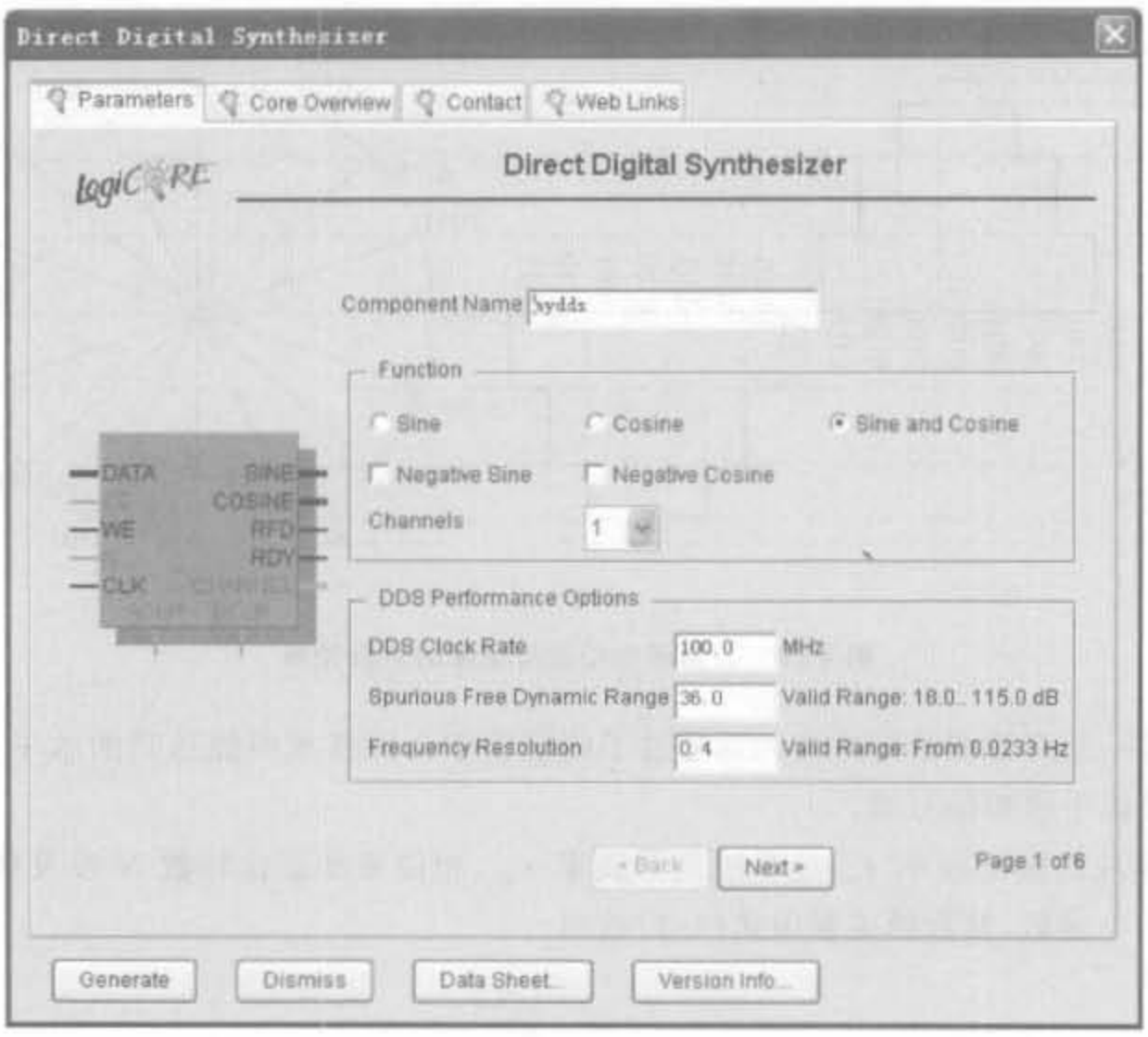


图 4-21　DDS IP Core 的用户界面

内部所有的寄存器都被清空，RDY 信号也会被拉低。

(6) SCLR：输入信号，同步的清空信号，高电平有效。当 SCLR 等于 1 时，DDS 模块内部所有的寄存器都被清空，RDY 信号也会被拉低。

(7) RDY：输出信号，输出握手信号。当其为高电平时，标志输出信号已经准备好。

(8) CHANNEL：输出信号，输出通路的下标。用于表明当前时刻输出端为哪一路输出，其位宽由通道数决定。

(9) SINE：输出信号，用于输出正弦的时间序列。

(10) COSINE：输出信号，用于输出余弦的时间序列。

例 4-2　使用 DDS IP Core 实例化一个 4MHz，分辨率为 0.1Hz，带外抑制比为 60dB 的正、余弦信号发生器，假设工作时钟为 100MHz。

IP Core 直接生成 DDS 的 Verilog 模块接口为：

```
module mydds(
    DATA,
    WE,
    A,
    CLK,
    SINE,
    COSINE
    ); // synthesis black_box
```

```
    input [27:0] DATA;
    input WE;
    input [4:0] A;
    input CLK;
    output [9:0] SINE;
    output [9:0] COSINE;
......
endmodule
```

在使用时，直接调用 mydds 模块即可，如

```
module dds1(DATA,WE,A,CLK,SINE,COSINE);
    input [27:0] DATA; //经过计算,DATA = 10737418.
    input WE;
    input [4:0] A;
    input CLK;
    output [9:0] SINE;
    output [9:0] COSINE;

    mydds mydds1(
    .DATA(DATA),
    .WE(WE),
    .A(A),
    .CLK(CLK),
    .SINE(SINE),
    .COSINE(COSINE)
    );

endmodule
```

上述程序经过综合后，得到如图 4-22 所示的 RTL 结构图。

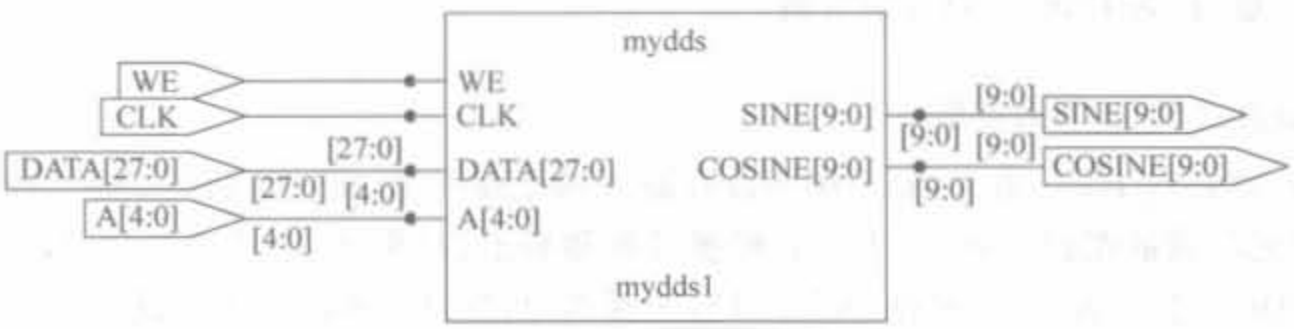


图 4-22 DDS 模块的 RTL 结构图

经过 ModelSim 仿真测试，得到的功能波形图如图 4-23 所示。

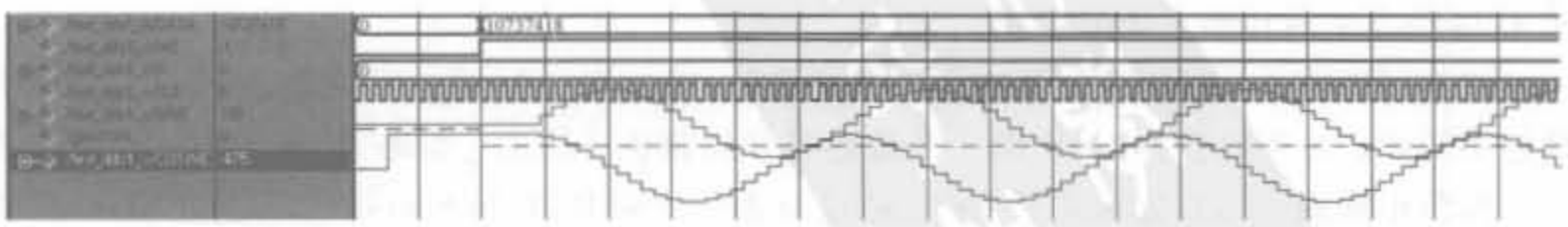


图 4-23 DDS 模块的局部功能仿真波形图

4.3 基于 ISE 的开发流程

Xilinx 公司的 ISE 软件是一套用以开发 Xilinx 公司的 FPGA&CPLD 的集成开发软件，它提供给用户一个从设计输入到综合、布线、仿真、下载的全套解决方案，并可以很方便地与其他 EDA 工具接口。其中，原理图输入用的是第三方软件 ECS，HDL 综合可以使用 Xilinx 公司开发的 XST（Xilinx Synthesis Technology）、Synopsys 的 FPGA Express 和 Synplicity 公司的 Synplify/Synplify Pro，测试台输入是图形化的 HDL Bencher，状态图输入用的是 StateCAD，前、后仿真则可以使用 Modelsim XE（Xilinx Edition）或 Modelsim SE 等。本节主要介绍基于 ISE 的 FPGA 开发流程，第三方 EDA 工具的使用方法在 4.5 节介绍。本节所有的讲解都以例 4-3 所示的代码为基础展开。

例 4-3 ISE 开发流程演示代码，将输入的数据加 1 寄存并输出。

```
module test(clk,din,dout);
    input clk;
    input  [7:0] din;
    output [7:0] dout;

    reg [7:0] dout;

    always @(posedge clk) begin
        dout<=din+1;
    end

endmodule
```

4.3.1 基于 Xilinx XST 的综合

1. 综合工具 XST 的使用

综合，就是将 HDL 语言、原理图等设计输入翻译成由与、或、非门和 RAM、触发器等基本逻辑单元组成的逻辑连接（网表），并根据目标和要求（约束条件）优化所生成的逻辑连接而生成 EDF 文件。XST 内嵌在 ISE 3 以后的版本中，并且在不断完善。此外，由于 XST 是 Xilinx 公司自己的综合工具，对于部分 Xilinx 芯片独有的结构具有更好的融合性。

完成了输入、仿真以及管脚分配后，就可以进行综合和实现了。在过程管理区双击 Synthesize-XST，如图 4-24 所示，就可以完成综合，并且能够给出初步的资源消耗情况。图 4-25 给出了模块所占用的资源。

综合可能有 3 种结果：如果综合后完全正确，则在 Synthesize-XST 前面有一个打钩的绿色小圆圈；如果有警告，则出现一个带感叹号的黄色小圆圈；如果有错误，则出现一个带叉的红色小圆圈。综合完成之后，可以通过双击“View RTL Schematic”来查看 RTL 级结构图，查看综合结构是否按照设计意图来实现电路。ISE 会自动调用原理图编辑器 ECS 来浏览 RTL 结构。对于例 4-3，所得到的 RTL 结构图如图 4-26 所示，综合结果符合设计者的意图，调用了加法器和寄存器来完成逻辑。

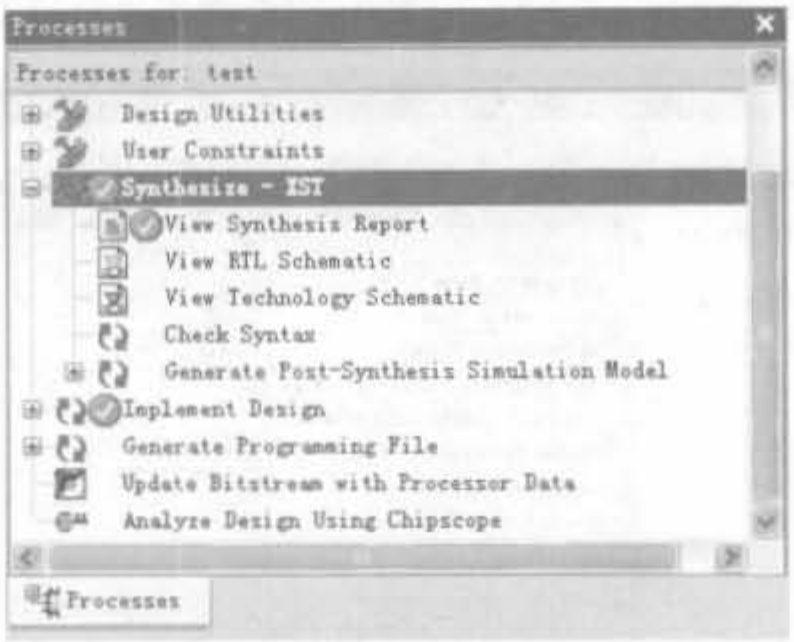


图 4-24 设计综合窗口

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	8	30,720	1%	
Logic Distribution				
Number of occupied Slices	5	15,360	1%	
Number of Slices containing only related logic	5	5	100%	
Number of Slices containing unrelated logic	0	5	0%	
Total Number of 4 input LUTs	9	30,720	1%	
Number used as logic	8			
Number used as a route-thru	1			
Number of bonded IOBs	17	448	3%	
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Number used as BUFGCTRLs	0			
Total equivalent gate count for design	115			
Additional JTAG gate count for IOBs	816			

图 4-25 综合结果报告

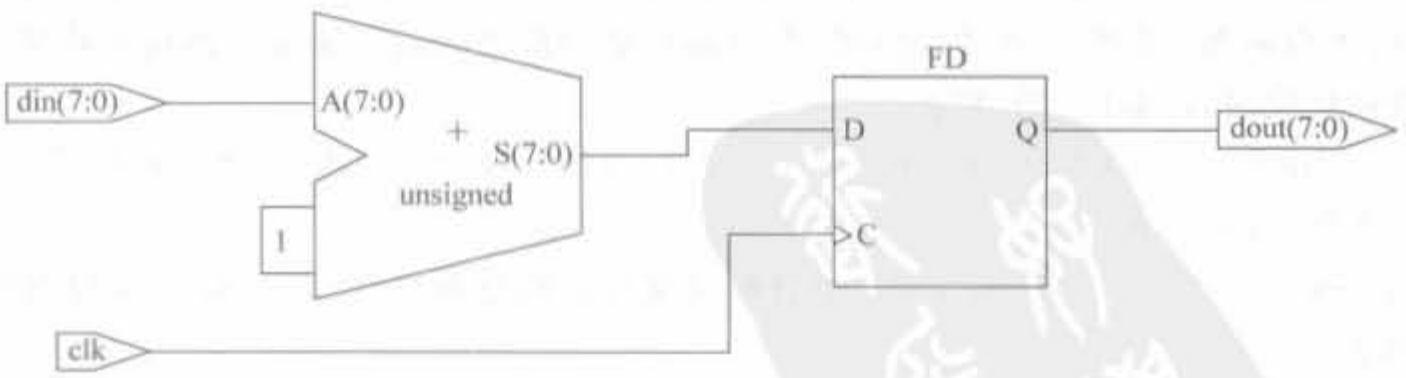


图 4-26 经过综合后的 RTL 结构图

2. 综合参数的设置

一般在使用 XST 时，所有的属性都采用默认值。其实，XST 对不同的逻辑设计可提供丰富、灵活的属性配置。下面对 ISE 9.1 中内嵌的 XST 属性进行说明。打开 ISE 中的设计

工程，在过程管理区选中“Synthesis-XST”并单击右键，弹出界面如图 4-27 所示。

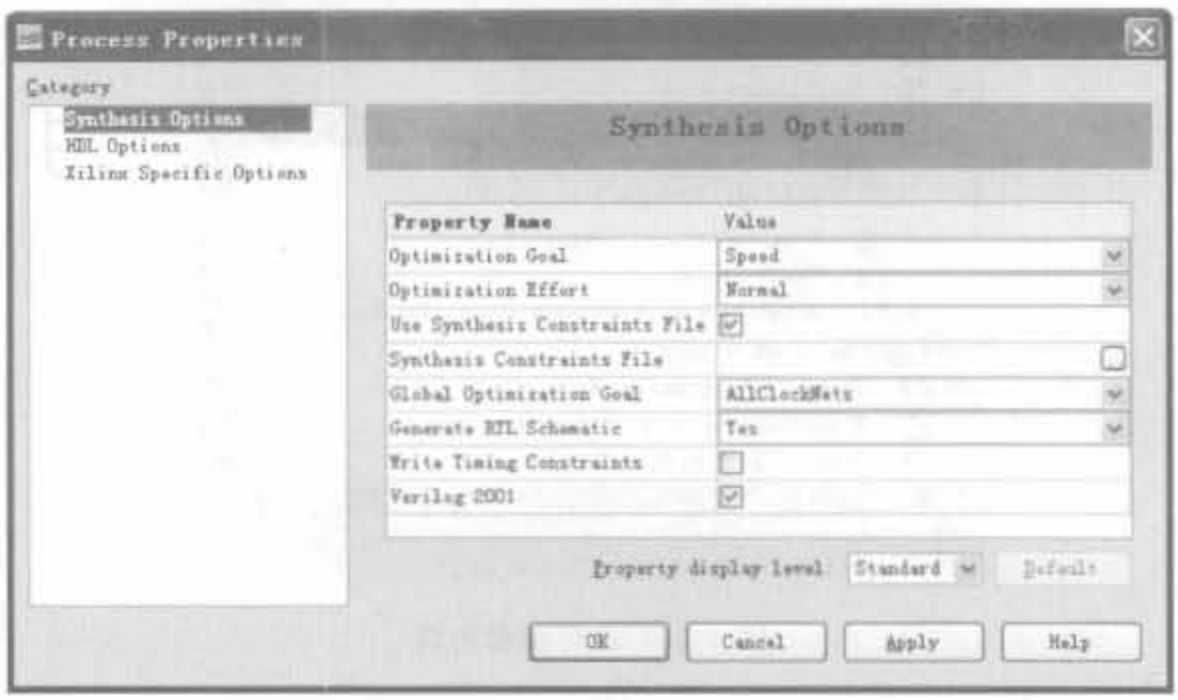


图 4-27　综合选项

由图 4-27 可以看出，XST 配置页面分为综合选项（Synthesis Options）、HDL 语言选项（HDL Options）以及 Xilinx 特殊选项（Xilinx Specific Options）等三大类，分别用于设置综合的全局目标和整体策略、HDL 硬件语法规则以及 Xilinx 特有的结构属性。

1）综合选项参数

综合参数配置界面如图 4-27 所示，包括 8 个选项，具体如下所述：

（1）Optimization Goal：优化的目标。该参数决定了综合工具对设计进行优化时，是以面积还是以速度作为优化原则。面积优化原则可以节省器件内部的逻辑资源，即尽可能地采用串行逻辑结构，但这是以牺牲速度为代价的。而速度优化原则保证了器件的整体工作速度，即尽可能地采用并行逻辑结构，但这样将会浪费器件内部大量的逻辑资源，因此，它是以牺牲逻辑资源为代价的。

（2）Optimization Effort：优化器努力程度。这里有 normal 和 high 两种选择方式。对于 normal，优化器对逻辑设计仅仅进行普通的优化处理，其结果可能并不是最好的，但是综合和优化流程执行得较快。如果选择 high，优化器对逻辑设计进行反复的优化处理和分析，并能生成最理想的综合和优化结果，在对高性能和最终的设计通常采用这种模式；当然在综合和优化时，需要的时间较长。

（3）Use Synthesis Constraints File：使用综合约束文件。如果选择了该选项，那么综合约束文件 XCF 有效。

（4）Synthesis Constraints File：综合约束文件。该选项用于指定 XST 综合约束文件 XCF 的路径。

（5）Global Optimization Goal：全局优化目标。可以选择的属性包括 AllClockNets、Inpad To Outpad、Offest In Before、Offest Out After 和 Maximm Delay。该参数仅对 FPGA 器件有效，可用于选择所设定的寄存器之间、输入管脚到寄存器之间、寄存器到输出管脚之间，或者是输入管脚到输出管脚之间逻辑的优化策略。

（6）Generate RTL Schematic：生成寄存器传输级视图文件。该参数用于将综合结果

生成 RTL 视图。

(7) Write Timing Constraints：写时序约束。该参数仅对 FPGA 有效，用来设置是否将 HDL 源代码中用于控制综合的时序约束传给 NGC 网表文件。该文件用于布局和布线。

(8) Verilog 2001：选择是否支持 Verilog 2001 版本。

2) HDL 语言选项

HDL 语言选项的配置界面如图 4-28 所示，包括 16 个选项，具体如下所述：

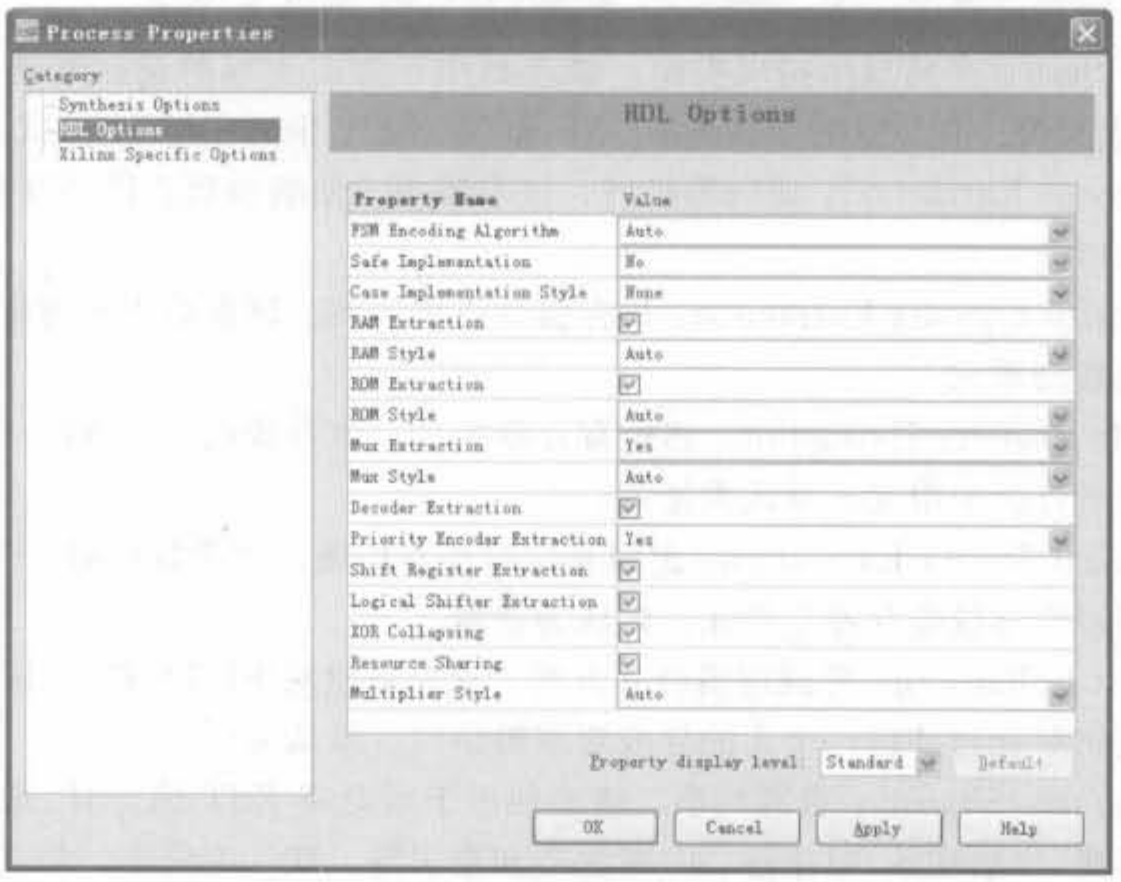


图 4-28 HDL 语言选项的配置界面选项

(1) FSM Encoding Algorithm：有限状态机编码算法。该参数用于指定有限状态机的编码方式。选项有 Auto、One-Hot、Compact、Sequential、Gray、Johnson、User、Speed1、None 编码方式，默认为 Auto 编码方式。

(2) Safe Implementation：将添加安全模式约束来实现有限状态机，将添加额外的逻辑将状态机从无效状态调转到有效状态，否则只能复位来实现。它有 Yes、No 两种选择，默认为 No。

(3) Case Implementation Style：条件语句实现类型。该参数用于控制 XST 综合工具解释和推论 Verilog 的条件语句，其选项有 None、Full、Parallel、Full-Parallel，默认为 None。对于这 4 种选项，区别如下：①None，XST 将保留程序中条件语句的原型，不进行任何处理；②Full，XST 认为条件语句是完整的，避免锁存器的产生；③Parallel，XST 认为在条件语句中不能产生分支，并且不使用优先级编码器；④Full-Parallel，XST 认为条件语句是完整的，并且在内部没有分支，不使用锁存器和优先级编码器。

(4) RAM Extraction：存储器扩展。该参数仅对 FPGA 有效，用于使能和禁止 RAM 宏接口。默认为允许使用 RAM 宏接口。

(5) RAM Style：RAM 实现类型。该参数仅对 FPGA 有效，用于选择是采用块 RAM 还是分布式 RAM 来作为 RAM 的实现类型。默认为 Auto。

(6) ROM Extraction：只读存储器扩展。该参数仅对 FPGA 有效，用于使能和禁止只读存储器 ROM 宏接口。默认为允许使用 ROM 宏接口。

(7) ROM Style：ROM 实现类型。该参数仅对 FPGA 有效，用于选择是采用块 RAM 还是分布式 RAM 来作为 ROM 的实现和推论类型。默认为 Auto。

(8) Mux Extraction：多路复用器扩展。该参数用于使能和禁止多路复用器的宏接口。根据某些内定的算法，对于每个已识别的多路复用/选择器，XST 能够创建一个宏，并进行逻辑的优化。可以选择 Yes、No 和 Force 中的任何一种，默认为 Yes。

(9) Mux Style：多路复用实现类型。该参数用于为宏生成器选择实现和推论多路复用/选择器的宏类型。可以选择 Auto、MUXF 和 MUXCY 中的任何一种，默认为 Auto。

(10) Decoder Extraction：译码器扩展。该参数用于使能和禁止译码器宏接口，默认为允许使用该接口。

(11) Priority Encoder Extraction：优先级译码器扩展。该参数用于指定是否使用带有优先级的编码器宏单元。

(12) Shift Register Extraction：移位寄存器扩展。该参数仅对 FPGA 有效，用于指定是否使用移位寄存器宏单元。默认为使能。

(13) Logical Shifter Extraction：逻辑移位寄存器扩展。该参数仅对 FPGA 有效，用于指定是否使用逻辑移位寄存器宏单元。默认为使能。

(14) XOR Collapsing：异或逻辑合并方式。该参数仅对 FPGA 有效，用于指定是否将级联的异或逻辑单元合并成一个大的异或宏逻辑结构。默认为使能。

(15) Resource Sharing：资源共享。该参数用于指定在 XST 综合时，是否允许复用一些运算处理模块，如加法器、减法器、加/减法器和乘法器。默认为使能。如果综合工具的选择是以速度为优先原则的，那么就不考虑资源共享。

(16) Multiplier Style：乘法器实现类型。该参数仅对 FPGA 有效，用于指定宏生成器使用乘法器宏单元的方式。选项有 Auto、Block、LUT 和 Pipe_LUT。默认为 Auto。选择的乘法器实现类型和所选择的器件有关。

3) Xilinx 特殊选项

Xilinx 特殊选项用于将用户逻辑适配到 Xilinx 芯片的特殊结构中，这不仅能节省资源，还能提高设计的工作频率，其配置界面如图 4-29 所示，包括 10 个配置选项，具体如下所述。

(1) Add I/O Buffers：插入 I/O 缓冲器。该参数用于控制对所综合的模块是否自动插入 I/O 缓冲器。默认为自动插入。

(2) Max Fanout：最大扇出数。该参数用于指定信号和网线的最大扇出数。这里，扇出数的选择与设计的性能有直接的关系，需要用户合理选择。

(3) Register Duplication：寄存器复制。该参数用于控制是否允许寄存器的复制。对于高扇出和时序不能满足要求的寄存器进行复制，可以减少缓冲器输出的数目以及逻辑级数，改变时序的某些特性，提高设计的工作频率。默认为允许寄存器复制。

(4) Equivalent Register Removal：等效寄存器删除。该参数用于指定是否把传输级功能等效的寄存器删除，这样可以减少寄存器资源的使用。如果某个寄存器是用 Xilinx 的硬件原语指定的，那么就不会被删除。默认为使能。

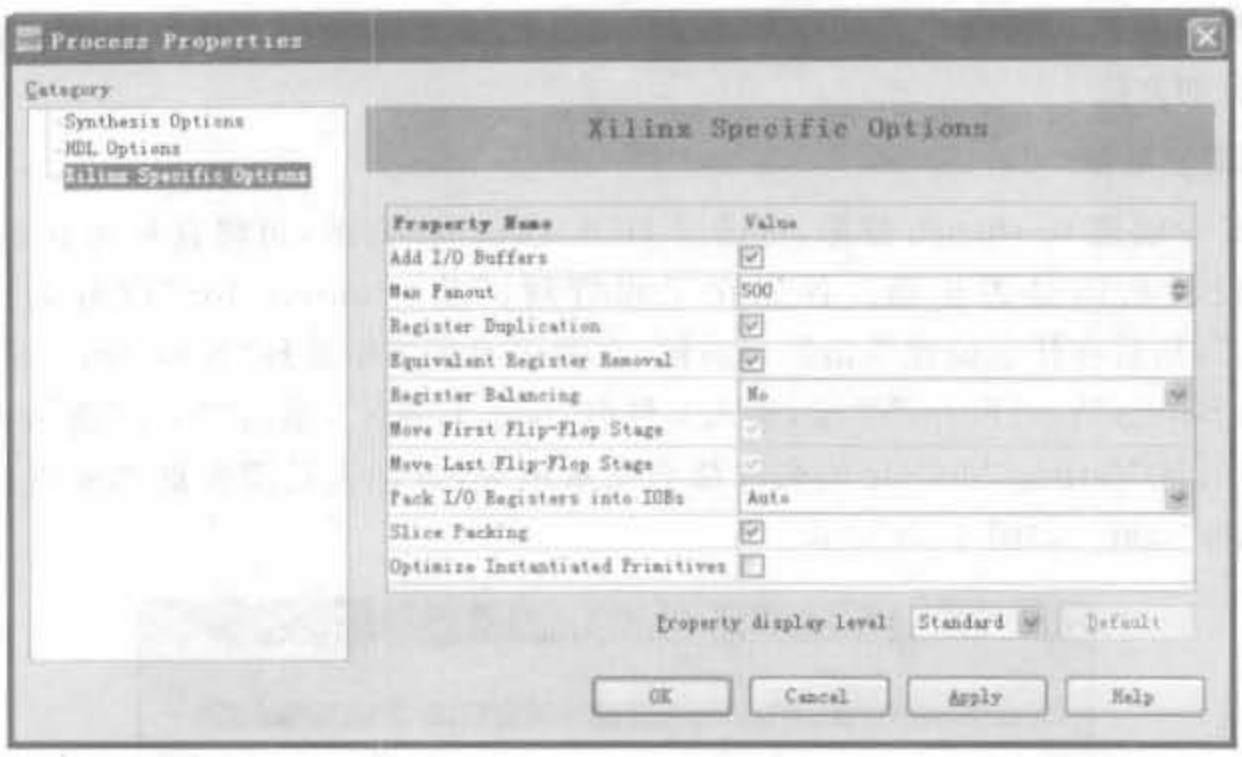


图 4-29 Xilinx 指定的选项

(5) Register Balancing：寄存器配平。该参数仅对 FPGA 有效，用于指定是否允许平衡寄存器。可选项有 No、Yes、Forward 和 Backward。采用寄存器配平技术，可以改善某些设计的时序条件。其中，Forward 为前移寄存器配平，Backward 为后移寄存器配平。采用寄存器配平后，所用到的寄存器数会相应地增、减。默认为寄存器不配平。

(6) Move First Flip-Flop Stage：移动前级寄存器。该参数仅对 FPGA 有效，用于控制在进行寄存器配平时，是否允许移动前级寄存器。如果 Register Balancing 的设置为 No，那么该参数的设置无效。

(7) Move Last Flip-Flop Stage：移动后级寄存器。该参数仅对 FPGA 有效，用于控制在进行寄存器配平时，是否允许移动后级寄存器。如果 Register Balancing 的设置为 No，那么该参数的设置无效。

(8) Pack I/O Registers into IOBs：I/O 寄存器置于输入/输出块。该参数仅对 FPGA 有效，用于控制是否将逻辑设计中的寄存器用 IOB 内部寄存器实现。在 Xilinx 系列 FPGA 的 IOB 中分别有输入和输出寄存器。如果将设计中的第一级寄存器或最后一级寄存器用 IOB 内部寄存器实现，可以缩短 I/O 管脚到寄存器之间的路径，通常可以缩短大约 1～2ns 传输延时。默认为 Auto。

(9) Slice Packing：优化 Slice 结构。该参数仅对 FPGA 有效，用于控制是否将关键路径的查找表逻辑尽量配置在同一个 Slice 或者 CLB 模块中，由此来缩短 LUT 之间的布线。这一功能对于提高设计的工作频率、改善时序特性是非常有用的。默认为允许优化 Slice 结构。

(10) Optimize Instantiated Primitives：优化已例化的原语。该参数控制是否需要优化在 HDL 代码中已例化的原语。默认为不优化。

4.3.2 基于 ISE 的仿真

在代码编写完毕后，需要借助于测试平台来验证所设计的模块是否满足要求。ISE 提供了两种测试平台的建立方法，一种是使用 HDL Bencher 的图形化波形编辑功能编写，另

一种就是利用 HDL 语言，相对于前者使用简单、功能强大。下面简要介绍利用上述平台建立测试平台的方法。

1. 测试波形法

在 ISE 中创建 testbench 波形，可通过 HDL Bencher 修改，再将其和仿真器连接起来，然后验证设计功能是否正确。首先在工程管理区将“Sources for”设置为“Behavioral Simulation”，然后在任意位置单击鼠标右键，在弹出的菜单中选择“New Source”命令，然后选中“Test Bench WaveForm”类型，输入文件名“test_bench”，单击“Next”进入下一页。这时，工程中所有 Verilog Module 的名称都会显示出来，设计人员需要选择要进行测试的模块，这里选择“test”，如图 4-30 所示。

图 4-30　选择待测模块对话框

用鼠标选中“test”，单击“Next”进入下一页，直接单击“Finish”按钮。此时 HDL Bencher 程序自动启动，等待用户输入所需的时序要求，如图 4-31 所示。

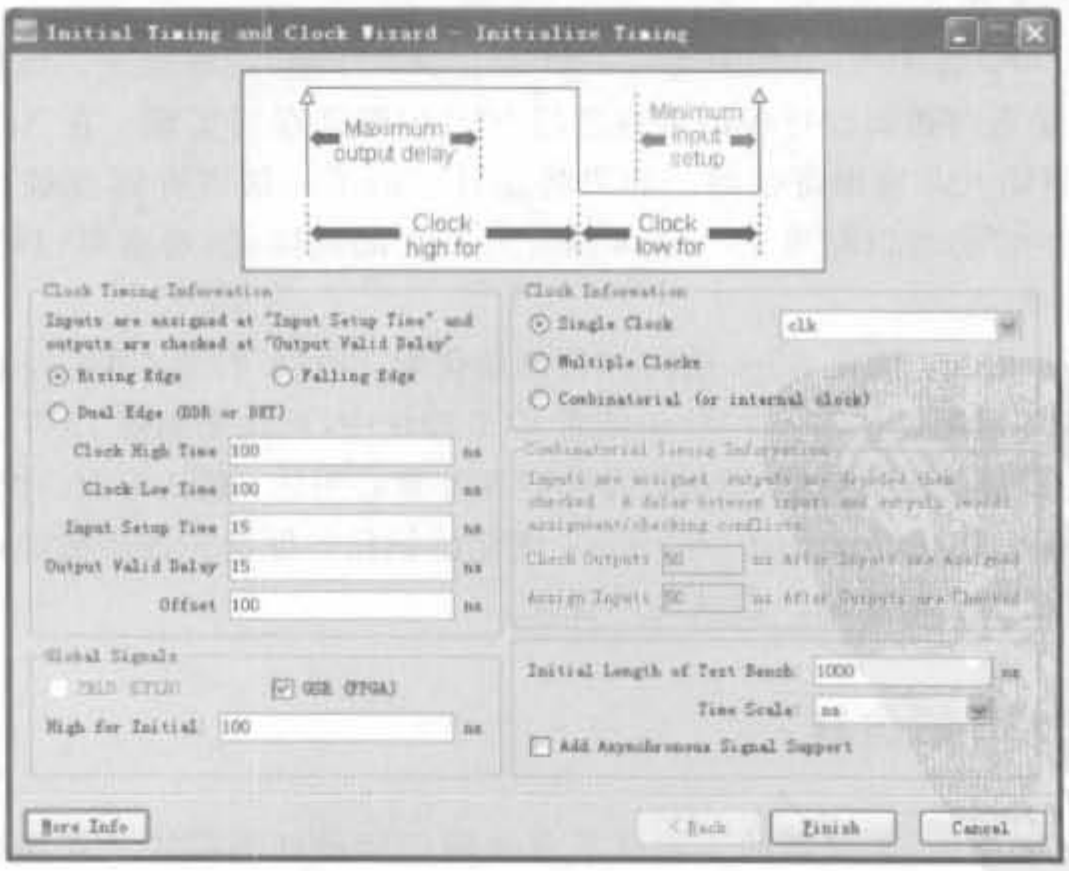


图 4-31　时序初始化窗口

时钟高电平时间和时钟低电平时间一起定义了设计操作必须达到的时钟周期，输入建立时间定义了输入在什么时候必须有效；输出有效延时定义了有效时钟延时到达后多久必须输出有效数据。默认的初始化时间设置如下：

- 时钟高电平时间(Clock High Time)：100ns；
- 时钟低电平时间(Clock Low Time)：100ns；
- 输入建立时间(Input Setup)：15ns；
- 输出有效时间(Output Valid)：15ns；
- 偏移时间(Offset)：100ns。

单击“OK”按钮，接受默认的时间设定。测试矢量波形显示如图 4-32 所示。

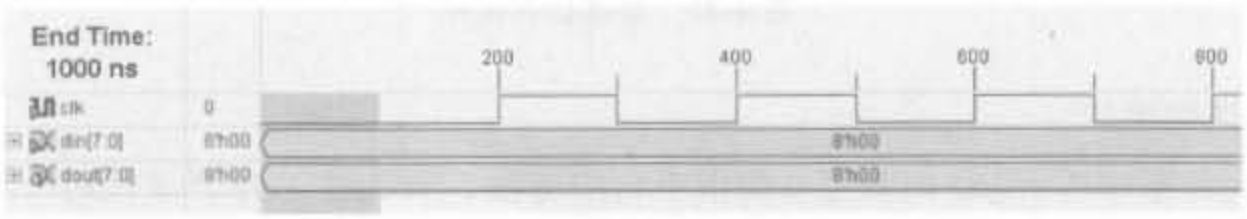


图 4-32 测试矢量波形

接下来，初始化输入(注意，灰色的部分不允许用户修改)，修改的方法为：选中信号，在其波形上单击。从该点所在周期开始，在往后所有的时间单元内，该信号电平反相。单击 din 信号前面的“+”号，在 din[7]的第 2 个时钟周期内单击，使其变高；在 din[6]的第 3 个时钟周期内单击，使其变高；用同样的方法修改 din[5]～din[0]信号，使其如图 4-33 所示。

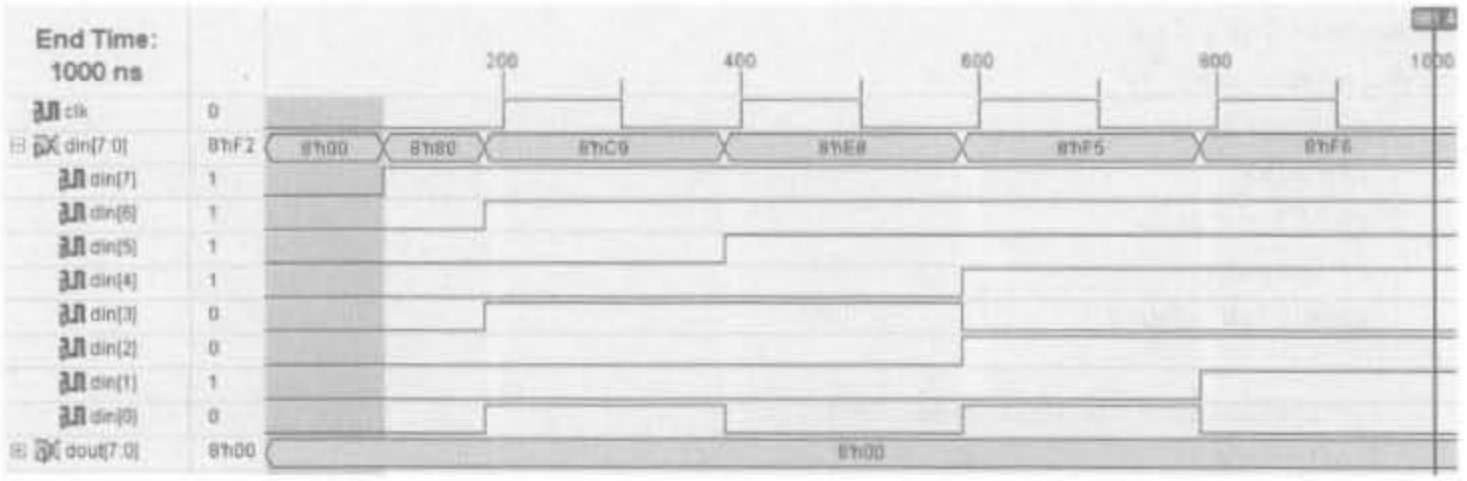


图 4-33 初始化输入

然后，将 testbench 文件存盘，ISE 会自动将其加入到仿真的分层结构中，在代码管理区会列出刚生成的测试文件 test_bench.tbw，如图 4-34 所示。

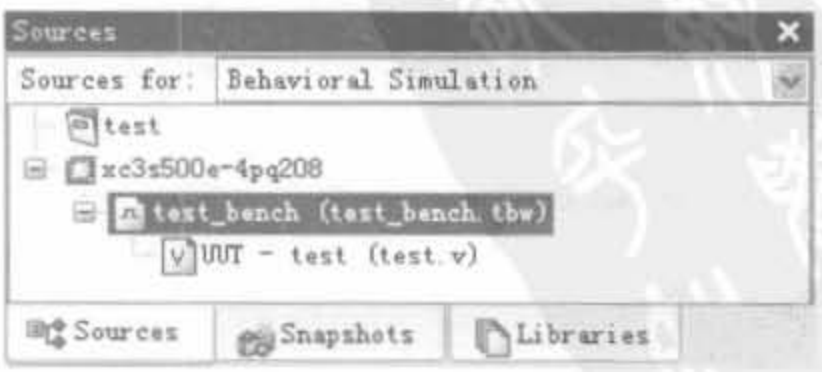


图 4-34 测试文件列表

选中 test_bench. tbw 文件，然后双击过程管理区的“Simulate Behavioral Model”，即可完成功能仿真。同样，可在“Simulate Behavioral Model”选项上单击鼠标右键，设置仿真时间等。例 4-3 的仿真结果如图 4-35 所示。从图中可以看出，dout 信号等于 din 信号加 1，功能正确。

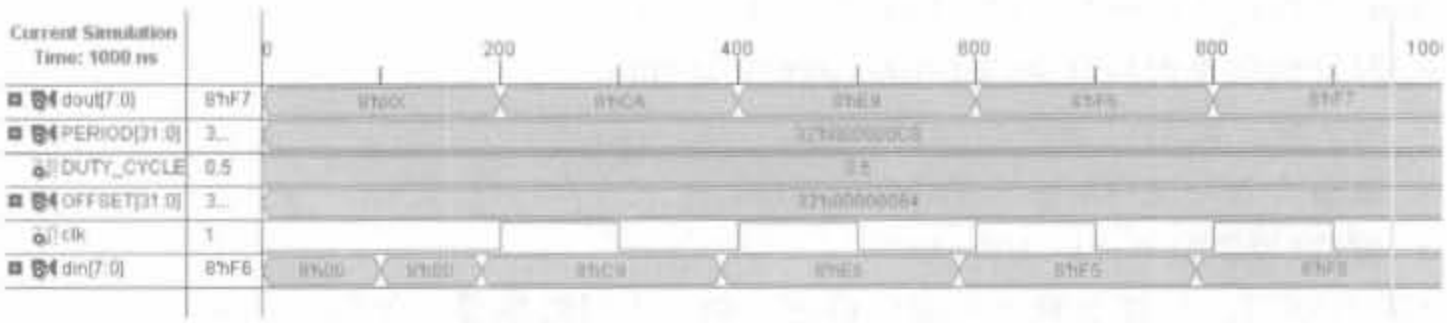


图 4-35　功能仿真结果

2. 测试代码法

下面介绍基于 Verilog 语言建立测试平台的方法。首先在工程管理区将“Sources for”设置为“Behavioral Simulation”，在任意位置单击鼠标右键，并在弹出的菜单中选择“New Source”命令，然后选中“Verilog Test Fixture”类型，输入文件名“test_test”，再单击“Next”进入下一页。这时，工程中所有 Verilog Module 的名称都会显示出来，设计人员需要选择要进行测试的模块。

用鼠标选中“test”，单击“Next”后进入下一页，直接单击“Finish”按钮，ISE 会在源代码编辑区自动显示测试模块的代码：

```
timescale 1ns / 1ps
module test_test_v;
    // Inputs
    reg clk;
    reg [7:0] din;
    // Outputs
    wire [7:0] dout;

    // Instantiate the Unit Under Test (UUT)
    test uut (
        .clk(clk),
        .din(din),
        .dout(dout)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        din = 0;
        // Wait 100ns for global reset to finish
        #100;
        // Add stimulus here
    end

endmodule
```

由此可见，ISE 自动生成了测试平台的完整架构，包括所需信号、端口声明以及模块调用的完成。所需完成的工作就是在 initial...end 模块中的“// Add stimulus here”后面添加测试向量生成代码。添加的测试代码如下：

```
forever begin
    #5;
    clk = !clk;
    if(clk == 1)
        din = din + 1;
    else
        din = din;
end
```

完成测试平台后，在工程管理区将“Sources for”选项设置为“Behavioral Simulation”，这时在过程管理区会显示与仿真有关的进程，如图 4-36 所示。

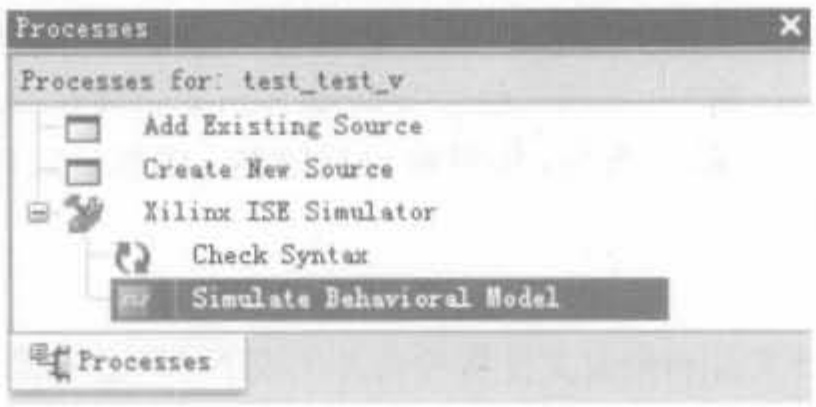


图 4-36 仿真过程示意图

选中图 4-36 中“Xilinx ISE Simulator”下的“Simulate Behavioral Model”项，单击鼠标右键，选择弹出菜单的“Properties”项，会弹出如图 4-37 所示的属性设置对话框。对话框最后一行的“Simulation Run Time”就是仿真时间的设置，可将其修改为任意时长。本例采用默认值。

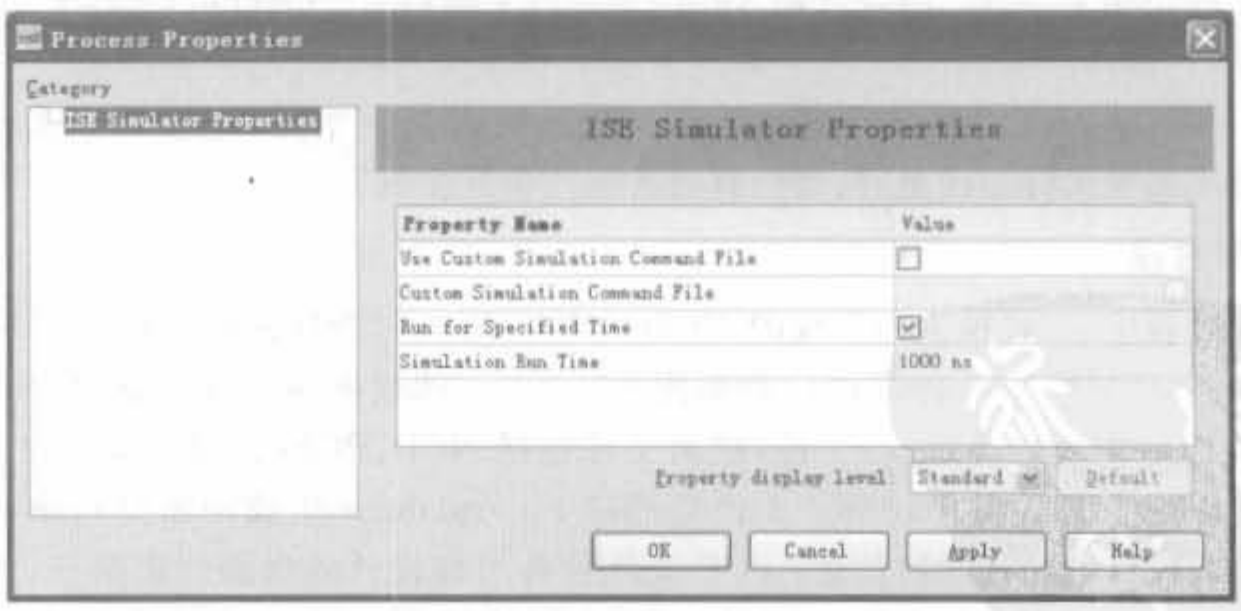


图 4-37 仿真属性设置对话框

仿真参数设置完后，就可以进行仿真了。直接双击 ISE Simulator 软件中的“Simulate Behavioral Model”，ISE 会自动启动 ISE Simulator 软件，并得到如图 4-38 所示的仿真结果。从中可以看到，设计达到了预计目标。

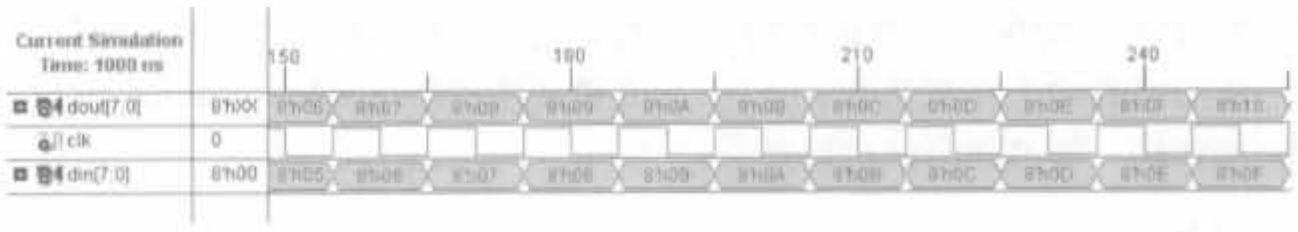


图 4-38　test 模块的仿真结果

4.3.3　基于 ISE 的实现

实现(Implement),是将综合输出的逻辑网表翻译成所选器件的底层模块与硬件原语,将设计映射到器件结构上,进行布局布线,达到在选定器件上实现设计的目的。实现主要分为 3 个步骤:翻译(Translate)逻辑网表、映射(Map)到器件单元和布局布线(Place & Route)。翻译的主要作用是将综合输出的逻辑网表翻译为 Xilinx 特定器件的底层结构和硬件原语(具体的原语见第 3 章中的原语介绍)。映射的主要作用是将设计映射到具体型号的器件上(LUT、FF、Carry 等)。布局布线步骤调用 Xilinx 布局布线器,根据用户约束和物理约束,对设计模块进行实际的布局,并根据设计连接,对布局后的模块进行布线,产生 FPGA/CPLD 配置文件。

1. 翻译过程

在翻译过程中,设计文件和约束文件将被合并生成 NGD(原始类型数据库)输出文件和 BLD 文件。其中,NGD 文件包含了当前设计的全部逻辑描述,BLD 文件是转换的运行和结果报告。实现工具可以导入 EDN、EDF、EDIF、SEDIF 格式的设计文件,以及 UCF(用户约束文件)、NCF(网表约束文件)、NMC(物理宏库文件)、NGC(含有约束信息的网表)格式的约束文件。翻译项目包括 3 个命令:

(1) Translation Report:用以显示翻译步骤的报告。

(2) Floorplan Design:用以启动 Xilinx 布局规划器(Floorplanner)进行手动布局,提高布局器效率。

(3) Generate Post-Translate Simulation Model:用以产生翻译步骤后仿真模型。由于该仿真模型不包含实际布线延时,所以有时省略此仿真步骤。

2. 映射过程

在映射过程中,由转换流程生成的 NGD 文件将被映射为目标器件的特定物理逻辑单元,并保存在 NCD(展开的物理设计数据库)文件中。映射的输入文件包括 NGD、NMC、NCD 和 MFP(映射布局规划器)文件,输出文件包括 NCD、PCF(物理约束文件)、NGM 和 MRP(映射报告)文件。其中,MRP 文件是通过 Floorplanner 生成的布局约束文件;NCD 文件包含当前设计的物理映射信息;PCF 文件包含当前设计的物理约束信息;NGM 文件与当前设计的静态时序分析有关;MRP 文件是映射的运行报告,主要包括映射的命令行参数、目标设计占用的逻辑资源、映射过程中出现的错误和告警、优化过程中删除的逻辑等内容。映射项目包括如下命令:

(1) Map Report:用以显示映射步骤的报告。

(2) Generate Post-Map Static Timing:产生映射静态时序分析报告,启动时序分析器

(Timing Analyzer)分析映射后静态时序。

(3) Manually Place & Route(FPGA Editor)：用以启动 FPGA 底层编辑器进行手动布局布线，指导 Xilinx 自动布局布线器，解决布局布线异常，提高布局布线效率。

(4) Generate Post-Map Simulation Model：用以产生映射步骤后仿真模型。由于该仿真模型不包含实际布线延时，所以有时也省略此仿真步骤。

3. 布局和布线过程

布局和布线(Place & Route)是指通过读取当前设计的 NCD 文件，将映射后生成的物理逻辑单元在目标系统中放置和连线，并提取相应的时间参数。布局布线的输入文件包括 NCD 和 PCF 模板文件，输出文件包括 NCD、DLY(延时文件)、PAD 和 PAR 文件。在布局布线的输出文件中，NCD 包含当前设计的全部物理实现信息，DLY 文件包含当前设计的网络延时信息，PAD 文件包含当前设计的输入/输出(I/O)管脚配置信息，PAR 文件主要包括布局布线的命令行参数、布局布线中出现的错误和告警、目标占用的资源、未布线网络、网络时序信息等内容。布局布线步骤的命令与工具非常多，主要的如下所述。

(1) Place & Route Report：用以显示布局布线报告。

(2) Asynchronous Delay Report：用以显示异步实现报告。

(3) Pad Report：用以显示管脚锁定报告。

(4) Guide Results Report：用以显示布局布线指导报告。该报告仅在使用布局布线指导文件 NCD 文件后才产生。

(5) Generate Post-Place & Route Static Timing：包含了进行布局布线后静态时序分析的一系列命令，可以启动 Timing Analyzer 分析布局布线后的静态时序。

(6) View/Edit Place Design (Floorplanner) 和 View/Edit Place Design (FPGA Editor)：用以启动 Floorplanner 和 FPGA Editor 完成 FPGA 布局布线的结果分析、编辑，手动更改布局布线结果，产生布局布线指导与约束文件，辅助 Xilinx 自动布局布线器，提高布局布线效率并解决布局布线中的问题。

(7) Analyze Power(XPower)：用以启动功耗仿真器分析设计功耗。

(8) Generate Post-Place & Route Simulation Model：用以产生布局布线后仿真模型。该仿真模型包含的延时信息最全，不仅包含门时延，还包含实际布线延时。该仿真步骤必须执行，以确保设计功能与 FPGA 实际运行结果一致。

(9) Generate IBIS Model：用以产生 IBIS 仿真模型，辅助 PCB 布板的仿真与设计。

(10) Multi Pass Place & Route：用以进行多周期反复布线。

(11) Back-annotate Pin Locations：用以反标管脚锁定信息。

经过综合后，在过程管理区双击“Implement Design”选项，就可以完成实现，如图 4-39 所示。经过实现后，能够得到精确的资源占用情况，如图 4-40 所示。

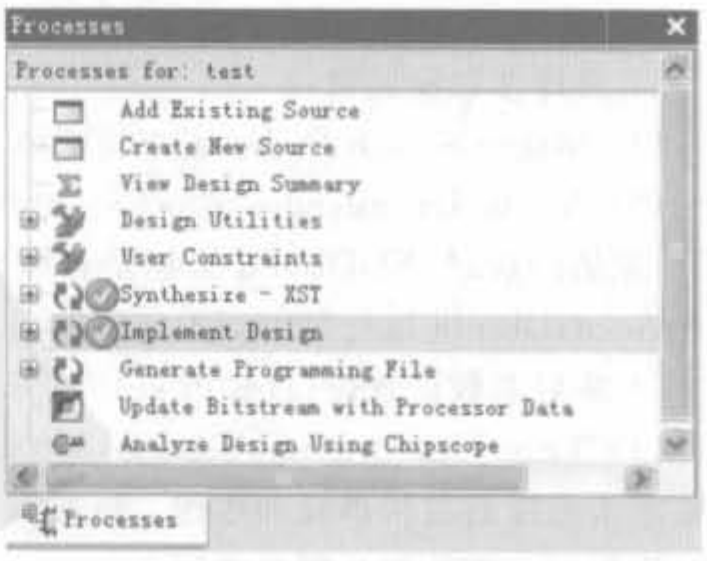


图 4-39 设计实现窗口

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    1
Logic Utilization:
  Number of 4 input LUTs:                8 out of  30,720    1%
Logic Distribution:
  Number of occupied Slices:                              5 out of  15,360    1%
    Number of Slices containing only related logic:       5 out of       5  100%
    Number of Slices containing unrelated logic:          0 out of       5    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:              9 out of  30,720    1%
  Number used as logic:                    8
  Number used as a route-thru:             1
  Number of bonded IOBs:                  17 out of     448    3%
  Number of BUFG/BUFGCTRLs:                1 out of      32    3%
    Number used as BUFGs:                  1
    Number used as BUFGCTRLs:              0
```

图 4-40　实现后的资源统计结果

4. 实现属性设置

一般在综合时，所有的属性都采用默认值。实际上，ISE 提供了丰富的实现属性设置。下面对 ISE 9.1 中内嵌的 XST 属性进行说明。打开 ISE 中的设计工程，在过程管理区选中“Implement Design”并单击鼠标右键，弹出界面如图 4-41 所示，包括翻译、映射、布局布线以及后仿时序参数等。

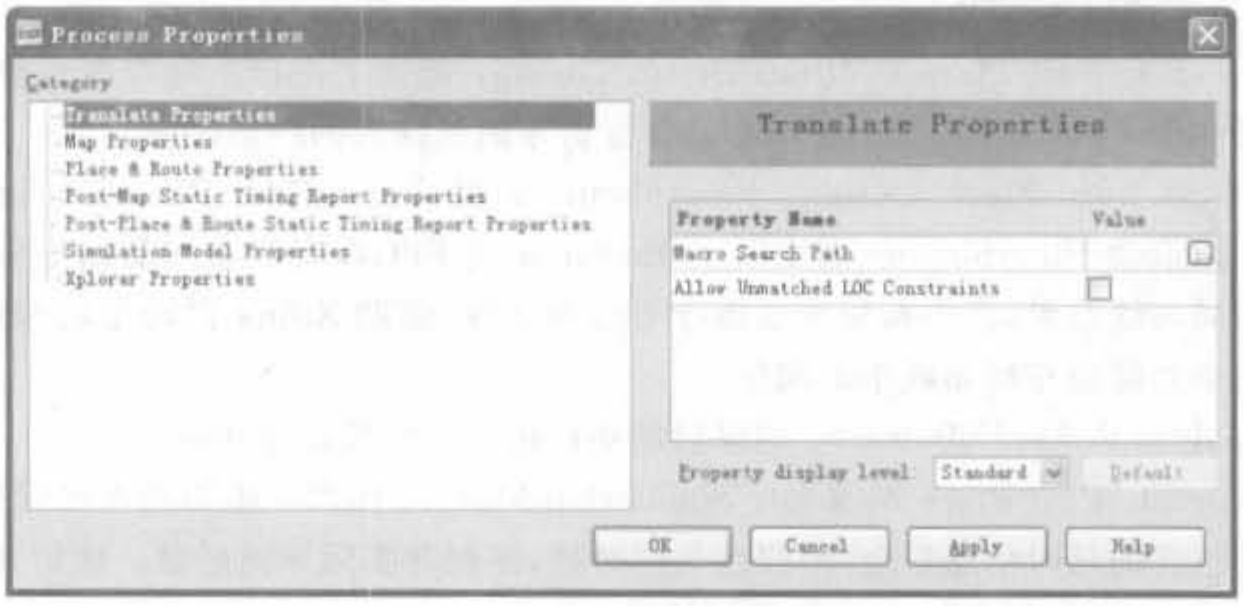


图 4-41　实现属性设置窗口

1）翻译参数设置窗口

(1) Macro Search Path：宏查找路径。用于提供宏的存放路径。

(2) Allow Unexpanded Blocks：允许未展开的逻辑块。用来说明当遇到不能展开 NGD 原语的块时，NGDBuild 工具是否继续运行。如果在设计中没有较低级的模块，该参数允许 NGDBuild 运行结束而不出现错误。默认值为 False。

2）映射参数设置窗口(如图 4-42 所示)

(1) Trim Unconnected Signals：整理未连接的信号。该参数用于控制在映射之前，是否整理未连接的逻辑单元和连线。该参数有助于评估设计中的逻辑资源，并获得部分设计的时序信息。默认值为需要整理。

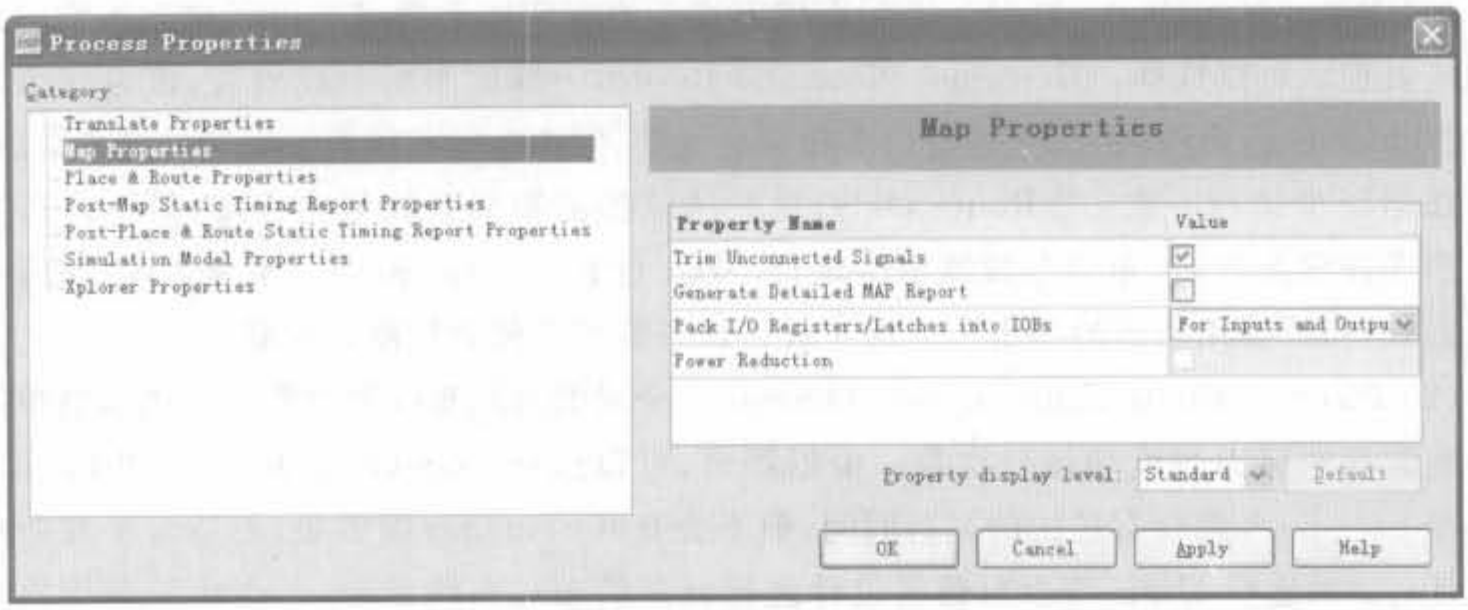


图 4-42 映射参数设置窗口

(2) Generate Detailed MAP Report：生成详细的映射报告。该参数用来选择是否需要生成详细的映射报告。详细的映射报告将提示在映射时去掉的多余逻辑块和信号，以及提示展开的逻辑，交叉引用的信号、符号等。默认值为不产生详细的映射报告。

(3) Pack I/O Registers/Latches into IOBs：选择输入/输出块中的寄存器/锁存器。该参数用来控制是否将器件内部的输入/输出寄存器用 IOB 中的寄存器/锁存器来取代。可以选择：①For Inputs and Outputs，尽可能将设计中的输入/输出寄存器放入 IOB；②For Inputs Only，仅考虑把输入寄存器放入 IOB；③For Outputs Only，仅考虑把输出寄存器放入 IOB；④Off，采用用户的设计要求进行处理，不考虑自动选择方式。

3) 布局布线参数设置窗口(如图 4-43 所示)

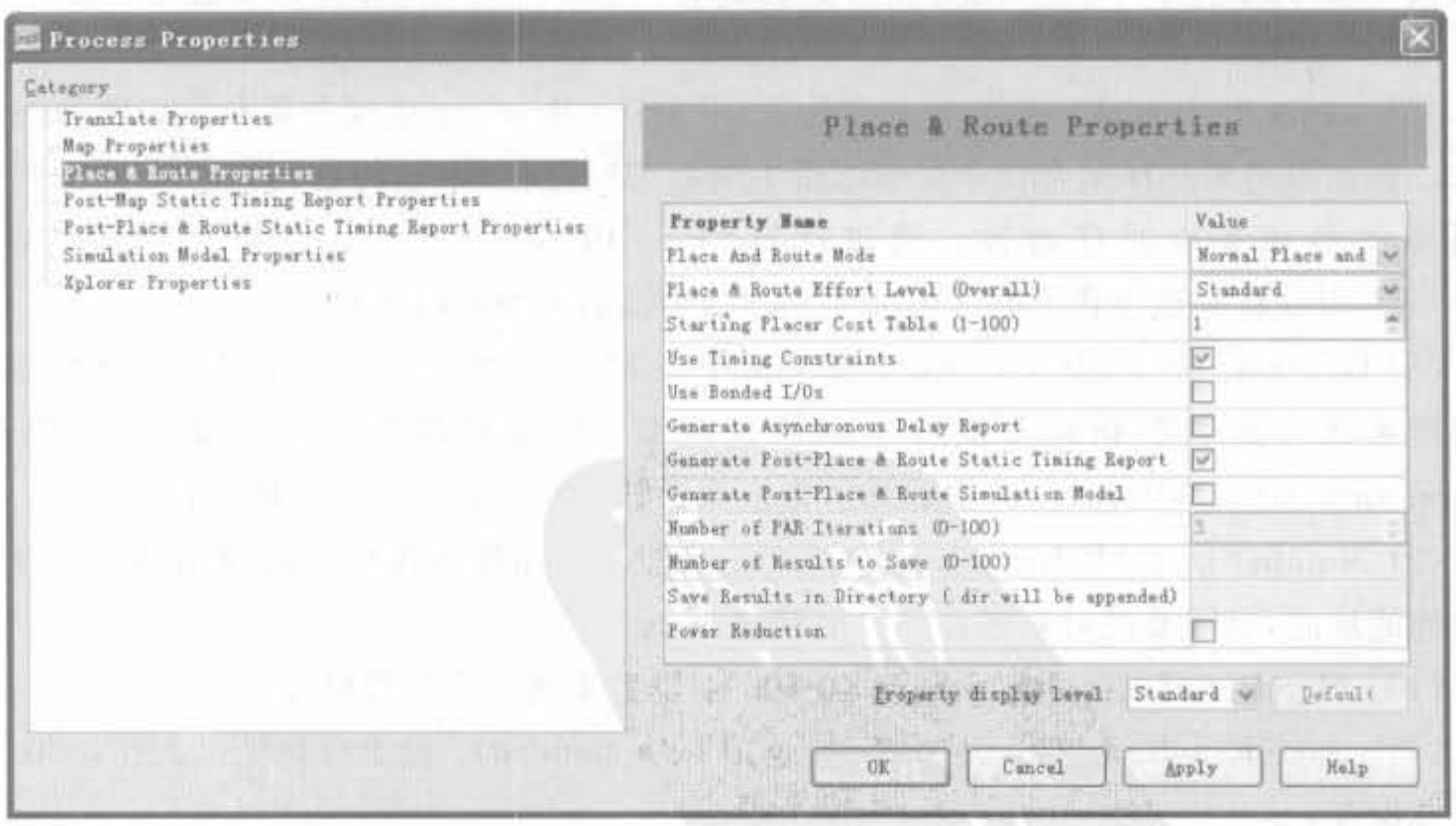


图 4-43 布局布线参数设置窗口

(1) Place And Route Mode：布局布线方式。该参数用来指定采用哪种方式来进行布局布线处理。可以选择：①Normal Place and Route，一般的布局布线处理，该方式为默认值；②Place Only，运行所选择的布局布线努力程度，但不运行布线器。当选择该参数后，布局布线器至少运行一次；③Route Only，运行所选择的布局布线努力程度，但不运行布局器。当选择该参数后，布局布线器至少运行一次；④Reentrant Route，重复布线，保持布局布线方式，布线器用当前的路由再一次布线。该布线器由努力程度来控制。

(2) Place & Route Effort Level (Overall)：全局的布局布线努力程度。该参数控制布局布线流程的努力程度和运行次数。根据需要，可以选择 Standard、Medium 和 High。如果选择 Standard，将会有最快的运行时间，但不会有好的布局布线效果，不适合于复杂的逻辑设计；如果选择 High，将会对逻辑设计进行反复的布局布线处理，并生成最理想的布局布线结果，高性能、复杂和最终的设计通常采用这种模式，但它比较费时。默认值为 Standard。

(3) Starting Placer Cost Table (1-100)：布局器运行开销表。默认值为 1。

(4) Use Timing Constraints：使用时序约束。在布局布线期间，需使用 UCF 和 PCF 文件中的时序约束条件。默认值为使用时序约束。

(5) Use Bonded I/Os：使用绑定的 I/O。该参数用来选择是否允许布局布线器将内部的输入/输出逻辑放到 I/O 脚未使用的用于绑定 I/O 的位置。该参数也允许布线资源穿过用于绑定 I/O 的位置。默认值为该参数无效。

(6) Generate Asynchronous Delay Report：生成异步延迟报告。该参数用来选择是否在布局布线运行时生成异步延迟报告。该报告列出了设计中所有的网线和网络上所有负载的延迟。通过执行“Asynchronous Delay Report Process”，可以打开该报告。默认值为不生成异步延迟报告。

(7) Generate Post-Place & Route Static Timing Report：生成布局布线后的静态时序报告。该参数用来选择是否在布局布线后生成静态时序报告，该报告列出了设计中所有信号通道的最坏条件时序特性。通过执行“Post-Place & Route Static Timing Report Process”，可以打开该报告。默认值为生成布局布线后的静态时序报告。

(8) Generate Post-Place & Route Simulation Model：生成布局布线后的仿真模型。该参数用来选择是否在布局布线后生成仿真模型。如果选择需要生成该模型，需要在“Simulation Model Properties”中选择仿真模型参数。默认值为不生成仿真模型。

(9) Number of PAR Iterations (0-100)：设置布局布线的迭代次数，数值越大，布局布线的性能越高，但需要的时间越长。一般默认为 3。

(10) Number of Results to Save (0-100)：设置实现结果的存储量。

(11) Save Results in Directory (.dir will be appended)：设定存储结果文件，.dir 文件也将被保存。

(12) Power Reduction：设置是否节省功率，默认为不选。

4) 映射后静态时序报告参数设置窗口(如图 4-44 所示)

(1) Report Type：设置时序报告类型，有 Error Report 和 Verbose Report 两种选择，

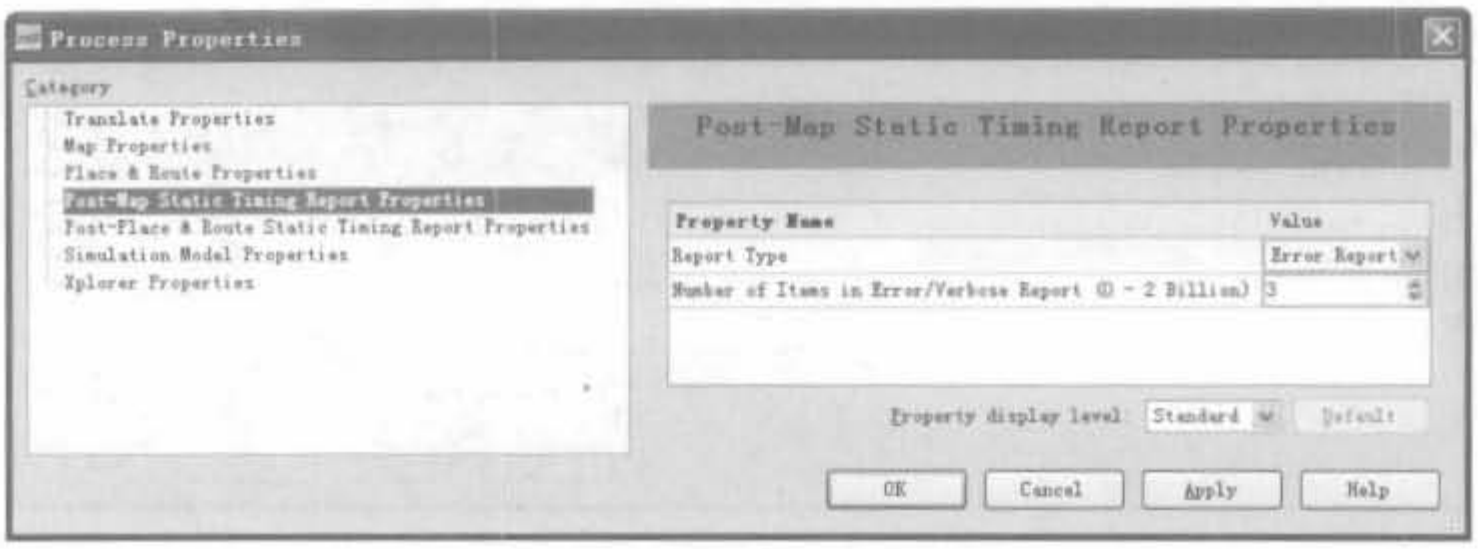


图 4-44 映射后静态时序报告参数设置窗口

默认为前者。前者只包括错误信息，比较简洁；后者是全部详细的分析结果，信息全面但冗余。

(2) Number of Items in Error/Verbose Report (0-2 Billion)：设置时序报告中保存的结果数，从 0 到 20 亿条。

5) 布局布线后静态时序报告参数设置窗口(如图 4-45 所示)

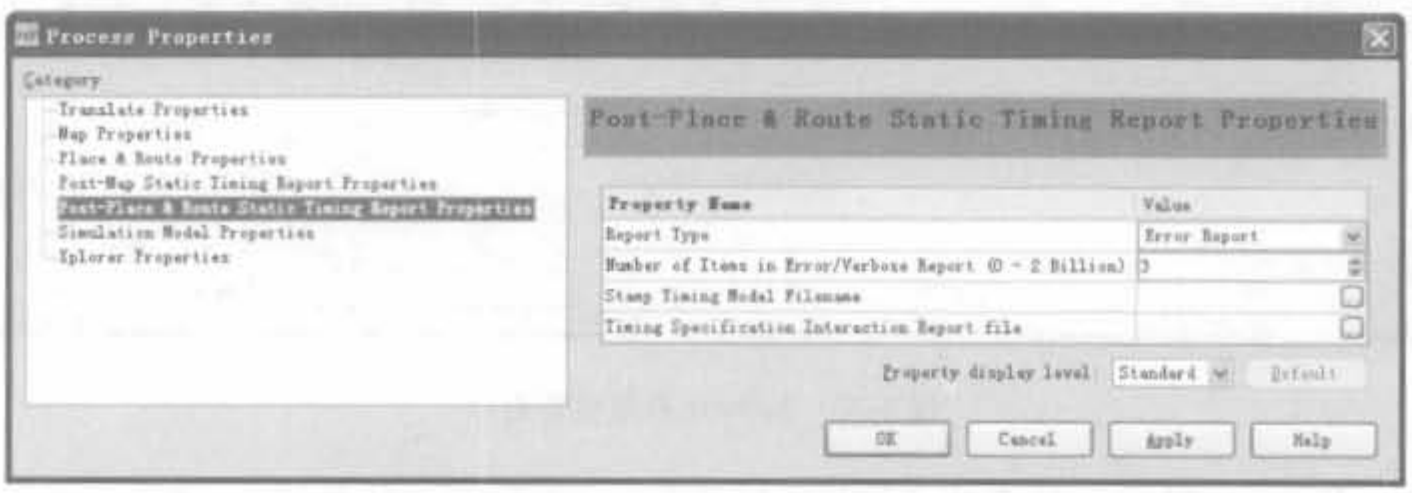


图 4-45 布局布线后静态时序报告参数设置窗口

(1) Report Type：设置时序报告类型，有 Error Report 和 Verbose Report 两种选择，默认为前者。

(2) Number of Items in Error/Verbose Report (0-2 Billion)：设置时序报告中保存的结果数，从 0 到 20 亿条。

(3) Stamp Timing Model Filename：用于标记时序模型的文件名。

(4) Timing Specification Interaction Report file：用于指定时序交互规范报告文件。

6) 仿真模型参数设置窗口(如图 4-46 所示)

(1) General Simulation Model Properties：普通的仿真模型特性。

(2) Simulation Model Target：用于选择仿真模型语言的种类，有 VHDL 和 Verilog 两种选择。

(3) Retain Hierarchy：用于设定是否保持分级结构，默认为保持。

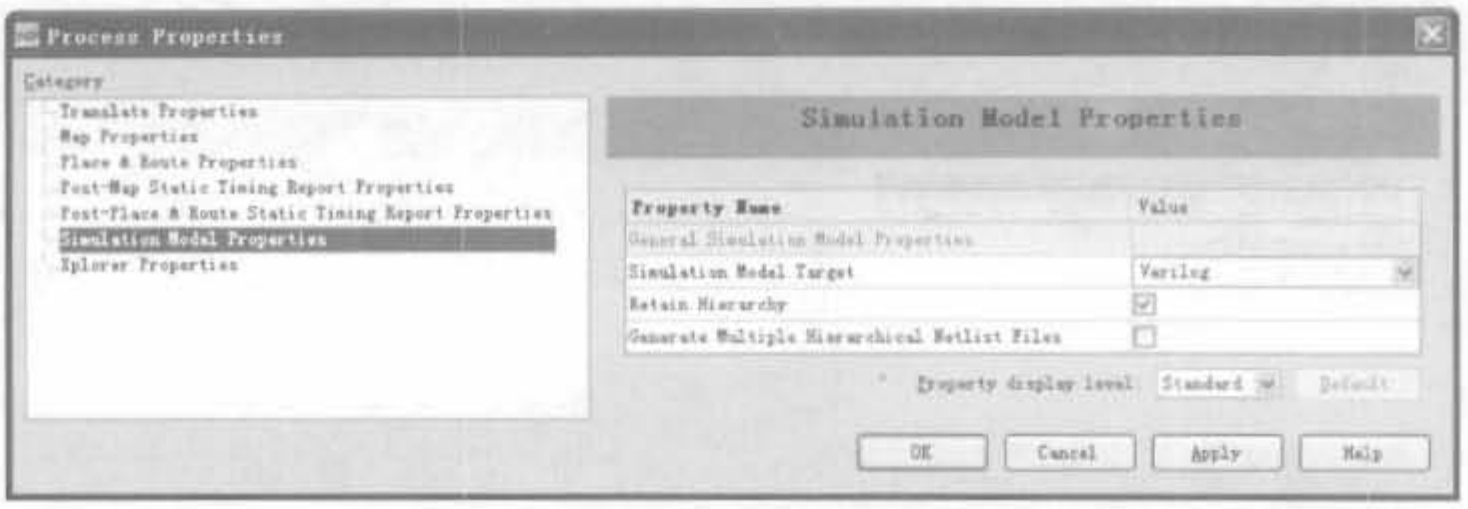


图 4-46　仿真模型参数设置窗口

(4) Generate Multiple Hierarchical Netlist Files：用于设定是否产生不同层次的网表文件，默认为不生成。

7) Xplorer 参数设置窗口(如图 4-47 所示)

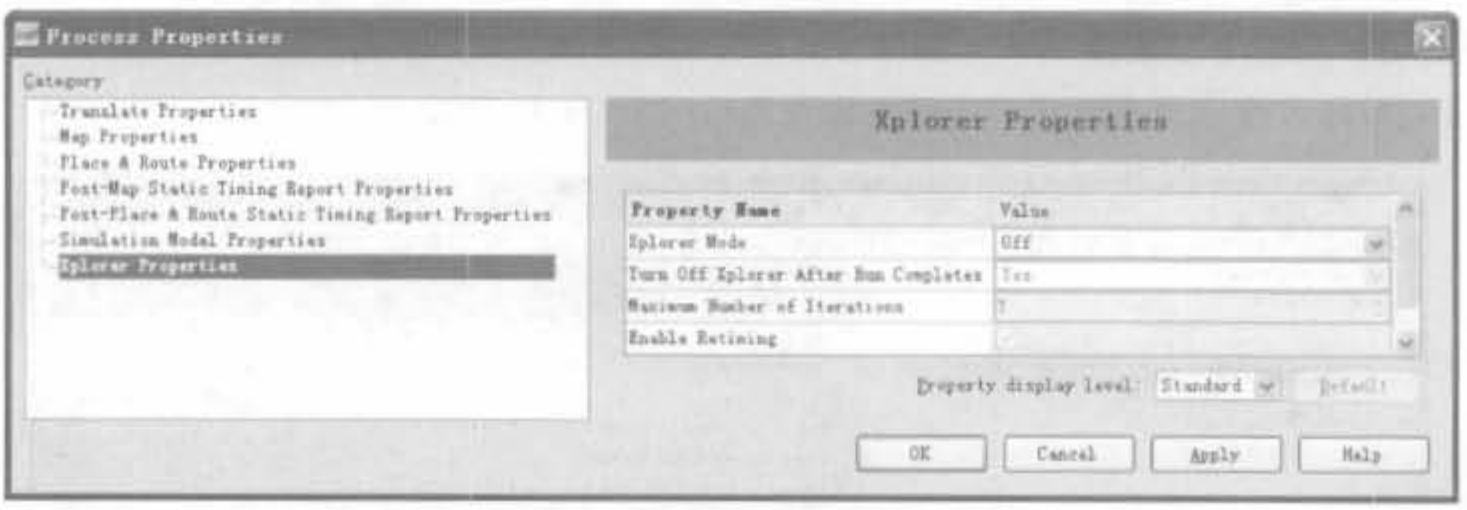


图 4-47　Xplorer 参数设置窗口

(1) Xplorer Mode：用于设定是否打开 Xplorer 模式，有 Off 和 Timing Closure 两种选择，默认为前者。

(2) Turn off Xplorer After Run Completes：是否在运行完毕后完毕 Xplorer 引擎，默认为关闭。

(3) Maximum Number of Iterations：用于设定最大迭代次数，默认为 7。

(4) Enable Retiming：是否使能时序重排，默认为使能。

4.3.4　基于 ISE 的芯片编程

本节简要介绍 ISE 软件中的芯片编程流程，详细的配置电路原理以及软件配置参数将在第 5 章讲解。生成二进制编程文件并下载到芯片中，也就是所谓的硬件编程和下载，是 FPGA 设计的最后一步。在 ISE 中，生成编程文件的操作非常简单，只需在过程管理区中双击“Generate Programming File”选项即可完成。完成后，该选项前面会出现一个打钩的圆

圈，如图 4-48 所示。生成的编程文件放在 ISE 工程目录下，是一个扩展名为. bit 的位流文件。

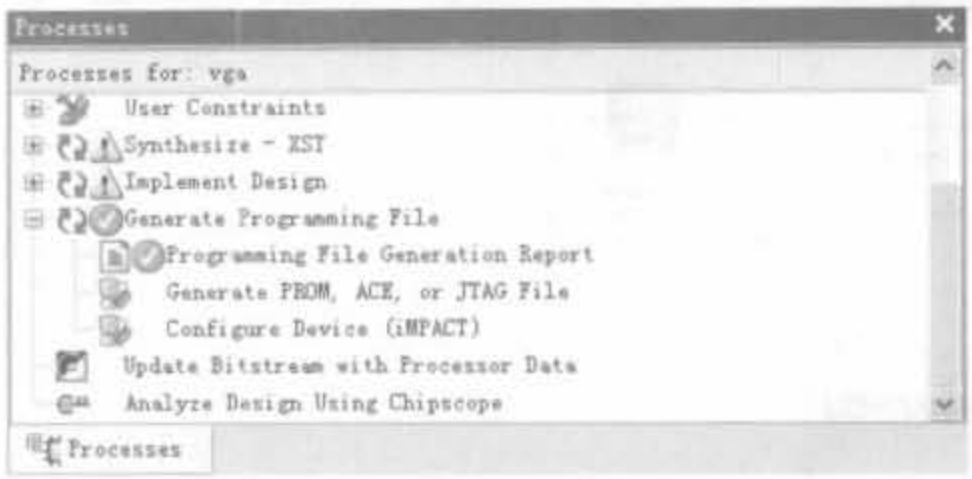


图 4-48 生成编程文件的窗口

到此，只剩下完成设计的最后一步——下载。双击过程管理区的“Generate Programming File”选项下面的“Configure Device(iMPACT)”项，然后在弹出的“Configure Device”对话框中选取合适的下载方式，ISE 会自动连接 FPGA 设备。成功检测到设备后，会出现如图 4-49 所示的 iMPACT 的主界面。

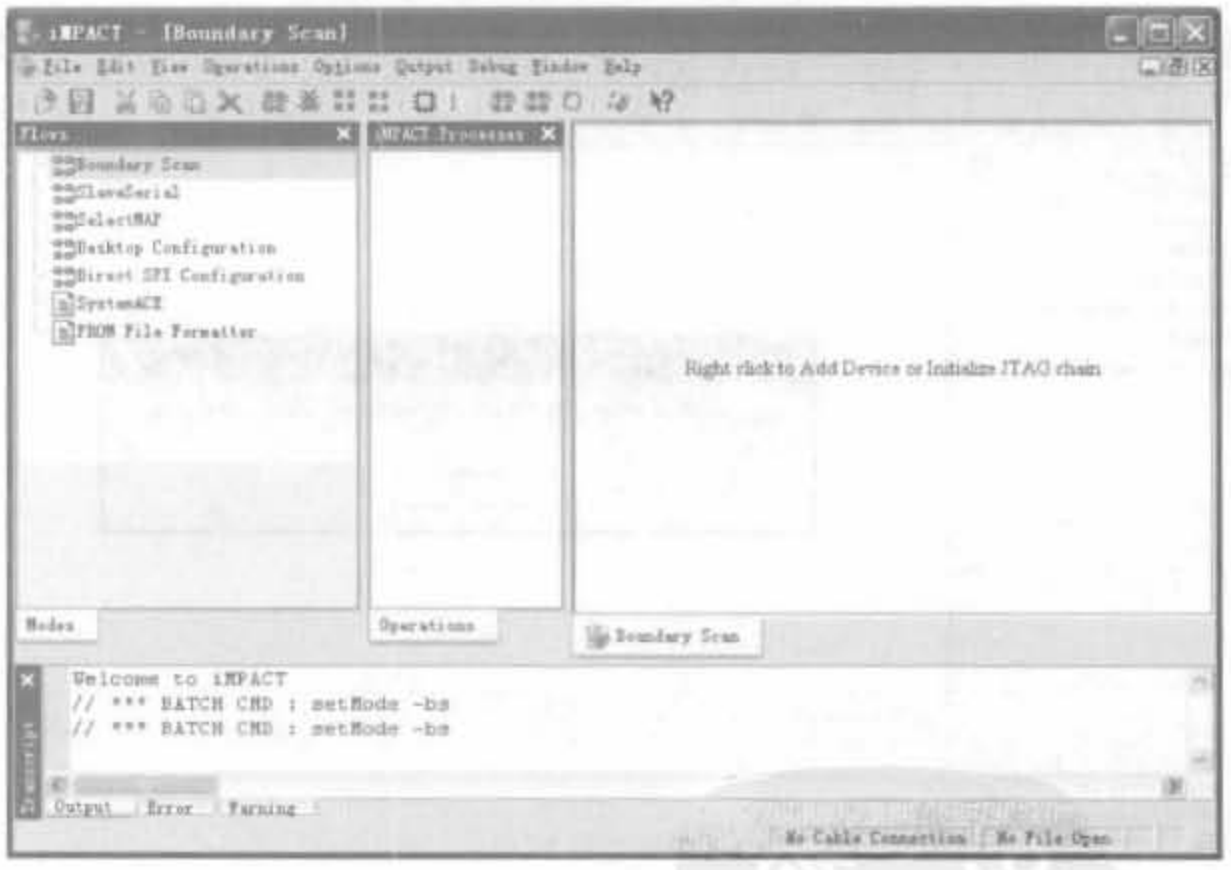
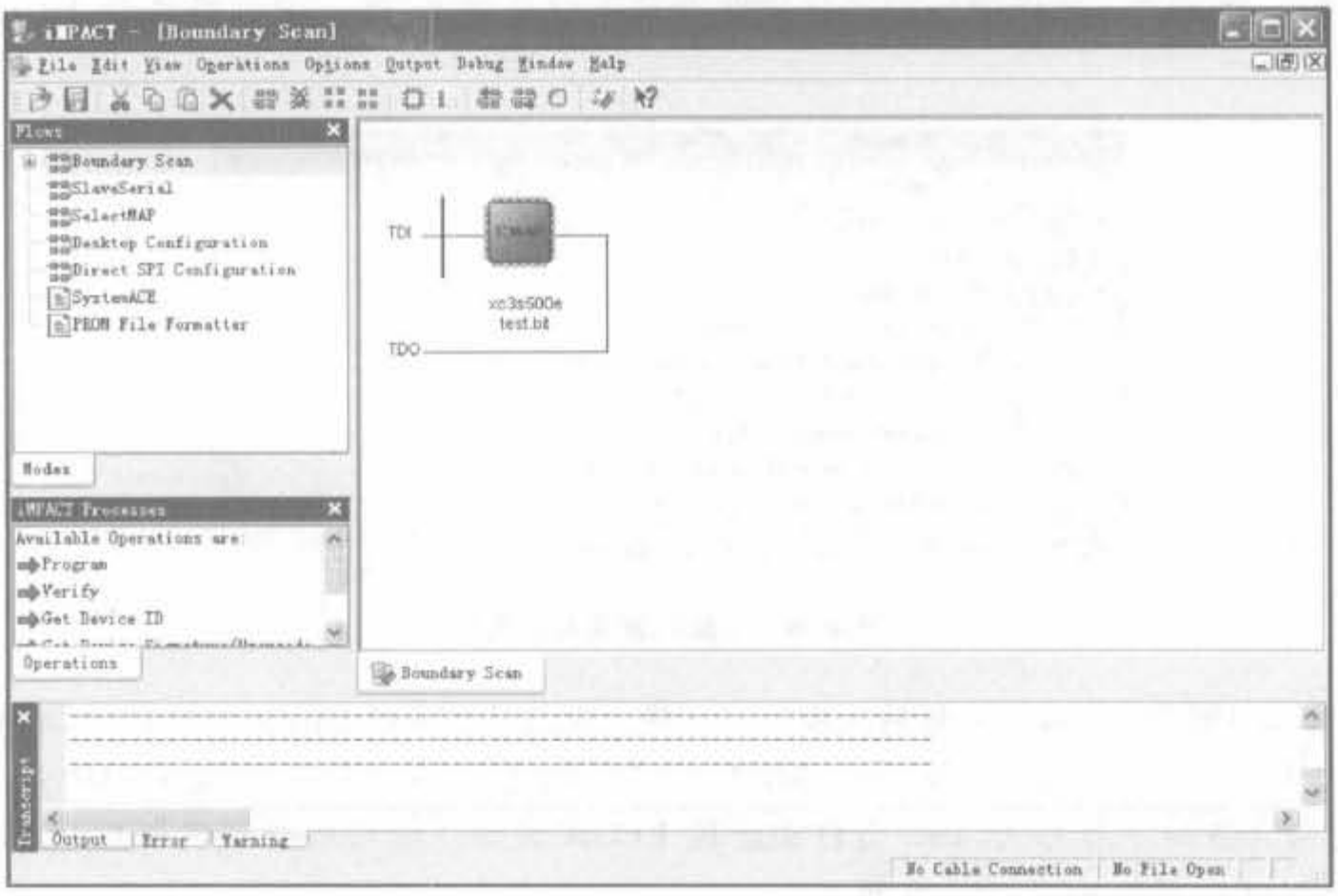


图 4-49 iMPACT 主界面

在主界面的中间区域内单击鼠标右键，并选择菜单的“Initialize Chain”选项。如果 FPGA 配置电路 JTAG 测试正确，会将 JTAG 链上扫描到的所有芯片在 iMPACT 主界面上列出来，如图 4-50(a)所示；如果 JTAG 链检测失败，弹出的对话框如图 4-50(b)所示。

JTAG 链检测正确后，在期望的 FPGA 芯片上单击鼠标右键，在弹出的菜单中选择“Assign New Configuration File”，会弹出如图 4-51 所示的窗口，让用户选择后缀为. bit 的二进制比特流文件。



(a) JTAG链扫描正确后的窗口界面

(b) JTAG链扫描错误后的窗口界面

图 4-50 JTAG 链扫描结果示意图

选中下载文件后，单击“打开”按钮，在 iMPACT 的主界面会出现一个芯片模型以及位流文件的标志。在此标志上单击鼠标右键，在弹出的对话框中选择“Program”选项，就可以对 FPGA 设备进行编程，如图 4-52 所示。

图 4-51 选择位流文件

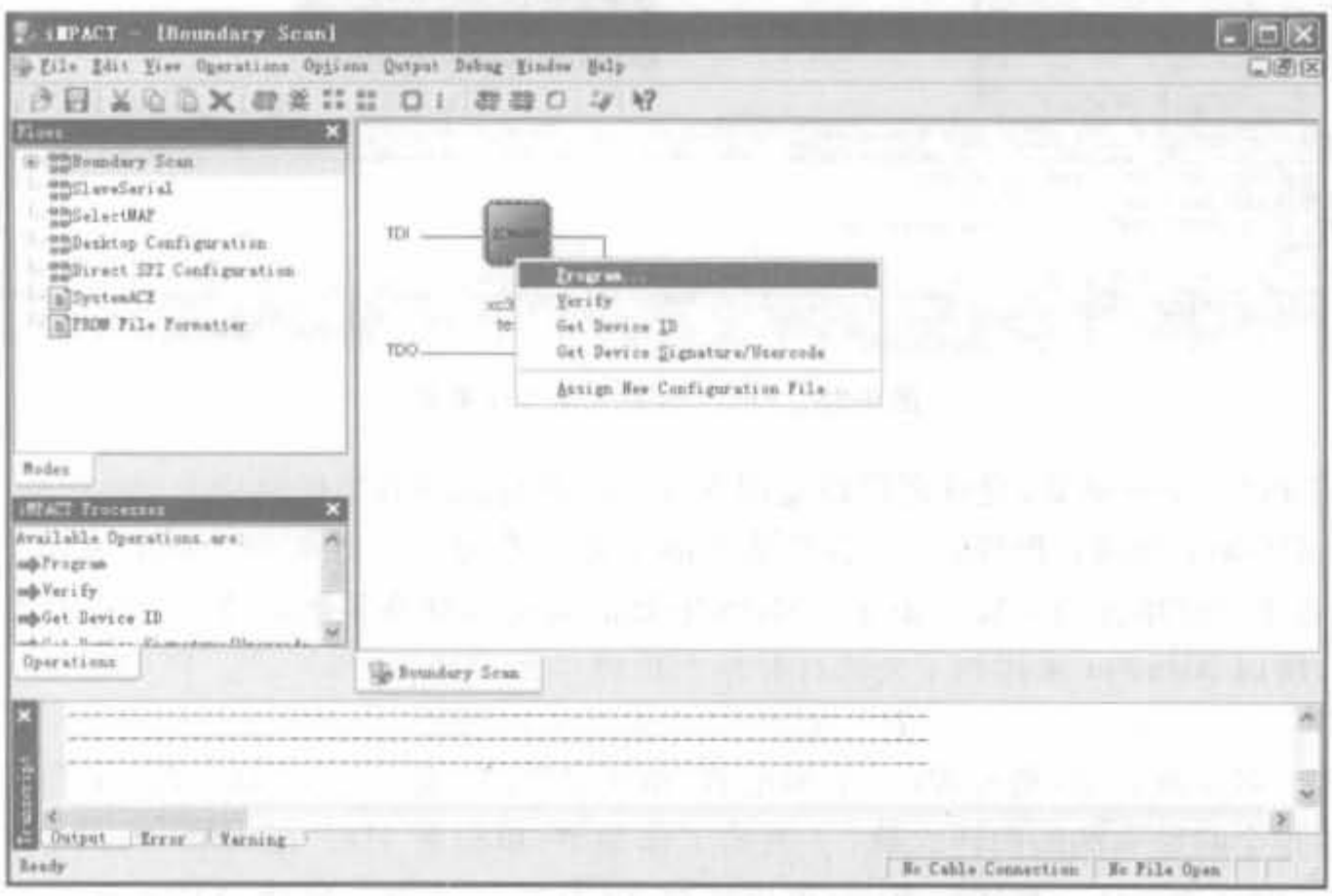


图 4-52 对 FPGA 设备进行编程示意图

配置成功后，弹出配置成功的界面，如图 4-53 所示。

至此，就完成了一个完整的 FPGA 设计流程。当然，ISE 的功能十分强大，以上介绍的只是其中最基本的操作，更多的内容和操作需要读者通过阅读 ISE 在线帮助来了解，并在大量的实践中来熟悉。

4.3.5 功耗分析以及 XPower 的使用

1. 功耗分析简介

在 FPGA 设计中，功耗分析是成功设计的重要环节。针对 FPGA 设计中的功耗分析，Xilinx 公司推出了简单的速查表格和专用的功耗分析工具——XPower。对于开发初期的 FPGA 功耗估算，设计者一般使用 Xilinx 公司提供的简单图表和公式。例如，对于 XC9500 系列器件的简单功耗分析，可以在其数据手册中找到估算公式。Virtex-E 和 Virtex-Ⅱ系列器件的简单功耗分析可以通过 Xilinx 公司提供的功耗估算表格完成。对于基本完成逻辑

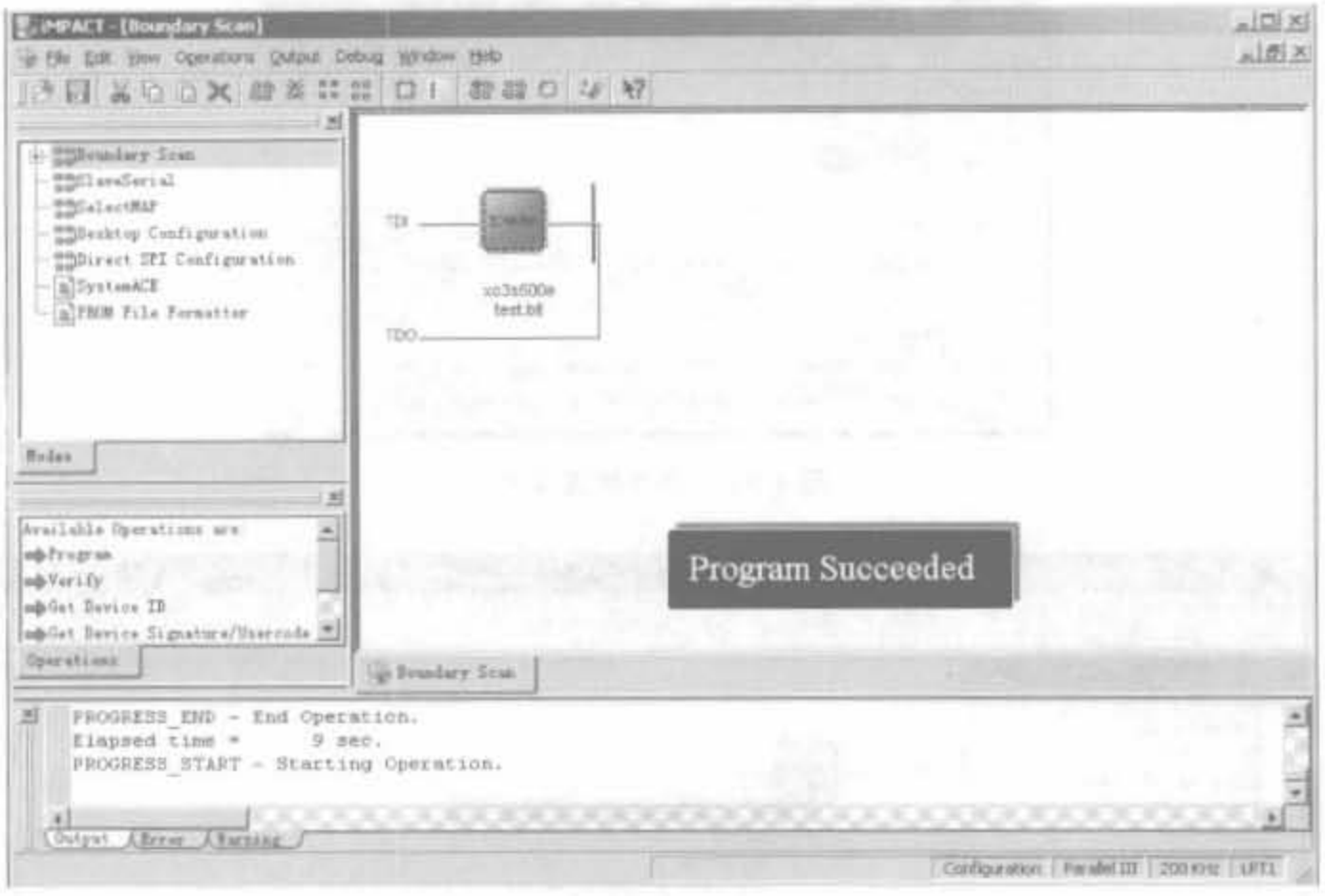


图 4-53 FPGA 配置成功指示界面

设计的 FPGA 功耗估算，设计者可以使用 XPower 进行详细的功耗分析。

在 XPower 功耗分析过程中，功耗估算的基本方法是：①计算每个设计单元的功耗；②累加各个设计单元的功耗。由于 CMOS 电路的动态功耗在很大程度上取决于电路的翻转频率，所以 XPower 采用如下公式计算单个设计单元的功耗[7]：

$$P = C \times V^2 \times E \times f \times 1000$$

式中，P 表示功耗，单位是 mW；C 表示电容，单位是 F；V 表示电压，单位是 V；E 表示翻转频率，指每个时钟周期的翻转次数；f 表示工作频率，单位是 Hz。在 XPower 中，翻转频率既可以采用全局默认的翻转频率，也可以通过 VCD 文件获得。另外，XPower 允许手工输入各个设计单元的翻转频率。

在 XPower 功耗分析过程中，主要涉及 NCD 文件、CTX 文件、PCF 文件、VCD 文件和 PWR 文件。其中，NCD 文件是经过实现的 FPGA 设计文件；CTX 文件是经过物理实现(FIT)的 CPLD 设计文件；PCF 文件是物理设计约束文件，该文件包含当前设计的时钟频率、电压等特性参数；PWR 文件是 XPower 的功耗分析报告；VCD 文件是对当前设计进行仿真后生成的文件，该文件包含了每个设计单元的翻转频率。

2. XPower 的用户界面以及使用流程

XPower 是一种设计后工具，用于分析实际器件利用率，并结合实际的适配后(post-fit)仿真数据(VCD 文件格式)，给出实际功耗数据。利用 XPower，用户可以在完全不接触芯片的情况下分析设计改变对总功耗的影响。

1) XPower 用户界面

XPower 的启动方法有两种：一种是单独启动 XPower，直接单击“开始”→“程序”→“Xilinx ISE 9.1i”→“Accessories”→“XPower”即可启动；另一种是在工程经过布局布线后，在过程管理区双击“Implementation”→“Place and Router”→“Analyze Power

(XPower)",打开 XPower,并自动加载当前工程。在例 4-3 所示工程中,采用后一种方法打开 XPower,其用户界面如图 4-54 所示。

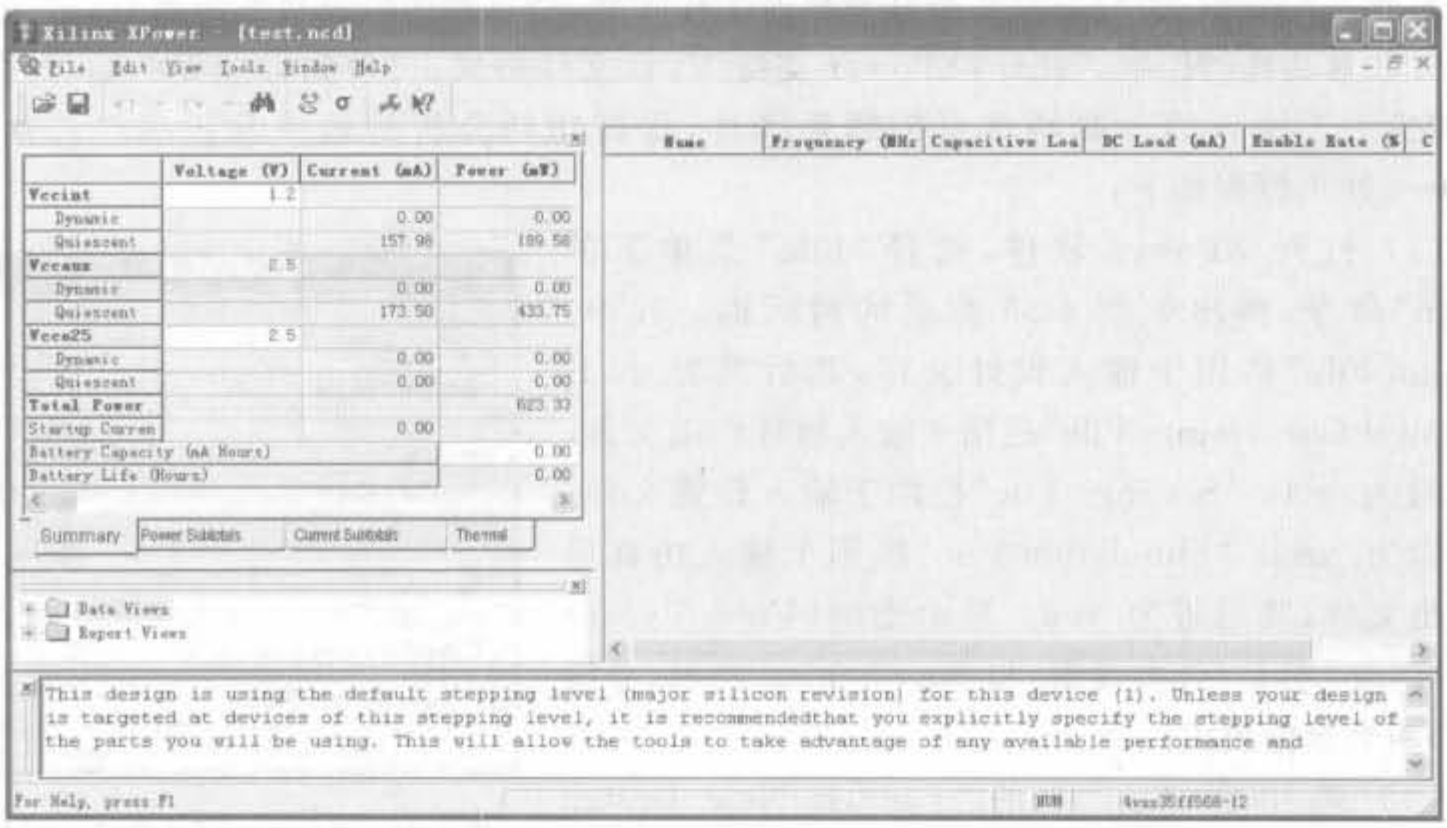


图 4-54 XPower 的启动

XPower 的用户界面由菜单栏、工具栏、功耗结果显示窗口、表格分析窗口、浏览窗口、信息显示窗口以及状态栏等部分组成。

在 ISE 过程管理区,双击"Generate Power Data"命令将自动生成设计的功耗分析报告,再双击"View XPower Report"命令来阅读报告。典型的分析报告(例 4-3 中的功耗分析报告)如图 4-55 所示。报告会给出与各个电压相对应的电流大小以及功耗,便于设计人员设计相应的电源模块。

```
--------------------------------------------------------------------
Release 9.1.03i - XPower SoftwareVersion:J.33
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
Design:       test.ncd
Preferences:  test.pcf
Part:         4vsx35ff668-12
Data version: ADVANCED,v1.0,08-26-04

Power summary:                          I(mA)     P(mW)
--------------------------------------------------------------------
Total estimated power consumption:                  623
                                        ---
                      Vccint 1.20V:      158      190
                      Vccaux 2.50V:      174      434
                      Vcco25 2.50V:        0        0
                                        ---
                           Inputs:         0        0
                            Logic:         0        0
                          Outputs:
                            Vcco25         0        0
                          Signals:         0        0
                                        ---
          Quiescent Vccint  1.20V:      158      190
          Quiescent Vccaux  2.50V:      174      434

Thermal summary:
--------------------------------------------------------------------
   Estimated junction temperature:                  33C
                  Ambient temp:  25C
                      Case temp:  33C
                Theta J-A range:  13 -  13C/W

Analysis completed: Sat Feb 09 18:33:11 2008
--------------------------------------------------------------------
```

图 4-55 典型的 XPower 分析报告

2）XPower 的操作流程

要准确利用 XPower 来分析功耗，关键是确定信号的工作频率、信号翻转率以及输出负载等参数。XPower 允许设计人员任意修改工作频率、信号翻转以及负载等参数，以便观察某一参数对功耗的影响。此外，XPower 支持 VCD 文件格式，可得到最全面的信号翻转信息，减少手工输入信号翻转信息的烦琐操作，提高功耗分析的效率和正确性。典型的 XPower 使用流程如下：

（1）打开 XPower 软件，选择“File”菜单下的“Open”命令，弹出如图 4-56 所示的对话框。其中，“Design File”栏用于输入设计文件，其后缀为. ncd；“Physical Constraints File”栏用于输入物理约束文件，其后缀为. pcf；“Settings File”栏用于输入设置文件，其后缀为. xml；“Simulation File”栏用于输入仿真后的输出文件，其后缀为. vcd。显示类型（View Types）分为 Types 类型（分类视窗）和 Hierarchical 类型（分层视窗）两类视窗模式。

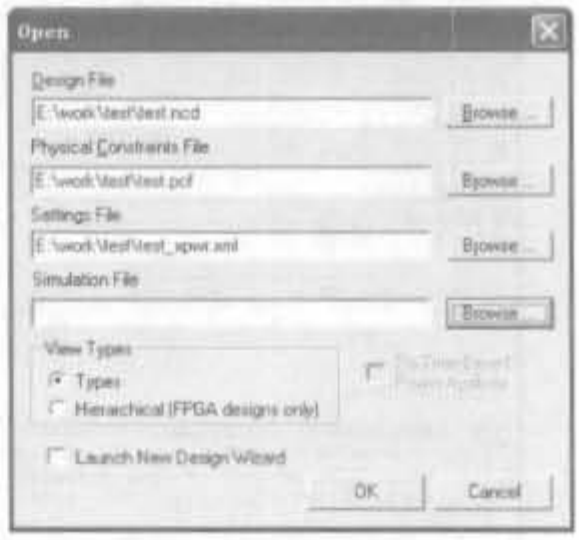


图 4-56　XPower 打开文件对话框

（2）选中图 4-56 中的“Launch New Design Wizard”参数，单击“OK”按钮，启动设计向导，弹出如图 4-57 所示的对话框。

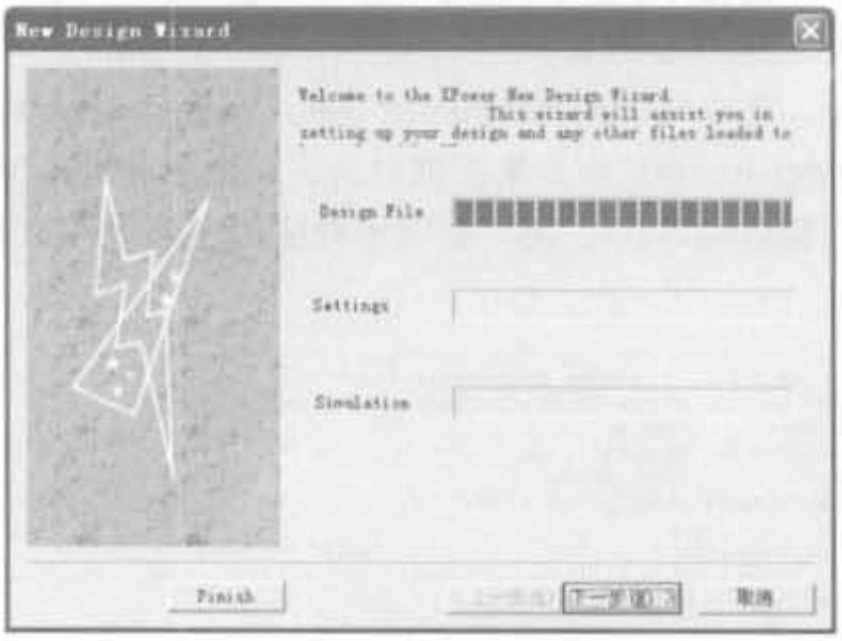


图 4-57　设计向导对话框

单击图 4-57 中的“下一步(N)”按钮，进入参数配置窗口。参数配置整体分为 4 个步骤。第 1 个步骤的界面如图 4-58 所示。

其中，“Voltages Sources”用于设置 FPGA 芯片的电源参数，包括核电压、辅助电压以及 I/O 端口参考电压。根据所选的器件输入相应的数值，一般选用默认值即可；“Battery Capacity and Battery Life”栏用于设定电池的容量和寿命，如输入电池容量为 1000mAH，则会自动给出电池的使用寿命为 3.02 小时；“Thermal”栏用于设定温度参数，“Ambient”用于设定环境温度，“Air”设定空气对流强度。

（3）单击图 4-58 中的“下一步(N)”按钮，进入设计向导第 2 步，设置各个信号的工作频

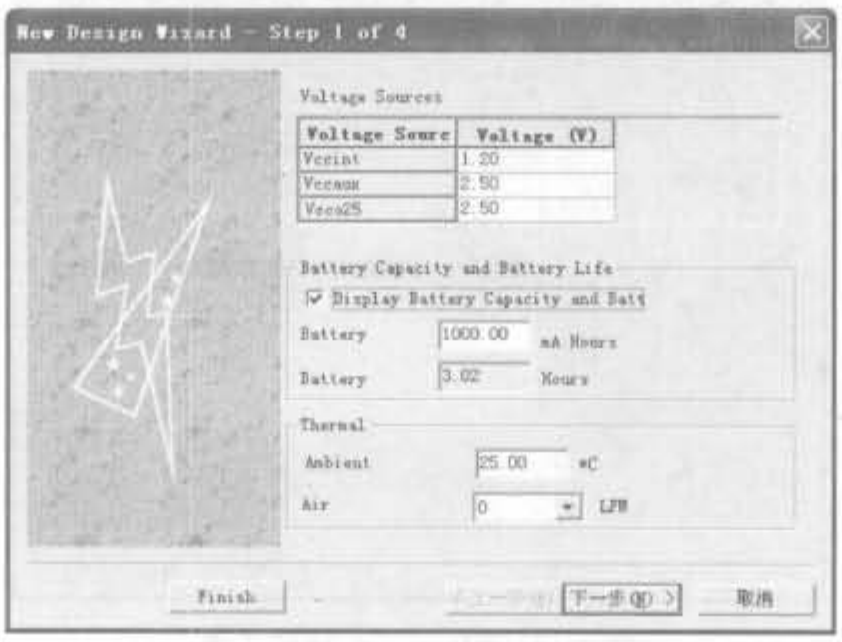


图 4-58 XPower 向导第 1 步

率,包括同步信号和异步信号,其界面如图 4-59 所示。选中相应的信号,在"Frequency"文本框中输入大小,并选择合适的单位,单击"Apply"按钮确认,也可单击"Reset"按钮复位到初始值。

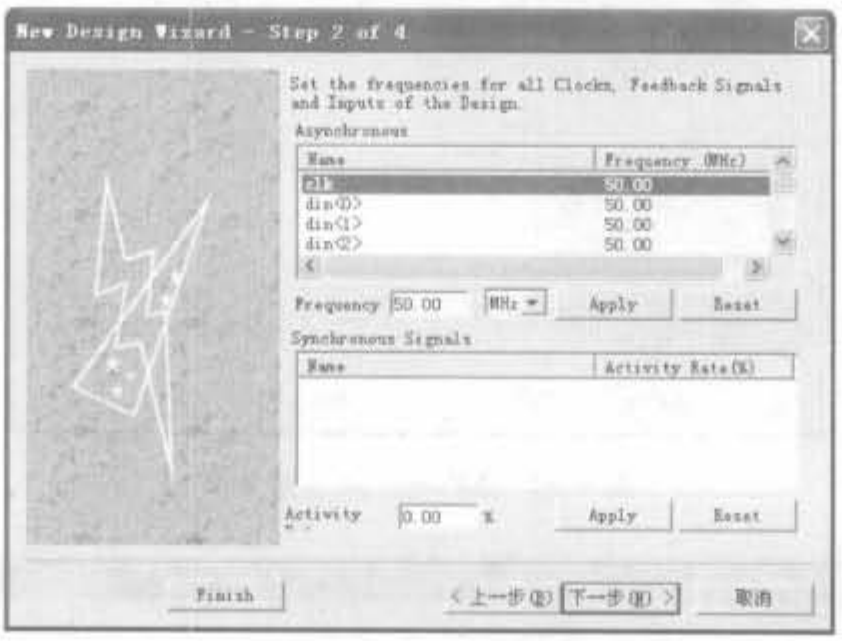


图 4-59 XPower 向导第 2 步

(4) 单击图 4-59 中的"下一步(N)"按钮,进入设计向导第 3 步,设置信号的输出负载电容,其界面如图 4-60 所示。选择信号后,输入电容大小,单击"Apply"按钮确认,也可以单击"Reset"按钮复位到初始值。

(5) 单击图 4-60 中的"下一步(N)"按钮,进入设计向导第 4 步,设置信号的直流电流负载,其界面如图 4-61 所示。选择信号后,输入电流大小,单击"Apply"按钮确认,也可以单击"Reset"按钮复位到初始值。此时,确认无误后,可单击"Finish"按钮完成设计向导;如有错误,单击"上一步(B)"按钮返回到先前的界面进行修改。

(6) 单击 XPower 软件"File"菜单下的"Open Setting File…"命令,可加入设置文件;再选择"Open Simulation File…"命令,加入仿真工具生成的.vcd 文件。选择相应的约束文件后,XPower 会在信息提示窗口给出加载成功的指示信息。

等加载完约束文件后,可单击浏览器窗口的"Report Views"→"Power Report (HTML)",查看设计功耗分析结果,其界面如图 4-62 所示。

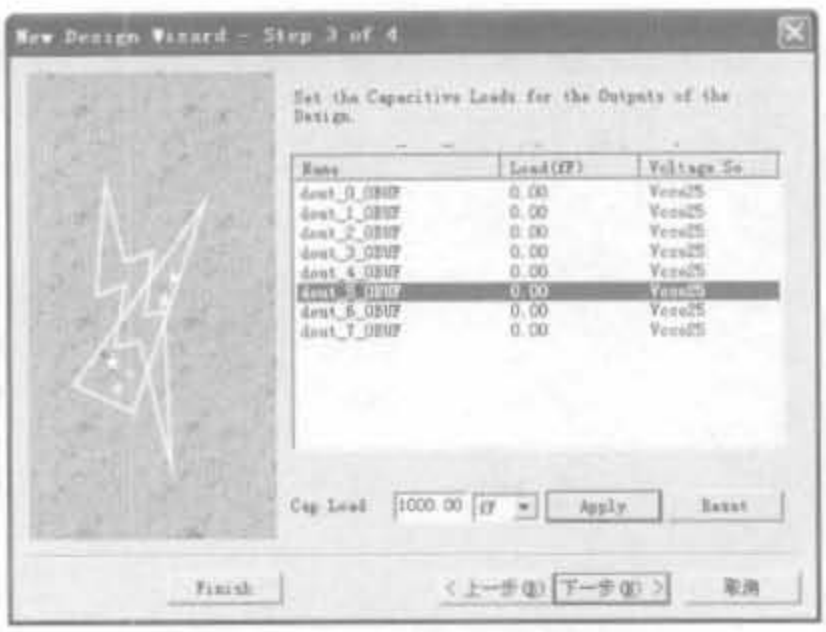


图 4-60　XPower 向导第 3 步

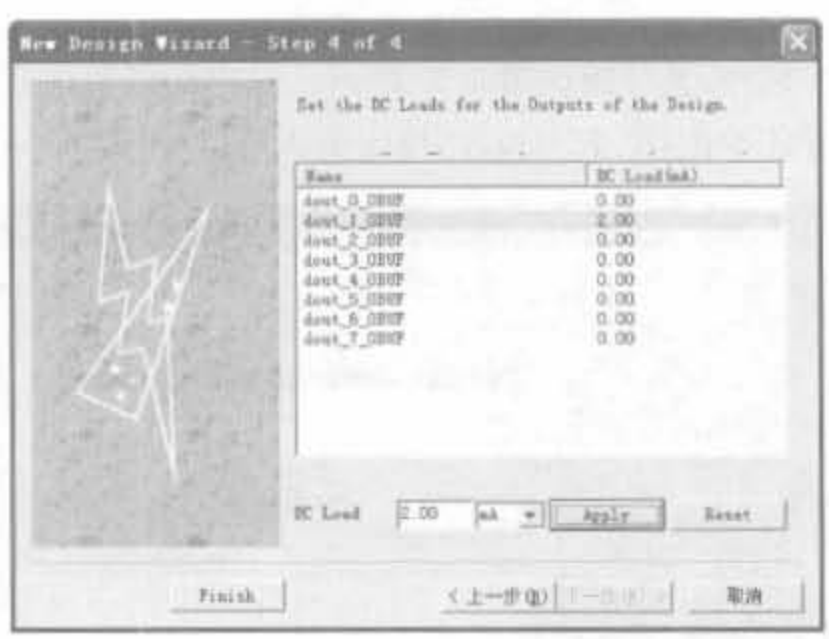


图 4-61　XPower 向导第 4 步

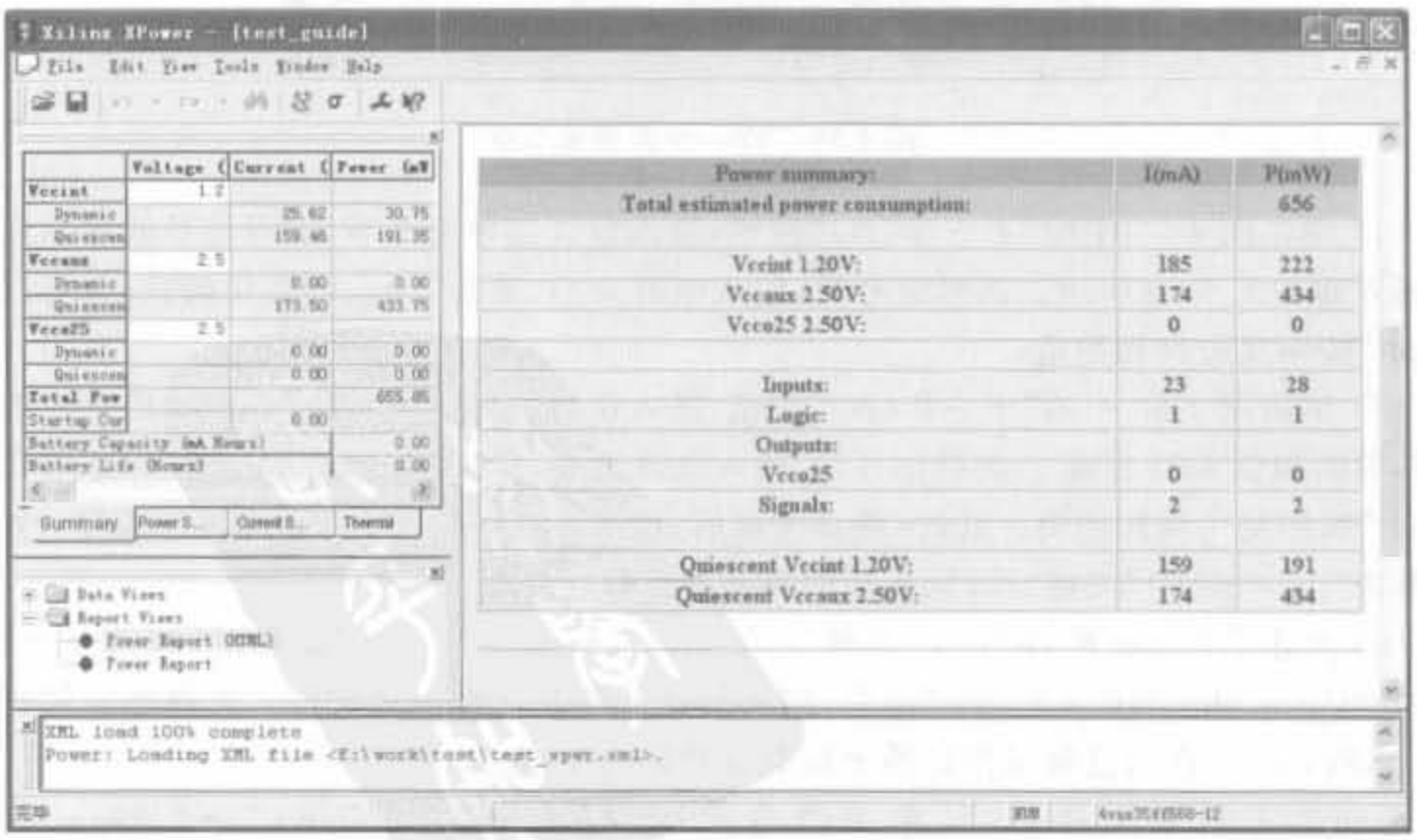


图 4-62　XPower 输出结果

3) VCD 文件

在 XPower 功耗分析过程中，为保证功耗分析报告基本符合实际情况，应该准确设置功耗分析参数。在 XPower 中，用户可以手工设置功耗分析参数，也可以通过读入 VCD 文件自动设置功耗分析参数。利用 VCD 文件设置功耗分析参数，可以使当前设计的信号翻转频率、输出负载等参数更符合实际情况。在使用 ModelSim 进行仿真时，VCD 文件不是默认的输出文件，用户需要在仿真过程中指定输出相应的 VCD 文件。其中，仿真 Verilog HDL 设计文件时，需要在 Testbench 文件中加入适当语句，如下所示：

```
initial begin
        // design.VCD 表示将要生成的 VCD 文件名
        $dumpfile("design.VCD");
        //testbench 表示测试名称，uut 表示待测试的模块名称
        $dumpvars(1,testbench.uut);
end
```

仿真 VHDL 设计文件时，需要在 DO 文件中加入适当语句，如下所示：

```
--design.VCD 表示将要生成的 VCD 文件名；
.VCD file design.VCD
--testbench 表示测试名称，uut 表示待测试的模块名称
.VCD add testbench/uut/*
```

3. 简易的功耗分析方法

Xilinx 提供了两种基于电子数据表和基于网站的简易功耗估计工具：一种叫做 Web Power Tools，另一种是 XPower 估计器。Web Power Tools 和 XPower 估计器可通过 http://china.xilinx.com/products/design_resources/power_central/获得，二者都提供了根据逻辑利用率大概估计做出的功耗估算，其区别在于：Web Power Tools 基于网站，用于早期芯片的功耗评估，支持 Virtex-Ⅱ Pro、Virtex-Ⅱ 以及 Virtex/Virtex-E 系列芯片；XPower 估计器基于电子数据表格，支持 Virtex-5、Virtex-4、Spartan-3A DSP、Spartan-3A/3AN、Spartan-3E 以及 Spartan-3 等较新系列芯片。

Xilinx 会自动更新各个系列芯片的电子数据表格，用户下载相应的电子数据表格时，可看到最后一次更新的时间。这两种简易功耗评估工具可以仅凭设计利用率估计就能获得初步的功耗评估，而无需实际设计文件，是在设计流程的早期获得器件功耗情况的最快捷和最方便的方法。此外，简易功耗评估工具不需要安装，只需要拥有互联网连接和 Web 浏览器，将电子表格下载到本地即可。

1) XPower 估计器的用户界面

每款型号所对应的 XPower 估计器的电子数据表格是不同的，下面以 Virtex-4 系列芯片的电子表格为例进行介绍。将网站上相应的电子数据表格下载到本地后，用 Office 系列的 Excel 工具打开后，其界面如图 4-63 所示。

从图 4-63 可以看出，XPower 估计器可分为不同的页面，包括 Summary 页面、CLOCK 页面、LOGIC 页面、I/O 页面、BRAM 页面、DCM 页面、PMCD 页面、DSP 页面、PPC 页面、MGT 页面、EMAC 页面、Graphic 页面以及 Release 页面。其中，主界面处输出最终结果，

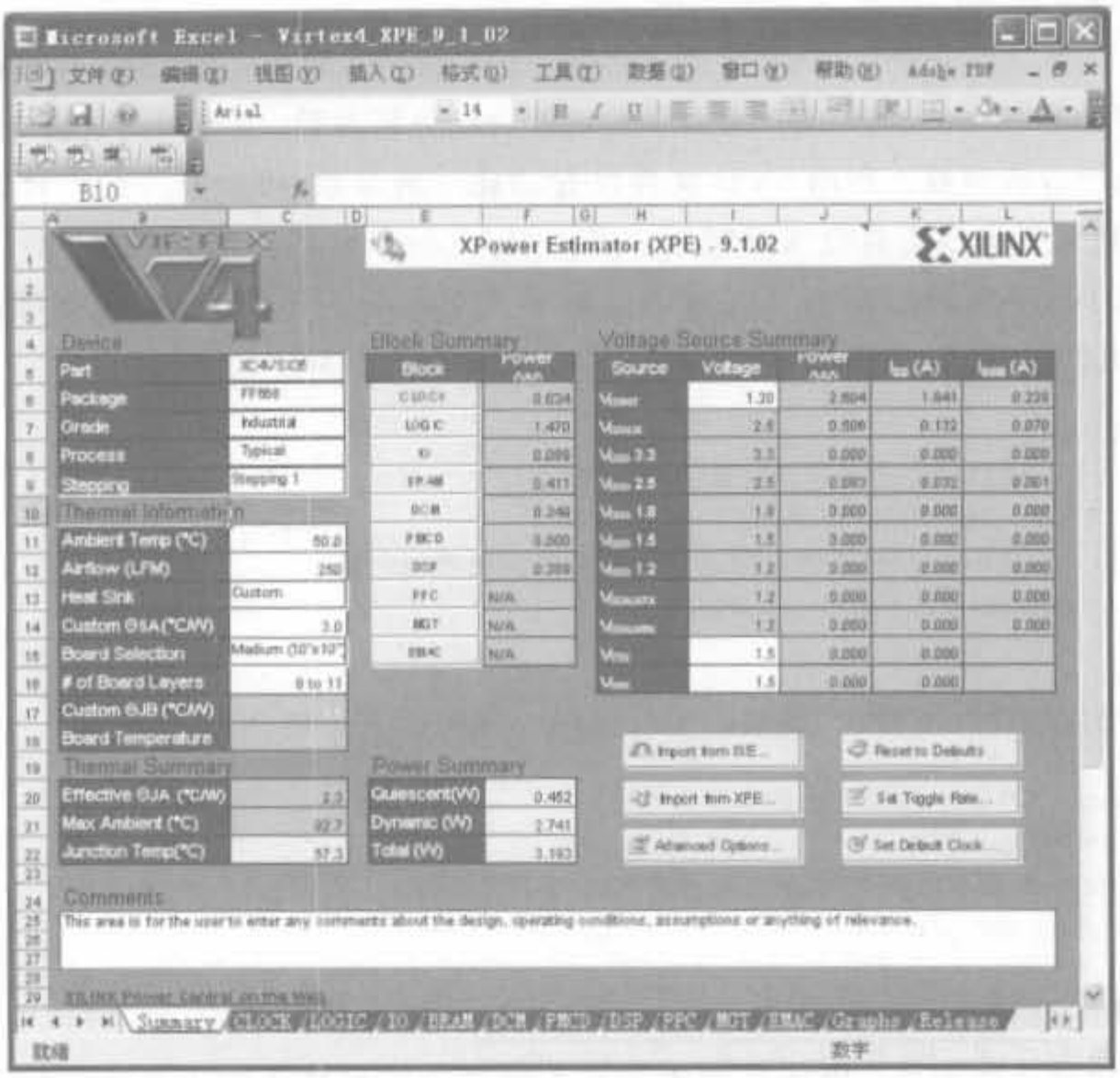


图 4-63　Virtex 系列芯片的 XPower 估计器

可分为芯片栏、热量信息栏、热量总结栏、注释栏、块总结栏、功率总结栏、电压源总结栏以及交互操作区域。表格不同的颜色表明了用户可进行的操作，详细说明如表 4-2 所示。

表 4-2　电子表格颜色说明

颜色	权限说明	补充
白色(White)	可写、可编辑	用户可编辑、输入相关参数
灰色(Grey)	只读、写保护	输出估计器的计算结果，不能编辑或输入数据
青色(Crey)	只读	用于给出功耗估计报告
黄色(Yellow)	可写、可编辑	专门用于警告提示
红色(Red)	可写、可编辑	专门用于错误提示

在 Release 页面可以查看电子表格的更新版本记录，如本书所用的电子表格版本为 9.1.02，更新时间为 2007-04-23，其相关信息如图 4-64 所示。

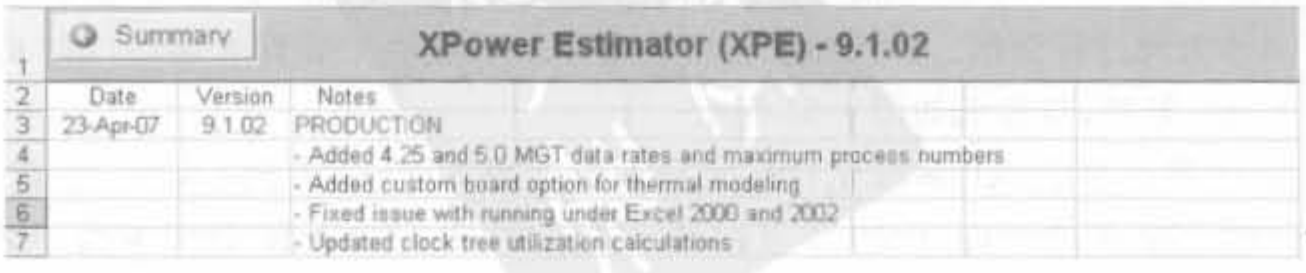


图 4-64　Virtex 系列芯片的 XPower 估计器

2) XPower 估计器的使用流程

XPower 估计器的使用方法非常简单，设计人员直接在各个分页面输入相应的资源利用率和时钟频率，然后回到 Summary 页面，就可以看到各个电压的电流值，并得到 FPGA 的工作功率和工作温度。下面以 XC4VSX35-12 芯片的设计为例，介绍 XPower 估计器的使用方法。

例 4-4 使用 XPower 估计器预算 XC4VSX35-12 应用的功耗。

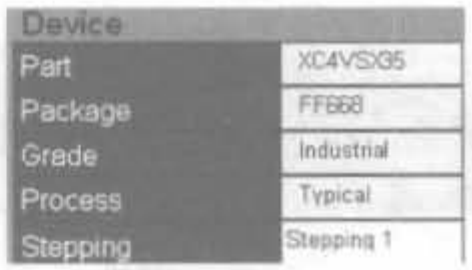


图 4-65 XPower 估计器的芯片配置界面

(1) 双击打开 XPower 估计器，在 Summary 页面的“Device”栏的“Part”选项下拉框中选择“XC4VSX35”，“Package”选项选择“FF668”，“Grade”选项选择“Industrial”，“Process”选项选择“Typical”，“Stepping”选项选择“Stepping1”，如图 4-65 所示。同时，在“Thermal Information”栏选择默认值。

(2) 由于 Virtex-4 族的 SX 系列芯片没有 PowerPC、MGT(吉比特收发器)以及 EMAC(以太网接口)控制器，所以只需要设定时钟、逻辑、I/O、BRAM、DSP 以及 DCM 即可。XPower 估计器只是粗略预算，所以在输入以上指标时，应预留 15%的裕量，即所有值按照真实值的 115%输入。在时钟栏输入所有可能用到的时钟，并给出其扇出(Fanout)大小(一般设定为 10 即可)。本例中，输入了 270MHz、120MHz 以及 60MHz 这 3 个频率的全局时钟，如图 4-66 所示。

Summary — Clock Tree Power

Name	Frequency (MHz)	Type	Fanout	Power (W)
clk_270MHz	270.0	Global	10	0.014
clk_120MHz	120.0	Global	10	0.011
clk_60MHz	60.0	Global	10	0.009

图 4-66 XPower 估计器的时钟配置界面

在逻辑页面，输入 a、b、c 3 个模块，分别工作于不同的频率，并给出了其大致的逻辑资源和平均扇出，如图 4-67 所示。

Summary — Logic Power

Module	Clock (MHz)	LUTs	Shift Registers	Select RAMs	FFs	Toggle Rate	Average Fanout	Power (W)
a	270.0	10000	10	10	10000	50.0%	3	1.386
b	120.0	500	1	2	1200	50.0%	3	0.048
c	60.0	1800	4	4	200	50.0%	3	0.036

图 4-67 XPower 估计器的逻辑配置界面

在 I/O 页面，输入各个模块的端口数(注意，这里的端口是直接引到顶层模块，和芯片管脚所对应的端口数)，选择工作电平标准以及工作频率等参数。本例的配置如图 4-68 所示。

接下来配置 BRAM 页面，需要输入 BRAM 各个端口的位宽、速率、数量以及应用类型(包括 BRAM、FIFO 以及 ECC 等类型)。本例的 BRAM 使用状况如图 4-69 所示。

Summary — I/O Power

Module	I/O Standard	Clock (MHz)	Input Pins	Output Pins	Bidir Pins	Banks	Toggle Rate	Output Enable	Output Load (pF)	Data Rate	V_{CCINT} Power (W)	V_{CCAUX} Power (W)	V_{CCO} On-chip Thermal Power (W)	V_{CCO} Supply Current(A)
a	LVCMOS 2.5V 12mA (Slow)	30.0	64	18	6	2	50.0%	100.0%	0	SDR	0.002	0.001	0.011	0.009
b	LVCMOS 2.5V 12mA (Slow)	10.0	10	10	0	1	50.0%	100.0%	0	SDR	0.000	0.000	0.001	0.001
c	LVCMOS 2.5V 12mA (Slow)	60.0	0	0	30	1	50.0%	100.0%	0	SDR	0.001	0.004	0.016	0.012

图 4-68　XPower 估计器的 I/O 配置界面

Summary — Block RAM Power

Module	BRAMs	Mode	Port A Toggle Rate	Port A Clock (MHz)	Port A Enable Rate	Port A Bit Width	Port A Write Mode	Port A Write Rate	Port B Clock (MHz)	Port B Enable Rate	Port B Bit Width	Port B Write Mode	Port B Write Rate	Power (W)
a	70	BRAM	50.0%	270.0	25.0%	18	WRITE_FIRST	50.0%	270.0	25.0%	18	WRITE_FIRST	50.0%	0.308
b	1	BRAM	50.0%	120.0	25.0%	18	WRITE_FIRST	50.0%	120.0	25.0%	18	WRITE_FIRST	50.0%	0.003
Count	10	BRAM	50.0%	60.0	25.0%	18	WRITE_FIRST	50.0%	60.0	25.0%	18	WRITE_FIRST	50.0%	0.013

图 4-69　XPower 估计器的 BRAM 配置界面

在 DCM 的配置页面中，填入所使用的 DCM 时钟的频率即可。本例用到了 270MHz、120MHz 以及 60MHz 这 3 个 DCM 输出，如图 4-70 所示。

Summary — DCM Power

Name	Clock (MHz)	Freq Mode	V_{CCINT} (W)	V_{CCAUX} (W)	$V_{CCAUXIO}$ (W)
clk_270MHz	270.0	High	0.014	0.153	0.029
clk_120MHz	120.0	Low	0.006	0.057	0.029
clk_60MHz	60.0	Low	0.003	0.029	0.029
		Subtotal	0.024	0.239	0.087
Utilization	37.5%		Total	0.349	

图 4-70　XPower 估计器的 DCM 配置界面

最后需要配置的是 DSP 模块。本例只在 a 模块中使用了 48 个 DSP 模块，工作在 270MHz，因此配置界面如图 4-71 所示。

Summary — DSP48 Power

Module	DSP48 Slices	Clock (MHz)	Toggle Rate	MREG Used?	Power (W)
a	48	270.0	50.0%	Yes	0.389
Count	48				
Utilization	25.0%			Total	0.389

图 4-71　XPower 估计器的 DSP 配置界面

配置完成后，单击 Graphic 页面，可以看到功耗随设计的逻辑功能、电压、电压过程温度以及工作环境温度的变化曲线，如图 4-72 所示。可以看到，本设计的逻辑所占功耗非常高，而大量的 BRAM 和 DSP 的功耗相对比较低。因此在设计中，应尽量使用芯片内部的硬核组件以降低功耗。

完成上述过程后，返回到 Summary 页面，可以得到所有的功耗汇总结果，以及不同电压的电流大小，为系统的电源模块设计提供大致的参考范围，如图 4-72 所示。总的功耗为 3.193W，1.2V 核电压的工作电流大致为 2.5A(1.941A+2.028A)。

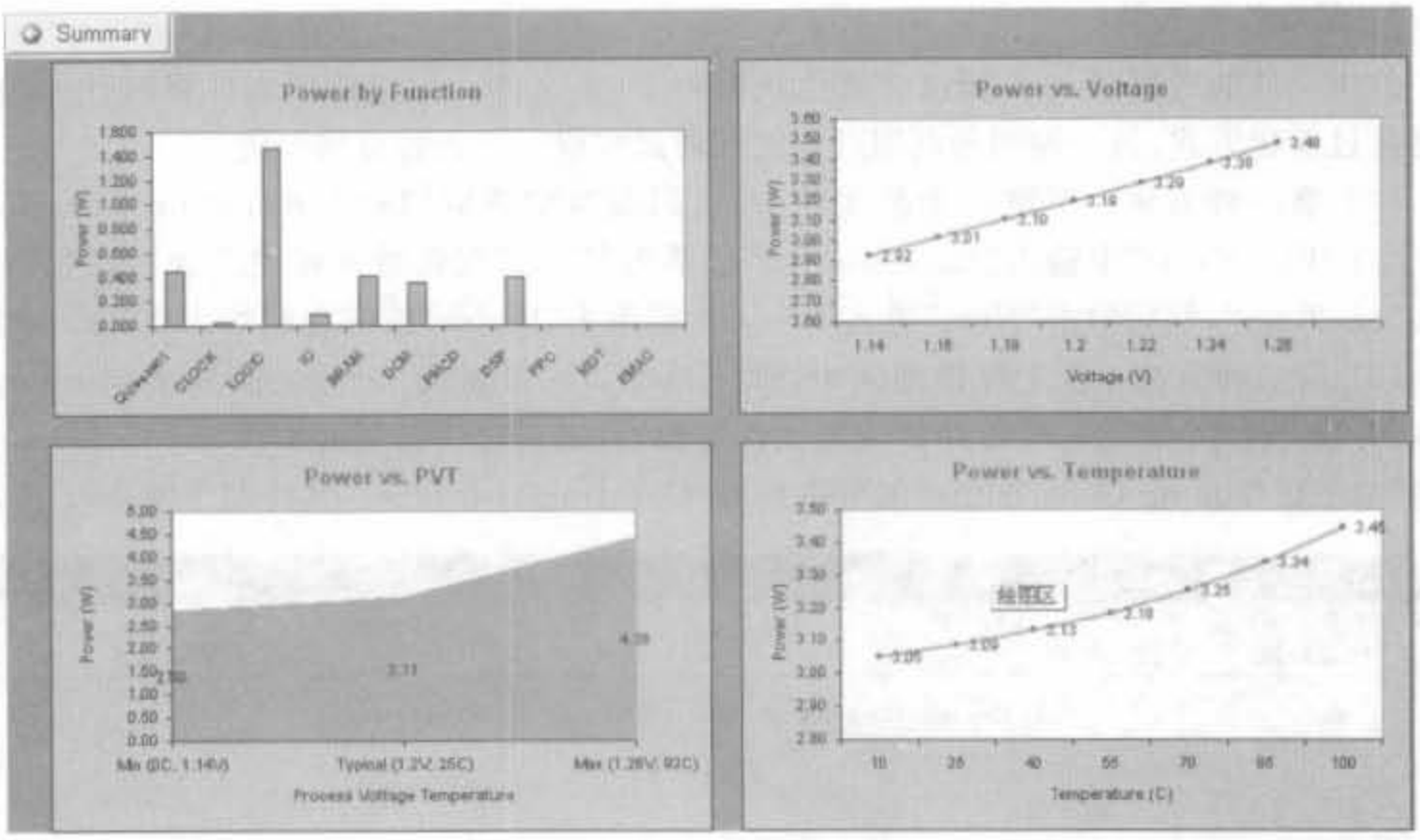


图 4-72 XPower 估计器的图形化分析结果

4.4 约束文件的编写

约束是 FPGA 设计所不可缺少的。例如，管脚约束将模块的端口和 FPGA 的管脚对应起来。在 ISE 中有多种约束，它们可以指定设计各个方面的设计要求。特别是高速电路中的时序约束，它保证了设计的可靠性，获得了设计人员的青睐。本节主要介绍约束文件的编写和基本操作。

4.4.1 约束文件的基本操作

1. 约束文件的概念

FPGA 设计中的约束文件有 3 类：用户设计文件(.UCF)、网表约束文件(.NCF)以及物理约束文件(.PCF)，可以完成时序约束、管脚约束以及区域约束。3 类约束文件的关系为：用户在设计输入阶段编写 UCF 文件，然后 UCF 文件和设计综合后生成 NCF 文件，经过实现后生成 PCF 文件。本节主要介绍 UCF 文件的使用方法。

UCF 文件是 ASCII 码文件，描述了逻辑设计的约束，可以用文本编辑器和 Xilinx 约束文件编辑器进行编辑。NCF 约束文件的语法和 UCF 文件相同，两者的区别在于：UCF 文件由用户输入，NCF 文件由综合工具自动生成。当两者发生冲突时，以 UCF 文件为准，这是因为 UCF 的优先级最高。PCF 文件可以分为两个部分：一部分是映射产生的物理约束，另一部分是用户输入的约束。同样，用户约束输入的优先级最高。一般情况下，用户约束都应在 UCF 文件中完成，不建议直接修改 NCF 文件和 PCF 文件。

2. 创建约束文件

约束文件的后缀是.ucf，所以一般也被称为 UCF 文件。创建约束文件有两种方法，一种是通过新建方式，另一种则是利用过程管理器来完成。下面将分别介绍。

(1) 第一种方法：新建一个源文件，在代码类型中选取“Implementation Constraints File”，在“File Name”中输入“one2two_ucf”。单击“Next”按钮进入模块选择对话框，选择模块“one2two”，然后单击“Next”进入下一页。再单击“Finish”按钮完成约束文件的创建。

(2) 第二种方法：在工程管理区中，将“Source for”设置为“Synthesis/Implementation”。“Constraints Editor”是一个专用的约束文件编辑器，双击过程管理区中“User Constraints”下的“Create Timing Constraints”就可以打开“Constraints Editor”，其界面如图 4-73 所示。

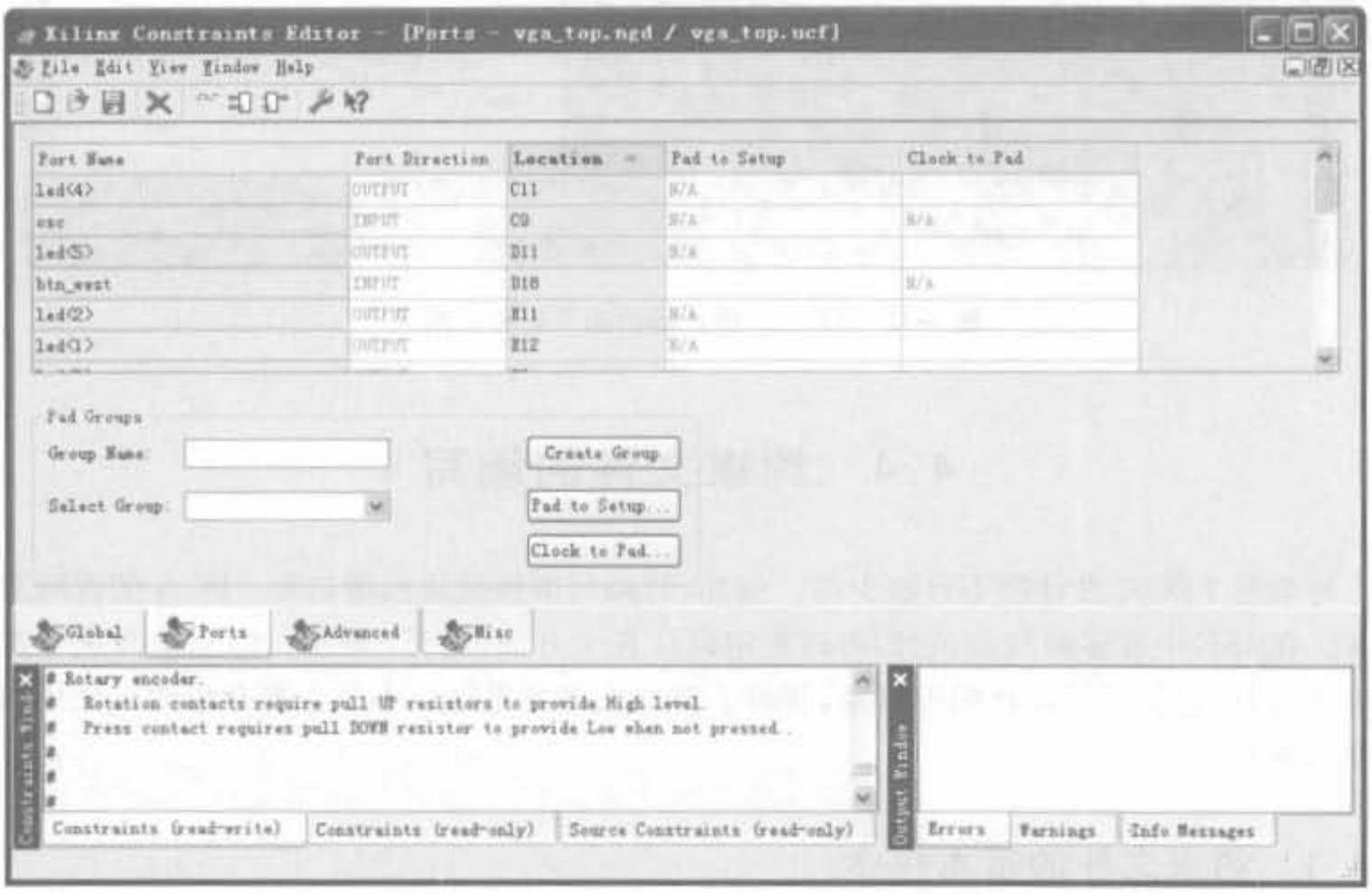


图 4-73 启动 Constrains Editor 管脚约束编辑

在“Ports”选项卡中可以看到，所有的端口都已经罗列出来了，如果要修改端口和 FPGA 管脚的对应关系，只需要在每个端口的“Location”列中填入管脚的编号即可。例如在 UCF 文件中，描述管脚分配的语法为：

```
NET "端口名称" LOC = 管脚编号;
```

需要注意的是，UCF 文件是大小写敏感的，端口名称必须和源代码中的名字一致，且端口名字不能和关键字一样。但是关键字 NET 不区分大小写。

3. 编辑约束文件

在工程管理区中，将“Source for”设置为“Synthesis/Implementation”，然后双击过程管理区中“User Constraints”下的“Edit Constraints (Text)”就可以打开约束文件编辑器，如图 4-74 所示。

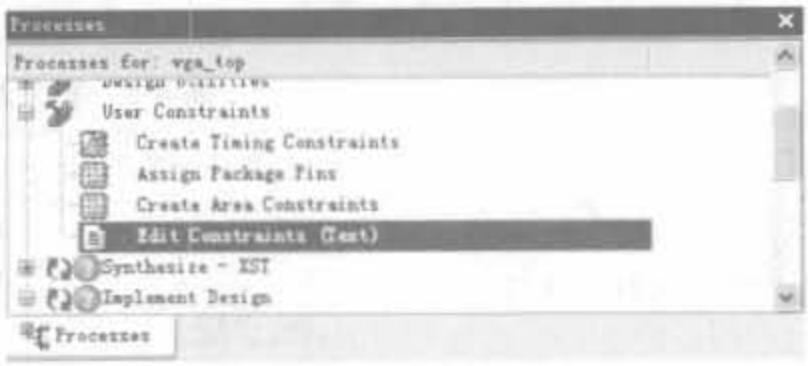


图 4-74 用户约束管理窗口

4.4.2 UCF 文件的语法说明

1. 语法

UCF 文件的语法为：

```
{NET|INST|PIN} "signal_name" Attribute;
```

其中，“signal_name”是指所约束对象的名字，包含了对象所在层次的描述；“Attribute”为约束的具体描述；语句必须以分号“;”结束。可以用“#”或“/* */”添加注释。需要注意的是，UCF 文件是大小写敏感的，信号名必须和设计中保持大小写一致，但约束的关键字可以是大写、小写甚至大小写混合。例如：

```
NET "CLK" LOC=P30;
```

“CLK”就是所约束信号名；“LOC=P30;”是约束具体的含义，将 CLK 信号分配到 FPGA 的 P30 管脚上。

对于所有的约束文件，使用与约束关键字或设计环境保留字相同的信号名会产生错误信息，除非将其用" "括起来。因此在输入约束文件时，最好用" "将所有的信号名括起来。

2. 通配符

在 UCF 文件中，通配符指的是“*”和“?”。“*”可以代表任何字符串以及空，“?”则代表一个字符。在编辑约束文件时，使用通配符可以快速选择一组信号，当然这些信号都要包含部分共有的字符串。例如：

```
NET "*CLK?" FAST;
```

将包含“CLK”字符并以一个字符结尾的所有信号，并提高了其速率。

在位置约束中，可以在行号和列号中使用通配符。例如：

```
INST "/CLK_logic/*" LOC=CLB_r*c7;
```

把 CLK_logic 层次中所有的实例放在第 7 列的 CLB 中。

3. 定义设计层次

在 UCF 文件中，通过通配符 * 可以指定信号的设计层次。其语法规则为：

```
*          遍历所有层次；
Level1/*   遍历 level1 及以下层次中的模块；
Level1/*/  遍历 level1 中的模块，但不遍历更低层的模块。
```

例 4-5 根据图 4-75 所示的结构，使用通配符遍历表 4-3 所要求的各个模块。

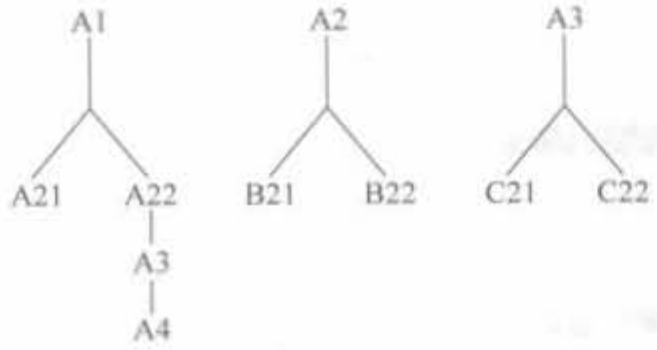


图 4-75 层次模块示意图

表 4-3 要求遍历的符号列表

要求遍历的符号	相应的约束语句
所有符号	INST * 或 INST/ *
A1、B1、C1	INST/ * /
A21、A22	INST A1/ * /
A3	INST A1/ * / * /
A3、A4	INST A1/ * / *
A22、B22、C22	INST / * / * 22/

4.4.3 管脚和区域约束语法

LOC(Location，位置)约束是 FPGA 设计中最基本的布局约束和综合约束，能够定义基本设计单元在 FPGA 芯片中的位置，可实现绝对定位、范围定位以及区域定位。此外，LOC 还能将一组基本单元约束在特定区域之中。LOC 语句既可以书写在约束文件中，也可以直接添加到设计文件中。换句话说，ISE 中的 FPGA 底层工具编辑器(FPGA Editor)、布局规划器(Floorplanner)以及管脚和区域约束编辑器的主要功能都可以通过 LOC 语句完成。

1. LOC 语句语法

基本的 LOC 语法为：

```
INST "instance_name " LOC = location;
```

其中，“location”可以是 FPGA 芯片中任一个或多个合法位置。如果为多个定位，需要用逗号“,”隔开，如下所示：

```
LOC = location1,location2,...,locationx;
```

目前，还不支持将多个逻辑置于同一位置，以及将多个逻辑置于多个位置上。需要说明的是，多位置约束并不是将设计定位到所有的位置上，而是在布局布线过程中，布局器任意挑选其中的一个作为最终的布局位置。

范围定位的语法为：

```
INST "instance_name" LOC = location; location [SOFT];
```

常用的 LOC 定位语句如表 4-4 所示。

表 4-4 常用的 LOC 定位语句

LOC 语句	说明	功能表述
INST “instance_name”LOC＝P12；	单定位语句	将 I/O 管脚 P12 分配给实例信号
INST “instance_name”LOC＝CLB_R3C5；	单定位语句	将逻辑置于坐标为 3 行 5 列的 CLB 中的任一个 Slice
INST “instance_name” LOC＝SLICE_X3Y2；	单定位语句	将逻辑置于 Slice xy 网格中的(3,2)位置上

续表

LOC 语句	说明	功能表述
INST "instance_name" LOC=TBUF_R1C2. * ;	单定位语句	将逻辑置于 1 行 2 列位置的两个 TBUF 中
INST "instance_name" LOC=MULT18×18_X0Y6;	单定位语句	将乘法器逻辑置于乘法器 xy 网格中(0,6)位置上的 MULT18×18 乘法器中
INST "instance_name" LOC=clb_r4c5. s1,clb_r4c6. * ;	多定位语句	将触发器置于 4 行 5 列 CLB 和 4 行 6 列的 CLB 中最右端的 Slice 中
INST "instance_name" LOC=SLICE_X2Y10,SLICE_X1Y10;	多定位语句	将逻辑置于 Slice xy 网格中(2,10)或(1,10)位置上的 Slice 中
INST "instance_name" LOC=SLICE_X3Y5: SLICE_X5Y20;	范围定位语句	将逻辑置于 4 行 4 列 CLB 左上角的任一 Slice 上
INST "instance_name" LOC=SLICE_X3Y5: SLICE_X5Y20;	范围定位语句	将逻辑置于 Slice xy 网格中,由(3,5)、(5,20)两点为对角线的矩形中的任一 Slice 中

使用 LOC 完成端口定义时,其语法如下:

```
NET "Top_Module_PORT" LOC = "Chip_Port";
```

其中,"Top_Module_PORT"为用户设计中顶层模块的信号端口,"Chip_Port"为 FPGA 芯片的管脚名。

LOC 语句中是存在优先级的,当同时指定 LOC 端口和其端口连线时,对其连线约束的优先级是最高的。例如,在图 4-76 中,LOC=11 的优先级高于 LOC=38。

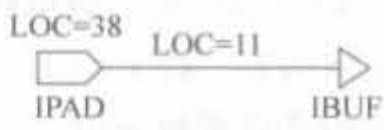


图 4-76 LOC 优先级示意图

2. LOC 属性说明

LOC 语句通过加载不同的属性可以约束管脚位置、CLB、Slice、TBUF、块 RAM、硬核乘法器、全局时钟、数字锁相环(DLL)以及 DCM 模块等资源,基本涵盖了 FPGA 芯片中所有类型的资源。由此可见,LOC 语句功能十分强大。表 4-5 列出了 LOC 的常用属性。

表 4-5 LOC 语句常用属性列表

约束类型	可用属性示例	属性含义
I/O 管脚约束	P12	将信号置于由芯片管脚号定位的端口上
	A12	将信号置于由芯片管脚阵列号定位的端口上
	B,L,T,R	将信号定位到芯片特定边界中(从物理位置上划分的上、下、左、右 4 部分)的端口上
	LB, RB, LT, RT, BR, TR, BL,TL	将信号定位到芯片特定边界上的一半(从物理位置上划分的上左、上右、下左、下右、左上、左下、右上以及右下 8 部分)位置中的端口上
	Bank0, Bank1, Bank2, Bank3, Bank4,Bank5,Bank6,Bank7	将信号置于特定管脚分组中的端口上
CLB	CLB_R4C3 (or . S0 or . S1)	指定确定位置的 CLB 来实现逻辑
	CLB_R6C8. S0 (or . S1)	指定确定位置的 CLB 中的 Slice 来实现逻辑

续表

约束类型	可用属性示例	属性含义
Slice	SLICE_X22Y3	直接通过 Slice 坐标来指定确定位置的 Slice 来实现逻辑
TBUF	TBUF_R6C7 (or .0 or .1)	指定确定位置的 TBUF 来实现逻辑
	TBUF_X6Y7	
块 RAM	RAMB4_R3C1	指定确定位置的块 RAM 来实现逻辑
	RAMB16_X2Y56	
硬核乘法器	MULT18×18_X55Y82	指定确定位置的硬核乘法器来实现逻辑
全局时钟	GCLKBUF0 (or 1,2,or 3)	指定确定位置的全局时钟缓存器来实现逻辑
	GCLKPAD0 (or 1,2,or 3)	指定确定位置的全局时钟端口来实现逻辑
DLL	DLL0P(or S) (or 1,2,or 3)	使用确定位置的 DLL 来实现逻辑
DCM	DCM_X0Y0	使用确定位置的 DCM 模块来实现逻辑

4.4.4 管脚和区域约束编辑器 PACE

ISE 中内嵌了图形化的管脚和区域约束编辑器 PACE(Pinout and Area Constraints Editor)可以将设计管脚映射到器件中,并对逻辑区块进行平面布置,方便地完成管脚约束和区域约束。在 PACE 中,可将管脚拖放到器件的显示图形上,通过容易识别的彩色编码将管脚进行逻辑分组,定义 I/O 标准和库,分配和放置差分 I/O 等。和使用约束文件相比,在中、大规模 FPGA 的开发中,使用 PACE 能大大简化管脚约束流程。

通过检查定义的 HDL 层级和核对逻辑区块与预计的门尺寸的关系,PACE 可以实现区块映射,使区块定义变得快速、准确和容易。在 HDL 编码开始之前,就可以使用 PACE 分配管脚,然后形成 HDL 模板,供开发人员编辑。可以通过标准 CSV 文件,将管脚信息导出或导入到 PCB 布局编辑器中,这大大简化了设计计划的编制。

1. PACE 用户界面

PACE 的启动方法有两种:一种是单独启动 PACE,直接单击“开始”→“程序”→“Xilinx ISE 9.1i”→“Accessories”→“PACE”即可启动;另一种是在工程经过布局布线后,在过程管理区双击“User Constraints”→“Assign Package Pins”来打开 PACE,并自动加载当前工程。需要注意的是,在启动 PACE 之前,要确保相应的设计中存在 UCF 文件,否则会提示错误。这是因为,通过 PACE 完成的操作,其最终的结果依然要写入到相应的 UCF 文件中。典型的 PACE 用户界面如图 4-77 所示。

PACE 的用户界面主要由菜单栏、工具栏、设计浏览区、设计对象列表区、芯片管脚封装视图区、芯片结构视图区以及信息显示窗口组成。

2. 使用 PACE 添加 I/O 约束

(1) 在分配管脚之前,首先需要确定芯片是否选择正确,可通过单击菜单“IOB”中的“Make Pin Compatible with..”命令来查看所选芯片型号,弹出的对话框如图 4-78 所示。如芯片型号错误,可重新选择。

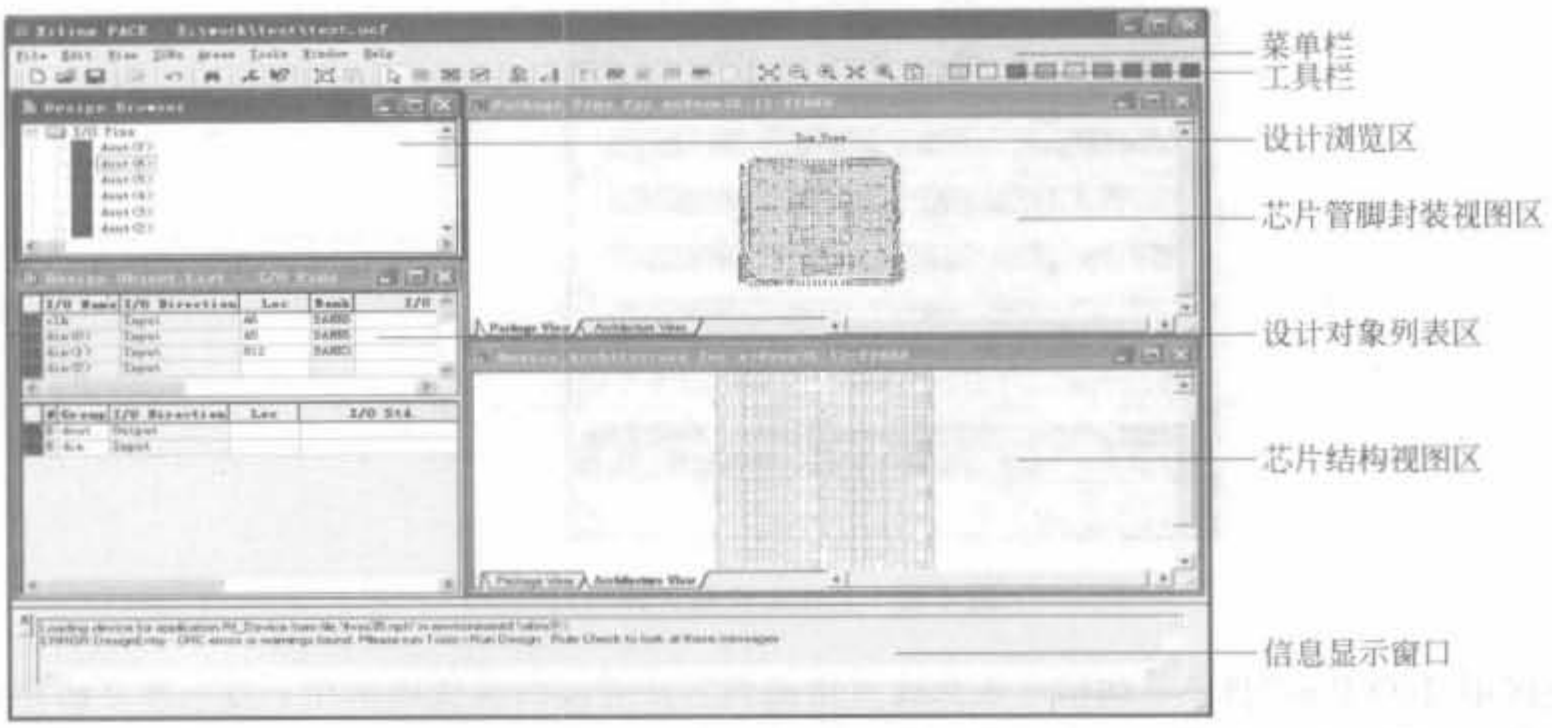


图 4-77 PACE 的用户界面

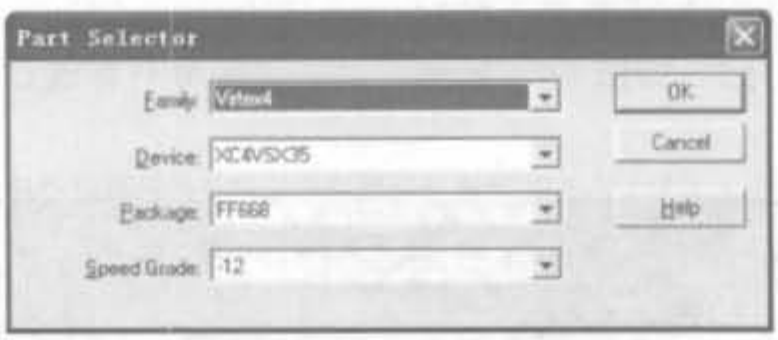


图 4-78 所选芯片的型号

（2）单击菜单“IOB”中的“Prohibit Special Pins”命令来禁止不可用的输入/输出管脚，弹出的对话框如图 4-79 所示。通过该菜单可完成所有复用管脚的控制，包括芯片配置管脚以及参考电压管脚。

图 4-79 部分输入/输出管脚的控制

（3）将信号分组或组成总线模式来加快管脚分配的速度。一般来讲，PACE 会自动将信号进行分组。此外，设计人员也可以手动添加信号分组，方法如下：在设计信号列表区按住“Ctrl”键，选取需要组合的多个信号，然后单击菜单“Edit”中的“Group”命令，即可将所选信号合并，并在信号列表区中的分组显示区显示出来，如图 4-80 所示。

对于新添加的分组信号，PACE 会以“UserGroupN”命名，其中 N 为添加的序号，用户可直接在“Group”列的对应表格中重新命名。对应信号“#”列单元格中的数字为分组或总线中的信号个数。

（4）最后是分配管脚。在 PACE 中有两种方法完成管脚分配，其一就是直接将设计浏

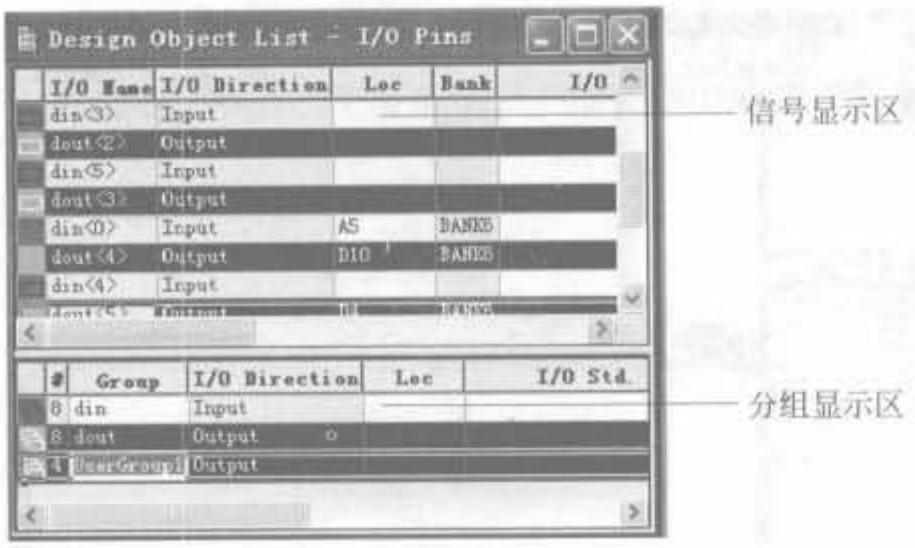


图 4-80　PACE 信号合并界面

览区中“I/O Pins”目录下的信号或总线直接拖到芯片管脚封装视图区中；另一种方法是在设计信号列表区中选中相应的信号，直接在“LOC”列所对应的表格中输入位置。分配完毕后，单击工具栏中的“保存”按钮即可。

此外，单击 PACE 中“IOBs”菜单下的“Show Differential Pairs”命令，可在芯片管脚封装视图区列出所有的差分对，如图 4-81 所示。每一对差分对都通过短线连接起来。

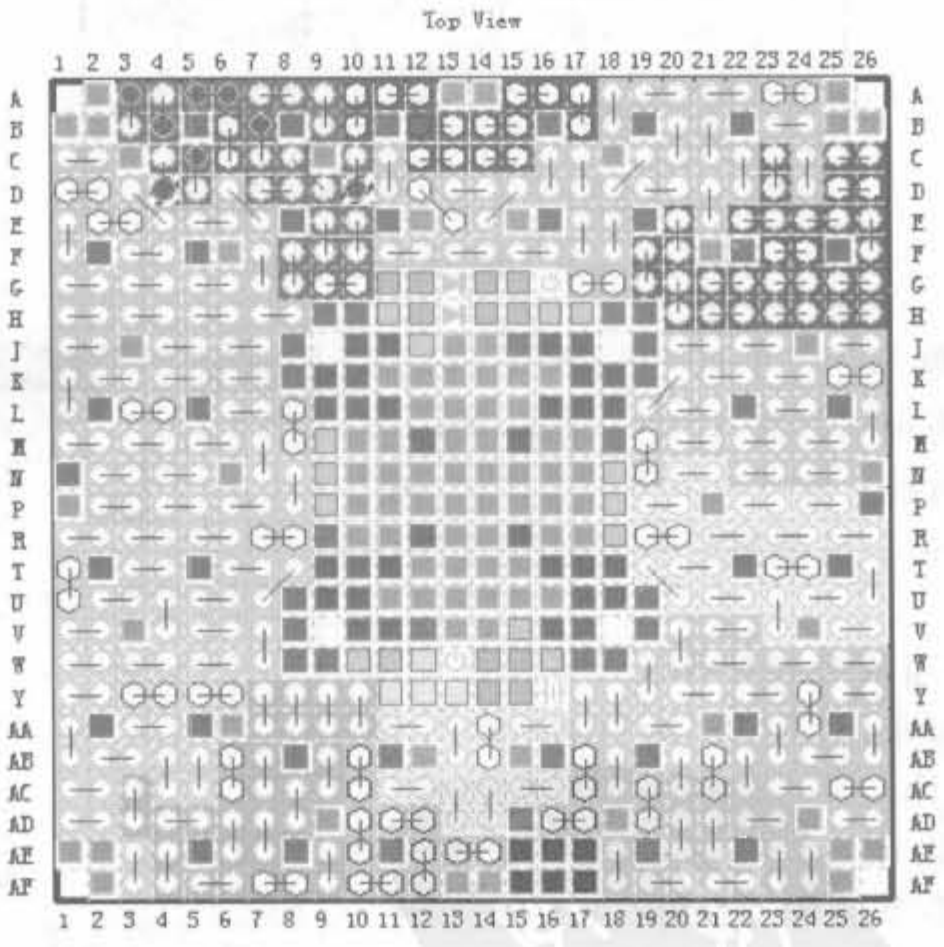


图 4-81　PACE 差分对示意图

3. 使用 PACE 添加区域约束

区域约束的主要目的是关联耦合逻辑，减少后续布线压力；其次是加大资源利用率。距离近的信号延迟不一定就小，信号线上的延迟主要来自线与线之间的转接（如 LUT、switch-box）。由于 FPGA 内部连接的结构是横、纵两向的，斜向的连接延迟会大于横、纵方向上最大跨度的连接。所以，在做位置约束时尽量避免斜向；而区域约束要松，如果没有资

源上的顾虑，约束面积建议为所需的 3 倍以上。需要注意的是，区域约束对时序的改善贡献很小，紧的约束甚至有恶化时序的可能。

通过 PACE 软件，可将设计中的所有逻辑资源，包括全局时钟缓冲器、硬核乘加器、块 RAM、硬核处理器、高速收发器以及数字时钟管理模块等放入器件架构（Device Architecture）的任何位置。下面通过实例介绍如何使用 PACE 完成区域约束。

例 4-5 使用 PACE 完成设计的区域约束。

（1）通过 ISE 打开设计工程以及其中的 PACE，在设计浏览区选中“Logic”文件夹，单击鼠标右键，选择“Object Properties”命令，将显示所用的资源，包括触发器、进位符、DCM 以及 BUFG 等。

（2）在 PACE 工具栏单击 图标（Assign Area constraint Mode），用鼠标在器件结构窗口划分出用于布局的区域，如图 4-82 所示。

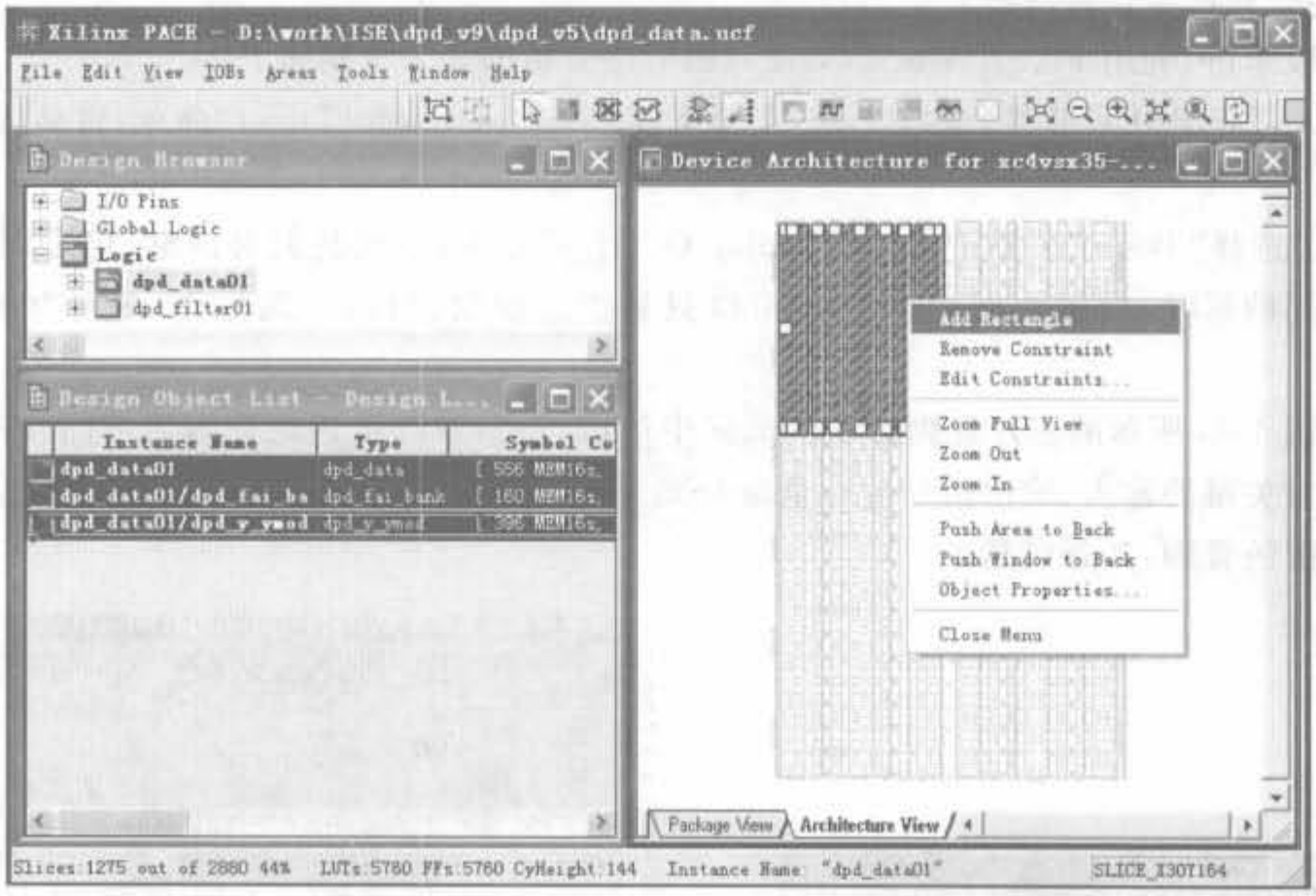


图 4-82 利用 PACE 划定约束区域

完成区域布局划分后，保存设计，用户即可在 ISE 的 UCF 文件中查阅相应的区域约束文件。附加的区域约束如图 4-83 所示。

```
#PACE: Start of PACE Area Constraints
AREA_GROUP "AG_dpd_data01" RANGE = SLICE_X0Y189:SLICE_X39Y118 ;
INST "dpd_data01/dpd_fai_bank01" AREA_GROUP = "AG_dpd_data01" ;
INST "dpd_data01/dpd_y_ymod01" AREA_GROUP = "AG_dpd_data01" ;

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

图 4-83 完成约束区域划分后所添加的约束文件

（3）对于复杂图形的区域，可通过添加多个长方形来完成。首先选中已经规划的区域，然后单击鼠标右键，添加新的长方形，依次下去，直到满足要求。如果需要修改，双击选中某

个长方形，单击鼠标右键，选择“Remove Constraint”命令，可删除相应的区域。

(4) 单击工具栏的 图标，可以在器件结构中划分出禁止布局布线的区域。单击 图标，可在禁止布局布线区域重新划定能用于布局布线的区域。

4. 使用 PACE 完成时序分析

目前，FPGA 的工作频率已达到数百兆赫以上，I/O 端口的数据速率已达到数十吉赫兹，因此在高速设计中对管脚、逻辑资源的布局显得特别重要，在低速设计中可以忽略芯片内部的布局布线延迟。PACE 可根据芯片尺寸、型号以及设计的约束，自动给出管脚和逻辑之间、逻辑和逻辑之间的信号延迟报告，该类延迟一般在 ps 数量级上，但对于数百兆赫兹的高速信号来讲，已经是非常宝贵的时隙裕量。PACE 软件会自动考虑输入、输出信号的抖动，将高速输出信号放在延时最小的管脚上。特别是对于时钟信号，会附加最优的布局和分配处理，提高同步设计性能。

一般来讲，利用 PACE 完成 FPGA 管脚时序分析的 3 个步骤如下：

(1) 打开 PACE 软件，选择“IOB”菜单下面的“Show Flight Times”命令，启动延时分析功能。

(2) 选择“Tools”菜单下面的“Display Overlay”命令，弹出延迟对话框，用不同的颜色表明不同的延时，如图 4-84 所示。该窗口只是信息输出窗口，不能操作，单击“OK”按钮即可。

在图 4-85 所示的芯片管脚封装测试区中可以看到，不同延迟的管脚用不同颜色标注，用户可对关键的输入、输出信号进行重新分配，提高设计性能。细心的读者会发现，越处于芯片外围的管脚，其延时越大。

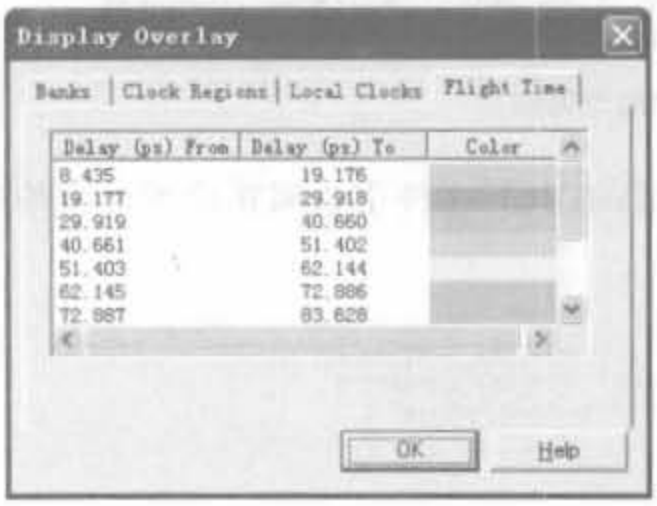


图 4-84 FPGA 管脚之间的传输延迟分类列表

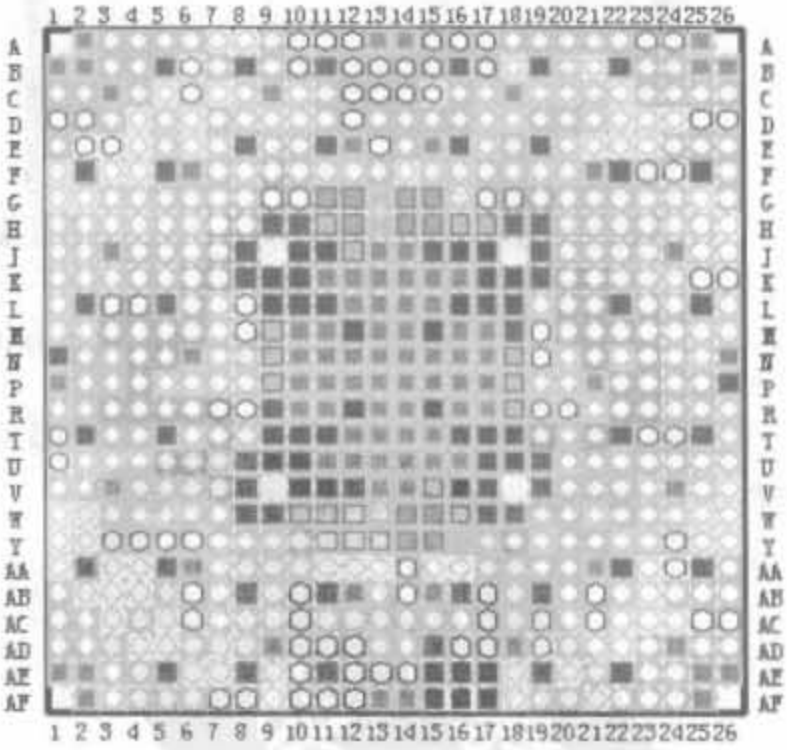


图 4-85 FPGA 管脚传输延迟分布示意图

(3) 对于多时钟设计，用户需要了解芯片的时钟分区。选择“IOB”菜单下面的“Show Clock Regions”命令，则在芯片架构视图区用不同的颜色显示 FPGA 芯片内部不同的时钟分区，如图 4-86 所示。

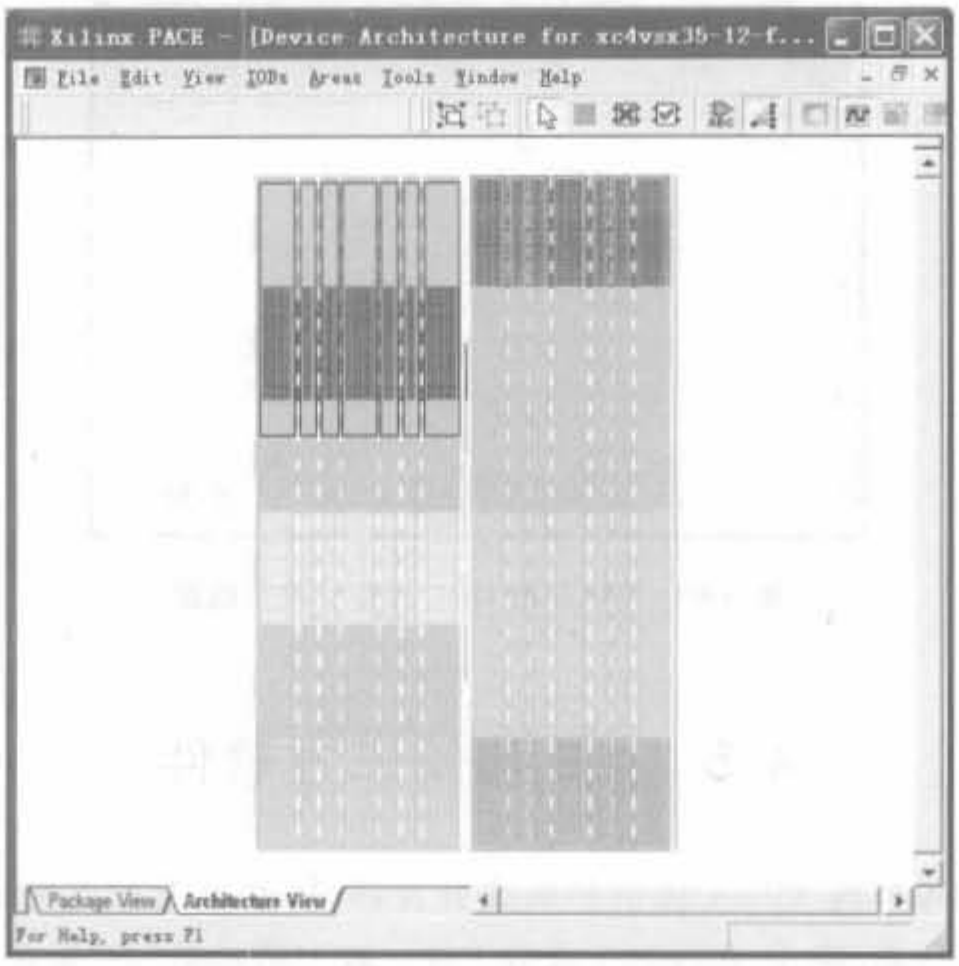


图 4-86 FPGA 时钟分区示意图

此时，设计者更关心时钟和时钟区域的对应关系，选择“Tools”菜单下面的“Display Overlay”命令，并选择“Clock Regions”页面，即可直观得到该关系，如图 4-87 所示。

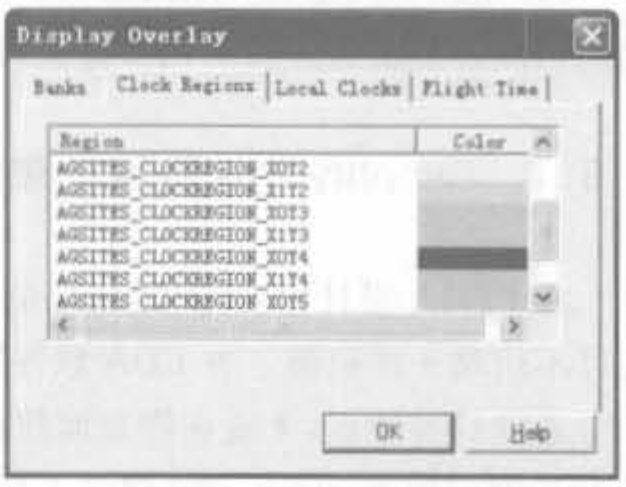


图 4-87 FPGA 时钟与时钟区域对应示意图

不同逻辑区域所允许的时钟数量是有限的，若分配到该区域的设计超过了该时钟上限，会导致设计错误。PACE 提供了时钟分析工具来帮助用户检查此类错误，通过选择“Tools”菜单下的“Clock Analysis”命令来分析。如果时钟信号超过设定值，会在其“Regions Per Clock”页面中以“*”号标出。

5. 使用 PACE 完成 DRC 分析

设计规则是电路或者芯片在版图设计中所必须遵循和满足的各种规定和要求，如果不能满足，生产出来的芯片将可能无法正常工作。DRC(Design Rule Check)即设计规则检查，就是根据设计规则所规定的各掩模图形的最小尺寸、最小间距等几何参数，对设计进行检查，找出不满足规则的偏差和错误，为用户修正设计提供依据。目前，FPGA 芯片的管脚越来越多，因此检查时钟管脚分配、I/O 端口输出电平标准与相应的 I/O 电压是否一致、核电压以及辅助电压是否正确成为任务繁重且容易出错的地方。PACE 提供的 DRC 检查可自动完成上述核查，选择“Tools”菜单下的“Run Design Rule Check(DRC)”命令，即可得到设计的 DRC 结果，如图 4-88 所示。

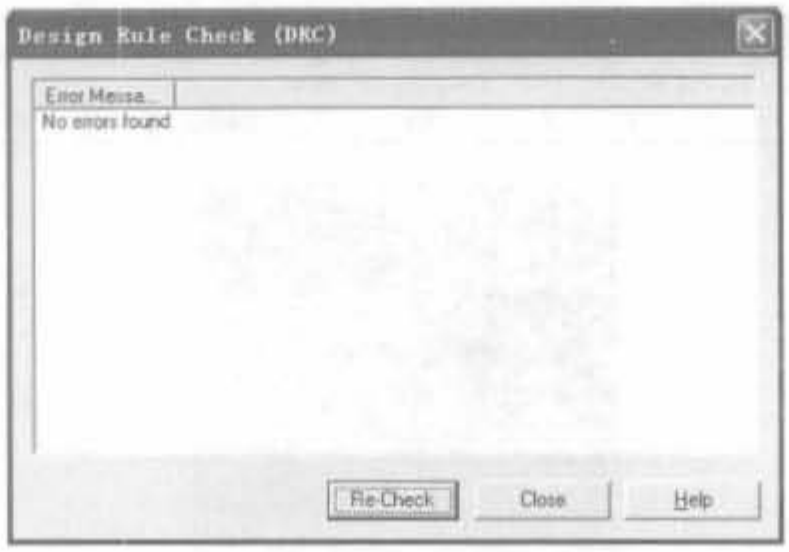


图 4-88　PACE 的 DRC 检查结果示意图

4.5　ISE 与第三方软件

ISE 作为 FPGA 厂商 Xilinx 提供的集成开发环境，基本上可以完成所有的设计输入（原理图或 HDL）、仿真、综合、布线和下载等工作。较多的初期用户采用 ISE 中这些功能对应的工具，但它们在设计仿真和逻辑综合方面不够理想，因此 ISE 提供第三方 EDA 工具的接口，让用户更方便地利用其他的第三方 EDA 工具。Synplify 和 ModelSim 一直是业界完成逻辑综合和逻辑仿真最优秀的工具，本节主要介绍这两个软件的使用。

4.5.1　Synplify Pro 软件的使用

在 FPGA 设计中，许多设计人员都习惯于使用综合工具 Synplify Pro。虽然 ISE 软件可以不依赖于任何第三方 EDA 软件完成整个设计，但 Synplify Pro 软件有综合性能高以及综合速度快等特点，无论在物理面积还是工作频率方面，都能达到较理想的效果。因此如何在 ISE 中调用 Synplify Pro 综合工具，并进行无缝的设计连接仍然是设计人员需要解决的一个设计流程问题。

1. Synplify Pro 综合软件的安装

下面介绍 Synplify Pro 的安装步骤。运行安装程序，欢迎界面过后，将出现如图 4-89 所示的安装选择界面，用户可以根据需要选择相应的组件。然后按照默认选项继续执行，即可完成安装。

在 Synplify 安装完成后，还需要安装 Identify。在“开始”→“程序”→“Synplify”菜单栏中会出现“Identify 211 Installation”，双击它即开始安装。一般来讲，可以按照默认选项继续执行，直至安装完毕。安装完成之后，需要添加授权的 License 文件，才能正常使用。

2. 关联 ISE 和 Synplify Pro

完成了 Synplify Pro 安装后，需要将其和 ISE 软件关联，才能使用 Synplify Pro 进行综合。运行 ISE 软件，在主界面中选择“Edit | Preferences”菜单项，进行“Reference”设定，如

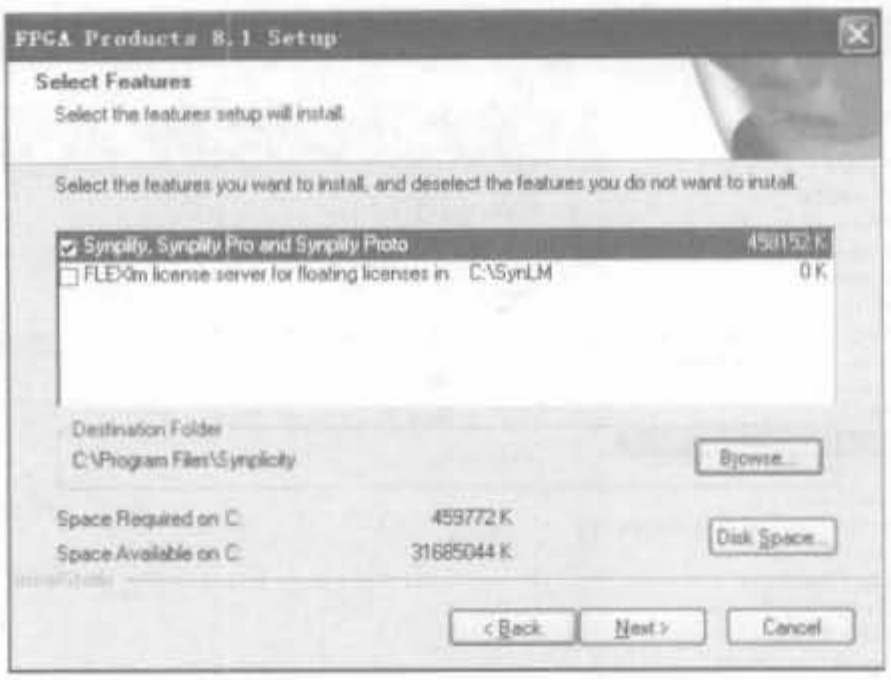


图 4-89 Synplify 的安装选择界面

图 4-90 所示。在弹出的“Preferences”对话框中选择“Integrated Tools”选项卡。该选项卡用于设定与 ISE 集成的软件的路径，第三项的 Synplify Pro 就用于设定 Synplify Pro 仿真软件的路径，如图 4-91 所示。

单击“Synplify Pro”文本框后面的按钮，会弹出一个文件选择对话框。选择“Synplify Pro”安装路径“bin”目录下的“synplify_pro. exe”文件即可。

注意，在“Integrated Tools”选项卡中还可以看到其他几个可以和 ISE 进行无缝链接的第三方软件，如 ModelSim、Synplify、LeonardoSpectrum、Chip Scope Analyzer 等。

图 4-90 选择“Preferences”菜单项

3. Synplify Pro 的使用方法简介

Synthesis 简单地说就是将 HDL 代码转化为门级网表的过程，它对电路的综合包括以下 3 个步骤：首先，HDL Compilation 把 HDL 的描述编译成已知的结构元素；其次，运用一些算法，对设计进行面积优化和减小延时。在没有目标库的情况下，Synplify 只能执行一些最基本的优化措施；最后，将设计映射到指定厂家的特定器件上，并执行一些附加的优化措施，包括根据由器件供应商提供的专用约束进行优化。工程文件以 *. prj 作为扩展名，以 tcl 的格式保留了以下信息：设计文件、约束文件、综合时开关选项的设置情况等。

1) Synplify Pro 用户界面介绍

Synplify Pro 是标准的 Windows 应用程序，所有功能均可以通过菜单选择来实现。下面按照图 4-92 中数字所标示的次序，对其界面作简要介绍。图中，①表示 Synplify 的主要工作窗口，在这个窗口中显示设计者所创建工程的详细信息，包括工程的源文件，以及综合后的各种结果文件。同时，综合完成后，每个源文件有多少错误或者警告，都会在这个窗口中显示出来。②表示 TCL 窗口，在这个窗口中，设计者可以通过 TCL 命令而不是菜单来完

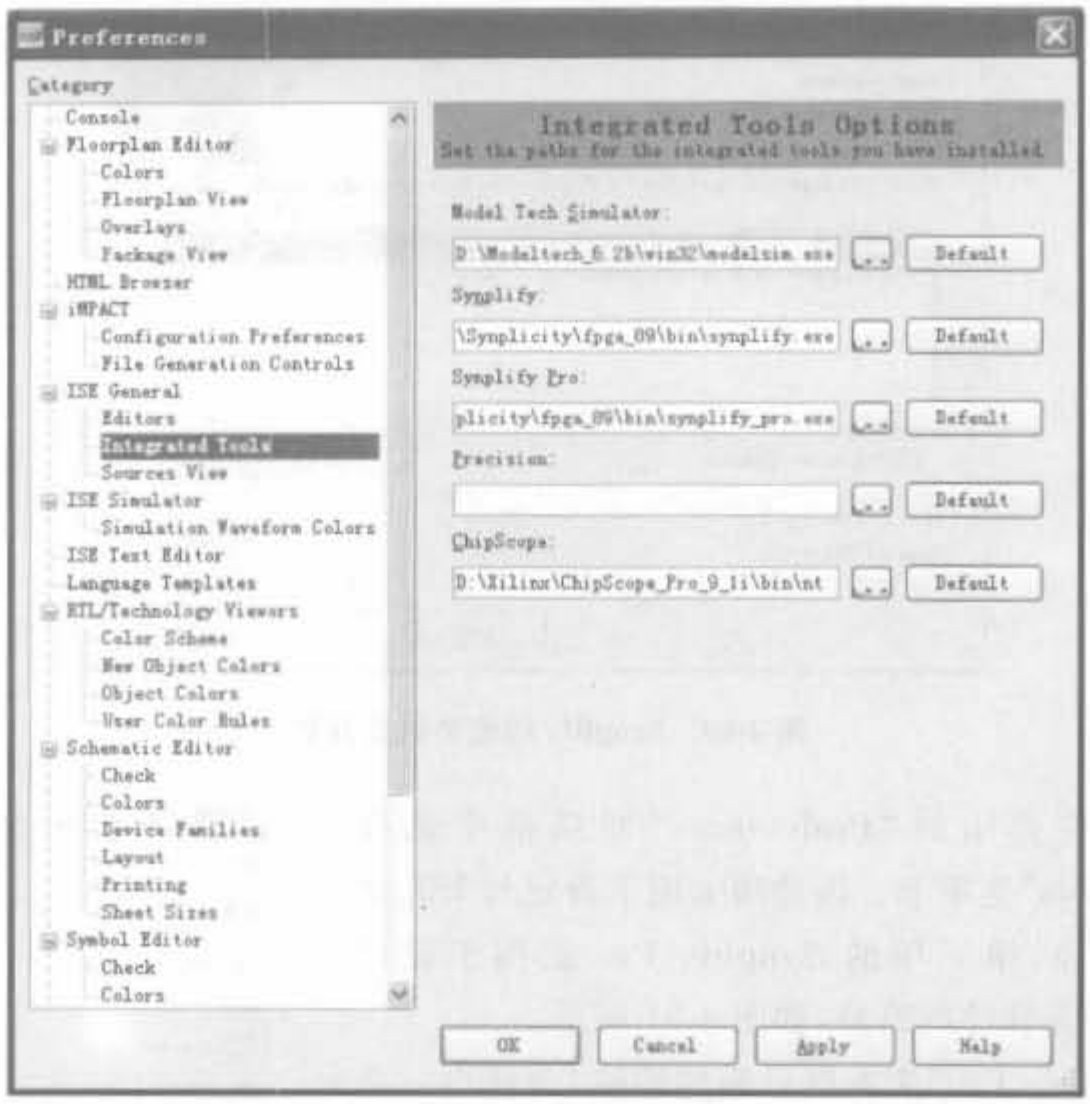


图 4-91　ISE 集成工具设定页面

成相应的功能。③表示观察窗口，在这里可以观察设计被综合后的一些特性，比如最高工作频率等。④是状态窗口，它表示现在 Synplify 所处的状态，比如下图表示 Synplify 处于闲置状态，在综合过程中会显示编译状态、映射状态等等。⑤所示的是一些复选框，可以对将要综合的设计的一些特性进行设置。Synplify 可以根据这些设置对设计进行相应的优化。⑥是运行按钮，当一个工程加入之后，单击“Run”按钮，Synplify 会对工程进行综合。⑦所示的是 Synplify 工具栏。

2）建立工程、添加源文件

建立工程首先需要打开 Synplicity Pro。单击“开始”菜单，依次选择“程序→Synplicity→Synplify Pro”，启动 Synplify Pro。在工程窗口中包含了以下内容：源文件信息、结果文件信息和目标器件信息。

默认情况下，当 Synplify 启动时，将自动建立一个新工程。这时，可以选择将工程以新名字保存。如果结束了一个工程的操作，再想新建一个工程，选择“File New”；然后选择工程文件，就可以建立一个新的工程。这项操作也可以通过工具条来执行，单击工具条的“P”图标，在弹出的对话框中选择工程文件即可。

新建工程之后，需要将源文件添加进来。单击“Add File”按钮，添加源文件和约束文件。Synplify Pro 把最后编译的“module/entity and the architecture”作为顶层设计，所以需要把顶层设计文件用鼠标左键拖拉到源文件菜单的末尾处，或者单击“Impl Options”按钮，在 Verilog 属性页中设置顶层模块的名称。

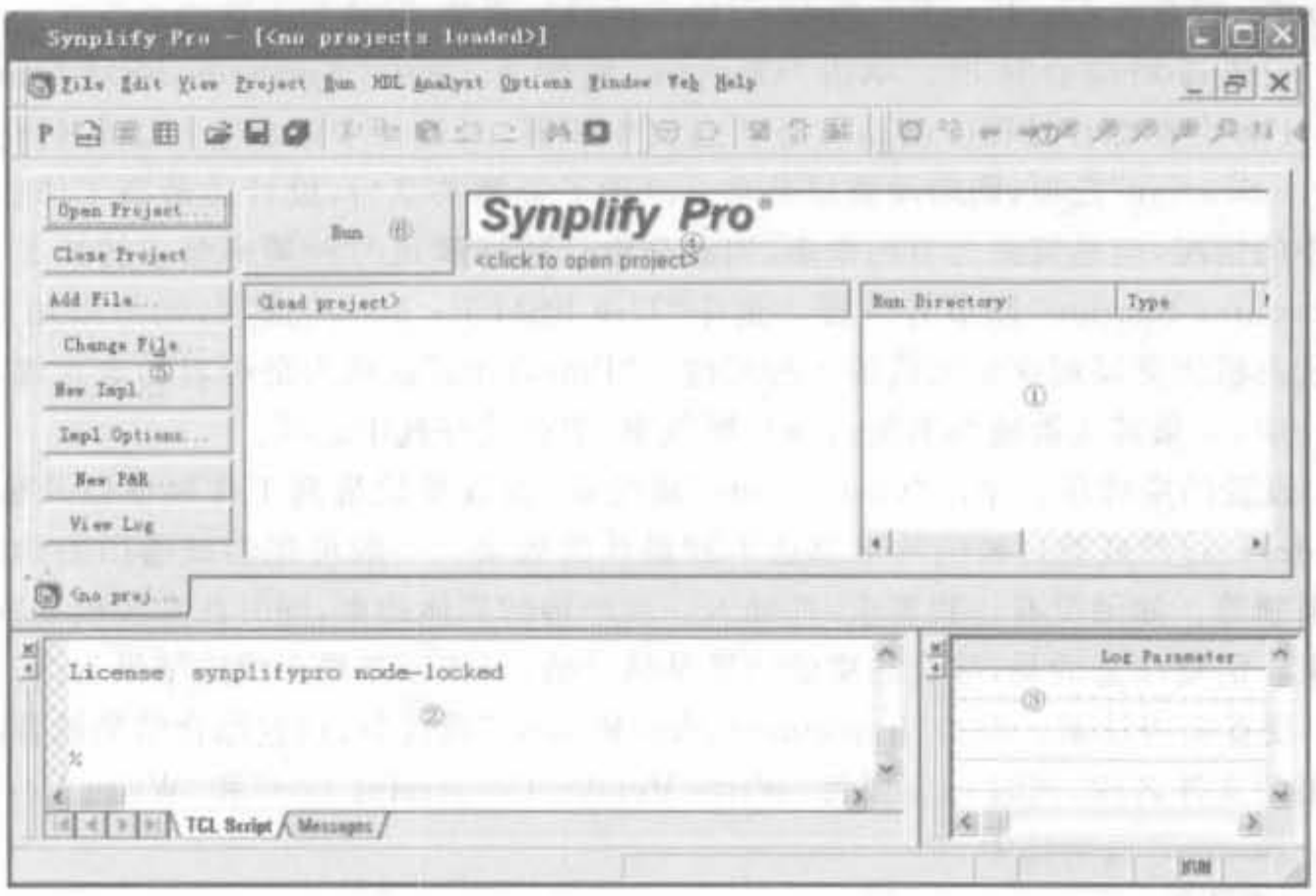


图 4-92 Synplify Pro 综合工具示意图

3）工程属性设置

添加完源文件后需要设置工程属性，单击“Impl Options”按钮，出现属性页对话框，如图 4-93 所示。下面介绍常用的芯片设置、综合选项、约束设置以及实现结果选项等参数的配置。

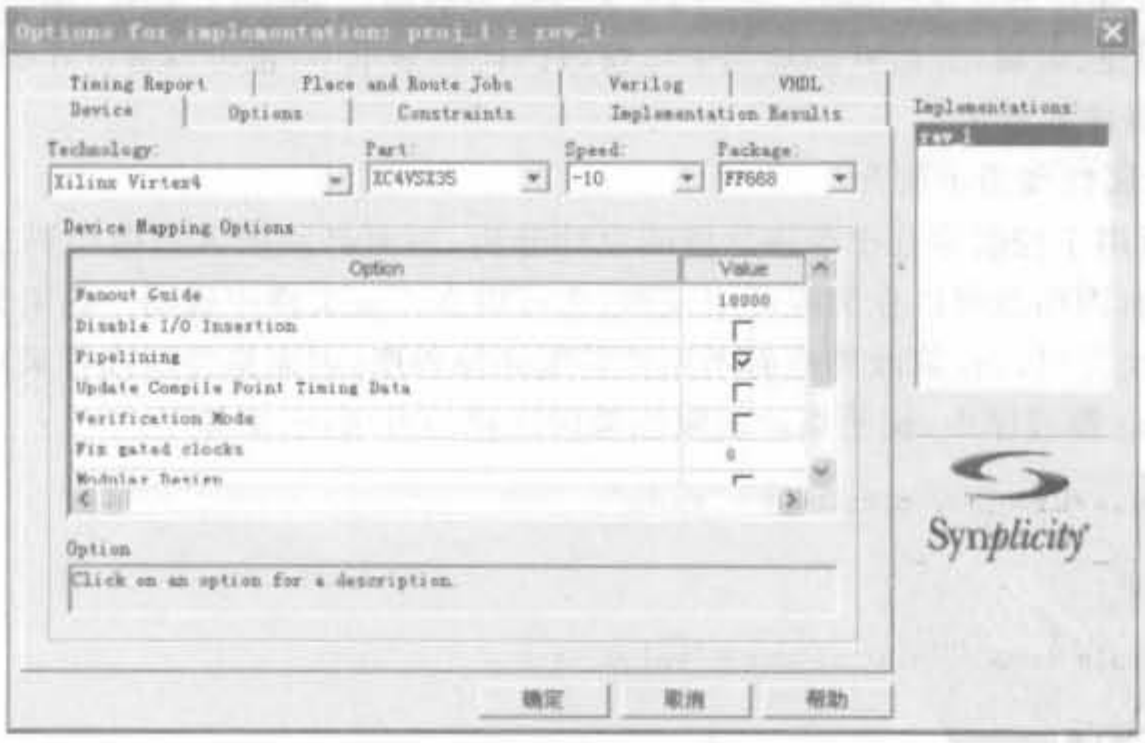


图 4-93 设置器件属性页

（1）设置 FPGA 芯片信息。打开“Device”属性页，分别设置器件厂家的器件型号、速度级别和封装信息。根据设计的速度和面积要求，可以设置最大扇出系数，默认是 10 000。根据该工程所属模块是否和片外有信号联系，选中或者不选中“Disable I/O Insertion”。如果

选中该选项，则 Synplify Pro 不会为输入/输出信号加缓冲，默认为不选。

(2) 设置通用综合选项。单击“Options”属性页，选中“Symbolic FSM Compiler”，Synplify Pro 会在综合过程中启动有限状态机编译器，对设计中的状态机进行优化。选中“Resource Sharing”选项，则启动资源共享。设置了资源共享后，设计的最高工作频率会低于不选中的情况，但是资源会节约很多，因此在设计能够满足时钟频率要求的情况下，一般选中“Resource Sharing”以节省资源。选中“Use FSM Explorer”选项，则可以用 Synplify 内置的状态机浏览器观察状态机的各种属性。“Pipelining”选项为是否启动流水线，在高速时钟设计中，如果其他措施都不能达到目标频率，那么最好选中此项。

(3) 设置约束选项。单击“Constraints”属性页，设置模块最高工作频率以及添加约束文件(.sdc)。过严或是过松的约束都达不到最佳的效果。一般可先尝试通用的约束，如时钟扇出限制等。如果没有达到要求，可加入一些严格的具体约束，同时注意放松一些可以放松的约束。需要注意的是，综合约束的结果是估计值，应该以布局布线的结果为准。

(4) 设置实现结果。单击“Implementation Results”属性页，设置综合结果放置的目录，综合结果的文件名称，同时一定要将“Write Vendor Constraint File”和“Write Verification Interface Format”选项选中。

4) 时序约束

定义时间约束是为了让综合结果满足预期的时序要求。时间约束通常分为两类：一是通用时间约束，用于目标结构的时序要求；二是黑盒时间约束，用于在设计中指定为黑盒的模块时间约束。在 Synplify Pro 中，可通过 SCOPE、约束文件以及综合属性和指示等 3 种方法添加时序。本书主要介绍利用约束文件添加约束的方法。

约束文件采用 Tcl 语言，以 *.sdc 形式保存，用来提供设计者定义的时序约束、综合属性以及 FPGA 生产商定义的属性等。约束文件既可以通过 SCOPE 创建、编辑，也可以使用正文编辑器创建、编辑，并可被添加到工程窗口的代码菜单中，也可以被 Tcl 脚本文件调用。

5) 综合属性和指示

(1) 综合属性和指示简介

综合指示用于控制综合中编译阶段的设计分析，因而必须加入到源代码中。属性是在编译后读入的，因而既可以在源程序中说明，也可以在约束文件中说明。约束文件提供了较大的灵活性，使得可以仅修改约束而不用重新编译源程序，因而是强烈推荐采用的方法。

在 Verilog 源程序中，说明指示或属性采用注释的方法，语法如下：

```
// synthesis directive|attribute = "value"
```

或

```
/* synthesis directive|attribute = "value" */
```

(2) 综合指示

综合指示用于通知 Synplify Pro 软件某些用户定制的设置，常以注释的形式出现在源代码后面，Synplify 软件会自动识别相应的说明，按照用户指令完成综合。常用的综合指示如下：

① black_box_pad_pin

声明用户定义的黑盒管脚作为外部环境可见的 I/O pad。如果有不止一个端口列在双

引号内，则以逗号分开。由于 Synplify 提供了预定义的 I/O，一般不需要这一属性。其语法如下：

```
object /* synthesis syn_black_box black_box_pad_pin = "port_list" */;
```

例如：

```
module BS(D,IN,PAD,Q) /* synthesis syn_black_box black_box_pad_pin = "PAD" */;
```

② block_box_tri_pins

声明黑盒的一个输出端口是三态，如不止一个列在双引号内，则以逗号分开。其语法如下：

```
object /* synthesis syn_black_box black_box_tri_pins = "port_list" */;
```

例如：

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q) /* synthesis syn_black_box
black_box_tri_pins = "PAD" */;
```

③ full_case

仅用于 Verilog 中的 case 语句，表明所有可能的状态都已经给出，不需要其他逻辑保持信号的值。其语法如下：

```
object /* synthesis full_case */
```

其中，object 可以是 case、casex、casez、statements 和 declaration。

④ parallel_case

仅用于 Verilog 中 case 语句，表明生成一个并行的多路选择结构而不是一个优先译码结构。其语法如下：

```
object /* synthesis parallel_case */
```

其中，object 可以是 case、casex、casez、statements 和 declaration。

⑤ syn_block_box

说明一个模块或组件为黑盒，仅利用其界面进行综合，而不管内部是否为空，也不进行优化。它一般应用于厂家原语或宏，或者是 IP 等用户定义的宏。其语法如下：

```
object /* synthesis syn_black_box */;
```

其中，object 可以是 module 和 declaration。

⑥ syn_encoding

强制选择自动机实现的方式，其可选值(value)如下：

- default：综合根据状态的数量选择编码方式。编码方式可以是 onehot gray sequential；
- onehot：采用 onehot 编码方式；
- gray：采用格雷码；
- sequential：采用自然码；
- safe：如果不能到达任一个状态时，让其回到复位态。

syn_encoding 的语法如下：

```
object /* synthesis syn_encoding="value" */;
```

其中，object 是状态寄存器定义。

⑦ syn_isclock

说明黑盒的一个输入是时钟信号。对名字为 clk、rclk、wclk 的黑盒输入信号，软件自动当作时钟，可以用这个属性说明任意输入信号为时钟信号。其语法如下：

```
object /* synthesis syn_isclock=0|1 */;
```

其中，object 是黑盒的 input port。

例如：

```
module ram4(myclk,out,opcode,a,b) /* synthesis syn_black_box */;
output [7:0] out;
input myclk /* synthesis syn_isclock=1 */;
input [2:0] opcode;
input [7:0] a,b;
/* Other coding */
```

⑧ syn_keep

保证被指定的 wire 在综合中保持不动，不会被优化掉，常用于 define_multicycle_path 或 define_false_path，用了-through 选项。如果使用了这一属性，将生成一个 keepbuf，可对其定义时间约束，且这个 Buffer 只占用一个位置，不出现在门级网表里。其语法如下：

```
object /* synthesis syn_keep=0|1 */;
```

其中，object 是 wire 或 reg 声明。

⑨ syn_noprune

用来保持一个或多个 component 的实例，而不管其输出能否完成映射。一般在没有该指示的情况下，未用输出端口的实例会从 EDIF 文件中删除。syn_noprune 可被置于约束文件中，其语法如下：

.sdc 文件中：

```
define_attribute {module|instance} syn_noprune {0|1}
```

Verilog 中：

```
object /* synthesis syn_noprune=0|1 */;
```

其中，object 可以是 module、declaration，也可以是实例。

⑩ syn_preserve

用在某些独立的寄存器上或模块中，使模块中的所有寄存器在优化时保持不动，也可用于保持某个自动机在优化时不动。其语法如下：

```
object /* synthesis syn_preserve=0|1 */;
```

其中，object 可以是寄存器定义信号，也可以是 Module。

⑪ syn_sharing

确定综合时是否对运算符进行资源共享。默认值是禁止，也可以在 Project 视窗里设置

这一选项。其语法如下：

```
object / * synthesis syn_sharing = " on|off " * /;
```

其中，object 可以是 module 定义语句。

⑫ syn_state_machine

对设计中的某组状态寄存器进行自动机优化，其语法如下：

```
object / * synthesis syn_state_machine = 0|1 * /;
```

其中，object 是该组状态寄存器。

⑬ syn_tco<n>

提供黑盒的输出延迟信息，其语法如下：

```
object / * syn_tcon = "[!]clock -> bundle = value" * /;
```

其中，bundle 是总线或标量信号的集合。

⑭ syn_tpd<n>

提供穿过黑盒的组合逻辑的传输延迟信息，其语法如下：

```
object / * syn_tpdn = "[!]clock -> bundle = value" * /;
```

其中，bundle 是总线或标量信号的集合。

⑮ syn_tristate

指定黑盒的一个输出端口为三态端口，其语法如下：

```
object / * synthesis syn_tristate = 0|1 * /;
```

其中，object 可以是黑盒的 output port。

⑯ syn_tsu<n>

说明一个黑盒的输入要求的建立时间，其语法如下：

```
object / * syn_tsun = "[!]clock -> bundle = value" * /;
```

⑰ translate_on/translate_off

用于与其他综合软件的兼容，二者经常配对使用。在这两个指示中间的所有代码将在综合时被忽略，也可以用于在源代码中插入一段仿真代码。其语法如下：

```
/ * synthesis translate_off * /
```

综合时忽略的代码

```
/ * synthesis translate_on * /
```

6）综合报告解读

综合报告主要由 3 个部分组成：编译报告、映射优化报告以及时序报告。但是该报告是冗长的，不容易快速找出用户所关心的结果。因此，Synplify 公司提供了综合报告观察窗，如图 4-92 中第③部分所示，可从综合报告文件中取出重要的信息。该窗口的使用非常简单，单击空白的参数显示栏，在下拉栏中选择要查看的项目，则会在同行的右侧显示出结果，如图 4-94 所示。

Log Parameter	dpd_v5
dpd_v5 Part	xc4vsx35ff668-12
Worst Slack	996.968
System - Slack	998.238
System - Estimated Frequency	567.5 MHz
dpd_v5 I/O primitives	194
dpd_v5 Total Luts	1028 (3%)
dpd_v5 Register bits (Non I/O)	2723 (8%)

图 4-94　Synplify 综合结果示意图

4.5.2　ModelSim 软件的使用

ModelSim 软件是一款强大的仿真软件，具有速度快、精度高和便于操作的特点，还具有代码分析能力，可以看出不同代码段消耗资源的情况。其功能侧重于编译和仿真，但不能制定编译的器件，也没有下载配置的能力，所以需要和 ISE 等软件关联。

1. ModelSim 仿真软件的安装

下面介绍 ModelSim 的安装步骤。

1)运行安装程序后，出现如图 4-95 所示的界面。如果拥有有效的 License，可以选择完全版(Full Product)安装；反之，应当选择评估版(Evaluation Edition)安装。

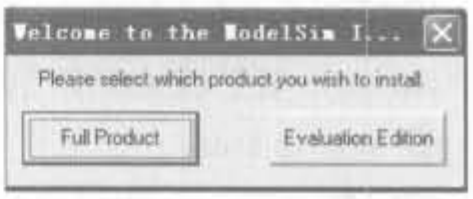


图 4-95　ModelSim 版本选择窗口

2) 选择完安装类型之后，下一个步骤就是设定安装路径，如图 4-96 所示。

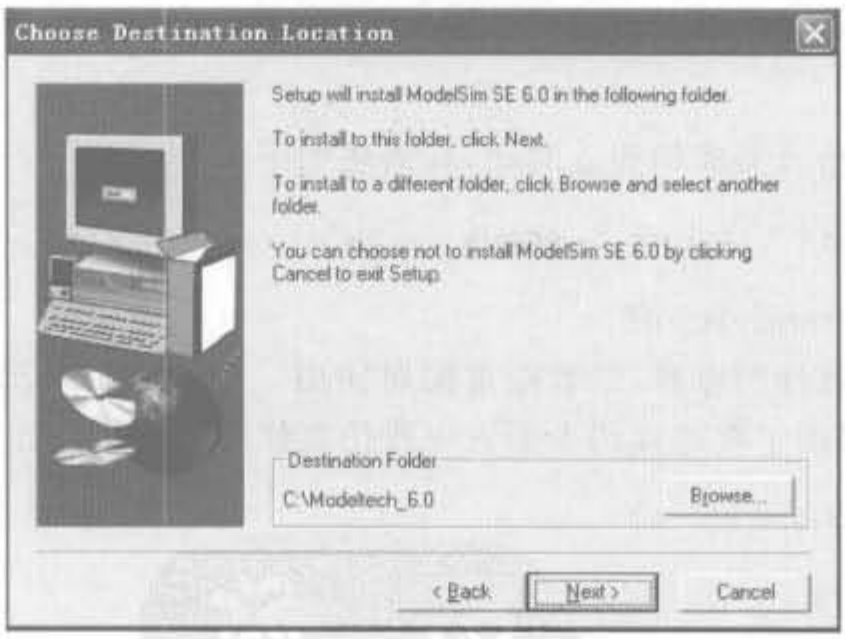


图 4-96　ModelSim 安装路径选择窗口

3) 如果选择的是完全版本，安装之后会出现“License Wizard”对话框，如图 4-97 所示。

4) 单击“Continue”按钮后，会出现 License 文件选择的对话框，用于选择有效的 License 文件。单击“OK”按钮后，系统会自动进行一系列有效性检查，只有合法的 License 文件才能使 ModelSim 正常工作。

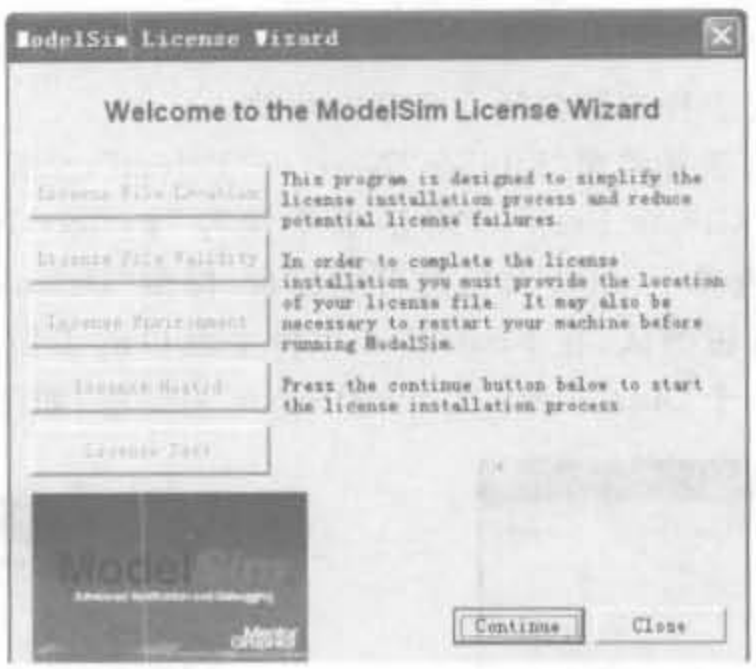


图 4-97 ModelSim 软件 License 管理向导

2. 关联 ISE 和 ModelSim

完成了 ModelSim 安装后，需要将其和 ISE 软件关联后才能使用 ModelSim 进行仿真。运行 ISE 软件，在主界面中选择“Edit | Preferences”菜单项，进行 Reference 设定，如图 4-4 所示。在弹出的“Preferences”对话框中选择“Integrated Tools”选项卡。该选项卡用于设定与 ISE 集成的软件的路径，第一项的“Model Tech Simulator”就用于设定 ModelSim 仿真软件的路径，如图 4-5 所示。单击“Model Tech Simulator”文本框后面的按钮，会弹出一个文件选择对话框，选择 ModelSim 安装路径“win32”目录下的“modelsim. exe”文件即可。

3. 在 ModelSim 中指定 Xilinx 的仿真库

ModelSim SE 版在发行时不带任何 FPGA 厂家的仿真库，因此用户必须手动编译这些库，由此面临的一个问题就是怎样建立各 FPGA 器件的仿真库。目前，各 FPGA 厂家都支持用户编译库，所以实现比较简单。

在 ModelSim 中编译 Xilinx 仿真库有很多方法，下面介绍一种比较常用的，分为 3 步来完成。

(1) 单击“开始/运行”按钮，执行下面的命令：

```
"compxlib -s mti_se -f all -l all-o f:\modeltech_6.2b\xilinx_libs -p f:\Modeltech_6.0\win32"
```

其中，“f:\modeltech_6.2b”为 ModelSim 软件的安装目录，用户可根据自己的目录来替换。等待上述命令运行完毕，其运行时间较长，用户不要中途中断。

(2) 在 Xilinx 本地库编辑成功后，在相应的目录下会自动生成 Modelsim. ini 文件，用任何一个文本编辑器将该文件中“Library”目录下(除 others 以外)的内容添加到硬盘上相应的另外的 ModelSim 安装目录下同名“modelsim. ini”文件中的相应“Library”位置。

(3) 进入 ISE 主界面，单击“Edit”下拉菜单按钮，选中下拉菜单中的“Preferences”选项，再选中“Integrated Tools”页面，重新指定 ModelSim 可执行文件即可。退出所有软件，以后再对 Xilinx 的设计进行仿真都不需要进行库的处理了。

4. ModelSim 使用方法简介

本节主要介绍 ModelSim SE 6.2b 的使用方法，包括建立工程和基本 Verilog 仿真，更多的操作方法需要用户在实际应用中熟悉并掌握。

1）建立工程

使用 ModelSim 建立工程主要包括 5 个基本步骤：

（1）启动 ModelSim，选择菜单“File”→“New”→“Poject”，会打开“Create Project”对话框，如图 4-98 所示。在“Create Project”对话框中填写“Project Name”为“test”，然后在“Project Location”栏中选择 Project 文件的存储目录，保留“Default Library Name”的设置为“work”。单击“OK”按钮确认，在 ModelSim 软件主窗口的工作区中即增加了一个空的 Project 标签，同时弹出一个“Add items to the Project”对话框，如图 4-99 所示。

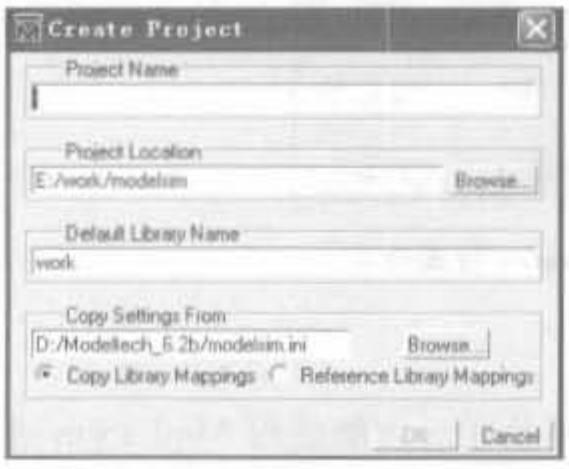


图 4-98　ModelSim 新建工程窗口

图 4-99　添加文件到工程向导示意图

（2）添加包含设计单元的文件。直接单击“Add items to the Project”对话框以后，在对话框中利用“Add Existing File”或“Create New File”选项，可以在工程中加入已经存在的文件或建立新文件。本节我们选择“Add Existing File”，弹出“Add file to Project”对话框，如图 4-100 所示。单击对话框中的“Browse”按钮，打开 ModelSim 安装路径中的“examples/tutorials /verilog/compare/”目录，选取 sm. v 和 sm. v 文件（选中多个文件时，只需要一直按住 Ctrl 键，用鼠标单击即可），再选中对话框下面的“Reference from current location”选项，然后单击“OK”按钮确认。

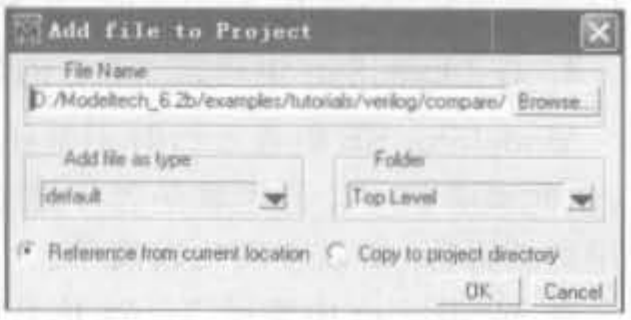


图 4-100　ModelSim 添加文件选项示意图

（3）在工作区的 Project 标签页中可以看到新加入的文件，单击鼠标右键，选取“Compile”→“Compile All”命令对加入的文件进行编译，如图 4-101 所示。

（4）两个文件编译完成后，用鼠标单击“Library”标签栏。在标签栏中用鼠标单击“work”库前面的“+”，展开 work 库，就会看到两个编译了的设计单元，如图 4-102 所示。

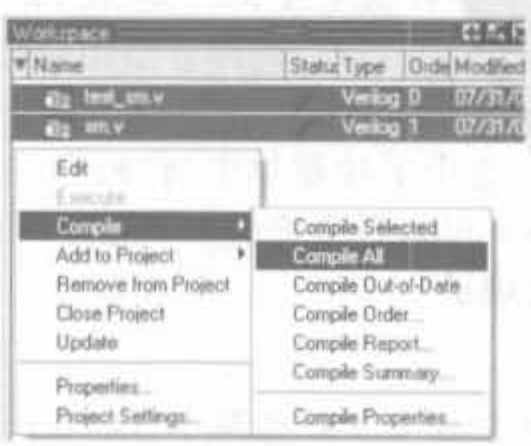


图 4-101　ModelSim 软件中的工程编译窗口

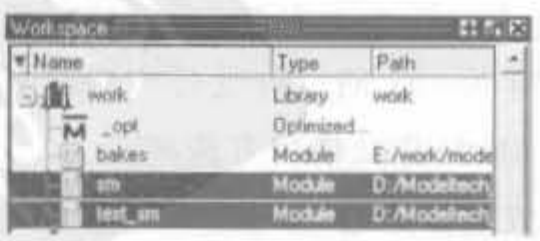


图 4-102　编译后的设计单元示意图

(5) 导入设计单元。双击“Library”标签页中的“test_sm”,在工作区中将会出现 sim 标签,并在右边的对象窗口列出了 test_sm 单元所用到的信号,如图 4-103 所示。

图 4-103 将 test_sm 模块加入工作区示意图

到此,一个工程就已经建立好了,接下来就可以开始运行仿真、分析和设计调试了。选择“File”→“Close”→“Poject”可以关闭当前目录。

2) 基本 Verilog 仿真

在准备仿真的时候,需要完成上述建立工程中的所有步骤,然后继续执行下面的步骤:

(1) 通过选择“View”→“<窗口名>”调出“signal”、“list”和“wave”窗口。也可以通过在主窗口命令行操作区的 VSIM 提示符下输入的命令来完成。

```
view signals list wave(回车)
```

(2) 向“wave”窗口添加信号。在“signal”窗口中单击鼠标右键,在弹出的菜单中选择“Add to Wave”选项中的“Signal in design”,将设计中用到的所有信号都列在“wave”窗口中,如图 4-104 所示。

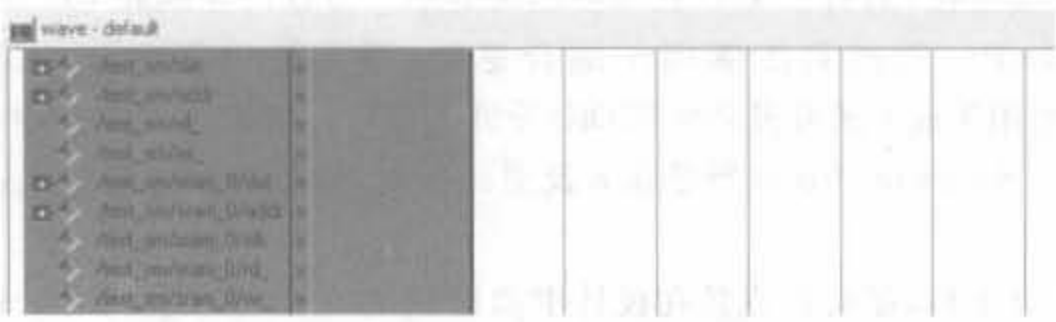


图 4-104 在 wave 窗口中添加信号

(3) 导入设计的时候,会在工作区打开一个新的 sim 标签,单击“+”展开设计层次结构,可以看到实例 test_sm、sm 等模块。单击 sim 标签中的顶层行,保证 test_sm 模块显示在“source”窗口中。

(4) 单击主窗口工具条的“Run”启动仿真,默认仿真长度为 100ns。或者在命令行输入“run”。可以在仿真途中单击“Break”中断运行,在“source”窗口中查看中断时执行的语句。

(5) 仿真完成,观察仿真波形如图 4-105 所示。确认无误后退出仿真;如果有错,则返回 source 区域修改代码。

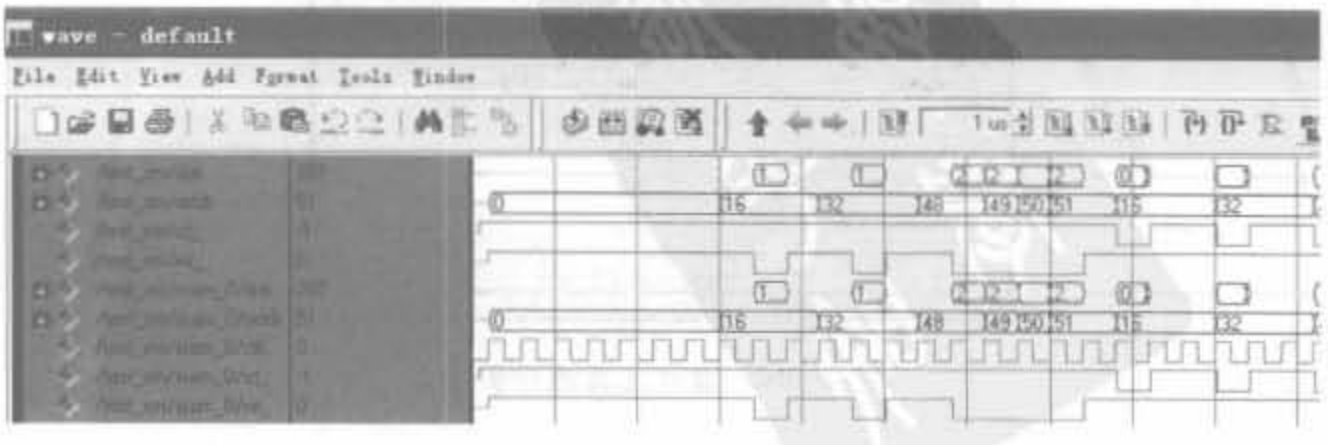


图 4-105 test_sm 模块的仿真结果示意图

4.5.3 Synplify Pro、ModelSim 和 ISE 的联合开发流程

利用 Synplify Pro、ModelSim 和 ISE 进行联合开发的步骤基本如下：当工程设计完成后，首先需要利用 ModelSim 软件完成功能仿真；然后利用 Synplify Pro 进行综合优化；再在 ISE 中完成映射与布局布线，并借助于 ModelSim 完成布局布线后的时序仿真；最后在 ISE 中完成. bit 文件的生成和 FPGA 芯片的配置。

通常，上述流程有两种实现方法：第一，将前两者作为 ISE 的第三方插件，在 ISE 中通过按键自动调用；其次，通过相应的接口文件，手动完成各个流程。自动调用比较简单，只需要在 ISE 中进行简单的设计即可。

1. 自动调用流程

完成了 Synplify Pro 和 ModelSim 的安装，并在集成工具设定页面完成与 ISE 的关联后，如图 4-91 所示，单击“ModelSim”、“Synplify /Synplify Pro”文本框后面的按钮，会弹出一个文件选择对话框，选择 ModelSim、Synplify /Synplify Pro 安装路径中“bin”目录下的“*. exe”文件即可。

需要注意的是，并不是任意的 ISE 版本和任意的 ModelSim、Synplify/Synplify Pro 版本都可以实现无缝连接，只有在 ISE 发布后出现的第三方软件才能和 ISE 无缝连接；否则，Synplify/Synplify Pro 软件只能够用于综合逻辑，而不能识别 Xilinx 提供的 IP core，ModelSim 将不能用于嵌入式开发环境(EDK)设计的仿真。根据个人操作的结果，和 ISE 9.1 匹配的 Synplify /Synplify Pro 应当是 8.8 及更高版本，相应的 ModelSim 应该为 6.1f 以后的版本。

完成了上述设定后，就可以直接在设计中调用 ModelSim 和 Synplify。在工程管理区的设计芯片上，单击鼠标右键，选择“Property”命令，即可打开用户设计的综合和仿真工具选择界面，如图 4-106 所示。在“Synthesis Tool”下拉框中选择“Synplify(Verilog)”，在“Simulator”中选择“Modelsim-SE Mixed”。

图 4-106 设计工具选择界面

2. 手动调用

由于 ModelSim 只是完成功能验证，与 ISE 没有直接的数据交互，手动操作就是分开单独操作。

手动调用 Synplify Pro 的方法比较灵活，但操作起来比较麻烦，用户可根据需要自行选择。首先单独启动 Synplify Pro 完成综合过程，再输出符合 ISE 格式的 EDIF 网表。在设置工程属性时选择 EDIF 设计流程，ISE 仅完成网表的转换、映射和布局布线等操作。同时，可在 Synplify Pro 中添加时序约束。

综合完成后，生成的后缀为.edif 的文件就是综合输出的重要文件，是实现过程的输入，直接将其导入 ISE 即可。

4.5.4 ISE 与 MATLAB 的联合使用

本节主要介绍 MATLAB 设计、ISE 实现以及二者联合测试的 FPGA 开发流程，这是全书的核心思想之一，也是目前最流行的设计方法之一。

MATLAB 软件是 MathWorks 公司的核心产品，具有用法简单、扩展性好、资源库丰富以及与其他软件接口方便的特点，它已成为电子信息和信号处理领域的科研人员必备的工具软件之一。MATLAB 和 ISE 的联合使用主要通过以下两个途径来实现：一种方法是 MATLAB 辅助 ISE，另一种方法是利用接口软件 System Generator。本书主要围绕着第一种方法展开讨论，但 System Generator 作为一种新兴的设计模式，具有强大的发展势头，本小节对其也作了简单介绍。关于 System Generator 的详细讨论，请参见第 8 章。

1. MATLAB 辅助 ISE 完成 FPGA

辅助，就是利用 MATLAB 来加速浮点算法的实现和功能测试。即在进行 FPGA 设计之前，先用 MATLAB 实现浮点算法，分析出算法的瓶颈所在，将程序的串行结构改造成并行结构；接着，利用 MATLAB 完成定点仿真，得到满足性能需求的最小定点位宽以及中间步骤计算结果的截取范围，然后在 ISE 中完成设计；最后，利用 MATLAB 的定点仿真结果对设计进行功能验证。整个流程如图 4-107 所示。

图 4-107 MATLAB 辅助 ISE 完成设计的流程图

关于怎样完成定点仿真，以及模块输出结果截取的原理和方法，可参阅有关数字信号处理的书籍。此外，利用 MATLAB 对 FPGA 设计进行功能仿真是比较关键的。下面将介绍如何将 MATLAB 和 ModelSim 结合起来使用。

首先，在 MATLAB 中产生仿真所需的输入信号，以十六进制的形式存放在数据文件中，通常放在后缀为.txt 的文本文件中；其次，在 ModelSim 中用 Verilog 编写仿真测试文件，并通过系统函数 $ readmemh 将上述仿真数据文件中的测试向量读入。在 ModelSim 中做功能仿真和时序仿真，并调用 $ fopen 函数打开另外一个数据文件，用 $ fdisplay 函数将 ModelSim 的仿真结果写入；再次，在 MATLAB 中将 ModelSim 仿真输出数据文件中的数据读入一个数组中，可以作图分析或者利用统计手段来分析。此外，还可以将 ModelSim 的

仿真输出与 MATLAB 的浮点性能作对比，来验证设计的性能。这样比利用其他方式要方便、直观，并且具有更高的正确性。

下面举例介绍 Verilog 语言中文件输入/输出函数的使用方法。

1）系统函数 $fopen 用于打开一个文件，并返回一个整数型的文件指针。然后，$fdisplay 就可以使用这个文件指针在文件中写入信息。写完后，用 $fclose 关闭文件。

其语法格式为：

```
integer <file_desc>;
<file_desc> = $fopen("<file_name>","<file_mode>");
$fwrite(<file_desc>,"<string>",variables);
$fclose(<file_desc>);
```

例如

```
integer W_file;     //定义文件指针
W_file = $fopen("W_file.txt");
$fdisplay (W_file,"@%h\n%h",a,b);
$fclose (W_file);
```

以上语句将“a”和“b”分别显示在“@%h\n%h”中的两个%h 位置，并将其写入 Write_out_file 指针所指的文件“W_file.txt”中。

2）读文件操作通过 $readmemh 和 $readmemb 来完成，分别对应的数据文件为十六进制和二进制。其语法格式为：

```
reg [<memory_width>] <reg_name> [<memory_depth>];
$readmemh ("<file_name>",<reg_name>);
```

例如：

```
reg [15:0] c [0:15];
$readmemh ("R_file.txt",c);
```

上例就是将 R_file 文件中的数据读入数组 c 中，然后就可以直接使用这些数据了。其中，数据文件的格式为：

```
@2c
45
```

@2c 表示地址，为十六进制数，45 表示该地址的数据。

2. System Generator 工具简介

System Generator 工具由 MathWorks 公司与 Xilinx 公司合作开发而成，DSP 设计人员可使用 MATLAB 和 Simulink 工具在 FPGA 内进行开发和仿真来完善 DSP 设计。新型 8.2 版本 System Generator 使 DSP 系统和算法开发商不用写 VHDL 或 Verilog 程序，只需要利用 MATLAB 及 Simulink 来开发他们的设计。一旦浮点建模完成，设计工程师采用 Xilinx 的比特及周期精确工具箱对其进行量化，并自动生成 HDL/RTL，用于 Xilinx FPGA 的网表或完整的比特流，包括新的 Virtex-5 LX 和 LXT 器件。最后，设计工程师在 Simulink 环境内采用高带宽硬件环境来验证并调试实际 FPGA 上的设计。

System Generator 的关键特性主要包括：

(1) DSP 建模。利用包含信号处理(如 FIR 滤波器、FFT)、纠错(如 Viterbi 解码器、Reed-Solomon 编码器/解码器)、算法、存储器(如 FIFO、RAM、ROM)及数字逻辑功能的 Xilinx 模块集,在 Simulink 内构建和调试高性能 DSP 系统。Xilinx 模块集提供的模块可以使用户导入 MATLAB 功能模块(如创建控制电路)及 HDL 模块(System Generator 为 Mentor Graphics 的 ModelSim 和 Xilinx ISE 仿真器提供了 HDL 协仿真接口)。

(2) Simulink 的 VHDL 或 Verilog 的自动代码生成。

(3) 硬件协同仿真。创建"FPGA 在环路(FPGA-in-the-loop)"仿真对象是代码生成选项,允许用户验证工作硬件并加速 Simulink 与 MATLAB 中的仿真。System Generator 支持以太网(10/100/吉比特)、PCI、Cardbus 及硬件平台与 Simulink 之间的 JTAG 通信。

(4) 嵌入式系统的硬件/软件协设计。为 Xilinx MicroBlaze™ 32 位 RISC 处理器构建和调试 DSP 协处理器。System Generator 提供了 HW/SW 接口的共享存储器提取功能,自动生成 DSP 协处理器、总线接口逻辑、软件驱动器及协处理器使用方面的软件技术文档。

一个典型的 System Generator 开发实例如图 4-108 所示,图中形如 Xilinx 公司标志的图标就是 System Generator,只需要双击该图标,就可以将浮点算法自动转化成 FPGA 实现。Xilinx 公司网站上提供了 System Generator 完整的使用手册和丰富的实例,读者如有兴趣,可自行阅读。

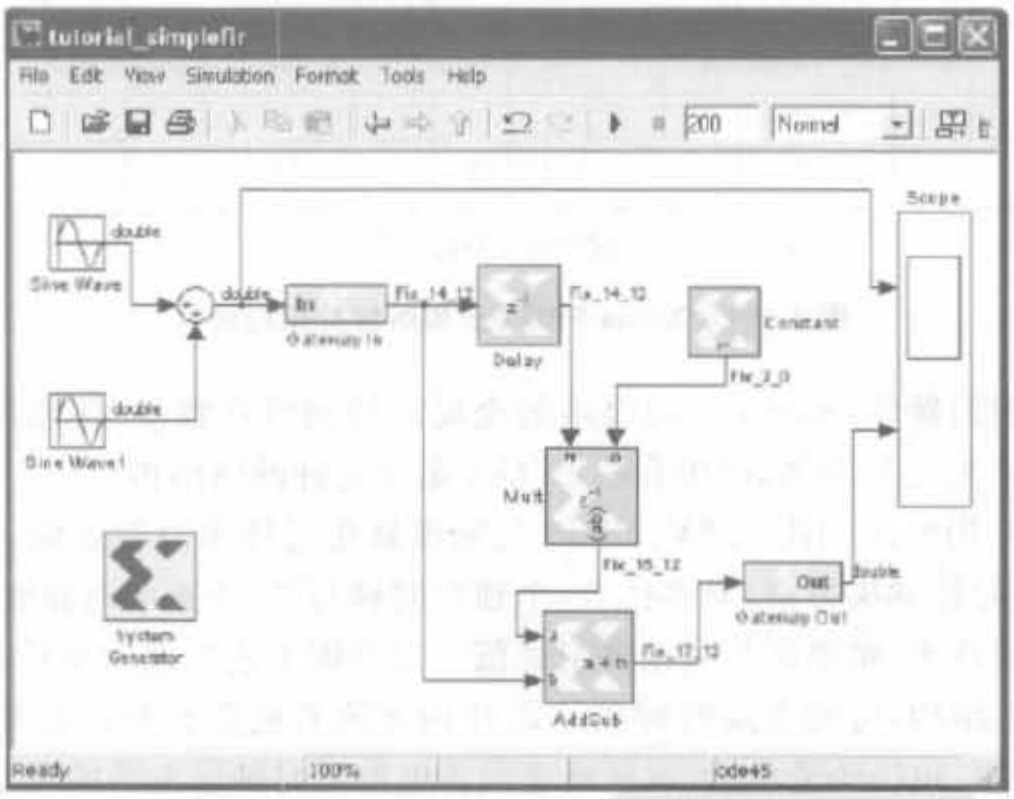


图 4-108 一个典型的 System Generator 开发实例

4.6 Xilinx FPGA 芯片底层单元的使用

Xilinx 系列 FPGA 具有丰富的底层单元,它们大都是专用硬件模块,不占用 Slice 资源,属于 FPGA 芯片中关键的硬件资源,在工程中具有重要意义。底层单元可通过第 3.4 节中介绍的硬件原语来调用,也可以通过 Core Generator 调用,二者的关系类似于 DOS 程序和 Windows 界面程序的区别。本节以 IP Core 方式为主,介绍调用 FPGA 底层单元的方法。

Xilinx 的底层单元包括全局时钟网络、DLL 模块、DCM 模块、内嵌的块存储单元、硬核乘法器、高速收发器以及嵌入式处理器等，它们都经过 ASIC 设计验证，可工作在芯片允许的最高频率。需要注意的是，不同系列芯片底层单元的属性是不同的。一般来讲，Virtex 系列的底层单元性能高于 Spartan 系列的性能。由于 Virtex-4 系列的底层单元较为全面，本节以 XC4VSX35 平台介绍 Xilinx 底层模块。

4.6.1 Xilinx 全局时钟网络的使用

在 Xilinx 系列 FPGA 产品中，全局时钟网络是一种全局布线资源，它可以保证时钟信号到达各个目标逻辑单元的延时基本相同。其时钟分配树结构如图 4-109 所示。

图 4-109 Xilinx FPGA 全局时钟分配树结构

针对不同类型的器件，Xilinx 公司提供的全局时钟网络在数量、性能等方面略有区别，下面以 Virtex-4 系列芯片为例，简单介绍 FPGA 全局时钟网络结构。

Virtex-4 系列 FPGA 利用 1.2V、90nm 三栅极氧化层技术制造而成，与前一代器件相比，它具备灵活的时钟解决方案，有多达 80 个独立时钟与 20 个数字时钟管理器。它采用了差分全局时钟控制技术，将歪斜与抖动降至最低。以全铜工艺实现的全局时钟网络，加上专用时钟缓冲与驱动结构，可使全局时钟到达芯片内部所有的逻辑可配置单元，且 I/O 单元以及块 RAM 的延时和抖动最小，可满足高速同步电路对时钟触发沿的苛刻需求。

在 FPGA 设计中，FPGA 全局时钟路径需要专用的时钟缓冲和驱动，具有最小偏移和最大扇出能力，因此最好的时钟方案是由专用的全局时钟输入管脚驱动的单个主时钟，去钟控设计项目中的每一个触发器。只要可能，应尽量在设计项目中采用全局时钟，因为对于一个设计项目来说，全局时钟是最简单和最可预测的时钟。

在软件代码中，可通过调用原语 IBUFGP 来使用全局时钟。IBUFGP 的基本用法是：

```
IBUFGP U1(.I(clk_in),.O(clk_out));
```

全局时钟网络对 FPGA 设计性能的影响很大，所以本书在第 11 章还会更深入、更全面地介绍全局时钟网络以及相关使用方法。

4.6.2 DCM 模块的使用

1. DCM 模块的组成和功能介绍

数字时钟管理模块(Digital Clock Manager,DCM)是基于 Xilinx 的其他系列器件所采用的数字延迟锁相环(Delay Locked Loop,DLL)模块。在时钟的管理与控制方面,DCM 与 DLL 相比,功能更强大,使用更灵活。DCM 的功能包括消除时钟的延时、频率的合成、时钟相位的调整等系统方面的需求。DCM 的主要优点在于:①实现零时钟偏移(Skew),消除时钟分配延迟,并实现时钟闭环控制;②时钟可以映射到 PCB 上,用于同步外部芯片,这样就减少了对外部芯片的要求,将芯片内、外的时钟控制一体化,以利于系统设计。对于 DCM 模块来说,其关键参数为输入时钟频率范围、输出时钟频率范围、输入/输出时钟允许抖动范围等。

DCM 共由 4 个部分组成,如图 4-110 所示。其中,最底层仍采用成熟的 DLL 模块;其次分别为数字频率合成器(Digital Frequency Synthesizer,DFS)、数字移相器(Digital Phase Shifter,DPS)和数字频谱扩展器(Digital Spread Spectrum,DSS)。不同芯片模块的 DCM 输入频率范围是不同的。例如,对于 Virtex -4SX 系列芯片,低输入模式的范围为 1～210MHz,高输入模式的范围为 50～350MHz;而 Spartan 3E 系列低、高两种模式的范围都只能是 0.2～333MHz。下面对 DCM 的 4 个部分分别进行介绍。

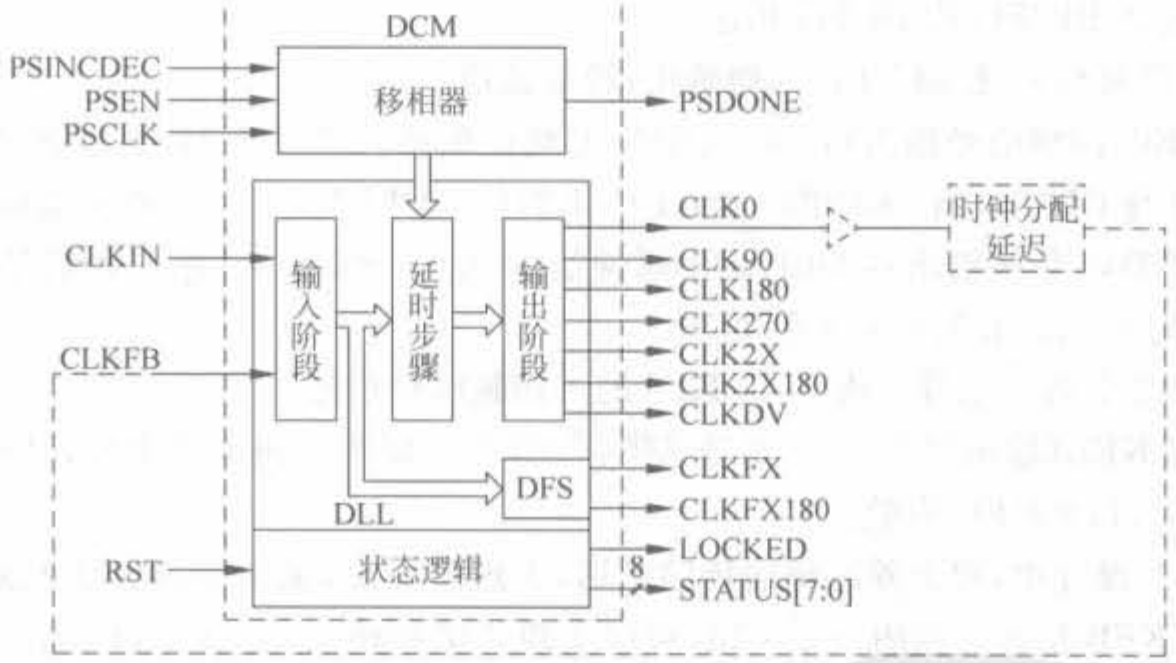


图 4-110 DCM 功能块和相应的信号

1) DLL 模块

DLL 主要由一个延时线和控制逻辑组成。延时线对时钟输入端 CLKIN 产生一个延时,时钟分布网线将该时钟分配到器件内的各个寄存器和时钟反馈端 CLKFB;控制逻辑在反馈时钟到达时采样输入时钟,以调整二者之间的偏差,实现输入和输出的零延时,如图 4-111 所示。具体工作原理是:控制逻辑在比较输入时钟和反馈时钟的偏差后,调整延时线参数。在输入时钟后不停地插入延时,直到输入时钟和反馈时钟的上升沿同步,锁定环路进入"锁定"状态。只要输入时钟不发生变化,输入时钟和反馈时钟就保持同步。DLL 可以被用来

实现一些电路以完善和简化系统级设计，如提供零传播延迟、低时钟相位差和高级时钟区域控制等。

图 4-111　DLL 简单模型示意图

在 Xilinx 芯片中，典型的 DLL 标准原型如图 4-112 所示，其管脚分别说明如下：

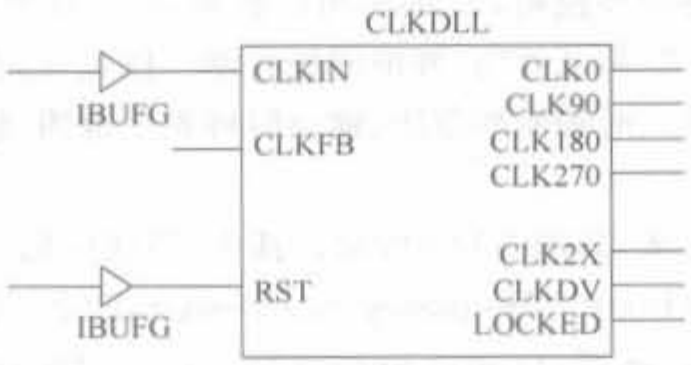


图 4-112　Xilinx DLL 的典型模型示意图

(1) CLKIN(源时钟输入)：DLL 输入时钟信号，通常来自 IBUFG 或 BUFG。

(2) CLKFB(反馈时钟输入)：DLL 时钟反馈信号，该反馈信号必须源自 CLK0 或 CLK2X，并通过 IBUFG 或 BUFG 相连。

(3) RST(复位)：控制 DLL 的初始化，通常接地。

(4) CLK0(同频信号输出)：与 CLKIN 无相位偏移；CLK90 与 CLKIN 有 90°相位偏移；CLK180 与 CLKIN 有 180°相位偏移；CLK270 与 CLKIN 有 270°相位偏移。

(5) CLKDV(分频输出)：DLL 输出时钟信号，是 CLKIN 的分频时钟信号。DLL 支持的分频系数为 1.5、2、2.5、3、4、5、8 和 16。

(6) CLK2X(两倍信号输出)：CLKIN 的 2 倍频时钟信号。

(7) LOCKED(输出锁存)：为了完成锁存，DLL 可能要检测上千个时钟周期。当 DLL 完成锁存之后，LOCKED 有效。

在 FPGA 设计中，要消除时钟的传输延迟，实现高扇出，最简单的方法就是用 DLL，把 CLK0 与 CLKFB 相连。利用一个 DLL 可以实现 2 倍频输出，如图 4-113 所示。利用两个 DLL 就可以实现 4 倍频输出，如图 4-114 所示。

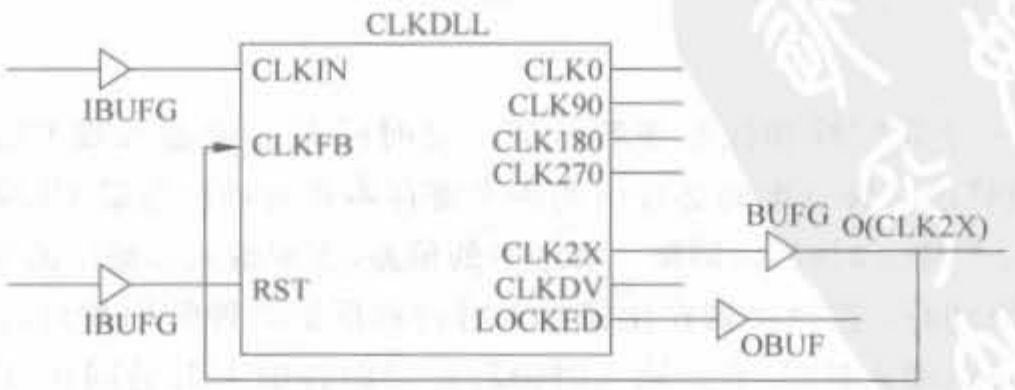


图 4-113　Xilinx DLL 2 倍频典型模型示意图

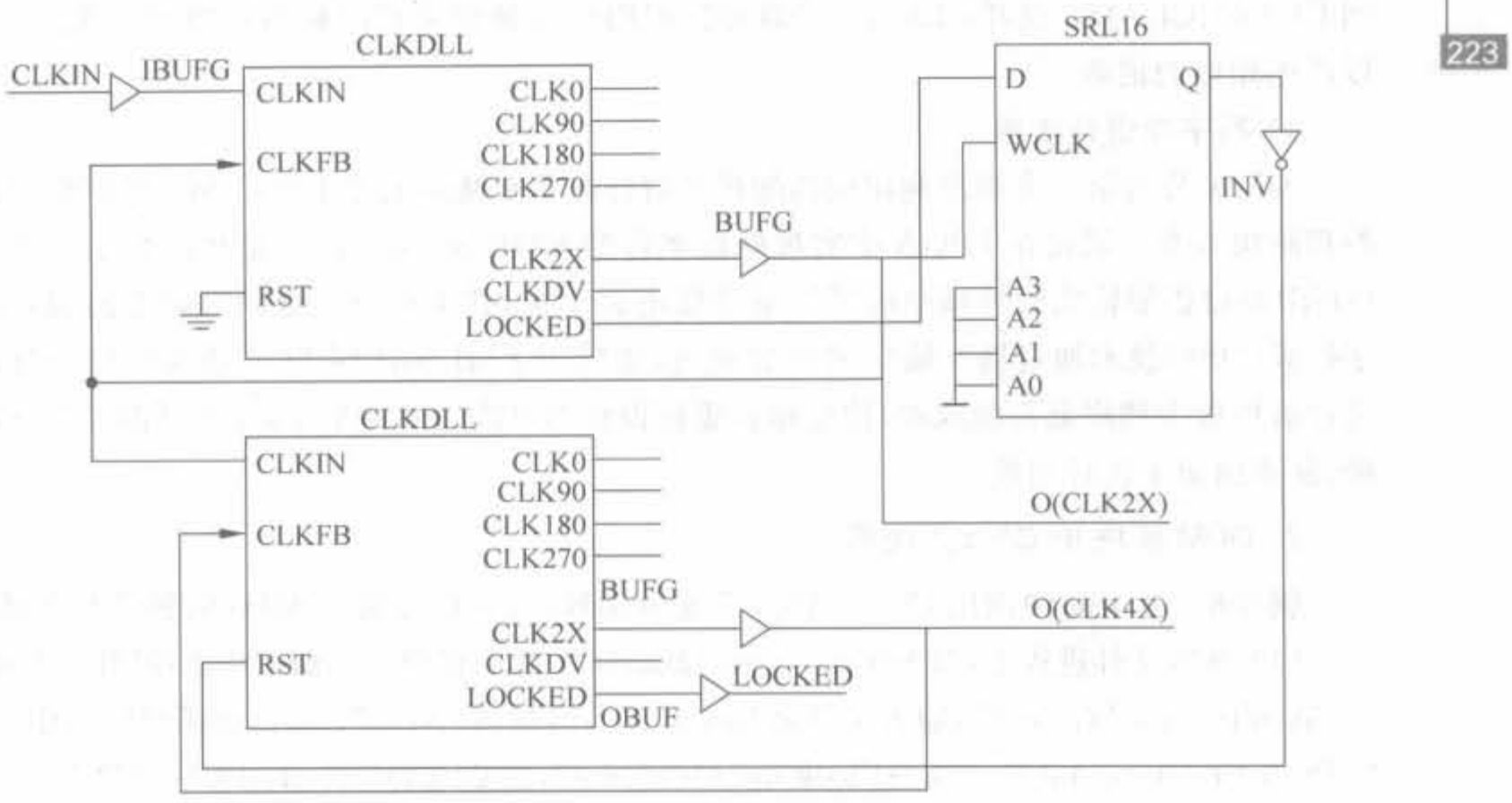


图 4-114 Xilinx DLL 4 倍频典型模型示意图

2）数字频率合成器

DFS 可以为系统产生丰富的频率合成时钟信号，输出信号为 CLKFB 和 CLKFX180，可提供输入时钟频率分数倍或整数倍的时钟输出频率方案，输出频率范围为 1.5～320MHz（不同芯片的输出频率范围是不同的）。这些频率基于用户自定义的两个整数比值，一个是乘因子(CLKFX_ MULTIPLY)，另外一个是除因子(CLKFX_ DIVIDE)，输入频率和输出频率之间的关系为

$$F_{CLKFX} = F_{CLKIN} \times \frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE}$$

比如，取 CLKFX_MULTIPLY＝3，CLKFX_DIVIDE＝1，PCB 上的源时钟为 100MHz，通过 DCM 3 倍频后，就能驱动时钟频率在 300MHz 的 FPGA，从而减少了板上的时钟路径，简化板子的设计，提供更好的信号完整性。

3）数字移相器

DCM 具有移动时钟信号相位的能力，因此能够调整 I/O 信号的建立和保持时间，能支持对其输出时钟进行 0°、90°、180°、270°的相移粗调和相移细调。其中，相移细调对相位的控制可以达到 1%输入时钟周期的精度（或者 50ps），并且具有补偿电压和温度漂移的动态相位调节能力。对于 DCM 输出时钟的相位调整，需要通过属性控制 PHASE_SHIFT 来设置。PS 设置范围为－255～＋255。比如，输入时钟为 200MHz，需要将输出时钟调整＋0.9ns 的话，PS＝(0.9ns/5ns)×256＝46。如果 PHASE_ SHIFT 值是一个负数，则表示时钟输出应该相对于 CLKIN 向后进行相位移动；如果 PHASE_SHIFT 是一个正值，则表示时钟输出应该相对于 CLKIN 向前进行相位移动。

移相用法的原理图与倍频用法的原理图很相似，只用把 CLK2X 输出端的输出缓存移

到 CLK90、CLK180 或者 CLK270 端即可。利用原时钟和移相时钟与计数器相配合，也可以产生相应的倍频。

4）数字频谱合成器

Xilinx 公司第一个提出利用创新的扩频时钟技术来减少电磁干扰(EMI)噪声辐射的可编程解决方案。最先在 FPGA 中实现电磁兼容的 EMIControl 技术，是利用数字扩频技术(DSS)通过扩展输出时钟频率的频谱来降低电磁干扰，减少用户在电磁屏蔽上的投资。数字扩频(DSS)技术通过展宽输出时钟的频谱，来减少 EMI 和达到 FCC 要求。这一特点使设计者可极大地降低系统成本，使电路板重新设计的可能性降到最小，并不再需要昂贵的屏蔽，从而缩短了设计周期。

2. DCM 模块 IP Core 的使用

例 4-6 在 ISE 中调用 DCM 模块，完成 50MHz 时钟信号到 75MHz 时钟信号的转换。

(1) 在源文件进程中，双击“Create New Source”；然后在源文件窗口中选择“IP (CoreGen & Architecture Wizard)”，输入文件名“my_dcm”，再单击“Next”；在选择类型窗口中，选择“FPGA Features and Design→Clocking→Virtex-4”，然后选择“Single DCM ADV v9.1i”，如图 4-115 所示。

图 4-115 新建 DCM 模块 IP Core 向导示意图

(2) 单击“Next”→“Finish”进入 Xilinx 时钟向导的建立窗口，如图 4-116 所示。ISE 默认选中 CLK0 和 LOCKED 这两个信号，用户根据需要添加输出时钟。在“Input Clock Frequency”输入栏中输入时钟的频率或周期，单位分别是 MHz 和 ns，其余配置保留默认值。为了演示，这里添加了 CLKFX 信号，并设定输入时钟为单端信号，频率为 50MHz，其余选项保持默认值。

(3) 单击“Next”，进入时钟缓存窗口，如图 4-117 所示。默认配置为 DCM 输出添加全局时钟缓存，以保证良好的时钟特性。如果设计全局时钟资源，用户亦可选择“Customize buffers”自行编辑输出缓存。一般选择默认配置即可。

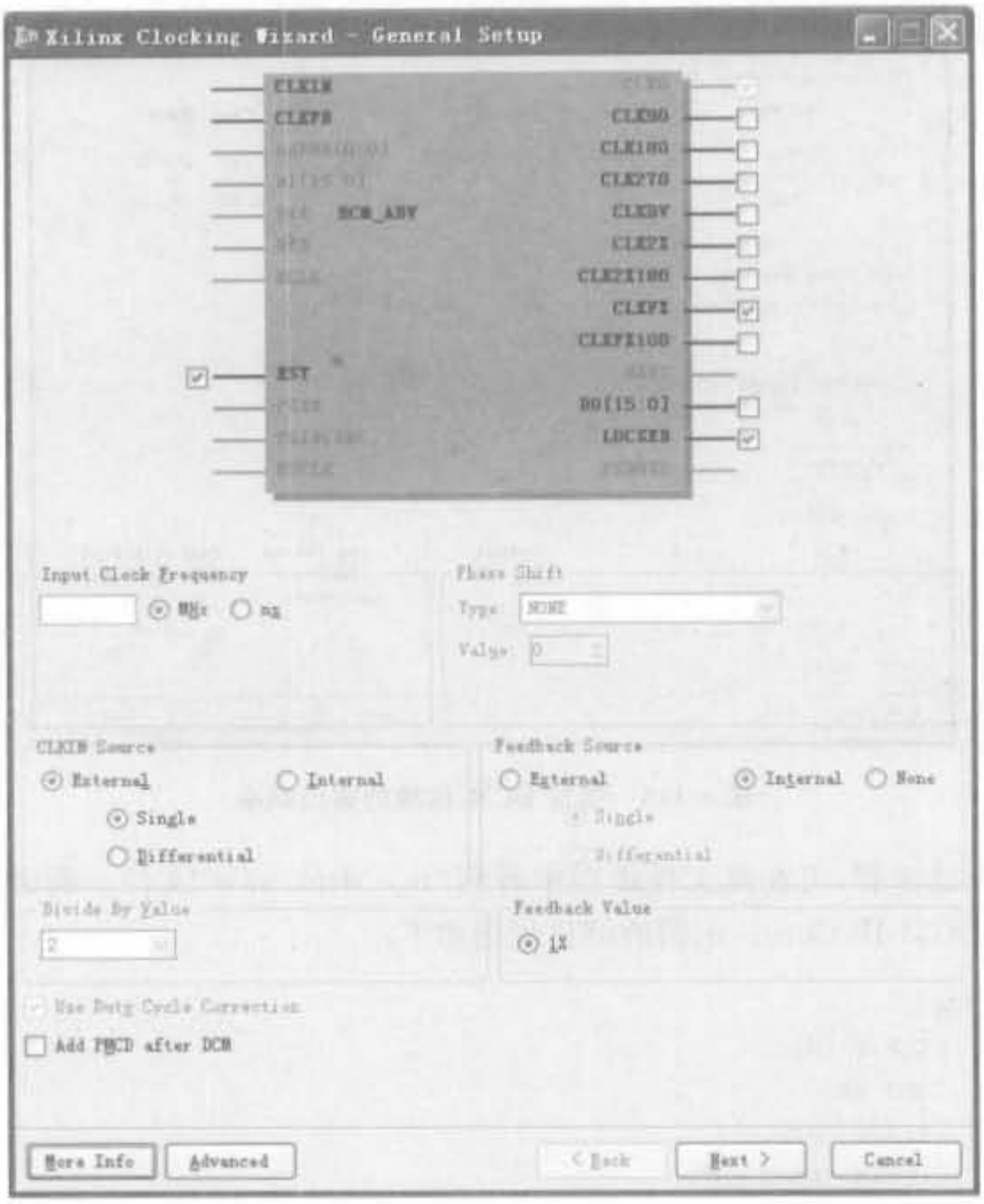


图 4-116 DCM 模块配置向导界面

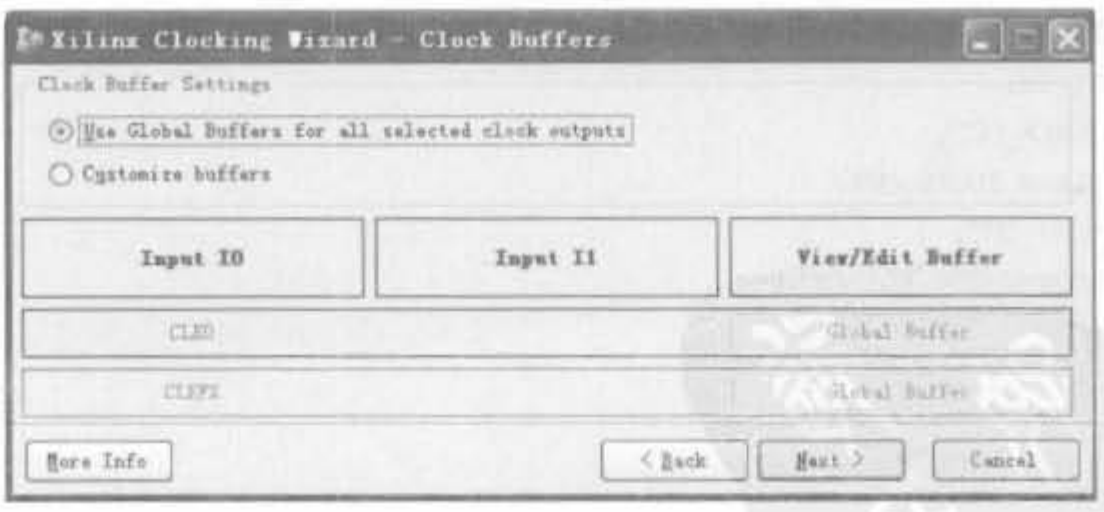


图 4-117 DCM 模块时钟缓存配置向导界面

(4) 单击“Next”,进入时钟频率配置窗口,如图 4-118 所示。键入输出频率的数值,或者将手动计算的分频比输入。最后单击“Next”→“Finish”,即可完成 DCM 模块 IP Core 的全部配置。本例直接输入输出频率 75MHz。

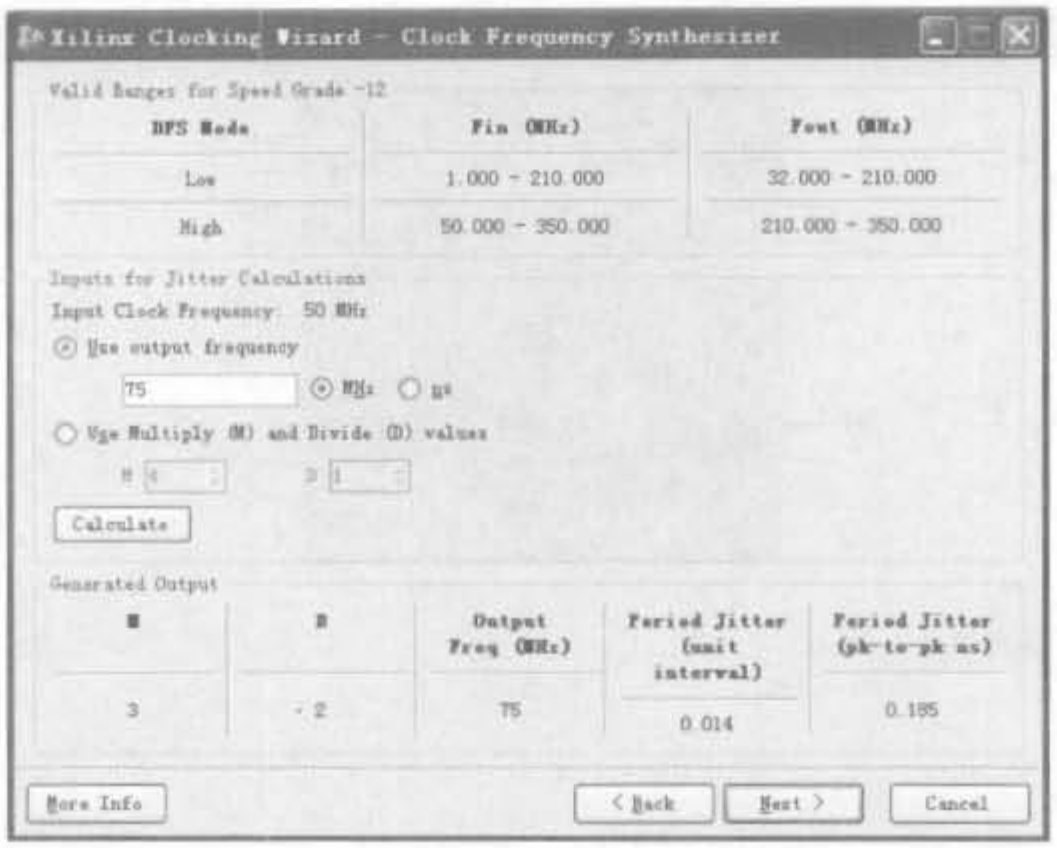


图 4-118　指定 DCM 模块的输出频率

(5) 经过上述步骤,可在源文件进程中看到“my_dcm. xaw”文件。剩余的工作就是在设计中调用该 DCM IP Core。示例的例化代码如下:

```
module dcm_top(
                CLKIN_IN,
                RST_IN,
                CLKFX_OUT,
                CLKIN_IBUFG_OUT,
                CLK0_OUT,
                LOCKED_OUT);

    input CLKIN_IN;
    input RST_IN;
    output CLKFX_OUT;
    output CLKIN_IBUFG_OUT;
    output CLK0_OUT;
    output LOCKED_OUT;

mydcm dcm1(
     .CLKIN_IN(CLKIN_IN),
     .RST_IN(RST_IN),
     .CLKFX_OUT(CLKFX_OUT),
     .CLKIN_IBUFG_OUT(CLKIN_IBUFG_OUT),
     .CLK0_OUT(CLK0_OUT),
     .LOCKED_OUT(LOCKED_OUT)
     );

endmodule
```

(6) 上述代码经过 Synplify Pro 综合后，得到的 RTL 级结构图如图 4-119 所示。

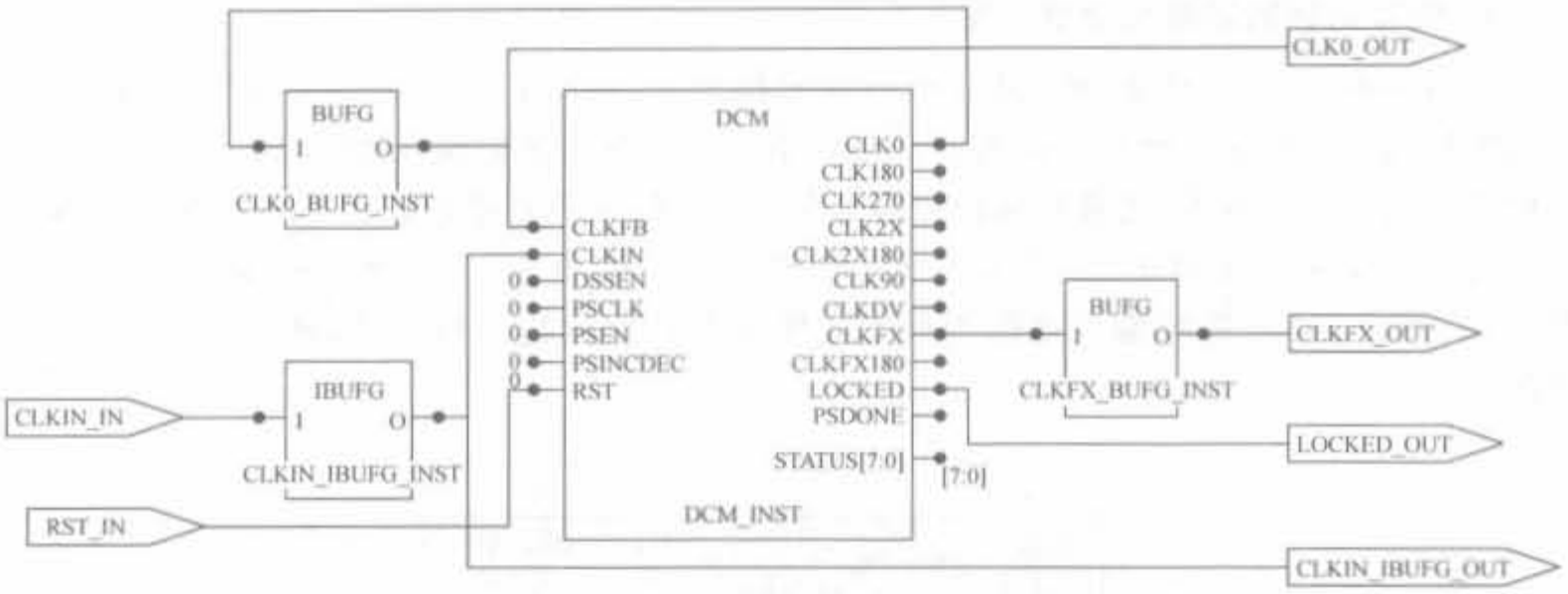


图 4-119 DCM 模块的 RTL 结构示意图

上述代码经过 ModelSim 仿真后，其局部仿真结果如图 4-120 所示。从中可以看出，当 LOCKED_OUT 信号变高时，DCM 模块稳定工作，输出时钟频率 CLKFX_OUT 为输入时钟 CLK_IN 频率的 1.5 倍，完成了预定功能。需要注意的是，复位信号 RST_IN 是高电平有效。

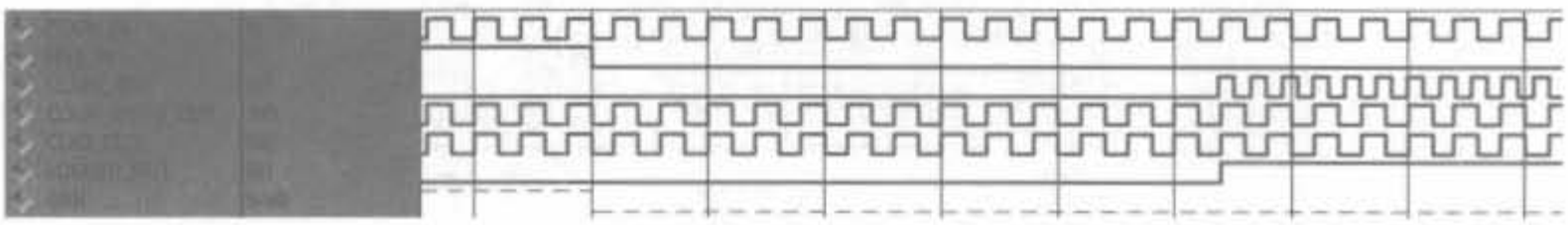


图 4-120 DCM 的仿真结果示意图

在实际中，如果在一片 FPGA 内使用两个 DCM，那么时钟从一个 clk 输入，再引到两个 DCM 的 clk_in。这里，在 DCM 模块操作时，需要注意两点：首先，用 CoreGen 生成 DCM 模块的时候，clk_in 源是内部的，不能直接连接到管脚，需要添加缓冲器；其次，手动例化一个 IBUFG，然后把 IBUFG 的输入连接到两个 DCM 的 clk_in。通常，如果没有设置 clk_in 源为内部的，而是完全按照单个 DCM 的使用流程，会造成 clk_in 信号有多个驱动。此时，ISE 不能做到两个 DCM 模块输出信号的相位对齐，只能做到一个 DCM 的输出是相位对齐的。而时钟管脚到两个 DCM 的路径和 DCM 输出的路径都有不同的延时，因此如果用户对相位还有要求，需要自己手动调整 DCM 模块在芯片中的位置。

4.6.3 Xilinx 内嵌块存储器的使用

Xilinx 公司提供了大量的存储器资源，包括内嵌的块存储器、分布式存储器以及 16 位移位寄存器。利用这些资源可以生成深度、位宽可配置的 RAM、ROM、FIFO 以及移位寄存器等存储逻辑。其中，块存储器是硬件存储器，不占用任何逻辑资源；其余两类都是 Xilinx 专有的存储结构，由 FPGA 芯片的查找表和触发器资源构建，每个查找表可构成 16×1 位的分布式存储器或移位寄存器。一般来讲，块存储器是宝贵的资源，通常用于大数据量场

合，而其余两类用于小数据量环境。

1. 块存储器的组成和功能介绍

在 Xilinx FPGA 中，块 RAM 是按照列来排列的，保证了每个 CLB 单元周围都有比较接近的块 RAM 用于存储和交换数据。与块 RAM 接近的是硬核乘加单元，这样不仅有利于提高乘法的运算速度，还能形成微处理器的雏形，在数字信号处理领域非常实用。例如，在 Spartan-3E 系列芯片中，块 RAM 分布于整个芯片的边缘，其外部一般有两列 CLB，如图 4-121 所示，可直接对输入数据进行大规模缓存以及数据同步操作，便于实现各种逻辑操作。

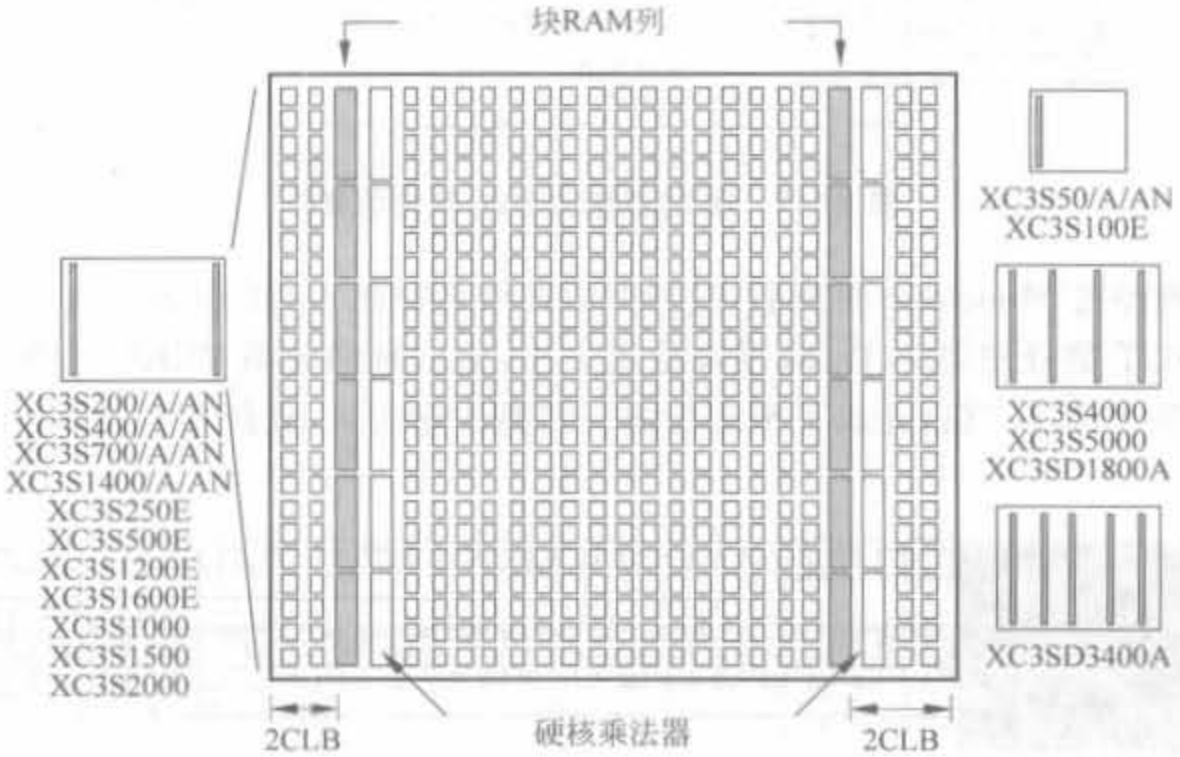


图 4-121　Spartan-3E 系统芯片中块 RAM 的分布图

块 RAM 几乎是 FPGA 器件中除了逻辑资源之外用得最多的功能块。Xilinx 的主流 FPGA 芯片内部都集成了数量不等的块 RAM 硬核资源，速度可以达到数百兆赫兹。它不会占用额外的 CLB 资源，而且可以在 ISE 环境的 IP 核生成器中灵活地对 RAM 进行配置，构成单端口 RAM、简单双口 RAM、真正双口 RAM、ROM（在 RAM 中存入初值）和 FIFO 等应用模式，如图 4-122 所示。同时，还可以将多个块 RAM 通过同步端口连接起来，构成容量更大的块 RAM。

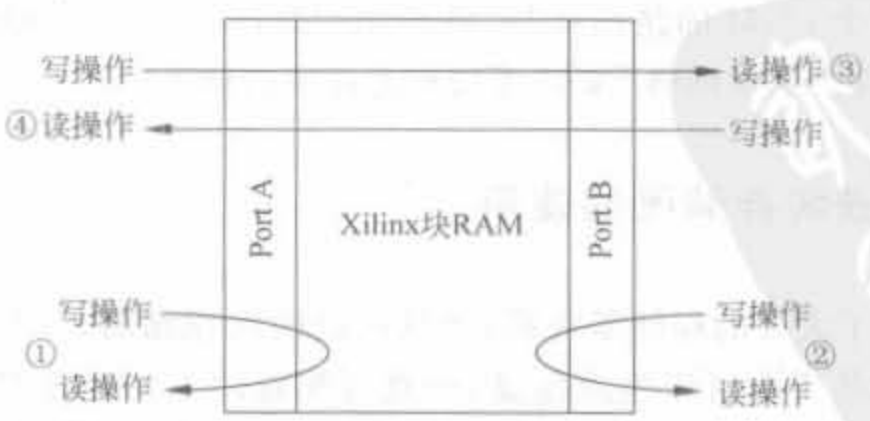


图 4-122　块 RAM 组合操作示意图

1）单端口 RAM 模式

单端口 RAM 的模型如图 4-123 所示，只有一个时钟源 CLK，WE 为写使能信号，EN 为单口 RAM 使能信号，SSR 为清零信号，ADDR 为地址信号，DI 和 DO 分别为写入和读出数据信号。

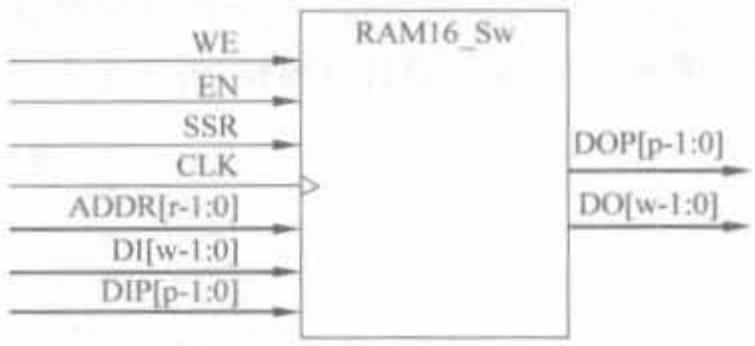


图 4-123 Xilinx 单端块 RAM 的示意模型

单端口 RAM 模式支持非同时的读写操作。同时，每个块 RAM 可以被分为两部分，分别实现两个独立的单端口 RAM。需要注意的是，当要实现两个独立的单端口 RAM 模块时，首先要保证每个模块所占用的存储空间小于块 RAM 存储空间的 1/2。在单端口 RAM 配置中，输出只在 read-during-write 模式有效，即只有在写操作有效时，写入到 RAM 的数据才能被读出。当输出寄存器被旁路时，新数据在其被写入时的时钟上升沿有效。

2）简单的双端口 RAM

简单双端口 RAM 模型如图 4-124 所示。图中，上边的端口只写，下边的端口只读，因此这种 RAM 也被称为伪双端口 RAM(Pseudo Dual Port RAM)。这种简单双端口 RAM 模式也支持同时的读写操作。

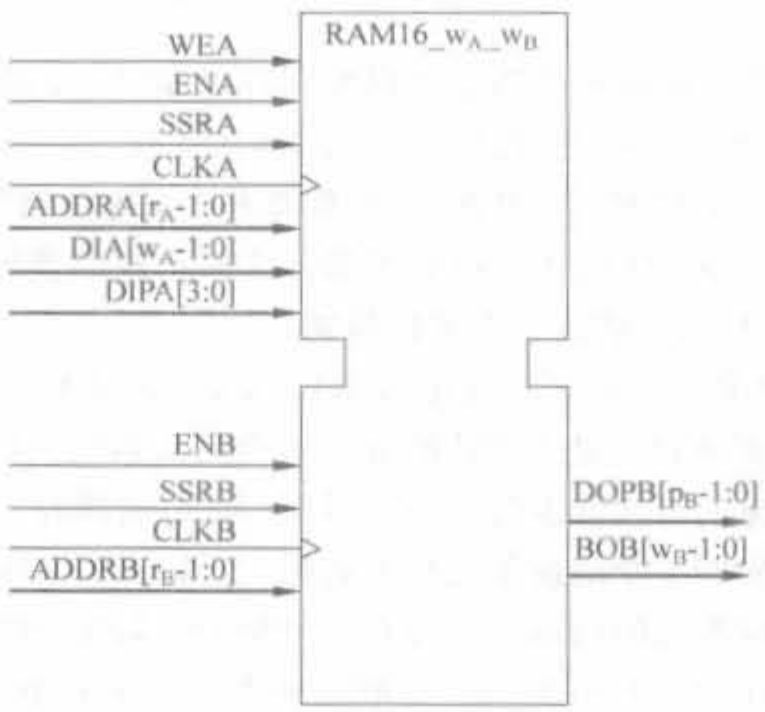


图 4-124 Xilinx 简单双端口块 RAM 的示意模型

块 RAM 支持不同的端口宽度设置，允许读端口宽度与写端口宽度不同。这一特性有着广泛地应用，例如不同总线宽度的并/串转换器等。在简单双端口 RAM 模式中，块 RAM 具有一个写使能信号 wren 和一个读使能信号 rden。当 rden 为高电平时，读操作有效；当读使能信号无效时，当前数据被保存在输出端口。当读操作和写操作同时对同一个地址单元时，简单双口 RAM 的输出或者是不确定值，或者是存储在此地址单元的原来的数据。

3）真正双端口 RAM 模式

真正双端口 RAM 模型如图 4-125 所示。图中，上边的端口 A 和下边的端口 B 都支持读写操作，WEA、WEB 信号为高电平时进行写操作，低电平时为读操作。同时，它支持两个

端口读写操作的任何组合：两个端口同时读操作、两个端口同时写操作；或者在两个不同的时钟下，一个端口执行写操作，另一个端口执行读操作。

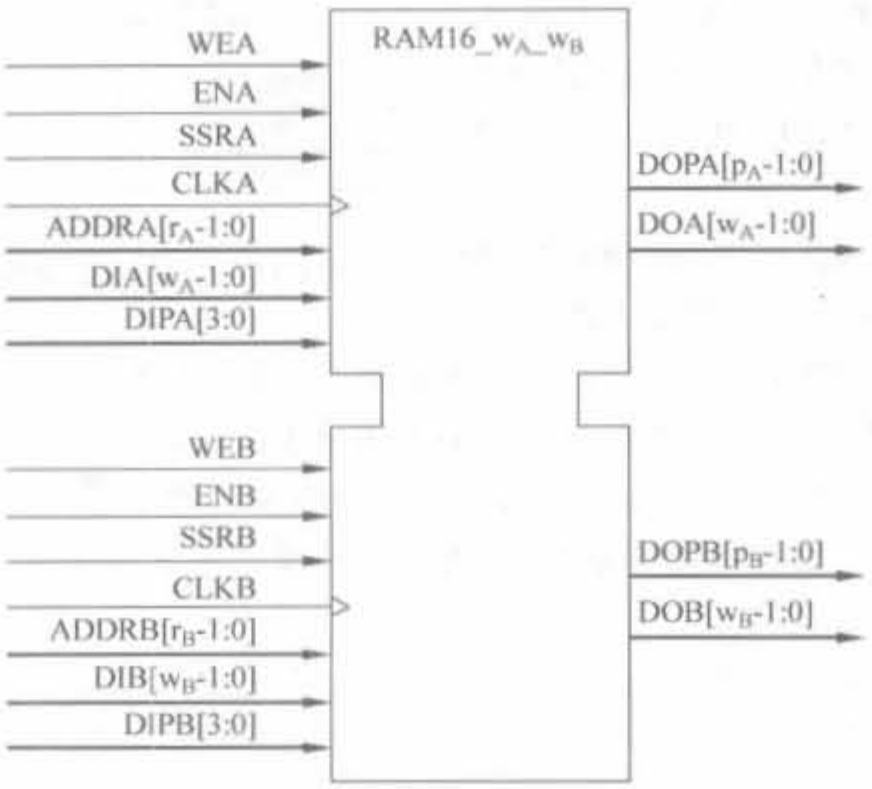


图 4-125　Xilinx 真正双端口块 RAM 的示意模型

真正双端口 RAM 模式在很多应用中可以增加存储带宽。例如，在包含嵌入式处理器 MiroBlaze 和 DMA 控制器系统中，采用真正双端口 RAM 模式会很方便；相反，如果在这样的一个系统中采用简单双端口 RAM 模式，当处理器和 DMA 控制器同时访问 RAM 时，就会出现问题。真正双端口 RAM 模式支持处理器和 DMA 控制器同时访问，这个特性避免了采用仲裁的麻烦，同时极大地提高了系统的带宽。

一般来讲，在单个块 RAM 实现的真正双端口 RAM 模式中，能达到的最宽数据位为 36bit×512，但可以采用级联多个块 RAM 的方式实现更宽数据位的双端口 RAM。当两个端口同时向同一个地址单元写入数据时，写冲突将会发生，这样存入该地址单元的信息将是未知的。要实现有效地向同一个地址单元写入数据，A 端口和 B 端口的时钟上升沿到来之间必须满足一个最小写周期时间间隔。因为在写时钟的下降沿，数据被写入块 RAM，所以 A 端口时钟的上升沿要比 B 端口时钟的上升沿晚到来 1/2 个最小写时钟周期，如果不满足这个时间要求，则存入此地址单元的数据无效。

4）ROM 模式

块 RAM 还可以配置成 ROM，使用存储器初始化文件(.coe)对 ROM 进行初始化，在上电后使其内部的内容保持不变，即实现了 ROM 功能。

5）FIFO 模式

FIFO 即先入先出，其模型如图 4-126 所示。在 FIFO 具体实现时，数据存储的部分是采用简单双端口模式操作的，一个端口只写数据，而另

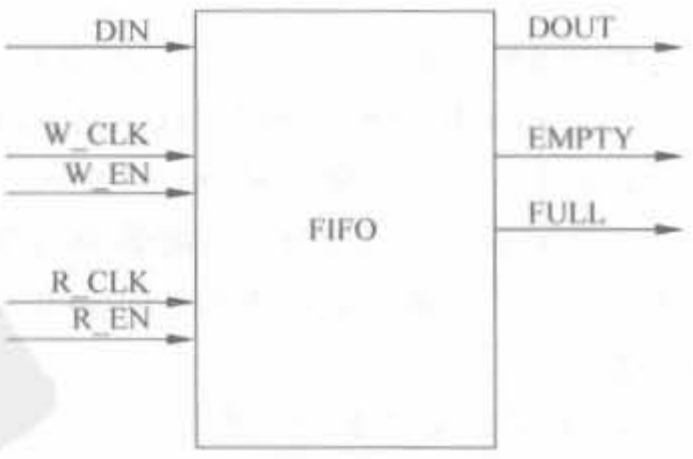


图 4-126　Xilinx FIFO 模块的示意模型

一个端口只读数据。另外,在RAM(块RAM和分布式RAM)周围加一些控制电路来输出指示信息。FIFO最重要的特征是具备“满(FULL)”和“空(EMPTY)”的指示信号,当FULL信号有效时(一般为高电平),就不能再往FIFO中写入数据,否则会造成数据丢失;当EMPTY信号有效时(一般为高电平),就不能再从FIFO中读取数据,此时输出端口处于高阻态。

2. 块RAM IP Core的使用

块RAM已在本书第3章有过介绍,这里不再赘述。

3. ROM存储器IP Core的使用

对于ROM模块,主要是生成相应的.coe文件。下面以一个实例介绍如何借助MATLAB生成ROM的.coe文件。

例4-7 生成定点正余弦波形数值,形成.coe文件并加载到块ROM中。

整体过程主要分为下面3步。

首先,利用MATLAB计算出正余弦波形的浮点值,并量化16bit定点波形数值:

```
x = linspace(0,6.28,1024); //在区间[0,6.28]之间等间隔地取1024个点
y1 = cos(x); //计算相应的正余弦值
y2 = sin(x);
//由于正余弦波形的值在[0,1]之间,需要量化成16bit,先将数值放大
y1 = y1 * 32678;
y2 = y2 * 32768;
//再将放大的浮点值量化,并写到存放在C盘的文本中
fid = fopen('c:/cos_coe.txt','wt');
fprintf(fid,'%16.0f\n',y1);   //在写文件的时候量化成16bit
fclose(fid)
fid = fopen('c:/sin_coe.txt','wt');
fprintf(fid,'%16.0f\n',y2);
fclose(fid)
```

其次,生成coe文件。在C盘根目录下,将cos_coe.txt和sin_coe.txt的后缀改成.coe。打开文件,把每一行之间的空格用文本的替换功能换成逗号“,”,并在最后一行添加一个分号“;”。最后,在文件的最开始添加下面两行:

```
memory_initialization_radix = 10;
memory_initialization_vector =
```

然后,保存文件退出。

最后,将coe文件加载到BLOCKROM所生成的ROM中。新建一个BLOCKRAM的IP Core,其位置为“Memories & Storage Elements→RAMs & ROMS→Block Memory Generator v2.4”。在第一页选择“single port rom”,在第二页选择位宽为16、深度为1024,在第三页下载coe文件,如图4-127所示,然后双击“Finish”,完成IP Core的生成。如果coe文件生成得不对,图中用椭圆标志之处是红色的,coe文件错误的类型主要有数据基数不对和数据的长度不对这两类。

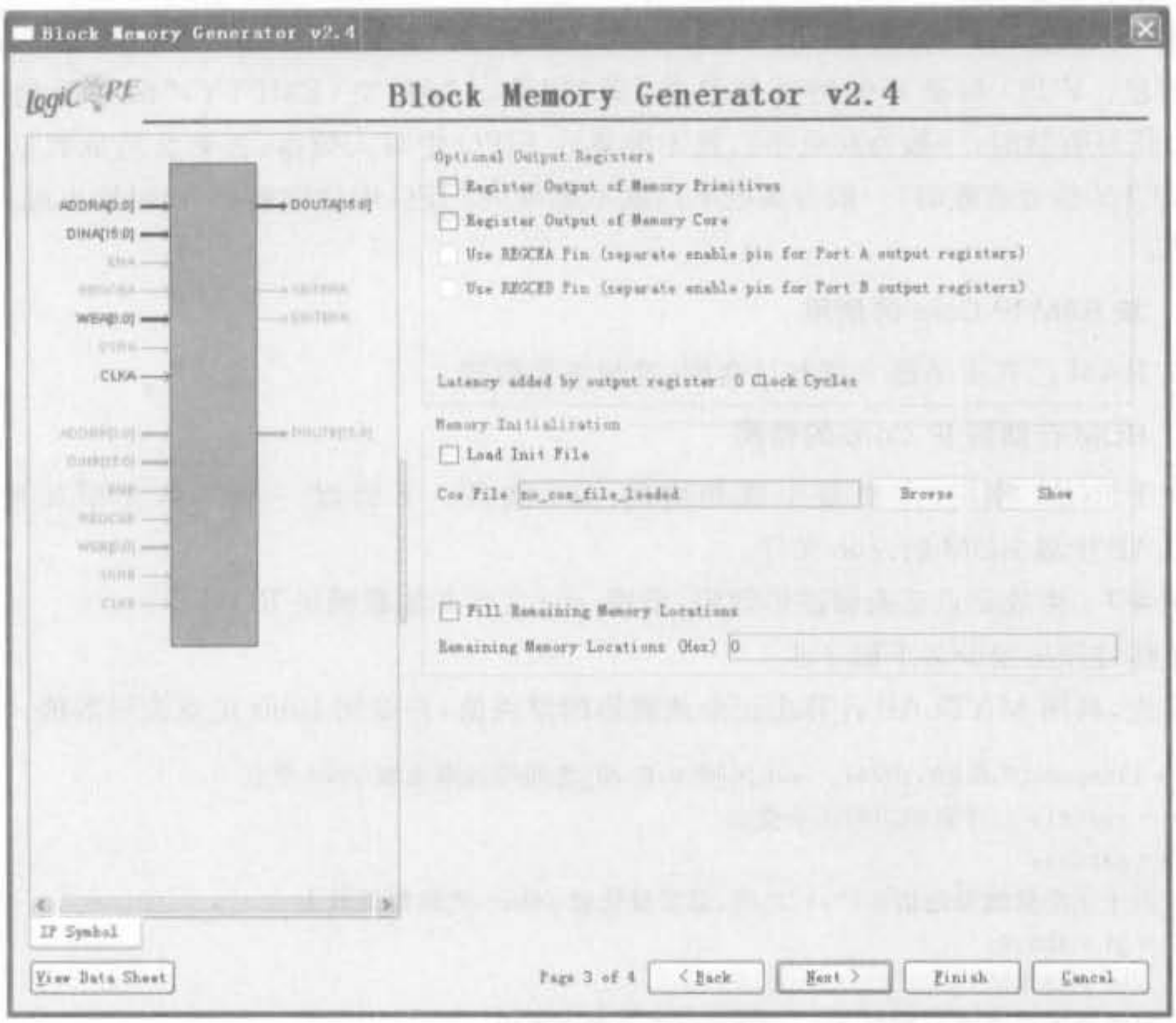


图 4-127　块 ROM 加载 coe 文件的用户配置界面

4.6.4　硬核乘加器的使用

随着 FPGA 芯片容量的提高和工艺的发展，很多 FPGA 内部都内嵌了硬件乘加器，所以 FPGA 内部的乘法器有两种实现方式：硬核乘法器的实现和用逻辑单元搭建的乘法器。由于内嵌了大量的硬核乘法器，FPGA 在数字信号处理系统方面的成本和功耗性能已经超越了专用的 DSP 处理器。

1. 硬核乘加器的组成和功能介绍

硬核乘加器是 Xilinx Xtreme DSP 解决方案的核心组成部分，利用它可以独立实现 500MHz 的性能，或在整合到一列中时实现 DSP 功能。它支持 40 多个动态控制的操作模式，包括乘法器、乘法器—累加器、乘法器—加法器/减法器、三输入加法器、桶形移位器、宽总线多路复用器或宽计数器，其组成结构如图 4-128 所示。此外，级联硬核乘加器无需使用 FPGA 逻辑和路由资源。

图 4-127 中的 OPMODE 是乘加器工作模式配置输入，可在 ISE 中通过软件方式指定。硬核乘加器的乘法器和加法器可以单独使用，但对于一个乘加器资源，只使用了其乘法器或加法器，则另外的加法器或乘法器不能再被使用。不同系列芯片中的乘加器的特点略有不同，从整体而言，都具有以下特点。

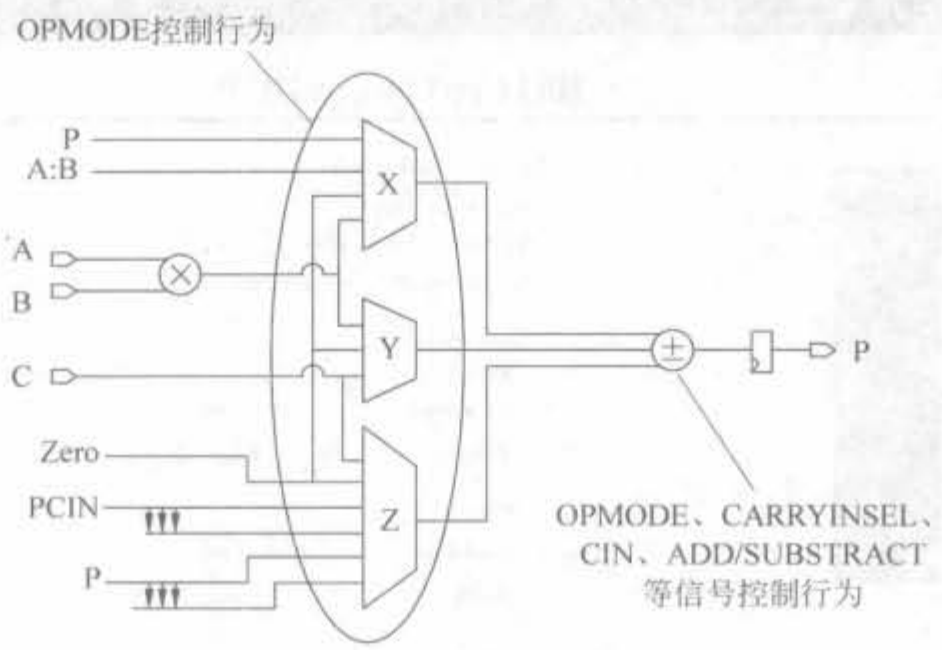


图 4-128 硬核乘加器的组成结构

- 18 位×18 位，两个补码乘法器具有完全准确的 36 位结果，符号可以扩展到 48 位。
- 三输入、灵活的 48 位加法器/减法器，具有可选的寄存器累加反馈。
- 40 多个动态用户控制器操作模式，使乘加器的功能适应从一个时钟周期到下一个时钟周期的变化。
- 级联的 18 位 B 总线，支持输入取样传递。
- 级联的 48 位 P 总线，支持部分结果的输出传递。
- 多精度乘法器和算法支持 17 位操作数右移位，以对准宽乘法器的部分乘积(并行或顺序乘法)。
- 对称智能舍入支持更高的计算精度。
- 控制和数据信号使用的、能够提高性能的流水线选项，可以通过配置位来进行选择。
- 输入端口“C”通常用作乘、加、进位的三操作数加或灵活的舍入模式。
- 独立的复位和时钟，实现了控制和数据寄存器。

2. 硬核乘法器 IP Core 的使用

硬核乘法器 IP Core 可以完成有符号数以及无符号数的乘法，还能够完成输出数据的位宽截取，支持流水线操作，功能强大。其用户操作界面如图 4-129 所示，单击“Next”按钮进入下一页，可以让用户选择是使用 FPGA 芯片上的硬乘法器(Use Mults)，还是用 Slice 来构建乘法器(Use LUT)。

乘法器具有丰富的控制信号，详细说明如下。

(1) A：乘法器的一个输入操作数，在使用时应当确定其位宽。位宽可以在右边的端口位宽编辑框中输入，并且可以有符号数、无符号数两种选择。

(2) B：乘法器的一个输入操作数，在使用时应当确定其位宽。位宽可以在右边的端口位宽编辑框中输入，而且可以与 A 口的输入操作数位宽不同。有符号数、无符号数的两种选择。注意，乘法器可以只接收 A 口的输入信号，而与一个常数相乘，此常数可以为固定的或可重导入的。对于可重导入数据(RCCM)的情况，此常数可以在 B 口重新导入而得到改变。

(3) CLK：乘法器的工作时钟。上升沿有效。

(4) CE：输入信号，指示时钟是否有效。

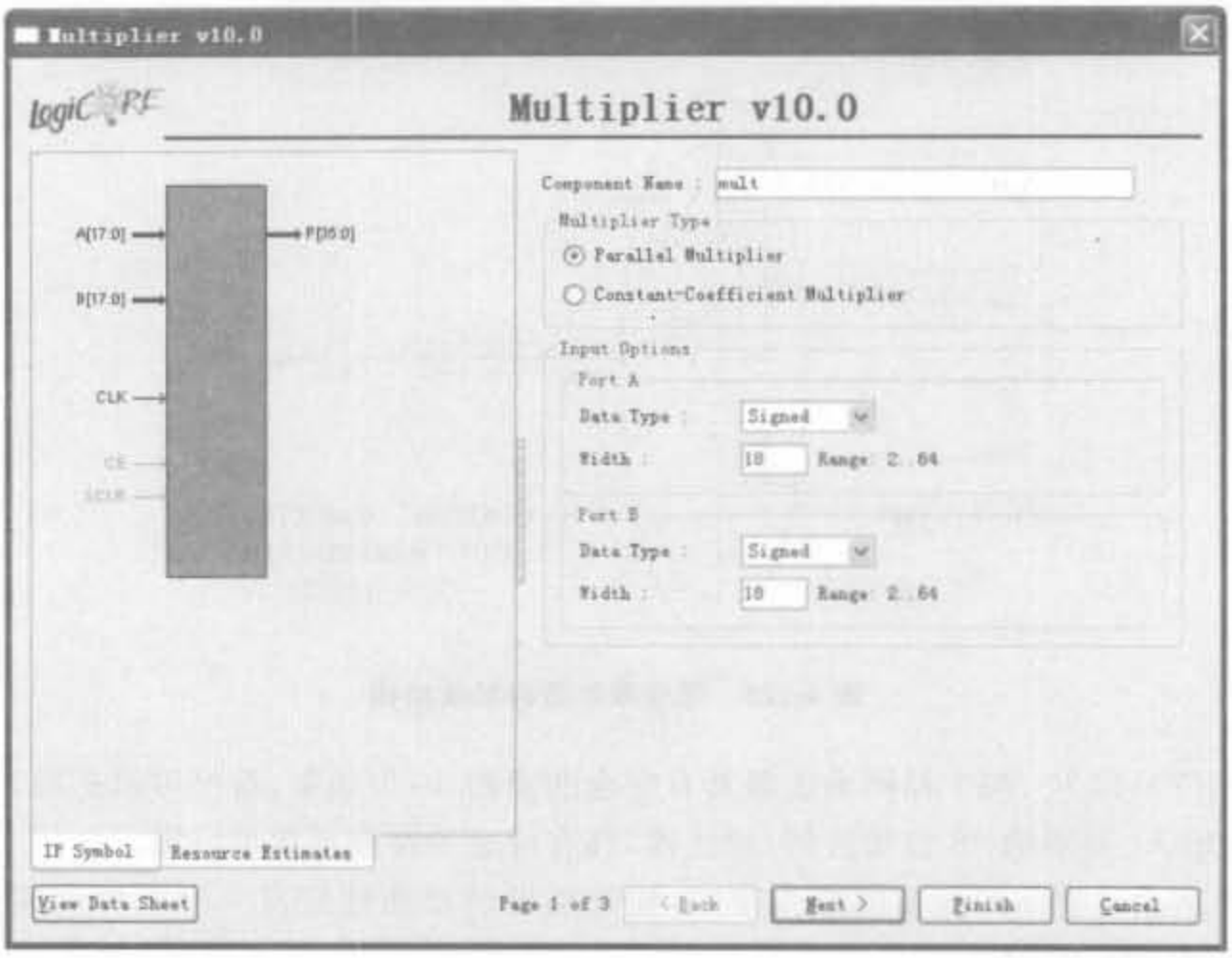


图 4-129　乘法器 IP Core 用户操作界面

(5) ND：握手信号。

(6) ACLR：异步清零信号。

(7) SCLR：同步清零信号。

(8) LOADB：当 RCCM 时才有效。当此信号为高电平时，B 端口新的输入可以重新写入计算模块的存储单元。

(9) SWAPB：当 RCCM 时才有效。当此信号为高电平时，在计算模块的存储单元已存有的多个常数中进行选择。

(10) RDY：输出的握手信号，当其变为高电平时，表明数据有效。

(11) RFD：握手信号，在其变为高电平后的下一个时钟上升沿，数据有效。

(12) O：异步输出信号，位宽根据输入信号的位宽而定。

(13) Q：同步输出信号，位宽根据输入信号的位宽而定。

(14) LOAD_DONE：当 RCCM 时才有效，指示重新导入数据的过程完成。

例 4-8　使用 IP Core 实例化一个 16 位乘法器

IP Core 直接生成的乘法器的 Verilog 模块接口为：

```
module multiply(sclr,rfd,rdy,nd,clk,a,b,q);
    input sclr;
    output rfd;
    output rdy;
    input nd;
    input clk;
    input [17:0] a;
    input [17:0] b;
    output [35:0] q;
```

```
......
endmodule
```

在使用时，直接调用 multiply 模块即可，如

```
module multiply1 (sclr,rfd,rdy,nd,clk,a,b,q);
    input sclr,nd,clk;
    output rfd,rdy;
    input [15:0] a,b;
    output [31:0] q;

multiply multiply1(.sclr(sclr),.rfd(rfd),.rdy(rdy),.nd(nd),.clk(clk),
                   .a(a),.b(b),.q(q));
endmuodule
```

上述程序经过 Synplify Pro 综合后，得到的 RTL 结构如图 4-130 所示。

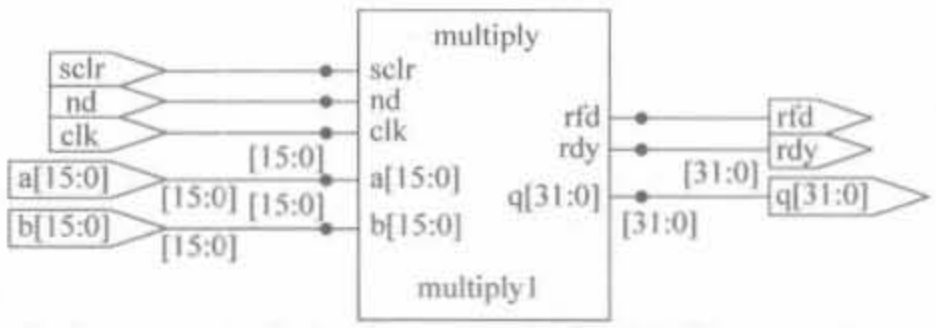


图 4-130 乘法器 IP Core 综合后的 RTL 结构图

经过仿真测试得到的功能波形图如图 4-131 所示，它正确地实现了乘法功能。

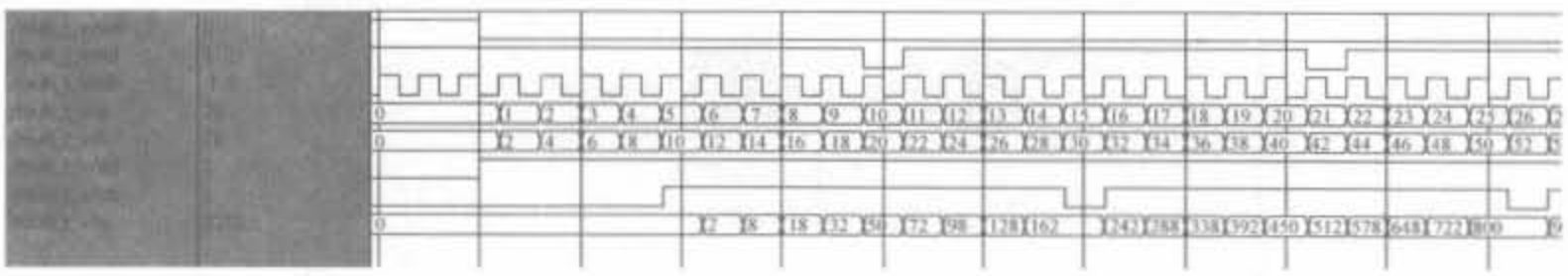


图 4-131 乘法器 IP Core 的仿真波形

3. 硬核乘加器 IP Core 的使用

硬核乘加器在乘法器后面级联了一个可控加法器，都可工作在芯片的最高工作频率。下面给出一个应用实例。

例 4-9 使用硬核乘加器完成两路输入数据的相乘，并将每 8 个乘积结果累加后送出。其中，输入数据为 16bit，工作频率为 50MHz。

(1) 在工程中添加硬核乘加器的 IP Core 文件，位于“FPGA Features and Design”→“XtremeDSP Slice”→“Multiply Accumulator v9.1i”。

(2) 配置 IP Core 参数。输入数据位宽设为 16bit，且输入/输出不与其余的 DSP Slice 级联，输出位宽设置为 19bit，如图 4-132 所示。单击“Next”进入下一页。无进位选项，且只选择加法输入，OPMODE 模式设置为“Normal accumulator mode”，如图 4-133 所示。单击“Next”进入下一页。A、B 流水线都设为 1，其余设置如图 4-134 所示。单击“Next”进入下一页。最后一页为乘加器的配置参数列表，如图 4-135 所示，单击“Finish”按钮，即可完成全部配置。

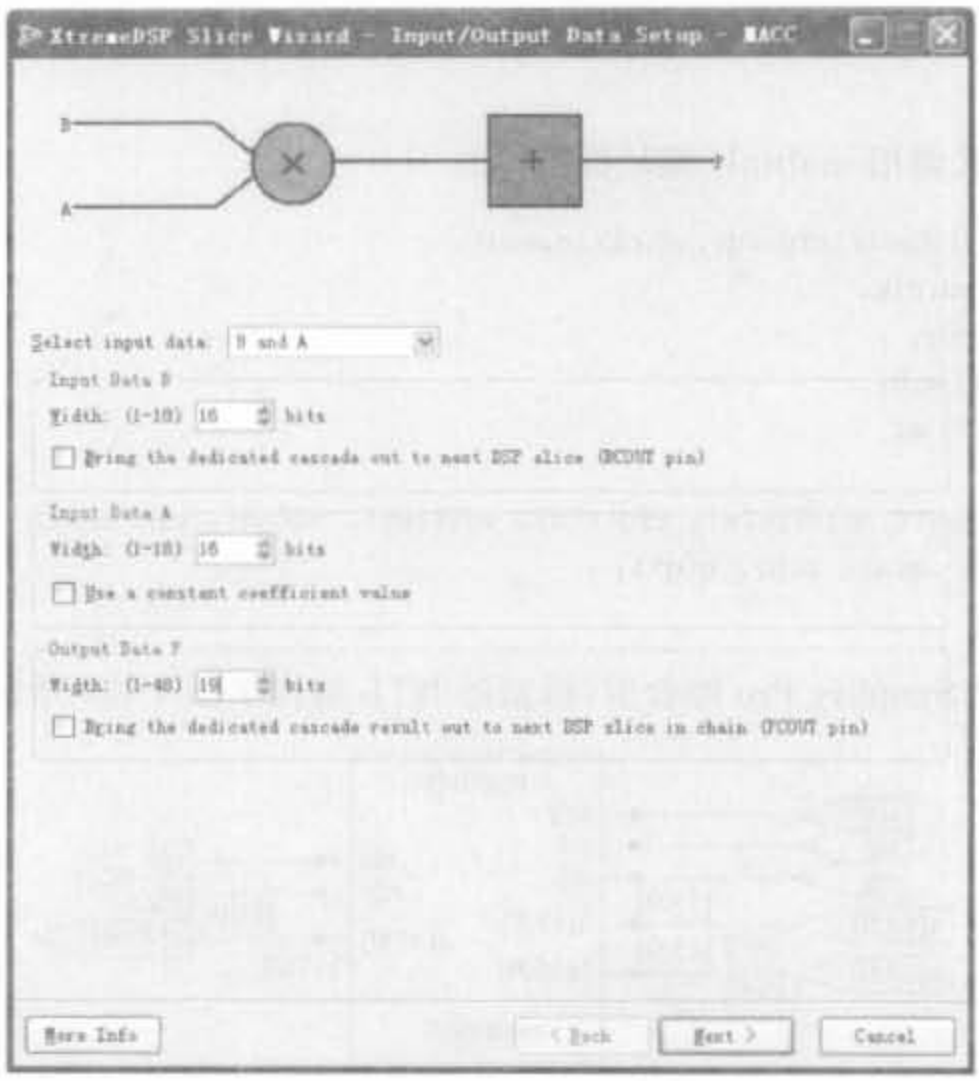


图 4-132　乘法器 IP Core 的配置界面(1)

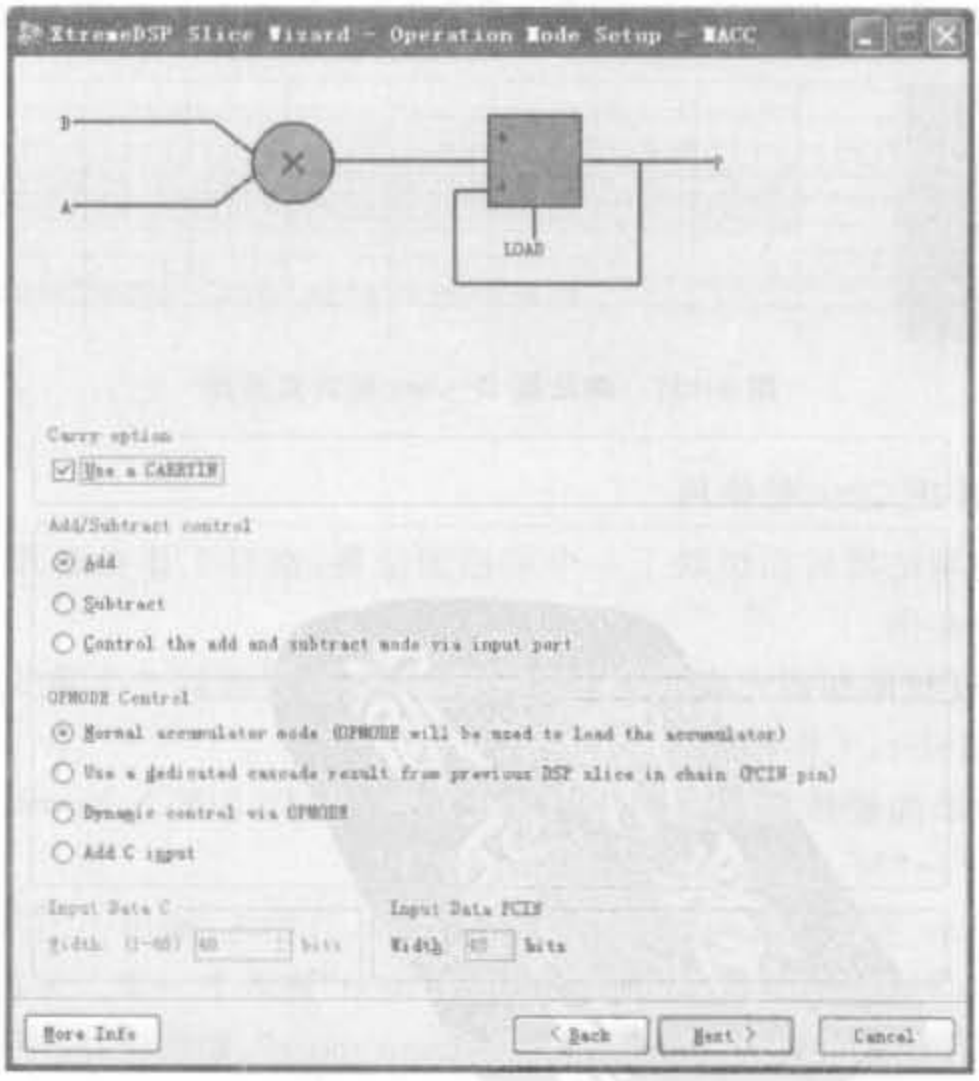


图 4-133　乘法器 IP Core 的配置界面(2)

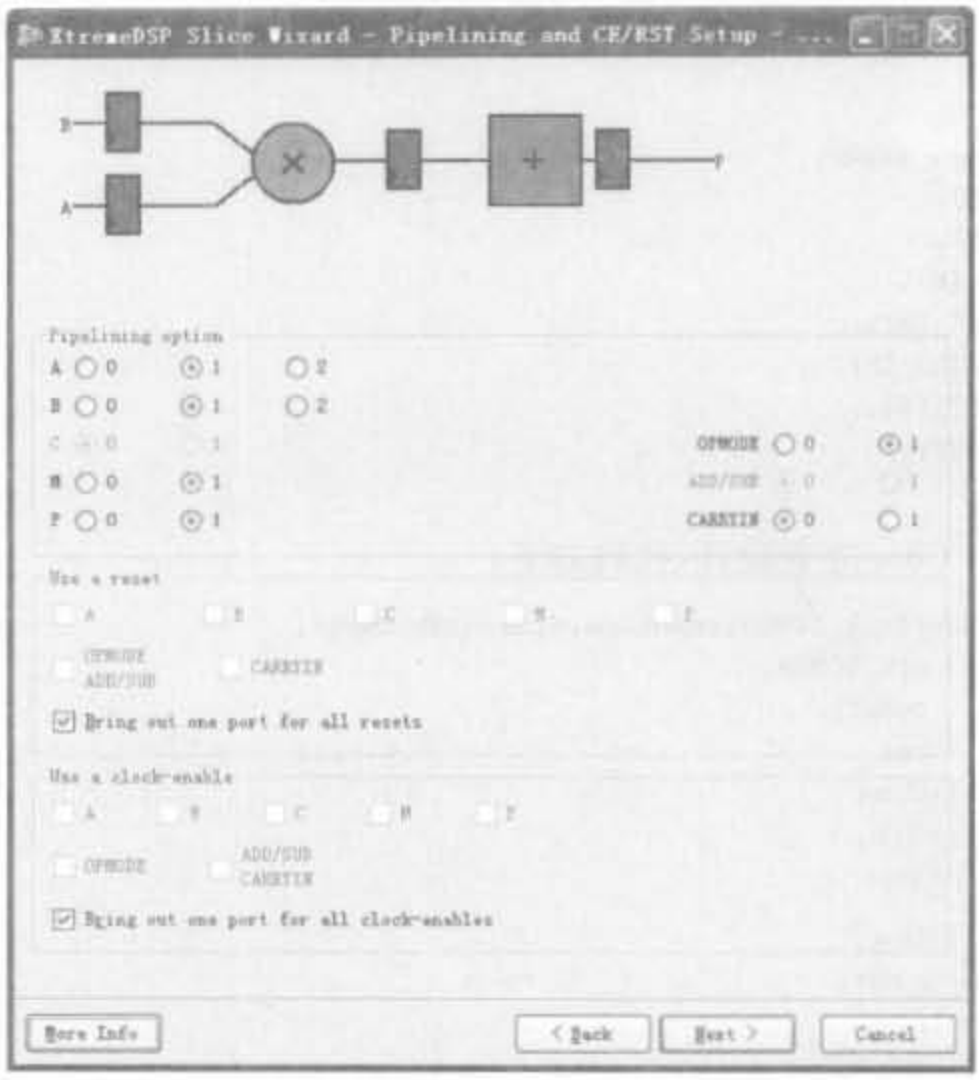


图 4-134 乘法器 IP Core 的配置界面(3)

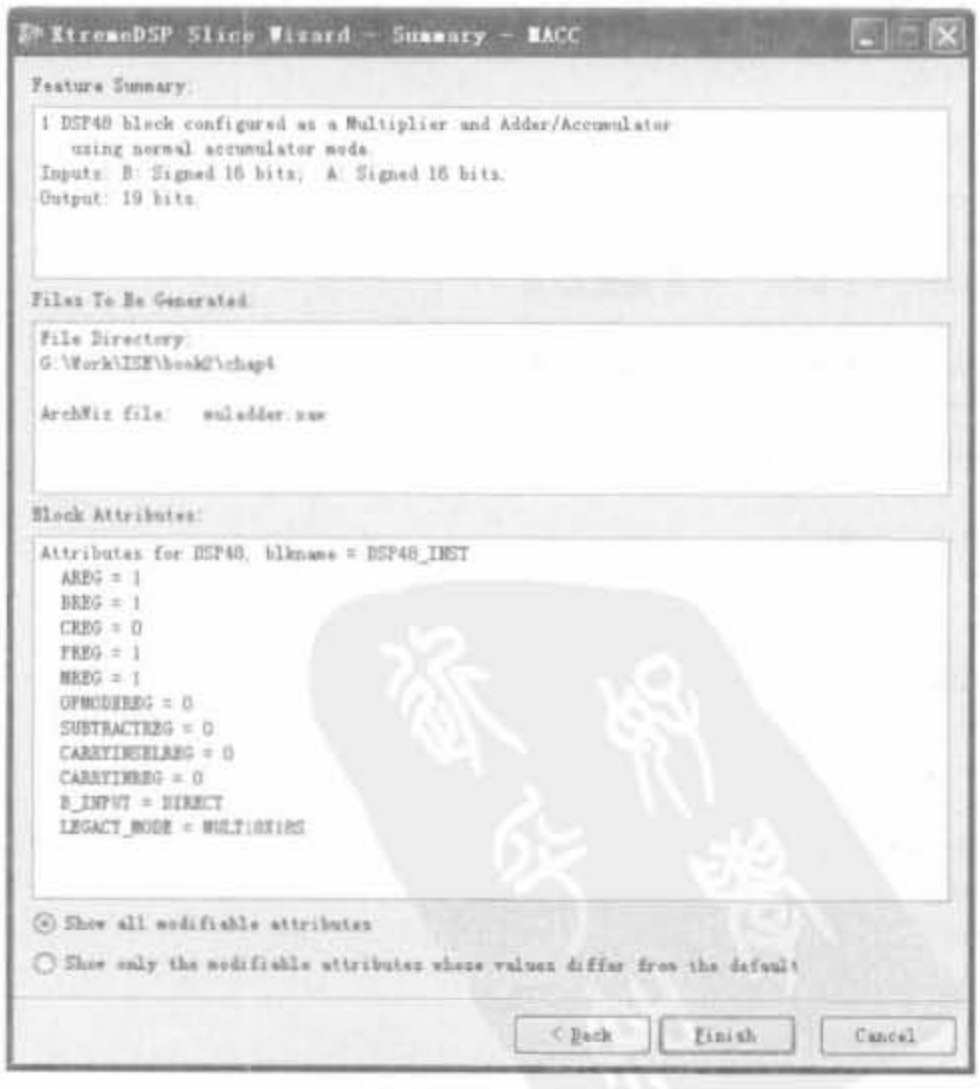


图 4-135 乘法器 IP Core 的配置界面(4)

(3) 在过程窗口中单击“View HDL Instantiation Template”命令,可查阅其代码例化模板,如下所示:

```
muladder instance_name (
    .A_IN(A_IN),
    .B_IN(B_IN),
    .CE_IN(CE_IN),
    .CLK_IN(CLK_IN),
    .LOAD_IN(LOAD_IN),
    .RST_IN(RST_IN),
    .P_OUT(P_OUT)
);
```

(4) 调用该 IP Core 完成设计,代码如下:

```
module my_muladder(clk_50MHz,reset,ce,dina,dinb,dout);
  input          clk_50MHz;
  input          reset;
  input          ce;
  input  [15:0] dina;
  input  [15:0] dinb;
  output [34:0] dout;

  reg    [34:0] dout;
  wire   [34:0] p_out;
  reg    load = 0;
  reg    [2:0]  cnt = 0;
  always @(posedge clk_50MHz) begin
     if(reset) begin
       cnt   <= 0;
       load<= 0;
       dout<= 0;
     end
     else begin
       cnt<= cnt + 1'b1;
       if(cnt == 0)
            load<= 1'b0;   //直通加法器
       else
            load<= 1'b1;   //load = 1 累加
       if(cnt == 2)
            dout<= p_out;
       else
            dout<= dout;
     end
  end

  muladder muladder(
     .A_IN(dina),
     .B_IN(dinb),
     .CE_IN(ce),
     .CLK_IN(clk_50MHz),
     .LOAD_IN(load),
     .RST_IN(reset),
     .P_OUT(p_out)
     );

endmodule
```

上述程序经过 Synplify Pro 综合后,得到的 RTL 结构如图 4-136 所示。

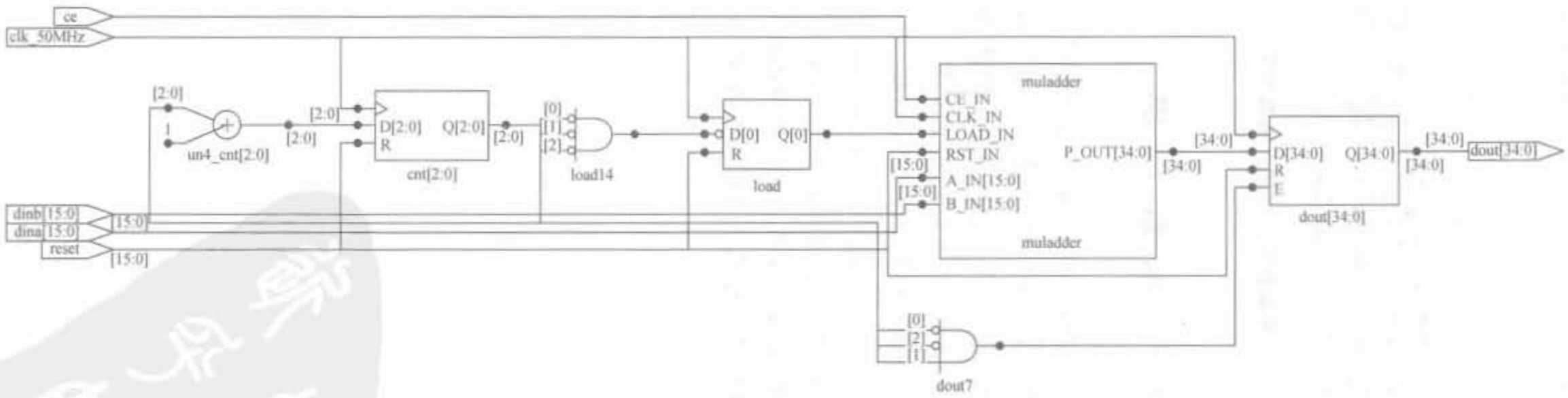


图 4-136　乘加器应用程序的 RTL 结构图

经过仿真测试得到的功能波形图如图 4-137 所示。可以看到，本例正确地实现了 8 个乘积结果累加的功能。

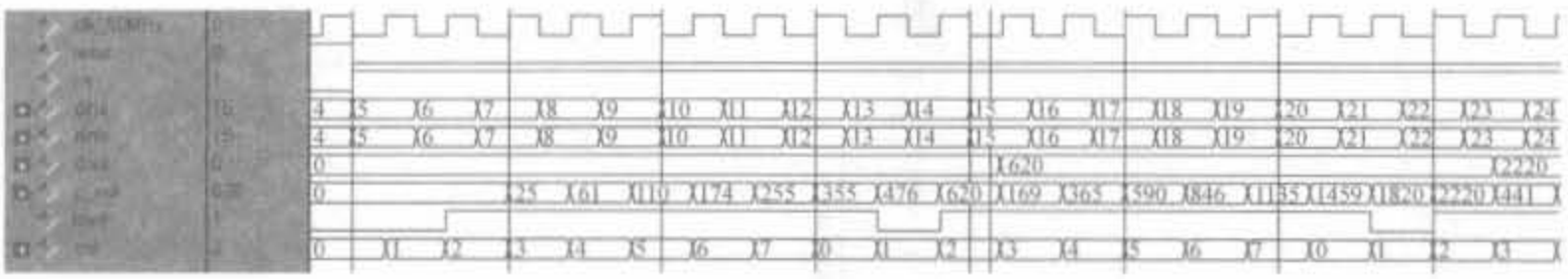


图 4-137　乘加器应用程序的仿真结果

4.7　本章小结

本章详细介绍了基于 ISE 的 FPGA 设计流程以及多个辅助工具(XST、XPower、PACE、ModelSim、Synplify 以及 MATLAB)的使用方法。本章首先介绍了 ISE 软件的主要特性及其安装流程，然后介绍了如何通过 ISE 完成 FPGA 设计，详细介绍了综合、仿真以及实现的软件操作和 XST、XPower、PACE 等工具的基本操作。之后，简单介绍了 Synplify Pro 和 ModelSim SE 的安装流程，并介绍在 ISE 关联 Synplify Pro 和 ModelSim SE 的使用方法以及和 ISE 的联合开发流程。然后，介绍了 MATLAB 和 ISE 的联合开发模式，其中利用 MATLAB 来辅助 ISE 开发是目前流行的设计方法之一，也是本书所强调的重点。最后，介绍了 Xilinx FPGA 底层单元(DCM、块 RAM 以及硬核乘加器)的原理和使用方法。限于篇幅，本章不可能对以上所提及软件的其他功能进行详细介绍，读者应该在实际工作中学习并熟练掌握。

第5章

FPGA配置电路及软件操作

CHAPTER 5

FPGA 器件基于 SRAM 结构来实现可编程特性，具有集成度高、逻辑功能强等特点，但掉电后，编程信息立即丢失。芯片在每次加电时，都必须重新下载设计文件所生成的配置数据。正是这一特性使得用 FPGA 来实现数字系统时，不但能减小电路板的面积，减少开发周期，提高系统运行的可靠性，而且可无限次重复擦写，完成数字系统的在线重配置，易于设备功能的更改和升级。因此，如何快速、高效地将配置数据写入目标器件，并且保证其在掉电后再次上电能自动、可靠地恢复配置，成为整个系统的关键所在。目前，配置 FPGA 的方式有多种，可以通过 JTAG 口配置(一般用在调试过程)，可以通过专用的 PROM 来配置，也可以通过 CPU 或者 CPLD 进行配置(常用在最终产品中)。本章主要介绍 Xilinx FPGA 常用配置电路的原理以及相应的软件操作。

5.1 FPGA 配置电路综述

FPGA 配置电路可看成用户设计和硬件电路之间的连接纽带。随着制造工艺的发展和应用范围的扩展，FPGA 配置电路呈现多元化发展，最终目的都是在一定的外部条件下，准确、快速地实现 FPGA 系统配置。只有正确地理解了配置电路，才能在实际开发中选择最优的解决方案。本节主要从宏观上介绍 FPGA 配置电路的组成、工作流程以及不同方式之间的差异。

5.1.1 Xilinx FPGA 配置电路综述

FPGA 芯片是基于 SRAM 工艺的，不具备非易失特性，因此断电后将丢失内部逻辑配置。芯片在每次上电后，都需要从外部非易失存储器中导入配置比特流。硬件配置是 FPGA 开发最关键的一步，只有将 HDL 代码下载到 FPGA 芯片中，才能调试并最终实现相应的功能。完成 FPGA 配置，必须要有类似于单片机仿真器的下载电缆。典型的 FPGA 配置系统如图 5-1 所示。

在 FPGA 配置系统中，编程软件由 FPGA 提供商提供，设计人员要掌握其操作方法；下载电缆是固定的 JTAG 电路，只要将其连接在 PC 上以及目标板上即可；只有目标板上的配置电路需要设计人员设计。其中，JTAG 链路是器件编程的关键传输枢纽，因此 JTAG 链路的工作原理、FPGA 的各种配置电路以及编程软件的操作是本章的重点内容。

将配置数据从 PC 上加载到 Xilinx FPGA 芯片中的整个配置过程，可分为以下步骤：

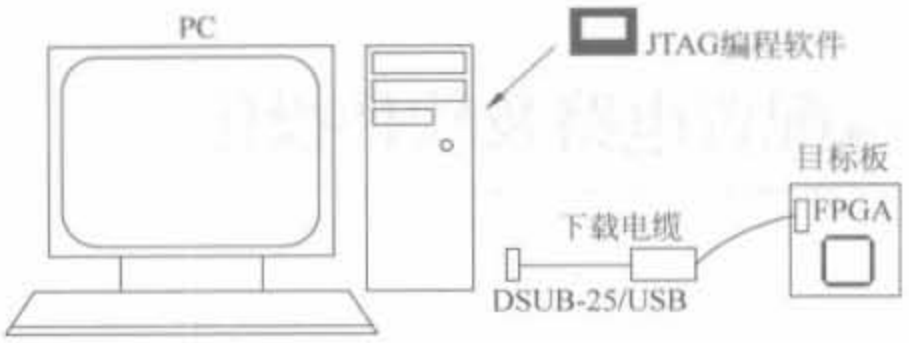


图 5-1 FPGA 配置系统组成

1. 初始化

通上电后，如果 FPGA 满足以下条件：Bank2 的 I/O 输出驱动电压 VCCO_2 大于 1V；器件内部的供电电压 VCCIONT 为 2.5V，器件便会自动进行初始化。在系统上电的情况下，通过对 PROG 管脚置低电平，便可以对 FPGA 重新配置。初始化过程完成后，DONE 信号将会变低。

2. 清空配置存储器

在完成初始化过程后，器件会将 INIT 信号置低电平，同时开始清空配置存储器。在清空完配置存储器后，INIT 信号将会重新被置为高电平。用户可以通过将 PROG 或 INIT 信号(INIT 为双向信号)置为低电平，达到延长清空配置存储器的时间，以确保存储器被清空的目的。

3. 加载配置数据

配置存储器的清空完成后，器件对配置模式脚 M2、N1、M0 进行采样，以确定用何种方式来加载配置数据。

4. CRC 错误检查

器件在加载配置数据的同时，会根据一定的算法产生一个 CRC 值，这个值将会和配置文件中内置的 CRC 值进行比较，如果两者不一致，说明加载发生错误，INIT 管脚将会被置低电平，加载过程被中断。此时，若要重新配置，只需将 PROG 置为低电平即可。

5. START-UP

START-UP 阶段是 FPGA 由配置状态过渡到用户状态的过程。在 START-UP 完成后，FPGA 便可实现用户编程的功能。在 START-UP 阶段中，FPGA 进行以下操作：

(1) 将 DONE 信号置高电平，若 DONE 信号没有置高，说明数据加载过程失败。

(2) 在配置过程中，器件的所有 I/O 管脚均为三态。此时，全局三态信号 GTS 置低电平，这些 I/O 脚将会从三态切换到用户设置的状态。

(3) 全局复位信号 GSR 置低电平，所有触发器进入工作状态。

(4) 全局写允许信号 GWE 置低电平，所有内部 RAM 有效。

整个过程需要 8 个时钟周期 C0-C7。在默认的情况下，这些操作都和配置时钟 CCLK 同步。在 DONE 信号置高电平之前，GTS、GSR、GWE 都保持高电平。如果选用 JTAG 配置电路，则所有操作都和 JTAG 电路的 TCK 保持同步。

5.1.2 Xilinx FPGA 常用的配置管脚

对于 FPGA 芯片而言，无论何种配置方式，都必须通过 FPGA 相应的管脚把设计加载到 FPGA 芯片中。和配置有关的管脚可以分为专用管脚和复用管脚两类，前者只能用于 FPGA 配置，而后者在配置过程结束后，还可当作普通 I/O 使用。

专用的配置管脚有配置模式脚 M2、M1、M0；配置时钟 CCLK；配置逻辑异步复位 PROG；启动控制 DONE 及边界扫描 TDI、TDO、TMS、TCK。非专用配置管脚有 Din、D0、D7、CS、WRITE、BUSY 和 INIT。

当然，部分专业配置管脚在配置结束后也可作为普通管脚使用。例如，在 Spartan-3E 系列 FPGA 中，3 个 FPGA 管脚 M2、M1 和 M0 用于选择配置模式，其配置说明如表 5-1 所示，M[2:0]的值在 INIT_B 输出变高后才有效。在 FPGA 配置完成后，M[2:0]可以作为普通 I/O 使用。

表 5-1 Spartan-3E 配置模式选择以及特性综述

	主串模式	SPI 模式	BPI 模式	从并模式	从串模式	JTAG
模式配置管脚 M[2:0]	<0:0:0>	<0:0:1>	<0:0:0>=UP <0:0:0>=DOWN	<0:0:0>	<0:0:0>	<0:0:0>
数据宽度	单比特	单比特	字节宽度	字节宽度	单比特	单比特
配置数据存储器类型	Xilinx 系列 PROM	标准 SPI 串行 FLASH	Xilinx 系列 PROM 或并行 NOR FLASH	外部 CPU、MCU 或 Xilinx 并行 PROM		外部 CPU、MCU
配置时钟源	FPGA 提供	FPGA 提供	FPGA 提供	CCLK 管脚上的外部时钟		TCK 信号
配置所需管脚数	8	13	46	21	8	0
配置菊花链中的 FPGA	从串	从串	从并	从并	从串	JTAG
是否需要额外配置主机	否	否	否	使用 Xilinx 并行 PROM 不需要，否则需要外部主机		否

5.1.3 Xilinx FPGA 配置电路分类

FPGA 配置方式灵活多样，根据芯片是否能够自己主动加载配置数据，分为主模式、从模式以及 JTAG 模式。典型的主模式都是加载片外非易失(断电不丢数据)性存储器中的配置比特流，配置所需的时钟信号(称为 CCLK)由 FPGA 内部产生，且 FPGA 控制整个配置过程。从模式需要外部的主智能终端(如处理器、微控制器或者 DSP 等)将数据下载到 FPGA 中，其最大的优点就是 FPGA 的配置数据可以放在系统的任何存储部位，包括 Flash、硬盘、网络，甚至在其余处理器的运行代码中。JTAG 模式为调试模式，可将 PC 中的比特文件流下载到 FPGA 中，断电即丢失。此外，目前 Xilinx 还有基于 Internet 的、成熟的可重构逻辑技术 System ACE 解决方案。

1. 主模式

在主模式下，FPGA 上电后，自动将配置数据从相应的外存储器读入到 SRAM 中，实现内部结构映射。主模式根据比特流的位宽又可以分为串行模式（单比特流）和并行模式（字节宽度比特流）两大类，如主串行模式、主 SPI Flash 串行模式、内部主 SPI Flash 串行模式、主 BPI 并行模式以及主并行模式，如图 5-2 所示。

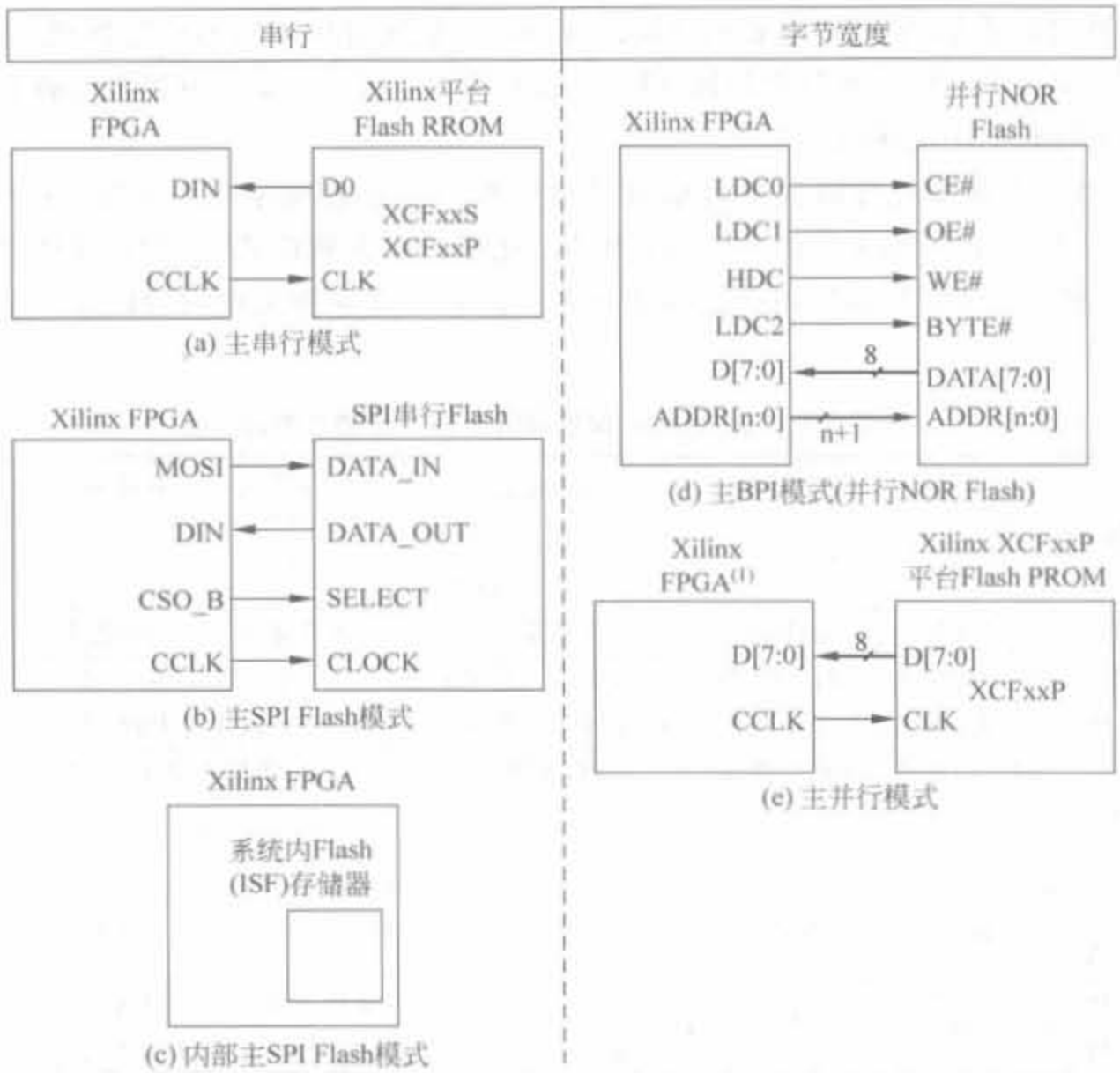


图 5-2　常用的主模式下载方式示意图

2. 从模式

在从模式下，FPGA 作为从属器件，由相应的控制电路或微处理器提供配置所需的时序，实现配置数据的下载。从模式也根据比特流的位宽不同，分为串、并模式两类，具体包括从串行模式、JTAG 模式和从并行模式三大类，其概要说明如图 5-3 所示。

3. JTAG 模式

在 JTAG 模式中，PC 和 FPGA 通信的时钟为 JTAG 接口的 TCLK，数据直接从 TDI 进入 FPGA，完成相应功能的配置。

目前，主流的 FPGA 芯片都支持各类常用的主、从配置模式以及 JTAG，以减少配置电路失配性对整体系统的影响。在主配置模式中，FPGA 自己产生时钟，并从外部存储器中加载配置数据，其位宽可以为单比特或者字节；在从模式中，外部的处理器通过同步串行接

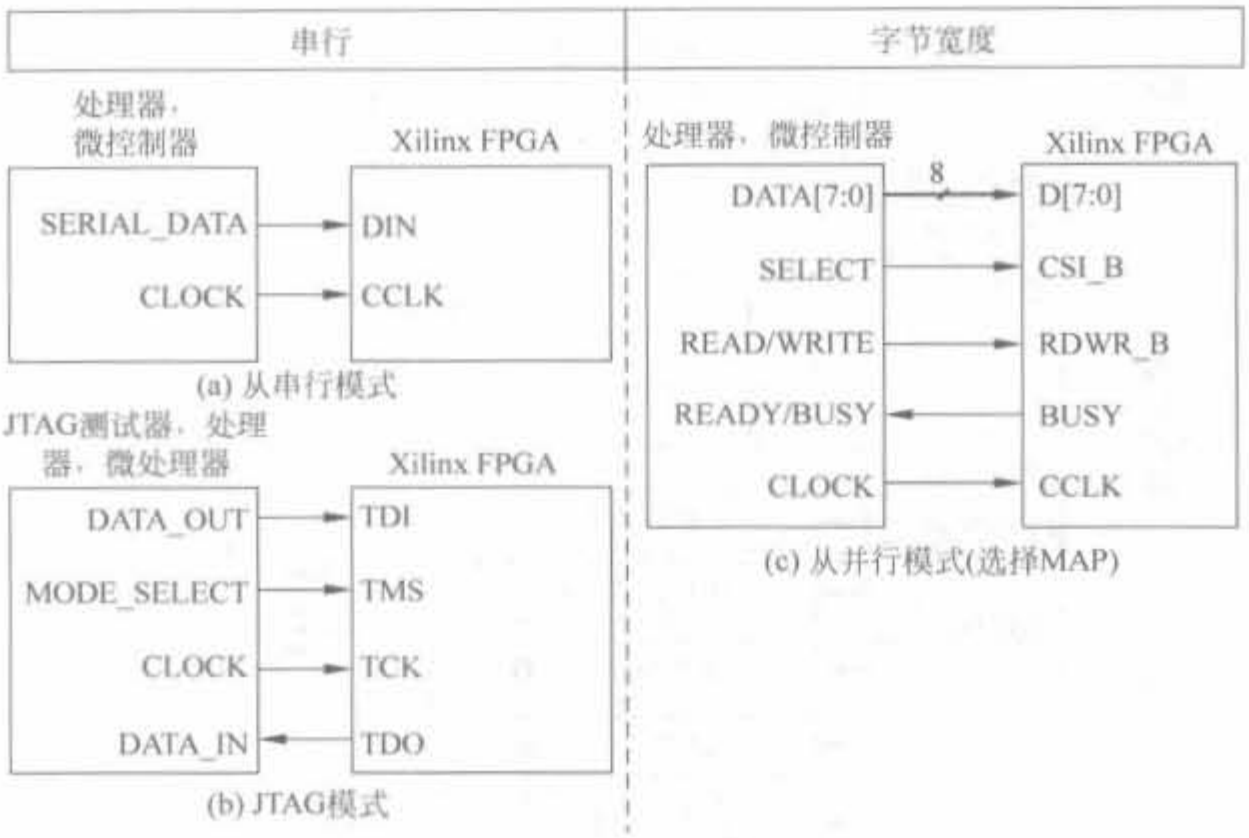


图 5-3 常用的从模式下载方式示意图

口，按照比特或字节宽度将配置数据送入 FPGA 芯片。此外，多片 FPGA 可以通过 JTAG 菊花链的形式共享同一块外部存储器；同样，一片/多片 FPGA 也可以从多片外部存储器中读取配置数据以及用户自定义数据。

5.2 JTAG 电路的原理与设计

5.2.1 JTAG 电路的工作原理

1. JTAG 电路简介

JTAG 的全称是 Joint Test Action Group，即联合测试行动小组。目前，JTAG 已成为一种国际标准测试协议，主要用于各类芯片的内部测试。现在的大多数高级器件(包括 FPGA、MCU、DSP 以及 CPU 等)都支持 JTAG 协议。标准的 JTAG 接口是 4 线接口，即 TMS、TCK、TDI 以及 TDO，分别为模式选择、时钟、数据输入和数据输出信号线。JTAG 电路的功能模块如图 5-4 所示。

JTAG 最初是用来对芯片进行测试的，基本原理是在器件内部定义一个 TAP(Test Access Port，测试访问口)端口，通过专用的 JTAG 测试工具对内部节点进行测试。此外，JTAG 协议允许多个器件通过 JTAG 接口串联在一起，形成一个 JTAG 链，能实现对各个器件分别测试。此外，JTAG 接口还常用于实现 ISP(In-System Programmable，在线编程)，对 Flash 等器件进行编程。JTAG 在线编程的特征也改变了传统生产流程，将以前先对芯片进行预编程再装到板上的工艺简化为：先固定器件到电路板上，再用 JTAG 编程，从而大大加快了工程进度。

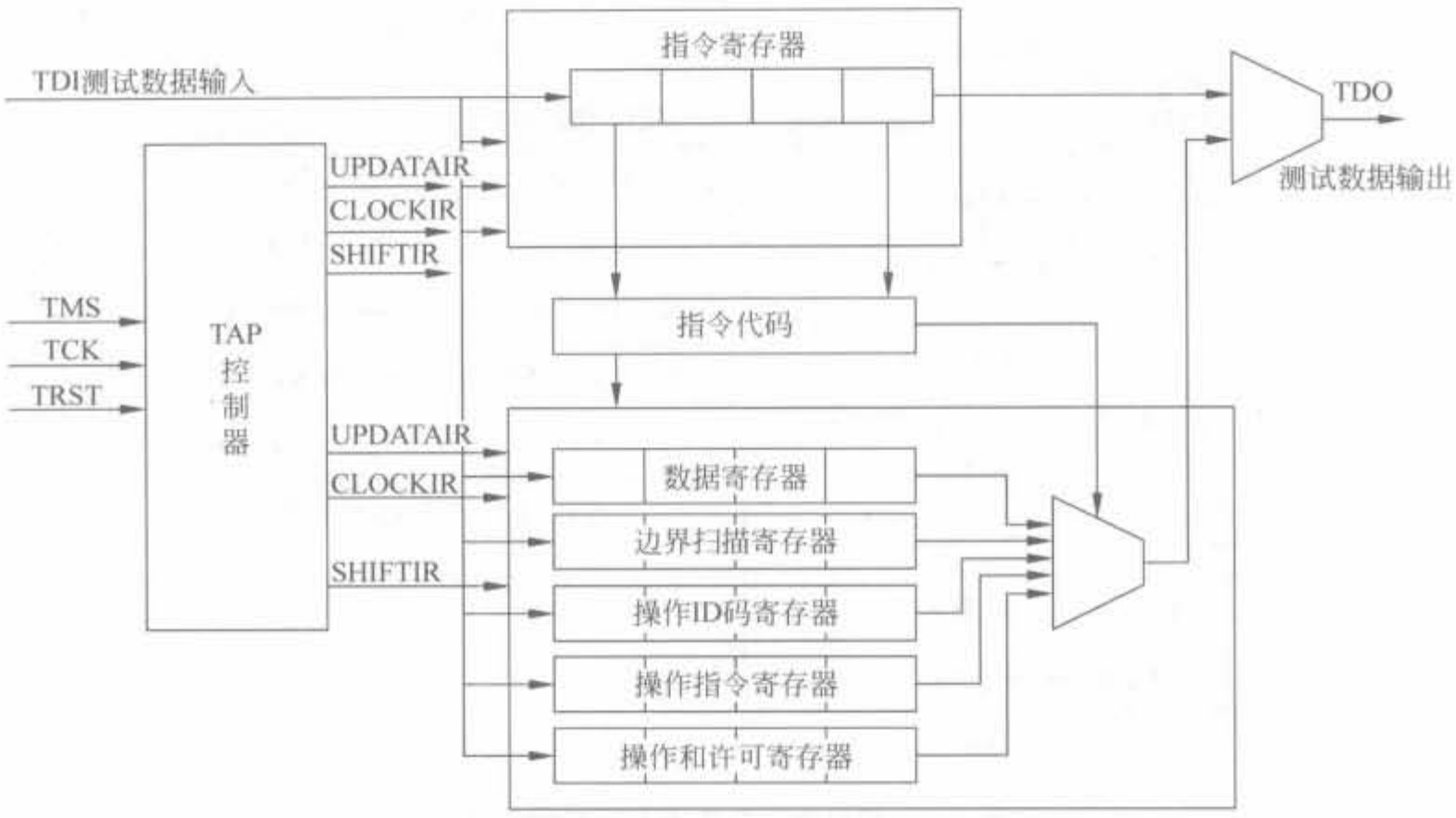


图 5-4　JTAG 电路的内部结构示意图

2. JTAG 边界扫描电路

边界扫描测试(Boundary Sean Test,BST)一般采用 4 线接口(在 5 线接口中,有一条为主复位信号),也可以通过 PC 的 RS-232 接口模拟 BST 的功能。BST 标准接口是用来对电路板进行测试的,可在器件正常工作时捕获功能数据。器件的边界扫描单元能够迫使逻辑追踪管脚信号,或从器件核心逻辑信号中捕获数据,和预期的结果进行比较,根据比较结果给出扫描状态,以提示用户电路设计是否正确。典型边界扫描测试电路的结构如图 5-5 所示。

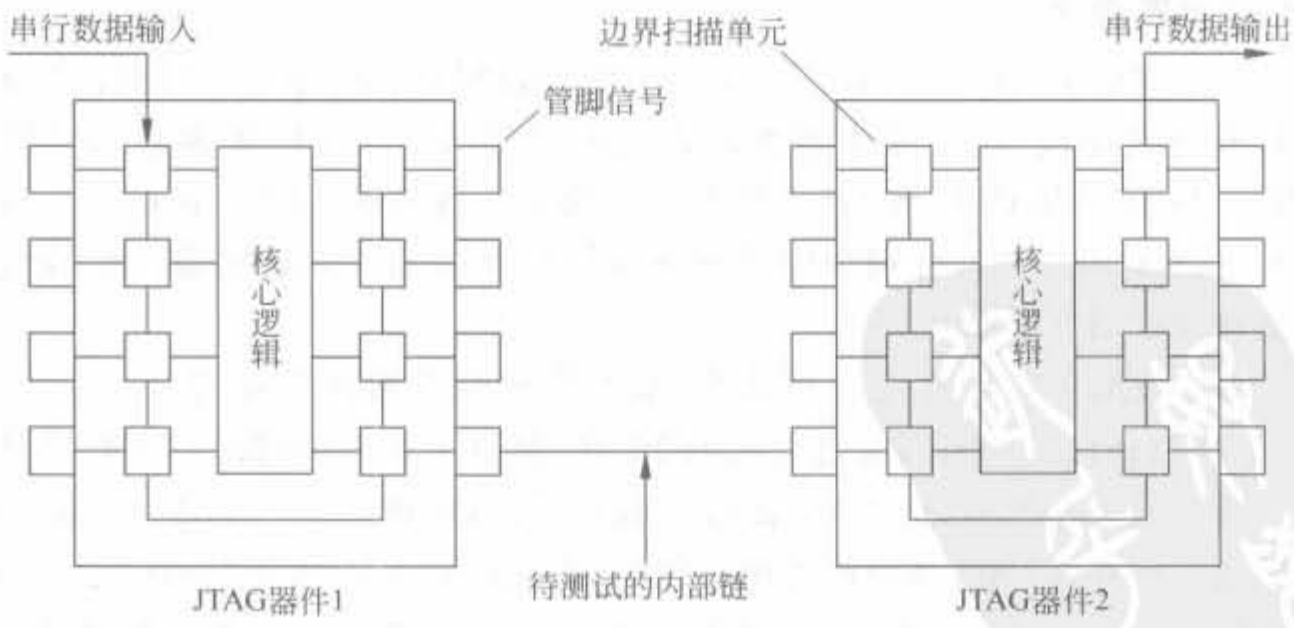


图 5-5　JTAG 链扫描结构示意图

边界扫描测试提供了一个串行扫描路径，遵守 IEEE 规范的器件之间的管脚连接情况。IEEE 1149.1 标准所规定的 BST 结构为：当器件工作在 JTAG BST 模式时，使用 4 个专用的 I/O 管脚和一个可选管脚 TRST 作为 JTAG 管脚。这 4 个专用 I/O 管脚为 TDI、TDO、TMS 和 TCK。所有 JTAG 管脚的核心功能如表 5-2 所示。

表 5-2 JTAG 管脚说明

管脚名	名　　称	功能描述
TDI	测试数据输入管脚	JTAG 指令和测试编程数据的输入管脚，数据在 TCK 信号的上升沿时刻读入
TDO	测试数据输出管脚	JTAG 指令和测试编程数据的串行输出管脚，数据在 TCK 信号的下降沿时刻读出。如果数据没有输出，则处于三态
TMS	测试模式选择管脚	该数据管脚是控制信号，它决定了 TAP 控制器的转换。TMS 信号必须在 TCK 上升沿之前建立，在用户状态下，TMS 信号应是高电平
TCK	测试时钟输入管脚	JTAG 链路的时钟信号，直接输入到边界扫描电路。所有操作都在其上升沿或下降沿时刻发生
TRST	测试复位输入管脚	用于异步初始化或复位 JTAG 边界扫描电路，低电平有效

3. JTAG 电路时序

JTAG 电路的时序如图 5-6 所示，所有基于 JTAG 的操作都必须同步于 JTAG 时钟信号 TCK。在 TCK 的上升沿读取或输出有效数据，有严格的建立、保持时间要求，因此一般情况下，JTAG 的时钟不会太高。

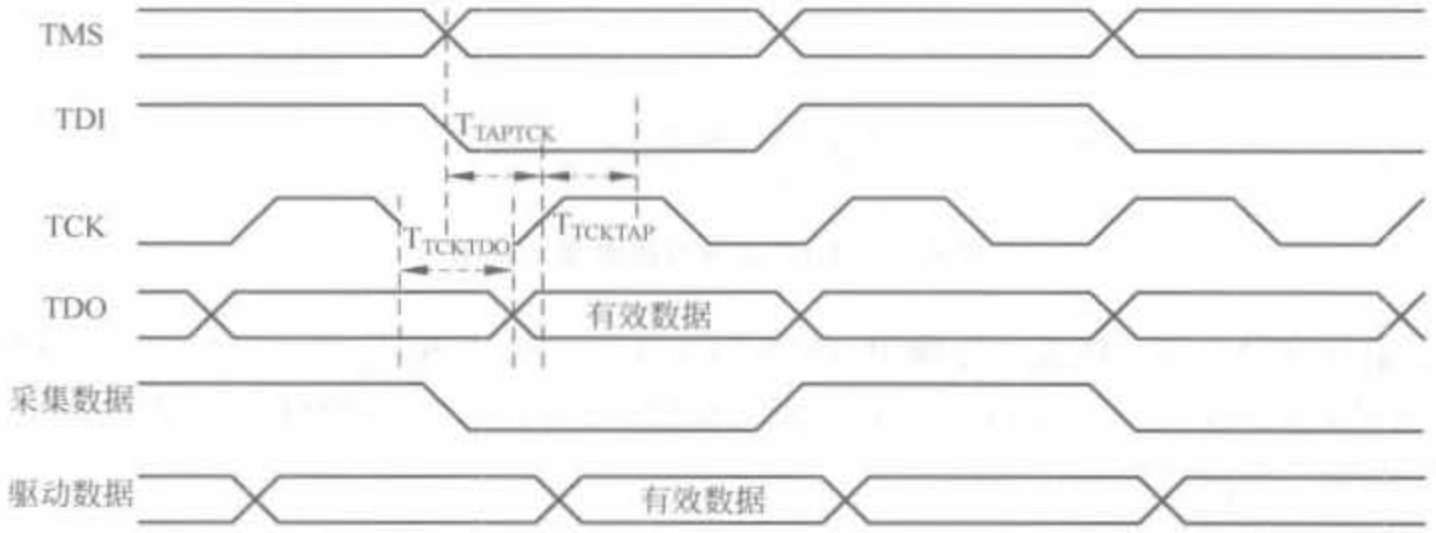


图 5-6 JTAG 电路的时序关系示意图

4. FPGA 芯片中 JTAG 扫描电路的工作流程

JTAG 边界扫描测试由测试访问端口的控制器管理，只要 FPGA 上电后电压正确，且 JTAG 链路完整，则 JTAG 电路可立即正常工作，清空 JTAG 配置寄存器而等待外界响应，整体流程如图 5-7 所示。

TMS、TRST 和 TCK 管脚管理 TAP 控制器的操作，TDI 和 TDO 为数据寄存器提供串行通道。TDI 也为指令寄存器提供数据，然后为数据寄存器产生控制逻辑。对于选择寄存器、装载数据、检测和将结果移出的控制信号，由测试时钟(TCK)和测试模式(TMS)选择两

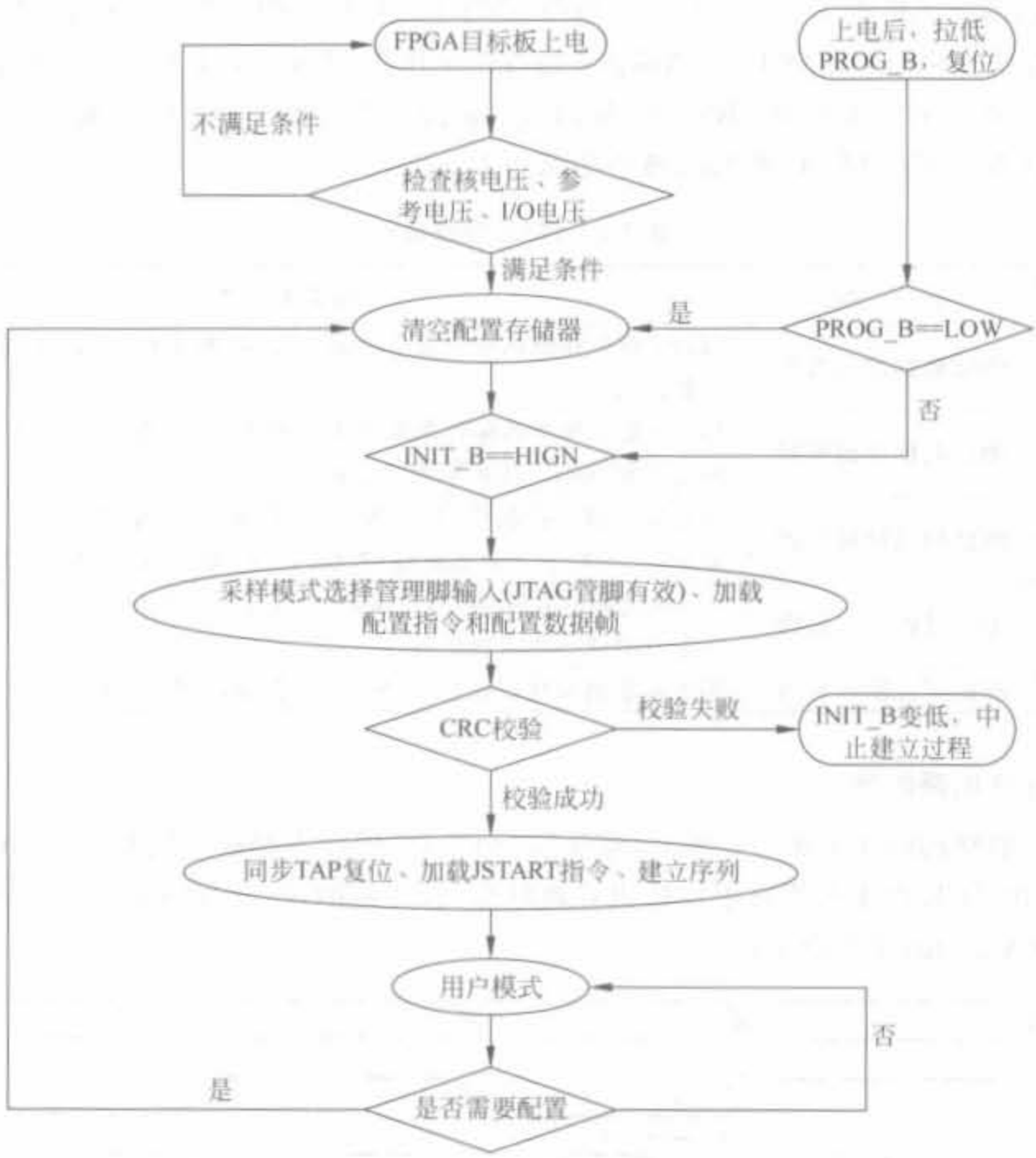


图 5-7　JTAG 边界扫描流程示意图

个控制信号决定。在 4 线接口标准中，利用 TDI、TDO、TCK 和 TMS 4 个信号，它们合成为 TAP 测试处理端口(Test Access Port)，测试复位信号(TRST，一般以低电平有效)一般作为可选的第 5 个端口信号。

5.2.2　Xilinx JTAG 下载线

下载线一端以 JTAG 的方式和 FPGA/PROM 芯片相连，另一端则通过 USB/并口和计算机相连，为设计人员提供了由 PC 配置 FPGA/PROM 芯片的数据链路。本节介绍目前常用的 Xilinx 下载线，以及简易下载线的制作方法。

1. Xilinx 下载线介绍

根据下载线和 PC 连接方式的不同，可以将其分为 USB 下载线和并口下载线两大类。USB 下载线速度快，稳定度高，当然价格也比较昂贵。目前，Xilinx 公司提供的 USB 下载线的价格为 149 美元。并口下载线根据下载速度的不同，可分为 Parallel Cable Ⅳ(简称为

PC4)和 Parallel Cable Ⅲ两类(简称为 PC3)。其中,PC4 适用于 Xilinx 公司的所有芯片,速度比 PC3 快 8 倍,价格大约为 USB 下载线的 1/3;PC3 采用简单的 EPP 模式,透过式实现,成本低廉,但下载速度缓慢,且不具备配置电压自适应的功能,已经不能用于 Xilinx 公司新型 FPGA 的开发,存在一定的应用局限性。

无论哪种下载线,在 FPGA 端都具有标准的 4 个 JTAG 接口、电源管脚以及地(VCC、GND、TCK、TMS、TDI 以及 TDO),共 6 个信号端口,也被称为 JTAG 连接器。也有一种常见的 10 脚 JTAG 连接器,其中多了 1 个 GND 信号以及 3 个悬空信号(NC)。

在实际工程中,有一条性能稳定的下载线,不仅能避免配置错误(如利用 PC3 并口下载线配置 Spartan-3E 等最新系列芯片时会经常出现错误),还能提高配置的成功率并缩短配置时间(在实际中,完整正确的配置电路也不能保证每次配置都成功)。下面对 Xilinx 各种下载线的特点和性能进行简要总结,如表 5-3 所示,供读者参考。

表 5-3 Xilinx 下载线性能的简要总结

	USB 下载线	Parallel Cable Ⅳ下载线	Parallel Cable Ⅲ下载线
器件编号	HW-USB-G	HW-PC4	HW-PC3
连接目标	USB 1.1 或 USB 2.0 接口	PC 并口(DB25)	PC 并口(DB25)
支持的 I/O 电压	1.5V、1.8V、2.5B、3.3V、5V	1.5V、1.8V、2.5B、3.3V、5V	3.3V、5V
工作电压需求	USB 总线供电	总线供电或使用外部电源	总线供电
硬件配置模式	支持 Xilinx 全系列配置模式	支持 Xilinx 全系列配置模式	支持 Xilinx 全系列配置模式
支持的操作系统	Windows 2000/XP/Vista Red Hat Enterprise Linux Workstation 3.0&4.0 SUSE Linux 9&10		
支持的芯片	所有的 Xilinx FPGA、CPLD、PROM		
目标最大时钟速率	24Mb/s	5Mb/s	
是否支持在系统编程	支持	支持	支持

2. PC3 并行下载的电路原理图

首先对 PC 的打印接口进行简单介绍。PC 上的打印接口共有 25 根连线(一般也称为 DB25),如图 5-8 所示。其中,18～25 都是地线,因此实际共有 17 根线,分成 3 类:8 根数据线,可进行数据输出;5 根状态线,作为输入;4 根控制线,作为输出。这 3 组线分别由打印口的 3 个寄存控制器控制,即数据口、状态口和控制口。只要对这 3 个寄存器读或写,就可以输入或输出数据。并口下载线主要完成 PC 并口和 FPGA 芯片 JTAG 接口之间的数据适配。Xilinx 公司 PC3 并口下载电缆的原理图是公开的,结构简单,成本简单,不超过 10 元,有兴趣的读者完全可以自己动手制作,在电子市场上则一般需要数十元。

下载电缆的接口电路比较简单,简易连接关系如图 5-8 所示,其制作时间非常短,半天时间即可。接口电路只完成了电平转换和简单的译码工作,不涉及信号时序的改变。

其中,74HC125 为 Philips 公司生产的 4 输入 3 态驱动器,用来增强信号强度;LED 用来给出下载信息指示。详细的元件清单如表 5-4 所示。

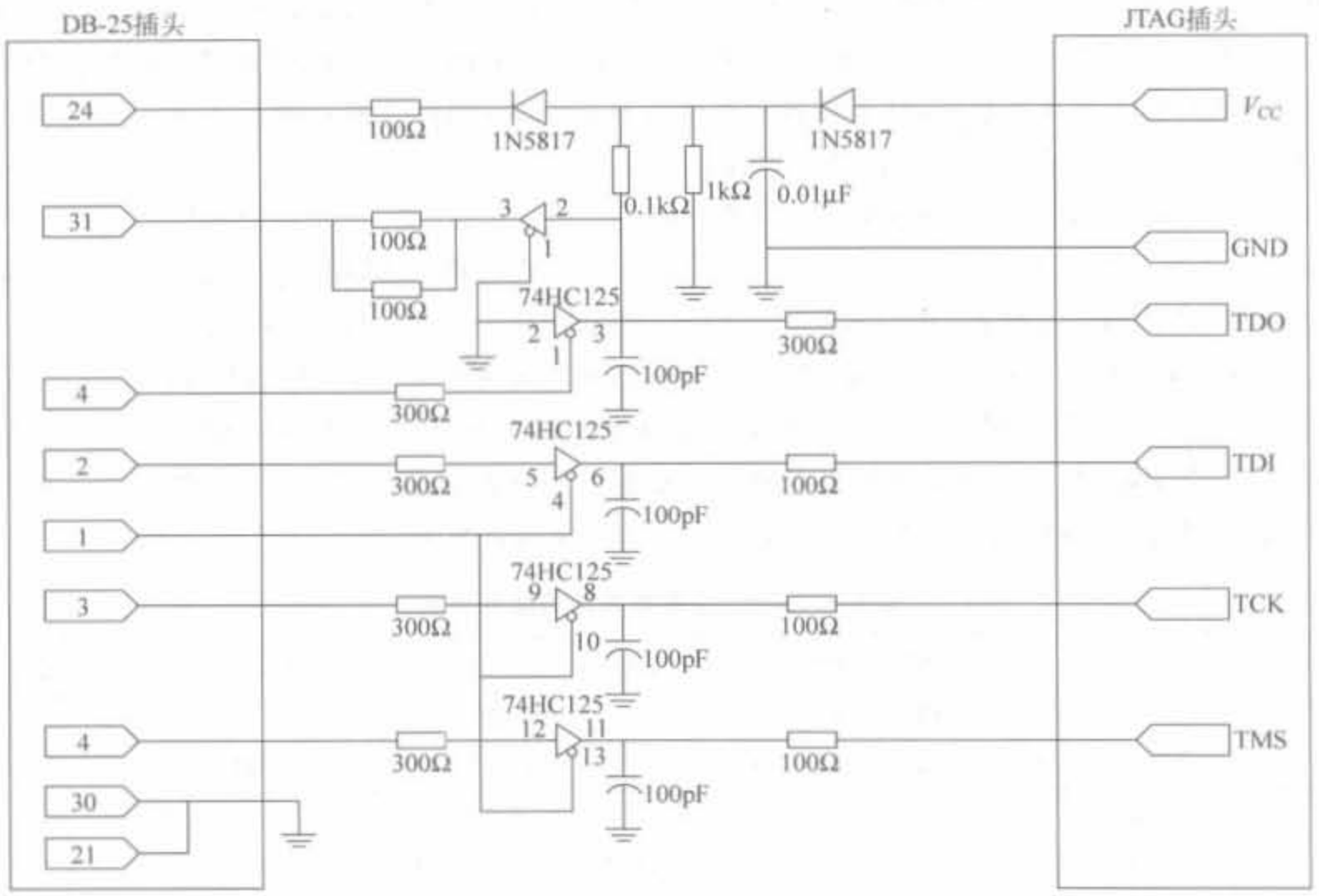


图 5-8　Xilinx 下载线接口电路图

表 5-4　并口下载线的元件清单

元件类型	元件型号	数量(个)	元件类型	元件型号	数量(个)
电阻	5.1kΩ	1	电容	100pF	4
电阻	1kΩ	1	肖特基二极管	1N5817	2
电阻	390Ω	1	3 态缓冲器	74HC125	2
电阻	300Ω	4	LED		1
电阻	100Ω	6	25 针插座		1
电阻	47Ω	1	6 针插头		1

下载线由电路板供电，即由 JTAG 连接器的 V_{cc} 提供，因此和电路板的距离不能太长，一般为 20cm 左右；和并口连接端可以适当延长，但也不应当超过 1m，一般选择 50cm 左右，否则容易发生下载错误。此外，肖特基二极管应选取低压降的管子，以保证下载线正常工作。

5.3　FPGA 的常用配置电路

Xilinx FPGA 的常用配置模式有 5 类：主串模式、从串模式、Select MAP 模式、Desktop 配置和直接 SPI 配置。在从串配置中，FPGA 接收来自于外部 PROM 或其他器件的配置比特数据，在 FPGA 产生的时钟 CCLK 的作用下完成配置，多个 FPGA 可以形成菊花链，从同

一配置源中获取数据。Select MAP 模式中，配置数据是并行的，它是速度最快的配置模式。SPI 配置主要在具有 SPI 接口的 Flash 电路中使用。下面以 Spartan-3E 系列芯片为例，给出各种模式的配置电路。

5.3.1 主串模式——最常用的 FPGA 配置模式

1. 配置单片 FPGA

在主串模式下，由 FPGA 的 CCLK 管脚给 PROM 提供工作时钟，相应的 PROM 在 CCLK 的上升沿将数据从 D0 管脚送到 FPGA 的 DIN 管脚。无论 PROM 芯片类型如何(即使其支持并行配置)，都只利用其串行配置功能。Spartan-3E 系列 FPGA 的单片主串配置电路如图 5-9 所示。主串模式是 Xilinx 公司提供的各种配置方式中最简单，也最常用的方式，基本上所有的可编程芯片都支持主串模式。

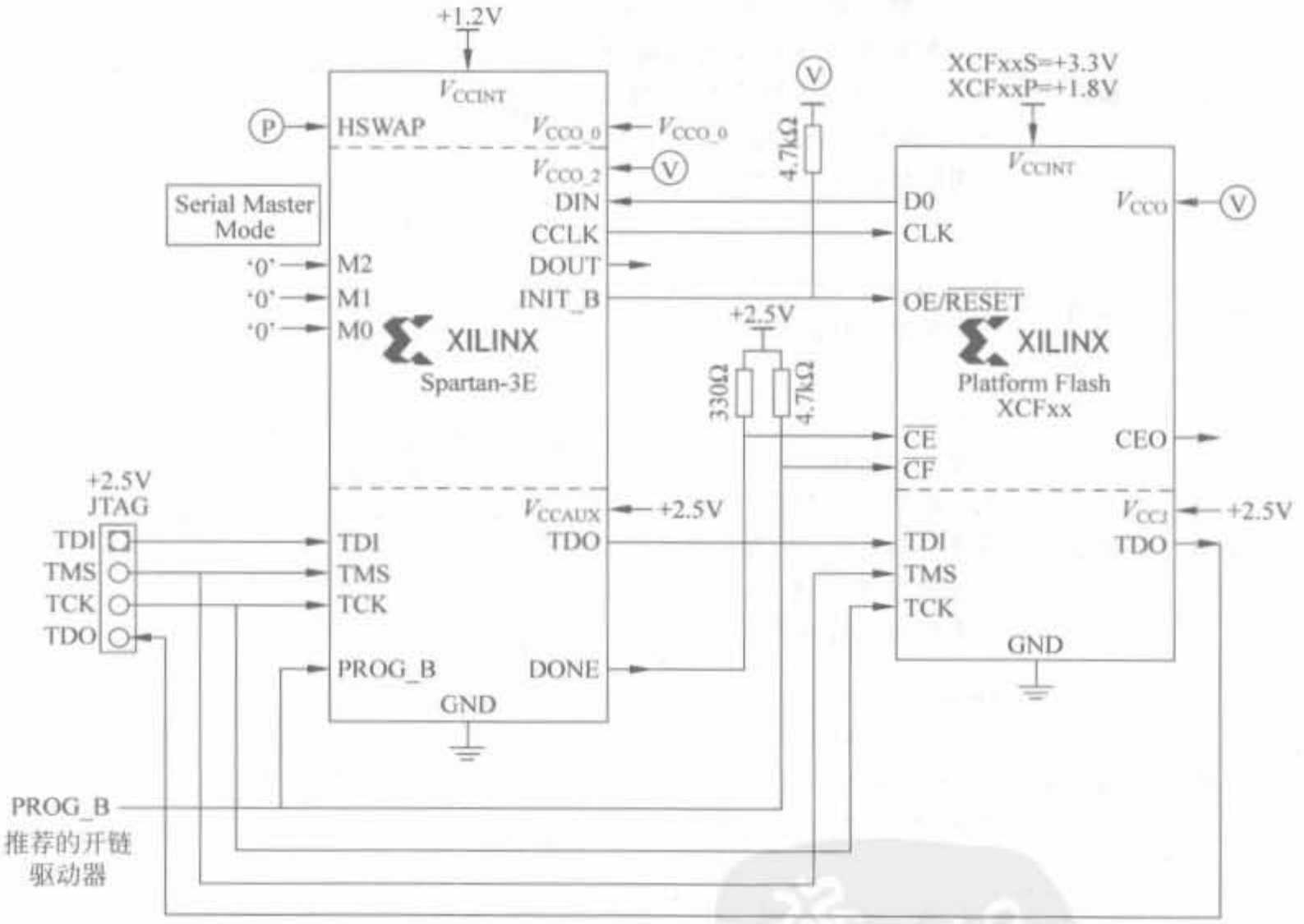


图 5-9 Spartan-3E 主串模式配置电路

电平拉低或拉高有啥区别？分别有啥用途？

1) 信号管脚说明

需要注意 3 类管脚的连接方式：首先，在配置过程中或者 INIT_B 变高时，模式选择管脚 M[2:0]必须设置为全 0，当 FPGA 的输出管脚 DONE 变为高电平后，模式配置管脚可以作为普通 I/O 管脚使用。其次，HSWAP 管脚的输入电平在器件配置阶段必须保持不变，可以将其电平拉低，使能 FPGA 所有 I/O 管脚的上拉电阻；也可以拉高电平，旁路 FPGA 所有 I/O 管脚的上拉电阻，当 FPGA 配置完毕，输出信号 DONE 变高后，可以作为普通 I/O

管脚使用。最后，FPGA 的 DOUT 管脚仅在多芯片配置时有效，在单芯片配置中悬空。

(1) 图 5-9 中，FPGA 芯片各个管脚的功能和配置的说明如表 5-5 所示。

表 5-5 主串模式下 FPGA 配置管脚说明

管脚名	信号方向	功能描述	配置时功能	配置后功能
HSWAP	输入	I/O 管脚上拉电阻控制信号。需要在配置过程中保持电平稳定，1：没有上拉电阻，0：使能上拉电阻，将每个 I/O 管脚上拉到其相应的 V_{cco}	在配置器件保持稳定电平驱动	用户 I/O
M[2:0]	输入	配置模式选择信号。主串模式为全 0	M2=0，M2=1，M0=0，在 INIY_B 变高后被采样	用户 I/O
DIN	输入	串行数据输入信号。用于串行接收来自于 PROM 的配置数据	从 PROM 芯片 DOUT0 管脚接收串行数据	用户 I/O
CCLK	输出	配置时钟信号。由 FPGA 产生，用于控制配置比特流的传输速率。对信号完整性有较高的影响，电路板上，CCLK 信号走线尽可能短且外部连接少	提供 PROM 器件的驱动时钟	用户 I/O
DOUT	输出	串行数据输出信号，仅在多片 FPGA 配置时有效	主动驱动，在单片 FPGA 应用中无效。在多片菊花链中，连接到下一片 FPGA 的 DIN 管脚，用于级联 FPGA	用户 I/O
INIT_B	开链输出	芯片初始化指示信号，低电平有效。在 FPGA 配置开始阶段，芯片清空初始化时变为低电平；清空过程结束后变为高电平。需要外接 4.7kΩ 电阻上拉到 V_{cco_2}	和 PROM 芯片的 OE/RESET 管脚相连。在配置开始阶段，清空 PROM 地址计数器。如果在配置中检测到 CRC 错误，会将其拉低	用户 I/O。如果不用，需要拉高
DONE	开链输出	FPGA 配置状态指示信号。当 FPGA 处于配置阶段时，为低电平；配置完成后变为高电平。需要外接 330Ω 上拉电阻上拉到 2.5V	连接到 PROM 芯片的 CE 管脚，在配置中使能 PROM，配置完成后，关闭 PROM	通过外部电阻上拉
PROG_B	输入	重配置 FPGA 信号。低电平有效，当其被拉低超过 500ns 时，会强制 FPGA 进入重配置阶段；当 PROG_B 信号变高时，会清空 FPGA 配置存储器，并将 DONE、INIT_B 拉低。需要通过 4.7kΩ 电阻上拉到 2.5V。如果工作在 3.3V 模式下，需要串接电阻，以限制电流大小	连接到 PROM 芯片的 CF 管脚，在配置过程中需要拉高来启动配置流程	拉低。如果拉高，意味着重配置

（2）必须要掌握从设备 PROM 的管脚信号。下面对图 5-9 中 PROM 芯片各个管脚的功能和配置进行简单介绍，如表 5-6 所示。

表 5-6 主串模式下 PROM 配置管脚说明

管脚名	信号方向	管脚描述
D0	输出	串行模式下，FPGA 配置数据的输出管脚。配置完成后，它处于高阻状态
CLK	输入	配置时钟输入管脚。在 CE 信号为低电平，OE/RESET 为高电平时，在 CLK 的每一个上升沿，将 PROM 内部地址计数器加 1
OE/RESET	输入	使能/重启信号输出管脚。当其为低电平时，PROM 内部地址计数器保持不变，且数据输出管脚 D0 处于高阻状态；当其为高电平时，在 CLK 的上升沿，内部地址计数器加 1
CE	输入	芯片片选输入信号，低电平有效，芯片开始工作；当其为高电平时，芯片进入休眠状态，并将地址计数器清空，数据输出端口 D0 处于高阻状态
CF	输入	配置指示信号，低电平有效。当其为低电平时，在不断电的情况下重新配置 FPGA，并将内部地址计数器清零
CEO	输出	芯片输出使能信号。一般用于多片 PROM 的连接，和下一片 PROM 的 CE 输出管脚相连。当 CE 为低电平，OE/RESET 信号为高电平，且内部地址计数器超越其有效范围时，输出低电平。当 OE/RESET 变低，或 CE 信号变高时，CEO 重新置高电平
V_{CCINT}	输入	PROM 工作核电压。对于 XCF××S，其值为 3.3V；对于 XCF××P，其值为 1.8V
V_{CCO}	输入	PROM 芯片的管脚电平电压管脚，输入信号，可以接 3.3V、2.5V 以及 1.8V
V_{CCJ}	输入	PROM 芯片的 JTAG 接口，输入管脚，为 TCK、TDO、TDI 以及 TMS 管脚提供电平，可以接 3.3V、2.5V 以及 1.8V

2）配置电路的关键点

主串配置电路最关键的 3 点就是 JTAG 链的完整性、电源电压的设置以及 CCLK 信号的考虑。只要这 3 步中的任何一个环节出现问题，都不能正确配置 PROM 芯片。

（1）JTAG 链的完整性

FPGA 和 PROM 芯片都有自身的 JTAG 接口电路。JTAG 链完整性，指的是将 JTAG 连接器、FPGA、PROM 的 TMS、TCK 连在一起，保证从 JTAG 连接器 TDI 到其 TDO 之间，形成 JTAG 连接器的“TDI→（TDI～TDO）→（TDI～TDO）→JTAG 连接器 TDO”的闭合回路。其中，（TDI～TDO）为 FPGA 或者 PROM 芯片自身的一对输入、输出管脚。在图 5-9 中，配置电路的 JTAG 链从连接器的 TDI 到 FPGA 的 TDI，再从 FPGA 的 TDO 到 PROM 的 TDI，最后从 PROM 的 TDO 到连接器的 TDO，形成了完整的 JTAG 链。FPGA 芯片被称为链首芯片。也可以根据需要调换 FPGA 和 PROM 的位置，使 PROM 成为链首芯片。

（2）电源适配性

如图 5-10 所示，由于 FPGA 和 PROM 要完成数据通信，二者的接口电平必须一致，即 FPGA 相应分组的管脚电压 V_{CCO_2} 必须和 PROM V_{CCO} 的输入电压大小一致，且理想值为

2.5V，这是由于 FPGA 的 PROG_B 和 DONE 管脚由 2.5V 的 V_{CCAUX} 供电。此外，由于 JTAG 连接器的电压也由 2.5V 的 V_{CCAUX} 提供，因此 PROM 的 V_{CCJ} 也必须为 2.5V。因此，如果接口电压和参考电压不同，在配置阶段需要将相应分组的管脚电压和参考电压设置为一致；在配置完成后，再将其切换到用户所需的工作电压。当然，FPGA 和 PROM 也可以自适应 3.3V 的 I/O 电平以及 JTAG 电平，但需要进行一定的改动，即添加几个外部限流电阻，如图 5-10 所示。在主串模式下，XCF××S 系列 PROM 的核电压必须为 3.3V，XCF××P 系列 PROM 的核电压必须为 1.8V。

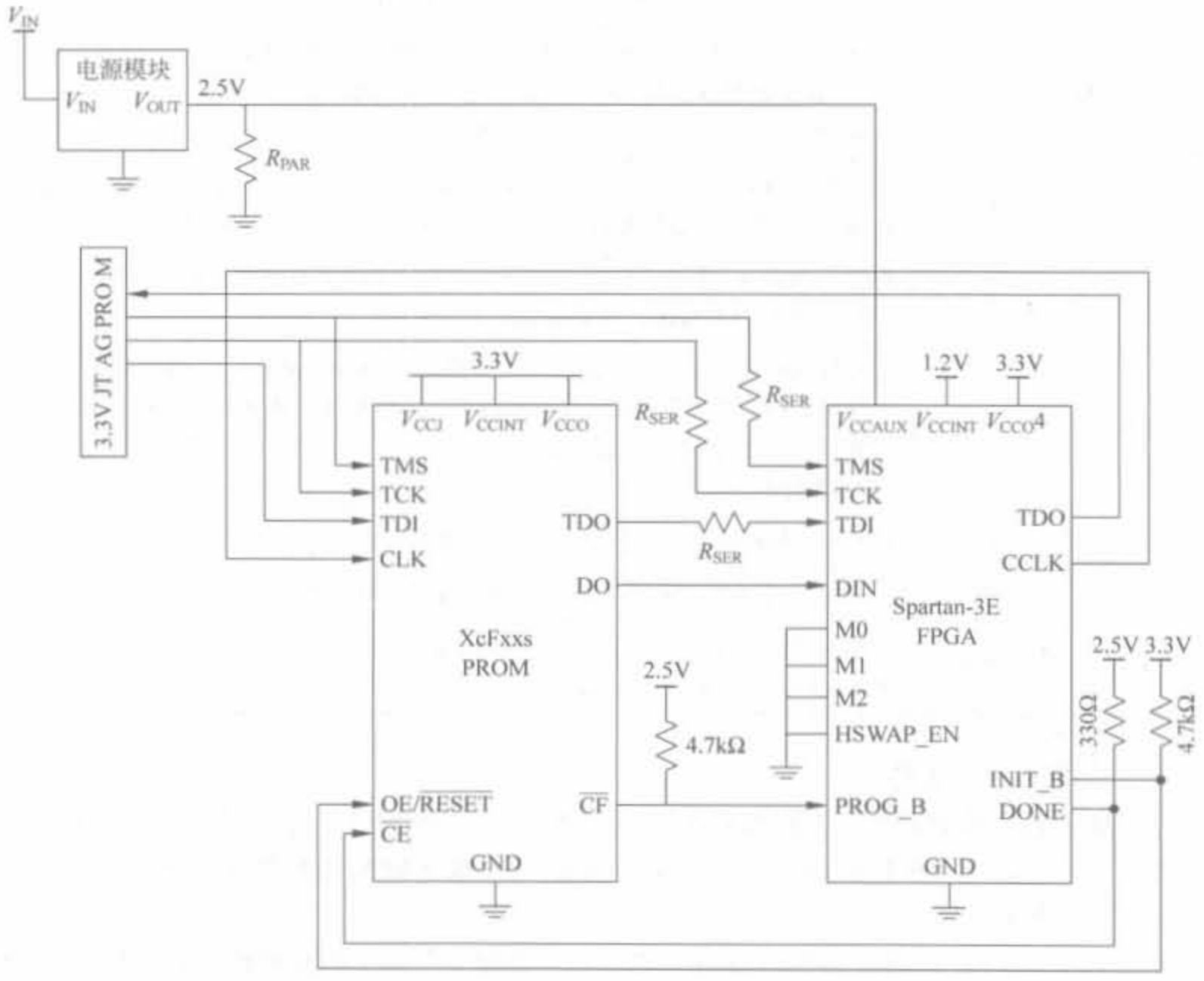


图 5-10　3.3V 的 JTAG 配置电路示意图

图 5-10 中的 R_{SER} 和 R_{PAR} 这两个电阻要特别注意。首先，$R_{SER}=68\Omega$ 将流入每个输入的电流限制到 9.5mA；其次，$N=3$，3 个输入的二极管导通，

$$R_{PAR}=V_{CCAUX}\ \text{min}/\ \text{NIIN}=2.375\text{V}/(3\times 9.5\text{mA})$$
$$=83\Omega \text{ 或 } 82\Omega(\text{与标准值误差小于 } 5\% \text{的电阻})$$

(3) CCLK 的信号完整性

CCLK 信号是 JTAG 配置数据传输的时钟信号，其信号完整性非常关键。FPGA 配置电路刚开始以最低时钟工作，如果没有特别指定，将逐渐提高频率。CCLK 信号是由 FPGA 内部产生的，对于不同的芯片和电平，其最大值如表 5-7 所示。

表 5-7 不同 PROM 芯片的最大配置时钟频率

PROM 芯片型号	I/O 电压 (FPGA V_{CCO2} 或 PROM V_{CCO})	Spartan-3E FPGA 最大配置时钟频率(MHz)
XCF01S	3.3V 或 2.5V	25
XCF02S XCF04S	1.8V	12
XCF08P XCF16P XCF32P	3.3V、2.5V 或 1.8V	25

3) 主串配置电路工作流程

一般的 FPGA 芯片都有两个配置触发事件：上电复位以及软件复位。不同配置模式的工作流程基本是一致的，下面对整个过程进行详细说明。

(1) 普通配置过程

当 FPGA 上电后，如果核电压、参考电压以及 I/O 电压正确，则进入配置模式。数据首先以 TCK 的速度通过 JTAG 连接器的 TDI 管脚进入 FPGA 芯片的 TDI 管脚，再以同样的速率从 FPGA 的 TDO 管脚将配置数据送入 PROM 芯片的 TDI 管脚，此时 PROM 通过其 TDO 向 JTAG 连接器的 TDO 环回数据，构成完整的 JTAG 链；又由于 FPGA 芯片 DONE 信号为低电平(片选 PROM 芯片)、INIT_B 输出电平为高(使能 PROM 数据输出管脚)，PROM 通过 DO 以 CCLK 的速率将配置数据送给 FPGA。第三，FPGA 开始接收配置数据，并完成 CRC 校验。若 CRC 校验通过，DONE 信号管脚输出高电平；若 CRC 校验失败，DONE 信号为低电平，配置过程失败，但此时 FPGA 不给出任何指示，这是由于需要在 DONE 管脚上添加 LED 以输出提示信号。最后，PROM 由于 CE 管脚输入为高电平，关闭数据输出管脚，清空地址计数器，进入休眠状态，配置结束。

(2) 复位配置过程

当 PROG_B 处于低电平超过 500ns 时，会强制 FPGA 进入重配置阶段；当 PROG_B 信号变高时，会清空 FPGA 配置存储器，并将 DONE、INIT_B 拉低。由于 DONE 信号和 PROM 芯片的 CE 信号相连，PROM 片选有效。CF 信号有效，将 PROM 内部地址计数器清零。当清空 FPGA 配置存储器后，OE/RESET 变为高电平，地址累加器开始在 CLK 的上升沿加 1。FPGA 配置结束后，DONE 信号管脚输出高电平，PROM 关闭数据输出管脚，清空地址计数器，进入休眠状态。复位配置的过程如图 5-11 所示。

图 5-11 复位后 FPGA 配置阶段示意图

2. 配置多片 FPGA

多片 FPGA 的配置电路和单片的类似，但是多片 FPGA 之间有主(Master)、从(Slave)之分，且需要选择不同的配置模式。两片 Spartan-3E 系列 FPGA 的典型配置电路如图 5-12 所示，两片 FPGA 存在主、从地位之分。

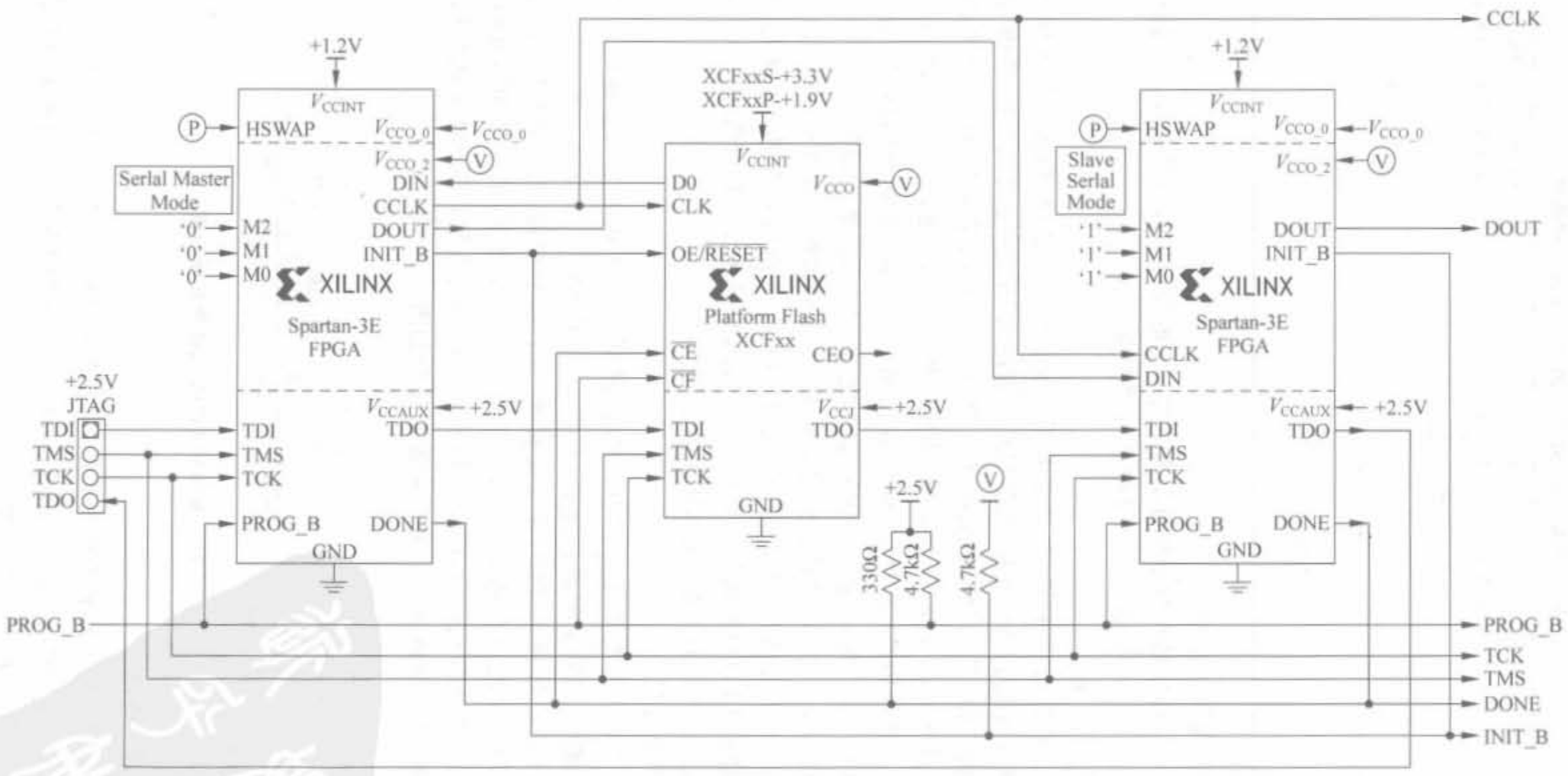


图 5-12　主从模式下两片 FPGA 的配置电路

如果系统中有更多的FPGA芯片，只需要在后面继续添加即可，即从链首FPGA获得CCLK，将芯片TCK、TMS和JTAG连接器的TCK、TMS连接在一起，最后把上一级FPGA的TDO连接到本地TDI，并将本地TDO和JTAG连接器的TDO连在一起，构成完整的JTAG链。当链首FPGA完成配置后，将利用其DOUT管脚在CCLK的下降沿为后续芯片传送配置数据，而其自身在CCLK的上升沿从PROM读取配置数据。注意，除了链首FPGA的模式选择信号M[2:0]=3'b000外，其余FPGA的模式选择信号M[2:0]=3'b111。

如果多片相同的FPGA配置相同的数据，可以采用图5-13所示的配置电路。

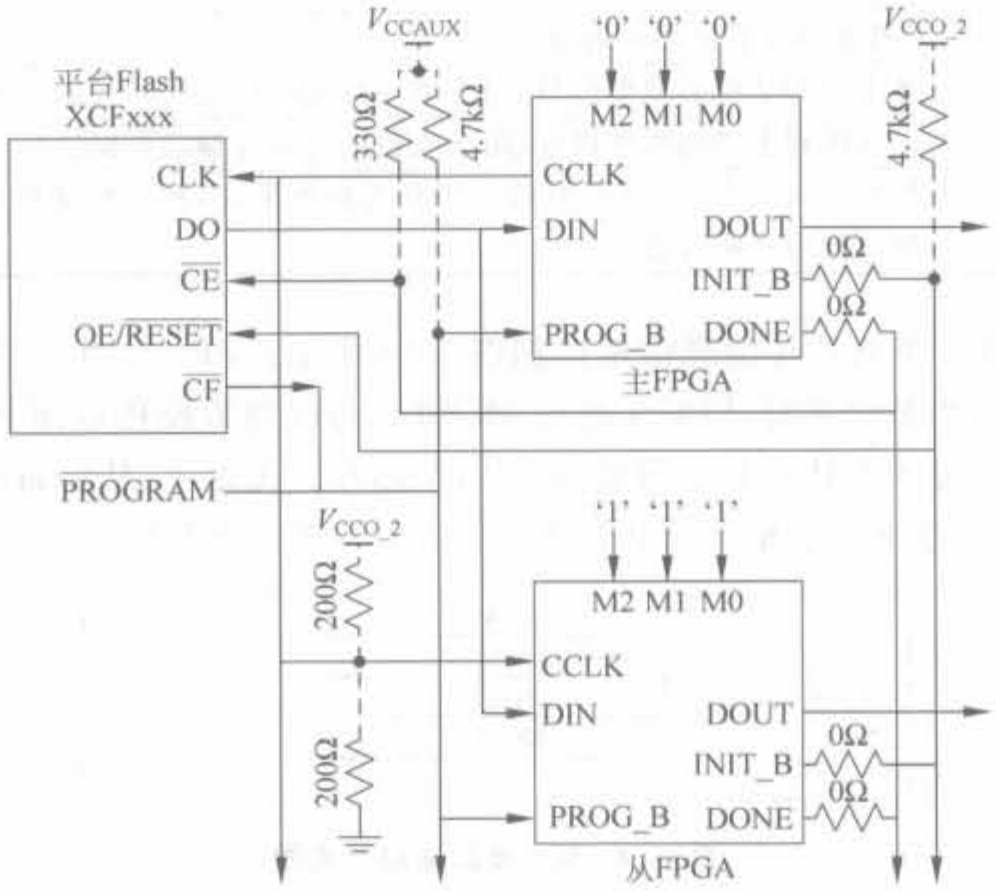


图5-13 配置数据相同的多片相同FPGA的配置电路

在实际中，可以修改图5-13所示电路，使FPGA在配置完成后仍然能通过主串接口以及PROM接口和其通信。这就允许用户将一些非易失的应用数据存放在PROM中，例如以太网的MAC地址或嵌入式处理器MICROBLAZE的应用数据等，这类PROM二次利用的原理和方法将在5.5节介绍。

5.3.2 SPI串行Flash配置模式

1. SPI串行配置介绍

串行Flash的特点是占用管脚比较少，作为系统的数据存储非常合适，一般都是采用串行外设接口(SPI总线接口)。Flash存储器与EEPROM根本不同的特征就是EEPROM可以按字节进行数据的改写，而Flash只能先擦除一个区间，然后改写其内容。一般情况下，这个擦除区间叫做扇区(Sector)，也有部分厂家引入了页面(Page)的概念。选择Flash产品时，最小擦除区间是比较重要的指标。在写入Flash时，如果写入的数据不能正好是一个最小擦除区间的尺寸，就需要把整个区间的数据全部保存在另外一个存储空间，擦除这个空间，才能重新对这个区间改写。大多数Flash工艺更容易实现较大的擦除区间，因此较小擦

除区间的 Flash 其价格一般稍贵。此外，SPI 是标准的 4 线同步串行双向总线，提供控制器和外设之间的串行通信数据链路，它广泛应用于嵌入式设备中。

Xilinx 公司的新款 FPGA 都支持 SPI 接口。SPI 总线通过 4 根信号线来完成主、从之间的通信，典型的 SPI 系统中常包含一个主设备以及至少一个从设备。在 FPGA 应用场合中，FPGA 芯片为主设备，SPI 串行 Flash 为从设备。4 个 SPI 接口信号的名称和功能如表 5-8 所示。

表 5-8　SPI 接口信号列表

SPI 接口信号名	信号功能描述
SCLK	SPI 接口工作的串行时钟
MOSI	从主机到从机的数据信号，用于将主机的执行代码和数据发送到从机上
MISO	从从机到主机的数据信号，用于收集从机所传输的数据信号
SS_n	从机片选信号，低电平有效。当其为高电平时，放弃对从机的控制，并将 MISO 端口置为高阻状态

一个主芯片和一个从芯片的通信接口如图 5-14 所示。FPGA 通过 SCLK 控制双方通信的时序，在 SS_n 为低电平时，FPGA 通过 MOSI 信号线将数据传送到 Flash，在同一个时钟周期中，Flash 通过 SOMI 将数据传输到 FPGA 芯片。无论主、从设备，数据都是在时钟电平跳转时输出，并在下一个相反的电平跳转沿送入另外一个芯片。

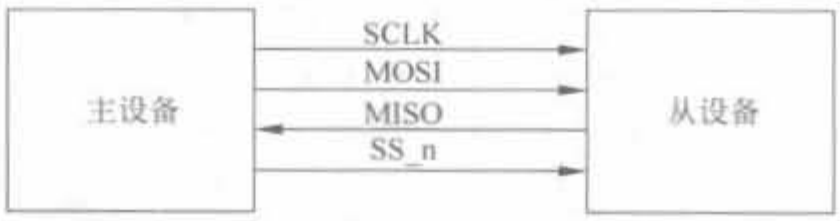


图 5-14　SPI 接口连接示意图

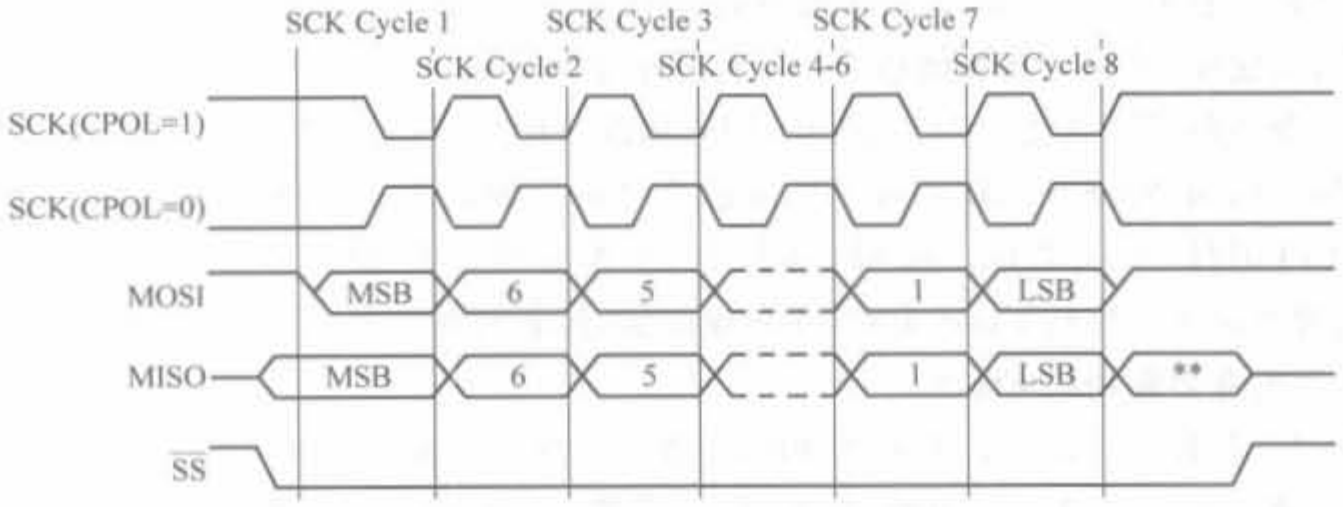
SCLK 信号支持不同的速率，一般常采用 20MHz。通过 SPI 接口中的 CPOL 和 CPHA 这两个比特定义了 4 种通信时序。其中，CPOL 信号定义了 SCLK 的空闲状态，当 CPOL 为低电平时，SCLK 的低电平为空闲状态，否则其空闲状态为高电平。CPHA 定义了数据有效的上升沿位置，当其为低电平时，数据在第 1 个电平跳转沿有效，否则数据在第 2 个电平跳转沿有效。其相应的时序逻辑如图 5-15 所示。

可以通过增加片选信号 SS_n 的位宽来支持多个从设备，SS_n 的位宽等于从设备的个数。对于某时刻被选中的从设备和主设备而言，其读写时序逻辑和图 5-16 一样。

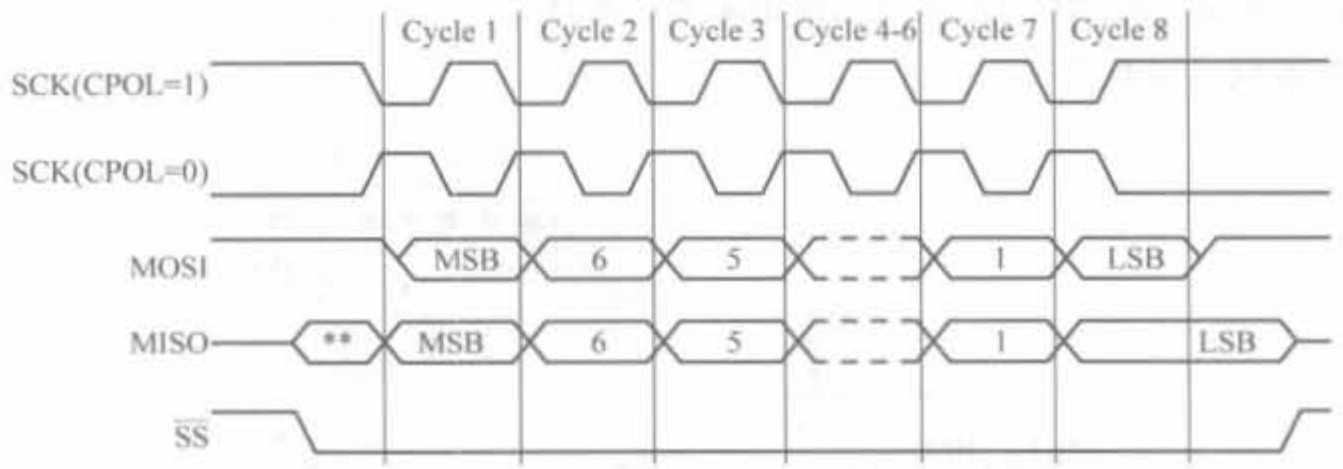
SPI 串行 Flash 作为一种新兴的高性能非易失性存储器，其有效读写次数高达百万次，不仅管脚数量少、封装小、容量大，可以节约电路板空间，还能够降低功耗和噪声。从功能上看，它可以用于代码存储以及大容量的数据和语音存储。对于以读为主，仅有少量擦写和写入时间的应用来说，支持分区(多页)擦除和页写入的串行存储是最佳方案。

2. SPI 串行 Flash 配置电路

SPI 串行配置模式常用于已采用了 SPI 串行 Flash PROM 的系统，在上电时将配置数据加载到 FPGA 中，这一过程只需向 SPI 串行发送一个 4 字节的指令，然后串行 Flash 中的数据就像 PROM 配置方式一样连续加载到 FPGA 中。一旦配置完成，SPI 中的额外存储空间还能用于其他应用目的。



(a) CPHA为低电平时SPI的总线时序示意图



(b) CPHA为高电平时SPI的总线时序示意图

图 5-15 CPHA 为低电平或高电平时，SPI 的总线时序示意图

1）SPI 配置电路

虽然 SPI 接口是标准的 4 线接口，但不同的 SPI Flash PROM 芯片采用了不同的指令协议。FPGA 芯片通过变量选择信号 VS[2:0]来定义 FPGA 和 SPI Flash 的通信方式、FPGA 的读指令以及在有效接收数据前插入的冗余比特数。常用 SPI Flash 与 FPGA 的有效操作配置如表 5-9 所示，其余的 VS[2:0]配置留有他用。

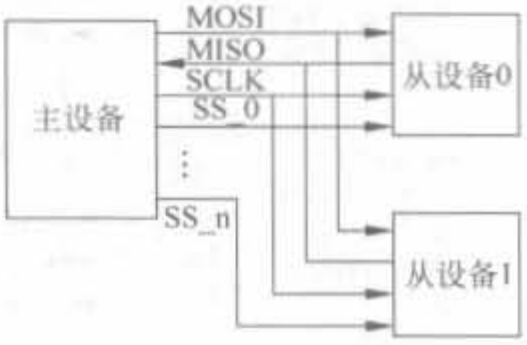


图 5-16 多个从芯片的连接电路图

表 5-9 Xilinx 芯片所支持的 SPI Flash 存储器以及配置列表

SPI Flash 型号		ISE 是否支持	读命令			容量（Mbit）								
			快速读（0X0B）	读（0X03）	读阵列（0XE8）									
			FPGA VS[2:0] 设置											
厂家	系列		1:1:1	1:0:1	1:1:0	0.512	1	2	4	8	16	32	64	128
STMicro	M25P	*	*	*		*	*	*	*	*	*	*	*	*
	M25PE	*	*	*			*	*	*	*				
	M45PE	*	*	*			*	*	*	*				
Atmel	AT45DB_D	*	*	*	*		*	*	*	*	*	*	*	
	AT45DB_B	*			*		*	*	*	*				

从整体上看来，控制 SPI 串行闪存比较容易，只需要使用简单的指令就能完成读取、擦除、编程、写使能/禁止以及其他功能。所有的指令都是通过 4 个 SPI 管脚串行移位输入的。

不同型号的 FPGA 芯片具有数目不同的从设备片选信号，因此所挂的串行芯片数目也就不一样。例如，Spartan-3E 系列 FPGA 芯片只有 1 位 SPI 从设备片选信号，因此只能外挂一片 SPI 串行 Flash 芯片。在 SPI 串行 Flash 配置模式下，M[2:0]=3'b001。FPGA 上电后，通过外部 SPI 串行 Flash PROM 完成配置，配置时钟信号由 FPGA 芯片提供时钟信号，支持两类业界常用的 Flash。

图 5-17 给出了 Spartan-3E 系列 FPGA 支持 0X0B 快速读写指令的 STMicro 25 系列 PROM 的典型配置电路。其中的 Flash 芯片需要 Flash 编程器来加载配置数据；单片的 FPGA 芯片构成了完整的 JTAG 链，仅用来测试芯片状态，以及支持 JTAG 在线调试模式，与 SPI 配置模式没有关系。

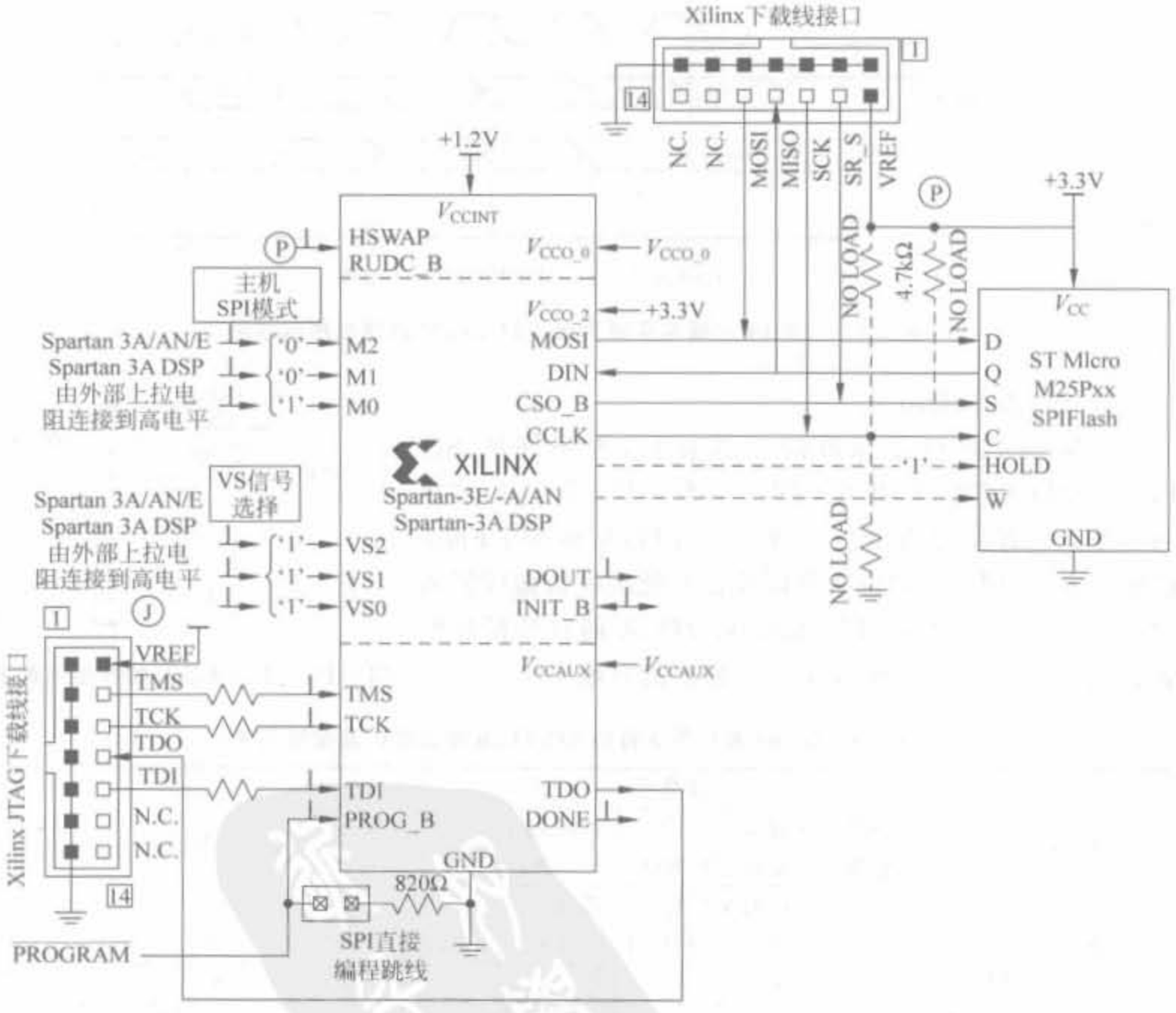


图 5-17　支持快读写的串行 Flash 配置电路示意图

从中可以看出，SPI Flash 容量大，适合于大规模设计场合。但由于 SPI 配置需要专门的 Flash 编程器，且操作起来比较麻烦，不适合在产品研发阶段调试 FPGA 芯片，因此一般

还会添加JTAG链专门用于在线调试。JTAG在线调试模式的原理以及注意事项将在5.3.5节详细说明。

图5-18给出了Spartan-3E系列FPGA支持SPI协议的Atmel公司"C"、"D"系列串行Flash芯片的典型配置电路。这两个系列的Flash芯片可以工作在很低温度,具有短的时钟建立时间。同样,单片的FPGA芯片构成了完整的JTAG链,仅用来测试芯片状态,以及支持JTAG在线调试模式,与SPI配置模式没有关系。

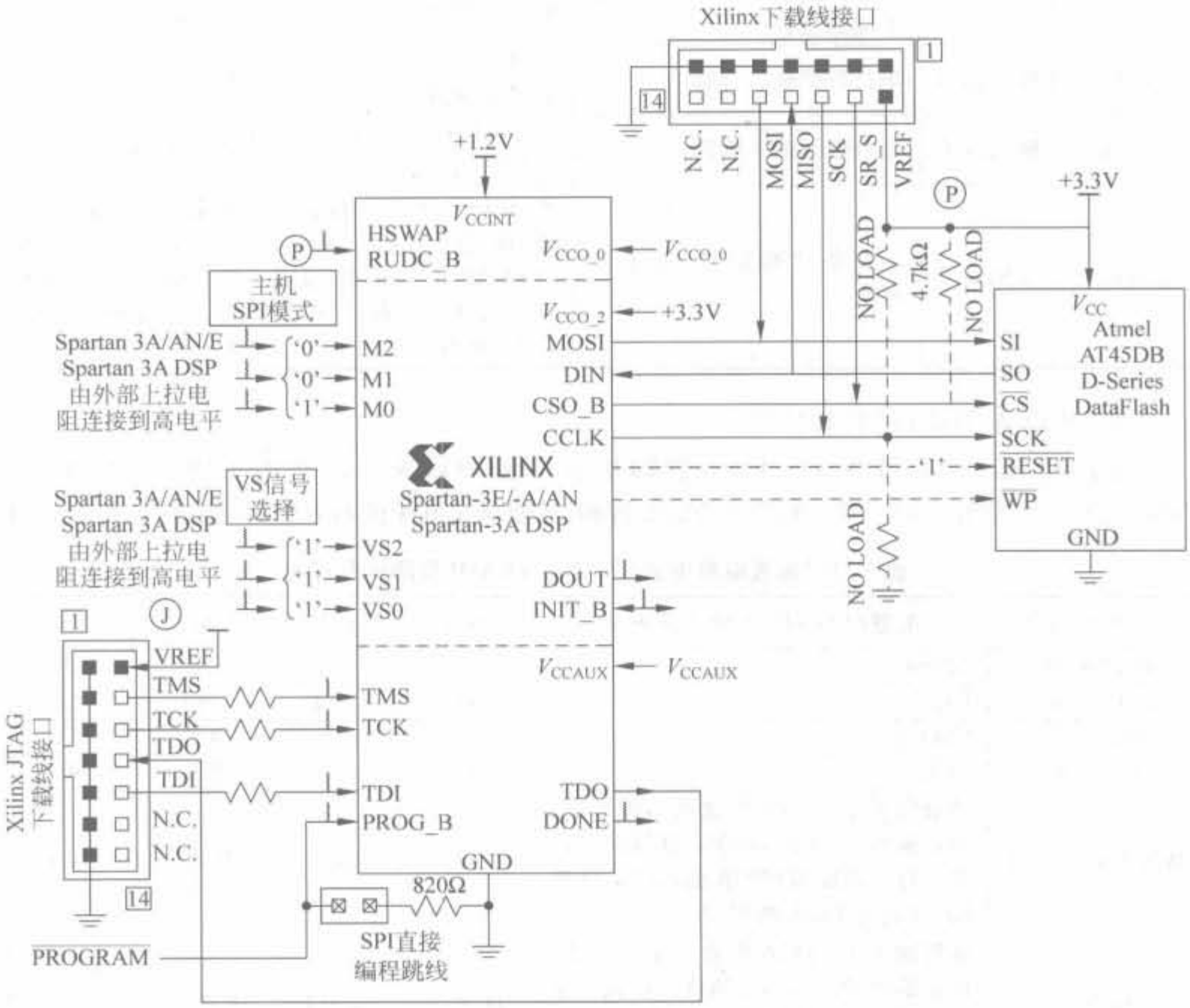


图5-18 Atmel SPI串行Flash配置电路示意图

表5-10给出了SPI配置接口的连线说明,每个SPI Flash PROM采用的名字略有不同,SPI Flash PROM的写保护信号和保持控制信号在FPGA配置阶段是不用的。其中,HOLD管脚在配置阶段必须为高电平;为了编程Flash存储器,写保护信号必须为高电平。

2) 相关信号说明

(1) FPGA端信号说明

对SPI配置模式下FPGA信号的说明,如表5-10所示。由于JTAG管脚已在多处提及,这里不再介绍。

表 5-10　配置电路中的 FPGA 管脚信号说明

管脚信号名	信号方向	功能描述	配置期间的功能	配置后的功能
M[2:0]	输入信号	FPGA 芯片配置模式选择管脚，对于 SPI 模式，需要设置为 M[2:0]=3'b001	配置管脚，在 INIT_B 信号变为高电平后采样其管脚电平	用户 I/O
VS[2:0]	输入信号	指令模式选择管脚，设置 FPGA 如何与 SPI 串行 Flash 通信，不同芯片具有不同的选择	指令选择管脚，在 INIT_B 信号变为高电平后采样其管脚电平	用户 I/O
MOSI	输出信号	串行数据输出管脚	用于向 SPI 发送读命令和起始地址	用户 I/O
DIN	输入信号	串行数据输入管脚	用于 FPGA 接收 Flash 的串行数据	用户 I/O
CSO_B	输出信号	从设备片选信号，低电平有效	和 SPI Flash 的片选信号连接在一起。如果 HSWAP＝1，需要通过 4.7kΩ 的电阻上拉到 3.3V 电源	配置完成后，CSO_B 会被置为高电平，放弃控制 Flash。也可以继续复用该管脚，继续和 Flash 通信

(2) 从设备 Flash 的管脚信号

由于不同公司、不同型号的 Flash 管脚并不一致，所以表 5-11 列出了所有出现在串行 Flash 芯片上的信号。对于某一特定的 Flash 管脚，需要挑选其中的有效管脚，见表中的 3～6 列。

表 5-11　配置电路中的 SPI 串行 FLASH 管脚信号说明

Flash 管脚名	配置时与 FPFA 的连接方式	STMicro	NexFLash	Silicon	Atmel
DATA_IN	MOSI	D	DI	SI	SI
DATA_OUT	DIN	Q	DO	SO	SO
SELECT	CSO_B	S	CS	CE	CS
CLOCK	CCLK	C	CLK	SCK	SCK
WR_PROTECT	该管脚对于 FPGA 配置是不需要的，但在编程 Flash 时，必须将其拉高。也可以将其连接到 FPGA 的用户 I/O 管脚，完成对 Flash 的写操作	W	WP	WP#	WP
HOLD	该管脚对于 FPGA 配置是不需要的，但在编程 Flash 时，必须将其拉高。也可以将其连接到 FPGA 的用户 I/O 管脚，完成对 Flash 的写操作	HOLD	HOLD	HOLD#	N/A
RESET	只有 Atmel 公司的 Flash 才有。用于 Flash 芯片复位，在配置 FPGA 时不需要该管脚	N/A	N/A	N/A	RESET
RDY/BUSY	只有 Atmel 公司的 Flash 才有，在配置 FPGA 时不需要该管脚	N/A	N/A	N/A	RDY/BUSY

5.3.3　从串配置模式

从串配置模式的特点已在前文介绍，所用管脚的说明和主串模式一样，因此本节直接介

绍从串配置电路原理以及注意事项。

在串行模式下，需要微处理器或微控制器等外部主机通过同步串行接口将配置数据串行写入 FPGA 芯片，其模式选择信号 M[2:0]=3'b111。典型的 Spartan-3E 系列 FPGA 单片配置电路如图 5-19 所示。DIN 输入管脚的串行配置数据需要在外部时钟 CCLK 信号前有足够的建立时间。其中，单片 FPGA 芯片构成了完整的 JTAG 链，仅用来测试芯片状态，以及支持 JTAG 在线调试模式，与从串配置模式没有关系。外部主机通过下拉 PROG_B 启动配置并检测 INIT_B 电平。当 INIT_B 为高电平时，表明 FPGA 做好准备，开始接收数据。此时，主机开始提供数据和时钟信号，直到 FPGA 配置完毕且 DONE 管脚为高电平，或者 INIT_B 变为低电平表明发生配置错误才停止。整个过程需要比配置文件大小更多的时钟周期，这是由于部分时钟用于时序建立，特别是当 FPGA 被配置为等待 DCM 锁存其时钟输入时。

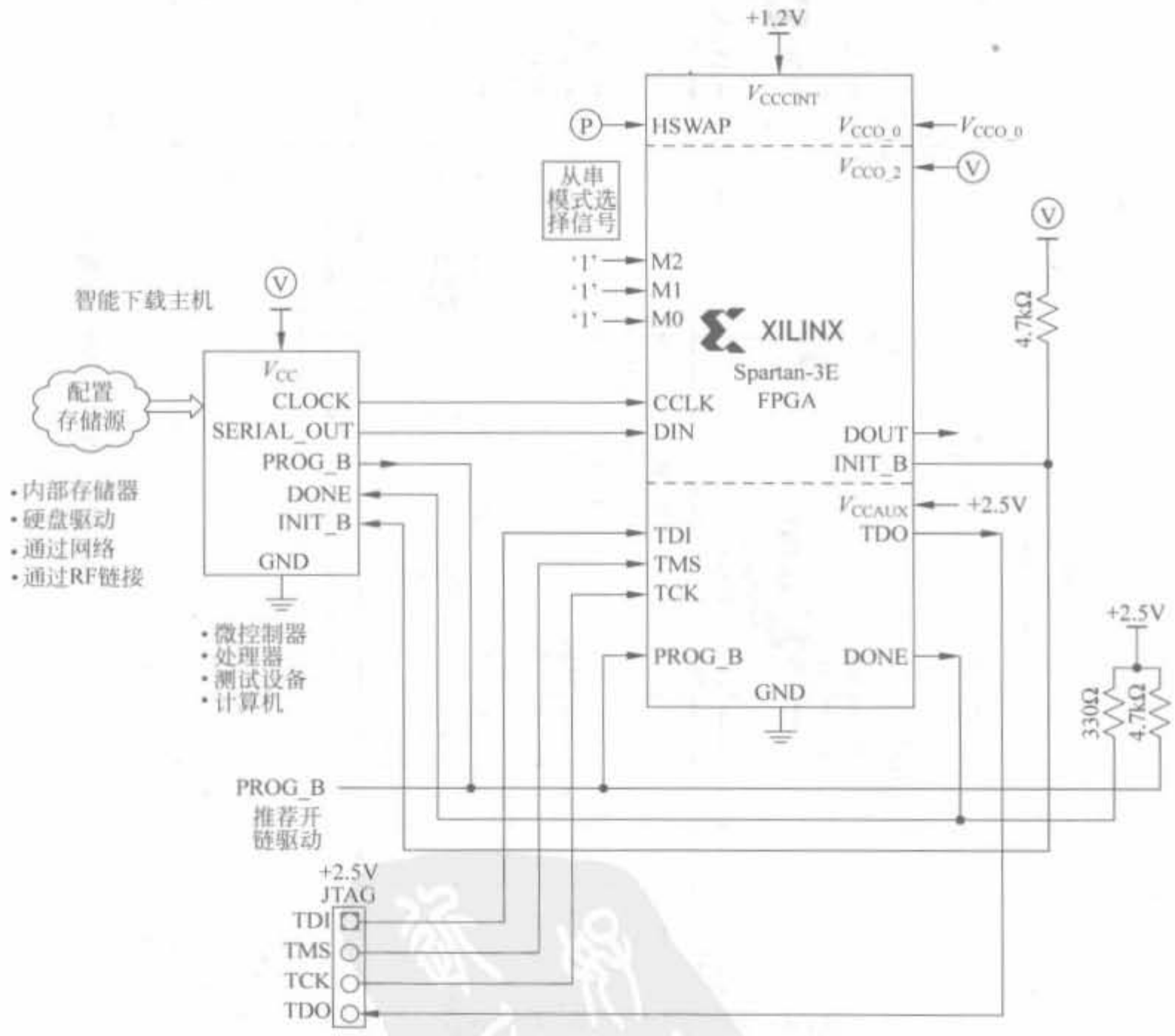


图 5-19 FPGA 从串配置电路示意图

此外，从串配置模式也可配置多片 FPGA 芯片，典型的两片 Spartan-3E 系列 FPGA 的从串配置电路如图 5-20 所示。所有芯片的 CCLK 信号都由主控设备提供，靠近主控设备的 FPGA 要充当桥梁的作用，将配置数据转发到第二个 FPGA 芯片。可以看到，采用从串配置的好处主要在于节省电路板面积，并使得系统具备更大的灵活性。

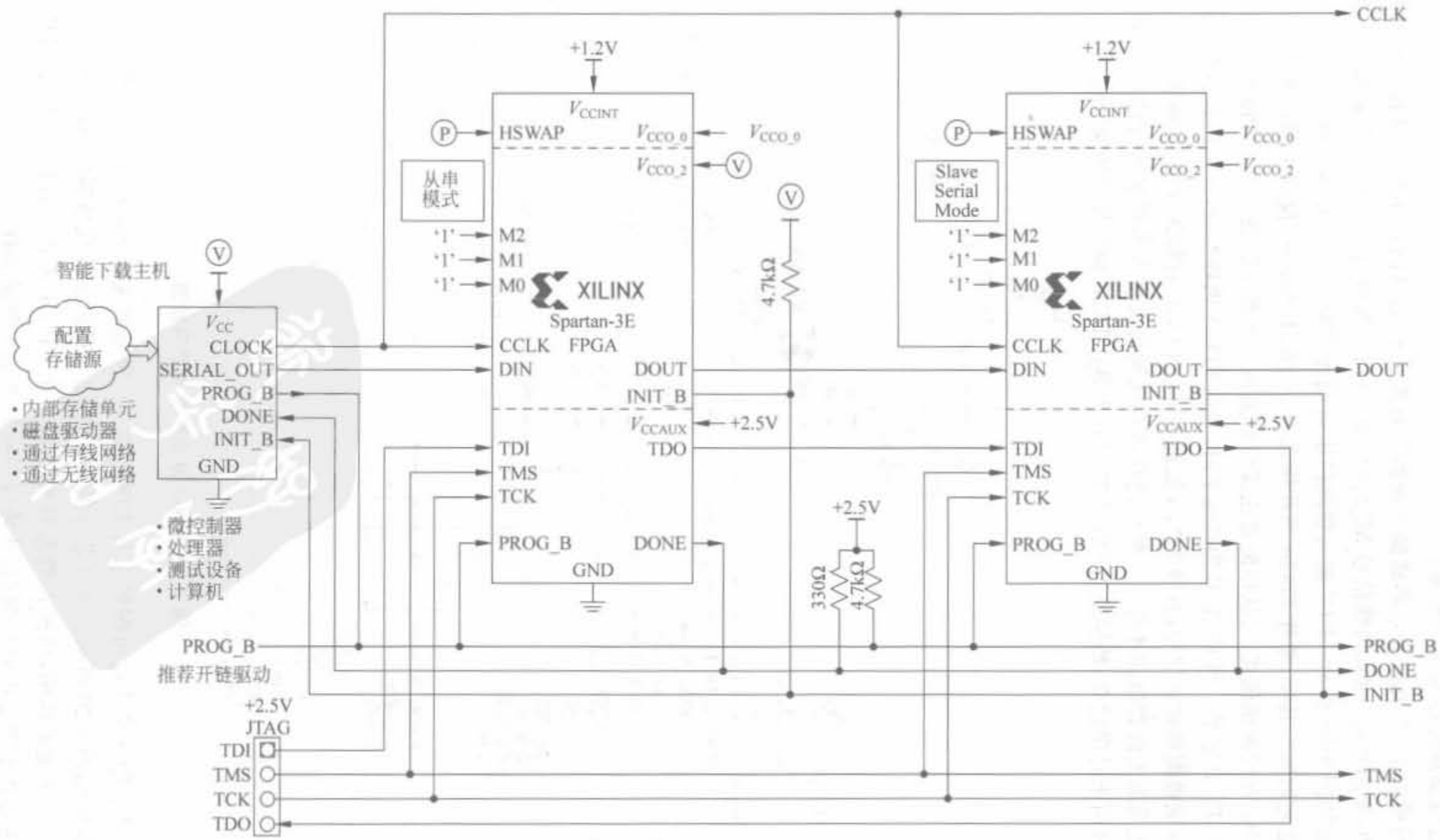


图 5-20　多片 FPGA 从串模式配置电路

5.3.4 字节宽度外部接口并行配置模式

本节主要介绍基于闪存(Flash)的 FPGA 配置方案,然后介绍字节宽度并行配置模式的配置电路,再对其配置信号进行说明,最后介绍多片 FPGA 的配置电路。

1. 并行 Flash 介绍

NOR 和 NAND 是现在市场上两种主要的非易失闪存技术。NOR 的特点是芯片内执行(Execute In Place,XIP),应用程序可以直接在 Flash 闪存内运行,不必再把代码读到系统 RAM 中。NOR 的传输效率很高,在 1～4MB 的小容量时具有很高的成本效益,但是很低的写入和擦除速度大大影响了它的性能。NAND 结构能提供极高的单元密度,可以达到高存储密度,并且写入和擦除的速度也很快,是高数据存储密度的理想解决方案。应用 NAND 的困难在于 Flash 的管理和需要特殊的系统接口。

2. BPI 单芯片配置模式

1) BPI 配置电路

BPI 配置接口主要用于支持标准的并行 NOR 闪存以及字节位宽或字位宽的 PROM 芯片。在 BPI 模式下,FPGA 从外部标准的 NOR 闪存或 NAND 闪存中,以字节宽度并行地获取配置数据,Spartan-3E 系列 FPGA 芯片在 BPI 模式下的 NOR Flash 电路如图 5-21 所示。当然,可以将该配置模式推广到其余并行配置外设中,地址、数据、片选(OE)以及写使能(WE)等控制信号都是通用的。

配置接口时序由 FPGA 芯片控制,最常用的方法是由 CCLK 管脚输出控制时钟,但是在单片 BPI 模式下并不使用 CCLK 信号,而是通过 LDC[2:0]和 HDC 管脚来作为闪存的控制输入。

根据访问 Flash 地址的递增和递减,可以将 BPI 模式分为 BPI UP 模式和 BPI DOWN 模式,由 M[0]管脚决定,其控制如表 5-12 所示。但无论哪种模式,地址总是在 CCLK 的下降沿变化。BPI UP 和 BPI DOWN 模式增加了 BPI 的灵活性,使其能够和其余嵌入式处理器或 CPU 等共享闪存。如果其余设备从 Flash 底部启动(Boot),FPGA 可采用 BPI UP 模式,否则可采用 BPI DOWN 模式共享存储器。

不同系列的芯片对 BPI 模式的支持是不一样的,在设计中要特别小心。表 5-13 列出了 Spartan-3、Spartan-3E 以及 Spartan-3A 系列 FPGA 芯片的 BPI 模式的支持差异性。

2) 配置信号说明

并不是所有 Xilinx 的所有 FPGA 都支持 BPI 配置模式,这里仍以 Spartan-3E 和 Spartan-3A 系列为例进行说明,各相关管脚的简要功能说明如表 5-14 所示。

3) 电压适配性

大多数并行 Flash 都采用 3.3V 单电源供电,而 BPI 配置模式所需的管脚一般至少分布在两个组(Bank)内,相应的 FPGA 分组必须使用 3.3V 电压来匹配并行 Flash。同样,也有部分 1.8V 的并行 Flash,因此相应的分组电平必须采用 1.8V。因此,设计之前要确定 FPGA 是否支持相应的电平。例如,由于 Spartan-3A 系列 FPGA 的上电复位(POR)电压不支持 1.8V,因此 Spartan-3A 不能外接并行 Flash。

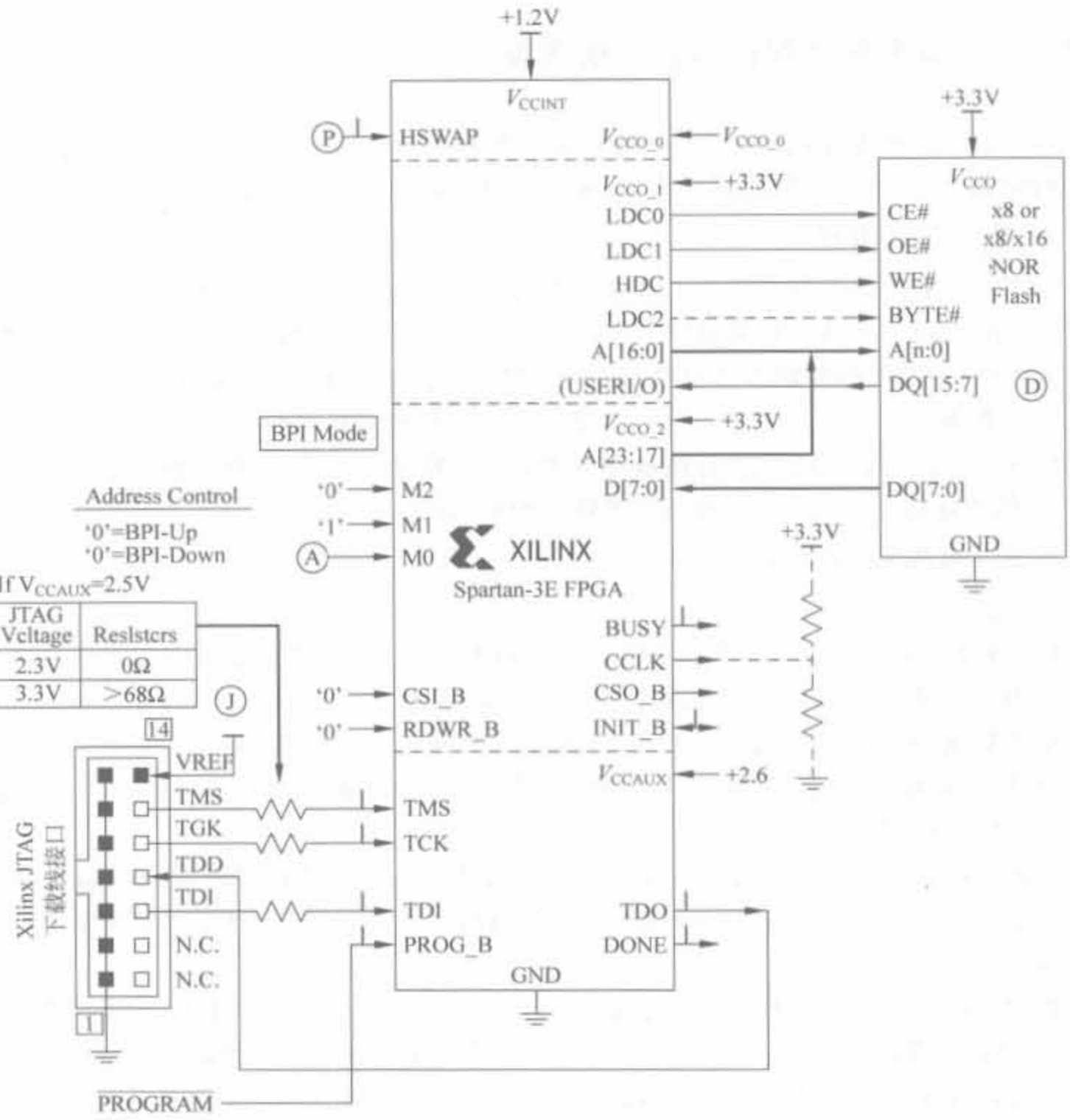


图 5-21　BPI 配置模式电路图

表 5-12　BPI 地址控制模式简要说明

M2	M1	M0	模式	起始地址	地址变化模式
0	1	0	BPI UP	0	递增
		1	BPI DOWN	0XFF_FFFF	递减

4）BPI 配置管脚的复用

当 FPGA 配置完成后，所有连接到闪存上的管脚都可以作为普通用户 I/O。如果配置完成后不再使用闪存，可以将 LDC0 信号拉高，使其片选信号无效。其余管脚，包括 A[25:0]或 A[23:0]地址线、D[7:0]数据线、LDC2、LDC1 以及 HDC 等控制管脚，在配置后都是用户 I/O，因此可以继续访问闪存。常见的闪存容量为 1～8Mbit，甚至更大，而一片 Spartan-3E 系列 FPGA 芯片最多只需 6Mbit，因此可以用闪存剩余空间来存储应用程序的数据，如 MicroBlaze 软核的应用数据以及以太网设备的 IP、MAC 地址等。

表 5-13 Spartan-3 代芯片对 BPI 模式支持的差异列表

	Spartan-3 FPGA	Spartan-3E FPGA	Spartan-3A/3AN Spartan-3A DSP FPGA
BPI UP 模式	Spartan-3 系列 FPGA 不支持 BPI 模式	是	是
BPI DOWN 模式		是	否
最大地址线数量		24	26
用于 Flash 寻址管脚的分布		Bank 1 和 Bank 2	Bank1
是否支持并行菊花链		是	是
是否支持串行菊花链		否	是
是否支持多启动(MultiBoot)模式		是	是
看门狗计数器重试		否	是
由配置速度设置控制的时序接口数		3	8
主 BPI 模式下 CCLK 信号的方向		输入/输出	输出
配置期间是否需要 RDWR_B 和 CSI_B 信号		是	否
芯片内 M[2:0]管脚是否具有上拉电阻		无,或可通过 HSWAP 管脚配置	是

表 5-14 BPI 模式下 FPGA 配置管脚说明列表

管脚名	方向	简要功能说明	配置期间的要求	配置后功能
M[2:0]	输入	模式选择管脚,用于确定 BPI 模式,其中对于 3A 系列 FPGA,内部具有上拉电路	M2=0,M1=1,M0=0,从地址 0 开始,启动 BPI UP 模式。对于 3E 系列,也可以使 M0=1,启动 BPI DOWN 模式	普通用户输入/输出端口
CSI_B (仅 3E 系列)	输入	片选信号,低电平有效	在配置期间必须为低电平。对于 3A 系列,则没有要求	普通用户输入/输出端口
RDWR_B (仅 3E 系列)	输入	读、写控制信号,低电平为写有效。读操作经常在配置完成之后使用	在配置期间必须为低电平。对于 3A 系列,则没有要求	普通用户输入/输出端口
LDC0	输出	PROM 芯片使能信号	和 PROM 芯片的 CS# 管脚相连,配置期间必须为低电平	普通用户输入/输出端口。如果配置完成之后不再访问 FPGA,将其拉高即可。此时,A[23:0]、LDC1、LDC2、HDC 以及 D[7:0]才能作为普通 I/O 管脚
LDC1	输出	PROM 芯片输出使能信号	和 PROM 芯片的 OE# 管脚相连,配置期间必须为低电平	普通用户输入/输出端口
LDC2	输出	PROM 字节模式选择信号	对于只支持字节模式的 PROM 无效。对支持 8bit 或 16bit 宽度的 PROM,可将其连接到 BYTE# 管脚,在配置期间为低电平	普通用户输入/输出端口。配置完毕后,可在 16bit 模式下将其拉高,进入字节模式

续表

管脚名	方向	简要功能说明	配置期间的要求	配置后功能
HDC	输出	PROM 芯片写使能信号	和 PROM 芯片的 WE# 管脚相连，配置期间必须为低电平	普通用户输入/输出端口
A[23:0]（仅 3E 系列）、A[25:0]（仅 3A 系列）	输出	地址信号	连接到 Flash 芯片的地址管脚上，对于某一特定芯片，并不需要连接所有管脚	普通用户输入/输出端口
D[7:0]	输入	数据输入信号	FPGA 接收字节宽度的配置数据，每次数据采样总发生在 CCLK 的下降沿	普通用户输入/输出端口
CSO_B	输出	片选输出信号，低电平有效	在单片 FPGA 配置中无用，常用于链状，连接到下一片 FPGA 的 CSI_B 管脚	普通用户输入/输出端口
BUSY（仅 3E 系列）	输出	芯片忙指示信号	在单片 FPGA 配置中无用，一般上拉即可	普通用户输入/输出端口
DOUT	输出	串行数据输出信号	在单片 FPGA 配置中无用，一般上拉即可。在链状结构中，连接到下一芯片的 DIN 管脚	普通用户输入/输出端口

例如，在嵌入式应用中，将 FPGA 逻辑电路和软核控制器 MicroBlaze 的应用数据所形成的比特文件存在闪存中，FPGA 首先从闪存中读取逻辑的配置文件；等逻辑配置完毕后，再利用 FPGA 内部已形成的逻辑加载软核的应用数据，即可直接读取执行，也可以先将应用数据映射到 DDR SDRAM，再从 DDR SDRAM 中读取程序并执行。当然，也可以将 FPGA 程序中所需要的大量非易失性应用数据存放在闪存中。

需要注意的是，不要将 FPGA 配置数据和用户数据存放在闪存的同一段中。

5）字节和字配置模式

目前市场上的中、小规模密度的闪存容量一般在 8Mbit 以下，只能作为比特宽度（8bit）的存储器来使用。大多数高密度的闪存芯片容量一般都在 16Mbit 以上，具有模式选择输入信号 BYTE，可以支持字节宽度和字宽度（16bit）这两种读写方式。在图 5-22 中，FPGA 芯片的 LDC2 管脚用来选择配置位宽模式，支持字模式读写。字节宽度和字宽度模式的电路连接是不同的。

虽然 Spartan-3E 系列 FPGA 支持字节/字模式且连接简单，但需要注意的是，不同厂家的闪存芯片地址线的管脚数和命名规则是不一样的，在连接时要确保 FPGA 和闪存连接正确。如 Intel、Micron 等公司采用简单思路，管脚名称和 FPGA 一致，虽然数量较多，但比较直观（如 A0、D15 等），但闪存的 A0 脚在字模式中是不用的，且需要一个额外的用户 I/O 连接到 D15 脚，如图 5-22(a)所示。

另外一类生产商，如 AMD、Atmel 等公司，采用高效的思路，管脚数较少，且通过管脚 I/O 来实现两种模式的选择，在配置时选用字节宽度，配置后应用程序使用字宽度读取数据，如图 5-22(b)所示。在字节宽度中，BYTE# =0，由 FPGA 芯片的 LDC2 控制。I/O 信号控制选择字节定位，地址线 A0 用于选择字定位。当 FPGA 配置成功后，应用程序驱动 BYTE# =1，选择 16bit 的字模式读取时钟，确保 D[14:8]连接到用户 I/O 管脚上，D15 连接到 FPGA 的 A0 管脚。其中 I/O 是最重要的数据比特，如果一款闪存芯片有 I/O 管脚或 DQ15/A-1，在连接时一定要根据图 5-22 来连接。

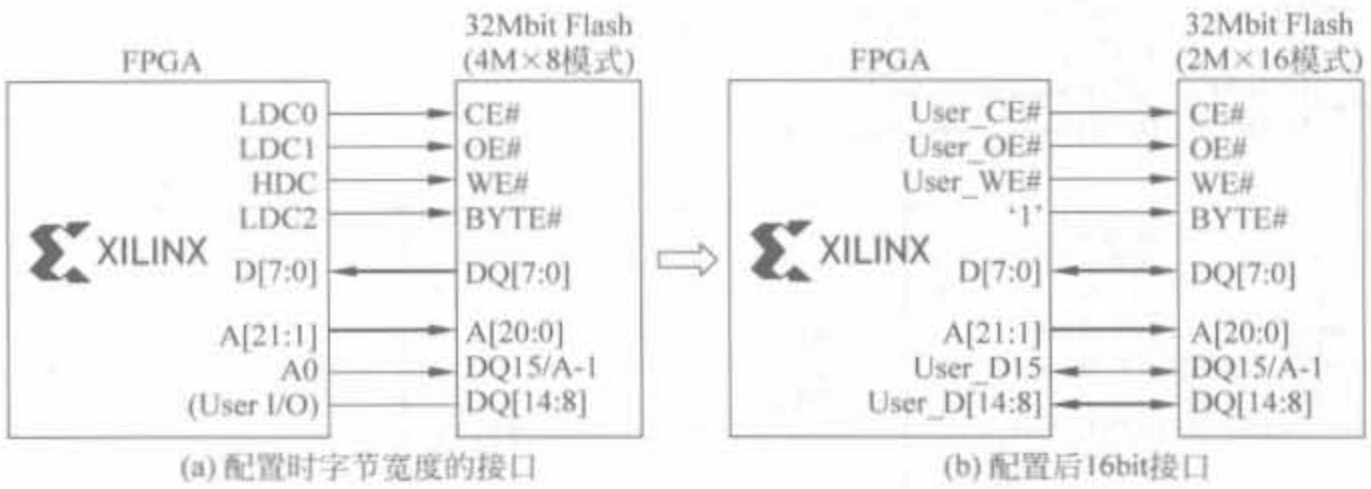


图 5-22 具有 DQ15/A-1 管脚的 FPGA 连接方式

为了突出 AMD 等闪存产品不同模式的连接区别，在表 5-15 中给出其连接说明。

表 5-15 具有 I/O 管脚的闪存连接说明

FPGA 管脚	与闪存的连接管脚	×8 模式配置后的 FPGA 管脚	×16 模式配置后的 FPGA 管脚
LDC2	BYTE#	将 LDC2 拉低或者悬空，并且将 PROM 的 BYTE# 信号拉低	将 LDC2 置高
LDC1	OE#	将 Flash 输出有效控制信号拉低	将 Flash 输出有效控制信号拉低
LDC0	CS#	将 Flash 片选信号拉低	将 Flash 片选信号拉低
HDC	WE#	用于控制 Flash 的写有效控制信号	用于控制 Flash 的写有效控制信号
A[23:1]	A[2:0]	A[n:0]	A[n:0]
A0	I/O	地址输入信号	重要数据输入信号，且为数据的最高位 User_D15
D[7:0]	I/O[7:0]	I/O[7:0]	I/O[7:0]
User I/O	只有在×16 模式下才需要连接	无需连接	I/O[14:8]

一些闪存芯片要求 BYTE# 信号具有长的建立时间，因此为了确保配置正确，需要在板子上电或者 FPGA 重配置时，使 BYTE# 信号为低电平，选择字节模式。如果需要进一步加大 BYTE# 信号的建立时间，可以给 FPGA 芯片的 LDC2 管脚添加 680Ω 下拉电阻，或延迟 CSI_B 信号再送入 FPGA。

3. BPI 多芯片菊花链配置模式

一般来讲，闪存的容量对于配置 FPGA 是绰绰有余的，因此 BPI 模式非常适合配置多

片 FPGA 芯片。例如，典型的多片 Spartan-3E 芯片的配置电路如图 5-23 所示，其中除了主芯片供给 CCLK 信号给各个从芯片外，还需要一个中间媒介 FPGA 连接在主芯片和其余的从芯片之间。该媒介芯片必须为 Spartan-3A/E/AN 或 Virtex-5 系列的芯片，其余的从芯片可以为任意 Xilinx 芯片。

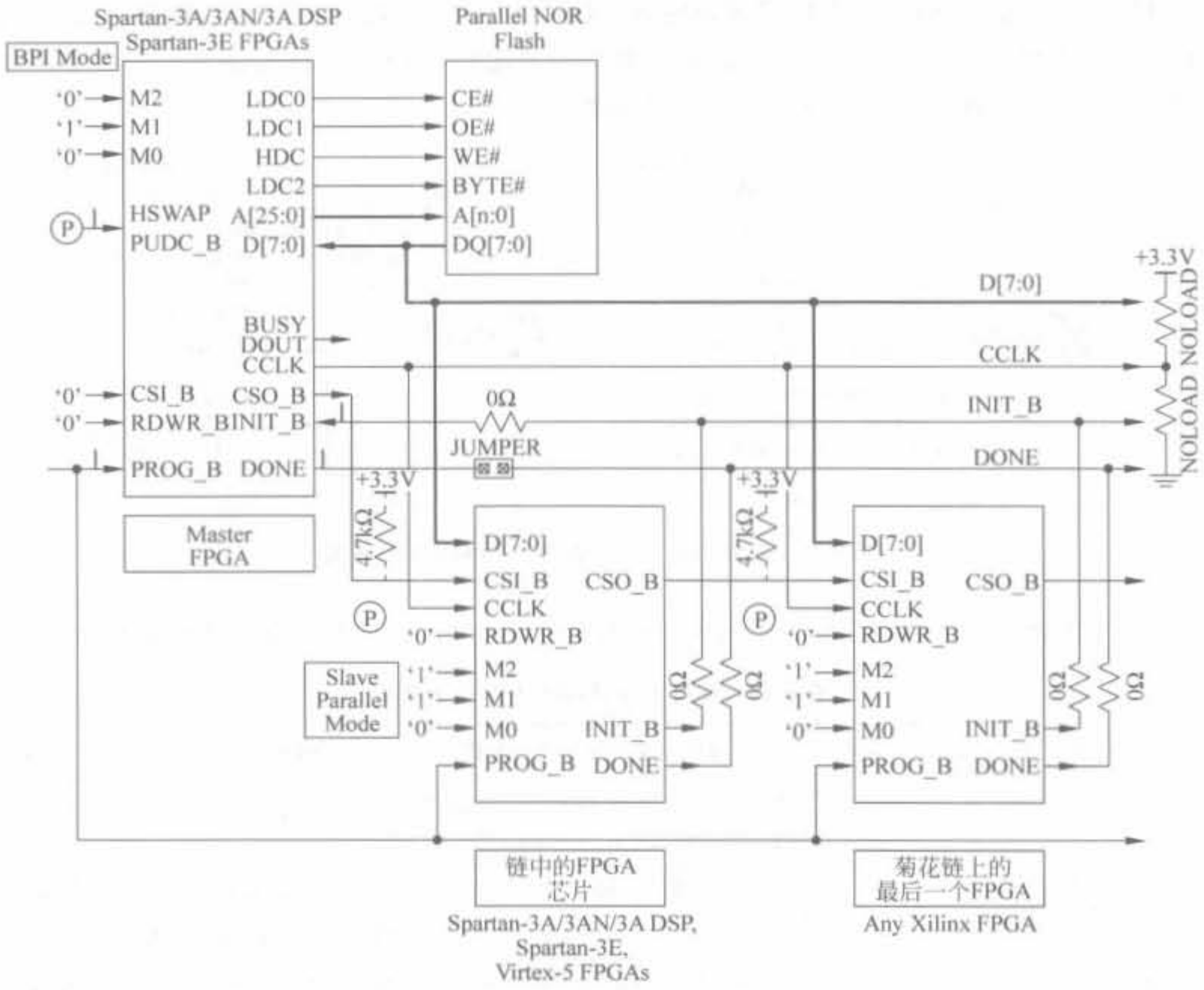


图 5-23　多芯片 BPI 模式配置电路

4. 基于 Xilinx PROM 的 BPI 配置模式电路

对于 BPI 配置模式，不仅可应用闪存芯片，也可采用 Xilinx 公司的并行 PROM 芯片搭建 BPI 配置电路。对于 Xilinx 并行 PROM 芯片，只能以×8 模式配置 FPGA，其效率和主并模式相同。基于 Xilinx 并行 PROM 芯片的 BPI 配置电路如图 5-24 所示，仍由 CCLK 管脚给 PROM 芯片 CLK 管脚提供输入信号。

5.3.5　JTAG 配置模式

1. JTAG 配置电路

Xilinx 公司的 FPGA 芯片具有 IEEE 1149.1/1532 协议所规定的 JTAG 接口，只要 FPGA 上电，不论模式选择管脚 M[2:0]的电平如何，都可采用该配置模式。但是将模式配

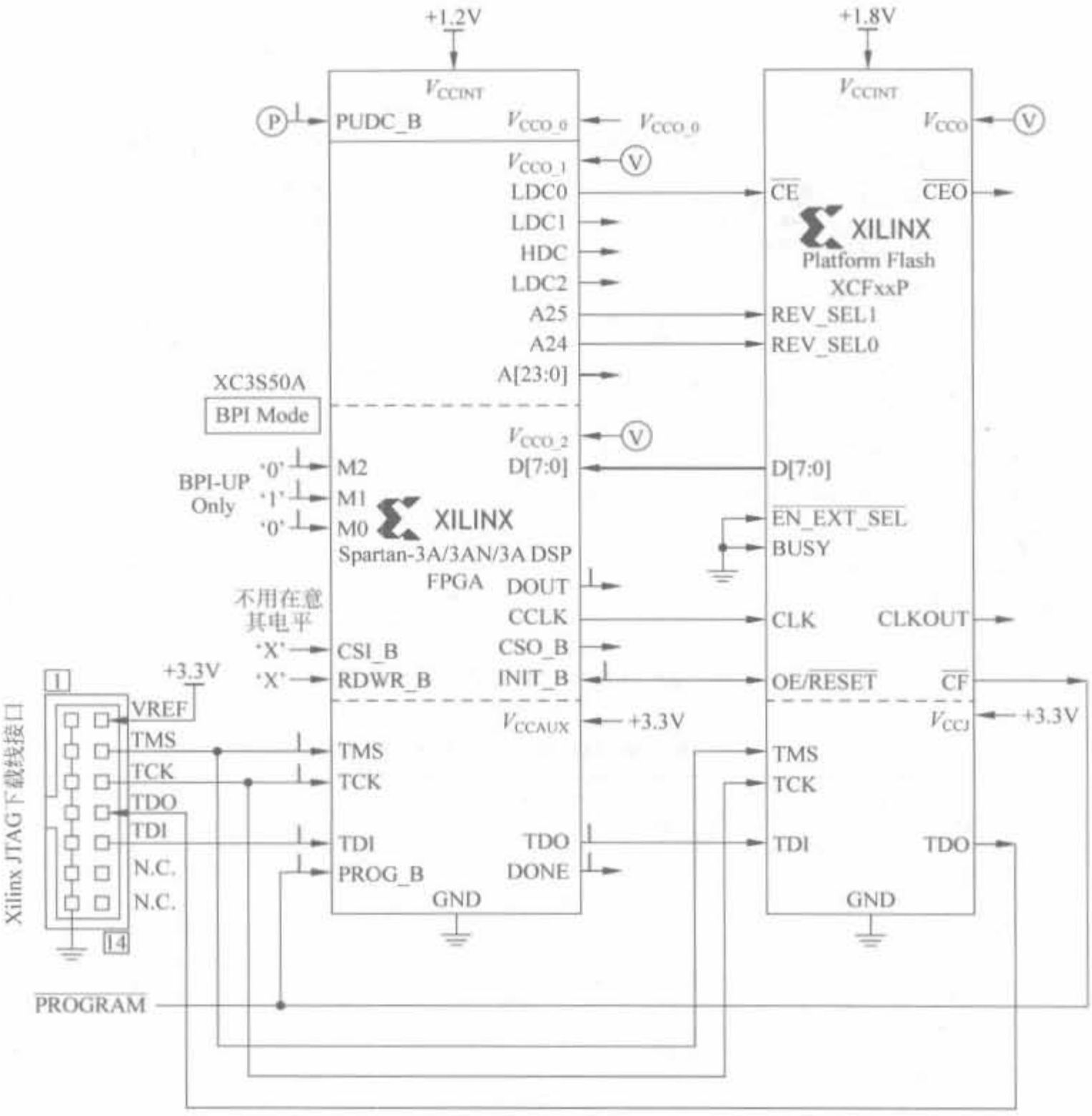


图 5-24 PROM 的 BPI 配置模式电路

置管脚设置为 JTAG 模式，即 M[2:0]=3'b101 时，FPGA 芯片上电后或者 PROG_B 管脚有低脉冲出现后，只能通过 JTAG 模式配置。JTAG 模式不需要额外的掉电非易失存储器，因此通过其配置的比特文件在 FPGA 断电后即丢失，每次上电后都需要重新配置。由于 JTAG 模式已更改，配置效率高，所以它是项目研发阶段必不可少的配置模式。典型的 Spartan-3E 系列芯片的 JTAG 配置电路如图 5-25 所示。

2. JTAG 芯片 ID 以及用户 ID

每片 Spartan-3E FPGA 芯片都有一个 32bit 的 JTAG 芯片识别号，如表 5-16 所示。其中的低 28 位表示 Xilinx 芯片向量和芯片识别标志；高 4 位常被大多数工具忽略，它代表了芯片电路中硅的修正版本号。表 5-16 中的修改版本号以分级形式给出。

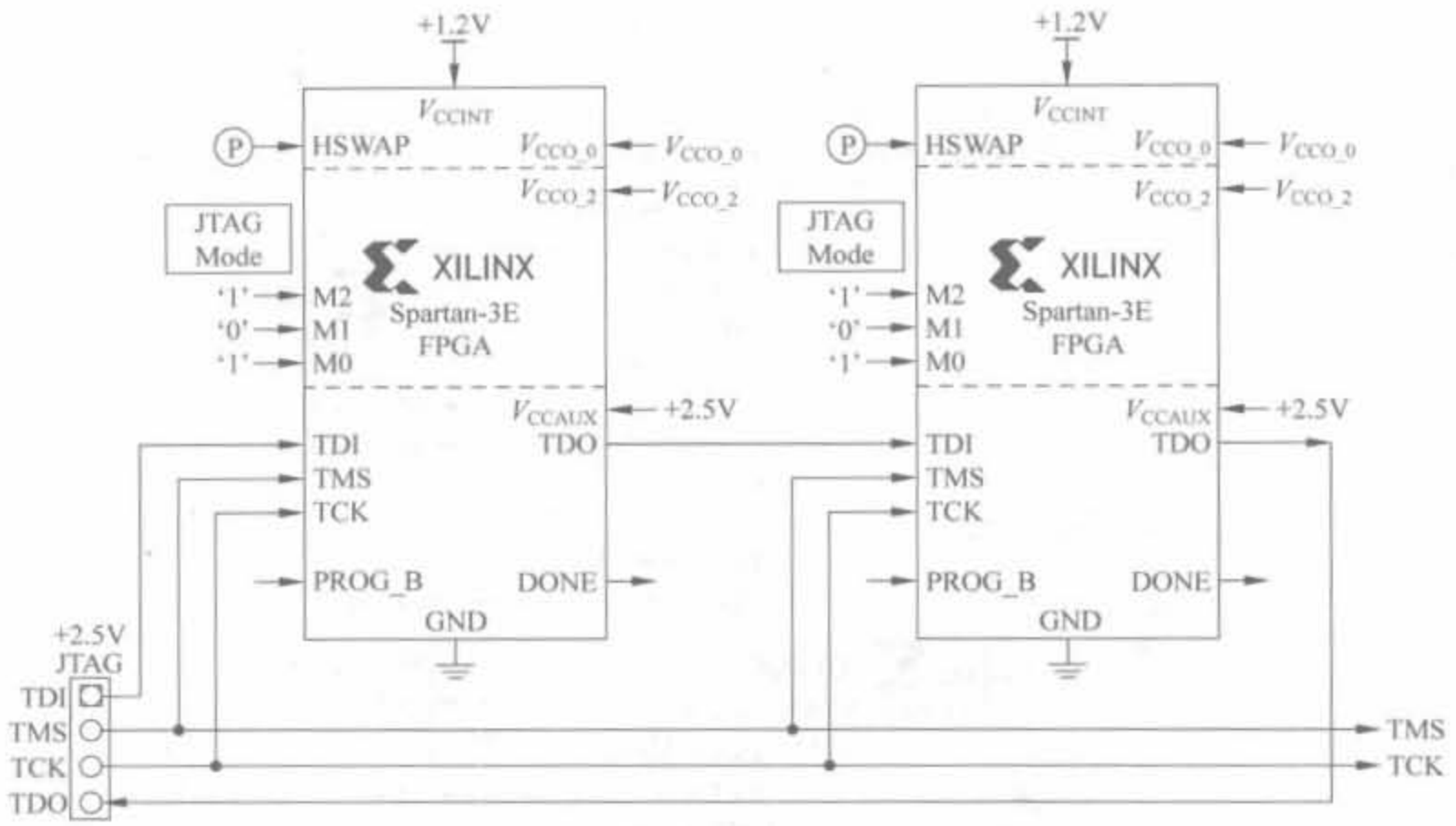


图 5-25　JTAG 模式配置电路示意图

表 5-16　Spartan-3E 芯片 JTAG ID 说明

Spartan-3E 系列 FPGA 型号	4 位版本号		28bit 芯片矢量识别号
	第 0 阶段	第 0 阶段	
XCS100E	0X0	0X1	0X1C10093
XCS250E	0X0	0X1	0X1C1A093
XCS500E	0X0 0X2	0X4	0X1C22093
XCS1200E	0X0 0X1	0X2	0X1C2E093
XCS1600E	0X0 0X1	0X2	0X1C3A093

JTAG 接口提供了配置过程存储用户 ID 的选项。用户 ID 值可以在配置比特文件中指定，其默认值为全 1，即 0xFFFF_FFFF。

3. 配置电压的适配问题

JTAG 连接器的电压最好与 FPGA 的参考电压 V_{CCAUX} 大小一致，否则其电路需要添加限流电阻，相对而言较为复杂。如果 JTAG 连接器必须工作在 3.3V，则需要在 JTAG 连接器和 FPGA 芯片的 TDI、TMS 以及 TCK 之间串接电阻，其大小如表 5-17 所示。FPGA 的 TDO 管脚一般通过 V_{CCAUX} 上拉，当 $V_{CCAUX}=2.5V$ 时，也可以通过外部电压直接上拉到 3.3V，不过这会降低 JTAG 电路的抗噪能力。

表 5-17 JTAG 电压和限流电阻的关系表

JTAG 连接器的接口电压	FPGA 芯片的参考电压 V_{CCAUX}	限流电阻的选取
2.5V	2.5V	电压相等，无需限流电阻
3.3V	2.5V	68Ω 或更大的限流电阻
3.3V	3.3V	电压相等，无需限流电阻

5.3.6 System ACE 配置方案

随着 FPGA 成为系统级解决方案的核心，大型、复杂设备常需要多片大规模的 FPGA。如果使用 PROM 进行配置，需要很大的 PCB 面积和高昂的成本，因此在很多情况下都利用微处理器从模式配置 FPGA 芯片，但该配置方案容易出现总线竞争且延长了系统启动时间。为了解决大规模 FPGA 的配置问题，Xilinx 公司推出了系统级的 System ACE (Advanced Configuration Environment)解决方案。

System ACE 可在一个系统内，甚至在多个板上，对 Xilinx 的所有 FPGA 进行配置，使用 Flash 存储卡或微硬盘保存配置数据，通过 System ACE 控制器把数据配置到 FPGA 中。目前，System ACE 有 System ACE CF(Compact Flash)、System ACE SC(Soft Controller)以及 System ACE MPM(Muti-Package Module)3 种。读者需要注意的是，System ACE SC/MPM 是和 System ACE CF 相互独立的解决方案。典型的 ACE 接口以及系统组成如图 5-26 所示。

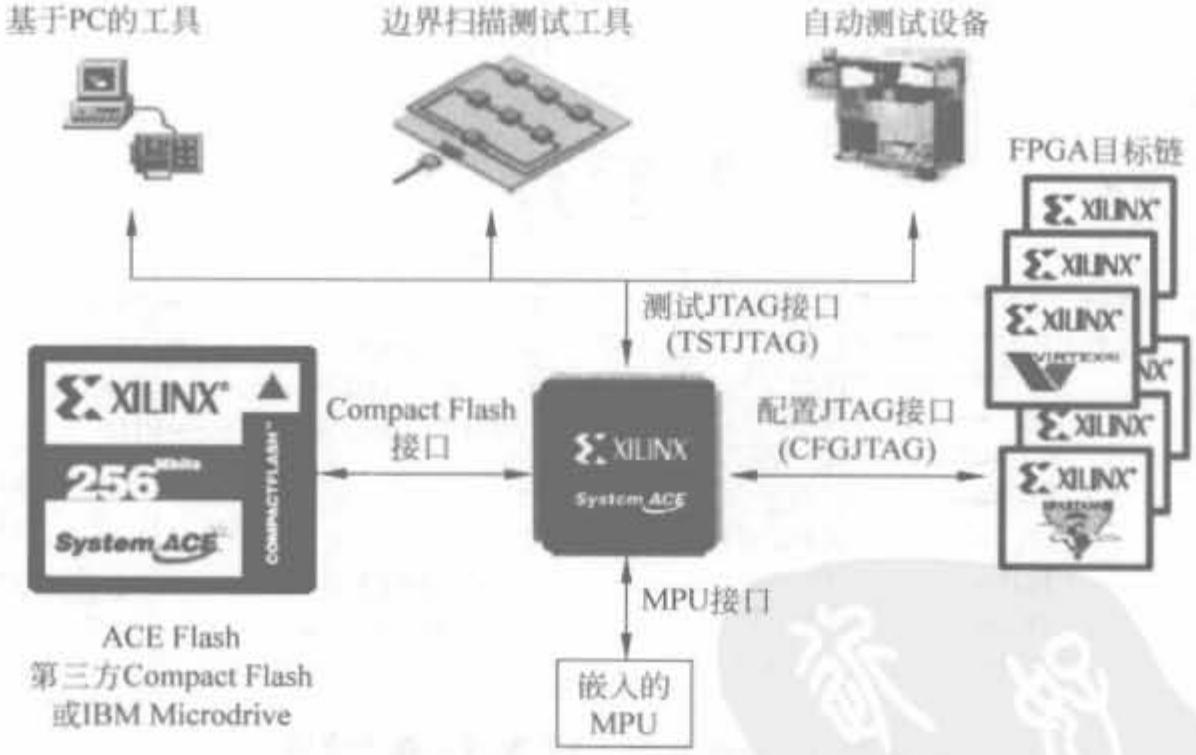


图 5-26 典型的 ACE 接口以及系统组成示意图

1. System ACE CF 解决方案

System ACE CF 的核心是 System ACE CF 存储设备和 System ACE 控制器芯片。System ACE CF 存储设备包括 Xilinx 的 ACE Flash 卡或其他厂家的 Compact Flash 卡以及 IBM 的微硬盘。Compact Flash 卡的容量为 32MB～4GB，微硬盘的容量为 2GB～6GB，

至少可配置数百片 FPGA 芯片。

System ACE CF 控制器提供了存储单元和 FPGA 器件之间的接口，以及 PC 和存储器的标准 JTAG 接口。控制器芯片默认的配置模式也是通过边界扫描的方式将数据配置到 FPGA 链中，同样可由边界扫描链的测试和编程接口来辅助进行系统原型的调试，其主要特点有：

- 支持 Xilinx 所有 FPGA 芯片的配置；
- 以最小的 PC 板空间实现多达 8GB 的配置；
- 包括高达 152Mb/s 的配置速率；
- 利用带有嵌入式处理器核的 FPGA 进行系统调节；
- 管理多个比特流(全部或部分)，并按需要对其进行激活；
- 包含处理器核初始化；
- 软件存储加密；
- 可移动存储器件；
- 降低了定制配置系统的成本，支持大多数 CompactFlash 卡，包括 Microdrive 单元；包含内置式微处理器接口，可以直接调整 FPGA 配置；释放设计资源。

Compact Flash 接口是 ACE 控制器的关键接口，可连接 Compact Flash 卡、标准的 Compact Flash 模块以及 IBM 微硬盘。Compact Flash 可以拆卸，因此对存储内容进行修改和升级以及更换容量都非常方便。Compact Flash 接口由 Compact Flash 控制器和 Compact Flash 仲裁器两部分组成。由 System ACE CF 配置 FPGA 的接口电路如图 5-27 所示。

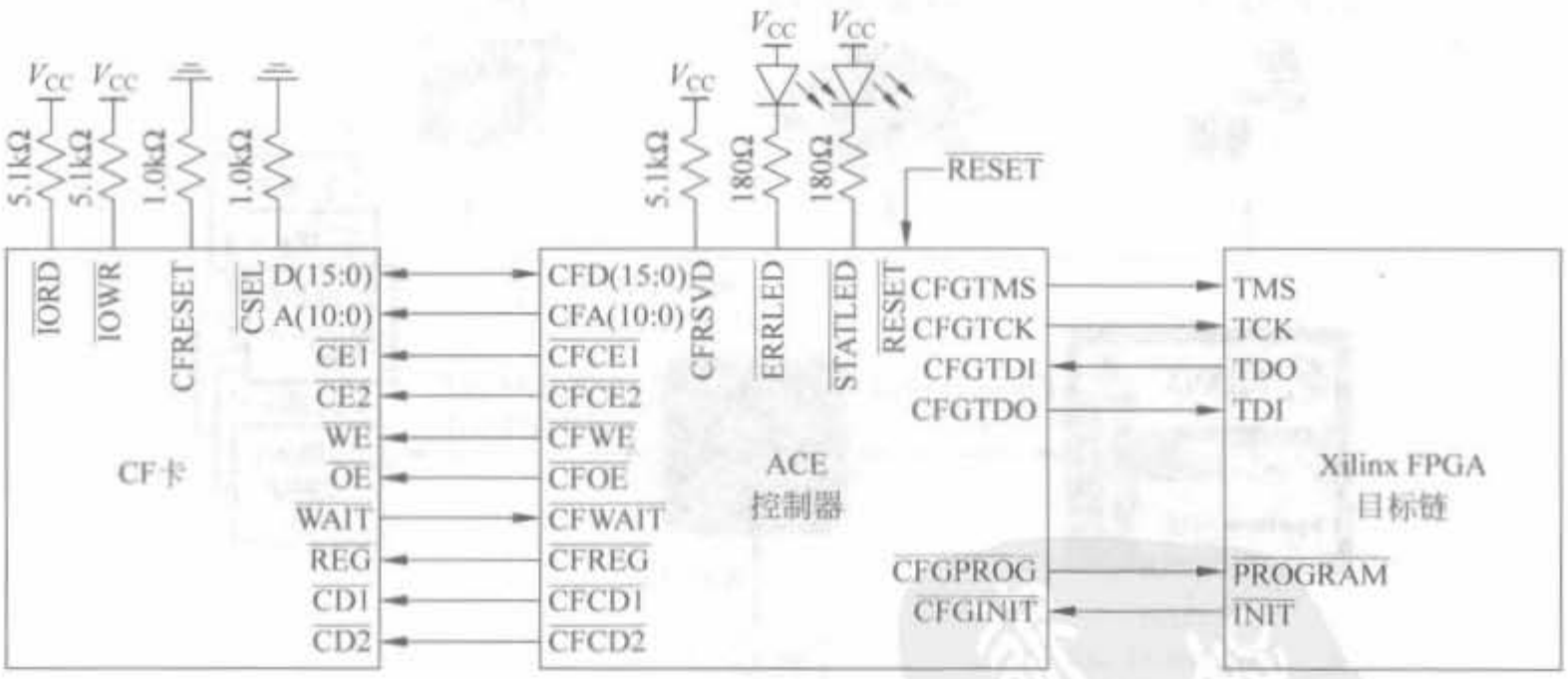


图 5-27　System ACE CF 配置电路示意图

2. System ACE SC 解决方案

System ACE SC 为用户提供了自主性，用户可以自由地选择每一部分的元件，可将其置于电路板的任何位置，且所有的功能在一个独立的 FPGA 中完成，并不需要整合其他组件。System ACE SC 有 4 个主要接口：边界扫描 JTAG 接口、系统控制接口、Flash 存储器接口以及 FPGA 接口，如图 5-28 所示。

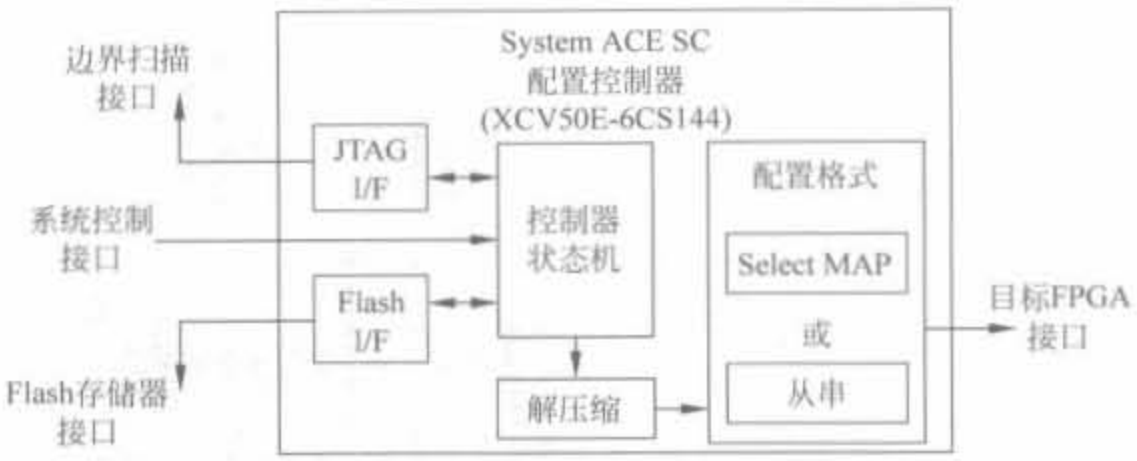


图 5-28 System ACE SC 接口示意图

其中,JTAG 接口主要提供边界扫描测试和对具有 JTAG 接口的 Flash 存储器通信;Flash 接口主要和外边的 Flash 芯片通信,读取存储器内的内容以及对存储器进行编程;系统控制接口主要提供输入时钟、配置控制信号和配置状态信号等;FPGA 接口主要用于配置 FPGA,可通过从串、从并以及 Select MAP 等配置模式。

System ACE SC 和 System ACE CF 的主要区别在于:System ACE SC 的控制器是一个软核逻辑,而不是芯片,需要和设计一起下载到 FPFA 中;其余区别如表 5-18 所示。

表 5-18 System ACE CF 和 System ACE SC 的区别

特性	System ACE CF	System ACE SC
存储器容量	高达 8GB	16/32/64Mbit
能否压缩	否	能
FPGA 配置模式	JTAG	Select MAP
最大配置速度	30Mb/s	152Mb/s
最大设计数	无限制	8
存储媒介	Compact Flash	AMD Flash
器件数目	2	3

典型的 System ACE SC 配置电路如图 5-29 所示。

3. System ACE MPM 解决方案

System ACE MPM 是一个整合的组件解决方案,包括 FPGA 和 PROM 组成的配置控制组件和一个 Flash 存储组件,并封装为一个模块,通过尽可能少的组件来实现配置电路。Xilinx 公司有 16Mbit、32Mbit 以及 64Mbit 低密度的 System ACE MPM。System ACE MPM 有 4 个主要接口,和 System ACE SC 的接口一样,其特征和功能也与 System ACE SC 一样。二者的区别在于:System ACE MPM 封装了整个配置模块,而 System ACE SC 允许用户自行配置,其接口电路如图 5-30 所示。

System ACE MPM 是 Xilinx 公司第一个支持位流压缩的配置方案,支持多种配置模式,可同时有多达 8 个 FPGA 链的从串配置模式和 4 个 FPGA 的 Select MAP 配置模式,最大配置速率为 152Mb/s,又可最大限度地减小电路板空间和连线。典型的 System ACE MPM 配置电路如图 5-31 所示。

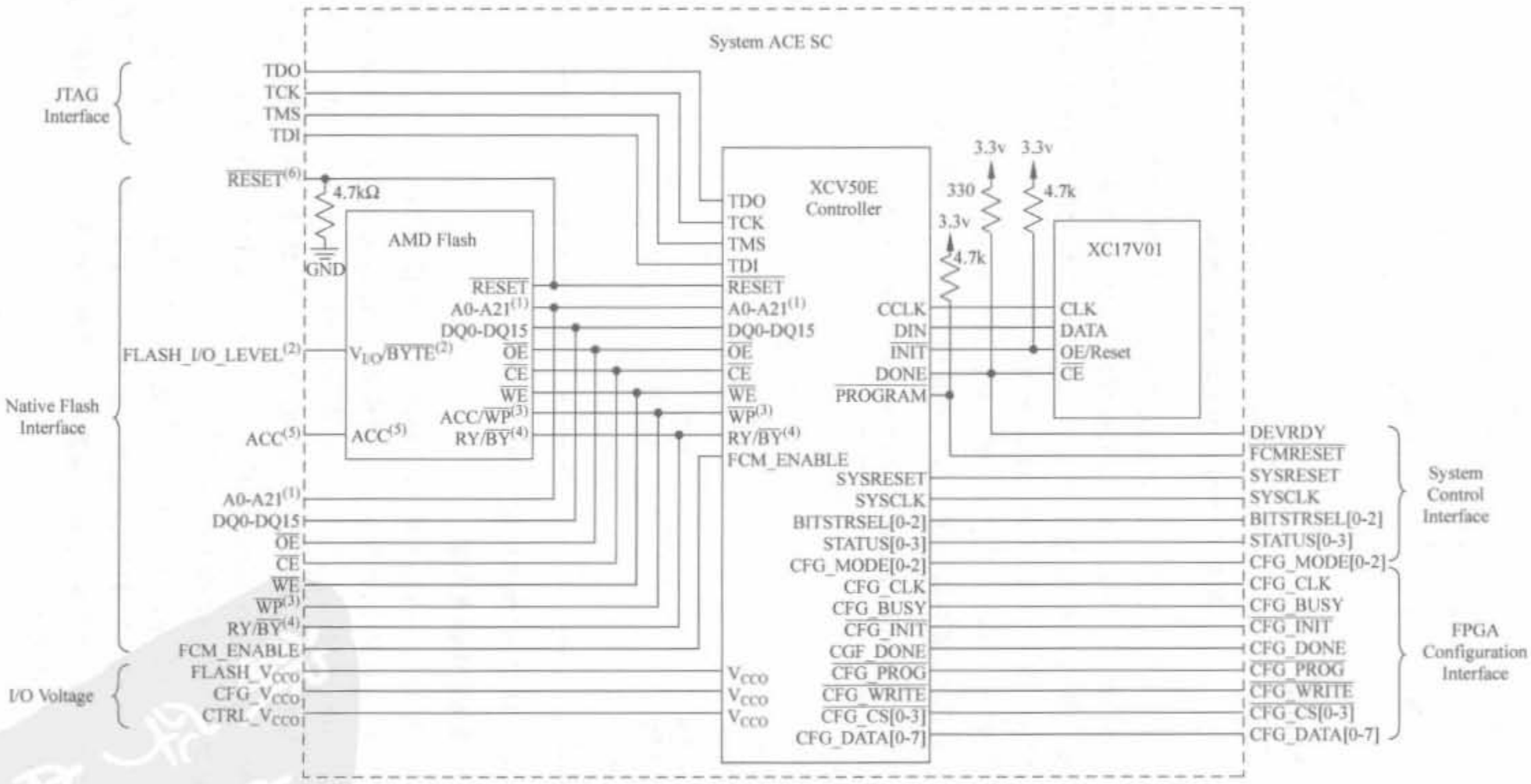


图 5-29　System ACE SC 配置电路示意图

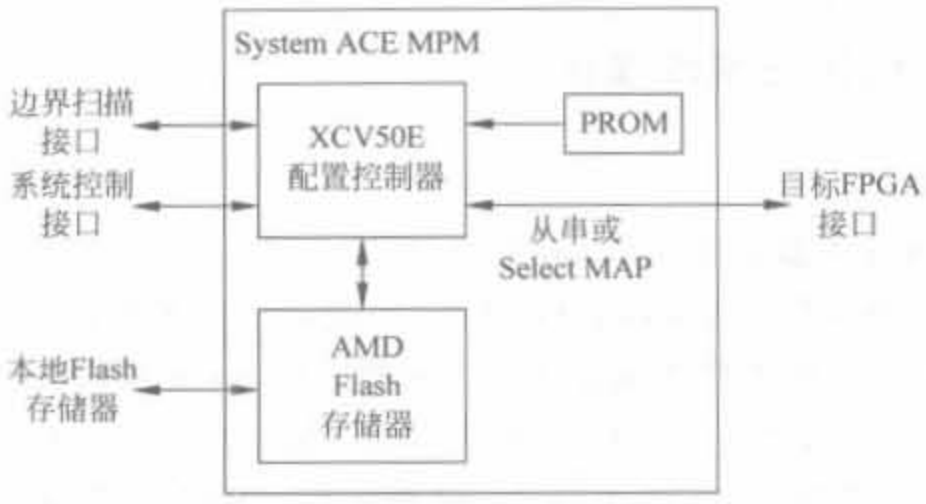


图 5-30 System ACE MPM 接口电路示意图

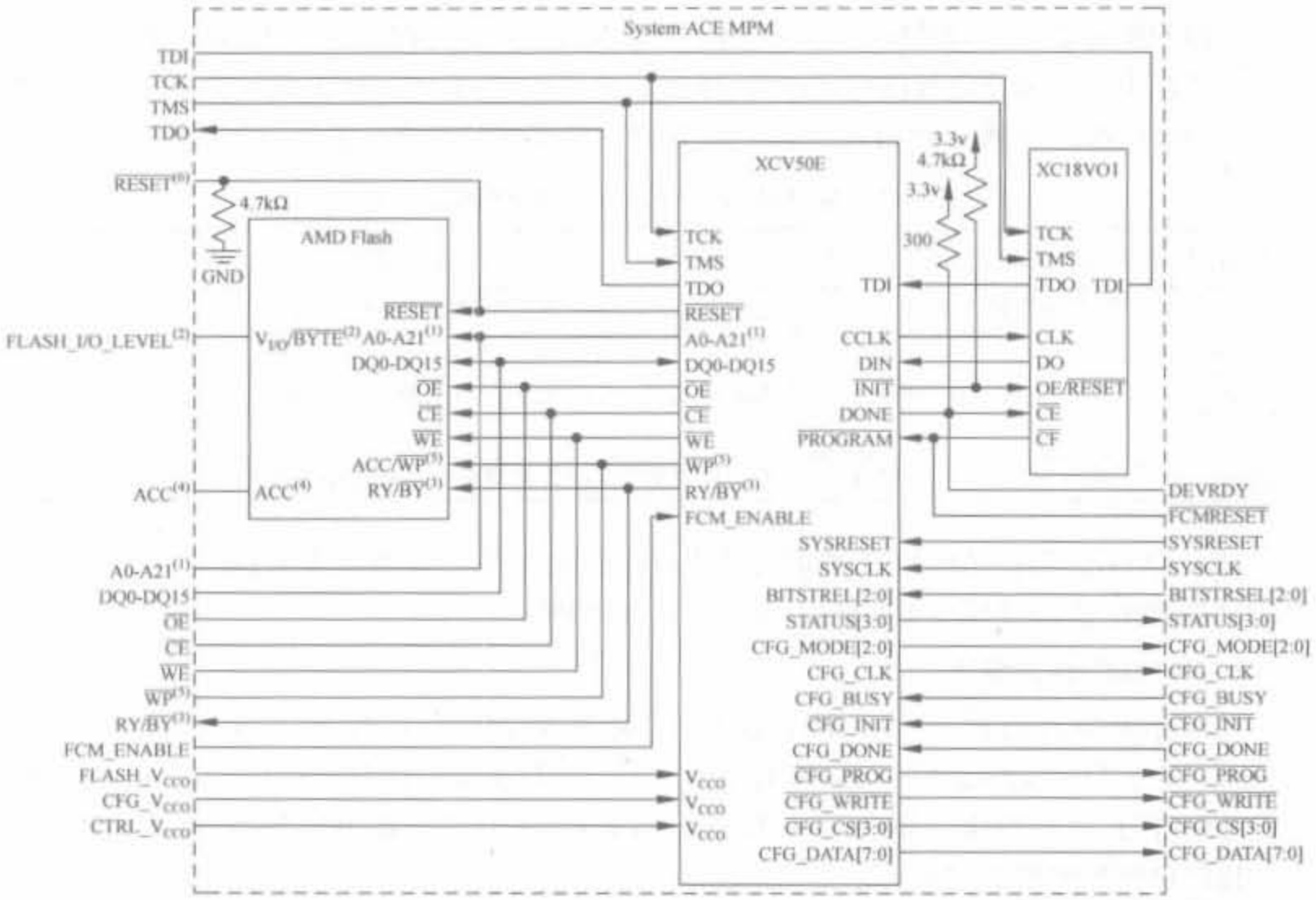


图 5-31 System ACE MPM 配置电路示意图

总之，System ACE 技术简化了大型 FPGA 系统的配置方案，使得开发人员将精力主要集中在提高系统性能和缩短开发时间方面。

5.4 iMPACT 软件使用

ISE 集成了功能强大的 FPGA 配置工具 iMPACT，Xilinx 公司的所有可编程芯片的配置过程都必须由软件 iMPACT 完成。iMPACT 能生成 PROM 各种格式的下载文件，并校验配置数据是否正确。本节主要介绍该软件的具体操作方法以及注意事项。

5.4.1 iMPACT 综述与基本操作

1. iMPACT 简介

iMPACT 支持 4 种下载模式：边界扫描、从串模式、Select Map 模式以及 Desktop 配置模式。从串模式是一种常用配置电路，可用 USB 口或并口完成配置。SelectMap 模式是一种并行配置模式，速度快，但需要使用多个信号管脚。Desktop 模式是一种高速配置模式，可配置 FPGA、PROM 以及 System ACE，但需要专用的硬件设备。在实际中，由于边界扫描模式标准统一、设备简单，且可通过 JTAG 链路配置 FPGA、CPLD 以及 PROM，使用最为广泛。因此本节主要基于边界扫描模式来介绍 IMPACT 的使用方法。

当设计完成后，ISE 调用 BitGEN 程序把布局布线后的. ncd 文件转化成. bit 文件，包括配置数据和配置指令。如果使用 JTAG 模式，可直接将. bit 文件通过 iMPACT 文件配置到 FPGA 芯片中。如果要用其他模式配置 FPGA，则需要通过 iMPACT 进行格式转化，生成. mcs、. exo 以及. hex 等文件格式。表 5-19 对常用的配置文件进行了比较和说明。

表 5-19　常用的 Xilinx FPGA 配置文件格式列表

文件后缀	字节交换方式	生成工具	文件描述
. bit	非字节交换	BitGEN	二进制文件，包含数据和配置信息，只能用于 JTAG 模式
. mcs	字节交换	iMPACT	ASCII 文件，专门用于配置 PROM，包含地址及校验和信息
. exo			
. tek			
. hex	用户定义	iMPACT	ASCII 文件，仅包含配置信息，主要用于定制配置方案

对于 FPGA 器件，iMPACT 能够直接将. bit 位流文件下载到芯片中，或者将其转换为 PROM 器件的 EXO/MCS 文件格式，并下载到 PROM 芯片中。

2. iMPACT 用户界面

有两种方法可以启动 iMPACT 软件，一种是在 ISE 过程管理区中单击“Generate Programming File”前面的“+”号，再双击“Configure Device”，在 ISE 环境下运行；另一种是通过单击“开始→程序→ISE9. 1→Accessories→iMPACT”，在 Windows 环境下单独运行。其用户界面如图 5-32 所示。

iMPACT 的用户界面主要由“File”、“Edit”、“View”、“Operations”、“Options”、“Output”、“Debug”、“Window”、“Help”菜单栏和常用工具栏组成。下面对常用的菜单栏操作进行简要介绍。

1）“File”菜单

“File”菜单包含常见的文件操作，其中的“Initialize Chain”用于自动完成边界扫描 JTAG 链上的器件类型和数目；“Export Project to CDF”用于把当前项目信息保存到 CDF (Chain Description File)文件中。

2）“Edit”菜单

“Edit”菜单包含常用的配置操作，其中的“Add Device”用于手动创建 JTAG 扫描链时添加 PROM 或 FPGA 芯片；“Assign Configuration File”用于指定配置文件；“EDIT

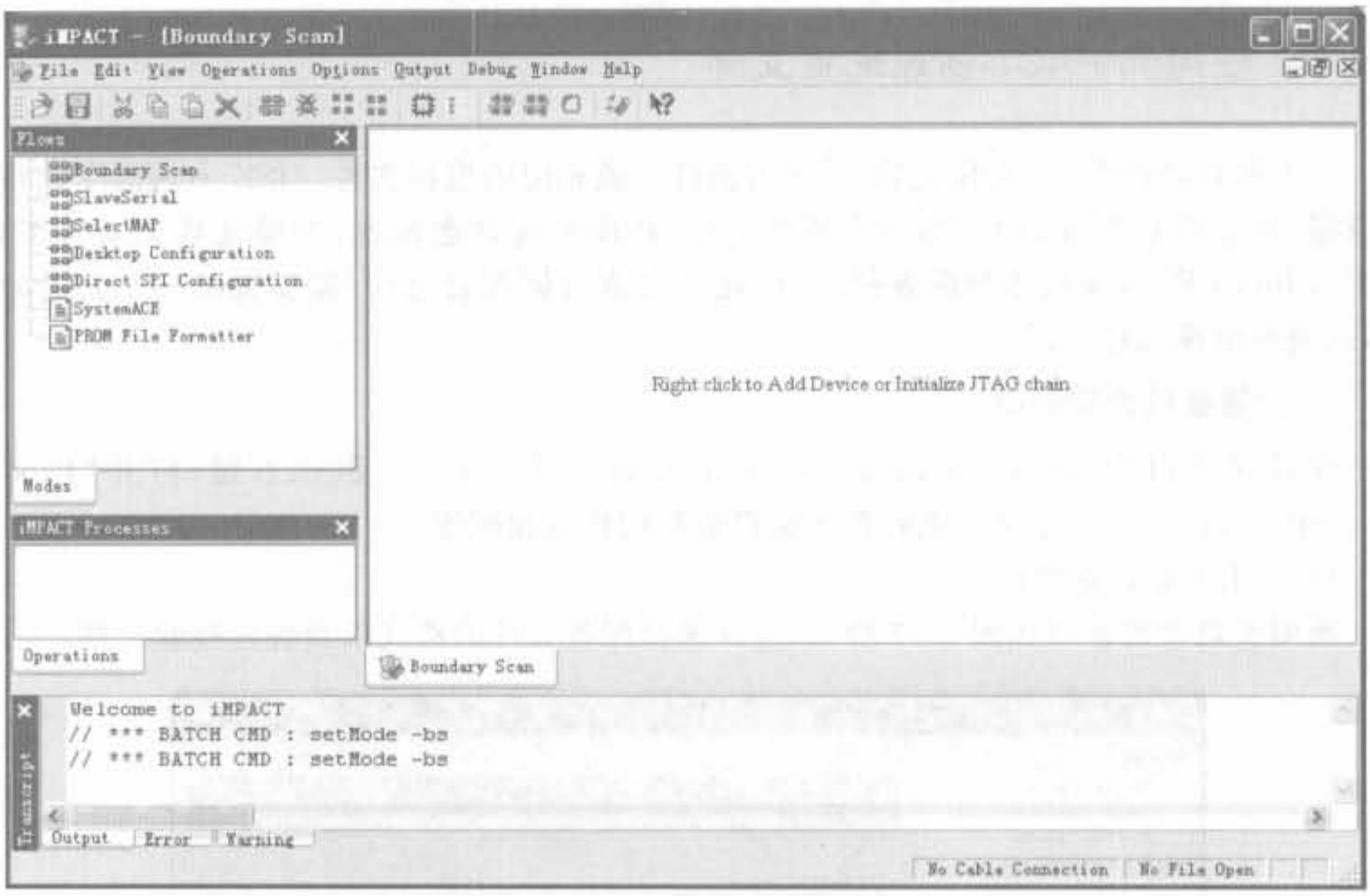


图 5-32 iMPACT 的用户界面

ROM"用于修改和删除 PROM 芯片；"Preference"用于设定 iMPACT 的通用选项。

3）"View"菜单

"View"菜单包含各个窗口显示/关闭的操作。

4）"Operations"菜单

"Operations"菜单包含配置、验证、擦除以及各类验证操作。其中"Program"用于对器件编程，下载相应的配置文件；"Verify"用于验证下载是否正确；"Erase"用于擦除 FPGA 或 PROM 芯片内的内容。

5）"Options"菜单

"Options"菜单包含编程、擦除以及回读等选项。

6）"Output"菜单

"Output"菜单包含常用的电缆操作。其中，"Cable Auto Connect"用于电缆自动连接；"Disconnect All Cables"用于断开所有电缆。

7）"Debug"菜单

"Debug"菜单包含 JTAG 扫描链所有的调试操作。其中，"Start/Stop Debug Chain"用于启动或停止调试；"Chain Integrity Test"用于下载链路完整性测试；"IDCODE Test"用于 IDCODE 测试。

8）"Window"菜单

"Window"菜单包含窗口管理操作，如关闭窗口、上一下/下一个窗口等。

9）"Help"菜单

"Help"菜单包含 iMPACT 的在线帮助和版本信息。

5.4.2 使用 iMPACT 创建配置文件

一个设计经过综合、实现之后，需要为器件生成相应的编程文件。ISE 中内嵌了比特流生成器，可生成 FPGA 以及 PROM 格式文件，从而实现动态配置，并验证数据是否正确。由于 Xilinx FPGA 支持多种配置模式，因此在完成数据配置之前，需要选择一个合适的模式，以避免出现编程错误。

1. 配置参数设置窗口

在过程窗口中，选中“Generate Programming File”并单击鼠标右键，打开“Process Properties”窗口，在其中可完成对各类编程参数的选择和配置。

1）通用参数设置窗口

通用参数设置窗口如图 5-33 所示，主要选择配置文件的格式以及各种校验规则。

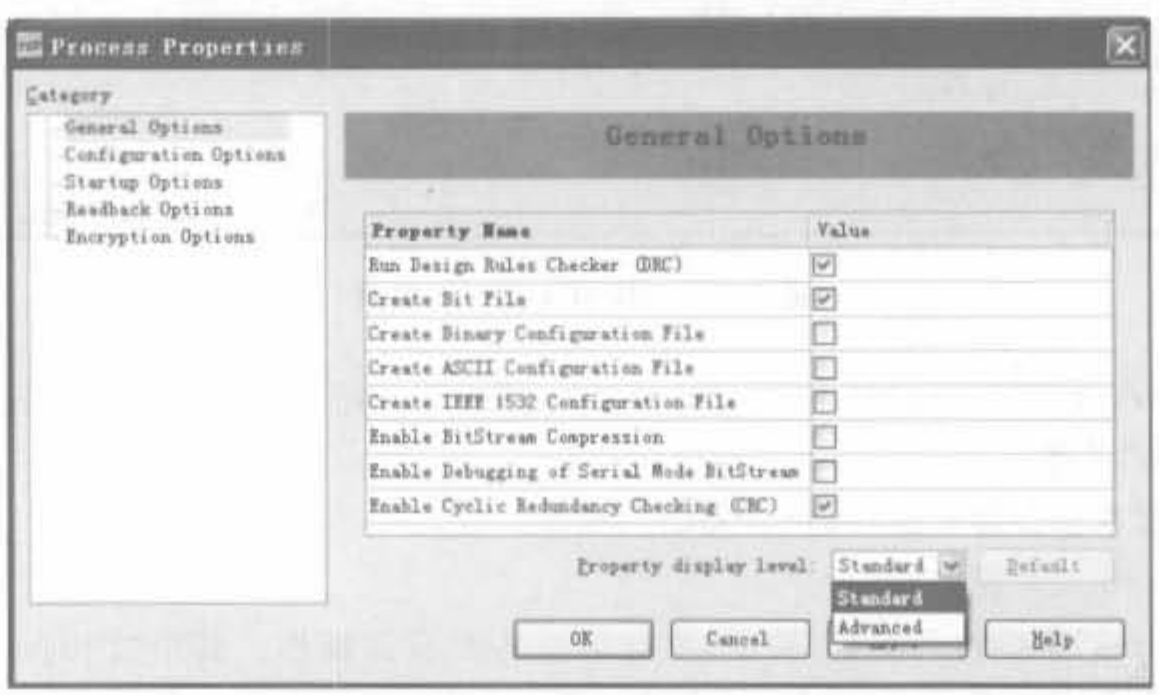


图 5-33 通用参数(General Options)设置窗口

其中，相应的选项说明如下：

(1) Run Design Rules Checker(DRC)：运行设计规则校验。建议使用该功能，在位流文件生成中进行规则校验，这样可对 NCD 文件进行评估。默认值为选中。

(2) Creat Bit File：创建位流文件。用于指示在实现后生成可配置的比特文件。默认值为选中。

(3) Create Binary Configuration File：创建二进制配置文件。默认值为不选中。

(4) Create ASCII Configuration File：创建 ASCII 配置文件。默认值为不选中。

(5) Create IEEE 1532 Configuration File：创建符合 IEEE 1532 标准的配置文件，仅对 Virtex 系列芯片有关。默认值为不选中。

(6) Enable BitStream Compression：使能比特文件压缩功能，可节约 PROM 的存储空间。默认值为不选中。

(7) Enable Debugging of Serial Mode BitStream：使能比特文件的调试功能。默认值为不选中。

(8) Enable Cyclic Redundancy Checking(CRC)：使能循环冗余校验，在配置数据中添加 4 位校验码。默认值为不选中。

2) 配置参数设置窗口

配置参数设置窗口如图 5-34 所示，主要完成配置电路所用管脚内部电阻的选择。

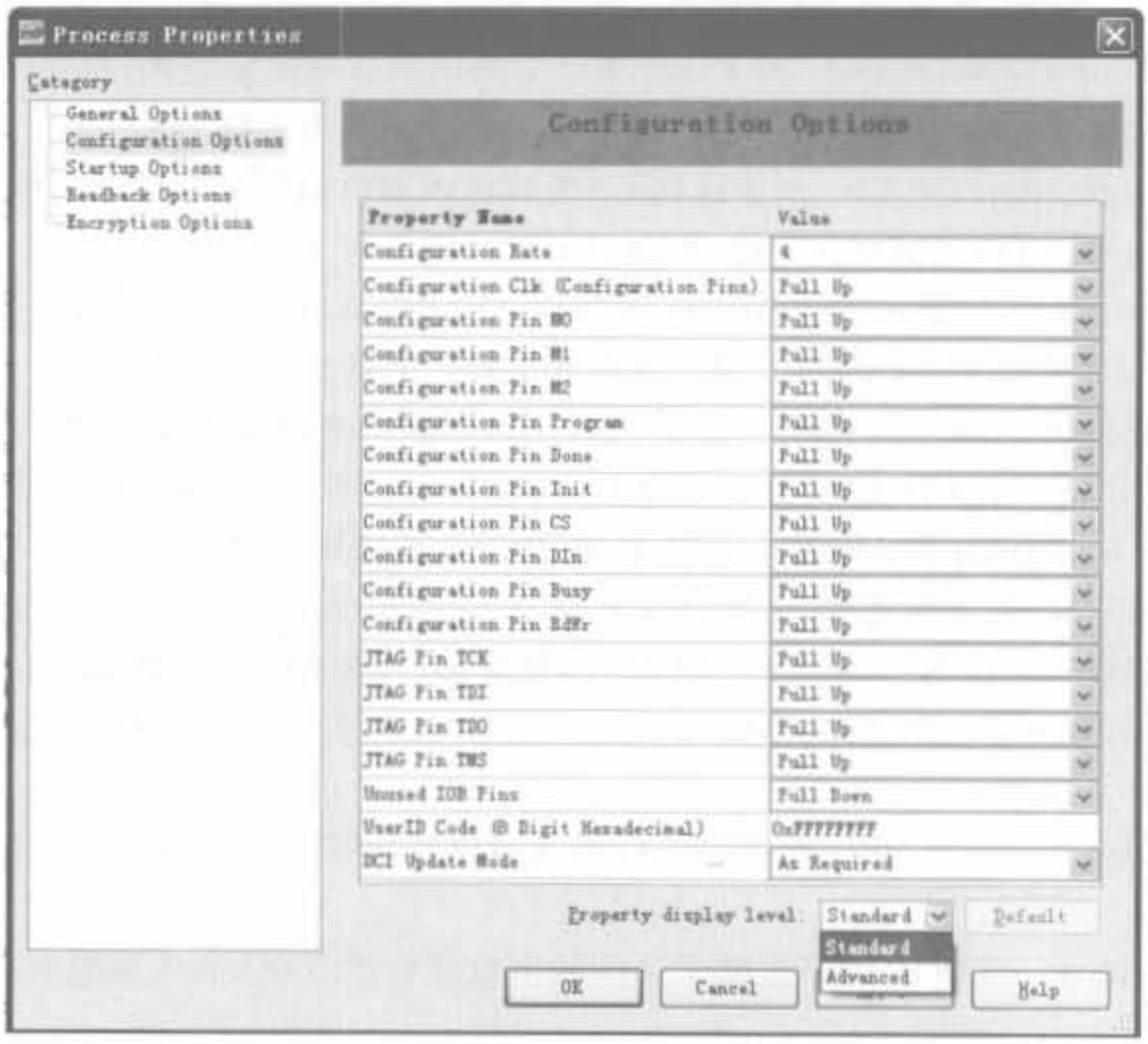


图 5-34 配置参数(Configuration Options)设置窗口

其相应的选项说明如下：

(1) Configuration Rate：配置数据速率。默认值为 4Mb/s。

(2) Configuration Clk(Configuration Pins)：用于选择配置时钟管脚 CCLK 内部是否使用上拉电阻，有“Pull Up”和“Float”两种选择。选择上拉可以减小时钟信号线上的干扰信号。默认值为选择内部上拉。

(3) Configuration Pin M0：用于选择模式控制管脚 M0 的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉，其电阻值的范围为 50～100kΩ，上拉和下拉能在一定程度上减小干扰。默认值为选择内部上拉。

(4) Configuration Pin M1：用于选择模式控制管脚 M1 的内部电阻阻值。同 M0 的说明。

(5) Configuration Pin M2：用于选择模式控制管脚 M2 的内部电阻阻值。同 M0 的说明。

(6) Configuration Pin Program：用于选择编程控制管脚 PROG 的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉。上拉和下拉能在一定程度上减小干扰，避免非法操作。默认值为选择内部上拉。

(7) Configuration Pin Done：用于选择 DONE 管脚的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉，其电阻值的范围为 2～18kΩ。由于 DONE 信号为集电极开路输出，必须有终端电阻才能正常工作。如果外部电路中没有上拉电阻，则必须选择“Pull Up”；同样，在选择“Float”时，要保证外部电路中已有上拉电阻。

(8) Configuration Pin Init：用于选择 Init 管脚的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉。默认为上拉 Pull Up。

(9) Configuration Pin CS：用于选择 CS 管脚的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉。默认为上拉 Pull Up。

(10) Configuration Pin Din：用于选择 Din 管脚的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉。默认为上拉 Pull Up。

(11) Configuration Pin Busy：用于选择 Busy 管脚的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉。默认为上拉 Pull Up。

(12) Configuration Pin RdWr：用于选择 RdWr 管脚的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉。默认为上拉 Pull Up。

(13) JTAG Pin TCK：用于选择 JTAG 时钟管脚 TCK 的内部电阻阻值，有“Pull Up”、“Float”和“Pull Down”3 种选择，分别对应上拉、悬空和下拉，建议选择内部上拉。默认值为选择内部上拉。

(14) JTAG Pin TDI：用于选择 JTAG 输入数据管脚 TDI 的内部电阻阻值，同 TCK 的说明。

(15) JTAG Pin TDO：用于选择 JTAG 输出数据管脚 TDO 的内部电阻阻值，同 TCK 的说明。

(16) JTAG Pin TMS：用于选择 JTAG 测试模式选择管脚 TMS 的内部电阻阻值，同 TCK 的说明。

(17) Unused I/O Pins：用于未用管脚的内部电阻选择，同 TCK 的说明。默认值为 FFFFFFFF。

(18) User ID Code(8 Digi Hexadecimal)：用户码身份输入，其格式为 8 个十六进制数。

(19) DCI Update Mode：用于选择设计 DCI 进行阻抗调整的模式，有“As Required”、“Continuous”和“Quiet(Off)”3 种选择，分别对应仅在需要时调整阻抗、连续调整阻抗以及达到初始后便不再调整阻抗这 3 种模式。默认值为“As Required”。

3) 配置启动参数设置窗口

配置启动参数设置窗口如图 5-35 所示，主要完成配置电路时钟信号以及时钟驱动方案的选择。

注意，图 5-38 所示的配置窗口对于不同系列的 FPGA 芯片是略有区别的。对于早期的 Virtex 和 Spartan-2 系列，还会有“Release Set/Reset(Output Events)”等选项，用于设置多少个时钟周期后，复位/置位内部锁存器、触发器。

其相应的选项说明如下：

(1) FPGA Start-Up Clock：用于选择 FPGA 芯片的配置时钟，有“CCLK”、“User

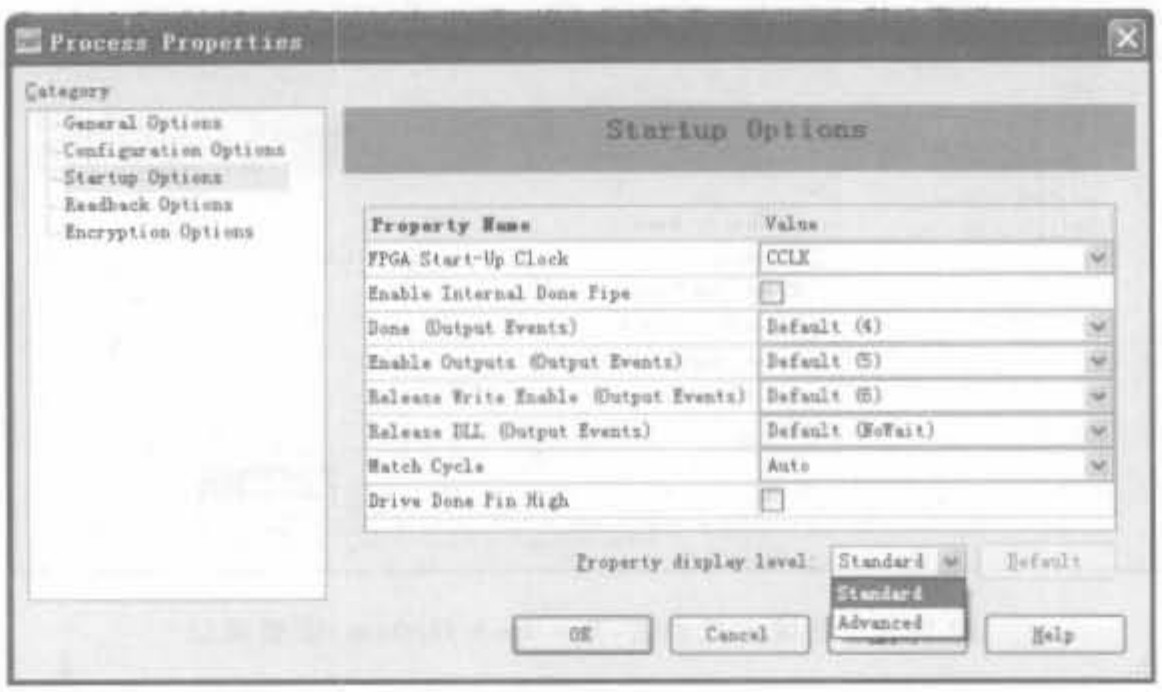


图 5-35 配置启动参数(Startup Options)设置窗口

Clock"和"JTAG Clock"3 个可选项。当配置模式为主模式时,配置时钟由 FPGA 芯片生成;当配置模式为从模式时,配置时钟由外部提供;当配置 PROM 器件时,必须选择 CCLK 时钟;当选择 JTAG 模式的配置时钟,该时钟由 JTAG 接口 TCK 信号提供。对于用户自定义的配置时钟 User Clock,目前很少使用。默认值为 CCLK。

(2) Enable Internal Done Pipe:用于选择是否等待插入的延迟信号 CFG_DONE 后,DONE 管脚有效,对于高速配置方案非常有效。默认值为不选择。

(3) Done(Output Events):用于设置多少个 CFG_DONE 周期后,使 DONE 信号有效。默认值为 4。

(4) Enable Outputs(Output Events):用于设置多少个时钟周期后,将输入、输出管脚从三态条件释放到实际的输入、输出结构。默认值为 5。

(5) Release Write Enable(Output Events):用于设置多少个时钟周期后,释放全局写信号到触发器和存储器。如果选择"Done"参数,表示当 Done 脚为高电平时,释放写使能信号;选择"Keep",用于保持当前的写使能信号。默认值为 6。

(6) Release DLL(Output Events):用于设置等待多少个时钟周期后,DLL 输出有效。默认值为"No Wait"。

(7) Match Cycle:用于设置是否等到 DCI 匹配后,再进入启动周期。默认值为"Auto"。

(8) Drive Done Pin High:用于设置是否将 Done 置高。默认值为不选中。

4) 回读方式参数设置窗口

回读方式参数设置窗口如图 5-36 所示,主要用于回读文件格式和回读模式的设置。

其相应的选项说明如下:

(1) Security:用于设置是否在回读和重新配置数据时设置保护模式,有"Enable Readback and Reconfiguration"、"Disable Readback"和"Disable Readback and Reconfiguration"3 个选项,分别对应使能回读和重新配置数据、禁止回读以及禁止回读和重新配置数据。其中,禁止回读和重配置是出于对设计保护的考虑;回读执行时,需要由

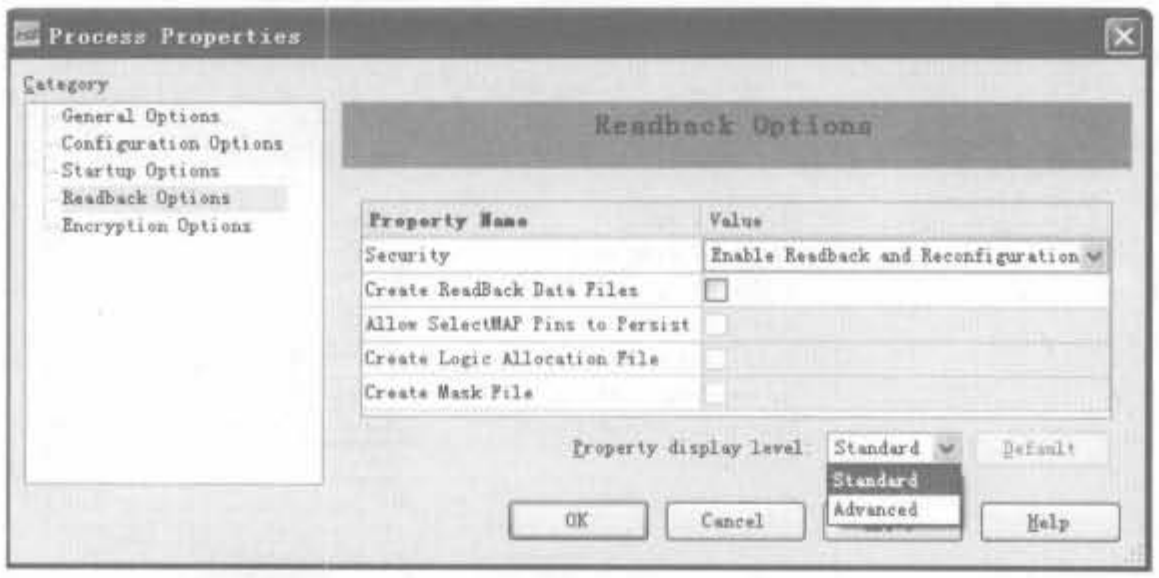


图 5-36　回读方式参数(Raedback Options)设置窗口

M0/RTRIG 脚产生一个上升沿来启动,需要一个外部的逻辑电路驱动 CCLK 时钟,以回读!RDATA 管脚上的每一位数据。

(2) Create ReadBack Data Files:用于创建回读文件。默认值为不选中。

(3) Allow SelectMAP Pins to Persist:用于选择在配置完成后是否保留 Select MAP 配置模式的配置管脚。使能该选项时,可利用其完成数据的回读;否则,当配置完成后,配置管脚将被释放,变成用户管脚。默认值为不保留配置管脚。

(4) Create Logic Allocation File:用于配置是否建立一个逻辑定位文件。该文件包含了锁存器、触发器、输入/输出管脚的位流位置和块存储器的位流位置。默认值为不选中。

(5) Create Mask File:用于配置是否选择建立屏蔽文件,用于确定位流文件中的一些位。默认值为不选择。

5) 加密参数设置窗口

加密参数设置窗口如图 5-37 所示,主要完成配置文件加密选项的设置。

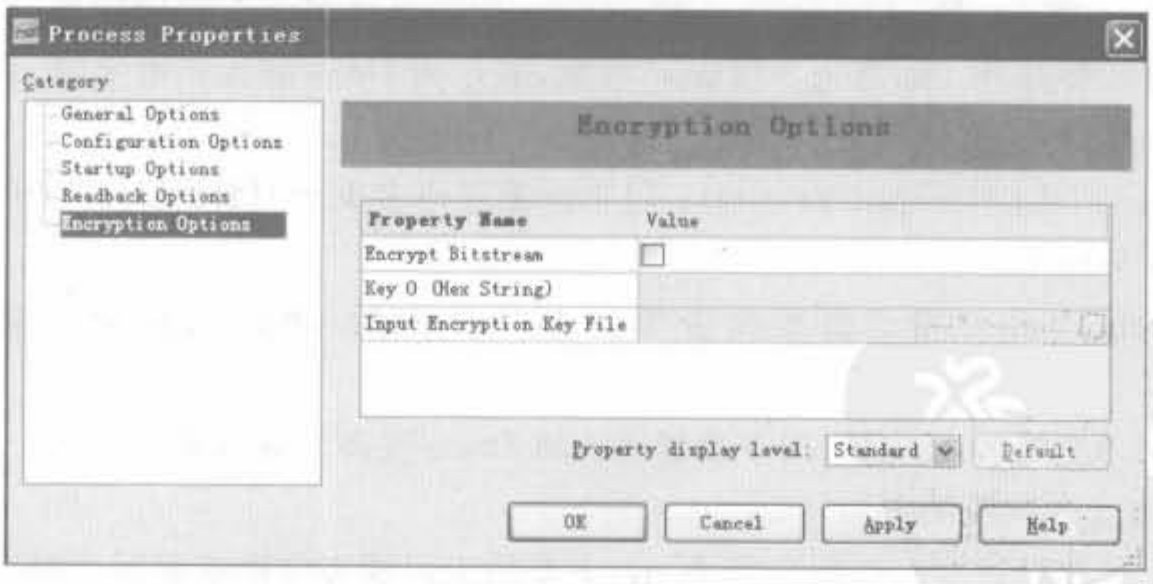


图 5-37　加密参数(Encryption Options)设置窗口

其相应的选项说明如下:

(1) Encrypt Bitstream:对比特流文件编码加密。

(2) Key 0(Hex String):用于输入十六进制加密的字符串 Key 0。

(3) Input Encryption Key File：用于加载加密文件。

2. 生成 FPGA 比特配置文件的操作

FPGA 配置文件主要用于调试阶段快速地通过 JTAG 模式配置 FPGA。断电后，芯片内的逻辑立刻消失，每次上电都需要重新配置。该操作比较简单，首先，在配置启动参数中选择配置时钟为 JTAG CLK，否则会产生警告，配置过程容易出错；其次，直接单击过程区的“Generate Programming File”即可，如图 5-38 所示。

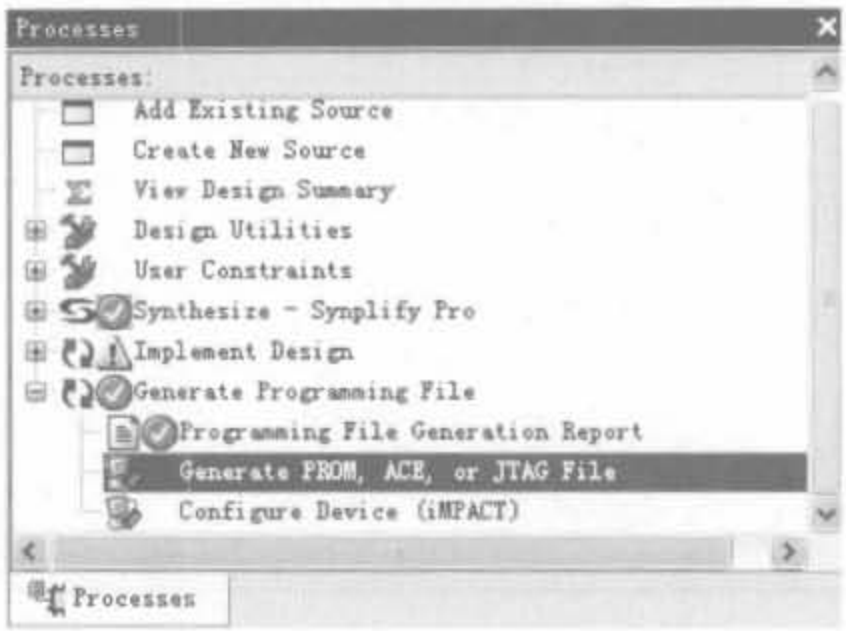


图 5-38 创建 FPGA 配置文件示意图

3. 生成 PROM 比特配置文件的操作

只有生成 PROM 文件并下载到 PROM 芯片中，才能保证 FPGA 上电后自动加载逻辑并正常工作。和生成 FPGA 配置文件相比，生成 PROM 配置文件较为麻烦，下面对其进行详细说明。

(1) 将设计经过前仿、综合、实现以及后仿，确保设计无误。单击过程管理区中“Generate Programming File”前面的“+”号，双击“Generate PROM，ACE，or JTAG File”运行文件生成工具，弹出的文件界面如图 5-39 所示。

(2) 单击“Next”按钮，进入 PROM 器件选择界面，如图 5-40 所示。下面以 Xilinx PROM 为例进行说明。选中 Xilinx PROM，在文件格式“PROM File Format”中选择“EXO”，将 PROM 配置文件的名字改成“sqrt_test”，确定 PROM 的存放位置。

(3) 单击“Next”按钮，选择 PROM 器件的型号，如图 5-41 所示。可以选中“Auto Select PROM”选项，由 iMPACT 自动选择合适的 PROM 芯片，也可以手动在“Select a PROM”选项的下拉框中选择合适的 PROM 芯片，然后单击“Add”按钮添加选中的器件。可根据需要反复多次，添加多个 PROM 芯片。此外，对于 XCF08P 以上的批 ROM 芯片，还可以使能修改和压缩功能。

(4) 单击“Next”按钮，进入 PROM 文件综合信息显示窗口，如图 5-42 所示。如果确认信息无误，单击“Finish”按钮，进入后续步骤；否则，返回前面进行修改。

(5) 单击“Finish”按钮后，弹出的配置文件加载窗口如图 5-43 所示。

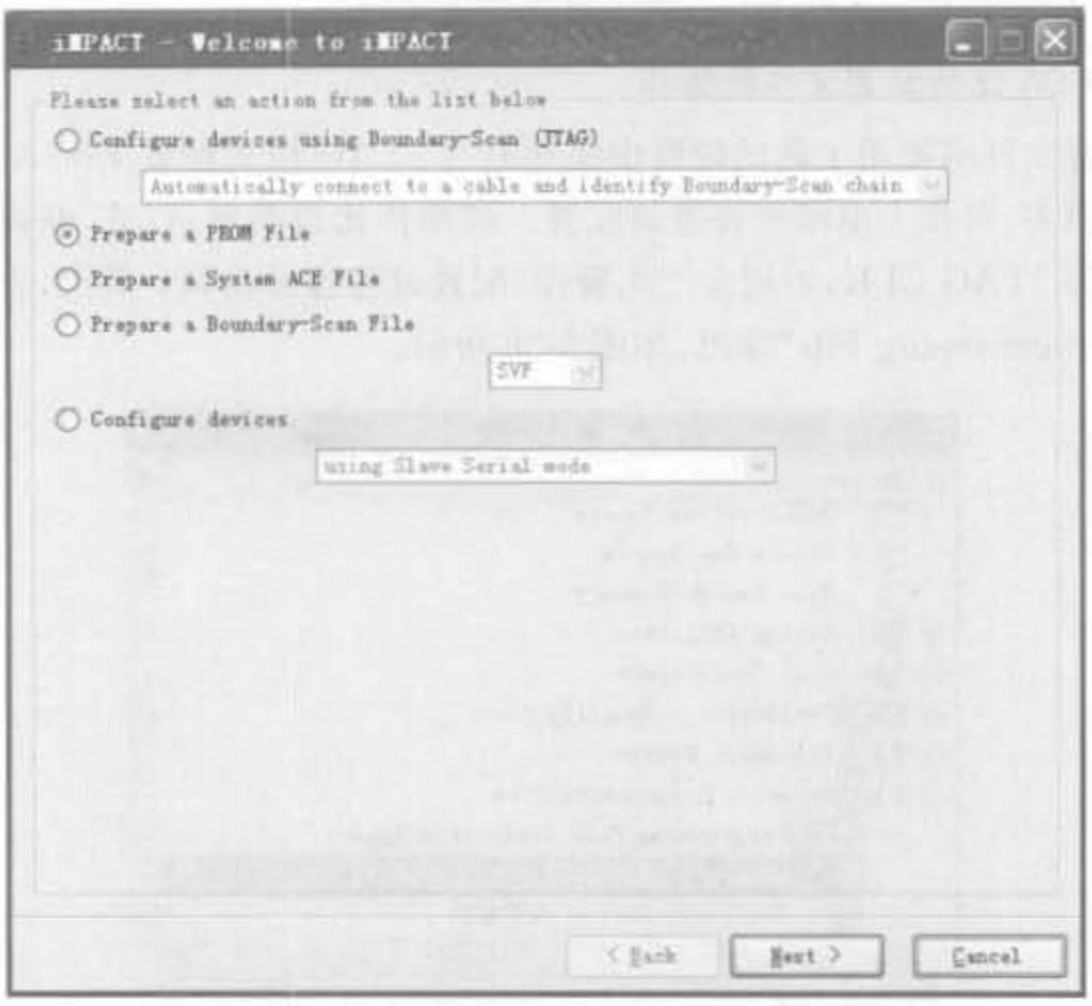


图 5-39　PROM 配置文件生成工具界面

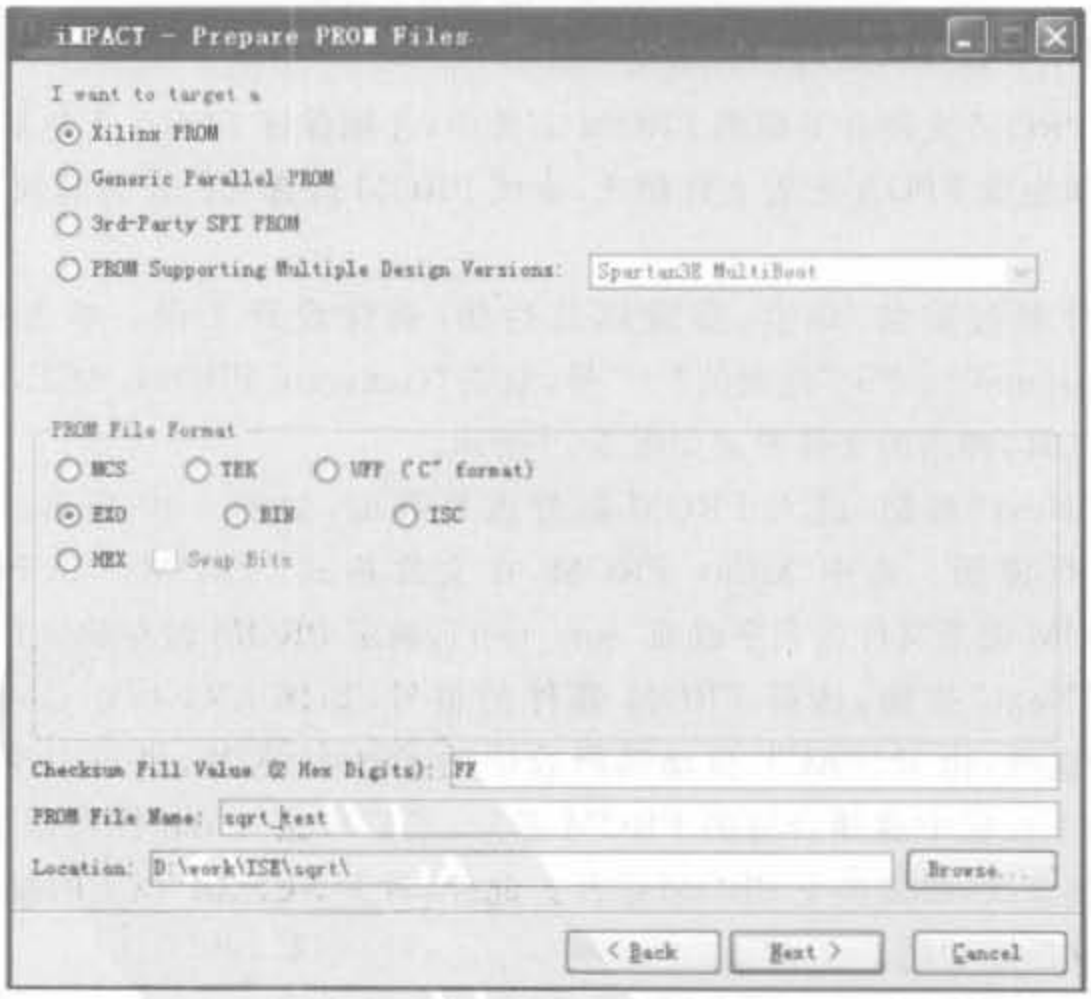


图 5-40　选择 PROM 芯片的类型和文件格式

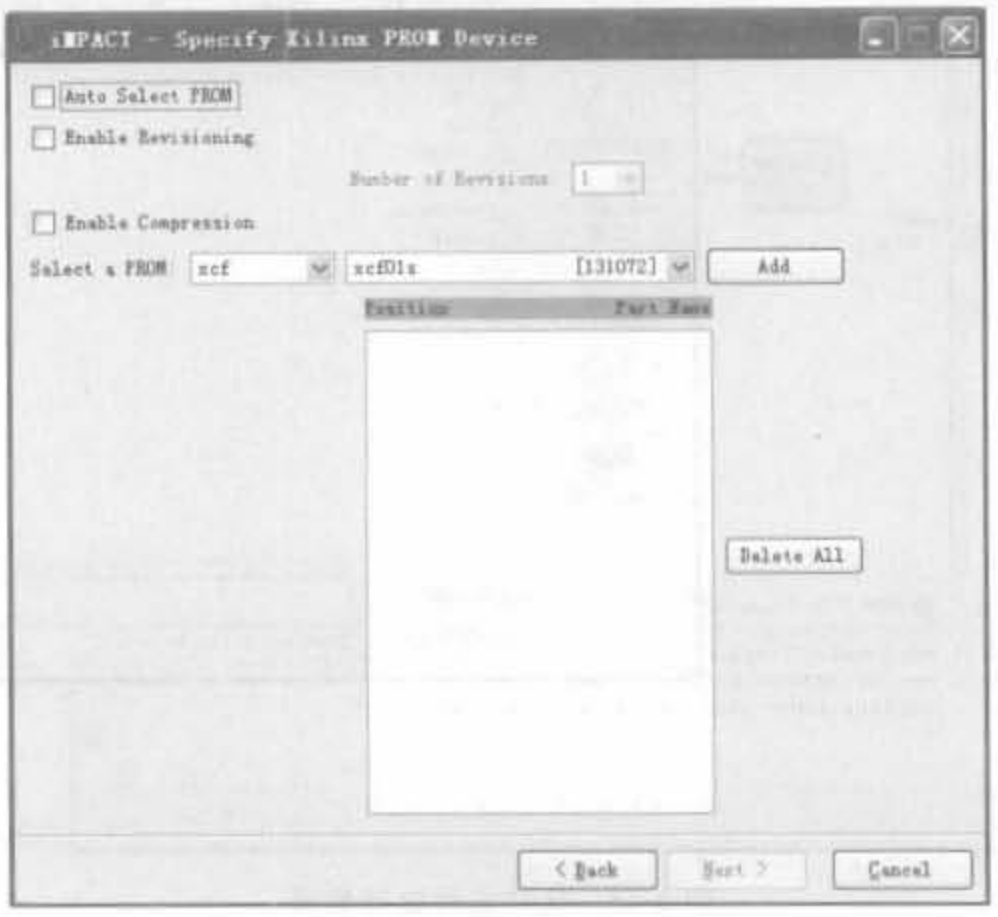


图 5-41 选择 PROM 芯片的型号

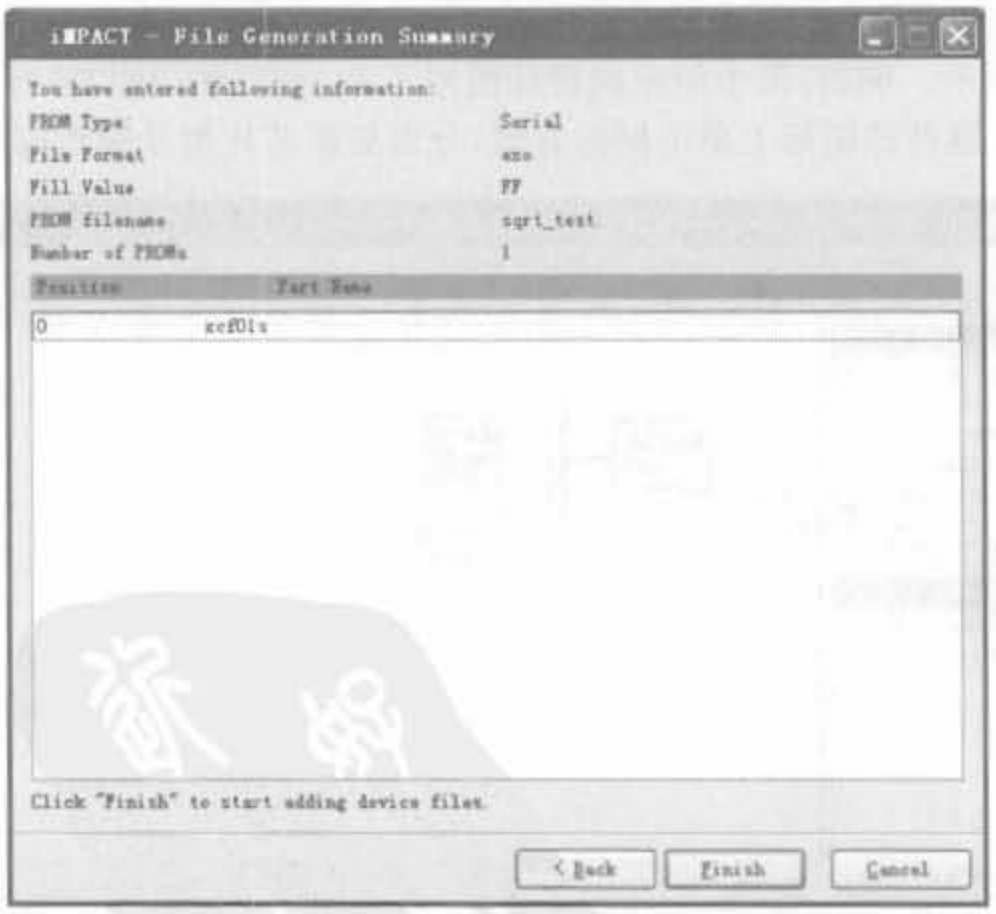


图 5-42 PROM 配置文件生成器综合信息显示窗口

图 5-43　比特文件选择界面

(6) 选择相应的文件后，单击“打开”按钮，将其加载。此时，iMPACT 会根据加载的 .bit 文件所对应的 FPGA 芯片计算 PROM 的容量。如果 PROM 容量不够，会主动提醒用户修改 PROM 型号或者添加更多的 PROM 芯片；如果容量富裕，会给出 PROM 的容量利用率，如图 5-44 所示。例如，图中给出的设计使用了 81.66%的 PROM 容量。此时，还可以在 PROM、FPGA 器件的图标上单击鼠标右键，分别更新芯片型号和相应的.bit 文件。

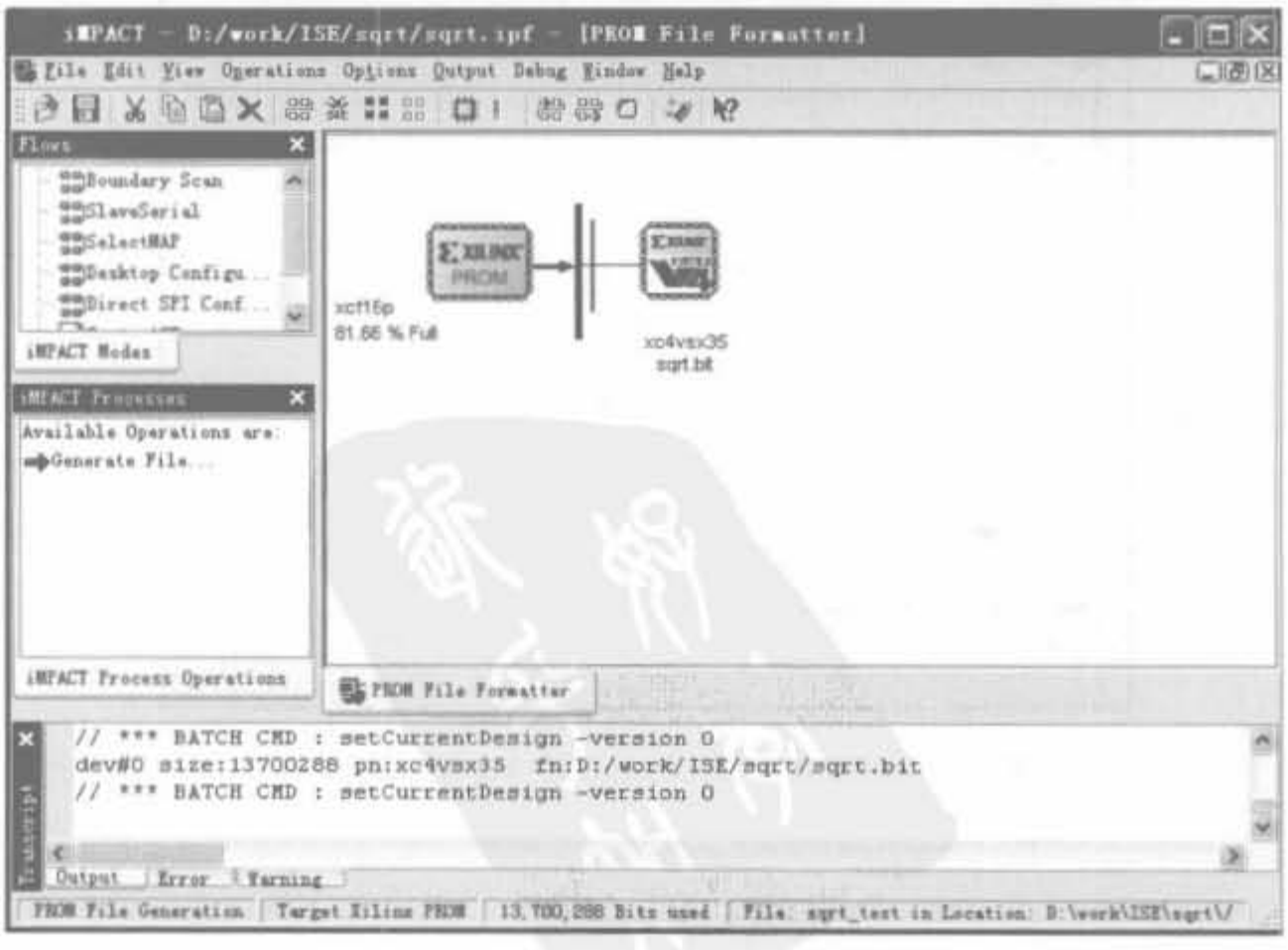


图 5-44　PROM 容量显示界面

(7) 在 iMPACT 的过程管理窗口双击“Generate File”，iMPACT 会自动创建 PROM 配置文件；或在 PROM 上单击鼠标右键，选择“Generate File”。当配置文件创建成功后，显示文件大小以及所占 PROM 的容量，并在 iMPACT 界面上显示“PROM File Generation Succeeded”，如图 5-45 所示。

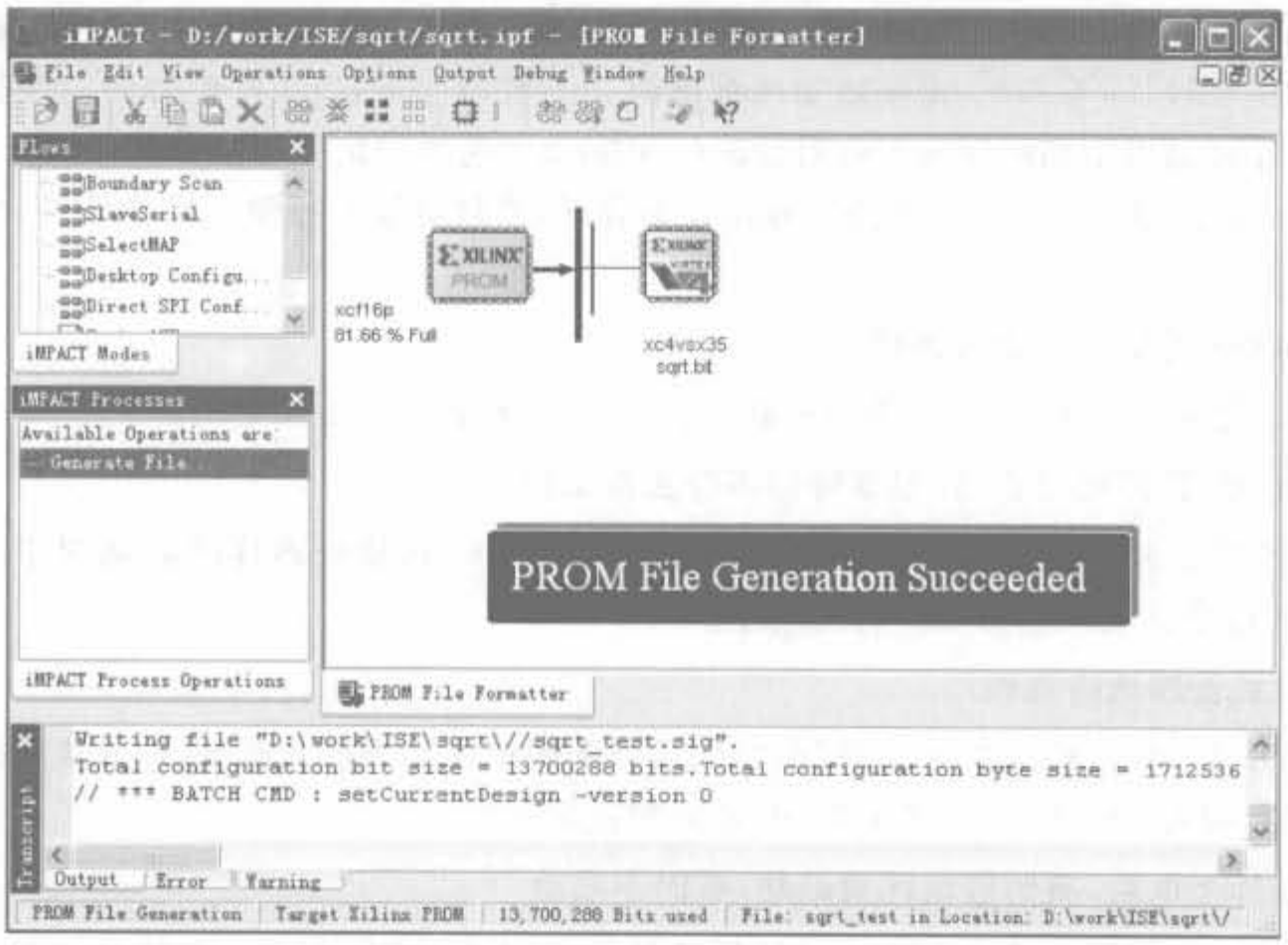


图 5-45 PROM 配置文件创建成功提示界面

5.4.3 使用 iMPACT 配置芯片

利用 iMPACT 配置芯片的操作流程见 4.3.4 节，这里不再重复说明。

5.4.4 FPGA 配置失败的常见问题

在配置 FPGA 器件时，经常会出现配置失败的情况，简单总结起来有下列几种情况，并给出相应的解决方案。

1. JTAG 链扫描失败

解决方法：首先，检查所有芯片的 TCK、TMS 管脚是否和 JTAG 接口的 TCK、TMS 连接在一起；其次，检查配置电路的 JTAG 链路是否完整，从 JTAG 接口的 TDI 到链首芯片的 TDO，…，再到链尾芯片的 TDO 是否连接到 JTAG 接口的 TDO；最后检查电源是否正确。

2. 无法通过计算机并口配置

解决方法：首先，检查计算机并口是否插好；其次，采用质量更好的并口配置电缆

(Parallel Cable-Ⅳ)或信号质量更好的 USB 配置电缆,排除下载线的问题。目前,最好采用速度更快、可靠性更高的 USB 下载线。

3. 无法正常配置

解决方法:检查配置时钟信号 CCLK 或 JTAG 时钟信号 TCK 是否存在干扰信号或过冲。如果存在干扰,判断干扰源,并增加滤波电路以消除干扰。如果有过冲,说明该信号线阻抗可能由于较长,不匹配,需要增加匹配电阻。一般情况下,CCLK 信号的引线长度不超过 10cm,还可通过增加源端匹配电阻(33~100Ω)来改善时钟信号质量。

此外,如果 FPGA 芯片的旁路电容设计不合理,或数据线上有地线反弹信号,也会导致配置失败。

4. DONE 管脚状态始终为低

解决方法:检测 DONE 管脚的负载是否太重,选择合适的上拉电阻。

5. DONE 管脚已经变高,但器件仍不能正常工作

解决方法:首先检查设计是否无误;如果设计无误,再检查器件的启动顺序。参考配置流程,通过设计工具重新设置启动顺序。

6. 模式管脚选择错误

解决方法:根据模式选择管脚 M[2:0]选择配置模式。当模式改变后,要修改位流文件中的配置时钟为 CCLK 或 TCK;否则,容易配置失败。

7. 器件上电后,有时候能配置成功,有时不成功

解决方法:这种情况经常是由于器件的复位未完成,就开始出现数据流而产生的。解决方法就是添加复位芯片,延长复位时间。

5.5 从配置 PROM 中读取用户数据

本节主要介绍如何从 Xilinx PROM 中引导嵌入式处理器(MicroBlaze、PowerPC)相关的软件代码、用户数据和配置数据,从而将所有的数据都存放于 PROM 芯片中,达到简化系统设计和降低成本的双重目的。关于嵌入式处理器本身将在第 9 章详细介绍。

5.5.1 从 PROM 中引导数据简介

目前,许多 FPGA 设计都集成了使用 MicroBlaze 和 PowerPC 处理器的软件嵌入式系统,这些设计同时使用外部易失性存储器来执行软件代码。使用易失性存储器的系统还必须包含一个非易失性器件,用来在断电期间存储软件代码。大多数 FPGA 系统都在电路板上使用 Xilinx PROM,用于在上电时加载 FPGA 配置数据。另外,许多应用还可能使用其他非易失性器件(如 SPI Flash、Parallel Flash 或 PIC)来保存 MAC 地址、系数、处理器代码以及 ASCII 数据等用户数据,因此导致系统电路板上存在大量非易失性器件。但如果只用一个 PROM 来存储 FPGA 配置数据、软件代码和用户数据,则可以节省电路板面积。

PROM 在存储多数据段时的内容如图 5-46 所示。软件应用段可处在 PROM 中的任意位置，用地址同步字标识。跟在地址同步字后面的是一个 32 位软件起始地址、32 位软件段（指定后面软件数据的字节数），然后是实际的软件数据。软件起始地址、字节数和其他软件数据可以在同一软件应用中多次重复。软件应用段的末尾用两个等于零的 32 位字标识。用户数据段由用户同步字定义，同步字之后紧随用户数据。由于任意 FPGA 配置数据、软件应用或用户数据之间的数据不确定，因此需要使用同步字。

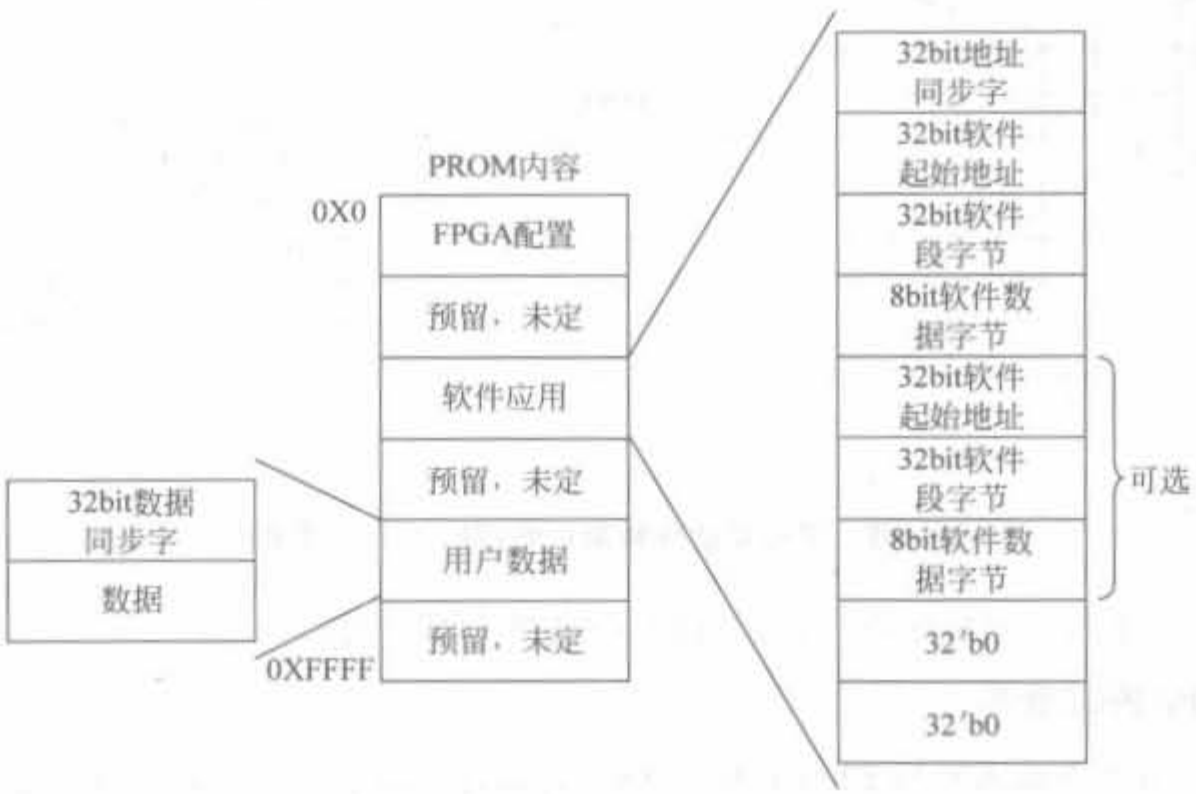


图 5-46 在 PROM 中存储多数据段的方法

5.5.2 硬件电路设计方法

在给出硬件电路之前，先介绍 PROM 的工作机制。PROM 芯片主要受控于片选（$\overline{CE}$）、复位（$\overline{RESET}$）以及输出使能信号（OE），在控制信号的组合下，完成对片内有效空间的遍历。PROM 芯片控制信号的真值表如表 5-20 所示。

表 5-20 PROM 控制信号的真值表

输入控制信号		内部地址(1)	输出
OE/$\overline{RESET}$	$\overline{CE}$		DATA
高	低	如果地址≤TC：增加	Active
		如果地址＞TC：不变化	High-Z
低	低	保持在复位状态	High-Z
高	高	保持在复位状态	High-Z
低	高	保持在复位状态	High-Z

注(1)：表中的 TC 为 Terminal Count of address counter；地址上限 CAC 为 Current Address Count，指当前地址寄存器的值。

如前所述，在传统的配置电路中，$\overline{CE}$管脚通常与 FPGA 的 DONE 管脚连接，在完成 FPGA 配置后，FPGA 会使 PROM 的片选信号无效。因此，为了能在配置 FPGA 之后继续

读取 PROM 内容，设计系统电路板时必须对配置电路进行一定的修改。例如，可引导软件和用户数据的主串配置电路如图 5-47 所示。

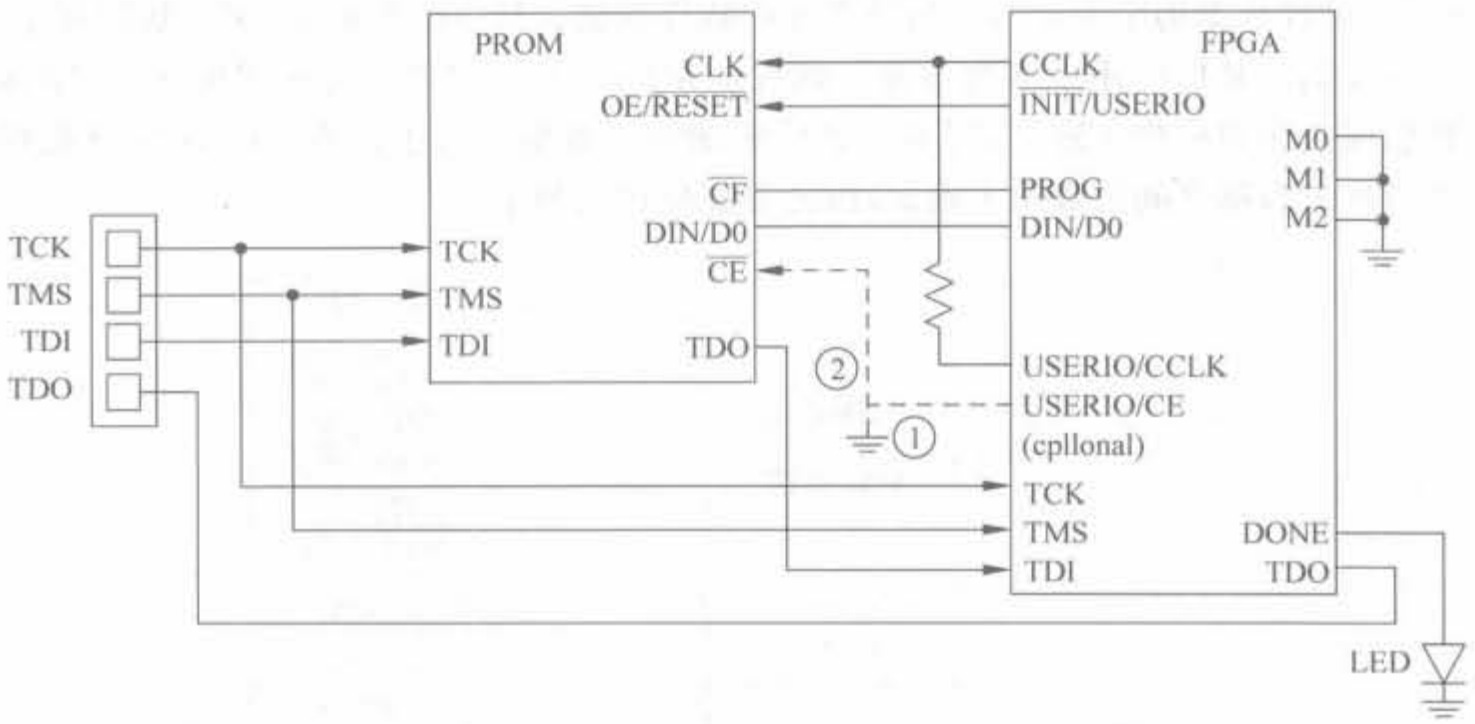


图 5-47　可引导软件和用户数据的主串配置电路

对于图 5-47 所示的配置电路，有下面几个细节需要说明。

1. PROM 的$\overline{CE}$管脚

PROM 的$\overline{CE}$管脚通常与 FPGA 的 DONE 管脚连接，用于在配置 FPGA 之后将 PROM 保持在待机模式。用户可以用此管脚启用或禁用 PROM，并且在不准备访问 PROM 时降低功耗。不过，如果连接了 DONE 管脚和 PROM 的$\overline{CE}$管脚，就不能在配置 FPGA 之后读 PROM。因此有两种方法来连接 PROM 的$\overline{CE}$管脚：①将$\overline{CE}$管脚直接接地，一直片选 PROM 芯片；②将$\overline{CE}$管脚连接到 FPGA 的通用 I/O 管脚，当 FPGA 配置时，其默认为低电平，片选 PROM；FPGA 配置完成后，当需要读取 PROM 数据时，可通过通用 I/O 片选 PROM，否则拉高该管脚，关闭 PROM 芯片。虽然后者需要为解决方案附加一个 I/O 管脚，但允许将 PROM 置为待机模式以降低功耗。由于 Xilinx PROM 的最大待机电流是 1mA，而其最大工作电流是 10mA，所以在某些对功耗敏感的应用中，通过软件驱动此管脚来启用或禁用 PROM，可节省一定的功耗。此时，由于不能将 FPGA 的 DONE 管脚连接 PROM 的$\overline{CE}$管脚，可直接连接至外部 LED，以显示 FPGA 配置完成的状态。

2. PROM 的 CLK 管脚

因为在任何主配置模式下，FPGA 生成的配置时钟 CLK 都会在成功配置 FPGA 之后停止翻转，以防止 PROM 的地址计数器的操作超出 PROM 中存储的 FPGA 设计，因此 PROM 的 CLK 管脚也需要连接到 FPGA 的一个附加用户 I/O 管脚，用来驱动 PROM 的 CLK 输入。当在配置 FPGA 之后读 PROM 时，附加的用户 I/O 为 PROM 的 CLK 管脚提供时钟。此连线上有一个 390Ω 的电阻，以避免 CLK 信号的两个可能的驱动器之间发生冲突。

3. PROM 的 OE/RESET 管脚

将 PROM 的 OE/RESET 管脚连接至 FPGA 的 INIT 管脚，以便在配置过程发生 CRC 错误时让 FPGA 重新开始配置。INIT 管脚在配置后成为一个用户 I/O，因此可以配置成输出逻辑“高”信号，以使 PROM 的输出保持有效。

4. PROM 的 DIN/D0 管脚

将 FPGA 的 DIN/D0 管脚连接到 PROM 的 DIN/D0 管脚，以便将 PROM 中的数据读入 FPGA。这不是此应用的专用连接，DIN 管脚在配置后不可用作普通用户 I/O。

对于其他配置模式，在硬件电路设计时都需要注意以上 4 点。

5.5.3 软件操作流程

在 FPGA 配置完成后，嵌入式处理器可通过通用 I/O(GPIO)外设来读取 PROM 中的用户数据。本节以 MicroBlaze 为例说明如何完成软件端的操作。

在 MicroBlaze 软核的片上外设总线(OPB)上添加具有 4 个通用输入/ 输出(GPIO)的核，如图 5-48 中的 Read PROM 模块。该 GPIO 核用来控制 INIT、CE、OE 和 DIN 管脚，需要使用 26 个四输入查找表(LUT)和 61 个触发器。另外，此参考设计还使用一个定制 OPB Block RAM 接口控制器核，其中只用一个块 RAM 来说明最小系统。EDK 9.1i 中的最小系统始终使用 8 个块 RAM。在不需要这么多块存储器的优化系统(如引导加载器)中，可使用一个定制块 RAM 接口控制器核来创建具有一个块 RAM 的系统。

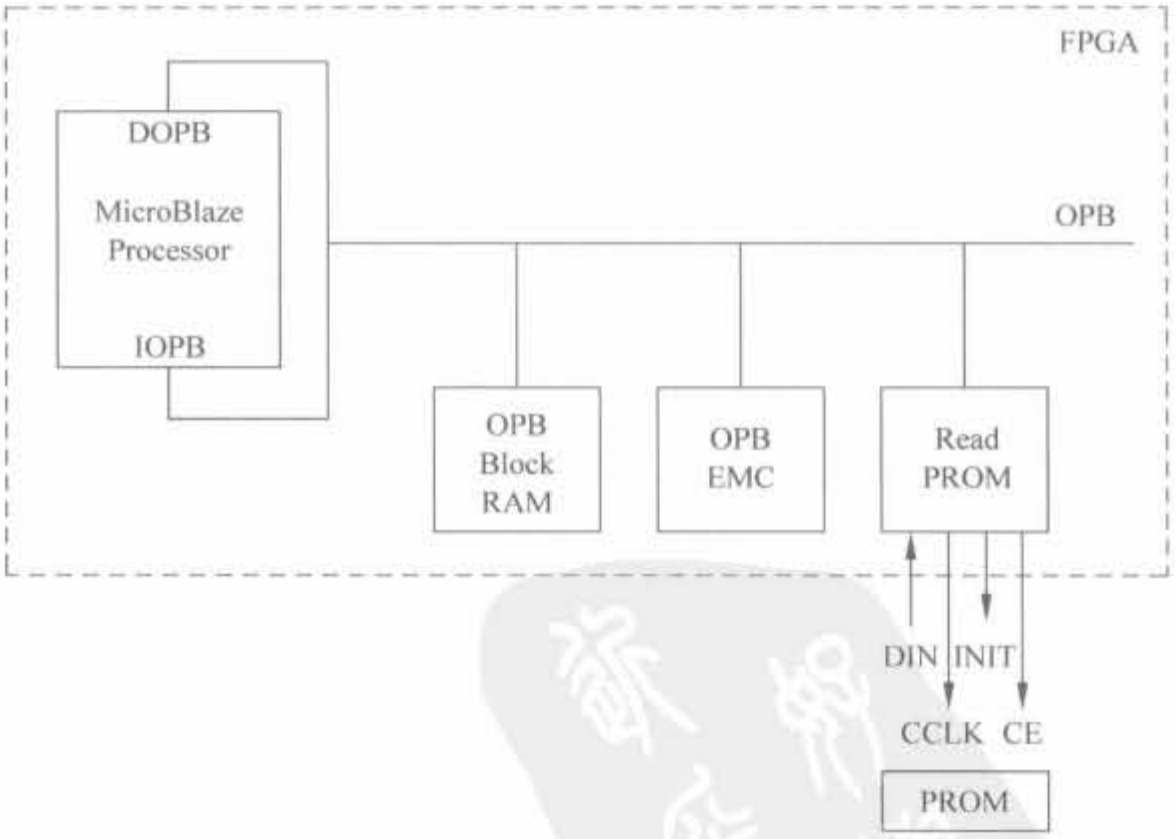


图 5-48 MicroBlaze 硬件系统框图

PROM 的控制通过 C 软件程序进行处理。完成了上述软件架构后，还需要注意驱动软件的基本原理、细节以及读取 PROM 的方法。由于本节主要介绍硬件电路，关于软件部分的设计细节请参见参考文献[8]。

5.6 本章小结

本章详细介绍了FPGA配置电路的基础知识、硬件电路以及软件操作。首先介绍了配置系统的组成以及Xilinx的下载线。其次，介绍了常用的配置电路。为了扩大FPGA的应用场合，Xilinx提供了多类配置模式，包括主串、主并、从串、从并、基于各类Flash的配置模式。在主模式以及用于测试的JTAG下，FPGA自身产生配置时钟；在从模式下，配置时钟由外部处理器提供。然后，介绍Xilinx配置软件iMPACT的使用方法以及各属性说明。最后，给出了如何读取配置PROM芯片中用户数据的软、硬件实现方法。配置电路是FPGA设计的重中之重，读者需要经过一定的实践才能更好地掌握。

第6章

在线逻辑分析仪ChipScope的使用

CHAPTER 6

逻辑分析仪(Logic Analyzer)是 FPGA 调试阶段不可缺少的工具,但是传统逻辑分析仪有两个弊端:首先价格昂贵;其次需要使用大量探头,不仅不合实际,且操作麻烦。Xilinx 公司为了解决用户的这两个难题,推出了在线逻辑分析仪(ChipScope Pro),通过软件方式为用户提供稳定和方便的解决方案。该在线逻辑分析仪不仅具有逻辑分析仪的功能,而且成本低廉、操作简单,因此具有极高的实用价值。ChipScope Pro 既可以独立使用,也可以在 ISE 集成环境中使用,非常灵活。本章主要介绍 ChipScope Pro 的使用方法以及工程案例。

6.1 ChipScope 介绍

6.1.1 ChipScope Pro 简介

ChipScope Pro 将逻辑分析器、总线分析器和虚拟 I/O 小型软件核直接插入到用户的设计当中,可以直接查看任何内部信号或节点,包括嵌入式硬或软处理器。信号在操作系统速度下或接近操作系统速度下被采集,并从编程接口中引出,再将采集到的信号通过 ChipScope Pro 逻辑分析器进行分析,从而为设计解放了更多的管脚。

ChipScope Pro 可以分析任何内部 FPGA 信号,包括嵌入式处理器总线;在设计采集或综合之后,插入小型的、可配置的软件核,将管脚影响降至最低;在板上以达到或接近目标工程运行的速度验证 FPGA 设计;利用 FPGA 的可重编程性能,可以在几分钟或几小时内确定设计问题并修改设计;内置的软件逻辑分析器可以用来识别设计问题并进行调试,包括高级触发、过滤和显示选项,无需重新综合即可改变探针指向;可利用远程调试(从办公室到实验室,或在全球范围内)通过互联网连接进行调试。此外,它还包括 Agilent 科技公司推出的、用于实现功能强大的验证功能的逻辑分析器可选配件,可以探测包括从 FPGA 内部到板上任何地方的交叉互联信号。

ChipScope Pro 为用户提供方便和稳定的逻辑分析解决方案,支持 Spartan 和 Virtex 全系列 FPGA 芯片,其典型的工作模式如图 6-1 所示。

ChipScope Pro 分析工具对 PC 和芯片之间的 JTAG 通信电缆有一定的要求,目前支持下面 4 类:

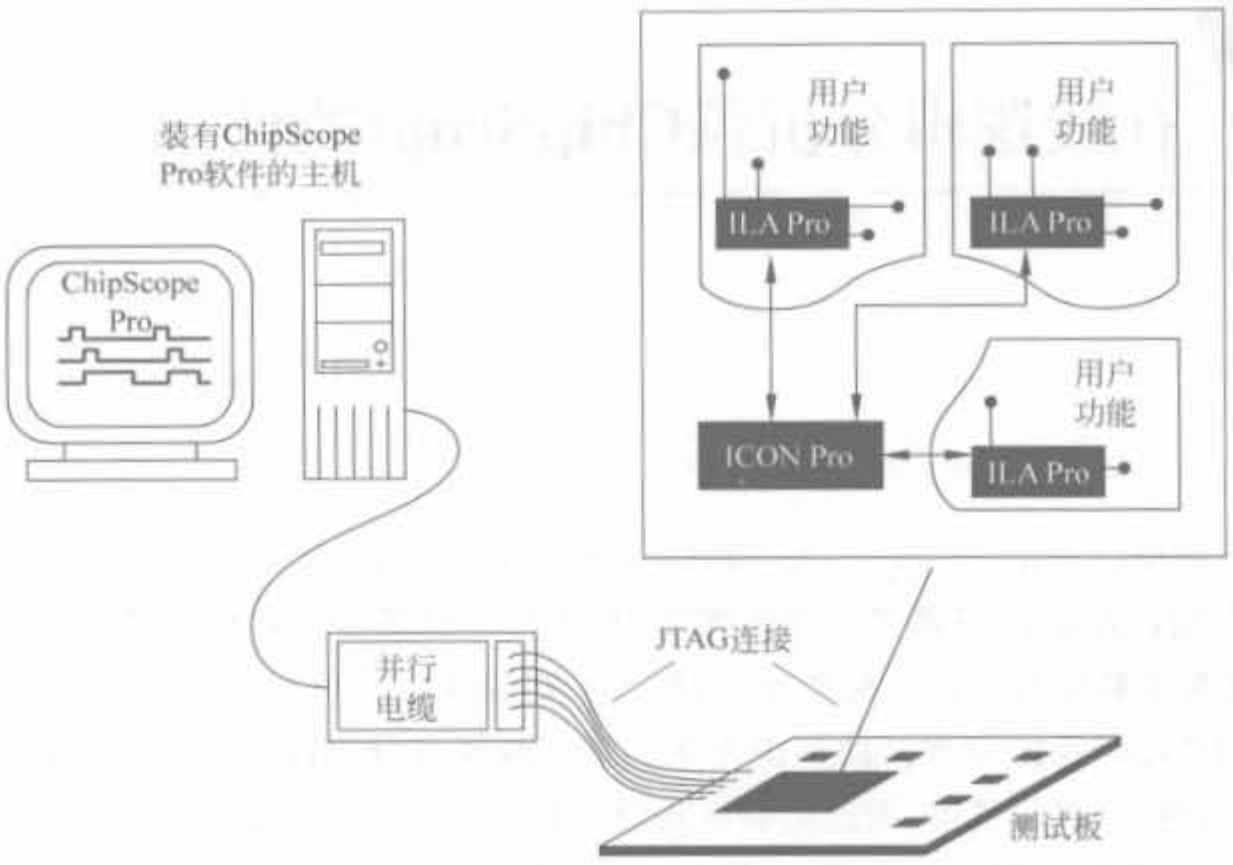


图 6-1 ChipScope Pro 的典型工作模式示意图

- Platform Cable USB
- Parallel Cable Ⅳ
- Parallel Cable Ⅲ
- MultiPRO (JTAG mode only)

ChipScope Pro 软件由 ChipScope Pro 核生成器(ChipScope Pro Core Generator)、ChipScope Pro 核插入器(ChipScope Pro Core Inserter)、ChipScope Pro 分析仪(ChipScope Pro Analyzer)以及 ChipScope Tcl 脚本接口(ChipScope Tcl Scripting Interface)4 个组件组成,支持普通 FPGA 设计以及基于 FPGA 的嵌入式、SOC 系统,其具体功能如表 6-1 所示。

表 6-1 ChipScope Pro 组件的功能简介

组件名称	功能描述
核生成器	提供下列网表和实例模板: • 集成控制核(Integrated Controller Pro core,ICON) • 集成逻辑分析仪核(Integrated Logic Analyzer Pro core,ILA) • 适用于处理器外设总线的集成总线分析核(On-Chip Peripheral Bus core,OPB/IBA) • 适用于处理器本地总线的集成总线分析核(Processor Local Bus core,PLB/IBA) • 虚拟输入、输出核(Virtual Input Output core,VIO) • 安捷伦跟踪核(Agilent Trace Core 2,ATC2) • 集成的误比特率测试核(Integrated Bit Error Ratio Tester core,IBERT)
核插入器	自动将 ICON、ILA 以及 ATC2 等核插入到用户经过综合的设计中
分析仪	完成 ILA、IBA/OPB、IBA/PLB、VIO 以及 IBERT 等核的芯片配置、触发设置以及跟踪显示等功能。其中,不同的核提供不同的触发、控制以及跟踪捕获能力。例如,ICON 核就能完成和专用边界扫描管脚的通信
Tcl 脚本接口	通过 Tcl 脚本语言和 JTAG 链,完成与芯片的交互通信

设计人员也可以直接将 ICON、ILA 以及 ATC2 等核直接插入到设计的综合网表中，然后通过实现工具完成布局布线，将生成的比特文件下载到芯片中，从而实现在线逻辑分析器。

6.1.2 ChipScope Pro 软件的安装

1. 系统要求

安装 ChipScope Pro 的硬件要求如下：

- IBM 兼容机型，主频 1GHz 以上；
- 内存 1024MB 或者更高；
- 100MB 以上的可用硬盘空间。

注意，由于运行 ChipScope Pro 需要消耗大量的资源，为了节省时间，建议使用 2048MB 以上内存，CPU 主频高于 2.4GHz。

ChipScope Pro 支持的操作系统如表 6-2 所示。

表 6-2 ChipScope Pro 支持的操作系统列表

操作系统	版　本
Windows	Windows 2000 SP2 Windows XP Professional edition
UNIX	Solaris 2.8(32bit) Solaris 2.9(32bit)
Linux	Red Hat Enterprise Linux WS3(32bit、64bit) Red Hat Enterprise Linux WS4(32bit、64bit)

此外，在安装 ChipScope Pro 时，必须确保系统已安装相应版本的 ISE 集成开发环境。下面以在 Windows 环境中的 ChipScope Pro 9.1 版本为例介绍其安装流程。

2. 软件安装

用户可以从 Xilinx 网站(www.xilinx.com)上直接下载其安装文件 ChipScope_Pro_9_1i_pc.exe，然后以管理员身份登录 PC 系统，具体的步骤如下：

(1) 双击 ChipScope_Pro_9_1i_pc.exe，弹出的欢迎界面如图 6-2 所示。单击“Next”按钮，进入下一步即可。

(2) 对弹出的软件授权界面，单击“Yes”按钮；选择合适的安装路径后，就进入注册 ID 输入界面，如图 6-3 所示。

注册 ID 是一个 16 位的数字，可在 Xilinx 网站输入产品 ID 后自动生成。输入 ID 后，单击“Next”，直至单击“Finish”按钮，完成安装。

图 6-2　ChipScope 软件安装的欢迎界面

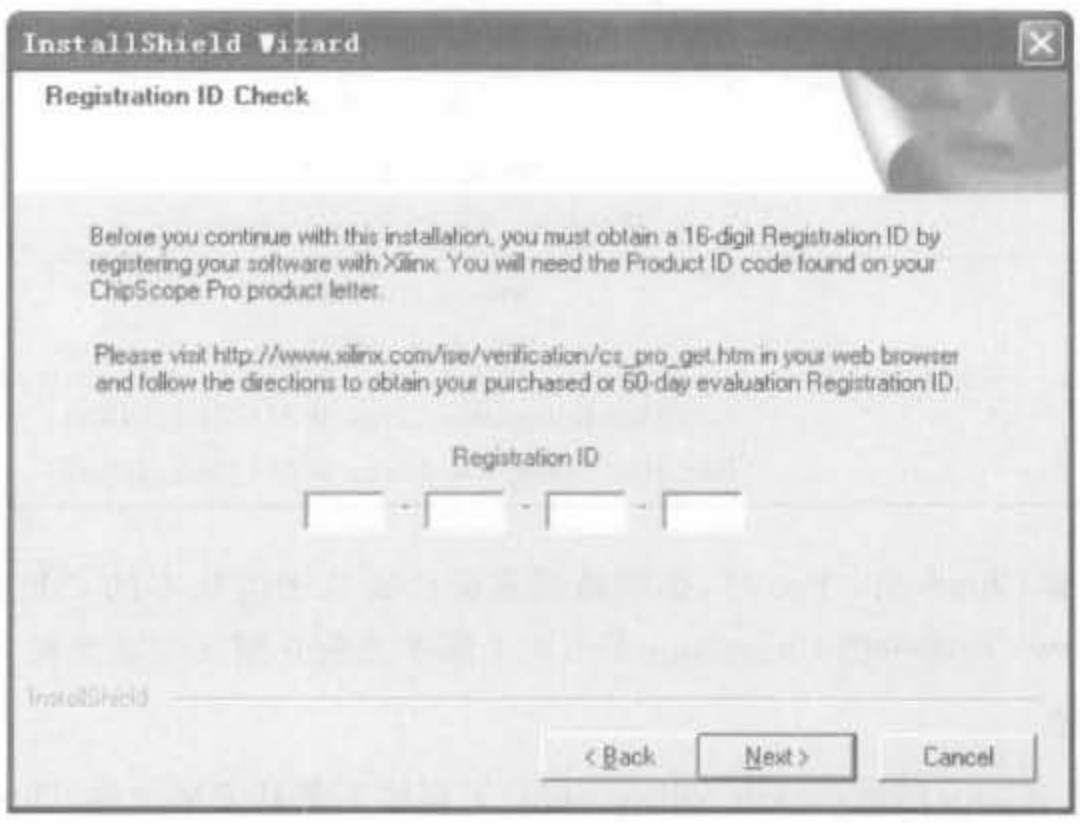


图 6-3　注册 ID 输入界面

6.1.3　ChipScope Pro 的使用流程

ChipScope Pro 的开发流程如图 6-4 所示。首先，要生成系统控制模块的 ICON 核；然后，生成各类逻辑分析核(ILA 核、VIO 核以及 ATC2 核等)，设定触发以及数据线宽度和采集长度，并将其和 ICON 核关联起来；接着，完成设计以及相关核的综合，将设计中期望观察的信号和分析核的触发以及数据信号连接起来；紧接着，完成整体系统的实现并下载到芯片中；最后，打开 ChipScope Analyzer 设定触发条件，观察波形。

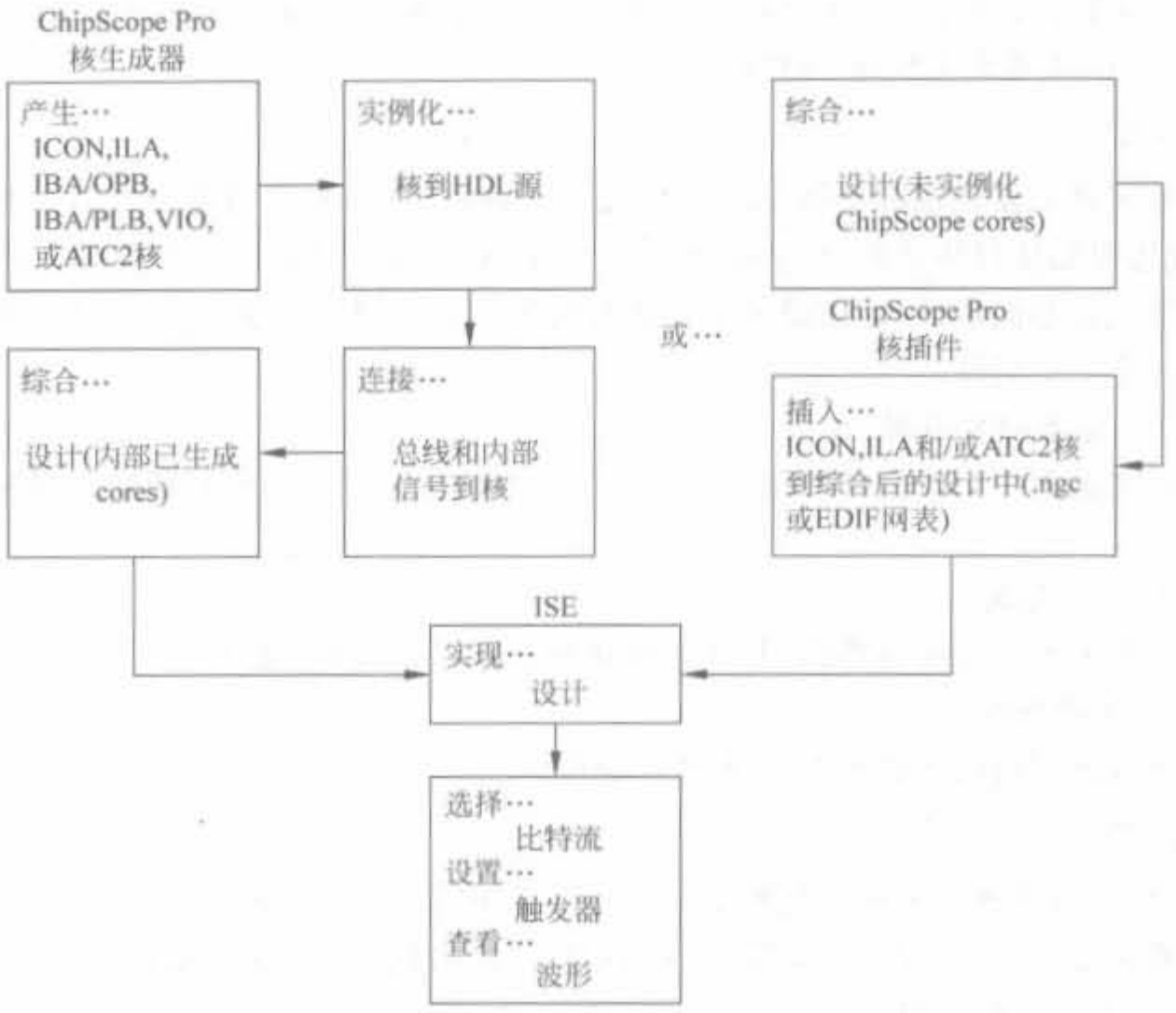


图 6-4 ChipScope 的使用流程示意图

6.2 ChipScope Core Generator 使用说明

ChipScope Pro 核生成器的作用就是根据设定条件生成在线逻辑分析仪的 IP 核，设计人员在原 HDL 代码中实例化生成的核，然后进行布局布线、下载配置文件，就可以利用 ChipScope Pro 分析仪设定触发条件，观察信号波形。本节主要介绍核的基本知识以及核生成器的基本使用方法。

6.2.1 ChipScope Pro 核的基本介绍

ChipScope Pro 提供了 7 类不同的核资源，其中 ICON 核、ILA 核、VIO 核以及 ATC2 核获得了广泛应用。下面对这 4 类核进行简要说明。

1. ICON 核

如前所述，所有的核都需要通过 JTAG 电缆完成计算机和芯片的通信。在 ChipScope Pro 中，只有 ICON 核具备和 JTAG 边界扫描端口通信的能力，因此 ICON 核是 ChipScope Pro 应用必不可缺的关键核。一个 ICON 核可以同时最多连接 15 个 ILA、IBA/OPB、IBA/PLB、VIO 或者 ATC2 核。在 Virtex-4 以及 Virtex-5 芯片中，可以通过 BSCAN_VIRTEX 原语来使用 USER1、USER2、USER3 或者 USER4 扫描链，且由于 BSCAN_VIRTEX 原语

实现了一条单独的扫描链，因此没有必要导出未使用的用户扫描链。在其余类型的芯片中，只能使用 USER1 或者 USER2 扫描链。

2. ILA 核

ILA 核提供触发和跟踪功能，根据用户设置的触发条件捕获数据；然后在 ICON 的控制下，通过边界扫描口将数据上传到 PC；最后在 Analyzer 中显示出信号波形。由于 ILA 核和被监控设计是同步的，因此设计中的所有时钟约束会被添加到相应的 ILA 核中。ILA 包括下面 3 个主要组件：

1）输入、输出触发逻辑

输入触发逻辑用于检测各种细微触发条件；输出触发逻辑用于触发外部测试设备以及其他逻辑。

2）数据捕获逻辑

数据捕获逻辑用于捕获数据，并将所捕获的数据存储到芯片的块 RAM 中。

3）控制和状态逻辑

控制和状态逻辑用于管理 ILA 的各种操作。

3. VIO 核

虚拟输入、输出核用于实时监控和驱动 FPGA 内部的信号，可以观测 FPGA 设计中任意信号的输出结果，以及添加虚拟输入，如 DIP 开关、按钮等，且不占用块 RAM。VIO 核面向模块操作，支持下面 4 类信号：

1）异步输入信号

对于异步输入信号，通过 JTAG 电缆的时钟信号（TCK）采样，周期地读入 PC，再将结果在 ChipScope Pro 分析仪界面上显示。

2）同步输入信号

对于同步输入信号则利用设计时钟采样，然后周期地读入 PC，在分析仪界面上显示。

3）异步输出信号

异步输出信号由用户在 ChipScope Pro 分析仪中定义，再将其送到周围的逻辑中，且其每个输出信号逻辑“1”、“0”的门限可以由用户自己定义。

4）同步输出信号

同步输出信号由用户定义，同步于设计时钟，其“1”、“0”的逻辑门限亦可独立定义。

4. ATC2 核

ATC2 核由 Xilinx 和 Agilent 合作开发，适配于 Agilent 最新一代的逻辑分析器，联合完成调试捕获，允许逻辑分析仪访问 FPGA 内部设计中任何一点的网表，提供更深的捕获深度、更复杂的触发设置，还支持网络远程调试，功能十分强大。与单独使用 ChipScope Pro 相比，它可以更加减少用于调试的 FPGA 内部块 RAM，使其更多地用于设计中。ATC2 可以为每个信号提供 2MB 的捕获深度，是 ILA 深度的 60 倍。此外，它最多允许在 FPGA 内部添加 64 个观测信号组，和逻辑分析仪的链接数最多可达 128 管脚。大的捕获深度以及多的观测信号对于查找设计故障原因是非常有用的。

6.2.2 ChipScope 核的生成流程

在 ChipScope 的各个核中，除了 ICON 核用于控制是必不可缺的之外，其余所有核的功能是类似的，都是用于不同场合的逻辑测试核，属性和配置过程基本一样。因此，本节只介绍 ICON 核和使用频率最高的 ILA 核的生成方法。

1. ICON 核的生成流程

单击“开始”→“所有程序”→“ChipScope Pro 9.1i”→“ChipScope Pro Core Generator”，进入 ChipScope Pro Core Generator 的向导界面，如图 6-5 所示。选中“ICON(Integrated Controller)”选项，单击“Next”进入配置页面。

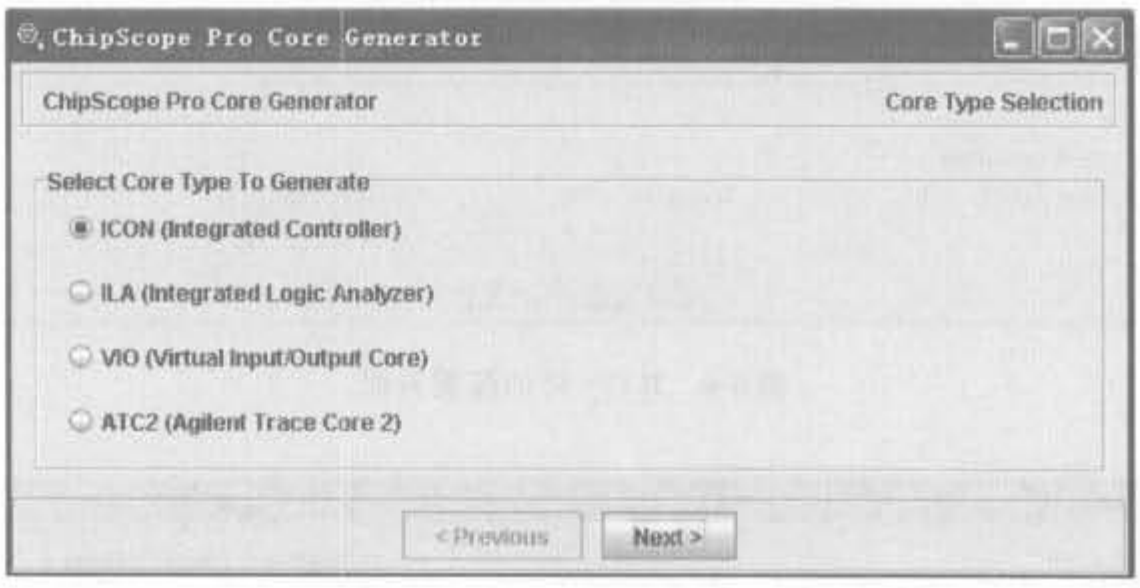


图 6-5 ChipScope Pro Core Generator 的向导界面

ICON 的配置页面如图 6-6 所示。“Design Files”选项用于设定生成目录。单击“Browse”按钮，选择 ICON 核输出网表的存放目录。根据目标芯片型号，在“Device Family”下拉框中选择相应的器件类型。然后，在“Number of Control Ports”下拉框中选择 Port 数。其余一般选用默认配置，则“Core Utilization”栏会给出 ICON 核所占的硬件资源。例如，带 1 个 Port 的 ICON 核要占用 97 个查找表(LUT)单元和 28 个触发器(FF)单元。

需要注意的是，Port 数并不是信号端口数，而是外挂的集成分析仪核(如 ILA、ATC2 以及 VIO 等)的个数。

配置完毕后，单击“Next”按钮，进入 ICON 核生成界面，如图 6-7 所示。其中，可根据源文件的硬件语言形式，选择 VHDL/Verilog HDL 示例代码，并选择合适的综合工具。单击“Generate Core”按钮，进入信息确认页面。单击“Start Over”按钮即可完成 ICON 核的生成，同时出现和图 6-7 一样的核生成向导界面，让用户继续添加所需的分析仪核。

2. ILA 核的生成流程

在图 6-5 所示的向导界面中选择“ILA(Integrated Logic Analyzer)”选项，进入 ILA 核的一般属性配置界面，如图 6-8 所示。“Design Files”选项用于设定生成目录。单击“Browse”按钮，选择 ICON 核输出网表的存放目录。“Device Settings”选项用于设定芯片型号，在“Device Family”下拉框中选择和 ICON 核相同的器件类型。“Clock Settings”用于

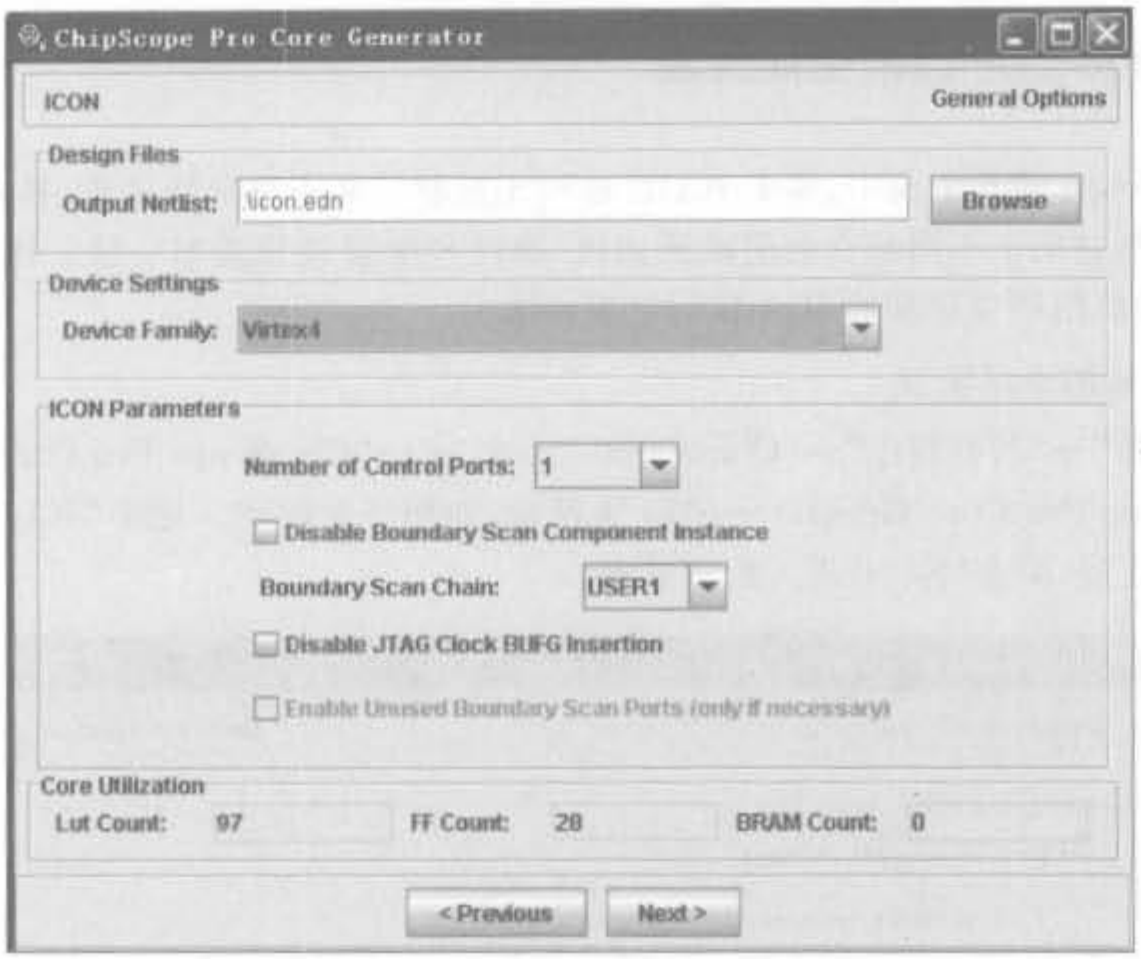


图 6-6　ICON 核的配置界面

图 6-7　ICON 核的生成界面

设定信号采样时刻，有上升沿和下降沿两种选择。配置完成后，单击“Next”按钮，进入 ILA 核触发条件设置界面。

ILA 核触发条件的设置界面如图 6-9 所示。“Number of Input Trigger Ports”用于设定触发端口数，每个端口 ILA 核可支持多路比特数据，最多可有 16 个端口，具体的比特数取决于用于测试的逻辑资源。如果选择默认值 N，则下面只会出现 TRIG0～TRIG(N－1)的配置栏。图 6-9 中选择了默认值 1，则只有 TRIG0 的配置栏，其中的“Trigger Width”用于设定每个端口的触发比特数，这里输入 8；“＃Match Units”为触发单元的级联级数，最多可设置 16 级，这里选择了 1 级；“Counter Width”选项选择 Disabled，以节省资源；“Match

图 6-8 ILA 核的配置界面

Type"为触发条件类型，有 6 类触发条件类型(Basic、Basic w/edges、Extended、Extended w/edges、Range 以及 Range w/edges)，其含义说明如表 6-3 所示。"w/edges"表明可以使用时钟的上升沿和下降沿来采样数据。其余选项选择默认值。配置完毕后，会在"Core Utilization"栏自动给出核所占用的资源。单击"Next"按钮，进入数据端口配置界面。

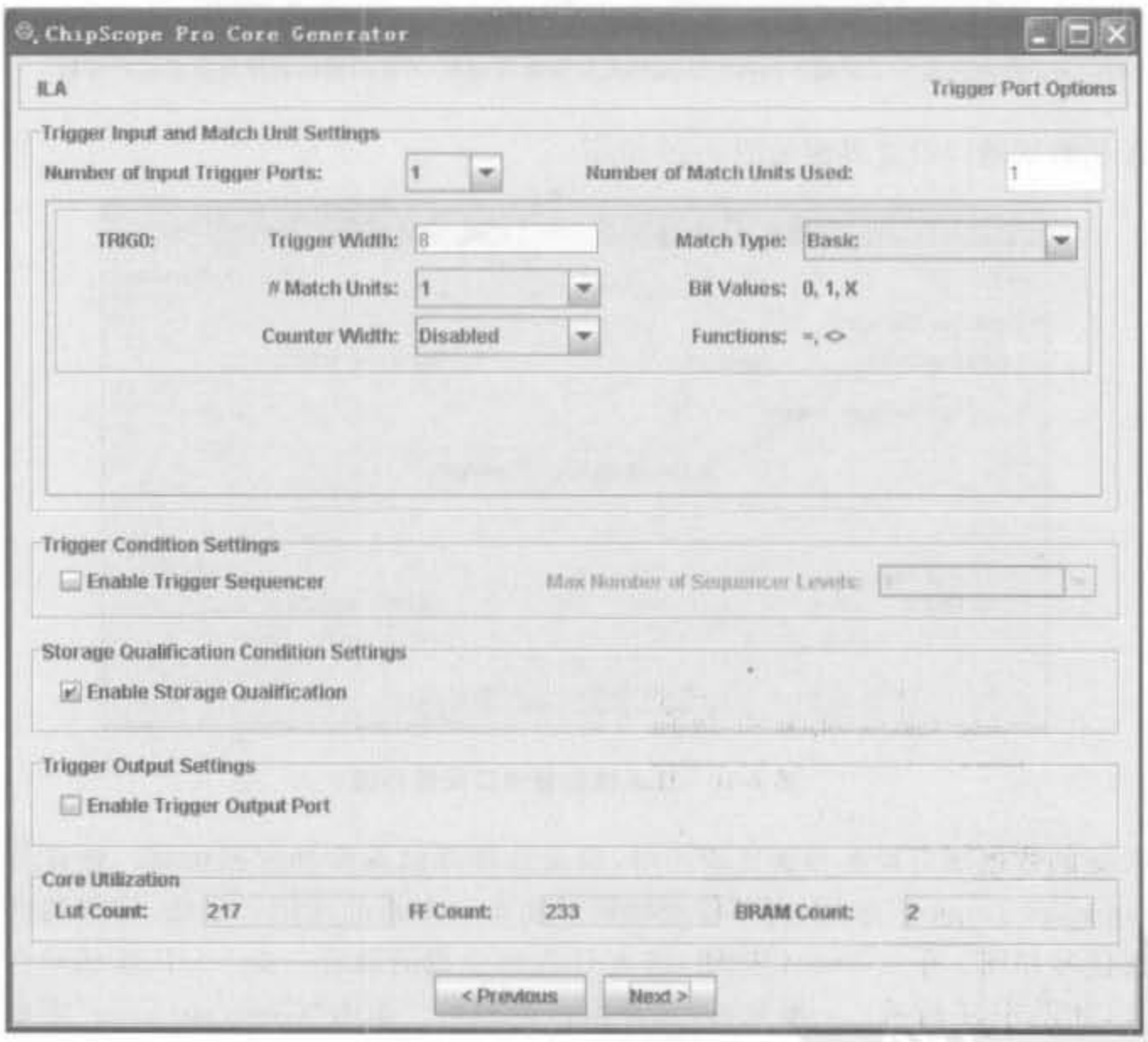


图 6-9 ILA 核触发条件配置界面

表 6-3 触发条件判断单元类型列表

类型	比特值	匹配函数	比特/Slice	说明
Basic	0,1,X	=,<>	Virtex-5：19 其余：8	用于数据信号的比较，在一个匹配单元中可实现最多的比特数
Basic w/edges	0,1,X, R,F,B	=,<>	Virtex-5：8 其余：4	用于控制信号的比较，可以检测信号的边沿跳变(低—高，或高—低)
Extended	0,1,X	=,<>,>,>=,<,<=	Virtex-5：16 其余：2	用于对数值敏感的地址以及数据总线信号的比较
Extended w/edges	0,1,X, R,F,B	=,<>,>,>=,<,<=	Virtex-5：8 其余：2	用于对数值和边沿跳变都敏感的地址以及数据总线信号的比较
Range	0,1,X	=,<>,>,>=,<,<=, 'in range', 'not in range'	Virtex-5：8 其余：1	用于仅对数值敏感的地址和数据总线信号的比较，判断其是否在特定的范围内
Range w/edges	0,1,X, R,F,B	=,<>,>,>=,<,<=, 'in range', 'not in range'	Virtex-5：4 其余：1	用于对数值和边沿跳变都敏感的地址和数据总线信号的比较，判断其是否在特定的范围内

相关说明：1. "0"表示"逻辑 0","1"表示"逻辑 1","X"表示"未知","R"表示"从 0 到 1 的跳转","F"表示"从 1 到 0 的跳转","B"表示"任何电平跳转"。

2. "比特/Slice"数值只是为了说明不同匹配单元的大致资源利用率，不能用精确的硬件资源消耗评估。

ILA 核数据端口设置界面如图 6-10 所示。

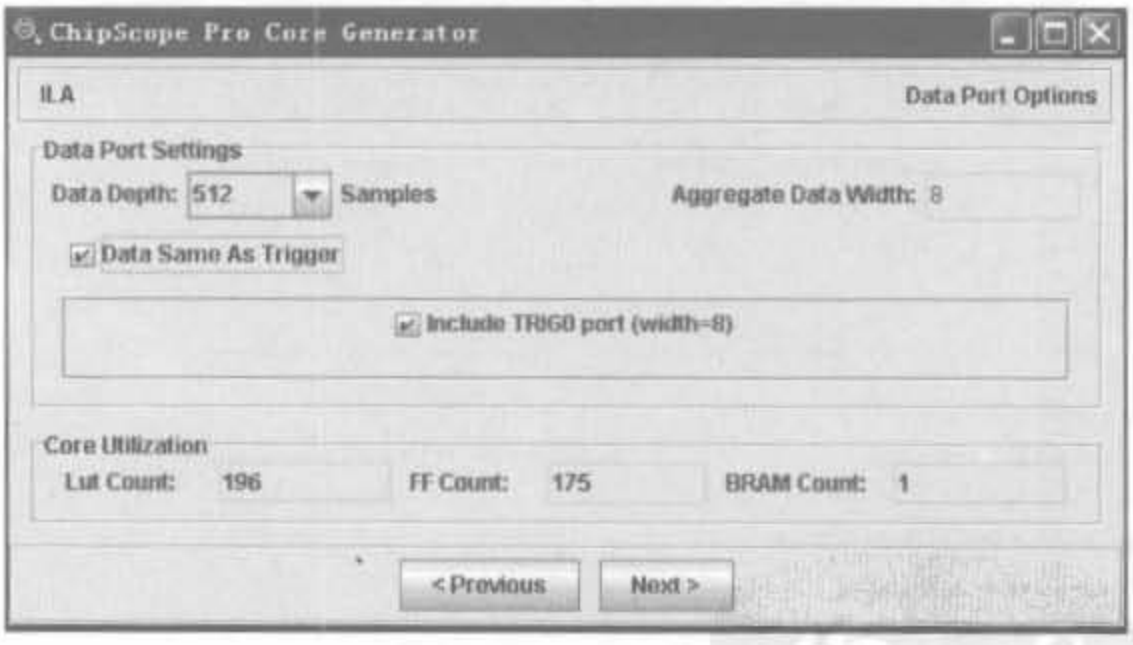


图 6-10 ILA 核数据端口配置界面

ILA 核的数据端口设置界面比较简单，只是选择存储深度和数据位宽。建议读者选择"Data Same As Trigger"选项，可节省逻辑资源和布局布线的使用。当然，建议的前提是数据与触发信号相同。单击"Next"按钮，进入 ILA 核生成的最后一步——ILA 核示例和模板配置界面，如图 6-11 所示。一般来讲，选择默认值即可。单击"Generate Core"完成核生成过程。

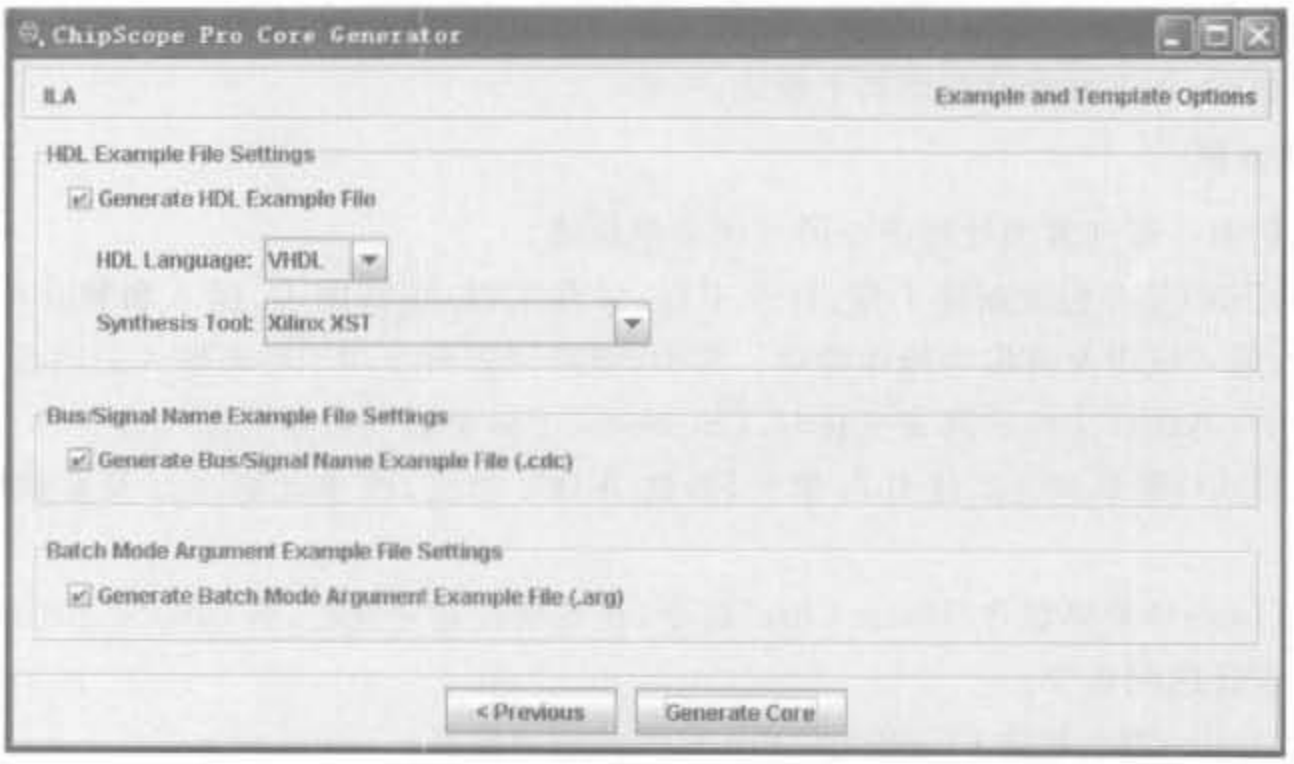


图 6-11 ILA 核示例和模板配置界面

6.3 ChipScope Core Inserter 使用说明

核插入器不仅能产生常用的核(除了 IBA/OPB、IBA/PLB、VIO 以及 IBERT 等核),还能将其自动插入设计网表,不需要手工在 HDL 代码中例化,它在很多场合下替代了核生成器的功能。本节主要介绍核插入器的使用方法。同样,本节只介绍 ICON 核以及 ILA 核在 Core Inserter 中的用法。

6.3.1 Core Inserter 的用户界面

单击 Windows 环境下的“开始”→“所有程序”→“ChipScope Pro 9.1i”→“ChipScope Pro Core Inserter”,即可启动核插入器,其用户界面如图 6-12 所示。

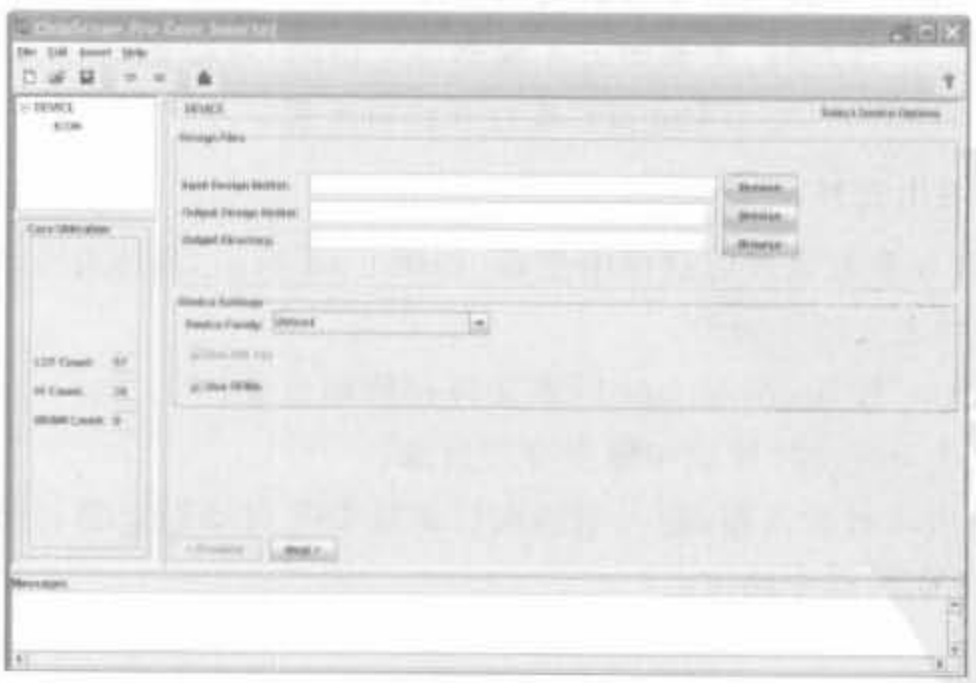


图 6-12 核插入器的用户界面

核生成器的用户界面由标题栏、菜单栏、工具栏、项目浏览器、参数设置对话框以及信息显示窗口组成，各主要部分功能如下所述。

1. 菜单栏

菜单栏由一系列常用处理命令的下拉菜单组成。

文件(File)菜单包含新建工程、打开工程、保存工程、刷新网表、插入和删除单元、参数选择设计、插入核以及推出等操作命令。其中，刷新网表命令用于更新输入的网表文件。在实际应用中，当设计文件的网表变化时，ChipScope Pro 会自动提示更新网表文件。

编辑(Edit)菜单包含新建 ILA 单元、新建 ATC2 单元、核单元删除以及参数配置等操作命令。

插入(Insert)菜单包含“Insert Core”命令，在各项设置完成后，将 ChipScope Pro 的网表插入到原设计的网表中。

帮助(Help)菜单包含 ChipScope Pro 软件的版本信息。

2. 项目浏览器

项目浏览器列出了插入到设计中的所有 ICON 核和 ILA 核，选中一个核后，就可以在参数设置对话框中查看或修改其参数。

3. 参数设置对话框

参数设置对话框用于查看和设置 ICON 核以及 ILA 核的参数。

4. 信息显示窗口

核生成器所有工作状态信息都在该窗口显示。

6.3.2 Core Inserter 的基本操作

核插入器中的工程文件(后缀为.cdc)保存了与源文件、目标文件、核参数以及配置等相关的信息，允许设计人员保存和找回不同阶段的配置信息。此外，.cdc 文件可以作为 ChipScope Pro 分析仪用来提取信号名的输入文件。

当核插入器第一次打开时，所有相关区域都是空白的，可以通过“File”→“New”来新建工程，也可以通过“File”→“Open Project”来打开已有工程。

1. 指定输入、输出文件

输入、输出文件在参数设置对话框中完成，如图 6-13 所示，具体有下面 3 个步骤：

(1) 指定输入网表文件(.ngc)。

(2) 单击“Browse”按钮，指定.ngc 网表文件存放的目录。

(3) 指定输出网表的存放目录和输出存放目录。

如果在 ISE 中启动核插入器，输入和输出目录是 ISE 自动设置的，且其参数只能在 ISE 中改变，而不能在核插入器中修改。

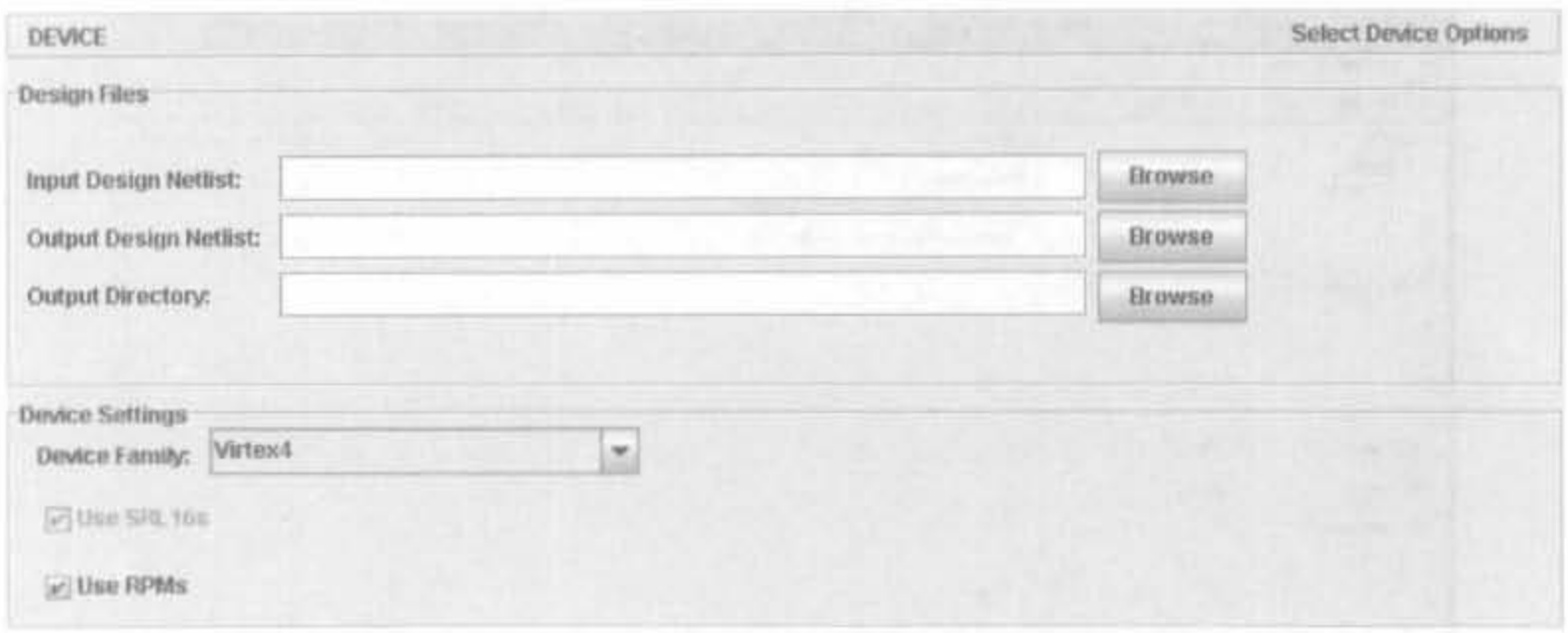


图 6-13 核插入器的配置界面

2. 工程级参数

对于芯片型号、SRL16 结构单元的使用以及 RPM 的使用这 3 项工程级参数，在每个工程中都必须设置。目标 FPGA 芯片的信号在“Device Settings”区域中的“Device Family”下拉框中选择。由于 ICON 核和捕获核的结构都是根据不同的芯片类型进行优化的，因此需要根据电路板选择合适的芯片型号。默认选项为 Virtex-4 系列。

“Use SRL 16s”检验栏用于配置生成核是否采用 SRL16 以及 SRL16E 组件。该选项只在 Virtex-2 以上或 Spartan-3 以上系列芯片可选。如果不选择该检验栏，将使用触发器和复用器来实现，从而提高时序性能，但增加了核所占的资源。在默认情况下，为了节省资源，该选项是选中的。

RPM 用于配置是否将单个的核资源进行相对布局。该选项将为布局布线工具添加约束，使所有 ChipScope Pro 逻辑达到最优布局。如果设计使用了大部分的资源，那么满足布线约束的难度非常大，通过“Use RPMs”选项可以辅助布局布线器满足相应的时序。默认情况下，该选项是选中的。

核插入器左边就是核的资源使用率面板，显示了插入到网表中的核的 LUT、FF 以及块 RAM 等资源的占用情况。资源占用数值取决于核的数量、种类及其参数配置情况。

3. 核配置选项

由于各个核的功能和配置参数的意义是不同的，因此下面对常用的 ICON 核、ILA 核以及 ATC2 核配置进行简单说明。

1) ICON 核配置说明

由于在所有核中，ICON 核是所有核和 JTAG 边缘扫描电路的通信控制器，因此首先对其进行说明。其参数配置界面如图 6-14 所示。

如果选中“Disable JTAG Clock BUFG Insertion”选项，则实现工具在对 JTAG 时钟布线时选用普通布线资源来代替全局时钟资源。默认情况下，ChipScope Pro 是选用全局时钟资源的。注意，在全局时钟资源不紧张时，尽量使用全局时钟，以保证时钟偏移最小化，从而

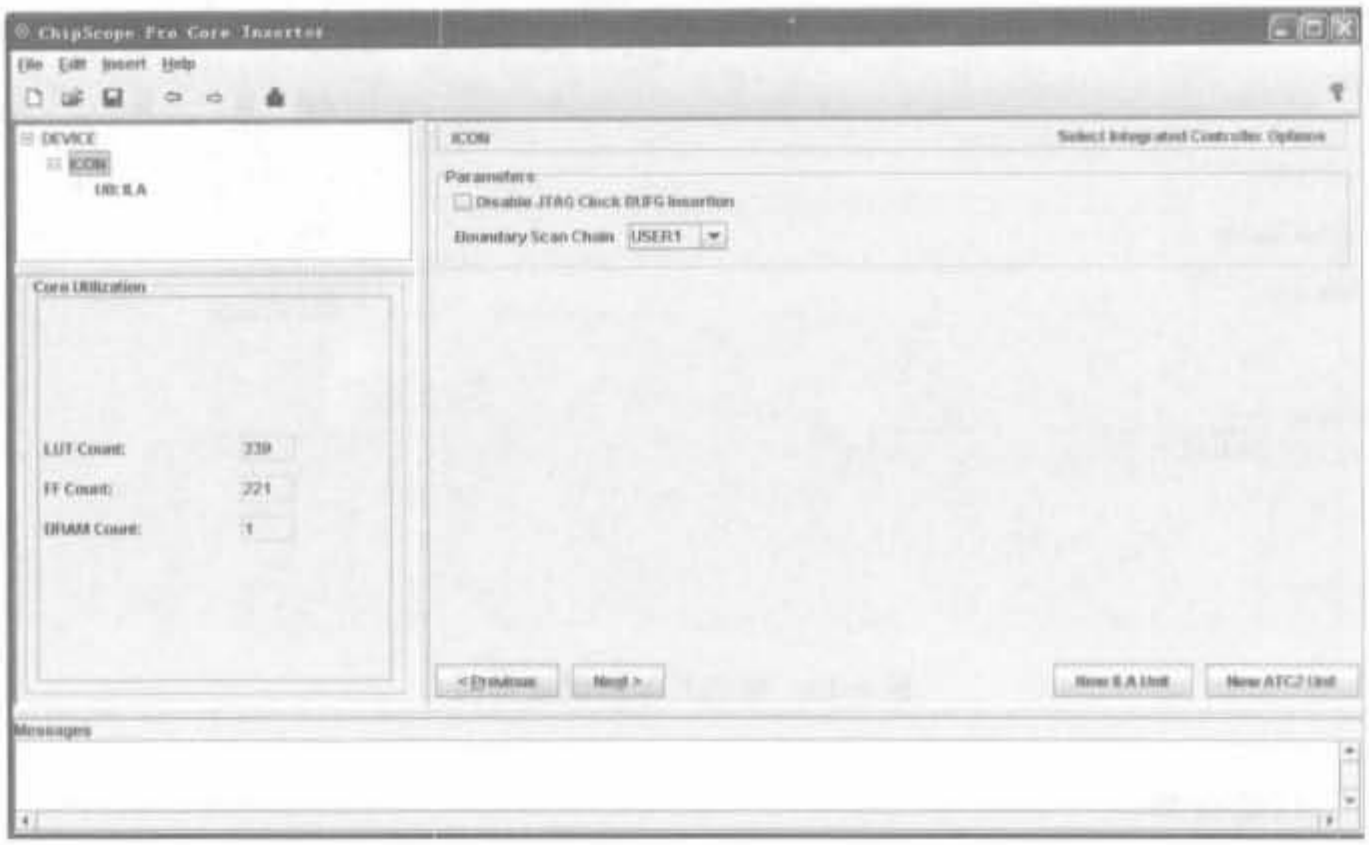


图 6-14　ICON 核的属性配置界面

保证时序稳定。即使全局时钟资源不够而不得不禁用 BUFG 时，也必须添加相应的约束，使 JTAG 时钟线上的抖动尽可能小。当 ICON 参数配置完后，单击“Next”或者新建 ILA 单元来自动生成 ILA 单元；当需要添加 ATC2 核时，新建 ATC2 单元即可。

2）ILA 核配置说明

ILA 核的配置“Selcet Integrated Logic Analyzer Options”分为 3 个部分：核触发条件的配置、核捕获条件的参数配置以及网线连接配置，如图 6-15 所示（设置完 ICON 核后，单击“Next”即可出现该界面）。

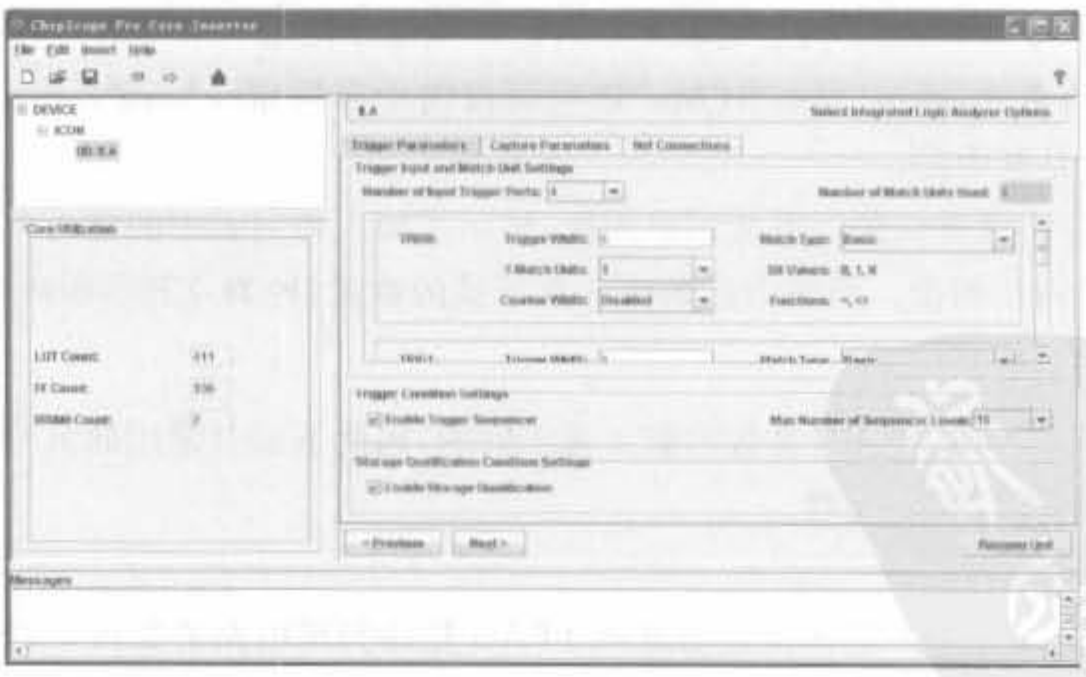


图 6-15　ILA 核触发条件的配置界面

“Trigger Parameters”选项用于设定触发输入信号以及触发条件判断单元。输入触发端口数在下拉栏“Number of Input Trigger Ports”中选择，最多有 16 个端口，并以 TRIG *n*

命名各个触发端口，n 的范围为 0～15。每个触发端口都有位宽(触发端口的信号线总数)、触发条件判断单元个数以及类型等参数，其中各个端口的位宽可独立配置，且其范围为 1～256；一个端口可以有多个触发条件匹配单元，在“# Match Units”下拉栏中选择，但一个 ILA 核总的触发条件匹配单元的个数超过 16。触发条件匹配单元越多，意味着采集信号的灵活性越大，占用的物理资源也就越多，因此在满足触发条件的情况下，应尽量减少触发条件判断单元的个数。

触发参数设置完后，单击“Next”按钮进入捕获参数选项卡“Capture Parameters”，完成对存储深度、数据位宽、采样时刻等参数的配置，如图 6-16 所示。

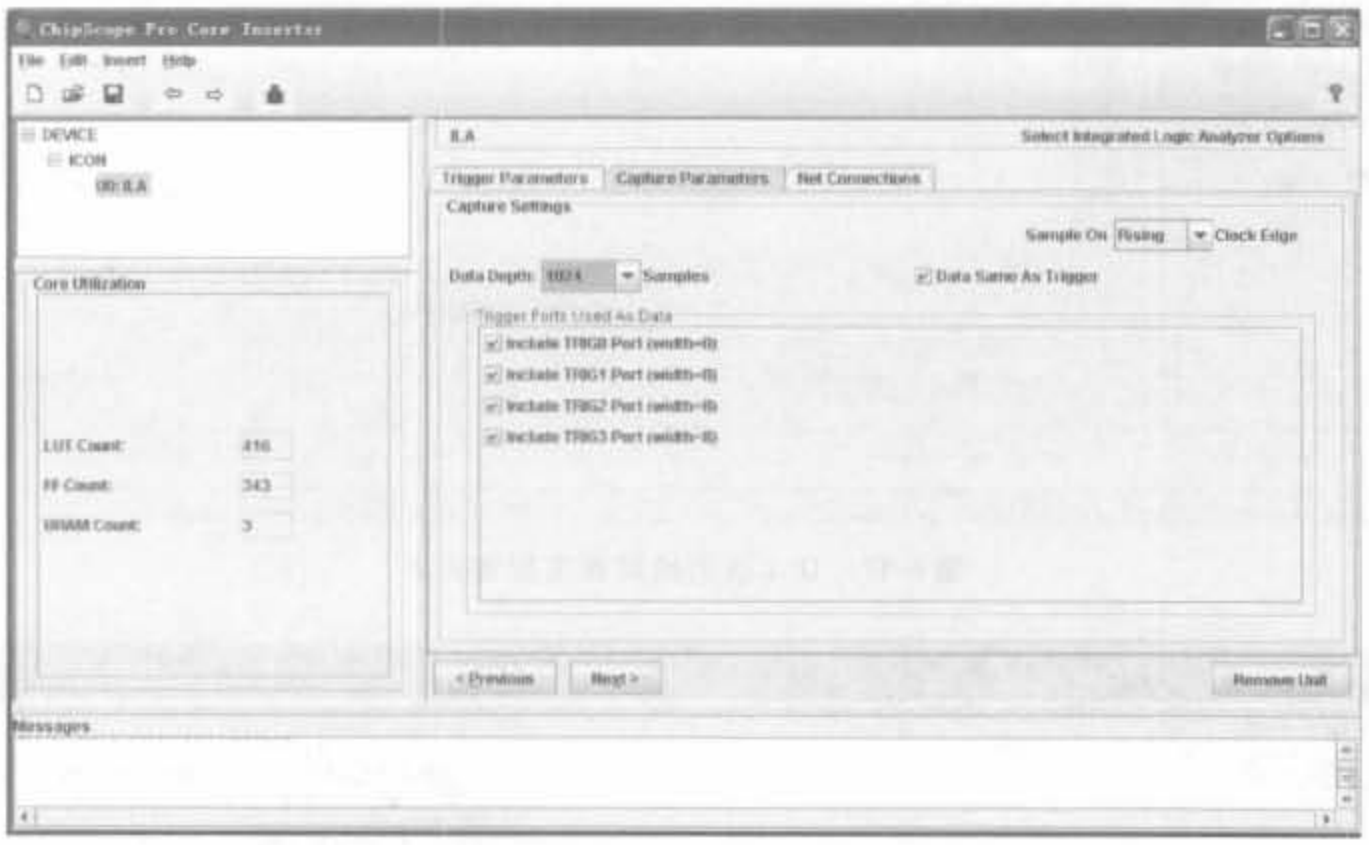


图 6-16 ILA 核捕获条件的配置界面

ILA 核缓存器所能够存储的最大采样值个数称为数据深度，与数据宽度共同决定了块 RAM 的占用数。核生成器和插入器能根据相应配置准确给出块 RAM 的个数。其单项数据位宽指标最大为 256bit，单项最大存储深度为 16384。例如在 Virtex-4 芯片中，其块 RAM 为 18Kbit，因此采集的数据深度和宽度满足：深度×宽度≤18×1024×块 RAM 的个数。如果选中“Data Same As Trigger”选项，则数据与触发信号相同，这是一种很常用的模式，可以捕获和采集触发逻辑分析仪的任何数据。然后在“Trigger Ports As Data”中选择作为数据的触发端口。在该模式下，ILA 核将忽略数据端口，以减少逻辑资源和布线资源，但总的数据宽度仍要小于等于 256。如果不选中“Data Same As Trigger”选项，数据和触发信号独立，在采样数据远小于触发位宽时，会增大采集数据量，浪费块 RAM 资源，因此建议读者选中该选项。完成上述设计后，可以在主窗口的左边看到资源占用情况。然后单击“Next”按钮进入线网连接界面，如图 6-17 所示。

如图 6-17 所示，由于目前为建立核端口和设计线网信号的连接，所以“Net Connections”中的所有信号都为红色显示。单击“Modify Connections”，会弹出“Select Net”配置界面，如图 6-18 所示。

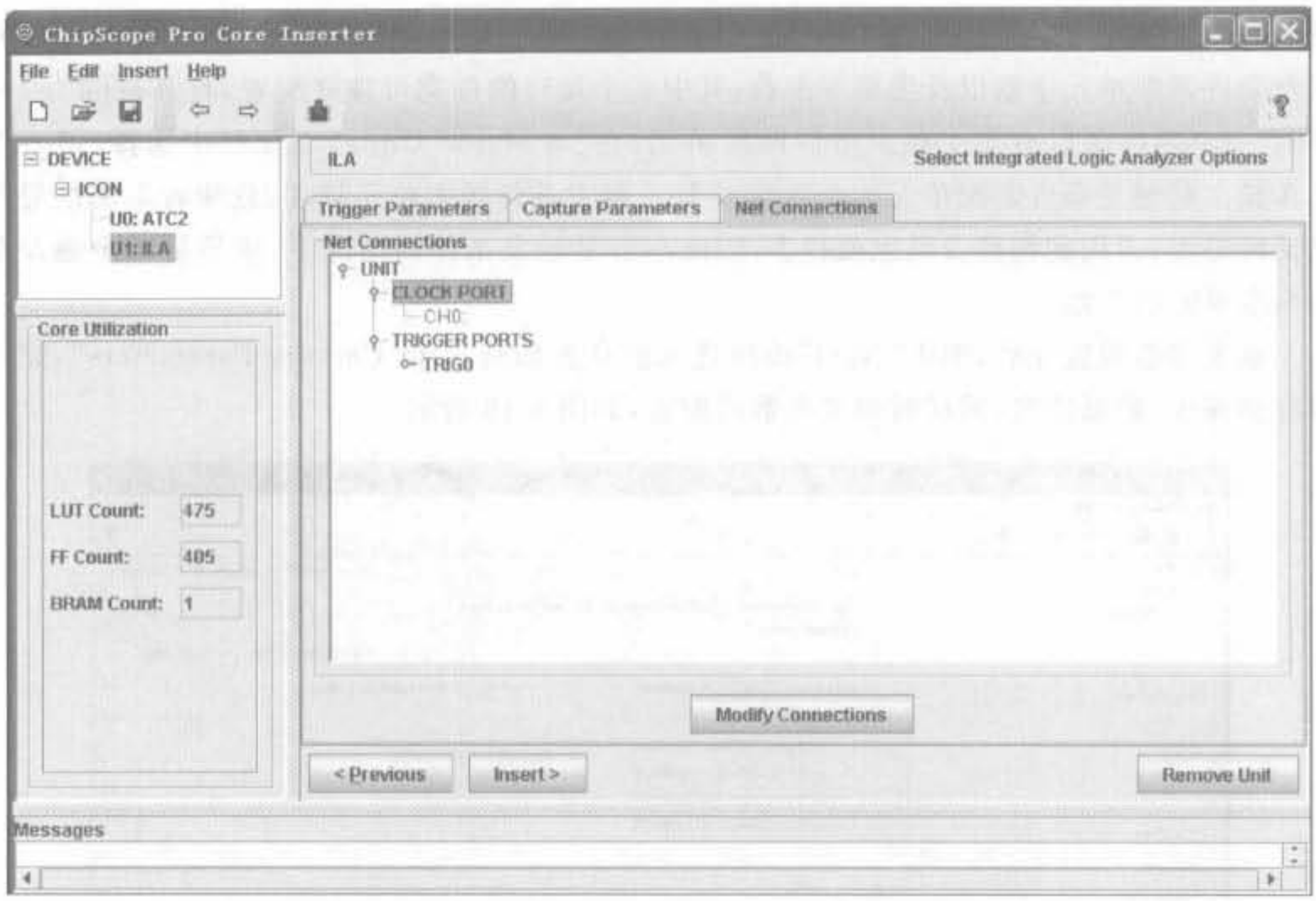


图 6-17　ILA 核的线网连接配置界面

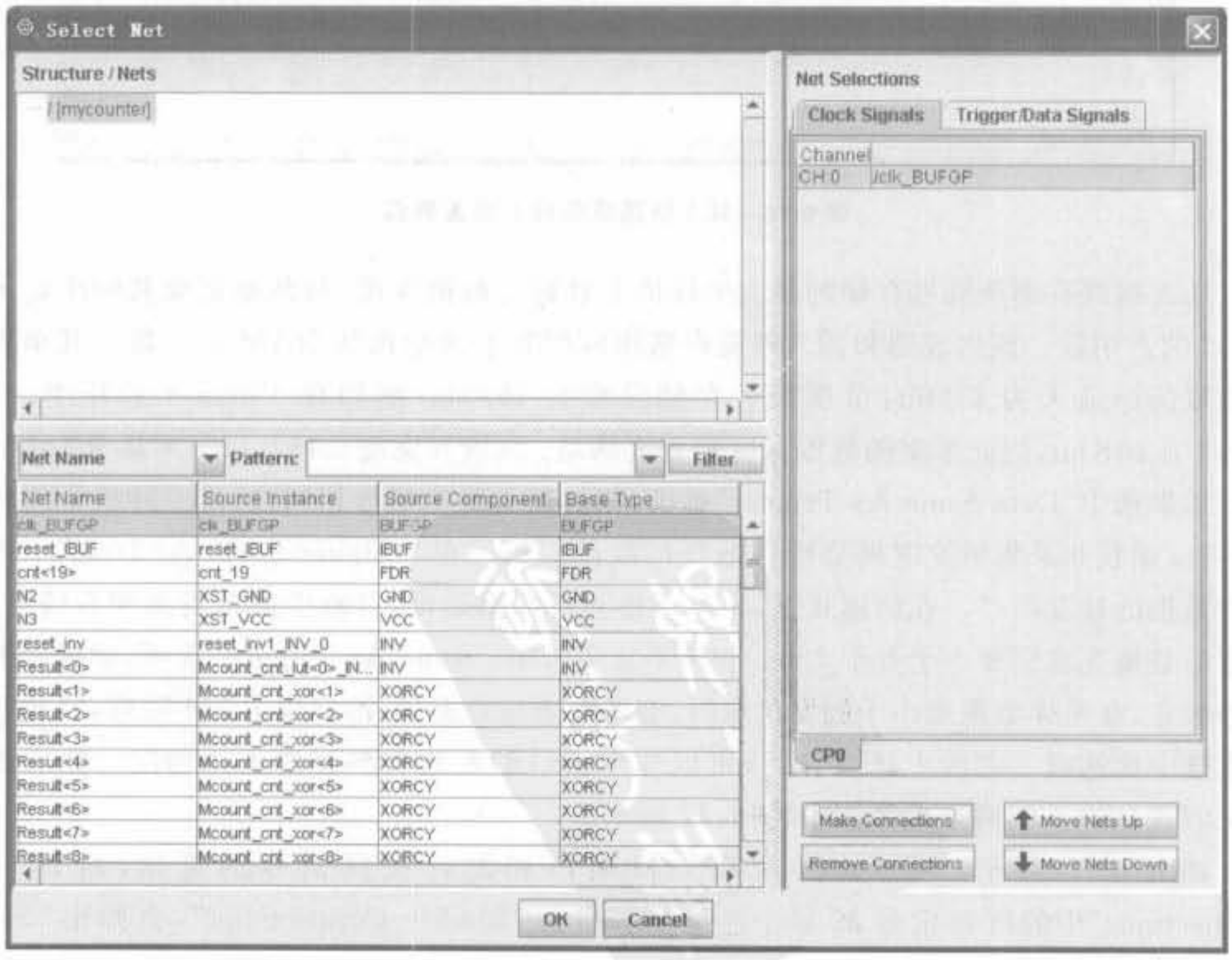


图 6-18　ILA 核与设计线网信号连接界面

首先在“Net Selections”栏选择“Clock Signals”或者“Trigger/Data Signals”子目录；然后在相应页面选中 ILA 核中的信号列表；再在左侧信号线网列表中找出期望观测的信号；最后单击右下角的“Make Connections”按钮，完成一次连接。只有将 Chip Scope 核中的所有信号都连接了相应的线网，设计才能被正确实现。

6.4 ChipScope Pro Analyzer 使用说明

ChipScope Pro 分析仪直接和 ICON、ILA、ATC 以及 VIO 等核交互，并允许用户配置器件、选择触发条件、建立控制台并通过 PC 显示分析结果。其数据观察方式和触发模式可根据设计进行灵活选择。

6.4.1 ChipScope 分析仪的用户界面

在 Windows 操作系统中，有两种方法可以启动 ChipScope Analyzer，一种方法就是单击“开始”→“所有程序”→“ChipScope Pro 9.1i”→“ChipScope Pro Analyzer”；另一种方法是设计综合实现后，在 ISE 的“Processes for Source”中双击“Analyzer Design Using ChipScope”。分析仪的用户界面如图 6-19 所示。

图 6-19 ChipScope 分析仪用户界面示意图

分析仪的用户界面主要由菜单栏、常用工具栏、项目浏览器、信号浏览器、主窗口以及信息显示窗口组成。各主要部分功能如下所述。

1. 菜单栏

“文件(File)”菜单包含“新建项目(New Project)”、“打开项目(Open Project)”、“保存项

目(Save Project)"、"项目另存为(Save Project As)"、"页面建立(Page Setup)"、"打印(Print)"、"导入(Import)"、"导出(Export)"以及"退出(Exit)"等命令。"导入"用于从设计文件中获取信号列表,"导出"用于提取捕获数据,以便后续观察和处理。

"视图(View)"菜单包含"显示项目浏览器(Project Tree)"、"显示信息显示窗口(Messages)"两个命令。

"边界扫描链路(JTAG Chain)"菜单包含下载电缆相关的命令,有"主机服务设置(Serve Host Setting)"、"JTAG 链建立(JTAG Chain Setup)"、"连接 Xilinx 并行下载线(Xilinx Parallel Cable)"、"连接 Xilinx 并行 USB 下载线(Xilinx Parallel USB Cable)"、"关闭电缆(Close Cable)"、"获取电缆信息(Get Cable Information)"以及"打开自动核状态查询(Auto Core Status Poll)"等命令。

"器件(Device)"菜单包含"边界扫描链设置(JTAG Device Chain Setup)"、"配置器件(Configure)"、"显示器件识别码(Show IDCODE)"和"显示用户码(Show USERCODE)"等命令。

"窗口(Windows)"菜单包含"新建窗口单元(New Unit Window)"、"关闭(Close)"、"自动重排(Auto Layout)"等相关命令。

"帮助(Help)"菜单包含版本信息以及所有核信息。

2. 项目浏览器

项目浏览器在 JTAG 边界扫描链正确初始化后会列出扫描链上所有能识别的器件,显示核的数目,并为每个核创建一个文件夹,其中包含触发条件设置和要观察信号的波形文件。在配置下载完成后,项目浏览器也会同时更新。

3. 信号浏览器

信号浏览器用于添加和删除视图中的信号,当在项目浏览器中选中一个核后,会列出其所有信号以及完成重命名和信号总线组合等操作。

4. 主窗口

主窗口主要用于显示"Trigger Setup"、"Waveform"、"Listing"以及"Bus Plot"等窗口。

5. 信息显示窗口

信息显示窗口会列出分析仪所有的状态信息,便于用户查看。

6.4.2 ChipScope Analyzer 的基本操作

使用 Analyzer 观察信号波形时,首先需要将设计和 ChipScope Pro 核共同生成的配置文件下载到 FPGA 芯片中,然后通过设定不同的触发条件捕获波形,将其存储在芯片的块 RAM 上,通过 JTAG 链回读到 PC 上观察波形。

1. 配置目标芯片

打开 Analyzer,在常用工具栏上单击图标"[illegible]",初始化边界扫描链。成功完成扫描后,项目浏览器会列出 JTAG 链上的器件,如图 6-20 所示。Analyzer 能自动识别出链上的所有 Xilinx 主流 CPLD、FPGA、PROM 以及 System ACE 芯片。

图 6-20 Analyzer 边界扫描结果示意图

当JTAG链扫描正确后，菜单项“Device”才能由灰色变为正常，用户可单击“Device”→“DEV：0 My Device0(XC3S500E)”→“Configure”进行配置，此时弹出如图 6-21 所示的配置对话框，提示用户选择需要下载的.bit 文件。需要注意的是，ChipScope 利用JTAG 链来观察芯片内部逻辑，因此在生成配置文件时只能利用.bit 格式的配置文件，且建立时钟只能选择“JTAG CLK”，选择“CCLK”可能会导致配置失败。

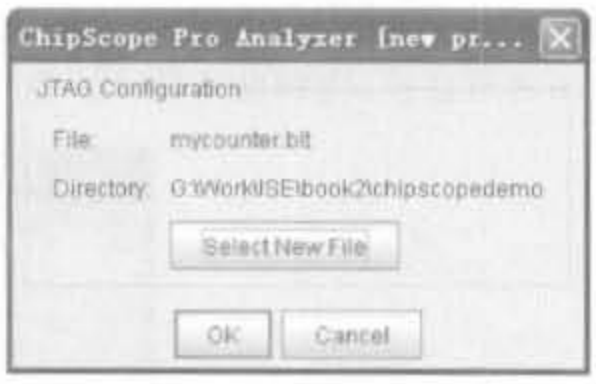


图 6-21 Analyzer 配置芯片示意图

在配置的过程中，Analyzer 的右下角会给出配置状态。配置成功后显示“Done”标志，提醒用户芯片配置已完成。

2. 设置触发条件

把 ChipScope 设计和工程下载到 FPGA 中以后，还需要设定触发条件，才能在 Analyzer 中捕获到有效波形。Analyzer 的触发设置由 Match(匹配)、Trig(触发)以及 Capture(捕获)3 个部分组成。其中，Match 用于设置匹配函数，Trig 用于把一个或多个触发条件组合起来构成最终的触发条件，Capture 用于设定窗口的数目和触发位置。典型的配置界面如图 6-22 所示。

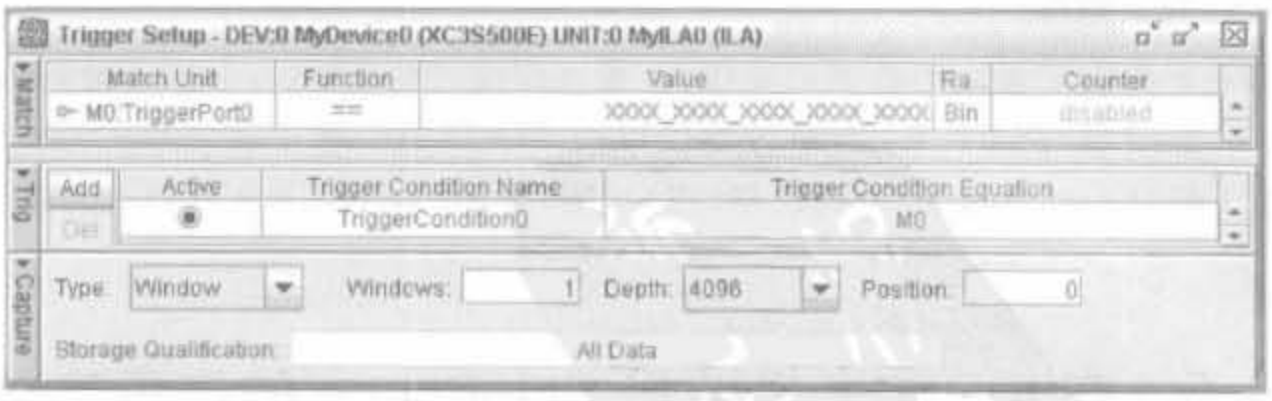


图 6-22 Analyzer 触发条件配置示意图

3. 观察信号波形

观察信号波形需要打开“Waveform”窗口，可在 JTAG 链目录下相应芯片的 ILA 核下单击“Waveform”命令完成，显示界面类似于逻辑分析仪，如图 6-23 所示。

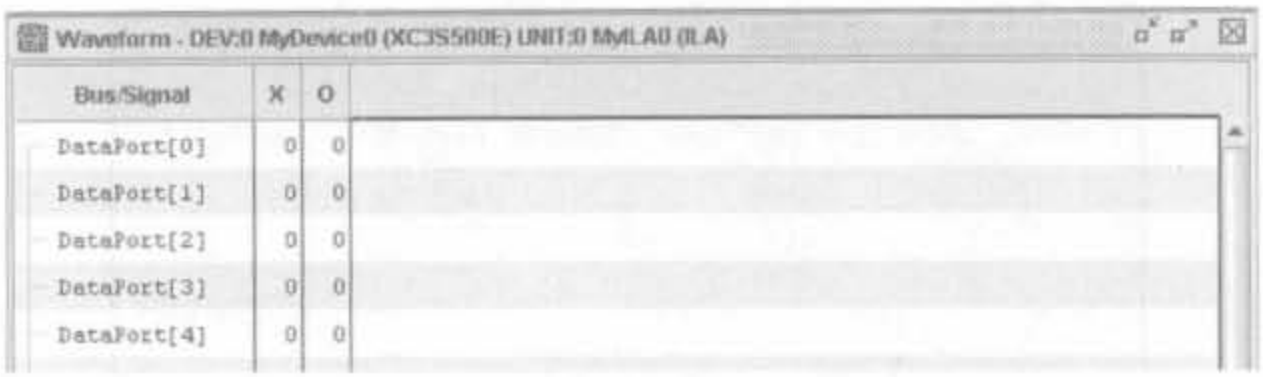


图 6-23　Analyzer 波形显示界面示意图

由于波形显示窗口列出的是 ChipScope 核中的信号，而不是设计中的线网信号，需要通过导入.cdc 文件来添加信号网表名。在“File”菜单下，单击“Import..”命令，会弹出如图 6-24 所示的对话框，选择设计目录下的.cdc 文件即可。

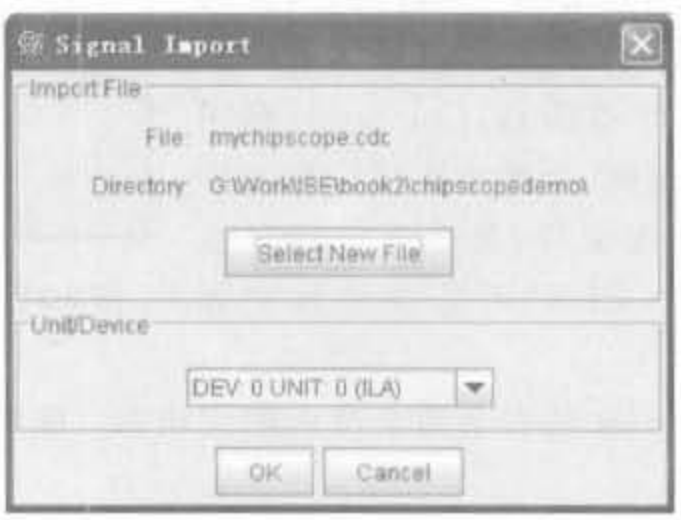


图 6-24　加载.cdc 文件操作示意图

添加了信号名后，可按住 Ctrl 键，选择多个总线信号。单击鼠标右键，选择“Add to Bus”命令，将其组合成相应的总线信号。

完成上述操作后，最后在“Trigger Setup”菜单下选择“Run”命令，或单击工具栏的“▶”图标，开始采集数据。

4. 导入、导出数据

ChipScope 提供了强大的数据采集能力，最大深度可达 16384，单靠肉眼观测是不可行的，需要将采集波形存储下来，再通过 VC 以及 MATLAB 等工具完成后续分析。选择“File”菜单下的“Export”命令，可完成相应的功能，导出.VCD、.ASCII 以及.FBDT 这 3 种类型的文件。

6.5　在 ISE 中直接调用 ChipScope 的应用实例

在 Xilinx 软件设计工具中，ISE 可集成 Xilinx 公司的所有工具和程序。ChipScope 也不例外，在 ISE 中将其作为一类源文件，和 HDL 源文件、IP Core 以及嵌入式系统的地位是等同的。本节通过在 XC3S500E 芯片上实现一个计数器实例，详细介绍如何在 ISE 中新建 ChipScope 应用以及观察、分析数据的操作。

6.5.1 在工程中添加 ChipScope Pro 文件

(1) 新建用户工程,相应的计数器代码如下所列:

```
module mycounter(clk,reset,sout);
  input clk;
   input reset;
   output sout;

   reg [19:0] cnt = 0;
   always @(posedge clk) begin
      if(!reset)
            cnt<=0;
      else
            cnt<=cnt+1;
   end

   assign sout = cnt[19];

endmodule
```

然后根据电路连接,添加相应的 UCF 文件。

(2) 综合工程,然后在 ISE 工程管理区单击鼠标右键,添加"ChipScope Definition and Connection File",并命名为 mychipscope.cdc。

(3) 在工程区双击 mychipscope.cdc,即可自动打开 ChipScope Pro Core Inserter 软件,按照 6.3 节介绍的步骤添加触发单元和触发位宽。其中,触发类型选为 Basic,位宽为 20bit;设置采样深度为 4096。

(4) 将 ChipScope 核信号和设计中的网表信号连接起来,单击"Return Project Navigator",退出 ChipScope,返回到 ISE 中。

(5) 在 ISE 中完成实现,并生成可编程文件。

6.5.2 在 ChipScope Pro 中完成下载和观察

(1) 在 ISE 中双击"Analyzer Design Using ChipScope",可自动打开 ChipScope Pro Analyzer 软件,触发条件采用默认值。

(2) 单击工具栏上的图标"",初始化边界扫描链。等扫描完成后,单击"Device"菜单下"DEV:0 My Device0(XC3S500E)"→"Configure"命令配置芯片。

(3) 按照 6.5 节的方法加载.cdc 文件,修改信号名,并组合出 cnt 总线。

(4) 单击工具栏的"▶"图标,开始采集数据。整体结果如图 6-25 所示,单击工具栏的"🔍"按钮,可放大信号,局部结果如图 6-26 所示。从分析结果可以看出,本设计在 FPGA 中成功地完成了 20bit 计数器的功能。

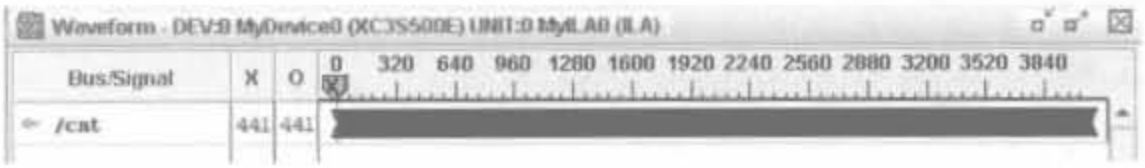


图 6-25　Analyzer 分析结果整体示意图

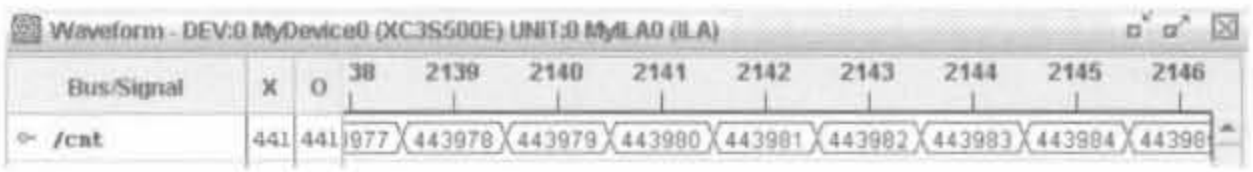


图 6-26　Analyzer 分析结果局部示意图

6.6　本章小结

ChipScope 软件具备传统逻辑分析仪的基本功能，在调试阶段可方便地观察 FPGA 内部信号，且价格便宜，在实际开发中应用广泛。本章首先介绍了 ChipScope 的特点以及使用流程。其次，较为详细地给出了 Core Generator/Inserter 以及 Analyzer 组件的用法，重点分析了 ICON 核和 ILA 核的特点和使用方法。最后，给出了如何在设计中将 ChipScope 作为用户子模块，并给出了 20bit 计数器的操作实例。通过本章的学习，读者应熟练掌握 ChipScope 的使用方法。

第7章

基于FPGA的数字信号处理技术

CHAPTER 7

数字信号处理(Digital Signal Processing)又称 DSP,就是信号的数字化以及数字处理。数字信号处理的研究开始于 20 世纪 60 年代,由于过去很长时间里受计算机集成电路技术和数字化器件发展水平的限制,数字信号处理理论的实时应用很难实现,数字信号处理的应用只限于理论概念的讲授和仿真。到了 20 世纪 90 年代,随着数字化硬件技术水平的飞速发展,数字信号处理的理论才在实际应用中大量实现。在 FPGA 领域,自从 Xilinx 公司推出了高性能、低功耗 XtremeDSP 模块后,FPGA 已经成为高性能的数字信号处理的核心解决方案之一。本章主要对数字信号处理的基本原理、Xilinx 公司的解决方案进行介绍,并给出相应的 FPGA 实现。

7.1 数字信号概述

信号可以从不同的角度来区分,如果从时间上是否连续来考虑,信号可以分为连续时间信号(模拟信号)、离散时间信号和数字信号。自然界的信号大多数都是模拟信号。数字信号指的是幅度取值是离散的信号,幅值表示被限制在有限个数值之内。数字信号的优势在于受噪声的影响小,易于被处理器和 PLD 等芯片处理。

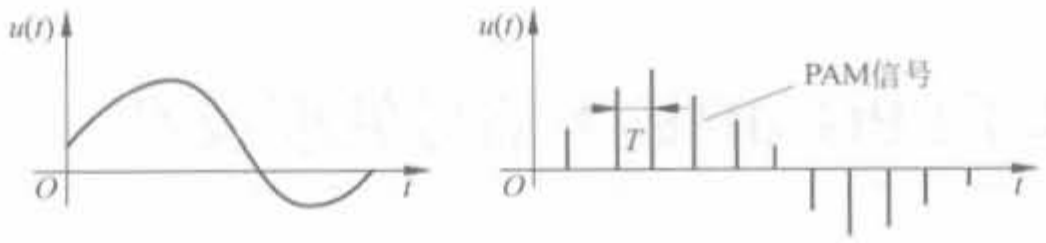
7.1.1 数字信号的产生

虽然数字信号具有处理上的优势,但在现实生活中它并不存在,人的感觉器官只能感受到模拟信号。数字信号都是由模拟信号转化而来的,经过数字处理后,还需要转化成模拟信号才能被人所感受到。因此,二者之间的相互转化是数字信号处理领域永远存在的课题。

1. 模拟信号和数字信号

模拟信号的信号波形随着信息的变化而变化,如图 7-1 所示的信号称为模拟信号,其特点是幅度连续(连续的含义是指在某一取值范围内可以取无限多个数值)。图 7-1(a)所示的信号是模拟信号,其信号波形在时间上也是连续的,因此它又是连续信号。图 7-1(b)所示的信号是对图 7-1(a)所示的模拟信号按一定的时间间隔 T 抽样后的抽样信号,由于其波形在时间上是离散的,它又叫离散信号,但此信号的幅度仍然是连续的,所以仍然是模拟信号。电话、传真、电视信号都是模拟信号。

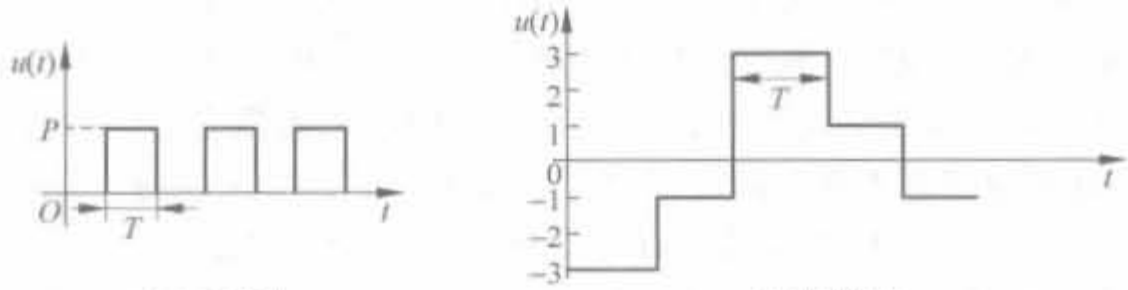
数字信号的波形如图 7-2 所示,其特点是幅值被限制在有限个数值之内,它不是连续的



(a) 连续信号　　(b) 离散信号

图 7-1　不同类型信号的波形示意图

而是离散的。图 7-2(a)所示是二进码，每一个码元只取两个幅值(0，P)。图 7-2(b)所示是四进码，每个码元取 4 个幅值(3，1，−1，−3)中的一个。从中可以看出，幅度也离散的离散信号就是数字信号。



(a) 二进码　　(b) 四进码

图 7-2　数字信号

2. 信号的数字化过程

信号的数字化需要 3 个步骤：抽样、量化和编码。抽样是指用每隔一定时间的信号样值序列来代替原来在时间上连续的信号，也就是在时间上将模拟信号离散化。量化是用有限个幅度值来近似原来连续变化的幅度值，把模拟信号的连续幅度变为有限数量的且有一定间隔的离散值。编码则是按照一定的规律，把量化后的值用二进制数字表示，然后转换成二值或多值的数字信号流。这样得到的数字信号可以通过电缆、微波干线、卫星通道等数字线路传输。在接收端，与上述模拟信号数字化过程相反，数字信号经过后置滤波恢复成原来的模拟信号。上述数字化过程又称为脉冲编码调制。

7.1.2　采样定理

当把模拟信号转化成离散信号且无失真时，采样必须满足著名的奈奎斯特采样定理。采样定理说明了一个问题，即当对时域模拟信号采样时，应以多大的采样周期(或称采样时间间隔)采样，才不至于丢失原始信号的信息，或者说，可由采样信号无失真地恢复出原始信号。

奈奎斯特采样定理　在进行模拟/数字信号的转换过程中，当采样频率 f_s 大于信号最高频率 f_{max} 的 2 倍，即 $f_s \geqslant 2f_{max}$ 时，采样之后的数字信号完整地保留了原始模拟信号中的有效信息；如果 $f_s < 2f_{max}$，则采样后会发生频谱混叠现象。

采样定理说明：如果 $\omega_s > 2\omega_{max}$，就不发生频混现象，因此对采样脉冲序列的间隔 T_s 必须加以限制，即采样频率 $\omega_s(=2\pi/T_s)$ 或 $f_s(=1/T_s)$ 必须大于或等于信号 $x(t)$ 中的最高频率 ω_{max} 的 2 倍，即 $\omega_s > 2\omega_{max}$，或 $f_s > 2f_{max}$。因此得到结论：为了保证采样后的信号能真实

地保留原始模拟信号的信息，采样信号的频率必须至少为原信号中最高频率成分的 2 倍。这是采样的基本法则，称为采样定理。

需要注意的是，在对信号进行采样时，满足了采样定理，只能保证不发生频率混叠，或者说只能保证对信号的频谱作逆傅里叶变换时，可以完全变换为原时域采样信号 $x_s(t)$，而不能保证此时的采样信号能真实地反映原模拟信号 $x(t)$。在工程实际中，采样频率通常大于信号中最高频率成分的 3～5 倍。

7.1.3　数字系统的主要性能指标

1. 信道传输速率

信道的传输速率通常是以每秒所传输的信息量多少来衡量的。信息论中定义信源发生信息量的度量单位是“比特”(bit)。一个二进制码元所含的信息量是一个“比特”，所以信息传输速率的单位是比特/秒(bit/s)。例如一个数字通信系统，它每秒传输 600 个二进制码元，它的信息传输速率是 600 比特/秒(600bit/s)。

2. 符号传输速率

它是指单位时间(秒)内传输的码元数目，其单位为波特。这里的码元可以是二进制的，也可以是多进制的。符号传输速率 M 和信息传输速率 R 的关系为 $R = N\log_2 M$。当码元为二进制时，M 为 2；码元为四进制时，M 为 4。如果符号速率为 600 波特，在二进制时，信息传输速率为 600 比特/秒，在四进制时为 1200 比特/秒。

3. 误码率

信码在传输过程中，由于信道不理想以及噪声的干扰，以致在接收端判决再生后的码元可能出现错误，这种现象就叫误码。误码的多少一般是用误码率来衡量的，误码率是数字通信系统中单位时间内错误码元数与发送总码元数之比。误码越多，误码率越大。

7.2　离散傅里叶变换基础

数字信号处理技术所使用的主要算法多是对数据进行滤波、卷积、相关和谱分析运算，算法的理论基础就是傅里叶分析法；由于离散傅里叶变换及快速傅里叶变换在数字信号处理中应用广泛，本节主要介绍离散傅里叶变换算法及快速傅里叶变换算法。

7.2.1　离散傅里叶变换

傅里叶变换是信号分析和处理的有力工具，在以快速傅里叶变换算法为代表的一系列有效算法出现后，傅里叶变换不但在信号处理领域起着支柱作用，而且在其他工程领域也获得了广泛的应用[6]。

根据信号的连续性、离散性、周期性、非周期性，傅里叶变换可以分为 4 种不同的形式，形成 4 种不同的傅里叶变换对：连续时间非周期信号、连续时间周期信号、离散时间非周期

信号以及离散时间周期信号。其中，前 3 种信号至少在时域或频域是连续的，因此都不适于在计算机上运行。

对于有限长序列，可以得出另外一种傅里叶表示，称为离散傅里叶变换(DFT)。离散傅里叶变换本身是一个序列，而不是一个连续变量的函数，它相当于对信号的傅里叶变换进行频率的等间隔取样的样本。因为计算 DFT 存在高效的算法，所以 DFT 作为序列的傅里叶表示，除了在理论上十分重要外，在实现各种数字信号处理算法中还起着核心作用。

离散傅里叶变换描述的是对有限长序列的分析，其本质是建立了以时间为自变量的信号与以频率为自变量的频谱函数之间的变换关系，换言之，离散傅里叶变换定义了时域与频域之间的一种变换或者说是映射。对于 DFT 而言，时间和频率变量都取离散值。

下面讨论有限长序列的离散傅里叶变换。对于一个长度为 N 的有限长序列 $x(n)$，等效于只在 $n=0$ 到$(N-1)$个点上为非零值，其余皆为零，我们可以把它看成周期为 N 的周期序列$\overline{x(n)}$中的一个周期，即

当 $0 \leqslant n \leqslant N-1$ 时，　　$x(n)=\overline{x(n)}$

当 n 为其他值时，　　$x(n)=0$

其中，

$$\overline{x(n)} = \sum_{r=-\infty}^{\infty} x(n+rN) \tag{7-1}$$

从而，有限长序列的傅里叶变换定义为

$$\text{正变换：} X(k) = \mathrm{DFT}[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad 0 \leqslant k \leqslant N-1$$

$$\text{反变换：} x(n) = \mathrm{IDFT}[X(k)] = \frac{1}{N} \sum_{n=0}^{N-1} X(k) W_N^{-nk} \qquad 0 \leqslant n \leqslant N-1$$

其中，$W_N^{nk} = \mathrm{e}^{-\mathrm{j}kn\frac{2\pi}{N}}$。

由此可见，离散傅里叶变换开辟了频域离散化的道路，使数字信号处理也可以在频域上采用数字运算方法进行。它作为一种数学工具来描述离散信号的时域和频域表示的关系，大大增加了数字信号处理的灵活性，特别是其多种快速算法，使信号的实时处理和设备的简化得以实现。所以，离散傅里叶变换不仅在理论上有重要意义，而且在各种数字信号处理中起着核心的作用。

7.2.2　频域应用

离散傅里叶变换作为傅里叶变换的一种近似而得到广泛应用，它的快速算法保证了 DFT 在实时信号处理中的应用。下面介绍两种常用的频域应用。

1. 功率谱

信号 $x(n)$的离散傅里叶变换 $X(k)$一般是一个复数，$X(k)$与其共轭 $X^*(k)$之积称为自功率谱，简称自谱或功率谱。也有文献称其为功率谱密度(函数)，表示为

$$P(k) = \frac{1}{N} X(k) X^*(k) = \frac{1}{N} |X(k)|^2, \quad 0 \leqslant k \leqslant N-1 \tag{7-2}$$

这里的系数 $1/N$ 只是为了满足能量定理而进行的归一化。$P(k)$反映的是信号的功率密

度，在图形上与幅度谱类似，只是大小不同。功率谱不含相位信息，所以由功率谱不能恢复原始信号。

可以证明，线性自相关函数和功率谱是一对离散时间傅里叶变换对(DTFT)，相应的循环自相关函数和功率谱是一对离散傅里叶变换对(DFT)。考虑序列 $x(n)$可能是复数，且实际得到的 $x(n)$是一段样本序列，因而

$$\begin{aligned}
\mathrm{DFT}[R_{xx}(m)] &= \mathrm{DFT}\left[\frac{1}{N}\sum_{n=0}^{N-1}x(n)x^*\left((n+m)\right)_N R_N(n)\right] \\
&= \frac{1}{N}\sum_{m=0}^{N-1}\left[\sum_{n=0}^{N-1}x(n)x^*\left((n+m)\right)_N R_N(n)\right]W_N^{mk} \\
&= \frac{1}{N}\sum_{n=0}^{N-1}x(n)\sum_{m=0}^{N-1}x^*\left((n+m)\right)_N R_N(n)W_N^{mk} \\
&= \frac{1}{N}\sum_{n=0}^{N-1}x(n)\left[W_N^{-nk}X\left((k)\right)_N\right]^* R_N(n) \\
&= \frac{1}{N}\sum_{n=0}^{N-1}x(n)W_N^{nk}X^*(k) = \frac{1}{N}X(k)X^*(k) = P(k)
\end{aligned} \tag{7-3}$$

当 $x(n)$是实序列时，自相关序列也是实序列，则功率谱是偶对称的。从功率谱和自相关函数之间的关系，我们知道功率谱蕴涵着集合统计的实质。一个随机信号的自相关和功率谱都表达了随机信号的统计平均特性。

2. 频域滤波

理想的数字滤波器幅度谱如图 7-3 所示，包括低通、高通、带通和带阻。这些频率特性都是以 2π 为周期的连续函数。当单位脉冲响应 $h(n)$是实数序列时，幅度谱周期偶对称，相位谱周期奇对称，因而只需要给出一半的频谱图即可。

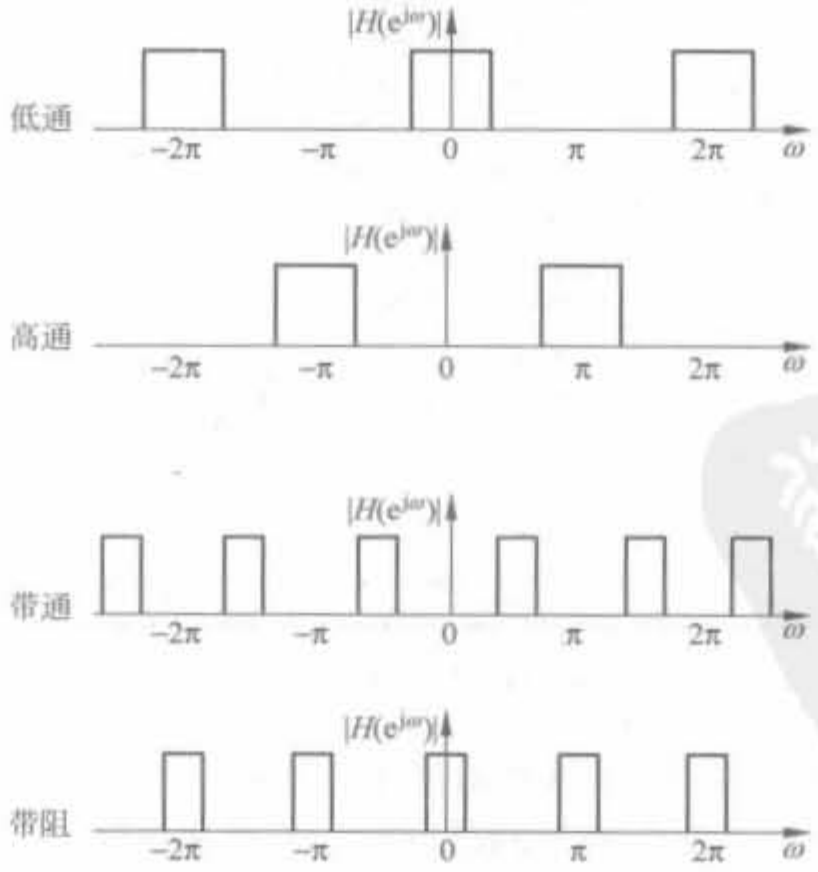


图 7-3　各理想数字滤波器频率特性

数字频域滤波可以用硬件和软件的方法实现，滤波器的设计方法多种多样，大致分为IIR 滤波器设计和 FIR 滤波器设计。前者主要利用传统的模拟滤波器设计方法，后者多采用窗函数和频率取样设计法。限于篇幅，这里不进行详细介绍，有兴趣的读者可参考相关文献。频域滤波如果用软件方法来实现，更为灵活。如不考虑信号的因果性(非实时分析)，则滤波的实现方法很简单，只需令需要滤波的频段所对应的滤波器幅度为 0，再作 IFFT 即可获得滤波后的时域波形，可高效、干净地完成滤波。如果要实现实时分析，就要设计出因果的滤波器，在 MATLAB 中有滤波器的设计工具箱，读者可以参考里面的应用例子。在随后的章节中也将介绍基于 FPGA 的滤波器实现。

7.2.3 FFT/IFFT IP Core 的使用

在 ISE 9.1.03i 版中，提供了 FFT/IFFT 的 IP Core，可以完成实数、复数信号的 FFT 以及 IFFT 运算。FFT 的 IP Core 提供 3 种结构，分别为：①流水线，Streaming I/O 结构：允许连续的数据处理。②基 4，Burst I/O 结构：提供数据导入/导出阶段和处理阶段。此结构拥有较小的结构，但转换时间较长。③基 2，Burst I/O 结构：使用最少的逻辑资源，与基 4 相同，提供两阶段的过程。其配置界面有 3 页，第 1 页如图 7-4 所示，主要用于配置实现结构；第 2 页配置数据位宽以及实现数据处理操作；第 3 页配置数据缓存空间。

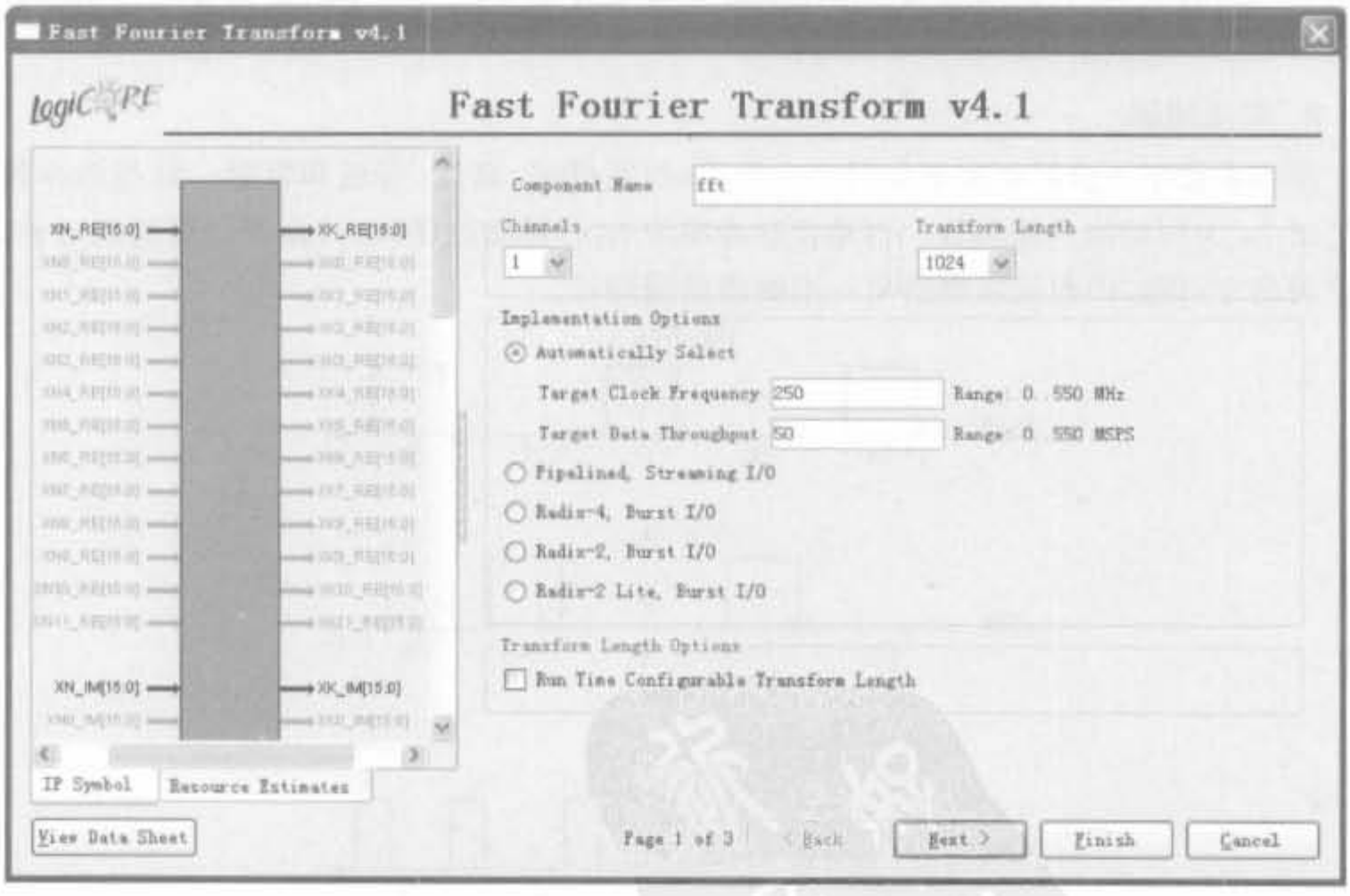


图 7-4 FFT IP Core 的用户界面

在实际硬件操作中，模块的执行速度是很重要的参数，所以本章分析第一种结构，即流水线 Streaming I/O 结构，以进行连续的数据处理。在处理当前帧的 N 点数据时，可加载下一帧的 N 点数据，同时输出前一帧的 N 点数据。此结构由多个基 2 的蝶形处理单元构成，每个单元都有自己的存储单元来存储输入和中间处理的数据，其结构如图 7-5 所示。

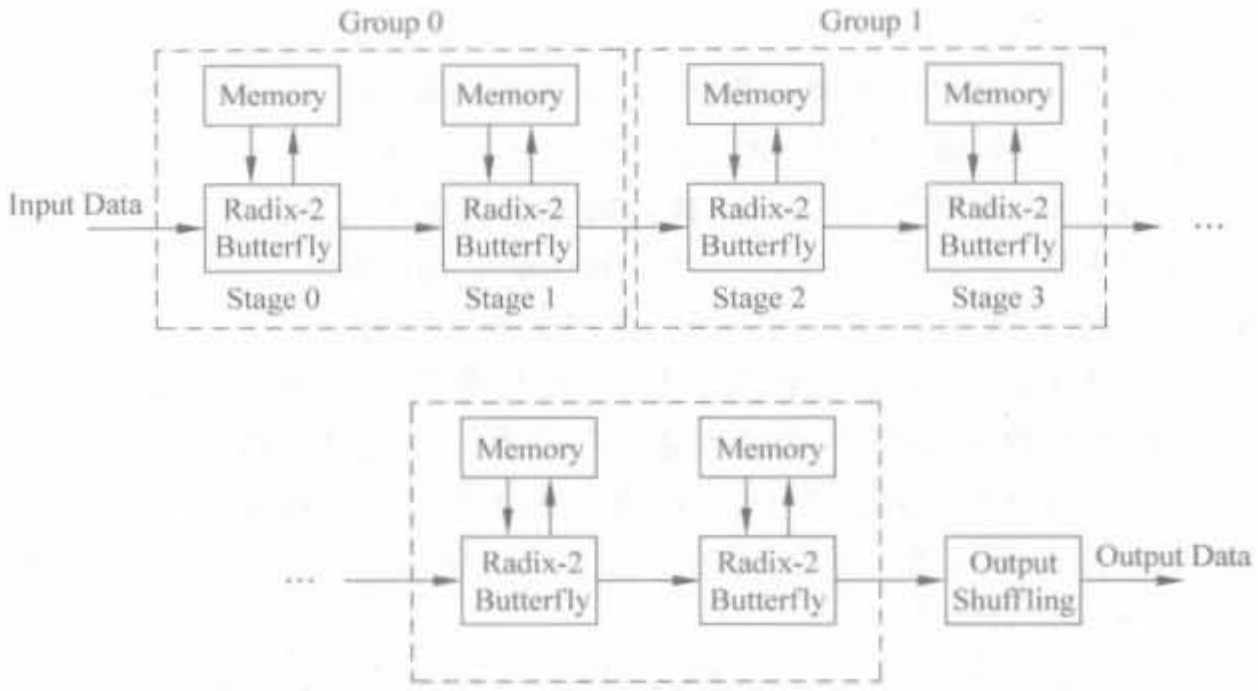


图 7-5 FFT 模块的流水线,Streaming I/O 结构

FFT 的计算单元具有丰富的控制信号,其详细说明见下文。

(1) XN_RE、XN_IM:输入操作数,分别为实部和虚部,以 2 的补码输入。在使用时应当确定其位宽。

(2) START:FFT 开始信号,高电平有效。当此信号变高时,开始输入数据,随后直接进行 FFT 转换操作和数据输出。一个 START 脉冲允许对一帧进行 FFT 转换。如果每 N 个时钟有一个 START 脉冲或者 START 始终为高电平,则都可以连续进行 FFT。如果在最初的 START 前,还没有 NFFT_WE、FWD_INV_WE 及 SCALE_SCH_WE 信号,则 START 变高后就使用这些信号的默认值。由于此 IP Core 支持非连续的数据流,因此在任何时间输入 START,即可开始数据的加载。当加载 N 个数据结束后,就开始 FFT 转换运算。

(3) UNLOAD:对于 Burst I/O 结构,此信号将开始输出处理的结果。对于流水线结构和比特逆序输出的情况,此端口不是必要的。

(4) NFFT:此端口只在实时可配置应用时有用。

(5) NFFT_WE:此端口是 NFFT 端口的使能信号。

(6) FWD_INV:用以指示 IP Core 为 FFT 还是 IFFT,等于 1 时,IP Core 进行 FFT 运算,否则进行 IFFT 运算。至于采用哪种转换运算,是可以逐帧变化的。这一端口给 FFT 的使用提供了很大的方便。

(7) FWD_INV_WE:作为 FWD_INV 端口的使能信号。

(8) SCALE_SCH:①在 IP Core 设计时,如果选择在计算过程中进行中间数据的缩减,那么此信号才可起作用。②输入的位宽等于 $2\times\mathrm{ceil}(\mathrm{NFFT}/2)$,其中 $\mathrm{NFFT}=\log_2(\text{点数})$。③流水线结构中,将每个基 2 的蝶形处理单元视为一个阶段,每个阶段进行一次数据的缩减,缩减的比例以此输入中对应阶段的 2 比特表示。④每阶段的 2 比特数可以是 3、2、1 或 0,它们表示了数据所需要移动的比特数。

(9) SCALE_SCH_WE:作为 SCALE_SCH 的使能信号。

(10) SCLR:可选端口。

(11) Reset:重置信号端口。Reset=1 时,所有工作都停止且初始化,但内部的帧缓存保留其内容。

(12) CE:可选端口。

(13) CLK：输入时钟。

(14) XK_RE,XK_IM：输出数据总线，以 2 的补码输出。SCALE_SCH_WE 有效时，输出位宽等于输入；否则，输出位宽＝输入位宽＋NFFT＋1。

(15) XN_INDEX：位宽等于 $\log_2$(点数)，输入数据的下标。

(16) XK_INDEX：位宽等于 $\log_2$(点数)，输出数据的下标。

(17) RFD：数据有效信号，高电平有效，在加载数据时为高电平。

(18) BUSY：IP Core 工作状态的指示信号，在计算 FFT 转换时为高电平。

(19) DV：数据有效指示信号，当输出端口存在有效数据时变高。

(20) EDONE：高电平有效。在 DONE 信号变高的前一个时钟变为高电平。

(21) DONE：高电平有效。在 FFT 完成后变高，且只存在一个时钟。在 DONE 变高后，IP Core 开始输出计算结果。

(22) BLK_EXP：当使用 Burst I/O 结构时可用，若选择流水线，此端口无效。

(23) OVFLO：算法溢出指示。在数据输出时，如每帧有溢出，此信号变高。在每帧开始处，此信号重置。

例 7-1 使用 IP Core 实例化一个 16 点、位宽为 16 位的 FFT 模块。

IP Core 直接生成的乘法器的 Verilog 模块接口为：

```
module fft16 (sclr,fwd_inv_we,rfd,start,fwd_inv,dv,scale_sch_we,done,clk,busy,edone,scale_
            sch,xn_re,xk_im,xn_index,xk_re,xn_im,xk_index);

  input sclr,fwd_inv_we,start,fwd_inv,scale_sch_we,clk;
  input [3:0] scale_sch;
  input [15:0] xn_re;
  output rfd,dv,done,busy,edone;
  output [15:0] xk_im;
  output [3:0] xn_index;
  output [15:0] xk_re;
  input [15:0] xn_im;
  output [3:0] xk_index;

...
endmodule
```

在使用时，直接调用 multiply 模块即可，如

```
module fft16 (sclr,fwd_inv_we,rfd,start,fwd_inv,dv,scale_sch_we,done,clk,busy,
            edone,scale_sch,xn_re,xk_im,xn_index,xk_re,xn_im,xk_index);

  input sclr,fwd_inv_we,start,fwd_inv,scale_sch_we,clk;
  input [3:0] scale_sch;
  input [15:0] xn_re;
  output rfd,dv,done,busy,edone;
  output [15:0] xk_im;
  output [3:0] xn_index;
  output [15:0] xk_re;
  input [15:0] xn_im;
  output [3:0] xk_index;

fft fft1( //调用 FFT 的 IPCore
```

```
.sclr(sclr),.fwd_inv_we(fwd_inv_we),.rfd(rfd),.start(start),.fwd_inv(fwd_inv),
.dv(dv),.scale_sch_we(scale_sch_we),.done(done),.clk(clk),.busy(busy),
.edone(edone),.scale_sch(scale_sch),.xn_re(xn_re),.xk_im(xk_im),
.xn_index(xn_index),.xk_re(xk_re),.xn_im(xn_im),.xk_index(xk_index));

endmodule
```

经过仿真测试得到的功能波形图如图 7-6 所示。

图 7-6　FFT 的 IP Core 仿真波形

7.3　Xtreme DSP 模块功能介绍

本节将对 Xtreme DSP 数字信号处理(DSP)组件 DSP48 Slice(块)进行介绍。DSP48 Slice 可以高效地执行大量的算术功能，包括加法器、减法器、累加器、MAC、乘法多路复用器、计算器、除法器、平方根函数和移位器。DSP Tile 中任选的流水线长度(级数)确保了高性能算法功能的实现。DSP48 Column 的结构及相关布线在 DSP Tile 间实现了快速的数据传输，并减少了到 FPGA 结构的布线阻塞。

Xtreme DSP 最基本的组件就是硬核乘加器，其模块如图 7-7 所示，它使得 Virtex-5/Virtex-4/Spartan-3 系列 FPGA 可以为高性能的数字信号处理提供理想的解决方案，达到传统上由 ASIC 或 ASSP 完成的高性能信号处理能力。DSP48 Slice 最初是随着 Virtex-4 FPGA 的发布而推出的，它具有“面向应用的组合模块”(ASMBL)架构，能提供 Virtex DSP 器件中的 DSP 功能。每个 Xtreme DSP Tile 都包含两个 DSP48 Slice，构成了一个通用粗粒度 DSP 架构的基础。

这种 DSP48 Slice 支持许多独立功能，包括乘法器、MAC、乘法器带加法器、3 输入加法器、桶式移位器、宽总线多路复用器、量级比较器或宽计数器。这种架构也支持将多个 DSP48 Slice 连在一起形成宽数学函数、DSP 过滤器和复杂算术函数，而无需使用总体 FPGA 架构，从而降低了功耗，同时达到高的性能和芯片使用率。

DSP48 模块是一个 18×18 位二进制补码乘法器，跟随一个 48 位符号扩展的加法器/减法器/累加器，适应 DSP 应用中的众多功能。它提高了操作数输入、中间积和累加器输出的可编程流水线操作，以及 48 位内部总线等的吞吐量和适应性，无需一般的结构布线就可以实现前一个 DSP48 的输出与后一个 DSP48 输入的级联，增强了它的功能。

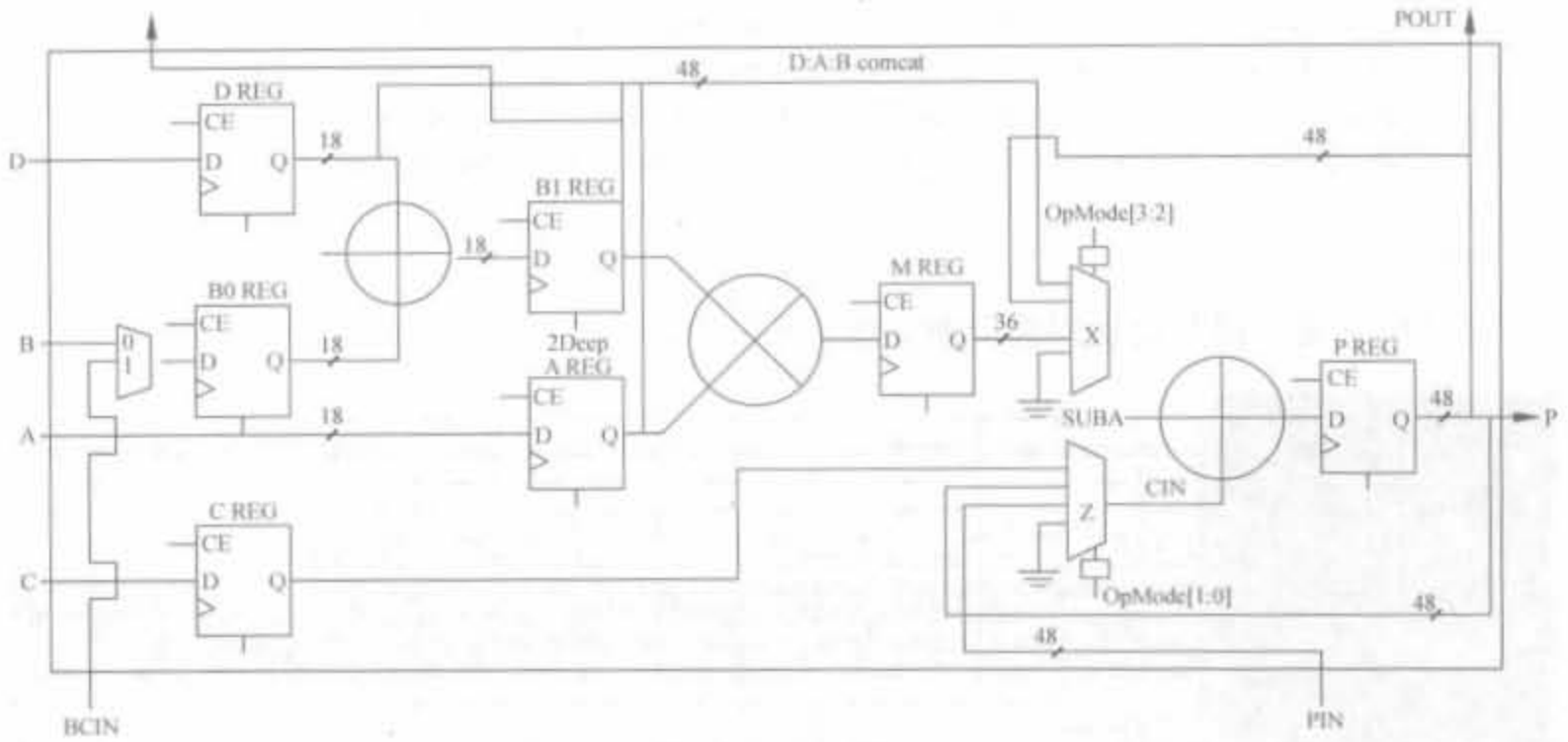


图 7-7　Xilinx Xtreme DSP 模块结构

采用数字技术对复杂算法进行硬件实现时，首先遇到的问题是没有预先规定的结构，因此需要首先对算法建模和仿真进行优化。与基于 RTL 针对结构清晰的设计方法不同，算法设计把焦点从针对结构的细节转移到对设计的整体要求和行为，在最高的算法层次上考虑如何进行设计，对系统的行为描述定义了设计要执行的算法，不涉及或很少涉及实现细节，因此行为描述比 RTL 描述要简洁得多。

7.4　乘累加结构的 FIR 滤波器

信号处理工程师可以采用多种方法来实现滤波器，但常见的滤波器可以采用单乘法器 MAC FIR 滤波器实现。本节主要用 DSP48 Slice 来实现有限脉冲响应(FIR)滤波器，主要说明两种串行滤波器架构：单乘法器 MAC FIR 滤波器和双乘法器 MAC FIR 滤波器。

7.4.1　单乘法器 MAC FIR 滤波器

单乘法器 MAC FIR 是最简单的 DSP 滤波器结构之一。与全并行滤波器相比，MAC 结构采用单一乘法器和累加器来顺序实现 FIR 滤波器。该折衷设计不仅使硬件数量减少了 N 倍，也使滤波器吞吐量按照同样的比例下降。通用的 FIR 滤波器公式是乘积(也称为内积)的总和，定义如下：

$$y_n = \sum_{i=0}^{N-1} x_{n-i} h_i \tag{7-4}$$

在该公式中，N 个系数与 N 个相应的数据采样相乘，然后对内积求和来产生单个结果。这里的系数值确定了滤波器的特性(如低通滤波器、带通滤波器、高通滤波器)。该公式可以采用不同架构、利用不同方法(如串行、半并行或并行)来实现。在采样速度慢、系数多时，采用单一的 MAC FIR 滤波器来实现非常合适。利用高速时钟来驱动乘加器，实现对低速数据的多倍计算，其结构如图 7-8 所示。

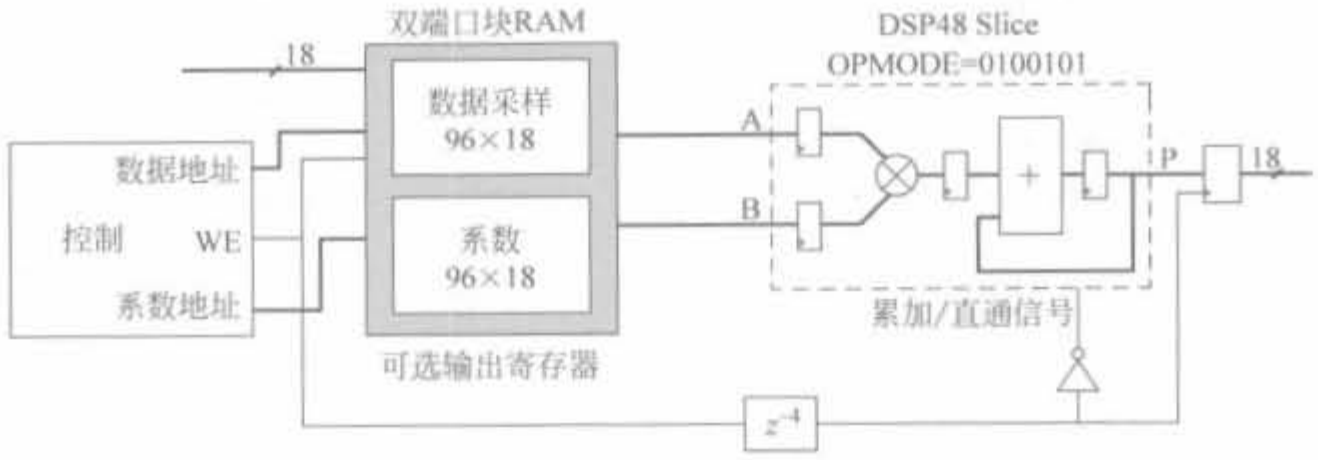


图 7-8 单乘法器 MAC FIR 滤波器

下面给出一个 MAC FIR 乘法器的实例。

例 7-2 使用 DSP48 Slice 并基于单乘法器结构，完成一个 4 阶 8bit 的 FIR 乘法器，数据速率为 1Mb/s，其系数为[8'd87,8'd127,8'd157,8'd137]。

为了通过 DSP48 Slice 完成单乘法器结构，其工作频率应为数据速率的 4 倍。此外，滤波计算所需的输入数据缓存在分布式 RAM 中。

```
module mac_fir_tap4(clk_4MHz,reset,ce,din,dout);
  input          clk_4MHz;
  input          reset;
  input          ce;
  input  [7:0] din;
  output [17:0] dout;

reg [17:0] dout;
wire [17:0]  dout1;
wire [7:0]   dtemp;
reg  [1:0]   cnt = 0;
reg          we = 0;
reg  [3:0]   spra = 0;

reg  [7:0]   A_IN = 0;
reg  [7:0]   B_IN = 0;
reg          LOAD_IN = 0;

// 控制数据缓冲期
always @(posedge clk_4MHz) begin
    if(reset) begin
        cnt<=0;
        we<=0;
        LOAD_IN<=0;
        dout<=0;
    end
    else begin
        spra<={1'b0,cnt}+3'b001;
        cnt<=cnt+1;
        if(cnt==2'b11) begin
            we<=1;
```

```
                LOAD_IN<=0;
            end
            else begin
                we<=0;
                LOAD_IN<=1;
            end
            if(cnt==2'b01)
                dout<=dout1;
            else
                dout<=dout;
        end
    end

    // 完成滤波器乘加计算
    always @(posedge clk_4MHz) begin
        A_IN<=dtemp;
        case(cnt)
            2'b10: B_IN<=8'd87;
            2'b11: B_IN<=8'd127;
            2'b00: B_IN<=8'd157;
            2'b01: B_IN<=8'd137;
      endcase
    end

    //
  fir_dram fir_dram(
      .CLK(clk_4MHz),
      .D(din),
      .WE(we),
      .QSPO_CE(ce),
      .SPRA(spra),
      .QSPO(dtemp)
      );

    //
    fir_mac fir_mac (
      .A_IN(A_IN),
      .B_IN(B_IN),
      .CE_IN(ce),
      .CLK_IN(clk_4MHz),
      .LOAD_IN(LOAD_IN),
      .RST_IN(reset),
      .P_OUT(dout1)
      );

endmodule
```

上述程序经过 Synplify Pro 综合后，其 RTL 级结构如图 7-9 所示。其中只使用了一个硬核乘加器完成滤波计算，数据缓冲以及移位利用 Xilinx 公司的 SRL16LUT 结构来实现。这种实现方法可节省很多硬件资源。

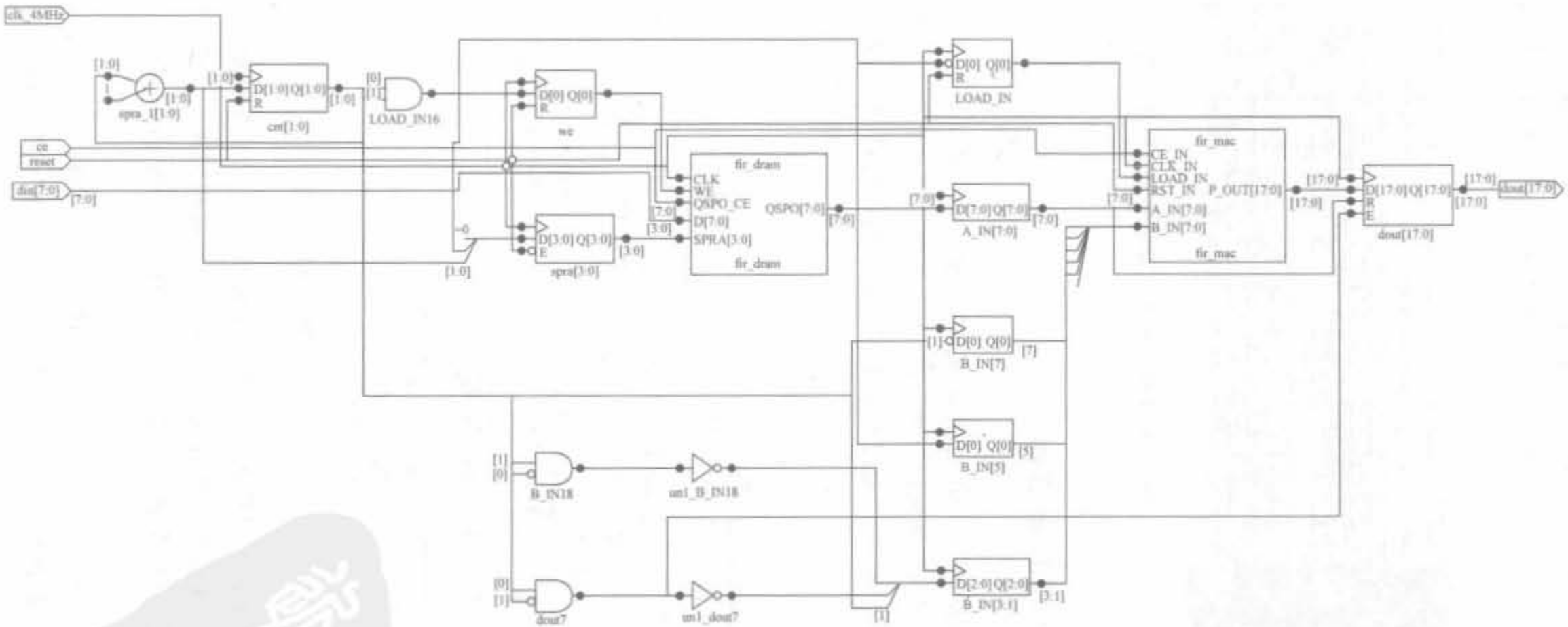


图 7-9　单 MAC 结构 FIR 滤波器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 7-10 所示。模块处理时钟是输入数据的 4 倍，例如当输入为 14、13、12 和 11 时，分别和系数相乘，再将结果累加，经过 8 个时钟周期后，输出滤波输出 372。

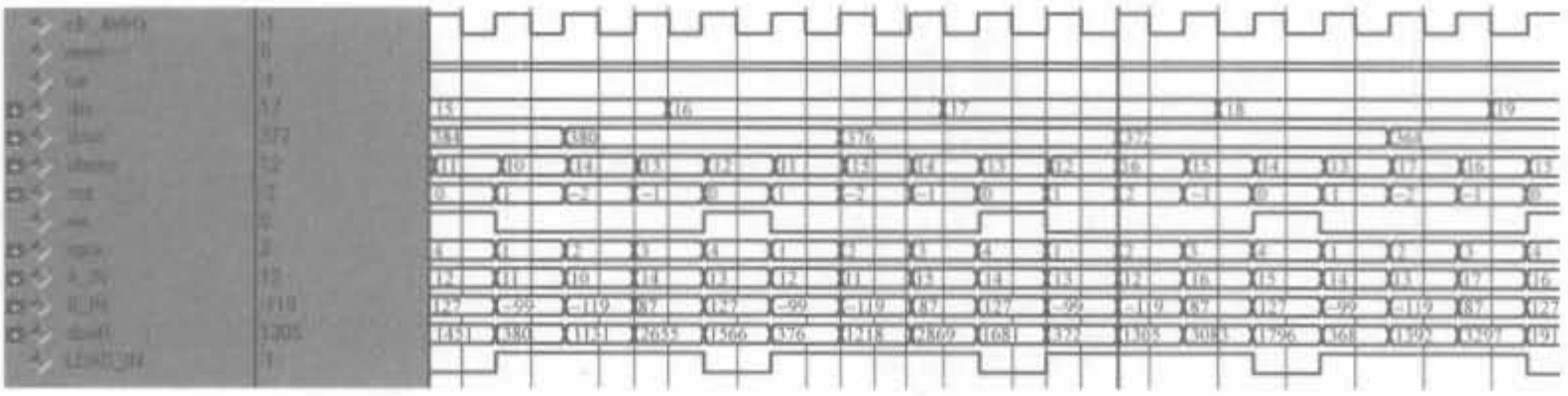


图 7-10　单 MAC 结构 FIR 滤波器的仿真结果示意图

7.4.2　对称 MAC FIR 滤波器

除了上节的单乘法 MAC FIR 滤波器，滤波器的实现也可以采用其他方法。本节说明根据 FIR 滤波器系数的对称性如何设计滤波器，可在同样的硬件资源的前提下使采样率性能翻倍(假定具有相同的工作时钟)。通过重新整理 FIR 滤波器公式，系数按如下方式使用：

$$(X_0 \times C_0)+(X_n \times C_n) \rightarrow (X_0 + X_n) \times C_0 \quad (\text{如果 } C_0 = C_n) \tag{7-5}$$

图 7-11 说明了对称 MAC FIR 滤波器的架构。

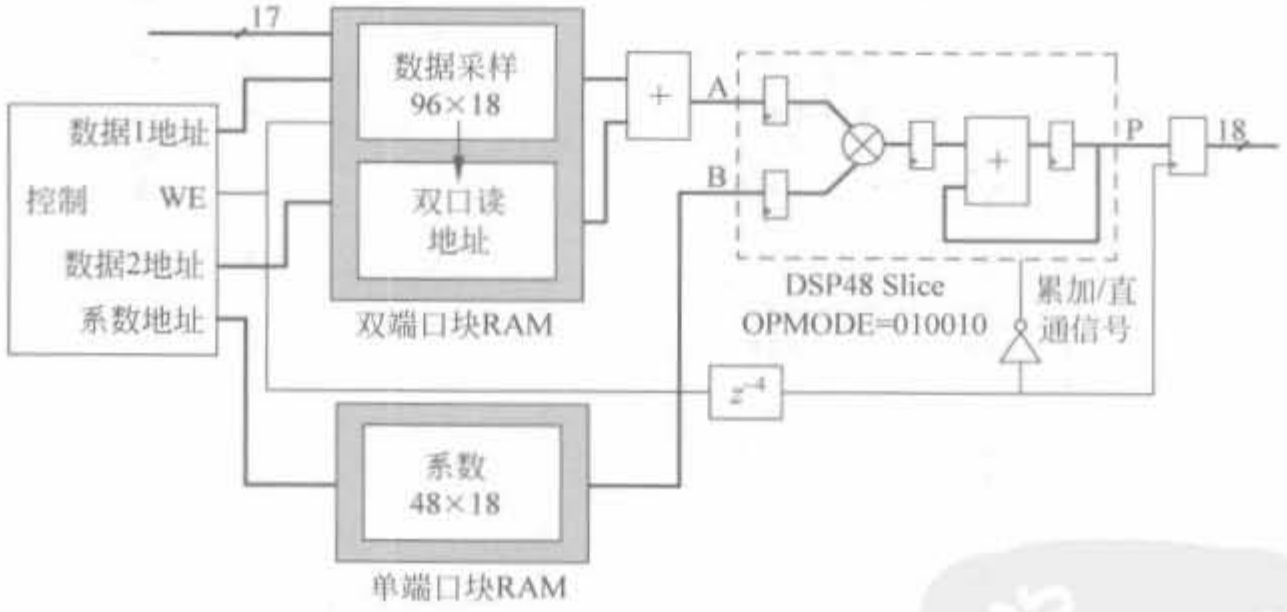


图 7-11　对称 MAC FIR 滤波器

一般都是利用基于逻辑结构的加法器将数据相加，代表了通过滤波器的关键通道，并限制了最高时钟速度，需要额外的资源支持对称。由于数据是在一个端口上正向读出和在另一个端口上反向读出，因此控制部分提高了一定的资源消耗，但相比之下可以节省乘法器这种更宝贵的资源。

例 7-3　使用 DSP48 Slice 并基于对称乘法器结构，完成一个 4 阶 8bit 的 FIR 乘法器，数据速率为 20Mb/s，乘加器模块工作频率为 40MHz，滤波器系数为[8'd7,8'd17,8'd17,8'd7]。

```
module mac_fir2_tap4(clk_80MHz,reset,ce,din,dout);
  input          clk_80MHz;
  input          reset;
  input          ce;
  input  [7:0]   din;
  output [18:0]  dout;

  reg  [18:0]   dout;
  wire [18:0]   dout1;
  wire [7:0]    dtemp;
  wire [8:0]    dtemp1;
  wire          clk_40MHz;
  reg  [1:0]    cnt = 0;
  reg           we = 0;
  reg  [3:0]    spra = 0;
  reg           bypass = 0;
  reg  [8:0]    A_IN = 0;
  reg  [8:0]    B_IN = 0;
  reg           load = 0;

  assign clk_40MHz = cnt[0];

  // 控制数据缓冲
  always @(posedge clk_80MHz) begin
    if(reset) begin
      cnt<=0;
      we<=0;
      load<=0;
      bypass<=0;
      A_IN<=0;
      B_IN<=0;
      dout<=0;
  end
  else begin
      cnt<=cnt+1;
      case(cnt)
          2'b00: begin
                    spra<=3'b000;
                      bypass<=0;
                      A_IN<=dtemp1;
                      B_IN<=9'd7;
                      load<=0;
                      we<=0;
          end
          2'b01: begin
                    spra<=3'b011;
                      bypass<=1;
                      dout<=dout1;
                      A_IN<=A_IN;
                      B_IN<=B_IN;
                      load<=0;
                      we<=0;
          end
          2'b10: begin
```

```
                    spra<=3'b001;
                    bypass<=0;
                    A_IN<=dtemp1;
                    B_IN<=9'd17;
                    load<=1;
                    we<=0;
                end
                2'b11: begin
                    spra<=3'b010;
                    bypass<=1;
                    A_IN<=A_IN;
                    B_IN<=B_IN;
                    load<=1;
                    we<=1;
                end
            endcase
        end
    end

//
fir_dram fir_dram(
    .CLK(clk_80MHz),
    .D(din),
    .WE(we),
    .QSPO_CE(ce),
    .SPRA(spra),
    .QSPO(dtemp)
);

//
fir_adder fir_adder(
    .B(dtemp),
    .Q(dtemp1),
    .CLK(clk_80MHz),
    .BYPASS(bypass)
    );

//
fir_mac2 fir_mac2 (
    .A_IN(A_IN),
    .B_IN(B_IN),
    .CE_IN(ce),
    .CLK_IN(clk_40MHz),
    .LOAD_IN(load),
    .RST_IN(reset),
    .P_OUT(dout1)
    );

endmodule
```

上述程序经过 Synplify Pro 综合后，其 RTL 级结构如图 7-12 所示。在乘加器模块前，添加了累加预处理单元。

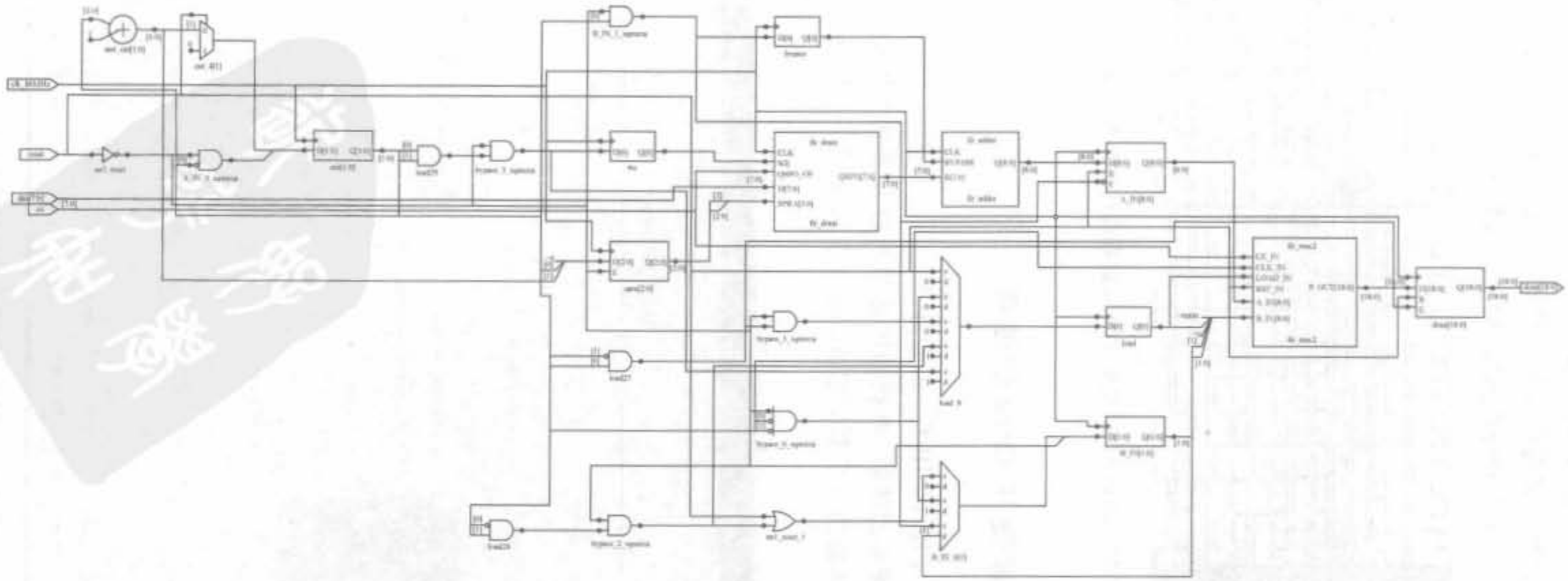


图 7-12　对称 MAC 结构 FIR 滤波器的 RTL 级结构图

在 ModelSim 6.2b 中完成仿真，其结果如图 7-13 所示。模块处理时钟是输入数据的 4 倍，但乘加器的工作时钟为输入数据的 2 倍。例如，当输入为 24、25、26 和 27 时，分别和系数相乘，再将结果累加，经过 9 个时钟周期后，输出滤波输出 1066。

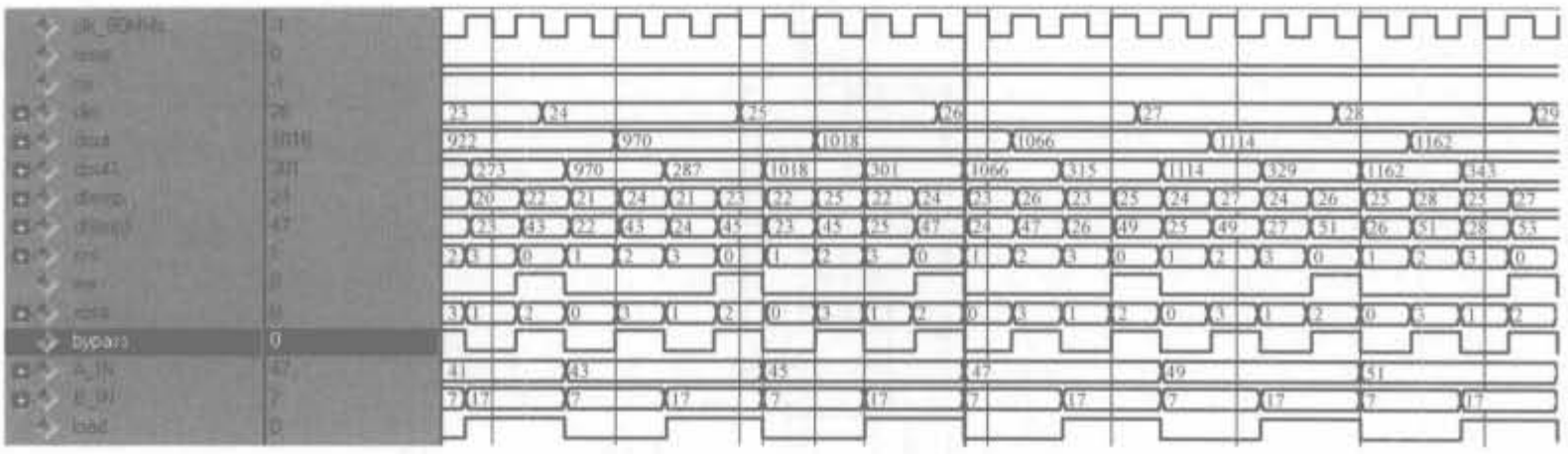


图 7-13　对称 MAC 结构 FIR 滤波器的仿真结果示意图

7.4.3　MAC FIR 滤波器 IP Core 的使用

Xilinx 提供了 MAC FIR 的 IP Core，位于 IP Core 列表栏 Digital Signal Processing Filter 目录下，可实现单速率滤波器、多相插值滤波器以及多相抽取滤波器，且支持通道数为 1～256。其用户配置界面由 5 个页面组成。第 1 页面如图 7-14 所示。单击“Next”按钮，进入下一页。

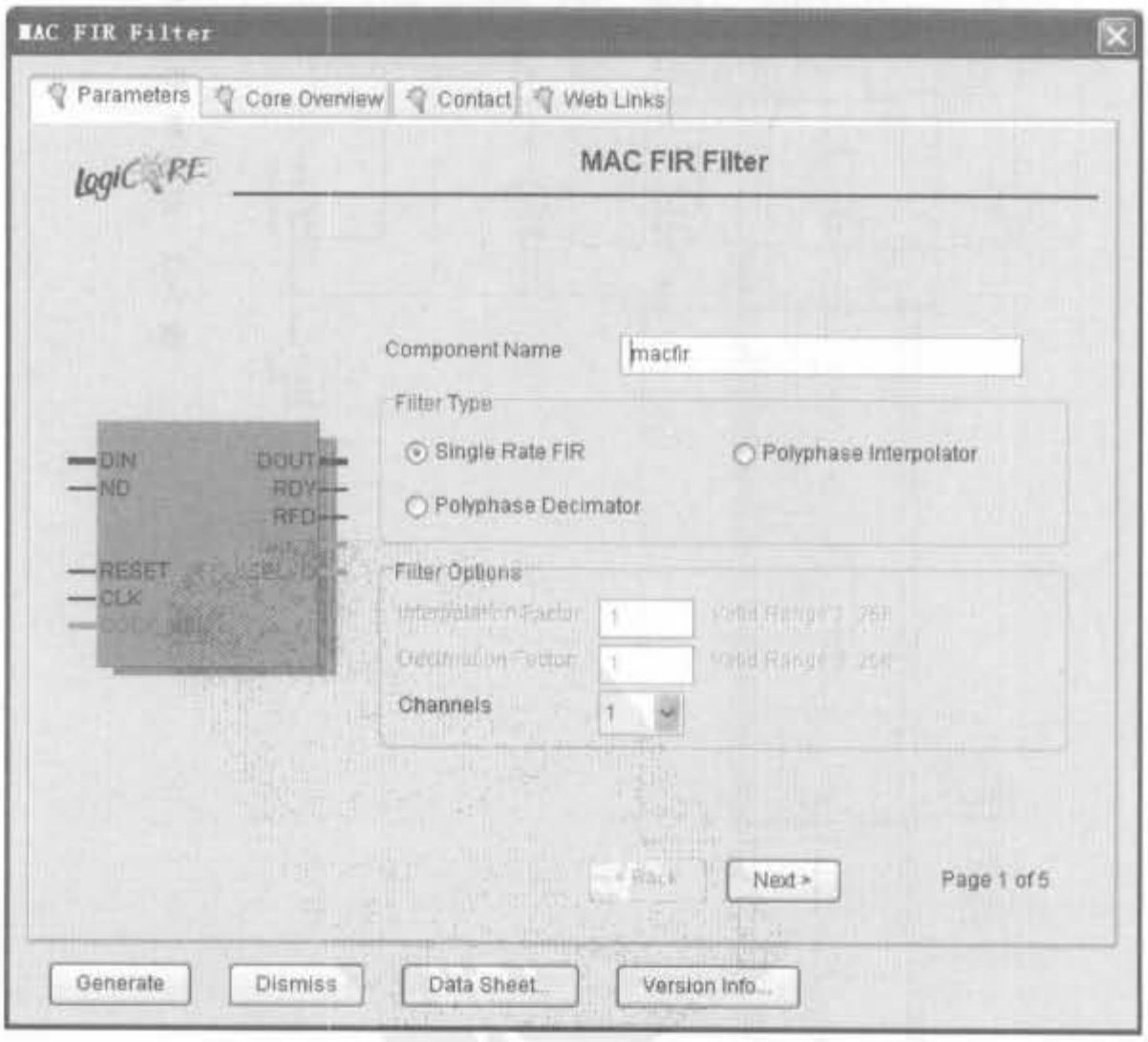


图 7-14　MAC FIR 滤波器配置界面(1)

第 2 页主要完成滤波器系数的配置，如图 7-15 所示。“Structure”栏的“Taps”文本框用于输入滤波器的抽头数，有效范围为 1～1024。“Impulse Response”栏用于设定滤波器冲激响应的形状，有 Symmetric(对称)、Non Symmetric(非对称)和 Negative Symmetric(负对称)3 种选择。一般来讲，FIR 滤波器的冲激响应都是对称的，可节省一半的资源。“Coefficients”栏的“Coefficient Width”用于输入系数位宽，“Number of Coefficient Sets”文本栏用于输入系数集的个数，“Coefficient Type”用于选择系数是有符号数还是无符号数，“Coefficient Buffer Type”用于选择将系数存放在块 RAM 中还是分布式 RAM 中。

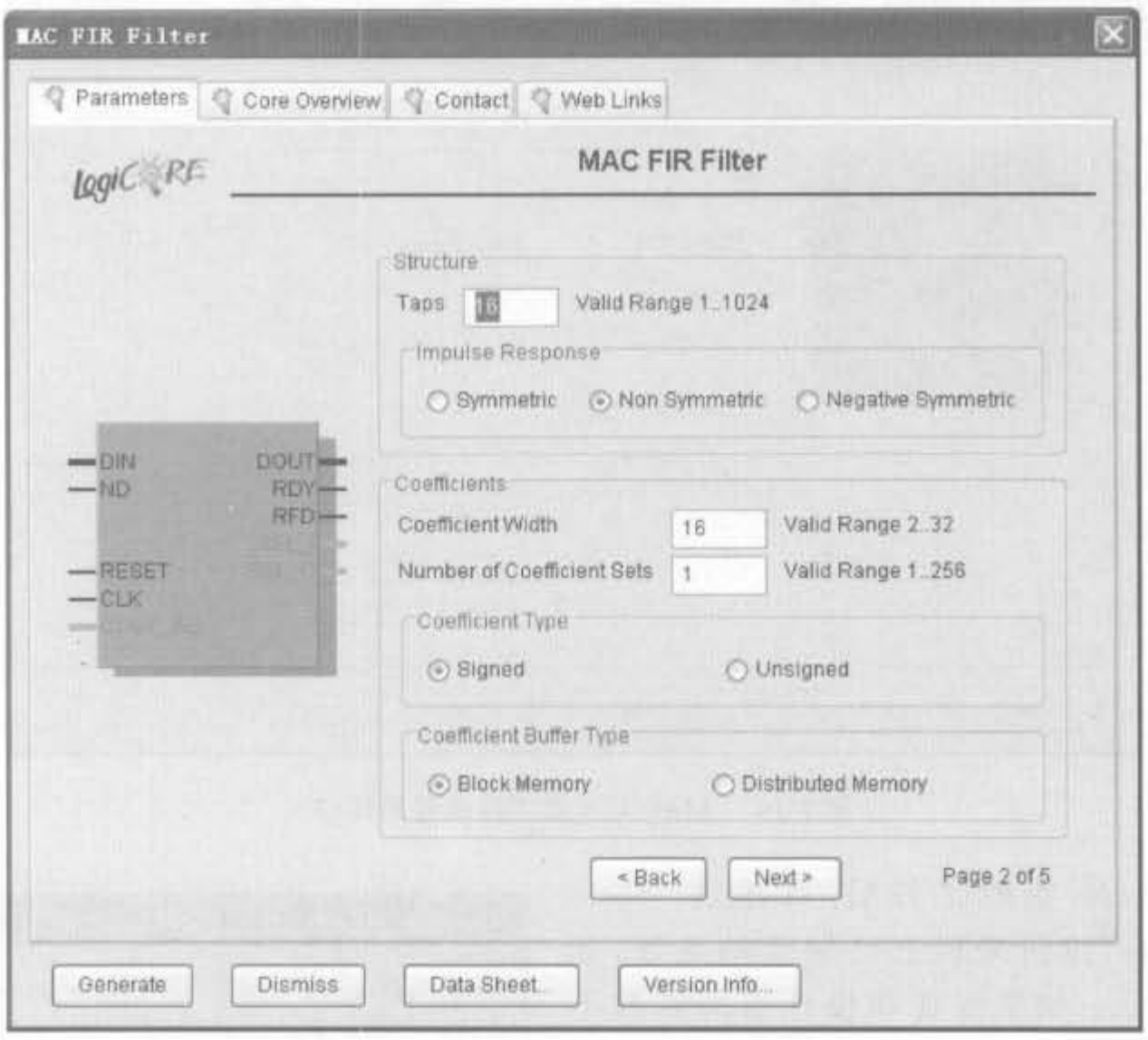


图 7-15　MAC FIR 滤波器配置界面(2)

第 3 页用于加载滤波器系数以及配置输入数据的类型，如图 7-16 所示。滤波器的系数要以.coe 文件的形式导入 IP Core。这里的 COE 文件格式和块 RAM 的 COE 文件格式略有不同，其格式如下所示：

```
radix = 10;
coefdata = …;
```

在本例中，加载的 COE 文件为：

```
radix = 10;
coefdata = -687, -1944,  -333,  2023,  -490, -3725,  3453,15920,
        15920,  3453, -3725,  -490,  2023,  -333, -1944,  -687;
```

如果加载的.coe 文件格式有错，或其和第 2 页面输入的抽头数不匹配，则“.COE File

Name"文本框的内容会以红色显示。"Data Type"用于选择输入数据是有符号数还是无符号数。"Data Buffer Type"用于选择将滤波器的寄存数据存放在块 RAM 中还是分布式 RAM 中。

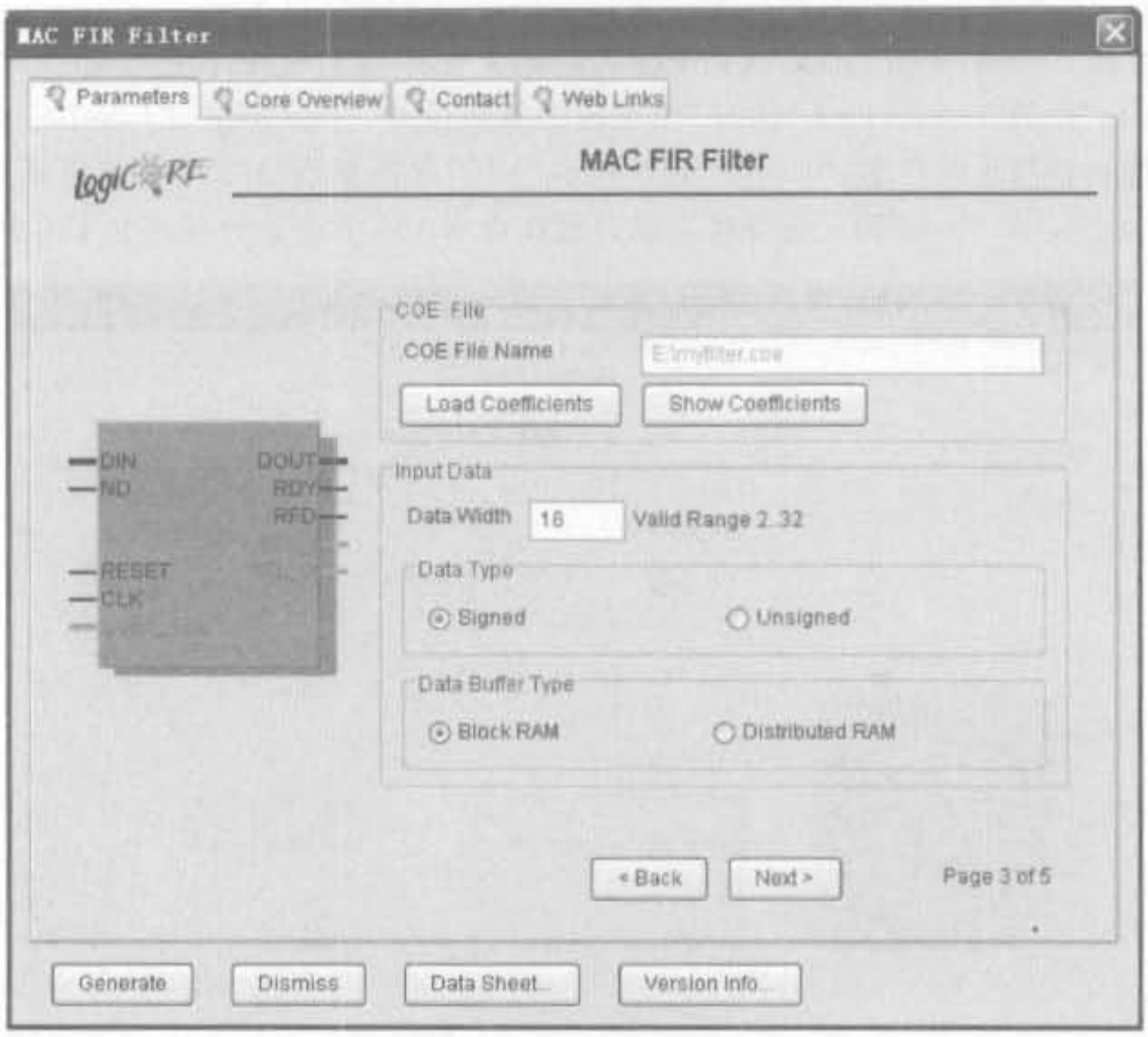


图 7-16　MAC FIR 滤波器配置界面(3)

加载. coe 系数文件后，可单击"Show Coefficients"按钮来核查已加载的系数，如图 7-17 所示。如果发现和设计出的系数不同，需要修改. coe 文件，重新加载并确认。

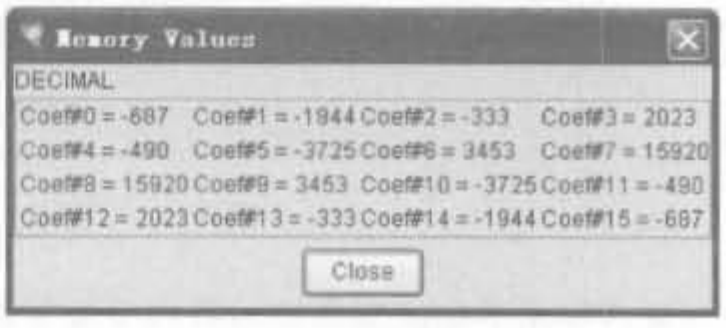


图 7-17　MAC FIR 滤波器系数查看器

单击"Next"按钮，进入速率配置界面，如图 7-18 所示。其中，"Performance Optimization"栏设定模块的优化指标，有 Auto(自动)、Minimum Area(面积最小化)、Minimal Area/High Speed(高速下的面积最小化)以及 Maximum Speed(速度最大化)供选择。"System Clock Rate"文本框用于设定滤波器工作频率；"Input Sample Rate"用于设定输入数据速率，这两个速率的比值越大，意味着可以完成更多倍数的复用，节省更多的资源。"Layout"用于设定模块版图，在高速设计中，即工作时钟超过 100MHz 时，尽量选中"Registered Output(输出寄存)"和"Create RPM(Relatively Placed Macros，相对布局宏)"这两项，后者对高速设计性能的提高有明显的影响。配置完成后，单击"Next"按钮进入下一页。

最后一页显示出了模块的配置信息、输出位宽以及延迟等信息，如图 7-19 所示。确认无误后，即单击"Generate"按钮，生成 MAC FIR 滤波器的 IP Core。

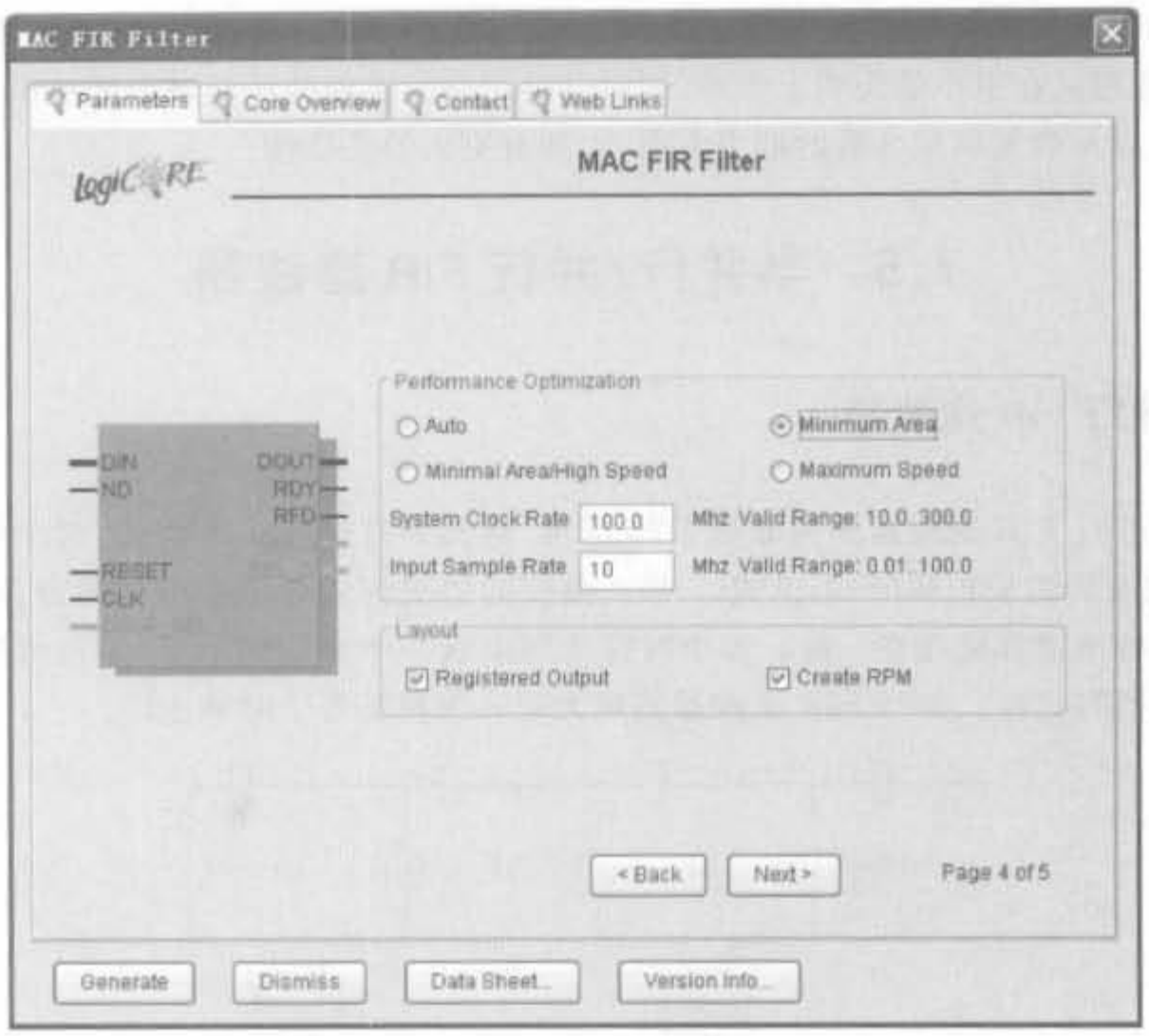


图 7-18 MAC FIR 滤波器配置界面(4)

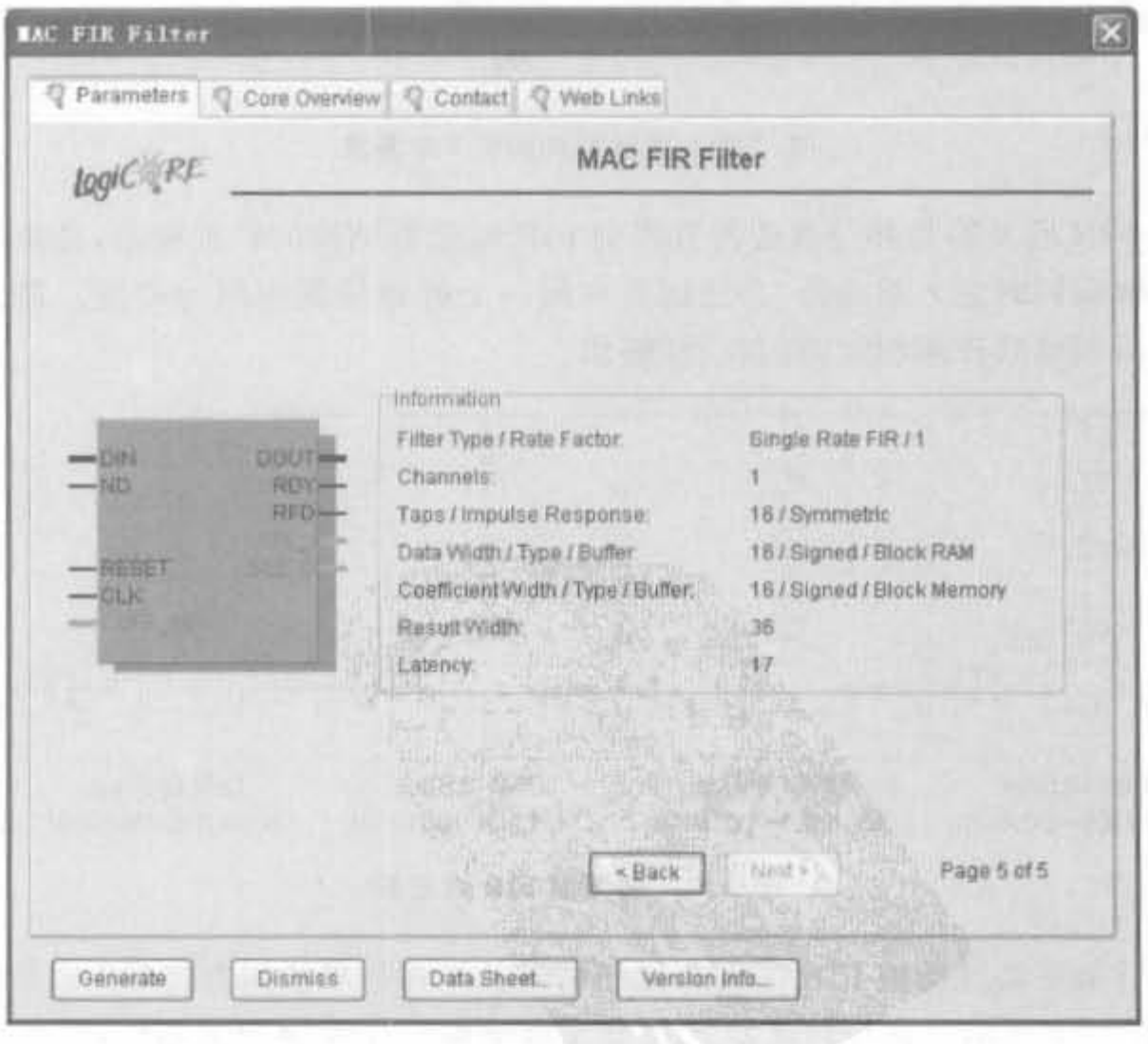


图 7-19 MAC FIR 滤波器配置界面(5)

MAC FIR 滤波器通常面向低速 DSP 应用，采用 Xilinx FPGA 芯片内部的 DSP48 Slice，该功能可能在更小的面积上实现，同时获得更高的性能并消耗更少的功率。在滤波器的实现方式以及改变实现参数的能力方面，它拥有更大的灵活性。

7.5 半并行/并行 FIR 滤波器

7.5.1 并行 FIR 滤波器

基本的并行 FIR 滤波器架构如图 7-20 所示，称其为直接形式类型 I。这个结构实现了通用 FIR 滤波器的乘积和形式，见式(7-4)。数据的历史记录存储在单独的寄存器中，这些寄存器在架构的顶部链接在一起。每个时钟周期生成一个全新的结果，而且所有的乘法和所需的算法同时进行。并行 FIR 滤波器的最大输入采样率等于时钟速率。

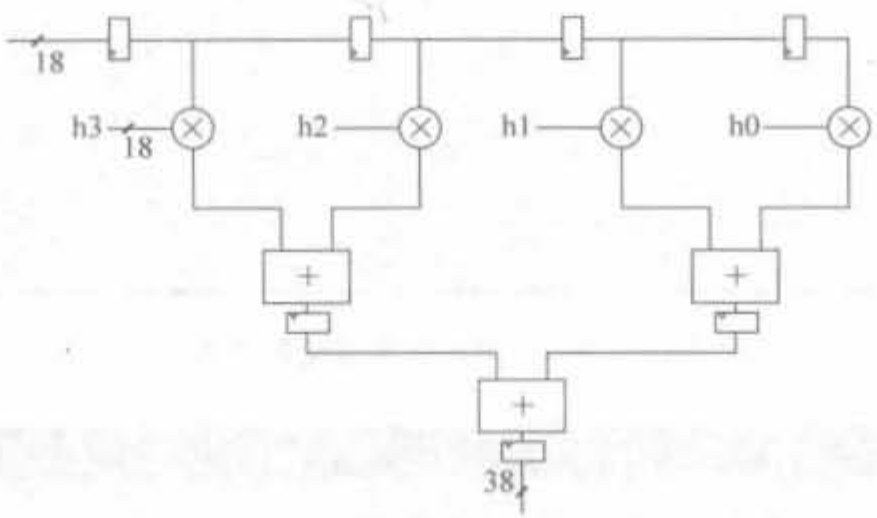


图 7-20 直接形式类型 I 滤波器

转置型 FIR 滤波器是并行乘法器直接型 FIR 滤波器结构的转置变形，其架构如图 7-21 所示。采样数据同时进入乘法器，乘法运算在同一个时钟周期内同时完成。通过流水线的加法器链将乘积结果流水相加，得到滤波输出。

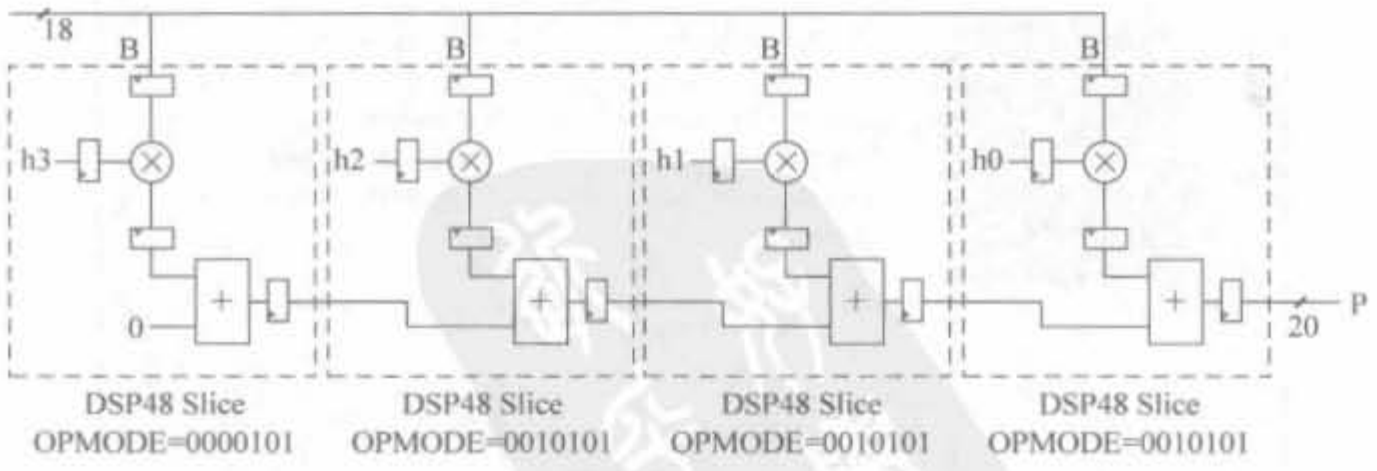


图 7-21 转置型 FIR 滤波器

基于并行乘法器脉动型 FIR 滤波器的结构是基于并行乘法器直接型 I FIR 滤波器的优化模型，如图 7-22 所示。它与转置型 FIR 滤波器结构一样使用加法器链，只是系数安排刚好与转置型相反；而与直接型相比，系数排列相同。采样数据保存在作为数据缓存和时间

延时器的寄存器链中，每个寄存器将采样数据传递给各自的乘法器，分别与各自的滤波器系数相乘，乘积结果通过加法器链流水相加，最后得到滤波输出。

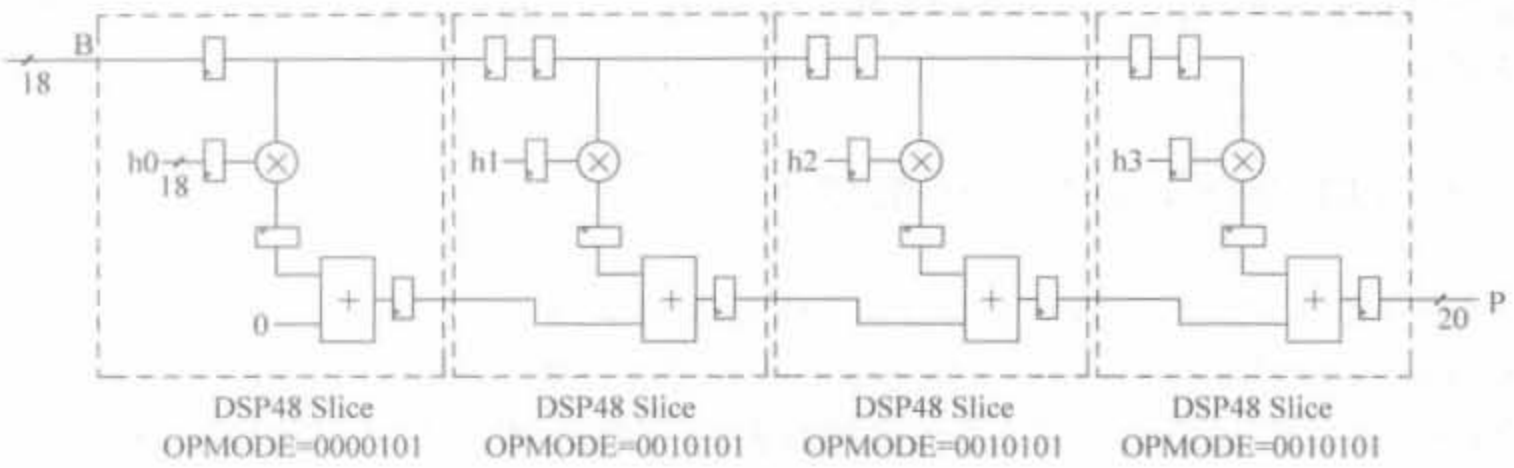


图 7-22 脉动型 FIR 滤波器

7.5.2 半并行 FIR 滤波器

选择不同的结构来实现 FIR 滤波器，有各自的优缺点，因此在考虑系统处理速度要求和系统输入扇出的情况下，必须合理选择 FIR 滤波器结构。乘累加结构在很大程度上能够节省硬件资源，但造成滤波器要在多时钟周期后才有输出，处理速度很慢，不能满足对高速系统的要求。并行乘法器结构尽管使处理速度提高了，但是它大量地耗费硬件资源，造成设计成本很高。对于某些系统，对信号处理速度要求不是很高，使用高速的并行乘法器就显得很不经济了。因此在系统要求的处理速度高于乘累加结构速度，但又成倍低于并行乘法器结构时，可以采用基于乘法器的半并行 FIR 滤波器结构。其原理就是将乘累加结构和三种并行结构中的一种结合起来，组合成满足自己系统要求的滤波器结构。图 7-23 给出了其中的一种组合形式。

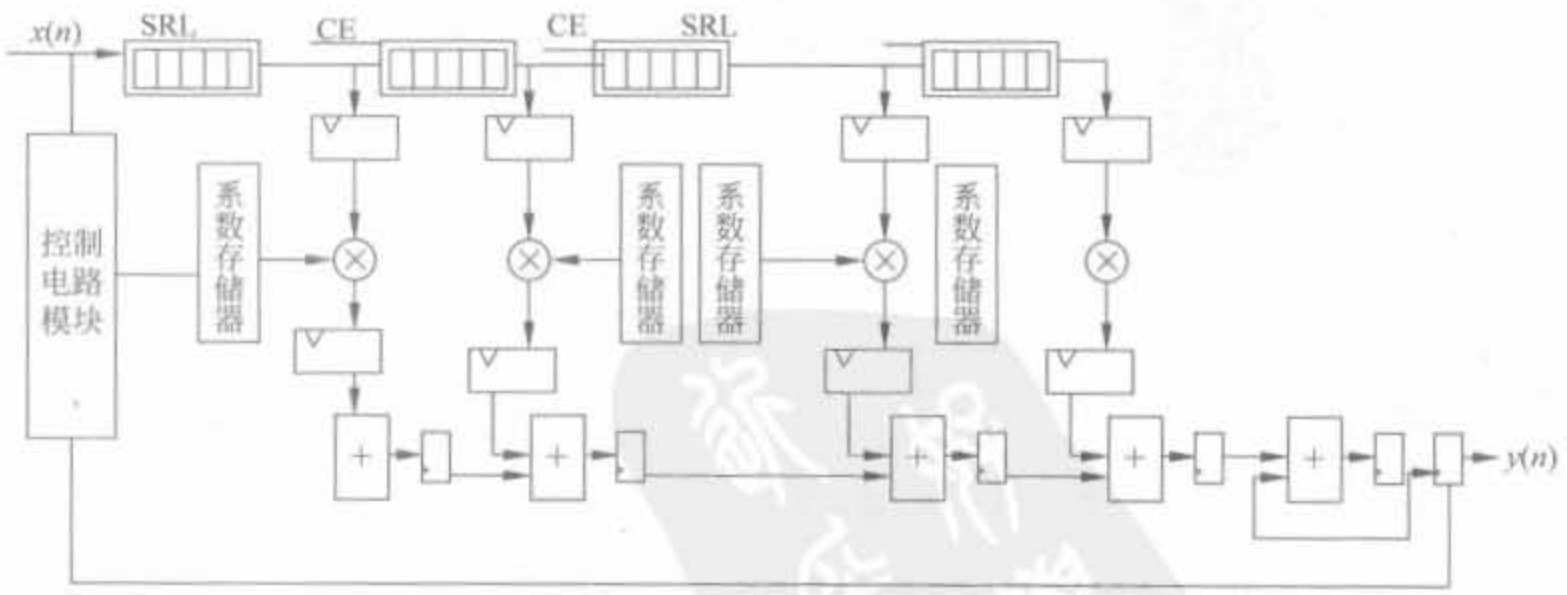


图 7-23 半并行 FIR 滤波器

在图 7-23 中，系数存储器存放 FIR 滤波器系数，SRL 存放采样数据，使得进入乘法器的采样数据与滤波器系数对应。同样，转置型 FIR 也可构成半并行 FIR 滤波器，控制电路模块与乘累加结构相结合也可构成半并行 FIR 滤波器。转置型结构有较小的输出流水延

迟，如图 7-21 所示，从第一个数据进入到第一个滤波输出，仅要 3 个流水时钟；脉动型与直接型都要多个流水时钟，脉动型要 $N-1$ 级流水，直接型要$[\log_2 N]+1$ 级流水。另一方面，转置型结构扇入系数较大，不利于更高阶扩展，而直接型与脉动在扇入系数方面不会对扩展带来影响。

7.5.3 FIR Compiler IP Core 的使用

在 ISE 9.1.03i 版中，提供了高性能 FIR 滤波器的 IP Core(有 Distributed Arithmetic FIR Filter、FIR Compiler、MAC FIR Filter 三类，其中 FIR Compiler 的功能最为强大，包含其余两个)供设计人员调用，可以完成多相抽取、多相插值、半带插值、半带抽取、希尔伯特变换和插值滤波器，具有乘加模式和分布式模式两种结构。本节主要介绍 FIR Compiler，而其他两个的使用方法与此类似。FIR Compiler 3.0 所支持的抽头数为 2～1024，位宽 1bit～32bit，并且支持多通道，最多可以同时支持 256 个通路，能够自动发掘系数的对称性来节省资源。其用户界面如图 7-24 所示。

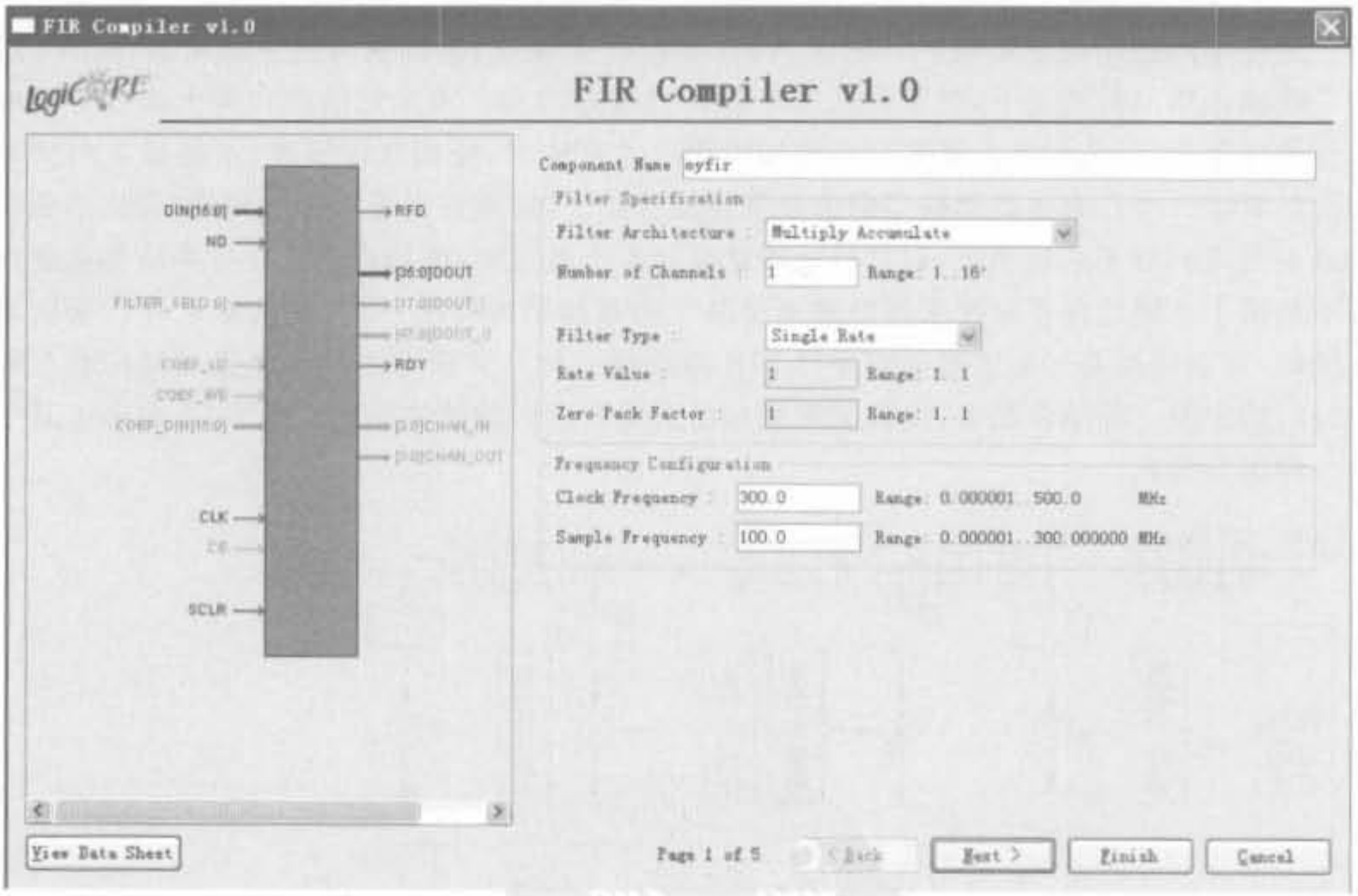


图 7-24 FIR 滤波器 IP Core 的用户界面

FIR 滤波器的 IP Core 具有丰富的控制信号，其详细说明如下所述。

(1) SCLR：输入信号，同步复位信号，高电平有效。可以重置滤波器内部的状态机，但并不清空数据寄存器的内容，是 IP Core 的可选管脚。

(2) CLK：输入信号，模块的工作时钟。

(3) CE：输入信号，模块时钟使能信号，是 IP Core 的可选管脚。

(4) DIN：输入信号，滤波器的输入数据，通过时分复用的方式来提供多通道的数据输入。

(5) ND：输入信号，新数据指示信号，高电平有效。只有当 ND 信号为高电平时，输入数据 DIN 才会被送进 FIR 的计算内核；当 RFD 为低电平时，将忽略任何输入信号，ND 指示无效。

(6) FILT_SEL：输入信号，用于多通路滤波器的模式下片选滤波器。

(7) COEF_LD：输入信号，加载系数指示信号，表明开始更换一组新的滤波器系数。

(8) COEF_WE：输入信号，系数写有效信号。

(9) COEF_DIN：输入信号，系数输入通路。

(10) DOUT：输出信号，滤波器的输出，其位宽由滤波器的精度、抽头数和系数的位宽决定，在 IP Core 中总是被配置成全精度以避免溢出。

(11) RDY：输出信号，滤波器输出有效指示信号。

(12) CHAN_IN：输出信号，用于指示当前输入数据的通道标号。

(13) CHAN_OUT：输出信号，用于指示当前数据的通道标号。

(14) DOUT_I：输出信号，仅在选择希尔伯特变换时有效，输出数据的通相分量。

(15) DOUT_Q：输出信号，仅在选择希尔伯特变换时有效，输出数据的正交分量。

例 7-4 使用 IP Core 实例化一个抽头数为 16、位宽为 16 位的可重配置低通滤波器模块。

由于该 IP Core 的步骤较多，下面对其使用进行详细说明。首先新建 IP Core，选择“Digital Signal Processing”→“Filter”→“FIR Compiler”，然后单击“确定”按钮。选择滤波器结构为乘加结构，通道数为 1，类型为单速率，模块工作时钟为 122.88MHz，采样速率为 30.72MHz。此外，还可以将滤波器配置成分布式结构，甚至多速率的抽取和插值滤波器。

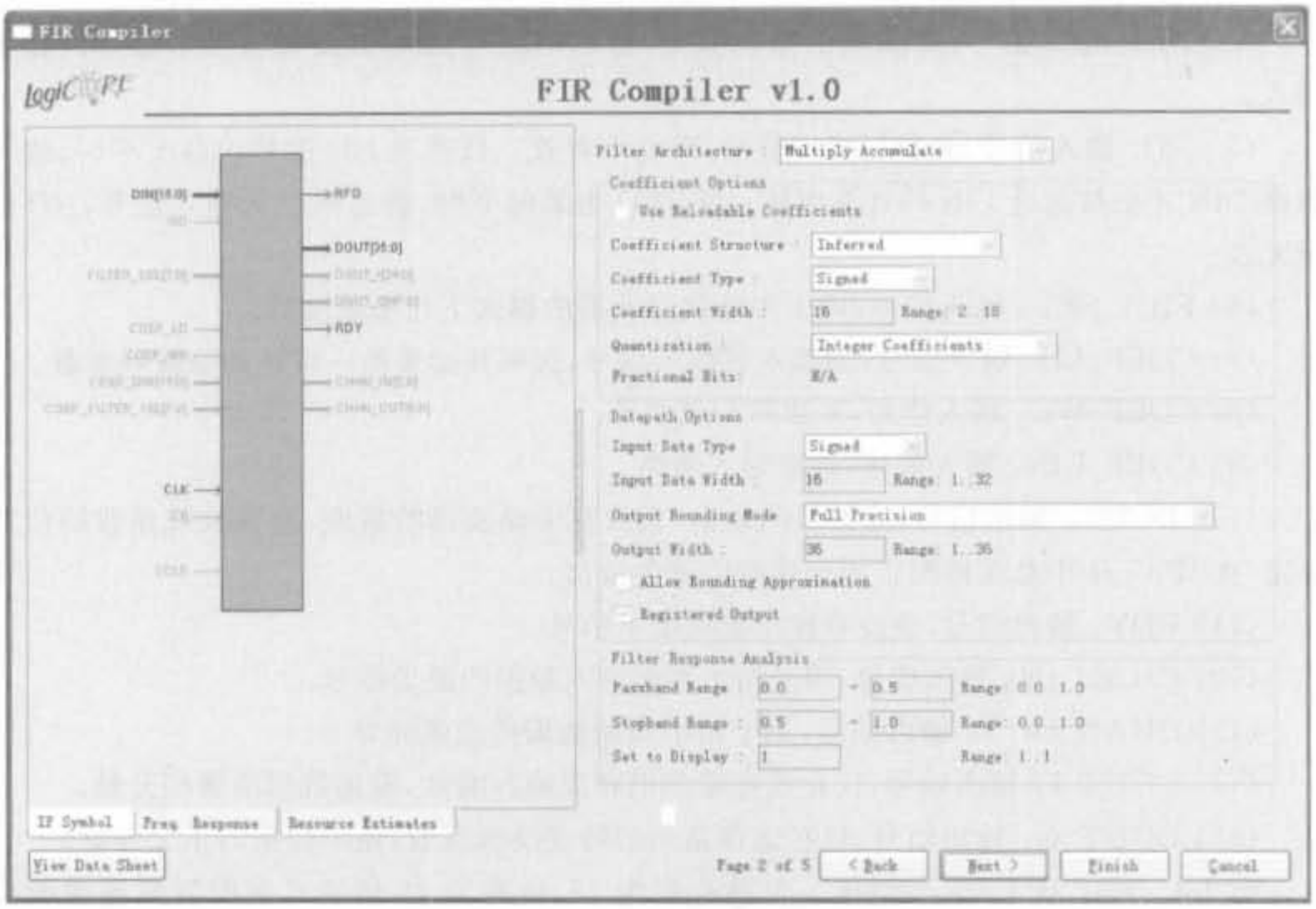
单击“Next”按钮，进入下一页，其界面如图 7-25(a)所示。选择输入数据为有符号数，位宽为 16bit，选中“Use Reloadable Coefficiencts”选项，这样，该滤波器才是可重配置的，且将系数选为有符号数、对称结构。然后单击“Browse”按钮，从计算机硬盘加载滤波器的初始系数配置文件(后缀仍为.coe)。

如果系数文件格式错误，图 7-26 中的 Coefficiencts files 后面的文本框为路径为红色的，只有添加了正确的配置文件，才会用正常的黑色显示。

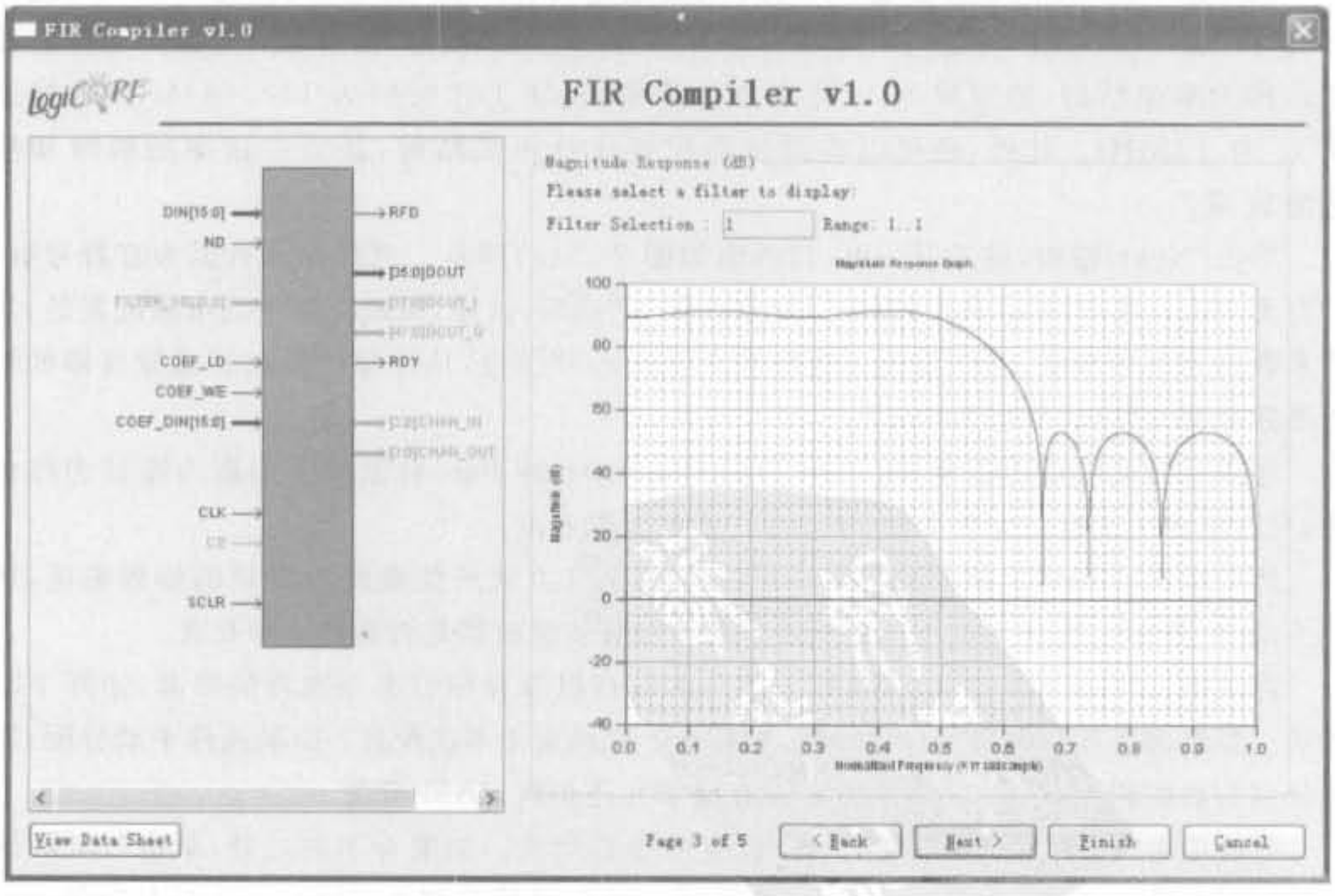
然后单击“Next”按钮，会显示出通过 COE 文件方式所加载的滤波器的幅频响应，如图 7-25(b)所示。设计人员可以此直接判断所加载的滤波器是否满足实际要求。

再次单击“Next”按钮，会出现握手信号的选择以及存储器类型选择的界面，如图 7-26 所示。如果选择存储类型为自动分配，则基本上就结束了本次配置；如果选择手动分配，需要分别为数据和系数指定是利用分布式存储单元还是块 RAM 资源。

最后单击“Next”按钮，将给出本次配置的参数列表。如果有不对之处，单击“Back”返回相应页面进行修改；如果确认无误，单击“Finish”，开始实例化 IP Core，生成相应的.xco 文件。



(a) FIR Compiler输入数据和系数存储类型选择界面



(b) FIR Compiler幅频特性显示界面

图 7-25　FIR Compiler 的幅频特性显示界面

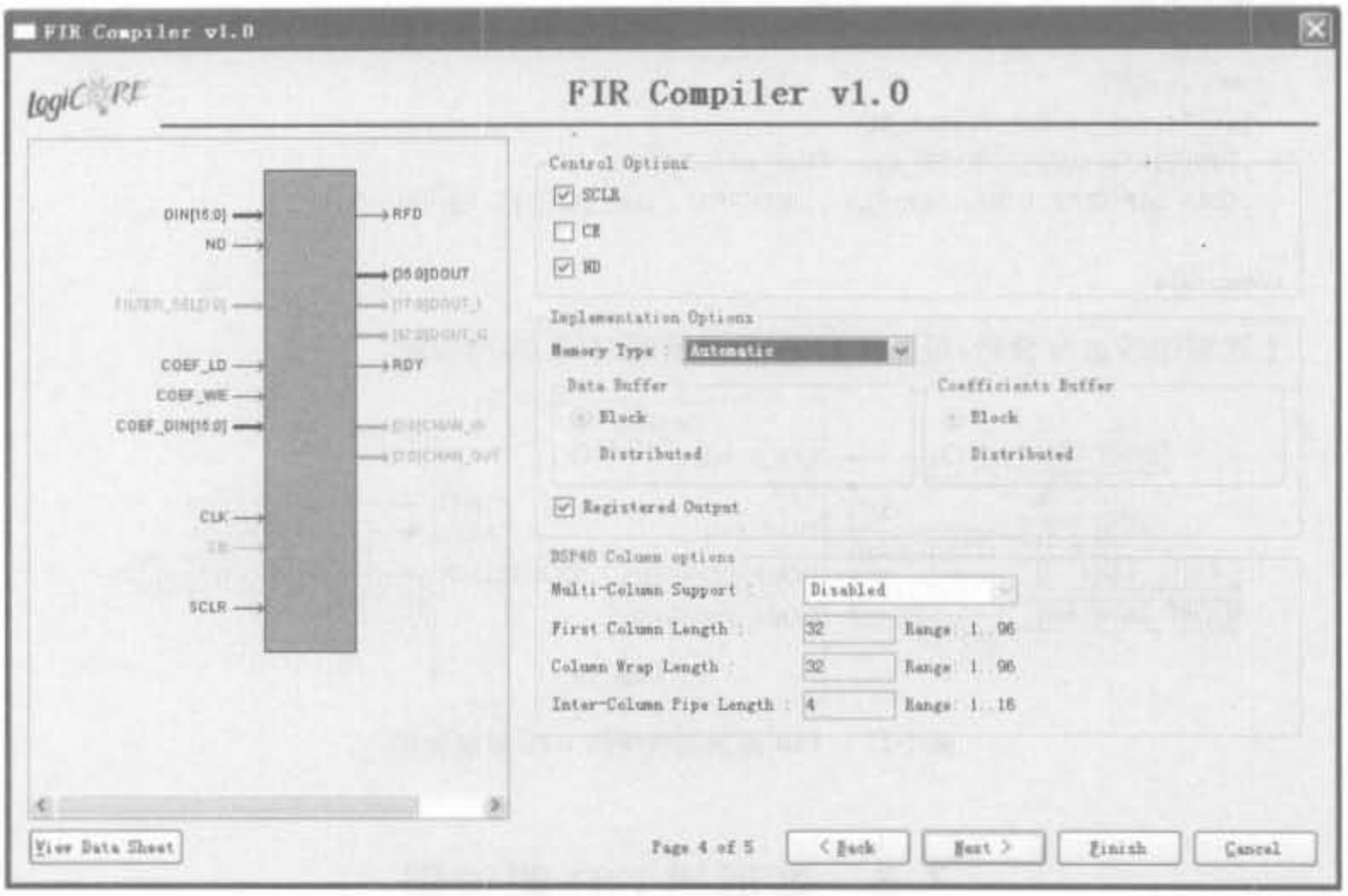


图 7-26 FIR Compiler 的存储器配置界面

经过例化 FIR Compiler,生成的 Verilog 接口为:

```
module fir_filter_16(DIN,COEF_LD,COEF_WE,COEF_DIN,CLK,RFD,DOUT,RDY);
  // synthesis black_box

  input [15:0]DIN;
  input [15:0]COEF_LD;
  input COEF_WE;
  input [15:0] COEF_DIN;
  input CLK;
  output RFD;
  output [35:0] DOUT;
  output RDY;
  ......

endmodule
```

在使用时,直接调用 fir_filter 模块即可。

```
module fir_filter(DIN,COEF_LD,COEF_WE,COEF_DIN,CLK,RFD,DOUT,RDY);

  input [15:0]DIN;
  input [15:0]COEF_LD;
  input COEF_WE;
  input [15:0] COEF_DIN;
  input CLK;
```

```
    output RFD;
    output [35:0] DOUT;
    output RDY;
    fir_filter_16 fir_filter_16(
    .DIN(DIN),.COEF_LD(COEF_LD),.COEF_WE(COEF_WE),
    .COEF_DIN(COEF_DIN),.CLK(CLK),.RFD(RFD),.DOUT(DOUT),.RDY(RDY));

endmodule
```

上述程序经过综合后，得到如图 7-27 所示的 RTL 级结构图。

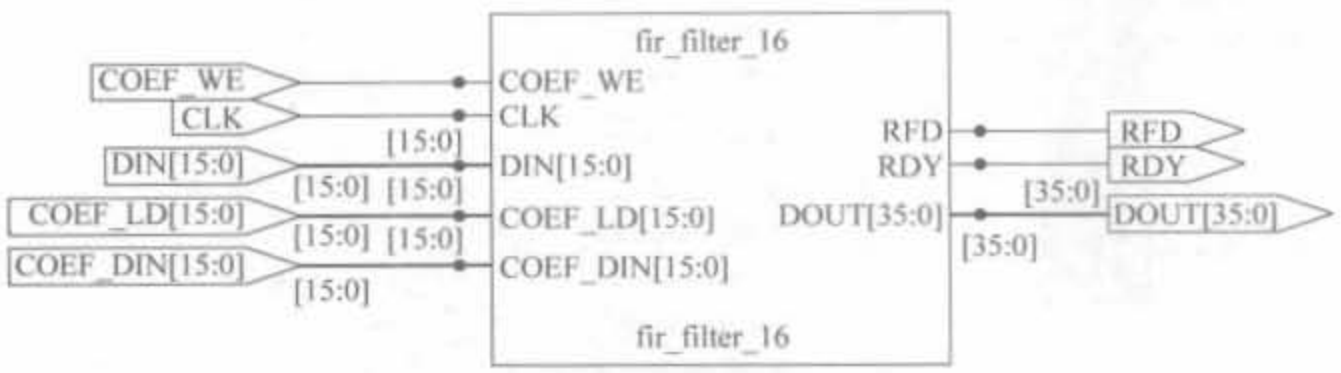


图 7-27　FIR 滤波器模块的 RTL 级结构图

7.6　多通道 FIR 滤波器

多通道滤波器被用来对多个输入数据流进行滤波，在通信、多媒体等领域被广泛使用。多通道的主要优势在于可以在输入数据流(通道)采样率较低的情况下，使用速度很快的运算单元。本节讲述了一种应用较广的 DSP——多通道 FIR 滤波器的实现。

7.6.1　滤波器组的基本概念

滤波器组是指一组滤波器，它们有着共同的输入，或有着共同的相加后的输出，如图 7-28 所示。

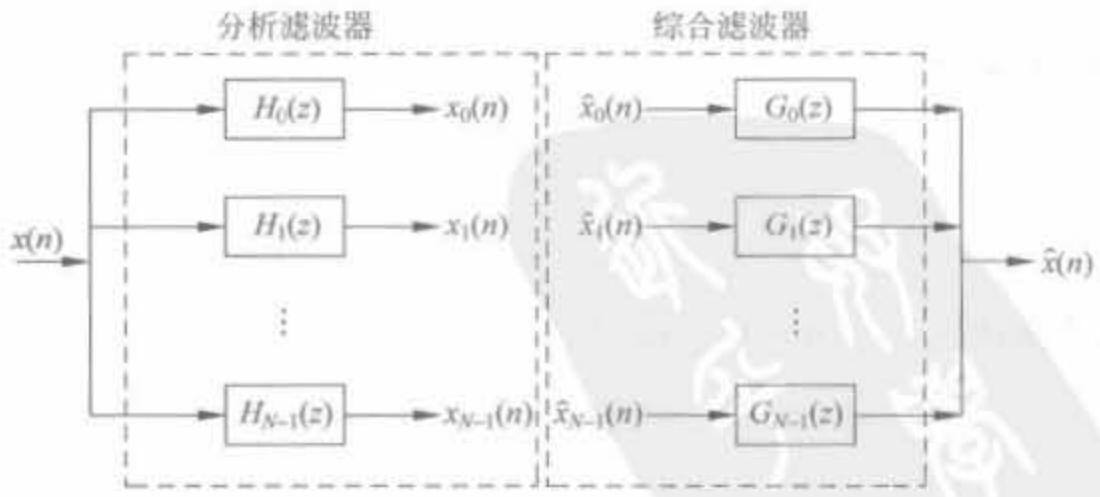


图 7-28　滤波器组示意图

信号 $x(n)$经过分析滤波器组后，变成一个个子带信号，最理想的情况就是各个子带信号没有重叠。但在实际中，是不可能实现这种情况的，各个子带信号 $x_i(n)$相互之间会有不

同程度的混叠。接着，综合滤波器组将 N 个子带信号综合为单一信号。设计滤波器组最重要的就是综合应用 $H_i(z)$和 $G_i(z)$来抵消混叠失真。

将 $x(n)$分成 N 个子带信号后，每个子带信号的带宽是原来的 $1/N$，因此，信号的速率(采样率)可以降低 N 倍。这样在实际应用中，为了其后系统处理方便，需要在分析滤波器组后面加上一个 N 倍抽取器。而在综合滤波器组后，希望重建后的信号$\hat{x}(n)$等于原信号 $x(n)$，那么应当使二者的速率相等，因此，在综合滤波器组之前还需要一个 N 倍插值器。完整的 N 通道滤波器组如图 7-29 所示。所以，分析滤波器组一方面将原信号分成子带信号，另一方面是作为抽取前的抗混叠滤波器；综合滤波器一方面重建原始信号，另一方面消除插值后的镜像。

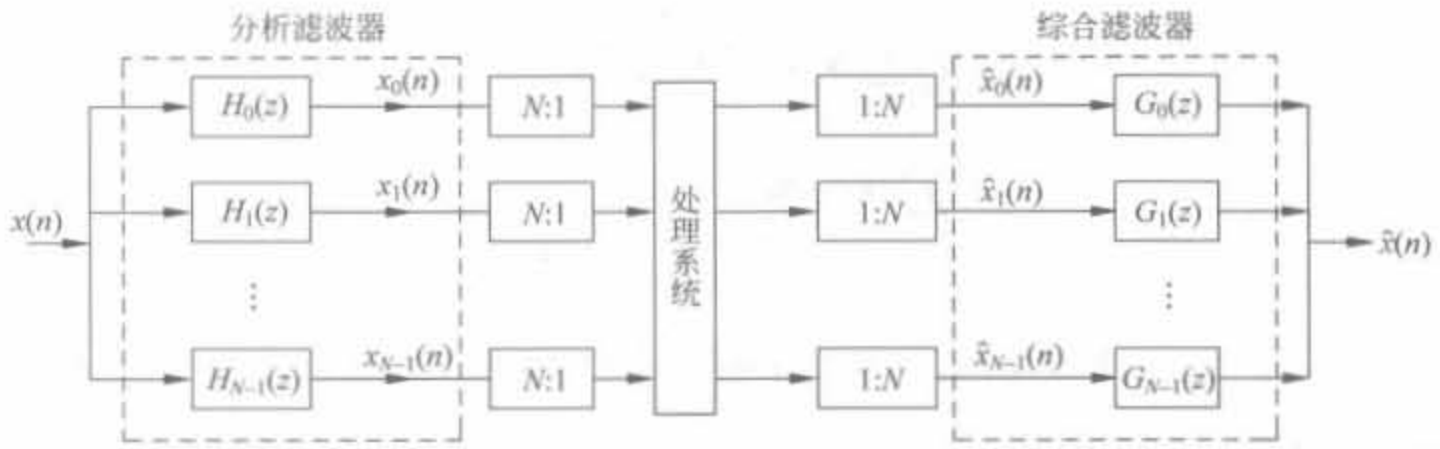


图 7-29　N 通道滤波器组系统

从图 7-29 中可以看出，我们关心的处理系统所处理的信号速率仅为原信号的 $1/N$，能大大减小系统实现的复杂度，并降低硬件的代价。从本质上讲，滤波器组是 FPGA 设计中串/并变换思想的完美体现。

区别不同滤波器组的一个重要特征就是通道数、带宽和各个滤波器中心频率之间的间隔。从通道数上可以将滤波器组分为双通道滤波器组和多通道滤波器组；根据带宽可以分为均匀带宽滤波器组和非均匀带宽滤波器组。从实现的角度讲，人们更倾向于均匀带宽滤波器组，因为其信号速率是一样的，便于处理，而且可以借助于 FFT 算法来实现。

7.6.2　多通道 FIR 滤波器的基本原理

双通道滤波器组是最基本的滤波器组，如图 7-30 所示，输入信号 $x(n)$经过一个包含有滤波器 $H_0(z)$和 $H_1(z)$的双路分析滤波器组，它们分别具有典型的低通和高通频率响应。此时经过滤波器组的两个信号，其带宽都近似为原来的 1/2。这样就可以进行 2 倍抽取，得到采样率为输入信号一半的 $x_0(n)$和 $x_1(n)$。但由于滤波器的过渡带不为零，因此抽取后存在着部分混叠。

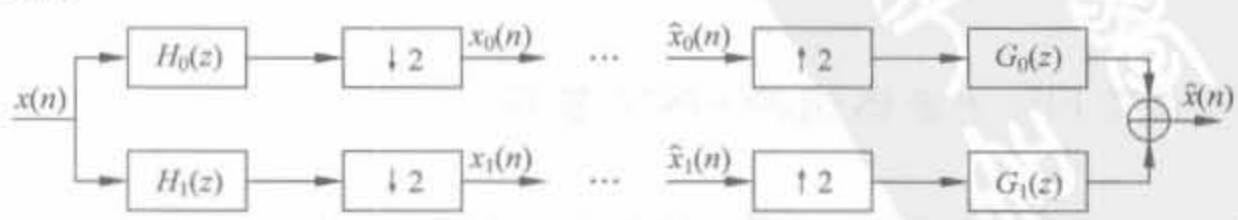


图 7-30　双通道滤波器组

对于 M 通道滤波器组，如果 M 是 2 的幂，那么可以用两通道滤波器组的级联实现 M 通道滤波器组。具体方法就是将各通道的输出作为下一个滤波器组的输入。通常把这种滤波器组称为树形结构 QMFB 的滤波器组。这种实现方式的缺点是：计算量较大，存储量大，延时较大，同时通道数只能是 2 的幂。它的优点是：可根据两通道滤波器组的特性推断 M 通道滤波器组的特性(如是否能够完全重建)。

树形结构的 QMFB 可以等效为平行结构的 QMFB。信号可以用其低频成分作粗略的近似，而以其高频部分作细节的补充。树形结构 QMFB 恰好在这一点上具有很好的特性。

平行结构 M 通道 QMFB 在输入端将原 $x(n)$ 分成 M 个子带信号，在输出端将这些 M 个子带信号综合，总的输入/输出关系为

$$\begin{aligned}\hat{X}(z_1) &= \frac{1}{M}\sum_{\ell=0}^{M-1} X(z_1W^{\ell})\sum_{k=0}^{M-1} H_k(z_1W^{\ell})G_k(z_1)\\ &= \sum_{\ell=0}^{M-1} X(z_1W^{\ell})A_{\ell}(z_1)\end{aligned} \tag{7-6}$$

式中

$$W = \mathrm{e}^{-\mathrm{j}\frac{2\pi}{M}},\quad A_{\ell}(z_1) \triangleq \frac{1}{M}\sum_{k=0}^{M-1} H_k(z_1W^{\ell})G_k(z_1) \tag{7-7}$$

用矩阵形式表示 $A_{\ell}(z_1)$ 为

$$\frac{1}{M}\begin{bmatrix} H_0(z_1) & H_1(z_1) & \cdots & H_{M-1}(z_1)\\ H_0(z_1W) & H_1(z_1W) & \cdots & H_{M-1}(z_1W)\\ \vdots & \vdots & \ddots & \vdots\\ H_0(z_1W^{M-1}) & H_1(z_1W^{M-1}) & \cdots & H_{M-1}(z_1W^{M-1})\end{bmatrix}\begin{bmatrix} G_0(z_1)\\ G_1(z_1)\\ \vdots\\ G_{M-1}(z_1)\end{bmatrix} \triangleq \begin{bmatrix} A_0(z_1)\\ A_1(z_1)\\ \vdots\\ A_{M-1}(z_1)\end{bmatrix}$$

式中，$M\times M$ 矩阵 $\boldsymbol{H}(z_1)=[H_k(z_1W^{\ell})]$ 称为混叠分量矩阵，简称 AC 矩阵。

若在输出 $\hat{X}(z_1)$ 中除去混叠影响，式(7-6)写成两部分，即

$$\hat{X}(z_1) = X(z_1)A_0(z_1) + \sum_{\ell=1}^{M-1} X(z_1W^{\ell})A_{\ell}(z_1) \tag{7-8}$$

使其中 $\ell\neq 0$ 的部分为 0，即 $A_{\ell}(z_{\ell})=0,\ell\neq 0$，这样输出信号就不含混叠成分，式(7-8)变为：

$$\hat{X}(z_1) = X(z_1)A_0(z_1) = \frac{1}{M}\sum_{k=0}^{M-1} H_k(z_1)G_k(z_1)X(z_1) \tag{7-9}$$

式(7-8)中，$\sum_{\ell=1}^{M-1} X(z_1W^{\ell})A_{\ell}(z_1)$ 引起的失真称为混叠失真。为消除混叠失真，设计 $G_k(z_1)$ 配合 $H_k(zW^{\ell})$，使 $A_{\ell}(z_1)=0,\ell\neq 0$。此时，$\hat{X}(z_1)$ 和 $X(z_1)$ 之间的差别都是由 $A_0(z_1)$ 引起的，所以称 $A_0(z_1)$ 为失真传递函数。

7.6.3 多通道 FIR 滤波器组的 FPGA 实现

以双通道为例，正交滤波器组满足共轭镜像滤波器的条件，即 $H(z)=z^{-N}G(-z^{-1})$。其分析和综合滤波器组都有两个实现方法：第一就是采用转置 FIR 滤波器结构，需要的

乘法器个数为阶数 N 的一半，但这是通过多相分解获得的好处，其处理速率仍然保持不变；另外一种就是采用格形滤波器来实现，图 7-31 给出了 N 阶分析滤波器组的网格结构图。

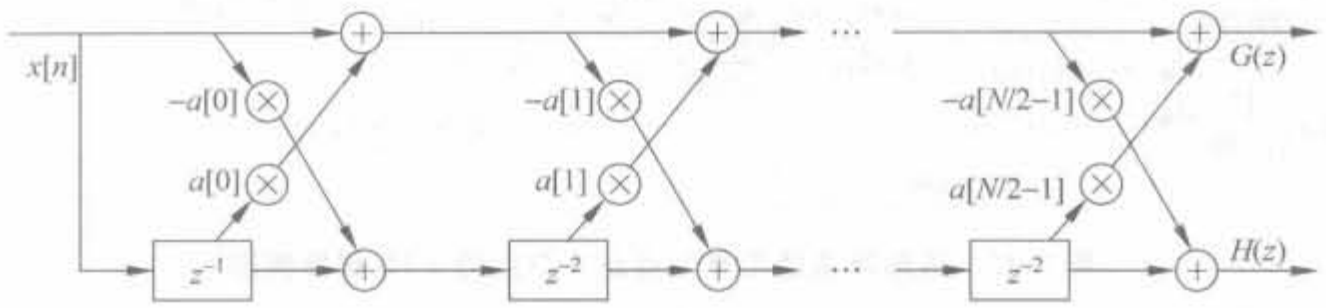


图 7-31　正交双通道分析滤波器组的格形结构图

例 7-5　使用 Verilog 实现一个 2 阶的格形双通道正交分析滤波器组。假设数据位宽为 16bit，分析滤波器的系数为

$$a[0] = 0.1568, \quad a[1] = 0.8764$$

1）将系数量化为 16bit，得到

$$a[0] = 5138, \quad a[1] = 28\,718$$

2）用 Verilog 实现所要求的正交分析滤波器组。程序分为顶层模块 filter_bank.v 和基本的格形单元 trellis_unit.v。

（1）顶层模块 filter_bank.v

```
module filter_bank(clk_32MHz,reset,x_in,y_out1,y_out2);
    input clk_32MHz;
    input reset;
    input [15:0] x_in;
    output [15:0] y_out1;
    output [15:0] y_out2;

    wire [15:0] coe1 = 5138; //量化后的滤波器系数
    wire [15:0] coe2 = 28718;

    //调用格形滤波单元的子模块
    trellis_unit trellis_unit1(
        .clk(clk),.reset(reset),.coe(coe1),
        .x_in1(x_in),.x_in2(x_in),.y_out1(y1),.y_out2(y2));

    trellis_unit trellis_unit2(
        .clk(clk),.reset(reset),.coe(coe1),
        .x_in1(y1),.x_in2(y2),.y_out1(y_out1),.y_out2(y_out2));

endmodule
```

上述程序经过 Synplify Pro 综合后，得到如图 7-32 所示的 RTL 级结构。可以看到，格形结构非常便于硬件设计，只需要设计一级格形单元就可以搭建出整个模块。

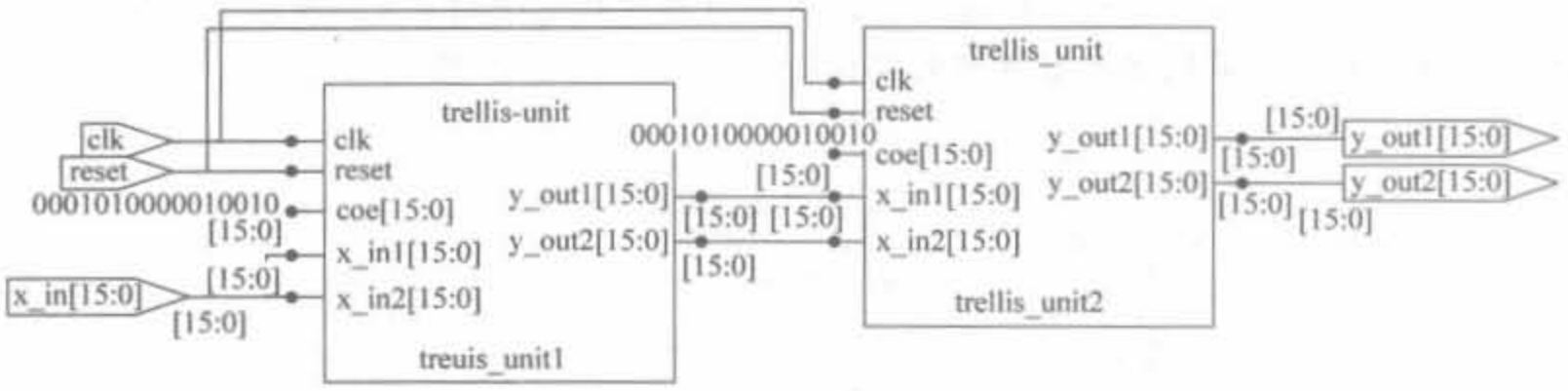


图 7-32　格形双通道正交分析滤波器组的 RTL 级结构图

(2) 格形单元 trellis_unit. v

```
module trellis_unit(clk,reset,coe,x_in1,x_in2,y_out1,y_out2);
    input clk;
    input reset;
    input [15:0] coe;
    input [15:0] x_in1;
    input [15:0] x_in2;
    output [15:0] y_out1,y_out2;

    reg [15:0] x_t2;
    always @(posedge clk) begin
        if(!reset)
            x_t2<=0;
        else //作一级寄存
            x_t2[15:0] <=  x_in2;
    end

    //完成格形运算
    assign y_out1 = x_t2[15:0] - x_in1[15:0] * coe[15:0];
    assign y_out2 = x_in1[15:0] + x_t2[15:0] * coe[15:0];

endmodule
```

上述程序经过 Synplify Pro 综合后，得到如图 7-33 所示的 RTL 级结构。

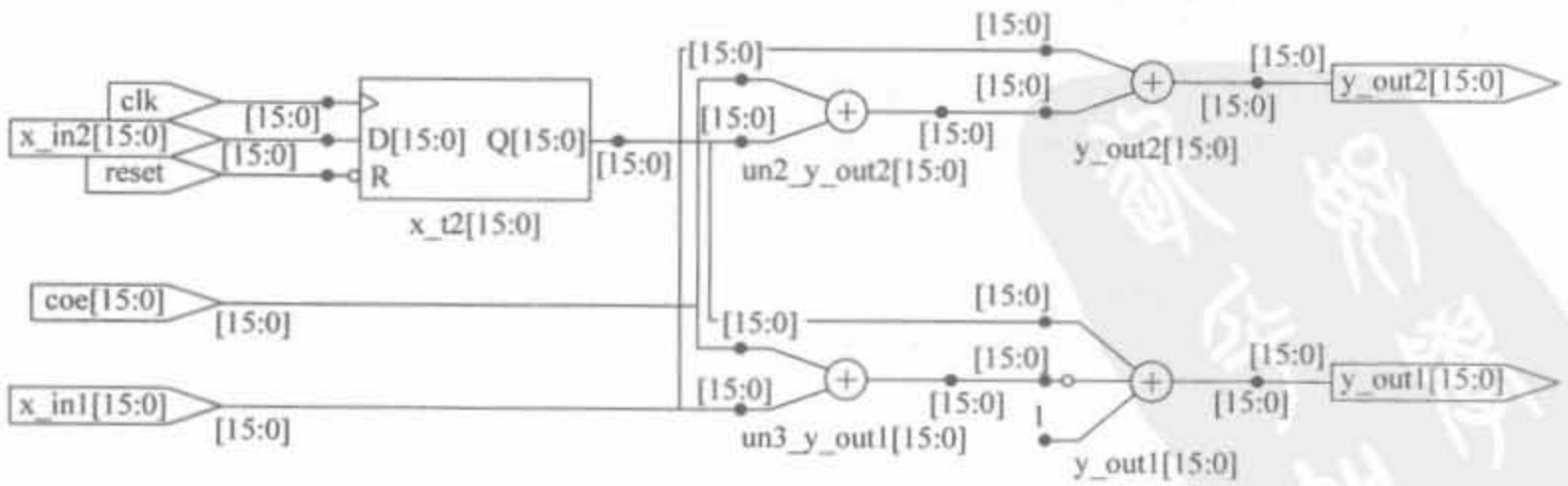


图 7-33　格形滤波单元的 RTL 级结构图

7.7 本章小结

本章首先介绍了数字信号和数字信号处理的基本理论。其次，介绍了快速傅里叶变换的原理和 Xilinx 公司的 IP Core 以及 Xtreme DSP 功能模块。接着，重点介绍了几类常用的滤波器：累加结构的 FIR 滤波器和(半)并行 FIR 滤波器，分析了它们的实现结构和用途，给出了相应的硬件代码。最后，介绍了滤波器组的基本概念，给出了相应的 Verilog 实现。希望读者通过本章的学习能够初步掌握数字信号处理这一广阔领域的基础知识和应用原则。

第8章

基于System Generator的DSP系统开发技术

CHAPTER 8

System Generator 是 Xilinx 公司的系统级建模工具，它在很多方面扩展了 MathWorks 公司的 Simulink 平台，提供了适合硬件设计的数字信号处理(DSP)建模环境，加速、简化了 FPGA 的 DSP 系统级硬件设计。System Generator 提供了系统级设计能力，允许在相同的环境内进行软、硬件仿真、执行和验证，并不需要书写 HDL 代码。此外，System Generator 工具还能完成高级提取，自动编译生成 FPGA 代码，也可通过低级的提取，对 FPGA 的底层资源进行访问，从而实现高效率 FPGA 设计构建。目前，基于 System Generator 的设计方法已在复杂系统实现中展现了强大的潜能，它必将成为未来流行的 FPGA 开发技术之一。本章重点讲述基于 System Generator 的 FPGA 开发技术，介绍了相关的基础知识和部分高级应用技巧，并给出了无线通信系统中变频器的工程实现。

8.1 System Generator 的简介与安装

8.1.1 System Generator 简介

目前的 FPGA 芯片不再扮演胶合逻辑的角色，而成为数字信号处理系统的核心器件。在芯片内，不仅包含了逻辑资源，还有多路复用器、存储器、硬核乘加单元以及内嵌的处理器等设备，并且具备高度并行计算的能力，使得 FPGA 已成为高性能数字信号处理的理想器件，特别适合于完成数字滤波、快速傅里叶变换等。但遗憾的是，FPGA 并未在数字信号处理领域获得广泛应用，主要原因就是：首先，大部分 DSP 设计者通常对 C 语言或 MATLAB 工具很熟悉，却不了解硬件描述语言 VHDL 和 Verilog HDL；其次，部分 DSP 工程师认为对 HDL 语言在语句可综合方面的要求限制了其编写算法的思路。基于此，Xilinx 公司推出了简化 FPGA 数字处理系统的集成开发工具 System Generator for DSP，快速、简易地将 DSP 系统的抽象算法转化成可综合的、可靠的硬件系统，为 DSP 设计者扫清了编程的障碍。

System Generator for DSP 是业内领先的高级系统级 FPGA 开发工具，借助 FPGA 来设计高性能 DSP 系统，其强大的提取功能可利用最先进的 FPGA 芯片来开发高度并行的系统，并和 Simulink(MathWorks 公司产品)实现无缝链接，快速建模并自动生成代码。此外，System Generator 是 Xilinx 公司 Xtreme DSP 解决方案的关键组成，它集成了先进的 FPGA 设计工具以及 IP 核，支持 Xilinx 公司全系列的 FPGA 芯片，提供从初始算法验证到

硬件设计的通道。System Generator 最大的特点就是可利用 Simulink 建模和仿真环境来实现 FPGA 设计，而无需了解和使用 RTL 级硬件语言，让 DSP 设计者能够发挥基于 FPGA 的 DSP 的最大性能和灵活性，并缩短整个设计的周期。

典型的 System Generator 工程设计实例如图 8-1 所示。可以将 System Generator 看成 MATLAB 软件中的一个硬件设计工具包，它提供了丰富的应用子模块，所有的硬件综合和实现信息都将被自动添加到芯片配置文件中，用户不必熟悉 FPGA 的设计流程以及 HDL 语言，只需要经过拖拽和连接将子模块搭成应用系统即可。

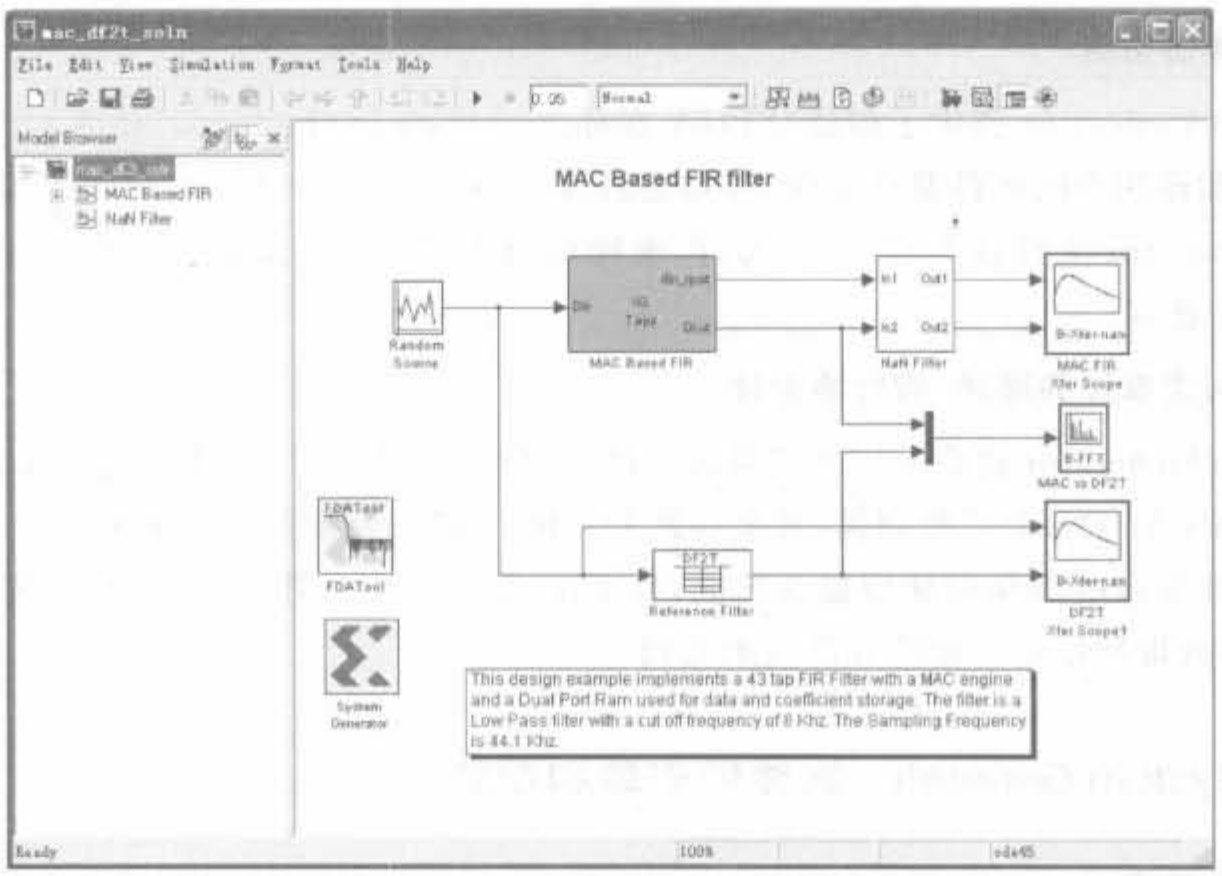


图 8-1　典型的 System Generator 工程设计实例示意图

8.1.2　System Generator 的主要特征

System Generator 是高性能 DSP 系统的快速建模和实现工具，是 DSP 高层系统和 Xilinx FPGA 之间的桥梁，其作用如图 8-2 所示。可在 MATLAB/Simulink 环境下对算法以及系统建模，并生成相应的工程，再调用 ISE 相应的组件进行仿真、综合、实现，并完成芯片的配置。整个开发的过程肯定是反复迭代、修正的，其中不可缺少的纽带就是 System Generator。

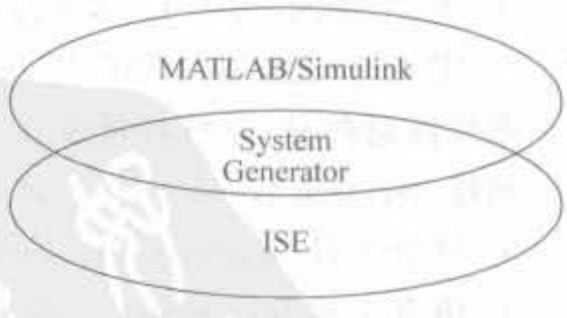


图 8-2　System Generator 的作用

System Generator 在 Simulink 中可当作一个用户程序包，自动将模型化的数字系统设计转换成硬件电路，其主要特征如下所述。

1. 丰富的 DSP 模块

System Generator 提供了包含信号处理（如 FIR 滤波器、FFT）、纠错（如 Viterbi 解码器、Reed-Solomon 编码器/解码器）、算法、存储器（如 FIFO、RAM、ROM）及数字逻辑功能

的 Xilinx 模块集，可快速、高效地在 Simulink 内构建和调试高性能 DSP 系统。此外，Xilinx 模块集提供的模块可以使用户导入.m 函数及 HDL 模块。

2. Simulink 设计的 VHDL 或 Verilog 的自动代码生成

设计人员无需掌握 HDL 语言即可开发数字处理系统，且所得到的设计具备 HDL 设计所有的优点。用户也可以通过基本的子模块生成功能特征明确的 System Generator IP Core，作为大型设计的一部分使用。同样，掌握了 RTL 设计的用户，可更好地利用 System Generator。

3. 硬件协仿真

System Generator 提供了创建“FPGA 在环路(FPGA-in-the-loop)”仿真对象的代码生成功能，可加速用户的硬件验证工作，并加速其在 Simulink 与 MATLAB 中的仿真。目前，System Generator 支持以太网(10/100/吉比特)、PCI、Cardbus 及硬件平台与 Simulink 之间的 JTAG 通信。

4. 嵌入式系统的硬件/软件协设计

System Generator 提供了嵌入式系统的硬件/软件协设计能力，可直接加载 Xilinx 公司的 MicroBlaze 32 位 RISC 处理器，甚至构建和调试 DSP 协处理器。System Generator 提供了硬件/软件接口的共享存储器提取功能，自动生成 DSP 协处理器、总线接口逻辑、软件驱动器以及协处理器使用方面的软件技术文档。

8.1.3 System Generator 软件的安装和配置

1. 软件需求

System Generator 软件版本必须和 ISE 版本一致，同时要有匹配的 MATLAB 版本才能正常工作。对于 System Generator 9.1，需要以下的软件环境：

- MATLAB 的版本为 MATLAB v7.3/Simulink v6.5(R2006b)或者 MATLAB v7.4/Simulink v6.6(R2007a)。注意，MATLAB 软件的安装路径上不能出现空格(例如，可以为 C:\MATLAB\R2007a)。
- ISE 的版本为 9.1.01i 或者更高，ISE Simulator 的版本为完全版。
- IP 核库的版本为 ISE IP 9.1i Update 1 或者更高。

需要注意的是，系统环境变量 $XILINX 必须设置为 ISE 的安装目录。

同样，System Generator 对常用的第三方软件也有相应的版本要求：

- 综合工具 Synplify Pro 的版本为 v8.6.2 或者 v8.8.0.4；
- 仿真工具 ModelSim 的版本至少为 PE 或者 SE v6.1f 以及更高版本。

2. System Generator 的安装

System Generator 的安装软件只能通过网站下载的方式得到，网址为 http://www.xilinx.com/ise/optional_prod/system_generator.htm。在安装 System Generator 之前，需要关闭所有的 ISE 以及 MATLAB 应用程序，然后双击安装软件的图标，即弹出如图 8-3 所示的欢迎界面。

图 8-3 System Generator 的安装欢迎界面

单击“Next”按钮进入软件安装协议说明界面，选中“I Agree”选项进入 MATLAB 版本选择界面，安装程序会自动列出已安装的 MATLAB 代码，如图 8-4 所示（由于只安装了 MATLAB 2006b 版本，所以只有一个版本）。

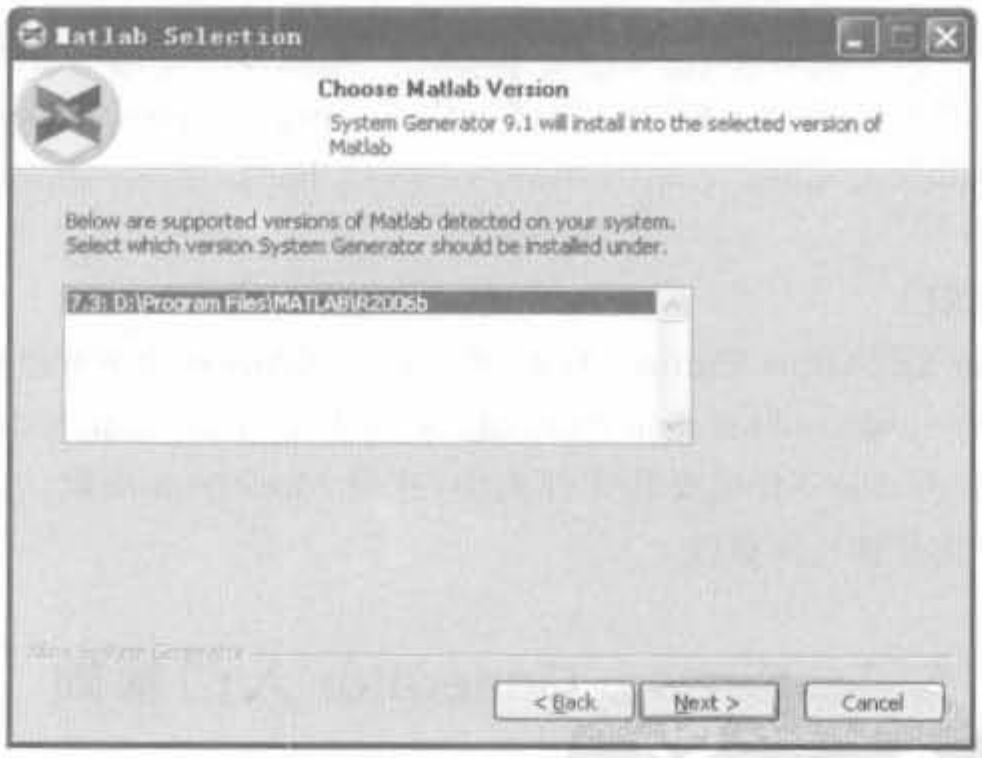


图 8-4 System Generator 安装时的 MATLAB 版本选择界面

继续单击“Next”按钮，进入安装路径选择界面，同时安装程序会给出所需的磁盘空间大小。对于 9.1 版，需要 118MB 的空间。再单击“Next”按钮进入安装进程界面，安装完成后的界面如图 8-5 所示，单击“Close”按钮完成安装。

3. Xilinx HDL 库的编译和配置

如果要在 ModelSim 中完成 System Generator 设计的仿真，需要编译所有的 IP 库模块。由于 ModelSim 存在 PE、SE 以及 XE 等不同的版本，下面分别介绍其编译方法。

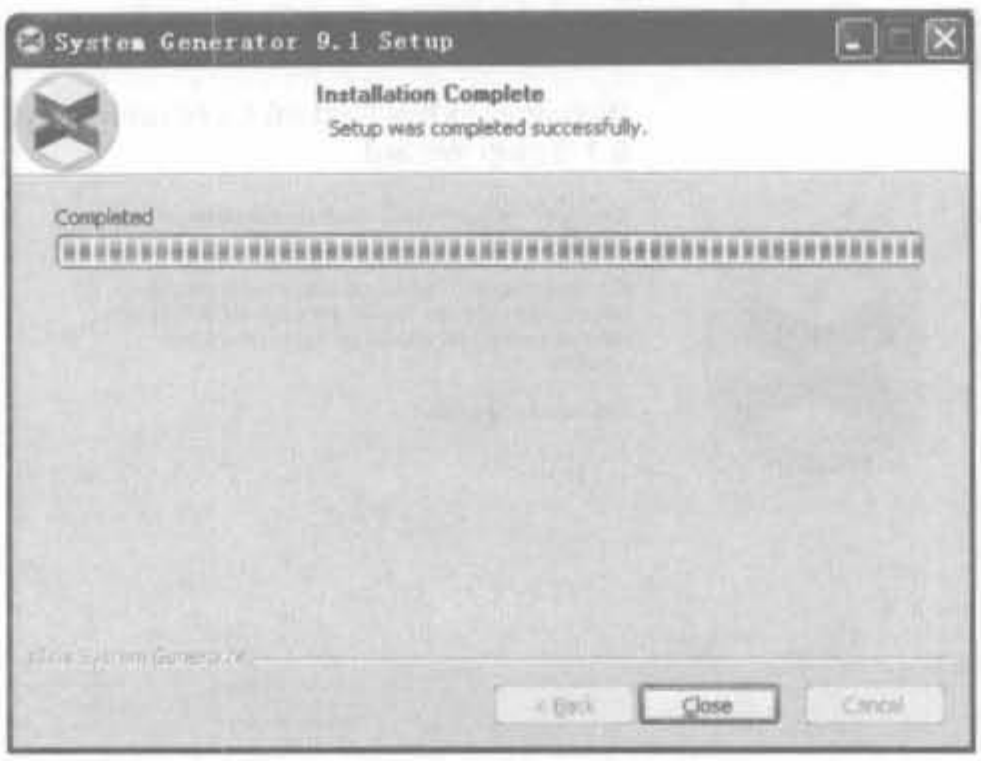


图 8-5 System Generator 安装完成提示界面

1) ModelSim(PE 或 EE/SE)

Xilinx 提供了名为 compxlib 的库编译工具，在 DOS 模式下完成库的编译。例如，命令

```
compxlib -s mti_se -f all -l all
```

可编译生成 ModelSim SE 版本中 VHDL 以及 Verilog HDL 两种语言的库。完成的 compxlib 编译指令可查阅 Xilinx 软件文档《Synthesis and Simulation Design Guide》，也可以直接在 http://toolbox.xilinx.com/docsan/xilinx82/books/docs/sim/sim.pdf 的网址中在线查阅。

2) ModelSim(XE)

对于 ModelSim XE(Xilinx Edition)版本，可直接从 Xilinx 网站下载已经过编译的库压缩包，其网址为：http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=mxe_libs。然后将其解压缩到 ModelSim XE 的安装文件夹中，这是 ModelSim 的默认寻找路径，因此不用对 modelsim.ini 文件作任何修改。

8.2 System Generator 入门基础

System Generator 提供了专门用于算法、系统和硬件开发的 DSP 开发流程和丰富的 IP Core 应用模块，给硬件工程师和软件工程师都带来一种新颖的设计模式，因此有必要对基础知识进行简单介绍，如开发流程、Simulink 以及综合工具 AccelDSP 的使用方法等。

8.2.1 System Generator 开发流程简介

本节介绍使用 System Generator 设计数字系统的常用步骤。在 Simulink 的可视化环境中，根据系统设计功能将 Xilinx 模块连接成所设计的系统，并定义合适的系统参数；而后

运用 System Generator 将 Simulink 模型转换成硬件可执行模型，将系统定义的参数对应至硬件实现的实体以及输入/输出端口，并自动完成综合、仿真与实现。整个开发流程分为浮点算法开发、定点算法实现、硬件系统设计以及代码优化 4 个步骤。

1. 浮点算法开发

利用 MATLAB 软件及其提供的工具包快速地完成浮点算法的开发、验证以及性能评估，借助于 Simulink 可快速完成原型设计和模型分析。

2. 定点算法实现

将 MATLAB 浮点算法通过 AccelDSP 在 Xilinx 器件上实现定点逻辑。AccelDSP 直接将浮点 MATLAB 算法的 M 文件自动生成可综合的 RTL 模型。AccelDSP 综合工具是基于高级 MATLAB 语言的工具，用于设计针对 Xilinx FPGA 的 DSP 块。该工具可自动地进行浮点/定点转换，生成可综合的 VHDL 或 Verilog HDL 设计，并创建用于验证的测试平台。并且，还能以报告的形式提供资源利用率、吞吐量和延迟等指标，从而根据实际工程需要来设置系统级要求，借助于 IP-Explorer 技术来实现面积和速度的折衷，快速地选择最佳的芯片设计。

3. 硬件系统设计与实现

定义使用 Xilinx IP 的详细硬件架构，采用 System Generator for DSP 划分协处理器和可编程器件之间的设计。System Generator 可满足 FPGA 流程中所有需要的功能要求，对于用户而言，通过单击按钮即可将模型设计转换成 HDL 语言。在此过程中会生成下列文件：

- 设计所对应的 HDL 程序代码；
- 时钟处理模块，包括系统时钟处理操作以及生成设计中所需的不同频率的时钟信号；
- 用于测试设计的 HDL 测试代码，可直接将其仿真结果和 Simulink 输出相比较；
- 工程文件以及综合、实现过程所产生的各种脚本文件。

4. 代码优化

利用 ISE RTL 设计环境生成优化的 FPGA 设计，属于高级应用，要求设计者不仅要熟悉算法的架构、瓶颈，还需要精通 RTL 设计。对于一般设计，如果系统硬件资源够用，再加上设计周期短，可忽略这一步。

在 Simulink 可视化设计环境中，重要的是在 Simulink 环境中实现定点算法，根据系统设计功能将 Xilinx 模块连接成设计系统，并定义合适的系统参数；而后利用 System Generator 将 Simulink 模型转换为可执行的硬件模型，将系统定义的参数对应到硬件实现的模块、输入/输出端口等属性；再借助于 ModelSim 软件验证相应的设计是否和 Simulink 输出一致，否则需要重新修改设计；最后将设计生成可对器件编程的比特流文件，将其下载到目标芯片中。因此，典型的开发流程如图 8-6 所示，其中 System Generator 会自动为 FPGA 的综合、HDL 仿真以及实现生成命令文件，用户只需完成 Simulink 设计以及比较最终的 RTL 输出结果。整个开发流程都是在可视化的环境中完成的。

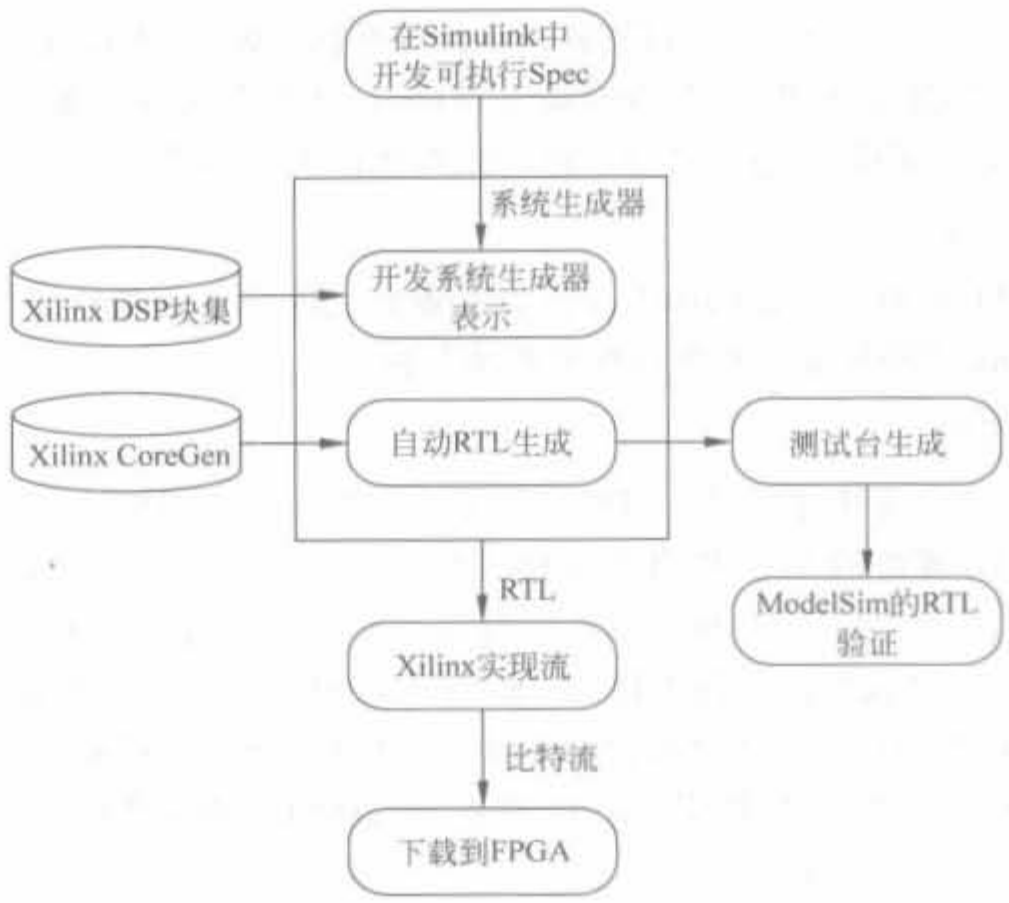


图 8-6　典型的 System Generator 设计流程

8.2.2　Simulink 基础

1. Simulink 简介

Simulink 是 MATLAB 中的一种可视化仿真工具，广泛用于线性系统、数字控制以及数字信号处理的建模和仿真中。Simulink 采用模块化的建模方式，每个模块都有自己的输入、输出端口，并能实现一定的功能。在 Simulink 中，模型表现为若干个仿真模块的集合以及各个模块之间的连接关系，并且这些模块可以组织成具有同等结构的子系统，具有内在的模块化设计要求。

根据输出信号和输入信号的关系，Simulink 提供了 3 种类型的模块：连续模块、离散模块和混合模块。连续模块指输出信号发生连续变化的模块；离散模块则是输出信号以固定间隔变化的模块；混合模块根据输入信号的类型来确定输出信号的类型，既能产生连续输出信号，也能够产生离散输出信号。如果一个仿真模型中只包括离散模块，Simulink 采用固定步长方式进行仿真；如果模型中只有连续模块，Simulink 将采用连续方式对模型进行仿真。如果模块中包含连续模块和离散模块，则采用两种仿真步长进行仿真。

2. Simulink 软件的安装

Simulink 可以随着 MATLAB 一起安装到计算机内，在 MATLAB 安装选项中选中 Simulink 组件，安装程序会自动将其安装到 MATLAB 目录下；如果已安装了 MATLAB 而没有安装 Simulink 的话，也可以继续使用安装程序将 Simulink 安装到计算机内。两者的安装方法是基本一样的。需要注意的是，Simulink 的运行需要 MATLAB 后台的支持，因此必须要安装 MATLAB 软件。

3. Simulink 的工作原理

Simulink 的工作包括两个阶段：初始化阶段和模型执行阶段。

1）模型初始化阶段

在初始化阶段，Simulink 主要完成以下工作：

（1）将模型参数传递给 MATLAB 进行估值，得到的数值结果将作为模型的实际参数。

（2）展开模型的各个层次，每一个非条件执行的子系统将被它所包含的模块代替。

（3）模型中的模块按更新的次序进行排序。

（4）决定模型中有无显示设定的信号属性，如名称、数据类型等，并且检查各个模块是否能连接到其相应的输入信号。

（5）决定所有未设定采样时间的模块的采样时间。

（6）分配和初始化用于存储每个模块的状态和输入当前值的存储空间。

完成上述工作后，就可以进入模型执行阶段。

2）模型执行

一般模型是使用数值积分来进行仿真的，所运用的仿真解法器依赖于模型提供的连续积分能力。计算微分可以分为两步来进行：首先，按照排序所确定的次序计算每个模块的输出；其次，根据当前时刻的输入和状态来决定状态的微分，得到微分向量后把它返回给解法器，用其来计算下一时刻采样点的状态向量。一旦新的状态向量计算完毕，被采样的数据源模块和接收模块才被更新。

在仿真开始时，模型设定待仿真系统的初始状态和输出。在每一个时间步长中，Simulink 计算系统的输入、状态和输出，并更新模型来反映计算出的值。在仿真结束时，模型得出系统的输入、状态和输出。

4. Simulink 设计简单举例

作为本节的最后一部分，给出一个 Simulink 的应用实例，增加读者对 Simulink 建模的直观印象。

例 8-1　建立一个调幅（AM）系统，信号频率为 100kHz，载波频率为 1MHz，调幅系数为 0.5，并在示波器中显示出来。

具体的操作步骤如下：

1）打开 MATLAB。在工具栏单击图标，启动 Simulink（也可以直接在 MATLAB 命令窗口直接键入“simulink”）。

2）在 Simulink 环境下新建一个模型。在 Simulink 工具栏单击图标实现，或通过菜单“File”中的“New”→“Model”命令来实现，将新模型保存为 am. mdl。

3）从“Simulink Library Browser”中加入基本模块。首先从“Simulink”→“Source”库中选中“Sine Wave”模块，直接将其拖拽到 am. mdl 的界面中，并复制该模块；然后双击“Sine Wave”模块，设定载波频率和幅度。按照同样的方法将“Sine Wave1”模块设定为信号产生模块；再从“Simulink”→“Math Operations”库中选择乘法器模块“Product”，最后从“Simulink”→“Sink”库中选择示波器“Scope”。

4）连接各个模块。选中目标模块，按住 Ctrl 键，再单击要连接的模块，Simulink 即可自动将两个模块连接起来，最终设计如图 8-7 所示。

5）在工具栏单击运行（RUN）图标 ▶，再直接单击“Scope”模块即可观察运行结果，如图 8-8 所示。至此，完成了一个简单的 AM 调制系统模型。

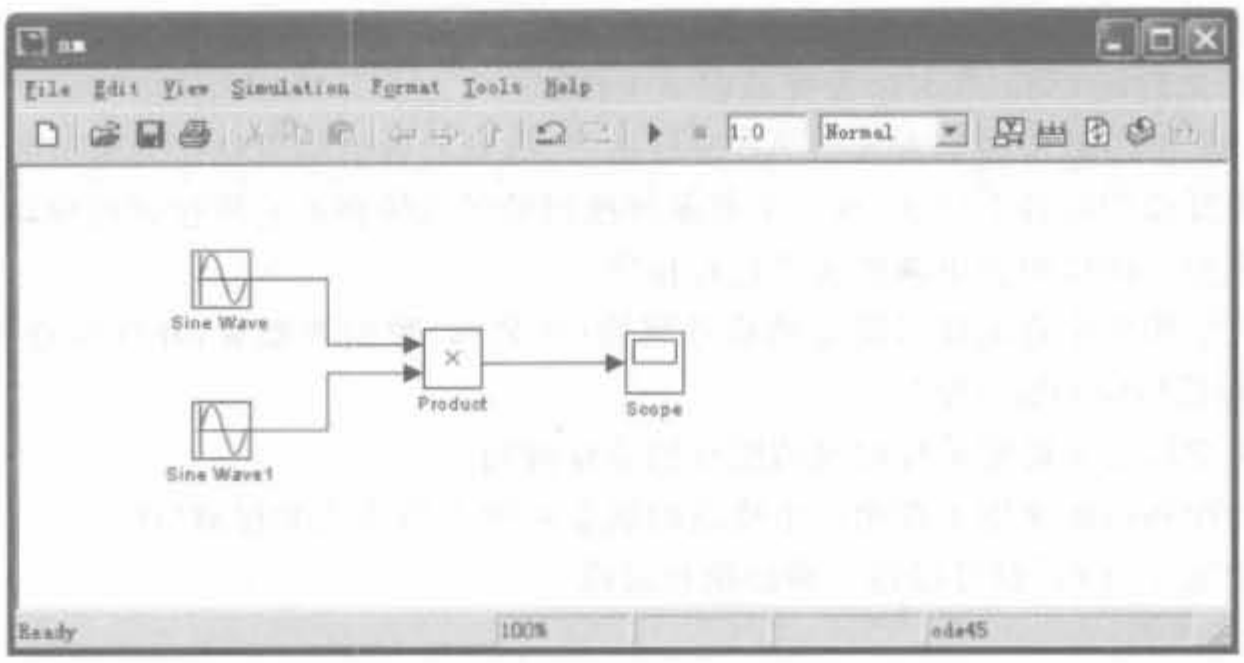


图 8-7　调幅系统的 Simulink 设计示意图

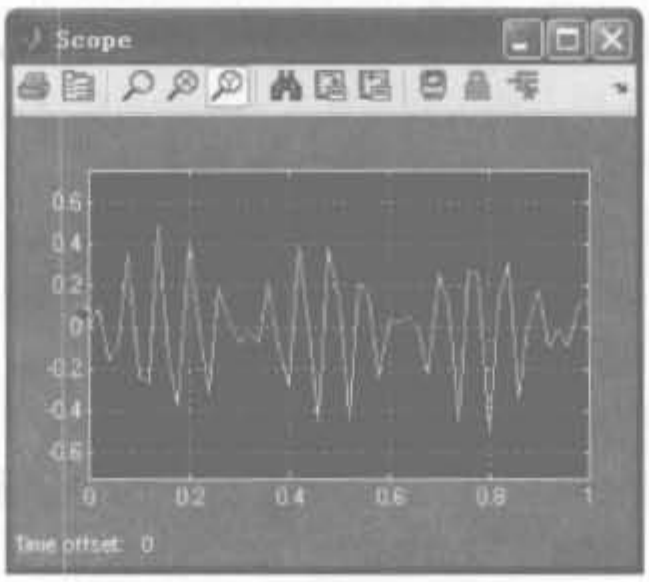


图 8-8　调幅系统的运行结果示意图

8.2.3　AccelDSP 软件工具

AccelDSP 是一款第三方综合软件，可将 MATLAB 浮点算法转换成为可综合 RTL 代码。Xilinx AccelDSP 是目前业界唯一能够将 MATLAB 浮点算法转换成为可综合 RTL 代码的开发工具。该工具可自动地进行浮点/定点转换，生成可综合的 VHDL 或 Verilog 代码，并创建用于验证的测试平台，同时生成定点 C++模型，或由 MATLAB 算法得到 System Generator 块。AccelDSP 综合工具是 Xilinx XtremeDSP 解决方案的重要组成部分。AccelDSP 产品体系由两个主要模块构成：AccelDSP 综合器和 AccelWare IP。

AccelDSP 综合器是一个综合和验证的环境，可以自动将 MATLAB 浮点代码转换成为定点代码，然后生成可综合的 VHDL 或 Verilog 代码，为设计者提供了验证算法和实现算法的功能。

AccelWare IP 与 AccelDSP 综合器联合实现滤波器、FFT 等 DSP 函数，通信算法函数以及高级数学运算函数。AccelWare 是一个 IP 库，包含一系列参数 DSP 模块，这些模块可以综合成为 RTL 代码(VHDL 或 Verilog)。每一个 IP 模块进行了预先验证，从而实现了

一旦生成即保证正确的算法开发流程。AccelWare IP 提供了 3 个专用工具箱(Toolkit)：信号处理工具包(包括 FIR 滤波器、CIC 抽取滤波器、CIC 内插滤波器、多相抽取滤波器、半带 FIR 滤波器、FFT 以及 IFFT 等模块)、通信工具包(包括直接数字合成器、BCH 编码器和解码器、卷积交织器和去交织器、卷积编码器、Reed-Solomon 编解码器、Viterbi 解码器、开方升余弦滤波器、加扰器、解扰器以及 ADC 采样—保持电路/正弦比较滤波器等模块)和高级数学运算工具包(包括 QR 分解法、Cholesky 分解法、QR 求逆、Cholesky 求逆、三角形矩阵求逆、特定排列旋转、多项式求值、奇异值分解以及 QRD-RLS 空间滤波器等模块)。

8.3　基于 System Generator 的 DSP 系统设计

System Generator 给用户提供了可视化的软、硬件一体化的设计环境，与传统的代码编写开发模式存在一定的差异。本节首先介绍如何利用 System Generator 的基本操作，以及在设计中应注意的问题，然后以实例给出如何实现简单的数字系统。

8.3.1　System Generator 快速入门

本节旨在给出 System Generator 设计的整体轮廓，让读者从宏观上把握，在后续章节的阅读中就不会一叶障目。

1. Xilinx Blockset 库的基本介绍

System Generator 和 Simulink 是无缝链接的，可以在 MATLAB 标准工具栏中直接启动，如图 8-9 所示。这些模块都根据其功能划分为不同的库。为了易于使用，又在某些库中添加了部分有广泛应用的模块，所有的模块都按字母顺序排列在 Xilinx Index 库中。读者需要注意的是，在 Simulink 环境中，只有通过 Xilinx 模块搭建的系统才能保证硬件可实现，其地位类似于 HDL 语言中的可综合语句。

图 8-9　Xilinx DSP 模块集

从设计流程中可以看出，熟悉 Xilinx DSP 基本模块库是设计流程中的关键环节，只有掌握了基本模块的特性和功能，才能更好地实现算法。由 Xilinx 模块库和 System Generator 一起，可生成 Xilinx 可编程器件的最优逻辑，这属于最低层的设计模块，地位等效于 IP Core，共有 90 多个。Xilinx 模块库的简要说明如表 8-1 所示。

表 8-1　Xilinx 模块库的简要说明

Xilinx 基本模块库名称	简要说明
Index	包含了所有的 Xilinx 模块
Basic Elements	包含了数字逻辑的标准组件模块
Communication	包含了数字通信系统中的通用模块
Control Logic	包含了用于创建控制逻辑以及状态机逻辑的模块
Data Types	包含了用于数据类型转换的模块
DSP	包含了用于数字信号处理的模块
Math	包含了用于完成数学运算的模块
Memory	包含了存储器操作模块
Shared Memory	包含了共享存储器操作模块
Tools	包含了使用函数模块，如代码生成、资源评估、协同仿真等

2. 基本单元模块

基本单元模块库中包含了数字逻辑的标准组件模块。使用这些模块可插入时间延迟，改变信号速率，引入常数、计数器以及多路复用器等。此外，库中还包含 3 个特殊的模块，即 System Generator 标志、黑盒子模块(Black Box)以及边界定义模块，后文将对其进行详细说明。该库中简要的模块说明如表 8-2 所示。

表 8-2　基本单元模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
System Generator	System Generator 标志，含有系统编译信息	Addressable Shift Register	长度可变的移位寄存器
Assert	声明模块，用于定义信号	Bit Basher	位操作模块，可提取或合并
Black Box	黑盒子模块，用于加载 HDL 模块	Clock Enable Probe	输入为 Simulink 设计的任意信号，输出为同频的布尔脉冲信号
Concat	将多个输入数据按位级联后作为一个输出数据	Constant	常数模块
Convert	数据格式转换模块，将输入信号按照要求转换成相应的格式	Counter	计数器模块
Delay	延迟模块	Down Sample	下采样模块
Expression	按照输入表达式的方式对输入信号按照位逻辑进行运算	Gateway In	Simulink 到 System Generator 的入口
Gateway Out	System Generator 到 Simulink 的入口	Invert	将输入数据按位取反
LFSR	线性反馈移位寄存器	Logic	可选择实现固定位数二进制数逻辑功能
Mux	多路选择器模块	Parallel to Serial	并/串转换模块
Register	寄存器模块	Reinterpret	改变输入数据的格式并输出

续表

基本模块名称	功能说明	基本模块名称	功能说明
Relational	比较器模块	Serial to Parallel	串/并转换模块
Slice	Slice 模块	Time Division Demultiplexer	时分解复用模块
Time Division Multiplexer	时分复用模块	Up Sample	上采样模块

3. 通信模块

通信应用是 FPGA 的主要应用领域之一，因此 Xilinx 的通信模块库提供了用于实现数字通信的各种函数，包括卷积编解码、RS 编解码以及交织器等模块。该库中简要的模块说明如表 8-3 所示。

表 8-3 通信模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
Convolutional Encode	卷积码编码模块	Depuncture	可在输入数据的特定位置插入要求的数据
Interleaver Deinterleaver	交织、解交织器	Puncture	从输入数据中移出指定的二进制向量数据
RS Decoder	RS 编码器	RS Encoder	RS 编码器
Viterbi Decoder	维特比译码器		

4. 控制逻辑模块

控制逻辑主要包括了用于创建各种控制逻辑和状态机的资源，包括逻辑表达式模块、软核控制器、复用器以及存储器，其简要说明如表 8-4 所示。

表 8-4 控制逻辑模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
Black Box	黑盒子模块	Constant	常数模块
Counter	计数器模块	Dual Port RAM	双口 RAM 模块
EDK Processor	EDK 处理器模块	Expression	表达式模块
FIFO	FIFO 模块	Inverter	将输入数据按位取反
Logical	可选择实现固定位数二进制数逻辑功能	MCode	MCode 模块，用于加载. m 函数
Mux	多路选择器模块	Picoblaze Microcontroller	Picoblaze 8 位处理器模块
Register	寄存器模块	Relational	比较器模块
ROM	ROM 模块	Shift	移位模块
Signal Port RAM	单口 RAM 模块	SLICE	Slice 模块

5. 数据类型模块

数据类型模块主要用于信号的数据类型转换，包括移位、量化、并/串、串/并转换以及精度调整模块，其简要说明如表 8-5 所示。

表 8-5 数据类型模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
Bitbasher	数据按位操作模块，可完成提取、并置以及扩充等功能	Concat	将多个输入数据按位级联后作为一个输出数据
Convert	数据格式转换模块，将输入信号按照要求转换成相应的格式	Gateway In	Simulink 到 System Generator 的入口
Gateway Out	System Generator 到 Simulink 的入口	Slice	Slice 模块
Reinterpret	改变输入数据的格式并输出	Scale	按照 2 的幂次方完成数据的放大和缩小
Serial to Paralleral	串/并转换模块	Shift	移位单元

6. DSP 模块

DSP 模块是 System Generator 的核心。该库包含了所有常用的 DSP 模块，其简要说明如表 8-6 所示。

表 8-6 DSP 模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
DAFIR	分布式 FIR 滤波器模块	DDS	数字频率合成器模块
DSP48	DSP48 硬核模块	DSP48 Macro	DSP48 宏模块
DSP48E	DSP48E 模块	FDATool	滤波器设计工具
FFT	FFT 模块	FIR Compiler	FIR 滤波器编译模块
LFSR	线性反馈移位寄存器模块	Opmode	DSP48 单元控制模块

7. 数学运算模块

数学运算是任何程序所不可避免的，Xilinx 提供了丰富的数学运算库，包括基本四则运算、三角运算以及矩阵运算等，其简要说明如表 8-7 所示。

表 8-7 数学运算模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
Accumulator	累加器模块	AddSub	加、减法模块
CMult	复数乘法器模块	Constant	常数模块
Convert	数据格式转换模块，将输入信号按照要求转换成相应的格式	Counter	计数器模块
Expression	表达式模块	Inverter	将输入数据按位取反
Logical	可选择实现固定位数二进制数逻辑功能	Mcode	用于加载. m 函数
Mult	乘法器模块	Negate	对输入数据取反模块
Reinterpret	改变输入数据的格式并输出	Relational	比较器模块
Scale	按照 2 的幂次方放大或缩小数据	Shift	移位操作
SineCosine	正、余弦模块	Threshold	门限处理模块

8. 存储器模块

该库包含了所有 Xilinx 存储器的 Logic Core，其简要说明如表 8-8 所示。

表 8-8 存储器模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
Addressable Shift Register	长度可变的移位寄存器	Delay	延迟模块
Dual Port RAM	双口 RAM 模块	FIFO	FIFO 模块
LFSR	线性反馈移位寄存器模块	Register	寄存器模块
ROM	ROM 模块	Shared Memory	共享存储器模块
Signal Port RAM	单口 RAM 模块		

9. 共享存储器模块

共享存储器模块主要用于共享存储器操作，其说明如表 8-9 所示，相关内容将在 8.4.3 节展开讨论。

表 8-9 共享存储器模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
From FIFO	从 FIFO 模块中读取数据	From Register	从寄存器中读取数据
Multiple Subsystem Generator	该模块可以使工作在不同时域的子系统很好地工作	Shared Memory	共享存储器模块
Shared Memory Read	共享存储器读模块	Shared Memory Write	共享存储器写模块
To FIFO	写数据到 FIFO 中	To Register	写数据到寄存器中

10. 工具模块

工具模块包含了 FPGA 设计流程中常用的 ModelSim、ChipScope、资源评估等模块，以及算法设计阶段的滤波器设计等模块。该库的模块在设计中起辅助作用，都是设计工具，一般不能生成 HDL 设计，其简要说明如表 8-10 所示。

表 8-10 工具模块的说明列表

基本模块名称	功能说明	基本模块名称	功能说明
System Generator	System Generator 标志	ChipScope	ChipScope 模块
Clock Probe	生成和系统时钟同频的、占空比为 50%的方波	Configurable Subsystem Manager	可配置子系统管理模块
Disregard Subsystem	专门置于子系统中，目前用得很少，未来可能会被取消	FDATool	滤波器设计工具
Indeterminate Probe	判断输入信号是否为确定逻辑	ModelSim	ModelSim 模块
Pause Simulation	暂停仿真模块	Picoblaze Microcontroller	Picoblaze 微处理器模块
Resource Estimate	资源估计模块	Sample Time	测量输入信号的采样周期
Simulation Multiplexer	允许设计中有两个功能相同的模块，一个仿真，一个综合	Signal-Step Simulation	单步仿真模块
Toolbar	快速进入工具栏模块	WaveScope	示波器模块

11. FPGA 边界定义模块

System Generator 是 FPGA 实现和算法开发之间桥梁，通过两个标准模块“Gateway In”和“Gateway Out”来定义 Simulink 仿真模型中 FPGA 的边界。“Gateway In”模块标志着 FPGA 边界的开始，能够将输入的浮点转换成定点数。“Gateway Out”模块标志着 FPGA 边界的结束，将芯片的输出数据转换成双精度数。在 Simulink 环境中双击这两个模块会弹出配置对话框，用于设定不同的转换规则，如图 8-10 所示。

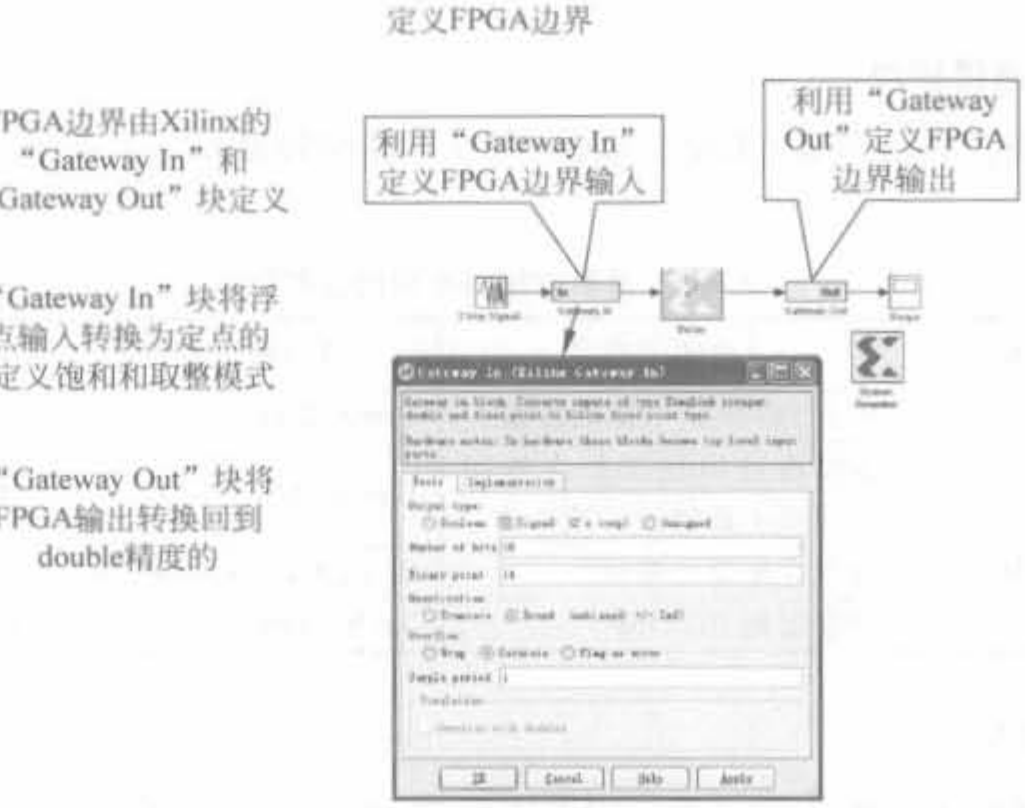


图 8-10　转换模块示意图

12. System Generator 标志

每个 System Generator 应用框图都必须至少包含一个 System Generator 标志，如图 8-11 所示，否则会提示错误。标志模块用来驱动整个 FPGA 实现过程，不与任何模块相连。双击标志模块，可以打开属性编辑框，能够设置目标网表、器件型号、目标性能以及系统时钟频率等指标。

13. 建立简易的 DSP 设计

一旦定义了 FPGA 边界，就可以通过 Xilinx DSP 模块集合来建立各种 DSP 设计，包括滤波器、存储器、算术运算器、逻辑和比特操作器等丰富资源，每个模块都有详细的工作频率和比特宽度定义。标准的 Simulink 模块不能在“Gateway In”和“Gateway Out”之间使用，但常用来产生测试数据，以及对 FPGA 的输出数据进行处理和分析。下面给出一个简单的 FPGA 系统设计实例。

例 8-2　使用 System Generator 建立一个 3 输入(a、b、c)的 DSP4 模块的计算电路，使得输出 $p=c+a\times b$，并利用标准的 Simulink 模块对延迟电路进行功能验证。

1）打开 Simulink 库浏览器并建立一个新的 Simulink 模型，并保存为 mydsp. mdl。

2）在浏览器中选择 Xilinx DSP48 模块，并将其拖拽到 mydspmydelay. mdl；按照同样的方法添加边界定义模块以及 System Generator 标志模块。

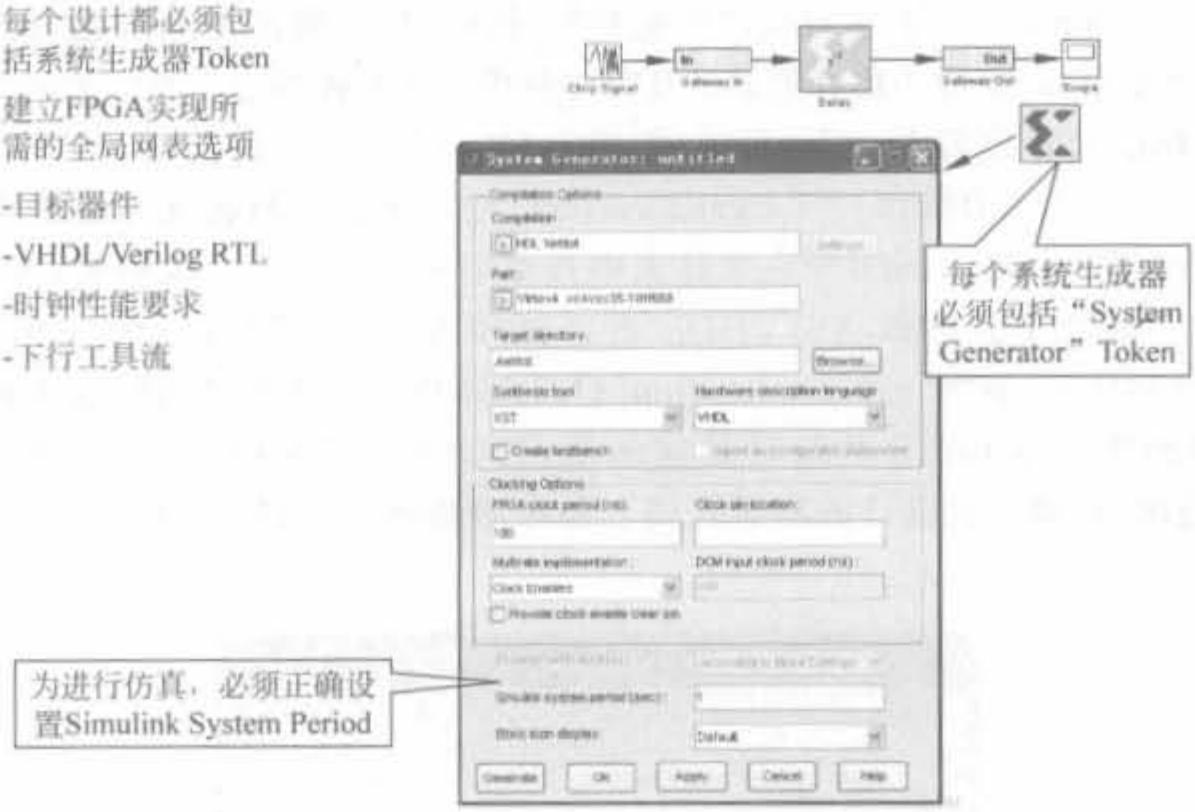


图 8-11 System Generator 标志模块示意图

3）为了测试 DSP 计算电路，添加 Simulink 标准库中的常数模块(Constant)和显示器(Display)模块。其中，常数模块用于向 DSP 计算电路灌数据，作为测试激励；显示器则用于观测输出数据。

4）连接模块，将所有的独立模块连成一个整体。其中，Xilinx 模块之间的端口可以直接相互连接，直接从一个端口拖拽鼠标到另一个端口来完成；或选中目标模块，按住 Ctrl 键，再单击要连接的模块，Simulink 即可自动将两个模块连接起来。Xilinx 模块和非 Xilinx 模块之间的连接需要边界模块(Gateway)来衔接。经过连接的设计如图 8-12 所示。

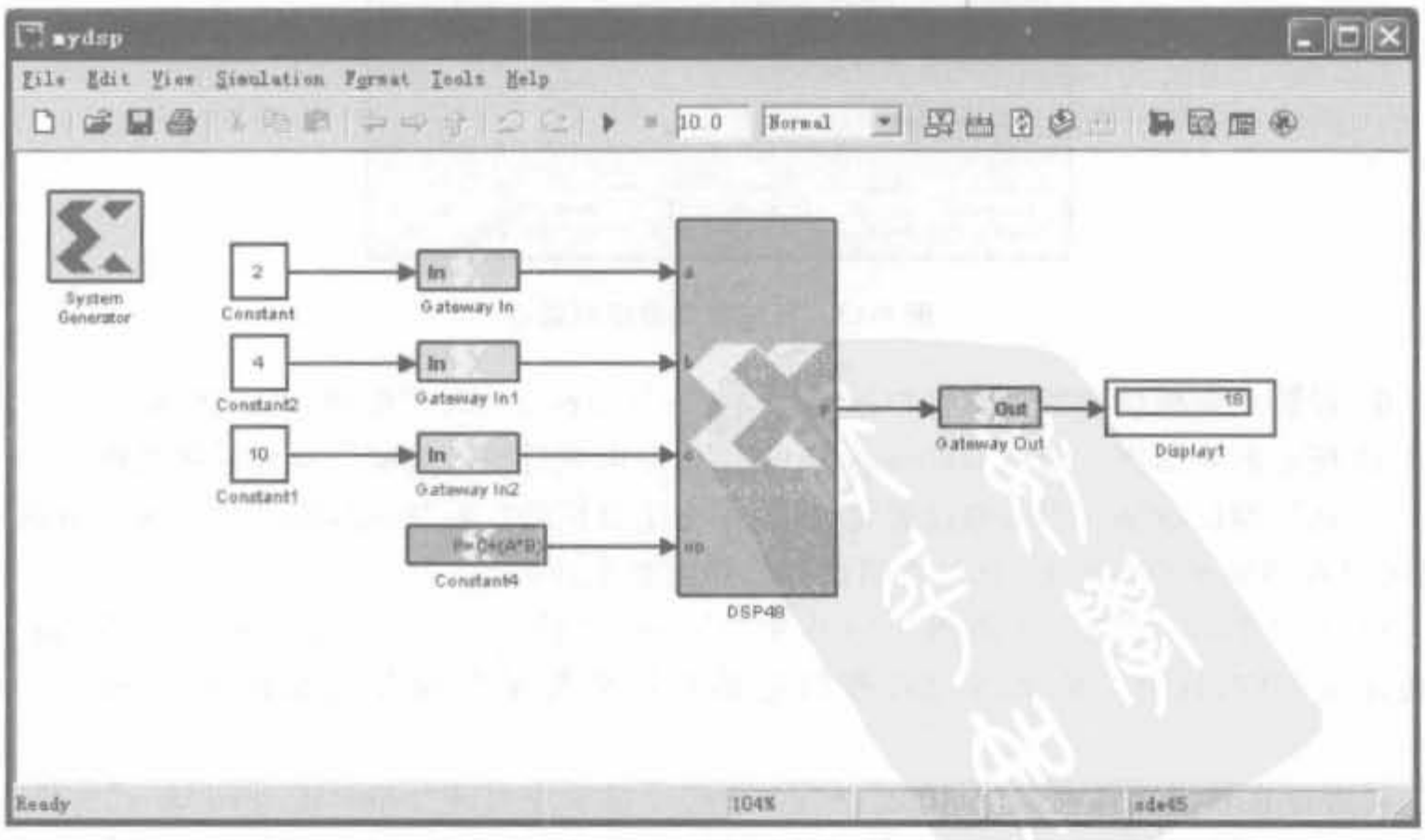


图 8-12 延迟模块以及测试平台的组成架构

5）设定系统参数。双击“System Generator”模块，会出现系统设定对话框，如图 8-13 所示。其中，“Compilation”栏选择编译生成对象，包括 HDL 网表、FPGA 配置比特流、NGC 网表、EDK 导出工具、硬件协仿真类型以及时序分析文件 6 种类型。本例选择 HDL 网表类型，会生成 ISE 工程以及相应的 HDL 代码。“Part”栏用于选择芯片型号，本例选择 Spartan3E xc4vsx55-12ff1148。“Target directory”栏用于选择目标文件存放路径，本例使用默认值，则会在 mydelay. mdl 所在文件夹中自动生成一个 netlist 文件夹，用于存放相应的输出文件。综合工具选择 XST，HDL 语言选择 Verilog 类型，系统时钟设的周期为 100ns，即为 10MHz。在“Clock pin location”栏的文本框中输入系统时钟输入管脚，则会自动生成管脚约束文件（由于本例只是演示版，所以该项空闲）。此外，可选中“Create testbench”选项，自动生成设计的测试代码。各项参数确认无误后，单击“OK”按钮，保存参数。

图 8-13　系统参数设定对话框

6）设置关键模块参数。双击“Gateway In”、“Gateway Out”模块，会弹出如图 8-14 和图 8-15 所示的对话框。通过 Gateway In 模块属性可查看输入数据位宽和量化规则。

7）运行测试激励。当参数设置完成后，单击工具栏的“▶”按钮，即可运行 Simulink 仿真，可以看到显示器输出为“18”，表明设计的功能是正确的。

8）生成 HDL 代码。单击图 8-13 中的“Generate”按钮，System Generator 可自动将设计转化成 HDL 代码。整个转化过程的起始和结束提示界面分别如图 8-16 和图 8-17 所示。

读者可在相应文件夹的“netlist”→“sysgen”子目录中打开“nonleaf_results. v”文件，查看相应的代码，如下所示（为了节约篇幅，分栏显示）。用户可将其作为子模块直接使用：

图 8-14 Gateway In 模块属性对话框

图 8-15 Gateway Out 模块属性对话框

图 8-16 自动生成代码过程的起始提示标志

图 8-17 自动生成代码过程的结束提示标志

```
module mydsp (

  ce_1,
  clk_1,
  gateway_in,
  gateway_in1,
  gateway_in2,
  gateway_out
);

  input [0:0] ce_1;
  input [0:0] clk_1;
  input [17:0] gateway_in;
  input [17:0] gateway_in1;
  input [47:0] gateway_in2;
  output [47:0] gateway_out;

  wire [0:0] ce_1_sg_x0;
  wire [0:0] clk_1_sg_x0;
  wire [10:0] constant4_op_net;
  wire [17:0] gateway_in1_net;
  wire [47:0] gateway_in2_net;
  wire [17:0] gateway_in_net;
  wire [47:0] gateway_out_net;

  assign ce_1_sg_x0 = ce_1;
  assign clk_1_sg_x0 = clk_1;
  assign gateway_in_net = gateway_in;
  assign gateway_in1_net = gateway_in1;
  assign gateway_in2_net = gateway_in2;
  assign gateway_out = gateway_out_net;


  constant4_8f31da0b95  constant4 (
    .ce(1'b0),
    .clk(1'b0),
    .clr(1'b0),
    .op(constant4_op_net)
```

```
    );

    xldsp48 #(

      .areg(1),
      .b_input("DIRECT"),
      .breg(1),
      .c_use_b(1),
      .c_use_bcin(0),
      .c_use_c(1),
      .c_use_ce_carry_in(0),
      .c_use_ce_ctrl(0),
      .c_use_ce_mult(0),
      .c_use_cea(0),
      .c_use_ceb(0),
      .c_use_cec(0),
      .c_use_cep(0),
      .c_use_en(0),
      .c_use_pcin(0),
      .c_use_rst(0),
      .c_use_rsta(0),
      .c_use_rstb(0),
      .c_use_rstc(0),
      .c_use_rstcarryin(0),
      .c_use_rstctrl(0),
      .c_use_rstm(0),
      .c_use_rstp(0),
      .carryinreg(1),
      .carryinselreg(1),
      .creg(1),
      .legacy_mode("MULT18X18S"),
      .mreg(1),
      .opmodereg(1),
      .preg(1),
      .subtractreg(1),
      .use_c_port(1),
      .use_op(1))
    dsp48_x0 (
      .a(gateway_in_net),
      .b(gateway_in1_net),
      .c(gateway_in2_net),
      .carryin(1'b0),
      .ce(ce_1_sg_x0),
      .cea(1'b1),
      .ceb(1'b1),
      .cec(1'b1),
      .cecarryin(1'b1),
      .cecinsub(1'b1),
      .cectrl(1'b1),
      .cem(1'b1),
      .cep(1'b1),
      .clk(clk_1_sg_x0),
      .clr(1'b0),

      .en(1'b1),
      .op(constant4_op_net),
      .rst(1'b0),
      .rsta(1'b0),
      .rstb(1'b0),
      .rstc(1'b0),
      .rstcarryin(1'b0),
      .rstctrl(1'b0),
      .rstm(1'b0),
      .rstp(1'b0),
      .subtract(1'b0),
      .p(gateway_out_net)
    );
endmodule
```

8.3.2 System Generator 中的信号类型

System Generator 是面向硬件设计的工具，因此数据类型只能是定点的，而 Simulink 中的基本数据类型是双精度浮点型，因此 Xilinx 模块和 Simulink 模块连接时需要通过边界模块来转换。“Gateway In”模块把浮点数转换成定点数，“Gateway Out”模块把定点数转换成浮点数。此外，对于 Simulink 中的连续时间信号，还必须经过“Gateway In”模块的采样转换才能使用。

System Generator 中的数据类型命名规则是非常简易且便于记忆的形式，如 Fix_8_6 表示此端口为 8bit 有符号数，其中 6bit 为小数部分。如果是无符号数，则带有“Ufix”前缀。在 System Generator 中，可通过选择 “Format” 菜单中的“Port/Signal Display”→“Port Data Types”命令，来显示所有端口的数据类型，形象显示整个系统的数据精度。

Xilinx 模块基本上都是多形态的，即可根据输入端口的数据类型来确定输出数据类型，但在有些情况下需要扩展信号宽度来保证不丢失有效数据。此外，也允许设计人员自定义模块的输入、输出数据的量化效果以及饱和处理。在图 8-14 所示的“Gateway In”模块属性对话框中，“Output type”选择数据为布尔型、有符号数还是无符号数；“Number of bits”即为定点数的位宽；“Binary point”为小数部分的宽度；“Quantization”选择定点量化模式；“Overflow”用于设定饱和处理模式；“Sample period”用于对连续时间信号的采样。因此按照 System Generator 的数据形式命名规则，“Gateway In”模块的数据类型为 Fix/UFix_(Number of bits)_(Binary point)。

此外，还有 DSP48 instruction，显示为“UFix_11_0”，是 Xilinx 针对数字信号处理的专用模块，用于实现乘加运算。

8.3.3　自动代码生成

System Generator 能够自动地将设计编译为低级的 HDL 描述，且编译方式多样，取决于 System Generator 标志中的设置。为了生成 HDL 代码，还需要生成一些辅助下载的文件，如工程文件、约束文件等，以及用于验证的测试代码(HDL testbench)。

1. 编译并仿真 System Generator 模块

前面已经提到，要对一个 System Generator 的设计进行仿真或者将其转化成硬件，设计中必须包含一个 System Generator 生成标志。也可以将多个生成标志分布于不同的层中(一层一个)。在层状结构中，处于别的层下的称为从模块，不属于从模块的则为主模块。但是特定的参数(如系统时钟频率)只能在主模块中设置。

对于任一添加的模块，都可以在 System Generator 模块中指定其代码生成方式和仿真处理形式。要编译整个系统，在顶层模块中利用 System Generator 模块生成代码即可。

不同编译类型的设定将会产生不同的输出文件，可选的编译类型包括两个网表文件类型(HDL 网表和 NGC 网表)、比特流文件类型、EDK 导出工具类型以及时序分析类型 4 类。

(1) HDL 网表类型是最常用的网表结构，其相应的输出结果包括 HDL 代码文件、EDIF 文件和一些用于简化下载过程的辅助文件。设计结果可以直接被综合工具(如 XST 等)综合，也可以反馈到 Xilinx 物理设计工具(如 ngdbuild、map、par 和 bitgen 等)来产生配置 FPGA 的比特流文件。编译产生的文件类型和 ISE 中是一致的。NGC 网表类型的编译结果和 HDL 网表类似，只是用 NGC 文件代替了 HDL 代码文件。

(2) 比特流文件类型的编译结果是直接能够配置 FPGA 的二进制比特流文件，并能直接在 FPGA 硬件平台上直接运行。如果安装了硬件协仿真平台，可以通过选择“Hardware Co-simulation”→“Xtreme DSP Development Kit”→“PCI and USB”，生成适合 Xtreme DSP 开发板的二进制比特流文件。

(3) EDK 导出工具类型的编译结果是可以生成直接导入 Xilinx 嵌入式开发工具(EDK)的工程文件以及不同类型的硬件协仿真文件。

(4) 时序分析类型的编译结果是该设计的时序分析报告。

2. 编译约束文件

在编译一个设计时，System Generator 会根据用户的配置产生相应的约束文件，通知下载配置工具如何处理设计输入，不仅可以完成更高质量的实现，还能够节省时间。

约束文件可控的指标包括：

- 系统时钟的周期；
- 系统工作速度，和系统时钟有关且设计的各个模块必须运行的速度；
- 管脚分配；
- 各个外部管脚以及内部端口的工作速度。

约束文件的格式取决于 System Generator 模块的综合工具。对于 XST，其文件为 XCF 格式；对于 Synplify/Synplify Pro，则使用 NCF 文件格式。

系统时钟在 System Generator 标志中设定，编译时将其写入约束文件，在实现时将其作为头等目标。在实际设计中，常常包含速度不同的多条路径，其中速度最高的采用系统时钟约束，其余路径的驱动时钟只能通过系统时钟的整数倍分频得到。当把设计转成硬件实现时，“Gateway In”和“Gateway Out”模块就变成了输入、输出端口，其管脚分配和接口数据速率必须在其参数对话框中设定，编译时会将其写入 I/O 时序约束文件中。

3. HDL 测试代码

通常，System Generator 设计的比特宽度和工作频率都是确定的，因此 Simulink 仿真结果也要在硬件上精确匹配，需要将 HDL 仿真结果和 Simulink 仿真结果进行比较，才能确认 HDL 代码的正确性。特别当其包含黑盒子模块时，这样的验证显得格外重要。System Generator 提供了自动生成测试代码的功能，并能给出 HDL 代码仿真正确与否的指示。

假设设计的名字是＜design＞，双击顶层模块的 System Generator 标志，将“Compilation”选项设为“HDL Netlist”，选中“Create Testbench”选项，然后单击“Generate”选项，不仅可以生成常用的设计文件，还有下面的测试文件：

- ＜design＞_tb. vhd/. v 文件，包含完整的 HDL 测试代码；
- Various. dat 文件，包含了测试代码仿真时的测试激励向量和期望向量；
- 脚本 Scripts vcom. do 和 vsim. do 文件，用于在 Modelsim 中完成测试代码的编译和仿真，并将其结果和自动编译产生的 HDL 测试向量进行比较。

Various. dat 文件是 System Generator 将通过“Gateway In/Out”模块的数据保存下来而形成的。其中，经过输入模块的数据是测试激励，通过输出模块的数据就是期望结果。测试代码只是简单的封装器，将测试激励送进生成的 HDL 代码，然后对输出结果和期望结果进行比较，给出正确指示。

8.3.4 编译 MATLAB 设计生成 FPGA 代码

Xilinx 公司提供了两种方法将 MATLAB 设计. m 文件转化为 HDL 设计，一种是利用 AccelDSP 综合器；另一种是直接利用 MCode 模块。前者多应用于复杂或高速设计中，常用来完成高层次的 IP 核开发；而后者使用方便，支持 MATLAB 语言的有限子集，对实现算术运算、有限状态机和逻辑控制是非常有用的。本节内容以介绍 MCode 模块为主。

MCode模块实现的是装载在里面的.m函数的功能。此外，还能够使用Xilinx的定点类型数对.m函数进行评估。该模块使用回归状态变量，以保证内部状态稳定不变，其输入、输出端口都由.m函数确定。

要使用MCode模块，必须实现编写.m函数，且代码文件必须和System Generator模型文件放在同一个文件夹中，或者处于MATLAB路径上的文件夹中。下面用两个实例来说明如何使用MCode模块。

例8-3　使用MATLAB编写一个简单的移位寄存器，完成对输入数据乘以8以及除以4的操作，并使用MCode将其编译成System Generator直接可用的定点模块。

(1) 相关的.m函数代码为：

```
function [lsh3,rsh2] = xlsimpleshift(din)
% [lsh3,rsh2] = xlsimpleshift(din) does a left shift 3 bits and a
% right shift 2 bits. The shift operation is accomplished by
% multiplication and division of power of two constant.
lsh3 = din * 8;
rsh2 = din/4;
```

(2) 将.m函数添加到下列三个位置之一：

- 模型文件存放的位置；
- 模型目录下名字为private的子文件夹；
- MATLAB路径下。

然后，新建一个System Generator设计，添加MCode模块。双击该模块，在弹出的页面中，通过Browse按钮将.m函数和模型设计关联起来，如图8-18所示。

图8-18　MCode模块关联界面示意图

(3) 添加边界模块、Sytem Generator模块、正弦波测试激励以及示波器模块构成完整的设计，如图8-19所示。

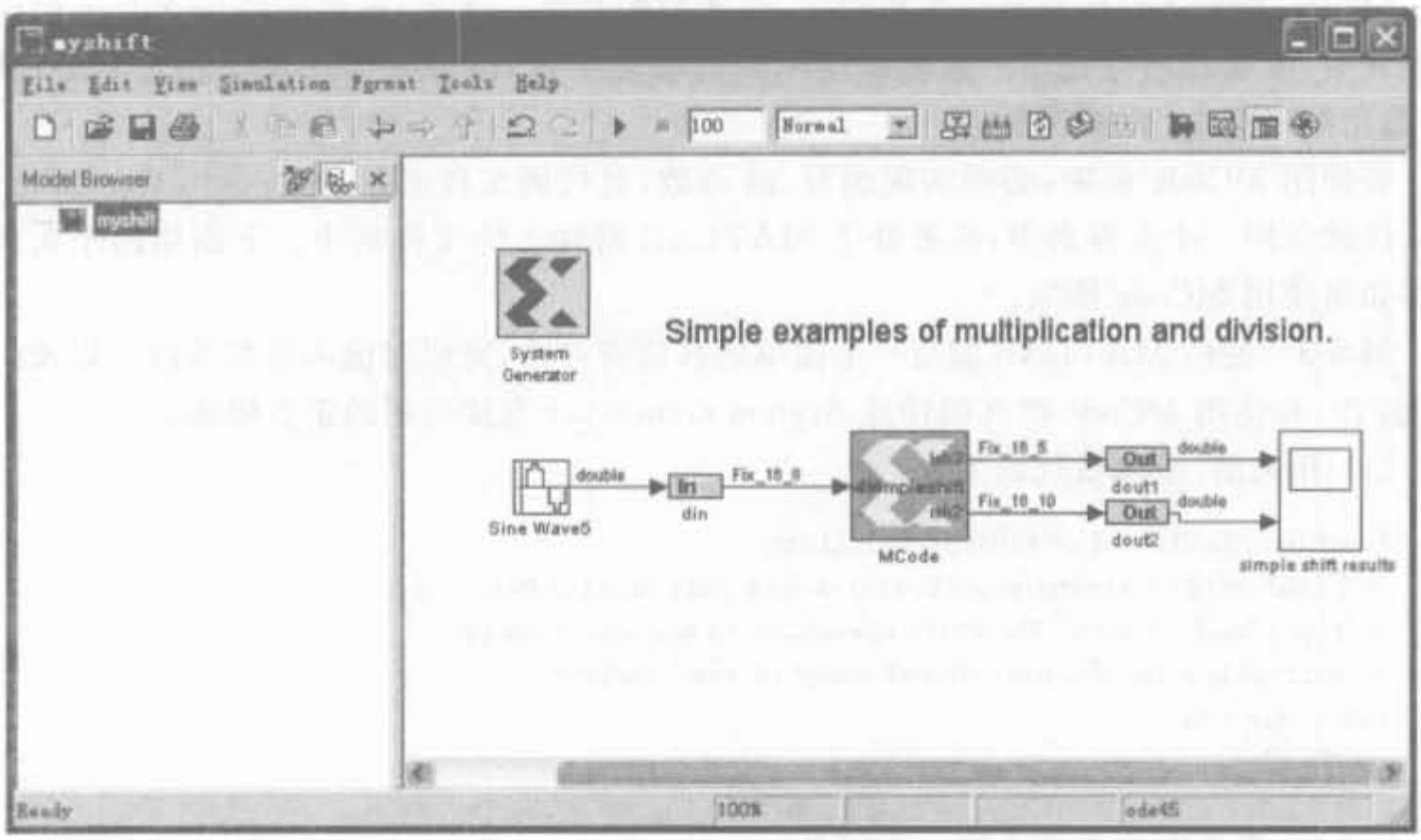


图 8-19　简单移位模块设计示意图

(4) 运行仿真,得到的结果如图 8-20 所示。从中可以看出,设计是正确的,它正确实现了.m 文件的功能。左图将信号放大了 8 倍,右图将信号缩小了 4 倍。

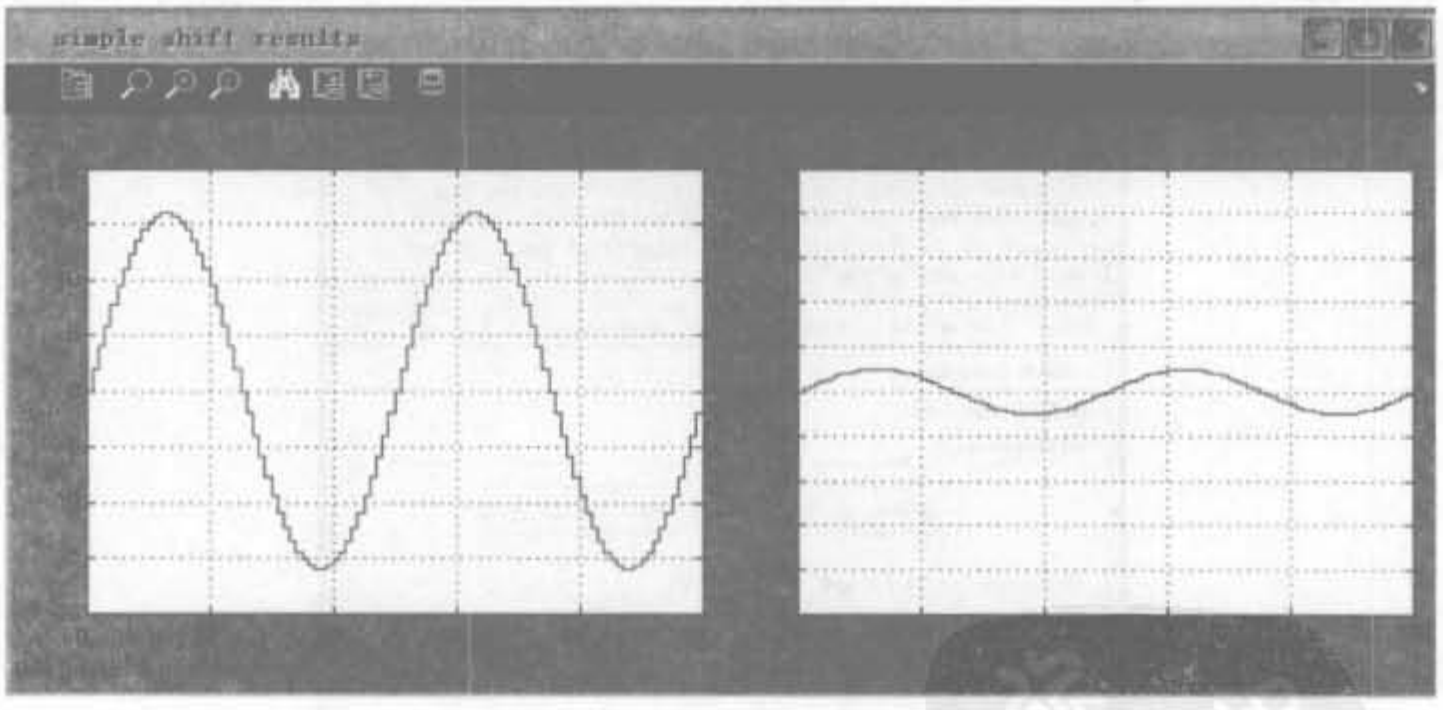


图 8-20　简单移位模块仿真结果示意图

(5) 自动生成代码,得到的 Verilog 文件如下所示:

```
module myshift (
  din,
  dout1,
  dout2
);
```

```
    input [15:0] din;
    output [15:0] dout1;
    output [15:0] dout2;

    wire [15:0] din_net;
    wire [15:0] dout1_net;
    wire [15:0] dout2_net;

    assign din_net = din;
    assign dout1 = dout1_net;
    assign dout2 = dout2_net;

    mcode_6b96190926  mcode (
      .ce(1'b0),
      .clk(1'b0),
      .clr(1'b0),
      .din(din_net),
      .lsh3(dout1_net),
      .rsh2(dout2_net)
    );
endmodule
```

8.3.5 子系统的建立和使用

System Generator 设计经常作为大型 HDL 设计的一部分，本节将介绍如何使用 System Generator 来建立子系统模块，以及如何在整个系统中对其进行仿真。

1. 子系统的建立以及仿真方法

子系统就是 HDL 语言中的模块，也类似于 C++语言中的函数，是有效执行自顶向下设计的必备手段。如果将一个复杂设计完全在一个单独设计中实现，则该设计的验证和复查工作将是设计人员的噩梦。此外，从设计复用的角度讲，子系统可以 IP 核的方式为多个设计使用，具有高的可重用性，能节省大系统的开发时间。

建立子系统最简单的方法就是利用 NGC 二进制网表文件，将 System Generator 设计封装成一个单独的二进制模块，综合工具将其作为黑盒子看待。在建立子系统时，管脚约束不能在 Gateway 模块中定义；同样，时钟管脚也不能在 System Generator 模块中定义，应通过网表编辑器来指定物理约束。这是因为 NGC 网表中不仅包括逻辑设计，还包括设计的约束信息。在复杂系统中建立子系统的设计流程如图 8-21 所示。

图 8-21 建立子系统的设计流程

1）NGC 网表文件

如图 8-21 所示，生成 NGC 网表是建立子系统的第一步。在 System Generator 标志中将编译生成

文件类型选为 NGC List，如图 8-22 所示。如果设计中有时钟驱动电路，单击“Generate”后，会在目标文件夹生成“<design>_cw. ngc”文件，否则生成“<design>. ngc”文件，其中<design>就是设计的名字。NGC 网表文件包括设计中所有的逻辑和约束信息，这意味着将 System Generator 生成的所有 HDL 文件、内核以及约束文件封装成一个单独的文件。

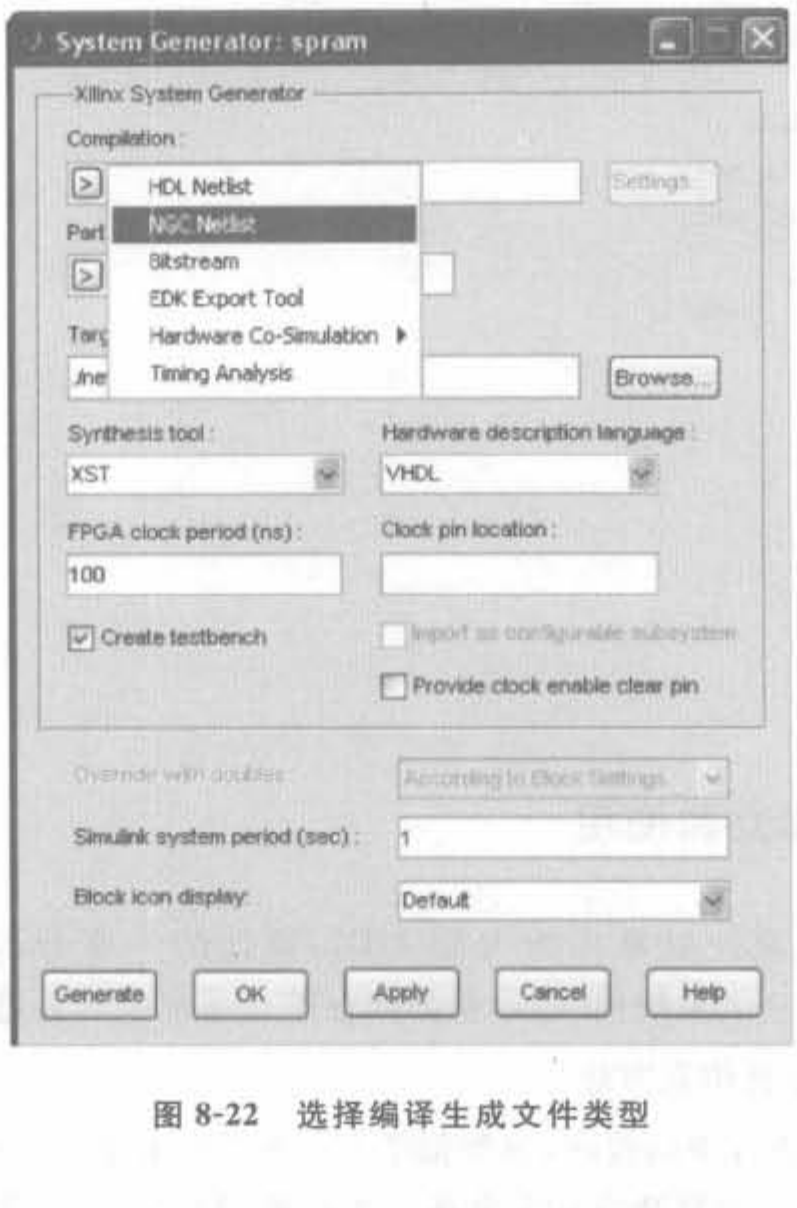


图 8-22　选择编译生成文件类型

2）设计规则

在复杂系统中使用子模块时，必须遵循下面两条规则：

首先，不能在子模块设计文件中出现“Gateway In”、“Gateway Out”以及 System Generator 标志模块，否则 NGDBuild 工具会产生下面的警告：

```
WARNING: NgdBuild: 483-Attribute "LOC" on "clk" is on the wrong type of
object. Please see the Constraints Guide for more information on this
attribute.
```

其次，不能在综合的时候往 NGC 网表文件中插入 I/O 缓存器，否则会报错。I/O 缓存器只能在顶层模块中使用。

3）逻辑综合

当使用子系统的 NGC 网表文件综合时，其流程如图 8-23 所示。NGC 模块可在顶层模块中以黑盒子的方式直接例化。为了简化该过程，当通过 NGC 目标编译后，System Generator 提供了 HDL 例化模板，保存在设计路径，且以“<design>_cw. veo”命名。当选择 VHDL 语言时，其模板名为“<design>_cw. vho”。

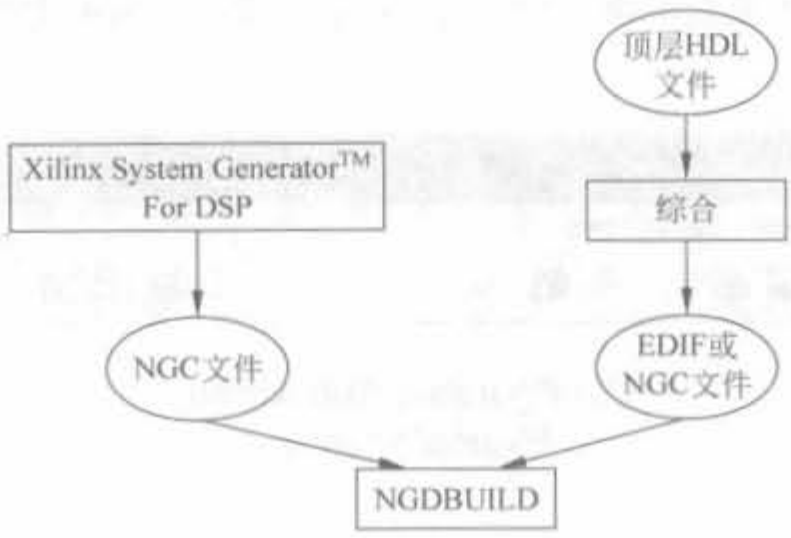


图 8-23　综合时的流程图

4）仿真

把 System Generator 模型编译成 NGC 目标后，生成的 HDL 文件只能完成 HDL 仿真，不能在 ISE 中综合。由于 HDL 文件不能包含在工程中，如果要在 HDL 仿真器中运行整个设计，必须指定用户.do 文件。除了 HDL 文件之外，还需要将内存初始化文件(.mif)和系数文件(.coe)及 HDL 文件放在同一文件夹中。

2. 可配置子系统的建立

可配置子系统是一类可以作为标准元件使用的 Simulink 模块，但又和标准元件模块不同，它存在多种可选功能，每一种功能都可以实现，让用户灵活选择。以可配置 FIR 滤波器为例，实现快速滤波器需要很多资源，而许多低速的滤波器只需要相对很少的资源，将其做成可配置的模块，就可以允许用户根据实际情况在速度和硬件代价之间做出最优平衡。这体现了可配置子模块最大的优势。

1）定义一个可配置子模块

可通过新建 Simulink 库来定义可配置子系统，且可选模块的实现也由库来管理。下面给出新建库的具体步骤。

(1) 新建一个空白库，如图 8-24 所示。

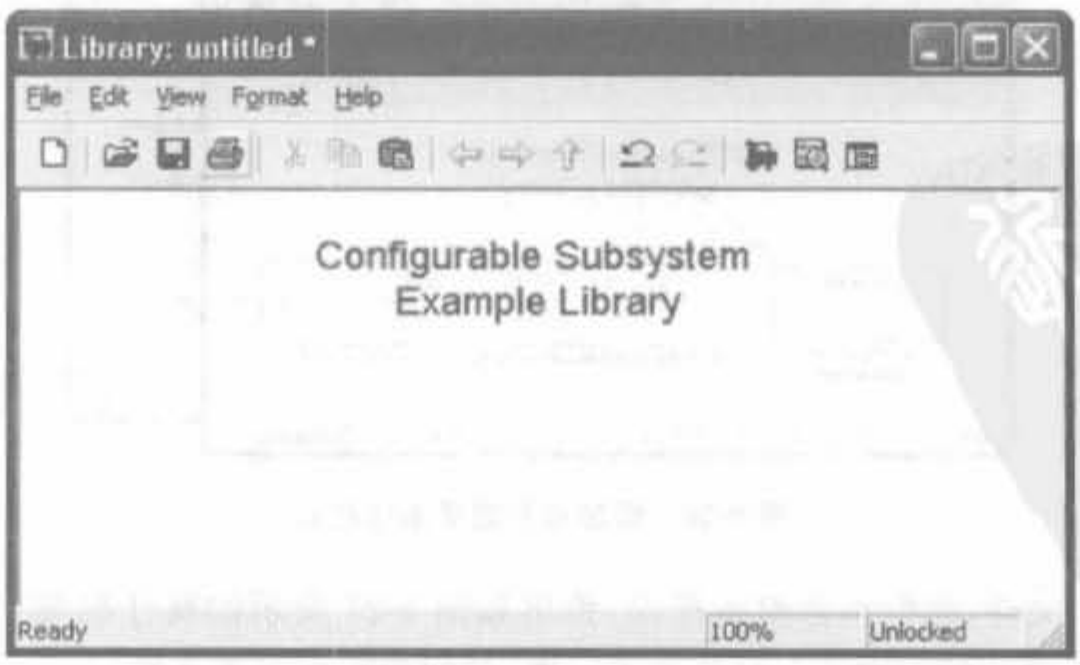


图 8-24　新建空白库

（2）在库里添加基本实例模块，如图 8-25 所示。基本实例模块可以是 System Generator 中的任意组件。

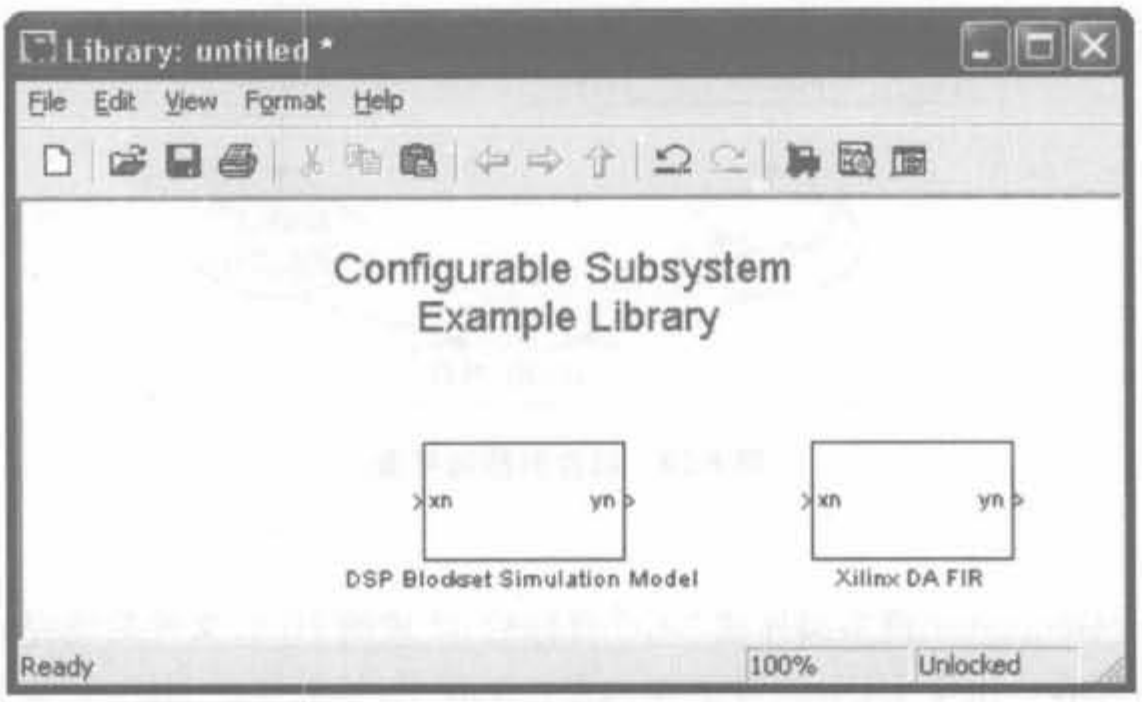


图 8-25　添加基本实例模块

（3）在库里面添加可配置子系统模板，如图 8-26 所示。模板可以在 Simulink 库浏览器中找到，其具体位置为“Simulink/Ports & Subsystems/Configurable Subsystem”。如果有需要，用户可以修改该模板的名字。

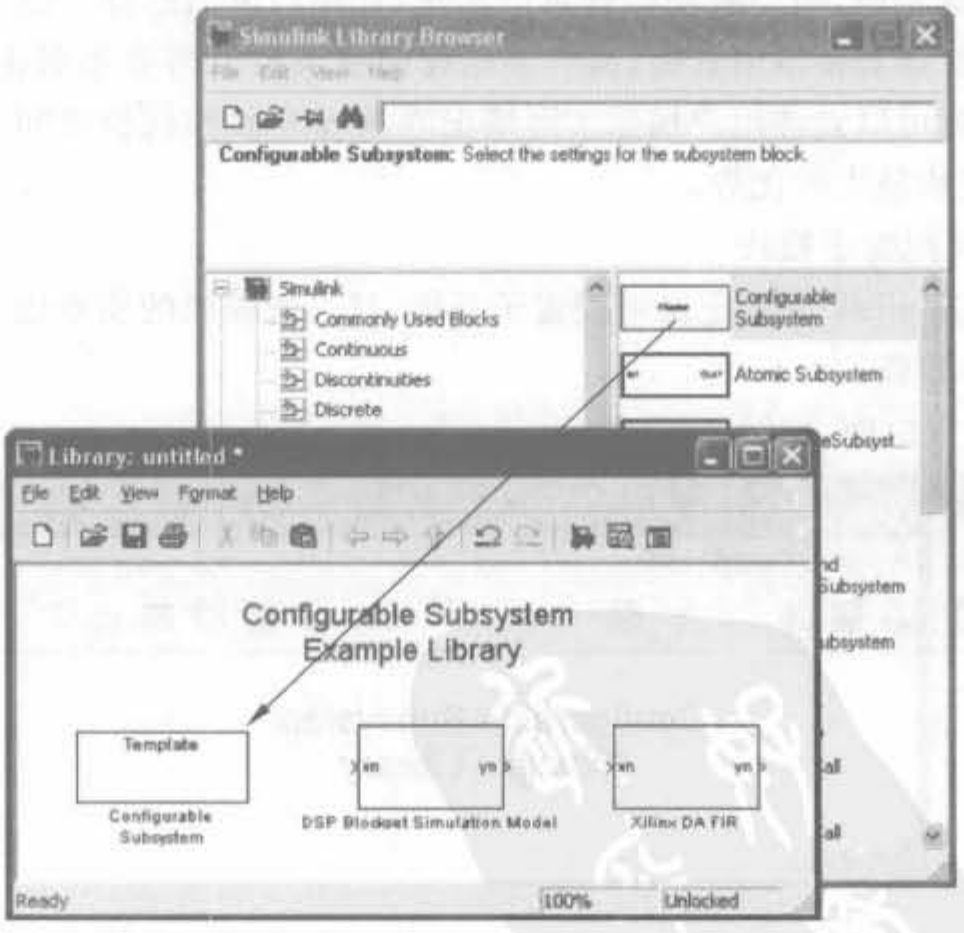


图 8-26　添加可配置子系统模板

（4）保存库文件，然后双击模板模块，弹出如图 8-27 所示的属性配置界面。根据实现的需要选中相关模块的检验框。最后单击“OK”按钮再次保存库文件。

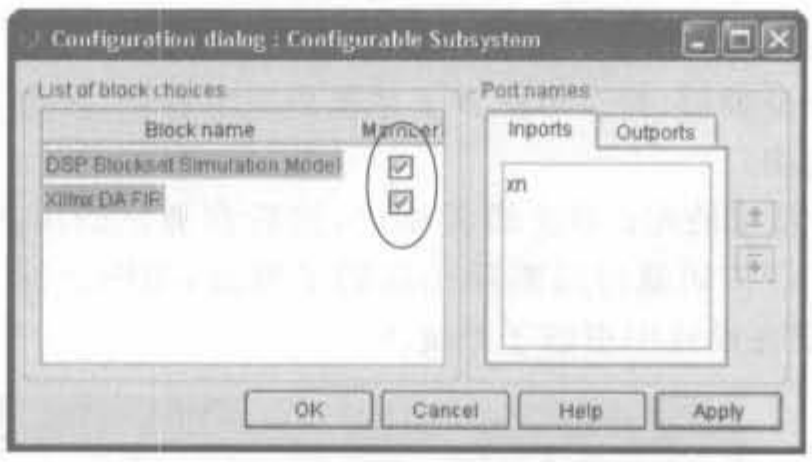


图 8-27　属性配置界面

2）使用可配置子系统

要在设计中使用可配置子系统，先按照上面的步骤定义子系统，然后打开库，将需要的模板模块拖到设计中合适的位置，设计中就有了可配置子系统的实例，如图 8-28 所示的 FIR 滤波器模块。

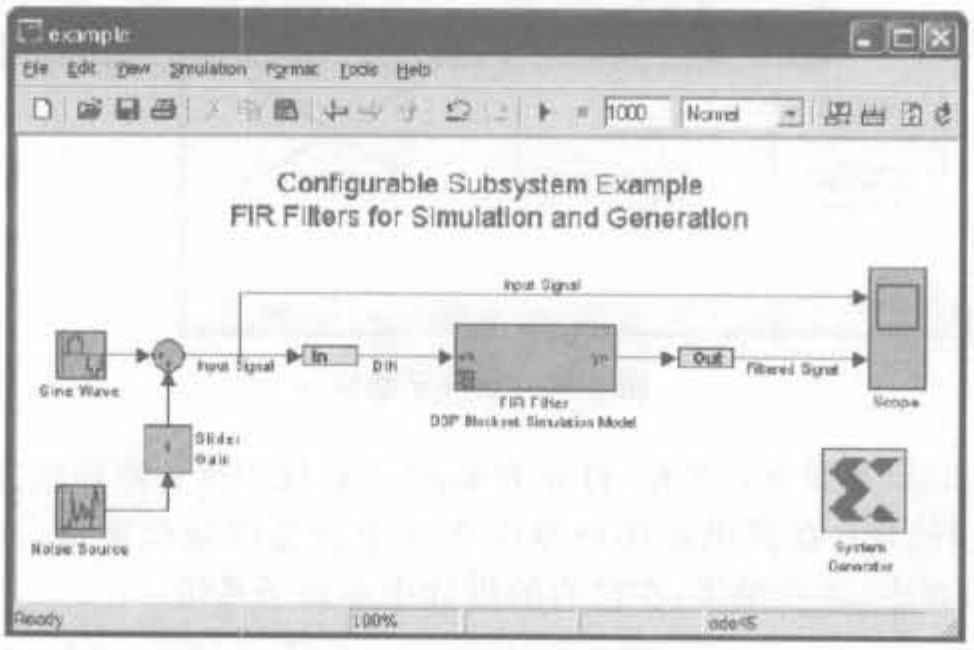


图 8-28　可配置子系统实例

在实例模块上单击鼠标右键，然后选择“Block Choice”选项中的“Xilinx DA FIR”，将实例作为基础实现模块使用，如图 8-29 所示。

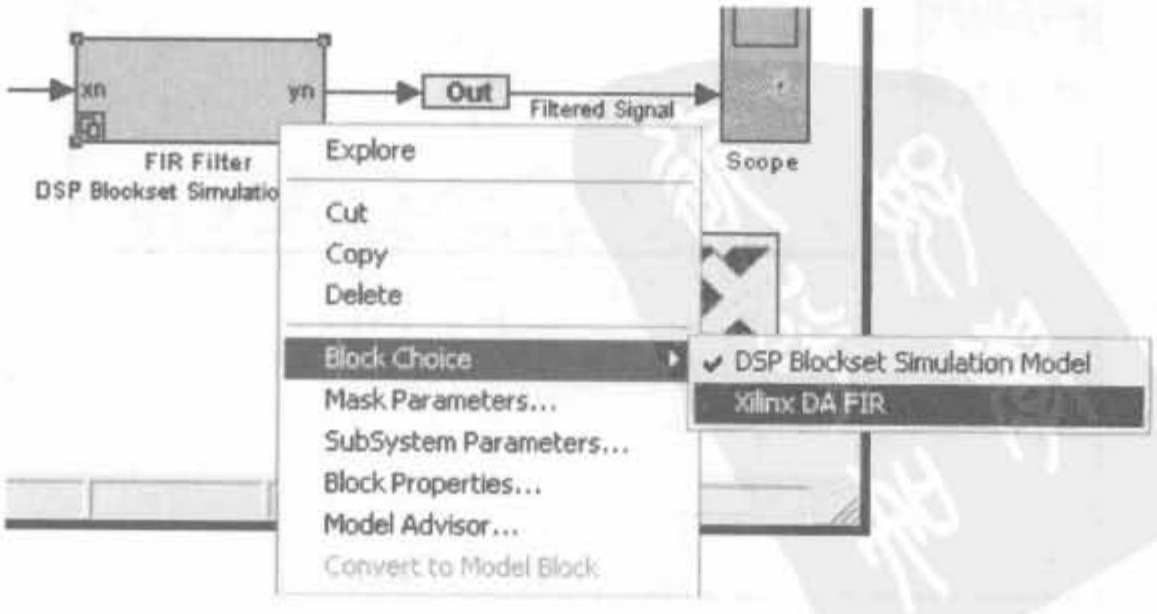


图 8-29　将实例作为基础实现模块

3）在可配置子模块中添加和删除实例模块

添加和删除子系统是编辑、修改可配置子系统的基本操作，按照下面的步骤可以从可配置子系统中删除实例模块：

（1）打开并解锁子系统的库；双击模板模块，然后在弹出的用户界面中取消相应模块检验框的选定，单击“OK”按钮就可以删除相应的子模块，如图 8-30 所示；再保存库，重新编译即可；最后，仍需要在设计中更新子系统。

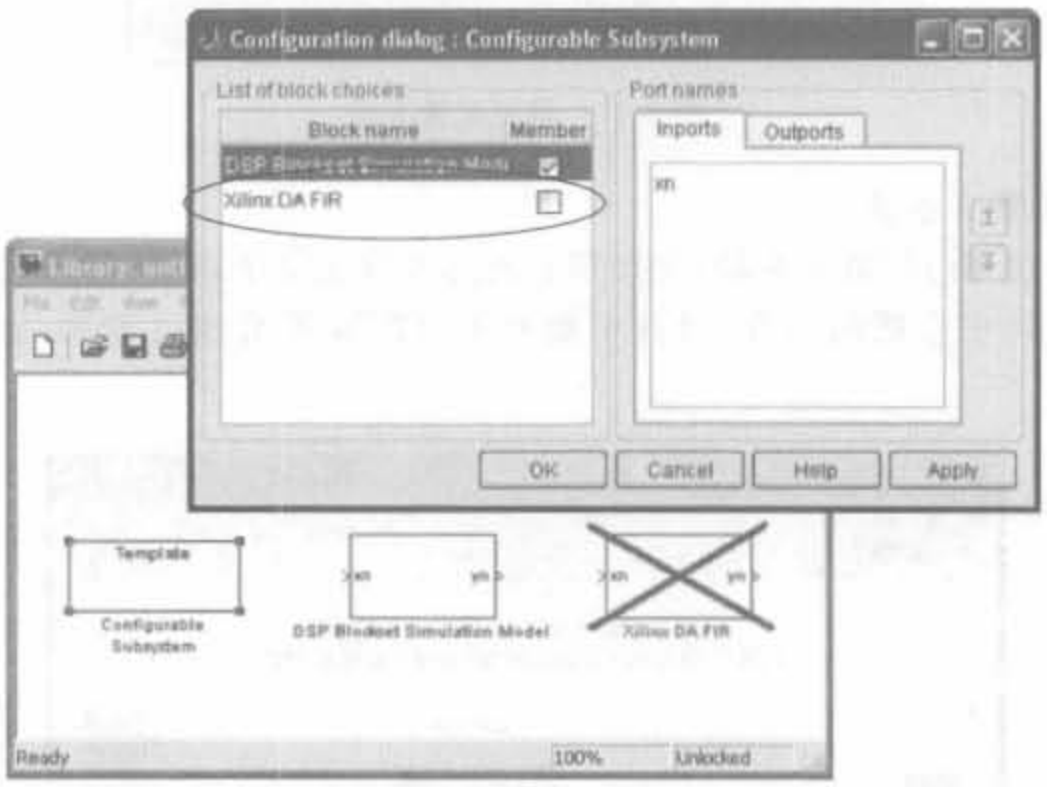


图 8-30　删除子模块

添加实例模块的步骤如下：首先，打开并解锁子系统的库；然后将实例模块拖到子系统设计中，双击模板模块；在弹出的用户界面中选中所需模块检验框，单击“OK”即可，如图 8-31 所示；再保存库，重新编译，在已有的设计中更新子系统。

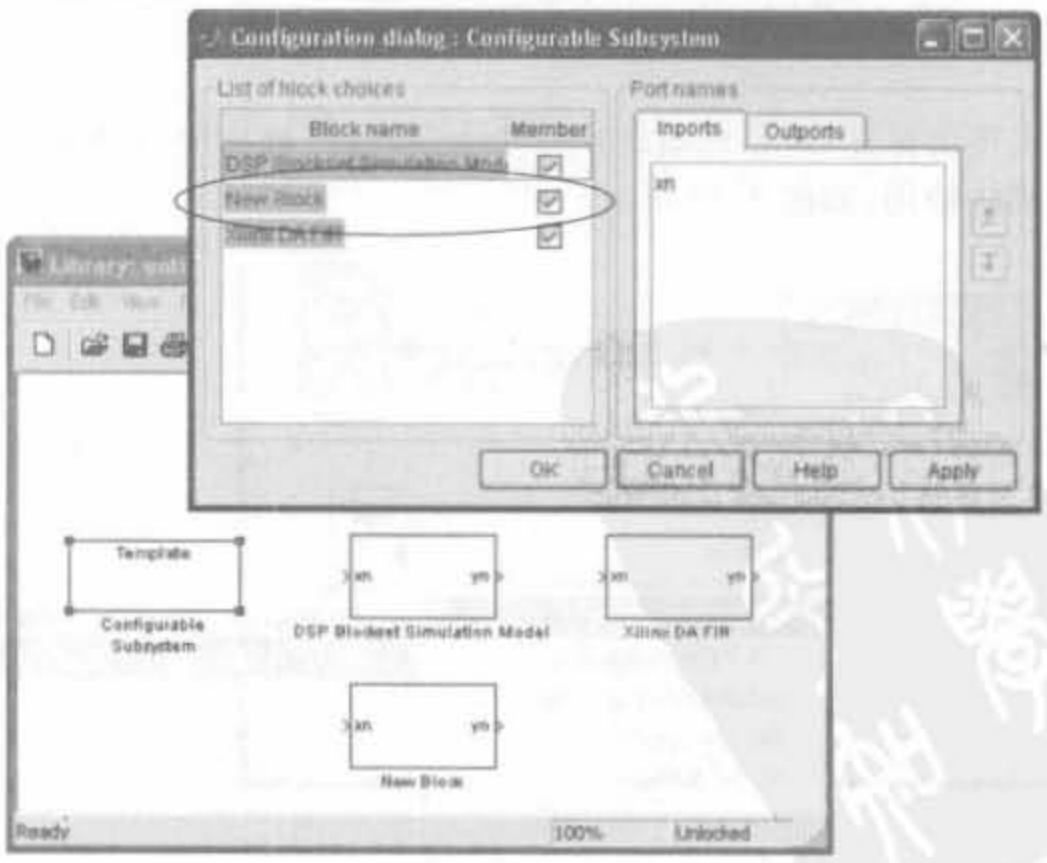


图 8-31　添加子系统

（2）生成可配置子模块的硬件电路。

在 System Generator 中，模块可用于仿真和产生硬件。有时对于一个可配置子系统，最佳的方式是其既可以成为仿真基础模块，也可以用来生成硬件电路。例如，将可配置子系统在仿真时作为普通的模块来产生仿真结果，在实现时作为黑盒子产生功能电路的 HDL 代码，就是一种常用方式。System Generator 提供了可配置子系统管理模块来达到上述目的，其使用方法如下：

首先，打开、解锁可配置子系统的库文件（除了模板模块），然后双击该模块。

其次，将可配置子系统管理模块拖到打开的库中，该管理模块可以在“Xilinx Blockset/Tools/Configurable Subsystem Manager”路径中找到，如图 8-32 所示。

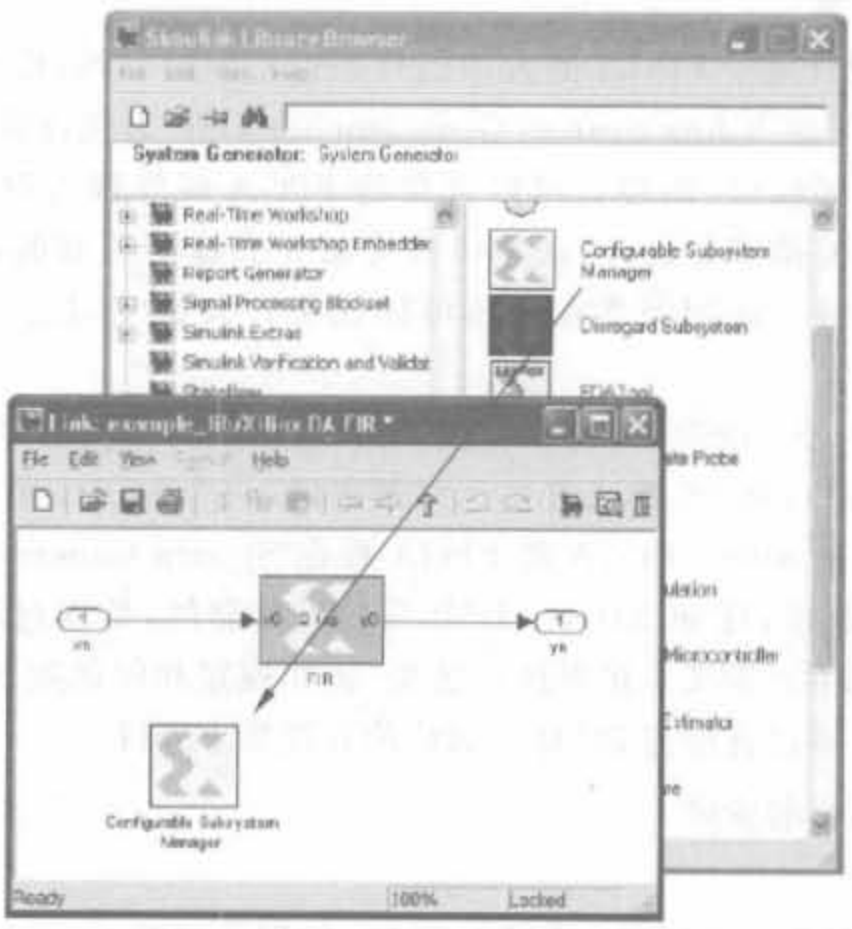


图 8-32　可配置子系统管理模块

最后，双击管理模块打开其属性配置 GUI，选择可配置子系统中生成硬件电路的模块，如图 8-33 所示。单击“OK”保存子系统并退出库文件。

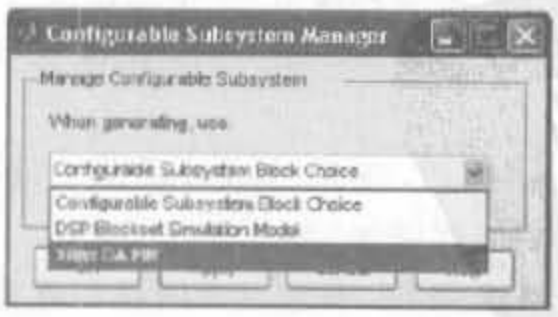


图 8-33　选择可配置子系统的硬件模块

8.4 基于 System Generator 的硬件协仿真

System Generator 允许验证工作通过实际的硬件系统来加速 Simulink 与 MATLAB 仿真，并支持以太网(10/100/千兆位)、PCI、Cardbus 及硬件平台与 Simulink 之间的 JTAG 通信。本节介绍硬件协仿真平台的建立和使用方法。

8.4.1 硬件协仿真平台的介绍与平台安装

1. 硬件协仿真介绍

通常情况下，在设计基于 FPGA 的大型信号处理系统的时候，设计人员往往需要进行费时费力的仿真。对于以 Xilinx System Generator for DSP 为代表的 FPGA 设计工具，通过提供可靠的硬件在环接口(该接口可以直接将 FPGA 硬件置入设计仿真)来解决这种问题。通过在硬件上模拟部分设计，这些接口可以大大提高仿真的速度——通常可以提高一个甚至多个数量级。使用硬件在环还可以让设计人员实时进行 FPGA 硬件调试和验证。

System Generator for DSP 可以为多类 FPGA 开发平台提供硬件在环接口。这些平台通常通过不同的物理接口和 PC 建立通信。举例来说，一个 JTAG 协仿真接口可以允许任何一个具备 JTAG 头和 Xilinx FPGA 的 FPGA 板在 System Generator for DSP 内部进行协仿真。其他类型的板卡，比如 Xtreme DSP 开发工具套件，是通过 PCI 总线通信的。目前，具有高存储带宽和吞吐率要求的系统协仿真(例如视频和图像处理)还只能在那些通过 PCI 或者是 PCMCIA 接口直接与 PC 建立通信的开发板上进行。

2. 硬件协仿真平台的安装

1) 安装必备条件

在安装前要确保具备以下硬件：电路板和 PC 的以太网接口、并口 PC4 下载线或 USB 下载线以及 14 脚的带状电缆。同时，要安装以下软件：ISE、System Generator 以及 WinPcap 4.0 版。

2) 基于以太网接口协仿真平台的安装

(1) 打开 PC“网络邻居”中的“网络连接”，在“本地连接”图标上单击鼠标右键，选择“属性”命令，双击“Internet 协议(TCP/IP)”，将 IP 地址设为 192.168.8.2，子网掩码设为 255.255.255.0。

(2) 单击属性页面的“配置”按钮，在弹出的对话框中选择“高级”页面，将属性“Flow Control”和“Speed & Duplex”的值都设为“Auto”。

(3) 通过以太网连线将电路板和 PC 通过以太网接口连接起来。

3) 基于 JTAG 接口协仿真平台的安装

基于 JTAG 接口协仿真平台不需要特殊安装，只要将 JTAG 接口通过 PC4/USB 下载线连接到 PC 即可。

8.4.2 硬件协仿真的基本操作

1. 编译硬件模型

一旦安装了硬件协仿真平台，接下来的基本操作就是建立能在硬件板中实际运行的System Generator模型或子系统。能进行协仿真的模型必须要包括一个System Generator模块，该模块定义了如何将模型编译成硬件。其制作的典型步骤如下：

1）选择编译目标

在设计中双击打开System Generator模块，在“Compilation”栏单击，选择“Hardware Co-Simulation”命令。如果用户使用Xilinx相关的开发板(ML402、ML506)，可直接选择；若使用用户板，则选择“New Compilation Target..”，如图8-34所示。

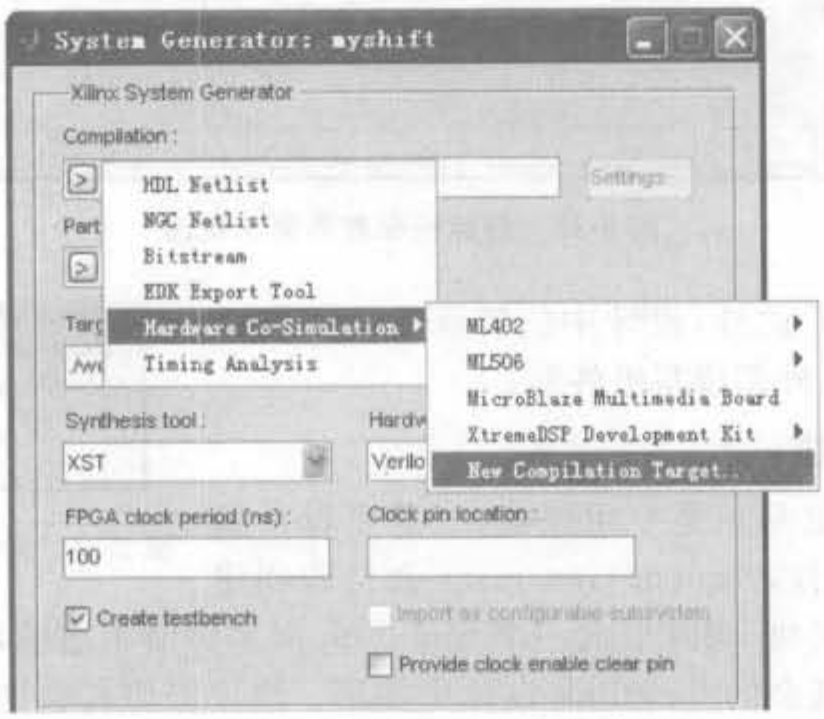


图8-34 硬件协仿真编译目标选择界面

选择了“New Compilation Target...”命令后，会弹出开发板参数配置界面。需要填入开发板的名称、系统时钟的频域与管脚约束、JTAG选项、开发板上FPGA芯片的型号以及非存储器映射端口。如果用户已具有电路板的SBDBuilder描述文件(后缀为.xml)，则直接单击“Load”按钮加载即可。输入电路板信息后，单击“Save Zip”按钮，保存当前设置到相关目录，且后缀为.zip。需要注意的是，要保证.zip的存放路径和MATLAB的当前路径一致。加载了Spartan-3E开发板的.xml文件后，配置页面如图8-35所示。

完成上述步骤后，System Generator会提示用户安装Plugin，也可取消后在MATLAB命令行输入“xlInstallPlugin('s3e_starter.zip')”，会自动弹出如图8-36所示的提示界面，单击“OK”确认。

2）调用代码生成器

在编译过程中，System Generator不仅生成了HDL代码和网表，还运行了下载工具，将网表转化成可配置比特文件。因此，单击图8-13中的“Generator”按钮，即可调用代码生成器生成适合硬件协仿真的配置比特流文件。该文件不仅包含了模型设计的硬件代码，还包括附加的额外逻辑，在PC和硬件平台之间建立一个物理接口，保障System Generator和

图 8-35　目标板配置界面示意图

平台之间的双向通信。此外，设计中的特殊电路也包含在其中，如 DCM 模块、外部读写组件等。

图 8-36　用户板安装完成提示界面

2. 硬件协仿真模块

硬件协仿真模块也是一类专用模块，一旦将设计编译成 FPGA 比特流文件，System Generator 会自动创建一个新的硬件协仿真模块，同时生成一个 Simulink 库来存储生成的模块。此外，用户可将该模块从库中复制到其余的 Simulink 设计中使用。硬件协仿真模块会自动指定驱动其模块或子系统的外部端口，而且它的端口名和类型都与原始设计的端口一致，其原理如图 8-37 和图 8-38 所示。

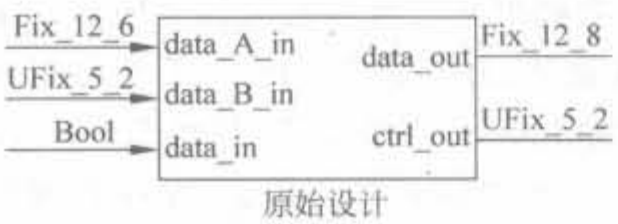


图 8-37　硬件协仿真 PC 端网络配置界面示意图

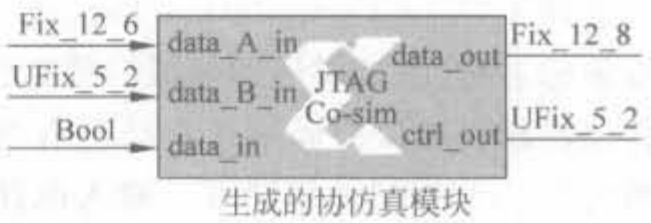


图 8-38　PC 属性值选择界面示意图

硬件协仿真模块在 Simulink 设计中和普通模块的使用方法一样的。在仿真期间，该模块和所使用的 FPGA 平台交互，自动完成芯片配置、数据传输和时钟等任务。一旦输入端口写入数据，协仿真模块会自动将相应的数据发送到硬件中合适的位置。同样，如果数据输出端口有变化，模块自动将数据从硬件中取回。

协仿真模块可以被 Xilinx 定点信号类型、Simulink 定点信号类型以及 Simulink 双精度类型驱动。输出信号类型取决于和其级联的模块。如果输出端口驱动了 System Generator 模块，则其输出数据为 Xilinx 定点信号类型；如果输出端口驱动了 Simulink 模块，则输出数据为 Simulink 数据类型。需要注意的是，一旦输入、输出数据为 Simulink 类型，则所有数

据通过四舍五入的方法来量化，溢出则采用饱和处理。

同样，协仿真模块也可具备参数化配置的能力，这是由 FPGA 平台决定的，不同的平台提供不同的参数化模型。

3. 硬件协仿真时钟

1）确定目标板时钟频率

在电路板上，一般只有单一系统时钟。到了 FPGA 内部，可通过 DCM 模块得到不同的频率，但是在协仿真中只能使用不高于系统时钟的某些特定频率值。例如，Xilinx 公司的 ML402 开发板，其系统时钟为 100MHz，但可用的硬件协仿真频率只能是 100MHz、66.7MHz、50MHz 以及 33.3MHz。

在 System Generator 中，单击协仿真模块的“Settings…”按钮，在弹出的配置对话框“Clock Frequency”栏的下拉框中选择相应的频率，如图 8-39 所示。

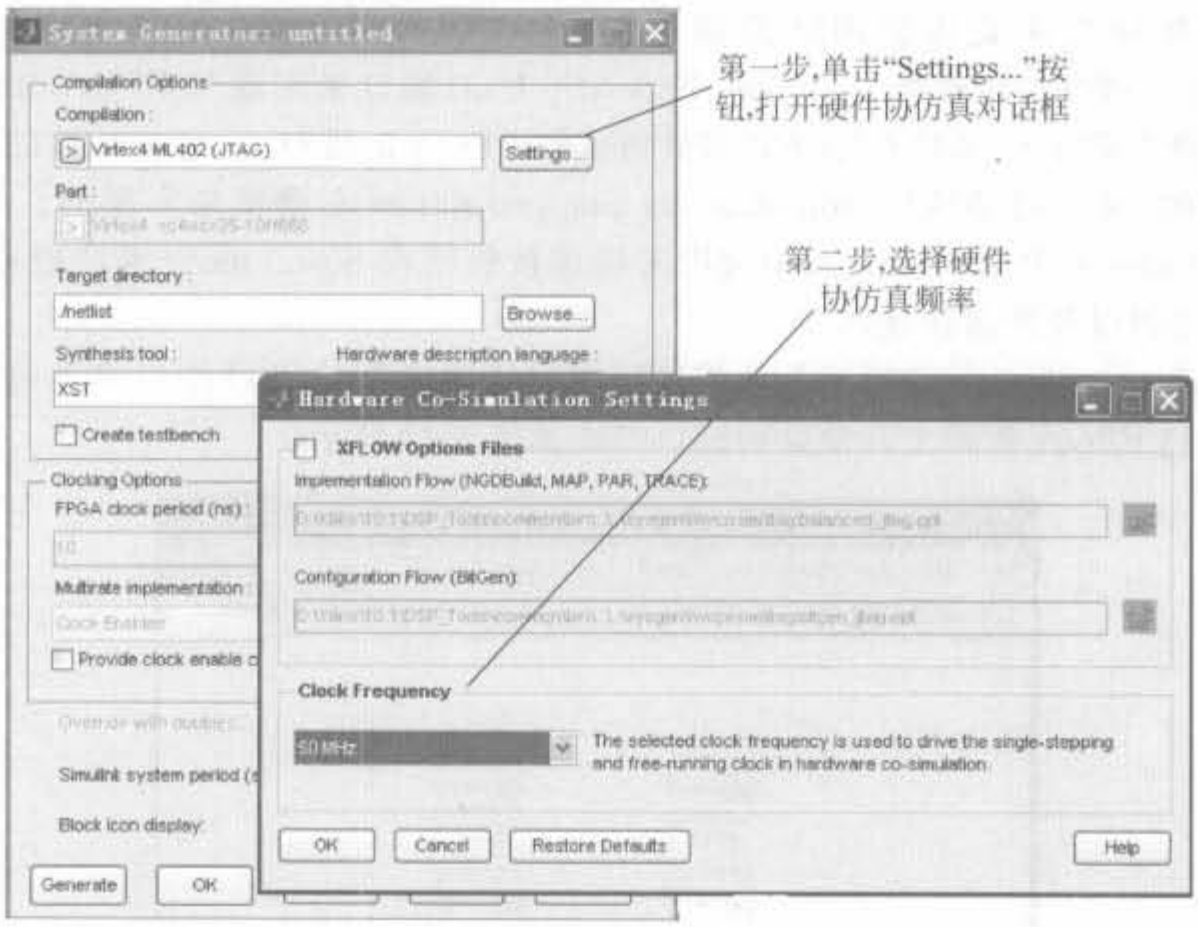


图 8-39　用户电路板安装向导界面

2）时钟模式

协仿真模块和相关联的 FPGA 芯片建立同步时钟的方法有两种：单步时钟（Signal-Step Clock）和自由运行时钟（Free-Running Clock）。在前一种模式下，FPGA 的时钟受 Simulink 控制，数据是同步采样的；在后一种模式下，FPGA 由自身内部提供时钟，当 Simulink 启动硬件仿真时，数据是通过异步采样得到的。其时钟模式配置界面如图 8-40 所示。

图 8-40　协仿真模块的时钟模式配置界面

在单步时钟模式下，硬件通过一个时钟脉冲在每个仿真阶段和软件同步，从而保证了硬件协仿真模块和原

始模型是真正意义上的比特一致或周期一致。但由于硬件协仿真模块只在 Simulink 调用它时才产生硬件工作所需的时钟信号，因此它与 Simulink 模块的通信开销以及 FPGA 芯片和 Simulink 的开销是比较大的，在一定程度上限制了硬件的性能。从经验上来讲，只要 FPGA 内部的计算量远大于通信开销，就能明显起到硬件加速的功能。

和单步时钟模式不同的是，在自由运行模式中，硬件运行和软件仿真是异步的，且不受软件控制，其时钟是连续运行的。这样，FPGA 的端口不再同步于 Simulink 仿真端口的事件，协仿真模块和原始模型就不是比特一致的。例如，当 Simulink 端口有事件发生时，需要硬件平台在此时将数据读进或写出到相应的端口上，但是由于延迟了未知的时钟周期后，硬件的状态已不能接收或发送数据，将造成仿真出错。因此，在自由运行模式中，必须在原始模型中就加入严格的同步机制。

4. 指定板级 I/O 端口

在实际的电路板中，一般都有多个芯片和 FPGA 通信，如外部存储器、ADC 以及 DAC 等。如果在协仿真中直接利用这类硬件资源，将会得到更高的仿真性能。在 System Generator 中可通过指定板级 I/O(board-specific I/O)端口来实现与 FPGA 外部芯片交互的功能。板级特殊 I/O 端口在编译时将被指定到 FPGA 的端口上，它与标准的硬件协仿真端口是不同的，是通过特殊的 non-memory mapped gateway 模块来实现的。non-memory mapped gateway 模块在 Simulink 中常用来描述被包括在 Simulink 子系统或库中的器件，包括 LED、按钮以及外部存储器等。

当 System Generator 将板级 I/O 模块编译成硬件代码时，I/O 端口将通过外部器件接口模块连接到 FPGA 管脚上。常见的接口模块如图 8-41 所示。

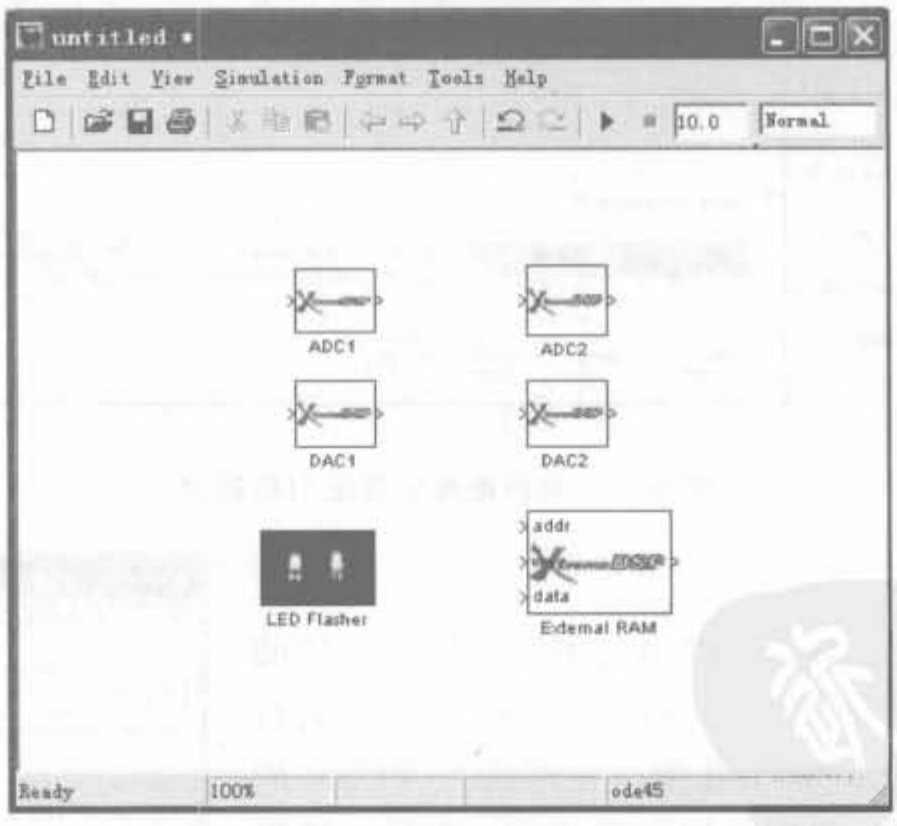


图 8-41　协仿真模块板级 I/O 接口示意图

注意，如果使用了“Gateway In”或“Gateway Out”模块，在编译时，不会将信号连接到 FPGA 管脚上，而是连接到相应的 Simulink 端口上。

5. 硬件协仿真示例

本章 8.7 节将结合 FIR 滤波器给出硬件协仿真的具体示例，这里就不再介绍。

8.4.3　共享存储器的操作

System Generator 硬件协仿真接口允许将共享存储器以及相关模块(共享 FIFO 以及共享寄存器等)编译到 FPGA 中完成协仿真。这些接口使得基于硬件的共享存储资源映射到 PC 的通用地址空间。共享存储单元有助于在 PC 和 FPGA 之间完成高速的数据传输，以及实时的硬件协仿真。本节以共享 FIFO 模块为例，介绍共享存储器在协仿真中的使用方法。其余类型共享存储器的操作步骤是类似的。

1. 编译共享存储器

包含共享存储器设计的硬件协仿真和普通协仿真设计的编译过程是一样的，选择编译目标，并在“System Generator”对话框上单击“Generate”按钮。在协仿真中被编译的共享存储器都会在硬件中通过 IP Core 或者 HDL 单元来实现。表 8-11 给出了共享存储单元和硬件实现的映射关系。

表 8-11　共享存储器模块列表

To Block	From Block	硬件实现说明
Shared Memory	Shared Memory	Dual Port Block Memory 6.1
To FIFO	To FIFO	Fifo Generator 2.1
To Register	To Register	Synth_reg_w_init.(vhd,v)

2. 含有 FIFO 的协仿真

在硬件协仿真中，可以生成 To FIFO 模块、From FIFO 模块以及共享 FIFO 对模块。其中，共享 FIFO 对模块由一个命名相同的 To FIFO 模块和 From FIFO 模块组成，To/From FIFO 可通过 Core Generator 配置成独立时钟。

异步 FIFO 在多时钟应用中是最安全的交换时钟的方法。如果选择自由运行时钟模式，则 FPGA 和 Simulink 仿真是异步的，也就是二者没有保持同步锁相，必然会建立两个时钟域：Simulink 仿真时钟域和 FPGA 自由运行时钟域。此时，共享 FIFO 提供了最可信、最安全的方式来完成 PC 和 FPGA 芯片的数据交互。此外，共享 FIFO 还在协仿真期间支持突发传送数据的能力，使得单次传送一个矢量或数据帧成为可能。

在协仿真中生成一个共享 FIFO 对，则仿真中的两个 FIFO 模块会被一个独立的异步 FIFO 代替。如图 8-42 所示，FIFO 的读、写端全部连接到用户设计逻辑(用户逻辑由原始 Simulink 设计驱动)。正因为如此，PC 端不能在设计中控制 FIFO。

从图 8-42 中可以看出，虽然异步 FIFO 的两个时钟全部预留了，但是其时钟源来源于同一信号，和多时钟域的工作还是不同。单个共享 FIFO 和共享 FIFO 对的使用是有所不同的。FIFO 的一端连接到 PC 接口逻辑上，另一端连接到原始的 To/From FIFO 模块上，这样，控制逻辑就分布在 PC 和 FPGA 之间。如图 8-43 所示，当一个 To FIFO 模块编译成硬件后，FIFO 的写控制端连接到用户逻辑上，读控制端连接到 PC 接口上，允许 PC 在仿真期间从 FIFO 接口读数。

如图 8-44 所示的应用中，FIFO 的读控制端连接到用户设计中，写控制端被连接到 PC 接口上，允许 PC 在仿真期间向 FIFO 中写数。

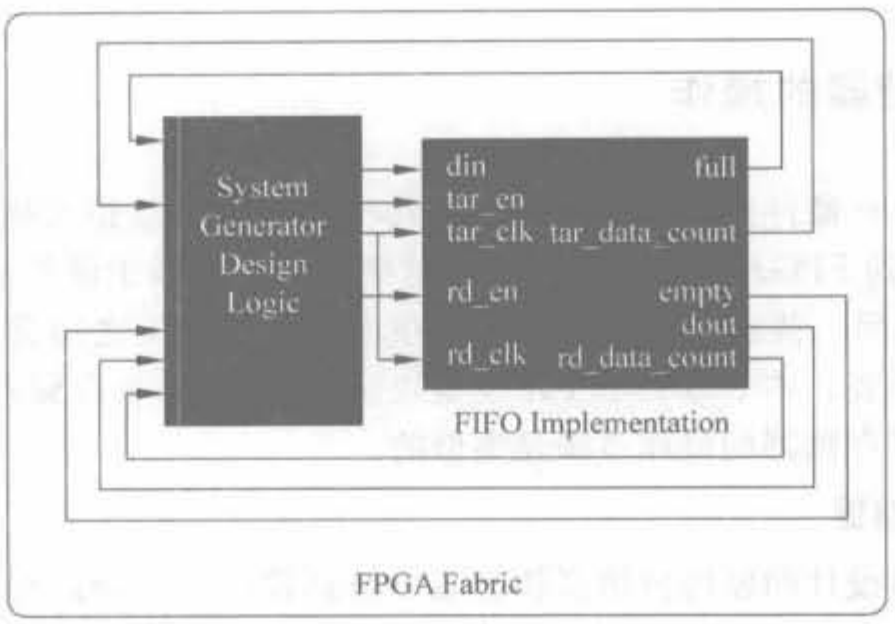


图 8-42　共享 FIFO 对的典型连接示意图

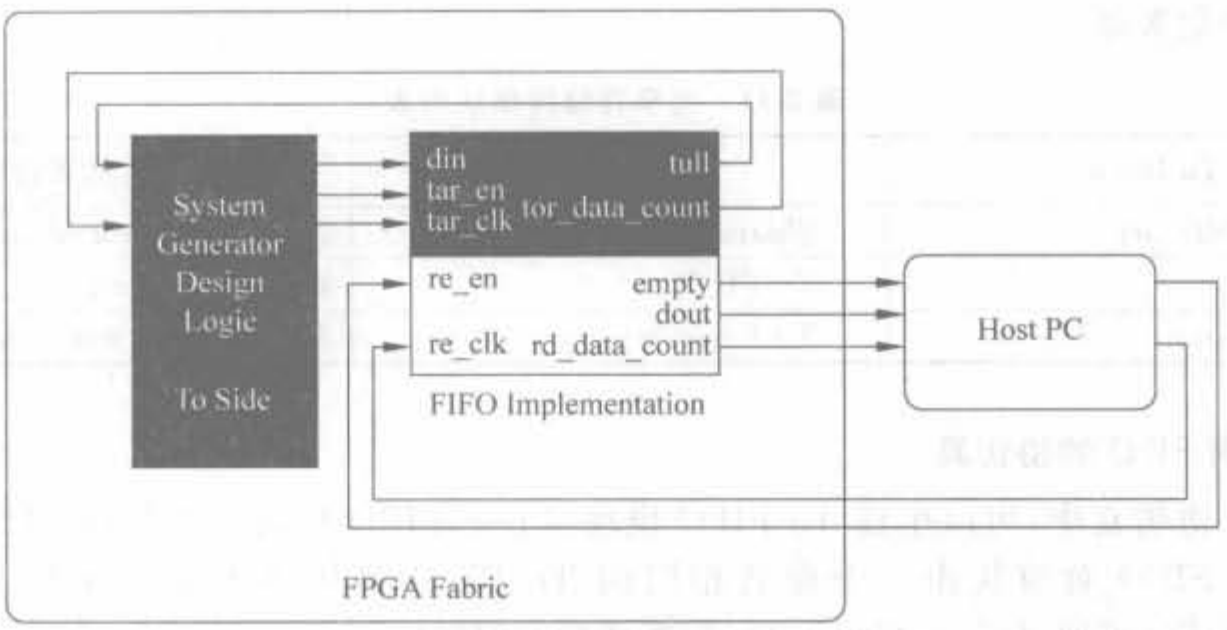


图 8-43　单个写 FIFO 的典型连接示意图

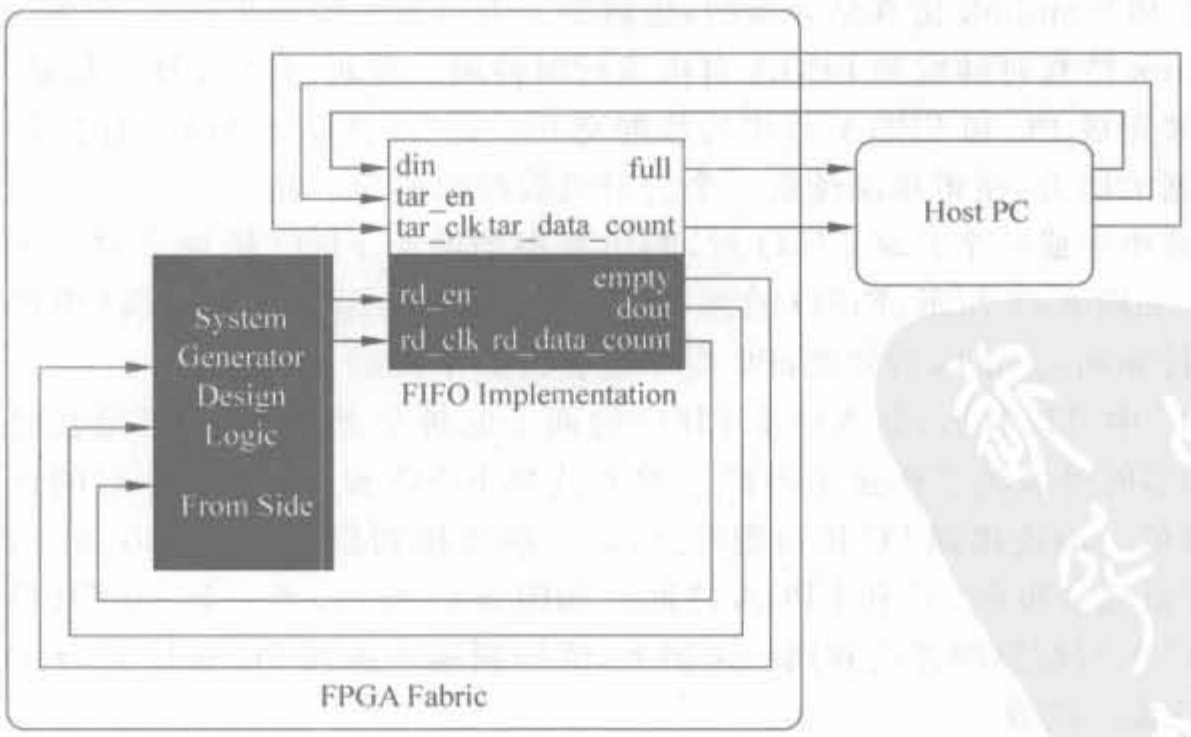


图 8-44　单个读 FIFO 的典型连接示意图

在硬件协仿真系统中，共享 FIFO 对经常分布于 Simulink 软件和 FPGA 硬件两端，也就是说，一部分在 FPGA 中实现，另一部分通过 To/From FIFO 在软件中模拟，将软件和硬件合起来就构成了一个完整的异步 FIFO。如果一个软件/硬件共享 FIFO 对被仿真，则 System Generator 会有效管理 PC 和 FPGA 之间的必要传输。在仿真中，当数据被写入软件端的 To FIFO 模块中时，相同的数据也会在硬件中写到 FIFO 中，硬件设计可通过读操作获得该数据。类似地，如果在硬件中，将数据写入 FIFO，则在软件端，可通过 From FIFO 模块将数据读出来。

需要注意的是，共享 FIFO 的空指示信号、满指示信号、读使能、写使能并不能很好地反映出硬件 FIFO 的状态。在硬件协仿真中，将一个软件共享 FIFO 连接到硬件共享 FIFO，最简单的方法就是通过共享 FIFO 的名字指定。

3. 共享内存的限制

在 System Generator 中使用共享内存，需要满足以下限制条件：

- 一旦编译后，共享存储器的访问特性不允许再改变；
- 共享存储器的地址数据位宽不能超过 24bit，最深的访问深度为 16 777 216 字；
- 共享存储器，包括寄存器、FIFO 等单元，其数据位宽不能超过 32bit；
- 全部的共享存储器都是通过内部的块 RAM 来实现的，目前还不支持利用分布式 RAM 以及外部存储器来实现的功能；
- 共享存储器是不能重复命名的。

8.5　System Generator 的高级应用

通过阅读 8.4 节，读者可基本掌握 System Generator 的基本用法，可开发简易的数字信号处理系统，但由于实际的开发环境是多样的，因此相应的高级设计方法会更复杂一些。本节给出一些高级应用，但更多的方法需要读者在实践中掌握。

8.5.1　导入外部的 HDL 程序模块

基于 HDL 的设计已经盛行了多年，现已积累了大量的 HDL 代码资源和丰富的 IP 核，因此在各种 FPGA 设计中，不可避免地要导入一些 HDL 代码，以达到最优设计。System Generator 支持导入 HDL 设计，能以黑盒子(Black Box)的方式导入 VHDL、Verilog HDL 以及 EDIF 设计文件。在模型设计中，黑盒子模块和普通的 System Generator 一样，能实现模块间的互相连接、参与仿真以及被编译成硬件电路。

1. 黑盒子 HDL 代码的要求

黑盒子模块对导入的 HDL 代码格式有一定的限制，其具体要求如下：

- HDL 实体的名字不能和设计中已有的模块名字重复。
- 顶层黑盒子实体中不能出现双向端口。
- 对于 Verilog 黑盒子，其模块和端口名必须小写，且命名要规范。
- HDL 模块的时钟信号以及时钟使能信号都必须是标准逻辑类型，即不能使用矢量输入信号。例如，“input clk1，clk2；”是合法的；“input [1：0] clk；”是不合法的。
- 黑盒子代码的时钟和时钟使能信号必须成对出现，即出现一个时钟信号，则必须有

一个时钟使能信号。且时钟信号的名字中必须包含字符串 clk，时钟使能信号的名字必须包含字符串 ce，一对时钟信号和时钟使能信号的名字只能有 clk 和 ce 不同，如 my_clk_1 和 my_ce_1。

2. 黑盒子配置向导

System Generator 提供了由 HDL 代码到黑盒子模块转化的可配置向导，用于简化整个流程。可配置向导首先检查 VHDL、Verilog 代码的语法，然后根据语义分析的结果将其转换成. m 函数，再将. m 函数和黑盒子模块关联起来。生成的可配置. m 函数是否可用，取决于导入的 HDL 代码的复杂度。下文会指出在哪些细节上，必须对其生成的代码进行手工修改，以修正可配置向导的不足。

1）启动可配置向导

当在设计中添加一个黑盒子模块时，配置向导会自动启动。注意，在添加黑盒子之前，要将导入的 HDL 文件放在设计模型的文件夹中。可配置向导只在设计文件. mdl 所在的文件中寻找. v 和. vhd 文件，如果没有找到相关文件，会弹出警告提示对话框，如图 8-45 所示。

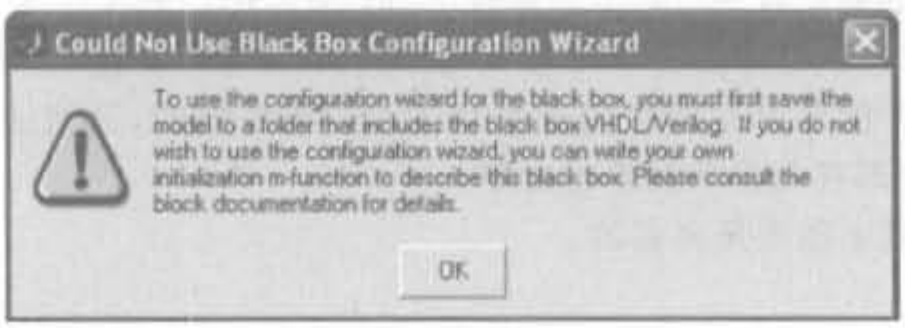


图 8-45　警告提示对话框

当找到. v 或者. vhd 文件后，可配置向导会弹出一个新的对话框，并列出所有 HDL 文件，如图 8-46 所示。选中要导入的文件，单击“Open”按钮即可完成整个流程。自动生成的. m 文件的名字为“<module>_config. m”，且存放在设计文件夹中。<module>就是导入的 HDL 文件的名字。

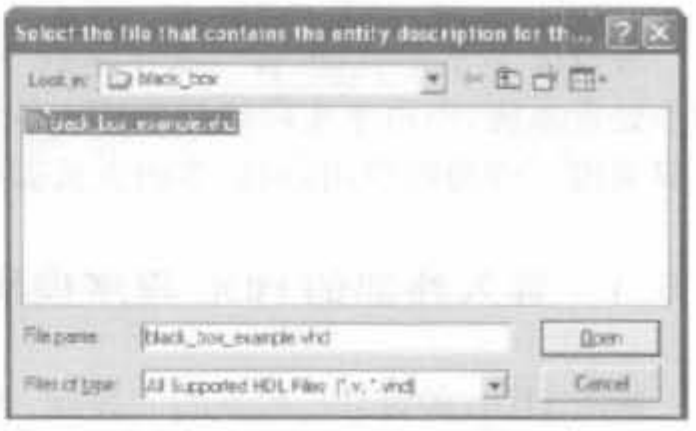


图 8-46　列出的 HDL 文件

2）可配置向导的一些细节

可配置向导在运行过程中会自动提取大量的信息，并执行相应的转换，但是有些细节必须通过手动修改才能完成，如表 8-12 所示。另外，可配置向导会在. m 函数相应的地方添加注释以提示设计人员。

表 8-12　向导细节描述列表

细节编号	细节描述
1	如果模型设计中存在组合逻辑路径，则必须调用模块 BlockDescriptor 对象的 tagAsCombinational 方法
2	可配置向导只能识别导入的顶层模块，与顶层模块相关的子模块必须手动启动. m 函数中添加文件的方法来一一添加
3	可配置向导生成的模块是单速率的，即所有端口的速率都是一样的。但实际中，各个端口的速率往往是不一样的，需要手动修改

3. 可配置.m 函数

导入的 HDL 文件都是以黑盒子模块来描述的，因此原 HDL 文件所有的信息都通过可配置.m 函数加载到黑盒子中。可配置.m 函数不仅定义了接口、物理实现以及仿真行为等信息，还包括以下配置信息：顶层模块的实体名字、VHDL 或 Verilog 语言选择标志、端口描述、模块的一般性需求、时钟和采样速率、和模块有关的所有文件信息以及模块中是否含有组合逻辑路径。

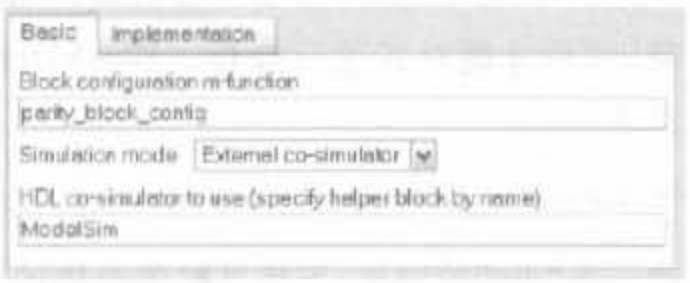


图 8-47　设定关联性的对话框

黑盒子和可配置.m 函数的关联性在其属性对话框中设定，如图 8-47 所示。可配置.m 函数采用面向对象的接口来说明黑盒子模块的信息，主要由模块描述符对象 BlockDescriptor 和端口描述符对象 PortDescriptor 来完成。当 System Generator 调用一个可配置.m 函数时，就向相应的.m 函数传递一个模块描述符：

```
function sample_block_config(this_block)
```

该模块描述符对象提供了说明黑盒子信息的方法，而相应的端口信息由端口描述符对象独立定义。

1）设置顶层实体

配置向导会自动添加该属性的说明。BlockDescriptor 对象提供的设计方法为 setEntityName，其设置顶层实体的语法为

```
this_block.setEntityName('foo');
```

2）语言选择

黑盒子模块的 BlockDescriptor 对象提供了设置顶层模块语言类型的函数，该方法只能在可配置.m 函数中调用一次。下面给出设置 VHDL 以及 Verilog 模块的语法。在操作中，向导会在.m 函数自动添加相应的代码。

```
VHDL Module:
    this_block.setTopLevelLanguage('VHDL');
Verilog Module:
    this_block.setTopLevelLanguage('Verilog');
```

3）端口定义

黑盒子模块的端口定义也是由可配置.m 函数通过调用端口描述符来定义的。端口描述符提供了配置端口属性、位宽、数据类型以及采样速率的方法，主要的操作包括添加新的端口、获取端口对象、配置端口类型、设置端口采样率以及动态输出端口。

（1）添加端口

在模型设计中，黑盒子输入、输出端口的添加由模块描述符对象的 addSimulinkInport 函数和 addSimulinkOutport 函数完成，其语法为

```
添加输入端口:
    this_block.addSimulinkInport('din');
添加输出端口:
    this_block.addSimulinkOutport('dout');
```

其中的字符参数就是端口名字，必须小写，且要和导入的 HDL 模块中的名字一致，否则会出错。也可以通过调用 setSimulinkPorts()函数来实现输入、输出端口的添加。该函数有两个输入参数，第一个参数是输入端口组成的胞元数组，第二个参数是由输出端口组成的胞元数组。

(2) 获取端口对象

定义了端口之后，需要配置其属性，但是在配置属性之前必须先获得相应端口的描述符对象，然后才能对其进行操作。BlockDescriptor 对象提供了 port()函数来获取对象，其使用语法为

```
din = this_block.port('din');
```

创建一个“din”的对象，再将其赋值给描述符变量。此外，还有两个函数 inport()和 outport()用来返回对象的端口下标，其类型为 1 到所有输入、输出端口总数之间的一个整数，方便了对模块端口的反复性操作。

(3) 配置端口类型

PortDescriptor 提供了完备的端口类型的配置方法，如一个名为“dout”、位宽为 12 以及位置下标为 8 的无符号端口的定义语法为

```
dout = this_block.port('dout');
dout.setWidth(12);
dout.setBinPt(8);
dout.makeUnsigned();
```

也可以通过下面的语句完成相同的功能：

```
dout = this_block.port('dout');
dout.setType('Ufix_12_8');
```

黑盒子支持具有单比特端口(如 std_logic)或向量端口(如 std_logic_vector(0 downto 0))的 HDL 代码，但对于二者的处理略有不同。System Generator 默认端口为向量端口，因此当端口为单比特时，需要通过方法 useHDLVector()来修改其默认配置。当其参数为真时，意味着向量端口；否则为单比特端口，其语法为：

```
dout.useHDLVector(true); % std_logic_vector
dout.useHDLVector(false); % std_logic
```

(4) 设置端口采样率

可以通过黑盒子设置不同的采样速率。在默认情况下，输出端口的采样速率和输入端口是一样的；但在多速率系统中，输出端口的速率和输入端口的速率是不一样的。另外在通常情况下，各个输入端口的速率也是不一样的，这就需要为各个端口独立设置采样率。PortDescriptor 类提供了 setrate()的方法，用于独立设置端口速率，其语法为

```
dout.setRate(N);
```

其中，N 为正整数，意味着采样周期包含 N 个系统周期，也就是说，其采样速率为系统时钟的 $1/N$。如果系统周期为 2s，则“dout.setRate(3);”就表示采样周期为 $2\times3=6$s。如果端口的采样频率不是一个常数，是变化的，可通过 PortDescriptor 类的 setConstant()方法来实现。

(5) 动态输出端口

黑盒子最有用的一个特征就是输出端口的类型和采样速率可以动态配置。例如,根据输入端口的位宽来动态调整输出端口位宽,是实际工程中所需要的。PortDescriptor 提供了设定端口动态可配置的成员变量,通过监测输入端口动态修改成员变量来达到目的。

首先,通过.port()函数获得输入端口的位宽和速率:

```
input_width = this_block.port('din').width;
input_rate = this_block.port('din').rate;
```

其次,BlockDescriptor 对象提供了布尔型成员变量 inputTypesKnown 和 inputRatesKnown 来判断端口类型和位宽配置信息是否被传递到模块。

最后,在成员变量 inputTypesKnown 和 inputRatesKnown 的判断条件语句中嵌套对输出端口的配置来达到动态输出端口的目的。动态设置输入端口位宽的语句如下:

```
if (this_block.inputTypesKnown)
  dout.setWidth(this_block.port('din').width);
end
```

动态设置输入端口采样速率的语句与上类似,其语法为:

```
if (this_block.inputRatesKnown)
  dout.setRate(this_block.port('din').rate * 2);
end
```

4) 黑盒子时钟

对于导入的多速率代码,必须在可配置.m 函数中对系统时钟进行相应的说明。无论是单速率系统还是多速率系统,System Generator 要求时钟和时钟使能信号必须成对出现。BlockDescriptor 对象提供了.addClkCEPair()函数,用于定义黑盒子的时钟和时钟使能信号。该函数有 3 个参数,第 1 个参数是 HDL 代码中的时钟信号名字,第 2 个参数是 HDL 代码中的时钟使能信号,第 3 个参数定义时钟信号和时钟使能信号之间的速率关系。其中,时钟端口信号必须包含字符串"clk",时钟使能信号必须包含字符串"ce"。除了"clk"和"ce"外,二者其余部分必须相同。速率参数不是 Simulink 模块的采样速率,它为一正整数,代表了系统时钟速率和相应的时钟使能速率的比值。

例如,假设系统时钟使能信号为 ce_3,时钟信号为 clk_3,且 ce_3 的周期是 clk_3 周期的 3 倍,则可以通过下面的语句完成说明:

```
addClkCEPair('clk_3','ce_3',3);
```

当 System Generator 将黑盒子编译成硬件电路时,能根据.m 函数生成合适的时钟有效信号,并自动将其连到相应的驱动端口上。

5) 模块中是否含有组合逻辑路径

由于输入端口的任何改变都会导致无时钟驱动的输出端口发生变化,因此如果导入的模型含有组合路径,则必须通过 tagAsCombinational 方法在.m 函数中进行说明,其语法为:

```
this_block.tagAsCombinational;
```

8.5.2 设计在线调试

System Generator 提供了使用 ChipScope 和共享存储器两种方法来实现系统的在线调试，它们都可以利用相关工具获得可视化的测试信号，都支持十六进制、十进制以及二进制数，允许在模型中比较参考信号，且支持逻辑值和模拟量两种信号显示方式。

使用 ChipScope 完成在线调试，需要在模型中插入 ChipScope 模块，在 FPGA 芯片中插入 ChipScope 探头，通过 JTAG 接口实时地完成芯片内部信号的采集。如果使用共享存储器的方式，则需要由 FPGA 硬件、Simulink 设计以及 MATLAB 软件共同提供一个独立的共享地址空间，将 FPGA 芯片中的数据实时地存储到空间中，然后再读回 Simulink 设计，可简化整个在线调试的数据分析。

本节以 ChipScope 法为例介绍如何完成设计的在线调试。

例 8-4 利用 System Generator 实现一个正弦波信号发生器，并通过 ChipScope 在 FPGA 芯片中实时分析。

(1) 在 MATLAB 中启动 Simulink，建立设计模型。从 Xilinx 模块库中添加 Counter 模块、SineCosine 模块、ChipScope 模块、Gateway Out 模块、System Generator 模块以及 Scope 模块，其连接关系如图 8-48 所示，并保存为 mychipscope. mdl。

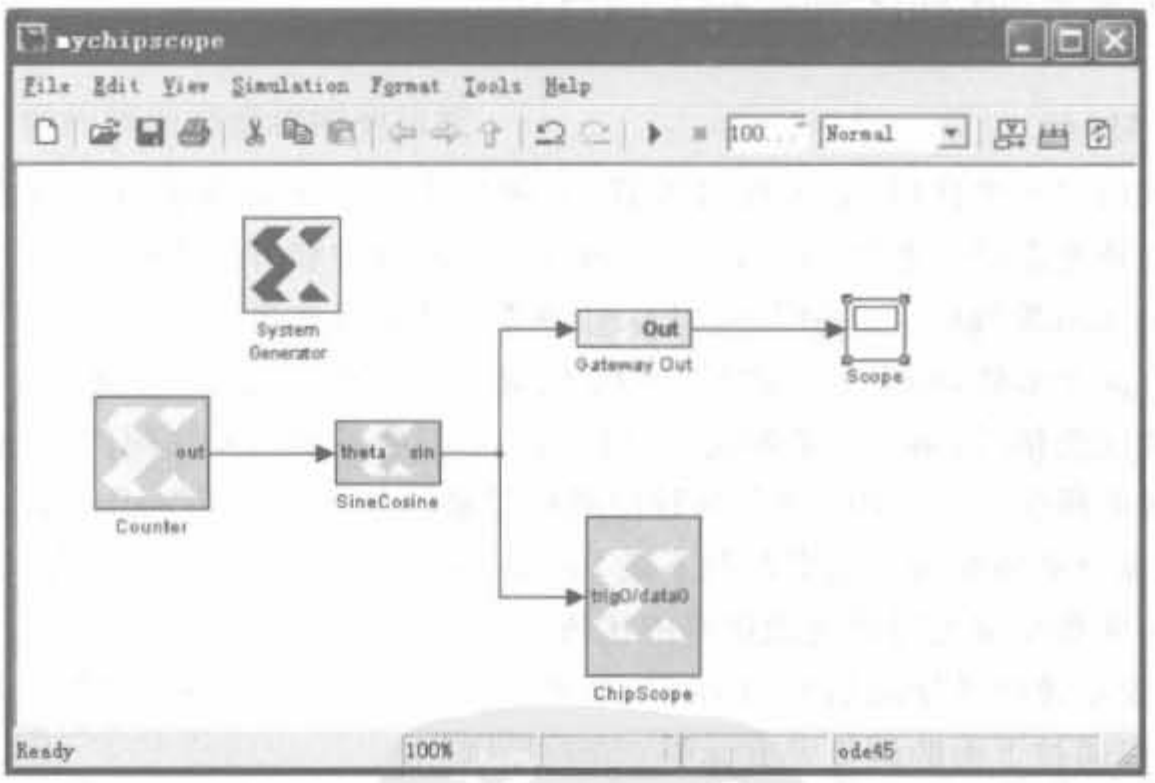


图 8-48 在线调试的正弦信号发生器设计

(2) 双击“SineCosine”模块，选定“Function”为“Sine”；双击“Counter”模块，设置步长为 1。

(3) 双击“ChipScope”模块，在“Triggers”栏中选择“Number of trigger ports”为“1”；在“Triggers Setting”栏中设定“Match type”为“Extended”；将“Nunber of data ports”设定为“1”，且选中“Use trigger ports as data”选项。

(4) 双击“Gateway Out”模块，选中“Specify IOB location constraints”选项，在“IOB pad location(cell array {'MSB',…,'LSB'})”中输入“{'F12','E12','E11','F11','C11','D11',

'E9','F9'}"。

(5) 在 Simulink 中单击"运行"按钮开始仿真，结果如图 8-49 所示。由于 Counter 的步长设置得比较小，所以正弦信号的频率较低，需要将仿真时间设置得长一些。

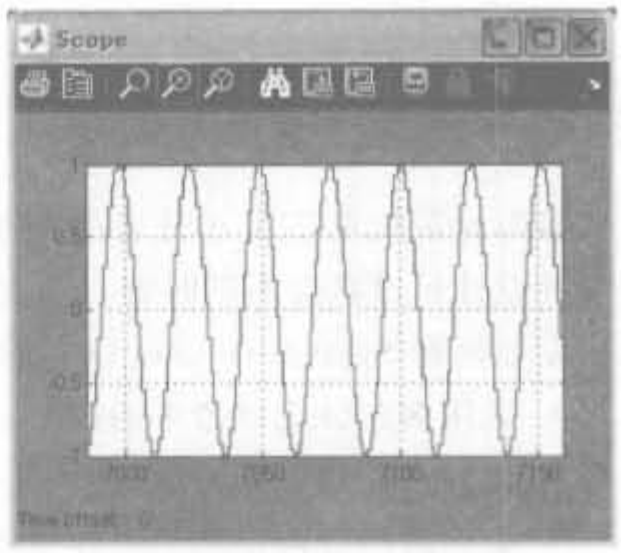


图 8-49 正弦波示例模块的 Simulink 设计示意图

(6) 双击 System Generator 模块，在"Compilation"下拉框中选择"Bitstream"；"Part"为 Spartan-3E 系列的 XC3S500E-fg320，时钟周期设为 20ns (50MHz)，时钟管脚为 C9。完成上述设置后，单击"Generator"按钮，生成相关文件。由于选择了编译生成比特文件，因此完成后，可在设计文件夹中找到 mychipscope_cw. bit 的配置文件。

(7) 从"开始"菜单中打开 ChipScope Pro，单击 JTAG 链扫描按钮，选择"XC3S500E"，按照 6.5.2 节的方法下载程序，组合总线信号，可得到观测波形如图 8-50 所示。

图 8-50 正弦波示例模块 ChipScope 采集结果的数字显示

为了更直观地观察波形，可通过"Bus Plot"功能得到其模拟波形，如图 8-51 所示。可以看出，本设计正确地实现了预期功能。

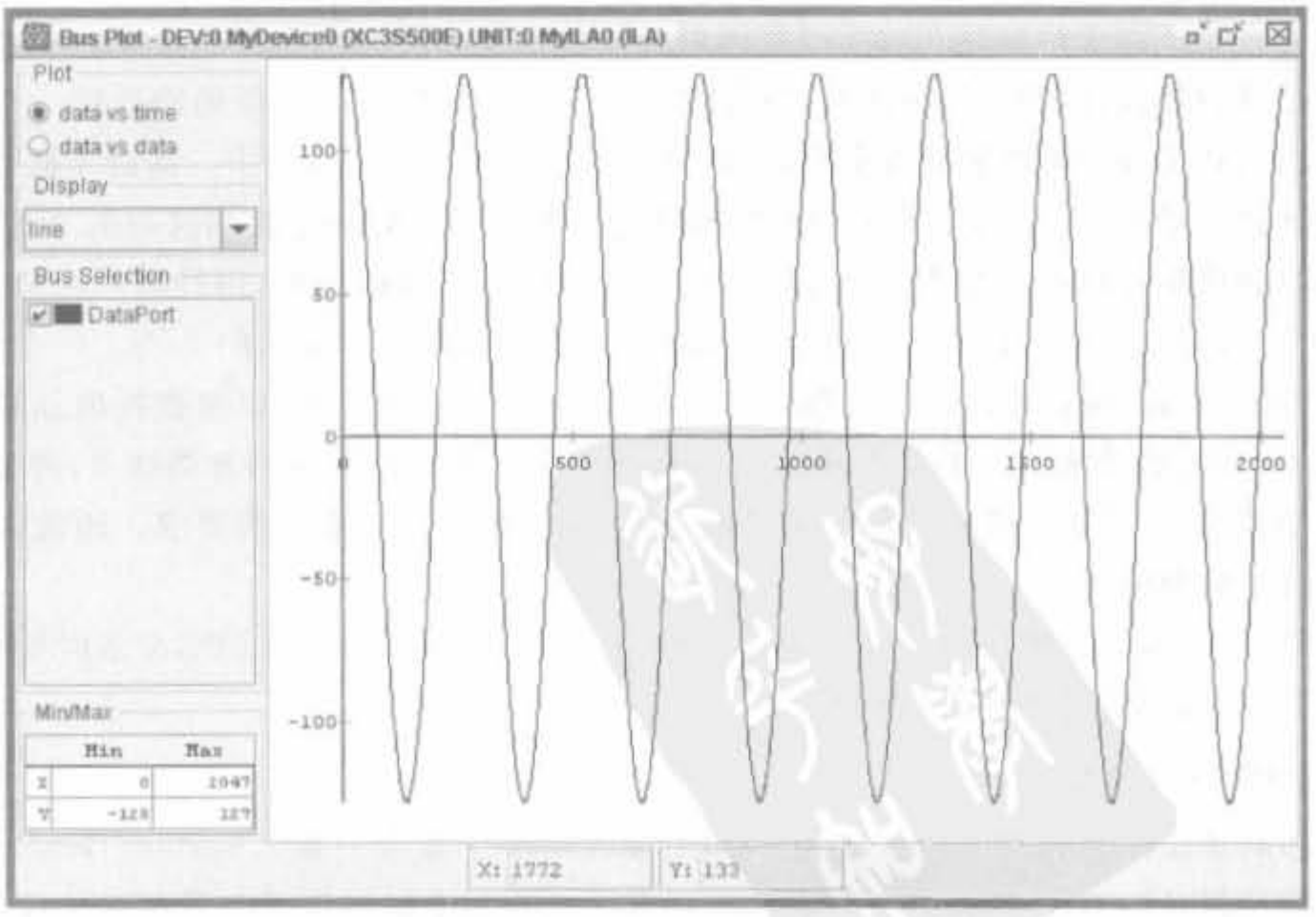


图 8-51 正弦波示例模块 ChipScope 采集结果的模拟显示

8.5.3 系统中的多时钟设计

多时钟系统在FPGA开发中占据重要地位。首先,存储器共享技术需要多时钟;其次,多时钟设计可以联合不同时钟和分支时钟来达到实现高级时钟的策略。本节主要从以下方面讨论如何在System Generator中使用多时钟开发技术,并在最后给出一个工程实例:

- 采用多时钟设计带来的好处;
- 如何采用层次划分法,将一个System Generator模型分割为两个或多个时钟区域;
- 如何跨时钟域使用共享存储器;
- 如何仿真多时钟设计以及将其转化成网表;
- 如何使用多个Xilinx子系统模块将多个时钟区域级联起来。

将设计划分为由多个不同时钟驱动的子系统,每个子系统被称为异步时钟孤岛,其中的共享内存将起到通信桥梁的作用。

1. 多时钟应用

多时钟区域的典型应用是和工作在不同时钟速率的外部硬件模块接口。例如,经常通过一组I/O接口,按照外部控制器的时钟来访问外部寄存器,然后在FPGA内部对数据进行重采样,以达到同步的目的;经过本地处理后,将数据以另外一个外部接口模块所需要的时钟送出FPGA。

多时钟的另外一个重要应用就是开发高速处理单元。以插值FIR滤波器为例,滤波器从外部模块单元接收符号数据,然后对每一个符号完成4倍插值,将每个符号变成4个符号,即数据速率提高了4倍,最后以数/模转换器(DAC)时钟采样数据,将其送到DAC中。

对于滤波器设计,常用的方法就是让滤波器工作在采样率下,一个周期处理一个采样,这种方法消耗了最大限度的硬件资源,因此只适合应用于高速的系统中。而对于低速数据,这样设计就太浪费资源了,需要采用复用的理念。假设工作时钟是数据速率的N倍,则一个处理时钟周期可以拆分为N个片段,拆分后的时长等于数据周期,则处理一级只需要一个时间片段,可以将同一份资源复用N次,即资源的使用降低为原来的$1/N$。一个优秀的FPGA设计应该尽可能采用高的处理时钟,以追求更多的复用倍数,以速度换取面积,降低硬件成本,同时给系统引入了多个时钟。当然,多倍复用的方法存在两方面缺点,首先,会导致芯片功耗变大;其次,增加了布局布线的难度,对编程人员有更高的要求。因此,需要在高速设计中划分时钟域。

此外,在异步接口模块中不可避免地要使用多时钟。因此,掌握FPGA多时钟的设计方法既是一种趋势,也是一项基本要求。

2. 时钟区域分割

将多时钟设计分割到多个区域是FPGA设计的一个重要方面。System Generator不仅以结构化设计的方式支持时钟区域分割,还要求把与每个时钟以及与其相关的逻辑封装成一个子系统。这样,单个子系统就形成了同步孤岛,是真正意义上的周期硬件电路,且其

性能和 Simulink 行为模型一致。前面所提的比特和周期精确的概念只在单个同步孤岛中成立。由于包含同步孤岛的高层模型是通过异步时钟来驱动孤岛的，因此它并不是真正的周期电路。需要说明的一点是，System Generator 和 Simulink 都通过理想的时钟源对设计进行仿真，因此调用异步时钟系统会导致软件仿真和真实的硬件性能存在一定的偏差。

在设计中通过子系统来分割时钟有诸多优点：首先，将物理时钟信号从模块结构图中提取出来；其次，Xilinx 提供的跨域转换器能够处理模块库中模块的亚稳态安全；第三，System Generator 能对时钟孤岛产生精确的时钟约束以及时序分析。

此外，这种抽象级的设计流程可以减少用户设计中的一些常见错误，如门时钟错误、异步复位信号以及锁存器。System Generator 在硬件生成阶段自动产生时钟，因此不会将非时钟信号连到时钟输入端口，发生门时钟错误，也不会生成锁存器。由于异步复位会导致各种时序问题，因此 System Generator 在编译时不允许异步复位，从而避免了该类错误。

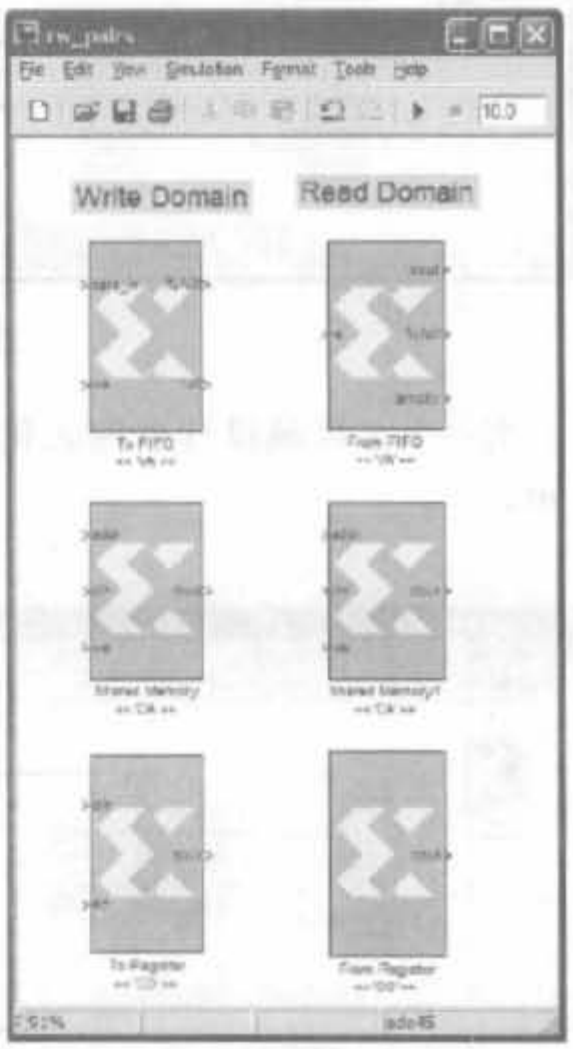


图 8-52　成对使用的模块示意图

3. 跨越时钟区域

System Generator 通过存储器共享的方式来跨越不同的时钟区域，主要有 Shared Memory、To FIFO、From FIFO、To Register 和 From Register 这 5 个模块，且在使用时必须配对，典型的示意图如图 8-52 所示。

To FIFO 模块用于写操作，From FIFO 用于读操作，且一对 FIFO 通过命名配对关联。在硬件实现时，这两个模块都通过 Xilinx 公司的 FIFO IP Core 生成。使用 FIFO 模块是 3 种跨越时钟区域的方法中最安全和最方便的方式，是高速数据传输的最优手段。

一对共享存储器模块是通过一个双口块 RAM 来实现的，也通过命名来配对关联，两个共享存储器模块分别工作于不同的时钟区域。由于块 RAM 是真正的双口 RAM，在每个时钟区域都可以独立地完成读写，但是要注意不能同时写同一个地址，也不能对同一个地址发生写操作和读操作。由于是基于块 RAM 实现的，因此它具备大容量和高速的特点。

To Register 模块用于写寄存器，From Register 用于读寄存器，一对寄存器通过命名配对关联，寄存器操作不同的时钟域中必须同步操作。在硬件实现时，一对寄存器通过一个触发器来实现，一对 1bit 的 To/From Register 模块需要一个触发器。需要注意的是，通过这种方式实现跨域是不安全的，只有在对硬件特征完全理解的基础上才能使用，一般不推荐这种方式。

4. 网表化多时钟设计

在 System Generator 设计中，每个时钟区域都有其自己的子系统。图 8-53 给出了一个双时钟区域的设计，每个子系统都可以独立设置系统周期。

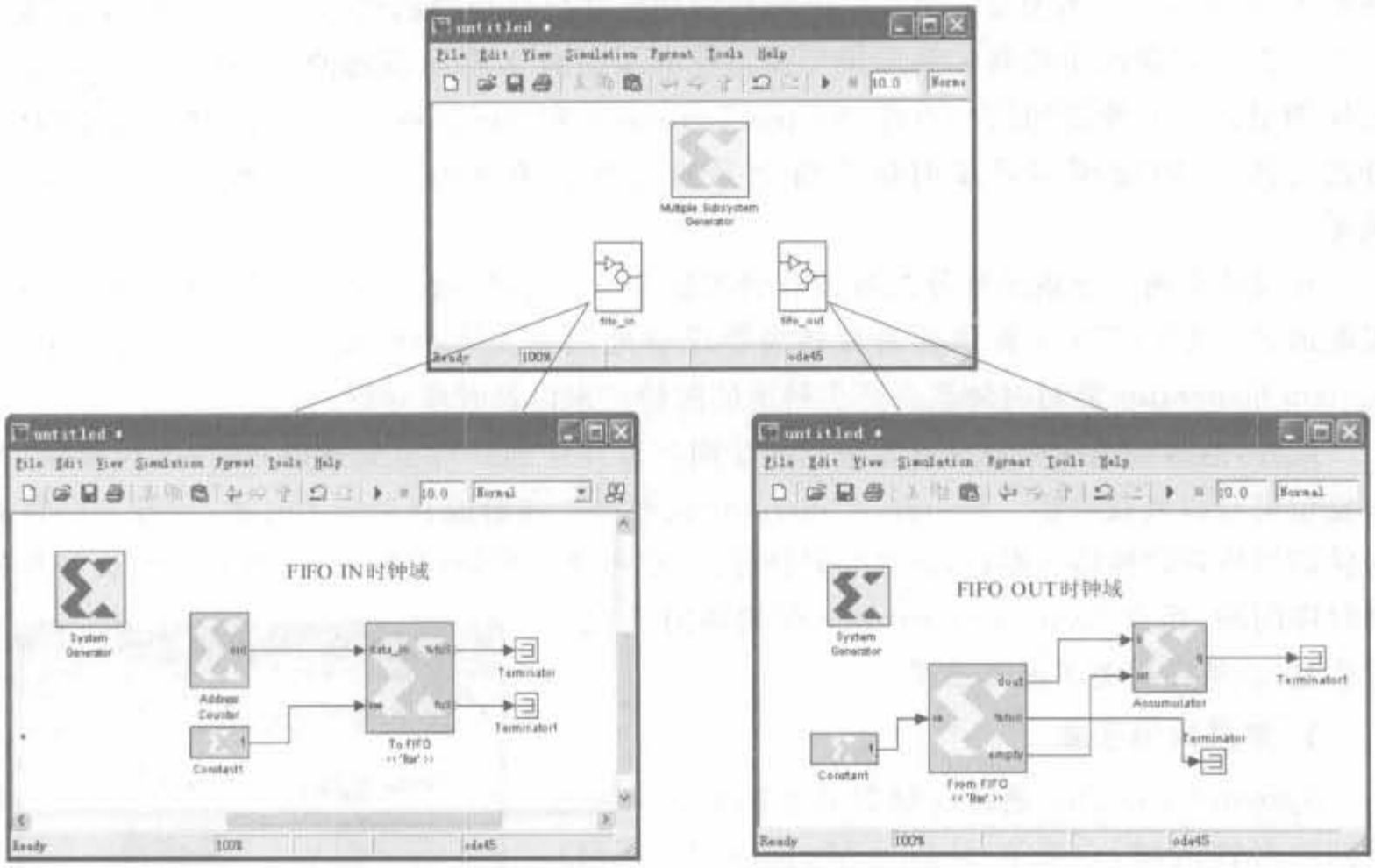


图 8-53　双时钟区域的设计

当一个多区域设计被网表化时，System Generator 需要执行下面的步骤，如图 8-54 所示。

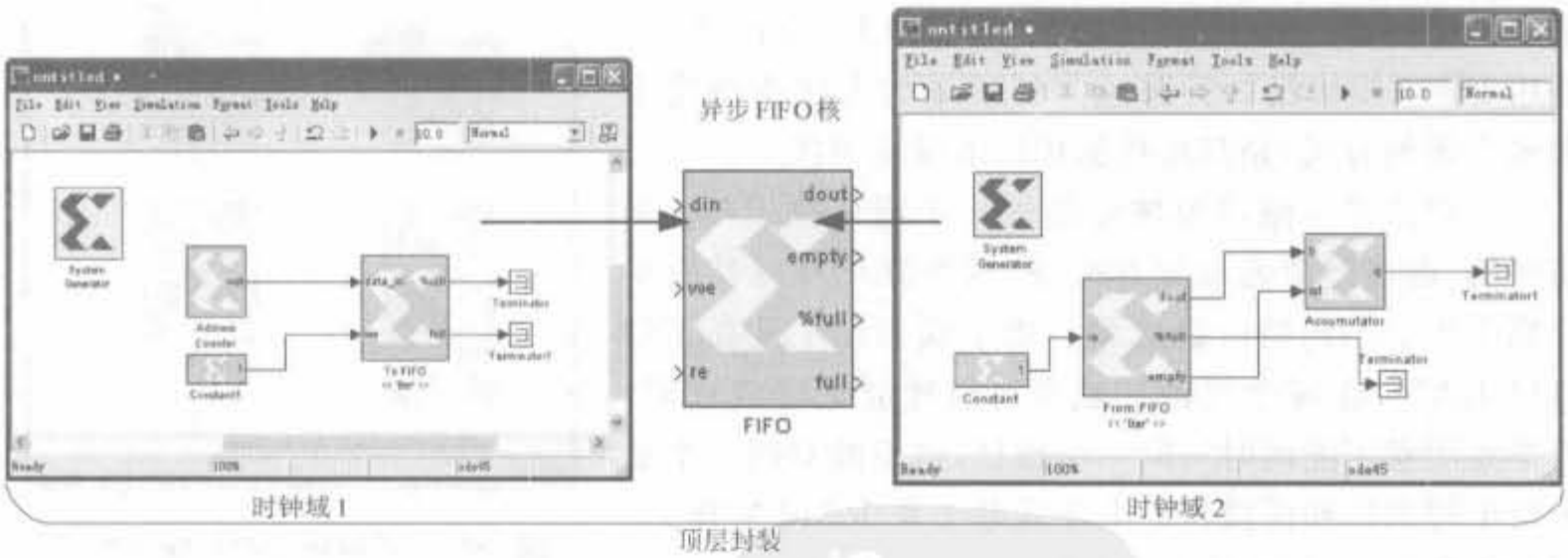


图 8-54　多时钟设计的顶层模块示意图

(1) 选中 0 号区域(左边)，除 To FIFO 模块外所有设计的 HDL 文件，并编译产生 NGC 文件。

(2) 选中 1 号区域(右边)，除 From FIFO 模块外所有设计的 HDL 文件，并编译产生 NGC 文件。

(3) 调用 Xilinx 核生成器来产生中间的 FIFO 模块。

(4) 产生顶层 HDL 模块，并在其中例化上述 3 个模块。

8.5.4　软、硬件联合开发

1. 软、硬件联合开发综述

软、硬件联合开发主要是在 FPGA 设计中添加微处理器，以 SOC 的模式完成设计。System Generator 提供了 3 类能够导入处理器的模块，即黑盒子模块、PicoBlaze 微控制器模块以及 EDK 处理器模块。

由 8.5.1 节可以知道，黑盒子模块可以导入外部的 HDL 设计，因此它提供了最灵活的设计方式，可以导入任何处理器的 HDL 代码。所有的总线和接口可根据需要在 System Generator 模型设计中灵活连接，具备对处理器的完全控制能力。

PicoBlaze 模块是最容易使用的，当然其灵活性也最小。Xilinx 公司的 PicoBlaze 微控制器模块利用 PicoBlaze 宏实现了一个 8 位的嵌入式控制器，通常只需要一个块 RAM 来存储程序。设计人员可以通过 PicoBlaze 汇编语言完成设计应用。

EDK 处理器模块提供了 MicroBlaze 处理器的接口，允许设计人员将共享存储器和它关联起来。一旦建立关联，MicroBlaze 就可以对它实现读写控制。同样，也可以将 System Generator 设计通过 EDK 处理器接口导入到 EDK 工程中。导出过程会产生一个基于 FSL 的 IP Core，能被添加到任何支持 FSL 接口的 EDK 系统中。软核 MicroBlaze 和硬核 PowerPC 都具有 FSL 接口。

2. 对 EDK 的支持

嵌入式设计是目前最流行的设计方向，System Generator 也支持在设计中添加嵌入式处理器，通过模块“EDK Processor”将 EDK 设计导入到 System Generator 中，其主要步骤如下：

(1) 在设计中添加 EDK Processor 模块。

(2) 双击 EDK Processor 模块，在“Configure processor for”下拉框中选择“HDL netlisting”选项后，单击“Import”按钮，会自动弹出选择 EDK 工程导入向导，其界面如图 8-55 所示。用户选择 EDK 工程的路径和相关的开发板型号即可。

图 8-55　EDK 设计导入设计示意图

(3) 单击“OK”按钮，即可开始加载工程，并弹出如图 8-56 所示对话框。

图 8-56　EDK 设计导入过程提示对话框

(4) 加载完成后，在“EDK Processor”页面中可以看到 EDK 工程名、所用的处理器类型(软核的 MicroBlaze 和硬核 PowerPC 两种)，如图 8-57 所示。单击“OK”即可完成整个嵌入式系统的加载。

需要注意的是，目前的 System Generator 版本还不支持加载双核的嵌入式系统设计，嵌入式处理器的外设不能和 System Generator 设计冲突。例如，如果协仿真使用了以太网接口，则嵌入式处理器不能使用以太网接口。此外，如果 EDK 设计有修改的话，需要重新加载。

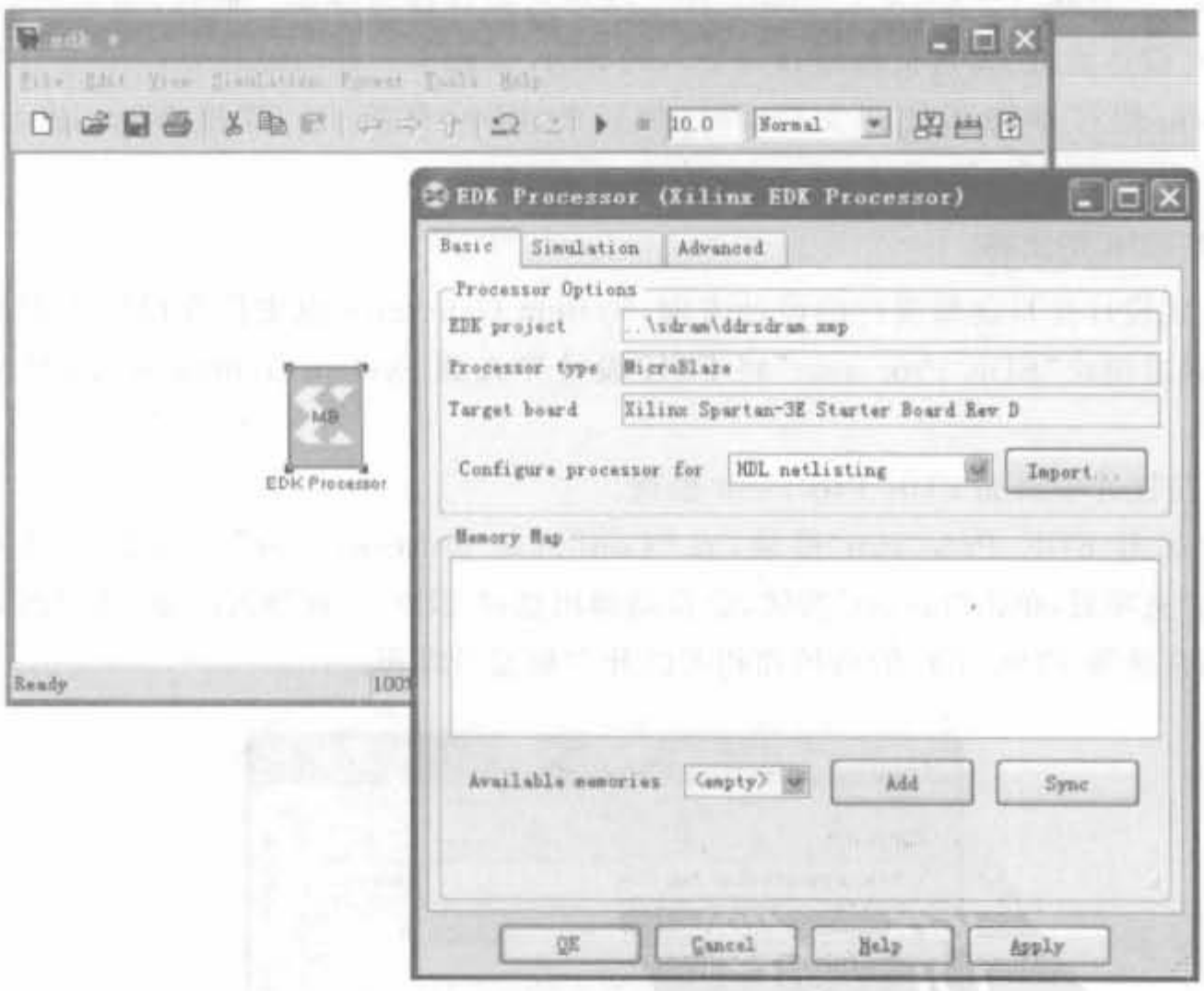


图 8-57　EDK 设计处理器类型示意图

3. 预留接口

处理器和 System Generator 之间交换数据的首选方法是通过共享存储器，但也可以通过在设计的顶层模块中预留处理器内部接口来完成通信。双击 EDK 模块，切换到“Advanced”页面，选择需要在顶层模块预留的接口，如图 8-58 所示。

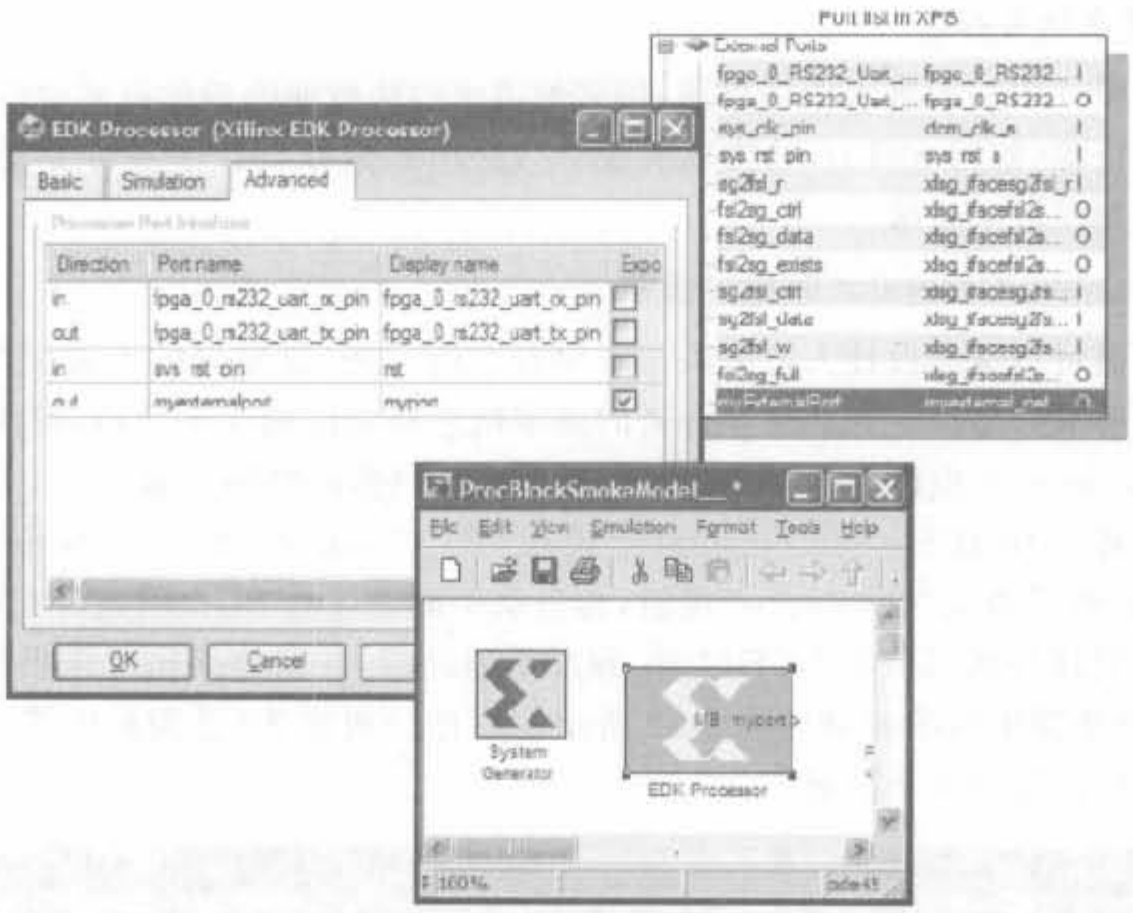


图 8-58 顶层模块预留接口示意图

其中的端口列表列出了处理器在顶层模块中所有可用的端口信号，其中不包括时钟信号以及 System Generator 用来实现存储映射的接口。选中“sys _ rst _ pin”与“myexternalport”行的“expose”选项，可在设计中直接操作 EDK 处理器中的串口。此外，用户还可根据需要对预留接口重命名，使之具有更高的可读性。

8.5.5 FPGA 设计的高级技巧

本节主要介绍一些能够使 System Generator 有效生成高性能硬件电路的方法和技巧。

1. 关注模块对话框中的注释

在基于 System Generator 设计时，需要特别注意其模块对话框中对硬件电路的说明。大多数 Xilinx 模块库中都有如何达到最高效硬件电路实现的注释，在使用时要仔细阅读和体会。例如，缩放模块(Scale block)的注释会指出该模块并不消耗任何硬件资源(因为可通过交换硬件连线实现)，而移位模块(Shift block)的注释就表明需要一定的硬件资源。对于这样的细节信息，只有从注释中才能获取。

2. 寄存设计的输入和输出

寄存输入、输出变量可以通过在“Gateway In“之后和“Gateway Out”之前加入延时为 1 的延迟模块或者寄存器模块实现。选择寄存器模块的任一属性即可添加硬件。在某些应用中，对数据需要进行双寄存才能达到目的，这可以通过例化两个独立的寄存器模块，或者例化两个延时为 1 的延迟模块实现。这样可以做到一个寄存器负责 I/O 端口，另外一个寄存器负责和 FPGA 内部逻辑接口。一个延时为 2 的寄存器模块并不能达到上述效果，这是因为它是由 SRL16 单元组成的，不能直接嵌入 I/O 端口。

3. 加入流水线寄存器

尽可能在设计中插入流水线寄存器，较多的流水线级数能有效地提高硬件实现的性能，且占用较少的资源(使用 SRL16 结构来实现)。此外，如果在设计中要将变量初始化，必须通过使用寄存器模块来完成。

4. 使用 System Generator 时序分析器

时序分析器可以帮助设计人员解决与时序有关的问题，以报告的形式反馈最慢的路径和不满足时序要求的路径。有关时序分析的基本概念和方法的详细介绍，可阅读本书第 11 章。System Generator 也提供了时序分析器来辅助用户提高电路性能。

双击打开设计中的 System Generator 模块，在“Compilation”下拉框中选择“Timing Analysis”选项，然后单击“Generator”按钮，会自动生成设计的 NGC 网表、NGD 网表文件以及 NCD 文件，时序分析器分析 NCD 文件，跟踪最差路径，并最终给出时序报告。该编译过程完成后，会自动弹出如图 8-59～图 8-62 所示的时序分析报告，分为最慢路径、图表显示、统计结果以及跟踪信息 4 个页面。

图 8-59　时序分析器最慢路径列表

1) 最慢路径分析结果

直接获取最慢路径可帮助用户快速修改设计，打破时序瓶颈。在图 8-59 中的“Source”区单击分析源之后，在“Path”区会显示该源的所有路径延迟以及延迟类型。选中右下角的“Display low-level names”选项，还可显示出相应路径在实现后详细的网表名称。

2) 图表分析结果

单击左侧栏的“Charts”图标，即可切换到图表分析结果。在一般的时序分析中，设计人

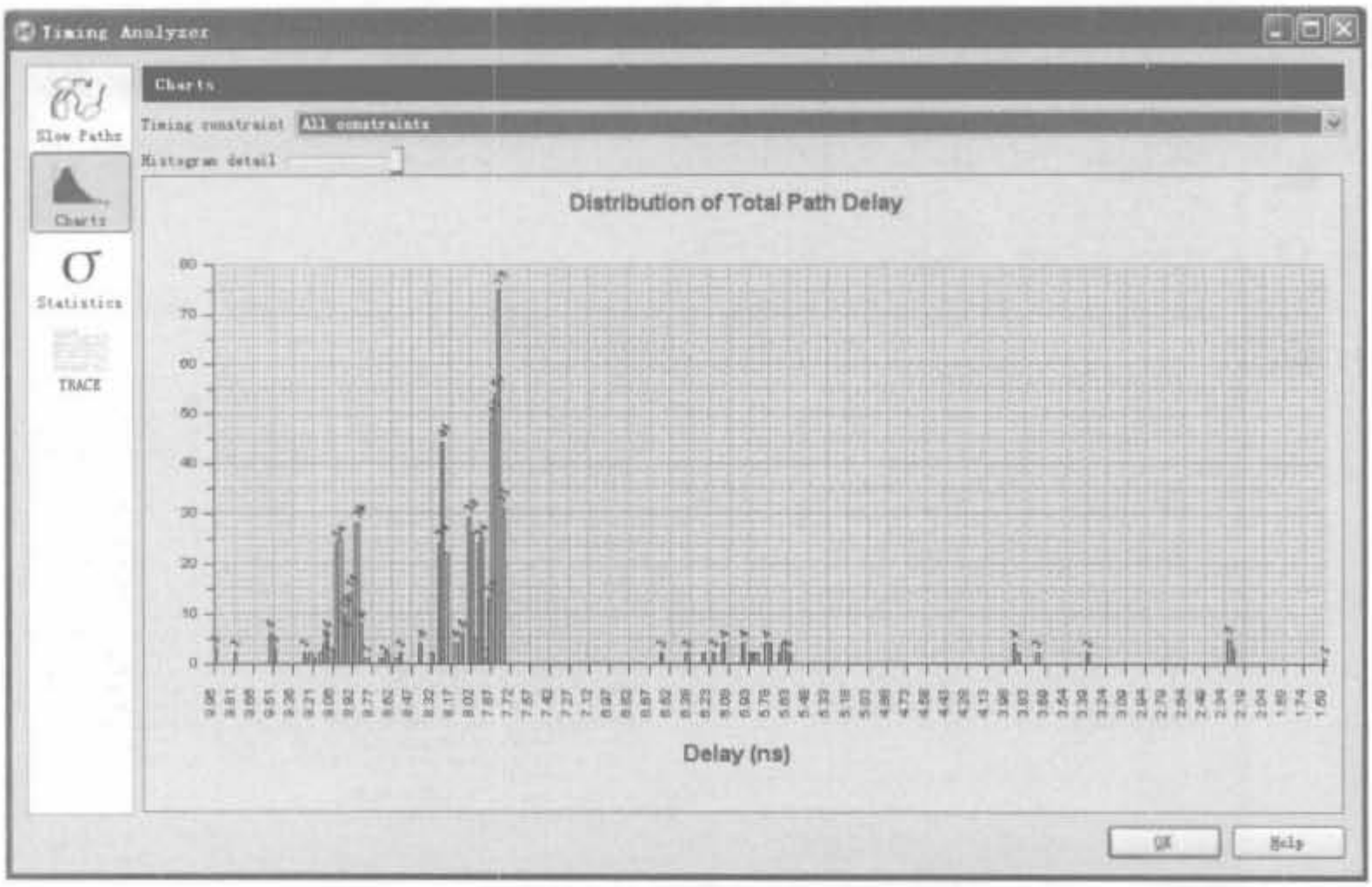


图 8-60　时序分析器性能直方图统计结果示意图

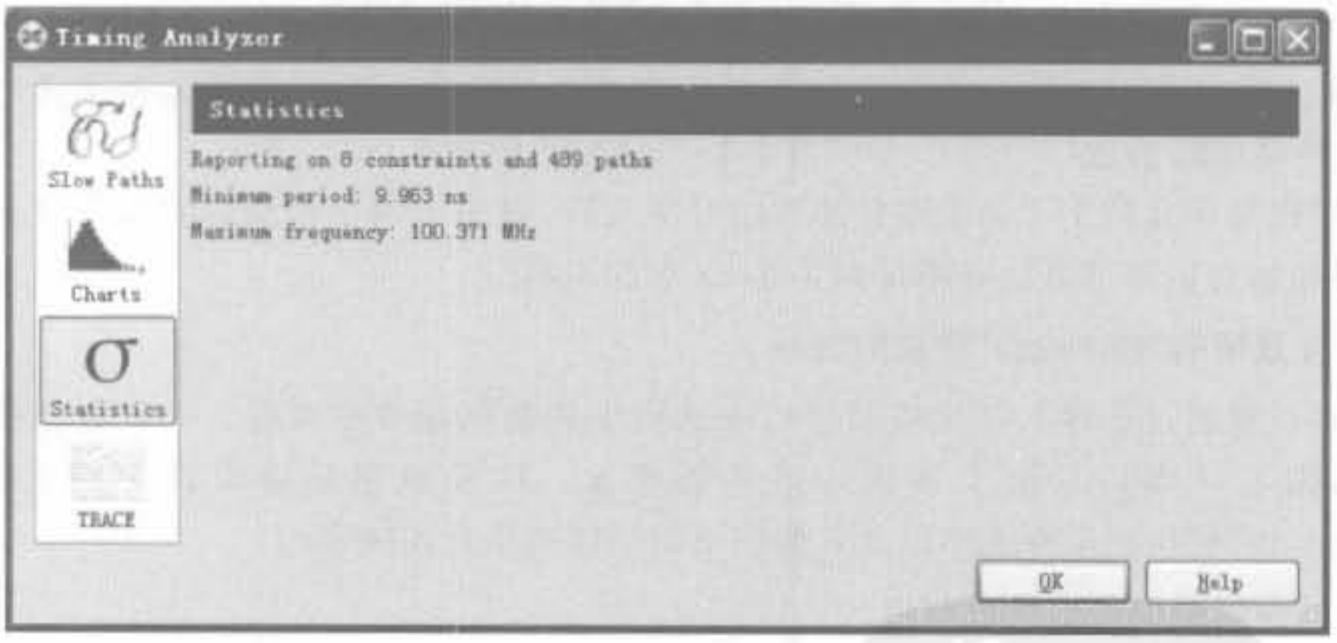


图 8-61　时序分析报告的简要指示信息

员只获取到最高时钟频率，不明白设计的整体时序性能。例如，用户期望设计达到100MHz，但经过时序分析后只达到99MHz，那么设计离约束的性能差距究竟有多大？是重新设计还是局部修改？这些困惑都是不确定的。图表分析结果弥补了这一缺陷，直观地将所有的路径列表统计，如图 8-60 所示。大多数路径的延迟处于 8ns 左右，只有一条路径为 9.96ns，比较接近 10ns。整体设计的时序裕量还是比较富裕的。

3）统计结果

统计结果列出设计的时序约束个数以及所分析的路径，并给出设计的最小周期和可达

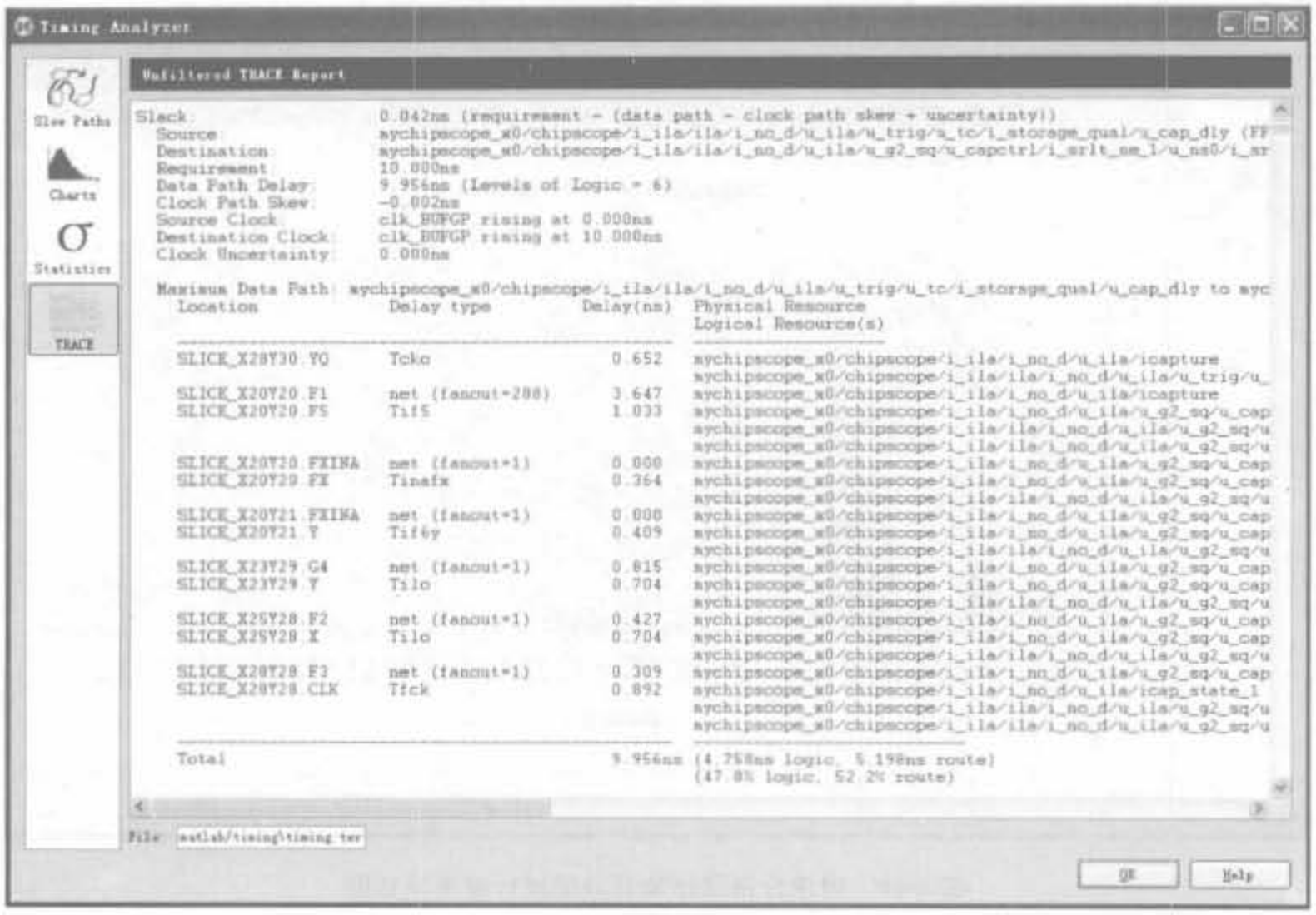


图 8-62 时序分析报告的路径跟踪指示信息

到的最高工作频率。如时钟约束为 100MHz，经过分析可达 100.371MHz，基本满足需求。

4）路径跟踪信息

路径跟踪信息列出了每条路径的详细时序报告，对于不满足的路径将以红色显示。其中，各路径信息的解读方法和指标参见第 11 章的内容。

5. 设置所有“Gateway”模块的速率

速率设置通过选择 I/O 端口时序约束选项中的数据速率来实现。一旦数据速率选项被选中，则 I/O 端口只能工作在约束的频率上。约束频率是由设计中的系统周期、“Gateway In”模块的采样速率以及其他模块的采样速率共同决定的。

6. 减少时钟使能信号的扇出

Xilinx ISE 的映射器在高扇出网络中采用复制和放置寄存器的方法递归地分割负载，意味着在 System Generator 中亦可以该方法来改善时钟使能信号高扇出的情况。System Generator 具备该项特征，但需要在实现过程中使能下列选项：

- Perform Timing-Driven Packing and Placement：on
- Map Effort Level：High
- Register Duplication：on

在 ISE 开发流程中，上述 MAP 选项是默认打开的。但要在 System Generator 中完成比特流的生成，需要通过修改 bitstream . opt 文件或提供用户自定义的. opt 文件才能打开上述选项。

8.5.6 设计资源评估

硬件资源评估是基于FPGA设计的必需步骤之一。System Generator提供了资源评估模块，可快速地估计出设计模型所需的硬件资源。该模块位于工具模块子集中，如图8-63所示。

此外，Xilinx模块库中的可综合模块都具备如图8-64所示的简易资源评估信息。由于在顶层模块中还包括大量的布局布线资源，因此，各个子模块的资源和小于整个顶层设计所占资源。

图8-63 资源评估模块示意图

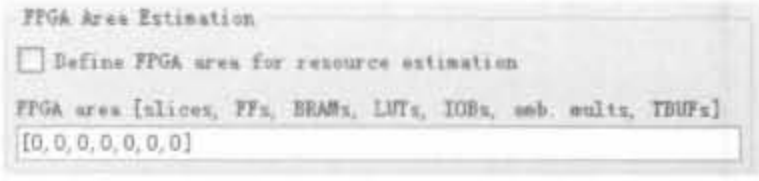


图8-64 Xilinx模块资源信息示意图

当设计规模比较庞大或功能复杂时，通常要采用层次化、模块化的设计方法，来降低设计难度且提高项目的可管理性。为了准确地评估资源且满足用户在各个设计阶段的不同需求，Xilinx资源评估模块提供了不同层次的估计方法。在设计中双击资源评估模块，在“Estimate options”下拉框中可看到4种不同的评估方法，如图8-65所示。

图8-65 Xilinx模块资源方法示意图

其中，不同选项的含义如下：

(1) Estimate：估计当前层模型以及所有从属层模块的资源。

(2) Quick：将当前层模块的所有模块资源相加，不包括从属模块的资源。

(3) Post Map：调用ISE的布局布线工具，并根据报告文件(Map Report File，MRP)进行资源评估。

(4) Read Mrp：不调用布局布线工具，直接读取已完成的布局布线报告并进行参数估计。

8.6 开发实例：基于FIR滤波器的协仿真实例

本节给出一个FIR滤波器的协仿真实例，包括滤波器设计、滤波器实现以及硬件协仿真等功能，并且统计设计所需资源，涵盖利用System Generator完成设计的主要步骤。

例8-5 利用System Generator中的分布式FIR滤波器模块，在Spartan-3E Starter开发板上完成基于该系统的硬件协仿真。

(1) 打开 Similink，新建如图 8-66 所示的工程。

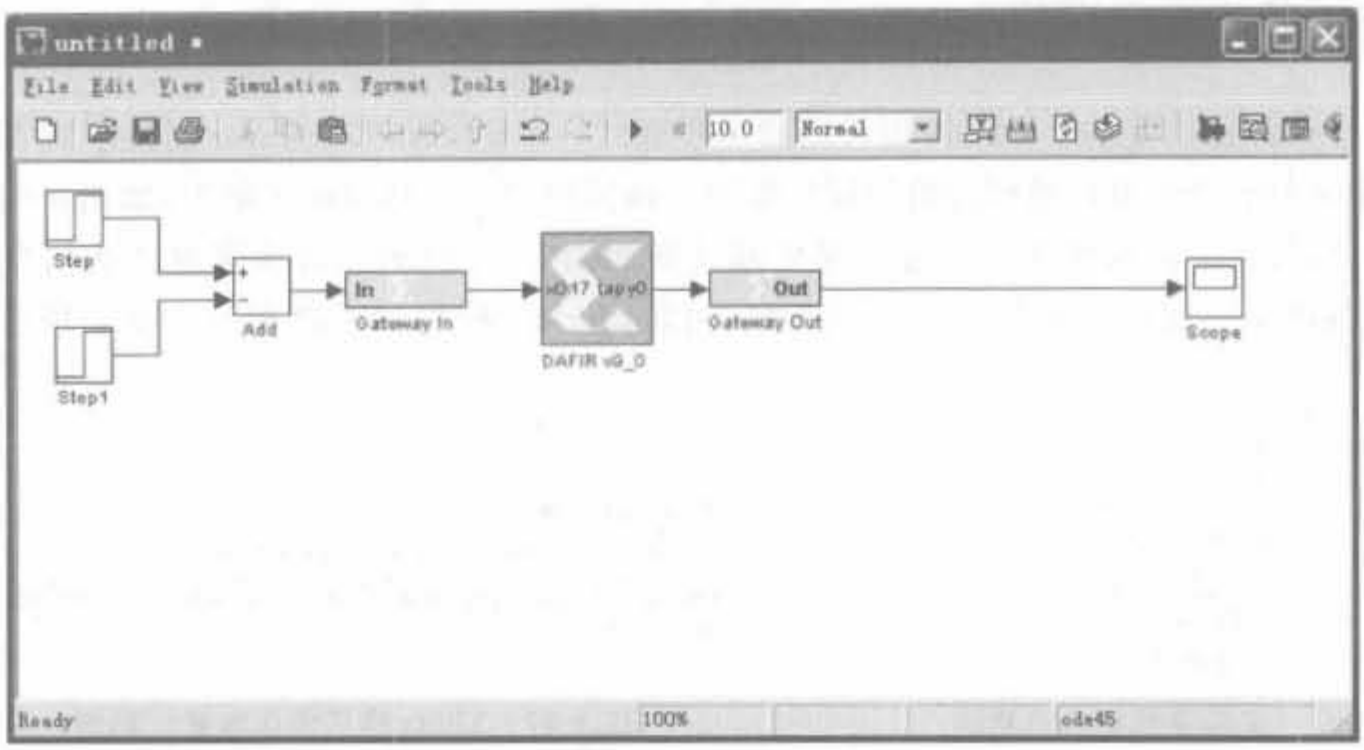


图 8-66　DA 滤波器系统

(2) 在 MATLAB 的命令窗口中键入 fdatool，打开滤波器设计工具，输入参数如图 8-67 所示。单击“Design”按钮，生成滤波器系数。在 FDATool 工具的“File”菜单中选择“Export”命令，将系数导入 MATLAB 的变量空间中。

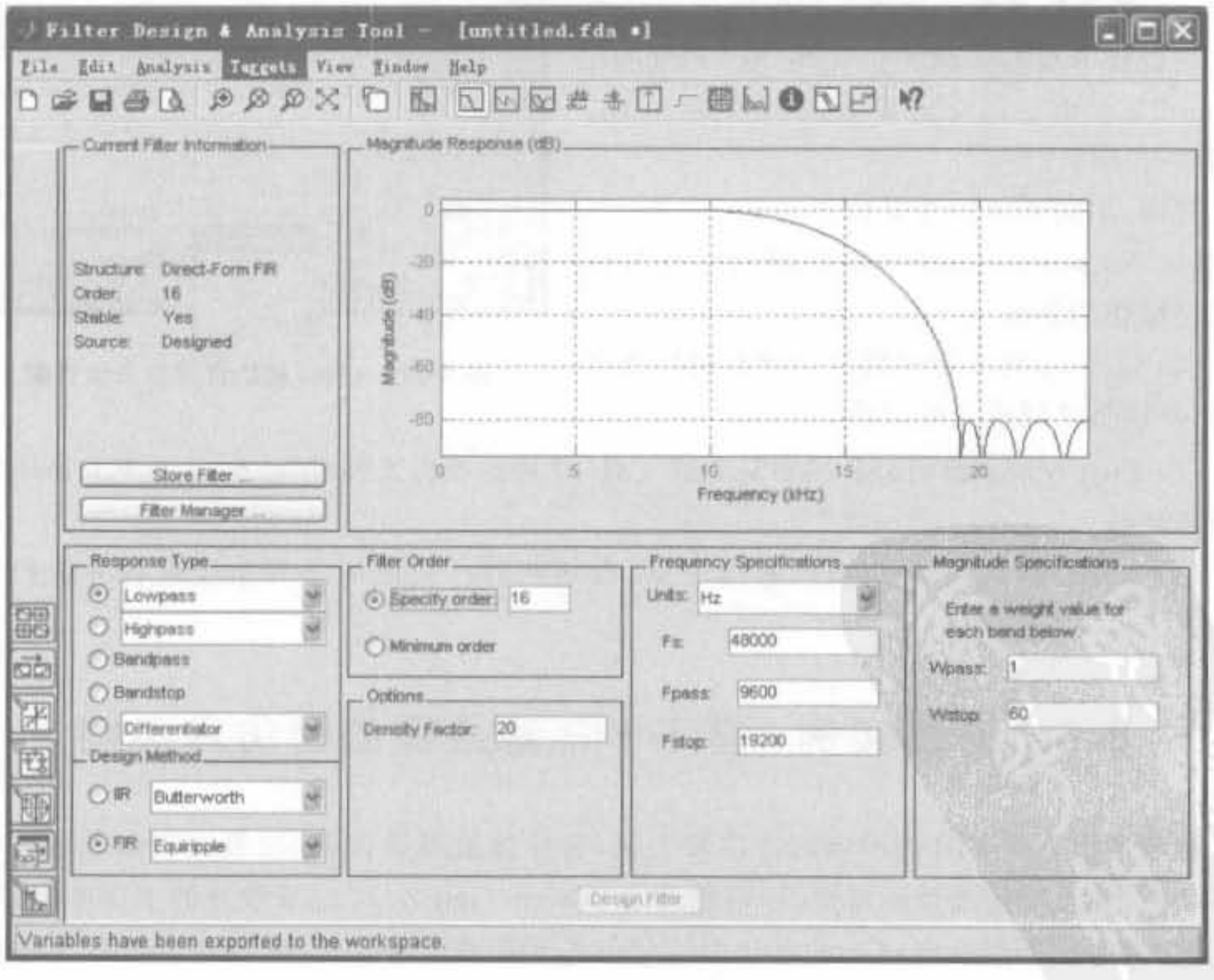


图 8-67　滤波器设计参数示意图

(3) 双击 DA_filter 模块,将该 MATLAB 变量空间的系数复制到"Coefficients"栏中,其余设置和图 8-68 保持一致。

图 8-68 DA 滤波器模块配置参数

(4) 设置 Gateway In 模块属性,各参数值如图 8-69 所示。

图 8-69 Gateway In 模块配置参数

(5) 双击 System Generator 模块，按照 8.4.1 节的方法将“Compilation”栏设置 Spartan-3E 开发板的硬件协仿真。设置完成后，单击“Generator”按钮，编译系统生成硬件代码。

(6) 编译完成后，生成 myfilter hwcosim 模块，如图 8-70 所示。

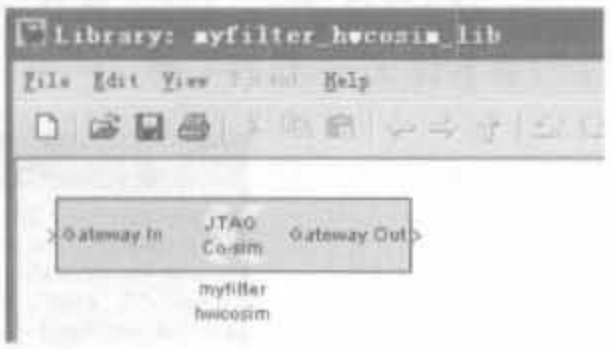


图 8-70 生成的硬件协仿真模块示意图

(7) 将 myfilter hwcosim 模块拖到 myfilter. mdl，连接关系如图 8-71 所示。

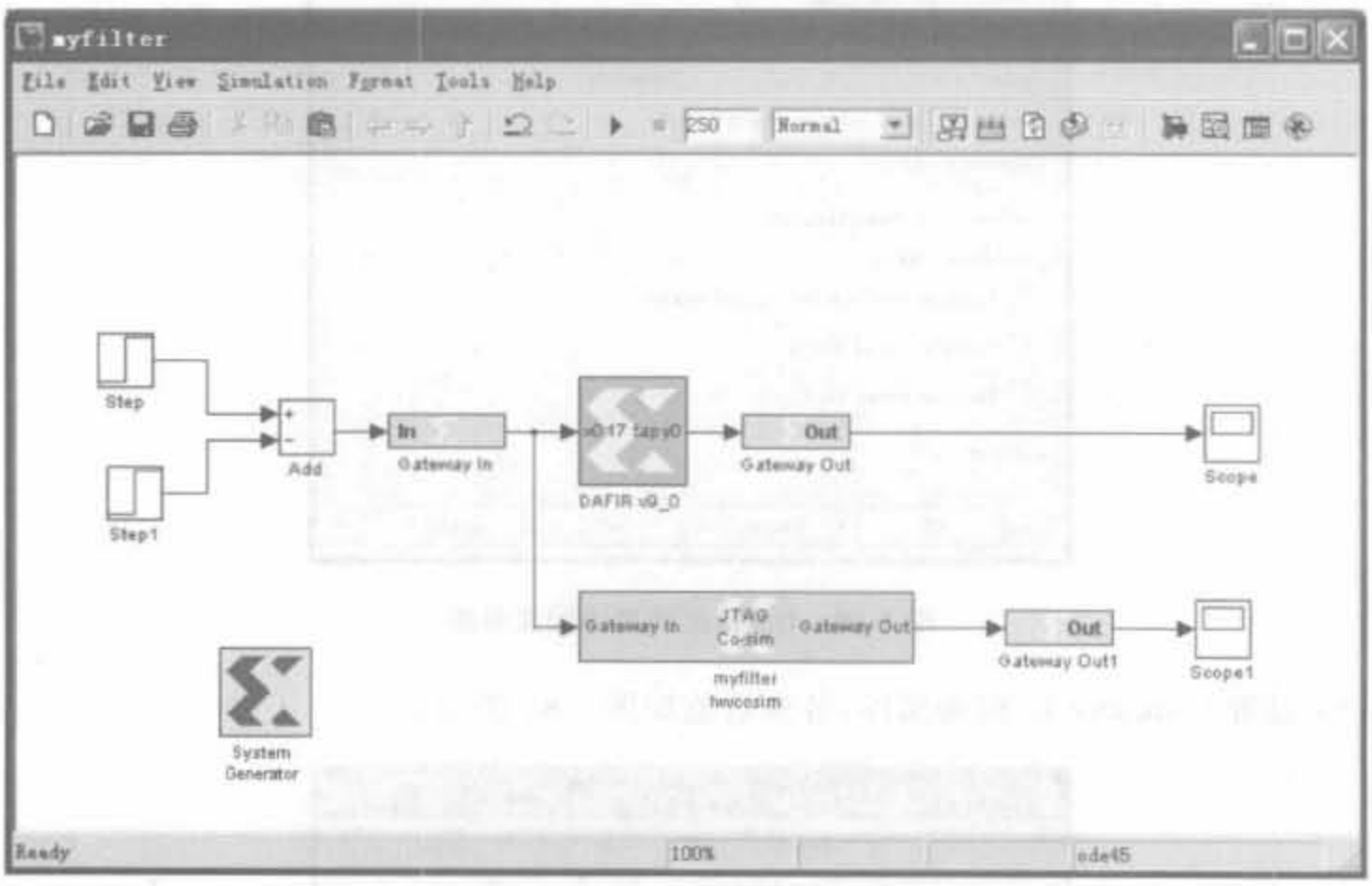


图 8-71 完整的协仿真平台示意图

(8) 双击 myfilter hwcosim 模块，在弹出的配置窗口中切换到“Cable”页面，如图 8-72 所示。选择“Platform USB”；否则，由于其默认“Parallel Cable IV”，不能正确初始化 JTAG 链路，无法下载程序。

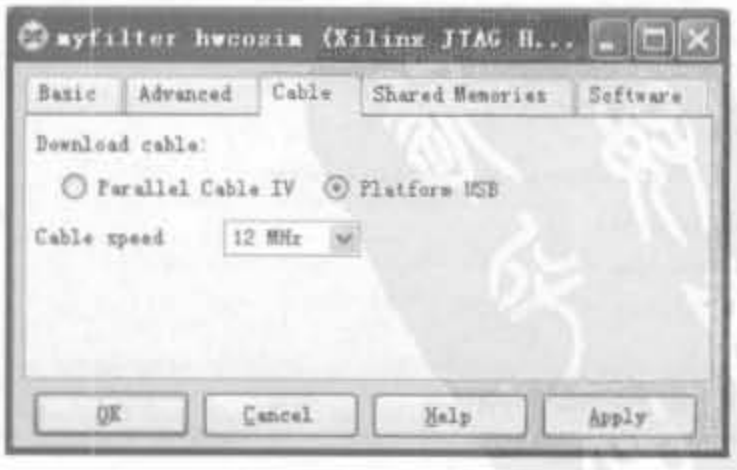


图 8-72 协仿真链路设置示意图

(9) 给开发板上电，并用 USB 连到 PC 上，单击“Start Simulation”开始仿真。System Generator 会自动扫描链路，并将比特文件下载到 FPGA 中，如图 8-73 和图 8-74 所示。

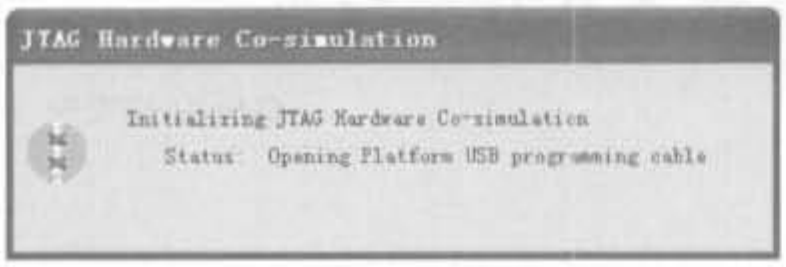


图 8-73　初始化链路状态示意图

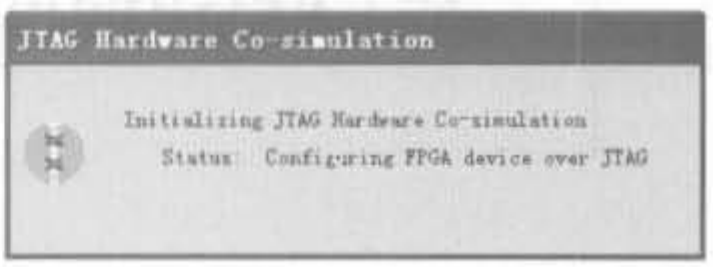


图 8-74　JTAG 链路下载状态示意图

(10) 打开两个 Scope 查看仿真结果，如图 8-75 和图 8-76 所示。可以看到，硬件协仿真结果同仿真结果几乎完全一致，表明了硬件协仿真的正确性。

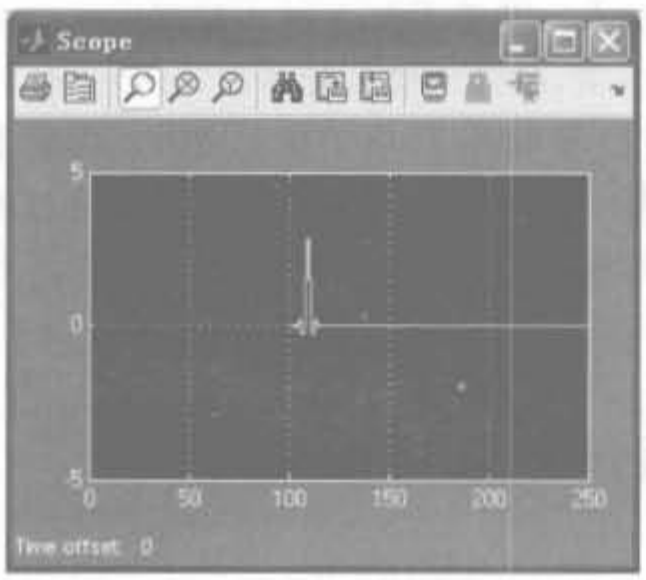


图 8-75　Simulink 软件仿真示意图

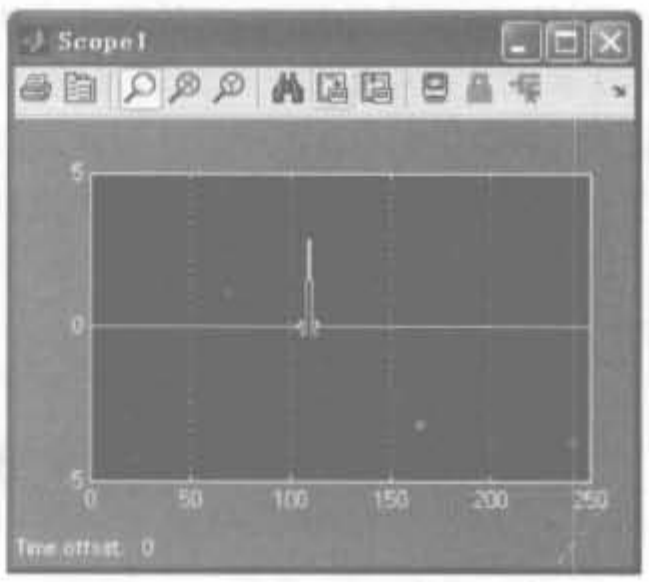


图 8-76　硬件协仿真结果示意图

8.7　本章小结

System Generator 是 Xilinx Xtreme DSP 计划的一部分，是业界最快的可编程 DSP 解决方案，包括领先的 FPGA 器件、DSP 整体设计方案以及丰富的 IP Core。此外，从设计学角度讲，System Generator 比 HDL 语言设计法先进，是未来的潮流。本章首先介绍了 System Generator 的特征以及安装方法。其次，给出了入门基础，包括 Simulink 的简要用法。接着，较为详细地讲解了设计方法和自动代码生成、子系统建立和使用、硬件协仿真等专题，并给出了导入 HDL 模块、在线调试、多时钟设计、软硬件联合开发等高级话题。最后，给出了 FIR 滤波器协仿真实例，以利读者加深理解。通过本章学习，读者应初步掌握 System Generator 的使用方法，具备开发中、小规模设计的能力。

第9章

基于FPGA的可编程嵌入式开发技术

CHAPTER 9

随着计算机与通信的融合以及广泛的多媒体处理需求，嵌入式系统得到了前所未有的蓬勃发展。嵌入式系统是以专用芯片为核心的专用系统，其特点是面向用户、面向应用、面向产品，软、硬件量体裁衣，满足行业应用个性化的要求。基于 FPGA 的可配置嵌入式系统开发技术以及相应的片上可编程系统(SOPC)解决方案，融入了微处理器技术、数字信号处理技术、可编程系统级芯片设计和软、硬件协同设计技术，不仅提供了基于嵌入式智能平台的嵌入式系统的设计方法，还降低了设计难度，缩短了研发周期，必成为未来的主流技术之一。本章主要介绍 Xilinx 公司提供的可编程嵌入式开发解决方案以及相应的开发平台和方法。

9.1 可编程嵌入式系统(EDK)介绍

9.1.1 基于 FPGA 的可编程嵌入式开发系统

嵌入式系统经历了从单片计算机、工业控制计算机、集中分布式控制系统，发展到嵌入式智能平台的几个阶段；从独立单机使用发展到联网设备；从以模拟电路为主发展到以数字电路为主、数/模混合型，进而进入全数字时代。总的来说，嵌入式系统向着更高性能、更小体积、更低功耗、更廉价、无处不在的方向发展。嵌入式系统的设计和实现朝着基于芯片，特别是片上可编程系统(SOPC)的方向发展。

从系统对上市时间的要求、可定制特性以及集成度等方面考虑，FPGA 在嵌入式系统中获得广泛应用，已经从早期的军事、通信系统等应用扩展到低成本消费电子类产品中。目前，FPGA 在嵌入式系统中主要有 3 种使用方式：

(1) 状态机模式：可以无外设、无总线结构和无实时操作系统，达到最低的成本，应用于 VGA 和 LCD 控制等，根据用户设计可达到不同的性能。

(2) 单片机模式：包括一定的外设，可以利用实时操作系统和总线结构，以中等的成本应用于控制和仪表，达到中等的性能。

(3) 定制嵌入模式：高度集成扩充的外设，实时操作系统和总线结构，达到高性能，应用于网络和无线通信等。

采用 90nm 生产工艺之后，FPGA 器件处理能力更强，且成本低、功耗少，已取代了相当数量的中、小规模 ASIC 器件和处理器，使嵌入式系统具备片上系统(SOC)的规模和动态可

编程的能力，具有明显的优势，成为嵌入式应用的主力军之一。

9.1.2 Xilinx 公司的解决方案

1. 解决方案

Xilinx 公司的嵌入式解决方案以 3 类 RISC 结构的微处理器为核心，涵盖了系统硬件设计和软件调试的各个方面。3 类嵌入式内核分别为 PicoBlaze、MicroBlaze 和 PowerPC，其中 PicoBlaze 和 MicroBlaze 是可裁剪的软核处理器，PowerPC 为硬核处理器。

PicoBlaze 是由 VHDL 语言在早期开发的小型 8 位软处理器内核包，其汇编器是简单的 DOS 可执行文件 KCPSM2.exe，用汇编语言编写的程序经过编译后放入 FPGA 的块 RAM 存储区，汇编器可在 3s 内编译完存储在块 RAM 中的程序。

MicroBlaze 采用功能强大的 32 位流水线 RISC 结构，包含 32 个 32 位通用寄存器和一个可选的 32 位移位寄存器，时钟可达 150MHz。在 Virtex-2 Pro 以及更高系列的平台上，其运行速度可达 120DMIP(DhrystoneMIPS)，占用资源不到 1000 个 Slice。

PowerPC 是 32 位 PowerPC 嵌入式环境架构，确定了若干系统参数，用以保证在应用程序级实现兼容，增加了其设备扩展的灵活性。Xilinx 将 PowerPC 405 处理器内核整合到 Virtex-2 Pro 系列以及更高等级系列的芯片中，允许该硬 IP 核能够深入到 FPGA 架构的任何部位，提供高端嵌入式应用的 FPGA 解决方案。Virtex-4 以及 Virtex-5 系列部分芯片中集成了 2～4 个 PowerPC 405 处理器核。

目前使用较多的嵌入式内核是 MicroBlaze 和 PowerPC，工作频率可达到数百兆赫兹，还添加了新的浮点单元选项，使嵌入式开发人员可将系统性能提升至软件仿真速度的百倍以上，非常适合针对网络、电信、数据通信、嵌入式和消费等市场的产品。

2. 开发工具

嵌入式开发套件(EDK)是用于设计嵌入式可编程系统的全面解决方案。该套件包括嵌入式软件工具(Platform Studio)以及嵌入式 IBM PowerPC 硬件处理器核和/或 Xilinx MicroBlaze 软处理器核进行 Xilinx 平台 FPGA 设计时所需的技术文档和 IP。注意，这里的嵌入式软件工具指用来产生、编辑、编译、链接、加载和调试高级编程语言(通常是 C 或 C++)代码的工具，以便在处理器引擎上执行。

此外，Xilinx 公司提供了大量的硬件平台(即评估板)，提供了大量的软、硬件设计参考，使得设计人员可以快速了解嵌入式系统的基本操作和大量的 IP 核的使用方法，并对其完成评估，以确定最优的设计方案。一般来讲，最快捷的硬件设计方式是在设计系统时以相应的评估板为母板，对其进行必要的修改。

9.2 Xilinx 嵌入式开发系统组成介绍

嵌入式开发套件(EDK)是设计嵌入式处理系统的集成软件解决方案，包含屡获殊荣的 Platform Studio 工具套件，以及利用嵌入式 PowerPC 硬处理器核和/或 Xilinx MicroBlaze 软处理器核进行 Xilinx 平台 FPGA 设计时所需的全部技术文档和 IP。本章主要介绍 EDK

的组成和内核结构。

9.2.1 片内微处理器软核 MicroBlaze

1. MicroBlaze 体系结构

MicroBlaze 软核是一种针对 Xilinx FPGA 器件而优化的功能强大的 32 位微处理器，是业界最快的软处理器 IP 核解决方案。它支持 CoreConnect 总线的标准外设集合，具有兼容性和重复利用性，最精简的核只需要将近 400 个 Slice。

MicroBlaze 软核内部采用 RISC 架构，以及哈佛结构的 32 位指令和数据总线，内部有 32 个通用寄存器 R0～R3、2 个特殊寄存器程序指针(PC)和处理器状态寄存器(MSR)、1 个 ALU 单元、1 个移位单元和两级中断响应单元等基本模块，还可具有 3/5 级流水线、桶形移位器、内存管理/内存保护单元、浮点单元(FPU)、高速缓存、异常处理和调试逻辑等可根据性能需求和逻辑区域成本任意裁剪的高级特性，极大地扩展了 MicroBlaze 的应用范围。MicroBlaze 处理器的内核仍在不断更新中，目前最新版为 MicroBlaze V7.0，其内部架构如图 9-1 所示。

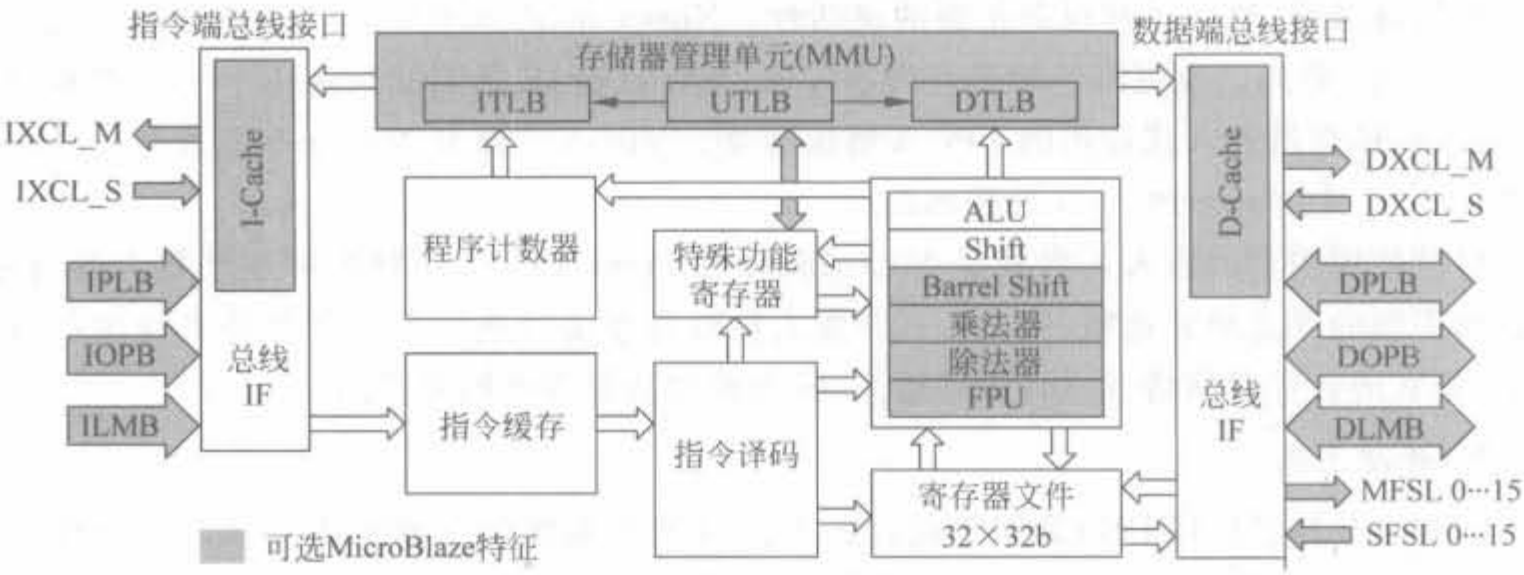


图 9-1 MicroBlaze 内部架构示意图

MicroBlaze 处理器架构均衡了执行性能和设计尺寸，但由于其最高工作频率由 FPGA 芯片提供，所以计算性能随处理器配置、实现工具结果、目标 FPGA 架构和器件速度级别的不同而不同。表 9-1 给出了不同 FPGA 芯片上的 MicroBlaze 性能对比表，其结果不代表一般嵌入式应用的计算性能。

表 9-1 MicroBlaze 内核的最大 Dhrystone 性能

FPGA 芯片	所占逻辑资源(LUT 数)	最高时钟频率(MHz)	Dhrystone 2.1(DMIPS)	性能(DMIPS/MHz)
Virtex-5(5VLX50)	1010	210	240	1.15
Spartan-3E(3S1600E-5)	1842	100	115	1.10.925
Spartan-3E(3S1600E-3)	1357	100	92	

注：DMIPS 表示每秒执行的 Dhrystone 指令数量。

2. MicroBlaze 的总线接口

CoreConnect 是由 IBM 开发的片上总线通信链，它使多个芯片核相互连接成为一个完整的新芯片成为可能。Xilinx 以 IBM CoreConnect 为嵌入式处理器的设计基础，具有丰富的接口资源。目前，最新版本的 MicroBlaze 软核支持的接口标准有：

- 带字节允许的 OPB(On-chip Peripheral Bus，片上外设总线) V2.0 接口；
- 高速的 LMB(Local Memory Bus，本地存储器总线)接口；
- FSL 主从设备接口；
- XCL(Xilinx Cache Link，Xilinx 缓存链路)接口；
- 与 MDM(Microprocessor Debug Module，微处理器调试模块)连接的调试接口。

其中，OPB 是对 IBM Core Connect 片上总线标准的部分实现，适用于将 IP 核作为外设连接到 MicroBlaze 系统中。LMB 用于实现对片上的块 RAM 的高速访问。FSL 是 MicroBlaze 软核特有的一个基于 FIFO 的单向链路，可以实现用户自定义 IP 核与 MicroBlaze 内部通用寄存器的直接相连。XCL 则是 MicroBlaze 软核新增加的，用于实现对片外存储器的高速访问。MicroBlaze 软核还有专门的调试接口，通过参数设置，开发人员可以只使用特定应用所需要的处理器特性。Xilinx 提供了大量的外设 IP Core，可外挂到 MicroBlaze 的 OPB 总线上，如 DMA 单元、以太网 MAC 层处理器、PCI/PCIe 接口、串口以及 USB 等，如图 9-2 所示。

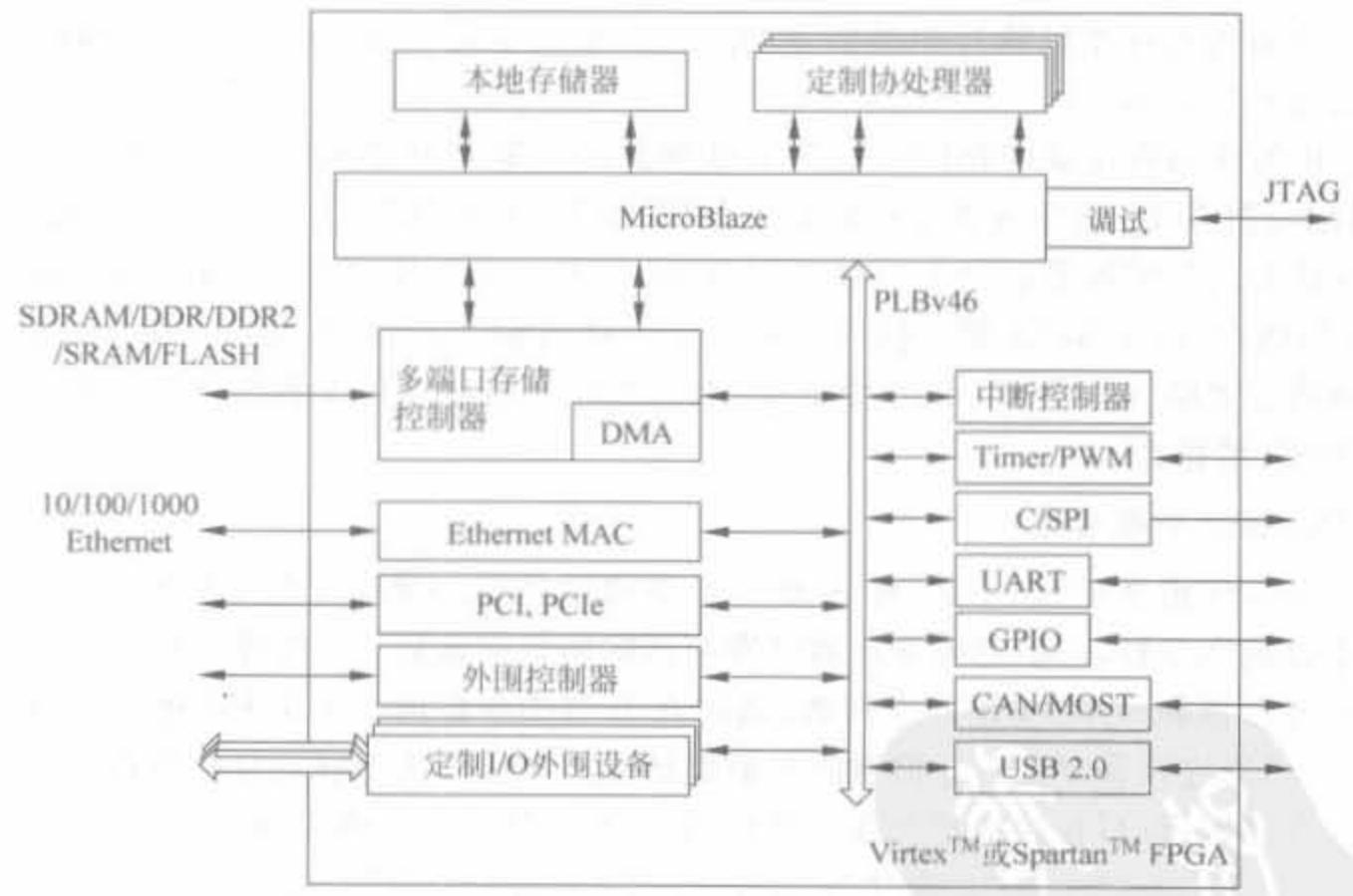


图 9-2　MicroBlaze 支持的外设接口示意图

其中，DMA 和多端口存储器控制器是高速接口，OPB 属于低速接口总线，一个外设一般不能同时和这两类总线相连。以太网 MAC 控制器模块之所以和两类总线连接，是因为其控制接口和 OPB 总线相连，数据接口和高速的 DMA 总线连接。

3. MicroBlaze 寄存器

1）通用寄存器

MicroBlaze 内核中的 32 个 32 位的通用寄存器记为 R0～R31。寄存器并不是由外部复位输入(如 reset 或 debug-rst 脚)复位，而是在比特流下载的时候复位。通用寄存器的描述如表 9-2 所示。

表 9-2 MicroBlaze 通用寄存器功能表

寄存器名	功能描述
R0	任意对 R0 的写操作均被忽略
R1～R13、R18～R31	一般通用寄存器
R14	存储中断的返回地址
R15	位通用寄存器
R16	存储跳转的返回地址
R17	若 MB 配置为支持硬件异常，此寄存器装载硬件异常的返回地址，否则作为通用寄存器使用

2）特殊寄存器

(1) 程序指针(PC)寄存器：存储下一条指令的地址。它可以由一个 MFS 指令读出，且不能由 MTS 指令写入。当使用 MFS 指令的时候，将 Sa 置为 00000，或 rpc 将定义 PC 寄存器。表中内容为程序指针执行中的指令的地址。也就是说，“mfs r2 rpc” 指令将把 mfs 指令自己的地址存入 R2 中。

(2) 机器状态寄存器(MSR)：包含了处理器的控制和状态比特，可以通过 MFS 指令读，也可以通过 MTS 指令或者专用的指令 MSRSET、MSRCLR 写。当对 MSR 进行读操作时，bit29 作为进位位被复制到 bit0 位。写 MSR 需要一个时钟周期的延时，当用 MTS 指令向 MSR 写指令时，在 MTS 指令执行结束一个时钟周期之后，写入的值才生效。所有写入 bit0 的值将被忽略。当使用 MTS 指令或 MFS 指令的时候，将 sx 置为 0001，或 rmsr 被定义为对 MSR 的操作。

4. MicroBlaze 指令集

MicroBlaze 指令字为 32 位，有 A 型和 B 型两种类型的指令。A 型指令有两个源寄存器和一个目的寄存器，用以完成寄存器到寄存器间的数据运算；B 型指令有一个源寄存器、一个目的寄存器和一个 16 位的立即数(通过在 B 型指令前加一个 IMM 指令，可将其扩展到 32 位)，可以进行寄存器和立即数间的数据运算。其指令按功能划分有逻辑运算、算术运算、分支、存储器读/写和特殊指令等。类型 A 和类型 B 的指令格式如图 9-3 所示。

图 9-3 MicroBlaze 指令格式

MicroBlaze 指令执行的流水线是并行流水线，它分为 3 级流水：取指、译码和执行，如图 9-4 所示。

图 9-4　MicroBlaze 的流水线

完整的 MicroBlaze 指令集见参考文献[11]。

9.2.2　片内微处理器 PowerPC

1. PowerPC 体系结构

PowerPC 是由苹果、IBM 和摩托罗拉公司共同开发的微处理器结构，采用精简指令计算架构(RISC)，并形成了一个开放的标准。Xilinx 芯片中内嵌的 PowerPC 结构经过 IBM 公司优化，以较简单的、快速的运算为基础，促成微处理器在一个给定的时钟速度下执行较多的指令，主要包括 PowerPC 405 系列。PowerPC 提供了 3 个不同层面的应用，从下往上分别是用户指令集结构(UISA)、虚拟环境结构(VEA)和操作环境结构(OEA)，各层功能如表 9-3 所示。

表 9-3　PowerPC 3 层结构的说明

用户指令集结构(UISA)	虚拟环境结构(VEA)	操作环境结构(OEA)
1. 定义了用户级软件所必须遵守的结构。 2. 定义了基本的用户指令集、寄存器、数据结构、浮点内存约定，还包括用户程序的异常处理模型、内存模型以及编程模型。 3. 所有的 PowerPC 都遵守相同的 UISA 结构。	1. 定义了超出典型用户软件需求的附加用户级需求功能。 2. 表述了多个芯片访问存储器环境下的存储器模型。 3. 定义了高速缓存(Cache)模型以及 Cache 控制指令集。 4. 定义了用户角度基于时间的资源。 5. VEA 结构必须和底层 UISA 结构保持一致。	1. 定义了典型的操作系统所要求的管理级资源。 2. 定义了内存管理模型、监控寄存器、同步需求以及异常模型。 3. 定义了监控角度的基于时间的资源。 4. OEA 结构必须和底层的 UISA 和 VEA 级结构保持一致。

这种层次结构提供了软件兼容的灵活性。此外，所有的 PowerPC 都符合 UISA 规范，保证 PowerPC 对应用程序的兼容性，而 VEA 和 OEA 可以有不同版本。Xilinx FPGA 芯片中内嵌的 32 位硬 PowerPC 核可以实现高性能嵌入式应用。目前，在单片 FPGA 芯片中可最多集成 2 个硬 PowerPC 核。PowerPC 集成了 5 级标量流水线，具有独立的指令缓存和数据缓存、1 个 JTAG 端口、Trace FIFO、多个定时器和 1 个内存管理单元(MMU)。此外，Xilinx 的高端器件还集成了辅助处理器单元控制器(APU)，可直接控制 FPGA 架构内的硬件指令协处理。PowerPC 不占用 FPGA 内部的任何逻辑资源，其内部架构如图 9-5 所示。

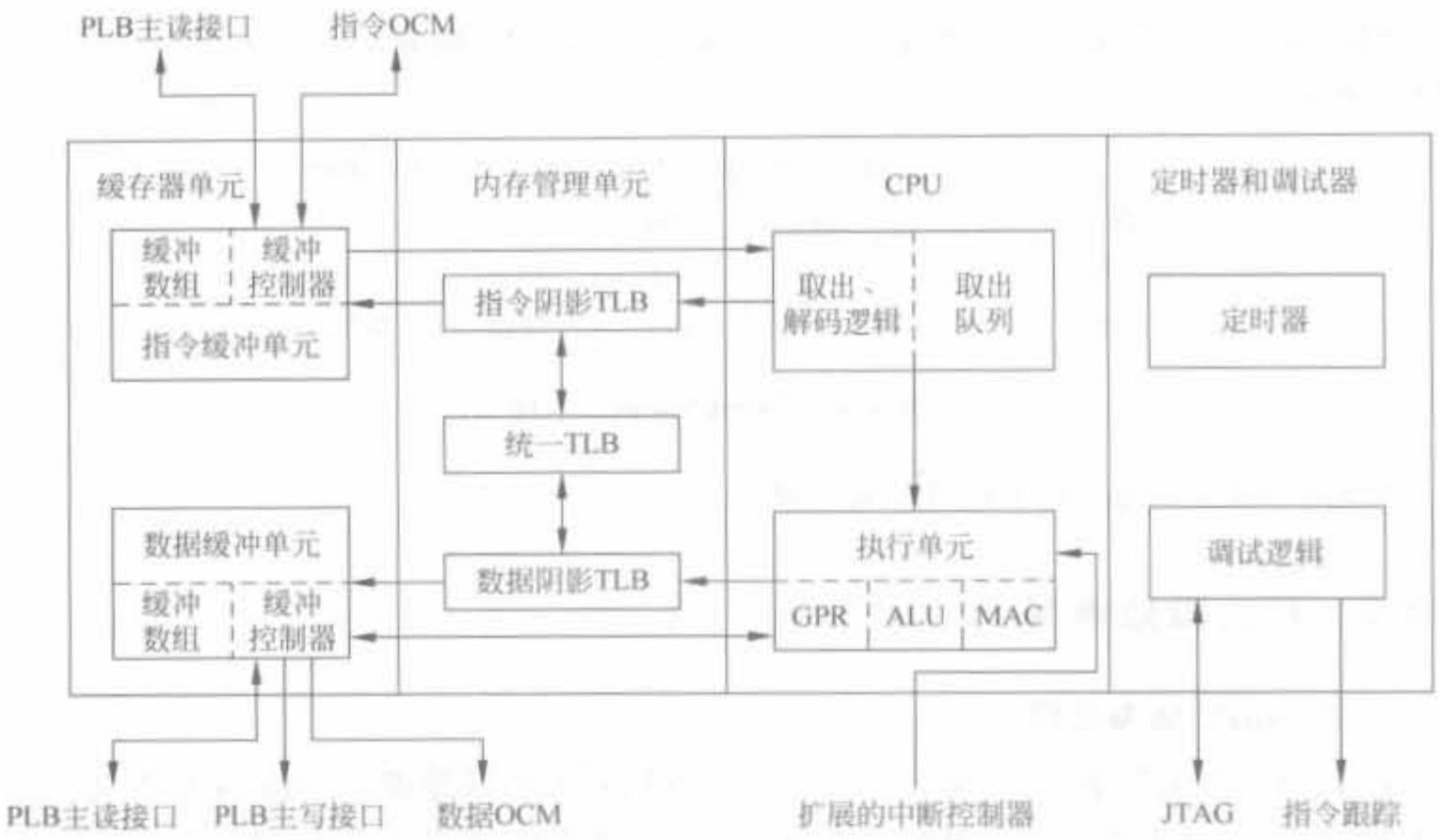


图 9-5 PowerPC 硬核的内部结构示意图

2. PowerPC 总线接口

与传统的总线接口不同，由于 PowerPC 处理器内核被嵌入到 FPGA 芯片中，利用 Xilinx 的 IP 植入和主动互连技术，几百个处理器结点直接连接到 FPGA 的逻辑和存储器阵列。这种总体植入在硬件/软件的系统结构中提供了超强的灵活性，可以有效地将复杂的功能成分在硬件中高速地实现和在软件中高度灵活地实现。这种直接连接的配置克服了利用总线在 FPGA 和附加外部处理器之间接口的“瓶颈”。

PowerPC 处理器也采用 CoreConnect 技术，可运行在 100MHz～133MHz 的高带宽 64 位总线。为了使灵活性达到最大，CoreConnect 结构是作为软 IP 在 FPGA 中实现的。和 MicroBlaze 软核一样，PowerPC 也具备 LMB 和 OPB 总线，分别用来接口高速和低速外设到 PowerPC 处理器。此外，PowerPC 还具有器件控制寄存器总线，完成对通用外设器件寄存器的访问。

3. PowerPC 寄存器

PowerPC 的寄存器可分为通用寄存器、专用寄存器、机器状态寄存器、条件寄存器和芯片控制寄存器 5 大类，如表 9-4 所示。

表 9-4 PowerPC 寄存器列表

寄存器分类	寄存器名称	读写权限	功能描述
通用寄存器	r0～r31	可读可写	完成和内存的数据交互
机器状态寄存器	MSR	可读可写	给出机器工作状态
条件寄存器	CR	可读可写	完成各类判决和跳转语句的相应
专用寄存器	众多，不一一列举	部分可读可写，部分只可读，部分只可写	操作 PowerPC 所有的系统资源
芯片控制寄存器	DCR	可读可写	用于控制同一芯片内的其余外设
基时寄存器	TBL、TBU	只读	TBL、TBU 各 32bit，合成 64bit 计数器

1）通用寄存器

PPC(PowerPC 的缩写)有 32 个 32bit 的通用寄存器，可通过加载指令从内存中读取数值，或通过存储指令将数值写入内存。所有的计算指令的操作数都是通用寄存器，且输出结果都存放于通用寄存器中。所有通用寄存器都可通过软件代码访问。

2）专用寄存器

PPC 有丰富的 32 位专用寄存器，可访问额外的处理器资源，如计数寄存器、连接寄存器、调试资源、计数器、中断寄存器以及其余寄存器资源等。大多数专用寄存器是应用程序所不能访问的，只有计数器和连接寄存器等少数专用寄存器能被所有的软件访问。

3）机器状态寄存器

32bit 机器状态寄存器(MSR)定义了 PowerPC 处理器的工作状态，允许用户修改。

4）条件寄存器

32 位的条件寄存器(CR)可分为 8 个区域(CR0～CR7)，每区域包含 4bit，可用于控制所有的条件分支。算术指令可配置 CR0，比较指令可配置所有的 CR 数值。应用软件可访问所有的 CR 数值。

5）芯片控制寄存器

32bit 芯片控制寄存器用于配置、控制和读取外部处理器，虽然芯片控制寄存器不是 PPC 的一部分，但仍可在特殊软件中通过 mtdcr 和 mfdcr 指令来访问。

4. PowerPC 指令集

PowerPC 实现 5 级流水线，包括取指、译码、执行、写回、加载写回。PowerPC 的指令包括数学运算、逻辑运算、比较、跳转、中断等指令，分为 B、D、I、M、SC、X、XFX、XL 以及 XO 类型，详细的指令集见参考文献[12]。

PowerPC 执行指令的速度接近每周期执行一条指令，各类指令的典型执行速度如表 9-5 所示。

表 9-5 PowerPC 的指令执行周期列表

指令类	执行周期	指令类	执行周期
算术	1	移入、移出设备控制寄存器	3
逻辑	1	移入、移出特殊寄存器	1
移位和翻转	1	分支发生	1 或 2
乘法(32 位、48 位、64 位)	1、2、4	分支未发生	1
除法	35	预测发生	1 或 2
加载	1	预测未发生	1
存储	1	分预测分支	2 或 3

9.2.3 常用的 IP 核以及设备驱动

Xilinx 在 EDK 环境中提供了嵌入式系统中常用的设备，包括通用 I/O 设备、中断控制器设备、定时器、外部存储器控制器以及以太网、串口等高、低通信设备。上述外设以及外设驱动都以 IP Core 的形式给出，便于使用。本节主要介绍 CPU 系统所必需的基本外设，如

通用 I/O、中断以及外存储器控制器。

1. 通用 I/O 设备(GPIO)

1) GPIO 结构

通用 I/O 设备是 32 位的 OPB 总线外设，每一位 GPIO 都可动态配置为输入、输出端口，包含一个寄存器和一个多路器。每个 GPIO 可最多包含两个通道，通过 IPIF 模块连接到 OPB 总线，如图 9-6 所示，其中的 IPIF 模块相当于外部总线控制器。

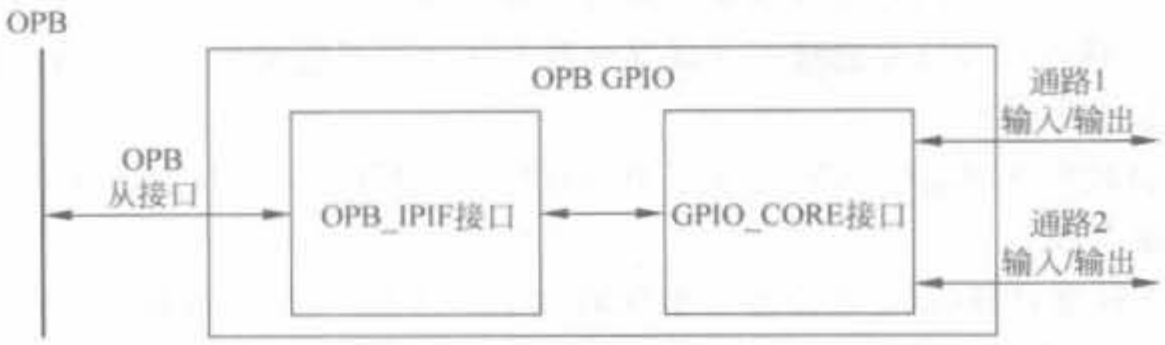


图 9-6　GPIO 模块的连接示意图

GPIO 的寄存器可以以双字(32bit)、字(16bit)以及字节(8bit)的方式访问。为了与 OPB 总线相连，需要和 IPIF 寄存器匹配，因此 GPIO 的寄存器是字边界访问的。GPIO 的数据格式如图 9-7 所示。

字节地址	N	N+1	N+2	N+3
字节序号	0	1	2	3
字节排列	MSByte			LSByte
位序号	0			31
位排列	MSBit			LSBit

图 9-7　GPIO 的数据格式示意图

GPIO 设备完整的端口信号如表 9-6 所示，在系统中可根据实际需求配置相应的参数。其中，C_OPB_AWIDTH 为 OPB 外设地址总线宽度；C_OPB_DWIDTH 为 OPB 外设数据总线宽度；C_GPIO_WIDTH 为 GPIO 的总线宽度；C_BASEADDR 为系统为 GPIO 设备分配的基地址；C_HIGH ADDR 为高地址。

表 9-6　GPIO 组件的端口信号

信号名	接口	I/O	位　宽	描　述
OPB_ABus	OPB	I	[0：(C_OPB_AWIDTH-1)]	OPB 地址总线
OPB_Be	OPB	I	[0：(C_OPB_DWIDTH/8-1)]	OPB 字节使能
OPB_Clk	OPB	I	1	OPB 时钟
OPB_DBus	OPB	I	[0：(C_OPB_DWIDTH-1)]	OPB 数据总线
OPB_RNW	OPB	I	1	OPB 只读指示
OPB_Select	OPB	I	1	OPB 片选信号
OPB_seqAddr	OPB	I	1	OPB 连续地址

续表

信号名	接口	I/O	位　宽	描　述
Sln_DBus	OPB	O	[0：(C_OPB_DWIDTH-1)]	从设备数据位宽
Sln_eerAck	OPB	O	1	从设备出错响应
Sln_retry	OPB	O	1	从设备重试
Sln_toutsup	OPB	O	1	从设备超时禁止
Sln_xferAck	OPB	O	1	从设备传输响应
IP2INTC_Irpt	OPB	I	1	中断输入信号
GPIO_I/O	外设	I/O	[0：(C_GPIO_WIDTH-1)]	外设交互的双向数据端口
GPIO_in	外设	I	[0：(C_GPIO_WIDTH-1)]	GPIO 输入数据端口
GPIO_d_out	外设	O	[0：(C_GPIO_WIDTH-1)]	GPIO 输出数据端口
GPIO_t_out	外设	O	[0：(C_GPIO_WIDTH-1)]	GPIO 3 态输出数据端口

OPB 总线对 GPIO 寄存器的读写时序如图 9-8 和图 9-9 所示。实际上，OPB 总线有 4 种不同的总线访问方式：访问寄存器接口、访问 SRAM 接口、访问 FIFO 接口以及突发传送，每种访问方式用于不同的总线操作，具有不同的时序。

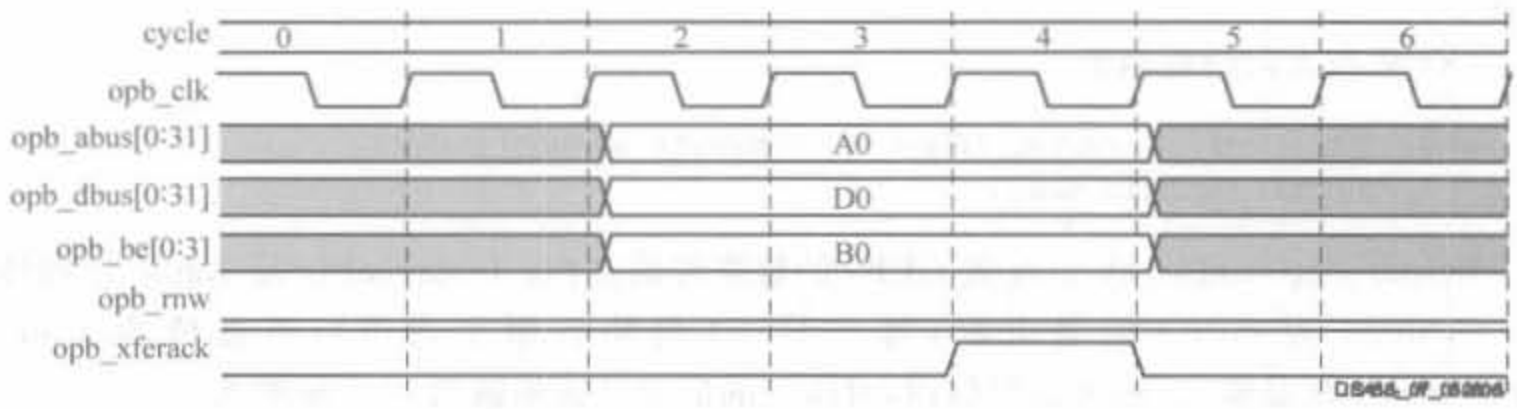


图 9-8　OPB 总线读取 GPIO 寄存器的时序逻辑

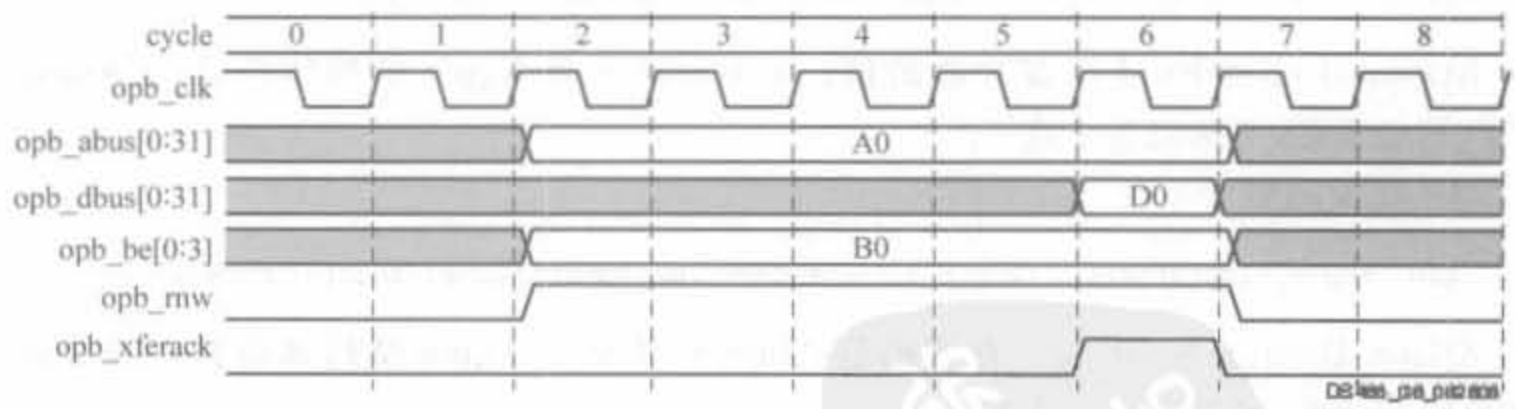


图 9-9　OPB 总线写 GPIO 寄存器的时序逻辑

无论读、写操作，都是由 OPB 总线发出请求，经过几个时钟周期后，收到应答信号表明操作成功，是 CPU、DSP 等处理器最常见的读、写寄存器操作。

2）GPIO 驱动

在 EDK 中，与 GPIO 有关的底层文件有 xgpio. c、xgpio. h、xgpio_i. h 以及 xgpio_l. h。其中，xgpio. c 定义了 GPIO 所有的驱动函数，所以在 GPIO 的用户代码中添加下列语句：

```
#include "xgpio.h"
#include "xgpio_l.h"
#include "xgpio_i.h"
```

在 xgpio.c 中定义了驱动函数，包含 GPIO 的初始化、配置、方向的设定、读取和赋值输出等函数。下面给出常用的 GPIO 操作函数。

(1) 初始化函数

```
XStatus  XGpio_Initialize (XGpio * InstancePtr,Xuint16 DeviceId);
```

其中，InstancePtr 是 Xgpio 结构体指针，存储器的指针参数必须被预先指定；DeviceId 是由 Xgpio 控制的唯一的设备 ID，可在 xparameter.h 文件中找到。通过 XGpio_Initialize 函数将唯一的设备 ID 和 Xgpio 结构体联系起来指定设备。

(2) 配置查找函数

```
XGpio_Config *   XGpio_LookupConfig (Xuint16 DeviceId);
```

该函数查找唯一标志符 DeviceId 所标识的设备配置，系统配置表里包含了每一个设备的配置信息。返回的 m_XGpio 即为设备配置结构指针，返回 m_XNULL 表明未找到标识设备。

(3) 数据方向设置函数

```
void  XGpio_SetDataDirection (XGpio * InstancePtr,unsigned Channel,
      Xuint32 DirectionMask);
```

XGpio_SetDataDirection 配置 GPIO 的数据传输方向；InstancePtr 是 Xgpio 结构体指针；Channel 为 GPIO 的通道数(每个 GPIO 模块有两个通道)，可选值为 1 或 2；DirectionMask 是输入/输出的位标识，对应位的值为 0 表示输出，1 表示输入。

(4) 数取函数

```
Xuint32  XGpio_DiscreteRead (XGpio * InstancePtr,unsigned Channel);
```

XGpio_DiscreteRead 读寄存器的值；InstancePtr 是 Xgpio 结构体指针；Channel 为 GPIO 的通道数，可选值为 1 或 0。

(5) 赋值函数

```
void  XGpio_DiscreteWrite (XGpio * InstancePtr,unsigned Channel,Xuint32 Mask);
```

XGpio_DiscreteWrite 写寄存器的值；InstancePtr 是 Xgpio 结构体指针；Channel 为 GPIO 的通道数，可选值为 1 或 0。

2. 中断控制器设备

1) GPIO 结构

中断控制器(opb_intc)由中断控制核和总线接口组成。中断核可通过参数配置，与相应的总线接口逻辑配合，接在 OPB 总线上或 DCR 总线上，即 opb_intc 和 dcr_intc，可用于 MicroBlaze 和 PowerPC 嵌入式系统。其和 OPB 总线的典型连接方式如图 9-10 所示。

中断控制器包括 8 个可访问的寄存器，如表 9-7 所示。通过寄存器的配置，中断输入可以通过电平触发或沿触发，包括高电平触发、低电平触发、上升沿触发和下降沿触发。

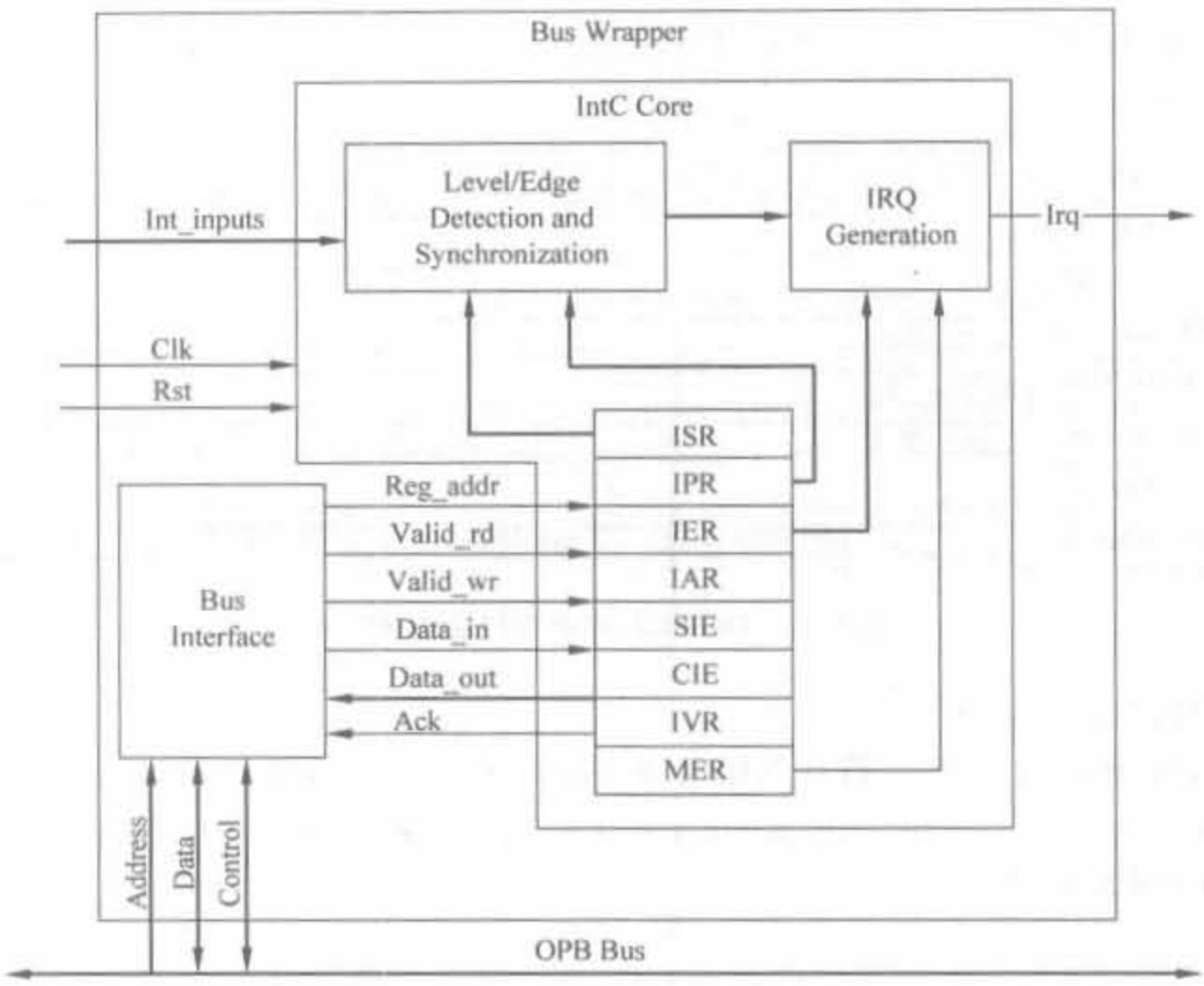


图 9-10 中断控制器和 OPB 总线的连接示意图

表 9-7 中断控制器以及地址偏移

寄存器名	功 能	OPB 偏移地址	寄存器名	功 能	OPB 偏移地址
ISR	中断状态寄存器	0	SIE	中断使能位设置寄存器	16
IPR	中断挂起寄存器	4	CIE	中断清除寄存器	20
IER	中断使能寄存器	8	IVR	中断向量寄存器	24
IAR	中断响应寄存器	12	MER	主使能寄存器	28

中断控制器完整的端口信号如表 9-8 所示，在系统中可根据实际需求配置相应的参数。

表 9-8 中断控制器的端口信号

信号名	接口	I/O	位 宽	描 述
OPB_ABus	OPB	I	[0：(C_OPB_AWIDTH-1)]	OPB 地址总线
OPB_Be	OPB	I	[0：(C_OPB_DWIDTH/8-1)]	OPB 字节使能
OPB_Clk	OPB	I	1	OPB 时钟
OPB_DBus	OPB	I	[0：(C_OPB_DWIDTH-1)]	OPB 数据总线
OPB_RNW	OPB	I	1	OPB 只读指示
OPB_Select	OPB	I	1	OPB 片选信号
OPB_rst	OPB	I	1	OPB 复位信号，高电平有效
OPB_seqAddr	OPB	I	1	OPB 连续地址使能
intC_xferAck	OPB	O	1	中断控制器传输响应
intC_ErrAck	OPB	O	1	中断控制器错误响应
intC_toutSup	OPB	O	1	中断控制器超时禁止
intC_retry	OPB	O	1	中断控制器重试请求

中断控制器的 OPB 读写时序如图 9-11 所示。

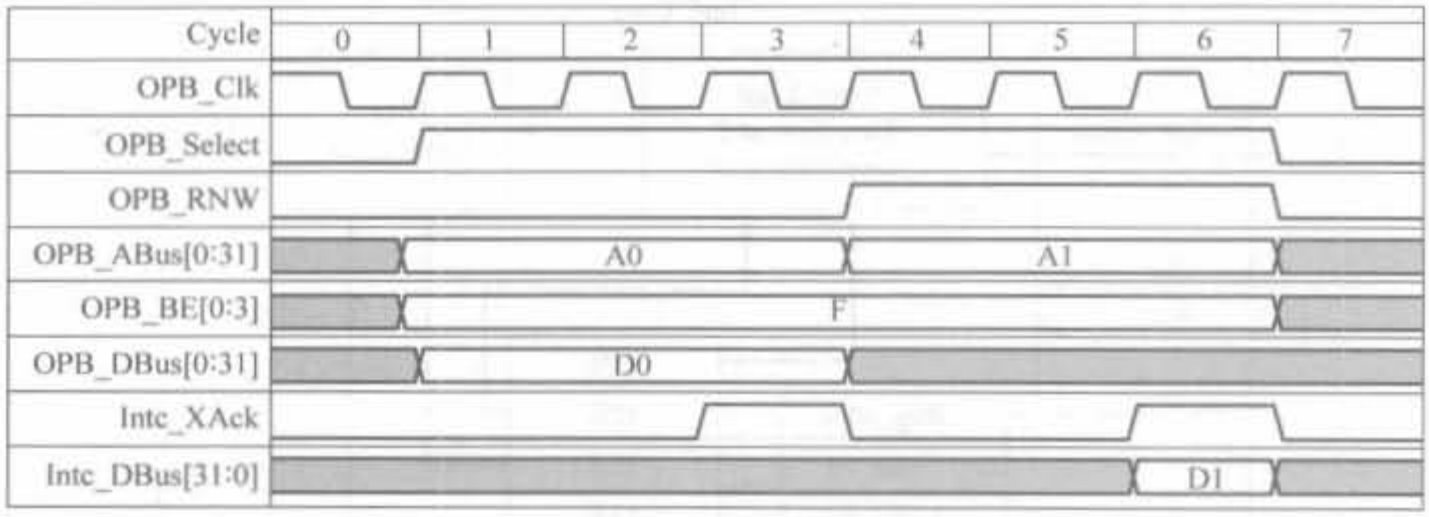


图 9-11　OPB 总线中断控制器读写时序

2）中断控制器驱动

在 EDK 中，与中断控制器有关的底层文件有 xintc. c、xintc. h、xintc_g. c、xintc_i. h、xintc_intr. c、xintc_l. c、xintc_l. h、xintc_options. c 以及 xintc_selftest. c。所以在 GPIO 的用户代码中添加下列语句：

```
#include "xintc.h"
#include "xintc_l.h"
#include "xintc_i.h"
```

中断控制器的驱动函数包含中断控制器的初始化、使能、撤销以及清除等函数。下面给出常用的中断控制器操作函数。

（1）初始化函数

```
XStatus XIntc_Initialize (XIntc * InstancePtr,Xuint16 DeviceId);
```

XIntc_Initialize 用于指定中断控制模块，同时初始化中断结构域、中断向量表、撤销中断源以及中断输出使能。其中，InstancePtr 是 Xintc 的对象；DeviceId 是中断模块的唯一设备 ID 号。返回 m_XST_SUCCESS 表明初始化成功，否则初始化失败。

（2）中断使能函数

```
XStatus XIntc_Start (XIntc * InstancePtr,Xuint8 Mode);
```

XIntc_Start 开始中断控制。其中，InstancePtr 是 Xintc 的对象；Mode 为中断模式，可使能模拟中断以及真实的硬件中断。

（3）中断撤销函数

```
void XIntc_Stop (XIntc * InstancePtr);
```

XIntc_Stop 输出停止中断控制，其中，InstancePtr 是 Xintc 的对象。

（4）中断源连接函数

```
XStatus XIntc_Connect (XIntc * InstancePtr,Xuint8 Id,XInterruptHandler Handler
        void * CallBackRef);
```

XIntc_Connect 连接中断源的 ID 以及与之关联的处理程序，当中断被确认后，处理程

序将运行。其中，InstancePtr 是 Xintc 的对象；Id 为中断源的序号，0 是最高级别的中断；Handler 是中断处理程序；CallBackRef 是返回参数，通常为连接驱动器的对象指针。若返回 mXST_SUCCESS，表明连接正确，否则连接失败。

（5）中断源撤销函数

```
void XIntc_Disconnect (XIntc * InstancePtr,Xuint8 Id);
```

XIntc_Disconnect 撤销与中断源 Id 关联的处理程序。其中，InstancePtr 是 Xintc 的对象；Id 为中断源的序号，0 是最高级别的中断。

（6）特定中断使能函数

```
void XIntc_Enable (XIntc * InstancePtr,Xuint8 Id);
```

XIntc_Enable 使能由竞争 ID 提供的中断源，任何一个未决的特定中断条件将导致一个功能调用。其中，InstancePtr 是 Xintc 的对象；Id 为中断源的序号，0 是最高级别的中断。

（7）特定中断撤销函数

```
void XIntc_Disable (XIntc * InstancePtr,Xuint8 Id);
```

XIntc_Disable 撤销由竞争 ID 提供的中断源，但中断控制器将不产生一个特定 Id 的中断，将继续保留该中断条件。其中，InstancePtr 是 Xintc 的对象；Id 为中断源的序号，0 是最高级别的中断。

（8）中断源响应函数

```
void XIntc_Acknowledge (XIntc * InstancePtr,Xuint8 Id);
```

XIntc_Acknowledge 响应竞争后 Id 的中断源。响应中断后将清除中断条件。其中，InstancePtr 是 Xintc 的对象；Id 为中断源的序号，0 是最高级别的中断。

3. 外部存储器控制器

外部存储器控制器基本涵盖了目前所有的外部存储器类型，包括片内 BRAM、sram、sdr sdram、ddr sdram、ddr2 sdram 以及 Flash 等器件。这里以 sdr sdram 控制器为例介绍其结构和底层驱动。

1）sdr sdram 控制器结构

sdr sdram 存储器是动态同步存储器的一种，采用单时钟，其时钟频率就是数据存储的频率，如时钟信号为 100MHz 或 133MHz，则数据读写速率也为 100MHz 或 133MHz。由于 sdram 依赖于电容的电量来区分逻辑“0”和“1”，但电容器会不断漏电，需要周期性地刷新 sdram 的每一个存储单元，因此读写时序比较复杂，且读写速率也达不到工作时钟的频率。sdram 控制器的作用就是将 sdram 存储器的初始化、刷新、地址转换、数据读取等操作封装起来，让使用者将其看成系统黑盒，读数直接按地址访问，写数直接往目的地址赋值即可。Xilinx 提供了 sdr sdram 控制器的 IP Core，在 XPS 中，直接将其添加到 OPB 总线上即可。sdr sdram 控制器的内部结构以及与 OPB 总线的接口如图 9-12 所示。

sdram 控制器完整的信号端口列表如表 9-9 所示。其中，C_SDRAM_DWIDTH 为 sdram 存储器的数据宽度，C_SDRAM_AWIDTH 为 sdram 存储器的地址宽度，C_SDRAM_BANK_ AWIDTH 为存储器 BANK 的地址。

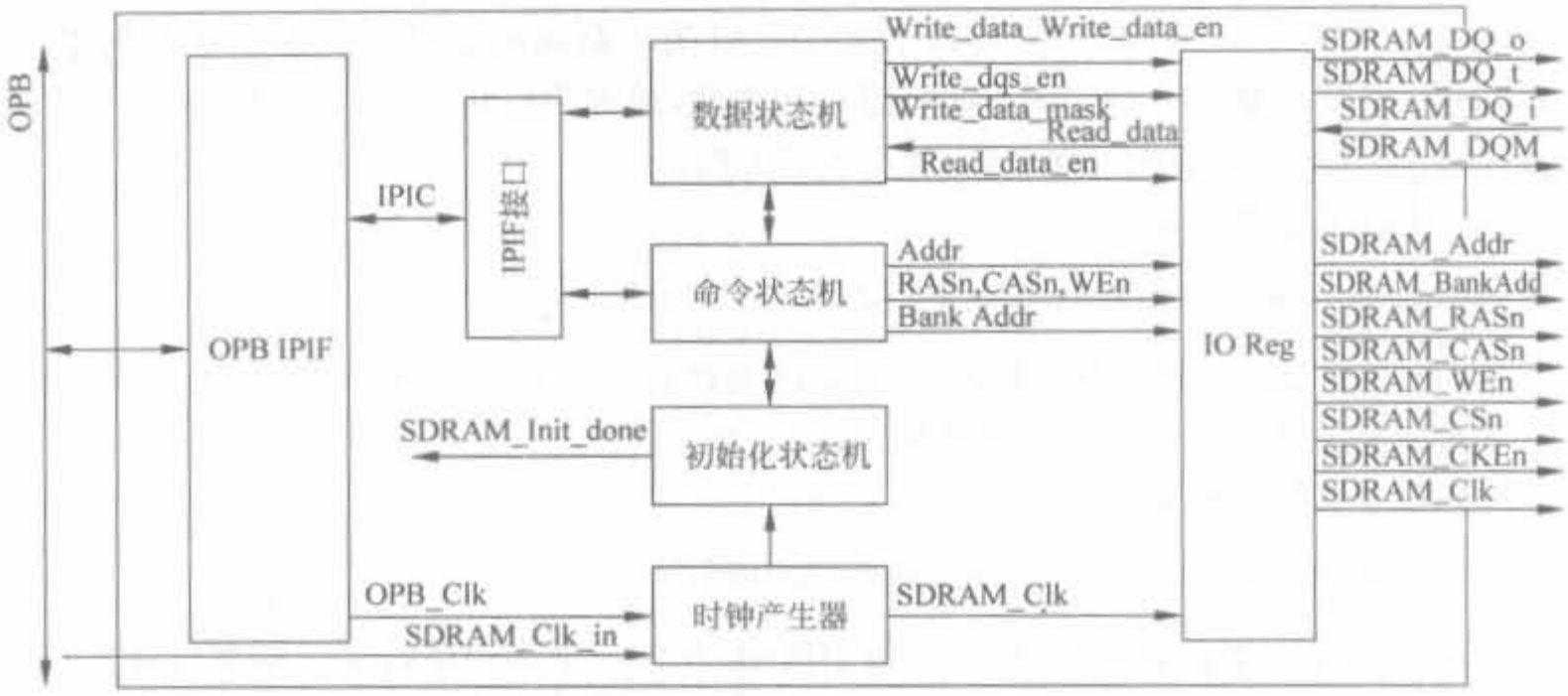


图 9-12　sdram 控制器和 OPB 总线的连接示意图

表 9-9　sdr sdram 控制器的端口信号列表

信号名	接口	I/O	位　宽	描　述
OPB_ABus	OPB	I	[0：(C_OPB_AWIDTH-1)]	OPB 地址总线
OPB_Be	OPB	I	[0：(C_OPB_DWIDTH/8-1)]	OPB 字节使能
OPB_Clk	OPB	I	1	OPB 时钟
OPB_DBus	OPB	I	[0：(C_OPB_DWIDTH-1)]	OPB 数据总线
OPB_RNW	OPB	I	1	OPB 只读指示
OPB_Select	OPB	I	1	OPB 片选信号
OPB_rst	OPB	I	1	OPB 复位信号，高电平有效
OPB_seqAddr	OPB	I	1	OPB 连续地址使能
Sln_DBus	OPB	O	[0：(C_OPB_DWIDTH-1)]	从设备数据位宽
Sln_eerAck	OPB	O	1	从设备出错响应
Sln_retry	OPB	O	1	从设备重试
Sln_toutsup	OPB	O	1	从设备超时禁止
Sln_xferAck	OPB	O	1	从设备传输响应
SDRAM_Clk	SDRAM	O	1	SDRAM 时钟信号
SDRAM_CKE	SDRAM	O	1	SDRAM 时钟使能信号
SDRAM_CSn	SDRAM	O	1	SDRAM 片选信号
SDRAM_RASn	SDRAM	O	1	SDRAM 地址行选信号
SDRAM_CASn	SDRAM	O	1	SDRAM 地址列选信号
SDRAM_WEn	SDRAM	O	1	SDRAM 写使能信号
SDRAM_DQM	SDRAM	O	[0：(C_SDRAM_DWIDTH/8-1)]	SDRAM 数据遮掩信号
SDRAM_BankAddr	SDRAM	O	[0：(C_SDRAM_BANK_AWIDTH-1)]	SDRAM Bank 地址信号
SDRAM_Addr	SDRAM	O	[0：(C_SDRAM _AWIDTH-1)]	SDRAM 地址信号
SDRAM_Init_done	SDRAM	O	1	SDRAM 初始化标记信号
SDRAM_Clk_in	SDRAM	I	1	SDRAM 输出时钟，连接到 OPB_CLK
SDRAM_DQ_I	SDRAM	I	[0：(C_SDRAM _DWIDTH-1)]	来自 SDRAM 的输入数据
SDRAM_DQ_Q	SDRAM	O	[0：(C_SDRAM _DWIDTH-1)]	输出到 SDRAM 的数据
SDRAM_DQ_T	SDRAM	O	[0：(C_SDRAM _DWIDTH-1)]	3 态 SDRAM 数据缓冲器

sdram 控制器支持不同数据位宽的读、写模式，且不同位宽以及不同刷新模式下的读写时序和配置都是不同的。例如，突发模式下，16bit 数据位宽的 sdram 控制器读写时序如图 9-13 和图 9-14 所示。

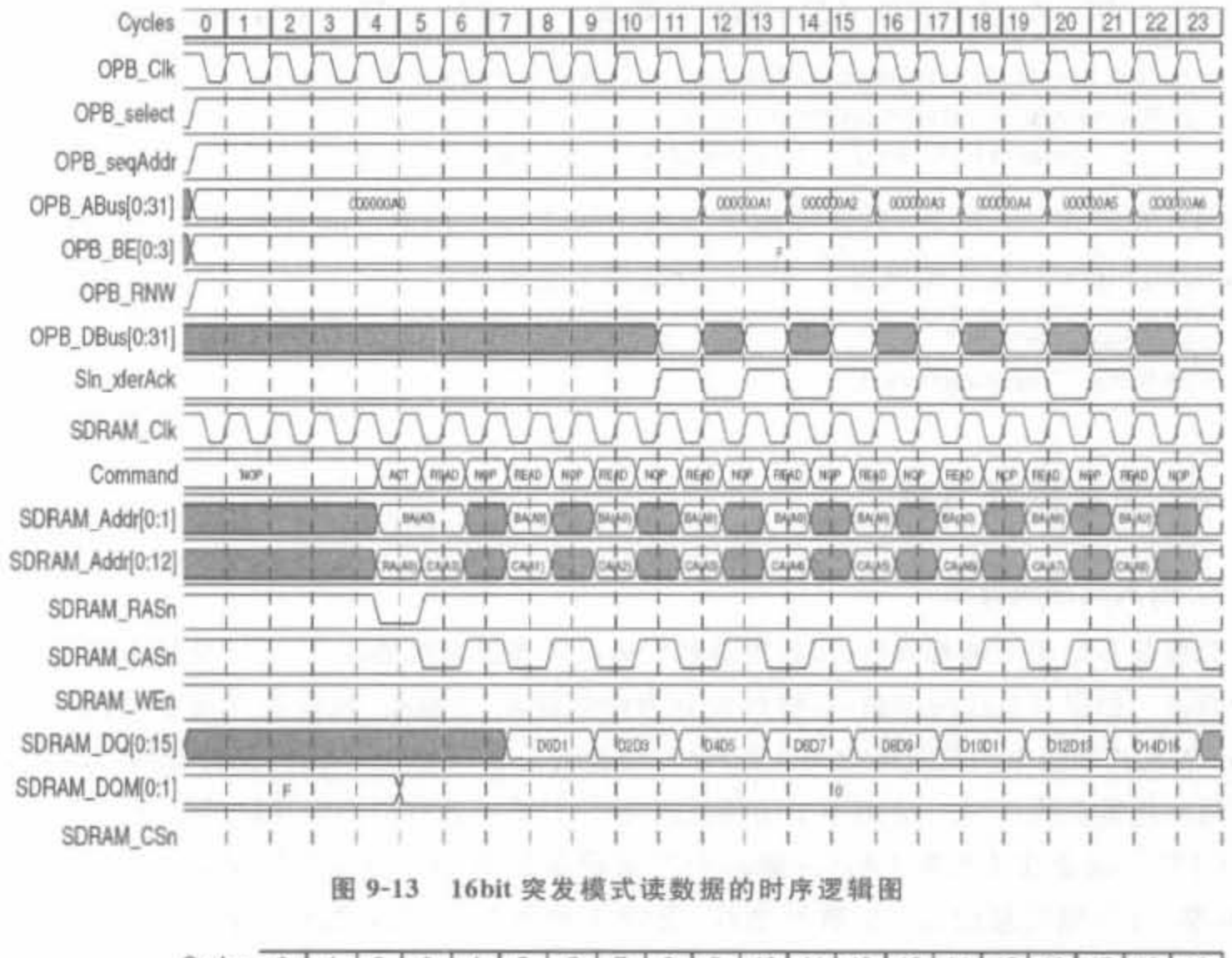


图 9-13　16bit 突发模式读数据的时序逻辑图

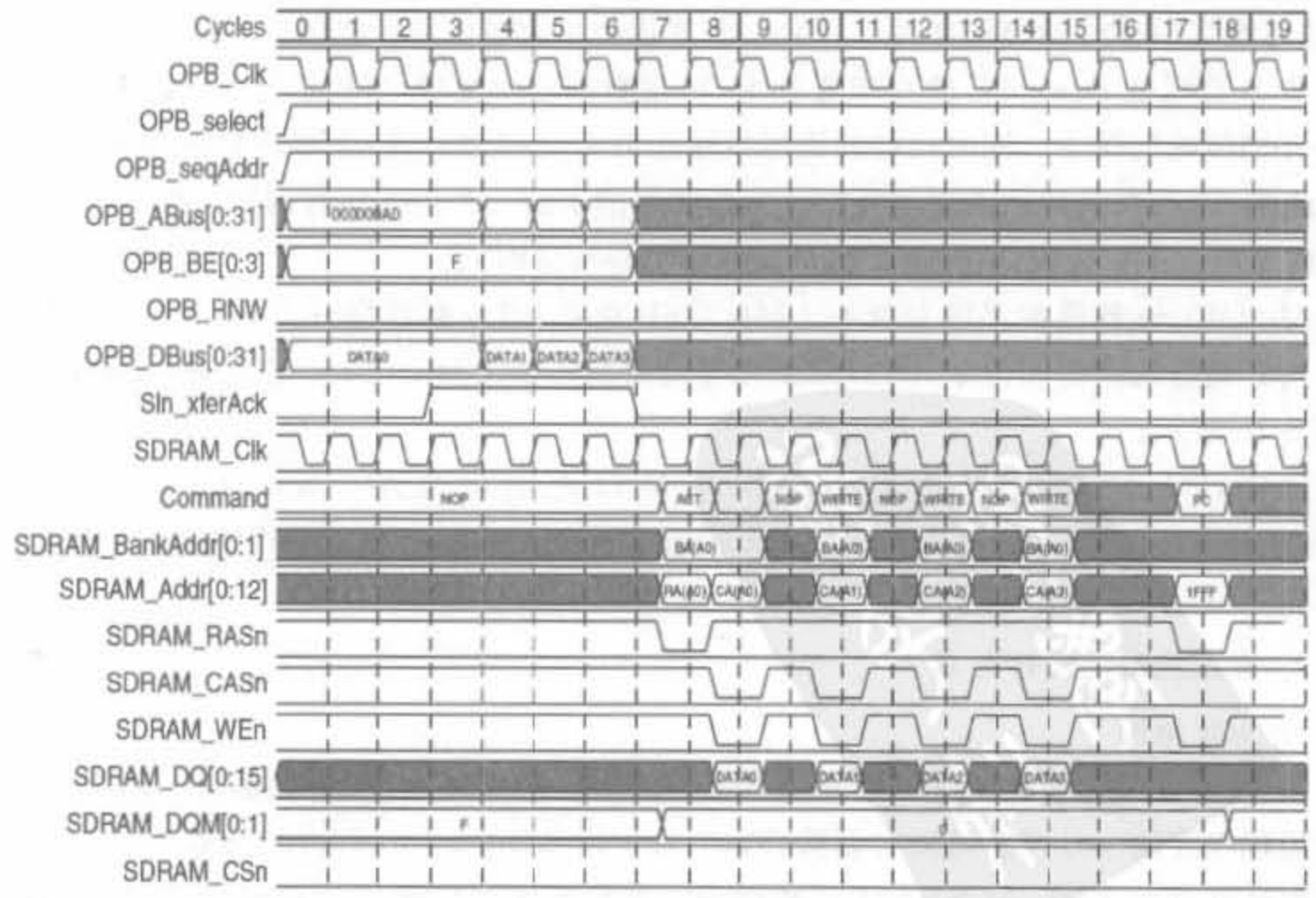


图 9-14　16bit 突发模式写数据的时序逻辑图

2) sdr sdram 控制器驱动

在嵌入式系统中，sdr sdram 控制器对于用户来讲是透明的，即用户在功能使用上没有初始化、刷新等操作，直接用 I/O 读写函数读写相应的地址即可。基本的 I/O 读写函数在文件 xio.h 中定义，其本质就是简单的指针赋值语句，32bit 位宽的读写函数如下所示：

```
#define XIo_In32(InputPtr) ( * (volatile Xuint32 * )(InputPtr))
  #define XIo_Out32(OutputPtr,Value) \
     ( * (volatile Xuint32 * )((OutputPtr)) = (Value))
```

因此在操作 sdram 时，只需添加下面两个头文件。其中，xio.h 定义了端口操作的函数，xparameters.h 文件则给出了 sdram 存储器的基地址。

```
#include " xio.h "
#include " xparameters.h "
```

9.2.4 系统设计方案

1. 嵌入式总线介绍

总线是多组信号的有效组合，处理器和周边设备通过总线地址、数据以及片选控制等信号进行通信。根据对总线的控制，可将外设分为仲裁设备、主设备、从设备以及主/从设备(桥)。Xilinx 嵌入式系统中的总线可分为 3 类，即片外设备总线(OPB)、本地存储器总线(LMB)以及快速简单连接总线(FSL)，分别用于访问低速外设、片内高速存储器以及高速外设。

OPB 总线能在不影响 OPB 仲裁器和其余设备的基础上添加周围设备，同步于系统时钟，共享 32 位地址总线、32 位数据总线，支持主设备和从设备之间的单一周期数据传送。LMB 总线提供单时钟访问片内双口 RAM，并提供了简单的同步协议。

PLB 总线是 PowerPC 的高带宽总线，64 位数据总线宽度，分离的地址、读写数据总线，具备分别传输的能力。同时执行的读写传输能最有效利用总线，在单周期内可传输两个数据；此外，PLB 总线地址通道能叠加一个新的写请求到一个正在执行的写操作上，以及最多 3 个读请求到正在执行的读操作上，从而减少总线反应时间。

MicroBlaze 软核的系统总线有 LMB、OPB 总线，指令和数据分开。LMB 总线用于访问片内存储器的数据和指令，OPB 用于片外设备的连接。MicroBlaze 典型的总线连接方式如图 9-15 所示。

PowerPC 硬核的系统总线分为 PLB、OCM 以及 DCM 总线。OPB 总线可通过总线桥与 PLB 总线连接，从而访问外部速度较低的设备。PowerPC 典型的总线连接方式如图 9-16 所示。

2. 嵌入式系统设计方案

1) 系统架构

基于 MicroBlaze 和 PowerPC 的嵌入式系统和计算器系统类似，满足冯·诺伊曼架构，将应用程序存放在外部非易失存储器(Flash 或 PROM)中，上电后将其加载到片内 BRAM 或映射到外部的 SDRAM/SRAM 存储器中。该类系统的架构是可裁剪的，完全根据需求

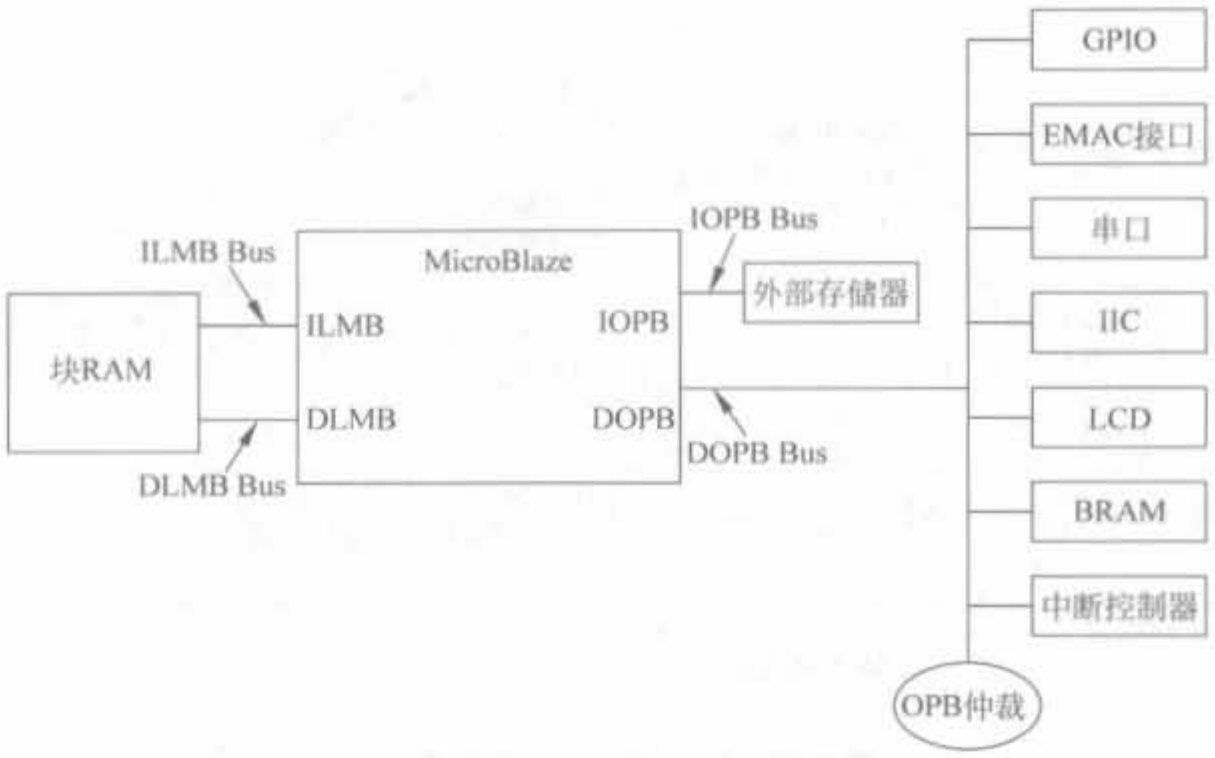


图 9-15　典型的 MicroBlaze 总线连接示意图

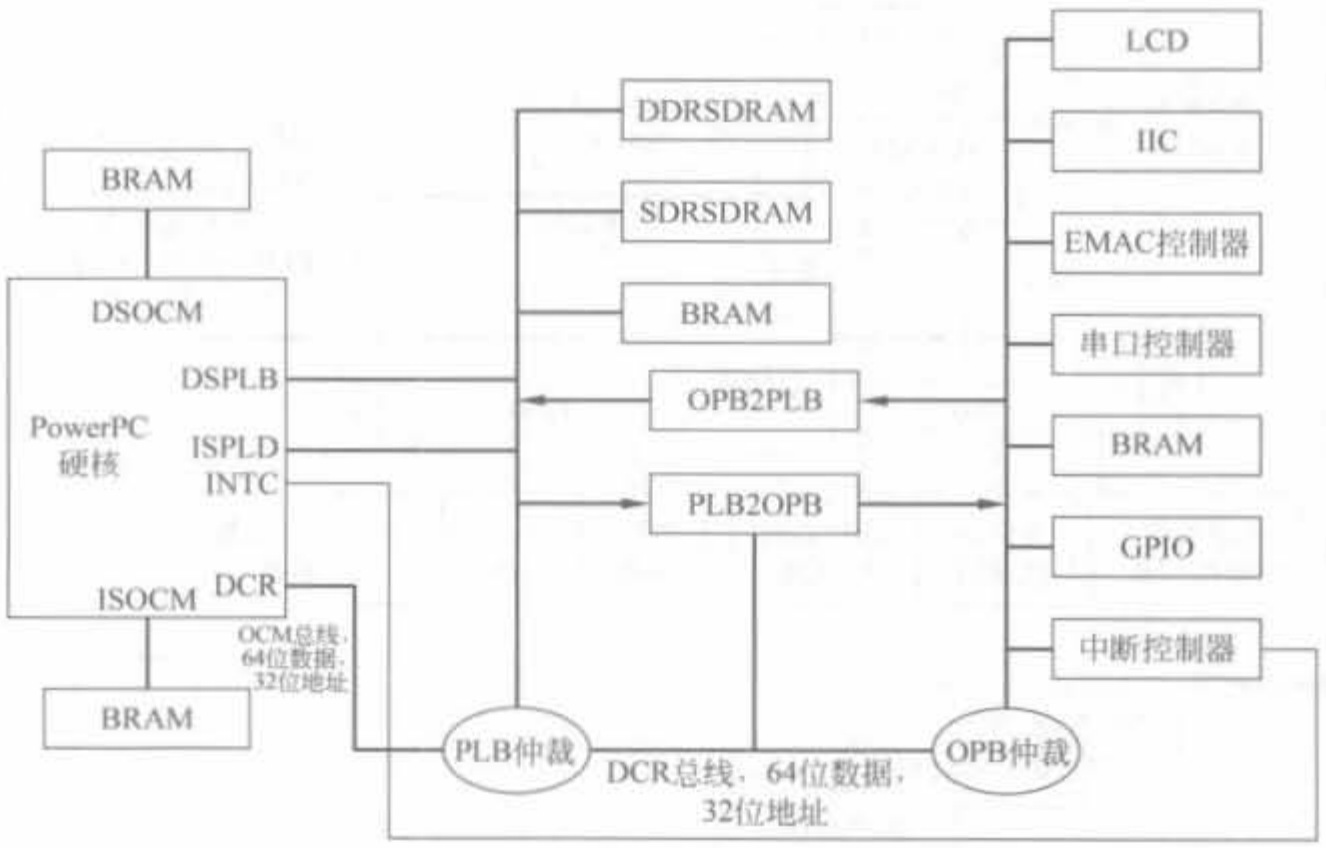


图 9-16　典型的 PowerPC 总线连接示意图

来添加外设，常用于实现专一的功能，具有价格低、速度快、功耗小以及软、硬件协同工作的特点。典型的 MicroBlaze 的 PowerPC 的嵌入式系统架构分别如图 9-17 和图 9-18 所示。

2）系统地址分配

MicroBlaze 采用 32 位地址，其中 0x0000_0000～0x0000_0017 用于特殊处理，0x0000_0018～0xFFFF_FFFF 是用户可用的部分，LMB 存储器从地址 0x0000_0018 开始。

PowerPC 采用 32 位地址，其中每一个 PowerPC 都有其系统引导（boot）区，地址为 0xFFFF_FFFC，默认的可用空间为 0xFFFF_0000～0xFFFF_FFFF。

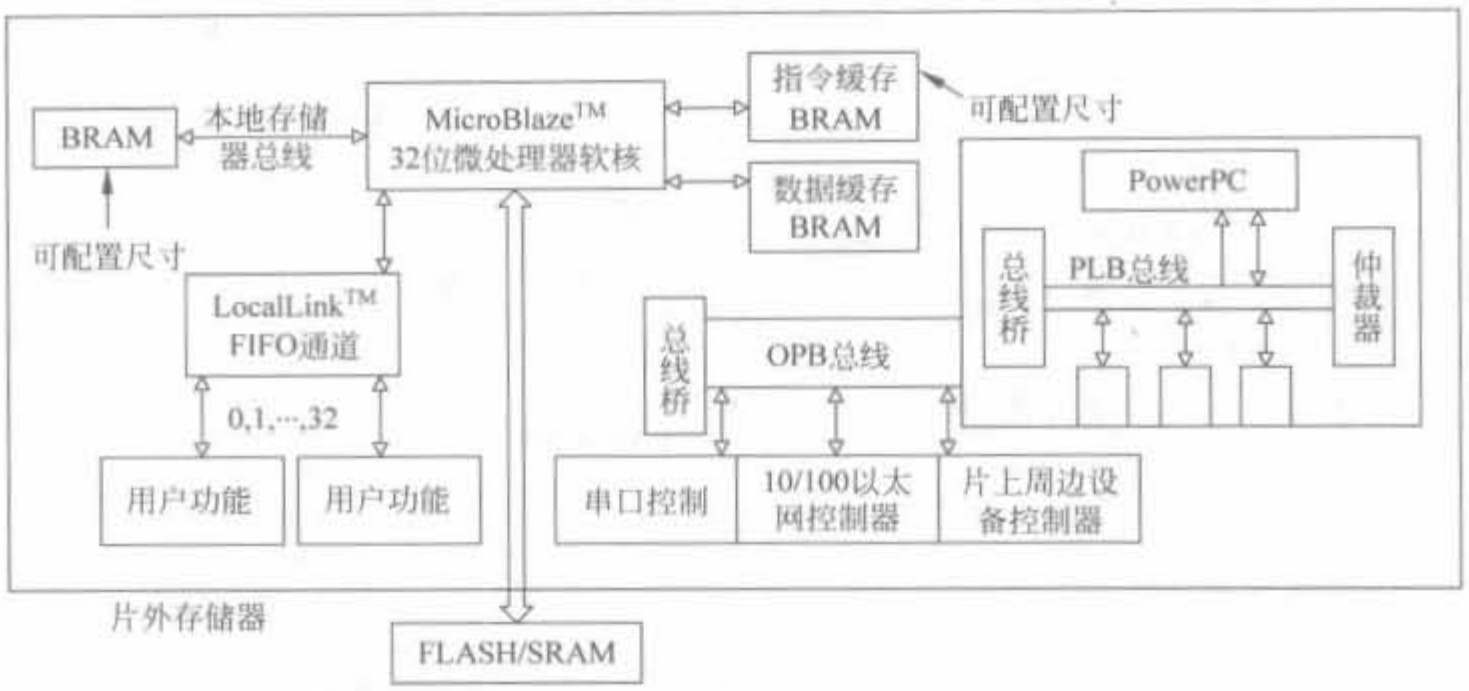


图 9-17　MicroBlaze 系统组成架构

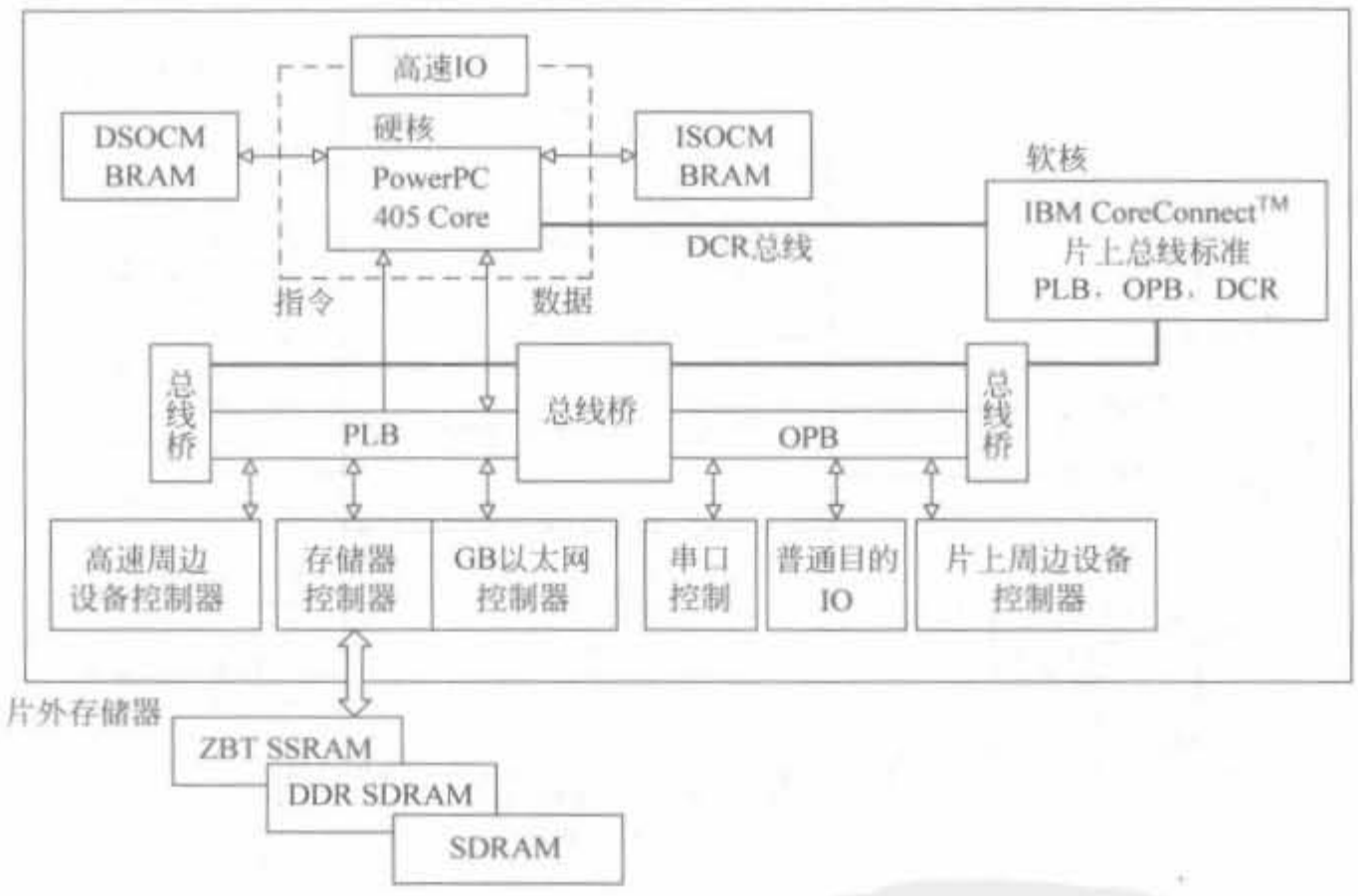


图 9-18　PowerPC 系统组成架构

9.3　EDK 软件基本介绍

Xilinx 的嵌入式开发套件(Embedded Development Kit,EDK)带有许多的工具和 IP,可以用来设计完整的嵌入式处理器系统,并在 Xilinx FPGA 芯片上运行。本节主要介绍基于 EDK 嵌入式设计流程和基本操作。

9.3.1　EDK 的介绍与安装

1. Xilinx 嵌入式开发工具集

一般而言，由于嵌入式系统涉及了软件和硬件的开发以及两者的综合设计，因此其开发是较为复杂的。Xilinx 为了简化基于 FPGA 的嵌入式开发流程，提供了功能强大、操作简单的工具集 ISE 和 EDK。

1）集成软件环境 ISE

ISE 是 Xilinx 公司 FPGA 逻辑设计的基础。在这个环境中，设计者可以进行约束文件的编写、时序分析、逻辑布局布线以及器件编程等。本书第 4 章对 ISE 的使用进行了较为详细的说明，这里就不再赘述。

2）嵌入式开发套件 EDK

EDK 自带了许多工具和 IP，可以用来设计完整的嵌入式处理器系统，主要包括 Xilinx 平台工作室 XPS 和软件开发套件 SDK。需要注意的是，只有安装了 ISE 软件，才能正常运行 EDK，且两者的版本要一致。

下面对 EDK 的组成模块进行简要说明。

（1）Xilinx 平台工作室（Xilinx Platform Studio，XPS）

XPS 用于设计嵌入式处理器系统硬件部分的开发环境或 GUI；是系统设计者构建 Xilinx 嵌入式系统时必用的工具套件。在 XPS 中，可以完成嵌入式系统架构的创建、软件代码的编写、设计的编译以及 FPGA 芯片的硬件配置。

（2）软件开发套件（Software Development Kit，SDK）

SDK 是集成的开发环境，基于 Eclipse。它支持 C/C++，用于嵌入式软件应用的开发和验证。

（3）其他 EDK 部分

EDK 还包括其他的一些部分，如用于 Xilinx 嵌入式处理器的硬 IP；用于嵌入式软件开发的驱动和库；在 MicroBlaze 和 PowerPC 处理器上用于 C/C++软件开发的 GNU 编译器和调试器；有关文档以及一些工程样例等。

2. EDK 软件的安装

EDK 9.1 软件安装的基本硬件要求如下：CPU 在 P Ⅲ以上，内存大于 256MB，硬盘大于 4GB。为了更好地使用软件，至少需要 512MB 内存，CPU 的主频在 2GHz 以上。需要注意的是，安装 EDK 之前，必须安装 ISE 9.1。EDK 的具体安装过程如下：

（1）将安装光盘放进 DVD 光驱，等待其自动运行（如果没有自动运行，直接执行光盘目录下的 Setup.exe 文件程序即可），欢迎界面后会出现如图 9-19 所示的获取注册码对话框，可以通过网站、邮件和传真方式申请注册码。如果已有注册码，单击“Next”按钮继续。

（2）下一个对话框是 Xilinx 软件的授权声明对话框，如图 9-20 所示，选中“I accept the terms of this software license”，单击“Next”后进入注册码输入对话框。输入正确的注册码后，单击“Next”按钮，出现安装路径对话框。单击“Browse”按钮后，选择自定义安装路径。单击“Next”按钮继续。

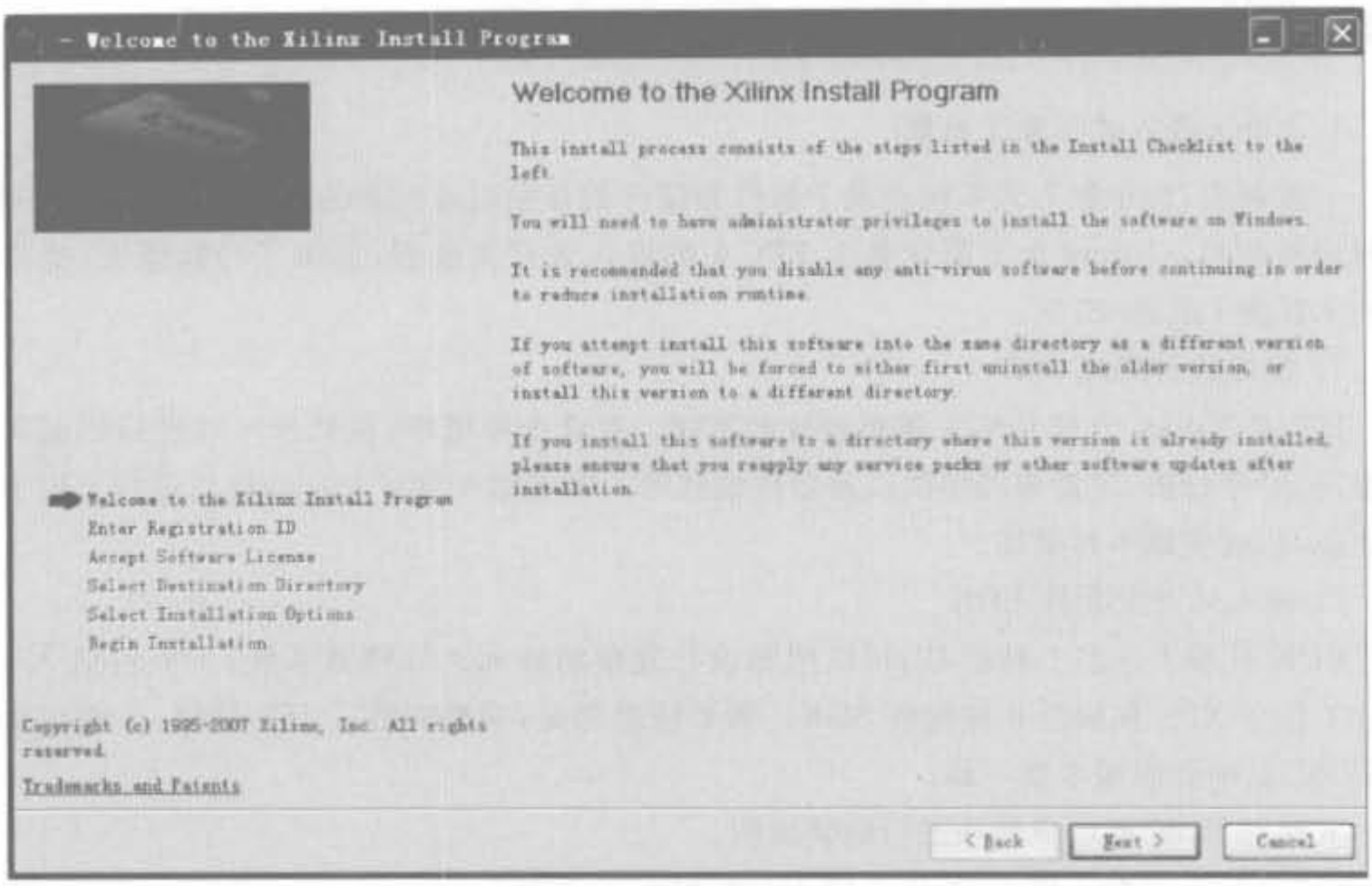


图 9-19　EDK 9.1 安装程序的欢迎界面

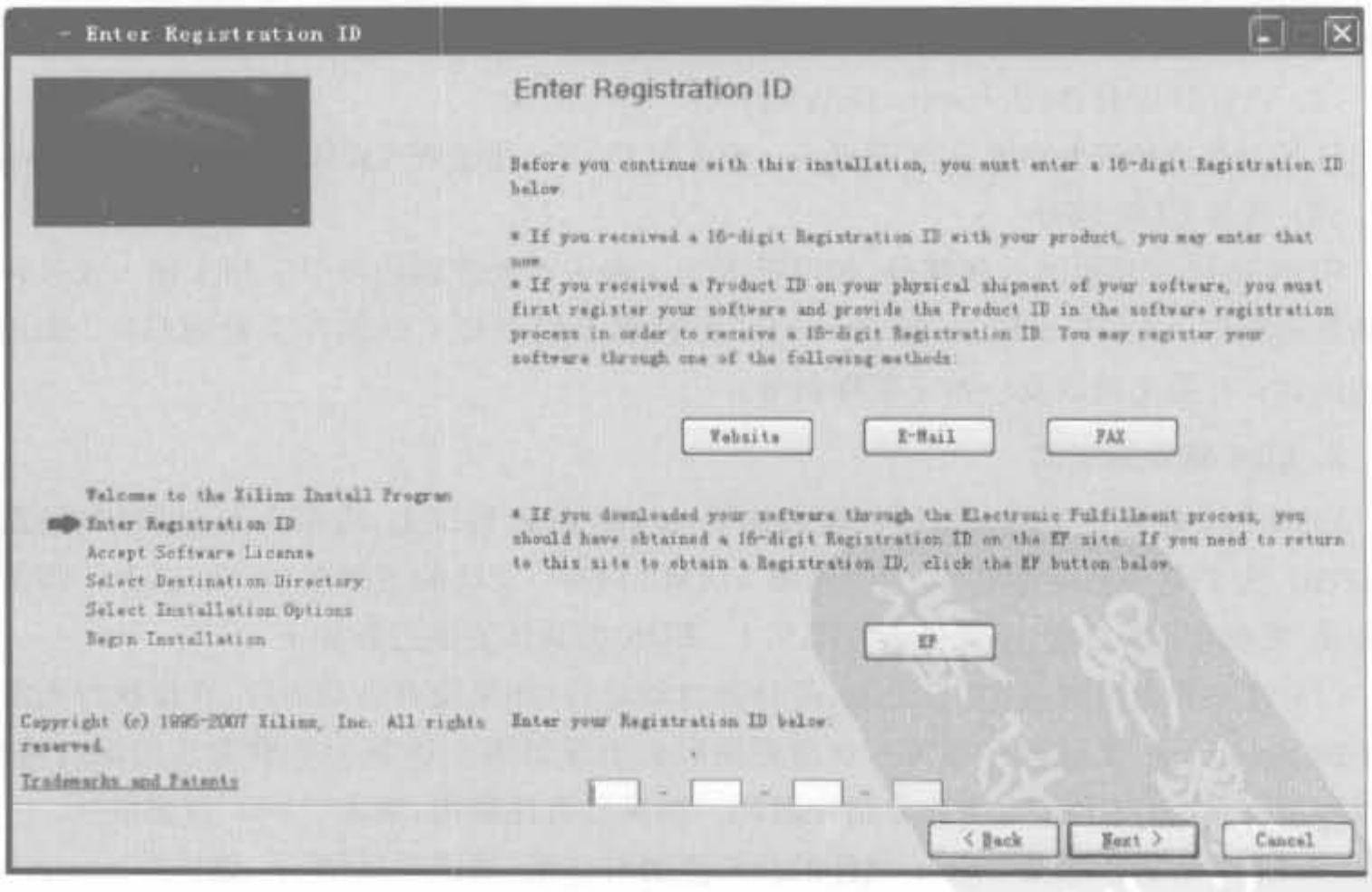


图 9-20　EDK 9.1 安装注册 ID 输入界面

(3) 接下来的几个对话框分别是选择安装设计环境、设置环境变量，这两个步骤保持默认即可。最后进入安装确认对话框，单击“Install”按钮，即可按照用户的设置自动安装 ISE。需要注意的是，在选择安装设计环境时，用户需要选择自己使用的芯片所对应的模块，这样才能在开发中使用这些模块，一般选择“Select All”。

(4) 安装完成后，会在桌面以及程序菜单中添加 EDK 的快捷方式。双击即可进入 EDK 集成开发环境。

9.3.2　EDK 设计的实现流程

1. 基于 EDK 的开发流程

一个完整的嵌入式设计流程包括硬件设计和调试、软件设计与调试，各个步骤相对独立，但又相辅相成。由于嵌入式应用场合多样，且软、硬件都可裁剪，因此并不是每个设计都要完成所有的步骤。图 9-21 为基于 EDK 的嵌入式设计的简化流程图。

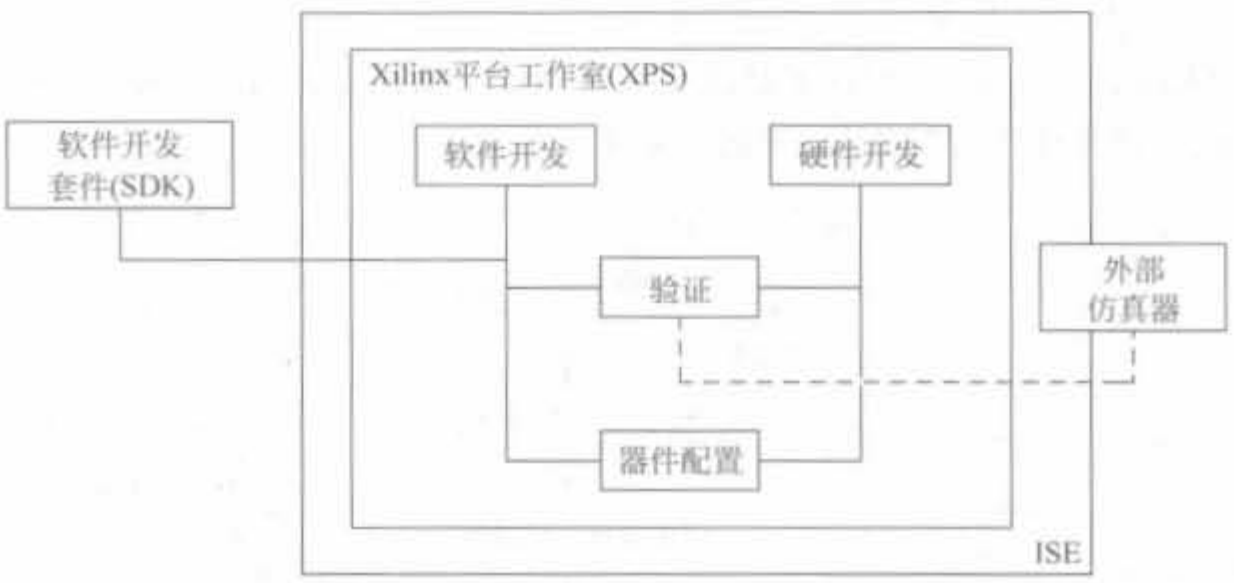


图 9-21　简化的嵌入式设计流程图

通常，ISE FPGA 开发软件在后台运行，XPS 工具调用 ISE 软件提供的功能。XPS 主要用于嵌入式处理器硬件系统的开发。微处理器、外围设备以及这些组件之间的连接问题，还有它们各自的属性设置都在 XPS 里进行。简单的软件开发可以在 XPS 里完成，而对于更复杂的应用开发和调试，Xilinx 推荐使用 SDK 工具。硬件平台的功能验证可以通过硬件描述语言 HDL 仿真器完成。XPS 提供了行为级、结构级以及定时精确级 3 种类型的仿真。验证过程结构由 XPS 自动产生，其中包括仿真的 HDL 文件。设计者只需要输入时钟时序、重配置信息以及一些应用代码即可。仿真细节将在下面的内容讲述。完成设计后，在 XPS 中将 FPGA 比特流和可执行可链接格式文件下载，就可以进行目标器件的配置。

完整的 EDK 开发流程如图 9-22 所示，其主要步骤有：

(1) 创建硬件平台：利用 XPS 的板级开发包向导(BSB Wizard)快速构建设计的硬件平台，是 EDK 设计的第一步。

(2) 添加 IP Core 以及用户定制外设：在 XPS 中添加所需的 IP Core，对于 XPS 库中缺

少的模块，需要用户自行设计。同样，XPS 提供了建立用户自定义外设的向导，可简化该过程。

（3）生成仿真文件并测试硬件系统：生成硬件系统的仿真文件，可选择行为级、结构级以及时序级仿真，利用 ModelSim 等工具测试系统，特别是用户自定义的外设；如果测试失败，需要返回上一步修改。

（4）生成硬件比特流：生成硬件网表和比特流文件，这个步骤类似于传统 FPGA 设计的综合、布局布线、生成编程文件这 3 个操作。

（5）开发软件系统：针对软件需求编写硬件代码，确定软件的操作系统、库、外设驱动等属性，针对每个应用软件工程，设置编译器、优化级别、使用的连接文件等信息。等设置完成后，编译生成. elf 格式的可执行代码。

（6）合并软、硬件比特流：编译软件后，需要将软、硬件可执行文件合并在一起，生成最终的二进制比特文件。

（7）下载：使用 JTAG 编程电缆或编程器将更新后的最终比特流烧写到 FPGA、PROM、FLASH 以及 CF 卡。

（8）在线调试：可利用 XMD 工具或 ChipScope 工具调试，通过 JTAG 编程电缆在线调试，下载可执行软件代码控制执行，并监控相关系信息。

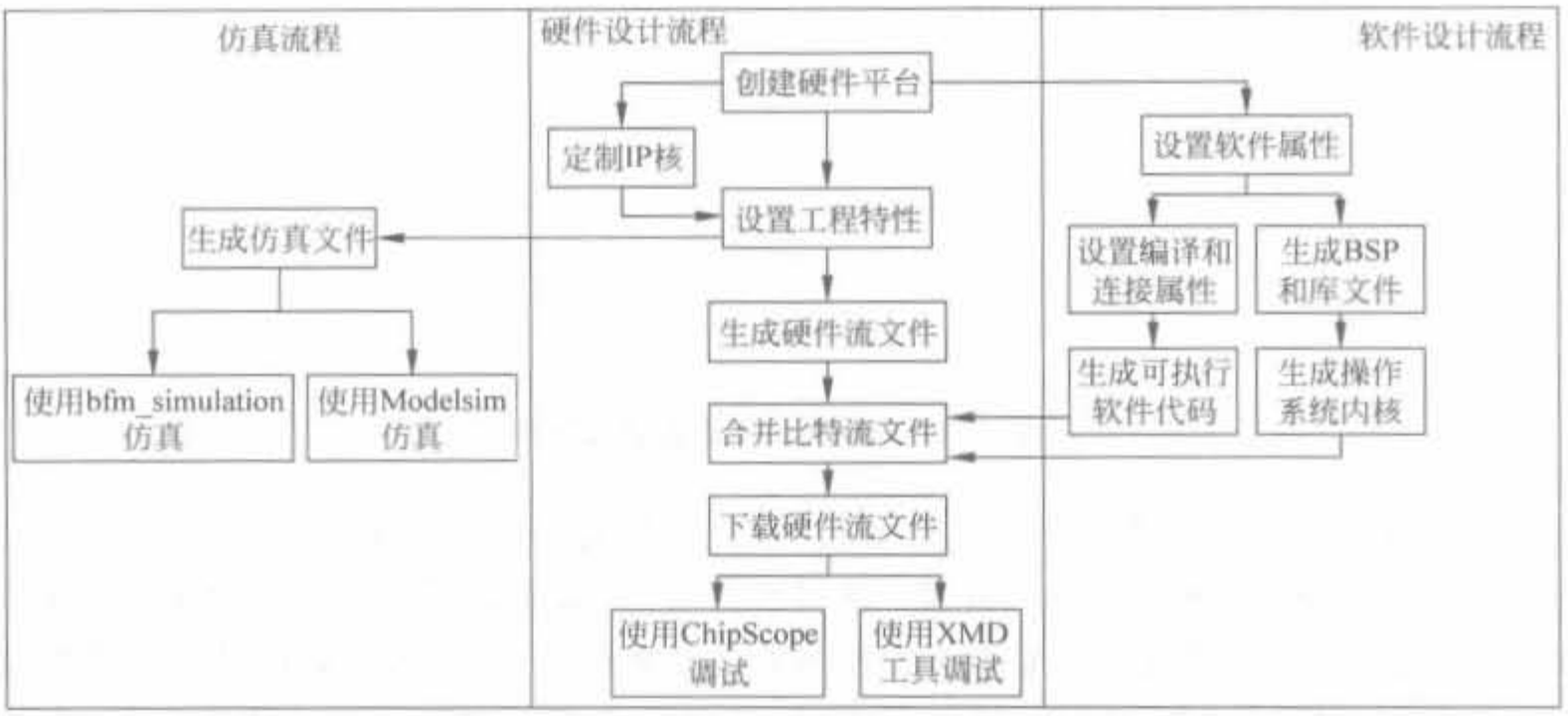


图 9-22　完整的嵌入式设计流程图

2. EDK 设计比特文件的组成

如前所述，最终下载到 FPGA 的嵌入式比特流文件是软、硬件比特流合并在一起的，详细的组成部分如图 9-23 所示。硬件部分比特流包括 MHS 文件、用户自定义 HDL 代码，二者经过综合实现后，产生. ngc 网表，生成硬件系统的比特流文件；软件系统包括 MSS 文件、用户. c/cpp/asm 文件，通过 GCC 编译器，生成目标文件. obj，再经过连接合成软件系统的比特流文件；最后通过 Data2MEM 过程，将软、硬件比特流合成完整系统比特流文件，通过 JTAG 链路下载到 FPGA 芯片中。

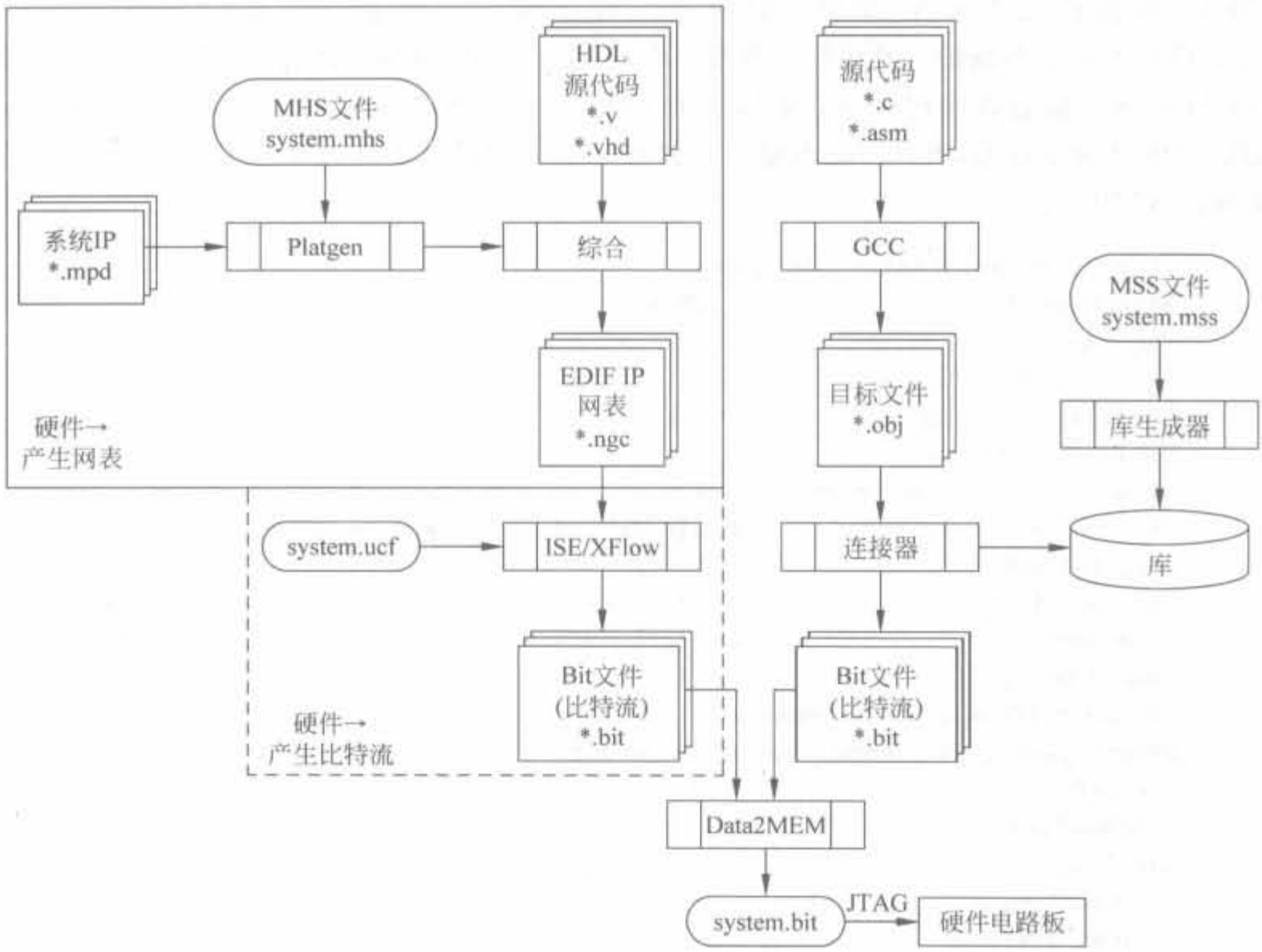


图 9-23 EDK 配置比特文件的组成结构

9.3.3 EDK 的文件管理架构

对于嵌入式应用来讲，软、硬件协同开发是非常重要的，虽然 EDK 提供了 XPS 工具和 SDK 工具这两个图形化平台，但仍以文件结构管理为基础，图形化平台只是方便用户操作的，所有的设置内容都会写入相应的文件中。了解相应格式的文件是掌握 EDK 工具操作的必备知识。本节将介绍 EDK 构建嵌入式系统软、硬件系统的文件，以及管理、存储数据文件的模式和流程。

1. **板级支持包** Board Support Package(BSP)

BSP 为每个处理器定义了系统的硬件元素。它包括不同的嵌入式软件元素，如软件驱动文件、所选的库、标准 I/O 设备、中断处理程序以及其他相关的特征。因此，在将处理器、外围设备组装到硬件系统上，且定义了地址映射后，可以利用 XPS 来产生 BSP。

与硬件组件类似，XPS 允许用户规定软件平台部分并管理软件应用。XPS 的应用标签包含用户所需要的工具和命令。

2. **XMP 工程文件**

EDK 设计的工程文件就是. xmp 格式，定义了 EDK 工具的版本、相关的硬件配置文件

(MHS)和软件配置文件(MSS)、目标器件的类型、软件的源码和库位置等信息。用UltraEditor等文本编辑工具可打开查看；ProgStart为程序的起始位置，对应内存的起始地址；StackSize标志着堆栈信息，与MSS中的信息是一致的。总的来讲，XMP文件是由XPS软件自动生成的，用户一般不要自行修改。下面通过在实际的XMP文件中添加注释来解读XMP文件。

```
# Please do not modify this file by hand
  XmpVersion: 9.1                    # EDK 版本
  VerMgmt: 9.1
  IntStyle: default
  MHS File: system.mhs
  MSS File: system.mss
  NPL File: projnav/system.ise
  Architecture: virtex2p             # 目标器件家族
  Device: xc2vp30                    # 目标器件型号
  Package: ff896                     # 目标器件封装
  SpeedGrade: -7                     # 目标器件速度等级
  UseProjNav: 0
  PNImportBitFile: projnav/system.bit
  PNImportBmmFile: implementation/system_bd.bmm
  UserCmd1:
  UserCmd1Type: 0
  UserCmd2:
  UserCmd2Type: 0
  TopInst: system_i
  GenSimTB: 0
  InsertNoPads: 0
  WarnForEAArch: 1
  HdlLang: VHDL                      # 所使用的 HDL 语言类型
  Simulator: mti                     # 仿真工具为 ModelSim
  SimModel: BEHAVIORAL               # 仿真模型为功能仿真模型
  MixLangSim: 1
  UcfFile: data/system.ucf
  FpgaImpMode: 0
  ShowLicenseDialog: 1
  Processor: ppc405_0                # 处理器类型为硬核 PowerPC
  BootLoop: 0
  XmdStub: 0
  Processor: ppc405_1
  BootLoop: 0
  XmdStub: 0
  SwProj: config_decoder_revb
  Processor: ppc405_0                # 下面是各类文件的说明
  Executable: config_decoder_revb/executable.elf
  Source: config_decoder_revb/src/xi2c_1.c
  Source: config_decoder_revb/src p_config_decoder.c
  Source: config_decoder_revb/src/uart.c
  Header: config_decoder_revb/src/xi2c_1.h
  Header: config_decoder_revb/src/uart.h
  DefaultInit: executable
```

```
InitBram: 1
Active: 1
CompilerOptLevel: 2
GlobPtrOpt: 0
DebugSym: 1
ProfileFlag: 0
ProgStart: 0X45000000          #程序起始地址
StackSize:                     #堆的大小
HeapSize:                      #栈的大小
LinkerScript:
ProgCCFlags:
CompileInXps: 1
NonXpsApp: 0
```

3. MHS 文件和其他相关的硬件平台元素

MHS 文件是硬件结构描述文件，定义了系统结构、外围设备和嵌入式处理器，也定义系统的连通性、系统中每个外围设备的地址分配和对每个外围设备的可配选项。该文件可随意更改，在图形界面中对硬件结构的任何改动，都要写入该文件中。同样，对于高级用户，可通过直接修改 MHS 文件来代替 XPS 中的图形操作。MHS 文件严格按照分层设计的思想，每个硬件模块都是一个独立的组件，再通过上层模块连接起来，形成整个系统。下面通过在实际的 MHS 文件中添加注释来解读 MHS 文件。

```
PARAMETER VERSION = 2.1.0    #定义了参数集版本
 PORT sys_clk_pin = dcm_clk_s,CLK_FREQ = 50000000,DIR = I,SIGIS = CLK
 #定义了系统时钟，大小为 50MHz，方向为输入，标记为时钟信号
 PORT sys_rst_pin = sys_rst_s,DIR = I,RST_POLARITY = 1,SIGIS = RST
 PORT sg2fsl_r = xlsg_ifacesg2fsl_r,DIR = I
 PORT fsl2sg_ctrl = xlsg_ifacefsl2sg_ctrl,DIR = O
 PORT fsl2sg_data = xlsg_ifacefsl2sg_data,DIR = O,VEC = [0: 31]
 PORT fsl2sg_exists = xlsg_ifacefsl2sg_exists,DIR = O
 PORT sg2fsl_ctrl = xlsg_ifacesg2fsl_ctrl,DIR = I
 PORT sg2fsl_data = xlsg_ifacesg2fsl_data,DIR = I,VEC = [0: 31]
 PORT sg2fsl_w = xlsg_ifacesg2fsl_w,DIR = I
 PORT fsl2sg_full = xlsg_ifacefsl2sg_full,DIR = O

#子模块定义，以 BEGIN 和 END 定义段
BEGIN microblaze                          #定义名称
 PARAMETER INSTANCE = microblaze_0        #定义例化名称
 PARAMETER HW_VER = 6.00.a                #定义软件版本
 PARAMETER C_USE_FPU = 0
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_NUMBER_OF_PC_BRK = 2
 PARAMETER C_AREA_OPTIMIZED = 1
 PARAMETER C_FSL_LINKS = 1
 #下面开始全是端口信号
 BUS_INTERFACE DLMB = dlmb
 BUS_INTERFACE ILMB = ilmb
 BUS_INTERFACE DOPB = mb_opb
 BUS_INTERFACE IOPB = mb_opb
```

```
 BUS_INTERFACE SFSL0 = xlsysgenfsl_v20_1
 BUS_INTERFACE MFSL0 = xlsysgenfsl_v20_0
 PORT DBG_CAPTURE = DBG_CAPTURE_s
 PORT DBG_CLK = DBG_CLK_s
 PORT DBG_REG_EN = DBG_REG_EN_s
 PORT DBG_TDI = DBG_TDI_s
 PORT DBG_TDO = DBG_TDO_s
 PORT DBG_UPDATE = DBG_UPDATE_s
END
......
```

4. MSS 文件和其他软件平台元素

除了硬件部分利用 MHS 文件来描述硬件元素，XPS 还利用了微处理器软件规范(MSS)文件进行一个类似的软件系统描述。此 MSS 文件和用户的软件应用一起，组成了描述嵌入式系统软件部分的主要源文件。利用这些文件以及 EDK 的库和驱动器，XPS 就可以编译用户的应用程序。而编译后的软件程序生成可执行可链接格式(ELF)的文件。

和 MHS 文件一样，高级用户也可通过直接修改 MSS 文件达到更改软件配置的目的。下面通过在实际的 MSS 文件中添加注释来解读 MSS 文件。

```
PARAMETER VERSION = 2.2.0    #定义参数版本

BEGIN OS      #定义了操作系统，以 BEGIN 和 END 定义段
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = RS232_DCE
 PARAMETER STDOUT = RS232_DCE
END

BEGIN PROCESSOR    #定义处理器类型，以 BEGIN 和 END 定义段
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = microblaze_0
 PARAMETER COMPILER = mb-gcc
 PARAMETER ARCHIVER = mb-ar
 PARAMETER XMDSTUB_PERIPHERAL = debug_module
END

BEGIN DRIVER        #定义驱动，以 BEGIN 和 END 定义段
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = debug_module
END
......
```

5. UCF 文件

EDK 工具和 ISE 一样，都通过 UCF 文件来添加信号的管脚约束与时序约束。在 EDK 设计中，UCF 文件用来指定管脚的功能是最常用的，可通过文本编辑器修改，相关的语法和

ISE 中是一致的，如 4.4.2 节所示。

6. CMD 文件

EDK 在配置 FPGA 时仍通过调用 iMPACT 软件来完成的，但没有相应的图形化界面，而是通过命令行的方式实现，将所需要的命令写在同一个文本中，然后采用批处理的方式实现。该文件就是.CMD 文件。CMD 文件可通过文本编辑器修改，其基本内容如下所示：

```
setMode - bscan
setCable - p auto
identify
assignfile - p 1 - file implementation/download.bit
program - p 1
quit
```

其中，setMode 用于设定边界扫描模式；setCable 用于设置编程电缆类型；assignfile 用于指定编程使用的比特流文件；program 为器件编程的指令；quit 为退出命令。

一般来讲，CMD 文件中修改最多的是 assignfile 指令和 program 指令，其后面的数字为器件在 JTAG 链上的位置。在实际中，设计人员要首先通过 iMPACT 软件获取到详细的 JTAG 链信息，然后将相应的数字填入 CMD 文件。其中，在 iMPACT 显示的 JTAG 链中，最左端的器件标号为 1，以后的器件编号从左向右依次递增。

7. 平台工作室软件开发套件(SDK)

平台工作室 SDK 方便了嵌入式软件应用工程的开发。SDK 有自己的 GUI，基于 Eclipse 开放资源工具组件。此 SDK 是 XPS 的补充部分，即利用 SDK 可以开发外围设备和处理器元件所使用的软件，而这些设备和元件都连接在 XPS 中。

对于每个复杂软件应用，用户都应该建立一个 SDK 工程。工程包括了用户的 C/C++ 源文件、可执行输出文件以及相应的功能文件，如用来建立工程的“make”文件。通常，每个 SDK 工程目录都位于嵌入式系统的 XPS 工程目录树下，每个 SDK 工程只产生一个可执行文件<project_name>.elf。因此对于一个 XPS 嵌入式系统而言，可能有多个相应的 SDK 工程。

9.4　XPS 软件的基本操作

XPS 软件是完备的 Xilinx 嵌入式系统开发工具，可完成嵌入式系统的全部硬件开发和软件开发，将软、硬件开发完美地封装在一起，是业界优秀的开发系统。本节主要介绍 XPS 软件的基本操作，为掌握后续内容做好铺垫。

9.4.1　XPS 的启动

XPS 有两种启动方式，一种是直接单击“开始”→“程序”→“Xilinx Platform Studio 9.1i”→“Xilinx Platform Studio PACE”即可启动；另一种是在 ISE 中通过双击 Embedded Processor 类型的源文件来打开。前者专门用于设计完备的嵌入式系统，也是本节主要讲述

的方式；后者可将嵌入式设计作为 ISE 设计的一个子模块，在 9.5.3 节介绍。

9.4.2 利用 BSB 创建新工程

在了解了 EDK 的基本概念以及完成软件安装后，接下来介绍如何利用 EDK 来开发嵌入式系统。首先需要利用基本系统创建器(Base System Builder，BSB)向导来快速创建一个设计，再对其进行定制。BSB 是帮助用户快速建立系统的软件工具。当用户希望创建一个新的系统时，XPS 会自动调用 BSB。在 BSB 中，可以创建工程文件，选择开发版，选择和配置处理器以及 I/O 设备，添加内部外围设备，生成系统报告等。BSB 向导会自动完成以下工作：

1. 生成顶层工程文件(.xmp 文件)

Xilinx 微处理器工程(Xilinx Microprocessor Project，XMP)文件是所开发嵌入式系统的顶层文件描述。所有 XPS 工程信息都在 XMP 文件中得以保存，此文件包括微处理器硬件规范(Microprocessor Hardware Specification，MHS)和微处理器软件规范(Microprocessor Software Specification，MSS)文件的存储位置等，有关 MHS 和 MSS 文件将在后面阐述。XMP 文件同时包括 XPS 将进行编译的 C 源文件和头文件的信息，以及 SDK 编译的可执行文件的信息。

2. 选择/新建电路板

这里包括两个选项：指定的目标板和用户自己设计的板。如果选择前者，则 BSB 允许用户选择板上的外围设备，且其 FPGA 端口可以自动地匹配板子，同时创建一个可以下载到板子上运行的完整平台和测试应用；而对于后者，用户可以基于一些已有的处理器核和外围设备核，按照需要添加处理器和外围设备。

3. 选择并配置处理器

可以选择的处理器有 MicroBlaze 和 PowerPC，同时可以选择器件类型、封装、速率等级、参考时钟频率及处理器总线时钟频率等。

4. 选择并配置多个 I/O 接口

BSB 可以决定在用户预定义板子上，哪些外部存储和 I/O 设备是有效的，并且对于指定的器件还可以选择波特率、外围设备类型、数据比特数及校验等。

5. 添加内部外围设备

外围设备包括芯片上存储控制器和计时器等。BSB 允许用户添加需要的外围设备。

6. 设定软件

可以在 BSB 中对标准的输入/输出器件进行说明，用户还可以选择希望 XPS 产生的 C 应用样例。每个应用程序都包括一个链接脚本。这里用户所选择的应用样例包括存储测试、外围设备测试，或者两者皆有。

7. 查看创建的系统

完成以上选择后，BSB 将显示已经生成的系统。用户可以选择产生这个工程，或者是返回到前述步骤进行设置的修改。下面以加载 Xilinx Spartan_3E_RevD 开发板(该开发板

的配置文件存放在 EDK 安装目录下的“board\Xilinx\boards\Xilinx_Spartan3E_RevD\data”文件夹中)为例,详细介绍利用 BSB 新建工程的全部流程。

例 9-1　通过 BSB 向导建立 Xilinx Spartan_3E_RevD 开发板的工程。

(1) 启动 XPS。当双击 XPS 的快捷方式后,BSB 向导就会自动打开,其界面如图 9-24 所示。选中“Base System Builder wizard(recommended)”选项,并单击“OK”按钮。

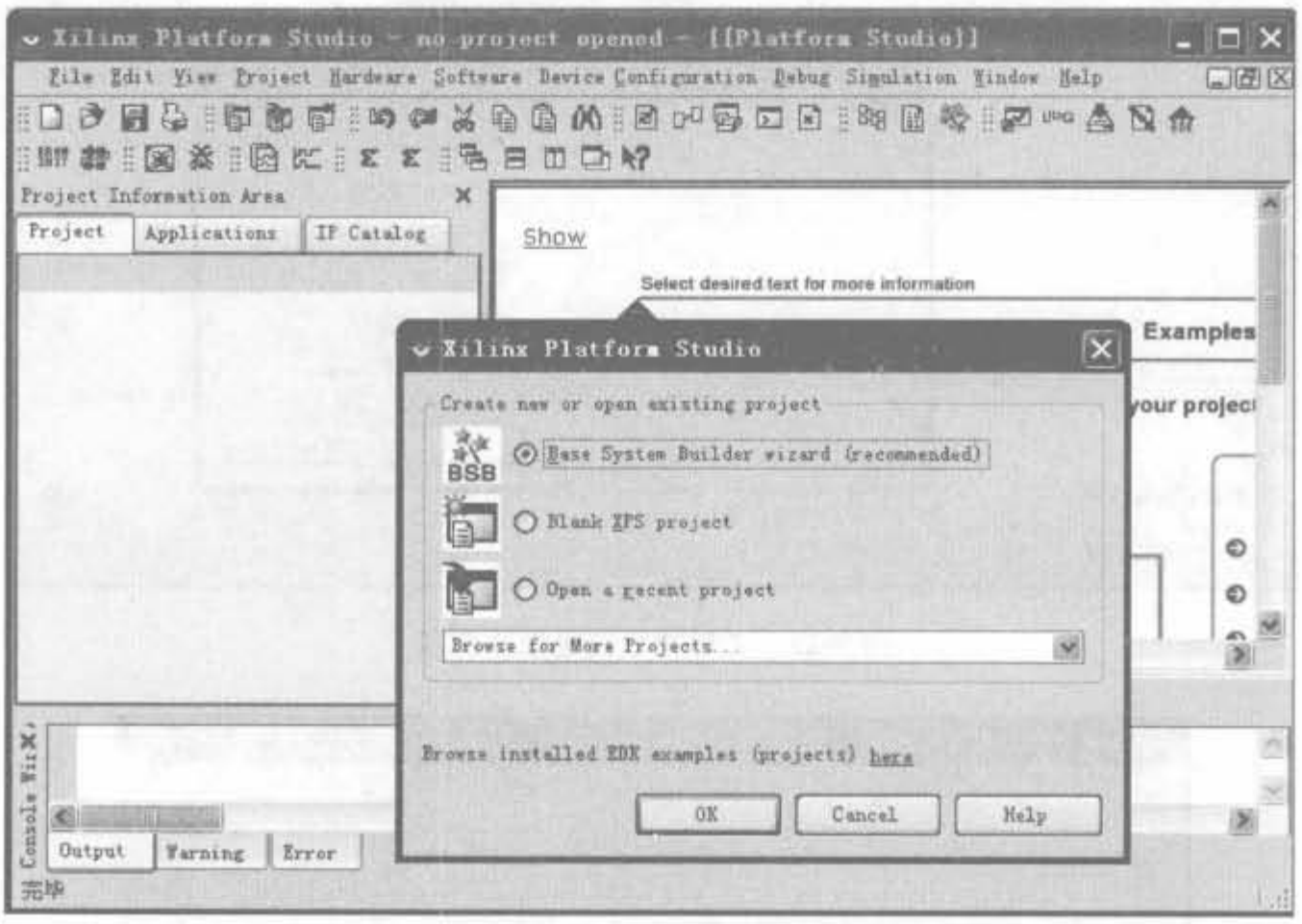


图 9-24　BSB 向导的启动画面

(2) 选择工程路径。在弹出的提示界面的“Project File”下的输入框中填入工程存放路径,或单击“Browse”按钮来选择合适的工程路径。在本例中,选择工程目录为“E:/work/xilinx/edk”,并将工程命名为 spartan3e500e,如图 9-25 所示。单击“OK”按钮,在弹出的对话框中选择“I would like to create a new design”,并单击“Next”按钮。

(3) 在弹出的板型选择对话框中,在“Select board”选中“I would like to create a system for the following development board”,在“Board vendor”的下拉框中选择“Xilinx”,在“Board name”的下拉框中选择“Spartan-3E Starter Board”,在“Board revision”的下拉框中选择“D”,如图 9-26 所示。如果使用用户自定义的开发板,则选择“I would like to create a system for a custom board”,单击“Next”按钮。

(4) 进入处理器选择界面。对于内部集成了 PowerPC 内核的 FPGA 芯片(Virtex-2 Pro 以上部分型号)可以选择 PowerPC 或 MicroBlaze,否则只能选择 MicroBlaze。由于本例选用的是 FPGA 的 Spartan-3E 系列,不支持 PowerPC,所以只能选择 MicroBlaze 软核,如图 9-27 所示,单击“Next”按钮。

(5) 配置处理器。配置 MicroBlaze 的参考时钟、总线处理时钟频率、处理器调试接口以及片上存储空间的大小,都使用默认值,如图 9-28 所示。时钟频率和参考时钟与硬件设计保持一致,定为 50MHz。“On chip H/W Debug Module”就是利用 JTAG 端口进行处理器

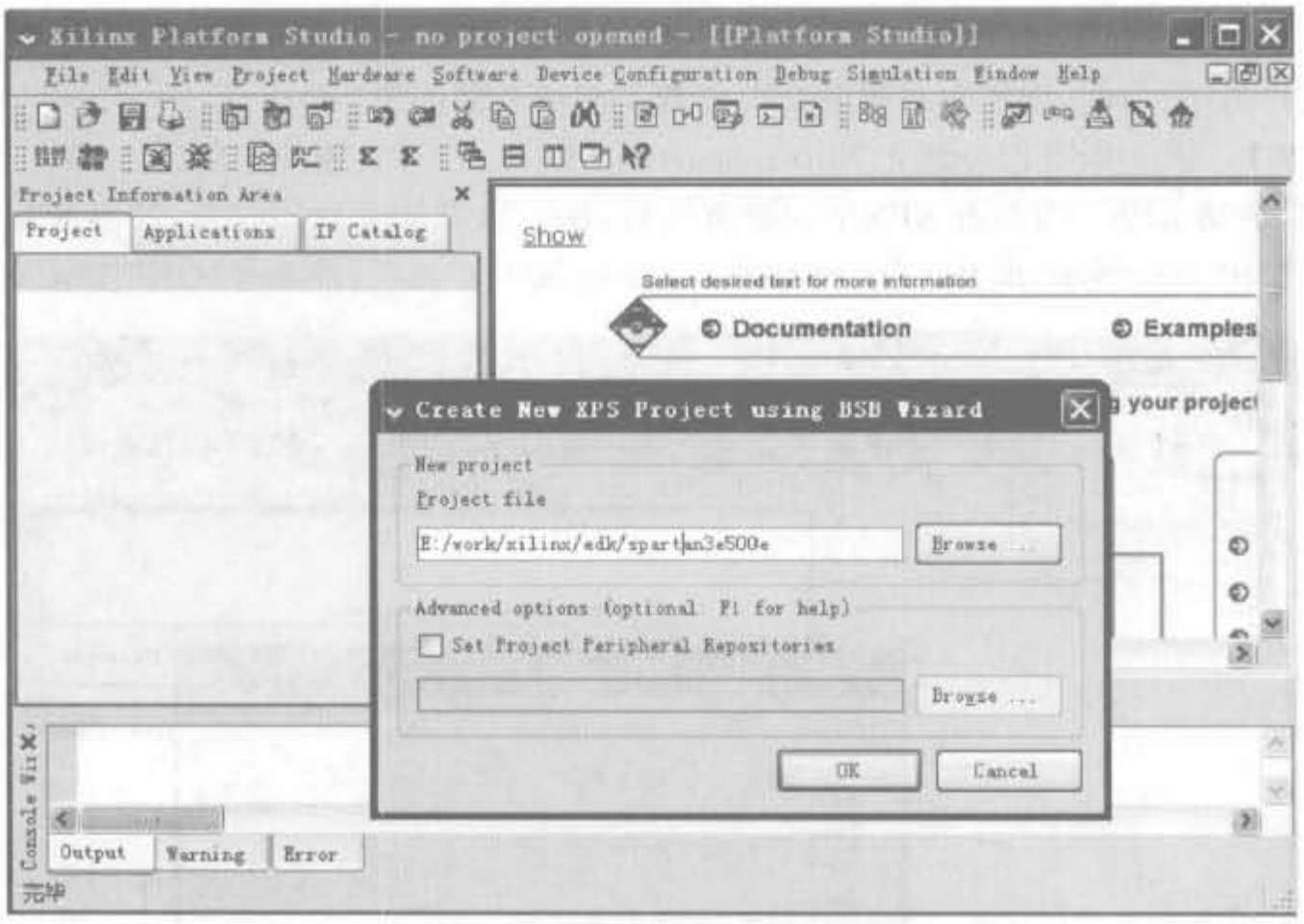


图 9-25　工程存放路径选择界面

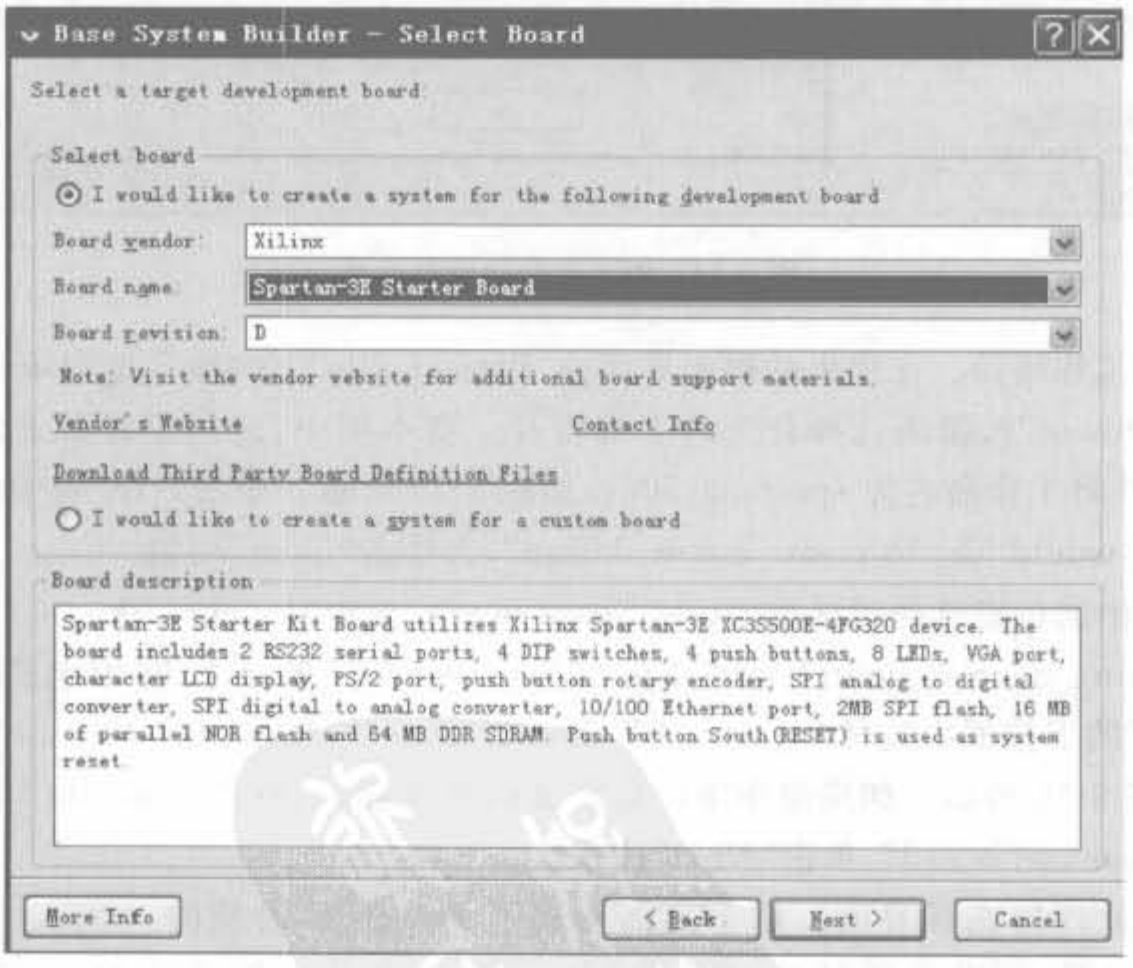


图 9-26　开发板选择界面

的调试。片上存储器由 FPGA 芯片中内嵌的块 RAM 组成，选定大小为 8KB。然后单击“Next”按钮。MicroBlaze 和 PowerPC 的工作时钟都是系统时钟经过 DCM 模块倍频后得到的，在一般情况下，建议 MicroBlaze 的时钟不超过 100MHz，PowerPC 的时钟不超过 250MHz。

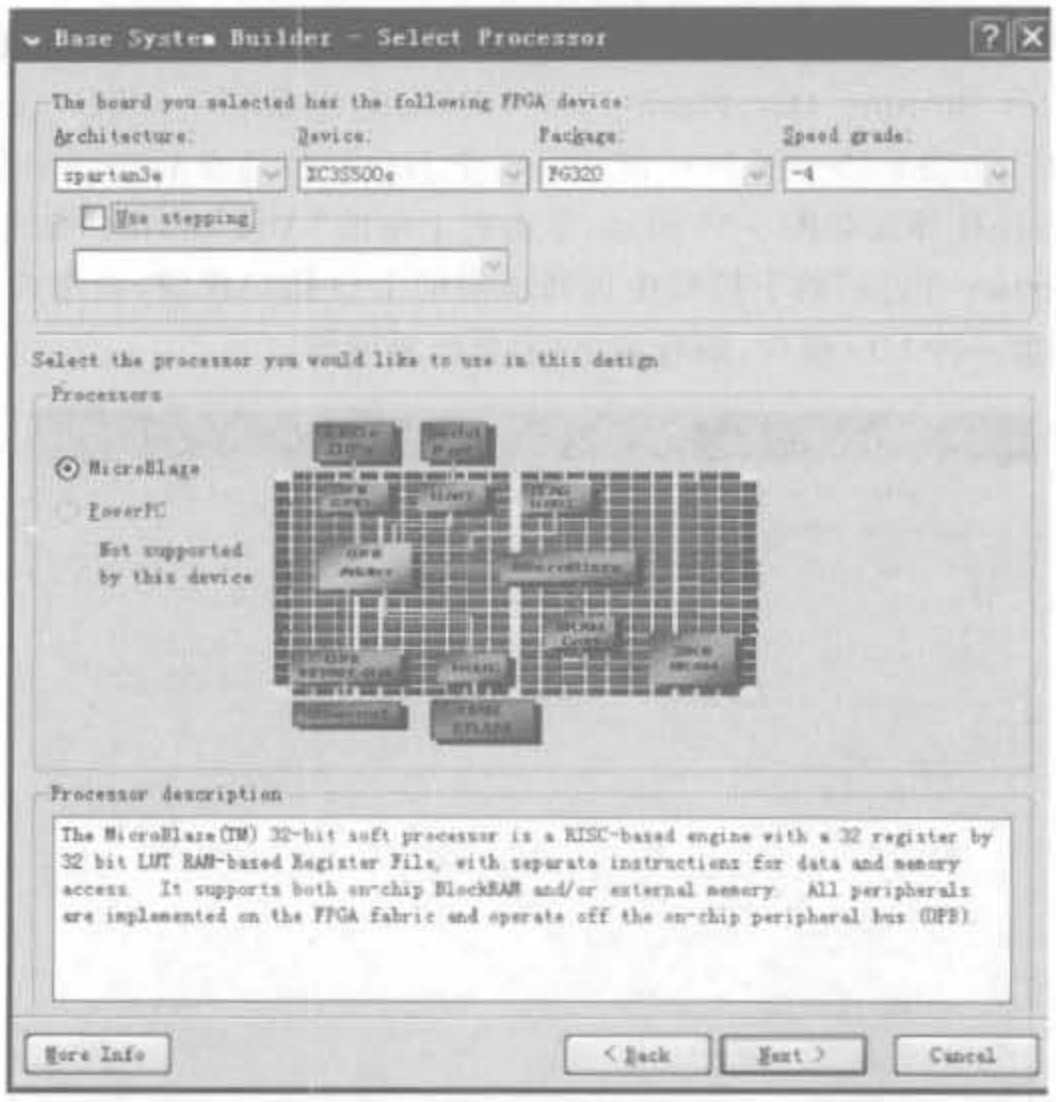


图 9-27 处理器参数配置界面

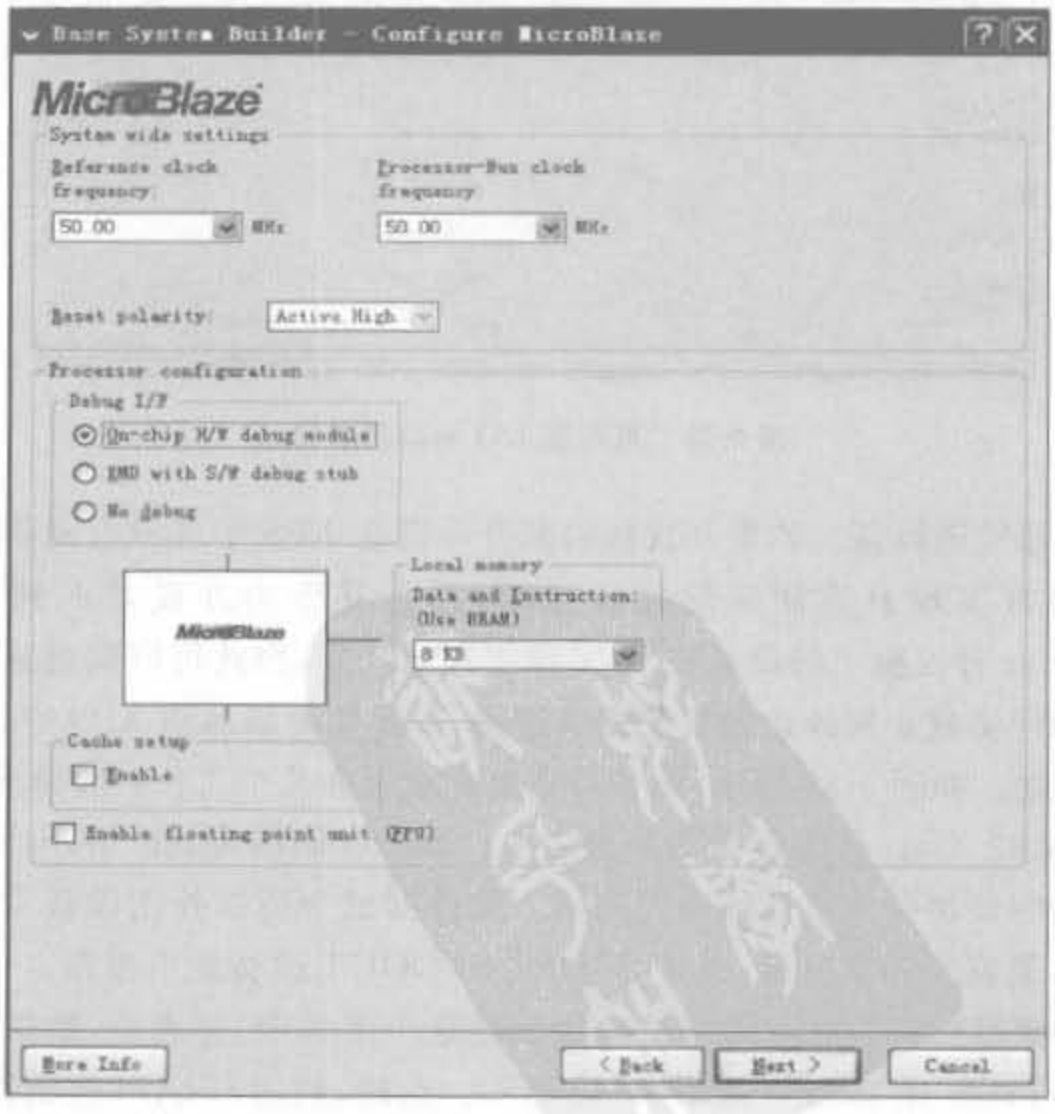


图 9-28 处理器选择界面

(6) 添加 I/O 接口。开发板配置文件选择了 RS232_DCE、RS232_DTE、LEDs_8bit、DIP_Switches_4bit、Buttons_4bit、Flash_16M×8、DDR_SDRAM_32M×16 以及 Ethernet_MAC 等 I/O 接口，如图 9-29～图 9-31 所示。各个 I/O 接口将在后文详细介绍。如果是用户自定义电路板，操作界面如图 9-30 所示，单击右上角的“Add Device”按钮，然后在弹出对话框的“I/O Interface Type”的下拉框中选择所需的 I/O 接口模块，再单击“OK”按钮。该过程每次只能添加一种 I/O 接口，如有多个，需要反复操作。

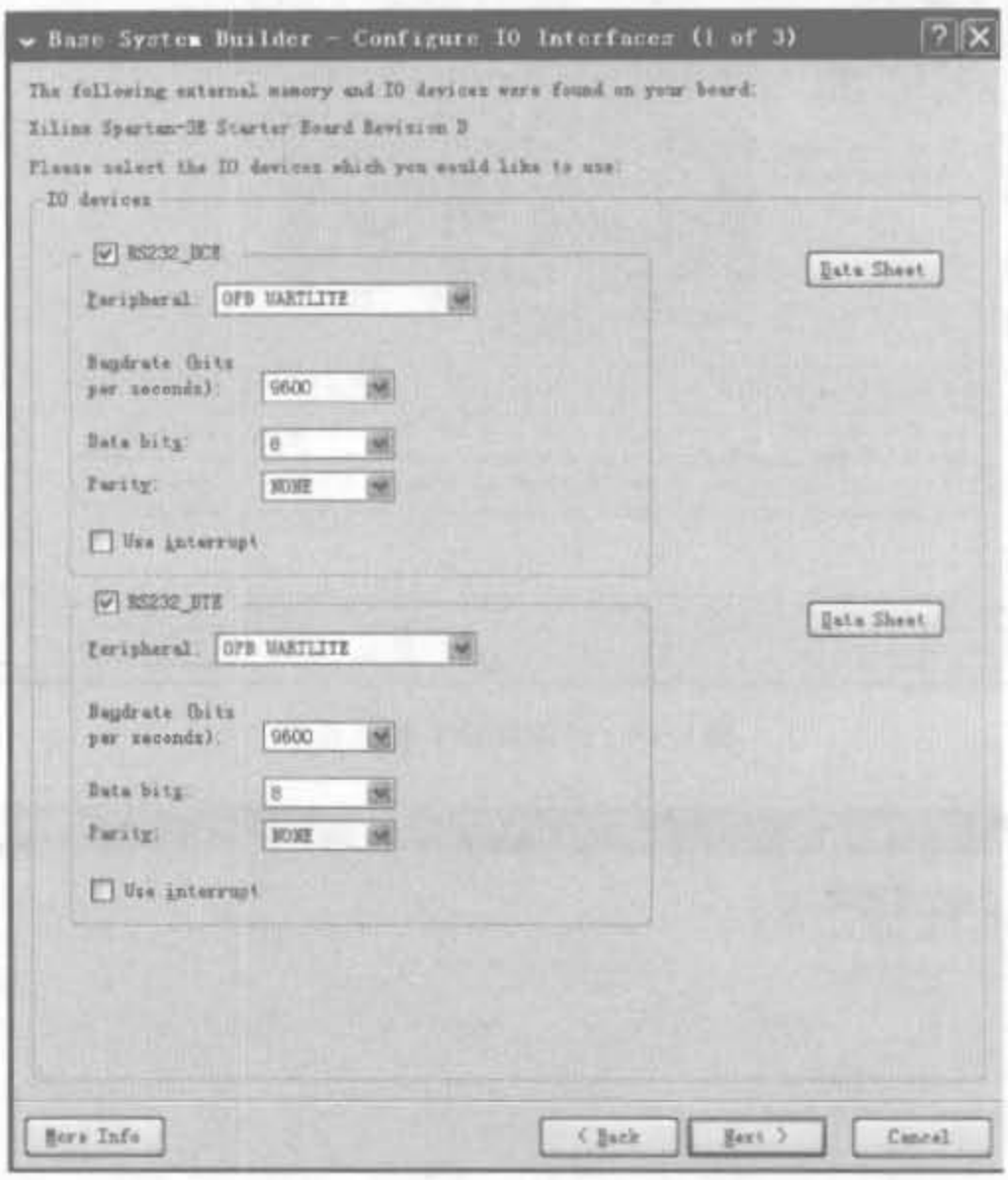


图 9-29　可配置 I/O 端口列表(1)

(7) 添加内部外围设备。如果开发板的部分外设在 XPS 中并没有提供，那么就需要用户通过 HDL 语言实现自定制底层接口逻辑，再将其作为外设导入到 XPS 中。由于 Spartan-3E Starter 开发板上的设备在 XPS 库中都能找到，因此可以跳过这一步骤。9.4.6 节将详细介绍如何实现定制外设的底层接口逻辑，以及如何添加到 XPS 中。

(8) 软件建立。如图 9-32 所示，该步骤需要完成下列几个工作：选择 STDIN/OUT 器件，这里注意 RS232_Uart 外围设备为所选之一；Boot 存储器为 plb_bram_if_cntlr_1；存储器和外围设备的测试都使用默认的应用测试。软件测试为所选外围设备发送或接收信息。微处理器得到外围设备的状况后，通过 STDIN/STDOUT 设备发出报告。

(9) 配置存储器和外围设备的测试应用。在这个工程中，将指令、数据和栈/堆的存储位置设置为 plb_bram_if_cntlr_1，如图 9-33 所示。这样，利用 BRAM 控制器“_if_cntlr_1”，程序代码在 FPGA“plb_bram”里的块 RAM 之外运行。

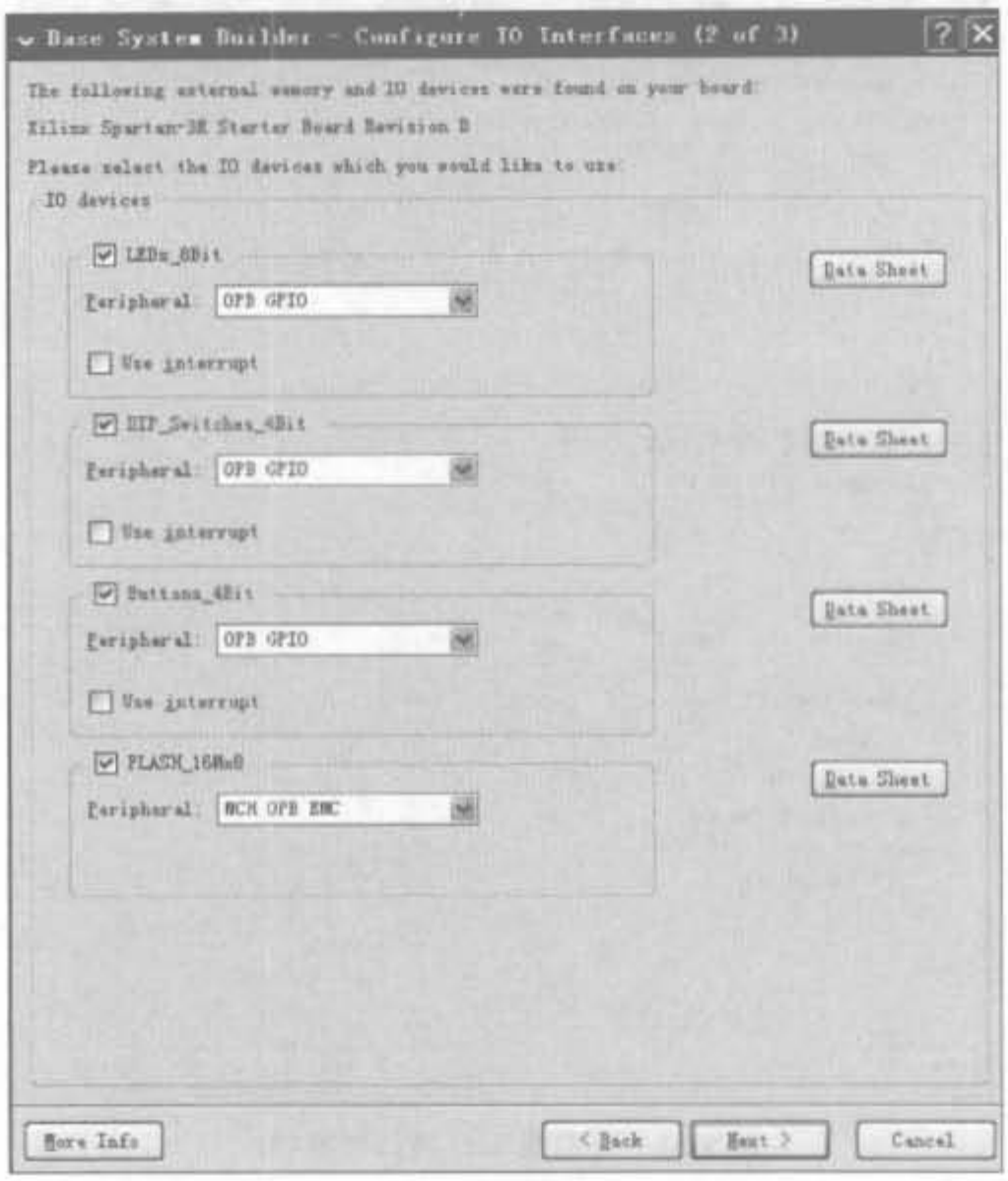


图 9-30 可配置 I/O 端口列表(2)

(10) 创建系统。在选择并配置系统各个部分后,用户就可以对 BSB 给出的系统进行检查。用户可以返回到之前的任何一步进行修改。检查后,单击"Generate"产生系统。BSB 给出的系统信息如图 9-34 所示,通过比较可以看出,它和开发板硬件是匹配的。

(11) 设计生成。在生成上述设计后,系统的目录结构同时得以创建。HDL 和其他文件根据用户的选择而生成,处理器、总线和外围设备以及其他逻辑实例间的连接也得到了处理。单击"Finish",XPS 即与所创建的系统相关联。至此,就完成了一个新系统工程的创建过程。

9.4.3 XPS 的用户界面

利用 BSB 建立了工程后,就可以利用 XPS 对此工程进行必要的修改。XPS 为创建硬件和软件流的 MHS(Microprocessor Hardware Specification)和 MSS(Microprocessor Software Specification)文件提供了一个图形用户界面(GUI),如图 9-35 所示,为文件编辑器功能和方案过程管理功能提供了源文件编辑器,用于管理整个工具流,包括硬件和软件执行流。XPS 的用户界面可以分为标题栏、菜单栏、工具栏和主窗口,其中主窗口又可分为 3 个部分:工程信息面板、系统组建面板和控制面板。虽然菜单栏和工具栏操作的功能等效于在主窗口的操作,但在实际中将近 80%的操作是在主窗口完成的,因此本节着重对其主窗口进行介绍。

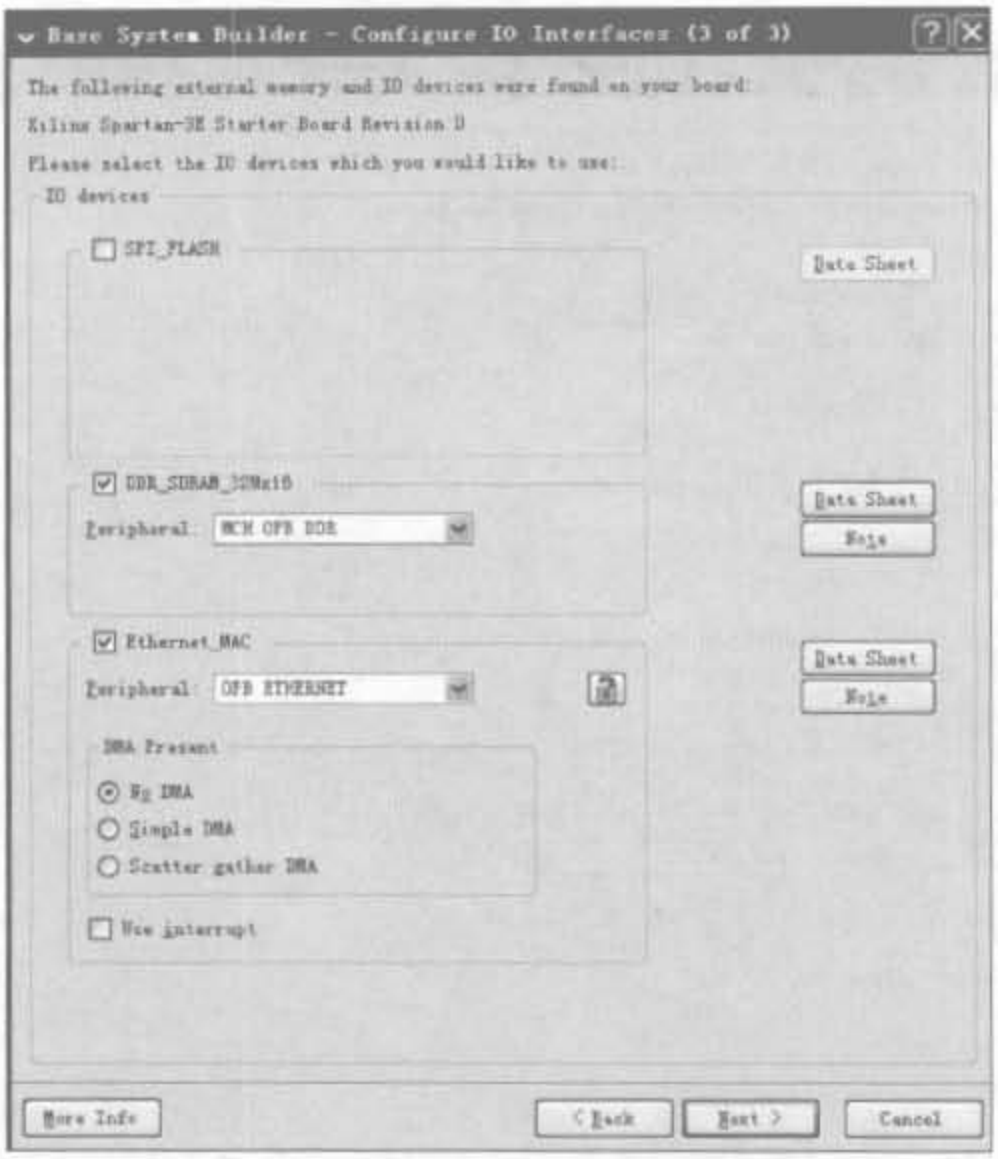


图 9-31　可配置 I/O 端口列表(3)

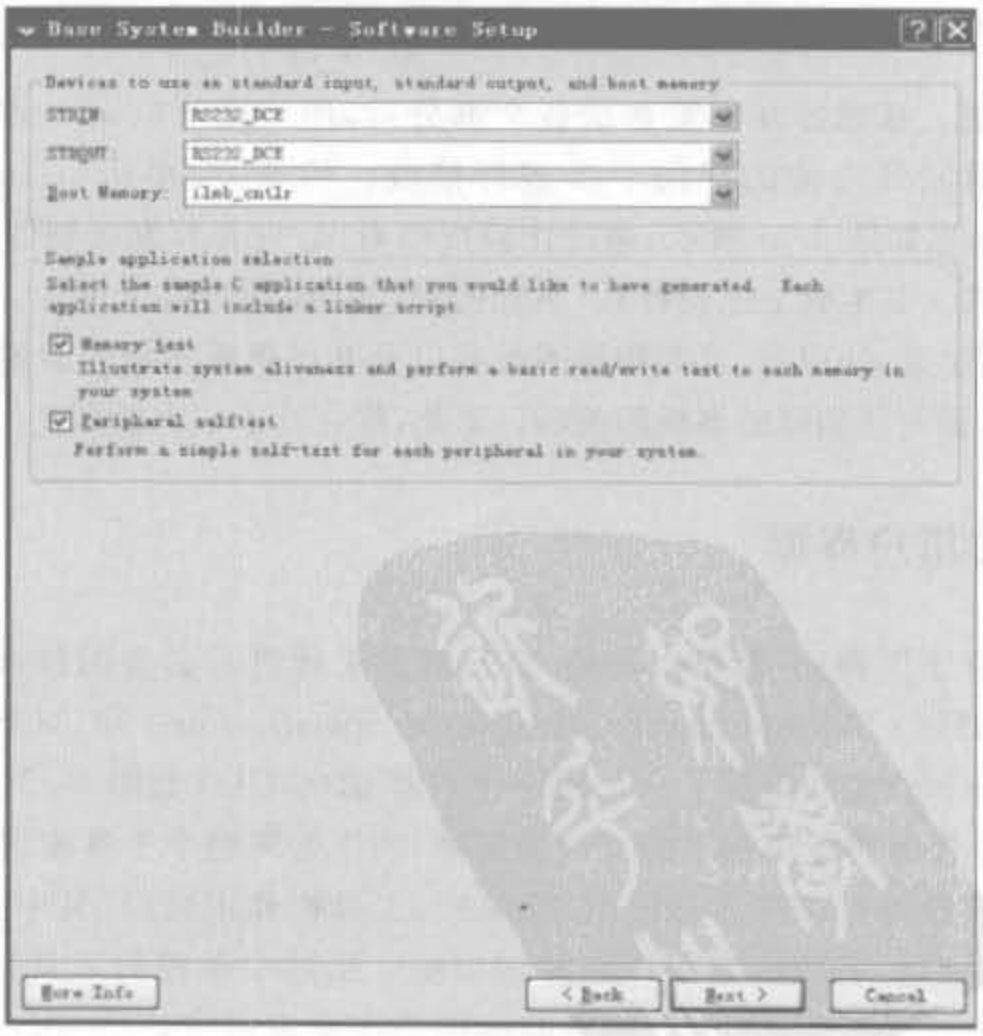


图 9-32　软件建立向导

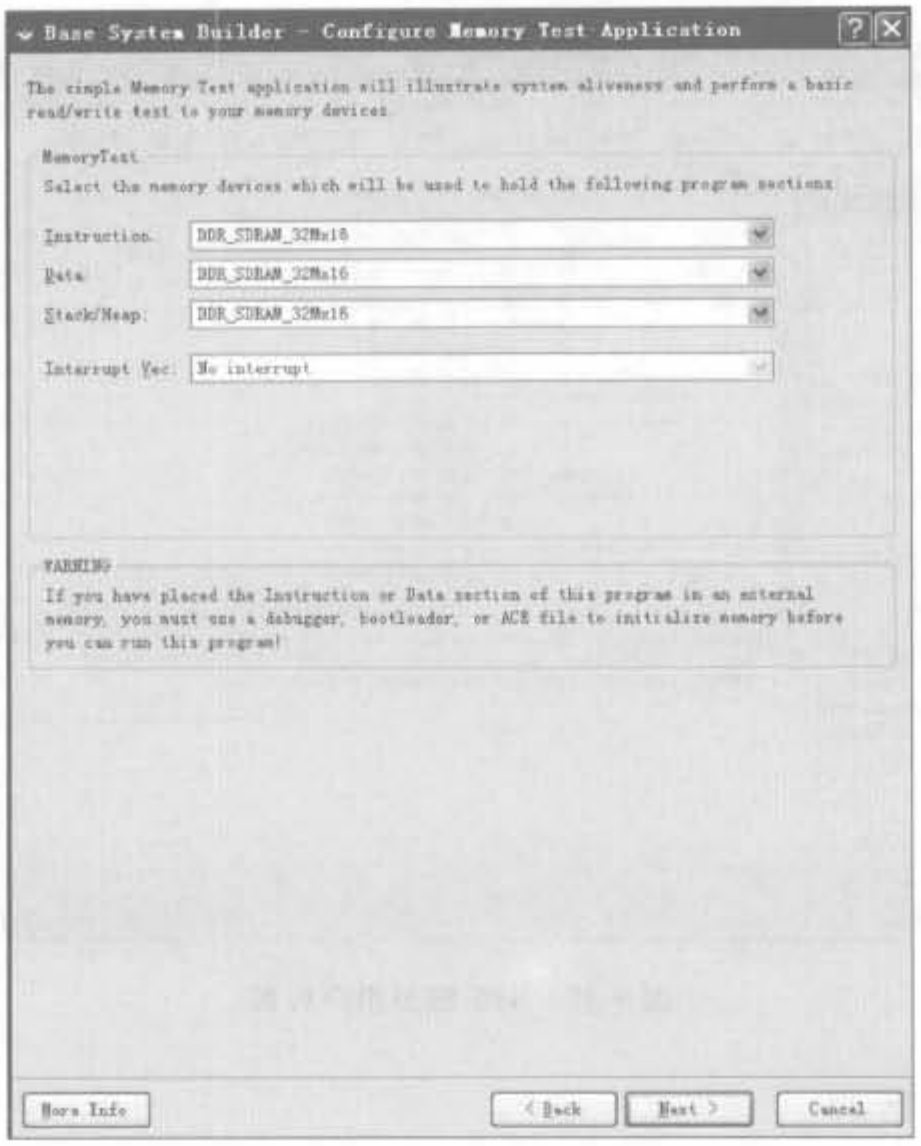


图 9-33 配置存储器和外围设备的测试应用

图 9-34 BSB 给出的系统信息

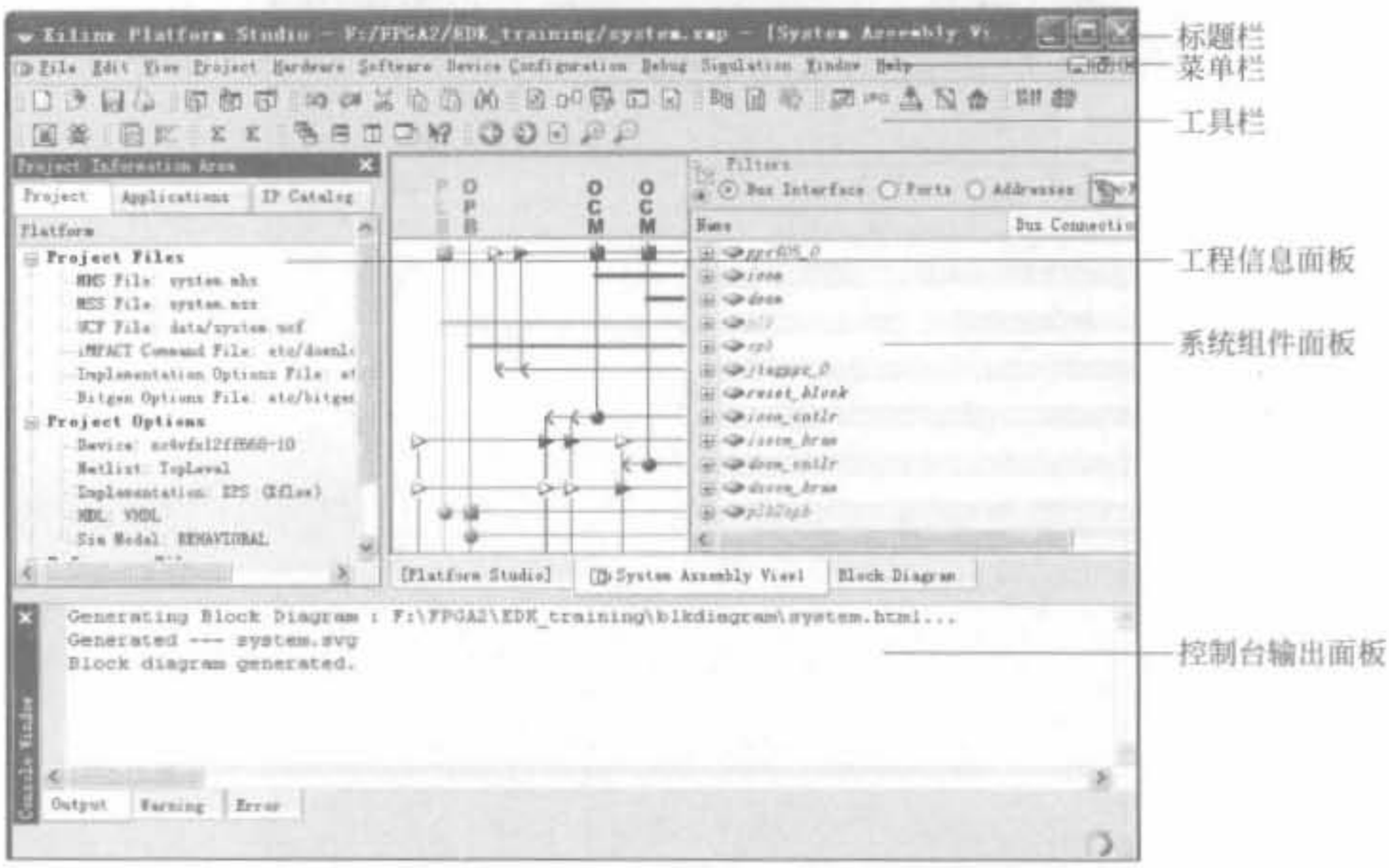


图 9-35　XPS 图形用户界面

1. 工程信息面板

工程信息面板主要对工程进行控制，包括工程(Project)、应用(Applications)和 IP 目录(IP Catalog)3 个页面。

1）工程页面

工程页面列出了与工程有关的文件，分为 3 个部分：工程文件、工程选项以及参考文件，如图 9-36 所示。其中，Project Files 的信息是从 BSB 向导中获取的，双击某文件即可修改，也可以进行编辑。.opt 文件和.ut 文件分别是加速编译的配置文件和生成比特流的配置文件，对于初学者，不建议修改。Project Options 的信息从 XMD 文件中获取，包括 FPGA 芯片的型号、网表信息、实现工具、HDL 语言种类以及仿真模型等信息，每一项都可以打开属性窗口进行设置。参考文件包括日志 log 文件和报告 srp 文件，前者记录了用户的操作，后者则记录每个执行过程所产生的报告，帮助用户了解设计结果。这两个文件是 XPS 自动生成的，不需要修改。

2）应用页面

应用页面的信息都是和应用软件相关的。在嵌入式系统中，软件的作用是十分重要的，硬件系统提供运行的平台，需要实现的全部功能都是通过软件系统来完成的。在 XPS 中，应用软件分为两种：一种是系统自动生成的 boot，由 XPS 自动生成，用户无法修改；另一种是用户编写的应用软件。应用的所有组件如图 9-37 所示，分为处理器属性、编译器属性、源代码以及库文件。右击用户定义的应用，可完成以下操作：

- Mark to Initialize BRAMs：将软件应用打包到初始化块 RAM 中，添加.bit 配置文件，形成最终的比特文件，可直接配置到 FPGA 芯片中；
- Build Project：将软件应用编译成可下载执行的.elf 文件；

图 9-36　工程信息区域：工程标签

图 9-37　工程信息区域：应用标签

- Make Project Inactive：将当前工程从初始化块 RAM 中剔除，使其失效；
- Generate Linker Script：自动生成连接脚本；
- Set Compiler Options：设置编译属性，单击后弹出编译属性子窗口，可设置连接脚本、堆、栈的大小以及程序的起始位置；代码优化级别和软件调试模块；第三方库以及编译器的绝对路径以及高级用户的定制编译指令。

3）IP 目录页面

IP 目录页面列出所有 EDK IP 核和用户生成的 IP 核，如图 9-38 所示。由于 Xilinx 提供的 IP 核种类繁多，因此在 XPS 中按照功能进行分类，涉及到模拟接口、系统总线、存储器控制器、通信接口以及调试接口等十多个类别，指明了每个 IP 的版本、类别、名称以及使用的处理器型号。选中并拖曳至系统组件面板，即可将其添加到系统中；同时，在 IP Core 上单击鼠标右键，用户便可阅读其数据手册中的相应功能和使用方法。需要注意的是，有加锁标志的 IP Core 不是免费的，用户需要购买相应的 License 才能使用。

图 9-38　工程信息面板的 IP 分类示意图

例 9-2　在本例中，给出一些操作来指导读者认识工程信息面板中的内容。

（1）单击“Project”页面标签，用户可以查看工程文件以及对应选项。

（2）单击“Applications”标签，展开“Project：TestApp_Peripheral”，可以看到应用对应的处理器信息、可执行文件存放路径、编译选项、源代码以及头文件等信息。

（3）在“Processor：ppc405_0”中的“xparameters. h”文件包括了系统的地址映射，是 BSP 的一个部分。

（4）在“Compiler Options”和“Sources”中，连接脚本和测试应用源已由 BSB 向导自动

生成。单击“IP Catalog”标签，展开“Communication Low-Speed IP”目录，右键单击“OPB_UART(Lite)”外围设备查看选项。这里可以选择直接视图或分层视图。单击图 9-38 中所标注圆圈内的图标，可在这两种视图间切换。

2. 系统组件(assembly)面板

系统组件面板是 XPS 软件使用频率最高的区域，几乎所有的操作都集中在这里。该窗口是可视化配置系统硬件结构的主要手段，其界面如图 9-39 所示，分为连接区域和显示区域。此外，代码编辑也集中在显示区域，用户以文本形式打开的所有可编辑文件都显示在该区域，并具备一般文件编辑器的功能，可完成编辑功能。

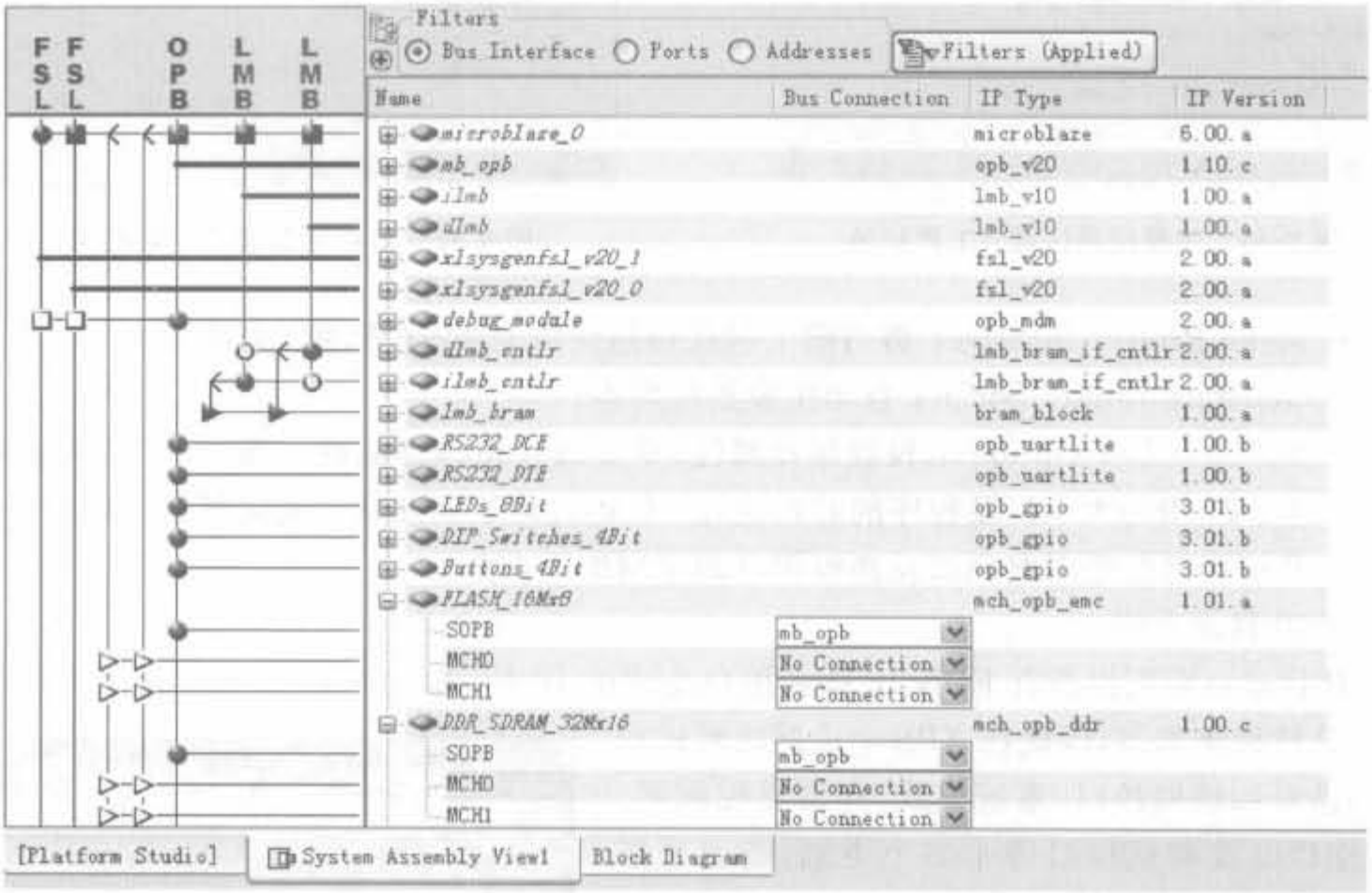


图 9-39　系统组件面板

1) XPS 连接区域

XPS 的连接面板提供了总线接口、端口和地址选项，用户可以方便地编辑硬件平台。如果选择总线接口选项，将出现连接面板，它给出了硬件平台互连图，其中的不同颜色和形状线条具有不同的物理意义：

- 竖直线表示总线，水平线表示到 IP 核的总线接口；
- 如果总线和设备相连，则在总线和 IP 核总线接口的交叉处会出现一个连接点；
- 线和连接器以不同的颜色标出；
- 不同形状的连接符号表示 IP 核总线接口的不同身份；
- 中空的连接器表示用户可以进行连接，而实心的连接器表示已经有连接。用户可以单击连接器符号来创建或删除连接。

2) 显示区域

如前所述，显示区域可分为系统组件页面和代码编辑页面，本节只介绍系统组件页面。XPS 系统组件页面分为总线接口(Bus Interface)、端口(Ports)以及地址(Address)3 个子窗

口，通过单击“Filters”栏下面的选项进行切换。在系统组件页面，所有的操作都会引起硬件配置的变化，并反映到 MHS 文件中。

(1) Bus Interface 子窗口

该窗口给出了各个硬件单元和总线的连接关系，连接到某总线的硬件单元将和该总线的颜色一致，如图 9-39 所示。在硬件单元上单击鼠标右键，可配置其参数、阅读相应的数据手册以及查看底层代码。

(2) Ports 子窗口

该窗口用于配置端口参数，包括顶层模块和各个子模块的端口，并可对其重命名。单击相应信号行 Net 列的下拉框，可选择连接的网表名，如图 9-40 所示。

图 9-40 Ports 子窗口示意图

(3) Address 子窗口

该窗口描述了各硬件单元的绝对地址和大小，可单击任意行的 Base Address 列和 Size 列设置不同的数值。设置完成后，High Address 列的数值会自动作出调整，如图 9-41 所示。此外，用户还可设置指令缓存(ICache)和数据缓存(DCache)的位置。

系统组件面板给出了两种视图选项：分层视图和直接视图，用户可以更容易地进行信息分类和设计。系统组件面板默认分层视图，此时设计信息基于硬件平台的 IP 核实例，并以可扩展树的结构进行组织。而在直接方式中，有关信息以字母的顺序显示。单击目录结构图标时，端口以分层方式或直接方式显示。

例 9-3 在本例中，给出一些操作来指导用户认识系统组件面板中的内容。

切换到在系统组件面板中后，

(1) 单击“Ports”按钮，展开“External Ports”目录查看 FPGA 器件外的信号。注意与“RS232_Uart”有关的信号。

(2) 用户可以查看 RS232_Uart 外围设备信号以及与之相连的 FPGA 内部模块端口信号线名称。UART RX 和 TX 信号线的名字与外部端口相对应，它们之间的对应关系在 UCF 文件中指定。

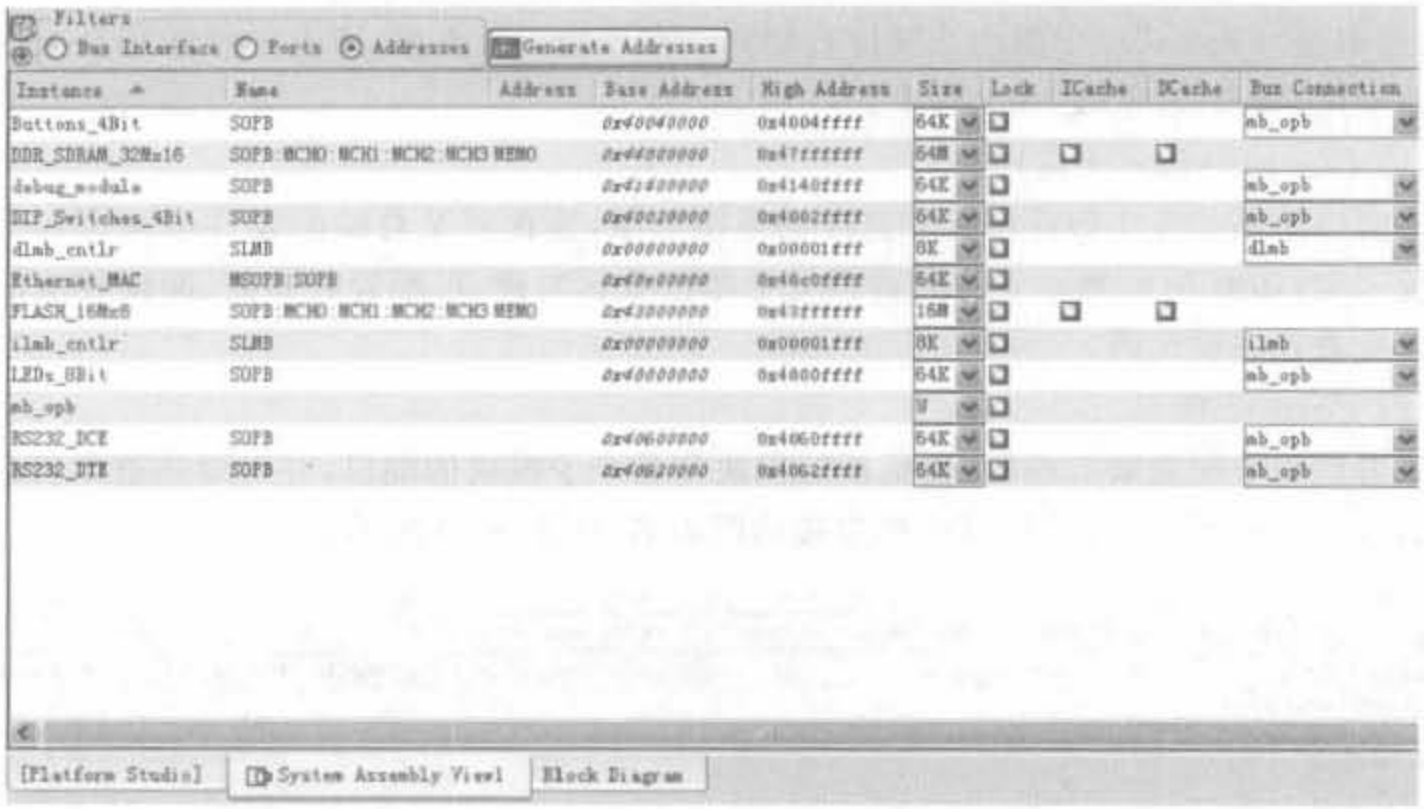


图 9-41　Address 子窗口示意图

(3) 双击 RS232_Uart 外围设备图标打开“RS232_Uart：opb_uartlite_v1_00_b”参数对话框。用户可以使用此参数对话框为 IP 调整不同的设置。

(4) 单击目录图标，在分层视图和直接视图间切换。

3. 控制台输出面板

控制台输出面板给出运行时给出的日志反馈信息，分为 3 个标签：

(1) Output：输出系统所有信息。

(2) Warnings：仅显示系统的警告信息。

(3) Errors：仅显示系统的错误信息。

通过这些信息可快速定位系统设计的问题，并找出相关原因。

9.4.4　XPS 的目录结构与硬件平台

1. 目录结构

启动 BSB 后，XPS 会自动生成工程目录结构，并创建一个完整的工程。所创建的目录结构如图 9-42 所示。

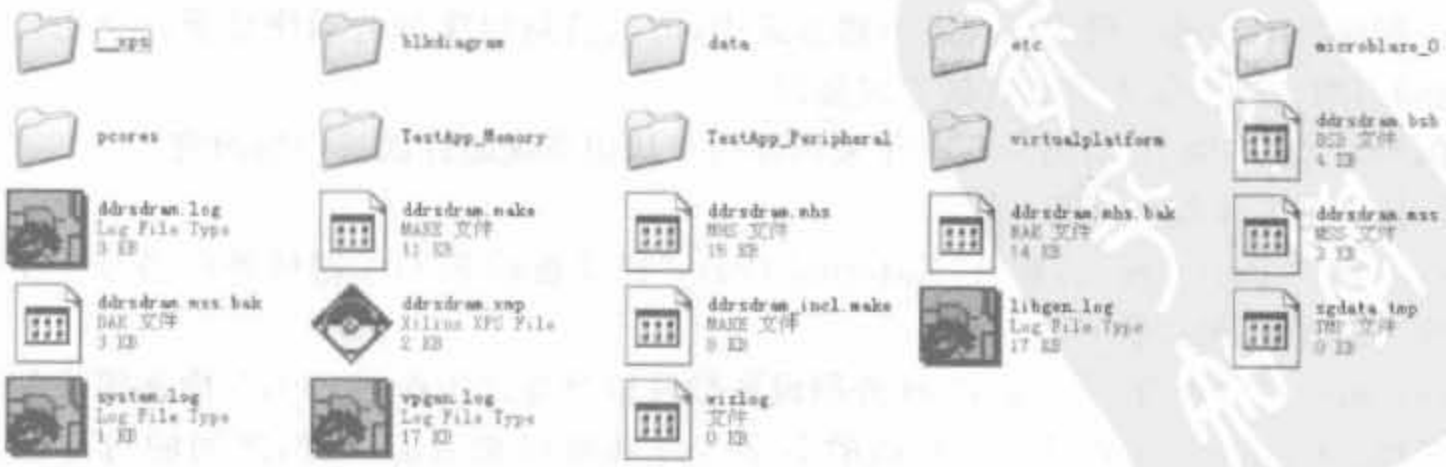


图 9-42　运行 BSB 向导创建的文件

下面对 BSB 自动生成的 4 个主要文件夹进行介绍。

(1) _ _xps 文件夹：包括 XPS 和内部工程管理的其他工具产生的中间文件。

(2) data 文件夹：包括用户约束文件(UCF)。

(3) etc 文件夹：此目录包括的文件给出了那些用来运行不同工具的选项。

(4) pcores 文件夹：包括了用户定制的硬件外围设备。

另外，在主工程目录下还有其他一些文件。例如，

(1) system. xmp：这是 EDK 的顶层工程设计文件。XPS 读取此文件，并在用户界面上给出此文件内容。

(2) system. mhs：系统微处理器硬件规范(或 MHS 文件)，给出系统元素、对应的参数以及连接。MHS 文件是项目的硬件基础。此外，UCF 文件也是和硬件结构对应的。

(3) system. mss：系统微处理器软件规范(或 MSS 文件)，给出设计的软件部分，描述了系统元素以及外围设备的不同软件参数。MSS 文件是项目的软件基础。

2. 硬件平台

嵌入式硬件平台包括一到多个处理器，以及多个外围设备和存储块。这些 IP 块利用之间的互连网络进行通信。每个处理器或外围设备核都可以由用户设计。通过设计参数，可以控制不同的可选参数。

1) Xilinx 平台工作室的硬件平台开发

XPS 提供了一个交互式的开发环境，允许用户对硬件平台的各个方面进行设置。其中，XPS 在高层对硬件平台描述进行维护，此高层形式即为微处理器硬件规范(MHS)文件。MHS 作为一个可以编辑的文本文件，是表示用户嵌入式系统硬件部分的主要源文件。XPS 将 MHS 源文件综合到硬件描述语言(HDL)网表中，后者用于 FPGA 的布局布线。

MHS 文件的内容和格式已在 9.3.3 节说明。读者可从 Project 标签中找到 MHS 文件，可在该文件中查看外围设备和端口的配置。

例 9-4　在本例中，指导用户查看 XPS 工程的 MHS 文件。

(1) 选择工程信息区的“Project”标签，可双击“MHS File: system. mhs”打开此文件，如图 9-43 所示。

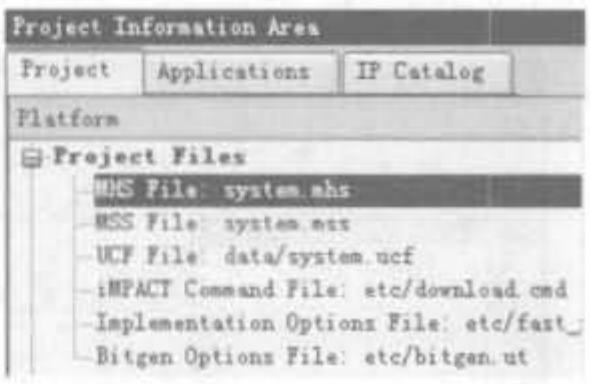


图 9-43　MHS 文件

(2) 在文件“system. mhs”中的“opb_uartlite”处，可以看到 MHS 文件里配置的外围设备、端口和它们的参数。

(3) 还可以查看其他 IP 核。查看后，关闭“system. mhs”文件。

2) 系统组件面板的硬件平台

XPS 的系统组件面板以展开式的树和表形式显示了所有的硬件平台 IP 实例，因此用户可以方便地查看自己的嵌入式设计。在此面板中，对 IP 元素、端口、属性以及参数进行的配置会直接写入到 MHS 文件中。即 XPS 会自动地将系统修改写入到 MHS 文件中的硬件数据库。因此，用户要编辑 MHS 文件时，Xilinx 推荐使用系统组件面板这一功能。

为产生硬件平台，用户必须告知 XPS 产生网表，并产生比特流。

(1) 在产生网表时，XPS 调用平台创建工具 Platgen 来进行一系列工作，包括读取 MHS

文件，产生 MHS 文件对应的 HDL，利用 Xilinx 综合技术来综合设计，最后产生网表文件。

(2) 在产生比特流时，Platgen 先检查是否存在更新后的网表。ISE 实现工具读取建立的网表文件，并连同用户约束文件(UCF)来产生包含硬件设计的 BIT 文件。

9.4.5 在 XPS 加入 IP Core

XPS 提供了数目众多的 IP Core，包括外设(块 RAM 接口控制器、中央 DMA 控制器、模/数转换器、数/模转换器、外设控制器、以太网 MAC 控制器、GPIO 控制器、I^2C 总线接口、中断控制器、多通道外部存储器控制器、串行外设接口、SystemACE 接口控制器、定时器、串口通信接口以及 USB 接口等)和基础设施(数据端的片上存储器总线、快速单工链路总线、单精度浮点计算内核、LMB 块 RAM 接口控制器、微处理器调试模块、处理器系统复位模块以及内部总线各种桥接模块)两大类。每个 IP Core 可直接实现用户所需的所有功能，能与 MicroBlaze 软处理器核和 PowerPC 处理器协同工作，从而减少了为外部设备编写驱动程序的工作。下面在例 9-1 的基础上，通过添加通用输入、输出 I/O(GPIO)模式来介绍如何在工程中添加 XPS 提供的 IP Core。

例 9-5 在例 9-1 的基础上添加一个 8 位的 GPIO 模块。

(1) 添加 GPIO 模块。在“Project Information Area”区域中选择“IP Catalog”选项，并在“OPB General Purpose I/O”上单击鼠标右键，然后在弹出的对话框中选择“Add IP”命令，可将该 IP Core 加入工程；或者选中“OPB General Purpose I/O”，直接将其拖到系统组件区，也能添加该 IP Core。

(2) 完成 GPIO 模块的总线接口配置。在工程区单击“opb_gpio_0”前面的“+”号，展开树形结构，然后单击“SOPB”左侧对应的 OPB 节点，将其从空心节点变成实心节点，完成和 OPB 总线连接，如图 9-44 所示。

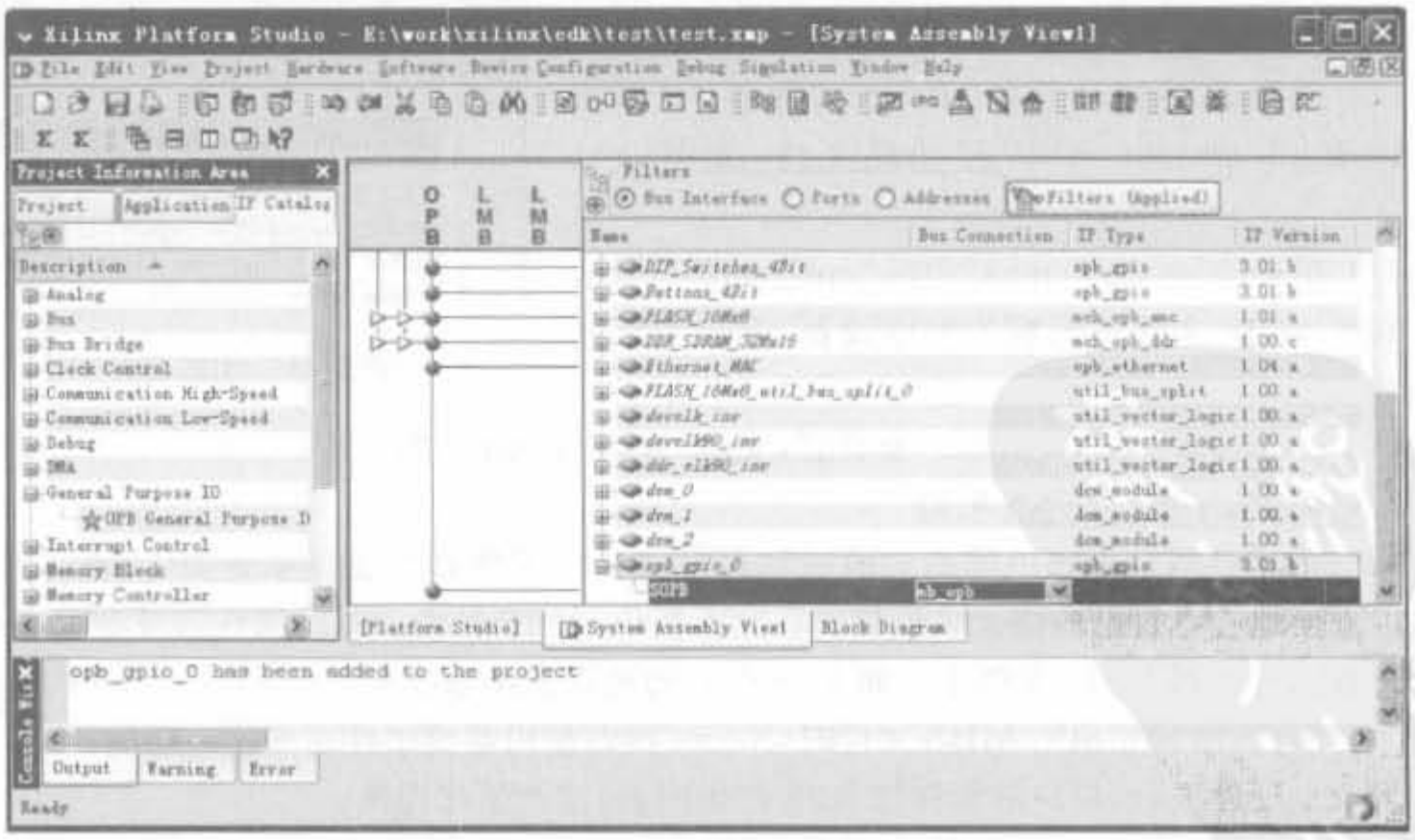


图 9-44 GPIO 模块和 OPB 总线连接示意图

(3) 完成 GPIO 模块的参数配置。在 opb_gpio_0 上单击鼠标右键，并选择“Configure IP”，或选中“opb_gpio_0”，在除“Name”列以外的地方双击，即可进入 IP Core 的配置页面。在其中，将 GPIO 的位宽设为 8，其余保持默认值，单击“OK”完成参数配置，如图 9-45 所示。如想查看该核的说明文档，可以单击“Datasheet”的图标。

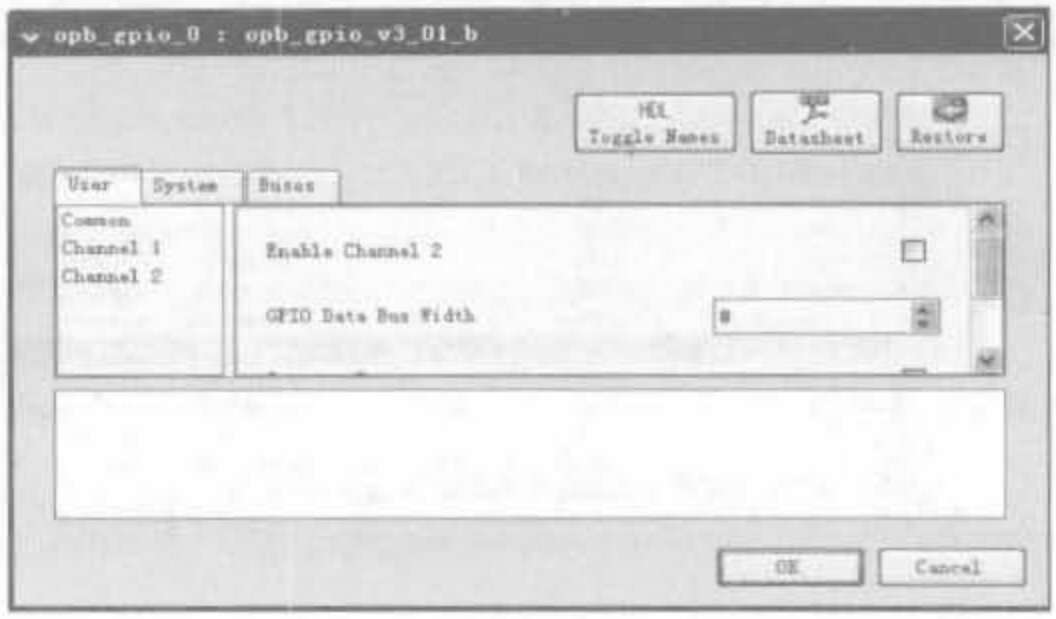


图 9-45　GPIO 模块的参数配置示意图

(4) 完成 GPIO 模块的端口配置。在工程区选择“Port”选项，单击“opb_gpio_0”的“+”号，展开树形结构。在“Net”列的下拉菜单内选择相应的连接。如果该端口要和外部设备通信，需要选择“Make external”，将其作为外部接口；否则选择内部连接即可。例如 GPIO_in，选择了 OPB_gpio_0_GPIO_in，就将其作为内部连接。同样配置 IP2INTC_Irpt、GPIO_I/O、GPIO_d_out、GPIO_t_out，其余端口采用默认值，如图 9-46 所示。

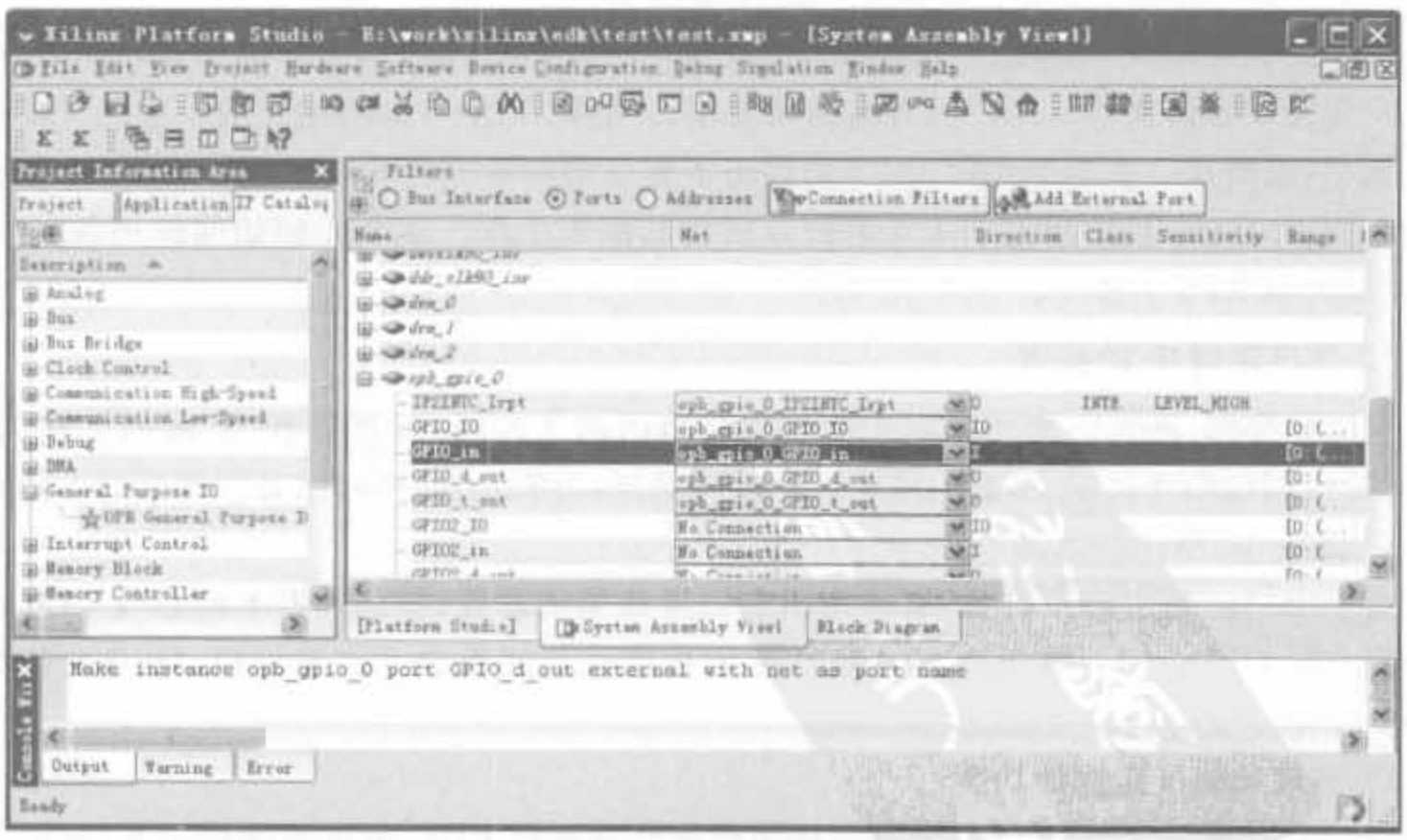


图 9-46　GPIO 模块的端口配置示意图

(5) 完成 GPIO 模块的地址配置。在工程区选择“Addresses”选项，然后单击“Generate Addresses”按钮，由 XPS 自动完成地址配置分配，如图 9-47 所示。

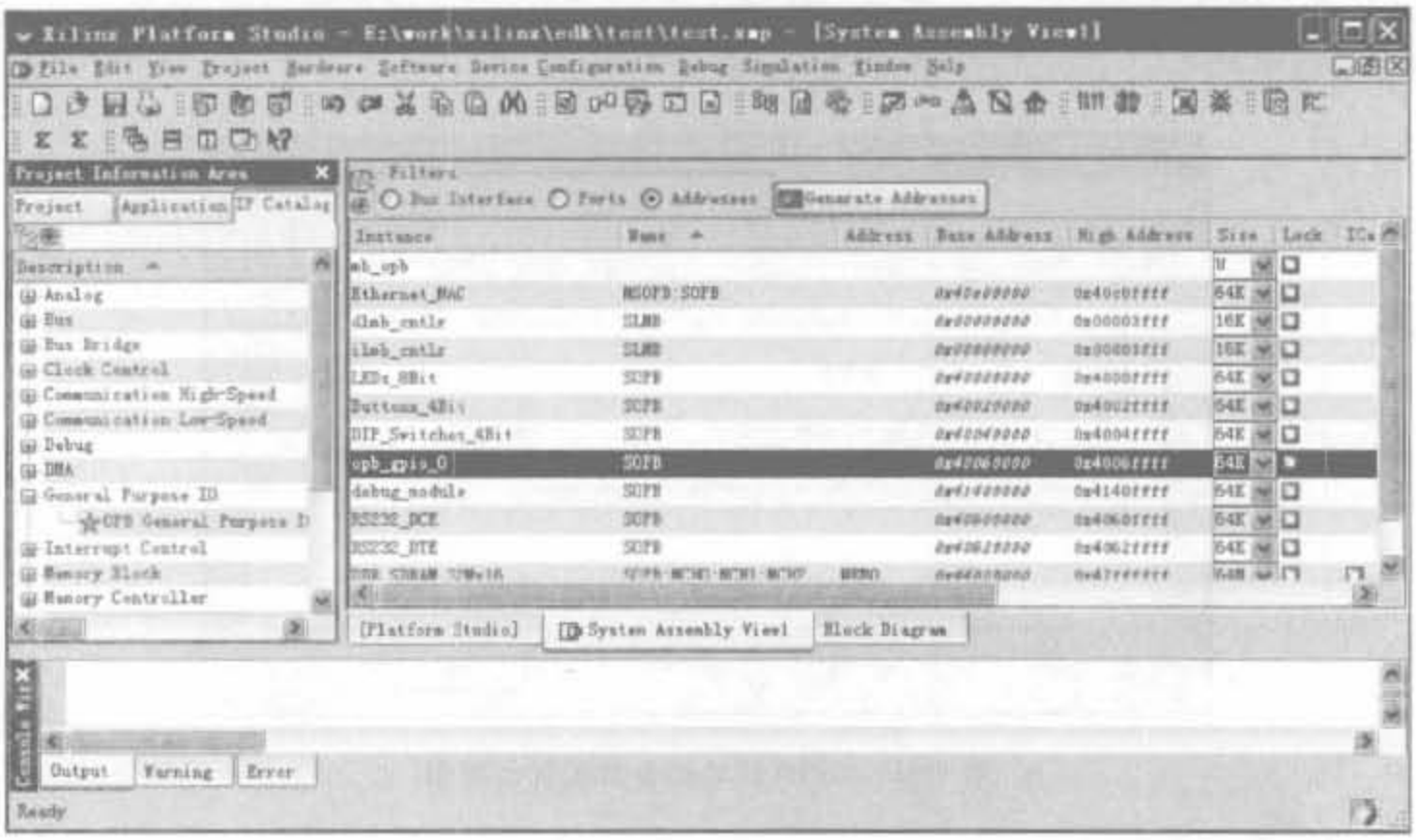


图 9-47　GPIO 模块的地址配置示意图

至此，已成功向工程中添加了 GPIO 模块的 IP Core，其他 IP Core 的接法也是类似的。

9.4.6　在 XPS 中定制用户设备的 IP

上文提到的嵌入式系统的开发基本是由 XPS 自动完成的，本节介绍如何在 XPS 中添加用户定义设备的操作。用户只有掌握该操作，才能推广 XPS 的应用领域。虽然外部设备种类繁多，总线接口也不尽相同，XPS 仍给出了相应的向导来简化此过程。首先，定制自定义设备的 IP Core，然后通过 9.4.5 节的方法将其加入工程。本节介绍如何使用 XPS 定制 IP，并添加到 XPS 工程中。

1. 用户定制 IP 的要求

前面已提到，在系统组件面板中的总线接口给出了总线、处理器和 IP 间的互连关系。注意，用户创建的任何 IP 都必须适应已生成的系统，为满足这一条件，必须做到以下两点：

1) 确定 IP 所需要的接口

对于用户定制的外围设备，必须指出它们所连接的总线，如处理器本地总线 PLB 或片上外围总线 OPB。PLB 在处理器和高性能的外围设备间提供了高速接口，OPB 允许处理器接到低速、低性能的系统资源。

2) 实现和验证定制的功能

(1) IP 必须导入到 EDK。用户的外围设备要复制到 EDK 中正确的目录下，同时必须创建平台规范格式(Platform Specification Format，PSF)接口文件(MPD 和 PAO)，这样，其他 EDK 工具才可以识别出用户的外围设备。

(2) 必须将外围设备加入到在 XPS 创建的处理器系统中。

2. IP 接口介绍

IP 接口(IP interface,IPIF)是个已经被验证、最优化的并且参数可调的接口,同时它提供了一系列简单总线协议,即 IP 互连(IPIC)。用户利用 IPIF 模块,使其参数匹配自己的需求,就可以极大地减少设计和测试的工作量。图 9-48 给出了总线、IPIF、IPIC 和用户逻辑之间的关系。

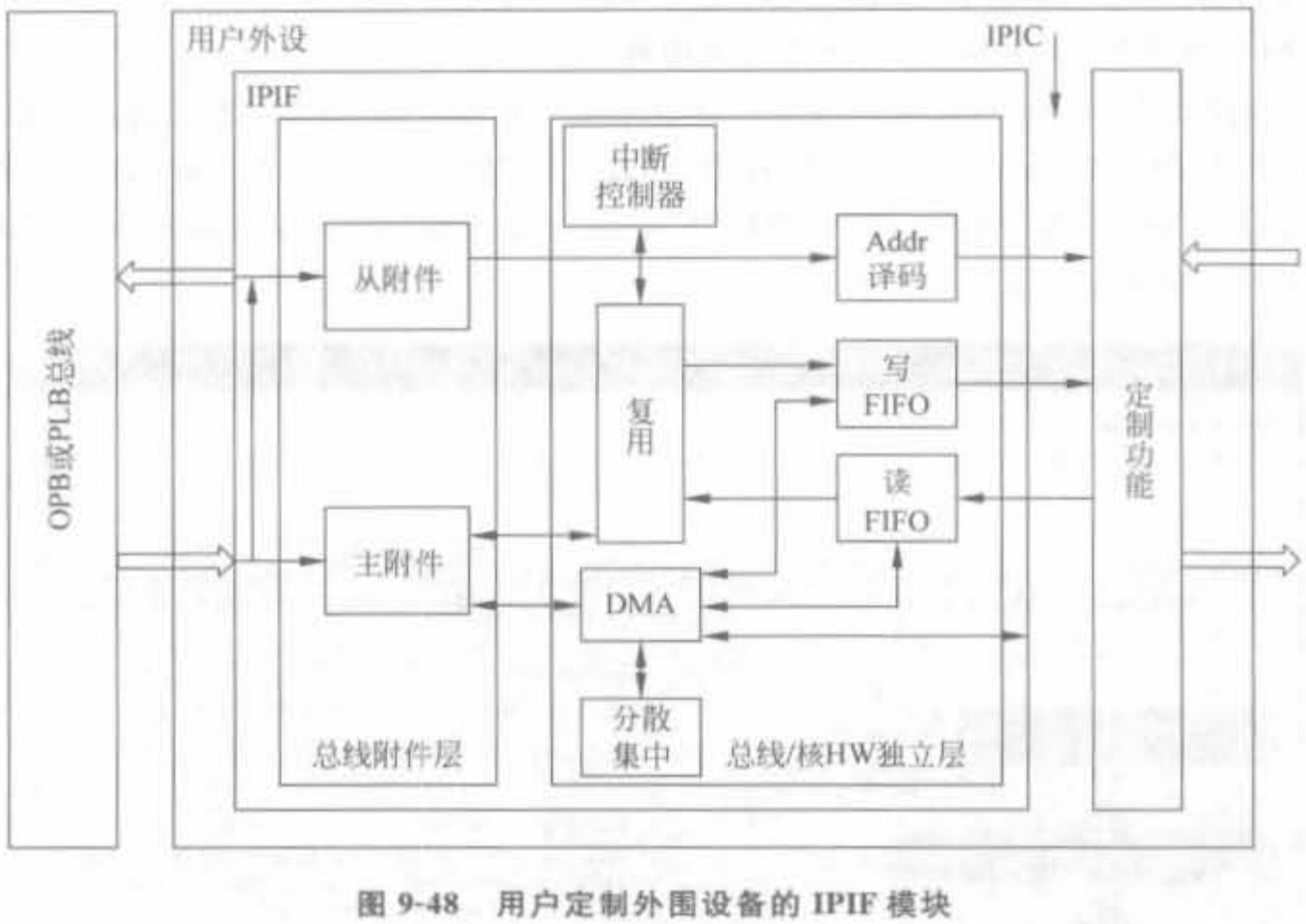


图 9-48 用户定制外围设备的 IPIF 模块

EDK 利用 IPIF 库在不同的处理器外围设备间执行相同的功能。在总线接口和 IPIF 服务面板上,CIP 向导要求用户定义目标总线以及 IP 所需要的服务。CIP 向导创建两个模板文件以协助 IP 连接。假设顶层文件名为 test_ip,用户定制逻辑连接到第二个文件 user_logic. vhd。用户可打开工程目录下的 pcores 文件夹,查看向导所创建的有关目录结构和文件,就会发现向导创建了两个 VHDL 模板文件:test_ip. vhd 和 user_logic. vhd。后者通过在 test_ip. vhd 里配置的 IPIF 核建立连接到 OPB 总线上。User_logic. vhd 文件相当于用户定制功能块,而 test_ip. vhd 文件相当于 IPIF 块。对于 IPIF,在例 9-6 中会有更详细的说明。

3. 利用 XPS 完成 IP 定制

XPS 提供的创建和导入外围设备向导(Create and Import Peripheral,CIP)可以协助用户完成 IP 的创建过程。此向导可以为用户建立多种模板用于不同的逻辑。除了创建 HDL 模板,CIP 向导还创建外围核(pcore)验证工程用于总线功能模型(Bus Functional Model,BFM)验证。模板和 BFM 工程的创建有利于 IP 的开发,并保证所创建 IP 和系统的兼容性。

使用 CIP 来简化用户定制外围设备的创建过程如下:

(1) 确定设备类别。

自定义设备必须和 CoreConnect 兼容。CIP 利用预先定义的 IPIF 库，可以创建和 CoreConnect 兼容的 4 种类型的外围设备：OPB 从属外围设备、OPB 主从结合的外围设备、PLB 从属外围设备和 PLB 主从结合的外围设备。

(2) 选择菜单“Hardware”中的“Create or Import Peripheral”命令，建立新的外围设备或者是导入已存在的外围设备。

下面通过一个实例，给出创建用户自定义外围设备的详细步骤。

例 9-6 在 XPS 中定制 8bit LED 的驱动外设。

(1) 选择菜单“Hardware”的“Create or Import Peripheral”命令，进入创建与输入外设(CIP)向导，如图 9-49 所示。从中可以看出，创建定制 IP Core 有 3 个步骤：创建模板、实现验证以及导入 XPS。在“Select flow”选择“Create templates for a new peripheral”选项，单击“Next”按钮进入下一页。

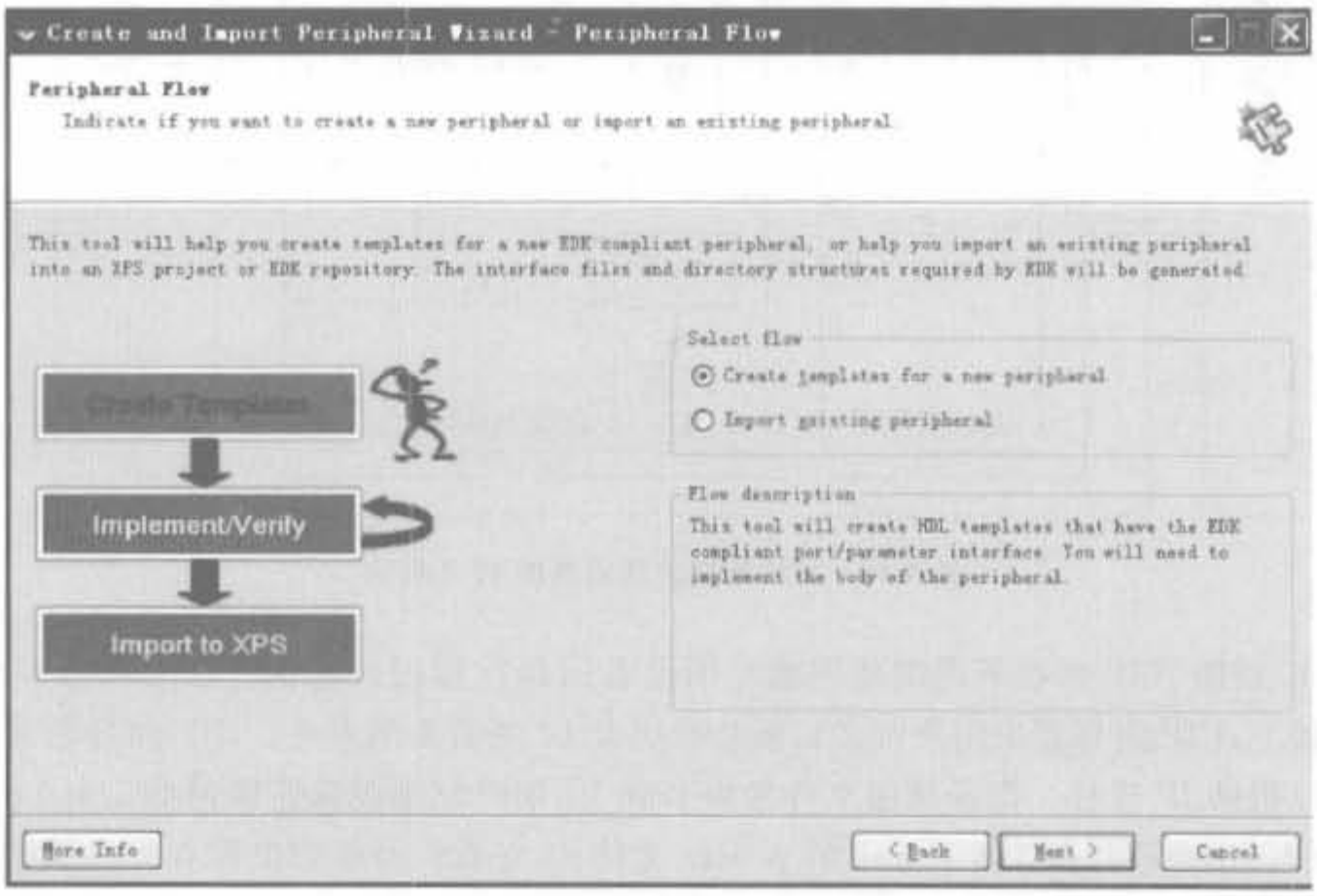


图 9-49 XPS CIP 向导界面

(2) 选择定制外设的存放位置。选择“To an EDK user repository”，可将其存放到 EDK 安装路径外的任何目录；选择“To an XPS project”，将其添加到 XPS 工程中。默认选择为存放到当前工程。本例采用默认值，如图 9-50 所示，单击“Next”按钮进入下一页。

(3) 定义定制 IP 的名字和版本。本例输入 my_led8 以及版本号为 1.00.a，如图 9-51 所示。这里 IP 名称需要小写，单击“Next”按钮进入下一页。需要注意的是，良好的版本管理对于 IP Core 设计是至关重要的。

(4) 选择定制 IP 所连接的总线。可选项有 OPB、PLB 以及 FSL，需要根据设备的速度以及设计人员的熟悉程度决定，因为 IP 的功能就是实现相应总线时序到用户设备时序的转化。本例选择 OPB 总线，如图 9-52 所示。单击“Next”按钮进入下一页。

(5) 选择 IPIF 服务。在 IPIF 中，可选择所有的基本服务，以及高级服务中的 FIFO。

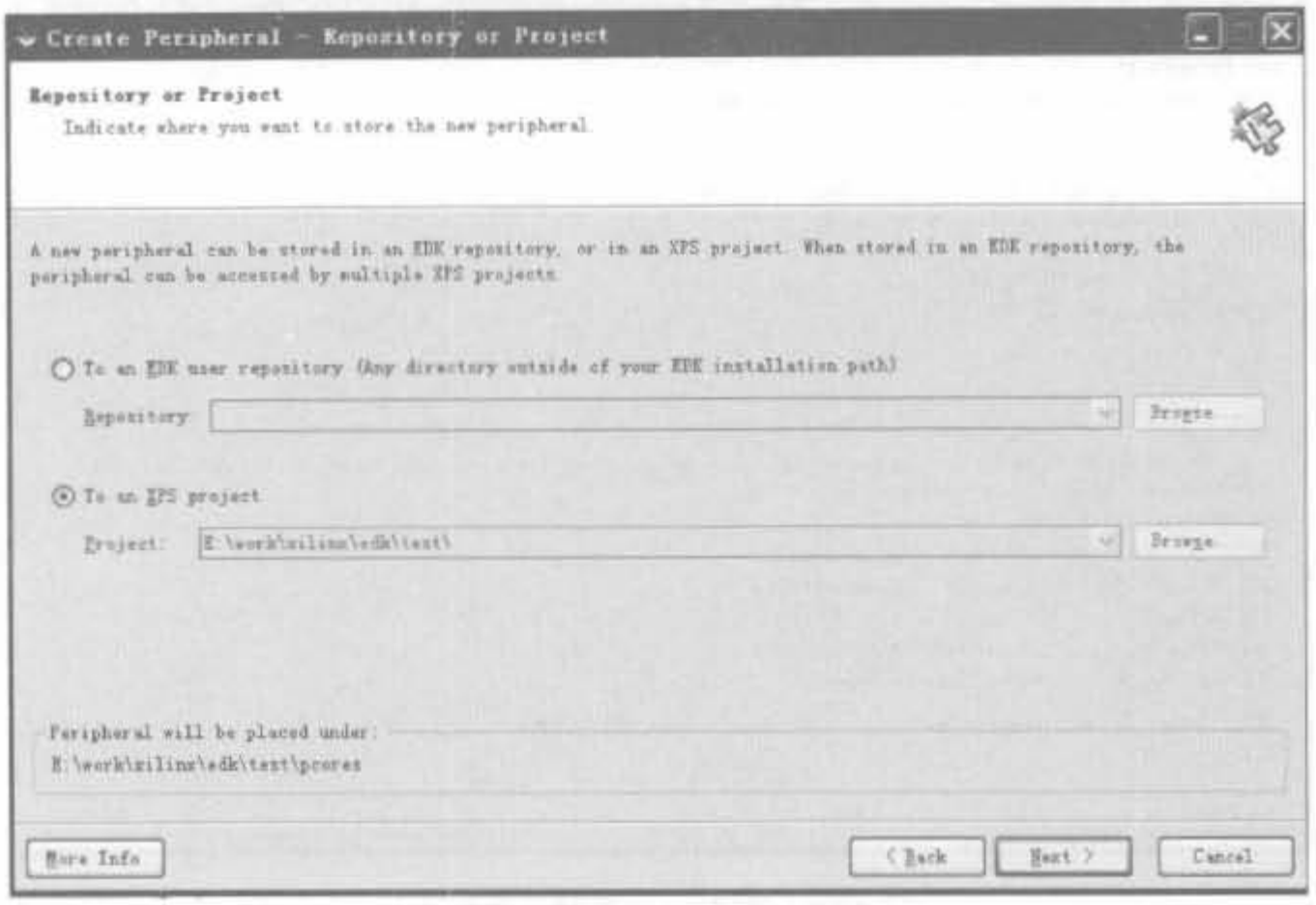


图 9-50 定制 IP 存放目录示意图

图 9-51 定制 IP 的名称和版本

IPIF 为用户提供一些基本的接口和服务，如从设备连接、地址译码等，还有一些可选择的服务，如中断、DAM 等。在 FIFO 服务和中断服务中使用默认值，此步骤需要设定由用户逻辑产生的中断的数量，以及中断的捕捉模式。本例选用默认值，即 ft245bm 的定制 IP Core 没有 FIFO、DMA 以及中断信号，如图 9-53 所示。单击“Next”按钮进入下一页。

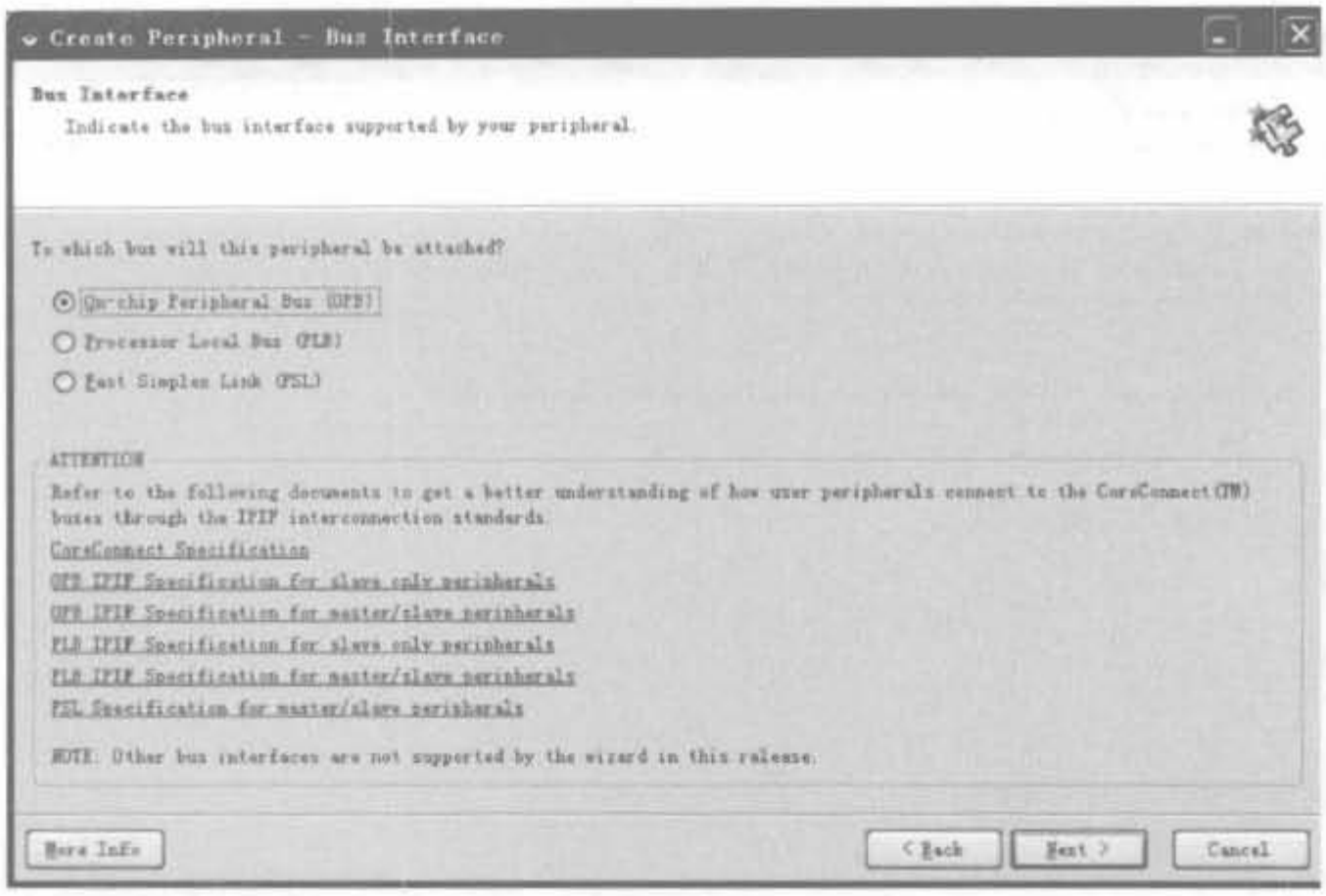


图 9-52　选择 IP 的总线连接方式

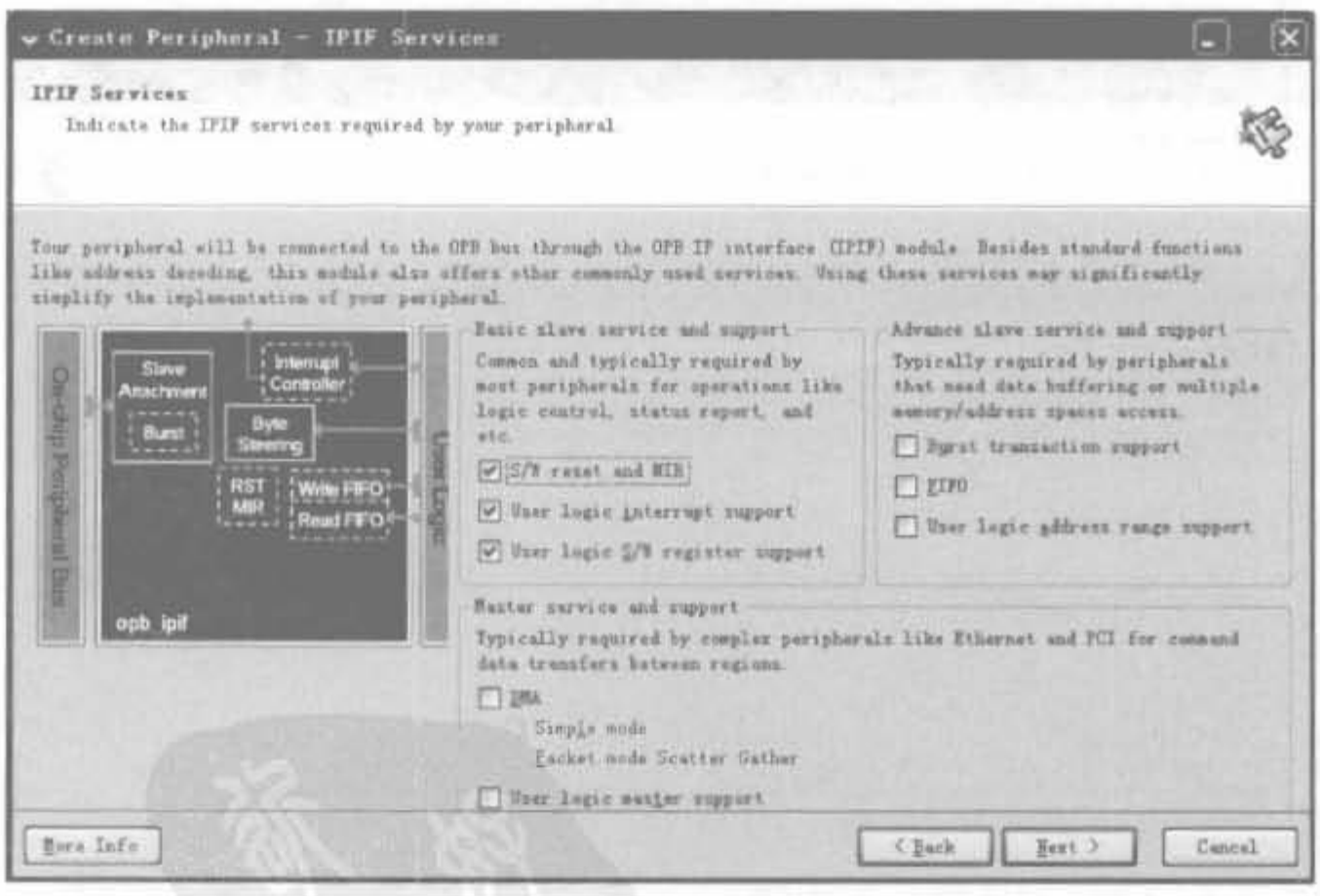


图 9-53　IPIF 服务模式选择

下面对可用的 IPIF 接口机制进行简单说明：

① 基本的从设备支持和服务

S/W reset and MIR：该选项会生成 RST 和 MIR 这两个寄存器。其中，RST 为只写寄存器，软件通过写 RST 寄存器实现对用户逻辑(User Logic)的复位；MIR 为只读寄存器，软件通过读 MIR 寄存器获取外设信息。

User logic interrupt support：该选项可支持用户逻辑通过总线发出中断请求。

User logic S/W register support：该选项可使用户逻辑具备软件可寻址、访问的寄存器。

以上3项对于每种外设都是默认选中的，是任何外设应该所具备的功能，建议用户采用默认值。

② 高级的从设备支持和服务

高级的从设备支持和服务一般应用于需要数据缓冲以及多存储器、地址访问的应用场合，可有效提高总线对外设的数据传输效率，特别适合于有连续、高速数据交互的应用场合。

Burst transaction support：该选项可使能突发传送和缓存线支持。其中，突发传送是一种高速的外设访问方式，可完成用户逻辑和处理器之间的高速数据交互；缓存线使用片外的存储器(支持各类SRAM以及DRAM)作为处理器核缓存的高速通路。需要注意的是，缓存线只能外挂到PLB总线上。

FIFO：该选项可使能IPIF内建的FIFO通路，以实现高速的数据访问。

User logic address range support：该选项可使用户逻辑具备逻辑地址范围的支持，为每段用户地址生成一个使能信号。

③ 主设备的支持和服务

主设备的支持能够带给用户逻辑更大的灵活性和实用性。

DMA：该选项将允许用户逻辑和总线以DMA的方式来交互数据，能提高CPU的工作效率。

User logic master support：该选项可使用户逻辑具备主设备访问的功能。

(6) 选择中断支持

IPIF提供了中断响应服务，并可根据一定的优先级对各个中断信号进行优先级排序。因此，需要为每个用户外设指定中断的个数和等级，以便CPU能更好地调度。这里采用默认值即可，如图9-54所示。单击“Next”按钮，进入下一页。

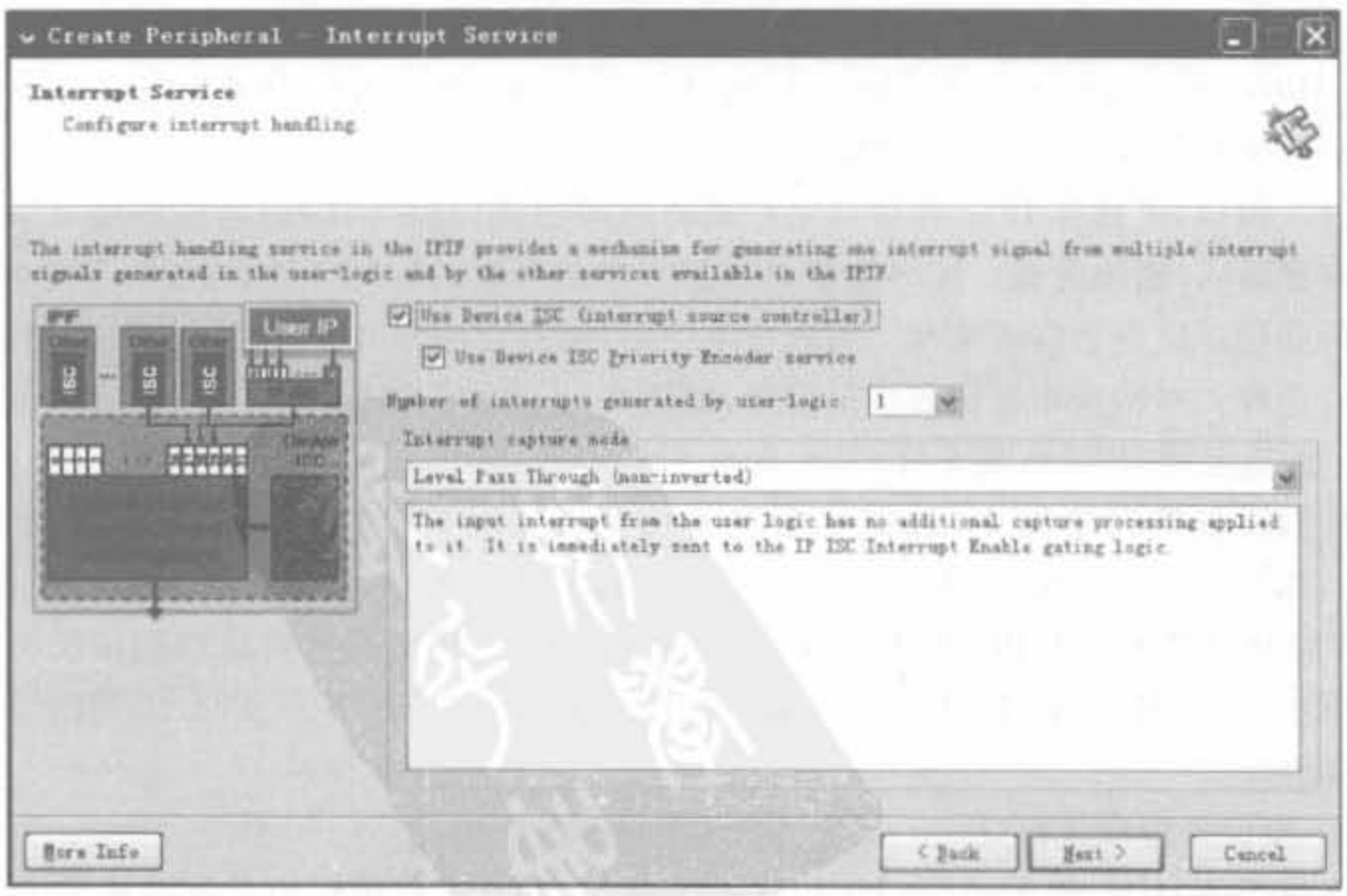


图9-54　用户逻辑的中断参数设置界面

(7) 软件寄存器设置

软件寄存器个数、位宽选择界面如图 9-55 所示，采用默认值：1 个寄存器、位宽为 8bit；寄存器写模式设置为“Enable posted write behavior”。单击“Next”按钮进入下一页。

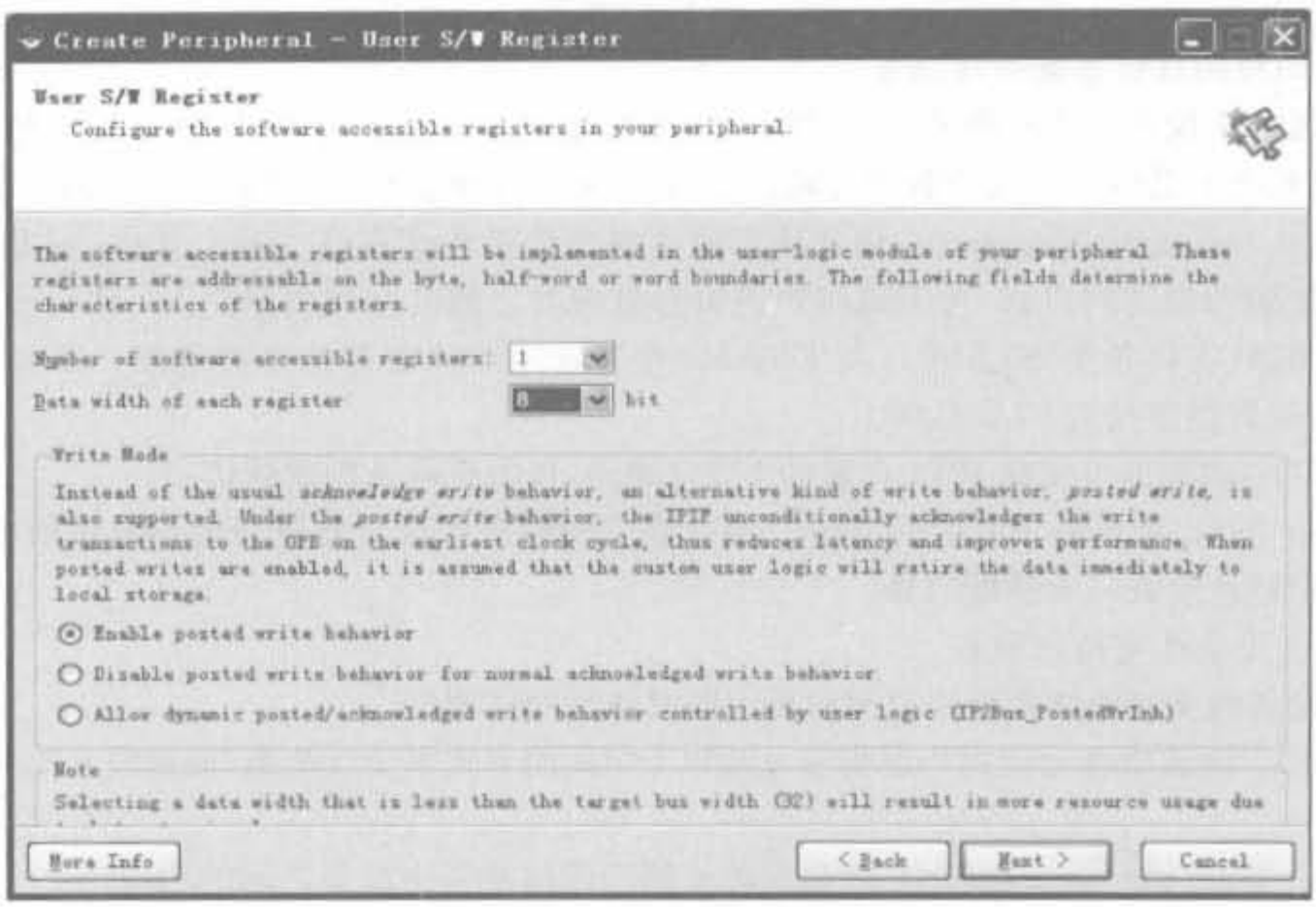


图 9-55　软件寄存器设置界面

寄存器具有 3 种写模式，具体包括：

- Enable posted write behavior；
- Disable posted write behavior for normal acknowledged write behavior；
- Allow dynamic posted/acknowledged write behavior controlled by user logic。

(8) 选择寄存器和 IP 互连(IPIC)

在这个窗口，可选择 IPIF 模块和用户逻辑之间的接口信号，应用这些连接可从硬件外设的寄存器输入/输出数据。XPS 会默认选择一些信号线，是 IPIC 所必需的，不能取消；其余的信号线则可以根据需要添加。当选中某一信号时，会在窗口右边显示相关的功能说明，帮助用户选择。本例选用默认的接口信号，如图 9-56 所示。单击“Next”按钮进入下一页。

(9) 选择定制 IP 的仿真支持

在外围设备仿真支持中，选择外围设备仿真工具可以产生一个总线功能模型文件(BFM)。借助于 ModelSim，可提供将设备连接到总线后的单元以及系统级仿真和验证。此外，BFM 仿真需要安装相关的仿真工具，单击该窗口的超链接可直接访问相关网页。本例选择不生成 BFM 文件，如图 9-57 所示，单击“Next”按钮进入下一页，有兴趣的读者可自行完成实验。

(10) 模板语言选择窗口

XPS 支持 VHDL 和 Verilog HDL 这两种语言，但大多数模板和系统模块都由 VHDL 语言编制。如果使用 VHDL 编写用户逻辑，可参考 XPS 中大量的实例；如果选用 Verilog，

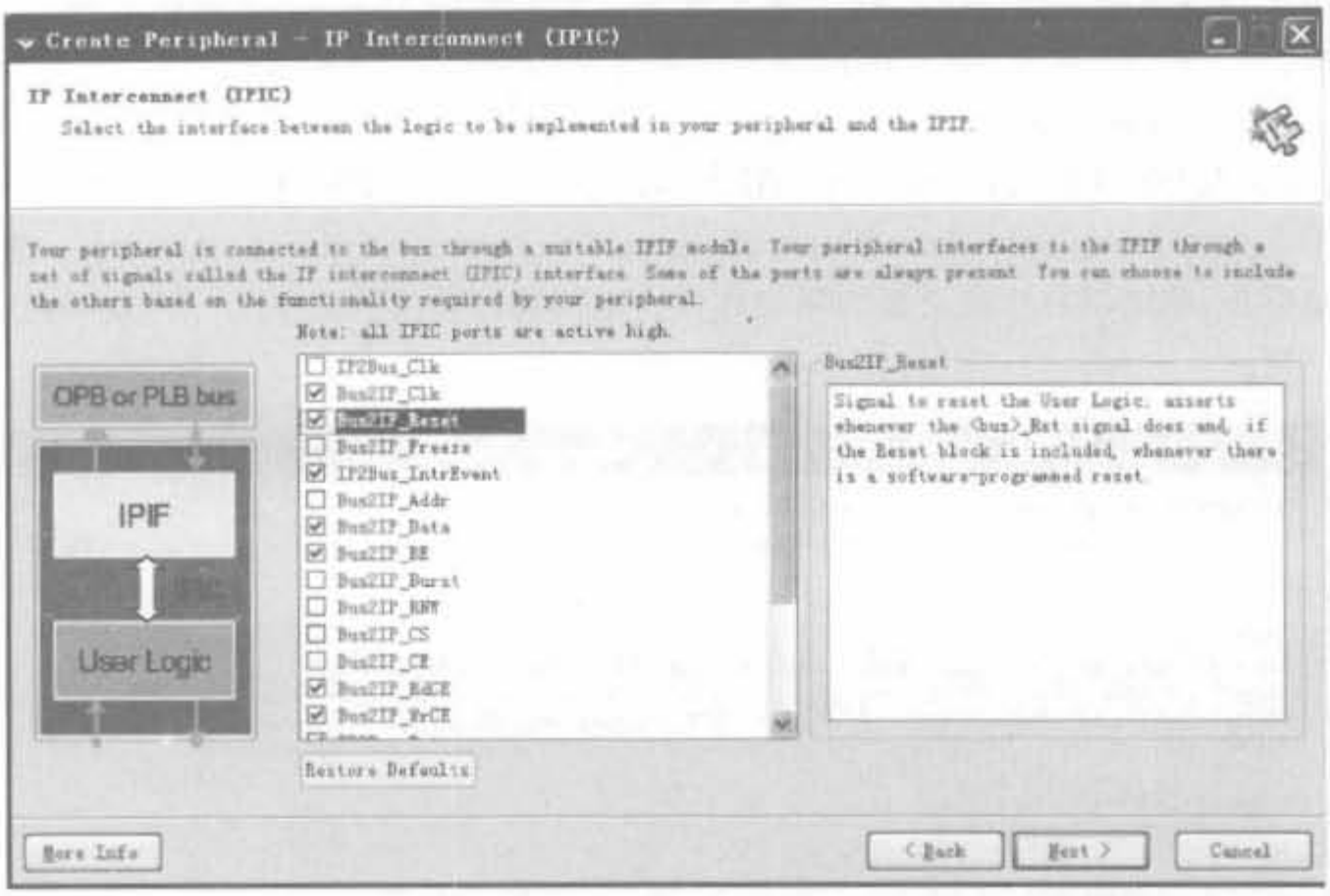


图 9-56　IPIC 的选择界面

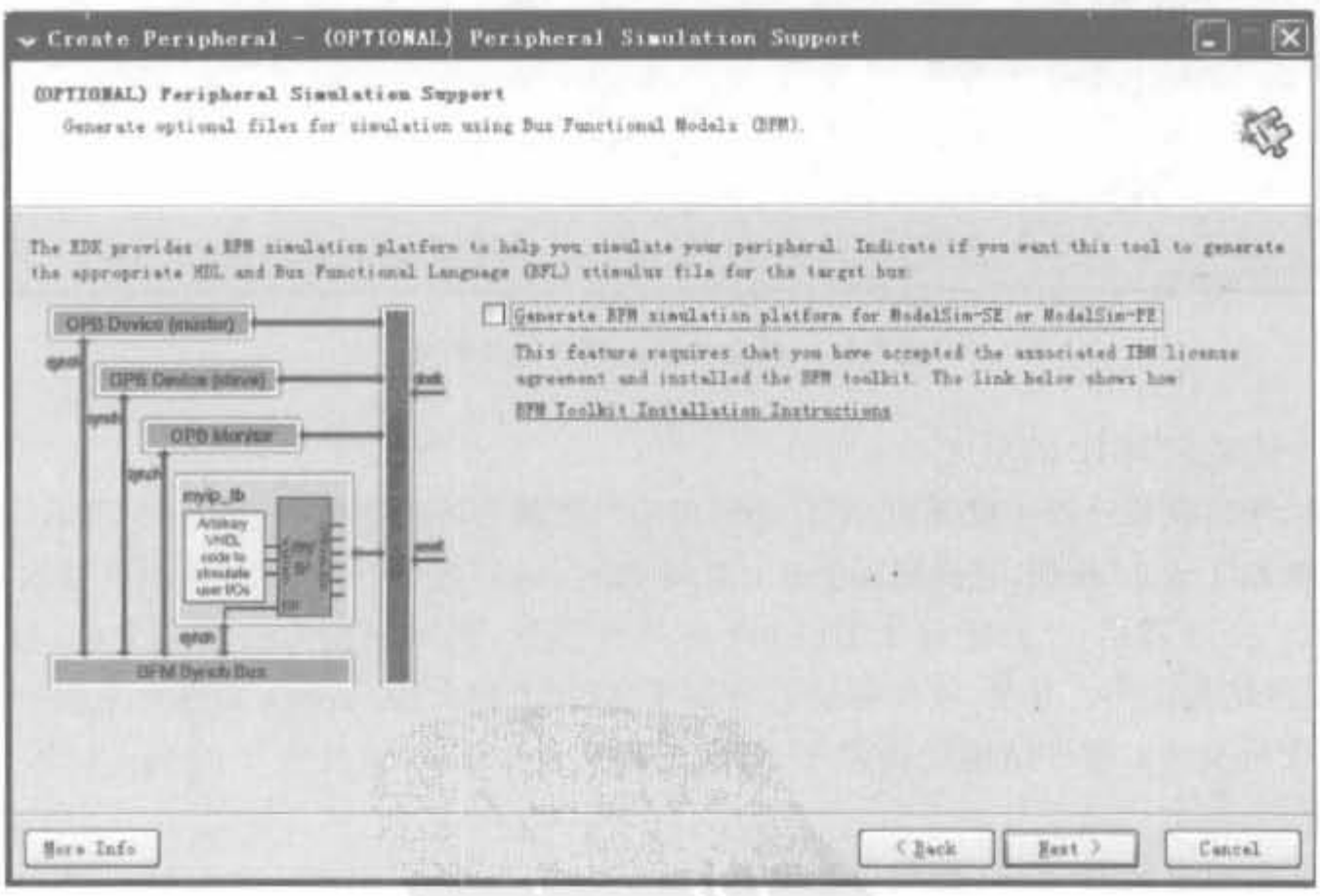


图 9-57　外设代码仿真支持的选择界面

则需要混合编译，且没有太多参考。不过，XPS 工具对混合编译有很好的支持，不会影响到用户逻辑的功能。

模板语言选择窗口有 3 个选项，其相应的功能如下所述：

① Generate stub 'user_logic' template in Verilog instead of VHDL：选择 Verilog 作

为模板中用户逻辑的设计语言。默认值为不选。如果要用 Verilog 实现相关逻辑，需手动选中该项。

② Generate ISE and XST project files to help you implement the peripheral using XST flow：选择生成 ISE 工程文件，帮助用户在 ISE 环境中完成用户逻辑设计。默认为选中。

③ Generate template driver files to help you implement software interface：选择生成软件驱动模板，帮助用户生成必要的驱动软件。本例选择如图 9-58 所示，单击“Next”按钮，进入下一页。

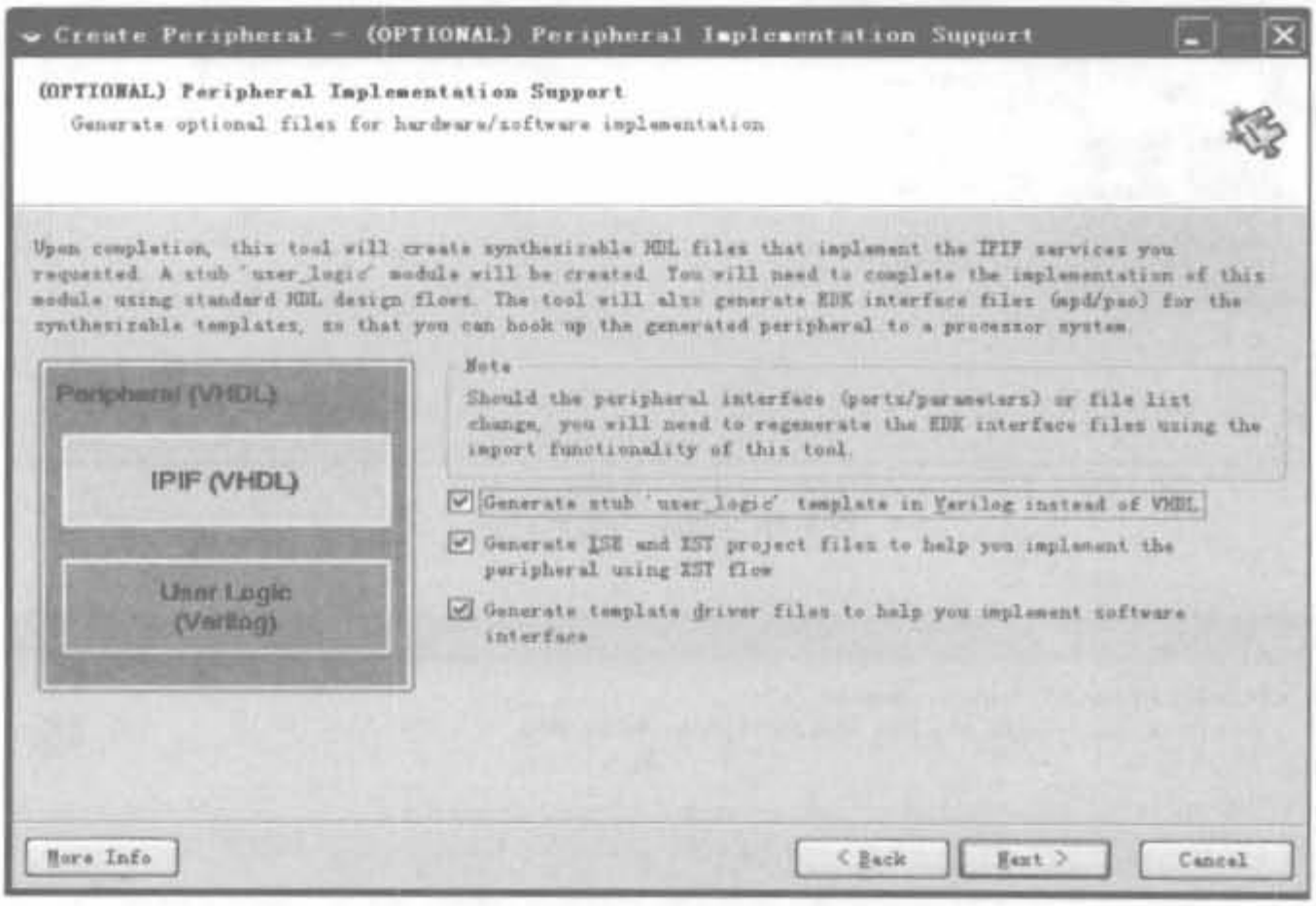


图 9-58　外设代码实现的选择界面

(11) 完成定制 IP 的生成

至此，代码模板已经生成成功，XPS 会列出用户逻辑的各项参数，如图 9-59 所示。如果确认无误，单击“Finish”按钮，完成模板创建；否则单击“Back”钮按，返回到相应页面修改参数。

完成以上步骤后，已经定制了 IP Core 的共性部分，会生成相关的诸多文件，包括软件驱动和硬件结构代码。其中，软件驱动部分位于 XPS 工程文件 drivers 文件夹下的 my_led8_v1 _00a 文件夹中；硬件结构代码位于 XPS 工程文件 pcores 文件夹下的 my_led8_v1_00a 文件夹中。

(12) 实现定制 IP Core 的用户逻辑部分

以上步骤只完成了 IP Core 的声明及其和 OPB 总线端口的连接，没有实现任何用户逻辑。因此用户需要在模板中添加用户逻辑，完成对 OPB 总线的响应。

① 查看向导生成的用户逻辑的模板文件

在 XPS 中选择“File”→“Open”，在“pcores\my_led8_v1_00a\hdl\vhdl”和“verilog”目录中分别列出了“my_led8. vhd”和“user_logic. v”文件。打开 user_logic. v 文件，可以看到用户逻辑的模块声明代码，如图 9-60 所示。

图 9-59　定制 IP 的完成界面

```
51  module user_logic
52  (
53    // -- ADD USER PORTS BELOW THIS LINE ---------------
54    // --USER ports added here
55    // -- ADD USER PORTS ABOVE THIS LINE ---------------
56
57    // -- DO NOT EDIT BELOW THIS LINE ------------------
58    // -- Bus protocol ports, do not add to or delete
59    Bus2IP_Clk,                     // Bus to IP clock
60    Bus2IP_Reset,                   // Bus to IP reset
61    IP2Bus_IntrEvent,               // IP to Bus interrupt event
62    Bus2IP_Data,                    // Bus to IP data bus for user logic
63    Bus2IP_BE,                      // Bus to IP byte enables for user logic
64    Bus2IP_RdCE,                    // Bus to IP read chip enable for user logic
65    Bus2IP_WrCE,                    // Bus to IP write chip enable for user logic
66    IP2Bus_Data,                    // IP to Bus data bus for user logic
67    IP2Bus_Ack,                     // IP to Bus acknowledgement
68    IP2Bus_Retry,                   // IP to Bus retry response
69    IP2Bus_Error,                   // IP to Bus error response
70    IP2Bus_ToutSup                  // IP to Bus timeout suppress
71    // -- DO NOT EDIT ABOVE THIS LINE ------------------
72  ); // user_logic
```

图 9-60　模板文件

由于模板创建的是 CoreConnect 兼容结构，因此不需要为其添加任何额外的逻辑。对于定制 IP 而言，所做的只是对 user_logic. v 文件的修改。

② 在 user_logic. v 文件中添加用户逻辑的端口

在 user_logic. v 中，注释语句“–DO NOT EDIT BELOW THIS LINE————————”和“–DO NOT EDIT ABOVE THIS LINE————————”之间的端口为总线端口，不能修改，否则需要再次按照上述步骤重新声明 IP Core。用户只能在端口声明的固定位置添加，如下所示：

```
// -- ADD USER PORTS BELOW THIS LINE ------------------
// -- USER ports added here
my_led;
// -- ADD USER PORTS ABOVE THIS LINE ------------------
```

并在代码段中添加位宽和方向声明，如下所示：

```
output   [0: C_DWIDTH - 1]              my_led;
```

③ 在 user_logic. v 文件中添加用户逻辑

```
assign my_led = slv_reg0;
```

其中，slv_reg0 为一个 8bit 的寄存器，分别驱动 8 个 LED 灯。此外，my_led8. vhd 例化了 IPIF 模块和 user_logic. v。由于修改了 user_logic. v，因此也需要修改 my_led8. vhd 文件，添加相应的端口信号并更新 user_logic. v 的例化模块。

(13) 修改 MPD 文件

每个系统外围设备都有相应的 MPD(Microprocessor Peripheral Description)文件。MPD 文件含有这个系统外设所有可用的端口和硬件参数。对于自定义的外设模块，需要自己定义 MPD 文件。外设向导会自动创建 MPD 文件模板，需要进一步修改才能使用。

MPD 文件位于工程的"pcores\my_led8_v1_00a\data"文件夹中。在 XPS 软件中是不可写的，因此需要利用文本编辑器打开。在"# #Port"行后添加下列语句：

```
PORT my_led = "",DIR = O,VEC = [0:7]
```

并保存 MPD 文件。至此，就完成了一个用户定制外设 IP Core 的建立。

4. 添加定制 IP Core 到 XPS 工程中

添加用户外设和创建外设的命令是一样的，都是通过菜单"Hardware"的"Create or Import Peripheral"命令来实现的，只是在如图 9-49 所示的界面中，要选择选项"Import existing peripheral"。下面通过添加已创建的 LED 外设的实例来详细说明其操作过程。

例 9-7 将例 9-6 创建的 LED 外设添加到 XPS 工程中。

(1) 单击"Hardware"菜单的"Create or Import Peripheral"命令，在如图 9-61 所示页面中选择"Import existing peripheral"。单击"Next"按钮进入下一页。

(2) 选择将创建的外设 my_led8 加入 XPS 工程。单击"Next"按钮进入下一页。

(3) 在源文件类型选择窗口的"HDL Source files"栏。单击"Next"按钮进入下一页。

(4) 在外设源代码属性窗口中选择实现语言类型。由于外设是由 VHDL 和 Verilog 两种语言实现的，因此要选择实现语言为 Mixed 类型，其余配置如图 9-62 所示。其中，mpd 文件和 XST 工程的选择是不可缺少的。单击"Next"按钮进入下一页。

(5) 在源代码分析窗口如图 9-63 中，XPS 通过 pao 文件读取相应语句，完成语法分析。如果该步出现语法错误，相关错误会在 XPS 的控制台窗口输出，并中断导入过程。同样，如果用户发现列举的库文件有遗漏，也可在此添加。单击"Next"按钮进入下一页。

(6) 总线接口窗口列出了 IP 和所使用的总线以及相关信号。由于在创建时，将 LED 挂在 OPB 总线上，因此图 9-64 中选中了 OPB 总线。这一步一般用户进行核查即可，无需修改。单击"Next"按钮进入下一页。

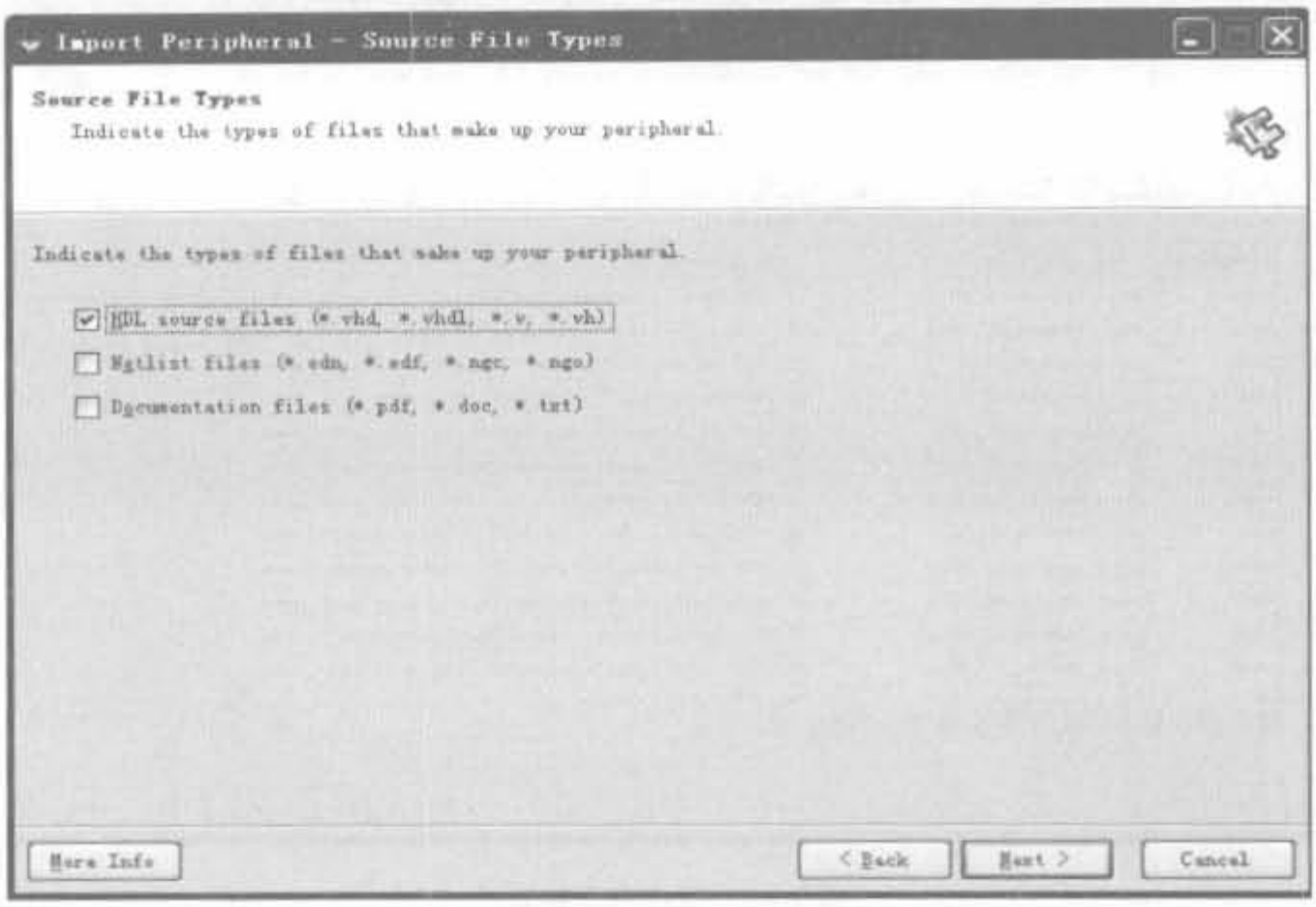


图 9-61 模板文件

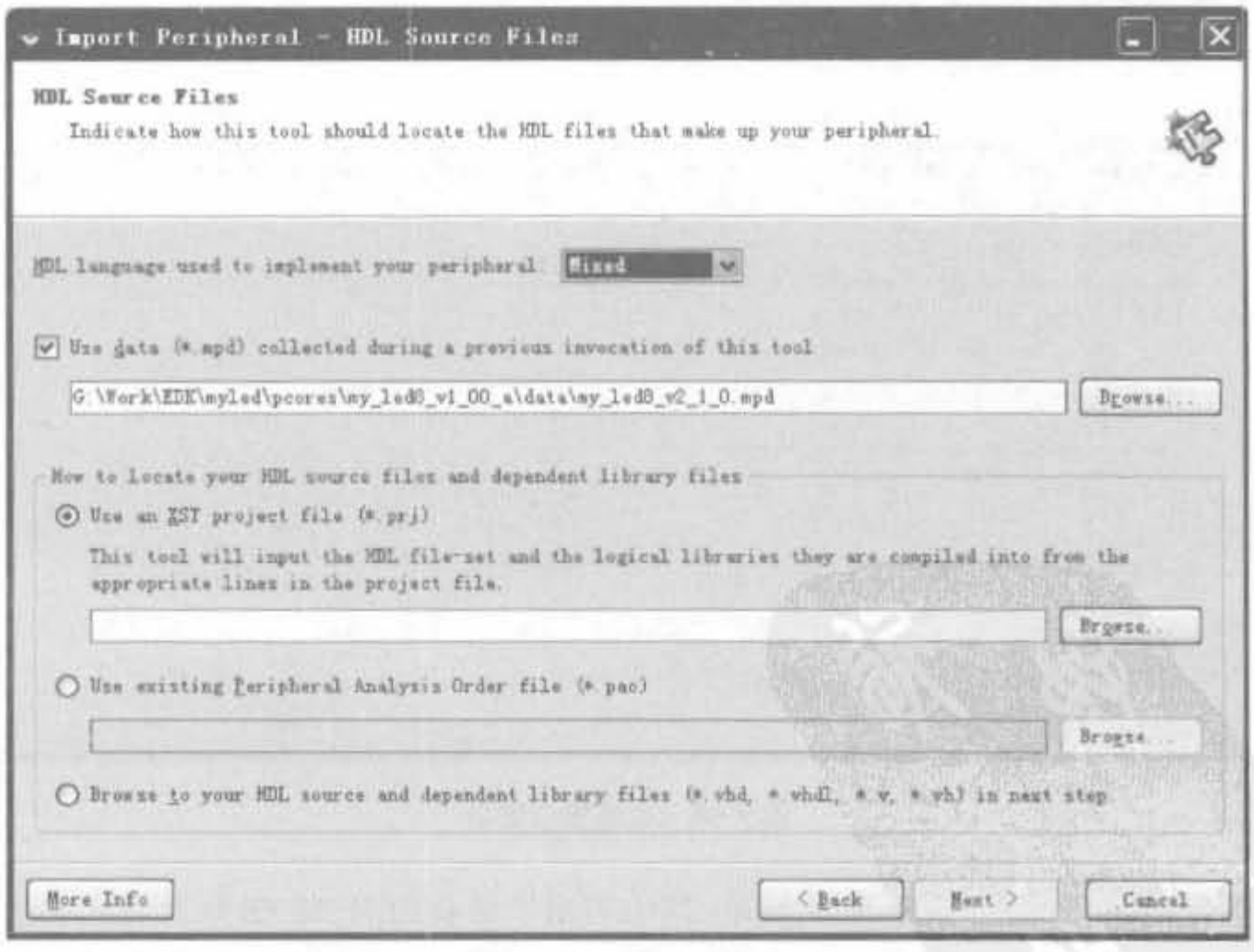


图 9-62 源代码属性设置窗口

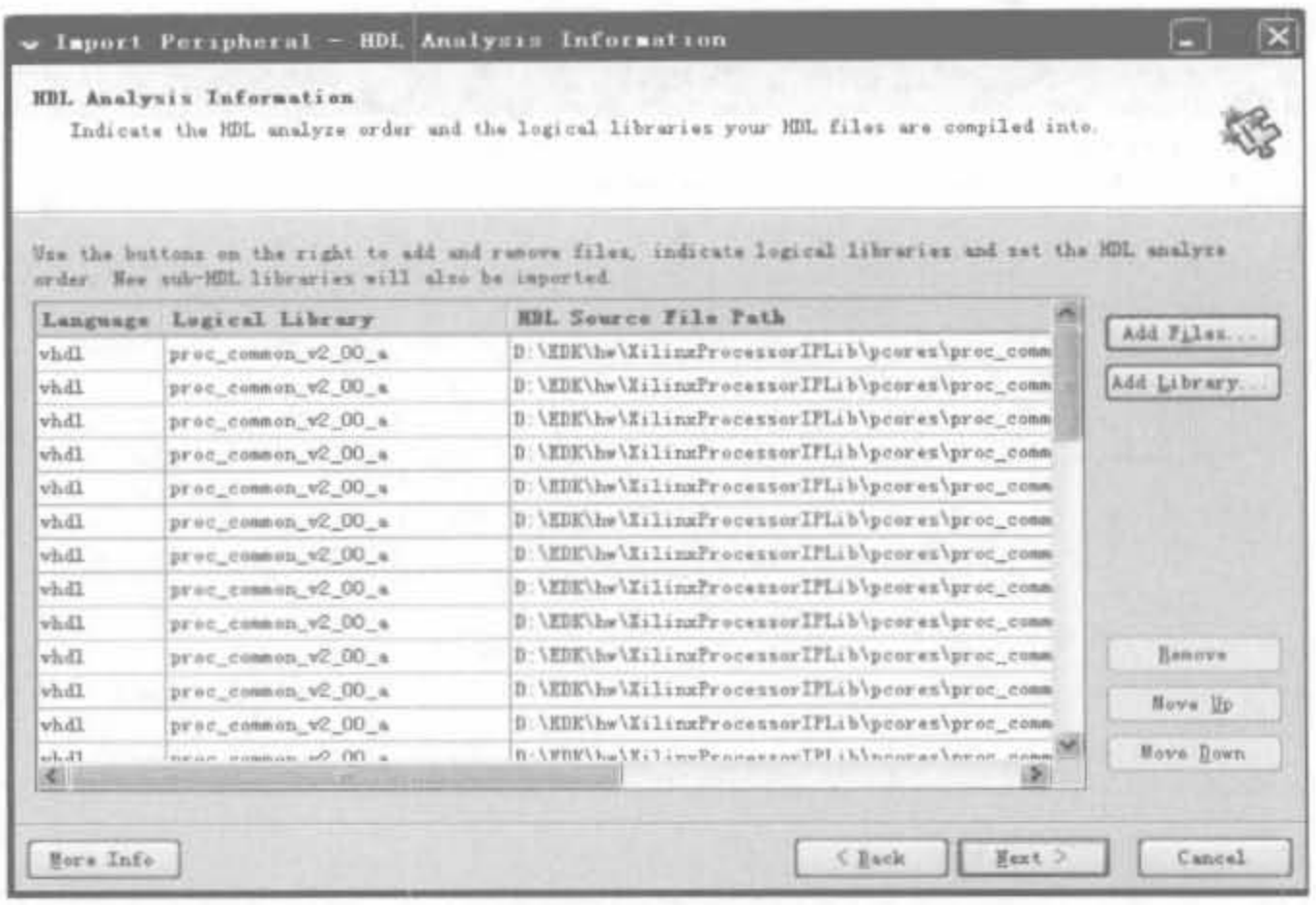


图 9-63　源代码分析窗口

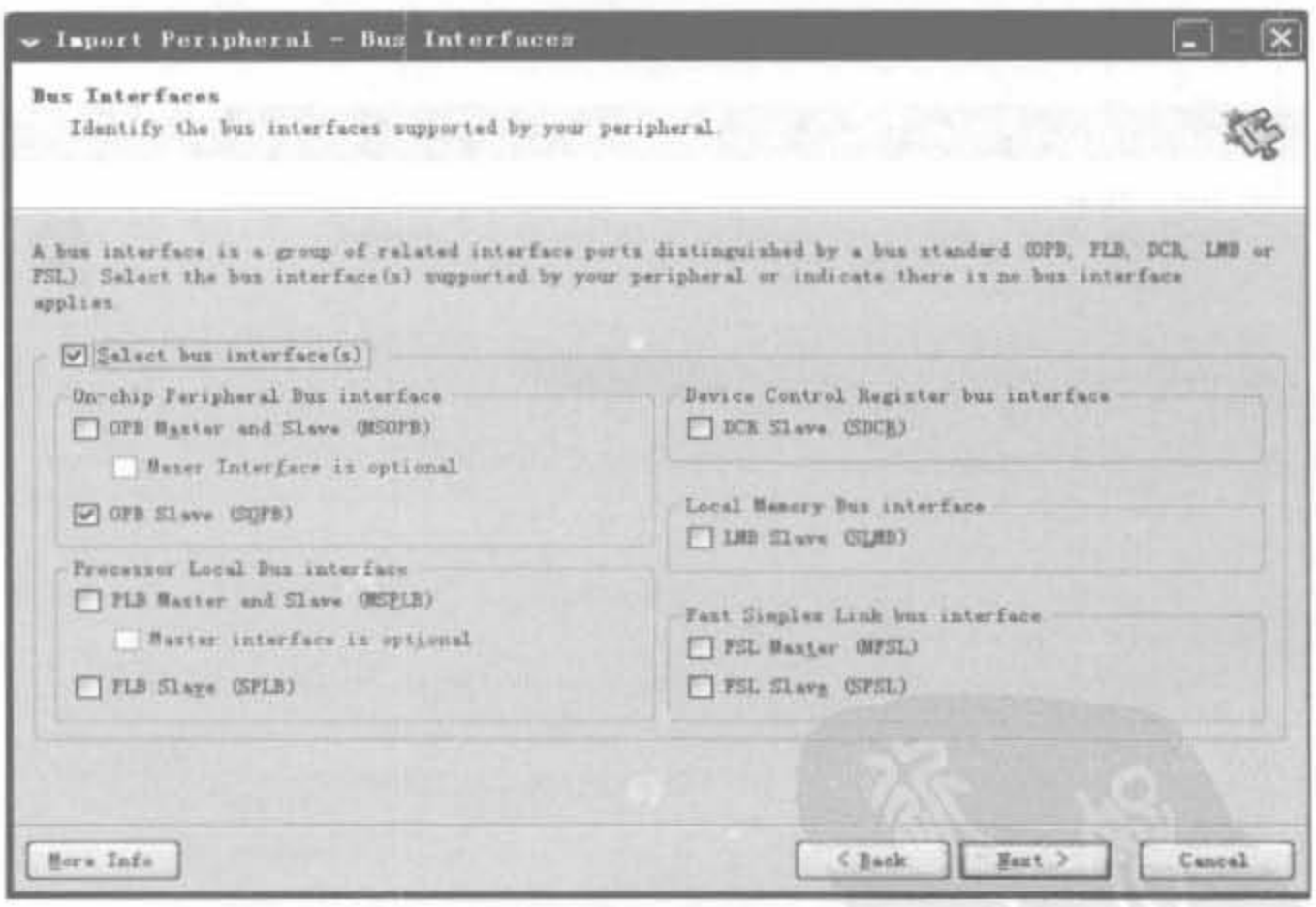


图 9-64　总线接口窗口

(7) SOPB 接口窗口如图 9-65 所示，其中列出了所有的 IP 和 OPB 总线相关信号的连接线以及相应命名。该步是由 XPS 自动完成的，无需修改。单击“Next”按钮进入下一页。

(8) SOPB 总线参数接口如图 9-66 所示，可设置 IP 核寄存器和用户寄存器的地址空间。由于 LED 模块没有存储空间，因此只用设置寄存器空间。单击“Next”按钮进入下一页。

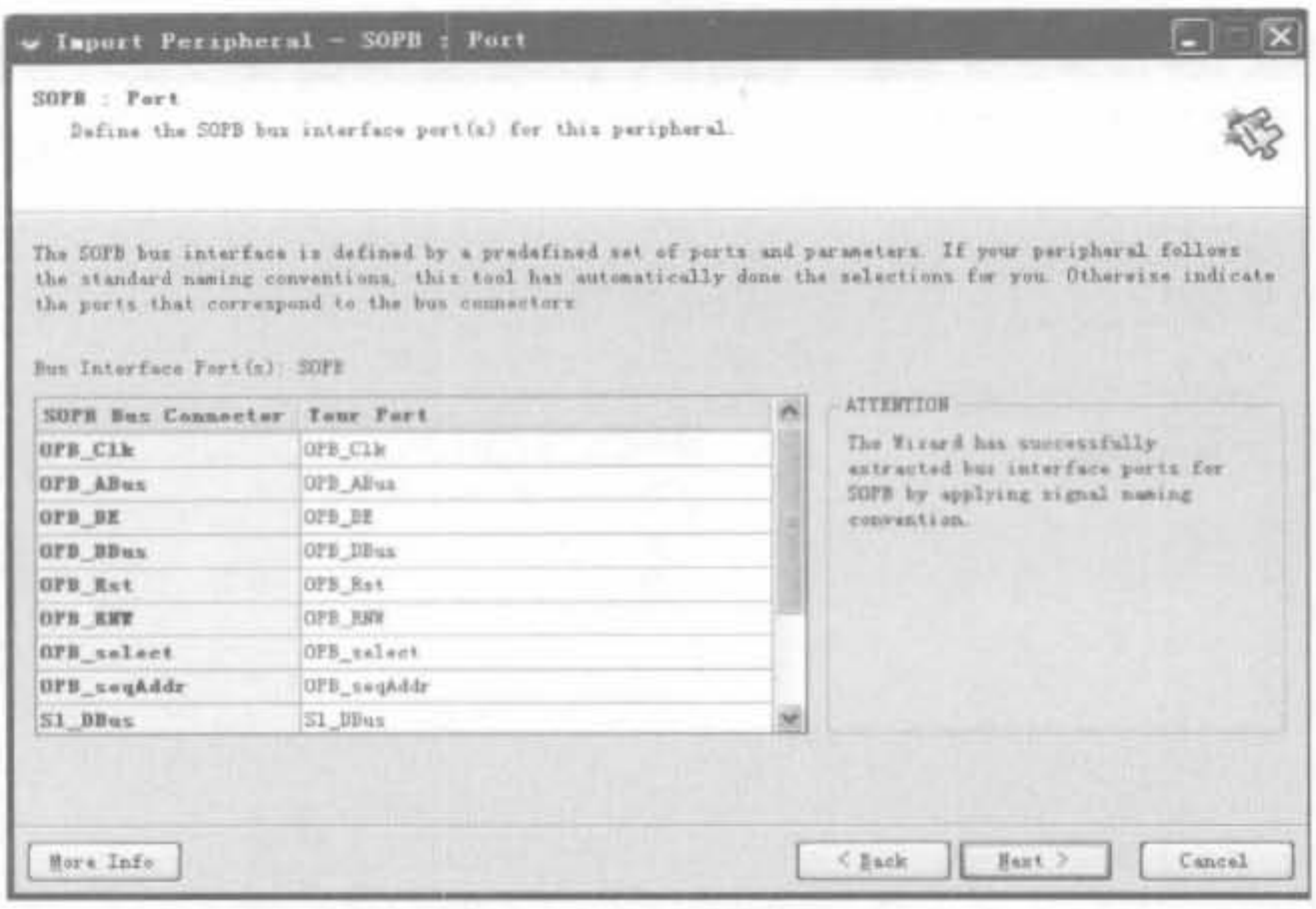


图 9-65　SOPB 总线端口窗口

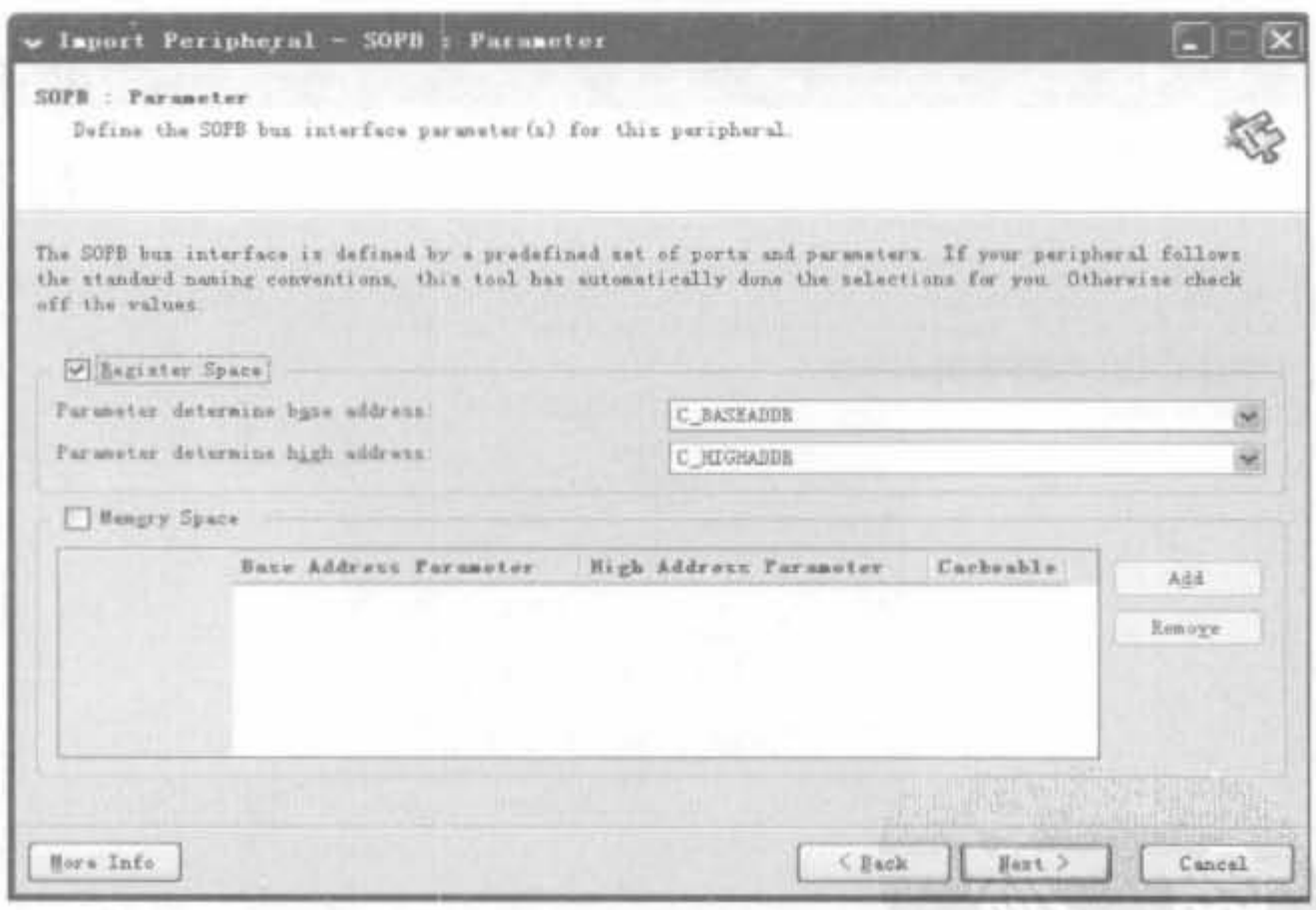


图 9-66　SOPB 总线参数窗口

(9) 中断配置窗口如图 9-67 所示，从中可指定哪个信号作为中断信号。它支持多信号中断，有 4 类中断方式：下降沿(Falling edge sensitive)、上升沿(Rising edge sensitive)、高电平(High level sensitive)以及低电平(Low level sensitive)。单击“Next”按钮进入下一页。

(10) IP 核参数配置窗口如图 9-68 所示。可在“List User Parameters only”下拉框中选择不同的参数，显示不同级别的参数。当选中某个参数时，在“Attributes”栏会列出这个参数的特征，可更改其默认值。修改完毕后，单击“Next”按钮进入下一页。

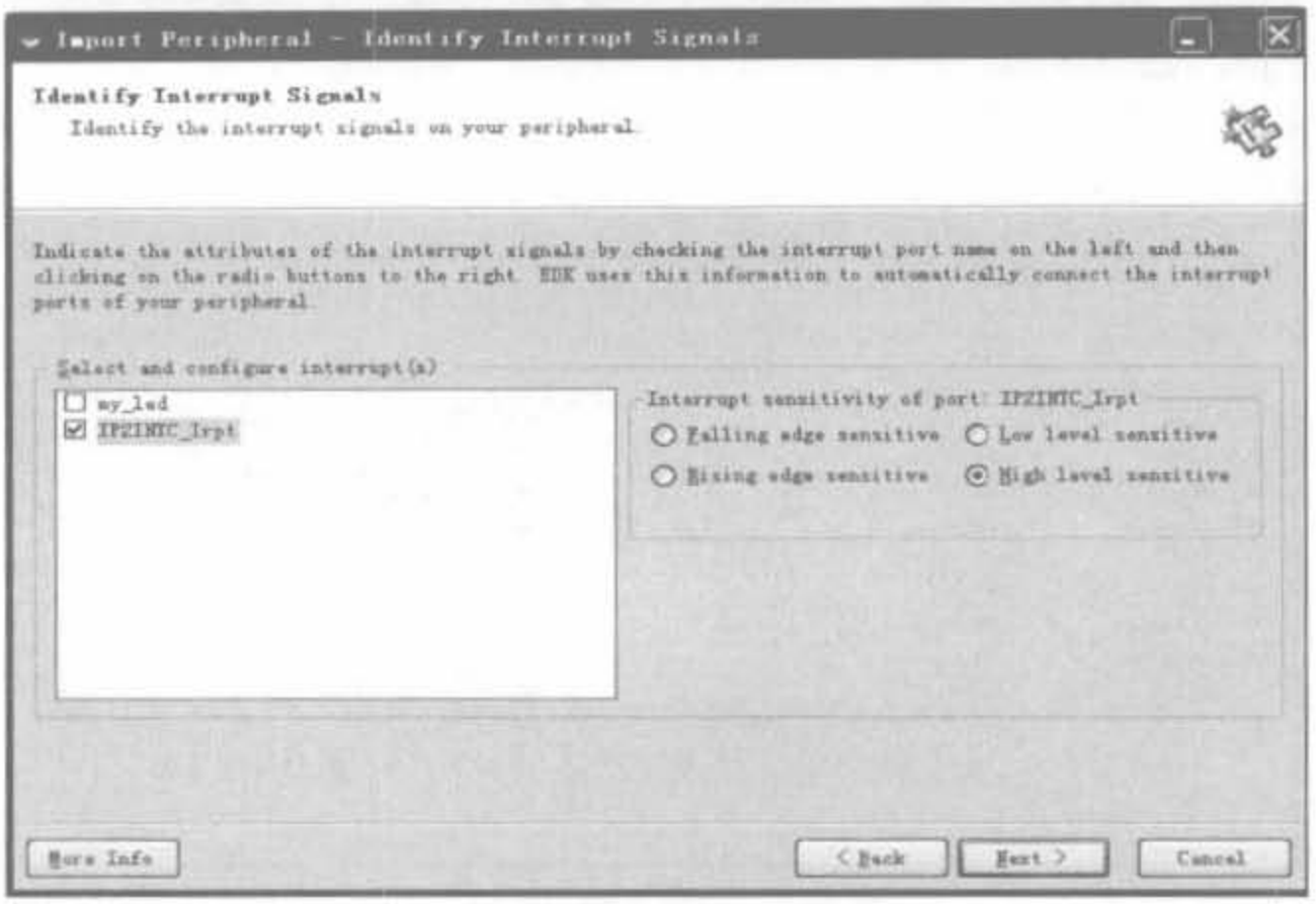


图 9-67　中断支持配置窗口

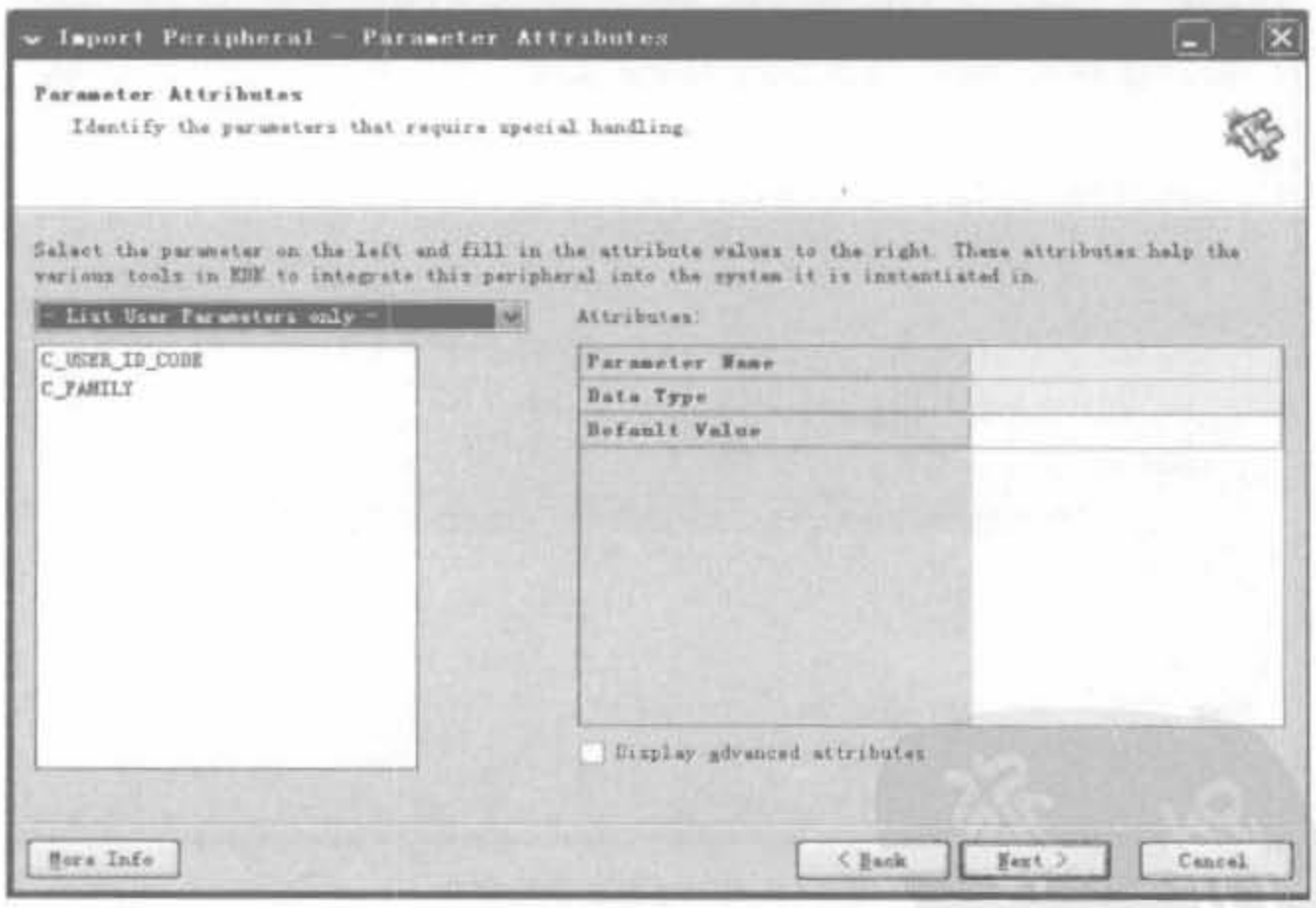


图 9-68　IP 核参数配置窗口

(11) 用户接口参数如图 9-69 所示。可在“List User Ports only”下拉框中选择不同的参数，显示不同级别的端口。对于非用户窗口，一般不建议修改。选中端口后，将在“Attributes”栏列出相关信息，不同类型的端口具有不同的属性配置。在该窗口中完成的任何操作，都会以文本的形式保存在 mpd 文件中，也可以直接通过修改 mpd 文件来达到相同的功能。修改完毕后，单击“Next”按钮进入下一页。

常见的端口属性如下所示：

① Port Name：表明该端口在 IP 核中的端口方向，不能更改。

② Direction Mode：端口的输入、输出方向，由 MPD 文件定义，不能修改。

③ Default Connection：默认连接，标明端口的网表名。

④ Vector Dimension：向量维数，表明总线信号的位宽。

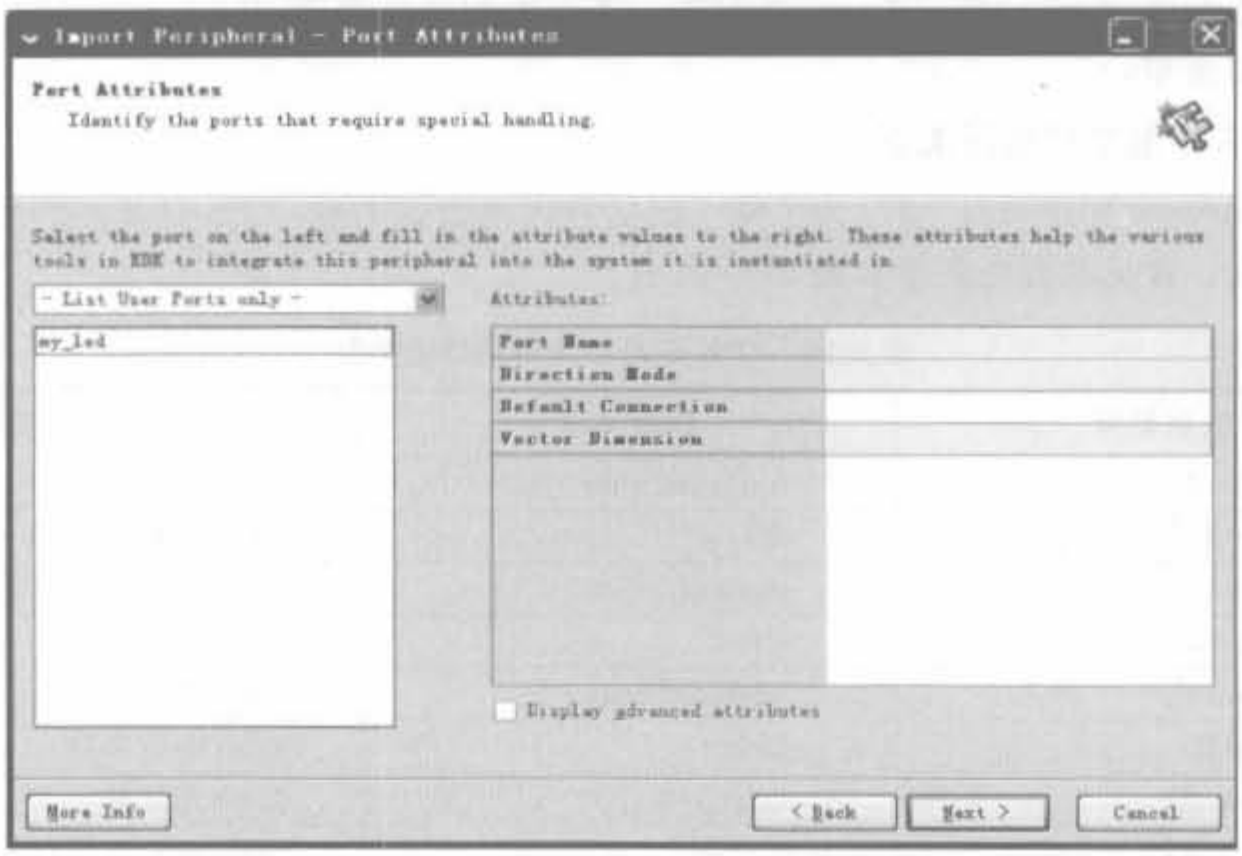


图 9-69　IP 核端口配置窗口

(12) 最后一页是 IP Core 小结，如图 9-70 所示，用户可查阅相关配置参数。如果确认无误，则单击“Finish”按钮，完成整个核的生成；如果有误，需要重新定制 IP，再次导入。

图 9-70　IP 核参数小结示意图

9.4.7 XPS 中 IP Core API 函数的查阅和使用方法

XPS 中自带的 IP Core 包含底层设备驱动以及各类 API 函数，类似于 Windows 编程的 API 函数，它功能强大，降低了设计人员的开发难度，但 XPS 并没有提供类似于 MSDN 的 API 函数查询库，给设计造成了一定的麻烦。本节介绍如何查阅 IP Core 的 API 函数，及其常用的数据类型。

1. XPS 中的常用数据类型

和 Windows 程序开发一样，为了便于记忆和管理数据类型，XPS 对基本的数据类型进行了重定义。常见的数据类型如表 9-10 所示。

表 9-10 EDK 开发中常用的数据类型

数据类型	原始类型	位宽及描述
Xuint8	unsigned char	8bit 无符号数
Xint8	char	8bit 有符号数
Xuint16	unsigned short	16bit 无符号数
Xint16	short	16bit 有符号数
Xuint32	unsigned long	32bit 无符号数
Xint32	long	32bit 有符号数
Xfloat32	float	32bit 浮点数
Xfloat64	double	64bit 双精度浮点数
Xboolean	unsigned long	逻辑符号，其取值为 XTRUE 或 XFALSE
Xuint64	结构体	typedef struct{ Xuint32 Upper; Xuint32 Lower; } Xuint64;

注意，所有的数据类型可在工程文件夹“microblaze_0\include”目录下的“xbasic_type.h”以及“xstatus.h”文件查询。和硬件以及外设相关的参数可在“xparameters.h”头文件中查询。

2. 在 XPS 中查阅 API 函数

XPS 中 IP Core 的风格是统一的，下面以 GPIO 的 IP Core 为例介绍如何查阅 API 函数，并对其常用函数进行简要说明。

(1) 将 GPIO 的 IP Core 加入到工程中，然后在总线接口界面中选中“GPIO”并单击鼠标右键，选择“Driver”→“View API Documentation”，如图 9-71 所示。

(2) 在弹出的浏览页面中单击“gpio”→“gpio.h”的超链接，进入 gpio.h 的浏览页面，可以发现两个结构体 XGpio 和 XGpio_Config 以及 16 个 API 成员函数，如图 9-72 所示。在 gpio.h 文件中，可直接查阅 API 函数文档，也可通过“File List”超链接查阅与 GPIO 有关的所有.h 和.c 文件的说明文档。本书已在 9.2.3 节中给出了 GPIO 模块常用函数的说明，这里就不再重复。

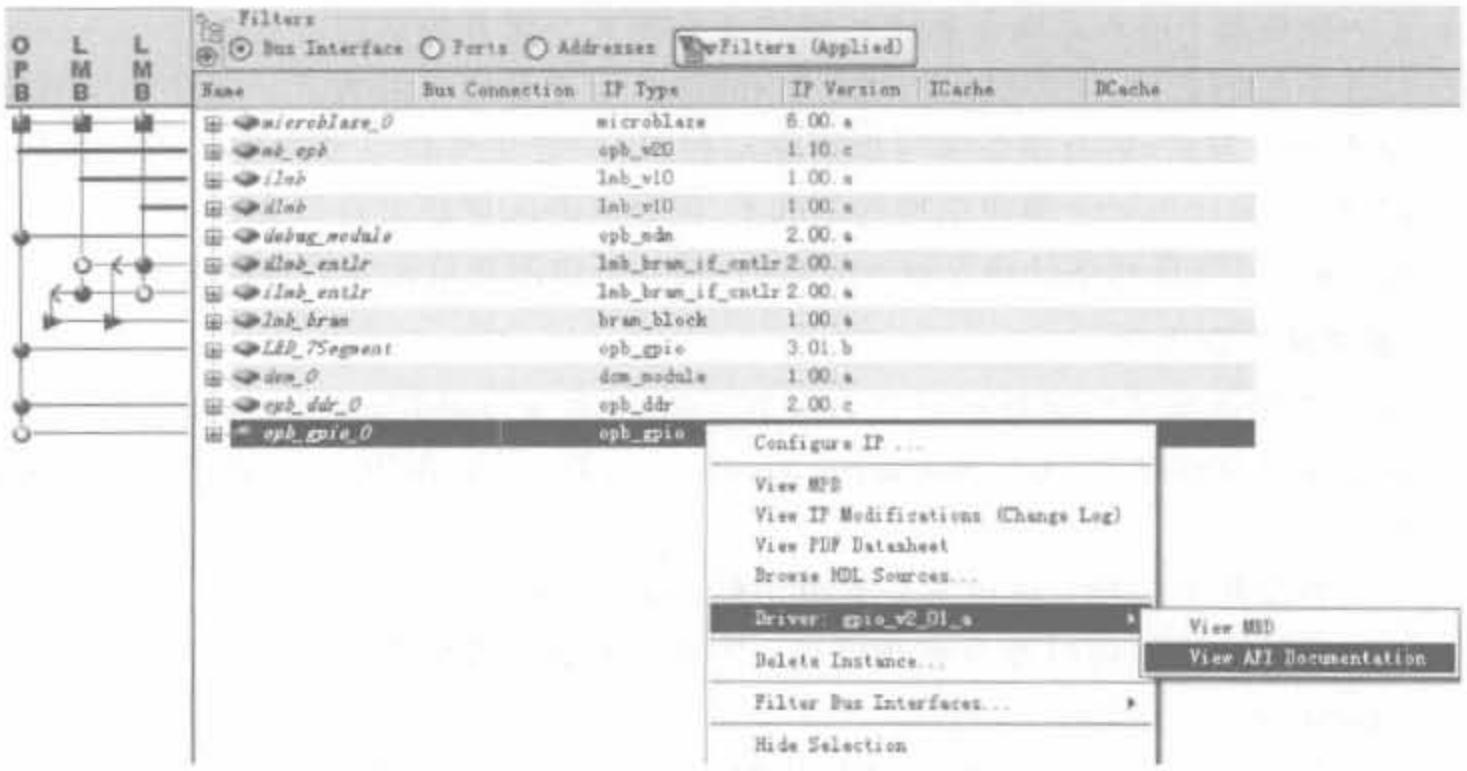


图 9-71 查阅 GPIO API 函数

Data Structures

struct XGpio
struct XGpio_Config

Functions

XStatus XGpio_Initialize (XGpio *InstancePtr, Xuint16 DeviceId)
XGpio_Config * XGpio_LookupConfig (Xuint16 DeviceId)
XStatus XGpio_CfgInitialize (XGpio *InstancePtr, XGpio_Config *Config, Xuint32 EffectiveAddr)
void XGpio_SetDataDirection (XGpio *InstancePtr, unsigned Channel, Xuint32 DirectionMask)
Xuint32 XGpio_DiscreteRead (XGpio *InstancePtr, unsigned Channel)
void XGpio_DiscreteWrite (XGpio *InstancePtr, unsigned Channel, Xuint32 Mask)
void XGpio_DiscreteSet (XGpio *InstancePtr, unsigned Channel, Xuint32 Mask)
void XGpio_DiscreteClear (XGpio *InstancePtr, unsigned Channel, Xuint32 Mask)
XStatus XGpio_SelfTest (XGpio *InstancePtr)
void XGpio_InterruptGlobalEnable (XGpio *InstancePtr)
void XGpio_InterruptGlobalDisable (XGpio *InstancePtr)
void XGpio_InterruptEnable (XGpio *InstancePtr, Xuint32 Mask)
void XGpio_InterruptDisable (XGpio *InstancePtr, Xuint32 Mask)
void XGpio_InterruptClear (XGpio *InstancePtr, Xuint32 Mask)
Xuint32 XGpio_InterruptGetEnabled (XGpio *InstancePtr)
Xuint32 XGpio_InterruptGetStatus (XGpio *InstancePtr)

图 9-72 gpio.h 中的数据和函数列表

9.5 XPS 软件的高级操作

XPS 是完整的嵌入式处理器系统硬件、软件集成开发工具，其功能包括创建工程、输入嵌入式系统设计、生成系统的硬件平台(Platgen)、仿真系统的行为和时序功能、实现并生成编程比特流以及对 FPGA 芯片配置的功能。本节主要介绍基于 XPS 完成嵌入式设计的完整流程。

9.5.1 XPS 的软件输入

EDK 支持的软件方式有两种：一种是 Standalone 方式，软件代码直接运行在裸 CPU

核上；另一种是基于嵌入式操作系统的软件开发模式。前者的运行方式无疑是最快的，虽然缺少操作系统的支持，能实现的功能也是有限的，但是最简单的方式。对于初学者来讲，使用 Standalone 方式，可直接在编写板级驱动包、用户定义外设以及软件代码后直接运行系统。此外，在 Standalone 简单结构的基础上，配合 Xilinx 的库文件和大量 IP，不仅可直接操作硬件，还可以实现网络处理等较为复杂的功能，是应用简单设备的最佳选择。

1. 源文件类型简介

XPS 的应用软件工程都是基于 C/C++语言的，如 9.4.3 节所述，在 XPS 工程信息面板的“Application”页面单击“Add Software Application Project”按钮，即可添加一个新的软件应用工程。

每个软件应用工程都包含以下 5 类源文件：xparameters.h 头文件、.c/cpp 源代码、.h 头文件、.ld 连接脚本文件以及.s 汇编文件。下面分别进行简要介绍。

1）xparameters.h 文件

该文件是 XPS 根据系统结构自动生成的头文件，包含了硬件系统的相关常量定义，如各个硬件单元的基地址、配置参数以及系统的运行频率等参数。典型的 xparameters 文件内容如下所示：

```
/* Definitions for driver GPIO */
// 定义 GPIO 的实例化编号为 3 号
#define XPAR_XGPIO_NUM_INSTANCES 3

/* Definitions for peripheral LEDS_8BIT */
// 定义 GPIO 作为 LED 应用的基地址、高地址等参数
#define XPAR_LEDS_8BIT_BASEADDR 0x40000000
#define XPAR_LEDS_8BIT_HIGHADDR 0x4000FFFF
#define XPAR_LEDS_8BIT_DEVICE_ID 0
#define XPAR_LEDS_8BIT_INTERRUPT_PRESENT 0
#define XPAR_LEDS_8BIT_IS_DUAL 0

/******************************************************************/
// 定义 CPU 工作频率
#define XPAR_CPU_CORE_CLOCK_FREQ_HZ 100000000
/******************************************************************/
```

2）c/cpp 文件

c/cpp 文件用户源程序基于标准的 C/C++语言，是应用程序的主要部分，也是需要用户手动添加的部分，根据需求编写相关代码。

3）h 文件

h 文件是用户编写的头文件，与 VC 环境中的用法是一致的。

4）ld 文件

ld 文件用于连接用户源代码、用户库以及 XPS 库文件，并指定生成的目标二进制文件保存具体的配置信息。

5）s 文件

s 文件是用汇编文件编写的，用于设置 CPU 核的指令数据缓存、处理中断等操作，一般

用于 BSP 和 Bootloop 应用中，不需要用户修改。

2. 软件编译设置

XPS 提供了设置软件编译、连接环境的快速配置窗口。在相应的软件工程上单击鼠标右键，选择“Set Complier Options”命令，可弹出衍生子窗口，包括编译环境页面、调试和优化页面以及路径页面 3 个页面。

1）编译环境页面

该页面如图 9-73 所示，在“Application Mode”栏选择可作为产品发布的执行模式(Executable)和具备辅助信息的调试模式(XmdStub)。“Output ELF file”栏用于设定编译后生成的目标二进制文件的保存路径。“Linker Script”栏用于设定连接脚本文件，在大型程序设计中应该使用用户定制的连接脚本文件，即选中“Use Custom Linker Script”；若对于小型程序，比较适合使用默认连接文件，可选用“Use Default Linker Script”。

图 9-73　软件编译环境页面

2）调试和优化页面

该页面如图 9-74 所示。其中，“Optimization Parameters”栏的“Optimization Level”用于设定优化层次，可选择无优化、低、中、高层次，以及生成最小目标二进制文件 5 个级别进行优化。用户要根据不同的需求来选择。一般来讲，在调试时最好选择无优化选项；当存储空间较小时，可采用最小二进制文件。

3）路径页面

该页面如图 9-75 所示。“Search Paths”栏的-L 行用于添加库文件的查找路径；-I 行用于添加头文件的查找路径。“Libraries to Link against(-l)”栏用于设定加入连接时需要的库文件路径。

图 9-74　调试和优化页面

图 9-75　路径页面

3. 连接脚本文件的配置

熟悉嵌入式开发的读者明白，连接脚本对软件是非常重要的。XPS 为用户提供了简便的配置方法。在“Software”菜单中单击“Generate Linker Script”命令，会弹出如图 9-76 所示的工程选择对话框。在下拉框中选择期望工程后单击“OK”按钮，将打开相应工程的连接脚本配置界面，如图 9-77 所示。

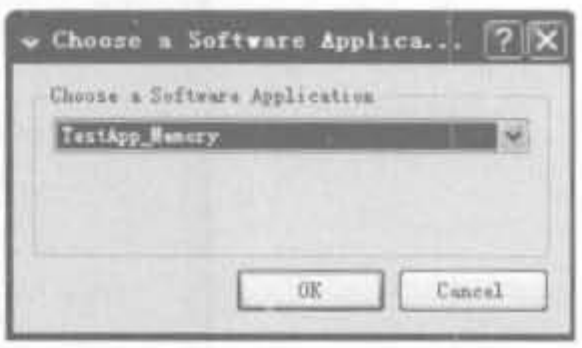


图 9-76　工程选择页面

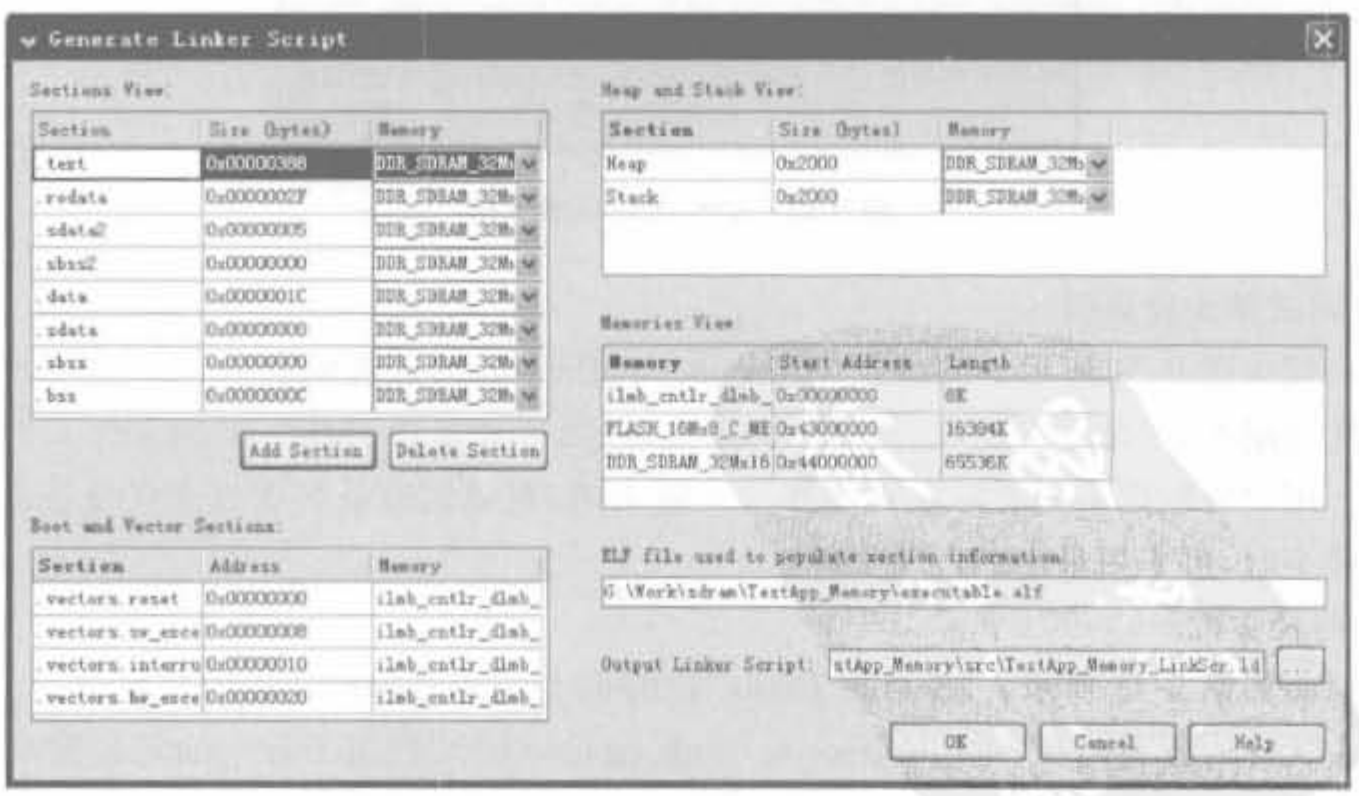


图 9-77　连接脚本配置页面

连接脚本页面的主要内容如下所示：

1) Sections View 栏

该栏用于设置工程最终的二进制比特文件各个段的映射位置，是连接脚本设置的核心。其中，Section 列为各个标识段的名称；Size 列给出各个标识段的大小；Memory 列给出了相应段的映射存储器，可选择处理器片内块 RAM(ilmb_cntlr_dlmb_cntlr)、SRAM、各种 DRAM(SDR、DDR 和 DDR2 等)以及 Flash 等。对于小的应用，可将所有段映射到片内块 RAM。如果文件较大，则必须映射到外部存储器中。

2) Heap and Stack View 栏

该栏标识了程序堆栈空间的大小和映射的存储器。同样，存储器可选择处理器片内块 RAM(ilmb_cntlr_dlmb_cntlr)、SRAM、各种 DRAM(SDR、DDR 和 DDR2 等)以及 Flash 等。

3) Boot and Vector Sections 栏

该栏给出了程序启动段和向量段的起始位置。前者指定芯片上电后第一个开始执行的程序；后者则包含了中断向量表。这两个段只能映射到片上内存中。

4) Memories View 栏

该栏给出了系统所有的存储器信息，包括起始地址和大小信息。该信息来自于 MHS 文件，用户是不能修改的。

5) 输出栏

该栏位于右下角，“ELF file used to populate section information”栏指定了 elf 文件的路径和文件名；“Output Linker Script”用于指定连接脚本文件的存放路径和名称。

9.5.2 XPS 中的设计仿真

1. 嵌入式系统的仿真介绍

如前所述，嵌入式系统开发分为硬件平台开发和软件开发。硬件平台开发即利用 HDL 语言编写系统最低层的驱动程序，每次驱动程序更改后都需要重新下载到 FPGA 芯片进行验证。如果在程序验证中遇到问题，需要在程序中加入一些调试手段，如借助于仿真软件进行功能验证。系统低层驱动程序完成后再编写用户框架程序，但这时已不涉及硬件实现部分，所以程序中的问题用户一般能够发现。因此，嵌入式系统的仿真一般都是指底层硬件模块的仿真。

2. 仿真前条件

(1) 必须有 SmartModel-capable 仿真器(ModelSim PE/SE 或 NCSim)。

(2) 仿真软件的 EDK 仿真库必须已经编译。后文会详细介绍如何生成 ModelSim 的 EDK 软件仿真库。

3. 仿真嵌入式设计的意义

(1) 利用仿真，用户在完成硬件之前就可以测试基本的软件操作函数，这非常有利于软件的开发。

(2) 仿真可以检查系统的内部工作情况。相比硬件而言，在仿真环境中更容易获得信号和寄存器的值。

(3) 在仿真中,用户可以完全控制系统。

4. XPS 仿真基础

XPS 支持在 ModelSim 或 NCSim 逻辑仿真器上嵌入式系统的仿真,通过输出嵌入式硬件平台的 VHDL 或 Verilog HDL 模型,来完成仿真需求。模型所包括的块 RAM(BRAM)存储器外围设备可以通过嵌入式软件 ELF 文件初始化。EDK 可以产生:

- 行为级模型(基于硬件平台规范);
- 后综合结构模型;
- 完全的后布局布线,精确定时模型。

行为级、结构级和时序仿真的验证可以在设计过程中的不同部分进行,如图 9-78 所示。仿真模型生成工具"Simgen"可自动创建 HDL 设计文件并对其进行配置。

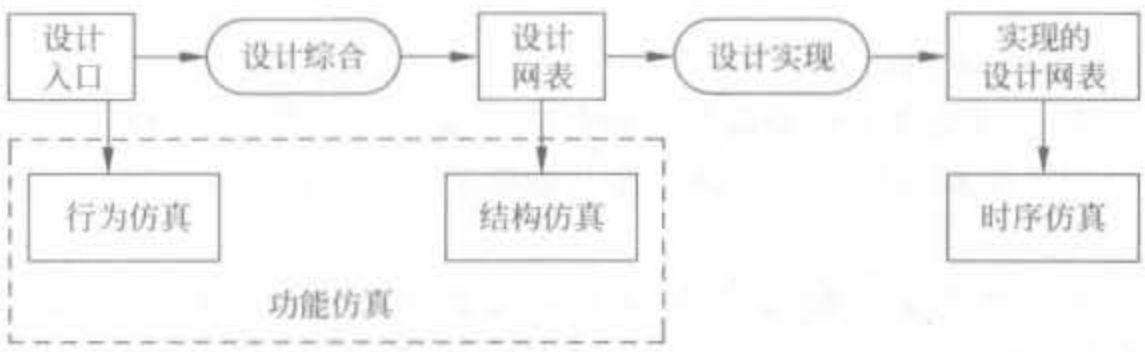


图 9-78 FPGA 设计仿真过程

对于支持 EDK 的仿真器而言,用户在利用它们设计仿真前必须先编译 HDL 库。编译 HDL 库的优点是加快执行速度以及有效利用存储器。仿真库需要用户自行编译才能得到。下面的例子详细介绍编译 ModelSim 软件仿真库的方法和步骤。

例 9-8 在 Windows 平台上生成 EDK 9.1 的 ModelSim 仿真库。

(1) 在 XPS 中,单击"Simulation"菜单下的"Compilier Simulation Libraries"命令,将弹出仿真库的编译向导,如图 9-79 所示。

图 9-79 EDK 仿真库编译向导

(2) 单击“Next”进入仿真工具选择界面，如图 9-80 所示。选择“ModelSim”，同时会列出 PC 上已安装的仿真软件的版本，如其中的“Current simulator version”列出了作者安装的 ModelSim 版本为 6.2.b。需要注意的是，编译 EDK 库，需要 ModelSim 版本在 6.0d 以上(如果选用 NCSim，其版本应在 05.70-s005 以上)。

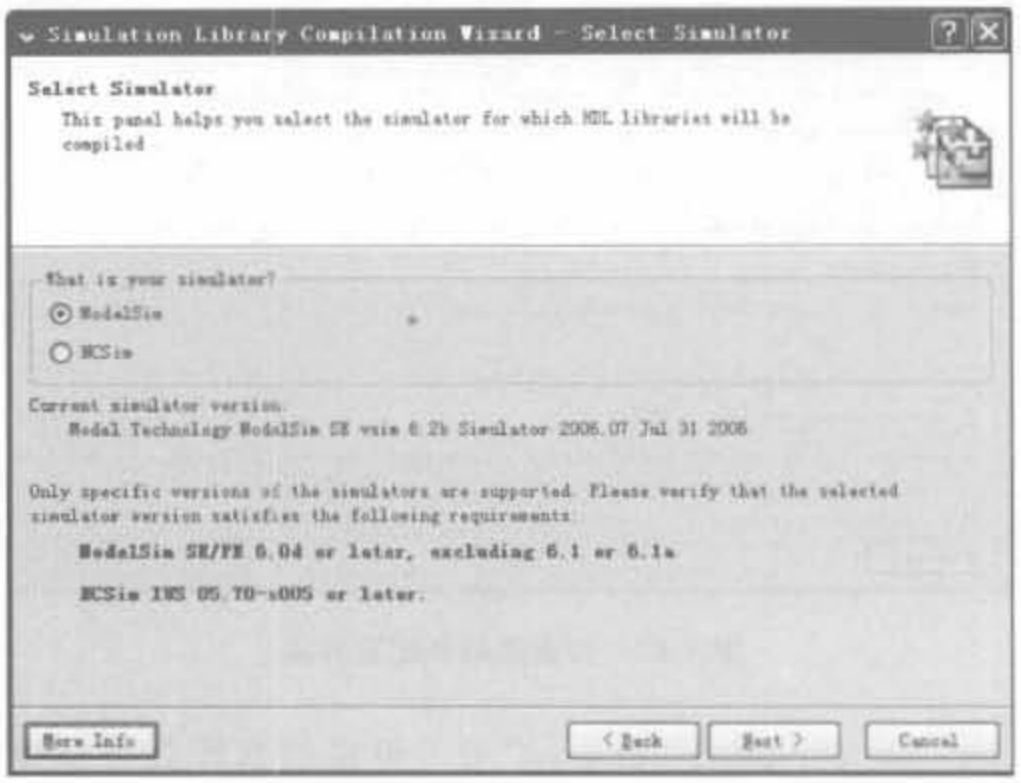


图 9-80　仿真工具选择界面

(3) 单击“Next”进入库所支持的硬件语言类型选择界面，如图 9-81 所示。一般选择“Both VHDL and Verilog”即可。

图 9-81　硬件语言类型选择界面

(4) 单击“Next”进入库编译配置界面。如果是第一次编译，需要选择“Compiler (or re-compiler) the libraries into directory indicated below”；如果已经生成了仿真库，只是修改路径的话，可以选择第二项“Use the complied libraries in the directory indicated below(do

not compiler)",如图 9-82 所示。

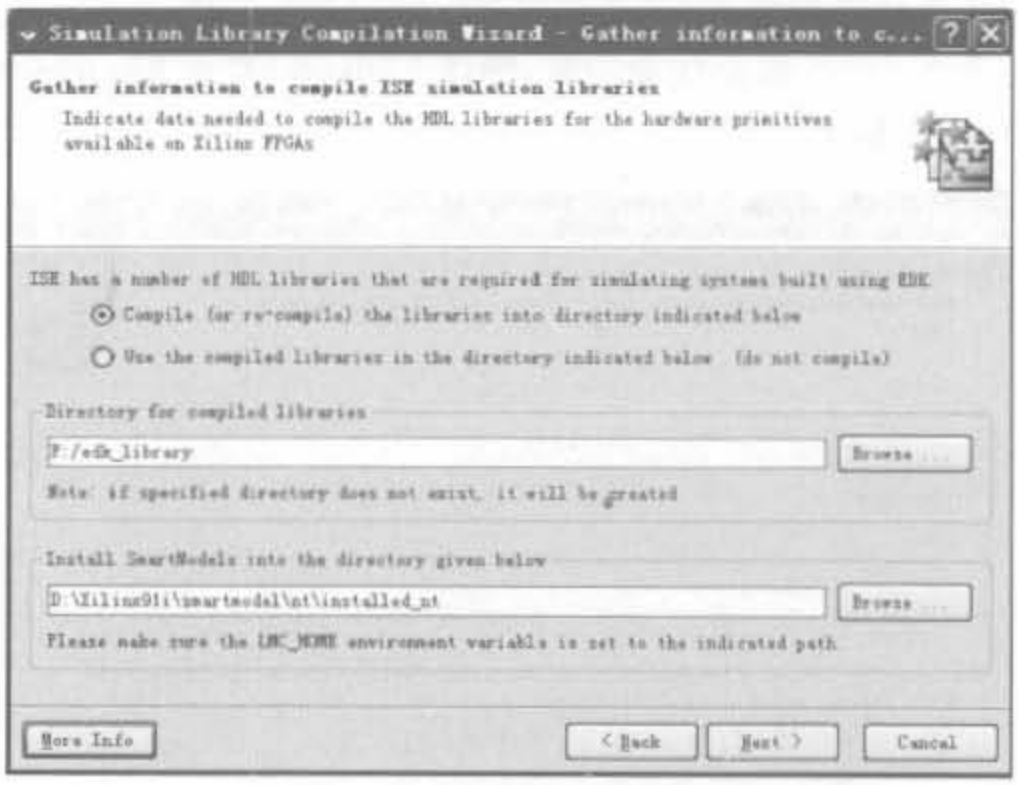


图 9-82　仿真库编译配置界面

其中,"Directory for complied libraries"用于设定仿真库的存放路径,一般不要和 ModelSim、EDK 的安装路径在同一盘符下,否则经常编译失败。如作者的 EDK 和 ModelSim 安装在 D 盘,将 EDK 库的生成路径设定为 F 盘下的根目录。"Install SmartModels into the directory given below"用于设定 SmartModels 的安装路径,需要在环境变量 LMC_HOME 中指定。该选项一般选用默认值即可。

(5) 单击"Next"进入编译路径信息确认页面,如图 9-83 所示。如果路径设置不对,可以在此修改。一般情况下,可以将"I want to compiler other project specific simulation libraries"选项选中,以获得更多的仿真库。

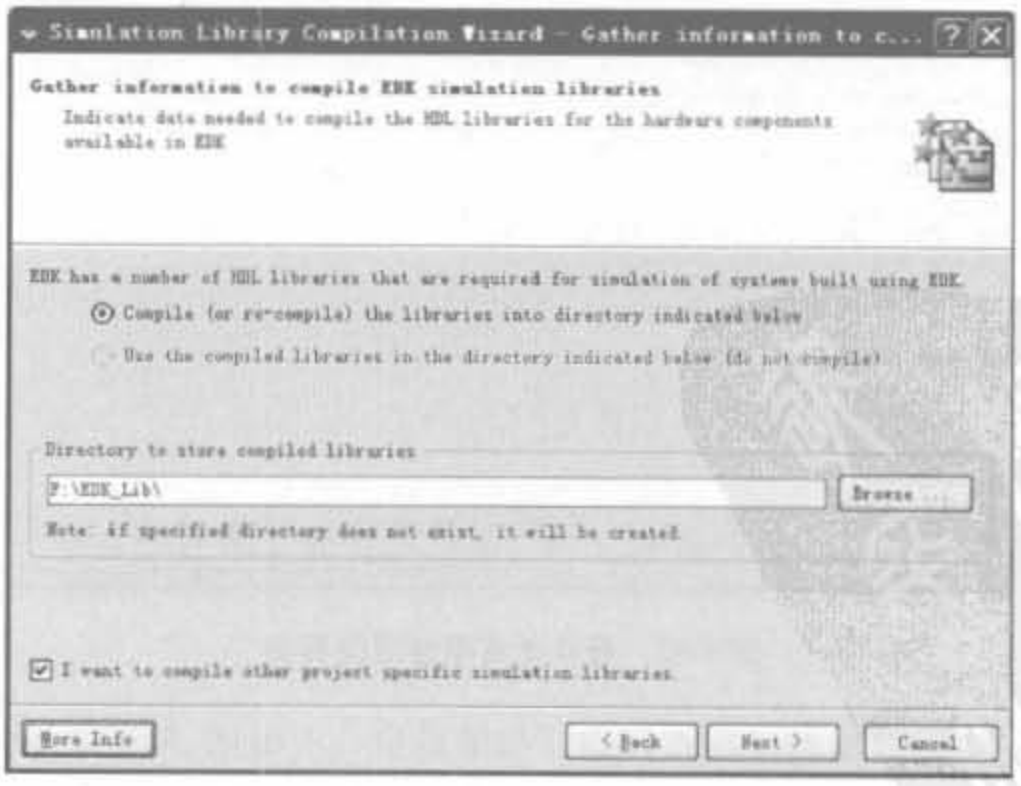


图 9-83　仿真库编译信息确认界面

(6) 单击“Next”进入编译类型选择页面，如图 9-84 所示。选择“Compile all library items”，虽然增加了编译时间，但获得了更全面的库，避免以后的多次编译仿真库。然后单击“Next”进入 XPS 工程中其余的 EDK 组件库选择界面，全部采用默认值即可。

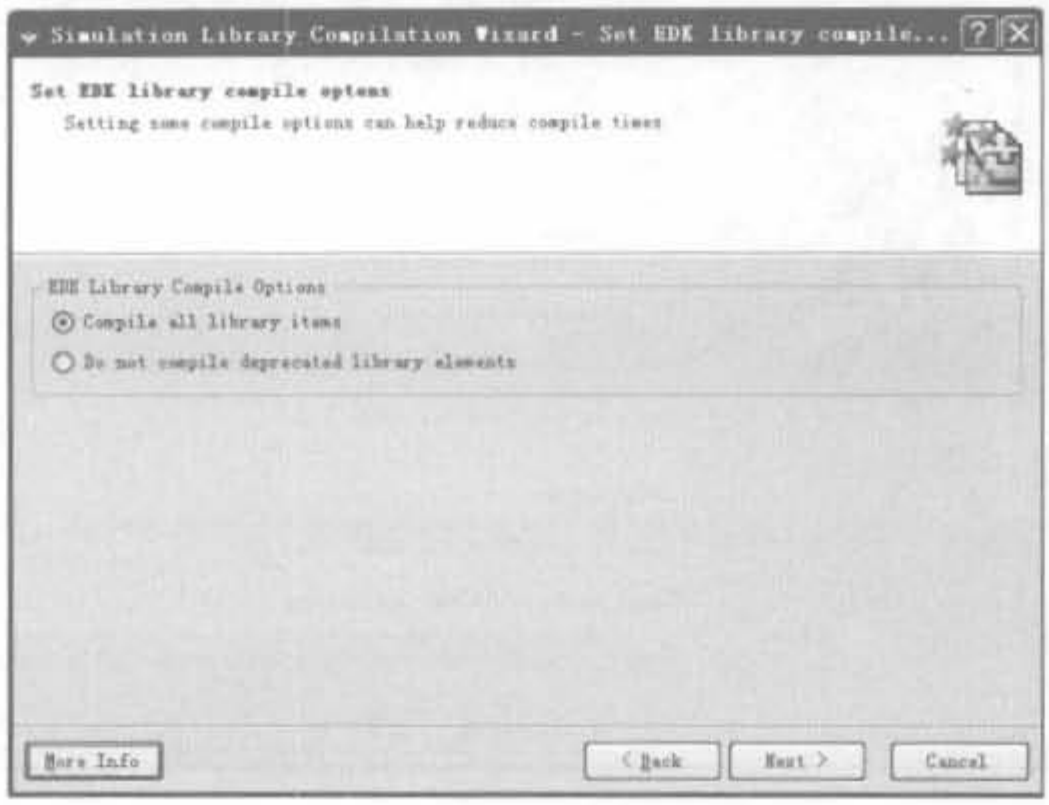


图 9-84　编译类型选择界面

(7) 单击“Next”进入仿真库编译页面，如图 9-85 所示。当编译完成后，两个进度条都到 100%，且“Next”按钮有效。单击“Next”按钮，进入仿真建立需求知识框，没有需要设定的地方，有兴趣的读者可单击查看。

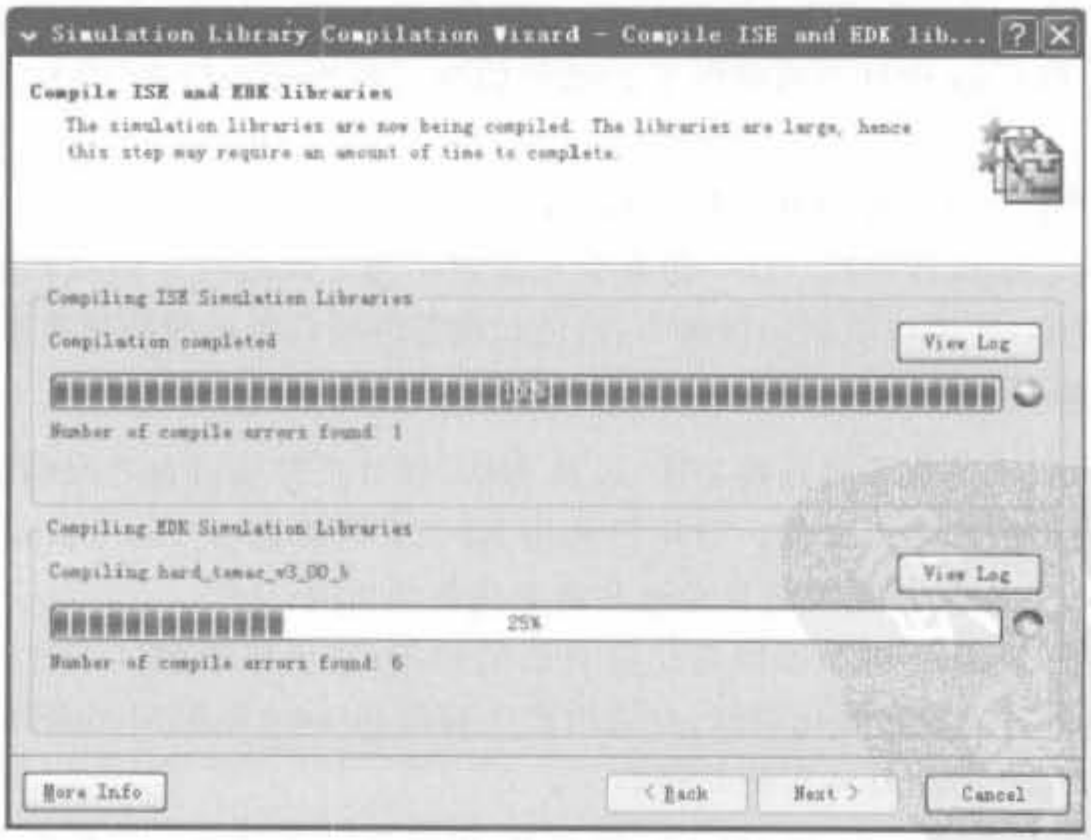


图 9-85　仿真库编译界面

(8) 单击“Next”进入最后一页，即编译完成指示界面，如图 9-86 所示。该界面显示了生成的仿真库的所有信息。读者一定要将“Save as default simulation library paths for new

XPS projects"的选项选中，以后就不用在新工程中重新编译。单击"Finish"按钮，完成整个仿真库的编译过程。

图 9-86　仿真库编译完成指示界面

5. 仿真应注意的问题

在仿真嵌入式系统时，必须考虑到下面几点：

1）某些系统值已得到了设定

设置全局复位、三态线以及时钟信号为一定的数值。Xilinx 综合软件环境(ISE)工具在仿真 VHDL 或 Verilog 设计方面提供了详细的信息。有兴趣的读者可以参见 ISE 的综合验证设计手册。

2）改变某些设置可以加快仿真的运行时间

与硬件上运行的设计相比，HDL 仿真会比较慢。为了改进这一仿真时间，用户可以调整一些参数。例如，在串口通信的仿真中，可通过增加外围设备的波特率来减少仿真时间。

6. 帮助手册

XPS 工具可自动完成系统连接功能，以及 HDL 设计文件和后台之间的关联，且可以为所测试的设计创建仿真指令文件。当用户调用 XPS 工具栏命令"Simulation"→"Generate Simulation HDL Files"时，系统将自动地具备上述各种功能。

另外，XPS 的 Helper 脚本可以简化仿真器的使用，它生成于测试平台(testbench)级。运行时，Helper 脚本完成初始化功能，并给出产生波形和 list(只在 ModelSim 中)窗口所使用的指令。顶层脚本调用具体实例的脚本。

在"simulation\<simulation type>"目录下，用户可以看到运行仿真的几个命令脚本。system_setup.do 文件是调用其他脚本的起始点。脚本中的命令可以按照用户需要进行定制。通过编辑顶层的波形(system_wave.do)或 list 脚本，用户可以加入信号或去除信号。对于时序仿真，只给出顶层端口。

注意，Simgen 并没有为外部存储器提供仿真模型，对仿真模型也没有提供自动的支持。外部存储器模型必须实例化，且连接到仿真测试台，并根据模型规范进行初始化。

7. 基于 ModelSim 的 XPS 仿真实例

例 9-9　在 XPS 中，利用 ModelSim 对串口通信模块进行仿真，并给出所创建 IP 块对收到请求的硬件和软件响应。

1）仿真设定

对于 RS232_UART 外围设备，如果仿真速度为 9600 波特率，那么仿真时间很长，而且在很多时刻设备并没有工作。如果将仿真速率加快 100 倍，仿真时间将相应地缩短，数据转换的区域也会变小，使得用户可以更容易地评估系统的仿真行为。为了加快 UART 的波特率，用户需要：

（1）在 XPS 系统组件视图中，双击 RS232_UART 外围设备打开 Core 配置对话框。在指明 UART 波特率选项的下拉菜单中，选择最大的值 921600。

（2）在工具栏中选择“Hardware”→“Generate Netlist”生成网表，使得新的数值可以提交到用户的设计中。

2）运行仿真

（1）在工程标签下，检查“Project Options”→“Sim Model”已被设置为“BEHAVIORAL”。如果没有，双击改变这一选项。

（2）选择“Simulation”→“Generate Simulation HDL Files”运行 Simgen 工具，产生仿真 HDL 文件以及帮助脚本。在调用命令产生仿真 HDL 文件时，XPS 会生成“simulation\behavioral”目录结构。在“\behavioral”目录下可以找到两个主要的文件类型：DO 文件和 VHDL 文件。

（3）对于 system.vhd 文件，它可以以文本形式打开，此文件即为所测试器件的顶层文件，它包括组成该器件的所有信号和端口映射。对于 system_setup.do 文件，该宏文件自动生成仿真中所运行的许多步骤。其中，alias 命令调用其他的 DO 文件。用户可以添加自己的 aliases 到这个文件中以进行定制的仿真操作。打开 system_wave.do 文件，此文件显示了用户设计的信号，许多信号都由此文件生成。为便于仿真，可注释掉下列代码，保存并关闭此文件。

```
#       do iocm_wave.do
#       do docm_wave.do
#       do plb_wave.do
#       do plb2opb_wave.do
#       do ppc405_0_wave.do
#       do reset_block_wave.do
#       do isocm_bram_wave.do
#       do dsocm_bram_wave.do
#       do plb_bram_if_cntlr_1_bram_wave.do
#       do sram_256kx32_util_bus_split_0_wave.do
#       do dcm_0_wave.do
#       do jtagppc_0_wave.do
#       do iocm_cntlr_wave.do
#       do docm_cntlr_wave.do
```

```
#     do push_buttons_position_wave.do
#     do sram_256kx32_wave.do
#     do plb_bram_if_cntlr_1_wave.do
```

(4) 先前产生的 ELF 文件可以提供有关执行地址以及运行代码的汇编指令等方面的信息。在 XPS 中,选择"Project"→"Launch EDK Shell"。此命令窗口可以用来运行 EDK 的命令。

(5) 在 XPS 命令界面处,改变目录如下:

```
cd SDK_projects/TestApp_Peripheral/Debug/
```

此目录为 ELF 文件所在处。为进行分解,输入以下命令:

```
Powerpc - eabi - objdump - S TestApp_Peripheral.elf >> TestApp_Peripheral.dis
```

这一命令调用 PowerPC 目标文件,显示具有混合源和分解信息的程序(powerpc-eabi-objdump)。输出被送往 TestApp_Peripheral.dis 文件。完成这一过程后,关闭 EDK 命令窗口回到 XPS。

(6) 在 XPS 中,选择"Simulation"→"Launch HDL Simulator"。如果用户已经安装了 EDK 支持的仿真器,将出现所调用的 system_setup.do 文件。现在仿真器可以编译并导入用户的设计了。选用 ModelSim 作为仿真工具时,可在命令窗口输入以下命令:

```
c          编译设计
s          导入设计
w          建立波形窗口
rst        固定复位端口,设置时钟频率为 100MHz
run 3ms    运行仿真 3ms
```

(7) 在仿真运行过程中启动 SDK,打开"SDK_projects\TestApp_Peripheral"目录中的 test_ip_selftest.c 文件。从中可以找到处理器和用户开发 IP 之间的交互语句,其作用是告知系统读入 test_ip 的中断寄存器值,如图 9-87 所示。用户必须找到处理器和外围设备间的这一交互语句,因为处理器需要从属设备作出某些反应。

```
65   xil_printf("   - write 0x%08x to software reset register \n\r", IPIF_RESET);
66   Reg32Value = TEST_IP_mReadMIR(baseaddr);
67   if ( Reg32Value == 0x30220301 )
68   {
69      xil_printf("   - read 0x%08x (expected) from module identification regist
70      xil_printf("   - RST/MIR write/read passed\n\n\r");
71   }
72   else
73   {
74      xil_printf("   - read 0x%08x (unexpected) from module identification regi
75      xil_printf("   - RST/MIR write/read failed\n\n\r");
76      return XST_FAILURE;
77   }
```

图 9-87　交互语句示意图

(8) 用文本打开 TestApp_Peripheral.dis 文件,找到代码行"TEST_IP_mReadMIR(baseaddr)"。这里的汇编代码形式如图 9-88 所示。

```
    Reg32Value = TEST_IP_mReadMIR(baseaddr);                    | C/C++ Source Code
ffffc3d4:     7f  a3  eb  78     mr    r3,r29
ffffc3d8:     48  00  12  3d     bl    ffffd614 <XIo_In32>
    if ( Reg32Value == 0x30220301 )                             | C/C++ Source Code
ffffc3dc:     3c  00  30  22     lis   r0,12322
ffffc3e0:     60  00  03  01     ori   r0,r0,769
| Memory Address for Code Execution | Machine Code Execution Values | Assembly Operands |
```

图 9-88 汇编代码示意图

(9) 在仿真器中，查找 FFFFC3D4 上的信号值或用户设计中所用的其他数值。在 PowerPC 内部寄存器 exeaddr 信号的 FFFFC3D4 处，用户可以放大此数值区域，以定位 test_ip 外围设备的正确重置操作。放大仿真中的区域，可以看到在 sl_dbus 上有数据线转换。sl_dbus 信号在这个位置处的值应当是 0x30220301，如图 9-89 所示。

图 9-89 仿真结果示意图

通过分解的源和仿真波形输出，用户可以更好地逐步进行设计，并理解内部操作以及硬件和软件的交互关系。

9.5.3 将 EDK 设计作为 ISE 设计的子系统

单独的嵌入式系统不能充分利用 FPGA 的硬件资源，在某些场合也不能满足用户需求，更多的时候，嵌入式系统是作为 FPGA 设计中控制和调度子系统来应用的。下面介绍嵌入式子系统的设计和应用。

1. 嵌入式系统作为子模块的适用情况

在下面两种情况下，用户必须利用 ISE 将嵌入式处理器系统的 FPGA 设计作为顶层 FPGA 设计源的子模块。

首先，用户的 FPGA 设计是嵌入式处理器系统和其他定制逻辑的结合，此时要利用 ISE 开发定制的逻辑部分并实现顶层 FPGA 设计。

其次，FPGA 设计包括嵌入式处理器系统，且用户选择使用工程导航来实现。工程导航允许用户使用由 ISE 提供的其他工具，如约束编辑器。

2. 嵌入式子模块的设计方法

利用 XPS 和 ISE 工具来处理嵌入式子模块设计的方法分别称为“top-down”和“bottom-up”。这两种方法都可以使用相同的工具集和功能，其中包括 BSB。

1）自顶向下的设计方法

在ISE工程中，可以像调用DCM、加法器等模块的IP Core一样，来调用EDK设计(MicroBlaze/PowerPC)。在新建工程时，可直接添加Embedded Processor类型的源代码，该选项在安装EDK软件之后才会出现。它与其他子模块是一样的，可在任何需要的地方例化。有关具体操作过程，将在后面通过实例来说明。在使用时需要注意的是，必须把EDK设计的C代码编译后的可执行文件初始化到FPGA的块RAM中。

在这种方法中，XPS自动获得ISE中所选的FPGA器件，创建嵌入式子模块(XMP文件)，并将其作为顶层FPGA设计的源文件。用户在ISE工程中只能添加一个嵌入式子模块，但是该嵌入式子模块源可以包括多个微处理器。下面通过一个点亮LED小灯的工程实例来介绍自顶向下的嵌入式子模块设计方法。

例9-10 利用自顶向下的设计方法，将一个嵌入式子模块添加到ISE工程中，要求该嵌入式系统具备点亮3个LED灯的功能。

下面的步骤详细给出了在ISE设计中创建一个嵌入式处理器子系统，并在XPS中进行设计，最终在FPGA中运行嵌入式子系统。

(1) 在工程中新建“Embedded Processor”类型的源代码模板，如图9-90所示。

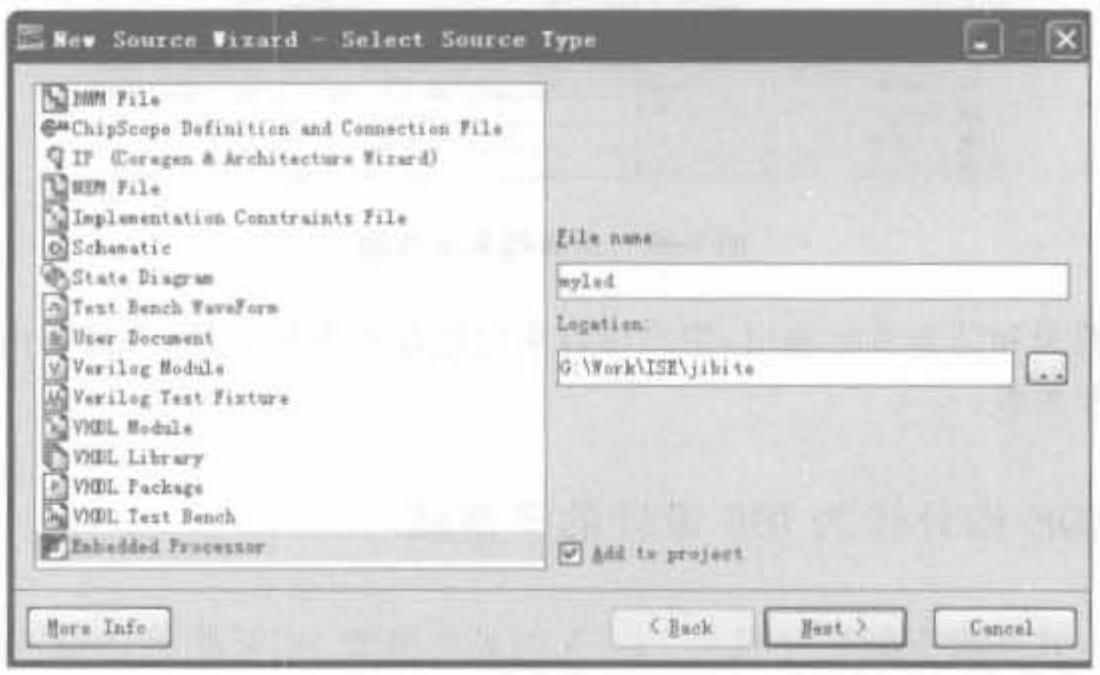


图9-90 Embedded Processor源代码使用模板

(2) 系统自动调用EDK，按照上一节的步骤添加GPIO，设置都是一样的。接着生成BSP、库，编写C代码，编译后生成网表。

(3) 在ISE下单击“View HDL Instantiation Template”，查看例化模板。

```
component cpu is
port (
sys_clk_pin: in std_logic;
sys_rst_pin: in std_logic;
fpga_0_LEDS_GPIO_d_out_pin: out std_logic_vector(0 to 2)
);
end component;
```

注意，CPU默认生成的vector都是(0 to X)，跟我们写的vector(X downto 0)直接连接

的话，是对应位连接的，即<2>-<2>、<1>-<1>、<0>-<0>。

在 ISE 中通过 XST 来查看其 RTL 级结构，如图 9-91 所示。

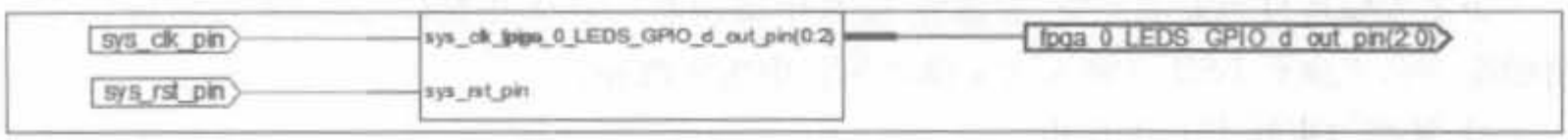


图 9-91　Embedded Processor 处理器的 RTL 结构示意图

(4) 接下来就是在 ISE 中实现、下载之类的工作了，不再赘述。我们修改 EDK 的 C 代码并且编译通过后，只需要在 ISE 里单击“Update Bitstream with Processor Data”命令即可，因为只更新软件比特流，所以速度是非常快的。也就是说，在硬件模块定下之后，只需要修改 C 代码，这样开发非常方便。

在使用 ISE 低版本时(ISE 8.2 以下)，每次更新比特流之前必须单击 XPS 中“Software”菜单下的“Clear Bit”命令，清除比特文件；否则，ISE 不会把 C/C++ 的可执行代码初始化到 FPGA 的块 RAM 中，导致 CPU 不能正常工作。

注意，在实例化用户顶层源文件的子模块后，用户的 XPS 子模块将出现在层次树的 ISE 顶层 HDL 源下。如果用户想修改嵌入式子模块设计中的任何部分，可以运行“Manage Processor Design”，或双击重新打开 XPS 以及导入的嵌入式工程。

2) 自下向上的设计方法

在这一方法中，用户必须选择和 XPS 以及 ISE 中相同的 FPGA 结构。为了实现自下向上的开发，用户首先调用 XPS 并开发嵌入式处理器设计，接着调用 ISE 并添加嵌入式子模块作为包括在用户顶层 ISE 工程中的源。其中，只有 XMP 文件才可以添加为源文件，块存储器映射(Block Memory Map，BMM)文件不能添加到 ISE 工程中。

如果用户利用 XPS 已经创建了一个嵌入式子模块设计，那么就可以执行如下步骤添加嵌入式子模块到顶层设计中：

(1) 打开 ISE 工程导航。

(2) 为用户顶层 FPGA 设计创建或打开 ISE 工程。

(3) 为用户 ISE 工程选择 FPGA 器件族、封装和速率等级，这些属性和嵌入式子模块中为 ISE 工程指定的属性相同。

(4) 选择“Project”→“Add Source”打开添加源的窗口。

(5) 在“Add Existing Sources”对话框中，为嵌入式子模块选择 XMP 格式的 XPS 工程文件。用户的 XPS 子模块将在工程面板中源的位置出现。

注意，在用户实例化顶层源中的子模块后，XPS 子模块将出现在层次树里的 ISE 顶层源目录下。用户可以选择 XMP 源并运行“View HDL Instantiation Template”来产生 HDL 模板样例。模板中的 HDL 部分可以复制到用户顶层 HDL 源文件中。

3. 嵌入式子模块的常用操作

1) 在顶层设计中加入嵌入式子模块

在用户的顶层 FPGA 设计中，用户必须实例化并连接嵌入式处理器系统。同时，用户必须复制(有时可以进行修改)任何由 XPS 生成的设计约束到 ISE 工程的 UCF 中。

2) 连接嵌入式子模块

用户可以连接嵌入式子模块的输出端口到顶层设计的输出端口。对于嵌入式设计的输

入端口，可以利用顶层设计的输入端口和其他逻辑来驱动。嵌入式子模块的组成端口和MHS文件中的外部端口可以进行一对一的通信。

为了方便端口约束的复制，如果在BSB中指定某一具体开发板，用户在顶层设计中使用的端口名字应和BSB在嵌入式子模块元件中生成的相同。

3）复制约束到ISE工程中

在XPS中运行BSB时，就会在XPS工程的data子文件夹中产生一UCF，即<projectname>.ucf。此UCF包括了一些基本时序约束，给出用户所选处理器的参考时钟频率。如果在BSB中用户选择某一特定的开发板，UCF还包括了板上外围设备的完整端口规范。UCF也可能包括某些端口的I/O约束，如IOSTANDARD。

（1）复制BSB生成的约束到已有的UCF中

如果用户已将UCF源文件添加到顶层ISE工程中，那么就可以将BSB生成的约束复制到其中。对于BSB生成约束中涉及的任何嵌入式子模块端口，如果连接到不同名称的顶层端口上，那么用户就必须相应地对约束里的net名称进行编辑。

另外，用户也可以使用ISE中的约束编辑器，从BSB生成的UCF中导入端口约束到ISE工程约束文件中。但是，只有在BSB生成约束中涉及的所有嵌入式子模块端口都连接到有相同名称的顶层端口时，才可使用这一方法。

（2）在顶层设计中使用BSB生成的UCF

如果用户的顶层设计并没有UCF，可以添加BSB生成的UCF副本来作为起始点。

（3）实现包括嵌入式子模块的FPGA设计

在利用XPS进入嵌入式硬件平台设计，以及利用工程导航进入顶层FPGA设计和相应的UCF后，用户就可以开始实现完整的FPGA硬件设计。

4. 在XPS中完成嵌入式子模块设计

前面介绍了如何在ISE中完成嵌入式子模块的设计。同样，在XPS中也可以完成嵌入式系统设计。当然，基于XPS的嵌入式系统设计方法属于自下向上的设计方法。

（1）打开XPS，选定芯片和硬件板，新建工程。

（2）选择“Project”菜单中的“Project Options”选项，选择“Hierarchy and Flow”页面，则弹出如图9-92所示的配置界面。

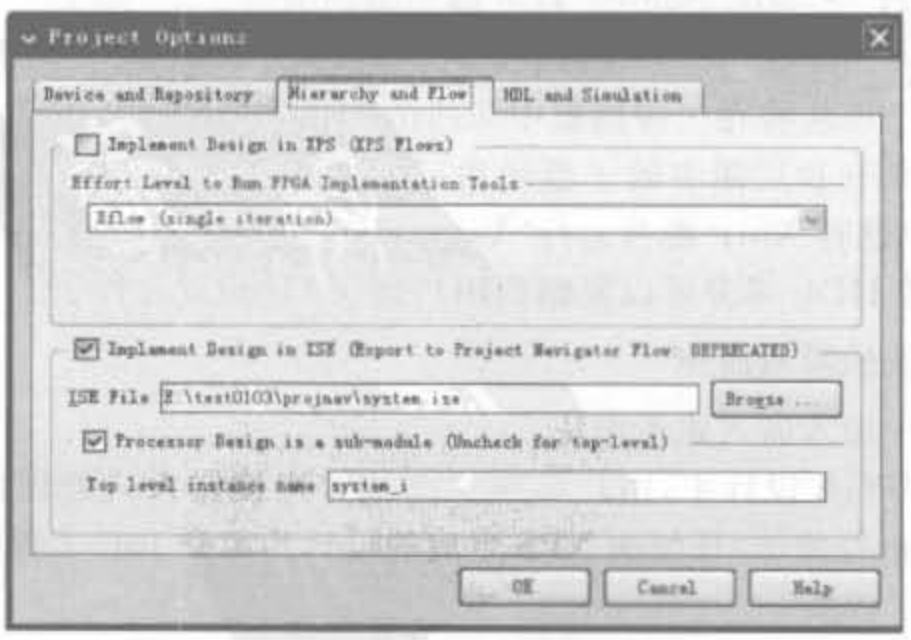


图9-92 XPS中系统层次设定界面

其默认选择为“Implement Design in XPS”，即在 XPS 中完成嵌入式系统的开发。直接选中“Implement Design in ISE(Export to Project Navigator Flow: DEPRECATED)”选项，在“ISE File”输入 ISE 工程的目录，也可以通过“Browse”按钮来选择相应的路径；选中“Processor Design is a sub-module(Uncheck for top-level)”选项，并在“Top level instance name”后的文本框中输入在高层例化的名字，即可自动将 XPS 作为高层模块的子系统。

9.5.4 XPS 对嵌入式操作系统的支持

嵌入式操作系统和嵌入式设计的应用密切相关，目前主要分为两类：一类是专门为嵌入式系统设计开发的实时操作系统，如 VxWorks、Palm OS 以及 uC/OS 等；另一类是由传统的操作系统根据嵌入式系统的特点简化得到的，如 WinCE 和各种嵌入式 Linux 系统。本节以 uCLinux 为例进行介绍。

要基于 XPS 完成嵌入式操作系统，开发 PC 要具备 Linux 开发环境，因此读者应首先建立自己的 uCLinux 交叉编译环境，将交叉编译器和内核源代码放到指令目录下，再添加到系统路径中即可。本节只介绍 XPS 对 uCLinux 的支持，至于 PC 端的配置和操作，读者可参见 Linux 方面的相关书籍。下面给出基于 uCLinux 的 MicroBlaze 系统配置的实例，提供一个可运行 uCLinux 的硬件平台。

例 9-11 在 XPS 中建立一个可运行 uCLinux 的 MicroBlaze 平台。

(1) 按照前文介绍的方法添加 BSP，因为操作系统需要 Flash 和内存模块，前者充当“硬盘”，后者主要用于运行程序，因此要根据实际情况添加相应的 Flash 控制器和内存控制器。

(2) 此外，还需要配置软件平台。单击“Software”菜单中的“Software Platform Seeting”命令，其余保持默认，在“Os and Libraries”配置项的下拉框中选择“uCLinux”即可。

(3) 单击“OK”按钮，退出界面。至此，基于 uCLinux 的基本平台就生成了。

9.5.5 XPS 工程的实现和下载

上面已经介绍了系统设计的相关软件操作，本节在此基础上介绍设计过程中的后续硬件操作步骤。实现和下载总共分成 4 步，首先，产生系统的硬件网表，即对嵌入式系统进行综合；其次，产生系统硬件架构的比特流文件；再次，编译软件，并将软件代码生成的比特流和系统硬件的比特流合二为一，构成完整的系统描述比特流；最后，将完整的比特流下载到 FPGA 芯片中。

1. 生成系统硬件网表

XPS 的硬件平台生成(Platgen)程序可以读入 MHS 文件中的硬件平台信息，以及 MPD 文件中的 IP 特征设置。Platgen 中的输出文件为 HDL 文件，可在“<project name>\hdl\<hdl lang>”中找到。XPS 会自动调用 XST 综合工具将所有的 HDL 设计文件转化成 IP 网表(NGC)文件，其原理如图 9-93 所示。

选择“Hardware”→“Generate Netlist”菜单项，这一命令能够完成对硬件平台的综合，如图 9-94 所示。

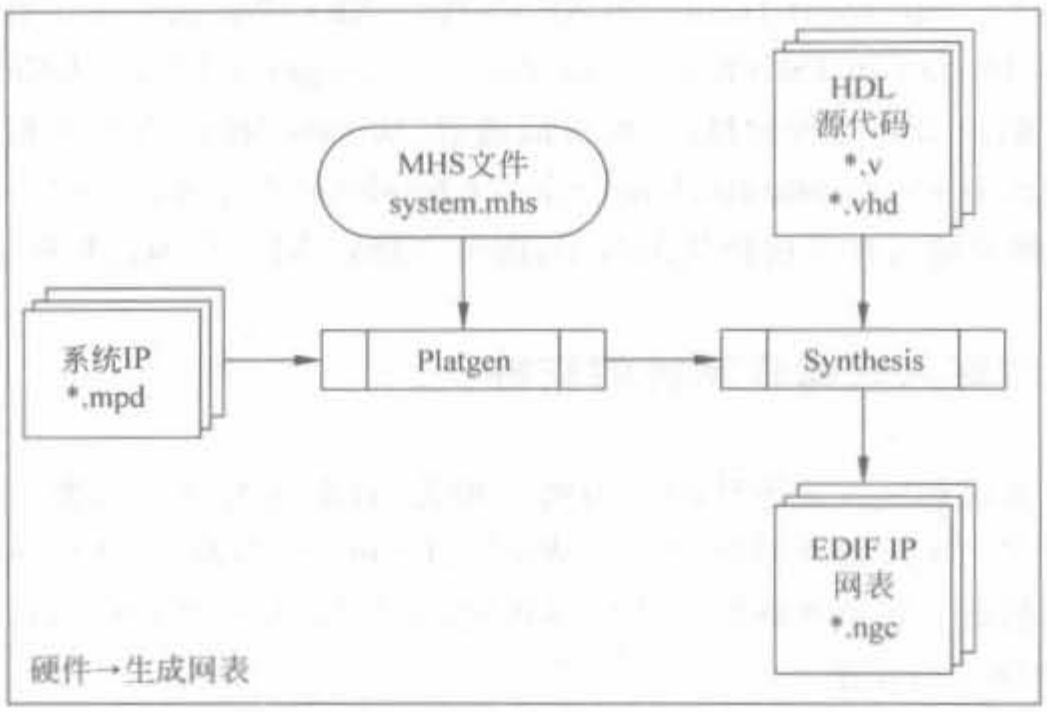


图 9-93　生成硬件网表的元素和过程

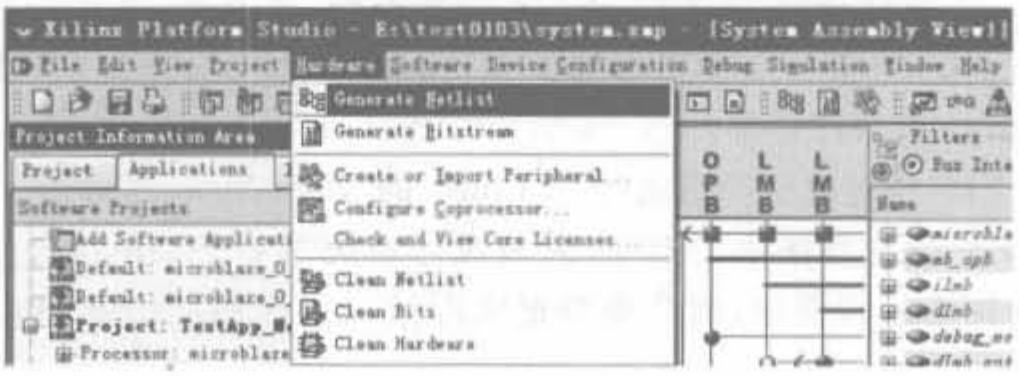


图 9-94　生成硬件网表命令的菜单示意图

单击“Generate Netlist”后，XPS 即调用 XST 综合，可在控制台窗口中看到相应的综合信息。当综合完毕后，会在控制台窗口中给出综合整体报告，如操作、警告和输出信息的个数，如图 9-95 所示。

图 9-95　系统硬件综合后的输出信息

2. 生成系统硬件网表

生成网表之后，就是在此基础上生成硬件比特流文件，即完成翻译、映射和布局布线。在 XPS 中，通过调用其他的 ISE 工具（NGDBuild、MAP、PAR 和 TRACE）对 NGC 文件和

系统约束文件进行处理。XFlow 读入输入设计文件、流文件以及可选文件来产生 FPGA 比特流。通常，用户不需要改变流或输入设计文件，但要根据实际硬件配置修改约束文件。其中，这些约束可能为简单的时钟信息或端口位置约束，也可能为复杂的布局、时序参数等约束。整个比特流生成过程如图 9-96 所示。

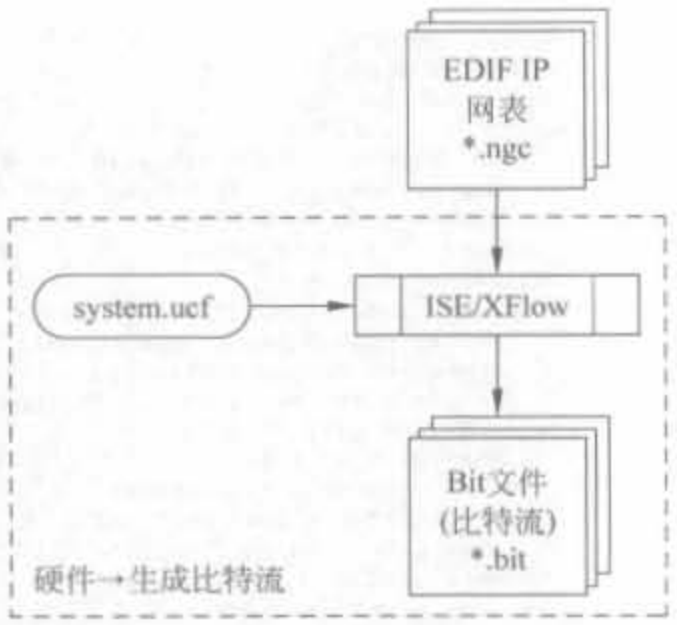


图 9-96　生成硬件比特流的元素和过程

其中，NGC 文件位于“<project name>\hdl\implementation”文件夹中。选择“Hardware”→“Generate Bitstream”菜单命令时，可启动实现的过程，将 NGC 网表文件转化为比特流文件，如图 9-97 所示。

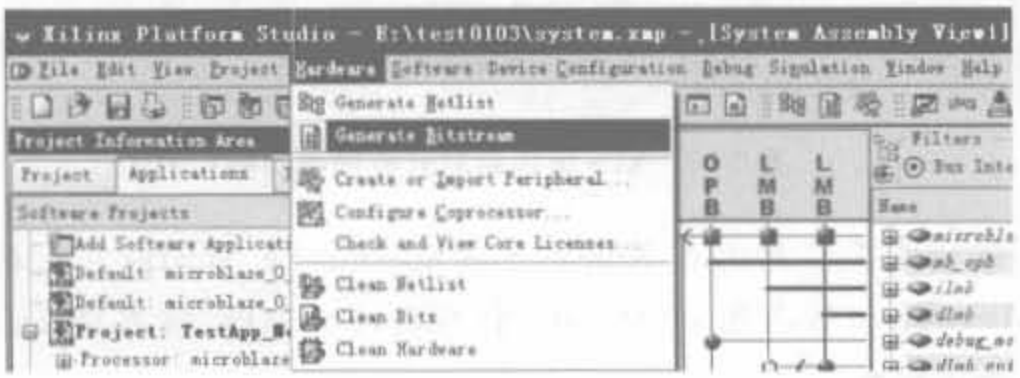


图 9-97　生成硬件比特流命令的菜单示意图

在比特流的生成阶段，以处理器为中心的设计以及所有的逻辑都和此嵌入式处理器系统相关联，因此在比特流生成前需要添加相应的约束文件。在运行 BSB 向导时，选择 Xilinx 开发板，则其约束都已预先设定。在完成 BSB 向导步骤后，会自动生成所有的这些约束，位于“<project name>\data”目录下。执行生成比特流命令后，会在控制台输出窗口列出相应的信息指示，和 ISE 中实现的过程的输出信息一致。当比特流生成后，再给出图 9-98 所示的指示信息。

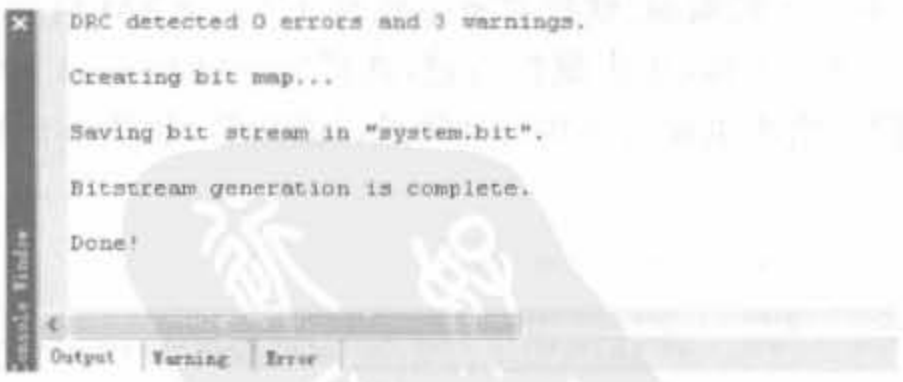


图 9-98　比特流生成的指示信息

当系统硬件比特流生成后，即可查看它所占全部硬件资源以及布局布线后的时序分析结果。后续的软件开发不再增加额外的硬件资源，也不会影响已有硬件的各种性能。在 XPS 中将视图切换到“Project”目录，在“Reference File”目录下的“Log File”的“Implementation/Xflow.log”中，可查看所有硬件实现结果。其硬件资源分析报告的模板如图 9-99 所示。

```
Design Summary:
Number of errors:      0
Number of warnings:    6
Logic Utilization:
  Total Number Slice Registers:     4,117 out of   9,312   44%
    Number used as Flip Flops:                  4,116
    Number used as Latches:                         1
  Number of 4 input LUTs:           4,467 out of   9,312   47%
Logic Distribution:
  Number of occupied Slices:                          4,382 out of   4,656   94%
    Number of Slices containing only related logic:   4,382 out of   4,382  100%
    Number of Slices containing unrelated logic:          0 out of   4,382    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:          5,767 out of   9,312   61%
  Number used as logic:                4,467
  Number used as a route-thru:           332
  Number used for Dual Port RAMs:        466
    (Two LUTs used per Dual Port RAM)
  Number used as Shift registers:        502
  Number of bonded IOBs:                 118 out of     232   50%
    IOB Flip Flops:                      154
  Number of ODDR2s used:                  22
    Number of DDR_ALIGNMENT = NONE        22
  Number of Block RAMs:                    8 out of      20   40%
  Number of GCLKs:                         9 out of      24   37%
  Number of DCMs:                          3 out of       4   75%
  Number of BSCANs:                        1 out of       1  100%
  Number of MULT18X18SIOs:                 3 out of      20   15%
```

图 9-99　XPS 硬件资源分析报告模板

图 9-99 给出的资源报告对应着 Xilinx 公司 Spartan-3E Starter 开发板，包含了很多外部资源。其中，MicroBlaze 核占用 500～600 个 Slice，PowerPC 核本身不占用额外的硬件。

3. 编译软件应用程序

XPS 中的软件代码一般都是基于 C 语言的，入口函数就是 main( )函数，当用户代码书写完毕后，调用 mb-gcc 编译器将其转化为可执行文件。如果第一次编译，会首先编译 XPS 提供的库函数；如果系统软件架构有误，则不能通过编译，需要对照 MSS 文件进行修改。

编译之前，首先要生成连接脚本，设定软件代码段、程序启动地址以及存储空间，具体方法如 9.5.1 节所述。其中，根据系统存储器资源选择各个程序段的存放地址，一般选择外部存储芯片用于存放可执行程序；如果没有外接 SDRAM 或 SRAM 等外存，只能选择片内 RAM，但对程序大小有一定的限制，软件程序必须通过片内 RAM 启动。

其次，编译工程。选中工程，单击鼠标右键，选择“Build Project”命令即可。编译完成后，会在控制台输出窗口给出生成的可执行文件的详细信息，如图 9-100 所示。

```
mb-size TestApp_Peripheral/executable.elf

   text    data     bss     dec     hex filename
  27732    1877   22879   52488    cd08 TestApp_Peripheral/executable.elf


Done!
```

Console Window　Output | Warning | Error

图 9-100　工程编译输出信息

4. 配置 FPGA 芯片

要在 FPGA 中实现一个嵌入式处理器系统，必须将硬件和软件系统部分都下载到 FPGA 和程序存储器中。XPS 提供了完整的配置方案，如图 9-101 所示。

在芯片配置阶段，用户可以通过 JTAG 电缆将主机和开发板的 JTAG 口相连，下载硬件比特流和软件 ELF 文件镜像。其中，软件程序需要整合到硬件比特流中，一起下载到 FPGA 芯片中。

(1) 需要将期望下载的目标工程设定为片内 RAM 的初始化程序。选中工程，单击鼠标右键，选中 "Mark to Initialize BRAMs"，如图 9-102 所示，并编译目标工程。

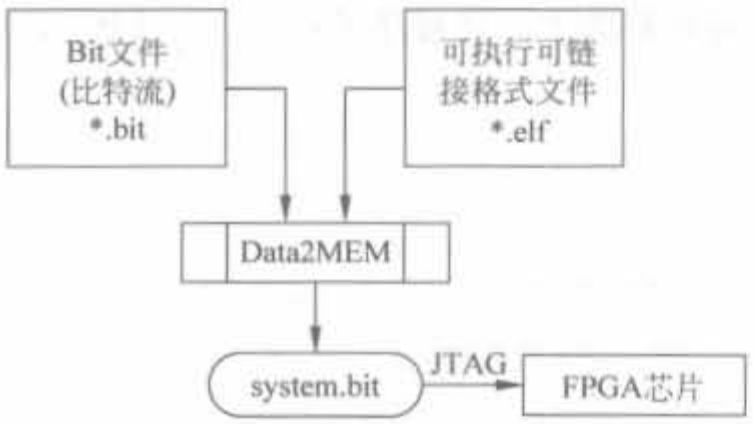


图 9-101　生成嵌入式系统比特流的元素和过程

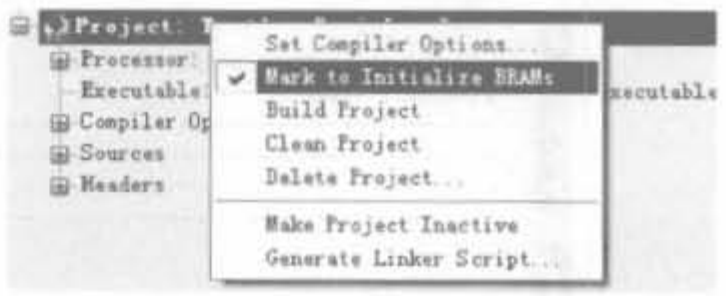


图 9-102　设定 BRAM 初始化程序

(2) 选中"Device Configuration"→"Update Bitstream"命令，将编译所生成的可执行文件和硬件比特流合在一起，如图 9-103 所示。

图 9-103　更新配置比特流

此后，对软件代码再进行修改，只需要重新编译并更新即可，不需要重新生成原有的硬件比特流文件。这一特性极大地节约了系统开发的时间。更新成功后，控制台窗口仍会给出完成指示信息，如图 9-104 所示。

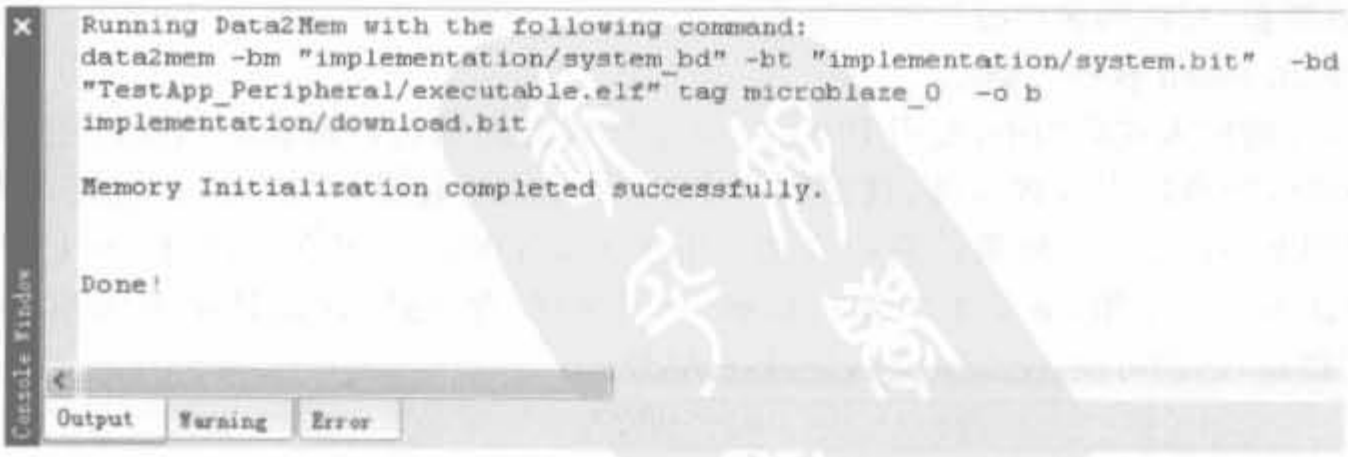


图 9-104　比特流更新成功指示信息

(3) 选择“Device Configuration”→“Download Bitstream”菜单命令，对 FPGA 芯片进行编程，如图 9-105 所示。

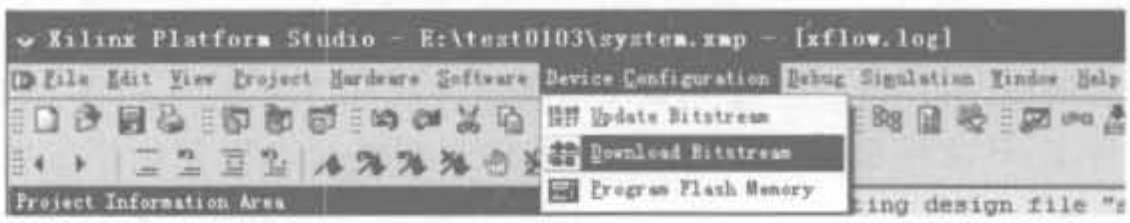


图 9-105 FPGA 配置命令示意图

执行了配置命令后，XPS 会调用 iMPACT 程序完成边界扫描和下载。需要注意的是，该命令只是将程序下载到 FPGA 芯片中，断电后程序即丢失。将程序成功下载到 FPGA 以后，控制台给出的指示信息如图 9-106 所示。

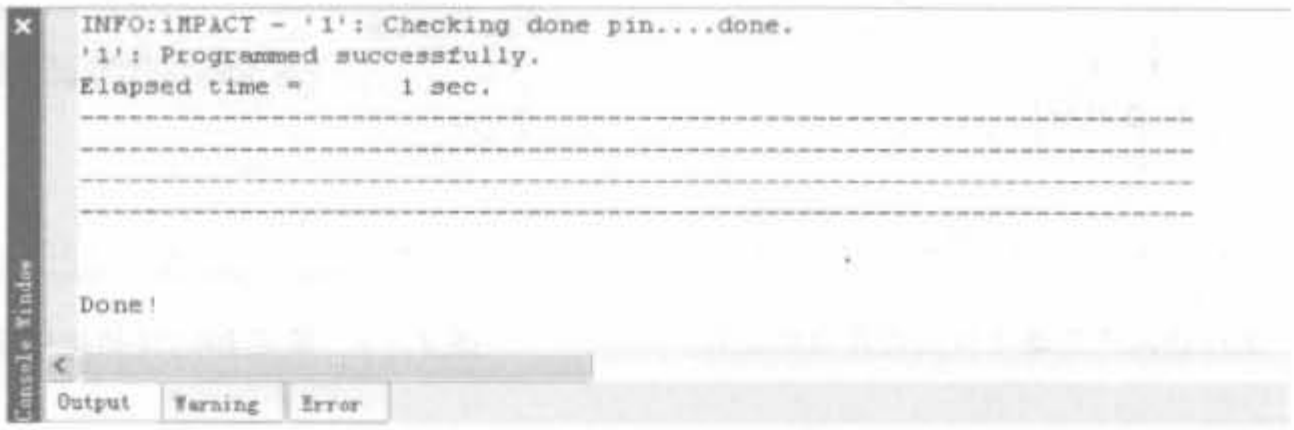


图 9-106 配置成功的指示信息

5. 固化嵌入式系统设计

要将 XPS 设计永久性地固化在 FPGA 系统中，即当 FPGA 上电后自动从 PROM 或 Flash 芯片中加载程序，有两个方法，即将系统完整的比特流转化成 PROM 配置文件；或在嵌入式系统中添加 Flash 控制器，直接配置 Flash 存储器。

1) 将. bit 文件转化成 PROM 配置文件

该操作利用 ISE 中的 iMPACT 组件，将. bit 文件转化成配置 PROM 的配置文件，再由 iMPACT 来配置到 PROM 中，详细操作可参考 5.4 节。该方法非常简单，可快速将嵌入式系统的软、硬件设计文件下载到 PROM 中。当然，其前提是硬件系统提供了 PROM 器件。

此外，还可将一定的用户数据存放在 PROM 中，在系统加载完成后，继续控制 PROM 读取所需数据，相应的硬件设计可参考 5.5 节。

2) 添加 Flash 软核控制器

在实际的嵌入式应用中，复用 PROM(包括处理器的加载以及相应软件应用数据的加载)并不适合所有的嵌入式系统，这是因为 PROM 的价格较高，对于大、中规模不太适用。因此，使用 Flash 也是一种很普遍的方法。由于 Flash 配置方式众多，包括 SPI、BPI 等方式，下面就不一一介绍，最基本的就是根据 Flash 的配置电路，在设计中添加相应类型的 Flash 控制器，达到可使用软核访问 Flash 的目的。

9.5.6 在线调试工具 XMD 的使用

在 EDK 开发环境中，嵌入式系统元素都位于 FPGA 内部，因此调试非常困难。为此，

Xilinx 提供了多种方法和工具使得用户可以方便地查看所设计的软件和硬件部分。例如，利用 Xilinx 微处理调试器(Xilinx Microprocessor Debugger，XMD)的硬件调试功能；平台工作室 SDK 软件调试器可以通过 XMD 接口与目标处理器通信；利用综合的逻辑分析仪硬件核，ChipScope Pro 工具可以与 Xilinx 器件内绝大部分的目标设计通信。本节将介绍一些 Xilinx 调试功能的关键部分。

1. XMD 调试工具

XMD 方便了用户所创建软件项目的调试，同时帮助用户验证系统。用户可以利用 XMD 来调试在硬件板子上运行的程序，或者是使用周期精确指令设置仿真器(Cycle-Accurate Instruction Set Simulator，ISS)的程序。图 9-107 和图 9-108 分别给出了 XMD 与 PowerPC 硬核处理器和 MicroBlaze 软核处理器的连接结构以及和调试软件的交互方式。

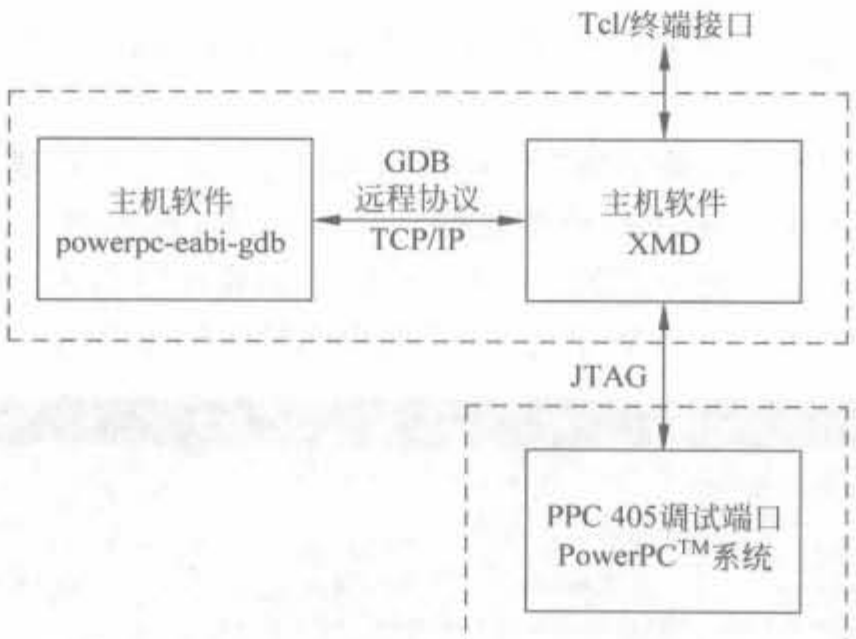


图 9-107　XMD 与 PowerPC 的系统连接示意图

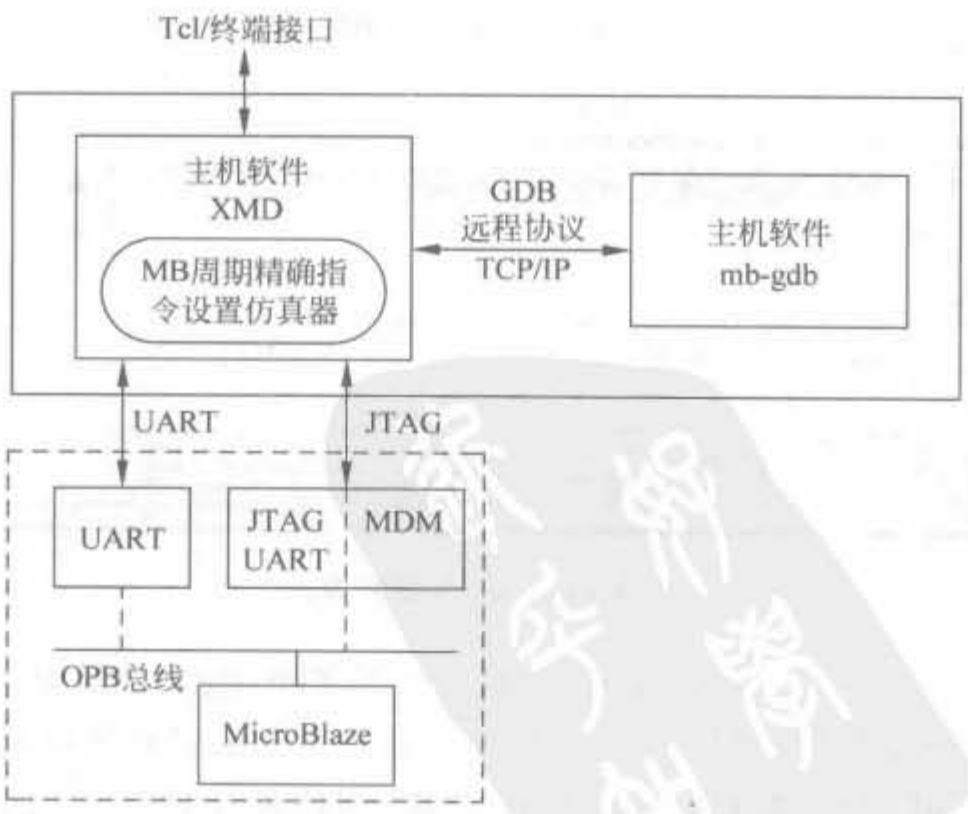


图 9-108　XMD MicroBlaze 系统连接

注意，XMD 并不是独立的，而是与其他功能实体同时工作的，比如 XPS GUI。通常，XMD 通过 JTAG 与目标处理器相连，通过 TCP/IP 协议完成通信和控制。在图 9-107 和图 9-108 中，基于所选微处理器的类型(PowerPC 或 MicroBlaze)以及用户配置系统的方法，XMD 将所测试设备的状态信息传送给 GUI(SDK)。基于用户在 SDK 中输入的请求，XMD 同时还控制处理器的操作。下面给出基于 MicroBlaze 软核的 XMD 调试实例。

例 9-12 利用 XMD 调试基于 MicroBlaze 的用户应用代码。

本例以 Xilinx 公司的 Spartan-3E starter 开发板为例，详细介绍 XPS 自动生成的外设测试工程的 XMD 调试方法。

(1) 新建工程，选择 Spartan-3E starter 开发板。

(2) 连接电源和 USB 连线，其 USB 连线作为 USB 下载线使用，可通过 iMPACT 软件来测试其连通性。

(3) 边界扫描通过后，将应用中的“TestApp_Peripheral”工程编译、更新并下载到 FPGA 中。

(4) 选择 XPS 中“Debug”菜单的“XMD Debug Options”选项，弹出如图 9-109 所示的配置对话框。在“JTAG Cable”下拉框中选择“USB”选项，右边的“Frequency”选项会自动调整为“750000”。如果是其他类型的 JTAG 下载线，需要在“JTAG Cable”类型中选择相应的类型，或者直接选择“Auto”，然后单击“OK”退出 XMD 配置界面。

图 9-109 XMD 配置界面

(5) 单击“Debug”菜单中的“Launch XMD”命令，XPS 会启动 GDB 服务，并自动连接到硬件板。默认的连接 ID 号为“0”，TCP 端口为“1234”。如果没有硬件错误，XPS 会弹出如图 9-110 所示的 XMD 命令窗口。如果连接失败，则该命令窗口不会显示“Connected to “mb” target. id=0. Starting GDB server for “mb” target <id=0> at TCP port no 1234”的指示信息。

图 9-110　XMD 后台启动的命令窗口示意图

（6）单击“Debug”菜单中的“Launch Software Debugger”命令，弹出如图 9-111 所示的工程选择对话框。

图 9-111　工程选择对话框

在下拉框中选择相应的工程，单击“OK”按钮，XPS 会弹出用于调试的“Source Window”对话框，如图 9-112 所示。

图 9-112　软件调试对话框

单击图 9-112 的 图标，即可弹出与目标板的连接对话框，如图 9-113 所示。直接单击“OK”即可进入调试模式。

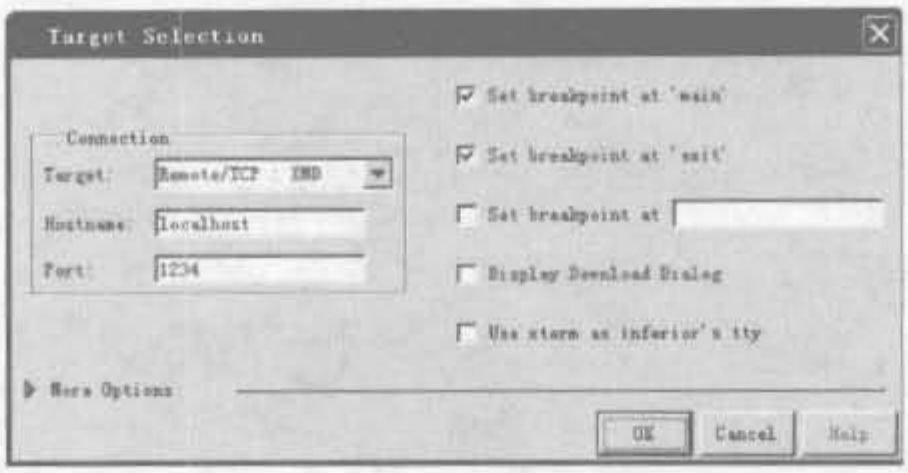
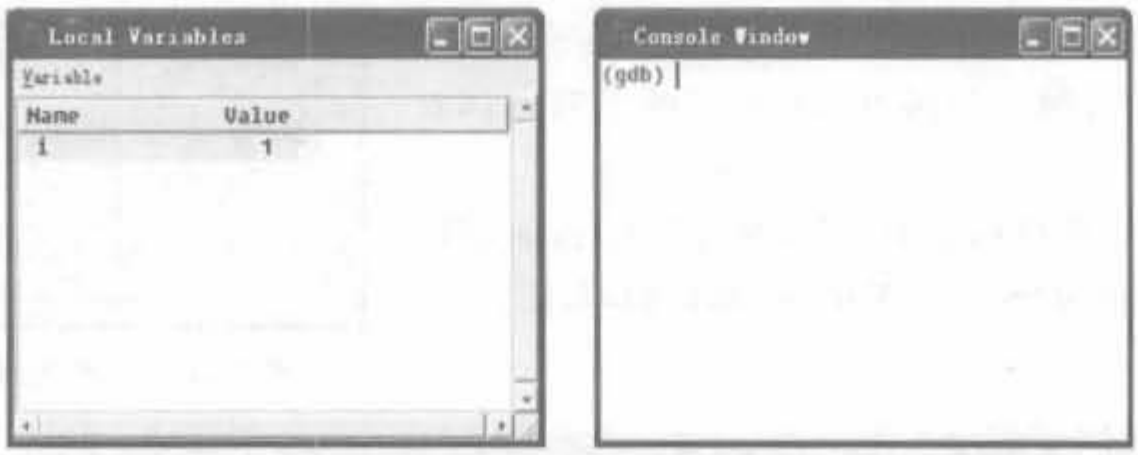


图 9-113　连接到目标板的配置对话框

然后，通过单击 图标来完成单步调试，单击 图标查看本地变量，单击 图标打开控制台，可以用于输出信息。本地变量查看器和控制台的界面分别如图 9-114(a)、(b)所示。



(a) 本地变量查看器的界面　　(b) 控制台的界面

图 9-114　软件调试工具界面

2. SDK 软件调试器

平台工作室 SDK 为无缝调试嵌入式设计提供一个综合的环境。MicroBlaze 和 PowerPC 的 ELF 文件都可以用 SDK 进行调试。通过控制软件调试器(比如 SDK 提供的)的开始、结束和中断(设置断点)操作，用户可以监控程序的执行过程。通过对存储器和/或可变变量的监测和调整功能，软件调试器也可以对程序的操作进行一些运行时间的控制。SDK 软件调试器的详细使用将在 9.5.8 节介绍。

3. ChipScope Pro 工具

ChipScope Pro 工具包括多个功能块，它们都被综合在同一个应用程序中。ChipScope Pro 分析仪为 ChipScope Pro 核提供了器件的配置、触发建立以及跟踪等功能，通过对总线和任意信号值的监测，以及利用 JTAG 连接对输入和输出的不连续控制，实现对 ChipScope Pro 核硬件的调试。ChipScope Pro 在 EDK 中调用的详细方法将在 9.5.7 节介绍。

4. 平台调试

如果用户单独使用上述调试功能实体，可以获得很大的帮助。但如果用户进一步地结合使用这些实体，那么这些工具可以提供更大的性能提高，即它们可以给用户提供嵌入式设

计中有关软、硬件交互的完整描述。对于隔离程序错误源头，这一能力是至关重要的。

1）概述

XPS中的平台工作室调试配置向导可以自动化软、硬件调试配置的工作。选择“Debug”→“Debug Configuration”，打开调试配置向导，如图9-115所示，其中有以下几个主要的视图。

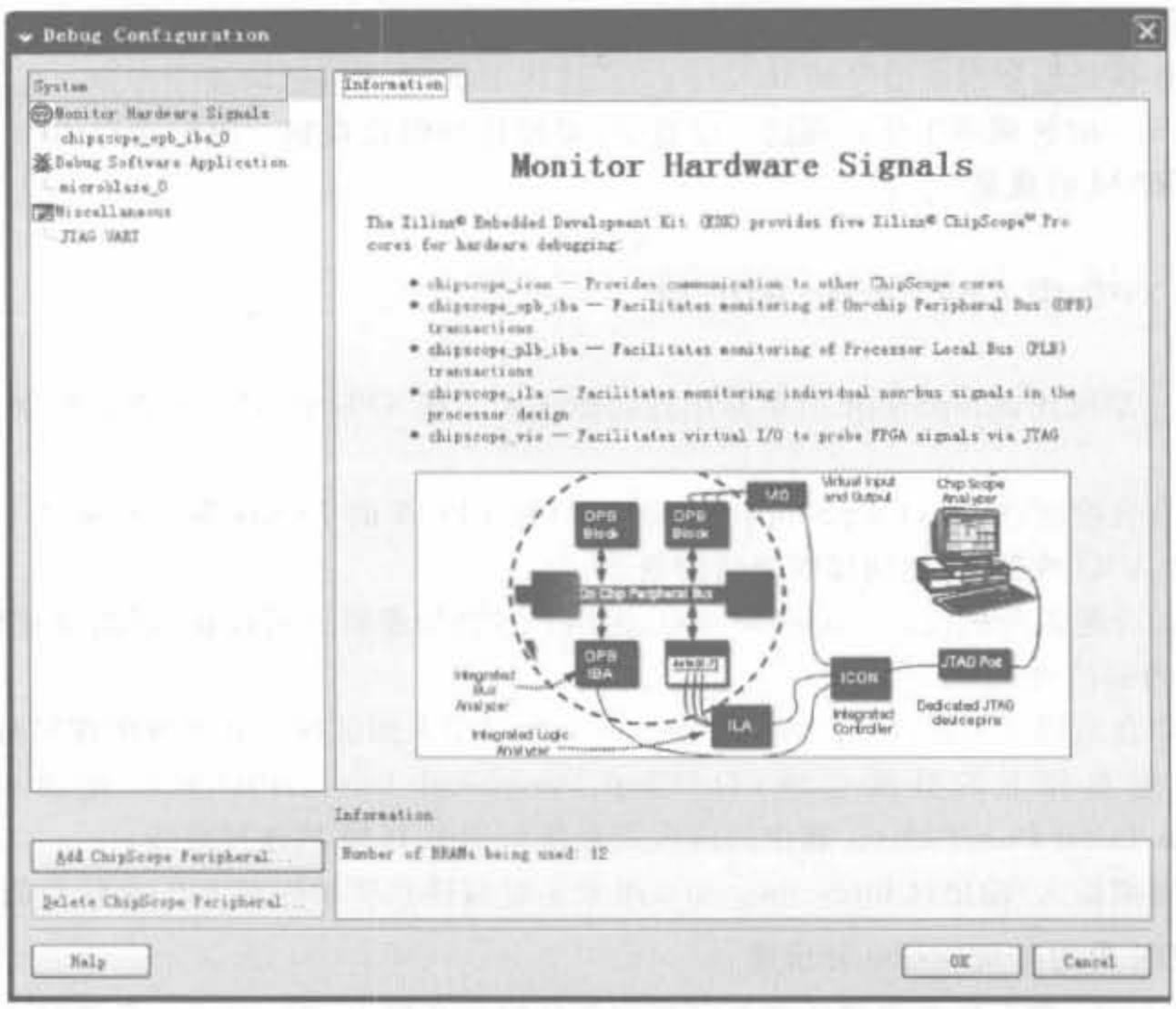


图9-115　调试配置界面示意图

（1）系统管理器

向导主窗口左边的选项允许用户选择不同的调试实体。这些选项可以为ChipScope核和处理器配置调试属性。

（2）配置面板

配置面板包括所需操作以及所选核设置的信息。用户可以在基本或高级调试控制属性中进行选择。

（3）控制窗口

控制窗口显示输出、警告、错误，以及来自调试配置向导的信息。

2）硬件和软件同时调试

调试配置向导方便了软、硬件的同时调试，它可以实现：

（1）连接IBA trig_out到处理器的停止信号，这样IBA就可以将处理器置于调试中断模式。简言之，ChipScope信号终止处理器的运行。

当处理器被终止时，软件调试器寄存处理器停止时的状态，并允许硬件触发器与软件中的活动相关联。处理器停止和调试器中对这一事件进行记录之间的时间差可以短达11个

时钟周期，这取决于所用的总线。因此，软件终止很有可能发生在引起硬件触发事件的同一个子程序内。

(2) 连接处理器终止信号到 IBA trig_in，这样处理器的终止就会触发 IBA 记录采样值。

调试事件，如断点事件，会强迫处理器停止它的运行过程。当这一事件发生时，总线分析仪会存储这一情况并给出这一时间点前的所有采样，这样用户就可以将软件事件和硬件状态相关联。

(3) 连接处理器指令信号到 IBA trig_in，这样 IBA 就可以记录指令序列。处理器和时钟必须在同一时钟频率工作。在这一设置下，可以进行跟踪观察。跟踪缓存的深度受限于芯片上 BRAM 的数量。

9.5.7 XPS 中 ChipScope 的使用

如第 7 章所述，ChipScope 简单易用且功能强大。在 XPS 中，可用的逻辑控制和分析核包括：

(1) 综合控制器 Pro(ChipScope_icon)，为目标 FPGA 的 JTAG 端口和多达 15 个的其他核(IBA、VIO 或 IBA)之间提供通信的通道。

(2) 综合逻辑分析仪(ChipScope_ila)，由用户定制的逻辑分析仪核，可以监控用户设计中的任何内部信号。

(3) 综合总线分析仪(ChipScope_iba)，一个专门用来调试嵌入式系统的逻辑分析仪核，它可以互连芯片上的外围总线(On-Chip Peripheral Bus，OPB)或处理器本地总线(Processor Local Bus，PLB)。其中的嵌入式系统包括了 IBM 核连接总线。

(4) 虚拟输入/输出(ChipScope_vio)，用来实时监控和驱动内部 FPGA 信号的核。

1. EDK 中的 ChipScope 分析核

在 EDK 中，所有的逻辑模块都必须添加到处理的总线上，ChipScope 分析核也不例外，全部外挂在 OPB/PLB 总线上。

1) ChipScope_icon

ChipScope_icon 本身并不能监控任何信号，但提供了统一的接口来连接 JTAG 接口和各个监控核，处于枢纽的地位，如图 9-116 所示。它和 ISE 中 ChipScope 的 ICON 核功能是一样的，最多可管理 15 个监控核。

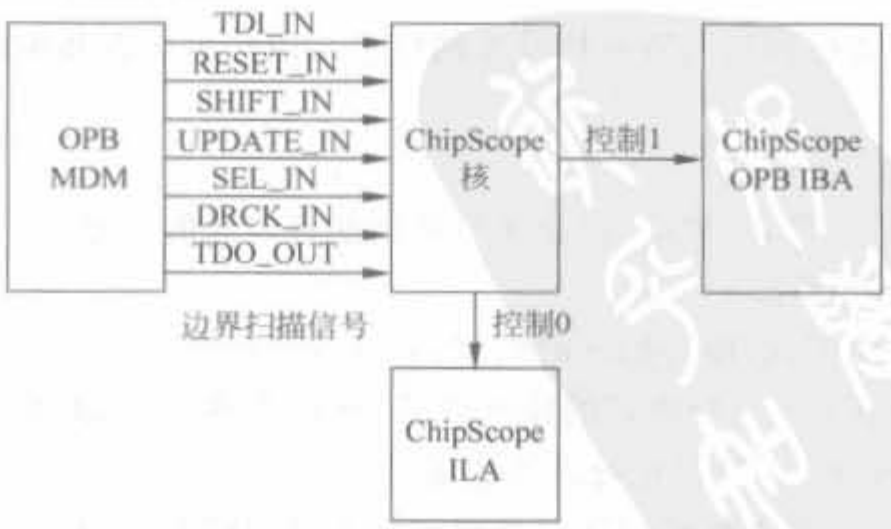


图 9-116 EDK 中 ChipScope 核的组成架构

ChipScope_icon 核的主要参数如表 9-11 所示。

表 9-11 ChipScope_icon 核参数列表

特征描述	参数名	允许的数值范围	默认值	VHDL 类型
连接到 ICON 核的 ChipScope 核的数目	C_NUM_CONTROL_PORTS	整数 1～15	1	Integer
系统是否包含外设 OPB_MDM，以便决定要不要例化边界扫描模块	C_SYSTEM_CONTAINS_MDM	整数 1 或 0，1 表示系统函数 MDM，0 则相反	0	Integer
JTAG 时钟是否使用 BUFG 组件	C_DISABLE_JTAG_CLOCK_BUFG_INSERTION	整数 1 或 0，1 表示不使用，0 表示使用	0	Integer
强制使用边界扫描端口 BSCAN_USER<n>	C_FORCE_BSCAN_USER_PORT	整数 1、3、4，其中 2 倍 OPB_MDM 使用	1	Integer
目标芯片家族	C_FAMILY	Xilinx 全系列 FPGA	Virtex-2	String

2) ChipScope_ila

ChipScope_ila 核可以监控所有自定义信号，与 ChipScope_icon 核的连接关系如图 9-117 所示。不过，要在 EDK 综合前就选择期望监控的信号，这一点与 ISE 中 ILA 核的使用是不一样的，后者需要在综合后才能选择信号连接。

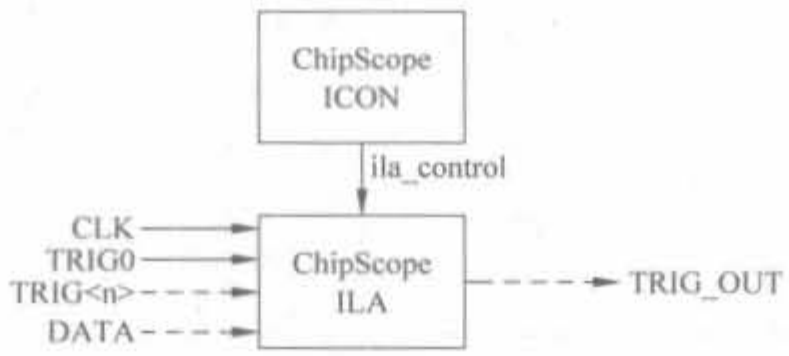


图 9-117 EDK 中 ChipScope_ila 核的连接架构

和 ISE 设计中的 ILA 核一样，ChipScope_ila 核是调试中使用最多的一类核，其主要参数如表 9-12 所示。

表 9-12 ChipScope_ila 核参数列表

特征描述	参数名	允许的数值范围	默认值	VHDL 类型
ILA 核采样的深度	C_NUM_DATA_SAMPLES	512、1024、2048、4096、8192 以及 16384	512	Integer
触发输出，可级联其他监控核	C_ENABLE_TRIGGER_OUT	整数 1 或 0，1 表示产生触发输出，0 表示禁止	0	Integer
目标芯片家族	C_FAMILY	Xilinx 全系列 FPGA	Virtex-2	String
是否将数据作为触发条件来使用	C_DATA_SAME_AS_TRIGGER	整数 1 或 0，1 表示数据就是触发条件，0 表示相反	0	Integer
采样数据的位宽，也就是需要标识的内部信号数	C_DATA_IN_WIDTH	整数 1～256	32	Integer

续表

特征描述	参数名	允许的数值范围	默认值	VHDL类型
在布局布线时，是否添加宏模块	C_DAISABLE_RPM	整数1或0，1表示不插入，0表示插入	0	Integer
是否禁止使用SRL16结构	C_DISABLE_SRL16S	整数1或0，1表示禁止，0表示允许使用	0	Integer
选择利用上升沿还是下降沿来采样数据	C_RISING_CLOCK_EDGE	整数1或0，1表示上升沿，0表示下降沿	1	Integer
是否使能ILA核中的顺序触发器	C_ENABLE_TRIGGER_SEQUENCER	整数1或0，1表示使能，0表示禁止	1	Integer
设置顺序触发器的触发级数	C_MAX_SEQUENCER_LEVELS	整数1～16	16	Integer
使能ILA核存储量化功能	C_ENABLE_STORAGE_QUALIFICATION	整数1或0，1表示使能，0表示禁止	1	Integer
设置匹配单元的数量，每个匹配单元可单独设置触发条件，且最多为256	C_TRIG<n>_UNITS	整数0～16，0表示禁止	0	Integer
每个匹配单元的触发条件数量，最多为256个	C_TRIG<n>_TRIGGER_IN_WIDTH	整数1～256	8	Integer
触发器计数器的位宽	C_TRIG<n>_UNIT_COUNTER_WIDTH	整数1～32	0	Integer
匹配类型	C_TRIG<n>_UNIT_MATCH_TYPE	同ISE中的触发匹配类型，包括 bisic、basic _ width _ edges、extend、extend_with_edges、range、range_with_edges	Basic	String

3) ChipScope_opb_iba

ChipScope_opb_iba是用于分析OPB总线的专用核。在EDK中，Xilinx公司建议不要使用ChipScope_ila核来分析总线信号，而专门提供了用于OPB总线信号分析的ChipScope_opb_iba核，可监控OPB总线的数据、地址以及控制信号总线，还可区分总线上的主、从设备。

ChipScope_opb_iba的连接结构如图9-118所示，通过MON_OPB来采样和监控OPB总线，所以其接口信号主要来自MON_OPB。

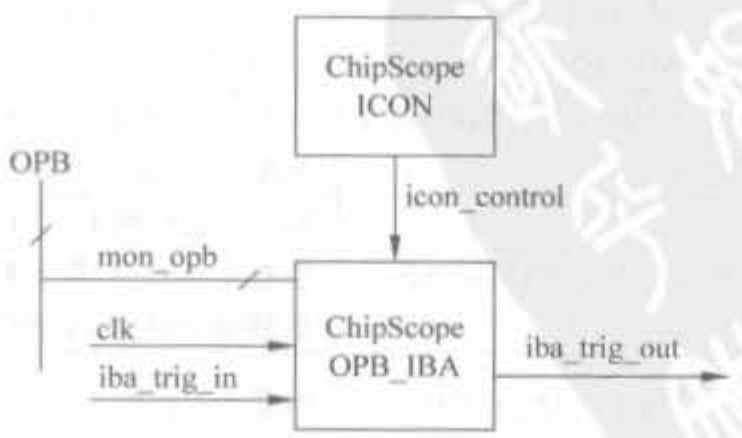


图9-118 ChipScope_opb_iba核的连接结构示意图

由于该核和 OPB 总线信号一一对应，使用前最好对 OPB 总线结构有一个大致的了解。该核在使用中非常简单，只需要直接将其和 OPB 总线连接起来即可，这里就不再过多介绍。

4) ChipScope_plb_iba

ChipScope_plb_iba 核主要用于监控 EDK 系统中的 PLB 总线信号，其内部结构、接口参数和 ChipScope_opb_iba 核一样，只是将 MON_OPB 换成 MON_PLB，这里不再介绍。

5) ChipScope_vio

ChipScope_vio 是一种用户可定制的实时监控核，主要用于观察 FPGA 内部信号，和其余核不同的是，它不占用芯片内部的块 RAM，可通过图形化界面配置，也可通过 Tcl 语言编写脚本，在网表级插入必要的监控点来实现。

由于 ChipScope_vio 核不使用块 RAM，只能通过带宽有限的 JTAG 时钟来采样，因此不适合监控高速信号。

2. XPS 中 ChipScope 核的应用实例

ChipScope 的原理和使用方法已在第 7 章进行了详细的说明，因此下面直接给出 XPS 中 ChipScope 的应用实例，将上文内容有机地衔接起来。

例 9-13　在 XPS 工程中插入 ChipScope_icon/ila/iba 核，完成设计的综合、实现与下载，并给出实际的采集结果。

1) 在 XPS 工程中添加和配置 ICON 核

在 EDK 中添加各个分析核，主要通过参数配置的方式来完成，和 ISE 中的图形化操作相比，显得不太直观，但需设置的内容是一致的。

(1) 添加、配置 ChipScope_icon 核

在 EDK 中使用 ChipScope 时，必须添加 ChipScope_icon 核，它位于“Debug”类 IP 列表中。打开 XPS 后，选中“ChipScope_icon”，单击鼠标右键，选择“Add IP”命令，将其添加到设计中。在“Bus Interface”窗口中，在“ChipScope_icon”上单击鼠标右键，选择“Config IP”命令，打开配置属性，如图 9-119 所示。

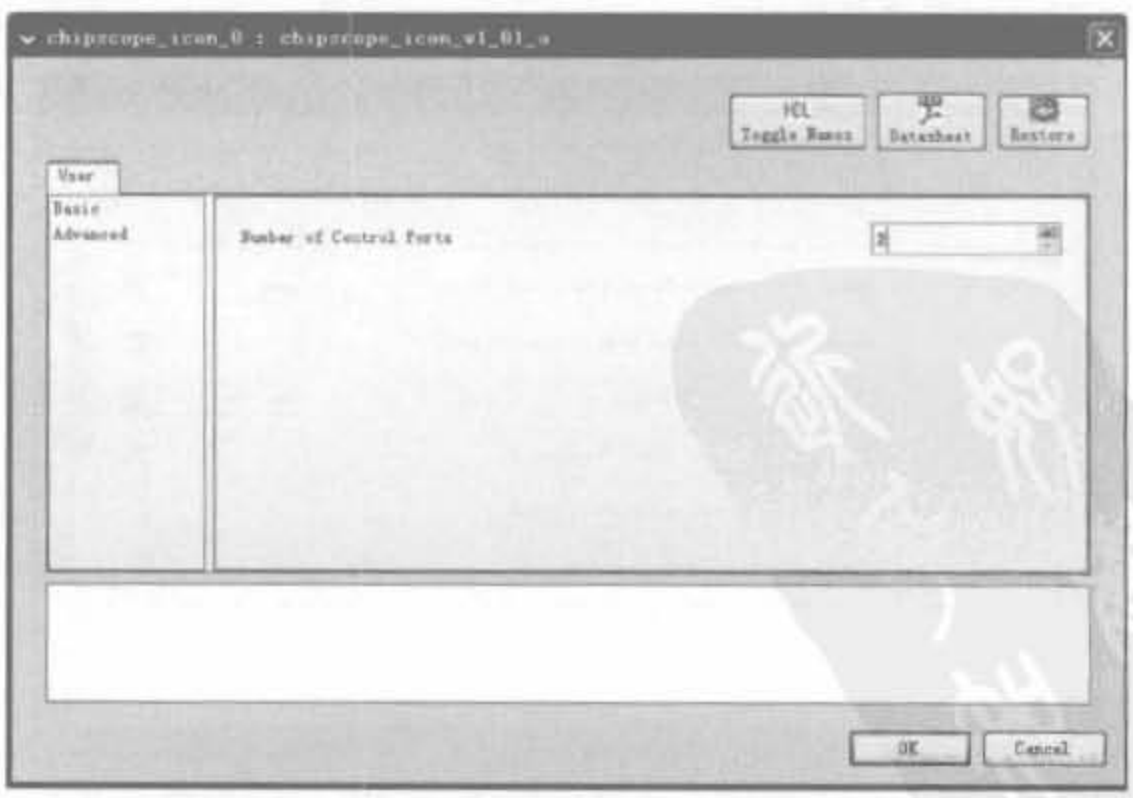


图 9-119　ChipScope_icon 核配置界面

本例计划采用 1 个 ChipScope_ila 核和 1 个 ChipScope_opb_iba 核，因此将“Number of Control Ports”设为 2，其他设置保持不变，单击“OK”即可。再切换到“Ports”窗口，在“Net”列完成控制端口命名。由于选择了两个核，所以只需要为 control0 和 control1 命名。

(2) 连接 ChipScope_icon 核

返回“Ports”窗口，展开 ChipScope_icon 核的端口。由于只添加了两个端口，因此只需在 control0、control1 通路行下拉框中选择 chipscope_icon_0_control0 和 chipscope_icon_0_control1，即可完成 ICON 核的端口连接，如图 9-120 所示。

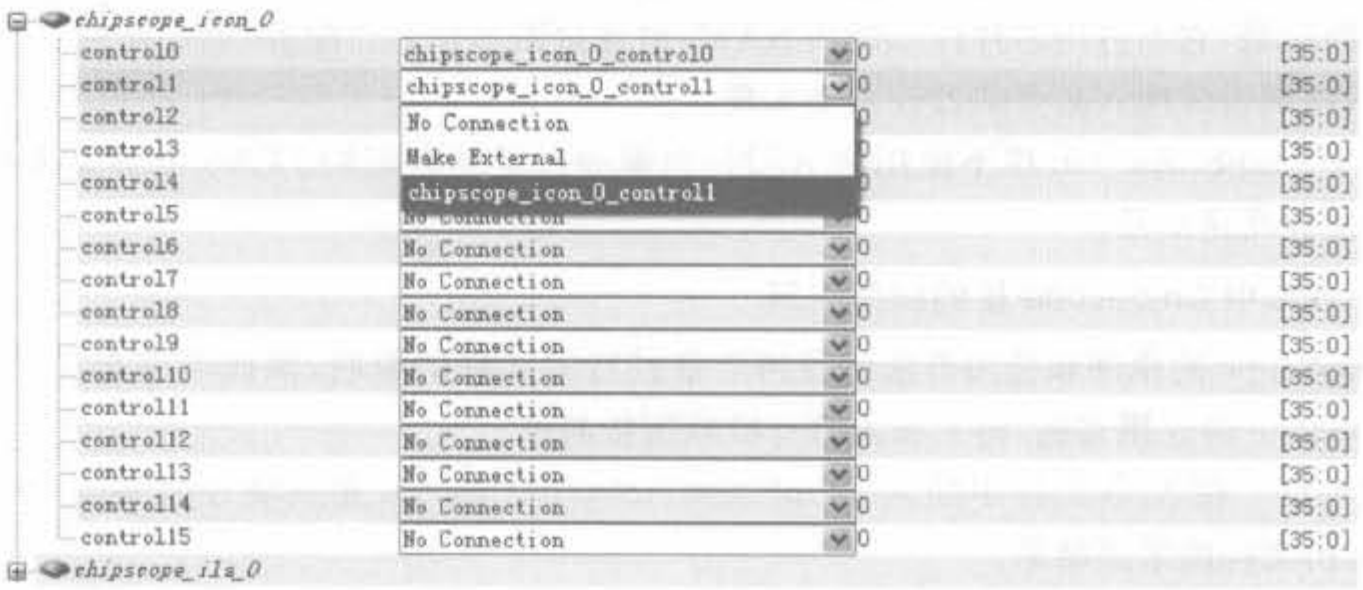


图 9-120　ChipScope_icon 核的端口连接界面

2) 在 EDK 中添加和配置 ILA 核

(1) 添加、配置 ChipScope_ila 核

ChipScope_ila 核是使用最多的核，也位于“Debug”类 IP 列表中。该核有众多的参数需要配置，分为“Misc”和“Trigger”两大类。Misc 参数是需要首先配置的，设置采样深度、信号位宽、是否将数据作为触发端口等关键参数，用户可根据需求选择，其界面如图 9-121 所示。可以看出，Misc 参数基本涵盖了 ISE 中 ILA 核的参数，各个参数的简要说明可查阅第 7 章中有关 ILA 核的相关参数的说明。

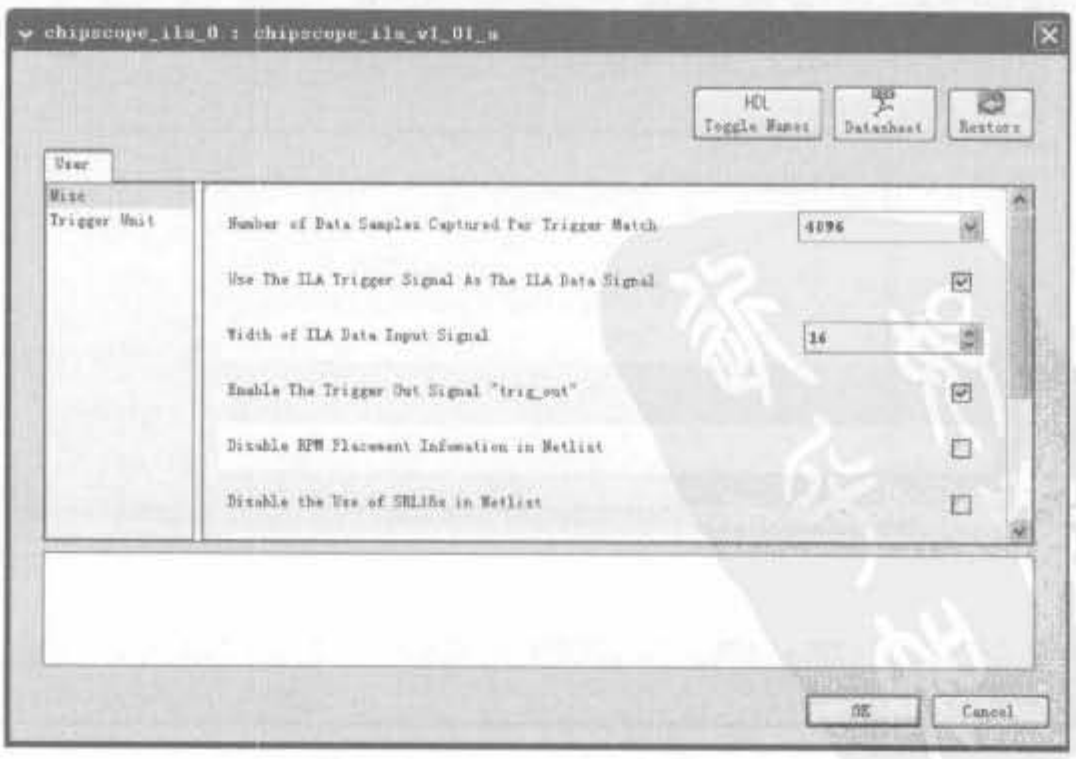


图 9-121　ChipScope_icon 核 MISC 配置界面

其次，需要配置 Trigger 参数，其界面如图 9-122 所示。其中，各类触发条件和 ISE 中 ILA 核的触发条件是一致的，且每个 ILA 可以配置 4 个触发单元参数。本例只需要 1 个触发单元，因此只需要配置触发端口 0。

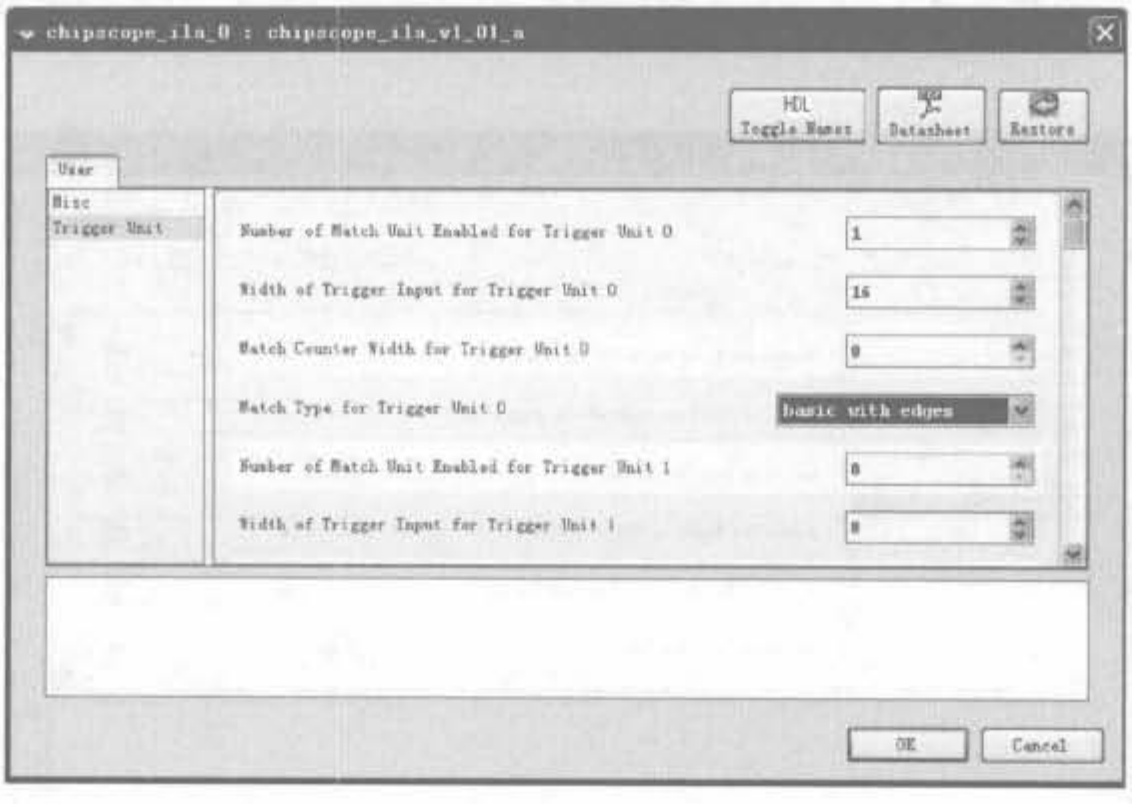


图 9-122　ChipScope_icon 核触发条件配置界面

(2) 连接 ChipScope_ila 核的端口

配置完毕并保存后，需要返回“Ports”窗口，展开 ChipScope_ila 核的端口，在 CHIPSCOPE_ILA_CONTROLD 行的 Net 列下拉框中选择 chipscope_icon_0_control0，将其和 ICON 核联系起来。如果使用 OPB 总线的时钟采样所有数据和触发信号，可在 CLK 行的 Net 列的下拉框中选择 sys_clk_s，如图 9-123 所示。在 TRIG0 行的 Net 列下拉框输入想要观测的信号，和 ISE 中的 ILA 核相比，其不足之处在于：这里不能通过鼠标操作添加多组信号，只能手动输入，多个信号之间通过“&”连接。

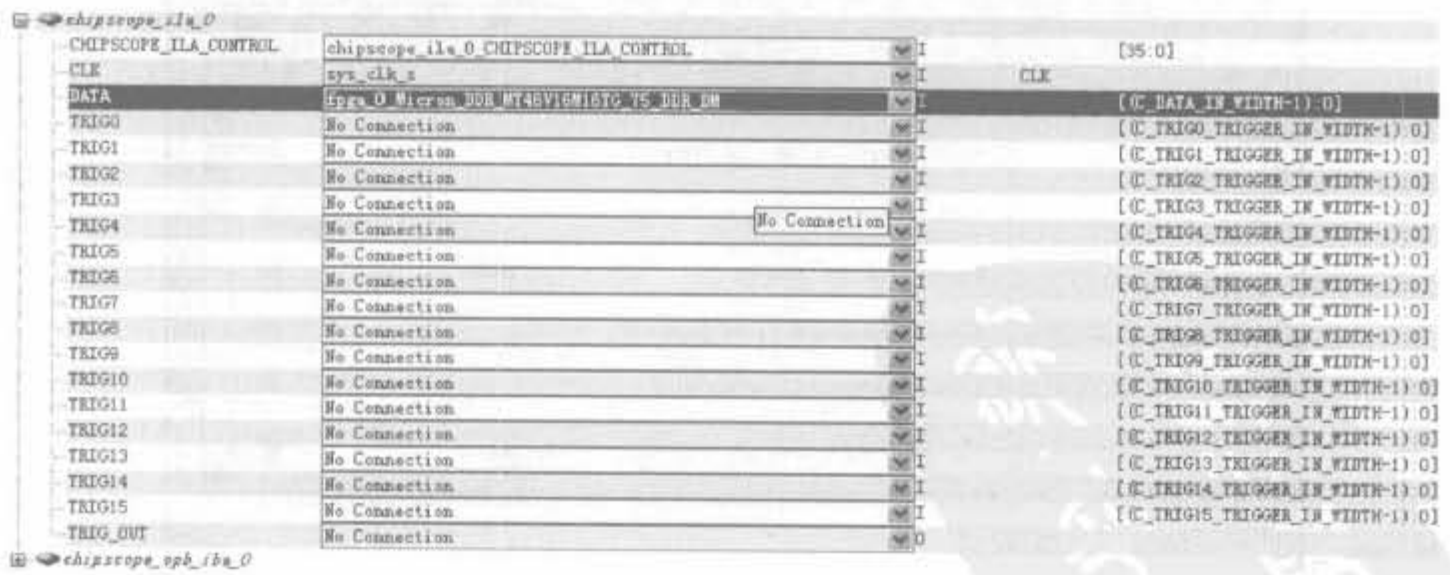


图 9-123　ChipScope_ila 核端口连接界面

3) 在 EDK 中添加 IBA 核

(1) 添加、配置 ChipScope_iba 核

ChipScope_iba 核的添加方法和 ChipScope_ila 的方法是相同的，只不过其配置比较麻

烦，需要配置 10 个子窗口，其中部分配置内容和 ChipScope_ila 的含义是相同的。下面主要介绍 OPB_Master 和 OPB_Slave 界面。

OPB_Master 页面主要配置 OPB 主设备的监控参数。在本例中，OPB 总线上只有 1 个主设备，因此 Master0 栏选择 1；不使用匹配计数器，将其宽度设为 0；其余配置参数如图 9-124 所示。

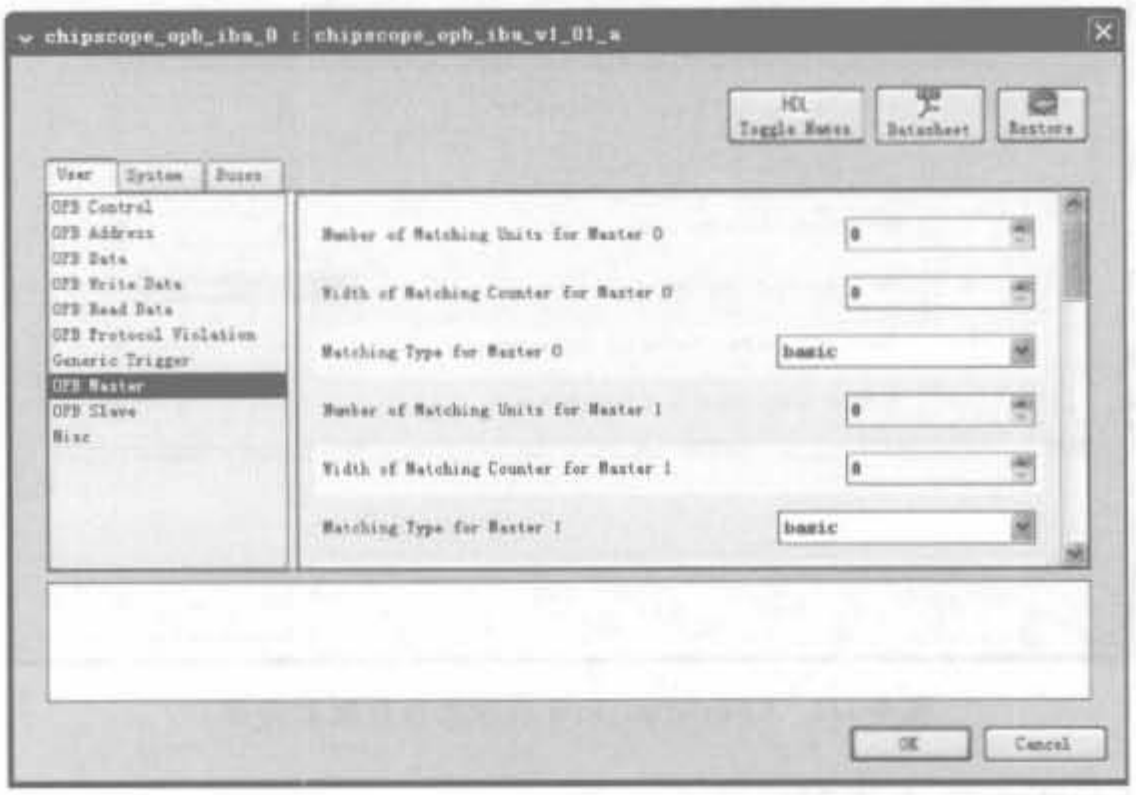


图 9-124　ChipScope_iba 核的 OPB_Master 配置界面

OPB_Slave 页面主要配置 OPB 从设备的监控参数，在本例中只监控 1 个 OPB 从设备，因此 Slave0 栏选择 1；不使用匹配计数器，将其宽度设为 0；其余配置参数如图 9-125 所示。

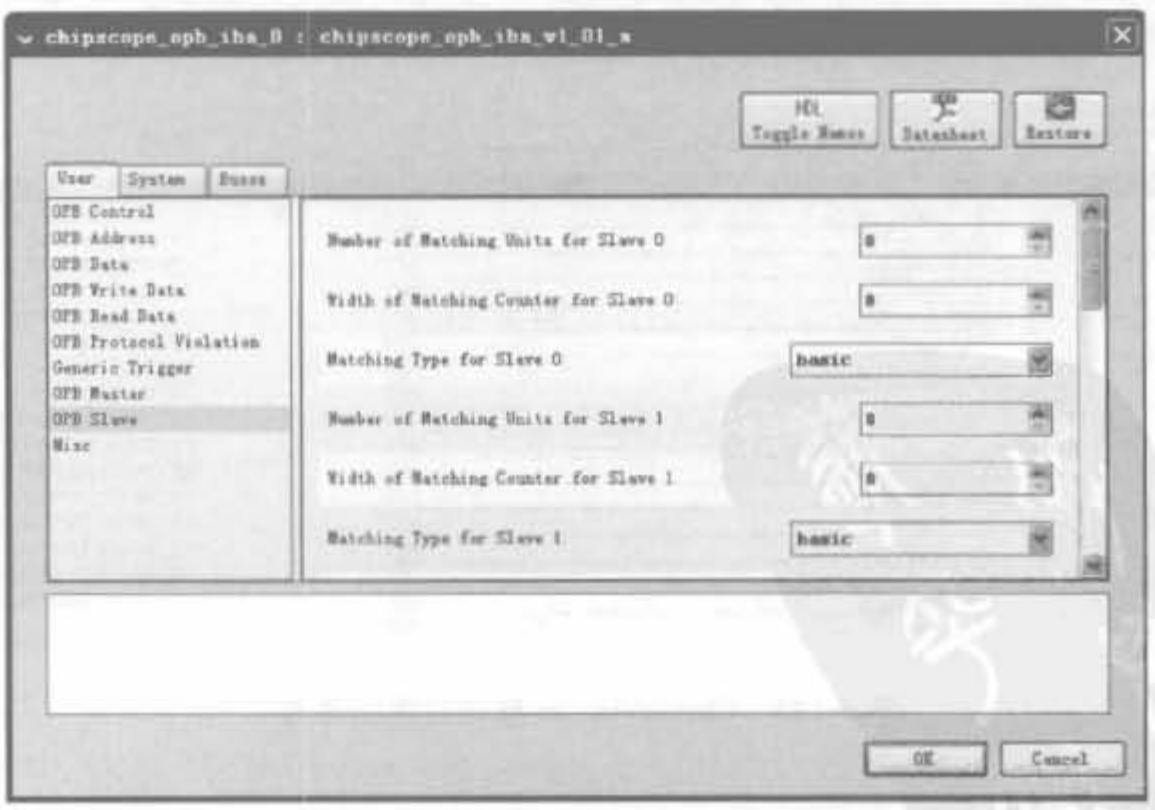


图 9-125　ChipScope_iba 核的 OPB_Slave 配置界面

(2) 连接 ChipScope_iba 核

配置完毕并保存后，需要返回“Ports”窗口。展开 ChipScope_iba 核的端口，在 CHIPSCOPE_ILA_CONTROLD 行的 Net 列下拉框中选择 chipscope_icon_0_control1，将其和 ICON 核联系起来；在 SYS_Rst 行的 Net 列选择 sys_rst_s 信号，如图 9-126 所示。

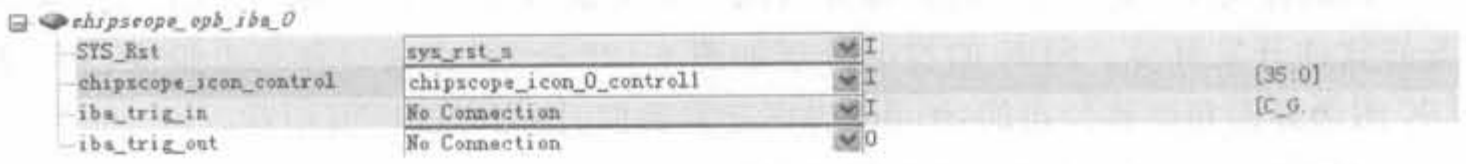


图 9-126　ChipScope_iba 核端口连接界面

4) 综合、实现

首先，选择“Hardware”→“Generate Netlist”命令，完成对硬件平台的综合；其次，选择“Hardware”→“Generate Bitstream”菜单命令，启动实现的过程，将 NGC 网表文件转化为硬件平台比特流文件；最后，选中“Device Configuration”→“Update Bitstream”命令，将编译所生成的可执行文件和硬件比特流合在一起，形成最终的二进制比特流文件，位于工程目录下的 implementation 文件夹中。

5) 使用 ChipScope Analyzer 采集数据

按照 6.5.2 节的内容，利用 Analyzer 将生成的比特文件下载到 FPGA 中，设定触发条件后，iba 核采集的数据波形如图 9-127 所示。

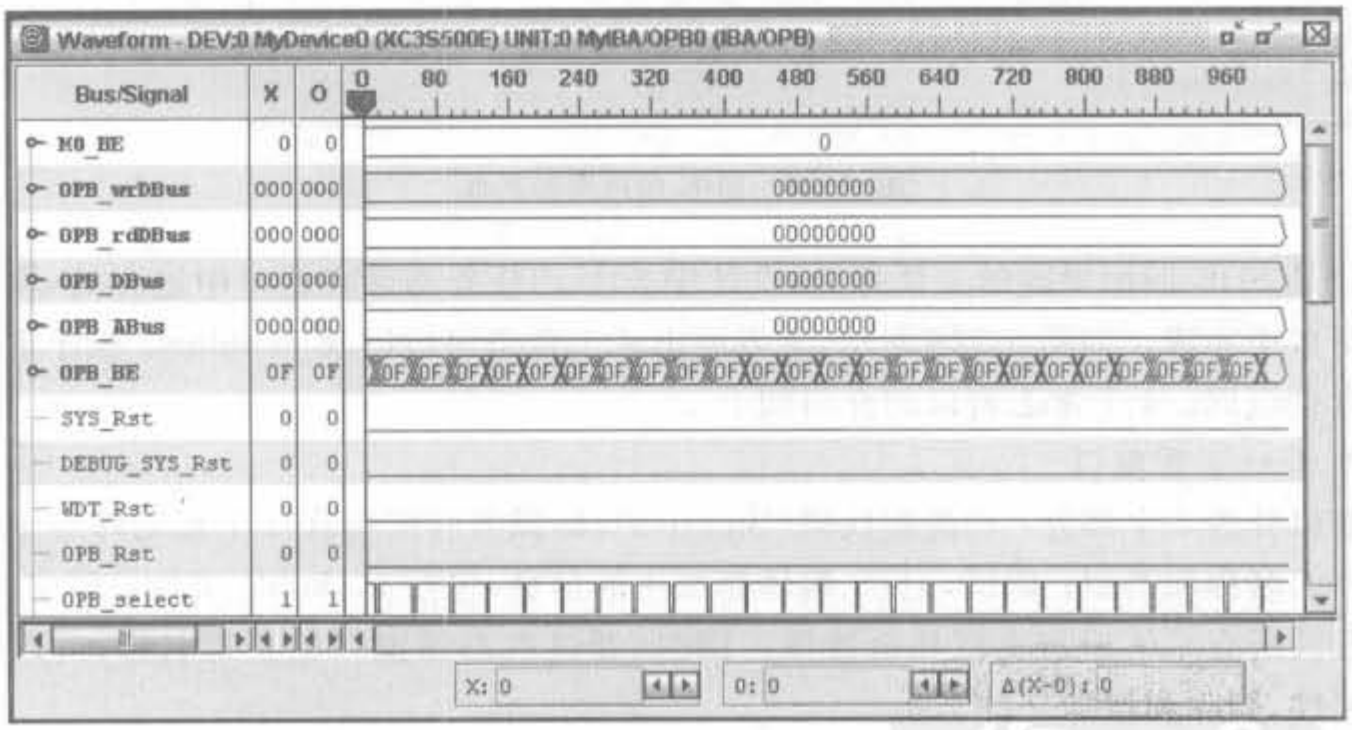


图 9-127　Analyzer 采集数据界面

读者需要注意的是，在“Waveform”窗口中，信号名称默认为 DataPort[n]，不易观察，因此需要从工程中通过导入 iba 核的.cdc 文件辅助显示信号名，它位于工程目录下的 implementation/chipscope_opb_iba_0_wrapper 目录中，文件名为 cs_coregen_chipscope_opb_ila_0.cdc。

9.5.8 软件平台 SDK 的使用

1. SDK 图形界面

在嵌入式开发中，设计人员的大部分时间用来完成软件开发，SDK 为设计工程师提供了完备的软件开发环境。SDK 的界面外观如图 9-128 所示，由窗口和菜单组成，从外形上看，SDK 图形界面和微软公司的 Studio 界面非常类似，使用方法也很相近。

图 9-128 SDK 用户图形界面

窗口是 SDK 图形界面的主要部分，设计中 80%的任务都是在窗口中进行的，调试和分析也在窗口中完成。如图 9-128 所示，主界面由 4 个常态窗口以及 Windows 程序常见的菜单和工具栏组成。4 个常态窗口的介绍如下。

1）C/C++工程窗口

该窗口分为 3 个部分：二进制代码(Binaries)，标识当前应用软件工程编译后生成的二进制文件的名称和类型；调试，记录、编译调试用的信息，包括 Makefile 文件；源文件，包含加入工程的所有 C/C++源文件和链接库。同时，通过右击该窗口的工程名称，可以设置该工程的属性，删除和增加文件。

2）源代码编辑窗口

源代码编辑窗口和大部分集成环境类似，此外还具备以下两个独特功能：双击任何一行的行号，可以在当前行设置/取消断点；在编译出错时，出错行的行标前会出现红叉标志错误。

3）大纲窗口

大纲窗口显示了工程的所有函数和所用到的库。单击任何函数，源代码编辑窗口会自动显示该函数体的源代码，便于用户浏览函数并修改函数的内容。

4）信息窗口

信息窗口主要用于调试阶段，帮助用户分析和定位错误，包括问题窗口(Problems)，显

示编译阶段的所有错误；控制台窗口(Console)，显示代码以及系统输出信息；属性窗口(Properties)，显示当前工程的属性设置；XMD 控制台，用于 XMD 实时调试的交互，可输入指令，也可显示调试信息；进程菜单(Process)，显示 SDK 正在运行的操作情况；搜索窗口(Search)，可查找字符，并返回查找结果。

另外，SDK 支持团队开发。对于每个应用软件工程来讲，必须要先建立工作域，在共同的工作域中建立多个应用软件工程，从而达到共同开发的目的；对于单一工程师来讲，也需要遵循这一过程。

2. 导入应用

运行 SDK，导入/新建应用软件。这里，我们先导入运行 BSB 向导时所产生的应用软件。

(1) 选择“Software”→“Launch Platform Studio SDK”，打开 SDK。

(2) 打开 SDK 后，出现的向导将帮助用户创建软件应用工程。在向导的对话框里，选择“Import XPS Application Projects”，然后单击“Next”，如图 9-129 所示(注：用户当然可以选择创建一个新的 SDK 应用)。对于导入的 XMP 文件，即顶层的 XPS 工程文件，会自动告知 SDK 在硬件平台里使用的处理器，并为每个处理器提供一个指向库的指针。SDK 只管理用户的应用软件；而 XPS 管理那些构成软件平台的库和驱动。

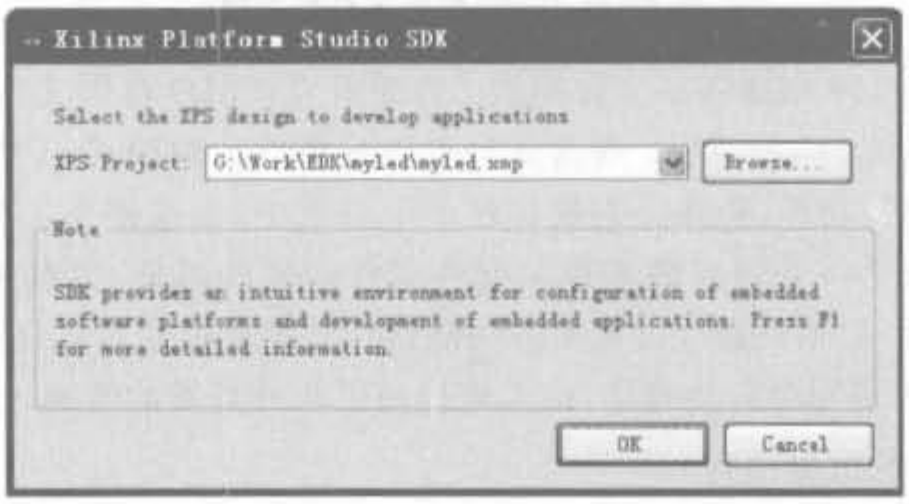


图 9-129　导入 XPS 工程的自动向导示意图

如果此向导没有自动打开，或者用户要更新 XPS 工程，可在 SDK 中选择“Xilinx Tools”→“Launch Application Wizard”命令打开向导，其界面如图 9-130 所示，且会显示所选工程是否标记为 BRAM 的初始化工程。

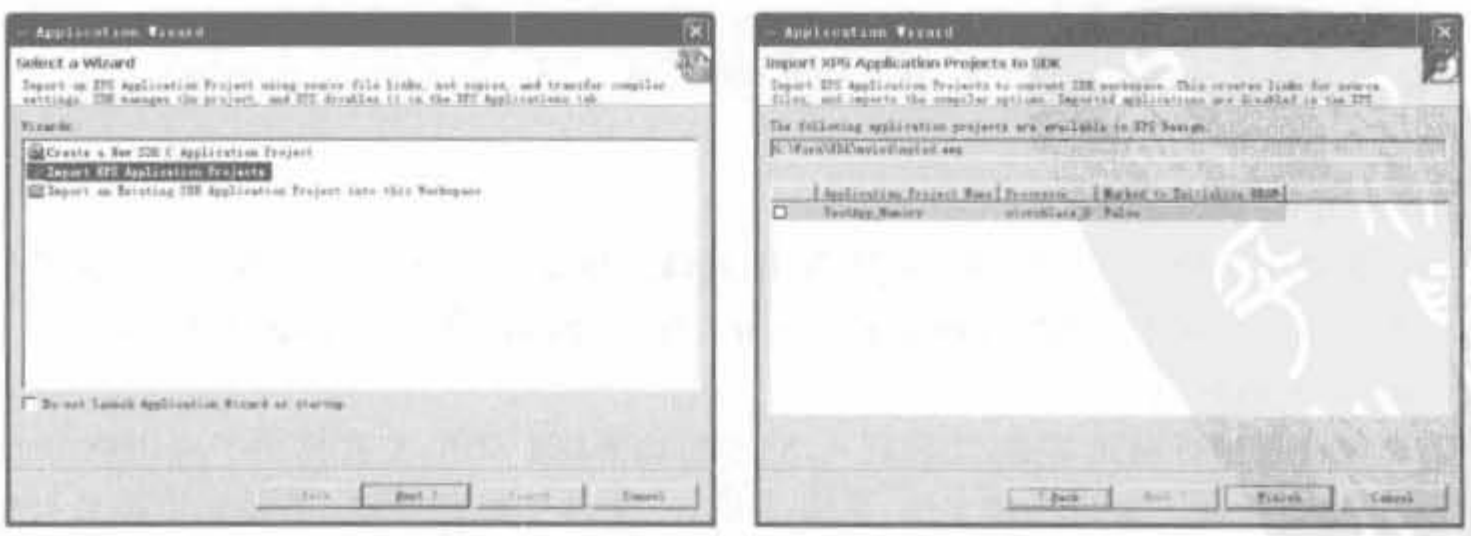


图 9-130　在 SDK 中导入 XPS 工程示意图

3. 创建工程以及调试代码

进入 SDK 界面后，选择“File”菜单下的“New”命令，进入工程建立窗口，如图 9-131 所示。设计人员可选择 C 语言应用软件工程、C++语言应用软件工程、CVS 应用软件工程以及简单软件工程 4 类不同的模板。

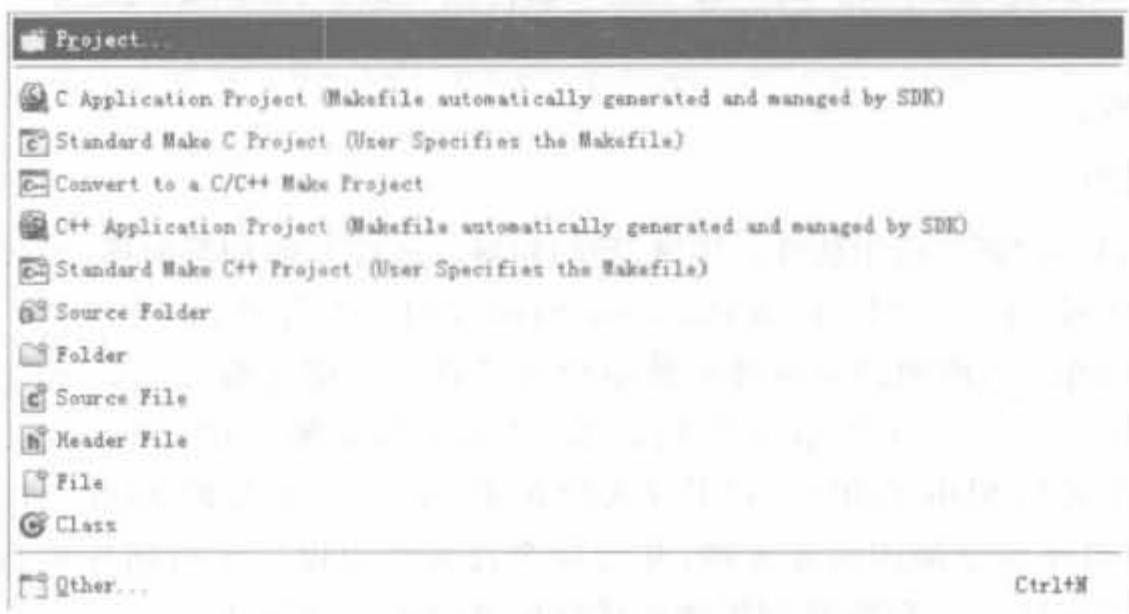


图 9-131 SDK 新建工程示意图

同时，SDK 支持两种 Makefile 文件模式：自动生成和用户自定义。后者主要用于高级开发。对于嵌入式系统的初学者，一般选择自动生成模式比较稳妥。当用户应用代码文件众多时，需要其按类型、功能、模块分别放在若干个目录中，但这样手工管理非常不方便，因此通过 makefile 定义了一系列规则来指定哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作将编译过程自动化。写好后，只需要一个 make 命令，整个工程完全自动编译，极大地提高了软件开发的效率。

4. XPS 和 SDK 的数据交互

在 EDK 解决方案中，XPS 可以直接将设计转移到 SDK，但没有提供 XPS 直接读入已在 SDK 中调试成功的应用软件工程，可以说是 EDK 整体解决方案的一个遗憾。但是，作为一个嵌入式解决方案，必须将软、硬件程序最终生成一个二进制比特文件下载到 FPGA/Flash 中。因此，下面介绍如何实现 XPS 和 SDK 的数据交互。

1）由 XPS 进入 SDK

(1) 手动交互

手动交互就是关闭 XPS 工程，然后打开 SDK，通过导入向导将 XPS 设计的. xmp 工程文件导入。

(2) 自动交互

EDK 提供了 XPS 到 SDK 的自动转换，因此该过程非常简单。在 XPS 中完成硬件开发后，单击“Software”菜单下的“Launch Platform Studio SDK”命令，直接进入 SDK。

2）由 SDK 返回 XPS

EDK 没有提供自动将 SDK 工程读入 XPS 的功能，但 SDK 工具提供了应用软件的 elf 文件，因此 XPS 生成下载目标板文件，只需要从 elf 文件中提取数据，不需要应用软件工程的其他支持。直接在 XPS 的工程名上单击鼠标右键，在出现的菜单中选择“Set Compiler

Options”，打开编译属性衍生子窗口，修改生成的 elf 文件路径和名称，指向 SDK 生成的应用软件 elf 文件，单击“OK”保存设置。这样，XPS 在生成比特文件时，会自动加入 SDK 生成的 elf 文件。需要注意的是，利用这种方法完成的交互，由于没有软件部分的源代码，因此所有的软件代码调试还需要在 SDK 中完成。

更多的时候，为了建立并运行仿真，用户需要回到 XPS 中。如前面介绍的，利用 SDK 完成了软件开发工作后，需要在 XPS 中定义一些有关工程管理的问题：

(1) 必须指定用于 BRAM 初始化的应用程序。应用标签提供了这一功能。

(2) 为了和“Test_App_Peripheral”协调工作，XPS 需要发现可能的工程管理冲突。由于这时我们还在利用 SDK 来管理这个软件工程，因此 XPS 将在工程例子中找到一个冲突。

这里可能出现一个问题：假设有两个用户操作这个 XPS 工程，一个使用 XPS，而另一个使用 SDK。那么，最后保存这个工程的用户将覆盖另一个用户的工作。为了避免这一情况的发生，XPS 会指出可能的冲突并创建一个可以工作的稳定文件环境。由于更倾向于选择 SDK 作为软件工程管理者，因此 XPS 只需知道 ELF 文件的位置，这样就可以将其和之后的 FPGA 比特流进行合并。注意，在 XPS 应用标签中还有其他工程，在执行下一个步骤前，用户需要确认在工程中是否存在下列部分：

(1) 默认的 ppc405_0_bootloop/MicroBlaze_0_bootloop 工程。Bootloop 工程启动处理器，并为其提供所需要的 jump-to-address 命令来寻找外部存储器。注意，此 bootloop 工程不能用来初始化 BRAM(可利用工程 Test_App_Peripheral 来执行这一任务)。

(2) 由 BSB 向导创建的工程，包括 TestApp_Memory 和 TestApp_Peripheral 工程。在前面介绍的 BSB 向导中，我们选择了测试存储器和其他外围设备。在下面的步骤中，我们将选择并配置有关软件，这样就可以对其进行仿真并下载到 FPGA 或 board 存储器中。

5. 在 SDK 中为用户定制的 IP 增加测试软件

在 SDK 环境中，可以给先前创建的用户定制外设(test_ip)增加一些测试软件。整个过程包括：

(1) 查找此核的软件测试文件。

(2) 用户将这些测试文件导入到应用工程 TestApp_Peripheral 中。

(3) 编辑 test_ip_selftest.c 文件，为 test_ip 核指明基地址(因为 test_ip_selftest 程序需要一个基地址指针)。用户可以参考 xparameters.h 文件获取这一信息。

(4) 重新创建工程(可以设置 SDK 来自动完成这一过程)。

下面通过实例来介绍如何为定制 IP 添加测试软件。

例 9-14　本例实现在定制 IP 中增加测试软件。

(1) 需要导入软件测试文件。首先，单击 SDK 主窗口处的“C/C++Projects”标签；其次，在“C/C++ Projects”面板中的“TestApp_Peripheral”工程处单击鼠标右键，选择“Import”；第三，在导入对话框中选择“File system”。浏览顶层工程下的驱动器目录，找到 TestApp_Peripheral/src 目录，最后选择所有的源文件，单击“Finish”，如图 9-132 所示。

(2) 编辑 test_app_peripheral.c 文件。在“C/C++Projects”标签中，找到“test_ip_selftest.c”文件，双击打开此文件；test_ip_selftest.c 文件包括了 TEST_IP_SelfTest 程序的功能定义，如图 9-133 所示。

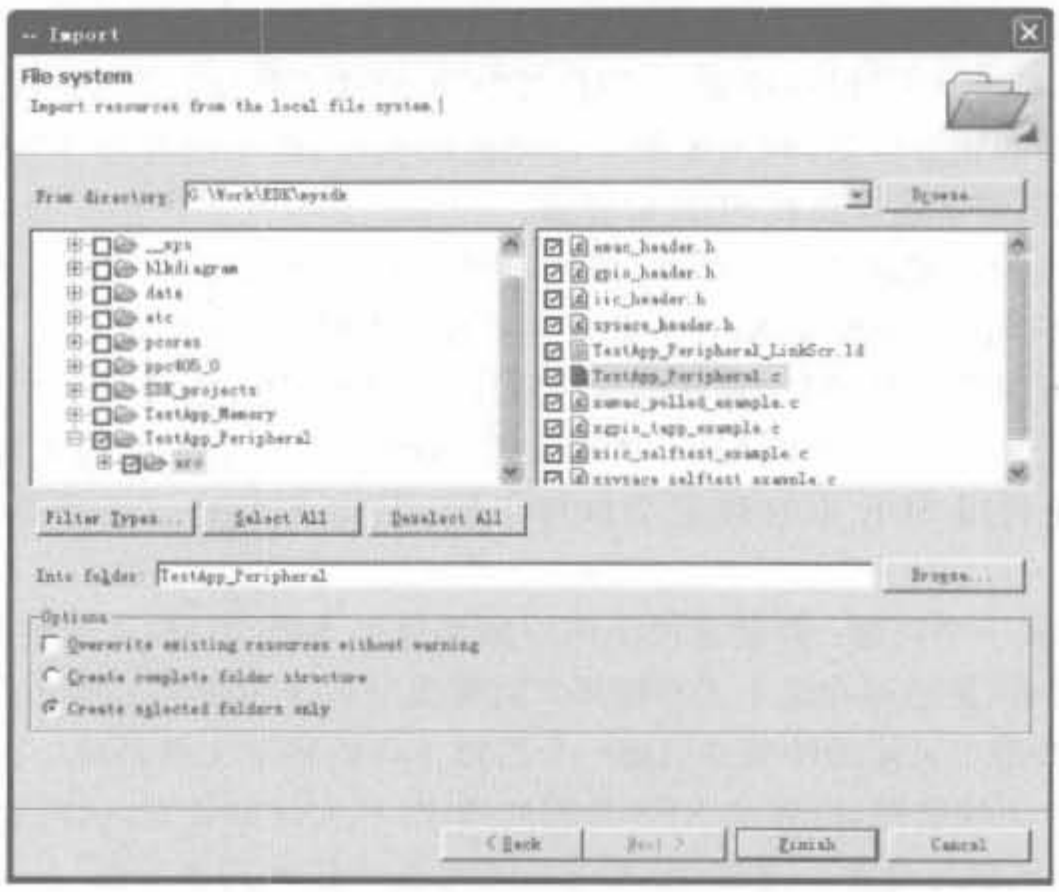


图 9-132　源文件选择窗口

```
119/*****************************************************************
120XStatus SysAceSelfTestExample(Xuint16 DeviceId)
121{
122    XStatus Status;
123
124    /*
125     * Initialize the instance. The device defaults to polled
126     */
127    Status = XSysAce_Initialize(&SysAce, DeviceId);
128    if (Status != XST_SUCCESS)
129    {
130        return XST_FAILURE;
131    }
```

图 9-133　test_ip_selftest.c 文件

其次，从图中可以看到，TEST_IP_SelfTest 程序需要一个基地址指针。用户可以按照如下步骤在 xparameters.h 中找到 TEST_IP 的基地址值：在“C/C++Projects”标签中，打开“ppc405_0_sw_platform/ppc405_0/include”目录以便显示 xparameters.h 文件。双击“xparameters.h”，在编辑窗口打开此文件。查找“TEST_IP_0_BASEADDR”。这样，设计人员就得到了基地址定义信息，在为 TestApp_Peripheral.c 文件增加功能时需要利用此信息。

最后，在 TestApp_Peripheral.c 文件中，在最后的打印语句前插入如下代码：

```
TEST_IP_SelfTest(XPAR_TEST_IP_0_BASEADDR);
```

(3) 重建工程。如果选择了“Build automatically”选项(在 Project 工具栏中)，在保存 TestApp_Peripheral.c 文件时，工程自动得到更新。如果没有选定此选项，那么选择“Project”→“Build Project”。完成创建后，可以看到，在“TestApp_Peripheral”工程下创建了 Debug 目录。工程的 ELF 文件即位于此目录中，在之后的测试驱动时我们将使用该 ELF 文件。此时，用户完成了 SDK 中所需要的所有工作。

(4) 利用“C/C++Build”的配置，用户可以控制所建立的工程类型。用户可以在 SDK 中

选择“Help”→“Help Contents”，利用其中的“C/C++ Development User Guide”→“Reference”→“C/C++ Project Properties”→“Managed Make Projects”→“C/C++ Build”→“Build Settings”得到更多的帮助信息。

(5) 返回 XPS 完成最终测试。首先，在 XPS 中选择“Applications”标签；其次，在“Project: TestApp_Memory”上单击鼠标右键，去掉初始化 BRAM 的选项(我们将利用“TestApp_Peripheral”来完成这一工作)；最后利用 XPS 来管理数据，在 TestApp_Peripheral 工程中将利用这些数据来初始化 BRAM。在 SDK 中创建了 TestApp_Peripheral 工程后，XPS 将认为用户正利用 SDK 管理此工程。为了操作 TestApp_Peripheral 工程，XPS 需要用户将其改变为 ELF-only 的工程。双击“Project: TestApp_Peripheral”打开如图 9-134 所示的对话框。

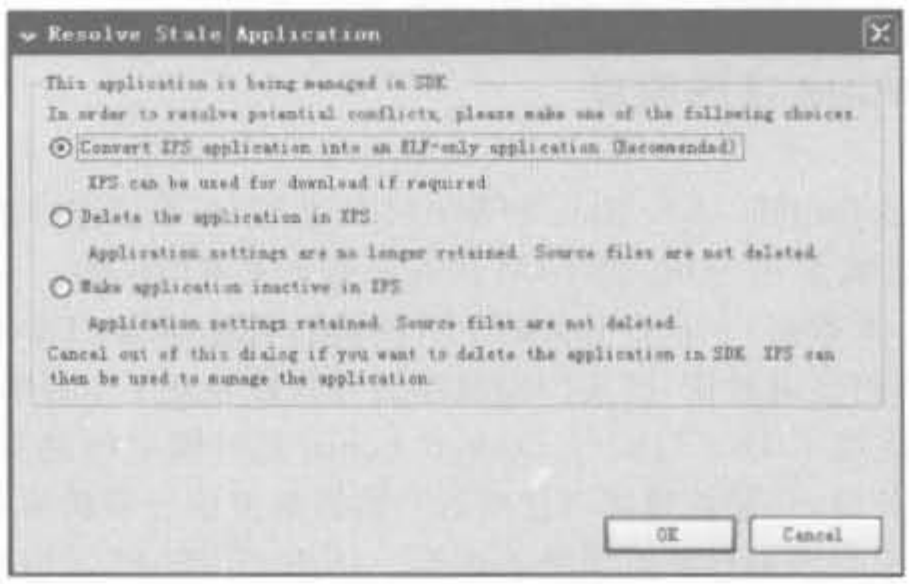


图 9-134　XPS ELF 文件管理选项

(6) 选择“Convert XPS application into an ELF-only application”。单击“OK”后，XPS 会继续管理初始化 BRAM 所用到的数据，但将软件工程管理功能转移给 SDK。在“Project: TestApp_Peripheral”工程上单击鼠标右键弹出的菜单中，选其为初始化 BRAM 的工程。此时用户可以看到在“Mark to Initialize BRAMs”边上有打勾的标记，如图 9-135 所示。

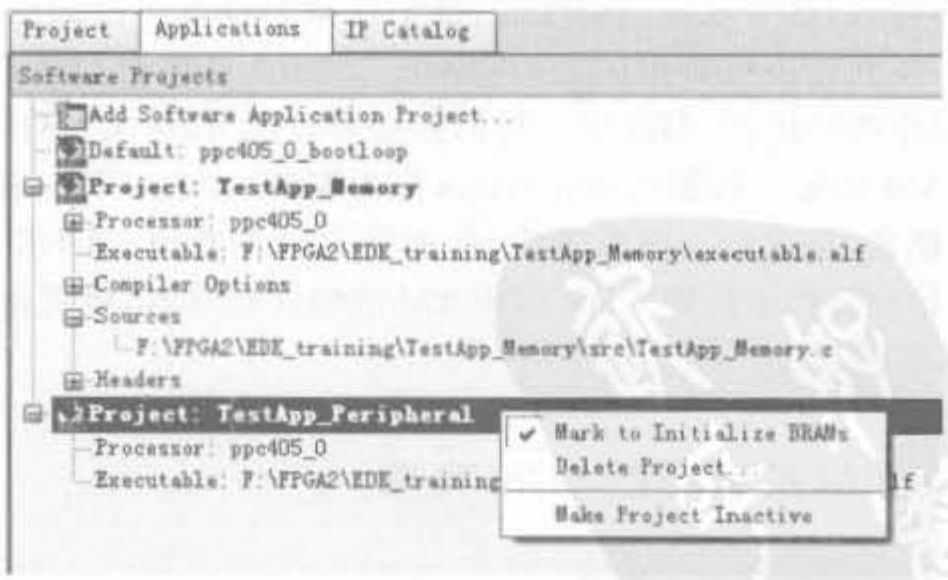


图 9-135　选择初始化 BRAM 的工程

(7) 在应用标签中选择 TestApp_Peripheral 工程右键菜单中的 Executable 选项。在“\SDK_projects\TestApp_Peripheral\Debug”目录中查找 SDK 生成的 ELF 文件(在前面

的例子中，SDK 工具已将此 ELF 文件放在 Debug 目录中）。

现在，软件和硬件部分都已创建，接下去的工作就是对它们进行测试。可以通过下载到实际的硬件电路板完成。由于硬件电路板的多样化，限于篇幅，这里就不再介绍。

9.6 EDK 开发实例——DDR SDRAM 接口控制器

存储器管理单元(MMU)是一个嵌入式系统最基本的组成部分，是加载操作系统所必不可少的。本节结合 Xilinx 公司的 Spartan-3E Starter 开发板平台，给出完整的 DDR SDRAM 控制器的开发与调试过程，将本章主要内容组织成一个有效整体，便于读者理解和掌握 Xilinx EDK 的工程开发流程。

9.6.1 DDR SDRAM 工作原理

DDR SDRAM 是 Double Data Rate SDRAM 的缩写，是双倍速率同步动态随机存储器的意思。DDR SDRAM 是在 SDR SDRAM 内存基础上发展而来的。SDR SDRAM 在一个时钟周期内只传输一次数据，它是在时钟的上升期进行数据传输；DDR SDRAM 则在时钟的上升沿和下降沿都可传输数据，因此传输数据的等效频率是工作频率的 2 倍。

DDR SDRAM 使用了 DLL(Delay Locked Loop，延时锁定回路提供一个数据滤波信号)技术。当数据有效时，存储控制器可使用这个数据滤波信号来精确定位数据，每 16 次输出 1 次，并重新同步来自不同存储器模块的数据。DDR SDRAM 采用的是支持 2.5V 电压的 SSTL2 标准，而不是 SDR SDRAM 使用的 3.3V 电压的 LVTTL 标准。

DDR SDRAM 芯片的主要供货商包括美光(Micron)、三星(Sungsang)、现代(Hynix)等。开发板上的 DDR 芯片型号为 Micron 公司的 MT46V16M16TG-75(注：不同厂家 DDR SDRAM 的芯片管脚的功能定义都是兼容的)。

DDR SDRAM 也是利用内部电容的电荷来记忆数据信息的，但电容的电荷会随着时间而泄漏，所以要在数据信息变得难以辨认之前完成数据刷新(更新)，即将数据读出(但并不送到芯片管脚上)再写入，这一般是周期性的，整个存储器进行一次刷新的时间间隔为刷新周期。在刷新期间，不允许进行数据的读、写操作。SDRAM 的存储体是按照行、列组织的二维矩阵，而刷新操作按行进行，每次对一行的数据同时读出、放大、整形和再写入。根据标准规定，DDR SDRAM 的每一行都必须在 64ms 以内刷新一次。DDR SDRAM 有自动刷新和自刷新两种刷新模式，且在每次突发读取时，都会自动预充电。DDR SDRAM 芯片在上电后必须由一个初始化操作来配置 DDR SDRAM 的模式寄存器，模式寄存器的设置决定了 DDR SDRAM 的刷新模式。

9.6.2 DDR SDRAM 控制器的 EDK 实现

1. DDR SDRAM 控制器的基本要求

内存控制器的功能是监督控制数据从内存载入/载出，并对数据的完整性进行检测。一般来讲，DDR 控制器的开发难度是比较大的，其基本要求有：

- 可配置的数据突发长度 2、4、8；

- 可配置的 CAS 等待时间 1.5、2、2.5、3；
- 支持的 DDR SDRAM 命令包括设置模式寄存器(LOAD_MR)、自动刷新(AUTO_REFRESH)、预充电(PRECHARGE)、激活(ACTIVE)、自动预充读(READA)、自动预充写(WR ITEA)、突发停止(BURST_STOP)、空操作(NOP)；
- 接口速率大于等于 50MHz，双倍数据速率。

2. DDR SDRAM 控制器的 EDK 实现

在 Xilinx 的 EDK 开发环境中，提供了 DDR SDRAM 的控制器 IP Core，可以让用户在短短几分钟之内完成 DDR 控制器的开发，极大地节约了研发周期。下面用一个实例详细说明 EDK 平台上快速开发 SDRAM 控制器的步骤和软、硬件调试方法。

例 9-15　在 Spartan-3E Starter 开发板上利用 EDK 实现 DDR SDRAM 接口控制器的软、硬件平台。

(1) 运行 ISE 9.1，利用 BSB 向导建立新的工程，如图 9-136 所示。

(2) 单击“OK”按钮，输入工程目录，如图 9-137 所示。

图 9-136　利用 EDK 建立新的工程

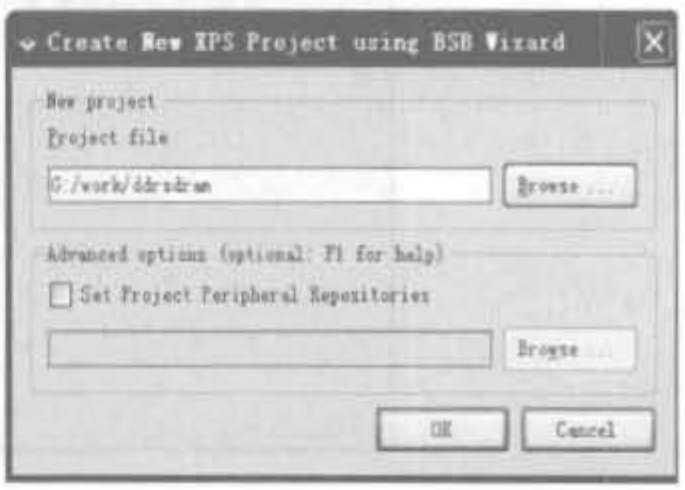


图 9-137　输入 EDK 工程路径示意图

(3) 单击“OK”按钮，进入 BSB 配置页面，选择新建工程选项，如图 9-138 所示。

图 9-138　BSB 建立工程示意图

(4) 单击“Next”按钮，进入电路板选择界面，在“Select board”栏选择“I would like to create a system for the following development board”；在“Board vendor”下拉框中选择“Xilinx”；在“Board name”下拉框选择“Spartan-3E Starter Board”；在“Board revision”下拉框中选择“D”，如图 9-139 所示。

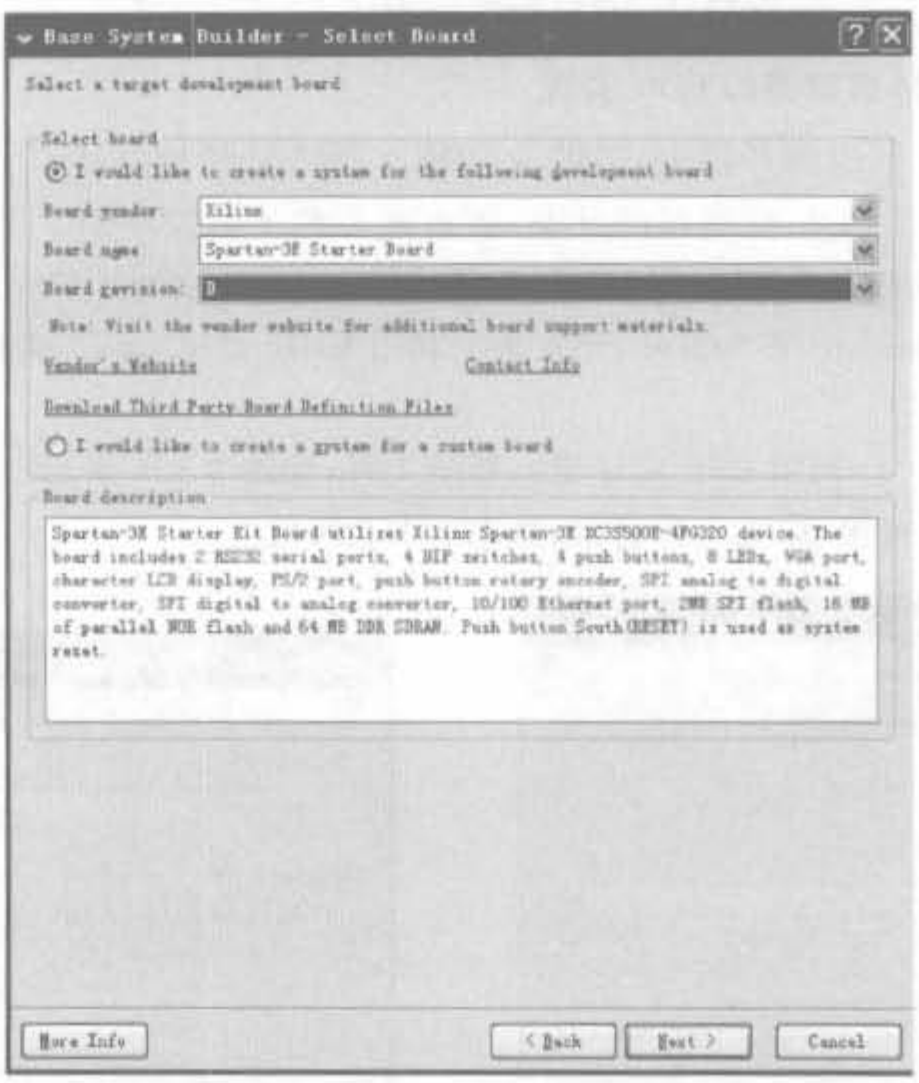


图 9-139 开发板选择界面

(5) 单击“Next”按钮，进入处理器选择界面，如图 9-140 所示。Spartan-3E 系列 FPGA 只能选择 MicroBlaze 软核处理器。

(6) 单击“Next”按钮，进入处理器配置页面。处理器时钟设为 50MHz，使能片上调试模块(On-chip H/W debug module)，片上存储器(Local memory)配为 8KB，如图 9-141 所示。

(7) 单击“Next”按钮，进入处理器 I/O 配置页面。其中，外部 I/O 配置页面总共有 3 页，如图 9-142～图 9-144 所示。第 1 页包括 RS232_DCE、RS232_DTE、LEDS_8Bit 等外设；第 2 页有 DIP_Switches_4Bit、Buttons_4Bit 以及 FLASH_16M×8 等外设；第 3 页包括 DDR_SDRAM_32M×16、Ethernet_MAC、DMA 等外设。I/O 设备都是引到芯片实际的管脚上的，和电路板的设计是一致的。由于 ISE 9.1 中自带了 Spartan-3E 开发板的外设，因此不需要用户修改，全部采用默认值。

(8) 单击“Next”按钮，进入处理器外设配置页面，如图 9-145 所示。I/O 列表添加的是通用的硬件驱动模块，且 Xilinx 提供了相应的驱动，直接使用即可，而外设的底层硬件驱动需要用户自己开发。由于开发板上的接口电路都是 I/O 设备，因此没有外设。

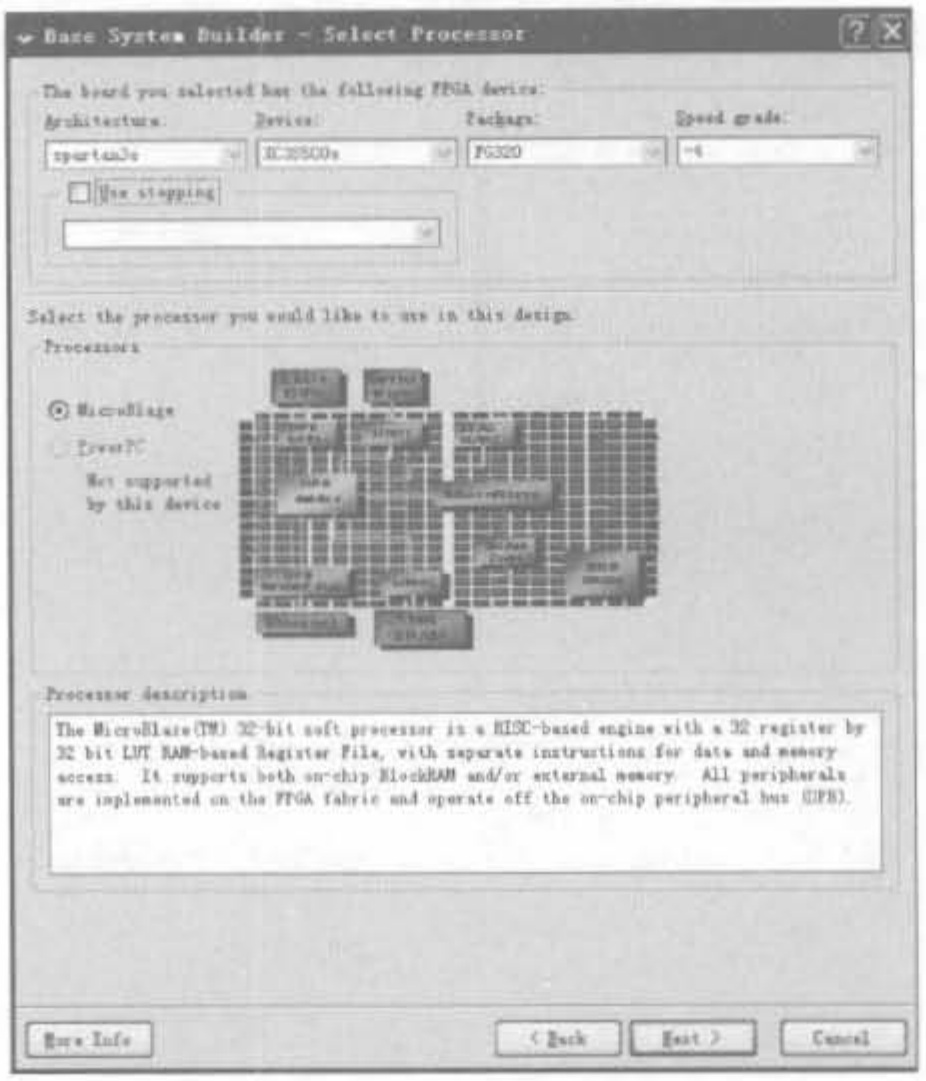


图 9-140 处理器选择界面

图 9-141 处理器配置界面

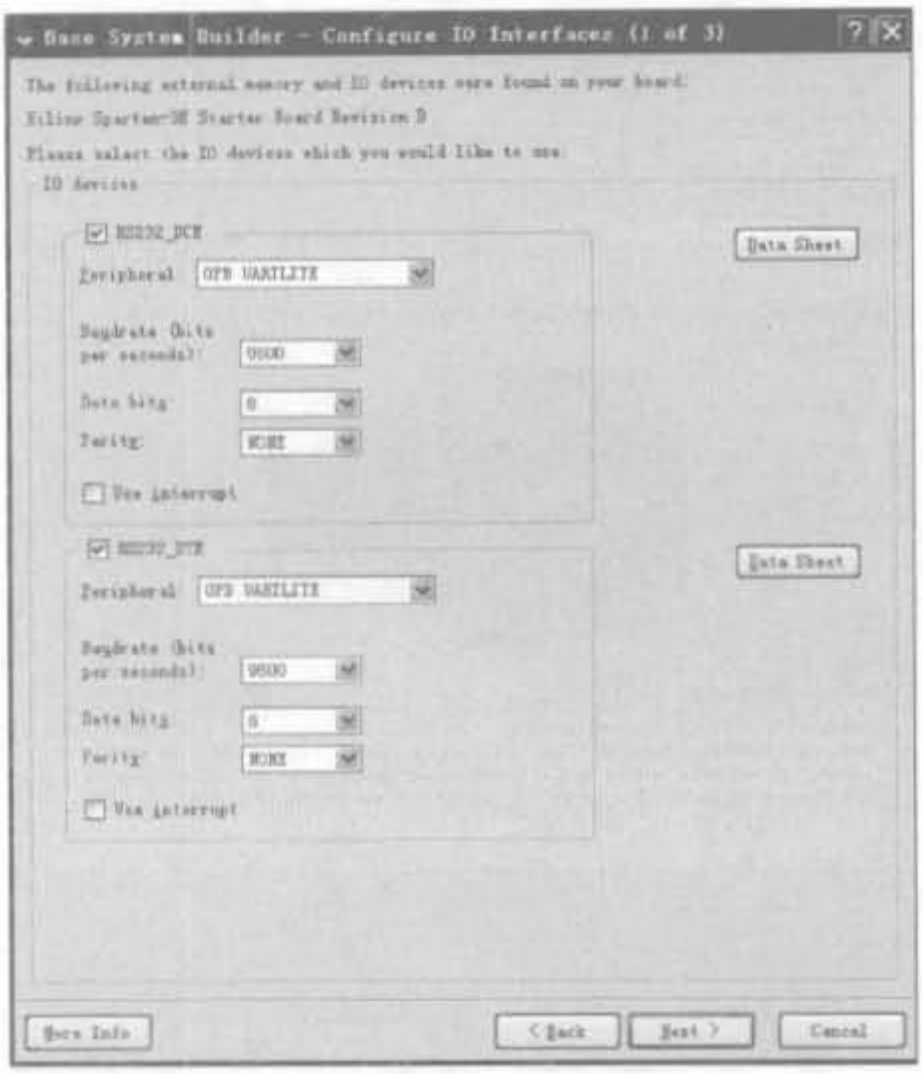


图 9-142　I/O 接口配置界面(1)

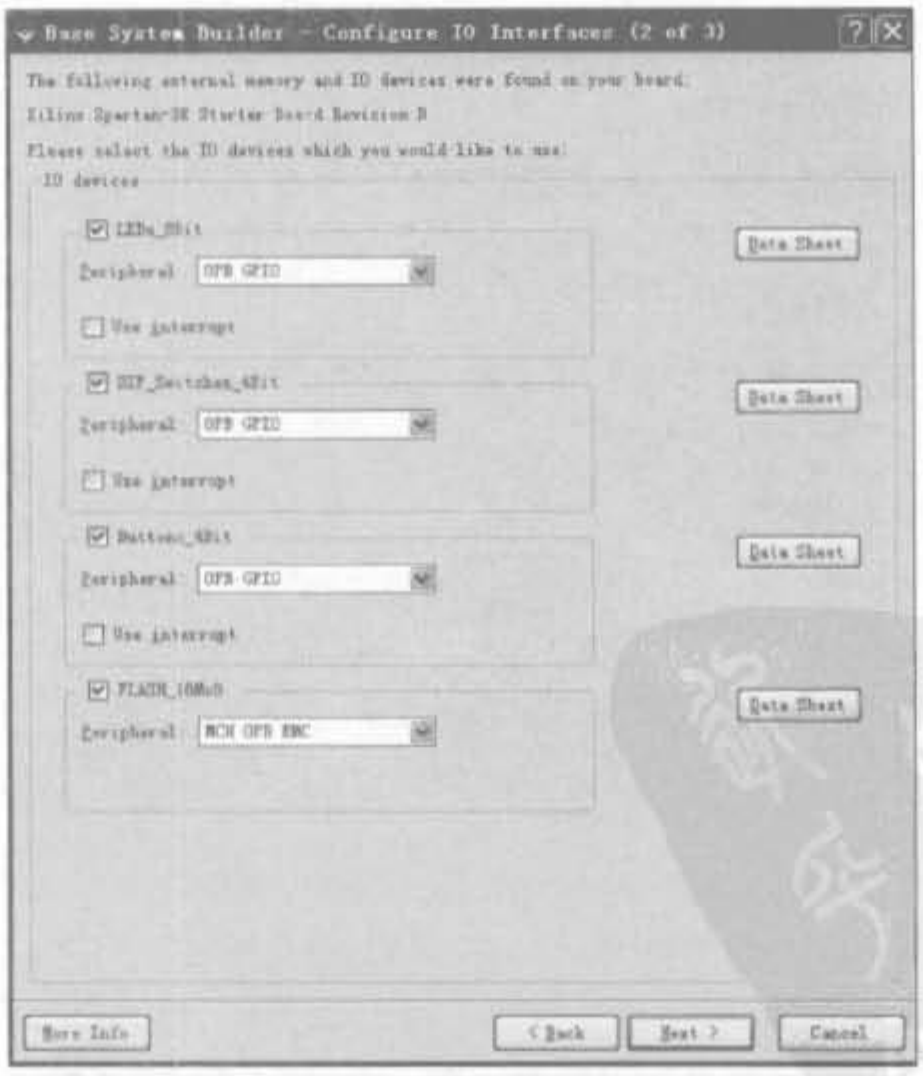


图 9-143　I/O 接口配置界面(2)

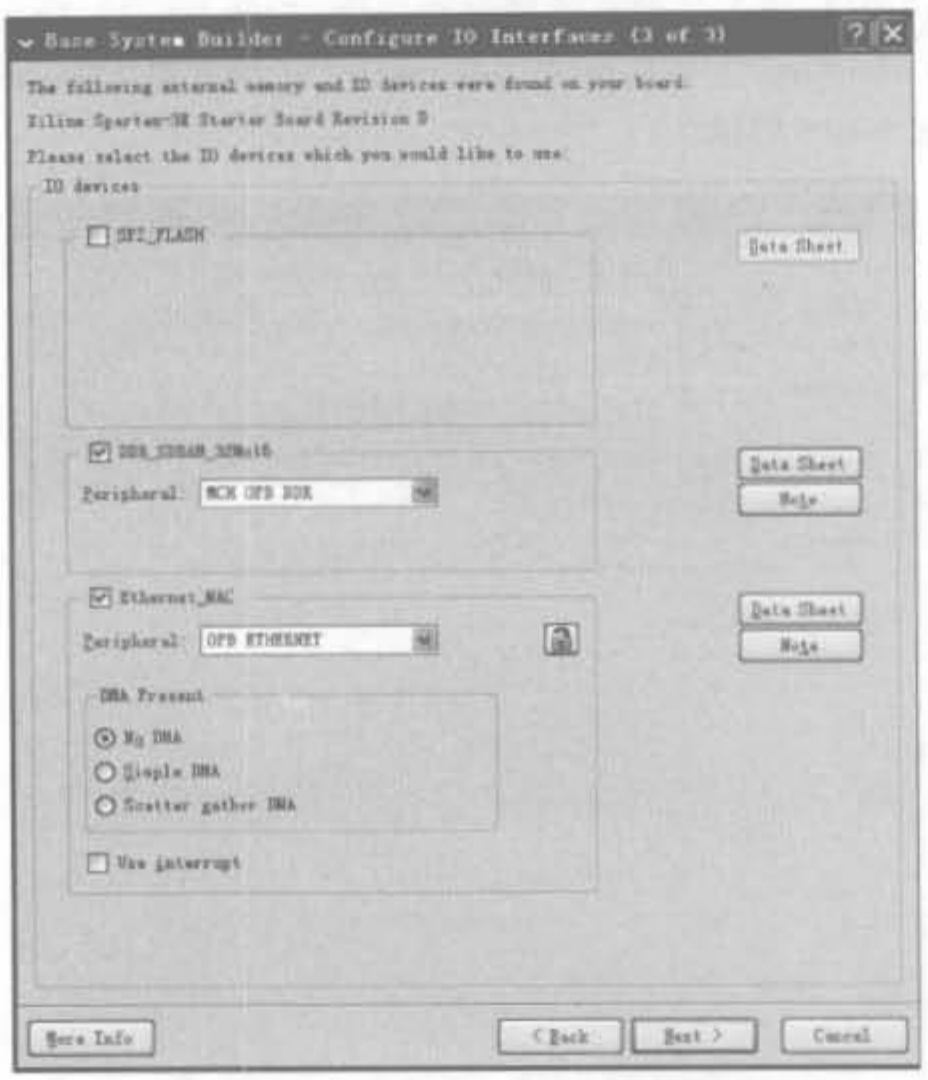


图 9-144 I/O接口配置界面(3)

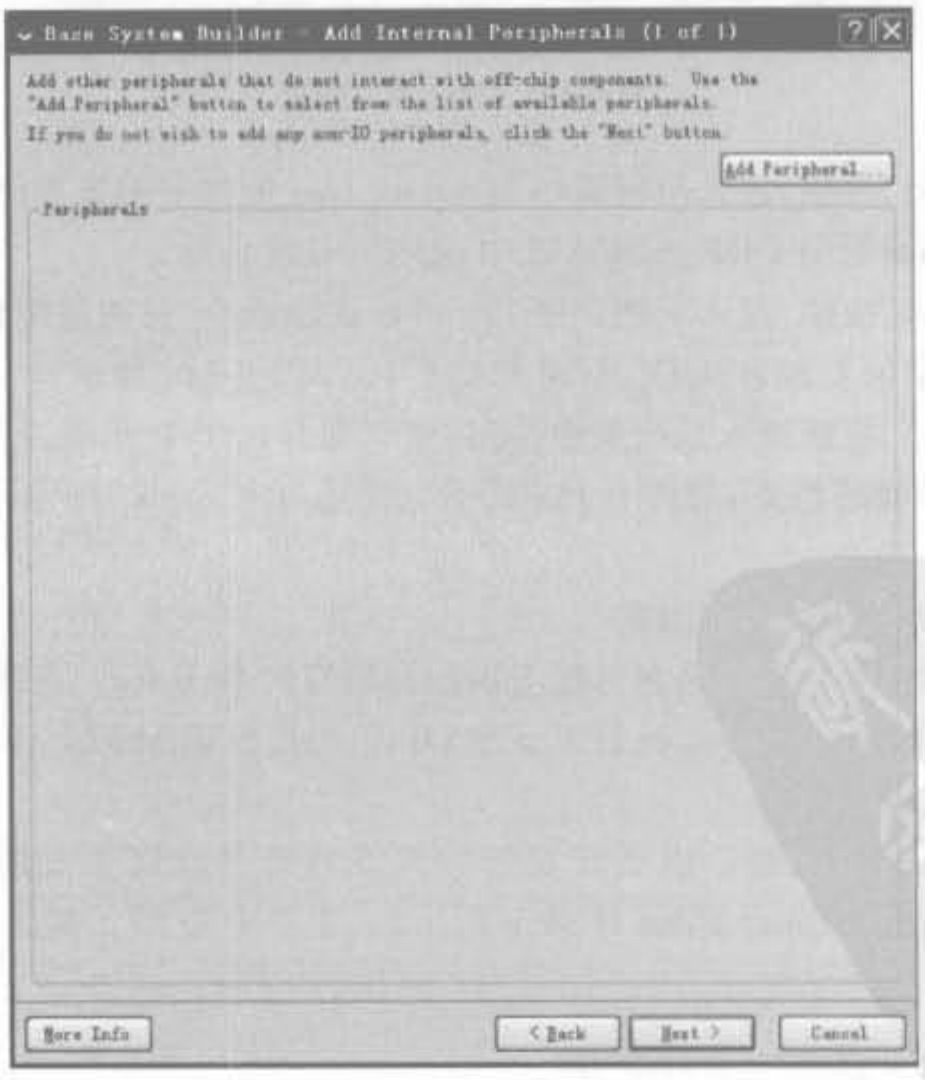


图 9-145 外设配置界面

(9) 单击“Next”按钮，进入处理器软件建立页面，选择 RS232_DCE 作为标准的输入、输出端口；在“Boot Memory”下拉框选择 ilmb_cntlr；示例程序 Memory test 和 Peripheral selftest 都选中，如图 9-146 所示。

图 9-146 软件建立界面

(10) 单击“Next”按钮，进入示例程序 Memory test 的映射配置界面，将其指令空间、数据空间、堆栈空间都映射到 DDR_SDRAM 中，如图 9-147 所示。

(11) 单击“Next”按钮，进入示例程序 Peripheral Selftest 的映射配置界面，将其指令空间、数据空间、堆栈空间都映射到 DDR_SDRAM 中，如图 9-148 所示。

(12) 单击“Next”按钮进入系统创建页面，该页面显示了处理器的系统参数、所有外设的地址以及 LMB 总线的特点，如图 9-149 所示。单击“Generate”按钮，即可完成整个系统的配置的生成。

(13) 添加 DDR SDRAM 控制器

硬件平台建成后，下一步就是在总线上添加控制器的 IP Core。直接将工程信息面板的 IP Catalog 页面 Memory Controller 目录下的 OPB DDR SDRAM Controller 拖到组建面板中，挂到 OPB 总线即可。

之后，需要生成外设地址。由于开发板的外设地址是直接生成的，直接在工程信息区域的 Project 页面的 Project Files 目录下的 mhs 文件中查阅。其关于 DDR 外设的地址为：

```
PARAMETER C_MEM0_BASEADDR = 0x44000000
PARAMETER C_MEM0_HIGHADDR = 0x47ffffff
```

图 9-147 示例程序 Memory test 映射配置界面

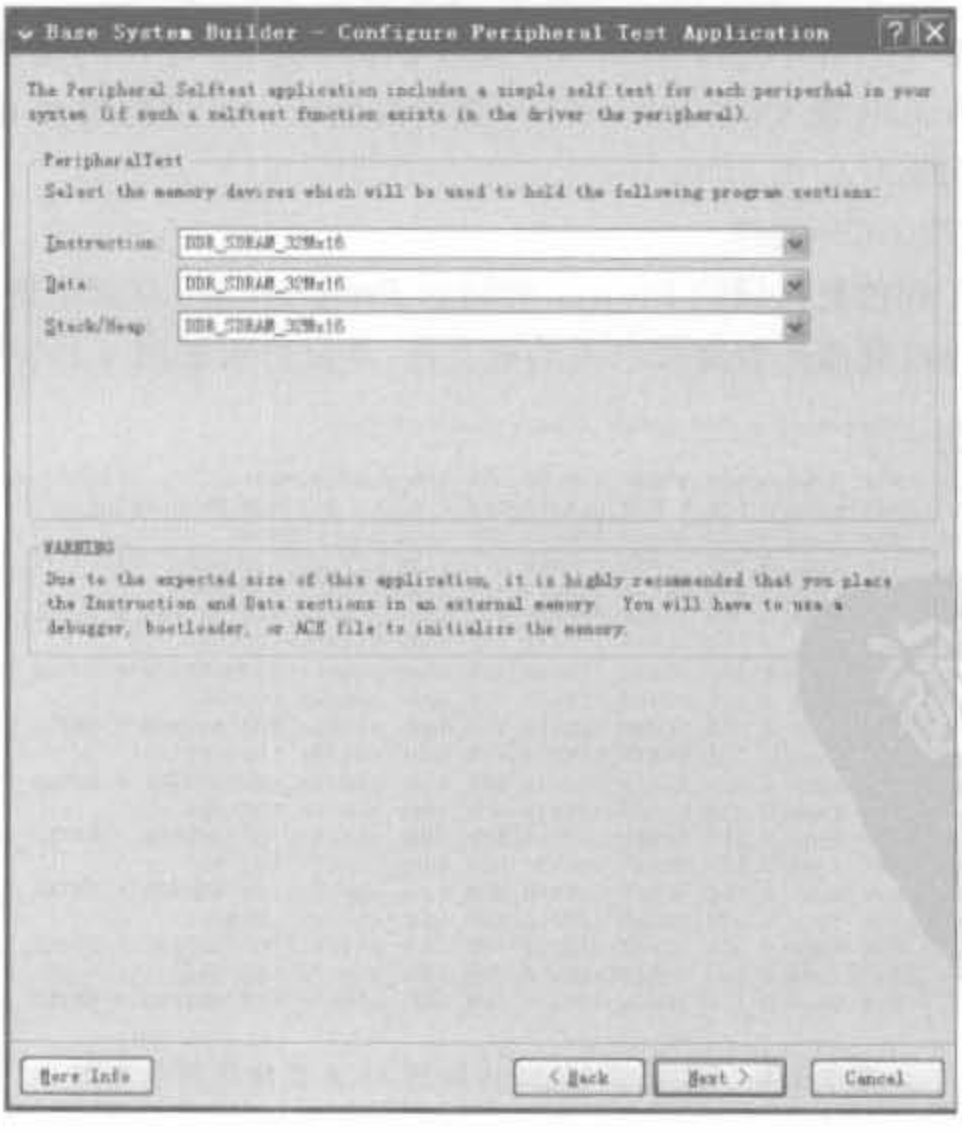


图 9-148 示例程序 Peripheral Selftest 映射配置界面

图 9-149　系统创建页面

其中，PARAMETER C_MEM0_BASEADDR 是 DDR SDRAM 外设的基地址，PARAMETER C_MEM0_HIGHADDR 是 DDR SDRAM 外设的高地址。外设的地址范围对于访问外设等操作是非常重要的。

(14) 查看/修改 UCF 文件

双击 XPS 中工程信息区域的 Project 页面的 Project Files 目录下的 UCF File，可看到 DDR SDRAM 接口控制器的管脚和电平约束文件，部分约束如图 9-150 所示。

```
151  ##### Module DDR_SDRAM_32Mx16 constraints
152
153  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clk_pin LOC=J5;
154  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clk_pin IOSTANDARD = SSTL2_I;
155  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clkn_pin LOC=J4;
156  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Clkn_pin IOSTANDARD = SSTL2_I;
157  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<0> LOC=P2;
158  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<0> IOSTANDARD = SSTL2_I;
159  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<1> LOC=N5;
160  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<1> IOSTANDARD = SSTL2_I;
161  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<2> LOC=T2;
162  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<2> IOSTANDARD = SSTL2_I;
163  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<3> LOC=N4;
164  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<3> IOSTANDARD = SSTL2_I;
165  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<4> LOC=M2;
166  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<4> IOSTANDARD = SSTL2_I;
167  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<5> LOC=M1;
168  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<5> IOSTANDARD = SSTL2_I;
169  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<6> LOC=M3;
170  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<6> IOSTANDARD = SSTL2_I;
171  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<7> LOC=M4;
172  Net fpga_0_DDR_SDRAM_32Mx16_DDR_Addr_pin<7> IOSTANDARD = SSTL2_I;
```

图 9-150　DDR SDRAM 控制器 UCF 文件局部示意图

如前所述，ISE 软件中包含了 Xilinx 开发板的板级开发包，因此会自动生成 UCF 文件。如果选择用户自定义电路板，则 UCF 文件是空的，需要用户添加，这是用户在开发自定义电路板 DDR SDRAM 控制器唯一的手动任务。需要注意的是，在 EDK 生成的 DDR SDRAM 控制器中，地址总线和数据总线下标是逆序的，如 IP Core 数据总线下标为 0 的信号线，对应着实际芯片数据总线的最高位。

(15) 综合硬件平台

修改完 UCF 文件后，可以综合硬件平台。单击“Hardware”菜单下的“Generate Netlist”命令，XPS 开始硬件平台构建。在综合时，XPS 右下角的浅蓝色圆图标一直在转动，并在输出窗口输出相关的综合指示信息。

综合完成后，再单击“Hardware”菜单下的“Generate Bitstream”命令，生成硬件配置比特流。

(16) 添加/编译测试代码

在创建工程时，选择添加示例程序 Memory test，因此 XPS 会提供存储器的测试代码。在 XPS 中工程信息区域的“Applications”页面中的“Project: TestAPP_Memory”工程上单击鼠标右键，选中“Mark to initialize BRAMs”，将其设定为片内初始化应用。

双击打开该应用 Source 目录下的 TestAPP_Memory.c，可看到 XPS 自动生成的测试代码。如果用户计算机具有串口接口，则不用修改原文件；否则需要修改原文件，添加头文件“#include "xio.h"”，并在 main()函数体的第一行加入以下简单的代码：

```
int i = 0;
int j = 0;
while(1)
{

    XIo_Out16(XPAR_DDR_SDRAM_32MX16_MEM0_BASEADDR + i,i);
    j = XIo_In16(XPAR_DDR_SDRAM_32MX16_MEM0_BASEADDR + i);
    i = i + 1;
}
```

在工程上单击鼠标右键，选择“Build Project”命令，编译软件代码。编译完成后，XPS 输出窗口会给出如图 9-151 所示的指示信息。

```
mb-size TestApp_Memory/executable.elf

   text    data     bss     dec     hex filename
   3944     332    2064    6340    18c4 TestApp_Memory/executable.elf


Done!
```

图 9-151　编译输出结果示意图

编译无误后，单击“Device Configuration”菜单下的“Updata Bitstream”命令，将软件比特流和硬件比特流合二为一。

9.6.3 DDR SDRAM 控制器的调试

完成 DDR SDRAM 控制器的软、硬件开发后，接下来的任务就是调试。本节基于没有串口计算机的场景来介绍控制器的在线调试方法。

1. 比特流下载

选择“Device Configuration”→“Download Bitstream”菜单命令，对 FPGA 芯片进行编程。执行了配置命令后，XPS 会调用 iMPACT 程序完成边界扫描和下载。配置成功后，在信息输出窗口将输出如图 9-152 所示的内容。

```
INFO:iMPACT:579 - '1': Completed downloading bit file to device.
INFO:iMPACT - '1': Checking done pin....done.
'1': Programmed successfully.
Elapsed time =      1 sec.
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------

Done!
```

图 9-152 配置成功输出信息示意图

2. 连接 XMD

(1) 单击“Debug”菜单中的“L unch XMD”命令，XPS 会启动 GDB 服务，并自动连接到硬件板。默认的连接 ID 号为“0”，TCP 端口为“1234”。单击“Debug”菜单中的“Launch Software Debugger”命令，弹出如图 9-153 所示的工程选择对话框。选择“TestApp_Memory”，单击“OK”按钮。

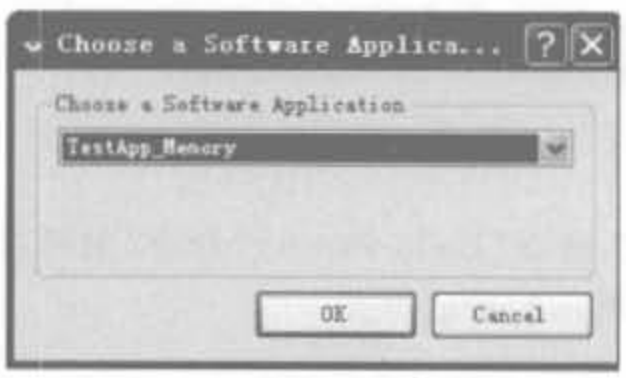


图 9-153 测试软件工程选择界面

(2) XPS 弹出用于调试的“Source Window”对话框，如图 9-154 所示。单击 图标，连接目标工程，即可弹出与目标板的连接对话框，直接单击“OK”即可进入调试模式。

3. 在线调试

通过单击 图标完成单步调试，单击 图标查看本地变量。多次单击调试内容，可发现 j 的值永远比 i 的值小 1，这和程序本意是一样的，如图 9-155 所示，说明 DDR 控制器工作完全正常。

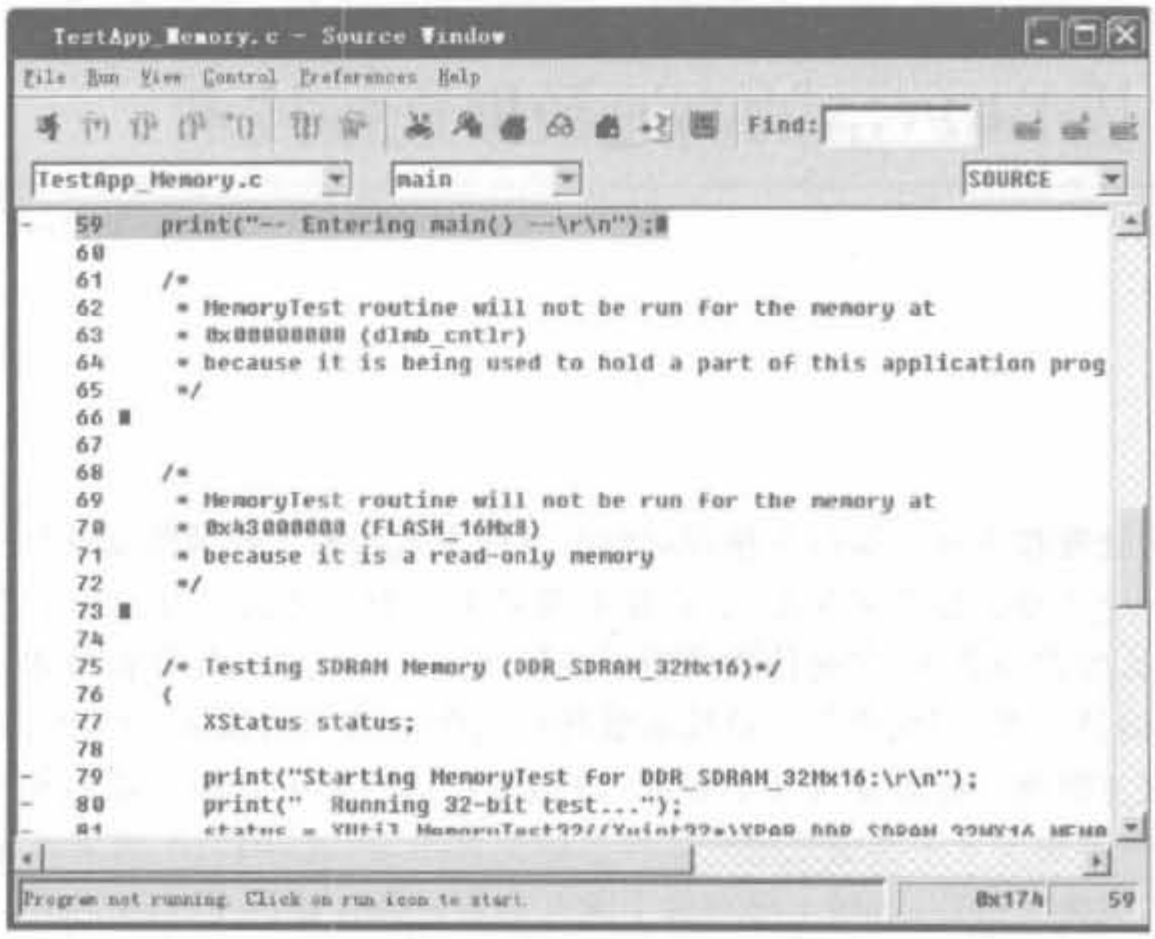


图 9-154 源代码调试界面

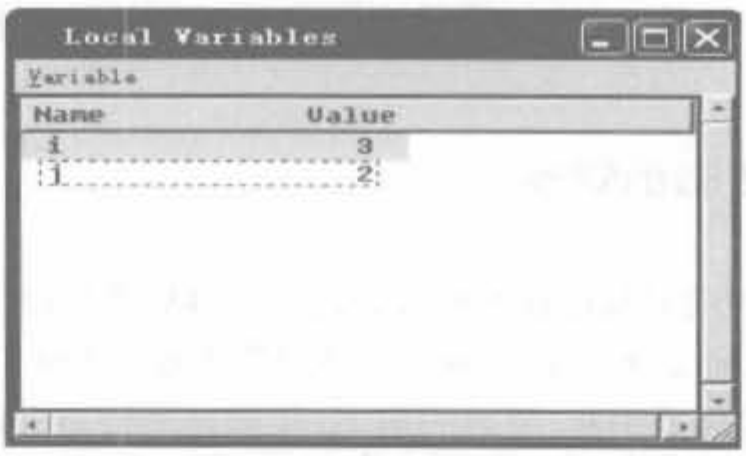


图 9-155 DDR 控制器调试结果示意图

9.7 本章小结

EDK 是 Xilinx 公司在嵌入式系统领域的完整解决方案，包括开发软件、PowerPC 硬件处理器核、Xilinx MicroBlaze 软处理器核以及进行 Xilinx 平台 FPGA 设计时所需的技术文档和 IP Core。本章首先介绍了 EDK 系统的组成，软、硬核处理器的架构；其次介绍了 EDK 软件的安装、基本设计流程和文件管理结构；接着介绍了 XPS 软件的基本操作和高级操作，包括 XPS 启动、创建新工程、加入 IP Core、定制用户外设、软件输入、系统仿真、各类在线调试手段和软件平台 SDK；最后通过一个 DDR SDRAM 控制器的实例将本章大部分内容串联起来。通过本章的学习，希望读者具备 EDK 开发的基本技能。

第10章

基于FPGA的高速数据连接技术

CHAPTER 10

现代数字处理技术和计算技术使得对复杂系统采集到的大容量数据进行实时处理成为可能。对于一个大型的数据采集系统，往往需要采集几十个甚至成百上千个参量的实时数据，对这样庞大数据的高速、实时传输成为了关键问题。传统的并性传输技术已接近理论上限，但仍不能满足需求，因此串行传输技术重新返回到高速传输领域，并引领新一代吉比特传输技术。高速数据传输对硬件要求很高（包括芯片接口和电路板走线），相应的 ASIC 电路价格昂贵、种类稀少且不能满足用户多种多样的需求，因此，FPGA 成为高速数据连接领域最为合适的载体。目前，市场上已有数十款支持吉比特传输的 FPGA 芯片。本节主要介绍基于 Xilinx FPGA 的新型串行高速传输技术及其相关应用（吉比特以太网）的解决方案。

10.1 高速数据连接功能简介

10.1.1 高速数据传输的背景

由于现代通信以及各类多媒体技术对带宽的需求迅猛增长，促使一系列基于差分、源同步、时钟数据恢复（Clock and Data Recovery，CDR）等先进技术的互连方式应运而生。在传统设计中，单端互连方式易受干扰、噪声的影响，传输速率最高只能达到 200Mb/s/通道～250Mb/s/通道；在更高速率的接口设计中，多采用包含有源同步时钟的差分串行传输方式（如 LVDS、LVPECL 等），但在传输过程中时钟与数据分别发送，各信号瞬时抖动不一致，破坏了接收数据与时钟之间的定时关系，因而传输速率很难超越 1Gb/s/通道。因此迫切需要新的高速数据传输技术。

在目前系统级互连速率已达到吉比特每秒的设计中，先进的高速串行技术迅速取代传统的并行技术，成为业界的主流。高速串行技术不仅能够带来更高的性能、更低的成本和更简化的设计，克服了并行的速度瓶颈，还节省了 I/O 资源，使印制板的布线更简单。因此，被越来越广泛地应用于各种系统设计中，包括 PC、消费电子、海量存储器、服务器、通信网络、工业计算和控制、测试设备等。高速串行传输一般采用差分线，迄今业界已经发展出了多种串行系统接口标准，例如千兆以太网、万兆以太网、PCI-Express、串行 Rapid I/O、串行 ATA 等。

10.1.2 Xilinx 公司高速连接功能的解决方案

基于高速的需求和传统技术的弊端，Xilinx 公司在 Virtex-2 Pro 以及更高系列的部分 FPGA 内部集成了能实现高速数据收发 Rocket I/O 模块，采用 CML（CurrentMode Logic）、CDR、线路编码（8b/10b）和预加重等技术的 Rocket I/O 硬核模块，可极大地减小时钟扭曲、信号衰减和线路噪声对接收性能的影响，从而使传输速率进一步提高，最高可达 10Gb/s 以上，可用于实现吉比特以太网、PCI-Express 等常用接口。

除了底层的物理层技术，Xilinx 还提供带 32 bit LocalLink 用户接口的 Aurora 协议引擎参考设计。Aurora 协议是为私有上层协议或标准上层协议提供透明接口的串行互连协议，它允许任何数据分组通过 Aurora 协议封装，并在芯片间、电路板间甚至机箱间传输。Aurora 链路层协议在物理层采用千兆位串行技术，每物理通道的传输波特率可从 622Mb/s 扩展到 3.125Gb/s。Aurora 还可将 1～16 个物理通道绑定在一起，形成一个 16 个通道绑定而成的虚拟链路，提供 50Gb/s 的传输波特率和最大 40Gb/s 的全双工数据传输速率。Aurora 可支持广泛的应用范围，如路由器和交换机、远程接入交换机、HDTV 广播系统、分布式服务器和存储子系统。在协议中，每个高速串行链接被称为“弄”。

协议引擎通过与高速收发器配合，可创建带 LocalLink 用户接口逻辑的串/并、并/串收发器。通过这一串行接口方案，用户无需自己设计有关串行接口所涉及的编/解码、同步、速率匹配等问题。用户接口部分包含了所有必要的信号，如协议引擎的状态信号等。

Xilinx 通过高品质的技术支持材料来支持其先进的芯片产品，这些材料包括广泛的知识产权核、参考设计、模拟电路模块、信号完整性（SI）设计套件、数字仿真的质量行为模型等。此外，Xilinx 还提供了众多设计服务、开发平台以及最佳的 FPGA 实现工具，可确保用户的所有设计需求都能获得最佳产品和技术支持。

10.2 实现吉比特高速串行 I/O 的相关技术

目前，串行 I/O 的最高速率已经超过 10Gb/s，因此，吉比特串行 I/O 泛指传输速率超过 1Gb/s 的串行收发技术。无论采用何种方案来实现吉比特串行 I/O，都具备底层编码和时钟修正等一系列必备技术。本节主要介绍吉比特串行 I/O 的普遍特点和通用技术。

10.2.1 吉比特高速串行 I/O 的特点和应用

吉比特收发器（MGT）是吉比特级串行器/解串器（SERDES）的别名。

1. 优点

（1）速度高。高速串行 I/O 的线速可超过吉比特，甚至数十吉位。而并行传输线速不可能超越吉比特。吉比特串行 I/O 的主要优势是速度。在从片内/片外、板内/板外或盒内/盒外获取数据时，没有任何技术可以超越高速串行链路。高速串行链路的线速范围为 1Gb/s～12Gb/s，有效负载范围为 0.8Gb～10Gb。

(2) 节省管脚数。将大量数据传入/出芯片或电路板时所遇到的第一个问题是管脚数，电路板设计时间和成本会随着管脚数的增加而急剧增加。在大数据量应用下，串行 I/O 可节省大量的管脚(在低速以及小数据量应用中，MGT 比传统并行模式需要更多的电源和接地管脚)。

(3) 简化同步转换输出。采用单端并行总线时，设计者应考虑同步转换输出(SSO，即大量寄存器的值在某一时刻同时翻转，会对电源和地平面产生一定的影响，甚至影响到器件内部时钟和逻辑的正常工作)。如果出现太多的同步转换，触地反弹会产生大量噪声。设计者还可以在所有 I/O 上都使用差分信号处理技术，以此来消除 SSO 问题，但是这样做会使管脚数翻倍。如果数据流需求比较适中，设计者可以使用具有适当管脚数的并行接口。

(4) EMI 指标优。经验表明：时钟越快，放射测试就越难进行，因此，吉比特设计的 EMI 测试看起来是不可能实现的。但是，通常高速串行链路的辐射量比以较低速度工作的大型总线低。这是因为运行时的吉比特链路需要出色的信号完整性，正如经典论断"辐射问题实际上就是信号完整性问题"所言，因此吉比特串行 I/O 具有更好的 EMI 指标。

(5) 成本低。采用 MGT 通常会降低系统总成本。连接器采用较小、较经济的封装时，管脚数较少，电路板设计也更简单。

(6) 预设协议。采用 MGT 的另一个好处是可以使用预先定义好的协议和接口标准。如 Xilinx 提供了从 Aurora 到 XAUI 的多种协议，满足不同的用户需求。

2. 缺点

吉比特高速串行 I/O 的最大缺点在于对信号完整性的严格要求，而且阻抗控制的 PC(印刷电路)板、高速连接器和电缆的费用较高。因此，必须处理数字仿真中的复杂性和时基较小的问题。并且在利用预设协议的时候，必须为集成过程计划时间，以及为协议的开销安排额外的逻辑电路或 CPU 时钟周期。

3. 应用范围

起初，吉比特级串行器/解串器(SERDES)仅局限于用在电信行业和少数缝隙市场(如广播视频)。如今，MGT 应用出现在电子行业的各个角落——军事、医疗、网络、视频、通信等等。MGT 也可以用于背板或机箱之间的 PCB 上。对于电子行业的发展前景而言，MGT 至关重要。下面是采用吉比特级 SERDES 的行业标准示例：

- 光纤通道(FC)
- PCI Express
- Rapid I/O 串行
- 先进的交换互连(Advanced Switching Interface)
- 串行 ATA
- 1Gb/s 以太网
- 10Gb/s 以太网(XAUI)
- Infiniband 1X、4X、12X

吉比特级通信似乎强加了一些苛刻限制，串行设计者必须考虑信号完整性、较小的时基以及可能出现的对额外门电路和 CPU 周期的需求。但是，在盒间以及芯片间通信中采用吉比特级技术的优势远远超过了那些可以察觉到的缺点，例如高速、管脚数少、低 EMI 和低

成本等，这些都使它成为了众多高速设计的理想之选，并保证了其在未来通信系统中得到广泛的使用。

10.2.2　吉比特串行 I/O 系统的组成

吉比特串行传输是一种通用的传输标准，虽然不同 FPGA 厂家的模块和组件名不同，但其关键技术都具备下列共同点。

1. 系统整体结构

吉比特串行传输的系统整体结构如图 10-1 所示。下面对其中的主要模块进行简要介绍。

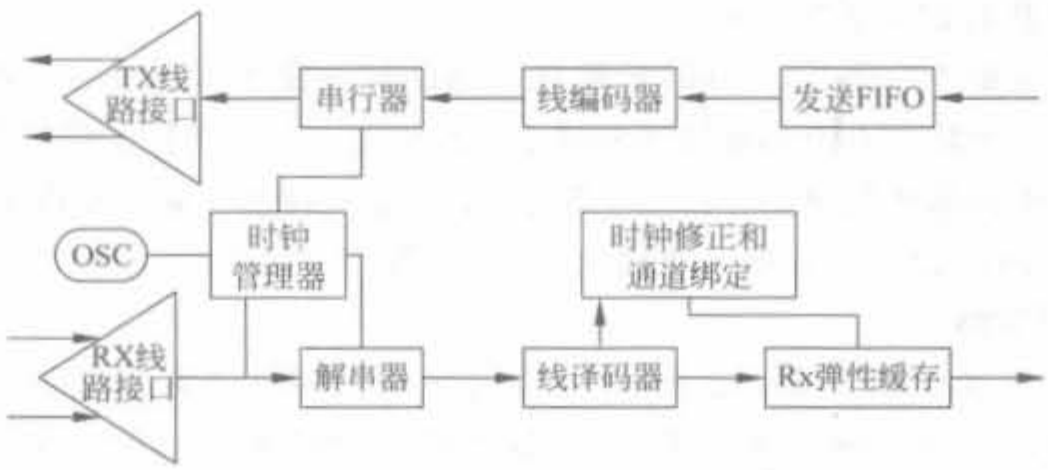


图 10-1　吉比特高速串行 I/O 的系统结构示意图

(1) 串行器：将速率为 y 的 n 位宽并行数据转变成速率为 $n\times y$ 的串行数据。

(2) 解串器：将速率为 $n\times y$ 的串行数据转变成速率为 y 的 n 位宽并行数据。

(3) Rx(接收)对齐：将接收的数据对齐到合适的字边界。可以使用不同的方法，从自动检测和对齐特殊的预留比特序列(通常也称作 comma 字符)，到用户控制的比特调整。

(4) 时钟管理器：管理各种时钟操作，包括时钟倍频、时钟分频和时钟恢复。

(5) 发送 FIFO(先进先出)：在输入数据发送之前，暂时保存数据。

(6) 接收 FIFO：在接收数据被提取之前，暂时保存数据。在需要时钟修正的系统中，接收 FIFO 是必须的。

(7) 接收线路接口：模拟接收电路，包括差分接收器，还可能包括有源或者无源均衡电路。

(8) 发送线路接口：模拟发送电路，可以支持多种驱动负荷。通常还带有转换的预加重部分。

(9) 线路编码器：将数据编码成适应不同线路的格式。编码器通常会消除长的无转变位的序列，还可以平衡数据中 0、1 的出现次数。需要注意的是，线路编码器是一个可选模块，某些 SERDES 可能没有。

(10) 线路译码器：将线路上的编码数据分解成原始数据(这是一个可选模块，编码可能在 SERDES 外完成)。

(11) 时钟修正和通道绑定：修正发送时钟和接收时钟之间的偏差，也可实现多通道间的歪斜修正(通道绑定是可选的，并不一定包含在 SERDES 中)。

其他可能包括的功能模块有：循环冗余检测(CRC)码生成器、CRC 检测器、多种编码和解码(4b/5b、8b/10b、64b/66b)、可调的扰码器、各种对齐和菊花链选项、可配置的时钟前端和后端以及不同等级的自环。

2. 参考时钟的要求

1) 时钟精度

吉比特级收发器的输入时钟或是参考时钟的规格定义是非常严格的，其中包含非常严格的频率要求，通常用每百万次容许频率错误的单位 PPM 来定义。抖动要求也是十分严格的，通常用时间(皮秒)或者时间间隔(UI)定义。下面给出这些相关定义：

- PPM：百万分之一；用来描述非常小的比率。
- UI：时间间隔，等价于一个符号的时间长度。例如，0.2UI=20%的符号时间。
- 抖动：理想传输位置的偏差。

如此严格的规定才使得 PLL 和时钟提取电路能够正常工作。通常系统的每一个印刷电路板都需要有一个精确石英晶体振荡器供 MGT 使用。这些晶体振荡器的精确度比大多数用在数字系统中的晶体振荡器要高一个级别，价格也要高出一截。在很多情况下，一般的时钟发生芯片和 PLL 因为带有很大的抖动，而不能用于 MGT。

2) 时钟修正策略

传输时钟有非常严格的抖动要求，所以吉比特 SERDES 通常不能将恢复时钟作为传输时钟。每一个 PCB 集合都有唯一的振荡器和唯一的频率。如果两个 1GHz 的振荡器仅仅有 1PPM 的频差，同时我们提供 1/20 的参考时钟，则数据流的时钟每秒钟可能会增加或者缺失 20 000 个周期。因此，在 8b/10b 编码的系统中，每秒将会额外增加或者损失 2 万个符号。

大多数 SERDES 都有时钟修正选项。时钟修正需要使用唯一的符号或者符号序列，它们在数据流中是不会出现的。因为时钟修正是对齐的后续处理，所以可以比较容易地通过保留一个 K 字符，或者一组有序的 K 字符，或者一个时钟修正数据序列来实现。时钟修正进行的频数必须足够多，从而可以通过丢弃或者重复来补偿时钟的差异。当然，有些系统并不需要时钟修正。例如，相同的参考时钟和相同的速率意味着不需要进行时钟修正。同样，如果所有接收电路的时钟都来自恢复时钟，那么时钟修正也是不需要的。如果 FIFO 的写入速率和读出速率相等，也没有必要进行时钟修正。如果所有的传输参考时钟都是通过一个外部的 PLL 锁定在一个公共的参考频率上，那么也不需要时钟修正。

3. 线路编码机制

线路编码机制将输入的原始数据转变成接收器可以接收的格式。同时，线路编码机制还必须保证有足够的切换提供给时钟恢复电路。编码器还提供一种将数据对齐到字的方法，同时线路可以保持良好的直流平衡。线路编码机制也可选择用来实现时钟修正、块同步、通道绑定和将带宽划分到子通道。线路编码机制主要有两种：数值查找机制和自修改数据流或扰码器机制。目前常用的有 8b/10b 编码、4b/5b 编码以及扰码。

1) 8b/10b 编码

8b/10b 编码机制是由 IBM 开发的，已经被广泛采用。8b/10b 编码机制是 Infiniband、吉比特以太网、FiberChannel 以及 XAUI 10G 以太网接口采用的编码机制。它是一种数值

查找类型的编码机制，可将 8 位的字转化为 10 位符号。这些符号可以保证有足够的跳变用于时钟恢复。8b/10b 编码具有良好的直流平衡特性，通过“运行不一致性”的方法来实现，即只使用有相同个数 0 和 1 的符号，但这会限制符号的数量。同时，8b/10b 中的 comma 字符（用于表示对齐序列的一个或两个符号）可辅助数据对齐。

8b/10b 机制能带来字对齐、时钟修正机制、通道绑定机制和子通道生成等功能，其唯一的缺陷是开销。为了获得 2.5Gbit 的带宽，它需要 3.125Gb/s 的线路速率。从减小开销的角度讲，下面所讲述的扰码技术可以很容易地解决时钟发送和直流偏置问题，并且不需要额外的带宽。

2）4b/5b 编码

4b/5b 和 8b/10b 是类似的，但是要简单些，将 4 个比特编码成 5 个比特。4b/5b 的控制字符要少一些，但不能处理直流平衡和不一致性问题。由于编码开销相同但是功能比较少，4b/5b 编码机制并不经常使用。它的最大优势是设计的尺寸，不过随着逻辑门价格的降低，这个优势也不再明显。目前，4b/5b 仍用在各种低速标准中，包括低速率版本的光纤通路、音频标准 AES-10 以及多通道数字音频复接标准 MADI 接口中。

3）扰码

扰码是一种将数据重新排列或者进行编码，以使其随机化的方法，但要求必须能够通过解扰恢复。加扰的目的就是打乱长的连 0 和长的连 1 序列，将数据随机化。一般将那些在解扰时不需要额外对齐信息的扰码称作自同步码。扰码发生器通常由移位寄存器组成，所占用的硬件资源很少。

扰码器消除了长连 0 和长连 1 序列，以及其他会对接收器的接收能力有负面影响的序列，但并不能取代 8b/10b 编码。在实际中，由于存在不允许的数值，所以需要设计数据流中不能出现连 0 或连 1 的长度。长的连 0、连 1 会被扰码器打乱，并在解扰时进行恢复。接收数据流的解扰逻辑在数据流中搜寻这些符号并对齐数据。

4. 接收和发送缓冲器

接收和发送缓冲器，是吉比特级收发器的主要数字接口，通常是高速 FIFO。发送端通常有一个小型的 FIFO，它要求读取和写入的时钟是等时同步的（频率匹配，但相位不一定匹配）。如果接收和发送的选通信号不是工作在精确的相同的频率，则通常需要使用一个较大的 FIFO，并持续检测 FIFO 的当前状态。如果 FIFO 被不断地填充，将最终导致溢出。在这种情况下，必须在输入数据流中检测 idle 符号。如果检测到 idle 符号，则不把 idle 符号写入 FIFO；反过来，如果 FIFO 运行较慢，则在输出数据流会出现 idle 符号，数据被传送给用户。此时，写指针保持不动，不断重复 idle 符号。相对于发送缓冲器而言，MGT 内建的接收 FIFO 通常需要有更深层次的考虑。它的主要目的是为了实现时钟修正和通道绑定。

5. 线路均衡

线路均衡主要用于补偿由频率不同而引起的阻抗/衰减差异。均衡器有很多种形式，但总体上可以分为有源和无源两种。均衡器通常包含在 SERDES 的模拟前端，或者作为系统的一个独立部分。

1）均衡技术简介

无源均衡器是无源电路，其频率响应可以补偿传输衰减。它也可以看作一个滤波器，将

传输线所使用的各个频率通过，而将传输线没有使用的其他频率滤除，那么整体的频率响应就会变得平坦许多。有源均衡器可以认为是依赖频率的放大器/衰减器。

有源均衡器主要有两种：固定形式有源均衡器和自适应有源均衡器。对于任意的输入数据流，固定形式有源均衡器的频率响应都是一样的。固定形式均衡器比较适合于不变系统中，例如芯片到芯片、平衡化的背板系统以及固定长度电缆的系统。自适应均衡器要复杂得多，自适应均衡器需要分析输入信号并检测哪些频率在传输通道中被削弱。在该均衡器中，测量和调节是以闭环形式实现的。自适应均衡器的频率响应取决于输入的比特流，它通常和特殊形式的线路编码机制协同工作。自适应均衡器对于可变通道的链路来说是最合适的。可变通道可以是可变的电缆长度，或是显著的位置依赖的背板系统。

2）预加重/去加重技术

预加重是一种非常普遍的均衡技术。在发送端，通过增加一串相同符号中首位符号的输出级，降低随后符号的输出级，来预先抬高输出信号频谱中的高频分量，以补偿传输通道的低通滤波效应。这样，在接收端就可以得到相对均衡的眼图，使接收器能够准确地接收和恢复信号。这种技术对于高数据速率的设计而言，简单而有效。发送预加重技术主要是通过采用 FIR 多抽头的有限冲激响应均衡滤波器来实现的。输入到滤波器中的是当前、过去和将来要发送的数据位。滤波的系数取决于通道特性，最佳的滤波长度取决于影响当前正在发送数据的比特数量。在具体实现中，一般采用二抽头 FIR，进行双电流控制。在电路转换时，为了克服传输通道的滤波效应，发送器分发额外的动态电流。在转换后，则提供一个更低的驱动电流。不同的通道损耗补偿需要不同程度的信号预加重，因此，在发送器设计中，预加重功能一般是编程可控的。为了和发送端预加重相匹配，在接收端必须有去加重。

6. 数据包的概念

通过吉比特串行链路传输的数据大都是嵌入在某种类型的数据包中的。包是一种确切定义的字节集合，包括头部、数据和尾部。如果系统通过包来完成时钟修正，发送特殊的比特序列或者 comma 字符。时钟修正序列常常是比较理想的字符，comma 字符是指示帧的开始和结束的天然标识。在数据中加入有序集合用于指示包的开始、结束以及包的特殊类型之后，就构成了简单而高速的传输通道。其中，空闲符号(Idle)或序列是包的概念的另一要点。如果没有信息需要发送，则发送 idle 符号，从而保证数据的连续传输，并使其保持对齐。

10.2.3 吉比特串行 I/O 的设计要点

解决工程问题的关键在于充分的理解。在设计吉比特级收发器时，设计人员面临的挑战包括理解收发器协议、信号完整性、阻抗和功率要求、屏蔽性要求、印刷电路板(PCB)设计要求以及连接器和电缆的选择要求。

1. 电源

电源传送也是使用吉比特级收发器时需要考虑的重要因素。多数的 MGT 都需要多个电源供电。典型的电源包括 RX 模拟电源、TX 模拟电源、模拟地、RX 终端电压、TX 终端电压、数字电源以及数字地。所有的模拟发送和接收电源以及相关的模拟地必须是极其干净的，

这一点十分重要。所以,MGT 制造商通常都会使用特定电路。因此至少要求每个电压值都有各自的模拟电压校准器(如果不是每个 MGT 都有各自独立的校准器),并且要求使用无源的电源滤波器(由一个电容和一个铁氧体磁珠组成)。典型的电源滤波电路如图 10-2 所示。

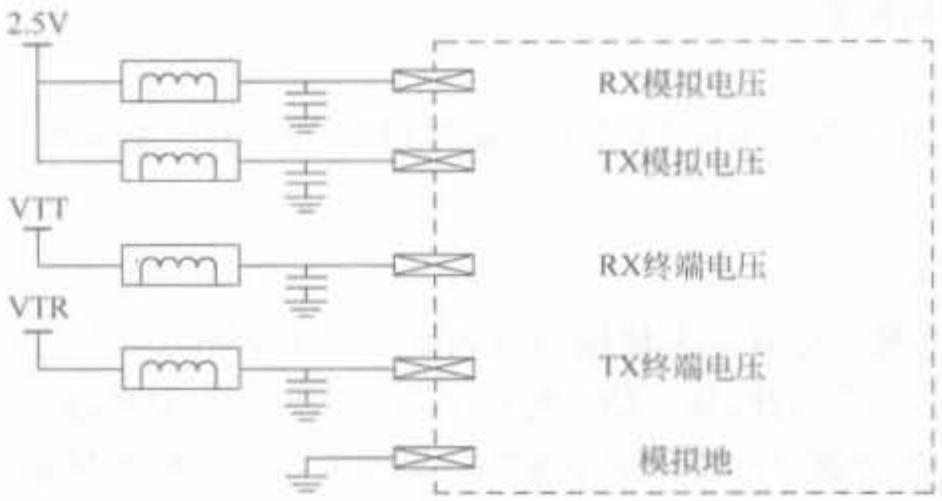


图 10-2 MGT 电源滤波电路

某些 MGT 将电容包含在封装的内部,此时通常只需要铁氧体磁珠。如果制造商建议使用特定电路,最好遵从其建议。原因之一,是在公共部分配置了多个 MGT 的情况下,通常只需要一个单独的线性调整器即可。滤波器电路可以防止电源噪声进入 MGT,同时它可以防止来自某个 MGT 的噪声滤进其他 MGT。此滤波器既是输入滤波器,也是输出滤波器。有时,制造商会基于自己所需的输出滤波性能,在输入滤波器和输出滤波器性能间做出折衷。

2. 匹配电阻

实现信号完整性的第一步就是在差分传输线上传送这些信号。根据通常的定义,传输线都有一定的固定阻抗。实际上,阻抗值并不是恒定的,而是变化的。这个问题在下面的几种情况下尤其突出:信号由一层转移到另一层时,信号遇到元件的焊盘时,或信号通过连接器或电缆时。当运行在吉比特级速率范围内,很小阻抗增加都会是潜在的问题。吉比特级链路需要使用阻抗无限制通道,否则将无法工作。

用户需要对传输通道进行模拟,并在布线之前使用 CAD 的信号完整性工具来最终确定连接器和电缆。在获取初始原型时,需要用时域反射测量法(TDR)来检验通道的阻抗。100Ω 和 50Ω 的传输线是最常用的。部分收发器可以适配两种传输线,而部分收发器可能只支持其中的一种。对于 10Gb/s 范围的应用,50Ω 显然是最通常的选择。如果收发器同时支持 100Ω 和 50Ω,那么连接器和电缆的选择应当慎重考虑。

3. 印刷电路板的设计

对于优秀的 PCB 设计者来说,设计用于吉比特级操作的 PCB 也会是一个挑战。设计者需要注意:差分线路必须匹配,阻抗受限的差分线路的几何形状必须随着层数的增加而相应变化,电源分配也必须严格分析。因为可能存在成千上万的独立的设计折衷和决定,所以全面列出所面临的问题可能会有所帮助,主要包括如下几个方面:

- 材料选择
- 叠层结构/板厚度
- 电源层和地层

- 差分线路对
- 差分线路的宽度和间隔
- 过孔
- 线路对之间的间隔
- 电源布局

在实际设计中，必须参考不同FPGA厂家对PCB设计的详细说明文档，否则，系统很可能不能正常工作。

4. 数字设计部分

吉比特级链路的模拟仿真需求使用户进入了一个崭新的EDA工具世界，而其数字部分的影响要小得多。尽管如此，数字仿真时还是有几个事项需要考虑。

(1) MGT行为模型都是以特殊的加密形式出现的。这些模型都是复杂的内核，而且都是非常有价值的知识产权(Intellectual Property，IP)核。所以，厂商为了保护他们的知识产权，通常只会以IP-safe的格式发布这些模型。最流行的格式称作smart模型或者swift。通常，模型经过加密后，仿真器可以读取模型，但是用户不能。模型内部的节点和层次对于用户来说是不可见的；用户只能看见模型的输入和输出。

(2) 数字仿真。MGT数字仿真的另一个问题是仿真速度。通常的数字电路大多数工作在100MHz～300MHz的范围，因此，需要把仿真的时标调整到几个纳秒。但是，如果添加一个MGT的线速度模型，信号速度就会比之前最快的信号还要快20倍以上。MGT模型都有一个并行的输入和输出端口，因此如果在大多数的测试台中使用这些措施，并创建一个实际运行在全数据速率的小型测试组件，则MGT对全局验证时间的影响将会大大减小。

10.3 基于Rocket I/O高速串行技术

Rocket I/O是Xilinx FPGA芯片内部集成的可编程高速串行收发器，其收发体系由Rocket I/O收发器和收发控制Aurora协议组成，其中Aurora协议封装在Aurora IP核内。本节介绍Rocket I/O的原理和使用方法。

10.3.1 Rocket I/O技术简介

Rocket I/O是一种高速的串行收发器，采用两对差分对来进行数据的发送和接收，可以实现两个单工或一对全双工的数据传输。Rocket I/O支持622Mb/s～3.75Gb/s的全双工传输速率，还具有8b/10b编解码(平衡编码)、时钟生成及恢复等功能，可以理想地适用于芯片之间或背板的高速串行数据传输。Aurora协议是为专有上层协议或行业标准的上层协议提供透明接口的第一款串行互连协议，可用于高速线性通路之间的点到点串行数据传输，同时其可扩展的带宽，为系统设计人员提供了所需要的灵活性。

Rocket I/O收发器发送和接收串行差分信号，工作于2.5V的直流电压下，采用CML(Current Mode Logic)模式，内部带有50Ω或75Ω的匹配电阻。此外，Rocket I/O采用了预加重技术，可以补偿传播媒质中的高频损耗，极大地降低了共模信噪比和线路衰减。由香

农公式其中 SNR 为信噪比。

$$C = W \cdot \log_2(1 + \mathrm{SNR})$$

可以得到，当信道容量一定时，信道带宽 W 的增加会造成信噪比下降。由于 Rocket I/O 单路传输速率最高可达 3.75Gb/s，因此可允许很低的信噪比。总体来讲，Rocket I/O 的显著特点包括：

- 速率范围介于 100Mb/s～3.75Gb/s 之间；
- 业内最低的功耗：在 3.2Gb/s 下，每个通道的功率均低于 100mW；
- 可在单个 FPGA 中实现多个协议(标准的和定制的)；
- 设计用来与 Virtex-5 LXT 和 SXT 平台 FPGA 内的 PCI Express 端点与三态以太网 MAC 模块一起使用；
- 符合芯片到芯片、背板与光学器件接口的常见标准和协议；
- 先进的 Tx/Rx 均衡技术，可以驱动背板和其他通道；
- 内置式 PRBS 发生器/检验器可以加速调试；
- 在 Virtex-5 LXT 平台器件中的收发器多达 24 个。

10.3.2　Aurora 协议

1. Aurora 协议简介

Aurora 是一个相对简单的协议，只控制链路层和物理层。Aurora 的设计理念是使其他高层协议，例如 TCP/IP 和以太网，可以很容易地运行在 Aurora 之上。Aurora 协议使用 1 个或多个高速的串行通道构成更高速的通路，如图 10-3 所示。

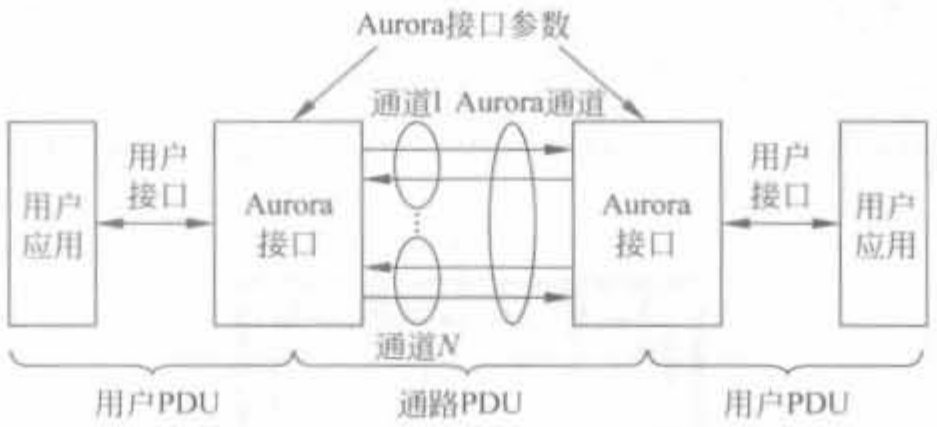


图 10-3　Aurora 链路结构示意图

Aurora 不仅定义了物理接口，而且定义了包结构、嵌入其他协议包的推荐程序、数据提取和流量控制。协议中定义了有效链路的初始化程序，还描述了禁止使用发生过量错误的链路的相关程序。由于协议中没有寻址时序，所以不支持交换。协议中也没有定义错误检测、重传或有效载荷的纠错。此协议是由 Xilinx 开发的，并且无限制地开放给公众自由使用，也可以将 Aurora 数据包加载到其他协议。

2. 定制协议

在某些情况下，用户可能希望制定自己的协议。特别是当标准协议不能满足要求，或者标准协议对于用户的应用来说太过宽泛时，制定用户自己的协议是个很好的选择。当然，有

时用户可能需要一个新的复杂协议，但是这种任务通常留给制定标准的专业协会。这里给出一个简单的例子，说明在制定自己的协议时应当考虑的各种事项。在这个简单的应用中，需要将恒定的 1.8GHz 信号流从一块板传送到另一块板。系统的输入/输出使用 12 位的总线，工作在 150MHz。针对这个简单的应用需求，协议中需要定义的内容包括数据帧结构、对齐和 idle(空闲)字符。此例中，我们使用 8b/10b 作为线路编码机制，并从其他 8b/10b 标准中借用标记及 comma 字符的定义。链路的基本结构如图 10-4 所示。

图 10-4　基本帧结构

首先需要为 SF(帧开始)、EF(帧结束)和 idle(空闲)指定字符或者有序的字符集，之后再确定线路速率和数据帧大小。适当设置数据帧的大小，以保证对齐时有充足的 SF 符号，并且进行时钟修正时能够有足够的 idle 符号。如果要传送 1.8GHz 的载荷，那么传输线速率为 2.5Gb/s，其有效载荷速率为 2Gb/s，可以满足 1.8GHz 的数据需求，其额外的容量还可用于所需开销。

10.3.3　Rocket I/O 硬核模块的体系结构

Xilinx 公司不同的系列芯片中集成的 Rocket I/O 是不同的，本节以经过大量应用的 Virtex-2 Pro 系列为例进行介绍。

1. Rocket I/O 架构

Virtex-2 Pro 系列 FPGA 最多可包含 16 个 Rocket I/O 模块，基本上分布于 FPGA 的上、下两端，如图 10-5 所示。每条通道可提供 622Mb/s～3.125Gb/s 的传输能力，且不需要在发送端配置串行数据速率，这是因为接收端的工作时钟是从接收数据中提取出来的。

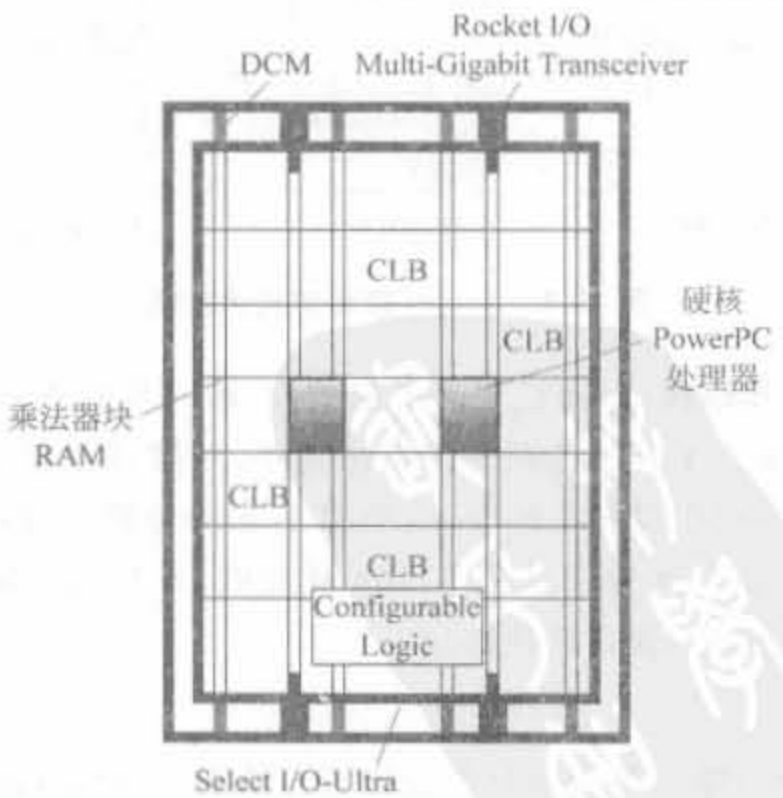


图 10-5　Rocket I/O 在 Virtex-2 Pro 芯片中的分布示意图

Rocket I/O 包括 PMA(Physical Media Attachment,物理媒介适配层)和 PCS(Physical Coding Sublayer,物理编码子层)两个子层,其内部结构如图 10-6 所示。其中,PMA 子层主要用于串行化和解串,PCS 主要包括线路编码和 CRC 校验编码。

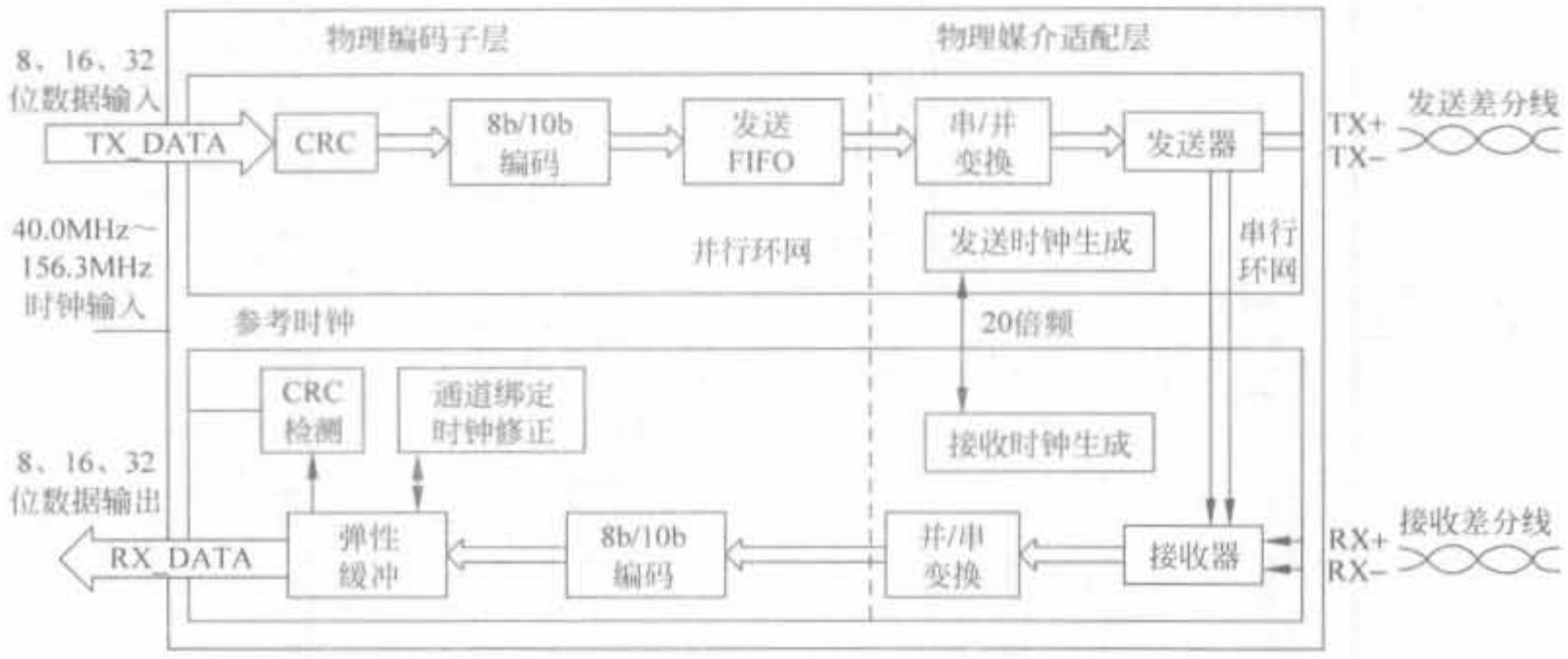


图 10-6　Rocket I/O 收发器的内部结构示意图

PMA 子层中集成了 SERDES、发送和接收缓冲、时钟发生器及时钟恢复电路。SERDES 是一个串/并转换器,负责 FPGA 中本地的 32 位并行数据(也可以是 16 位或 8 位)与 Rocket I/O 接口的串行数据之间的转换。采用串行数据收发,可以在高频条件下很好地避免数据间的串扰。时钟发生器及时钟恢复电路用于将时钟与数据绑定发送,以及将时钟从接收到的数据流中恢复出来,从而避免了在高速传输条件下时钟与数据分开传输所带来的时钟抖动等问题。

PCS 子层负责 8b/10b 编解码和 CRC 校验,并集成了负责通道绑定和时钟修正的弹性缓冲。8b/10b 编码可以避免数据流中出现连 0、连 1 的情况,便于时钟的恢复。通道绑定通过在发送数据流中加入 P 字符来将几个 Rocket I/O 通道绑定成一个一致的并行通道,从而提高数据的吞吐率。最多支持 24 个通道的绑定。弹性缓冲可以解决恢复时钟与本地时钟的不一致问题,并进行数据率的匹配,使得通道绑定成为可能。对 Rocket I/O 模块的配置,通过下面两种方式完成:静态特性可以通过 HDL 代码设置;动态特性可以通过 Rocket I/O 的原语端口进行配置。

2. Rocket I/O 硬核模块说明

Rocket I/O 硬核模块可通过原语和 Core Generator 调用,其模块结构如图 10-7 所示。可以看出,该硬核分为时钟合成器、时钟和数据恢复器、发送器、接收器、环回器、弹性传送缓冲器、CRC 校验模块、配置模块以及复位模块等 9 个主要组成部分。

1) 时钟合成器

在实际设计中,高性能的通信质量要求有高稳定性和高精度的时钟源,而抖动和频偏是衡量时钟源的两个重要指标。Rocket I/O 内部的工作时钟需要将输入时钟经过数十倍的倍频,但其可容忍的时钟偏差为 40ps,因此建议选择高精度的差分时钟。Xilinx 公司推荐选用 EPSON EG-2121CA 2.5V(LVPECL Outputs)或者 Pletronics LV1145B(LVDS

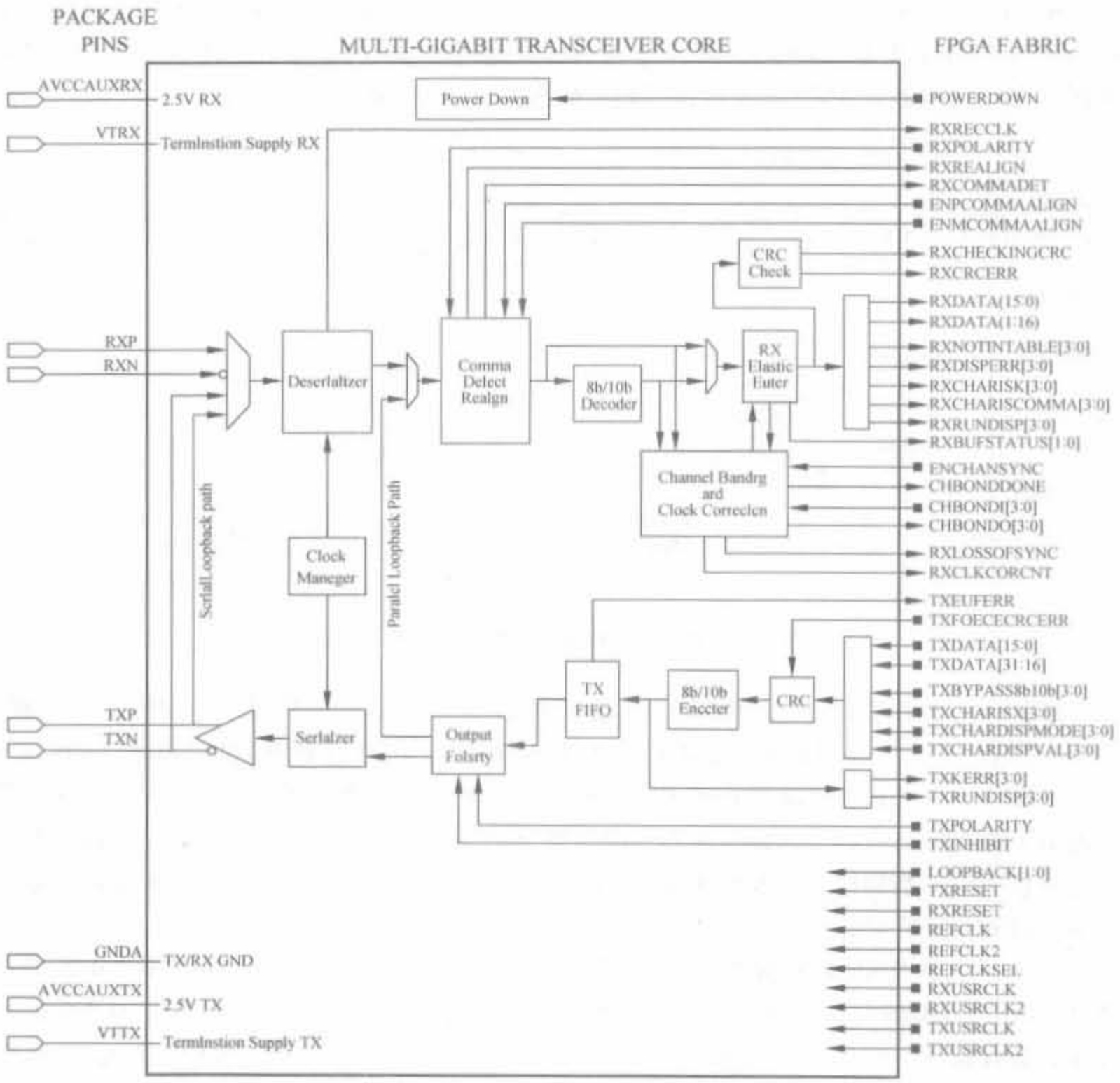


图 10-7　Virtex-2 Pro 系列 Rocket I/O 模块说明

Outputs)晶振。当 Rocket I/O 在 2.5G 波特以上时，参考时钟应采用差分输入方式(如 LVDS、LVPECL)，由专用差分时钟管脚输入，然后引到相同或相邻通道中 Rocket I/O 的参考时钟输入端；当 Rocket I/O 在 2.5G 波特以下时，不要使用 FPGA 内部的 DCM 模块来产生参考时钟，因为经过 DCM 倍频的时钟会引入较大的抖动，使 Rocket I/O 的接收锁相环无法稳定地锁定发送时钟。

Rocket I/O 采集数据的同步时钟则是通过时钟/数据恢复电路来提取的，该电路由一个单片的 PLL 集成块实现，不需要任何外部组件。恢复电路从接收的数据流中提取出时钟的频率和相位，并通过 20 倍分频后送到输出管脚 RXRECCLK 上。

当高精度输入到 FPGA 中后，吉比特发送器对参考时钟输入管脚 REFCLK 的信号完成 20 倍倍频操作，作为自己的工作时钟。同样，该倍频器已集成在芯片中，不需要额外的组件。RXRECCLK 和 REFCLK 二者之间没有固定的相位关系，且都为专用时钟信号，不能

连接到其他管脚上作为他用。当使用4字节或1字节数据接收路径时，RXUSRCLK和RXUSRCLK2具有不同的频率，但是频率低的时钟下降沿要和频率高的时钟下降沿对齐。同样的关系也适用于TXUSRCLK和TXUSRCLK2信号。

例如，在Virtex-2 Pro系列FPGA中，由于Rocket I/O模块内部将输入参考时钟20倍频，而Rocket I/O模块可容忍的输入参考时钟抖动公差仅为40ps，可见参考时钟的抖动对其性能有直接影响。

典型的时钟输入如图10-8所示，外部时钟由差分或单端管脚馈入后，只经过一级全局时钟缓冲(BUFG)布设到时钟树上，再连接到Rocket I/O的参考时钟上，可最大幅度地减小抖动。

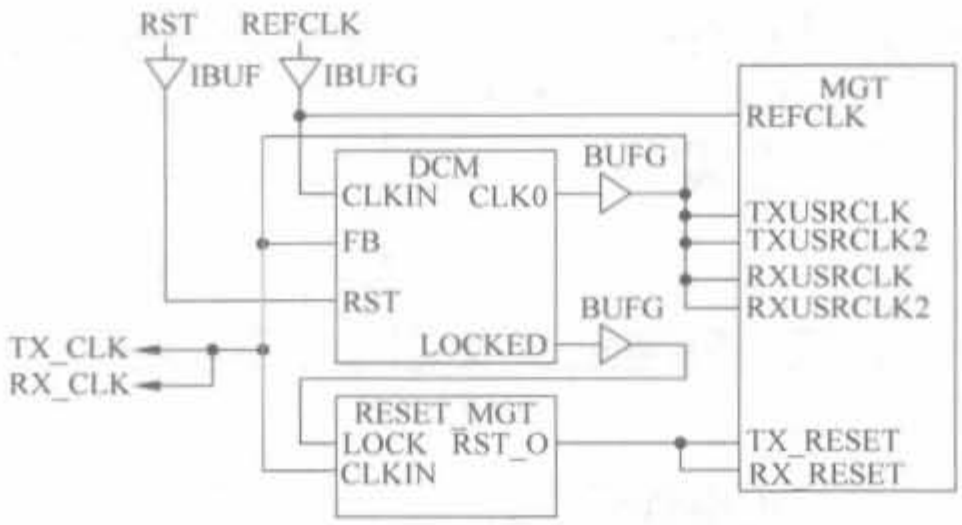


图10-8　Rocket I/O的时钟、复位连接示意图

2) 时钟和数据恢复器

如果没有数据存在，时钟/数据恢复器(CDR)电路会自动锁相到参考时钟上。为了使操作达到最佳性能，参考时钟的精度必须在100ppm之内，同时要满足供电系统的低噪声。如果有数据，恢复电路会自动同步锁相到输入数据上。

3) 发送器

发送器模块包括FPGA发送接口、8b/10b编码器、不均匀性控制、发送FIFO、串行化输出接口、发送终端以及预加重电路。

(1) FPGA发送接口

发送接口可发送1、2或4个数据字符到发送器，每个字符都是8bit或10bit位宽。当选择8bit位宽时，多出的2bit就变成8b/10b编码器的控制信号。如果同时将8b/10b编码旁路后，10bit字符的顺序如图10-9所示。

```
TXCHARDISPMODE[0]        第一个传输的比特
TXDATA[7:0]
TXCHARDISPVAL[0]         最后一个被传输的比特是TXDATA[0]
```

图10-9　旁路掉8b/10b编码后的字符顺序

(2) 8b/10b编码器

8b/10b编码器是一个可选的硬件组件，意味着它可以被旁路掉。在吉比特以太网、光纤信道以及Infiniband应用中，编码器都是用256个数据字符和12个控制字符。编码器在K字符(单比特)后读入8个数据比特，由这9个比特数据生成10bit编码输出。当K字符

为高时，数据将被编码成 8b/10b 码中可用的 12 个 K 字符组合中的 1 个；当 K 字符为低时，8bit 数据将被标准编码。

线路编码将 8 个数据位转换为不连续 5 个以上的“1”或“0”的 10 位比特码，以获取更好的直流平衡性，能提高数据传输速率，平衡码流中“0”和“1”的概率，并且减小码流中长连“0”和长连“1”串。

8b/10b 编码是属于基于块编码的 mbnb 线路码中的一种，很多串行标准，如 Infiniband、光纤通道千兆以太网 ATM ESCON 和 DVB-ASI 都针对原始数据流采用 8b/10b 编码和解码。其编码过程是将 8 个比特分成 5b/6b 与 3b/4b 两部分分别编码，如图 10-10 所示。

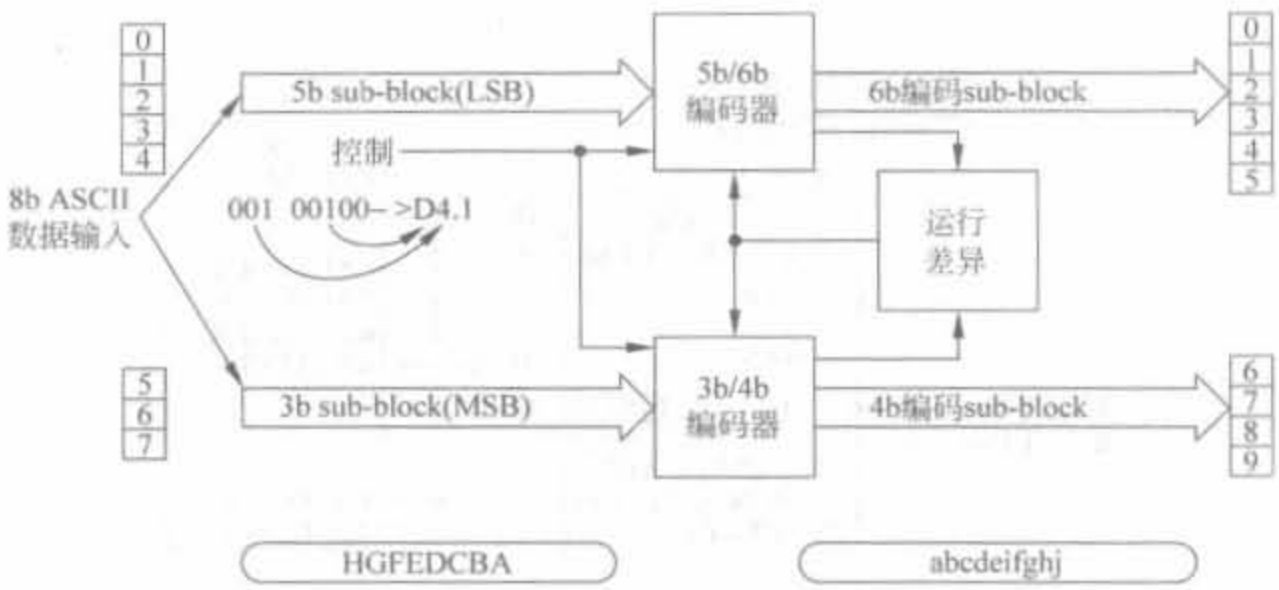


图 10-10　8b/10b 线路编码原理

8b/10b 编码集合中包括数据字串和控制字串两部分数据字串，包括 256 个可能的数值。其中，包括可作为控制字串 K23.7 K28.0～K28.7 K27.7 K29.7 K30.7 的码字。当传送的字串作为控制字串时，由 Rocket I/O 模块相应的控制字标志输入管脚，如 TX_CHARISK 指定该字为控制字串。尽管 8b/10b 编码后，数据的传输效率只有 80%，但还是获得广泛应用，其原因在于下面几个方面：

① 转换密度：其目的是保证在发送数据流中有足够的信号转换，以保证 PLL 正常工作。如果没有 8b/10b 编码方法，数据中的一串 1 或 0 有可能引起接收时钟漂移或同步丢失，从而引起数据丢失。

② DC 补偿：8b/10b 还保证对信号进行 DC 补偿，这意味着链路中不会随着时间推移而出现 DC 偏移。8b/10b 编码将用户数据按字节 8 位块变换成多个 10 位的输出值，用这些值进行 DC 补偿。

③ 纠错：8b/10b 编码遵循特定规则，根据这些规则，很多信号传输错误都可立即检测出来。

④ 特殊字符：8b/10b 编码采用 10 位字符，其数量是 8 位字符空间中字符数的 4 倍，这样可以将特殊字符编码放入数据流中，它们不会被解码成数据字符，这些特殊字符可用作分隔符或其他命令。

8b/10b 模块在设计中可以旁路掉，也可以用其余的线路编码方式代替，如 64b/66b 编码等。将发送器的 TXBYPASS8b10b[n]控制信号设置成 1，则可将发送端的线路编码模块旁路；在接收器中将 RX_DECODE_USE 设置为 FALSE，就可把接收端的解码模块旁路。

在实际设计中，依据互连双方接口的电气标准的不同，有两种耦合方式：直流耦合和交流耦合。在直流耦合方式下，并不一定要求采用 8b/10b 编码，可以选择其他的线路码型或不用线路编码。但在较高的传输速率条件下，如 2.5G 波特以上，为了达到良好的抗干扰性能和低的误码率，应该考虑采用 8b/10b 编码；而在交流耦合方式下必须要选择 8b/10b 编码，否则接收端的漂移会使接收器无法正常工作。

(3) 不均匀性控制(Disparity Control)

8b/10b 编码器被连续的"－"不均匀初始化，统一控制当前状态为不均匀性运行状态。TXRUNDISP 表明了不均匀性的状态。TXCHARDISPMODE 和 TXCHARDISPVAL 可控制对每个接收字节使能不均匀性控制。例如，收发器可通过反向指定第二、第四字节的不均匀性检测，产生以下序列：

K28.5+K28.5+K28.5−K28.5−　或　K28.5−K28.5−K28.5+K28.5+

(4) 发送 FIFO

发送 FIFO 的长度为 4，可通过配置旁路掉。只要当信号 TXUSRCLK 锁频到参考时钟 REFCLK 上，就可以使用 FIFO，允许 1 个时钟周期的相位差。

(5) 串行化输出接口

由于 Rocket I/O 将参考时钟完成 20 倍频作为自身的工作时钟，因此数据需要经过并/串转换后，才能通过 TXP 和 TXN 这一对差分端口发送出去。其中，时隙 0 发送第一个比特，时隙 19 发送最后一个比特。TXP 和 TXN 的电气连接特性是配置可改变的，可通过发送端的控制输入信号 TXPOLARITY 来实现。如果 PCB 上连线接反的话，可通过对 TXPOLARITY 信号的编程来修正。

(6) 发送终端

发送器提供了发送终端，有 75Ω 和 50Ω 两种可编程终端，无需额外的片外终端。默认值为 50Ω。

(7) 预加重

预加重的原理如 10.2.3 节所述，在 Rocket I/O 中，预加重电路有 4 个可选程度：10%、20%、25%以及 33%，默认值为 10%，可通过编程实现。选择最佳的预加重参数，可使发送器以最高波特率驱动 20 英寸的 FR4。

4) 接收器

接收器模块主要包括解串器、接收终端和 8b/10b 解码器。下面将简要介绍。

(1) 解串器

Rocket I/O 收发器核通过 RXP 和 RXN 这一对差分接口接收差分数据，并通过芯片内部的锁相环从中提取时钟，再按照此同步时钟来采样数据，无需片外 PLL 芯片。从数据中提取的同步时钟包括频率和相位信息，经过 20 分频后送到 RXRECCLK。

接收端不仅可以处理跳变丰富的 8b/10b 编码流或加扰流，也可以处理没有跳变的连续的 75bit 数据流。如果使能了 comma 检测器，收发器可能识别的字符最多为两个 10bit 预编码字符。如果检测到字符/字符串，comma 检测器的输出将被拉高，并且数据将被同步排列，这时不会发生队列更替的现象。如果收到一个 comma 且队列需要重排，数据会被重新排列，并在接收端给出指示信息。此时，收发器会连续检测数据，寻找 10bit 预编码字符。如果将 comma 检测旁路掉，数据将不会在任何模式下重排。

可编程选项允许用户以 comma+、comma-、comma+与 comma-混合或用户自定义的序列来排列数据。此外，接收器允许更改 RXP 和 RXN 上差分信号的极性，在 PCB 电路设计极性颠倒的情况下非常有用。

(2) 接收终端

接收机提供了片上的接收终端，可配置为 50Ω 或 75Ω。默认为 50Ω。

(3) 8b/10b 解码器

8b/10b 解码器是和编码器配对出现的，如果发送端存在编码器，接收端必须具备该模块，不能旁路。

5) 环回器

为了便于测试 Rocket I/O，提供了两类可编程的环回器，它们无需在外部添加吉比特的数据端和测试终端。

一种方法是串行环回，将吉比特发送端的数据直接送到吉比特接收端，可以验证 Rocket I/O 模块发送端和接收端的完整性。该环回路径在发送端的输出端口上。

另一类是并行环回，用于检查整个传输电路的正确性。当使能并行环回时，串行环回的功能将失效。但是发送端的输出仍然保持有效，且可将数据通过链路发送出去。如果将 TXINHIBIT 拉高，则 TXP 将被强制为 0，直到 TXINHIBIT 重新拉低为止。

6) 弹性传送缓冲器

(1) 接收缓冲器

接收缓冲器为深度 64、位宽 13bit 的 FIFO，写时钟为恢复时钟 RXRECCLK，读时钟为 RXUSRCLK，其作用有两个：一是用来调节读、写时钟的相位差和频率差；另一个是支持通路绑定，允许将接收流重组，以便被多个收发器读取。此外，接收缓冲器是一个弹性缓冲，其“弹性”特征指的是可以修正其读取指针。

当然，接收缓冲器可以被旁路掉，其控制属性为：

```
RX_BUFFER_USE = FALSE
```

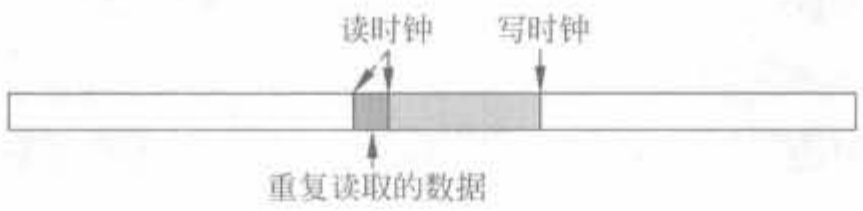
在该模式下，不能完成时钟校正和通路绑定，RXUSRCLK 必须由 RXRECCLK 直接产生，但由于布线等因素，二者的相位是无法预测的，无法保证接收端可靠地接收数据。因此，一般不允许将该缓冲器旁路。

(2) 接收端时钟校正器

在发送端，每隔一定的包长都会插入一些特定的修正字符；在接收端，这些字符仅用来实现时钟校正，然后被丢掉。恢复时钟 RXRECCLK 的频率反映了到来数据的速率，时钟 RXUSRCLK 定义了 FPGA 接收数据的速率，在理想情况下，二者应当是同步的，即频率和相位都一致。但由于在实际中，它们来自不同的时钟源，属于异步时钟，再加上抖动，因此在频率、相位上肯定存在一定的偏差，二者需要通过接收缓冲来调节。常见的异步时钟情况如图 10-11 所示。

其中，图(a)所示是理想情况，弹性缓冲器的读时钟 RXUSRCLK 和写时钟 RXRECCLK 保持同步，缓冲器处于半满状态；当没有接收到有效数据时，接收数据将插入空闲(Idle)字符以及其余无效数据。在图(b)所示情况下，读时钟快于写时钟，缓冲器将出现读空的状态。

(a) 理想情况下(读时钟=写时钟)，缓冲器半满示意图



(b) 读时钟＞写时钟时，缓冲器少于半满示意图

(c) 读时钟＜写时钟时，缓冲器多于半满示意图

图 10-11　不同读写时钟模式下，缓冲器状态示意图

为了避免这个后果，需要进行一些重复读取或空读的操作，调整读数数据指针，对时钟完成降低校正。如果字节序列长度大于 1，且属性 CLK_COR_REPEAT_WAIT 为 0，接收端会重复读取相同的内容，一直到缓冲器的长度达到理想的半长。在图(c)所示情况下，读时钟慢于写时钟，缓冲器将出现溢出的状态。为避免该后果，需要丢掉一些数据，调整数据指针，对时钟完成加快校正。当 CLK_COR_REPEAT_WAIT 为 1 时，接收端会跳过两个连续的、可删除的字节序列来清空缓冲器。因此，时钟校正器总是保持缓冲器处于半满状态。

以上操作要求时钟校正逻辑能够识别那些需要重复和删除的字节序列，这些冗余序列都是在发送端插入的。此外，时钟校正和振荡器的频率关系很大，因此晶振的精度一般应该在 50ppm 以内。

(3) 通道绑定

通道绑定是指将多个串行通道组合在一起构成一个并行通道，以此来提高收发的数据吞吐率。由于每个通道在收发器互连时钟再生和数据接收延迟上各不相同，会使接收到的数据产生“错位”的情况，因此要在发送端数据流中加入一个特殊的序列——通道绑定序列，如图 10-12 中的“P 字符”。每个绑定通道都设定“P 字符”为通道绑定序列，在接收端指定一个通道为主通道，其余通道都依据主通道的 CHBONDO 有效指示进入绑定状态，进而锁定本通道在弹性缓冲中接收到通道绑定序列的位置。由弹性缓冲向内部逻辑电路输出数据时，所有经过绑定的通道都以绑定序列指定的弹性缓冲中的偏移位置进行对齐输出。通道绑定完成后，为了使绑定维持在稳定状态，各通道收发器也要以主通道收发器为基准进行时钟修正操作。

在实现时，FPGA 的布线原则是使绑定指示信号在模块间传输的延迟尽量小，尽量使两个互连模块间的连线不要穿越整个芯片。在 FPGA 布线时，要对绑定指示互连线设置严格的延时约束参数。

典型的通道绑定的示意图如图 10-12 所示，左边为原始数据，右边为经过通道绑定的数据示意图。

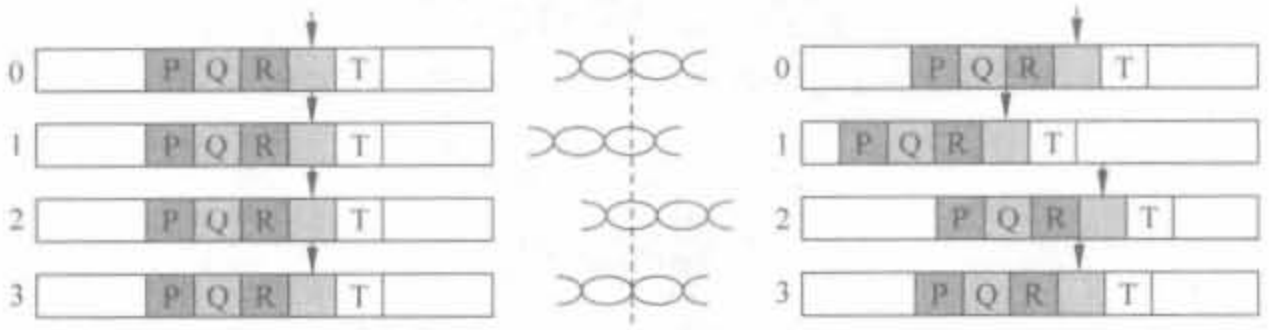


图 10-12　通道绑定原理图

(4) 发送缓冲器

发送缓冲器为深度 4、位宽 20bit 的 FIFO，写时钟为恢复时钟 TXUSRCLK，读时钟为收发器的参考时钟，其作用是用来调整读、写时钟之间的相位差，完成发送端时钟校正。

发送缓冲器也可以被旁路掉，其控制属性为：

```
TX_BUFFER_USE = FALSE
```

在该模式下，不能完成时钟校正和通路绑定，TXUSRCLK 和 REFCLK 的相位是无法预测的，无法保证发送端可靠发送数据。因此，一般不允许将该缓冲器旁路。

7) CRC 校验模块

Rocket I/O 收发器支持固定的 32bit CRC 校验算法，其编码公式为

$$CRC = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + 1$$

该算法可支持 Infiniband、光纤信道以及吉比特以太网等传输协议。在发送端，CRC 逻辑识别出 CRC 校验字节应添加的位置，统计包头和包尾，并在数据包尾上添加 4 个计算出的 CRC 占位符，因此需要在发送缓冲器预留 4 字节的 CRC 码空间。在接收端，CRC 完成 CRC 值的校验。同样，CRC 逻辑也支持用户模式，以用户定义的包头和包尾来定义简易数据包。

8) 配置模块

下面列出可通过配置选择或控制的功能，Xilinx 实现软件工具支持 16 个收发原语，其简要说明如表 10-1 所示。

表 10-1　Rocket I/O 可配置的原语说明

原语名称	简要功能说明
GT_CUSTOM	完全由用户自定义
GT_FIBRE_CHAN_1	光纤通路，1 个字节宽度的通路
GT_FIBRE_CHAN_2	光纤通路，2 个字节宽度的通路
GT_FIBRE_CHAN_4	光纤通路，4 个字节宽度的通路
GT_ETHERNET_1	吉比特以太网，1 个字节宽度的通路
GT_ETHERNET_2	吉比特以太网，2 个字节宽度的通路
GT_ETHERNET_4	吉比特以太网，4 个字节宽度的通路
GT_XAUI_1	10 吉比特以太网，1 个字节宽度的通路
GT_XAUI_2	10 吉比特以太网，2 个字节宽度的通路

续表

原语名称	简要功能说明
GT_XAUI_4	10 吉比特以太网，4 个字节宽度的通路
GT_INFINIBAND_1	Infiniband，1 个字节宽度的通路
GT_INFINIBAND_2	Infiniband，2 个字节宽度的通路
GT_INFINIBAND_4	Infiniband，4 个字节宽度的通路
GT_AURORA_1	Xilinx 制定的通信协议，1 个字节宽度的通路
GT_AURORA_2	Xilinx 制定的通信协议，2 个字节宽度的通路
GT_AURORA_4	Xilinx 制定的通信协议，4 个字节宽度的通路

以上每个原语都有其默认值，都允许修改，原语的使用方法见 3.4 节，详细的参数说明和更多的细节可参考文献[1]。

9）复位模块

Rocket I/O 模块的复位管脚分为发送复位和接收复位两部分。由于 DCM 在输出时钟锁定之前处于不稳定状态，不能用作内部逻辑电路时钟，所以要在 DCM 模块锁定后，经过适当延迟才能将片内逻辑复位。

发送部分的复位主要包括 TXPMARESET 和 TXPCSRESET，接收部分的复位主要包括 RXPMARESET 和 RXPCSRESET。TXPMARESET 复位用于复位 PMA 和重新初始化 PMA 功能。其管脚电平为高时，复位 PLL 控制逻辑和内部的 PMA 分频器，同时使发送器 PLL LOCK 信号为低电平，并且迫使 TX PLL 进行校验。TXPMARESET 管脚的高电平至少要持续 3 个 USRCLK 时钟周期。当 TXPCSRESET 管脚电平为高时，TX PCS 模块被复位。TX PCS 模块包括 TX Fabric 接口、8b/10b 编码器、10GBASER 编码器、TX 缓冲器、64b/66b 扰码器和 10GBASER 自适应同步器。TXPCSRESET 复位与 TXPMARESET 复位是相互独立，互不影响的。Rocket I/O 模块要求复位输入至少保持两个 USRCLK 时钟周期，才能完成 FIFO 的初始化。

TXPCSRESET 复位的要求如下：

（1）在 TXPCSRESET 复位时，TXUSRCLK 和 PCS 的 TXCLK 时钟必须已经保持稳定，以便初始化发送缓冲器。

（2）TXPCSRESET 管脚电平为高，至少要持续 3 个 TXUSRCLK 或 TXUSRCLK2 时钟周期。

（3）在 TXPCSRESET 复位结束后，TX PCS 模块至少需要 5 个时钟周期（以 TXUSRCLK 或 TXUSRCLK2 中最长的时钟周期为准）来完成各个子模块的复位。

发送部分的复位时序图如图 10-13 所示。接收部分的复位时序图和复位要求与发送部分类似，请参见 Xilinx 公司的技术文档[13]。

10）上电顺序

虽然 Rocket I/O 模块对于上电顺序没有要求，任意的上电顺序也不会损坏芯片，但为了减少上电的瞬间电流，最好按照下面的上电顺序：

（1）以任意顺序加载 FPGA 的 V_{CCINT} 以及 V_{CCAUX} 电源。

（2）加载 $AV_{CCAUXRX}$ 电源。

（3）以任意顺序加载 $AV_{CCAUXRX}$、V_{TTX} 以及 V_{TRX} 电源。

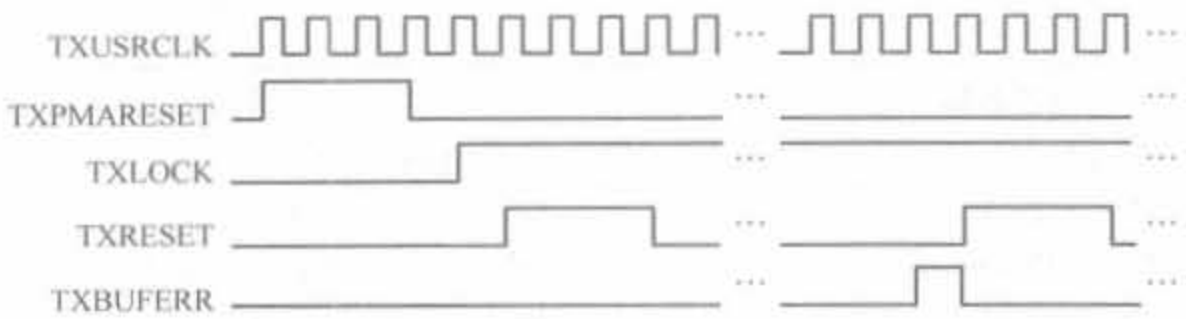


图 10-13　发送部分的复位时序图

3. Rocket I/O 硬核模块接口信号说明

1）时钟信号

Rocket I/O 时钟信号分为两类：一类用于收发器高质量的时钟，另一类用于接收/发送缓冲器进行数据交换的同步时钟，具体如表 10-2 所示。

表 10-2　Rocket I/O 时钟信号简要说明列表

信号名	方向	位宽/bit	功能描述
REFCLK	输入	1	高质量的时钟输入信号，是 TXFIFO 的读时钟，并为串行器、20 倍时钟倍频器以及时钟恢复电路提供基本的参考时钟
REFCLK2	输入	1	功能和 REFCLK 一样，可在一定场合下替代 REFCLK
REFCLKSEL	输入	1	参考时钟的选择控制信号。当 REF_CLK_V_SEL 为 0 时，REFCLKSEL 为 0，选择 REFCLK 作为输入时钟；REFCLKSEL 为 1，选择 REFCLK2 作为输入时钟。当 REF_CLK_V_SEL 为 1 时，REFCLKSEL 为 0，选择 BREFCLK 作为输入时钟；REFCLKSEL 为 1，选择 BREFCLK2 作为输入时钟
BREFCLK	输入	1	高质量的时钟输入信号，通常用于 2.5Gb/s 以上的传输速率。该信号具备专用路由，位于器件的左端，可将抖动降低到最低
BREFCLK2	输入	1	功能和 BREFCLK 一样，可在一定场合下替代 BREFCLK
RXUSRCLK	输入	1	用于读取接收缓冲器以及通道绑定，一般由 DCM 模块提供
RXUSRCLK2	输入	1	用于收发器和 FPGA 内部逻辑之间的接口以及状态信息的控制时钟。一般由 DCM 模块提供
TXUSRCLK	输入	1	用于写发送/接收缓冲器，一般由 DCM 模块提供
TXUSRCLK2	输入	1	用于收发器和 FPGA 内部逻辑之间的接口以及状态信息的控制时钟。一般由 DCM 模块提供

2）数据发送模块信号

数据发送模块的信号类型众多，包括数据信号、发送模块的控制信号、状态控制信号、内部组件的配置信号等，其简要功能如表 10-3 所示。

表 10-3　Rocket I/O 数据发送模块信号简要说明列表

信号名	方向	位宽	功能描述
TXDATA	输出	1、2、4Byte	发送数据寄存器，位宽可配置
TXBYPASS8b10b	输入	1bit	8b/10b 编码器控制输入，为 1 时表示旁路编码器，为 0 时表示使能编码器
TXCHRISK	输入	1bit	用于指示所发送的是数据还是 K 字符。设置为 1 时，表示发送的是 K 字符；为 0 时，表示的是数据

续表

信号名	方向	位宽	功能描述
TXCHARDISPMODE	输入	1bit	设置不均匀模式
TXP	输出	1bit	高速差分串行信号的正端
TXN	输出	1bit	高速差分串行信号的负端
TXBUFFERR	输入	1bit	当设置为1时，表明发送缓冲器已经溢出。溢出后需要通过TSRESET信号来复位
TXKERR	输入	1bit	当设置为1时，表明非法K字符被发送
TXRUNDISP	输出	1bit	统计编码后字符中0和1的比例。当其为0时，表明0的个数多于1的个数；当其为1时，表明1的个数多于0的个数
TXRESET	输入	1bit	发送器的复位信号，可复位发送缓冲器、8b/10b编码器以及其余内部状态寄存器
TXINHIBIT	输入	1bit	当设置为1时，将强制差分端口输出1/0
TXPOLARITY	输入	1bit	发送端极性控制输入，一般设置为0
TXFOECECRCERR	输入	1bit	当其设置为1时，强制发送口出现CRC错误
ENCHANSYNC	输入	1bit	输入到发送器中的控制信号，使能收发器通道绑定的功能
CHBONDDONE	输出	1bit	当其为1时，表明接收器已成功完成了通道绑定
CHBONDI	输出	1bit	通道绑定的级联控制信号，只能用于从收发器
CHBONDO	输出	1bit	通道绑定的级联控制信号，用于控制其他收发器进行通道绑定和时钟修正

3）数据接收模块信号

数据接收模块信号和发送模块的信号对应，包括数据信号、发送模块的控制信号、状态控制信号、内部组件的配置信号等，其简要功能如表10-4所示。

表10-4 Rocket I/O数据接收模块信号简要说明列表

信号名	方向	位宽	功能描述
RXDATA	输入	1、2、4Byte	接收数据寄存器，位宽可配置
RXNOTINTABLE	输出	1bit	当其为1时，表示接收器所译码的数据是一个非法字符
RXDISPERR	输出	1bit	在8b/10b编码器有效的情况下，在串行链路中出现不均匀错误
RXCHARISK	输出	1bit	在8b/10b编码器有效的情况下，当其为1时，表明所接收的数据为K字符
RXCHARISCOMMA	输出	1bit	在8b/10b编码器有效的情况下，当其为1时，表明所接收的数据是一个comma字符
RXRUNDISP	输出	1bit	统计接收字符中0和1的比例。当其为0时，表明0的个数多于1的个数；当其为1时，表明1的个数多于0的个数
RXRESET	输入	1bit	接收器的复位信号，可复位8b/10b解码器、comma检测器、接收缓冲器、通道绑定逻辑、时钟修正逻辑。在接收缓冲溢出后，必须要进行复位处理
ENPCOMMAALIGN	输入	1bit	用于字节队列的同步控制。当设置为1时，将检测到的“+”不均匀加入字节队列

续表

信号名	方向	位宽	功能描述
ENMCOMMAALIGN	输入	1bit	用于字节队列的同步控制。当设置为1时，将检测到的“—”不均匀加入字节队列
RXPOLARITY	输入	1bit	接收端极性控制信号
RXP	输入	1bit	高速差分串行信号的正端
RXN	输入	1bit	高速差分串行信号的负端
RXCOMMADET	输出	1bit	当其为1时，表明接收到comma字符
RXREALIGN	输出	1bit	当其为1时，表明在检测到comma字符时，出现字节队列
RXLOSSOFSYNC	输出	1bit	反映接收端的同步状态信息，0表示同步，1表示未同步
RXBUFSTATUS	输出	1bit	反映接收缓冲器的状态，bit1为1时，表明出现溢出错误；bit0为0时，表明接收缓冲器已经半满。当接收缓冲器出现溢出错误后，必须由RXRESET复位
RXCLKCORCNT	输出	3bit	3bit时钟修正和通道绑定指示信息。000表示没有通道绑定和时钟修正；001表示对于当前数据，跳过1个时钟修正序列；010表示跳过2个时钟修正序列；011表示跳过3个时钟修正序列；100表示跳过4个时钟修正序列；101表示执行通道绑定；110表示重复了2个时钟修正序列；111表示重复了1个时钟修正序列
RXCHECKINGCRC	输出	1bit	接收端的CRC校验状态，1表示已识别出数据包和CRC码
RXCRCERR	输出	1bit	CRC状态信息，1表示CRC校验错误

4）模块控制信号

模块控制信号主要用于使能Rocket I/O模块以及控制反馈输入，其简要功能如表10-5所示。

表10-5 Rocket I/O模块控制信号简要说明列表

POWERDOWN	输入	1bit	用于写发送/接收缓冲器，一般由DCM模块提供
LOOPBACK	输入	1bit	用于收发器和FPGA内部逻辑之间的接口以及状态信息的控制时钟，一般由DCM模块提供

10.3.4 Rocket I/O的时钟设计方案

Virtex-2 Pro FPGA内嵌的Rocket I/O模块支持全速率(Full Rate)和半速率(Half Rate)两种数据传输速率。前者将外部参考时钟倍频20倍，单周期传输20bit数据，数据速率为1.0Gb/s～3.125Gb/s；后者将外部参考时钟倍频10倍，单周期传输10bit数据，数据速率为0.5Gb/s～1Gb/s。

1. Rocket I/O的时钟简介

每个Rocket I/O的收发器具备8个时钟输入，按照功能可以分为3类。

1）外部输入时钟

REFCLK、REFCLK2、BREFCLK以及BREFCLK2都是由外部时钟源提供的差分参

考时钟，但只有一个时钟能驱动 Rocket I/O 模块，通过 REFCLKSET 信号来选择，其组成结构如图 10-14 所示。

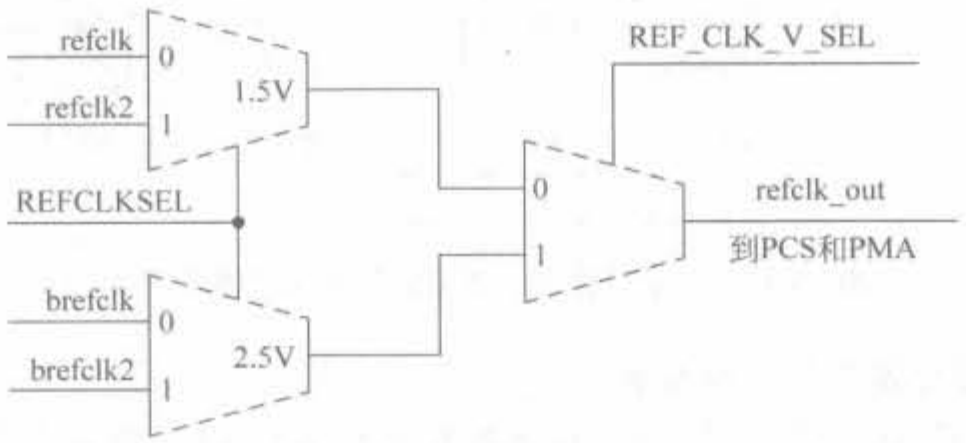


图 10-14　外部差分时钟的内部选择结构示意图

当数据速率高于 2.5Gb/s 时，必须选择 BREFCLK(BREFCLK2)作为参考时钟；在其他情况下可任意挑选。BREFCLK(BREFCLK2)要求低抖动的时钟源，用来驱动串/并转换、并/串转换以及 DCM 模块的时钟输入。

2）接收时钟

接收时钟包括 TXUSRCLK2 和 RXUSRCLK2，主要用于控制 Rocket I/O 和 FPGA 的接口逻辑，包括缓冲器和数据交换器，由 DCM 模块提供。

3）内部时钟

内部时钟包括 TXUSRCLK 和 RXUSRCLK，主要用于 Rocket I/O 模块发送、接收缓冲器数据的处理和时钟修正。一般由 DCM 模块提供，可以利用外部参考时钟。

Rocket I/O 的速度等级由 SERDES_10B 选择。若 SERDES_10B＝FALSE，则为全速率数据传输；否则为半速率数据传输。此外，TXUSRCLK2 和 RXUSRCLK2 控制着 Rocket I/O 和 FPGA 的交互接口，当数据位宽不同时，其与 TXUSRCLK 和 RXUSRCLK 的频率值比值也是不同的，如表 10-6 所示。默认情况下 TXUSRCLK2＝TXUSRCLK，即支持两字节数据传输。

表 10-6　数据位宽和 TXUSRCLK/RXUSRCLK 列表

数据位宽/Byte	RXUSRCLK/RXUSRCLK2(TXUSRCLK/TXUSRCLK2)
4	2/1
2	1/1
1	1/2

2. 全速率的时钟方案

全速率模式下，需要将 SERDES_10B 设为 FALSE，REFCLK 时钟的倍频倍数为 20 倍。

1）单字节位宽全速率的时钟方案

单字节位宽全速率方案中，TXUSRCLK2 和 RXUSRCLK2 的频率是 TXUSRCLK 和 RXUSRCLK 的 2 倍，相位差 180°。REFCLK、TXUSRCLK 和 RXUSRCLK 的时钟频率为 40MHz～85MHz，相应地，TXUSRCLK2 和 RXUSRCLK2 的时钟频率为 80MHz～170MHz，相应的时钟提供方案如图 10-15 所示。

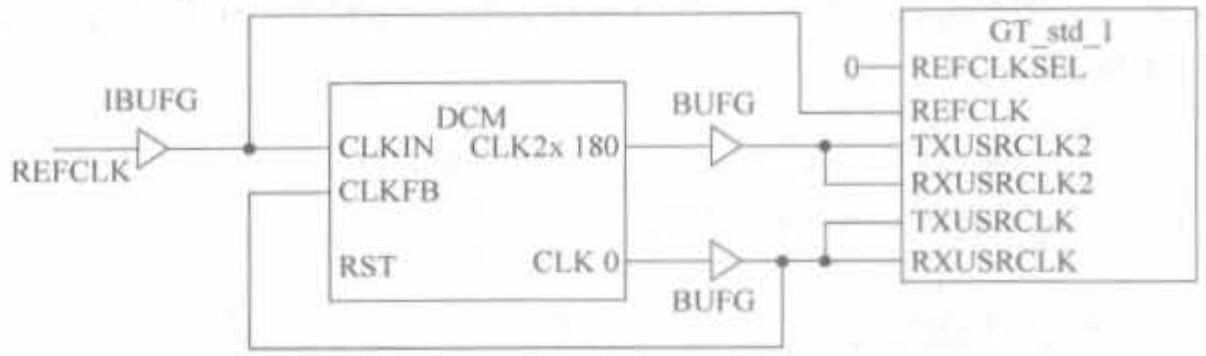


图 10-15　单字节位宽全速率时钟解决方案示意图

2）双字节位宽全速率的时钟方案

双字节位宽全速率方案中，TXUSRCLK2 和 RXUSRCLK2 的频率和 TXUSRCLK 和 RXUSRCLK 的频率相等，无相位差。REFCLK、TXUSRCLK 和 RXUSRCLK 的时钟频率为 40MHz～156.25MHz，TXUSRCLK2 和 RXUSRCLK2 直接由 REFCLK 经过 DCM 模块的 CLK0 信号提供，相应的时钟提供方案如图 10-16 所示。

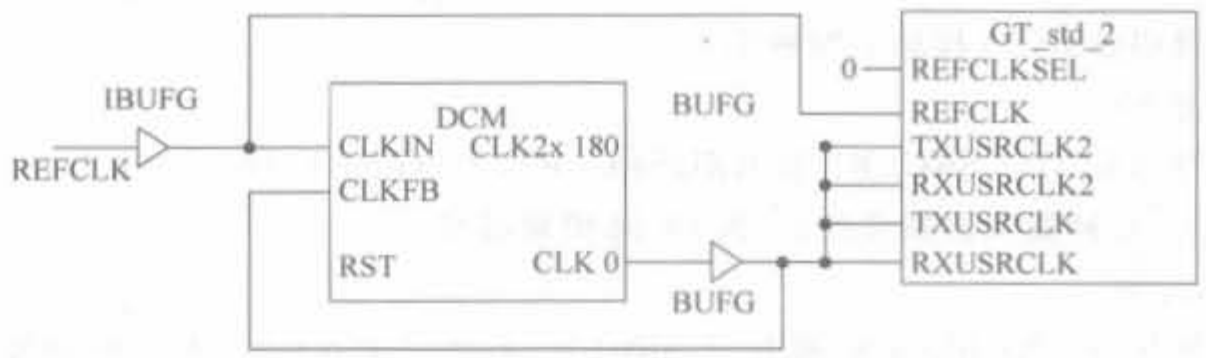


图 10-16　双字节位宽全速率时钟解决方案示意图

3）四字节位宽全速率的时钟方案

四字节位宽全速率方案中，TXUSRCLK2 和 RXUSRCLK2 的频率是 TXUSRCLK 和 RXUSRCLK 的一半。REFCLK、TXUSRCLK 和 RXUSRCLK 的时钟频率范围为 40MHz～156.25MHz。相应地，TXUSRCLK2 和 RXUSRCLK2 的时钟频率范围为 280MHz～78.125MHz，相应的时钟提供方案如图 10-17 所示。

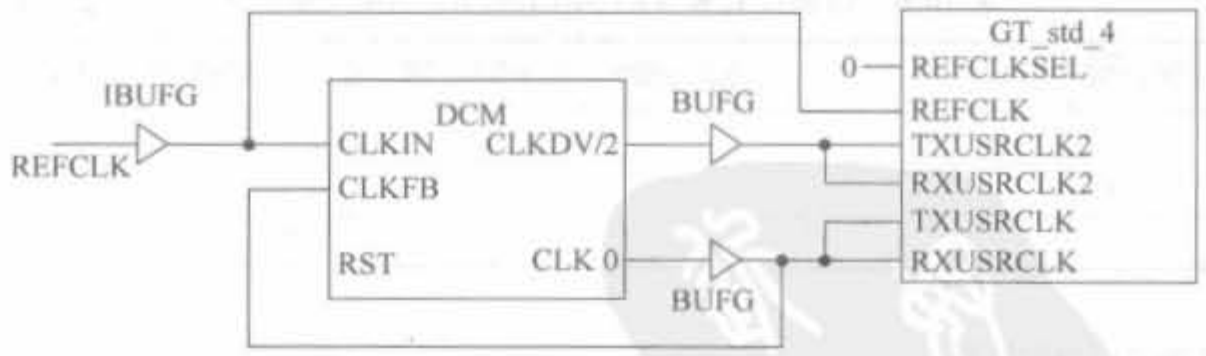


图 10-17　四字节位宽全速率时钟解决方案示意图

3. 半速率的时钟方案

半速率模式下，需要将 SERDES_10B 设为 TRUE，REFCLK 时钟的倍频倍数为 10 倍。

1）单字节位宽半速率的时钟方案

在该方案中，REFCLK 的倍频倍数为 10，RXUSRCLK2/TXUSRCLK2＝2×RXUSRCLK/

TXUSRCLK=REFCLK，REFCLK、TXUSRCLK 以及 RXUSRCLK 的时钟频率范围为 50MHz～100MHz，TXUSRCLK2/RXUSRCLK2 的时钟频率范围为 25MHz～50MHz，相应的时钟解决方案如图 10-18 所示。

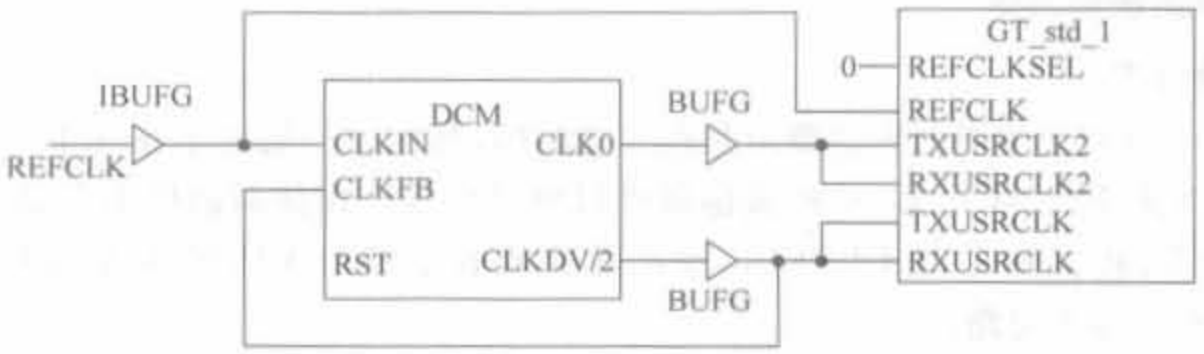


图 10-18　单字节位宽半速率时钟解决方案示意图

2）双字节位宽半速率的时钟方案

在该方案中，REFCLK 的倍频倍数为 10，RXUSRCLK2/TXUSRCLK2＝RXUSRCLK/TXUSRCLK＝REFCLK/2，REFCLK、TXUSRCLK、RXUSRCLK 以及 TXUSRCLK2、RXUSRCLK2 的时钟频率范围为 25MHz～50MHz，相应的时钟解决方案如图 10-19 所示。

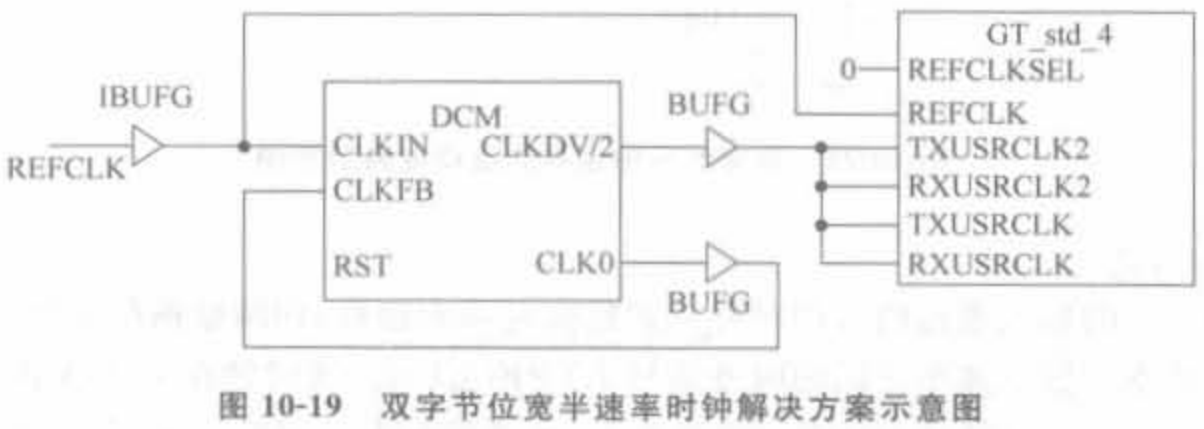


图 10-19　双字节位宽半速率时钟解决方案示意图

3）四字节位宽半速率的时钟方案

在该方案中，REFCLK 的倍频倍数为 10，2×RXUSRCLK2/2×TXUSRCLK2＝RXUSRCLK/TXUSRCLK＝REFCLK/2，REFCLK 的时钟频率范围为 50MHz～100MHz，TXUSRCLK 和 RXUSRCLK 的时钟频率范围为 25MHz～50MHz，TXUSRCLK2 和 RXUSRCLK2 的时钟频率范围为 12.5MHz～25MHz，相应的时钟解决方案如图 10-20 所示。

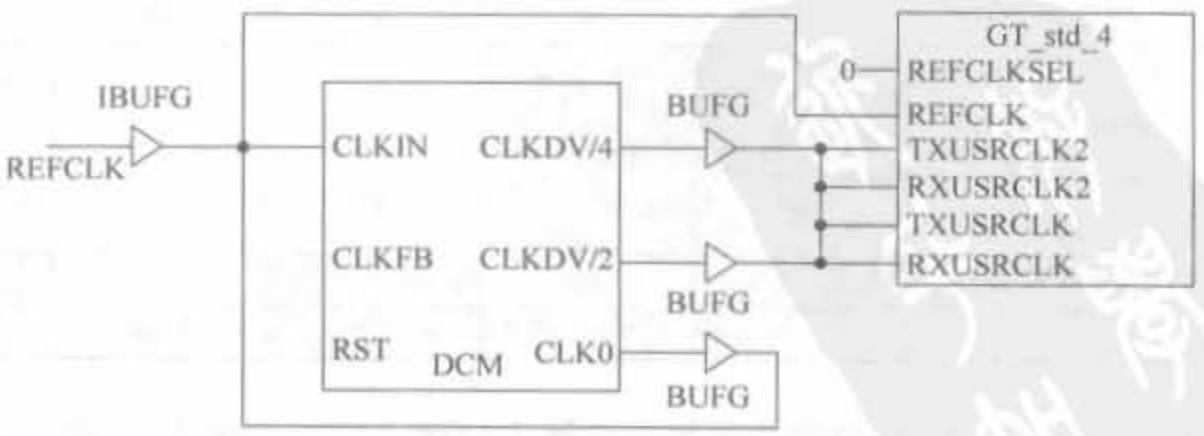


图 10-20　四字节位宽半速率时钟解决方案示意图

10.3.5 Rocket I/O 的开发要素

1. 时钟和数据恢复

1) 时钟考虑

Rocket I/O 对于参考时钟的要求是较为苛刻的，其精度一般要求在几十个 ppm 以下。例如，EPSON EG-2121CA 2.5V 的振荡器可以满足要求。振荡器的供电方案可参考其数据手册。此外，将振荡器的 LVPECL 输出转化成收发器的 LVDS 参考时钟，必须利用图 10-21 所示的参考电路。

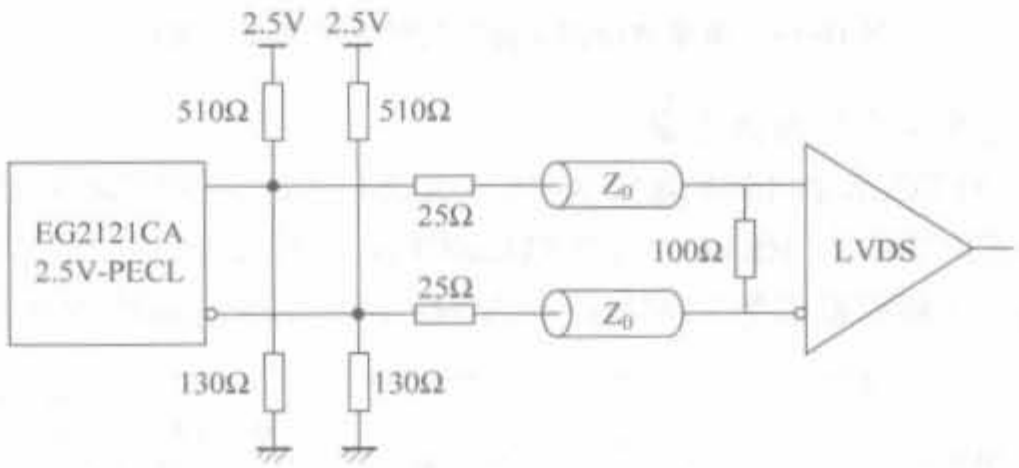


图 10-21 参考时钟振荡器的接口电路示意图

2) 数据恢复

串行收发器的输入通过内嵌的时钟和数据恢复单元锁存，切断数据的下降沿和上升沿，并驱动一个频率和数据速率相同的时钟信号 RXRECCLK。该时钟在 FPGA 内部表现为数据速率的 1/20。只有时钟频率处于一定范围内，才能被收发器提取出来，各项条件如表 10-7 所示。

表 10-7 时钟和数据恢复电路的参数范围

参数		最小值	典型值	最大值	单位	条件
频率范围	RXP/RXN 串行差分信号频率	175		1000	MHz	峰—峰测量结果
频率偏置					ppm	
T_{DCREF}	信号占空比	45	50	55	%	
T_{RCLK}/T_{FCLK}	REFCLK 的上升和下降时间			75	ps	
T_{GITT}	REFCLK 的全部抖动			40	ps	峰—峰测量结果
T_{LOCK}	时钟恢复时间		10		μs	
T_{UNLOCK}					周期	

为了使 CDR 正常工作，在数据流里要有足够多的跳变发生。此外，还要保证 CDR 电路可以与 8b/10b 编码协同工作。在上电时，CDR 大概需要 5000 次跳变才可以锁定到输入数据速率上。一旦完成了该锁定，在丢失对输入数据流的锁定之前，允许遗漏 75 个跳变。如

果用户对其串行数据流进行设计，那么必须注意满足 8b/10b 编码所需的跳变频率。CDR 的另一个特征是它可以接收一个外部精确时钟(REFCLK)作为可选的输入，REFCLK 要么是输入数据的时钟，要么是通过同步得到的 RXRECCLK。更明确地讲，对于来自 FPGA 核且送往 TX FIFO 的数据，使用 TXUSRCLK 来作为它们的时钟。FIFO 的深度决定了这两个时钟之间的微小相位差别，如果时钟在频率上得以锁定，FIFO 就相当于一个直通的缓存。

2. PCB 设计要求

为了保证 Rocket I/O 收发器的可靠操作，设计者必须注意满足 PCB 设计要求，主要涉及电源过滤网络、高速差分信号路径和参考时钟等方面。如果不能满足下列要求，则 Rocket I/O 可能无法正常工作。

1) 电源以及电源滤波电路

每个 Rocket I/O 收发器包括 5 个电源管脚，所有的管脚对噪声的影响都比较敏感。为保证电路可靠工作，Rocket I/O 收发器需要和外围噪声源进行一定的隔离，需要专门的供电。为 Rocket I/O 收发器供电的电压调节器可以和收发器中其他有相同供电电压的管脚共用，但是不能和其他部分电路共用电源，每一个电源管脚必须有自己独立的 LC 滤波网络。表 10-8 中列出了其管脚名以及相应的电压。

表 10-8　收发器的电源

电　　源	2.5V	1.5V~2.8V	描　　述
AVCCAUXRX	X		模拟 RX 的电源
AVCCAUXTX	X		模拟 TX 的电源
VTRX		X	RX 终端的电源
VTTX		X	TX 终端的电源
GNDA			发送的模拟地和接收的模拟电源

为了确保 Rocket I/O 收发器正常工作，必须对周围的噪声源进行隔离。因此，对电源芯片有一定的要求，即需要用专用电压校准器来给 Rocket I/O 供电。这些电源电路不能被其他任何电源所共享(包括 FPGA 电源 VCCINT、VCCO、VCCAUX 和 VREF)，但电压校准器可以在有相同电压的收、发电源之间共享。这里，所需的电压校准器型号为凌特公司(Linear Technology)的 LT1963-2.5。在原理图设计时，必须按照数据手册上的说明来完成，如图 10-22 所示的 2.5V 电源的示意图，该电源可用于 AVCCAUXRX 和 AVCCAUXTX。读者可以参考网站 http://www.linear-tech.com 以获得有关该设备的更多信息。

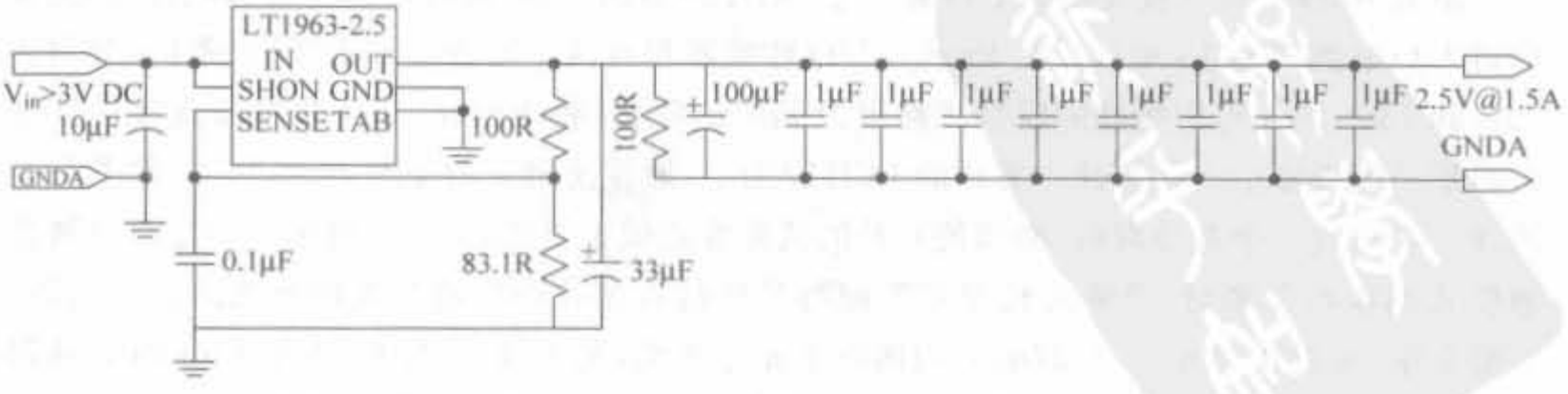


图 10-22　Rocket 电源电路示意图

为了达到所需的电源噪声隔离，需要在电源管脚处设置滤波网络。在图 10-23 中给出了这些电容器和铁氧体磁珠电路的结构。

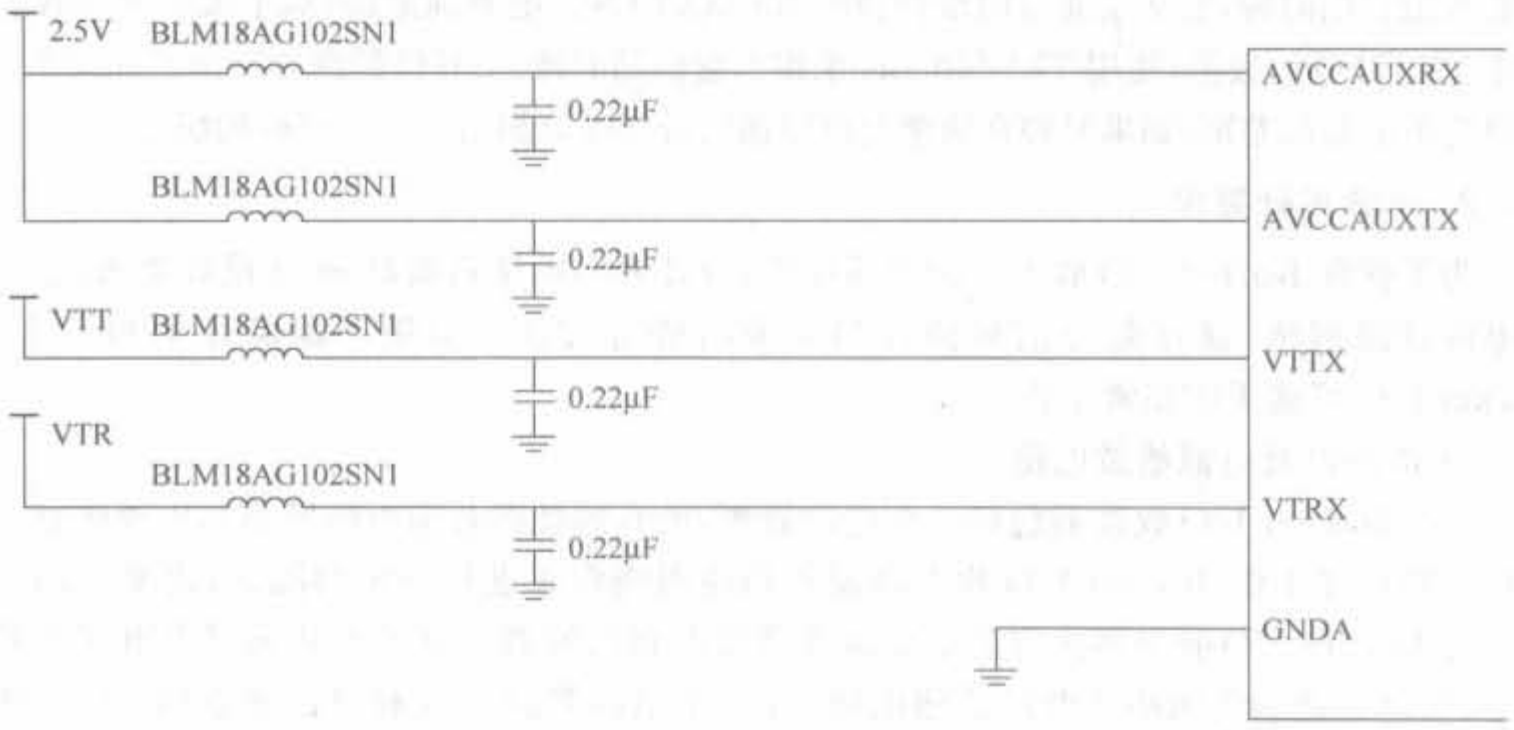


图 10-23　收发器的电源滤波电路示意图

每个收发电源管脚都需要一个电容器和一个铁氧体磁珠来构成滤波电路，其中电容为 0.22μF，在 X7R 绝缘材料的 0603 SMT 封装中，容许误差为 10%，电压为 5V，且必须放在距离管脚的 1cm 内。磁珠型号为 Murata BLM18AG102SN1。

为验证已完成的 PCB 上的差分阻抗，不仅要利用时域反射(Time Domain Reflectometry，TDR)来进行测量，还要对 PCB 上的电源和地的线路进行设计，使其有最小的自感系数。为了达到这一目的，必须添加地线专用层。如果转接线路是电源分布线路的组成部分(当连接一个旁路电容到它相应的电源和地层时，就会发生这一现象)，必须使用多个转接线路来降低电源分布线路上总的自感。如果电源滤波电容与 FPGA 在板子的同一侧，设计者可使电源层更靠近板子的表面，缩短电源线的路径长度。

2) 高速布线设计

对于高速设计来讲，正确布线是一项非常艰巨的任务，因为要克服串扰、电磁辐射、阻抗匹配以及等长线等诸多不利现象。下面对 Rocket I/O 的布线进行说明。

(1) 串行路径的布线

所有 Rocket I/O 收发器的 I/O 都位于 BGA 封装的外围，这可方便布线和检查(因为在串行 I/O 管脚没有 JTAG)。在 Rocket I/O 收发器里有两个可选的输出/输入阻抗，分别为 50Ω 和 75Ω，因此可用受控的阻抗线路以及相应的阻抗，将 Rocket I/O 收发器连接到其他可匹配的收发器上。在芯片—芯片的 PCB 应用中，推荐使用 50Ω 的终端和 100Ω 的差分传输线。在布线一个差分对时，必须使互补的线路在走线上等长，否则长度的不匹配会导致普通模式的噪声和辐射，严重的长度不匹配将导致接收端处的抖动和无法预测的时序问题。一般来讲，如果在 50mils(1.27mm)内匹配了差分线路，是比较稳定的。由于 FR4 PCB 线路中的信号大概以每英寸 180ps 的速度传送，因此 50mils 的差别会导致大约 9ps 的时序偏移。在具体板的设计时，可以利用相应的 CAD 工具来验证上述假设条件。

所有的信号线路在其之下都必须有一个完整的参考面，该面板可采用带状线或微波传输带结构。而参考面到线路任何一边的延伸距离不可以超过5倍线路宽度，以保证可以预测传输线的特性。差分对的布线要以点到点的最优方式来完成，并且最好在相同的PCB布线层上，这是由于转接线路会带来阻抗的不连续性。因此，应当尽可能地避免层到层的变化，但可以横穿PCB的多层结构而直接到达发送和接收封装的管脚。

如果串行线路必须换层，那么必须注意应当保证有一个完整的电流回环路径。因此，高速串行线路的布线必须在信号层上，而这些信号层共享一个参考面。如果信号层不共享一个参考面，那么必须在两个参考面板间跨接一个0.01μF的电容，并且靠近于信号换层处的转接线路。如果两个参考层都是DC耦合的，它们可以通过转接线路来连接，且该线路靠近信号换层的位置。

为了控制串扰，串行差分线路必须至少与其他PCB布线间隔5个线路宽度，这里的其他PCB布线包括了其他串行对。如果其他PCB布线承载噪声很大的信号，如TTL以及类似的布线，该间距必须更大。利用两个高带宽的连接器，即可让Rocket I/O收发器在20英寸的FR4上实现3.125Gb/s的传输速率。如果线路更长，需要一个低损耗的绝缘体或者是更宽的串行线路。

(2) 差分信号的走线设计

Rocket I/O收发器的高速差分信号线应尽量选用性能良好的微波传输线和带状线。由于传输的差分信号频率很高，两根差分信号线必须在长度上尽量匹配，长度失配会产生共模噪声和辐射，严重的失配会产生时钟抖动(Jitter)和不可预知的时序问题。差分线必须尽量匹配终端电阻50Ω和75Ω，可选50Ω用于芯片和芯片之间互连，75Ω用于芯片和电缆之间互连。高速差分线不要打孔，要布在电路板中的同一层。

Rocket I/O收发器需要100Ω或150Ω的差分线路阻抗(取决于是否选择了50Ω或75Ω终端的可选项)。但实际上，差分线路对的特征阻抗不仅与每条线路的宽度有关，还与两者间的距离有关。为了实现这一差分阻抗要求，每个线路的特征阻抗必须稍稍高于目标差分阻抗的一半，通过电磁场仿真软件来确定适合于特定应用的准确线路结构，如图10-24所示。

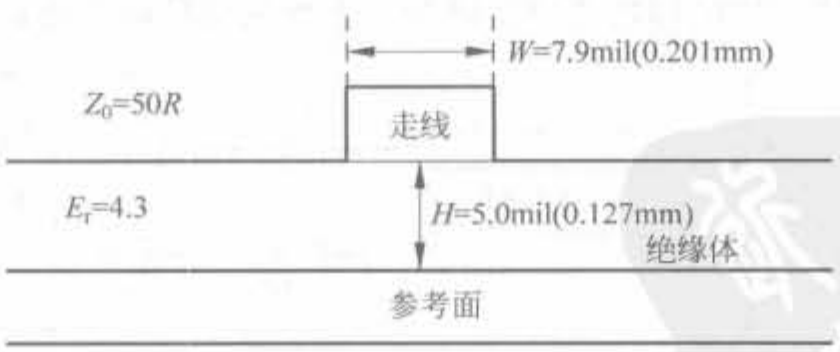


图10-24　单端路径和几何结构示意图

Xilinx公司推荐差分线路采用紧密耦合，相对于宽松耦合，它可使得两者在所有位置都与对方很靠近。由于紧密耦合线路的差分阻抗在很大程度上取决于它们和对方的靠近程度，因此它们的间距必须在所有位置都保持不变。如果要使布线通过一个管脚区域或其他PCB障碍，必须分离线路。一个有效的办法是在靠近障碍处修改线路结构，纠正阻抗的不

连续性(在线路分离处,增加每个线路的宽度)。图 10-25 和图 10-26 给出了微带线和带状线差分信号的 PCB 几何结构的示意图,这两种几何结构具有 100Ω 的差分阻抗。

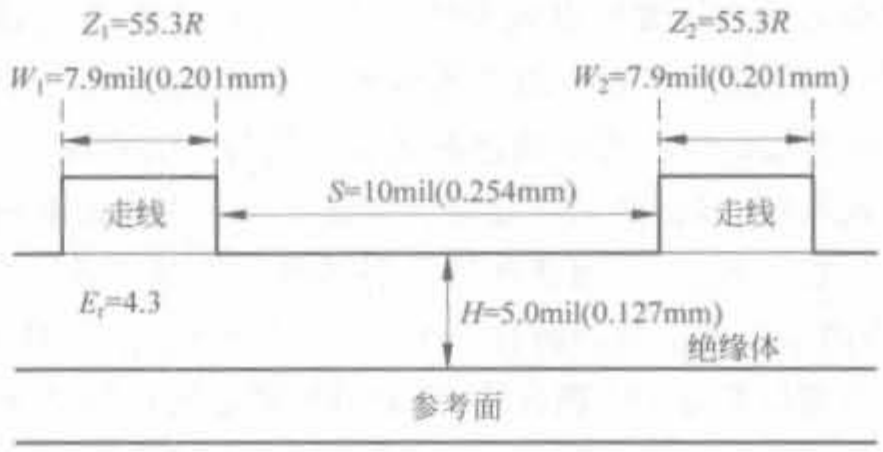


图 10-25 微带线边沿耦合差分对连接示意图

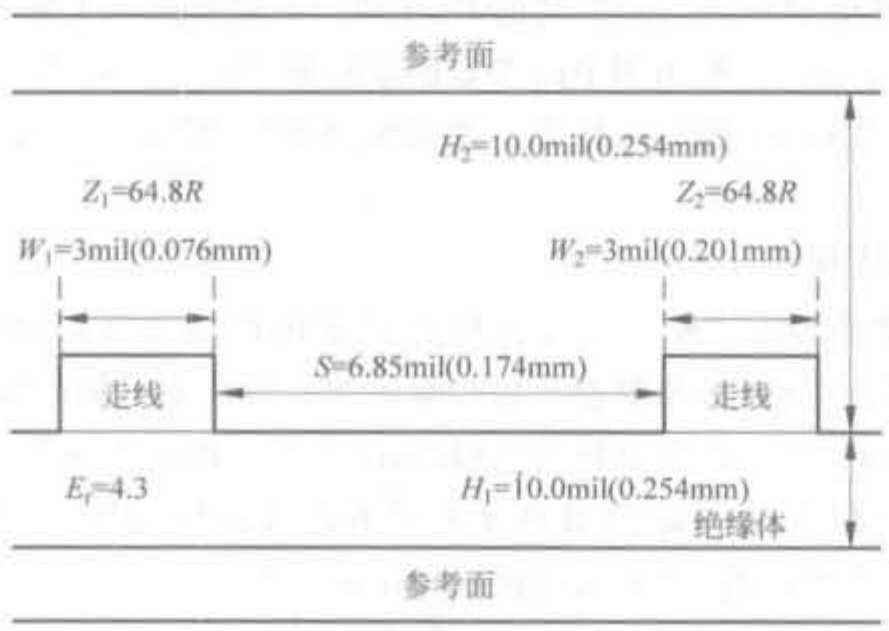


图 10-26 带状线边沿耦合差分对连接示意图

(3) AC、DC 耦合

在收发器差分电压兼容时,必须使用 AC 耦合(在信号路径上使用 DC 阻塞电容),但是对于普通模式的电压就不需要了。此外,某些设计需要 AC 耦合以适用热插拔,和/或在不同收发器处有不同电源电压也都需要 AC 耦合。典型的 AC 耦合电路如图 10-27 所示。

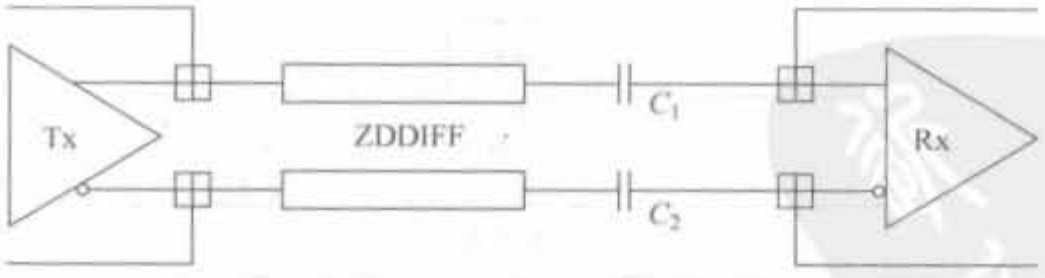


图 10-27 AC 耦合电路示意图

在 Rocket I/O 收发器与其他 Rocket I/O 收发器,或其他 Mindspeed 的收发器(该收发器有兼容的差分和普通模式电压规格)连接时,DC 耦合更为可取。在使用 DC 耦合时,不需要额外的无源器件。典型的 DC 耦合电路如图 10-28 所示。

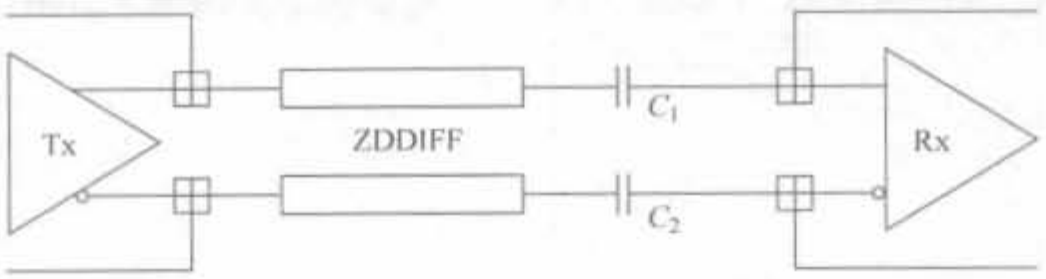


图 10-28　DC 耦合电路示意图

10.3.6　Rocket I/O IP Core 的使用

1. 支持 Rocket I/O 芯片族

如前所述，Rocket I/O 作为 Xilinx FPGA 芯片中内嵌的硬件模块，并不是任何一款 FPGA 都提供的，只有在 Virtex-2 Pro 以上的部分高端 FPGA 内部才具备。支持 Rocket I/O 的 FPGA 型号如表 10-9 所示。

表 10-9　内嵌 Rocket I/O 的 FPGA 芯片型号列表

芯片系列	芯片族	Rocket I/O 最大个数	制造工艺	芯片种类数
Virtex-2 Pro	Virtex-2	24	130nm	10
Virtex-4 FX	Virtex-4	24	90nm	5
Virtex-5 LXT	Virtex-5	24	65nm	6
Virtex-5 SXT	Virtex-5	16	65nm	3

2. Rocket I/O IP Core 的配置页面

Rocket I/O 和 DCM 模块、硬核乘法器、块 RAM 一样，属于 FPGA 内部的集成固件，可通过原语或者 IP Core 调用，其相关参数已在 10.3.3 节、10.3.4 节进行了详细说明。

该 IP Core 位于“FPGA Feature and Design”→“I/O Interfaces”目录下，后缀为. xaw，表明其属于 FPGA 底层基本组件单元。Rocket I/O 的 IP Core 有 5 个配置向导页面，分别如图 10-29(a)～(e)所示。

1) 配置页面 1——General Setup

本页面主要用于配置收发器工作模式、参考时钟以及终端阻抗。在“Select the transceiver”栏下拉框的可选工作模式有：Aurora 协议、Custom、Ethernet、FibreChannel、Infiniband 以及 XAUI 等 6 种；在“Data Width”下拉框可设定数据位宽，有 1Byte、2Byte 和 4Byte 等 3 种选择；选中“Transmitter”和“Receiver”复选框，表示在 Rocket I/O 硬核中使能相应的组件，否则旁路掉相应的组件，默认值为都选中。“Reference clock selection”栏用于选择时钟，有 CLKREF 和 BCLKREF 两种选择。当收发器的工作频率高于 2.5Gb/s 时，必须选择 BCLKREF 时钟；选中“Bypass PCS features(Uses SERDES with CDR Only)”复选框，可旁路 PCS 功能；选中“Half-rate Mode”复选框，可启动半速率模式。“Termination impedence”栏用于配置终端阻抗，有 50Ω 和 75Ω 两种选择。

(a) Rocket I/O配置向导1

(b) Rocket I/O配置向导2

(c) Rocket I/O配置向导3

(d) Rocket I/O配置向导4

(e) Rocket I/O配置向导5

图 10-29　Rocket I/O IP Core 的用户配置界面

2）配置页面 2——Transmitter Setup

本页面主要配置发送端功能。“Encoding”栏用于线路编码配置栏，有 Disable encoding（旁路编码器）、Dynamic encoding control（动态控制编码器）以及 Static encoding control（使能 8B/10B 编码）等 3 个选项。“Pre-emphasis”栏为预加重控制栏，从弱到强，有 10%、20%、25%、33%等 4 类强度。“Output Voltage swing”为输出电压摆动功率设置栏，有 400mW、500mW、600mW、700mW 以及 800mW 等 5 级数值。“CRC”栏用于配置发送端校验编码器，选中“Use CRC for transmitter”复选框后，才能配置相关参数；“CRC Format”栏用于选择校验模式，有 User-Mode、Ethernet、FibreChannel 以及 Infiniband 等 4 种模式；“Start-Of-Packet K-Parameter”用于选择起始包的 K 参数；“End-Of-Packet K-Parameter”用于选择结束包的 K 参数。

3）配置页面 3——Receiver Setup

本页面主要配置接收端功能。“Decoding”栏的参数和配置页面两种“Encoding”栏的选

项是一致的，接收端选择的解码方式由输入数据在远端的编码方式决定，二者必须保持一致。“8B/10B Synchronization”栏用于设定检测同步头，选中“Detect the loss of sync on RXLOSSODSYNC”复选框，可设定同步操作的有效步长以及门限值。同样，“USE CRC for recevier”复选框由输入数据决定，如果输入数据经过编码，则选中该复选框。

4）配置页面4——Receiver Clock Correction Setup

本页面用于配置接收机时钟修正参数。选中“Retain at least one clock correction sequence”可以启用至少保持一个时钟修正序列的功能。“The minimum number of RXUSERCLK without clock”栏用于设定没有时钟时，最小的 RXUSERCLK 周期数。“Length of clock correction sequence(in Byte)”栏用于设定时钟修正序列的长度，可选值有1、2、3 和 4，单位为 Byte。“Define clock correction sequence(11-bit vector)”栏用于设定时钟修正序列。

5）配置页面5——Summary

本页面主要显示用户定义 Rocket I/O 的特性。“Feature Summary”栏用于显示用户配置参数的主要特征，如参考时钟的配置。“Files To Be Generated”栏用于显示即将生成的文件，如 my_rocketio. xaw。“Block Attributes”栏用于显示模块属性，如 CRC 起始包和结束包的 K 字符、发送数据宽度等。“Show all modifiable attributes”复选框用于显示所有可修改的属性。“Show only the modifiable attributes whose values differ form the default”复选框只显示和默认值不同的可修改属性。

配置完成后，可在工程管理区选中“Rocket I/O”，在过程管理区单击“View HDL Instantiation Template”命令查看其例化代码，代码中的例化方法和一般 IP Core 的方法是一样的。

10.4 基于 Xilinx FPGA 的千兆以太网控制器的开发

目前多媒体应用对网络带宽提出了极大的挑战，因此千兆以太网应运而生。千兆以太网是建立在以太网标准基础之上的技术，提供千兆的带宽，同时完全兼容目前大量使用的10M/100M 以太网。本节主要介绍基于 FPGA 的千兆以太网开发技术。

10.4.1 千兆以太网技术

1. 千兆以太网技术简介

以太网技术是当今应用广泛的网络技术，千兆以太网技术继承了以往以太网技术的许多优点，又具有许多新的特性，例如传输介质包括光纤和铜缆，使用 8b/10b 的编解码方案，采用载波扩展和分组突发技术等。正是因为具有良好的继承性和许多优秀的新特性，千兆以太网已经成为目前局域网的主流解决方案。

千兆以太网利用了原以太网标准所规定的全部技术规范，其中包括 CSMA/CD 协议、以太网帧、全双工、流量控制以及 IEEE 802.3 标准中所定义的管理对象。千兆以太网的关键技术是千兆以太网二层(MAC 层)的交换与以太网接口的实现。随着多媒体应用的普

及，千兆以太网必将成为各类以太网技术的主力军。

2. Xilinx 的千兆以太网解决方案

1）IP 的支持

Xilinx 提供了可参数化的 10/1Gb/s 以太网媒体访问控制器功能 LogiCORE 解决方案。该核设计用来同最新的 Virtex-5、Virtex-4 和 Virtex-Ⅱ Pro 平台 FPGA 一起工作，并可以无缝集成到 Xilinx 设计流程中。吉比特级以太网媒体访问控制器核（GEMAC）是针对 1Gb/s 以太网媒体访问控制器功能的可参数化的 LogiCORE IP 解决方案。GEMAC 核的设计符合 IEEE 802.3-2002 规范。GWMAC 核支持两个 PHY 端接口选项：GMII 或 RGMII。并且，Xilinx 全面的 1Gb/s 以太网解决方案包含吉比特 MAC 和 PCS/PMA IP 核产品。Xilinx 吉比特以太网 MAC 解决方案还包括带有内置处理器本地总线（PLB）接口（PLB GEMAC）的配置，该配置通过 Xilinx 嵌入式开发套件（EDK）提供。GEMAC LogiCORE IP 可以实现与 1000 Base-X PCS/PMA 或 SGMII 核的无缝集成，并提供 3 种选项用来与 PHY 器件接口：1000 Base-X 或 10 位接口（TBI）或 SGMII。

GEMAC 核非常适合开发高密度吉比特级以太网通信和存储设备，其关键特性有：

- 单速全双工 11Gb/s MAC 控制器；
- 设计符合 IEEE 802.3-2002 规范；
- 具有最小缓冲的直通操作，以最大限度地实现客户端接口的灵活性；
- 通过可选的独立微处理器中的接口进行配置和监控；
- 直接与以太网统计数据核接口，以便实现功能强大的统计数据收集；
- 通过 MAC 控制暂停帧实现对称的或非对称的可选的流程控制；
- VLAN 帧的可选技术支持符合 IEEE 802.3-2002 规范的要求；
- 支持任意长度的“jumbo 帧”（可选）；
- 可选的地址滤波器，具有数量可选的地址表输入。

2）相应的开发板套件

Xilinx 提供的千兆以太网开发套件为 Virtex-5 ML505/ML506 开发板（使用的 FPGA 芯片为 XC5VLX50T-1FF1136）。该开发板支持 10/100 兆、1/10 吉以太网，加上 Xilinx 公开的基于 ML505/506 的设计，可为基于以太网开发的设计提供全方位的参考。此外，ML505/506 还具备 SFP、PCI E、SATA 以及 SMA 接口等其余吉比特接口，是学习和研发高速连接设备的理想平台。

10.4.2 基于 FPGA 的千兆以太网 MAC 控制器实现方案

1. 整体设计方案

以太网控制器的 FPGA 设计工作包括以太网 MAC 子层的 FPGA 设计、MAC 子层与上层协议的接口设计以及 MAC 与物理层（PHY）的 MII 接口设计。该以太网控制器的总体结构设计框图如图 10-30 所示。整个系统分为发送模块、接收模块、MAC 状态模块、MAC 控制模块、MII 管理模块和主机接口模块 6 个部分。发送模块和接收模块主要提供

MAC 帧的发送和接收功能，其主要操作有 MAC 帧的封装与解包以及错误检测，它直接提供了到外部物理层芯片的并行数据接口。在实现中，物理层处理直接利用商用的千兆 PHY 芯片，主要开发量集中在 MAC 控制器的开发上。

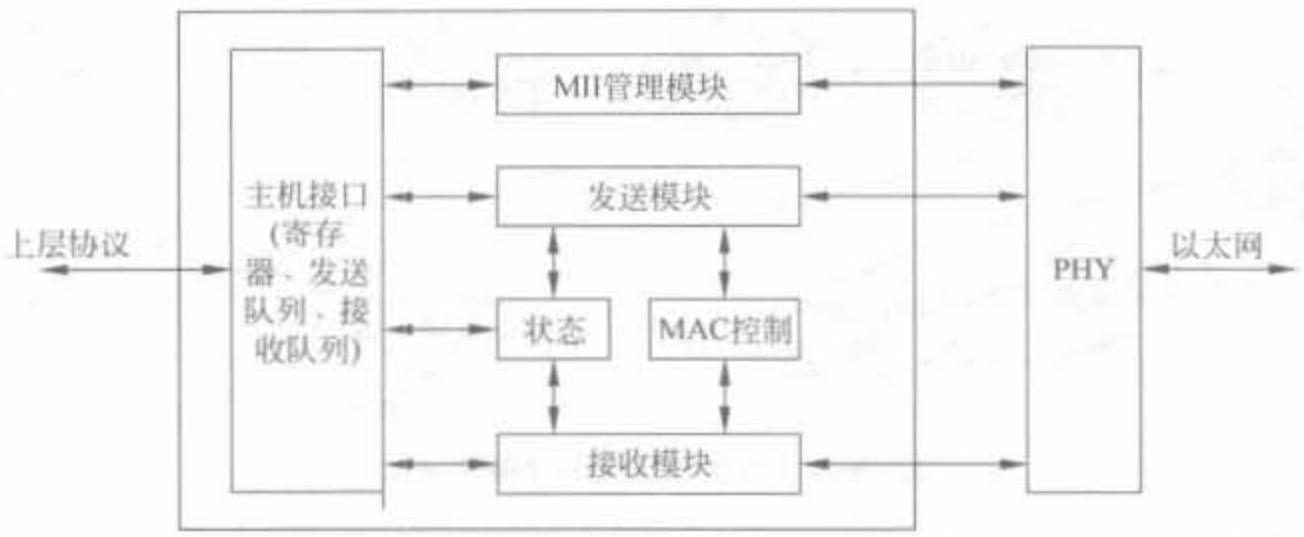


图 10-30　以太网控制器的结构设计框图

MAC 控制模块用于执行全双工模式中的流量控制功能。MAC 状态模块可用来监视 MAC 操作过程的各种状态信息，并作修改。MII 管理模块提供了标准的 IEEE 802.3 介质独立接口，可用于连接以太网的链路层与物理层。主机接口则提供以太网控制器与上层协议(如 TCP/IP 协议)之间的接口，用于数据的发送、接收以及对控制器内各种寄存器(控制、状态和命令寄存器)的设置。

2. MAC 控制器结构和工作流程

1) MAC 发送模块

MAC 发送模块可将上层协议提供的数据封装之后通过 MII 接口发送给 PHY。发送模块可接收主机接口模块的数据帧开始和数据帧结束标志，并通过主机接口从外部存储器中读取要发送的数据，然后对数据进行封装，再通过 PHY 提供的载波侦听和冲突检测信号，在信道空闲时由 MII 接口将数据以 4 位的宽度发送给 PHY 芯片，最后由 PHY 将数据发送到网络上。

发送模块由 CRC 生成模块(crc_gen)、随机数生成模块(random_gen)、发送计数模块(tx_cnt)和发送状态机(tx_statem_machine)模块等 4 个主要子模块组成，其内部结构如图 10-31 所示。

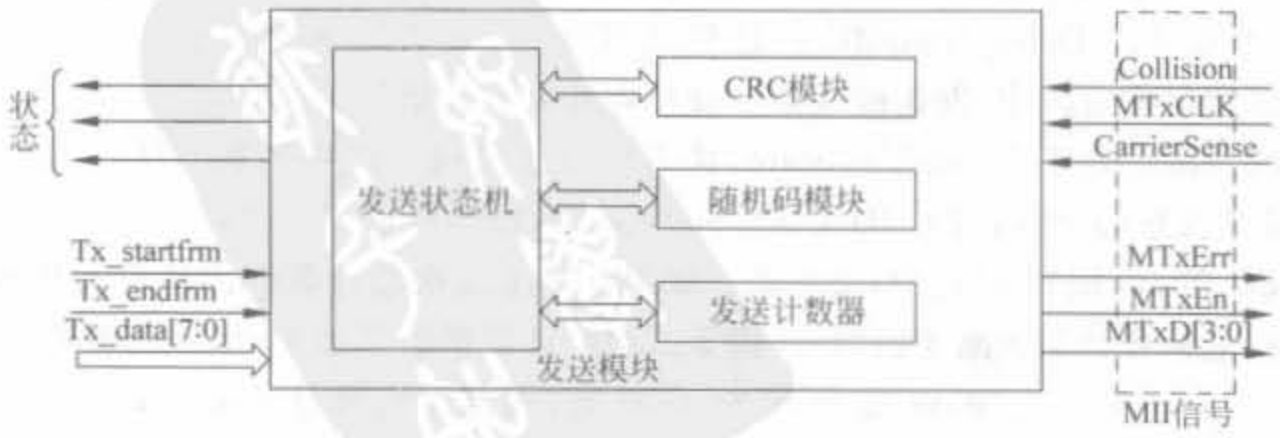


图 10-31　以太网 MAC 发送模块的结构示意图

发送状态机由 Idle_State、Preamble_State、Data0_State、Data1_State、PAD_State、FCS_State、IPG_State、Jam_State、BackOff_State、Defer_State 等 10 个状态组成，其状态转移图如图 10-32 所示。

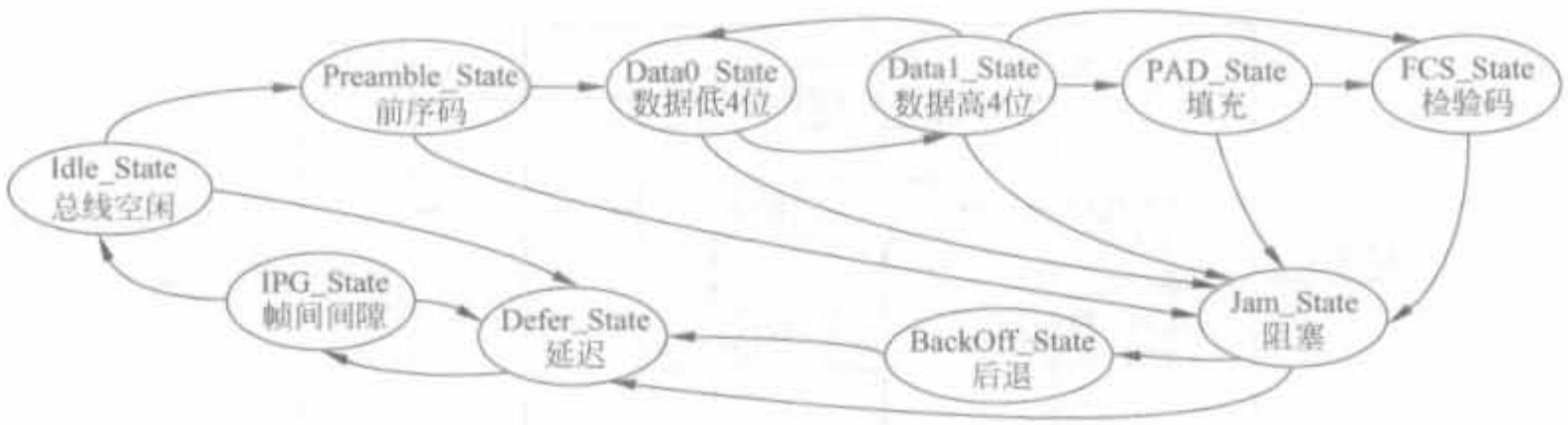


图 10-32　发送状态机的状态转移图

系统复位后，发送模块即进入 Defer_State 状态，并一直检测载波侦听(CarrierSense)信号。当载波侦听信号变成无效(表示信道空闲)时，状态机进入 IPG_State 状态。在等待一个帧间间隙之后，状态机进入 Idle_State 状态。如果在帧间间隙的前 2/3 个周期检测到信道忙信号，状态机将重新回到 Defer_State 状态。

状态机进入 Idle_State 状态之后，发送模块将检测载波侦听信号和主机接口的发送请求。若主机模块请求发送，状态机将进入 Preamble_State 状态，发送模块即通知 PHY 发送开始，同时开始发送前序码(7 个 0x5)，然后发送帧起始定界符(SFD，0xd)。状态机进入 Data0_State 后，发送模块将发送一个数据字节的低 4 位(LSB nibble)，当其进入 Data1_State 状态后，发送模块发送数据字节的高 4 位(MSB nibble)。随后，状态机一直在 Data0_State 和 Data1_State 之间循环，直到数据发送完毕。当还剩一个字节时，主机模块将通过发送帧结束信号来通知发送模块。如果数据帧的长度大于最小帧并且小于最大帧，状态机就进入 FCS_State 状态。此时，发送模块将 CRC 生成模块生成的 CRC 值添加到帧的 FCS 字段中并发送给 PHY。

帧发送完之后，状态机进入 Defer_State 状态，之后是 IPG_State 和 Idle_State 状态。此后，状态机又回到初始状态，以重新等待新的发送请求。如果数据帧的长度小于最短帧，状态机就进入 PAD_State 状态，发送模块根据系统设置确定是否在数据之后添加填充码。然后，状态机进入 FCS_State 状态。如果数据帧的长度大于最大帧，而系统设置又支持发送超长帧，那么，状态机就进入 FCS_State 状态；如果不支持发送超长帧，发送模块将放弃发送，状态机直接进入 Defer_State 状态，然后是 IPG_State 状态，最后回到 Idle_State 状态。

在发送数据的过程中，发送模块会一直检查冲突检测信号(collision detected)。如果发现冲突且状态机正处于 Preamble_State，状态机将在发送完前序码和 SFD 之后进入 Jam_State，并发送拥塞码，然后进入 BackOff_State 状态，以等待重试。之后，状态机经过 Defer_State 和 IPG_State 回到 Idle_State 状态。如果此时重试次数计数器的值没有达到额定值，发送模块将重新开始发送刚才的帧，并将重试次数计数器的值加 1；如果发现冲突且状态机处于 Data0_State、Data1_State 或 FCS_State 状态，而且没有超过冲突时间窗，状态机将马上进入 Jam_State 状态发送拥塞码，之后经过 BackOff_State、Defer_State、IPG_State，回到 Idle_State，并根据重试计数器的值决定是否重新发送刚才的数据帧；如果检测到发生冲突

的时间超过了冲突时间窗，状态机将进入 Defer_State 状态，然后经过 IPG_State 到 Idle_State 状态，并放弃重试。

在全双工模式中发送帧时，不会进行延迟(defer)，发送的过程中也不会产生冲突。此时，发送模块将忽略 PHY 的载波侦听和冲突检测信号。当然，帧与帧之间仍然需遵守帧间间隙的规则。因此，全双工模式下的发送状态机没有 Jam_State、BackOff_State 和 Defer_State 3 个状态。

2) MAC 接收模块

MAC 接收模块结构如图 10-33 所示，负责数据帧的接收。当外部 PHY 将通信信道的串行数据转换为半字节长的并行数据并发送给接收模块后，接收模块会将这些半字节数据转换为字节数据，然后经过地址识别、CRC 校验、长度判断等操作，再通过主机接口写入外部存储器，并在主机接口模块的接收队列中记录帧的相关信息。此外，接收模块还负责前序码和 CRC 的移除。

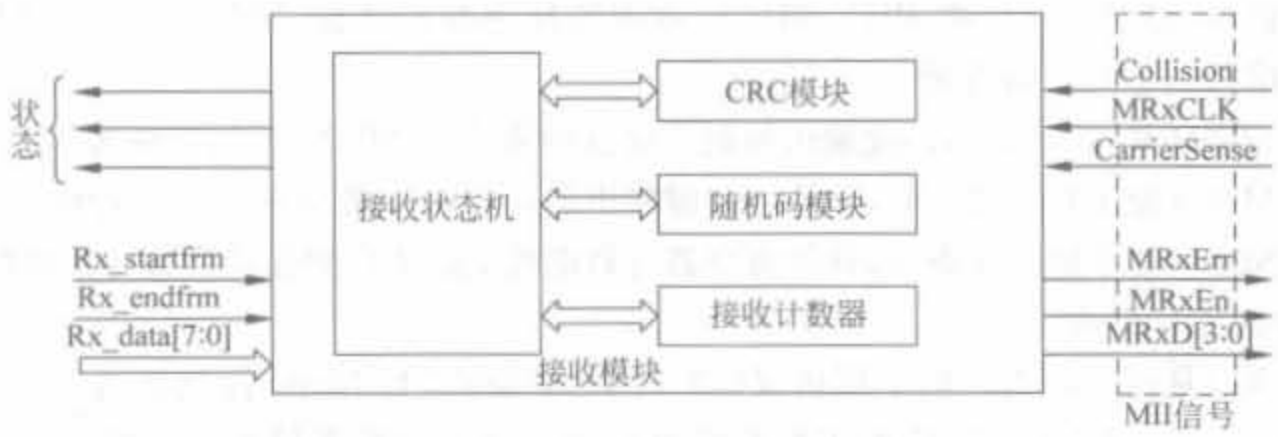


图 10-33　以太网 MAC 接收模块的结构示意图

接收过程的接收状态机由 Idle_State、Drop_State、Preamble_State、SFD_State、Data0_State、Data1_State 等 6 个状态组成，其状态转移图如图 10-34 所示。

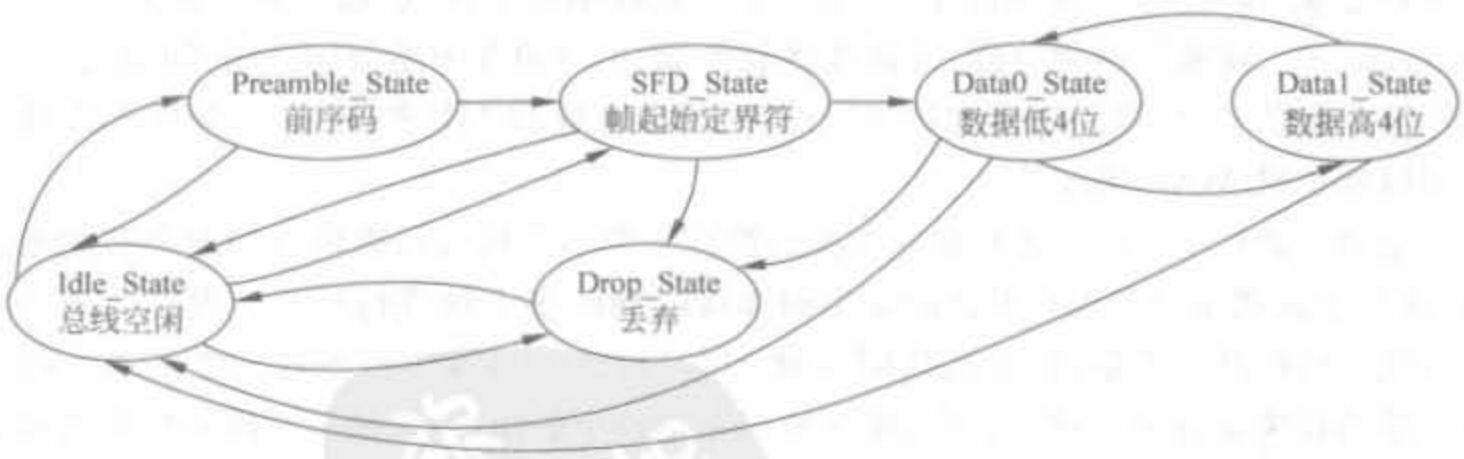


图 10-34　接收状态机的状态转移图

当接收模块检测到数据有效信号之后，状态机将进入 Preamble_State，并开始接收前序码。此后，状态机进入 SFD_State，接收一个字节的帧起始定界符，然后根据 IFGcnt 计数器的值进入不同的状态。如果 IFGcnt 所确定的时间大于 96 个比特时间，状态机将进入 Data0_State 状态以接收字节的低 4 位；然后是 Data1_State 状态，并接收字节的高 4 位，之后又回到 Data0_State 状态。状态机就一直在这两个状态之间循环，直到数据接收完毕(PHY 清除 MRxDV 信号)后进入 Idle_State，以重新等待接收新的数据。如果接收到帧起

始定界符，IFGcnt 计数器所确定的时间小于 96 个比特时间，那么状态机将进入 Drop_State 状态，并一直维持该状态直到数据有效信号结束(PHY 清除 MRxDV 信号)。之后，状态机再回到 Idle_State 等待接收新的数据。如果在接收前序码、帧起始定界符和数据期间，数据有效信号被清除，那么状态机将回到 Idle_State。

3) MII 管理模块

MII 管理模块用于控制 MAC 与外部 PHY 之间的接口，用于对 PHY 进行配置并读取其状态信息。该接口由时钟信号 MDC 和双向数据信号 MDIO 组成。MII 管理模块则由时钟生成模块、移位寄存器模块和输出控制模块 3 个部分组成。

时钟生成模块可以根据系统时钟和系统设置中的分频系数来产生 MII 管理模块的时钟信号 MDC(10Mb/s 速率时为 2.5MHz，100Mb/s 速率时为 25MHz)。移位寄存器模块既可用于对 PHY 的控制数据进行写入操作，也可用于对 PHY 的状态信息进行读出操作。写控制数据时，移位寄存器根据其他模块的控制信号将并行控制数据转换为串行数据；而在读状态信息时，移位寄存器将 PHY 的串行数据转换为并行数据，MAC 中的其他模块可将该并行数据写入适当的寄存器。

由于 MDIO 是双向信号，因此输出控制模块就用来决定 MDIO 是处于输入状态还是输出状态。当 MDIO 处于输出状态时，移位寄存器输出的串行控制数据在经过时钟同步后发送到 PHY；当 MDIO 处于输入状态时，移位寄存器将数据线上的串行数据转换为并行数据。

4) 主机接口模块

主机接口是运行以太网的上层协议(如 TCP/IP 协议)与 MAC 控制器的接口。通过该接口，上层协议可以设置 MAC 的工作模式并读取 MAC 的状态信息。该接口还可用于上层协议与 MAC 之间的数据交换。

主机接口模块内有一组寄存器，可用于存储上层协议对 MAC 设置的参数以及 MAC 的状态信息。上层协议对 MAC 设置的参数包括接收超短帧的使能、添加填充码使能、发送超长帧的使能、添加 CRC 校验值使能、全双工模式或半双工模式、超长延迟使能、混杂模式(Promiscuous)、接收广播帧使能、发送和接收使能、中断源和中断使能、帧间间隙的长度、最大帧和最小帧的长度、重试限制和冲突时间窗、MII 地址和 MII 控制命令、接收和发送队列的长度以及本机 MAC 地址等。

上层协议通过 MAC 发送和接收数据的操作主要由主机接口模块内的两个队列来进行管理，这两个队列用于对等待发送的多个帧和接收到的多个帧进行排队。发送队列主要记录等待发送的帧的相关信息、发送该帧时对 MAC 的设置以及该帧发送完之后产生的状态信息。帧的相关信息包括帧的长度、帧在外部存储器中的地址、该帧是否准备好发送以及队列中是否还有其他帧等待发送；对 MAC 的设置则包括中断使能、填充使能、CRC 使能；产生的状态信息包括成功发送之前的重试次数、由于达到重试限制而放弃发送、发送时产生的滞后冲突以及成功发送之前发生过的延迟。

接收队列主要对接收到的数据帧进行排队，并记录每个接收到的帧信息。这些信息包括帧的长度、是控制帧还是普通数据帧、帧中包含无效符号、接收到的帧太长或太短、发生 CRC 错误、接收的过程中发生滞后冲突、帧是否接收完、队列中是否还有其他已接收到的帧以及帧存储在外部存储器中的地址等。同时，队列中还有针对每个帧的设置位，用来设置是否在接收到帧时产生中断。发送队列和接收队列的长度都可以在控制寄存器中设置。

10.4.3　Xilinx 千兆以太网 MAC IP Core

1. GMAC IP Core 的应用场景和架构

Xilinx 提供了三态以太网 MAC 控制器的 IP Core，可实现单条吉比特以太网链路，通过交换机或路由器可与任意以太网端口相连。由于 MAC 控制器的速率很高，底层传输必须依赖 Rocket I/O，在客户端还需要利用 FIFO 来交换数据，其完整的设计方案如图 10-35 所示。

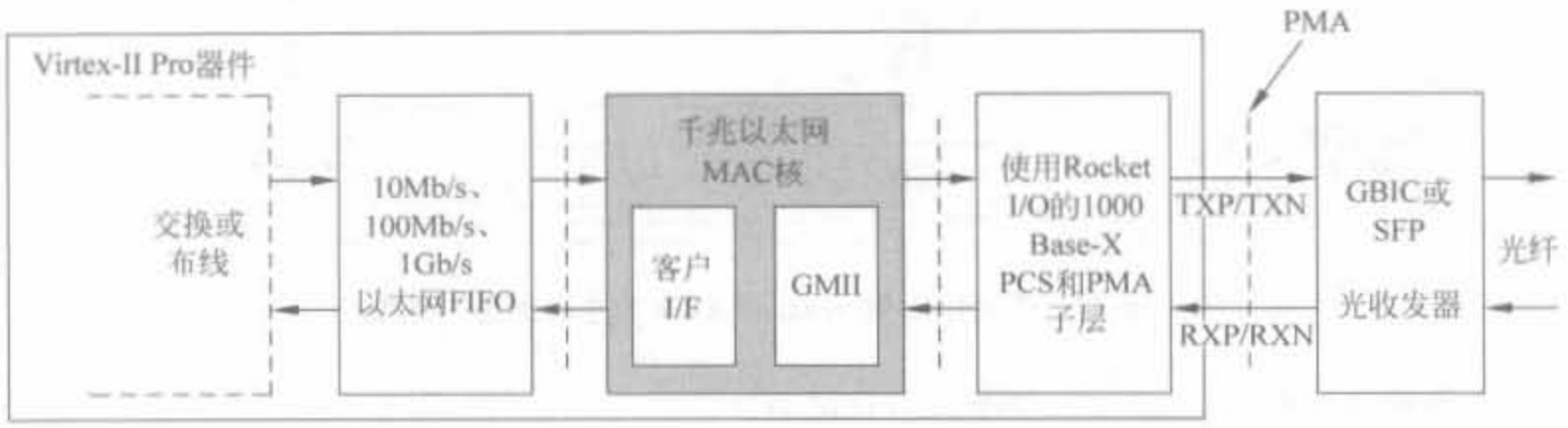


图 10-35　千兆以太网 MAC IP Core 的应用场合示意图

2. GMAC IP Core 的配置界面和接口信号

千兆以太网 MAC IP Core 的配置界面如图 10-36 所示。GEMAC 控制器所实现的主要功能如图 10-37 所示，包括发送引擎、接收引擎、流控制、GMII 接口、客户发送接口、客户接收接口以及客户管理接口。

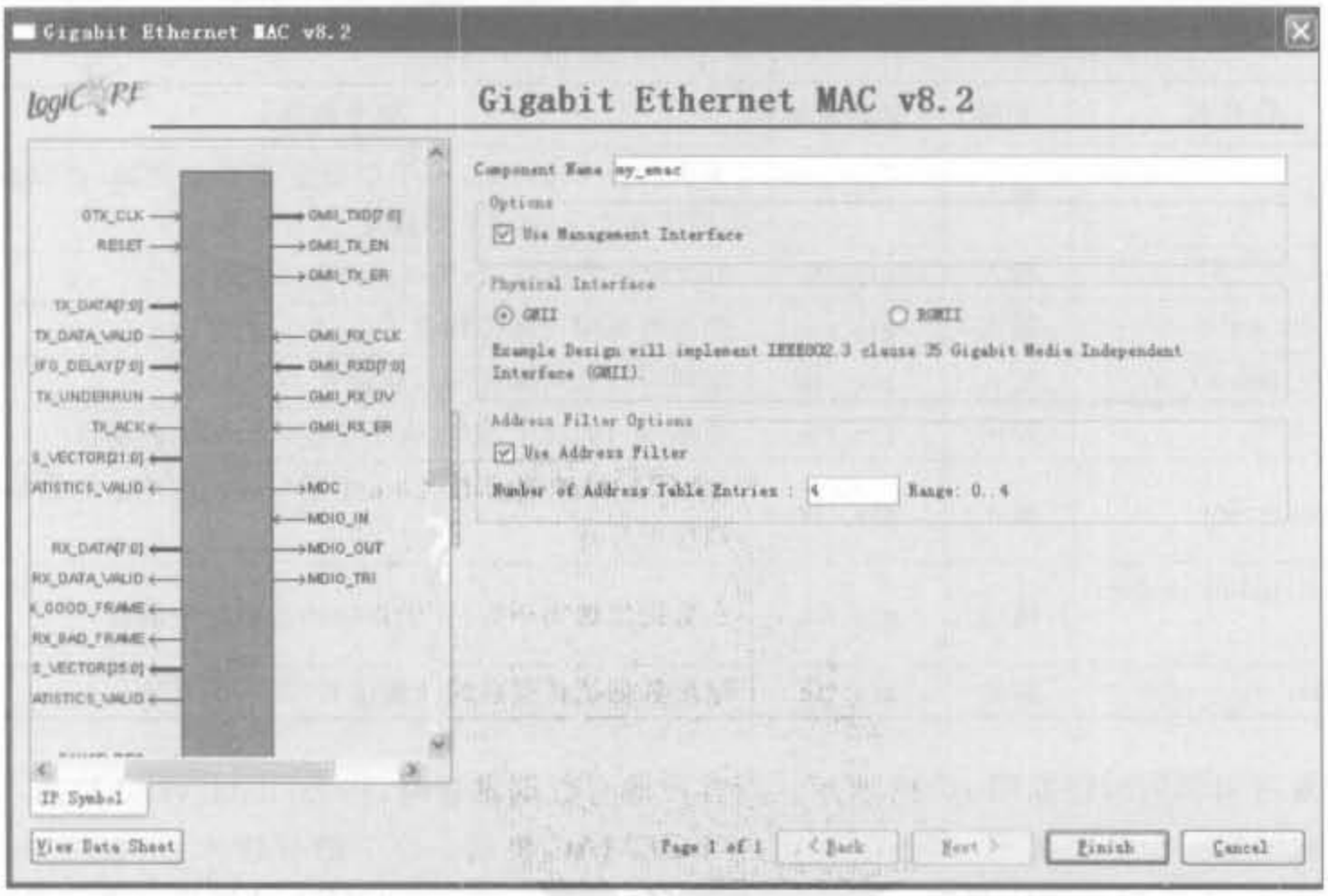


图 10-36　千兆以太网 MAC IP Core 的配置界面

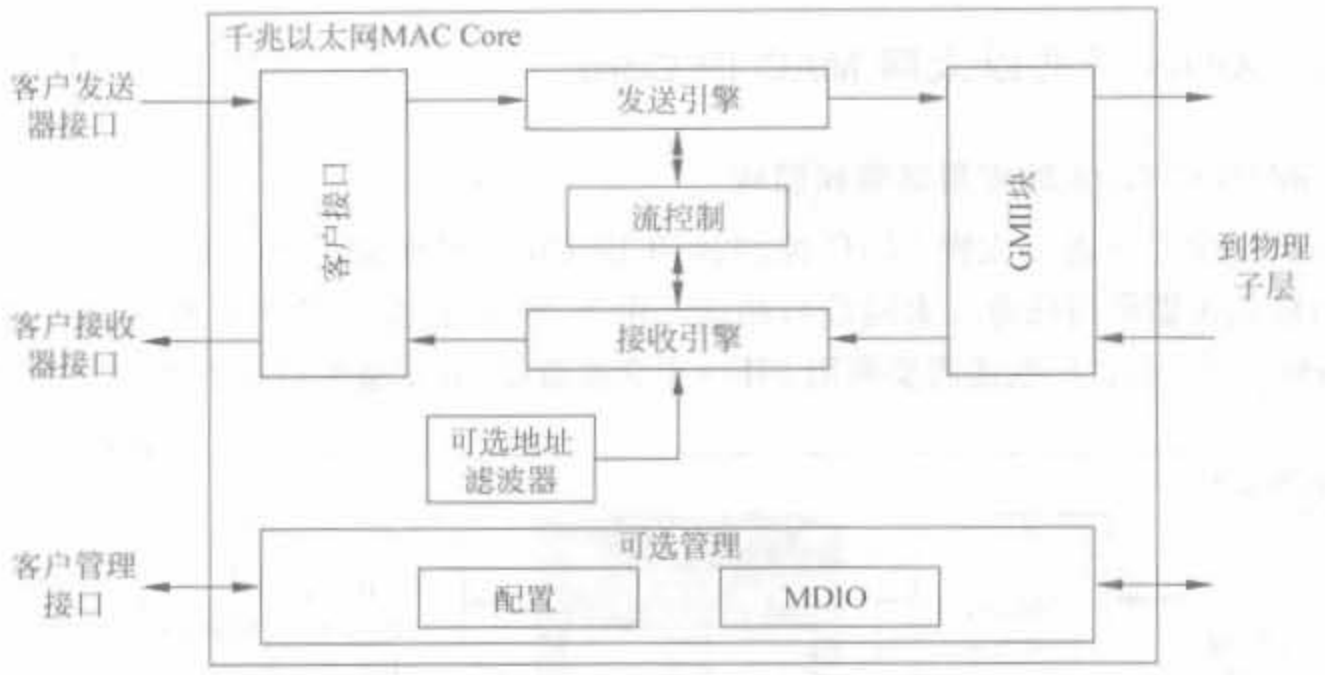


图 10-37　千兆以太网 MAC IP Core 的主要功能示意图

下面给出相应模块的功能和接口信号说明。

1）发送引擎

发送引擎通过客户发送端口接收以太网的数据帧，并在帧头添加帧引导区域，甚至在帧长小于最短要求时，添加一定的冗余比特。同时，该模块会在连续的数据帧之间插入以太网协议所规定的最小延时，将用户数据转化成标准的 GMII 数据帧，并送至 GMII 模块。在应用时，面向用户的操作就是将用户数据读入 GEMAC 核内部。

该模块接口信号的简要信息如表 10-10 所示。

表 10-10　发送引擎接口信号列表

信号名	方向	驱动时钟	简要描述
gtx_clk	输入	N/A	系统时钟 125MHz，用于驱动全部发送逻辑，必须满足 IEEE 802.3-2002 中的规定
tx_data[7:0]	输入	gtx_clk	8bit 发送数据端口，接收用户输入数据
tx_data_valid	输入	gtx_clk	发送数据端口的控制信号
tx_ifg_delat[7:0]	输入	gtx_clk	帧间可控延时调整的控制信号
tx_ack	输出	gtx_clk	数据端口的握手信号，表明已经将数据加载到核中
tx_underrum	输入	gtx_clk	强制 MAC 控制器 IP Core 中断当前正在传输的数据帧，高电平有效
tx_statistics_vector[21:0]	输出	gtx_clk	在数据发送完毕后，提供相应的数据统计信息
tx_statistics_valid	输出	gtx_clk	输出数据统计信息的使能信号，高电平有效

发送引擎的时序如图 10-38 所示。当客户端有数据发送时，将 tx_data_valid 拉高，同时将数据的第一个字节置于 tx_data 端口；当 GEMAC 将第一字节数据读入后，会将 tx_ack 信号拉高，用户端逻辑检测到 tx_ack 为高电平时，要在下一个时钟上升沿将其余的数据发送到数据端口上；当数据发送完毕后，将 tx_data_valid 拉低。

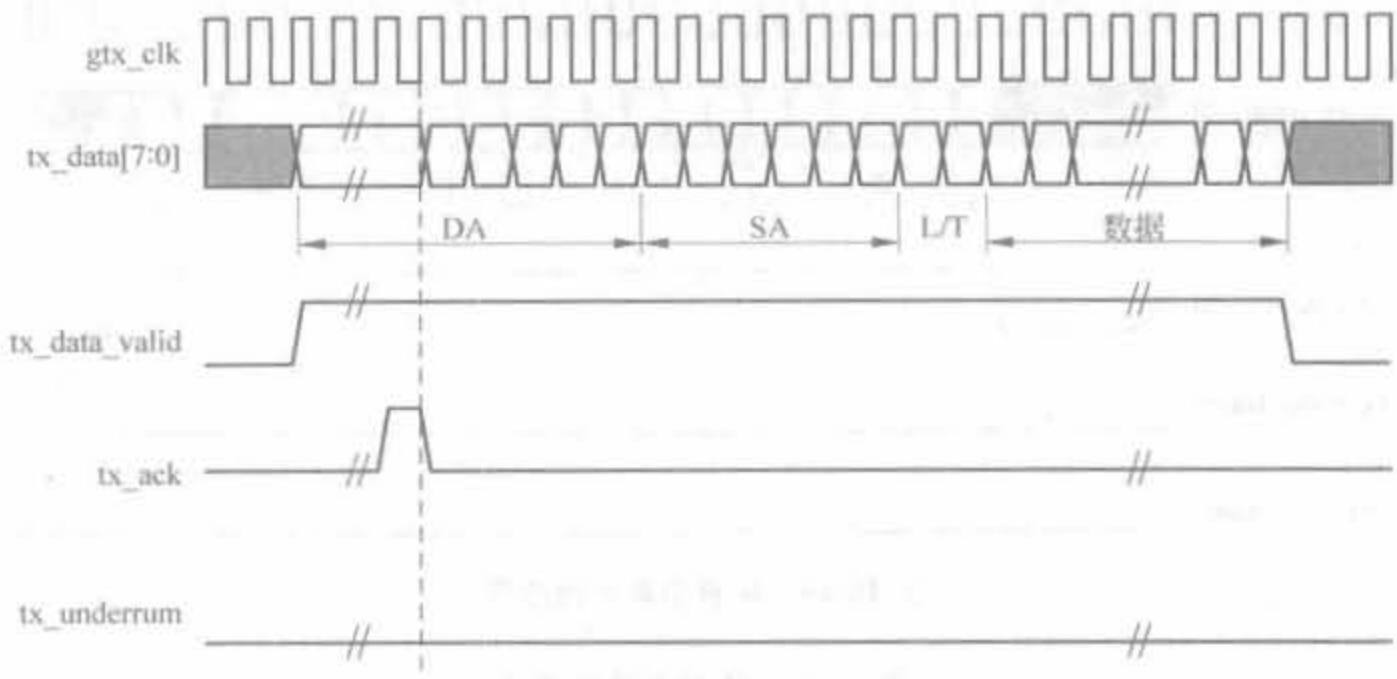


图 10-38　发送引擎的时序图

2）接收引擎

接收引擎接收来自 GMII 模块的数据，去掉帧头的引导区域，包括为了增加帧长的冗余比特。此外，该模块还能根据数据帧中的检验序列区域、接收到的 GMII 错误码字以及帧长信息完成错误检测，其主要接口信号的简要说明如表 10-11 所示。

表 10-11　接收引擎接口信号列表

信号名	方向	驱动时钟	简要描述
rx_data[7:0]	输出	gmii_rx_clk	8bit 接收数据端口
rx_data_valid	输出	gmii_rx_clk	接收数据端口的控制信号
rx_good_frame	输出	gmii_rx_clk	在数据帧发送完毕后拉高，表明 MAC 用户端逻辑应该接收该帧数据
rx_bad_frame	输出	gmii_rx_clk	在数据帧发送完毕后拉高，表明 MAC 用户端逻辑应该丢弃该帧数据
rx_statistics_vector[21:0]	输出	gmii_rx_clk	在数据接收完毕后，提供相应的数据统计信息
rx_statistics_valid	输出	gmii_rx_clk	输出数据统计信息的使能信号，高电平有效

接收引擎的时序如图 10-39 所示。客户端逻辑必须在任何时候都准备好接收数据，因为在 GEMAC 核中没有接收缓存，有延时就会丢失数据，因此用户可自己添加缓冲逻辑。rx_data_valid 为高电平时立即开始接收连续数据。在检测到 rx_data_valid 变低后，判断 rx_good_frame 信号的电平，如果为高，则继续处理该帧数据，否则直接丢弃。

3）流控制

流控制模块是根据 IEEE 802.3-2002 标准的 31 项条款设计的，在发送时附带暂停帧，接收时也需要对其处理。在 GEMAC 核中，它是自动配置的，同时提供了用户自定义的配置端口，如表 10-12 所示。

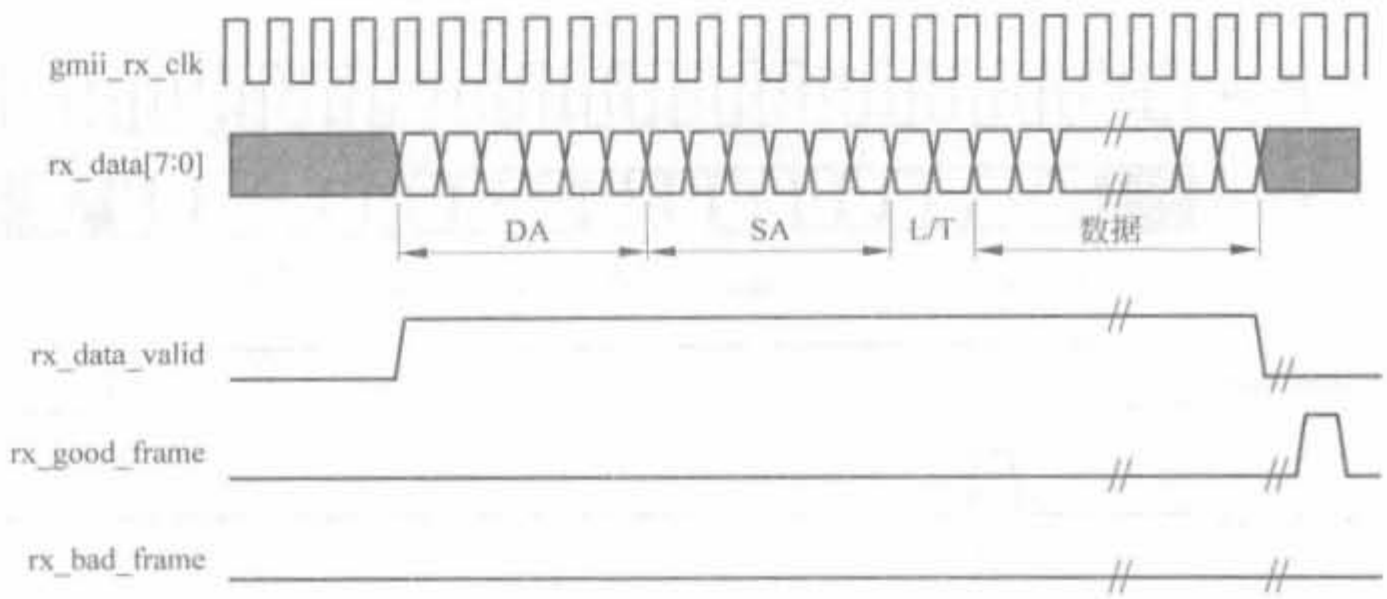


图 10-39　接收引擎的时序图

表 10-12　流控制接口信号

信号名	方向	驱动时钟	简要描述
pause_reg	输入	gtx_clk	暂停帧发送请求信号
pause_val[15:0]	输入	gtx_clk	暂停帧中引导其余的暂停数值

4）可选的管理接口

管理接口是一个独立的可选端口，其地址、数据和控制信号相对于其他模块是独立的。它提供了和 CoreConnec 总线的交互能力，可挂在 MicroBlaze 软核或 PowerPC 硬核处理器，可用于配置 GEMAC 核以及通过 MDIO 接口直接读写外部 PHY 芯片的内部配置寄存器。该模块包括用户管理的接口和 MDIO 接口两部分，后者直接与 PHY 芯片相连，全部接口信号的简要说明如表 10-13 所示。

表 10-13　管理模块的接口信号

信号名	方向	驱动时钟	简要描述
host_clk	输入	N/A	管理模块的时钟信号，必须大于 10MHz
host_opcode[1]	输入	host_clk	定义了 MDIO 接口的操作，且高比特还作为 PHY 内部配置寄存器的读写操作标志
host_addr[9:0]	输入	host_clk	访问寄存器的地址
host_wr_data[31:0]	输入	host_clk	写寄存器的数据
host_rd_data[31:0]	输出	host_clk	读寄存器的数据
host_miim_sel	输入	host_clk	MDIO 接口的控制信号，拉高后可访问 PHY 的内部寄存器
host_reg	输入	host_clk	MDIO 接口控制信号，通知发起一次传送
host_miim_rdy	输出	host_clk	MDIO 接口状态指示信息。高电平表明 MDIO 接口已经完成了一次传送，可接收新数据
mdc	输出	host_clk	MDIO 接口的时钟信号，根据寄存器 ClockDivide[4:0]的值，通过 host_clk 分频得到
mdio_in	输入	host_clk	MDIO 输入信号，其数值将被用于控制 PHY 芯片，不用时应当拉高
mdio_out	输出	host_clk	MDIO 输出信号，其数值来自于 PHY 芯片
mdio_tri	输出	host_clk	三态 MDIO 控制信号，低电平表明 mdio_out 的数值应该被连到 mdio 总线上

5）复位操作

管理模块具备自己独立的软件复位信号，但当管理接口模块被旁路掉时，其相应的配置信号被作为 configuration_vector[64:0]输入信号使用。此外，GEMAC 核提供了硬件复位信号，如表 10-14 所示。

表 10-14 复位信号列表

信号名	方向	驱动时钟	简要描述
reset	输入	无	GEMAC 核的全局异步复位信号，高电平有效

6）GMII 模块

GMII 模块的接收信号一般都是直接连到 PHY 芯片上，负责和 PHY 芯片的数据交互，其信号和 PHY 芯片的接口是一一对应的，如表 10-15 所示。

表 10-15 GMII 模块接口信号列表

信号名	方向	驱动时钟	简要描述
gmii_txd[7:0]	输出	gtx_clk	GMII 发送端口，将从发送引擎得到的数据发送到 PHY 芯片中
gmii_tx_en	输出	gtx_clk	GMII 发送控制信号
gmii_tx_er	输出	gtx_clk	GMII 发送控制信号
gmii_rx_clk	输入	N/A	从 PHY 芯片获取的 GMII 接收时钟信号，大小为 125MHz
gmii_rxd[7:0]	输入	gmii_tx_clk	GMII 接收端口，将从 PHY 得到的数据发送到接收引擎中
gmii_rx_dv	输入	gmii_tx_clk	GMII 接收控制信号
gmii_rx_er	输入	gmii_tx_clk	GMII 接收控制信号

配置完成后，可在工程管理区选中 GEMAC 核。在过程管理区单击“View HDL Instantiation Template”命令，查看其例化代码。它在代码中的例化方法和一般 IP Core 的方法是一样的。

10.5 本章小结

高速串行传输技术是 FPGA 未来的三大应用领域之一，本章主要介绍了 Xilinx 公司的 Rocket I/O 解决方案。首先给出高速传输的背景，指出串行方式是吉比特以及更高速率链路的必然选择。其次，给出了吉比特串行传输的通用架构，为后文做好铺垫。接着重点介绍了 Xilinx 公司 Rocket I/O 的系统组成、相关协议、时钟设计方案、开发要素以及 Rocket I/O 的使用方法。最后说明了千兆以太网 MAC 控制器 IP Core 的使用方法。读者需要注意的是，Rocket I/O 是 Xilinx 高端 FPGA 中的内嵌组件，和 DCM、硬核乘法器、块 RAM 等的使用方法是一样的，可通过 IP Core 调用。

高速传输是一种新技术，开发难度较大，本章只是介绍了其中的主要核心部分和基本原理，读者需要阅读大量的文献并进行实际操作，才能熟练开发相关系统。

第11章

时序分析原理以及时序分析器的使用

CHAPTER 11

工作频率对数字电路而言至关重要，因为高的工作频率意味着更加强大的处理能力，但也带来了时序瓶颈，主要表现在两个方面：时序冲突的概率变大以及电路的稳定性降低。为了使电路的性能达到设计者的预期目标，并满足电路工作环境的要求，必须对一个电路设计进行诸如时序、面积、负载等多方面的约束，并自始至终使用这些约束条件来驱动 EDA 软件的工作。ISE 具有一定的时序自动优化能力，对于一般的低速设计（处理时钟不超过 50MHz），基本上不需要时序方面的任何手动分析和处理。但对于高速和大规模设计，仅依赖 ISE 是不现实的，需要设计人员自行添加时序方面的控制和处理，通过反复操作，根据每次的反馈结果来逐步调整设定，直到满足要求为止。本章主要介绍时序分析、时序约束的原理，以及 ISE 中时序约束编辑器和时序分析工具的使用方法。

11.1 时序分析的作用和原理

时序分析贯穿于整个 FPGA 开发流程。在映射（Map）、布局（Place）和布线（Router）后都可以进行时序分析。任何阶段的时序分析不能满足，都需要重新修改源代码或者调整时序约束。本节简要介绍时序分析的作用和原理。

11.1.1 时序分析的作用

在以往的小规模 FPGA 设计中，验证环节通常只需要做动态的门级时序仿真，就可同时完成对 DUT（Device Under Test，被测试设计）的逻辑功能验证和时序验证。随着 FPGA 设计规模和速度的不断提高，要得到较高的测试覆盖率，必须编写大量的测试向量，这使得完成一次门级时序仿真的时间越来越长。为了提高验证效率，有必要将 DUT 的逻辑功能验证和时序验证分开，分别采用不同的验证手段加以验证[14]。

首先，电路逻辑功能的正确性可以由 RTL 或门级的功能仿真来保证；其次，电路时序是否满足，通过 STA（Static Timing Analysis，静态时序分析）得到。两种验证手段相辅相成，确保验证工作高效、可靠地完成。时序分析的主要作用就是查看 FPGA 内部逻辑和布线的延时，验证其是否满足设计者的约束。在工程实践中，主要体现在以下 3 点。

1. 确定芯片最高工作频率

更高的工作频率意味着更强的处理能力。通过时序分析，可以控制工程的综合、映射、

布局布线等关键环节，减少逻辑和布线延迟，从而尽可能提高工作频率。一般情况下，当处理时钟高于 100MHz 时，必须添加合理的时序约束文件，以通过相应的时序分析。

2. 检查时序约束是否满足

可以通过时序分析来查看目标模块是否满足约束。如果不能满足，可以通过时序分析器来定位程序中不满足约束的部分，并给出具体原因。然后，设计人员依此修改程序，直到满足时序约束为止。

3. 分析时钟质量

时钟是数字系统的动力系统，但存在抖动、偏移和占空比失真等 3 大类不可避免的缺陷。要验证其对目标模块的影响有多大，必须通过时序分析。当采用了全局时钟等优质资源后，如果仍然是时钟造成目标模块不满足约束，则需要降低所约束的时钟频率。

4. 确定分配管脚特性

FPGA 的可编程特性使电路板设计、加工和 FPGA 设计可以同时进行，而不必等 FPGA 管脚位置完全确定后再进行，从而节省了系统开发时间。通过时序分析，可以指定 I/O 管脚所支持的接口标准、接口速率和其他电气特性。

11.1.2　静态时序分析原理

早期的电路设计通常采用动态时序验证的方法来测试设计的正确性。但是随着 FPGA 工艺向着深亚微米技术的发展，动态时序验证所需要的输入向量将随着规模增大以指数增长，导致验证时间占据整个芯片开发周期的很大比重。此外，动态验证还会忽略测试向量没有覆盖的逻辑电路。因此静态时序分析(STA，Static Timing Analysis)应运而生，它不需要测试向量，即使没有仿真条件，也能快速地分析电路中的所有时序路径是否满足约束要求。STA 的目的就是要保证 DUT 中的所有路径满足内部时序单元对建立时间和保持时间的要求。信号可以及时地从任一时序路径的起点传递到终点，同时要求在电路正常工作所需的时间内保持恒定。整体上讲，静态时序分析具有不需要外部测试激励、效率高和全覆盖的优点，但其精确度不高。

STA 是通过穷举法抽取整个设计电路的所有时序路径，按照约束条件分析电路中是否有违反设计规则的问题，并计算出设计的最高频率。和动态时序分析不同，STA 仅着重于时序性能的分析，并不涉及逻辑功能。STA 是基于时序路径的，它将 DUT 分解为 4 种主要的时序路径，如图 11-1 所示。每条路径包含一个起点和一个终点，时序路径的起点只能是设计的基本输入端口或内部寄存器的时钟输入端，终点只能是内部寄存器的数据输入端或设计的基本输出端口。

STA 的 4 类基本时序电路为：

(1) 从输入端口到触发器的数据 D 端；

(2) 从触发器的时钟 CLK 端到触发器的数据 D 端；

(3) 从触发器的时钟 CLK 端到输出端口；

(4) 从输入端口到输出端口。

静态时序分析在分析过程中计算时序路径上数据信号的到达时间和要求时间的差值，

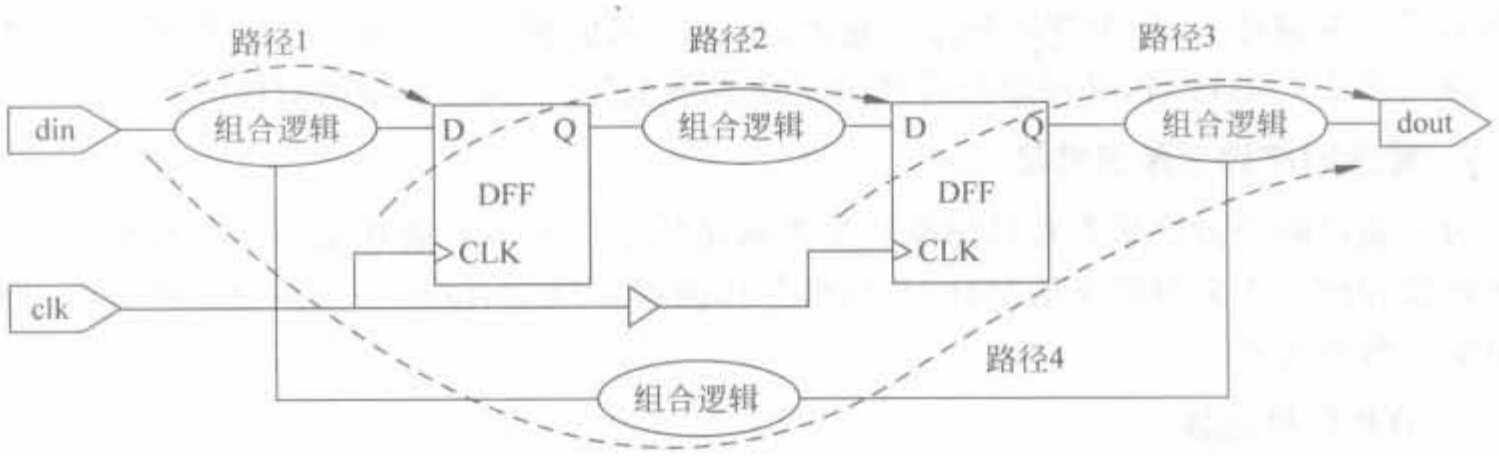


图 11-1　静态时序分析的基本路径

以判断是否存在违反设计规则的错误。数据的到达时间指的是：数据沿路从起点到终点经过的所有器件和连线延迟时间之和。要求时间是根据约束条件(包括工艺库和 STA 过程中设置的设计约束)计算出的从起点到达终点的理论时间，默认的参考值是一个时钟周期。如果数据能够在要求时间内到达终点，那么可以说这条路径是符合设计规则的。其计算公式如式(11-1)所示：

$$Slack = Trequired\_time - Tarrival\_time \tag{11-1}$$

其中，*Trequired_time* 为约束时长；*Tarrival_time* 为实际延时；*Slack* 为时序裕量标志，正值表示满足时序，负值表示不满足时序。*Tarrival_time* 的具体计算见 11.1.3 节。STA 把式(11-1)作为理论依据，分析设计电路中的所有时序路径。如果得到的 STA 报告中 *Slack* 为负值，那么此时序路径存在时序问题，是一条影响整个设计电路工作性能的关键路径。在逻辑综合、整体规划、时钟树插入、布局布线等阶段进行静态时序分析，就能及时发现并修改关键路径上存在的时序问题，达到修正错误、优化设计的目的。

11.1.3　时序分析的基础知识

本节主要介绍时序分析的基础知识，重点说明时钟抖动、时钟建立时间、保持时间以及基本时序路径等基本概念。

1. 时钟的时序特性

时钟的时序特性主要分为偏移(Skew)、抖动(Jitter)和占空比失真(Duty Cycle Distortion)这三点。对于低速设计，基本上不用考虑这些特征，但随着高速设计时代的到来，由于时钟本身所造成的时序问题的现象越来越普遍，因此有必要关注高速时钟本身的时序特性。

1) 时钟偏移

由于时钟信号要提供给整个电路的时序单元，从而导致时钟线非常长，并构成分布式 RC 网络。它的延时与时钟线的长度及被时钟线驱动的时序单元的负载电容、个数有关。由于时钟线长度及负载不同，会导致时钟信号到达相邻两个时序单元的时间不同，产生所谓的时钟偏移(Skew)。时钟偏移指的是同一个时钟信号到达两个不同的寄存器之间的时间差值，根据差值的正、负可以分为正偏移和负偏移。时钟偏移是永远存在的，当其大到一定程度时，就会严重影响电路的时序[15]。

两条时钟路径的长度不同是造成时钟抖动的原因，如图 11-2 所示为一条局部路径，R1、R2 为两个寄存器，C1 和 C2 来自同一个时钟源，时钟信号沿时钟树到达寄存器 R1 和 R2 的延迟时间分别为 TC1 和 TC2，用 Tskew 表示它们之间的时钟偏移，则 Tskew＝TC1－TC2。当 C1 比 C2 后到时，Tskew 为正；当 C1 比 C2 先到时，Tskew 为负。为了消除该类现象发生，在 FPGA 设计中，主要时钟信号应该走全局时钟网络，以避免时钟偏移现象。该网络采用全铜工艺和树状结构，并设计了专用时钟缓冲和驱动网络，到所有的 CLB、I/O 单元和块 RAM 的偏移非常小，可以忽略不计。

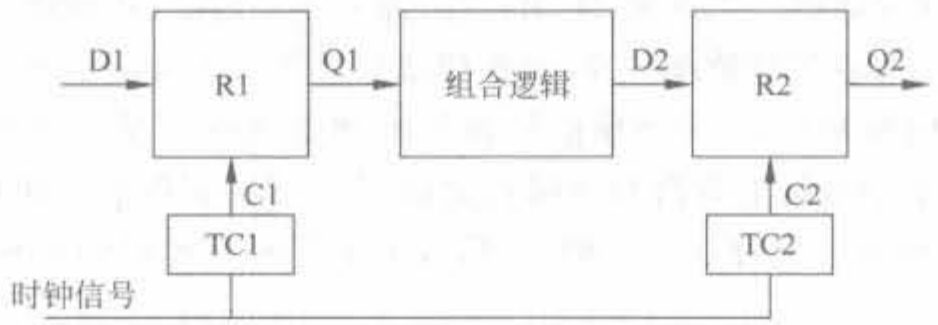


图 11-2　时钟抖动的示意图

在实际电路中，时钟信号到达每一个触发器时钟管脚的延时不可能完全相等，时钟偏差是肯定存在的，这是 STA 必须考虑的因素。在不同阶段，STA 设置时钟偏差的方式不同。静态时序分析可以分为布局布线前和布局布线后两个阶段，两者的主要区别在于：后者有具体的互连线长度、宽度、信号分布情况等信息，所以后者可以更加准确地估计线上延迟，以及时钟树的延迟；而前者只能根据设计电路面积的大小，来粗略估计线上延迟和时钟树的延迟。

2）时钟抖动

抖动是时钟的一个重要参数，如图 11-3 所示。对于抖动已有多种定义，两个最常用的抖动参数称为周期抖动和周期间抖动。周期抖动一般比较大也比较确定，常由于第三方原因造成，如干扰、电源、噪声等。周期间抖动由环境因素造成，具有不确定性，满足高斯分布，一般难以跟踪。

图 11-3　时钟抖动的示意图

时钟抖动是永远存在的，当其大到可以和时钟周期相比拟时，必然会影响到设计时序，这样的时钟抖动就是不可接受的。

3）时钟占空比失真

时钟占空比失真(Duty Cycle Distortion，DCD)即时钟不对称性，指信号在传输过程中由于变形、延时等原因，脉冲宽度所发生的变化，该变化使有脉冲和无脉冲持续时间的比例发生改变，如图 11-4 所示。

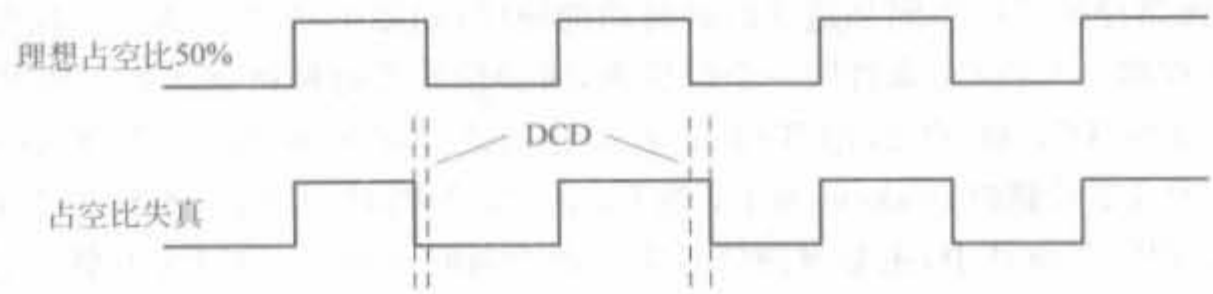


图 11-4　时钟占空比失真示意图

现在的片外高速存储器(如 DDR、DDR Ⅱ 等)都采用双数据,甚至多数据速率接口,在时钟的上升沿和下降沿都需要对数据采样,每次读或写操作至少有两拍时钟。在此类应用时,DCD 会吞噬大量的时序裕量,造成数字信号的失真,使过零区间偏离理想位置,向上或向下移动。DCD 通常是由信号的上升沿和下降沿之间时序不同而造成。如果非平衡系统中存在地电位漂移,差分输入之间存在电压偏移,信号的上升和下降时间出现变化等,也可能造成这种失真。

4) 时钟建立、保持时间

建立时间(Setup Time)是指在触发器的时钟信号上升沿到来以前,数据稳定不变的时间,常用 t_{SU} 表示。如果建立时间不够,数据将不能在这个时钟上升沿被打入触发器。保持时间(Hold Time)是指在触发器的时钟信号上升沿到来以后,数据稳定不变的时间,常用 t_H 表示。如果保持时间不够,数据同样不能被打入触发器。建立时间和保持时间的示意图如图 11-5 所示。要使数据稳定传输,必须满足建立时间和保持时间的要求。

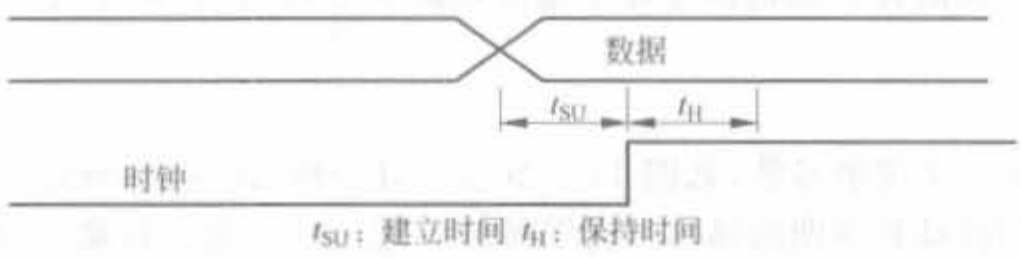


图 11-5　时钟建立、保持时间示意图

在如图 11-5 所示的电路中,时钟的 t_{SU}、t_H 的计算公式如式(11-2)和式(11-3)所示。

$$t_{SU} = \text{Data\_Delay} - \text{Clock\_Delay} + \text{Micro\_}t_{SU} \tag{11-2}$$

$$t_H = \text{Clock\_Delay} - \text{Data\_Delay} + \text{Micro\_}t_H \tag{11-3}$$

其中的 Micro_t_{SU} 和 Micro_t_H 指的是触发器内部的固有建立时间和保持时间,是触发器的固有属性,其典型值一般小于 1ns。

当数据在建立时间或保持时间内发生变化,即数据建立时间和保持时间不满足时,就可能发生亚稳态现象。亚稳态的信号不稳定,其数值在 0 和 1 之间震荡,尽管无法确定信号为 0 或 1,但其一定是 0 或 1。亚稳态恢复时间的长短在信号发生跳变时,随时钟相位的不同而不同,最坏情况下需要 100～1000 个时钟周期。亚稳态对设计可靠性的影响很大,可能导致数据采样错误,在严重的时候甚至会导致系统崩溃,因此要充分考虑亚稳态所带来的影响。在单时钟设计中,只要满足最高频率要求,就不会出现亚稳态;在多时钟设计中,由于时钟异步,亚稳态不可避免,但可采用一些方法使电路在亚稳态仍正常工作,如通过块 RAM 交换时钟,使数据同步。

2. FPGA 设计的基本时序路径

在高速的同步电路设计中，时序决定了一切，要求所有时序路径延迟都必须在约束限制的时钟周期内，这成为设计人员最大的难题。因此，首先确定和分析基本时序路径有助于设计者快速、准确地计算时序裕量，使系统稳定工作。下面介绍 Xilinx 公司提倡的几种常用基本路径。

1) Clock-to-Setup 路径

Clock-to-Setup 路径从触发器的输出端开始，结束于下一级触发器、锁存器或者 RAM 的输入端，对终止端的数据信号要求一定的建立时间，如图 11-6 所示。

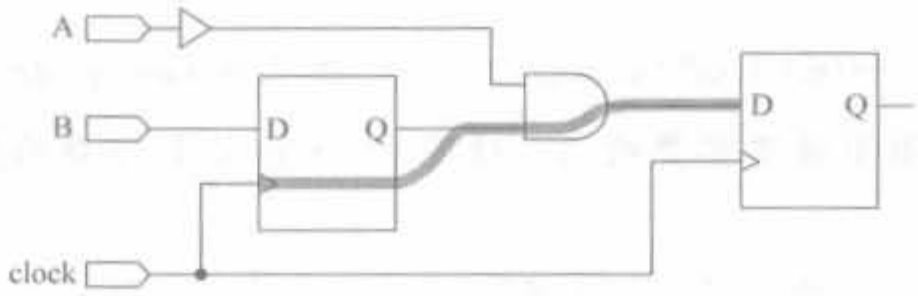
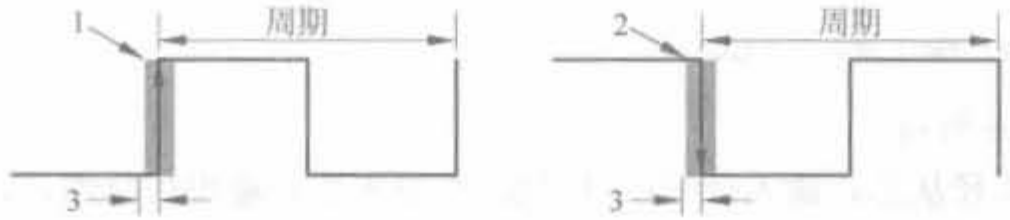


图 11-6　Clock-to-Setup 路径示意图

该条路径包括了触发器内部 Clock-to-Q 的延迟，触发器之间由组合造成的路径延迟以及目标触发器的建立时间，其延时是数据从源触发器开始，在下一个时钟沿到来之前通过中间组合逻辑和布线的最大时间。Clock-to-Setup 时间可通过约束文件中的周期约束来限制，如图 11-7 所示。



1. 用于OFFSET值的初始边缘为高电平
2. 用于OFFSET值的初始边缘为低电平
3. 输入抖动

图 11-7　约束参数示意图

需要注意的两点是：①当源触发器和目标触发器的驱动时钟不同，且时钟的占空比不是 50%时，Clock-to-Setup 路径被限制为两个时钟高、低电平中最短的那一段。②如果源、目的触发器使用不同的时钟网络驱动，则要求目标触发器的时钟周期必须大于 Clock-to-Setup 路径延时，且将其作为所允许的最大延时值。

2) Clock-to-Pad 路径

Clock-to-Pad 路径从寄存器或者锁存器的时钟输入端开始，终止于芯片的输出管脚，中间经过了触发器输出端以及所有的组合逻辑，如图 11-8 所示。

该条路径包括了经过触发器的延时和从触发器到输出管脚之间的逻辑延迟。在约束文件中，可以通过 OFFSET 语句和 FROM：TO 来约束。如果使用 OFFSET 语句，那么延时计算时会包含时钟输入 BUFFER/ROUTING 延时；如果使用 FROM：TO 约束，则延时从触发器自身开始，不包括输入路径，比较精确，所以使用得相对更频繁一些。

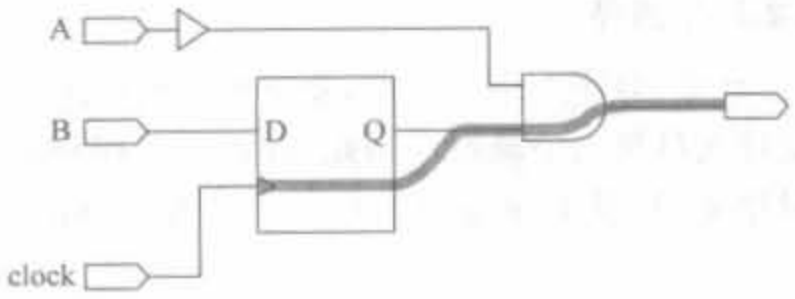


图 11-8　Clock-to-Pad 路径示意图

OFFSET 说明了外部时钟和与其相关的输入、输出数据管脚之间的时序关系，其语法规则为：

```
OFFSET = {IN|OUT} "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP "group_name"];
```

OFFSET 可以用于设置多类约束。对于 Clock-to-Pad，需要将属性配置为 OUT AFTER，例如：

```
NET Q_OUT OFFSET = OUT 35.0 AFTER "CLK_SYS";
```

FROM：TO 定义了两组信号之间时序关系，其语法规则为：

```
# TIMESPEC "TSname" = FROM "group1" TO "group2" value;
```

其中的 TSname 必须以 TS 开头，group1 是起始路径，group2 是目的路径。value 值的默认单位为 ns，也可以使用 MHz。

下面给出一个应用实例：

```
TIMESPEC TS_aa = FROM FFS TO PADS 10;
```

3）Pad-to-Pad 路径

Pad-to-Pad 路径从芯片输入信号端口开始，结束于芯片输出信号端口，中间包含所有的组合逻辑，但并不包含任何同步逻辑，如图 11-9 所示。

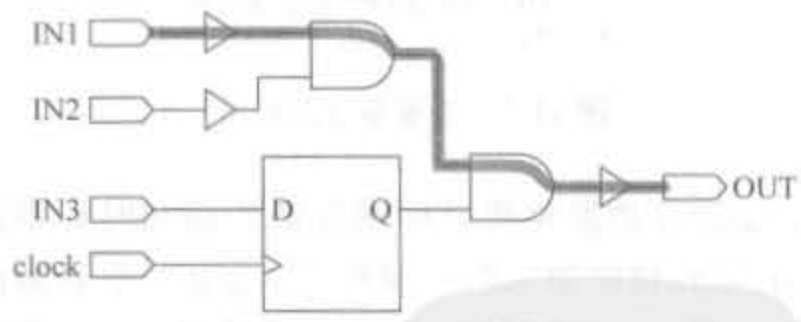


图 11-9　Pad-to-Pad 路径示意图

Pad-to-Pad 路径延时是数据输入到芯片，经过逻辑延时和布线时延后再输出芯片的最大时间要求。在约束文件中仍然通过 FROM：TO 来约束，其语法为：

```
TIMESPEC TS_aa = FROM PADS TO PADS 10;
```

4）Pad-to-Setup 路径

Pad-to-Setup 路径从芯片的输入信号端口开始，结束于同步电路模块（触发器、锁存器和 RAM），对相应的数据信号要求一定的建立时间，如图 11-10 所示。

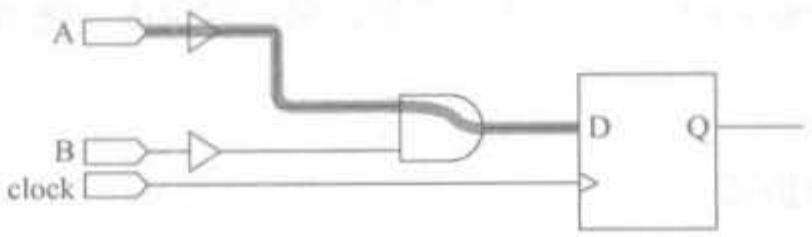


图 11-10　Pad-to-Setup 路径示意图

该路径可以经过输入 BUFFER 和所有的组合逻辑，不包含任何同步电路模块和双向端口，是数据到达芯片的最大时间要求。和 Clock-to-Pad 一样，该路径可以通过 OFFSET 和 FROM：TO 来设计。其中，OFFSET 语句的属性设置为 OFFSET IN BEFORE。例如：

```
OFFSET = IN 10ns BEFORE my_clk TIMEGRP My_FFS;
```

3. 时序裕量

时序裕量(Slack)是约束文件要求时钟周期与实际布局布线后时钟周期的差值。当其为正值时，表示满足时序(正裕量)；为当其为负值时，表明不满足时序(负裕量)。其计算公式为：

$$\text{Slack} = \text{RClk}_{\text{period}} - \text{AClk}_{\text{period}} \tag{11-4}$$

其中，$\text{RClk}_{\text{period}}$为约束要求的时钟周期，$\text{AClk}_{\text{period}}$为实际的时钟周期。

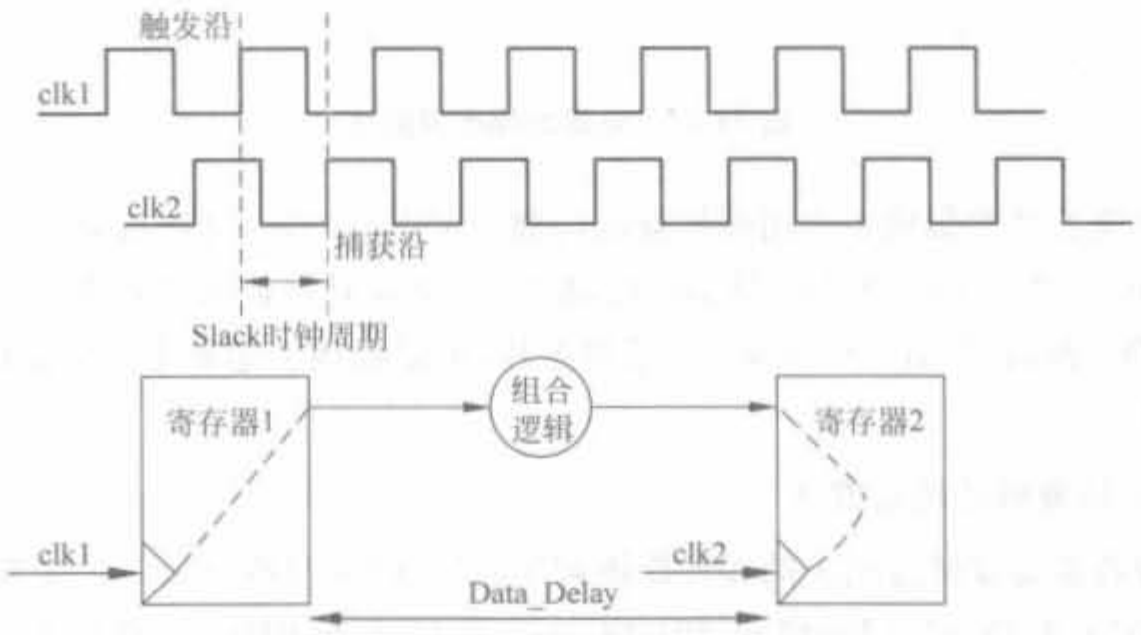


图 11-11　Slack 计算的示例场景

下面以如图 11-11 所示简易场景为例，给出 Slack 的计算公式如式(11-5)所示[16]。

$$\text{Slack} = \text{Slack}_{\text{period}} - (\text{Micro\_}t_{\text{CO}} + T_{\text{delay}} + \text{Micro\_}t_{\text{SU}}) \tag{11-5}$$

其中，$\text{Micro_}t_{\text{CO}}$和 $\text{Micro_}t_{\text{SU}}$都是寄存器的固有特性，由芯片的制造工艺和速度等级决定。$\text{Micro_}t_{\text{SU}}$在前文已有介绍；$\text{Micro_}t_{\text{CO}}$指寄存器在时钟有效沿将数据送到输出端口的内部延时参数，一般小于 1ns。

11.2　Xilinx FPGA 中的时钟资源

全局时钟和第二全局时钟资源是 FPGA 芯片的重要资源之一，合理地利用该资源可以改善设计的综合和实现效果；如果使用不当，不但会影响设计的工作频率和稳定性，甚至会

导致设计的综合、实现过程出错。本节主要介绍两类时钟资源的使用方法，并强调了应用的注意事项。

11.2.1 全局时钟资源

1. 全局时钟资源简介

Xilinx FPGA 中的全局时钟采用全铜工艺实现，并设计了专用时钟缓冲与驱动结构，可到达芯片内部任何一个逻辑单元，包括可配置逻辑单元(CLB)、I/O 管脚、内嵌的块 RAM 以及硬核乘法器等模块，且延时和抖动都最小。因此，对于 FPGA 设计而言，全局时钟是最简单和最可预测的时钟。最好的时钟方案就是：由专用的全局时钟输入管脚驱动单个全局时钟，并用后者去控制设计中的每一个触发器，如图 11-12 所示。

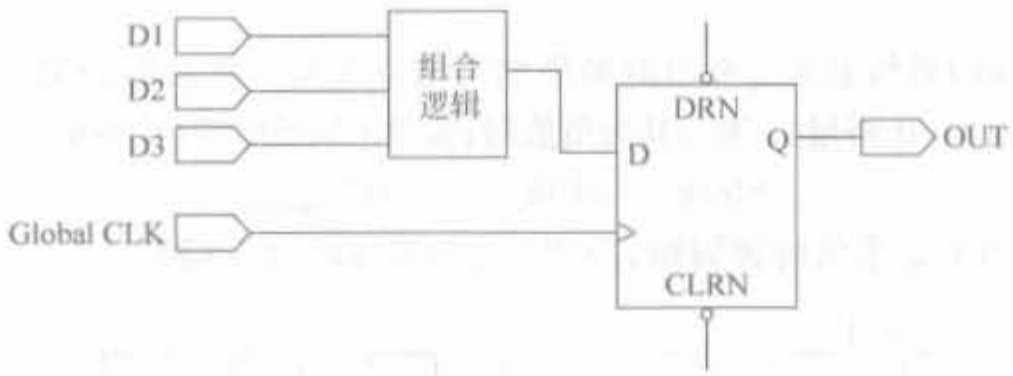


图 11-12 全局时钟应用策略

目前的主流芯片都集成了专用时钟资源与数字延迟锁相环，且数目众多。如面向中、低端应用的 Spartan-3E 系列 FPGA，最多可提供 16 个全局时钟输入端口和 8 个数字时钟管理模块(DCM)；面向高端的 Virtex-4/5 系列芯片，可以提供多达数十个全局时钟输入端口和 DCM 模块。

2. 全局时钟资源的使用方法

全局时钟资源需要通过相关的器件原语调用，常用的时钟原语见 3.4.2 节，包括全局时钟缓冲(IBUFG)、差分全局时钟缓冲(IBUFGDS)、全局缓冲(BUFG)、带时钟使能信号的全局缓冲(BUFGP)、全局时钟缓冲复用(BUFGMUX)、全局时钟延迟锁相环(BUFGDLL)和数字时钟管理单元(DCM)等。下面对其功能进行简单说明。

IBUFG 即输入全局缓冲，是与专用全局时钟输入管脚相连接的首级全局缓冲。所有从全局时钟管脚输入的信号必须经过 IBUF 元，否则在布局布线时会报错。IBUFGDS 是 IBUFG 的差分形式，当信号从一对差分全局时钟管脚输入时，必须使用 IBUFGDS 作为全局时钟输入缓冲。BUFG 是全局缓冲，它的输入是 IBUFG 的输出。BUFG 的输出到达 FPGA 内部的 IOB、CLB、选择性块 RAM 的时钟延迟和抖动最小。BUFGCE 是带有时钟使能端的全局缓冲。

BUFGMUX 是全局时钟选择缓冲，它有 I0 和 I1 两个输入，一个控制端 S，一个输出端 O。当 S 为低电平时，输出时钟为 I0，反之为 I1。需要指出的是，BUFGMUX 的应用十分灵活，I0 和 I1 两个输入时钟甚至可以为异步关系。

BUFGDLL是全局缓冲延迟锁相环，相当于BUFG与DLL的结合。BUFGDLL在早期设计中经常使用，用以完成全局时钟的同步和驱动等功能。随着数字时钟管理单元(DCM)的日益完善，目前BUFGDLL的应用已经逐渐被DCM所取代。

DCM模块的原理和使用已在本书中多个地方出现，这里就不再介绍。

不同原语的组合构成了不同的全局时钟使用方法，主要有下面4种组合。

1) IBUFG/IBUFGDS+BUFG的方法

IBUFG后面连接BUFG的方法是最基本的全局时钟资源使用方法，由于IBUFG组合BUFG相当于BUFGP，所以这种使用方法也称为BUFGP方法。其相应的语法为：

```
IBUFG CLKIN_IBUFG_INST(.I(CLKIN_IN),
                       .O(CLKIN_IBUFG));

//调用 BUFG 原语，将 CLKIN_IBUFGDS 转换成最终输出
BUFG CLK_BUFG_INST (.I(CLKIN_IBUFGDS),
                    .O(CLK_OUT));
//示例结束
```

当输入时钟信号为差分信号时，需要使用IBUFGDS代替IBUFG，相应的语法为：

```
//调用 IBUFGDS 原语，将差分时钟转换成单端输出 CLKIN_IBUFGDS
IBUFGDS CLKIN_IBUFGDS_INST (.I(CLKIN_P_IN),
                            .IB(CLKIN_N_IN),
                            .O(CLKIN_IBUFGDS));

//调用 BUFG 原语，将 CLKIN_IBUFGDS 转换成最终输出
BUFG CLK_BUFG_INST (.I(CLKIN_IBUFGDS),
                    .O(CLK_OUT));
//示例结束
```

需要注意的是，当信号从全局时钟管脚输入时，不论其是否为时钟信号，都必须使用IBUFG或IBUFGDS；反之，如果对信号使用了IBUFG或IBUFGDS硬件原语，则该信号一定要从全局时钟管脚输入，否则在布局布线时会报错。IBUFG和IBUFGDS的输入端仅仅与芯片的专用全局时钟输入管脚有物理连接，与普通I/O和其他内部CLB等没有物理连接。

2) LOGIC+BUFG的方法

BUFG不但可以驱动IBUFG的输出，还可以驱动其他普通信号的输出。当某个信号(时钟、使能、快速路径)的扇出非常大，并且要求抖动延迟最小时，可以使用BUFG驱动该信号，使该信号利用全局时钟资源。但需要注意的是，普通I/O的输入或普通片内信号进入全局时钟布线层需要一个固有的延时，一般在10ns左右，即普通I/O和普通片内信号从输入到BUFG输出有一个约10ns左右的固有延时，但是BUFG的输出到片内所有单元(IOB、CLB、选择性块RAM)的延时可以忽略不计，为“0”ns[9]。

相应的调用语法为：

```
//调用 BUFG 原语，将内部输入 CLK_IN 转换成最终输出 CLK_OUT
BUFG CLK_BUFG_INST (.I(CLK_IN),
                    .O(CLK_OUT));
//示例结束
```

3) IBUFG/IBUFGDS+DCM+BUFG 的方法

这种使用方法最灵活，对全局时钟的控制更加有效。通过 DCM 模块，不仅能对时钟进行同步、移相、分频和倍频等变换，而且可以使全局时钟的输出达到无抖动延迟。IBUFG 适用于外部输入单端时钟，如果在调用 DCM 模块时选择外部输入单端时钟，就采用这种应用模块，其相应的例化模板，也就是使用方法，如下所示：

```
// 例化 IBUFG 原语，输入 CLKIN_IN 信号，产生 CLKIN_IBUFG 信号送往 DCM 模块
IBUFG CLKIN_IBUFG_INST (.I(CLKIN_IN),
                        .O(CLKIN_IBUFG));

// 例化 DCM 模块，产生时钟信号 CLK CLKFX_BUF、CLK0_BUF
DCM_ADV DCM_ADV_INST (.CLKFB(CLKFB_IN),
                      .CLKIN(CLKIN_IBUFG),
                      .DADDR(GND1[6:0]),
                      .DCLK(GND3),
                      .DEN(GND3),
                      .DI(GND2[15:0]),
                      .DWE(GND3),
                      .PSCLK(GND3),
                      .PSEN(GND3),
                      .PSINCDEC(GND3),
                      .RST(RST_IN),
                      .CLKFX(CLKFX_BUF),
                      .CLK0(CLK0_BUF),
                      .LOCKED(LOCKED_OUT)
);

// 例化 BUFG 原语，将 DCM 模块送出的 CLKFX_BUF 经过 BUFG 模块，
// 产生最终输出信号 CLKFX_OUT
BUFG CLKFX_BUFG_INST (.I(CLKFX_BUF),
                      .O(CLKFX_OUT));
// 示例模板结束
```

当外部输入为差分时钟时，需要选用 IBUFGDS 原语。在实现时，在 DCM 模块 IP Core 的配置界面选择外部输入差分时钟即可。

4) LOGIC+DCM+BUFG 的方法

在介绍 DCM 模块时，指出其输入时钟可以为内部还是外部的。如果是外部输入，需要保证输出时钟信号由芯片管脚引入；如果选择内部输入，可以选择内部逻辑中的任意信号。对应于本应用模式，其相应的例化模板如下所示。在 FPGA 芯片内部，是没有差分信号的，所有内部时钟信号都是单端信号。

```
// 例化 DCM 模块，直接由内部输入 CLKIN_IN 产生时钟信号 CLK CLKFX_BUF
DCM_ADV DCM_ADV_INST (.CLKFB(CLKFB_IN),
                      .CLKIN(CLKIN_IN),
                      .DADDR(GND1[6:0]),
                      .DCLK(GND3),
                      .DEN(GND3),
                      .DI(GND2[15:0]),
```

```
                .DWE(GND3),
                .PSCLK(GND3),
                .PSEN(GND3),
                .PSINCDEC(GND3),
                .RST(RST_IN),
                .CLKFX(CLKFX_BUF),
                .CLK0(CLK0_BUF),
                .LOCKED(LOCKED_OUT)
);

// 例化 BUFG 原语,将 DCM 模块送出的 CLKFX_BUF 经过 BUFG 模块,
// 产生最终输出信号 CLKFX_OUT
BUFG CLKFX_BUFG_INST (.I(CLKFX_BUF),
                      .O(CLKFX_OUT));
// 示例模板结束
```

全局时钟资源是专用布线资源,存在于全铜布线层上,用户使用全局时钟资源并不会影响芯片的其他布线资源,因此在可能使用全局时钟资源的场合尽可能使用该资源。

11.2.2　第二全局时钟资源

1. 第二全局时钟资源简介

第二全局时钟属于长线资源,分布于芯片内部的行、列栅栏(Bank)上,其长度和驱动能力仅次于全局时钟资源,也可驱动芯片内部任何一个逻辑资源,其抖动和延时指标仅次于全局时钟信号。在设计中,一般将高频率、高扇出的时钟使能信号以及高速路径上的关键信号指定为全局第二时钟信号。

需要注意的是,第二全局时钟资源和全局时钟资源的区别在于,使用全局时钟资源并不占用逻辑资源,也不会影响其他布线资源;而第二全局时钟资源占用的是芯片内部的资源,需要占用部分逻辑资源,各个部分的布线会相互影响,因此建议在逻辑设计占用资源不超过芯片资源 70%时使用。

2. 第二全局时钟资源的使用方法

第二全局时钟信号的驱动能力和时钟抖动延迟等指标仅次于全局时钟信号。Xilinx 的 FPGA 中一般有比较丰富的第二全局时钟资源(很多器件有 24 个第二全局时钟资源),以满足高速、复杂时序逻辑设计的需要。在设计中,一般将频率较高,扇出数目较多的时钟、使能、高速路径信号指定为第二全局时钟信号[5]。

第二全局时钟资源的使用方法比较简单,可通过在约束编辑器的专用约束(Misc)选项卡中指定所选信号使用低抖动延迟资源“Low Skew”来制定。但最直接的方法是在指导 Xilinx 实现步骤的用户约束文件(UCF)中添加“USELOWSKEWLINES”约束命令。在约束编辑器中的操作等效于在用户约束文件中添加如下内容:

```
NET "s1" USELOWSKEWLINES;
NET "s2" USELOWSKEWLINES;
NET "s3" USELOWSKEWLINES;
```

11.3 时序约束

时序约束主要包括周期约束、偏移约束和静态路径约束等3种。时序约束文件可通知布局布线器调整映射和布局布线过程，使设计尽量达到时序要求。一般的约束策略是先添加全局约束，再根据链路速率添加专门的局部约束。时序约束只是通知实现工具在映射和布局布线时做出优化调整，具备一定的调整功能，但更多的是验证功能，检查电路是否满足实际需求。因此读者需要明白的是：时序是设计出来的，而不是通过约束得到的；时序分析本质上只是一种检查手段，辅助用户快速、优质地完成设计。掌握时序约束和分析的方法是每一个中、高级FPGA开发人员所必须掌握的。

11.3.1 使用约束文件添加时序约束

一般来讲，添加约束的原则为先附加全局约束，再补充局部约束，而且局部约束比较宽松。其目的是在可能的地方尽量放松约束，提高布线成功概率，减少ISE布局布线时间。典型的全局约束包括周期约束和偏移约束。

在添加全局时序约束时，需要根据时钟频率划分不同的时钟域，添加各自的周期约束；然后对输入/输出端口信号添加偏移约束，对片内逻辑添加附加约束。

1. 周期约束

周期约束是附加在时钟网络上的基本时序约束，以保证时钟区域内所有同步组件的时序满足要求。在分析时序时，周期约束能自动处理寄存器时钟端的反相问题。如果相邻的同步元件时钟相位相反，则其延迟会被自动限制为周期约束值的一半，这其实相当于降低了时钟周期约束的数值，所以在实际中一般不要同时使用时钟信号的上升沿和下降沿。

硬件设计电路所能工作的最高频率取决于芯片内部元件本身固有的建立保持时间，以及同步元件之间的逻辑和布线延迟。所以电路最高频率由代码和芯片两部分共同决定，相同的程序，在速度等级高的芯片上能达到更高的最高工作频率；同样，在同一芯片内，经过速度优化的代码具有更高的工作频率，在实际中往往取二者的平衡。

在添加时钟周期之前，需要对电路的期望时钟周期有一个合理的估计，这样才不会附加过松或过紧的周期约束。过松的约束不能达到性能要求；过紧的约束会增加布局布线的难度，实现的结果也不一定理想。常用的工程策略是：附加的时钟周期约束的时长为期望值的90%，即约束的最高频率是实际工作频率的110%左右。

附加时钟周期约束的方法有两个：一是简易方法，二是推荐方法。简易方式是直接将周期约束附加到寄存器时钟网线上，其语法如下所示：

[约束信号] PERIOD=〈周期长度〉〈HIGH|LOW〉[脉冲持续时间]；

其中，[]内的内容为可选项，〈〉中的内容为必选项，“|”表示选择项。[约束信号]可为“Net net_name”或“TIMEGRP group_name”，前者表示周期约束作用到线网所驱动的同步元件上，后者表示约束到TIMEGRP所定义的信号分组上（如触发器、锁存器以及RAM等）。

〈周期长度〉为要求的时钟周期，可选用 ms、s、ns 以及 ps 等单位，默认值为 ns，对单位不区分大小写。〈HIGH|LOW〉用于指定周期内第一个脉冲是高电平还是低电平。[脉冲持续时间]用于指定第一个脉冲的持续时间，可选用 ms、s、ns 以及 ps 等单位，默认值为 ns，如果缺省该项，则默认为 50%的占空比。如语句：

Net "clk_100MHz" period=10ns High 5ns;

指定了信号 clk_100MHz 的周期为 10ns，高电平持续的时间为 5ns，该约束将被添加到信号 clk_100MHz 所驱动的元件上。

推荐方法常用于约束具有复杂派生关系的时钟网络，其基本语法为：

```
TIMESPEC "TS_identifier" = PERIOD "TNM_reference" 〈周期长度〉
                          〈HIGH|LOW〉[脉冲持续时间];
```

其中，TIMESPEC 是一个基本时序相关约束，用于标志时序规范。“TS_identifier”由关键字 TS 和用户定义的 identifier 表示，二者共同构成一种时序规范，称为 TS 属性定义，可在约束文件中任意引用，大大地丰富了派生时钟的定义。在使用时，首先要定义时钟分组，然后添加相应的约束，如

```
NET"clk_50MHz" = "syn_clk";
TIMESPEC "TS_syn_clk" = PERIOD "syn_clk" 20 HIGH 10;
```

TIMESPEC 利用识别符定义派生时钟的语法为：

```
TIMESPEC "TS_identifier2" = PERIOD "timegroup_name" "TS_identifier1"
                          [*|/] 倍数因子 [+|-] phasevalue [单位]
```

其中，TS_identifier2 是要派生定义的时钟，TS_identifier1 为已定义的时钟，“倍数因子”用于给出二者周期的倍数关系，phasevalue 给出二者之间的相位关系。如定义系统时钟 clk_syn：

```
TIMESPEC "clk_syn" = PERIOD "clk" 5ns;
```

下面给出其反相时钟 clk_syn_180 以及 2 分频时钟 clk_syn_half：

```
TIMESPEC "clk_syn_180" = PERIOD "clk_180" clk_syn PHASE + 2.5ns;
    TIMESPEC "clk_syn_180" = PERIOD "clk_half" clk_syn/2;
```

2. 偏移约束

偏移约束也是一类基本时序约束，规定了外部时钟和数据输入/输出管脚之间的相对时序关系，只能用于端口信号，不能应用于内部信号，包括 OFFSET_IN_BEFORE、OFFSET_IN_AFTER、OFFSET_OUT_BEFORE 和 OFFSET_OUT_ AFTER 等 4 类基本约束。偏移约束的基本语法为：

```
OFFSET = [IN|OUT] "offset_time" [units] 〈BEFORE|AFTER〉 "clk_name"
         [TIMEGRP "group_name"];
```

其中，[IN|OUT]说明约束的是输入还是输出。“offset_time”为数据和有效时钟沿之间的时间差，〈BEFORE|AFTER〉表明该时间差是在有效时钟之前还是之后，“clk_name”为有效时钟的名字。[TIMEGRP "group_name"]是用户添加的分组信号，在缺省时，默认为时钟 clk_name 所驱动的所有触发器。偏移约束通知布局布线器输入数据的到达时刻，从而可准

确调整布局布线的过程，使约束信号建立时间满足要求。

1）“IN”偏移约束

“IN”偏移约束是输入偏移约束，有 OFFSET_IN_AFTER 和 OFFSET_IN_BEFORE 两种，前者定义了输入数据在有效时钟到达多长时间后可以到达芯片的输入管脚，这样可以得到芯片内部的延迟上限，从而对那些与输入管脚相连的组合逻辑进行约束；后者定义数据比相应的有效时钟沿提前多少时间到来，是与其相连的组合逻辑的最大延时，否则在时钟沿到来时，数据不稳定，会发生采样错误。输入偏移的时序关系如图 11-13 所示。

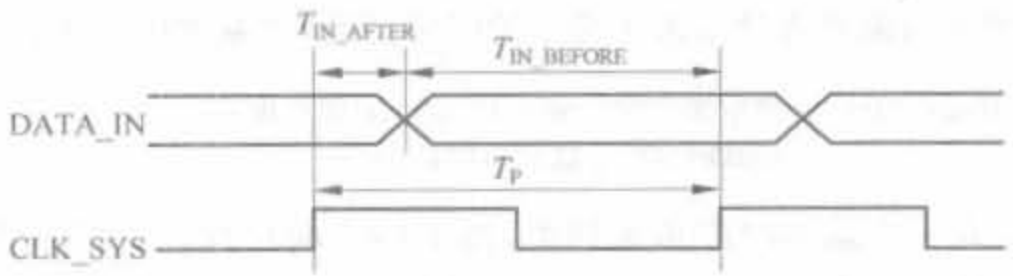


图 11-13　输入偏移的时序关系

例如，

```
NET "DATA_IN" OFFSET = IN 10.0 BEFORE "CLK_50MHz";
```

表明在时钟信号 CLK_50MHz 上升沿到达前的 10ns 内，输入信号 DATA_IN 必须到达数据输入管脚。

```
NET "DATA_IN" OFFSET = IN 10.0 AFTER "CLK_50MHz";
```

表明在时钟信号 CLK_50MHz 上升沿到达后的 10ns 内，输入信号 DATA_IN 必须到达数据输入管脚。

2）“OUT”偏移约束

“OUT”偏移约束是输出偏移约束，有 OFFSET_OUT_AFTER 和 OFFSET_OUT_BEFORE 两种。前者定义了输出数据在有效时钟沿之后多长时间稳定下来，是芯片内部输出延时的上限；后者定义了在下一个时钟信号到来之前多长时间必须输出数据，是下一级逻辑建立时间的上限。输出偏移的时序关系如图 11-14 所示。

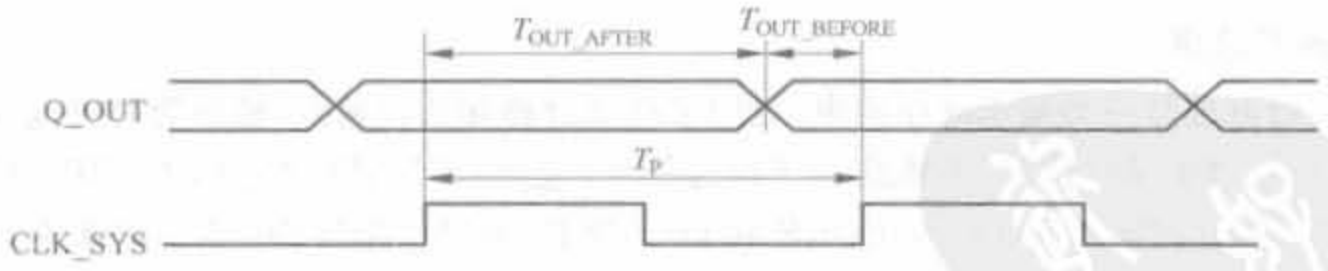


图 11-14　输出偏移的时序关系

例如，

```
NET "DATA_OUT" OFFSET = OUT 10.0 BEFORE "CLK_50MHz";
```

表明在时钟信号 CLK_50MHz 上升沿到达前的 10ns 内，输出信号 DATA_OUT 信号必须离开数据输出管脚。

```
NET "DATA_OUT" OFFSET = OUT 10.0 AFTER "CLK_50MHz";
```

表明在时钟信号 CLK_50MHz 上升沿到达后的 10ns 内，输出信号 DATA_OUT 信号必须一直保持在数据输出管脚上。

3. 分组约束

分组约束可有效管理大量的触发器、寄存器和存储器单元，将其分为不同的组，每组附加各自的约束，在大型设计中有着广泛的应用。

1) TNM/TNM_NET 约束

TNM/TNM_NET 约束用于选出可构成一个分组的元件，并对其重新命名，然后整体添加约束。除了 IBUFG 和 BUFG 外，所有的 FPGA 内部元件都可以用 TNM 来命名，其语法规则为：

```
{NET|INST|PIN} "ob_name" TNM = "New_name";
```

其中，“ob_name”为 NET、INST 以及 PIN 的名称，New_name 为分组的名称。例如，

```
INST ff1 TNM = MY_FF1;
INST ff2 TNM = MY_FF1;
```

将实例 ff1 与 ff2 添加到新分组 MY_FF1 中。

此外，TNM 语法也支持通配符“?”和“*”，提高了在大规模设计中添加分组约束的效率。

当 TNM 约束附加在线网上时，该路径上所有的同步元件都会被添加到分组中，但不会穿过 IBUFG 组件；当 TNM 约束附加到宏或原语的管脚上，则被该管脚驱动的所有同步元件会被添加到新分组中；当 TNM 约束附加到原语或宏上，则将原语或宏添加到新的分组中。

TNM_NET 约束专门用来完成网线的分组，与 TNM 不同的是，TNM 可以穿越 IBUFG/BUFG。因此，如果把 TNM 约束添加到端口上，只能定义该端口；而要是把 TNM_NET 添加到端口上，则可穿越 BUFG，受该端口驱动的所有组件都将被添加到分组中。

2) TIMEGRP 约束

TIMEGRP 约束用于分组合并和拆分，将多个分组形成一个新的分组。其合并分组的语法为：

```
TIMEGRP "New_group" = "Old_group1" "Old_group2" …;
```

其中，New_group 为新建的分组，而 Old_group1 和 Old_group2 以及…为要合并的已有分组。

拆分分组的语法为：

```
TIMEGRP "New_group" = "Old_group1" EXCEPT "Old_group2";
```

其中，Old_group2 是 Old_group1 的子集，New_group 为 Old_group1 中除去 Old_group2 之外所有的部分。

3) TPSYNC 约束

TPSYNC 约束用于将那些不是管脚和同步元件的组件定义成同步元件，以便利用任意点来作为时序规范的终点和起点。其相应的语法为：

```
{NET|INST|PIN} "ob_name" TPSYNC = "New_part";
```

将 TPSYNC 约束附加在网线上，则该网线的驱动源为同步点；附加在同步元件的输出管脚上，则同步元件中驱动该管脚的源为同步点；附加在同步元件上，则输出管脚为同步点；附加在同步元件的输入管脚上，则该管脚被定义成同步点。

4）TPTHRU 约束

TPTHRU 约束用于定义一个或一组路径上的关键点，可使用户定义出任意期望的路径。其相应的语法为：

```
{NET|INST|PIN} "ob_name" TPTHRU = "New_name";
```

例如，在图 11-15 所示场景中，从 A1 到 A2 有两条路径，其中逻辑 1 的延迟很大，需要提取出来完成特定的约束：

```
INST "A1" TNM = "S";
INST "A2" TNM = "E";
NET "A1toA2_1" TPTHRU = "M";
TIMESPEC "SME" = FROM "S" THRU "M" TO "B" 10;
```

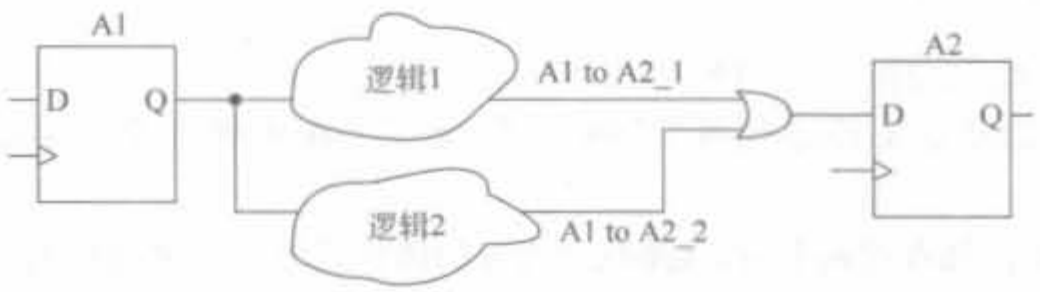


图 11-15　TPTHRU 约束示例场景

其中，第三句指令利用 TPTHRU 定义了中间点“M”，第 4 句在此基础上定义了通过 M 点的整条路径，从两条平行的路径中挑出了期望路径。

4. 局部约束

局部约束包括 FROM_TO 约束、最大延时约束、最大偏移约束、虚假路径、系统时钟抖动约束、多周期路径和多时钟域约束等。在实际开发中，正如本章前文所述，时序是设计出来，而不是靠约束自动得到的，因此这里不再对局部约束作过多讨论，有兴趣的读者可参阅文献[7,9]。

11.3.2　使用约束编辑器添加时序约束

在 FPGA 开发中，映射前输入的约束称为逻辑约束，保存在 UCF 文件中，其语法如 4.4.2 节所述，比较麻烦，容易出错。因此，Xilinx 提供了图形化的约束编辑器(Constraints Editor)，用以降低创建和修改时序约束的难度，提高工作效率。

1. Constraints Editor 用户界面

有两种方法启动约束编辑器：一种是直接运行“开始”→“程序”→“Xilinx ISE 6”→“Accessories”→“Constraints Editor”命令；另一种就是在 ISE 中新建 UCF 文件，在

"Processes for Source"窗口中双击"Creating Timing Constraints"。Constraints Editor 的用户界面如图 11-16 所示。

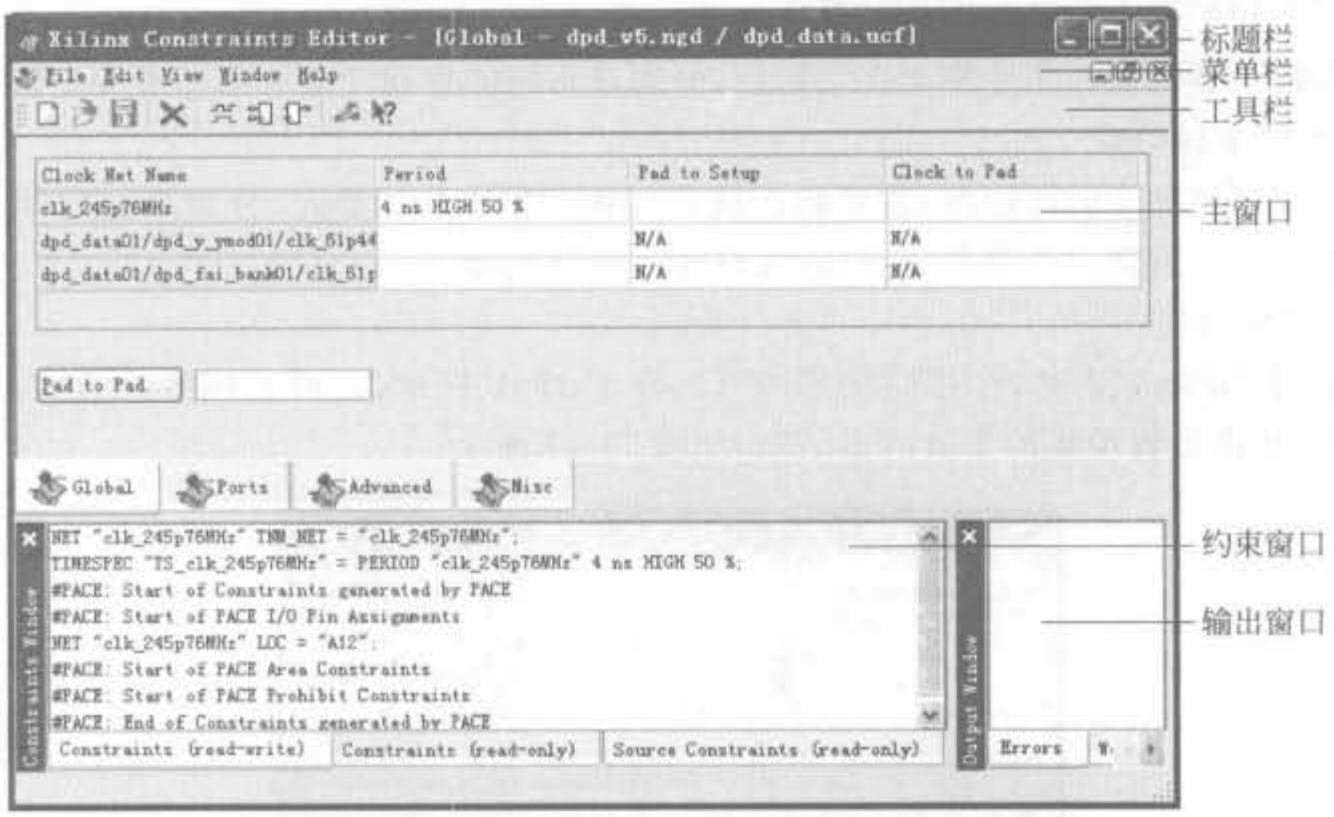


图 11-16 Constraints Editor 的用户界面

约束编辑器的用户界面主要由标题栏、菜单栏、工具栏、主窗口、约束窗口以及输出窗口组成，其主要功能如下所述。

1）菜单栏

菜单栏中的"文件(File)"、"视图(View)"、"窗口(Window)"、"帮助(Help)"菜单都是标准的 Window 组件，只有"编辑(Edit)"菜单包含了编辑约束的命令，包括删除约束(Delete)、附加周期约束(Period)、附加输入延迟约束(Pad to Setup)以及附加输出延迟约束(Clock to Pad)。

2）工具栏

工具栏的快捷键主要集成了"编辑"菜单中的命令，如下所列：

：删除按键，和"Edit"→"Delete"命令等效；

：周期设置按键，和"Edit"→"Period. . "命令等效；

：输入延迟约束按键，和"Edit"→"Pad to Setup. . "命令等效；

：输出延迟约束按键，和"Edit"→"Clock to Pad. . "命令等效。

3）主窗口

主窗口主要用于附加约束，包括全局(Global)、端口(Ports)、高级(Advanced)以及其他(Misc)这 4 个选择页面，可分别用于附加全局约束、端口约束、分组和时序约束以及其他约束，是用户操作最多的窗口。

4）约束窗口

在主窗口添加的任何操作都将被加载到 UCF 文件中，而约束窗口就是用来查看 UCF 文件的。在"UCF Constraints(read-write)"页面中直接双击约束语句，可弹出相应的编辑对话框，允许用户直接修改。

5）输出窗口

输出窗口主要输出约束编辑器的状态和控制信息，给予用户相应的提示。

2. Constraints Editor 的常用操作

约束编辑器的操作主要集中在主窗口，分别在不同的页面完成不同类型的配置。

1）附加全局约束

在完成全局约束的添加时，需要将主窗口切换到“Global”页面，分别附加周期、输入与输出延迟约束。需要注意的是，全局约束只针对时钟信号。

（1）周期约束

切换到“Global”页面，在相应信号的“Clock Period”栏单击，可直接修改数值；也可以双击该栏，弹出设置项更加丰富的对话框，如图 11-17 所示。

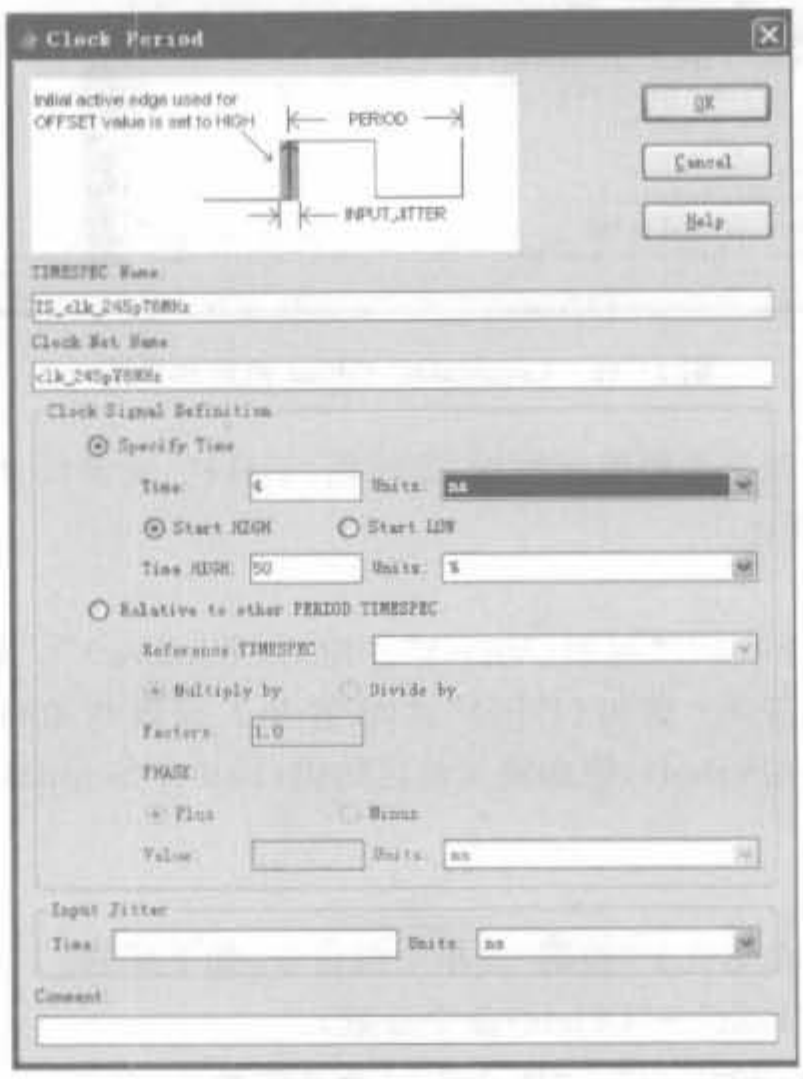


图 11-17　周期约束设置对话框

其中，在“TIMESPEC Name”文本框中输入 TS_signalname 格式的时序规范名；“Clock Net Name”文本框用于输入需要约束的信号名；在“Clock Signal Definition”栏“Specify Time”选项的“Time”文本框中输入约束周期的具体数值，单位在“Units”下拉框中选择；“Time HIGH”文本框用于设定占空比；“Input Jitter”文本框用于设定时钟抖动；“Comment”文本框为该周期约束的注释部分，可添加所附加数值的原因等，便于以后检查。完成如图 11-17 所示的配置后，可在约束窗口看到添加了下述语句：

```
NET "clk_245p76MHz" TNM_NET = "clk_245p76MHz";
TIMESPEC "TS_clk_245p76MHz" = PERIOD "clk_245p76MHz" 4ns HIGH 50 %;
```

(2) 输入延迟约束

切换到“Global”页面，在相应信号行的“Pad to Setup”列单击，可直接修改输入延迟数值；也可以双击该栏，弹出设置项更加丰富的对话框，如图 11-18 所示。

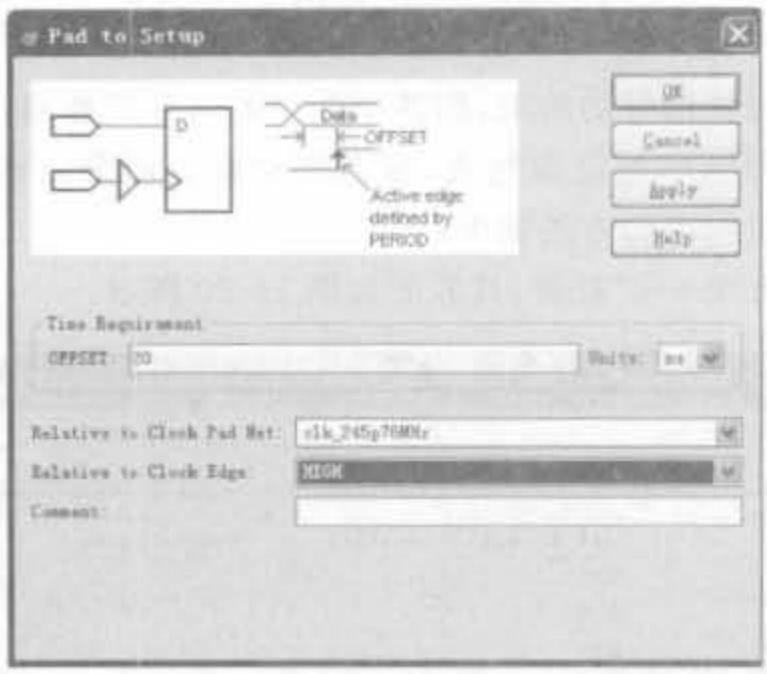


图 11-18　输入延迟约束设置对话框

在“Time Requirement”栏的“OFFSET”文本框中输入延迟数值，在“Units”下拉框中选择单位；在“Relative to Clock Pad Net”下拉表中选择输入数据的时钟；在“Relative to Clock Edge”下拉表中选择时钟有效沿，有“HIGH”和“LOW”两个选项，分别对应上升沿和下降沿；“Comment”文本框用于添加注释。完成如图 11-18 所示的配置后，可在约束窗口看到添加了下述语句：

```
OFFSET = IN 20ns BEFORE "clk_245p76MHz" HIGH;
```

(3) 输出延迟约束

切换到“Global”页面，在相应信号行的“Clock to Pad”列单击，可直接修改输出延迟数值；也可以双击该栏，弹出设置项更加丰富的对话框，如图 11-19 所示。

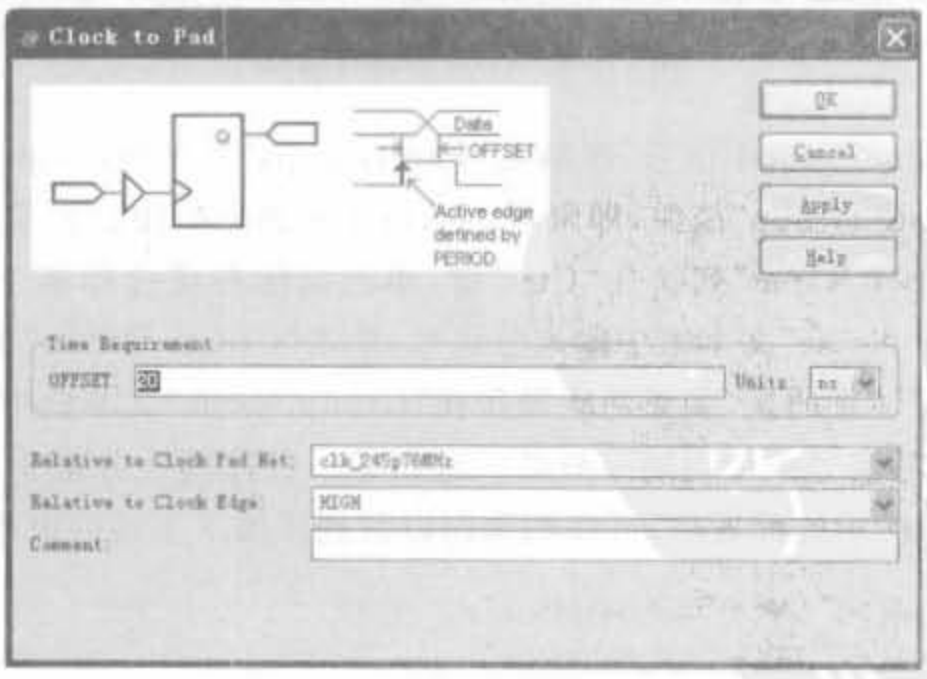


图 11-19　输出延迟约束设置对话框

其中，各项参数的意义和输入延迟配置界面的参数一样。完成如图 11-19 所示的配置后，可在约束窗口看到添加了下述约束语句：

```
OFFSET = OUT 20ns AFTER "clk_245p76MHz" HIGH;
```

2）附加端口约束

全局约束只能添加时钟信号的约束，而各个端口的约束需要通过附加端口约束来实现。在以前的低版本 ISE 中可以添加位置约束，现在则只能完成分组端口的分组约束、输入约束与输出约束。添加分组约束的方法如下：

（1）将主窗口切换到“Ports”页面，其界面如图 11-20 所示。

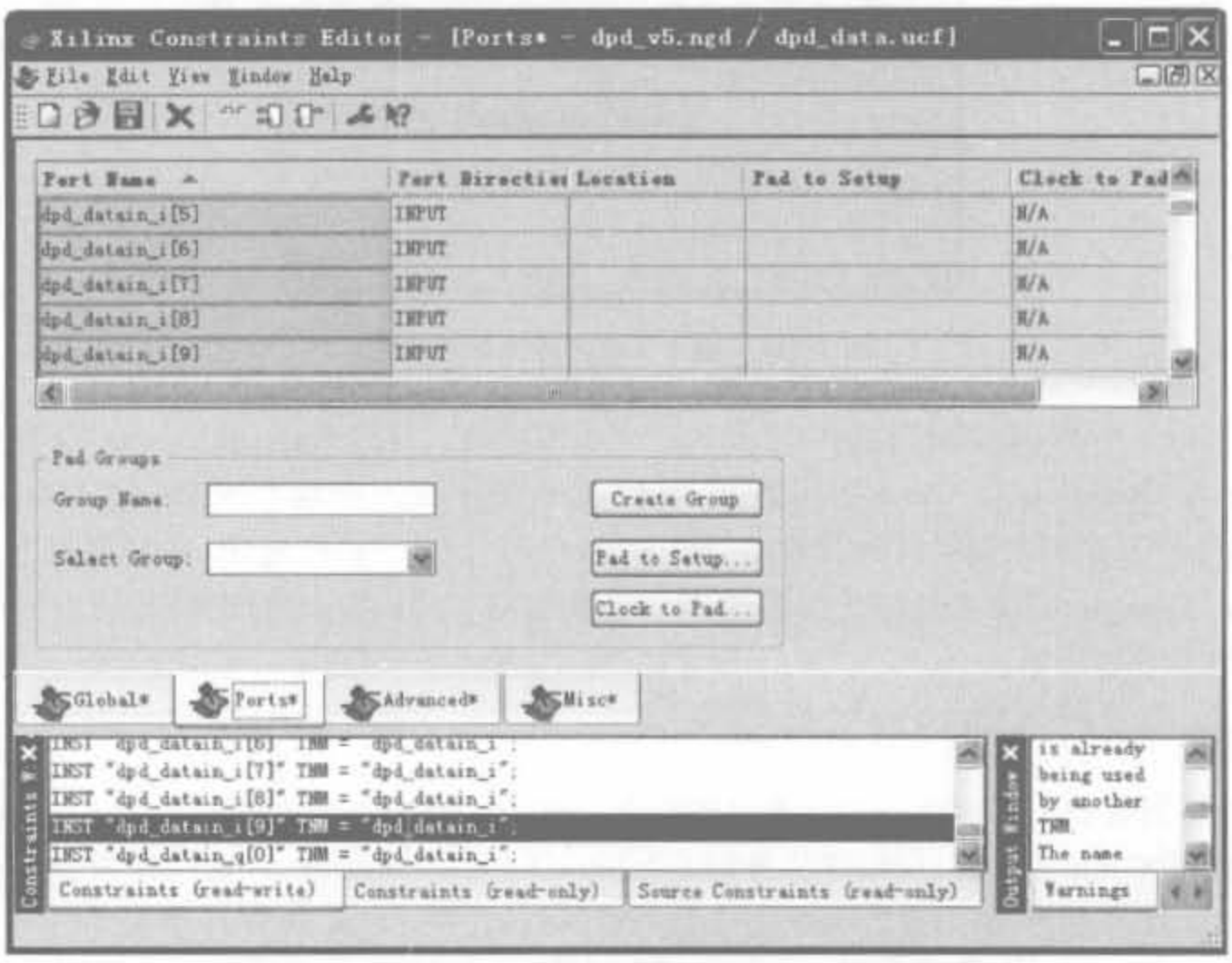


图 11-20　Ports 设置对话框

（2）在“Port Name”列选中信号，然后在“Pad Groups”栏的“Group Name”文本框中输入分组号，单击“Create Group”按钮，即可将该端口添加到分组中。如果要加入多个端口到一个分组中，可在“Port Name”列按住“Ctrl”键，单击鼠标左键选择多个信号，然后在“Pad Groups”栏的“Group Name”文本框中输入分组号，单击“Create Group”按钮，加入一组信号到特定分组中。需要注意的是，如果两次操作在“Group Name”文本框中输入的分组名相同的话，会将两次操作的信号添加到同一分组。这也是一种往同一分组中添加多个信号的方法。完成如图 11-20 所示的配置后，可在约束窗口看到添加了下述约束语句：

```
INST "dpd_datain_i[5]" TNM = "dpd_datain_i";
INST "dpd_datain_i[6]" TNM = "dpd_datain_i";
INST "dpd_datain_i[7]" TNM = "dpd_datain_i";
INST "dpd_datain_i[8]" TNM = "dpd_datain_i";
```

```
INST "dpd_datain_i[9]" TNM = "dpd_datain_i";
```

各端口的输入、输出延迟约束的添加方法和时钟信号的操作是一样的，这里就不再介绍。

3）附加高级约束

高级约束在“Advanced”页面设置，主要包括 TNM/TNM_NET/TPT_THRU 等分组约束和 OFFSET/FROM_TO 等时序约束。由于操作具有相似性，下面只介绍 TNM_NET 分组约束和源同步输入延迟时序约束这两个具有代表性的操作，其余约束的附加方法都是类似的。“Advanced”页面的界面如图 11-21 所示。

图 11-21　Advanced 页面示意图

（1）添加 TNM_NET 分组约束

在“Grouping”栏单击“Group elements associated by Nets（TNM_NET）”行的“Create..”按钮，会弹出图 11-22 所示的对话框。在“Time Name”文本框中输入新的分组名；“Design Element Type”下拉框用于选择网线类型，有“All Nets”、“Clock Nets”和“Enable Nets”3 种选择；“Filter”栏帮助用户在大量网线中快速挑出期望的线网；“Comment”文本框用于添加注释。然后，在“Available”栏中选择网线类型，单击“Add”按钮

添加，直到添加完为止，同时还有“Remove”、“Add All”以及“Remove All”等操作，所有选中的网线名均会在“Time Name Targets”栏列出。

图 11-22　TNM_NET 分组设置对话框

完成如图 11-22 所示的配置后，单击“OK”按钮，即可完成整个过程。在约束窗口，可以看到添加了下列语句：

```
NET "clk_245p76MHz" TNM_NET = "my_net_245p76MHz";
NET "clk_245p76MHz_ibuf/IBUFG" TNM_NET = "my_net_245p76MHz";
NET "clk_245p76MHz_c" TNM_NET = "my_net_245p76MHz";
```

(2) 添加源同步时序约束

在“Timing Constraints”栏单击 Pad to Setup for Source Synchronous(OFFSET IN BEFORE)”行的“Create..”按钮，会弹出如图 11-23 所示的对话框。在“Pad Group”下拉框中选择已建立的端口分组；“Time Requirement”栏用于输入偏移时间的数值和单位；在“Relative to Clock Pad Net”下拉表中选择输入数据的时钟；在“Relative to Clock Edge”下拉表选择时钟有效沿，有“HIGH”和“LOW”两个选项，分别对应上升沿和下降沿；“Comment”文本框用于添加注释。

完成如图 11-23 所示的配置后，单击“OK”按钮，即可完成整个过程。在约束窗口，可以看到添加了下列语句：

```
TIMEGRP "dpd_datain_i" OFFSET = IN 20ns BEFORE "clk_245p76MHz" HIGH;
```

4) 附加其他约束

Misc 页面的界面如图 11-24 所示，其配置内容基本上都是局部约束，包括 I/O 端口寄存器、异步寄存器、工作电压、温度、长线资源的使用、区域分组、DCM 反馈路径延迟以及各类寄存器与 RAM 内容初始化的控制。其中在寄存器初始化控制中，使用频率较高的就是该选项，下面对其进行简单介绍。

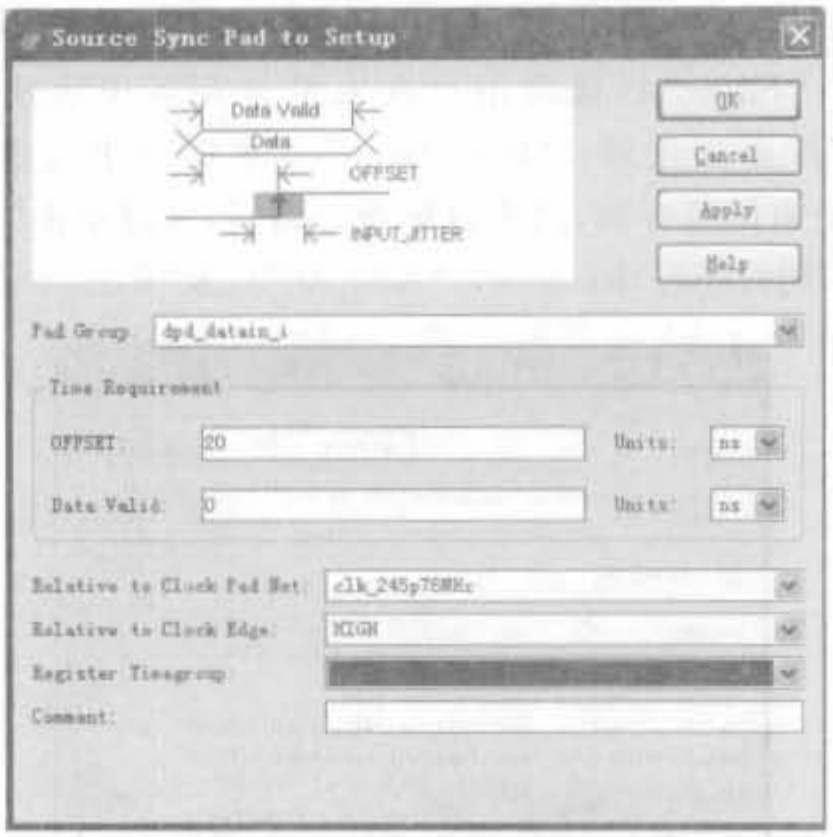


图 11-23 源同步输入延迟设置对话框

图 11-24 Misc 页面示意图

单击“INIT Values for”栏的“FFS/LATCH”按钮，则弹出寄存器初始化配置对话框，如图 11-25 所示。其中，“Filter”栏帮助用户在大量寄存器中快速挑出期望的寄存器；“Comment”文本框用于添加注释；“Init Value”文本框用于设定寄存器的初始化值；“Available”栏列出了所有的寄存器，选中目标寄存器后，通过单击“Add”按钮将其添加到“Register Targets”栏，同时还有“Remove”、“Add All”以及“Remove All”等操作。

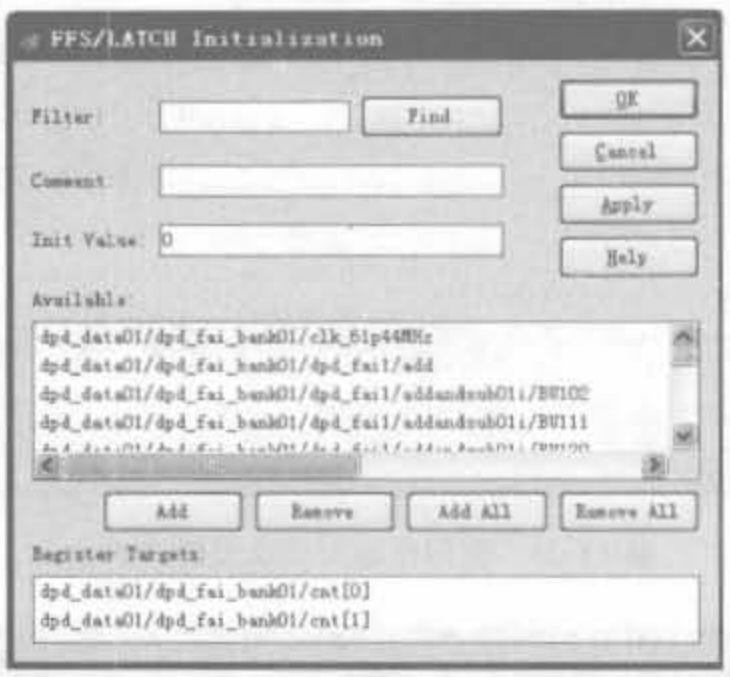


图 11-25　FFS/LATCH 初始化界面

完成如图 11-25 所示的配置后，单击“OK”按钮，即可完成整个过程。在约束窗口，可以看到添加了下列语句：

```
INST "dpd_data01/dpd_fai_bank01/cnt[0]" INIT = 0;
INST "dpd_data01/dpd_fai_bank01/cnt[1]" INIT = 0;
```

3. Constraints Editor 和 UCF 文件编辑法的比较

Constraints Editor 和 UCF 文件编辑法的目的是一样的，都是附加设计的各种约束。相比起来，前者不需要用户熟悉 UCF 文件的语法以及设计中网线、端口、触发器以及 RAM 模块的命名，且操作简单，可大幅度提高工作效率；但是后者的功能更加强大，并不是 UCF 文件的全部功能，Constraints Editor 都具备。从整体上而言，Constraints Editor 已能满足大多数设计的需求，对于初学者来讲非常方便；UCF 文件适合高级工程师在大规模的项目开发中使用。

11.4　ISE 时序分析器

Xilinx 公司的时序分析器(Timing Analyzer)为 Xilinx FPGA 和 CPLD 的设计提供了一种高效、灵活进行静态时序分析的方法，可通过简单的菜单方式对整个设计进行时序分析，并提供目录树结构的时序报告。

11.4.1　时序分析器简介

ISE 中内嵌的时序分析器通过图形用户界面(由菜单调出)或宏命令语言(在控制台命令窗口中输入)工具来控制,整体而言,具有下列功能:

- 能够完成 FPGA/CPLD 设计的静态时序分析;
- 可以在映射(Mapping)、布局(Placing)或者布线(Routing)等任一阶段后立刻执行不同层次的静态时序分析;
- 可通过用户图形界面(GUI)、批处理文件或者宏命令语言来交互运行时序分析器;
- 可以查看任一给定路径的延迟,报告其在指定约束下的时序裕量,能够以树形结构管理和显示分析结果,可以指出电路的关键路径、最高运行时钟、指定路径的延迟以及具有最大延迟的路径;
- 对那些由组合逻辑和同步时序逻辑组成的同步电路有很强的分析能力,考虑所有的路径延迟,包括 clock-to-out 和建立需求,同时计算设计中最坏情况下的时序结果;
- 根据时序约束文件和用户指定路径来生成时序分析报告;
- 以层次化结构提供时序报告,允许设计人员在报告的不同部分快速地进行切换;
- 通过插入交叉探针,可以将时序路径的结果导入第三方的综合工具(如 Synplify 等)、Floorplanner 以及 FPGA Editor 等工具。

在 Xilinx FPGA 设计中,可在多个阶段完成静态时序分析。在任何一步,只要时序分析不满足条件,就需要重新修改设计。无论时序分析是否通过,ISE 时序分析器都输出反馈信息,为用户设计的修改和优化提供一定的指导。

11.4.2　时序分析器的文件类型

时序分析器的文件类型可以分为输入和输出两大类。输入文件以物理设计和约束文件为主,输出文件包括各类时序报告。

1. 输入文件类型

FPGA 的物理设计文件的后缀为. NCD,物理约束文件的后缀为. PCF。

NCD 文件是 FPGA 设计数据库文件,在映射和布局布线阶段生成,包含了目标器件中物理资源和逻辑之间的映射信息以及布局布线信息。

PCF 文件包含了设计中的所有约束,时序约束是其中的一部分,在映射阶段,根据 UCF 文件自动生成。

2. 输出文件类型

输出文件类型较多,主要包含以下几类:

(1) 时序向导 XML 报告(TWX),其文件后缀为. TWX,包含所有时序分析结果,且只能在时序分析器中打开。

(2) 时序向导 XML 报告(TWR),其文件后缀为. TWR,内容和 TWX 一样,以通用的 ASCII 码编码,可被其他软件打开。

(3) 时序分析宏文件(XTM),其文件后缀为.XTM,是时序分析器中唯一能由用户直接编辑的文档。运行 XTM 文件,相当于将其内容在 ISE 命令控制台一一执行。

在操作中,时序分析报告是根据时序约束和用户指定的时序路径而产生的,采用层次结构管理,可方便地浏览和查找。

11.4.3 时序分析器的调用与用户界面

1. 在 ISE 中调用时序分析器

时序分析器有两种启动方法:一种是直接运行"开始"→"程序"→"Xilinx ISE 9.1"→"Accessories"→"Timing Analyzer"命令;另一种就是在 ISE 中经过综合后的模块,在多个环节直接启动时序分析器,这也意味着对于同一个设计,可以提供多份不同阶段的时序分析报告。由于前一种方式需要手工输入设计网表文件(.NCD),比较麻烦。在 ISE 中启动时序分析器是最方便的分析方法,且不易出错,是本书推荐的调用方法。

在 ISE 软件中,从"Source"视图中选中顶层设计模块,在相应的当前资源操作窗口中的实现阶段双击"Analyze Post-Map Static Timing(Timing Analyzer)"或"Analyze Post-Place & Route Static Timing(Timing Analyzer)"就可以启动时序分析器,如图 11-26 所示。不同阶段时序分析的输入文件(.NCO 文件或.VM6 文件)由 ISE 自动生成。

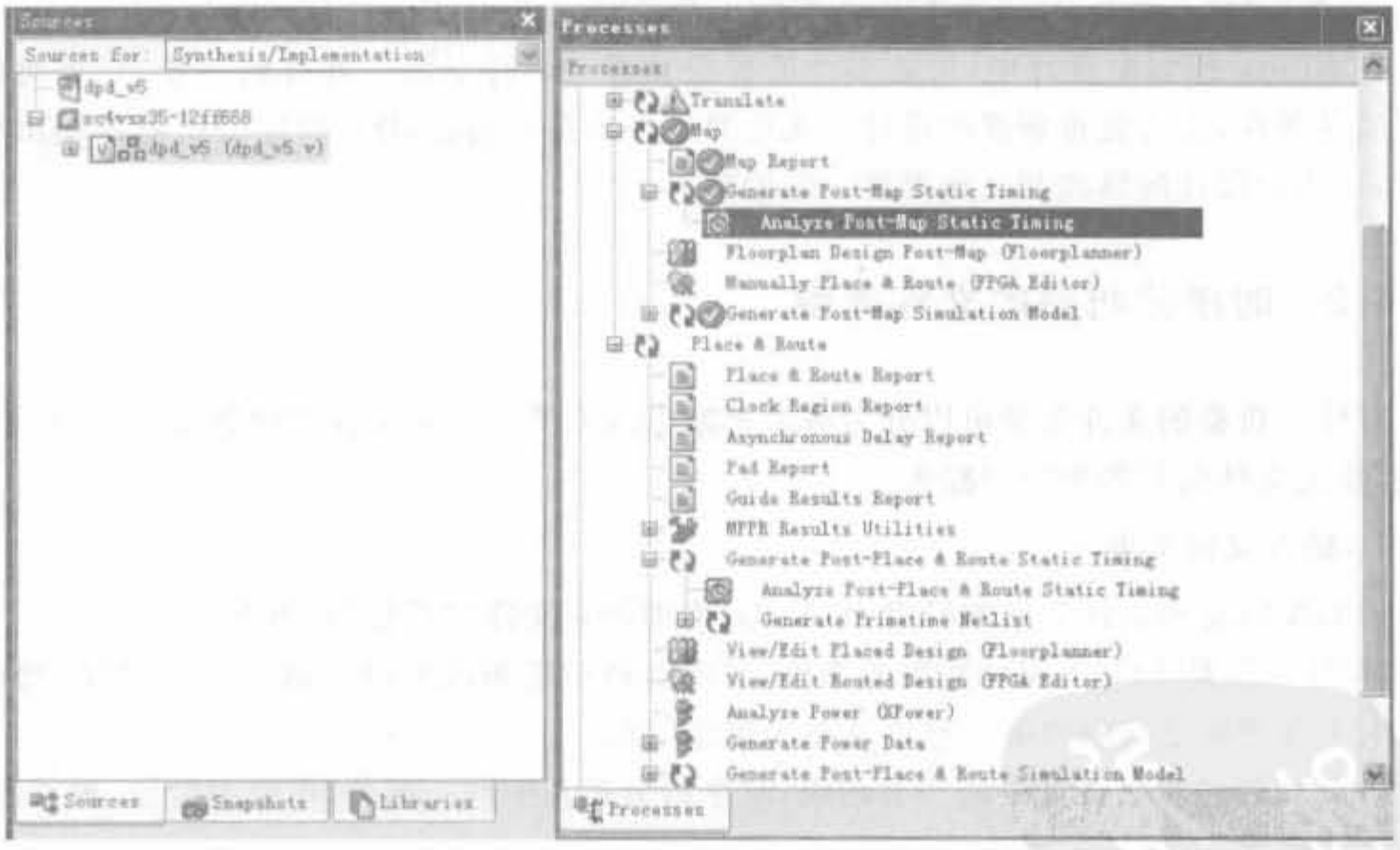


图 11-26 ISE 中启动时序分析器的操作界面

时序分析器可在经过映射(MAP)或布局布线(PAR)的工程中打开,自动加载 NCD 文件和 PCF 文件。在时序分析器中,用户可根据实际需求分析电路并产生类型丰富的时序报告。

2. 时序分析器的用户界面

在布局布线后启动时序分析器,其界面如图 11-27 所示。其中包含了几个不同的区域,

左边为时序报告的分类层次，右边为具体的时序描述，其中的蓝色超链接指示表明可以与布局规划器进行交叉探查。工具栏中的按钮 专门用于生成时序分析报告，其中所有的参数都取默认值。

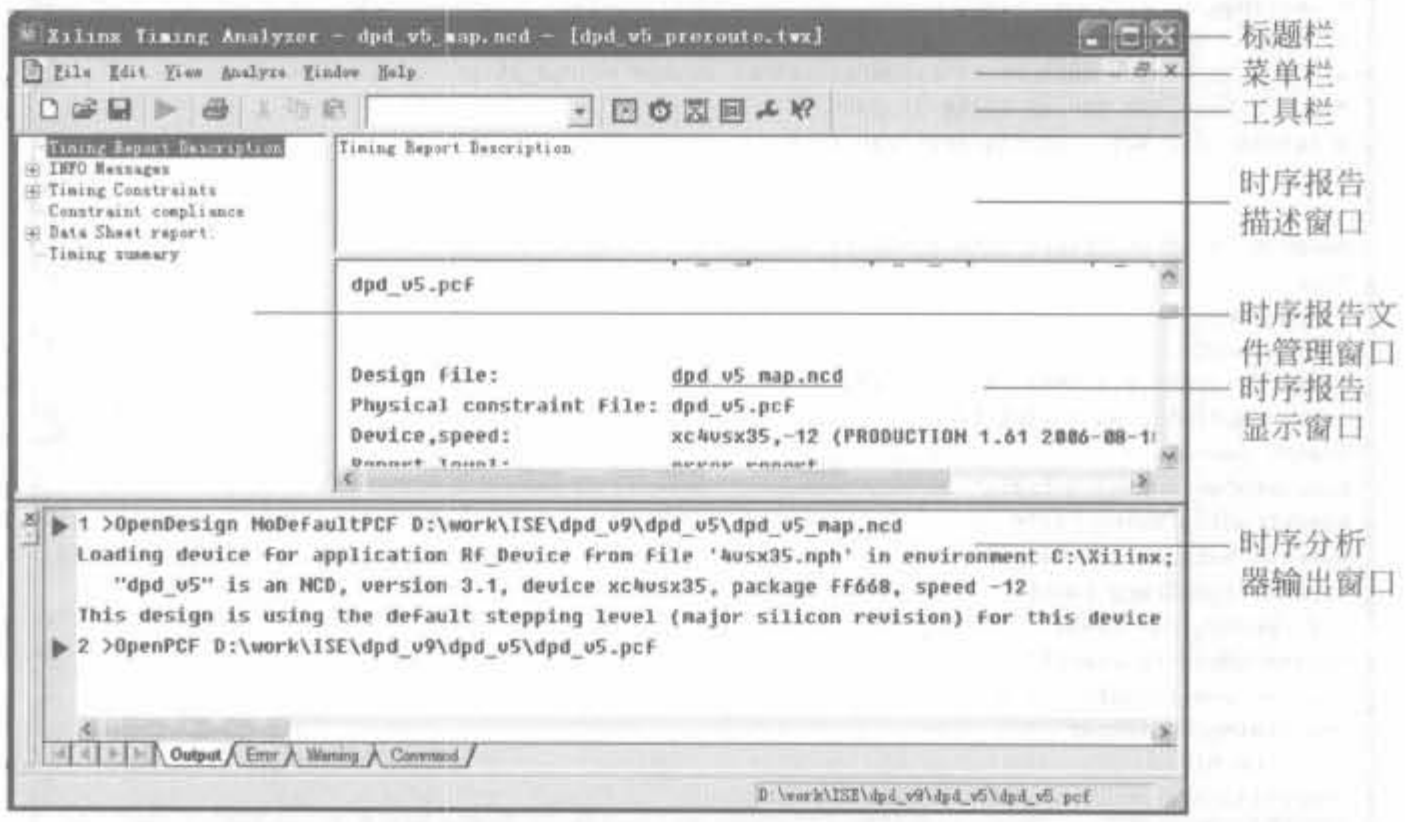


图 11-27　时序分析器的用户界面

时序分析器的用户界面主要由标题栏、菜单栏、工具栏、时序报告文件管理窗口、时序报告描述窗口、时序报告显示窗口以及输出窗口组成，其主要功能如下所述。

1）菜单栏

菜单栏包括“文件(File)”、“查看(View)”、“编辑(Edit)”、“分析(Analyze)”、“窗口(Window)”以及“帮助(Help)”等菜单栏。

(1) “File”菜单

“File”菜单用于设计文件管理，主要命令包括：“New Macro”用于创建一个新的宏，其格式为 XTM 文件；“Open Design”用于打开设计文件；“Open”用于打开时序分析器文件；“Apply Physical File”用于替换原有的约束文件；“Close”/“Close Design”/“Close All”分别用于关闭时序报告、关闭设计以及关闭工程；“Run Macro”用于运行宏；“Page Setup”用于设定页面大小；“Save/Save Design/Save All”分别用于保存报告、设计和整个工程。

(2) “View”菜单

“View”菜单用于设置显示窗口，当选择某窗口后，将会自动弹出相应的界面。其主要命令包括：“Standard Toolbar”用于显示标准工具栏，默认为始终打开；“Status Bar”用于打开状态栏，简要显示所选择的命名和执行的状态，默认为始终打开；“Console”用于显示控制台，记录所有已执行的操作和命令；“Clock”用于显示时钟窗口，该命令可建立时钟报告；“Seeting”用于打开时序分析器设置对话框，如图 11-28 所示；“Floorplanner for Crossprobing”用于启动布局规划器 Floorplanner，进行交叉探点分析；“FPGA Editor for Crossprobing”用于启动底层编辑器 FPGA Editor，进行交叉探点分析。

图 11-28 中的“Open PCF”项表明了物理约束文件；“Speed”项显示了芯片的速度等级；

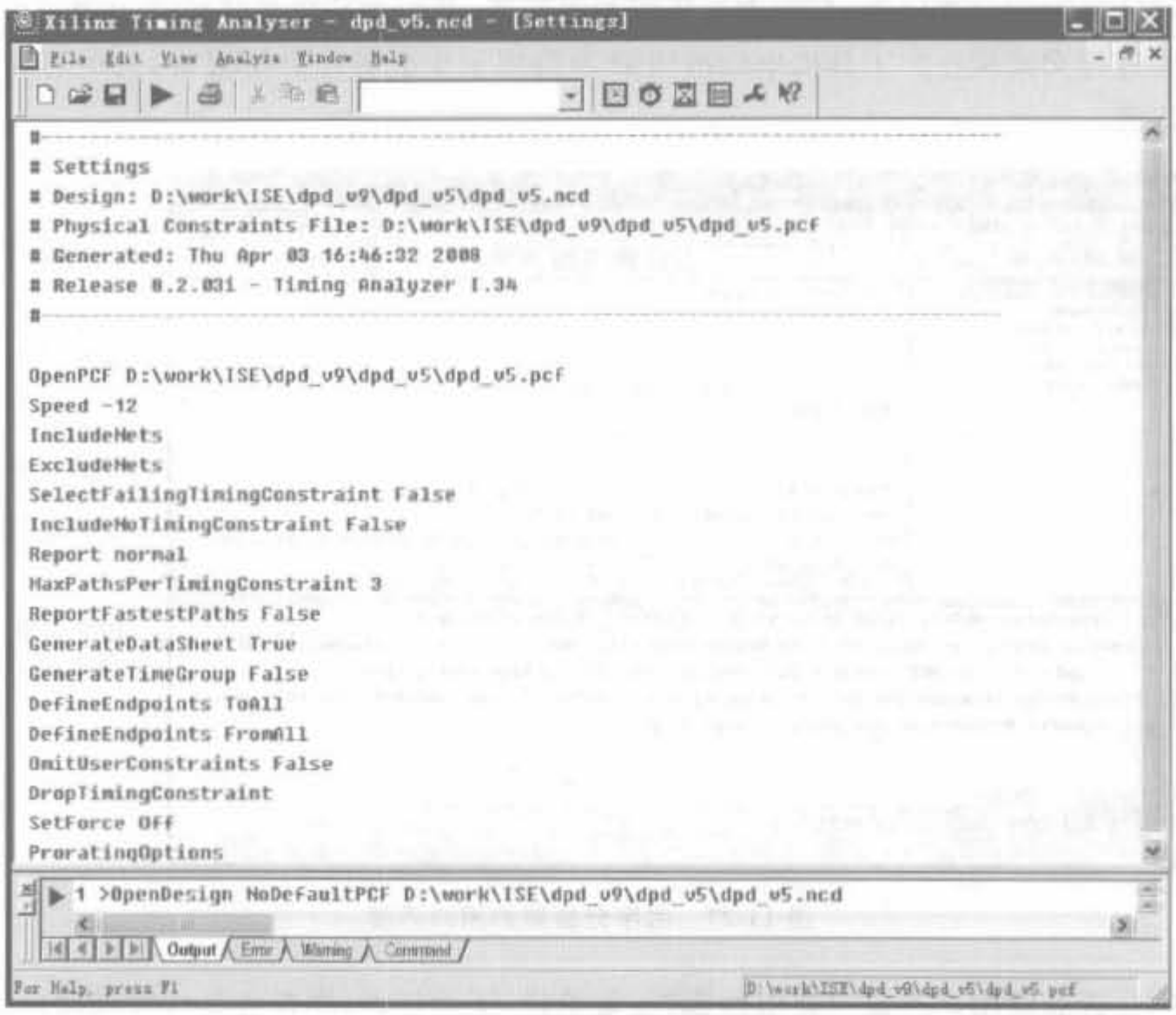


图 11-28 “View”菜单“Setting”对话框示意图

“IncludeNets”是时序分析时包含的网线，默认值为包含所有网线；“ExcludeNets”是时序分析时不包含的网线；“SelectFailingTimingConstraint”指定时序报告是否仅出现不满足时序约束的通道，可选参数为 True 和 False，默认值为 False；“IncludeNoTimingConstraint”指定时序报告是否仅出现不受约束的通道，可选参数为 True 和 False，默认值为 False；“Report normal”指定时序报告的字符/行的宽度；“MaxPathsPerTimingConstraint”指定约束允许显示的最大通道数；“ReportFastestPaths”用于指定是否只显示最快的路径，可选参数为 True 和 False，默认值为 False；“GenerateDataSheet”指定是否产生数据表格，默认值为 True；“GenerateTimeGroup”指定是否产生时序分组，默认值为 False；“DefineEndpoints”指定是否列出所选终/起点上的所有资源；“OmitUserConstraints”指定是否对用户设置部分禁止约束，可选参数为 True 和 False，默认值为 False；“DropTimingConstraint 用于指定是否丢弃部分时序约束；“SetForce”指定是否关闭控制台输出窗口的显示功能，On 表示关闭，Off 表示打开，默认值为 Off；“ProratingOptions”指定是否需要列出控制电压和温度的分配等。

（3）“Analyze”菜单

“Analyze”菜单是时序分析器的关键操作所在，主要用于调用、控制时序分析器的各个功能和模块。其主要命令包括：“Against Timing Constraints”执行基于时序约束的时序分析，将弹出参数设置和分析对话框，如图 11-29 所示；“Against Auto Generated Design Constraints”

执行基于自动生成约束的时序分析；“Against User Specified Paths”执行基于用户指定路径的时序分析，有两种方法：一种是基于用户定义端点之间的延迟进行分析，另一是基于用户定义时钟和 I/O 时序进行分析，如图 11-29 所示；“Query Nets”执行可疑网线分析；“Query Timinggroups”执行可疑时序组分析；“Reset All Path Filters”复位所有的路径过滤器。

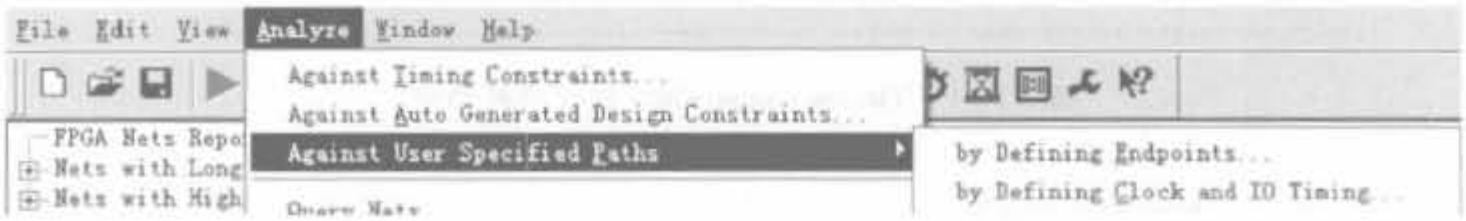


图 11-29　用户指定路径分析的两种模式

(4) 其余菜单

“Edit”菜单、“Window”菜单和“Help”菜单是 Windows 应用程序的典型控件，就不再介绍。

2) 工具栏

工具栏的快捷键主要集成了“File”菜单和“Analyze”中的命令，如下所列：

▶：运行宏，和“File”菜单中的“Run Macro”命令等效。

🖸：基于时序约束的时序分析，和“Analyze”菜单中的“Against Timing Constraints”命令等效。

⏲：基于自动生成约束的时序分析，和“Analyze”菜单中的“Against Auto Generated Design Constraints”命令等效。

🖸：基于用户定义端点的特定路径的时序分析，和“Analyze”菜单中的“Against User Specified Paths”→“by Defining Endpoints..”命令等效。

🖸：基于用户定义时钟和 I/O 时序的特定路径的时序分析，和“Analyze”菜单中的“Against User Specified Paths”→“by Defining Clock and I/O Timing..”命令等效。

3) 时序报告文件管理窗口

时序报告文件管理窗口采用层次化的方法管理时序报告，还能管理用户所添加的时序约束。其基本内容如图 11-30 所示，包括“INFO Messages”、“Timing Constraints”、“Constraint compliance”、“Data Sheet report”以及“Timing summary”5 个部分。

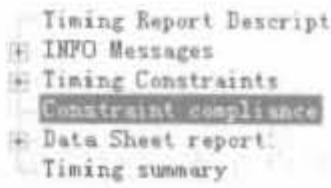


图 11-30　时序报告基本内容示意图

“Timing Constraints”中的内容是用户比较关心的，它列出了用户添加的约束和经过实现后在芯片内部的实际值，并会给出在时序分析时，针对某个特定约束所分析的元件数以及时序错误数。典型的分析结果如图 11-31 所示，设定时钟信号 clk_245p76MHz 的周期为 4ns，占空比为 50%，经过时序分析后，实际值为 3.992ns。过程中分析了 48 274 个元件，没有时序错误(未发生建立时间和保持时间不足的错误)。

“Data Sheet report”会给出设计中完整的、详细的分析结果，包括各个约束信号到相关的内部寄存器的详细时序参数，便于用户定位时序瓶颈，优化设计。图 11-32 给出了典型的数据手册报告，列出了所有端口信号、内部寄存器的输入建立时间和输出保持时间等参数。

```
================================================================================
Timing constraint: TS_clk_245p76MHz = PERIOD TIMEGRP "clk_245p76MHz" 4 ns HIGH 50%;

 48274 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum period is   3.992ns.
--------------------------------------------------------------------------------

================================================================================
```

图 11-31　Timing Constraints 分析结果示例

```
Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk_245p76MHz
------------------+------------+------------+------------------+---------+
                  |  Setup to  |  Hold to   |                  |  Clock  |
Source            | clk (edge) | clk (edge) |Internal Clock(s) |  Phase  |
------------------+------------+------------+------------------+---------+
dpd_datain_i[0] |   -1.727(R)|    2.684(R)|clk_245p76MHz_c   |   0.000|
dpd_datain_i[1] |   -2.271(R)|    3.184(R)|clk_245p76MHz_c   |   0.000|
dpd_datain_i[2] |   -2.071(R)|    3.015(R)|clk_245p76MHz_c   |   0.000|
```

图 11-32　Data Sheet report 结果示例

"Timing summary"顾名思义就是时序分析总结报告，会给出时序分析的错误数、约束覆盖的路径数、网线数、连接数以及分析完成时间，其典型输出报告如图 11-33 所示。

```
Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 52205 paths, 0 nets, and 15309 connections

Analysis completed Thu Apr 03 15:06:19 2008
--------------------------------------------------------------------------------
```

图 11-33　时序约束报告示例

4）时序报告描述窗口

当用户在查阅时序报告时，时序报告描述窗口会列出当前用户查看部分所归属的时序约束。在时序出错时，这个窗口使用起来非常方便。

5）时序报告显示窗口

时序报告显示窗口主要用于显示时序分析输出结果，所有结果都需要在该窗口中查看。

6）输出窗口

输出窗口主要输出约束编辑器的状态和控制信息、出错和警告信息，给予用户相应的提示。

3. 时序分析报告解读

时序报告并不只在时序分析器中才能查阅。在设计中添加了时序约束后，经过实现时已经完成了时序分析，并会在 Place & Route Report 中给出。因此，用户解读时序报告的第一步就是查阅设计的布局布线报告。布局布线报告中典型的时序分析结果如图 11-34 所示。该报告会简要列出用户添加的约束，并给出实际达到的数值。如果没有达到时序要求，

会以“*”号在相应的约束中标出，并给出未满足的约束数目。另外，将列出每个约束覆盖路径的逻辑级数(Logic Levels)，逻辑级数越大，则时序性能越差。由于图 11-34 中所有的时序约束都满足用户要求，因此报告中显示“All constraints were met.”。

```
Timing Score: 0

Asterisk (*) preceding a constraint indicates it was not met.
   This may be due to a setup or hold violation.

----------------------------------------------------------------------------------------------------
  Constraint                                    | Requested  | Actual     | Logic  | Absolute   |Number of
                                                |            |            | Levels | Slack      |errors
----------------------------------------------------------------------------------------------------
  TS_clk_245p76MHz = PERIOD TIMEGRP "clk_24     | 4.000ns    | 3.992ns    | 0      | 0.008ns    | 0
  5p76MHz" 4 ns HIGH 50%                        |            |            |        |            |
----------------------------------------------------------------------------------------------------
  OFFSET = OUT 20 ns AFTER COMP "clk_245p76     | 20.000ns   | 9.661ns    | 1      | 10.339ns   | 0
  MHz" HIGH                                     |            |            |        |            |
----------------------------------------------------------------------------------------------------
  OFFSET = IN 20 ns BEFORE COMP "clk_245p76     | 20.000ns   | 5.248ns    | 3      | 14.752ns   | 0
  MHz" HIGH                                     |            |            |        |            |
----------------------------------------------------------------------------------------------------
  TIMEGRP "dpd_datain_1" OFFSET = IN 20 ns      | 20.000ns   | -1.727ns   | 2      | 21.727ns   | 0
  BEFORE COMP "clk_245p76MHz" HIGH              |            |            |        |            |
----------------------------------------------------------------------------------------------------

All constraints were met.
```

图 11-34 布局布线报告中的时序分析报告

如果时序分析不满足，可打开时序分析器完成进一步分析。时序分析报告可通过单击时序报告管理窗口的“Timing Constraints”栏目查阅。静态时序分析报告的内容是比较全面和复杂的，下面分类介绍其主要组成部分。

1) 周期约束报告

周期约束报告是时序分析报告中最基本的组成部分，每个信号的周期分析都由图 11-35 所示的多个 Slack 段组成。其中的蓝色超链接为交叉探查的分析和对设计的局部布局进行调整提供了方便。每行信息的含义为：

```
--------------------------------------------------------------------------------

Slack:                  8.888ns (requirement - (data path - clock path skew + uncertainty))
  Source:               dpd_data01/dpd_y_ymod01/dpd_ss0_1_2_3_4/b2_i[13] (FF)
  Destination:          dpd_data01/dpd_y_ymod01/dpd_ss0_1_2_3_4/mult_01i/BU2/U0/virtex4.pmf.v4pm48/single_dsp.v4_parr
  Requirement:          4.000ns
  Data Path Delay:      3.899ns (Levels of Logic = 0)
  Clock Path Skew:      -0.093ns
  Source Clock:         clk_245p76MHz_c rising at 0.000ns
  Destination Clock:    clk_245p76MHz_c rising at 4.000ns
  Clock Uncertainty:    0.000ns

  Data Path: dpd_data01/dpd_y_ymod01/dpd_ss0_1_2_3_4/b2_i[13] to dpd_data01/dpd_y_ymod01/dpd_ss0_1_2_3_4/mult_01i/BU2
    Delay type            Delay(ns)  Logical Resource(s)
    ----------------------  ---------  -----------------
    Tcko                    0.258   dpd_data01/dpd_y_ymod01/dpd_ss0_1_2_3_4/b2_i[13]
    net (fanout=2)          0.671   dpd_data01/dpd_y_ymod01/dpd_ss0_1_2_3_4/b2_i[13]
    Tdspdck_BPIDL           2.970   dpd_data01/dpd_y_ymod01/dpd_ss0_1_2_3_4/mult_01i/BU2/U0/virtex4.pmf.v4pm48/single_d
    ----------------------  ---------  ---------------------------
    Total                   3.899ns (3.228ns logic, 0.671ns route)
                                    (82.8% logic, 17.2% route)
```

图 11-35 周期分析报告基本段落示意图

(1) Slack 行给出了时序裕量(Slack)的计算公式：

$$\text{Slack} = \text{requirement} - (\text{data path} - \text{clock path skew} + \text{uncertainty}) \quad (11\text{-}6)$$

等号右端变量的值在 Slack 行下面分别给出，因此图 11-35 中的裕量 0.008ns 的计算过程为：

$$\text{Slack} = 4 - (3.899 - (-0.093) + 0) = 0.008\text{ns} \tag{11-7}$$

如果 Slack 为负值，表示时序分析失败。

(2) Source：指明了源端元件以及基本类型，图 11-35 中为触发器(FF)。

(3) Destination：指明了源端元件以及基本类型，图 11-35 中为硬核乘法器(DSP)。

(4) Requirement：指明了所约束的时钟周期，图 11-35 中为 4ns。

(5) Data Path Delay：给出了路径延迟及逻辑级数(Logic Levels)。图 11-35 中的路径延迟为 3.899ns，逻辑级数为 0，这是由在设计中将寄存器的值送到硬核乘法器输入端而造成的。

(6) Clock Path Skew：时钟路径偏移，图 11-35 中为－0.093ns。

(7) Source/Destination Clock：分别为源/目的时钟，在图 11-35 中为同一时钟，且相差 1 个周期。

(8) Clock Uncertainty：时钟不确定度，图 11-35 中为 0ns。

(9) Data Path：数据路径，按照自顶向下的方式列出模块路径，一直到网线为止，具有超链接功能，用户可单击查看。在一个 Slack 分析单元中，可能会存在多条数据路径。

(10) Delay type：数据类型，以列表的形式给出各个路径的延迟，用户可单击每项数值进行交叉探点分析。其中的 Tcko(同步元件时钟输出时间)在单击后可链接到 Xilinx 网站，给出延迟类型和路径的详细结构说明。

(11) Total：基本分析单元时序小结，会给出设计逻辑和布线所占的百分比。在图 11-35 中，逻辑用时 3.228ns，布线用时 0.671ns，二者比值分别为 82.8%、17.2%。在一般设计中，二者最佳比例为 60%/40%。

当时序分析出错后，在时序分析器中，各个错误会用红色的超链接标出，便于用户逐级跟踪。选定不匹配路径后，会显示造成时序错误的原因和具体数值，同时提供“Timing Improvement Wizard”选项，单击后将弹出时序设计向导和改进建议。

2) OFFSET_IN/OFFSET_OUT 约束报告

OFFSET_IN/OFFSET_OUT 分析报告也由多个 Slack 段组成。典型的 Slack 报告如图 11-36 和图 11-37 所示。其中，各个参数的含义与周期报告中的一致。对于每个 Slack 段，一般含有多个数据路径，每条数据路径都有自身的详细分析报告，其内容也和周期报告中的内容类似。

```
--------------------------------------------------------------------------------
Timing constraint: OFFSET = IN 20 ns BEFORE COMP "clk_245p76MHz" HIGH;

 3891 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum allowable offset is   5.248ns.
--------------------------------------------------------------------------------

Slack:                  14.752ns (requirement - (data path - clock path - clock arrival + uncertainty))
  Source:               reset (PAD)
  Destination:          dpd_filter01/dpd_mac_coeff01/dpd_sr116_02/BU2/U0/gen_sr116_ram.srlram_inst (RAM)
  Destination Clock:    clk_245p76MHz_c rising at 0.000ns
  Requirement:          20.000ns
  Data Path Delay:      0.878ns (Levels of Logic = 3)
  Clock Path Delay:     3.630ns (Levels of Logic = 2)
  Clock Uncertainty:    0.000ns
```

图 11-36　OFFSET_IN 报告基本段落示意图

```
--------------------------------------------------------------------------------
Timing constraint: OFFSET = OUT 20 ns AFTER COMP "clk_245p76MHz" HIGH;

 32 items analyzed, 0 timing errors detected.
 Minimum allowable offset is   9.661ns.
--------------------------------------------------------------------------------
Slack:                  10.339ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:               dpd_filter01/dpd_mac_adder/dpd_out_q[0] (FF)
  Destination:          dpd_out_q[0] (PAD)
  Source Clock:         clk_245p76MHz_c rising at 0.000ns
  Requirement:          20.000ns
  Data Path Delay:      5.483ns (Levels of Logic = 1)
  Clock Path Delay:     4.178ns (Levels of Logic = 2)
  Clock Uncertainty:    0.000ns
```

图 11-37　OFFSET_OUT 报告基本段落示意图

11.4.4　时序分析器的基本使用方法

如前所述，时序分析器提供了 3 种时序分析方法，即基于时序约束、基于自动约束以及基于用户定义路径的方法。下面分别介绍其基本操作。

1. 基于时序约束的时序分析方法

基于时序约束的时序分析方法在用户添加时序约束文件的基础上，分析约束所覆盖的路径，是设计人员最常使用的方式。可通过两种方式来启动时序约束。

1）工具栏启动

当用户没有特别要求时，可通过单击工具栏的“ ”按钮，按照系统默认参数来启动时序分析器。单击该按钮后，时序分析器会弹出如图 11-38 所示的窗口，等其自动消失后，即完成了相关的时序分析。

图 11-38　时序分析器提示对话框

2）菜单栏启动

通过菜单栏启动可以设置更多的参数，使分析场景更满足用户需求。单击“Analyze”菜单下的“Against Timing Constraints”命令，即可弹出配置对话框，有“Timing Constraints”、“Options”、“Filter Paths by Net”以及“Path Tracing”等 4 个配置页面，如图 11-39 所示。

“Timing Constraints”配置页面如图 11-39 所示，用于为时序分析提供报告，指定通道和时序约束属性，参数含义如下：“Report paths against timing constraints”用于设定将时序约束覆盖的全部路径添加到时序报告中，默认值为选择；“Report failing paths against timing constraints”用于设定只将发生时序冲突的路径添加到时序报告中，默认值为不选择；“Report unconstrained paths”用于设定是否将没有约束的路径也添加到时序报告中，默认值为不选择；“No limit”和“Limit unconstrained report to”为“Report unconstrained paths”的补充参数，前者表示不限制路径数量，后者用于设定限制的路径数；“Find what”用于在“Timing Constraints”栏查找时序约束；“Timing Constraints”用于使能相关的时序约束，选中“Enabled”列的复选框，表示使能，默认值为使能全部约束。

“Options”页面如图 11-40 所示，各参数含义为：“Speed grade”用于设定芯片的速度等级，不会影响设计性能；“Timing Constraints Details”用于设定报告中的路径细节，有“Summary only”（只显示路径的终点和起点）、“No limit”（路径数不受限制）、“Limit report to”（输入限制的路径数）以及“Report fastest paths/verbose hold paths”（显示最快/详细的

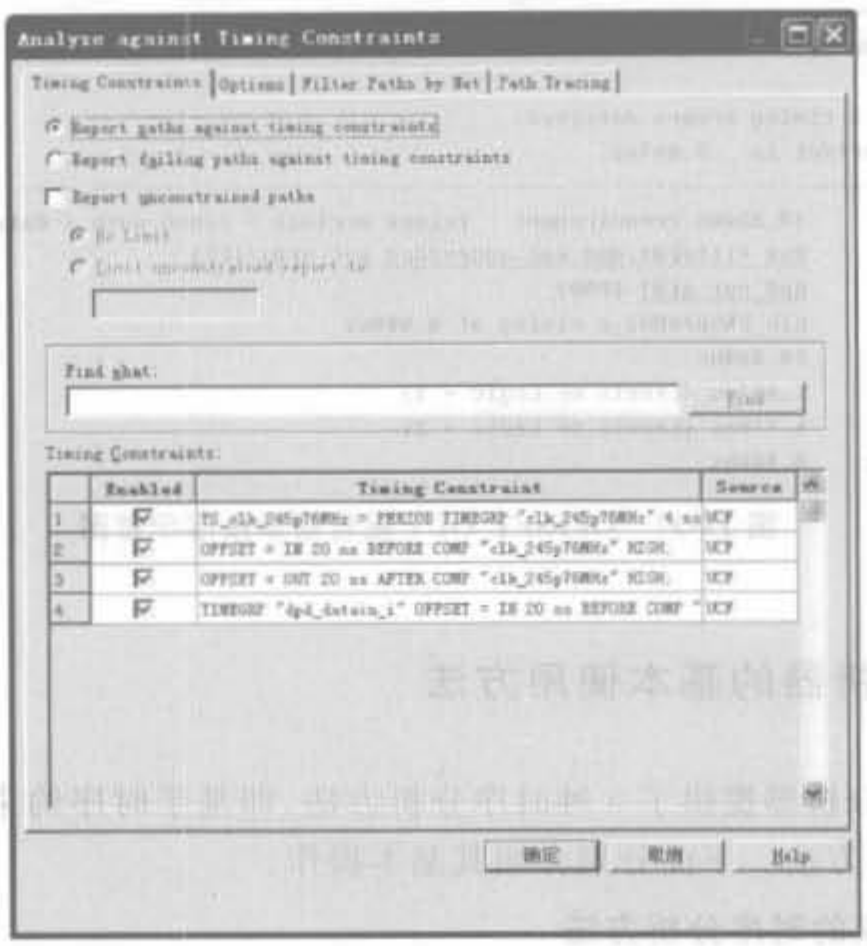


图 11-39 基于时序分析参数设置对话框

有效通道）等 4 个选项；“Timing Report Contents”用于选择报告内容，有“Generate Datasheet report”（生成数据手册报告）和“Generate Timegroup report”（生成时序分组报告）两个可选项；“Voltage Prorating”用于设定器件的工作电压，有最差情况和用户指定两种选择，默认值为最差情况；“Temperature Prorating”用于设定器件的工作温度，也有最差情况和用户指定两种选择，默认值为最差情况。

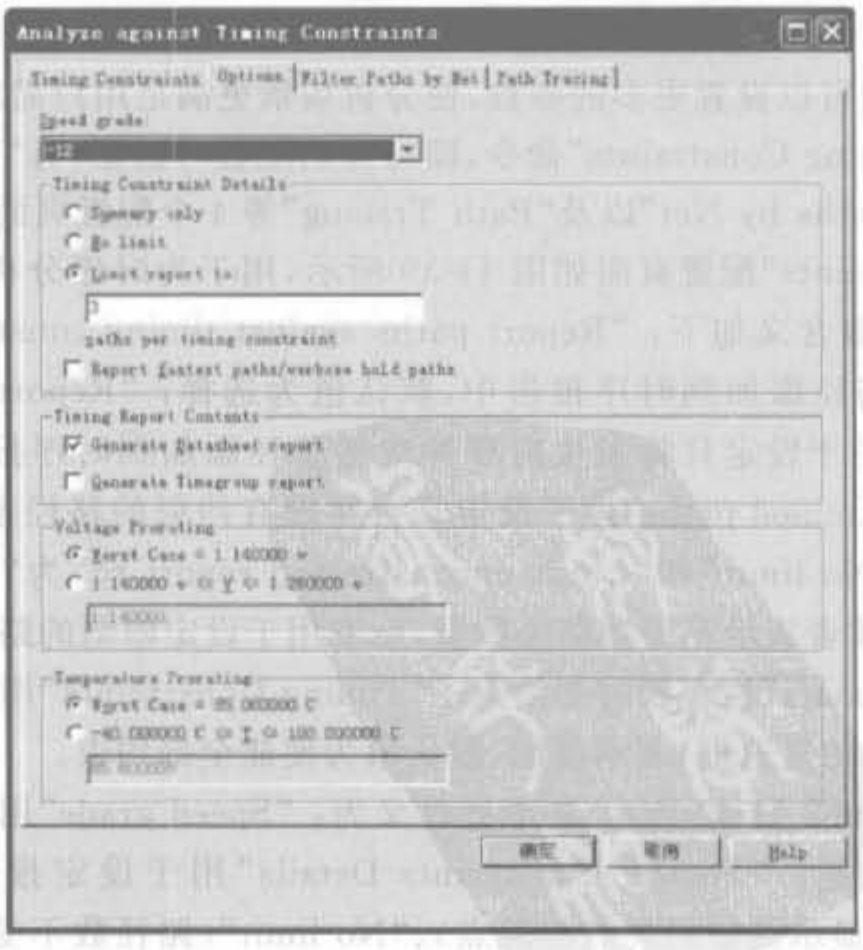


图 11-40 “Options”页面参数设置对话框

“Filter Paths by Net”页面的对话框如图 11-41 所示，主要用于挑选要进行时序分析的网线。当选中一条网线后，在“Net Filter”列单击即可出现“Include only”、“Exclude”和“Default”3 个可选项，默认值为“Default”。“Find what”用于挑选期望路径。

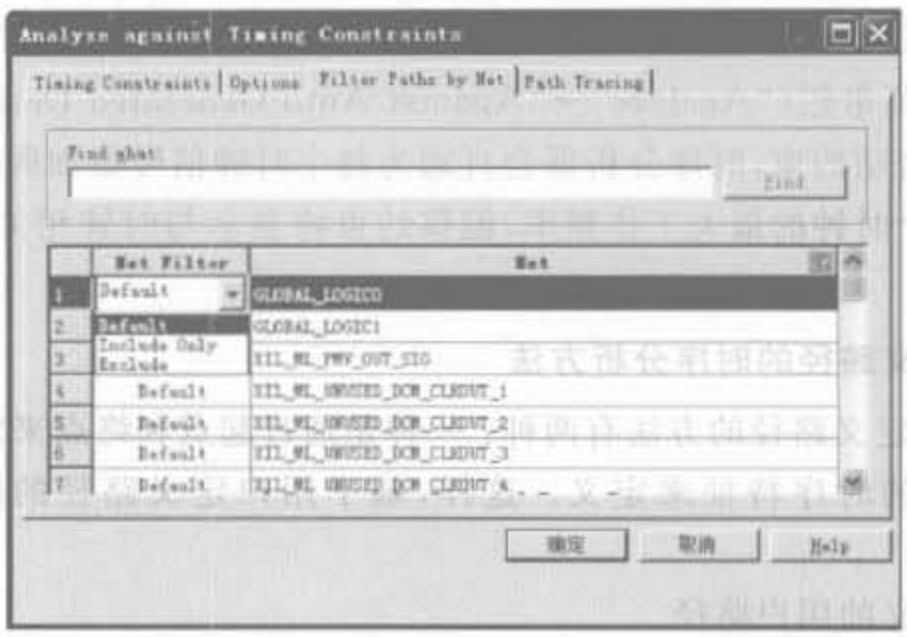


图 11-41　“Filter Paths by Net”页面参数设置对话框

“Path Tracing”页面如图 11-42 所示，主要用于配置跟踪路径的参数，各个参数含义如下：“Path types”用于选择路径类型，有“io_o_pad”(I/O 输出到 pad)、“io_pad_i”(I/O pad 到输入)以及“reg_sr_clk”(输出和恢复的同步复位与置位)3 种类型，默认值为全部选中；“Find what”用于快速查找路径；“Path comps”用于使能已选路径类型中的具体路径，默认值为全部选择。

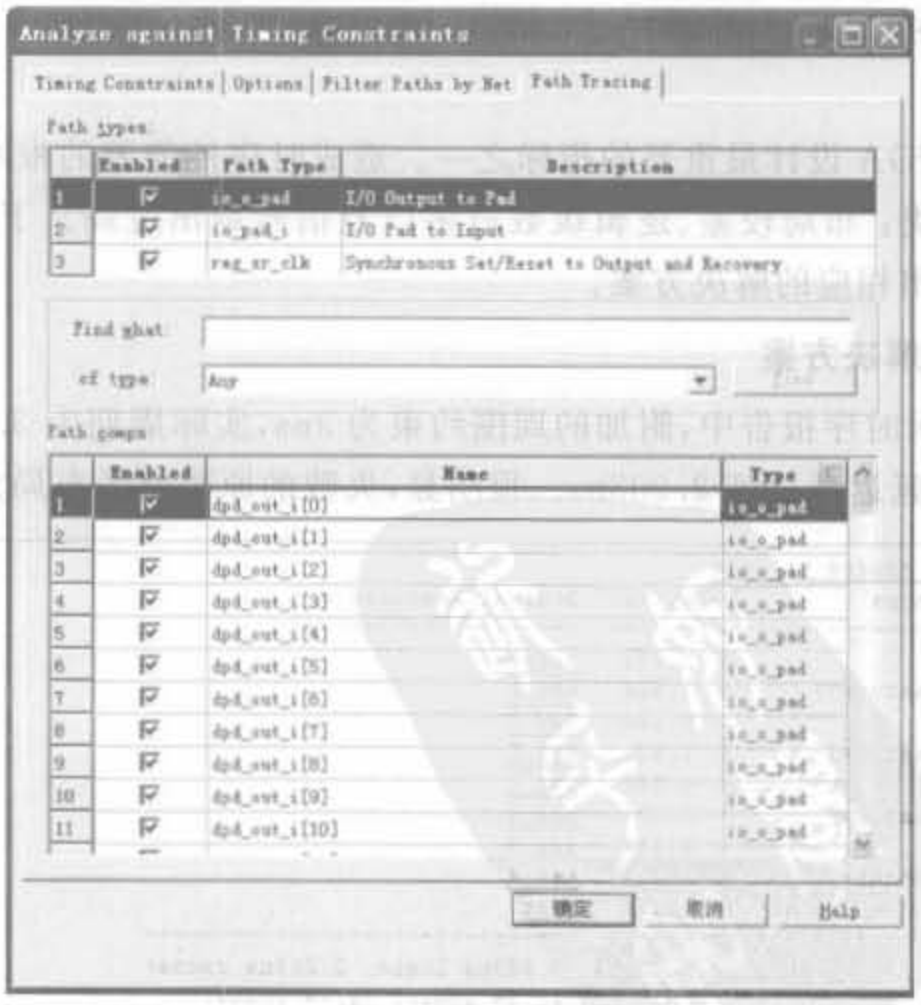


图 11-42　“Path Tracing”页面参数设置对话框

完成上述配置后，单击“确定”，即可开始时序分析，弹出的执行对话框和图 11-27 是一致的。

2. 基于自动约束的时序分析方法

基于自动约束的时序分析方法没有实际应用价值，常用来完成设计的性能评估。也有工具栏（图标）和菜单栏（“Analyze”→“Against Auto Generated Design Constraints”）两种启动方法。使用自动约束，时序分析器会自动为每个时钟信号添加周期约束和偏移约束，周期约束将显示每个时钟的最大工作频率，偏移约束将显示与时钟相关的最小建立时间和最大输出时间。

3. 基于用户定义路径的时序分析方法

一般来讲，用户定义路径的方法有两种：一种是通过起点和终点来定义，另一种就是通过时钟和 I/O 端口的时序特征来定义。这样，基于用户定义路径的时序分析方法就有两种。

1）基于端点定义的用户路径

该方法对用户指定的所有路径都有详细的时序分析，有工具栏（图标）和菜单栏（“Analyze”→“Against User Specified Paths”→“by Defining Endpoints. .”）两种启动方法。

2）基于时序特征定义的用户路径

该方法适用于整体没有时序约束，但对时钟和 I/O 时序要求严格的场合，有工具栏（图标）和菜单栏（“Analyze”→“Against User Specified Paths”→“by Defining Clock and I/O Timing. .”）两种启动方法。

11.4.5 提高时序性能的手段

时序性能是 FPGA 设计最重要的指标之一。造成时序性能差的根本原因有很多，但其直接原因可分为 3 类：布局较差、逻辑级数过多以及信号扇出过高。下面通过时序分析实例来定位原因并给出相应的解决方案。

1. 布局较差及解决方案

在图 11-43 所示时序报告中，附加的周围约束为 3ns，实际周期为 3.027ns，逻辑时间只有 0.869ns，而布线延迟竟达到 2.203ns。很明显，失败的原因就是布局太差。

```
Data Path: source to dest
    Delay type              Delay(ns)   Logical Resource(s)
    ----------------------  ----------  --------------------
    Tcko                    0.272       source
    net (fanout=7)          0.325       net 1
    Tilo                    0.146       lut 1
    net (fanout=1)          1.500       net 2
    Tilo                    0.146       lut 2
    net (fanout=1)          0.174       net 3
    Tilo                    0.146       lut 3
    net (fanout=1)          0.204       net 4
    Tas                     0.189       dest
    ----------------------  ----------  ------------------------------
    Total                   3.072ns     (0.869ns logic, 2.203ns route)
                                        (28.3% logic, 71.7% route)
```

图 11-43 布局较差的时序报告示意图

相应的解决方案有：

(1) 在 ISE 布局工具中调整布局的努力程度(effort level)。

(2) 利用布局布线工具的特别努力程度(extra effort)或 MPPR 选项。

(3) 如果用户熟悉区域约束，则利用 Floorplanner 相对区域约束(RLOC)，重新对设计进行布局规划。

2. 逻辑级数过多及解决方案

在 FPGA 设计中，逻辑级数越高，意味着资源的利用率越高，但对设计工作频率的影响也越大。在如图 11-44 所示的例子中，附加的周围约束为 3ns，实际周期为 3.205ns，逻辑时间为 1.307ns，已经对设计的实际性能造成了一定的影响。对于这种情况，ISE 实现工具是没有任何改善的，必须通过修改代码来提高性能。

相应的解决方案有：

(1) 使用流水线技术，在组合逻辑中插入寄存器，简化原有的逻辑结构。

(2) 检查该路径是否是多周期路径，如果是，添加相应的多周期约束。

(3) 具备良好的编码习惯，不要嵌套 if 语句或 if、case 语句，并且尽量用 case 语句代替 if 语句。

```
Data Path: source to dest
    Delay type              Delay(ns)  Logical Resource(s)
    ----------------------  ---------  --------------------
    Tcko                    0.272      source
    net (fanout=7)          0.621      net 1
    Tilo                    0.146      lut 1
    net (fanout=1)          0.180      net 2
    Tilo                    0.146      lut 2
    net (fanout=1)          0.223      net 3
    Tilo                    0.146      lut 3
    net (fanout=1)          0.123      net 4
    Tilo                    0.146      lut 4
    net (fanout=1)          0.310      net 5
    Tilo                    0.146      lut 5
    net (fanout=1)          0.233      net 6
    Tilo                    0.146      lut 6
    net (fanout=1)          0.309      net 7
    Tas                     0.159      dest
    ----------------------  ---------  ------------------------------------
    Total                   3.205ns    (1.307ns logic, 1.898ns route)
                                       (40.8% logic, 59.2% route)
```

图 11-44　逻辑级数过多的时序报告示意图

3. 信号扇出过高及解决方案

高扇出会造成信号传输路径过长，从而降低时序性能。如图 11-45 所示，附加的周期约束为 3ns，而实际周期为 3.927ns，其中网线的扇出已经高达 187，从而导致布线延时达到 3.003ns，占实际延时的 77.64%。这种情况是任何设计所不能容忍的。

相应的解决方案有：

(1) 通过逻辑复制的方法来降低信号的高扇出，可在 HDL 代码中手动复制，或通过在综合工具中设置达到目的。

(2) 可利用区域约束，将相关逻辑放置在一起。当然，本方法仅限于高级用户。

```
Data Path: source to dest
    Delay type              Delay(ns)   Logical Resource(s)
    -------------------------   ------------------
    Tcko                    0.272   source
    net (fanout=7)          0.125   net 1
    Tilo                    0.146   lut 1
    net (fanout=187)        2.500   net 2
    Tilo                    0.146   lut 2
    net (fanout=1)          0.174   net 3
    Tilo                    0.146   lut 3
    net (fanout=1)          0.204   net 4
    Tas                     0.159   dest
    -------------------------   ------------------------------
    Total                   3.872ns (0.869ns logic, 3.003ns route)
                                    (22.4% logic, 77.6% route)
```

图 11-45　信号扇出过高的时序报告示意图

4. Xilinx 的最优时序解决方案

FPGA 设计的时序性能是由物理器件、用户代码设计以及 EDA 软件共同决定的，忽略了任何一方面的因素，都会对时序性能有很大的影响。本节主要给出 Xilinx 物理器件和 EDA 软件的最优使用方案。

1）I/O 约束技巧

优秀的设计必须要考虑 I/O 约束的技巧。对于 Xilinx 器件来讲，进位链是垂直分布的，逻辑排列块之间也有水平方向上三态缓冲线的直接连接，且硬核单元基本都是按列分布（在水平方向就具备最短路径）。因此，最优的方案为：将用于控制信号的 I/O 置于器件的顶部或底部，且垂直布置；数据总线的 I/O 置于器件的左部和右部，且水平布置，如图 11-46 所示。

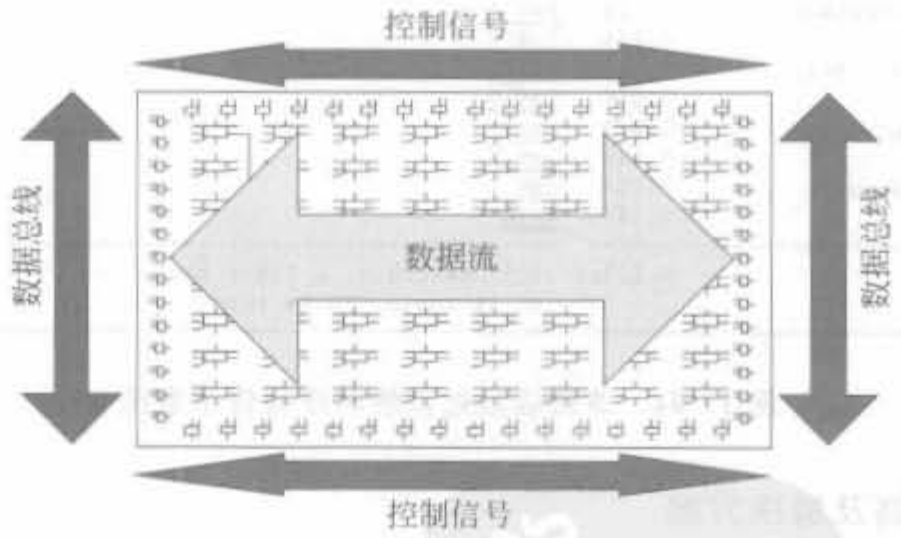


图 11-46　Xilinx 器件的最佳 I/O 布局示意图

这种 I/O 分配方式充分利用了 Xilinx FPGA 芯片架构的特点，如进位链排列方式以及块 RAM、硬核乘法器位置。进位链的结构如图 11-47 所示，能解决多位宽加法、乘法从最低位向最高位的进位延时问题；块 RAM 和硬核乘法器可节约大量的逻辑资源且保证时序，二者在 FPGA 芯片中都是自上而下成条状分布，因此数据流水平、控制流垂直，可最大限度地利用芯片底层架构。当然，在实际系统设计中可能无法完全做到上述要求，但应该尽可能地将高速率、多位宽的信号布置在芯片左、右两侧。

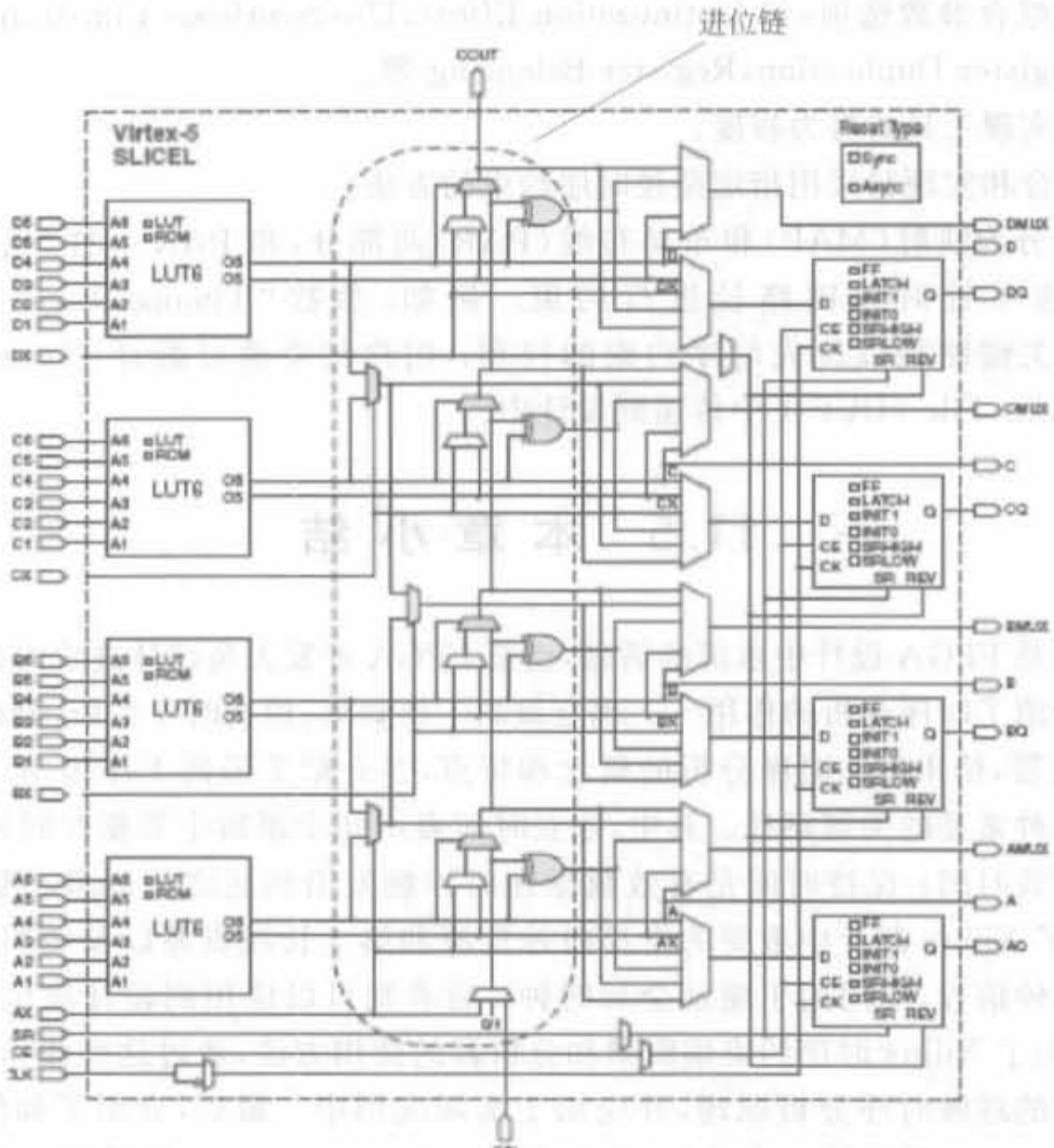


图 11-47　Xilinx 器件的进位链结构示意图

2）ISE 的实现工具

ISE 中集成的实现工具具备不同的努力程度(Effort Level)。当然，使用最高级别的可以提高时序性能，而不必采取其他措施(如施加更高级的时序约束，使用高级工具或者更改代码等)，但需要花费很长的计算时间。为此，Xilinx 推荐的最佳流程如图 11-48 所示。

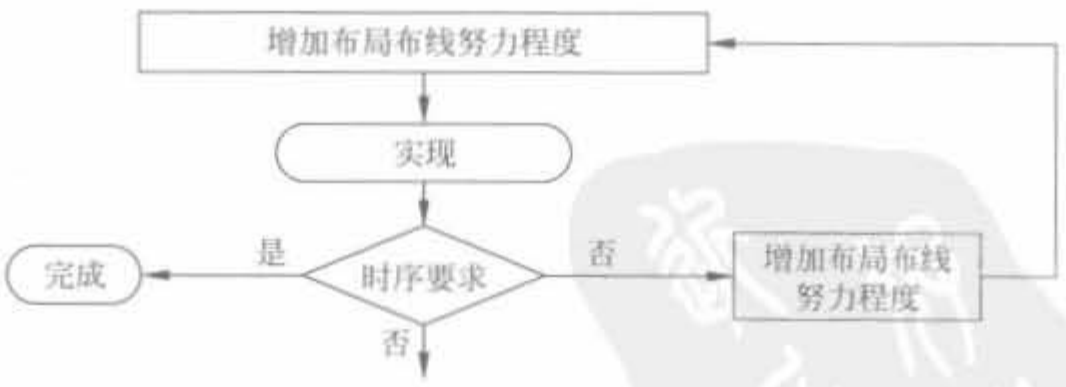


图 11-48　Xilinx 实现工具的最佳使用策略

在第一遍实现时，使用全局时序约束和默认的实现参数选项。如果不能满足时序要求，可尝试以下方法：

(1) 尝试修改代码，如使用合适的代码风格，增加流水线等。

(2) 修改综合参数选项，如 Optimization Effort，Use Synthesis Constraints File，Keep Hierarchy，Register Duplication，Register Balancing 等。

(3) 增加实现工具的努力程度。

(4) 在综合和实现时采用指定路径时序约束的方法。

实现工具分为映射(MAP)和布局布线(PAR)两部分，和 PAR 一样，也可使用 Map-timing 参数选项针对关键路径进行约束。例如，参数“Timing-Driven Packing and Placement”给关键路径以优先时序约束的权利；用户约束通过翻译(Translate)过程从 User Constraints File (UCF) 中传递到设计中。

11.5 本章小结

时序分析是 FPGA 设计中永恒的话题，也是 FPGA 开发人员设计进阶的必经之路。本章首先详细介绍了时序分析的作用——确定最高工作频率、检查时序约束、分析时钟质量以及确定 I/O 位置，给出静态时序分析的概念和特点，并介绍了最高工作频率、建立时间、保持时间以及几种常见的关键路径。其中，建立时间表示同步逻辑中数据在时钟触发沿到达之前的最小有效时间；保持时间是有效数据在时钟触发沿到达之后的最小稳定时间。其次，本章讲述了 Xilinx 芯片中重要的全局时钟资源和第二长线资源以及使用方法，前者是性能最好的时钟信号，专门用于驱动全局时钟；后者则可以应用到设计扇出大的信号上。然后，本章给出了 Xilinx 时序约束编辑器和分析器的使用方法，通过这些优秀软件的示例，便于读者更好的理解时序分析原理，并应用于实际应用中。最后，介绍了如何基于 Xilinx 的软、硬件特点，达到最优的时序性能，即在设计 I/O 分配时，高速信号尽量分布在左右两侧；在操作实现软件时，采用相关配置先默认再修改的方式。希望读者经过本章的学习，能对时序分析有一个较为系统的认识，并在实际操作中逐步加深理解。

缩 略 语

A

ACE(Advanced Configuration Environment)：先进配置环境

ASIC(Application Specific Integrated Circuits)：专用集成电路

ATC2(Agilent Trace Core 2)：安捷伦跟踪核

B

BFC(Bus Functional Compiler)：总线功能编译器

BFM(Bus Functional Model)：总线功能模型

BMM(Block Memory Map)：块存储器映射

BSB(Base System Builder)：基本系统创建器

BSP(Board Support Package)：板卡支持包

C

CAM(Content Addressable Memory)：内容地址存储器

CDR(Clock and Data Recovery)：时钟数据恢复

CIP(Create and Import Peripheral)：创建和导入外围设备向导

CLB(Configurable Logic Blocks)：可配置逻辑块

CPLD(Complex Programmable Logic Device)：复杂可编程逻辑器件

D

DCM(Digital Clock Manager)：数字时钟管理

DFS(Digital Frequency Synthesizer)：数字频率合成器

DLL(Delay Locked Loop)：延迟锁相环

DMA(Direct Memory Access)：直接内存存取

DPS(Digital Phase Shifter)：数字移相器

DRC(Design Rule Check)：设计规则检查

DSS(Digital Spread Spectrum)：数字频谱扩展器

DUT(Device Under Test)：被测试设计

E

EDA(Electronic Design Automation)：电子设计自动化

EDK(Embedded Development Kit)：嵌入式系统开发套件

EEPROM(Electrically Erasable Programmable Read-Only Memory)：电可擦除可编程只读存储器

ELF(Executable Linking Format)：执行可链接格式

EPLA(Erasable Programmable Logic Array)：可擦除的可编程逻辑阵列

F

FPGA(Field Programmable Gate Array)：现场可编程门阵列
FPLA(Field Programmable Logic Array)：现场可编程逻辑阵列
FSL(Fast Simplex Link)：快速单向链路

G

GAL(Generic Array Logic)：通用阵列逻辑
GPIO(General Purpose I/O Port)：通用输入/输出端口

H

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

I

IBA(Integrated Bus Analyzer)：综合总线分析仪
IBERT(Integrated Bit Error Ratio Tester core)：集成的误比特率测试核
IC(Integrate Circuit)：集成电路
ICON(Integrated Controller Pro core)：集成控制核
ILA(Integrated Logic Analyzer Pro cores)：综合逻辑分析仪
IOB(Input/Output Buffer)：输入/输出缓存
IP(Intellectual Property)：知识产权
IPIF(IP InterFace)：IP 接口
ISP(In-System Programmable)：在系统可编程
ISS(Cycle-Accurate Instruction Set Simulator)：周期精确指令设置仿真器

J

JTAG(Joint Test Action Group)：联合测试行动小组

L

LA(Logic Analyzer)：逻辑分析仪
LMB(Local Memory Bus)：本地存储器总线
LUT(Look Up Table)：查找表

M

MDM(Microprocessor Debug Module)：微处理器调试模块
MHS(Microprocessor Hardware Specification)：微处理器硬件规范
MMU(Memory Management Unit)：内存管理单元
MSR(Machine State Register)：机器状态寄存器
MSS(Microprocessor Software Specification)：微处理器软件规范

N

NCF(Netlist Constraint File)：网表约束文件

O

OPB(On-chip Peripheral Bus)：片上外设总线

P

PACE(Pinout and Area Constraints Editor)：管脚和区域约束编辑器
PAL(Programmable Array Logic)：可编程阵列逻辑
PCB(Printing Circuit Board)：印刷电路板
PCF(Physical Constraint File)：物理约束文件
PCS(Physical Coding Sublayer)：物理编码子层
PLD(Programmable Logic Device)：可编程逻辑器件
PLL(Phase Locked Loop)：锁相环
PROM(Programmable Read Only Memory)：编程只读存储器
PSF(Platform Specification Format)：平台规范格式

R

RPM(Relatively Placed Macros)：相对布局宏
RTL(Register Transfer Level)：寄存器传输级
Rocket I/O：Xilinx 公司的高速串行传输硬核模块

S

SDK(Software Development Kit)：软件开发套件
SOC(System On Chip)：片上系统
SOPC(System On Programmable Chip)：片上可编程系统
STA(Static Timing Analysis)：静态时序分析

U

UCF(User Constraints File)：用户约束文件

V

VIO(Virtual Input/Output)：虚拟输入/输出

X

XCL(Xilinx Cache Link)：Xilinx 缓存链路
XMD(Xilinx Microprocessor Debugger)：Xilinx 微处理调试器
XMP(Xilinx Microprocessor Project)：Xilinx 微处理器工程
XPS(Xilinx Platform Studio)：Xilinx 平台工作室

参 考 文 献

1 田耘,徐文波等. 无线通信 FPGA 设计. 北京：电子工业出版社,2008

2 张雅绮等译.(美)Michael D. Ciletti 著. Verilog HDL 高级数字设计. 北京：电子工业出版社,2005,1

3 王钿,卓兴旺. 基于 Verilog HDL 的数字系统应用设计. 北京：国防工业出版社,2007

4 J. Volder. The CORDIC computing technique. IRE Trans. Comput.,Sept. 1959

5 夏宇闻. Verilog 数字系统设计教程. 北京：北京航空航天大学出版社,2003

6 谷鋆等. FPGA 动态可重构理论及其研究进展. 计算机测量与控制,2007,15：1415～1418

7 孙航. Xilinx 可编程逻辑器件的高级应用与设计技巧. 北京：电子工业出版社,2004

8 Xilinx 公司. XAPP482- MicroBlaze Platform Flash/PROM 引导加载器和用户数据存储

9 王诚等. FPGA/CPLD 设计工具 Xilinx ISE 使用详解. 北京：人民邮电出版社,2005

10 杨浩强等. 基于 EDK 的 FPGA 嵌入式系统开发. 北京：机械工业出版社,2008

11 Xilinx 公司. MicroBlaze Processor Reference Guide

12 Xilinx 公司. PowerPC Processor Reference Guide

13 Xilinx 公司. Rocket I/O User Guide

14 周海斌. 静态时序分析在高速 FPGA 设计中的应用. 电子工程师,Nov. 2005,31(11)：41～44

15 康军等. 同步电路设计中 CLOCK SKEW 的分析. 电子器件,Dec. 2002,25(4)：134～434

16 吴继华等. Altera FPGA/CPLD 设计(高级篇). 北京：人民邮电出版社,2005

17 Xilinx 公司. UG070 Virtex-4 User Guide

18 Xilinx 公司. DS312 Spartan-3E FPGA Complete Data Sheet