

FPGA/CPLD 设计工具

——Xilinx ISE 使用详解

王 诚
EDA 先锋工作室 薛小刚 编著
钟信潮



人民邮电出版社

图书在版编目 (CIP) 数据

FPGA/CPLD 设计工具：Xilinx ISE 使用详解/王诚，薛小刚，钟信潮编著.

—北京：人民邮电出版社，2005.1

ISBN 7-115-12914-2

I. F... II. ①王... ②薛... ③钟... III. 可编程序逻辑器件—基本知识 IV. TP332.1

中国版本图书馆 CIP 数据核字 (2004) 第 127150 号

内 容 提 要

本书以 FPGA/CPLD 设计流程为主线，阐述了如何合理地利用 ISE 设计平台集成的各种设计工具，高效地完成 FPGA/CPLD 的设计方法与技巧。全书在介绍 FPGA/CPLD 概念和设计流程的基础上，依次论述了工程管理与设计输入、仿真、综合、约束、实现与布局布线、配置调试等主要设计步骤在 ISE 集成环境中的实现方法与技巧。

本书立足于工程实践，结合作者多年工作经验，选用大量典型实例，并配有一定数量的练习题。本书配套光盘收录了所有实例的完整工程目录、源代码、详细操作步骤和使用说明，利于读者边学边练，提高实际应用能力。

本书可作为高等院校通信工程、电子工程、计算机、微电子与半导体学等专业的教材，也可作为硬件工程师和 IC 工程师的实用工具书。

FPGA/CPLD 设计工具——Xilinx ISE 使用详解

◆ 编 著 EDA 先锋工作室 王 诚 薛小刚 钟信潮
责任编辑 李永涛

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号

邮编 100061 电子函件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

读者热线 010-67132692

北京鸿佳印刷厂印刷

新华书店总店北京发行所经销

◆ 开本：787×1092 1/16

印张：30.75

字数：750 千字 2005 年 1 月第 1 版

印数：1—6 000 册 2005 年 1 月北京第 1 次印刷

ISBN 7-115-12914-2/TP · 4342

定价：52.00 元（附光盘）

本书如有印装质量问题，请与本社联系 电话：(010) 67129223



EDA 先锋工作室

主 编：王 诚

副主编：薛小刚 钟信潮

编 委：李 楠 吴继华 庞 健 由武军 袁 园
周海涛 侯小辉 寿开宇 范丽珍 薛 宁
路 远 梁晓明 伊贵业 吴义涛 张世卓
张伟平 王书松 吴 蕾 胡安琪 吴卫旋
董振东 于春华

序

美国赛灵思公司 (Xilinx) 是全球领先的可编程逻辑完整解决方案的供应商，赛灵思研发、制造并销售应用范围广泛的高级集成电路、软件设计工具以及作为预定义系统级功能的 IP (Intellectual Property) 核。

赛灵思公司成立于 1984 年，首创了现场可编程逻辑阵列 (FPGA) 这一创新性的技术，并于 1985 年首次推出商业化产品。至今，赛灵思公司已经推出从 Spartan3 到 Virtex-II Pro 等系列的 FPGA 产品以及 CoolRunner 等系列的 CPLD 产品，满足了全世界对可编程逻辑器件 (PLD) 产品一半以上的需求，同时还提供了互联网软件解决方案与核心解决方案等软件产品。ISE 是赛灵思公司最主要的设计软件之一，目前已经推出的最新版本是 ISE 6.2i。该软件提供了可加强现有可编程设计流程，并可适应客户特有设计方法的全面、丰富的设计选项。

在此，我们郑重推荐由人民邮电出版社出版的《FPGA/CPLD 设计工具——Xilinx ISE 使用详解》，希望通过本书使对在设计中应用 ISE 解决方案或对其感兴趣的读者能更深入地了解基于 ISE 进行 FPGA/CPLD 设计的基本原理与方法。也希望通过本书的编写，能大力促进中国的数字系统现场集成技术的应用普及和推广。

XILINX ASIA PACIFIC

2004 年 10 月

目 录

第 1 章 ISE 系统简介	1
1.1 FPGA/CPLD 简介.....	1
1.1.1 FPGA/CPLD 的基本原理.....	2
1.1.2 FPGA 和 CPLD 的特点.....	7
1.2 FPGA/CPLD 的设计流程.....	9
1.3 ISE 系列产品的特点.....	11
1.3.1 特点综述.....	11
1.3.2 ISE 的新增特性.....	12
1.4 ISE 6.x 支持的器件.....	14
1.5 ISE 的 4 个软件系列.....	14
1.6 ISE 的系统配置与安装.....	18
1.6.1 推荐的系统配置.....	18
1.6.2 ISE 的安装.....	19
1.7 ISE 中集成工具及其基本功能.....	21
1.8 常用专有名词解释.....	27
1.9 小结.....	29
1.10 问题与思考.....	29
第 2 章 工程管理器与设计输入工具	31
2.1 ISE 工程管理器——Project Navigator.....	31
2.1.1 Project Navigator 综述.....	31
2.1.2 Project Navigator 的用户界面.....	32
2.1.3 使用 Project Navigator 创建并管理工程.....	38
2.2 HDL 语言的输入工具——HDL Editor.....	43
2.2.1 HDL Editor 综述.....	43
2.2.2 源代码输入的好助手——Language Templates.....	44
2.3 状态机输入工具——StateCAD.....	45
2.3.1 StateCAD 综述.....	46
2.3.2 StateCAD 的用户界面.....	46
2.3.3 使用 StateCAD 设计状态机.....	51
2.4 原理图输入工具——ECS.....	62
2.4.1 ECS 综述.....	63
2.4.2 ECS 的用户界面.....	63
2.4.3 使用 ECS 完成原理图输入设计.....	66
2.4.4 使用 ECS 进行混合设计的方法.....	73
2.5 IP 核生成工具——CORE Generator.....	74

2.5.1	CORE Generator 综述	74
2.5.2	CORE Generator 的用户界面	75
2.5.3	使用 CORE Generator 生成 IP 核的方法与技巧	78
2.6	测试激励生成器——HDL Bencher	83
2.6.1	HDL Bencher 综述	83
2.6.2	使用 HDL Bencher 生成测试激励	85
2.7	设计结构向导——Architecture Wizard	91
2.7.1	Architecture Wizard 综述	91
2.7.2	Architecture Wizard 使用方法	91
2.8	小结	94
2.9	问题与思考	94
第 3 章	仿真工具	95
3.1	ModelSim 的用户接口	97
3.1.1	行为仿真	99
3.1.2	时序仿真	101
3.1.3	高级设置	102
3.2	ModelSim 仿真窗口综述	104
3.3	仿真环境的建立	114
3.3.1	各仿真切入点需要的库文件	114
3.3.2	仿真库文件说明	115
3.3.3	库文件编译	116
3.3.4	仿真库的命名	120
3.4	一个简单的仿真示例	121
3.4.1	在 ModelSim 环境下进行仿真	121
3.4.2	在 ISE 集成环境中进行仿真	128
3.5	ModelSim 中的调试方法	130
3.5.1	源文件窗口调试	130
3.5.2	波形窗口调试	132
3.5.3	数据流窗口调试	135
3.5.4	存储器窗口调试	140
3.5.5	变量窗口调试	142
3.5.6	列表窗口调试	145
3.6	ModelSim 的其他常用操作	146
3.6.1	自动仿真	146
3.6.2	WLF 文件	149
3.6.3	波形比较	151
3.6.4	SDF 文件	156
3.6.5	VCD 文件	157

3.7 小结	159
3.8 问题与思考	159
第4章 ISE 中集成的综合工具	161
4.1 新兴的高效综合工具——Synplify/Synplify Pro	161
4.1.1 Synplify/Synplify Pro 的功能与特点	161
4.1.2 Synplify Pro 的用户界面	168
4.1.3 Synplify Pro 综合流程	171
4.1.4 Synplify Pro 的其他综合技巧	193
4.2 Xilinx 最早的合作伙伴——Synopsys 综合工具	205
4.2.1 设计流程	206
4.2.2 FE 综合优化过程	208
4.2.3 FST 操作说明	217
4.3 Xilinx 内嵌的综合工具——XST	220
4.3.1 XST 综述	220
4.3.2 XST 综合属性设置	221
4.3.3 使用 XST 的综合流程	226
4.4 全局时钟与第二全局时钟资源	229
4.4.1 全局时钟资源简介	229
4.4.2 常用的与全局时钟资源相关的 Xilinx 器件原语	230
4.4.3 Xilinx 全局时钟资源的使用方法	232
4.4.4 使用 Xilinx 全局时钟资源的注意事项	233
4.4.5 第二全局时钟资源	235
4.5 小结	236
4.6 问题与思考	236
第5章 约束	237
5.1 概述	237
5.2 时序约束	239
5.2.1 周期约束 (PERIOD 约束)	239
5.2.2 偏移约束 (OFFSET 约束)	242
5.2.3 专门约束	245
5.3 分组合约	248
5.3.1 TNM 约束	248
5.3.2 TNM_NET 约束	251
5.3.3 TIMEGRP 约束	251
5.3.4 TPTHU 约束	252
5.3.5 TPSYNC 约束	252
5.4 约束编辑器——Constraints Editor	253
5.4.1 Constraints Editor 的用户界面	253

5.4.2	附加全局约束.....	254
5.4.3	附加端口约束.....	256
5.4.4	附加分组约束和时序约束.....	257
5.4.5	附加专用约束.....	261
5.5	引脚与区域约束编辑器——PACE.....	262
5.5.1	PACE 的用户界面.....	263
5.5.2	附加区域约束.....	266
5.5.3	附加 I/O 引脚约束.....	267
5.6	约束文件.....	268
5.6.1	约束文件的概念.....	268
5.6.2	UCF、NCF 文件的基本语法规则.....	269
5.7	小结.....	271
5.8	问题与思考.....	271
第 6 章	辅助设计工具.....	273
6.1	时序分析器——Timing Analyzer.....	273
6.1.1	时序分析器的用户界面.....	274
6.1.2	时序分析器的作用及设计流程.....	275
6.1.3	基本时序路径.....	276
6.1.4	时序分析器的使用方法.....	282
6.2	布局规划器——Floorplanner.....	286
6.2.1	布局规划器的用户界面.....	286
6.2.2	布局规划器的特点及作用.....	288
6.2.3	布局规划设计流程.....	289
6.2.4	设计示例.....	292
6.3	FPGA 底层编辑器——FPGA Editor.....	297
6.3.1	FPGA 底层编辑器的用户接口.....	297
6.3.2	FPGA 底层编辑器的作用.....	298
6.3.3	FPGA 底层编辑器输入输出文件.....	300
6.3.4	FPGA 底层编辑器的工作流程.....	301
6.3.5	使用 FPGA 底层编辑器的预备知识.....	301
6.3.6	设计示例.....	303
6.4	小结.....	308
6.5	问题与思考.....	308
第 7 章	XPower、iMPACT 和 ChipScope Pro.....	309
7.1	XPower.....	309
7.1.1	XPower 综述.....	309
7.1.2	XPower 的用户界面.....	310
7.1.3	用 XPower 分析功耗.....	313

7.2	iMPACT	316
7.2.1	iMPACT 综述	316
7.2.2	iMPACT 的用户界面	317
7.2.3	用 iMPACT 下载配置文件	320
7.3	ChipScope Pro	328
7.3.1	ChipScope Pro 综述	328
7.3.2	ChipScope Pro Core Inserter	330
7.3.3	ChipScope Pro Analyzer	336
7.4	小结	341
7.5	问题与思考	341
第 8 章	模块化与增量式设计方法	343
8.1	模块化设计方法的基本概念	343
8.2	模块化设计方法的设计流程	344
8.2.1	Modular Design 的设计输入与综合步骤	345
8.2.2	Modular Design 的实现步骤	346
8.3	模块化设计方法的注意事项	350
8.3.1	Modular Design 的应用场合	350
8.3.2	Modular Design 的设计效能	351
8.3.3	Modular Design 的目录管理	352
8.3.4	Modular Design 的常用约束	352
8.3.5	Modular Design 的报告查看	353
8.3.6	使用 XFLOW 自动进行模块化设计	353
8.4	模块化设计方法的设计实例	355
8.5	增量式设计方法的基本概念	364
8.6	增量设计方法的设计流程	366
8.6.1	增量综合	367
8.6.2	增量实现	369
8.7	增量设计方法的设计实例	372
8.8	小结	379
8.9	问题与思考	381
第 9 章	融会贯通——“运动计时表”设计	383
9.1	示例背景	384
9.2	多元混合设计输入方法	385
9.2.1	新建工程“watch_sc”	385
9.2.2	使用 ECS 绘制“cnt60”和“outs3”模块原理图	386
9.2.3	使用 Core Generator 生成“tenths”IP 核	392
9.2.4	使用 StateCAD 设计“stmach_v”状态机	395
9.2.5	使用 Architecture Wizard 生成时钟管理模块“dcm1”	403

9.2.6	使用语言模板设计“hex2led”和“decode”的HDL源代码	405
9.2.7	使用ECS设计顶层原理图	408
9.3	测试激励与行为级功能仿真	409
9.3.1	使用HDL Benchner生成测试激励	410
9.3.2	调用ModelSim进行行为级功能仿真	412
9.4	Synplify Pro和XST综合方法	413
9.4.1	使用XST综合整个设计	413
9.4.2	使用Synplify Pro的特色工具分析、优化设计	415
9.5	设计用户约束文件与实现结果的分析	421
9.5.1	使用Constraints Editor设计UCF文件	422
9.5.2	使用PACE设计UCF	425
9.5.3	实现步骤与实现结果分析	427
9.6	使用ModelSim进行布线后仿真	432
9.7	使用iMPACT配置FPGA/CPLD	433
9.8	小结	438
9.9	问题与思考	438
第10章	ISE 实战——I²C 接口设计	439
10.1	EFX-SP200——实验开发系统简介	439
10.2	I ² C 总线简介	440
10.2.1	I ² C 总线上的数据传输	441
10.2.2	I ² C 总线寻址	443
10.2.3	时钟同步与仲裁	445
10.2.4	I ² C 协议的扩展	446
10.3	I ² C 总线应用实例——AT24C 系列EEPROM	446
10.3.1	AT24C02 概述	447
10.3.2	写操作	448
10.3.3	读操作	448
10.3.4	AT24C 在IC卡中的应用简介	449
10.4	I ² C 总线控制器设计详解	450
10.4.1	I ² C 总线控制器总体描述	451
10.4.2	μC 接口设计	452
10.4.3	I ² C 接口设计	458
10.4.4	混合仿真验证	469
10.4.5	上板调试	478
10.5	小结	478
10.6	问题与思考	478

第1章 ISE 系统简介

本章在介绍 FPGA/CPLD 基本理论的基础上，引入 FPGA/CPLD 的完整设计流程，重点介绍 Xilinx 的 ISE 集成开发环境的特点与 ISE 所集成的工具包的基本功能。

本章主要内容如下：

- FPGA/CPLD 简介；
- FPGA/CPLD 的设计流程；
- ISE 系列产品的新特点；
- ISE 6.x 支持的器件；
- ISE 的 4 个软件系列；
- ISE 的系统配置与安装；
- ISE 中集成的工具及其基本功能；
- 常用专有名词解释。

1.1 FPGA/CPLD 简介

在数字化、信息化的时代，数字集成电路应用得非常广泛。随着微电子技术与工艺的发展，数字集成电路从电子管、晶体管、中小规模集成电路、超大规模集成电路（VLSIC）逐步发展到今天的专用集成电路（ASIC）。ASIC 的出现降低了产品的生产成本，提高了系统的可靠性，减少了产品的物理尺寸，推动了社会的数字化进程。但是 ASIC 因其设计周期长，改版投资大，灵活性差等缺陷制约着它的应用范围。硬件工程师希望有一种更灵活的设计方法，根据需要，在实验室就能设计、更改大规模数字逻辑，研制自己的 ASIC 并马上投入使用。这就是可编程逻辑器件提出的基本思想。

可编程逻辑器件随着微电子制造工艺的发展取得了长足的进步。从早期的只能存储少量数据，完成简单逻辑功能的可编程只读存储器（PROM）、紫外线可擦除只读存储器（EPROM）和电可擦除只读存储器（EEPROM），发展到能完成中大规模的数字逻辑功能的可编程阵列逻辑（PAL）和通用阵列逻辑（GAL），今天已经发展成为可以完成超大规模的复杂组合逻辑与时序逻辑的现场可编程逻辑器件（FPGA）和复杂可编程逻辑器件（CPLD）。随着工艺技术的发展与市场需求，超大规模、高速、低功耗的新型 FPGA/CPLD 不断推陈出新。新一代的 FPGA 甚至集成了中央处理器（CPU）或数字处理器（DSP）内核，在一片 FPGA 上进行软硬件协同设计，为实现片上可编程系统（SOPC：System On Programmable Chip）提供了强大的硬件支持。

1.1.1.1 FPGA/CPLD 的基本原理

简化的 FPGA 基本由 6 部分组成：可编程输入/输出单元、基本可编程逻辑单元、嵌入式块 RAM、丰富的布线资源、底层嵌入功能单元和内嵌专用硬核，如图 1-1 所示。

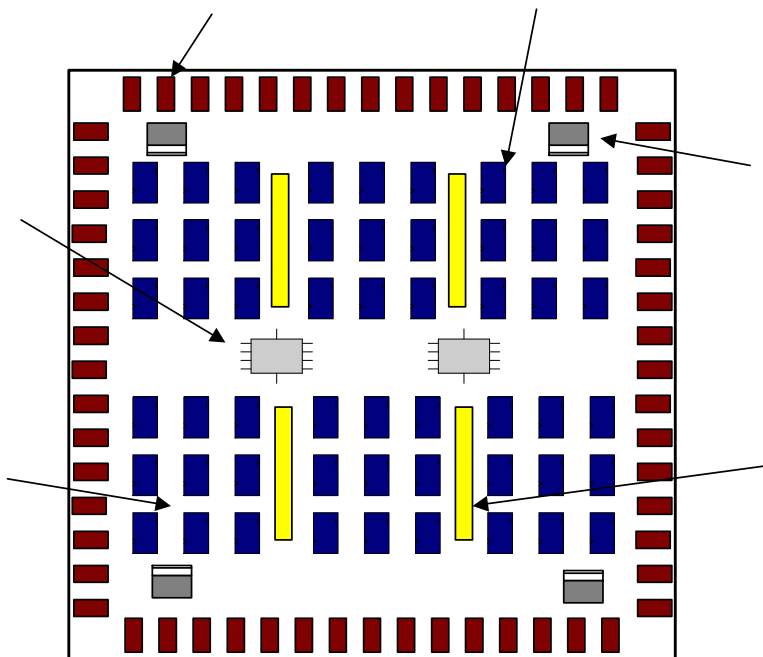


图1-1 可编程逻辑器件的结构原理图

(1) 可编程输入/输出单元

输入/输出（Input/Output）单元简称 I/O 单元，它们是芯片与外界电路的接口部分，完成不同电气特性下对输入/输出信号的驱动与匹配需求。为了使 FPGA 有更灵活的应用，目前大多数 FPGA 的 I/O 单元被设计为可编程模式，即通过软件的灵活设置，可以匹配不同的电气标准与 I/O 物理特性；可以调整匹配阻抗特性，上下拉电阻；可以调整输出驱动电流的大小等。

可编程 I/O 单元支持的电气标准因工艺而异，不同器件商或不同器件族的 FPGA 支持的 I/O 标准也不同，一般说来，常见的电气标准有 LVTTTL、LVCMOS、SSTL、HSTL、LVDS、LVPECL 和 PCI 等。值得一提的是，随着 ASIC 工艺的飞速发展，目前可编程 I/O 支持的最高频率越来越高，一些高端 FPGA 通过 DDR 寄存器存取技术，甚至可以支持高达 2GHz 的频率。

(2) 基本可编程逻辑单元

基本可编程逻辑单元是可编程逻辑的主体，可以根据设计灵活地改变其内部连接与配置，完成不同的逻辑功能。FPGA 一般是基于 SRAM 工艺的，其基本可编程逻辑单元通常是由查找表（LUT，Look Up Table）和寄存器

(Register) 组成的。FPGA 内部查找表一般为 4 输入（注：Altera Stratix II 的自适应逻辑模块 ALM 结构比较特殊），查找表一般完成纯组合逻辑功能。FPGA 内部寄存器结构相当灵活，可以配置为带同步/异步复位或置位、时钟使能的触发器（FF, Flip Flop），也可以配置成为锁存器（Latch）。FPGA 中一般依赖寄存器完成同步时序逻辑设计。比较经典的基本可编程单元配置为一个寄存器加一个查找表。但是不同厂商的寄存器和查找表的内部结构有一定的差异，而且寄存器和查找表的组合模式也不同。例如，Xilinx 可编程逻辑单元叫做 Slice，它由上下两部分构成，每部分都由一个 Register 加一个 LUT 组成，被称为 LC (Logic Cell, 逻辑单元)，两个 LC 之间有一些共用逻辑，可以完成 LC 之间的配合工作与级连。Altera 可编程逻辑单元被称为 LE (Logic Element, 逻辑单元)，由一个 Register 加一个 LUT 构成。Lattice 的底层逻辑单元叫做 PFU (Programmable Function Unit, 可编程功能单元)，它由 8 个 LUT 和 9 个 Register 构成。当然这些可编程单元的配置结构随着器件的发展也在不断更新，最新的一些可编程逻辑器件常常根据设计需求推出一些新的 LUT 和 Register 的配置比率，并更新其内部的连接构造。

学习底层配置单元的 LUT 和 Register 比率的一个重要意义在于器件选型和规模估算。很多器件手册上用器件的 ASIC 门数或等效的系统门数表示器件的规模。但是由于目前 FPGA 内部除了基本可编程逻辑单元外，还含有丰富的嵌入式 RAM、PLL 或 DLL，专用 Hard IP Core（硬知识产权功能核）等。这些功能模块也会等效出相当规模的系统门，所以用系统门考核基本可编程逻辑单元的数量是不准确的。比较科学的方法是用器件的 Register 或 LUT 的数量衡量（因为一般来说两者比率为 1:1）。例如，Xilinx 的 Spartan-III 系列的 XC3S1000 有 15360 个 LUT，而 Lattice 的 EC 系列 LFEC15E 也有 15360 个 LUT，所以这两款 FPGA 的可编程逻辑单元数量基本相当，属于同一规模的产品。同样道理，Altera 的 Cyclone 器件族的 EP1C12 的 LUT 数量是 12060 个，就比前面提到的两款 FPGA 规模略小。需要说明的是，器件选型是一个综合性问题，需要将设计的需求、成本压力、规模、速度等级、时钟资源、I/O 特性、封装和专用功能模块等诸多因素综合考虑。

(3) 嵌入式块 RAM

目前大多数 FPGA 都有内嵌的块 RAM (Block RAM)。FPGA 内部嵌入可编程 RAM 模块，大大地拓展了 FPGA 的应用范围和使用灵活性。FPGA 内嵌的块 RAM 一般可以灵活地配置为单口 RAM (SPRAM, Single Port RAM)、双口 RAM (DPRAM, Double Ports RAM)、伪双口 RAM (Pseudo DPRAM)、CAM (Content Addressable Memory) 和 FIFO (First In First Out) 等常用存储结构。RAM 的概念和功能读者应该非常熟悉，不再赘述。FPGA 中其实并没有专用的 ROM 硬件资源，实现 ROM 的思路是对 RAM 赋予初值，并保持该初值。所谓 CAM，即内容地址存储器。CAM 这种存储器在其每个存储单元都包含了一个内嵌的比较逻辑，写入 CAM 的数据会和其内部存储的每一个数据进行比较，并返回与端口数据相同的所有内部数据的地址。概括地讲，RAM

是一种根据地址读、写数据的存储单元；而 CAM 和 RAM 恰恰相反，它返回的是与端口数据相匹配的内部地址。CAM 的应用也非常广泛，比如在路由器中的地址交换表等。FIFO 即先进先出存储队列。FPGA 内部实现 RAM、ROM、CAM 和 FIFO 等存储结构都可以基于嵌入式块 RAM 单元，并根据需求生成相应的粘合逻辑（Glue Logic）以完成地址和片选等控制逻辑。

不同器件商或不同器件族的内嵌块 RAM 的结构不同，Xilinx 常见的块 RAM 大小是 4Kbit 和 18Kbit 两种结构，Lattice 常用的块 RAM 大小是 9Kbit，Altera 的块 RAM 最为灵活，一些高端器件内部同时含有 3 种块 RAM 结构，分别是 M512 RAM (512bit)，M4K RAM (4Kbit) 和 M-RAM (512Kbit)。

需要补充一点的是：除了块 RAM 以外，Xilinx 和 Lattice FPGA 还可以灵活的将 LUT 配置成 RAM、ROM、FIFO 等存储结构，这种技术被称为分布式 RAM (Distributed RAM)。分布式 RAM 适用于多块小容量 RAM 的设计。

根据设计需求，块 RAM 的数量和配置方式也是器件选型的一个重要标准。

(4) 丰富的布线资源

布线资源连通 FPGA 内部所有单元，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 内部有着非常丰富的布线资源，这些布线资源根据工艺、长度、宽度和分布位置的不同而划分为不同的等级，有一些是全局性的专用布线资源，用以完成器件内部的全局时钟和全局复位/置位的布线；一些叫做长线资源，用以完成器件 Bank 间的一些高速信号和一些第二全局时钟信号的布线；还要一些叫做短线资源，用以完成基本逻辑单元之间的逻辑互联与布线；另外在基本逻辑单元内部还有着各式各样的布线资源和专用时钟、复位等控制信号线。

设计者通常不需要直接选择布线资源，实现过程中一般是由布局布线器自动根据输入的逻辑网表的拓扑结构和约束条件选择可用的布线资源连通所用的底层单元模块，所以设计者通常忽略布线资源。其实布线资源的使用和设计的实现结果有直接关系。例如本书第 4 章第 3 节就介绍了全局时钟资源和第二全局时钟资源的一些使用事项。在本书的第 5 章中，很多时序约束属性就是通过调整布线资源以使设计的布局布线结果达到所需的时序性能。

(5) 底层嵌入功能单元

这个概念比较笼统，这里我们指的是那些通用程度较高的嵌入式功能模块。比如 PLL (Phase Locked Loop)、DLL (Delay Locked Loop)、DSP 和 CPU 等。随着 FPGA 的发展，这些模块被越来越多地嵌入到 FPGA 的内部，以满足不同场合的需求。

目前大多数 FPGA 厂商都在 FPGA 内部集成了硬的 DLL (Delay-Locked Loop) 或者 PLL (Phase-Locked Loop)，用以完成时钟的高精度、低抖动的倍频、分频、占空比调整、移相等功能。目前高端 FPGA 产品集成的 DLL 和 PLL 资源越来越丰富，功能越来越复杂，精度越来越高（一般在 100 ps 的数量级）。Xilinx 芯片主要集成的是 DLL，Altera 芯片集成的是 PLL，Lattice 的新

型 FPGA 同时集成了 PLL 与 DLL 以适应不同的需求。Xilinx 芯片 DLL 的模块名称为 CLKDLL，在高端 FPGA 中，CLKDLL 的增强型模块为 DCM (Digital Clock Manager，数字时钟管理模块)。Altera 芯片的 PLL 模块也分为增强型 PLL (Enhanced PLL) 和高速 PLL (Fast PLL) 等。这些时钟模块的生成和配置方法一般分为两种，一种是在 HDL 代码和原理图中直接实例化，另一种方法是在 IP 核生成器中配置相关参数，自动生成 IP。Xilinx 的 IP 核生成器叫做 Core Generator，另外在 Xilinx ISE 6.x 版本中通过 Architecture Wizard 生成 DCM 模块。Altera 的 IP 核生成器叫做 Mega Wizard。Lattice 的 IP 核生成器被称为 Module/IP Manager。另外可以通过在综合、实现步骤的约束文件中编写约束属性完成时钟模块的约束。

越来越多的高端 FPGA 产品将包含 DSP 或 CPU 等软处理核，从而 FPGA 将由传统的硬件设计手段逐步过渡为系统级设计工具。例如 Xilinx 的 Virtex II 和 Virtex II Pro 系列 FPGA 内部集成了 Power PC 450 的 CPU Core 和 MicroBlaze RISC 处理器 Core；而 Altera 的 Stratix、Stratix GX 和 Stratix II 等器件族内部集成了 DSP Core；Lattice 的 ECP 系列 FPGA 内部集成了系统 DSP Core 模块。这些 CPU 或 DSP 处理模块的硬件主要由一些加、乘、快速进位链、Pipelining 和 Mux 等结构组成，加上用逻辑资源和块 RAM 实现的软核部分，就组成了功能强大的软计算中心。这种 CPU 或 DSP 比较适合实现 FIR 滤波器、编码解码和 FFT (快速傅立叶变换) 等运算。FPGA 内部嵌入 CPU 或 DSP 等处理器，使 FPGA 在一定程度上具备了实现软硬件联合系统的能力，FPGA 正逐步称为 SOC (System On Chip) 的高效设计平台。Xilinx 的 SOC 设计工具是 EDK 和 Platform Studio，在 ISE 6.x 中已经集成了 EDK 和 MontaVista Linux、Wind River VxWorks、QNX Neutrino 等常用设计软件的接口。

(6) 内嵌专用硬核

内嵌专用硬核是相对于前文所述“底层嵌入单元”而言的，它主要指那些通用性相对较差，不为大多数 FPGA 器件所包含的硬核 (Hard Core)。我们称 FPGA 和 CPLD 为通用逻辑器件，是区分于专用集成电路 (ASIC) 而言的。其实 FPGA 内部也有两个阵营：一方面是通用性较强，目标市场范围很广，价格适中的 FPGA；另一方面是针对性较强，目标市场明确，价格较高的 FPGA。前者主要指低成本 (Low Cost) FPGA，后者主要指某些高端通信市场的可编程逻辑器件。为了提高 FPGA 性能，适用高速通信总线与接口标准，很多高端 FPGA 集成了 SERDES (串并收发器) 等专用 Hard Core。例如 Xilinx 的 Virtex II Pro 内部集成了 3.125G SERDES，支持 Rocket IO 标准；Altera 的对应器件族为 Stratix GX；Lattice 器件的专用 Hard Core 的比重更大，有两类器件族支持 SERDES 功能，分别是 Lattice 高端 SC 系列 FPGA 和现场可编程系统芯片 (FPSC, Field Programmable System Chip)。需要补充的是目前 Xilinx 和 Lattice 都已经推出内嵌 10 Gbps SERDES 模块的系统级可编程逻辑器件。

CPLD 在工艺和结构上都与 FPGA 有一定的区别，如前面介绍，FPGA 一般都是 SRAM 工艺的，如 Xilinx、Altera 和 Lattice 的系列 FPGA 器件，其基本结构都是基于查找表加寄存

器结构的。而 CPLD，一般都是基于乘积项结构的，如 Altera 的 MAX7000、MAX3000 (E²PROM 工艺) 系列器件，Lattice 的 ispMACH4000、ispMACH5000 (0.18μm E²COMS 工艺) 系列器件，Xilinx 的 XC9500 (0.35μmCMOS Fast Flash 工艺)、CoolRunner2 (0.18μmCMOS 工艺) 系列器件等都是基于乘积项的 CPLD。

CPLD 的结构相对比较简单，主要由可编程 I/O 单元、基本逻辑单元、布线池和其他辅助功能模块构成，如图 1-2 所示。

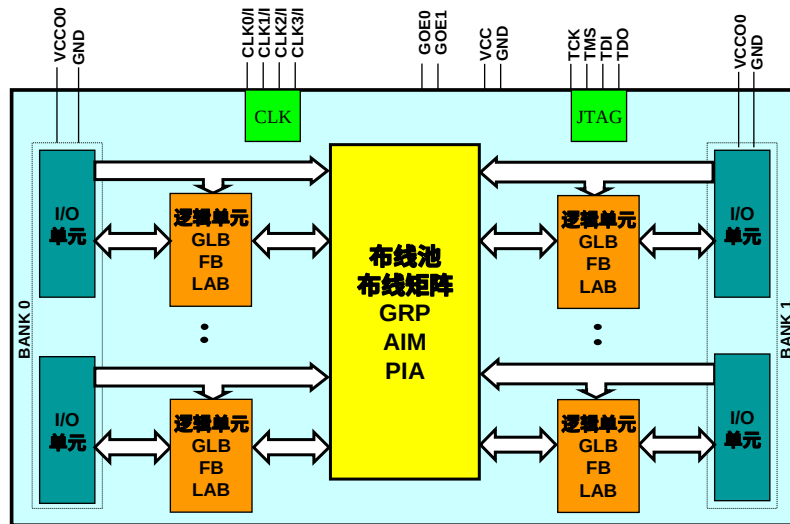


图1-2 CPLD 的结构示意图

(7) 可编程 I/O 单元

CPLD 的可编程 I/O 单元和 FPGA 的可编程 I/O 单元的功能一致，同样完成不同电气特性下对输入/输出信号的驱动与匹配。由于 CPLD 的应用范围局限性较大，所以其可编程 I/O 的性能和复杂度与 FPGA 相比有相当的差距。CPLD 的可编程 I/O 支持的 I/O 电气标准较少，频率也较低。

(8) 基本逻辑单元

与 FPGA 相似，基本逻辑单元是 CPLD 的主体，通过不同的配置方式，CPLD 的基本逻辑单元可以完成不同类型的逻辑功能。需要强调的是，CPLD 的基本逻辑单元的结构与 FPGA 相差较大。前面介绍过，FPGA 的基本逻辑单元基本是由 LUT 和 Register 按照 1:1 的比例组成的；而 CPLD 中没有 LUT 这种概念，其基本逻辑单元是一种被称为宏单元 (Macro Cell，简称 MC) 的结构。所谓宏单元，其本质是由一些与、或阵列加触发器构成，其中与、或阵列完成组合逻辑功能，触发器完成时序逻辑。CPLD 器件规模一般用 MC 的数目表示，器件标称中的数字一般都包含有该器件的 MC 数量。CPLD 厂商通过将若干个 MC 连接起来完成相对复杂一些的逻辑功能。不同厂商的这种 MC 集合的名称不同，Xilinx 9500 和 CoolRunner2 将之称为功能模块 (FB, Function Block)；Lattice 的 LC4000、ispLSI5000、ispLSI2000 系列 CPLD 将之称为通用逻辑模块 (GLB, Generic Logic Block)；Altera 的 MAX7000、MAX3000 系列 EPLD 将之称为逻辑阵列模块 (LAB, Logic Array Block)，其功能一致，但结

构略有不同。

与 CPLD 基本逻辑单元相关的另外一个重要概念是乘积项。所谓乘积项即 MC 中与阵列的输出，其数量标志了 CPLD 容量，对 CPLD 的性能也有一定的影响，不同厂商的 CPLD 定制的乘积项数目不同。乘积项阵列，实际上就是一个与或阵列，每一个交叉点都是一个可编程熔丝，如果导通就是实现“与”逻辑，在与阵列后一般还有一个“或”阵列，用以完成最小逻辑表达式中的“或”关系。两者配合工作，完成复杂的组合逻辑功能。MC 中的可编程触发器与 FPGA 内部的可编程触发器相似，一般也包含时钟、复位/置位的配置功能，用以完成寄存器或者锁存器等功能。

(9) 布线池、布线矩阵

CPLD 的布线与连通方式与 FPGA 差异较大。前面讲过，FPGA 内部有不同速度、不同驱动能力的丰富连线资源，用以完成 FPGA 内部所有单元之间的互联互通。而 CPLD 的结构比较简单，其布线资源也相对有限，一般采用集中式布线池结构。所谓布线池其本质就是一个开关矩阵，通过打结点可以完成不同 MC 的输入与输出项之间的连接。Xilinx 9500 系列 CPLD 的布线池被称为高速互联与交叉矩阵 (FastCONNECT II Switch Matrix)，而 CoolRunner II 系列 CPLD 则被称为先进的互联矩阵 (AIM, Advanced Interconnect Matrix)；Lattice 的布线池被称为全局布线池 (GRP, Global Routing Pool)；Altera 的布线池叫做可编程互联阵列 (PIA, Programmable Interconnect Array)。由于 CPLD 的器件内部互联资源比较缺乏，所以在某些情况下器件布线时会遇到一定的困难，Lattice 的 LC4000 系列器件在输出 I/O Bank 和功能模块 GLB 之间还添加了一层输出布线池 (ORP, Output Routing Pool)，在一定程度上提高了设计的布通率。

由于 CPLD 的布线池结构固定，所以 CPLD 的输入管脚到输出管脚的标准延时固定，被称为 Pin to Pin 延时，用 T_{pd} 表示。Pin to Pin 延时反应了 CPLD 器件可以实现的最高频率，也就清晰地标明了 CPLD 器件的速度等级。

(10) 其他辅助功能模块

CPLD 中还有一些其他的辅助功能模块，如 JTAG (IEEE 1532、IEEE 1149.1) 编程模块，一些全局时钟、全局使能、全局复位/置位单元等。

1.1.2 FPGA 和 CPLD 的特点

FPGA/CPLD 既继承了 ASIC 的大规模、高集成度、高可靠性的优点，又克服了普通 ASIC 设计周期长、投资大、灵活性差的缺点，逐步成为复杂数字硬件电路设计的理想首选。当代 FPGA、CPLD 有以下特点：

- 规模越来越大。随着 VLSI (Very Large Scale IC, 超大规模集成电路) 工艺的不断提高，单一芯片内部可以容纳上百万个晶体管，FPGA 芯片的规模也越来越大。单片逻辑门数已愈百万，如 Xilinx Virtex-II Pro XC2VP125 已经达到 1250 万门以上的规模。芯片的规模越大所能实现的功能就越强，同时也更适

于实现片上系统 (SOC)。

- 开发过程投资小。FPGA/CPLD 芯片在出厂之前都做过百分之百的测试，而且 FPGA/CPLD 设计灵活，发现错误时可直接更改设计，减少了投片风险，节省了许多潜在的花费。所以不但许多复杂系统使用 FPGA 完成，甚至设计 ASIC 也要把实现 FPGA 功能样机作为必需的步骤。
- FPGA/CPLD 一般可以反复地编程、擦除。在不改变外围电路的情况下，设计不同片内逻辑就能实现不同的电路功能。所以，用 FPGA/CPLD 试制功能样机，能以最快的速度占领市场。甚至在有些领域，因为相关标准协议发展太快，设计 ASIC 可能跟不上技术的更新，只能用 FPGA/CPLD 完成系统的研制与开发。
- FPGA/CPLD 开发工具智能化，功能强大。现在，FPGA/CPLD 开发工具种类繁多、智能化高、功能强大。应用各种工具可以完成从输入、综合、实现到配置芯片等一系列功能。还有很多工具可以完成对设计的仿真、优化、约束、在线调试等功能。这些工具易学易用，可以使设计人员更能集中精力进行电路设计，快速将产品推向市场。
- 新型 FPGA 内嵌 CPU 或 DSP 内核，支持软硬件协同设计，可以作为片上可编程系统 (SOPC) 的硬件平台。

表 1-1 为 CPLD 与 FPGA 的区别与联系对比表，希望通过对照，加深读者对 FPGA 和 CPLD 特点的整体把握。

表 1-1 CPLD 与 FPGA 的区别与联系

项目	FPGA	CPLD	备注
工艺	SRAM	乘积项，物理上有多种实现方法：EPROM、EEPROM、Flash、反熔丝等	
触发器数量	多	少	所以 FPGA 更适合实现时序逻辑，CPLD 多用于实现组合逻辑
Pin to Pin 延时	不可预测	固定	所以对 FPGA 而言时序约束和仿真非常重要
规模与逻辑复杂度	规模大，逻辑复杂度高，新型器件高达千万门级	规模小，逻辑复杂度低	FPGA 用以实现复杂设计，CPLD 用以实现简单设计
成本与价格	成本高，价格高	成本低，价格低	CPLD 用于实现低成本设计
编程与配置	方法多，一般为 RAM 型	一般为 ROM 型。有两种编程方式，一种是通过编程器烧写 ROM，另一种较方便的方式是通过 ISP 模式	FPGA 如果不外挂存储器 (EPROM 等)，掉电将丢失逻辑功能
保密性	一般保密性较差	好	一般 FPGA 不容易实现加密，但是目前一些采用 EECMOS 加 SRAM 工艺的新型器件在内部嵌入了加载 FLASH，如 Lattice XP 系列与 Altera MaxII 系列等
互联结构，连线资源	分布式，丰富的连线资源	集总式，相对连线资源有限	
适用的设计类型	复杂的时序功能	简单的逻辑功能	

尽管 FPGA 与 CPLD 在硬件结构上有一定的差异，但是对用户而言，FPGA 和 CPLD 的

(1) 电路设计与输入

电路设计与输入是根据工程师的设计方法将所设计的功能描述给 EDA 软件。常用的设计输入方法有硬件描述语言 (HDL) 和原理图设计输入法。原理图设计输入法在早期应用得比较广泛, 它根据设计要求, 选用器件、绘制原理图、完成输入过程。这种方法的优点是直观、便于理解、元件库资源丰富。但是在大型设计中, 这种方法的可维护性较差, 不利于模块建设与重用。更主要的缺点是: 当所选用芯片升级换代后, 所有的原理图都要做相应的改动。目前进行大型工程设计时, 最常用的设计方法是 HDL 设计输入法。其中影响最为广泛的 HDL 语言是 VHDL 和 Verilog HDL。它们的共同特点是利于由顶向下设计, 利于模块的划分与复用, 可移植性好, 通用性好, 设计不因芯片的工艺与结构的变化而变化, 更利于向 ASIC 的移植。波形输入和状态机输入方法是两种常用的辅助设计输入方法。使用波形输入法时, 只要绘制出激励波形和输出波形, EDA 软件就能自动地根据响应关系进行设计。而使用状态机输入法时, 设计者只需画出状态转移图, EDA 软件就能生成相应的 HDL 代码或者原理图, 使用十分方便。Xilinx ISE 工具包中的 StateCAD 就能完成状态机输入的功能。但是需要指出的是, 后两种设计方法只能在某些特殊情况下缓解设计者的工作量, 并不适合所有的设计。

(2) 功能仿真

电路设计完成后, 要用专用的仿真工具对设计进行功能仿真, 验证电路功能是否符合设计要求。功能仿真有时也被称为前仿真。常用的仿真工具有 Model Tech 公司的 ModelSim, Synopsys 公司的 VCS, Cadence 公司的 NC-Verilog 和 NC-VHDL, Aldec 公司的 Active HDL VHDL/Verilog HDL 等。通过仿真能及时发现设计中的错误, 加快设计进度, 提高设计的可靠性。

(3) 综合优化

综合优化 (Synthesize) 是指将 HDL 语言、原理图等设计输入翻译成由与、或、非门, RAM, 触发器等基本逻辑单元组成的逻辑连接 (网表), 并根据目标与要求 (约束条件) 优化所生成的逻辑连接, 输出 edf 和 edn 等文件, 供 FPGA/CPLD 厂家的布局布线器进行实现。常用的专业综合优化工具有 Synplicity 公司的 Synplify/Synplify Pro、Amplify, Synopsys 公司的 FPGA Compiler II (Synopsys 公司将停止发展 FPGA Express 软件, 致力于 FPGA Compiler II 平台的开发), Mentor 公司旗下的 Exemplar Logic 公司出品的 LeonardoSpectrum 等。另外, FPGA/CPLD 厂商的集成开发环境也带有一些综合工具, 如 Xilinx ISE 中的 XST 等。

(4) 综合后仿真

综合完成后需要检查综合结果是否与原设计一致, 需要做综合后仿真。在仿真时, 把综合生成的延时文件反标到综合仿真模型中去, 可估计门延时带来的影响。综合后仿真虽然比功能仿真精确一些, 但是只能估计门延时, 而不能估计线延时, 仿真结果与布线后的实际情况还有一定的差距, 并不十分准确。这种仿真的主要目的在于检查综合器的综合结果是否与设计输入一致。

(5) 实现

综合结果的本质是一些由与、或、非门, 触发器, RAM 等基本逻辑单元组成的逻辑网表, 它与芯片实际的配置情况还有较大差距。此时应该使用 FPGA/CPLD 厂商提供的工具软

件，根据所选芯片的型号，将综合输出的逻辑网表适配到具体 FPGA/CPLD 器件上，这个过程就叫做实现（Implementation）过程。Xilinx 的实现过程分为翻译（Translate）、映射（Map）和布局布线（Place & Route）等 3 个步骤。因为只有器件开发商最了解器件的内部结构，所以实现步骤必须选用器件开发商提供的工具软件。

(6) 时序仿真与验证

布局布线之后首先应该做时序仿真。时序仿真中应该将布局布线的时延文件反标到设计中，使仿真既包含门延时，又包含线延时信息。与前面各种仿真相比，这种后仿真包含的延时信息最为全面、准确，能较好地反映芯片的实际工作情况。有时为了保证设计的可靠性，在时序仿真后还要做一些验证。验证的手段比较丰富，可以用 ISE 内嵌时序分析工具完成静态时序分析（STA，Static Timing Analyzer），也可以用第三方验证工具（如 Synopsys 的 Formality 验证工具，PrimeTime 静态时序分析工具等）进行验证。可以用 ISE 内嵌的 FPGA Editor 和 Chip Viewer 观察芯片内部的连接与配置情况，或者用 ISE 内嵌的 ChipScope Pro 对设计进行在线逻辑分析。在有些高速设计情况下还需要使用第三方的板级验证工具进行仿真与验证，如 Mentor Tau、Forte Design - Timing Designer、Mentor Hyperlynx、Mentor ICX、Cadence SPECCTRAQuest、Synopsys HSPICE 等工具。需要强调的是，对于一般的逻辑设计，可能选择或者忽略复杂的验证手段，但是布局布线后时序仿真是必选项，只有通过时序仿真检查布局布线结果有无时序违规，才能在较大程度上保证设计的稳定性与可靠性。

(7) 调试与加载配置

设计开发的最后步骤就是在线调试或者将生成的配置文件写入芯片中进行测试。在 ISE 中对应的工具是 iMPACT。

任何仿真或验证步骤出现问题，就需要根据错误的定位返回到相应的步骤更改或者重新设计。

1.3 ISE 系列产品的特点

本节将介绍 ISE 系列产品的特点，并重点介绍 ISE 的新增功能。

1.3.1 特点综述

Xilinx 作为当今世界上最大的 FPGA/CPLD 生产商之一，长期以来一直推动着 FPGA/CPLD 技术的发展。其开发的软件也不断升级换代，由早期的 Foundation 系列逐步发展到目前的 ISE 6.x 系列。ISE 是集成综合环境的简称，它是 Xilinx FPGA/CPLD 的综合性集成设计平台，该平台集成了从设计输入、仿真、逻辑综合、布局布线与实现、时序分析、芯片下载与配置、功率分析等几乎所有设计流程所需工具。

利用 ISE 集成的工具可以完成上述整个 FPGA/CPLD 的开发过程。

ISE 的主要特点如下：

- ISE 是一个集成环境，可以完成整个 FPGA/CPLD 开发过程。ISE 集成了很多著名的 FPGA/CPLD 设计工具，根据设计流程合理应用这些工具，会使工程师的设计工作如鱼得水。

- ISE 界面风格简洁流畅，易学易用。ISE 的界面秉承了可视化编程技术，界面根据设计流程而组织，整个设计过程只需按照界面组织结构依次点击相应的按钮或选择相应的选项即可。
- ISE 有丰富的在线帮助信息，结合 Xilinx 的技术支持网站，一般设计过程中可能遇到的问题都能得到很好的解决。
- 强大的设计辅助功能。ISE 秉承了 Xilinx 设计软件的强大辅助功能。在编写代码时可以使用编写向导生成文件头和模块框架，也可使用语言模板（Language Templates）帮助编写代码。在图形输入时可以使用 ECS 的辅助项帮助设计原理图。另外，ISE 的 Core Generator 和 LogiBLOX 工具可以方便地生成 IP Core（IP 核）与高效模块为用户所用，大大减少了设计者的工作量，提高了设计效率与质量。

1.3.2 ISE 的新增特性

目前，ISE 的最新版本是 ISE 6.2，工程界流行的 ISE 主要版本为 ISE 5.x 与 ISE 6.x，它们的新特性如下：

- ISE 6.2 支持包括最新 Spartan 3 系列器件的所有 Xilinx FPGA/CPLD 主流器件。这些器件从速度到容量上涵盖了大多数用户的需求。
- ISE 6.2 支持局部重构技术（Partial Reconfiguration），即在 FPGA 在线工作情况下，对 FPGA 局部进行重新加载与配置。这项技术能更好的支持模块化设计（Modular Design）方法，并且使 FPGA 的加载与配置更加灵活。Xilinx 是首家支持局部重构技术的 FPGA 制造商。
- ISE 使大多数设计工作更高效。这主要表现在设计的速度和设计所占用的面积两个方面。新版 ISE 通过改进综合、实现等关键环节的优化手段与方法，提高了设计的工作频率，减小了设计消耗的面积，使设计结果能更好地调用芯片的内部资源，工作更高效。
- ISE 采用了增量式设计（Incremental Design）流程。这里的增量设计主要包含两方面：增量式综合与增量式布局布线。增量式综合是在综合过程仅对修改过的模块进行重新编译，保持未改变设计的原有综合结果。增量式布局布线与之相似，对未变化模块保持原有布局布线结果，仅对更改部分进行重新布局布线。增量设计流程大大减少了综合与布局布线所消耗的时间，保证了未改变部分的实现结果与上次一致，良好地继承了以往的成功设计，提高了设计的可靠性。
- 结构设计向导（Architecture Wizard）是 ISE 5.x 新增的一个重要工具。它能方便地定制数字时钟管理器（DCM，Digital Clock Manager）和高速 I/O（Rocket I/O™），生成可综合的 HDL 源代码。ISE 6.x 对 Architecture Wizard 进行了改进，即在 Rocket I/O Wizard 中可以通过选项直接控制编码、解码、参考时钟、接收指示信号（Receiver Comma）等参数特性。
- 引脚与区域约束编辑器（PACE，Pinout and Area Constraints Editor）也是 ISE

5.1 及更高版本的一个非常用的新增工具。它能方便地指定 I/O 管脚约束，直接引出测试脚，并能对设计进行区域约束，自动生成用户约束文件，是约束编辑器（Constraints Editor）的一个有益补充。

- 对结构化仿真支持得更好，从实现结果提炼出的仿真模型与原设计的结构和端口吻合得更好，增进了仿真的可读性。
- ISE 工程管理器（Project Navigator）的信息显示区增加了对所有操作步骤的命令显示，使用这些命令可以采用批处理命令方式运行 ISE 内部工具，完成综合、翻译、映射、布局布线直至配置文件生成。
- IP 核生成器（Core Generator）新增了生成用户 IP Core（知识产权模块）功能。在 IP Capture 工具指导下，用户可以生成自己的 IP 核。一方面实现了模块复用，另一方面保护了知识产权。在 ISE 6 版本中新增了 Core Generator 使用向导以方便用户生成 IP Core。
- Xilinx FPGA/CPLD 配置软件（iMPACT）自动对下载环境进行边界扫描，检测所选器件的类型，根据设置将配置文件烧写入 FPGA/CPLD 或者配套型号的 EEPROM 中。ISE 5 及其高级版本将 iMPACT 与配置文件生成器（BitGen）集成起来，使生成配置文件和配置 FPGA/CPLD 的过程更加符合操作习惯。
- 综合、实现过程的时序分析报告更加明确易懂。从时序分析报告中，用户可以方便地读出设计的工作频率、关键路径和不满足约束的每个细节情况。
- HDL 编辑器（HDL Editor）也新增了一些功能，如每行源代码前有行标号显示，增进了对用户约束文件的色彩区分显示和语言模板支持，改善了“查找”和“替换”等功能。
- 新增了布局规划器（Floorplanner）与时序分析器（Timing Analyzer）之间的互动交叉链接的功能。在 Timing Analyzer 的时序报告文件中点击某一路径，就会自动切换到布局规划器的用户界面下，并使布线图中相应路径高亮显示。
- 新增了时序提高向导（Timing Improvement Wizard）功能。该功能可帮助用户在时序分析器界面下，对不满足时序约束的关键路径的设计加以改正。
- 增强了工程管理器（Project Navigator）的功能。ISE 的程序浏览器和第三方软件的接口更友好，能方便地生成第三方仿真文件。可以用 EDIF 文件作为工程的源文件，也可以在更多的操作系统上安装应用，如 Linux RedHat、Solaris 和 Windows NT/98 se/Me/2000 Professional/XP 等。
- ISE 的约束编辑器（Constraints Editor）支持更多的约束内容。如对第二全局时钟、输入/输出端口寄存器进行设置等。在 ISE 6.2 中还可以直接生成控制偏移约束（OFFSET）和限定性约束（Qualifier），从而使 DDR 寄存器的相关约束更加丰富。
- 图形设计输入界面（ECS）功能也有所增强，增加了自动布线和层次化元件库模型等功能。
- 集成的 HDL 测试文件自动生成器（HDL Benchner）。HDL Benchner 可以帮助用户根据设计输入方便地生成测试文件。

- 集成的图形化状态机输入辅助设计软件 (StateCAD)。StateCAD 可以根据从图形界面输入状态转移图自动生成相应的代码, 完成状态机的设计。
- 功耗计算软件 (XPOWER) 支持更全面的器件功耗计算并改善了用户界面, 提高了易用性。
- 综合软件支持更多器件, 而且综合优化手段更加有效。ISE 集成的第三方综合优化软件有 Synplify/Synplify Pro 和 LeonardoSpectrum 等。另外, Xilinx 的综合工具 XST 也有较大幅度的改善与提高。
- 在线逻辑分析仪 (ChipScope Pro) 的 Core Inserter 集成到 Project Navigator 界面下, 用户可以在 Project Navigator 中直接调用该工具。
- 模块化设计 (Modular Design) 工具包已经直接集成到 ISE 5.2 以及更高版本中, ISE 用户不需另行购买。Modular Design 是一种适用于大型复杂设计的并行协同性设计工具。
- 对第三方工具 EDA 设计、验证工具的支持越来越完善。随着 Xilinx 与其他 EDA 厂商的合作日趋紧密, ISE 对第三方工具的支持越来越完善。可与 ISE 配合使用的 EDA 工具有 LEDA 寄存器传输级分析工具, Synplicity 公司的 Synplify、Synplify Pro、Amplify Physical Optimizer 等综合工具与 Identify 验证工具, Mentor 公司的 LeonardoSpectrum 综合工具和 Seamless-CVE 联合验证工具, Synopsys 公司的 Formality 验证工具和 PrimeTime 时序分析工具, VVerplex 公司的 Conformal 逻辑验证工具, CoWare 公司的软硬件联合设计工具, Celoxica 公司的 DK1.1 联合设计与验证工具包等。

ISE 作为高效的 EDA 设计工具的集合, 与第三方软件取长补短, 使软件功能越来越强大, 为用户提供了更加丰富的 Xilinx FPGA/CPLD 设计平台。

1.4 ISE 6.x 支持的器件

ISE 6.x 支持所有 Xilinx 的 FPGA/CPLD 主流产品, 而对于逐步淘汰的 Spartan、SpartanXL 和 XC4000E/EX/L/XL/XLA 系列 FPGA 将不再支持。如果用户仍在使用上述老型号 FPGA, 必须安装 ISE 4 等早期产品。ISE 6.x 支持如下系列的 FPGA/CPLD:

- Virtex -E/-II/-II Pro/-II Pro X;
- QPro Virtex E/-II Hi-Rel;
- QPro Virtex -II Rad-Hard;
- Spartan -II/-IIE/-3;
- XC9500 XL/XV;
- CoolRunner XPLA3/-II。

1.5 ISE 的 4 个软件系列

ISE 系列软件分为 4 个系列: WebPACK、BaseX、Foundation 和 Alliance。ISE WebPACK 系列可以在 www.Xilinx.com 网站上直接下载, 是一个免费软件, 支持一些常用

器件族；ISE BaseX 系列支持的器件最大规模不超过 700K 系统门；ISE Foundation 系列是早期 Foundation 系列的延伸；ISE Alliance 系列支持的器件族最全，功能强大，是 Xilinx 的主推设计平台。

这几个系列支持器件信息如表 1-2 所示，从表中可以看到 ISE Alliance 系列支持全系列器件。

表 1-2 ISE 不同系列支持器件情况对比表

Device Family/Series	ISE WebPACK	ISE BaseX	ISE Foundation	ISE Alliance
Virtex™ Series	Virtex-E: V50E - V300E Virtex-II: 2V40 - 2V250 Virtex-II Pro: 2VP2	Virtex: V50 - V600 Virtex-E: V50E - V600E Virtex-II: 2V40 - 2V500 Virtex-II Pro: 2VP2, 2VP4, 2VP7	All	All
Spartan™ Series	Spartan II/IIE: All (except XC2S400E and XC2S600E) Spartan-3: 3S50, 3S200, 3S400	Spartan II/IIE: All Spartan-3: 3S50, 3S200, 3S400	All	All
CoolRunner™ XPLA3 CoolRunner-II	All	All	All	All
XC9500™ Series	All	All	All	All

另一方面，ISE 的不同系列内嵌的设计工具与支持的第三方工具的数量也不同，表 1-3 列出了 ISE 不同系列支持的工具特性。从表中可以看到 ISE Alliance 系列支持的工具种类最全。

表 1-3 ISE 不同系列支持的工具特性

	功能特性	ISE WebPACK	ISE BaseX	ISE Foundation	ISE Alliance
器件族支持情况	Virtex 系列	Virtex-E: V50E~V300E Virtex-II: 2V40~2V250 VirtexII Pro: 2VP2	Virtex: V50~V600 Virtex-E: V50E~V600E Virtex-II: 2V40~2V500 VirtexII Pro: 2VP2, 2VP4, 2VP7	所有 Virtex、Viretex E、Virtex II、Virtex II Pro 器件族	所有 Virtex、Viretex E、Virtex II、Virtex II Pro 器件族
	Spartan 系列	Spartan II/IIE : 除 XC2S400E 和 XC2S600E 外的全部器件 Spartan-3 : 3S50, 3S200, 3S400	Spartan II/IIE : 全部器件 Spartan-3 : 3S50, 3S200, 3S400	Spartan II/IIE : 全部器件 Spartan-3 : 全部器件	Spartan II/IIE : 全部器件 Spartan-3 : 全部器件
	CoolRunner XPLA3 CoolRunner II	全部器件	全部器件	全部器件	全部器件
	XC9500 系列	全部器件	全部器件	全部器件	全部器件

续表

	功能特性	ISE WebPACK	ISE BaseX	ISE Foundation	ISE Alliance
设计输入方式	原理图编辑器 Schematic Editor	支持	支持 (Windows 和 Linux 版本)	支持 (Windows 和 Linux 版本)	支持
	语法编辑器 HDL Editor	支持	支持	支持	支持
	状态转移图输入方式 State Diagram Editor	支持	支持 (Windows 版本)	支持 (Windows 版本)	支持 (Windows 版本)
	IP Core 生成器 Core Generation System	支持	支持	支持	支持
	引脚与区域约束编辑器 PACE	不支持	支持	支持	支持
	结构向导 Architecture Wizards DCM: Digital Clock Management MGT: Multi-Gigabit Transceivers	支持	支持	支持	支持
	第三方 IP	支持	支持	支持	支持
	DSP 系统生成器 Xilinx System Generator for DSP	不支持	可选付费项目	可选付费项目	可选付费项目
嵌入式系统设计	GNU: 嵌入式工具 GCC: GNU 编译器 GDB: GNU 软件调试器	不支持	支持 (在可选工具 EDK 中集成)	支持 (在可选工具 EDK 中集成)	支持 (在可选工具 EDK 中集成)
	WindRiver Xilinx Edition 开发工具: Diab C/C++编译器 SingleStep Debugger Vision PROBE II target connection	不支持	可选付费项目	可选付费项目	可选付费项目
综合工具	XST	支持	支持	集成界面 (Windows)	支持
	Synplicity Synplify/Synplify Pro	集成接口 需要另外购买	集成接口 (Windows) 需要另外购买	集成接口 (Windows) 需要另外购买	集成接口 (Windows) 需要另外购买
	Mentor Graphics Leonardo Spectrum	集成接口 需要另外购买	集成接口 需要另外购买	集成接口 需要另外购买	集成接口 需要另外购买
	Mentor Graphics Precision RTL	仅支持 EDIF 流程	仅支持 EDIF 流程	仅支持 EDIF 流程	仅支持 EDIF 流程
	Synopsys FPGA Compiler II	仅支持 EDIF 流程	仅支持 EDIF 流程	仅支持 EDIF 流程	仅支持 EDIF 流程
	ABEL	支持 CPLD	支持 CPLD (Windows)	支持 CPLD (Windows)	支持 CPLD (Windows)

续表

	功能特性	ISE WebPACK	ISE BaseX	ISE Foundation	ISE Alliance
实现相关工具	布局规划器 Floorplanner	支持	支持	支持	支持
	约束编辑器 Constraints Editor	支持	支持	支持	支持
	时序驱动布局布线 Timing Driven PAR	支持	支持	支持	支持
	模块化设计 Modular Design	不支持	支持	支持	支持
	时序改进向导 Timing Improvement Wizard	支持	支持	支持	支持
编程配置	iMPACT	支持	支持	支持	支持
	ACE 编程配置管理器 System ACE configuration Manager	支持	支持	支持	支持
板级验证	IBIS 模型	支持	支持	支持	支持
	STAMP 模型	支持	支持	支持	支持
	HSPICE 模型	支持	支持	支持	支持
仿真与验证	测试激励生成器 HDL Benchmer	支持	支持 (Windows)	支持 (Windows)	支持 (Windows)
	ModelSim Xilinx 版本 (MXE II)	ModelSim XE II Starter	ModelSim XE II Starter	ModelSim XE II Starter	ModelSim XE II Starter
	静态时序分析 STA	支持	支持	支持	支持
	在线逻辑分析仪 ChipScope Pro	可选付费工具	可选付费工具	可选付费工具	可选付费工具
	FPGA Editor 包含 Probe 功能	不支持	支持	支持	支持
	芯片观察器 Chip Viewer	支持	支持	支持	支持
	XPower 功耗分析器	支持	支持	支持	支持
	第三方 Equivalency Check (等效检查)	支持	支持	支持	支持
	PPC 和 RocketIO 的 SMART 模型	不支持	支持	支持	支持
	第三方仿真工具	支持	支持	支持	支持
操作系统		PC (Windows 2000/XP)	PC (Windows 2000/XP), Linux	PC (Windows 2000/XP), Linux, Sun Solaris	PC (Windows 2000/XP), Linux, Sun Solaris

由于 ISE Alliance 系列支持的器件族最全，支持的工具种类也最全，所以推荐大家安装

ISE Alliance 系列。本书约定，书中所有论述和范例也是基于 ISE Alliance 6.2 系列软件的。

1.6 ISE 的系统配置与安装

本节简单介绍 ISE 的推荐系统配置与安装流程。

1.6.1 推荐的系统配置

ISE 推荐的系统配置与设计所选的芯片型号有关。总的来说因为综合与实现过程运算量巨大，系统配置要求较高。当工作的芯片规模较大，而且设计复杂，时序要求较高时，系统配置低的计算机将无法完成设计流程。

系统推荐的配置较高，有的需要配置工作站。为了提高综合、实现过程的速度，除了要提高系统 CPU、主板和硬盘的工作速度外，最主要的是提高内存的配置。系统推荐的内存配置如表 1-4 所示。

表 1-4 ISE 6.2 系统推荐内存配置

Xilinx 器件类型	RAM	虚拟内存
XC9500./XL/XV CoolRunner., CoolRunnerII. Spartan-II. XC2S15 到 XC2S200 Spartan-II E.XC2S50E 到 XC2S200E Spartan-3. XC3S50 到 XC3S200 Virtex. XCV50 到 XCV150 Virtex-E. XCV50E 到 XCV200E Virtex-II. XC2V40 到 XC2V250 QPro Virtex./E/-II Hi-Rel QPro Virtex./-II Rad-Hard	128MB	128MB
Spartan-II E XC2S300E 到 XC2S600E Spartan-3 XC3S400 Virtex XCV300 到 XCV400 Virtex-E XCV400E Virtex-II XC2V500 Virtex-II PRO XC2VP2 QPro Virtex/E/-II Hi-Rel QPro Virtex/-II Rad-Hard	256MB	256MB
Spartan-3 XC3S1000 Virtex XCV600 到 XCV800 Virtex-E XCV600E Virtex-II XC2V1000 Virtex-II PRO XC2VP4, XC2VP7 QPro Virtex/E/-II Hi-Rel QPro Virtex/-II Rad-Hard	512MB	512MB

续表

Xilinx 器件类型	RAM	虚拟内存
Spartan-3 XC3S1500 到 XC3S2000 Virtex XCV1000 Virtex-E XCV1000E, XCV2000E Virtex-II XC2V1500 Virtex-II XC2VPX Virtex-II PRO XC2VP20 到 XC2VP40 QPro Virtex/E/-II Hi-Rel QPro Virtex/-II Rad-Hard	1GB	1GB
Spartan-3 XC3S4000 and XC3S5000 Virtex-E XCV2600E, XCV3200E Virtex-II XC2V2000, 到 XC2V6000 Virtex-II PRO XC2VP50 到 XC2VP100 QPro Virtex/E/-II Hi-Rel QPro Virtex/-II Rad-Hard	2GB	2GB
Virtex-II XC2V8000 Virtex-II PRO XC2VP125 QPro Virtex/E/-II Hi-Rel QPro Virtex/-II Rad-Hard	3GB	3GB

ISE 6.2 所支持的操作系统如表 1-5 所示。

表 1-5 ISE 6.2 所支持的操作系统

操作系统	版本名称
Windows	NT/98 se/Me/2000 Professional/XP
Solaris	2.6/2.7/2.8/5.8
Linux/Wine	Red Hat Linux 8.0 (Full support), 9.0(Limited support)

1.6.2 ISE 的安装

ISE 5.x 和 ISE 6.x 改变了 license 管理方式，这两个版本在安装后并不需要任何 license 支持，仅仅是在安装过程中输入 ISE 的注册序列号即可 (Register ID)。安装 ISE 时只需根据所选的版本是在 PC 或工作站上，然后根据软件的提示安装即可，这里不再赘述。但需要说明如下 3 点。

1. 环境变量的设置

安装过程结束后，为了能正常使用 ISE，还需设置 ISE 的环境变量。假设在 PC 上 ISE 的安装目录为 c:\Xilinx。

- 如果操作系统是 Windows 98，需要在 autoexec.bat 文件中加入
set Xilinx=c:\Xilinx

set PATH=%Xilinx%\bin\nt

- 如果操作系统是 Windows 2000，选择【开始】/【设置】/【控制面板】命令，打开控制面板，双击【系统】选项，在【系统特性】对话框中选择【高级】/【环境变量】选项，在环境变量中加入

变量名：Xilinx

变量值：C:\Xilinx

具体设置如图 1-4 所示。

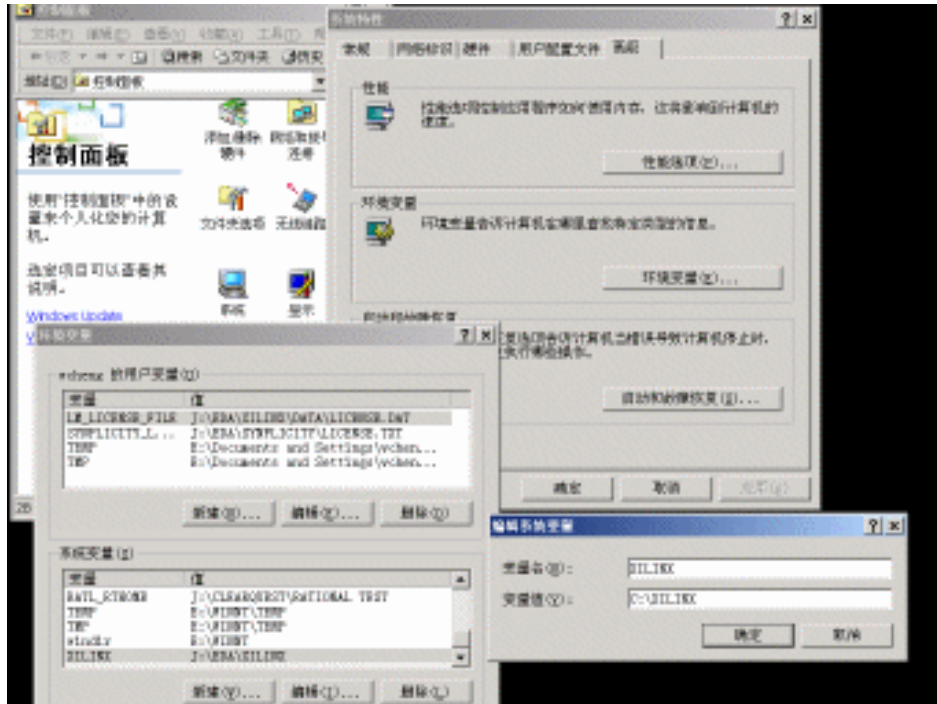


图1-4 Windows 2000 系统环境变量配置

- 如果操作系统是 Windows NT，加入过程与 Windows 2000 相似。选择【开始】/【设置】/【控制面板】命令，打开控制面板，双击【系统】选项，在【系统特性】对话框中选择【高级】/【环境变量】选项，在环境变量中加入
变量名：Xilinx
变量值：C:\Xilinx
- 在工作站上安装 ISE 时，设置参数有两种方法，一是在 shell 下运行 settings.csh 或 settings.sh，二是手动设计参数。假设安装 ISE 到 Xilinx 目录，则在手动设置参数时，需要输入命令 setenv Xilinx/Xilinx，然后将 Xilinx 工作目录加入路径 set path= (\$Xilinx/bin/platform \$path)。在 Solaris (sol) 系统为 setenv LD_LIBRARY_PATH \$ (Xilinx) /bin/，platform: (LD_LIBRARY_PATH)；在 .HP (hp) SHLIB_PATH 系统为 setenv SHLIB_PATH \$ (Xilinx) /bin/hp:lib:/usr/。
- 如果操作系统是 Linux，编辑 WINE 配置文件/usr/local/etc/wine.conf，加入

Xilinx=C:\<your_Xilinx_dir> 。 并 且 设 置 环 境 变 量 setenv
XIL_LINUX_WINE=1。

2. 安装 ISE 的帮助文档

ISE 的帮助文件非常丰富，除了在线帮助文件外，在 Xilinx 公司的网站上还有丰富的 html 和 pdf 帮助文件。下面说明各种 pdf 文件的内容：

- cgd.pdf——约束指南；
- cgn.pdf——CORE Generator 生成 IP Core 使用手册；
- dev.pdf——开发系统参考；
- gls.pdf——全局名词说明；
- pac.pdf——iMPACT 配置工具使用手册；
- irm.pdf——ISE 版本说明与安装指南；
- lib.pdf——器件库手册；
- men.pdf——Mentor 图形界面使用说明（ModelSim，LeonardoSpectrum 都系 Mentor 公司子公司开发）；
- sim.pdf——综合和仿真工具使用手册；
- hdl.pdf——Xilinx/Concept-HDL 语言说明；
- qst.pdf——ISE 快速教学浏览（包含于 ISE 6 资料文档中）；
- xsi.pdf——Xilinx/Synopsys 接口说明；
- xst.pdf——Xilinx 综合工具使用手册。

3. 安装第三方软件

安装 ISE 后，还要安装第三方软件，如 FPGA Express、ModelSim 和 LeonardoSpectrum 等。安装方法十分简单，只要选择安装光盘上相应的目录，运行 Setup 程序，按照提示完成即可。但是要注意两点事项：一是应该先安装 ISE，后安装第三方工具；二是第三方工具需要申请相应的 license，并设置 license 管理项目。

1.7 ISE 中集成工具及其基本功能

本节主要介绍 ISE 中各种集成工具的名称与它们在设计中的作用。

ISE 中的工具可以分为两部分，一是 Xilinx 自己提供的软件工具，另外一些是其他 EDA 厂商提供的软件工具，后者统称为第三方工具。Xilinx 自带软件在 ISE 安装完成后即可直接使用，大多数不需任何 License 授权，包括了 FPGA/CPLD 设计流程中的常用工具。比较常用的 Xilinx 自带软件有：HDL Editor（HDL 代码编辑器）、Core Generator（IP 核生成器）、Architecture Wizard（结构设计向导）、System Generator for DSP（DSP 生成器）、ECS（原理图编辑器）、XST 综合工具、Constraints Editor（约束编辑器）、PACE（引脚与区域约束编辑器）、Timing Analyzer（时序分析器）、FPGA Editor（FPGA 底层编辑器）、Chip Viewer（芯片观察窗）、Floorplanner（布局规划器）、HDL Benchmer（测试激励生成器）、Modular Design（模块化设计）、Incremental Design（增量设计）、Timing Improve Wizard（时序优化向导）、iMPACT 下载配置器、Xpower（功耗仿真器）和 ChipScope Pro（在线逻

辑分析仪)。

第三方软件指专门 EDA 工具生产商提供的设计工具，ISE 集成了与这些设计工具的友好接口，在 ISE 中可以直接调用这些工具。第三方工具一般需要 License 授权方可使用。ISE 中集成的第三方工具接口有：State CAD (状态机编辑器)、Synplify/Synplify Pro 综合工具、Amplify 综合工具、Identify 验证工具、Certify ASIC 设计工具、Mentor Precision RTL 综合工具、Mentor Leonardo 综合工具、Synopsys FPGA Compiler II 综合工具、ModelSim Xilinx Edition 仿真工具、Synopsys 的 Formality 验证工具、Synopsys PrimeTime 静态时序分析工具、板级仿真验证工具 (Mentor Tau、Forte Design - Timing Designer、Mentor Hyperlynx、Mentor ICX、Cadence SPECCTRAQuest 和 Synopsys HSPICE 等)、SOC 设计工具 (Wind River Xilinx Edition、System ACE) 等。

表 1-6 详细罗列了以功能划分的 ISE 6.2 支持的第三方工具名称。

表 1-6 ISE 6.2 支持的第三方工具详细列表

Xilinx 器件族	Virtex-E	Virtex-II	Virtex-II Pro	Spartan-II	Spartan-IIE	Spartan-3	CoolRunner-II
HDL 代码检查工具							
Synopsys LEDA 4.0	Y	Y	Y	Y	Y	Y	
HDL 综合工具							
Synplicity Synplify Pro	Y	Y	Y	Y	Y	Y	Y
Mentor LeonardoSpectrum	Y	Y	Y	Y	Y	Y	Y
Mentor Precision RTL Synthesis	Y	Y	Y	Y	Y	Y	Y
Synopsys FPGA Compiler II	Y	Y	Y	Y	Y	Y	
Synopsys Design Compiler	Y	Y	Y	Y	Y	Y	
物理优化工具							
Synplicity Amplify	Y	Y	Y	Y	Y	Y	
Magma/Aplus Design PALACE	Y	Y	Y	Y	Y	Y	
Hier Design PlanAhead		Y	Y			Y	
RTL 调试工具							
Synplicity Identify	Y	Y	Y	Y	Y	Y	
ASIC Prototyping 工具							
Synplicity Certify	Y	Y	Y				
Co-Design 软硬件联合设计							
Celoxica DK-1	Y	Y	Y	Y	Y		
Endeavor Co-Simple			Y				
Mentor Seamless			Y				
Novilit Protocol AnyWare		Y	Y				
Cadence Concept	Y	Y		Y	Y		Y
Impulse Codeveloper	Y	Y	Y	Y	Y	Y	

续表

Xilinx 器件族	Virtex-E	Virtex-II	Virtex-II Pro	Spartan-II	Spartan-IIE	Spartan-3	CoolRunner-II
HDL 仿真工具							
Mentor ModelSim(Verilog/VHDL)	Y	Y	Y	Y	Y	Y	Y
Cadence NC-Sim(Verilog/VHDL)	Y	Y	Y	Y	Y	Y	Y
Synopsys VCS-MX/VCS-MXi(Verilog/VHDL)	Y	Y	仅 Linux & Solaris 支持	Y	Y	Y	Y
形式验证工具							
Synopsys Formality	Y	Y	Y	Y	Y	Y	
芯片级 STA 分析工具							
Synopsys Prime Time		Y	Y			Y	
板级 STA 分析							
Mentor Tau	Y	Y	Y	Y	Y	Y	Y
Forte Design Timing Designer	Y	Y	Y	Y	Y	Y	Y
板级信号完整性分析							
Mentor Hyperlynx	Y	Y	Y	Y	Y	Y	Y
Mentor ICX	Y	Y	Y	Y	Y	Y	Y
Cadence SPECCTRAQuest	Y	Y	Y	Y	Y	Y	Y
Synopsys HSPICE			Y				

可见 ISE 中集成的设计工具非常丰富，限于篇幅，本书只能对传统 FPGA 设计流程涉及的常用工具加以讨论。根据设计流程与功能划分，ISE 的集成工具主要分为设计输入工具、综合工具、仿真工具、实现工具和辅助设计工具等 5 类。

1. 设计输入工具

设计输入是工程设计的第一步，ISE 集成的设计工具主要包括 HDL 编辑器（HDL Editor）、状态机编辑器（StateCAD）、原理图编辑器（ECS）、IP 核生成器（Core Generator）和测试激励生成器（HDL Bencher）等。

- HDL 编辑器（HDL Editor）可以完成设计电路的 HDL 语言的输入。它能根据语法来彩色显示关键字。支持 VHDL、ABEL 和 Verilog HDL 的输入。
- 状态机编辑器（StateCAD）采用最自然的方式——状态转移图设计状态机。设计者只需画出状态转移图，状态机编辑器就能自动生成相应的 VHDL、ABEL 或 Verilog HDL 语言模型。并且状态机编辑器能生成状态转移的测试激励文件，验证寄存器传输级（RTL）模型，优化并分析状态机设计结果。使用 StateCAD 设计状态机，生成的代码规范、清晰，能在一定程度上缓解设计者的工作量。
- 原理图编辑器 ECS（Schematic Editor）用于完成电路的原理图输入。它功能强大、元件库齐全、设计方便。原理图输入方式在大规模设计中逐渐被 HDL

语言输入方式所取代，所以本书推荐初学者尽量采用 HDL 语言方式设计电路。

- IP 核生成器 (Core Generator) 是 Xilinx FPGA 设计中的一个重要设计输入工具。它提供了大量 Xilinx 和第三方公司设计的成熟、高效 IP 核 (IP Core) 为用户所用。IP 核生成器可生成的 IP 核功能繁多，从简单的基本设计模块到复杂的处理器等一应俱全，分为基本模块，通信与网络模块，数字信号处理模块，数学功能模块，存储器模块，微处理器、控制器与外设模块，标准与协议设计模块，语音处理模块，标准总线模块和视频与图像处理模块等 10 大功能模块。配合 Xilinx 网站的 IP 中心使用，能大幅度地减轻工程师的设计工作量，提高设计质量。
- 测试激励生成器 (HDL Benchner) 辅助用户设计测试激励文件。它将 VHDL 源代码、Verilog 源代码和 ECS 原理图等设计输入导入其测试环境，根据用户在图形界面下编辑的激励波形，直接生成测试激励文件，然后调用 ISE 中集成的仿真工具进行仿真验证，并分析测试激励的覆盖率。

2. 综合工具

ISE 集成的综合工具主要有 Synplicity 公司的 Synplify/Synplify Pro, Synopsys 公司的 FPGA Compiler II/Express, Exemplar Logic 公司的 LeonardoSpectrum 和 Xilinx ISE 中的 XST 等。

- Synplify/Synplify Pro 作为新兴的综合工具，在综合策略和优化手段上有较大幅度的提高，特别是其先进的 Timing Driven (时序驱动) 和 B.E.S.T (行为级综合提取技术) 算法引擎，使其综合结果往往面积较小，速度较快，在业界口碑很好。如果结合 Synplicity 公司的 Amplify 物理约束功能，对很多设计能大幅度地减少资源，优化面积达到 30% 以上。
- Synopsys 公司作为较早与 Xilinx 合作的 EDA 软件公司，对 Xilinx 器件内部结构比较了解。在 Xilinx 较早版本的集成开发环境 Foundation 系列软件中，FPGA Express 是唯一集成的综合工具。FPGA Express 的综合结果比较忠实于原设计，其升级版本 FPGA Compiler II 是最好的 ASIC/FPGA 设计工具之一。需要指出的是 ISE 5 系列不再直接集成 FPGA Express 综合工具，如果需要使用 Synopsys 的 FPGA Express / Compiler II 系列综合工具，需要使用 ISE 4 等早期 ISE 版本，在 FPGA Express / Compiler II 系列综合工具中完成综合，导出 EDIF 网表，在 ISE 中使用 EDIF 流程对设计进行布局布线。
- Mentor 的子公司 Exemplar Logic 出品的 LeonardoSpectrum 也是一款非常流行的综合工具。它的综合优化能力也非常高，随着 Exemplar Logic 与 Xilinx 的合作日趋紧密，LeonardoSpectrum 对 Xilinx 器件的支持也越来越好。
- XST (Xilinx Synthesis Technology) 是 Xilinx 自主开发的综合工具。虽然 Xilinx 设计综合软件的经验还不够丰富，但只有 Xilinx 自己对其芯片的内部结构最了解，所以 XST 的一些优化策略是其他综合工具无法比拟的。XST 对某些使用到 Xilinx 内部核心的设计的综合结果甚至要比其他综合工具优越很多。

3. 仿真工具

ISE 集成的仿真工具主要有 Model Tech 公司的仿真工具 ModelSim 和测试激励生成器 HDL Benchner 等。

- ModelSim 可以说是业界最流行的仿真工具之一。其主要特点是仿真速度快，仿真精度高。ModelSim 支持 VHDL、Verilog HDL 以及 VHDL 和 Verilog HDL 混合编程的仿真。ModelSim 的 PC 版的仿真速度也很快，甚至和 workstation 版不相上下。
- HDL Benchner 是一种根据电路设计输入，自动生成测试激励的工具，它可以把工程师从书写测试激励文件的繁重工作中部分解脱出来。HDL Benchner 的 Xilinx 版本可以支持 VHDL 语言输入，Verilog HDL 语言输入和 Xilinx 原理图输入等 3 种输入方法。将这些设计输入导入到 HDL Benchner 中，就能自动生成相应的测试激励文件。

4. 实现工具

实现工具包含的面比较广。如果能较好地掌握这些工具，将大幅度提高设计者的水平，使设计工作更加游刃有余。ISE 集成的实现工具主要有约束编辑器 (Constraints Editor)、引脚与区域约束编辑器 (PACE)、时序分析器 (Timing Analyzer)、FPGA 底层编辑器 (FPGA Editor)、芯片观察窗 (Chip Viewer) 和布局规划器 (Floorplanner) 等。

- 约束编辑器是帮助工程师设计用户约束文件 (ucf) 的工具。用户约束文件是指导实现过程的约束文件。它与指导综合过程的约束文件既有区别又有联系。约束文件包含时钟属性、延时特性、管脚位置、寄存器分组、布局布线要求和特殊属性等信息，这些信息指导实现过程，是由用户设计的决定电路实现的目标与标准。设计约束文件有较高的技巧性，如果约束文件设计得当，会帮助 ISE 达到用户的设计目标，如果过约束或者约束不当，会影响电路特性。调用 Constraints Editor 的方法有两种：一是在 Windows 系统中选择【开始】/【程序】/【Xilinx ISE】/【Accessories】/【Constraints Editor】命令；二是在 ISE 工程管理器 (Project Navigator) 界面下打开操作流程调用 Constraints Editor。完成翻译网表 (Translate) 后调用 Constraints Editor 可以充分显示综合网表中的时钟路径和关键路径等信息。
- 引脚与区域约束编辑器 (PACE, Pinout and Area Constraints Editor) 是 ISE 5 的一个新增工具，它可以直接将信号指定到 I/O 管脚，方便地拉出测试信号，对设计进行面积约束，自动生成用户约束文件，是约束编辑器的有益补充。
- 时序分析器是分析实现结果是否满足约束条件、芯片的工作速率以及关键路径等延时信息的工具。能方便地将实现过程生成的各种时延报告分类显示，并对比约束文件，分析是否满足时序要求。
- 芯片观察窗给用户提供一个图形界面观察适配前 (Pre-fitting) 和适配后 (Post-fitting) 的输入输出、管脚锁定、宏单元结构等信息。适配前信息来源于 .ngd 文件，适配后信息来源于 .vm6 文件。
- FPGA 底层编辑器读取 FPGA 的布线信息 (.ncd) 文件，用图形化界面显示

FPGA 内部的 CLB 和 IOB 模块结构，根据用户的设置与修改生成 Xilinx 物理约束文件（PCF 文件）。使用 FPGA Editor 可以完成如下功能：在自动布线前，手工布置关键路径，提高电路工作频率；帮助布线器完成自动布线难以实现的路径；在 FPGA 内部的任何一个节点设置探针，拉出待测信号到 IO 端口，这种方法显然要比在 HDL 源代码中逐层用语言描述，将待测信号拉到 IO 端口的的方法更灵活；改变内部在线逻辑分析仪（ILA）的连线 and 配置；FPGA Editor 的高级用户甚至可以手动添加和连接一个个内部元件（CLB、IOB）以完成电路设计与实现。

- 布局规划器与 FPGA Editor 相似，也能改变 FPGA 内部的 CLB 和 IOB 的连接配置情况。通过交互图形界面，用户可以观察到 FPGA 内的连接情况，并且手动进行物理位置约束。它比 FPGA Editor 更灵活，可以在实现过程的映射（Mapping）前、映射后、布局布线（Place & Route）等不同阶段约束设计，发挥功能。它可以导入 NGD、NCD、FNF 和 UCF 等格式的文件，根据用户需要，生成 UCF 和 MFP 等约束关系。对 Floorplanner 善加利用，可以有效提高设计的工作效率。

5. 辅助设计工具

ISE 还集成了许多辅助设计工具，主要工具如下：

- PROM 配置文件分割器（PROM File Formatter）可以完成配置文件的分割。有时芯片的配置文件（.bit 文件）要下载到外置存储器中（一般为 EEPROM），使系统在掉电后配置文件也不会丢失，当芯片加电后，自动从存储器中加载配置文件，重新配置芯片内部结构，开始工作。PROM 配置文件分割器的主要功能有 3 个：一是将 Xilinx 的配置文件（.bit 文件）转换成外置存储器能识别的格式；二是当 FPGA/CPLD 菊花链连接时，将每个芯片的配置文件组合起来并重新分割；三是在 Xilinx FPGA 多重配置时，将不同应用的配置文件合为一个配置文件。PROM 配置文件分割器支持的输入文件格式有 4 种：Intel MCS-86 文件格式（扩展名为.mcs）、Tektronix TEKHEX 文件格式（扩展名为.tek）、Motorola EXORmacs 文件格式（扩展名为.exo）和 HEX 文件格式（扩展名为.hex）。ISE 5.x 中将该工具集成到 iMPACT 中，使 FPGA/CPLD 配置文件的分割和下载更符合用户的操作习惯。
- iMPACT 配置器可以实现将配置文件下载到 FPGA/CPLD 或相应的存储器等功能。它的主要功能是下载、回读与校验配置数据，调试配置过程中出现的问题，生成 SVF 和 STAPL 文件。ISE 中 iMPACT 与 Foundation 系列较低版本的配置器相比有了很大的改进与提高，它的边界扫描、芯片检查、下载功能越来越完善。
- 功耗仿真器（XPower）是估计设计功耗的工具。当整个设计实现过程完成后，调用功耗仿真器。功耗仿真器可以根据设计所使用的门的数量，驱动电压、电流的大小，环境温度等估算芯片的结温、静态功耗、逻辑模块功耗、时钟功耗、输出功耗和总功耗等信息，帮助 PCB 系统设计人员设计系统。功耗

仿真器可以分析 Xilinx 的 FPGA 和其 CoolRunner 系列 CPLD 的功耗。

- 在线逻辑分析仪 (Chip Scope Pro) 是 Xilinx 与 Agilent 合作开发的针对 Xilinx Virtex-II Pro/Virtex/Virtex-II/Virtex-E/Virtex-EM/Spartan-IIE/Spartan-IIE 系列 FPGA 的在线片内信号分析工具, 它的主要功能是通过 JTAG 口, 在线、实时地读出 FPGA 的内部信号。ChipScope Pro 的基本原理根据用户设定的触发条件将信号实时地保存到 FPGA 中未使用到的块 RAM 中, 然后通过 JTAG 口将信号传送到计算机, 并在计算机屏幕上显示出时序波形。
- 模块化设计 (Modular Design) 是一种进行并行工作、协同设计的工作方法和设计工具。其最显著优势有两个, 一是协同设计, 即所有设计小组成员可以在最大程度上互不干扰地设计自己的子模块, 从而加速了项目进度; 二是在调试、更改某个有缺陷的子模块时, 并不会影响到其他模块的实现结果, 从而保证了设计的稳定性与可靠性。
- 增量式设计 (Incremental Design) 方法是一种能在小范围改动情况下节约综合、实现时间并继承以往设计成果的设计手段。合理的运用增量设计能带来两方面优点: 一是减少综合、实现过程 (特别是布局布线过程) 的耗时; 二是能够继承未修改区域的实现成果, 这里的实现成果主要指在时序和面积两个方面。

1.8 常用专有名词解释

为了便于学习, 我们将本书常用的一些专有名词和约定集中在这里加以解释。

- Architecture Wizard: 即结构向导, 是 ISE 5.x 的新增工具, 用于辅助设计数字时钟管理模块 (DCM, Digital Clock Manager) 和高速 I/O 收发器 (Rocket I/O™ transceivers)。
- Back - Annotation: 即反标, 这是一个在后仿真中常用的概念, 指将门延时、布线延时等插入到原设计中, 使后仿真结果更精确。
- Buffer: 即缓冲, 指增强信号驱动能力的硬件单元, 通常用于驱动大扇出信号、I/O 端口信号、存储器输出信号等。
- Chip Scope Pro: 即在线逻辑分析仪, 指 Xilinx 与 Agilent 合作开发的 FPGA 测试工具。它在 FPGA 内部创建存储、触发、控制逻辑, 通过 JTAG 口在线、实时地读出 FPGA 内部的任何信号, 实现 FPGA 片内信号的在线分析。
- CLB: 即可配置逻辑模块 (Configurable Logic Block), 是 Xilinx FPGA 的基本硬件单元。
- Constraints Editor: 即约束编辑器, 是 ISE 用户约束文件 (UCF 文件) 的重要设计工具之一。
- Core Generator: 即 IP 核生成器, 是 Xilinx FPGA 设计中的一个重要设计输入工具, 提供了大量 Xilinx 和第三方公司设计的成熟、高效 IP 核为用户所用。
- Chip Viewer: 即芯片观察窗, 以图形化界面观察 FPGA/CPLD 适配结果的工具。

- Design Manager：即设计管理器，这个工具是 ISE 5.x 以前版本中 FPGA/CPLD 实现与配置的集成环境。其功能相当于在 ISE 5.x 中完成 EDIF 设计流程，ISE 5.x 取消了该工具。
- E2PROM：即电擦除可编程只读存储器。
- EDIF：是逻辑网表格式的一个工业标准，为大多数综合、布局布线工具所支持。
- Floorplanner：即 Xilinx 布局规划器。通过交互图形界面，用户可以观察到 FPGA 内部连接情况，支持手动更改 CLB 和 IOB 的连接配置，进行物理位置约束。
- FPGA Editor：即 FPGA 底层编辑器，与布局规划器功能相似，以图形化界面显示 FPGA 内部的 CLB 和 IOB 模块结构，根据用户的设置与修改生成 Xilinx 物理约束文件（PCF 文件）指导 FPGA 布局布线。
- HDL Benchmer：即测试激励生成器，辅助用户自动生成设计的测试激励，并可以调用仿真工具，完成设计的验证。
- HDL：即硬件描述语言（Hardware Description Language），以代码形式描述硬件的连接关系，是当代大型 FPGA/CPLD 设计的首选设计方式。业界普遍使用的 HDL 语言是 Verilog 和 VHDL。
- HDL Editor：即 HDL 编辑器，ISE 中 HDL 语言的编辑器，内嵌语法辅助模板（Language Templates），能帮助用户更好地进行源代码设计。
- iMPACT：即 Xilinx 配置工具，自动检测 FPGA/CPLD 和配置 EEPROM 的硬件连接情况，实现 FPGA/CPLD 配置文件的分割、下载等系列功能。
- Implementation：即实现，指将综合输出的逻辑网表适配到具体 FPGA/CPLD 的过程，包括翻译（Translate）、映射（Map）和布局布线（Place & Route）等 3 个步骤。
- Instance：即实例，指在设计中对调用的声明。其动词形式为 Instantiation，即实例化。
- I/O：即输入/输出（Input/Output）。
- IP Core：即 IP 核，已经设计好且受知识产权保护的标准模块单元。
- LogiBLOX：与 IP 核生成器相似，可方便地生成大量成熟、高效的设计模块。
- PACE：即引脚与区域约束编辑器（Pinout and Area Constraints Editor），是 ISE 5.x 中增加的工具，完成 I/O 管脚和布局区域的约束，是约束编辑器的有益补充。
- PAR：即布局布线（Place And Route）。指将逻辑网表适配到 FPGA/CPLD 的底层硬件结构上的操作，其输出结果为 FPGA/CPLD 配置文件。
- Primitive：即硬件原语，它有两层含义：第一，硬件原语是 FGPA/CPLD 的基本逻辑单元，如门结构（与门 AND，或门 OR，异或门 XOR 等）和寄存器结构；第二，硬件原语不需要逻辑优化，直接被映射到 FPGA/CPLD 的底层硬件单元。
- Project Navigator：即工程管理器，它贯穿着 ISE 设计流程的始终，是 ISE 所

有集成工具的连接纽带。

- Schematic：即原理图，指用图形方式描述设计原理。
- StateCAD：即状态机编辑器，辅助进行状态机的设计与仿真的工具。
- Symbol：即器件符号，是模块单元的图形化代表形式。
- Synthesize：即综合，指将设计输入（HDL 代码、原理图等）编译并优化为由与、或、非门，RAM，寄存器等基本逻辑单元组成的逻辑连接。
- Timing Analyzer：即时序分析器，分析实现结果是否满足约束条件、芯片的工作速率以及关键路径等时序信息的工具。
- XPower：即功耗仿真器，是 Xilinx 估计设计功耗的工具。

1.9 小结

本章主要介绍了 FPGA/CPLD 的基础知识，FPGA/CPLD 开发的完整设计流程以及 ISE 集成的各种工具的名称、分类、功能与作用。

通过本章的学习，读者应该掌握 FPGA/CPLD 的基本原理、结构、特点与应用。从技术角度讲，FPGA/CPLD 能够实现任何数字器件的功能，上至高性能的微处理器，下至简单的 74 电路。而且现在 FPGA/CPLD 支持的接口、标准越来越多，可直接利用的 IP 核与高性能的基本模块覆盖到各个专业领域。EDA 软件易学易用，逐渐成为当代系统设计与硬件设计的首选方式之一。可以说 FPGA/CPLD 的出现使数字硬件设计工作变得轻松愉快。

一个完整的 FPGA/CPLD 开发设计流程应当包括电路设计与输入、功能仿真、综合、综合后仿真、实现、布线后仿真、下板调试等主要步骤。读者应在后续章节的学习中深刻体会这个设计开发过程。

ISE 的工具包分为综合工具、仿真工具、实现工具、辅助设计工具和其他工具等 5 类。本章依次介绍了这些工具的名称、功能与特点。ISE 功能强大，界面友好，易于掌握。希望读者能够结合后面各章的论述熟练掌握这些工具的使用技巧。

1.10 问题与思考

1. 简述 FPGA/CPLD 的原理、特点与应用。
2. 一个完整的设计流程包含哪些主要步骤，这些步骤的作用分别是什么？
3. 简述 ISE 集成工具的名称、分类、功能与作用。

第2章 工程管理器与设计输入工具

本章在介绍 ISE 工程管理器 (Project Navigator) 的基础上介绍多种设计输入工具及设计输入的方法。工程管理器贯穿 ISE 操作的始终,它连接 ISE 集成的所有工具。通过工程管理器的学习,读者能够了解 ISE 的用户界面和设计流程。本章将重点介绍 ISE 集成的 3 种常用的设计输入工具:HDL 编辑器 (HDL Editor)、状态机编辑器 (StateCAD) 和原理图编辑器 (ECS) 的使用方法。CORE Generator 提供了大量实用高效的设计模块的 IP 核。灵活运用 CORE Generator 不但能减少设计工作量,节约设计时间,在一定程度上还能提高设计的质量。HDL Benchner 是一个测试激励生成器,它能辅助设计者快捷地编写出测试激励。Architecture Wizard 是自 ISE 5.1 后新增的工具,用于辅助生成数字时钟管理模块 (DCM, Digital Clock Manager) 和高速 I/O 收发器 (Rocket I/O™ transceivers)。

本章主要内容如下:

- ISE 的工程管理器——Project Navigator;
- HDL 语言的输入工具——HDL Editor;
- 状态机输入工具——StateCAD;
- 原理图输入工具——ECS;
- IP 核的生成工具——CORE Generator;
- 测试激励生成器——HDL Benchner;
- 设计结构向导——Architecture Wizard。

2.1 ISE 工程管理器——Project Navigator

Project Navigator 是 ISE 的工程管理器,它是 ISE 所有集成工具的连接纽带。通过使用工程管理器,设计者可以创建、组织和管理自己的设计。

2.1.1 Project Navigator 综述

工程管理器的主要功能是创建一个用户工程,并按照第 1 章介绍的 FPGA/CPLD 设计流程调用相应的工具,完成从电路设计与输入、功能仿真、综合、综合后仿真、实现、布线后仿真、生成配置文件与配置 FPGA 等设计步骤。工程管理器支持第 1 章介绍的所有工具,这些工具的基本功能如表 2-1 所示。

表 2-1 工程管理器支持的工具及其功能

工具名称	工具功能	对应 FPGA/CPLD 设计流程的步骤
HDL 编辑器（HDL Editor）	HDL 源代码的生成与编辑	设计输入
状态机编辑器（StateCAD）	辅助状态机设计	
原理图编辑器（ECS）	完成原理图的设计与输入	
LogiBLOX	模块复用工具	
IP 核生成器（CORE Generator）	IP 核生成工具	
Xilinx 综合工具（XST）	综合工具	综合
FPGA Express	综合工具	
FPGA Compiler II		
Synplify & Synplify Pro	综合工具	
LeonardoSpectrum	综合工具	
测试激励生成器（HDL Bencher）	辅助测试激励的设计	仿真（包括功能仿真、综合后仿真、布线后仿真）
ModelSim XE/PE/SE	仿真工具	
约束编辑器（Constraints Editor）	编辑用于指导实现步骤的用户约束文件（.ucf）	实现
引脚与区域约束编辑器（PACE）	编辑与 I/O 管脚与面积约束相关的用户约束文件	
FPGA 底层编辑器（FPGA Editor）	手工布局布线	
时序分析器（Timing Analyzer）	实现结果时序分析器	
布局规划器（Floorplanner）	底层物理布线	
芯片观察器（ChipViewer）	观察设计在 FPGA 上的实现结果	生成配置文件与配置 FPGA/CPLD
功耗仿真器（XPower）	分析不同工作条件下 FPGA 的功耗	
配置文件分割器（PROM File Formatter）	完成配置文件的分割	
iMPACT	使用配置文件配置 FPGA/CPLD	

注：每个工具的功能与特点详见第 1.7 节。

2.1.2 Project Navigator 的用户界面

Project Navigator 的主界面由标题栏、菜单栏、工具栏、资源管理窗、当前资源操作窗、工程工作区、HDL 编辑器工作区、信息显示窗和状态栏等部分组成，如图 2-1 所示。

下面简要介绍各个部分的作用和使用方法。

1. 标题栏 (Title Bar)

标题栏显示当前工程的路径和程序的名称。

2. 菜单栏 (Menu Bar)

菜单栏主要由文件 (File)、编辑 (Edit)、视图 (View)、工程 (Project)、资源文件 (Source)、操作 (Process)、窗口 (Window) 和帮助 (Help) 等 8 个下拉菜单组成。

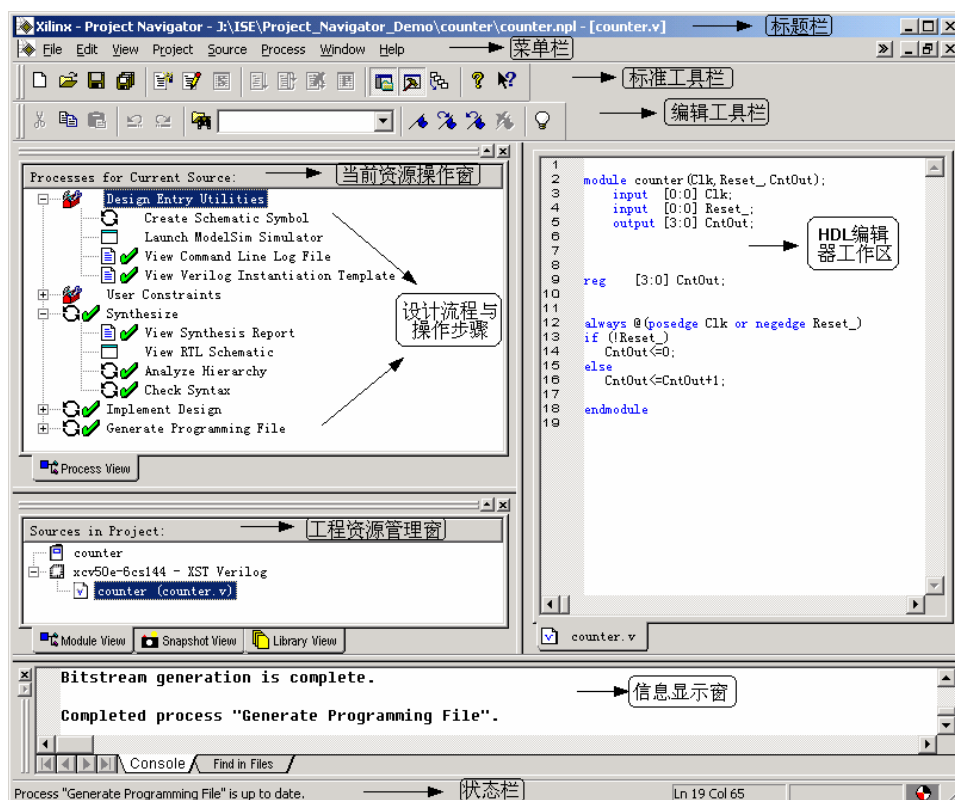


图2-1 Project Navigator 的主界面

- 【File】菜单的主要功能是新建、打开和保存一个工程或者资源文件。
- 【Edit】菜单主要包含了一些与文本编辑相关的功能选项。
- 【View】菜单的主要功能是隐藏或者显示某个视图。
- 【Project】菜单的主要功能是对工程的一些操作。如添加或新建某种资源文件，删除工程实现数据，压缩保存工程等。
- 【Source】菜单的主要功能是对资源文件的相关操作。这里的资源文件主要指源代码文件、工程说明文件、CORE Generator 生成的 IP 核、原理图、状态转移图、测试激励波形和块 RAM 映射文件（BMM 文件）等。
- 【Process】菜单包含了一些对当前资源操作的命令，如运行（Run）、重运行（Rerun）、所有步骤重运行（Rerun All）、停止（Stop）、视图（View）和属性（Properties）等。
- 【Window】菜单的主要功能是排列规划所有窗口，使窗口易读易管理。通过这个菜单可以看到当前打开的所有窗口的名称，并能直接切换到某个打开的窗口。
- 【Help】菜单包含了 ISE 和 Project Navigator 的帮助信息。使用 **[F1]** 键可以打开 Project Navigator 的在线帮助窗口。

3. 工具栏（Tool Bar）

工具栏中包含了常用命令的快捷按钮。Project Navigator 的工具栏分为标准工具栏和编

编辑器工具栏。灵活运用工具栏，可以极大地方便用户在 Project Navigator 中的操作。工具栏在工程管理中运用得较为频繁，下面介绍工具栏中常用按钮的功能，对于那些通用工具按钮，这里不再赘述。

标准工具栏包含了文件操作、资源操作、步骤操作、窗口操作和帮助操作等方面的常用命令按钮。

- 按钮 (New Source) 的功能是产生一个新的资源文件，单击该按钮会弹出如图 2-2 所示的对话框。

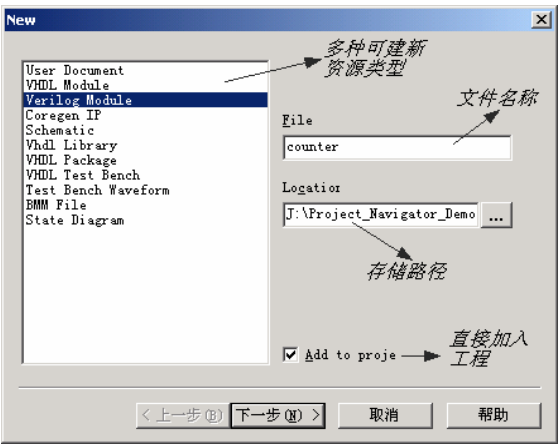


图2-2 新建资源对话框

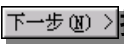
在该对话框的左边列出了所有可以新建的资源类型：工程说明文件（User Document）、源代码文件（Verilog Module）、CORE Generator 生成的 IP 核（Coregen IP）、原理图（Schematic）、测试激励波形（Test Bench Waveform）、块 RAM 映射文件（BMM 文件）、状态转移图（State Diagram）等。选择所要生成的文件类型，在文件（File）栏中填写文件名称，在位置（Location）栏中填写存储路径。勾选加入工程（Add to Project）选项，单击按钮，程序自动根据所选资源类型调用相应的生成工具。不同资源类型对应的处理工具如表 2-2 所示。

表 2-2 不同资源类型对应处理工具表

资源类型	ISE 中对应的处理工具
工程说明文件（User Document）	文本编辑器
VHDL 源代码文件（VHDL Module）	HDL 语言编辑器（HDL Editor）
VHDL 库（VHDL Library）	HDL 语言编辑器（HDL Editor）
VHDL 包（VHDL Package）	HDL 语言编辑器（HDL Editor）
VHDL 测试激励（VHDL Test Bench）	HDL 语言编辑器（HDL Editor）
Verilog 源代码文件（Verilog Module）	HDL 语言编辑器（HDL Editor）
CORE Generator 生成的 IP 核（Coregen IP）	IP 核的生成工具（CORE Generator）

续表

资源类型	ISE 中对应的处理工具
原理图 (Schematic)	原理图编辑器 (ECS)
测试激励波形 (Test Bench Waveform)	测试激励生成器 (HDL Bench)
块 RAM 映射文件 (BMM 文件)	块 RAM 生成工具 (Data2BRAM)
状态转移图 (State Diagram)	状态机编辑器 (StateCAD)
用户约束文件 (UCF)	约束编辑器 (Constraints Editor)
	管脚与区域约束编辑器 (PACE)

- 按钮  (Open Source) 的功能是打开选中的资源文件。在工程资源管理窗中选择某个资源文件，单击该按钮就可以直接编辑、修改和处理资源文件。
- 按钮  (Edit Project Properties) 的功能是编辑工程的属性。在资源管理窗选择工程，单击该按钮就产生图 2-3 所示的工程属性对话框，在该对话框中可以更改工程的属性。

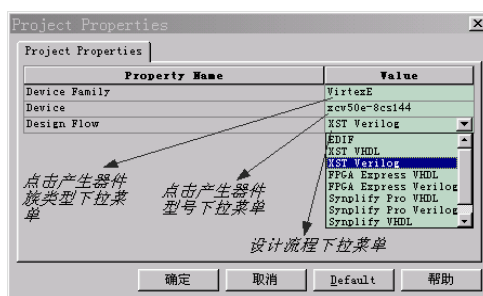


图2-3 工程属性对话框

工程属性分为 3 项：器件族类型 (Device Family)、器件型号 (Device) 和设计流程 (Design Flow)。每个项目都隐含一个下拉列表，单击属性值 (Value) 域，在相应的下拉列表可以更改属性。器件族类型指 Xilinx 根据 FPGA/CPLD 架构区分的功能类型，详细说明见 1.4 节。同一族中的器件又根据封装、容量和速度的不同划分为不同的型号。设计流程比较重要，这个属性在工程建立时指定，在工程属性对话框修改。设计流程根据工程输入类型大致分为两类：第一类的工程输入为 EDIF 网表，设计者使用外部综合工具综合设计，生成网表，选用 EDIF 设计流程在 ISE 中直接进行实现、配置等操作，这时 ISE 相当于 Xilinx 早期设计工具 Design Manager 的功能；第二类的工程输入为源代码，设计者根据 HDL 源代码类型 (VHDL、Verilog HDL) 和需要选用的综合工具选择相应的设计流程，完成从设计输入、综合、实现到配置的完整 FPGA/CPLD 设计流程。



不同的器件类型和设计流程决定了在 Project Navigator 中可以新建和使用的资源的种类。如 Architecture Wizard 生成的数字时钟管理 (DCM) 和高速 I/O Rocket 收发器模块仅能在 Virtex II 和 Virtex-II Pro 器件族使用。

- 按钮  (Run Process) 的功能是运行当前操作。选中当前资源操作窗 (Processes for Current Source Window) 的某一操作, 单击  按钮, 就运行相应的操作。
- 按钮  (Rerun Process) 的功能是重新运行当前操作。选中当前资源操作窗的某一操作, 单击  按钮, 就重新运行该操作。所有过去的运行结果将被覆盖。
- 按钮  (Stop Process) 的功能是停止当前操作。选中当前资源操作窗的某一操作, 单击  按钮, 强迫停止操作。
- 按钮  (Edit Process Properties) 的功能是编辑当前操作的属性。选中当前资源操作窗的某一操作, 单击  按钮, 就会产生相应的操作属性对话框。
- 按钮  (Toggle Workspace Window) 的功能是显示或者隐藏工程工作区, 即显示或者隐藏资源管理窗和当前资源操作窗。
- 按钮  (Toggle Transcript) 的功能是显示或者隐藏信息显示窗。
- 按钮  (Minimize All) 的功能是最小化与当前工程相关的所有窗口。

编辑器工具栏涵盖了 HDL 编辑器中的所有命令按钮。

- 按钮  (Find in Open Files) 的功能是在当前所有 HDL 编辑器工作区打开的文件中查找字符串。
- 搜索条  的功能是在当前文件中查找字符串。
- 按钮  (Toggle Bookmark) 的功能是在 HDL 编辑器当前文件中的光标所在行加入或者删除书签。
- 按钮  (Next Bookmark) 的功能是将光标移动到下一个书签处。
- 按钮  (Previous Bookmark) 的功能是将光标移动到前一个书签处。
- 按钮  (Clear All Bookmarks) 的功能是清除当前文件的所有书签。
- 按钮  (Language Templates) 的功能是打开语言辅助模板。关于语言辅助模板的使用在 2.2.3 节将详细介绍。

4. 资源管理窗 (Sources in Project Window)

资源管理窗用于显示当前工程中所有相关的资源文件, 如图 2-4 所示。

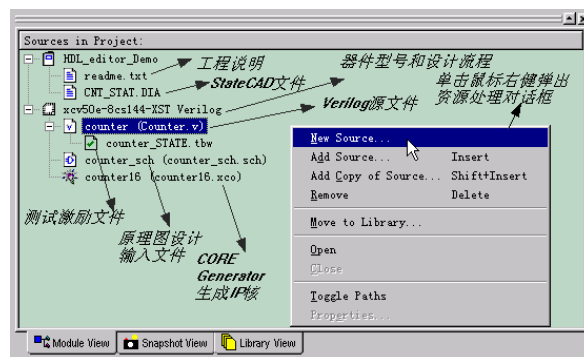


图2-4 资源管理窗

资源文件的含义在前面介绍工具栏时已经讲过。图 2-4 所示示例工程中包含的资源文件

有工程说明文件 readme.txt，状态机输入文件 CNT_STAT.DIA，Verilog HDL 源代码 counter.v，测试激励文件 counter_STATE.tbw，原理图输入文件 counter_sch.sch，CORE Generator 生成的 IP 核 counter16.xco。在资源管理窗中还可以看到此工程选用的器件类型为 xcv50e-8cs144，设计流程为 XST Verilog。

资源文件的操作可以通过 3 种途径完成：利用工具栏中相关的资源操作按钮；使用菜单栏的【Source】菜单；利用鼠标右键弹出菜单中的相应命令。前两种操作方法已经讨论过。第三种操作方法为：在资源管理窗选中需要操作的资源文件，单击鼠标右键，弹出菜单，然后选择相应的操作命令即可。细心的读者会发现，单击鼠标右键，弹出的菜单的功能恰好包括了前面两种操作方法的所有功能。



ISE 中使用鼠标右键弹出菜单的操作方法很具代表性，不仅在资源管理窗（Sources in Project Window）中非常有效，在当前资源操作窗（Processes for Current Source Window）中同样适用。

资源管理窗左下角有 3 个标签，分别对应模块视图（Module View）、快照视图（Snapshot View）和库视图（Library View）。模块视图最为常用，它层次化显示了工程中的所有资源的组织结构。快照视图显示工程中的快照，所谓快照是指对工作目录和远程资源的一个只读记录，产生快照的方法如图 2-5 所示。库视图显示了工程中用户产生的库的内容。

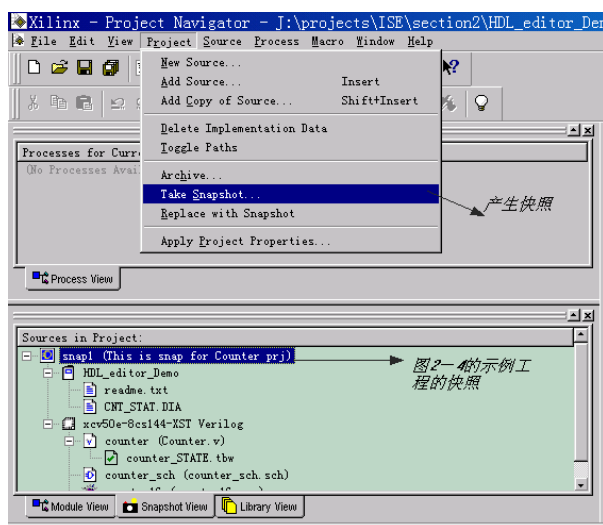


图2-5 产生快照的方法和快照视图

5. 当前资源操作窗（Processes for Current Source Window）

当前资源操作窗（Processes for Current Source）在 ISE 6 及以上版本改名为资源操作窗（Processes for Source），该窗口用以显示资源管理窗中高亮选中的资源文件的所有内容和相关操作。相关操作和 FPGA/CPLD 设计流程紧密联系，包括设计输入、综合、仿真、实现和生成配置文件等。不同类型的资源、不同的设计流程或是不同类型的器件型号可以进行的相关操作都不一样。当对某资源文件以选定设计流程在当前器件上完成相关操作后，在每个操作步骤前会显示一个小图标，表示该步骤的完成情况。图标示意如下：

-  图标表示该操作步骤成功完成；
-  图标表示该操作步骤虽完成但有警告信息；
-  图标表示该操作步骤因错误而未完成。

对当前资源文件的操作同样可以通过 3 种途径完成：利用工具栏中相关的操作按钮；利用菜单栏中【Process】菜单以及当前资源操作窗中选中操作步骤；单击鼠标右键，选择弹出菜单中的相关操作命令。鼠标右键弹出菜单的功能和前面两种操作方法一致，如图 2-6 所示。

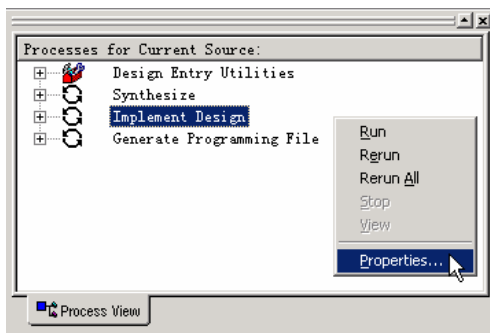


图2-6 功能菜单

6. 工程工作区 (Project Workspace)

工程工作区是资源管理窗和当前资源操作窗的统称。

7. HDL 编辑器工作区 (HDL Editor Workspace)

HDL 编辑器工作区是 Project Navigator 中比较重要的窗口。所有 HDL 源文件、报告文件和工程说明文件等文本文件都在这个工作区内被编辑。这个工作区和 HDL 编辑器窗口在 2.2 节将详细地介绍。

8. 信息显示窗 (Transcript Window)

信息显示窗显示 Project Navigator 中的处理信息，如操作步骤信息、警告信息和错误信息等。信息显示窗左下脚有两个标签，分别对应控制台 (Console) 信息窗和文件查找 (Find in Files) 窗。控制台信息窗的交叉链接功能使调试变得更为方便，如果设计出现了警告或错误，双击信息显示窗口的警告或者错误标志，就能自动切换到源代码出错的地方，帮助设计者查找代码中的设计缺陷。文件查找窗显示在 HDL 编辑器工作区打开的所有文件中查找某个字符串的结果。

9. 状态栏 (Status Bar)

状态栏显示相关命令或者操作的信息。



把鼠标箭头移动到某个操作步骤、菜单选项或者工具栏按钮时，就会在状态栏显示简短的功能提示信息。

2.1.3 使用 Project Navigator 创建并管理工程

通过前面的介绍，读者应该对 Project Navigator 的界面和各个窗口、菜单、工具栏、按

钮的功能和使用都有了初步了解，但是对如何使用 Project Navigator 创建、管理工程，完成 ISE 设计流程仍然没有一个整体上的把握。下面通过一个简单的例子带领读者创建一个工程，并且依次完成由设计输入、功能仿真、综合、综合后仿真、实现、布线后仿真、生成配置文件与配置 FPGA 等步骤组成的完整设计流程。

🔑 同步 4bit 加法计数器在 Project Navigator 的完整设计流程，参考示例详见本书附带光盘的“Example-2-1”目录

1. 创建工程。

首先创建一个工程目录，目录名称为“Project_Navigator_Demo”。启动 ISE 工程管理器，每次启动 Project Navigator 时都会默认恢复到最近使用过的工程界面，由于此时没有前面的工程记录，所以出现了空白工程界面。选择【File】/【New Project】选项，弹出新建工程对话框，在工程名称中输入“counter”作为工程名。在工程路径中单击  按钮，将工程指定到目录“Project_Navigator_Demo”。器件族类型（Device Family）选“VirtexE”，器件型号（Device）选“xcv50e cs144 -6”，设计流程（Design Flow）为“XST Verilog”。如前面介绍，不同的设计流程对应的输入资源类型、综合工具和相关操作都不同，如图 2-7 所示。如果因为误操作或者需要更改器件型号与设计流程，可以通过单击工具栏中的  按钮，编辑工程的属性。

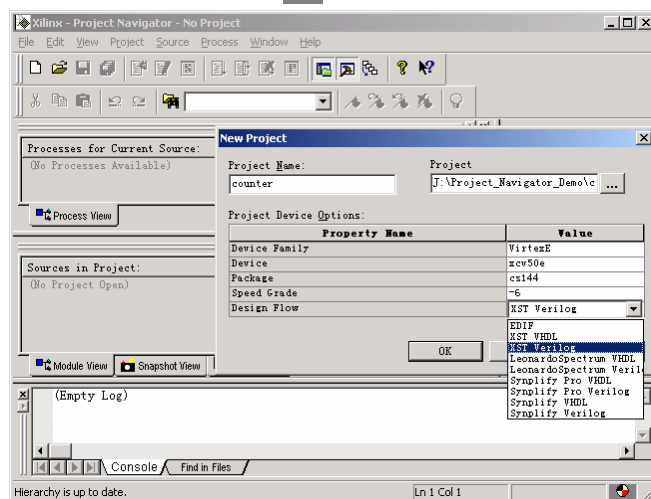


图2-7 创建一个工程

2. 设计输入。

大型设计中一般都采用 HDL 语言设计方法，所以这里就采用最为流行的 HDL 语言——Verilog HDL 作为设计输入语言。这就与创建工程时所选择的设计流程为“XST Verilog”一致了。单击  按钮产生一个新的资源文件，资源类型为 Verilog 源代码，文件名称为 counter，存储到工程目录下，选中“Add to Project”直接将资源加入到工程中去，如图 2-8 所示。

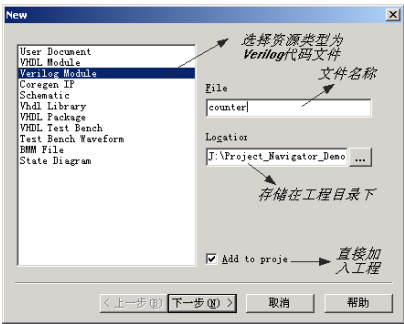


图2-8 新建资源文件

单击 **下一步(N) >** 按钮，程序自动根据所选资源类型调用相应生成工具。由于所选资源类型为 Verilog 源代码，所以 Project Navigator 调用 Verilog 源代码辅助模板。我们设计一个最简单的程序，“4bit 计数器”。输入为时钟信号“Clk”，异步复位信号“Reset_”，输出为计数结果“CntOut”。在 Verilog 源代码辅助模板中填入这些信号，如图 2-9 所示。



图2-9 Verilog 源代码辅助模板

单击 **下一步(N) >** 按钮，确认资源信息，然后单击 **完成** 按钮，Project Navigator 自动调用 HDL 编辑器，并根据源代码辅助模板信息，生成程序头。在 HDL 编辑器中编写代码，单击  按钮，结果如图 2-10 所示。HDL 编辑器的详细介绍参见下节。

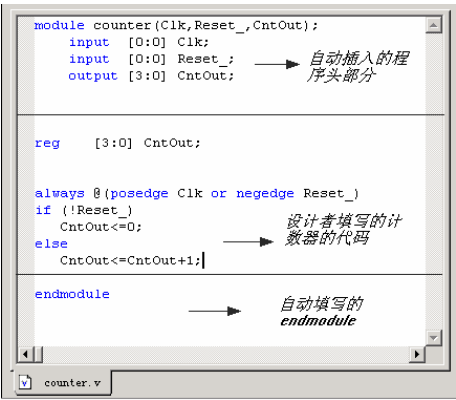


图2-10 源代码的输入

3. 功能仿真。

功能仿真有时也被称为前仿真，它的主要目的是验证设计的逻辑功能是否正确，功能仿真不包含任何门延迟和路径延迟信息，是独立于器件信息之外的仿真，一般调用专用仿真工具进行仿真，常用仿真工具参见 1.2 节。ISE 中一般使用 ModelSim 进行仿真，ModelSim 的使用详见第 3 章。

4. 综合。

综合的主要功能是将 HDL 语言翻译成最基本的与、或、非门，RAM，触发器等基本逻辑单元的连接关系（网表），并根据要求（约束条件）优化所生成的门级逻辑连接，输出 edf 和 edn 等文件，供 CPLD/FPGA 厂商的软件进行实现。第 4 章将会对综合过程详细介绍，这里仅仅联系前面介绍的功能按钮、菜单与窗口给读者一个综合过程的初步概念。当设置详细的综合优化、约束条件时，应当独立调用综合工具。如果综合优化、约束条件比较简单，可以在综合属性中直接设置。编辑综合属性的途径有 3 种：在当前资源操作窗中选中综合步骤（Synthesize），然后单击工具栏中的 （Edit Process Properties）按钮；第二种方法是选择菜单栏中【Process】菜单中的相应选项；第三种方法是在当前资源操作窗中单击鼠标右键，在弹出的菜单中选择【Properties】选项。根据所选设计流程的输入和综合工具的类型不同，综合属性的设置略有不同。图 2-11 示例了本工程中 XST Verilog 设计流程的综合属性。

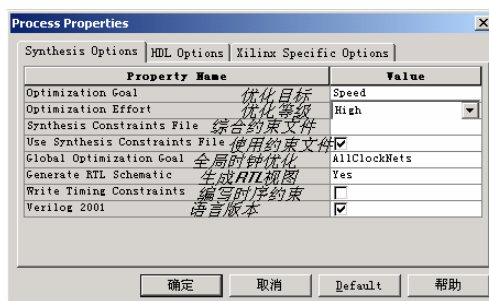


图2-11 XST Verilog 设计流程的综合属性设置

确认设置后，单击 （Run Process）按钮进行综合，综合过程的步骤信息与警告等会在信息显示窗显示。综合过程产生的报告可以在 HDL 编辑工作区打开查看。

5. 综合后仿真。

进行综合后仿真的目的是检查综合结果是否与原设计思路一致。仿真时需要把综合生成的时延文件反标到综合仿真文件中。



综合后仿真不包含布线的路径延迟信息，并不能真实反映设计在器件中的实际时序情况，像这类仅包含部分信息的仿真还有翻译网表后仿真（Post-Translate Simulation）和映射后仿真（Post-Map Simulation）。

6. 实现。

实现（Implement）是将设计的逻辑网表信息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。从上面定义可以看出，实现主要分为 3 个步骤：翻译逻辑网表（Translate），映射到器件单元（Map）与布局布线（Place & Route）。其实 ISE 的实现流程也恰恰是这 3 个步骤。实现前应该设计实现约束条件，约束条件一般包括管脚锁定、时钟约束、全局时钟、第二全局时钟（全局时钟和第二全局时钟资源的详细论述请参考 4.4 节）、分组约束和物理特性约束等信息。ISE 中可以使用约束编辑器（Constraints Editor）生成约束文件（UCF）。约束文件和其他实现信息的设置，都可以通过编辑实现属性对话框的设置完成。同样有 3 种途径编辑实现属性对话框：首先在当前资源操作窗中选中实现步骤（Implement Design），第一种方法是单击工具栏中的 （Edit Process Properties）按钮；第二种方法是选择菜单栏中【Process】菜单的【Properties】命令；最后一种方法是在当前资源操作窗中单击鼠标右键，在弹出的菜单中选择【Properties】命令。图 2-12 所示为实现属性对话框。

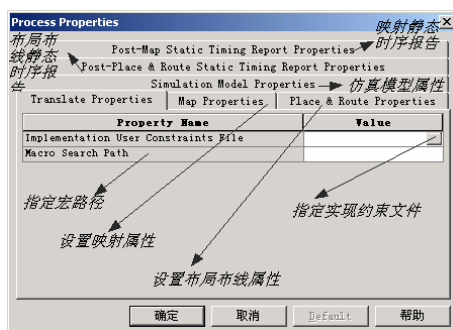


图2-12 实现属性对话框

从图 2-12 可以看到，实现属性对话框包括翻译属性（Translate Properties）、映射属性（Map Properties）、布局布线属性（Place & Route Properties）、仿真属性（Simulation Model Properties）、映射静态时序报告属性（Post-Map Static Timing Report Properties）和布局布线静态时序报告属性（Post-Place & Route Static Timing Report Properties）等 6 个选项卡。前 3 种属性设置对应 ISE 中实现的 3 个步骤，包含了实现的约束设置。仿真属性主要是对生成后仿真模型而言的，通过仿真属性的设置，使 ISE 更好地与仿真工具结合起来。最后两种时序分析报告的设置帮助设计者更好地分析设计的时序，检查设计能否满足时序约束条件。



实现过程的用户约束文件（UCF）非常重要，几乎所有时间、管脚、位置等约束条件都可以通过 UCF 文件实现。

设置好实现属性后，单击  按钮进行实现，实现过程的一切信息会在信息显示窗显示。实现过程产生的报告可以在 HDL 编辑工作区打开查看。

7. 布线后仿真。

该仿真运用布局布线后的仿真模型，反标器件的延迟信息，能够较为准确地模拟设计在器件上的实际运行情况。布线后仿真的延迟信息不仅仅包括门延迟，而且包括了布局布线引起的路径延迟。详细介绍参见第 3 章。

8. 生成配置文件与配置 FPGA。

这是设计流程的最后一步，根据器件型号与存储器件型号将实现结构生成可以配置 FPGA/CPLD 的配置文件。在 ISE 中使用 PROM File Formatter 分割配置文件，用 iMPACT 配置器件。在当前资源操作窗选中生成配置文件（Generate Programming File）项，单击  按钮，编辑配置文件属性，如图 2-13 所示。该属性对话框包含 4 个选项卡，分别是通用设置（General Options）、配置设置（Configuration Options）、初始化设置（Startup Options）和回读设置（Readback Options），每项的详细说明参见第 7 章。

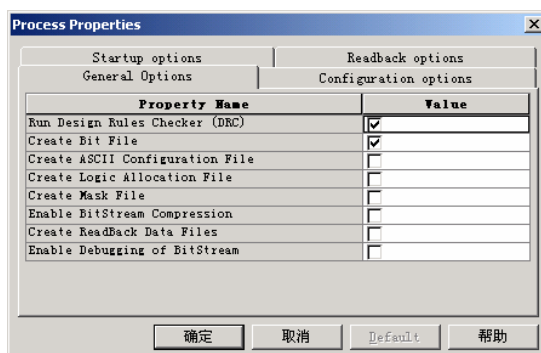


图2-13 编辑生成配置文件属性对话框

2.2 HDL 语言的输入工具——HDL Editor

本节将介绍 ISE 中集成的 HDL 语言输入工具——HDL Editor 的使用方法。重点介绍如何使用 Language Templates 辅助源代码的设计。

2.2.1 HDL Editor 综述

如第 1 章所介绍的，HDL 语言设计方法是当前大型工程中最常用的设计方法。目前较为流行的 HDL 语言有：VHDL、Verilog HDL 和 ABEL 语言等。它们共同的特点是易于使用由顶向下的设计方法，利于模块的划分与复用，可移植性好，通用性好，设计不因芯片的工艺与结构的变化而变化，更利于向 ASIC 的移植。其实 HDL 语言源代码文件是纯文本文件，使用任何文本编辑器都可以编辑 HDL 源代码。但是许多专用 HDL 编辑器集成了语法检查，语法辅助模板，代码调试等功能，这些功能给 HDL 源代码的设计与调试带来了极大的方便。

ISE 中集成的 HDL 编辑器叫做 HDL Editor。HDL Editor 基本涵盖了专用 HDL 编辑器的常用功能。值得一提的是它有一个 Language Templates 语法设计辅助模板。Language Templates 提供了 VHDL、Verilog HDL、ABEL 语言和 UCF 用户约束的语法说明和示例，并

能方便引入到设计者的当前文件中，大大方便了源代码设计。

使用 HDL 编辑器输入源代码的方法前面已经介绍过。需要注意的是，在 Project Navigator 中，HDL 编辑器的所有相关命令除了使用菜单栏的【Edit】菜单实现外，还可以使用编辑工具栏的快捷按钮实现。

为了帮助设计 HDL 源代码，HDL 编辑器采用了不同颜色区分语法结构，如表 2-3 所示。

表 2-3 语法结构与颜色对应表

语言 颜色	ABEL	VHDL	Verilog	UCF
绿色	注释	注释	注释	注释
蓝色	关键字	关键字	关键字	关键字，全局关键字
黄色	无	无	指示原语	无
粉红色	Dot 扩展，特殊常量，@指示	STD_LOGIC_1164	函数	无
黑色	其他	其他	其他	属性关键字

2.2.2 源代码输入的好助手——Language Templates

语言辅助模板 (Language Templates) 是 ISE 的一个特色工具。在语言模板中存放了很多 HDL 语言的通用语法结构和使用范例，特别是许多根据 Xilinx 器件 IP 核与硬件原语编写的实例化程序。使用语言模板，可以方便地把这些语法结构和范例插入到设计者自己的代码文件中，大大方便了程序的编写，提高了工作效率。语言模板按照 ABEL、COREGEN、UCF、Verilog 和 VHDL 的顺序存放在模板视窗中。其中 COREGEN 和 UCF 是从 ISE 5 版本后新增的辅助模板。COREGEN 模板由两个目录组成，分别存放当前工程中生成的 IP 核的 Verilog 和 VHDL 实例化文件，其内容与 IP 核生成器生成 IP 核时自动生成的实例化文件 (.veo、.vho) 相同。UCF 模板也分成两个目录，依次存放 CPLD 和 FPGA 的用户约束文件 (.ucf) 的约束范例。这个模板使手工编辑 UCF 文件更加容易。

ABEL、Verilog 和 VHDL 三大语言模板大致可以分为下列 4 个项目。

1. 器件实例化 (Component Instantiation)

该模板仅存在于 Verilog 和 VHDL 中，给出了块 RAM (Block RAM)、数字时钟延迟锁相环 (Clock DLL)、数字时钟管理单元 (DCM)、分布式 RAM/ROM (Distributed RAM/ROM)、全局时钟缓冲 (Global Clock Buffer)、查找表 (LUT)、基于查找表的移位寄存组 (SRLUT)、I/O 器件、乘法器和选择器 (MUX) 等器件模块的实例化范例。

2. 语法模板

该模板给出了基本语法规则和应用范例。

3. 综合模板

该模板给出了可综合实现的一些基本单元模块的范例，如乘法器、计数器和触发器等。

4. 用户自定义模板

该模板存放用户自己创建的特定结构，是语言模板的功能扩展。

单击  (Language Templates) 按钮，打开语言辅助模板，其界面如图 2-14 所示。

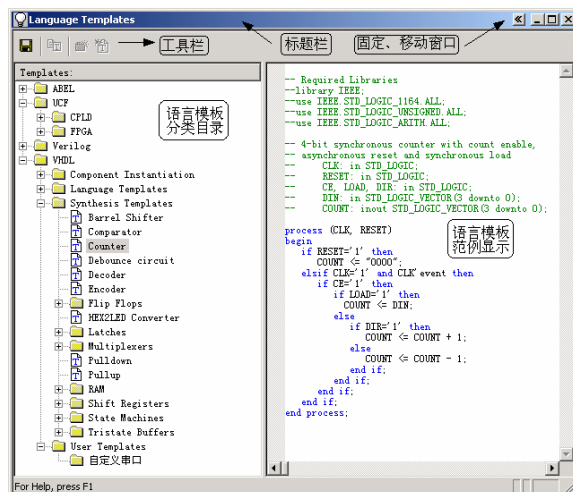


图2-14 语言辅助模板

语言辅助模板由标题栏、语言辅助工具栏、语言模板组织目录和语言模板范例等 4 部分构成。语言模板工具栏包含了所有使用语言模板的工具。语言模板组织目录在左边窗口，以目录结构显示前面介绍的 4 个模板类型。右边的范例显示区显示所选模板的代码。语言模板的工具栏非常简单，只有 4 个按钮，下面依次说明。

- 按钮  最为常用，其功能是将模板中显示的范例应用到当前的代码原文件中，也就是将范例全文插入到当前代码文件的光标后面。
- 按钮  的作用是保存新建的用户模板。
- 按钮  的作用是在用户自定义模板目录下面新建子目录，利于用户自定义模板目录的分类与管理。
- 按钮  的作用是产生一个新的用户模板文件。

选中语言辅助模板窗口，点击 Project Navigator 中的【Edit】菜单，会发现工具栏中所有工具在【Edit】菜单中都有相应的命令选项。

语言辅助模板的使用非常简单，只要根据设计找到所需的设计范例，单击  按钮，将范例插入到自己的源代码中，然后根据自己的信号定义修改范例即可。用户自定义模板作为语言辅助模板功能的扩充，使用户能更好地建立自己的代码模块资源，便于用户代码的维护。用户自定义模板的使用也非常方便，选中用户自定义模板目录 (User Templates)，单击  按钮在用户自定义模板目录下面新建功能子目录，然后单击  按钮新建空白模板文件，编辑空白文件，写入用户自定义的程序范例和注释等，单击  按钮保存。

2.3 状态机输入工具——StateCAD

本节将介绍 ISE 中集成的状态机输入工具——StateCAD 的使用方法。

2.3.1 StateCAD 综述

StateCAD 是 ISE 中集成的一个非常实用的工具。过去使用 HDL 语言简洁、完备、安全地描述状态机需要一定的功底，现在，StateCAD 使状态机的设计、优化和测试变得简单快捷。

StateCAD 的最大特点就是符合人们设计状态机的习惯。表述状态机最自然的方式就是用状态转移图，在 StateCAD 中，设计者只要画出状态转移图就可以了，StateCAD 自动将状态转移图翻译成 HDL 语言代码，而且翻译出来的代码规范、可读性好、可综合、易维护。StateCAD 能自动检测状态机的完备性和正确性，对状态转移图中的冗余状态、自锁状态、歧义转移条件和不完备状态机等隐含错误都会报警，并协助设计者更正错误。然后 StateCAD 根据设计者所选的优化方式，进一步优化代码。最后 StateCAD 自动生成设计的测试激励，并调用仿真程序，验证状态机的正确性，这个测试激励甚至可在后仿真中使用。总之，StateCAD 提供了状态机的输入、翻译、检测、优化和测试等一条龙的服务，使状态机的设计变得安全、可靠、快速、便捷。

StateCAD 支持的语言有 VHDL、Verilog 和 ABEL。StateCAD 不但能将状态机设计翻译成这些标准语言格式，而且可以根据选用器件类型（FPGA、CPLD 等）和综合器类型（Xilinx XST、FPGA Express 等）生成经过优化的可综合源代码。



状态机设计是 FPGA/CPLD 的一个重要设计内容，使用 StateCAD 进行状态机设计可以事半功倍。

2.3.2 StateCAD 的用户界面

StateCAD 是一个完全独立于 Project Navigator 的第三方工具。打开 StateCAD 的方法有两种：一是在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【StateCAD】命令，另一种方法是在 Project Navigator 中新建 StateCAD 类型的资源，如图 2-15 所示。需要说明一点，StateCAD 不支持长文件名，所以文件命名不要超过 8 个字符。

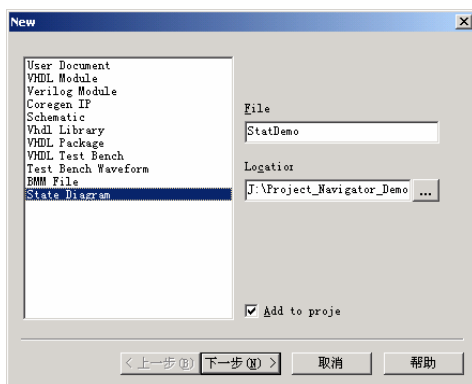


图2-15 在 Project Navigator 中新建 StateCAD 类型资源文件

单击 **下一步(N) >** 按钮，单击 **完成** 按钮确认新建资源信息，进入 StateCAD。图 2-16

所示为打开了一个 StateCAD 示例工程的用户界面。

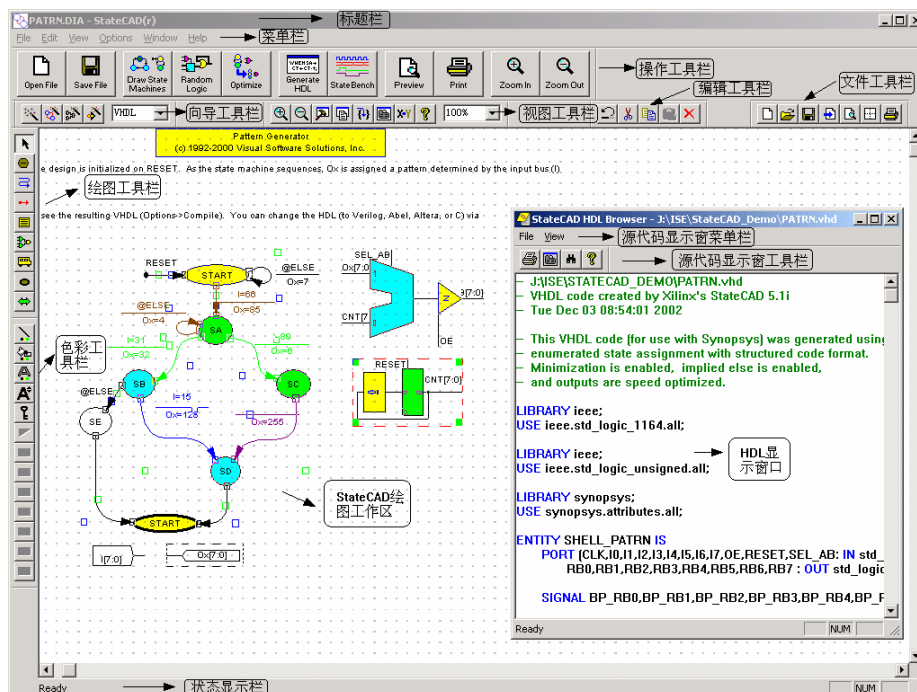


图2-16 StateCAD 的用户界面

StateCAD 的用户界面主要由绘图工作区、源代码显示窗、标题栏、菜单栏、工具栏和状态显示栏等部分组成。下面分别介绍各部分的功能。

1. 绘图工作区

绘图工作区用于绘制和编辑状态转移图。

2. 源代码显示窗

源代码显示窗用于显示生成的 HDL 源代码。

3. 标题栏

标题栏主要显示当前工程的名称。

4. 菜单栏

菜单栏由一系列下拉菜单组成，这些下拉菜单覆盖了 StateCAD 中的常用命令。

5. 状态指示栏

状态指示栏提示鼠标指向命令的操作信息。

6. 工具栏

StateCAD 的界面组织简单明了，操作简单，几乎所有的操作都可以利用工具栏完成。打开【Window】菜单，在所有工具栏选项前打勾，显示所有工具栏，如图 2-17 所示。

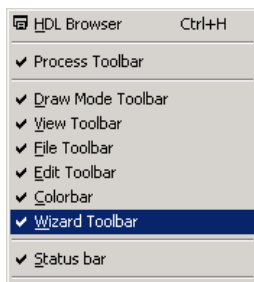


图2-17 设置【Window】菜单，显示所有工具栏

StateCAD 的工具栏主要由操作工具栏 (Process Toolbar)、绘图工具栏 (Draw Mode Toolbar)、视图工具栏 (View Toolbar)、文件工具栏 (File Toolbar)、编辑工具栏 (Edit Toolbar)、色彩工具栏 (Colorbar) 和向导工具栏 (Wizard Toolbar) 等组成。下面依次介绍各常用工具栏中的按钮及其功能。

操作工具栏是 StateCAD 中最常用，也是重要的工具栏。它贯穿着 StateCAD 始终，涵盖了所有主要操作步骤。按照从左到右的顺序依次使用操作工具栏的按钮，将引导设计者完成状态机的设计。操作工具栏如图 2-18 所示。



图2-18 操作工具栏

- 按钮  的功能是打开一个设计文档，StateCAD 文件的扩展名为 DIA。
- 按钮  的功能是启动状态机设计向导 (State Machine Wizard)，如图 2-19 所示。它的功能和向导工具栏中的  按钮的功能一致。状态机设计向导根据工程师所选的状态机转移图类型 (列状、双列状、行状和几何图形状转移图等)，复位模式 (同步、异步模式)，转移状态数目等生成一个基本的状态转移图模板。使用该模板能大幅度减轻绘制状态转移图的工作量。

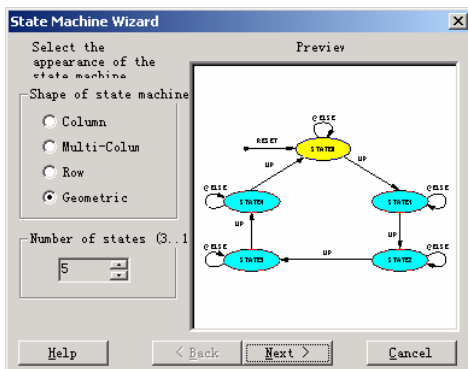


图2-19 启动状态机设计向导

- 按钮  的功能是启动逻辑向导 (Logic Wizard)。与之作用相同的按钮还有向导工具栏中的  按钮和绘图工具栏中的  按钮。当状态转移图的框架结构绘制完成后，启动逻辑向导辅助设计数据流的逻辑规则。逻辑向导与状态编辑器

(编辑摩尔型状态)，转移条件编辑器(编辑米勒型状态)和逻辑编辑器(编辑纯逻辑)密切联系。逻辑向导根据工程师所选择数据流逻辑规则和命名的信号名称自动生成逻辑表达式、端口信号定义和控制信号定义。逻辑向导支持计数器、选择器、移位寄存器、锁存器、乘法器等常用功能结构。逻辑向导的基本界面如图 2-20 所示。

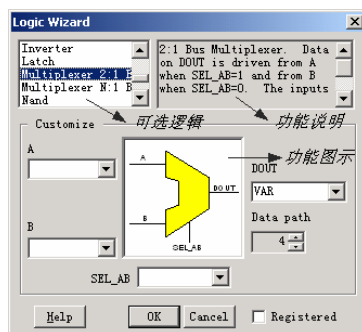


图2-20 逻辑向导界面

- 按钮  的功能是启动优化向导 (Optimization Wizard)，该按钮同向导工具栏中的  按钮功能相同。当状态转移图绘制完毕后，启动优化向导优化状态机，使整个设计在速度、面积、I/O 端口和综合结果等方面达到全面优化。使用优化向导时，工程师只需设置器件类型、优化目标、语言类型和综合工具类型等信息，优化向导自动完成所有的优化过程。图 2-21 所示为在优化向导中设置优化目标。

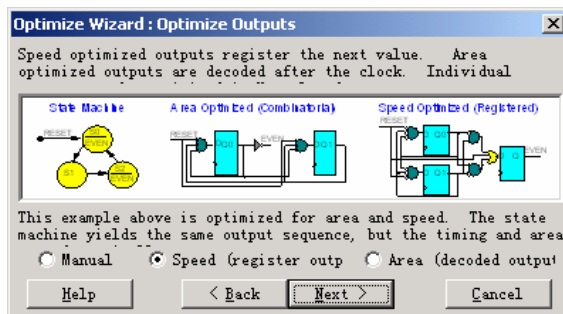


图2-21 优化向导中设置优化目标

- 按钮  的功能是编译状态转移图并生成 HDL 代码。该功能还可以通过单击菜单栏中的【Options】/【Compile】选项，或者单击视图工具栏中的  按钮来实现。StateCAD 在编译状态转移图并转换成 HDL 源代码前先对设计进行检错，检查语法和设计方面的问题。如果没有任何错误，编译成功，StateCAD 调用 HDL 浏览窗显示转换后的 HDL 源代码。如果发现错误，编译过程将停止，并弹出对话框显示错误信息，指出错误类型和可能的解决方式，如图 2-22 所示。并在状态转移图中出现错误的地方高亮显示。如果出现警告，并不影响编译的完成，仅仅在警告信息窗显示警告内容，引起设计者的注意。

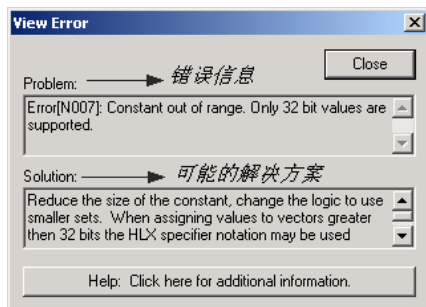


图2-22 编译错误信息窗

- 按钮  的功能是设计状态机的测试激励。该功能还可通过单击菜单栏中的【Options】/【Simulation】选项，或者单击视图工具栏中的  按钮来实现。当状态转移图正确编译并成功转换为 HDL 代码后，一般还要做功能仿真以验证设计逻辑的正确性，综合、布局布线后还要做后仿真以验证设计时序的正确性。通过调用测试激励生成器（StateBench）能方便地满足用户的仿真要求，设计出比较完备的测试激励来。图 2-23 所示为测试激励生成器窗口。

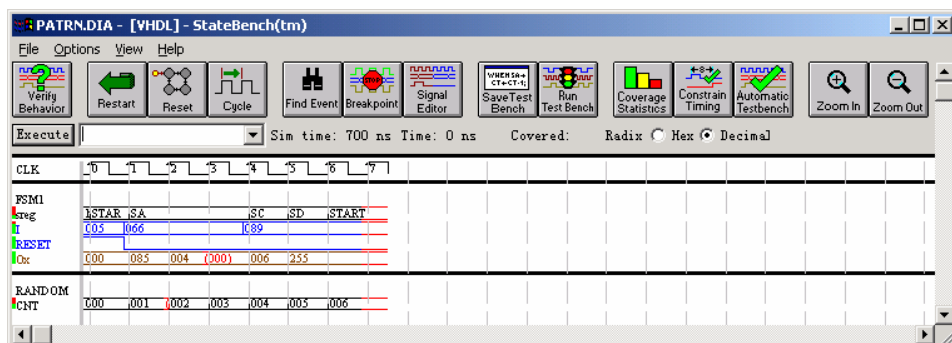
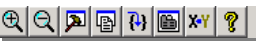
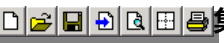


图2-23 测试激励生成器窗口

- 按钮  的功能是打印预览。
- 按钮  的功能是打印文件或图形。
- 按钮  的功能是放大视图。
- 按钮  的功能是缩小视图。

绘图工具栏  集合了所有绘制状态转移图的快捷按钮，这些按钮的基本功能依次为：选择目标、画转移状态、画转移连线、画复位信号、添加注释、启动逻辑向导、画向量、给状态起别称和绘制宏单元。

视图工具栏  集合了视图控制和与编译相关的一些选项的快捷按钮，这些按钮的基本功能依次为：放大视图、缩小视图、编译状态图并转换为 HDL 源代码、启动 HDL 浏览窗、启动测试激励生成器、编译配置选项、编辑信号、在线帮助和显示比例。

文件工具栏  集合了通用文件操作命令的快捷按钮，这些按钮的基本功能依次为：新建设计、打开设计、保存设计、引用设计、打印预览、分页设置和打印。

编辑工具栏集合了通用编辑操作命令的快捷按钮，这些按钮的基本功能依次为：撤销操作、剪切、拷贝、粘贴和删除。

色彩工具栏集合了所有与色彩相关的快捷按钮。给状态转移图适当地赋色，增加了状态转移图的可视性。前面 5 个按钮的基本功能依次为：边框颜色、填充颜色、文字颜色、字体大小和设为默认颜色。最后 8 个按钮依次为：默认颜色、紫、棕、绿、蓝、浅蓝、橙、黄等色彩按钮。

向导工具栏也非常有用。向导一步步引导设计者完成设计项目，使设计工作变得更加轻松。按钮的功能是启动状态机设计向导（State Machine Wizard），按钮的功能是启动逻辑向导（Logic Wizard），按钮的功能是启动优化向导（Optimization Wizard），这 3 个向导前面已经介绍过。下面介绍按钮，该按钮的功能是启动设计向导（Design Wizard）。设计向导引导工程师在 StateCAD 环境中完成整个状态机的设计过程。其功能实际上就是前面 3 个向导的综合，相当于依次启动设计向导、逻辑向导和优化向导，并且在设计优化完成后，调用编译和检错分析功能，帮助工程师调试自己的状态机设计。

2.3.3 使用 StateCAD 设计状态机

通过对 StateCAD 用户界面的介绍，读者对 StateCAD 的各个主要功能项目与工具栏已经有了一个整体上的认识。

下面通过设计一个串进并出的加法器，介绍如何使用 StateCAD 绘制状态转移图、调试错误、优化设计、翻译成 HDL 源代码，用测试激励生成器生成测试激励和仿真。大家在学习过程中一定要注意把握操作工具栏这条主线。

使用 StateCAD 设计串进并出的加法器状态机，参考示例详见本书附带光盘的“Example-2-2”目录

1. 启动 StateCAD，新建 StateCAD 文件。

启动 StateCAD 的方法有两种：一种是运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【StateCAD】命令。另一种是在 Project Navigator 中新建 StateCAD 类型的资源。

2. 利用状态机设计向导生成状态转移图模板。

前面已经反复强调过，使用 StateCAD 设计状态机的流程就是从左至右依次使用操作工具栏按钮的过程。下面单击按钮，启动状态机设计向导，生成状态转移图辅助模板。状态机设计向导第一个界面的功能是选择状态转移图的类型和状态的个数。状态转移图的类型分为列状转移图、多列状转移图、行状转移图和几何转移图。几何转移图最为常用，这里选用几何转移图。状态数选 5，如图 2-24 所示。

单击按钮进入状态机设计向导的第二个界面，设计状态机的复位机制。复位类型分为同步复位和异步复位两种方式。这里选择同步复位方式，如图 2-25 所示。

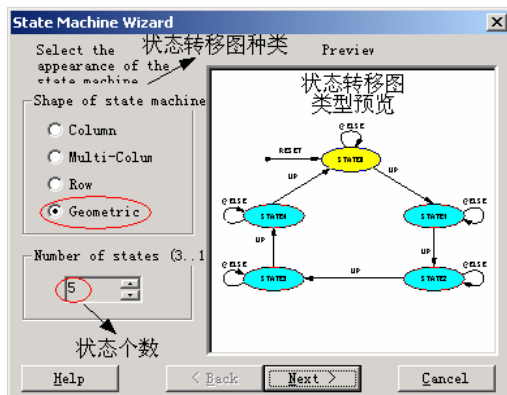


图2-24 状态机设计向导之一，转移图类型和状态数选择

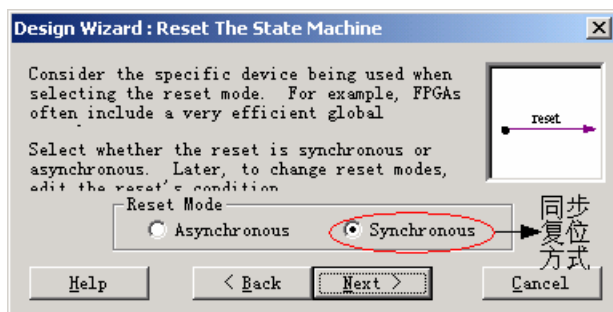


图2-25 状态机设计向导之二，复位方式设置

单击 **Next >** 按钮进入状态机设计向导的第三个界面，设置状态转移方式。状态转移方式分为自循环、下行和上行等 3 种方式。一般比较常用的是下行方式。需要注意的是这 3 种方式可以通过打勾复选，如图 2-26 所示。

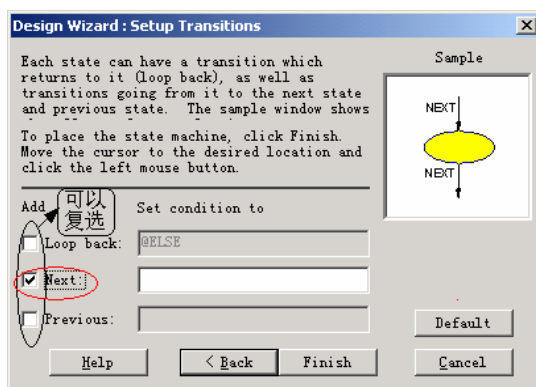


图2-26 状态机设计向导之三，转移方式设置

单击 **Finish** 按钮完成状态机向导设置，然后在绘图工作区居中位置单击鼠标左键，将状态转移图模板放置在绘图窗中，如图 2-27 所示。

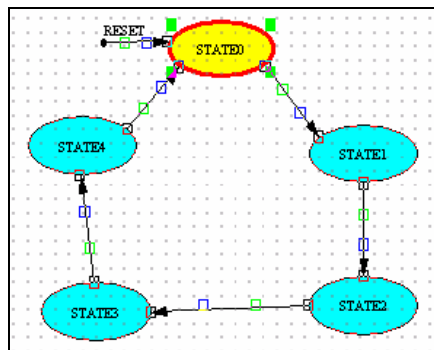


图2-27 状态机向导产生的状态转移图模板

3. 编辑状态。

状态机向导仅提供了一个状态转移图的模板，设计者还需修改这个模板，以符合自己的设计要求。双击任何一个状态，就可以进入状态编辑窗，在状态编辑窗，可以更改状态的名称、输出名称和输出逻辑等。双击状态“STATE0”，按照图 2-28 所示更改该状态，更改后单击 **OK** 按钮确认。

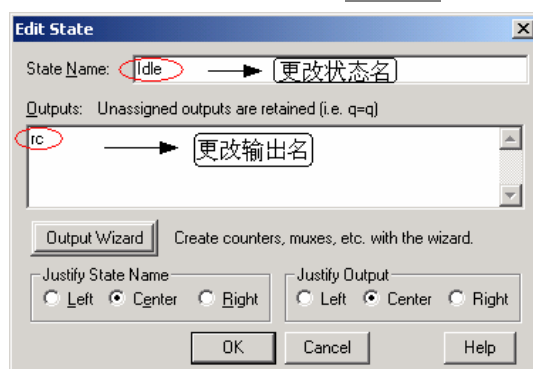


图2-28 编辑状态

4. 使用逻辑向导设计数据流逻辑。

下面启动逻辑向导辅助完成状态转移图数据流的逻辑设计。在 StateCAD 中启动逻辑向导的方法十分灵活，除了前面介绍的 3 种快捷按钮的方法，在状态编辑时也可以使用逻辑向导。双击状态“STATE1”，将状态名改为“Get_1”，然后单击 **Output Wizard** 按钮，打开逻辑向导，看到可用的逻辑设计资源非常丰富，按照图 2-29 所示，选择逻辑类型、输入和输出。

然后单击 **OK** 按钮确认设计。逻辑向导向设计者提供了丰富的逻辑资源，下面为状态“Get_1”添加逻辑设计。双击状态“Get_1”，然后单击 **Output Wizard** 按钮，打开逻辑向导，如图 2-30 所示，添加逻辑类型、输入和输出。

然后单击 **OK** 按钮确认设计逻辑向导，再单击 **OK** 按钮确认对状态“Get_1”的编辑。按照同样的方法设计者还可以为状态添加更多的逻辑设计。

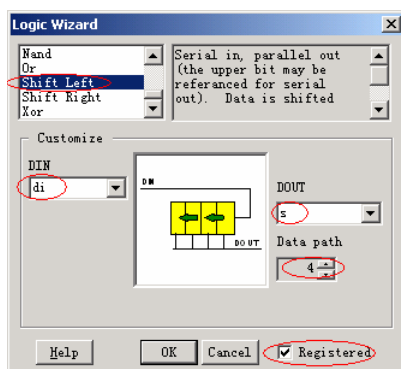


图2-29 使用逻辑向导添加输出逻辑

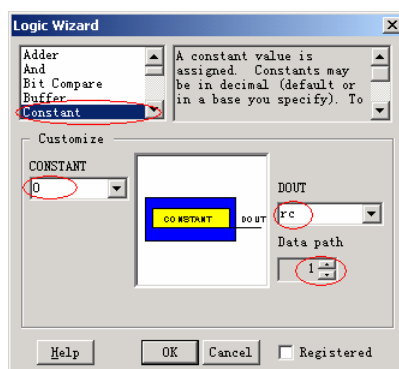


图 2-30 使用逻辑向导添加逻辑

5. 编辑其他状态。

参照第 3、4 步的操作编辑剩余状态，重命名状态“STATE2”为“Split”。重命名状态“STATE3”为“Get_2”，并依图 2-31 所示更改输出逻辑。重命名状态“STATE4”为“Add”，并依图 2-32 所示更改输出逻辑。

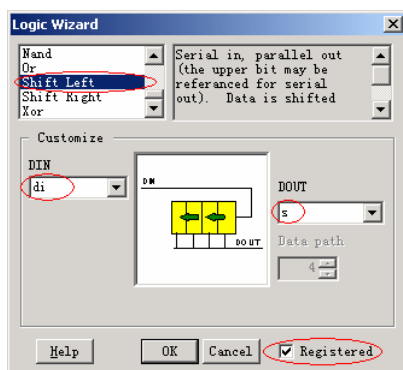


图 2-31 编辑状态“Get_2”

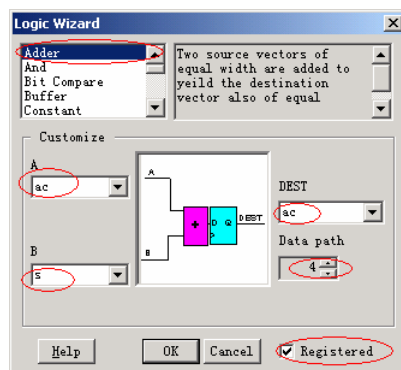


图 2-32 编辑状态“Add”

6. 编辑状态转移条件。

将鼠标光标放在“Idle”到“Get_1”的状态转移线上，双击转移线，进入转移条件编辑框。按照图 2-33 所示，编辑转移条件。

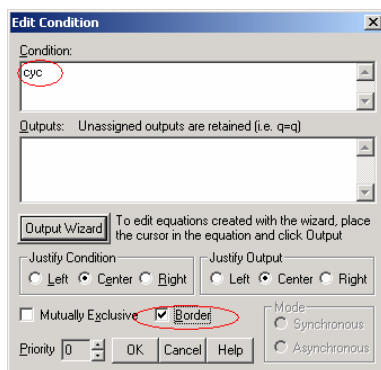


图2-33 编辑状态转移条件

单击  按钮确认编辑，然后给状态“Get_1”到“Split”之间的转移线添加条件“cnt=9”，给状态“Get_2”到“Add”之间的转移线添加条件“cnt=0”。

7. 给状态机添加注释。

给图纸添加适当的注释便于他人读图。单击绘图工具栏中的  按钮，然后在图纸顶部单击鼠标左键，弹出文本编辑框，添加注释“串进并出加法器”。

8. 设置个性化色彩。

设置状态转移图色彩，可以增加图纸的可视性。下面将刚才添加的注释改成绿色。单击鼠标左键选择注释文本，然后单击色彩工具栏中的  按钮，单击按钮  选择绿色，最后点击文本注释框填充颜色。使用色彩工具栏的其他按钮还可以更改边框颜色、文字颜色和字体大小等。

9. 使用逻辑向导进行复杂逻辑设计。

在 StateCAD 中，几乎所有状态的逻辑设计都可以通过逻辑向导完成。如前面介绍，逻辑向导有丰富的逻辑资源可供选择。这些逻辑资源包括加法器、减法器、乘法器、计数器、锁存器、比较器、选择器、移位寄存器，以及各种逻辑门。这些逻辑资源都是完全参数化设计的，工程师只要根据提示选中逻辑类型，改变总线宽度，复位类型和输入输出方式就可以满足自己设计的需求。下面给状态图再添加一些逻辑设计。单击  按钮，在状态“Get_1”左侧单击鼠标左键，启动逻辑向导，按图 2-34 所示为设计添加逻辑功能。

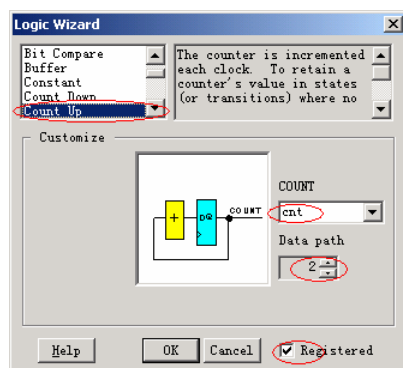


图2-34 Random Logic 向导之一，选择逻辑功能

单击  按钮进入逻辑表达式编辑框，在该窗口可以设计复位类型、总线宽度、逻辑公式和电路形式（寄存器型或组合电路型），如图 2-35 所示。

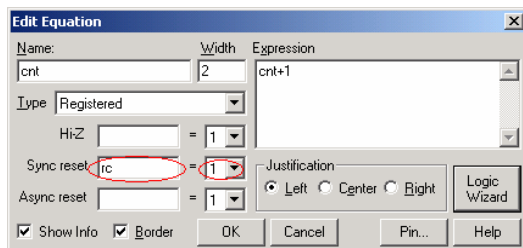


图2-35 Random Logic 向导之二，编辑逻辑表达式

单击  按钮确认逻辑表达式的编辑。

10. 保存设计。

至此，已经使用状态机向导和逻辑向导完成了状态转移图的绘制，单击  按钮保存设计。

11. 启动优化向导，优化设计。

单击  按钮启动优化向导。优化向导的第一个界面如图 2-36 所示，这个界面解释了优化的过程与方法。



图2-36 优化向导之一，优化说明

单击  按钮进入优化向导的第二个界面，选择器件类型，如图 2-37 所示。

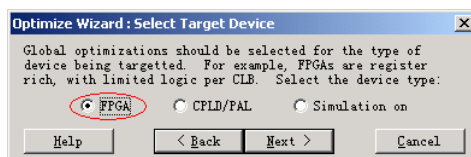


图2-37 优化向导之二，器件选型

单击  按钮进入优化向导的第三个界面，选择优化目标，如图 2-38 所示。

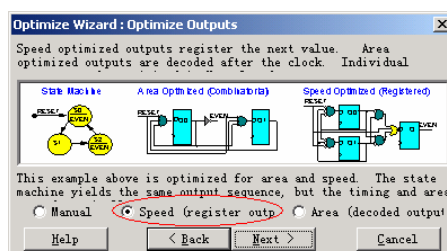


图2-38 优化向导之三，选择优化目标

单击  按钮进入优化向导的第四个界面，提高性能选项，如图 2-39 所示。

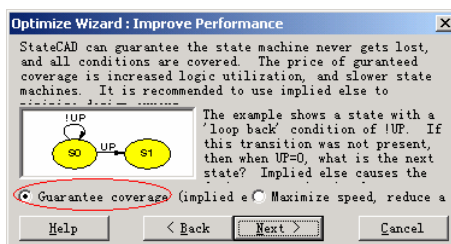


图2-39 优化向导之四，提高性能优化选项

单击 **Next >** 按钮进入优化向导的第五个界面，选择装载类型，如图 2-40 所示。

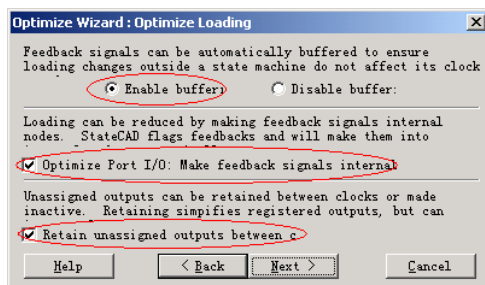


图2-40 优化向导之五，选择装载类型

单击 **Next >** 按钮进入优化向导的第六个界面，选择源代码类型，如图 2-41 所示。

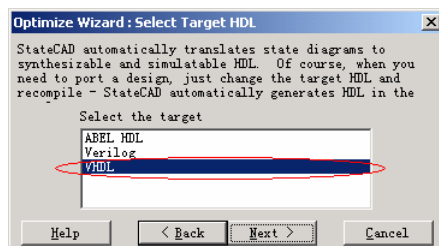


图2-41 优化向导之六，选择源代码类型

单击 **Next >** 按钮进入优化向导的第七个界面，选择综合工具，如图 2-42 所示。

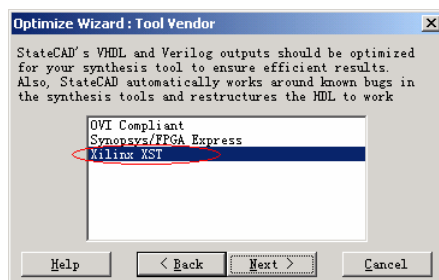


图2-42 优化向导之七，选择综合工具类型

单击 **Next >** 按钮完成这个设计的优化。经过以上步骤，StateCAD 完成了速度、面积、I/O 端口和综合结果等项目的全面优化。

12. 编译设计生成 HDL 代码。

下面编译状态机，并将设计的状态机翻译成 HDL 源代码。编译时 StateCAD 先检查状态机的正确性和完备性，当状态机完全正确后才能编译并翻译成 HDL 代码。单击  按钮，进行编译。这时弹出错误信息对话框，如图 2-43 所示，阅读错误信息发现，由于“cnt”的宽度为 2bit，所以“cnt=9”是无效条件。在错误信息下面是该错误的可能解决方案。这时观察状态转移图，发现在

“cnt=9”的转移条件处被高亮显示，可见在 StateCAD 中检错很方便。

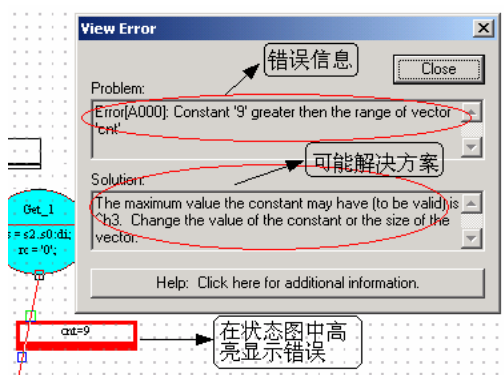


图2-43 检查错误

发现了错误，改正就不难了。将转移条件“cnt=9”改为“cnt=3”，重新编译。这时弹出如图 2-44 所示的对话框。这个对话框的作用是指定状态转移图中信号的 I/O 属性。通过选择信号，告诉 StateCAD 哪些信号仅仅是内部信号，哪些是和外部的接口信号。本例指定“ac”为外部接口信号，其他为内部信号，单击信号“ac”，去掉前面的“X”符号。

单击 **Optimize** 按钮确认，看到正确编译信息窗，如图 2-45 所示。

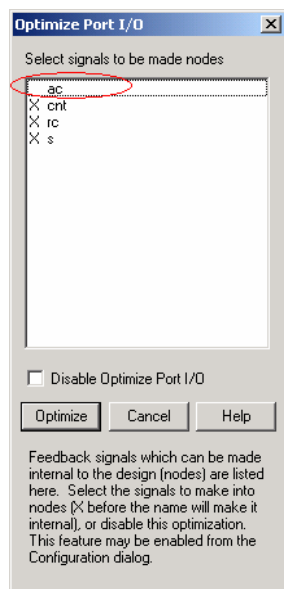


图 2-44 指定信号的 I/O 属性

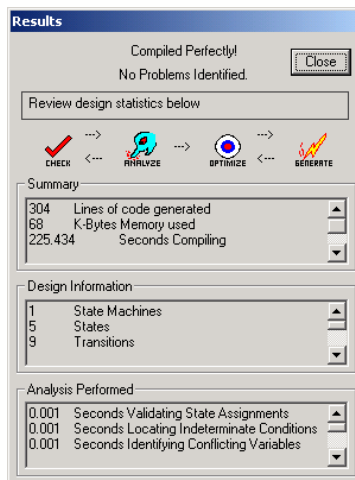


图 2-45 正确编译信息窗

关闭正确编译信息窗，StateCAD 自动将翻译出的 HDL 代码显示在 HDL 浏览器。图 2-46 显示了这个示例的状态转移图和 HDL 代码。至此，状态机已经设计完成。请大家回忆一下设计过程，是不是围绕着操作工具栏的使用呢？

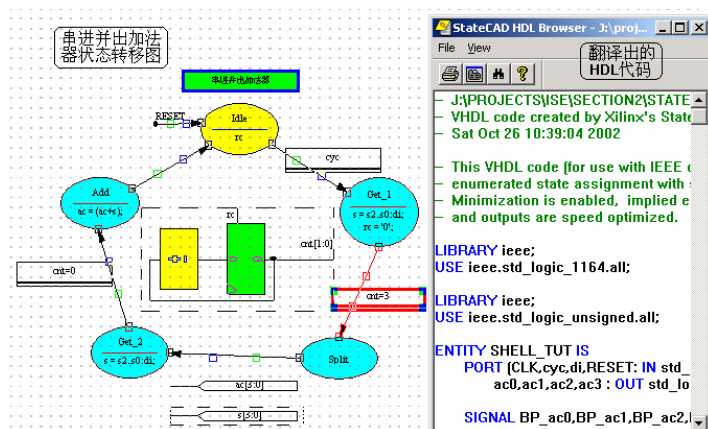


图2-46 串进并出加法器的状态转移图和 HDL 代码

使用 StateCAD 完成状态机的设计后，一般还要通过仿真验证设计的正确性。StateCAD 测试激励生成器能够帮助工程师方便地生成状态机的测试激励。其生成的测试激励不但能够在行为级前仿真中使用，甚至可以作为后仿真的测试激励。下面沿用“串进并出加法器”的例子，使用 StateCAD 的测试激励生成器验证刚才的状态机设计。保持刚才“串进并出加法器”状态机打开，单击 按钮，启动测试激励生成器，可以看到如图 2-47 所示的测试原理图。图中清晰地标明了每个状态和转移条件的逻辑功能。

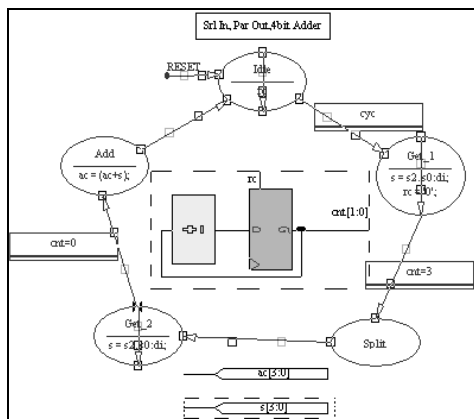


图2-47 串进并出加法器测试图

🔑 使用 StateCAD 测试激励生成器设计测试激励，验证“串进并出加法器”状态机

1. 行为级仿真实验。

单击测试激励生成器工具栏中的 按钮，弹出行为级仿真对话框，在该对话框中依照图 2-48 所示设置仿真的初始化条件，测试目标，测试信号等。

单击 按钮运行仿真，本次仿真结束后，StateCAD 会询问是否需要仿真下个测试目标，选择 按钮。

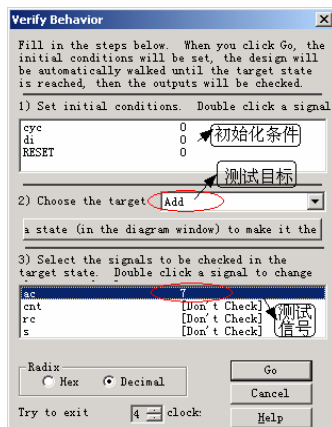


图2-48 行为级仿真对话框

2. 改变输入波形。

下面改变输入信号“di”的波形，使在第一次输入的被加数为“5”，即二进制“0101”，第二次输入的被加数为“2”，即二进制“0010”。具体方法是在时钟“CLK”为“2”，“4”，“8”时，双击“di”，确认更改即可，如图 2-49 所示。现在大家可以预期，加法结果“ac”应该为“7”。

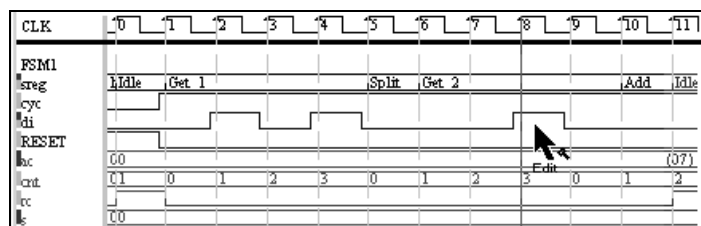


图2-49 改变输入波形

3. 运行测试激励。

单击  按钮，运行测试激励。运行后会弹出报错信息对话框，检查错误信息发现：加法器的输出“ac”结果并不是预期值“7”，而是“2”。

4. 定位错误。

观察仿真结果，在第 5 个时钟周期时，“s”经过移位已经为“5”，但是并没有传给信号“ac”，导致“ac”没有保存第一个被加数，如图 2-50 所示。按钮  的作用是在仿真结果中搜索某个信号的值。使用这个按钮可以辅助定位错误。

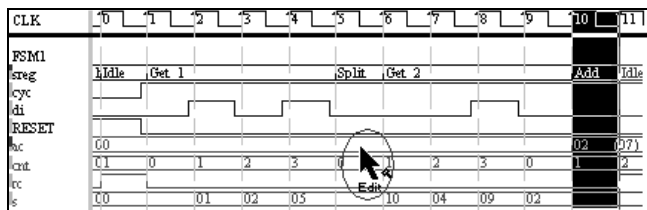


图2-50 定位错误

5. 改正错误。

改正错误的方法是在每次“cnt=3”的时候将“s”赋值给“ac”。可以在状态转移图中更改，然后重新仿真，也可以直接在测试激励生成器的测试波形上修改，这里的修改会直接传递到 StateCAD 的状态机设计。在时钟信号“CLK”为“5”处，双击信号“ac”，点击位置即为图 2-50 所示鼠标光标位置。弹出更改对话框，按图 2-51 所示进行设置。

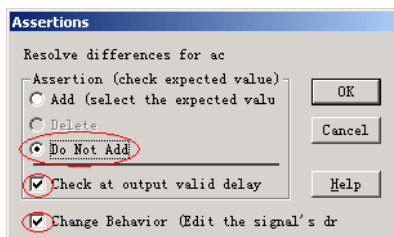


图2-51 更改说明对话框

单击  按钮确认，进入条件编辑框。添加条件“cnt=3”时，“ac=s”。如图 2-52 所示。

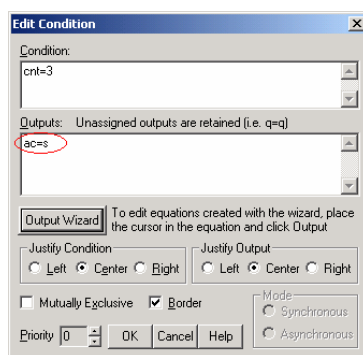


图2-52 添加条件“cnt=3”时，“ac=s”

单击  按钮确认添加条件，重新仿真，发现结果正确了。

6. 保存测试激励，更新设计。

单击  按钮保存测试激励，更新设计。弹出保存对话框，选择自动更新状态机设计，如图 2-53 所示。

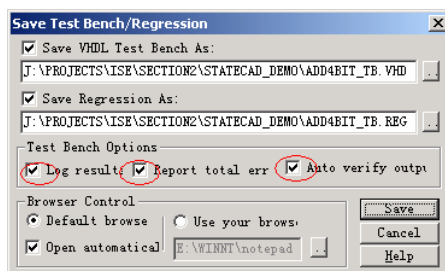


图2-53 保存测试激励，更新设计

确认保存后，测试激励生成器自动生成 HDL 测试激励文件，并用 HDL 浏览器显示。

7. 生成后仿真测试激励。

行为级验证通过后，后仿真测试激励的生成较为简单，其关键之处在于指定时间约束条件。单击  按钮，指定时间约束条件如图 2-54 所示。

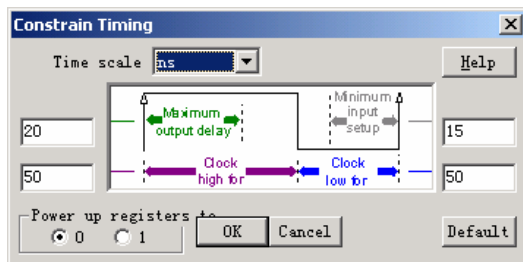


图2-54 指定后仿真时间约束条件

8. 自动生成测试激励。

确认时钟约束条件后，直接单击  按钮，自动生成后仿真测试激励。确认保存测试激励更新工程后，测试激励生成器自动生成后仿真 HDL 代码测试激励，并用 HDL 浏览器显示。

9. 总结覆盖率。

最后测试激励生成器会对测试激励的覆盖性进行总结，并弹出如图 2-55 所示的测试覆盖对话框。至此这个“串进并出加法器”的前、后仿真激励都已经生成。

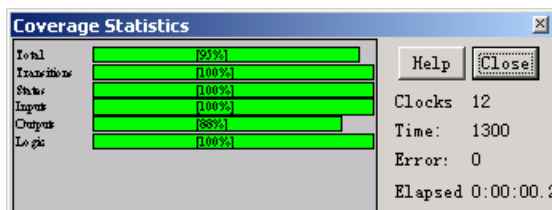


图2-55 测试覆盖对话框

通过前面的介绍，读者应该对使用 StateCAD 设计状态机的流程有了较深的理解。只要理解 StateCAD 设计状态机的步骤，把握住操作工具栏按钮的使用这一主线，相信大家对使用 StateCAD 绘制状态转移图，优化设计，编译检错并转换成 HDL 代码，生成测试激励等操作都会掌握得很好。

2.4 原理图输入工具——ECS

本节将介绍 ISE 中集成的原理图输入工具——ECS (Engineering Capture System) 的基本使用方法。原理图输入方法是 FPGA/CPLD 设计的基本输入方法之一，几乎所有的 FPGA/CPLD 设计集成环境都有原理图输入方法。在 ECS 的图形化用户界面下，用户可以编

辑原理图和模块符号。

2.4.1 ECS 综述

早期的 EDA 设计软件几乎都采用原理图设计输入方法，这种设计方法直观、易学易用，为广大设计工程师所接受。原理图设计方法的主要支撑是功能强大、分门别类的器件库。而不同厂商甚至不同型号的 FPGA/CPLD 的库元件的许多电气特性与结构特点有较大差异。器件库元件通用性差导致了原理图设计的可移植性差。常常会发生这种情况，当设计实现的芯片型号或者提供厂商变化后，整个原理图需要做较大修改甚至全部重新设计。在 FPGA/CPLD 不断推陈出新的今天，这个矛盾越来越尖锐。为了克服原理图设计可移植性差等缺陷，硬件描述语言（HDL）应运而生。硬件描述语言以其可移植性好，可维护性高，利于超大规模设计，利于 EDA 软件的自动化辅助设计等特点迅速成为 FPGA/CPLD 设计的主流输入方法。

所以，对刚接触 FPGA/CPLD 设计的初学者，建议学习 HDL 设计方法（VHDL 和 Verilog HDL）。ECS 在 ISE 中仅仅是一种辅助设计方法，其最常用的场合是在混合设计中。在混合设计中，使用 ECS 将 Verilog HDL、VHDL 和 IP 核生成器（Core Generator）产生的 IP 核、LogiBLOX 生成的功能模块、数字时钟管理模块（DCM）和 StateCAD 设计状态机等设计资源生成原理图模块符号（Symbol），用原理图（Schematic）将这些模块符号组织起来，完成混合设计。

ECS 与大多数原理图设计工具的界面相似，相信会使用 PCB 原理图设计软件的读者一定能够很快掌握 ECS 的基本功能。本节重点介绍使用 ECS 设计原理图的基本方法，同时讲解在混合设计中如何将各种设计资源生成原理图模块符号，至于所用到的各种资源的具体设计方法详见本章其他小节，另外书中第 9 章有一个利用 ECS 进行混合设计的典型的实例，请大家结合本节内容认真学习。



利用 ECS 进行混合设计是 ECS 的一个重要功能。

2.4.2 ECS 的用户界面

Xilinx 的原理图输入工具不断革新，从早期 Foundation 系列原理图输入工具到 ISE 系列 ECS 原理图输入工具，用户界面和功能都有一定的提高。ISE 6.2 中的 ECS 与早期 ISE 版本中的 ECS 相比，在界面、提示信息（Data Tips）、动态放缩视图和易操作性上也有一定的改进。

同样可以使用新建原理图（Schematic）资源的方法启动 ECS。ECS 用户界面如图 2-56 所示。

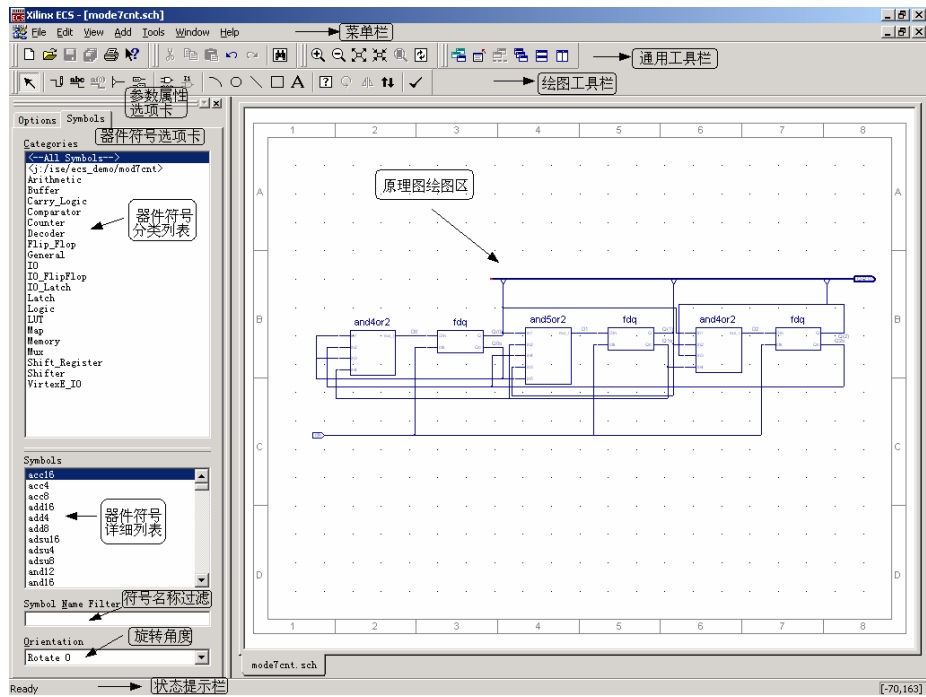


图2-56 ECS 的用户界面

ECS 的用户界面主要由菜单栏、通用工具栏、绘图工具栏、属性设置选项卡、器件符号选项卡、绘图工作区和状态提示栏组成，各部分功能如下。

1. 菜单栏

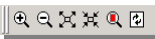
菜单栏由一系列常用处理命令的下拉菜单组成。

- **文件（File）** 菜单除了包含新建、打开和保存等通用文件处理命令外，还包含最近原理图列表（Recent Schematics）、最近符号列表（Recent Symbols）和最近 NGR 文件列表（Recent NGR Files）等命令。
- **编辑（Edit）** 菜单包含了复制、粘贴等通用编辑命令以及符号编辑（Symbol）、更新符号（Update Obsolete Symbols）、重命名所选总线（Rename Selected Bus）、重命名所选网线（Rename Selected Net）、镜像翻转（Mirror）、旋转（Rotate）、选择对象（Select Object）、查找（Find）、对象属性（Object Properties）和显示属性（Preferences）等命令。
- **视图（View）** 菜单包含了一些视图显示方面的常用命令，主要包括显示工具栏（Toolbars）、显示状态栏（Status Bar）、放缩（Zoom）、移动显示区域（Pan）、层次递进（Push Into Symbol or Return To Calling Schematic）和刷新（Refresh）等命令。
- **添加（Add）** 菜单较为重要，包含了绘图常用的命令，主要包括绘线（Wire）、网络名（Net Name）、总线引脚（Bus Tap）、I/O 管脚（I/O Marker）、器件符号（Symbol）、实例化名称（Instance Name）、画弧线（Arc）、画圆（Circle）、画直线（Line）、画矩形（Rectangle）和添加文本（Text）等命令。

- **工具（Tools）菜单**包含了 ECS 中常用的工具命令，主要包括原理图错误检查（Check Schematic）、生成器件符号向导（Symbol Wizard）、生成 IO 管脚（Create IO Markers）和查询（Query）等命令。
- **窗口（Window）菜单**由一些通用窗口调整与排列命令组成。
- **帮助（Help）菜单**包含了 ECS 在线帮助和版本信息等。

2. 通用工具栏

通用工具栏分为文件工具栏、编辑工具栏、视图放缩工具栏和窗口工具栏等 4 部分。

- **文件工具栏**  中的按钮与菜单栏的中文文件（File）菜单中包含的命令功能基本相似。
- **编辑工具栏**  中的按钮与菜单栏中的编辑（Edit）菜单中包含的命令功能相似。
- **视图放缩工具栏**  中的按钮与菜单栏中的视图（View）菜单中包含的命令功能相似。
- **窗口工具栏**  中的按钮与菜单栏中的窗口（Window）菜单中包含的命令功能相似。

3. 绘图工具栏

绘图工具栏  最为常用，包含了绘制原理图的基本命令的快捷按钮，与菜单栏中的添加（Add）菜单包含的命令功能一致，依次为选择（Select）、添加网线（Add Wire）、添加网线名称（Add Net Name）、总线重命名（Rename Bus）、添加总线引脚（Add Bus Tap）、添加 I/O 管脚（Add I/O Marker）、添加器件符号（Add Symbol）、添加实例化名称（Add Instance Name）、画弧线（Add Arc）、画圆（Add Circle）、画直线（Add Line）、画矩形（Add Rectangle）、添加文本（Add Text）、查询（Query）、旋转（Rotate）、镜像翻转（Mirror）、层次递进（Push Into Symbol or Return To Calling Schematic）和原理图错误检查（Check Schematic）等。

4. 属性设置选项卡

当在绘图工具栏中使用某绘图快捷按钮后，ECS 自动在属性设置选项卡提示用户设置该命令参数。属性设置选项卡的参数设置配合绘图工具栏的快捷按钮可以完成绘制原理图中的所有基本操作。

5. 器件符号选项卡

器件符号选项卡是绘图中选取器件符号的工作区域，器件符号按照功能排列，包含本工程资源产生的模块符号。可以使用符号名称过滤缩小查选范围，选择符号后在原理图合适位置单击鼠标左键放置该器件符号。不同器件族的可用器件符号有一定差异。

6. 绘图工作区

绘图工作区是 ECS 的主工作区，原理图的绘制编辑在该窗口中完成。

7. 状态提示栏

状态提示栏提示 ECS 操作状态和快捷按钮功能。另外当鼠标指针指向工具栏中快捷按

钮时，会自动弹出该快捷按钮命令的提示信息。

绘制原理图时，使用快捷键配合滚轮鼠标将大大提高工作效率。

使用键盘方向键←、↑、→和↓能够按照相应方向移动视图和符号的显示区域，相当于菜单栏的移动显示区域（Pan）命令。

按下Ctrl键，滚动鼠标滚轮，能够放大、缩小视图。

滚动鼠标滚轮，能够上下移动视图。

按下Shift键，滚动鼠标滚轮，能够左右移动视图。

按下Ctrl键，按住鼠标右键在原理图某部分画方框，可以将该部分图形放大，相当于放大方框内视图（Zoom To Box）命令。

2.4.3 使用 ECS 完成原理图输入设计

本节将通过一个原理图设计输入实例，带领读者完成新建原理图资源、选择器件符号、连线、绘制总线、生成器件符号和检查错误等一系列原理图绘制的基本操作。

🔑 使用 ECS 完成“模 7 同步加法计数器”的原理图输入，参考示例详见本书附带光盘的“Example-2-3”目录

本节要用 ECS 完成一个模 7 同步加法计数器的设计输入。这个计数器带有自启动功能，其状态转移图如图 2-57 所示。

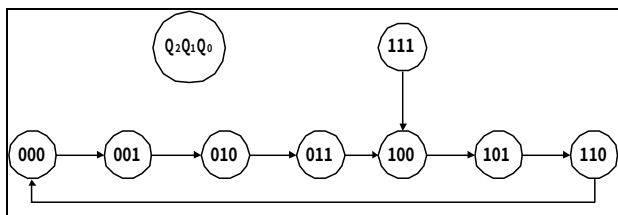


图2-57 模 7 同步加法计数器状态转移图

如采用 D 触发器设计，则其逻辑表达式为：

$$D_2 = Q_0 Q_1 + \overline{Q_1} Q_2 \quad D_1 = \overline{Q_0} Q_1 \overline{Q_2} + Q_0 \overline{Q_1} \quad D_0 = \overline{Q_0} \overline{Q_2} + \overline{Q_0} Q_1$$

整个设计分两步完成：第 1 步，设计所需与或门和 D 触发器，并将它们生成器件符号；第 2 步，使用所生成的器件符号，绘制模 7 加法计数器。

1. 设计所需与或门和 D 触发器，并将它们生成器件符号。这一步骤的学习重点是：新建原理图资源，选择并放置器件符号，连线，添加网线名称，设置 I/O 管脚，生成器件符号。

- (1) 新建原理图资源。

打开 ISE，新建工程 Mod7Cnt。由于这个模 7 加法计数器仅仅采用了一些通用器件模块，在 Xilinx 所有器件上都可实现，所以器件族和型号可以任意选择，综合方式任选。新建原理图资源文件“and4or2”，如图 2-58 所示。

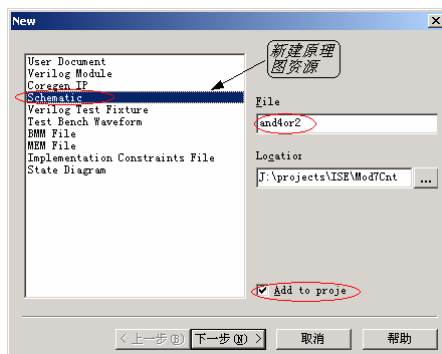


图2-58 新建原理图资源文件

单击 **下一步(N) >** 按钮，显示新建资源信息，单击 **完成** 按钮确认新建资源信息，启动 ECS。

(2) 选择并放置器件符号。

选择并放置器件符号是绘制原理图的第一步。与或门需要的组成器件是逻辑与门，或门等。单击绘图工具栏中的  按钮，在器件符号分类显示目录列表选择“Logic”逻辑目录，在器件符号详细列表选择“and2”两输入与门。也可以在器件符号分类显示目录选择所有器件 (All Symbols)，符号名称过滤框输入“and2”直接选择器件。器件选择完成后，在原理图合适位置单击鼠标左键放置两个两输入与门。同样方法，在器件符号分类显示目录选择所有器件 (All Symbols)，符号名称过滤框输入“or2”直接选择两输入或门器件，在原理图合适位置放置两输入或门。单击 **Esc** 键或者双击鼠标右键结束器件选择与放置，结果如图 2-59 所示。

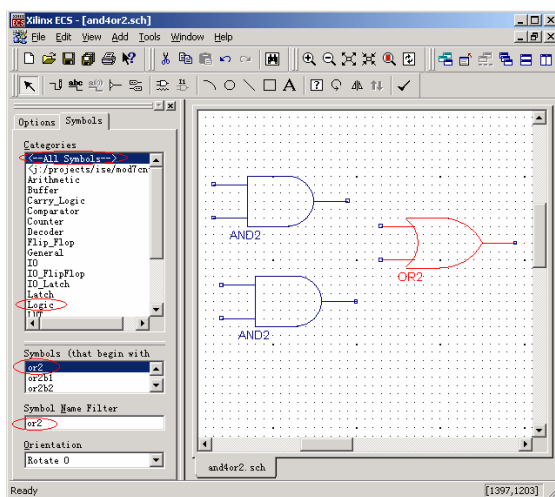


图2-59 选择并放置器件



使用快捷键组合 **Ctrl+R** 可以旋转所选器件。

(3) 添加连线。

单击绘图工具栏中的按钮，在属性设置选项卡选择智能连线方式，当鼠标指针移动到原理图可画线区域时鼠标符号自动变为“+”形状，点击连线开始点的器件管脚两次（第一次选择器件，第二次开始连线），然后点击终点连线的管脚，智能连线器自动帮助工程师在起始与终点管脚之间布置一条连线。单击 **Esc** 键或者双击鼠标右键结束操作，如图 2-60 所示。

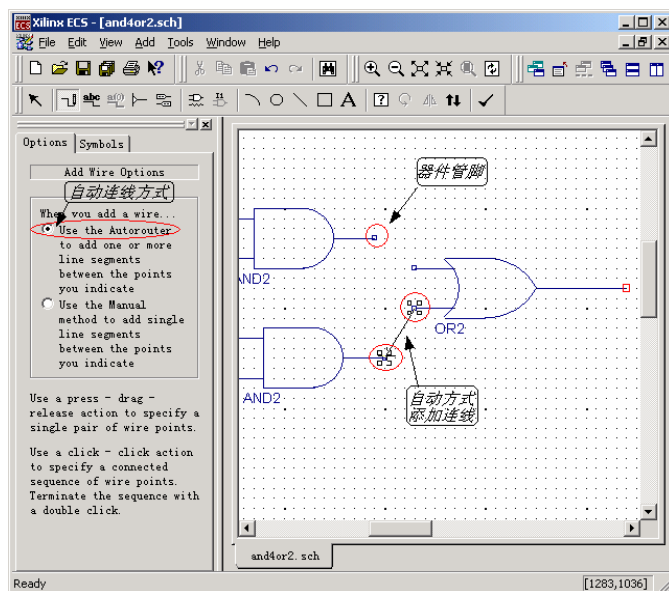


图2-60 添加连线

连接其他网线，如图 2-61 所示，并在器件端口拉出一小段线段为以后添加网线名称并设置 I/O 管脚做准备。

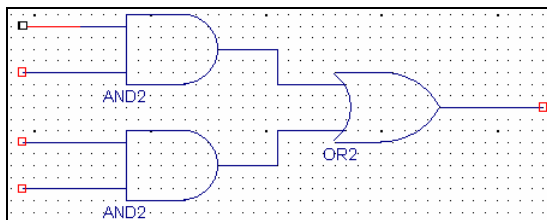


图2-61 与或门连线



如果绘图过程中出错，可以使用快捷键组合 **Ctrl+Z** 撤销上次操作。

(4) 添加网线名称。

添加网线名称的作用有两个：一是增强了原理图的可读性，在综合与实现时也易于发现某条网线的时延与布线情况；另外，ECS 如大多数原理图编辑工具一样，默认同名网线就是一条网线。换句话说，如果两条网线名称相同，即使

在原理图上没有明确连接，ECS 也认为它们是同一条网线，有了连接关系，添加网线名称可以区别网线。

单击绘图工具栏中的按钮，在属性设置选项卡选择网线分支命名（Name the branch），在网线名称栏填入“in1”作为与门的第一个输入管脚名称，当鼠标指针移动到原理图时发现鼠标指针处已有了“in1”网线名称，在与门的第一输入连线左端单击鼠标左键，为该网线添加名称。如果单击网线名称栏旁边的按钮，网线名称序号自动加 1 或者减 1。按同样的方法为其他网线添加名称，如图 2-62 所示，单击 [Esc] 键或者双击鼠标右键结束操作。

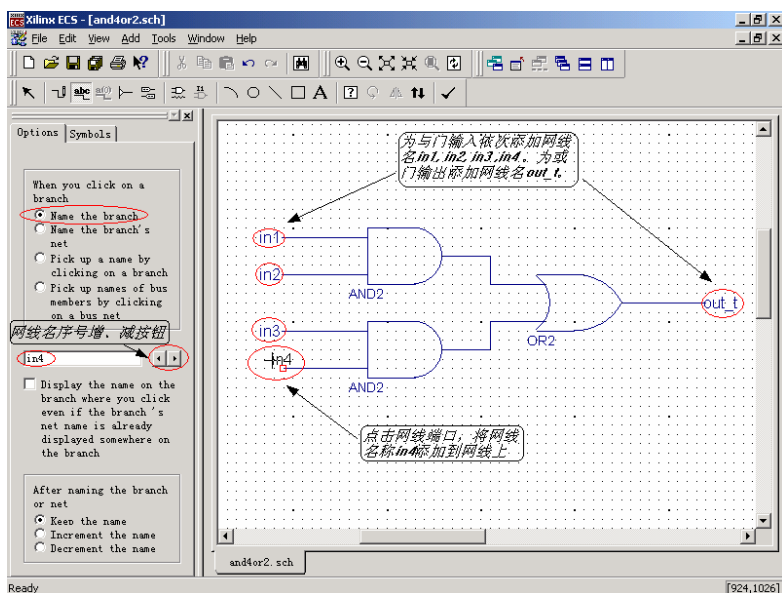


图2-62 添加网线名称

(5) 设置 I/O 管脚。

为已经添加过名称的网线设置 I/O 端口管脚比较简便，单击绘图工具栏中的按钮，在属性设置选项卡选择添加输入管脚（Add an input marker），然后在原理图需要添加 I/O 管脚的网线端口，按住鼠标左键拖出一个框，放开鼠标左键，自动产生一个与网线同名的输出端口。在设置选项卡选择添加输出管脚（Add an output marker），与添加输入管脚相同，添加输出管脚。单击 [Esc] 键或者双击鼠标右键结束操作，结果如图 2-63 所示。

(6) 生成器件符号。

至此，与或门原理图设计已经完成。单击通用工具栏中的按钮保存设计。回到 Project Navigator，在资源管理窗选中“and4or2”原理图资源，打开当前资源操作窗的设计输入工具（Design Entry Utilities），双击“Create Schematic Symbol”选项生成与或门的模块符号，如图 2-64 所示。在 ISE 中几乎所有资源都可以生成模块符号（Symbol），在 ECS 原理图设计中可以调用所生成的模块符号，从而将不同的资源组织起来，这就是混合资源设计的基本方法。

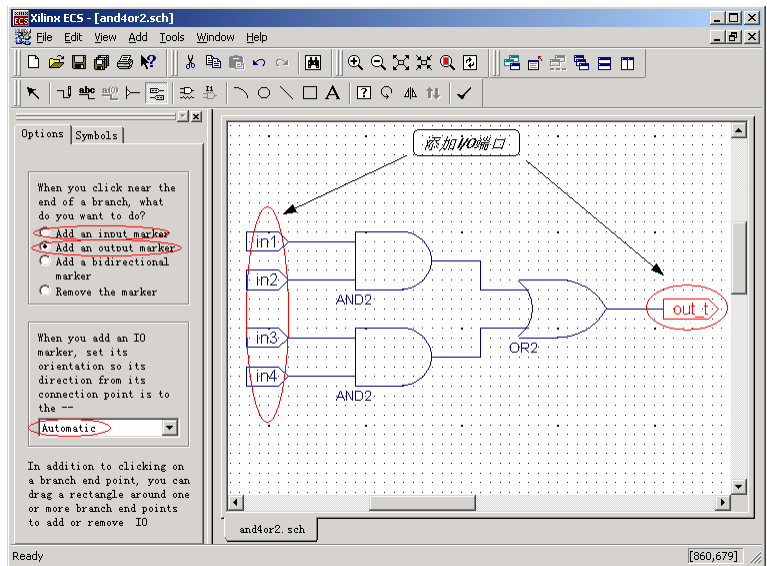


图2-63 设置 I/O 管脚

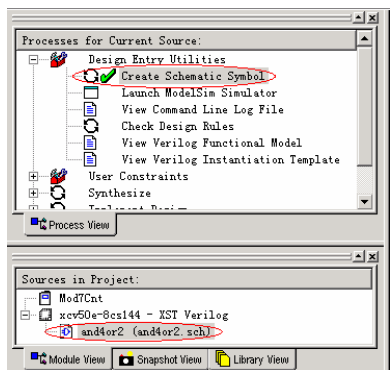


图2-64 生成模块符号

根据图 2-65 和图 2-66 所示的原理图，依照步骤（2）～（6）绘制 5 输入与或门和 D 触发器，并生成模块符号。

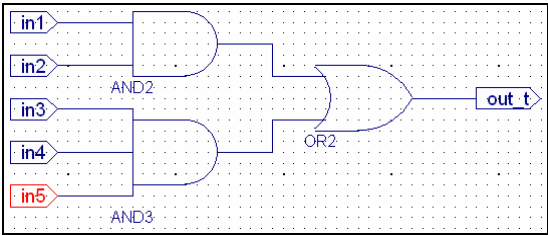


图2-65 与或门“and5or2”原理图

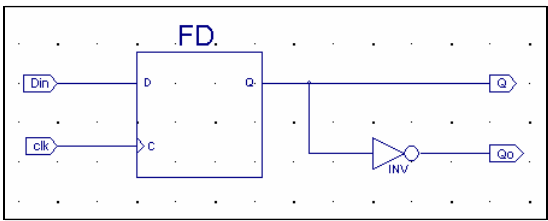


图 2-66 D 触发器 fdq 原理图

2. 调用所生成的器件符号，完成模 7 同步加法计数器原理图设计。这一步骤中的学习重点是：调用本工程资源生成的器件符号，添加总线和错误检查等。
- (1) 调用本工程资源生成的器件符号。

本工程现有资源为 3 张原理图：“and4or2”、“and5or2”和“fdq”，而且都已经生成了器件符号，下面使用顶层原理图将这些资源组织起来。新建顶层原理图“mode7cnt”。调用本工程资源生成的器件符号与调用普通器件符号从操作上讲没有太大区别。单击绘图工具栏中的  按钮，在器件符号分类显示目录列表选择“工程路径/mod7cnt”目录，在器件符号详细列表依次选择新生成的“and4or2”、“and5or2”和“fdq”模块符号，放置在顶层原理图中即可，如图 2-67 所示。

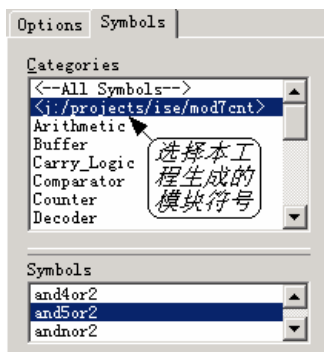


图2-67 调用本工程模块符号

- (2) 按照图 2-68 所示绘制模 7 同步加法器计数器的原理图，绘制方法参照第 1 步中的步骤 (2) ~ (5)。详细原理图参见本书附盘中的文件。

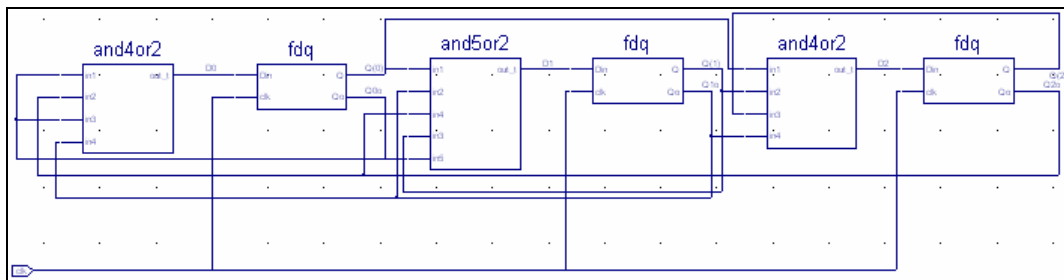


图2-68 模 7 同步加法器计数器原理图

- (3) 绘制总线。

在 ECS 中绘制总线的方法与其他原理图工具相比略有区别。ECS 并没有专门的总线绘制命令，绘制总线仍然采用绘制连线命令“Add Wire”，仅仅利用网线命名区分总线与普通连线。总线名称的命名格式为 BusName (X:Y)，BusName 是总线名称，“()”为总线表示符号，X 为 MSB，Y 为 LSB。下面将模 7 计数器的 3 个寄存器输出绘制成总线 ($Q_2Q_1Q_0$)。单击绘图工具栏中的  按钮，在原理图上方绘制一条线段，然后单击  按钮，在网线名称栏填入“Q (2:0)”即在网线的右端口添加总线名称，总线名称一旦加入，就发现网线变粗，表示由普通连线变成了总线，如图 2-69 所示。

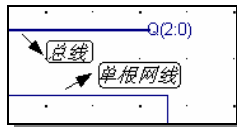


图2-69 总线与单根连线

然后单击  按钮，在总线上添加 3 个总线分支。总线分支表示总线和单根信号线之间的连接转换。适当旋转总线分支方向，使总线分支的三角形连接在总线上，如图 2-70 所示。

单击绘图工具栏中的  按钮，绘制连线连接总线分支与单根信号线，如图 2-71 所示。

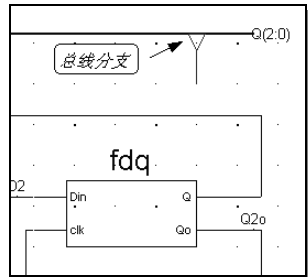


图 2-70 添加总线分支

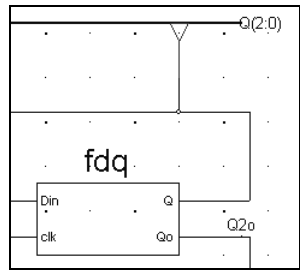


图 2-71 连接总线分支与单根信号线

最后为汇入总线的信号线添加网线名称。添加的网线名称格式为 “BusName (a)”，其中 “BusName” 为总线名称，“a” 为在总线 MSB 与 LSB 之间的一个阿拉伯数字，表示信号线标号，如图 2-72 所示。

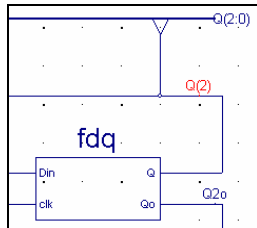


图2-72 添加汇入总线的信号线名称

至此已经将“模 7 同步加法计数器”原理图绘制完毕，完成的原理图如图 2-73 所示，详细原理图参见本书附盘文件。

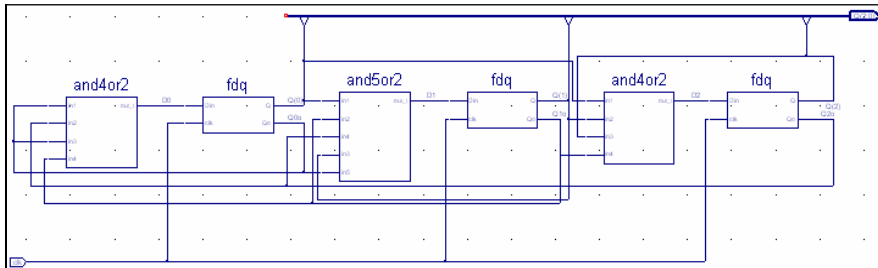


图2-73 完整的“模 7 同步加法计数器”原理图

(4) 检查错误。

原理图绘制完成后还必须检查错误，原理图检错可以在 ECS 中完成，也可以在 Project Navigator 中选中原理图，单击检查错误  Check Design Rules 选项完成。在 ECS 中检查错误比较直观，更改方便，所有一般在 ECS 中检查原理图。具体操作为单击绘图工具栏中的  按钮，ECS 自动检查当前原理图的连接逻辑。如果正确将出现图 2-74 所示对话框。如果原理图有错，将罗列所有错误，使用居中 (Center)、放大 (Zoom In) 和缩小 (Zoom Out) 按钮可以帮助设计者迅速定位错误位置。

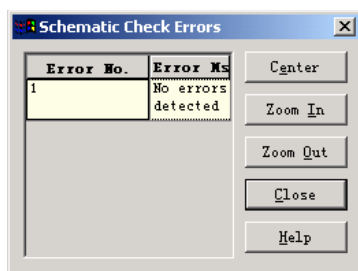


图2-74 检查错误对话框

至此模 7 同步加法计数器的原理图设计输入已经全部完成。有兴趣的读者可以自己做仿真与综合实现。通过这个例子的学习，读者应该对使用 ECS 进行原理图设计输入有个整体上的把握，学会新建原理图资源，选择并放置器件符号，添加连线，添加网线名称，设置 I/O 端口，添加总线，添加总线分支，原理图检错，生成模块符号等基本操作与命令。

2.4.4 使用 ECS 进行混合设计的方法

前面介绍过，ECS 的一个重要功能是完成混合资源设计。使用 ECS 进行混合资源设计的基本方法是利用 ECS 将 Verilog HDL、VHDL 和 IP 核生成器产生的 IP 核，LogiBLOX 生成的功能模块，数字时钟管理模块 (DCM)，StateCAD 设计状态机等设计资源生成模块符号 (Symbol)，用原理图 (Schematic) 作为设计的顶层，调用所生成的模块符号，达到组织各种不同设计资源进行混合设计的目的。

在 2.3 节已经介绍了如何将原理图生成模块符号，其实 ISE 中几乎所有资源都可以方便地生成模块符号。在 ECS 中生成不同资源的模块符号的基本方法相同，都是在 Project Navigator 的资源管理窗中选中某种资源文件，打开当前资源操作窗的设计输入工具 (Design Entry Utilities)，双击  Create Schematic Symbol 选项生成模块符号。

本节仅介绍使用 ECS 进行混合设计的基本概念和方法，并演示如何把不同资源生成模块符号，详细的实例请参考第 9 章的范例。该范例将带领读者复习设计 HDL 源文件、StateCAD 状态机、Core Generator 生成的 IP 核、DCM 生成的时钟模块和原理图等不同资源的方法，并将所有资源生成模块符号，然后在顶层原理图组织这些资源，完成混合资源设计。

2.5 IP 核生成工具——CORE Generator

IP 核生成器是 Xilinx FPGA 设计中的一个重要设计输入工具。它提供了大量的成熟、高效 IP 核为用户所用。IP 核生成器可生成的 IP 核功能繁多，从简单的基本设计模块到复杂的处理器等一应俱全。配合 Xilinx 网站的 IP 中心使用，能大幅度减轻工程师的工作量，提高设计质量。

2.5.1 CORE Generator 综述

Core Generator 可生成的 IP 核大致分为：基本模块、通信与网络模块、数字信号处理模块、数学功能模块、存储器模块、微处理器、控制器与外设模块、标准与协议设计模块、语音处理模块、标准总线模块、视频与图像处理模块等。这些功能涵盖了从基本设计单元到复杂功能样机的众多成熟设计。而且每次 ISE 的升级补丁中都会有 IP 核的升级，另外用户可以通过 Xilinx 的 IP 中心查询更多的 IP 信息。

如果用户设计只是针对 FPGA 应用的，使用 IP 核能避免重复设计，缩短工程时间，提高工作效率。这些 IP 核是根据 Xilinx 的 FPGA 器件特点和结构而设计，直接用 Xilinx FPGA 底层硬件原语描述，充分发挥了 FPGA 的功能，其实现结果在面积和速度上都令人满意。



在进行 FPGA 设计时，根据设计功能要求与器件型号，在 Core Generator 中查找是否有相应的 IP 核可用是一条设计的捷径。

ISE 5 及其以上版本的 CORE Generator 支持 Spartan™ -II/-IIE/-III、Virtex /-E /-II/-II PRO 等器件，不再支持 XC4000 和 Spartan 等老型号器件的 IP 核，如果用户仍然需要使用这些型号器件的 IP 核，必须安装 ISE 4 等早期版本。与早期版本相比，ISE 6 系列的 IP 核生成器在网表的总线格式、用户界面和存储器编辑器（Memory Editor）等方面有一定改进。

下面对 IP 核生成器的输入输出文件加以简单解释，使用户对这些文件有个整体上的把握。IP 核生成器的输入文件如表 2-4 所示。

表 2-4 IP 核生成器输入文件

扩展名	关联工具	文件功能
.coe	Memory Editor 文本编辑器	参数与初值文本文件。以 ASCII 存储 IP 核的参数和初值。如 FIR 滤波器的系数、相关器的掩图和存储器初值等。可以使用 IP 核生成器提供的存储器编辑器（Memory Editor）协助设计存储器的初值文件
.xco	文本编辑器	XCO 文件既可以是输入文件也可以是输出文件，它包含了当前工程的属性与 IP 核的参数信息。当作为输入文件时，作为菜单栏的【File】菜单中【Execute Command File】命令的执行脚本
.xcp	文本编辑器	即为简化的.xco 文件，包含了 IP 核的参数信息

IP 核生成器的输出文件如表 2-5 所示。

表 2-5 IP 核生成器输出文件

扩展名	关联工具	文件功能
.asy	ECS	原理图编辑器（ECS）的图形符号文件
.edn	标准 EDIF 网表文件	该网表文件是 IP 核最重要的输出。Xilinx 实现工具根据该网表实现 IP 核的映射与布局布线
corename_padded.edn	管脚文件	生成 IP 核时，选择“Add Pads”生成的管脚文件
corename_flist.txt	文本编辑器	说明 IP 核生成器生成的所有文件名称的文本文件
CoreGen.log	文本编辑器	显示用户信息、版本信息、器件族、工程属性、IP 核参数和输出文件格式等所有相关的工程信息
.mif	编辑器	和 .ceo 文件相似，包含了 RAM 的初值，FIR 滤波器系数，相关器掩图等参数的初始文件
.ndf	综合、实现工具	与 .ngc 文件相关，向第三方综合工具提供了 IP 核的面积利用和时延等信息
.ngc	综合、实现工具	Xilinx 实现后的二进制网表文件
.v	Verilog 文件	为仿真生成的 Verilog 源代码文件
.veo	Verilog 模块文件	Verilog 源代码，模块实例化时使用
verilog_analyze_order		Verilog 源代码的行为级建议编译顺序
.vhd	VHDL 源文件	为仿真生成的 VHDL 源代码文件
.vho	VHDL 源文件	VHDL 源代码，模块实例化时使用
vhdl_analyze_order		VHDL 源代码的行为级编译顺序
vlink.log		该文件在新版 IP 核生成器取消
.xco	文本编辑器	XCO 文件既可以是输入文件也可以是输出文件，它包含了当前工程的属性与 IP 核的参数信息。IP 核生成后自动产生该脚本文件。在 Project Navigator 的当前资源窗口，该文件作为 IP 核资源的表示文件
.xcp	文本编辑器	即为简化的 .xco 文件，包含了 IP 核的参数信息
.xsf	Mentor 工具 文本编辑器	为 Mentor 工具设计的端口和 IP 核的参数信息
XilinxCoreLib/*.v	Verilog 源代码	所有 IP 核的 Verilog 仿真模型。编译后作为仿真库使用
XilinxCoreLib/*.vhd	VHDL 源代码	所有 IP 核的 VHDL 仿真模型。编译后作为仿真库使用
XilinxCoreLib/*_comp.vhd	VHDL 源代码	VHDL 器件声明文件

2.5.2 CORE Generator 的用户界面

IP 核生成器的启动方法有两种，一是直接在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【CORE Generator System】命令，另一种方法是在 Project Navigator 中新建 Coregen IP 类型的资源。IP 核生成器的用户界面如图 2-75 所示。

这个标准 Windows 界面主要分为以下几个部分：菜单栏、标准工具栏、查询方式工具栏、IP 核目录分类与详细信息窗、已生成的 IP 核信息窗、信息显示窗、Web 服务工具栏和状态提示信息栏等。下面分别介绍这些窗口与工具栏的基本使用方法。

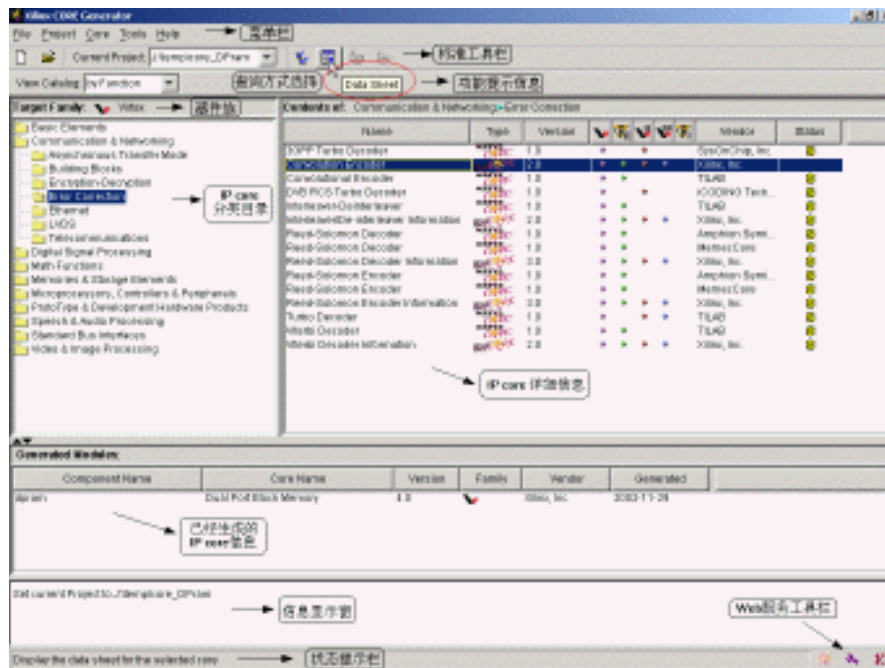


图2-75 IP 核生成器主用户界面

1. 菜单栏

菜单栏由一系列下拉菜单组成，这些菜单涵盖了 IP 核生成器的所有命令。单击带“...”的命令会弹出命令执行对话框。

- 【File】菜单包括 3 个命令：【Execute Command File】为批处理文件 IP 核生成模式命令。【Recustomize Core】命令可对已生成的 IP 核进行修改。【Exit】为退出 IP 核生成器命令。
- 【Project】菜单集成了所有工程处理的命令。如新建工程（New），打开工程（Open），复制工程（Copy），删除工程（Remove）等。另外用户可用【Recustomize】命令修改已经生成的 IP 核，用【Regenerate】命令重新生成 IP 核，用【Cores Catalog Display】命令显示 IP 核的组织目录。【Project Options】命令用于设置工程基本信息。【Preferences】命令指定 PDF 阅读器和网络浏览器的位置，并设置代理服务器等互联方式。
- 【Core】菜单有两个命令：【Customize】命令用于设置 IP 核参数，【Data Sheet】命令用于打开所选 IP 核的 PDF 说明文档。
- 【Tools】菜单包含了 IP 核生成器中的 4 个工具命令：【Get Models】命令的作用是在升级 IP 核工具包后自动更新 IP 核的仿真模型的信息。使用【Updates Installer】命令升级可用的 IP 核。【IP Capture】是一个帮助用户生成自己 IP 核的命令，在帮助的引导下，用户可以快捷地生成自己的 IP 核，一方面实现了模块复用，另一方面保护了知识产权。【Memory Editor】命令的主要功能是设计存储器的初始化文件（扩展名为 COE）。
- 【Help】菜单包含的命令有在线帮助、网络帮助和版本信息等。

2. 标准工具栏

标准工具栏包含了一些常用命令的快捷按钮。当鼠标箭头指向标准工具栏的按钮时会自动弹出图 2-75 所示的功能提示信息。

- 按钮的功能是新建工程。
- 按钮的功能是打开工程。
- 通过 Current Project: J:\Project_Navigator_Demo\SE51st 列表框显示当前工程名称。
- 按钮的功能是设置所选 IP 核的参数。
- 按钮的功能是显示所选 IP 核的 PDF 说明文档。
- 按钮的功能是修改已经生成的 IP 核。
- 按钮的功能是重新生成 IP 核。

3. 查询方式工具栏

查询方式工具栏中的下拉列表框指定了不同的 IP 核目录组织分类方式。IP 核目录的组织显示分为：以功能分类、以字母顺序排列、以供应商分类、以器件族分类和以 IP 类型分类等 5 种形式。

4. IP 核目录分类与详细信息窗

IP 核目录分类与详细信息窗是 IP 核生成器的主体窗口。它分为两部分，左边窗口以“查询方式工具栏”所选分类方式显示所有可用 IP 核的组织目录，右边窗口显示所选分类的详细信息，如图 2-76 所示。

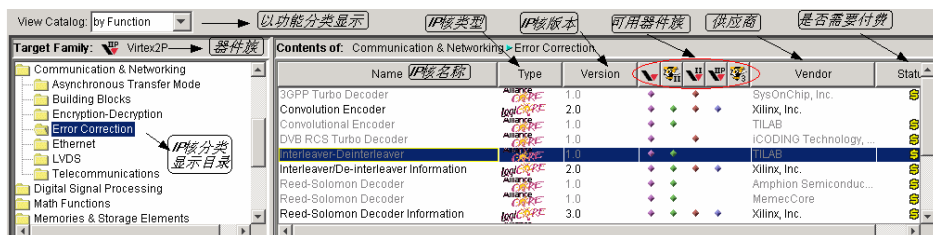


图 2-76 IP 核目录分类与详细信息窗

用户在 IP 核目录分类与详细信息窗中查找所选型号器件可用的 IP 核，然后双击所选 IP 核条目，就能自动唤出该 IP 核的参数设置窗口，如图 2-77 所示。在该窗口可以启动 IP 核的 PDF 说明文档，设置 IP 核的参数信息，并生成 IP 核。



图 2-77 IP 核的参数设置窗口

5. 已生成的 IP 核信息窗

已生成的 IP 核信息窗显示了当前工程已经生成的 IP 核的基本信息，包括 IP 核模块名称、IP 核名称、版本号、器件族、供应商和生成日期等，如图 2-78 所示。

Generated Modules:					
Component Name	Core Name	Version	Family	Vendor	Generated
conv_encoder	Convolution Encoder	2.0		Xilinx, Inc.	2002-11-2
mem_core	Dual Port Block Memory	4.0		Xilinx, Inc.	2002-11-2

图2-78 已生成的 IP 核信息窗

6. 信息显示窗

信息显示窗显示了 IP 核生成器和用户之间的交互信息，如工程的报警、错误等基本信息。

7. Web 服务工具栏

Web 服务工具栏有 3 个按钮，其功能和菜单栏的帮助（Help）菜单的网络帮助（Help on the Web）命令的功能相同。当鼠标箭头指向按钮时会有相应的功能提示信息。

- 按钮  指向 Xilinx 的 IP 核中心网页 <http://www.xilinx.com/ipcenter>。在 IP 核中心，用户可以查找到 Xilinx 最新的 IP 核信息与第三方 IP 核及工具。
- 按钮  指向 Xilinx 技术支持网页 <http://support.xilinx.com>，为用户提供个性化的网上技术支持。
- 按钮  指向 Xilinx 公司的网站 <http://www.xilinx.com>。

8. 状态提示信息栏

状态提示信息栏显示了当前工程状态与鼠标箭头所指按钮的提示信息。

2.5.3 使用 CORE Generator 生成 IP 核的方法与技巧

CORE Generator 作为 Xilinx FPGA 设计的一个实用工具，希望初学者能够迅速学会它的使用。

IP 核生成器的使用基本可以分为以下几个步骤：工程的创建与管理，查找适用的 IP 核，IP 核的参数设计与生成，IP 核的仿真与综合。下面以生成 Virtex-II Pro 的双口 RAM IP 核为例，说明各个步骤的含义与操作技巧。

使用 CORE Generator 生成 Virtex-II Pro 的双口 RAM IP 核，参考示例详见本书附带光盘的“Example-2-4”目录

1. 工程的创建与管理。

- (1) IP 核生成器可以通过 Windows 启动菜单中的 CORE Generator System 项目启动，也可以在 Project Navigator 中新建 Coregen IP 类型的资源，自动调用 IP 核生成器。新建资源方法已经在本章第 1 节介绍过，在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【CORE Generator System】命令启动 IP 核生成器。IP 核生成器启动后自动弹出如图 2-79 所示的

启动对话框。在启动对话框显示了以往工程的记录，可以打开一个工程或者新建一个工程。

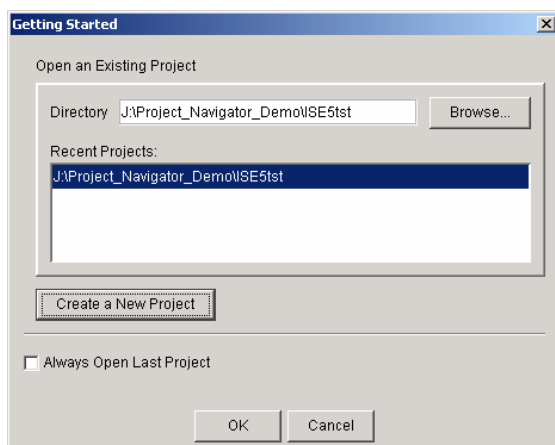


图2-79 启动对话框

- (2) 单击 **Create a New Project** 按钮新建一个工程，自动弹出工程属性对话框，如图 2-80 所示。这个对话框十分重要，当前工程的重要属性信息都在其中设置。如果工程已经建立，可以通过执行【Project】/【Project Options】命令修改工程属性信息。

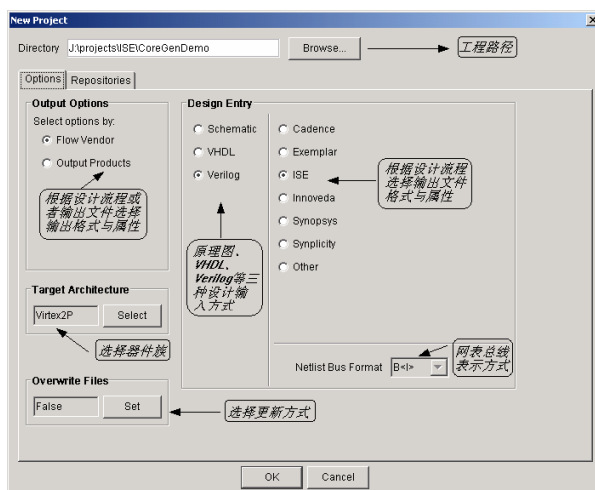


图2-80 工程属性的设置

工程属性的主要设置都在属性选项卡上完成。属性选项卡上主要设置以下项目：输出属性、设计输入方式、器件族类型、文件更新与覆盖方式等。下面依次介绍每项属性的设置。

设置输出属性时，先在供应商流程（Flow Vendor）和输出文件（Output Products）两个单选框中二选一，然后根据所选内容在右边的输出文件格式与属性单选项中选择合适的文件格式，这时可以看到右下脚的网表总线表示方式会发生相应的变化。如果选择供应商流程（Flow Vendor）选项，IP 核生成器会根据所需的设计流程的供应商名称自动配置输出文件的类型与格式，如果选择输出文件（Output Products）选项，可以手动指定所需的输出文件

格式的属性，如图 2-81 所示。

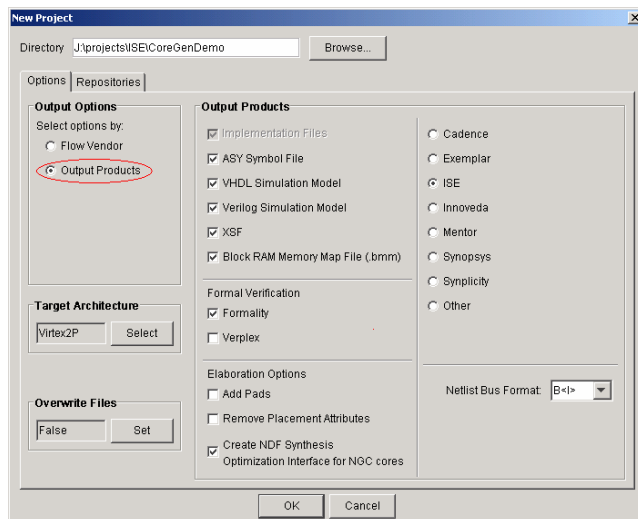


图2-81 输出文件格式与属性

IP 核生成器支持的设计输入方式包括原理图、VHDL 和 Verilog 等 3 种。通过设置输出文件格式的属性和设计输入方式，IP 核生成器以适当的方式输出 IP 核的生成结果，并用兼容的语法描述总线结构和综合仿真命令等。

不同族的器件的内部结构有较大的差异，可用 IP 核不一样，所以必须选择器件族类型。单击 **Select** 按钮，弹出器件族选择框，可选择当前工程使用的器件族，如图 2-82 所示。

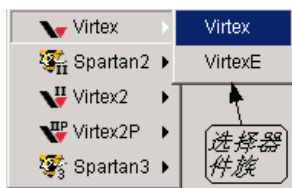


图2-82 选择器件族

文件更新与覆盖分为"True"、"False"、"Default"等三种方式。含义分别如下：

- True: 当新生成的 IP core 和已生成的 IP core 同名时，不做任何提示，自动覆盖原有文件。
- False：提示是否覆盖已生成的文件。
- Default：选择该选项后，文件的覆盖由 "Project" -> "Preferences" 下面的 "Automatically overwrite output files" 的选择情况所确定。

按照图 2-80 设置工程属性，单击 **OK** 按钮，IP 核生成器显示仿真提示信息，单击 **确定** 按钮完成新建工程，进入 IP 核生成器主界面。

2. 查找适用的 IP 核。

正确设置工程的属性信息后，就可以根据需求选择合适的 IP 核。一般在查询方式工具栏中指定以功能分类方式显示 IP 核的类型与组织目录。例如需要一块双口 RAM，

如图 2-83 所示，在 IP 核功能目录中选择存储器模块（Memories & Storage Elements），然后选择“RAMs & ROMs”。在左边的详细信息窗中可以看到共有 5 种存储模块。其中灰色的项目表示由于所选器件族不支持此 IP 核，所以该 IP 核在当前工程不可用；而状态栏有  标志的项目表示该 IP 核已经被废止，但是如果用户确实需要，这种被废止的 IP 核仍然可以使用；状态栏显示“\$”标志的项目表示该 IP 核需要付费购买。版本号 4.0 的双口 RAM 可以在 Virtex-II Pro 器件族使用，双击该 IP 核项目，打开 IP 核参数设置窗，点击  按钮，启动该 IP 核的 PDF 器件说明文件。根据器件说明判断这个 IP 是否满足设计的需要。

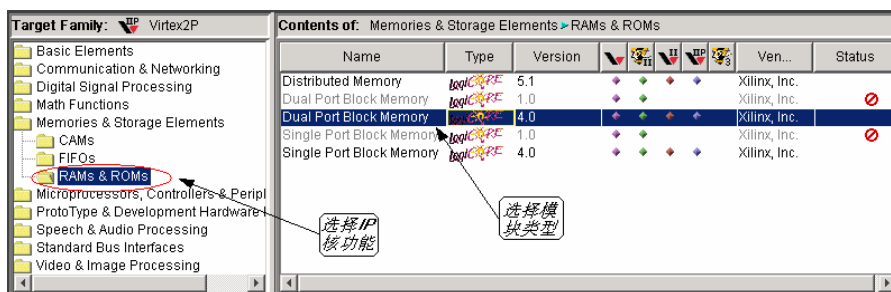


图2-83 根据功能需求选择 IP 核

3. IP 核的参数设计与生成。

双击双口 RAM IP 核项目，打开 IP 核参数设置窗，如图 2-84 所示。设置 A、B 口宽度，读写特性，握手信号，初值（前面简单介绍过 RAM 的初值可以使用 RAM Editor 编辑成 COE 文件赋值）等。如果 IP 的设置参数不止一页，可以单击  进入下一页参数的设置。设置完成后，单击  按钮，立即生成双口 RAM。

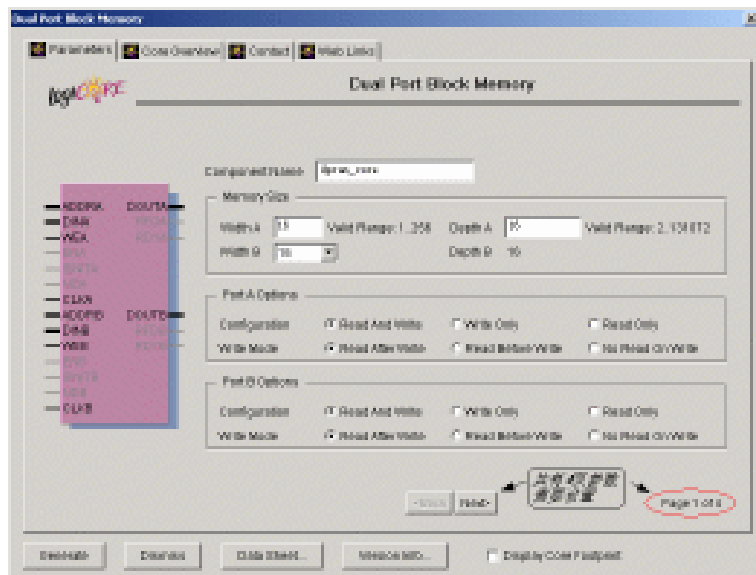


图2-84 IP 核的参数设计与生成

如果需要对生成的 IP 进行参数修改或者重新生成，可以使用标准工具栏中的  和  按钮。

4. IP 核的仿真与综合。

IP 核在综合时一般被认为是黑盒子 (Black Box)，综合器不对黑盒子做任何编译。将 IP 核加入工程的方法一般有两种。一种是在工程中新建 Coregen IP 类型资源，在新建资源文件窗口选中【Add to Project】选项，直接将资源加入到工程中去。这种方法在使用 ISE 设计流程时非常方便，IP 核生成器生成的 IP 核将被直接加入当前工程并在当前资源窗口显示为 Core Generation IP 资源，如图 2-85 所示，修改参数直接双击 IP 核的 XCO 文件图标即可。但是这种调用方法也有它的缺陷，如果用源文件新建一个工程，则需要把已生成的 IP 核的 XCO 参数文件加入到新工程中去，然后执行【Regenerate Core】命令，重新生成 IP 核。

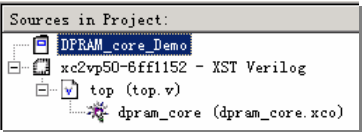


图2-85 当前资源窗的 IP 核资源

第二种调用方法主要是针对第三方综合工具而言的，同时也避免了在新工程中需要重新加入 IP 核资源的麻烦。这种调用方法是将 IP 核声明成黑盒子，具体操作时可以利用 IP 核生成时产生的仿真文件 (.v、.vhd) 和 IP 核实例化文件 (.veo、.vho)，将仿真文件中的 IP 核的相关部分原封不动的拷贝到顶层文件中去，声明 IP 核模块，然后将实例化文件 (.veo、.vho) 内容粘贴到模块的实例化部分。如果生成的 IP 核在 ECS 原理图编辑器中使用，只需将 IP 核生成时自动产生的原理图符号文件 (.asy、.xsf) 加入到原理图中即可。

需要补充的是，使用 Synplify & Synplify Pro 等综合工具综合 IP 核等 Xilinx 硬件原语时，需要调用相应 Xilinx 器件的硬件原语声明文件。位于 Synplify & Synplify Pro 安装目录下 “\lib\Xilinx ” 子目录中的 virtex.v、virtex.vhd、virtexe.v、virtexe.vhd、virtex2.v、virtex2.vhd、virtex2p.v 和 virtex2p.vhd 等文件即为相应器件族的硬件原语声明文件。调用时使用 “include ” 命令。表 2-6 罗列了常用综合工具在源代码中将模块综合为黑盒子的语法。



不论哪种 FPGA/CPLD 集成设计环境，第三方综合工具一般都把器件商提供的底层模块与硬件原语和 IP 核综合成“黑盒子”，在布线器实现过程中才真正地按照实际结构实现。

表 2-6 常用综合工具在源代码中将模块综合为黑盒子的语法

综合工具	Verilog 语法声明	VHDL 语法声明
Synplicity- Synplify, Synplify Pro, Amplify	//synthesis black_box	attribute syn_black_box : boolean; attribute syn_black_box of core_name : component is true;
Synopsys-FPGA Compiler II , FPGA Express	//synopsys translate_off //synopsys translate_on	attribute fpga_dont_touch: string; attribute fpga_dont_touch of core_name : component is "true";
Xilinx-XST	//box_type "black_box"	attribute box_type : string; attribute box_type of core_name : component is "black_box";

IP 核的仿真主要是运用 CORE Generator 的 IP 核仿真模型完成。早期版本 CORE Generator 附带的 IP 核的仿真模型是一些独立的源代码，后来 IP 核的仿真模型逐渐改为调用特定参数的硬件原语（Primitive）描述。IP 核生成时会自动产生扩展名为“.v”的 Verilog 仿真模型和扩展名为“.vhd”的 VHDL 仿真模型。这些仿真模型的共同特点是描述 IP 核的行为级功能。

做行为级前仿真时需要编译相应的 IP 核仿真模型或者硬件原语作为仿真库。Verilog 的仿真模型存放在目录\$XILINX/verilog/src/XilinxCoreLib 下，VHDL 的仿真模型存放在目录\$XILINX/vhdl/src/XilinxCoreLib 下（\$XILINX 为 Xilinx 的安装目录）。这些仿真模型随 IP 包升级而自动升级。仿真模型的具体编译方法详见第 3 章。IP 核在映射、布局布线后被实现为 Xilinx 底层的基本硬件单元，实现后提炼出的后仿真模型与其他设计没有本质区别，所以后仿真过程与其他设计完全一致。

2.6 测试激励生成器——HDL Bench

本节将介绍 ISE 中集成的测试激励生成器——HDL Bench 的使用方法。测试激励生成器帮助用户自动生成设计的测试激励，并调用仿真命令，完成设计的验证。

2.6.1 HDL Bench 综述

HDL Bench 是 ISE 中集成的一个比较有特色的仿真辅助工具。它将 VHDL 源代码、Verilog 源代码和 ECS 原理图等设计输入导入其测试环境，根据用户在图形界面下编辑的测试波形，直接生成测试激励文件，然后调用 ISE 中集成的仿真工具进行仿真验证。

测试激励文件的作用是在仿真过程中作为设计的激励源。ISE 中辅助设计测试激励分为两种方法。第一种方法是根据设计源文件的模块名称与输入输出端口生成测试激励的文件模板，使用该模板书写测试激励，省去了许多格式化部分，如文件头、测试激励模块名、测试激励端口信号、原设计模块的调用和信号的初始化等部分。

下面用一个简单的 VHDL 模块为例讲解测试激励文件模板的生成。在 Project Navigator 中生成测试激励模板的方法是在资源管理窗单击鼠标右键，在弹出的菜单中选择【New Source】命令，或者在工具栏点击新建资源快捷按钮，选择 Test Bench 型（Verilog Test Bench 或 VHDL Test Bench）测试激励资源，如图 2-86 所示。

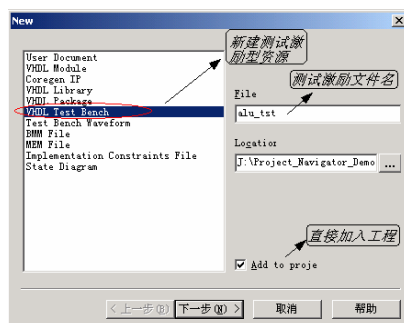


图2-86 新建测试激励类型资源

输入激励名称，选择【Add to Project】选项将直接加入工程，然后单击 **下一步(N) >** 按钮，选择需要仿真的模块，新建测试激励文件将与该模块关联，如图 2-87 所示。

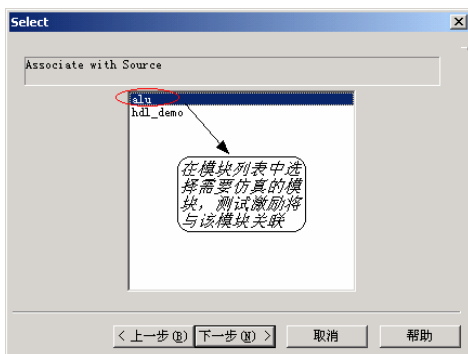


图2-87 选择需要仿真的模块

然后单击 **下一步(N) >** 按钮，进入新建资源信息对话框，单击 **完成** 按钮确认新建资源信息，生成测试激励模板。ISE 会在 HDL 编辑器中显示新建的测试激励文件，用户只需在“用户自定义”部分添加测试激励的行为，保存后直接被加入到当前工程中。这时可以看到在当前资源操作窗中有有效的操作是“ModelSim Simulator”，只要双击相应的仿真按钮，ISE 就会自动调用 ModelSim 对测试激励与原设计模块进行仿真，如图 2-88 所示。

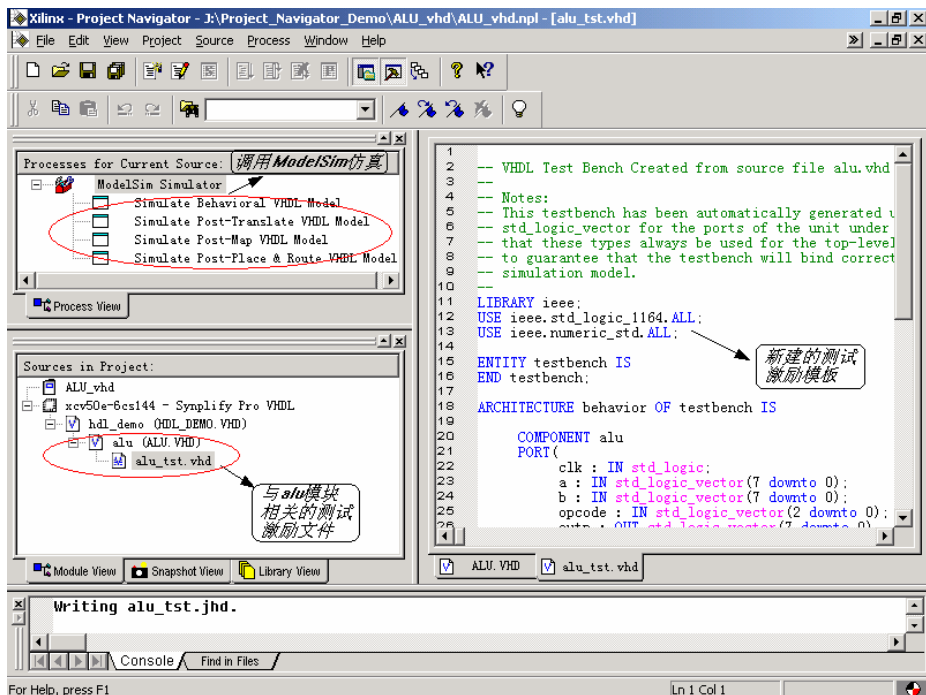


图2-88 测试激励文件与仿真操作

第二种方法是使用测试激励生成器生成测试激励文件。下节将详细讲述测试激励生成器的使用方法。

2.6.2 使用 HDL Benchner 生成测试激励

HDL Benchner 易学易用，所以在介绍 HDL Benchner 时不再赘述其界面与各个功能按钮，而是带领读者一起实际生成一个测试激励，以达到迅速掌握的目的。

上节在讲解生成测试激励模板时使用了 VHDL 模块的例子，本节使用一个 Verilog 的算术逻辑运算单元（ALU）为例，讲解如何用 HDL Benchner 生成测试激励文件。这个简化的 ALU 单元十分简单，仅有加、减、与、或、非 5 种基本运算。

⑧ 使用 HDL Benchner 生成用 Verilog 编写的“算术逻辑运算单元（ALU）”模块的测试激励，源代码与参考示例详见本书附带光盘的“Example-2-5”目录

HDL Benchner 的启动方法有两种：一种是在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【HDL Benchner】命令，另一种方法是在 Project Navigator 中新建“Test Bench Waveform”类型的资源。由于使用新建资源方法能直接将设计源文件导入到 HDL Benchner，并加入工程，而且在 ISE 中调用仿真工具也十分方便，所以建议使用新建资源方法启动 HDL Benchner。

1. 启动工程后，在资源管理窗单击鼠标右键，在弹出的菜单中选择【New Source】命令，弹出新建资源对话框，选择新建资源类型为“Test Bench Waveform”，键入测试激励文件名，如图 2-89 所示。

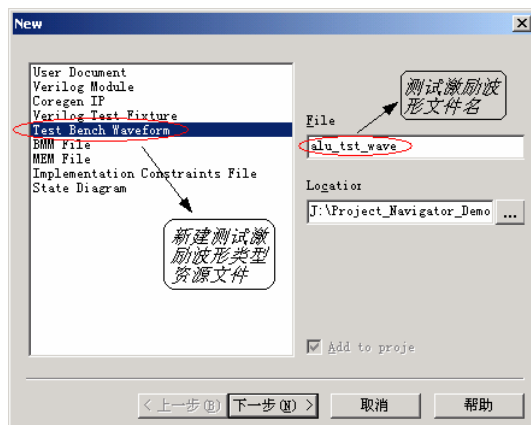


图2-89 新建测试激励波形资源

2. 单击 **下一步(N) >** 按钮，选择需要仿真的模块，新建测试激励波形文件将与该模块关联。本例仅有一个模块，所以默认选择需要仿真模块为“alu”。
3. 单击 **下一步(N) >** 按钮，进入新建资源信息对话框，单击 **完成** 按钮确认新建资源信息，启动 HDL Benchner。
4. 首先遇到的是初始时间设置对话框。这个对话框主要完成时钟周期、设计类型、全局信号等参数的设置。时钟周期需要设置的参数有：触发沿（上升沿触发、下降沿触发、两个沿都触发），时钟高、低电平保持时间，输入建立时

间，输出有效时间，时钟偏移等。设计类型分为 3 类：单时钟同步时序电路、多时钟同步时序电路和组合电路等类型。全局信号设置主要选择是否使用 GSR (FPGA)，PRLD (CPLD) 等全局置位、复位信号。另外还可以设置仿真所用的时间单位 (ms, ns, ps) 等项目，如图 2-90 所示。

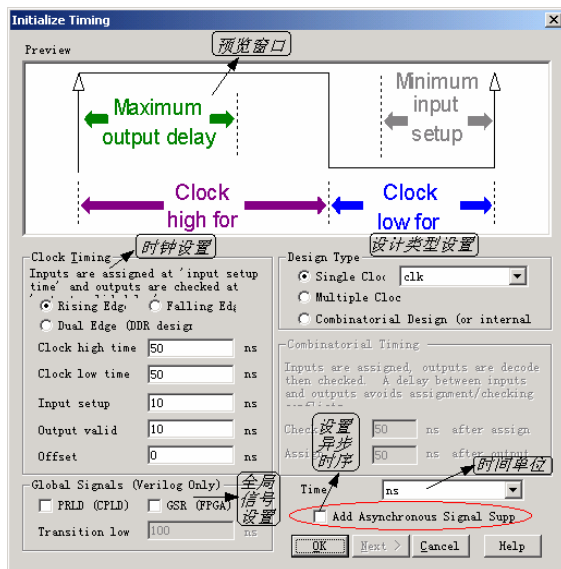


图2-90 时间设置窗

- 如果整个设计都是同步时序设计，那么单击 **OK** 按钮，完成时间设置，进入 HDL Benchner 测试波形编辑窗口。如果设计不完全是同步时序设计，就需要设置异步时序关系。在“Add Asynchronous Signal support”选项前打勾，单击 **Next >** 按钮，进入主时钟选择对话框。如果设计有多个时钟，需要选择主时钟，主时钟的周期特性将决定异步组合电路的时序是否满足条件。本例只有一个时钟，选择“clk”为主时钟，如图 2-91 所示。

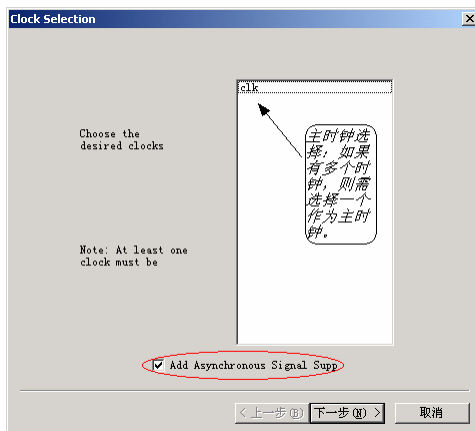


图2-91 选择设计的主时钟

- 单击 **下一步(N) >** 按钮，选择信号属性，即将信号分为同步时序信号和异步时序信

号两类。设置异步时序信号的步骤为：在时钟列表中，选择异步时序信号（Asynchronous Signals），然后在未选择信号列表中选择设计中所有异步时序信号，单击 **<-- Add** 按钮加入到已选择信号列表中。本例输出信号“outp_a”是异步信号。

设置同步时序信号的步骤为：在时钟列表中，选择主时钟信号，然后在未选择信号列表中选择设计中所有同步时序信号，单击 **<-- Add** 按钮加入到已选择信号列表中。本例“a”、“b”、“opcode”和“outp_s”是同步时序信号，如图 2-92 所示。

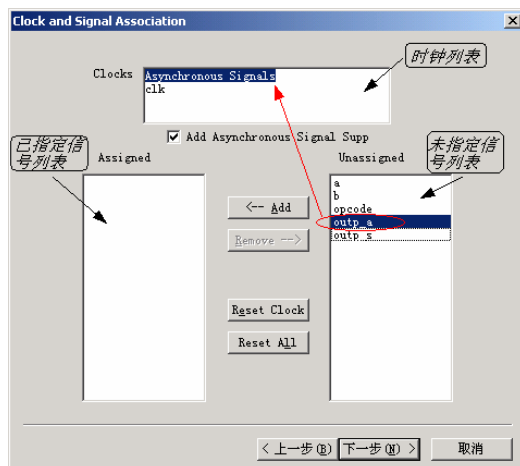


图2-92 选择信号属性

7. 然后单击 **下一步(N) >** 按钮，设置时钟属性。这个窗口与图 2-90 中时钟属性设置一致，完成时钟的触发沿（上升沿触发、下降沿触发、两个沿都触发），时钟高、低电平保持时间，输入建立时间，输出有效时间，时钟偏移等属性的设置，如图 2-93 所示。

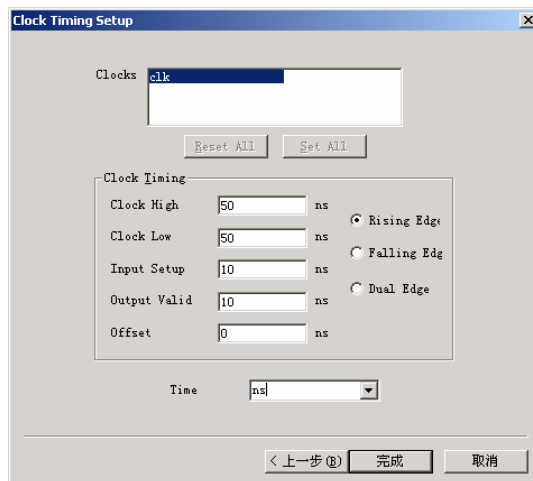


图2-93 时钟属性设置

8. 单击 **完成** 按钮，完成异步时序设置，进入 HDL Bencher 测试波形编辑窗口。整个窗口分为两个部分，上面是测试激励波形编辑窗，下面是源代码显示窗，如图 2-94 所示。

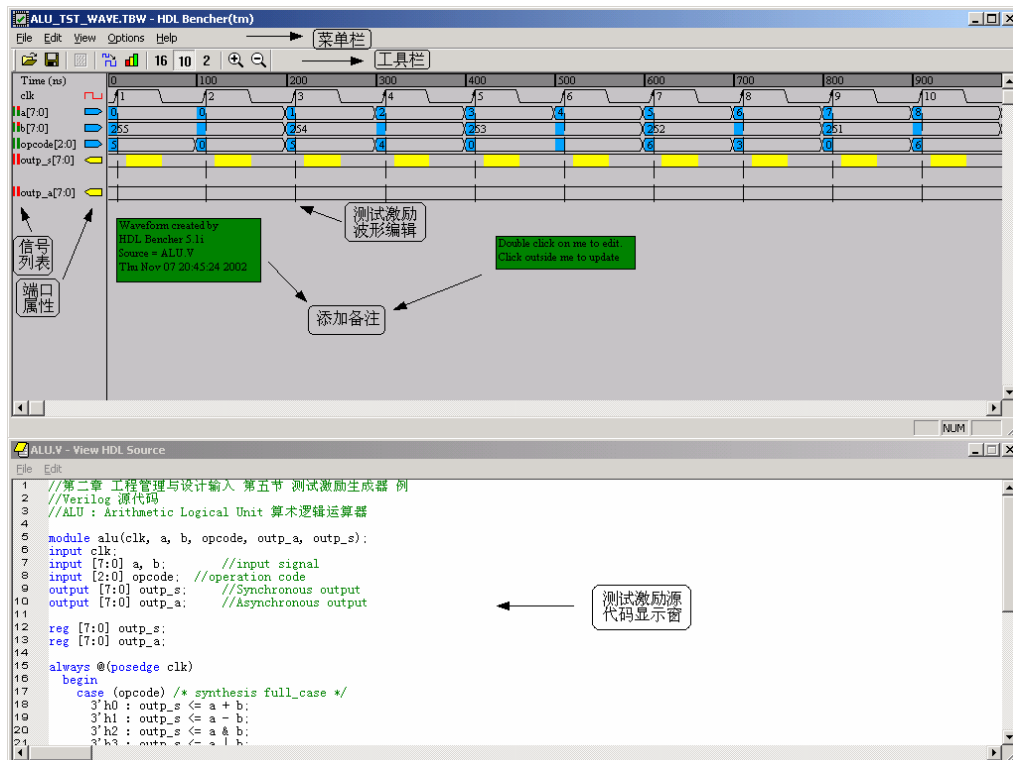


图2-94 测试激励波形编辑窗

测试激励波形编辑窗简单明了，由菜单栏、工具栏、信号列表、端口属性和信号波形编辑区等项目组成。菜单栏由一系列通用操作命令下拉菜单组成。工具栏包含了 HDL Bencher 中常用命令的快捷按钮。信号列表罗列了所有仿真信号，端口属性用图标区分了时钟、输入、输出信号。信号波形编辑区是 HDL Bencher 的主体，完成激励信号的波形编辑。

测试激励信号波形的编辑有许多方法，其中使用波形向导“Pattern Wizard”编辑测试激励的波形是一种省时省力的好方法。具体操作如下：在波形编辑区双击某个信号，出现 **Pattern** 按钮，如图 2-95 所示。单击 **Pattern** 按钮，呼出波形向导。波形向导界面基本由选择波形、波形说明、进制表示、测试循环数目和参数设置等项目构成。波形向导有许多常用波形可供选择，包括翻转变换、递增、递减、循环左移、循环右移和随机数等波形变化方式。在波形说明窗对所选择波形变化方式给予文字说明。进制表示可以设为 16、10 或 2 进制等 3 种方式。测试循环数目可以任意设置。根据图 2-96 所示，设置波形向导的参数，然后单击 **OK** 按钮确认测试激励波形的编辑。

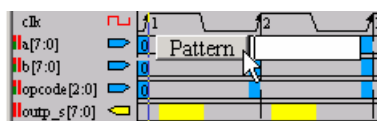


图2-95 呼出波形向导按钮

9. 同上方法，按图 2-97 所示设计输入信号“b”的波形。

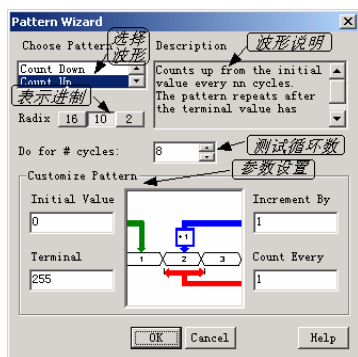


图2-96 使用波形向导设计信号“a”的波形

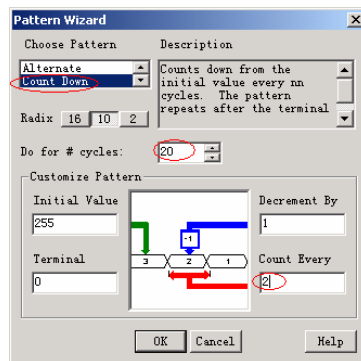


图 2-97 使用波形向导设计信号“b”的波形

10. 同上方法，按图 2-98 所示设计运算控制信号“opcode”的波形。
11. 完成激励信号波形输入后单击工具栏中的 按钮，保存设计。接着单击工具栏中的 按钮，查看测试激励的覆盖率，如图 2-99 所示。

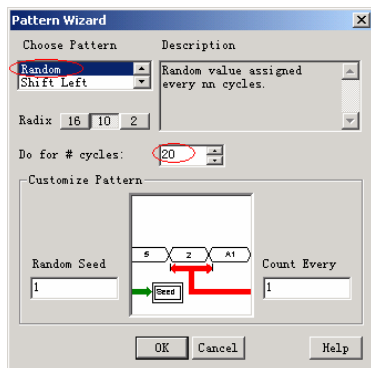


图 2-98 使用波形向导设计信号“opcode”的波形

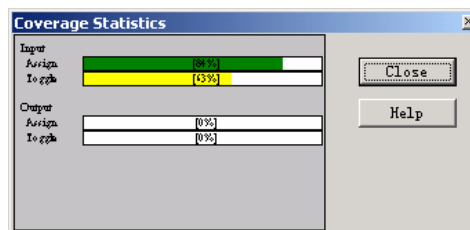


图 2-99 测试激励覆盖率

激励可测试的情况越多，覆盖率就越高。如果覆盖率不满意，就需要重新更改测试激励波形。

12. 完成测试激励波形的编辑后，关闭 HDL Bencher 退出测试激励编辑器，会发现在 Project Navigator 的资源管理窗中新建了一个测试激励波形资源，如图 2-100 所示，双击测试激励波形资源，可以启动 HDL Bencher 修改测试激励波形。

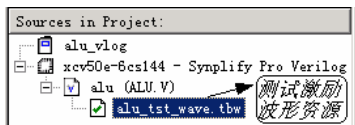


图2-100 新建的测试激励波形资源

13. 选择测试激励波形资源，发现当前资源操作窗中对该资源的操作是仿真。如果要做前行为级仿真就双击行为级仿真“Simulate Behavioral Verilog Model”命令，如图 2-101 所示。

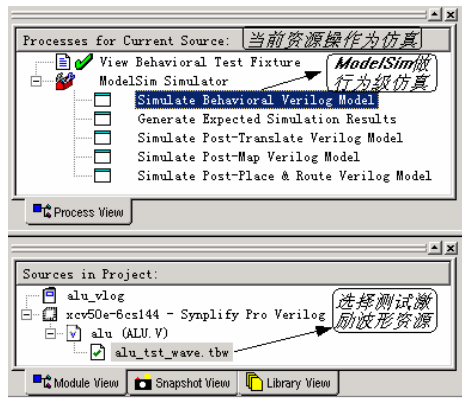


图2-101 使用测试激励波形启动仿真

14. ISE 自动调用 ModelSim 仿真工具，以测试激励波形为测试激励，对设计进行仿真，如图 2-102 所示。

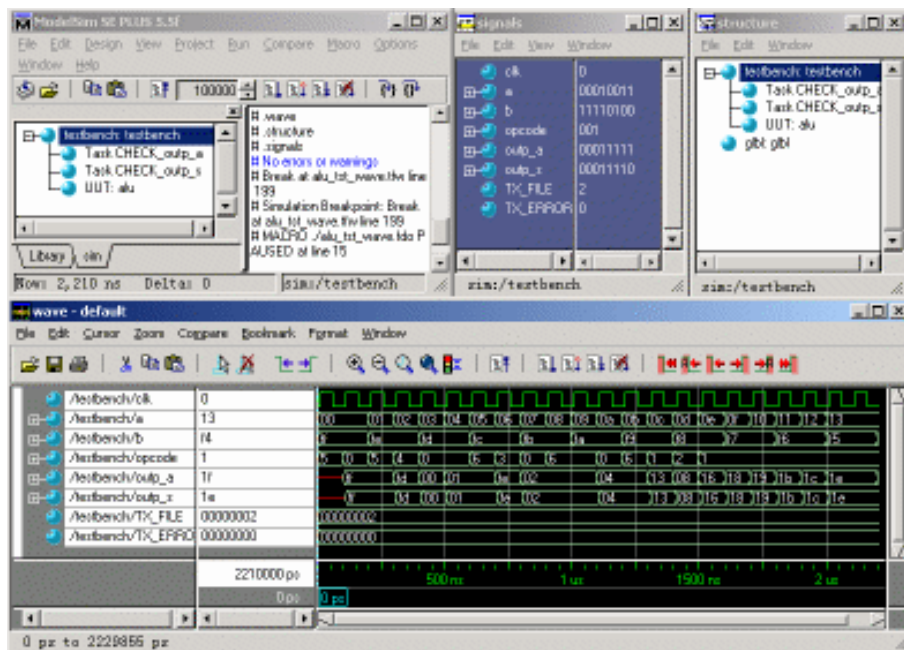


图2-102 ISE 调用 ModelSim 进行仿真

ModelSim 仿真工具的详细介绍请参考第 3 章。至此使用 HDL Benchner 生成测试激励波形并仿真的所有步骤已经完成。使用 HDL Benchner 生成测试激励波形并仿真基本可以分为 4 大步骤：第一步，新建测试激励波形资源；第二步，设置时间条件，包括时钟设置与异步时序设置两个部分；第三步，编辑测试激励波形；第四步，以波形为测试激励，在 ISE 中调用 ModelSim 等第三方仿真工具对设计进行仿真。

2.7 设计结构向导——Architecture Wizard

本节将介绍 ISE 新集成的设计工具——Architecture Wizard，它能辅助设计数字时钟管理模块（DCM，Digital Clock Manager）和高速 I/O 收发器（Rocket I/O™ transceivers）。

2.7.1 Architecture Wizard 综述

Architecture Wizard 是从 ISE 5.1 起集成的实用辅助设计工具。其基本功能是辅助设计者生成 DCM 和 Rocket I/O™ transceivers 模块单元。DCM 即数字时钟管理模块，是 Xilinx 器件的 DLL（数字时钟锁相环）的发展与升级，使用 DCM 可以在相当高的精度下，完成时钟的分频、倍频、移相和整形等功能。而 Rocket I/O™ 是 Xilinx 为了满足日益提高的数据通信速率而在新型 FPGA 器件上设计的超高速（在 Gbit/s 数量级）I/O 端口标准。

目前 Architecture Wizard 支持的器件族有 Virtex-II 和 Virtex-II Pro。根据器件族结构，Virtex-II Pro 器件可以生成 DCM 和 Rocket I/O™ transceivers 模块，而 Virtex-II 器件只能生成 DCM 模块。Architecture Wizard 的输出是用 Verilog HDL 和 VHDL 语言描述的，并能根据所需综合工具的区别自动调整语法表达方式。



仅 ISE 5.1 及更高版本包含 Architecture Wizard 设计向导。

2.7.2 Architecture Wizard 使用方法

Architecture Wizard 的使用非常简单，如果不区分 DCM 和 Rocket I/O™ transceivers 模块与 IP 核，那么 Architecture Wizard 不论从界面风格还是使用方法上都与 Core Generator 十分相似。都是选择器件与输出方式后，在生成向导的引导下设计模块参数，然后自动生成模块。

启动 Architecture Wizard 的方法也有两种：一是在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【Architecture Wizard】命令。另一种方法是在 Project Navigator 中新建 Architecture Wizard 类型的资源。使用 Project Navigator 新建资源方法启动 Architecture Wizard，该工具能够自动根据工程属性设置的器件族信息确定需要生成 DCM 还是 Rocket I/O™ transceivers，并能直接将生成资源（XAW 文件）加入到工程中去。

 **使用 Architecture Wizard 生成 DCM 和 Rocket I/O™ transceivers 模块，参考示例详见本书附带光盘的“Example-2-6”目录**

下面用新建资源方法新建 DCM 和 Rocket I/O™ transceivers 模块。新建一个工程，为了能够同时演示生成上述两种模块，选择器件族为 Virtex-II Pro，器件型号和综合工具可任选，如图 2-103 所示。

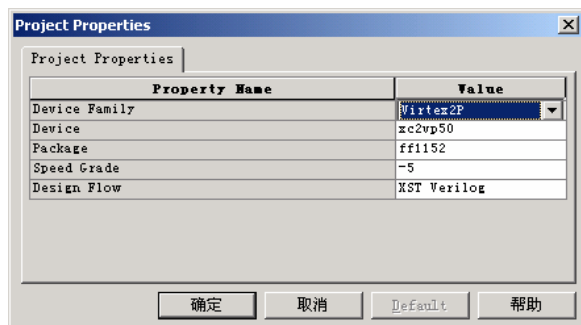


图2-103 设置器件族为 Virtex2P

1. 在 Project Navigator 中新建 Architecture Wizard 资源, 如图 2-104 所示。
2. 单击 **下一步(N) >** 按钮, 显示新建资源信息, 单击 **完成** 按钮确认新建资源信息, 启动 Architecture Wizard, 选择需要新建的模块类型, 如图 2-105 所示。

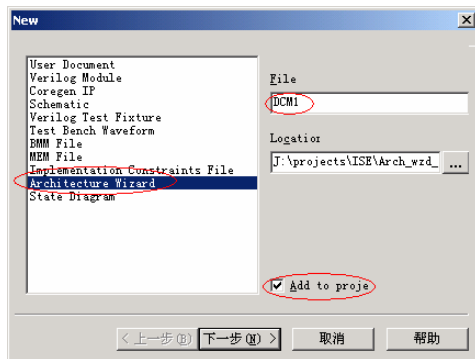


图2-104 新建 Architecture Wizard 资源



图 2-105 选择 DCM 类型

3. 单击 **OK** 按钮启动 DCM 向导。DCM 向导界面与 Core Generator 生成 IP 核的界面非常相似, 如图 2-106 所示, 设置 DCM 输入时钟频率、时钟源、反馈、移相、占空比整形和输出管脚等参数。

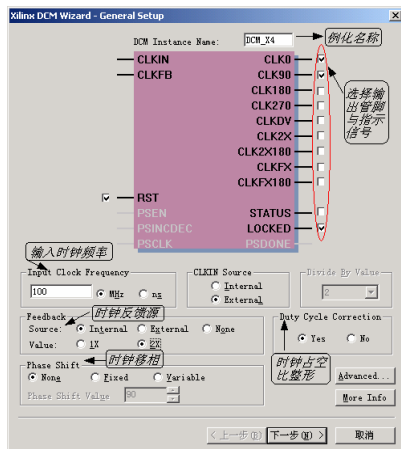


图2-106 DCM 参数设置

4. 单击 **Advanced...** 按钮，设置 DCM 锁定信号顺序和输入是否 2 分频等高级参数，如图 2-107 所示。



图2-107 设置高级参数

5. 单击 **OK** 按钮确认高级参数设置，然后单击 **下一步(N) >** 按钮确认其他参数设置，进入全局时钟设置界面。为了达到更好的性能与精度，Xilinx 强烈建议 DCM 模块时钟信号使用全局时钟缓冲 (BUFGP, Global Buffer)，单击 **完成** 按钮使用全局时钟资源，完成 DCM 模块的生成。全局时钟资源的详细论述请读者参考第 4.4 节。

接着带领读者生成 Rocket I/O™ transceivers 模块。

6. 同前所述，新建 Architecture Wizard 资源，在图 2-105 所示处，选择生成 Rocket I/O™ transceivers 资源，单击 **OK** 按钮启动 Rocket I/O 向导。根据所选 Virtex-II Pro 器件型号，传输速率为 3.125Gb/s 或 10Gb/s，可选多种转发器类型和匹配阻抗，如图 2-108 所示。
7. 单击 **下一步(N) >** 按钮，继续设置预加重程度，输出电平选择 CRC 校验方式，如图 2-109 所示。

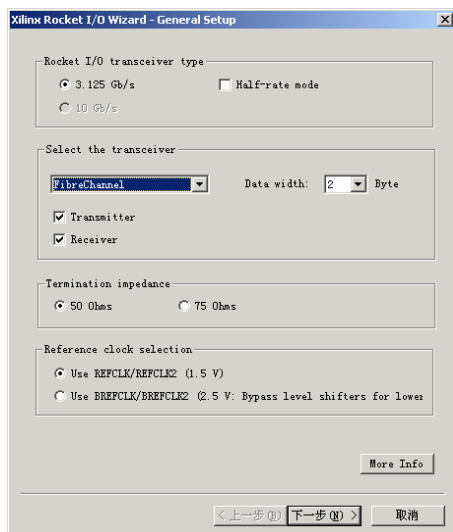


图2-108 设置 Rocket I/O 参数之一

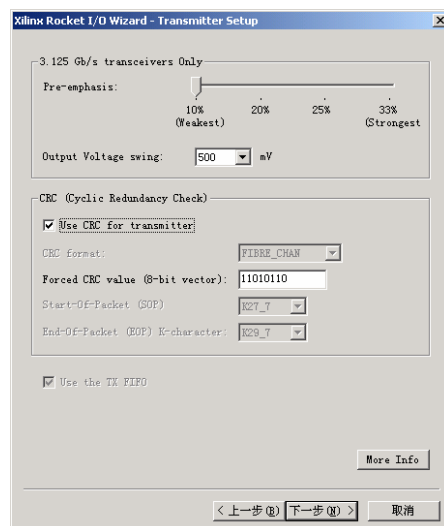


图 2-109 Rocket IO 参数设置之二

8. 单击 **下一步(N) >** 按钮，设置时钟纠错和接收机失步处理方式等参数，如图 2-110 所示。

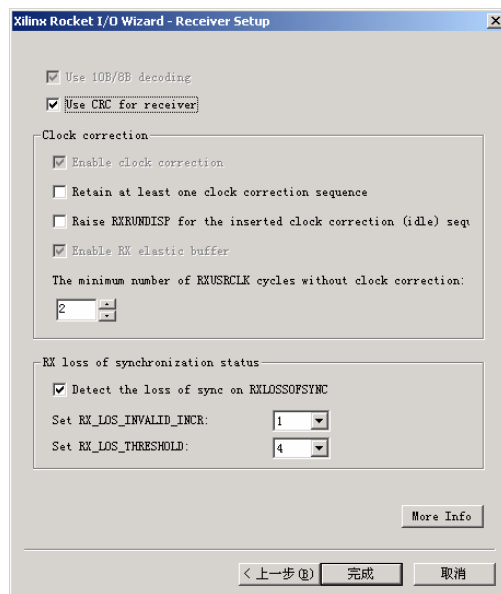


图2-110 Rocket IO 参数设置之三

9. 单击 **完成** 按钮确认所选参数，完成 Rocket IO 模块的生成。

2.8 小结

本章对 ISE 的工程管理器 (Project Navigator) 进行了详细介绍，讨论在 Project Navigator 中实现完整的 FPGA/CPLD 设计流程的方法。同时介绍了源代码编辑器 (HDL Editor)、状态机编辑器 (StateCAD)、原理图编辑器 (ECS)、IP 核生成器 (CORE Generator)、测试激励生成器 (HDL Bencher) 和设计结构向导 (Architecture Wizard) 等工具的使用方法。

2.9 问题与思考

1. 结合 FPGA/CPLD 设计流程，熟悉工程管理器 Project Navigator 的基本使用方法。
2. 简述 ISE 中集成的设计输入工具的名称、特点与基本使用方法。
3. Language Templates 中有哪几类可用辅助模板？
4. 简述使用 StateCAD 设计、仿真状态机的基本步骤。
5. 在 ISE 中如何完成多元混合输入设计？
6. 简述生成 IP Core 的基本步骤和 IP Core 的综合、仿真方法。

第3章 仿真工具

上一章讲述了如何在 ISE 集成环境中建立并管理一个 FPGA/CPLD 的工程项目，介绍了各种设计输入工具的使用方法。在讲解的过程中列举了一些设计程序，那么这些设计程序是否满足设计者最初的设计构想呢？这些设计程序配置到逻辑芯片之后是否可以运行呢？这些问题需要通过逻辑仿真和实际的验证加以解决。

在软件设计中，设计验证的最基本方法是观察程序在计算机上的设计输出，例如每一个刚开始学习 C 语言的人，遇到的第一个程序通常都是“HELLO WORLD”，这个程序的最终目的是在计算机屏幕上输出一个“HELLO WORLD”字符串。为了达到这个目的，设计者需要通过一个文本编辑器编辑 C 语言代码，然后使用特定的工具对代码进行编译、实现，最后得到一个可执行的文件，如果运行这个程序之后在计算机屏幕上可以看到“HELLO WORLD”字样，那么就说明程序正确了，反之就需要对 C 语言代码进行修改，这个过程是软件测试的一般方法。对于可编程逻辑的编程者来说，为了验证编写的程序是否正确也需要一个类似的过程，所不同的是硬件描述语言的输出不能直接在计算机操作系统上看到结果，而需要通过一个硬件描述语言的模拟器来完成。这种模拟器一般也叫做仿真器，其实这两个概念有所不同，本书不加以区分。常用的硬件描述语言的仿真器有很多种，例如 VCS、Ncsim、Affirima、Verilog-XL、SpeedWave、Finsim 和 ModelSim 等，根据所使用的编程语言的不同可以将这些工具分为 Verilog 语言仿真器和 VHDL 语言仿真器，也可以根据工作方式的不同分为事件驱动（Event-Driven）的仿真器和时钟驱动（Cycle-Driven）的仿真器等类型，这些工具中有些侧重于 IC 设计，而有些则侧重于 FPGA/CPLD 设计。其中 ModelSim 仿真器在 FPGA/CPLD 设计中使用得比较广泛，这是因为 Model Technology 公司为各个 FPGA/CPLD 厂家都提供了 OEM 版本的 ModelSim 工具，同时 Model Technology 公司也对 ModelSim 工具进行了大力推广。ModelSim 仿真器是基于事件驱动的，它可以用来仿真 Verilog 语言，也可以用来仿真 VHDL 语言，同时也支持两种语言的混合仿真。

在整个设计流程中，完成了设计输入并成功地进行了编译，只能说明设计符合一定的语法规则，并不能保证设计可以获得所期望的功能，这时就需要通过仿真对设计进行验证。数字电路设计中一般有源代码输入、综合、实现等 3 个比较大的阶段，而电路仿真的切入点也基本与这些阶段相吻合。根据设计阶段不同，仿真可以分为 RTL 行为级仿真、综合后门级功能仿真和时序仿真等 3 大类型。这种仿真的切入不仅适合 FPGA/CPLD 设计，同样适合 IC 设计。数字电路设计中的主要仿真切入点如图 3-1 所示。

在大部分设计中执行的第一个仿真将是 RTL 行为级仿真。这个阶段的仿真可以用来检查代码中的语法错误以及代码行为的正确性，其中不包括延时信息。如果没有实例化一些与器件相关的特殊底层元件的话，这个阶段的仿真可以做到与器件无关。因此在设计的初期阶

段不使用特殊底层元件即可以提高代码的可读性、可维护性，又可以提高仿真效率，且容易被重用。

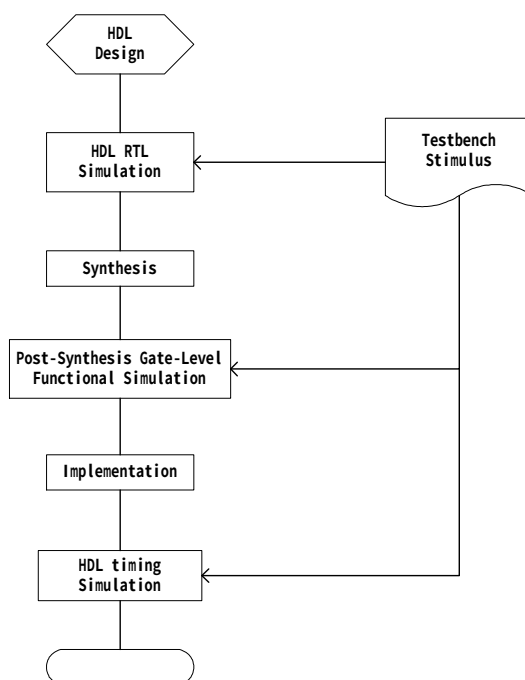


图3-1 数字电路设计中主要仿真切入点

设计流程中的第二个仿真是综合后门级功能仿真。绝大多数的综合工具除了可以输出一个标准网表文件以外，还可以输出 Verilog 或者 VHDL 网表，其中标准网表文件是用来在各个工具之间传递设计数据的，并不能用来做仿真使用，而输出的 Verilog 或者 VHDL 网表可以用来仿真，之所以叫门级仿真是因为综合工具给出的仿真网表已经是与生产厂家的器件的底层元件模型对应起来了，所以为了进行综合后仿真必须在仿真过程中加入厂家的器件库，对仿真器进行一些必要的配置，不然仿真器并不认识其中的底层元件，无法进行仿真。Xilinx 公司的集成开发环境 ISE Project Navigator 中并不支持综合后仿真，而是使用映射前门级仿真代替，对于 Xilinx 开发环境来说，这两个仿真之间差异很小。

设计流程中的最后一个仿真是时序仿真。在设计布局布线完成以后可以提供一个时序仿真模型，这种模型中包括了器件的信息，同时还会提供一个 SDF 时序标注文件（Standard Delay format Timing Annotation），SDF 时序标注最初使用在 Verilog 语言的设计中，现在 VHDL 语言的设计中也引用了这个概念。对于一般的设计者来说并不需知道 SDF 文件的详细细节，因为这个文件一般由器件厂家提供给设计者。例如，在 FPGA/CPLD 设计中，SDF 时序标注文件都是器件厂家通过自己的开发工具提供给设计者的，在 Xilinx 公司的 FPGA/CPLD 设计中使用 SDF 作为时序标注文件的扩展名，在 Altera 公司的 FPGA 设计中使用 SDO 作为时序标注文件的扩展名。在 SDF 时序标注文件中对每一个底层逻辑门提供了 3 种不同的延时值，分别是典型延时值、最小延时值和最大延时值，在对 SDF 标注文件进

行实例化说明时必须指定使用了哪一种延时值。虽然在设计的最初阶段就已经定义了设计的功能，但是只有当设计布局布线到一个器件中之后，才会得到精确的时延信息，在这个阶段才可以模拟到比较接近实际电路的行为。

Xilinx 在 3.3 版本以前的 Foundation 软件中嵌套了 Aldec 公司的 Active HDL 工具，而在此后版本的 Foundation 软件以及 ISE 集成开发环境中都使用了 Model Technology 公司的 ModelSim XE (Xilinx 公司 OEM 版本的 ModelSim) 作为软件嵌套的仿真工具，本章将对 ModelSim 的使用方法做详细的介绍。

本章主要内容如下：

- ModelSim 的用户接口；
- ModelSim 仿真环境的建立；
- ModelSim 仿真过程的简单示例；
- VHDL 和 Verilog 混合仿真；
- ModelSim 中的调试方法；
- 使用批处理模式进行仿真；
- 波形比较与 WLF 文件；
- 仿真中的代码覆盖率；
- ModelSim 中的 VCD 命令以及 VCD 文件。

3.1 ModelSim 的用户接口

ModelSim 是一个独立的仿真工具，它在工作的时候并不需要其他软件的协助，在 Xilinx 公司的 ISE 集成开发环境中给 ModelSim 仿真软件预留了接口，通过这个接口可以从 ISE 集成环境中直接启动 ModelSim 工具进行仿真。这个过程通常会给初学者一个错觉，以为 ISE 集成环境和 ModelSim 工具是联合工作的，其实 ISE 并没有集成 ModelSim 工具，只是预留了软件接口，它们之间是如何工作的，这个问题将在 3.3 节中使用两个不同的例子进行讲述。本节为了说明 ModelSim 的用户接口，将使用从 ISE 集成开发环境中直接启动 ModelSim 仿真工具的方法。使用此种方法启动 ModelSim 工具需要具备 3 个条件：第一，启动 ISE 集成开发环境并建立了一个 FPGA/CPLD 的工程项目；第二，添加设计源代码并且编译通过；第三，使用 ISE 中的 Test Fixture 或者 Test Bench Waveform 工具为当前的设计提供一个测试模板 (Testbench)，并且在测试模板中添加设计激励 (具体使用参见第 2 章)。只有上述条件具备之后才可以从 ISE 的当前资源操作窗中直接启动 ModelSim 工具并运行仿真，以下使用 ISE 自带的一个例子加以说明。

在 ISE 中直接启动 ModelSim

1. 在 Windows 操作系统中选择【开始】/【程序】/【Xilinx ISE 6】/【Project Navigator】命令，启动 ISE 集成开发环境。
2. 在 ISE 主窗口中选择【File】/【Open Example】命令，弹出【Open Example】对话框，如图 3-2 所示，然后在【Select an Example Project】栏目中选择“gold_code_ver_217”，在【Destination Directory】中选择项目存放的目录，

单击 **OK** 按钮打开例子程序。

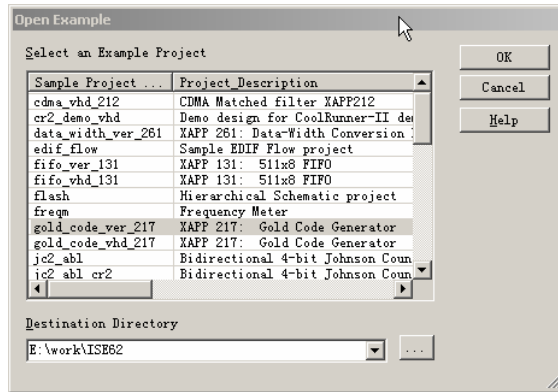


图3-2 【Open Project】对话框

3. 在资源管理窗口（Source in Project）中的模块视图（Module View）中选中测试文件“testbench.tf”，在相应的当前资源操作窗口（Processes for Current Source）中将会出现与 ModelSim 仿真器相关的行为仿真（Simulate Behavioral Verilog Model）、翻译后仿真（Simulate Post-Translate Verilog Model）、映射后仿真（Simulate Post-Map Verilog Model）和布局布线后仿真（Simulate Post-Place & Route Verilog Model）等 4 个不同的操作选项，如图 3-3 所示。

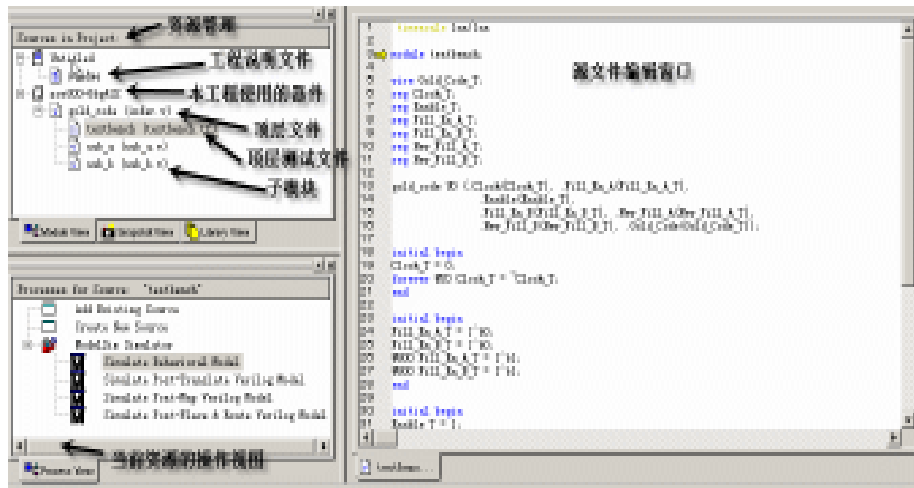


图3-3 在 ISE 中打开“gold_code_ver_217”工程项目

4. 双击【Simulate Behavioral Model】操作选项，将启动 ModelSim 仿真器。
5. 在 ModelSim 的主窗口中选择【View】/【All】命令，将显示所有的窗口。

在当前资源操作窗口中选择任何一个操作选项都可以启动相应阶段的仿真操作，如果相应阶段需要的仿真文件不存在，那么集成环境将自动生成仿真文件。例如，当双击当前资源操作窗口中的【Simulate Post-Place & Route Verilog Model】操作选项时将启动时序仿真，而这一仿真过程需要的布局布线后仿真文件以及时序标注文件都还没有产生，那么 ISE 集成开发环境将自动开始对这个工程进行编译、综合、翻译、映射和布局布线等操作，得到布局布

线后仿真文件以及时序标注文件，然后再启动 ModelSim 仿真器进行时序仿真（时序仿真需要的仿真库文件请参考 3.3 节）。

下面以一个异步 FIFO 为例讲述【ISE Project Navigator】集成环境中与仿真有关的大部分选项的设置。异步 FIFO 是数字电路的一个典型的设计，因为 FIFO 的读写使用了异步的时钟，所以设计的难度大大增加。在最初学习数字电路设计时很多初学者都想当然地对异步 FIFO 进行编码，其实绝大部分的设计方法都有一些理论支持来保证设计的正确性。

3.1.1 行为仿真

1. 启动 Project Navigator 集成开发环境，在主窗口中选择【File】/【Open examples】命令，打开例子选择窗口，在其中选择【fifo_ver_131】工程，单击  按钮，打开工程。
2. 在这个工程中包括了两个顶层文件“fifoctrl_cc”和“fifoctrl_ic”，其中“fifoctrl_cc”文件实现了一个同步 FIFO，“fifoctrl_ic”文件实现了一个异步 FIFO 的设计。
3. 选择顶层文件“fifoctrl_ic”，这个文件包括了 3 个子模块，分别为【muxor_p】、【xor4_p】和【xor5_p】。同时包括了两个功能仿真文件【tb_ic_func1】、【tb_ic_func2】，两个时序仿真文件【tb_ic_time1】和【tb_ic_time2】。选中顶层文件后，可以在【Process View】窗口中对一个属性进行配置。
4. 选择功能仿真文件【tb_ic_func1 (fifoctrl_ic_tb2.tf)】，然后在相应的【Process View】窗口中选中【Simulate Behavioral Model】仿真的阶段，单击鼠标右键，在弹出菜单中选择【Properties】命令，如图 3-4 所示。

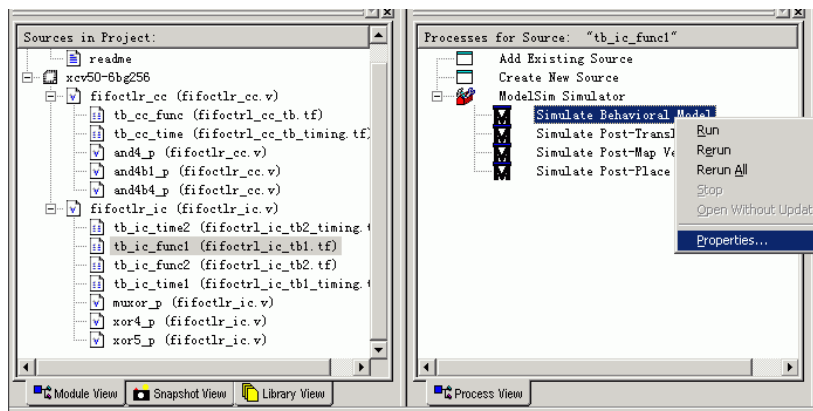


图3-4 在 ISE 中对前仿真属性进行设置

5. 以上操作将打开程序属性对话框，在这个对话框中有两个不同的选项卡如图 3-5 所示。在【Simulation Properties】中有以下 4 个不同的选项。
 - Use Custom Do File
 - Custom Do File
 - Use Automatic Do File
 - Simulation Run Times

在这些选项中可以选择不使用定制的 DO 文件，是否使用自动的 DO 文件。如果选择了使用自动的 DO 文件，针对于这个例子来说程序将在当前目录下自动生成一个【tb_ic_func1.fdo】文件。【Displays Properties】选项卡用来设置显示的属性，其中包含以下 8 个复选框。

- Signal window
- Wave window
- Structure window
- Source window
- List window
- Variables window
- Process window
- Data Flow window

这些复选框分别用来定义是否在仿真过程中自动打开相应的窗口。

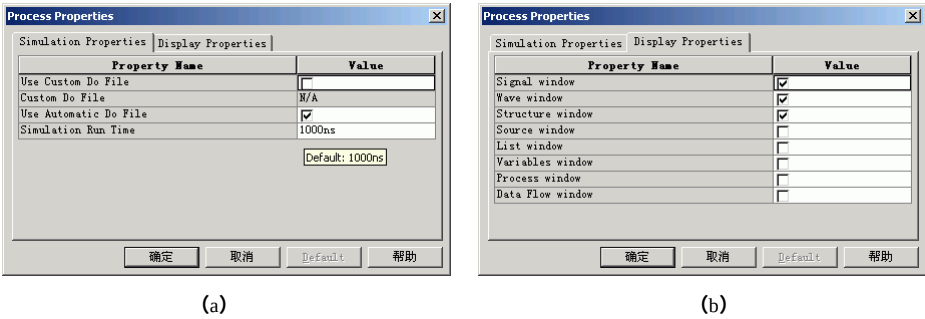


图3-5 ISE 中行为仿真属性对话框

6. 选择完成之后，可以进行相应的行为仿真。

在以上的操作中，程序在当前目录下自动生成一个【tb_ic_func1.fdo】文件，具体内容如下：

```
vlib work
vlog fifoctrl_ic.v
vlog fifoctrl_ic_tb1.tf
vlog D:/Xilinx/verilog/src/glbl.v
vsim -t 1ps -L xilinxcorelib_ver -L unisims_ver -lib work
tb_ic_func1 glbl
do tb_ic_func1.udo
view wave
add wave *
view structure
view signals
run 1000ns
```

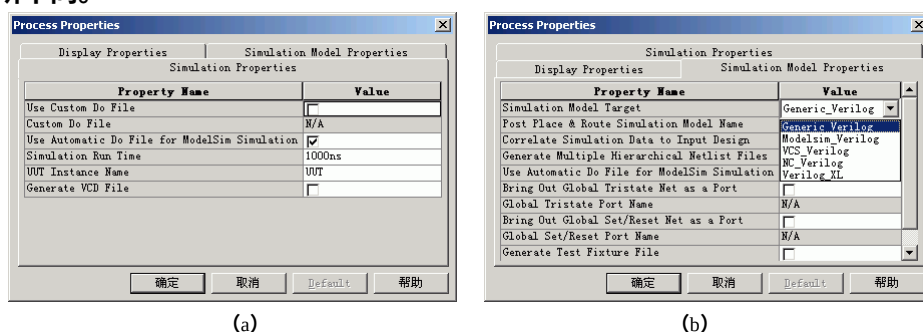
可以看到 DO 文件中包含了一系列 ModelSim 的命令，DO（宏）文件就是将 ModelSim 的命令集合在一起使用的一个命令列表文件，对以上的命令解释如下：

- vlib work：建立一个工作目录。
- vlog f1foctrl_ic.v：编译 Verilog 源文件。
- vlog f1foctrl_ic_tb1.tf：编译测试文件。
- vlog D:/Xilinx/verilog/src/glbl.v：编译全局复位以及三态信号文件。
- vsim -t 1ps -L xilinxcorelib_ver -L unisims_ver -lib work tb_ic_func1 glbl：使用相应的库执行仿真，并设置仿真精度。
- do tb_ic_func1.udo：执行 Project Navigator 软件生成的默认用户 DO 文件。
- view wave：打开波形窗口。
- add wave *：将顶层信号添加道波形窗口中。
- view structure：打开结构窗口。
- view signals：打开信号窗口。
- run 1000ns：运行仿真 1000ns。

用户通过 GUI 界面设置的 ModelSim 程序属性都可以在此类文件中找到对应的命令。例如，如果选择了【Use Custom Do File】选项，就需要在【Custom Do File】的【Value】域中给出定制 DO 文件的位置，指定了 <user_do.do> 文件以后，程序自动生成的【tb_ic_func1.fdo】中就会多出一个命令 do <user_do.do>。如果在图 3-5 (b) 所示的复选框中进行了选择，在 fdo 文件中就会相应地产生这些命令。

3.1.2 时序仿真

如图 3-4 所示，选择时序仿真文件【tb_ic_time1 (f1foctrl_ic_tb1_timing.tf)】，然后在相应的【Process View】窗口中选中【Simulate Post-Place & Route Verilog Model】仿真，单击鼠标右键，在弹出菜单中选择【Properties】命令，打开的属性对话框如图 3-6 所示，与图 3-5 相比多了【Simulation Model Properties】选项卡，同时【Simulation Properties】选项卡的属性也有所不同。



(a)

(b)

图3-6 ISE 中时序仿真属性对话框

【Simulation Properties】选项卡中多了以下两个项目，其他选项与行为仿真属性的相关选项卡项目相同。

- UUT Instance Name：用来定义实例化名。
- Generate VCD File：是否产生 VCD 文件。

【Simulation Model Properties】选项卡中各个选项如下。

- Simulation Model Target
- Post Place & Route Simulation Model Name
- Correlate Simulation Data to Input Design
- Generate Multiple Hierarchical Netlist Files
- Use Automatic Do File for ModelSim Simulation
- Bring Out Global Tristate Net as a Port
- Global Tristate Port Name
- Bring out Global Set/Reset Net as a port
- Global Set/Reset Port Name
- Generate Test Fixture File

可以看到在【Simulation Properties】选项卡和【Simulation Model Properties】选项卡中都有【Use Automatic Do File for ModelSim Simulation】参数，原因将在高级模式设置中进行说明。

以上这些都是在 Project Navigator 标准模式下关于 ModelSim 仿真器的一些设置，如果使用 Project Navigator 的高级模式可以有更为详细的设置过程。

3.1.3 高级设置

在 Project Navigator 集成环境中使用【Edit】/【Preferences】命令打开属性选择窗口，如图 3-7 所示。将【Processes】选项卡中的【Process Settings】设置为【Advanced】，单击 **确定** 按钮保存之后就可以启动 Project Navigator 高级操作模式。

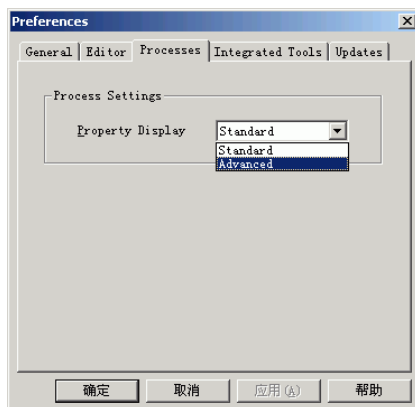


图3-7 ISE 属性选择窗口

在高级操作模式下，【Displays Properties】选项卡与标准模式相比没有发生任何变化，而其他两个选项卡都发生了变化。

在【Simulation Properties】选项卡中除了包括标准模式下的选项外还增加了如下选项。

- Other VSIM Command Line Options：输入其他的 VSIM 命令行参数，多个参数之间使用空格。
- Other VLOG Command Line Options：输入其他的 VLOG 命令行参数，多个参数之间使用空格。

- Other VCOM Command Line Options：输入其他的 VCOM 命令行参数，多个参数之间使用空格。
- Simulation Resolution：指定仿真器的仿真时间精度，开始仿真后可以使用仿真器的【report】命令检查当前的仿真时间精度。
- Simulation Mode：设置仿真延时模型，使用最大、最小或者典型的延时模型，在默认条件下，SDF 中的三类延时都使用最大延时。
- VHDL Syntax：指定 VHDL 的语法版本，默认使用 93 版。

在【Simulation Model Properties】标签页面中各个选项说明如下。

- Simulation Model Target：指定产生的网表所适用的仿真器。
- Post Place & Route Simulation Model Name：指定仿真模型名称。
- Rename Top Level Module To：指定使用 NetGen 程序产生的顶层模块的名称，默认条件下，输出的模块名将继承输入设计的顶层文件名称。
- Change Device Speed To：改变生成仿真模型时使用的目标器件的速度。
- Correlate Simulation Data to Input Design：该参数需要配合【Retain Hierarchy】参数以及设计源代码一起使用，如果该参数以及【Retain Hierarchy】参数都设置为【True】，同时在源代码中都使用【Keep Hierarchical】属性，就可以在时序仿真时将输出的时序仿真模型的信号与源代码中的信号关联起来。
- Retain Hierarchy：该参数主要配合【Correlate Simulation Data to Input Design】参数，完成时序仿真时信号的检查，详细内容参考【Correlate Simulation Data to Input Design】参数说明。
- Generate Multiple Hierarchical Netlist Files：该参数指定是否对设计中保留层次的每一个层次产生时序仿真模型以及 SDF 时序标注文件。这个参数有效的前提有：【Retain Hierarchy】参数设置为【True】的同时在源代码中各个模块都使用【Keep Hierarchical】属性。如果这个参数设置为【True】之后，【Use Automatic Do File for ModelSim Simulation】参数将被设置为【False】，原因在于如果需要 Project Navigator 环境中运行 ModelSim 仿真器就必须指定相应的 DO 文件，产生了多个时序仿真模型之后，其仿真的 DO 文件就需要明确指定，而不能使用自动生成的 DO 文件。
- Use Automatic Do File for ModelSim Simulation：设置使用自动 DO 文件，该参数仅仅在 Project Navigator 集成环境中启动 ModelSim 时有效。
- Bring Out Global Tristate Net as a Port：该参数指定是否在输出的仿真模型的顶层添加一个端口来连接全局三态网线（全局三态网线将强制所有的输出为高阻态），全局三态信号没有驱动的情况下该参数有效。
- Global Tristate Port Name：指定全局三态端口名称，默认为 GST。
- Bring out Global Set/Reset Net as a port：这个参数指定是否在输出的仿真模型的顶层添加一个端口来连接全局的置位/复位网线，全局置位/复位网线没有驱动的情况下这个参数有效。
- Global Set/Reset Port Name：指定全局复位/置位端口名称，默认为 GSR。
- Generate Test Fixture File：指定是否从输出的网表中创建一个测试模板。

- Rename Design Instance In Test Fixture File to：指定测试模板中设计的实例化名称，默认为 UUT。
- Include 'uselib Directive in Verilog File：指定是否在输出的网表文件中使用 Verilog 'uselib 指示，总是指向系统中的 SIMPRIM 源文件处，该参数在大多数仿真器中不推荐使用。默认条件下设置为【False】。
- Path Used in \$SDF_annotate：指定 SDF 标注文件的路径，一般情况下，当网表文件与 SDF 标注文件不在一个目录下时需要使用该参数。
- Do Not Escape Signal and Instance Names in Netlist：是否在网表中对信号以及实例化名称进行转义。
- Other NETGEN Command Line Options：输入其他的 NETGEN 命令行参数，多个参数之间使用空格。

以上说明了在 Xilinx 公司的 Project Navigator 开发环境配置 ModelSim 会遇到的绝大多数参数，仿真 VHDL 使用的参数与 Verilog 类似，不再详细解释。

3.2 ModelSim 仿真窗口综述

ModelSim 5.8 版本仿真工具在默认条件下提供 11 种不同的用户窗口：主窗口、结构窗口、源程序窗口、信号窗口、进程窗口、变量窗口、数据流窗口、波形窗口、存储器窗口和列表窗口等，除此之外还提供了一个窗口叫断言窗口，如图 3-8 所示。存储器窗口和断言窗口是 ModelSim 5.8 版本最新提供的，以前的各种版本中并没有。

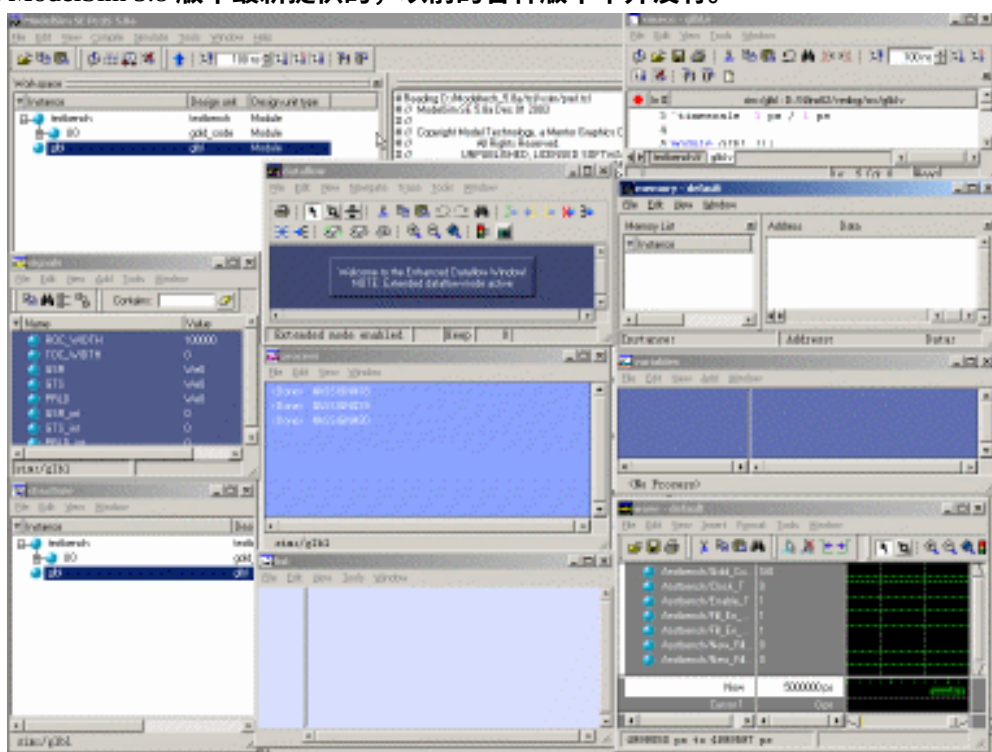


图3-8 ModelSim 仿真器窗口

ModelSim 软件的用户窗口全部是基于 Tcl/Tk 语言编写的，所以可以通过使用 Tcl/Tk 语言来添加各种不同的用户窗口，也可以通过软件自带的一些工具来定制仿真环境，工具的这种可扩展性大大提高了仿真的效率，这也是 ModelSim 工具的一大特点。在仿真过程中，除了主窗口之外的其他类型窗口都可以打开多个不同的副本，并且窗口中的对象都可以从一个窗口直接拖动到另外的窗口，使用起来和 Windows 的资源管理器类似。

1. 主窗口

主窗口在 ModelSim 启动时就直接打开，是所有其他窗口运行的基础。通常情况下主窗口分为工作区和脚本区（也叫命令控制台）两个部分，通过工作区可以很方便地对当前工程的工作库以及所有打开的数据集合等进行控制（数据集合的概念请参考 3.6.2 和 3.6.3 节），通过命令控制台可以在 ModelSim 的提示符下输入所有 ModelSim 的各种命令，并且可以将命令执行后的信息反馈回来，便于用户了解命令的执行情况，如图 3-9 所示。

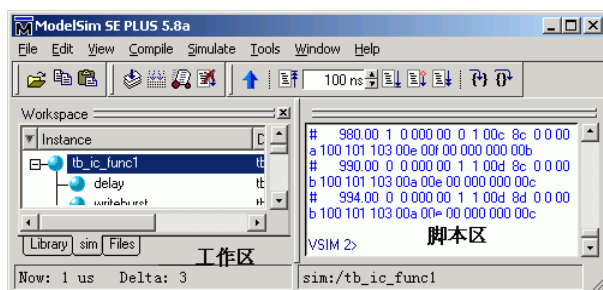


图3-9 ModelSim 主窗口

在 5.8 以后的版本中，主窗口中的工作区以及命令控制台（脚本区）都是浮动窗口，同时还可以在主窗口中使用【View】/【Active Processes】命令打开一个活动进程窗口【Active Processes】。因此 ModelSim 5.8 典型的主窗口形式可能会如图 3-10 所示。

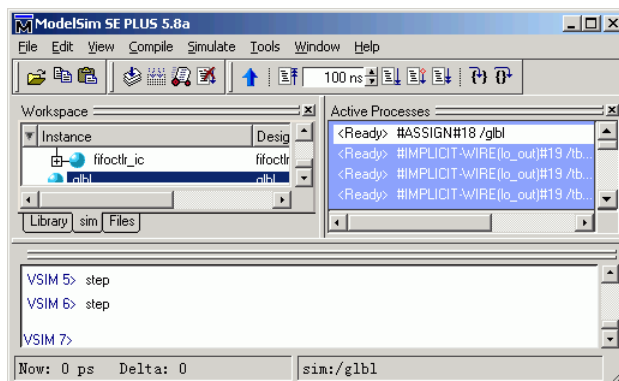


图3-10 典型的 ModelSim 主窗口

可以通过在主窗口中选择【View】/【Workspace】命令来显示或者隐藏工作区。通过【View】/【Active Processes】命令来显示或者隐藏活动进程窗口。

工作区的视图数量会根据仿真流程的不同而发生变化，其中除了库文件视图（Library）以外，其他视图都为可选项。

- 库文件视图 (Library)：在该视图中显示了当前工作库中包含的所有设计单元信息。
- 项目视图 (Project)：在该视图中显示了当前工程中包含的所有文件信息。
- 仿真视图 (sim)：在该视图中显示了当前仿真用到的各种元件的结构层次，该视图与结构窗口中显示的信息保持一致。
- 数据集合视图 (vsim)：该视图是因为打开了一个名称为 vsim 的数据集合而产生的，该视图的命名与数据集合的名称相同，其中显示了数据集合中用到的各种元件的结构层次。在 ModelSim 5.3 以后的各种版本中可以同时打开多个数据集合，每一个数据集合将产生一个数据集合视图。
- 比较视图 (Compare)：该视图只有在进行波形比较的时候才会出现，其中显示了波形比较过程中创建的各种比较对象的结构层次（详细内容请参考波形比较章节）。

命令控制台则保留了所有运行过的命令，并显示了命令运行后的反馈信息，同时还可以显示测试激励文件中各种输出信息。在仿真运行过程中，对于运行的所有命令以及命令的反馈信息可以通过脚本文件的形式保存下来，默认情况下此文件保存在工程的根目录下，文件名为“transcript”。如果希望使用不同的名称以及路径来保存这个脚本文件，可以通过选择主窗口中的【File】/【Transcript】/【Save Transcript as】命令来完成。

2. 数据流窗口

数据流窗口 (Dataflow) 是一般仿真软件都提供的一个通用窗口，通过该窗口可以跟踪设计中的物理连接，跟踪设计中事件的传播，也可以用来跟踪寄存器、网线和进程等，极大地丰富了调试方法，数据流窗口如图 3-11 所示。

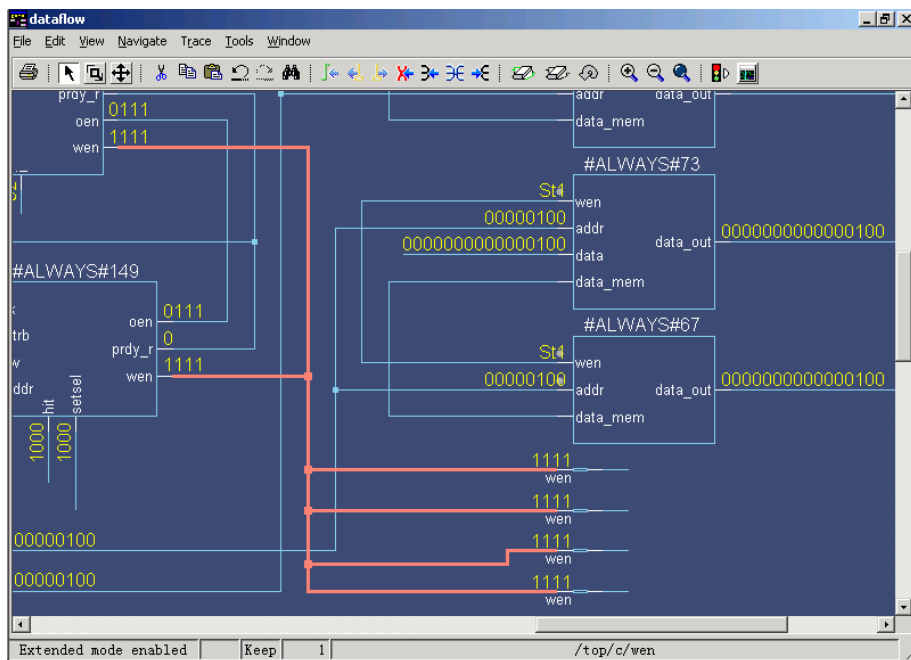


图3-11 数据流窗口

数据流窗口中可以显示进程（可以是 Verilog 的一个模块）、信号、网线和寄存器等，也可以显示设计中的内部连接。窗口中有一个内置的符号表，映射了所有的 Verilog 基本门，例如与门、非门等，这些符号可以在数据流窗口中显示。其他的 Verilog 基本组件可以使用模块或者用户定义的符号在数据流窗口中显示。

数据流窗口中的符号都使用了类似【#ASSIGN#23#2】、【#ALWAYS#56】或【<module_name>】等信息进行说明，其中第一个#说明了产生这个符号的语句，第二个#后面紧跟了产生这个符号的语句所在的源文件中的行号，第三个#说明了这个语句在源文件中属于当前行的第几个语句。

可以通过以下几种方法向数据流窗口中添加项目：

- 从其他窗口中选中相关项目并拖动。
- 使用数据流窗口中的【Navigate】菜单。
- 在主窗口命令行使用“add dataflow”命令。
- 从波形窗口中双击任何一个信号波形。

对数据流窗口中一般菜单不进行说明，比较特殊的菜单有：

(1) 【Navigate】菜单

- 【Expand net to drivers】：显示所选择信号、网线以及寄存器的驱动。
- 【Expand net to readers】：显示所选择信号、网线以及寄存器的输出。
- 【Expand net】：显示所选择信号、网线以及寄存器的驱动以及输出。
- 【Expand to design inputs】：显示网线的顶层驱动，一般在 testbench 或者顶层模块中。
- 【Expand to hierarchy input】：显示网线所在层次的一级驱动。

(2) 【Trace】菜单

- 【TraceX】：单步跟踪不确定值的上一个驱动。
- 【ChaseX】：直接跟踪不确定值的源头。
- 【Trace next event】：移动。
- 【Trace event set】：跳转到输入时间的源头。

数据流窗口最大的用途是进行追踪，方便查找引起意外输出的原因，在使用这个功能时会用到数据流窗口中内嵌的波形窗口。这个波形窗口中的活动指针与数据流窗口相关联，移动指针将影响在数据流窗口中信号值的变化。在数据流窗口中跟踪一般信号的变化可以通过双击鼠标左键来完成。

3. 列表窗口

列表窗口使用表格的形式显示仿真的结果。窗口被分为两个可调整的部分，右边为信号列表，左边为仿真运行时间以及仿真的 Delta 时间，如图 3-12 所示。可以从主窗口中创建列表窗口的第二个副本，两个列表窗口可以进行不同的设置，便于仿真结果的比较，同时也可以在波形比较时对相应的数据进行列表对比。

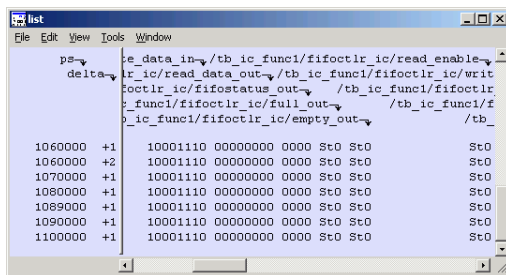


图3-12 列表窗口

可以通过以下几种方法向列表窗口中添加项目：

- 从其他窗口中选中相关项目并拖动到列表窗口。
- 从主窗口命令行使用“add list”命令。
- 可以加载从列表窗口保存的格式文件。

对列表窗口中一般的菜单项不进行说明，比较特殊的菜单项有：

(1) 【File】菜单

- 【Open Dataset】：打开一个现有的 WLF 数据集文件。
- 【Save Dataset】：保存当前的仿真数据成一个 WLF 数据集文件。
- 【Write List】：保存列表窗口数据到一个文本文件中。
- 【Save Format】：保存当前的列表窗口显示以及信号属性到一个 DO 文件。
- 【Load Format】：打开原来已经保存的 DO 文件。

(2) 【View】菜单

- 【Signal Properties】：设置标签、数据显示模式以及触发器的打开/关闭等。
- 【Goto】：直接从当前时间标签跳转到一个时间标签。

(3) 【Tools】菜单

【Window Preferences】：设置窗口中全部项目的属性。

在列表窗口中通过选择【Tools】/【Window Properties】命令可以弹出【Modify Display Properties(list)】对话框，使用这个对话框可以对列表窗口的显示属性进行设置，其中包括窗口属性设置【Window Properties】和触发设置【Triggers】两个选项卡，如图 3-13 所示。

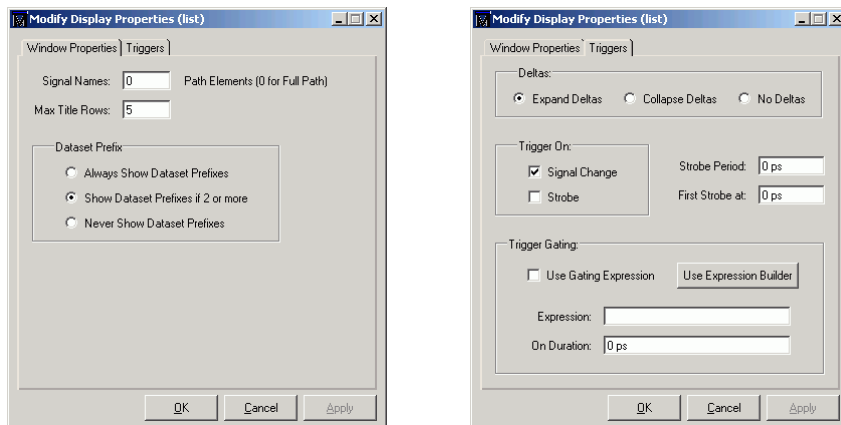


图3-13 【Modify Display Properties(list)】对话框

(1) 【Window Properties】选项卡

- 【Signal Names】选项用来设置列表窗口中信号路径的显示方法，“0”表示列表窗口中的信号将使用完整路径显示，“1”表示列表窗口中的信号仅使用信号名显示。
- 【Max Title Rower】选项用来设置每一行信号数量的最大值，如果信号数量超过这个最大值则需要另起一行进行显示。
- 【Dataset Prefix】：数据集合的前缀控制选项，在这个选项中包括以下 3 个单选项。
 - 【Always Show Dataset Prefixes】：总是显示数据集合前缀。
 - 【Show Dataset Prefixes if 2 or more】：如果数据集合多于两项，才显示数据集合前缀。
 - 【Never Show Dataset Prefixes】：不显示数据集合前缀。

(2) 【Triggers】选项卡

该选项卡用来控制在列表窗口中显示新的数据行所需要的触发条件。

- 【Deltas】：其中有以下 3 个单选项。
 - 【Expand Deltas】：展开 Deltas 显示，当在触发条件中选择了【Signal Change】选项时，在信号变化的每一个仿真 Delta 时间都显示一个新的数据行。
 - 【Collapse Deltas】：不显示内容 Deltas 时间变化影响，只显示最后一个时间单元的数值。
 - 【No Deltas】：隐藏仿真的 Delta 时间
- 【Trigger on】：包含以下两个选项。
 - 【Signal Change】：使用信号的变化作为触发条件，默认条件下，包括所有信号。
 - 【Strobe】：使用指定的一个周期作为触发条件。
- 【Trigger Gating】：使用门级表达式作为触发条件。

4. 存储器窗口

存储器窗口是 5.8 版本最新提供的一个用户界面，通过这个窗口可以显示设计中存储器的内容，左边为存储器列表，右边为对应的存储器的内容，如图 3-14 所示。

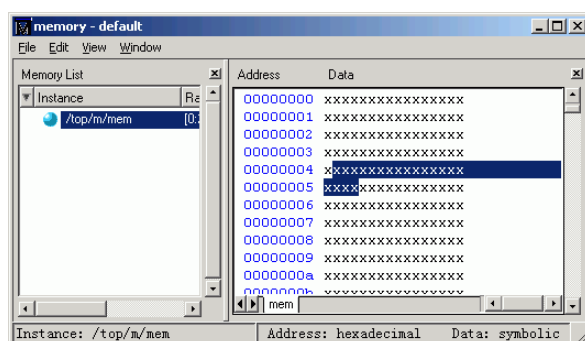


图3-14 存储器窗口

存储器窗口的菜单都比较容易理解，这里不做说明。

使用存储器窗口可以对设计中的存储器进行初始化，即可以通过加载原有文件的方法也可以通过改变存储器中的内容的方式。同时存储器窗口中当前的数据可以保存成一个存储器文件，便于以后使用。

5. 进程窗口

ModelSim 5.7 以及以后的各种版本中，进程窗口可以作为主窗口的一个可选窗口进行显示，为了在主窗口中显示进程窗口可以使用【View】/【Active Processes】命令。

进程窗口显示了仿真中用到的所有进程的列表，如果在该窗口中选择【View】/【Active】命令，那么在窗口中将显示当前工程中的全部进程，如果选择【View】/【In Region】命令，那么在窗口中只显示当前层次中包含的进程（这里的层次通过结构窗口来指定），如图 3-15 所示。

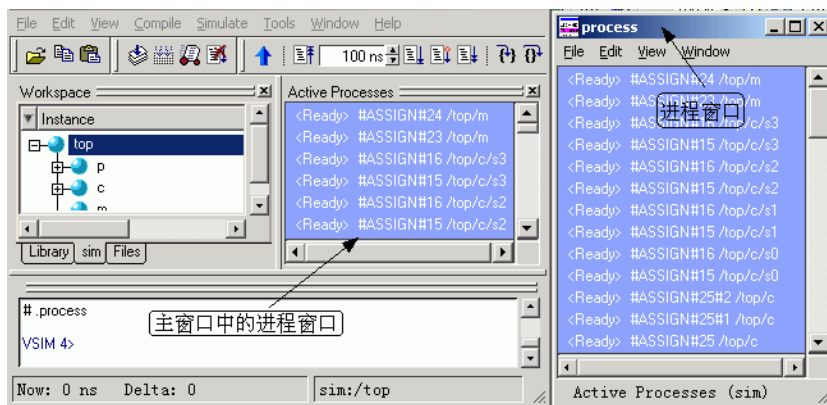


图3-15 进程窗口

每一个进程都有【Ready】、【Wait】和【Done】3个不同的状态。

- 【Ready】表示这个进程将在当前的 delta 时间内被执行。
- 【Wait】表示进程正在等待 VHDL 信号或者 Verilog 的网线和信号的变化。
- 【Done】表示进程正在执行 VHDL 的等待语句。

如果在进程窗口中选择一个进程，在以下各种窗口中进程都会被更新：在数据流窗口中选中的进程会被高亮显示，在存储器窗口中将会显示选定的进程中包含的存储器单元，信号窗口中显示选定的进程区域内的信号，源代码窗口中显示相关的源代码，结构窗口中显示进程相关的区域，变量窗口中显示进程区域的变量。

6. 信号窗口

信号窗口主要用来选择需要查看的信号，在结构窗口中选择不同的层次，信号窗口中的信号将根据设计层次的变化而发生变化。如果在信号窗口中双击某一个信号，那么源文件中将会对相应的信号高亮显示。

信号窗口的另外一个主要作用就是在仿真过程中可以强制使某一个信号的值发生变化，也可以将任意信号强制成时钟信号。这一功能的实现需要通过在信号窗口中选择【Edit】/【Force】和【Edit】/【Clock】命令来执行。

7. 源文件窗口

源文件窗口主要用来显示和编辑 HDL 源文件代码。当第一次加载一个设计时，设计的源代码自动在源文件窗口中打开。

ModelSim 源文件窗口是一个很优秀的硬件描述语言编辑工具，在这个窗口中可以显示文件的行号，同时可以使用  按钮打开语言模板来方便源代码的编写，语言模板会根据编写的源文件的类型自动调整，如图 3-16 所示。

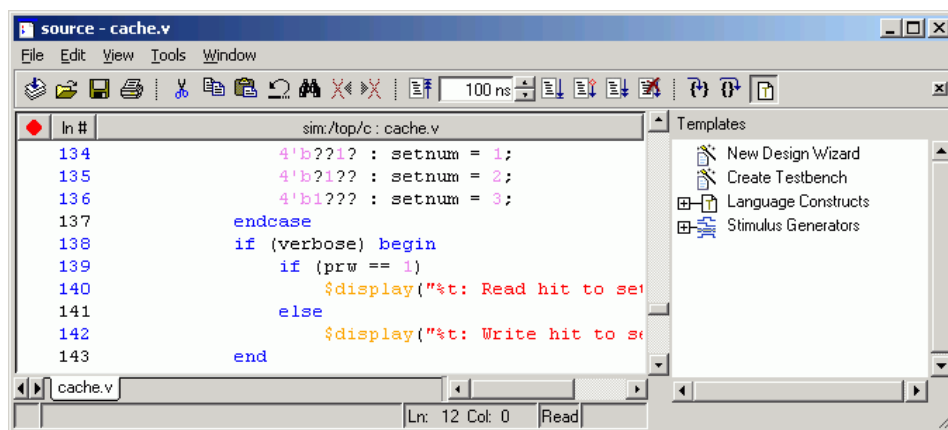


图3-16 源文件窗口

在源文件窗口中蓝色的行号表示此行可以用来设置断点用于调试，蓝色箭头指示当前选中的进程，红色菱形指示在本行设置了断点，而空心红色菱形指示当前断点取消。文件标签表示了当前打开的文件名称，默认条件下，打开的文件为只读文件，可以使用【Edit】/【read only】命令取消或者恢复只读属性。

8. 结构窗口

在 ModelSim 5.5 以后的版本中，主窗口的工作区中包含了一个仿真视图，这个视图中显示的信息和结构窗口中显示的信息相同，用来显示设计的结构层次，可以在主窗口中使用【View】/【Structure】命令打开结构窗口。

在结构窗口中查看当前设计的层次，如果在结构窗口中选择某一个设计层次后，进程窗口、变量窗口等都会立即更新。

在结构窗口中 VHDL 的模块使用深蓝色正方形表示，Verilog 模块使用浅蓝色圆形表示，SystemC 模块使用浅绿色菱形表示，虚拟对象例如用户定义的总线等使用黄色菱形表示。

9. 变量窗口

变量窗口被分为两个不同的部分，左边为当前进程中用到的所有变量，右边为与变量相关的当前值，如图 3-17 所示。在变量窗口中可以查看 VHDL 中的常数、类属、变量以及 Verilog 中的寄存器和变量等。

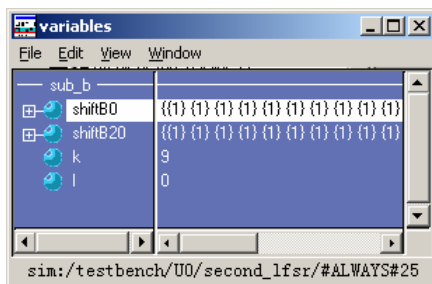


图3-17 变量窗口

10. 波形窗口

就像列表窗口可以用来查看仿真结果一样，波形窗口也可以用来显示仿真波形，并且比列表窗口更直观，所以波形窗口是最常用的仿真窗口之一。

波形窗口一般分为三个不同的区域，分别用来显示信号名称以及路径、光标所在位置信号的当前值、波形等，如图 3-18 所示。

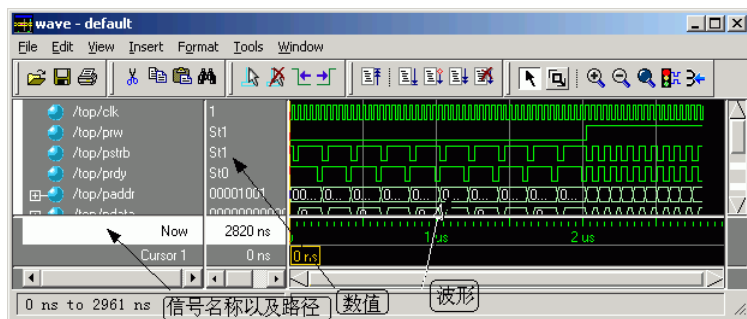


图3-18 波形窗口

波形窗口可以使用【Insert】/【Divider】命令分为多个部分，同时可以使用【Insert】/【Window Pane】命令分为多个窗口，所以信号可以在多个窗口和区域之间进行拷贝、复制、移动等。

可以通过以下方式向波形窗口中添加项目。

- 从其他窗口中选中相关项目并拖动到波形窗口。
- 在其他窗口选中相关的项目区域，在命令行使用“add wave *”命令。
- 可以使用【File】/【Open】/【Format】命令加载原有的波形窗口数据。

波形窗口中一般的菜单项不进行说明，比较特殊的菜单介绍如下。

(1) 【File】菜单

• 【Open】命令

【Dataset】：打开一个数据集文件，参考 WLF 文件一节。

【Format】：运行一个原先存储好的波形窗口格式文件（DO 文件）。

• 【Save】命令

【Dataset】：存储当前的仿真为一个 WLF 文件，参考 WLF 文件一节。

【Format】：存储当前的波形窗口中的显示属性以及信号属性等变量成一个 DO 文件，运行这个 DO 文件将重现当前的配置。

【Image】：将当前文件存储成一个位图文件。

(2) 【Tools】 菜单

- 【Waveform Compare】：波形比较
- 【Breakpoints】：设置断点。
- 【Dataset Snapshot】：周期地存储当前的仿真为一个 WLF 数据文件。

可以使用  按钮打开数据流窗口并在其中显示波形窗口中选定信号的驱动，在波形窗口移动光标时，数据流窗口中相应的信号值会发生变化，这样方便对仿真中的问题进行定位。

在波形窗口中选定一个信号之后，单击鼠标右键会出现一个浮动的下拉式菜单，在这个菜单中有一个比较特殊的命令【Format】，这个命令有以下 4 个子选项。

- 【Literal】：在波形窗口中显示当前的值，在列表窗口中只可以使用这种格式对数据进行存储。
- 【Logic】：在波形窗口中显示逻辑值：U/X/0/1/Z/W/L/H/-等。
- 【Event】：在波形窗口中将每一个信号翻转作为一个事件标记。
- 【Analog】：在波形窗口中的数据模拟化，有以下 3 种显示方式。
【Step】：使用等距节点模拟显示。

【Interpolated】：使用内插模拟显示。

【Backstep】：使用 Backstep 模拟显示，经常用于能量计算。

如果使用数据模拟化显示需要设置【Offset】以及【Scale】，【Offset】以当前信号位置为 0 点加上显示器像素个数，这个值可以为正或者为负数。【Scale】用来调节显示比例，如图 3-19 所示为【pdata】信号经过模拟化之后的显示。

波形窗口中可以使用光标对信号的时间区间进行测量，也可以锁定某些特殊光标，锁定的方法是，使用鼠标选中需要锁定的光标，然后选择【Edit】/【Edit Cursor】命令打开光标编译对话框，选中其中【Lock cursor to specified time】复选框即可。锁定后的光标为红色，如图 3-19 所示。

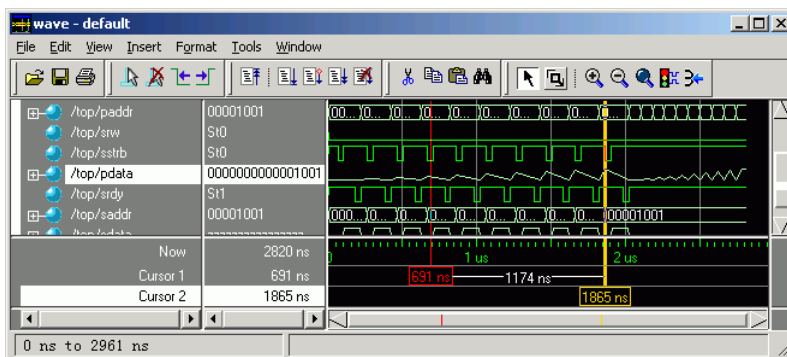


图3-19 波形窗口中信号模拟化以及光标锁定

11. 断言窗口

在 ModelSim 5.8 以后的版本中还新增加了断言窗口，如果对其中含有断言的程序进行仿真就会启动断言窗口。

基于断言的验证方法已经讨论了很久，随着 PSL (Property Specification Language Reference Manual) 标准的出现，设计者对于基于断言进行仿真的兴趣也与日俱增。

断言允许设计人员将一些判断语句嵌入到设计中，可以对某些设计的意向行为做成的假设进行说明。传统的硬件调试是假定了错误能够在整个设计当中传播并可以在输出端口捕获到，为了确定错误出现的位置，设计者不得不花费大量的时间来查找问题。而使用断言则能够在错误发生时及时捕获，允许设计人员迅速明确地对错误进行定位，大大简化了纠错的工作。

3.3 仿真环境的建立

ModelSim 有很多不同的版本，如 ModelSim/Verilog、ModelSim/VHDL，以及各个公司的 OEM 版本等。Xilinx 公司使用的 ModelSim 为 ModelSim XE 的 OEM 版，这个版本只支持 Xilinx 公司的所有器件的时序仿真，但是不支持其他公司的 FPGA/CPLD 的仿真。其他公司 OEM 版本的 ModelSim 也存在类似的问题，一般的 FPGA/CPLD 设计者都会同时使用各种不同厂家的 FPGA/CPLD 器件，所以通常情况下都会选择使用支持全部器件的 ModelSim 完全版。在完全版本发行时，不发行任何公司的 FPGA/CPLD 的仿真库，所以在使用之前需要建立相应 FPGA/CPLD 器件的仿真库，这里所讲到的关于仿真库的所有问题都是针对于 ModelSim SE/PE 版本，如果读者使用的是 ModelSim XE 版本就不需要了解这么多，直接使用就可以。

在讲述仿真的切入点的时候提到，RTL 行为级仿真时，如果没有实例化一些与器件相关的特殊底层元件的话，这个阶段的仿真也可以做到与器件无关，但是很多时候不得不使用一些特殊的底层元件，这时就需要一些器件库模型的支持。而综合后门级功能仿真以及实现后时序仿真都需要厂家器件库的支持。

3.3.1 各仿真切入点需要的库文件

Xilinx 公司为仿真设计了 5 个切入点，表 3-1 列出了在每一个仿真的切入点所需要的仿真库模型。

表 3-1 各个仿真阶段需要的仿真库模型

仿真		UNISIM	XilinxCoreLib	SmartModel	SIMPRIM	SDF
1	RTL 行为	X	X	X		
2	综合后功能	X	X	X		
3	NGDBuild 后功能仿真			X	X	
4	映射后时序			X	X	X
5	布局布线后时序			X	X	X

RTL 行为级仿真可能需要的文件包括：RTL 代码、实例化的 UNISIM 库元件、XilinxCoreLib 和 UNISIM 门级模型、SmartModels。

综合后功能仿真可能需要的文件包括：综合工具输出的包含了 UNISIM 库元件的门级

网表、XilinxCoreLib 和 UNISIM 门级模型、SmartModels 模型等。

NGDBuild 后功能仿真可能需要的文件包括：包含了 SIMPRIM 库元件的门级网表和 SmartModels。

映射后时序仿真可能需要的文件包括：包含了 SIMPRIM 库元件的门级网表、SmartModels、标准延时格式文件 SDF。这个阶段的设计没有进行布局布线，仿真也类似于 IC 设计过程中的门级仿真，提供了各种门的延时，但是没有连线延时，可以认为是不完全的时序仿真。

布局布线后时序仿真可能需要的文件包括：包含了 SIMPRIM 库元件的门级网表、SmartModels、标准延时格式文件 SDF。

3.3.2 仿真库文件说明

UNISIM (Library of Unified component simulation models) 库仅仅用来做功能仿真，在这个库中包括了 Xilinx 公司全部的标准元件，可以被绝大多数的综合工具推论。UNISIM 库被分为 VHDL 和 Verilog 两种。

VHDL UNISIM 库被分成 4 个不同的文件，包括元件声明 (unisim_VCOMP.vhd)、程序包文件 (unisim_VPKG.vhd)、实体和结构体声明 (unisim_VITAL.vhd) 和 SmartModel 声明 (unisim_SMODEL.vhd)。在这个文件中包括了 Xilinx 全部器件族的标准元件。VHDL UNISIM 库源文件位于 \$XILINX/vhdl/src/unisims 目录下。

Verilog UNISIM 库文件中每一个元件使用一个独立的文件，这样做是为了方便一些特殊的编译向导指令如 'uselib 等。根据器件族的不同，Verilog UNISIM 库分为两个目录，对于 FPGA 器件族如 Spartan-II™、Spartan-II E™、Spartan-3™、Virtex™、Virtex™-E、Virtex-II™、Virtex-II Pro™ 和 Virtex-II Pro X™ 等 UNISIM 库源文件位于 \$XILINX/verilog/src/unisims 目录下，而对于 CPLD 器件族如 XC9500XL™、XC9500XV™、CoolRunner-XPLA3™、CoolRunner-II™ 和 CoolRunner-IIS™ 等 UNISIM 库源文件位于 \$XILINX/verilog/src/uni9000 目录下。

XilinxCoreLib (CORE Generator™ HDL library models) 库仅仅用来做功能仿真，在这个库中包括了使用 Xilinx Core Generator 工具产生的 IP 的仿真模型，例如 FIR 滤波器、各种 FIFO，大的 RAM、CAM 等。在 Xilinx Core Generator 工具中可以定制很多有用的 IP，而这些 IP 都已经做了很好的优化处理，非常适合于当前的器件结构，在这个工具中产生的模块不能被综合工具推论，而是当做黑盒进行处理。对于 VHDL 代码来说，这个库位于 \$XILINX/vhdl/src/XilinxCoreLib 目录。对于 Verilog 代码来说，这个库位于 \$XILINX/verilog/src/XilinxCoreLib 目录。

SIMPRIM (Library of generic simulation primitives) 库用来做时序仿真或者门级功能仿真。对于 VHDL 代码来说，这个库位于 \$XILINX/vhdl/src/simprims 目录。对于 Verilog 代码来说，这个库位于 \$XILINX/verilog/src/simprims 目录。

SmartModel 库被用来模拟非常复杂的一些 FPGA 设计，其中用到了 PowerPC (PPC) 或者 RocketIO 等。这个库的源代码是加密的，它通过 SWIFT 接口与仿真器通信，库文件位于 \$XILINX/smartmodel 目录。

3.3.3 库文件编译

不同阶段的仿真都使用了不同的库文件，所以在开始仿真之前就必须将需要的库编译好，这样可以提高仿真效率。

3.3.3.1 在仿真器中编译仿真库

可以用很多种方法对库文件进行编译，同时各个器件厂家为了方便用户也提供了多种不同的编译方法，其中最常用的是在仿真器中对厂家的库文件进行编译，这种方法的适用范围很广。

本节将使用 Verilog 语言中时序仿真库 `simprims` 为例来说明仿真库的手动建立过程，其他仿真库的建立方法与此类似。整个手动建立过程分为以下 4 个步骤。

🔑 仿真器中仿真库的编译

1. 将 ModelSim 根目录下的配置文件 `modelsim.ini` 的属性由只读改为可写，这个操作是为了使软件可以记录仿真库建立的路径以及映射关系。以后每次启动 ModelSim 仿真工具时，软件都会根据 `modelsim.ini` 文件中的配置寻找仿真库，并且形成映射关系。
2. 启动 ModelSim 仿真工具，在主窗口中选择【File】/【Change Directory】命令，将工作目录改变到想要存放仿真库的目录下。
3. 在主窗口中选择【File】/【New】/【Library】命令，弹出【Create a New Library】窗口，如图 3-20 所示。将【Create】选项设置为【a new library and a logical mapping to it】，在【Library Name】栏中输入“`simprims_ver`”，同时软件在【Library Maps to】栏中自动输入“`simprims_ver`”，单击 **OK** 按钮保存。这一个操作过程实质上相当于在 ModelSim 主窗口的脚本区域中输入了 `vlib simprims_ver` 和 `vmap simprims_ver simprims_ver` 两个命令。

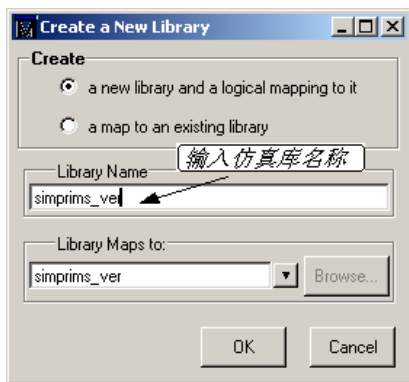


图3-20 创建一个新的仿真库

4. 在主窗口中选择【Compile】/【Compile】命令，弹出【Compile Source Files】窗口，如图 3-21 所示。在【Library】选项的下拉列表中选择“`simprims_ver`”仿真库，在【查找范围】选项中选中【`Xilinx\verilog\src\simprims`】目录下的

全部文件，单击 **Compile** 按钮进行编译。

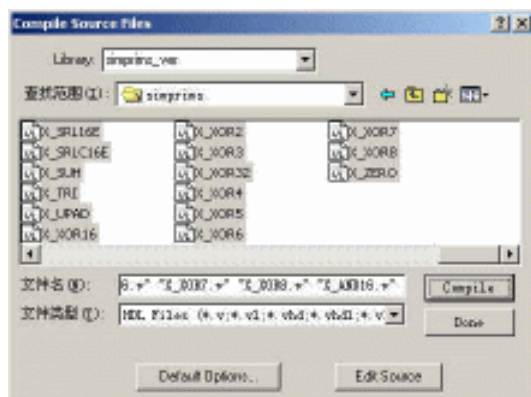


图3-21 编译源文件对话框

提示：因为操作系统的原因，如果源文件数量太多，请分多次进行编译。

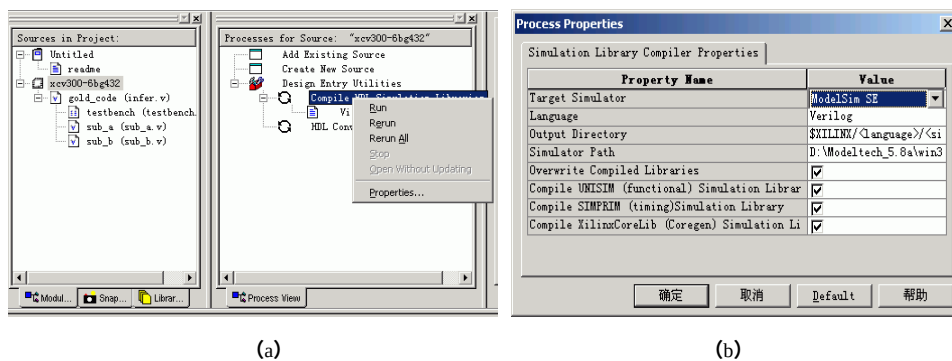
可以使用类似的方法对其他的仿真库进行编译，而 VHDL 语言仿真库的编译也可以同样进行。

3.3.3.2 在 Project Navigator 中编译仿真库

在 Project Navigator 中可以编译需要的仿真库，在运行仿真时当前工程目录下会自动产生一个 modelsim.ini 配置文件，文件中添加了编译后的仿真库。

🔑 Project Navigator 中仿真库的编译

1. 使用 Project Navigator 创建一个新的工程或者打开一个现有的工程。
2. 选择设计的目标器件，在【Source in Project】窗口中选中器件，如图 3-22 (a) 所示，【Processes for Source】窗口中会出现【Design Entry Utilities】菜单，使用右键选中其中的【Compile HDL simulation Libraries】项，在下拉菜单中选中【Properties】命令，打开属性对话框，如图 3-22 (b) 所示。



(a)

(b)

图3-22 Project Navigator 中的对话框

3. 【Process Properties】对话框中的各项很明了，根据需要选择相应的仿真库文件，然后单击 **确定** 关闭对话框。

4. 双击【Compile HDL simulation Libraries】，程序将自动对完成对库文件的编译。
5. 库文件编译完成以后，在 Project Navigator 集成环境中运行 ModelSim 仿真器进行仿真时，软件将自动在当前工程目录下生成一个 modelsim.ini 配置文件，在这个配置文件中添加了编译后的各种仿真库路径。

提示：如果已经使用以上方法编辑了仿真库，那么再新建其他项目后，并不需要重新编译仿真库文件，而仅仅需要修改仿真器工作目录下的 modelsim.ini 配置文件即可。

3.3.3.3 使用命令行编译仿真库

上一小节，在 Project Navigator 集成环境中编译仿真库时调用了 COMPNLIB 命令，这个命令可以直接在控制台的命令行运行。可以通过在命令行键入 compxlib -help 命令查看帮助。

COMPNLIB 命令可以支持 Xilinx 的所有器件族，支持编译如下的仿真库文件：

- UNISIM (Functional)
- Uni9000 (Timing)
- SIMPRIM (Timing)
- XilinxCoreLib (Functional)
- SmartModel Library™ (Functional & Timing)
- CoolRunner™ (Functional)
- Abel (Functional)

可以支持如下各种仿真器：

- ModelSim™ SE (all Xilinx® supported platforms)
- ModelSim™ PE (all Xilinx® supported platforms)
- NCSIM™ (all Xilinx® supported platforms)
- VCS™ (only on Solaris® and Linux based platforms)
- VCSi™ (only on Solaris® and Linux based platforms)
- Scirocco™ (only on Solaris® and Linux based platforms)

例如：使用 `compxlib -s mti_se -f all:u:s -l verilog -o $MYAREA` 命令可以编译针对于仿真器 ModelSim_SE 版本的 Verilog 语言的 UNISIM、UNI9000 以及 SIMPRIM 库文件，并且将编译结果保存在 \$MYAREA 目录下，其中所有的路径名称都可以使用绝对路径。

COMPNLIB 程序常用的命令行参数分别为：

- -help (Print COMPNLIB help)，用来打印 COMPNLIB 的帮助文件。
- -s (Target Simulator)，用来指定目标仿真器，如果这个参数没有指定程序将退出。-s 的有效值如下：
 - s mti_se
 - s mti_pe
 - s ncsim
 - s vcs
 - s vcsi

- s scirocco
- -l (Language) ，用来指定编译何种语言的仿真库。默认条件下，程序可以从目标仿真器中检查仿真器所支持的语言类型，如果仿真器支持 VHDL 以及 Verilog 程序，会将-l 参数设置为 all，否则设置为仿真器支持的类型。如果对这个参数进行了特殊的指定将按照指定的类型编译。这个参数的有效值如下：
 - l verilog
 - l vhdl
 - l all
- -f (Device Family) ，用来指定目标器件族。如果这个参数没有指定程序将退出。这个参数的有效值如下：
 - f all
 - f virtex
 - f virtexe
 - f virtex2
 - f virtex2p
 - f spartan2
 - f spartan2e
 - f cpld
 - f xpla3
 - f xbr
 - f xc9500
 - f xc9500xl
 - f xc9500xv

同时-f 参数后还可以指定对应器件的仿真库类型，例如 compxlib -s vcs -f virtex2p:u -l verilog，这行命令表示只为 vcs 仿真器编译 Virtex-II Pro 器件族的 UNISIM 仿真库。仿真库类型的有效值为以下第一列，代表的仿真库为以下第二列。

u	UNISIM
s	SIMPRIM
n	UNI9000
c	XilinxCoreLib
m	SmartModel
r	CoolRunner
a	Abel

- -o (Output Directory) ，用来指定编译后文件的存放位置，默认条件下文件将存放在\$XILINX/language/target_simulator 目录下。
- -p (Simulator Path) ，用来指定仿真器执行文件所在的目录，默认条件下程序将在\$PATH 环境变量指定的范围内自动搜索，如果没有指定\$PATH 变量，那就必须指定-p 参数。

- -w (Overwrite Compiled Library) , 用来指定是否覆盖以前编译保存在当前目录下的仿真库文件, 默认条件下程序不覆盖以前的仿真库。

3.3.4 仿真库的命名

在 ModelSim 中编译器件的仿真库时, 使用的仿真库名称可以随意定义, 只要满足操作系统的命名规则就可以。但是如果希望从 ISE Project Navigator 集成开发环境中直接启动 ModelSim 仿真器时, 对仿真库命名就需要特殊指定, 否则 ModelSim 仿真器无法对仿真库进行映射, 这是因为从 ISE 集成开发环境直接调用 ModelSim 进行仿真时, 在工程目录下就会产生一些对 ModelSim 仿真器有控制作用的文件, 这些文件是由 ISE 在执行过程中通过 creatfdo.pl 文件生成的, 分别使用 fdo、tdo 和 udo 等作为扩展名, 其中 udo 文件为用户宏文件, 用户可以对其进行编辑并添加一些对 ModelSim 的控制命令, 而 fdo、tdo 文件一般是不建议进行编辑的。对 Verilog 进行仿真时, 在这些文件中包含了 vsim -t 1ps -L Xilinxcorelib_ver -L unisims_ver -L simprims_ver -lib work \$desunit glbl 命令, 对 VHDL 项目进行仿真时也有类似的命令。其中 vsim 是 ModelSim 运行仿真的命令, -L 表示仿真时需要的库文件。由以上可以看出, 在对使用 Verilog 语言编写的项目进行仿真时, 默认条件下需要 Xilinxcorelib_ver、unisims_ver 和 simprims_ver 三个库文件, 其中, Xilinxcorelib_ver 库对应 Xilinx 提供的软核的功能仿真库, unisims_ver 库对应使用 ECS 所做原理图的功能仿真库, simprims_ver 库对应项目布局布线以后的时序仿真库。由此可见, 在对仿真库进行命名时需要考虑这些规则的影响。

综合以上内容, 在 ISE 集成开发环境中直接启动 ModelSim 进行仿真时, Verilog 仿真库的命名使用 Xilinxcorelib_ver、unisims_ver 和 simprims_ver 三种名称, 分别对应 Xilinx/verilog/src/目录下的 Xilinxcorelib、unisims 和 simprims 三个库。VHDL 仿真库的命名使用 Xilinxcore、unisims 和 simprim 三种名称分别对应 Xilinx/vhdl/src/目录下的 Xilinxcorelib、unisims 和 simprims 三个库。

有了以上的各种关于仿真库文件的知识后, 就可以通过类似以下的命令给 ModelSim 仿真器编译 Xilinx 公司所有的仿真库。输出的仿真库默认存储在 Xilinx 的目录下。

```
compxlib -s mti_se -l all -f all -p D:\Modeltech_5.8\win32
```

库文件编译好之后, 为了方便以后使用可以修改 modelsim.ini 配置文件。打开配置文件之后, 可添加对 Xilinx 仿真库的映射。经过修改便可以直接从 ISE 环境下启动 ModelSim 仿真器进行各个阶段的仿真而不用担心报告错误。

```
[Library]
std = $MODEL_TECH/../../std
ieee = $MODEL_TECH/../../ieee
verilog = $MODEL_TECH/../../verilog
vital2000 = $MODEL_TECH/../../vital2000
std_developerskit = $MODEL_TECH/../../std_developerskit
synopsys = $MODEL_TECH/../../synopsys
modelsim_lib = $MODEL_TECH/../../modelsim_lib
```

以上为没有修改的 modelsim.ini 文件，以下为添加的内容。

```

abel_ver      = D:/Xilinx/Verilog/mti_se/abel_ver
cpld_ver      = D:/Xilinx/Verilog/mti_se/cpld_ver
unisims_ver   = D:/Xilinx/Verilog/mti_se/unisims_ver
simprims_ver  = D:/Xilinx/Verilog/mti_se/simprims_ver
XilinxCoreLib_ver = D:/Xilinx/Verilog/mti_se/XilinxCoreLib_ver
abel         = D:/Xilinx/vhdl/mti_se/abel
cpld         = D:/Xilinx/vhdl/mti_se/cpld
simprim      = D:/Xilinx/vhdl/mti_se/simprim
unisim       = D:/Xilinx/vhdl/mti_se/unisim
XilinxCoreLib = D:/Xilinx/vhdl/mti_se/XilinxCoreLib

```

其中可以使用绝对路径或相对路径。

3.4 一个简单的仿真示例

本节通过一个简单示例说明 ModelSim 仿真的一般流程。在前面提到 ModelSim 本身是一个独立的仿真工具，它并不需要依赖于其他的软件运行，所以本节将说明在 ModelSim 独立环境下进行仿真的各种仿真流程，同时说明在 ISE 集成开发环境启动 ModelSim 进行仿真的方法。

3.4.1 在 ModelSim 环境下进行仿真

如果希望在独立 ModelSim 环境下进行仿真，一般的仿真流程有基本仿真流程和工程仿真流程两种，如图 3-23 所示。

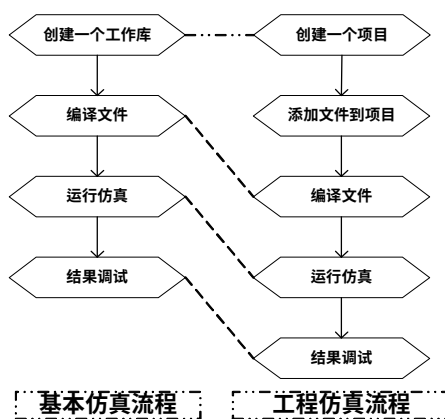


图3-23 仿真流程

1. 基本仿真流程

- (1) 创建工作库。使用 ModelSim 进行仿真时，全部的设计文件不管是 Verilog、

VHDL 或者 Verilog 与 VHDL 的混合文件都需要编译到一个库中。默认条件下，需要创建一个名称为“work”的工作库，所有文件都编译到这个库当中。

- (2) 编译。创建完一个库之后，就可以在库中编译设计单元。在 ModelSim 所支持的所有平台中库文件格式是兼容的，如果不需要重新编译，那么可以在任何平台下进行仿真。
- (3) 运行仿真。编译完成之后可以从设计顶层模块或者实体运行仿真器，一旦设计在仿真器中加载成功，仿真时间就会停留在 0 时刻，然后就可以在仿真器中运行仿真命令。
- (4) 结果调试。如果仿真结果与最初预期不相符，就需要对程序进行调试。

ModelSim 提供的调试手段与工具有以下几种：

- 在源文件窗口中设置断点或者单步执行。
- 观察波形测量时间。
- 在数据流窗口中浏览设计的物理连接。
- 查看或者初始化存储器。
- 分析仿真效率。
- 测试代码覆盖率。
- 波形比较。

2. 工程仿真流程

一个工程是在某种规范下一组 HDL 代码的一个集合。使用 ModelSim 的工程仿真流程，可以很方便地对工具进行各种设置，便于交互以及调试，因此工程流程是使用最多的一种仿真流程。

通过图 3-23 可以看出，工程仿真流程与基本仿真流程类似，主要有两个比较大的区别：

- 在工程仿真流程中不需要创建一个工作目录“work”，软件将会自动完成。
- 每次启动仿真器时都会打开上次进行的工程，除非工程已经关闭，因为工程总是持续的。

除了以上两种常用的仿真流程之外，还有一种比较特殊的流程，在这个流程中将会涉及到一个新概念——资源库。ModelSim 有两种使用库的方式，一种是本地工作库，另一种是资源库。当设计发生变化或者重新编译之后本地工作库都会进行更新，而资源库是静态的，可以在设计的多个地方使用，代码的重新编译也不会引起资源库的更新。设计人员可以创建自己的资源库提供给其他设计团队或者其他工具。

如果在设计中使用了资源库，那么设计流程将如图 3-24 所示。

不管使用何种仿真流程，在仿真时还需要提供设计源文件和测试激励文件，很多情况下可能会有多个设计源文件和多个测试激励文件，这样可以将这些文件全部编译，在加载仿真时选择所需要的顶层文件即可。

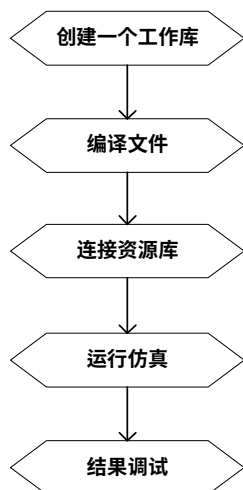


图3-24 使用了资源库之后的仿真流程

3.4.1.1 基本仿真流程

本节使用一个例子说明如何使用 ModelSim 的基本仿真流程。

🔑 基本仿真流程，参考示例详见本书附带光盘的“Example-3-1”目录

1. 将随书所附光盘“Example-3-1”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-1”目录下。
2. 启动 ModelSim。使用 Windows 系统中的 ModelSim 快捷图标或者在命令控制台使用“vsim”命令都可以启动仿真器。
3. 在 ModelSim 主窗口中选择【File】/【Change Directory】命令，改变当前目录到“Book/Chapter3/Examples3-1”。
4. 创建工作库。在主窗口选择【File】/【New】/【Library】命令，打开创建库对话框，如果软件没有自动填写库的程序则输出“work”，如图 3-25（a）所示。单击 **OK** 按钮完成库的创建以及映射。这个操作过程实质上相当于在 ModelSim 的主窗口命令控制台输入了 `vlib work` 和 `vmap work work` 命令。ModelSim 将在当前的目录下创建一个名称为 work 的库并且在其中产生一个 `_info` 文件，在整个使用过程中不要对这个文件进行编辑。
5. 编译源文件。在主窗口中选择【Compile】/【Compile】命令，打开源文件编译窗口，如图 3-25（b）所示。选中文件“counter.v”和“tcounter.v”，单击 **Compile** 按钮对选中的文件进行编译，如果编译过程没有错误则选择 **Done** 按钮关闭源文件编译对话框。在主窗口的工作区点击【work】库左边的“+”号，可以看到编译后的两个设计单元分别是“counter”和“test_counter”。

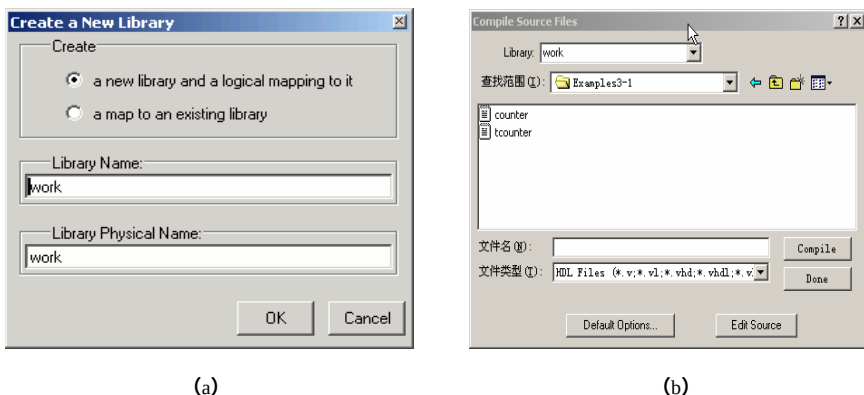


图3-25 创建库和编译源文件对话框

6. 双击设计单元“test_counter”就可以加载设计仿真。主窗口的工作区会产生一个【sim】标签和一个【Files】标签。在【sim】标签中显示了设计的层次，可以使用“+”或者“-”展开和合并设计的层次。在【Files】标签中显示了设计中包含的文件名称、类型以及存放路径等信息，如图 3-26 所示。

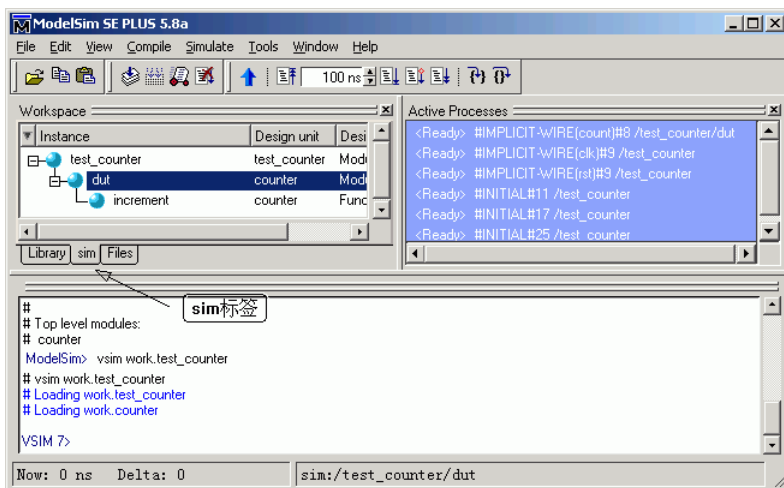


图3-26 【sim】和【Files】标签

7. 运行仿真。从主菜单中选择【View】/【All Windows】命令，打开所有 ModelSim 的窗口，根据设计的不同可使用的全部窗口有所不同。在信号窗口中使用【Add】/【Wave】/【Signal in Region】命令，添加所在层次所有信号到波形窗口。然后在主窗口、波形窗口或者源文件窗口中单击 按钮，仿真运行 100ns，100ns 是默认的仿真长度，波形窗口中将显示仿真的波形如图 3-27 所示。在主窗口“VSIM”提示符下键入 run 500 命令，仿真将会再继续运行 500ns，总共运行了 600ns。
8. 在主窗口、波形窗口或者源文件窗口中单击 按钮，仿真将连续执行直到执行了一个中断或者仿真遇到一个 \$stop 命令。要执行中断可使用 按钮。

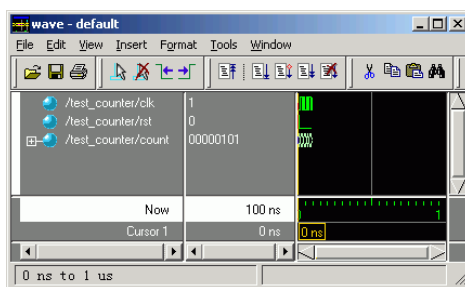


图3-27 仿真运行后的波形窗口

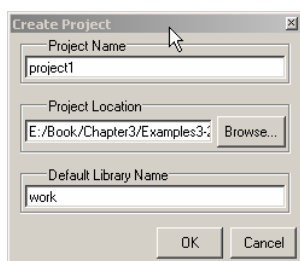
3.4.1.2 工程仿真流程

下面通过实例说明如何使用 ModelSim 的工程仿真流程。一个工程包括一个工作库和一个后缀为 mpf 的工程文件，工程文件中包括了工程参数以及工程状态等信息。一个工程中可能会包括以下内容：

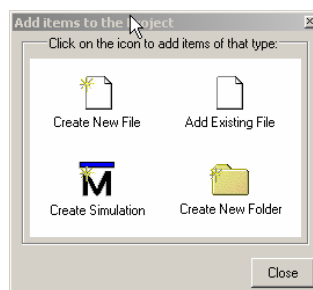
- HDL 源代码或者相关的源文件。
- 工程文档等其他文件。
- 本地库。
- 涉及到的全局库。

🔑 工程仿真流程，参考示例详见本书附带光盘的“Example-3-1”目录

1. 将随书所附光盘“Example-3-1”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-2”目录下。
2. 启动 ModelSim。
3. 创建一个工程。在主窗口中选择【File】/【New】/【Project】命令，打开创建工程的对话框，如图 3-28（a）所示。在工程名称域（Project Name）输入 project1，单击 Browse... 按钮，选择工程存放位置“Book/Chapter3/Examples3-2”，默认库文件名称域（Default Library Name）设置为 work，单击 OK 按钮保存设置。



(a) 创建工程对话框



(b) 添加项目对话框

图3-28 创建工程和添加项目对话框

4. 保存了新工程的设置以后，在主窗口中的【Project】标签中是一个空的工程，同时会弹出一个添加项目到工程中的对话框，如图 3-28（b）所示。在这个对话框中可以创建一个新的设计文件、添加一个现有的文件、添加一个目录或

者创建一个仿真配置。

5. 添加现有的两个文件。在添加项目对话框中选择【Add Existing file】，再次弹出一个添加文件对话框，如图 3-29 所示。在这个对话框中可以使用 **Browse...** 按钮浏览计算机的资源，从中选择需要的文件，对于选中的文件可以选择【Reference from current location】或者【Copy to project directory】属性。选中【counter.v】和【tcounter.v】文件后，单击 **OK** 按钮，关闭添加文件对话框；单击 **Close** 按钮关闭添加项目对话框。在主窗口的工程标签中可以看到两个文件，文件后面的“？”符号表示文件没有编译。

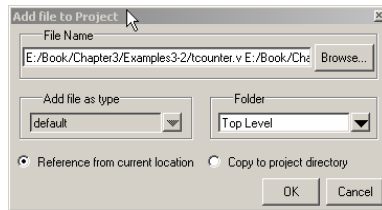


图3-29 添加文件对话框

6. 编译并加载设计。在工程标签的任何位置单击鼠标右键，从弹出的菜单中选择【Compile】/【Compile all】命令。ModelSim 对两个文件编译完成之后，文件后面的状态栏将出现对号，表示编译成功；如果编译不成功将使用错误符号表示，同时主窗口的命令控制台可以看到错误的信息。选择【Library】标签，点击 work 库前方的“+”展开库的内容，work 库中包括了两个设计单元【counter】和【test_counter】。双击【test_counter】设计单元加载仿真。
7. 进行仿真。可以使用仿真的其他命令对这个设计进行仿真。

3.4.1.3 使用资源库的仿真流程

在使用第三方提供的 IP 或者共享一些仿真的公共部分给其他项目成员时都有可能用到资源库仿真流程。本节首先创建一个包含【counter】设计单元的资源库，然后创建一个工程编译一个测试平台，最后连接资源库中包含的【counter】模块进行仿真。

使用资源库的仿真流程，参考示例详见随书所附光盘的“Example-3-1”目录

1. 创建资源库。将随书所附光盘“Example-3-1”目录中的【counter.v】文件拷贝到本地硬盘目录“Book/Chapter3/Examples3-3/ResourceLib”下，作为资源库目录，将目录中【tcounter.v】文件拷贝到本地硬盘目录“Book/Chapter3/Examples3-3”下，作为源文件目录。这里创建了两个目录是为了模拟第三方的资源库使用。
2. 启动 ModelSim。
3. 在主窗口使用【File】/【Change Directory】命令，改变工作目录到“Book/Chapter3/Examples3-3/ResourceLib”目录下，选择【File】/【New】/【Library】命令，打开新建库对话框，选择【a new library and logical mapping to it】选项，并在库名称域（Library Name）填写【parts_lib】，库的物理名称

域 (Library Physical Name) 中软件会自动填写相同的名称, 单击 **OK** 按钮则创建了【parts_lib】资源库, 同时修改了 Modelsim.ini 配置文件。

4. 编译【counter】模块到资源库中。在主窗口中使用  按钮或者使用【Compile】/【Complie】命令打开编译源文件对话框, 如图 3-30 所示, 在【Library】下拉列表中选择【parts_lib】库, 然后双击【counter.v】文件进行编译, 单击 **Done** 按钮关闭编译源文件对话框。

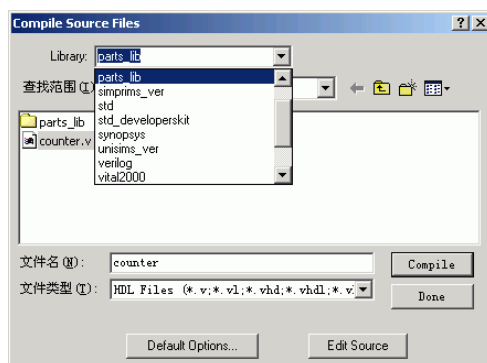


图3-30 源文件编译对话框

5. 创建一个工程。在主窗口使用【File】/【Change Directory】命令改变目录到“Book/Chapter3/Examples3-3”。
6. 在主窗口选择【File】/【New】/【Project】命令, 打开创建工程对话框, 在工程名称域 (Project Name Field) 填写【counter】, 单击 **OK** 按钮关闭对话框。
7. 在出现的添加项目对话框中选择【Add Existing File】, 然后选择【tcounter.v】文件添加到工程中。
8. 使用【Compile】/【Compile Selected】命令编译【tcounter.v】文件。
9. 在主窗口【Library】标签的【work】库下双击【test_counter】启动仿真, 仿真报错如图 3-31 所示, 在提示符后面使用 `error <错误代码>` 命令查看错误的详细说明。

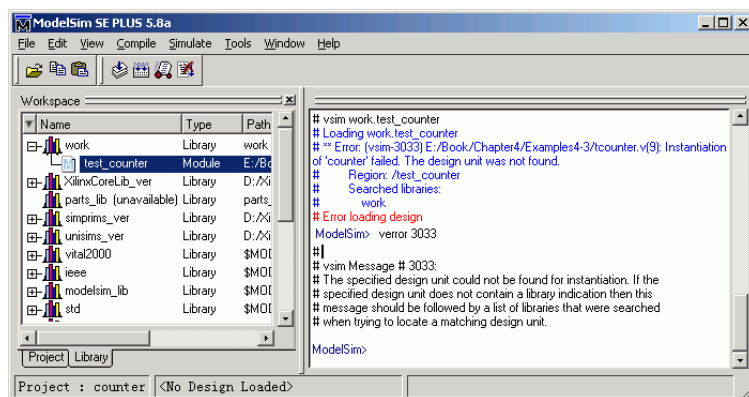


图3-31 错误说明示例

10. 可以看出出错的原因是无法找到指定的设计单元, 因为设计文件存放在资源

库中，所以需要对资源库进行连接。在主窗口中使用  按钮，或者选择【Simulate】/【Simulate】命令，打开的对话框如图 3-32 所示，这个对话框与仿真配置对话框极其类似。单击【Library】标签，在【Search Libraries】栏中单击  按钮，选择编译后的【parts_lib】资源库添加到其中，完成之后  按钮仍然为灰色状态，需要使用鼠标点击【Design】选项卡切换页面，在【work】库中选中【test_counter】设计单元，仿真域【Simulate】中会自动加入 `work.test_counter` 命令，单击  按钮仿真自动执行。

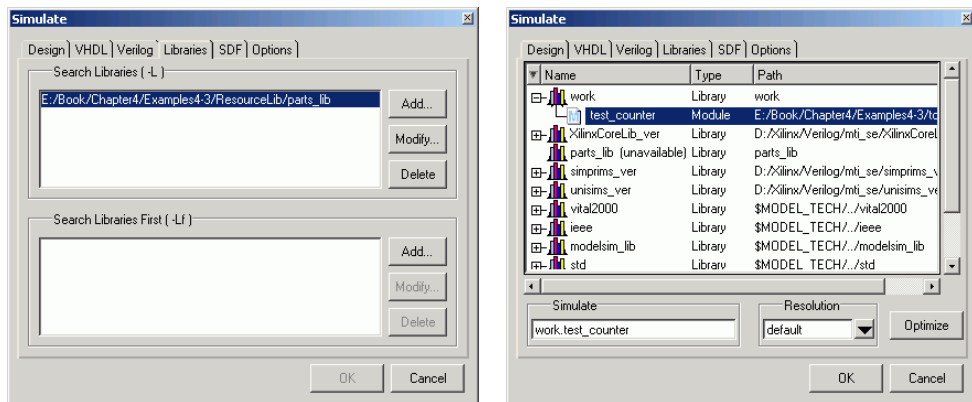


图3-32 Verilog 中连接资源库

11. 进行其他仿真操作。

如果使用 VHDL 编程语言，在建立了资源库，准备使用资源库以前，必须在源代码中使用【LIBRARY】以及【USE】命令对资源库进行引用。

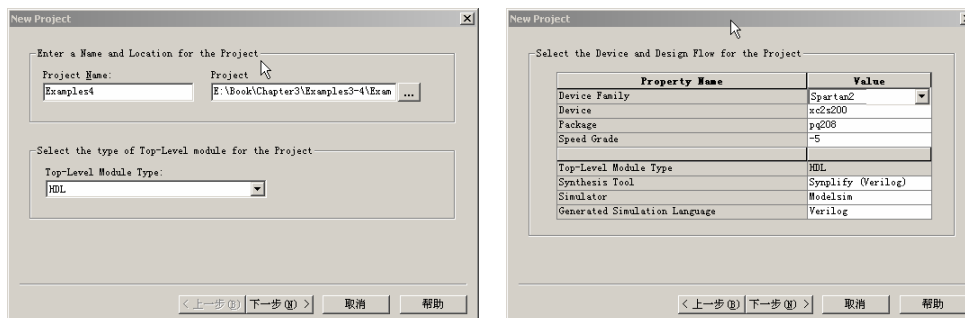
3.4.2 在 ISE 集成环境中进行仿真

ISE 集成开发环境为 ModelSim 仿真软件预留了接口，因此可以从 ISE 集成环境中直接启动 ModelSim 工具进行仿真，下面使用同样的例子说明在 ISE 集成环境中进行仿真的方法。

 **使用 ISE 集成环境中的仿真流程，参考示例详见随书所附光盘的“Example-3-1”目录**

1. 将随书所附光盘“Example-3-1”目录拷贝到本地硬盘的“Book/Chapter3/Examples3-4”目录下。
2. 在 Windows 操作系统中选择【开始】/【程序】/【Xilinx ISE 6】/【Project Navigator】命令，启动 ISE Project Navigator 集成开发环境。
3. 在 ISE 主窗口中选择【File】/【New Project】命令，将弹出【New Project】对话框，如图 3-33 (a) 所示，在【Project】栏中单击  按钮，弹出【浏览文件夹】对话框，在其中选择本地工作目录“E:\Book\Chapter3\Examples3-4”。在【Project Name】栏中输入项目名称“Examples4”，【Project】栏会自动设置为“E:\Book\Chapter3\Examples3-4\Examples4”。

4. 单击 **下一步(N) >** 按钮，打开新建工程对话框的下一个页面，如图 3-33 (b) 所示。在其中选择所使用的器件、模块类型、综合器和仿真器等信息，参考图 3-33 (b) 所示进行设置。

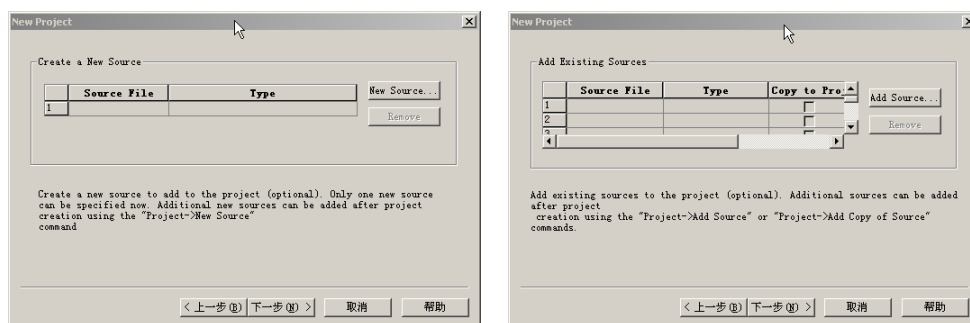


(a)

(b)

图3-33 新建工程对话框

5. 单击 **下一步(N) >** 按钮，打开新建工程对话框的下一个页面，如图 3-34 (a) 所示，询问是否需要新建一个源文件，因为本例中使用了已经编写好的源文件，所以单击 **下一步(N) >** 按钮，打开新建工程对话框的下一个页面，如图 3-34 (b) 所示，单击 **Add Source...** 按钮，打开资源浏览器在其中选择 “counter.v” 和 “tcounter.v” 两个源文件，会弹出一个文件类型选择对话框，如图 3-35 所示，在这个对话框中给 “tcounter.v” 文件选择 “Verilog Test Fixture File” 类型，“counter.v” 文件选择 “Verilog Design File” 类型。



(a)

(b)

图3-34 新建工程对话框

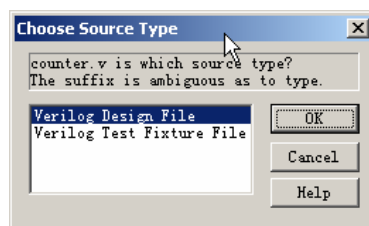


图3-35 文件类型选择对话框

6. 单击 **OK** 按钮后, 出现新建工程内容报告对话框, 其中包括了前面几个步骤的选择, 如果从报告中发现有些内容设置不正确, 可以使用 **< 上一步(B)** 按钮返回并重新选择。
7. 在工程内容报告对话框中单击 **完成** 按钮完成设置, 回到 ISE 主界面, 从源文件视图中选择 “counter.v” 源文件, 当前源文件可以进行的操作出现在资源操作窗口, 如图 3-36 (a) 所示, 选择 “tcounter.v” 源文件, 资源操作窗口如图 3-36 (b) 所示。

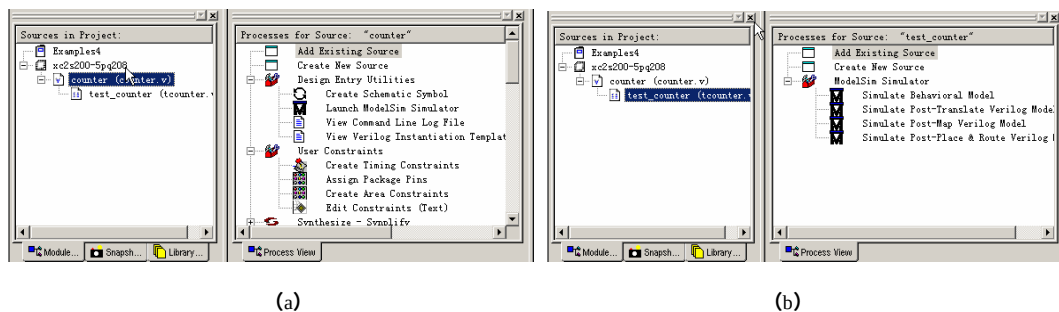


图3-36 源文件以及操作窗口

8. 如图 3-36 (b) 所示, 在 ModelSim 仿真器下面有 4 个不同的仿真选项, 对应 4 个不同的仿真阶段, 行为仿真、翻译后仿真、映射后仿真和布局布线后仿真。左键双击其中任何一个选项都将启动相应阶段的仿真, 如果对应的 Xilinx 器件的仿真库已经按照 3.2 节所述经过了处理, 那么在设计无误的情况下从 ISE 集成开发环境中启动任何一个阶段仿真都可以顺利进行, 如果不是使用 ModelSim XE 版本, 同时没有添加仿真时用到的各种仿真库, 那么只可以顺利进行行为级仿真。

仿真器属性对话框的意义请参考 3.1 节。

3.5 ModelSim 中的调试方法

ModelSim 仿真工具提供了很多调试的方法, 利用好工具的这些特点可以有效提高开发效率。本节就 ModelSim 中常用的调试方法进行说明。

3.5.1 源文件窗口调试

源文件窗口主要用来显示和编辑 HDL 源文件代码, 这个窗口的功能在 ModelSim 5.8 中已经有了很大提高。使用 ModelSim 源文件编辑器以及编辑器中的语言模板可以很容易地编写硬件描述语言的代码, 同时语言模板还提供了一种快捷的源代码编写方法, 如图 3-37 所示, 在编辑 VHDL 或者 Verilog 源代码时都可以通过语言模板来完成, 很多保留的文字等可以通过弹出菜单来选择得到。

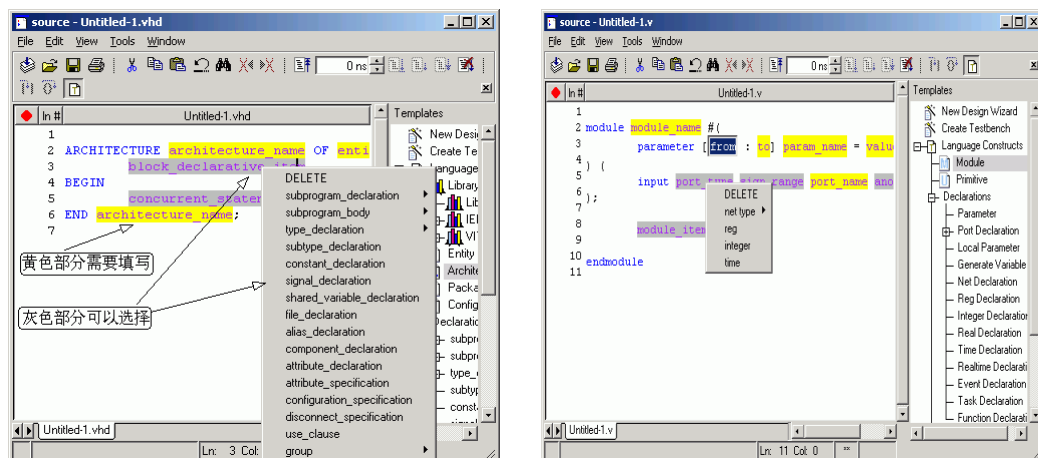
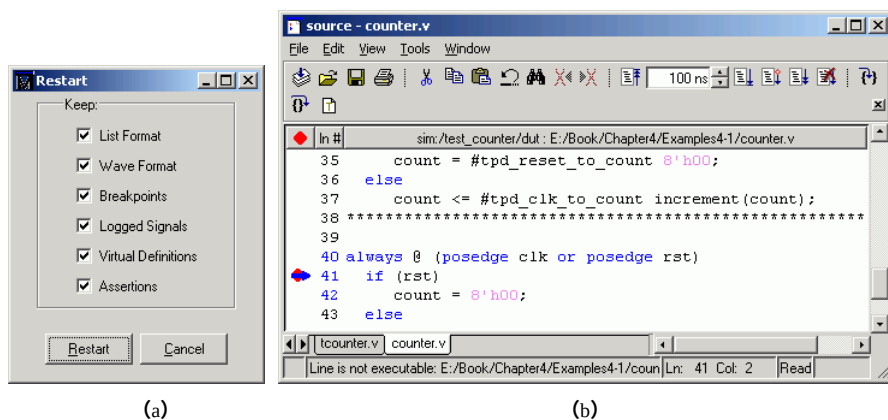


图3-37 使用源文件窗口进行编译

源文件窗口不仅可以用来编辑源代码，还可以用来对程序进行调试。在源文件窗口中可以设置断点调试，同时支持单步调试等，为了方便说明，这里使用“基本仿真流程示例”【Book/Chapter3/Examples3-1】。在执行完基本仿真流程后，再按照以下步骤执行。

- (1) 在【counter.v】程序的 41 行设置一个断点。在主窗口工作区的【sim】标签中选择【dut】层次之后，在源文件窗口中将会打开【counter.v】文件。
- (2) 移动文件找到 41 行，然后使用鼠标在蓝色的行号上点击就会在行号前方出现一个红色的菱形，表示此行设置了一个断点。注意设置断点时需要寻找蓝色的行号，黑色行号说明本行语句不能设置断点，同时需要使用鼠标点击行号。如果再次点击行号，则行号前变成一个空心的红色菱形，表示本行断点已经取消。
- (3) 重新运行仿真。在主窗口、波形窗口或者源文件窗口中单击 按钮，将打开重新运行仿真的对话框，在其中可以选择哪些窗口的信息需要保持，信息保持的意义是项目、信号等保持不变，只是将仿真时间回退到仿真零点，如图 3-38 (a) 所示。单击对话框中的 按钮重新开始仿真。



(a)

(b)

图3-38 【Restart】对话框以及断点显示

- (4) 在主窗口、波形窗口或者源文件窗口中单击 按钮，仿真将一直运行到断点

位置停止，这时红色菱形上出现一个蓝色的箭头，如图 3-38 (b) 所示，在主窗口打印一个中断信息。当仿真停止在断点之后，可以有以下几种方法检查信号的当前值。

- 可以在信号窗口中检查各个信号当前值。
 - 可以在源文件窗口中将鼠标光标悬停在需要查看的信号上，会弹出一个注释说明信号当前值。
 - 可以在主窗口命令控制台使用【examine】命令查看当前值，例如为了得到信号【count】的当前值，可以输入 *examine count* 命令。
- (5) 单步调试。在源文件窗口中使用  按钮进行单步调试，在这种调试方式中源文件窗口中的蓝色箭头会根据仿真器的执行顺序逐行显示。

3.5.2 波形窗口调试

波形窗口是除了主窗口之外使用最多的一个窗口，通过下面的实例将学习在波形窗口中查看仿真波形的一般方法。

波形窗口中的调试方法，参考示例详见随书所附光盘的“Example-3-1”目录

1. 将随书所附光盘“Example-3-1”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-5”目录下。
2. 启动 ModelSim，并改变当前目录到“Book/Chapter3/Examples3-5”。
3. 加载【test_counter】设计单元进行仿真（以上步骤可以参考基本仿真流程示例）。
4. 向波形窗口中添加项目（一般在波形窗口中只可以添加信号，这里说项目是因为可以通过添加例如实例化的模块等方式来添加模块中相关的信号）
ModelSim 提供了多种向波形窗口添加项目的方法。
 - 从信号窗口添加。
在主窗口选择【View】/【Wave】命令打开波形窗口。
在主窗口选择【View】/【Signal】命令打开信号窗口。
在信号窗口中选择【Add】/【Wave】/【Signals in Design】命令，将 3 个信号添加到波形窗口。
在波形窗口中选择【Edit】/【Select All】命令，选中所有信号，然后使用【Edit】/【Delete】命令删除选中信号。
 - 从其他窗口选中并拖动。
可以从主窗口、信号窗口、变量窗口或结构窗口等多个窗中中选择需要查看的项目并拖动到波形窗口中。例如在主窗口中的【sim】视图选择实例化后的【dut】模块，拖动到波形窗口中，在波形窗口中将显示与这个模块相关的所有信号。
在波形窗口中使用【Edit】/【Select All】命令选中所有信号，然后使用

【Edit】 / 【Delete】 命令删除。

- 使用命令行添加项目。

在主窗口命令控制台【VSIM>】命令提示符后键入 `add wave *` 命令将加入当前区域内的所有信号。需要注意这个命令键入时应该是 `add wave <空格> *`，不然会报错。

5. 缩放波形显示。

- (1) 在波形窗口的工具栏中有很多用来进行显示缩放的快捷按钮，通过这些按钮可以很方便地控制波形的显示大小，如图 3-39 所示。

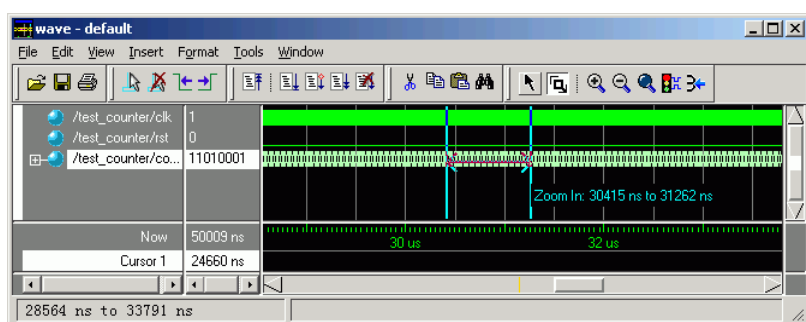


图3-39 缩放波形显示

- (2) 在工具栏中单击【Zoom Mode】按钮，光标将显示为形态，在波形窗口中点击一次将会出现一个蓝色线，再次点击出现第二个蓝色线，两次点击完成之后，两条线中间的部分将会被放大到整个波形窗口视图区域。如果将变成形态的光标向上方移动将将进行缩小显示操作。
- (3) 选择【View】 / 【Zoom】 / 【Zoom last】命令，波形窗口的视图区将显示上一次显示的结果。
- (4) 选择工具栏中的以及按钮相应地可以将波形窗口缩小一半或者放大一倍。
- (5) 选择工具条栏的按钮或者使用【View】 / 【Zoom】 / 【Zoom Full】命令可以将波形完整地放置在显示区域。
6. 在波形窗口中使用指针。
 - (1) 在波形窗口中指针标记着仿真的时间点，ModelSim 第一次绘制一个波形时，指针总是停留在仿真零点，在窗口显示区域的任何位置点击鼠标都会将指针移动到此处。在波形窗口中可以对指针进行命名、锁定、添加和删除等操作，使用指针可以测量时间间隔、快速查找信号翻转等。
 - (2) 在波形窗口的工具栏中选择按钮或者使用【View】 / 【Mouse Mode】 / 【Select Mode】命令，按住鼠标左键，光标将随着鼠标在窗口中移动，鼠标点击之后光标将停留在点击的位置，如图 3-40 所示。
 - (3) 光标在一定的范围之内总会自动附着在信号前向的翻转处，这个特性可以通过【Tools】 / 【Window Preferences】命令中的【Snap Distance】参数来调节，这个参数中为显示的像素数目，默认值为 10。

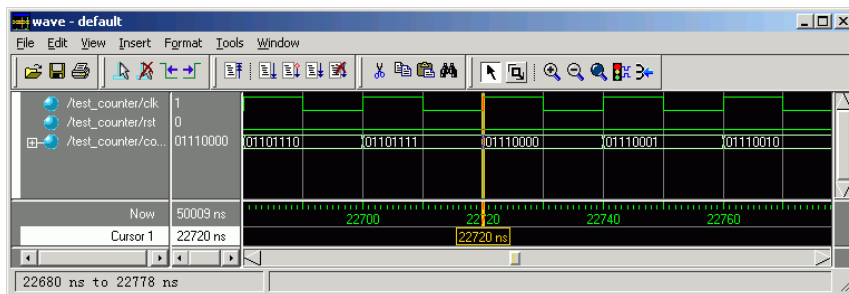


图3-40 波形窗口中的光标

- (4) 使用鼠标右键点击光标的名称，这个区域就变得可以编辑，直接在其中输出光标的名称【A】即可，如图 3-41 所示，请与 3-40 图示的同一个区域进行对比。

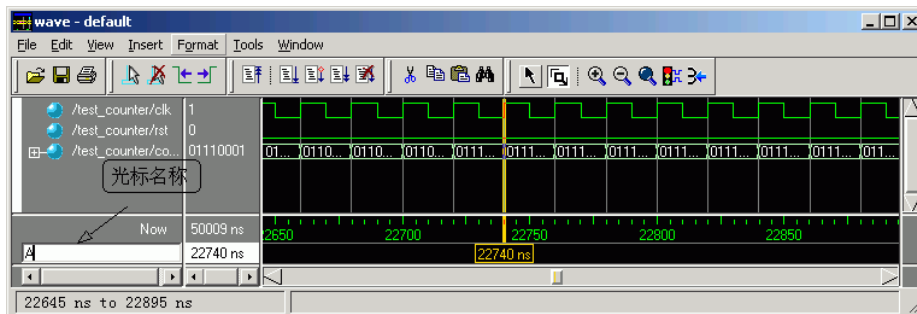


图3-41 光标重命名

- (5) 选择信号【count】，单击工具栏中的 按钮，光标将移动到【count】信号的下一个翻转处，同样如果单击工具栏中的 按钮，光标将移动到【count】信号的前一个翻转处。
- (6) 单击工具栏中的 按钮可以在添加第二个光标的同时修改光标名称为【B】，选中光标【B】并移动可以动态测量与光标【A】之间的距离。
- (7) 为了测量方便，可以将光标【B】锁定在 180ns 处。选中光标【B】，选择【Edit】 / 【Edit Cursor】命令，弹出如图 3-42 (a) 所示的对话框，选择【Lock cursor to specified time】项前的选择框对光标锁定，锁定后的光标变成红色，如图 3.42 (b) 所示。

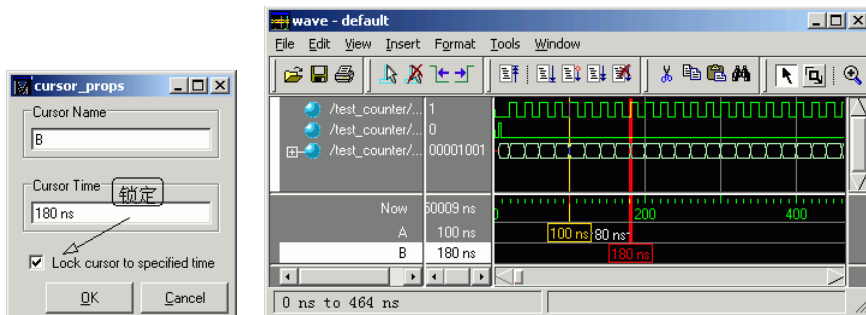


图3-42 光标锁定对话框以及锁定后的光标

- (8) 选中某个光标之后可以使用【Edit】/【Delete Cursor】命令将选定的光标删除。

7. 保存波形窗口格式。

- (1) 一般情况下，当关闭波形窗口后，窗口中的配置信息如添加的信号、设置的光标等丢失了，每次打开需要重新配置。其实可以通过存储窗口格式的方法将信号设置等属性存储成一个 DO 文件，下次打开波形窗口时加载 DO 文件就可以了。

- (2) 保存 DO 文件。

在波形窗口中选择【File】/【Save】/【Format】命令打开保存对话框，设置文件名称为 wave.do 并保存。

保存后的 DO 文件内容大致如下：

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic /test_counter/clk
add wave -noupdate -format Logic /test_counter/rst
add wave -noupdate -format Literal /test_counter/count
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {A {100 ns} 0} {B {180 ns} 1}
WaveRestoreZoom {0 ns} {464 ns}
configure wave -namecolwidth 119
configure wave -valuecolwidth 54
```

可见 DO 文件中保存的都是在命令提示符后面可以输入的一些命令。

- (3) 加载 DO 文件。

可以在打开的波形窗口中使用【File】/【Open】/【Format】命令打开 DO 文件。也可以在主窗口命令提示符后键入 do wave.do 命令打开波形窗口并进行相应的配置。

3.5.3 数据流窗口调试

数据流窗口（Dataflow）是一般仿真软件都提供的一个通用窗口，通过这个窗口可以跟踪设计中的物理连接以及设计中各种事件的传播，也可以用来跟踪寄存器、网线、进程等，数据流窗口的使用极大地丰富了 ModelSim 工具的调试方法。数据流窗口最大的用途是进行信号追踪，方便查找那些输出错误的原因，通过下面这个例子将介绍如何在仿真过程中使用数据流窗口进行调试。

这个例子是用来验证一个高速缓存模块和一个存储器模块，其中有一个处理器模块用来进行读写请求。在这个程序仿真中有一段出现了未知态，通过数据流窗口将很快找到产生未知态的原因，可以及时进行修改。

🔑 数据流窗口中的调试方法，参考示例详见随书所附光盘的“Example-3-6”目录

1. 将随书所附光盘“Example-3-6”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-6”目录下。
2. 启动仿真器并改变目录到“Book/Chapter3/Examples3-6”。
3. 在命令提示符后键入 `do run.do` 命令执行程序。使用文本编辑器打开 DO 文件对其中的指令进行分析。

```
# Copyright Model Technology, a Mentor Graphics
# Corporation company 2003, - All rights reserved.
# Simulation script for Dataflow tutorial
onbreak {resume}
# create library
if [file exists work] {
    vdel -all
}
vlib work
# compile all source files
vlog gates.v and2.v cache.v memory.v proc.v set.v top.v
# open debugging windows
set PrefDataflow(geometry) 582x378+1+562
set PrefSignals(geometry) 287x249+738+307
set PrefWave(geometry) 682x352+590+614
set PrefSource(geometry) 534x249+746+0
set PrefMain(geometry) 730x503+1+1
view da si wa
# start simulator
vsim top
# wave signals
add wave /top/p/*
add log -r *
# run simulation
run -all
```

使用“#”符号作为注释的标识是很多 UNIX 系统以及 TCL 语言的要求。第一个命令 `onbreak {resume}` 中断正在运行的仿真；`if [file exists work] {` 检查是否存在一个 `work` 目录，如果存在则进行删除操作，因为 ModelSim 会加载 `work` 库中的所有设计，重新运行前并不更新 `work` 库，所以这种检查是很必要的，这一点需要在仿真过程中格外注意；`vlib work` 创建一个新的 `work` 库；`vlog gates.v and2.v cache.v memory.v proc.v set.v top.v` 调用 Verilog 语言编译器对设计进行编译产生仿真使用的伪代码；`set PrefDataflow(geometry)`

582x378+1+562 等语句设置各个窗口的位置；view da si wa 命令打开 dataflow/signals/wave 窗口，这里使用简写也符合 UNIX 平台的习惯；vsim top 使用 top 模块调用仿真器；add wave /top/p/*向波形窗口中添加模块 p 相关的各种信号；run -all 运行仿真。

4. 数据流窗口的一个主要作用就是浏览设计中的各种连接，可以用来检查指定信号、网线寄存器的驱动以及接收等特点。

- (1) 给数据流窗口添加一个要观察的信号。

在主窗口的【sim】视图中选择实例 p（可以在结构窗口中进行同样的操作）。

在信号窗口中选择信号【strb】拖动到数据流窗口中，如图 3-43（a）所示。

- (2) 浏览设计。

在数据流窗口中双击红色的网线，将展开连接【strb】信号的进程，如图 3-43（b）所示。

选择进程【#NAND#41】中的信号【test】，单击 （Expand net to all drivers）按钮显示所有驱动。

选择进程【#ALWAYS#146】中的信号【oen】，单击 （Expand net to all readers）按钮显示所有扇出。

使用类似的方法选择某一信号，单击 （Expand net to all drivers and all readers）按钮显示所有驱动以及扇出。

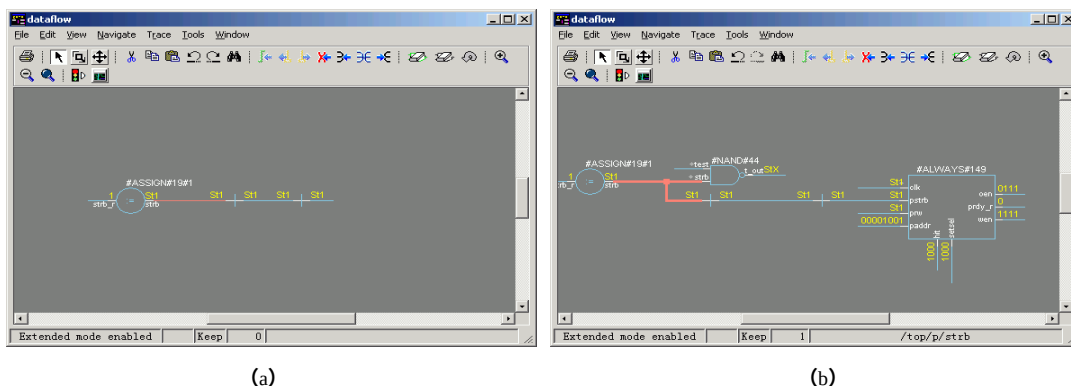


图3-43 数据流窗口中的信号显示

- (3) 单击  按钮，擦除数据流窗口中所有的显示。

5. 数据流窗口中的另外一个重要的特点是通过事件追踪来定位输出信号，数据流窗口中内置的波形查看窗口给类似操作提供了极大的方便。

- (1) 添加一个项目到数据流窗口中。

在主窗口的【sim】视图中选择实例【p】（可以在结构窗口中进行同样的操作）。

从信号窗口中选择【t_out】信号，并拖动到数据流窗口中。

在数据流窗口中选择【View】/【Show Wave】命令或者使用  按钮打开内置的波形查看器，如图 3-44（a）所示。

- (2) 追踪与非门的输入。

在数据流区域内选择【#NAND#41】进程，与这个进程有关的所有输入以及输

出信号都将自动显示在内置的波形窗口中。

在波形窗口中将光标滚动到 2785ns 位置，信号【t_out】的最后一个翻转点，点击设置一个光标，如图 3-44 (b) 所示。

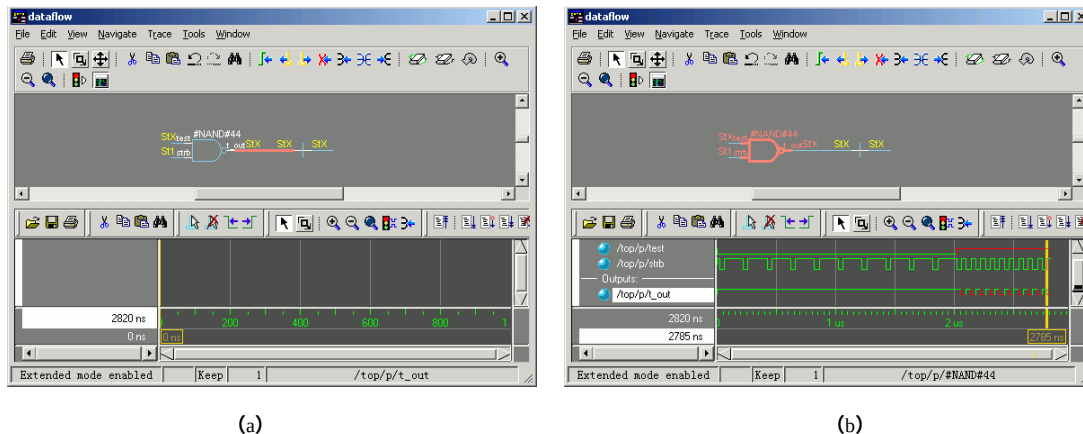


图3-44 数据流中信号追踪

- (3) 选择【Trace】/【Trace next event】命令追踪前一个相关事件，同时在这个事件上添加了一个光标，使用类似的方法可以追踪更多相关事件。
- (4) 在进行下面的练习前关闭数据流窗口。
6. 追踪未知信号‘X’。使用数据流窗口可以很快确定产生未知信号的原因，数据流窗口和标准的波形窗口是互相联系的，所以可以在波形窗口中观察一个信号然后在数据流窗口中对这个信号进行追踪，进而查找问题的源头。
- (1) 在数据流窗口以及波形窗口中查看信号【t_out】。

在波形窗口中找到信号【/top/p/t_out】，这个信号从 2065ns 开始出现了一个未知态（在波形窗口中使用红色线段表示），然后一直在 1 和未知态之间进行翻转，如图 3-45 所示。

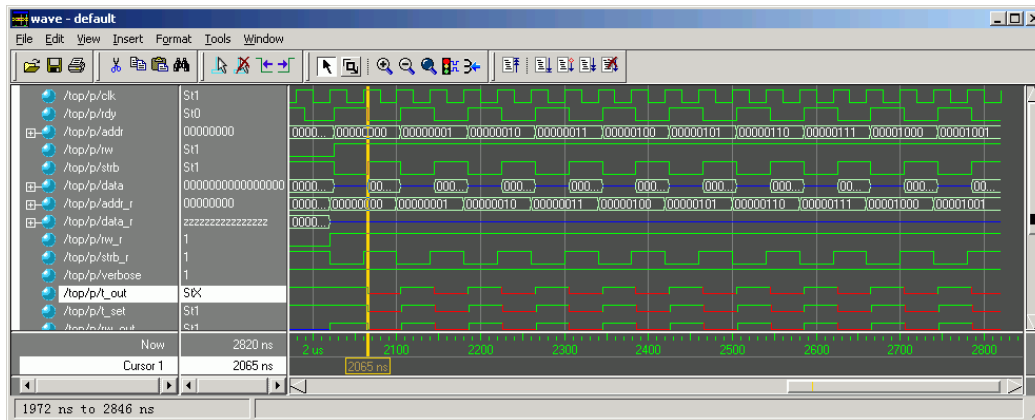


图3-45 波形窗口中的未知态

在信号【t_out】的最后一个翻转 2785ns 处双击这个信号，将自动打开数据流

窗口并显示【t_out】的波形，在波形窗口中移动鼠标光标，数据流窗口中的对应信号值会随着更新，如图 3-46 (a) 所示。

(2) 追踪未知态。

在数据流窗口中选中【t_out】，然后选择【Trace】/【ChaseX】命令对未知态进行追踪，可见产生未知态的原因是【#NOR#43】的输入是一个 st0 (_rw) 与一个 HiZ (test_in)，因此输出【test2】是未知态，在内置的波形窗口中所有的与这个过程有关的信号全部添加到窗口中，如图 3-46 (b) 所示。

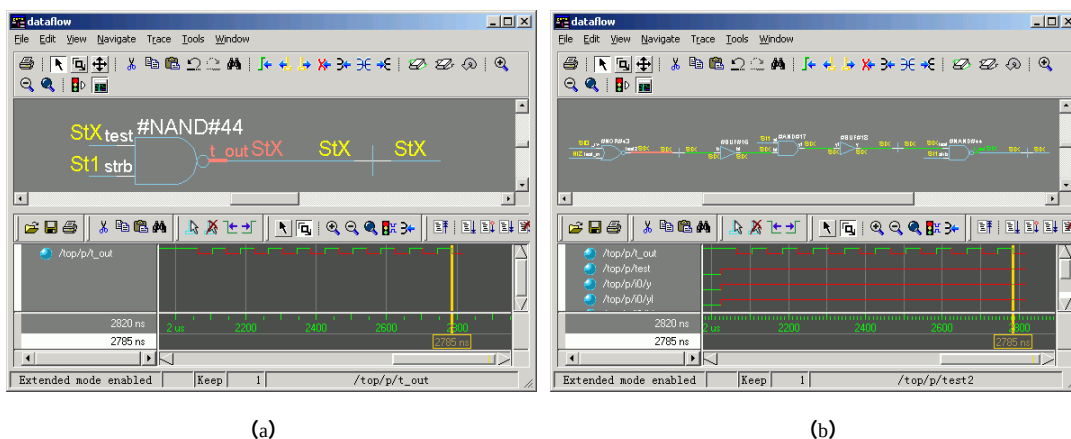


图3-46 追踪未知态

(3) 单击 按钮，擦除数据流窗口中所有的显示，使用 按钮关闭数据流窗口中内置的波形查看器。

7. 在数据流窗口中显示层次。

(1) 改变显示层次选项。

在数据流窗口中菜单栏中使用【Tools】/【Options】命令打开数据流选项对话框，这个对话框有两个不同的选项卡，如图 3-47 所示。

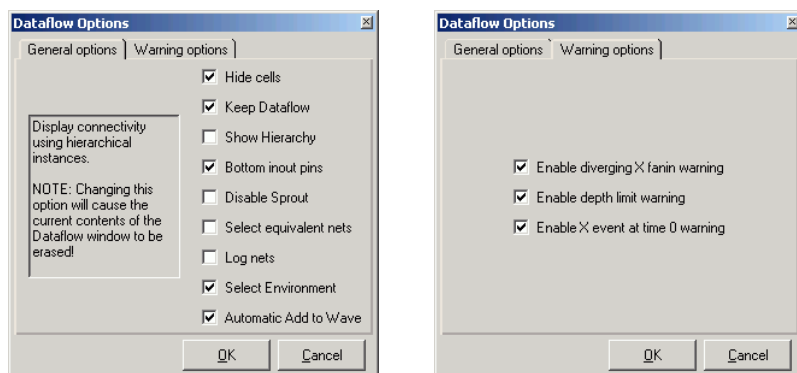


图3-47 数据流选项对话框

在【General options】选项卡中选择【Show Hierarchy】复选框，然后单击 按钮。

(2) 添加【t_out】信号到数据流窗口中。

在主窗口中【VSIM>】提示符后键入 `add dataflow /top/p/t_out` 命令。

在数据流窗口中将显示与信号【t_out】相关的各种模块层次关系。

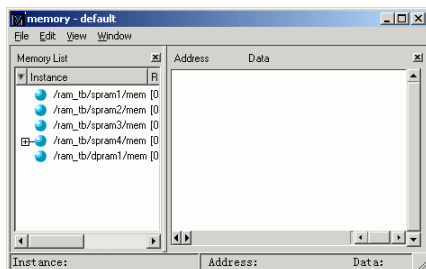
- (3) 在主窗口中【VSIM>】提示符后键入 `quit -sim` 命令结束仿真。

3.5.4 存储器窗口调试

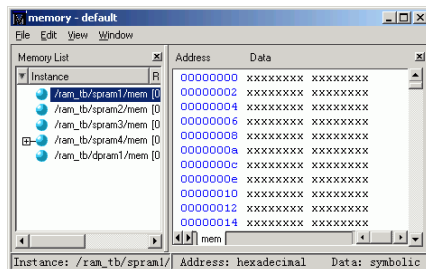
存储器窗口是 ModelSim 5.8 以后的版本中新添加的一个窗口，在这个窗口中可以查看以及初始化设计中的存储，同时在存储器窗口中当前的数据可以保存成一个存储器文件，便于以后使用。通过下面的例子可以学习如何在存储器窗口中查看存储器内容，如何通过各种方式对存储器进行初始化等操作。

🔑 数据流窗口中的调试方法，参考示例详见随书所附光盘的“Example-3-7”目录

1. 将随书所附光盘“Example-3-7”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-7”目录下。
2. 启动 ModelSim 仿真器并改变目录到“Book/Chapter3/Examples3-7”。
3. 创建工作目录、编译设计。
 - (1) 在主窗口的【ModelSim>】提示符后键入 `vlib work` 命令创建工作目录。
 - (2) 在主窗口的【ModelSim>】提示符后键入 `vlog sp_syn_ram.v dp_syn_ram.v ram_tb.v` 命令编译设计文件。
 - (3) 在主窗口的【ModelSim>】提示符后键入 `set NumericStdNoWarnings 1` 命令。
4. 加载设计。
 - (1) 在主窗口的【Library】视图中，点击【work】库旁边的“+”号。
 - (2) 双击【ram_tb】设计单元，加载设计进行仿真。
5. 查看一个存储器。
 - (1) 在主窗口中选择【View】/【Memory】命令打开存储器窗口，在存储器窗口的【List】区域显示了当前设计中使用到的所有存储器，如图 3-48 (a) 所示。
 - (2) 在存储器列表中选择【ram_tb/spram1】实例，存储器窗口中将显示这个存储器的地址以及数据等内容，如图 3-48 (b) 所示。



(a)



(b)

图3-48 存储器窗口

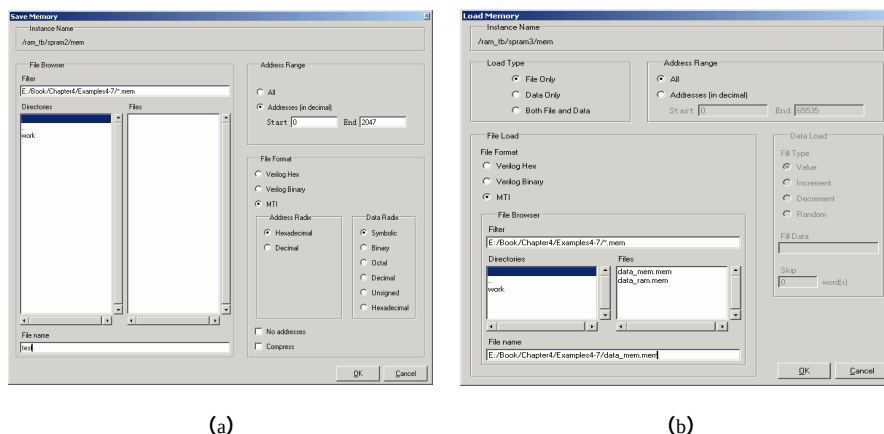
- (3) 在主窗口【sim】视图选择【spram2】实例，存储器列表区域自动更新显示为【spram2】实例。但是，存储器的地址数据区域仍然显示【spram1】实例的信息，直到在存储器列表中选择了【spram2】以后地址数据区域才会更新。一般条件下，存储器列表是根据当前的设计单元动态更新的，例如在主窗口的【sim】视图选择某个存储器后列表会立即进行更新，也可以通过【File】/【Environment】/【Fix to Current context】命令将存储器窗口中的信息固定起来。
- (4) 单击主窗口工具栏中的按钮，运行仿真到结束点。
- (5) 存储器窗口中的地址数据区域随着仿真的进行自动更新。可以通过存储器窗口中的【View】/【Display options】命令打开显示选择对话框改变显示的效果，也可以使用【Edit】菜单中的各种命令对存储器中的内容进行查找、改变等各种操作，这些操作都比较简单，请自行学习使用。
6. 将存储器的内容保存成一个文件。在主窗口的【sim】视图选择【/ram_tb/spram1】，在存储器窗口的存储器列表中选中【/ram_tb/spram1】实例，地址数据区域将显示存储器内容；在存储器窗口中选择【File】/【Save】命令，打开保存窗口，如图 3-49 (a) 所示，在窗口中可以选择存储的地址边界、文件类型等。文件类型中选择【MTI】，地址基数中选择【Decimal】，数据基数中选择【Binary】，文件名称区域中键入 data_mem.mem，单击  按钮保存文件。使用一般的文本编辑器打开保存后的文件，显示如下：

```
// memory data file (do not edit the following line - required for mem
load use)
// instance=/ram_tb/spram1/mem
// format=mti addressradix=d dataradix=b version=1.0
0: 00101000 00101001 00101010 00101011 00101100 00101101 00101110
00101111
8: 00110000 00110001 00110010 00110011 00110100 00110101 00110110
00110111
16: 00111000 00111001 00111010 00111011 00111100 00111101 00111110
00111111
```
7. 初始化一个存储器。在 ModelSim 中有 3 种方法来初始化一个存储器，分别如下：
 - 通过加载保存好的文件初始化存储器。
在存储器窗口中选择【ram_tb/spram3】实例。
选择【File】/【Load】命令，打开一个文件加载对话框，如图 3-49 (b) 所示，在文件列表中选择【data_mem.mem】，单击  按钮加载存储器初始化文件。
 - 部分使用保存好的文件，部分使用模板。
在存储器窗口中选择【ram_tb/spram3】实例。
选择【File】/【Load】命令，打开一个文件加载对话框，在【Load type】域选

择【Both File and Data】，在文件列表中选择【data_mem.mem】，在【Address】域输入地址区域 0 到 300。这些选择意味着需要给 0 到 300 之间的地址空间赋值，如果【data_mem.mem】中的内容只包含 0 到 250 地址空间，那么剩余空间 251 到 300 使用制定的数据模板填充。

填充类型中选择【Increment】，在填充数据中填写初始化数据种子。

- 使用数据模板填充的方式与第 2 种方法类似。



(a)

(b)

图3-49 保存存储器内容

3.5.5 变量窗口调试

在变量窗口中可以查看 VHDL 中的常数、类属和变量等，可以查看 Verilog 中的寄存器以及变量等。使用变量窗口中显示的信息可以帮助对问题的定位。本节将使用一个加法器的例子说明在实际工作中如何使用 ModelSim 进行代码调试。这个例子程序是一个 8bit 加法器，其中包含了行为级和结构级的描述，并且提供了一个含有测试向量的测试程序，在这个测试程序中有一个测试向量的设计是错误的，可以通过变量窗口查找这个错误并且纠正。通过这个例子还可以学习如何映射一个逻辑库到实际的工作库当中，如何改变默认的仿真时间精度，如何从命令控制台中得到断言消息，如何改变断言违规级别，如何使用 *restart* 命令重新开始仿真等操作。

🔑 数据流窗口中的调试方法，参考示例详见随书所附光盘的“Example-3-8”目录

1. 将随书所附光盘“Example-3-8”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-8”目录下。
2. 启动仿真器并改变目录到“Book/Chapter3/Examples3-8”。
3. 在主窗口的命令控制台【Modelsim>】提示符后键入 `vlib library_4` 命令创建一个新的工作库。
4. 在命令提示符后键入 `vcom -work library_4 gates.vhd adder.vhd testadder.vhd` 命令，编译这三个文件到工作库中。

5. 在命令提示符后键入 `vmap work library_4` 命令，映射新建的库到工作库中，同时这个命令将修改 ModelSim 的配置文件 `Modelsim.ini`。没有运行这个命令以前的操作对 ModelSim 的图形界面没有影响，即在没有执行 `vmap` 以前在主窗口的【library】视图没有新的工作库添加。
6. 在主窗口中选择【Simulate】/【Simulate】命令，打开仿真参数对话框，如图 3-50 所示。在对话框的【Design】选项卡中单击【work】库旁边的“+”号，显示其中包含的所有实例，选中【test_addr_structural】，在【Simulate】栏中将自动添加 `work.test_addr_structural`，在【Resolution】中选择【default】，通过这个选项可以控制仿真事件精度。点击 **OK** 按钮加载仿真。

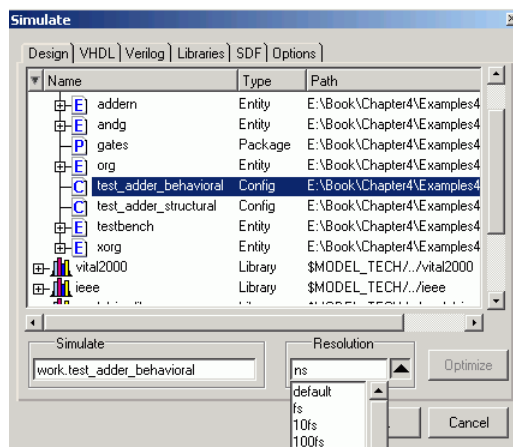


图3-50 加载仿真对话框

7. 在主窗口输入 `view *` 命令，弹出仿真器的全部窗口。
8. 输入 `add list *` 命令将顶层模块的全部信号添加到列表窗口中。
9. 输入 `add wave *` 命令将顶层模块的全部信号添加到波形窗口中。
10. 输入 `run 1000ns` 命令，运行仿真 1000ns。
11. 从命令控制台反馈的信息可以看出，仿真在运行过程中出现了断言失败，如图 3-51 所示。

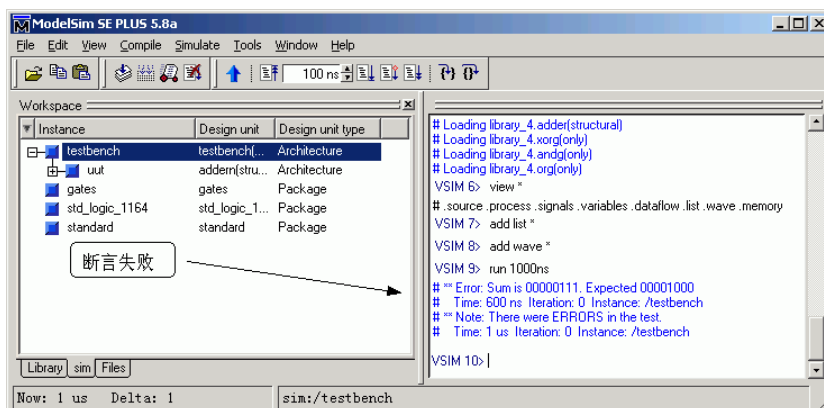


图3-51 断言失败反馈信息

12. 为了找到错误所在，进行如下操作。改变仿真的断言选项，在主窗口中选择【Simulate】/【Simulation Options】命令打开【Simulation Options】对话框，如图 3-52 (a) 所示，选择【Assertions】选项卡，将【Break on Assertion】设置为【Error】，然后单击 **Apply** 按钮，这个选项将使得仿真在断言失败时停止运行，可以对引起断言失败的代码进行定位。
13. 重新运行仿真。单击主窗口或者波形窗口中的  按钮，弹出【Restart】对话框，如图 3-52 (b) 所示，单击 **Restart** 按钮，仿真重新回到起始点，在命令控制台输入 `run 1000ns` 命令重新进行仿真。

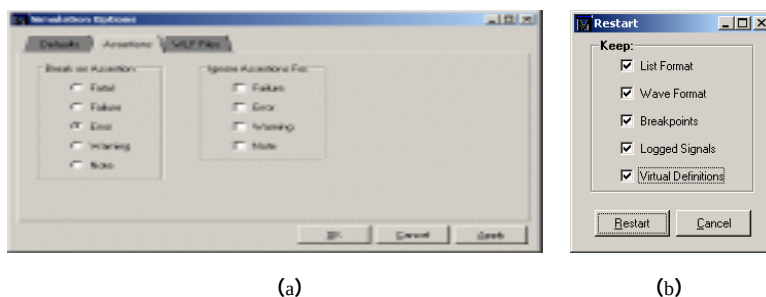


图3-52 仿真控制以及【Restart】对话框

14. 仿真将在断言失败时停止，在源文件窗口中，仿真运行指针停止在 108 行，说明这行代码引起了断言失败，


```
-- check the results
if (sum /= vector.sum) then
  assert false
    report "Sum is " & to_string(sum)
    & ". Expected " & to_string(vector.sum);
  found_error := true;
end if;
```
15. 此时的变量窗口显示如图 3-53 所示，可以看到其中变量“`I`”的当前值为 6，这意味着【`test_pattern`】进程在进行第 6 次迭代时出现了错误。点击【`test_pattern`】左边的“+”符号展开层次，再次点击“[6]”左边的“+”符号，可以看到其中“`.a`”的值为“00000101”；“`.b`”的值为“00000001”；“`.cin`”的值为“1”；“`.sum`”的值为“1000”；这个程序为一个 8bit 的全加器，信号 `a`、`b`、`cin` 为输入，`sum` 信号为输出，使用现在的这些值输出的 `sum` 应该为“00000111”，而在测试激励向量中 `sum` 的值为“00001000”，由此可见错误在测试向量中。为了验证这个判断是否正确，必须重新运行仿真，需要仿真器回到仿真起始点，在命令控制台中使用 `restart -f` 命令将仿真返回到起始点（在命令行使用 `-f` 选项后仿真器将不再询问直接回到起始点，不会出现图 3-52 (b) 所示的对话框）。

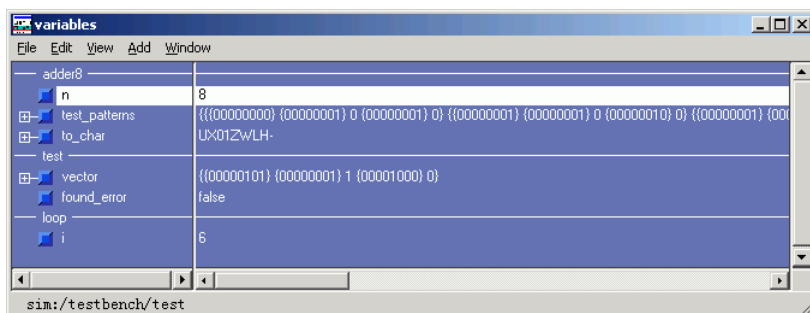


图3-53 变量窗口调试

16. 仿真复位后，变量窗口中的显示被清除。在进程【process】窗口中选中【test】进程，变量窗口中就会出现与这个进程相关的变量，点击【test_pattern】左边的“+”符号展开层次，再次点击“[6]”左边的“+”符号展开，然后选中“.sum”变量，选择菜单栏中的【Edit】/【Change】命令，弹出【Change Selected Variable】对话框，如图 3-54 所示。将其中的【Value】设置为“00000111”，单击 **Change** 按钮保存。



图3-54 改变变量当前值

17. 在主窗口命令控制台使用 `run 1000ns` 命令重新进行仿真，控制台将显示“Test completed with no errors”表示仿真成功。

3.5.6 列表窗口调试

默认条件下，列表窗口中任何一个信号发生变化都会触发列表窗口新一行数据的显示，但是有时用户并不需要这样的信息，大量的信息反而干扰了正常工作，此时可以通过对列表窗口触发条件的一些设置来得到自己关心的数据。

下面将继续使用【Book/Chapter3/Examples3-8】例子学习通过一些必要的设置改变列表窗口的触发条件。

1. 在列表窗口中选择【Tools】/【Window Preference】命令，打开列表窗口属性对话框。
2. 选择其中的触发选项卡【Triggers】，并进行以下操作：在【Trigger On】域中，取消【Signals Change】选项，选中【Strobe】选项；在【Strobe Period】栏中输入“100ns”；在【First Strobe at】栏中输入“70”，选项设置完成后，单击 **OK** 按钮确认。
3. 在窗口中选中某一个信号或者全部信号，然后在菜单栏中选择【View】/

【Signal Properties】命令打开相应的对话框，可以在对话框中改变信号显示的名称以及信号数据显示的数值类型。

4. 图 3-55 显示了【Book/Chapter3/Examples3-8】项目中列表窗口的情形，图 3-55 (a) 所示为设置前默认的显示效果，图 3-55 (b) 所示为设置后的显示效果。

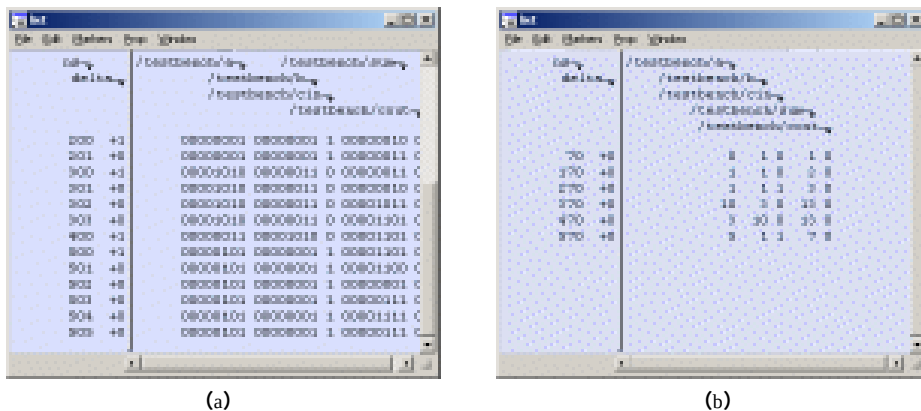


图3-55 不同设置的列表窗口显示效果

3.6 ModelSim 的其他常用操作

在本小节中，将说明仿真中其他的常用操作，之所以与前面的操作区别开来因为这些操作并不是必须的，但是有效使用这些操作可以大大提高仿真的效率。

3.6.1 自动仿真

在使用 ModelSim 进行仿真时，有时需要添加一些仿真库文件或者一些标注文件，还需要对仿真进行有效的控制，在这种情况下使用 ModelSim 的图形用户接口在每一次仿真前都需要经过很多重复配置，对于复杂的仿真显得及其繁琐，因此 ModelSim 提供了另外的一些方法进行仿真，其中一种就是通过宏文件自动执行仿真过程。

在 ModelSim 中宏文件通常使用“do”作为文件后缀，所以一般将宏文件也叫 DO 文件，DO 文件是一个包含了很多命令的脚本文件，这个文件可以使用一系列的 ModelSim 命令组成，也可以使用 TCL 语言编写，也可以混合使用两种方式。这种 DO 文件可以在 ModelSim 的 GUI 中执行，也可以不启动 ModelSim 而直接在操作系统的命令行中执行。

可以在任何一个文本编辑器里创建一个 DO 文件，也可以在 ModelSim 主窗口中使用【File】/【Transcript】/【Save Transcript as】命令将执行过的所有命令保存成一个 DO 文件。

可以在主窗口中选择【Tools】/【Execute Macro】命令执行一个 DO 文件，或者直接使用 do <your_file_name>.do 命令执行 DO 文件，需要注意的是 do 命令只可以在 ModelSim 的命令控制台中使用而不能用在操作系统的命令行中。

为了增加 DO 文件的灵活性，可以在 do 命令行配置各种命令行参数，这些参数将传递

给 DO 文件中的\$1 到\$9 变量，例如如果一个 DO 文件“testdo”中包含一行 bp \$1 \$2，那么通过执行下面的命令将在文件【design.vhd】的 127 行设置一个断点。

```
do testdo design.vhd 127
```

以下是一个简单的 DO 文件的例子，在这个 DO 文件中首先建立了一个工作库并映射它，编译设计源文件，加载设计，添加一些信号到波形窗口中，然后提供了测试激励。

```
vlib work
vmap work work
vlog cntr_rtl.v
vsim work.cntr_rtl
add wave ld
add wave rst
add wave clk
add wave d
add wave q
force -freeze clk 0 0, 1 {50 ns} -r 100
force rst 1
force rst 0 10
force ld 0
force d 1010
run 1700
force ld 1
run 100
force ld 0
run 400
force rst 1
run 200
force rst 0 10
run 1500
```

自动仿真示例，参考示例详见随书所附光盘的“Example-3-9”目录

1. 将随书所附光盘“Example-3-9”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-9”目录下。
2. 启动仿真器并改变目录到“Book/Chapter3/Examples3-9”。
3. 在命令控制台键入 do stimulus.do 命令运行仿真。

ModelSim 中有很多命令可以直接在操作系统的命令控制台进行操作，这个特性给仿真提供了极大方便，通过下面的例子介绍自动仿真的一些其他方法。

自动仿真的其他方法

1. 将随书所附光盘“Example-3-1”目录中的文件拷贝到本地

“Book/Chapter3/Examples3-10”目录下。

2. 打开操作系统的命令控制台，并改变当前目录到“Book/Chapter3/Examples3-10”，在命令控制台键入 `vlib work`。
3. 在操作系统命令控制台键入 `vmap work work`。
4. 在操作系统命令控制台键入 `vlog counter.v tcounter.v`。
5. 使用文本编辑器编写文件“test.do”，这个文件中包括以下内容。

```
view wave
add wave *
add list -decimal *
vcd dumpports -file counter.vcd /test_counter/dut/*
run
quit -f
```

对这些命令说明如下：

- `view wave`：打开一个波形窗口。
 - `add wave *`：添加顶层所有信号到波形窗口中。
 - `add list -decimal *`：添加信号到列表窗口中，并且这些信号使用十进制方式在列表窗口中显示。
 - `vcd dumpports -file counter.vhd ./test_counter/dut/*`：将仿真存储为一个 VCD 文件，文件名为 `counter.vcd`，需要存储的信号为顶层所有信号。
 - `run`：运行仿真。
 - `quit -f`：强制退出。
6. 在命令控制台键入 `vsim -do test.do -wlf waved.wlf test_counter` 命令进行仿真，其中 `vsim` 启动仿真器，命令参数 `-do` 指定使用 DO 文件，命令参数 `-wlf` 指定产生一个名称为“waved.wlf”的波形日志文件。
 7. 执行了上面命令后在目录“Book/Chapter3/Examples3-10”中会产生一个波形日志文件“waved.wlf”，在操作系统的命令控制台输入 `vsim -view waved.wlf` 命令可以打开 ModelSim 仿真软件加载“waved.wlf”文件中的数据。在 ModelSim 的命令控制台中输入 `view wave` 和 `add wave *` 命令后，在波形窗口中就可以看到“waved.wlf”文件中的信号波形。
 8. 上面提到的所有命令（包括对工作目录的改变）都可以在 DOS 命令提示符后直接执行，所以可以将这些命令存储成一个 bat 文件，而将 DOS 不能识别的一些命令存储成宏文件（以 `do` 为扩展名）在 `vsim` 命令中调用，这样就可以将整个仿真过程中使用到的所有命令编辑成几个不同的文本文件，使用这些文件来控制仿真过程。
 9. 根据以上说明可以将这个示例用到的命令编辑成批处理文件“gen_wlf.bat”和“view_wave.bat”，将只能在 ModelSim 环境使用的命令编辑成宏文件，然后采用 `vsim` 的命令行参数提供给仿真器。
 10. 有了这些文件之后可以在操作系统的命令控制台输入 `gen_wlf` 命令，则示例的前 6 个步骤都执行完毕，然后再输入 `view_wave` 命令，那么这个示例的第 7~

8 步操作开始执行。

3.6.2 WLF 文件

ModelSim 仿真器运行仿真过程中会自动产生一个波形日志格式文件（WLF, Wave Log Format）。WLF 文件提供了一组仿真的数据，在这个数据集中记录了指定层次中信号、变量等的仿真数据，可以在仿真结束后使用这个文件对仿真过程进行精确回放，同时可以使用这个文件与正在进行的仿真数据进行对比，得到不同仿真波形的时序差异。

在 ModelSim 中可以同时加载很多个这样的数据集，每一个数据集使用一个逻辑前缀来指示，当前活动的仿真数据集使用“sim:”前缀，其他数据集模型使用 WLF 文件的名称作为前缀，同时打开的所有数据集使用的仿真时间精度要一致。

如果在仿真过程中给数据流窗口、列表窗口或者波形窗口中添加了项目，那么在当前的工作目录下会产生一个名称为“vsim.wlf”的 WLF 文件，如果在相同的目录进行了新的仿真，这个文件会被新产生的仿真数据覆盖。

可以通过以下各种方法保存一个 WLF 文件：

- 在主窗口中使用【File】/【Save】/【sim dataet】命令。
- 在波形窗口中使用【File】/【Save Dataset】/【sim】命令。
- 在命令行使用 vsim 命令时给出-wlf 参数。
- 在命令控制台直接使用 dataset save 命令。

如果没有使用 dataset save 或者 dataset snapshot 命令，为了保存一个有效的 WLF 文件必须使用 quit 或者 quit sim 命令，不然，WLF 文件将不正常关闭，下次打开这个文件时可能会产生“Bad Magic Number”的错误信息。这是因为在仿真结束以前 vsim.wlf 文件的前 8K 地址区域是不写信息的，仿真完整结束后这些地址区域将添加相应的数据。如果仿真崩溃或者因其他原因引起这些信息不完整将产生“Bad Magic Number”报告。

打开一个数据集可以使用以下两种方法：

- 在主窗口中使用【File】/【Open】/【Dataset】命令。
- 在命令控制台直接使用 dataset open 命令。

在打开数据集的对话框中有两个不同的域：【Dataset Pathname】用来指定存放 WLF 文件的路径以及文件名称；【Logical Name for Dataset】可以指定数据集的逻辑名称，这个名称将显示在数据集的前缀中，如果不指定将默认使用 WLF 文件的文件名称。

每一个数据集都将在主窗口的工作区创建一个结构标签，这个标签的名称使用数据集的前缀名称，如图 3-56 所示，其中“sim”是当前的仿真的结构标签，而“waved”是打开的一个 WLF 文件产生的结构标签。在结构窗口中的视图将随着主窗口结构标签的选择而随时更新。

如果打开了多个 WLF 文件，可以使用 GUI 或者命令行的方式来对这些数据集进行管理。在主窗口中使用【View】/【Dataset】命令打开数据集浏览对话框，如图 3-57 所示。

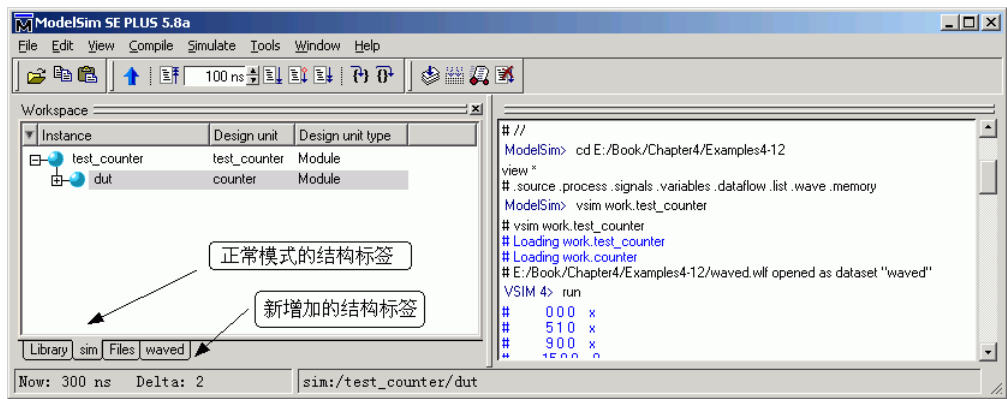


图3-56 增加了数据集的主窗口视图

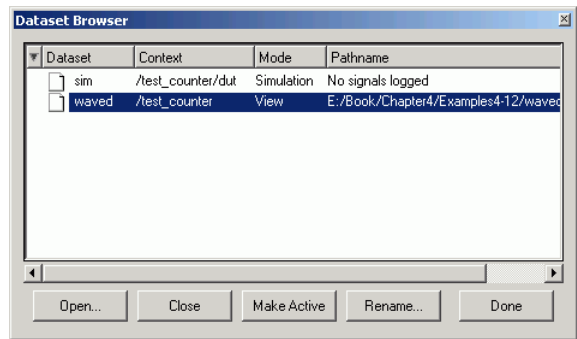


图3-57 数据集的管理

对话框中几个主要按钮的功能介绍如下。

- Open：打开其他的数据集合。
- Close：关闭选中的数据集合，同时从主窗口的结构标签中删除。
- Make Active：确保选中的数据集合是活动的，所谓的活动是指相关的操作都是针对选中的数据集合。
- Rename：数据集合更名。

在波形窗口中使用【Tools】/【Dataset Snapshot】命令可以打开数据集合快照对话框（参见图 3-58），使用数据集合快照可以周期性地将当前的仿真数据拷贝到其他的文件中，使用这个方法可以有效减少 WLF 文件的大小，在大型仿真中数据文件过大会导致操作系统崩溃。

对话框中的主要内容介绍如下。

- 【Dataset Snapshot State】：是否启动数据集合快照。
- 【Snapshot Type】：包含以下两项。
 - 【Simulation Time】：指定每多少个事件单位复制一次，默认是一百万个事件单位。
 - 【WLF File Size】：指定当前仿真的 WLF 文件的大小达到多少兆字节以后复制一次，默认是 100MB。

- 【Snapshot Contents】：包含以下两项。
【Snapshot contains onlu data since previous snapshot】：每一个快照仅仅包括上次快照之后的数据，这个操作将使得 ModelSim 在每一次快照后清除当前的 WLF 文件中的数据。
【Snapshot contains all previous data】：每一次快照将包括全部的仿真信息。
- 【Snapshot Drectory and File】：包含以下两项。
【Directory】：存放文件的目录。
【File Prefix】：文件前缀。
- 【Overwrite/Increment】：包含以下两项。
【Always replace snapshot file】：使用一个文件保存全部快照，每一次新的快照信息将覆盖原先的信息。
【Use incrementing suffix on snapshot files】：为每一个快照创建一个快照文件，添加不同的后缀（例如 vsim_snapshot_1.wlf, vsim_snapshot_2.wlf 等）。

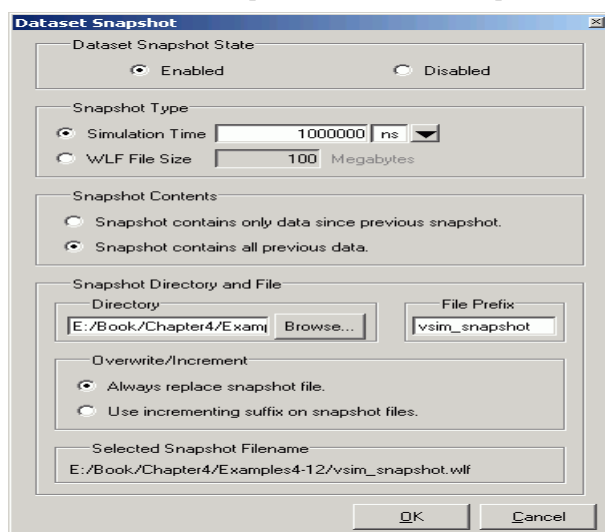


图3-58 数据集快照对话框

3.6.3 波形比较

ModelSim 工具提供了波形比较的功能，使用这个功能可以将当前正在进行的仿真与一个参考数据结合（WLF 文件）进行比较，比较的结果可以在波形窗口或者列表窗口中查看，也可以将比较的结果生成一个文本文件。

波形比较时可以指定特定的信号或者边界进行比较，还可以定义比较公差，可以设置比较的开始以及结束时间点等。

进行波形比较时，工具提供连续模式和时钟模式两种不同的比较模式，连续模式是每个参考信号进行跳变时都将测试信号和参考信号进行比较的一种方式；时钟模式是测试信号和参考信号都使用相同的时钟进行采样，然后对两者的采样值进行比较，在这种模式下可以使用时钟的上升沿、下降沿或者双沿进行采样。

一般的波形比较大致需要以下几个步骤：

- 指定需要进行比较的数据集合或者仿真过程。
- 指定比较的边界以及信号。
- 运行比较过程。
- 查看比较结果。

在下面的练习中，首先将运行一个混合的 HDL 仿真，然后对其中一个文件进行编辑，改变其延时控制，再次运行仿真，最后将两次仿真运行结果进行比较。通过这个练习将介绍如何使用波形比较向导进行波形比较，如何在波形窗口中标记和查看比较结果，如何使用波形比较专用按钮，如何设置比较公差等操作。

波形比较示例，参考示例详见随书所附光盘的“Example-3-11”目录

1. 将随书所附光盘“Example-3-11”目录中的文件拷贝到本地硬盘的“Book/Chapter3/Examples3-11”目录下。
2. 启动 ModelSim 仿真器，并改变工作目录到“Book/Chapter3/Examples3-11”。
3. 在 ModelSim 的命令提示符后键入 `do gold_sim.do` 命令，这个 DO 文件中包括以下命令，这些命令在前面已经详细说明，此处不再重复。

```
vlib work
vmap work work
vlog test_sm.v sm_seq.v sm.v beh_sram.v
vsim -wlf gold.wlf test_sm
add wave *
onbreak {resume}
run 750 ns
quit -sim
```

4. 执行完这个 DO 文件之后，在当前工作目录下会产生一个名称为“gold.wlf”的波形日志文件，在以下的操作中将使用这个文件作为波形比较的参考数据集合。测试数据集合用来和参考数据集合进行比较，同参考数据集合一样，测试数据集合可以是一个已经存储好的数据，也可以是当前仿真的数据集合，或者是当前仿真的数据集合的一部分。
5. 在主窗口的【Library】选项卡中点击工作库【work】旁边的“+”符号，然后选中【test_sm】设计单元，使用右键菜单中的【edit】命令，在源文件窗口中打开“test_sm.v”文件，选择【Edit】/【read only】命令去掉文件只读属性，移动鼠标到 116 行将代码“`@(posedge clk ) wt_wd('h10,'haa);`”修改为“`@(posedge clk) wt_wd('h10,'hab) ;`”，选择【File】/【Save】命令保存修改。
6. 在主窗口的命令控制台键入 `do sec_sim.do` 命令，这个 DO 文件中包括以下命令，对“test_sm.v”文件重新进行编译并重新运行仿真。

```
vlog test_sm.v
vsim test_sm
```

```
add wave *
```

```
run 750 ns
```

7. 进行波形比较。在 ModelSim 中有一个波形比较的向导可以帮助用户一步一步地设置波形比较的条件并进行比较。在波形窗口或者主窗口中选择【Tools】/【Waveform Compare】/【Comparison Wizard】命令打开比较向导，单击 **Browse...** 按钮选择“gold.wlf”文件作为参考数据集，在【Test Dataset】中选择【Use Current Simulation】，如图 3-59（a）所示，单击 **Next >** 按钮进入下一个对话框，如图 3-59（b）所示。在【Comparison Method】中选择【Compare All Signals】。

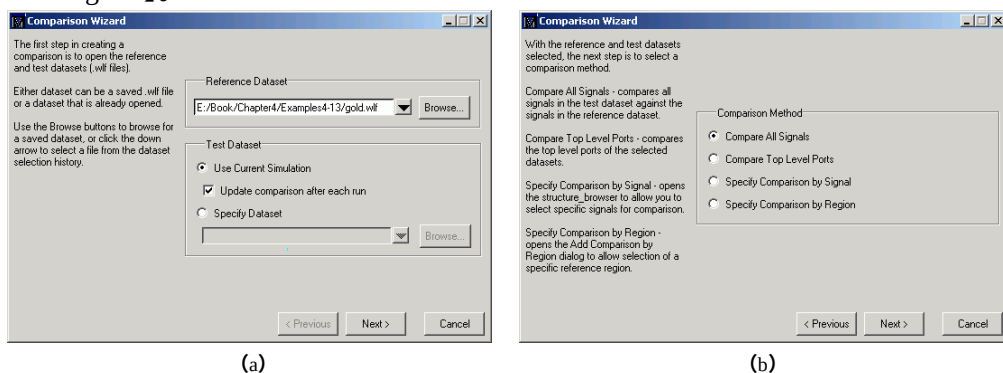


图3-59 波形比较对话框

8. 在接下来的对话框中将【Would you like to add more signals to the comparison】设置为【No】，进入第 4 个对话框后单击 **Compute Differences Now** 按钮，软件将开始计算数据集之间的差异，如图 3-60 所示。

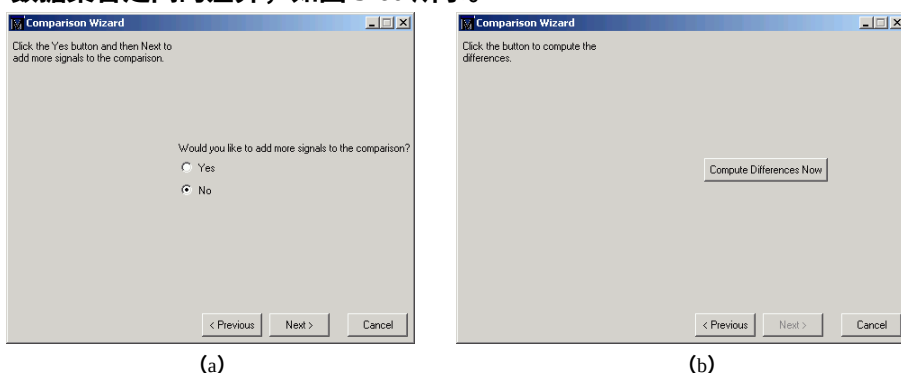


图3-60 波形比较对话框

9. 比较完成后在弹出的对话框中单击 **Finish** 按钮完成比较。

波形比较的结果可以在波形窗口和列表窗口中进行查看，也可以在主窗口中将结果存储成一个文本文件进行查看。比较结束后主窗口中将显示比较的范围以及比较的结果，如图 3-61 所示，可以看到在主窗口的工作区中多了两个视图，其中【gold】视图代表了参考数据集 gold，【compare】视图代表了比较的边界。

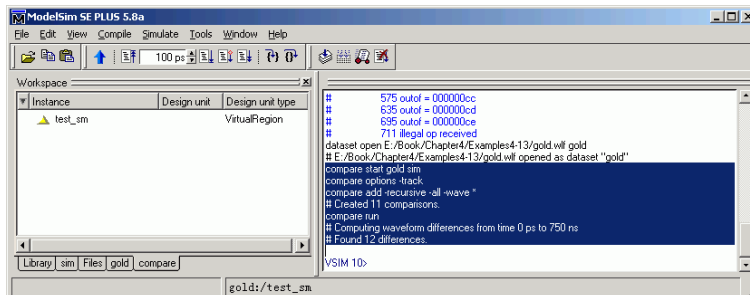


图3-61 在控制台中反馈的波形比较结果

在波形窗口中两个数据集合之间的差异部分使用红色表示，如果将鼠标光标移动到红色区域，就会弹出一个窗口对这个红色区域进行说明。在波形窗口中还包括多个波形比较专用的按钮，利用这些按钮可以快速在不同的信号之间跳动，如图 3-62 所示。

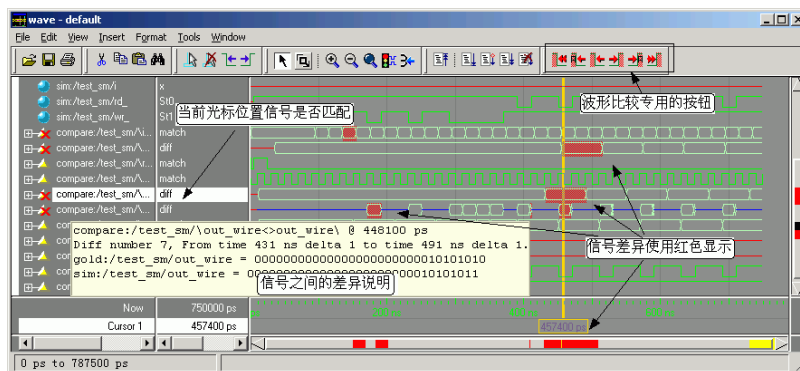


图3-62 波形窗口中比较结果显示

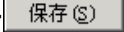
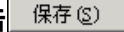
也可以在列表窗口中查看波形比较结果。在主窗口中选择【View】/【List】命令打开列表窗口，并从主窗口的【compare】视图选择波形比较的层次，按住鼠标左键拖动到列表窗口中，在列表窗口中，数据集合之间的不同使用黄色来表示。

除了可以在这两个窗口中对波形比较的结果进行查看以外，还可以将波形比较的结果存储成一个文本文件，在以后需要的时候重新从 ModelSim 中加载。为了达到这样的目的必须保存两个文件，一个文件保存了信号之间的差异，一个文件保存了比较的规则以及配置。

操作过程如下：

1. 在主窗口或者波形窗口中选择【Tools】/【Waveform Compare】/【Differences】/【Write Report】命令，在打开的对话框中单击  按钮将存储一个“compare.txt”文件到当前目录下。
2. 在 ModelSim 的命令控制台键入 notepad compare.txt 命令打开保存的文件，内容如下：

```
Total secondary differences = 6
Number of primary signals with differences = 4
Diff number 1, From time 135 ns delta 0 to time 155 ns delta 0.
gold:/test_sm/into = 000000000000000000000000000010101010
```

- 在主窗口或者波形窗口中选择【Tools】/【Waveform Compare】/【Differences】/【Save】命令，在打开的对话框中单击  按钮将保存一个信号差异文件“compare.dif”到当前目录下。
- 在主窗口或者波形窗口中选择【Tools】/【Waveform Compare】/【Rules】/【Save】命令，在打开的对话框中单击  按钮将保存一个比较规则文件“compare.rul”到当前目录下。
- 在主窗口或者波形窗口选择【Tools】/【Waveform Compare】/【End Comparison】命令结束波形比较。
- 通过以上操作保存了波形比较的结果，如果需要重新加载比较文件时必须事先在波形窗口中打开需要用到的数据集合，例如在本例中可以先使用【File】/【Open】/【Dataset】命令打开参考数据集合“gold.wlf”文件，然后选择【Tools】/【Waveform Compare】/【Reload】命令重新加载波形比较的结果。
- 在波形比较过程中，可以设置比较公差来约束差异到一定的程序才显示。单击波形比较专用的  按钮，光标显示在 135ns。
- 在波形窗口中选中信号【compare:test_sm\into<>into\】，单击鼠标右键选择【Properties】命令，弹出【Wave Properties】对话框，如图 3-63 所示。在【Compare】选项卡中选中【Continuous Comparison】单选项，并且在【Leading Tolerance】栏目中添加数值“50”，在其后的下拉列表中选择【ns】选项，单击  按钮保存设置并关闭对话框。

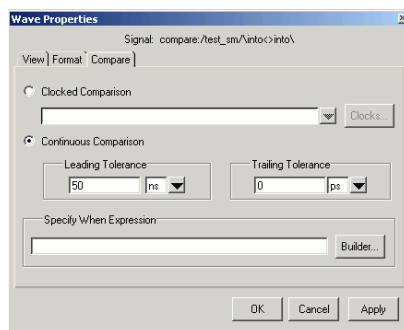


图3-63 比较公差设置

9. 在波形窗口或者主窗口中选择【Tools】/【Waveform Compare】/【Run Compare】命令重新运行波形比较，可以看见信号【compare:/test_sm/into<>into\】上红色的高亮显示消失了，这是因为使用了不同的比较公差引起的。

3.6.4 SDF 文件

SDF 时序标注 (Standard Delay Format Timing Annotation) 是在仿真中经常提到的一个名词, 时序仿真就是经过实现以后产生一个仿真模型, 给这个仿真模型添加了时序标注之后的仿真。关于 SDF 时序标注文件在设计流程中的作用等一些概念请参阅相关书籍, 本节只介绍 SDF 的基本概念, 同时讲解在 ModelSim 使用 SDF 时序标注文件的一些方法。

SDF 时序标注最初使用在 Verilog 语言的设计中, 现在在 VHDL 语言的设计中也引用了这个概念。对于一般的设计者来说并不需知道 SDF 文件的细节, 这个文件一般由器件厂家提供一些工具来产生对应于自己器件的时序标注文件。例如, 在 FPGA/CPLD 设计中, SDF 时序标注文件都是器件厂家通过自己的开发工具提供给设计者的, 在 Xilinx 公司的 FPGA/CPLD 设计中使用 “.sdf” 作为时序标注文件的扩展名, 而在 Altera 公司的 FPGA 设计中使用 “.sdo” 作为时序标注文件的扩展名。在 SDF 标注文件中对每一个底层逻辑门提供了 3 种不同的延时值, 分别是典型延时值、最小延时值和最大延时值。在对 SDF 标注文件进行实例化说明时必须指定使用了哪一种延时值, 例如需要使用 SDF 文件 myfpga.sdf 中的最大延时值标注顶层【testbench】下的【u1】模块, 那么在 ModelSim 中就需要使用 `vsim -sdfmax /testbench/u1=myfpga.sdf testbench` 命令。

一般的设计中都会有很多子模块, 而每一个模块都可以使用不同的 SDF 标注文件, 例如 `vsim -sdfmax /system/u1=fpga1.sdf -sdfmin /system/u2=fpga2.sdf system` 命令使用 fpga1.sdf 文件中的最大延时标注【system】系统下的子模块【u1】, 而使用 fpga2.sdf 文件中的最小延时标注【system】系统下的子模块【u2】。

用户可以在加载仿真时指定时序标注文件, 例如在“时序仿真流程”一节的示例中执行第 6 个步骤时如果没有将时序标注文件拷贝到当前的目录下, 那么也可以通过 GUI 来指定当前的标注文件。在“时序仿真流程”一节示例的第 10 个步骤中增加以下操作: 使用【Simulate】/【Simulate】命令打开【Simulate】对话框, 从中选择【SDF】选项卡, 单击 **Add...** 按钮, 在弹出的对话框中选择需要的时序标注文件, 在【Apply to Region】选择时序标注的边界, 这个边界的含义与命令行中的模块边界类似, 在这个例子中应该填写【/dut】, 这个参数一定要填写正确, 不然无法进行仿真, 单击 **OK** 按钮保存设置 (参见图 3-64)。如果标注文件以及仿真模型文件在同一个目录下, ModelSim 工具将自动对仿真模型进行标注。

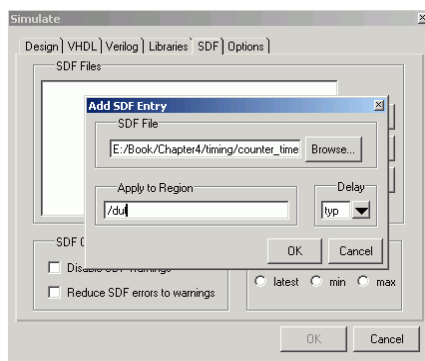


图3-64 添加时序标注文件

3.6.5 VCD 文件

VCD 文件是在 IEEE 1364 标准中定义的一种 ASCII 文件，在这个文件中包含了头信息、变量的预定义和变量值的变化等信息。在 Verilog 语言中支持 VCD 的系统任务，并可以通过在 Verilog 源代码中使用 VCD 系统任务来生成 VCD 文件。ModelSim 中提供了与 IEEE 1364 定义的 VCD 系统任务等效的仿真命令，并且对它进行了扩展，使其可以支持 VHDL 的设计，所以在 ModelSim 中等效的 VCD 命令可以在 VHDL 和 Verilog 设计中使用。

VCD 命令和 IEEE 1364 系统任务的映射关系如图 3-65 所示。

VCD commands	VCD system tasks
vcd add (CR-221)	\$dumpvars
vcd checkpoint (CR-222)	\$dumpall
vcd file (CR-231) ▲	\$dumpfile
vcd flush (CR-235)	\$dumpflush
vcd limit (CR-236)	\$dumplimit
vcd off (CR-237)	\$dumpoff
vcd on (CR-238)	\$dumpson

图3-65 VCD 命令和 IEEE 1364 系统任务的映射关系

ModelSim 5.5 以后的版本支持的扩展 VCD 命名与相关的系统任务关系如图 3-66 所示。

VCD dumpports commands	VCD system tasks
vcd dumpports (CR-224)	\$dumpports
vcd dumpportsall (CR-226)	\$dumpportsall
vcd dumpportsflush (CR-227)	\$dumpportsflush
vcd dumpportslimit (CR-228)	\$dumpportslimit
vcd dumpportsoff (CR-229)	\$dumpportsoff
vcd dumpportson (CR-230)	\$dumpportson

图3-66 扩展 VCD 命名与相关的系统任务

3.6.5.1 创建 VCD 文件

在 ModelSim 中创建 VCD 文件的过程包含两个不同的流程，一个流程可以提供四态的 VCD 文件，参数在 0\1\X\Z 之间变化，没有信号的强度信息；另外一个流程提供一个扩展的 VCD 文件，这个文件包括参数的全部状态变化以及强度信息。

🔑 创建一个四态 VCD 文件，参考示例详见本书附带光盘的“Example-3-1”目录

1. 将随书所附光盘“Example-3-1”目录下的源文件拷贝到本地硬盘的“Book/Chapter3/Examples3-12”目录下。
2. 启动 ModelSim 仿真工具，并改变目录为“Book/Chapter3/Examples3-12”。
3. 在 ModelSim 的命令控制台使用以下命令编译并加载仿真。

```
vlib work
vlog counter.v tcounter.v
vsim test_counter
```

4. 在设计加载以后,使用 `vcd file` 命令指定一个 VCD 文件,利用 `vcd add` 命令添加仿真条目到文件中。命令格式如下:

```
vsim 1 >vcd file myvcdfile.vcd
vsim 2 >vcd add /test_counter/dut/*
vsim 3 >run
vsim 4 >quit -f
```

通过以上的操作就可以在工作目录下生成一个 VCD 文件。

🔑 扩展 VCD 文件流程

1. 在 ModelSim 的命令控制台使用以下命令编译并再次加载以上的设计。

```
vlib work
vlog counter.v tcounter.v
vsim test_counter
```

2. 设计加载之后,使用如下命令创建一个扩展的 VCD 文件并添加仿真条目。

```
vsim 1>vcd dumptports -file muvcdfile.vcd /test_counter/dut/*
vsim 3 >run
vsim 4 >quit -f
```

通过以上的命令,可以创建一个扩展的 VCD 文件。

3.6.5.2 使用一个 VCD 文件重新进行仿真

在使用一个 VCD 文件保存了仿真数据以后,可以在需要的时候将这个仿真重新运行。

一般使用 VCD 文件重新进行仿真包括以下两个步骤:

- 使用 `vcd dumptports` 命令创建一个 VCD 文件。
- 不使用 Testbench,在 vsim 中使用 `-vcdstim` 参数进行仿真。

在下面的例子中,仿真一个 Verilog 计数器来说明具体的操作过程。

🔑 仿真一个 Verilog 计数器程序

1. 继续使用“Book/Chapter3/Examples3-12”中的例子加以说明。启动 ModelSim 后,改变工作目录为“Book/Chapter3/Examples3-12”。
2. 在 ModelSim 的命令控制台使用以下命令编译源文件并运行仿真。

```
vlib work
vlog counter.v tcounter.v
vsim test_counter
vsim 1>vcd dumptports -file counter.vcd /test_counter/dut/*
vsim 2>run
vsim 3>quit -f
```

3. 以上的操作执行完之后,可以在 vsim 命令中使用 `-vcdstim` 参数打开 VCD 文件

重新运行仿真，可以使用如下的命令：

```
vsim -vcdstim counter.vcd counter  
vsim 1 > add wave /*  
vsim 2 > run 200
```

3.7 小结

本章主要介绍了 ModelSim 仿真工具的使用，详细说明了仿真工具的各个窗口用途以及使用方法。通过本章学习读者可以使用 ModelSim 仿真工具完成各种 HDL 语言的仿真。ModelSim 是一个独立的仿真工具，它在工作的时候并不需要其他软件的协助，在 Xilinx 公司的 ISE 集成开发环境中给 ModelSim 仿真软件预留了接口。本章在详细说明 ModelSim 工作过程以及使用方法的同时兼顾了这个软件在 ISE 集成环境中的使用方法。

3.8 问题与思考

1. 为什么在进行仿真以前需要对厂家的仿真库进行预编译？在对库文件进行命名时，命名原则由什么决定？
2. 混合仿真时需要注意哪些事项，如果没有注意到这些事项，对仿真会产生哪些影响？
3. ModelSim 中各个窗口的作用是什么？除了波形窗口外，其他窗口对程序的调试有什么帮助？
4. 波形比较的过程是什么？波形比较共有几种类型？可以使用哪些窗口查看波形比较的结果？
5. 什么是时序标注文件？它在仿真过程的地位是什么？怎样才是真正的时序仿真？

第4章 ISE 中集成的综合工具

综合是 FPGA/CPLD 设计流程中的重要环节，综合结果的优劣直接影响布局布线结果的最终效能。好的综合器能够使设计占用芯片的物理面积最小，工作频率最快，这也是评定综合器优劣的两个最重要指标。细心的读者会注意到，面积和速度这两个要求贯穿 FPGA/CPLD 设计的始终，它们是设计效果的终极评定标准。相比之下，满足时序、工作频率的要求更重要一些，当两者冲突时，一般采用速度优先的准则。

本章将介绍 ISE 中集成的典型综合工具的特点、使用方法及技巧。ISE 除了自带综合工具 XST 外，几乎和所有目前流行的综合工具都有接口。Synplify/Synplify Pro 是 Synplicity 公司出品的综合工具，以综合速度快、优化效果好而备受关注，成为目前业界最流行的高效综合工具之一。Synopsys 是最早的 EDA 工具厂商之一，其 FPGA Express 是 Xilinx 早期产品（Foundation 系列，低版本 ISE）惟一集成的综合工具，熟悉 Xilinx 早期产品的工程师对它一定非常熟悉。FPGA Express 的高级版本 FPGA Compiler II 是一个非常成熟的 FPGA、ASIC 设计平台。XST 是 Xilinx 自己设计的综合工具，因为只有 Xilinx 自己对其器件的底层设计与内部结构最为了解，所以使用 XST 综合某些设计，常常会得到意想不到的优良结果。更重要的是 XST 在安装 ISE 后就能直接使用，不需另外付费。本章的最后引入了全局时钟与第二全局时钟资源的概念，这两个概念与综合、约束、实现有着密切的联系，正确使用全局时钟与第二全局时钟资源，能够在一定程度上改善设计的性能。

本章主要内容如下：

- 新兴的高效综合工具——Synplify / Synplify Pro；
- Xilinx 最早的合作伙伴——FPGA Express；
- Xilinx 自带综合工具——XST；
- 全局时钟与第二全局时钟资源。

4.1 新兴的高效综合工具——Synplify/Synplify Pro

本节将重点介绍 ISE 中集成的 Synplicity 公司出品的高效综合工具 Synplify/Synplify Pro 的功能与特点。

4.1.1 Synplify/Synplify Pro 的功能与特点

Synplicity 公司出品的 Synplify/Synplify Pro 综合工具是目前备受推崇的 FPGA/CPLD 综合工具之一。Synplicity 综合工具的两个最显著的特点是：B.E.S.T 和 Timing Driven（时序驱动）综合引擎，这两项核心综合优化技术辅之于系列整体性优化策略和方法，使它们对设计

的综合无论在物理面积还是工作频率上都能达到较理想的效果。

4.1.1.1 Synplify/Synplify Pro 在 ISE 中的作用

Synplify/Synplify Pro 综合工具在 ISE 中主要完成综合过程。综合过程包括两个内容，一是对 HDL 源代码输入进行编译与逻辑层次上的优化，二是对编译结果进行逻辑映射与结构层次上的优化，最后生成逻辑网表。另外，Synplify/Synplify Pro 内嵌了 HDL 编辑器，也可完成 HDL 语言的源代码编辑与语法检错的功能，它在 ISE 设计流程中的作用如图 4-1 所示。

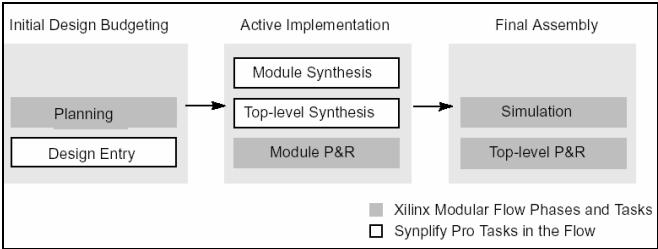


图4-1 Synplify/Synplify Pro 在 ISE 设计流程中的作用

4.1.1.2 Synplify/Synplify Pro 的异同

Synplify 是 Synplify Pro 的简装版。换句话说，Synplify 的功能选项是 Synplify Pro 的功能选项的子集。两者的综合原理与机制是完全相同的，但是 Synplify Pro 比 Synplify 功能强大得多，两者的功能比较如表 4-1 所示。

表 4-1 Synplify 与 Synplify Pro 功能对照

项目 功能	Synplify Pro 是否 有该项功能	Synplify 是否有该 项功能
行为提取综合技术——Behavior Extracting Synthesis Technology (B.E.S.T)	有	有
文本编辑框——Text editing Window	有	有
HDL 源代码分析器——HDL Analyst	有	可选功能
有限状态机编译器——FSM Compiler	有	有
综合约束条件编辑器——SCOPE	有	有
新用户界面——New user interface	有	无
命令执行窗——Tcl Window	有	无
综合报告观察窗——Log Watch Window	有	无
有限状态机观察器——FSM Viewer	有	无
有限状态机探测器——FSM Explorer	有	无
时序分析专家——Timing Analyst	有	无
多重实现手段——Multiple implementations	有	无
探针功能——Probe point extraction	有	无
文件互链切换功能——File Cross-Probing	有	无
时序优化功能——Pipelining	有	无

续表

项目 功能	Synplify Pro 是否 有该项功能	Synplify 是否有该 项功能
组合逻辑时序优化功能——Retiming	有	无
黑盒子 (Black Box) 延时定义描述语言——STAMP	有	无
Xilinx Virtex/Virtex-II 器件族验证设计流程——Verplex formal verification flow	有	无
多位置综合增量设计流程——MultiPoint flow	有	无
模块化设计流程——Modular flow	有	无

为了使读者能够更全面地了解和掌握 Synplify/Synplify Pro，下面以 Synplify Pro 为例介绍它们的使用技巧与功能。

4.1.1.3 Synplify Pro 特色工具与新增功能

目前 Synplify/Synplify Pro 的最新版本是 7.6 版，其特色工具与新增功能如下：

1. 支持的操作系统

Synplify / Synplify Pro 支持下列操作系统：

- Windows 98/Me/ NT 4.0/2000/ XP。
- Sun Solaris 系列。
- HP-UX 系列。
- Red Hat Linux 系列。

2. 支持的语言标准

Synplify/Synplify Pro 支持下列 HDL 语言输入方式。

- Verilog：包括 Verilog 95 (IEEE 1346)、Verilog 2000 和 Verilog 2001 等版本标准。
- VHDL：包括 VHDL 93 (即 IEEE 1076) 和 std_logic_1164、numeric_std、numeric_bit、std_logic_unsigned、std_logic_signed、std_logic_arith 等标准库。
- Verilog 和 VHDL 混合编程：Synplify/Synplify Pro 对 Verilog 和 VHDL 混合编程支持得很好。在一个工程中源程序可以既包含 Verilog 源代码又包含 VHDL 源代码。这个特性加强了模块的再用性，可帮助用户更好地继承以往的工程设计。

3. HDL 源代码编辑器

Synplify Pro 内嵌的 HDL 源代码编辑器与 ISE 的 HDL Editor 各有千秋。Synplify Pro 内嵌的 HDL 编辑器的语法检错功能十分强大，有语法错误检查 (Syntax Check) 和综合错误检查 (Synthesis Check) 两个层次。Synplify Pro 内嵌 HDL 编辑器结合 Cross-Probing 功能，能在源代码和寄存器传输级视图 (RTL View) 以及结构视图 (Technology View) 之间方便地互链切换，是源代码调试的有力工具。

4. 逻辑优化步骤——编译

编译是综合的第一步。Synplify Pro 将输入的源代码翻译成布尔 (Boolean) 表达式，并优化逻辑关系。优化的方法与效果和所选择的优化策略密切相关，通过优化可以消除部分冗

余设计，并对一些模块复用，使设计在面积和工作速度两方面都得到一定程度的改善。

5. 逻辑映射步骤——映射

逻辑映射是综合的第二步。Synplify Pro 将编译生成的逻辑关系映射为 FPGA/CPLD 的底层模块和硬件原语，如 Xilinx 的 IOBs、CLBs、Block RAM 和 Clk DLL 等。根据用户设置的综合优化条件，Synplify Pro 会根据优先级对电路的频率、面积进行全面优化。Synplify Pro 采用专用结构映射技术完成逻辑与结构映射，使综合效果更好。

6. 行为提取综合技术——B.E.S.T 算法

B.E.S.T (Behavior Extraction Synthesis Technology) 即行为级提取技术的缩写，是 Synplicity 综合工具高效综合的核心之一。过去综合技术主要是基于门级的，比如 Synnopsys 早期的综合工具，它们将用 HDL 代码描述的电路展开到门级，用与或非门逻辑和一些最基本的触发器、锁存器等基本单元表示，然后将这些门级网表适配到不同类型的 FPGA 底层结构上。这种门级综合方式在 ASIC 和早期 FPGA 设计中应用广泛，其特点是忠于原设计。但是由于 FPGA 的底层单元是依固定结构捆绑好的触发器和 LUT 组成的底层逻辑单元，以及一些固定结构的底层硬件原语，如 PLL、DLL、Block RAM 等。所以采用门级综合技术相当于将寄存器传输级 (RTL) 电路描述打平到门级，然后再组合到 FPGA 的底层硬件原语，有一个从高抽象层次 (RTL 级) 打平到低抽象层次 (门级)，然后再返回到一个较高一些的抽象层次 (FPGA 的硬件原语描述) 的过程。这无疑在综合效率上是不十分合理的。于是新的综合思路——行为级综合技术应运而生。行为级提取技术的核心是将 RTL 级 HDL 语言描述的电路结构并不再直接打平到门级去适配到 FPGA，而是将综合结构抽象成一个个行为级标准模块，而根据 FPGA 厂商 (Vendor) 提供的大量 FPGA 底层硬件原语和基本可编程逻辑单元的数据库，将行为级提取的模块直接适配到 FPGA 底层单元中去。行为级综合技术适配到 FPGA 的层次不再是门级，而是行为级，更高的综合层次大大节省了综合时间，并有效地提高了综合效率，使优化效果更加显著。

使用 B.E.S.T 技术，Synplify Pro 在分析源代码后，抽象出设计的模块化模型，用寄存器传输级视图 (RTL View) 图像化显示出来。这种高度抽象的视图由功能模块、RAM/ROM、有限状态机等模型组成。使用层次进出 (Push) 命令还可以进入模块内部观察设计逻辑结构。该算法能将寄存器传输级视图模块映射为结构视图 (Technology View) 元器件，使工程师对经编译、逻辑映射后生成的综合网表有更深刻的认识，大大方便了对设计的理解和调试。

该技术结合 CrossProbing 互链切换功能与视图的打平 (flattening)、过滤 (filtering) 等命令，将源代码、RTL 视图、结构视图、综合结果报告 (log 文件) 和有限状态机等有机地结合起来，帮助设计者更有效地分析综合的结果与关键路径等信息。

7. 整体性的综合策略

Synplify Pro 的综合策略非常重视在整体上对设计的分析与优化。这种综合策略能在更大程度上对设计进行优化，使设计的映射结果占用面积最小。

8. 快速编译与类推

Synplify Pro 的综合速度非常快，即使是百万门的大规模 FPGA 的综合，也能在相对较

短的时间内完成。而且 Synplify Pro 7.0 以上版本对综合引擎又有了较大的改善，对 RAM 的编译与类推速度更快。

9. SCOPE 集成化综合约束器

SCOPE (Synthesis Constraints Optimization Environment) 采用图形化集成界面，使用户可以方便、全面、有效地对设计进行综合约束。新版 SCOPE 的功能又有一定的增强，如支持 Multipoint Synthesis Flow (多位置综合增量设计流程) 等。

10. DST 直接综合技术

DST (Direct Synthesis Technology) 技术是根据 FPGA/CPLD 器件制造商的芯片内部硬件结构，直接将源代码优化、映射为该器件的底层模块和硬件原语。这样一方面减少了综合时间，提高了综合效果 (面积和速度)；另一方面也使 Synplicity 综合工具能以最快的速度支持芯片制造的新技术和新型芯片。

11. 自动 RAM 类推

Synplify Pro 对 RAM 的类推支持得很好，可以根据所选器件的型号将设计中的 RAM 自动类推为该 FPGA/CPLD 固有的 RAM 结构。Synplify Pro 的 RAM 类推技术支持的芯片生产商多、器件型号广、RAM 类型全。表 4-2 列出了 Synplify Pro 对不同器件族的 RAM 类推资源。该技术极大地方便了 RAM 资源的实例化和使用。在最新的 Synplify 版本中，支持 Xilinx 最新的 Virtex /II /Pro 和 Spartan3 FPGA 中的 Block RAM 的奇偶校验位，并可以将过大的 ROM 自动拆分到合适的 Block RAM 资源中去。

表 4-2 Synplify Pro RAM 资源类推表

器件商	器件族	RAM 类推原语 syn_ramstyle 的允许值	默认情况下使用的硬件资源
Altera	Flex10K, ACEX	Registers, block_ram	小容量 RAM 使用寄存器, 大容量 RAM 使用 EAB 块 RAM
	APEX, Stratix	Registers, block_ram	小容量 RAM 使用寄存器, 大容量 RAM 使用 ESB 块 RAM
Atmel	40K	registers	LPM 资源
Cypress	38K, 39K	registers	LPM 资源
QuickLogic	QuickRAM, QuickPCI	registers	专用 RAM 资源
Lattice	ORCA 2 / 3 / 4 EC/ECP/XP/SC	registers, block_ram	小容量可指定使用分布式 RAM, 大容量使用 EBR 块 RAM
Xilinx	XC4000, Spartan	select_ram, registers	小容量可指定使用分布式 RAM, 大容量使用块 RAM
	Virtex	select_ram, registers, block_ram, no_rw_check	小容量使用分布式 RAM, 大容量使用块 RAM

12. Cross-Probing 功能

Cross-Probing 是一种联系代码、视图、仿真、报告和关键路径的链接指针，使用该功能可以方便在代码、视图、仿真、报告和关键路径等之间互相切换。例如在有限状态机观察器中，双击某一状态，就可以自动切换到相应的源代码段落。在 RTL 视图或者

Technology 视图中双击某一模块，同样能自动切换到描述该模块的源代码段落。新版本的 Synplify Pro 甚至支持到第三方工具的 Cross-Probing 功能，可以在 ModelSim 等仿真工具以及一些原理图输入工具之间实现互链切换。Cross-Probing 功能大大方便了工程师对设计的理解与调试。

13. RTL View

RTL View 即寄存器传输级视图，它是 Synplify Pro 在对源代码编译后应用 B.E.S.T 技术再现的设计的寄存器传输级原理图。该图高度抽象为模块化结构，帮助用户理解源代码对应的具体电路结构，检验设计的正确性。

14. Technology View

Technology View 即结构视图，它是将设计用 FPGA/CPLD 的硬件原语（Primitive）和底层模块描述的门级结构原理图。它与 RTL 视图既相似又不同，结构视图对设计的描述更深刻，可以说结构视图是 RTL 视图向具体器件进行结构映射的结果。Technology 视图结合 RTL 视图能有效地帮助用户对整个综合过程有一个较深刻的认识，也能帮助用户理解设计电路，分析关键路径。

15. FSM Compiler

FSM Compiler 即有限状态机编译器，它是综合有限状态机的一个强大工具。Synplify Pro 的有限状态机编译器不但能自动从代码中分析出所使用的有限状态机，而且可以分析出有限状态机是否存在不可到达的状态或歧义状态转移等。综合时可以选用自然码、格雷码（Gray Code）或独热码（One-hot Code）等不同的编码方式进行状态机编码。

16. FSM Explorer

FSM Explorer 即有限状态机探索器，比有限状态机编译器（FSM Compiler）功能更加强大。它除了能自动发现状态集，并选用最佳的编码方式描述安全、高效的状态转移功能外，还能结合有限状态机观察器（FSM Viewer）方便地调试有限状态机的状态转移方式。FSM Explorer 的本质是基于 Timing Driven 的状态机优化技术，它是 Synplify Pro 比 Synplify 增强的重要功能选项之一。

17. FSM Viewer

FSM Viewer 即有限状态机观察器，它使用状态转移图描绘有限状态机的编译、优化结果，为状态机调试提供了图形化界面。

18. Resource Sharing

资源共享功能可以提高芯片的利用率，有助于减少设计所占用的物理面积。例如在 CASE 语句或者模块调用时出现互斥条件下的一些运算与操作，如加法器、乘法器、除法等，Synplify Pro 会在结构上共享同一个运算模块，从而达到节省资源的目的。

19. Retiming

Retiming 可以在不改变逻辑功能的前提下，自动用寄存器分割组合逻辑，在组合电路中插入寄存器平衡时延，提高芯片的工作频率。Synplify Pro 可以改善带有复位/置位端口的寄存器，I/O 单元的寄存器，电平敏感的锁存器，Xilinx 的基于查找表的移位寄存器（SRL）

和 RAM 等结构的时延。选择 Retiming 参数后，Synplify Pro 将对设计整体性使用 Retiming 技术。另外，Retiming 属于一种时域驱动（Timing Driven）优化技术。

20. Pipelining

Pipelining 与 Retiming 相似，都是改善时延、提高芯片工作频率的综合手段。两者的显著区别在于 Retiming 可以运用于整个设计，而 Pipelining 主要运用于一些算法路径，比如乘法器、加法器、ROM 的数据通路等。Pipelining 可以将一段较大的逻辑用寄存器分割成若干较小的逻辑，从而使该逻辑从输入到输出的时延减少。Pipelining 能自动优化乘法器、加法树、ROM 等结构，减少输出时延，提高工作频率。

21. Probe

Probe 也称探针功能，是 Synplify Pro 的又一特色功能。使用探针功能不需在 HDL 代码中做任何更改，就能方便地拉出任意信号到输出引脚。使用 Probe 将测试信号拉到测试引脚的最大好处是省掉了添加、删除测试信号时需要由子模块向父模块一层层向上传递的麻烦，保证了代码清晰、干净，简化了 FPGA/CPLD 的调试过程。

22. MultiPoint Synthesis Flow

MultiPoint Synthesis Flow 是 Synplify Pro 的重要新增功能。它使综合工具仅对修改过的模块或新增模块进行增量编译，而对未更改的部分保持原有的综合结果，配合增量式实现工具，可以很好地集成已有的设计成果，更好地支持模块化设计，缩短综合与实现的周期。

23. Verplex Conformal Flow

这是针对 Xilinx Virtex/Virtex-II 系列器件族而设计的验证工具，可以更好地评定综合结果，验证综合的正确性。

24. Timing Analyst

Timing Analyst（时序分析专家）是 Synplify Pro 7.2 新增的时序分析工具。它能对 Xilinx 和 Altera 器件进行点到点的路径时序分析，是结构化视图（Technology View）时序分析功能的有益补充。

25. Automatic Gate Clock Conversion

Automatic Gate Clock Conversion 即自动门控时钟转换，它是 Synplify Pro 7.3 及以后版本的新增功能。我们知道在 FPGA 设计中推荐采用同步时序设计方式，时钟的分频、倍频等推荐使用 DLL 或者 PLL 的 IP Core 完成，即使设计用户自己的时钟，也尽量采用寄存器设计同步时钟，一定要避免使用门控时钟。使用门控时钟会带来时钟的抖动、延迟等诸多时序问题，直接影响设计的可靠性和最高频率。而在 IC 设计中却恰恰相反，门控时钟是时钟树的一种常见设计手段，IC 设计中采用门控时钟结构可以在一定的程度上节约面积、减少功耗、提高速度，这是由于 IC 和 FPGA 的内部结构和设计方法不同决定的。所以当有些场合需要将 IC 设计转到 FPGA 平台上时，门控时钟的转换就变得异常重要，甚至有很多专业工具专门完成此项功能。Synplify Pro 7.3 及更高版本在一定程度上可以较好地完成门控时钟向同步时钟的转换，从而大大方便了对以往 IC 设计成果的继承。

26. Auto Constraints

Auto Constraints 即自动约束，是 Synplify Pro 7.5 以上版本新添的一项功能。通过 Auto Constraints 能为 Xilinx 和 Altera 的某些最新器件族，在默认情况下添加一系列提高综合优化性能的约束属性。需要声明的是这种自动约束并不能完全取代手动设置约束属性，对于相对复杂一些的设计，在 Synplify Pro 中通过 SCOPE 设置约束属性仍然是最有效的方法。Auto Constraints 在默认条件下，为用户提供了一种一键式约束方法。Auto Constraints 首先自动类推所有设计中的所有时钟，将每个时钟自设为一个时钟组，使用该参数可能会综合优化出在默认条件下的最佳时钟效果。同时选择 Auto Constraints 还会对 I/O、时序例外、DLL (Xilinx)、DCM (Xilinx) 和 PLL (Altera) 等产生一系列的默认优化参数。该参数对于 Xilinx 的 Spartan、Spartan XL、Spartan 2、Spartan 2E、Spartan 3、Virtex、Virtex E、Virtex II、VirtexII Pro 等系列 FPGA 有效。

4.1.2 Synplify Pro 的用户界面

Synplify Pro 的界面组织清晰明了，符合 Windows 界面操作习惯。功能按钮按照综合操作顺序组织，操作简便。Synplify Pro 的主界面如图 4-2 所示，由菜单栏、工具栏、状态显示栏、操作步骤按钮、重要综合优化参数选项设置、工程管理窗、工程文件显示区、消息显示窗和综合结果观察窗等部分组成。

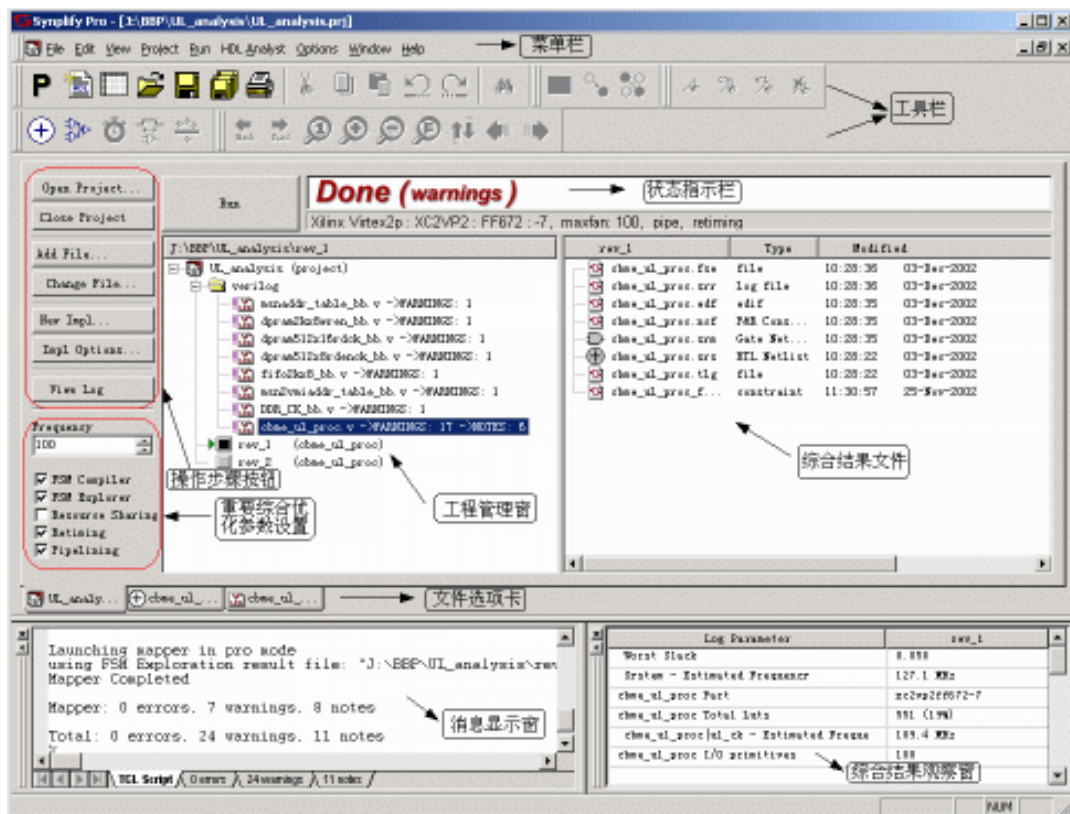


图4-2 Synplify Pro 的用户界面

1. 菜单栏

菜单栏包含了常用命令的下拉菜单。

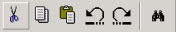
- 文件 (File) 菜单包含了通用文件处理命令。
- 编辑 (Edit) 菜单包含了通用编辑命令。
- 视图 (View) 菜单包含了 Synplify Pro 中各个窗口的显示和隐藏控制选项。
- 工程 (Project) 菜单包含了综合参数设置 (Implementation Options)、添加源文件 (Add Source File)、从工程中删除源文件 (Remove Files From Project)、替换文件 (Change File)、设置 VHDL 库 (Set VHDL Library) 和新建综合项目 (New Implementation) 等工程管理和参数设置命令。
- 运行 (Run) 菜单包含了 Synplify Pro 主要操作步骤的运行命令，如综合 (Synthesize)、重新综合所有项目 (Resynthesize All)、编译 (Compile Only)、有限状态机探测器 (FSM Explorer)、语法检查 (Syntax Check)、综合检查 (Synthesize Check)、运行工具命令语言脚本 (Run TCL Script)、运行所有综合项目 (Run All Implementations)、工作状态 (Job Status)、下一个错误定位 (Next Error) 和前一个错误定位 (Previous Error) 等命令。
- HDL 视图分析 (HDL Analyst) 是 Synplify Pro 中比较重要的一个菜单，包含了寄存器传输级 (RTL) 视图分析和结构级 (Technology) 视图分析的相关命令。这两种视图是 Synplify Pro 的特色工具，它们以图形方式表达设计编译后的寄存器传输级和综合后的结构级原理图。
- 选项 (Options) 菜单包含了 Synplify Pro 的参数设置和个性化界面设置等命令。如设置 VHDL 编译参数 (Configure VHDL Compiler)、设置 Verilog 编译参数 (Configure Verilog Compiler)、个性化界面配置 (Customize)、工程显示设置 (Project View Options)、编辑器设置 (Editor Options)、原理图设置 (Schematic Options) 和 Xilinx 命令选项等。Xilinx 命令选项包含了在 Synplify Pro 中直接启动 Xilinx 布局规划器 (Floorplanner) 和工程管理器 (ISE Project Navigator) 的命令。
- 窗口 (Window) 菜单集中了窗口调整命令，如层叠显示 (Cascade)、水平排列显示 (Tile Horizontally)、垂直排列显示 (Tile Vertically)、排列图标 (Arrange Icons) 和全部关闭 (Close All) 等。
- 帮助 (Help) 菜单集合了 Synplify Pro 在线帮助、使用演示和 license 管理等命令与信息。

2. 工具栏

工具栏集合了常用命令的快捷按钮，几乎所有综合操作都可以通过工具栏中的快捷按钮来完成。工具栏按照功能划分为下面几个项目：

- 文件工具栏  和菜单栏中文件菜单功能相同。其中按钮  的功能是启动工程向导 (Project Wizard)，完成所有工程和工作区的相关管理。按钮  的功能是自动启动 Synplify Pro 内嵌的 HDL 源代码编辑器。按钮  的功能是启动 SCOPE (Synthesis Constraints Optimization Environment)，采用图形

化界面，帮助用户方便、快捷地设计综合过程约束文件。

- 编辑工具栏  为菜单栏的编辑菜单中命令对应的快捷按钮。
- 有限状态机工具栏包含了常用 FSM 设计命令的快捷按钮。按钮  (Toggle FSM) 的作用是在状态与转移条件间切换显示。按钮  (Unfilter FSM) 的作用是重新显示 FSM 状态转移图。按钮  (Filter by outputs) 的作用是仅仅显示所选状态的输出与输出相关状态，屏蔽显示状态图其他部分。
- 书签工具栏  集合了 HDL 编辑器中设置书签的相关选项按钮，依次为设置/取消书签 (Toggle Bookmark)、到下一个书签处 (Next Bookmark)、到前一个书签处 (Previous Bookmark) 和清除所有书签 (Clear All) 等按钮。
- 寄存器传输级和结构级视图工具栏比较重要，它能帮助用户更好的使用 Synplify Pro 的特色图形化分析工具——寄存器传输级视图和结构视图，分析设计的综合结果。按钮  (RTL View) 的功能是启动 RTL 级视图，该视图是 Synplify Pro 采用 B.E.S.T 技术再现的设计的寄存器传输级原理图，帮助用户深刻理解设计的源代码。按钮  (Technology View) 的功能是启动结构视图，结构视图是综合优化后的底层视图，视图模块为 FPGA/CPLD 的硬件原语，帮助用户理解电路综合结果，分析关键路径。按钮  (Critical Path) 的功能是显示设计的关键路径，关键路径是最影响工作速度或时序约束的路径。按钮  (Timing Analyst) 的功能是启动 Timing Analyst，Timing Analyst 是 Synplify Pro 7.2 及其后续版本新增的时序分析工具，用于分析点到点之间的路径与时延。按钮  (Filter on Selected Gates) 的功能是将所选结构从繁杂的设计图中滤出，在一张新的结构图上显示，增加了视图的可读性。



RTL View 配合 Technology View 使用，从不同层次观察设计抽象出的原理图，不但能使用户更深刻地分析设计，确定设计的关键路径，还能帮助用户更好地理解综合过程的每个操作步骤。

- 视图操作工具栏  集合了使用寄存器传输级视图和结构级视图时一些常用视图调整命令的快捷按钮。

3. 状态显示栏

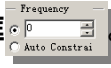
状态显示栏显示综合器当前的工作状态。

4. 操作步骤按钮

操作步骤按钮依照综合的实际操作顺序组织，体现了 Synplify Pro 综合流程，依次使用操作步骤按钮，就可以完成综合的全过程。

5. 重要综合优化参数选项设置

重要综合优化参数选项设置集中了综合优化过程中最重要的一些参数，这些参数的设置将直接决定综合的目标与结果。这些参数包括综合频率设置、FSM Compiler、FSM Explorer、Resource Sharing 和 Retiming、Pipelining 等。最新的 Synplify Pro 7.5 及以上版本中，在频率设置参数栏后新添了一个 Auto

Constraints (自动约束) 选择框 。通过这个这个选项，可以为 Xilinx、Altera 最新器件添加默认条件下的综合优化约束属性，以达到最佳约束效果。



这些重要参数在单击 Impl Options... 按钮，启动综合参数设置对话框后也可以设置，而且效果完全一致，只是在外部设置比较方便而已。

6. 工程管理窗

工程管理窗显示了工程结构和资源文件等信息。

7. 工程文件显示区

工程文件显示区用于显示工程输入文件和综合结果文件的内容。可通过已打开文件选项标签切换所显示的文件。

8. 消息显示窗

消息显示窗显示综合过程 TCL (Tool Command Language) 语言描述和工程信息，如警告 (Warning)、错误 (Error) 和注解 (Notes) 等信息。

9. 综合结果观察窗

综合结果观察窗是为了方便地观察综合结果的主要参数而设计，它能显示综合结果的时钟频率、I/O 引脚、寄存器资源、查找表 (LUT) 资源和所选器件等重要信息。

4.1.3 Synplify Pro 综合流程

在 ISE 中使用 Synplify/Synplify Pro 综合工具的基本方法有两种：第一种方法是在设置工程属性时选择 Synplify/Synplify Pro 设计流程，ISE 自动调用 Synplify/Synplify Pro 综合工具完成设计的综合过程，如图 4-3 所示；第二种方法是单独启动 Synplify/Synplify Pro 来完成综合过程，输出符合 Xilinx 格式要求的 EDIF 网表，在设置工程属性时选择 EDIF 设计流程，ISE 仅完成对网表的翻译、映射和布局布线等操作。

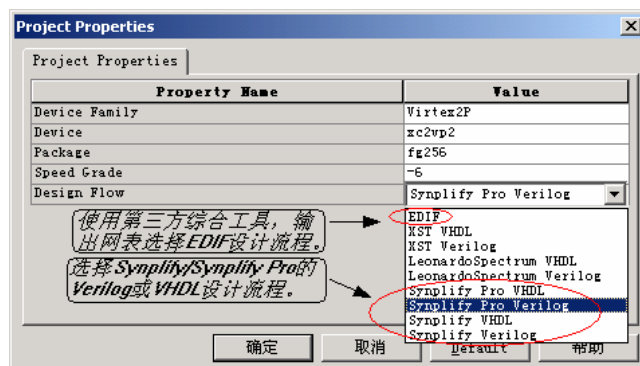


图4-3 选择 Synplify/Synplify Pro 设计流程

第二种使用方法更加灵活，可以充分发挥 Synplify/Synplify Pro 的强大功能，也是本书

介绍的重点。使用 Synplify Pro 进行综合的基本步骤可以细化为：创建工程、源代码检错、RTL View 观察、使用 SCOPE 设计综合约束条件、设置综合优化参数、综合和分析综合结果等 7 大步骤。

下面结合一个 ALU（算术逻辑单元）的简单例子介绍 Synplify Pro 综合流程。此例在本书配套光盘中有 Verilog 和 VHDL 两种类型的源代码，读者可以根据个人习惯选择不同语言类型的设计输入。参考示例详见本书附带光盘的“Example-4-1”目录。

4.1.3.1 创建工程

在 Synplify Pro 中工程创建和管理的方法有两种，一是在工程向导（Project Wizard）指导下完成工程的建立与管理，另一种是使用工具栏快捷按钮直接创建并管理工程。下面分别介绍这两种方法。

⑧ 使用 Project Wizard 创建工程

1. 工程向导可以完成工程和工作区（Workspace）的新建和管理工作。单击 **P** 按钮，启动打开工程对话框，这个对话框包含了打开工程、新建工程、新建工作区和启动工程向导等命令的快捷按钮，如图 4-4 所示。

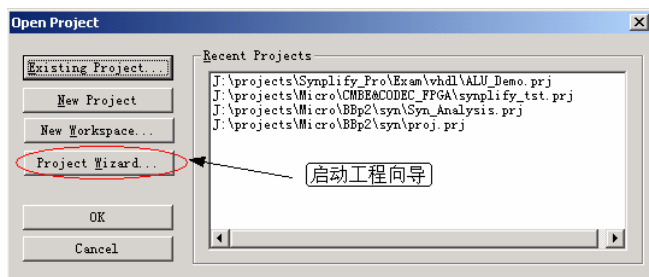


图4-4 打开工程对话框

2. 单击 **P** 按钮启动工程向导，进入新建工程对话框，如图 4-5 所示。在对话框中指定新建内容，以及工程名称、路径等基本信息。

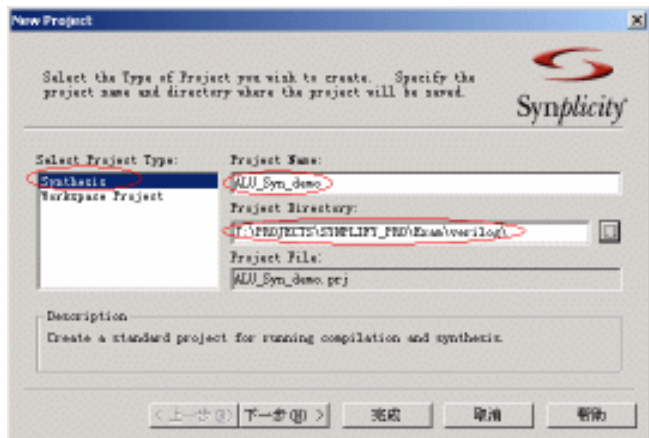


图4-5 新建工程

3. 单击 **下一步(N) >** 按钮添加工程文件，单击 **Add Files ...** 按钮，指定工程资源文件。要

注意在 Synplify Pro 中，一般默认最底下的文件为工程的顶层文件，所以在添加文件时请将顶层文件移至最下面，如图 4-6 所示。

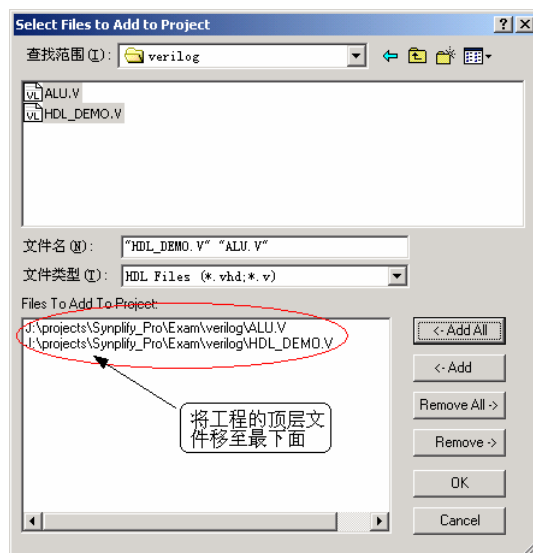


图4-6 添加工程文件

- 单击 **OK** 按钮确认添加文件顺序，单击 **下一步(N) >** 按钮选择器件类型，设置工程重要参数。这个界面与 **Impl Options...** 按钮所启动的综合参数设置对话框完全一致，可以单击 **Impl Options...** 按钮启动综合参数设置对话框，在这里对设置的参数进行修改，参数设置如图 4-7 所示。

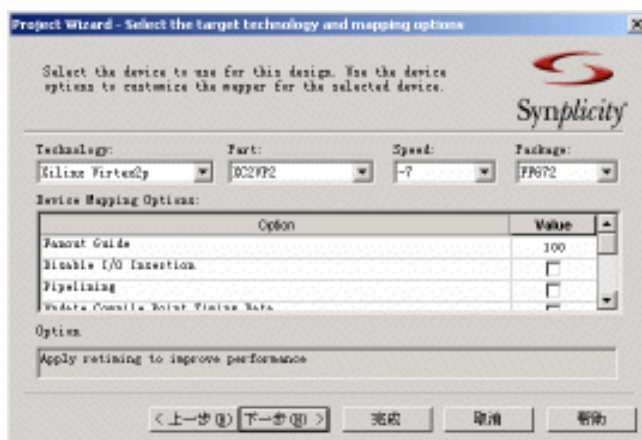


图4-7 选择器件类型，设置工程重要参数

- 单击 **下一步(N) >** 按钮，进入优化参数设置对话框，设置工程重要参数。其实这个界面的功能与上个界面有重复的地方。这个界面里的选项和图 4-2 所示的 Synplify Pro 主界面中的“重要优化参数设置选项”完全一致，其选项设置也可在 Synplify Pro 主界面中更改。按图 4-8 所示设置工程优化参数。

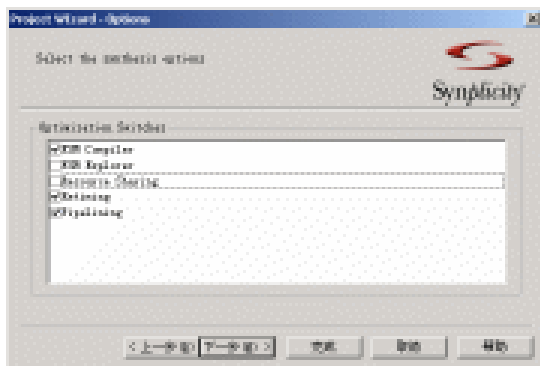


图4-8 设置工程优化参数

6. 单击 **下一步(N) >** 按钮设置工程频率，如图 4-9 所示。该处设置的频率是工程的全局频率，可以使用 SCOPE 详细设置综合过程约束条件。设计频率不易约束太高（过约束），也不易约束过低（必须满足设计要求），一般比设计需求频率高 5~10%。这里必须强调的一点是：通常只有非常简单的设计，或者单时钟设计才使用全局频率设置代替 SCOPE 中具体频率约束，对于复杂的设计不推荐笼统地使用全局频率设置，而应该在 SCOPE 中分别设置时钟、路径、I/O 等频率约束，否则可能会恶化优化结果。如果使用的软件版本是 Synplify Pro 7.5 以上版本，则由于本例所选器件族为 Virtex II Pro，所以此对话框中“Auto Constrain”选择栏有效，本例不选则“Auto Constrain”选项。在本对话框的最下面还有一个“Use clock period for unconstrained IO”选项，选择该选项则在最后综合优化时对于显化约束 I/O 属性的端口，遵从事先做好的显化约束属性；对于未做显化约束的 I/O，使用默认的时钟频率进行约束。不选则该选项则对所有的 I/O 都以默认的时钟频率进行约束。默认情况下是选择该选项，使手动做的显化约束优先级高于默认情况。

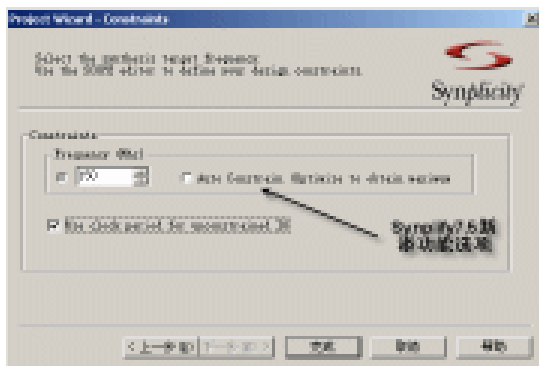


图4-9 设置工程默认频率

7. 单击 **下一步(N) >** 按钮，设置工程顶层文件和“Include”头文件路径，如果是 Verilog 工程，还可设置 Verilog 语法参数。
8. 单击 **完成** 按钮，确认设计。发现工程向导已经按照用户设置建好工程，将 HDL 源代码添加到工程中，并且 Synplify Pro 主界面下所有综合优化参数也同

用户设置一致。在信息观察窗可以看到工程向导所有操作的命令描述，如图 4-10 所示。

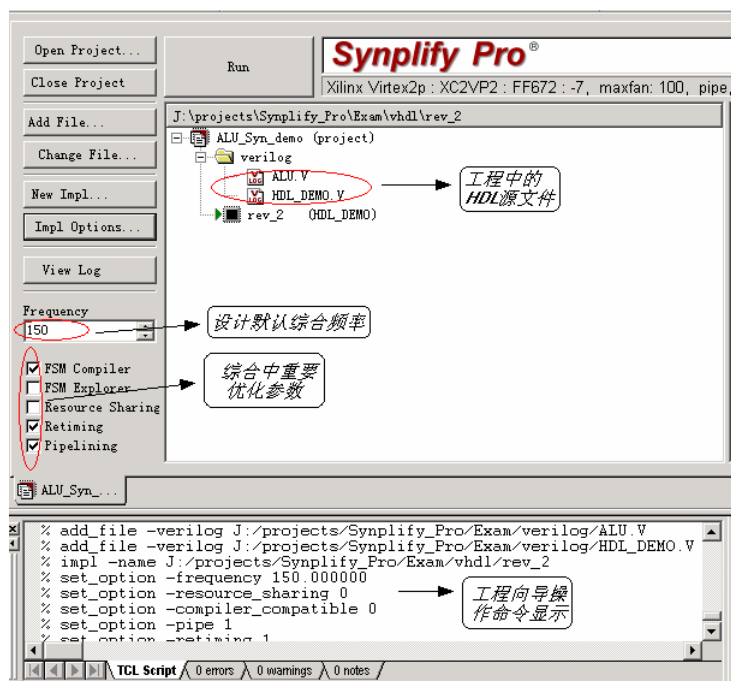


图4-10 工程向导新建工程结果

若不使用工程向导创建工程，则需要手工完成以上所有步骤。可以通过文件（File）和工程（Project）菜单中的工程相关命令完成，也可以通过打开工程、新建工程等快捷按钮完成，另外还可以在工程管理窗使用鼠标右键完成相应命令。在 Synplify Pro 中鼠标右键的使用方法非常丰富，为了加深印象，本例演示在工程管理窗利用鼠标右键创建工程的方法。

✎ 使用工具栏快捷按钮直接创建并管理工程

1. 在工程管理窗空白处，单击鼠标右键，弹出工程管理命令菜单，如图 4-11 所示。

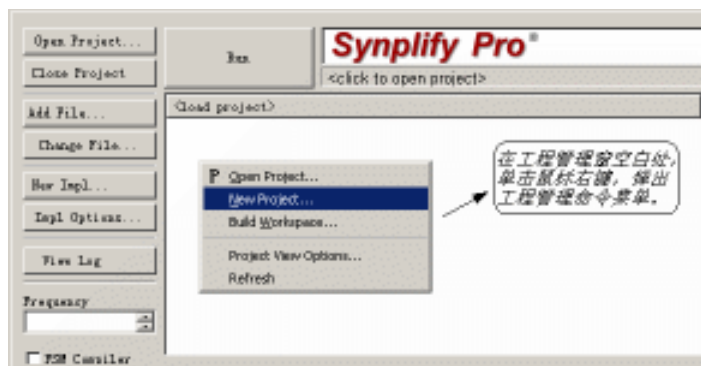


图4-11 工程管理命令菜单

2. 选择新建工程（New Project）选项，新建的工程在保存前默认工程名为

“proj”，用鼠标选中工程，单击鼠标右键，弹出命令菜单，选择【Add Source File】选项，如图 4-12 所示。

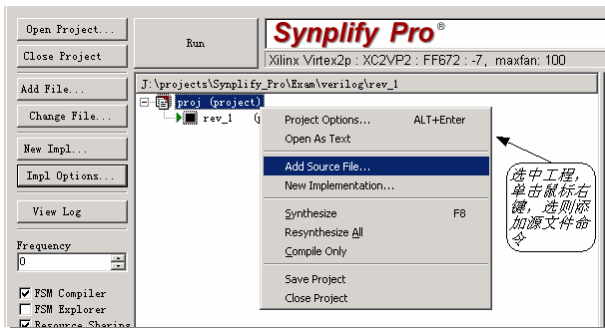


图4-12 添加源文件

3. 添加源文件方法与注意事项同利用工程向导创建工程所需注意事项相同。然后单击 **Impl Options...** 按钮启动综合参数设置对话框，设置目标器件型号、优化参数与综合频率，或者在 Synplify Pro 主界面下完成上述设置。最后单击  按钮，保存工程设置。这时会弹出【Save Project As】对话框，如图 4-13 所示，输入适当的工程名称后保存即可。

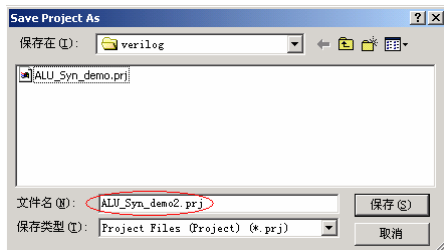


图4-13 存储工程设置



在 Synplify Pro 中，选中操作对象，单击鼠标右键弹出相应的命令菜单的操作很具代表性，几乎所有的命令都可以采用这种方式完成。

4.1.3.2 源代码错误检查

Synplify Pro 内嵌 HDL 编辑器的语法检错功能十分强大，有语法错误检查（Syntax Check）和综合错误检查（Synthesis Check）两个层次。



语法检查用于检查源代码中是否有语法错误；综合检查用于检查源代码是否是可综合的。Synplify Pro 这两个层次的错误检查工具组合是笔者在源代码调试阶段的首选。



源代码错误检查

在工程管理窗双击源代码图标，Synplify Pro 自动调用内嵌源代码编辑器编辑该源文件。

1. 双击 ALU 模块源文件，在源代码中将“case”改为错误的“caes”，保存文件。
2. 在菜单栏中选择【Run】/【Syntax Check】命令，则在信息显示窗显示错误警告，如图 4-14 所示。

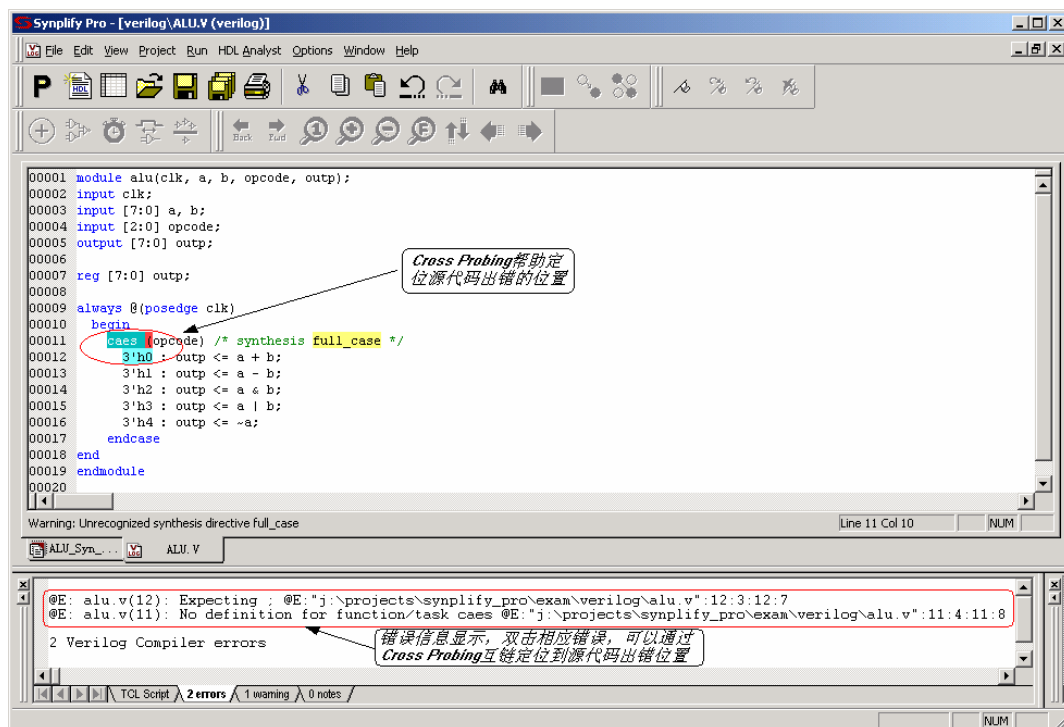


图4-14 语法检错

双击错误信息，可以自动切换到源代码错误位置，这个功能十分有用，叫做资源互链（Cross-Probing），可以帮助迅速定位错误。语法检错的快捷键组合为 **Shift+F7** 键。

3. 更正错误拼写“caes”为“case”，保存文件。
4. 再次语法检错，信息显示窗显示“Verilog Compiler: 0 errors, 0 warnings, 0 notes”，语法完全正确。

初学者经常会遇到这样的问题：语法检错完全正确，但是综合器仍然无法对设计进行综合，告警设计中有综合错误。Synplify Pro 综合检错工具就是分析代码的可综合性、定位综合错误的有力工具。

5. 将信号“outp”的端口声明由“output”改为“input”，进行语法检错，仍然没有任何语法错误。
6. 运行菜单命令【Run】/【Synthesis Check】，则综合检错功能会在“消息显示窗”报错，并指出“outp”应该为“wire”型变量。这是因为输入变量不能为寄存器型“reg”，也就不能进行非阻塞式赋值“<=”操作。双击错误信息，同样可以用 Cross-Probing 定位错误位置，如图 4-15 所示。

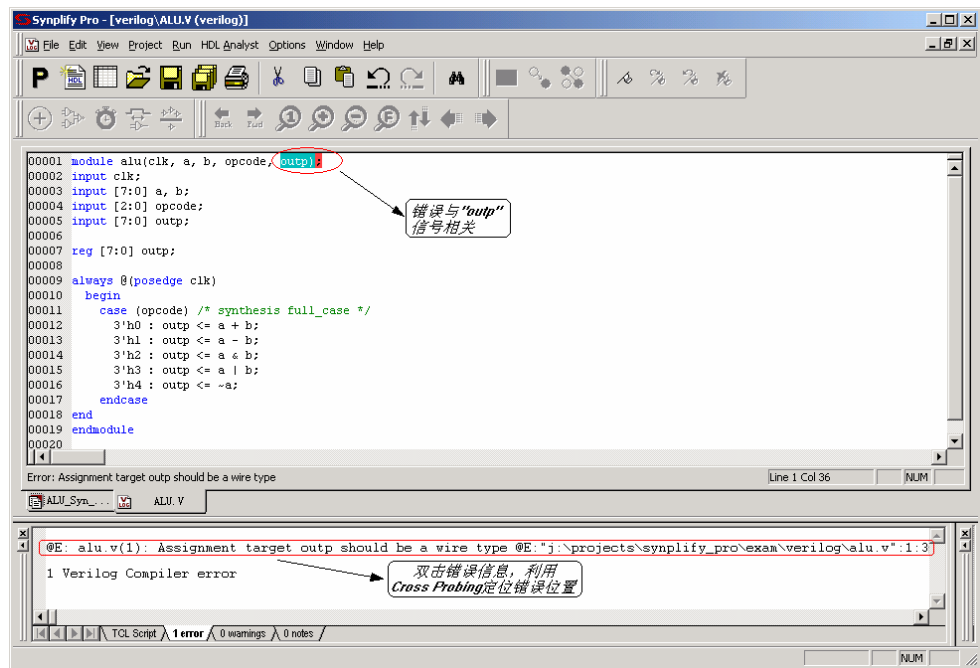


图4-15 综合检错

7. 改正错误，保存，再次进行综合检错，结果完全正确。
8. 用同样方法对“HDL_DEMO”源文件进行语法检错与综合检错。综合检错的快捷键组合为 **Shift+F8** 键。

4.1.3.3 使用 RTL View 观察编译结果

前面反复强调了 RTL 视图的重要性，下面就对 RTL 视图进行详细阐述。单击菜单栏中的【Run】/【Compile Only】命令，对设计进行编译。编译完成后发现在工程结果文件区有 3 个文件：扩展名为“srr”的文件是工程报告，包含了工程检错、编译、综合和时序等所有工程信息，在分析综合结果时要用到；扩展名为“tlg”的文件是工程组织结构信息文件；扩展名为“srs”、对应图标为  的文件就是 RTL 视图文件，如图 4-16 所示。

rev_2	Type	Modified
 HDL_DEMO.srr	log file	09:46:29 17-Dec-2002
 HDL_DEMO.srs	RTL Netlist	09:46:29 17-Dec-2002
 HDL_DEMO.tlg	file	09:46:29 17-Dec-2002

图4-16 预编译后生成的 RTL 视图

双击 RTL 视图文件，打开 RTL 视图。RTL 视图由两部分组成，左边区域显示模块、结构的分类目录，包括实例化（Instances）、端口（Ports）、网线（Nets）和时钟树（Clock Tree）等 4 部分内容。右边区域显示设计的 RTL 视图。

模块、结构的分类目录中各个部分的具体含义如下：

- 实例化：本工程所有调用的模块的实例名称和硬件原语（Primitives）依次罗列。
- 端口：模块各个层次的 I/O 端口组织关系。

- 网线：模块中所有连线名称。
- 时钟树：以树状结构图显示时钟依赖关系。

使用视图操作工具栏  可以对 RTL 视图完成缩放、进出结构层次等控制，协助用户更好地分析设计的寄存器传输级结构。

需要指出的是，RTL 视图也有强大的 Cross-Probing 互链切换功能：用户可以通过双击鼠标方便地在模块、结构分类目录，RTL 原理图，源代码之间进行互链切换。如图 4-17 所示，模块、结构的分类目录中的“alu”模块，RTL 原理图中的“alu”原理图符号，源代码中的“alu”模块实例化等 3 个高亮部分，用鼠标双击任何一部分都能唤出其他两部分。在 RTL 视图使用 Cross-Probing 互链切换功能，能更好地理解源代码，完成对设计的调试。

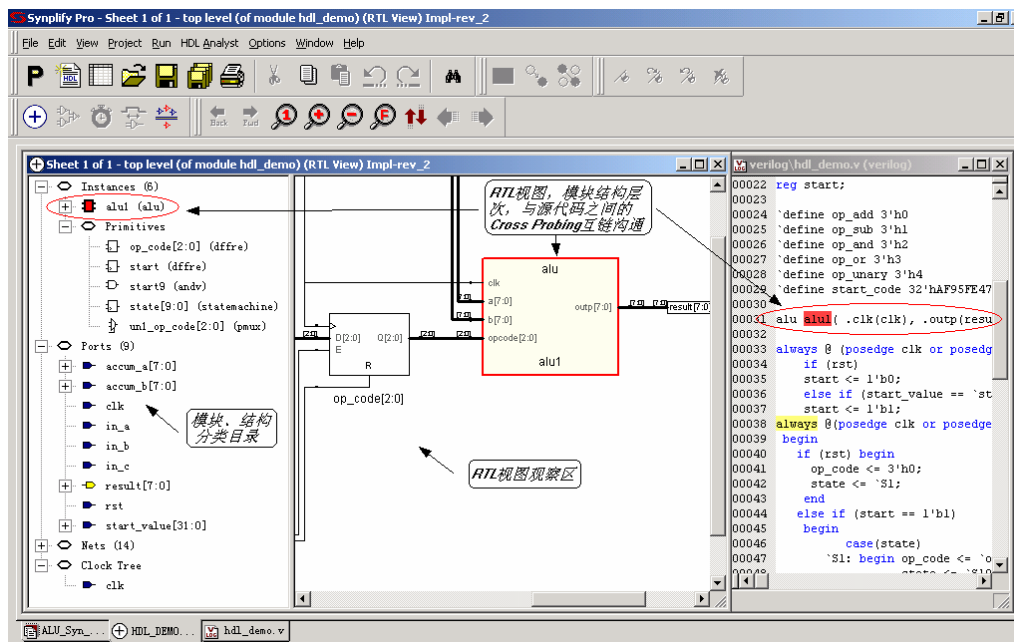


图4-17 RTL 视图与 Cross-Probing 互链切换

4.1.3.4 使用 SCOPE 设计综合约束条件

综合约束条件非常重要，它影响设计的综合结果并决定所综合出的硬件结构。Synplify Pro 的综合约束文件是 SDC 文件，SCOPE (Synthesis Constraints Optimization Environment, 综合约束优化环境) 使用图形界面辅助用户设计该综合约束文件。SCOPE 在 Synplify Pro 中地位非常重要，甚至可以说用好 SCOPE 是控制 Synplify Pro 达到预期优化效果的有利保障，掌握 SCOPE 是用好 Synplify Pro 的基本要求。新版的 SCOPE 对模块位置的约束有很大程度的增强，支持多位置综合增量设计流程 (Multipoint Synthesis Flow)。

单击  按钮，启动 SCOPE，设计综合约束文件。首先弹出的是新建 SCOPE 文件对话框，如图 4-18 所示。该对话框包括两个选项卡：一是初始约束内容，二是约束文件类型。初始约束内容选项卡主要确定时钟和 I/O 的约束是否需要 SCOPE 做自动初始化。设计在编译 (Compile) 后，SCOPE 可以自动提取出设计中应用的时钟和 I/O 引脚信息，在约束时非常方便，所以强烈建议两个选项都选。约束文件类型选项卡是针对 Multipoint Synthesis Flow

多位置设计流程新增加的功能，可以选择约束是相对于顶层还是某一模块进行。如果对整个设计进行约束，需要选择对顶层约束（Top Level）。

单击 **确定** 按钮，进入 SCOPE 主界面。SCOPE 主界面主要由时钟（Clocks）、时钟延迟（Clock to Clock）、输入输出（Inputs/Outputs）、寄存器（Registers）、多周期路径（Multi-Cycle Paths）、弱约束路径（False Paths）、路径最大延迟（Max_Delay Path）、约束属性（Attributes）、多位置编译（Compile Points）和其他（Other）等 10 个约束选项卡组成。其中 Clock to Clock 和 Max_Delay Path 两个约束页面是 Synplify Pro 7.3 及以后版本的新增功能。时间约束的基本含义如图 4-19 所示。



图 4-18 新建 SCOPE 文件

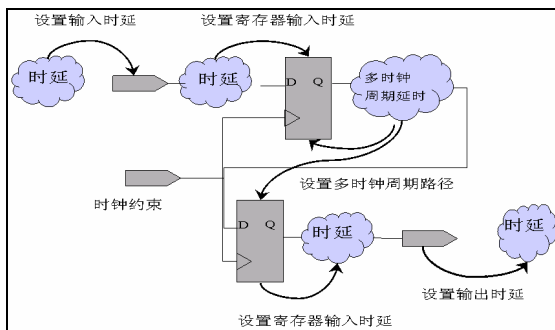


图 4-19 时间约束示意图

每个选项卡的主要内容如下：

1. Clocks 约束选项卡 MS1

约束内容主要为时钟频率（Frequency）、周期（Period）、时钟分组（Clock Group）、占空比（Duty Cycle）、上升沿（Rise At）、下降沿（Fall At）、布线延迟（Route）和是否是其他设计的虚拟时钟（Virtual Clock）等。这个选项卡对应的约束属性命令为“define_clock”。该约束选项卡通常情况下属于必选设置项目。

时钟频率、周期、占空比等通用概念不再赘述，仅讲解 Synplify Pro 中比较特殊的时钟约束含义和注意事项：

- 为了得到较好的优化结果，请用户尽量在时钟约束中对设计中的所有时钟加以约束，即在“Clock”域中列出所有时钟。有时在预编译后，Synplify Pro 并不能自动提取出用户设计的所有时钟，此时可以手动在“Clock”域中键入时钟名称，并做相应的约束。如果该信号不是真正的时钟，或者被优化掉，此无效约束并不会对设计产生负面影响，而仅仅是一个空约束而已。
- 准确指定每个时钟的周期（Frequency 域）或频率（Period 域）。
- 时钟分组（Clock Group）这个约束属性在多个时钟有密切关联时才使用，比如从某个基时钟分频、倍频出其他时钟，则最好将它们分成一组，综合器在综合与优化过程中会认为同一组中的时钟是同步的，并自动分析时钟之间的 Cross-clock。而对于不相关的时钟一定要分别分配到不同的分组中去，此时综

合工具将认为这些时钟是异步的，从而默认定义这些时钟域之间路径为 false paths 约束路径。

- 如果 Synplify Pro 综合结果估算出的时钟频率和实际布局布线后的时钟最高频率相差甚远，可以在“Route”域中填入估算的布线时延，以符合实际布线情况。
- 虚拟时钟（Virtual Clock）一般选设计的主时钟，设计中没有约束的电路的延迟默认通过虚拟时钟来约束。
- 约束完时钟属性后，需要勾选【Enabled】选项方能起作用。
- Synplify Pro 一般会对扇出（Fanout）最多的时钟自动使用全局时钟资源，以达到更高时钟性能，较小的时钟抖动和延时。如果用户不希望对某个时钟自动使用全局时钟资源，可以在 Attribute 约束属性选项卡中，对应的“Object Type”约束类型下拉菜单中选择“clock”，在“Object”域中选择需要约束的时钟，在“Attribute”下拉菜单中选择“syn_noclockbuf”约束属性。当然在 SDC 约束文件或者 HDL 代码中手写指定“syn_noclockbuf”约束属性也可以达到同样的效果。关于在 SDC 约束文件和 HDL 代码手写指定约束属性的方法参见本章 4.1.4.5 节综合约束的补充说明。
- 本约束属性选项卡的约束内容也可以在 HDL 代码中直接使用“define_clock”约束属性完成，约束属性和相关参数的添加方法参见 synplify Pro 在线帮助的 Defining clock 的说明。

本例的时钟约束非常简单，如图 4-20 所示。

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock
1	☒	clk	100	10.000	default_clkgroup	2	2	0.000		☒
2	☒									☐
3	☒									☐
4	☒									☐

图4-20 SCOPE 中的时钟约束



除非对于非常简单或单时钟设计，否则不要使用全局频率域，否则会恶化综合优化结果。对于相对复杂一些的设计应该使用 SCOPE 显化的约束时钟、路径、I/O 等对象的频率和延时信息。

2. Clock to Clock 约束选项卡

Clock to Clock 约束选项卡是在 Synplify Pro 7.3 及其以上版本新添的一个约束选项。该约束属性用于约束和所指定的两个时钟沿相关的所有数据路径的延时。如果不指定两个时钟沿的延迟，则系统默认情况下用前面 Clock 约束选项卡的约束内容计算所有时钟之间的延迟；如果在本选项卡中显化地定义了两个时钟之间的延时，则此显化约束优先级较高，也就是说系统在分析这两个时钟域之间的数据路径时，会自动以 Clock to Clock 约束的 Delay 值为准。Clock to Clock 约束选项卡通常情况下不属于必选设置项目。这个选项卡对应的约束属性命令为“define_clock_delay”。Clock to Clock 选项卡主要有 3 个参数域：From Clock Edge、To Clock Edge 和 Delay(ns)，含义分别如下：

- From Clock Edge
该参数用于指定第一个参考时钟的参考沿。单击选项栏目，会弹出所有时钟沿

列表，选择相应的时钟沿即可。时钟名称后面有一个冒号，后面有“r”或“f”，分别表示时钟的上升沿和下降沿。

- To Clock Edge
该参数用于指定第二个参考时钟的参考沿。同理单击选项栏目，弹出所有时钟沿列表，选择相应的时钟上升沿（“r”）或下降沿（“f”）即可。
- Delay(ns)
该参数用以指定两个参考时钟选定沿之间的数据路径延迟。单位是 ns（纳秒）。如果在该参数下拉列表中选择“false”而不指定任何延时时间，则表示这两个时钟是异步的，这两个时钟域之间的数据路径是“false path”关系的弱约束路径。

例如一个设置中有如下时钟关系，“CLK2”与“CLK3”是完全异步的，“CLK4”和“CLK3”的上升沿相差 5ns，如图 4-21 所示，则可以在 Clock to Clock 选项页面做如图 4-22 所示的约束。其实定义了“CLK2”与“CLK3”是异步的，就相当于定义了这两个时钟域之间的数据路径是 false path，即 Q2 到 Q3 之间的路径是 false path。定义了“CLK4”和“CLK3”的上升沿相差 5ns，等于说明了由于“CLK4”和“CLK3”两个时钟域之间的路径必须要考虑因为 clock delay 而造成的 5ns 的 skew，这个概念与“CLK4”和“CLK3”时钟域之间的 Q1 到 D2 的数据路径有 5 ns 的 clock skew 的概念一致。

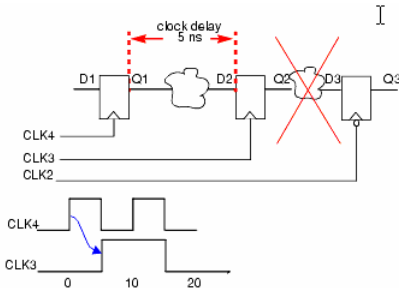


图4-21 Clock to Clock 约束时序示意

	Enabled	From Clock Edge	To Clock Edge	Delay (ns)	Co
1	☒	CLK4:r	CLK3:r	5	CLK4的上升沿到CLK3的上升沿之间的延迟为5ns
2	☒	CLK3:r	CLK2:r	false	
3	☒				CLK3与CLK2是异步时钟
4	☒				

图4-22 Clock to Clock 约束添加方法

3. Inputs/Outputs 约束选项卡

I/O 约束选项卡主要用于设置 Input Delay（输入延迟）与 Output Delay（输出延迟），分别对应的约束属性命令为“define_input_delay”和“define_output_delay”。需要说明的是，I/O 延迟属性约束不是一个必须约束属性，如果对芯片的输入和输出没有明确和特别的需求，可以不设置输入延迟与输出延迟。

所谓输入延迟指芯片外部寄存器到达芯片输入端口的路径延迟（ T_{Logic} ）加上外部寄存

器的固有输出延迟 (T_{CKO})，如图 4-23 所示。用公式表示如下：

$$T_{\text{Input Delay}} = T_{CKO} + T_{\text{Logic}}$$

其中 T_{CKO} 为寄存器的固有输出延迟，指寄存器从时钟到输出的固有延迟。

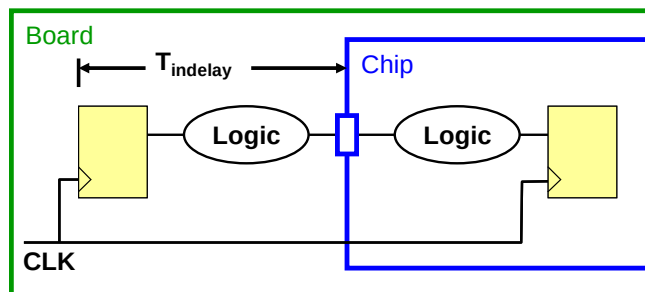


图4-23 Input Delay 输入延迟的含义

所谓输出延迟指从芯片输出管脚到外部寄存器的路径延迟 (T_{Logic}) 与该外部寄存器的固有建立时间 (T_{SU}) 之和，如图 4-24 所示。用公式表示如下：

$$T_{\text{Output Delay}} = T_{\text{SU}} + T_{\text{Logic}}$$

其中 T_{SU} 为寄存器的固有建立时间。

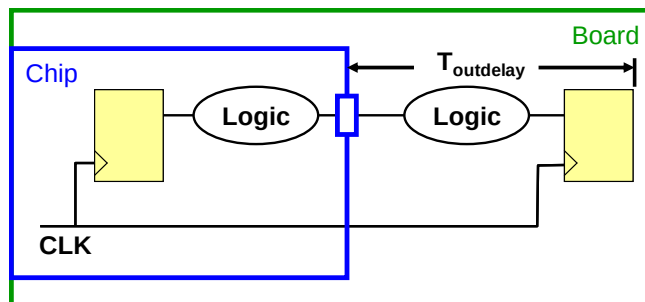


图4-24 Output Delay 输出延迟的含义

I/O 约束选项卡包括如下约束属性域：Port（端口）、Type（类型）、Clock Edge（时钟沿）、Value（数值）和 Route（布线延迟），含义分别如下：

- Port：用于指定 I/O 端口的名称。在最上面两项是 “input_default” 和 “output_default”，分别用以设置非显化指定的 I/O 管脚的默认状态下的输入、输出延迟。
- Type：用于指定输入延迟 (input_delay) 还是输出延迟 (output_delay)。
- Clock Edge：用于参考时钟的上升沿有效还是下降沿有效。
- Value：用于指定延时的数值，单位是 ns。
- Route：这是一个高级约束属性命令，用以指定因布线延时引起的输入延迟和输出延迟的变换量。

本例的 I/O 延时约束设置比较简单，如图 4-25 所示。

	Enabled	Port	Type	Clock Edge	Value (ns)	Route (ns)
1	☒	<input default>	input_delay	clk:r		5
2	☒	<output default>	output_delay	clk:r		5
3	☐	accus_a[7:0]	input_delay			
4	☐	accus_b[7:0]	input_delay			
5	☒	in_a	input_delay	clk:r	5	
6	☒	in_b	input_delay	clk:r	5	
7	☒	in_c	input_delay	clk:r	5	
8	☐	result[7:0]	output_delay			
9	☐	rst	input_delay			
10	☐	start_value[31:0]	input_delay			

图4-25 SCOPE 中 I/O 延时约束

4. Registers 约束选项卡

使用此约束属性可以让高级用户指定寄存器输入和输出路径的附加延时，该约束属性对应的约束属性命令为“define_reg_input_delay”和“define_reg _output_delay”。Registers 约束选项卡在通常情况下不属于必选设置项目。点击某个空白的寄存器域，SCOPE 根据编译提炼出的寄存器信息自动罗列出设计所使用的寄存器，本例设置如图 4-26 所示。

	Enabled	Register	Type	Route (ns)
1	☒	op_code[2:0]	reg_input_delay	4
2	☒	alul_outp[7:0]	reg_output_delay	4
3	☒			

图4-26 SCOPE 中的寄存器约束

5. Multi-Cycle Path 约束选项卡

多周期路径约束是一种慢速例外时间约束，指出某些寄存器或者路径需要多个时钟周期完成（如计数器的高比特位等），这些路径和寄存器变化较缓慢，如果也约束成在一个时钟周期内完成，就是过强的约束了。该约束属性对应的约束属性命令为“define_multicycle_path”。Multi-Cycle Path 约束选项卡在通常情况下属于必选设置项目。

Multi-Cycle Path 约束选项卡中主要约束域的含义如下：

- From：用以指定所有从所选寄存器出发的路径是多时钟周期路径。
- To：用以指定所有到达所选寄存器的路径是多时钟周期路径。
- Though：用以指定所有穿过所选寄存器的路径是多时钟周期路径。
- Cycles：用以指定多时钟周期路径的时钟周期数。

一般来说，Multi-Cycle Path 约束属性是必须设置的，合理设置 Multi-Cycle Path 约束会使综合引擎在做时序驱动（Timing Driven）综合优化过程中将优化的重点放在真正的关键路径上（Critical Paths），而不会因为对多时钟周期路径的过约束造成整体优化结果的恶化。同样的道理也可以适用到“False Paths”约束属性上。如图 4-27 所示为使用 Multi-Cycle Path 约束属性前后对同一宽度计数器的综合结果的结构视图，从图中可以看出使用 Multi-Cycle Path 约束属性后，其综合结果所占的逻辑资源明显减少。另外，使用 Multi-Cycle Path 约束属性还可以使综合优化时间大大缩短，避免综合工具错误的纠缠在多时钟周期路径上，综合结果报告中上报的最高频率也更加准确。

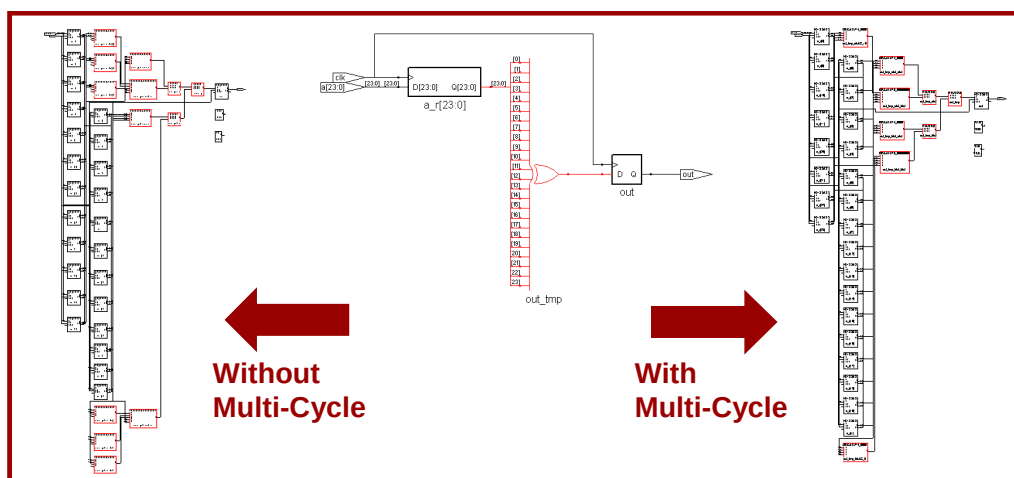


图4-27 SCOPE 中的寄存器约束

6. False Paths 约束选项卡

False Paths 约束也是一种慢速例外时间约束，其意义和 Multi-Cycle Path 非常相似，被指定有 False Paths 的路径将不受通用约束的限制，一般作为某些慢速路径和特殊路径的约束。该约束属性对应的约束属性命令为“define_false_path”。False Paths 约束选项卡在通常情况下属于必选设置项目。False Paths 约束选项卡中“From”、“To”、“Through”和“Cycles”约束域的含义与 Multi-Cycle Path 约束选项卡中各项的含义一致，不再赘述。

通过前面的介绍，细心的读者会发现通过设置 Clock to Clock 约束选项卡、Multi-Cycle Path 约束选项卡、False Paths 约束选项卡都能完成 False Paths 的约束。通常，False paths 约束的实现方法有如下几种：

- 一是直接在 False Paths 选项卡中指定 False Paths 路径，或者在 SDC 文件或 HDL 代码中手动添加“define_false_path”约束属性。这种方法是显化的定义 False Paths 的方法。
- 前面提过，在 Clock 约束选项卡中将两个时钟设置在不同的时钟组，则 Synplify Pro 会认为两个时钟异步，默认推论两个时钟域的数据路径是 False Path。这是一种默认的定义 False Paths 的方法。
- 或者在 Clock to Clock 将两个时钟的 Delay 选择为“false”，则 Synplify Pro 同样会认为两个时钟异步，默认推论两个时钟域的数据路径是 False Path。这也是一种默认的定义 False Paths 的方法。

这 3 种方法在设置 False Paths 约束属性的层面上是完全等效的（当然后两种还有其他的约束属性含义），例如对于图 4-21 所示时序关系，要想指明 Q2 到 D3 直接路径为 False Path，放松时序约束强度，下面的 3 种设置方法完全等效：

- 在 False Paths 选项卡中指定路径 Q2 到 D3 路径是“False Paths”。
- 在 Clock 约束选项卡中将时钟“CLK2”和“CLK3”分配到不同的时钟分组。
- 如图 4-22 所示，设置时钟“CLK3”到“CLK2”之间的延时属性为

“false”。

7. Max Delay 约束选项卡

这个约束属性用于设置点到点的路径的最大约束，其含义与 5.2.3 节所述在布局布线阶段的 Max Delay 约束属性一致。这个属性十分灵活，为用户方便地提供了约束单条路径延迟的手段。该约束属性对应的约束属性命令为“define_path_delay -max”。Max Delay 约束选项卡在通常情况下属于必选设置项目。

8. Attributes 约束选项卡

约束属性是指在 Synplify/Synplify Pro 语法中通过约束属性关键字指定的约束条件。约束属性多为“syn_”（代表通用综合约束属性）、“altera_”（代表 Altera 专用约束属性）或“xc_”（代表 Xilinx 专用约束属性）开头的关键字。如果在前面的约束属性选项卡中不能完成用户的约束需求，可以在 Synplify Pro 帮助文件中找到对应的约束命令，在 Attributes 约束选项卡中选择用户需要附加的约束属性。

在 SCOPE 中添加约束属性比较方便。SCOPE 中约束属性根据性质划分为全局（global）、端口（port）、输入（input_port）、输出（output_port）、时钟（clock）、视图（view）、实例化（instance）、寄存器（register）、有限状态机（fsm）和黑盒子（blackbox）等 10 大类。添加时，先选择添加类型（Object Type），再选择约束对象（Object），然后点击空白属性栏，在属性下拉列表中选择需要添加的属性，最后选中【Enabled】选项，使新添加的属性生效。图 4-28 所示为如何附加最大扇出和禁用全局时钟资源两种约束属性。

☒	<any>	op_code[2:0]	syn_maxfan	50	integer Overrides the default fanout
☐	<any>	clk	syn_noclockbuf	1	boolean Use normal input buffer
☐					
☐					

图4-28 在 SCOPE 中添加约束属性



在 SCOPE 的属性页面设置 Synplify Pro 综合属性十分方便，所有综合属性都可以通过 SCOPE 设置。而且在 SCOPE 中设置约束属性，仅仅改变了 Synplify Pro 综合约束文件（SDC 文件），并不需在代码上做任何改动，增加了代码的可维护性。

9. Compile Points 约束选项卡

多位置编译（Compile Points）与多位置综合增量设计流程（Multipoint Synthesis Flow）相关，是 Synplify Pro 7.2 中新增的功能。当某个模块被设为多位置编译模块并被锁定（locked）后，该模块在增量综合与优化其父模块过程中，与该模块相关的接口和边界不做任何优化，保持原有接口和层次。将该综合结果导入能够进行增量实现的布局布线器，则该模块的实现结果就可以保证与原有设计完全一致，从而继承了以往综合与实现的成果，提高了综合与实现的效率。如果不使用增量设计方法，本约束选项卡在通常情况下不属于必选设置项目。

前面介绍过 ISE 6 及以上版本增强了对增量设计（Incremental Design）的支持。增量设计主要涉及两个工具：增量综合工具和增量实现工具。使用 Synplify Pro 的“Compile Points”选项可以完成增量综合，将增量综合的逻辑网表导入到 ISE 集成的增量实现工具完成区域锁定与布局布线，这样就能继承以往的设计结果，并支持设计团队并行协作。关于增

量综合技术与增量实现详见本书第 8 章——增量设计与模块化设计技术。

图 4-29 演示了如何将模块“alu”设为多位置编译模块（仅仅演示，并没有勾选【Enabled】选项）。

Enabled	Module	Type
☐	alu	locked
☒		
☒		

图4-29 在 SCOPE 中添加多位置编译约束

10. Other 约束选项卡

在该选项卡中可以手动附加一些新增的约束属性和命令。本约束选项卡为高级用户提供了一种约束功能扩展，不是必须添加的约束属性。

设置完以上 10 个约束选项卡之后，就完成了 SCOPE 中的约束文件设计。单击  按钮保存约束文件，弹出保存对话框，输入约束文件名（扩展名为 sdc），单击  按钮确认。这时会弹出对话框询问是否将约束文件加入到工程，单击  按钮确认，将新建约束文件加入到当前工程。退出 SCOPE，可以看到新建的约束已经加入到工程中了，如图 4-30 所示。

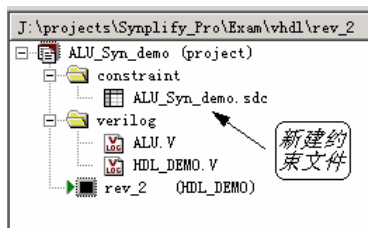


图4-30 新建的约束文件

4.1.3.5 设置综合优化参数

常用参数可以在主用户界面下直接输入，但是最全面的设置综合优化参数的方法是单击  按钮，启动综合参数设置对话框进行参数设置，如图 4-31 所示。

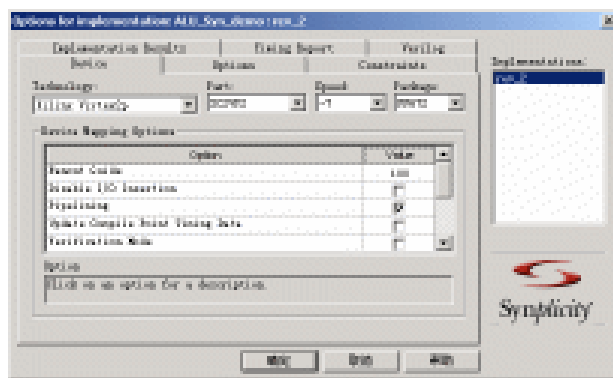


图4-31 综合优化参数设置

综合优化参数设置对话框主要由器件（Device）、优化参数（Options）、综合约束文件

(Constraints)、综合结果存储 (Implementation Results)、时序报告 (Timing Report) 和语言参数 (Verilog 或 VHDL) 等 6 个选项卡组成。下面分别解释每个选项卡的作用。

1. 器件选项卡

器件选项卡与利用工程向导创建工程时所弹出的参数设置对话框 (参见图 4-7) 内容完全一致。这个选项卡主要完成器件选型以及一些综合与优化的参数设置。可以设置的参数有：扇出数目 (Fanout Guide)、是否插入 I/O (Disable I/O Insertion)、快速通道时序优化 (Pipelining)、更新多位置综合时序数据 (Update Compile Point Timing Data)、模块验证 (Verification Mode)、模块化设计 (Modular Design) 和时序调整优化 (Retiming) 等。

2. 优化参数选项卡

优化参数选项卡与利用工程向导创建工程时所弹出的参数设置对话框 (参见图 4-8) 内容完全一致，其功能都是设置优化参数。需要选择的优化参数有：有限状态机编译器 (FSM Compiler)、有限状态机探索器 (FSM Explorer)、资源共享 (Resource Sharing)、Retiming 和 Pipelining 等。

3. 综合约束文件选项卡

综合约束文件选项卡主要设置全局频率和添加综合约束文件 (SDC 文件)。全局频率是整个设计的默认约束频率，所有未特殊指定的对象都默认通过全局频率约束。由于在 SCOPE 设计综合约束文件后，已经选择综合约束文件有效，所以这里可以直接看到刚才利用 SCOPE 设计的约束文件，如图 4-32 所示。

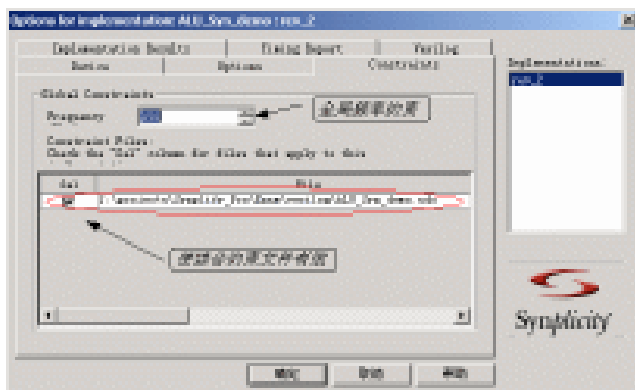


图4-32 约束选项卡设置

4. 综合结果存储选项卡

综合结果存储选项卡用于设置综合结果的输出文件名称与格式。Synplify Pro 综合结果的输出根据器件的供应商不同而不同。一般为两部分：综合网表输出 (EDIF 文件或 VQM 文件) 和综合约束传递文件。Xilinx 的综合网表输出文件为标准 EDIF 文件，综合约束传递文件为 NCF 文件。NCF 文件将 SDC 文件中的综合约束设置传递给 Xilinx 布线器，使综合约束条件在布局布线过程中起到应有的作用。在 Synplify Pro 中，编译 (Compile) 结果不包含任何时延、器件等具体约束信息，而“综合实现” (Implementation，与 Xilinx 的由翻译、映射、布局布线等 3 个步骤组成的实现过程虽然英文相同，但是含义不同) 结果则根据

器件和约束条件的不同而相差甚远。为了达到最佳效果，有时需要改变器件速度等级、时序约束和优化参数等条件，做多次“综合实现”的实验。这时可以在一个工程下面建立多个“综合实现”工程名，选项卡设置如图 4-33 所示。

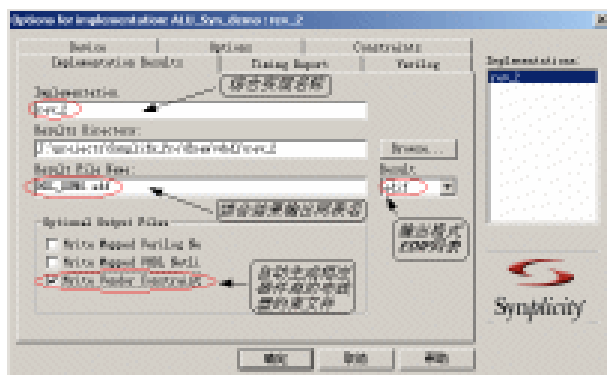


图4-33 综合结果存储设置

5. 时序报告选项卡

时序报告选项卡主要设置在综合报告（.srr 文件）中报告的关键路径和起点到终点之间路径时序的数目。根据设计的复杂程度和时序要求设置这两个参数，如图 4-34 所示。

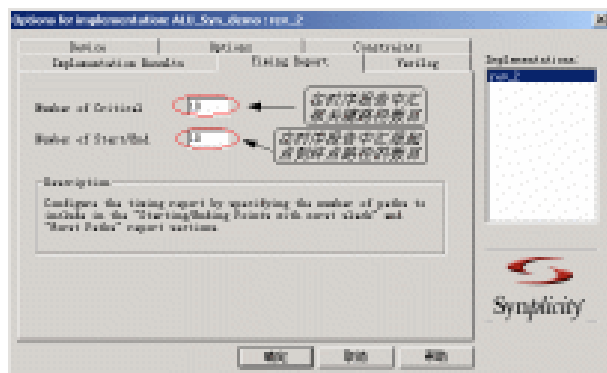


图4-34 时序报告设置

6. 语言参数选项卡

这个选项卡根据工程源文件类型不同而分为 Verilog 和 VHDL 两种类型，主要设置支持语言标准和库文件的引入路径等。

完成综合优化参数设置后，单击 **确定** 按钮保存参数设置。

4.1.3.6 综合

完成了综合时序约束文件设计和综合优化参数设置，进行综合就变得非常简单了。单击 Synplify Pro 主界面中的按钮 **Run**，综合器自动根据要求进行综合，这时可以看到状态显示栏依次显示编译“Compiling”，映射“Mapping”。当综合完成后，状态显示栏显示“Done”表示已完成综合过程。

4.1.3.7 分析综合结果

综合完成后发现综合结果显示窗口多了一些文件，如图 4-35 所示。前面已经介绍过扩

展名为 srs、tlg 和 srr 的文件，这里不再赘述。扩展名为 edif 的文件是综合器输出的网表文件，该网表是实现步骤的最重要输入，将该网表导入到器件商的布线器，就可以根据综合结果完成翻译、映射、布局布线等实现步骤。扩展名为 fse 的文件是有限状态机编码文件。扩展名为 ncf 的文件是综合器产生的 Xilinx 布局布线约束文件。扩展名为 srm 的文件是结构级视图文件。

rev_2	Type	Modified
HDL_DEMO.edf	edif	19:56:34 17-Dec-2002
HDL_DEMO.fse	file	19:56:34 17-Dec-2002
HDL_DEMO.ncf	P&R Constraints	19:56:34 17-Dec-2002
HDL_DEMO.srr	log file	19:56:34 17-Dec-2002
HDL_DEMO.srm	Gate Netlist	19:56:33 17-Dec-2002
HDL_DEMO.srs	RTL Netlist	09:46:29 17-Dec-2002
HDL_DEMO.tlg	file	09:46:29 17-Dec-2002

图4-35 综合完成后的结果文件



不同器件商的器件和不同器件型号的综合结果文件不同。

综合结果分析主要是利用结构视图、综合报告分析综合结果是否满足时序要求，分析综合结果的频率、面积等信息。

• 使用综合报告观察窗查看重要综合结果

综合报告观察窗（Log Watch Window）是 Synplify Pro 特有的一个小工具，它能从综合报告文件中提取出重要的信息显示在观察窗中，省去了查看冗长的报告文件的麻烦。特别是在同时打开几个工程，对比相互之间综合结果差异时尤为方便。综合报告观察窗的使用非常简单，单击空白的参数显示栏，弹出报告下拉列表，选择需要观察的项目即可，如图 4-36 所示。

Log Parameter	rev_2
clk - Estimated Frequency	57.1 MHz
clk - Requested Frequency	100.0 MHz
hdl_demo clk - Slack	-6.665
hdl_demo clk - Estimated Frequency	75.0 MHz
hdl_demo clk - Requested Frequency	150.0 MHz
hdl_demo clk - Slack	-6.665
hdl_demo Part	xc2vp2ff672-7
hdl_demo I/O primitives	60
hdl_demo I/O Register bits	3
<clear>	
Worst Slack	70 (2%)
clk - Estimated Frequency	62 (2%)
clk - Requested Frequency	
clk - Estimated Period	
clk - Requested Period	
clk - Slack	
hdl_demo clk - Estimated Frequency	

图4-36 综合报告观察窗

• 使用结构视图观察综合结果

结构视图（Technology View）是重要的综合结果分析工具。与 RTL 视图相比较，结构视图对设计的描述更深刻，是 RTL 视图根据器件底层结构进行结构级映射的结果。结构视图结合时序分析专家（Timing Analyst）能较为准确地分析综合结果的时序关系，帮助用户

发现设计的关键路径。

双击结构视图文件，打开结构视图。结构视图的基本界面结构和 RTL 视图完全一致，不再赘述。同样可使用视图操作工具栏  对视图进行缩放、进出结构层次等控制。结构视图也有强大的 Cross-Probing 互链切换功能：可在模块、结构的分类目录、结构视图原理图和源代码之间方便地互链切换，如图 4-37 所示。

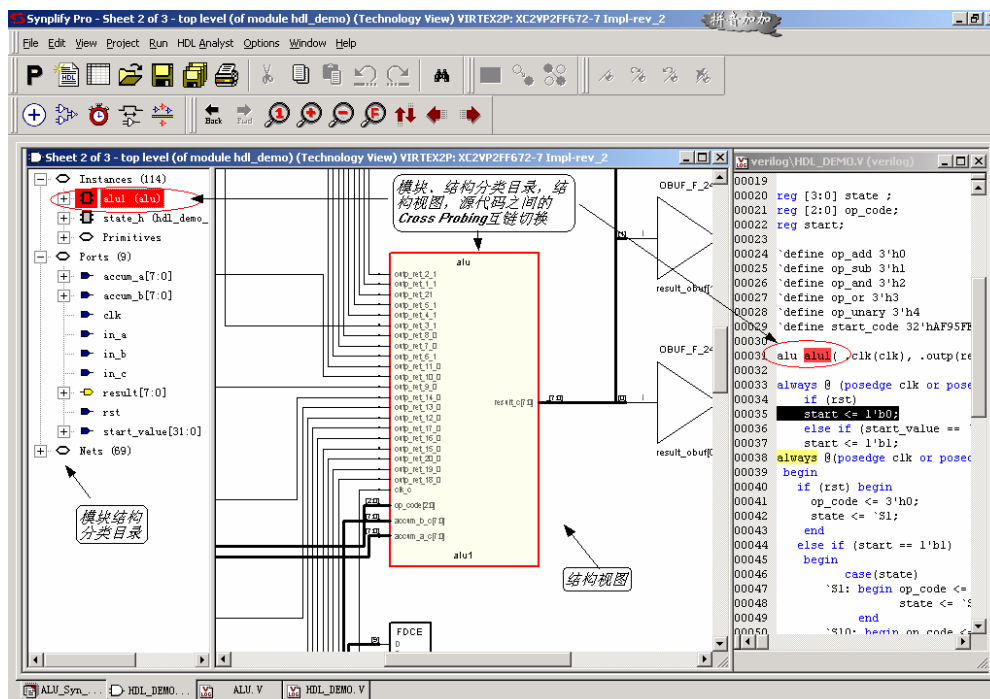


图4-37 在结构视图中进行 Cross-Probing 互链切换

• 阅读综合报告

Synplify Pro 的综合报告是 srr 文件，这个报告文件的内容十分丰富，从综合器基本信息、工程信息、检错信息、有限状态机编译信息、时序约束信息、I/O 信息、寄存器优化信息、组合电路时序优化信息、路径延迟信息到器件综合频率面积总结等，一应俱全。该文件是纯文本文件，可以在综合结果窗双击 srr 文件图标打开该文件。

• 检查时序约束条件是否满足，分析关键路径

检查设计是否满足时序要求可以从 3 方面入手：先从综合报告观察窗观察每个时钟和整体时序是否满足要求，如果“Slack”时间为负数，则不满足时序要求；然后打开综合结果 srr 文件，查找不满足时序要求的关键路径的列表信息；最后结合结构视图，分析关键路径。

本工程由于设置了过强的时序约束，所以不满足时序约束条件。

下面演示如何使用结构视图分析关键路径。

🔑 使用结构视图分析关键路径

1. 打开结构视图，在结构视图观察窗空白处单击鼠标右键，弹出如图 4-38 所示

的菜单，选择【Flatten Schematic】（打平原理图）选项。

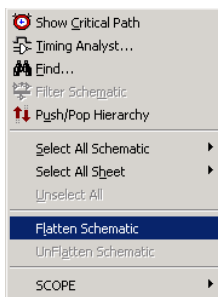


图4-38 选择【Flatten Schematic】选项

2. 单击按钮，过滤出关键路径，如图 4-39 所示。

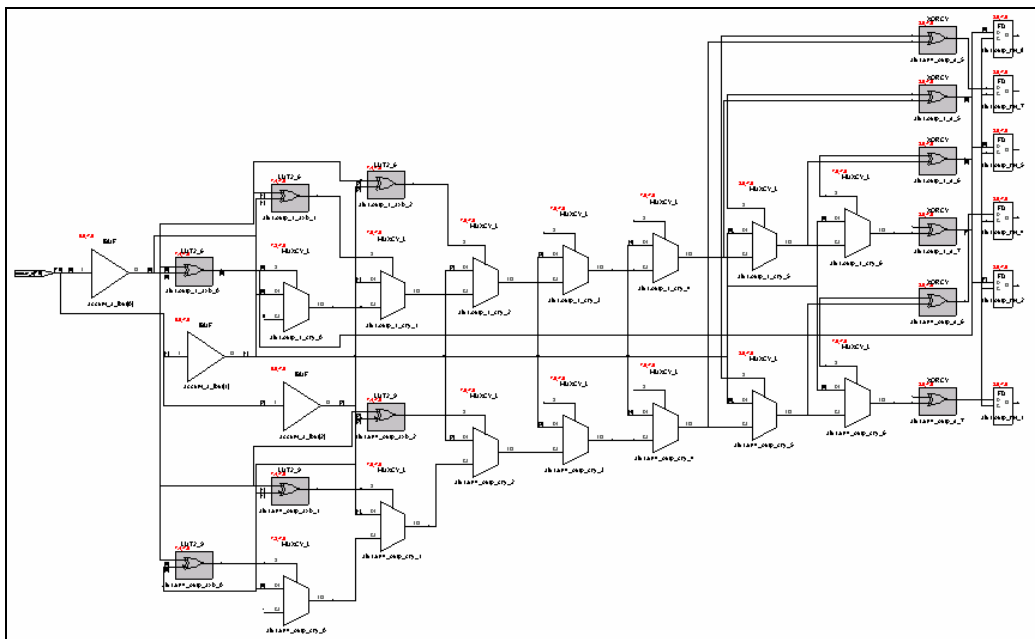


图4-39 Synplify Pro 自动过滤出的关键路径

在分析关键路径的过程中，合理运用过滤功能可以加强视图的可读性，简化问题。使用过滤功能首先要选择视图中某些对象（模块、网线、端口等），单击鼠标右键弹出命令菜单，使用丰富的命令，选择所需对象。常用的命令如下（参见图 4-40）：

- 【Select Net Driver】，选择某网线的驱动单元。
- 【Select Net Instances】，选择与某网线相连接的模块。
- 【Level】，层次递进。
- 【Expand】，扩展与所选对象相连接的模块。
- 【Select All Schematic】，选择原理图中所有对象。

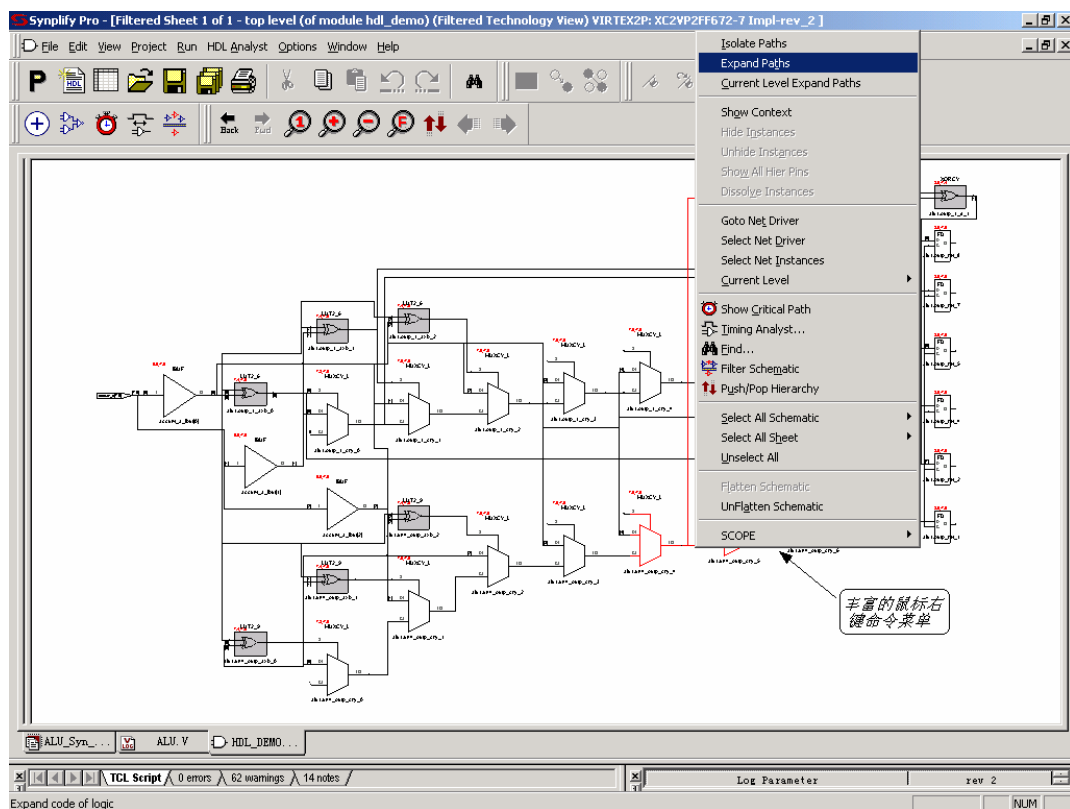


图4-40 使用鼠标右键命令，选择原理图中的对象

3. 选择对象后单击  按钮将对象滤出到另外一张图中，使用 Expand Path 命令可以扩展与滤出对象相关联的模块。

至此，完整的 Synplify Pro 综合流程已经完成，将综合结果——EDIF 网表和 NCF 约束文件导给 ISE，用 EDIF 设计流程进行实现步骤即可。

4.1.4 Synplify Pro 的其他综合技巧

上一节已经介绍了一些 Synplify Pro 综合技巧，但是由于上节的重点在于讲解 Synplify Pro 的综合流程，所举示例不能面面俱到，作为补充，本节将进一步深入讨论 Synplify Pro 的其他使用经验与技巧。

4.1.4.1 工程管理技巧

作为工程管理器使用技巧的补充，本小节简单介绍工程管理的其他应用：创建工作区 (Workspace) 和混合设计工程管理。

 使用工程向导建立工作区，参考示例详见本书附带光盘的“Example-4-1”目录

工作区是一种工程组合，通过工作区将相关工程组合起来，有助于在工程间进行分析对比。工作区的建立方法有两种，一种是直接建立，另一种是通过工程向导 (Project Wizard)

来建立。下面演示使用工程向导建立工作区的基本方法。

1. 单击 **P** 按钮，启动打开工程对话框，单击 **Project Wizard...** 按钮启动工程向导。选择新建工程类型为 Workspace，键入工程名和存储路径，如图 4-41 所示。

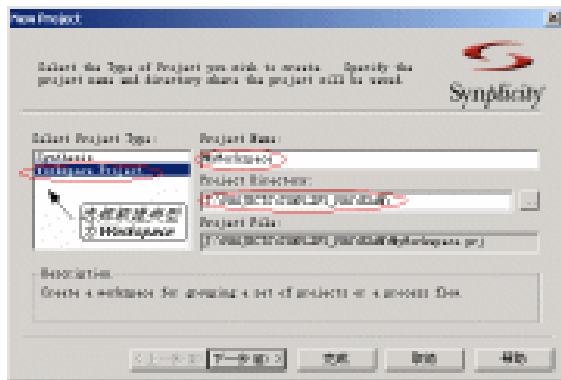


图4-41 使用工程向导新建 Workspace

2. 单击 **下一步(N) >** 按钮，在接下来的对话框中单击 **Add Files...** 按钮，指定需要添加的工程项目（prj 文件）。工程项目添加到 Workspace 后，单击 **完成** 按钮确认 Workspace 的创建，这时在 Synplify Pro 工程目录下发现工程向导已经按照设置创建了“ MyWorksapce ”工作区。在工作区中所有操作命令一般仅针对高亮显示的选中工程有效，而所有未高亮显示工程保持原有状态，如图 4-42 所示。

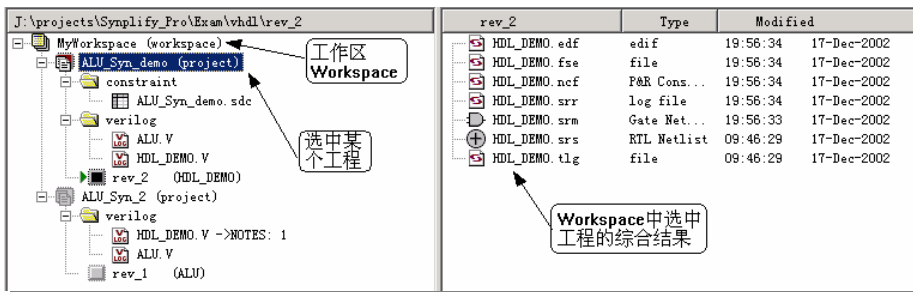


图4-42 新建好的 Workspace

有时因为需要继承一些以往的设计，造成资源文件类型不统一，既有 Verilog 源代码又有 VHDL 源代码。对混合资源设计的管理一般比较麻烦，但是在 Synplify Pro 中仍然简单易行。下面用一个 SDRAM 的例子，演示混合语言工程的管理方法。

🔑 混合语言工程管理，参考示例详见本书附带光盘的“ Example-4-1 ”目录

1. 打开 Synplify Pro，在用户本地硬盘新建目录，新建工程“ Mix_scr ”。
2. 将本书光盘“ Example-4-1 ”目录下的 6 个 verilog 文件（brst_cntr.v、define.v、ki_cntr.v、ref_cntr.v、sdrm.v、sys_int.v）和 5 个 VHDL 文件（cslt_cntr.vhd、mti_pkg.vhd、rcd_cntr.vhd、sdrm_t.vhd、sdrmc_state.vhd）都拷贝到新建目录中。再从光盘“ Example-4-1\Synplify_Pro\Mix\xilinx_lib ”目录

下拷贝“virtex.v”文件到新建目录中，这个文件是 VirtexE 器件族的库声明文件。打开该文件可以看到，里面是 VirtexE 器件的模块和端口声明，使用这个文件可以将 VirtexE 所用到的硬件原语和宏都综合成黑匣子。同理使用 VirtexE.vhd 也可以完成端口声明。如果选用是其他器件族的器件，如 Virtex2，就请拷贝 Virtex2.v 或者 Virtex2.vhd 文件。单击 **Add File...** 按钮添加上述的 12 个.v 和.vhd 的源代码文件，如图 4-43 所示。

- 单击 **OK** 按钮完成混合语言文件的添加，在 Synplify Pro 主界面可以看到源代码已经按照 VHDL 和 Verilog 分类组织在工程中，如图 4-44 所示。

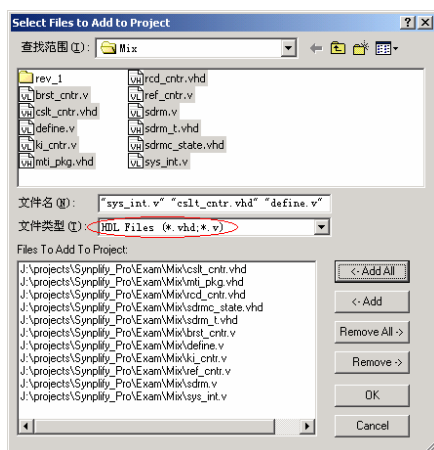


图 4-43 添加混合语言文件

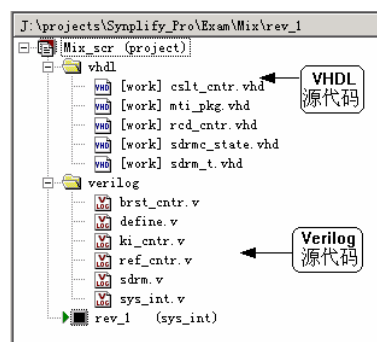


图 4-44 源代码按照 VHDL 和 Verilog 分类组织在工程中

- 在混合资源工程中必须明确指定工程的顶层模块名称。具体方法是单击 **Impl Options...** 按钮，启动综合参数设置对话框，如果工程的顶层文件是 Verilog 文件，就在【Verilog】选项卡的【Top Level】栏输入顶层模块名称；如果工程的顶层文件是 VHDL 文件，就在【VHDL】选项卡的【Top Level】栏输入顶层模块名称。本工程的顶层模块“sdrm”是用 Verilog 语言描述，所以在【Verilog】选项卡设置，本例不用指定库路径“Include Path Order”，单击 **确定** 按钮确认，如图 4-45 所示。

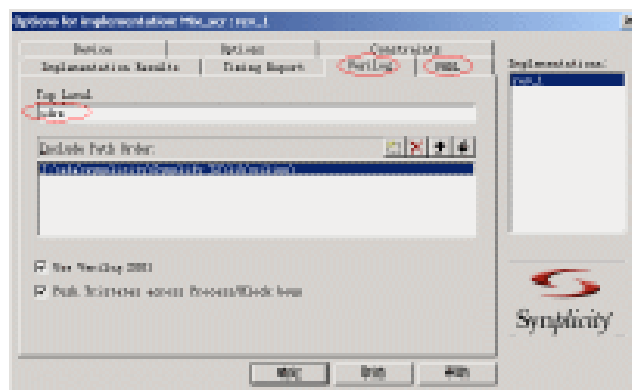


图4-45 【Verilog】选项卡的设置

- 5. 混合语言工程的其他设计方法与普通工程一致。请读者按照 4.1.3 小节叙述的步骤，自行完成混合语言工程的源代码检错，综合约束文件设计，综合优化参数设置，综合，分析综合结果等步骤。

4.1.4.2 有限状态机（FSM）的设计技巧

Synplify Pro 提供了 3 个有限状态机设计工具：FSM Compiler、FSM Explorer 和 FSM Viewer，灵活地使用这 3 个有限状态机工具分析、编译、优化 FSM，可使 FSM 的综合结果达到最优。下面逐一讨论它们的使用方法。

1. 有限状态机编译器（FSM Compiler）

一般的综合工具将 FSM 按照普通逻辑综合，而 Synplify Pro 与之不同。Synplify Pro 使用 FSM Compiler，先将 FSM 编译为类似状态转移图的连接图，然后对 FSM 重新编码、优化以达到更好的综合效果。

FSM Compiler 适应于有以下需求的场合：需要优化 FSM 设计，达到更好的综合效果；使用 FSM Viewer 调试状态机；使用 FSM Explorer 进一步优化有限状态机。

FSM Compiler 的使用非常灵活，可以对整个设计的所有状态机都用 FSM Compiler 进行优化，也可以仅仅对指定的状态机进行优化。对整个设计使用 FSM Compiler 进行优化，只需在主界面重要综合优化参数中选择【FSM Compiler】选项，或者在综合优化参数设置时选中【FSM Compiler】选项即可。如果觉得设计其他 FSM 已经满意，而仅对某个 FSM 不满意时，可以在源代码或综合约束文件中手动添加综合属性，指定对单独状态机的编译与优化。FSM Compiler 综合属性如表 4-3 所示。

表 4-3 FSM Compiler 综合属性用法

语法 语言	禁止优化所指定的 FSM	优化所指定的 FSM
Verilog	reg [3:0] curstate /* synthesis syn_state_machine=0 */;	reg [3:0] curstate /* synthesis syn_state_machine=1 */
VHDL	signal StateName: state_type; attribute syn_state_machine :boolean; attribute syn_state_machine of curstate: signal is false;	signal StateName: state_type; attribute syn_state_machine : boolean; attribute syn_state_machine of curstate: signal is true;

2. 有限状态机探测器（FSM Explorer）

FSM Explorer 使用 FSM Compiler 的编译结果，遴选不同的编码方式进行状态机编码试探，从而达到对 FSM 编码的最佳优化效果。与 FSM Compiler 相比，FSM Explorer 的优化效果往往更好，编译优化所花费的时间也更长。

对设计使用 FSM Explorer 的方法也有两种：第一种是对整个设计的所有 FSM 自动运用 FSM Explorer；第二种是对设计中特定的 FSM 使用 FSM Explorer。第一种方法可以在 Synplify Pro 主界面重要综合优化参数中选择 FSM Explorer 有效，或者在综合优化参数设置对话框选中【FSM Explorer】选项；第二种方法需要在源代码或者综合约束文件中添加使用 FSM Explorer 的综合属性声明。FSM Explorer 综合属性如表 4-4 所示。

表 4-4 FSM Explorer 综合属性用法

语法 语言	手动指定 FSM 是否优化	手动指定 FSM 编码方式
Verilog	reg [3:0] curstate /* synthesis state_machine */;	reg [3:0] curstate /* synthesis syn_encoding "gray"*/
VHDL	signal curstate : state_type; attribute syn_state_machine : boolean; attribute syn_state_machine of curstate : signal is true;	signal curstate : state_type; attribute syn_encoding : string; attribute syn_encoding of curstate : signal is true;

3. 有限状态机观察器 (FSM Viewer)

在 Synplify Pro 中除了可以使用 RTL 视图和结构视图观察、分析 FSM 外，还可以使用专用 FSM 观察器 (FSM Viewer) 分析 FSM。FSM Viewer 将源代码中描述的 FSM 根据 FSM Compiler 和 FSM Explorer 的编译优化结果，用状态转移图显示有限状态机。

下面以 4.1.3 节设计的“ALU”为例，讲述 FSM Viewer 的使用方法。

🔑 使用 FSM Viewer 分析有限状态机，参考示例详见本书附带光盘的“Example-4-1”目录

1. 打开前面创建的工程“ALU_Syn_demo”，启动 RTL 视图，选择状态机模块“statemachine”，单击  按钮进入状态机层次结构，或者单击鼠标右键，在弹出命令菜单中选择【View FSM】命令，如图 4-46 所示。

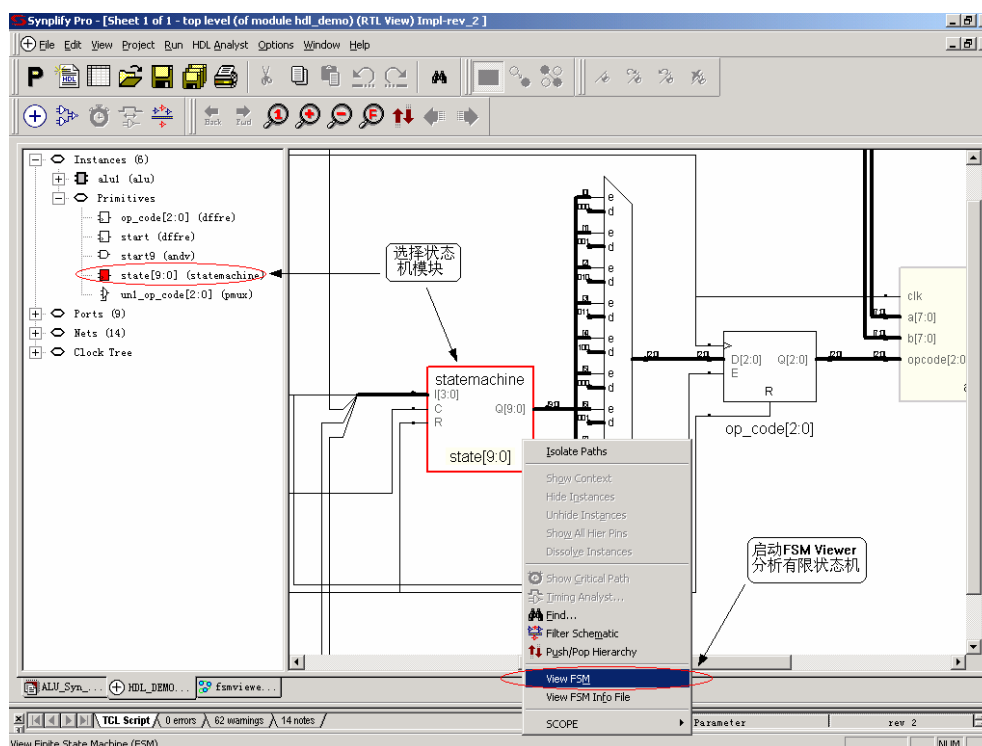


图4-46 启动 FSM Viewer 分析有限状态机

2. FSM Viewer 的主界面主要由状态转移图和 FSM 信息显示选项卡组成。状态转

移图是源代码经过编译再现的状态机。FSM 信息显示包含转移条件 (Transitions)、寄存器传输级状态编码 (RTL Encodings) 和映射后状态编码 (Mapped Encodings) 3 个选项卡, 如图 4-47 所示。

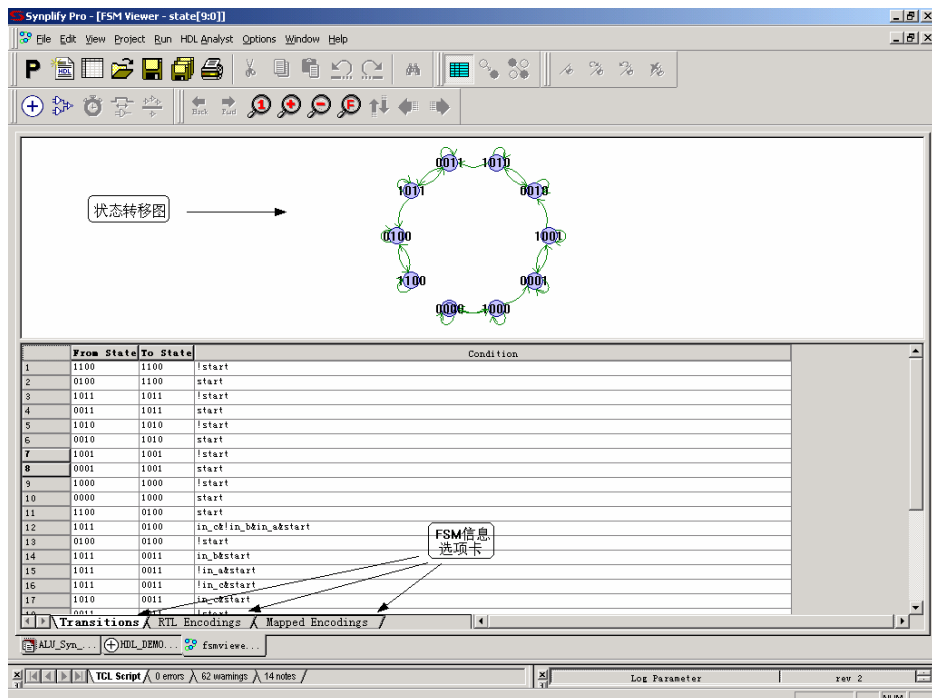


图4-47 FSM Viewer 主界面

3. 需要指出的是, 在 FSM Viewer 中, 状态转移图和转移条件, 寄存器传输级状态编码和映射后状态编码等对象之间也能通过 Cross-Probing 相互链接与切换, 如图 4-48 所示。

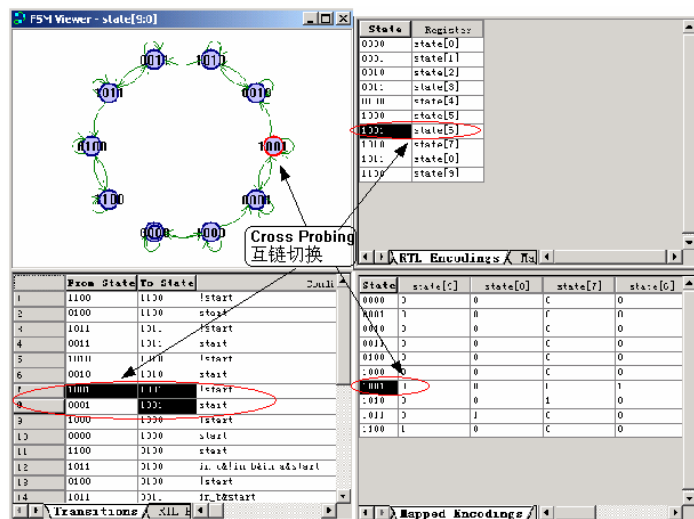


图4-48 FSM Viewer 中的 Cross-Probing 互链切换

4. 通过  工具栏或者在状态转移图中选择相应的对象（转移条件、状态等），单击鼠标右键，在弹出的菜单中可以完成显示对象的选择和屏蔽，有利于理解状态之间关系，增加状态转移图的可读性，如图 4-49 所示。

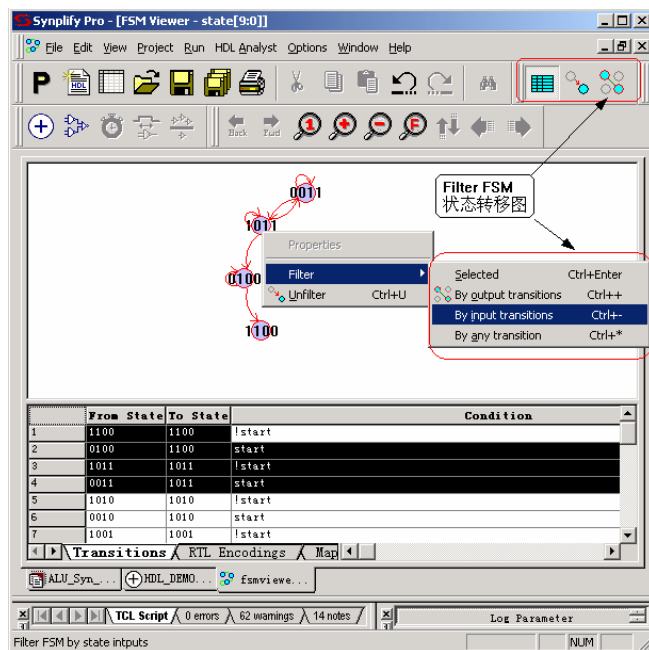


图4-49 在 FSM Viewer 中选择和屏蔽状态

4.1.4.3 使用 Pipelining 选项优化设计的技巧

Pipelining 是 Synplify Pro 中的一项非常实用的时序优化技术。它通过向某些特殊逻辑中插入寄存器，以达到减少逻辑级数，改善时序条件的目的。Pipelining 根据不同器件型号采取不同的具体操作。如在 Xilinx Virtex 器件中，Pipelining 向 ROM 或者乘法器中插入寄存器，以达到改善时序的目的。Pipelining 的原理示意图如图 4-50 所示。

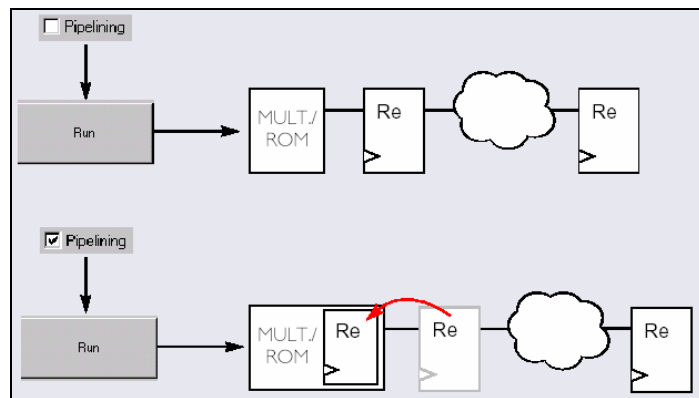


图4-50 Pipelining 原理示意图

Pipelining 选项的添加方法多种多样，可以在主界面重要综合优化参数下选择有效，可以在综合优化参数对话框中选择有效，也可以在 SCOPE 中添加约束属性，还可以在源代码

中添加约束属性。

4.1.4.4 使用 Retiming 选项优化综合时序的技巧

Retiming 是 Synplify Pro 提供的一个改善时序电路综合频率的强力工具。与 Pipelining 相比, Retiming 的应用范围更加广泛, 优化效果更加明显。使用 Retiming, 设计者不用更改任何设计逻辑, Retiming 自动向组合逻辑和查找表 (LUT, Look Up Table) 中插入寄存器, 这种寄存器的插入其本质是将组合逻辑和 LUTs 以外的寄存器移入组合逻辑和 LUTs 中, 从宏观考虑, 被优化部分的输入和输出之间的寄存器级数没有变化, 所以并不会破坏设计的时序。需要说明的是, Retiming 仅对沿触发的寄存器有效, 并不能移动电平敏感的锁存器。另外, 目前 Retiming 级数仅仅对某些特定型号的 Xilinx 和 Altera 器件有效。Retiming 的原理示意图如图 4-51 所示。

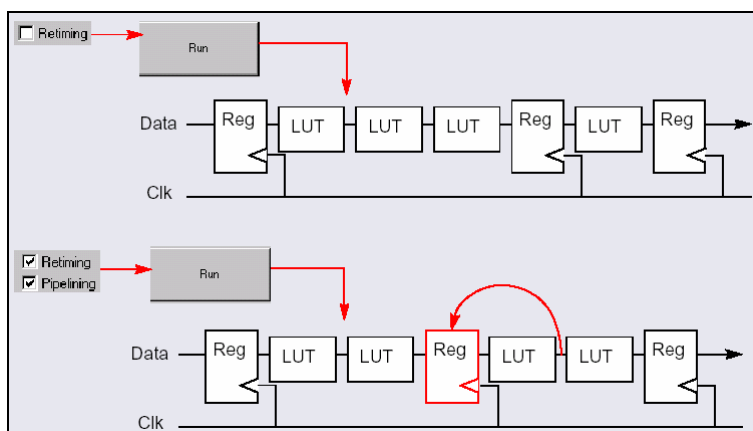


图4-51 Retiming 原理示意图

Retiming 的添加方法也有 4 种：在主界面重要综合优化参数下选择有效；在综合优化参数对话框中选择有效；在 SCOPE 中添加约束属性；在源代码中添加约束属性。如果选择 Retiming 有效, 在综合结果报告中会有详细的 Retiming 操作报告, 包括添加、删除寄存器数目, 删除的寄存器名称, Retiming 添加的寄存器名称等内容。

4.1.4.5 综合约束补充说明

前面已经介绍了很多和综合约束相关的内容, 现在对有些文件做进一步澄清。

• 多种途径添加约束方法

一般来说添加综合约束的方法有两种：第一种方法是使用 SCOPE 添加, 特别是使用属性 (Attribute) 设置页面添加综合约束属性；第二种方法是从 HDL 源代码中添加综合约束属性。对于重要的综合优化参数添加方法有四种, 除了上述两种方法外, 还可以在主界面重要综合优化参数选项中选择有效, 或者在综合优化参数对话框中选择有效。这四种添加方法前面已有相应的介绍。

• 综合约束与布局布线阶段约束的区别与联系

综合约束和布局布线阶段的约束既有区别又有联系。在 Synplify Pro 中, 综合约束文件为 SDC 文件, 在 Xilinx ISE 中布局布线约束常通过用户约束文件 UCF 实现。两者的联系为：随着综合器与布线器的协调性越来越好, 几乎所有的布线约束都可以在综合阶段添加。

这些约束包括时钟约束、寄存器约束、路径约束、引脚约束和 FPGA/CPLD 底层原语约束等等。只是其中有些约束在综合阶段并不起实际作用，仅仅由综合器传递到布线器中去，在布线阶段才真正发挥作用。两者的区别为：综合约束和布线约束的重点及目的不同，所以产生的效果也不同。布线约束与器件底层结构结合得更紧密，直接决定设计在芯片上的实现方式。另外，布线约束的优先等级比综合约束高，当两者发生冲突时，布线时的约束优先。

Synplify Pro 综合步骤的约束属性和 Xilinx 实现步骤的约束属性的沟通工具是 NCF 文件。Synplify Pro 将综合约束 SDC 文件中的约束条件进行适当地语法转换，生成 NCF 约束文件，用户只需将 NCF 约束文件拷到 Xilinx 实现目录下即可，NCF 文件会自动指导 Xilinx 的实现步骤。所以说，Synplify Pro 综合约束向 Xilinx 实现约束传递的工具就是 NCF 文件。NCF 文件与 Xilinx 用户约束文件 UCF 的最主要区别是，UCF 约束的优先级比 NCF 文件高，当两者发生冲突时，以 UCF 约束为准。

如果在 Synplify Pro 综合阶段设计了非常完备的综合约束文件，通过 Synplify Pro 自动产生的 NCF 文件将约束传递到 Xilinx 布局布线器，就大大降低了设计 UCF 文件的要求。

4.1.4.6 使用 Cross-Probing 的技巧

通过前面章节的学习，读者也应该对 Cross-Probing 留下了深刻的印象。Cross-Probing 是 Synplify Pro 的一个功能强大的调试工具，它联系源代码、RTL 视图、结构视图、FSM Viewer 状态转移图、综合报告和信息显示等，可以说 Cross-Probing 虽然没有任何命令与图标，但是却在 Synplify Pro 中无处不在。使用 Cross-Probing 技术明确了设计各个部分之间的对应关系，加深了用户对设计的理解，使设计的调试变得更加轻松。

4.1.4.7 多位置综合流程 (MultiPoint Synthesis Flow) 的使用技巧

MultiPoint Synthesis Flow 是 Synplify Pro 的一项重要新增功能。其本质是对设计运用增量综合与增量实现技术。在综合过程中，通过 SCOPE 在多位置编译 (Compile Points) 页面设置那些没有改变的模块为锁定状态 (locked)，综合器在综合与优化这些锁定模块的父模块和相关设计时，保证与锁定模块相关的接口和边界不做任何优化，从而保证了锁定模块的综合结果与以前完全一致。

另一方面，Xilinx 等布线器也支持增量布局布线的实现设计。所以将综合结果导入布线器后，锁定模块的布局布线结果与以前也完全一致，从而继承了以往综合与实现的成果，提高了综合与实现的效率。关于增量综合技术与增量实现详见本书第 8 章——增量设计与模块化设计技术。

4.1.4.8 时序分析专家 (Timing Analyst) 的使用技巧

时序分析专家也是 Synplify Pro 新版本的一个新增功能。Timing Analyst 对 Xilinx 和 Altera 器件进行点到点的路径时序分析，是结构化视图 (Technology View) 时序分析功能的有益补充。下面仍用“综合流程”小节的设计为例，讲述 Timing Analyst 的使用方法。

使用 Timing Analyst 分析点到点的路径时序，参考示例详见本书附带光盘的“Example-4-1”目录

1. 打开前面创建的工程“ALU_Syn_demo”，启动结构视图，单击按钮，启动 Timing Analyst。如果需要分析从输入控制信号“op_code”到输出“result”之间的路径时序，则添加路径的起点和终点，如图 4-52 所示。

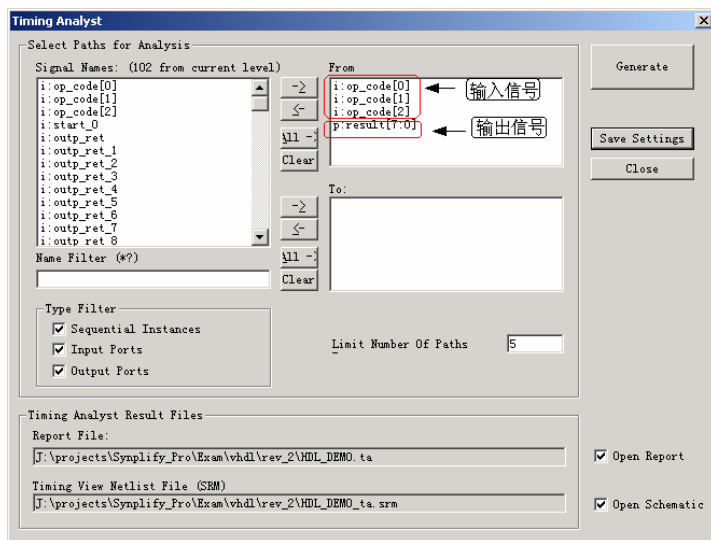


图4-52 在 Timing Analyst 中添加路径的起点和终点

2. 保存设置后, 单击 **Generate** 按钮则自动产生起点到终点间的路径结构图, 如图 4-53 所示。

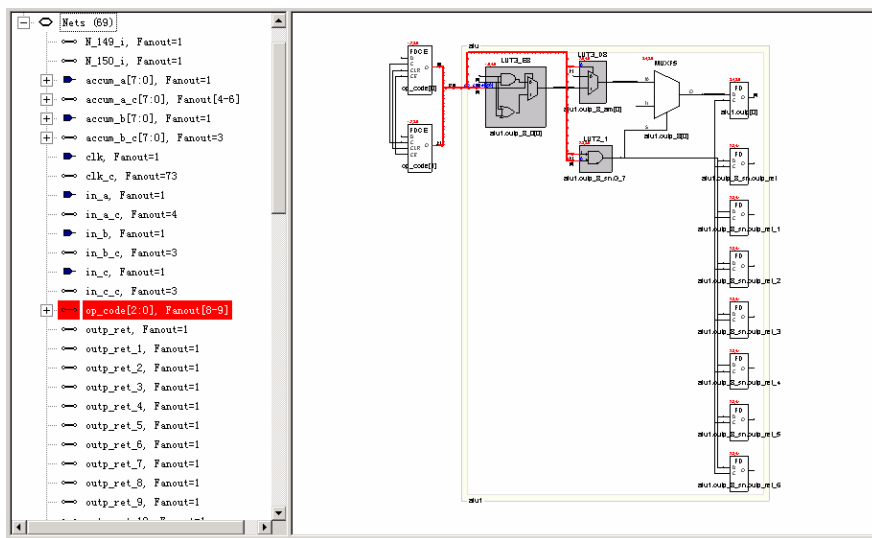


图4-53 输入信号“op_code”到输出“result”之间的路径

4.1.4.9 处理不满足时序的设计 (Negative Slack) 的技巧

不满足时序约束条件则在时序报告中可以看到负数的“Slack”。改善时序约束的基本方法有以下 3 种。

- 适当的约束能在一定程度上提高工作频率。
分析设计的引脚路径, 将关键路径单独约束, 而且远远满足设计要求的逻辑松约束, 重新综合。另外可以尝试 Synplify Pro 的一些时序改善工具如: Pipelining、Retiming、FSM Explorer 等。适当减小关键路径的扇出 (Fanout), 因为高扇出需要多级缓冲 (Buffer) 驱动, 会影响速度。取消资源共享

(Resources Sharing) 选项的选择。多做一些不同综合优化参数下的综合实验，找出频率最高的参数组合方式。

- 使用 Amplify 等物理约束综合器。

对于某些设计，采用 Synplicity 公司的物理综合器 Amplify 进行关键路径的物理位置约束，可以较大幅度提高布局布线效率，改善工作频率。

- 合理的编码风格 (Coding Style) 是提高设计质量的根本。

实际上提高工作频率的最根本方法还是改善编程代码风格。为了提高设计频率和工作可靠性，一般采用同步时序电路设计。如果工程师对设计的硬件实现方式了然于胸，而且积累了好的代码风格与设计表示技巧，那么可以使设计在速度和面积上达到最优。

4.1.4.10 声明黑盒子 (Black Box) 的综合方法与技巧

黑盒子的概念在第 2 章中已经介绍过。综合器一般把与 FPGA/CPLD 硬件相关的底层模块和硬件原语、IP Core 等都综合成黑盒子。在 Synplify Pro 中将某个模块综合成黑盒子的方法有两种：一是对该模块仅仅进行模块名称和端口声明，其功能实体为空，则 Synplify Pro 自动将之综合成黑盒子；第二种方法是使用综合约束条件，在 SCOPE 中属性 (Attribute) 设置页面或者 HDL 源代码中添加综合成黑盒子的属性，如表 4-5 所示。

表 4-5 Black Box 综合属性声明

语法 语言	综合为黑盒子	指定黑盒子引脚
Verilog	<code>/* synthesis syn_black_box */;</code>	<code>/* synthesis syn_black_box black_box_pad_pin="引脚名"*/</code>
VHDL	<code>attribute syn_black_box of bbox: component is true;</code>	<code>attribute black_box_pad_pin of 模块名: component is "引脚名";</code>

• Xilinx 模块声明文件

如果在设计中使用了如“CLKDLL”、“BUFG”等 Xilinx 的底层模块和硬件原语，一般需要综合成黑盒子。除了使用上述两种方法声明黑盒子外，Synplify Pro 还为 Xilinx 器件专门作了模块声明文件，在源程序中直接使用“include”命令包含相应的器件族的模块声明文件，或者将相应器件族的模块声明文件加入到工程中，就可以直接将所用到的底层模块和硬件原语综合成黑盒子。

这些模块声明文件存放在 Synplify 安装目录下的“lib\Xilinx”子目录下，以器件族（如 Virtex、VirtexE、Virtex2、Virtex2P、CoolRunner、CoolRunner2 等）分类，有 Verilog 和 VHDL 两种语言形式，使用非常方便。

• 使用 STAMP 模块定义黑盒子时序

STAMP 即 Synopsys 模块描述语言，该语言为大多数 EDA 软件制造商和器件制造商所支持。使用 STAMP 可以描述黑盒子的时序，如延时、建立保持时间等。在 Synplify Pro 中对黑盒子使用 STAMP 描述时序关系可以使 Synplify Pro 综合结果估算出的频率关系与实际布局布线后的频率关系更加接近。

4.1.4.11 Probe 探针功能的使用技巧

Probe 探针是 Synplify Pro 的一个非常有用的测试辅助工具，在设计中通过约束属性插

入 Probe，就能方便地将任何信号作为输出信号，而源代码不用做任何更改。使用 Probe 将测试信号拉到测试引脚的最大好处是省掉了添加、删除测试信号时需要由子模块向父模块一层层向上传递的麻烦，保证了代码清晰、干净。除了不能对输出信号和双向信号添加 Probe 探针外（这两者也没有必要添加探针），可以对设计中任何寄存器和网线信号添加 Probe 探针属性。另外自 ISE 5.1 以后的版本中，添加的 PACE 引脚与区域约束编辑器也可以完成同样的功能。

• Probe 的插入方法

添加 Probe 约束属性的方法有两种：使用 SCOPE 属性（Attribute）设置页面添加探针或者从 HDL 源代码添加探针。Probe 探针综合约束属性语法如表 4-6 所示。

表 4-6 添加 Probe 探针综合属性

语法 语言	添加 Probe 探针综合属性
Verilog	reg [7:0] alu_tmp /* synthesis syn_probe=1 */; 信号声明后面直接加/* synthesis syn_probe=1 */;综合属性
VHDL	signal alu_tmp : std_logic_vector(7 downto 0) ; attribute syn_probe : boolean; attribute syn_probe of alu_tmp : signal is true;

相比之下，在 SCOPE 属性页面添加 Probe 探针不需要在源代码中做任何添加，操作更加方便。具体操作为：打开 RTL 视图和 SCOPE，在 RTL 视图的网线列表中选中需要添加 Probe 的信号，将该信号用鼠标拖拉到 SCOPE 的综合属性（Attributes）页面，然后选择综合属性为“syn_probe”，设值为“1”即可，如图 4-54 所示。

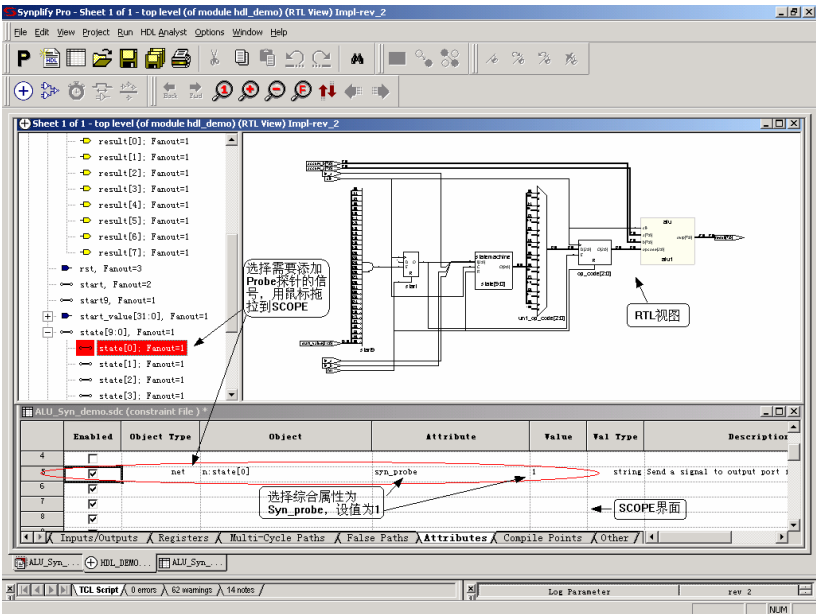


图4-54 对信号添加 Probe 探针属性拉到输出端口

保存综合约束文件，重新综合，添加了 Probe 探针属性的信号将自动被拉到输出端口。

4.1.4.12 针对 Xilinx 器件的一些使用技巧

随着 Synplify Pro 与 Xilinx 合作日趋密切，Synplify Pro 对 Xilinx 器件的支持力度也越来越大，主要表现为 Synplify Pro 可以自动将设计类推为 Xilinx 的底层模块和硬件原语，通过专用硬件结构，提高了设计速度，减少了占用面积。例如 Synplify Pro 可以自动将移位寄存器类推为“SRL”结构。SRL，即基于查找表的移位寄存器，其本质是用一个查找表和一个触发器结构，实现最大级数为 16 的固定级数或可变级数的移位寄存器。恰当地使用 SRL 结构能大幅度节约 FPGA 资源。Synplify Pro 还可以自动类推 Xilinx 的块 RAM (Block RAM)，时钟数字锁相环等结构。

另外值得一提的是在 Synplify Pro 界面下完成综合后，可以直接运行【Options】菜单中的【Start Floorplanner】和【Start ISE Project Navigator】命令，直接启动 Xilinx ISE 工程管理器 and 布局规划器进行翻译、映射、布局布线等实现步骤。

4.2 Xilinx 最早的合作伙伴——Synopsys 综合工具

Synopsys FPGA Express 综合工具是 Xilinx 最早集成的综合工具之一。

Synopsys 是最早的 EDA 工具厂商之一，其 FPGA Express 是 Xilinx 早期产品 (Foundation 系列，低版本 ISE) 惟一集成的综合工具，熟悉 Xilinx 早期产品的工程师一定对它非常熟悉。FPGA Express (以下简称 FE) 的高级版本 FPGA Compiler II 是一个非常成熟的 FPGA、ASIC 设计平台。虽然目前在 FPGA 设计中使用 Synopsys 综合工具的用户数量越来越少，但是为了照顾使用早期 ISE 和 Foundation 软件平台的用户，本节简要介绍一下 Synopsys 综合工具的使用方法。

FE 的主要功能是将 VHDL 代码、Verilog 代码综合优化后生成 FPGA 网表文件。FE 支持 Windows、Solaris 和 HP-UX 等操作系统，支持图形化用户界面和脚本文件接口两种方式的操作。FE 在 FPGA 设计流程中所处的位置如图 4-55 所示。

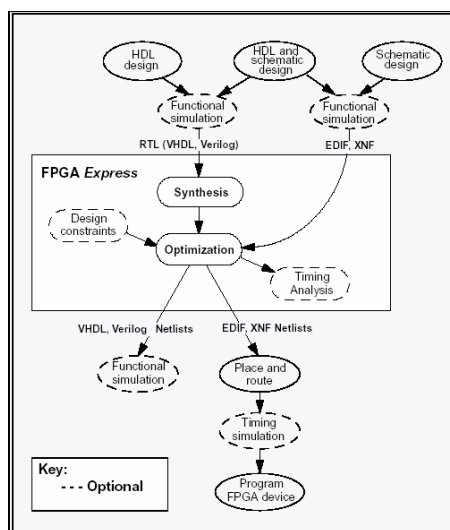


图4-55 FE 在设计流程中的位置

FE 支持的输入文件有 VHDL、Verilog 源代码，未经优化的 EDIF 网表和 XNF 网表等。FE 的输出主要有两部分：一是优化过的 EDIF 网表文件或者 XNF 网表文件；二是综合后的仿真模型与门时延文件。通常情况下没有经过优化的网表文件是从图形编辑器或者状态机编辑器中生成的，也可以是经过预优化的宏单元库。

4.2.1 设计流程

FE 支持 DesignWizard、Push-button、Performance (constraint-driven)、Multiple-device (hierarchical) 和 Script-based 等设计流程，这些设计流程体现了 FE 的基本操作方法。这些流程基本大同小异，主要区别在于应用场合和设计目的不同。下面简要介绍这些设计流程。

4.2.1.1 DesignWizard 设计流程

DesignWizard 设计流程即设计向导流程。设计者在 FE 设计向导的逐步引导下完成整个综合优化过程。默认设置下运行 FE 工具的时候，DesignWizard 设计流程会自动运行。选择【File】/【DesignWizard】命令也可启动 DesignWizard 设计流程。在这个流程中忽略了约束文件的编辑和修改，所以 DesignWizard 设计流程一般使用在第一次预综合一个新的设计工程的时候。

DesignWizard 设计流程包括以下步骤：

- 添加源文件，指定设计的顶层；
- 指定目标器件、优化参数、输出类型；
- 运行综合优化过程，自动输出综合网表。

其中源文件可以选择 FE 所支持的任何类型，也可以混合使用 Verilog、VHDL 两种寄存器传输级描述语言输入。在选择目标器件的时候，可以根据需要设置综合参数，其中包括时钟频率、CPU 的工作难度、在指定的情形下进行面积或者速度优化、是否保留设计层次等。也可以设置输出参数，其中包括工程名称、输出文件存储目录、输出网表格式以及仿真模型格式等项目。

4.2.1.2 Push-button 设计流程

Push-button 设计流程即按钮流程，是指在建立了一个工程项目后，依次使用工具栏中的各个按钮完成综合过程。按钮流程也忽略了约束文件的编辑修改，与设计向导流程相似，仅仅在第一次综合新的设计工程时使用。使用上述两种 FE 综合流程，可以方便地得到新设计的基本综合信息，为下面详细综合与优化步骤提供所需信息和依据。

Push-button 设计流程包括以下步骤：

- 新建一个工程，设置工程名称；
- 添加设计源文件；
- 选择目标器件；
- 输出网表。

选择【File】/【New Project】命令，新建一个工程，设置工程名称。单击按钮，添加设计源文件；当源文件修改后，单击更新源文件，对源代码检错；然后单击按钮，在弹出的综合参数对话框中，设置器件型号、时钟频率、CPU 的工作难度、优化目标（面积

或者速度优化)、是否保留设计层次、是否插入 I/O 引脚, 以及是否跳过综合约束编辑等参数。单击  按钮就可以完成综合优化, 得到输出结果。Push-button 设计流程如图 4-56 所示。

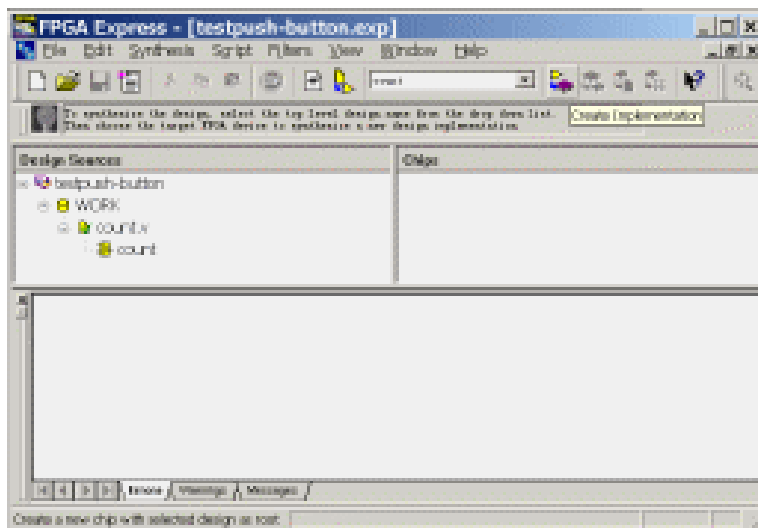


图4-56 Push-button 设计流程

4.2.1.3 Performance 设计流程

Performance 设计流程即约束驱动流程, 当需要提高设计密度或者工作频率等指标时, 可以使用约束驱动流程来综合设计。使用约束驱动流程时, 必须在综合参数对话框中取消【skip constraint entry】选项的勾选。这样 FE 首先仅对设计进行检查和细化, 但并不进行综合优化, 当用户使用 FE 约束编辑器设计综合约束文件后, FE 才根据综合约束文件的目标对设计进行全面的编译、综合、优化。

4.2.1.4 Script-based 设计流程

Script-based 设计流程即使用 FE 提供的命令脚本工具来控制综合优化过程, 此流程可以提供其他流程所提供的所有功能。

4.2.1.5 Hierarchical (Multiple-Device) 设计流程

Multiple-Device 设计流程即多器件流程。有时需要对同一个设计比较不同厂商器件的实现结果, 这时就可以选择多器件流程对设计进行综合优化。使用该流程可以在同一工程项目中使用相同的源代码和综合约束文件对不同型号器件的综合优化结果进行对比。

多器件流程可以通过以下几个步骤完成：

- 新建工程；
- 选择目标工艺, 细化自己的设计；
- 输入约束和控制；
- 优化输出网表、生成报告, 输出约束文件；
- 分析时序报告；

- 重新指定的综合参数（器件型号、速度、综合目标等），创建新的综合项目；
- 导入约束文件；
- 重新优化、输出网表和报告。

以上介绍的设计流程，其操作过程基本相同，在实际使用中可以根据自己的需要选择。这些流程之间也没有什么特别的规定，通常都是混合使用各种流程来达到自己的需求。总的来说，FE 的操作过程可以分为如下步骤：

- 设计源文件；
- 启动 FE；
- 新建工程；
- 添加并调试源文件；
- 细化设计；
- 设计综合约束文件；
- 优化设计；
- 分析综合结果；
- 导出优化后的 EDIF 或 XNF 网表文件；
- 生成综合后仿真的 Verilog 或者 VHDL 模型。

4.2.2 FE 综合优化过程

本节使用一个简单的实例，按照上节介绍的 FE 综合过程，演示 FE 中综合优化设计的方法。

 以一个简单的计数器为例，讨论 FE 综合优化过程，参考示例详见本书附带光盘的“Example-4-2”目录

将目录“Example-4-1”中的内容拷贝到自己的工作目录下，读者可以浏览源代码以便对设计有初步了解，但是不要对程序做任何修改。为了说明方便，请将文件拷贝到“D:\work\example\4_1\source”目录下。

4.2.2.1 运行 FE

从操作系统主窗口中运行 FE。在默认情况下，FE 自动打开设计向导辅助用户创建综合工程，请先关闭设计向导，进入 FE 主用户界面。FE 主用户界面包括工具栏、消息栏、设计源文件窗口、Chip 窗口和输出窗口等。

4.2.2.2 新建工程

单击  按钮，或选择【File】/【New project】命令创建新的工程。

在主窗口中，FE 将显示创建新工程的对话框。在这个对话框中可以选择工程存放的目录，工程可以存储在硬盘的任何一个地方，选择“work\example\4_1”目录，这时在 name 域键入工程的名称“tutor_porj”，然后单击  按钮。FE 会在 4_1 目录下创建一个名为“tutor_porj”的目录，在 tutor_porj 目录下还会创建“work”子目录，用来存放工程数据文件。

4.2.2.3 添加源文件

向工程添加源文件可以通过添加源文件对话框完成，有以下 3 种方式启动添加源文件对话框：

- 单击工具栏中的  按钮添加文件。
- 选择【Synthesis】/【Add source files】命令。
- 在设计源文件窗口单击鼠标右键，在弹出的菜单中选择【Add Sources in WORK】命令。

在添加源文件对话框中，指定所需添加文件的路径，选择源文件，单击  按钮完成源文件添加。文件添加过程中，FE 自动对源文件进行分析。分析完成后，每一个文件左边会有图标指示源文件的分析结果。图标  表示源文件中至少会有一个错误；图标  表示源文件完全正确，图标  表示源文件更新后还没有进行语法分析。

同样可以在 FE 中对源文件进行调试。如果在分析设计中的某个源文件时，在输出窗口出现了错误或者警告信息，那么需要对源程序进行调试。

输出窗口显示的输出信息中将包括发生错误或者警告的文件名称和错误或警告所在位置等信息，可以使用 FE 内嵌的 HDL 编辑器来修改文件的错误。

例如在添加源文件对话框中选择 example\4_1\source 目录下的 tutor.v 文件和 counter.v 文件，单击  按钮后，FE 就开始对源文件进行分析。如果文件中有错误，那么在【Design Sources】窗口中相对应的文件上会出现一个红色的叉号，如图 4-57 所示。

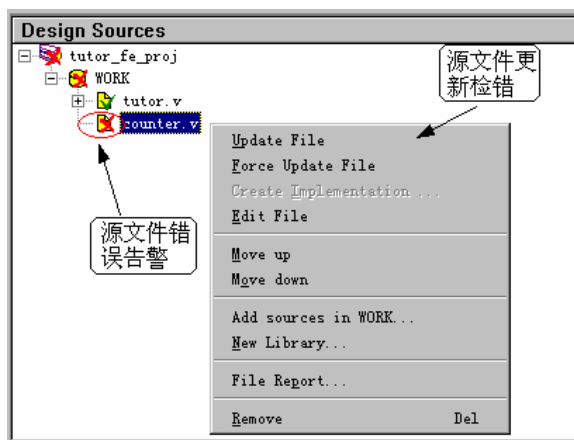


图4-57 文件分析

4.2.2.4 修改错误和警告

在源文件窗口中双击出现错误的文件名，在输出窗口中就会显示相应错误和警告的说明，说明中包含错误出现的行号、错误类型等信息。可以打开 FE 内嵌的 HDL 编辑器对错误进行修改。

有两种方式可以启动 HDL 编辑器：

- 在设计源文件窗口中单击鼠标右键，在弹出的菜单中选择【Edit File】命令。
- 在输出窗口中双击错误或者警告信息。

打开源文件编辑器以后，出现图 4-58 所示界面，编辑器使用红色标示出错误出现的地方。在这个例子中，错误出现在“fi (NCLR==0'b0)”行，在这一行有个笔误——将“if”写成了“fi”，修改错误后保存源文件。

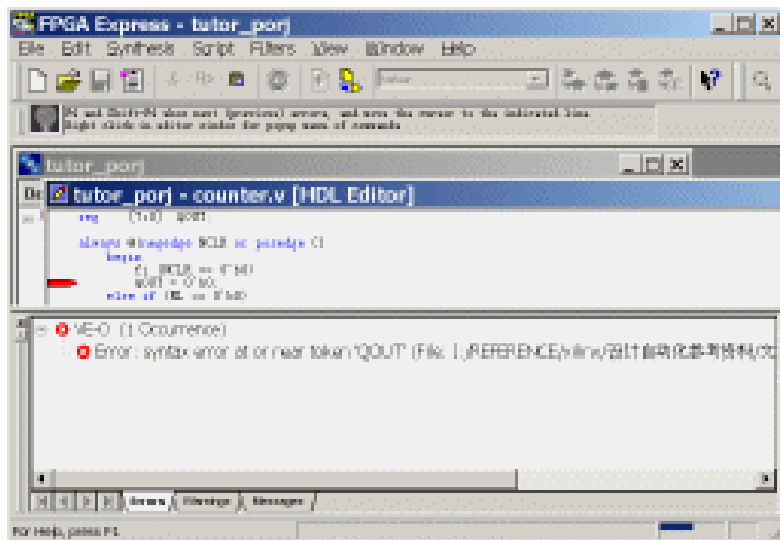


图4-58 FE 中的编辑界面

4.2.2.5 更新文件

当修改完以后，可以在 HDL 编辑器中单击鼠标右键，在弹出的菜单中选择【Analyze File】命令对修改后的文件进行检错分析。如果没有进行这一步操作，那么关闭文件编辑器后，在源文件设计窗中“counter”文件左边就会出现  图标，表示该文件的当前状态是未知的。

如果源文件状态未知，则必须对文件进行更新，可以通过以下两种方式完成：

- 单击设计源文件窗口中的 按钮。
- 单击鼠标右键，在弹出的菜单中选择【Update Files】命令。

如果文件更新结果完全正确，该文件左边显示 图标，表示文件无语法错误。

4.2.2.6 细化设计 (Elaborating Designs)

细化设计的本质是编译设计源文件，并生成一个未经优化的逻辑网表的过程。FE 在这个过程中将设计源文件转化为使用了工艺基本单元来描述的逻辑门电路。细化设计时，需要指定设计的顶层模块、设置综合优化参数（例如目标器件、时钟速度、优化目标等）。

在 FE 综合流程中，需要明确指出设计的顶层模块，FE 根据指定的顶层模块生成设计层次结构，从而完成内部的信号连接。在模块化设计流程中，也可以对各个子模块进行编译与细化。这时顶层模块并不是设计整体上的顶层，而仅仅是子模块的顶层而已。

可以通过以下几种方式指定顶层文件：

从顶层设计的菜单中选取顶层模块名称，如图 4-59 所示，选择“tutor”作为顶层模块，指定顶层模块后，FE 自动弹出综合参数设置对话框。

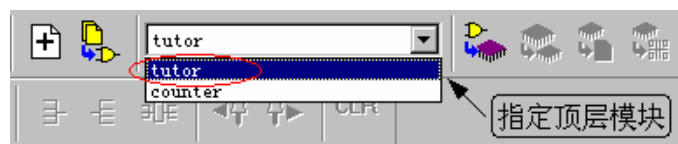


图4-59 选取顶层模块

也可以在设计源文件窗口中选取顶层文件。这时需要展开顶层设计的模块，从这些单元中选取顶层模块。然后将鼠标停留在顶层模块上。单击工具栏中的按钮，弹出综合参数对话框。

不管通过哪种方式，选取了顶层模块后都会弹出如图 4-60 所示的综合参数设置对话框，通过该对话框中指定重要的综合优化参数。

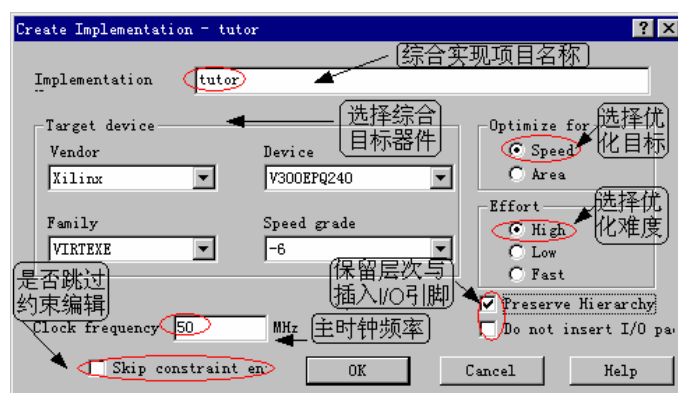


图4-60 FE 中的综合参数设置

- (1) “综合实现”项目名称。这里的“实现”是指综合步骤中将逻辑网表映射的具体器件并优化设计的过程，与 Xilinx 的由翻译、映射、布局布线等 3 个步骤组成的实现过程虽然英文名称相同，但是意义不同。同一个综合工程中，可以选择不同的综合参数（目标器件、时钟频率、优化目标等）进行多次“综合实现”，所以需要指定不同的“综合实现”项目名称加以区分。输入“综合实现”的项目名称，如果不输入名称，FE 将以顶层模块的名称作为实现项目名称。本例使用默认项目名称。
- (2) 选择目标器件和速度等级。器件选型应该在设计阶段根据设计的速度和面积需求完成。本例设置如下：
 - 【Vendor】：Xilinx
 - 【Device】：V300EPQ240
 - 【Family】：VIRTEXE
 - 【Speed grade】：-6
- (3) 设置时钟频率。设置了默认的时钟频率后，设计的全部时钟都使用默认时钟进行约束。该时钟频率是对整个设计的全局时序约束。综合步骤设置的约束条件会直接影响实现步骤的布局布线结果。因此，不要对设计进行随意约束，而要根据具体的设计需求设置主时钟的频率，约束过紧或者过松都不会

达到好的综合效果。一般相对设计需求有 5~10% 的余量比较合适。本例使用默认设置 50MHz。

- (4) 选择优化目标和优化等级。优化目标为速度优先或是面积优先，一般来说设计的时序要求比占用芯片面积要求更重要，所以选择速度优化优先。默认情况下优化难度选择高 (High)，以达到更好的优化效果。
- (5) 【Do Not insert I/O pads】选项。该选项用于指定在优化时是否需要插入输入/输出引脚 (I/O pad)。如果当前工程的顶层模块就是整个设计的顶层，其输入/输出信号作为 FPGA/CPLD 的输入/输出，就需要选择插入 I/O pad；如果某模块仅仅是其他模块的子模块，产生的综合网表被其他模块调用，其输入/输出信号仅仅是设计内部的输入/输出信号，这种情况不需要插入 I/O pad。本例使用默认选项，插入 I/O pad。
- (6) 是否需要跳过约束编辑。因为本例需要对设计添加综合约束，所以不选本项。
- (7) 是否需要保留层次。该选项用于指定综合时是否保留设计的层次结构。如果保留层次，在后仿真时比较容易查找模块、信号和路径。如果不保留层次，综合器就将设计打平，整个设计的所有结构都可以充分复用，以达到综合结果的物理面积更小的效果，但是综合耗费时间更长，后仿真时也不易查找模块、信号和路径。

4.2.2.7 设计综合约束文件

前面提到，系统时钟频率是一种针对主时钟添加的时序约束，但是该约束仅仅是综合约束很小的一个部分。本节将详细讲述如何设计综合约束条件。

在将一个设计综合到特定的目标器件以前，可以设置综合约束条件。FE 中设计综合约束的方法有两种：一是通过约束编辑器生成综合约束文件，二是直接手动编辑约束文件。两者的区别在于：约束编辑器的图形用户界面比较友好，便于使用和维护，其缺点是可以使用的约束命令的种类有限，约束的灵活性受到了一定的限制。在图形用户界面中，用户仅可以对最常使用的约束属性进行设置。手工编写约束文件的优点是可以支持所有的约束命令，缺点是 FE 的约束语法不易掌握，约束文件不易维护。虽然设置约束条件本身是个可选项，但是笔者强烈推荐这种做法，在设计中添加必要的约束可以改善布局布线的最终结果。如果设计比较大，并且有较多层次，就可以通过层次约束提高布局布线的结果。有一点需要注意，约束过紧会对布局布线有不利影响，所以应对需要的地方进行约束。

在 FE 的约束编辑器中，从逻辑关系上将需要约束的各个项目分为几个不同的组，如时钟、路径、端口、模块等，如图 4-61 所示。FE 软件可以自动区别出这些分组的一些特性，如时钟名、端口名、设计层次，并将这些设计信息显示在表格中。

设置约束属性与控制参数：

- (1) 在 Chip 窗口内，使用鼠标选中未优化的器件符号，或者选中未优化的模块名称，单击鼠标右键，在弹出的菜单中选择【Edit Constraints】命令，启动约束编辑器，如图 4-61 所示。图中各种约束根据其功能分为时钟、路径、端口和模块等。另外还有一个附加寄存器选项，该选项和工艺与电气特性相关。也

就是说，仅仅对于某些特定工艺与电气特性的器件才能设置这个附件寄存器选项。

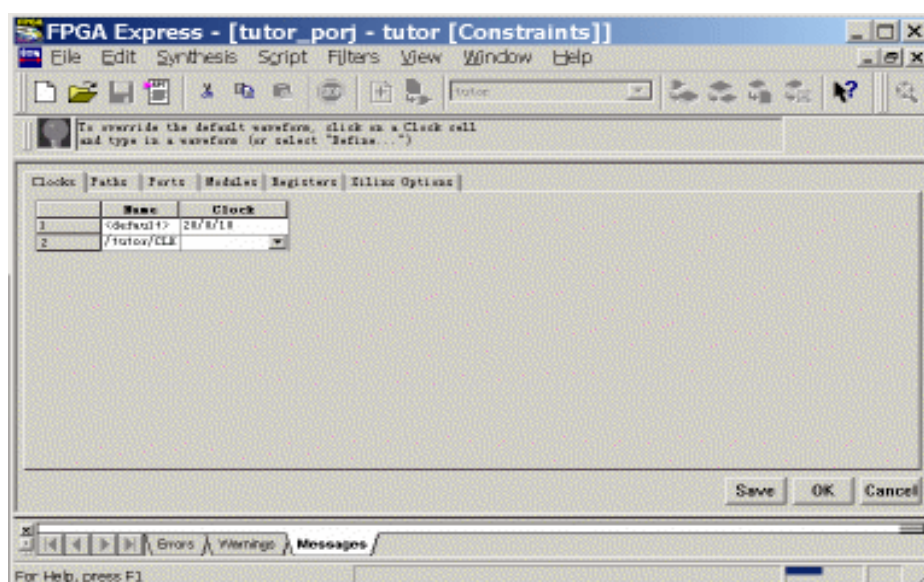


图4-61 约束编辑器

- (2) 设置时钟约束：第一个选项便是 clock，因为全局的时钟是最重要的约束条件。时钟周期的格式为 Period/Rise/Fall，度量单位是 ns。本例使用默认约束，默认约束是 20/0/10，频率为 50MHz。
- (3) 设置路径约束：使用路径约束来指定不同的时序分组之间的时间约束，在 FE 中将具有相同时序特性的端口或者寄存器的集合叫做一个时序分组。例如全部的输入端口和双向端口属于一个相同的时序分组；全部的输出端口和双向端口也属于一个相同的时序分组；使用时钟同一个边沿的全部寄存器属于同一个时序分组；使用同一信号同一电平有效的所有触发器属于同一个时序分组。默认情况下，路径延时使用时钟周期值，本例为“20”。
- (4) 设置多周期时序约束：设计中不是所有路径都使用一个时钟周期完成，有的路径可能需要多个时钟周期才能完成，该部分的运算速度比系统速度慢。为了减小布局布线的压力，可以使用多时钟周期路径将这个路径速度降低，也就是通常所说的慢速例外。例如在一个系统中包含一个高速的接口逻辑，工作在“40MHz”，同时它又有一个 Core 运行在“10MHz”，这个系统主时钟为“40MHz”，在内部分频得到一个“10MHz”的“ena”时钟使能信号，其时序关系如图 4-62 所示。

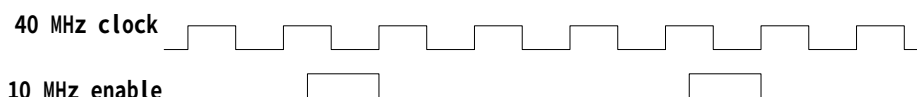


图4-62 时序关系

在 FPGA 内部，“40MHz”时钟作为每一个寄存器的时钟输入，“10MHz”时钟作为时

钟“ena”使能信号。如图 4-63 所示的移位寄存器逻辑中，“reg1”是一个串行输入、并行输出的 4 位寄存器，“reg2”是一个“load”置位有效的保持电路。那么从“reg1”的输出到“reg2”的输入即网线“q”段就是一个多时钟周期的电路。因为只有经历 4 个系统时钟后，“q”才满足系统的要求，而本例 reg-to-reg 的时间是“25”，所以需要放慢速例外处理。

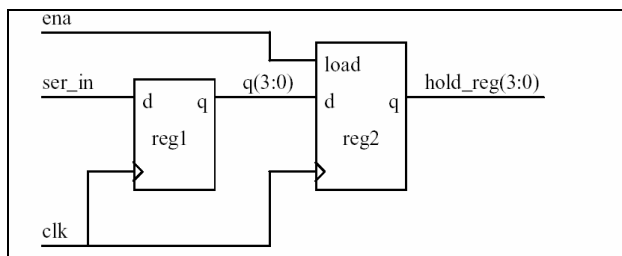


图4-63 移位寄存器逻辑

在路径延时中点击鼠标右键弹出的命令菜单，选择【New Sub path】命令，弹出如图 4-64 所示的慢速例外设计对话框。

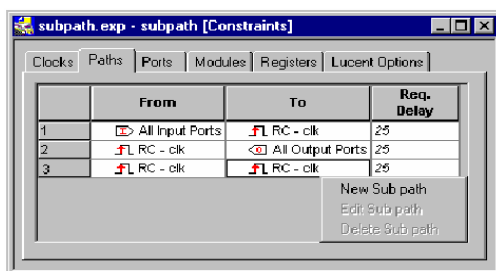


图4-64 慢速例外设计对话框

如需创建慢速例外路径，则需设置子路径名称、延迟时间、路径起点、路径终点等参数，如图 4-65 所示。本例中没有慢速例外，所以不需设置。

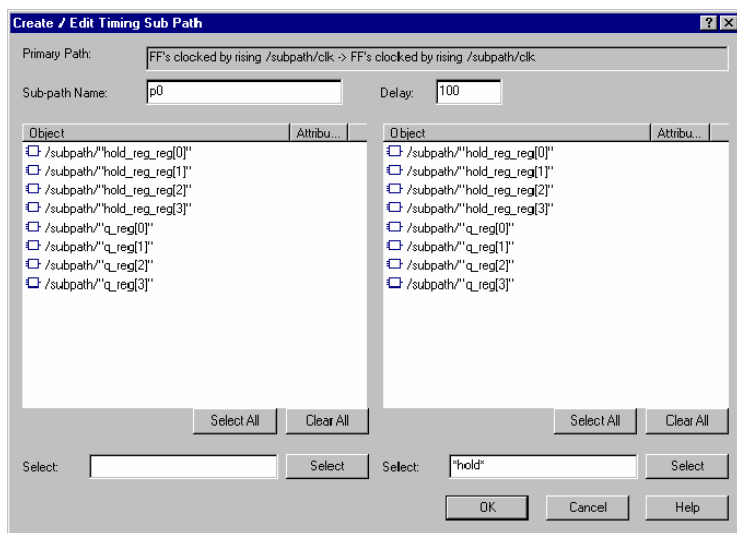


图4-65 慢速例外参数设置对话框

- **端口约束：**在端口约束中可以特殊指定某些端口的输入延时或者输出延时、是否选择全局缓冲（Buffer）、匹配电阻类型、是否使用 IOB 中的寄存器、电平接口标准、引脚锁定等属性。特别需要说明一下全局时钟（Global Buffer）资源的使用方法：在 FE 中默认全局时钟资源使用为自动使用（Automatic）全局时钟，在该状态下，FE 自动根据信号的扇出数目和速度要求，为设计中高扇出、快速信号插入全局时钟资源。如果某个信号虽然扇出很大，但是在 PCB 布线时并没有与全局时钟输入缓冲（IBUFG）引脚连接，则必须指定不使用全局时钟资源，否则在布局布线过程中会报错；反之在 PCB 上与 FPGA/CPLD 的全局输入端口连接了信号，不管该信号是否为时钟信号，在 FE 中都必须指定该信号使用全局时钟资源。本例指定“CLK”信号使用全局资源类为“BUFGP”。
- **模块约束：**在模块约束中可以指定是否保留某些模块的层次，是否优化某些基本原语，是否禁止优化某些模块，是否进行模块分割，是否共享资源，以速度优先还是面积优先为准则进行优化，优化的努力程度的高低，复制寄存器的规则等。在本例中使用默认约束。
- **寄存器约束：**在寄存器约束中，可以约束寄存器的最大扇出。
- **Xilinx 器件特有选项：**在“GSR”映射时，是否忽略没有连接的单元，是否在高扇出的网线上自动插入全局缓冲。在本例使用默认选项。

添加完这些约束以后，单击 **Save** 按钮保存，然后单击 **OK** 按钮退出约束编辑器，回到工程管理主窗口。

4.2.2.8 优化设计

优化步骤以细化步骤的输出网表作为输入，根据细化步骤设置的综合优化参数和综合约束文件对设计进行全面优化，并将优化结果映射到选定器件上。可以使用工具栏中的  按钮，或者在 Chips 窗口中单击鼠标右键弹出命令菜单，选择【Optimize Chip】命令启动设计优化过程，如图 4-66 所示。

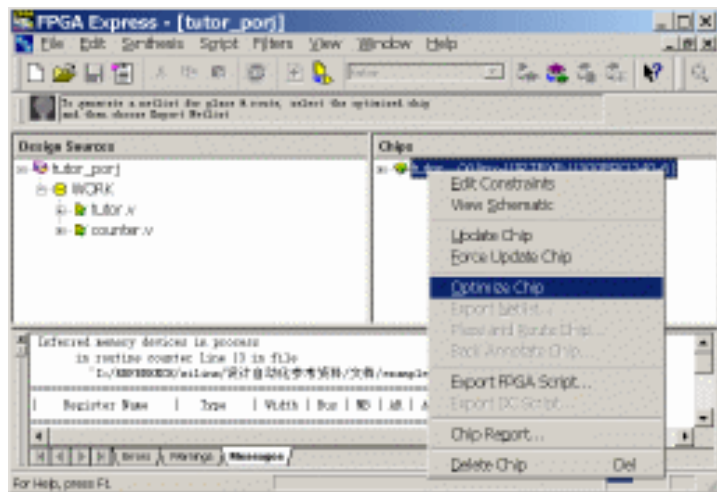


图4-66 优化设计方法

4.2.2.9 观察设计视图

在 FE 中，当编译和优化两个阶段完成后都可以观察设计视图，但是两者的目标不同。观察编译后视图的目的是验证编译结果是否与设计要求一致；而优化后进行的视图观察，是为了进行时序分析，从而改进设计或者约束来提高设计的时序性能。两个阶段进行示意图观察的操作方法基本相同。

在 Chips 窗口，使用鼠标右键点击目录树，在弹出的菜单中选择【View Schematic】命令，就可以启动示意图观察器。

时序追踪指使用时序追踪器来检验设计中某个特殊路径的时序结果，包括关键路径、Cell 类型、延时、扇入、扇出和 slack 等。使用方法如下：

- (1) 在 Chips 窗口中打开示意图观察器，启动时序追踪。程序会打开示意图观察窗、约束编辑器窗口和时序路径窗口等 3 个不同的窗口。
- (2) 点击约束编辑器中的路径约束标签，出现路径约束表格。
- (3) 在路径组窗口中点击所要观察的路径组，这一个组内的所有信号都在下面的表格中列出。选择其中的某个网线，它所经过的路径延时在延时窗口中显示，同时左面窗口中显示整条路径所经过的 Cell、每一个 Cell 的延时和扇出等信息，如图 4-67 所示。
- (4) 将鼠标停留在时序路径窗口中的 Cell 上，就会显示这个 Cell 的名称、类型、延时和扇出等信息。

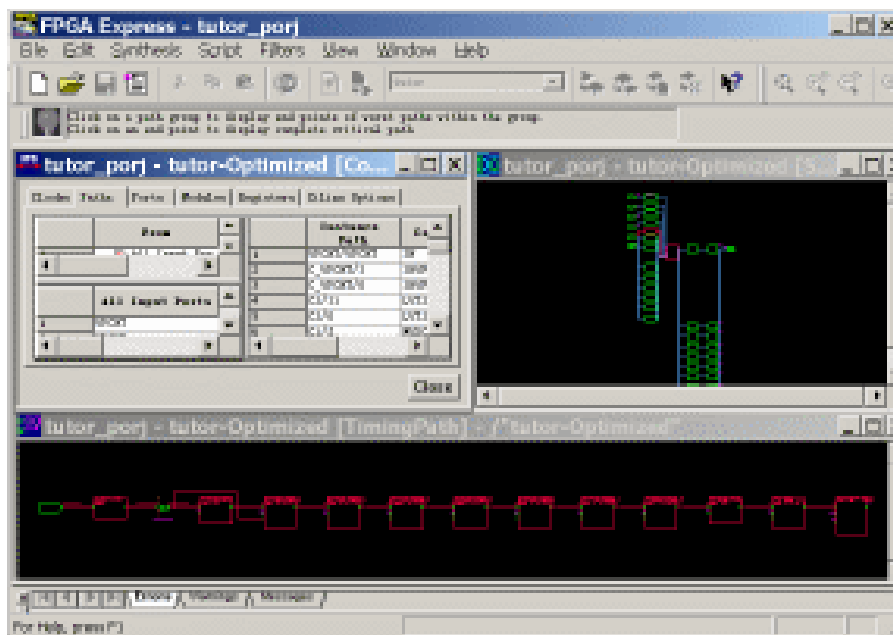


图4-67 示意图观察窗

4.2.2.10 生成网表

FE 可以给器件商的布局布线工具提供 EDIF 或者 XNF 格式的网表文件，甚至可以直接

启动布线器进行布局布线流程。FE 也可以生成 VHDL 或者 Verilog 的仿真文件，进行综合后仿真。

- (1) 用鼠标右键激活 Chips 窗口中经过优化后的目录树，然后单击工具栏中的按钮，或者从弹出菜单中选择【Export Netlist】命令。弹出输出网表对话框，如图 4-68 所示。

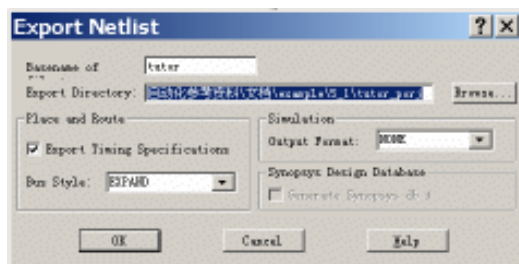


图4-68 生成网表对话框

- (2) 在对话框中可以改变网表的名称、存储网表的路径。
- (3) 选择是否为布局布线工具导出时序约束文件，如果选择导出时序约束，那么会将综合时的各种约束生成 NCF 文件提供给布局布线器。NCF 文件格式描述见本书 5.6.2 小节。
- (4) 选择总线类型：EXPAND 为默认值，这也是最常用的一种格式，即顶层的总线被展开成独立的 I/O。也可以选择不同的总线类型，其表示方法为 %s[%d : %d]，其中 %s 代表总线名称，%d 代表总线上标和下标。各种总线的区别在于使用的括号不同。需要注意的是：在编写用户约束文件时，总线标志符必须和输出网表时选择的总线类型一致，默认参数输出的网表的总线标志符为“尖括号”“<>”。本例使用默认参数。
- (5) 仿真文件格式：如果需要做综合后仿真，需要选择相应的仿真文件格式。FE 可以自动生成 VHDL 或者 Verilog 格式的仿真文件。

4.2.2.11 生成报告

可以使用工具栏中的工具来产生各种报告，也可以在 Chips 窗口中使用右键菜单的【Reports】命令，产生各个阶段的报告文件。

4.2.3 FST 操作说明

FST (FPGA Script Tool) 是使用 FE 的另外一种方式，它是基于 TCL 语言，通过命令行调用 FE 的综合优化工具，完成 FE 的综合流程，该方式在 UNIX 系统中很常用。使用这种方式一般基于两种原因：第一是某些用户习惯了使用命令行操作，这类用户的很多设计都是在 UNIX 环境下开发的，感觉命令比图形界面更加直接，就执行效率而言，批处理命令的确比图形界面下操作高效，而且使用脚本执行综合过程，不会因为反复而遗漏了某个选项或者操作环节。所以为了追求速度和操作的一致性，可以选择使用 FST，虽然编写 FST 脚本比较复杂，但是却可以一劳永逸；另一方面，FST 使用的语言与 Design Compiler 的脚本命令

基本相同，可以很方便地将 FE 的脚本转换成 DC (Design Compiler, Synopsys ASIC 设计的重要工具之一) 命令脚本使用。

🔑 FST 的基本使用方法，参考示例详见本书附带光盘的“Example-4-3”目录

1. 设置变量，命令如下：

```
set proj am2910_proj
set top AM2910
set target XC4000E
set Chip AM2910
set export_dir export_dir
```

命令 `set proj am2910_proj` 的作用是将变量 `proj` 的值设置为 `am2910_proj`，方便在以后脚本中的调用，同时将这些变量集中在一起，便于维护和修改。使用 `man set` 查看帮助，学习 `set` 命令的详细用法。

2. 删除原有的同名工程项目，建立新的工程项目，命令如下：

```
exec cmd /c rd /q /s am2910_proj
create_project -dir . $proj
```

命令 `exec cmd /c rd /q /s am2910_proj` 的作用是使用 Windows 内置命令 `rd` 删除原有的 `am2910_proj` 工程项目，`exec` 为 tcl 内置命令。

命令 `create_project -dir . $proj` 的作用是在当前目录下创建名为 `$proj` 的工程项目，而变量 `proj` 在前面已经设置为 `am2910_proj`，所以执行这个命令后在当前目录下会出现名为 `am2910_proj` 的工程目录。其中包括一个工程文件和中间转换需要的一个子目录。

3. 打开当前工程项目，命令如下：

```
open_project $proj
```

执行这个命令可以打开一个现存的工程作为当前的工程。

4. 设置工程项目变量，命令如下：

```
proj_export_timing_constraint = "yes"
```

这个命令说明在这个工程中将导入一个时序约束文件。

5. 设置缺省时钟频率，命令如下：

```
default_clock_frequency = 45
```

6. 创建库，添加库文件，命令如下：

```
create_library LIB
add_file -library LIB -format VHDL am2910/lib.vhd
```

这个命令是创建 `LIB` 库，添加 `VHDL` 格式的库文件到 `LIB` 库中。

7. 添加设计源文件，命令如下：

```
add_file -format VHDL am2910/am2910.vhd
add_file -format VHDL am2910/bts4.vhd
add_file -format VHDL am2910/control.vhd
```

```
add_file -format VHDL am2910/regcnt.vhd
add_file -format VHDL am2910/stack.vhd
add_file -format VHDL am2910/upc.vhd
add_file -format VHDL am2910/y.vhd
```

8. 分析源文件，命令如下：

```
analyze_file -progress
```

这个命令是分析工程项目中添加的所有源文件，参数 `progress` 表示在分析的同时可以在控制台显示分析结果。可以使用参数 `force` 对源文件进行强制分析，这时可以对项目中已经分析过的文件进行强制分析。

9. 创建 Chip，命令如下：

```
create_Chip -progress -target $target -name $Chip $stop
```

这个命令创建了一个名为 `$Chip`、顶层文件为 `$stop`、选择的目标器件为 `$target` 的网表文件。其中 `progress` 参数表示在执行过程中显示执行结果。

10. 设置当前 Chip，添加约束文件，命令如下：

```
current_Chip $Chip
source constraints.fst
```

可以对所选择的 Chip 进行约束，约束使用类似的 TCL 语言编写在 `constraints.fst` 文件中。

11. 优化当前的 Chip，命令如下：

```
set opt_Chip [format "%s-Optimized" $Chip]
optimize_Chip -progress -name $opt_Chip
```

设置优化 Chip 的目录以及名称，进行优化操作。

12. 显示过程中的错误和警告信息，命令如下：

```
list_message
```

13. 创建时序报告，命令如下：

```
report_timing
```

14. 输出布局布线使用的网表和约束，命令如下：

```
exec cmd /c rd /s /q $export_dir
exec cmd /c mkdir $export_dir
export_Chip -progress -dir $export_dir
```

15. 关闭工程项目，命令如下：

```
close_project
```

编写约束脚本文件

在上面的控制脚本中调用了约束脚本文件，在一般情况下，需要对设计添加一些约束条件，在用户图形接口中，都是通过 UCF 或者约束编辑器对设计进行约束的。在命令行中，通过约束脚本文件对设计添加约束。

1. 设置时钟波形，命令如下：

```
set_clock -period 20 -rise 0 -fall 10 CLOCK
```

指明设计中的时钟信号并设置时钟周期以及占空比。

2. 创建子路径，命令如下：

```
create_subpath -from_name my_from -to_name my_top -from_list  
/AM2910/U1\STACK_TOP_reg<3> -to_list /AM2910/U1\STACK_TOP_reg<1> -maxd  
10 (RC,CLOCK):(RC,CLOCK)
```

3. 确定路径延时，命令如下：

```
set_max_delay -path my_from:my_to 77
```

4. 设置输入延时，命令如下：

```
set_input_delay -group "(I)" 21 "/AM2910/D<1>"
```

5. 设置输出延时，命令如下：

```
set_output_delay -group "(O)" 23 "/AM2910/Y_OUTPUT<1>"
```

6. 设置与默认组相关的输出延时，命令如下：

```
set_output_delay 24 "/AM2910/Y_OUTPUT<2>"
```

7. 设置保留层次，命令如下：

```
set_module_primitive preserve "/AM2910/U4"
```

4.3 Xilinx 内嵌的综合工具——XST

本节介绍 ISE 内嵌的综合工具 XST 的使用方法。

4.3.1 XST 综述

XST (Xilinx Synthesis Technology) 是 Xilinx ISE 内嵌的综合工具。虽然 XST 与 Synplify Pro 等业界流行的综合工具相比特点并不突出，功能也不全面，但是 Xilinx 对自己的 FPGA/CPLD 内部结构最为了解，所以 XST 对 Xilinx 器件的支持也最为直接，更重要的是 XST 内嵌在 ISE 中，安装 ISE 后可以直接使用，不需另外付费。

XST 的输入文件一般是 HDL 源文件，在 ISE 高级版本中，XST 已经支持 Verilog 和 VHDL 混合语言源代码输入；XST 的输出文件是 NGC 网表，XST 的报告文件是 Log 文件。ISE 中 XST 设计流程的综合阶段约束文件与实现阶段约束文件的概念并不分明，综合阶段的约束条件常常通过实现阶段的约束文件来完成。XST 的综合约束文件是 XCF (XST Constraint File)，而布局布线阶段最重要约束文件是用户约束文件 UCF (User Constraint File)，两者有着千丝万缕的联系，UCF 几乎支持 XCF 的所有约束语言与命令。通常在使用 XST 综合流程时，仅仅通过综合属性设置来设置全局性的综合策略与参数，细化的约束是通过实现阶段的约束文件 UCF 完成的。

本节主要讲述 XST 综合流程与综合属性参数设置，至于 Xilinx 的约束概念、约束语言和命令以及 UCF、XCF、NCF、PCF 等不同类型的约束文件的设计将在第 5 章详细论述。

XST 综合主要分为以下 3 个步骤：

- **HDL 源代码分析**

此步骤 XST 的主要功能为完成对 HDL 代码的语法检错。

• HDL 代码综合

这是 XST 综合最主要的一个步骤，此步骤中 XST 完成宏的标识（Macro Recognition）、有限状态机的扩展（FSM Extraction）和资源共享（Resource Sharing）等功能。所谓宏的标识，是指 XST 在 HDL 的编译结果中搜寻 Xilinx 器件的底层硬件原语与宏的过程。直接将硬件原语映射到器件的底层宏单元可以大大缩短综合时间并优化综合结果，这个道理和前面论述的行为级综合技术一致。ISE 6 及以上软件版本内嵌的 XST 综合引擎支持增量式综合。

• 底层优化

此步骤中，XST 的主要功能为宏的实现（Macro Implementation）、时序优化（Timing Optimization）、查找表的映射（LUT Mapping）和寄存器复制（Register Replication）等。需要补充说明的是 XST 综合工具在综合 FPGA 器件时也是采用时序驱动（Timing Driven）引擎的，而在综合 CPLD 器件时并非采用时序驱动引擎，而是以尽量减少逻辑层次和硬件单元数目为目标。这是由 FPGA 和 CPLD 器件的内部结构不同决定的。

4.3.2 XST 综合属性设置

XST 的基本使用方法非常简单：在 Project Navigator 工程属性选项设置中指定采用 XST Verilog 或 XST VHDL 设计流程，然后设置综合属性参数，进行综合即可。在 Project Navigator 设置工程属性与设计流程在 2.1 节已经讨论过，但是 XST 的综合属性参数设置根据选用器件类型（FPGA/CPLD）和选用语言（Verilog/VHDL）不同而略有差别。

图 4-69 和图 4-70 分别是基于 FPGA 设计的 XST Verilog 和 XST VHDL 设计流程综合属性设置对话框，基于 CPLD 的设计该综合属性设置页面略有不同。

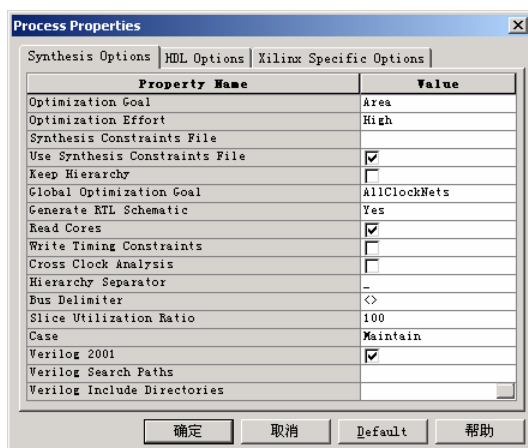


图4-69 XST Verilog 综合属性设置对话框

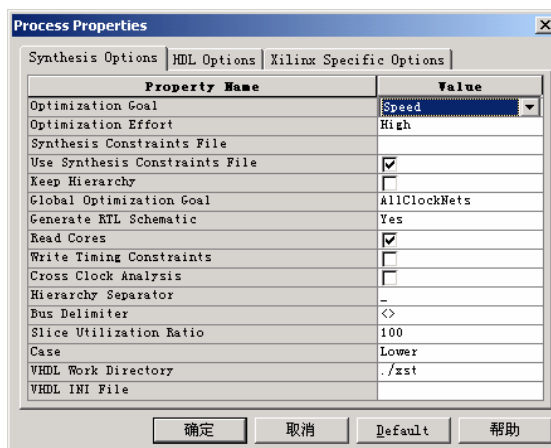


图 4-70 XST VHDL 综合属性设置对话框

综合属性设置对话框由综合参数设置、HDL 源代码参数设置和 Xilinx 专用参数设置等 3 个选项卡组成。下面依次讲解这 3 个选项卡在不同器件类型（FPGA/CPLD）和不同语言（Verilog/VHDL）下的参数含义及其使用方法。

1. 综合参数设置 (Synthesis Options)

综合参数设置选项卡主要用于设置一些与综合的全局目标和整体策略相关的参数。

- **综合目标 (Optimization Goal)**
该参数用以指定优化的全局目标是面积 (Area) 优先还是速度 (Speed) 优先。由于时序要求一般来说比器件面积要求更重要，所以一般选择速度优先策略。
- **综合难度 (Optimization Effort)**
该参数用于指定综合器工作的工作难度，如果设计时序和面积要求较高，选择难度指标为 “High”。
- **综合约束文件 (Synthesis Constraint File)**
该选项用于指定 XST 综合约束文件 XCF 文件的路径。
- **使用综合约束文件 (Use Synthesis Constraint File)**
指定综合文件路径后，该选项打勾，使综合约束文件有效。
- **保留层次 (Keep Hierarchy)**
为了达到更好的优化效果，XST 默认将设计层次打平，选择该参数有效，则 XST 保留设计层次。保留设计层次虽然会在较小程度上影响综合效果，但是如果保留设计层次，则在进行综合后仿真与综合结果分析中，在信号繁多的网表中查找待分析信号、端口、模块就会变得容易得多。
- **全局优化目标 (Global Optimization Goal)**
该参数仅对 FPGA 有效，用于指定寄存器之间，输入引脚到首级寄存器之间，末级寄存器到输出引脚之间，输入引脚到输出引脚之间的优化策略。包含的优化属性有 ALLCLOCKNETS、OFFSET_IN_BEFORE、OFFSET_OUT_AFTER、INPAD_TO_OUTPAD 和 MAX_DELAY，具体含义参见第 5 章。
- **生成寄存器传输级原理图 (Generate RTL Schematic)**
该参数用以将综合结果生成 RTL 视图，作用和 Synplify Pro 中的 “RTL View” 十分相似。除了在综合参数设置对话框可以指定生成 RTL 视图外，在当前资源操作窗中综合 (Synthesis) 步骤中也有相应的命令用以生成 RTL 视图。
- **读核模块指示 (Read Cores)**
该参数仅对 FPGA 有效，用以指定 XST 是否预分析 EDIF/NGC 网表格式的核的时序约束和面积利用率等信息。
- **写时序约束条件 (Write Timing Constraints)**
该参数仅对 FPGA 有效，用以指定在 HDL 源代码或是 XST 约束文件中指定的时序约束关系是否对 Xilinx 的 NGC 网表有效。
- **时钟关联分析 (Cross Clock Analysis)**
该参数用以指定 XST 是否进行内部时钟分组优化与分析，默认为不进行分析。
- **层次分隔符 (Hierarchy Separator)**

XST 默认状态为将设计的层次打平，使用此处指定的符号区分层次之间的关系。

- **总线标志符 (Bus Delimiter)**
此处指定的符号用以表示总线上下有效界限。
- **Slice 结构利用率 (Slice Utilization Ratio)**
该参数仅对 FPGA 有效。前面已经反复强调过，设计的速度与面积是对立统一的矛盾体，有时综合器为了优化时序，会占用更多的芯片面积。该参数用以指定在进行时序优化过程中，不能超过的芯片面积占用率。需要说明的是，如果综合优化过程发现在指定面积利用率条件下无法满足时序要求，则根据“时序第一位，面积第二位”的优化原则，该面积利用率无效。
- **大小写指定 (Case)**
该参数用以指定模块实例名称和网线名称用大写还是小写字母表示。
- **VHDL 库文件映射路径 (VHDL Work Directory)**
该参数仅对 VHDL 设计类型有效，用以指定 VHDL 库文件的映射路径。
- **VHDL INI 文件 (VHDL INI File)**
该参数仅对 VHDL 设计类型有效，用以指定 VHDL INI 文件的路径。
- **Verilog 环境路径 (Verilog Search Paths)**
该参数仅对 Verilog 设计类型有效，用以指定搜索 Verilog 源文件的路径。
- **Verilog 库文件目录 (Verilog Include Directories)**
该参数仅对 Verilog 设计类型有效，用以指定引入 Verilog 库文件的路径。

2. HDL 源代码参数设置 (HDL Options)

HDL 源代码参数设置选项卡主要指定与源代码规则和编译类推方式相关的属性，前面 5 个属性对 FPGA 和 CPLD 设计都有效，后面的属性仅对 FGPA 设计有效。

- **有限状态机编码方式 (FSM Encoding Algorithm)**
该参数用以指定有限状态机的编码方式为独热码 (One-Hot)、格雷码 (Gray) 等，默认选项为 XST 自动选择编码方式 (Auto)。
- **条件语句实现方式 (Case Implementation Style)**
该参数用以指定“case”语句的编译方式为全条件“FULL_CASE”还是并行条件“PARALLEL_CASE”。
- **选择器宏的扩展方式 (Mux Extraction)**
该参数用以指定选择器 (Mux) 宏的扩展方式，默认为 XST 自动选择合适的编译方式。
- **资源共享 (Resource Sharing)**
该参数用以指定是否允许 XST 复用一些运算模块。
- **带有复杂使能信号时钟单元的扩展方式 (Complex Clock Enable Extraction)**
XST 的综合策略一般将时序电路综合为带有时钟使能信号的时序单元。如果选择本参数有效，XST 在综合带有时钟使能信号的时序模块时会考虑更多情况，将那些不明显的情况也尽量使用带有时钟使能的时序模块。

- **RAM 的扩展方式 (RAM Extraction)**
该参数仅对 FPGA 有效，用以指定是否使用 RAM 宏单元。
- **RAM 实现类型 (RAM Style)**
该参数仅对 FPGA 有效，用以指定使用 RAM 的类型为块 RAM (Block RAM) 还是分布式 RAM (Distributed RAM)。
- **ROM 的扩展方式 (ROM Extraction)**
该参数仅对 FPGA 有效，用以指定是否使用 ROM 宏单元。
- **选择器实现类型 (Mux Style)**
该参数仅对 FPGA 有效，用以指定使用选择器 (Mux) 宏的类型。
- **译码器的扩展方式 (Decoder Extraction)**
该参数仅对 FPGA 有效，用以指定是否使用译码器 (Decoder) 宏单元。
- **带有优先级的编码器的扩展方式 (Priority Encoder Extraction)**
该参数仅对 FPGA 有效，用以指定是否使用带有优先级的编码器 (Priority Encoder) 宏单元。
- **移位寄存器的扩展方式 (Shift Register Extraction)**
该参数仅对 FPGA 有效，用以指定是否使用移位寄存器 (Shift Register) 宏单元。
- **逻辑移位器的扩展方式 (Logical Shifter Extraction)**
该参数仅对 FPGA 有效，用以指定是否使用逻辑移位器 (Logical Shifter Extraction) 宏单元。
- **异或单元合并方式 (XOR Collapsing)**
该参数仅对 FPGA 有效，用以指定相关联的异或 (XOR) 单元是否合并为一个大的异或宏实现。
- **乘法器实现类型 (Multiplier Style)**
该参数仅对 FPGA 有效，用以指定是否使用乘法器 (Multiplier) 宏单元。在 Virtex-II 和 Virtex-II Pro 器件族中甚至可以手动指定乘法器宏单元的类型为块乘法器 (Block Multiplier) 还是查找表 (LUT) 乘法器。

3. Xilinx 专用参数设置 (Xilinx Specific Options)

这个选项卡用以设置与 Xilinx 专用结构相关的综合属性。

- **添加 I/O 缓冲 (Add IO Buffers)**
该参数对 FPGA/CPLD 都有效，用以指定是否对所综合的模块插入 I/O 缓冲 (I/O Buffer)。
- **等效寄存器删除 (Equivalent Register Removal)**
该参数对 FPGA/CPLD 都有效，用以指定是否把寄存器传输级功能等效的寄存器删除，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 (Primitive) 指定的，将不会被删除。
- **乘法器实现类型 (Multiplier Style)**
该参数仅对 FPGA 有效，用以指定编译类推时使用的乘法器 (Multiplier) 宏单

元的类型为块乘法器 (Block Multiplier) 还是查找表 (LUT) 乘法器。在 Virtex-II 和 Virtex-II Pro 器件族中甚至可以手动地指定乘法器宏单元的类型。

- **最大扇出数 (Max Fanout)**
该参数仅对 FPGA 有效, 用以指定信号和网线的最大扇出数目。
- **全局时钟资源数目 (Number of Clock Buffers)**
该参数仅对 FPGA 有效, 用以指定在该型号 FPGA 中可用的全局时钟资源 (BUFGP) 的数目。
- **增量编译 (Incremental Synthesis)**
该参数仅对 FPGA 有效, 用以指定是否采用增量综合技术。在 ISE 5 及后续版本中, 对增量综合与增量实现技术的支持越来越强大。使用增量综合与实现技术, 可以更好地集成以往的设计成果, 缩短综合与实现所耗费的时间, 提高设计效率。
- **寄存器复制 (Register Duplication)**
该参数仅对 FPGA 有效, 用以指定是否允许进行寄存器复制。对高扇出和时序比较紧张的寄存器进行复制, 减少驱动缓冲 (Buffer) 的数目与逻辑级数, 能部分缓解时序紧张的状态, 提高设计频率。
- **寄存器平衡 (Register Balancing)**
该参数仅对 FPGA 有效, 用以指定是否允许平衡寄存器。这个参数和 Synplify Pro 的 “Retiming” 参数命令十分相似。采用寄存器平衡技术, 可以改善某些设计的时序条件, 提高设计频率。
- **首级寄存器属性 (Move First Stage)**
该参数仅对 FPGA 有效, 用以指定在采用 “Retiming” 技术改善时序条件时自输入的第一级寄存器的属性。
- **末级寄存器属性 (Move Last Stage)**
该参数仅对 FPGA 有效, 用以指定在采用 “Retiming” 技术改善时序条件时到输出的最后一级寄存器的属性。
- **Slice 时序优化 (Slice Packing)**
该参数仅对 FPGA 有效, 用以指定是否允许采用 Slice Packing 时序优化技术。Slice Packing 将关键路径的查找表 (LUT) 逻辑尽量布置在同一个 Slice 或者 CLB 模块中, 从而缩短了 LUT 之间的布线, 是提高设计频率的一项时序优化技术。
- **I/O 的寄存器是否使用 IOB 内部寄存器实现 (Pack I/O Registers into IOBs)**
该参数仅对 FPGA 有效, 用以指定是否将设计逻辑中的第一级寄存器用 IOB 内部的寄存器实现。事实上, Xilinx FPGA 的 IOB 内部都有一级寄存器, 如果将设计的第一级寄存器用 IOB 内部的寄存器实现, 可以缩短 IO PAD (引脚) 到寄存器之间的路径, 缩短时延约 1~2ns, 在高速 I/O 设计中是一个重要的时序优化参数。
- **时钟使能信号实现方式 (Clock Enable)**
该参数仅对 CPLD 有效, 用以指定如何实现 CPLD 的时钟使能信号。如果

CPLD 设计中使用了时钟使能信号，通过设置该参数指定时钟使能信号是采用专用结构还是等效替代结构来实现。

- 是否保留宏单元 (Macro Preserve)
该参数仅对 CPLD 有效，用以指定是否保留宏单元。该参数有效时，综合时即使选择了层次打平选项，也会保留宏单元结构。
- 是否保留异或宏单元 (XOR Preserve)
该参数仅对 CPLD 有效，用以指定将异或宏单元层次打平还是保留异或宏单元结构。
- WYSIWYG
该参数仅对 CPLD 有效，用以指定是否保留所有用户指定信号和模块。

4.3.3 使用 XST 的综合流程

本节用一个简单的实例演示使用 XST 综合的过程。使用 XST 的综合过程主要分为选择 XST 综合流程，设置综合参数，分析综合报告和观察 RTL 视图等步骤。

✎ 使用 XST 综合“4bit 计数器”，参考示例详见本书附带光盘的“Example-4-4”目录

该设计用 Verilog 和 VHDL 两种语言描述，读者可根据个人习惯选用一种类型的源代码。

1. 创建一个工程，利用源代码编辑器 (HDL Editor) 的语言模板 (Language Template) 生成一个带有异步复位和同步置位的 4bit 同步计数器。根据源代码类型设置工程属性为 XST Verilog 或 XST VHDL 设计流程，器件选择为任何一种 FPGA 器件，如图 4-71 所示。

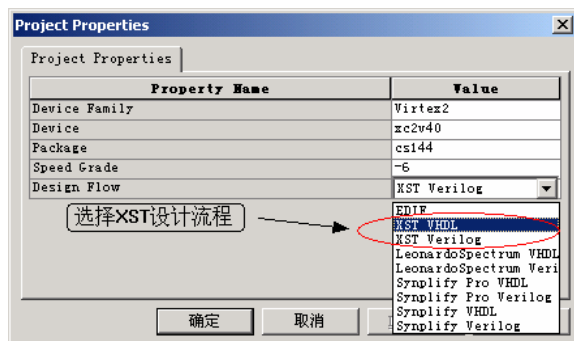


图4-71 选择 XST 设计流程

2. 在工程资源窗 (Sources in Project) 选中顶层文件，然后在当前资源操作窗 (Processes for Current Source) 选中综合步骤 (Synthesize)，单击  按钮，编辑操作步骤属性。XST 综合属性的设置包括综合参数设置，HDL 源代码参数设置，Xilinx 专用参数设置，按照图 4-72~图 4-74 所示分别设置。

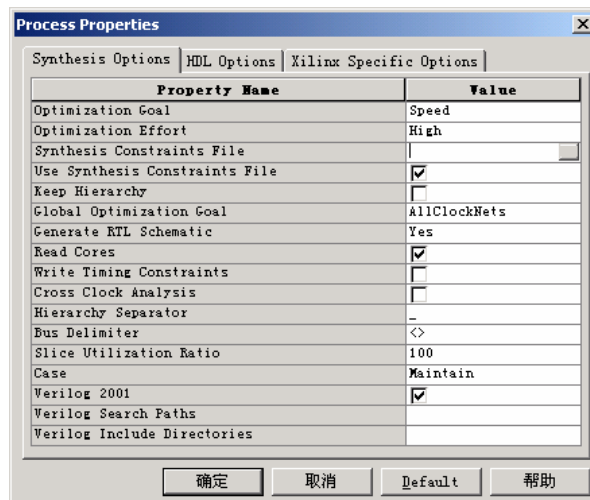


图4-72 设置 XST 综合参数

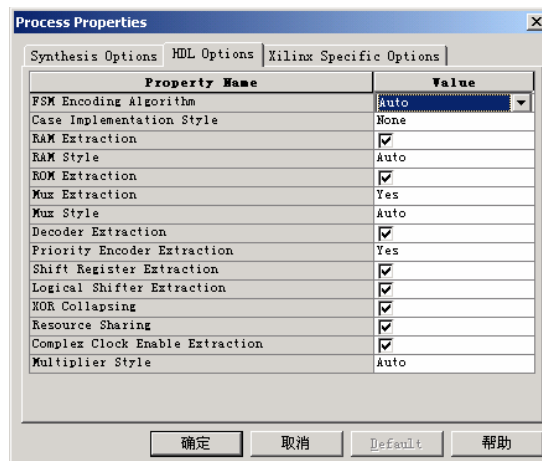


图4-73 设置 XST HDL 源代码参数

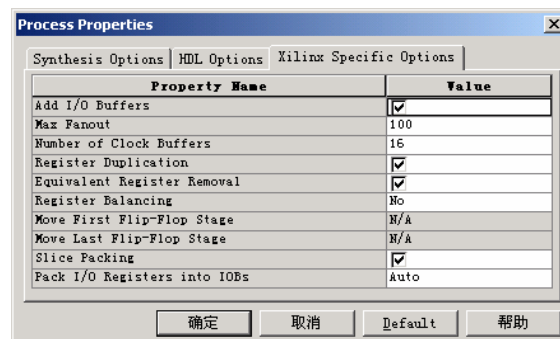


图4-74 设置 XST Xilinx 专用参数

3. 单击  按钮保存综合参数设置。在工程资源窗选中顶层文件，然后在当前资源操作窗选中综合步骤，单击  按钮，运行综合。
4. 完成 XST 综合后，展开当前资源操作窗的综合步骤，双击“View Synthesis Report”图标观察综合报告，双击“View RTL Schematic”图标观察寄存器传输级原理图，分析综合结果，如图 4-75 所示。

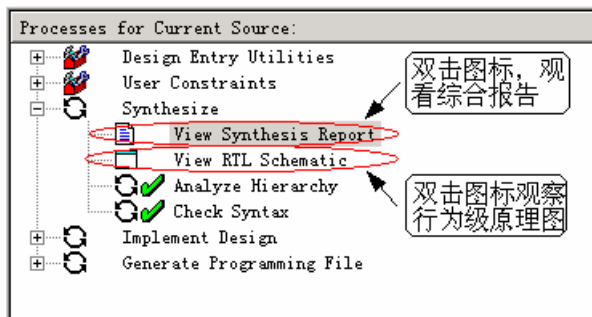


图4-75 综合步骤完成后，分析综合结果

5. XST 产生的寄存器传输级视图功能和 Synplify Pro 的“RTL View”相似，但是没有 Synplify Pro 的“RTL View”功能强大。双击“View RTL Schematic”图标，ISE 自动调用原理图编辑器 ECS 来打开综合产生的寄存器传输级视图，如图 4-76 所示。

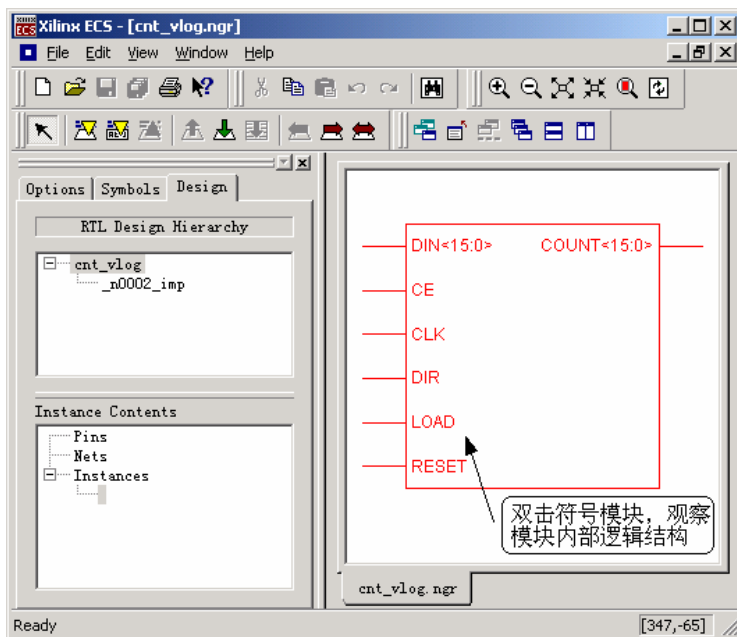


图4-76 XST 综合产生的寄存器传输级视图模块符号

6. 双击模块符号，观察模块内部逻辑结构，如图 4-77 所示。

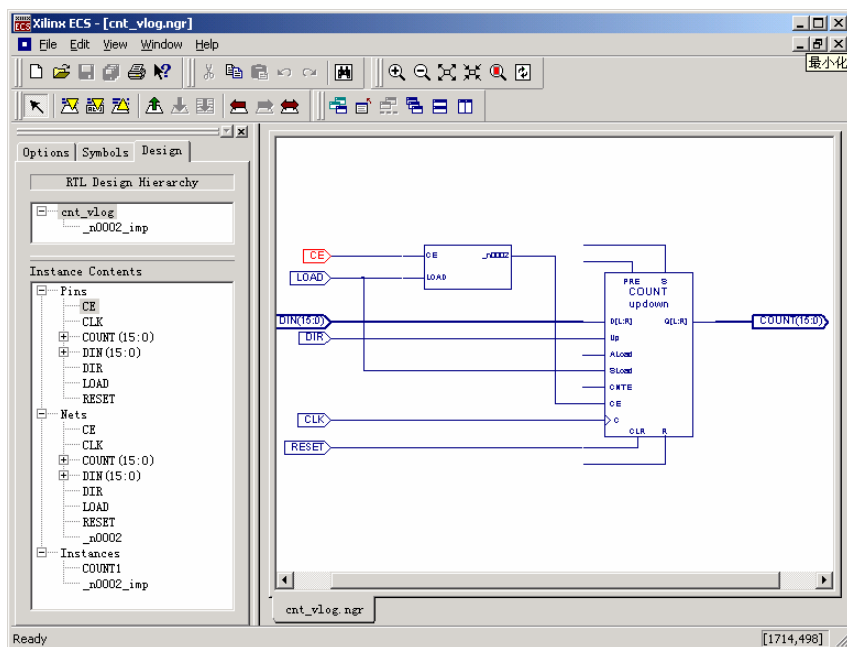


图4-77 模块内部逻辑结构

4.4 全局时钟与第二全局时钟资源

本节重点介绍“全局时钟与第二全局时钟资源”的概念和基本使用方法。这个概念在第4章综合工具、第5章约束、第6章实现工具中都有涉及，是FPGA/CPLD同步设计的一个重要概念。如果善加利用全局时钟与第二全局时钟资源，能在一定程度上改善设计的综合、实现效果；如果使用不当，不但会影响设计的工作频率、稳定性等性能指标，甚至会导致设计的综合、实现过程出错。

4.4.1 全局时钟资源简介

目前大型设计一般推荐使用同步时序电路。同步时序电路基于时钟触发沿设计，对时钟的周期、占空比、延时、抖动提出了更高的要求。为了满足同步时序设计的要求，一般在FPGA/CPLD设计中采用全局时钟资源驱动设计的主时钟，以达到最低的时钟抖动和延迟。在Xilinx FPGA/CPLD中的全局时钟资源一般使用全铜层工艺实现，并设计了专用时钟缓冲与驱动结构，从而使全局时钟到达芯片内部的所有可配置单元（CLB）、I/O单元（IOB）和选择性块RAM（Block Select RAM）的时延和抖动都为最小。为了适应复杂设计的需要，Xilinx的FPGA/CPLD中集成的专用时钟资源与数字延迟锁相环（DLL）的数目不断增加，最新的Virtex-II器件最多可以提供16个全局时钟输入端口和8个数字时钟管理模块（DCM）。

4.4.2 常用的与全局时钟资源相关的 Xilinx 器件原语

常用的与全局时钟资源相关的 Xilinx 器件原语包括 IBUFG、IBUFGDS、BUFG、BUFGP、BUFGCE、BUFGMUX、BUFGDLL 和 DCM 等。

- (1) 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 IBUFG 单元，否则在布局布线时会报错。IBUFG 支持 AGP、CTT、GTL、GTLP、HSTL、LVCMOS、LVDCI、LVDS、LVPECL、LVTTL、PCI、PCIX、SSTL 等多种格式的 IO 标准。图 4-78 所示为 IBUFG 示意图。

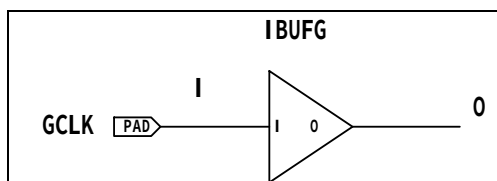


图4-78 IBUFG 示意图

- (2) 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。IBUFGDS 支持 BLVDS、LDT、LVDSEXT、LVDS、LVPECL、ULVDS 等多种格式的 IO 标准。图 4-79 所示为 IBUFGDS 示意图。

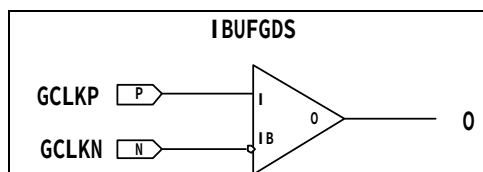


图4-79 IBUFGDS 示意图

- (3) BUFG 即全局缓冲，它的输入是 IBUFG 的输出，BUFG 的输出到达 FPGA/CPLD 内部的 IOB、CLB、Block Select RAM 的时钟延迟和抖动最小。图 4-80 所示为 BUFG 示意图。

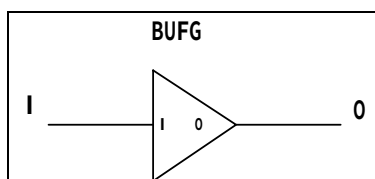


图4-80 BUFG 示意图

- (4) BUFGCE 是带有时钟使能端的全局缓冲。它有一个输入 I、一个使能端 CE、一个输出端 O。仅当 BUFGCE 的使能端 CE 有效（高电平）时，BUFGCE 才有输出。图 4-81 所示为 BUFGCE 示意图。

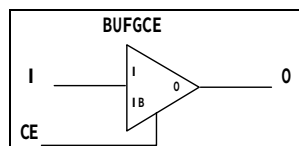


图4-81 BUFGCE 示意图

- (5) 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。图 4-82 所示为 BUFGMUX 示意图。

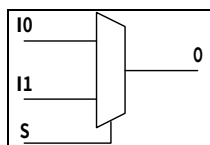


图4-82 BUFGMUX 示意图

- (6) BUFGP 相当于 IBUFG 加上 BUFG。图 4-83 所示为 BUFGP 示意图。

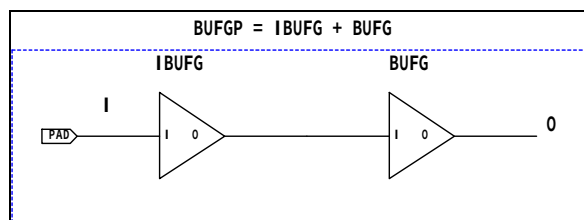


图4-83 BUFGP 示意图

- (7) BUFGDLL 是全局缓冲延迟锁相环，相当于 BUFG 与 DLL 的结合。在早期设计中经常使用，用以完成全局时钟的同步、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。图 4-84 所示为 BUFGDLL 示意图。

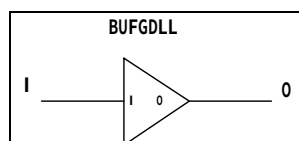


图4-84 BUFGDLL 示意图

- (8) DCM 即数字时钟管理单元，主要完成时钟的同步、移相、分频、倍频、去抖动（skew）等。严格来说 DCM 本身并不是全局时钟资源，但是由于 DCM 与全局时钟有着密不可分的联系，为了达到最小的延迟和抖动，几乎所有的 DCM 应用都要使用全局缓冲资源。DCM 的详细使用方法请参考本书 2.7.2 节和 9.2.5 节。图 4-85 所示为 DCM 模块示意图。

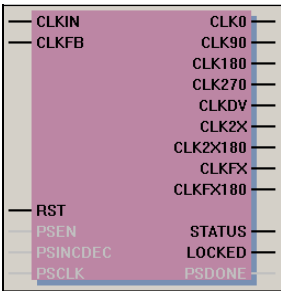


图4-85 DCM 模块示意图

4.4.3 Xilinx 全局时钟资源的使用方法

Xilinx 全局时钟资源的使用方法有以下 5 种：

1. IBUFG + BUFG 的使用方法

IBUFG 后面连接 BUFG 的方法是最基本的全局时钟资源的使用方法，由于 IBUFG 组合 BUFG 相当于 BUFGP，所以在这种使用方法也称为 BUFGP 方法，如图 4-86 所示。

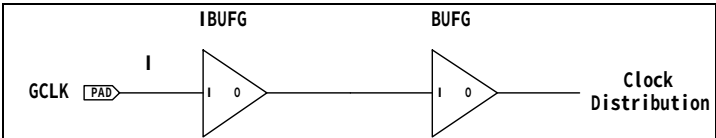


图4-86 IBUFG + BUFG 的使用方法

2. IBUFGDS + BUFG 的使用方法

当输入时钟信号为差分信号时，需要使用 IBUFGDS 代替 IBUFG，如图 4-87 所示。

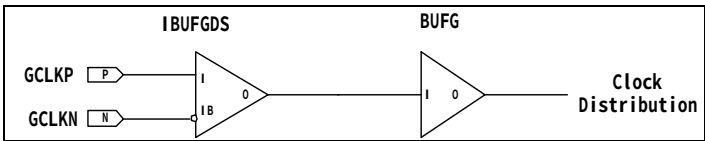


图4-87 IBUFGDS + BUFG 的使用方法

3. IBUFG + DCM+BUFG 的使用方法

这种使用方法最为灵活，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、倍频等变换，而且可以使全局时钟的输出达到无抖动延迟（“0” skew），如图 4-88 所示。

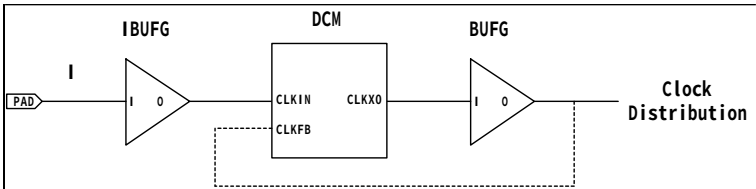


图4-88 IBUFG + DCM+BUFG 的使用方法

4. Logic+BUFG 的使用方法

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 IO 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，也就是说普通 IO 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、Block Select RAM）的延时可以忽略不计为“0”ns，如图 4-89 所示。

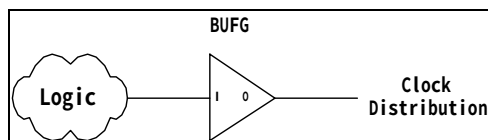


图4-89 Logic + BUFG 的使用方法

5. Logic + DCM+BUFG 的使用方法

DCM 同样也可以控制并变换普通时钟信号。也就是说 DCM 的输入也可以是普通片内信号，如图 4-90 所示。

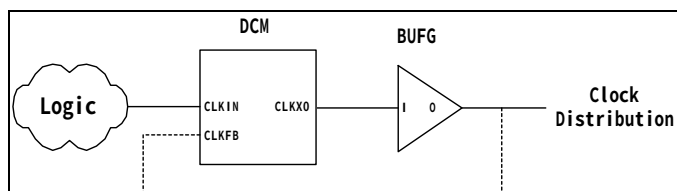


图4-90 Logic + DCM+BUFG 的使用方法

4.4.4 使用 Xilinx 全局时钟资源的注意事项

Xilinx 全局时钟资源必须满足的重要原则是：“使用 IBUFG 或 IBUFGDS 的充分必要条件是信号从专用全局时钟管脚输入”。换句话说，当某个信号从全局时钟管脚输入，不论它是否为时钟信号，都必须使用 IBUFG 或 IBUFGDS；如果对某个信号使用了 IBUFG 或 IBUFGDS 硬件原语，则这个信号必定是从全局时钟管脚输入的。如果违反了这条原则，那么在布局布线时会报错。这条规则是由 Xilinx FPGA/CPLD 的内部结构所决定的：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 IO 和其他内部 CLB 等没有物理连接。

另外，BUFGP 相当于 IBUFG 和 BUFG 的组合，所以 BUFGP 的使用也必须遵循上述的原则。

全局时钟资源的例化方法大概可分为两种：一是在程序中直接例化全局时钟资源，二是通过综合阶段约束或者实现阶段约束完成对全局时钟资源的使用。

第一种方法比较简单，用户只需按照前面讲述的 5 种全局时钟资源的基本使用方法编写代码或者绘制原理图即可。第二种方法是通过综合阶段约束或实现阶段的约束完成对全局时钟资源的调用。这种方法根据综合工具和布局布线工具的不同而异。另外大多数综合工具会

自动分析时钟信号的扇出数目，在全局时钟资源富余的情况下，将扇出数目最大的信号自动指定使用全局时钟资源。这时用户必须检查综合结果是否满足上述使用 IBUFG、IBUFGDS、BUFGP 的原则。如果某个信号因扇出很大，被综合器自动指定使用 IBUFG、IBUFGDS、BUFGP 全局时钟资源，而该信号并没有从专用全局时钟管脚输入，在布局布线时会报错。这时应该在综合时指定该信号不使用全局时钟资源。具体的综合约束命令和操作因综合工具不同而异，下面仅仅举例说明 Synplify/Synplify Pro 和 FPGA Express/FPGA Compiler II 中指定不使用全局时钟资源的方法。

在 Synplify/Synplify Pro 中，打开 SCOPE（Synthesis Constraints Optimization Environment，综合约束优化环境），选择约束属性（Attributes）选项卡。选择约束对象类型（Object Type）为“clock”，然后再约束对象（Object）为设计中的某个时钟，选择约束属性（Attribute）为“syn_noclockbuf”，设置约束值域（Value）为“1”。图 4-91 所示为在 Synplify Pro 的 SCOPE 中指定信号不使用全局时钟资源的方法。

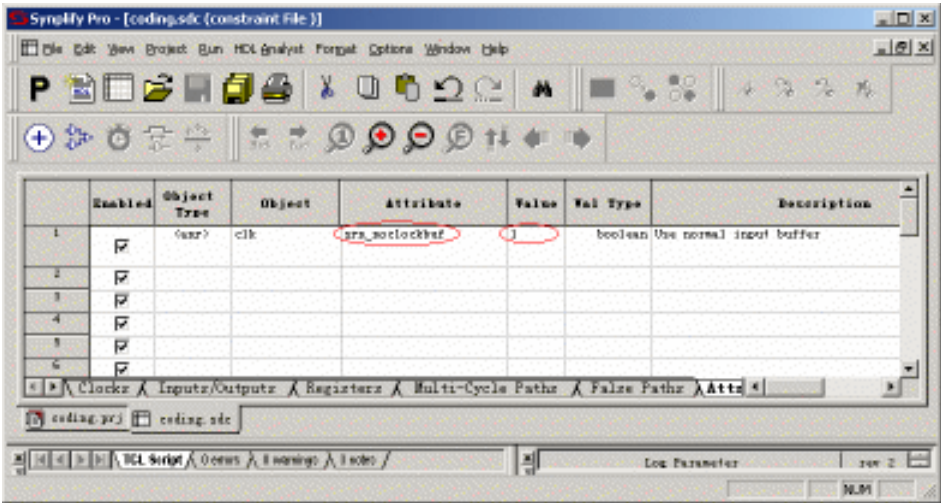


图4-91 在 SCOPE 中指定信号不使用全局时钟资源的方法

也可以在 Synplify/Synplify Pro 约束文件（扩展名为 SDC）或者源代码中使用约束命令指定不使用全局时钟资源，如表 4-7 所示。

表 4-7 Synplify/Synplify Pro 中不使用全局时钟资源的约束命令

文件类型	约束命令
SDC 的约束文件	define_attribute {clk} syn_noclockbuf {1}
Verilog 源代码	input clk /* synthesis syn_noclockbuf = 1 */;
VHDL 源代码	attribute syn_noclockbuf : boolean;
	attribute syn_noclockbuf of clk : signal is true;

在 FPGA Express/FPGA Compiler II 中，用鼠标右键单击编译后的芯片图标，在弹出的菜单中选择【Edit Constraints】命令编辑综合约束文件（扩展名为 CTL），选择端口（Ports）选项卡，指定所需信号的全局时钟域为“DONT USE”。图 4-92 所示为在 FPGA

Express 综合约束编辑器中指定信号不使用全局时钟资源的方法。

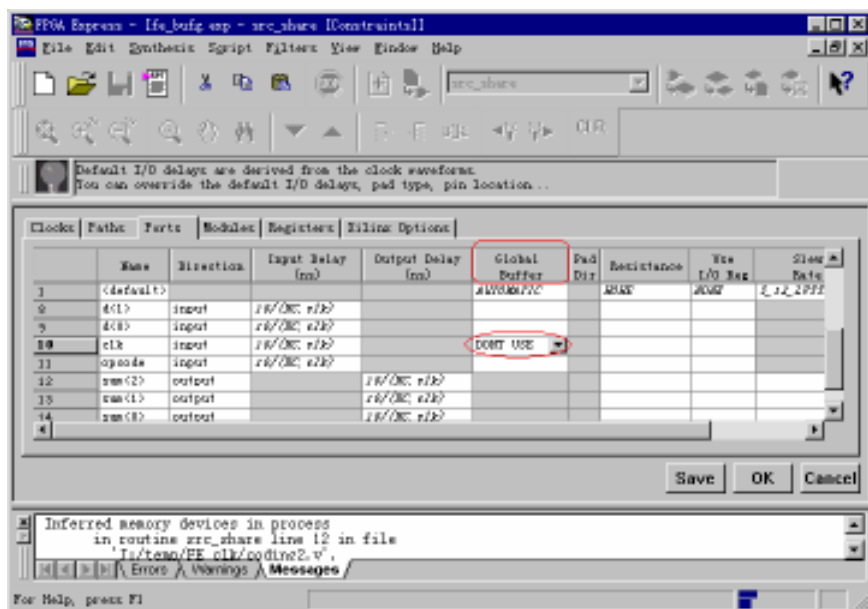


图4-92 在 FPGA Express 综合约束编辑器中指定信号不使用全局时钟资源的方法

也可以在 FPGA Express/FPGA Compiler II 综合约束文件（扩展名为 CTL）中直接使用“port clk global_buffer "DONT USE";”命令指定输入端口信号“clk”不使用全局时钟资源。

4.4.5 第二全局时钟资源

第二全局时钟资源，也叫长线资源。它是分布在芯片的行、列的栅栏（Bank）上，一般采用铜、铝工艺，其长度和驱动能力仅次于全局时钟资源。与全局时钟相似，第二全局时钟资源直接同 IOB、CLB、Block Select RAM 等逻辑单元连接，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/CPLD 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高、扇出数多的时钟、使能、高速路径等信号指定为第二全局时钟信号。第二全局时钟资源的使用方法比较简单，一般是在 Xilinx 的约束编辑器（Constraints Editor）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”。如图 4-93 所示。

也可以直接在指导 Xilinx 实现步骤的用户约束文件（UCF）中添加“USELOWSKEWLINES”约束命令。上面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 "sum<0>" USELOWSKEWLINES;
NET "sum<1>" USELOWSKEWLINES;
NET "sum<2>" USELOWSKEWLINES;
```

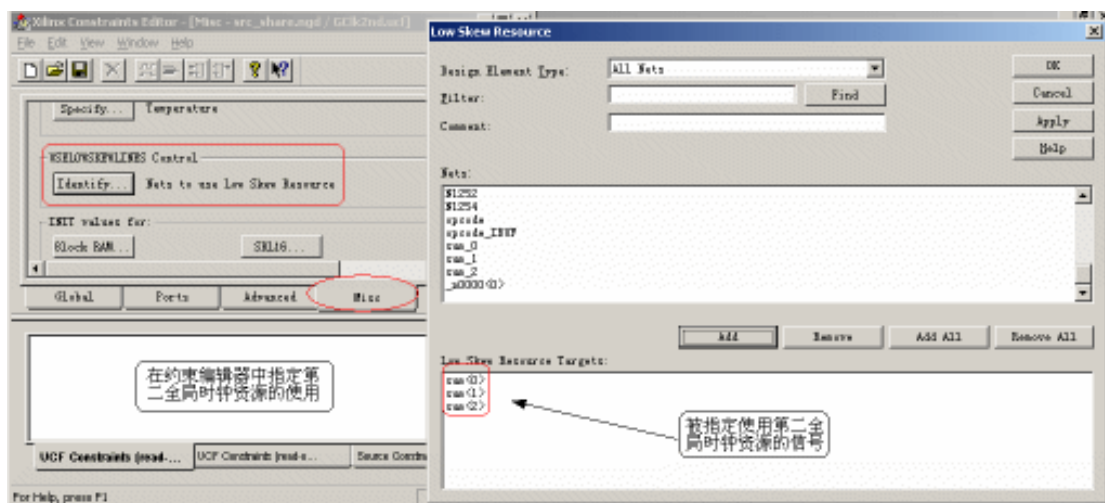


图4-93 使用 Xilinx 约束编辑器指定信号使用第二全局时钟资源的方法

全局时钟资源是专用布线资源，它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。换句话说，全局时钟资源是“不用白不用”的布线资源。而第二全局时钟资源与之不同，它使用的是芯片内部的布线资源。在设计的密度非常大（面积利用率高于 90 %）的情况下，使用所有的第二全局时钟资源会给设计的布局布线带来负面影响，可能导致芯片布线不成功。

如果读者能够在设计中合理地使用全局时钟和第二全局时钟资源，可以提高同步时序电路时钟的可靠性、降低时钟的延迟抖动、提高时钟的驱动能力，从而提高整个设计的可靠性和频率。

4.5 小结

本章介绍了综合在 FPGA/CPLD 设计流程中的作用，同时介绍了 ISE 中集成的 3 种典型综合工具的使用方法与技巧。并在最后引入了全局时钟资源与第二全局时钟资源的概念。综合是 FPGA/CPLD 设计流程中的关键步骤之一，请读者认真学习本章内容，为后面的学习打好基础。

4.6 问题与思考

1. 综合在 FPGA/CPLD 设计中的作用是什么？综合步骤的输入、输出是什么？
2. 简述 Synplify Pro 的特点、功能和使用方法。
3. 在 Synplify Pro 中，如何使用 HDL Analyst、RTL View、Technology View、Timing Analyst 和 Cross-Probing 等工具分析并调试用户设计？
4. 在 Synplify Pro 中，如何使用 FSM Compiler、FSM Explorer 和 FSM Viewer 等工具分析并优化有限状态机？
5. 简述 FPGA Express 的特点、功能和使用方法。
6. 简述全局时钟资源与第二全局时钟资源的概念与基本使用方法。

第5章 约束

在进行高速数字电路设计时，经常需要在综合、实现阶段附加约束，以便控制综合、实现过程，使设计满足运行速度、引脚位置等方面的要求。通常的做法是设计者编写约束文件并导入到综合、实现工具，在 FPGA/CPLD 综合、实现步骤时指导逻辑映射、布局布线。

本章将重点介绍时序约束和分组约束的基本概念、语法和用途，使读者对这两种最常用的约束有比较清晰的认识，然后详细介绍两个约束输入工具：约束编辑器（Constraints Editor）和引脚与区域约束编辑器（PACE）的使用方法。

本章主要内容如下：

- 时序约束；
- 分组约束；
- Constraints Editor；
- 引脚与区域约束编辑器（PACE）；
- 约束文件。

5.1 概述

附加约束主要有以下作用。

(1) 提高设计的工作频率

对很多数字电路设计来说，提高工作频率非常重要，因为高工作频率意味着高处理能力。通过附加约束可以控制逻辑的综合、映射、布局和布线，以减小逻辑和布线延时，从而提高工作频率。一般来说，当时钟频率低于 50MHz，而且设计只有一个时钟时，不需要对设计附加约束条件，这是因为时钟频率比较低时，不加约束自动综合也能达到要求。但是，当设计的时钟频率较高，或者设计中有复杂时序路径（如多周期路径等）时，需要附加约束条件以确保综合、实现的结果满足用户的时序要求。

(2) 获得正确的时序分析报告

Timing Analyzer 是 ISE 中集成的静态时序分析工具，利用这个工具可以获得映射或布局布线后的时序分析报告，从而对设计的性能做出评估。由于 Timing Analyzer 以约束作为判断时序是否满足设计要求的标准，因此要求设计者正确输入约束，以便 Timing Analyzer 输出正确的时序分析报告。

(3) 指定 FPGA/CPLD 引脚位置与电气标准

FPGA/CPLD 的可编程特性使电路板设计加工和 FPGA/CPLD 设计可以同时进行，而不必等 FPGA/CPLD 引脚位置完全确定，从而节省了系统开发时间。这样，电路板加工完成后，设计者要根据电路板的走线对 FPGA/CPLD 加上引脚位置约束，使 FPGA/CPLD 与电路

板正确连接。另外通过约束还可以指定 IO 引脚所支持的接口标准和其他电气特性。为了满足日新月异的通信发展，Xilinx 新型 FPGA/CPLD 可以通过 IO 引脚约束设置支持诸如 AGP、BLVDS、CTT、GTL、GTLP、HSTL、LDT、LVCMOS、LVDCI、LVDS、LVPECL、LVDSEXT、LVTTTL、PCI、PCIX、SSTL、ULVDS 等丰富的 IO 接口标准。

为了对各类约束有个初步的了解，先看一个简单的例子。假设要做一个 32 位的高速计数器，由于计数器的速度取决于最低位到最高位的进位延迟，为了提高速度采用了预定标计数器的结构，也就是把计数器分成一个小计数器和一个大计数器，如图 5-1 所示。其中小计数器是两位的，大计数器是 30 位，它们由同一时钟驱动，但是大计数器的使能端 EN 受小计数器进位驱动，这样小计数器每 4 个 CLK 进位一次，使 EN 持续有效一个 CLK 的时间，此时有效时钟沿到来那么大计数器加 1。可见，小计数器的寄存器可能每个 CLK 翻转一次，低位寄存器输出的数据必须在 1 个 CLK 内到达高位寄存器的输入端，即寄存器之间的最大延时为 1 个 CLK。而大计数器内部的寄存器每 4 个时钟周期才可能翻转一次，低位寄存器输出的数据在 4 个 CLK 内到达高位寄存器的输入端即可，即寄存器之间的最大延迟为 4 个 CLK，因此降低了计数器的时序要求，可以实现规模较大的高速计数器。

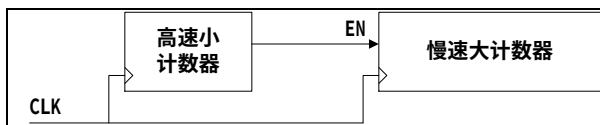


图5-1 预定标计数器

32 位同步预定标计数器参考示例详见本书附带光盘的“Example-5-1”目录。

在用户约束文件 prescale_counter.ucf 中加入以下约束：

```
NET "clk" TNM_NET="clk";
TIMESPEC "TS_clk"=PERIOD "clk" 5ns;
INST "counter_0" TNM="lower_counter";
INST "counter_1" TNM="lower_counter";
TIMEGRP "upper_counter"=FFS(*) EXCEPT "lower_counter";
TIMESPEC "TS_upper_counter"=FROM "upper_counter" TO "upper_counter"
TS_clk*4;
OFFSET=OUT 10ns AFTER "clk";
INST "reset" LOC=L26;
```

其中第 1、3、4、5 条约束是分组约束。第 1 条约束把 clk 驱动的所有同步元件分成一组，名为 clk；第 3、4 条约束将小计数器的两个寄存器构成一个组，组名为 lower_counter；第 5 条约束把小计数器以外的所有寄存器（也就是大计数器的寄存器）命名为 upper_counter。第 2 条约束为周期约束，规定 clk 的周期为 5ns。第 6 条为专门约束，定义 upper_counter 组内的寄存器之间路径的时延为时钟周期的 4 倍，因为预定标计数器的原理决定了这些路径是多周期路径，信号在 4 个时钟周期内稳定下来就可以满足要求。第 7 条约束为偏移约束，定义了输出数据在有效时钟沿之后多长时间稳定下来。最后一条约束为物理位置约束，把 reset 信号锁定在 L26 对应的引脚上。

从综合实现的报告或 Timing Analyzer 可以看出设计是否满足约束文件的要求，如果不

满足，可以修改设计或约束文件，直到满足设计要求。

prescale_counter.ucf 主要包含了时序约束中的周期和偏移约束、分组约束、专门约束。其实约束的种类非常多，主要有时序约束、分组约束、映射约束和布局布线约束等，其中又以时序约束、分组约束最为常用，也最重要，因为对大多数设计来说，提高时钟频率是最重要的，而加时序约束之前一般要用分组约束对触发器等元件进行分组，因此本章主要介绍时序约束和分组约束。

附加时序约束的方法很多，常用的有直接编辑 UCF 文件、从 Constraints Editor 输入、在 HDL 代码或原理图编辑器中附加 Attribute、编辑 Xilinx Constraints File 等，其中以直接编辑 UCF 文件用得最多，可以使用的约束也最多。利用 Constraints Editor 实际上也是编辑 UCF 文件，只是它提供了一个图形化的界面，方便设计者使用，但是功能受到了限制，因此在讲述约束的使用时，以 UCF 文件的语法为主，读者清楚编写 UCF 文件的原理和语法后，使用 Constraints Editor 是非常轻松的事，UCF 文件的语法请参考 5.6.2 小节。

5.2 时序约束

时序约束主要包括周期约束（FFS 到 FFS，即触发器到触发器）和偏移约束（IPAD 到 FFS、FFS 到 OPAD）以及静态路径约束（IPAD 到 OPAD）等 3 种，如图 5-2 所示。通过附加约束条件可以使综合布线工具调整映射和布局布线过程，使设计达到时序要求。例如用 OFFSET_IN_BEFORE 约束可以告诉综合布线工具输入信号在时钟之前什么时候准备好，综合布线工具就可以根据这个约束调整与 IPAD 相连的 Logic Circuitry 的综合实现过程，使结果满足 FFS 的建立时间要求。

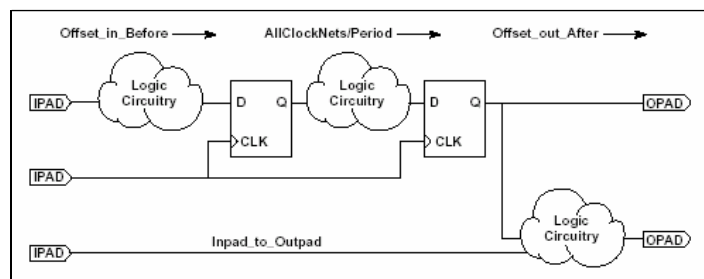


图5-2 4种时序约束

附加时序约束的一般策略是先附加全局约束，然后对快速和慢速例外路径附加专门约束。附加全局约束时，首先定义设计的所有时钟，对各时钟域内的同步元件进行分组，对分组附加周期约束，然后对 FPGA/CPLD 输入输出 PAD 附加偏移约束、对全组合逻辑的 PAD TO PAD 路径附加约束。附加专门约束时，首先约束分组之间的路径，然后约束快、慢速例外路径和多周期路径，以及其他特殊路径。

5.2.1 周期约束（PERIOD 约束）

PERIOD 约束是一个基本时序和综合约束，它附加在时钟网线上，时序分析工具根据 PERIOD 约束检查时钟域内所有同步元件的时序是否满足要求。PERIOD 约束会自动处理寄

寄存器时钟端的反相问题，如果相邻同步元件时钟相位相反，那么它们之间的延迟将被默认限制为 PERIOD 约束值的一半。

PERIOD 约束的作用范围如图 5-3 中粗线所示，图中 D0、D1、OUT0 和 OUT1 为输入/输出 PAD，EN 为 FPGA/CPLD 内部使能信号。PERIOD 约束加在 CLK 网线上，时序分析工具将检查与同步时序约束端口（指有建立、保持时间要求的端口）相连接的所有路径的延迟，包括数据路径 CLB1.Q 至 CLB2.D 和使能端路径 EN 至 CLB2.EC。但是加 PERIOD 约束时，时序分析工具不检查 PAD 到寄存器的路径，即图 5-3 中的 D1 至 CLB1.D。

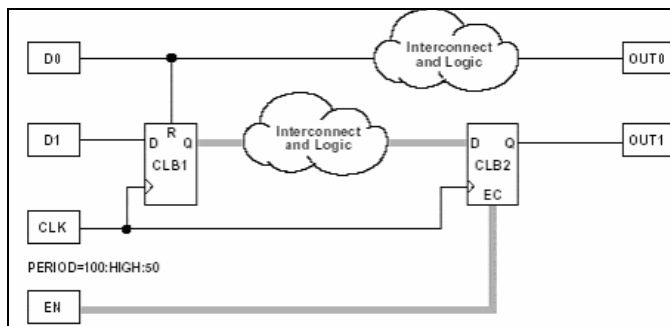


图5-3 PERIOD 约束的作用范围

在附加 PERIOD 约束之前，首先要对电路的时钟周期有一定的估计，这样才不会附加过紧或过松的约束。约束过松，性能达不到要求；约束过紧会增加布局布线的时间，实现的结果也不一定理想，会出现约束过紧性能反而变差的情况。

设计内部电路所能达到的最高运行频率取决于同步元件本身的建立保持时间，以及同步元件之间的逻辑和布线延迟，如图 5-4 所示。

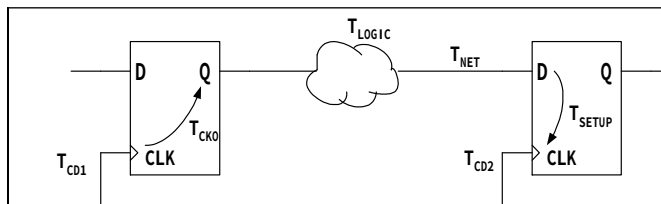


图5-4 时钟周期的计算

根据图 5-4 所示可知，时钟的最小周期为：

$$T_{CLK} = T_{CKO} + T_{LOGIC} + T_{NET} + T_{SETUP} - T_{CLK_SKEW}$$

$$T_{CLK_SKEW} = T_{CD2} - T_{CD1}$$

其中 T_{CKO} 为时钟输出时间， T_{LOGIC} 为同步元件之间的组合逻辑延迟， T_{NET} 为网线延迟， T_{SETUP} 为同步元件的建立时间， T_{CLK_SKEW} 为时钟信号延迟的差别。

附加时钟周期约束有两种方法，一是简单方法（Simple Method），二是首选方法（Preferred Method）。定义时钟周期约束的简单方法是将下面的约束直接附加在寄存器时钟端的路径的网线上。

```
[period_item] PERIOD = period {HIGH | LOW} [high_or_low_time];
```

其中“[]”内的输入或参数为可选项，“{}”的内容是必选项，“|”用于分隔列表中各

选项。

period_item 是 NET “*net_name*” 或 TIMEGRP “*group_name*” 中的一项，分别表示 PERIOD 约束的前溯（forward trace）的方法，前者表示 PERIOD 约束作用到名为 “*net_name*” 的时钟网线所驱动的同步元件上，后者表示 PERIOD 约束作用到 TIMEGRP 所定义的分组上，该分组一般包括 FFS、LATCH 和 RAM 等同步元件，关于分组约束 TIMEGRP 的用法在 5.3 节将详细介绍。

参数 *period* 为要求的时钟周期，可以使用 ps、ns、**us** 或 ms 等单位，大小写都可以，缺省单位为 ns。HIGH|LOW 关键词指出时钟周期里的第一个脉冲是高电平还是低电平，而 *high_or_low_time* 为该脉冲的持续时间，缺省单位为 ns。如果不提供 *high_or_low_time* 参数，那么时钟信号的占空比为 50%。

例如，“NET SYS_CLK PERIOD= 10 ns HIGH 4ns;” 说明时钟 SYS_CLK 的周期为 10ns，高电平的持续时间为 4ns，这个约束将被附加到 SYS_CLK 所驱动的所有同步元件上。当 PERIOD 约束附加到 NET 上时，约束通过与 TNM 类似的前溯方式把约束附加在寄存器上。

为了能定义比较复杂派生关系的时钟周期，需要使用首选方法，其语法如下：

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" period {HIGH|LOW}
[high_or_low_time];
```

其中 TIMESPEC 是一个基本时序相关约束，它在时序规范（timing specification）中作为标识符，表示本约束为时序规范。*TSidentifier* 包括字母 TS 和一个标识符 *identifier*，它们共同构成一个时序规范，称为 TS 属性定义。*TSidentifier* 的值代表了具体的时序规范，可以在设计的其他地方引用。关于 TIMESPEC 和 *TSidentifier* 的更详细的说明请参看 5.2.3 节。

例如定义时钟周期约束时，首先在时钟网线 clk 上附加了 TNM_NET 约束，把 clk 驱动的所有同步元件定义成一个名为 sys_clk 的分组，然后使用 TIMESPEC 约束定义时钟周期：

```
NET "clk" TNM_NET="sys_clk"
TIMESPEC "TS_sys_clk" = PERIOD "sys_clk" 50 HIGH 30;
```

这种定义时钟周期的方法使用了识别符，这样在定义其他时钟周期时就可以引用这个识别符，大大方便了派生时钟的定义。定义派生时钟的语法如下：

```
TIMESPEC "TSidentifier_2"=PERIOD "timegroup_name" "TSidentifier_1" [*
or /] factor PHASE [+ |-] phase_value [units];
```

其中 *TSidentifier_2* 为要定义的派生时钟，*TSidentifier_1* 为已经定义的时钟，*factor* 指出两者周期的倍数关系，是一个浮点数。*phase_value* 指出两者之间的相位关系，为浮点数。

例如定义派生时钟，方法如下。

定义主时钟 clk0：

```
TIMESPEC "TS01" = PERIOD "clk0" 10.0 ns;
```

定义派生时钟 clk180，其相位与主时钟相差 180°：

```
TIMESPEC "TS02" = PERIOD "clk180" TS01 PHASE + 5.0 ns;
```

定义派生时钟 clk90，其相位落后主时钟 90°：

```
TIMESPEC "TS03" = PERIOD "clk90" TS01 PHASE - 2.5 ns;
```

定义派生时钟 clk180_2，其周期为主时钟的 1/2，并延迟 2.5ns：

```
TIMESPEC "TS04" = PERIOD "clk180_2" TS01 / 2 PHASE + 2.5nS;
```

PERIOD 是一个全局约束，通过它可以方便地对时钟线驱动的同步元件的时钟周期进行约束。PERIOD 约束有两种方法，简单方法和首选方法。简单方法是將 PERIOD 直接附加在网线上，首选方法结合 TIMESPEC 和 TSIdentifier 约束使用，这样可以定义具有复杂关系的派生时钟。

5.2.2 偏移约束（OFFSET 约束）

偏移约束包括 OFFSET_IN_BEFORE、OFFSET_IN_AFTER、OFFSET_OUT_BEFORE 和 OFFSET_OUT_AFTER 等 4 种约束，属于基本时序约束。偏移约束规定了外部时钟和数据输入输出引脚之间的时序关系，只用于与 PAD 相连的信号，不能用于内部信号。

使用偏移约束可以为综合实现工具指出输入数据到达的时刻，或者输出数据稳定的时刻，从而在综合实现过程中调整布局布线过程，使正在开发的 FPGA/CPLD 的输入建立时间以及下一级电路的输入建立时间满足要求。

偏移约束的基本语法如下：

```
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 FPGA/CPLD 引脚数据变化与有效时钟沿之间的时间差，{BEFORE|AFTER} 说明该时间差在有效时钟沿的前面还是后面，“clk_name”是时钟名，[TIMEGRP “group_name”] 定义了约束的触发器组，缺省时约束 clk_name 驱动的所有触发器。

1. OFFSET_IN_BEFORE 约束和 OFFSET_IN_AFTER 约束

OFFSET_IN_BEFORE 和 OFFSET_IN_AFTER 都是输入偏移约束，OFFSET_IN_BEFORE 说明了输入数据比有效时钟沿提前多长时间准备好，于是芯片内部与输入引脚相连的组合逻辑的延迟就不能大于该时间，否则有效时钟沿到来时，数据还没有稳定，采样将发生错误；OFFSET_IN_AFTER 指出输入数据在有效时钟沿之后多长时间到达芯片的输入引脚，这样也可以得到芯片内部延迟的上限，从而对与输入引脚相连的组合逻辑进行约束。

输入数据到达芯片引脚延迟的计算可参见图 5-5。

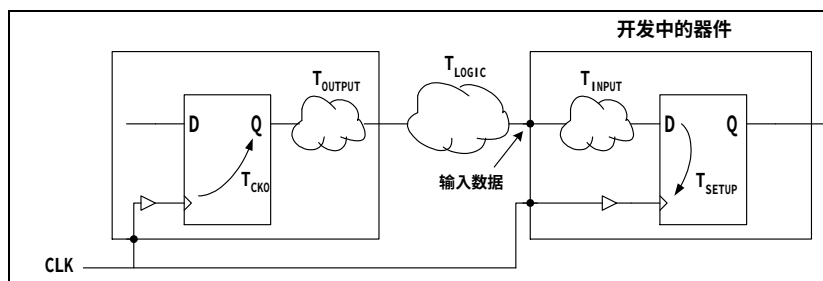


图5-5 输入到达时间的计算

计算公式如下：

$$T_{ARRIVAL} = T_{CKO} + T_{OUTPUT} + T_{LOGIC}$$

即输入数据在有效时钟沿之后的 T_{ARRIVAL} 时刻到达。有了这个数据之后，可以对设计输入端附加 `OFFSET_IN_AFTER` 约束。例如，`NET DATA_IN OFFSET=IN  $T_{\text{ARRIVAL}}$  AFTER CLK`，这样在对同步元件输入端的逻辑进行约束后，综合实现工具会努力使输入端的延迟 T_{INPUT} 满足以下关系：

$$T_{\text{ARRIVAL}} + T_{\text{INPUT}} + T_{\text{SETUP}} < T_{\text{CLK}}$$

其中 T_{INPUT} 为输入端的组合逻辑、网线和 PAD 的延迟之和， T_{SETUP} 为输入同步元件的建立时间。

例如，假设时钟周期为 20ns，前级寄存器的 T_{CKO} 为 1ns，前级输出逻辑延时 T_{OUTPUT} 为 3ns，中间逻辑 T_{LOGIC} 的延时为 10ns，那么 $T_{\text{ARRIVAL}}=14\text{ns}$ ，于是可以在数据输入引脚附加 `NET DATA_IN OFFSET=IN 14ns AFTER CLK` 约束，也可以使用 `OFFSET_IN_BEFORE` 对芯片内部的输入逻辑进行约束，其语法如下：

`NET DATA_IN OFFSET=IN  $T_{\text{DELAY}}$  BEFORE CLK`

其中 T_{DELAY} 为要求的芯片内部输入延迟，其最大值与输入数据到达时间 T_{ARRIVAL} 的关系如图 5-6 所示。

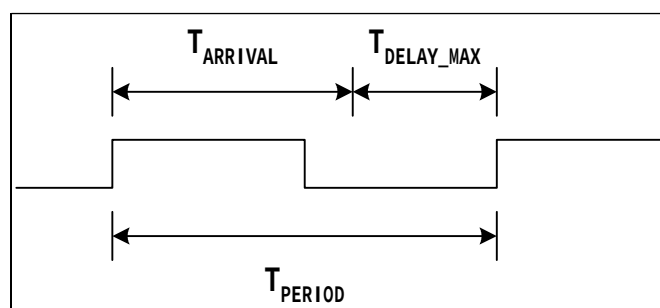


图5-6 输入延迟与输入数据到达时间的关系

因此 T_{DELAY} 应小于 $(T_{\text{PERIOD}} - T_{\text{ARRIVAL}})$ ， T_{PERIOD} 为时钟周期。

满足以上约束时，设计的电路能够可靠地采样输入数据。在上一个例子中，附加下面的约束有同样的作用：

`NET DATA_IN OFFSET=IN 6ns BEFORE CLK`

2. OFFSET_OUT_AFTER 约束和 OFFSET_OUT_BEFORE 约束

两者都是输出偏移约束，`OFFSET_OUT_AFTER` 规定了输出数据在有效时钟沿之后多长时间稳定下来，那么芯片内部的输出延迟就必须小于这个值。`OFFSET_OUT_BEFORE` 指出下一级芯片的输入数据应在有效时钟沿之前多长时间准备好。

从下一级输入端的延迟可以计算出当前设计输出的数据必须在何时稳定下来，根据这个数据对设计输出端的逻辑布线进行约束，以满足下一级的建立时间要求，保证下一级采样的数据是稳定的。

计算要求的输出稳定时间如图 5-7 所示。

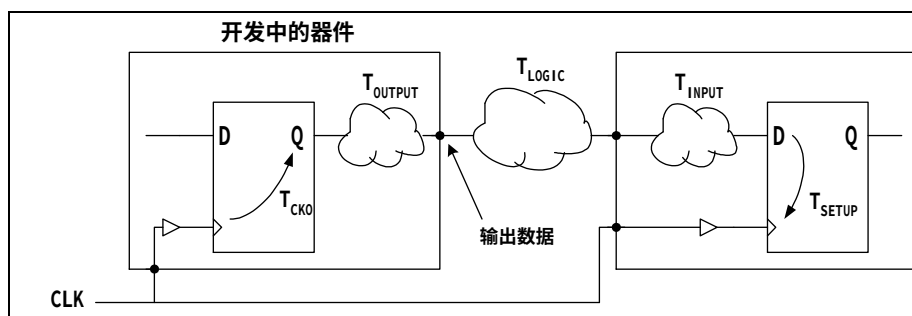


图5-7 计算要求的输出稳定时间

公式如下：

$$T_{\text{STABLE}} = T_{\text{LOGIC}} + T_{\text{INPUT}} + T_{\text{SETUP}}$$

那么只要当前设计输出端的数据比时钟上升沿提早 T_{STABLE} 时间稳定下来，下一级就可以正确地采样数据。

根据这个数据可以对设计输出端附加 `OFFET_OUT_BEFORE` 约束，例如，`NET DATA_OUT OFFET=OUT TSTABLE BEFORE CLK`，这样对同步元件输出端的逻辑进行约束后，综合实现工具会努力调整该逻辑的实现，使输出端的延迟满足以下关系：

$$T_{\text{CKO}} + T_{\text{OUTPUT}} + T_{\text{STABLE}} < T_{\text{CLK}}$$

其中 T_{OUTPUT} 为设计中连接同步元件输出端的组合逻辑、网线和 PAD 的延迟之和， T_{CKO} 为同步元件时钟输出时间。

例如，假设时钟周期为 20ns，后级输入逻辑延时 T_{INPUT} 为 4ns、建立时间 T_{SETUP} 为 1ns，中间逻辑 T_{LOGIC} 的延时为 10ns，那么 $T_{\text{STABLE}}=15\text{ns}$ ，于是可以在数据输入引脚附加“`NET DATA_OUT OFFET=OUT 15ns BEFORE CLK`”约束，也可以直接对芯片内部的输出逻辑直接进行约束，如“`NET DATA_OUT OFFET=OUT TOUTPUT_DELAY AFTER CLK`”，其中 $T_{\text{OUTPUT_DELAY}}$ 为要求的芯片内部输出延迟，其最大值与要求的输出数据稳定时间 T_{STABLE} 的关系如图 5-8 所示。

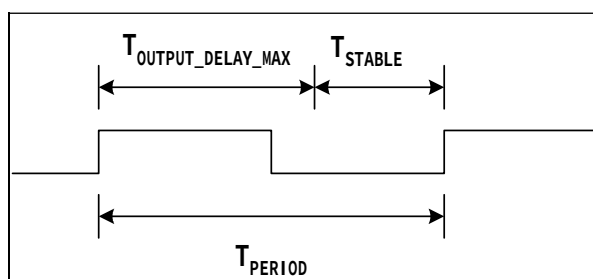


图5-8 输出延迟与要求的输出数据稳定时间

因此 $T_{\text{OUTPUT_DELAY}}$ 应小于 $(T_{\text{PERIOD}} - T_{\text{STABLE}})$ ， T_{PERIOD} 为时钟周期。

在这个例子中，附加“`NET DATA_OUT OFFET=OUT 5ns AFTER CLK`”约束有同样的作用。

5.2.3 专门约束

附加约束的一般策略是首先附加整体约束，例如 PERIOD、OFFSET 等，然后对局部的电路附加专门约束，这些专门约束通常比整体约束松，通过在可能的地方尽量放松约束可以提高布线通过率，减小布局布线的时间。当然，也有可能在局部附加比较紧的约束，这也需要通过专门约束来实现。

(1) FROM_TO 约束

FROM_TO 在两个组之间定义时序约束，对两者之间的逻辑和布线延迟进行控制，这两个组可以是用户定义的，也可以是预定义的。用户可以使用 TNM_NET、TNM 和 TIMEGRP 定义组，而预定义的组主要包括 FFS、LATCHES、PADS 和 RAMS 等，如表 5-1 所示。

表 5-1 预定义的组及其描述

关键字	描述
FFS	只包括 CLB 或 IOB 中的触发器，不包括用 function generators 组成的触发器
LATCHES	只包括 CLB 或 IOB 中的锁存器，不包括用 function generators 组成的锁存器
PADS	输入输出 pads
RAMS	用于有 RAM 的器件（在 Virtex/-E/-II 和 Spartan-II 器件中包含 LUT RAMS 和 Block RAMS，但不包括 Shift Register LUTs）
BRAMS_PORTA	用法与预定义的 RAMS 相同，但是分组具体到双口 Block RAM 的特定端口上
BRAMS_PORTB	

FROM_TO 的语法如下：

```
TIMESPEC "TSname"=FROM "group1" TO "group2" value;
```

其中 *group1*、*group2* 分别为路径的起点和终点，*value* 为延迟时间，可以是具体数值或表达式，如 TS_C2S/2、TS_C2S*2 等。

图 5-9 画出了可以用 FROM_TO 约束的 4 种典型路径，包括输入逻辑延迟 TS_P2S、同步元件之间的延迟 TS_C2S、输出逻辑延迟 TS_C2P 和静态路径延迟 TS_P2P。使用预定义组时，可用以下语句附加约束：

```
TIMESPEC TS_P2S=FROM PADS TO FFS 25;
TIMESPEC TS_C2S=FROM FFS TO FFS 30;
TIMESPEC TS_C2P=FROM FFS TO PADS 10;
TIMESPEC TS_P2P=FROM PADS TO PADS 23;
```

FROM_TO 约束的一个重要应用是慢速例外，在有些设计中，局部电路的约束可以比全局约束松，这时一般的做法是首先附加全局约束，然后再使用 FROM_TO 等语句对局部电路附加较松的约束。

以图 5-10 为例，假设大部分电路的延时要求与 FF1、FF2 之间的延迟要求相同，为 30ns，而 FF2 和 FF3 之间的路径延迟为 60ns，附加约束的方法主要有两种，第一种方法是对延迟要求不同的电路分别约束，使用以下语句：

```
TIMESPEC FF1_FF2=FROM FF1 TO FF2 30ns;
TIMESPEC FF2_FF3=FROM FF2 TO FF3 60ns;
```

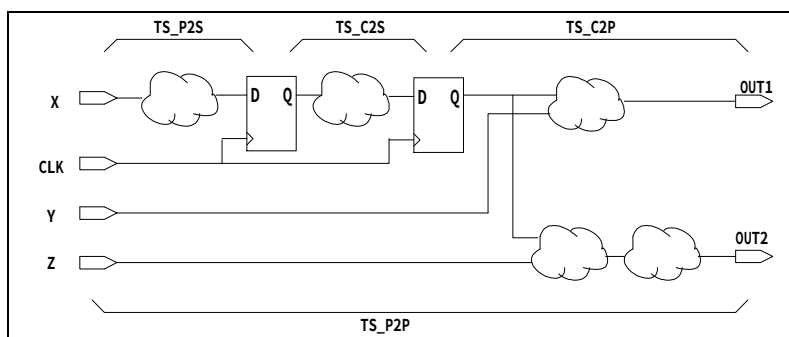


图5-9 FROM_TO 约束的典型路径

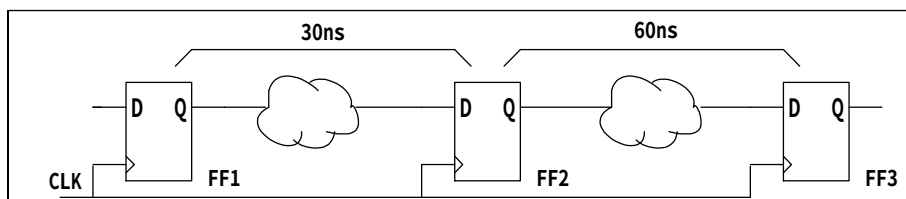


图5-10 慢速例外示例

第二种方法是先做全局约束，再对延迟要求比较低的部分做约束，作为慢速例外，使用以下语句实现。

```
TIMESPEC TS_CLK=PERIOD CLK 30ns;
TIMESPEC FF2_FF3=FROM FF2 TO FF3 60ns;
```

由于约束的优先级由高到低依次为 TIG、FROM_THRU_TO、FROM_TO、PERIOD，也就是 FROM_TO 的级别比 PERIOD 高，因此 FF2 到 FF3 的路径将被约束为 60ns。

以上两种约束方法都是正确的，但是第二种使用慢速例外的方法更好，因为布局布线工具和时序分析工具可以执行得更快。

(2) MAXDELAY 约束

MAXDELAY 约束定义了特定网线上的最大延时，其语法如下：

```
NET "net_name" MAXDELAY=value units;
```

其中 value 为延迟时间，unit 可以是 us、ns、ms 或 ps 之一。

例如：

```
NET "EN" MAXDELAY=5 ns;
NET "$1I3245/$SIG_4" MAXDELAY=10 ns;
```

(3) MAXSKEW 约束

MAXSKEW 是高级时序约束，通过把 MAXSKEW 约束附加到某一网线上，可以控制该网线上的 skew 不大于某一数值。

skew 是指由同一点驱动的信号到达两个或两个以上终点的时间延迟的最大差别，如图 5-11 所示，时钟输入到 A 的延迟为 $t_a=2\text{ns}$ ，时钟输入到 B 的延迟为 $t_b=4\text{ns}$ ，那么时钟网线上的 skew 为 $4-2=2\text{ns}$ 。

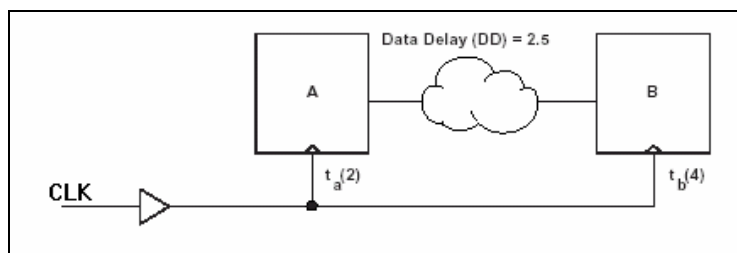


图5-11 skew 示意图

MAXSKEW 约束的语法如下：

```
NET "net_name" MAXSKEW=allowable_skew [units];
```

例如：

```
NET "SIG_3" MAXSKEW=3ns;
```

(4) TIMESPEC 约束

TIMESPEC 是一个基本时序相关约束，它在时序规范中作为标识符，这个时序规范称为 TS 属性定义。

TIMESPEC 约束一般用在以下场合：

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" value [units];
```

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" value  
units;
```

例如：

```
TIMESPEC "TS_sys_clk"=PERIOD "sys_clk" 50ns HIGH 30;
```

```
TIMESPEC "TS_to_RAM"=FROM FFS TO RAMS 35ns;
```

(5) TSidentifier 约束

TSidentifier 是一个基本时序约束，TSidentifier 以“TS”开头，在 UCF 文件中与 TIMESPEC 一起使用。TSidentifier 的值代表了具体的时序规范，可以在设计的其他地方引用，其中 identifier 是 ASCII 码字符串，由 A-Z、a-z、0-9 以及下划线组成。

TSidentifier 约束一般用在以下场合：

- 定义时钟周期

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" value [units] [{HIGH |  
LOW} [high_or_low_time[hi_lo_units]]];
```

- 定义派生时钟

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference"  
"another_PERIOD_identifier" [/ | *] number [{HIGH |LOW} [high_or_low_time  
[hi_lo_units]]];
```

- 定义最大延迟

```
TIMESPEC "TSidentifier"=[MAXDELAY] FROM "source_group" TO "dest_group"  
allowable_delay [units];
```

- 定义相关时序规范

```
TIMESPEC "TSidentifier"=FROM "source_group" TO
```

```
"dest_group" another_TSid[/ | *] number;
```

- 定义中间点

```
TIMESPEC "TSidentifier"=FROM "source_group" THRU "thru_point" [THRU  
"thru_point1"... "thru_pointn"] TO "dest_group" allowable_delay [units];
```

- 定义无关路径

```
TIMESPEC "TSidentifier"=FROM "source_group" TO  
"dest_group" TIG;
```

(6) TIG 约束

TIG (Timing Ignore) 是基本时序约束和综合约束，用于定义无关路径。把 TIG 约束附加在网线上，那么做时序分析时从该网线开始的路径将被忽略。

例如在图 5-12 中附加 “NET NET_C TIG”，那么时序分析工具将忽略从 NET_C 开始的路径（即图 5-12 中的阴影线）。

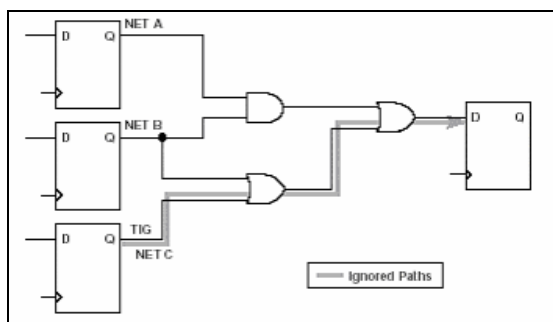


图5-12 TIG 附加在 NET_C 上

TIG 约束的语法如下：

```
NET "net_name" TIG;
```

例如：

```
NET "Sig_5" TIG
```

```
NET "$1I6745/$SIG_4" TIG
```

5.3 分组约束

一个真正的设计往往包括大量的触发器、寄存器和 RAM 等元件，为了方便附加约束需要把它们分成不同的组，然后根据需要对这些组分别附加不同的约束，例如，本章开始介绍的同步预定标计数器附加约束时，首先使用分组约束 TNM 和 TIMEGRP 把寄存器分成两组，再附加不同的约束。正确分组才能正确、方便地附加约束。

5.3.1 TNM 约束

使用 TNM (Timing Name) 约束可以选出构成一个分组的元件，并赋予一个名字，以便给它们附加约束，这些元件包括 FFS、RAMs、LATCHES、PADS、BRAMS_PORTA、BRAMS_PORTB 等。

TNM 约束可以附加在 Net、Primitive、Macro 以及 Macro/Primitive 的 PINS 上。TNM 的语法如下：

```
{NET | INST | PIN} "object_name" TNM="identifier";
```

其中 *object_name* 为 NET、INST 或 PIN 的名称，*identifier* 为分组的名称。

- (1) 当 TNM 约束加在网线上，该网线所在路径的有效同步元件将被赋予该 TNM 值，如图 5-13 所示。图中 TNM=FLOPS 约束加在名为 A0 的网线上，于是阴影标记的触发器被分到名为 FLOPS 的分组中。

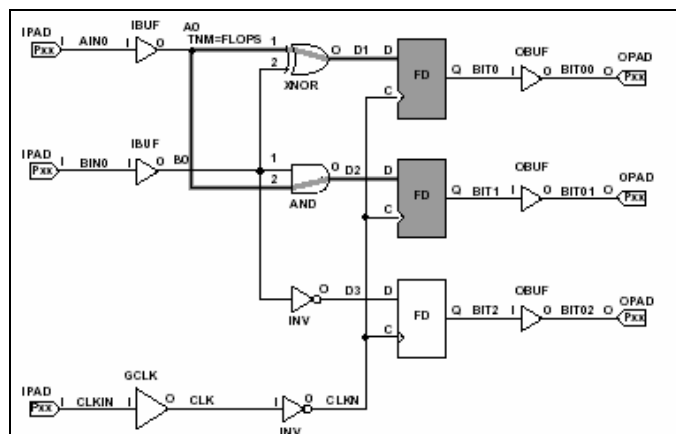


图5-13 TNM 约束附加在网线上

在 UCF 文件中附加的语句如下：

```
NET "A0" TNM="FLOPS";
```

需要注意的是，当 TNM 加到 PAD NET 上时，TNM 的值将被赋予该 PAD，不会穿过 IBUF 向前传播。

- (2) 当 TNM 约束加在 Macro/Primitive 的引脚上，则该引脚驱动的所有有效同步元件将被赋予该 TNM 值，如图 5-14 所示。图中 TNM=FFS:FLOPS 约束加在 Macro 的一个引脚上，于是该引脚所在路径上的所有触发器归到名为 FLOPS 的分组中。

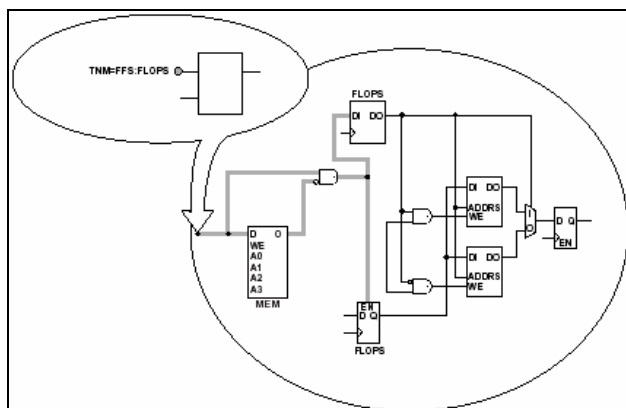


图5-14 TNM 约束附加在 Macro/Primitive 的引脚上

在 UCF 文件中附加的语句如下：

```
PIN "pin_name" TNM="FLOPS";
```

- (3) 当 TNM 约束直接附加在 Primitive 上，那么这个 Primitive 将作为命名组的一部分，如图 5-15 所示。图中 TNM=FLOPS 约束直接附加在阴影标记的触发器上，所以这些触发器被归到 FLOPS 分组中。

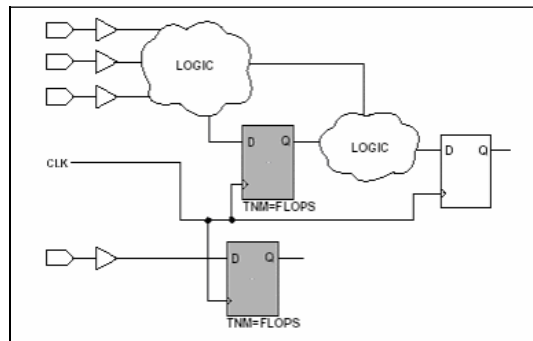


图5-15 TNM 约束附加在 Primitive 上

在 UCF 文件中附加的语句如下：

```
INST "instance_name" TNM=FLOPS;
```

图 5-15 中的两个触发器都属于名为 FLOPS 的分组。

- (4) 如果 TNM 约束附加在一个 Macro 上，那么该 Macro 所有层次下的有效同步元件和 PAD 将成为命名组的一部分，如图 5-16 所示。

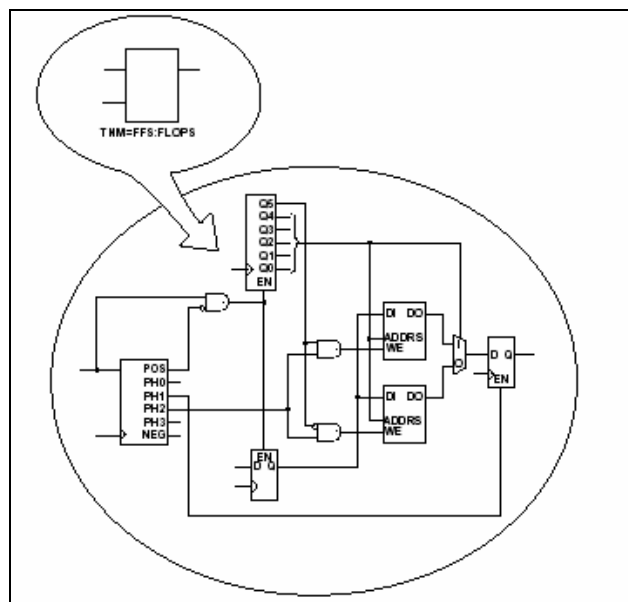


图5-16 TNM 约束附加在 Macro 上

在图 5-16 中，TNM=FFS:FLOPS 约束附加在 Macro 上，其作用是把 Macro 中的所有触发器定义为一个分组，名为 FLOPS。

5.3.2 TNM_NET 约束

TNM_NET (timing name for nets) 约束只加在网线上，其作用与 TNM 约束加在网线时基本相同，即把该网线所在路径上的所有有效同步元件作为命名组的一部分。不同之处在于当 TNM 约束加在 PAD NET 上时，TNM 的值将被赋予 PAD，而不是该网线所在的路径上的同步元件，即 TNM 约束不能穿过 IBUF。用 TNM_NET 约束就不会出现这种情况。

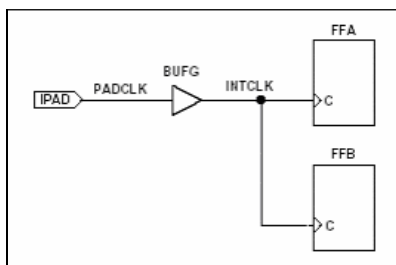


图5-17 时钟输入电路

图 5-17 所示的设计中，如果用 TNM 加在 PADCLK 上，即使用下面语句：

```
NET "PADCLK" TNM= "PADGRP";
```

则所得到的命名组“PADGRP”仅包含 IPAD。

使用 NET “PADCLK” TNM=FFS(*) “FFGRP”;语句无法把 FFA、FFB 包含在命名组里，因为 TNM 的值“FFGRP”被转移到 IPAD 上，因而找不到与 FFS(*)匹配的元件。

为了解决这个问题，在采用 Schematic 输入时，可以把 TNM 约束附加在 INTCLK 上，即

```
NET "INTCLK" TNM=FFS(*)"FFGRP";
```

但是在 HDL 文件中，只有与 PAD 相连的网线才是有意义的，也就是难以引用 INTCLK，这时要使用 TNM_NET，即

```
NET "PADCLK" TNM_NET=FFS(*)"FFGRP";
```

TNM_NET 的值将会正确地穿过 BUFG 传播到 FFA、FFB 上，从而把它们包含在分组中。

TNM_NET 的语法如下：

```
NET "net_name" TNM_NET=[predefined_group:]identifier;
```

TNM_NET 只能用在网线上，否则会出现警告，同时该约束将被忽略。

5.3.3 TIMEGRP 约束

除了使用 TNM 约束定义分组之外，还可以通过 TIMEGRP 约束使已有的分组构成新的分组。已有的分组包括预定义的和用 TNM/TIMEGRP 定义的。

使用 TIMEGRP 约束可以合并多个分组以构成一个新分组，或用排除的方法构成新的分组，例如下列语句：

```
TIMEGRP "big_group"="small_group" "medium_group";
```

把 *small_group*、*medium_group* 合成一个新分组 *big_group*。

在语句

```
TIMEGRP "group1"="group2" EXCEPT "group3";
```

中，*group2*、*group3* 可以是预定义的分组或用户已经定义过的分组，新的分组 *group1* 包含了 *group2* 中除了 *group3* 以外的所有内容。

也可以利用上升沿或下降沿触发的不同定义新的分组，例如下列语句

```
TIMEGRP "group1"="RISING" ffs_group";
```

把分组 *ffs_group* 中由上升沿触发的触发器命名为 *group1*。

```
TIMEGRP "group2"="FALLING" ffs;
```

把所有下降沿触发的触发器命名为 *group2*。

5.3.4 TPTHUR 约束

TPTHUR 约束用于定义一个或一组中间点，以便在后面的时序规范中用来标识特定的路径，其语法如下：

```
{NET | INST | PIN} "net_name" TPTHUR=identifier;
```

例如在图 5-18 中，要对阴影的路径单独进行约束，首先要标识出这条路径，这就要定义一个中间点 ABC，以区别于另外一条平行路径，在 UCF 文件中使用的语句如下：

```
INST "FLOPA" TNM="A";
```

```
INST "FLOPB" TNM="B";
```

```
NET "MYNET" TPTHUR="ABC";
```

```
TIMESPEC "TSpath1"="FROM "A" THRU "ABC" TO "B" 30;
```

其中第三条语句把名为“MYNET”的网线定义为中间点，中间点的名字为“ABC”，第四条语句在此基础上规定了通过该点的整条路径的时延。

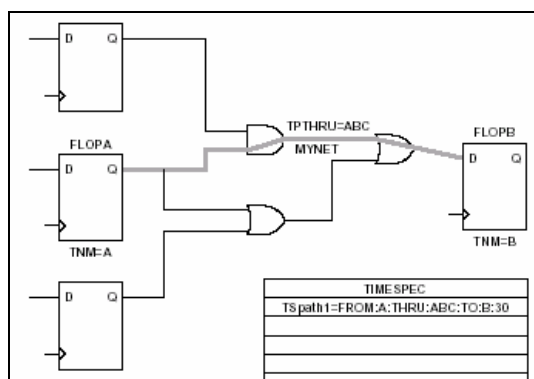


图5-18 用 TPTHUR 定义一个中间点

5.3.5 TPSYNC 约束

通常时序规范的起点和终点都是 FFS、LATCHES、RAMS 等同步元件或 PADS，如果

要使用除此之外的点做时序规范的起点或终点，那么要用 TPSYNC 定义该点为一个同步点。

TPSYNC 约束的语法如下：

```
{NET | INST | PIN} "net_name" TPSYNC=identifier;
```

如果 TPSYNC 约束附加在网线，那么驱动网线的点定义为同步点。当附加在宏的引脚时，如果是输出引脚，那么宏里面驱动该引脚的驱动源被定义为同步点，如果该引脚是输入引脚，那么该引脚驱动的所有 PIN 被定义为同步点。如果 TPSYNC 约束附加在 Primitive 的输入输出引脚，那么该引脚被定义为同步点。如果附加在 INST，它的输出定义为同步点。如果附加在 Primitive Symbol，其输出定义为同步点。

5.4 约束编辑器——Constraints Editor

在 Xilinx 开发环境中，映射前输入的约束称为逻辑约束，逻辑约束保存在 UCF 文件中。设计者创建和修改约束时可以直接编辑 UCF 文件，这要求熟悉 UCF 文件的语法以及设计中各触发器、网线、PAD 等元件的命名，否则容易发生错误。使用 Constraints Editor 创建和修改约束可以降低这方面要求，提高工作效率。本节将详细介绍 Constraints Editor 的用法。

5.4.1 Constraints Editor 的用户界面

Constraints Editor 的启动方法有两种：一是直接在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【Constraints Editor】命令；另一种方法是在 Project Navigator 中新建 UCF 文件，选中该文件，然后在“Processes for Source”窗口中双击“Create Timing Constraints”。Constraints Editor 的用户界面如图 5-19 所示。

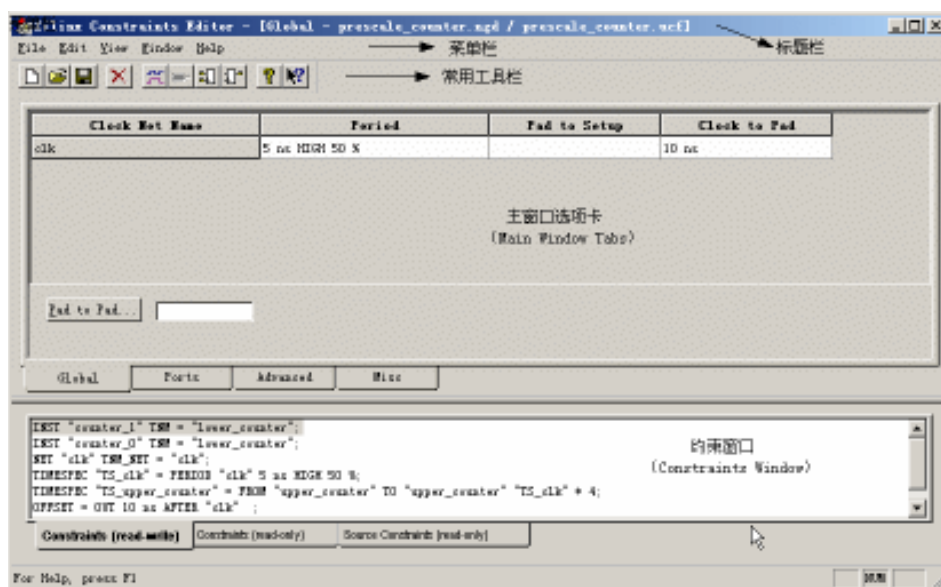


图5-19 Constraints Editor 的用户界面

Constraints Editor 的用户界面主要由标题栏、菜单栏、常用工具栏、主窗口选项卡

(Main Window Tabs) 和约束窗口 (Constraints Window) 等部分组成, 下面简要介绍各部分的功能。

1. 菜单栏

菜单栏由包含一系列常用处理命令的菜单组成。

- 文件 (File) 菜单包含了常见的文件操作命令。
- 编辑 (Edit) 菜单包含了编辑约束的命令, 主要有【Delete】(删除约束)、【Period】(附加周期约束)、【Pad to Setup】(附加输入延迟约束) 和【Clock to Pad】(附加输出延迟约束) 等命令。
- 视图 (View) 菜单主要包括了显示工具栏 (Toolbars) 和显示状态栏 (Status Bar) 等命令。
- 窗口 (Window) 菜单由一些通用窗口调整与排列命令组成。
- 帮助 (Help) 菜单包含了 Constraints Editor 在线帮助和版本信息等。

2. 常用工具栏

常用工具栏集成了 Constraints Editor 中一些主要功能的快捷工具按钮, 下面简要介绍主要工具按钮的功能。

-  (Delete) 按钮的功能是删除约束。在约束窗口中选中一条约束, 单击该按钮可以删除这条约束。
-  (Period) 按钮的功能是加入周期约束。单击该按钮会弹出【Clock Period】对话框, 在对话框中可以进行周期约束的编辑, 其作用和在【Global】选项卡的 clk 行中双击 period 列相同。
-  (Location) 按钮的功能是对输入输出端口附加物理位置约束。
-  (Pad to Setup) 按钮的功能是对输入端口附加输入延迟约束。
-  (Clock to Pad) 按钮的功能是对输出端口附加输出延迟约束。

3. 主窗口选项卡 (Main Window Tabs)

主窗口选项卡用于附加约束, 包含【Global】(全局)、【Ports】(端口)、【Advanced】(高级) 和【Misc】(其他) 等 4 个选项卡, 可以分别用于附加全局约束、端口约束、分组和时序约束以及附加其他约束。

4. 约束窗口 (Constraints Window)

在主窗口选项卡中附加约束后, 该约束将被添加到 UCF 文件中, 在约束窗口可以看到 UCF 文件中的语句。在【UCF Constraints (read-write)】选项卡中可以对约束进行编辑。

在认识了 Constraints Editor 的用户界面以后, 接下来以本章开始介绍的同步预定标计数器为例, 说明用 Constraints Editor 附加约束的方法。附加的约束按先后次序有全局约束、端口约束、分组和时序约束。

5.4.2 附加全局约束

用户界面的主窗口选项卡中的【Global】选项卡用于附加全局约束, 包括周期约束、输入延迟约束和输出延迟约束。

1. 附加周期约束

在【Global】选项卡中，双击 clk 行 Period 列会弹出【Clock Period】对话框，如图 5-20 所示。

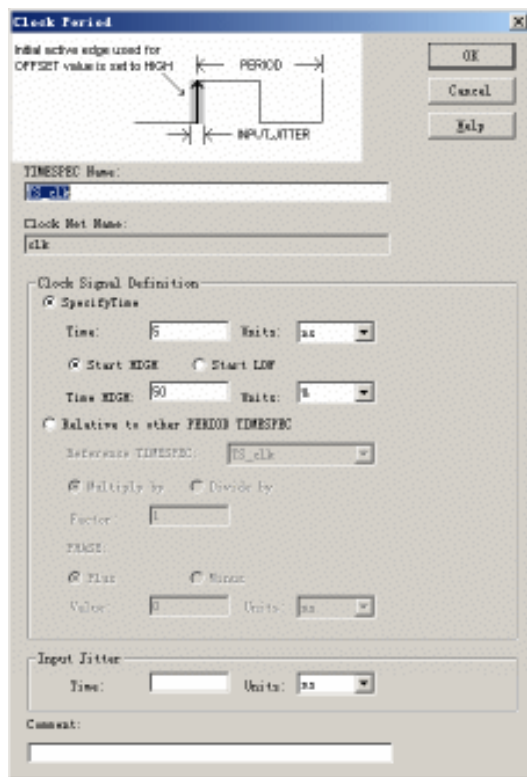


图5-20 【Clock Period】对话框

在【TIMESPEC Name】文本框中填入 TSIdentifier 格式的时序规范名，【Clock Net Name】栏为需要附加 PERIOD 约束的时钟网线。在【Clock Signal Definition】中可以输入周期的具体数值，或者相对其他 PERIOD TIMESPEC 的倍数和相位。设计者还可以在【Comment】文本框里对这条周期约束加上注释。

ISE 6 的 Constraints Editor 比以往版本功能有所增强，例如在【Input Jitter】中设置时钟抖动，同时增加了图示说明（图 5-20 上半部分），使用起来非常方便。

在同步预定标计数器的例子中，按图 5-20 所示设置时钟周期为 5ns，占空比为 50%，按 **OK** 按钮后，可以在下方的约束窗口看到相应的 UCF 语句：

```
NET "clk" TNM_NET = "clk";  
TIMESPEC "TS_clk" = PERIOD "clk" 5 ns HIGH 50 %;
```

2. 附加输入延迟约束

在【Global】选项卡中，双击 clk 行 Pad to Setup 列会弹出【Pad to Setup】对话框，如图 5-21 所示。

在【Time Requirement】中输入延迟要求，并在【Relative to Clock Pad Net】下拉列表中选择输入数据的时钟，单击 **OK** 按钮便完成全局输入延迟约束的设置。按图 5-21 所示

设置完成后，在约束窗口看到新增的 UCF 约束语句如下：

```
OFFSET = IN 20 ns BEFORE "clk";
```

在同步预定标计数器例子中，由于没有同步输入端，因此不必设置全局输入延迟约束。

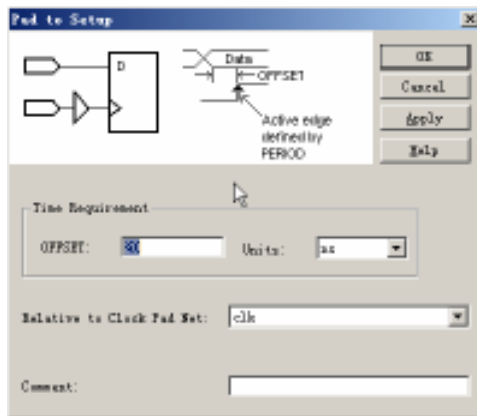


图5-21 【Pad to Setup】对话框

3. 附加输出延迟约束

在【Global】选项卡中，双击 clk 行 Clock to Pad 列将弹出【Clock to Pad】对话框，如图 5-22 所示。

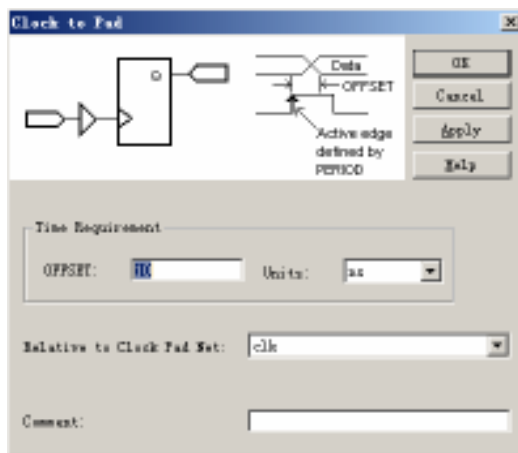


图5-22 【Clock to Pad】对话框

填写了输出延迟要求、输出数据的时钟和注释后，单击 **OK** 按钮即可。

同步预定标计数器例子要求输出延迟为 10ns，按图 5-22 设置后，新增的 UCF 语句如下：

```
OFFSET = OUT 10.000 ns AFTER "clk";
```

5.4.3 附加端口约束

在 Constraints Editor 主窗口的【Ports】选项卡中，可以对每一个端口单独进行设置。选中【I/O Configuration Options】选项时，可设置的项目包括【Location】（物理位置）、【Pad

to Setup】（输入延迟）、【Clock to Pad】（输出延迟）、【FAST/SLOW】（输出速度）、【PULLUP/ PULLDOWN】（上拉/下拉）、【IOSTANDARD】（接口标准）和【DRIVE】（驱动电流）等，如图 5-23 所示。

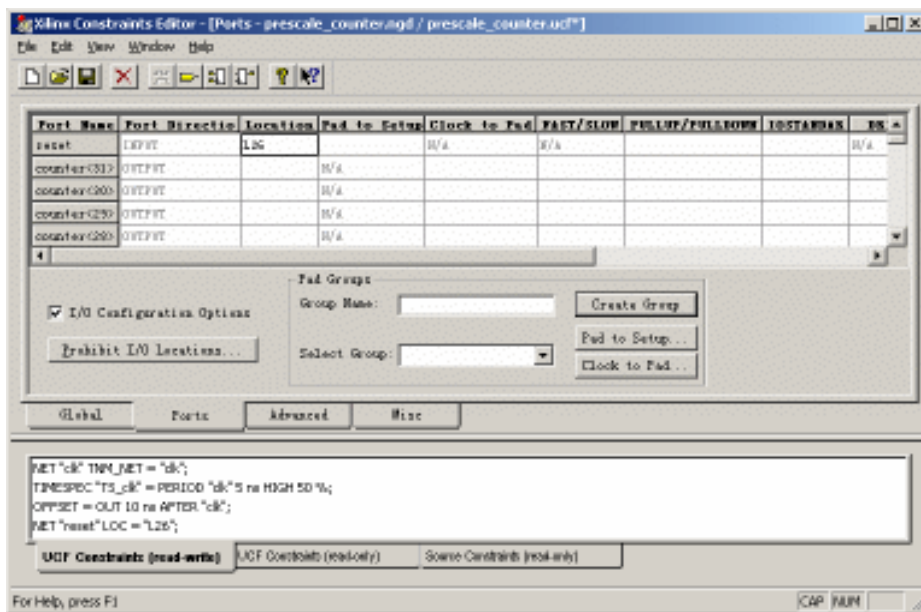


图5-23 【Ports】选项卡



设计者可以对输入输出端口创建分组，然后统一设置输入输出延迟约束。首先在 Port Name 列选中一个或多个端口，然后在【Group Name】栏中输入分组名，单击 **Create Group** 按钮完成分组的创建。在【Select Group】下拉列表中选择该分组，单击 **Pad to Setup...** 或 **Clock to Pad...** 按钮即可设置输入输出延迟约束。

为了把同步预定标计数器的 reset 端锁定在 L26 上，双击 reset 行 Location 列，输入锁定位置即可，约束窗口输出的约束为：

```
NET "reset" LOC="L26";
```

这样，我们已经附加了全局周期约束、全局输入输出延迟约束、特定引脚的位置约束，还要附加快、慢速例外约束，这要用到分组约束和时序约束。

5.4.4 附加分组约束和时序约束

【Advanced】选项卡主要包含 TNM/TNM_NET/TIMEGRP 等分组约束和 FROM_TO/TIG/OFFSET 等时序约束，如图 5-24 所示。

1. 附加分组约束

【Advanced】选项卡的上半部分【Grouping】区域为分组约束按钮，第 1 个按钮附加的是 TNM_NET 约束，通过网线把元件分组；第 2 个按钮附加的是 TNM 约束，通过实例（如 FF、PAD、LATCH、RAM）的名称分组；第 3 个按钮把 TNM 约束加到宏或实例化模块上，从而使该模块内部所有层次的同步元件包含在 TNM 定义的分组中；第 4 个按钮利用元

件的输出网线名定义分组，用的是 TIMEGRP 约束；第 5 个按钮是 TPTHRU，用于标识一条路径，以便附加时序约束；第 6 个按钮通过不同的时钟延实现分组。单击以上 6 个按钮弹出的对话框基本一样，对话框的用法也类似，这里以第 4 个按钮为例说明它们的用法。

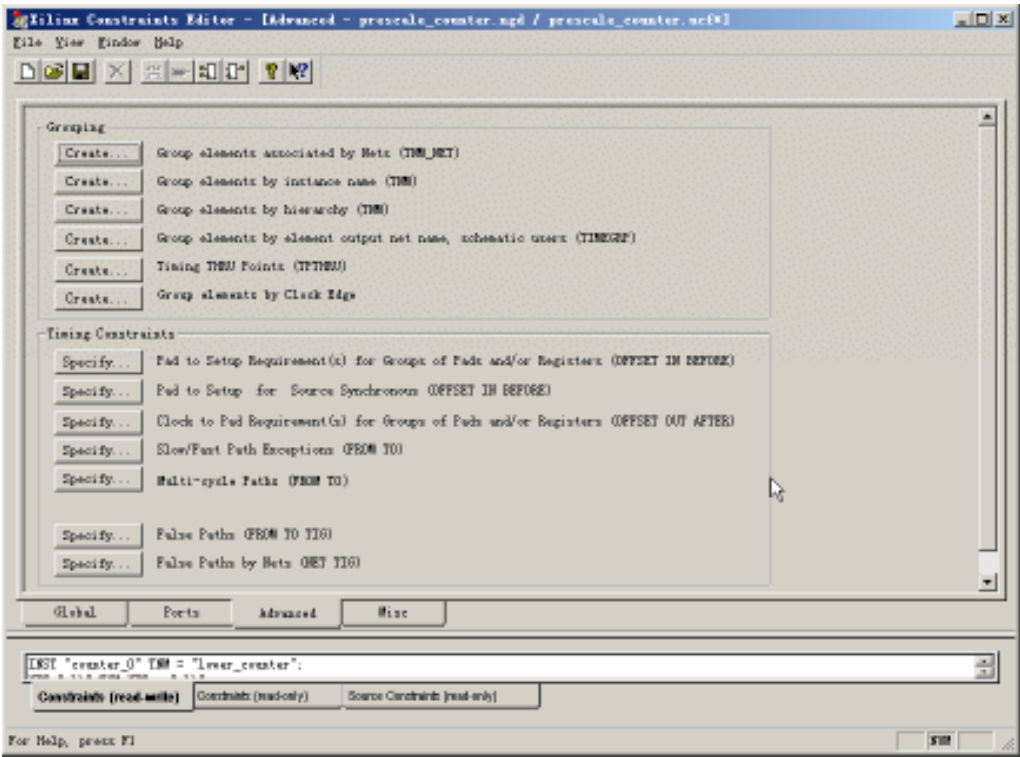


图5-24 【Advanced】选项卡

单击分组约束按钮中的第 4 个按钮，弹出的对话框如图 5-25 所示。

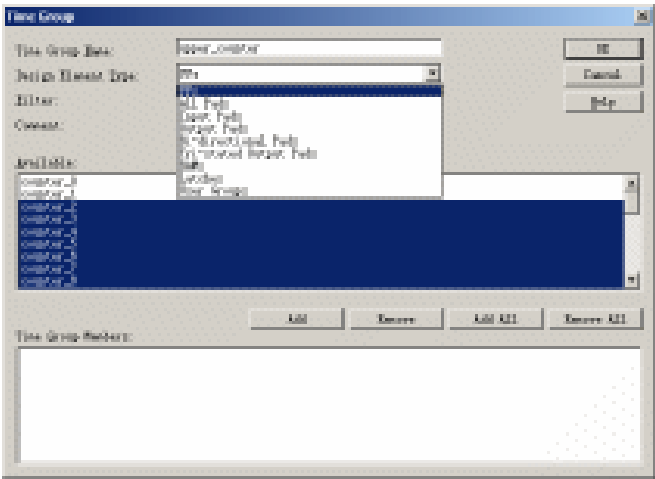


图5-25 【Time Group】对话框

在【Design Element Type】下拉列表中选择元件的类型，属于该类型的所有元件将在

【Available】文本框自动列出。选中构成分组的元件，单击 **Add** 按钮，选中的元件将显示在【Time Group Members】文本框中，然后在【Time Group Name】文本框中填入分组的名字，单击 **OK** 按钮完成操作，从约束窗口可以看到新增的约束。

在同步预定标计数器例子中，高位计数器的寄存器之间为多周期路径，其延迟小于低位计数器时钟周期的 25% 即可，因此需要对这些路径附加慢速例外约束。加约束之前要标识出高位计数器的所有寄存器，参见图 5-25，用 TIMEGRP 约束把 counter_2~counter_31 分成一个分组，名为 upper_counter，新增的约束如下：

```
TIMEGRP "upper_counter" = FFS( "counter_2" "counter_3" "counter_4"
"counter_5" "counter_6" "counter_7" "counter_8" "counter_9" "counter_10"
"counter_11" "counter_12" "counter_13" "counter_14" "counter_15"
"counter_16" "counter_17" "counter_18" "counter_19" "counter_20"
"counter_21" "counter_22" "counter_23" "counter_24" "counter_25"
"counter_26" "counter_27" "counter_28" "counter_29" "counter_30"
"counter_31" );
```

2. 附加时序约束

【Advanced】选项卡下半部分的【Timing Constraints】区域用于附加时序约束，主要有 OFFSET_IN_BEFORE、OFFSET_OUT_AFTER、FROM_TO 和 TIG 约束。和【Ports】选项卡中附加的 OFFSET 约束相比，此处可以附加比较复杂 OFFSET 约束。

例如要把同步预定标计数器最低两位的输出延迟约束到 8ns，可以使用以下方法：首先用分组约束 TIMEGRP 把所有输出 PAD 命名为“all_output_pads”，然后把计数器最低两位的寄存器命名为“lower_counter”，单击【Timing Constraints】区的第 3 个按钮，弹出【Clock to Pad】对话框，如图 5-26 所示。

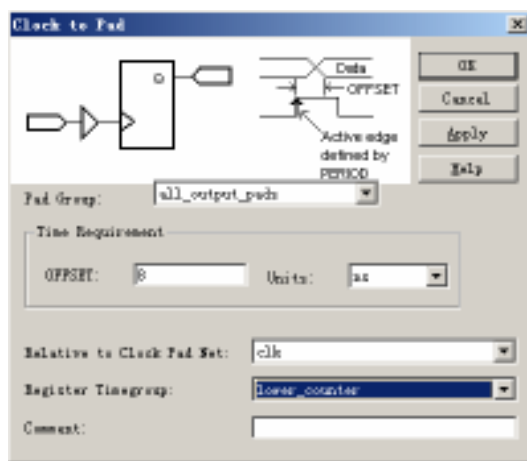


图5-26 【Clock to Pad】对话框

在【Pad Group】下拉列表中选择【all_output_pads】选项，在【Register Timegroup】下拉列表中选择【lower_counter】选项，然后在【Time Requirement】中输入延迟要求 8ns，单击 **OK** 按钮完成输入，生成的约束如下：

```
TIMEGRP "all_output_pads" OFFSET = OUT 8 ns AFTER "clk" TIMEGRP
```

"lower_counter";

其意义是把 OFFSET_OUT_AFTER 约束附加到 all_output_pad 分组中名为 lower_counter 的寄存器上。

【Timing Constraints】区的第4和第5个按钮是附加 FROM_TO 约束，分别用于快、慢速路径和多周期路径，第6个和第7个按钮附加的是 TIG 约束。单击这4个按钮弹出的对话框基本相同，如图 5-27 所示，不同之处在于【Type】的默认选项有所不同，分别是【Explicit】、【Relative to other path specification】或【Mark as false paths】。

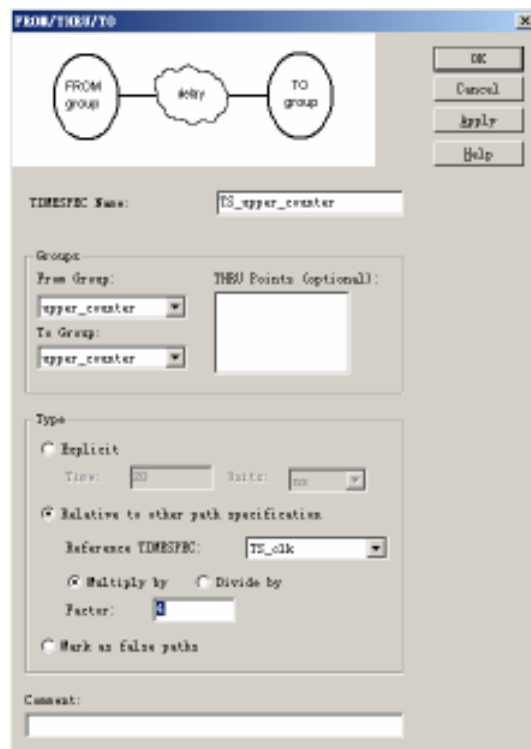


图5-27 【FROM/THRU/TO】对话框

附加 FROM_TO 时，首先在【Groups】中设置起点和终点分组，必要时可以在【THRU Points (optional)】中设定路径的中间点，然后在【Type】中指定时序规范的数值，可以选中【Explicit】，然后用具体数字表示时序规范，或者选中【Relative to other path specification】，然后以另一时序规范数值的倍数或分数表示，最后在【TIMESPEC Name】栏中输入时序规范的名称，单击 **OK** 完成设置。

在同步预定标计数器的例子中，高位的30个寄存器之间的路径是多周期路径，需要附加慢速例外约束。单击【Timing Constraints】区的第4个按钮，弹出【FROM/THRU/TO】对话框，按图 5-27 所示设置各项参数后，产生的约束如下：

```
TIMESPEC "TS_upper_counter" = FROM "upper_counter" TO "upper_counter"
"TS_clk" * 4;
```

5.4.5 附加专用约束

【Misc】选项卡的界面如图 5-28 所示，利用该选项卡可以附加电压、结温和初始值等约束。

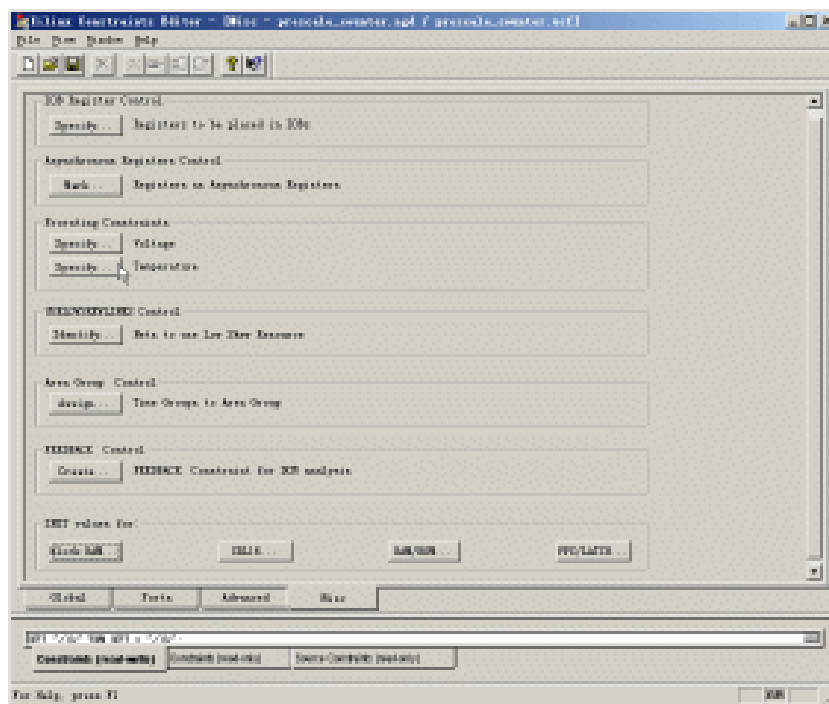


图5-28 【Misc】选项卡

【IOB Register Control】区用于控制输入输出的寄存器是否放到 IOB 中。

【Asynchronous Registers Control】区可将特定的寄存器标记为异步寄存器。

【Prorating Constraints】区允许对工作电压和结温进行设定，默认为最恶劣情况，例如对 1.8V 的 VertexE 器件，默认工作电压为 1.7V，结温为 85°C。

【Area Group Control】用于附加 AREA_GROUP 约束，可以把设计分为不同的物理区域以便进行映射和布局布线，在 Modular Design 和 Incremental Design 流程中有重要作用。

【FEEDBACK Control】用于定义 DCM 分析使用的反馈路径延时。

【USELOSKEWLINES Control】区可以使 PAR 工具对指定网线使用第二全局时钟资源，关于第二全局时钟资源的详细论述请读者参考第 4.4 节。

【INIT Values for】区可以对 BlockRAM、SRL16、RAM/ROM 和 FFS/LATCH 设置初值。

【Misc】选项卡设置比较简单，而且所设置的约束比较少用，因此这里不详细介绍它们的使用方法，有需要的用户可以查阅相关手册。

附加约束可以使用 Constraints Editor，也可以直接编辑 UCF 文件，为了方便比较两者的异同，这里把 Constraints Editor 产生的同步预定标计数器的约束重新整理如下：

```
NET "clk" TNM_NET = "clk";
```

```

TIMESPEC "TS_clk" = PERIOD "clk" 5 ns HIGH 50 %;
OFFSET = IN 20 ns BEFORE "clk";
NET "reset" LOC="L26";
TIMEGRP "upper_counter" = FFS( "counter_2" "counter_3" "counter_4"
"counter_5" "counter_6" "counter_7" "counter_8" "counter_9" "counter_10"
"counter_11" "counter_12" "counter_13" "counter_14" "counter_15"
"counter_16" "counter_17" "counter_18" "counter_19" "counter_20"
"counter_21" "counter_22" "counter_23" "counter_24" "counter_25"
"counter_26" "counter_27" "counter_28" "counter_29" "counter_30"
"counter_31" );
TIMESPEC "TS_upper_counter" = FROM "upper_counter" TO "upper_counter"
"TS_clk" * 4;

```

这些约束与本章开始部分在 UCF 中写的约束用的语句不一样，但是有相同的作用，附加约束的策略也相同，那就是先附加全局约束，再附加快慢速例外约束。

Constraints Editor 能产生的约束通过直接编写 UCF 文件都可以实现，反之则不然，也就是说 Constraints Editor 的功能受到了限制，但是能满足一般设计的需要，而且其优点也很明显，例如操作简便、不要求设计者熟悉 UCF 文件的语法以及触发器、网线、PAD 等元件的命名等，因此 Constraints Editor 是提高工作效率的一个非常有用的工具。

5.5 引脚与区域约束编辑器——PACE

引脚与区域约束编辑器 PACE (Pinout and Area Constraints Editor) 是一个具有图形化界面的约束输入工具，其主要功能如下：

(1) 指定引脚分配

设计者可以使用引脚分配功能指定 I/O 位置、I/O 组 (bank)、I/O 标准和禁止 I/O 分配至特定引脚、使用 DRC 检查 I/O 分配是否正确。

(2) 附加区域约束 (area constraints)

PACE 能够以图形化的方式显示器件资源和引脚的分布，设计者可以编辑区域约束并观察逻辑和引脚之间的连接情况。

(3) 浏览设计层次

PACE 的设计层次浏览器能显示设计的层次，以及各层次的资源占用情况，这对复杂的设计有很重要的作用。

在 ISE 5.1 以前的版本中，附加引脚位置约束和区域约束时确定引脚编号和区域的位置是相当麻烦的，ISE 5.1 集成了 PACE 之后，使用 PACE 就可以方便地观察和编辑输入输出引脚位置约束、区域约束，同时可以指定 FPGA/CPLD 的资源布局以及设计通过位置和区域约束映射到 FPGA/CPLD 的方式。PACE 输入 NGD、UCF、LPF 文件，输出 UCF、LPF 文件。其中 LPF 文件为 PACE 自己使用的文件，它把设计文件和 UCF 文件关联起来，同时包含了分组和颜色等信息。LPF 文件在起动 PACE 时自动读入，在 PACE 中所做的工作也会自动保存在 LPF 文件中，设计者不用直接修改 LPF 文件。

5.5.1 PACE 的用户界面

PACE 的启动方法有两种，一是直接在 Windows 环境中运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【PACE】命令。另一种方法是在 ISE Project Navigator 的【Source in Project】窗口中选中要综合实现的代码，在【Processes for Source】窗口中单击【User Constraints】左边的“+”号，然后双击【Assign Package Pins】或【Create Area Constraints】就可以起动 PACE。需要注意的是，在起动 PACE 之前，应保证设计中有相应的 UCF 文件，否则会提示错误。PACE 的用户界面如图 5-29 所示。

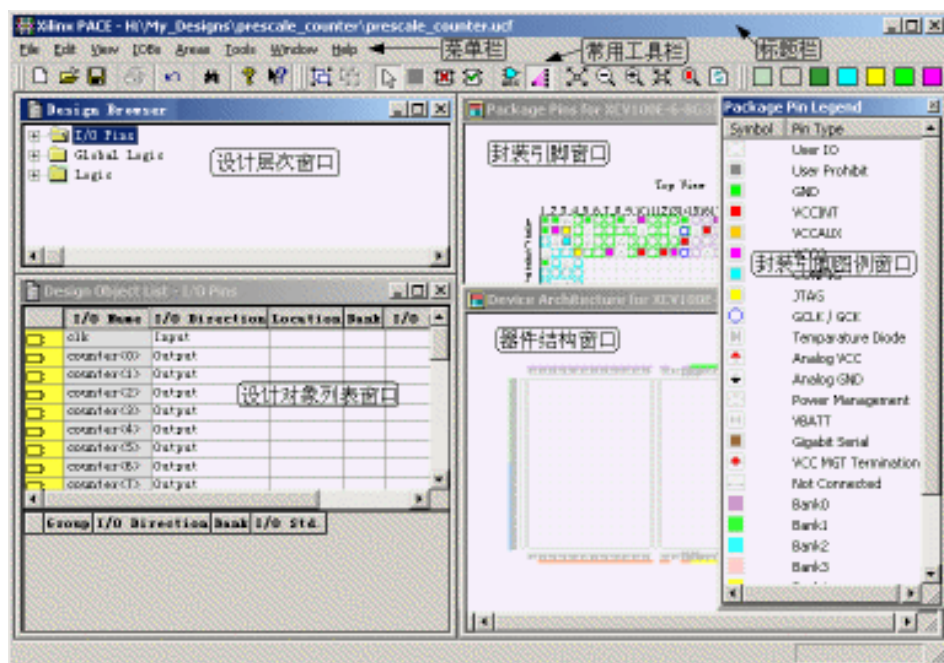


图5-29 PACE 的用户界面

PACE 的用户界面主要由标题栏、菜单栏、常用工具栏、设计浏览器窗口 (Design Browser Window)、设计对象列表窗口 (Design Object List Window)、器件结构窗口 (Device Architecture Window)、封装引脚窗口 (Package Pins Window) 和封装引脚图例窗口 (Package Pins Legend Window) 等部分组成。

1. 菜单栏

菜单栏由一系列包含常用处理命令的下拉菜单组成。

- 文件 (File) 菜单包含了常见文件操作、打印等命令，其中【Read Constraints】命令用于读入约束文件，选中此命令会弹出【Select Constraints file】对话框，所选的约束文件将会读入，其中的 I/O 位置和标准约束、全局逻辑约束和区域约束将会被赋予当前设计的相应的元件，其他约束会被忽略。
- 编辑 (Edit) 菜单包含了常用的编辑操作命令，以及【Group】(组合)、【Ungroup】(取消组合)、【Remove Constraint】(删除约束)、【Remove All Constraints】(删除所有约束)、【Object Properties】(对象属性) 和

【Preferences】（用户选项）等命令。选中【Edit】/【Preferences】命令会弹出图 5-30 所示的对话框，在【Resources】选项卡中可以设置是否显示 IO 组、是否显示差分 IO 对和封装引脚视图为顶视图还是底视图。在【Logic】选项卡中可以设置在器件结构窗口中与逻辑相关的文本是否显示、选中的逻辑是否闪烁以及并在所有视图中是否显示箭头提示。在【Ratsnest】选项卡中可以设置是否显示与逻辑相连的鼠线，其作用与常用工具栏中的工具相同。

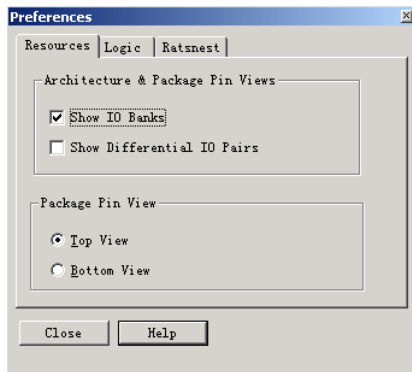


图5-30 【Preferences】对话框

- 视图（View）菜单包含了一些视图显示方面的常用命令，主要包括刷新（Refresh）、缩放（Zoom）、显示【Design Hierarchy】窗口（Design Hierarchy）、显示【Design Object List】窗口（Design Object List）、显示【Package Pins】窗口（Package Pins）、显示【Device Architecture】窗口（Device Architecture）、显示工具栏（Toolbars）和显示状态栏（Status Bar）等。
- IOBs 菜单包含与引脚相关的常用命令，主要包括封装顶视图（Package Top View）、封装底视图（Package Bottom View）、显示差分引脚（Show Differential Pairs）、显示 IO Banks（Show IO Banks）、显示时钟区域（Show Clock Regions）、禁止特定引脚（Prohibit Special Pins）和引脚兼容（Make Pin Compatible with）等命令。
- 区域（Areas）菜单包含了划定区域的命令，有加入长方形（Add Rectangle）、前移区域（Bring Area To Front）、后移区域（Push Area To Back）、限定到网格（Snap To Grid）、允许重叠（Allow Overlap）、区域裕量设定（Area Padding）和编辑约束（Edit Constraints）等命令。
- 工具（Tools）菜单包含了模式选择等命令，主要有选择模式（Select Mode）、划定区域模式（Assign Area Mode）、禁止模式（Prohibit Mode）、允许模式（Allow Mode）、Run DRC、SS0 分析和时钟分析等命令。工具菜单中包含的模式选择功能与常用工具栏中的相同，具体使用方法请参考常用工具栏的说明。
- 窗口（Window）菜单由一些通用窗口调整与排列命令组成。
- 帮助（Help）菜单包含了 PACE 在线帮助和版本信息等。

2. 常用工具栏

-  (Group) 按钮的功能是使选中的 I/O 成为一个组。具体的使用方法是在 DOL (Design Object List) 中选中若干个 I/O 引脚，单击  按钮，然后给组命名。
-  (Ungroup) 按钮的功能是取消分组。
-  (Normal Select Mode) 按钮的功能是使鼠标箭头恢复到正常选择模式。
-  (Assign Area Constraint Mode) 按钮用于创建一个区域约束。在设计浏览器窗口的“Logic”文件夹下选中一组元件，单击  按钮，然后在器件结构窗口中选定一个区域，这样就创建了一个区域约束，所选的元件将被放置在该区域中。
-  (Prohibit Mode) 按钮的功能是标识器件某一区域的资源不能用于布局布线。
-  (Allow Mode) 按钮使被标识为不能使用的资源可重新用于布局布线。
-  (Toggle Labels) 按钮用于隐藏或显示标号。
-  (Toggle Rubberbands) 按钮用于确定是否在器件结构窗口中显示元件之间连接关系。
-  中的工具用于调整器件结构窗口的显示，其作用依次为显示器件整体、缩小、放大、放大一个区域、放大选中的元件和刷新所有打开的窗口。
-  等 12 个不同颜色的按钮为“Assign Color”按钮，用于改变器件结构窗口中选中区域的颜色，以便区分不同的区域，也可以用于标识设计浏览器窗口和 DOL 窗口的不同元件。

3. 设计浏览器窗口 (Design Browser Window)

使用设计浏览器窗口可以方便地浏览整个设计包含的资源。

设计浏览器窗口包含“I/O Pins”、“Global Logic”和“Logic”三个文件夹，双击展开“I/O Pins”文件夹会显示设计包含的 I/O，这些 I/O 同时会在 DOL 窗口中列出。设计者可以在设计浏览器窗口中选中一个 I/O，拖放到器件结构窗口或封装引脚窗口中。“Global Logic”文件夹中包含了设计用到的 BUFG、BRAM、MULT、PPC405、GT、DLL 和 DCM 等全局逻辑。“Logic”文件夹以层次化的形式显示了设计用到的逻辑。

4. 设计对象列表窗口 (DOL Window, Design Object List Window)

在设计浏览器窗口中选中某一层次或一个组时，设计对象列表窗口会列出所包含的元件。

根据设计浏览器窗口选中的文件夹的不同，设计对象列表包含的内容以及可以进行的操作也不同。选中“I/O Pins”文件夹时，DOL 窗口会列出 I/O 的名字、方向、位置、组和 I/O 标准等信息，其中 I/O 方向由 NGD 文件获得，I/O 的具体位置和所在的组在把 IO 拖放到器件结构窗口后会自动显示。I/O 标准可以在相应的下拉列表框中选择。

I/O 位置也可以在下拉列表框中选择，可供选择的选项有：T (TOP)、L (LEFT)、B (BOTTOM)、R (RIGHT)、TL (TOP-LEFT)、BR (BOTTOM-RIGHT)、LT (LEFT-

TOP)、RT (RIGHT-TOP)、TR (TOP-RIGHT)、LB (LEFT-BOTTOM)、BL (BOTTOM-LEFT) 和 RB (RIGHT-BOTTOM)。

在 DOL 窗口中也可以定义组,方便引脚的放置,方法是选中一个或多个 I/O 引脚,选中【Edit】/【Group】命令或单击常用工具栏中的  按钮,然后在 DOL 窗口的下方写入组的名字即可。把一组 I/O 拖放至器件结构窗口时,I/O 的摆放顺序由菜单栏中的【IOB】/【Default Distribute Action】命令控制。

5. 器件结构窗口 (Device Architecture Window)

器件结构窗口以图形化的形式显示了所选器件的资源,是 PACE 的主要工作窗口。设计者可以把选中的 I/O 放到器件结构窗口,完成 I/O 位置约束,也可以在器件结构窗口为特定逻辑划定布局布线区域,完成区域约束的附加。

6. 封装引脚窗口 (Package Pins Window)

FPGA/CPLD 有 BGA (球状矩阵排列封装)、PGA (引脚网格阵列封装)、FBGA (纤细球状矩阵排列封装)、PQFP (塑料方形扁平式封装) 等多种封装形式,封装引脚窗口显示了所选封装的引脚的空间分布。PACE 用不同的符号表示不同种类的引脚,封装引脚图例窗口说明了这些符号代表的引脚类型。

在封装引脚窗口中,未使用的引脚用中心为白色的符号表示,已经使用的引脚的中心将填以特定的颜色。在【IOB】菜单中可以选择顶视图或底视图、控制差分引脚对的显示以方便查找。设计者可以从设计浏览器窗口或 DOL 窗口把引脚拖放到封装引脚窗口中。

7. 封装引脚图例窗口 (Package Pins Legend Window)

封装引脚图例窗口说明了封装引脚窗口中各种引脚符号的意义。

5.5.2 附加区域约束

下面以预定标计数器为例说明附加区域约束的方法。

在 ISE Navigator 中启动 PACE,然后在设计浏览器窗口中展开“Logic”文件夹,设计用到的资源显示在 DOL 窗口中。假设要把计数器放在 FPGA 的左边,在 DOL 窗口中选中“prescale_counter”,其中包含了计数器用到的 FF、函数发生器、IOB 等资源,然后单击  按钮,用鼠标在器件结构窗口中划出用于布局的区域,如图 5-31 所示。保存后可以在 ISE Navigator 中打开 UCF 文件观察生成的约束。

在 PACE 中可以用添加多个长方形的办法来划出复杂的区域。例如在图 5-31 中,选中已经划定的区域,点击鼠标右键可以添加新的长方形,如此类推可以划出任意复杂的区域。如果要修改复杂区域中的某个长方形,需要单击该长方形两次以选中它。

在 PACE 中也可以划定禁止布局布线的区域,方法是单击  按钮,然后用鼠标在器件结构窗口中划出不想用于布局布线的区域。单击  按钮,用鼠标在禁止布局布线的区域中间重新划定能用于布局布线的区域。

利用【Areas】菜单可以进行区域约束相关的操作,其中【Add Rectangle】命令可以附加一个用于区域约束的长方形,【Bring Area To Front】(【Bring Area To Back】)命令用于使选中的区域前移(后移),以便观察。【Snap To Grid】用于强制划定的长方形的边界必须在

CLB 的边界处。【Allow Overlap】命令用于确定是否允许属于不同区域约束的长方形重叠。划定区域时，所包含的资源应不小于实现逻辑所需的资源再加上一定的裕量，【Area Padding】命令用于确定该裕量，当选中这一命令时，会出现【Area Padding】对话框，设计者可以在对话框中输入与实现特定逻辑所需资源相比应增加的百分比，默认值为 25%，这样 PACE 可以保证划定的区域不会过小。

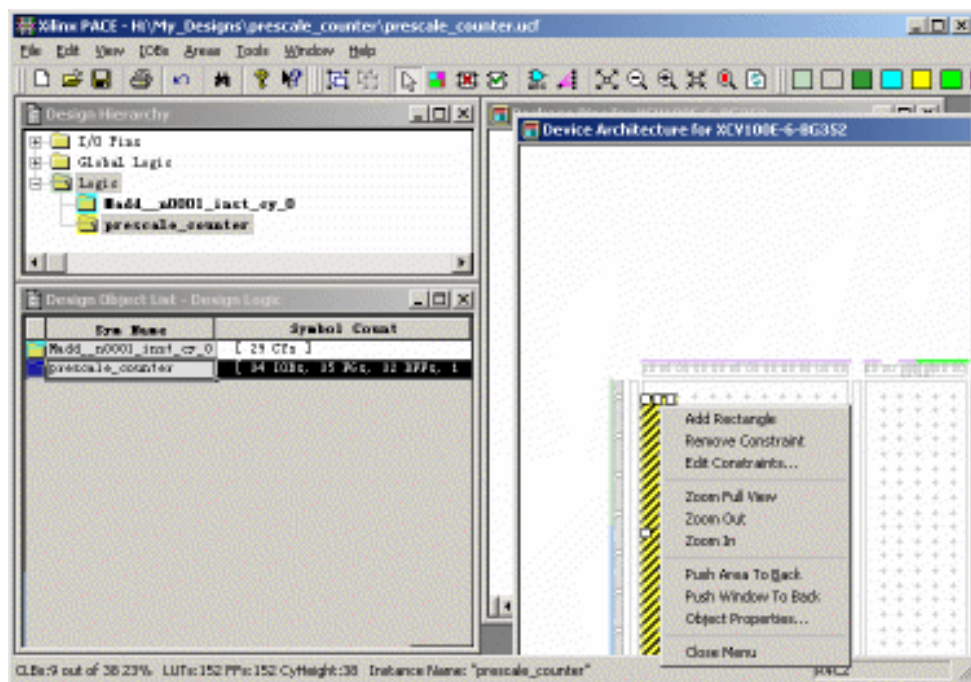


图5-31 划定布局区域

5.5.3 附加 I/O 引脚约束

用 PACE 附加 I/O 引脚约束具有方便、直观的优点，这里以预定标计数器为例简要说明其使用方法。由于在附加区域约束时设计被放在 FPGA 的左边，相应地我们把 I/O 也约束到 FPGA 的左边。

起动 PACE 后，在设计浏览器窗口中选中“ I/O Pins ”文件夹，DOL 窗口会列出设计用到的 I/O。在 DOL 窗口中依次选中各个引脚，拖放到器件结构窗口中特定的引脚上就实现了引脚位置约束，在 DOL 窗口还可以设定引脚采用的标准。

为了避免手工逐个放置引脚的麻烦，可以把 counter<0>~counter<31>分为两个组，这样一次可以放置多个引脚。方法是在 DOL 窗口中选中 counter<0>和 counter<1>，然后单击  按钮，在 DOL 窗口的下方把组命名为“ lower_counter ”，用同样的方法把 counter<2>~counter<31>合成名为“ upper_counter ”的组。放置引脚时，选中“ lower_counter ”或“ upper_counter ”，拖放到器件结构窗口，可以把组内所有的引脚一次放好。一次放置多个引脚时，用【IOB】/【Default Distribute Action】命令来控制 DOL 中从上到下排列的这些引脚在器件结构窗口中是从上到下、从下到上，还是其他排列方式。

引脚位置和引脚标准约束附加完毕后,需要检查 I/O 配置的合法性,选中【Tools】/【Check Pin Assignment】命令,PACE 会进行 I/O 标准合法性、I/O 标准的 VREF 是否兼容、VCCO 是否兼容等项目的检查,检查结果显示在【Check Pin Assignment】窗口中。

需要注意的是,为了使附加的区域约束或引脚约束生效,必须保存所作的修改。

5.6 约束文件

本节主要介绍 ISE 支持的 3 种约束文件的概念以及基本语法规则。

5.6.1 约束文件的概念

约束文件有 3 种,包括用户约束文件(UCF)、网表约束文件(NCF)和物理约束文件(PCF),这里对这些类型的约束文件的特点、在 FPGA/CPLD 设计流程中的位置和它们之间的关系作详细说明,使读者对约束文件有一个比较清楚的认识。

UCF 文件是设计者使用得最多的文件,它是 ASCII 码文件,描述了逻辑设计的约束,可以用文本编辑器或 Xilinx Constraints Editor 编辑。UCF 文件中的约束影响所设计的逻辑在目标器件中的实现方式。UCF 文件在 FPGA/CPLD 设计流程中的位置如图 5-32 所示。NGDBuild 读入 UCF 文件,输出 NGD 文件,UCF 文件中的约束被包含在 NGD 文件中。

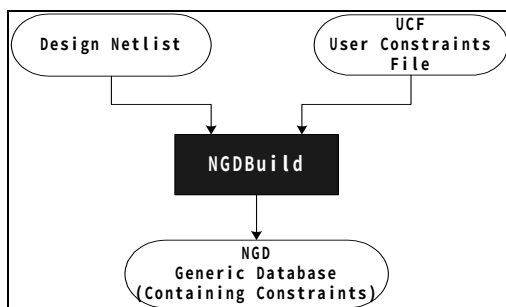


图5-32 UCF 文件在 FPGA/CPLD 设计流程中的位置

图 5-32 对应 ISE 中使用 XST 综合的 Translate 步骤。如果使用的是其他综合工具,综合后输入到 NGDBuild 的是网表文件 EDF 和网表约束文件 NCF。例如使用 Synplify Pro 综合时, Synplify Pro 根据综合约束 SDC 文件,综合设计逻辑产生网表文件 EDF,同时将所有约束条件翻译成 NCF 文件,这两个文件将作为 Translate 等后续流程的输入,通过 NCF 文件约束布局布线过程的行为。NCF 文件的语法和 UCF 文件相同,NCF 文件与用户约束文件 UCF 的最主要区别是,UCF 文件由用户输入,NCF 文件由综合工具产生。UCF 约束的优先级比 NCF 文件高,当两者发生冲突时,以 UCF 约束为准。

网表读入到 Xilinx Development System 时生成的 NGD 文件会含有逻辑约束,这些约束来源于原理图、HDL 文件、UCF 文件和 CAE 工具产生的 NCF 文件等。MAP 读入这些逻辑约束,一部分用于做映射,其他的将被转换为物理约束,被写入到物理约束文件(PCF)中。

PCF 文件为 ASCII 文件,包含两个部分:第一部分是映射(Map)产生的物理约束,第

二部分是用户输入的物理约束。这个顺序使用户输入的物理约束的优先级较高，当两种物理约束发生冲突时，用户输入物理约束将覆盖映射产生的物理约束。

PCF 文件的映射生成部分以关键字“SCHEMATIC START”开头，“SCHEMATIC END”结尾，两种关键字都独占一行。用户输入的物理约束在 SCHEMATIC END 之后，在 SCHEMATIC START 之前、SCHEMATIC START 和 SCHEMATIC END 之间输入的物理约束将被忽略。用户约束可以直接手动输入物理约束，也可以通过 FPGA Editor 间接输入。PCF 可用于 PAR、FPGA Editor、Timing Analyzer、NGDAnno 和 BitGen 等工具中。

对所有的约束文件（NCF/UCF/PCF），与保留字相同的实例或变量名会产生错误信息，除非名字在“”号中，因此在写约束时，用“”号把实例名、网线名、变量名等包括起来是个好的设计习惯。

一般情况下，用户约束应该在 UCF 文件中输入，不要直接修改 NCF 文件和 PCF 文件。

5.6.2 UCF、NCF 文件的基本语法规则

1. 通用规则

- UCF、NCF 文件是大小写敏感的，标识符（即设计中的对象的名称，如网线名等）的大小写必须与网表中的名字一致。但是约束的关键字（如 LOC、PERIOD、HIGH、LOW 等）可以是大写、小写或大小写混合。
- 语句以分号（;）结束。
- 如果一个语句的长度超过一行，不用加续行的符号，因为分号表示了语句的结束。
- 可以使用“#”或“/* */”附加注释。
- 语句之间不区分先后次序。
- Xilinx 建议 NET 或 INST 的名字使用“”号括起来，以避免错误，但是不作强制要求。另外，含有波浪线的反相信号名（如~OUTSIG1）必须使用双引号括起来。

2. 语法

UCF/NCF 文件中约束的语法如下：

```
{NET|INST|PIN} "full_name" constraint;
```

- full_name 是所指约束对象的名字，这个名字包含了对象所在层次的描述。
- constraint 为约束的具体描述，其形式与在原理图中附加 Attribute 时相同，例如 LOC=P38 或者 FAST 等。

3. 使用保留字

在所有的约束文件（UCF/NCF/PCF）中，不能使用与保留字相同实例或变量名，除非这个名字用双引号括起来。把所有实例或变量名用双引号括起来是一个好习惯。

例如，以下的约束是不正确的，因为 net 是保留字。

```
NET net OFFSET=IN 20 BEFORE CLOCK;
```


建议使用下面的方式输入约束：

```
NET "net" OFFSET=IN 20 BEFORE CLOCK;
```

或者

```
NET "$SIG_0" OFFSET=IN 20 BEFORE CLOCK;
```

反相信号（例如~OUTSIG1）必须包含在双引号中，如以下语句：

```
NET "~OUTSIG1" OFFSET=IN 20 BEFORE CLOCK;
```

4. 通配符

在约束语句中可以使用通配符“*”和“？”，“*”可以代表任何字符串，字符串的长度为 0 或更长。“？”代表一个字符。

当用于网线名时，使用通配符可以选择一组符号，这些符号的输出网线与指定的字符串或模式匹配。例如以下的约束提高了与特定网线相连的 PAD 的速度，这些网线的名字以任何字符开头、包含字符串“AT”并以一个字符结束。

```
NET "*AT?" FAST;
```

以下的约束把一组 ROM 初始化为特定的十六进制数。

```
INST "$1I3*/ROM2" INIT=5555;
```

在位置约束中，可以在行号或列号中使用通配符。例如下面的约束把 loads_of_logic 层次中的所有实例放在第八列的 CLB 中。

```
INST "/loads_of_logic/*" LOC=CLB_r*c8;
```

通配符可以用在“.”之后，例如 CLB_R1C3.*。

5. 遍历设计层次

在 UCF/NCF 文件中可以使用通配符“*”来遍历设计中的特定层次。例如：

- * 遍历所有层次。
- level1/* 遍历 level1 及以下层次中的模块。
- level1/*/ 匹配 level1 中的模块，但不遍历更低层的模块。

下面以图 5-33 所示设计层次为例，介绍通配符的作用。

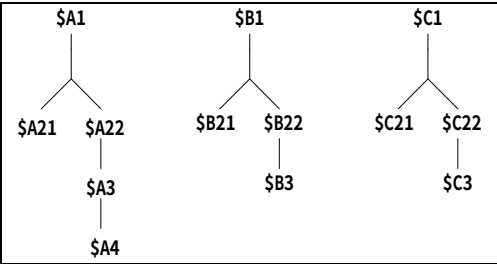


图5-33 示例层次结构

INST *	所有符号
INST /*	所有符号
INST /*/	<\$A1, \$B1, \$C1>
INST \$A1/*	<\$A21, \$A22, \$A3, \$A4>

```

INST $A1/*/          <$A21, $A22>
INST $A1/*/*         <$A3, $A4>
INST $A1/*/*/*       <$A3>
INST $A1/*/*/*/*     <$A4>
INST $A1/*/*/*/*/*   <$A4>
INST /*/*22/         <$A22, $B22, $C22>
INST /*/*22          <$A22, $A3, $A4, $B22, $B3, $C3>

```

6. 文件名

在默认情况下，NGDBuild 读入与输入设计同名的、扩展名为“ucf”的约束文件。但是，运行 NGDBuild 时也可以使用 -uc 选项指定与设计不同名的约束文件。NGDBuild 自动读入与输入 XNF、EDIF 文件放在同一目录下的、与设计同名的、扩展名为“ncf”的约束文件。实现工具（NGDBuild、MAP、PAR 等）要求约束文件名的后缀为小写。

5.7 小结

为了提高设计的工作频率、获得正确的时序分析报告、确定 FPGA/CPLD 引脚位置，需要在综合实现阶段加入约束，通常的做法是设计者编写约束文件并输入到综合实现工具，在 FPGA/CPLD 综合实现时指导逻辑映射、布局、布线。

约束的种类和数目非常多，但是以时序约束和分组约束最为常用，本章重点介绍了时序约束和分组约束的基本概念、语法，然后详细介绍了两个约束输入工具：约束编辑器（Constraints Editor）和引脚与区域约束编辑器（PACE）的使用方法。通过本章的学习，应该掌握附加时序约束的一般策略，就是先附加全局约束，然后附加快慢速例外。全局约束主要包括周期约束、偏移约束，快慢速例外通过附加专门约束实现。为了方便附加约束，需要使用分组约束把元件、网线进行分组。使用 Constraints Editor 附加约束具有操作简便、不要求设计者熟悉 UCF 文件的语法以及触发器、网线、PAD 等元件的命名等优点，是提高工作效率的一个非常有用的工具。使用 PACE 就可以方便地观察和编辑引脚位置约束、区域约束，同时可以指定 FPGA/CPLD 的资源布局以及设计通过位置和区域约束映射到 FPGA/CPLD 的方式，非常实用。

5.8 问题与思考

1. 附加约束的主要目的有哪些？约束的主要种类有哪些？
2. 附加时序约束的一般策略是什么？
3. 使用 Constraints Editor 附加约束与直接编辑 UCF 文件相比，有什么优缺点？
4. PACE 有哪些主要作用？

第6章 辅助设计工具

综合 (Synthesize) 完成后, 接下来是实现 (Implement Design) 的过程。实现过程由翻译 (Translate)、映射 (Map)、布局和布线 (Place and Route) 等环节组成。一般情况下, 实现过程不需要人为干预, 所有操作由工具自动完成, 实现工具只是根据设计者提供的用户约束文件对设计进行翻译、映射和布局布线等操作。其实, 为了达到更好的设计性能, 需要对这些过程进行有效的控制, Xilinx 公司也提供了一整套的工具来帮助用户实现这些目标。

本章就实现过程中用到的时序分析器 (Timing Analyzer)、布局规划器 (Floorplanner) 和 FPGA 底层编辑器 (FPGA Editor) 的使用方法进行说明。

时序分析器可以帮助设计者对设计中的时序冲突以及时序约束等进行细致的检查, 找到设计中的时序瓶颈, 然后设计者可以通过一些方法对时序进行调整。

布局规划器可以对设计元件的物理位置进行约束, 以提高设计的密度和速度等性能。

FPGA 底层编辑器可以对底层单元进行独立的布局和布线, 可以在设计完成后对设计进行局部的调整, 该工具还可以提供一些调试的方法。

本章主要内容如下:

- 时序分析器的使用方法;
- 布局规划器的使用方法;
- FPGA 底层编辑器的使用方法。

6.1 时序分析器——Timing Analyzer

时序分析器为 FPGA 和 CPLD 的设计提供了一种高效、灵活地进行静态时序分析的方法。在时序分析器中可以通过简单的菜单方式对整个设计进行时序分析。同时还提供了一种使用目录树方式组织的时序报告, 在这个报告中对各种时序问题进行了分类, 有利于用户对特殊路径进行查找。在时序分析器中, 每一个显示的路径都可以和综合器 (Synplify) 或者布局规划器进行时序的交叉探查 (Cross-Probing)。通过这种方式可以和其他设计工具进行协同工作, 极大地方便了关键路径的改进, 也有助于用户理解整个设计的时序特性。在 ISE 开发套件中, 时序分析器和布局布线工具对设计时序的计算使用了相同算法, 这增强了整个设计流程的连贯性以及继承性。

对于 FPGA 器件进行时序分析主要是基于现有的时序约束和用户定义的时序路径之上, 而对于 CPLD 器件进行时序分析主要是基于系统定义的各种路径之上, 两者之间略有差异。就设计流程来说, 时序分析可以在 FPGA 器件映射以后或者布局布线以后进行, 而 CPLD 器件进行时序分析必须在布局布线以后。

时序分析器只可以用来分析由组合逻辑和触发器等构成的同步系统。时序分析器可以分

析给定路径的各种延时信息，也可以用来分析用户的时序约束是否满足，可以组织和显示路径的周期数量、设计的最大延时等信息，同时也可以对同一设计在不同速度等级下的性能提供一个快速的分析对比。

6.1.1 时序分析器的用户界面

时序分析是 FPGA/CPLD 设计流程的一个重要组成部分，当一个设计项目经过了综合以后，就可以通过某种方式直接启动时序分析工具进行时序分析。在 ISE 集成开发环境中，如果从模块视图（Module View）中选中了一个设计模块，那么在相应的当前资源操作窗口中的实现阶段就可以启动时序分析器，如图 6-1 所示。使用鼠标单击不同的操作选项就可以启动相应阶段的时序分析，不同的阶段需要的输入文件由 ISE 集成环境自动生成。

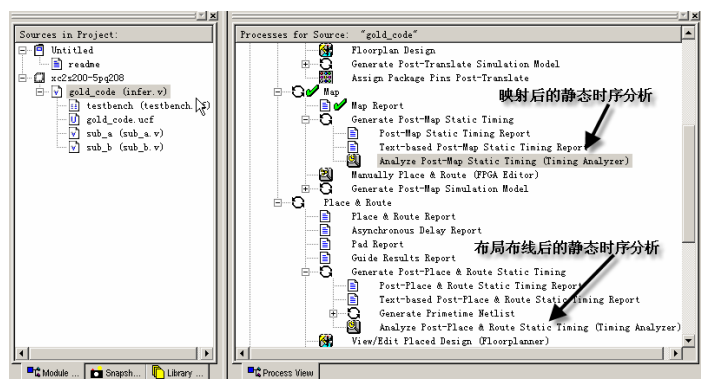


图6-1 ISE 中启动时序分析的方法

例如，为了启动映射后的时序分析过程，需要在图 6-1 所示的界面中使用鼠标左键双击【Analyze Post-Map Static Timing (Timing Analyzer)】选项，这样就可以打开时序分析器的用户界面，如图 6-2 所示。在这个用户界面中包括几个不同的区域，左边部分为时序报告的层次，右边部分为时序分析的详细描述，工具栏中有专门用来做时序分析的工具按钮，时序报告窗口中蓝色的超级链接表示可以与布局规划器进行交叉探查。



图6-2 时序分析器用户界面

6.1.2 时序分析器的作用及设计流程

下面简要介绍时序分析器的作用及其设计流程。

时序分析器的作用

反馈环路

如果在设计中使用了异步反馈环路，可能会引起时序误报，而事实上在进行时序分析时很多误报的延时路径并不存在。一般情况下，触发器使用异步复位、置位都会产生反馈环路，使用锁存器结构也会引入反馈环路。

为了正确地进行时序分析，需要排除这些有可能产生误报的特殊网线。例如，在对一个使用了异步复位逻辑的状态机进行时序分析时，需要在时序报告中将包含复位逻辑的所有路径全部排除，这样才会得到正确的分析结果。

检查时序约束

如果在设计中使用了时序约束，那么就可以通过时序分析器来检查约束是否满足，如果约束不能满足，就可以通过时序分析器与布局规划器的交叉探查功能对设计的局部布局进行调整，使得设计满足要求。

分析并报告设计中的时钟抖动等信息

时钟抖动是指一个时钟网线到达源触发器和目标触发器之间时间的差异。因为相对于全局布线资源来说，其他的布线资源可预知性都比较差，所以，如果时钟网络没有使用全局布线资源，一般都会引起大的时钟抖动，当然使用全局布线资源也会有时钟抖动，在进行时序分析时，这些问题都需要考虑在内。时钟抖动将会影响设计对时钟周期的要求。

确定片间延时

为了确定系统级的时钟速率，必须添加外部的延时信息到芯片传输路径上，这样时序分析器在计算路径的延时时就会包括外部的延时，除非设计者指定了芯片之间的延时信息，否则时序分析器并不对片间延时进行分析。

总的来说，时序分析器具有以下特点：

- 与第三方综合工具和布局规划器等工具可以进行时序交叉探查；
- 由时序分析器产生的 XML 格式的时序报告很容易进行浏览以及分析；
- 可以分析现有的所有时序约束是否满足；
- 可以分析用户定义的时钟以及 IO 的时序约束；
- 可以分析用户定义的路径；
- 可以分析用户指定的 PIN TO PIN 时延；
- 时序改进向导可以对用户时序约束失败的路径进行一些时序改进的建议；
- 层次化的时序报告浏览器可以对延时信息等数据进行分类，有利于阅读和观察。

时序分析在设计流程中的位置

在 FPGA 的实现流程中，从多个环节都可以进行静态时序分析，例如在映射完成后、布局完成后或者布线完成后都可以进行时序分析。如果时序分析不能满足，则需要重新修改

源代码或者调整用户的时序约束，时序分析在整个设计流程中的位置如图 6-3 所示。

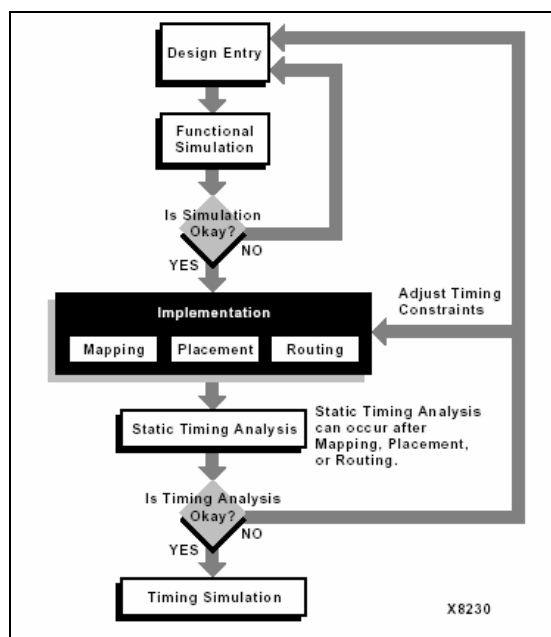


图6-3 时序分析在设计流程中的位置

6.1.3 基本时序路径

为了更好地使用时序分析器，需要对时序分析中涉及到的一些基本概念有所了解，本节首先对时钟抖动进行说明，然后对各种基本时序路径进行简要的说明。

0. 时钟抖动

前面提到了时钟抖动将会影响整体设计对时钟的要求，时钟抖动是指一个时钟网线到达源触发器和目标触发器之间时间的差异。根据时钟到达的先后顺序，有负时钟抖动和正时钟抖动之分。

- Negative Clock Skew

如果目标触发器的时钟信号在源触发器的时钟信号以前到达，这种情况引起的时钟抖动叫做负时钟抖动，负时钟抖动意味着设计中需要的最小时钟周期为路径延时和时钟抖动之和，其原理如图 6-4 所示。

- Positive Clock Skew

如果源触发器的时钟信号在目标触发器的时钟信号以前到达输入端，这种情况引起的时钟抖动叫做正时钟抖动。正时钟抖动意味着最小时钟周期为路径延时与时钟抖动的差。可见正时钟抖动没有对时钟周期产生不利影响，所以在时序分析器中并不计算正时钟抖动，时序分析器认为正时钟抖动为 0，如图 6-5 所示。

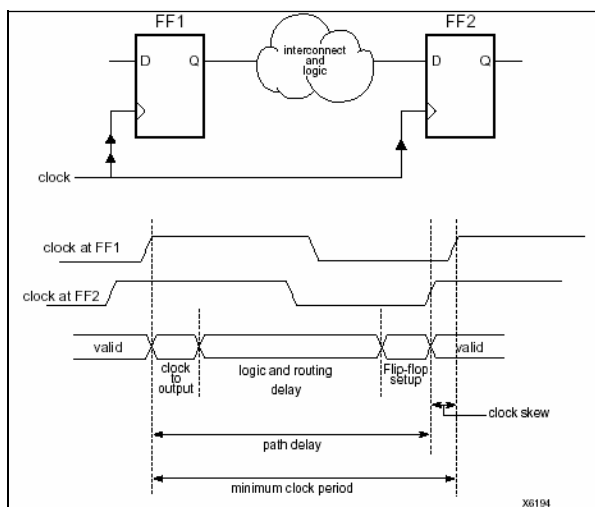


图6-4 负时钟抖动以及相应的时序图

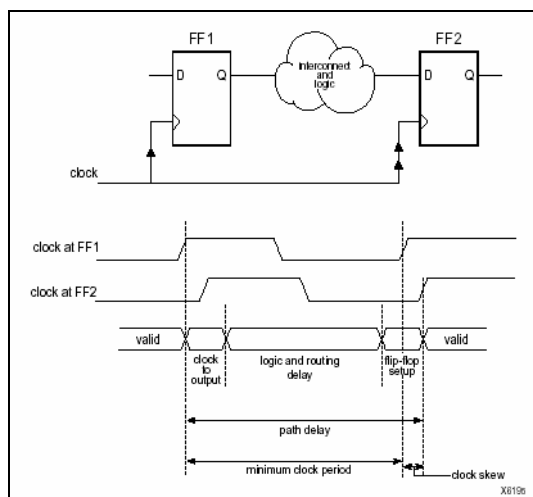


图6-5 正时钟抖动以及相应的时序图

1. 基本时序路径

• Clock to Setup 路径

Clock to Setup 路径从触发器的时钟输入端开始，结束于另外一个触发器的数据输入端。它包括了触发器的 Clock to Q 延时，触发器之间的路径延时（穿越了两个触发器之间的全部的组合逻辑），还有目标触发器的建立时间。Clock to Setup 路径延时是数据从源触发器开始，在下一个时钟沿到达之前穿过中间所有的逻辑和布线的最大时间。当路径上这些触发器使用相同的时钟时，时钟周期大于路径延时的所有设计都能满足设计的频率要求。图 6-6 展示了使用相同时钟的 Clock to Setup 路径。

如果源触发器和目标触发器使用相同时钟的不同时钟沿，并且时钟的占空比不是 50% 时，那么 Clock to Setup 路径延时将被限制在时钟高低电平中最小的那一个时间段中，如图

6-7 所示。

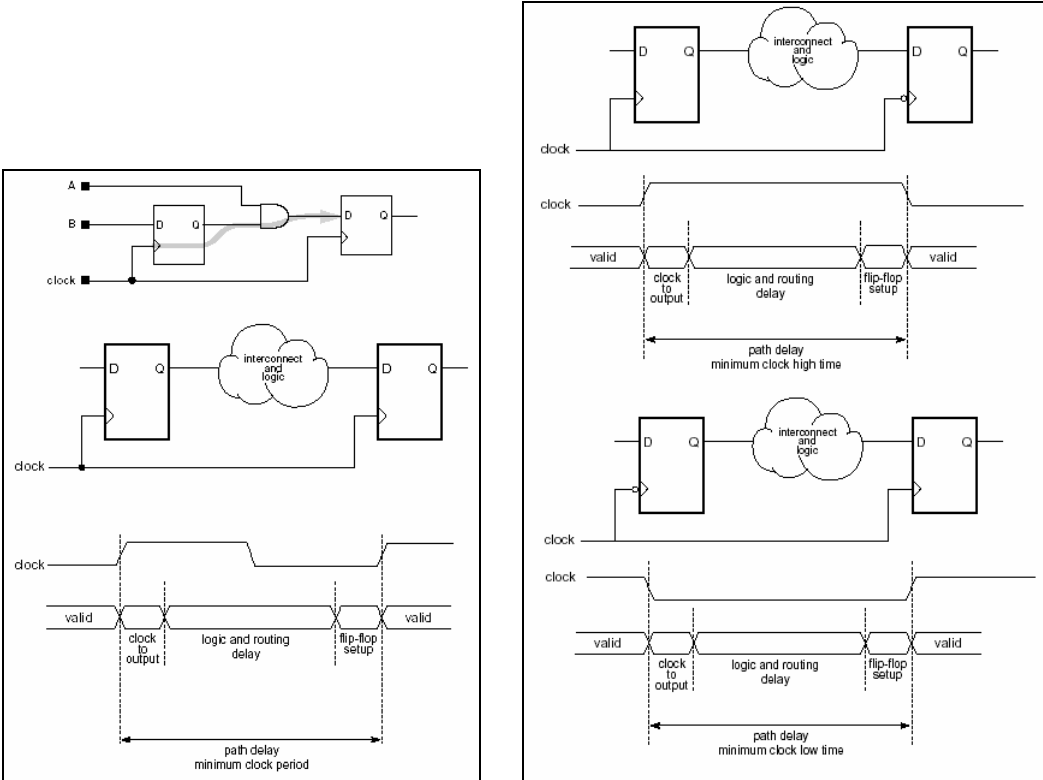


图 6-6 Clock to Setup 路径以及时序图 图 6-7 Clock to Setup 路径以及时序图（使用相同时钟的不同沿）

如果源触发器和目标触发器使用不同的时钟网线驱动，那么目标触发器上的时钟周期必须大于 Clock to Setup 路径延时，同时在时序分析器中将使用目标触发器的周期值作为这个路径允许的最大延时值。这个类型的路径延时如图 6-8 所示。

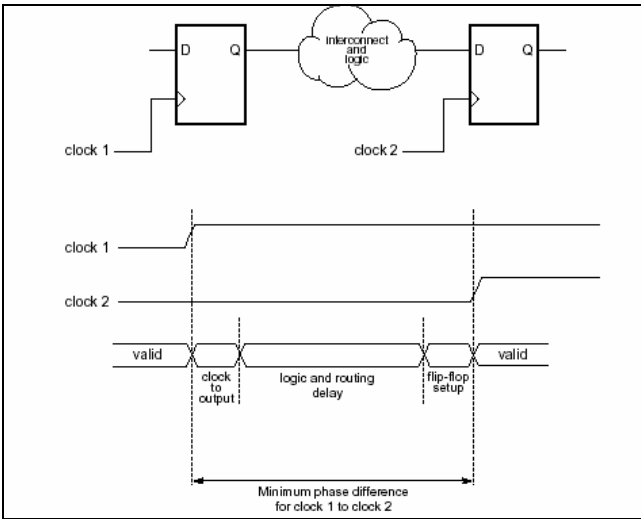


图6-8 Clock to Setup 路径以及时序图（使用不同的时钟驱动）

Clock to Setup 路径延伸到触发器的 Q 端以后，在其他触发器的时钟输入、异步复位和置位端终止，同时这类路径都将在双向端口处终止，如图 6-9 所示。

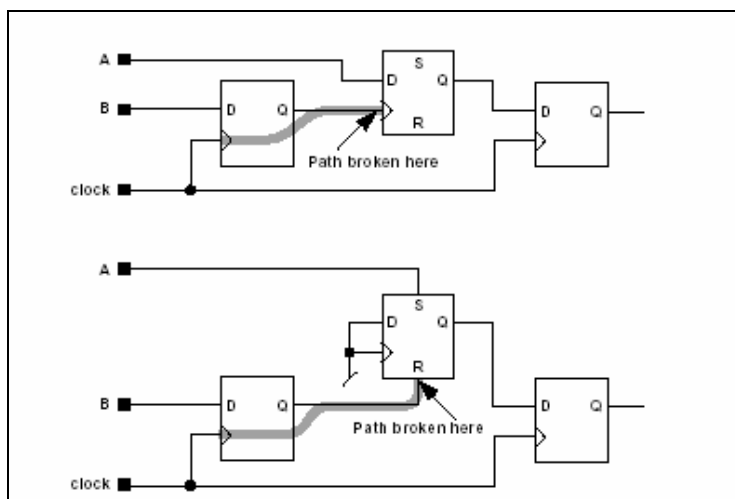


图6-9 Clock to Setup 路径终止

• Clock to Pad 路径

Clock to Pad 路径从一个触发器的时钟输入端开始，经过触发器的 Q 输出端和所有的组合逻辑层，在一个输出 PAD 处终止。它包括触发器 Clock to Q 的延时、从触发器输出到芯片输出的路径延时。Clock to Pad 延时是数据从触发器输出，经过逻辑和布线，最后离开芯片的最大的时间要求。在进行约束编辑时，如果使用了 OFFSET 语句，那么时延计算将自动包含了时钟 BUFFER/ROUTING 延时；如果使用 FROM:TO 约束，如 TIMESPEC TSF2P = FROM : FFS : TO : PADS : 25ns，那么在这种约束中的延时计算将从触发器自身开始，并不包括时钟输入路径。所以推荐对 Clock to Pad 路径进行约束时使用 OFFSET 语句，其路径和时序如图 6-10 所示。

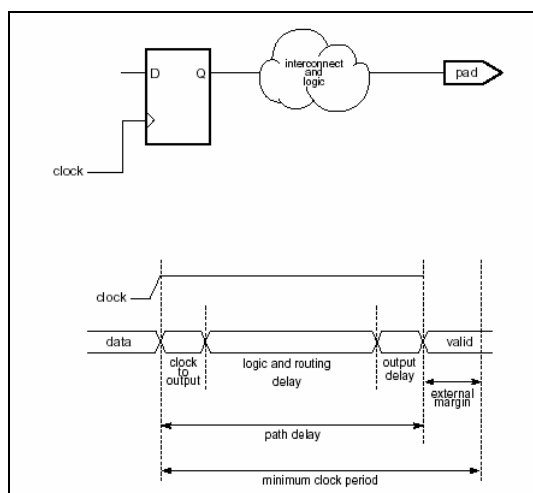


图6-10 Clock to Pad 路径及时序图

Clock to Pad 路径终止于下级触发器的时钟输入端、异步复位和异步置位端，也就是这种情况下并不能构成一个 Clock to Pad 路径，如图 6-11 所示左边部分。Clock to Pad 路径可以通过三态器件的使能端延伸，同时这类路径都会在双向端口处终止，如图 6-11 所示。

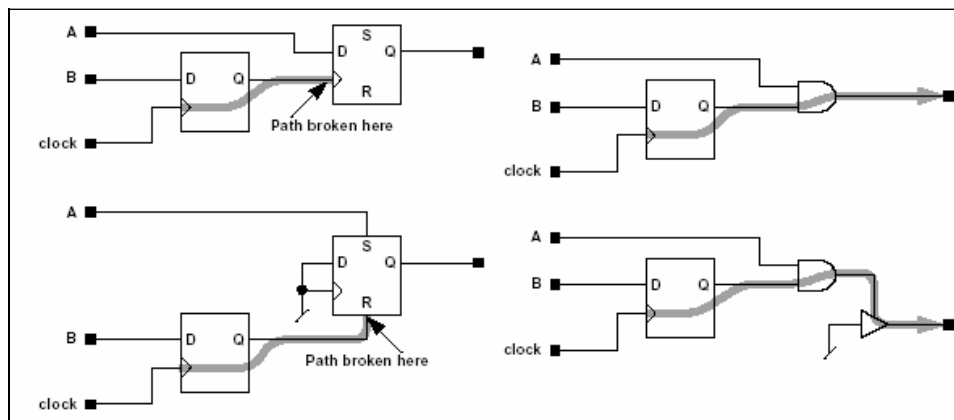


图6-11 Clock to Pad 路径延伸以及终止

• Paths Ending at clock Pin of Filp-Flops 路径

Paths Ending at clock Pin of Filp-Flops 路径从芯片的输入引脚（如果用来计算下一级芯片的时钟输入路径，那么从本级的输出引脚算起）开始，穿过所有的组合逻辑层次，在触发器的时钟输入端或者寄存器的使能端终止。这类路径延时是对到达触发器时钟输入端信号的最大时间要求，可以帮助检验系统级时序设计的问题，如图 6-12 所示。

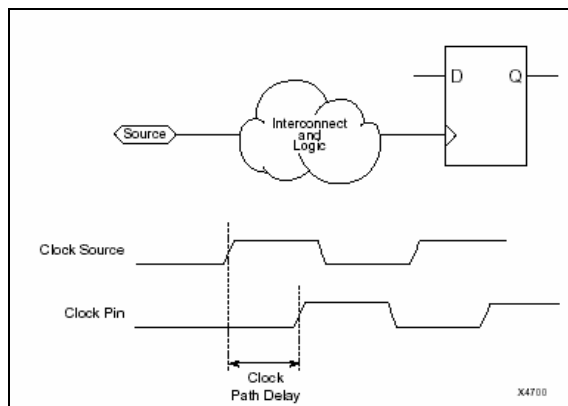


图6-12 Paths Ending at clock Pin of Filp-Flops 路径与时序图

• Setup to Clock at the Pad 路径

一个 Setup to Clock at the Pad 路径从一个输入 PAD 开始，穿过输入驱动器（Input buffer）和所有的组合逻辑层次，在触发器的 D/T 输入端终止。它包括了目标触发器的建立时间 T_{su} ，这个路径不会在触发器处终止，同时这类路径终止于双向口。

全局时钟路径从一个全局时钟的 PAD 开始，穿过全局时钟 BUFFER，在一个寄存器的时钟输入端结束，如图 6-13 所示。

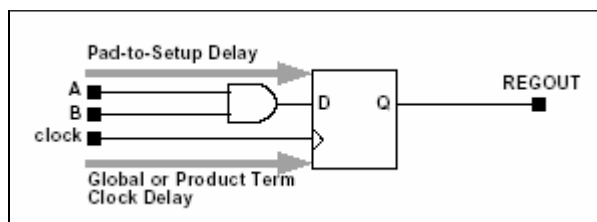


图6-13 Setup to Clock at the Pad 路径

- **Clock Pad to Output Pad 路径**

Clock Pad to Output Pad 路径开始于输入 PAD，通过触发器进行延伸（不包括路径中触发器异步复位位置位输入），结束于输出 PAD。

- **Pad To Pad 路径**

Pad To Pad 路径从芯片的输入 PAD 开始并穿越全部的组合逻辑，结束于芯片的输出 PAD。Pad To Pad 路径延时是数据输入到芯片，经过各种组合逻辑和布线，最后输出芯片的最大的时间要求。这个约束不影响和约束任何一个时钟信号，组合逻辑路径不会穿越触发器，这些路径在双向端口时被中断，如图 6-14 所示。

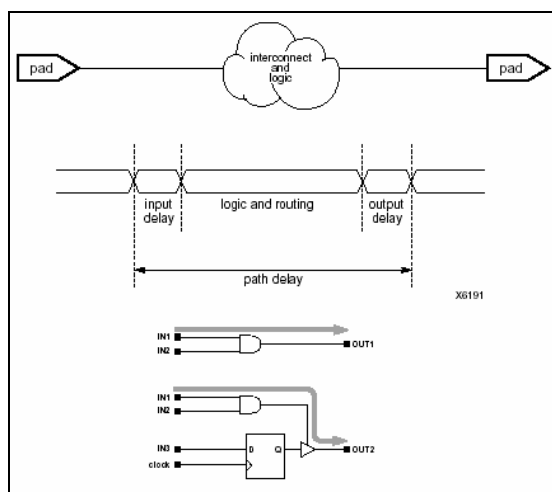


图6-14 Pad to Pad 路径以及时序图

- **Pad To Setup 路径**

Pad to Setup 路径从芯片的一个输入 PAD 开始，在触发器、锁存器的 D/T 输入端结束，或者在 RAM 的输入端结束，这种路径可以穿越输入 BUFFER 和所有的组合逻辑层次，但是不能穿越触发器和双向端口。Pad to Setup 路径延时是数据到芯片的最大时间要求。在编辑设计的约束时，如果使用 OFFSET 语句对 Pad to Setup 路径时序进行约束，则时序分析器在进行延时计算时会自动包含时钟延时，如果使用 FROM:TO 进行约束时，并不考虑时钟的延时。因此，建议在对 Pad to Setup 路径进行约束时使用 OFFSET 语句。Pad to Setup 路径与时序如图 6-15 所示。

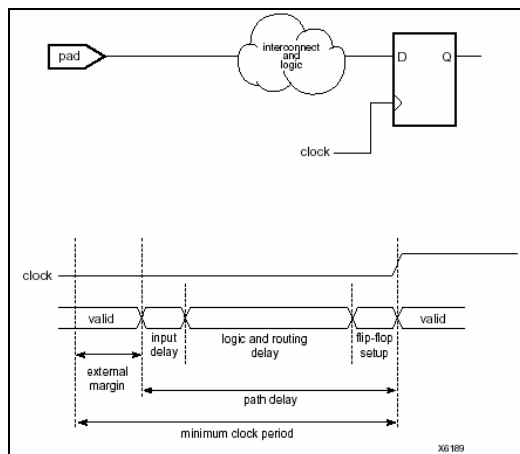


图6-15 Pad to Setup 路径以及时序图

6.1.4 时序分析器的使用方法

使用时序分析器创建时序报告以前，先要加载一个映射过的 NCD 文件（FPGA）或者经过布局布线的 VM6（CPLD）设计文件。在打开设计的对话框中可以选择对应的 PCF 文件，如果没有输入 PCF 文件名称，那么时序分析器将使用和设计文件同名的 PCF 文件作为物理位置约束文件。在加载了设计之后，用户就可以根据自己的需要产生不同类型的时序报告。

1. 基于时序约束的时序分析

使用这种分析方法，时序分析器将报告所有时序约束覆盖的路径。可以通过两种方式产生这种报告，第一种方式是直接单击时序分析器中的  按钮，系统使用默认的设置进行分析，最后得到的分析结果如图 6-16 所示。

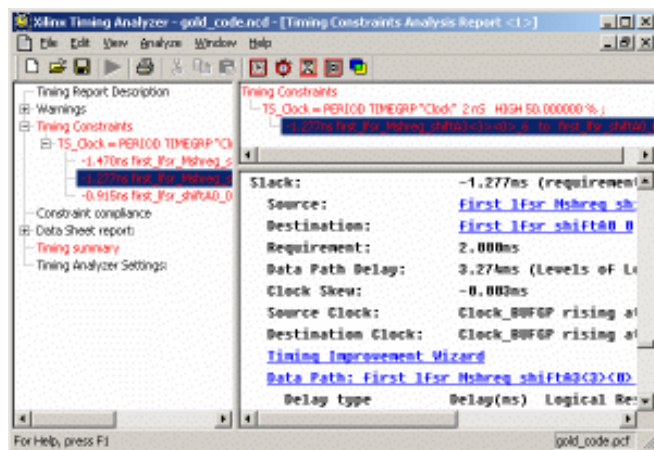


图6-16 基于时序约束的时序分析

第二种方式是从时序分析器的菜单栏中选择【Analyze】/【Against Timing Constraints】命令，会弹出【Analyze against Timing Constraints】（基于时序约束的时序分析）对话框，使

用这个对话框可以对分析过程进行设置。在这个对话框中包括 4 个选项卡，如图 6-17 所示。

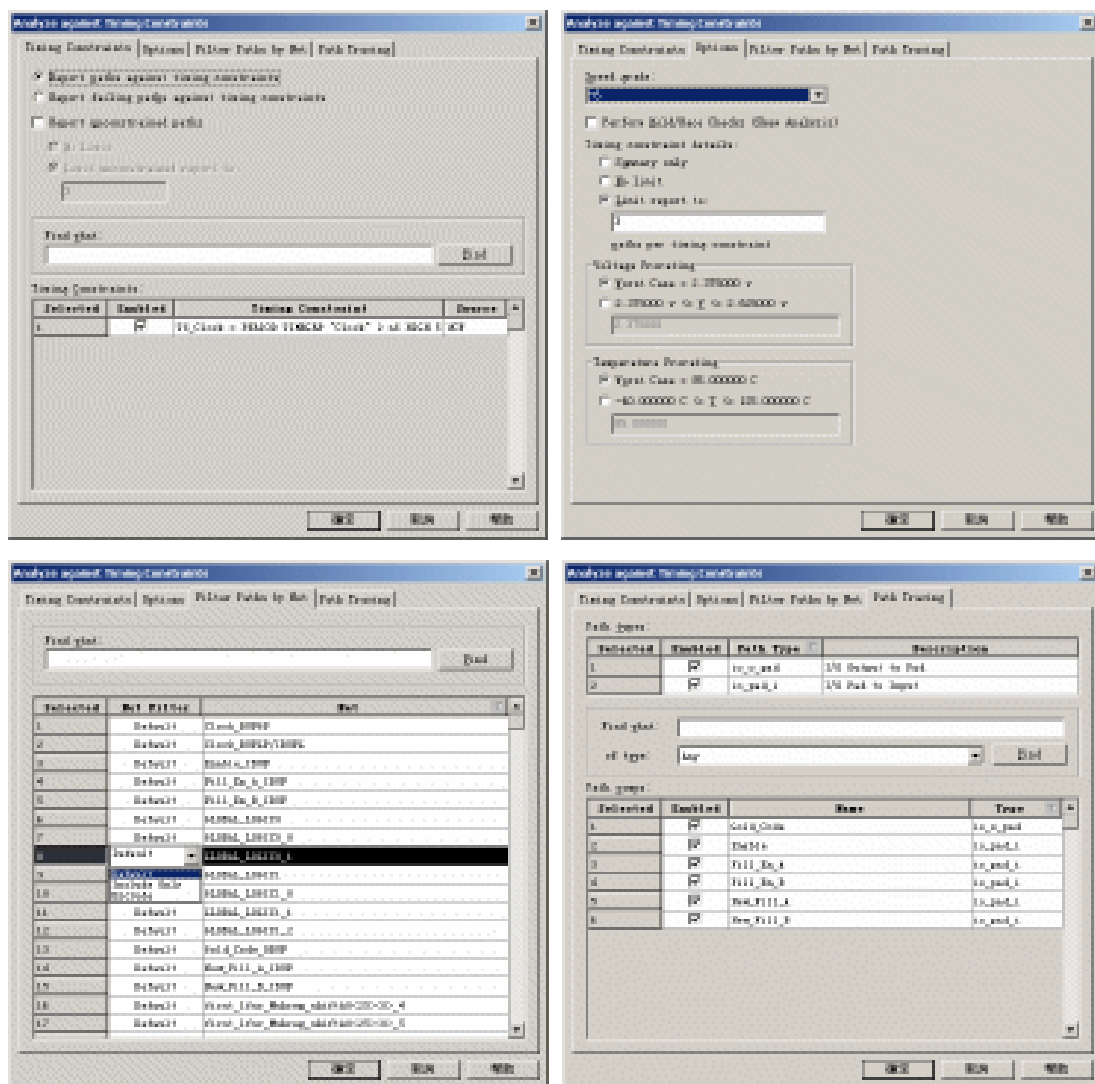


图6-17 时序分析对话框的 4 个选项卡

(1) 【Timing Constraints】选项卡包括如下选项。

- 【Report paths against timing constraints】：如果选择这个单选项，那么时序报告中将包括时序约束覆盖的全部路径。
- 【Report failing paths against timing constraints】：如果选择这个单选项，那么时序报告中只包括时序冲突的路径。
- 【Report unconstrained paths】：如果选择这个复选项，那么时序报告中将包括没有约束的路径。这个选项下还包括下面两个单选项。
 【No Limit】：如果选中这个单选项，那么在时序报告中将不限制路径的数量。
 【Limit unconstrained report to】：如果选中这个单选项，那么需要在相应栏目

中添加限制的路径数量。

- 【Find what】：使用这个功能可以在【Timing Constraints】栏查找时序约束。
 - 【Timing Constraints】：通过这个选项可以设置相关的时序约束是否有效。
- (2) 【Options】选项卡包括如下选项。
- 【Speed grade】：使用这个选项帮助用户确定设计需要的最快速度等级，这个选项只用来做设计评估，并不改变设计的频率。
 - 【Perform Hold/Race Checks (Skew Analysis)】：如果选中这个复选项，那么将执行严格的时钟抖动分析。
 - 【Timing constraints details】：这个选项包括 3 个单选项。
 - 【Summary only】：如果选择这个单选项，那么在时序报告中将仅仅显示路径的起始点和终点。
 - 【No limit】：如果选择这个单选项，那么在时序报告中每一个时序约束的路径数量将不受限制。
 - 【Limit report to】：如果选中这个单选项，那么需要在相应栏目中添加数据，限制每一个时序约束的路径数量。
 - 【Voltage Prorating】：在这个选项中设置器件的工作电压。
 - 【Temperature Prorating】：在这个选项中设置器件的工作温度。
- (3) 【Filter Paths by Net Tab】选项卡主要用来筛选需要进行时序分析的网线，选中某个网线后，单击鼠标右键可以弹出如下的筛选命令：
- 【Include Only】：仅对与选中的网线有关的路径进行分析。
 - 【Exclude】：排除与选中的网线有关的路径。
 - 【Default】：对全部路径进行分析。
- (4) 【Path Tracing Tab】选项卡仅在对 FPGA 进行时序分析时出现。设置完成后，单击  按钮，产生基于时序约束的时序分析报告。

2. 基于自动约束的时序分析

当一个设计中没有时序约束或者没有全面的时序约束时，一般使用基于自动约束的时序分析方法。使用这种分析方法，工具将自动对每一个时钟产生一个周期约束和一个全局偏移约束，周期约束将显示每一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以及最大输出时间。可以通过两种方式产生基于自动约束的时序分析报告：第一种方式是使用工具栏中的  按钮；第二种方式是从时序分析器的菜单栏中选择【Analyze】/【Against Auto Generated Design Constraints】命令，将弹出与图 6-17 类似的对话框，对话框设置完成后，单击  按钮，产生基于自动约束的时序分析报告。

3. 基于用户定义路径的时序分析

基于用户定义路径的时序分析包括两类不同的分析方式：

一类为【Analyze against User Defined Paths by Defining Path Endpoints】，这种分析对用户指定的所有路径都有详细的时序分析。为了启动这种分析方法，可以从工具栏中单击  按钮，弹出的对话框如图 6-18 所示。也可以从菜单栏中选择【Analyze】/【User Specified Paths】/【by Defining Endpoints】命令，打开如图 6-18 所示的对话框。在这个对话框中有 4

个不同的选项卡，除了【Path Endpoints】选项卡之外，其他选项卡的内容与图 6-17 中的各选项卡内容相同。

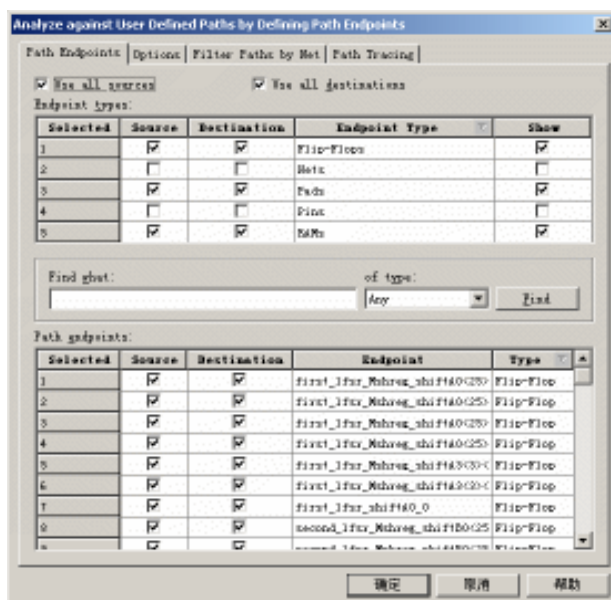


图6-18 【Path Endpoints】选项卡

使用【Path Endpoints】选项卡定义从“源”到“目的”的路径约束，其中包括如下选项。

- 【Use all sources】：选择这个复选项，将使用全部的源端点。
- 【Use all destinations】：选择这个复选项，将使用全部的目的端点。
- 【Endpoint types】：在这个栏目中对端点类型进行了分类。
- 【Path endpoints】：在这个栏目中选择指定路径的“源”和“目的”。

一类为【Analyze against User Defined Paths by Defining Clock and IO Timing】，这种分析适用于整体没有时序约束，但是对时钟和 IO 时序有要求的设计。为了启动这种分析方法，可以从工具栏中单击  按钮，弹出的对话框如图 6-19 所示。也可以从菜单栏中选择【Analyze】/【User Specified Paths】/【by Defining Clock and IO Timing】命令，打开图 6-19 所示的对话框。在这个对话框中有 4 个选项卡，除了【Clock/IO】选项卡之外，其他选项卡的内容与图 6-17 所示的各选项卡内容相同。

使用【Clock/IO】选项卡可以为时序分析过程定义一个时序约束，其中主要包括时钟约束、IO 约束、IO 组约束以及 Pad to Pad 约束等 4 个部分，涉及到如下几个概念：

- 【Period】时钟约束，使用这个栏目指定一个时钟的周期，在使用相同时钟相同沿工作时，相当于【Clock to Setup】路径延时。
- 【Pad to Setup】路径约束，使用这个栏目指定 Pad to Setup 路径约束。
- 【Clock to Pad】路径约束，使用这个栏目指定 Clock to Pad 路径约束。
- 【Pad to Pad】路径约束，使用这个栏目指定 Pad to Pad 路径约束。



图6-19 【Clock/IO】选项卡

6.2 布局规划器——Floorplanner

布局规划器是一个用来查看和编辑物理位置约束的交互式图形接口程序，在这个程序中可以通过物理位置约束映射成目标器件资源的布局约束。对于选定的器件而言，可以通过人工或者自动的方式在布局规划器中进行布局。同时布局器的一个新的特性是可以与时序分析器进行时序交叉探查，这个特性允许在设计中通过时序分析软件启动布局规划器软件，从而对关键路径等信息有更详细的了解，有助于设计的优化。

6.2.1 布局规划器的用户界面

在 ISE 集成开发环境中，如果从模块视图（Module View）中选中一个设计模块，那么在相应的当前资源操作（Processes for Current Source）窗口中的实现阶段就可以直接启动布局规划器，需要的输入文件由 ISE 集成环境自动产生，如图 6-20 所示。

使用鼠标左键双击【Floorplan Design】选项，打开布局规划器的用户界面，如图 6-21 所示。这个窗口分为几个不同的区域，左边上半部分显示了设计的层次，前面的符号代表了具体实现时使用的底层基本单元，如图中【second_lfsr_Mshreg_shiftB0<20><0>_0】具体实现时将使用触发器。在进行布局时可以直接将这些基本元件拖动到布局规划窗口中进行布局。在布局规划窗口中有很多可以利用的资源，如图 6-22 所示，其中有查找表、触发器、快速进位链等资源，在进行布局时需要逻辑与资源进行合理利用。

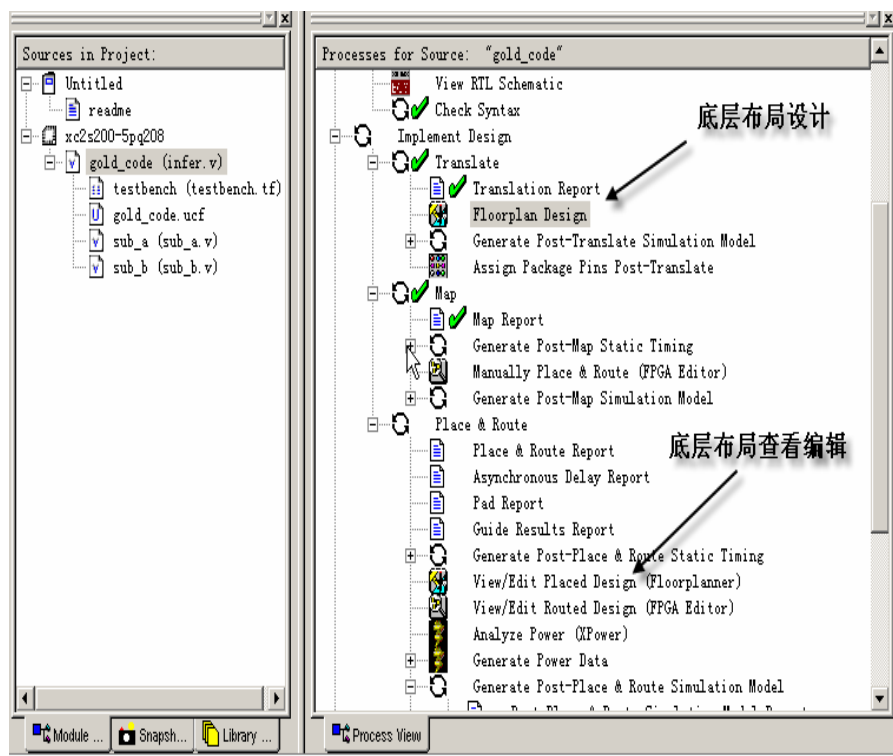


图6-20 启动布局规划器

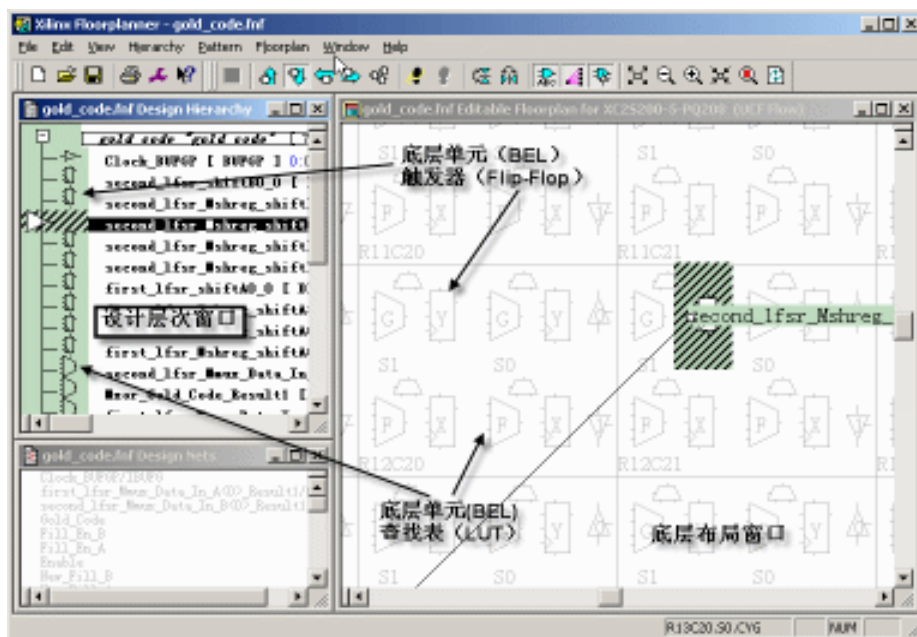


图6-21 布局规划器用户界面

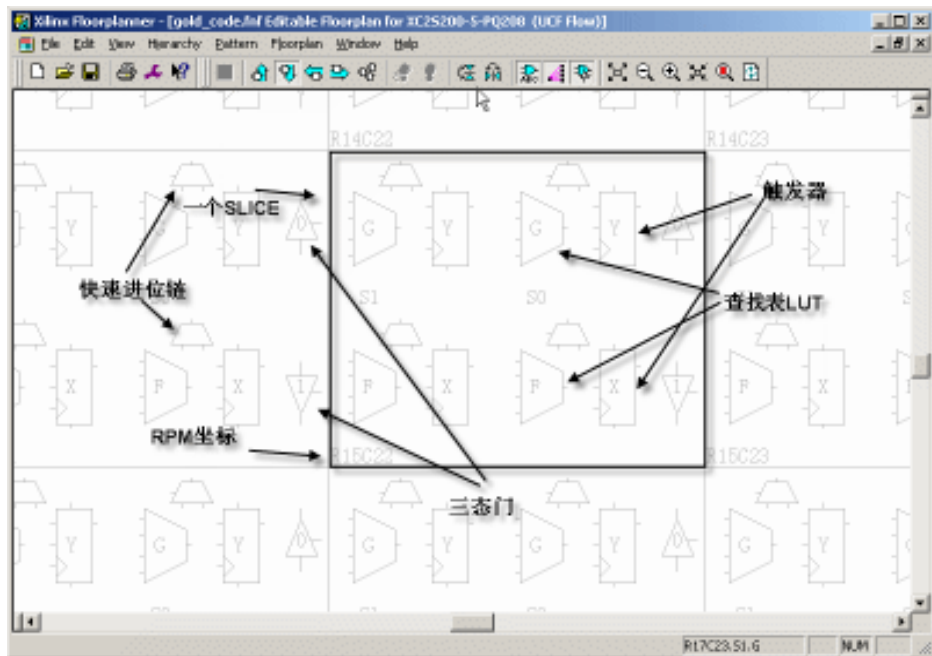


图6-22 布局规划器中的布局窗口

6.2.2 布局规划器的特点及作用

下面简要介绍布局规划器的特点及作用。

1. 布局规划器的特点

布局规划器提供了一个非常友好的图形接口，它的特点如下：

- 可以提供高级设计层次与低级设计层次之间的连接，如可以确定 IOB、功能发生器、三态缓存、触发器和 RAM/ROM 等底层基本单元的位置。
- 自动对逻辑进行分配。
- 允许加载 NGD、NCD、FNF 和 UCF 等文件。
- 通过对 UCF 或者 MFP 文件的修改对设计进行约束。
- 允许对 RPM 进行绑定或者解除绑定。
- 支持时序分析器到布局规划器的交叉探查。

2. 布局规划器的作用

使用布局规划器对设计进行手动布局是提高自动布局布线性能的一种可选方法。布局规划器在结构化设计和数据路径设计上的作用非常明显，可以通过布局规划器确定在何处布置何种逻辑才能达到最优的结果，同时可以通过布局规划器将关键的数据路径布置在恰当的位置上。布局规划器可以根据用户的需要对逻辑单元进行布局，并且可以通过 RPM 绑定的方式将相关单元的物理相对位置固定。

可以使用以前流程产生的 NGD 文件进行布局规划和手动指定 IOB 位置，布局规划器通过新建一个物理位置约束来修改 UCF 文件，物理位置约束添加在原来的 UCF 文件中，或者

布局规划器通过 MFP 文件来存储对设计的物理约束。

根据布局规划器所完成的功能可以将布局规划分为以下几种：

(1) 基于面积的布局规划

在这种布局中，设计者可以按照功能的不同将设计分为几个功能块，每一个功能块都通过布局规划被限制在有效的功能范围内。基于面积的布局规划的主要目标是将相互联系的逻辑布置在同一个功能框架里，进而减少相互之间的连线延时。

(2) 详细的布局规划

在这种布局方式下，可以对单独的逻辑 SLICE 等进行布局，本质上来说，就是完全对设计进行手动布局，这个操作的目的是对细节部分进行优化得到高效的时序，因为这种方式的效率很低，所以只适合于小的设计或者设计中很小一部分的逻辑。

(3) 迭代布局规划

这里的迭代是指布局规划和布局布线之间的反复操作。为了达到设计的目标，在必要的时候设计者需要对逻辑布局反复修改，然后在布局布线后观察是否满足设计要求。

因为使用布局规划器进行设计时需要根据设计的要求手动对布局进行调整，所以设计者首先必须了解所使用的器件结构，知道怎样做才可以提高设计的性能。布局规划本身是一个反复的过程，需要花大量的时间才可以得到一个比自动布局好的效果，同时布局规划器需要和映射、布局布线程序交互操作。基于布局规划器的这些特性，为了更好地在设计中利用布局规划器这个工具，需要注意以下几点。

- 详细了解目标器件的体系结构。
- 详细理解设计实现的过程。
- 必须对将要进行布局规划器的设计本身很清楚。
- 需要有反复多次进行布局规划的思想准备。
- 设计目标的指定者给定的设计性能和面积目标是可实现的，这一点很重要，所有的努力都是基于一个可实现的目标。

6.2.3 布局规划设计流程

在设计综合的多个阶段都可以进行布局规划设计，可以在映射以前、在布局布线以后进行布局规划设计。

1. 在映射之前进行布局规划

在映射之前进行布局规划时，在设计流程中允许设计者在设计输入综合成网表以后、映射之前进行布局规划，这种设计流程如图 6-23 所示。

这种布局规划的设计步骤如下：

- (1) 输入设计以后，进行综合得到网表文件并翻译（Translate）成 NGD 文件。NGD 文件包含了设计开始时输入的物理位置约束 UCF 文件或者 NCF 文件，同时 NGD 文件还包含了设计中实例化的 IP 宏。
- (2) 在布局规划器中使用 UCF 流程选项打开 NGD 文件。布局规划器将读入 NGD 文件的全部数据以及引用的 IP 宏，并使用这些设计数据创建初始化布局规划

器位置约束。

- (3) 使用布局规划器工具定义一个布局约束区间，手动将选择的逻辑和器件资源放置在约束区间中。
- (4) 将布局规划的结果存储成 UCF 文件。
- (5) 执行其他的设计流程。

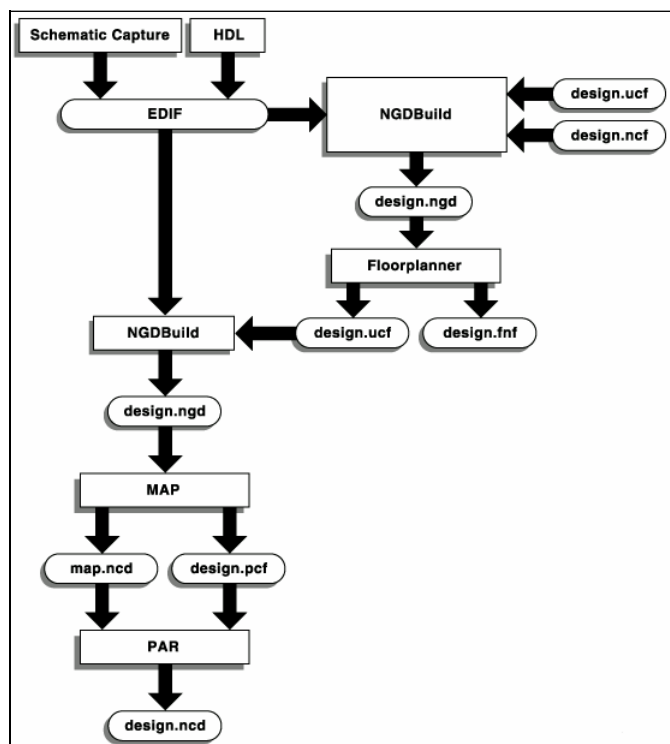


图6-23 在映射之前的布局规划设计流程

2. 在布局布线后进行布局规划

可以在布局布线以后进行布局规划，设计者可以将自动布局布线和物理约束下的布局布线结果进行对比，这样可以比较容易明确设计的改进情况，它的设计流程如图 6-24 所示。

这种布局规划的操作步骤如下：

- (1) 用 HDL 或者图形接口输入设计。
- (2) 运行 NGDDBuild 程序，创建 NGD 文件。
- (3) 运行映射和布局布线程序对设计在目标器件中进行布局布线。
- (4) 为了查看和提供自动实现的结果，可以在布局规划器中打开 NGD 和 NCD 文件，并使用布局规划器对关键时序路径进行约束或者通过布局规划器调整自动布局的结果。
- (5) 如果想对自动布局的结果进行查看，可以使用【View】/【Placement】命令。
- (6) 如果想改变自动布局的结果，可以使用【Floorplan】/【Replace all with Placement】命令。
- (7) 将新的设计存储成 FNF 文件和 UCF 文件。

- (8) 在最新的 UCF 文件的指导下运行映射和布局布线程序，从而得到布局规划器的设计结果。

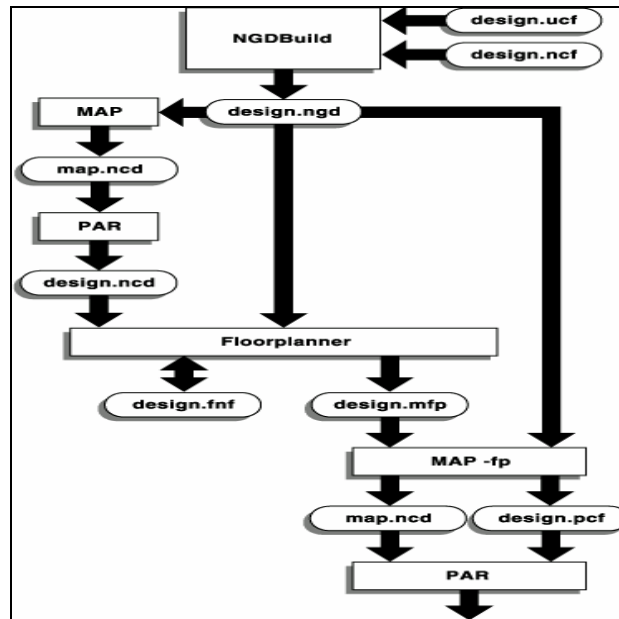


图6-24 布局规划流程

3. 迭代流程

这个流程允许布局规划器、映射、布局布线等设计过程再三迭代，直达到达要求的性能目标，设计流程如图 6-25 所示。

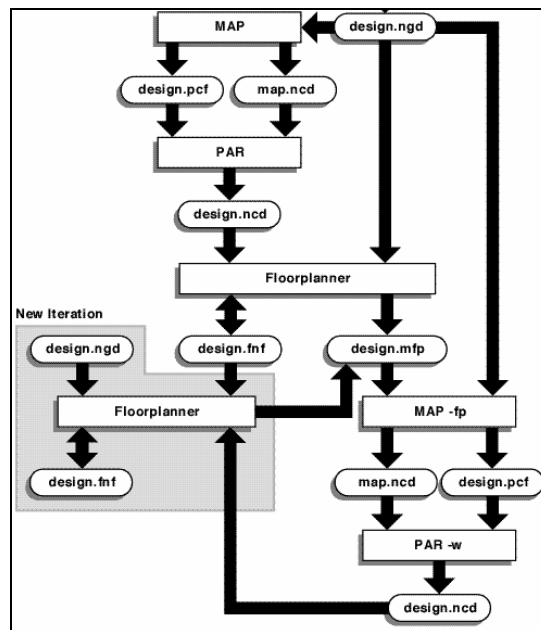


图6-25 迭代流程

这种布局规划的操作步骤如下：

- (1) 用 HDL 或者图形接口输入设计。
- (2) 运行 NGDBuild 创建 NGD 文件。
- (3) 使用布局规划器对设计进行约束划分，并将约束保存成一个 UCF 文件。
- (4) 运行映射、布局布线程序将设计实现到目标器件中。
- (5) 基于自动布局布线工具实现的结果，修改当前的布局规划逻辑。
- (6) 在最新的布局规划约束的基础上运行映射、布局布线等程序实现设计。

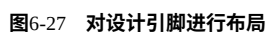
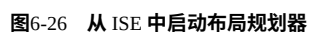
从上述的各种布局规划的设计流程可以看到，设计流程只是输入输出文件有所差别，而具体操作起来差异并不明显。在工作实践中，一般都需要多种流程交叉使用，从而缩短开发周期。在 ISE 的整个设计流程中，约束编辑器提供了进行时序约束的接口，而布局规划器提供了进行位置约束的用户接口。在 ISE 集成环境开发的项目中，设计者可以使用 UCF 作为实现指导文件，也可以使用布局规划器产生的 MFP 文件作为实现指导文件，这两种工具的作用和分工都很明确。

6.2.4 设计示例

下面使用 Xilinx 的应用文档 xapp233 中设计 622MHz 的 LVDS 为例来说明时序分析器以及布局规划器的使用方法。应用文档中的设计是使用原理图输入的方式完成的，在本书中将这个设计改写成 HDL 文件，存储在随书所附光盘的“Example-6-1\Source”目录下。关于 LVDS 的概念性介绍将参阅 Xilinx 的应用文档，编号为 230、232 和 233 等。本节将设计一个两路数据信道、一路时钟信道的 LVDS 发送器，其中使用了双数据速率时钟触发（Double Data Rate Clocking）来提高 LVDS 的工作频率，所以本设计的目标为时钟频率 311MHz。

设计 LVDS 发送器，参考示例详见本书附带光盘的“Example-6-1”目录

1. 将随书所附光盘“Example-6-1”目录下的 Verilog 文件拷贝到本地硬盘的“E:\Book\Chapter6\Example-6-1”目录下。
2. 启动 ISE 集成开发环境，建立一个新的工程项目“6-1pro”，选用器件【xcv1000e-6hq240】，选用综合器【XST Verilog】，将“source”目录下的全部源文件添加到工程项目中。
3. 从工程浏览器的【Module View】视图中选择顶层模块【tx2bit】，从【Process View】视图中选择映射阶段的【FloorPlanner Design】选项启动布局设计，如图 6-26 所示。
4. 因为在每一个 FPGA 上的 LVDS 的端口都是成对设计的，不可以随意布局，所以在本例中首先说明通过布局规划器对 LVDS 的引脚进行约束的方法。从 Xilinx 的数据手册上查阅可以使用的 LVDS 对，在本例中选择使用 (p110,p111)、(p113,p114) 和 (p117,p118) 三对引脚作为 LVDS 信号的输出，如图 6-27 所示，将设计输出信号按照 (da_x->p110,da->p111)、(ck_x->p113,ck->p114)、(db_x->p117,db->p118) 进行布局。



5. 保存设计文件，默认使用顶层模块名作为布局规划器设计的文件名，在本设计中将产生 tx2bit.fnf 文件和 tx2bit.ucf 文件，退出布局规划器。
6. 添加 tx2bit.ucf 作为当前工程的约束文件，选中【Module】视图中的 tx2bit.ucf 文件，在【Process View】窗口中选择【User Constraints】/【Edit Constraints(Text)】，单击鼠标打开约束文件，在其中添加“NET "clk4x" LOC = "P89"”和“NET "clk" LOC = "P213";”全局时钟位置约束，添加全局时钟时序约束如下：


```
NET "clk4x" TNM_NET = "clk4x";
TIMESPEC "TS_clk4x" = PERIOD "clk4x" 3.2 ns HIGH 50 %;
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 12.8 ns HIGH 50 %;
```
7. 保存文件，关闭编辑窗口。
8. 重新对设计在约束条件下进行布局布线。在【Process View】窗口中，选择【Implement Design】/【Place & Route】/【View/Edit Placed Design】，打开窗口可以看到输出的 LVDS 的信号已经按照布局规划器中指导的位置放置了，但是其中的逻辑部分因为没有指定物理位置约束，所以是自动布局布线器根据其自身的推论进行布局的。
9. 对这个程序最终实现的情况做时序分析，检查是否满足最初的设计。在这个程序中，LVDS 的数据信号需要工作在 622MHz，在【Process View】窗口中，选择【Implement Design】/【Place & Route】/【Generate Post-Place & Route Static Timing】/【Post-Place & Rout Static Timing Report】选项，启动时序分析器对设计进行时序检查，可以看到设计【Timing Constraints】为红色，说明当前的时序约束并不能满足要求，如图 6-28 所示。

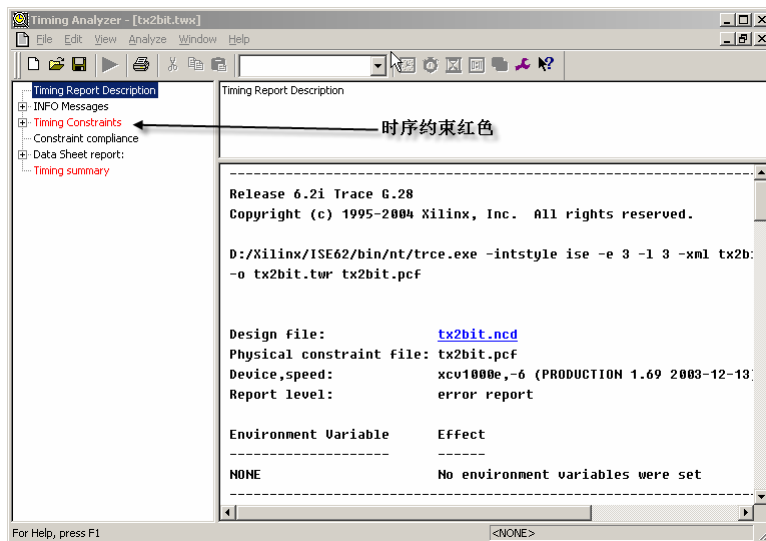


图6-28 静态时序报告窗口

10. 点击左侧红色部分右侧文件跳转，如图 6-29 所示，可以通过时序改进向导帮

助提高设计时序。启动时序改进向导，如图 6-30 所示。

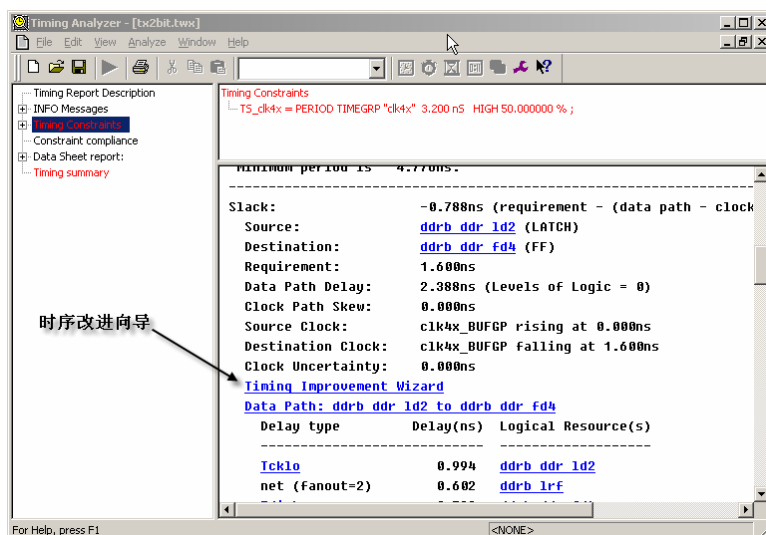


图6-29 时序分析窗口

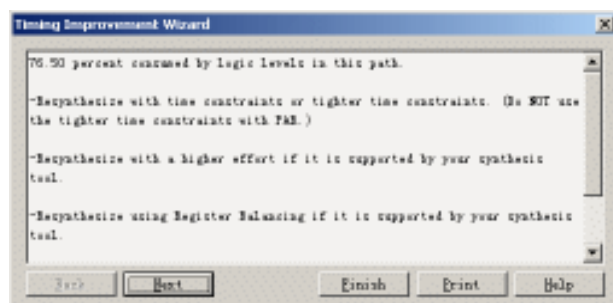


图6-30 时序改进向导

11. 通过分析可以看到还需要添加其他的时序约束才能达到设计目标。使用“Example-6-1\Source\tx2bit.ucf”文件，其中约束命令的说明请参见约束章节，因为在这个约束文件中约束的编写使用了设计层次，所以在综合的属性中必须选择保留设计的层次（保留设计层次是一个高级的综合约束，在 ISE 集成环境中默认条件下不能对这个属性进行设置，并且默认条件下 XST 综合器在进行综合时并不保留层次。选择【Edit】/【Perference 命令，打开属性设置对话框，选择“Processes”标签页面，在“Property Display”中选择【Advanced】确定保存设置。然后打开综合器属性设置，就可以在综合时选择保留设计层次】。
12. 综合实现后使用时序分析器再次分许设计时序，可以看到，使用了更为详细的约束后整体的性能再次得到提高，但还没有达到设计要求，使用 FPGA Editor 打开布局布线后的结果文件，但是布局布线的结果却比较凌乱，如图 6-31 所示。这样的布局结果也影响了设计的性能，可以通过物理位置约束来改善这种状况。

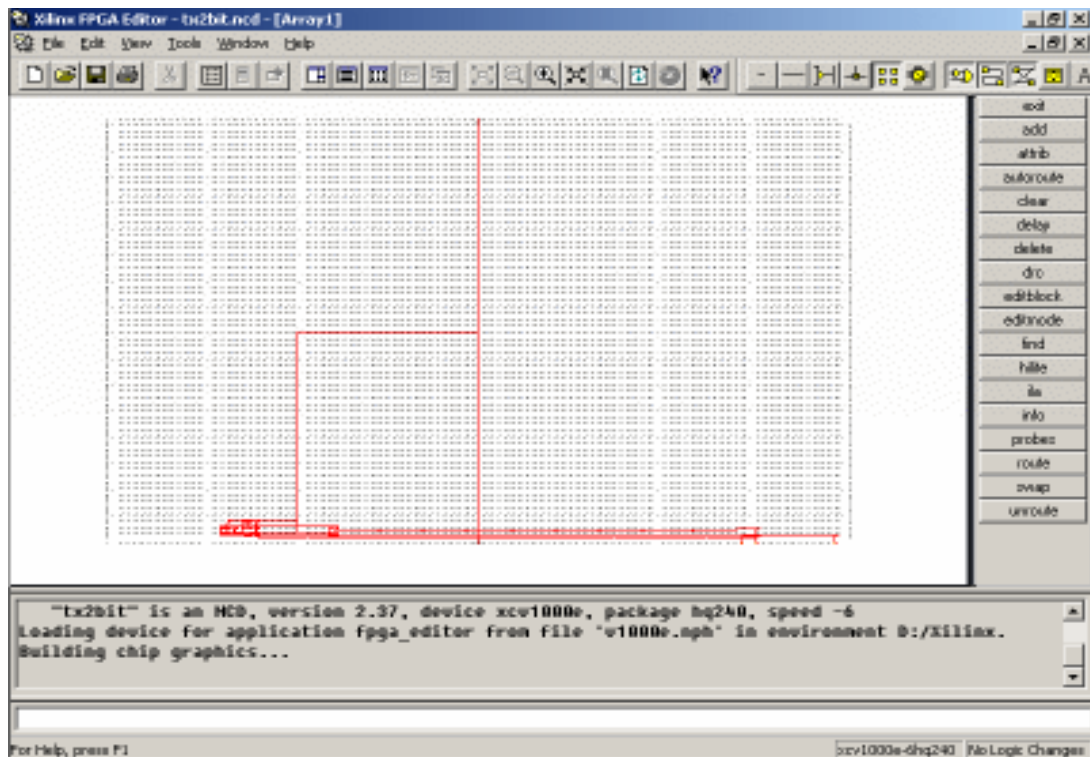


图6-31 布局布线后的结果

13. 下面使用物理位置约束来改善设计的性能。在【Process View】中选择【Implement Design】/【Map】/【Floorplan Design】选项，启动布局规划器。在布局规划器设计层次中，选中全部的设计资源，然后在主窗口中选择【FloorPlan】/【Assign Area Constraint】命令，在布局规划窗口中出现十字光标，选择需要布局的区域，创建区域约束。可以通过布局规划器实现区域约束，如图 6-32 所示。区域约束是一个基本的布局、布线和模块化设计的约束，同时它是一个高级的布线约束。在设计中使用区域约束可以将一个芯片上的逻辑设计分为一个或者多个不同物理区域。在一个区域约束中可以定义区域的边界，也可以定义这个区域的压缩因子等信息。
 14. 保存设计文件，默认使用顶层模块名作为布局规划器设计的文件名，保存 tx2bit.fnf 和 tx2bit.ucf 文件，退出布局规划器。
 15. 注意如果没有在布局规划器中对其他端口例如时钟进行位置约束，那么在布局规划器中保存了设置以后，需要对没有进行布局规划的端口重新进行位置约束。
 16. 使用最新的 tx2bit.ucf 作为当前工程的约束文件，重新进行布局布线，然后进行时序分析，查看是否设计达到设计要求，如果还没有达到要求，可以重新调整区域约束等条件进行尝试。
- 本单元描述的设计流程以及设计方法不是必要的也不是最好的，通过这个例子希望告诉

读者其他的一些设计思路和方法，细节部分需要大家多多探索。

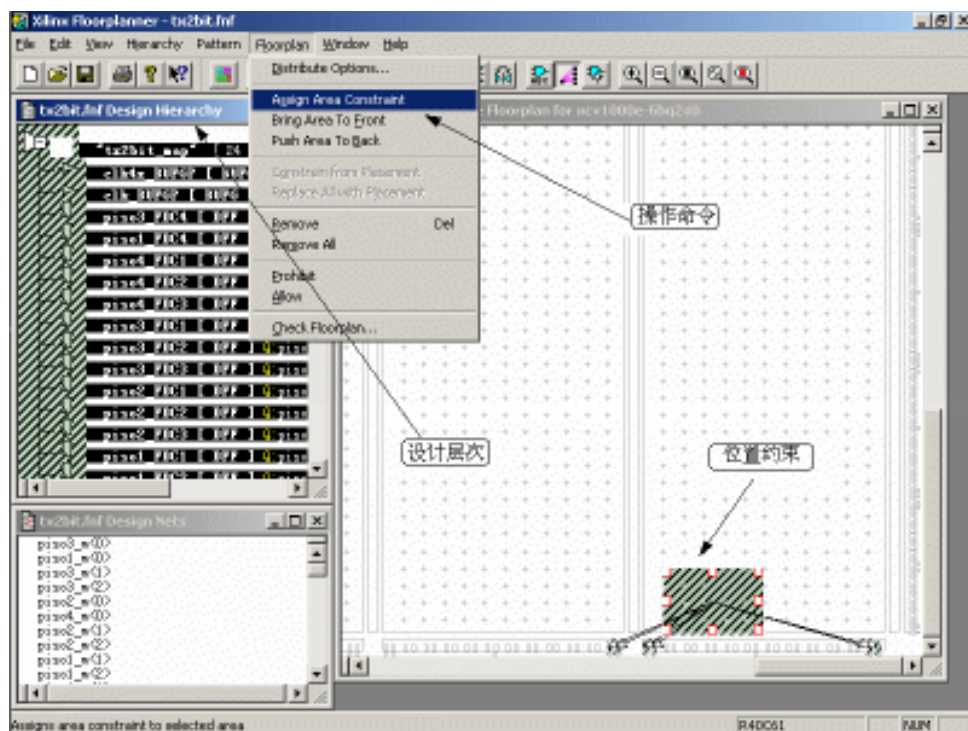


图6-32 物理位置约束示意图

以上通过添加详细的时序约束以及区域约束，改善了整体设计的性能，使得设计的时钟频率可以达到 311MHz，满足了设计的要求。

6.3 FPGA 底层编辑器——FPGA Editor

FPGA Editor 是 Xilinx 公司提供的 FPGA 设计套件中的一个手工布局布线设计工具。通过这个工具可以编辑或者查看可配置逻辑功能块（CLB）、输入输出功能块（IOB）等一些 FPGA 内部的基本单元。对于 Xilinx 公司的 FPGA 设计来说，FPGA Editor 是最底层的一种设计工具。有效的使用这个工具，可以最大程度地提高设计的密度和设计的性能。

6.3.1 FPGA 底层编辑器的用户接口

在 ISE 集成开发环境中，如果从【Module View】视图中选中一个设计模块，那么在相应的【Processes for Current Source】窗口中的实现阶段就可以直接启动 FPGA 底层编辑器，这个软件需要的输入文件由 ISE 集成环境自动产生并提供，如图 6-33 所示。可以使用鼠标在两个不同的选项【Manually Place & Route (FPGA Editor)】和【View/Edit Routed Design (FPGA Editor)】之间进行选择。例如使用鼠标双击【View/Edit Routed Design (FPGA Editor)】选项就可以启动布局布线后设计的查看程序，如图 6-34 所示。FPGA 底层编辑器的用户接口分为几个不同的区域，有布局窗口、交互输出窗口、命令输入行、底层元件列表

窗口以及快捷按钮窗口等。

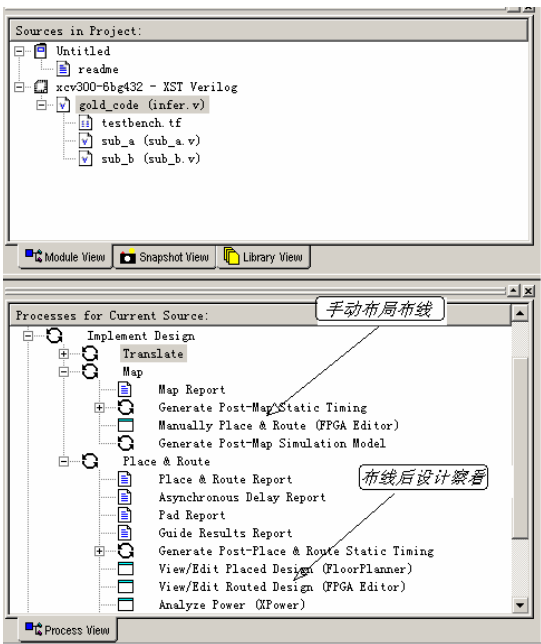


图6-33 启动 FPGA 底层编辑器

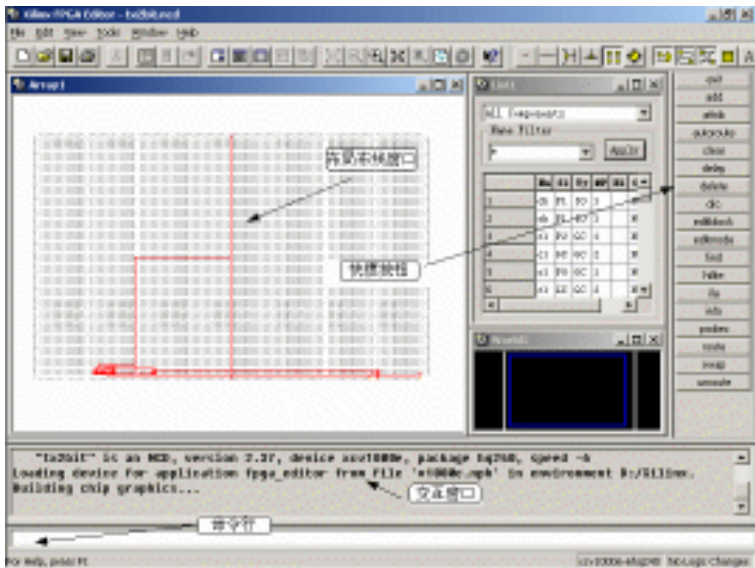


图6-34 FPGA 底层编辑器的用户接口

6.3.2 FPGA 底层编辑器的作用

在设计中使用手动布局布线工具可以达到以下几个目的：

- (1) 在运行自动布局布线器之前对设计的关键组件和路径进行布局布线。

- (2) 如果自动布局布线器没有完成设计，可以通过手动的方法完成。
- (3) 可以在设计中添加探针，用来检测在目标器件中信号的状态，探针可以将一个内部的网线连接到一个 IOB 上，通过这种方式可以调试器件内部的任意信号。
- (4) 可以生成配置文件并且可以直接配置流文件到目标器件中。
- (5) 可以查看插入了 ILA 核之后对设计网表的影响，可以很方便地改变需要测试的信号与 ILA 核的连接。
- (6) 可以手工创建一个完整的设计（工作效率因人而异）。

总的来说，手动布局布线在设计中主要有以下作用：通过手动布线、关键路径预处理等方法提高设计的布通率；通过手动设计的方式提高设计的性能，减小设计的面积；通过添加探针或者对 ILA 的修改来方便板上调试，加快研发进度。

对于大型的设计或者时序要求比较严格的设计来说，在使用自动布局布线器时，可能会存在这样一种情况：虽然布线器经过了最大的努力，但最后还是无法完成设计整个布线过程，而设计的代码在力所能及的范围内下已经是最优化了。这时，可以使用手工布线来完成自动布局布线器没有完成的布线工作，提高设计的布通率。同样还可以使用 FPGA 底层编辑器对设计的关键路径做一些预处理，从而提高设计的布通率。例如，在设计的自动布局布线运行之前，对整个设计的关键路径进行预处理，然后使用 FPGA 底层编辑器打开映射以后的 NCD 文件，对关键路径所涉及到的基本单元预先进行布局，这种布局的形式要比使用 Floorplanner 进行布局精确很多，它可以对每一个 SLICE 中的逻辑进行布局，如果有必要也可以在布局以后对关键路径进行预布线，并且将这个布局布线的结果通过直接布线约束输出到相同工程下的 NCF 文件中。在自动布局布线时使用这个预布局结果和直接布线约束对关键路径的布局布线做指导，可以使关键路径的布局布线完全按照设计者的意图来实现，从而提高整个设计的布通率。

如果对 FPGA 的内部架构有了深刻的认识，那么可以通过 FPGA 底层编辑器来完成整个设计，并且可以通过这种方式得到高质量的设计结果。当然，这种设计方法的工作效率比较低，现在的设计门数越来越大，少则几十万门，多则几百万门，这样的设计规模如果想全部使用手动方式来完成，显然不太实际。这时可以通过 FPGA Editor 对关键部分进行手动设计，形成物理硬宏，而对于整个设计使用 HDL 等相对高级的设计方式，这样既可以保证关键部分的时序又可以保证设计的效率。一般情况下，在使用 FPGA 底层编辑器的时候，需要根据不同的设计要求采用不同的设计方式组合，以达到特定的设计要求。

在多数情况下，为了增加程序的可调试性，我们会在设计中使用 ILA（在线逻辑分析仪）对内部的信号或者内部总线进行观察与分析。当使用 ChipScope Pro Core Inserter（一种产生 ILA 网表的方法）在原先的设计中插入 ILA 网表时，一般会将采样时钟、触发条件、需要观测的数据等与内部的某些网线连接，但是如果设计比较大、需要观察的网线比较多，而又没有太多的资源用来插入多个 ILA 时，可以首先对部分信号进行观测，然后调整采样时钟、触发条件和观测数据与内部网线的连接关系，再观察另外的网线。这时，如果每一次的调整都重新插入 ILA，重新综合、布局布线，那么工作效率会很低，同时由于每一次的布局布线的结果会有少许不同，可能会影响分析的可信度。所以一个比较好的解决方法是：首先对将一些信号与 ILA 连接进行观察，为了观察另外的一些信号线，可以使用 FPGA 底层

编辑器打开布局布线后的 NCD 文件，改变内部网线和 ILA 观测数据线的连接关系，然后直接在 FPGA Editor 中启动配置程序对 FPGA 进行配置，重新观测信号线。这种改变 ILA 连接的方法只能通过 FPGA Editor 来实现。

在上面已经提及了一种在线的调试方式 ILA，在 FPGA Editor 中还可以实现另外一种调试方法——探针。这两种调试方式都是在设计实现到目标板上之后进行的，所以是最接近 FPGA 实际情况的调试和分析方式。

探针功能的实施很简单，在打开的 NCD 网表中找到需要观测的网线名称，然后使用【Tools】菜单中的探针选项，就可以直接将这个内部网线连接到外部的 PAD 上，并且可以通过探针的编辑功能改变输出的 PAD 位置，同时可以得到网线到 PAD 的连接延时。其中 PAD 位置可以修改到任何可以利用的空闲 PAD 上，这样有利于在实际中进行测试。而且软件可以给出添加的网线的确定时延，也为实际测试带来了方便。这种确定的延时信息在 ILA 测试中无法给出，所以这两种测试方法各有长处。

6.3.3 FPGA 底层编辑器输入输出文件

FPGA 底层编辑器在设计流程中的位置如图 6-35 所示，从图中可以看出 FPGA Editor 可以接收 3 种文件作为输入，同时也可以将这些文件作为输出。

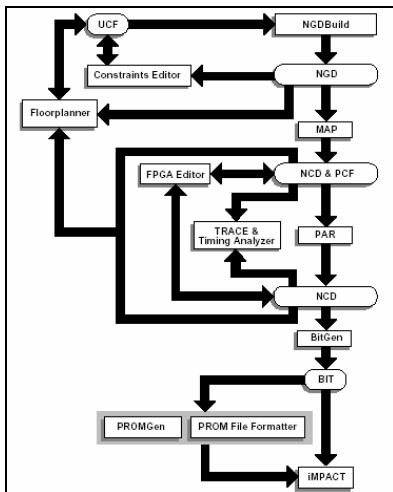


图6-35 FPGA 底层编辑器在设计流程中的位置

- **NCD 文件**

NCD 文件文件是在映射之后产生的，在这个文件中使用了 CLB 和 IOB 等描述了一个设计，使用 FPGA Editor 可以用来编辑 NCD 文件也可以将最后的结果存储成 NCD 文件。

- **PCF 文件**

PCF 文件是映射程序生成的包括物理约束的文本文件，它和 UCF 中的约束基本相同，可以在 FPGA Editor 中编辑 PCF 文件。

- **NMC 文件**

NMC 文件是物理宏库文件，在这个文件中包含了物理宏库的定义，同时这个物理宏可

以在 FPGA 底层编辑器中以及 HDL 编码中实例化。

6.3.4 FPGA 底层编辑器的工作流程

FPGA Editor 的基本设计流程如下：

- (1) 将设计保存成为 EDIF 文件。
- (2) 运行 NGDBuild 程序，创建 NGD 文件。
- (3) 运行映射程序，创建 NCD 文件。
- (4) 使用 FPGA 底层编辑器加载设计，进行必要的改动，存储成 NCD 文件。
- (5) 对修改后的 NCD 文件进行布局布线。

6.3.5 使用 FPGA 底层编辑器的预备知识

为了有效地利用 FPGA Editor，必须对 FPGA 的内部架构非常清楚。Xilinx 有 SpartanII、Virtex、Virtex-E、Virtex-II 和 Virtex-II pro 等系列的 FPGA 产品，各种器件的内部架构会有所不同，并且每一类产品的电气特性、开关特性等区别比较大，所以详细地理解这些器件之间的差别是有效利用 FPGA Editor 的前提条件。这里主要以 Virtex-E 系列为例做简单介绍，其他系列请读者阅读 Xilinx 公司的数据手册。

Xilinx 的 Virtex-E 系列用户可编程门阵列包括两个主要的可配置单元：可配置的逻辑功能块（CLB）和可配置的输入输出功能块（IOB），同时包括专用存储块（BRAM）、用于时钟分布延时补偿和时钟域控制的时钟数字锁相环路（DLL）、与每一个 CLB 相连的三态缓存器等资源。其中 CLB 提供构造用户逻辑的功能单元；IOB 提供封装引脚和内部信号的接口。

1. 可配置的逻辑功能块（CLB）

可配置逻辑功能块是由逻辑单元（LC）积木式组合而成。一个 LC 包括一个四输入函数发生器、进位逻辑和一个存储单元。在每一个 LC 中，函数发生器的输出既可以作为 CLB 的输出也可以作为触发器的输入。每一个 Virtex-E 的 CLB 在两个几乎相同的 SLICE 上包含 4 个逻辑单元。

除了 4 个基本的逻辑单元之外，CLB 还包括使函数发生器提供五或者六输入的函数逻辑。因此，在估计一个给定器件的系统门数时，每一个 CLB 按照 4.5 个逻辑单元计算。

• 查找表

Virtex-E 的函数发生器可以用来实现四输入的查找表（LUT），除了作为函数发生器，每一个 LUT 还可以提供 16×1 位同步 RAM，一个片上的两个 LUT 连接可以生成 16×2 位或者 32×1 位的同步 RAM 或者一个 16×1 位的双端口同步 RAM。同时 LUT 可以作为一个 16 位的移位寄存器，可以捕获高速或者突发模式的数据。

• 存储元件

在 Virtex-E 芯片中，存储元件可以被配置成边沿触发的 D 触发器或者电平敏感的锁存器。D 触发器可以使用函数发生器的输出驱动，也可以使用 SLICE 以外的输入而绕过函数发生器。除了时钟和时钟使能信号以外，每一个 SLICE 上都有同步的置位和复位信号

(SR、BY)。SR 强制存储元件进入配置的初始化状态，而 BY 强迫存储元件进入相反的状态。这些信号也可以配置成异步操作。为了增加配置的灵活性，这些控制信号都可以翻转，并且控制信号在一个 SLICE 的两个寄存器的操作中共享。

- **附加逻辑**

每一个 SLICE 的 F5 多路转换器连接了两个函数发生器的输出，这个连接可以实现五输入的函数发生器或者一个 4:1 的多路转换器，或者可以配置成一个九输入的函数发生器。

同样，在一个 CLB 中，F6 多路转换器连接了 4 个函数发生器的输出和一个 F5 多路转换器的输出，这种连接可以实现六输入的函数或者一个 8:1 多路转换或者高达 19 个输入的选择函数发生器。

每一个 CLB 有 4 个直接输入通路，每一个 SLICE 有两个。它们不占用任何逻辑资源，但是可以提供额外的数据输入线和附加的局部路线。

- **算术逻辑**

在每一个 SLICE 中有专用进位逻辑为高速算术功能提供快速的算术进位能力。Virtex-E CLB 支持两个独立的进位链，每一个 SLICE 一个，进位链的高度是每一个 CLB 进两位。

算术逻辑包括一个 XOR 门，这样允许在一个 LC 中实现一个全加器。此外，一个专用的 AND 门提高了乘法器实现的效率。进位路径可以作为级联函数发生器以实现更宽的逻辑功能。

2. 输入输出块 (IOB)

Virtex 的 IOB 可以支持很多类的 I/O 接口信号标准。在 IOB 中有 3 个存储单元，可以配置为边沿触发的 D 触发器，或者电平敏感的锁存器。每一个 IOB 中的 3 个触发器共享一个时钟信号，但是都独立使用时钟使能信号，除了 CLK 和 CE 控制信号外，这 3 个触发器还共享同一个置位/复位信号 (RS)，对于每一个触发器来说，RS 信号可以被独立配置为同步置位、同步复位、异步预复位、异步清零等。输入输出缓冲器和所有的 IOB 控制信号都有独立的极性控制电路。

所有的焊盘都做了防静电和瞬态过压保护，提供了两种方式的瞬态过压保护，即一种允许对 5V 兼容，而另外一种不允许。对 5V 兼容的过压保护来说，有一个接地的类似齐纳的结构，当输入大于 6.5V 时，这个结构被打开。当不兼容 5V 时，常规的二极管将连接到输出供电电压 V_{CC} 上。每一个焊盘都可以独立地选择使用什么样的过压保护。

每一个焊盘可以选择使用上拉、下拉电阻和可选的弱保持电路，在配置前，所有没有用到的 IO 都强制为高阻态，下拉电阻和弱保持电路是无效的，但是 IO 可以被配置为上拉。

所有的 Virtex-E 支持与 IEEE 1149.1 兼容的边界扫描测试。

- **输入路径**

在 Virtex IOB 中，输入路径的缓冲器将输入信号直接连接到内部逻辑中或者通过一个可选的输入触发器连接到逻辑。在这个触发器的 D 输入端，有一个可选的延时单元，这个延时单元可以用来与 FPGA 内部时钟分布延时相匹配，当应用时，可以确保 PAD 到 PAD 的保持时间为零。

每一个输入缓冲器可以被配置为所支持的任何一种低压信号标准。在配置后，每一个输

入端都有上拉或者下拉电阻可以选用，其电阻值在 $50\Omega \sim 100k\Omega$ 之间。

- **输出路径**

输出路径包括一个三态缓冲器，它驱动焊盘上的输出信号，这个信号可以由内部逻辑直接送到缓冲器或者可以通过 IOB 直接输出到触发器。输出的三态控制也能直接从内部逻辑或者可以从提供同步使能和禁止的触发器来布线。

每一个输出驱动器能被独立编程。每一个输出缓冲器输入 24mA，输出达 48mA。驱动力和瞬态率的控制使得总线的瞬态最小。

在大多数的信号标准中，输出的高压依赖于外部的 V_{CC} 值。其中每一个输出端都可以选择是否采用弱保持电路。当选择这种电路时，它将监视焊盘上的电压并试图进行与输入信号的电压匹配。如果引脚被连接到一个多源信号，当所有的驱动都不允许时，弱保持电路将保持信号的最后一个状态，通过这种方式保持一个有效的逻辑电平从而消除总线的干扰。这种作用也决定了弱保持电路使用的范围，因为弱保持电路使用 IOB 的输入缓冲器来检测输入电平，如果信号标准需要参考电压，这时必须提供一个适当的参考电压，同时这个电压必须遵循 IO 的排列规则。

- **IO 组**

以上所说的很多 IO 标准都需要参考电压，这些电压从外部连接到器件的引脚上，并供给 IOB 使用。所以，哪一个组使用什么信号标准就有限制，因为在同一个组中只能有一个参考电压。所以在 FPGA 中将每一个边的引脚都分为两个组，从而形成了 8 个不同的组。在同一个组中，当使用同一个标准电压时，输入信号的标准可以兼容。如果一个输入信号需要用户提供参考电压，那么 IO 引脚可以被配置作为参考电压的输入。在每一个组中参考电压引脚的内部是互联的，因此，在每一个组中仅需要一个参考电压。用来连接参考电压的引脚的输入缓冲器不能连接高于 5V 的电压。

在给定的封装中，参考电压的引脚数是根据器件的尺寸变化的。在大的器件中，有更多的 IO 可以作为参考电压引脚。

6.3.6 设计示例

前面讲解了 FPGA 底层编辑器设计流程以及设计的预备知识，下面将使用一个简单的 2bit 加法器为例，对 FPGA 底层编辑器的使用方法进行说明，通过这个例子，读者可以学习如何在 FPGA 底层编辑器中手工进行设计。

请看下面的程序：

```
module add2(a,b,cin,c,cout);
  input [1:0] a;
  input [1:0] b;
  input cin;
  output [1:0] c;
  output cout;
  assign {cout,c} = a + b + cin;
endmodule
```

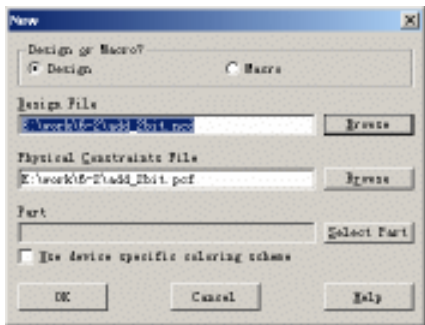
如果将这个程序使用综合工具进行综合并进行实现，然后使用 FPGA 底层编辑器进行查看，将会发现在这个加法实现中并没有使用 CLB 中的快速进位链这个最有效的资源，这样的实现方式没有最大限度地发挥器件的潜力。如果不使用快速进位链，则增大了实现的面积（使用了 3 个 SLICE），并且浪费了布线资源。下面将使用 FPGA 底层编辑器设计一个 2bit 的全加器。

🔑 设计一个 2bit 全加器，参考示例详见本书附带光盘的“Example-6-2”目录

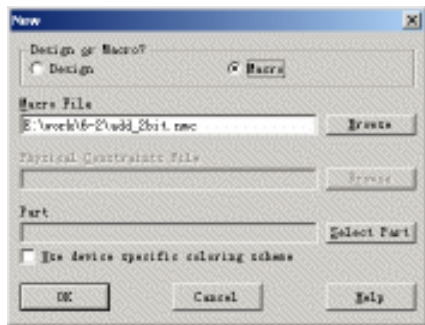
1. 在 Windows 的【开始】菜单中选择【程序】/【Xilinx ISE 6】/【Accessories】/【FPGA Editor】命令，启动 FPGA Editor。
2. 在 FPGA Editor 中选择【File】/【New】命令，弹出【New】对话框，如图 6-36 (a) 所示。

在这个对话框中包含如下选项：

- 【Design or Macro】：这个选项有两个单选项，用来确定设计的目标以及流程。如果选择【Design】单选项，那么下面的【Design File】、【Physical Constraints File】和【Part】三个选项都可以进行配置，如果选择了【Macro】单选项，那么下面的【Physical Constraints File】选项将不可用。
 - 【Design File】：如果第一个选项选择【Design】，那么在这个栏目中将是设计文件的名称，使用 ncd 作为文件扩展名，如果第一个选项选择【Macro】，那么在这个栏目中将是设计硬宏的文件名称，使用 nmc 作为文件扩展名。
 - 【Physical Constraints File】：如果第一个选项使用【Design】，那么在这个栏目中将是物理约束文件的名称，使用 pcf 作为文件扩展名。如果第一个选项中使用【Macro】选项，那么这个栏目将变为不可用，在设计的硬宏中并不需要物理位置约束文件。
 - 【Part】：无论第一选项选择【Design】或【Macro】，在这个选项中选择使用的器件系列以及器件封装等信息。
3. 本例选中【Macro】选项，各种项目选择如图 6-36 (b) 所示，设计文件的名称为 add_2bit.nmc。



(a)



(b)

图6-36 FPGA 底层编辑器的文件类型

4. 在 FPGA 底层编辑器中，SLICE 以及 BRAM 的坐标系使用左上角作为坐标原

点，坐标值向右向下增长。使用鼠标选中第八行第二列 SLICE，在列表下方将出现所选中的 SLICE 在器件中的物理位置坐标。如图 6-37 中坐标为【R8C2】，其中 S0 和 S1 表示每一个 CLB 中 SLICE 的位置。

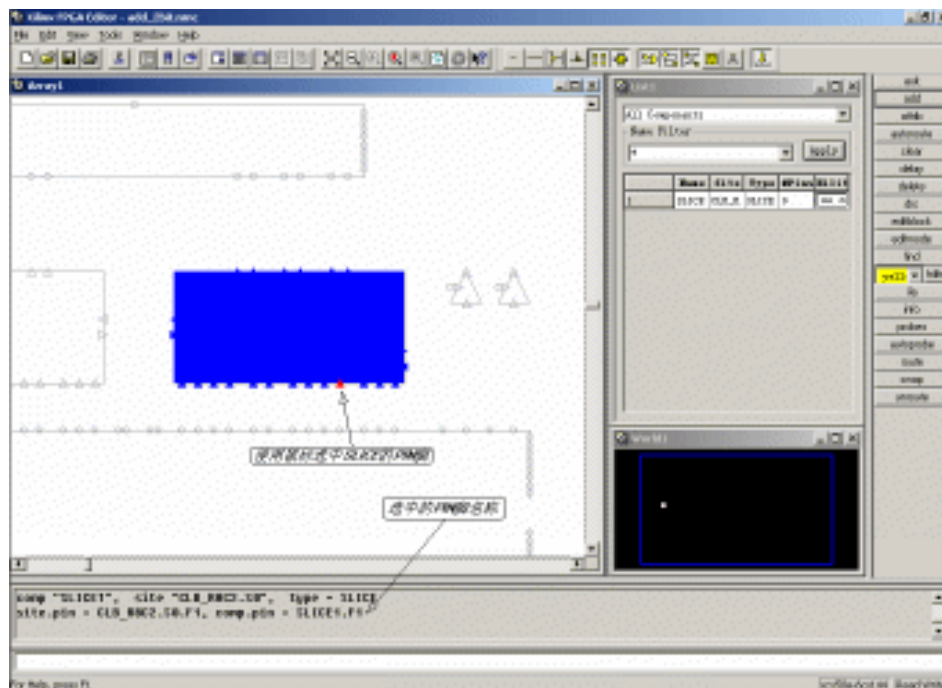


图6-37 SLICE 在 FPGA 底层编辑器中的坐标

- 单击右边按钮栏中的 **add** 按钮，打开组件属性对话框，如图 6-38 所示。在这个对话框中包括【General】、【Configuration】和【Physical Constraints】等 3 个选项卡，在【General】选项卡中可以改变组件的名称，在【Physical Constraints】选项卡中可以对当前的组件进行物理位置约束，这些选项卡的具体设置可参考相关帮助。本例在组件【Name】栏中输入“SLICE1”。

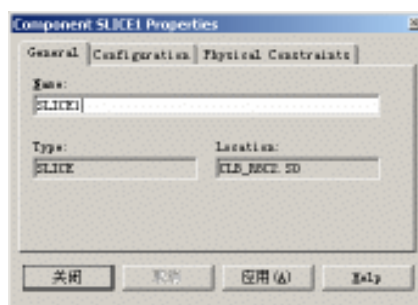


图6-38 组件属性对话框

- 在设计中使用的 SLICE 单元，在【Array】窗口中将显示为蓝色，如图 3-37 所示。为了可以在 SLICE 里面实现一定的逻辑，首先需要添加 SLICE 的端口，

本设计的目的是设计成硬宏，所以需要添加硬宏的引脚。使用鼠标选中 SLICE1 的 F1 引脚，选择菜单栏中的【Edit】/【Add Macro External Pin】命令，弹出的对话框如图 6-39 所示。在【External Name】栏中输入“ $a<0>$ ”，在【Type of Pin】中选择【INPUT】选项。使用相同的方式完成 F2-> $b<0>$ ，G1-> $a<1>$ ，G2-> $b<1>$ ，X-> $c<0>$ ，Y-> $c<1>$ ，CIN-> cin ，COUT-> $cout$ 。

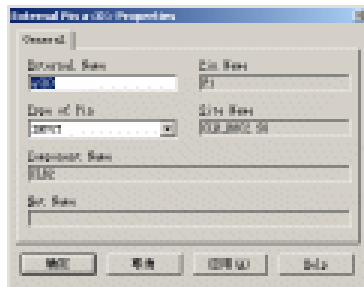


图6-39 外部引脚属性对话框

注意：这里数据线使用的括号类型与综合器有关系，主要取决于综合器总线扩展时使用的括号类型。

7. 添加了 SLICE 的端口以后，需要在 SLICE 内部添加相关的逻辑。使用鼠标选中蓝色 SLICE1，然后单击右边的 **editblock** 按钮，打开内部逻辑窗口，如图 6-40 所示。

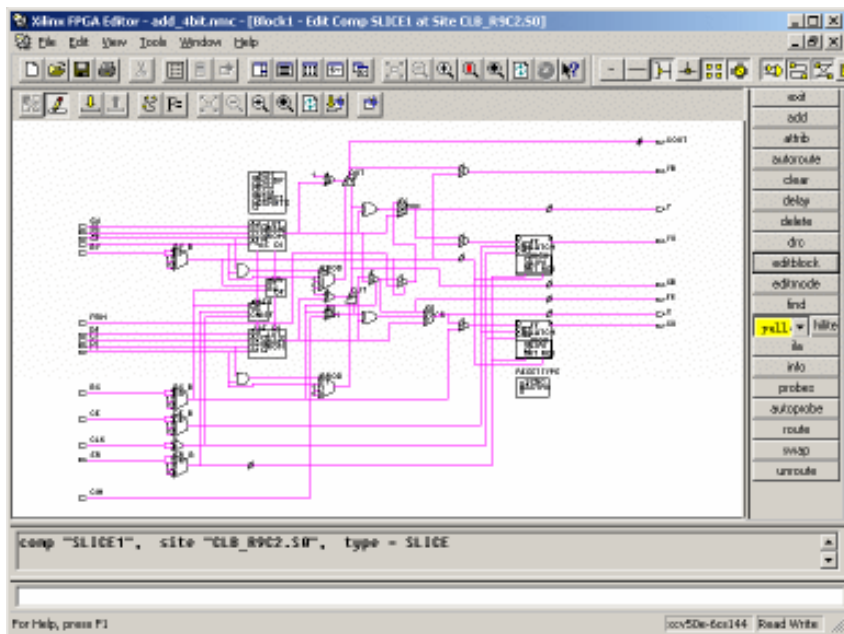


图6-40 内部逻辑编辑窗口 1

8. 单击编辑窗口上方的 **F=** 按钮，在编辑窗口的下方出现逻辑编辑窗口，如图 6-41 所示。在【Eqn】和【Geqn】栏中添加逻辑代数“ $(A1@A2)$ ”，然后使用鼠标单击上方的 **↓** 按钮，将这些逻辑应用到 SLICE 中，那么与引脚 A1 和

A2 相关的两个连线变为蓝色。在 SLICE 中，基本的逻辑包括以下几种：

- ~ NOT
- + OR
- * AND
- @ XOR

在 F 函数发生器【Feqn】和 G 函数发生器【Geqn】中都输入“(A1@A2)”。表示将地址线 A1 和地址线 A2 做异或运算。

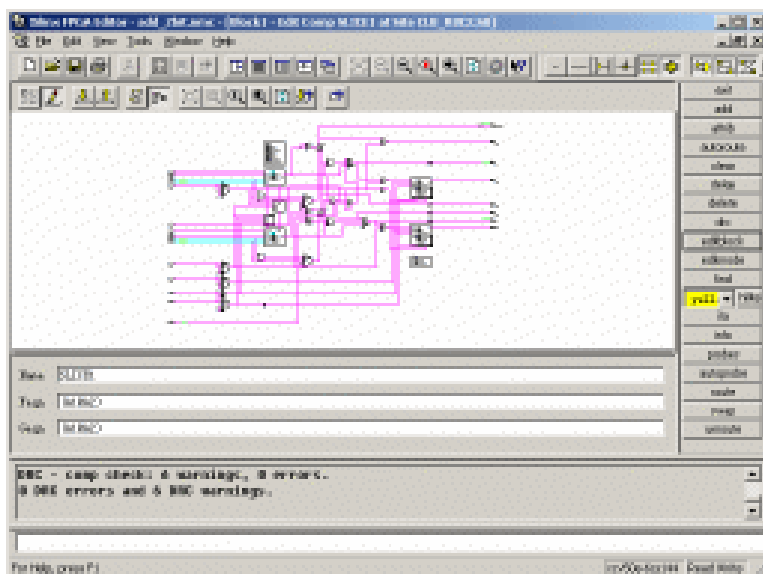


图6-41 内部逻辑编辑窗口 2

9. 选择快速进位链，使用鼠标选中 CYINIT 逻辑上的 CIN 脚，那么可以将 CIN 引入到逻辑里面。如图 6-42 所示，所有小三角部分都是可以配置的。

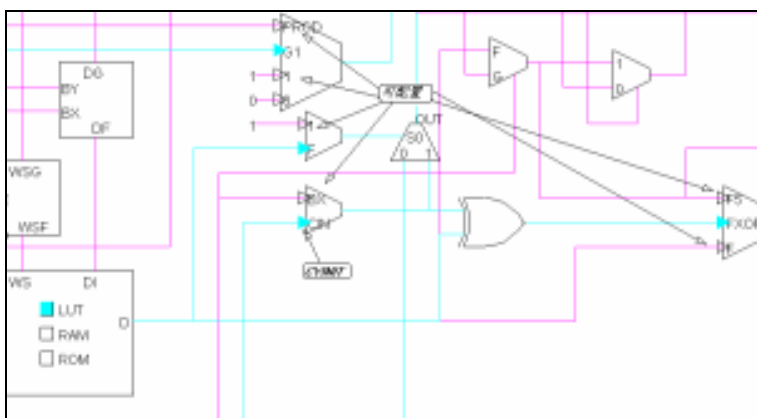


图6-42 内部逻辑的编辑方法

10. 完成 2bit 全加器以后的 SLICE 内部逻辑如图 6-43 所示。将这个设计存储以后就可以得到一个 2bit 全加器的设计硬宏单元，这个设计可以在任何 HDL 设计中进行实例化，与其他的模块并没有区别，硬宏实例化后可以保证设计的密

度与速度。

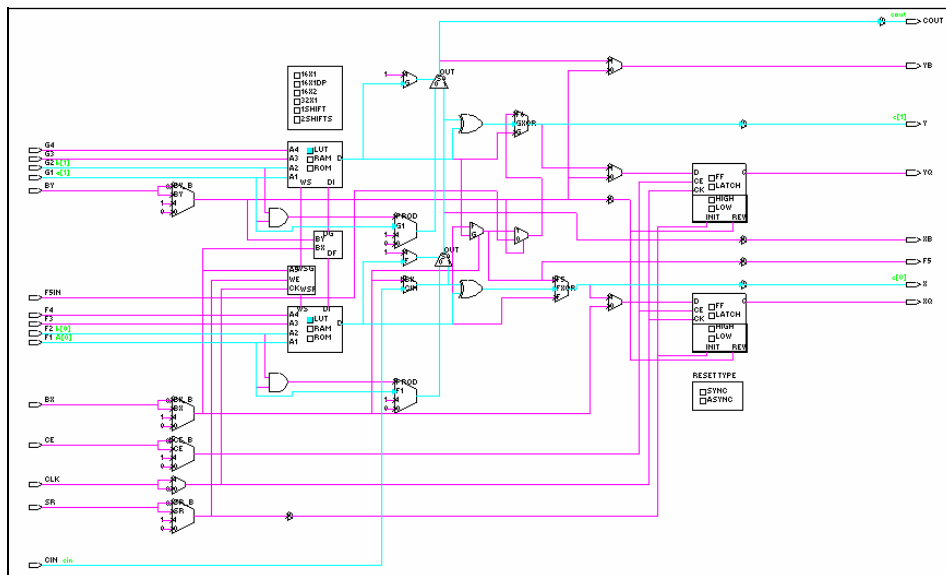


图6-43 2比特全加器内部逻辑实现图

11. 实例化方法请参考配套光盘中的“Example-6-2\Source\test_add2_macro.v”源文件，对这个程序进行综合时必须选择【XST Verilog】综合器，因为这个综合器可以将总线扩展为使用<>表示，并且需要将 add_2bit.nmc 文件拷贝到当前工程目录下。
12. 将这个模块综合实现后可以看到一个 2bit 的全加器使用了一个 SLICE 完成，达到了设计的目的。

6.4 小结

本章主要介绍了各种实现工具的使用方法，其中包括时序分析器、布局规划器以及 FPGA 底层编辑器等，合理使用这些工具可以提高设计的效率。为了使用这些工具需要对 FPGA 综合、约束、实现等过程有很清楚的认识，理解 FPGA 的时序约束有助于掌握时序分析器的使用方法。在进行布局规划的过程中需要对 FPGA 的内部结构比较了解，使用 FPGA 底层编辑器进行设计就需要对所选用的器件的内部结构以及底层单元都十分清楚，同时需要对器件的电气特性、开关特性等有深刻认识，这些特性决定了设计所能达到的性能，有助于设计者确定可实现的设计目标。

6.5 问题与思考

1. 在设计的哪些环节可以使用时序分析器进行时序分析？时序分析中各种基本时序路径与时序约束中的哪些约束相对应？
2. 设计的各个阶段使用的布局规划各自有什么特点？相互之间有什么联系？
3. FPGA 底层编辑器的探针功能如何实现？

第7章 XPower、iMPACT 和 ChipScope Pro

功耗仿真器 XPower 用于计算 FPGA/CPLD 的功耗，为选择电源模块、决定是否加散热片等提供依据，功耗的准确估计对使用电池作为电源的设计尤为重要。配置工具 iMPACT 用于生成和下载配置文件。在线逻辑分析仪 ChipScope Pro 具有传统逻辑分析仪的功能，而且价格便宜、使用方便，具有很高的实用价值。本章将详细介绍功耗仿真器 XPower、配置工具 iMPACT 和在线逻辑分析仪 ChipScope Pro 的使用方法与技巧。

本章主要内容如下：

- 功耗仿真器 Xpower；
- 配置工具 iMPACT；
- 在线逻辑分析仪 ChipScope Pro。

7.1 XPower

功耗仿真器 XPower 是第一个用于可编程逻辑器件的功耗分析软件，可以分析 FPGA 和 CPLD 的整体以及特定网线的功耗。进行功耗分析时，XPower 可以读入 ModelSim 生成的 VCD 文件，从而使功耗分析更快捷、准确。

7.1.1 XPower 综述

CMOS 电路的功耗主要来源于开关状态切换，XPower 计算功耗时给每个开关元件（LUT、FF、BRAM、布线等）建立一个电容模型，然后根据用户设置的时钟和输入信号开关频率、同步元件的翻转率、特定器件的电容、静态功耗和其他数据估算 FPGA 的功耗。准确估算功率的关键是获得准确的信号翻转率数据。

XPower 把各元件的功耗求和作为 FPGA 的总功耗，每个元件的功耗可以表示为：

$$P = C * V^2 * E * F * 1000$$

其中 P 为功耗，单位为 mW； C 为电容量，单位为法拉； V 为工作电压； E 为翻转率（每时钟周期的平均跳变次数）； F 为频率。

电容量取决于用户的设计，在选定器件、完成布局布线之后电容量也随之确定。工作电压可以在 XPower 界面中设置。 $E * F$ 为一秒钟信号开关的次数，其中 F （即元件的开关或翻转频率）是最难确定的参数，设计者可以通过设置全局默认翻转率、在 XPower 界面中为特定信号指定翻转率、读入仿真文件等方法为 XPower 提供这个参数。

XPower 的主要输入文件有 NCD、PCF、VCD、XML 和 PWR 文件，其中 PCF 文件提

供了设计的时钟频率、工作电压等信息。VCD (Value Change Dump) 文件在使用 ModelSim 进行时序仿真时产生, 它提供了所有网线的翻转率情况, 用这个文件进行功率分析可以得到更接近实际情况的结果。XML 为用户设置文件, 用于保存 XPower 的设置 (如工作电压、结温、翻转率等), 在下次打开同一设计时不必重复这些设置。XPower 的主要输出文件为 PWR 文件, 即功耗报告文件。

使用 XPower 时, 首先打开设计的 NCD 文件, 相应的 PCF 文件将自动读入, 设计者根据需要可以再读入 VCD 和 XML 文件, 然后对工作频率、电压、环境温度、输入信号的频率、输出负载电容和驱动电流、内部信号的翻转率等参数进行设置。设置完毕后, XPower 将完成功耗分析, 并生成 FPGA 功耗分析报告, 下面以同步预定标计数器为例说明 Xpower 的使用方法, 参考示例详见本书附带光盘的“Example-7-1”目录。

7.1.2 XPower 的用户界面

XPower 的启动方法有两种: 一种是直接在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【XPower】命令; 另一种方法是设计综合实现后, 在 Project Navigator 的【Processes for Source】中, 双击【Implement Design】下的【Analyze Power (XPower)】选项。XPower 的用户界面如图 7-1 所示。

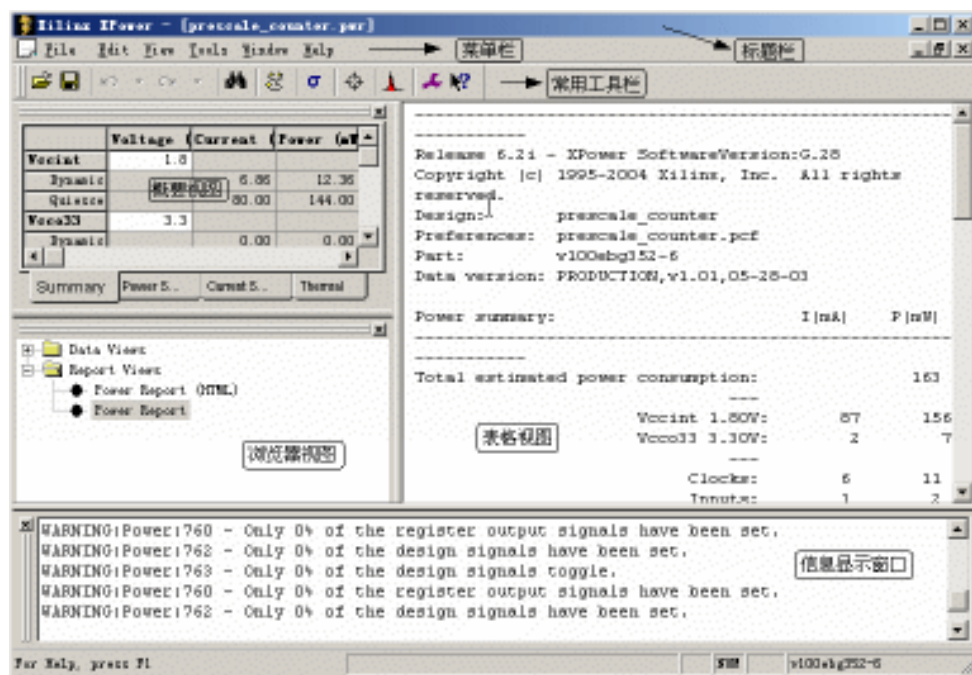


图7-1 XPower 的用户界面

XPower 的用户界面主要由标题栏、菜单栏、常用工具栏、概要视图 (Summary View)、浏览器视图 (Explorer View)、信息显示窗口 (History Bar)、表格视图 (Table View) 等组成, 各主要部分功能如下:

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）菜单除了包含打开、关闭、退出等通用文件处理命令外，还包含打开设置文件（Open Settings File）、打开仿真文件（Open Simulation File）、保存设置文件（Save Settings File）、设置另存为（Save Settings As）和从电子表格中输入数据（Import Data from Worksheet）等命令。其中从电子表格中输入数据是 ISE 6 新增加的命令，设计者可以通过 Xilinx 公司网页上的电子表格工具产生 XML 文件，然后通过这个命令读入 XML 文件进行功率分析。
- 编辑（Edit）菜单包含了撤销（Undo）、重做（Redo）、查找（Find）和用户设置（Preferences）等命令。当选择【Edit】/【Preferences】命令时，将打开【Preferences】对话框，如图 7-2 所示。

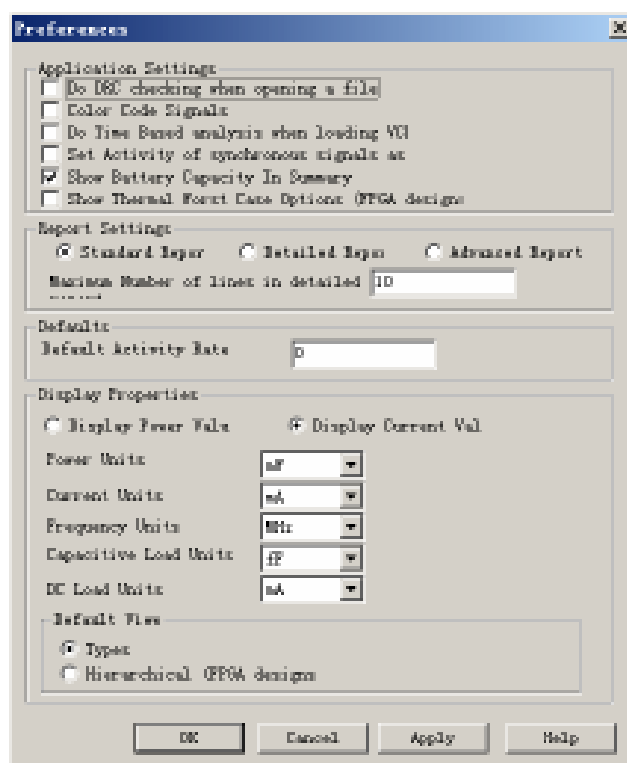


图7-2 【Preferences】对话框

【Preference】对话框的内容主要分为 5 部分，分别是【Application Settings】、【Report Settings】、【Defaults】、【Display Properties】和【Default View】。

在【Application Settings】中，可以选择 Xpower 打开文件时是否做设计规则检查（DRC）、是否使用不同的颜色区别表格视图中各种信号的设置来源（不同颜色的含义如表 7-1 所示），是否在概要视图中显示电池容量、电池寿命栏目，是否在概要视图的“Thermal”选项卡中显示最高环境温度、最大结温等。

表 7-1 信号设置来源与颜色

信号设置来源	颜色
默认设置	黑
用户设置	蓝
XPower 设置	红
来自仿真数据	橙
估计的翻转率	绿
用户默认设置	黄

在【Report Settings】中可以设置报告的繁简程度。在【Default】中可以设置默认翻转率，仿真时没有翻转、用户也没有手工设置的信号的翻转率将被制定为默认翻转率。【Display Properties】用于设置功耗分析结果以功率还是电流的形式显示，以及数值单位等。【Default View】用于设置默认视图类型，可以是类型视图（Types View）或层次视图（Hierarchical View），类型视图根据元件的类型分组显示功耗信息，层次视图根据设计的层次分组显示。



在类型视图中信号不会重复列出，易于设置初始翻转率，因此打开文件时一般选择类型视图，设置翻转率后再利用【View】/【Data Views】命令切换至其他视图。

- 视图（View）菜单中包含了数据视图（Data Views）、工具栏（Tool Bar）、状态栏（Status Bar）、概要视图（Summary View）、信息显示窗口（History Bar）和浏览器栏（Explorer View）等命令。在数据视图中，用户可以选择功耗数据的分类显示方式，包括按类型、层次、时钟域、详细时钟域等方式。
- 工具（Tools）菜单包含了 DRC 检查（Run DRC）、估算翻转率（Estimate Activity Rates）、设置默认翻转率（Set Default Activity）和报告置信细节（Report Confidence Details）等命令。
- 窗口（Window）菜单用于设置窗口排列方式等。
- 帮助（Help）菜单包含了 XPower 的版本和帮助信息。

2. 常用工具栏

XPower 的常用工具栏比较简单，各个按钮的功能分别是打开文件、保存设置文件、撤销、重做、查找、DRC 检查、估计翻转率、在信息显示窗口中显示置信水平、进入/退出最恶劣情况功耗估计模式等。

3. 浏览器视图（Explorer View）

浏览器视图包含了【Data Views】和【Reports】两项，【Data Views】中包括了信号和逻辑的名称，选中这些信号和逻辑可以在表格视图中查看和设置参数。双击【Reports】下的【Power Report】可以生成功耗报告。

4. 概要视图 (Summary View)

概要视图显示了当前计算的结果，各选项卡中白色的项目用户可以设置，其他灰色的文本编辑框为计算结果，不能编辑。

5. 表格视图 (Table View)

表格视图用于显示设计元件的功耗和功耗报告。

6. 信息显示窗口 (History Bar) (翻译不准确，请修改)

信息显示窗口显示了 XPower 的工作状态。

7.1.3 用 XPower 分析功耗

当设计完成布局布线之后，在 ISE Navigator 中可以直接启动 XPower，这时 XPower 会读入 NCD 和 PCF 文件，如果工作目录中有 XML 文件，XPower 会自动读入，在信息显示窗口可以看到相关的记录。设计者根据需要可以再读入 VCD 文件，设置电压、温度、信号翻转率等参数之后，就可以生成功耗报告。

设计者也可以单独打开 XPower，然后选择【File】/【Open File】命令打开相关文件，弹出的对话框如图 7-3 所示。



图7-3 【Open】对话框

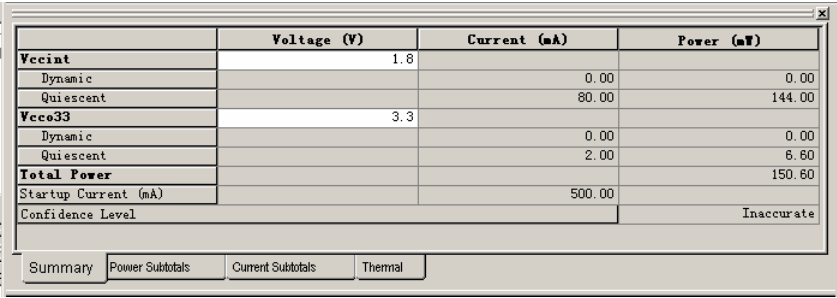
打开文件时，首先选择设计文件（NCD 文件），相应的 PCF、XML 文件将会自动加载，然后再选择仿真文件（VCD 文件）和浏览器栏视图的显示类型，单击 **OK** 按钮完成相关文件的读入。

进行功耗分析时，需要设置电压、温度、翻转率等参数，分别说明如下。

1. 设置电压和温度

进行功耗分析时，首先要在概要视图中设置 FPGA/CPLD 的工作电压和环境温度。

概要视图有 4 个选项卡，设计者可以在概要（Summary）选项卡和热（Thermal）选项卡中设置参数。概要选项卡如图 7-4 所示，列出了 Vccint 和 Vcco 的电压、电流、功耗，以及总功耗、启动电流和功耗估计的置信度，设计者可以设置其中的 Vccint、Vcco 电压。

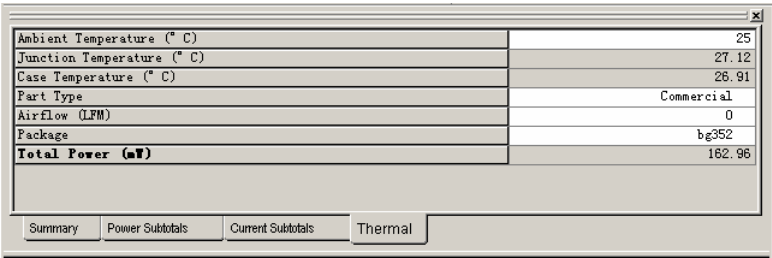


	Voltage (V)	Current (mA)	Power (mW)
Vccint	1.8		
Dynamic		0.00	0.00
Quiescent		80.00	144.00
Vcco33	3.3		
Dynamic		0.00	0.00
Quiescent		2.00	6.60
Total Power			150.60
Startup Current (mA)		500.00	
Confidence Level			Inaccurate

Summary Power Subtotals Current Subtotals Thermal

图7-4 概要选项卡

热选项卡列出了环境温度、结温、封装温度、器件级别、气流、封装类型和设计的总功耗，如图 7-5 所示。



Ambient Temperature (°C)	25
Junction Temperature (°C)	27.12
Case Temperature (°C)	26.91
Part Type	Commercial
Airflow (LFM)	0
Package	bg352
Total Power (mW)	162.96

Summary Power Subtotals Current Subtotals Thermal

图7-5 热选项卡

设计者可以尝试改变 FPGA 的环境温度、气流、器件封装等参数来观察总功耗的变化情况。

2. 设置翻转率

信号的翻转率是指在一个时钟周期内信号发生一次跳变的概率，如果一个信号在每个时钟周期发生一次跳变，那么信号的翻转率为 100%，该信号的频率为时钟频率的一半，也就是 100%的翻转率对应 1/2 的时钟频率。

XPower 允许改变时钟、信号、逻辑和输出的翻转率，以便设计者观察特定元件的翻转率变化对功耗的影响。为了得到更准确的功耗估计，改变不同元件的翻转率时应按以下顺序进行：

- (1) 设置或检查工作电压和环境温度。
- (2) 设置或检查时钟频率。
- (3) 设置或检查输入频率。
- (4) 设置或检查输出负载。
- (5) 估计并设置全局默认翻转率。
- (6) 估计并设置特定信号的特性。

工作电压和环境温度都可以在概要视图中设置。设置全局默认翻转率时，选择【Tools】/【Set Default Activity】命令，然后在弹出的对话框中输入相应的数值，这个默认翻转率将被用于在 VCD 文件中没有设置翻转率的元件。

The screenshot shows the Icing XPower application interface. The main window displays a table of power analysis results for a file named "prescale_counter.pof". The table has columns for Name, Frequency, Capacitive Load (fF), DC Load (mA), Enable Rate (%), and Current (A). The results are organized by component type, starting with counters and followed by various multiplexers (Mux) and registers.

Name	Frequency	Capacitive Load (fF)	DC Load (mA)	Enable Rate (%)	Current (A)
counter_0	5.00				
_n0000	5.00			0.00	
counter_1	2.50				
_n0002[1]	2.50			0.00	
Muxd_n0001_inout	1.25				
Muxd_n0001_inout	1.25				
Muxd_n0001_inout	0.31				
Muxd_n0001_inout	0.16				
Muxd_n0001_inout	0.04				
_n0001[7]	0.03				
Muxd_n0001_inout	0.01				
Muxd_n0001_inout	0.00				
Muxd_n0001_inout	0.00				
reset_IBUF	0.00				
Muxd_n0001_inout	0.00				
Muxd_n0001_inout	0.00				
Muxd_n0001_inout	0.00				
Muxd_n0001_inout	0.00				
Muxd_n0001_inout	0.00				
Muxd_n0001_inout	0.00				
_n0001[21]	0.00				
counter_10	0.00				
counter_11	0.00				
counter_12	0.00				
counter_13	0.00				

The left sidebar shows a tree view of Data Views, including Types, Clocks, Inputs, Logic, Outputs, Signals, and Report Views. The bottom status bar indicates the current file is "prescale_counter.pof" and provides additional information like "For Help, press F1" and "x100ahg352-6".

图7-6 信号显示在表格视图中

设计者可以在表格视图对翻转率等参数进行设置，“Power”列为信号的功耗。有些参数不允许设置，例如图 7-6 的表格视图中列出了所有内部信号，设计者可以设置翻转率，即图中的“Frequency”，但是“Capacitive Load”、“DC Load”和“Enable Rate”等参数不允许设置。“Clocks”、“Inputs”、“Logic”和“Outputs”等文件夹中的信号也有类似的情况。打开“Logics”文件夹时，所显示的信号参数都不能修改，这些参数都是在“Signals”文件夹中修改后自动映射过来的。

经过以上步骤，用于计算 FPGA/CPLD 功耗的工作电压、环境温度、时钟频率、信号翻转率、输出负载等参数都设置完毕，选中【File】/【Save Settings File】命令保存设置，那么下次分析功耗时就不用重新设置所有参数。然后打开“Reports”文件夹，选中“Power Report”可以得到 FPGA/CPLD 的功耗报告，报告的格式可以通过选择【Edit】/【Preferences】命令，在弹出的对话框中进行设置。

3. 加载仿真数据

XPower 可以加载 ModelSim 生成的 VCD 文件，这个文件可以提供最全面的翻转率信息，减少了手工输入信号翻转率的繁琐，也减少了误差，提高了 FPGA 功耗估计的效率和准确性。

选择【File】/【Open Simulation File】命令可以加载仿真文件。选择【Edit】/【Preferences】命令，在弹出的对话框中可以设置每次打开设计文件时可以同时打开相应的

仿真文件。当设计较大和仿真时间较长时，VCD 文件可能很大，加载时间可能长达数分钟。

对 Verilog，在 testbench 中加入以下语句即可生成 VCD 文件。

```
initial begin
    $dumpfile ("my_design.vcd");
    $dumpvars(1, t.uut);
end
```

其中 mydesign.vcd 为生成的 VCD 文件名，可以根据需要修改。t 为 testbench 的模块名，uut 为待测试模块的实例名，读者可以参考附带光盘“Example-7-1”目录下的 testbench.tf 文件。

对于 VHDL，使用 ModelSim 时，可以在 do 文件中加入以下语句以生成 VCD 文件。

```
vcd file my_design.vcd
vcd add testbench/uut/*
```

VCD 文件有使用方便、估计准确的特点，建议读者尽量使用。

7.2 iMPACT

iMPACT 是 ISE 集成的配置工具，具有生成 PROM 格式的下载文件、向 FPGA/CPLD/PROM 下载配置文件、验证配置数据是否正确等功能。ISE 将 PROM 配置文件分割器（PROM File Formatter）集成到 iMPACT 中，使 FPGA/CPLD 配置文件的分割和下载更符合用户的操作习惯。

7.2.1 iMPACT 综述

FPGA 的配置模式有 4 种，分别是主串模式（Master Serial）、从串模式（Slave Serial）模式、SelectMap 模式和边界扫描模式（Boundary Scan）。在从串模式中，FPGA 接受来自外部 PROM 或其他器件的串行配置数据，在外部时钟 CCLK 的作用下完成配置，多个 FPGA 可以连接成菊花链（Daisy-chain），以便从一个数据源中获得配置数据。在主串模式中，待配置的 FPGA 产生配置时钟 CCLK 驱动外部串行 PROM，从而读入配置数据。Virtex 系列 FPGA 支持 SelectMap 模式，其配置数据是并行的，因此是最快的配置模式。在 SelectMap 模式下，FPGA 接受来自外部的并行配置数据和配置时钟 CCLK。边界扫描模式（即 JTAG 模式）通过 IEEE 1149.1 Test Access Port（即 JTAG 口）配置 FPGA/CPLD/PROM 等器件，由于这些器件都有 JTAG 口，因此可以节省管脚资源，在这种模式下，每个 TCK 传送 1bit 配置数据。

iMPACT 支持 4 种下载模式：边界扫描（Boundary Scan）模式、从串模式（Slave Serial）模式、SelectMap 模式和 Desktop 配置模式。其中 Desktop 配置模式是一个高速配置模式，可以配置 Xilinx FPGA/CPLD/ISP PROM/System ACE MPM，但是要用专门的硬件设备（Xilinx MultiPRO Desktop Tool）来实现。边界扫描模式标准统一、设备简单，可以通过 JTAG 口配置 FPGA/CPLD/PROM 等多种器件，因此用得最多，本节主要以边界扫描模式为

例说明 iMPACT 的使用方法，其他模式的使用方法与边界扫描模式基本相同。

iMPACT 的主要输入文件是 JEDEC 文件和 BIT 文件。JEDEC 文件由 Xilinx CPLD Fitter 产生，用于配置 CPLD。BIT 文件由 FPGA 综合实现工具产生，用于配置 FPGA，JTAG 边界扫描链的每一个 FPGA 需要一个 BIT 文件。iMPACT 的输出文件主要有 MCS/EXO 等格式的 PROM 文件、边界扫描链描述文件（CDF，Chain Description File）等。PROM 文件的大小与所使用的 PROM 相适应，以保证文件能下载到器件中。CDF 文件描述了边界扫描链结构、器件类型和配置文件路径等信息。

使用并口电缆下载 FPGA 配置文件到 PROM 时，主要工作过程是启动 iMPACT，在文件模式中用 PROM Formatter 把 BIT 转为 MCS/EXO 格式的 PROM 文件，然后在配置模式中初始化边界扫描链，以便 iMPACT 读入硬件连接信息，接着就可以下载配置文件。

7.2.2 iMPACT 的用户界面

iMPACT 的启动方法有两种：一种是直接在 Windows 界面下运行【开始】/【程序】/【Xilinx ISE 6】/【Accessories】/【iMPACT】命令；另一种方法是设计综合实现后，在 Project Navigator 的【Processes for Source】中的【Generate Programming File】下双击【Generate PROM,ACE,or JTAG File】或【Configure Device (iMPACT)】。iMPACT 的用户界面如图 7-7 所示。

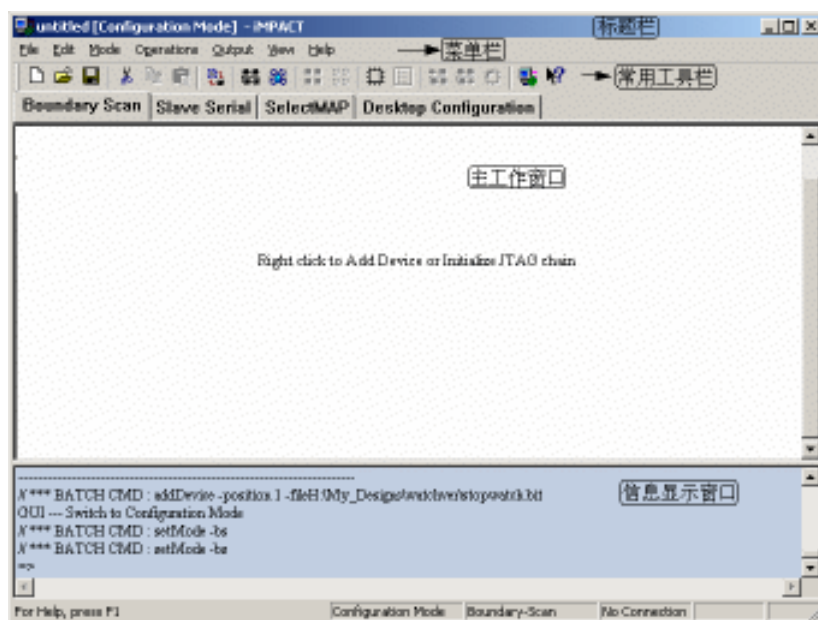


图7-7 iMPACT 的用户界面

iMPACT 的用户界面主要由标题栏、菜单栏、常用工具栏、主工作窗口和信息显示窗口组成，各主要部分功能如下：

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）菜单包含了常见文件操作等命令，其中【Initialize Chain】命令自动检测边界扫描链中器件的数目和类型，【Start Debug Chain】命令用于对边界扫描链进行测试。
- 编辑（Edit）菜单包含了常见的编辑操作命令以及一些和配置相关的命令，其中【Add Device】命令用于手动创建边界扫描链时添加 PROM 或 FPGA/CPLD 器件，【Assign Configuration File】命令的作用是给器件指定配置文件。【Edit Collection】命令用于对 System ACE CF 进行操作，System ACE CF 与 System ACE MPM、System ACE SC 都是 Xilinx 提供的配置解决方案，包含存储单元和配置控制电路等单元，统称 SystemACE（System Advanced Configuration Environment）。【Edit PROM】命令用于修改和删除边界扫描链中的 PROM。【Preferences】命令提供了操作的选项，选择该命令后弹出的【Preferences】对话框如图 7-8 所示。

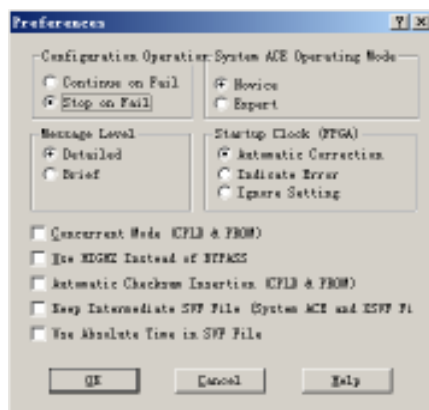


图7-8 【Preferences】对话框

在【Preferences】对话框中，【Configuration Operation】用于确定配置过程发生错误时继续或中止配置，【Message Level】用于选择显示详细或简要的信息，【System ACE Operating Mode】用于选择 System ACE 的操作模式，选项有新手/专家两种，【Startup Clock (FPGA)】用于确定当 iMPACT 发现包含在 FPGA Bitstream 文件中的初始配置时钟与当前使用模式不匹配时，iMPACT 改正驻留在计算机内存的初始配置时钟设置（但是 BIT 文件不变）、提示错误并中止配置过程、还是显示警告信息并忽略初始配置时钟不匹配的问题。选中【Concurrent Mode (CPLD & PROM)】选项可以擦除边界扫描链中多个 PLD，节省时间。选中【Automatic Checksum Insertion (CPLD & PROM)】选项可以使 iMPACT 自动在 CPLD 或 PROM 的 USERCODE 寄存器中插入校验和。选中【Keep intermediate SVF file (System ACE)】选项可以保留 System ACE 文件生成过程中的 SVF 中间文件。



通常对边界扫描链中的一个器件进行配置时，其他器件处于 BYPASS 状态，如果怀疑噪声影响了配置过程，可以选中【Use HIGHZ instead of BYPASS】选项，使其他的 Xilinx XC9500/XL/XV 和 Virtex 系列器件处于高阻状态，有利于减小配置操作过程中的干扰。

【Launch Wizard】命令用于启动配置向导，配置向导的界面和当前所处的模式以及选

中的选项卡有关，在配置模式下选中【Boundary-Scan】选项卡时，配置向导会让设计者选择自动检测边界扫描链还是手动输入扫描链，如果已经连接硬件，建议使用前者。在文件模式下选中【PROM Formatter】选项卡时，选择【Launch Wizard】命令，弹出的对话框如图 7-9 所示，设计者可以从中选择 PROM 的形式和 PROM 文件的格式、文件名、路径等。除了手动启动配置向导外，iMPACT 每次启动时会自动打开。

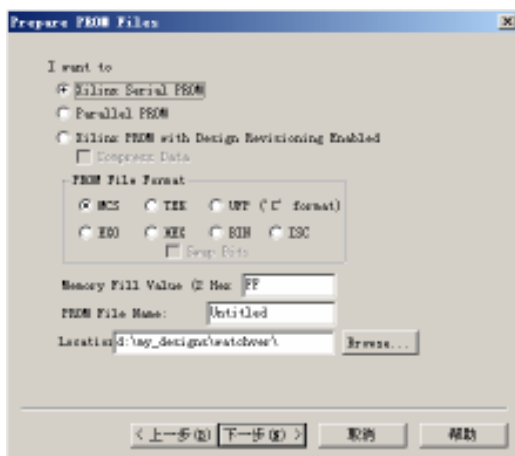


图7-9 配置向导对话框

- 模式 (Mode) 菜单提供了使 iMPACT 在文件模式和配置模式之间进行切换的命令，文件模式用于产生配置文件，配置模式用于下载配置文件。使用 iMPACT 时要注意当前所处的模式，因为不同模式下的菜单、操作都有很大的不同。
- 操作 (Operations) 菜单在 iMPACT 的不同模式下有不同的命令，在配置模式下，【Operations】菜单包含了下载配置文件 (Program)、验证 (Verify)、擦除 (Erase)、功能测试 (Functional Test)、空白检查 (Blank Check)、回读 (Readback)、下载用户电子签名 (Program XPLA UES)、读取器件识别码 (Get Device ID)、计算校验和 (Get Device Checksum)、读取签名和用户码 (Get Device Signature/Usercode)、读取用户电子签名 (Get XPLA device UES)、激活 On-The-Fly 模式 (Activate On-The-Fly pattern)、验证用户码 (Validate Usercode)、循环读取器件识别码 (IDCODE Looping) 等命令，如图 7-10 所示。在文件模式下，【Operations】菜单包括了生成文件 (Generate File)、拷贝至 Compact Flash (Copy to Compact Flash)、Compact Flash 完整性检查 (Compact Flash Integrity Check) 等命令，其中后两个命令用于 System ACE。
- 输出 (Output) 菜单包含了电缆自动连接 (Cable Auto Connect)、电缆设置 (Cable Setup)、电缆复位 (Cable Reset)、断开电缆 (Cable Disconnect) 和 SVF、STAPL 文件操作命令。
- 视图 (View) 菜单包含了状态栏 (Status Bar)、查看记录文件 (View Log File)、清除信息显示窗口 (Clear Log Window)、设置文本编辑器 (Set Text

Editor) 和刷新 (Refresh) 等命令。

- 帮助 (Help) 菜单包含了 iMPACT 在线帮助和版本等信息。

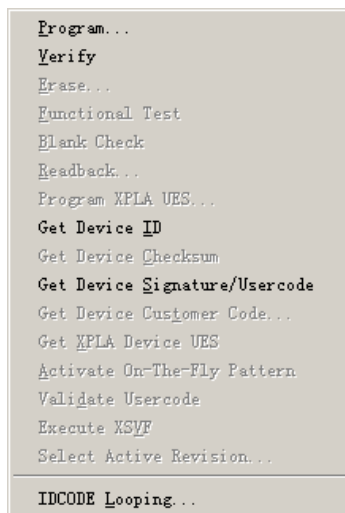


图7-10 【Operations】菜单

2. 常用工具栏

常用工具栏提供了各种常用命令的快捷操作按钮。

-  (Toggle Mode) 按钮的功能是使 iMPACT 在配置模式和文件模式之间切换。
-  (初始化边界扫描链) 按钮的功能是自动检测边界扫描链中器件的数目和类型。
-  (Debug Chain) 按钮的功能是调试边界扫描链。
-  (Select All) 按钮的功能是选中边界扫描链中所有器件。
-  (Unselect All) 按钮的功能是取消选中边界扫描链中的器件。
-  (Add Xilinx Device) 按钮的功能是在边界扫描链中加入新的 Xilinx 器件。
-  (Assign New Configuration File) 按钮的功能是为当前器件指定新的配置文件。
-  (Program) 按钮的功能是向选中的器件下载配置文件。
-  (Verify) 按钮的功能是验证下载的配置文件是否正确。
-  (Erase) 按钮的功能是擦除器件的内容。

7.2.3 用 iMPACT 下载配置文件

iMPACT 支持的 4 种下载模式的界面和操作基本相同，其中边界扫描模式用得最多，下面以边界扫描模式为例说明用 iMPACT 下载配置文件的方法。

用 iMPACT 配置 FPGA/CPLD 的过程主要分两步，第一步是准备 PROM 等配置文件，第二步是下载配置文件，这两步操作都可以利用配置向导完成，这里以秒表设计为例说明使

用 iMPACT 下载配置文件的过程, 参考示例详见本书附带光盘的“Example-7-2”目录。

🔑 准备 PROM 文件

1. 在 ISE 的“Processes for source”窗口中双击“Generate PROM,ACE or JTAG File”启动 iMPACT, 如图 7-11 所示, 配置向导自动弹出, 选中【Prepare Configuration】单选项, 单击 **下一步(N) >** 按钮, 弹出【Prepare Configuration Files】对话框, 如图 7-12 所示。

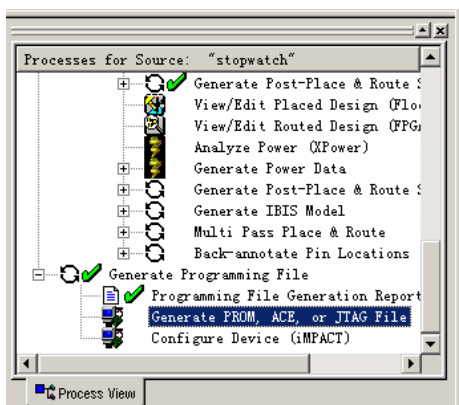


图 7-11 启动 iMPACT

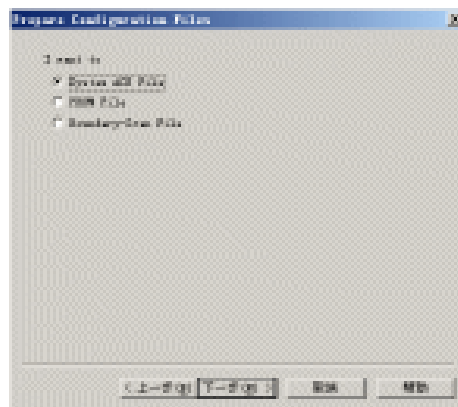


图 7-12 【Prepare Configuration Files】对话框

2. 在实际使用中, 硬件结构以 FPGA+PROM 为主, 因此选中【PROM file】单选项, 准备产生 PROM 文件, 然后单击 **下一步(N) >** 按钮, 出现【Prepare PROM Files】对话框, 如图 7-13 所示。

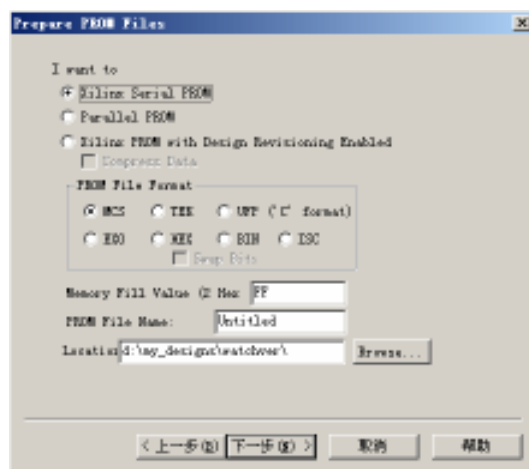


图7-13 【Prepare PROM Files】对话框

3. 使用 Xilinx 的 XC17 和 XC18V 等串行 PROM 器件时, 选中【Xilinx Serial PROM】、【MCS】或【EXO】选项, 在【PROM file】和【Location】栏分别设置生成的 PROM 文件的名字和路径, 单击 **下一步(N) >** 按钮, 出现【Specify Xilinx Serial PROM device】对话框, 如图 7-14 所示。



图7-14 【Specify Xilinx Serial PROM device】对话框

4. 根据边界扫描链所用的 PROM 在下拉列表中选择相应的型号，其中 XC17 系列为一次可编程器件（OTP，One-time-programmable），XC18 系列可以擦写多次，XCF 系列是比较新的产品，采用 FLASH 工艺，可以重复编程多次，价格比 XC18 降低了很多。如果不清楚应该选什么型号的 PROM，可以选中【Auto select PROM】单选项，iMPACT 会自动选择合适的 PROM。确定 PROM 型号之后，单击 **Add** 按钮把 PROM 添加到列表中，在列表中可以添加多个 PROM，直到满足 FPGA/CPLD 的容量要求，然后单击 **下一步(N) >** 按钮，弹出【File Generation Summary】对话框，如图 7-15 所示。

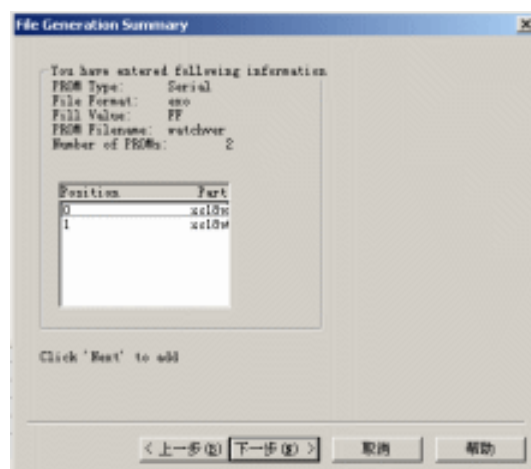


图7-15 【File Generation Summary】对话框

5. 【File Generation Summary】对话框中列出了选择的 PROM 的种类、型号、PROM 文件的格式和文件名等信息，确认无误后，单击 **下一步(N) >** 按钮，出现【Add Device File】对话框，如图 7-16 所示。

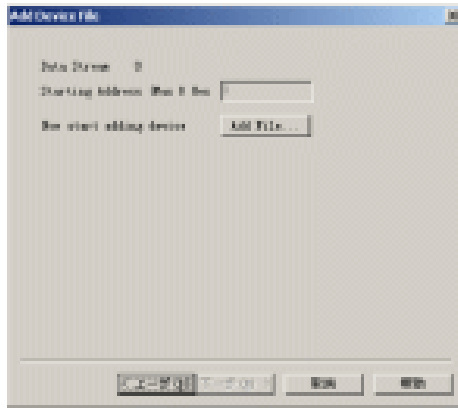


图7-16 【Add Device File】对话框

6. 在【Add Device File】对话框中单击 **Add File...** 按钮，弹出【Add Device】对话框，如图 7-17 所示。

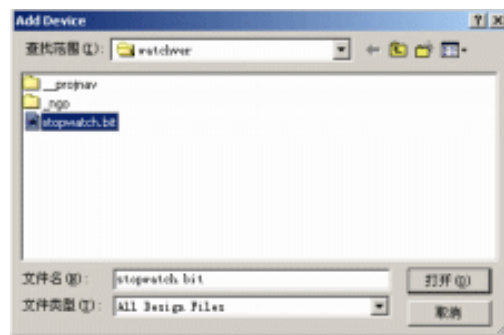


图7-17 【Add Device】对话框

7. 在【Add Device】对话框中选择需要转换的 BIT 或 RBT 文件，然后单击 **打开(O)** 按钮，iMPACT 会读入选中的设计文件，并提示是否需要添加另一个设计文件，如图 7-18 所示。

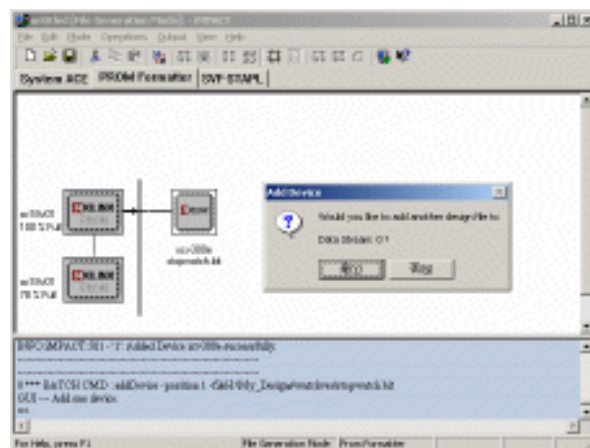


图7-18 提示是否需要添加另一个设计文件

8. 如果边界扫描链中有多个 FPGA/CPLD，可以单击 **是(Y)** 按钮，继续添加 BIT 或 RBT 文件。设计文件添加完成后，单击 **否(N)** 按钮，出现图 7-19 所示的对话框，其中显示了部分配置信息。

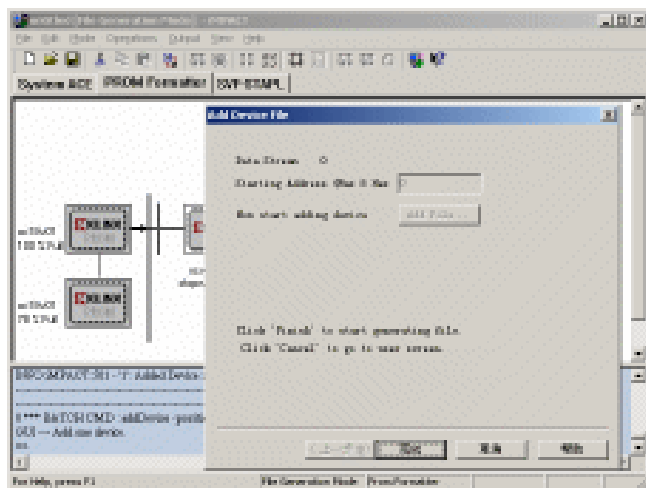


图7-19 显示配置信息

9. 单击 **完成** 按钮，iMPACT 会弹出【PROM File Generation】对话框，要求确认是否生成 PROM 文件，如图 7-20 所示，单击 **是(Y)** 按钮，iMPACT 将生成 PROM 文件。

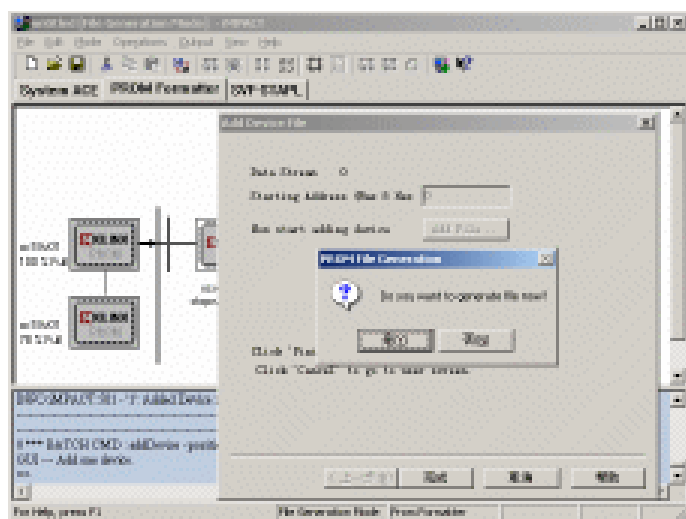


图7-20 确认是否生成 PROM 文件

需要注意的是，生成 BIT 文件时，“FPGA Start-Up Clock”应设置为 CCLK，而不是 JTAG Clock，如图 7-21 所示。前者用于 Slave-serial 下载模式，后者用于 JTAG 下载模式，不能互换。

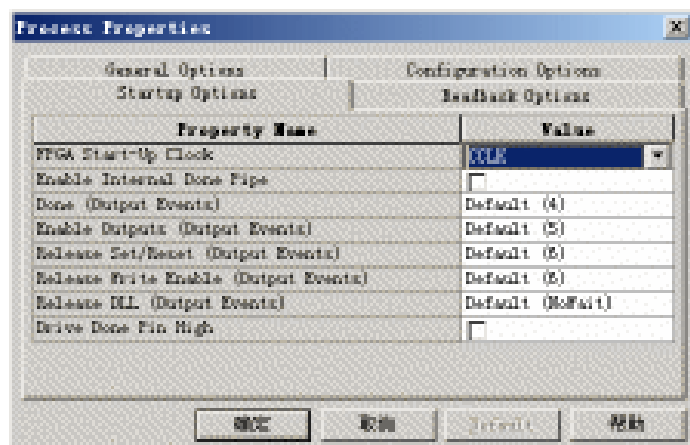


图7-21 Slave-serial 模式下 FPGA Start-Up Clock 的设置

生成 PROM 文件后，可以选择【File】/【Save As】命令把边界扫描链的结构和各器件所指定的文件的相关信息保存在 CDF 文件中，下次改变个别器件的型号或配置文件时，打开 CDF 文件就可以得到边界扫描链的信息，不必重复以上步骤，修改之后选择【Edit】/【Generate File】命令就可以生成配置文件。

🔑 下载配置文件

Slave-serial 配置模式使用 PROM 文件配置 FPGA，JTAG 模式使用 BIT 文件配置 FPGA，调试时 JTAG 模式用得比较多，下面以 JTAG 模式为例说明下载配置文件的过程。

1. 首先生成 BIT 文件，在【Processes for source】窗口中双击“Generate Programming File”即可，需要注意的是“FPGA Start-Up Clock”应设置为 JTAG Clock，如图 7-22 所示。

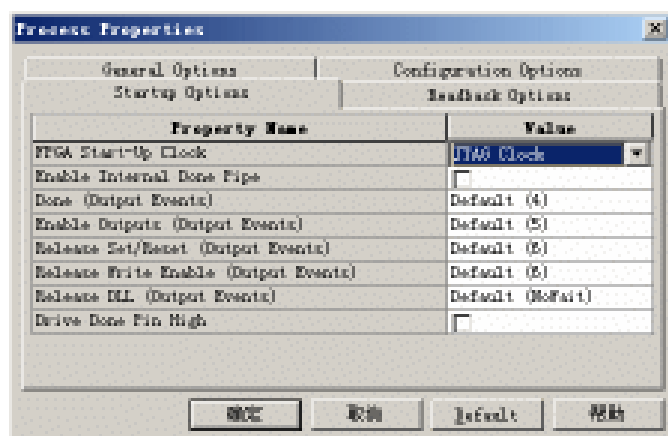


图7-22 JTAG 模式下 FPGA Start-Up Clock 的设置

2. 在配置模式下选择【Edit】/【Launch Wizard】命令，iMPACT 提示将清除当前模式中的内容，如图 7-23 所示，单击 按钮后，iMPACT 提示把之前的工作保存在 CDF 文件中，设计者可以根据需要选择。

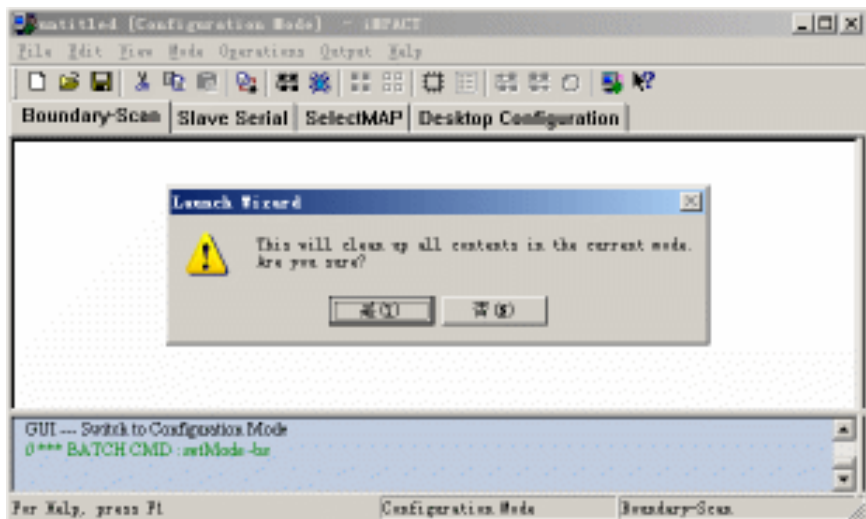


图7-23 提示清除当前内容

处理完后，出现【Boundary-Scan Mode Selection】对话框，如图 7-24 所示。

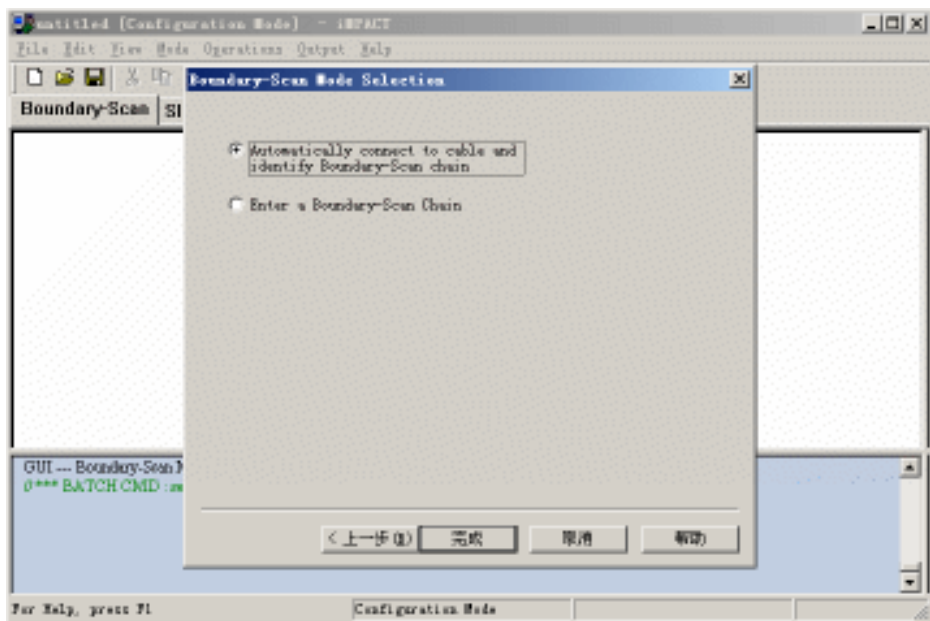


图7-24 【Boundary-Scan Mode Selection】对话框

3. 【Boundary-Scan Mode Selection】对话框包含两个单选项，用于确定如何得到边界扫描链的结构。第一项是自动连接电缆并识别边界扫描链的结构，第二项是手工输入边界扫描链。由于手工输入比较麻烦、容易出错，因此在计算机连接了硬件电路时应尽可能选择第一项。这里选择了自动识别边界扫描链，单击 **完成** 按钮，iMPACT 将显示实际的边界扫描链的结构，同时提示将为边界扫描链中的器件指定配置文件，如图 7-25 所示。

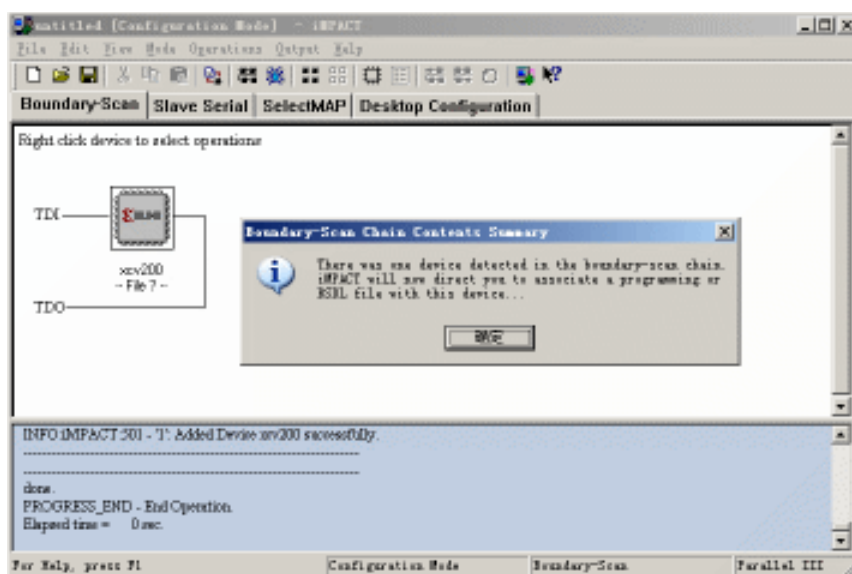


图7-25 显示实际的边界扫描链

4. 单击 **确定** 按钮，弹出【Assign New Configuration File】对话框，要求为各个器件指定配置文件，如图 7-26 所示。

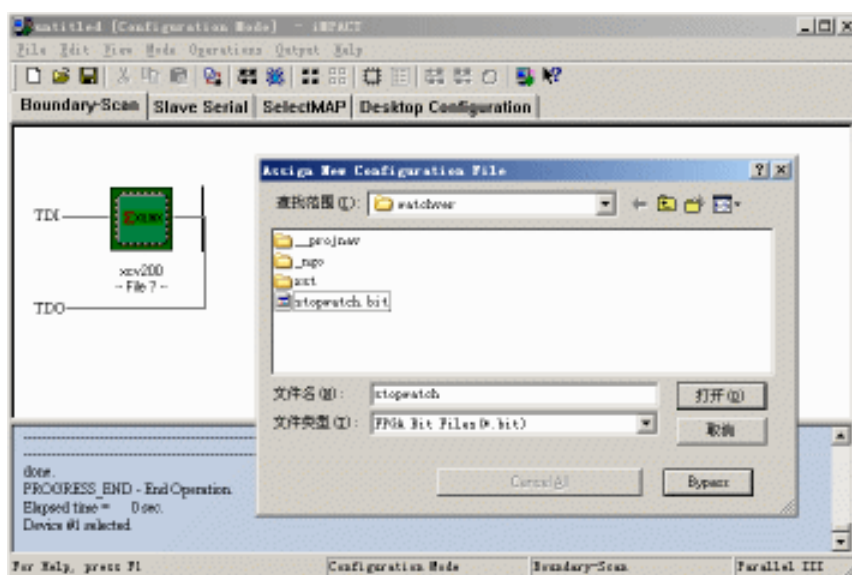


图7-26 为器件指定配置文件

FPGA JTAG 模式的配置文件分别是 BIT 文件，在综合实现流程中由 BitGen 程序产生，在进行配置文件下载之前，要准备好所用到的配置文件。在图 7-26 所示的对话框中选中 stopwatch.bit 文件，单击 **打开(O)** 按钮就为选中的器件指定了配置文件。如果扫描链中存在多个 FPGA，重复这个过程直到为要进行配置的所有器件都指定了配置文件，然后就可以下载配置文件了。

5. 下载配置文件时，选中一个或多个 FPGA，单击右键弹出命令菜单，选中

【Program】命令，会弹出【Program Options】对话框，如图 7-27 所示。

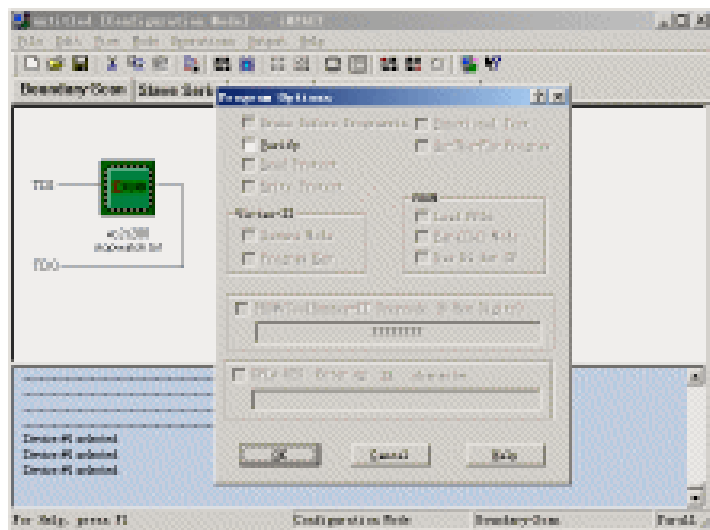


图7-27 【Program Options】对话框

设计者可以对下载配置文件的选项进行设置，例如可以选中【Erase Before Programming】（下载配置文件之前先擦除）、【Verify】（下载配置文件之后验证芯片的内容是否正确）等选项，选项设置完毕后，单击  按钮下载配置文件，下载配置文件的状态显示在信息显示窗口中，下载结束后将提示是否成功。



在【PROM/CoolRunner- II Usercode】文本框中输入用户码时，可以按照一定的规律设计用户码，以便标识 FPGA 的用户、版本和配置日期等信息，这在实际设计中有重要作用。

7.3 ChipScope Pro

在 FPGA 调试阶段，一般都要使用逻辑分析仪观察内部信号，但是传统逻辑分析仪价格高昂，而且随着 FPGA 器件规模的增大，使用传统逻辑分析仪加上大量探头来观察内部信号变得越来越困难。ChipScope Pro（在线逻辑分析仪）具有传统逻辑分析仪的功能，而且价格便宜、不需要附加大量探头，具有很高的实用价值。

7.3.1 ChipScope Pro 综述

在 FPGA 调试阶段，传统的信号分析手段是用逻辑分析仪分析信号，设计时要求 FPGA 和 PCB 设计人员保留一定数量 FPGA 管脚作为测试管脚，编写 FPGA 代码时将需要观察的信号作为模块的输出信号，在综合实现时再把这些输出信号锁定到测试管脚上，然后连接逻辑分析仪的探头到这些测试脚，设定触发条件，进行观测。这个过程比较复杂、灵活性差，PCB 布线后测试脚的数量就固定了，不能灵活增加，当测试脚不够用时影响测试，如果测试脚太多又影响 PCB 布局布线。另外，一台逻辑分析仪要几万到几十万元，每个公司拥有的数量有限，在研发

高峰期间往往供不应求，影响进度。使用 ChipScope Pro 能较好地解决这些问题。

ChipScope Pro 是针对 Xilinx Virtex-II Pro/Virtex/Virtex-II/Virtex-E/Virtex-EM/Spartan-IIIE/Spartan-IIIE 系列 FPGA 的在线片内信号分析工具，它的主要功能是通过 JTAG 口，在线、实时地读出 FPGA 的内部信号。ChipScope Pro 的基本原理是利用 FPGA 中未使用的 BlockRam，根据用户设定的触发条件将信号实时地保存到这些 BlockRam 中，然后通过 JTAG 口传送到计算机，最后在计算机屏幕上显示出时序波形。

使用 ChipScope Pro 观察 FPGA 内部信号具有以下优点：

- (1) 成本低廉。只要用这套软件加上一根 JTAG 电缆就可进行信号分析。
- (2) 灵活性大。观察信号的数量和存储深度由器件剩余的 Block-Ram 数量决定，剩余 Block-Ram 越多，可观察信号的数量和存储深度就越大。
- (3) 使用方便。ChipScope Pro 可以自动读取设计网表（NGC、EDF、EDN 文件），自动将其 IP 核的网表插入到原设计的网表中。可以方便地选择待观测信号，也可以设置复杂的触发条件。ChipScope Pro 的 IP 核只使用少量的查找表资源和寄存器资源，对原设计的影响很小。
- (4) 使 FPGA 不再是黑匣子。ChipScope Pro 可以方便地观测 FPGA 内部的所有信号，对 FPGA 内部逻辑调试非常方便。

ChipScope Pro 应用框图如图 7-28 所示，其中 ILA、ICON 是为了使用 ChipScope Pro 观察信号而插入的核。一般来说，ChipScope Pro 工作时需要在用户设计中实例化两种核：一是集成逻辑分析仪核（ILA core，Integrated Logic Analyzer core），提供触发和跟踪捕获的功能；二是集成控制器核（ICON core，Integrated Controller core），负责 ILA core 和边界扫描端口的通信，一个 ICON core 可以连接 1~15 个 ILA core。ChipScope Pro 工作时，ILA core 根据用户设置的触发条件捕获数据，然后在 ICON core 的控制下，通过边界扫描端口上传到计算机，最后用 ChipScope Pro Analyzer 显示出信号波形。

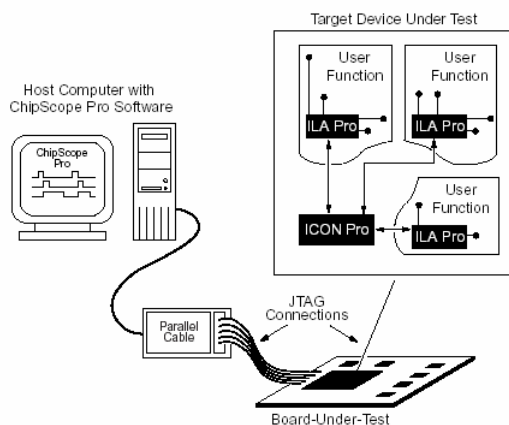


图7-28 ChipScope Pro 应用框图

ChipScope Pro 还集成了 ILA/ATC core（Integrated Logic Analyzer with Agilent Trace Core），这种 core 是 Xilinx 和 Agilent 合作开发的，它把要调试的 FPGA 和 Agilent E5904B Option 500 Trace Port Analyzer（Agilent TPA，需要单独购置）连接起来，可以提供更大的

捕获深度、更复杂的触发设置，并支持网络远程调试，功能十分强大。使用时数据由 Agilent TPA 捕获，与单独使用 ChipScope Pro 相比，可以减少用于调试的 FPGA 片内 RAM，使这种珍贵资源尽可能地用于设计中。Agilent TPA 可以为每个信号提供 2M samples（二百万个样本）的捕获深度，这是 ILA 的最大捕获深度（32k samples）的 60 倍，大的捕获深度对于查找原因和症状相隔时间较长的故障非常有用。

如果设计中用到 PowerPC 405 核或 MicroBlaze 软核（这两种核使用的外围总线都是 IBM CoreConnect On-Chip Peripheral Bus），可以实例化 IBA/OPB（Integrated Bus Analyzer for On-Chip Peripheral Bus）核用于观察总线上的信号。

ChipScope Pro 5.1i 以后的版本增加了 IBA/PLB（Integrated Bus Analyzer for Processor Local Bus）用于监测和调试 CoreConnect Processor Local Bus，还提供了 VIO（Virtual I/O Core），可以在任何 ChipScope Pro core 中加入虚拟输入，如 DIP 开关、按钮等，可以输出 Agilent Fast Binary Data Format (FBDF) 格式的数据文件，以便使用 Agilent 16700 逻辑分析仪进行调试。ChipScope Pro 5.2i 整合到 ISE Project Navigator 中，更加方便易用。ChipScope Pro 和 Virtex-II Pro 器件、ISE 软件结合可以提供最强大的设计和实时调试解决方案，缩短验证周期并降低项目的研发费用。

ChipScope Pro 工具箱包含 3 个工具：ChipScope Pro Core Generator、ChipScope Pro Core Inserter 和 ChipScope Pro Analyzer。ChipScope Core Generator 的作用是根据设定条件生成在线逻辑分析仪的 IP 核，包括 ICON core、ILA core、ILA/ATC core 和 IBA/OPB core 等，设计人员在原 HDL 代码中实例化这些核，然后进行布局布线、下载配置文件，就可以利用 ChipScope Pro Analyzer 设定触发条件、观察信号波形。ChipScope Pro Core Inserter 除了不能生成 IBA/OPB core 和 ILA/ATC core 以外，功能与 ChipScope Core Generator 类似，可以生成 ICON core 和 ILA core，但是它能自动完成在设计网表中插入这些核的工作，不用手工在 HDL 代码中实例化，在实际工作中用得最多，因此下面将详细介绍 ChipScope Pro Core Inserter 的使用方法。

7.3.2 ChipScope Pro Core Inserter

ChipScope Pro Core Inserter 的启动方法是直接在 Windows 环境下运行【开始】/【程序】/【ChipScope Pro 6.2i】/【ChipScope Pro Core Inserter】命令，ChipScope Pro Core Inserter 的用户界面如图 7-29 所示。

ChipScope Pro 的用户界面主要由标题栏、菜单栏、常用工具栏、项目浏览器、参数设置对话框和信息显示窗口组成，各主要部分功能如下：

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）菜单包含了常见文件操作等命令，其中【Refresh Netlist】命令用于更新网表，当输入的网表发生变化时，ChipScope Pro 会提示自动更新网表，设计者也可以使用【Refresh Netlist】命令自己更新网表。
- 编辑（Edit）菜单包含了创建新的 ILA 单元（New ILA Unit）、创建新的 ATC2

单元 (New ATC2 Unit)、创建新的 ILA/ATC 单元 (New ILA/ATC Unit)、删除单元 (Remove Unit) 和选项 (Preferences) 等命令。其中 ILA ATC (Agilent Trace Core) 单元是一种新型的 IP 核, 可以与安捷伦的 FPGA 端口跟踪分析仪 (Agilent FPGA Trace Port Analyzer) 结合, 提供比传统逻辑分析仪更强大的跟踪捕获和触发功能。

- 插入 (Insert) 菜单包含了【Insert Core】命令, ILA core 的各项参数设置完后可以利用这个命令把在线逻辑分析仪的网表插入到原设计的网表中。
- 帮助 (Help) 菜单包含了 ChipScope Pro 的版本信息。

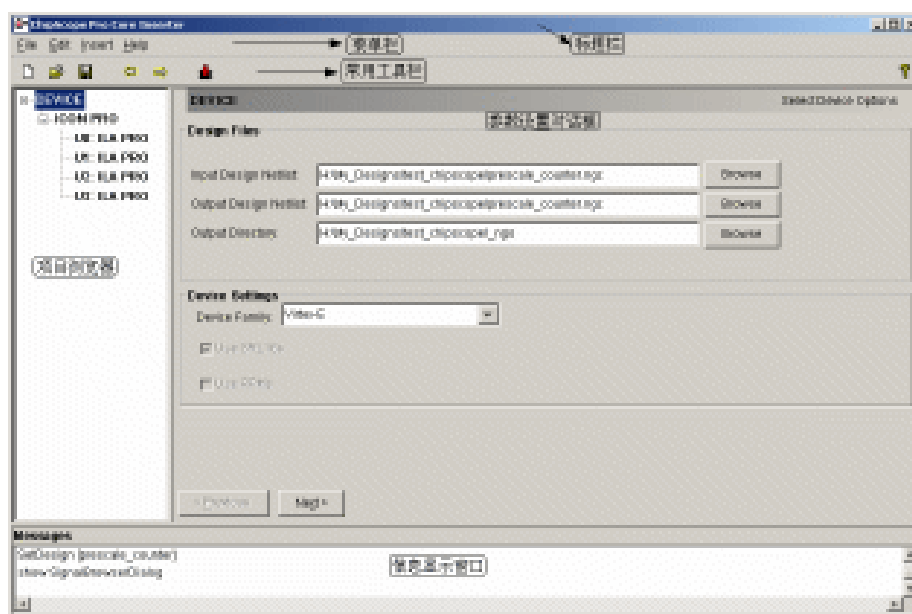


图7-29 ChipScope Pro Core Inserter 的用户界面

2. 项目浏览器

项目浏览器列出了插入到设计中的所有 ICON core 和 ILA core, 选中一个 ILA core 后, 可以在参数设置对话框查看或修改其参数。

3. 参数设置对话框

ICON core 和 ILA core 的参数在此对话框中设置。

4. 信息显示窗口

信息显示窗口用于显示 ChipScope Pro Core Inserter 的工作状态信息。

下面简要说明使用 ChipScope Pro Core Inserter 插入在线逻辑分析仪网表的步骤。首先在 ISE Navigator 中完成 Translate (翻译逻辑网表) 步骤, 然后启动 ChipScope Pro Core Inserter (参见图 7-29), 在【Input Design Netlist】文本框中输入设计网表的路径, 【Output Design Netlist】和【Output Directory】将会自动指定。在【Device Family】下拉列表框中选取设计所用的 FPGA 的系列, 单击 **Next >** 按钮, 弹出【Select Integrated Controller Options】对话框, 如图 7-30 所示。(与图不符, 请检查修改)

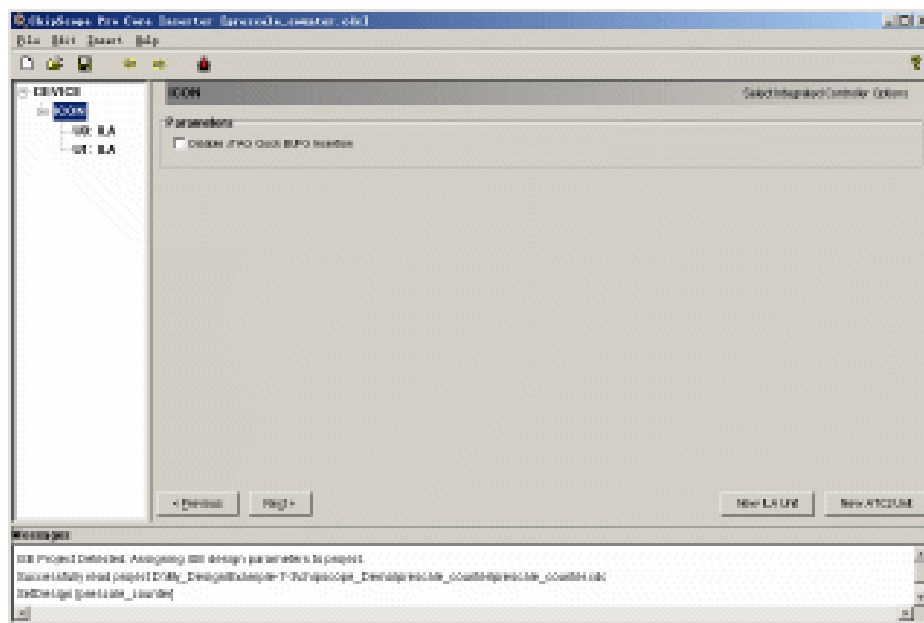


图 7-30 【Select Integrated Controller Options】对话框

图 7-30 所示的对话框用于指定是否禁止在 JTAG 时钟上插入 BUFG。如果选中此项，JTAG 时钟将使用普通布线资源，而不是全局时钟布线，这样会在 JTAG 时钟线上产生较大的布线延时差别（skew），破坏待分析信号之间的时序关系。因此，在全局时钟资源足够的情况下，应该尽量使 JTAG 时钟使用 BUFG 资源，即使由于全局时钟资源不够而不得不禁用 BUFG 时，也必须附加相应的约束，使 JTAG 时钟线上的延迟抖动尽量小。这里不选中此项，使用 BUFG 资源。单击 **Next >** 按钮，弹出【Select Integrated Logic Analyzer Options】对话框，如图 7-31 所示。（与图不符，请检查修改）

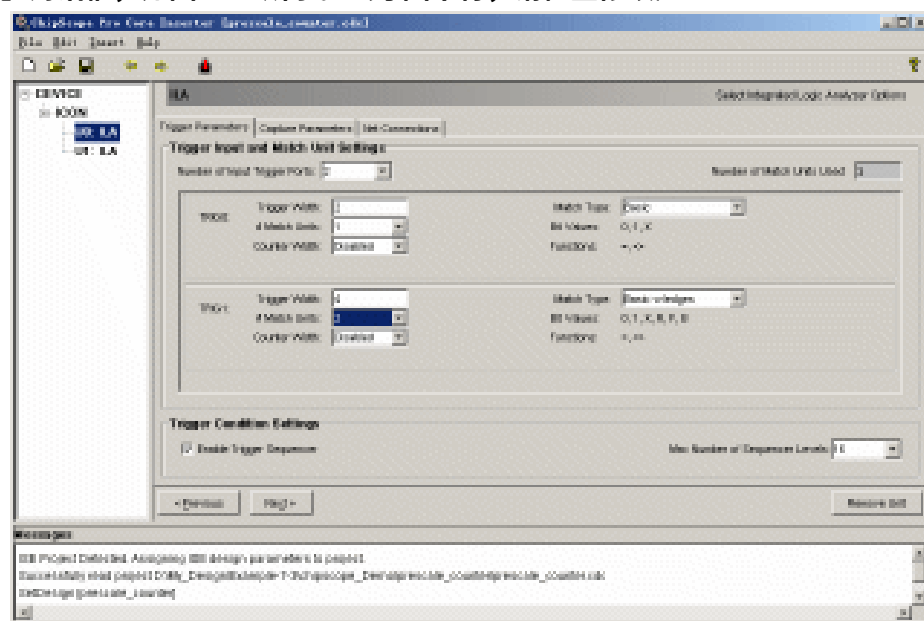


图 7-31 【Select Integrated Logic Analyzer Options】对话框

【Select Integrated Logic Analyzer Options】对话框包含 3 个选项卡，分别为【Trigger Parameters】（触发参数）、【Capture Parameters】（捕获参数）和【Net Connections】（网线连接）。

【Trigger Parameters】选项卡用于设置触发输入和触发条件判断单元。在【Number of Input Trigger Ports】（输入触发端口数）下拉列表框中可以选择 ILA core 输入触发端口的数目，每个 ILA core 最多可以有 16 个输入触发端口，但是各个 ILA core 触发端口包含的触发条件判断单元（Match Unit）数量之和不大于 16。各个触发端口的参数将在下方分别列出，包括触发宽度、触发条件判断单元的类型和数目等。

触发端口由一根或多根信号线组成，信号线的总数称为触发宽度。触发条件判断单元是一个比较器，它和触发端口相连，用于检测触发端口是否发生了特定事件。一个触发端口可以有 1~4 个触发条件判断单元，这些触发条件判断单元可以组合起来构成逻辑分析仪的触发条件，用于捕获数据。触发条件判断单元越多，占用的逻辑资源越多，因此在满足设定触发条件要求的情况下，应尽量减少触发条件判断单元的数目。触发条件判断单元的类型如表 7-2 所示。

表 7-2 触发条件判断单元的类型

类型	匹配功能	bit/Slice	说明
Basic	=, <>	8	用于数据信号的比较，这种类型在一个 Slice 中可以实现最多的比特数
Basic (w/trans)	=, <>, transition detection	4	用于控制信号的比较，可以检测跳变的发生
Extended	=, <>, >, >=, <, <=	2	用于比较地址和数据信号，可以进行数值大小的比较
Extended (w/trans)	=, <>, >, >=, <, <=, transition detection	2	用于比较地址和数据信号，可以比较数值大小和检测跳变的发生
Range	=, <>, >, >=, <, <=, in Range, not in Range	1	用于比较地址和数据信号，可以检测其数值是否在特定范围内
Range (w/trans)	=, <>, >, >=, <, <=, in Range, not in Range, transition detection	1	用于比较地址和数据信号，可以检测其数值是否在特定范围内，也可以检测跳变的发生

触发参数设置完毕后，单击  按钮，参数设置对话框切换为【Capture Parameters】选项卡，如图 7-32 所示。

在【Capture Parameters】选项卡中可以对存储深度、数据位宽、采样时刻等参数进行设置。ChipScope Pro 可能的最大存储深度为 16384，最大数据位宽为 256bit，实际的数据存储深度和位宽由 FPGA 内部剩余的 Block Ram 的数量决定，例如 Xilinx Virtex 系列 FPGA 的一个 Block Ram 为 4096bit，则采集数据的深度和宽度要满足：采集数据的深度×采集数据的宽度≤4096×剩余的 Block Ram 块数。

如果选中【Data Same As Trigger】选项，那么数据与触发信号相同，在【Trigger Ports Used As Data】表单中可以选取哪些触发端口作为数据。数据与触发信号相同是很常用的模式，因为用户可以捕获和采集触发逻辑分析仪的任何数据。在这种模式下，ILA Core 省略了数据输入端口，因此可以减小 CLB 和布线资源的占用，但是总的的数据宽度不能大于 256bit。不选中【Data Same As Trigger】选项时，数据和触发信号完全独立，当采样的数据

位宽小于触发宽度时，这种模式能减少采集的数据，节省 Block RAM 资源。
完成捕获参数的设置后，可以在选项卡的下方看到占用 Block RAM 的数量。

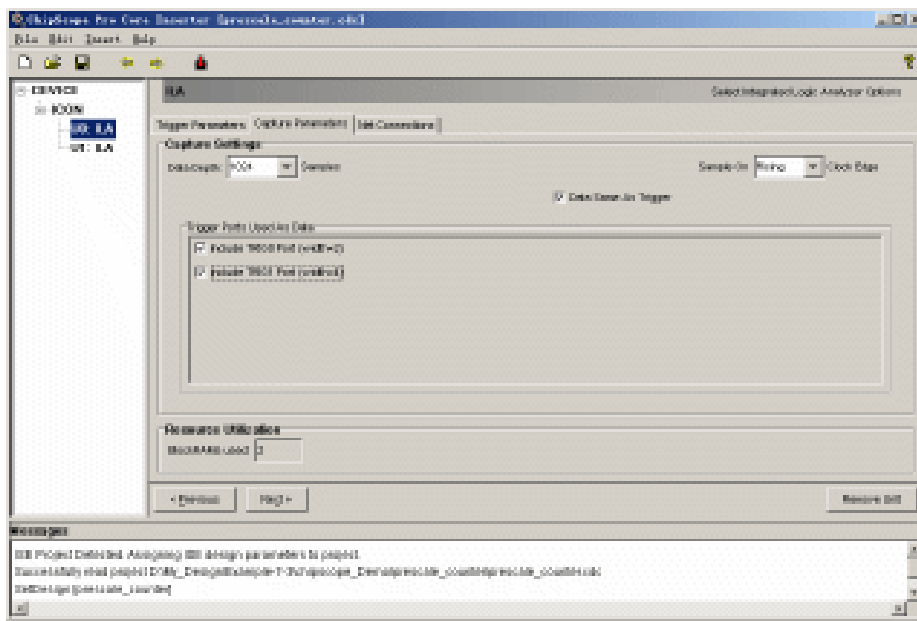


图7-32 【Capture Parameters】选项卡

捕获参数设置完毕后，单击 **Next >** 按钮，参数设置对话框切换为【Net Connections】选项卡，准备把在线逻辑分析仪的输入信号和设计中的网线连接起来，如图 7-33 所示。

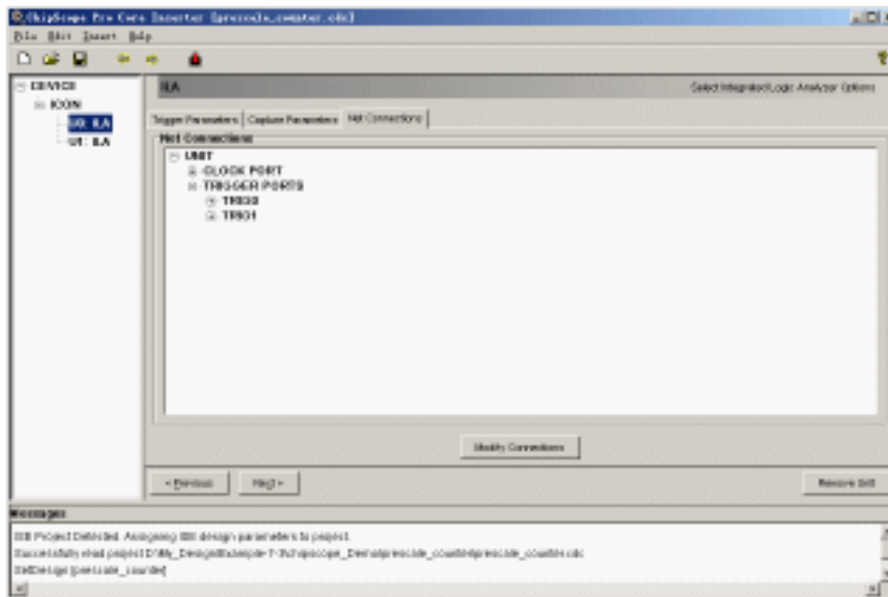


图7-33 准备连接在线逻辑分析仪和网线

单击 **Modify Connections** 按钮，弹出【Select Net】对话框，如图 7-34 所示。

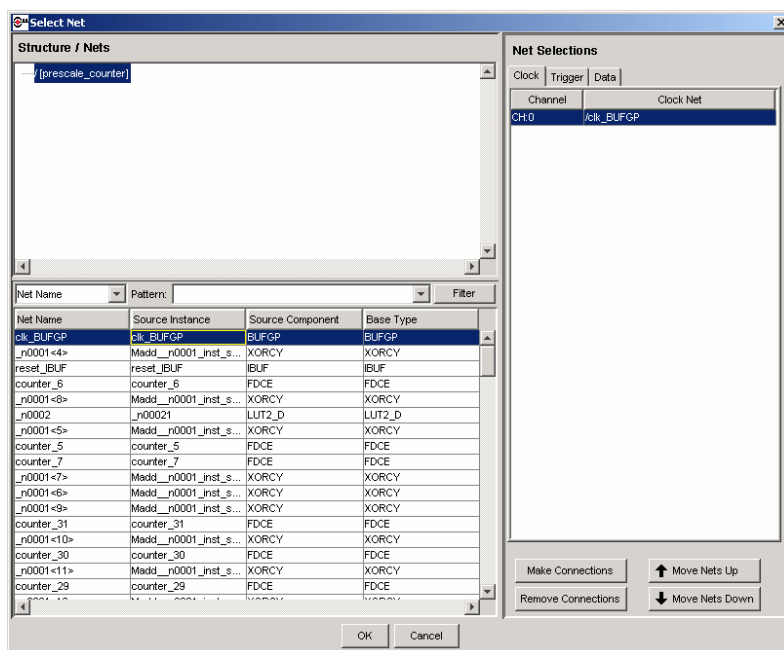


图7-34 连接在线逻辑分析仪和网线

利用【Select Net】对话框可以把在线逻辑分析仪的工作时钟、触发信号、数据信号与设计中的网线连接起来，方法是在对话框的右侧选中在线逻辑分析仪的信号，在对话框的左下方选中需要连接的网线，单击 **Make Connections** 按钮，即可完成一条信号线的连接。在线逻辑分析仪的所有信号都连接好之后，单击 **OK** 按钮，弹出【Select Integrated Logic Analyzer Options】对话框，如图 7-35 所示。（与图不符，请检查修改）

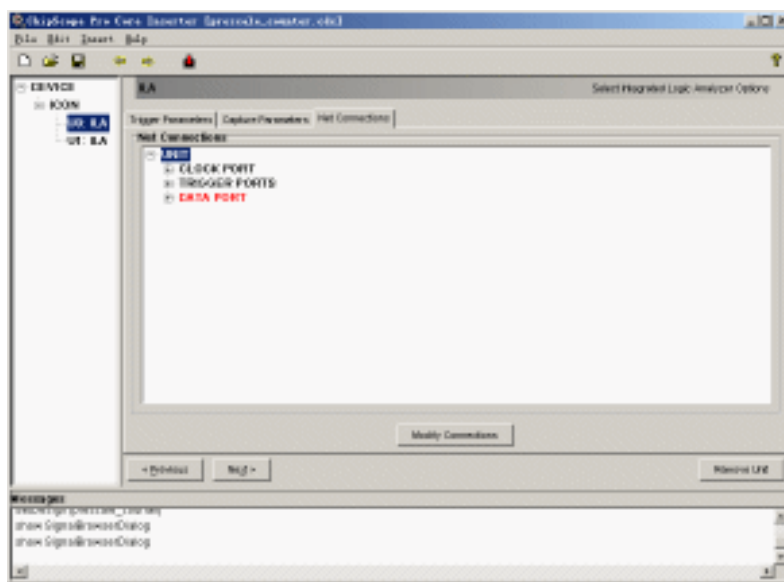


图7-35 【Select Integrated Logic Analyzer Options】对话框

在线逻辑分析仪包括 3 类端口，即时钟端口（CLOCK PORT）、触发端口（TRIGGER

PORTS) 和数据端口 (DATA PORT)，使用时应保证所有端口信号都已经连接，否则插入核时将提示错误信息。当端口中的信号全部连接时端口名字为黑色，否则为红色。

完成信号连接后，单击 **Insert >** 按钮，ChipScope Pro Core Inserter 提示“是否插入 core”，单击 **是(Y)** 按钮后，在线逻辑分析仪的网表将被插入到原来的设计网表中，然后在 ISE Navigator 中重新执行翻译逻辑网表步骤，设计完成布局布线并生成 BIT 文件后，就可以下载到 FPGA 中，利用 ChipScope Pro Core Analyzer 分析信号波形。

7.3.3 ChipScope Pro Analyzer

ChipScope Pro Analyzer (以下简称 Analyzer) 的启动方法是直接在 Windows 界面下运行 **【开始】 / 【程序】 / 【ChipScope Pro 6.2i】 / 【ChipScope Pro Analyzer】** 命令，另一种方法是设计综合实现后，在 Project Navigator 的 **【Processes for Source】** 中，双击 **【Analyze Design Using Chipscope】**，Analyzer 的用户界面如图 7-36 所示。



图7-36 ChipScope Pro Analyzer 的用户界面

Analyzer 的用户界面主要由菜单栏、常用工具栏、项目浏览器 (Project Tree)、信号浏览器 (Signal Browser)、主窗口 (Main Window) 和信息显示窗口 (Message Pane) 组成，下面简要介绍各部分的功能。

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)** 菜单包含了新建项目 (New Project)、打开项目 (Open Project)、保存项目 (Save Project)、项目另存为 (Save Project As)、引入 (Import) 和导出 (Export) 等命令，其中 Import 命令用于从文件中引入信号名，Export 命令用于导出捕获的数据，以便以后观察和处理。

- 视图 (View) 菜单包含了显示项目浏览器 (Project Tree)、显示信息显示窗口 (Messages) 两个命令。
- 边界扫描链 (JTAG Chain) 菜单包含了下载电缆相关的命令, 主要有连接 Xilinx 并行电缆 (Xilinx Parallel Cable)、安捷伦 E5904 电缆 (Agilent E5904 Cable)、关闭电缆 (Close Cable)、获取电缆信息 (Get Cable Information) 和打开自动核状态查询 (Auto Core Status Poll) 等。
- 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 (Window) 菜单包含了窗口排列的相关命令。
- 帮助 (Help) 菜单包含了 Analyzer 的版本信息。

2. 项目浏览器 (Project Tree)

项目浏览器在边界扫描链正确初始化之后会列出扫描链上的所有能识别的器件, 在配置下载完成后, 同时项目浏览器会自动更新, 显示 core 的数目, 并为每个 core 创建一个文件夹, 其中包含了 Trigger Setup、Waveform 和 Listing 等项目, 分别用于触发条件设置和观察信号波形。

3. 信号浏览器 (Signal Browser)

在项目浏览器中选中某一个 core 时, 信号浏览器会显示该 core 的所有信号。在信号浏览器中可以进行增加或删除视图中的信号、对信号进行重新命名、组合为总线 (方便观察) 等操作。

4. 主窗口 (Main Window)

主窗口用于显示 “Trigger Setup”、“Waveform”、“Listing” 和 “Bus Plot” 等窗口。

5. 信息显示窗口 (Message Pane)

信息显示窗口用于显示 Analyzer 的工作状态信息。

使用 Analyzer 观察信号主要分为配置器件、设置触发参数和观察波形等 3 个步骤, 下面简要说明。

🔑 使用 Analyzer 观察信号

1. 配置器件。

使用 Analyzer 观察信号时, 首先需要配置器件。启动 Analyzer, 在常用工具栏中初始化边界扫描链, 这时项目浏览器会列出边界扫描链上的器件, 如图 7-37 所示。Analyzer 能自动识别扫描链上的 Virtex/-E/-EM/-II/-II Pro、Spartan-II/-IIE、Spartan-XL、9500/XL/XV、4000XL/XLA、18V00 PROM、CoolRunner、CoolRunner-II 和 System ACE 等器件。

然后利用 【Device】 / 【Configure】 菜单进行器件的配置, 如图 7-38 所示。

在图 7-38 中, 选中 【Device】 / 【Configure】 / 【Device 5】 命令后, Analyzer 会弹出如图 7-39 所示的对话框, 让设计者选择要下载的配置文件。

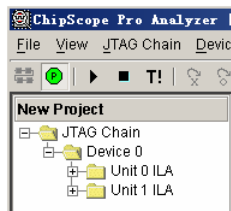


图 7-37 边界扫描链上的器件

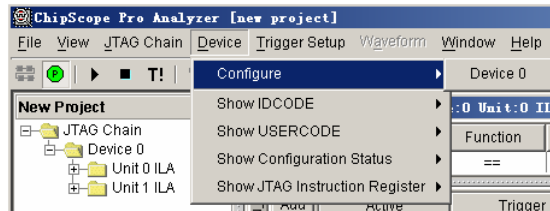


图 7-38 器件配置

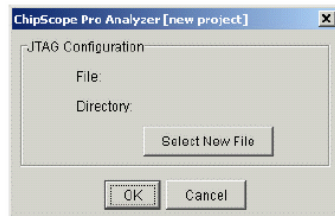


图7-39 选择下载配置文件对话框

选择 BIT 配置文件，单击 **OK** 按钮，进行器件配置。下载 BIT 文件过程中，FPGA 的配置状态信息显示在 Analyzer 窗口的右下角，配置正确时会显示“DONE”状态，同时项目浏览器会自动更新，显示 core 的数目，并为每个 core 创建一个文件夹，其中包含了 Trigger Setup、Waveform 和 Listing 等项目。配置不正确时将会弹出对话框，说明遇到的问题。

需要注意的是：ChipScope Pro 采用 JTAG 方式观测 FPGA/CPLD 内部信号，这就要求在生成下载文件（BitGen）时，启动时钟（Start-Up Clock）选项必须设置为 JTAG 时钟方式（JTAG Clock）。如果“Start-Up Clock”选项设置为“CCLK”方式，ChipScope Pro Analyzer 将不能正确配置器件。

2. 设置触发参数。

为了设置 ChipScope Pro core 的触发条件，需要在【Window】/【New Unit Windows】菜单选中某一个 ChipScope Pro core，然后单击鼠标右键，从中可以选择打开【Open Waveform】、【Open Trigger Setup】、【Open Listing】或【Open Bus Plot】窗口，如图 7-40 所示。为了设置触发条件，选中【Trigger Setup】选项，打开触发条件设置窗口。这个操作也可以在【Project Tree】窗口中双击相应器件和 ILA core 下面的【Trigger Setup】项实现。

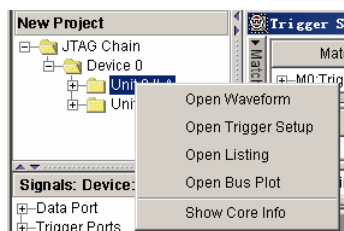


图7-40 打开触发参数设置对话框

Trigger Setup - Device:2 Unit:0 ILA					
Match	Match Unit	Function	Value	Ra...	Counter
☒	M0.TriggerPort0	==	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Bin	at least once
☒	M1.TriggerPort1	==	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Bin	at least once
☒	M2.TriggerPort2	==	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Bin	at least once
☒	M3.TriggerPort3	==	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Bin	at least once

Trigger			
Add	Active	Trigger Condition Name	Trigger Condition Equation
Del	☒	TriggerCondition1	M0

Cap					
Type:	Window	Windows:	1	Depth:	512
				Position:	0
☐ Timestamp					

图7-41 展开后的触发参数设置窗口

在“Capture Settings”窗口中可以设置窗口的数目和窗口中触发事件的位置，这里说的窗口是指包含由一个事件触发的连续的多个样本。【Type】列表框定义了所用窗口的类型，如果选择“Window”类型，那么窗口中的样本数必须为 2 的若干次方，触发可以在窗口的任何位置，这时可以在【Windows】文本框中确定窗口的数目，数值从 1 到捕获缓冲区的深度之间选取。在【Depth】列表框中选择每个捕获窗口的深度，可供选择的值由窗口数目决定，触发的位置由【Position】文本框决定，有效取值范围在 1 到捕获缓冲区深度减 1 之间。如果在【Type】列表框中选择“N Samples”类型，缓冲 buffer 中将包括多个窗口，每个窗口包含 N 个样本，这时只能在窗口的第一个样本触发，每次触发捕获的样本数由【Samples Per Trigger】“Samples Per Trigger”文本框决定，数值范围在 1 到捕获缓冲区深度之间。

“Match Functions”窗口用于设置触发条件判断单元的触发条件函数，多个函数的逻辑组合构成 ChipScope Pro core 完整的触发条件。在“Match Functions”设置窗口中，“Match Unit”用于指定该函数用于哪个触发条件判断单元，“Function”用于选择比较的类型，列表框中只列出了相应触发条件判断单元所支持的比较类型，这由触发条件判断单元本身的类型决定，读者可以参考 7.3.2 小节表 7-2 中的说明。“Value”用于设置进行比较的具体数值，“Radix”决定该数值是二进制或是十六进制。“Counter”规定了触发条件函数所定义事件发生多少次之后才认为满足触发条件函数，如果选择了“at least n times”，那么事件连续或不连续发生 n 次后，满足触发条件函数。如果选择了“for n cycles”，那么事件只有连续发生 n 次才满足触发条件函数。

在“Trigger Conditions”窗口中可以把一个或多个触发条件判断单元组合起来，构成最终的触发条件。触发条件是一个或多个触发条件函数的逻辑组合，

ChipScope Pro core 根据触发条件判断是否捕捉数据。在 Analyzer 中可以定义多个触发条件。在“Trigger Conditions”设置窗口中，使用“Add”可以加入新的触发条件，选中一个触发条件后选择“Del”可以删除它。“Active”域决定了当前有效的触发条件，“Trigger Condition Name”允许使用者为触发条件起名字，以便记忆，“Trigger Condition Equation”（触发条件等式）显示了触发条件的构成，默认值为来自不同触发条件判断单元的触发条件函数的逻辑与，如果希望改变设置，可以点击该域，这时会弹出【Trigger Condition】对话框，如图 7-42 所示。

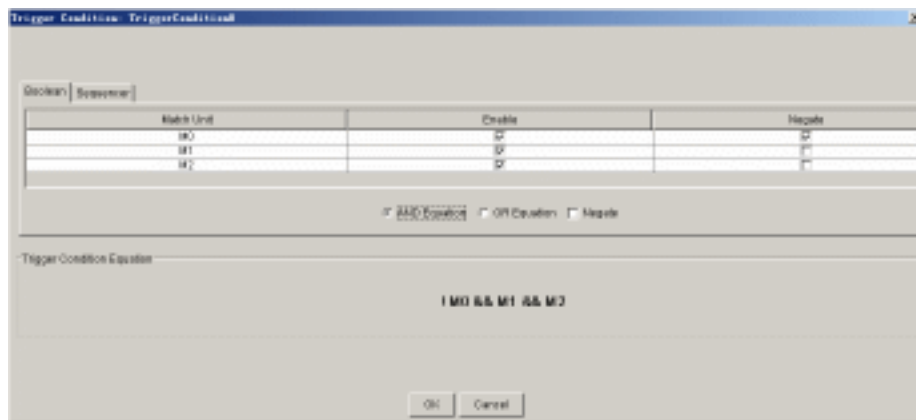


图7-42 设置触发条件等式

设计者可以根据需要进行设置，设置好的触发条件等式显示在对话框的下方。触发条件可以在运行时更改，更改后不需要重新综合实现。选择【Trigger Setup】/【Save Trigger Setup】命令可以把触发设置保存到后缀为 ctj 的文件中，方便下次使用。选择【Trigger Setup】/【Read Trigger Setup】命令可以从 ctj 文件中读入触发设置。

触发条件设置好之后，选中【Trigger Setup】/【Run】命令使 ChipScope Pro core 进入 arm（伺机捕获）状态，当触发条件满足时，core 根据捕获设置捕获数据，并在缓冲区填满后停止捕获数据，然后上传数据，由 Waveform 和 Listing 窗口显示出来。ChipScope Pro core 在触发并采样数据后退出 arm 状态，通过选中【Trigger Setup】/【Stop Acquisition】命令也可以使 core 退出 arm 状态，如果之前触发条件曾经满足过，那么 Analyzer 会上传并显示捕获的数据。

3. 观察信号波形。

为了观察特定 core 的信号波形，需要打开波形观察窗口，方法是在【Window】/【New Unit Windows】菜单中选择某一个 ChipScope Pro core，这时会弹出【New Window Selection】对话框，选择打开“Waveform”窗口。这个操作也可以在“Project Tree”窗口中双击相应器件和 ILA core 下面的“Waveform”项实现。波形观察窗口把采样缓冲区的数据显示出来，显示的方式和很多仿真器或逻辑分析仪类似。包含 ChipScope Pro core 的 BIT 文件下

载到 FPGA/CPLD 并设置好触发条件后,选中【Trigger Setup】/【Run】命令,ChipScope Pro core 触发后会把采样到的数据显示在波形观察窗口中,如图 7-43 所示。

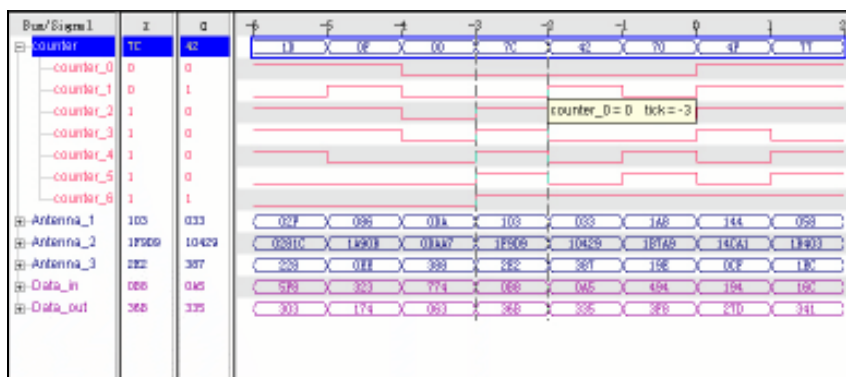


图7-43 在波形观察窗口观察采集的数据波形

设计者可以在波形观察窗口中进行信号波形重新排序、光标测量、简单波形变换等操作,这些操作都比较简单,与实际使用逻辑分析仪非常相似,这里不再赘述。

7.4 小结

本章详细介绍了有关 XPower、iMPACT 和 ChipScope Pro 的基础知识以及这 3 个工具的使用方法。通过本章的学习,读者应该掌握这些工具的功能、在 FPGA/CPLD 设计流程中的位置以及工具的基本使用方法。利用 XPower 准确分析功耗的关键是正确设置信号的翻转率,而 ModelSim 生成的 VCD 文件包含了信号的翻转率的信息,因此进行功耗分析时要尽量使用 VCD 文件。iMPACT 是 Xilinx 提供的配置工具,可以配置 FPGA、CPLD 和 PROM,它支持多种配置模式,其中边界扫描模式最为常用,通过 JTAG 口,iMPACT 可以向 FPGA 下载 BIT 文件,向 CPLD 下载 JEDEC 文件,向 PROM 下载 EXO、MCS 等格式的 PROM 文件。ChipScope Pro 用在 FPGA 调试阶段,它具有传统逻辑分析仪的功能,可以观察 FPGA 内部的任何信号,触发条件、数据宽度和深度等的设置也非常方便,而且价格便宜,在实际工作中应用得很广泛。

7.5 问题与思考

1. 在 XPower 中设置信号的翻转率时,为什么 100%的翻转率对应时钟频率的一半?
2. FPGA 有哪几种配置模式,各种配置模式的特点是什么?
3. 使用 ChipScope Pro 时,需要插入哪几种核,它们的作用分别是什么?

第8章 模块化与增量式设计方法

本章重点讨论模块化设计方法和增量式设计方法的基本概念、设计流程与操作方法。虽然 Modular Design 和 Incremental Design 都要用到区域分组、位置约束、Floorplan 等设计手段，但是两者的核心思想却不同：Modular Design 的核心思想体现的是团队并行工作、协同设计；Incremental Design 强调的是在微小改动情况下节约综合实现时间，确保继承以往设计成果。

本章主要内容如下：

- 模块化设计方法的基本概念；
- 模块化设计方法的设计流程；
- 模块化设计方法的注意事项；
- 模块化设计方法的设计实例；
- 增量式设计方法的基本概念；
- 增量式设计方法的设计流程；
- 增量式设计方法的设计实例。

8.1 模块化设计方法的基本概念

随着可编程技术的发展，FPGA/CPLD 被广泛应用于电子设计的各个领域。越来越多复杂系统的核心电路利用 FPGA 设计完成，这些复杂系统经常需要使用百万门以上的大规模 FPGA 来设计。另一方面，为了对市场需求做出最迅速的反映，就要求这些电子产品的设计周期尽量缩短，只有以第一时间推出成熟稳定的产品，才能获得更大的市场份额。于是一方面需要百万门以上的大规模 FPGA 以满足设计需要，另一方面需要在最短的时间内高质量地完成设计以满足市场需求，这两者出现了矛盾。

解决这个矛盾的出路之一是投入更多的人力，进行并行工作、协同设计。并行工作、协同设计在一般的工业生产中比较容易实现，例如生产一部汽车，可以在不同车间并行生产出汽车的各个配件，然后统一将这些配件组装起来即可。如果每个零件的制造都能满足图纸的公差标准，那么每个零部件的制造的关联就不是那么紧密，完全可以并行进行生产制造。使制造每个零部件之间的联系尽量小，然后运用并行、协同的工作方式相对独立地生产每个零部件，最后将这些零部件组合起来的工作方法就是一种模块化设计方法。

将这种模块化设计思路运用于 FPGA/CPLD 设计，将大规模复杂系统按照一定规则划分成若干模块，然后对每个模块进行设计输入、综合，并将实现结果约束在预先设置好的区域内，最后将所有模块的实现结果有机地组织起来，就能完成整个系统的设计。

Xilinx ISE 系列中的模块化设计方法叫做“Modular Design”。在 ISE 5.2 以前的版本中，Modular Design 是需要单独付费购买的工具包，在 ISE 5.2 及其升级版本中，Modular Design 已经直接作为标准工具内嵌到 ISE 软件中，安装 ISE 后即可直接使用，大大提高了 Modular Design 的用户数量，提高了 Xilinx Modular Design 工具的知名度。Xilinx Modular Design 的显著优势在于下面 3 个方面：

- 团队式并行工作、协同设计，即所有设计小组成员可以在最大程度上互不干扰地设计自己的子模块，从而加速了项目开发进度；
- 每个子模块可以有独立的优化目标。便于灵活使用综合和实现工具以达到更好的优化结果。
- 调试、更改某个有缺陷的子模块时，并不会影响到其他模块的实现结果，从而保证了设计的稳定性与可靠性。

Modular Design 支持的器件族有 Virtex-II Pro、Virtex/-II/-E 和 Spartan-II/-IIE/III。

8.2 模块化设计方法的设计流程

Modular Design 设计方法的设计流程也遵循第 1 章讲述的 FPGA/CPLD 设计流程。只是在设计输入、综合和实现等 3 个主要操作步骤上有一些特殊的要求和操作方法。图 8-1 所示为 Modular Design 的设计流程。

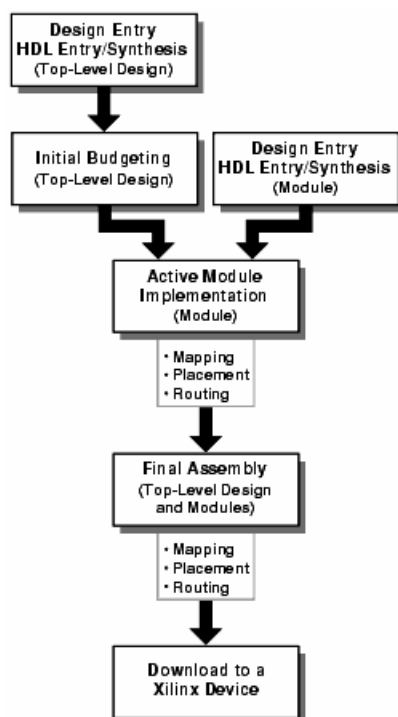


图8-1 Modular Design 设计流程

Modular Design 设计流程大致可以分为以下步骤：

1. Modular Design 的设计输入与综合步骤

Modular Design 设计输入与综合主要完成以下两个方面的设计：

(1) 顶层模块的设计。

项目管理者需要完成顶层模块的设计输入与综合，为进行 Modular Design 实现阶段的第一步——初始预算阶段（Initial Budgeting Phase）做准备。

(2) 子模块的设计。

每个项目成员相对独立地并行完成各自子模块的设计输入和综合，为进行 Modular Design 实现阶段的第二步——子模块的激活模式实现（Active Module Implementation）做准备。

2. Modular Design 的实现步骤

Modular Design 的实现步骤是整个 Modular Design 设计流程中最重要、最特殊的，它包括以下 3 个步骤：

(1) 初始预算（Initial Budgeting）。

在该阶段，项目管理者对顶层模块完成顶层约束。顶层约束包括对整个设计的全局区域约束、对每个子模块的规模和区域的约束、对每个模块的输入/输出约束、对整个设计的时序约束等内容。仅需要对设计的顶层模块进行初始预算估计，对子模块则没有必要。

(2) 子模块的激活模式实现（Active Module Implementation）。

在该阶段，每个项目成员并行完成各自子模块的实现。

(3) 模块的最后合并（Final Assembly）。

在该阶段，项目管理者将设计的顶层和子模块的激活模式实现结果有机地组织起来，完成整个设计的实现步骤。

在进行大规模复杂系统的设计时，采用 Modular Design 设计方法能在最大程度上发挥项目组所有工程师的作用，提高设计效率，缩短项目开发周期。但是 Modular Design 设计方法对项目管理者提出了更高的要求：项目管理者应该合理地划分模块，对项目进度做出很好的权衡，使所有子模块设计任务并行完成，从而不耽误模块的最后合并（Final Assembly）阶段的工作进度。

Modular Design 模块划分的基本原则为：子模块功能相对独立，模块内部联系尽量紧密，而模块间的连接尽量简单。所以对于那些难以满足模块划分准则的具有强内部关联的复杂设计，并不适合采用 Modular Design 设计方法。

8.2.1 Modular Design 的设计输入与综合步骤

上节已经介绍过在设计输入与综合步骤，项目管理者要完成顶层模块的设计输入与综合，而其他项目成员应该并行完成各自子模块的设计输入和综合。本节我们较为详细地讨论这两个层次上的设计输入与综合的相关问题。

Modular Design 设计方法更适用于大规模复杂系统的设计，而大规模系统复杂设计一般常用的设计输入方式为 HDL 语言设计输入方式。顶层模块描述了设计的全局逻辑，主要包括设计的输入/输出，所有子模块的黑盒子（Black Box）声明，以及子模块之间的连接关系。所谓黑盒子声明是指在顶层模块中仅仅对子模块进行端口描述与信号属性声明，并不包

含任何实际逻辑与时序关系的描述。完成顶层模块的设计输入和综合是开展 Modular Design 实现步骤第一阶段——初始预算（Initial Budgeting）的必要条件，初始预算估计的输入就是顶层模块的综合网表。

子模块的设计通常也使用 HDL 语言输入方式，项目成员根据规划设计各自子模块的逻辑实体，并综合所设计的子模块。由于子模块的输入/输出并不是整个设计的外部接口，所以在综合过程中应该禁止子模块插入 I/O 端口，而仅在综合顶层模块时才插入 I/O 端口。在明确了子模块间的接口以及子模块与顶层模块的接口后，子模块的设计应该并行进行，甚至可以与顶层模块的设计同步开展。在子模块的综合完成后方可开展 Modular Design 实现步骤第二阶段——子模块的激活模式实现（Active Module Implementation）。

图 8-2 所示为 Modular Design 设计输入和综合步骤的示意流程图。顶层模块与子模块使用 HDL 语言进行设计输入的方法请参考本书 2.2 节的描述，顶层模块与子模块的综合可以使用本书第 4 章介绍的任何一种综合工具，具体使用方法请参考第 4 章论述。

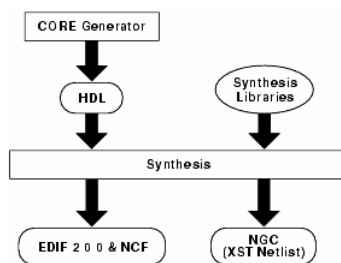


图8-2 Modular Design 设计输入和综合步骤的示意流程图

8.2.2 Modular Design 的实现步骤

前面介绍过 Modular Design 的实现步骤分为：初始预算（Initial Budgeting）、子模块的激活模式实现（Active Module Implementation）、模块的最后合并（Final Assembly）等 3 个阶段。下面依次详细介绍。

1. 初始预算（Initial Budgeting）

本阶段是实现步骤的第一步，对整个 Modular Design 起着指导性的作用。在初始预算阶段，项目管理者需要为设计的整体进行位置布局，只有布局合理，才能够在最大程度上体现 Modular Design 的优势；反之，如果因布局不合理而在较后的阶段需要再次进行 Initial Budgeting，则需要对整个实现步骤全面返工。在 Initial Budgeting 阶段的主要任务为：

- 对设计进行全局区域布局；
- 约束每个子模块的规模和区域；
- 定位每个模块的输入/输出；
- 对设计进行全局时序约束。

本阶段主要使用的 Xilinx 工具是“NGDBuild”。NGDBuild 是 Xilinx 实现步骤的重要内部工具，其基本功能为将综合生成的逻辑网表 EDIF 或 NGC 文件翻译为 Xilinx 底层硬件原语网表 NGD 文件。NGDBuild 的输入是 EDIF 或 NGC 网表，这两种文件都是综合步骤产生

的，它们的本质是用与、或、非门，触发器，RAM 等基本逻辑单元描述的设计的逻辑结构与连接，所以这两种网表通常被称为逻辑网表。NGDBuild 的输出是 NGD 网表，它的本质是使用 Xilinx 内部通用数据库（NGD，Native Generic Database）硬件原语（Primitive）描述的设计的结构连接，是逻辑网表向 Xilinx 内部硬件原语翻译的结果。NGDBuild 就是完成逻辑网表向 Xilinx 硬件原语网表翻译的工具，对应 ISE 实现过程 3 个步骤——翻译（Translate）、映射（Map）和布局布线（Place & Route）的第一步。

在本阶段使用 NGDBuild 工具生成顶层设计的 NGD 文件，该文件中所有子模块的实例（Instance）被表述为未扩展的模块，虽然 NGD 文件不能被映射到芯片上，但是可以使用约束编辑器（Constraints Editor）、引脚与区域约束编辑器（PACE）、Xilinx 布局规划器（Floorplanner）等工具打开。使用 Constraints Editor 可以约束模块的时序，使用 PACE 可以锁定引脚位置，使用 Floorplanner 可以对每个子模块进行区域约束。

为顶层逻辑和子模块进行区域约束，明确它们在目标芯片上的区域位置是至关重要的，只有这样每个子模块的实现结果才不会发生冲突，在 Final Assembly 阶段才能将所有子模块的实现结果有序地组织在目标芯片上。区域约束的主要内容包含以下几个方面。

(1) 顶层 I/O 端口约束。

一般来说，FPGA/CPLD 设计的 I/O 端口在项目规划时就应该完成分配，在 PCB 布板时就被设定。使用 PACE 或 Floorplanner 可以完成 I/O 端口的位置锁定与属性约束。

(2) 顶层逻辑的区域约束。

一般来说，设计的顶层的作用主要是连接各个子模块，而子模块才是功能实体，所以顶层模块一般仅仅包含一些用全局时钟资源、三态缓冲、触发器和查找表描述的简单逻辑。对缓冲单元（BUFT）进行位置约束时应该遵循以下规则：

- 如果多个 BUFT 驱动某一相同的网线，这些 BUFT 应该被安置到同一行。
- 如果 BUFT 驱动不同的网线，BUFT 应该被安置在不同的行。
- 如果设计中有较多三态网线被三态 BUFT 所驱动，那么每根三态网线都应该被一一对应的三态 BUFT 所驱动。



从第 1 章“FPGA/CPLD 的基本原理”可知，FPGA 的内部 CLB 等器件资源是分行（Row）、列（Column）排列的。

(3) 子模块的区域约束。

在对子模块进行区域约束时，首先应该对每个子模块估算规模，然后在芯片上规划出足够实现该子模块的矩形区域。子模块对应的矩形区域不易太大也不易过小，如果太大势必造成芯片面积的浪费，如果太小则有可能影响布线效果，一般经验值为资源使用率不高于 80%。接着在芯片上全局性地规划所有子模块对应的矩形区域，一般来说两个子模块之间应该留有空白面积以布置子模块的 I/O 端口。子模块的区域约束的详细方法请参考本书 6.2 节“布局规划器（Floorplanner）”中的论述。

(4) “伪逻辑”（Pseudo Logic）的区域约束。

“伪逻辑”是本书首次引入的概念。前面介绍过，在 Modular Design 设计方法中顶层仅包含子模块的端口声明，而不包含子模块的功能实体，所有子模块在顶层中都是黑盒子！

因此，两个模块之间无法直接连接，而是通过所谓的“伪逻辑”连接。“伪逻辑”分为“伪驱动逻辑（pseudo drive）”和“伪装载逻辑（pseudo load）”。因为模块的功能实体未知，模块间的实际装载和驱动单元也就不得而知，所以不得不使用伪驱动逻辑和伪装载逻辑作为模块的暂时性驱动和装载单元。图 8-3 示意了“伪逻辑”与模块的连接关系。

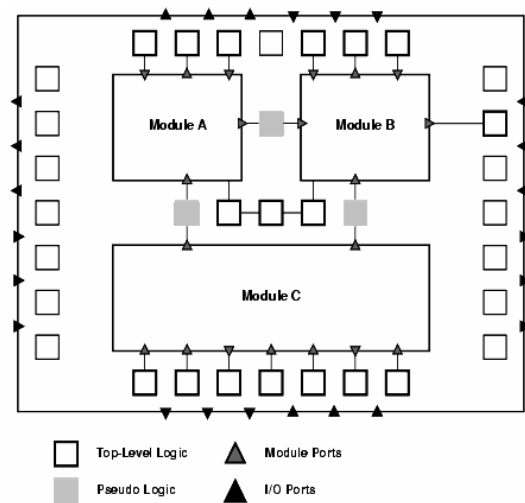


图8-3 “伪逻辑”与模块连接示意图

图 8-4 所示为更详细的“伪逻辑”示意图。未知功能实体的模块 A 的输出端口与伪装载逻辑相连接，未知功能实体的模块 B 的输入逻辑与伪驱动逻辑相连接。之所以称这些连接逻辑为“伪逻辑”是因为它们仅仅是暂时性的被插入，以帮助连接子模块，而在实现的 Final Assembly 阶段，这些“伪逻辑”将被删除，取而代之的是模块之间的实际连接。

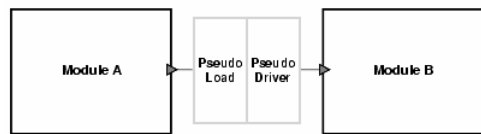


图8-4 详细的“伪逻辑”示意图

图 8-5 所示为初始预算阶段流程图。

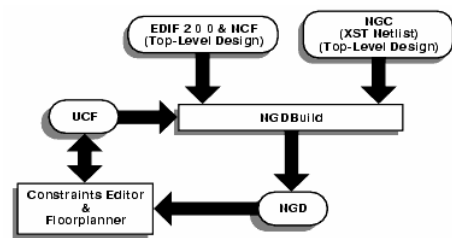


图8-5 初始预算（Initial Budgeting）阶段流程图

2. 子模块的激活模式实现（Active Module Implementation）

在本阶段，项目成员使用自己子模块的综合网表与初始预算阶段产生的顶层模块的实现

结果，对顶层模块中本子模块单独地进行扩展性实现，也就是说，子模块嵌入到顶层模块的实现结果中去。特别需要注意的是，每个子模块的激活模式实现过程必须单独进行，并且使用独立的实现目录，而不能直接在顶层模块的实现目录中进行。关于 Modular Design 的目录管理，在后面章节会有较为详细的介绍。

Active Module Implementation 的具体方式为：在激活（active）模式下运行 NGDBuild 工具，NGDBuild 读取顶层设计、顶层的用户约束文件（UCF）和当前子模块的逻辑网表（综合结果）作为输入，生成 NGD 文件作为输出，该 NGD 文件中仅有当前子模块被“激活”，也就是说仅有当前子模块被填充入功能实体，而其他子模块仍然是类似黑盒子的结构框架。在“激活”模式下，可以使用约束编辑器为当前子模块附加时序约束。然后根据新的用户约束文件，对所生成的 NGD 文件进行映射、布局布线。

如果某个子模块完成了布局布线，而且最终实现结果满足时序要求，则完成了对该子模块的 Active Module Implementation 步骤，可以将子模块提交给项目管理者了。这时我们称这个子模块为物理实现模块（PIM, Physically Implemented Module）。然后调用 pimcreate 命令，自动将 PIM 结果 NGO、NCD 和 NGM 等文件重命名并拷贝到最后合并目录，以供整个设计的最后合并使用。

图 8-6 所示为子模块的激活模式实现阶段流程图。

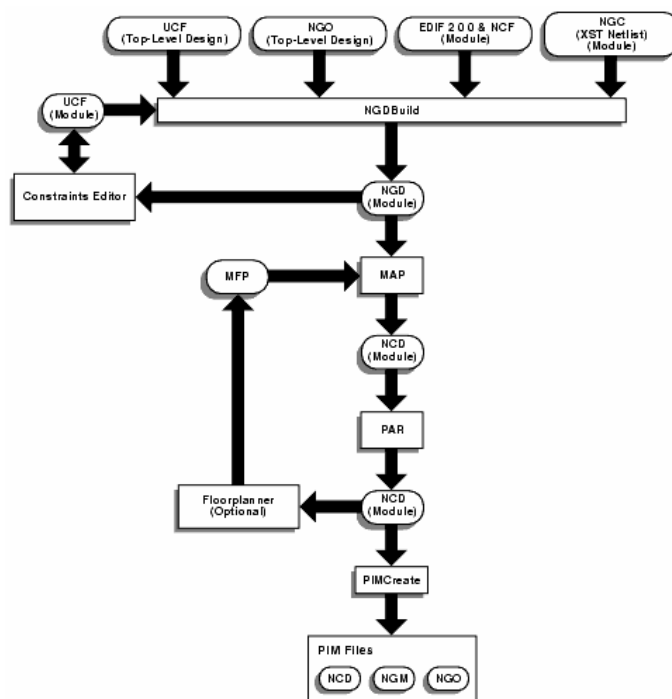


图8-6 子模块的激活模式实现（Active Module Implementation）阶段流程图

3. 模块的最后合并（Final Assembly）

在此阶段，项目管理者将所有子模块的激活模式实现结果和顶层的实现结果有机地组织起来，完成整个设计的实现。在 Final Assembly 模式下运行 NGDBuild 工具，NGDBuild 读取顶层设计的 NGO 文件、顶层的用户约束文件（UCF）和所有物理实现模块（PIM）结果

文件作为输入，对整个设计顶层进行全面的实体扩展，接着对合并后的设计进行映射、布局布线等操作，完成整个设计的完整实现步骤。在布局布线过程中，布线器读取 PIM 的布局布线信息，继承所有子模块的激活模式实现成果，从而满足所有子模块的区域位置与时序约束条件。

图 8-7 所示为模块的最后合并阶段流程图。

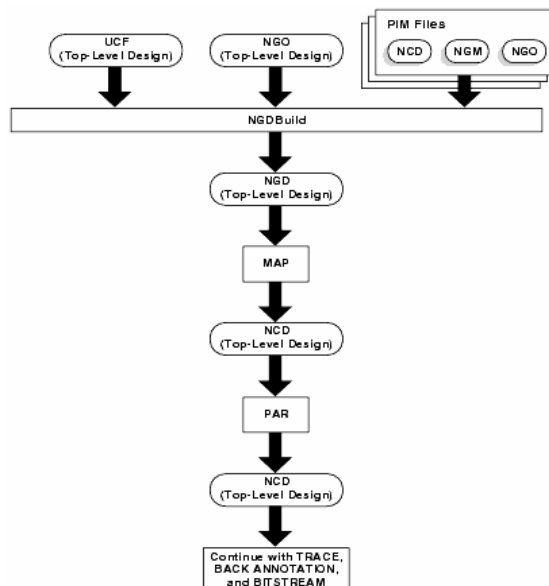


图8-7 模块的最后合并（Final Assembly）阶段流程图

8.3 模块化设计方法的注意事项

本节讨论 Modular Design 的注意事项与设计技巧，包括 Modular Design 的应用场合，目录管理，面积与时序约束，设计效能，实现报告的观察，使用 XFLOW 工具自动完成 Modular Design 流程等内容。

8.3.1 Modular Design 的应用场合

只有合理地使用 Modular Design，才能发挥其最佳效能。Modular Design 的合理应用场合应该具备以下几个特点：

- 大型规模复杂系统设计、高性能指标系统设计。只有大型复杂系统和高性能指标系统才能最大程度地发挥 Modular Design 的优势。关于 Modular Design 的设计效能的讨论详见下一小节。
- 设计本身可以方便地划分为互联很弱的逻辑模块。为了有效地进行模块的区域约束和时序约束，Modular Design 模块划分的基本原则是：子模块功能相对独立，模块内部联系尽量紧密，而模块间的连接尽量简单。对于那些难以满足模块划分准则的具有强内部关联的复杂设计，并不适合采用 Modular Design

设计方法。

- Modular Design 适合采用大型复杂系统的常用设计方法——自顶向下（Top-Down）设计方法。Modular Design 的设计流程很自然地支持了 Top-Down 设计方法。
- Modular Design 适合采用团队式（Team Based）设计。这是 Modular Design 设计方法的初衷。

8.3.2 Modular Design 的设计效能

本小节我们探讨一下 Modular Design 的性价比。

Modular Design 是 Xilinx FPGA 的一种高级设计方法，它需要使用到 Xilinx FPGA 的区域约束、时序约束等多种约束概念，并涉及 Floorplanner、Constraints Editor、PACE 和 Xilinx 底层实现命令等多种设计工具，其设计步骤与实现方法都是相当复杂，需要耗费一定的工作量。对于小规模或者一般性设计，使用 Modular Design 不仅仅是“大材小用”的问题，如果在设计过程中不能很好地掌握 Modular Design 的设计技巧，比如说在 Initial budgeting 的整体规划不恰当，反而会适得其反，影响设计效率、恶化实现结果！而对于大型复杂设计，由于使用的 FPGA 资源种类繁多，时序要求也相对复杂，如果采用传统设计方式，只能单线工作，无法提示设计效率，将需要相对的开发周期，影响产品的反应时间。而使用 Modular Design，虽然在初期规划时难度较大，但是并行设计，协同工作将大大提高设计效能，这也是 Modular Design 的初衷。在大规模设计中应用 Modular Design 设计方法，更改某个子模块并不影响其他子模块的 PIM 结果，只需将该子模块重新综合、激活模式实现，然后直接继承所有其他子模块的 PIM 结果，进行最后合并，就能完成整个设计的实现。由于在最大程度上继承了以往的设计成果，使变动最小化，所以大大节省了实现步骤所需时间，缩短了设计流程。这种继承以往成果的设计方法的另一个好处就是不会因为当前子模块的修改而影响设计其他部分的时序，增强了设计的可靠性，这点和 Incremental Design 一致。

在 Modular Design 设计方法中，项目领导者（Team Leader）的能力是至关重要的。在进行 Modular Design 设计时，项目领导者具有两重身份：他（她）既需要是一个设计经验丰富的技术能手，又必须是一个效率管理专家。

项目领导者负责系统顶层的设计，并进行模块的划分。更重要的是还必须完成 Modular Design 最重要的一步——Initial Budgeting！Initial Budgeting 的难点在于对设计的顶层进行合理的约束（全局时序、每个子模块的位置约束等）。另外在使用 Floorplanner 对每个子模块进行位置规划的时候，必须要对子模块的面积和所使用的资源（寄存器、RAM、全局时钟资源等）有一个较为准确的估计，制定适合的规划区域。这就不仅仅要求项目领导者是一个设计专家（对 Floorplanner、约束编辑器等要非常熟悉），同时还要求项目领导者具有丰富的实践经验。反之，如果因布局不合理而在较后的阶段需要再次进行 Initial Budgeting，则需要对整个实现步骤全面返工，造成不堪设想的局面。

另一方面，Modular Design 的优势所在即为支持并行工作模式，这就要求项目领导者要有较高的资源（人力、物力等）平衡与配置的意识与能力，能对项目进度做出很好的权衡，

使所有子模块设计任务并行完成，从而不影响模块的最后合并（Final Assembly）阶段的工作进度。

所以说，Modular Design 的最终结果在较大程度上取决于项目领导者这个角色扮演是否成功。

8.3.3 Modular Design 的目录管理

Modular Design 是一种协同合作的设计方式，所以进行有序的目录管理是非常必要的。项目管理者必须为项目成员设计好不同层次与类别的工作目录，以保证整个设计协调、有序地进行。目录组织结构至少应该能区分开综合目录、顶层模块初始预算阶段的工作目录、最后合并阶段的工作目录等。Xilinx 推荐的目录组织结构包含以下独立目录：

1. 综合目录

该目录应该包含设计的顶层模块的 HDL 代码和综合工程文件。而每个子模块的综合目录可以建立在项目成员的个人目录下，其中也应该包含相应子模块的 HDL 代码和综合工程文件。

2. 实现目录

实现目录又分为以下两个子目录：

- 顶层模块实现目录

该目录进行顶层模块的 Initial Budgeting，在所有子模块的 PIM 生成后，也可以在该目录下进行整个设计的 Final Assembly。

- 子模块 PIM 目录

该目录存放所有子模块的 PIM 结果文件。项目成员可以在个人目录下完成子模块的 Active Module Implementation，但是必须将 PIM 结果文件使用“pimcreate”命令全部提交到此目录下。

8.3.4 Modular Design 的常用约束

在 Modular Design 中需要使用时序约束和物理区域约束以保证设计结果满足时序条件和布局规范。常用的约束属性有下述几类，这些约束的具体含义和用法请参考本书第 5 章的论述。

- AREA_GROUP
- COMPGRP
- FROM-TO
- LOC（在模块端口声明中用 PIN 声明实现端口位置约束）
- NET TPSYNC
- OFFSET（在模块端口声明中通过 TPSYNC 分组实现约束）
- PERIOD
- PIN
- RESERVED_SITES（该约束属性不支持 TBUFT、Slice、乘法器、块 RAM 和

CLB 等结构)

- ROUTE
- TIG



以上约束大多数可以由 Floorplanner、Constraints Editor、PACE 等工具根据用户操作自动生成，并不需要用户手动编辑 UCF 文件以实现约束。Floorplanner 的使用方法请参考本书 6.2 节，Constraints Editor 和 PACE 的使用方法请参考本书 5.4 节和 5.5 节。

8.3.5 Modular Design 的报告查看

Modular Design 实现报告主要指映射 (Map) 报告和布局布线 (PAR, Place And Route) 报告。在进行完激活模式实现和最后合并后，可以观察 Map 报告以帮助设计者改进顶层模块的 Floorplan 区域分组。在 Map 报告中应该注意的内容如下：

- 模块化设计总结 (Modular Design Summary)。
这部分内容主要告诉设计者哪些模块使用哪些元件，目前进行的部分化还是全面性的模块最终合并。
- 指导报告 (Guide Report)。
如果在映射过程中使用了指导文件 (Guide file)，报告中此部分将指示所使用的指导模式为 EXACT 或 LEVERAGE，并显示成功指导的比例数。
- 区域分组总结 (Area Group Summary)。
报告中该部分用以说明区域分组的基本信息，所谓区域分组指将某个逻辑结构映射到芯片上规范好的物理区域的约束属性。

在进行完最后合并步骤后，可以观察布局布线报告 (扩展名为 par) 中指导总结报告 (Guide Summary Report) 部分，该报告给出了布线器使用 PIM 结果文件指导元件和信号布局布线的基本信息。也可以观察详细指导报告 (扩展名为 grf)，获得指导性布局布线的详细信息。该报告主要包含以下内容：

- 成功被指导布线的元器件信息。
这部分内容主要列出了所有与 PIM 指导文件相匹配，并成功布线的元器件列表和这些器件的布局布线的实际位置等信息。
- 不能成功被指导布线的元器件信息。
这部分内容主要列出了所有与 PIM 指导文件相匹配，但是不能成功布线的元器件列表和这些器件的布局布线的实际位置等信息。
- 指导布线文件与实际布线结果不匹配的元器件信息。
这部分内容主要列出了所有与 PIM 指导文件不相匹配的元器件的信息。
- 指导布线文件与实际布线结果不匹配的信号信息。
这部分内容主要列出了所有与 PIM 指导文件不相匹配的信号的信息。

8.3.6 使用 XFLOW 自动进行模块化设计

XFLOW 是 Xilinx 提供的一个命令行工具，使用该工具可以帮助用户自动完成实现、仿真等设计流程。XFLOW 的输入除了设计输入文件外还有 XFLOW 流程文件 (FLOW 文件，

扩展名为 `flw`) 和 XFLOW 参数文件 (Option 文件, 扩展名为 `opt`)。其中流程文件是 Xilinx 根据一些常用设计流程提供的标准命令行模板, 流程文件通常使用 `NGDBuild`、`Map`、`PAR`、`Trace` 等实现命令完成设计的实现步骤。用户可以直接使用 Xilinx 提供的标准流程文件, 也可以用这些流程文件为模板, 设计用户自己的流程文件。参数文件主要指明了流程文件所使用的各种命令的参数属性。XFLOW 将用户从烦琐的命令与参数记忆中解脱出来, 帮助用户自动使用 Xilinx 底层操作命令, 完成实现、仿真等设计流程。

在了解了 Modular Design 的设计流程后, 读者可以直接使用 XFLOW 工具的 Modular Design 流程文件自动完成整个实现流程。前面已经反复强调过 Modular Design 实现分为 3 个步骤, 其实 XFLOW 的流程文件也是按照这 3 个实现步骤组织的。

- 参数 “`-initial`”。

该参数对应于 Modular Design 的第一个综合阶段 Initial Budgeting。参数基本使用格式为 “`-initial budget.opt`”, 使用该参数可以自动调用 `NGDBuild` 完成初始预算阶段的所有工作。例如所选 FPGA 器件类型为 “`xc2v40cs144-5`”, 综合生成的顶层逻辑网表文件为 “`top.edf`”, 则 Initial Budgeting 阶段的对应的 XFLOW 命令为 “`xflow -p xc2v40cs144-5 -initial budget.opt top.edf`”。

- 参数 “`-module`”。

该参数对应于 Modular Design 的第二个综合阶段 Active Module Implementation。参数基本使用格式为 “`-module option_file -active module_name`”, 使用该参数可以自动完成子模块激活模式实现阶段的所有工作, 生成 PIM 结果文件, 并调用 `pimcreate` 命令将 PIM 结果文件导入到最后合并目录下。继续使用前面的例子, 则 Active Module Implementation 阶段对应的 XFLOW 命令为 “`xflow -p xc2v40cs144-5 -module balanced.opt -active module_name ~teamleader/mod_des/implemented/top/top.ngo`”。其中 “`balanced.opt`” 为 XFLOW 参数文件, 表述优化目标为在面积与速度之间追求平衡。其他常用参数文件还有 “`fast_runtime.opt`”, 表示以牺牲优化性能为代价, 使运行时间最短; “`high_effort.opt`” 表示优化努力程度为高, 即以较长的运行时间为代价, 获得较好的优化效果。命令行中明确指定了 `top.ngo` 文件的存储目录是 “`~teamleader/mod_des/implemented/top/`”, 读者可以分析一下这个目录结构, 看看是否符合前面所述的 Modular Design 目录结构要求。将 PIM 结果文件导入到最后合并目录下的命令为 “`pimcreate pim_directory -ncd top_routed.ncd`”。

- 参数 “`-assemble`”。

该参数对应于 Modular Design 的第三个综合阶段 Final Assembly。参数基本使用格式为 “`-assemble option_file -pd pim_directory_path`”, 使用该参数可以自动完成模块的最后合并阶段的所有工作。常用参数文件也是 “`fast_runtime.opt`”、“`fast_runtime.opt`” 和 “`high_effort.opt`”, 具体含义与使用方法与上相同。继续使用前面的例子, 则 Final Assembly 阶段的对应的 XFLOW 命令为 “`xflow -p xc2v40cs144-5 -assemble balanced.opt -pd ../pimstop.ngo`”。

8.4 模块化设计方法的设计实例

本节进行一个简单的 Modular Design 设计，希望读者通过对本例的实际操作，熟悉 Modular Design 设计方法的基本操作步骤。需要补充的是，Modular Design 在 ISE 5.2 及其后续版本才作为标准工具包直接集成在 ISE 软件中，ISE 5.2 以前版本中，Modular Design 是需要另外付费购买的，请读者在进行本节操作前首先确认自己的软件版本。

这个例子很简单，顶层包含 3 个子模块：“module_a”、“module_b”和“module_c”。顶层和子模块仅仅包含一些非常简单的逻辑。

✎ 完成一个简单的 Modular Design 设计，参考示例详见本书附带光盘的“Example-8-1”目录

1. 设计的输入与综合。

- (1) 创建 Modular Design 工作目录。虽然顶层模块和子模块的全部设计工作都将由读者一人完成，但是如上节所述，创建合理的工作目录是 Modular Design 的一个重要设计步骤。为了叙述方便起见，请读者完全按照附带光盘“Example-8-1\Modular_Design”下的目录组织结构创建自己的工作路径。
- (2) 设计输入。为了节约时间，笔者已经写好所有模块的源代码，请读者根据个人习惯将 VHDL 或 Verilog 源代码拷贝到自己创建的 Modular Design 工作目录下。源代码存放在本书附带光盘的“Example-8-1\source”目录中。
- (3) 综合顶层模块与所有子模块。需要注意的是仅在综合顶层模块时才插入 I/O 端口，在综合子模块时一律禁止插入 I/O。为了适应不同读者的需求，本书在综合 module_a 时使用了 Synplify Pro 综合工具；在综合 module_b 时使用了 XST 综合工具；在综合 module_c 时使用了 FPGA Express 综合工具，重点演示了如何允许、禁止插入 I/O 端口的选项。图 8-8~图 8-11 分别演示了顶层模块和子模块选用不同工具综合时的参数设置，供读者参考。

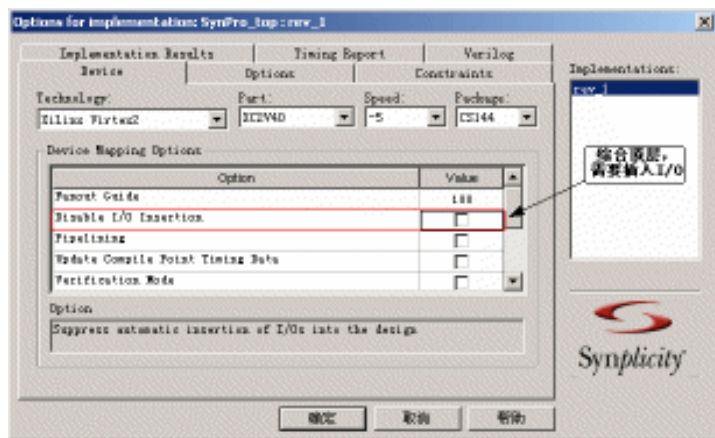


图8-8 使用 Synplify Pro 综合顶层模块时的参数设置

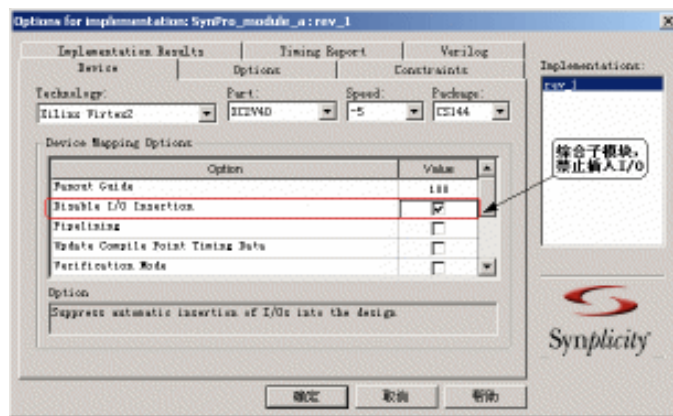


图8-9 使用 Synplify Pro 综合子模块时的参数设置

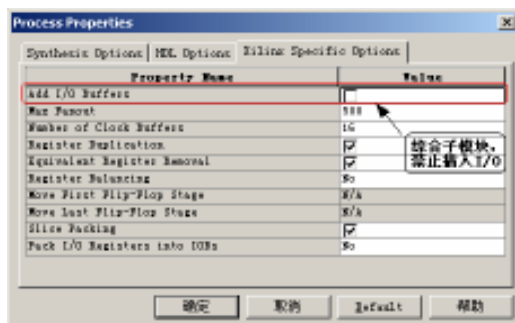


图8-10 使用 XST 综合子模块时的参数设置

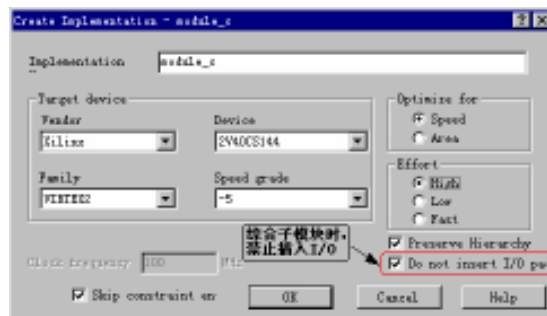


图8-11 使用 FPGA Express 综合子模块时的参数设置

提醒读者注意一点：使用 Synplify/Synplify Pro，FPGA Express 等工具综合时，输出的逻辑网表的扩展名是“edf”，使用 XST 综合时，输出的逻辑网表的扩展名是“ngc”。

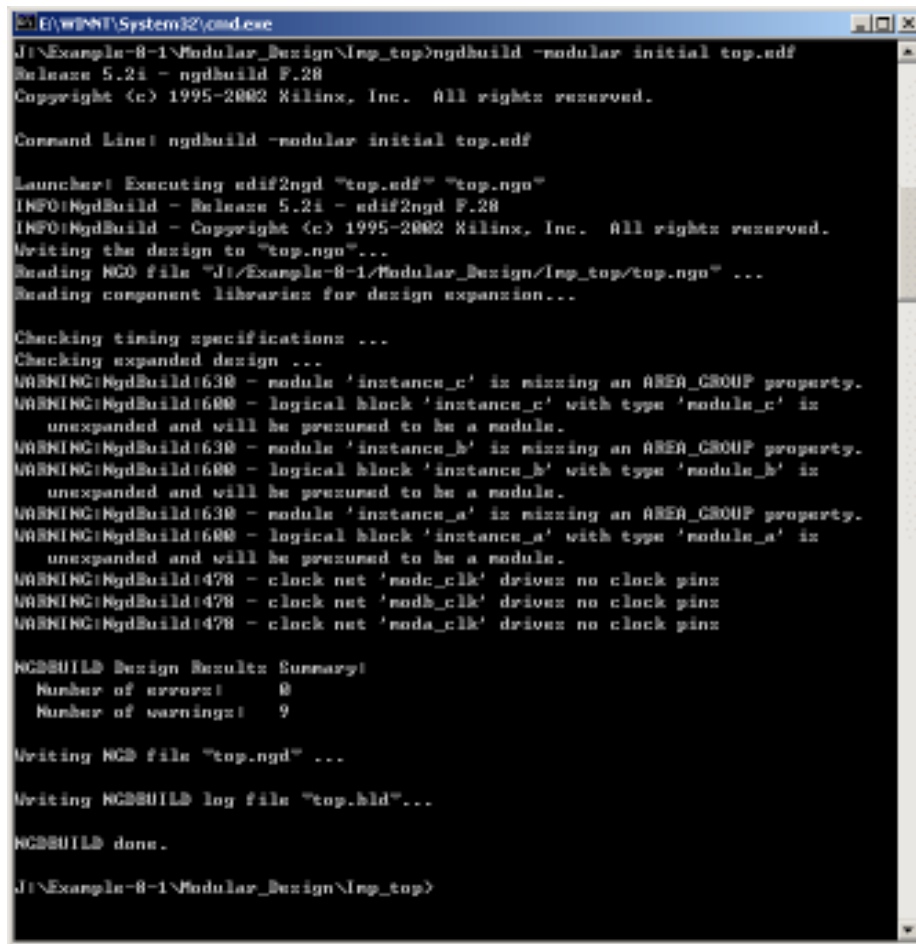
2. 顶层模块的初始预算 (Initial Budgeting)。

顶层模块的 Initial Budgeting 应该由项目管理者完成。

- (1) 首先将综合生成的顶层逻辑网表拷贝到顶层的实现目录“Imp_top”下，运行 Windows 的【开始】/【程序】/【附件】/【命令提示符】命令，唤出命令行窗口。假设读者的计算机上顶层实现目录为“J:\Example-8-1\Modular_Design\Imp_top”，则运行“J:”命令进入逻辑盘符为“J:”的硬

盘，运行“`cd J:\Example-8-1\Modular_Design\Imp_top`”命令进入顶层实现目录。

- (2) 运行“`ngdbuild -modular initial top.edf`”命令进行顶层模块的 Initial Budgeting，如果读者的顶层由 XST 综合生成，则将上述命令的顶层逻辑网表替换为“`top.ngc`”。图 8-12 示意了如何使用 NGDBuild 进行 Initial Budgeting。



```

E:\WINNT\System32\cmd.exe
J:\Example-8-1\Modular_Design\Imp_top>ngdbuild -modular initial top.edf
Release 5.2i - ngdbuild F.28
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

Command Line: ngdbuild -modular initial top.edf

Launcher: Executing edif2ngd "top.edf" "top.ngo"
INFO:NgdBuild - Release 5.2i - edif2ngd F.28
INFO:NgdBuild - Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.
Writing the design to "top.ngo"...
Reading NGO file "J:\Example-8-1\Modular_Design\Imp_top\top.ngo" ...
Reading component libraries for design expansion...

Checking timing specifications ...
Checking expanded design ...
WARNING:NgdBuild:630 - module 'instance_c' is missing an AREA_CSOUF property.
WARNING:NgdBuild:630 - logical block 'instance_c' with type 'module_c' is
unexpanded and will be presumed to be a module.
WARNING:NgdBuild:630 - module 'instance_b' is missing an AREA_CSOUF property.
WARNING:NgdBuild:630 - logical block 'instance_b' with type 'module_b' is
unexpanded and will be presumed to be a module.
WARNING:NgdBuild:630 - module 'instance_a' is missing an AREA_CSOUF property.
WARNING:NgdBuild:630 - logical block 'instance_a' with type 'module_a' is
unexpanded and will be presumed to be a module.
WARNING:NgdBuild:478 - clock net 'modc_clk' drives no clock pins
WARNING:NgdBuild:478 - clock net 'modb_clk' drives no clock pins
WARNING:NgdBuild:478 - clock net 'moda_clk' drives no clock pins

NGDBUILD Design Results Summary:
  Number of errors:    0
  Number of warnings:  9

Writing NGD file "top.ngd" ...
Writing NGDBUILD log file "top.bld"...

NGDBUILD done.

J:\Example-8-1\Modular_Design\Imp_top>
  
```

图8-12 使用 NGDBuild 命令进行 Initial Budgeting



如果 NGDBuild 无法正常完成，并显示“`ERROR:NgdBuild:579 - Can't run ngdbuild in modular design mode for 'device part' architecture`”信息，说明用户的 ISE 因为版本问题不支持 Modular Design。请读者重新安装 ISE 5.2 或更高版本的软件。

- (3) 接着运行“`constraints_editor top.ngd`”命令，启动用户约束编辑器对顶层进行全局时序约束。Constraints Editor 启动后，首先进入新建 ucf 对话框，在约束文件（Constraints File）栏目中填入“`top.ucf`”作为顶层约束文件名，单击  按钮，确认新建约束文件，进入约束编辑器的主界面。出于对问题的简化，这里我们仅仅做一些简单的时序约束：约束时钟周期为 20ns，“Pad to Pad”时间为 20ns，所有输入信号的“Pad to Setup”时间为 10ns，所有输出

信号的“Clock to Pad”时间为 10ns。约束编辑器的具体使用方法请参考 5.4 节。图 8-13 简单示意了如何在 Constraints Editor 中约束全局时序。

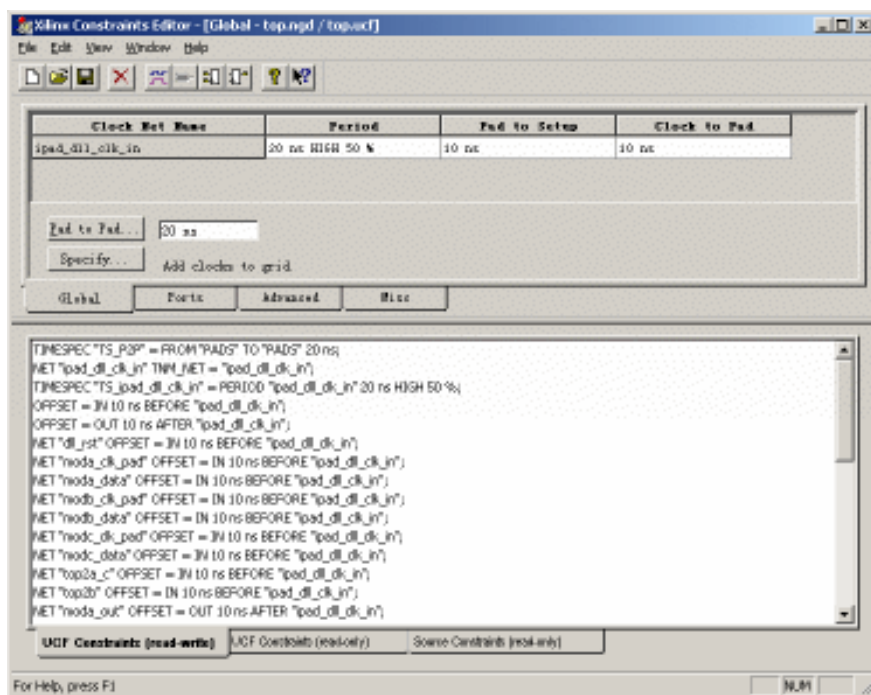


图8-13 使用约束编辑器约束全局时序

细心的读者也许注意到，时钟信号“moda_clk”、“modb_clk”、“modc_clk”并未出现在时钟约束信号列表中，这是为什么呢？请读者观察图 8-12，有 3 条警告信息显示这些时钟信号“drives no clock pins”。在 Modular Design 中，由于子模块都是黑盒子，造成某些时钟信号并未真正驱动时钟输入端，如果需要对这类信号进行约束，必须手动添加约束命令。如对时钟信号“moda_clk”约束周期，需要使用下列约束命令：

```
NET "moda_clk" TNM_NET = "moda_clk";
TIMESPEC "TS_moda_clk" = PERIOD "moda_clk" 20 ns HIGH 50 %;
```

完成时序约束后，保存 top.ucf 文件，退出 Constraints Editor。

- (4) 然后运行命令“**floorplanner top.ngd**”，启动 Floorplanner 进行区域位置约束，指定子模块的面积大小、位置，并约束所有模块的端口位置。选择【File】/【Read Constraints】命令，读入前面在 Constraints Editor 中设置的约束信息。在 Floorplanner 中需要约束的项目主要有以下几个方面：

- 约束顶层 I/O 端口位置
约束方法为在“Design Hierarchy”窗口中展开“Primitives”图标，拖曳端口到“Floorplan”窗口即可。
- 约束顶层逻辑（全局时钟缓冲、三态缓冲、触发器、查找表等）的位置
约束方法为在“Design Hierarchy”窗口中展开“Primitives”图标，拖曳相应的逻辑模块到“Floorplan”窗口即可。约束时需要注意 BUFT 的约束需要满足本

章 8.2.2.1 小节所述的原则。

- 使用“伪逻辑 (Pseudo logic)”连接模块端口

利用 Pseudo logic 连接模块端口的最简便方式为：选择【Floorplan】/【Distribute Options】命令，在弹出对话框中选择“Autofloorplan as needed”选项，这样在进行区域面积约束时，floorplanner 会自动插入 Pseudo logic 连接模块端口。当然如果对自动布局效果不满意，某些逻辑不能自动底层布局，就需要使用手动方式进行底层布局。

- 约束子模块的区域位置

约束方法为：在“Design Hierarchy”窗口展开 top 层次，选中子模块，单击  按钮，然后在“Floorplan”窗口用鼠标画出适当大小的矩形区域，并把子模块的器件符号拖拽到所约定的矩形区域的适当位置即可。

使用 Floorplanner 完成位置区域约束后，保存 UCF、FNF 文件，退出 Floorplanner。Floorplanner 的详细使用方法参见 6.2 节。图 8-14 所示为使用 Floorplanner 进行位置区域约束的示意图。

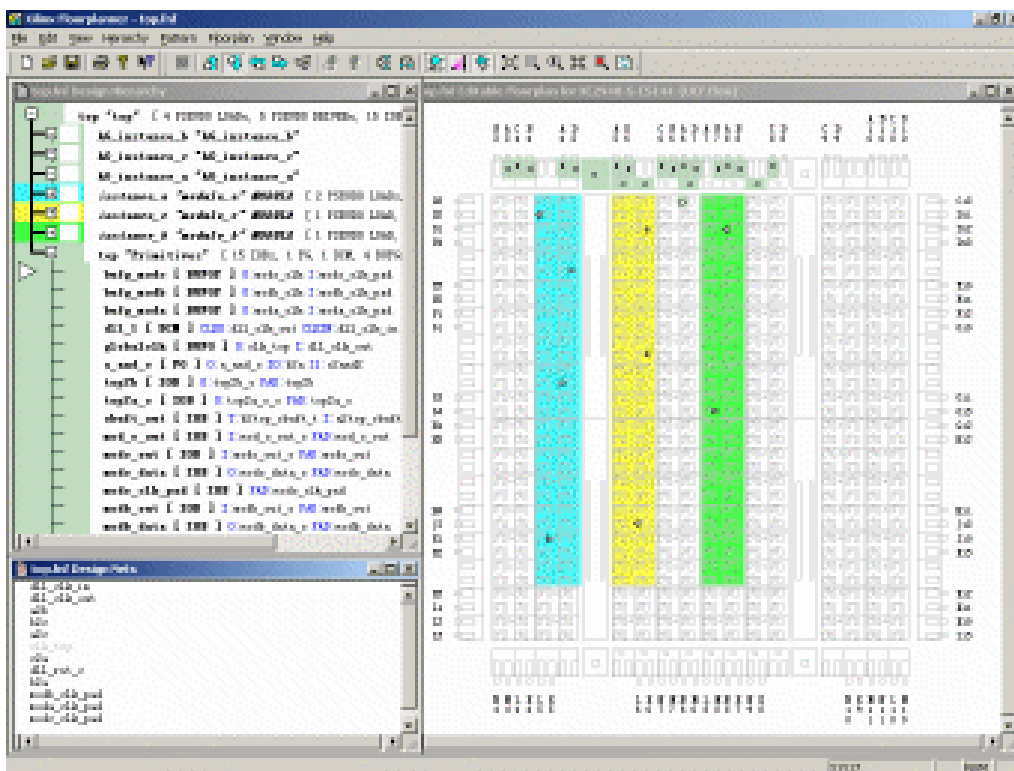


图8-14 使用 Floorplanner 进行位置区域约束示意图

3. 子模块的激活模式实现 (Active Module Implementation)。

子模块的激活实现应该由项目成员完成。本步骤中，几乎所有 Xilinx 实现命令和工具都可以被使用，用户可以根据需要使用“map”、“par”等实现命令，也可以使用 Constraints Editor、Floorplanner 等实现工具。本例有 3 个子模块，我们仅以 modular_a 为例，完成该子

模块的激活模式实现。其他模块的实现由读者参照下列步骤自行完成。

- (1) 文件准备工作。首先将“Modular_Design\syn_modules\module_a”目录下生成的综合网表文件“module_a.edf”拷贝到“Modular_Design\Imp_modules\module_a”目录下；然后将上步在“Modular_Design\Imp_top”目录下生成的“top.ucf”文件拷贝到“Modular_Design\Imp_modules\module_a”目录下，并重命名为“module_a.ucf”。在 Windows 命令行窗口运行“cd \Example-8-1\Modular_Design\Imp_modules\module_a”命令进入到 module_a 的激活实现目录。
- (2) 在命令行窗口运行“ngdbuild -uc module_a.ucf -modular module -active module_a \Example-8-1\Modular_Design\Imp_top\top.ngo”命令，调用 NGDBuild 进行激活模式实现，如图 8-15 所示。对 module_a 的激活状态模式实现完成后，“Modular_Design\Imp_modules\module_a”目录下新生成的 top.ngd 网表中关于 module_a 的结构已经被扩展。

```

E:\WINNT\System32\cmd.exe
J:\Example-8-1\Modular_Design\Imp_modules\module_a>ngdbuild -uc module_a.ucf -mo
dular module -active module_a \Example-8-1\Modular_Design\Imp_top\top.ngo
Release 5.2i - ngdbuild F.28
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

Command Line: ngdbuild -uc module_a.ucf -modular module -active module_a
\Example-8-1\Modular_Design\Imp_top\top.ngo

Launcher: "top.ngo" is up to date.
Reading NGO file "J:\Example-8-1\Modular_Design\Imp_modules\module_a\top.ngo"
...
Reading component libraries for design expansion...
Launcher: "module_a.ngo" is up to date.
Loading design module
"J:\Example-8-1\Modular_Design\Imp_modules\module_a\module_a.ngo"...
INFO:NgdBuild:782 - output buffer 'moda_out_obuf' driving design level port
'moda_out' is being pushed into module 'instance_a' to enable I/O register
usage.
INFO:NgdBuild:784 - input buffer 'moda_data_ihuf' driving design level port
'moda_data' is being pushed into module 'instance_a' to enable I/O register
usage.

Annotating constraints to design from file "module_a.ucf" ...

Checking timing specifications ...
INFO:XdcHelpers:862 - INH "igad_dll_clk_in", used in period specification
"TS_igad_dll_clk_in", was traced through BCM instance "dll_1" to the loads on
the CLAO output.
Checking expanded design ...
WARNING:NgdBuild:680 - logical block 'instance_c' with type 'module_c' is
unexpanded and will be presumed to be a module.
WARNING:NgdBuild:680 - logical block 'instance_b' with type 'module_b' is
unexpanded and will be presumed to be a module.
WARNING:NgdBuild:478 - clock net 'modc_clk' drives no clock pins
WARNING:NgdBuild:478 - clock net 'modb_clk' drives no clock pins

NGDBUILD Design Results Summary:
  Number of errors:    0
  Number of warnings:  4

Writing NGD file "top.ngd" ...

Writing NGDBUILD log file "top.blk"...

NGDBUILD done.
  
```

图8-15 在命令行窗口运行命令进行子模块激活模式实现

- (3) 在 “Modular_Design\Imp_modules\module_a” 目录下，再次执行 “constraints_editor top.ngd” 命令，启动用户约束编辑器对扩充了 module_a 子模块的顶层进行更为详细的时序约束。在新建 ucf 对话框的 Constraints File 栏中选择约束文件 “module_a.ucf”，单击  按钮，确认新建用户约束文件，进入约束编辑器的主界面。约束时注意不要破坏在 Initial Budgeting 阶段项目管理者设计的全局性约束条件，而仅仅对扩展的子模块进行时序约束，如图 8-16 所示。

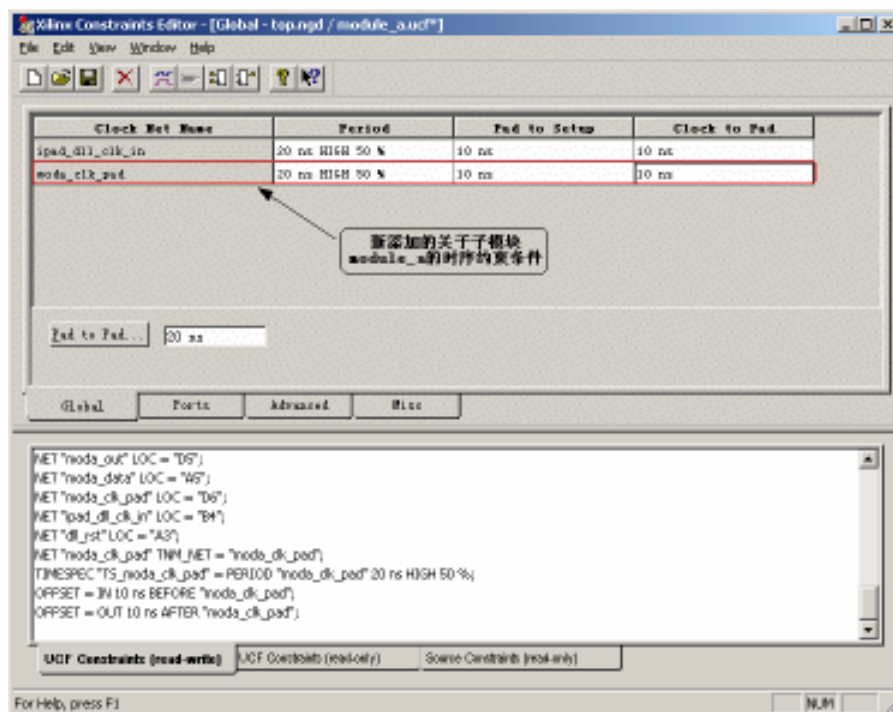


图8-16 对扩展的子模块 module_a 进行时序约束

对扩展子模块时序约束完成后，保存用户约束文件 module_a.ucf，退出约束编辑器。

- (4) 将新设计的用户约束文件 module_a.ucf 反标到设计中，重新运行 “ngdbuild -uc module_a.ucf -modular module -active module_a \Example-8-1\Modular_Design\Imp_top\top.ngo” 命令，再次对子模块 module_a 激活模式实现。
- (5) 使用 “map top.ngd” 命令，对设计进行映射，如图 8-17 所示。这里并不需要添加任何关于 Modular Design 的特殊参数，所有映射所需信息已经包含在激活实现产生的 top.ngd 网表中。
- (6) 运行 “par -w top.ncd top_routed.ncd” 命令，完成设计的布局布线，如图 8-18 所示。参数 “-w” 表示重新覆盖以前的 NCD 网表，参数 “_routed” 表示不用重新覆盖映射结果，直接布局布线。

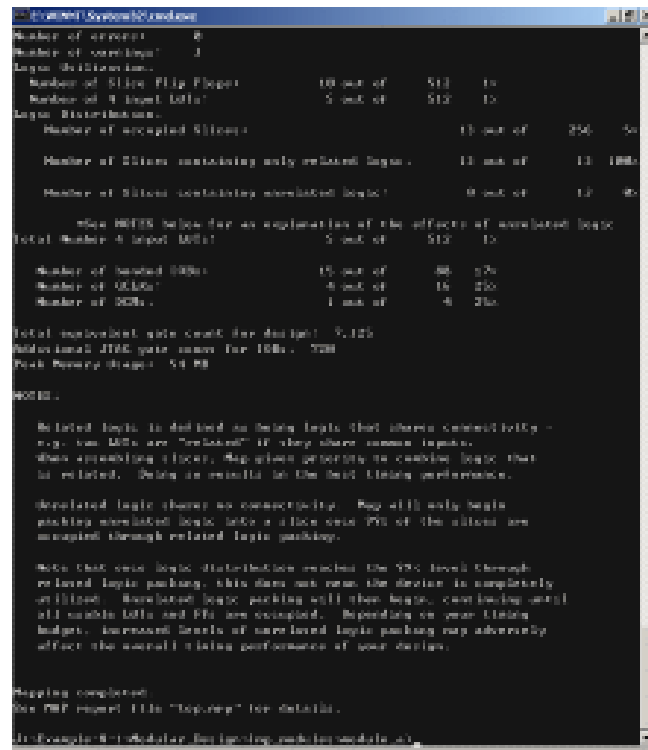


图8-17 使用 map 命令映射设计

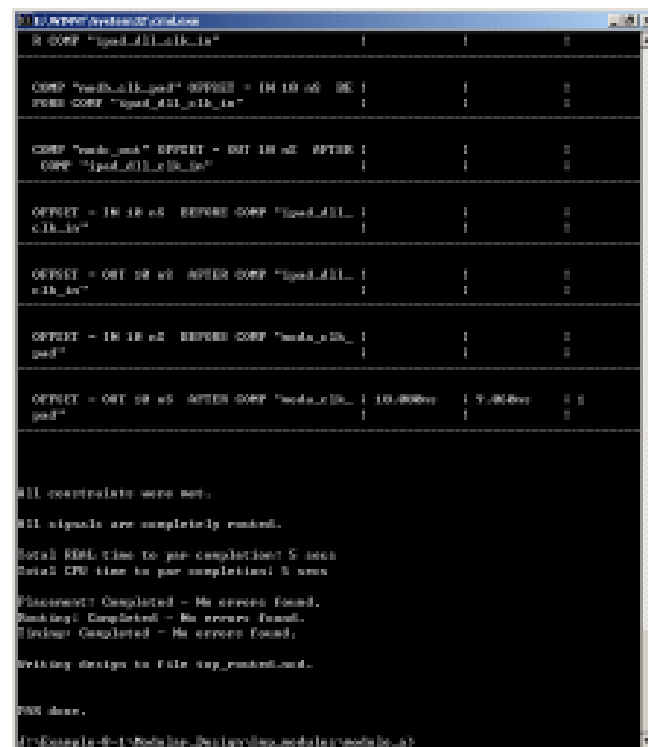


图8-18 使用 par 命令对设计进行布局布线

- (7) 运行“`trce top_routed.ncd`”命令，检查实现结果的静态时序，如图 8-19 所示。静态时序报告的扩展名为“`twr`”或“`twx`”，可用文本编辑器观察。

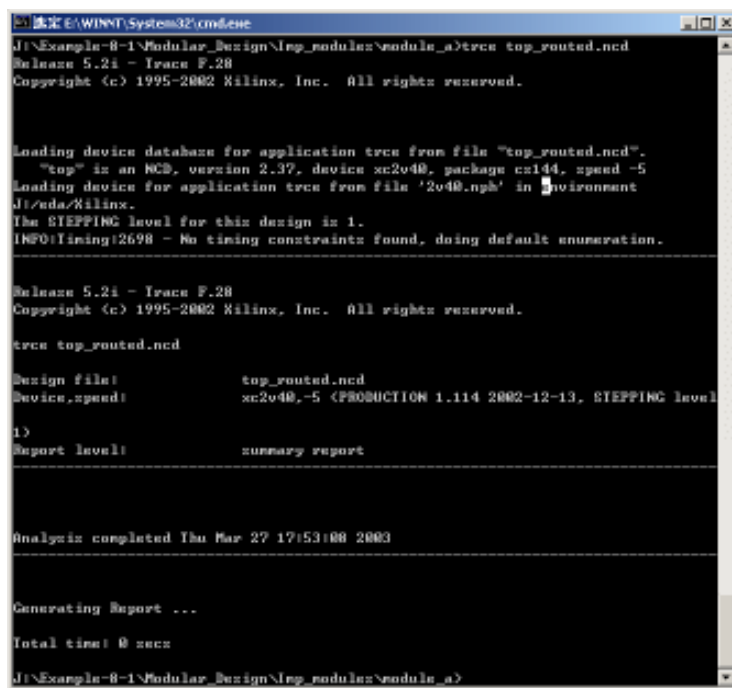


图8-19 使用 trce 命令生成静态时序报告文件

- (8) 如果设计的时序和区域约束不满足条件，则需要再次运行“`floorplanner top.ncd`”命令，调整设计；如果设计的时序与区域约束完全正确，则没有必要运行 Floorplanner。
- (9) 使用“`pimcreate \Example-8-1\Modular_Design\PIMs -ncd top_routed.ncd`”命令，将子模块激活实现的 PIM 结果拷贝到“`\Example-8-1\Modular_Design\PIMs`”目录下。module_a 的 PIM 结果文件如图 8-20 所示。

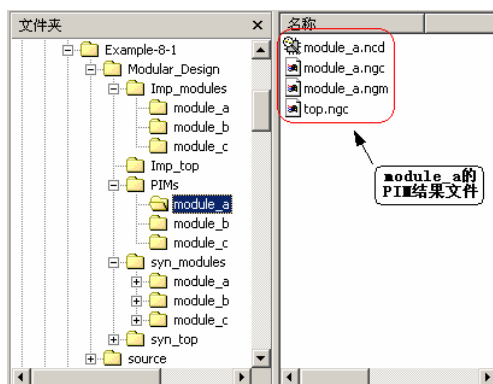


图8-20 module_a 的 PIM 结果文件

至此，模块 module_a 的激活模式实现全部完成，请读者以同样的方式完成子模块

module_b、module_c 的激活模式实现。

4. 所有模块的最后合并 (Final Assembly)。

模块的最后合并应该由项目管理者完成。基本操作步骤如下：

- (1) 首先在命令行窗口运行 “`cd \Example-8-1\ Modular_Design\Imp_top`” 命令进入顶层实现目录。
- (2) 在命令行窗口运行 “`ngdbuild -modular assemble -pimpath \Example-8-1\Modular_Design\PIMs top.ngo`” 命令，进行模块的最后合并。
- (3) 运行 “`map top.ngd`” 命令，合并后的设计进行映射。
- (4) 运行 “`par -w top.ncd top_routed.ncd`” 命令，对合并后的设计进行布局布线。
- (5) 运行 “`trce top_routed.ncd`” 命令，生成扩展名为 “`twr`” 或 “`twx`” 的静态时序报告，检查实现结果的静态时序。
- (6) 如果需要进行时序仿真，可用运行 “`ngdanno -o top.nga top.ncd top.ngm`” 命令，进行 ngd 网表的反标，使用 “`ngd2vhdl top.nga`” 命令生成 VHDL 仿真文件，使用 “`ngd2ver top.nga`” 命令生成 Verilog 仿真文件。

至此，Modular Design 设计过程全部完成。需要补充说明以下几点：

- 在 Modular Design 设计中，最重要的是实现步骤的 Initial Budgeting 阶段。在该阶段需要对设计进行全局性的规划和约束，但是由于在设计初期对子模块的信息所知较少，所以有可能会发生时序约束和区域位置约束不够准确、甚至不够合理，导致子模块的 Active Module Implementation 发生错误，这时需要返回 Initial Budgeting 阶段，根据子模块激活实现的信息反馈，修改顶层的全局性时序与区域位置约束。使用子模块激活实现的反馈信息，调整顶层的全局性约束，可以改善约束的准确性，从而在一定程度上优化设计。
- 如果读者因为不熟悉 Constraints Editor 和 Floorplanner 而造成设计时序、区域位置约束不合理，导致子模块 Active Module Implementation 发生错误，可以参考附录光盘中 “`Example-8-1\Modular_Design\Imp_top`” 目录下的 `top.ucf` 文件，修改时序、位置约束。
- Modular Design 实现步骤完成后，整个 FPGA/CPLD 设计流程并未完成，还需进行配置文件的生成与下载，运行 “`bitgen top_routed.ncd top.bit`” 命令可以生成配置 BIT 文件。
- 本章在进行 Modular Design 设计时，为了叙述方便，使用命令行格式调用 Constraints Editor、Floorplanner 等工具完成各种操作，其实许多操作也可以在相应工具的图形界面下完成。另外使用上节介绍的 XFLOW 工具，可以自动完成 Modular Design 命令行的调用。

8.5 增量式设计方法的基本概念

增量式设计 (Incremental Design) 方法是一种能在小范围改动情况下节约综合、实现时间并继承以往设计成果的设计手段。增量式设计方法主要应用于下列 3 种情况：

- 在实验室调试时需要对设计进行一些小的修改；

- 在仿真时需要对设计进行一些小的修改；
- 因设计的局部不满足时序要求而需要做一些小的修改以改进时序。

合理地运用增量设计能带来两方面优点：

- 能够减少综合、实现过程（特别是布局布线过程）的耗时。当仅仅对某个大型设计的局部加以细微改动时，增量综合、实现工具仅仅对改动部分重新编译，如果改动模块的接口设计恰当，将不会影响其余部分的综合与实现结果，在布局布线时只需以上次的实现结果作为指引即可，从而节约了整个编译、布局布线与优化的耗时。
- 能够继承未修改区域的实现成果，这里的实现成果主要指在时序和面积两个方面。如果一个设计经过多次调试，附加合适的约束，设置恰当的参数达到了最佳实现成果，特别是满足比较苛刻时序要求，但是因为对某个细节进行了代码修改，而必须全部重新综合、布局布线，这样可能使前面所做的精心调整工作无效。这个问题的解决途径就是增量式设计方法。增量实现时，综合工具仅仅对细微修改的模块重新综合，得到新的网表，而对这个模块与其余部分的接口尽量不变动，其余部分的综合结果也保持一致。在增量实现时，其余模块的实现结果将完全被继承下来，仅仅对新改动的网表进行布局布线，从而保证在最大程度上继承以往的实现结果。

增量设计中常用的概念有如下 4 个：

- Logic Group：逻辑分组是惟一的逻辑层次中的若干逻辑实体的划分。比如在顶层逻辑层次中每个子模块即为一个逻辑分组。在代码中即为顶层中“module”（Verilog）和“entity”（VHDL）定义的子模块的实体。需要说明的是当父模块无任何实际逻辑功能时，Logic Group 也可以是更低层次的子模块实体。
- AREA GROUP：区域分组是 Xilinx 的一种区域约束，用于将指定模块划分到同一位置区域中去。
- AREA GROUP Range：区域分组的范围指用 FPGA 的行、列序号标明的矩形区域范围。
- Guide：指引（Guide）是 Xilinx 的一种布局布线模式，使用 Guide 模式布局布线，布局布线器将网表中未变化的部分直接按照前一次的实现结果布局布线，从而在最大程度上集成前一次的实现结果。

增量设计的效果关键取决于应用和合理性。如果在不适当的设计中运用增量设计，不但不能达到预期的效果，反而可能适得其反，造成综合实现时间增加或者实现结果恶化。合理采用增量实现必须要注意以下几点：

- 综合过程使用增量模式，在保留结构层次模式（hierarchical mode）下综合。这要求设计时必须采用层次化编码。顶层按照功能和电路结构划分为若干逻辑模块，合理的划分使需要修改的细节尽量集中在某一个逻辑模块中，则其余未改动的模块将不再被重新综合，而仅仅被改动的模块被重新综合，这样才能体现增量综合的优势。ISE 中集成的 XST、Synplicity 和 Exemplar 综合工具都支持增量综合模式。

- 逻辑模块分组必须以寄存器为输出，即用寄存器分割模块。这一点其实不仅仅是增量设计的需求，也是合理划分模块的一个基本要求。如果采用同步时序方式设计电路，用寄存器分割逻辑模块，模块间的接口尽量简单，则时序优化路径集中在同一模块内部而不是模块之间的边界上，这样能够使综合器完整地掌握需要时序优化的路径，从而避免了因一个模块内部改变而通过边界影响到其他模块的时序这种不利于增量设计的情况的发生。另外，即使综合引擎没有对模块间的组合逻辑做优化，Xilinx ISE 的内部布局布线引擎在映射时也会超越模块的边界进行时序调整与优化，这样将影响到其他相邻的模块，使它们不能正常地被引导 (guide) 布局布线，从而影响了增量实现的效果。
- 顶层模块不应该包含复杂逻辑，仅仅包含一些 I/O 定义和时钟资源分配等。理想的模块划分模式是顶层模块仅仅包含 I/O 定义、时钟分配逻辑和所有子模块的实例化，真正的功能实体用子模块的逻辑描述。增量设计方法希望将所有的逻辑实体分割到子模块中去，而顶层模块不含任何实际的逻辑功能，以便于做相应的区域约束。顶层包含实际逻辑功能的缺点在于：当顶层改变时，相关的 Logic Group 的接口将发生变化，从而影响了 Logic Group 的结构，在做编译和布局布线时，就无法对相关的 Logic Group 进行 Guide 模式实现，从而影响了 Incremental Design 的效能。
- 顶层模块包含的逻辑模块数适当。这点主要因为每个 Logic Group 都要做相应的 AREA GROUP 区域约束，如果划分过多的 Logic Group 势必会对使用 PACE 和 Floorplanner 做区域约束造成麻烦；如果划分的 Logic Group 过少，每个 Logic Group 的功能势必繁杂，改动某一点将会造成整个 Logic Group 的重新综合和布局布线，影响 Incremental Design 的效能。

8.6 增量设计方法的设计流程

整个增量设计的流程分为如下几个步骤：

- (1) 为每个 Logic Group 规划区域分组 (Area Group)，将逻辑网表中的 Logic Group 用 Area Group 锁定。这个步骤其实就是一个位置与区域约束的过程。
- (2) 基于 Area Group 的区域约束做布局布线，得到布局布线初始指引文件 (Guide)。
- (3) 根据仿真或调试需求，对某个 Logic Group 做一些细微改动。
- (4) 对某个 Logic Group 改动后，运行增量综合工具，更新修改后的 Logic Group 的逻辑网表。
- (5) 使用 ISE 增量实现参数设置进行映射 (Map) 与布局布线 (P&R, Place and Route)。
- (6) 依据步骤 (2) 得到的初始 Guide 文件做增量模式布局布线。

如果需要对某个 Logic Group 做进一步修改，则重复步骤 (3) ~ (5)。并使用上次步骤 (5) 的 Map 和 P&R 结果作为新的实现 Guide，替代步骤 (2) 的 Guide。

增量设计的流程主要分为增量综合和增量实现两个步骤。这也是增量设计流程与传统设

计流程的显著区别，至于设计输入、仿真、配置调试等设计步骤与传统设计流程完全一致。

8.6.1 增量综合

增量综合指仅仅对改变部分重新综合，而对所有未改变部分保持原有的综合结果的一种综合技术，对应于增量设计流程中第（4）步。传统的综合技术即使有细微的改动（源代码、综合约束属性、综合优化参数等方面），也会对整个设计重新综合，以得到崭新的综合结果。

目前 ISE 中集成的综合工具基本都支持增量综合技术，这些工具中进行增量综合的要点介绍如下：

1. Synplify/Synplify Pro 综合工具

Synplicity 公司出品的 Synplify/Synplify Pro 综合工具是本书第 4 章重点介绍的综合工具之一，Synplicity 综合工具通过 B.E.S.T（行为级提取技术）和改进的时序驱动（Timing Driven）综合引擎以及多项整体性综合优化技术闻名于业界，其综合耗时较短，且综合结果在速度和面积上都能令人满意。关于 Synplify/Synplify Pro 综合技术的细节请参阅本书第 4 章。

Synplify Pro 7.2 及更高版本新添的 MultiPoint Synthesis（多点综合）特性即为增量综合技术。使用 MultiPoint 综合时，Logic Group 被称为编译结点（Compile Point），Synplify Pro 对每个 Compile Point 独立进行编译、优化。从而保证了每个 Compile Point 内部的修改仅仅需要对自身重新综合，对其他 Compile Point 没有影响。

关于如何在 Synplify Pro 中使用 MultiPoint 进行增量综合的详细步骤参见本章 8.7 节。

2. Leonardo Spectrum 综合工具

Leonardo Spectrum 是 Mentor 公司出品的高效综合工具，其界面流畅，在综合耗时和综合结果（在面积和频率两个方面）上与 Synplify Pro 不相上下，这两者是目前最流行的综合工具。限于篇幅，本书不对 Leonardo 做过多讨论。为了照顾使用 Leonardo 的用户，仅对增量综合的要点加以强调。

Leonardo 支持自顶向下（Top-Down）和自底向上（Bottom-Up）两种增量综合模式，并推荐使用 Bottom-Up 综合方法。Leonardo 使用 Bottom-Up 综合方法时，会将每个 Logic Group 综合出单独的 EDIF 逻辑网表，并将顶层模块也综合出一个 EDIF 逻辑网表。此时，如果某个 Logic Group 被修改后，仅仅需要对这个模块进行重新综合，得到单独的 EDIF 网表。在实现时只需根据顶层模块描述的接口关系，将所有子模块的逻辑网表合并，就完成了增量化设计。这种思路在早期综合工具中经常使用，能有效地节约大型复杂设计的综合时间。

Leonardo 综合工具 Bottom-Up 增量综合方法分为如下几个步骤：

- (1) 首先将每个 Logic Group 单独综合，得到逻辑网表。前面说过，Logic Group 简单地说也就是顶层模块下完成逻辑功能的子实体，在 Verilog 语言中即顶层的一个个“module”的实例化，在 VHDL 语言中即顶层的一个个“entity”的实例化。这个步骤的关键在于选用“macro”综合模式，这个综合模式告诉综

合引擎所综合优化的仅仅是一个内部模块，而不是设计的顶层，综合器自动指定不插入 I/O 缓冲（I/O Buffer）和时钟缓冲（Clock Buffer）。假设 Logic Group 的名称为“a”，用 Verilog 源代码“a.v”描述，则选择 macro 综合模式的 TCL 命令脚本为：

```
Read {a.v}
Optimize.work.a.INTERFACE -macro
auto-write a.edf
```

如果用户不习惯使用 TCL 命令脚本模式，也可以在 Leonardo 的 GUI 下直接设定不插入 I/O 和时钟 Buffer。

- (2) 第二步要综合顶层模块。前面已经论述过，理想的顶层模块应该仅仅包含 I/O 和时钟资源分配以及各个子模块实例声明，所有实际的逻辑功能在 Logic Group 中完成。所以，综合顶层模块时就不应该在对各个 Logic Group 做任何优化，而仅仅将子模块的接口与连接关系描述清晰即可。为了保证在综合顶层模块时不对 Logic Group 做任何优化，必须使用“dont_touch”综合参数。另外还有一个相关参数“noopt”必须使用，该参数指明了前面对所有 Logic Group 的优化结果并不写入顶层 top.edf 中去。如果不使用“noopt”参数，则 Leonardo 在综合顶层时，会将前面所有 Logic Group 的综合优化结果合并到顶层模块的 EDIF 网表中去，这样当对某个子模块进行修改并重新编译后，必须对顶层模块也重新编译，从而浪费了综合时间，所以在做增量综合时应该附加“noopt”参数。如果顶层模块为 top.v，而 Logic Group 是由 a.v、b.v、c.v 三个文件描述，则增量式综合顶层模块的 TCL 命令脚本为：

```
Read {a.xdb b.xdb c.xdb}
Read top.v
dont_touch {a b c}
noopt {a b c}
Optimize .work.top.INTERFACE .. -chip
```

其中 a.xdb、b.xdb 和 c.xdb 分别是 Logic Group a、b、c 的子模块综合优化结果数据库文件。

- (3) 第三步就是在实现过程的第一步编译（Translate）步骤中，用 ngdbuild 实现命令将顶层模块的逻辑网表和每个子模块的逻辑网表合并起来，构建成一个完整设计的 ngd 数据文件。关于 ngdbuild 命令的含义与基本使用方法参考本章 8.2.2.1 小节的论述。

3. FPGA Compiler II 综合工具

Synopsys 的 FPGA Compiler II 综合工具也具备增量综合功能。在综合时使用模块级增量综合技术（BLIS，Block Level Incremental Synthesis）。

4. XST 综合工具

XST 支持单一工程的模块级增量综合（BLSI）。实现方法为在 XST 的约束文件（扩展名为 xcf）中附加逻辑分约束，从而告知 XST Logic Group 的边界。XST 在综合时，所有

的编译与优化都不超越用户在 XCF 文件中约定的 Logic Group 的边界，以达到在细微修改后仅仅对 Logic Group 内部进行重新综合的目的。例如仅仅对 Logic Group A 做了细微修改而需要重新编译，而 Logic Group B 和 Logic Group C 不需要重新编译，则 XCF 文件中增量综合的约束为：

```
MODEL "top"
incremental_synthesis=yes;
MODEL "A"
incremental_synthesis=yes;
MODEL "B"
incremental_synthesis=yes;
MODEL "C"
incremental_synthesis=yes;
MODEL "top" resynthesize=no;
MODEL "A" resynthesize=yes;
MODEL "B" resynthesize=no;
```

8.6.2 增量实现

整个增量设计流程中除第 3 步外，都属于增量实现步骤。增量实现大概可以分为以下 3 步：

- (1) 规划区域分组约束；
- (2) 初始布局布线得到第一次指引文件；
- (3) 调试修改并做增量综合后，在初始指引文件下进行第二次映射与布局布线。

通过前面的学习读者会清晰的感觉到：增量设计方式与 Modular Design 设计方法有很多相似之处，特别是增量实现步骤，也必须创建区域分组约束。其实两者创建区域分组约束使用的工具和命令也完全一致，如 PACE，Floorplanner，Constraints Editor 等。

并且 Incremental Design 与 Modular Design 相似，最重要的都是第一步——创建区域分组约束。只有合理的区域分组约束，才能使每个 Logic Group 恰当地被映射并布局布线到芯片事先规划好的面积区域，区域分组的大小、位置、边界、包含的可编程逻辑资源（LUT、Register、Block RAM、嵌入式 Hard Core 等、乘法器）的数量和种类都直接影响着区域分组的效能，决定着 Incremental Design 和 Modular Design 的结果。

1. 增量设计创建合理的区域分组约束的原则

简而言之，好的区域分组约束才能发挥 Incremental Design 和 Modular Design 的效能，不恰当的区域分组约束往往会起到适得其反的效果。为增量设计创建合理的区域分组约束要遵循以下原则：

- 所有 I/O 引脚必须被位置锁定。
- 区域分组约束的范围不应该重叠。这里需要强调一下，在 Xilinx 实现步骤中，每个区域分组约束的位置必须是惟一的，不允许重叠、交叉。这与 Amplify 做物理综合优化时规划的区域分组约束不同。Amplify 逻辑优化设置

的区域分组是弱约束，比较笼统；而 Xilinx 实现步骤做的物理区域分组是强约束，是显化位置的约束。

- 将逻辑上联系紧密的 Logic Group 对应的 Area Group 规划一起，以使两者直接的通信路径的延时最小。
- 将与 I/O 端口密切联系的区域分组布置到 I/O 单元附近。
- 将某个区域分组联系密切的 I/O 尽量规划在这个区域分组的周围。
- 每个区域分组中预留的资源余量应该尽量一致。比如每个区域分组的 Slice 利用率应该尽量一致，平稳变化。
- 区域分组包含的 FPGA 资源（Slice、Block RAM、TBUF、Multipliers 等）应该与对应 Logic Group 所使用的资源一致。如果一个模块无法避免地使用了多项 FPGA 底层资源，这时也可以将几个不同的位置区域拼合为一个区域分组。可以使用 PACE、Floorplanner 等工具完成，也可以在约束文件中使用如下语法约束：

```
INST Logic_Group_A AREA_GROUP = AG_Logic_Group_A ;
AREA_GROUP "AG_Logic_Group_A" RANGE = SLICE_X0Y20:SLICE_X20Y30 ;
AREA_GROUP "AG_Logic_Group_A" RANGE = RAMB16_X0Y2:RAMB16_X0Y2 ;
AREA_GROUP "AG_Logic_Group_A" RANGE = MULT18X18_X0Y1:MULT18X18_X0Y1;
AREA_GROUP "AG_Logic_Group_A" RANGE = TBUF_X0Y0:X1Y0;
```



Incremental Design 和 Modular Design 的重点和难点都在于如何合理地创建区域分组约束。这不仅要灵活运用 Floorplanner、Constraints Editor 和 PACE 等工具，还要熟悉 FPGA 内部的结构和资源分布。

2. 使用 PACE 创建区域分组约束

PACE 是 ISE 5.1 新添的工具软件，主要用于区域约束与管脚锁定。在 Incremental Design 中 PACE 是一个重要工具，通过 PACE 创立区域分组约束，生产用户约束文件（UCF）指导映射与布局布线步骤。关于 PACE 的详细论述请参考本书 5.5 节的论述。当然使用 Floorplanner 也可以完成区域分组约束，只是操作相对复杂一些，关于 Floorplaner 的使用请参考本书 6.2 节的详细论述。

3. 创建初始指引文件

初始指引文件（Guide File）使用上次成功实现的结果引导本次映射和布局布线，包括满足区域分组内时序要求和区域分组约束的 Map 和 P&R 的结果文件等。假设设计的名称为“design_name”，则 Map 输出的结果文件名称为“design_name_map.ngm”和“design_name_map.ncd”；而 P&R 输出的结果文件名称为“design_name.ncd”。

Guide File 的创建步骤如下：

- (1) 以包含创建区域分组约束的 UCF 文件为约束，对增量综合生成的 EDIF 逻辑网表进行实现（包含 Translate、Map 和 P&R 等 3 个步骤）。
- (2) 在映射（Map）步骤完成后，观察映射报告（扩展名为 map），检查区域分组约束是否发挥了作用，并检查每个区域分组的面积利用率，确保每个区域分

组都没有超过 100%的面积利用率，且每个分组的利用率基本一致。

- (3) 确保每个区域分组内部的时序约束都被满足，如若不然，则需要返回到增量实现的第一步，重新进行区域分组。

满足步骤 (2)、(3) 所述条件的 Map 和 P&R 的结果文件 (ngm、ncd 文件) 即为下次增量实现的指引文件。在下次增量实现时，布局布线器会对比 Guide 文件与本次实现目标，对于未改变的部分将直接用 Guide 文件所描述结构完成映射与布局布线。

4. 使用 Guide 文件指引映射

在增量实现模式下，映射器将根据上次映射得到的 Guide 文件 (design_name_map.ngm 和 design_name_map.ncd) 指导映射，对未改变的 Logic Group 不做任何修改与优化，完全按照 Guide 文件描述的结构映射到目标芯片上去。而对于做了修改的 Logic Group 所描述的逻辑，映射器将重新进行映射。

如图 8-21 所示，在 ISE 中设置映射 Guide 文件的方法为：打开资源操作窗中 MAP 步骤的属性对话框，选择“MAP Guide Mode”为“Incremental”，然后指定指引文件“design_name_map.ncd”的路径。用户应该将 ngm 与 ngd 文件拷贝在同一个目录下，且保证两个文件的文件名一致，否则 ISE 将不能正确引导 ngm 文件。



图8-21 ISE 中设置 Map 步骤 Guide 文件

用 TCL 命令脚本设置 Guide 文件的语法如下：

```
map -gm incremental -gf <design_name>_map_guide.ncd
```

补充说明一点，映射指引模式 (Map Guide Mode) 共有 4 个选项，除了“Incremental”以往还有“None”、“Exact”和“Leverage”，这 4 个选项在布局布线属性对话框中 PAR Guide Mode 同样会出现，各个选项在两个阶段的含义基本一致。

- None：顾名思义，即不采用 Guide 文件指导 Map 或 PAR，是 Map 或 PAR Guide Mode 的默认值。
- Exact：在该模式下，与 Guide 文件相匹配的逻辑将被锁定，更改的逻辑部分只能做较小的改动与优化，这种模式常常被使用在对一些 IP Core 的映射与布局布线中。
- Leverage：在该模式下，映射器或布局布线从与 Guide 文件匹配的逻辑器件开始做 Map 或 PAR，整个实现过程尽量遵循 Guide 文件，但是也会做一些小的修改与优化也便于 Map 或 PAR，以达到更好的时序和面积效果。这是一种整

体性的 Guide 方式，其实现结果一般较好，但是耗时较长。

5. 使用 Guide 文件指引布局布线

如图 8-22 所示，在 ISE 中设置布局布线 Guide 文件的方法为：打开资源操作窗中 PAR 步骤的属性对话框，选择“PAR Guide Mode”为“Incremental”，然后指定指引文件“design_name.ncd”的路径。

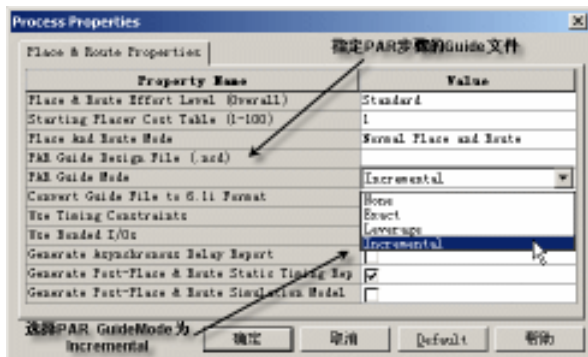


图8-22 ISE 中设置 PAR 步骤 Guide 文件

用 TCL 命令脚本设置 Guide 文件的语法如下：

```
par -gm incremental -gf <design_name>_par_guide.ncd
```

选择增量式布局布线，布局布线引擎在 PAR 时，对于没有做任何修改的 Logic Group 将保持 Guide 文件的布局布线结果，对于改变的 Logic Group，布局布线器在确保边界条件不变的条件下，对该 Logic Group 内部进行最大程度的优化。从而达到了 PAR 步骤耗时较小，而对局部进行修改和优化的目的。

整个增量实现步骤完成后，需要检查映射报告（扩展名为 map）和布局布线报告（扩展名为 par），阅读报告的注意事项与 8.6.3 小节所述“Modular Design 的报告查看”完全一致。

8.7 增量设计方法的设计实例

本节我们用一个简单的示例帮助读者熟悉 Incremental Design 的操作流程。为了方便论述，我们沿用本章 8.4 节 Modular Design 示例的源代码。目前多数工程师习惯使用 Synplify Pro 综合工具，本例遵循这一习惯，使用 Synplify Pro 综合工具做演示。需要注意的是，Synplify Pro 只有 7.2 及更高版本才支持增量综合，ISE 只有 5.1 及更高版本才支持增量实现，请读者学习前先确认自己软件的版本号。

🔑 完成一个简单的 Incremental Design 设计，参考示例详见本书附带光盘的“Example-8-2”目录

1. 使用 Synplify Pro 综合工程。
- (1) 单击 **P** 按钮，启动打开工程对话框，单击 **Project Wizard...** 按钮启动工程向导。选择新建工程类型为“Synthesis”，键入工程名和存储路径，如图 8-23 所示。

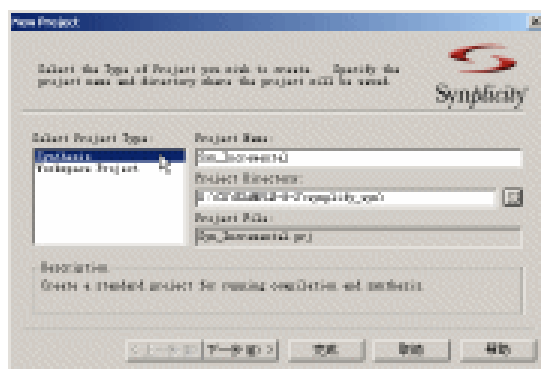


图8-23 在 Synplify Pro 中创建工程

- (2) 单击 **下一步(N) >** 按钮，在接下来的对话框中通过单击 **Add Files ...** 按钮添加工程文件（参见图 8-24）。其中 module_a、module_b 和 module_c 是 3 个 Logic Group，top.v 是设计的顶层模块，Virtex2.v 用于声明工程中使用到的 Xilinx 硬件原语。所有源文件存放在本书附带光盘的“Example-8-2\source”目录下。

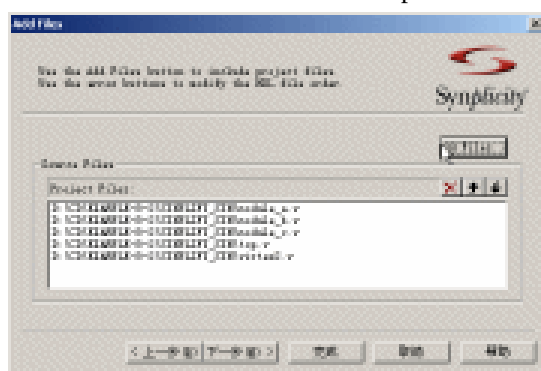


图8-24 向工程中添加源文件

- (3) 单击 **下一步(N) >** 按钮，按图 8-25 所示设置器件类型与优化参数。

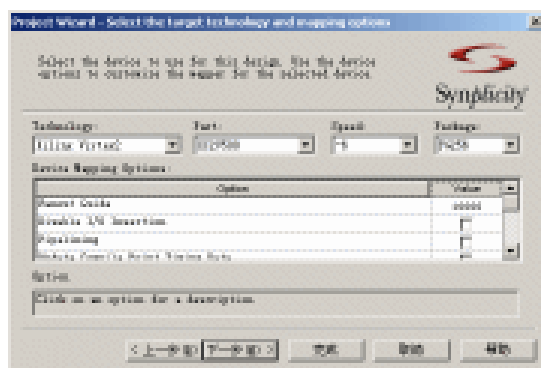


图8-25 设置器件类型与优化参数

- (4) 单击 **下一步(N) >** 按钮，选择默认综合优化参数，单击 **完成** 按钮完成工程的创建。在工程管理窗中将这个设计的顶层模块 top.v 用鼠标拖到最下面，Synplify Pro 默认最后一个源文件为顶层模块，如图 8-26 所示。

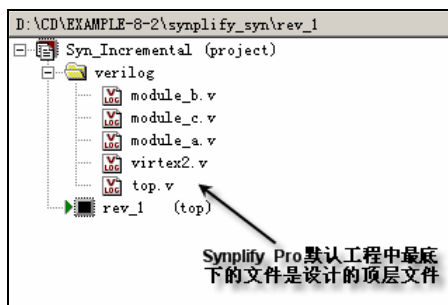


图8-26 设置顶层模块

- (5) 选择【Run】/【Compile Only】命令对工程进行编译。
- (6) 单击 按钮，启动 SCOPE，单击 按钮，进入 SCOPE 主界面。在约束属性（Attributes）选项卡中可以使用“xc_area_group”进行 Xilinx 物理区域约束，这个约束有两种常用格式，可以通过 Slice 或 CLB 的位置标明区域分组的范围。需要强调的是，这个约束在 Synplify Pro 综合中并不直接起作用，而是通过综合约束文件（SDC）透明传输至 Xilinx NCF 约束文件中，在布局布线过程中起作用。xc_area_group 在 SDC 文件中的语法如下：

通过 Slice 定义分组区域 (VirtexII and VirtexII-Pro)

```
define_attribute {v:work.module} xc_area_group {SLICE_X0Y0:SLICE_X9Y9}
```

Defined by CLB Row/Column (Virtex and VirtexE)

```
define_attribute {v:work.module} xc_area_group {CLB_R10C29:CLB_R16C35}
```

由于使用 Slice 或者 CLB 的坐标位置标明区域分组不如图形化界面直接，所以这里我们并不使用 xc_area_group 做任何区域分组约束，而是在 PACE 的图形界面下完成等同的功能。Synplify Pro 推荐将所有的约束前提到综合步骤完成，并支持将实现步骤的约束（管脚、时序、区域等）属性反标到 SDC 文件中。

- (7) 在 Compile Points 约束选项卡设置 Logic Group module_a、module_b、module_c 为增量综合的 Compile Point，如图 8-27 所示。

	Enabled	Module	Type
1	☒	module_a	locked
2	☒	module_b	locked
3	☒	module_c	locked
4	☒		
5	☒	在 Compile Points 选项卡中设置锁定所有 Logic Group	
6	☒		

图8-27 设置 Logic Group 为 Compile Point



SCOPE 中所有约束选项卡的含义参见本书 4.1.3.4 节。

- (8) 通常需要按照本书第 4 章 4.1.3.4 小节的论述添加那些必选项目（如 Clocks 等约束选项卡）的综合约束并对每个 Logic Group 单独做综合约束，本例出于论述方便，省略了这些步骤。单击 按钮保存约束文件，弹出保存对话框，输

入约束文件名（扩展名为 sdc），单击 **保存(S)** 按钮确认。这时会弹出对话框询问是否将约束文件加入到工程，单击 **是(Y)** 按钮确认，将新建约束文件加入到当前工程。退出 SCOPE，可以看到新建的约束已经加入到工程中。

- (9) 单击 Synplify Pro 主界面中的 **Run** 按钮，综合器自动综合，这时可以看到状态显示栏依次显示编译（Compiling），映射（Mapping）。当综合完成后，状态显示栏显示“Done”表示已完成综合过程。这时在综合工程目录下可以看到所有 Logic Group 的综合结果被分别存放在不同的目录下。
2. 在 ISE 中创建工程。
- (1) 将综合输出的逻辑网表 top.edf 和传递的约束文件 top.ncf 拷贝到实现目录下，启动 ISE，选择【File】/【New Project】命令创建工程。选择设计流程为 EDIF 流程，如图 8-28 所示。

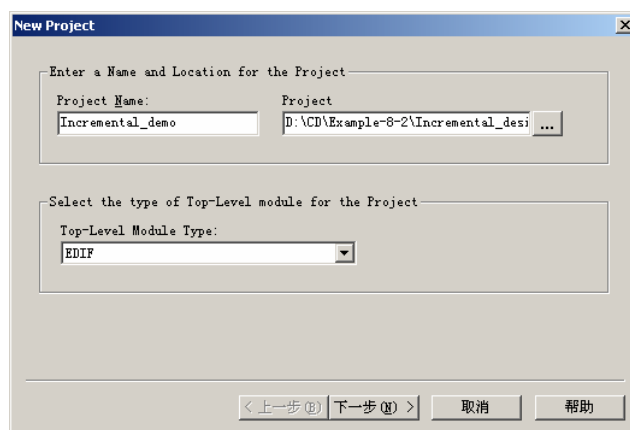


图8-28 选择新建工程的设计流程为 EDIF 模式

- (2) 单击 **下一步(N) >** 按钮，按图 8-29 所示指定综合输出的逻辑网表 top.edf 和传递的约束文件 top.ncf。

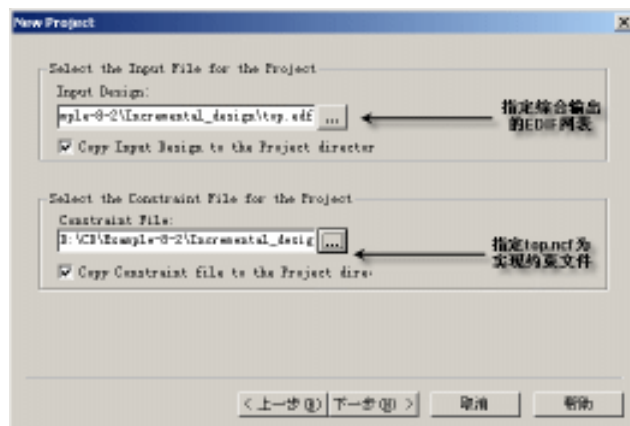


图8-29 指定 EDIF 网表和 ncf 约束文件

- (3) 单击 **下一步(N) >** 按钮，按图 8-30 所示设置工程属性。

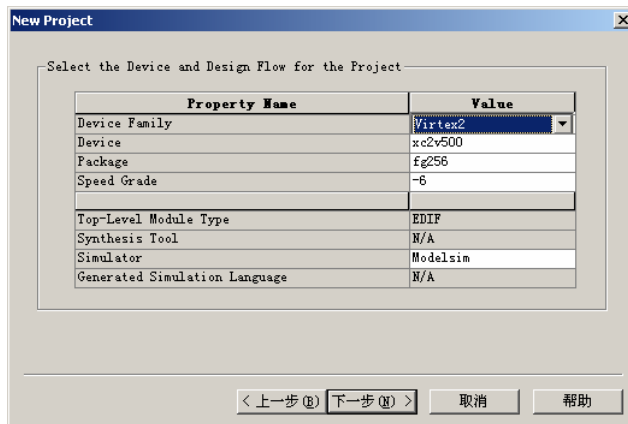


图8-30 设置工程属性

- (4) 单击“下一步(N) >”按钮，确认工程建立信息后，单击“完成”按钮创立新工程。
3. 使用 PACE 规划区域分组。
- (1) 在资源操作窗中，展开用户约束步骤，如图 8-31 所示。双击“Creat Area Constraints”启动 PACE，创建区域约束。

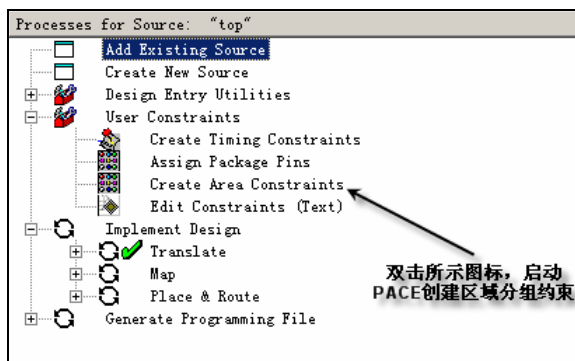


图8-31 启动 PACE 创建区域约束

- (2) 在 PACE 用户界面中单击图 8-32 所示图标，进入“Assign Area Constraints Mode”。



图8-32 进入区域约束模式

- (3) 在设计浏览 (Design Browser) 窗口展开逻辑 (Logic) 项目，选中某个 Logic Group，然后在中间的结构视图 (Architectute View) 页面规划位置分组区域。区域规划后可以选择彩色高亮标注，如图 8-33 所示。

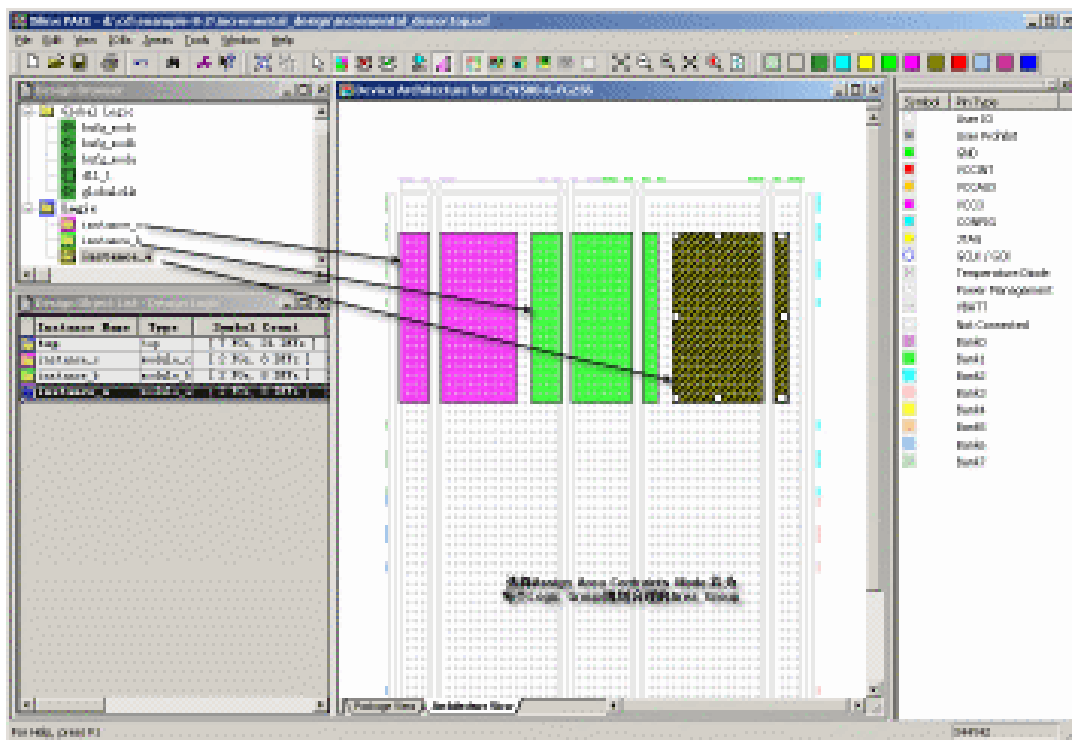


图8-33 规划位置区域约束

- (4) 所有的 Area Group 规划好后，保存 UCF 文件，退出 PACE。此时 UCF 文件应该有许多用“AREA_GROUP”关键字描述的区域分组，如下文所示：

```
#PACE: Start of Constraints generated by PACE
```

```
#PACE: Start of PACE I/O Pin Assignments
```

```
#PACE: Start of PACE Area Constraints
```

```
AREA_GROUP "AG_instance_a" RANGE = SLICE_X32Y59:SLICE_X45Y42 ;
```

```
INST "instance_a" AREA_GROUP = "AG_instance_a" ;
```

```
AREA_GROUP "AG_instance_b" RANGE = SLICE_X16Y59:SLICE_X29Y42 ;
```

```
INST "instance_b" AREA_GROUP = "AG_instance_b" ;
```

```
AREA_GROUP "AG_instance_c" RANGE = SLICE_X0Y59:SLICE_X13Y42 ;
```

```
INST "instance_c" AREA_GROUP = "AG_instance_c" ;
```

```
#PACE: Start of PACE Prohibit Constraints
```

```
#PACE: End of Constraints generated by PACE
```

4. 生成初始 Guide 文件。

- (1) 在 ISE 中完成实现流程，如图 8-34 所示。

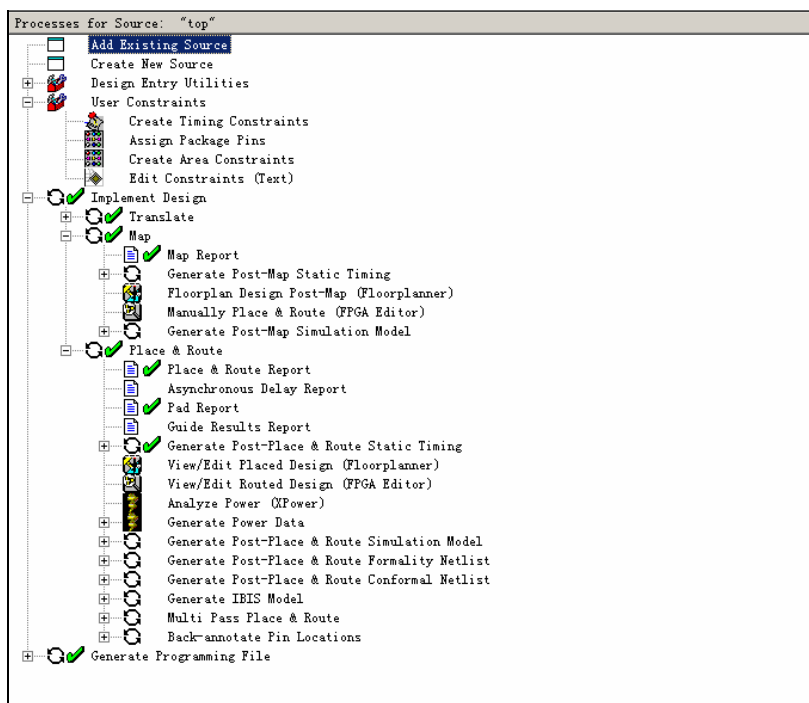


图8-34 在 ISE 中完成实现流程

- (2) 如本章 8.2.6 节所述检查 Map 和 PAR 步骤的报告文件，如果区域分组约束和每个 Logic Group 的时序约束都已经满足，则将 Map 的结果文件“top_map.ngm”和“top_map.ncd”与 P&R 输出的结果文件“top.ncd”重新命名为“top_map_guide.ngm”、“top_map_guide.ncd”和“top_guide.ncd”作为增量实现的 Guide 文件。重新命名的作用在于防止实现时覆盖掉 Guide 文件。
5. 修改 Logic Group 中的部分代码，并再次进行增量综合。
 - (1) 如果在调试过程发现某个 Logic Group 需要修改，比如发现 module_b.v 中的第 24 行代码写错了。
`assign OR4_OUT = Q0_OUT || Q1_OUT || Q2_OUT || Q3_OUT ;`
 需要修改为：
`assign OR4_OUT = !(Q0_OUT || Q1_OUT || Q2_OUT || Q3_OUT) ;`
 则可以运用增量设计方法以继承其他 Logic Group 的实现结果，并节约综合、实现耗时。
 - (2) 打开 Synplify Pro 中的工程，修改 module_b.v 中的代码，保存结果。其他任何约束条件不变，单击  按钮，进行增量综合。此时我们可以看到 Synplify Pro 仅仅对 Logic Group module_b 重新编译，而 module_a 和 module_c 没有做任何重新的综合与优化。



可以如第 4 章所述用 RTL View 或 Technology View 对比综合结果，也可以直接分析 EDIF 网表对比修改前后的综合结果，以加深对增量综合效果的体会。

6. 使用 Guide 文件进行增量实现。

- (1) 将增量实现更新的 top.edf 逻辑网表拷贝到增量实现目录下。并将前面生成的“top_map_guide.ngm”、“top_map_guide.ncd”和“top_guide.ncd”等 3 个 Guide 文件拷贝到增量实现目录中。
- (2) 使用 Guide 文件进行增量 Map。如图 8-35 所示，打开 ISE 中资源操作窗中 MAP 步骤的属性对话框，选择“MAP Guide Mode”为“Incremental”，然后指定指引文件“top_map_guide.ncd”的路径。单击 **确定** 按钮，保存 Map 属性设置。进行完 Map 步骤后需要查看 Map 结果报告，如果区域分组、Logic Group 时序、Logic Group 资源利用率等方面都没有问题，则标志着增量映射已经成功。

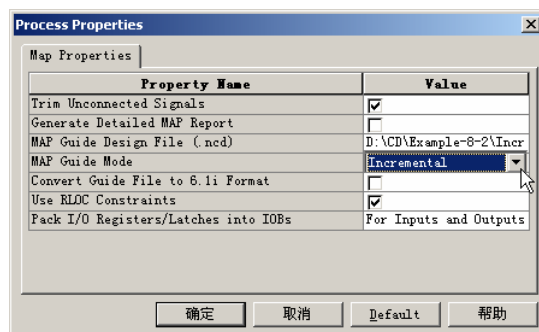


图8-35 ISE 中设置 Map 步骤 Guide 文件

- (3) 使用 Guide 文件进行增量 PAR。如图 8-36 所示，打开 ISE 中资源操作窗中 PAR 步骤的属性对话框，选择“PAR Guide Mode”为“Incremental”，然后指定指引文件“top_guide.ncd”的路径。单击 **确定** 按钮，保存 PAR 属性设置。查看 PAR 结果报告，如果区域分组、Logic Group 时序、Logic Group 资源利用率等方面都没有问题，则标志着整个增量实现过程已经成功。

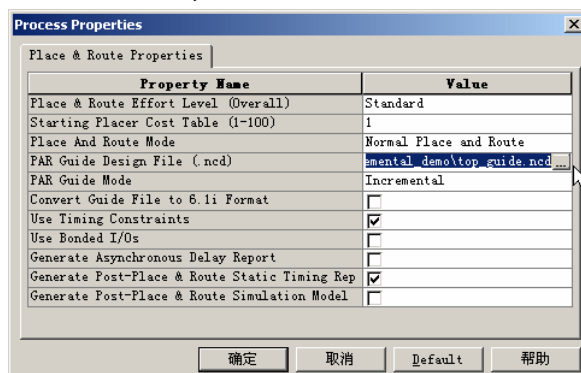


图8-36 ISE 中设置 Map 步骤 Guide 文件

至此，整个增量设计过程圆满完成。如果用户在某个实现步骤发生错误，或者通过报告观察到区域分组约束不合理，则需要返回到第 3 个操作步骤重新规划区域分组约束。

8.8 小结

本章前半部分重点介绍了 Modular Design 设计方法，后半部分介绍了 Incremental

Design 设计方法。

学习 Modular Design 除了要重点掌握 Modular Design 实现步骤的 3 个阶段的含义和操作方法，更重要的是要领会目前大型复杂系统推荐的设计方式——Modular Design 的设计思想，认识到 Modular Design 的核心精神是团队协作、并行设计。合理的使用 Modular Design 设计方法不仅能够缩短项目开发周期，提高设计效率，还能提高设计的可靠性与稳定性。

Incremental Design 的主要作用是缩短综合、实现的耗时，继承以往的实现成果。读者首先要掌握何种设计适合使用 Incremental Design，学习 Incremental Design 操作方法的关键是要掌握增量综合和增量实现两个步骤。

学习 Incremental Design 设计方法，首先要掌握增量式设计方法的应用场合和应用合理性要求，学习 Incremental Design 操作方法的关键是要掌握增量综合和增量实现两个步骤。合理使用 Incremental Design 能有效地缩短综合、实现耗时并继承以往的实现成果。

通过本章学习读者也许感到 Modular Design 和 Incremental Design 有许多相似之处。为了方便读者掌握，这里集中将两者加以比较。

Modular Design 与 Incremental Design 的区别如下：

- 两者应用场合不同。前者主要应用于大型复杂系统的团队协同设计，后者主要应用于单兵作战，特别是进行后期仿真与调试时应用。
- 两者侧重点不同。前者侧重用并行设计与协作分工的设计思想，后者侧重于在节约综合、实现的时间，继承以往的设计成果。
- 两者复杂度不同。无论从工具使用还是开发流程上看，前者的复杂度都明显高于后者。
- 两者所需约束类型不同。完成前者所需的约束主要有两个方面：区域分组约束和模块的时序约束，完成后者流程仅仅需要区域分组约束。
- 两者实现步骤不同。前者每个子模块都是并行设计的，每个子模块独立完成综合实现流程，并在最后合并阶段将每个子模块提交的 PIM 合并起来填充到顶层设计中去，并且整个设计流程中，每个子模块都需要进行两次实现与约束。后者整个实现流程一次完成，每个子模块只需经历一次实现与约束步骤。
- 两者的目录管理不同。前者非常强调目录管理，有严格的目录管理体制，后者只需遵循一般性目录管理方法即可。

Modular Design 与 Incremental Design 的联系如下：

- 两者都属于 ISE 高级设计方式。特别需要注意应用场合，合理的应用事半功倍，不合理的应用事倍功半。强调这点的意义在于，笔者发现很多初学者不论工程实际情况、复杂度、代码风格、工具熟练水平，希望在所有情况下都使用这些高级设计技巧，结果常常适得其反。
- 两者都需要对子模块做区域位置约束。区域分组的大小、位置、边界、包含的可编程资源的种类和数量等因素都直接影响着区域分组的效能，决定 Modular Design 和 Incremental Design 的最后结果。区域分组包含的 FPGA 资源（Slice、Block RAM、TBUF、Multipliers 和 I/O 相关资源等）应该与逻辑模块所使用的资源一致。两者做区域约束的工具一致，都可以使用 PACE 或 floorplanner，但是前者需要附加的约束种类更多、更细，后者需要附加的约束

较为简单。

- 两者的报告查看方法相似。两者完成实现步骤后都必须检查映射报告和布局布线报告，阅读报告的注意事项相似。

8.9 问题与思考

1. 模块化设计方法的优点有哪些？
2. Modular Design 实现步骤的 3 个阶段是什么，这 3 个阶段的主要任务分别是什么？
3. 论述 Modular Design 设计与 Incremental Design 设计方法的适用场合。
4. 如何使用 XFLOW 工具自动完成 Modular Design 设计流程？
5. 在进行 Modular Design 时需要使用的设计工具和命令有哪些？
6. Incremental Design 的应用场合和合理性要求主要是哪几方面？
7. Incremental Design 的实现流程有哪几个步骤？
8. Modular Design 与 Incremental Design 的区别、联系是什么？

第9章 融会贯通——“运动计时表”设计

本章利用“运动计时表”设计实例总结前面各章内容，使读者加深对 FPGA/CPLD 设计流程的理解，体会 ISE 集成的各种设计工具的使用方法与技巧。

本章内容主要围绕着以下几个问题展开：

- FPGA/CPLD 的完整设计流程是什么？这个问题是本书第 1 章的重点内容之一。完整的 FPGA/CPLD 设计流程包括电路设计与输入、功能仿真、综合、综合后仿真、实现、布线后仿真和下板调试等主要步骤。
- 在 ISE 中如何进行多元混合输入设计？本书第 2 章详细介绍了 ISE 支持的各种设计输入方法，那么如何统一这些设计输入资源，进行多元混合设计输入呢？在 2.4 节曾经提出：ECS 可以将 Verilog、VHDL 源代码，IP 核生成器（Core Generator）产生的 IP Core，结构向导（Architecture Wizard）生成的数字时钟管理模块（DCM），StateCAD 设计的状态机等多元混合设计输入方式有机地结合起来，完成复杂逻辑设计。具体方法是：首先使用 ECS 将各种设计输入资源生成模块符号（Symbol），然后用 ECS 绘制设计的顶层原理图，在顶层原理图中调用所生成的各种输入资源对应的模块符号，从而达到组织各种不同设计资源进行混合设计的目的。
- 如何完成功能仿真和布线后仿真？仿真在 FPGA/CPLD 设计中非常重要，占据了相当多的设计任务量和工作时间。本书第 3 章重点讨论了目前业界流行的 HDL 仿真工具 ModelSim 的使用方法和技巧。另外本书 2.6 节还介绍了 ISE 集成的很有用的仿真激励设计工具——HDL Benchner 的使用方法。结合使用测试激励编辑器 HDL Benchner 和仿真器 ModelSim 可以满足 ISE 中绝大部分仿真需要。
- 如何高效地进行综合？这个问题正是本书第 4 章的讨论重点。在第 4 章，介绍了 3 种典型的综合工具。为了照顾没有 Synplify Pro 的用户，本例使用 XST 综合整个设计，做综合时序分析。另外，还将演示如何使用 Synplify Pro 进行综合优化，如何使用 RTL View 和 Technology View 分析设计电路与关键路径。
- 如何设计用户约束文件？这个问题是本书第 5 章的讨论重点。第 5 章在讨论了各种约束的基本含义的基础上重点讨论了用户约束文件的设计方法。可以说 ISE 的实现步骤的学习重点就是如何设计用户约束文件（UCF）。UCF 文件是设计者与布局布线器之间的沟通桥梁。好的 UCF 文件能使设计的实现效果达到面积与频率上最优，满足用户的设计需求。本例将演示如何使用约束编辑器（Constraints Editor）和引脚与区域约束编辑器（PACE）设计 UCF 文件。
- 如何分析并改进实现结果？这个问题是本书第 6 章的讨论重点。第 6 章重点介

绍了与实现相关的时序分析器 (Timing Analyzer)、FPGA 底层编辑器 (FPGA Editor)、芯片观察窗 (Chip Viewer) 和布局规划器 (Floorplanner) 等高级用户工具的特点与使用方法。本例将简单演示如何使用 Timing Analyzer、Floorplanner、FPGA Editor 和 XPower 等工具分析实现结果。

- 如何生成并下载 FPGA/CPLD 配置文件？这个问题是本书第 7 章的讨论重点。在 ISE 中 iMPACT 生成并下载 FPGA/CPLD、PROM 配置文件。本例简单演示 iMPACT 的使用方法和技巧。

通过这 7 个问题的分析，读者可以清晰地把握本书的脉络。本书各章内容实际就是按照 FPGA/CPLD 在 ISE 中的设计流程组织的。希望读者通过学习本例，加深对本书各章内容的理解，能够切实掌握 ISE 中 FPGA/CPLD 的设计方法与技巧。

9.1 示例背景

本例是一个“运动计时表”设计，利用 ECS 完成多元混合输入设计，其顶层为 ECS 设计的原理图，如图 9-1 所示。

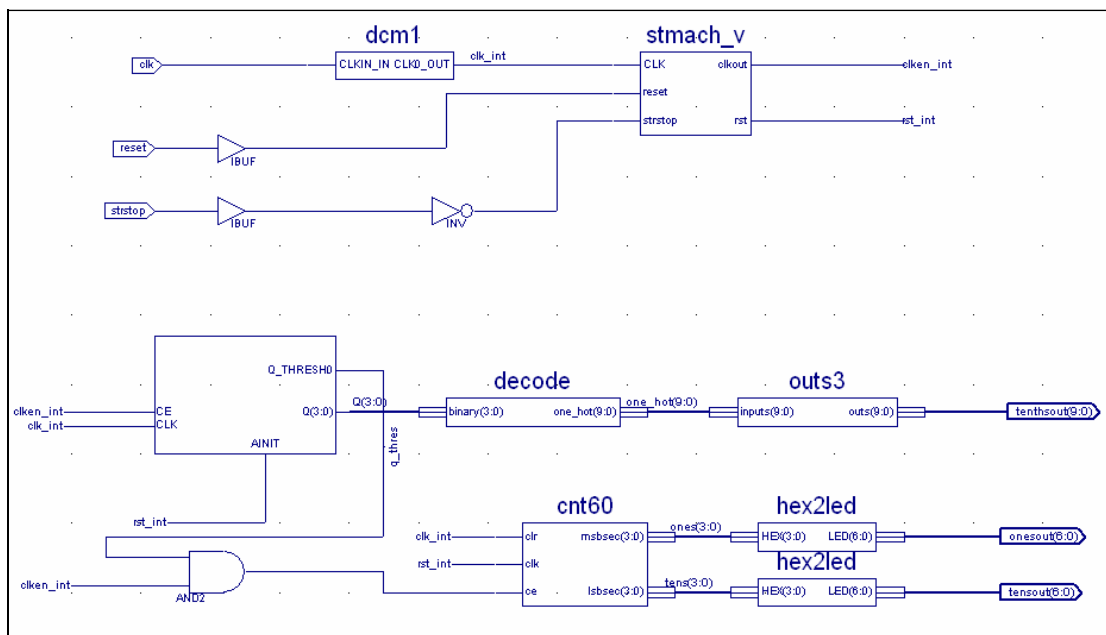


图9-1 “运动计时表”顶层原理图

设计有“strstop”、“reset”和“clk”等 3 个输入，具体含义如下。

- strstop：低有效信号，用以启动/停止表计时。其功能同实际计时表的启动/停止按钮相同。
- reset：当计时表停止后，使用该信号将计时复位为“0.00”。
- clk：本计时表的设计是同步时序设计，这个信号是整个设计的主时钟。

设计有“tensout (6:0)”、“onesout (6:0)”和“tenthout (9:0)”等 3 个输出，具体

含义如下。

- tensout (6:0) : 计时表“十位”计时结果的 7 段译码显示, 是“十位”数值的 7 段译码 LED 显示输入。
- onesout (6:0) : 计时表“个位”计时结果的 7 段译码显示, 是“个位”数值的 7 段译码 LED 显示输入。
- tenthsout (9:0) : 采用“独热”(One-Hot) 编码的秒表计时显示。

整个设计包含以下功能模块, 每个模块需要用户按照本章演示通过自己设计生成。另外在附带光盘中, 本章完整的示例工程有这些模块的源代码, 供读者参考。

- stmach_v: 使用 StateCAD 设计的计时表的状态机。
- cnt60: 使用 ECS 设计的秒表计数器, 该模块的两个 4bit 输出代表计时的个位与十位显示, 是 7 段译码器的输入。
- tenths: 使用 CORE Generator 生成的 4bit、二进制计数器。
- hex2led: HDL 语言实现的七段译码模块, 将 cnt60 的个位和十位 4bit 计时结果翻译为 7 段译码显示格式。
- outs3: 使用 ECS 原理图设计的输出模块, 因为 7 段译码器一般是低电平有效, 所以需要对输出进行电平转换。
- decode: 使用 HDL 语言设计的译码模块, 用以将 4bit 计时结果译码为 10bit “独热”(One-Hot) 编码。
- dcm1: 使用 Architecture Wizard 设计的 DCM 时钟管理模块, 其输出作为整个同步时序设计的全局时钟。
- stopwatch: “计时表”顶层原理图, 调用以上所有模块的器件符号, 实现多元混合输入设计。

下面利用 ISE 中集成的多种设计资源与工具, 按照 FPGA/CPLD 设计流程, 设计一个“运动计时表”。在本书配套光盘中, 有本章范例的文件介绍和操作指示。为了叙述方便, 请建立目录“ISE”, 并将工程源程序拷贝到本地硬盘下。在配套光盘本例工程目录下有 Verilog 和 VHDL 两种语言实现的源代码, 请读者根据个人习惯选用。

9.2 多元混合设计输入方法

本例的设计输入方式多种多样, 有 ECS 设计的原理图, StateCAD 设计的状态机, Architecture Wizard 设计的时钟管理 DCM 模块, Core Generator 生成的 IP Core 和 HDL 源代码。利用 ECS 把这些混合资源生成器件符号, 然后在顶层原理图中调用这些器件符号, 将多元混合设计有机地组织起来。

9.2.1 新建工程“watch_sc”

双击桌面上 ISE Project Navigator 的快捷图标 , 启动 ISE。选择菜单栏中的【File】/【New Project】命令, 弹出新建工程对话框, 新建工程“watch_sc”, 如图 9-2 所示。

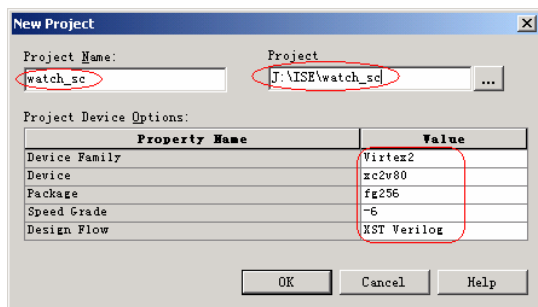


图9-2 新建工程“watch_sc”

根据习惯可选择 XST VHDL 或 XST Verilog 设计流程，单击 **OK** 按钮确认新建工程。

9.2.2 使用 ECS 绘制“cnt60”和“outs3”模块原理图

本节使用 ECS 设计两个子模块，一个是“cnt60”计数器，另一个是“outs3”输出模块。使用 ECS 设计原理图可以分为：新建原理图；选择、放置器件符号；添加连线；添加网线名称；设置 I/O 管脚；绘制总线；原理图检查；生成器件符号等步骤。

✎ 绘制“cnt60”和“outs3”模块原理图，参考示例详见本书附带光盘的“Example-9-1”目录

1. 新建原理图资源文件“cnt60”。

- (1) 单击新建资源快捷按钮 ，弹出新建资源对话框，选择新建资源类型为原理图（Schematic）类型，新建文件名为“cnt60”，并且直接添加到工程中去，如图 9-3 所示。

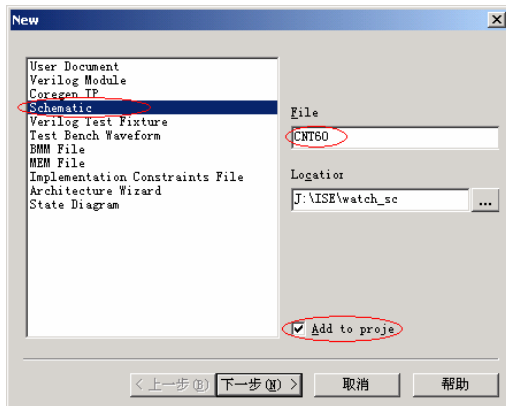


图9-3 新建原理图“cnt60”

- (2) 单击 **下一步(N) >** 按钮，单击 **完成** 按钮确认新建资源信息，启动 ECS。
2. 选择并放置器件符号。

“cnt60”是时间计数器，有两个 4bit 总线输出代表时间的个位与十位数值。它的原理图如图 9-4 所示。

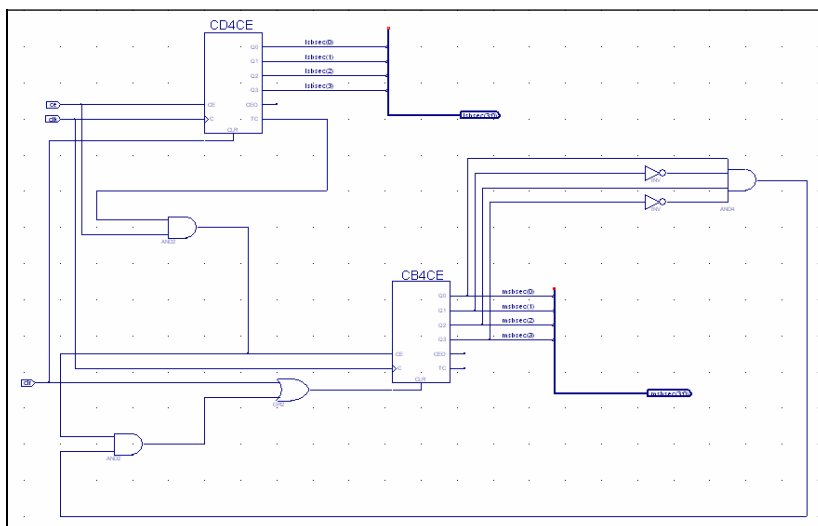


图9-4 时间计数器“cnt60”原理图

- (1) 单击绘图工具栏中的  按钮，在器件符号分类显示目录选择所有器件（All Symbols），在符号名称过滤框输入“cb4ce”，按 **[Enter]** 键并在器件符号详细列表点击“cb4ce”计数器符号，在原理图适当位置单击鼠标左键放置两个“cb4ce”计数器。
- (2) 根据同样方法，在原理图适当位置放置两输入与门、或门、四输入与门。
- (3) 按 **[Esc]** 键或者双击鼠标右键结束器件选择与放置，如图 9-5 所示。

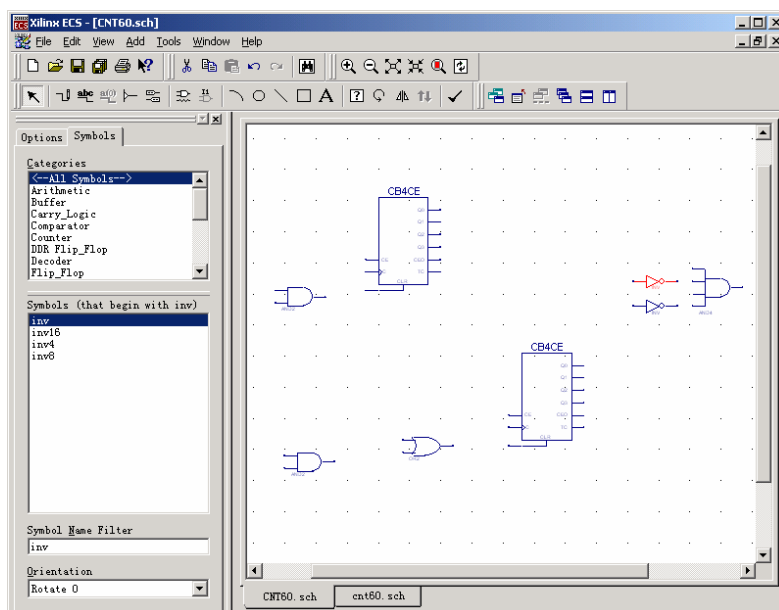


图9-5 在“cnt60”原理图中放置器件符号

3. 添加连线。
- (1) 单击绘图工具栏中的  按钮，选择智能连线方式，当鼠标指针移动到原理图

可画线区域时，鼠标符号自动变为“+”形状，点击始点管脚两次（第1次选择器件，第2次开始连线），然后点击连线终点管脚，智能连线器自动在起始与终点管脚之间布置一条连线。

- (2) 按 **Esc** 键或者双击鼠标右键结束操作，如图 9-6 所示。

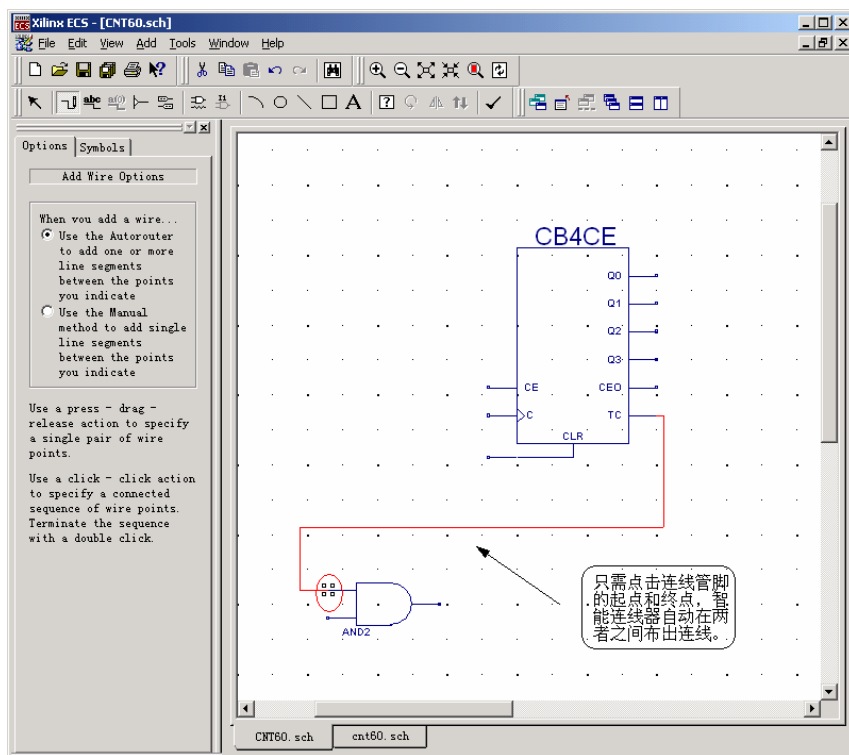


图9-6 在“cnt60”原理图中添加连线

- (3) 依照原理图 9-4 所示连接其他连线，并在器件端口拉出一小段线段为添加网线名称与 I/O 管脚做准备。

4. 添加网线名称。

添加网线名称有两个用途：一是增强了原理图的可读性，在综合与实现时也易于查看某条网线的时延与布线情况；另外，ECS 如大多数原理图编辑工具一样，默认同名网线就是一条网线。换句话说，如果两条网线名称相同，即使在原理图上没有明确连接，ECS 也认为它们是同一条网线，有了连接关系。

- (1) 单击绘图工具栏中的 按钮，选择网线分支命名 (Name the Branch)，在网线名称栏填入“ce”作为“cb4ce”的使能输入管脚名称，当鼠标指针移动到原理图中时，发现鼠标指针处已有了“ce”网线名称。在“cb4ce”的“CE”引线末端单击鼠标左键，为该网线添加名称。如果单击网线名称栏旁边的 按钮，网线名称序号自动加 1 或者减 1。
- (2) 按同样的方法为其他网线添加名称，如图 9-7 所示。

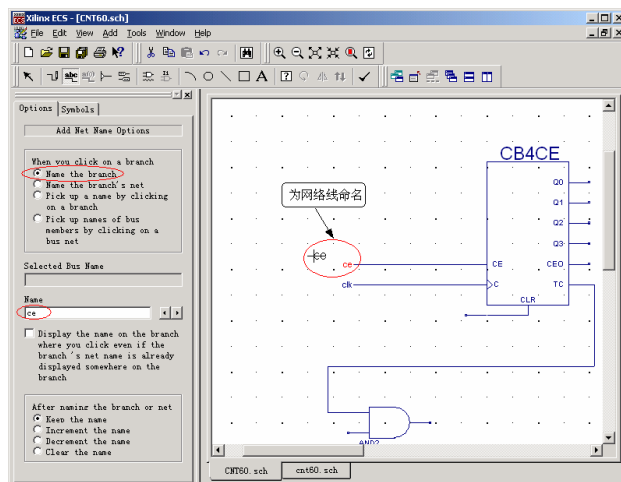


图9-7 为网线添加名称

- (3) 按 **Esc** 键或者双击鼠标右键结束操作。
5. 设置 I/O 管脚。
- (1) 为已经命名的网线设置 I/O 端口管脚比较简便，单击绘图工具栏中的  按钮，选择添加输入管脚（Add an input marker），然后在原理图需要添加 I/O 管脚的网线端口，按住鼠标左键拖出一个框，放开鼠标左键，自动产生一个与网线同名的输出端口。
- (2) 在设置选项卡选择添加输出管脚（Add an output marker），与添加输入管脚的方法相同，添加输出管脚。
- (3) 按 **Esc** 键或者双击鼠标右键结束操作，结果如图 9-8 所示。

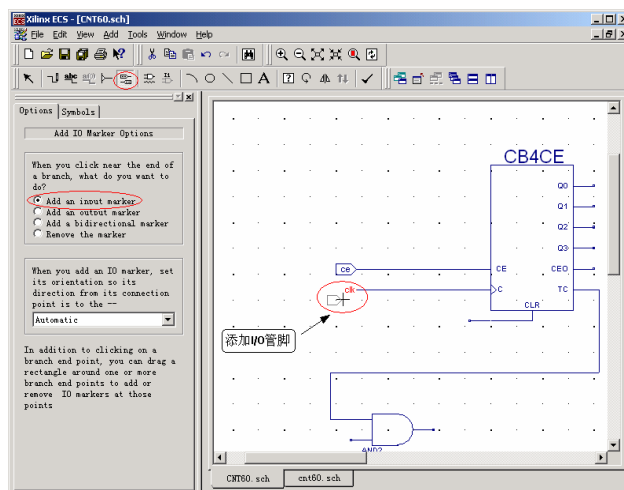


图9-8 添加 I/O 管脚

6. 绘制总线。

ECS 中并没有专用的总线绘制命令，绘制总线时仍然采用绘制连线命令【Add Wire】，仅仅利用网线命名区分总线与普通连线。总线的名称命名格式为 BusName (X:Y)，其中

BusName 是总线名称,“()”为总线表示符号,X 为 MSB,Y 为 LSB。在“cnt60”计数器中,“cb4ce”的寄存器输出($Q_3Q_2Q_1Q_0$)组成七段译码器的输入总线。绘制总线的完整过程分为3步:绘制添加总线名称;添加总线分支;命名总线分支信号。

- (1) 单击绘图工具栏中的按钮,在原理图上方绘制一条线段,然后单击按钮,在网线名称栏输入“lsbsec(3:0)”。在线段右端口添加总线名称,总线名称一旦加入,就会发现网线变粗,表示该网线变成了总线,如图9-9所示。

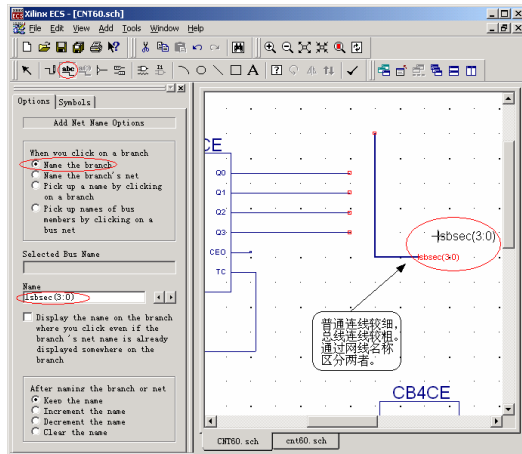


图9-9 添加总线名称

- (2) 单击按钮,为总线添加总线分支。总线分支表示总线和单根网线之间的连接转换。适当旋转总线分支方向,使总线分支的三角形连接在总线上。
- (3) 接着单击绘图工具栏中的按钮,绘制连线连接总线分支与单根信号线,如图9-10所示。

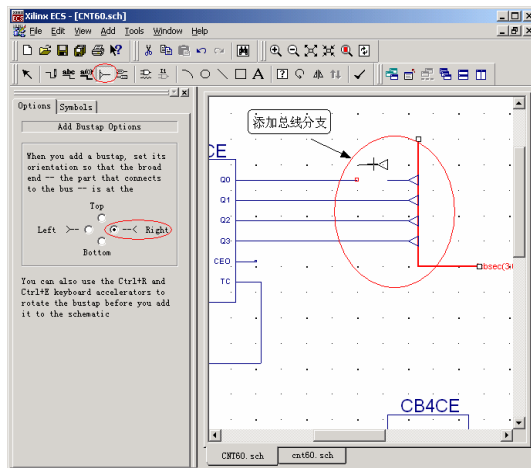


图9-10 添加总线分支

- (4) 最后为总线的信号添加网线名称(参见图9-11)。总线分支信号网线名称格式为“BusName (a)”,其中“BusName”为总线名称,“a”为在总线 MSB 与

LSB 之间的一个阿拉伯数字，表示信号线标号。要注意的是：总线分支信号的名称一定要与总线信号名称一致，否则会报错。

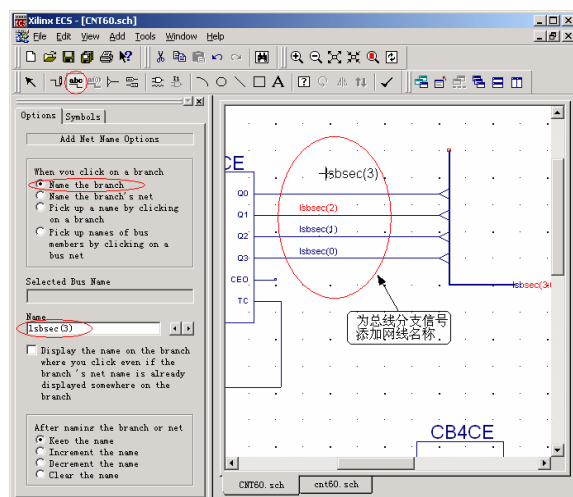


图9-11 为总线分支信号添加网线名称

7. 运用步骤 1~6 的操作方法，按照图 9-4 所示的“cnt60”原理图完成其余部分设计，单击 按钮保存全部设计。

原理图绘制完成后，还必须检查错误。单击绘图工具栏中的 按钮，ECS 自动检查当前原理图。如果原理图完全正确，则在检错对话框中显示“No errors detected”信息。如果原理图有错，检错对话框内将罗列所有错误，使用居中（Center）、放大（Zoom In）和缩小（Zoom Out）工具可以辅助设计者迅速定位错误位置。

8. 至此，“cnt60”计数器的原理图已经完成，保存设计，退出 ECS。
9. 在 Project Navigator 的资源管理窗选中“cnt60”原理图资源，展开当前资源操作窗的“Design Entry Utilities”项目，双击【Create Schematic Symbol】命令生成“cnt60”的模块符号作为顶层原理图的用户模块符号，如图 9-12 所示。

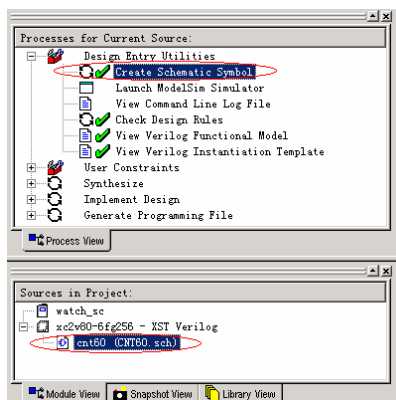


图9-12 生成用户模块符号

10. 重复前面的设计方法，完成“outs3”模块的原理图，并生成模块符号。

“outs3”是一个转换模块，其主要功能是将输入电平反向输出，原理图如图9-13所示。

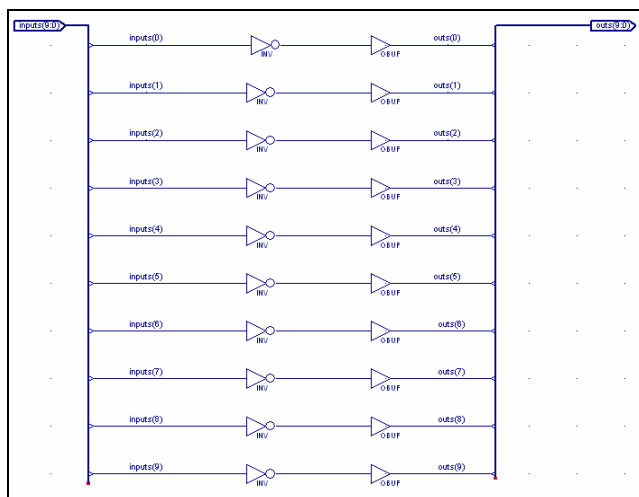


图9-13 “outs3”输出模块原理图

9.2.3 使用 Core Generator 生成“tenths”IP 核

本节使用 ISE 中集成的实用 IP Core 生成工具——Core Generator，生成一个名为“tenths”的4bit、2进制计数器。

✎ 使用 Core Generator 生成“tenths”IP 核，参考示例详见本书附带光盘的“Example-9-1”目录

1. 新建 Coregen IP 类型资源。

(1) 单击新建资源快捷按钮 ，弹出新建资源对话框，选择新建资源类型为“Coregen IP”类型，新建文件名为“tenths”，并且直接添加到工程中，如图9-14所示。

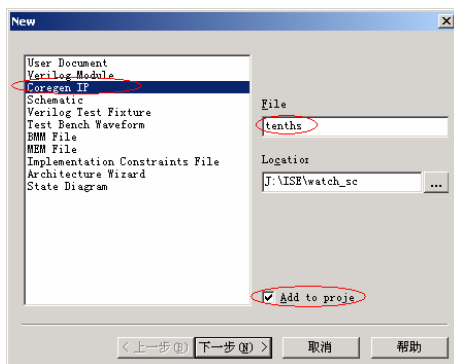


图9-14 新建“Coregen IP”类型资源

(2) 单击  按钮，然后单击  按钮确认新建资源信息，启动 Core

Generator。采用新建 IP Core 资源方式启动 Core Generator，器件族、设计流程类型等基本工程属性信息自动导入 Core Generator 中，不需另外更改 Core Generator 工程属性。

2. 选择适用的 IP 核。

- (1) 在 IP Core 功能目录中选择“基本单元（Basic Elements）”，“计数器（Counters）”。左边的详细信息窗中会显示二进制计数器（Binary Counter）IP Core 项目如图 9-15 所示。

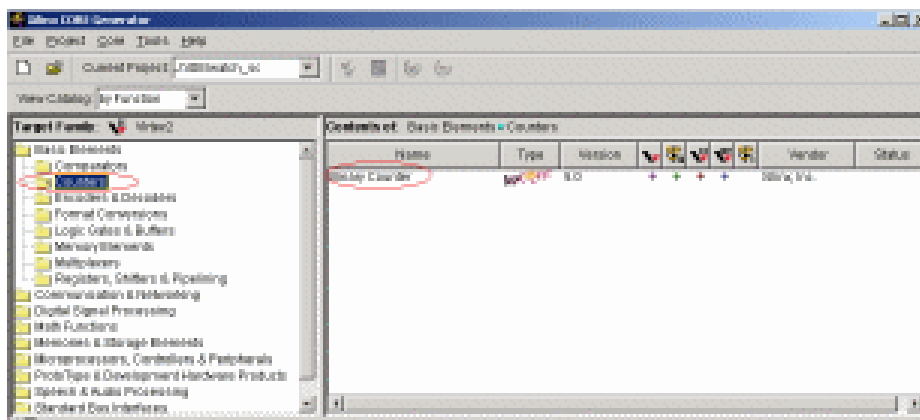


图9-15 选择“Coregen IP”类型资源

- (2) 双击二进制计数器（Binary Counter）项目，打开 IP 核参数设置窗，然后单击 **Data Sheet...** 按钮，启动该 IP Core 的 PDF 器件说明文件。根据器件说明判断这个 IP Core 恰好满足设计需要。

3. IP 核的参数设计。

- (1) 在 IP 核参数设置窗，如图 9-16 所示，设置计数器参数。

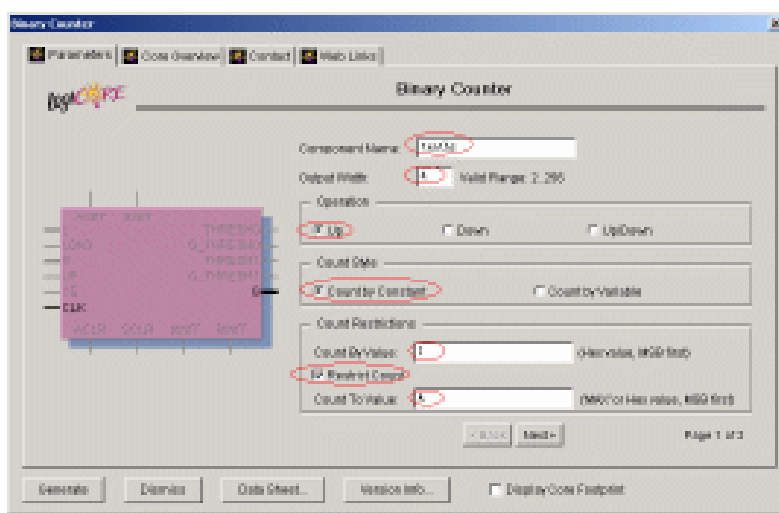


图9-16 设置计数器参数之一

- (2) 由于 IP 的设置参数项目很多，单击 **Next>** 按钮进入下几项参数的设置，如图 9-17 所示。

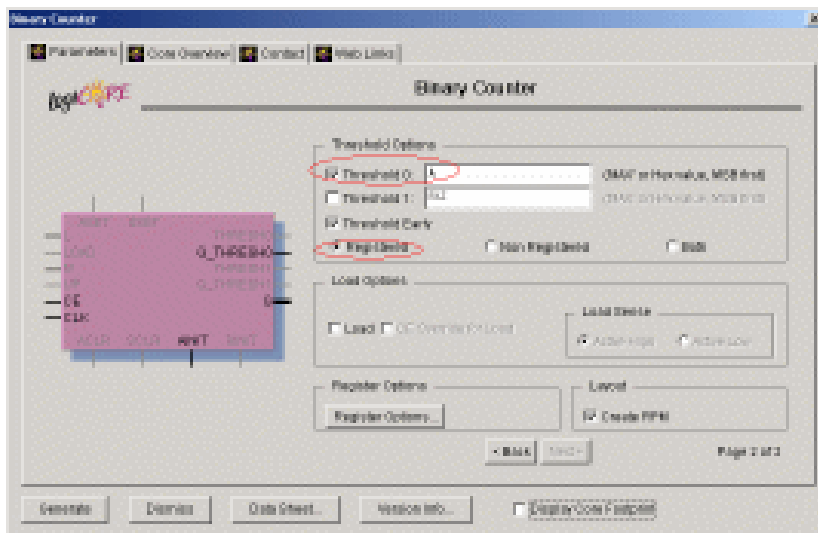


图9-17 设置计数器参数之二

- (3) 单击 **Register Options...** 按钮，设置寄存器参数，如图 9-18 所示。

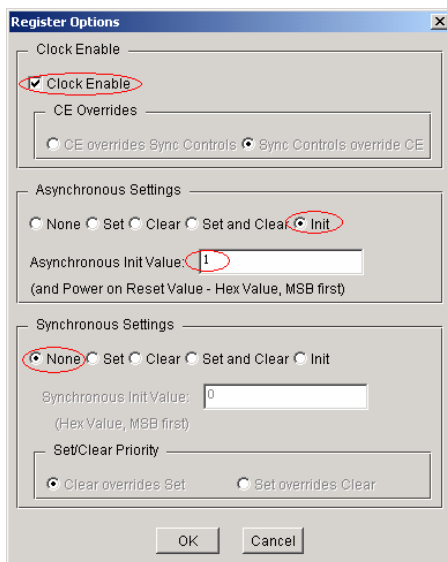


图9-18 设置寄存器参数

- (4) 设置完成后，单击 **Generate** 按钮，立即生成二进制计数器“tenths”。
- (5) 退出 Core Generator，可以看到在 Project Navigator 的资源管理窗中已经产生 IP Core 资源符号（.xco 文件），如图 9-19 所示。Xilinx IP Core 生成器生成的 IP Core 不用另外生成原理图器件符号，在原理图中可以直接调用。

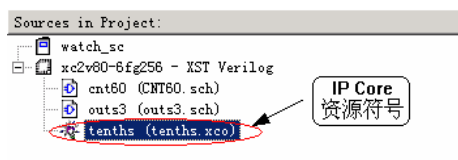


图9-19 IP Core 资源已经加入到工程

9.2.4 使用 StateCAD 设计 “stmach_v” 状态机

本书 2.3 节已经重点讨论过 StateCAD 设计状态机的方法，下面利用 StateCAD 设计这个“运动计时表”的状态机“stmach_v”模块。该模块的状态机如图 9-20 所示，由“CLEAR”、“zero”、“start”、“counting”、“stop”和“stopped”等 6 个状态构成，通过判断条件“strstop”进行状态转移。

使用 StateCAD 设计状态机的基本步骤是：新建 StateCAD 资源；利用状态机设计向导生成状态转移图模板；编辑状态；编辑状态转移条件；启动优化向导，优化设计；编译设计生成 HDL 代码等。

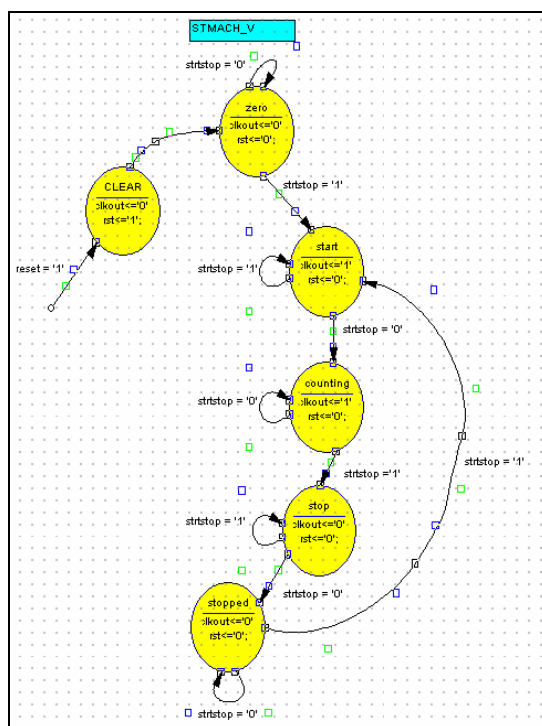


图9-20 “stmach_v” 状态转移图

🔑 使用 StateCAD 设计 “stmach_v” 状态机，参考示例详见本书附带光盘的“Example-9-1”目录

1. 新建 StateCAD 资源。

- (1) 单击新建资源快捷按钮，弹出新建资源对话框，选择新建资源类型为“Schematic”类型，新建文件名为“stmach_v”，并且直接添加到工程中去，如图 9-21 所示。

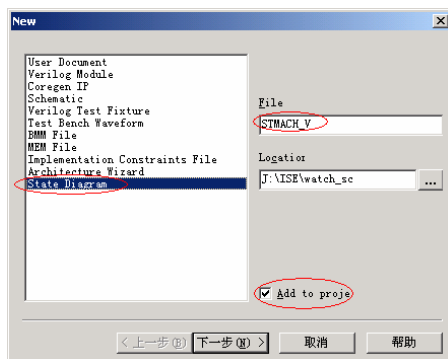


图9-21 新建状态机资源“stmach_v”

- (2) 单击按钮，单击按钮确认新建资源信息，启动 StateCAD。
2. 利用状态机设计向导生成状态转移图模板。

StateCAD 设计状态机的流程实际就是从左至右依次使用操作工具栏按钮的过程。

- (1) 单击按钮，启动状态机设计向导，设计状态转移图辅助模板。状态机设计向导第 1 个对话框的功能是选择状态转移图的类型和状态的数目。从图 9-20 所示的状态转移图可以看出本设计分为 6 个状态，状态转移图的类型为列状转移图，如图 9-22 所示。

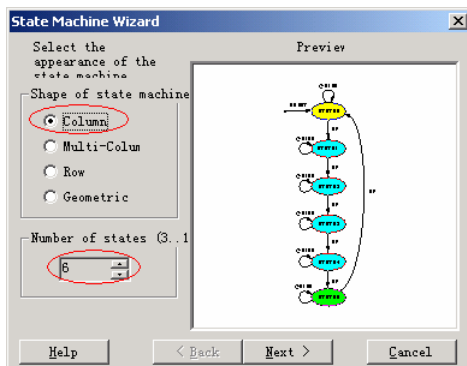


图9-22 设置转移图类型和状态数

- (2) 单击按钮进入状态机设计向导的第 2 个界面，设计状态机的复位机制。复位类型分为同步复位和异步复位两种方式。这里选择异步复位 (Asynchronous) 方式。
- (3) 单击按钮进入状态机设计向导的第 3 个界面，设置状态转移方式。状态转移方式分为自循环、下行和上行 3 种方式。由本设计的状态转移图可以看出，应当选择带有自循环的下行方式。注意到本设计的所有转移条件都和“strstop”信号相关，为了方便绘图，事先在转移条件上填入“strstop =

‘0’”，如图 9-23 所示。

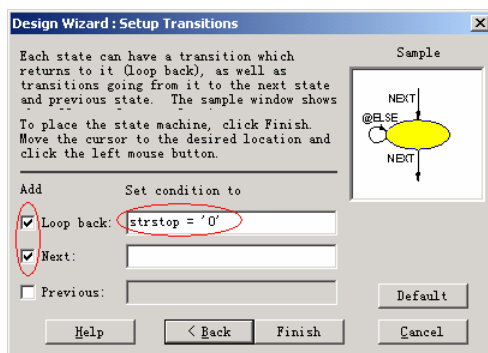


图9-23 设置状态转移方式

- (4) 单击 **Finish** 按钮完成状态机向导设置，然后在绘图区居中的位置单击鼠标左键，将状态转移图模板放置在绘图窗中，如图 9-24 所示。

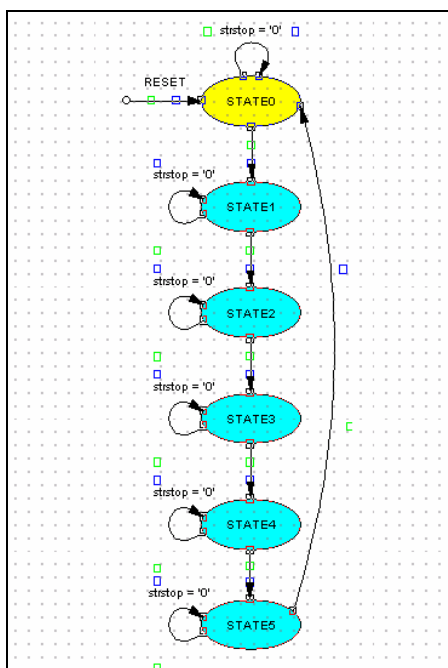


图9-24 状态转移图模板

3. 修改状态转移图模板。

状态转移图模板仅仅是一个通用的状态图形式，为了满足用户的具体设计需求，还必须做适当的修改。对比图 9-20 与图 9-24，发现需要进行两处修改：将状态“STATE0”的自循环条件删除；将状态“STATE5”的转移线的到达状态改为“STATE2”。第 1 项修改的方法：单击绘图工具栏中的  按钮，选择状态“STATE0”的自循环条件，单击鼠标右键，弹出命令菜单，选择【Delete】命令删除该状态转移线即可。第 2 项修改的方法：单击绘图工具栏中的  按钮，选择自“STATE5”状态出发、到达“STATE0”状态的转移条件的末端箭

头，用鼠标拖到状态“STATE2”，操作时必须注意一定要与状态“STATE2”相连接。

修改后的状态转移图已经与图 9-20 所示的状态转移图等效了。读者还可以调整各个状态的位置，使整个图的形状与原理图中的形状更相似。通过设计、修改状态图模板应该可以体会到，使用状态转移图模板辅助绘制状态图是一个省时省力的好方法。修改后的状态转移图如图 9-25 所示。

4. 编辑状态。

为了方便读图，下面依次将“STATE0”到“STATE5”6 个状态改为“CLEAR”、“zero”、“start”、“counting”、“stop”、“stopped”等 6 个状态，并添加状态逻辑。

- (1) 双击“STATE0”状态，进入状态编辑窗，在状态编辑窗更改状态的名称为“CLEAR”。单击 **Output Wizard** 按钮，打开逻辑向导，按照图 9-26 所示添加状态逻辑“clkout=‘0’”，单击 **OK** 按钮确认设计；单击 **Output Wizard** 按钮，按照图 9-27 所示添加状态逻辑“rst=‘1’”，单击 **OK** 按钮确认设计。

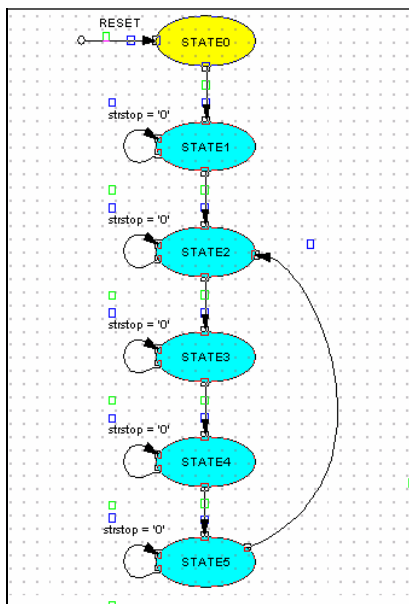


图9-25 修改后的状态转移图

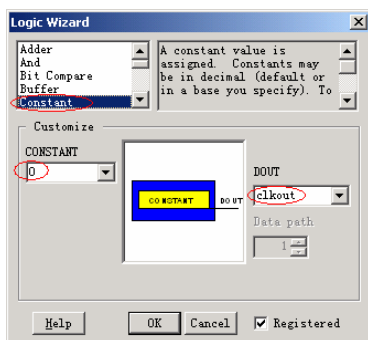


图9-26 添加状态逻辑“clkout=‘0’”

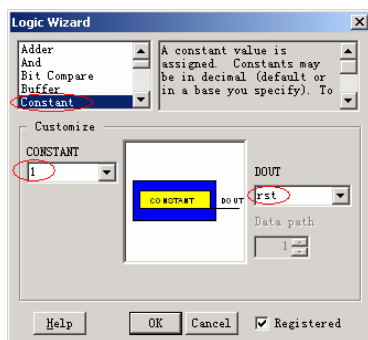


图 9-27 添加状态逻辑“rst=‘1’”

- (2) 单击 **OK** 按钮确认逻辑向导添加的状态输出逻辑，回到状态编辑窗，看到输出逻辑 “clkout= ‘ 0 ’, rst= ‘ 1 ’” 已经添加到状态 “CLEAR” 中。将阻塞赋值符号 “=” 修改为非阻塞赋值符号 “<=”，更改输出逻辑为 “clkout<= ‘ 0 ’, rst<= ‘ 1 ’”，如图 9-28 所示。

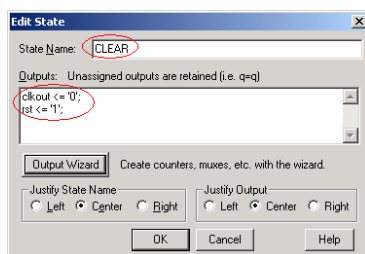


图9-28 状态编辑窗

- (3) 更改状态名 “STATE1” 为 “zero”，并添加输出逻辑 “clkout<= ‘ 0 ’, rst<= ‘ 0 ’”。
- (4) 更改状态名 “STATE2” 为 “start”，并添加输出逻辑 “clkout<= ‘ 1 ’, rst<= ‘ 0 ’”。
- (5) 更改状态名 “STATE3” 为 “counting”，并添加输出逻辑 “clkout<= ‘ 1 ’, rst<= ‘ 0 ’”。
- (6) 更改状态名 “STATE4” 为 “stop”，并添加输出逻辑 “clkout<= ‘ 0 ’, rst<= ‘ 0 ’”。
- (7) 更改状态名 “STATE5” 为 “stopped”，并添加输出逻辑 “clkout<= ‘ 0 ’, rst<= ‘ 0 ’”。

状态编辑完成后的状态转移图如图 9-29 所示。

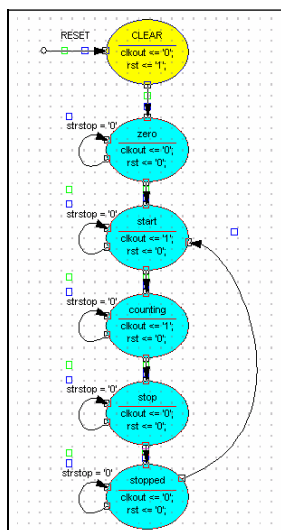


图9-29 完成状态编辑后的转移图

逻辑向导的功能十分强大，不仅能够添加常数逻辑而且可以设计多种其他逻辑，如加

法、比特比较器、递增、递减、乘法器、按位左移、按位右移、门逻辑（与、或、非、取反、异或等）、锁存器、触发器等。

5. 编辑状态转移条件。

对比原状态图，需要修改的转移条件有 3 种：第 1 种是自循环转移条件，由于本例在创建状态图模板时已经事先设置自循环转移条件为“strstop = ‘0’”，下面仅需将状态“start”、“stop”的自循环转移条件改为“strstop = ‘1’”；第 2 种是复位条件，将异步复位条件改为“reset = ‘1’”；第 3 种是添加状态间的转移条件。

- (1) 编辑状态转移条件与编辑状态十分相似。双击状态图中的转移条件，进入转移条件编辑框，即可编辑状态转移条件。同样可以单击 **Output Wizard** 按钮，启动逻辑向导辅助设计状态转移条件。
- (2) 用鼠标双击状态“start”的自循环转移条件，进入转移条件编辑框，更改状态转移条件为“strstop = ‘1’”，如图 9-30 所示，单击 **OK** 按钮确认对状态转移条件的编辑。

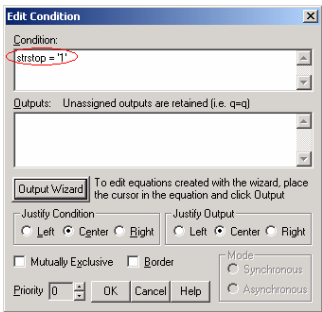


图9-30 编辑状态转移条件

- (3) 使用同样的方法编辑其他状态转移条件，修改后的状态转移图如图 9-31 所示。

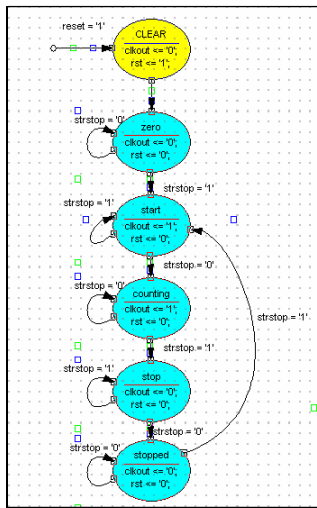


图9-31 完成状态转移条件编辑后的转移图

6. 为状态转移图添加注释与个性化色彩设置。

对比图 9-20 与图 9-31，状态转移图的主体已经绘制完毕，读者还可以根据个人喜好为状态转移图添加注释并设置个性化色彩。

- (1) 单击绘图工具栏中的  按钮，在图纸顶部单击鼠标左键，弹出文本编辑框，添加注释“ My_Stopwatch ”。单击鼠标左键选择状态“ zero ”，单击色彩工具栏中的  按钮，单击  按钮选择绿色，最后点击状态“ zero ”填充颜色。使用色彩工具栏的其他按钮还可以完成更改边框颜色、文字颜色和字体大小等功能。
- (2) 单击  按钮保存设计。至此状态转移图的绘制过程全部结束，绘制好的“计时表”的状态转移图如图 9-32 所示。

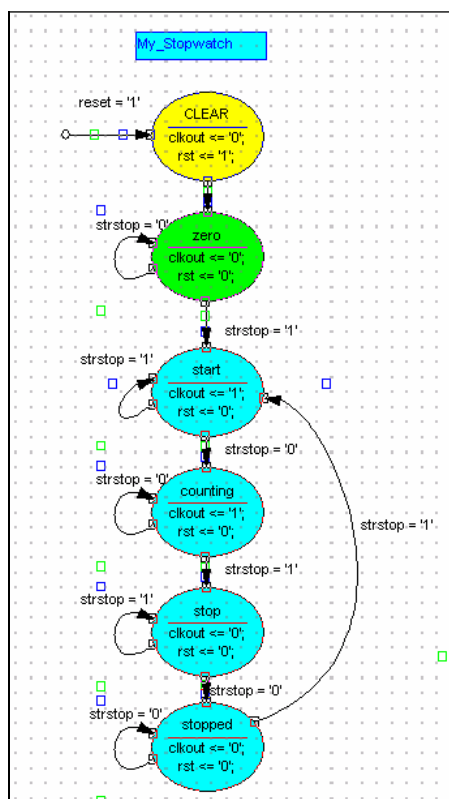


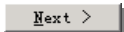
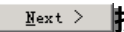
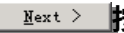
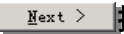
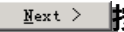
图9-32 绘制完成的“计时表”状态转移图

7. 使用优化向导优化设计。

状态转移图绘制完毕后，启动优化向导，可以使整个状态机在速度、面积、I/O 端口和综合结果等方面达到全面优化。

- (1) 单击  按钮启动优化向导。优化向导的第 1 个界面解释了优化的过程与方法。
- (2) 单击  按钮进入优化向导的第 2 个界面，选择器件类型为“FPGA”。
- (3) 单击  按钮进入优化向导的第 3 个界面，选择优化目标为“速度”。

(Speed) 优先”。

- (4) 单击  按钮进入优化向导的第 4 个界面，选择保证状态覆盖 (Guarantee coverage) 选项。
- (5) 单击  按钮进入优化向导的第 5 个界面，选择优化参数：“缓冲有效 (Enable buffer)”，“优化 I/O 端口 (Optimize Port I/O)”，“未连接输出自动回归 (Retain unassigned outputs between clock)”。
- (6) 单击  按钮进入优化向导的第 6 个界面，选择源代码类型为“Verilog” (使用 VHDL 综合流程的用户请选择代码类型为“VHDL”)。
- (7) 单击  按钮进入优化向导的第 7 个界面，选择综合工具为“Xilinx XST”。
- (8) 单击  按钮完成这个设计的优化。

经过以上操作，StateCAD 完成了速度、面积、I/O 端口和综合结果等项目的全面优化。

8. 编译设计生成 HDL 代码。

StateCAD 通过编译状态机，将用户绘制的状态转移图自动翻译成 HDL 源代码。编译时 StateCAD 首先检查状态机的正确性和完备性，当状态机完全正确后才能编译通过，翻译生成 HDL 代码。

- (1) 单击  按钮，进行编译，选择删除冗余条件。编译结束后，看到编译正确信息对话框如图 9-33 所示。

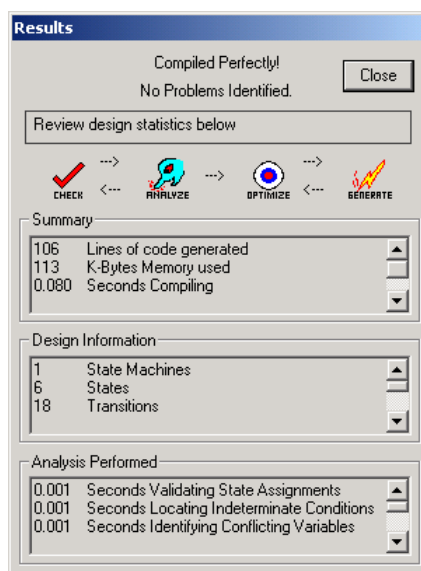


图9-33 编译正确信息对话框

- (2) 单击  按钮关闭编译正确信息对话框。StateCAD 自动翻译生成 HDL 源代码，并在 HDL 浏览窗显示。图 9-34 显示了“计时表”状态转移图和 HDL 代码。

至此，在 StateCAD 中设计“计时表”的所有步骤全部完成。

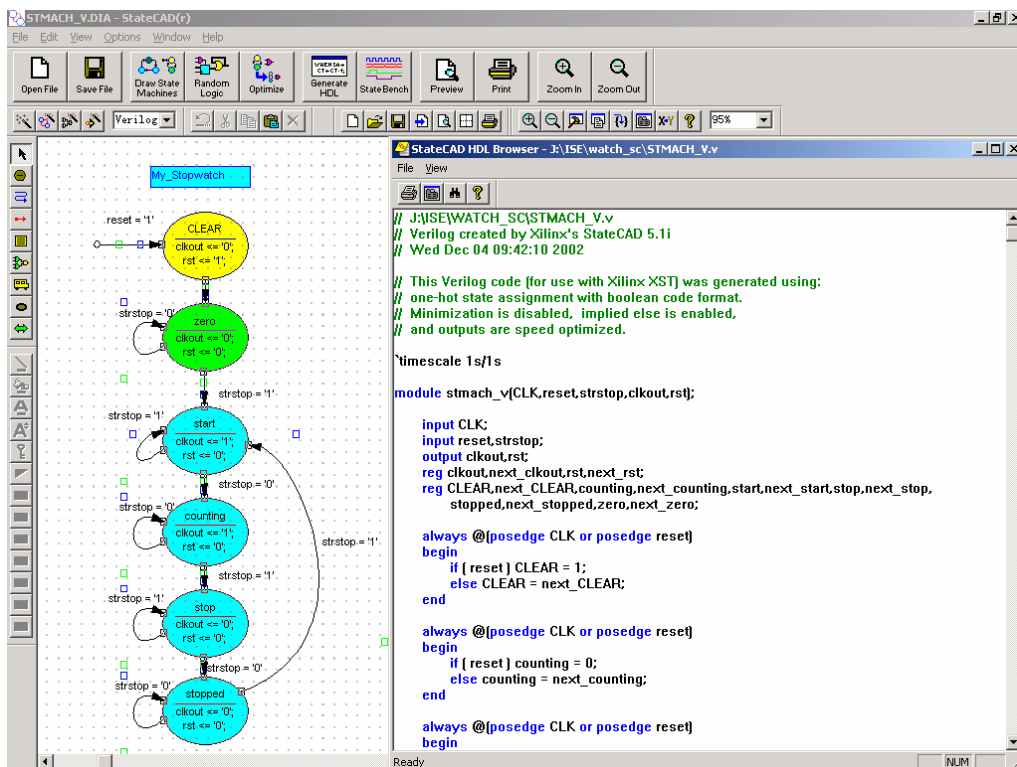


图9-34 “计时表”状态转移图和 HDL 代码

退出 StateCAD，在 Project Navigator 中将“stmach_v.v”源代码添加到“watch_sc”工程中（使用 VHDL 综合流程的用户请添加“stmach_v.vhd”源代码），在资源管理窗选中“stmach_v”源代码资源，展开当前资源操作窗的“Design Entry Utilities”项目，双击“Create Schematic Symbol”命令生成“stmach_v”状态机的模块符号。

9.2.5 使用 Architecture Wizard 生成时钟管理模块“dcm1”

本书 2.5 节已经讨论过使用 Architecture Wizard 设计数字时钟管理模块（DCM，Digital Clock Manager）和高速 I/O（Rocket I/O™ transceivers）的方法。下面使用 Architecture Wizard 设计本例数字时钟管理模块“dcm1”。

使用 Architecture Wizard 设计 DCM 基本分为：新建 Architecture Wizard 资源；选择 DCM 模块类型；设计 DCM 参数并生成用户 DCM 模块等步骤。

🔑 使用 Architecture Wizard 生成时钟管理模块“dcm1”，参考示例详见本书附带光盘的“Example-9-1”目录

1. 新建 Architecture Wizard 资源。

(1) 单击新建资源快捷按钮 ，弹出新建资源对话框，选择新建资源类型为“Architecture Wizard”，新建文件名为“DCM_1”，并且直接添加到工程中，如图 9-35 所示。

(2) 单击  按钮，单击  按钮确认新建资源信息。

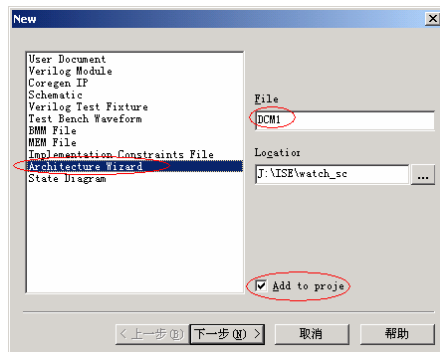


图9-35 新建 Architecture Wizard 资源

2. 选择 DCM 模块类型。

本例选用的器件类型“xc2v80 fg256 -6”属于 Virtex-II 器件族，使用 Architecture Wizard 仅能生成 DCM 模块。仅当选用 Virtex-II Pro 器件族器件时，使用 Architecture Wizard 才能生成 DCM 和 Rocket I/O™ transceivers 两种模块。

选择“DCM Wizard”，单击 按钮启动 DCM 向导。

3. 设计 DCM 参数并生成用户 DCM 模块。

(1) DCM 向导界面与 Core Generator 生成 IP 核的界面非常相似, 按照图 9-36 所示设置 DCM 基本参数。各选项的设置含义如下:

- 取消打勾，删除“RST”和“LOCKED”信号；
- 输入时钟频率为“50MHz”；
- 输入时钟源（Clkin Source）：“External”；
- 反馈时钟源（Feedback Source）：“Internal”；
- 反馈倍频（Feedback Value）：“1X”；
- 移相（Phase Shift）：“None”；
- 占空比整形（Duty Cycle Correction）：“Yes”。

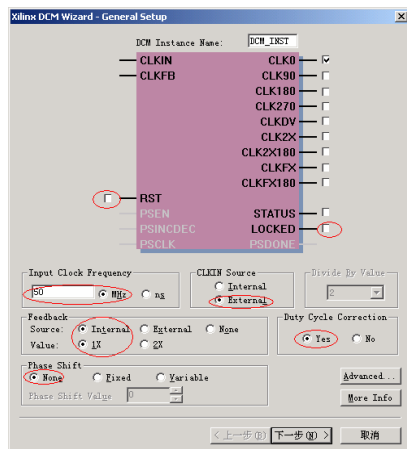


图9-36 设置 DCM 基本参数

- (2) 单击 **Advanced...** 按钮，按照图 9-37 所示设置 DCM 高级参数。
- 在 “DONE” 信号变为高电平前，等待 DCM 锁定：“Yes”；
 - 输入时钟 2 分频：“No”。

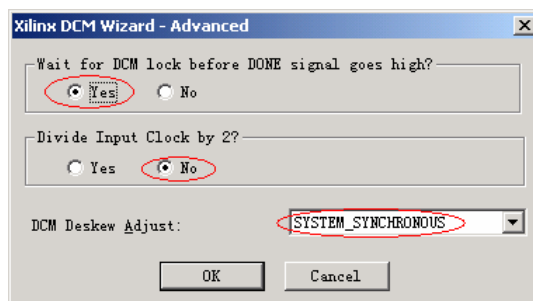


图9-37 设置 DCM 高级参数

- (3) 单击 **OK** 按钮确认高级参数设置，单击 **下一步(N) >** 按钮，弹出关于 “Locked” 信号的信息框，单击 **OK** 按钮确认，弹出全局时钟缓冲使用设置对话框。为了更好的性能与精度，Xilinx 强烈建议 DCM 模块时钟信号使用全局时钟缓冲（BUFGP，Global Buffer）。单击 **完成** 按钮使用全局时钟资源，完成 DCM 模块的生成。
4. 在 Project Navigator 资源管理窗中已经自动加入 “dcm1” 资源。选择该资源，展开当前资源操作窗的 “Design Entry Utilities” 项目，双击 “Create Schematic Symbol” 命令生成 “dcm1” 数字时钟管理模块符号。

9.2.6 使用语言模板设计 “hex2led” 和 “decode” 的 HDL 源代码

在 2.2 节已经介绍过 ISE 中集成的 HDL Editor 的使用方法，并重点介绍了如何使用语言模板（Language Templates）辅助源代码设计。本节将利用语言模板设计七段显示译码器 “hex2led” 和 “独热” 译码器 “decode” 的 HDL 源代码。

使用源代码编辑器设计 HDL 代码主要分为：新建 HDL 源代码资源；设计源代码辅助模板；利用语言模板辅助设计源代码；源代码检错等步骤。下面先设计七段显示译码模块 “hex2led”。

🔑 使用语言模板设计七段显示译码模块 “hex2led”，参考示例详见本书附带光盘的 “Example-9-1” 目录

- 新建 HDL 源代码资源。
单击新建资源快捷按钮 ，会弹出新建资源对话框。选择新建资源类型为源代码模块 “Verilog Module”（使用 VHDL 综合流程的请选择 “VHDL Module”），新建文件名为 “hex2led”，并且直接添加到工程中去，如图 9-38 所示。

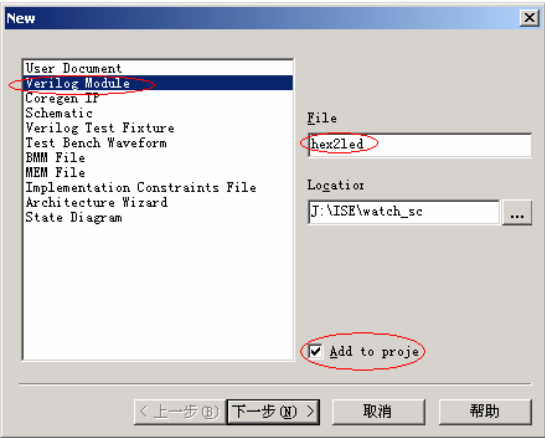


图9-38 新建源代码模块资源

2. 设计源代码辅助模板。
- (1) 单击 **下一步(N) >** 按钮，进入源代码设计模板，按图 9-39 所示编辑输入、输出信号名称与总线上下界。
 - (2) 单击 **下一步(N) >** 按钮，单击 **完成** 按钮确认新建源代码模块。

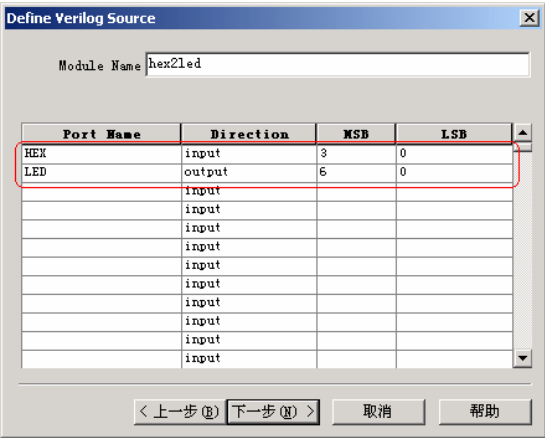


图9-39 编辑模块的输入、输出信号属性

3. 利用语言模板辅助设计源代码。
- HDL Editor 自动根据模块名称和输入、输出信号属性编写程序头等固定框架，减少了设计者的重复劳动。
- (1) 单击 **语言模板** 按钮启动语言辅助模板。在模板分类目录中选择 Verilog 目录（使用 VHDL 综合流程的读者请打开 VHDL 目录），在综合模板（Synthesis Templates）分类下选择 HEX2LED Converter 模板，如图 9-40 所示。
 - (2) 单击 **插入** 按钮，将范例程序插入到“hex2led”模块中。根据范例程序，对源代码稍作修改即可，如图 9-41 所示。

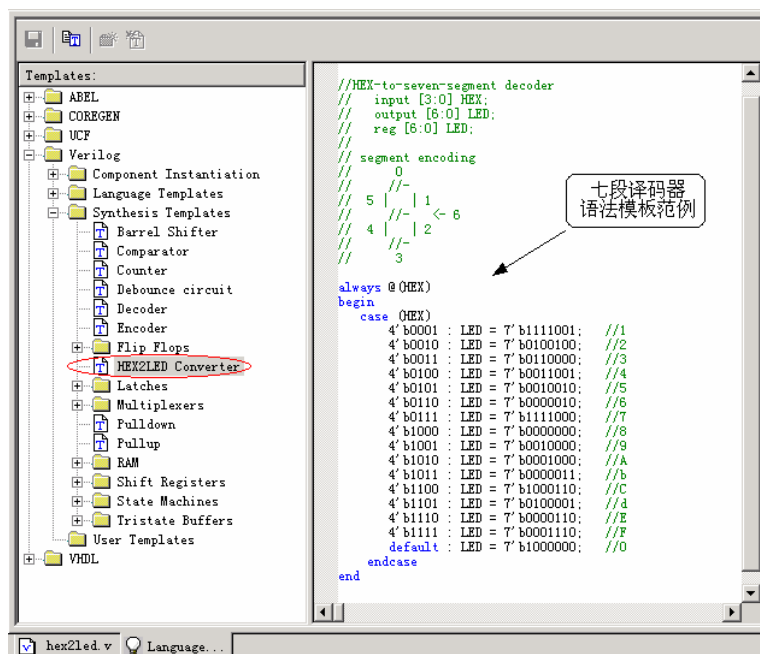


图9-40 七段译码显示语法模板范例

```

1 module hex2led(HEX, LED);
2   input [3:0] HEX;
3   output [6:0] LED;
4   reg [6:0] LED;
5   // segment encoding
6   0
7   // 5 | 1
8   // 4 | 2
9   // 3
10  // 4 < 6
11  // 4 | 2
12  // 3
13  // 3
14
15  always @ (HEX)
16  begin
17    case (HEX)
18      4'b0001 : LED = 7'b1111001; //1
19      4'b0010 : LED = 7'b0100100; //2
20      4'b0011 : LED = 7'b0110000; //3
21      4'b0100 : LED = 7'b0011001; //4
22      4'b0101 : LED = 7'b0010010; //5
23      4'b0110 : LED = 7'b0000010; //6
24      4'b0111 : LED = 7'b1111000; //7
25      4'b1000 : LED = 7'b0000000; //8
26      4'b1001 : LED = 7'b0010000; //9
27      4'b1010 : LED = 7'b0001000; //A
28      4'b1011 : LED = 7'b0000011; //b
29      4'b1100 : LED = 7'b1000110; //C
30      4'b1101 : LED = 7'b0100001; //d
31      4'b1110 : LED = 7'b0000110; //E
32      4'b1111 : LED = 7'b0001110; //F
33      default : LED = 7'b1000000; //0
34    endcase
35  end
36
37
38 endmodule
39

```

图9-41 七段译码显示模块“hex2led”

4. 源代码检错。

在 ISE 中源代码的检错一般在综合步骤完成。

- 在 Project Navigator 资源管理窗选择源代码资源，展开当前资源操作窗的“Synthesis”项目，双击“Check Syntax”命令对“hex2led”进行检错。“hex2led”没有语法错误，接着展开当前资源操作窗的“Design Entry”

Utilities”项目，双击“Create Schematic Symbol”命令生成器件符号。

运用同样的设计方法设计“独热”译码器“decode”，并生成器件符号。“decode”的源代码如图 9-42 所示。

```

1 module decode (binary, one_hot);
2 input [3:0] binary;
3 output[9:0] one_hot;
4
5 reg [9:0] one_hot;
6
7 always @(binary)
8 begin
9     case (binary)
10        4'b0001 : one_hot = 10'b0000000001; //1
11        4'b0010 : one_hot = 10'b0000000010; //2
12        4'b0011 : one_hot = 10'b0000000100; //3
13        4'b0100 : one_hot = 10'b0000001000; //4
14        4'b0101 : one_hot = 10'b0000010000; //5
15        4'b0110 : one_hot = 10'b0000100000; //6
16        4'b0111 : one_hot = 10'b0001000000; //7
17        4'b1000 : one_hot = 10'b0010000000; //8
18        4'b1001 : one_hot = 10'b0100000000; //9
19        4'b1010 : one_hot = 10'b1000000000; //10
20        default : one_hot = 10'b0000000001; //1
21    endcase
22 end
23
24
25 endmodule
26

```

图9-42 “独热”译码器“decode”源代码

9.2.7 使用 ECS 设计顶层原理图

至此，“计时表”的 7 个子模块全部设计完毕，如图 9-43 所示。在设计过程中，已经将源代码、IP Core、DCM、状态机、原理图等资源生成模块符号（Symbol）。下面将使用 ECS 设计顶层原理图，调用所生成的模块符号，达到组织各种不同设计资源的目的。这就是 ISE 中实现多元混合设计输入的基本方法。

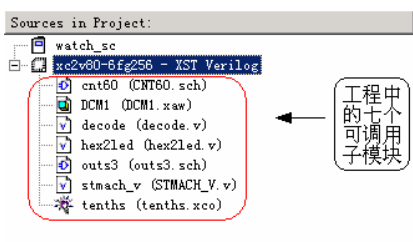


图9-43 “计时表”的 7 个子模块

设计顶层原理图的方法与绘制其他原理图基本一样，唯一的区别是选择器件符号时使用本工程中用户自建的模块符号。本节对原理图的具体绘制过程一带而过，而将重点放在用 ECS 原理图实现多元混合设计输入的概念上。

🔑 使用 ECS 设计顶层原理图，实现多元混合设计输入，参考示例详见本书附带光盘的“Example-9-1”目录

1. 新建顶层原理图“stopwatch”。

单击新建资源快捷按钮 ，弹出新建资源对话框，选择新建资源类型为原理图（Schematic）类型，新建文件名为“stopwatch”，并且直接添加到工程中去。

单击  按钮，单击  按钮确认新建资源信息，启动 ECS。

2. 选择并放置器件符号。

参看图 9-1 所示的顶层原理图，在顶层原理图中除了需要放置标准的“IBUF”、“BUFG”、“INV”和“AND2”等标准器件符号，还需要放置用户生成的“stmach_v”、“DCM_1”、“decode”、“outs3”、“cnt60”和“hex2led”等器件符号。放置用户器件符号的方法和放置标准器件符号完全一致，仅仅是在选择器件符号分类显示目录时选择当前工程路径即可。请参考顶层原理图的器件布局，在用户新建的顶层原理图中布置器件符号，如图 9-44 所示。

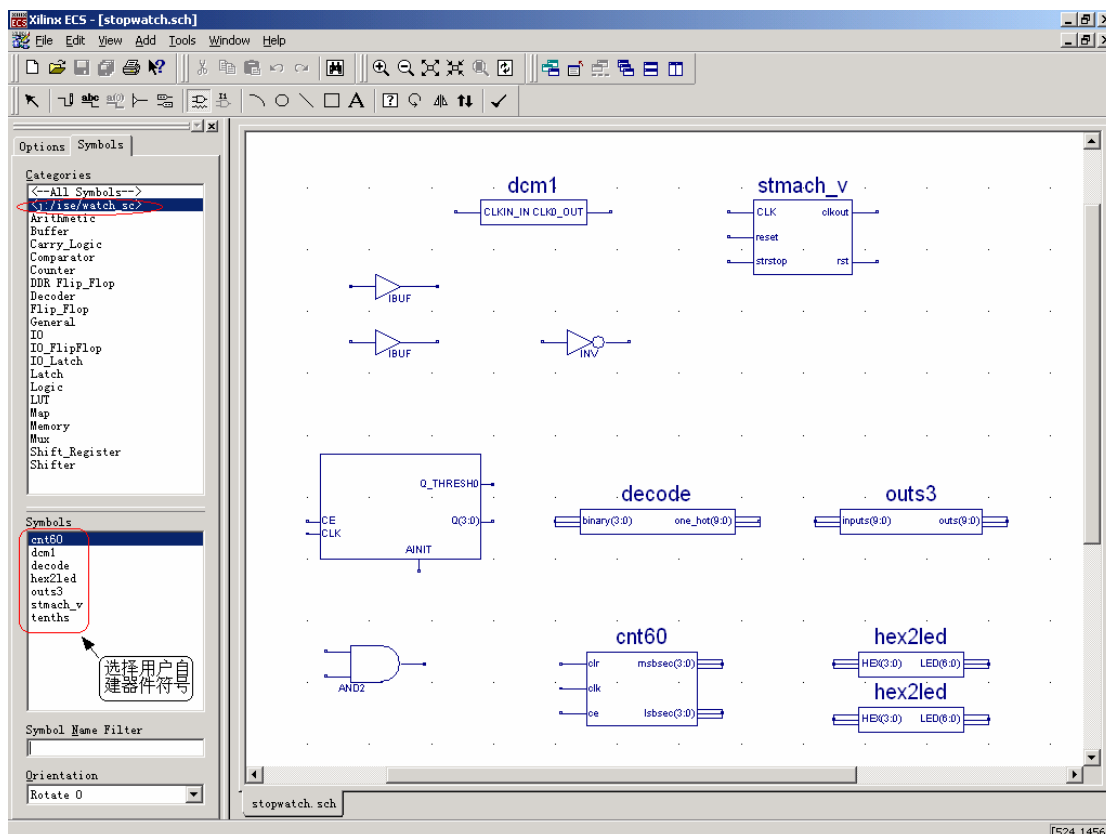


图9-44 选择并放置用户自建器件符号

请读者参照图 9-1 所示的原理图完成添加连线、添加网线名称、设置 I/O 管脚、绘制总线、原理图检查等剩余步骤。

9.3 测试激励与行为级功能仿真

ISE 中集成的仿真工具主要有两个：一是 HDL Bench 测试激励生成器；二是 ModelSim 仿真器。HDL Bench 将 VHDL、Verilog 源代码、ECS 原理图等设计输入导入其测试环境，根据用户在图形界面下编辑的测试波形，直接生成测试激励文件，然后调用 ISE 中集成的仿真工具进行仿真验证。ModelSim 可以说是业界最流行的仿真工具之一，它的主要特点是仿真速度快、仿真精度高。ModelSim 支持 VHDL、Verilog 语言和这两种语言混合

编程的仿真。

本节将演示如何使用测试激励生成器产生这个“计时表”的测试激励，并使用 ModelSim 做行为级功能仿真。

9.3.1 使用 HDL Bencher 生成测试激励

在 2.6 节详细介绍了测试激励生成器 HDL Bencher 的特点与使用方法。HDL Bencher 生成测试激励基本可以分为新建测试激励资源文件；设置基本仿真参数；编辑激励波形；生成测试激励等步骤。

✎ 使用 HDL Bencher 生成测试激励，参考示例详见本书附带光盘的“Example-9-1”目录

1. 新建测试激励资源文件。

- (1) 单击新建资源快捷按钮 ，弹出新建资源对话框，选择新建资源类型为“Test Bench Waveform”，新建文件名为“stopwatch_top_tb.tbw”，如图 9-45 所示。
- (2) 单击  按钮，选择需要仿真的模块，新建测试激励波形文件将与该模块关联。这里的仿真目标为顶层文件“stopwatch”，如图 9-46 所示。单击  按钮，单击  按钮确认新建资源信息，启动 HDL Bencher。

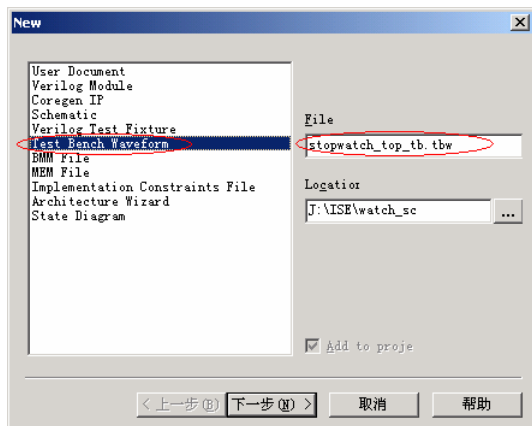


图9-45 新建测试激励文件

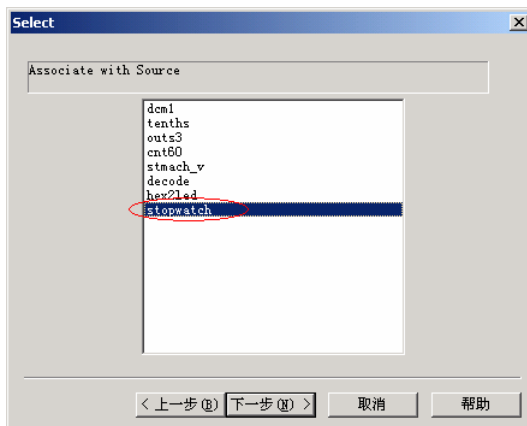


图 9-46 选择仿真目标

2. 设置基本仿真参数。

首先需要设置基本仿真参数设置对话框，在这个对话框中主要完成时钟周期、设计类型、全局信号等参数的设置。时钟周期需要设置的参数有触发沿（上升沿触发、下降沿触发、两个沿都触发），时钟高、低电平保持时间，输入建立时间，输出有效时间，时钟偏移等。设计类型分为 3 类：单时钟同步时序电路、多时钟同步时序电路和组合电路。全局信号设置主要选择是否使用 GSR（FPGA），PRLD（CPLD）全局置位、复位信号。另外还可以设置仿真所用的时间单位（ms、ns 或 ps）等项目。按图 9-47 所示设置基本仿真参数。

如果设计不是同步时序设计，就需勾选【Add Asynchronous Signal support】选项，单击  按钮，设置异步时序参数。本设计是同步时序设计，直接单击  按钮，完成时间

设置，进入 HDL Benchner 测试波形编辑窗口。

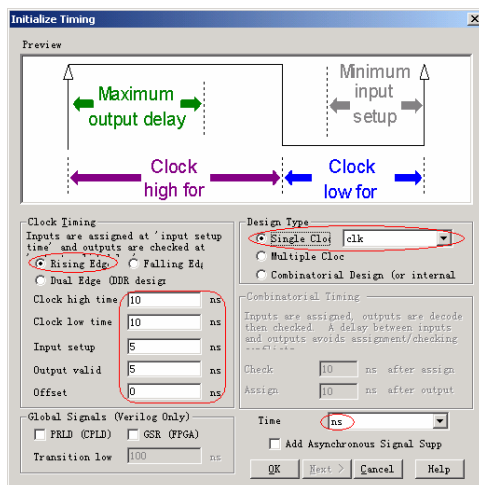


图9-47 设置基本仿真参数

3. 编辑激励波形。

HDL Benchner 测试波形编辑窗口分为两部分：上面是测试激励波形编辑窗，完成测试激励波形的设计与编辑；下面是源代码显示窗，显示被仿真模块的源代码。测试激励波形的编辑方法有许多种，在 2.6 节详细介绍了使用“Pattern Wizard”编辑测试激励波形的方法。这个设计的输入信号非常简单，除时钟信号“clk”外，“reset”信号完成“计时表”的复位与时间清零，“strstop”信号是“计时表”的启动/停止计时的控制信号。在这两个信号的适当位置单击鼠标，就能改变信号电平，编辑测试激励，如图 9-48 所示。

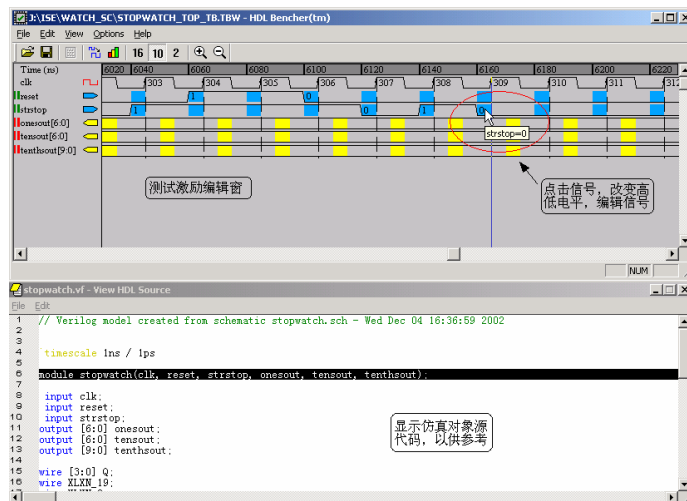


图9-48 编辑测试激励波形

4. 生成测试激励。

完成激励波形设计后，单击工具栏中的  按钮，保存设计。接着单击工具栏中的  按钮，查看设计测试激励的覆盖率，如图 9-49 所示。激励可测试的情况越多，覆盖率就越高。如果覆盖率不满意，就需要重新更改测试激励波形。

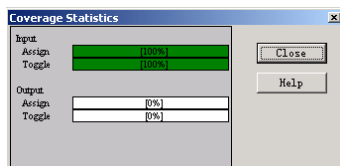


图9-49 测试激励覆盖率

完成测试激励波形的编辑后，退出测试激励编辑器，会看到在 Project Navigator 的资源管理窗中新建了一个测试激励波形资源。双击测试激励波形资源，可以启动 HDL Bench 修改测试激励波形。

9.3.2 调用 ModelSim 进行行为级功能仿真

本节使用 ModelSim 进行行为级功能仿真，也就是通常所说的前仿真。在 9.6 节将介绍如何使用 ModelSim 进行布线后仿真。

HDL Bench 在生成测试激励的同时自动生成扩展名为“fdo”的文件，该文件是 ModelSim 批处理命令文件，包含了编译测试激励源代码、仿真对象源代码，调用 Xilinx 仿真库，打开波形观察窗、结构层次窗、信号列表窗等窗口，添加信号，启动仿真等多条批处理命令。所以如果使用 HDL Bench 设计测试激励，调用 ModelSim 进行仿真非常容易操作。

如果设计使用到了 Xilinx 的底层原语、IP Core 等专用结构，使用 ModelSim 仿真就必须调用 Xilinx 仿真库。Xilinx 仿真库的设置方法详见本书第 3 章 3.3 小节“仿真环境的建立”。

Xilinx 仿真库设置正确后，选择测试激励波形资源“stopwatch_top_tb.tbw”，看到当前资源操作窗中对测试激励波形资源的可用命令是各种类型的仿真操作。双击行为级仿真“Simulate Behavioral Verilog Model”命令，ISE 自动调用 ModelSim 完成行为级功能仿真，仿真结果如图 9-50 所示。参考示例详见本书附带光盘的“Example-9-1”目录。

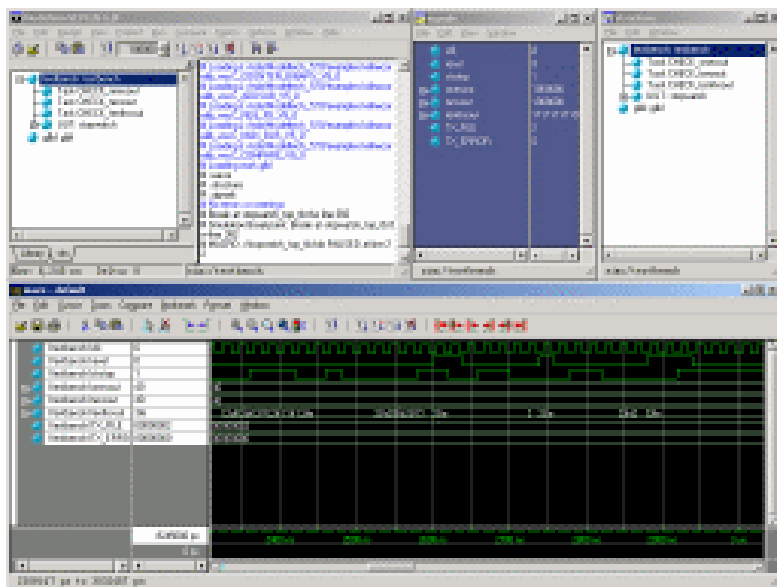


图9-50 使用 ModelSim 进行“计时表”行为级功能仿真

9.4 Synplify Pro 和 XST 综合方法

为了照顾没有 Synplify Pro 的用户，本例使用 XST 综合整个设计，分析综合结果。另外本节还将演示如何使用 Synplify Pro 综合优化设计；如何使用 RTL View 和 Technology View 分析设计综合效果与关键路径。

9.4.1 使用 XST 综合整个设计

本书 4.4 节重点介绍了 XST 的特点与基本使用方法。XST (Xilinx Synthesis Technology) 是 Xilinx ISE 内嵌的综合工具。其基本特点有 3 个：对 Xilinx 器件的支持最为直接；直接集成在 ISE 中，不需另外付费；操作简便。

XST 的基本使用方法很简单，在 Project Navigator 工程属性选项设置中指定采用 XST Verilog 或 XST VHDL 设计流程，然后设置综合属性参数，进行综合即可。

✎ 使用 XST 综合整个设计，参考示例详见本书附带光盘的“Example-9-1”目录

1. 设置 XST 综合属性参数。

(1) 在工程资源窗选中顶层文件“stopwatch”，然后展开当前资源操作窗的综合 (Synthesize) 项目，单击  按钮，编辑操作步骤属性。

(2) XST 综合属性的设置分为综合参数设置，HDL 源代码参数设置和 Xilinx 专用参数设置 3 个选项卡，按照图 9-51~图 9-53 所示设置这 3 个选项卡，设置完成后单击  按钮保存综合参数设置。每个参数选项的具体含义参见第 4 章。

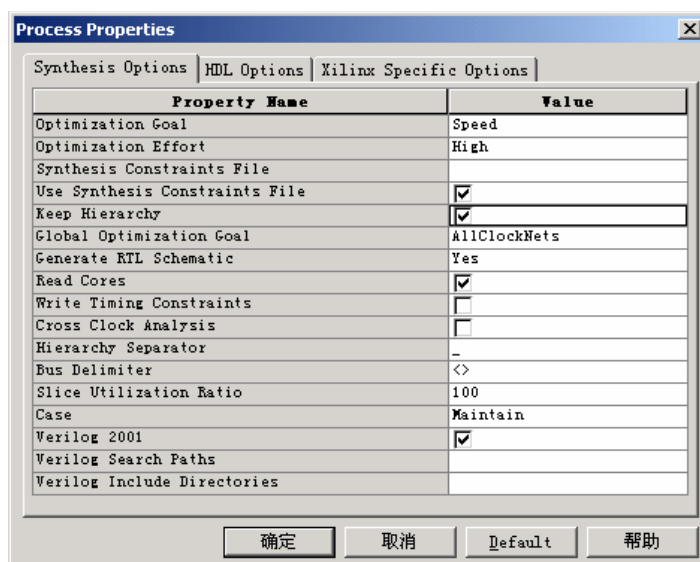


图9-51 设置 XST 综合参数



图9-52 设置 XST HDL 源代码参数

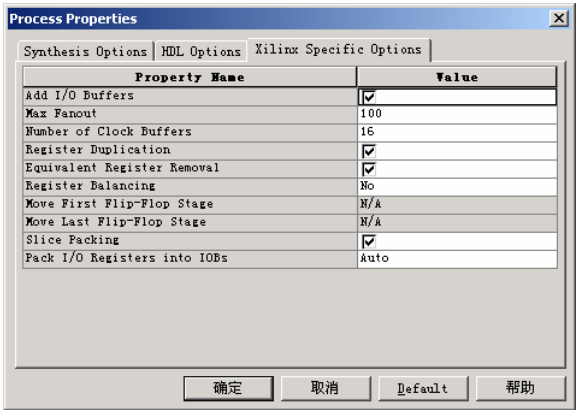


图9-53 设置 XST Xilinx 专用参数

2. 使用 XST 综合。
- 在工程资源窗（Sources in Project）选中顶层文件，然后在当前资源操作窗（Processes for Current Source）选中综合项目（Synthesize），单击  按钮，运行综合步骤。综合过程所有信息可在信息显示窗查看。
3. 分析综合结果。
- (1) XST 综合完成后，展开当前资源操作窗的综合项目，可以看到有 3 项操作：“View Synthesis Report”使用源代码编辑器显示综合报告；“View RTL Schematic”显示设计的寄存器传输级原理图；“Analyze Hierarchy”分析设计的结构层次。其中“View RTL Schematic”用图形界面显示从综合结果提炼出来的设计的原理图，功能与 Synplify Pro 的 RTL View 和 Technology View 相似。
- (2) 双击“View RTL Schematic”图标，ISE 调用 ECS 打开设计的原理图，帮助用户分析综合结果。图 9-54 所示为“计时表”的顶层“stopwatch”的模块符号。

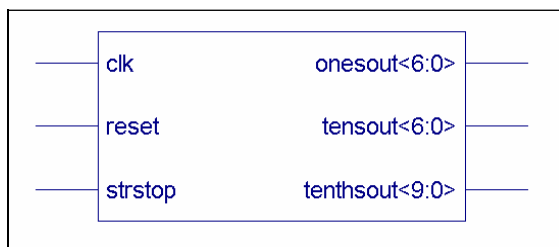


图9-54 顶层“stopwatch”的模块符号

- (3) 双击顶层模块符号，进入根据综合结果提炼出的“计时表”的原理图，如图 9-55 所示。

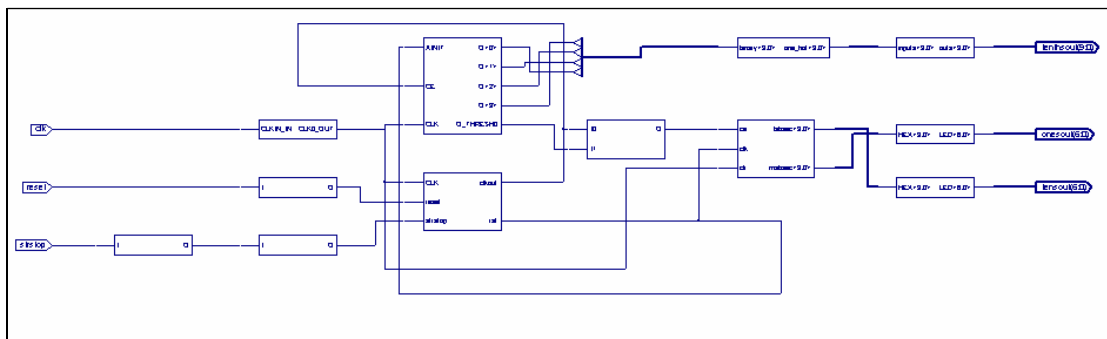


图9-55 根据综合结果提炼出的“计时表”的原理图

- (4) 双击对应“计时表”状态机“stmach_v”的模块符号，进入根据该状态机综合结果提炼出的原理图，如图 9-56 所示。

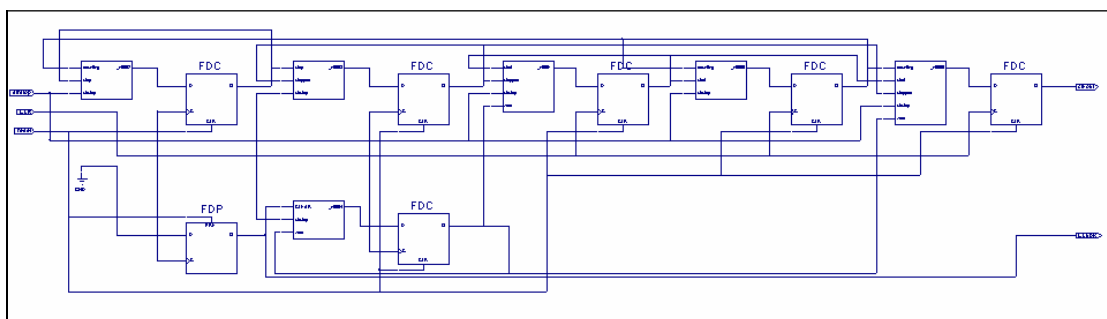


图9-56 根据综合结果提炼出的状态机“stmach_v”的原理图

- (5) 使用 ECS 的  工具栏，可以完成在 RTL 视图中的层次进出和原理图切换。

9.4.2 使用 Synplify Pro 的特色工具分析、优化设计

Synplicity 公司出品的 Synplify/Synplify Pro 综合工具是目前备受推崇的 FPGA/CPLD 综合工具之一。Synplify/Synplify Pro 采用了很多独特的整体性优化策略和方法，使它们对设

设计的综合无论在物理面积还是工作频率上都能达到较理想的效果。

“运动计时表”设计比较简单，还不能充分体现 Synplify Pro 的综合优势，本节仅演示如何使用 Synplify Pro 的 RTL View 和 Technology View 分析设计综合结果，确认设计的关键路径。

Synplify Pro 综合设计的基本流程包括创建工程、源代码错误检查、RTL View、使用 SCOPE 设计综合约束条件、设置综合优化参数、综合、分析综合结果等步骤。

使用 Synplify Pro 综合整个设计，参考示例详见本书附带光盘的“Example-9-1”目录

1. 创建工程。

本例采用多元混合资源设计，而 Synplify Pro 的输入仅仅为 HDL 源代码，所以这里存在一个将原理图等其他设计资源转换为 HDL 代码的问题。其实 XST 等其他综合工具也有相同的问题，只是 XST 的资源类型转换在 ISE 中自动完成，用户一般不会注意到罢了。资源转换的一般方法为：在资源管理窗中选中需要转换的资源，如 ECS 原理图、DCM 模块等，展开当前资源操作窗的“Design Entry Utilities”项目，双击“View Verilog Functional Model”（VHDL 综合流程请选择“View VHDL Functional Model”）命令，ISE 自动将该资源转换为 HDL 代码并在源代码编辑器中显示，该源代码的功能与原来资源模块完全一致。

在工程目录下新建“Synplify_Pro”目录，将“tenths.v”（Core Generator 生成的 IP Core 的源代码声明）、“decode.v”、“hex2led.v”、“STMACH_V.v”（StateCAD 生成的状态机模块的源代码）、“dcm1.v”（Architecture Wizard 生成的 DCM 模块的源代码）、“cnt60.vf”（ECS 原理图模块的源代码）、“outs3.vf”（ECS 原理图模块的源代码）和“stopwatch.vf”（ECS 原理图模块的源代码）等源代码拷贝到“\watch_sc\Synplify_Pro”工程目录下（VHDL 综合流程为扩展名是“vhd”和“vhf”的对应文件）。

本例使用了 Xilinx 的底层硬件原语如“AND2”、“OR2”等模块，所以需要将这底层硬件原语声明为黑盒子。声明方法为仅仅对模块做端口声明而无实际功能，或者使用 Synplify Pro 黑盒子属性命令声明。另外需要将 Synplify Pro 安装目录下“lib\Xilinx”子目录中的 Virtex-II 器件族的 Verilog 或 VHDL 模块声明文件拷贝到“\watch_sc\Synplify_Pro”工程目录下。

以上所有源代码文件在本书配套光盘第9章示例工程的源代码目录下，分为 Verilog 和 VHDL 两种语言存放，请读者根据自己的综合流程选用。

启动 Synplify Pro，本例不使用工程向导创建工程，手工完成工程的创建。在工程管理窗空白处，单击鼠标右键，弹出工程命令对话框，选择新建工程（New Project）命令。新建的工程在保存前默认工程名为“Proj”。用鼠标选中工程，单击鼠标右键，弹出命令菜单，选择添加源文件“Add Source File”命令，如图 9-57 所示，添加“\watch_sc\Synplify_Pro”工程目录下所有文件。

完成源代码添加后，必须指定模块文件，将顶层模块“stopwatch.vf”（VHDL 综合流程为“stopwatch.vhf”文件）用鼠标拖到源文件的最后面，或者在工程属性设置中选择顶层文件为“stopwatch.vf”（VHDL 综合流程为“stopwatch.vhf”文件）。单击按钮保存工程设置。这时会弹出【Save Project As】对话框，选择工程存储路径为

“\watch_sc\Synplify_Pro”，并在【Project Name】中输入工程名称“Syn_Pro_stopwatch”，单击 **保存(S)** 按钮保存工程。

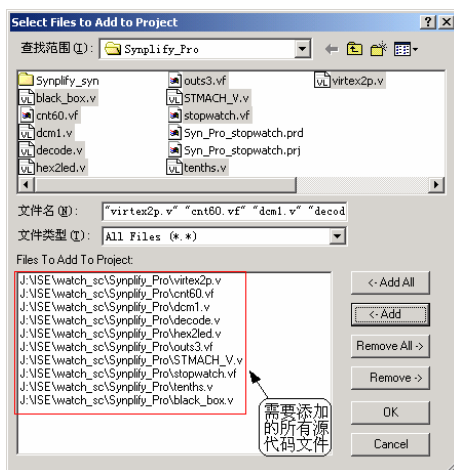


图9-57 添加源文件

2. 源代码错误检查。

Synplify Pro 内嵌 HDL 编辑器检错功能强大。语法检查 (Syntax Check) 检查源代码中是否有语法错误；综合检查 (Synthesis Check) 检查源代码是否是可综合的。

使用快捷键组合 **Shift+F7** 进行语法检错，**Shift+F8** 进行综合检错，经检错验证，工程内源代码无语法、综合错误，可以正确综合。

3. 使用 RTL View 分析设计的寄存器传输级结构。

选择菜单栏中的【Run】(运行) / 【Compile Only】(预编译) 命令，对设计进行编译。编译完成后，双击扩展名为“srs”、对应图标为  的 RTL 视图文件，启动 RTL 视图，如图 9-58 所示。使用视图操作工具栏  对 RTL 视图完成放缩、进出结构层次等控制，协助用户更好地分析设计的寄存器传输级结构。

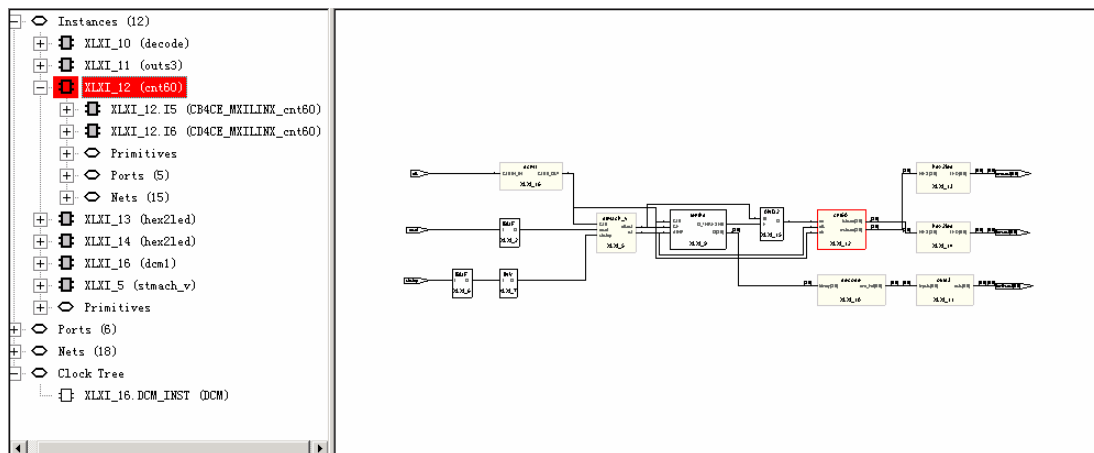


图9-58 Synplify Pro 产生的“计时表”RTL View

RTL 视图有强大的互链切换（Cross-Probing）功能：用户可以通过在相应窗口选中模块，双击鼠标就可以方便地完成模块、结构分类目录，RTL 原理图，源代码之间的互链切换。如图 9-59 所示，时钟树中的“DCM”时钟模块，RTL 视图中的原理图符号，源代码等 3 个高亮部分，用鼠标双击任何一个部分都能唤出其他两个部分。在 RTL 视图使用互链切换功能，能更好地理解源代码，完成对设计的调试。

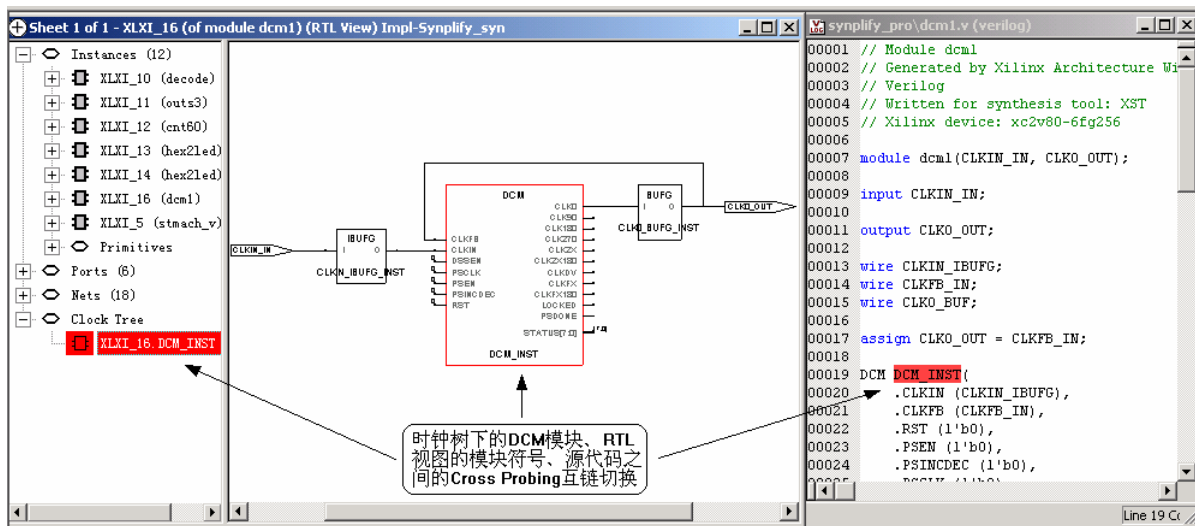


图9-59 时钟树中的“DCM”时钟模块，RTL 视图中的原理图符号，源代码三者之间的互链切换

4. 使用 SCOPE 设计综合约束条件。

综合约束条件非常重要，直接决定综合结果的硬件结构。Synplify Pro 的综合约束文件是 SDC 文件，SCOPE (Synthesis Constraints Optimization Environment, 综合约束优化环境) 使用图形界面辅助用户设计该综合约束文件。

单击 按钮，启动 SCOPE，首先弹出的是新建 SCOPE 文件对话框。该对话框包括两个选项卡：一是初始约束内容，二是约束文件类型。初始约束内容选项卡主要确定时钟和 I/O 的约束是否需要 SCOPE 做自动初始化。设计在编译 (Compile) 后，SCOPE 可以自动提取出设计中应用的时钟和 I/O 引脚信息，在约束时非常方便，所以强烈建议两个选项都选。约束文件类型选项卡是针对多位置设计流程 (Multipoint Synthesis Flow) 新增加的功能，可以选择约束是相对于顶层还是某一模块进行的。如果对整个设计进行约束，需要选择对顶层约束 (Top Level)。

单击 按钮，进入 SCOPE 主界面。SCOPE 主界面主要由时钟 (Clocks)、输入输出 (Inputs/Outputs)、寄存器 (Registers)、多周期路径 (Multi-Cycle Paths)、弱约束路径 (False Paths)、约束属性 (Attributes)、多位置编译 (Compile Points) 和其他 (Other) 等 8 个约束项目选项卡组成。本例综合约束文件的设计比较简单，仅仅做如下约束 (参见图 9-60)：

- 约束主时钟 clk 频率为 100MHz。
- 约束输入、输出的默认布线延时为 2ns。
- 约束 DCM 输出时钟 clkout 的布线延时小于 5ns。

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock
1	☒	clk	100	10.000	default_clkgroup			50		☒

	Enabled	Port	Type	Relative to Clock	Clock Edge	Value (ns)	Route (ns)
1	☒	<input default>	input_delay	clk	clk:r		2
2	☒	<output default>	output_delay	clk	clk:r		2

	Enabled	Register	Type	Route (ns)
1	☒	XLXI_5.clkout	reg_input_delay	5
2	☒	XLXI_5.clkout	reg_input_delay	5

图9-60 使用 SCOPE 设计综合约束

单击  按钮保存约束文件，弹出保存对话框，输入约束文件名（扩展名为 .sdc），单击  按钮确认。这时会弹出对话框询问是否将约束文件加入到工程，单击  按钮确认，将新建约束文件加入到当前工程。退出 SCOPE，可以看到新建的约束已经加入到工程中。

5. 设置综合优化参数。

常用的综合优化参数可以在主界面下直接输入，但是最全面的设置综合优化参数的方法是在 Synplify Pro 的主用户界面下单击  按钮，启动综合参数设置对话框进行参数设置。综合优化参数设置对话框主要由器件（Device）、优化选项（Options）、综合约束文件（Constraints）、综合结果存储（Implementation Results）、时序报告（Timing Report）和相应语言选项（Verilog 或 VHDL）等 6 个选项卡组成。每个选项卡的设置如图 9-61~图 9-66 所示。

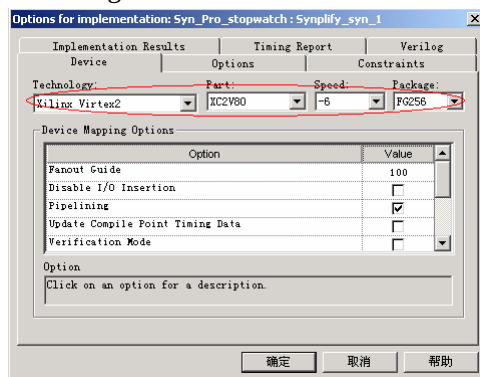


图9-61 设置器件综合参数

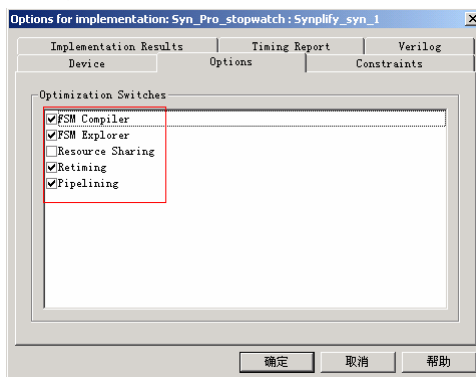


图 9-62 设置优化选项综合参数

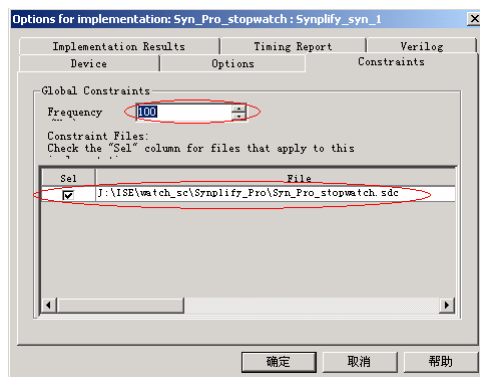


图9-63 设置综合约束文件综合参数

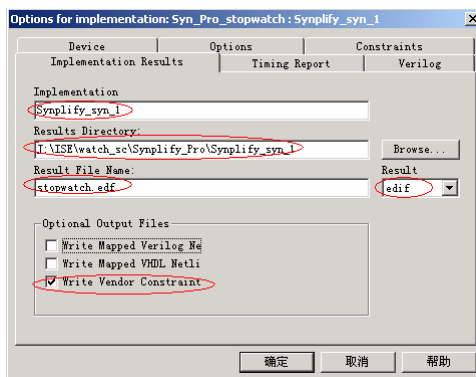


图 9-64 设置综合结果存储综合参数

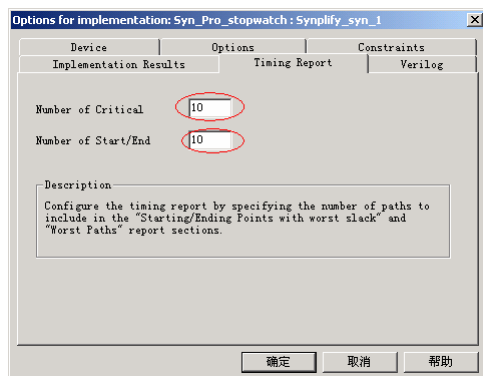


图9-65 设置时序报告综合参数

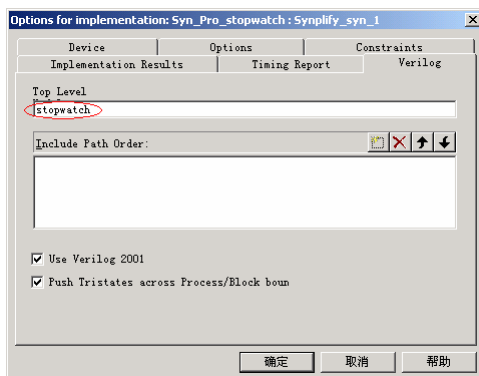


图 9-66 设置语言选项综合参数

6. 综合。

完成了综合时序约束文件设计和综合优化参数设置，进行综合就变得非常简单。单击 Synplify Pro 主界面中最大的按钮 ，综合器自动根据要求进行综合，这时可以看到状态显示栏依次显示“Compiling”（编译）和“Mapping”（映射）。当综合完成后，状态显示栏显示“Done”表示已完成综合过程。

7. 分析综合结果。

综合完成后发现综合结果显示窗新增了 7 个文件：扩展名为“srr”的文件是工程报告，包含了工程检错、编译、综合、时序等所有工程信息，用于分析综合结果；扩展名为“tlg”的文件是工程组织结构信息文件；扩展名为“srs”的文件是 RTL 视图文件；扩展名为“edif”的文件是综合器输出的网表，导入到器件商的布线器，进行布局布线等操作；扩展名为“fse”的文件是有限状态机编码文件；扩展名为“ncf”的文件是综合器产生的 Xilinx 布局布线约束文件；扩展名为“srm”的文件是结构级视图文件。

分析综合结果包含 3 项内容：使用综合报告观察窗（Log Watch window）查看重要综合信息；阅读综合报告，查看工程信息、检错信息、有限状态机编译信息、时序约束信息、I/O 信息、寄存器优化信息、组合电路时序优化信息、路径时延信息、器件综合频率面积总结等详细综合信息；使用结构视图（Technology View）、时序分析专家（Timing Analyst）等分析关键时序与综合结果。

本例仅仅演示使用结构视图分析综合结果的方法。结构视图是将设计用 FPGA/CPLD 的硬件原语（Primitive）和底层模块描述的门级结构原理图。它与 RTL 视图既相似又不同，结构视图对设计的描述更深刻，可以说结构视图是 RTL 视图向具体器件进行结构映射的结果。结构视图结合 RTL 视图能很好地帮助工程师对整个综合过程有一个较深刻的认识，帮助工程师理解设计电路，分析关键路径。

双击结构视图文件，打开结构视图。结构视图的基本界面结构和 RTL 视图完全一致，同样使用视图操作工具栏  对视图进行放缩、进出结构层次等控制。结构视图也有强大的互链切换功能：可在模块、结构的分类目录，结构视图原理图，源代码之间方便地进行互链切换，如图 9-67 所示。

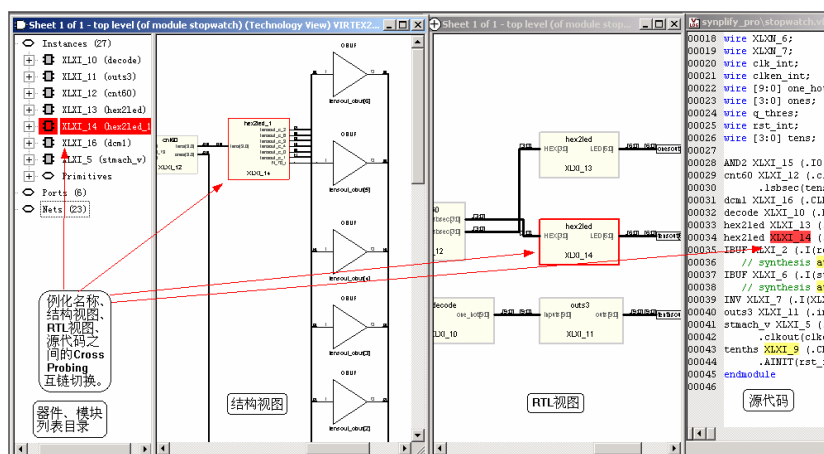


图9-67 模块名称、结构视图、RTL视图、源代码之间的互链切换

结构视图的一个重要功能是分析关键路径。在结构视图观察窗空白处单击鼠标右键，在弹出的命令菜单中选择【Flatten Schematic】（打平原理图）命令，然后单击  按钮，滤出关键路径，如图 9-68 所示。

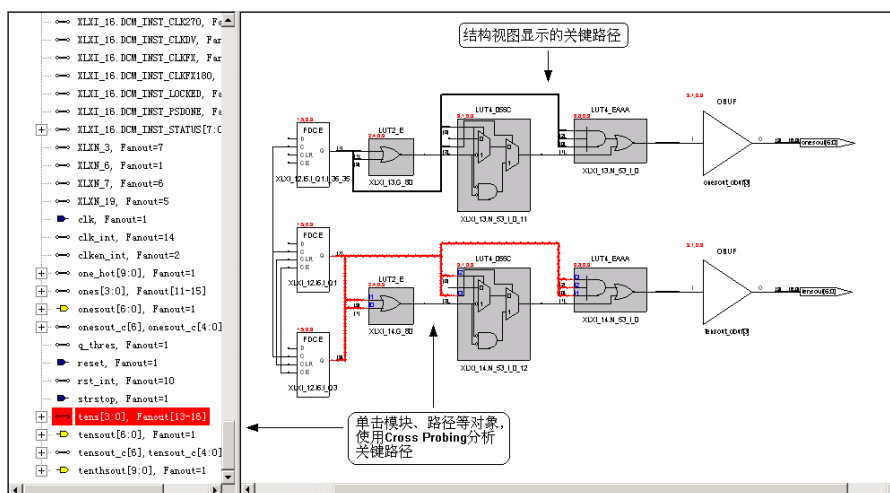


图9-68 使用结构视图分析关键路径

9.5 设计用户约束文件与实现结果的分析

实现的基本含义是将综合输出的逻辑网表适配到具体 FPGA/CPLD 的过程。在 ISE 中，实现包括翻译（Translate）、映射（Map）和布局布线（Place & Route）等 3 个步骤。实现涉及的工具比较多，包括约束编辑器（Constraints Editor）、引脚与区域约束编辑器（PACE）、时序分析器（Timing Analyzer）、FPGA 底层编辑器（FPGA Editor）、芯片观察窗（Chip Viewer）和布局规划器（Floorplanner）等。掌握好这些工具，将大幅度提高设计者的水平，使设计工作更加游刃有余。

为了使读者迅速掌握实现步骤的基本操作方法，本节将重点演示使用 Constraints Editor

和 PACE 设计 UCF 文件指导实现过程，分析实现结果。

9.5.1 使用 Constraints Editor 设计 UCF 文件

约束编辑器的输入文件是 NGD (Native Generic Databse) 文件，输出是 UCF (User Constraints File) 文件。约束编辑器的基本功能是将工程师从烦琐的 UCF 约束命令解脱出来，用图形化的界面帮助用户设计 UCF 文件。

使用约束编辑器设计用户约束文件包括新建用户约束文件和利用约束编辑器设置约束属性两大步骤。

🔑 使用 Constraints Editor 设计 UCF 文件，参考示例详见本书附带光盘的“Example-9-1”目录

1. 新建用户约束文件。

使用约束编辑器设计 UCF 文件前应先预实现设计一遍，因为约束编辑器需要从 NGD 文件中读取设计的基本信息，如时钟、寄存器、I/O 端口等信息。请读者在资源管理窗选中设计的顶层模块“stopwatch”，在当前资源操作窗选中实现 (Implement Design) 步骤，单击工具栏中的按钮，不加任何约束与设置，完成工程的预实现。

- (1) 当工程实现完成后单击新建资源快捷按钮，弹出新建资源对话框，选择新建资源类型为“Implementation Constraints File”，新建文件名为“stopwatch_CE”，并直接加入到工程中，如图 9-69 所示。

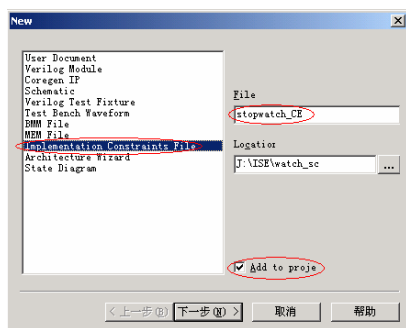


图9-69 新建实现约束文件资源

- (2) 单击按钮，选择需要约束的模块，新建 UCF 文件将与该模块关联。这里选择约束目标为顶层文件“stopwatch_CE”。单击按钮，单击按钮确认新建资源信息。ISE 自动创建一个空的用户约束文件。
- (3) 在资源管理窗选中设计的顶层模块“stopwatch”，展开当前资源操作窗中的“Design Entry Utilities”项目，选择【Creat Timing Constraints】命令启动约束编辑器，设计新建的空白用户约束文件“stopwatch_CE.ucf”。

2. 利用约束编辑器设置约束属性。

约束编辑器的约束属性分为全局约束 (Global)、端口约束 (Ports)、高级约束 (Advanced)、专用约束 (Misc) 等 4 个选项卡。设计约束属性的基本方法是先对需要约束的对象进行分组，然后对分组进行约束。

“计时表”设计比较简单，本例仅演示一些基本的约束设计。

• 全局约束

设置全局时钟信号“clk”的周期为 16ns，复位信号“rst_int”的周期为 50ns，并设置管脚“Pad to Pad”时间为 20ns。在约束编辑器的约束命令显示窗可以看到相应约束的 UCF 命令声明方法，如图 9-70 所示。

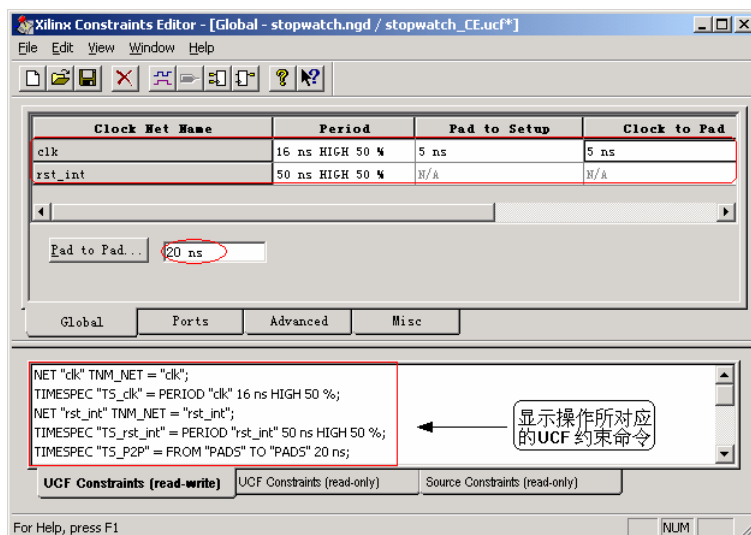


图9-70 全局约束

• 端口约束

在端口约束中主要约束与 I/O 端口相关的“Pad to Setup”和“Clock to Pad”时间与管脚位置约束。双击输入时钟“clk”的位置属性域“Location”，弹出位置锁定对话框，按图 9-71 所示填写锁定管脚，注意“clk”使用了全局时钟资源（BUFG）所以必须锁定在全局时钟输入端口上。

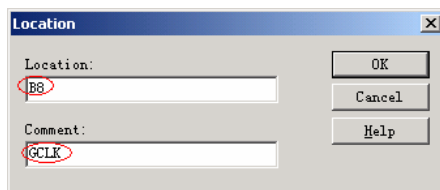


图9-71 管脚锁定

如何对一组具有相同地位的信号进行相同的属性约束呢？对这种信号的基本约束方法是先创建分组，然后对分组进行约束。下面演示如何创建管脚分组，并对分组进行约束。单击端口信号“onesout<0>”，按 **Shift** 键的同时，用鼠标单击端口信号“onesout<5>”，选择从“onesout<0>”到“onesout<5>”之间的所有信号，在端口分组名称“Group Name”域填入“onesout_OutPort”作为分组名称，单击 **Create Group** 按钮，创建分组，如图 9-72 所示。

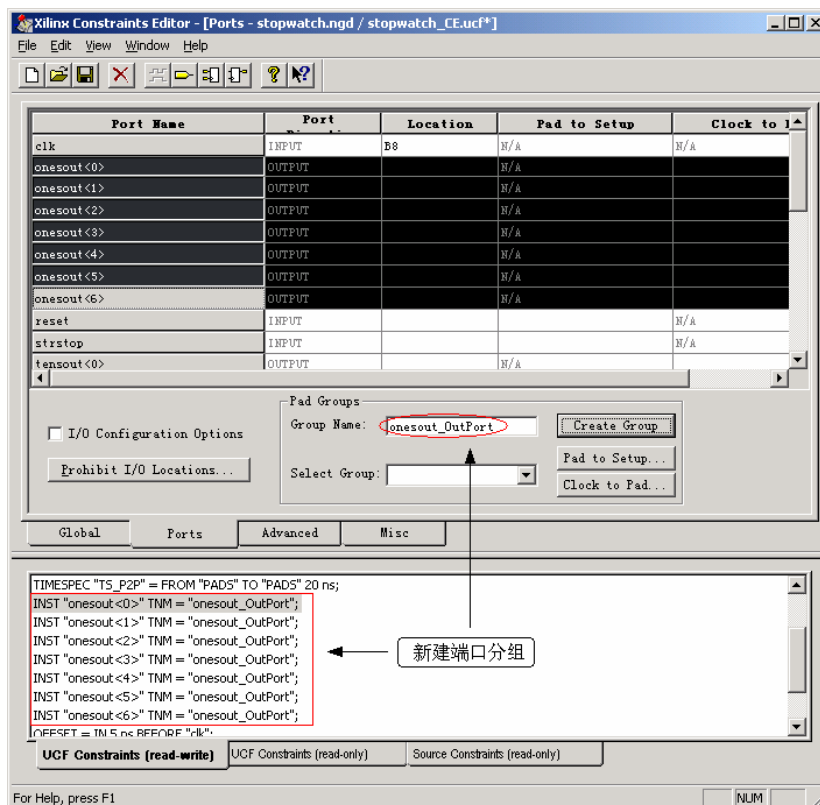


图9-72 新建管脚分组

分组创建后就可以对分组进行约束了。在选择分组（Select Group）下拉列表中选择刚才创建的“onesout_OutPort”分组，单击 **Clock to Pad...** 按钮，启动【Clock to Pad】约束对话框，如图 9-73 所示，设置约束参数。然后单击 **OK** 按钮确认约束。

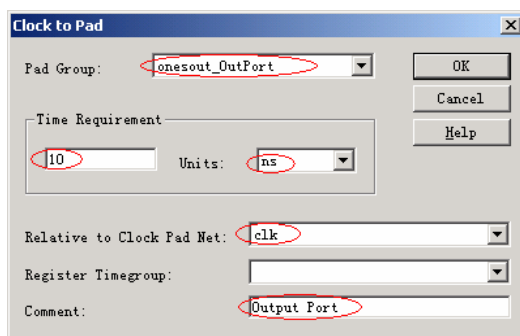


图9-73 对分组进行约束

同理，可以对“tensout”总线创建分组，并约束“Clock to Pad”时间。

- 高级约束

高级约束的基本约束方法也是“先创建分组，然后对分组进行约束”，这种思路是 Xilinx 实现各种约束设置的基本思路。高级约束创建分组的方法灵活：可以通过与某根网线

的连接关系，与输出网线相关联，指定实例名称，指定某个结构层次（Hierarchy），指定时间“THRU”节点等多种方式创建分组。对分组可以附加的约束也多种多样，如“OFFSET IN BEFORE”、“OFFSET IN AFTER”、“FROM TO TIG”和“NET TIG”等约束属性。这些约束属性的具体含义参见第 5 章。

- 专用约束

专用约束主要指一些与 Xilinx 器件特性相关的约束项目。包括 IOB 寄存器，异步模式寄存器，第二全局时钟，RAM/ROM、基于查找表的移位寄存器（SRL16）、寄存器/锁存器等的初值设置，管脚驱动电压，环境温度等约束参数。

在约束编辑器中设置好全局约束、端口约束、高级约束和专用约束后，单击  按钮保存用户约束文件，退出约束编辑器。如果不再更改用户约束文件，就可以重新执行实现步骤，并使用 UCF 文件指导新的实现步骤。

9.5.2 使用 PACE 设计 UCF

引脚与区域约束编辑器（PACE，Pinout and Area Constraints Editor）是 ISE 5.1 的新增工具，完成 I/O 管脚和布局区域的约束，是约束编辑器（Constraints Editor）的有益补充。它的输入、输出文件与 Constraints Editor 一样，也是 NGD 和 UCF 文件。

在 ISE 5.1 以前版本进行约束引脚是一件相当烦琐的事情，而在 PACE 中完成引脚约束却十分方便。在 PACE 的图形化界面下可以方便地完成 I/O 管脚位置约束、I/O Bank 约束、指定 I/O 标准、I/O 管脚位置禁止使用约束、I/O 管脚电气特性约束等与引脚相关的约束属性设置。约束完成后，PACE 还会自动使用设计规则检查（DRC，Design Rule Check）进行约束属性的合法性校验。

PACE 的另外一个重要功能是进行布局区域约束。PACE 从 NGD 文件中提炼出设计的层次结构，显示 I/O 与外设模块连接情况，帮助用户在图形界面下对设计的布局进行调整与约束。适当地使用布局区域约束不仅能提高布局布线的合理性，还能更好地支持增量化综合、实现设计流程。

由于篇幅限制，本例仅介绍如何使用 PACE 进行引脚位置锁定，其他约束项目请参考第 5 章。

使用 PACE 进行引脚位置锁定，参考示例详见本书附带光盘的“Example-9-1”目录

1. 启动 PACE。

刚才已经使用 Constraints Editor 进行了时序约束，生成了 UCF 文件，这时可以直接启动 PACE；如果一个新工程中没有 UCF 文件，首先必须新建 UCF 文件，方能启动 PACE。

在资源管理窗选中设计的顶层模块“stopwatch”，展开当前资源操作窗中“Design Entry Utilities”项目，选择【Assign Package Pins】命令就可以启动 PACE 进行引脚位置锁定。

PACE 主要由设计层次窗（Design Hierarchy Window）、设计对象列表窗（Design Object List Window）、器件结构窗（Device Architecture Window）、引脚封装窗（Package Pins Window）和引脚封装图例窗（Package Pins Legend Window）等部分组成。

2. 约束引脚位置。

PACE 约束引脚位置的基本方法有两种：

第一种引脚锁定的方法是在【Design Object List Window】窗口双击信号的位置属性（Location）选项，直接指定引脚位置。这种直接指定引脚位置的约束方法与在 Constraints Editor 的端口约束选项卡中设置引脚位置的方法基本一致。另外需要指出的是，前面在 Constraints Editor 的端口约束选项卡中指定的“clk”信号管脚为“B8”仍然有效，并且自动在 PACE 中 I/O 管脚位置窗显示出来。如图 9-74 所示，使用直接指定法锁定输出总线“onesout”的引脚位置。

	I/O Name	I/O Direction	Location	Bank	I/O Std.
☐	strstop	Input			LVTTL
☐	reset	Input			LVTTL
☐	onesout(6)	Output	C1	BANK7	
☐	onesout(5)	Output	D1	BANK7	
☐	onesout(4)	Output	D3	BANK7	
☐	onesout(3)	Output	D2	BANK7	
☐	onesout(2)	Output	E4	BANK7	
☐	onesout(1)	Output	E3	BANK7	
☐	onesout(0)	Output	H4	BANK7	
☐	clk	Input	B8	BANK0	

直接指定 I/O 管脚锁定位置

图9-74 直接指定法锁定引脚位置

第二种引脚锁定的方法是在【Design Object List Window】窗口选定需要约束 I/O 管脚位置的信号，用鼠标拖到引脚封装窗（Package Pins Window）的 I/O 位置。如图 9-75 所示，使用鼠标拖曳法锁定输出总线“tensout”的引脚位置。

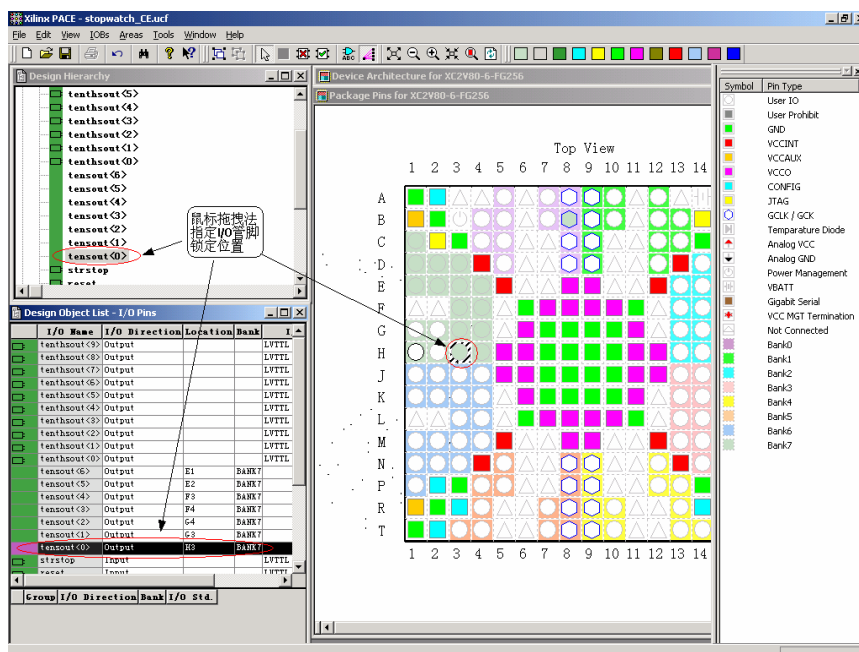


图9-75 鼠标拖曳法锁定引脚位置

引脚位置锁定完成后，单击  按钮保存用户约束文件，退出 PACE。PACE 自动将刚才添加的引脚位置约束属性附加在当前工程目录中的 UCF 文件后面。展开当前资源操作窗中的

【Design Entry Utilities】项目，选择【Edit Constraints (Text)】命令使用 HDL 编辑器编辑本工程 UCF 文件，如图 9-76 所示。重新进行实现步骤，使用新的 UCF 文件指导实现过程。

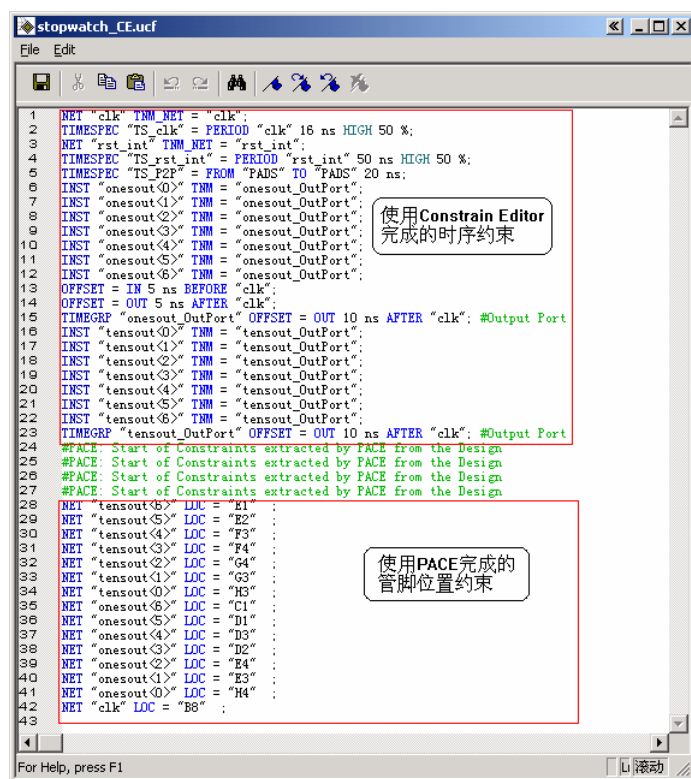


图9-76 使用 Constraints Editor 和 PACE 设计的 UCF 文件

9.5.3 实现步骤与实现结果分析

所谓实现（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。从上面定义可以看出，实现主要分为 3 个步骤：翻译（Translate）逻辑网表，映射（Map）到器件单元与布局布线（Place & Route）。ISE 的实现流程也恰恰是这 3 个步骤。

- 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。展开翻译项目可以看到，该项目包括 3 个命令：【Translation Report】用以显示翻译步骤的报告；【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；【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时延，所以有时省略此仿真步骤。
- 映射的主要作用是将设计映射到具体型号的器件上。展开映射项目可以看到，该项目包括如下命令：【Map Report】用以显示映射步骤的报告；【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；【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；【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时延，所以有时也省略此仿真步骤。

- 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。布局布线步骤的命令与工具非常多：【Place & Route Report】用以显示布局布线报告；【Asynchronous Delay Report】用以显示异步实现报告；【Pad Report】用以显示管脚锁定报告；【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；【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；【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；【Analyze Power (XPower)】用以启动功耗仿真器分析设计功耗；【Generate Post-Place & Route Simulation Model】用以产生布局布线后仿真模型，该仿真模型包含的时延信息最全，不仅包含门延时，还包含了实际布线延时。该仿真步骤必须进行，以确保设计功能与 FPGA 实际运行结果一致；【Generate IBIS Model】用以产生 IBIS 仿真模型，辅助 PCB 布板的仿真与设计；【Multi Pass Place & Route】用以进行多周期反复布线；【Back-annotate Pin Locations】用以反标管脚锁定信息。

实现过程的 3 个步骤的属性可以分别设置，也可以统一设置。设置方法是在资源管理窗选择顶层模块文件，在当前资源操作窗中选中相应的步骤，单击工具栏中的  (Edit Process Properties) 按钮编辑该步骤属性。按图 9-77~图 9-81 所示设置实现属性。

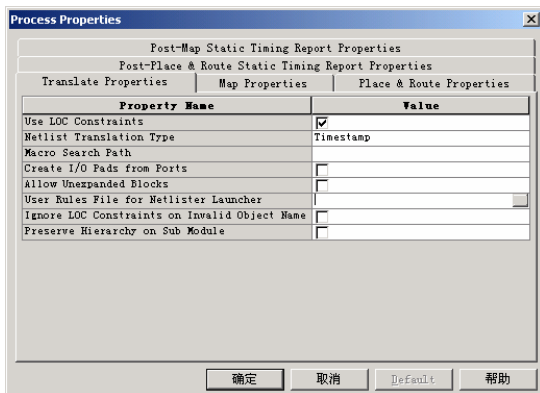


图9-77 设置翻译步骤实现属性参数

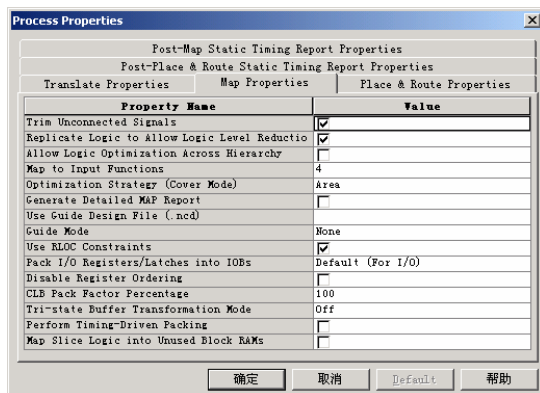


图 9-78 设置映射步骤实现属性参数

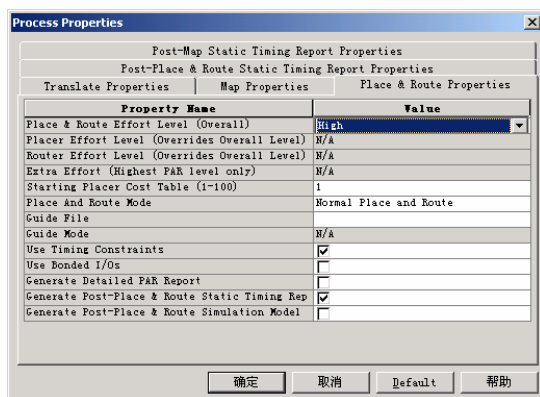


图9-79 设置布局布线步骤实现属性参数

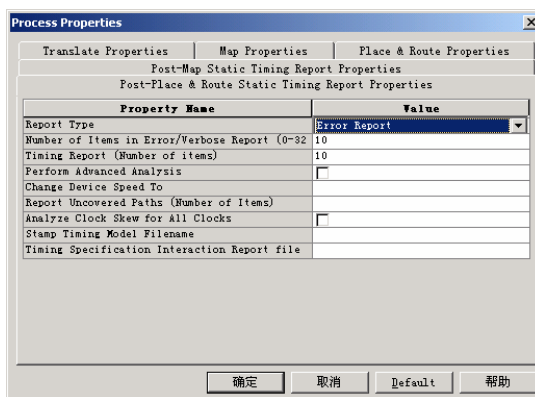


图 9-80 设置布局布线后静态时序属性参数

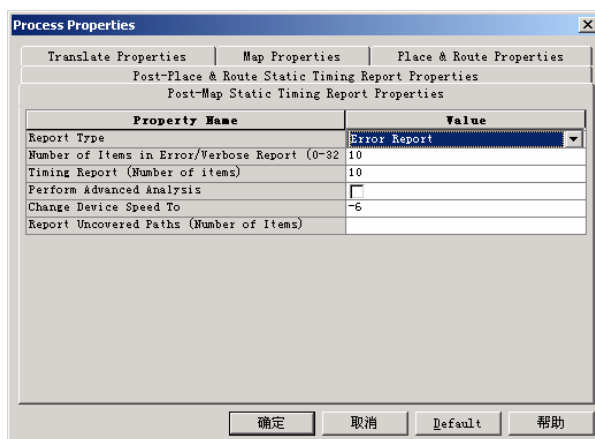


图9-81 设置映射后静态时序属性参数

设置好实现属性后，单击  (Run Process) 按钮进行实现，实现过程的一切信息会在信息显示窗显示。

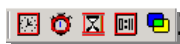
实现结果的分析主要从两个方面入手：一是分析每个实现步骤的报告；二是利用 Xilinx 的实现步骤设计工具，如利用时序分析器、FPGA 底层编辑器、布局规划器和功耗仿真器等分析实现结果。

布局布线后的最终实现结果包含的信息最多，所以仅简单演示布局布线后的实现结果分析。

🔑 布局布线后的实现结果分析，参考示例详见本书附带光盘的“Example-9-1”目录

1. 使用时序分析器分析布局布线后静态时序。

时序分析器以图形化界面辅助设计者进行不同实现阶段的静态时序分析。展开当前资源操作窗的实现项目，双击布局布线步骤下的【Analyze Post-Place & Route Static Timing (Timing Analyzer)】命令，启动时序分析器。使用



工具栏启动不同类型的时序分析报告，如图 9-82 所示。

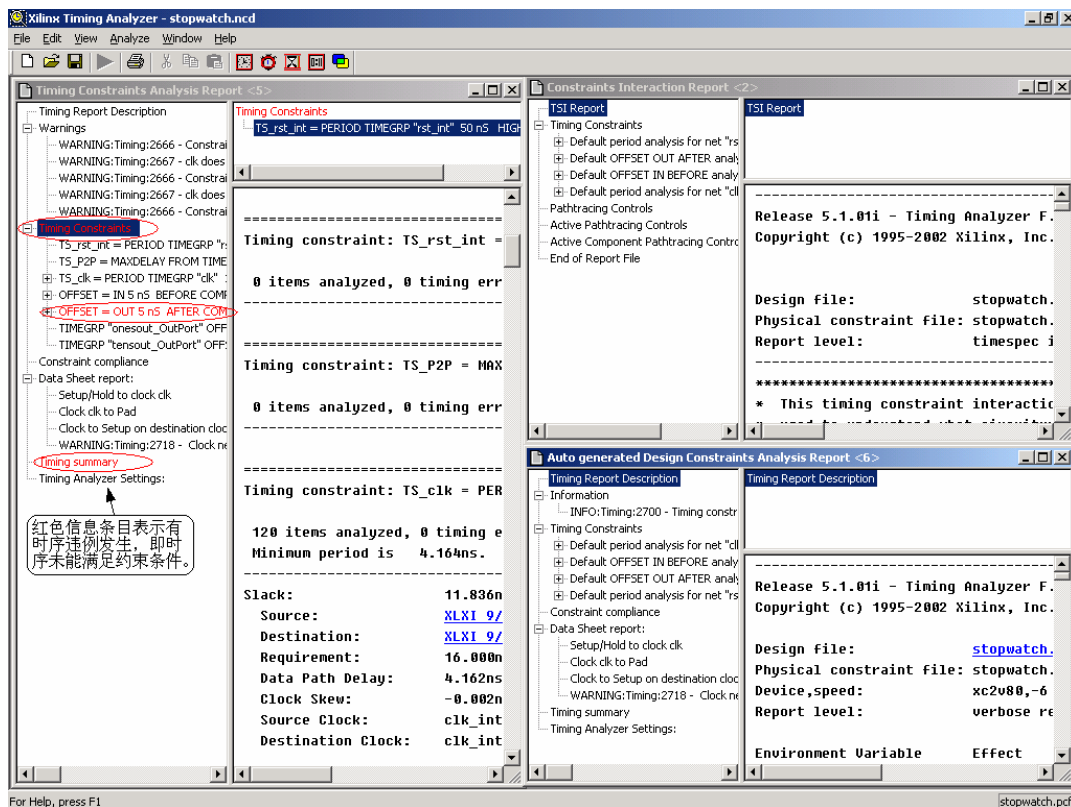


图9-82 使用 Timing Analyzer 分析布局布线后静态时序

2. 使用布局规划器分析布局布线后的实际布局结构。

Floorplanner 以图形界面显示设计在 FPGA 的基本硬件结构和底层原语上的布局情况。Floorplanner 自 ISE 4.1 起新增了 Cross-Probing 功能,可以在设计结构层次、Floorplanner 布局图、Timing Alayzer 的时序报告之间方便地进行互链切换,帮助设计者更好地调整设计布局,从而提高实现结果的工作频率。展开当前资源操作窗的实现项目,双击布局布线项目下的【View/Edit Place Design (Floorplanner)】命令,启动 Floorplanner。图 9-83 所示为本设计的 Floorplanner 布局图。

3. 使用 FPGA 底层编辑器分析布局布线后的实际布局布线情况。

FPGA Editor 是 Xilinx 最底层的手动布局布线工具,通过这个工具可以编辑或者查看可配置逻辑功能块 (CLB)、输入输出功能块 (IOB) 等一些 FPGA 内部的基本单元。在复杂和较高要求的设计中,适当地使用 FPGA Editor 指导布局布线,能最大程度地提高设计的密度和性能指标。展开当前资源操作窗的实现项目,双击布局布线项目下的【View/Edit Place Design (FPGA Editor)】命令,启动 FPGA Editor。图 9-84 所示为本设计的 FPGA Editor 布局布线图。

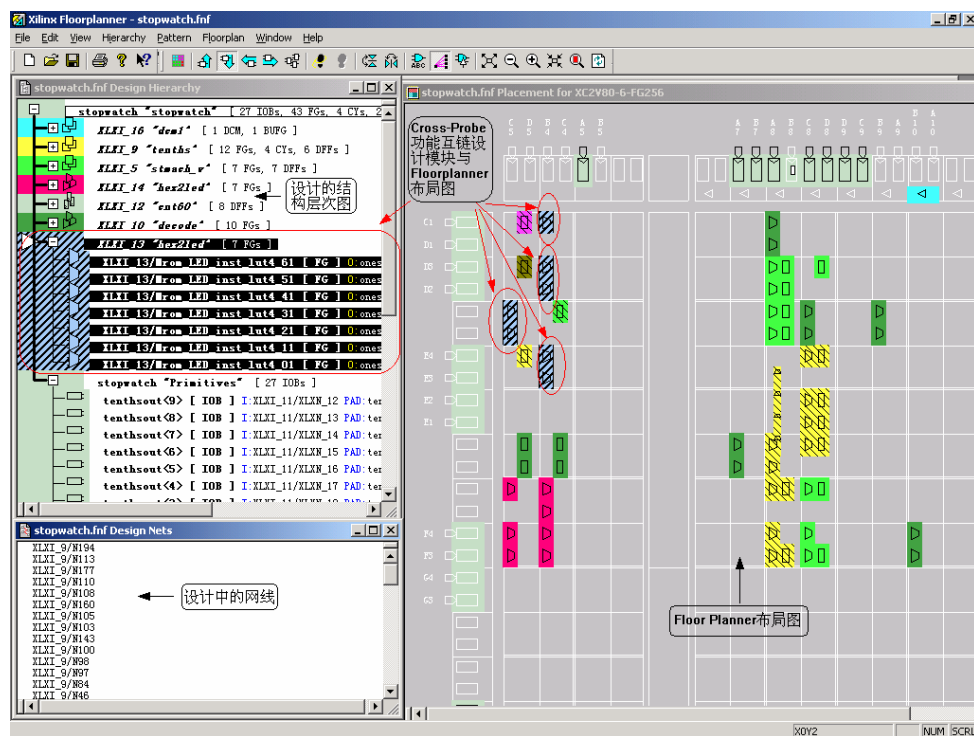


图9-83 “计时表”设计的 Floorplanner 布局图

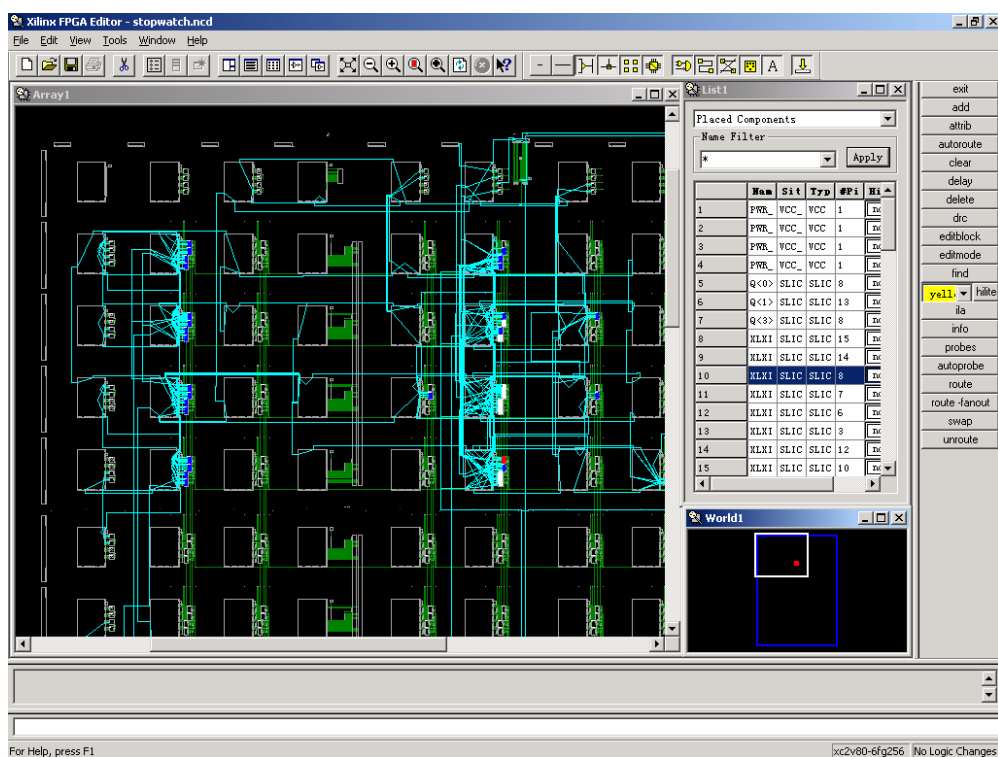


图9-84 “计时表”设计的 FPGA Editor 布局布线图

4. 使用功耗仿真器 (XPower) 分析设计功耗。

XPower 是 Xilinx 的功耗仿真工具, 用以仿真不同工作条件下设计的功耗。通过该工具计算出 FPGA/CPLD 的大致功耗, 作为设计 PCB 电源模块功率的参考值。展开当前资源操作窗的实现项目, 双击布局布线项目下的【Analyze Power (XPower)】命令, 启动 XPower 进行功耗仿真, 如图 9-85 所示。

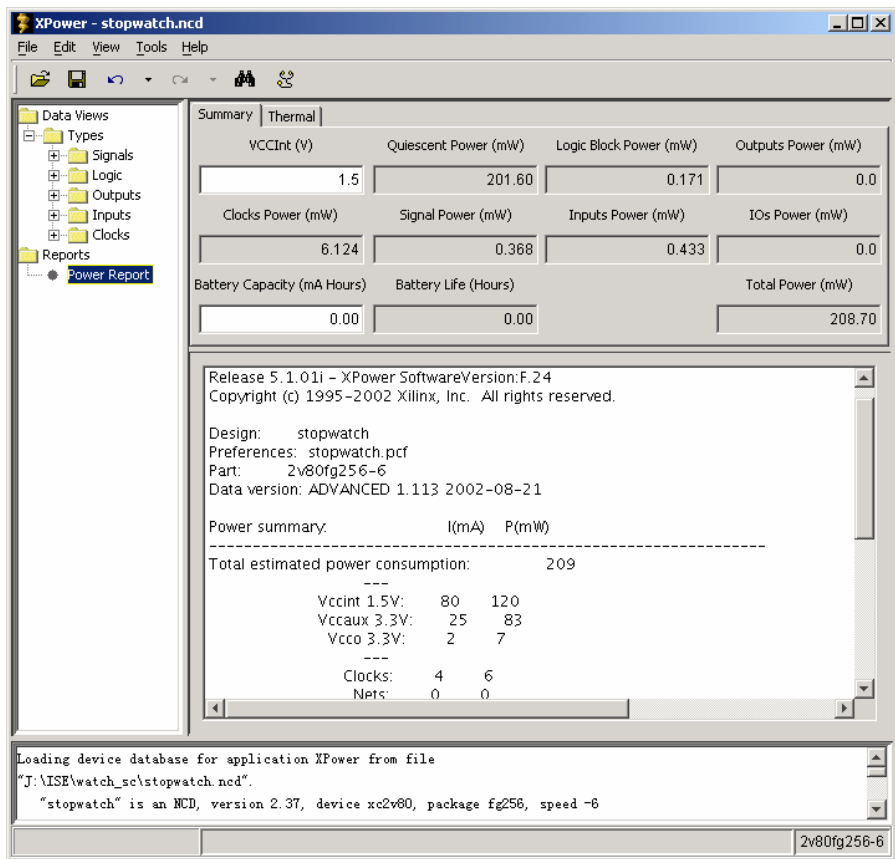


图9-85 使用 XPower 分析“计时表”设计的功耗

9.6 使用 ModelSim 进行布线后仿真

将布局布线的时延信息反标到设计网表中, 所进行的时序仿真就叫布局布线后仿真, 简称布线后仿真。布局布线之后生成的仿真时延文件包含的时延信息最全, 不仅包含门延时, 还包含实际布线延时, 所以布线后仿真最准确, 能较好地反映芯片的实际工作情况。布线后仿真步骤必须进行, 以确保设计功能与 FPGA 实际运行情况相一致。

展开当前资源操作窗的实现项目, 双击布局布线项目下的【Generate Post-Place & Route Simulation Model】命令, 产生布局布线后仿真模型。Xilinx 自动产生的仿真模型名为“PrjName_timesim.v”或“PrjName_timesim.vhd”。仿真延时信息文件名为“PrjName_timesim.sdf”, 其中“PrjName”为工程名称。后仿真模型文件自动调用 SDF 延

时文件，将延时信息反标到仿真模型中去。

由于设计中用到了 Xilinx IP Core 等专用结构与硬件原语，所以使用 ModelSim 仿真时必须调用 Xilinx 仿真库。仿真库的设置方法请参见 3.3 节中的论述。

本例采用 HDL Benchner 产生测试激励，而 HDL Benchner 在生成测试激励的同时自动生成 ModelSim 仿真批处理命令文件，该文件包含了编译测试激励源代码和仿真对象源代码。调用 Xilinx 仿真库，打开波形观察窗、结构层次窗、信号列表窗等窗口，添加信号，启动仿真等多条批处理命令。如果使用 HDL Benchner 设计测试激励，调用 ModelSim 进行仿真就非常容易操作。选择测试激励波形资源“stopwatch_top_tb.tbw”，看到当前资源操作窗中对测试激励波形资源的可用命令是各种类型的仿真操作。双击布线后仿真【Simulate Post-Place & Route Verilog Model】命令（使用 VHDL 综合流程的请调用 VHDL 后仿真命令），ISE 自动调用 ModelSim 完成布线后仿真，仿真结果如图 9-86 所示。

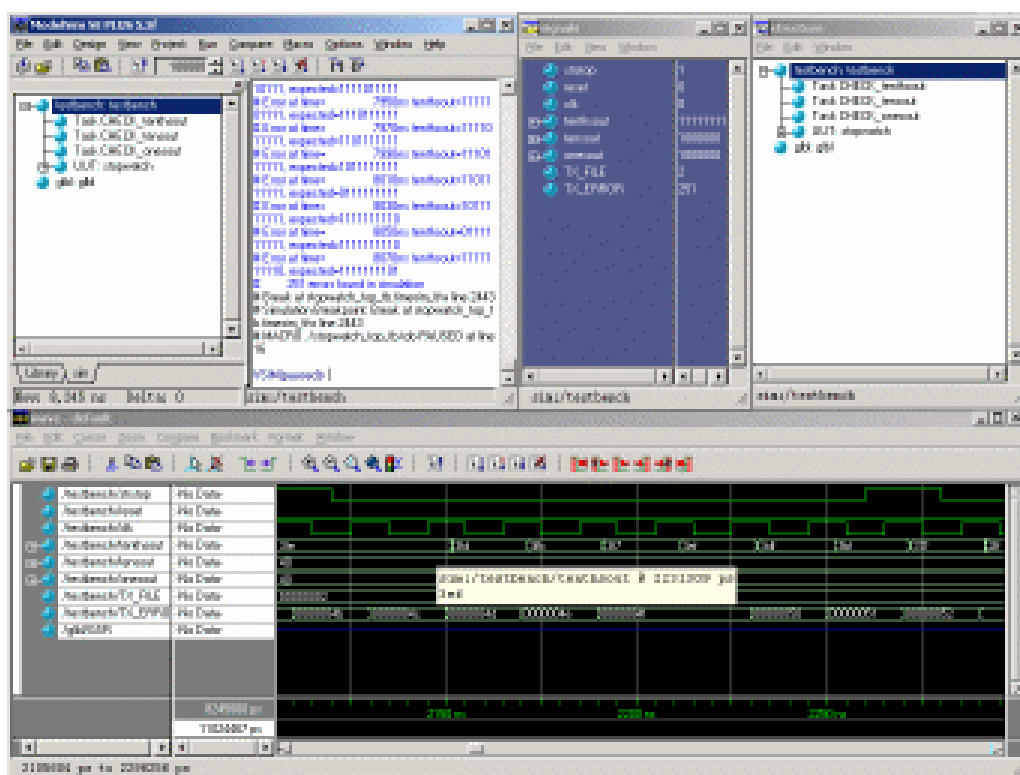


图9-86 使用 ModelSim 进行布线后仿真

9.7 使用 iMPACT 配置 FPGA/CPLD

iMPACT 是 ISE 5.x 集成的配置工具，用于生成 PROM 格式的配置文件、向 FPGA/CPLD/PROM 下载配置文件、回读和验证配置的数据是否正确以及调试配置过程出现的问题。在 ISE 5.1 以前版本中，PROM 配置文件分割器（PROM File Formatter）和 iMPACT 配置器是两个独立工具，自 ISE 5.1 版本起，两者合二为一，使生成配置文件和配

置 FPGA/CPLD 的过程更加符合操作习惯。

iMPACT 支持 4 种配置模式：边界扫描（Boundary-Scan）模式、从串模式（Slave Serial）模式、SelectMap 模式和 Desktop 配置模式。其中边界扫描模式标准统一，设备简单，可以通过 JTAG 口配置 FPGA/CPLD/PROM 等多种器件，最为通用。本节将以边界扫描模式为例演示如何使用 iMPACT 配置 FPGA/CPLD/PROM。

PROM 配置文件分割器支持的文件格式有 4 种：Intel MCS-86 文件格式（扩展名为“mcs”）、Tektronix TEKHEX 文件格式（扩展名为“tek”）、Motorola EXORmacs 文件格式（扩展名为“exo”）和 HEX 文件格式（扩展名为“hex”）。

本例使用芯片型号为 XC2V80，其配置文件大小为 635360bit，使用一片 XC18V01 PROM 就可以装载该配置文件，所以不需要进行配置文件的分割。假设本例采用边界扫描模式连接 PROM XC18V01 和 FPGA XC2V80，下面简单演示如何使用 iMPACT 进行 FPGA/CPLD 的配置。

✎ 使用 iMPACT 下载配置文件到 XC18V01 PROM，参考示例详见本书附带光盘的“Example-9-1”目录

1. 生成配置文件的选项。

- (1) 在资源管理窗选择顶层模块文件“stopwatch”，选中当前资源操作窗中的“Generate Programming File”的项目，单击工具栏中的 （Edit Process Properties）按钮设置配置文件生成属性。在配置文件生成属性中，最重要的属性之一是 FPGA 初始化时钟（FPGA Start-Up Clock）设置。如果需要下载 BIT 配置文件到 FPGA，则需选择【JTAG Clock】选项，如果需要下载到 PROM，则需选择【CCLK】选项，如图 9-87 所示。

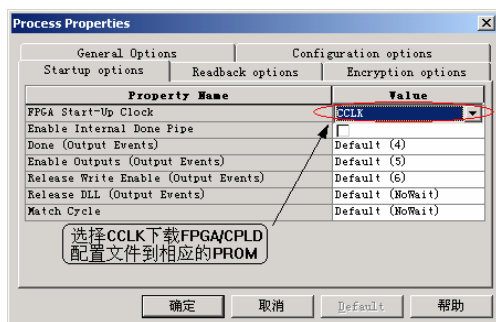


图9-87 FPGA 初始化时钟设置

- (2) 设置完配置文件生成属性后，单击工具栏中的  按钮，生成 FPGA/CPLD 配置文件。
2. 转换配置文件格式。
- (1) 配置文件生成后，展开当前资源操作窗中的“Generate Programming File”项目，双击【Generate PROM,ACE,or JTAG File】命令，启动 iMPACT。选择操作模式为【Prepare Configuration】，如图 9-88 所示。
- (2) 单击  按钮，按图 9-89 所示选择配置文件类型为【PROM File】。

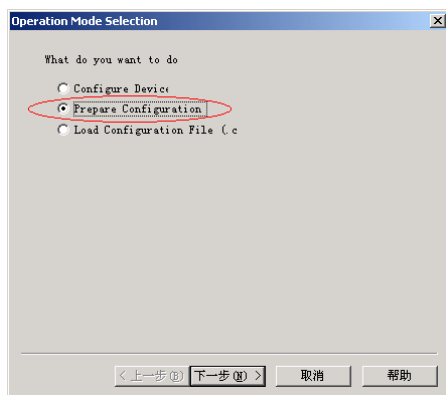


图9-88 选择操作模式

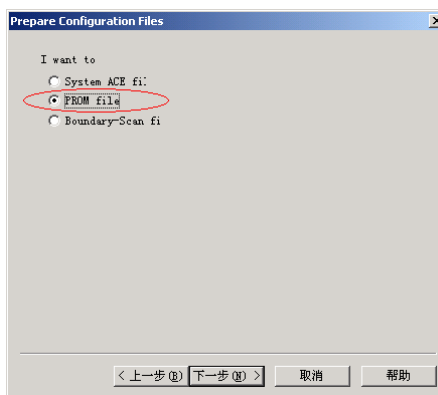


图 9-89 选择配置文件类型

- (3) 单击 **下一步(N) >** 按钮，按图 9-90 所示配置 PROM 文件参数。
- (4) 单击 **下一步(N) >** 按钮，按图 9-91 所示选择 PROM 器件型号，单击 **Add** 按钮把 PROM 添加到列表中，在列表中可以添加多个 PROM，直到满足 FPGA/CPLD 配置文件的容量要求。

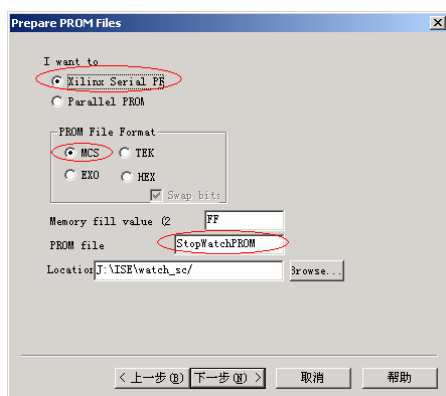


图9-90 配置 PROM 文件参数

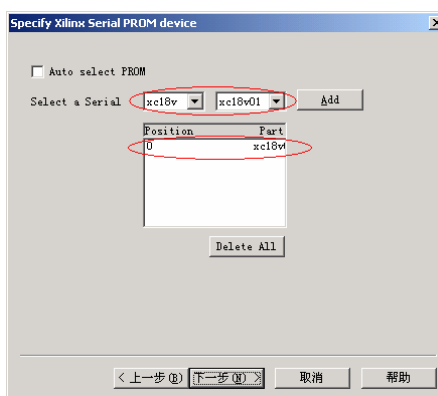


图 9-91 选择 PROM 器件型号

- (5) 单击 **下一步(N) >** 按钮显示配置文件生成信息。单击 **下一步(N) >** 按钮确认该信息，如图 9-92 所示，添加 BIT 文件“stopwatch.bit”。

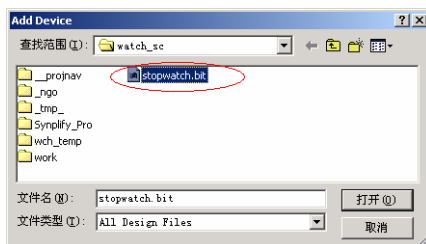


图9-92 添加 FPGA/CPLD BIT 配置文件

- (6) iMPACT 根据 BIT 文件内容，提取器件信息，自动生成器件配置连接图。接着 iMPACT 会询问是否需要继续添加 BIT 文件，该选项在多个 BIT 文件合并

配置时使用，本设计仅使用一片 FPGA，所以选择 **否(N)** 按钮。

- (7) BIT 文件添加完成后，单击 **完成** 按钮，iMPACT 会弹出【PROM File Generation】对话框，询问是否生成 PROM 文件，单击 **是(Y)** 按钮，iMPACT 会提示是否需要改变 PROM 文件的文件名。由于文件名在之前的步骤已经设置好，因此单击 **否(N)** 按钮，iMPACT 将生成 PROM 文件，如图 9-93 所示。

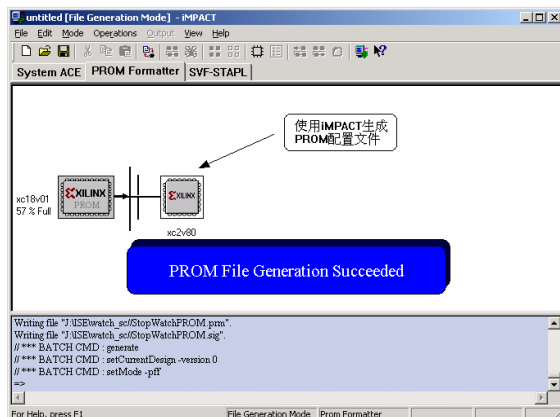


图9-93 使用 iMPACT 生成 PROM 配置文件

- (8) 配置文件生成后，如果生成的配置图恰好与用户 PCB 上实际的配置方式一致，单击 **保存** 按钮，保存 FPGA/CPLD、PROM 配置图。在使用 iMPACT 配置 FPGA/CPLD、PROM 时，直接调用该配置文件下载配置，省去了重新检测连接方式、指定配置文件的麻烦。
3. 使用 iMPACT 下载配置文件。
- (1) 使用 iMPACT 下载配置文件的过程与生成配置文件的过程相似。展开当前资源操作窗中的“Generate Programming File”项目，双击【Configure Device (iMPACT)】命令，启动 iMPACT。
- (2) 按图 9-94 所示选择操作模式为【Configuration Device】。
- (3) 单击 **下一步(N) >** 按钮，按图 9-95 所示选择配置连接方式为【Boundary-Scan Mode】。

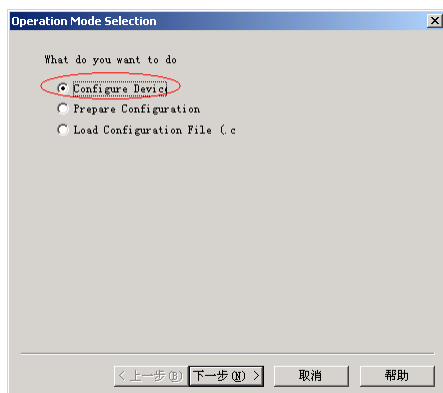


图9-94 选择操作模式

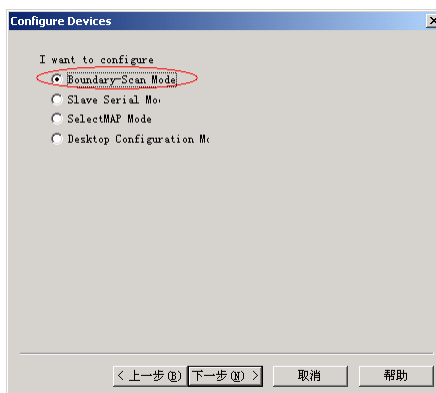


图 9-95 选择配置连接方式

- (4) 单击 **下一步(N) >** 按钮，按图 9-96 所示选择自动连接到下载电缆并检测 Boundary-Scan 连接链 “Automatically connect to cable and identify Boundary-Scan chain”。

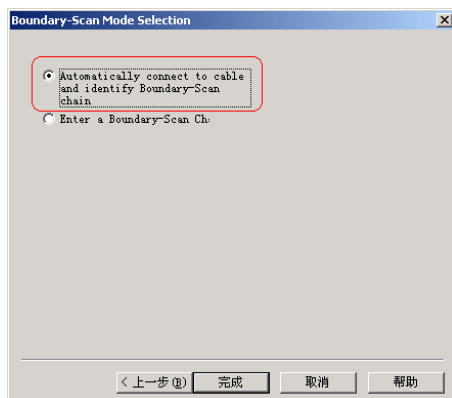


图9-96 选择 Boundary-Scan 连接检测方式

- (5) 单击 **完成** 按钮，iMPACT 将自动连接到下载电缆并检测 Boundary-Scan 连接链，显示边界扫描链的结构，同时提示将为边界扫描链中的器件指定配置文件。单击常用工具栏的  (Debug Chain) 按钮可以完成相同的功能。
- (6) 完成配置文件指定后，用鼠标选中器件，单击鼠标右键弹出命令菜单，选中 **【Program】** 命令，按图 9-97 所示设置编程属性，单击 **OK** 按钮对 PROM 进行编程。

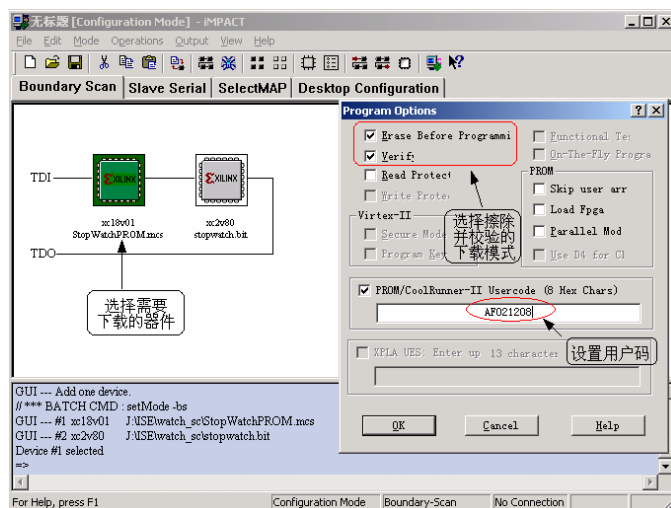


图9-97 FPGA/CPLD、PROM 下载配置图



用户可以对 PROM 和 CoolRunnerII 器件族设置用户码 (User Code)，该码为 8 个 16 进制数，对用户码进行意义规定（如版本日期、性质等）并在下载 PROM 时烧写用户码，能帮助用户更好地进行设计的版本管理。

至此，这个“运动计时表”设计实例全部演示完毕。

9.8 小结

本章的实例以 FPGA/CPLD 设计流程为主线，充分利用了 ISE 中的各种设计资源与工具。通过本章的学习，可以加深读者对 ISE 中集成的设计输入、仿真、综合、约束与实现、下载配置等各种工具的认识，从而对 ISE 集成的 FPGA/CPLD 设计工具有一个系统而全面的把握。

9.9 问题与思考

1. 认真体会本章开篇提出的 7 个问题，理解全书脉络。
2. 结合“运动计时表”设计实例，体会 FPGA/CPLD 设计流程。
3. 根据 FPGA/CPLD 设计流程，对 ISE 集成的所有 EDA 设计工具进行分类，并简述这些工具的功能。
4. 对比本书 1~7 章各种设计工具操作实例与“运动计时表”相关部分操作实例，提高自己的实际动手能力。

第10章 ISE 实战——I²C 接口设计

通过前面的学习，相信大家已经熟悉了 ISE 的使用方法，现在可以做一个富有实践意义的设计了。在 FPGA/IC 设计领域，实践经验是至关重要的，概念、理论的学习只是手段，其作用是指导实践，只有把概念、理论运用到实践，设计出实用的东西，才能创造价值，达到学习的目的。

通过本章的学习，读者不仅可以通过实际操作熟练掌握 ISE、Synplify Pro 和 ModelSim 等软件的使用方法，还可以深刻地理解应用广泛的 I²C 协议，掌握微控制器接口设计、状态机设计、混合语言仿真、后仿真、EDIF 设计流程等技术，最后把 BIT 文件下载到 FPGA 实验板中，真正完成自己第一个有挑战性的设计。

本章的特点是紧密结合实践，引导读者边学边做，通过实际设计与上板调试加深对全书的理解。可以说本书第 9 章是一个“软件性”的融会贯通全书的例子，而本章是一个“硬件性”的综合应用全书的例子。因为 I²C 本身具有较高的工程实践意义，是电子、通信系统中常用的总线方式，所以本章所述内容更具备工程指导意义，也能更有效地提高读者实际设计水平。

本章的主要内容包括：

- EFX-SP200 实验系统简介；
- I²C 总线简介；
- I²C 总线应用实例——AT24C 系列 EEPROM；
- I²C 总线控制器设计详解。

10.1 EFX-SP200——实验开发系统简介

为了方便读者学习和练习，真正掌握 FPGA/IC 开发技术，在求职和工作中领先一步，笔者联合经验丰富的专业 EDA 设计人员，结合多年实践经验和切身体会，开发了一系列的 FPGA 实验开发板，并提供了丰富的实例和详尽的讲解，务求使读者快速、高效地掌握 FPGA/IC 开发技术，并通过实践积累丰富的设计经验，其中 EFX-SP200 实验开发系统的结构示意图如图 10-1 所示。

EFX-SP200 实验开发系统具有以下特点：

- 采用 Xilinx 公司 Spartan II 系列的 XC2S200-PQ208 作为核心芯片，规模为 20 万门，管脚数多达 208 个，满足复杂设计的需要；
- 具有 100MHz 的 SDRAM，可以完成高速 SDRAM 控制器设计；
- 具有 32KB 静态 RAM，为系统设计提供方便的大容量内存；
- 支持 I²C 接口，板上有 I²C 接口串行 EEPROM—AT24C02；

- 两个 RS232 接口，满足各种通信应用需求；
- 具有两个 PS/2 和 VGA 接口、7 段 LED、多个按钮等丰富的人机接口；
- 可利用 Xilinx 丰富的 IP Core 资源构建 RISC CPU、PCI 接口、PicoBlaze 处理器、数字信号处理等高级应用；
- 独有 80C51 单片机，可以完成 SOC（片上系统）设计实验，体会合理的软硬件划分给设计带来的巨大好处。

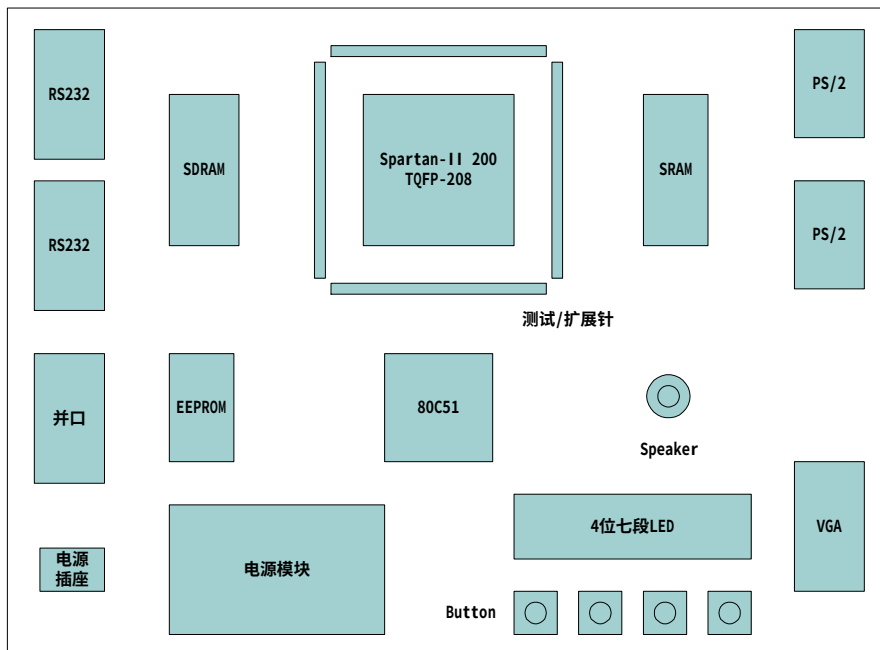


图10-1 EFX-SP200 结构示意图

如果读者想了解 EFX-SP200 实验开发系统的进一步介绍，可以登陆“专业 EDA 论坛” (<http://www.edacn.net>)。

本章后面的内容详细讲解了 I²C 总线协议和 I²C 总线控制器的设计，建议读者认真阅读和亲自完成整个设计过程，并在 EFX-SP200 实验板上调试、验证自己的设计。

10.2 I²C 总线简介

串行总线和并行总线相比具有结构简单、占用引脚少、成本低的优点。常见的串行总线有 USB、IEEE1394、1-wire 和 I²C 等，其中 I²C 总线具有简单实用的特点，在单片机、串行 EEPROM、LCD 等器件中有广泛的应用。

I²C 总线的全称是 Inter IC BUS，是 Philips 公司开发的用于芯片之间连接的总线，I²C 总线使用两根信号线来进行数据传输，一根是串行数据线（SDA），另一根是串行时钟线（SCL）。它允许若干兼容器件（如存储器、A/D 和 D/A 转换器、LCD 驱动器等）共享总线。I²C 总线理论上可以允许的最大设备数，是以总线上所有器件的电容总和不超过 400pF 为限（其中包括连线本身的电容和与它连接端的引出电容），总线上所有器件依靠 SDA 发送的地址信号寻址，不需要片选线。任何时刻总线只能由一个主器件控制，各从器件在总线空

闲时启动数据传送。I²C 总线数据传输的标准速率为 100kbit/s，最高速率为 400kbit/s。I²C 总线的一个典型应用如图 10-2 所示。

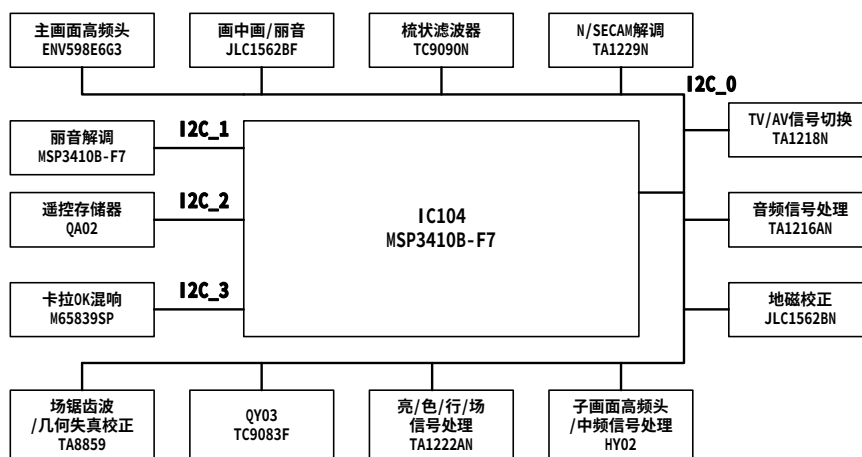


图10-2 I²C 总线彩电机芯的芯片互连

图 10-2 所示为 NC-6 彩电机芯的芯片互连图，CPU 和外围各芯片通过 4 组 I²C 总线连接起来。该机芯用于较高档的遥控彩电，如长虹 G2966、G2967 等。

I²C 总线的另一个典型应用是存储器 IC 卡，请见 10.2.4 节。I²C 总线还有很多应用，Philips Semiconductor 等半导体厂商提供了大量 I²C 总线接口的器件，在 <http://www.semiconductors.philips.com/buses/i2c/products/index.html> 上就有超过 100 种，从中读者可以看到 I²C 总线在各领域中的应用。

下面对 I²C 总线规范做一个简要的介绍，以便读者尽快掌握。完整的 I²C 规范可以到 <http://www.edacn.net> 下载。

10.2.1 I²C 总线上的数据传输

I²C 总线包含 SCL 和 SDA 两条连线，SCL 为时钟线，用于传送时钟信号，时钟信号由主机产生；SDA 为数据线，用于传送数据。I²C 总线的传输格式如图 10-3 所示。

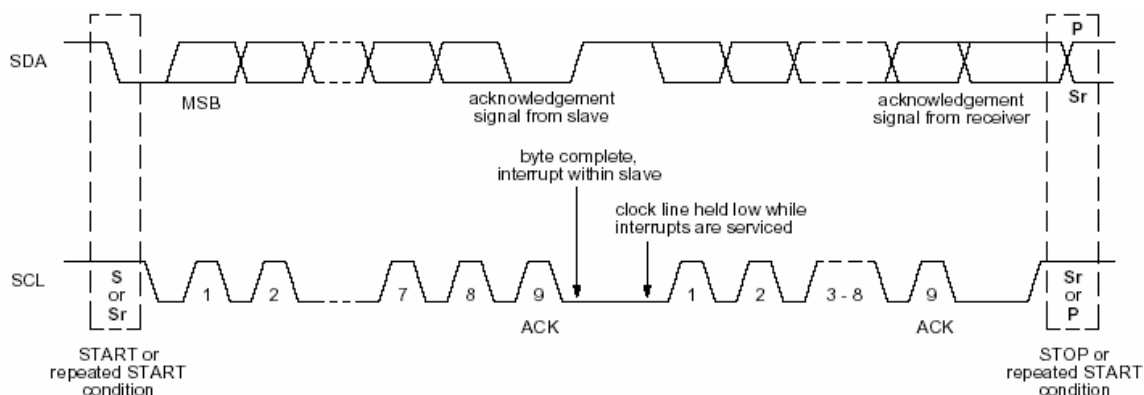


图10-3 I²C 总线的数据传输

从图 10-3 可知 I²C 总线传送数据的大致过程，首先主机产生起始位，然后传送 8 比特数据，从机收到后发响应位，主机收到响应位后接着发送第二字节的数据。数据发送完毕后产生结束位，数据传送结束。

I²C 总线的位传输如图 10-4 所示。时钟线 SCL 为低电平时 SDA 才允许切换，SCL 为高电平时，SDA 必须稳定，此时 SDA 的电平就是 I²C 总线传送的数值。

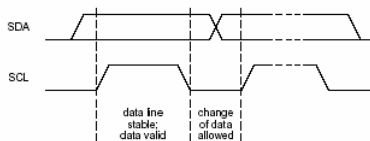
图10-4 I²C 总线的位传输

图 10-5 所示为 I²C 总线的起始位和停止位的示意图，在 SCL 线是高电平时，SDA 线从高电平向低电平切换表示起始位，SDA 线由低电平向高电平切换表示停止位。

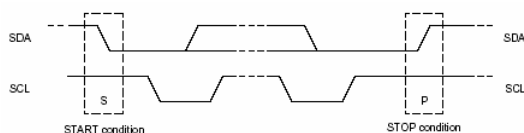
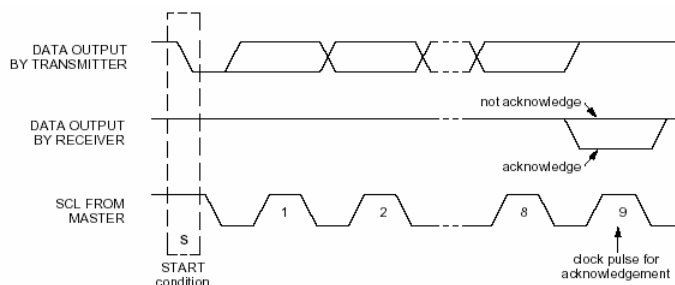


图10-5 起始位和停止位示意图

起始位和停止位由主机产生，在起始位出现后总线处于忙的状态，停止位出现并经过一定时间后总线进入空闲状态。

发送器每发送一个字节后，接收器必须产生一个响应位。响应位的时钟脉冲由主机产生，接收器将 SDA 线拉低产生响应位，如图 10-6 所示。

图10-6 I²C 总线的响应

如果主机是接收器，接收最后一个字节时，不产生响应位（或者说响应位等于 1），以通知从机结束发送，否则响应位应为 0。

当从机不能响应从机地址（例如它正在执行一些实时函数，不能接收或发送）时，或响应了从机地址，但是在传输了一段时间后不能接收更多数据字节，此时从机可以通过不产生响应位来通知主机终止当前的传输，于是主机产生一个停止位终止传输，或者产生重复起始位开始新的传输。

SDA 线上传送的数据必须为 8 位，每次传输可以发送的字节数量不受限制，首先传输的是数据的最高位 MSB，如果从机要完成一些其他功能（例如执行一个内部中断服务程

序)后,才能接收或发送下一个数据字节,从机可以使时钟线 SCL 保持低电平,迫使主机进入等待状态。从机准备好接收或发送下一个数据字节后,释放时钟线 SCL,数据传输继续。

SDA 和 SCL 都是双向线路,使用时通过上拉电阻连接到电源上,当总线空闲时这两条线路都是高电平,连接到总线的器件输出级必须是漏极开路或集电极开路,如图 10-7 所示,这样总线才能执行线与的功能。

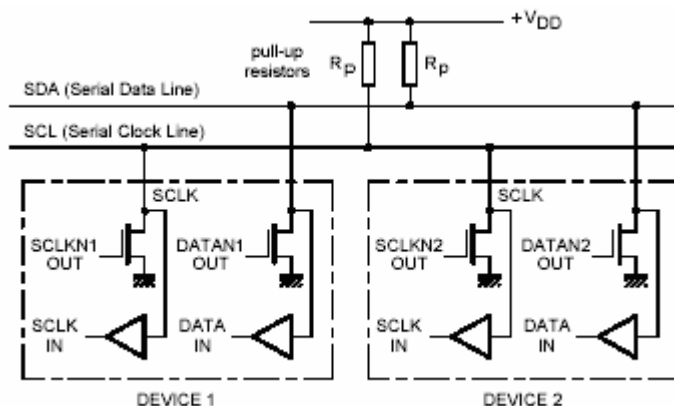


图10-7 I²C 总线和器件的连接



如果只看 SDA 线, I²C 协议和 RS232 协议比较类似,也是首先产生起始位,然后传送数据,数据传送完毕后发出结束位。不同之处在于, I²C 协议要求从机每收到一字节的数据就要产生响应位,而且 I²C 协议要求每次传送 8 比特的数据, RS232 则不需要产生响应位,每次可以传送 5~9 比特的数据, RS232 可以有奇偶校验比特, I²C 没有。

10.2.2 I²C 总线寻址

I²C 总线可以连接多个器件,主机可以通过寻址操作和特定的器件进行通信。I²C 总线通信时,起始位后的第一个字节用于寻址,该字节包含 7 比特的从机地址和 1 比特的读写指示比特,如图 10-8 所示。



图10-8 起始位后的第一个字节

图 10-8 中,字节的高 7 位组成了从机地址,最低位 LSB 决定主机发起的是读还是写操作,1 表示主机读信息,0 表示主机会写信息到被选中的从机。

一般而言,从机地址由一个固定部分和一个可编程部分构成,由于在一个系统中可能有几个同样的器件,从机地址的可编程部分使最大数量的同样器件可以连接到 I²C 总线上,器件可编程地址位的数量由它可使用的管脚决定,例如 AT24C 系列 I²C 接口串行 EEPROM

有 4 个固定的和 3 个可编程的地址位，那么一条 I²C 总线上，最多共可以连接 8 个这种型号的 EEPROM。

现在我们可以看看 I²C 总线完整的数据传输了，它包括主机产生起始位、发送从机地址和读写指示比特、从机响应寻址、数据传输、产生响应比特、主机产生停止位等，如图 10-9 所示。

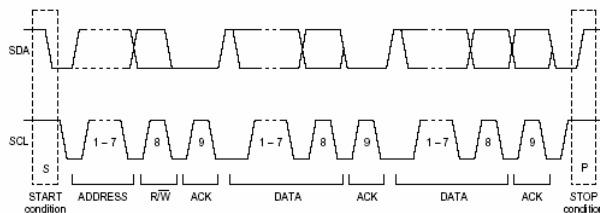


图10-9 完整的数据传输

实际应用中，主机发完第一字节后，数据的传输方向可能会发生变化，仔细研究传输方向的变化有助于对主机、从机、发送器、接收器等概念的理解，数据传输方向的变化有以下 3 种情况。

- 传输方向不变。主机发出第一字节，从机响应后，主机发送数据到从机，此时第一字节和之后传送的字节方向一样，都是从主机到从机，如图 10-10 所示。主机向从机写数据属于这种情况。

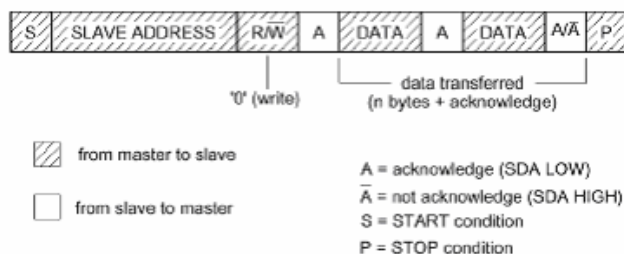


图10-10 传输方向不变

- 传输方向改变。主机发出第一字节，从机响应后，主机变为接收状态，从机变为发送状态，数据的传输方向发生变化，如图 10-11 所示。主机从从机读数据属于这种情况。

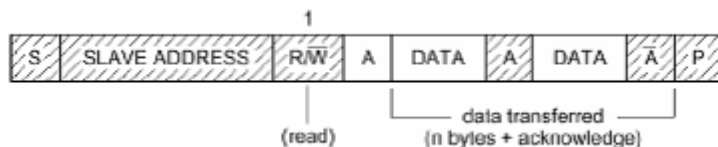


图10-11 传输方向改变

- 复合格式。此时主机连续对从机进行多次读写操作，因此在产生起始位、收发数据、产生停止位的整个传输过程中，数据的方向发生多次改变，传输改变

方向时，主机会重新发出重复起始位和从机地址，如图 10-12 所示。

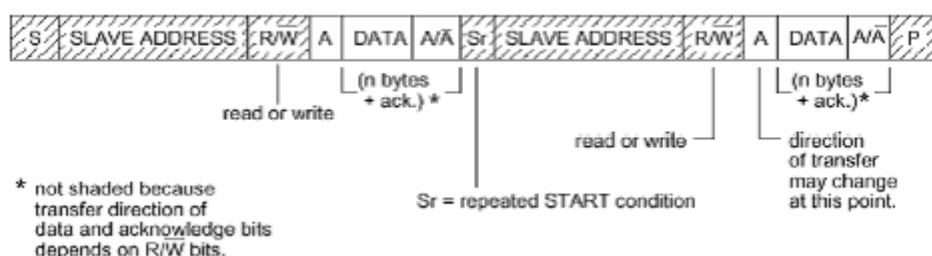


图10-12 数据传输的复合格式

10.2.3 时钟同步与仲裁

I²C 总线在一个时刻只能有一个主机，当 I²C 总线同时有两个或更多的器件想成为主机时，就需要进行仲裁，时钟同步过程的目的是为仲裁提供一个确定的时钟。

如果 I²C 总线上只有一个主机，它驱动 SCL 线的过程是，SCL 线由高变低后主机将以内部工作时钟周期为单位开始计数，直到满足 I²C 协议对 SCL 低电平持续时间的要求，然后释放 SCL 线，SCL 线变高之后，主机以内部工作时钟周期为单位开始计数，直到 SCL 高电平持续时间满足协议要求，然后再拉低 SCL 线，不断重复，就可以产生 SCL 时钟信号。

I²C 总线可以有多个主机，当这些主机发起通信时，都会按上述方式在 SCL 线上产生时钟信号，这些时钟信号的同步通过线与来执行。例如在图 10-13 中，主机 1 产生的时钟 CLK1 首先变低，SCL 线随之被拉低，主机 2 内部的计数器被 SCL 的下降沿复位，同时输出级导通，之后两个主机都开始对 SCL 的低电平时间计数。假设主机 1 首先数完并关闭输出级，而主机 2 没有完成计数，输出级保持导通，此时 SCL 仍为低电平，这时主机 1 进入高电平等待状态，直到主机 2 完成计数并释放 SCL 线，SCL 变为高电平。

检测到 SCL 变高后，两个主机同时清 0，然后开始对 SCL 线高电平时间计数，在图 10-13 中，主机 1 首先完成 SCL 线高电平计数并拉低 SCL 线，检测到 SCL 线下降沿后，两个主机的计数器同时清 0，准备开始低电平时间的计数。

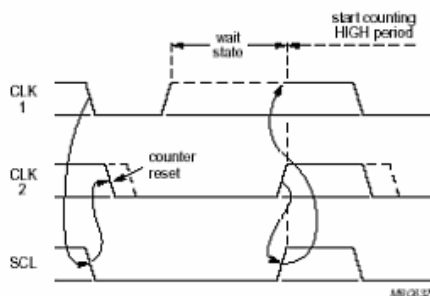


图10-13 I²C 总线的时钟同步

综上所述，SCL 线低电平时间取决于低电平时间最长的主机，高电平时间取决于高电平时间最短的主机。

主机只会在 I²C 总线空闲时产生起始位，但是在起始位的保持时间 $t_{HD,STA}$ （参见图 10-24）内可能有两个或以上的主机产生起始位，最终总线上的起始位由它们之间的线与运算决定，仲裁在随后 SDA 线上发送的比特中进行。仲裁的过程是，当 SCL 为高电平时，如果在 SDA 线上有主机发送低电平，那么发送高电平的主机将关闭输出级，因为总线的状态和自身内部不一样，于是发送低电平的主机赢得仲裁。例如在图 10-14 中，产生起始位后开始仲裁，在第一个比特，SCL 为高电平时，两个主机都输出高电平，仲裁继续，在第二个比特，两个主机都输出低电平，仲裁继续，到了第三个比特，主机 1 输出高电平，而 SDA 线为低电平（被主机 2 拉低），于是主机 1 断开内部输出级，主机 2 赢得仲裁。

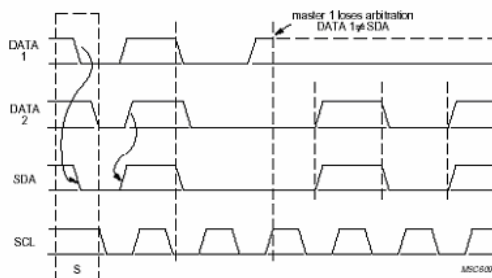


图10-14 两个主机的仲裁过程

仲裁可以持续多个比特，在实际通信过程中，仲裁的第一阶段是比较地址位，如果多个主机寻址同一个从机，那么继续比较数据比特（如果主机是发送器）或响应比特（如果主机是接收器）。由于 I²C 总线上的地址和数据由赢得总线的主机决定，因此仲裁过程中不会丢失信息。

如果一个主机具有从机功能，那么当它失去仲裁时，必须立即切换到从机状态，因为它可能正在被其他主机寻址。

10.2.4 I²C 协议的扩展

标准模式 I²C 总线规范规定了 100kbit/s 的数据传输速率和 7 比特寻址，这个规范在 20 世纪 80 年代初期已经存在，至今已经得到广泛接受，成为集成电路互连的一个事实标准，Philips Semiconductors 和其他供应商提供了大量的兼容 IC。为了满足更高的速度要求、更多的寻址空间，I²C 总线规范不断升级，现在提供的扩展包括：

- 快速模式，速率高达 400kbit/s；
- 高速模式（Hs-mode），速率高达 3.4Mbit/s；
- 10 位寻址，可以使用多达 1024 从机地址。

有关快速模式、高速模式、10 位寻址的技术细节可以查阅 I²C 总线规范。

10.3 I²C 总线应用实例——AT24C 系列 EEPROM

AT24C 系列芯片是采用 I²C 总线协议的串行 EEPROM 存储芯片，容量由 1Kbit 到 512Kbit，可重复擦/写 100 万次，数据保存期大于 100 年。该系列器件具有价格低、使用方

便的特点，在 IC 卡、消费电子等领域应用得非常广泛。这里以 AT24C02 为例说明 AT24C 系列串行 EEPROM 的结构和使用方法。

10.3.1 AT24C02 概述

AT24C02 的容量为 2Kbit，即 256Byte，常见封装有 8 管脚的 TSSOP 和 SOIC 等，如图 10-15 所示。

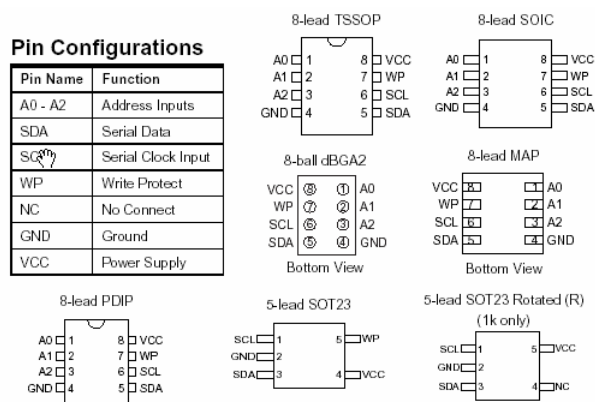


图10-15 AT24C02 封装及引脚图

其中 SDA、SCL 为 I²C 总线，WP 脚为写保护，高电平时器件内部所有的内容都被写保护。A0~A2 为 7bit 器件地址（即 I²C 总线的从机地址）的低 3bit，高 4bit 固定为“1010”，这样在 I²C 总线上可以同时连接 8 个 AT24C02，它们之间用不同的 A0~A2 值进行区别，I²C 总线 Master 可以方便地寻址其中的任何一个。

AT24C02 的内部结构如图 10-16 所示，其中 EEPROM 存储器分为 32 页，每页 8 字节。

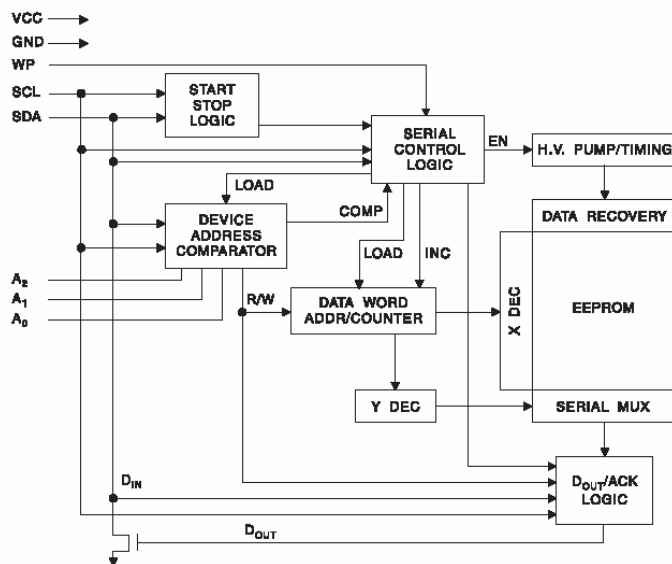


图10-16 AT24C02 内部结构图

10.3.2 写操作

写 AT24C 的操作有 Byte Write 和 Page Write 两种操作模式，可以一次写入一个字节或一页（8 个字节）。两种操作模式的时序如图 10-17 所示。

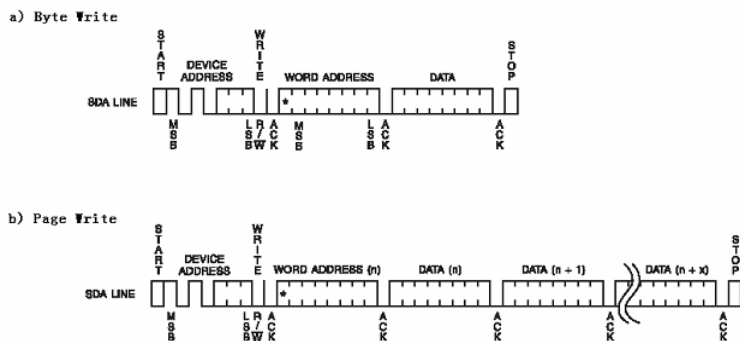


图10-17 AT24C02 的写操作模式

在 Byte Write 操作中，Master（例如微处理器）向 AT24C02 写数据时，首先寻址所需操作的 AT24C02，AT24C02 响应后，Master 发送字节地址（即图 10-17 中的 WORD ADDRESS），告诉 AT24C02 数据存储的地方，AT24C02 响应后，Master 发送数据，如果 AT24C02 正常接收，会发出响应比特，Master 收到后产生停止位，写操作结束，此时 AT24C02 开始内部数据的擦写，不再响应外部信号。

在 Page Write 操作中，AT24C02 可一次写入 8 个字节，Page Write 操作时也是先发送 Slave 地址，然后发送字节地址，不同的是传送了一字节数据后 Master 不产生停止位，而是继续发送 7 个字节。每收到一个字节，AT24C02 产生一个响应比特，并将地址计数器低位加 1，高位不变。如果在发送停止位之前，Master 发送超过 8 个字节，地址计数器将自动翻转，指向当前页的开头，先前写入的数据被随后写入的数据覆盖。发送完 8 字节数据后，Master 产生停止位，AT24C02 接收到停止位后，启动内部擦写操作，不再响应外部信号。

进行写操作时，Master 产生停止位后，AT24C02 进入内部擦写状态，不响应外部信号，利用这一点，Master 可以查询 AT24C02 内部擦写是否结束，方法是发送特定 AT24C02 的器件地址，如果响应比特为 0，说明 AT24C02 内部擦写已经结束，Master 可以进行下一步操作，否则应继续等待。

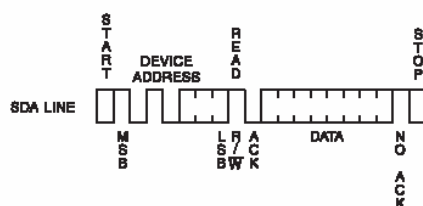
10.3.3 读操作

读 AT24C02 有 3 种操作模式，分别是 Current Address Read（当前地址读）、Random Read（随机读）和 Sequential Read（顺序读），时序如图 10-18 所示。读操作的初始化方式和写操作类似，也是先发送 Slave 地址，不同之处在于 R/W 位等于 1。

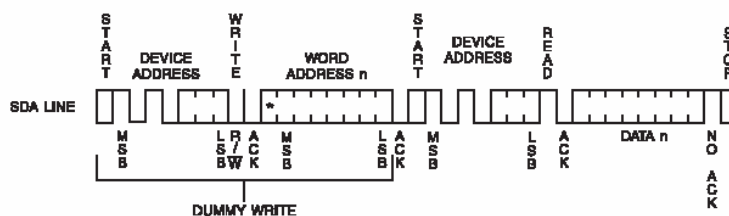
在 Current Address Read 操作中，读地址是上次的读写操作地址加 1，如果上次读、写操作地址为 N ，则 Current Address Read 的地址为 $N+1$ ，如果 $N=255$ ，那么从 0 地址开始读。AT24C02 输出一个字节的的数据后，Master 不产生响应位（即通知 AT24C02 结束发

送)，但是 Master 要产生停止位。

a) Current Address Read



b) Random Read



c) Sequential Read

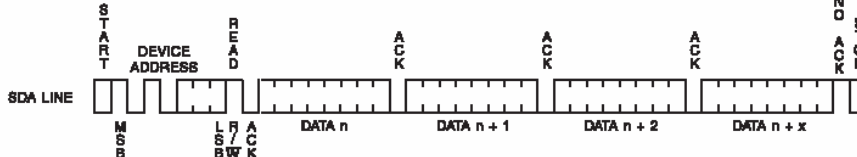


图10-18 AT24C02 的读操作模式

Random Read 操作允许 Master 读取 AT24C02 任意地址的内容，操作时 Master 首先发送起始位、Slave 地址 (R/W=0，即写操作) 和数据地址，这是一个伪写操作 (Dummy Write)，目的是把读数据地址通知 AT24C02。AT24C02 应答后，Master 重新发送起始位和 Slave 地址，此时 R/W 位置 1，AT24C02 响应并发送响应信号，然后输出所要求的一个 8 比特数据，最后 Master 不作应答，并产生停止位结束传输。

Sequential Read 可以由 Current Address Read 或 Random Read 操作启动，不同的是 Master 收到一个字节后，要发送响应比特 (ACK=0)，AT24C02 收到响应比特后将继续发送下一个地址的数据，直到 Master 不作响应，并产生停止位结束传输。读完最高地址的字节后，AT24C02 内部的地址计数器翻转，下一次读 0 地址中的内容 (在 Page Write 操作时，地址计数器翻转后指向本页的开头)。

10.3.4 AT24C 在 IC 卡中的应用简介

AT24C 系列 I²C 接口串行 EEPROM 在存储器 IC 卡中有广泛的应用。IC 卡的种类非常多，简单说有接触式 IC 卡和非接触式 IC 卡两类，接触式 IC 卡又可以分为存储器 IC 卡、逻辑加密 IC 卡和 CPU 卡 3 大类。其中存储式 IC 卡具有结构简单、成本低廉、使用方便的优点，在实际产品中应用得最多。

存储器 IC 卡的结构如图 10-19 所示。

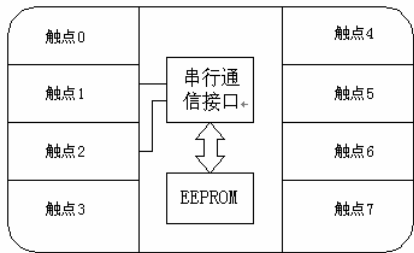


图10-19 存储器 IC 卡结构示意图

从图 10-19 可见，存储器 IC 卡的核心就是 EEPROM 和串行接口，其中又以 I²C 总线接口最常用。AT24C 系列 EEPROM 具有了存储器 IC 卡的所有部件。实际上 ATMEL 还提供了 IC 卡芯片形式封装的 AT24CXXSC 系列 I²C 接口 EEPROM，把它们安装在塑料片等载体上，就成为一张完整的存储器 IC 卡了。

如果掌握 I²C 总线接口技术，就可以设计出读卡器，从而制作出完整的存储器 IC 卡应用系统。后面的章节将详细讲述 I²C 总线接口控制器的设计，该控制器结合单片机可以设计出 IC 卡读卡器，或者把两者结合起来做成一块芯片，会有广泛的应用。

10.4 I²C 总线控制器设计详解

I²C 总线控制器的主要作用是提供μC（Microcontroller，微控制器或单片机）和 I²C 总线之间的接口，为两者之间的通信提供物理层协议的转换，典型的应用是在一个单片机系统（如 80C51 等）中，大多数器件，如 SRAM、LCD、FLASH 等可以直接挂在单片机外围总线上，但是对于串行 EEPROM、串行实时时钟等使用 I²C 协议的器件，就不能直接和单片机外围总线相连，这些器件可以挂在一套 I²C 总线上，再通过 I²C 总线控制器和μC 连起来，如图 10-20 所示。在 SOC 设计中类似的协议转换模块用得非常多。

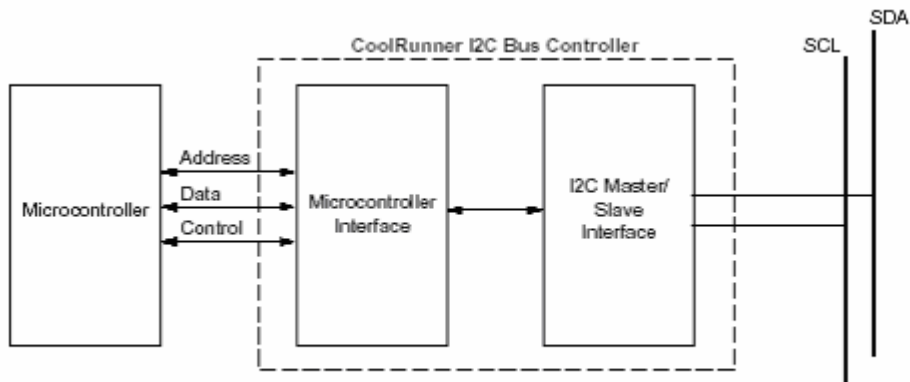


图10-20 I²C 总线控制器

图 10-20 所示的 I²C 总线控制器包含两个主要部分，一是微控制器接口，简称μC 接口，二是 I²C Master/Slave 接口，即 I²C 接口，通过这两个接口，I²C 总线控制器实现了微控

制器外围总线和 I²C 总线的连接。

本节将详细介绍 I²C 总线控制器的设计，主要参考 Xilinx 公司提供的资料，该资料在 I²C 接口设计、状态机设计、代码编写等方面都具有很高的参考价值。读者可以到 Xilinx 公司网站下载相关资料。



很多公司都有 I²C 接口串行 EEPROM 器件，型号一般也包含“24”字样，例如 Catalyst 公司的 CAT24WC 系列、ST 公司的 M24C 系列串行 EEPROM 等。由于商业上的原因，有些写做 2-Wire Bus 接口，而不写 I²C 接口，实际上是兼容的，可以互换。

10.4.1 I²C 总线控制器总体描述

I²C 总线控制器的内部框图如图 10-21 所示。

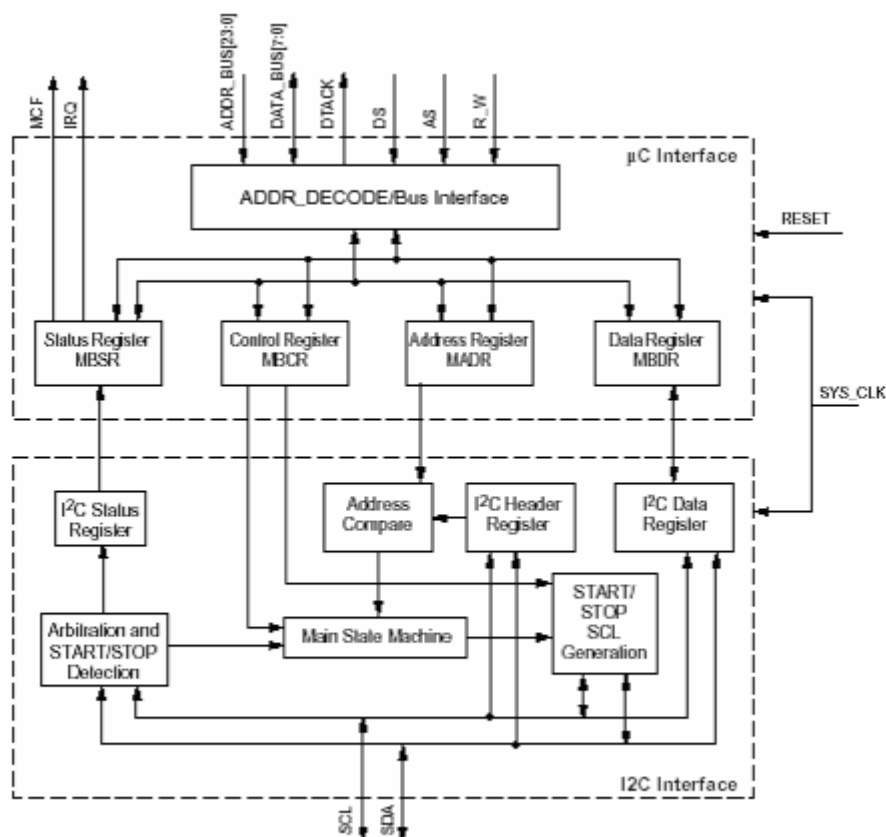


图10-21 I²C 总线控制器内部框图

图 10-21 所示的两个虚线框分别是 I²C 接口的两个组成部分，即 μ C 接口和 I²C 接口。 μ C 接口主要包含状态寄存器 MBSR、控制寄存器 MBSR、地址寄存器 MADR、数据寄存器 MBDR 和地址译码/总线接口模块。状态寄存器指示 I²C 总线控制器的当前状态，如传送是否完成、总线是否忙等信息，控制寄存器是 μ C 控制 I²C 总线控制器的主要途径，通过置 0/置 1 可以完成 I²C 总线控制器使能、中断使能、Master/Slave 模式选择、产生起始位等操作。

作。地址寄存器保存着 I²C 总线控制器作为 Slave 时的地址。数据寄存器用于保存接收或是待发送的数据。

在 I²C 接口的核心是主状态机，它控制着整个 I²C 接口的运作。和 I²C 总线直接相连的模块有起始/停止位产生、I²C Header 寄存器、I²C 数据寄存器和仲裁及起始/停止位检测模块。当控制器是 Master 时，起始/停止位产生模块用于在 I²C 总线上产生起始和停止位；I²C Header 寄存器用于保存每次通信的第一个字节，即从机地址和读写指示比特；I²C 数据寄存器用于保存总线上传送的数据；仲裁及起始/停止位检测模块的作用是执行仲裁，并检测 I²C 总线上的起始/停止位，以便为主状态机提供输入。其他模块包括：I²C 状态寄存器，用于记录 I²C 总线的状态；地址比较模块，用于比较总线上传送的地址和本机的 Slave 地址是否一致，如果一致，说明其他 Master 正在寻址本控制器，控制器必须立即切换到 Slave 状态，同时发出响应比特。

I²C 总线控制器的引脚描述如表 10-1 所示，其中 SDA、SCL 为 I²C 总线，用于连接 I²C 接口器件，CLK 是 I²C 总线控制器的工作时钟，其他为 I²C 控制器和μC 之间的连线。

表 10-1 I²C 总线控制器引脚描述

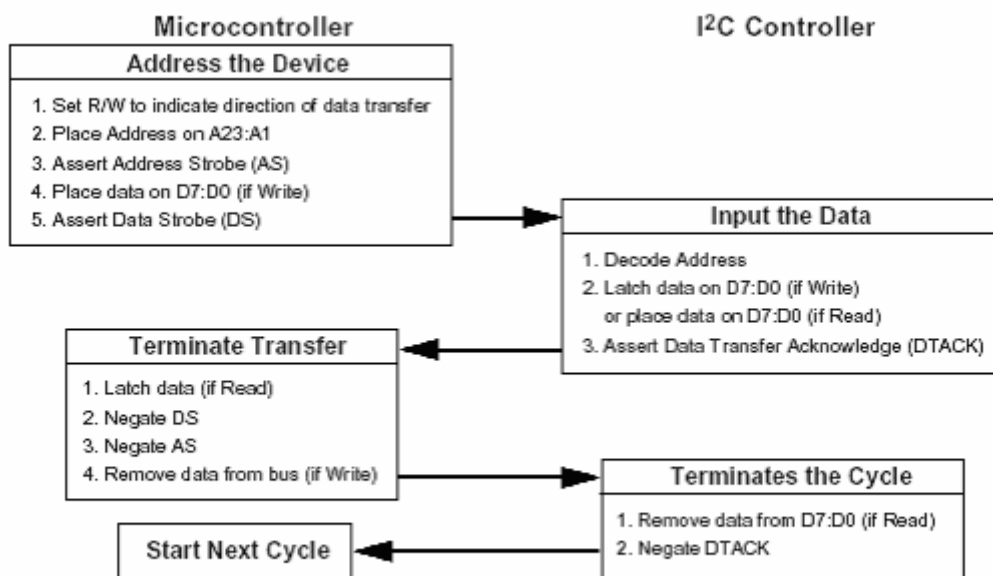
引脚名称	方向	描述
SDA	双向	I ² C 串行数据线
SCL	双向	I ² C 串行时钟线
ADDR_BUS[23:0]	输入	μC 地址总线
DATA_BUS[7:0]	双向	μC 数据总线
AS	输入	地址选通信号，低有效，表示总线上的地址有效
DS	输入	数据选通信号，低有效，表示总线上的数据有效，或者μC 释放数据总线
R_W	输入	读/写控制信号，1 为读，0 为写
DTACK	输出	数据传输响应信号，低有效，读操作时表示 I ² C 已经把数据放到μC 总线上，写操作时表示 I ² C 总线控制器已经锁存总线上的数据
IRQ	输出	中断请求，低有效
MCF	输出	数据传输指示，用于通知μC 字节传输是否结束。I ² C 总线正在传输某些字节时 MCF 清 0，在字节传输的第 9 个比特的下降沿置 1
CLK	输入	时钟输入

10.4.2 μC 接口设计

μC 接口用于连接 I²C 总线控制器和μC，设计时需要仔细考虑两者交互的流程（也就是握手机制），然后用 HDL 语言实现这个流程。在很多设计中都存在交互流程的设计问题，完善、简介的交互流程是系统正常工作的基础。

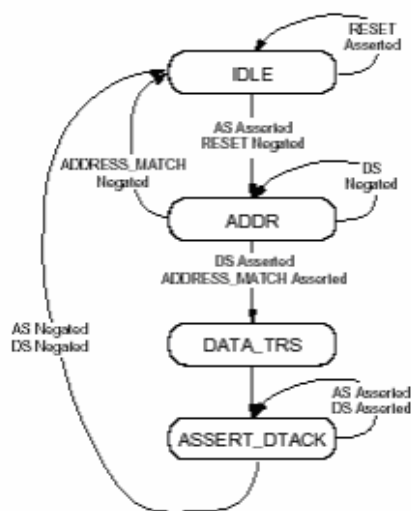
10.4.2.1 μC 和 I²C 总线控制器交互流程

I²C 总线控制器通过μC 接口和微控制器实现交互，两者交互的流程如图 10-22 所示。

图10-22 μ C 和 I²C 总线控制器的交互流程

10.4.2.2 μ C 接口状态机设计

μ C 接口设计使用 VHDL 语言和状态机描述，状态转移图如图 10-23 所示。状态机描述方法具有思路清晰、编码容易、代码可读性好、容易维护等优点，建议读者牢牢掌握，并尽量运用到自己的设计中。

图10-23 μ C 接口状态机的状态转移图

从图 10-23 可知， μ C 接口状态机复位后处于 IDLE 状态，AS (Address Strobe，地址选

通) 信号有效时, 状态机转到 ADDR 状态, 比较 μ C 地址总线上的地址和 I²C 总线控制器本身的地址是否相符, 以确定 I²C 总线控制器是否被 μ C 寻址。如果 I²C 总线控制器被寻址, 而且 DS (Data Strobe, 数据选通) 信号有效, 那么状态机转移到 DATA_TRS 状态。

在 DATA_TRS 状态, 如果当前的操作是 μ C 读数据, 那么请求的数据将放到 μ C 数据总线上, 如果当前的操作是 μ C 写数据, 那么 μ C 总线上的数据将锁存到 I²C 总线控制器内部指定的寄存器中。之后, I²C 总线控制器自动进入 ASSERT_DTACK 状态并把 DTACK 引脚置为有效, 表示 μ C 请求的数据已经放到数据总线上 (对读操作), 或 I²C 总线控制器已经锁存数据 (对写操作)。DTACK 置为有效之后, μ C 锁存数据 (对读操作) 或释放数据和地址总线 (对写操作), 同时把 AS 和 DS 信号置为无效, 使状态机回到 IDLE 状态。

本例是一个非常典型的 μ C 接口设计, 包含了完整的握手信号和握手机制, μ C 提供了 AS/DS/R_W 信号为 I²C 总线控制器指示地址/数据有效, 以及读/写操作, I²C 总线控制器提供了 DTACK 信号指示是否锁存了数据或者数据是否放到 μ C 总线上, 即指示 I²C 是否完成操作, 以便 μ C 进行下一步操作, 握手机制的设计是 FPGA/ASIC 设计中非常重要, 也是常常遇到的问题。

μ C 接口状态机用 VHDL 语言编写, 关键代码如下 (完整的代码请见本书附带光盘的“Example-10-1”目录):

```
-- Process: SEQUENTIAL
-- Function: Synchronize target state machine
SEQUENTIAL: process (clk, reset)
begin
    if reset = RESET_ACTIVE then
        prs_state <= IDLE;
        dtack_int <= '1';

    elsif clk'event and clk = '1' then
        prs_state <= next_state;
        dtack_int <= dtack_com;

    end if;

end process;

-- Process: COMBINATIONAL
-- Function: Contains the synchronous target state machine to mediate
--           the handshaking taking place with the uProc parallel bus
COMBINATIONAL: process (prs_state, as, as_int_d1, ds_int,
address_match)

begin
```

```

next_state <= prs_state;
dtack_com <= '1';
dtack_oe <= '0';

case prs_state is

    ----- IDLE State (00) -----
when IDLE =>

    -- Wait for falling edge of as
    if as_int_d1 = '1' and as = '0' then
        -- falling edge of AS
        next_state <= ADDR;
    end if;

    ----- ADDR State (01) -----
when ADDR =>
    -- Check that this module is being addressed
    if address_match = '1' then
        -- Wait for ds to be asserted, active low
        if ds_int = '0' then -- source of ds_int: ds_int <= ds;
            next_state <= DATA_TRS;
        else
            next_state <= ADDR;
        end if;
    else
        -- this module is not being addressed
        next_state <= IDLE;
    end if;

    ----- DATA_TRS State (10) -----
when DATA_TRS =>
    -- Read or write from enabled register
    next_state <= ASSERT_DTACK;
    dtack_oe <= '1';

    ----- ASSERT_DTACK State (11) -----
when ASSERT_DTACK =>

```

```
-- Assert dtack to uProcessor
dtack_com <= '0';
dtack_oe <= '1';

-- Wait for rising edge of as and ds
if (as_int_d1 = '0') and (ds_int = '0') then
    next_state <= ASSERT_DTACK;
elsif (as_int_d1 = '1') and (ds_int = '1') then
    next_state <= IDLE;
end if;

end case;

end process;
```

10.4.2.3 I²C 总线控制器内部寄存器

μ C 和 I²C 总线控制器之间的交互要用到 I²C 总线控制器内部的寄存器（包括状态寄存器 MBSR、控制寄存器 MBCR、地址寄存器 MADR 和数据寄存器 MBDR），这些寄存器的地址如表 10-2 所示。

表 10-2 I²C 总线控制器内部寄存器地址

地址	寄存器	描述
MBASE+\$141	MADR	I ² C 地址寄存器
MBASE+\$145	MBCR	I ² C 控制寄存器
MBASE+\$147	MBSR	I ² C 状态寄存器
MBASE+\$149	MBDR	I ² C 数据寄存器

寄存器的地址是 24 位的，其中高 16 比特为 I²C 总线控制器的基址，即表 10-2 中的 MBASE，可以在 VHDL 代码中指定，以便控制器映射到 μ C 特定的地址空间，低 8 位用于区别不同寄存器。下面简要介绍各寄存器。

• 地址寄存器（MADR）

地址寄存器保存了 I²C 总线控制器作为 Slave 被寻址时的地址，其格式如表 10-3 所示。编写 VHDL 代码时，这是一个常量。

表 10-3 地址寄存器描述

比特	名称	μ C 操作	描述
7~1	从机地址	读/写	I ² C 总线控制器的从机地址
0			未使用

• 控制寄存器（MBCR）

控制寄存器用于配置 I²C 总线控制器，各比特的含义如表 10-4 所示。

表 10-4

控制寄存器描述

比特	名称	μC 操作	描述
7	MEN	读/写	I ² C 总线控制器使能，MEN 置位后 MBCR 各比特才能生效 1：I ² C 总线控制器使能 0：复位和不使能 I ² C 总线控制器
6	MIEN	读/写	中断使能 1：中断使能，如果状态寄存器 MIF 为 1，则产生中断 0：不使能中断
5	MSTA	读/写	Master/Slave 模式选择，在 Master 模式下，μC 把该比特从 0 写为 1 时，I ² C 总线控制器产生起始位，清 0 时产生停止位，同时 I ² C 总线控制器切换到 Slave 模式
4	MTX	读/写	发送/接收模式选择，用于选择 I ² C 总线控制器的传输方向 1：发送 0：接收
3	TXAK	读/写	发送响应使能，用于指定 I ² C 控制器为接收机时发出的响应比特 1：响应比特为 1，即不响应 0：响应比特为 0，即响应 当 I ² C 控制器为主机接收器时，不响应表示接收结束
2	RSTA	读/写	重复起始产生位，如果 I ² C 控制器是主机，写入 1 会在 I ² C 总线上产生一个重复起始位，如果 I ² C 控制器不是主机，写入 1 会导致仲裁丢失。RSTA 读出总是 0
1~0	保留		

• 状态寄存器 (MBSR)

状态寄存器保存着 I²C 总线控制器的状态信息，除了 MIF/MAL 比特以外的其他比特是只读的。各比特的含义如表 10-5 所示。

表 10-5

状态寄存器描述

比特	名称	μC 操作	描述
7	MCF	读	数据传输指示，I ² C 总线正在传输某些字节时 MCF 清 0，在字节传输的第 9 个比特的下降沿置 1。在本 I ² C 控制器中，MCF 也是一个输出引脚，这样不用通过读寄存器就可以知道传输是否完成
6	MAAS	读	Addressed as Slave Bit，当 I ² C 总线上的地址和 MADR 中保存的地址一致时，I ² C 总线控制器被寻址，切换到 Slave mode，MAAS 置位
5	MBB	读	Bus Busy Bit，总线忙比特，用于指示 I ² C 总线的状态。检测到起始位时，MBB 置位，检测到停止位时，MBB 复位
4	MAL	读/写	Arbitration Lost Bit，仲裁丢失比特，I ² C 控制器丢失仲裁时 MAL 置位，通过 μC 写入 0 复位 MAL
3	保留		
2	SRW	读	当 I ² C 控制器被寻址时，SRW 指示了主机发送的 read/write 比特。只有在一个传输完成、新传输尚未开始时 SRW 有效 1：主机读从机 0：主机写从机
1	MIF	读/写	中断比特，有中断申请时，MIF 置位，如果 MIEN 为 1，那么 I ² C 控制器会向 μC 发出中断请求。应在中断服务程序中写入 0 以清除 MIF 比特
0	RXAK	读	Received Acknowledge Bit，接收到的响应比特。RXAK 指示了响应周期中 SDA 信号线的值

• 数据寄存器 (MBDR)

数据寄存器用于保存 I²C 总线控制器发送或接收的数据。在发送模式，μC 写入数据寄存器的数据将发送到 I²C 总线，在接收模式，I²C 总线控制器从 I²C 总线接收到的数据保存到数据寄存器中。

数据寄存器的格式如表 10-6 所示。

表 10-6 数据寄存器描述			
比特	名称	μC 操作	描述
7~0	D7~D0	读/写	I ² C 数据



编写 VHDL 代码时，MBDR 由两个 8 比特的寄存器组成，这两个寄存器分别用于读写操作，在μC 寻址空间中共享相同的地址。由于μC 向 I²C 总线控制器写数据和 I²C 总线控制器从 I²C 总线接收数据两个操作都要写数据寄存器，这样的安排避免了两两者之间可能的冲突。类似的设计在单片机、微处理器设计中很常见。

10.4.3 I²C 接口设计

I²C 接口用于连接 I²C 总线控制器和 I²C 总线，由两个状态机构成，一个是 I²C 接口主状态机，用于执行发送和接收操作，另一个名为 SCL/SDA/STOP Generation 状态机，当 I²C 总线控制器为 Master 时，这个状态机产生 SCL/SDA/STOP 信号。为了方便理解，这里先分析 SCL/SDA/STOP Generation 状态机的设计。

10.4.3.1 SCL/SDA/STOP Generation 状态机

I²C 接口用于 I²C 总线的驱动或接收，当 I²C 总线控制器为 Master 时，I²C 接口必须按 I²C 总线规范驱动 I²C 总线，I²C 总线控制器为 Slave 时，I²C 接口必须能正确接收满足 I²C 总线规范的信号。

I²C 设计规范对 I²C 总线的时序做了详细的定义，如图 10-24 和表 10-7 所示。

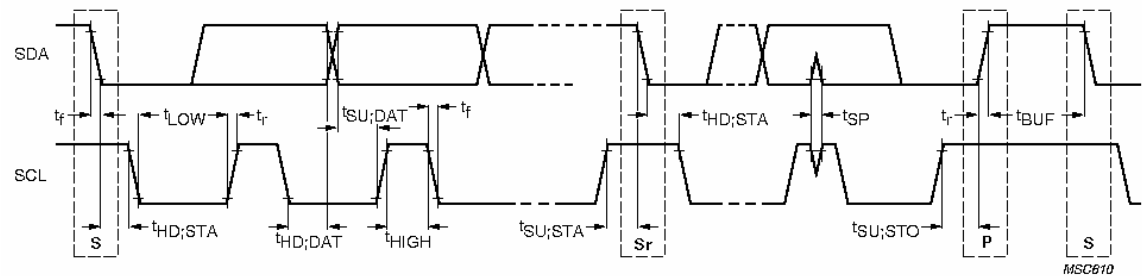


图10-24 I²C 总线的时序定义（标准模式）

表 10-7

标准模式 I²C 总线器件的时序特性

参数	符号	标准模式		快速模式		单位
		最小值	最大值	最小值	最大值	
SCL 时钟频率	fSCL	0	100	0	400	kHz
(重复) 起始条件的保持时间, 在这个周期后, 产生第一个时钟脉冲	tHD;STA	4.0	-	0.6	-	us
SCL 时钟的低电平周期	tLOW	4.7	-	1.3	-	us
SCL 时钟的高电平周期	tHIGH	4.0	-	0.6	-	us
重复起始条件的建立时间	tSU;STA	4.7	-	0.6	-	us
数据保持时间:						
兼容 CBUS 的主机	tHD;DAT	5.0	-	-	-	us
I ² C 总线器件		0	3.45	0	0.9	us
数据建立时间	tSU;STA	250	-	100	-	ns
SDA 和 SCL 信号的上升时间	tr	-	1000	20+0.1 Cb	300	ns
SDA 和 SCL 信号的下降时间	tf	-	300	20+0.1 Cb	300	ns
停止条件的建立时间	tSU;STO	4.0	-	0.6	-	us
停止和启动条件之间的总线空闲时间	tBUF	4.7	-	1.3	-	us
每条总线线路的电容负载	Cb	-	400	-	400	pF
每个连接的器件低电平时的噪声容限 (包括迟滞)	VnL	0.1 VDD	-	0.1 VDD	-	V
每个连接的器件高电平时的噪声容限 (包括迟滞)	VnH	0.2 VDD	-	0.2 VDD	-	V

设计 I²C 接口时, 使用一个频率比较高的工作时钟, 通过对这个时钟进行计数获得所需要的时序。例如 I²C 接口工作时钟 CLK 为 2MHz, 为了产生 4.0ns 的起始条件保持时间, 只需要数 8 个 CLK; 为了产生 4.7ns 的停止和启动条件之间的总线空闲时间, 只需要数 9 个 CLK, 同理可以满足 SCL 高低电平周期、重复起始条件的建立时间等时序要求, 相应的 CLK 计数常数定义在 VHDL 代码中, 如下所示, 可以根据目标系统的实际时钟调整这些常数。

```

constant    CNT_100KHZ: std_logic_vector(4 downto 0) := "10100";
-- number of 2MHz clocks in 100KHZ

constant    HIGH_CNT   : std_logic_vector(3 downto 0) := "1000";
-- number of 2MHz clocks in half
-- 100KHZ period -1 since count from 0
-- and -1 for "edge" state

constant    LOW_CNT    : std_logic_vector(3 downto 0) := "1000";
-- number of 2MHz clocks in half
-- 100KHZ period-1 since count from 0
-- and -1 for "edge" state

```

```

constant    HIGH_CNT_2 :    std_logic_vector(3 downto 0) := "0100";
                                -- half of HIGH_CNT
constant    TBUF       :    std_logic_vector(3 downto 0) := "1001";
                                -- number of 2MHz clocks in 4.7uS
constant    DATA_HOLD :    std_logic_vector(3 downto 0) := "0001";
                                -- number of 2MHz clocks in 300ns
constant    START_HOLD: std_logic_vector(3 downto 0) := "1000";
                                -- number of 2MHz clocks in 4.0uS

```

SCL/SDA/STOP Generation 状态机的状态转移图如图 10-25 所示。

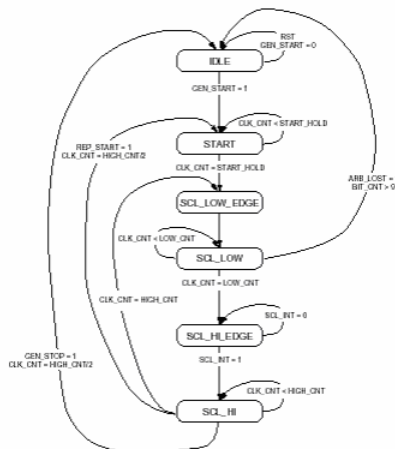


图10-25 SCL/SDA/STOP Generation 状态机的状态转移图

I²C 总线控制器复位后处于 IDLE 状态，不驱动 SCL 和 SDA，此时 I²C 总线上的其他 Master 可以控制 SCL 和 SDA。如果 I²C 总线控制器处于 Master 模式，而且 I²C 总线处于空闲状态， μ C 通过置位 MBCR 寄存器的 MSTA 比特使 GEN_START 信号为高，那么状态机进入 START 状态。

在 START 状态，SCL 保持为高电平，同时驱动 SDA 信号变低，从而在 I²C 总线上产生一个起始位。系统时钟计数器启动计数，直到满足 I²C 规范要求的起始条件保持时间 (>4ns)，状态机进入 SCL_LOW_EDGE 状态。

在 SCL_LOW_EDGE 状态，状态机使 SCL 产生一个下降沿并复位系统时钟计数器，然后在下一个时钟沿到来时进入 SCL_LOW 状态。在 SCL_LOW 状态，SCL 保持为低，同时进行计数，直到产生规定的 SCL 低电平时间 (>4.7ns)。

产生规定的 SCL 低电平时间后，如果失去仲裁，那么完成一个字节的传输之后状态机回到 IDLE 状态，否则状态机进入 SCL_HI_EDGE 状态。

在 SCL_HI_EDGE 状态中，状态机释放 SCL 线，希望产生 SCL 上升沿，但是 SCL 线可能被其他 Master 置低，因此状态机并不直接转移到 SCL_HI 状态，而是等待 SCL 信号变高之后才进入 SCL_HI 状态。

进入 SCL_HI 状态后，系统时钟计数器进行计数，以产生 I²C 规范要求的 SCL 高电平时间 (>4.0ns)，如果检测到重复起始条件或停止条件，状态机将在 1/2 SCL 高电平时间之

后转移到 START 状态重新开始，或转移到 IDLE 状态，否则产生要求的 SCL 高电平时间后状态机进入 SCL_LOW_EDGE 状态，继续产生下一个 SCL 脉冲。

实现 SCL/SDA/STOP Generation 状态机的关键代码如下（完整的代码请见本书附带光盘的“Example-10-1”目录）：

```
-- *****SCL_Generator Process *****
-- This process generates SCL and SDA when in Master mode. It
-- generates the START and STOP conditions. If arbitration is lost,
-- SCL will be generated until the end of the byte transfer.

scl_generator_comb: process (scl_state, arb_lost, sm_stop, gen_stop,
                             rep_start, bus_busy, gen_start, master_slave,
                             stop_scl_reg, clk_cnt, bit_cnt, scl_in, state, sda_out,
                             sda_out_reg, master_sda)

begin

-- state machine defaults
scl_out <= '1';
sda_out <= sda_out_reg;
stop_scl <= stop_scl_reg;
clk_cnt_en <= '0';
clk_cnt_rst <= '1';
next_scl_state <= scl_state;
rsta_rst <= not(RESET_ACTIVE);

case scl_state is

    when SCL_IDLE =>
        sda_out <= '1';
        stop_scl <= '0';
        -- leave IDLE state when master, bus is idle, and gen_start
        if master_slave = '1' and bus_busy = '0' and gen_start =
'1' then
            next_scl_state <= START;
        end if;

    when START =>
        -- generate start condition
        clk_cnt_en <= '1';
```

```

        clk_cnt_rst <= '0';
        sda_out <= '0';
        stop_scl <= '0';
    -- generate reset for repeat start bit if repeat start condition
    if rep_start = '1' then
        rsta_rst <= RESET_ACTIVE;
    end if;
    if clk_cnt = START_HOLD then
        next_scl_state <= SCL_LOW_EDGE;
    else
        next_scl_state <= START;
    end if;

    when SCL_LOW_EDGE =>
        clk_cnt_rst <= '1';
        scl_out <= '0';
        next_scl_state <= SCL_LOW;
        stop_scl <= '0';
    when SCL_LOW =>
        clk_cnt_en <= '1';
        clk_cnt_rst <= '0';
        scl_out <= '0';

    -- set SDA_OUT based on control signals
    if arb_lost = '1' then
        stop_scl <= '0';
    elsif ((sm_stop = '1' or gen_stop = '1') and
        (state /= ACK_DATA and state /= ACK_HEADER and state
        /= WAIT_ACK)) then
        sda_out <= '0';
        stop_scl <= '1';
    elsif rep_start = '1' then
        sda_out <= '1';
        stop_scl <= '0';
    elsif clk_cnt = DATA_HOLD then
        sda_out <= master_sda;
        stop_scl <= '0';
    else
        stop_scl <= '0';

```

```

end if;

-- determine next state
if clk_cnt = LOW_CNT then

    if bit_cnt = CNT_DONE and arb_lost = '1' then
        next_scl_state <= SCL_IDLE;
    else
        next_scl_state <= SCL_HIGH_EDGE;
    end if;
else
    next_scl_state <= SCL_LOW;
end if;

when SCL_HIGH_EDGE =>
    clk_cnt_rst <= '1';
    scl_out <= '1';
    if ((sm_stop = '1' or gen_stop = '1') and
        (state /= ACK_DATA and state /= ACK_HEADER and state /=
WAIT_ACK)) then
        stop_scl <= '1';
    else
        stop_scl <= '0';
    end if;

    if scl_in = '0' then --wait until SCL switching 1
        next_scl_state <= SCL_HIGH_EDGE;
    else
        next_scl_state <= SCL_HIGH;
    end if;

when SCL_HIGH =>
    -- now all SCL drivers have set SCL to '1'
    -- begin count for high time
    clk_cnt_en <= '1';
    clk_cnt_rst <= '0';
    scl_out <= '1';
    -- check to see if a repeated start or a stop needs to be
    -- generated. If so, only hold SCL high for half of the high time

```

```

        if clk_cnt = HIGH_CNT_2 then

            if rep_start = '1' then
                next_scl_state <= START;
                clk_cnt_rst <= '1';
            elsif stop_scl_reg = '1' then
                next_scl_state <= STOP_WAIT;
                clk_cnt_rst <= '1';
            end if;
        elsif clk_cnt = HIGH_CNT then
            next_scl_state <= SCL_LOW_EDGE;
        else
            next_scl_state <= SCL_HIGH;
        end if;

        when STOP_WAIT =>
            --this state gives the required free time between stop and start -
            -conditions
            clk_cnt_en <='1';
            clk_cnt_rst <= '0';
            sda_out <= '1';
            stop_scl <= '0';
            if clk_cnt = TBUF then
                next_scl_state <= SCL_IDLE;
            else
                next_scl_state <= STOP_WAIT;
            end if;

        end case;
    end process;

    scl_generator_regs: process (sys_clk, reset)
    begin
        if reset = RESET_ACTIVE then
            scl_state <= SCL_IDLE;
            sda_out_reg <= '1';
            scl_out_reg <= '1';
            stop_scl_reg <= '0';
        elsif sys_clk'event and sys_clk='1' then

```

```

scl_state <= next_scl_state;
sda_out_reg <= sda_out;
scl_out_reg <= scl_out;
stop_scl_reg <= stop_scl;
end if;
end process;

```

10.4.3.2 I²C 接口主状态机

I²C 接口主状态机的状态转移图如图 10-26 所示，不管 I²C 总线控制器处于 Master 还是 Slave 模式，状态转移图是相同的，不同的是状态机根据当前所处的模式决定状态机的输出和转移到下一个状态的条件。这样的设计允许 I²C 总线控制器在失去仲裁和被作为 Slave 寻址时立即从 Master 模式切换到 Slave 模式。

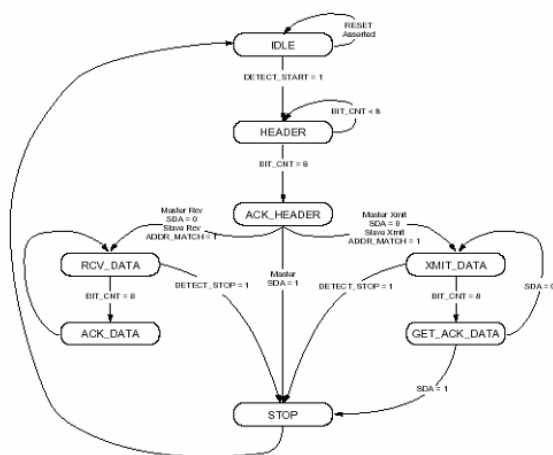


图10-26 I²C 接口主状态机的状态转移图

复位后，状态机在 IDLE 状态，当检测到 START 信号时，转移到 HEADER 状态。START 信号由 I²C 总线上的起始位触发，触发这个起始位的 Master 可以是 I²C 总线控制器本身或其他的 I²C 总线主机。

在 HEADER 状态，如果 I²C 总线控制器处于 Master 模式，它会把 MBDR 中的数据作为 HEADER 发送到 I²C 总线上，以寻址特定的 Slave。不管 I²C 总线控制器处于 Master 还是 Slave 模式，在 HEADER 状态时，I²C 总线控制器都会接收总线上的数据，保存到 I²C Header Shift Register 中，收到 8 个比特后，状态机转移至 ACK_HEADER 状态。

在 ACK_HEADER 状态，如果 I²C 总线控制器处于 Master 模式，它会采样 SDA 线，以判断所寻址的 Slave 是否响应。如果没有响应，状态机转移到 STOP 状态，通知 SCL/START/STOP Generator 产生 STOP 信号，中止传输。如果 Slave 产生了响应比特，状态机根据 Header 的最低位判断发起的是发送操作还是接收操作，然后转移到 RCV_DATA 状态或 XMIT_DATA 状态。如果 I²C 控制器处于 Slave 模式，电路会不断比较 Header Shift Register 的内容和 I²C 总线控制器地址寄存器 MADR 的内容是否相等，如果相等，说明本 I²C 总线控制器被其他 Master 寻址，于是 I²C 总线控制器立即转换到 Slave 模式，并把状态

寄存器 MBSR 的 MAAS 比特置 1，指示 I²C 总线控制器被其他 Master 寻址。同时 MBSR 的 SRW 比特记录 Header 的最低位，以便 μ C 判断 Master 请求的是读还是写操作。

在 RCV_DATA 状态，I²C 总线控制器处于接收状态（即主机接收状态或从机接收状态），状态机读入 I²C 总线上的数据并保存到移位寄存器中，读完 8 比特的数据后进入 ACK_DATA 状态，发出响应比特。响应比特的取值根据 I²C 总线控制器是 Master 还是 Slave 有所不同，当 I²C 总线控制器是 Slave 时，响应比特应为 0，表示正常接收；当 I²C 总线控制器是 Master 时，如果已经收到了足够的的数据，响应比特要设置为 1，通知 Slave 停止发送，否则响应比特应为 0，通知 Slave 继续发送。响应比特的值由 MBSR 的 TXAK 位决定， μ C 可以在适当的时候写入。检测到 I²C 总线上的停止位时，状态机转移到 STOP 状态。

在 XMIT_DATA 状态，I²C 总线控制器处于发送状态（即主机发送，或从机发送状态），状态机把数据寄存器 MBDR 的数据移位输出到 SDA 线上，发送 8 比特后进入 GET_ACK_DATA 状态，收到响应比特后，状态机回到 XMIT_DATA 状态，继续发送下一个字节。如果没有收到响应比特，说明发送结束或出错，状态机转到 STOP 状态。

在 STOP 状态，如果处于 Master 模式，主状态机通知 SCL/START/STOP Generator 产生停止位。下一个时钟沿到来时，状态机自动转移到 IDLE 状态。

I²C 接口主状态机的关键代码如下（完整的代码请见本书附带光盘的“Example-10-1”目录）：

```
-- ***** Main State Machine Process *****
-- The following process contains the main I2C state machine for both
-- master and slave modes. This state machine is clocked on the falling
-- edge of SCL. DETECT_STOP must stay as an asynchronous reset because
-- once STOP has been generated, SCL clock stops.
state_machine: process (scl, reset, detect_stop)

begin

if reset = RESET_ACTIVE or detect_stop = '1' then
    state <= IDLE;
    sm_stop <= '0';
elsif scl'event and scl = '0' then

    case state is

        ----- IDLE STATE -----
        when IDLE =>
            if detect_start = '1' then
                state <= HEADER;
            end if;
```

```
----- HEADER STATE -----
when HEADER =>
    if bit_cnt = CNT_DONE then
        state <= ACK_HEADER;
    end if;
----- ACK_HEADER STATE -----
when ACK_HEADER =>
    if arb_lost = '1' then
        state <= IDLE;
    elsif sda_in = '0' then
        -- ack has been received, check for master/slave
        if master_slave = '1' then
            -- master, so check mtx bit for direction
            if mtx = '0' then
                -- receive mode
                state <= RCV_DATA;
            else
                --transmit mode
                state <= XMIT_DATA;
            end if;
        else
            if addr_match = '1' then
                --if maas = '1' then
                -- addressed slave, so check
                -- I2C_HEADER(0) for direction
                if i2c_header(0) = '0' then
                    -- receive mode
                    state <= RCV_DATA;
                else
                    -- transmit mode
                    state <= XMIT_DATA;
                end if;
            else
                -- not addressed, go back to IDLE
                state <= IDLE;
            end if;
        end if;
    else
        -- no ack received, stop
    end if;
end if;
```

```

        state <= IDLE;
        if master_slave = '1' then
            sm_stop <= '1';
        end if;

    end if;

    ----- RCV_DATA State -----
    when RCV_DATA =>

        -- check for repeated start
        if (detect_start = '1') then
            state <= HEADER;
        elsif bit_cnt = CNT_DONE then

            -- Send an acknowledge
            state <= ACK_DATA;

        end if;

    ----- XMIT_DATA State -----
    when XMIT_DATA =>
        -- check for repeated start
        if (detect_start = '1') then
            state <= HEADER;

        elsif bit_cnt = CNT_DONE then

            -- Wait for acknowledge
            state <= WAIT_ACK;

        end if;

    ----- ACK_DATA State -----
    when ACK_DATA =>

        state <= RCV_DATA;

    ----- WAIT_ACK State -----
    when WAIT_ACK =>
        if arb_lost = '1' then

```

```
        state <= IDLE;
    elsif (sda = '0') then
        state <= XMIT_DATA;
    else
        -- no ack received, generate a stop and return
        -- to IDLE state
        if master_slave = '1' then
            sm_stop <= '1';
        end if;
        state <= IDLE;
    end if;

end case;

end if;

end process;
```

需要注意的是，I²C 接口主状态机和相关计数器、移位寄存器的工作时钟是 SCL，因此必须保证 SCL 信号的质量，如果 I²C 总线上的器件比较多，SCL 的上升和下降时间可能过长，导致系统的抗干扰能力下降，此时需要在 SCL 线上加驱动器。

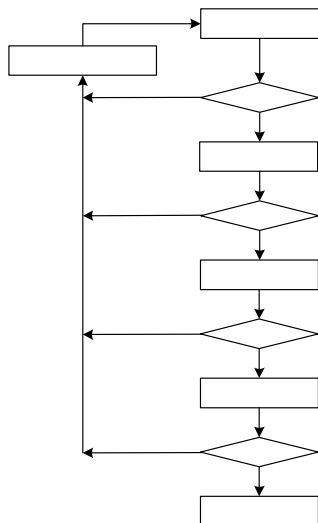
10.4.4 混合仿真验证

写完 VHDL 代码后，必须做细致、全面的仿真，只有仿真通过后才能做综合、实现，否则上板调试出问题几乎是必然的。由于上板调试缺乏足够的故障定位和调试手段，设计运行不正确时，靠逻辑分析仪等工具来定位故障非常困难，大多数情况下还是要回到仿真平台定位和排除故障。仿真可以方便地模拟各种输入，覆盖各种情况，对设计做全面的验证，通常做 FPGA 测试不容易做到这一点，例如 I²C 总线上存在多个 Master 时的仲裁，在 FPGA 板上就很难验证，而仿真并不困难。总之，综合、实现之前做全面的仿真验证可以高效地检查设计是否正确，是必不可少的。

10.4.4.1 验证流程

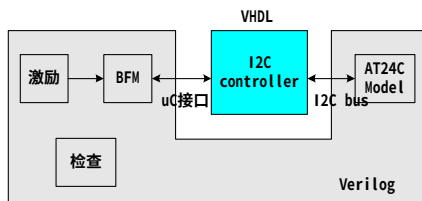
I²C 总线控制器的验证流程如图 10-27 所示。

在图 10-27 中，功能仿真就是常说的前仿真，用于检查 RTL 代码的正确性，综合后仿真用于检查网表是否正确，疏忽大意或代码风格不严谨很容易导致综合出来的网表的功能和预想不一致，综合后仿真能发现这样的问题。PAR 后仿真的意思是布局布线（Place and Route）后仿真，也就是常说的后仿真或时序仿真，它可以检验网表和时序的正确性。I²C 总线控制器验证的最后一步是 FPGA 验证，也就是把 BIT 文件下载到 FPGA 中运行，观察结果是否正确，可以非常直观地判断设计是否正确。

图10-27 I²C 总线控制器验证流程

10.4.4.2 建立 Testbench

仿真验证时需要建立 Testbench（测试环境），I²C 总线控制器验证环境的结构如图 10-28 所示。

图10-28 I²C 总线控制器的 Testbench 结构

其中，激励模块产生仿真所需的数据，BFM 模块为 Bus Function Model（总线功能模型）的简称，其作用是模拟一个微机接口，实现高层次数据（激励等）和底层信号（ μ C 接口）之间的转换，AT24C02 Model 是 ATMEL 公司 AT24C02 串行 EEPROM 的仿真模型，这个模型和 I²C 总线控制器通过 I²C 总线相连。I²C 总线上还可以挂很多其他的仿真模型，以便进行更全面的仿真验证。检查模块用于检查测试的结果是否正确，并给出相应的信息。整个 Testbench 根据 Transaction Based Verification（基于事务级验证）的思想构建，思路清晰、层次分明。

AT24C02 仿真模型来自网上，不必自己编写。AT24C02 使用行为级描述，与 RTL 级描述不同，行为级是不能综合的。由于行为级描述具有较高的抽象层次，可以方便地描述对象的行为，因此在搭建 Testbench 时用得非常多，读者可以仔细分析 AT24C02 的仿真模型代码，学习行为级描述的思路、方法。

注意到 I²C 总线控制器使用 VHDL 语言，激励、BFM、AT24C02 仿真模型模块使用 Verilog 语言，因此这是一个混合语言仿真环境。随着 EDA 技术的发展，FPGA/ASIC 设计越来越多地采用 IP（知识产权模块）复用的设计方法，特别在 SOC（片上系统）设计中，

独立完成所有模块的设计几乎是不可能的，至少是不符合成本效益的。基于 IP 复用的设计方法的特点是购买不同厂商的 IP 模块，集成到自己的系统中，和自己开发的模块连接起来，然后进行仿真验证。这些 IP 可能是 Verilog 语言编写的，也可能是 VHDL 语言编写的，因此验证环境往往是混合语言仿真环境，这种情况越来越普遍。通过 I²C 总线控制器验证，读者可以尽快熟悉和掌握混合语言仿真技术。

验证流程是在 μ C 的控制下，I²C 总线控制器通过 I²C 总线向 EEPROM 写入 0~255，然后读出检查，从而判断 I²C 总线控制器的工作是否正常。

编写 Testbench 时，需要根据 I²C 总线控制器的读写流程进行，如图 10-29 所示。

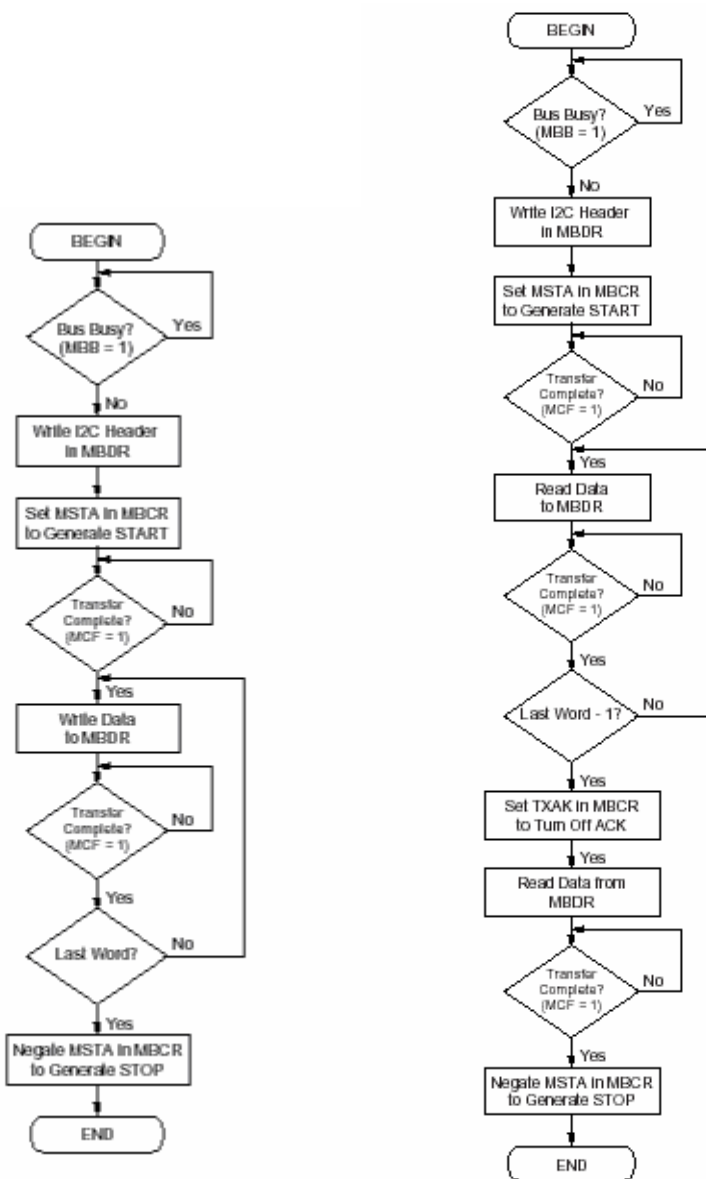


图10-29 I²C 总线控制器读写流程

这是 I²C 总线控制器为 Master 时的发送和接收流程，两者的主要差别是，接收时， μ C 需要在接收将近结束时配置 I²C 总线控制器 MBCR 寄存器，以便 I²C 总线控制器在最后一个字节不发出响应比特，让 Slave 结束发送。Testbench 向 EEPROM 写入测试数据的代码如下：

```
//***** write data to EEPROM *****
task current_write_eeprom;
begin
    'mbcr=8'b0;
    men=1'b1;
    mbb=1'b1;
    txak=1'b0; //ack
    uc_write_i2c(I2C_MBCR_ADDR,'mbcr); //enable I2C
    while(mbb==1) // =1, bus busy
        begin
            uc_read_i2c(read_out,I2C_MBSR_ADDR);
            'mbsr=read_out;
        end

    uc_write_i2c(I2C_MBDR_ADDR,{EEPROM_Address,1'b0}); //write i2c header
    in MBDR; 1'b0 means master write data

    mtx=1; //necessary
    msta=1; //Set MSTA in MBCR to Generate START
    uc_write_i2c(I2C_MBCR_ADDR,'mbcr);

    @(posedge mcf_wire);

    for(count=255;count>=1;count=count-1)
        begin
            @(posedge mcf_wire)
            uc_write_i2c(I2C_MBDR_ADDR,count); //write data to MBDR
            $display("write data: %d",count);
        end

    @(posedge mcf_wire);
    msta=1'b0;//reset MSTA in MBCR to Generate stop
    uc_write_i2c(I2C_MBCR_ADDR,'mbcr);
end
endtask
```


由于调用了 BFM 里的 task，因此代码结构清晰，阅读、维护都非常容易，整个 Testbench 代码可以参见请见本书附带光盘的“Example-10-1”目录。

需要注意的是，受到篇幅的限制，本 Testbench 的编写并不完善，覆盖率、结果自动检查等很多问题也未能涉及，笔者将会在以后出版的书籍中详细论述验证相关的问题，有兴趣的读者请关注人民邮电出版社的出版动态。



在复杂 FPGA/IC 项目中，HDL 代码设计只占工作量的 30%，60%~70%的时间会用在仿真验证工作上。在基于 IP 复用的设计中，由于用了比较多的 IP，设计的工作量大大减少，验证工作量却依然巨大。写好验证计划和 Testbench 是非常有挑战性的工作，也是项目成败的关键，必须给予足够的重视。

10.4.4.3 BFM 模块编写

这里向读者介绍 FPGA/IC 验证技术的一个重要概念——BFM 模块，BFM 模块可以理解为 Testbench 的物理层，可以把验证所需的操作，例如 μ C 对 I²C 总线控制器的读、写操作转换为 μ C 和 I²C 总线控制器之间的信号变化，从而驱动 I²C 总线控制器执行相应的操作。使用 BFM 构建 Testbench 的优点如下：

- (1) Testbench 结构清晰，使用 BFM 后，Testbench 可以简单划分为激励生成和驱动两大部分，前者产生验证所需的数据和控制整个验证的流程，是抽象层次比较高的部分；后者也就是 BFM，和 DUT 直接相连，负责 DUT 的驱动和控制，是抽象层次比较低的部分，采用周期精确（cycle-accurate）的编写风格。
- (2) 可维护性好，采用 BFM 后，如果 DUT 接口信号或时序发生变化，只需要修改 BFM，不用改动 Testbench 的其他部分。如果想改变送到 DUT 的数据或流程，则只需要修改激励生成部分。

I²C 接口验证环境中，BFM 主要包含 μ C 读写 I²C 总线控制器的 task，Verilog 代码如下：

```
task uc_write_i2c;
input [23:0] addr;
input [7:0] data;
begin
  @(negedge clk)
  r_w=1'b0;
  addr_bus=addr;
  as=1'b0;
  data_bus=data;
  ds=1'b0;
  data_bus_drive=1;
  @(negedge dtack)
  as=1'b1;
  ds=1'b1;
  data_bus=8'hzz;
end
```

```

    r_w=1'b1;          //reset the r_w line
    data_bus_drive=0;
    #1000;             //2clk, wait for uc_interface to transit to idle state
end
endtask

task uc_read_i2c;
output [7:0] data;
input [23:0] addr;
begin
    @(negedge clk)      //write i2c header in MBDR
    r_w=1'b1;
    addr_bus=addr;
    as=1'b0;
    ds=1'b0;            //wait for state transition from addr to data_trs
    data_bus_drive=1'b0;
    @(negedge dtack)
    as=1'b1;
    ds=1'b1;
    data=data_bus_tri;
    #1000 ;            //2clk, wait for uc_interface to transite to idle state
end
endtask

```

读者可以发现，以上代码的编写风格比较特别，这是周期精确的行为级描述，是编写 BFM 的最常用方式，如果不是和硬件交互，应尽量避免使用周期精确的行为级描述，因为它描述了对象在每个时钟周期的行为，因此比一般的行为级描述复杂，在多数场合是不必要的。

有了 BFM，读写 I²C 总线控制器中的寄存器时，只需要调用相应的 task，不用逐一驱动相关的引线，非常方便。例如写控制寄存器 MBCR：

```
uc_write_i2c(I2C_MBCR_ADDR,'mbcr);
```

读状态寄存器 MBSR：

```
uc_read_i2c(read_out,I2C_MBSR_ADDR);
```

如果 μ C 和 I²C 总线控制器之间的接口发生变化，例如改变了引脚名称，那么只需要改变 BFM 里相关的代码，非常简便，大大降低了发生错误的机会。

10.4.4.4 仿真验证

I²C 总线控制器的仿真验证步骤包括功能仿真、综合后仿真和布局布线后仿真，仿真时使用同一个 Testbench。如果 Testbench 和 DUT 使用的是同一种 HDL 语言，可以直接把 Testbench

也加入到 ISE 中，通过 ISE 调用 ModelSim 完成各个仿真验证步骤。对于混合语言仿真，ISE 不允许在一个工程中存在两种 HDL 语言，因此只能单独使用 ModelSim 进行仿真。

做功能仿真时，在 ModelSim 中建立工程，把 I²C 的所有 VHDL 文件，以及 Testbench 用到的 Verilog 文件加入到工程中，然后调整编译顺序，保证首先编译 Verilog 代码，然后才编译 VHDL 代码，如图 10-30 所示。

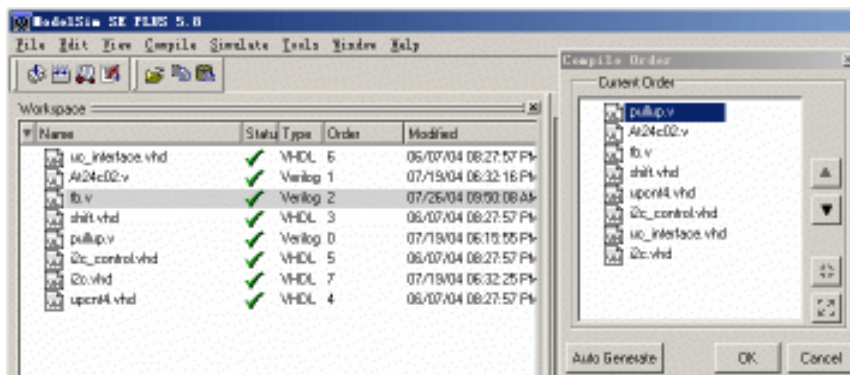


图10-30 功能仿真使用的文件及编译顺序

仿真波形如图 10-31 所示。

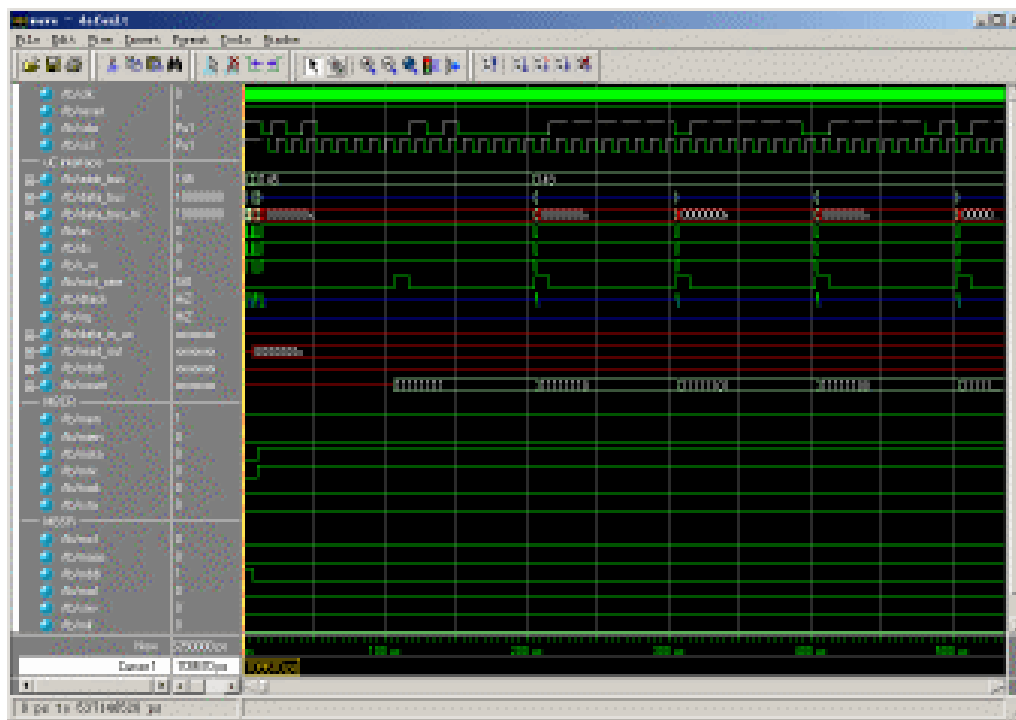


图10-31 功能仿真波形

在仿真调试的过程中，波形观察器是必不可少的工具，它能提供信号状态和变化的详细信息，但是依靠观察波形来判断 DUT 工作正常与否是非常繁重和容易出错的，在编写 Testbench 应尽量实现响应检查自动化。

响应检查自动化的实现方法主要有波形比较、数据文件比较和自检查 Testbench 等，其中波形比较使用仿真器的波形比较功能比较 DUT 仿真的波形和参考波形是否一致，从而判断 DUT 功能是否正确。在功能仿真过程中，假设某个版本的 RTL 代码通过了验证，得到了正确的仿真波形，可以把这个波形保存起来，如果代码需要修改，那么可以把修改后的 RTL 仿真结果和该波形进行比较，波形观察器会自动显示两者之间的差异，设计者可以从判断修改后 DUT 功能是否正确。把综合/布局布线后的仿真波形和 RTL 代码的仿真波形做比较，可以判断综合/布局布线后 DUT 的功能是否发生了改变。

数据文件比较的方法比较适合以数据处理为主的设计，例如图像处理器、编码/译码器、数字信号处理器等，方法是首先使用其他工具、语言产生参考激励和参考响应，分别保存到数据文件中，然后使用参考激励驱动 DUT，Testbench 在仿真过程中比较 DUT 的响应和参考响应是否一致，或者把处理结果也存到数据文件中，然后使用文件比较命令比较 DUT 输出的数据文件和参考响应数据文件是否一致，以判断 DUT 的功能是否正确。

总的来说，波形比较和数据文件比较需要较多的人为干预，自动化程度不够高，在大多数情况下，使用自检查 Testbench 是更好的选择。仿真时，自检查 Testbench 根据实际的激励产生参考响应，同时比较 DUT 响应和参考响应是否相同，然后给出提示信息，以便设计者判断。典型的自检查 Testbench 的结构如图 10-32 所示。

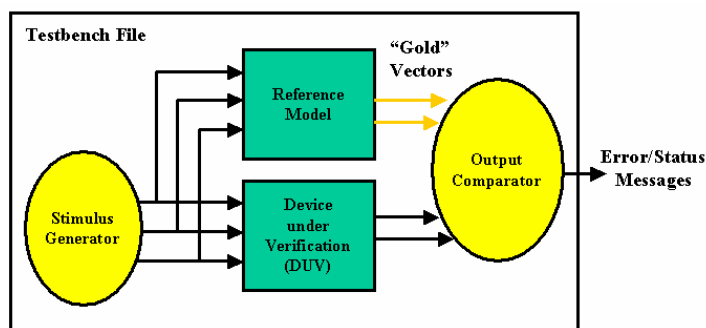


图10-32 自检查 Testbench 的结构

图中“Reference Model”是参考模型，其行为和 DUT 相同，一般使用行为描述实现，并经过一定的验证（或者自身足够简单，正确性容易得到保证），可以为结果比较模块提供参考输出（即 Gold Vectors）。有了自检查 Testbench 后，仿真工作会变得非常轻松，设计者只需要观察提示的信息就可以判断 DUT 是否正常，不再需要分析繁琐的波形了。

我们为 I²C 模块的验证编写了自检查 Testbench，仿真时 Modelsim 的提示信息如图 10-33 所示，从中可以清楚地看出整个验证过程，可以很方便地判断 I²C 接口控制器工作是否正确。

为了做综合后仿真以及布局布线后仿真，需要生成相应的仿真模型，方法是在 ISE 中建立工程，把 RTL 代码加入到工程中，根据使用的仿真器设置“Simulation Model Target”，如图 10-34 所示。然后双击“Generate Post-Map Simulation Model”，ISE 生成的 i2c_map.vhd 就是综合后仿真使用的 VHDL 文件。点击“Generate Post-Place & Route Simulation Model”生成的 i2c_timesim.vhd 和 i2c_timesim.sdf 为后仿真使用的仿真模型和反标文件。在

ModelSim 中，分别用这些文件代替功能仿真所用的 VHDL 代码就可以进行综合后仿真和布局布线后仿真了。

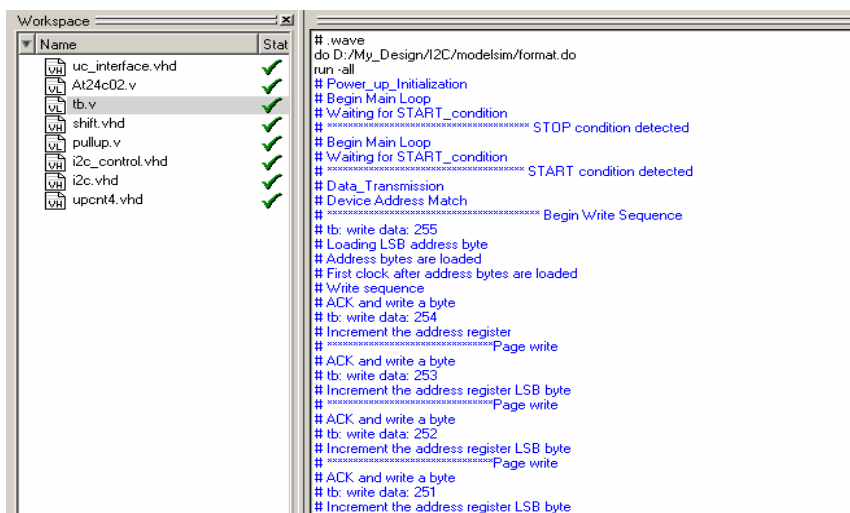


图10-33 功能仿真结果

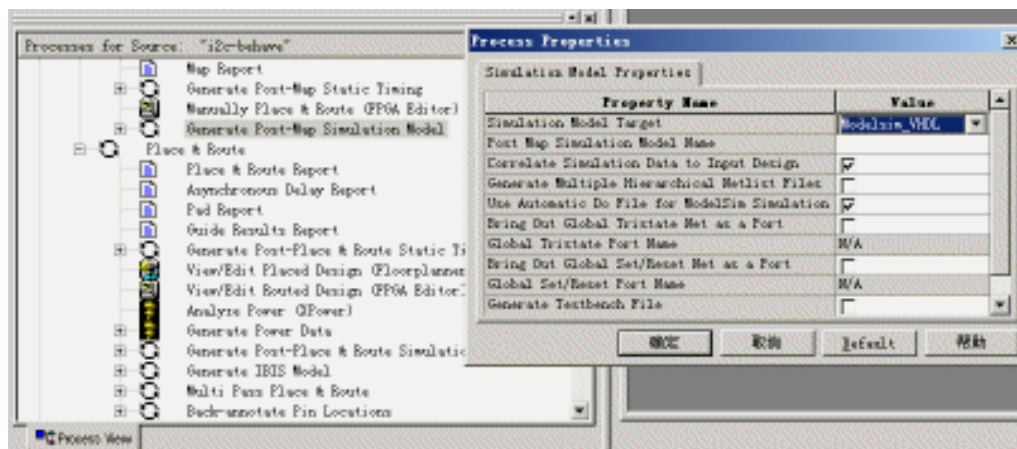


图10-34 产生 Post-map 仿真模型

如果使用 Synplify pro 做综合，在“Implementation Results”中选中“Write Mapped VHDL Netlist”后，会生成*.vhm 文件，该文件就是综合后的仿真模型，把后缀改为.vhd 后，就可以代替 RTL 代码在 ModelSim 中做综合后仿真了。

仿真所用到的文件、工程和仿真库请参考本书附带光盘的“Example-10-1”目录，读者可以把整个目录拷贝到自己的计算机上运行。后仿真要用到 SIMPRIM 和 VITAL2000 库，请读者检查当前工程是否已经引用了这两个库。

由于后仿真速度很慢，特别是设计比较大的时候，因此设计者要根据实际情况决定是否需要做后仿真。后仿真的目的是验证实现后的网表以及时序的正确性，一般而言，实现后的网表出问题往往是 RTL 代码风格不严谨所致，只要有好的代码设计风格，工具能正确产生网表。而时序的正确性可以通过静态时序分析解决，效率要比后仿真高得多。

10.4.5 上板调试

I²C 总线控制器仿真通过之后，就可以进行 FPGA 验证，即上板调试了。也许读者会问，后仿真通过了，还需要上板调试吗？答案是肯定的，主要原因是编写 Testbench 覆盖所有功能点有时候非常困难，可能也需要很多仿真时间。这也是在复杂设计中，验证工作要消耗 60%~70% 时间的原因，而且有时候 Testbench 并没有真实地模拟设计的工作环境，在这种情况下很难保证验证结果的正确性，因此做上板调试一般也是必要的。

I²C 总线控制器上板调试之前，首先要综合、布局布线和生成 BIT 文件，综合工具建议使用 Synplify Pro（创建 I²C 总线控制器工程时进行选择），功能强大、使用方便，可以用于多个 FPGA 厂家的产品，由于工作频率不高，不用加约束。

EFX-SP200 实验开发系统板上有 80C51 单片机，可以和 I²C 微处理器接口相连，以便对 I²C 总线控制器进行控制，如果读者的板子上没有微处理器和 FPGA 相连，那么使用起来就会非常不方便。用于调试 I²C 总线控制器的单片机程序并不复杂，可以说是把 Testbench 转换为 80C51 的 C 语言或汇编语言代码而已，详细的代码请到专业 EDA 论坛 (<http://www.edacn.net>) 下载。

如果读者没有接触过单片机，那应该找个机会学习一下。单片机非常有用，可以设计很多产品，可以为以后学习 DSP（如 TI 的 6416）、Microprocessor（如 Motorola 的 8260）打下坚实的基础，而 80C51 单片机是最常用的。市场上有非常多的教材可以参考，掌握单片机知识后，就可以用 EFX-SP200 实验开发系统设计包含硬件（FPGA，做高速处理）和软件（80C51 控制器，做较低速处理及控制）的复杂系统，逐步迈入 SOC 设计的殿堂。

10.5 小结

FPGA/IC 设计一般包含协议学习、结构定义、代码编写、仿真、调试等过程，一般的初学者由于缺乏实践机会，往往只关注 HDL 代码的编写，而轻视协议学习和深入细致的仿真调试，因而不能全面的掌握 FPGA/IC 设计技能。本章以 I²C 总线控制器的设计为例，详细介绍了 I²C 总线协议、I²C 控制器的编码、仿真和调试。通过本章的学习，读者可以比较清楚地了解 FPGA/IC 的设计流程，掌握微控制器接口设计、状态机设计、混合语言仿真、后仿真、EDIF 设计流程等技术，完成一个有工程实践意义的设计，成为一个真正的 FPGA/ASIC 设计工程师。

10.6 问题与思考

1. I²C 总线与其他并行、串行总线相比，有什么优点和不足？
2. I²C 总线控制器的作用是什么？
3. 用状态机完成设计有什么优缺点？什么样的设计适合用状态机完成？
4. 为什么要使用混合仿真环境？实现混合仿真验证的关键步骤是什么？
5. 使用 BFM 构建 Testbench 有什么优点？