

目 录

第一章	绪论	(1)
§ 1.1	可编程 ASIC 综述	(1)
§ 1.2	可编程 ASIC 的分类	(5)
§ 1.2.1	从互连特性上分类	(5)
§ 1.2.2	从可编程特性上分类	(5)
§ 1.2.3	从器件的容量上分类	(6)
§ 1.2.4	从结构的复杂程度上分类	(6)
§ 1.3	可编程 ASIC 的结构分类	(7)
§ 1.3.1	简单 PLD	(7)
§ 1.3.2	复杂 PLD	(7)
§ 1.3.3	FPGA	(8)
第二章	可编程逻辑器件的分类和特点	(10)
§ 2.1	概述	(10)
§ 2.2	可编程逻辑器件的阵列结构	(10)
§ 2.1.1	PROM	(11)
§ 2.1.2	PLA	(12)
§ 2.1.3	PAL 和 GAL	(12)
§ 2.1.4	EPLD	(13)
§ 2.3	复杂可编程逻辑器件	(14)
第三章	可编程门阵列的分类和特点	(18)
§ 3.1	概述	(18)
§ 3.2	查找表型 FPGA	(20)
§ 3.3	多路开关型 FPGA	(25)
§ 3.4	多级与非门型 FPGA	(26)
第四章	可编程 ASIC 的编程元件	(29)
§ 4.1	熔丝型开关	(29)
§ 4.2	反熔丝型开关	(30)
§ 4.3	浮栅编程技术	(32)
§ 4.3.1	EPROM	(32)
§ 4.3.2	EEPROM	(32)
§ 4.3.3	闪速存储器(Flash Memory)	(33)
§ 4.4	晶体管开关	(35)
§ 4.4.1	五管静态存储单元	(35)
§ 4.4.2	考虑供电电压的起伏	(36)

第五章	可擦除式可编程逻辑器件(EPLD)	(39)
§ 5.1	概 述	(39)
§ 5.2	XC7200A 系列	(40)
§ 5.2.1	概述	(40)
§ 5.2.2	结构	(40)
§ 5.2.3	应用特性	(44)
§ 5.3	XC7200系列	(44)
§ 5.4	XC7300系列	(45)
§ 5.4.1	概述	(45)
§ 5.4.2	结构	(46)
§ 5.4.3	应用特性	(50)
第六章	可编程门阵列	(52)
§ 6.1	概述	(52)
§ 6.2	逻辑单元阵列—LCA	(53)
§ 6.3	XC4000系列 LCA	(55)
§ 6.3.1	XC4000系列的 CLB	(55)
§ 6.3.2	XC4000系列的 I/OB	(57)
§ 6.3.3	XC4000系列的布线资源	(59)
§ 6.3.4	边界扫描电路	(63)
§ 6.4	XC3000/3100系列 LCA	(66)
§ 6.4.1	XC3000器件的 CLB	(66)
§ 6.4.2	XC3000器件的 I/OB	(67)
§ 6.4.3	XC3000/XC3100系列的布线资源	(70)
§ 6.5	XC2000系列 LCA	(73)
§ 6.5.1	XC2000器件的 CLB	(73)
§ 6.5.2	XC2000器件的 I/OB	(74)
§ 6.5.3	XC2000系列的布线资源	(74)
§ 6.6	XC5200 FPGA 系列	(75)
§ 6.7	XC8100 FPGA 系列	(77)
§ 6.7.1	特性概述	(77)
§ 6.7.2	结构	(78)
第七章	XilinxEPLD 和 FPGA 性能特点与比较	(82)
§ 7.1	FPGA 与 TTL 电路, PAL, 门阵列的比较	(82)
§ 7.2	Xilinx 的 EPLD 和 FPGA 性能比较	(83)
§ 7.3	XC4000与 XC3000比较	(85)
§ 7.4	XC4000H 与 XC4000的比较	(86)
§ 7.5	XC3000A/L 及 XC3100与 XC3000的比较	(86)
§ 7.6	XC5000与 XC4000的比较	(88)
§ 7.7	EPLD 与 FPGA 设计方法的比较	(89)

§ 7.7.1	逻辑综合	(89)
§ 7.7.2	状态机设计	(89)
第八章	Xilinx 的 FPGA 开发系统	(93)
§ 8.1	设计流程概述	(93)
§ 8.2	设计输入	(97)
§ 8.2.1	原理图输入	(97)
§ 8.2.2	文本输入	(98)
§ 8.2.3	层次设计	(99)
§ 8.2.4	控制实现	(100)
§ 8.3	设计实现	(100)
§ 8.3.1	XNF 转换	(100)
§ 8.3.2	优化	(101)
§ 8.3.3	合并	(101)
§ 8.3.4	映射	(101)
§ 8.3.5	布局	(102)
§ 8.3.6	布线	(102)
§ 8.3.7	产生位流文件	(102)
§ 8.3.8	XACT 设计编辑器(XDE)	(103)
§ 8.3.9	设计的大小及性能	(103)
§ 8.4	设计验证	(104)
§ 8.4.1	验证工具概述	(105)
§ 8.4.2	仿真	(105)
§ 8.4.3	静态时序分析	(107)
§ 8.4.4	电路上验证	(108)
§ 8.4.5	读回功能	(111)
§ 8.5	先进的设计准则	(112)
§ 8.5.1	设计分割	(112)
§ 8.5.2	流水线	(114)
§ 8.5.3	平面布置	(115)
§ 8.5.4	硬宏单元	(117)
§ 8.5.5	锁定 I/O	(117)
§ 8.6	软件技术	(118)
§ 8.6.1	XACT 的性能	(118)
§ 8.6.2	PPR 选项	(121)
§ 8.7	XC8100 FPGA 的设计实现	(125)
第九章	Xilinx FPGA 的配置	(126)
§ 9.1	数据产生	(126)
§ 9.1.1	MakeBits	(126)
§ 9.1.2	MakePROM	(127)

§ 9.2	数据格式	(127)
§ 9.2.1	产生位流	(128)
§ 9.2.2	单个器件位流	(128)
§ 9.2.3	菊花链位流	(128)
§ 9.2.4	PROM	(128)
§ 9.3	配置模式	(129)
§ 9.3.1	主模式与非主模式比较	(129)
§ 9.3.2	XC2000/XC3000配置模式	(129)
§ 9.3.3	XC4000配置模式	(135)
§ 9.4	配置数据的加载和转成信息段	(139)
§ 9.4.1	菊花链的串行加载	(140)
§ 9.4.2	多个器件的并行加载	(141)
§ 9.4.3	加载交替配置	(141)
§ 9.5	XC2000和 XC3000器件的状态	(142)
§ 9.5.1	加电和初始化	(142)
§ 9.5.2	清除	(142)
§ 9.5.3	配置	(143)
§ 9.5.4	起动	(143)
§ 9.5.5	重新编程	(145)
§ 9.6	XC4000器件的状态	(145)
§ 9.6.1	加电和清除存储器	(145)
§ 9.6.2	初始化	(145)
§ 9.6.3	配置	(146)
§ 9.6.4	起动	(146)
§ 9.6.5	重新配置	(148)
第十章	XATC 设计编辑器(XDE)	(149)
§ 10.1	XDE 的实现过程	(149)
§ 10.1.1	设置模式	(149)
§ 10.1.2	设置目录	(149)
§ 10.1.3	选择设计文件	(149)
§ 10.1.4	加载设计	(149)
§ 10.1.5	剖视 FPGA	(150)
§ 10.1.6	设置轮廓(profile)	(151)
§ 10.1.7	利用 EditBlk 显示	(152)
§ 10.2	高层次编辑	(155)
§ 10.2.1	SwapSig	(155)
§ 10.2.2	SwapBlk	(157)
§ 10.3	低层次编辑	(158)
§ 10.3.1	屏幕选项	(159)

§ 10.3.2	开关矩阵.....	(159)
§ 10.3.3	可编程互连点(PIP)	(160)
§ 10.3.4	布线资源.....	(160)
§ 10.3.5	人工布线.....	(163)
§ 10.3.6	通过开关矩阵布线.....	(165)
§ 10.3.7	利用长线布线.....	(166)
§ 10.4	执行类似功能的指令.....	(167)
§ 10.4.1	Unroute 和 UnroutePin	(167)
§ 10.4.2	Route 和 RoutePin	(168)
§ 10.4.3	Route, EditNet 和 RoutePin	(168)
§ 10.4.4	Hiligh 和 ColorNet	(168)
§ 10.5	建立四位乘法器.....	(169)
第十一章	XEPLD 的设计方法	(177)
§ 11.1	行为设计概述.....	(177)
§ 11.1.1	行为设计方法概述.....	(177)
§ 11.1.2	利用 PLUSASM 产生新的设计	(177)
§ 11.1.3	利用 Xilinx ABEL 产生新设计.....	(178)
§ 11.1.4	利用第三方的 PLD 编译器	(178)
§ 11.1.5	变换 JEDEC 文件	(178)
§ 11.1.6	设计例子—变换 PAL 文件	(179)
§ 11.2	用 PLUSASM 产生设计	(181)
§ 11.2.1	PLUSASM 概述	(181)
§ 11.2.2	产生基本的设计.....	(183)
§ 11.3	变换 PAL 设计	(189)
§ 11.3.1	PAL 变换方法概述	(189)
§ 11.3.2	PAL 变换要求	(190)
§ 11.3.3	PAL 变换顺序	(191)
§ 11.3.4	编辑顶层文件.....	(193)
§ 11.3.5	利用方程报告校验设计时序.....	(198)
§ 11.4	原理图中利用 PLD 文件	(199)
§ 11.4.1	选择库元件.....	(199)
§ 11.4.2	选择 PLD 开发方法	(200)
§ 11.4.3	设计流程.....	(202)
§ 11.5	OrCAD 软件设计 Xilinx EPLD	(202)
§ 11.5.1	利用 XEPLD 软件	(202)
§ 11.5.2	在 DRAFT 中画设计原理图	(203)
§ 11.5.3	产生新的原理图.....	(205)
§ 11.5.4	定义 PLD 方程	(208)
§ 11.5.5	选配设计.....	(210)

§ 11.5.6	仿真设计	(210)
第十二章	FPGA 设计应用	(213)
§ 12.1	FPGA 设计应用实例	(213)
§ 12.2	Viewlogic 软件设计 Xilinx FPGA	(218)
§ 12.2.1	Viewlogic 软件概述	(218)
§ 12.2.2	由 XDM 到 WORKVIEW	(222)
§ 12.2.3	产生一个符号	(224)
§ 12.2.4	产生原理图	(225)
§ 12.2.5	VIEWSIM 使用说明	(226)
§ 12.2.6	原理图中可加属性表	(229)
§ 12.3	XACT 5.0 版本 FPGA 实验实例	(230)
§ 12.3.1	实验一:计数器设计	(230)
§ 12.3.2	实验二:计算器设计	(231)
§ 12.3.3	实验三:XACT 性能实验—COUNT	(234)
§ 12.3.4	实验四:沿边译码器实验—CALC	(235)
§ 12.3.5	实验五:MemGen 实验	(236)
§ 12.3.6	实验六:RAM 实验—CALC	(237)
§ 12.3.7	实验七:X—BLOX 实验—COUNT	(238)
§ 12.3.8	实验八:X—BLOX 实验—CALC	(239)
§ 12.3.9	实验九:Xilinx—ABEL 实验—COUNT	(241)
§ 12.3.10	实验十:Xilinx—ABEL 实验—CALC	(243)
§ 12.3.11	实验十一:仿真实验—CALC	(247)
§ 12.3.12	实验十二:RPM 实验—COUNT	(247)
§ 12.3.13	实验十三:RPM 实验—CALC	(248)
§ 12.3.14	实验十四:读回实验—COUNT	(248)
§ 12.4	FPGA 设计变换到 EPLD	(250)
§ 12.4.1	变换 FPGA 设计到 EPLD 的一般顺序	(250)
§ 12.4.2	变换行为描述模块	(251)
§ 12.4.3	变换 FPGA 设计到 EPLD 的例子	(251)
附录	附录	(267)
附录 A	XBLOX	(267)
§ A.1	X—BLOX 的特性	(267)
§ A.2	产生 X—BLOX 的设计	(268)
附录 B	Xilinx FPGA 器件的引脚	(278)

前 言

八十年代末,在开展系统集成的研究工作时,由于受国内半导体工艺的制约,而不得不寻求新的出路,恰逢此时,国外出现了现场可编程门阵列(FPGA)等器件,其用户可编程的特性和类似门阵列的高集成度,为进行系统集成的设计者提供了不受半导体工艺制约,独立进行设计工作的“一线生机”。当时从收集和掌握到的资料进行分析和比较,选择了几种产品开始探索,XILINX的FPGA是在走访香港科汇公司时,拿到了提供的样片而开始试用的,于1991年开发成功第一个设计,真正体会到FPGA的优越性。

巧合的是在茅于海教授的推荐下,开始还选用了Plus Logic公司的FPGA产品,正当设计应用遇到困难时,该公司已联系不上,但不久它归入XILINX公司,其产品成为XILINX目前的EPLD,这样使我们在两个方面对XILINX的产品都有所了解。

在以后的几年里,一方面XILINX的产品发展很快,另一方面许多厂家也争相开发其可编程ASIC的产品,因此到目前为止,可选的品种和类型越来越多,但XILINX作为第一家FPGA的发明者,它在多年用户需求的基础上开发的品种系列更多,性能也更优越,加上售后服务周全,用户涉及面广,资料实例详尽,是开发和研制产品时最优选者之一。

参考国外各著名大学电机和计算机工程系本科生、硕士生和博士生的教学大纲,在数字电子学和逻辑设计等课程中,利用XILINX的可再编程FPGA器件对教学实验带来很大的方便,已成为教学不可缺少的内容之一,因此我们从92年开始也在教学中引入了FPGA的内容,开设了相应的实验。

但是真正能满足教学的需求,还是到1994年9月与香港科汇(亚太)有限公司建立了清华—科汇(亚太)XILINX培训中心之后,能更大范围地改革教学和增设相应的实验,同时也为科研工作提供了有力的设计手段。培训中心也为国内推广应用可编程ASIC技术做了些应做的工作,通过每月一期的培训班与许多用户建立了联系,了解到他们的许多需求。本书就是在近几年教学和技术交流的基础上,根据用户使用的要求而编写的,全书力求内容新颖,技术先进,实例丰富,既能反映当前可编程ASIC发展的最新水平;还要介绍数字系统集成的最新手段;以及提供能为教学和培训的最新方法,使其成为一本具有实用价值的高新技术的应用参考书和教科书,为推动我国电子产业在系统集成方面紧跟当前世界的发展水平作出一些贡献。

本书编写过程中得到了香港科汇(亚太)有限公司杨飞先生的大力支持,提供了许多实用和最新的资料,并对全书内容提出了许多宝贵的意见。研究生周朝辉、余志勇、郑立中、马青中,信息工程学院本科生汪晓东,宋颖凤,李志勇,顾素闲在整理资料和实验内容上都为本书作了大量工作,在此一并致谢。

本书编写出版过程中得到电子工业出版社和清华大学电子工程系的大力支持,特此表示感谢。

由于编写时间紧,技术发展更新快,编者水平有限,书中错误之处恳请读者批评指正。

编 者

一九九五年八月于清华大学

第一章 绪 论

§ 1.1 可编程 ASIC 综述

为专门限定的产品或应用而设计的芯片被称为专用集成电路(ASIC),除了全定制的专用集成电路之外,目前有五种半定制的元素可实现专用集成电路的要求,它们是:

- (1) 可编程逻辑器件(PLD)
- (2) 复杂可编程逻辑器件(CPLD)
- (3) 现场可编程门阵列(FPGA)
- (4) 门阵列
- (5) 标准单元。

在这些器件中,尤其是前三种器件的出现,使得电子系统的设计工程师利用与器件相应的 CAD 软件,在办公室或实验室里就可以设计自己的 ASIC 器件,其中以近几年发展起来的 CPLD 和 FPGA 器件格外受到重视。这三类器件都具有用户可编程的特性,又能实现用户规定的各种专门用途,因此构成了可编程专用集成电路的一类器件。采用可编程 ASIC,对于半导体制造厂家可按照一定的规格以通用器件大量地生产,对于用户可按通用器件从市场上选购,再由用户自己通过编程实现 ASIC 的要求,由于这种方式对厂家和用户都带来好处而受到欢迎,发展也就特别迅速,已经成为一个很重要的实现 ASIC 的手段。

可编程 ASIC 中的可编程门阵列(FPGA)将半定制的门阵列电路的优点和可编程逻辑器件的用户可编程特性二者结合在一起,使其不仅包含大量的门电路,具有高的速度,使设计的电子产品达到小型化,集成化和高可靠性,而且器件具有用户可编程特性,大大缩短了设计周期,减少了设计费用,降低了设计风险。因此 FPGA 已经成为可编程 ASIC 中更受欢迎的一类器件。

如果说一个电子系统可以象积木块一样堆积起来的话,那么随着半导体技术的发展,进入九十年代之后,构造许多电子系统仅仅需要三种标准的积木块,这就是微处理器,存储器和包括 CPLD 和 FPGA 的可编程逻辑器件。电子系统设计的这场革命是从 70 年代开始的,当时存储器已经作为标准产品进入市场,而八十年代微处理器也成为一种标准的产品,微处理器及存储器作为电子系统中两个主要积木块,一直都是可编程的。在电子系统中以前尽管采用了微处理器和存储器,但是仍然需要大量的中小规模的半导体器件来进行系统的各种逻辑控制,直到最近的 10 年之内,随着可编程逻辑器件的出现,才给电子系统的逻辑控制提供了可编程的性能和灵活性,而可编程门阵列作为一种高密度,通用的可编程逻辑器件与它的开发系统一起更为电子系统的逻辑设计确立了一种新的工业标准。现在已有愈来愈多的电子系统的设计工程师用复杂可编程逻辑器件或可编程门阵列作为系统逻辑控制的第三个积木块,来实现一个电子系统。

实际上,半导体的 CMOS 技术的不断发展推动了电子系统逻辑设计的这一变革,历来互补金属氧化物半导体(CMOS)一直被认为速度太慢而不能满足高性能系统设计的需要,这些设计只能用一次可编程(OTP)双极型可编程逻辑器件(PLD)来完成。现今许多 CMOS

的可编程逻辑器件实际上已追上或超过了双极型的性能。同时还具有低功耗和可编程能力等吸引人的优点。

可编程 ASIC 器件的发展速度,以 FPGA 为例,从 1985 年第一个 FPGA 问世到 1993 年的前八年发展速度完全可以与微处理器和存储器的前十几年的发展速度相比,基本上处于二者之间,表 1.1 给出这三者在速度,密度和成本三方面的发展比较。

在 FPGA 的发展中,和微处理器一样,结构的改进可提高其性能;和 DRAM 一样,由于有很规则的排列结构,工艺的改进也促使性能提高。所以对今后可编程 ASIC 性能的提高,60%依靠结构上的改进,如有更有效的逻辑块,更好的互连资源和不断改进的软件,而 40%由于工艺的改进,如采用 0.5 μ m 以下的工艺,三层的金属连线等。如 Xilinx 已采用 MicroVia 工艺,将工艺和结构改进结合在一起,来提高 FPGA 的性能,降低其成本。1994 年,Xilinx 已生产出约 25000 可用门的 XC4025,估计到 1995 年末可达到 65000 可用门,预计到本世纪末可用门将达到 15 万门,平均年增长率为 42%。1993 年可编程 ASIC 的销售额占 CMOS 逻辑电路 120 亿美元总销售额的 8.3%,预计到本世纪末将增长到占 CMOS 逻辑电路的四分之一。

表 1.1 微处理器、DRAM 和 FPGA 的发展比较

	微处理器			DRAM			FPGA		
	1971	1985	年增长率	1970	1983	年增长率	1985	1993	年增长率
速度	指令数/秒(MIPS)			存取时间(ns)			系统时钟频率(MHz)		
	0.06	6	36%	300	80	10%	5	50	30%
密度	晶体管数/芯片			晶体管数/芯片			晶体管数/芯片		
	2300	27.5 万	41%	1000	28 万	54%	8.6 万	340 万	58%
成本	美元数/MIPS			价格/位 $\times$ 存取时间			价格/门数 $\times$ 时钟频率		
	5030	50	29%			31%			30%

目前可编程 ASIC 正朝着为设计者提供系统内可再编程(或可再配置)的能力方面发展,即可编程 ASIC 器件不仅要具有可编程和可再编程的能力,而且只要把器件插在系统内或电路板上,就能对其进行编程或再编程,这就为设计者进行电子系统设计和开发提供了可实现的最新手段,而在以前是不可能做到的。采用系统内可再编程的技术,使得系统内硬件的功能可以象软件一样地被编程来配置,从而可以实时地进行灵活和方便的更改和开发。这种称为“软”硬件的全新的系统设计概念,使新一代的电子系统具有极强的灵活性和适应性,它不仅使电子系统的设计和产品性能的改进和扩充变得十分简易和方便,而且使电子系统具有多功能性的适应能力,从而可以为许多复杂的信号处理和信息加工的实现提供新的思路和方法。

应用可编程 ASIC 在系统中可再编程或可再配置的能力可以有多种多样的方法。一个较普遍的用途是实现系统的诊断,即实现系统逻辑的同一个 FPGA 可以再配置成测试系统的诊断逻辑,诊断测试逻辑基本上是“无成本”的,只是为额外的 FPGA 配置程序附加要求的存储器空间。这个方法已在许多包含诊断逻辑的应用中采用,如计算机外设、工业控制、医学仪器和集成电路测试仪等。

可再配置逻辑的另一广泛用途是使单个硬件可适合多种任务和环境。在这类系统中,任何大量可能配置的程序可加载到系统的 FPGA,以便按实际应用改变逻辑或按需要进行运行,所以较多的功能可用较少的部件来实现,硬件设计成本可以在大量的系统中分期偿还,设计周期也大大缩短。在某些系统中,可再配置的 FPGA 是系统运行的规范部件,这意味着系统有某些“互不相容”的功能,在需要的时候从 FPGA 调进或调出。基于 SRAM 的这些硬件多用途的应用是成本特别有效的,因为用传统的逻辑器件本来需要至少两倍的硬件才能实现同样的功能。此类原理已在导弹制导系统、印刷机控制器和远程通信测试仪等中应用。

可再配置 FPGA 技术已经在构划着未来的计算,国外一些大学和研究所已经利用 Xilinx 的 FPGA 实现多用途,高速的协处理器来加速计算机系统的运行,对于专门的应用,可再配置 FPGA 可把超级计算机的性能带到桌上系统,不难预测,未来的计算机自动地改变其硬件以适合正在运行的程序,被称为在最终用户的产品中实现“设计的可再编程性”(Engineered Reprogrammable)的年代已经开始。对于计算密集的算法协处理器将它们从主机的处理器中卸载下来,即主程序由主处理器执行,而某些任务分配到协处理器加速其执行,利用 FPGA 构成可配置协处理器意味着不同的任务可时分多路到相同的芯片达到加速的功能,针对协处理器应用的需要,Xilinx 公司已开发出 XC6200 系列的可再配置协处理器,它不象其它 FPGA 通过串行口或八位并行口加载其配置模式,而是有一个类似主处理器存储器的接口,可以设置成 8、16 或 32 位操作,这个快映射(FastMAP)接口直接与大多数嵌入式处理器接口而不耗费任何 FPGA 的资源,它允许配置数据以数据总线的速度转移,整个配置文件的转移少于 200 μ s;协处理的真正特性意味对不同的算法要求数量不等的存储器,按此需要提供了高容量分布式存储器,使其有比其它 FPGA 多 8—10 倍的存储器,用户可作逻辑或存储器的分配来适合每个算法;作为可再配置协处理器,算法可改变的速度是决定性的,它包括的片内存储器可以比普通基于 RAM 的 FPGA 快 1000 倍地转换应用的内容,也可以实现复杂的数据通道功能,在主机总线上的数据转移要求每次写入到配置存储器为 80ns,如果仅仅部分芯片的功能需要在系统运行期间的任何时候作修改,部分再配置同样是可能的;为加速规则功能(多位数据通道和其它重复的功能)的配置,XC6200 包括把配置数据按规定路线发送到多处的通配符(Wild-card)寄存器。由于以典型的数据总线速率转移数据,XC6200 器件可以比主机一般中断服务更少的时间再配置其自身,高速的配置或再配置允许可编程器件适合需要对条件变化快速响应的系统,除类似存储器的主机接口之外,FPGA 内部是专门的高速总线,它允许对到寄存器和大量的内部 RAM 高速存取。

可再配置计算技术在科学计算、设计自动化、数据库管理、图象处理和密码技术等方面得到应用,可编程 ASIC 的发展使电子技术进一步成为当今信息时代功能强大的技术手段。

可编程 ASIC 器件规模的不断发展,使其可以实现电子系统的集成,对于复杂系统的快速和准确的设计必须依赖计算机辅助设计(CAD),由顶向下的设计和综合工具。

低层的综合工具采取逻辑描述,并执行逻辑优化,进行映射的器件选择,布局和布线的网表优化,产生最终的设计实现。虽然综合工具一般包括从原理图作图输入的能力和较高层的工具,逻辑描述基本上是根据布尔方程。所有可编程逻辑销售厂家提供低层的综合工具,选择适合器件特性的优化和映射算法,以便把逻辑设计装配进其产品。低层的 FPGA 工具已作为与销售无关的 CAD 软件包内的选配器(Fitter)。布局和布线的问题只是在 FPGA 中才成为问题,而且当它们的单元规模增加和布线资源变复杂时也变得更严重,一个明显的趋

势是增加硬宏单元(hard Macro)的使用,定义一个公共逻辑函数的固定布局和布线,这样节省计算时间,也允许预先估计性能,但也可能造成单元利用率较低的缺点。PLD结构的逻辑最小化基本上由布尔表达式中文字数目最小化直接决定的。在FPGA中,限制到逻辑单元的扇入是更重要的因素。

对于简单的设计,原理图输入或布尔方程可能是适合一个设计的起点,对于复杂的设计,它可能刚好用少数最新一代的现场可编程器件来实现,但设计的模型是繁杂的,也增加错误的可能性,因此考虑高层的设计输入是必要的。控制器综合工具采用寄存器转换语言(RTL),有限状态机(FSM)或硬件描述语言(HDL)结合被选的结构进行描述,分配状态变量,产生逻辑描述。一般为了把HDL描述转换到组合逻辑和寄存器元件,主要的任务是执行二进制状态分配。对于PLD的结构,问题是清楚的,基本上是找出最小状态分配来解决;对于FPGA问题是复杂的,"one-hot"编码对查找表型FPGA是适合的,因为每个逻辑功能块有一个触发器是可用的。

高层综合工具采用高层的行为描述,如VHDL或编程语言,并把它扩展到RTL,FSM或HDL。行为描述不需要说明有关设计实现方式的任何事情,虽然当特定的结构模板被预先规定时,结构事实上可以隐含地被描述。高层综合包含选择特定的结构模板,然后执行资源分配,寄存器分配和定时,所以在高层设计期间,设计者基本在速度和资源之间作取舍,例如,相同的行为设计,当速度不是临界时,可以由简单的微处理器实现;当速度是主要因素,且有足够的硬件资源可用时,用完全流水的逻辑设计来实现。从一个合意的行为描述开始的好处是设计者能够探索较广范围的实现选择,找出哪一个最适合特定的实现环境,同时,高层设计方法的关键是顶层的描述可能比低层的描述更容易理解和测试。

电子设计自动化的软件已经从CAD,CAT,发展到目前为"电子系统设计自动化(ES-DA)"的第三代,软件的工作平台也从小型计算机,到工作站,到高性能的PC机,一般都包含了符合IEEE-1076标准的VHDL高层次综合工具,这些都为ASIC和可编程ASIC的设计和实现带来了极大的方便,特别对于中小规模的系统集成,可编程ASIC器件成为首选的方案,这也是可编程ASIC器件得到广泛应用的原因。

可编程ASIC独特的器件性能和广阔的应用前景使许多半导体厂家和公司纷纷投入力量开发出愈来愈多的器件和产品,所以本书的选材上,将主要是围绕复杂可编程逻辑器件和可编程门阵列来讨论可编程ASIC技术的内容,在众多的产品中以生产了世界上第一个现场可编程门阵列的Xilinx公司的产品为主作介绍。Xilinx的产品包括可擦除的复杂可编程逻辑器件(EPLD)和基于SRAM的可编程门阵列(FPGA),由于它们具有可再编程的能力和其FPGA更具有在系统内再编程的能力而格外受到重视,得到了广泛地应用。在Xilinx的开发系统中,可以方便地在EPLD和FPGA之间进行选择,按照实际需要编程合理地实现设计的要求。此外,Xilinx一直按照广大用户的需要从工艺和结构上不断改进开发出众多的新产品,不仅使其一直保持了在这一领域领先的地位,而且其产品有典型性和代表性,适合于作为可编程ASIC技术的内容来介绍。全书分为三大部份,第一部份为可编程ASIC的分类和结构特点,并在此基础上介绍Xilinx的FPGA和EPLD的结构和特点。通过多种途径的比较使读者能对其它厂家的产品也有了解,以便有助于合理选择可编程ASIC器件;第二部份为以Xilinx的开发系统介绍可编程ASIC的设计过程;第三部份结合Xilinx开发系统的实验例子等说明可编程ASIC的应用。

§ 1.2 可编程 ASIC 的分类

可编程 ASIC 的器件包括一个“未配置的”逻辑单元阵列,此阵列可以安排成所需的逻辑功能或逻辑功能组,并可通过可编程互连接到所要求的输入输出,这些逻辑单元可以是门,反相器,触发器或较大的宏单元。连接到可配置开关系统的成型的信号线互连这些逻辑单元,开关系统可以采用熔丝,反熔丝或传输晶体管作为开关。

互连系统可以是内部的和/或外部的与逻辑单元连接,在许多结构中,逻辑单元是可编程逻辑元件(或微型 PLD)本身,每个可以有它自身的可编程局部互连系统。

可编程 ASIC 目前有许多不同的分类方法,下面介绍四种。

§ 1.2.1 从互连特性上分类

从互连特性上将可编程 ASIC 的器件结构分为确定型和统计型两大类:

即一类是类似 PAL 的确定型结构,它是由 PROM 的结构演变出来的;

另一类是类似于门阵列的统计型结构。

确定型结构的 PLD 包括 PROM、PLA、FPLA、PAL、GAL、EPLD 和 E²PLD。它提供的互连结构每次用相同的互连线实现布线。所以,这类 PLD 的定时特性常常可以从数据手册上查阅而不需要通过设计工具很快地被确定。实现的功能是否要布线,或所选器件上可用的可编程资源是否能达到信号的时序要求这些都可相对简单地确定。

另一类型的器件是首先由 Xilinx 和 Actel 公司推向市场的真正的现场可编程门阵列,它们被称为统计型结构是因为这种类型的设计系统每次执行相同的功能却给出不同的布线模式。所以设计系统必须允许系统工程师限制功能中关键通路的时序变化以使它们不超出技术条件的要求。

此外,为了布线在设计系统中可以指出信号的要求,如强调速度或者使逻辑单元的数目最小,对于最大速度的布线将使用较多的逻辑单元,把要达到的功能限制到最简单的器件上将牺牲速度。

按这种分类方法无论是确定型还是统计型的,只要可编程 ASIC 器件的等效门数超过 500 门以上都认为是高复杂度的 PLD,或高密度的 PLD。所以在此分类方法中,不论是 PLD,还是 FPGA,把可编程 ASIC 统称为可编程逻辑器件,而 FPGA 是最高复杂度的 PLD。

§ 1.2.2 从可编程特性上分类

无论是复杂可编程逻辑器件(CPLD)还是可编程门阵列(FPGA)目前均采用 CMOS 技术,它们为用户提供的编程手段有以下四种:

(1) 紫外线擦除和电气编程存储单元(UV EPROM),为紫外光互补金属氧化半导体 UVCMOS。

(2) 电擦除和再编程存储单元,一类是 EEPROM,为电气擦除式互补金属氧化半导体 EECMOS;另一类是结构与 UV EPROM 类似但是电擦除的闪速存储单元。

(3) 一次编程反熔丝或一次编程 EPROM

(4) 基于静态存储器(SRAM)的编程结构

所以从可再编程能力将它们分为两类:

一类是不可再编程的。确定型的 PAL 和统计型的反熔丝 FPGA 都是不可再编程的。

另一类是可再编程的。确定型的 GAL 和统计型中用 SRAM 的 FPGA 属于此类,有时将高密度的 PLD 和 FPGA 统称为 FPGA 来讨论,再按其采用的编程技术分类。

表 1.2 给出采用这四种编程技术生产 FPGA 和 CPLD 的代表性厂商。应该说明的是 FPGA/CPLD 的供应厂商已经不像几年前那样只提供单一工艺的产品,当前一个供应商在提供 SRAM 的 FPGA 系列的同时,可能还提供 Flash—EPROM 或反熔丝等工艺的第二个或第三个系列的产品,甚至除 5V 供电的产品外,还提供为便携式和许多工业系统要求的低功耗 3V/5V 兼容的产品,此外,所有类型的可编程器件的供应商已利用外设元件接口 (PCI) 总线作为必须提供的接口选择。

表 1.2 FPGA 和 CPLD 的供应厂商

反熔丝 FPGA	SRAM FPGA/CPLD	EEPROM	UV PROM
Actel	Altera		
Crosspoint Technol.	Atmel		
QuickLogic	(Concurrent Logic)	AMD	Altera
Texas Instr.	AT&T Microelec.	AMI	Atmel
Xilinx	Intel(包含 EPROM)	GateFiled Inc.	Cypress Semicond.
	Motorola Semicond.	Lattice Semicond.	Philips Semicond.
	Xilinx	National Semicond.	Texas Instr.
	Aptix(互连芯片)	SGS—Thomson	Xilinx
	Plessey Semicond.		
	Tosiba		

在第四章中还要专门介绍这四种编程技术,在该章最后的表 4.2 中,对于这四种编程技术给出了编程性能和技术特性方面的比较,以提供参考。正如 § 1.1 所述,可编程 ASIC 器件具有系统内可再编程的能力已成为其重要的技术特性,这一特性为用户提供了更为灵活和方便的开发 ASIC 的手段,其中基于 SRAM 的编程技术在这方面更为突出,许多新的应用正是利用了它的这一特性而实现的。

§ 1.2.3 从器件的容量上分类

由于可编程 ASIC 器件本身结构上和半导体生产工艺的不断改进和提高,器件的密度在不断增加,性能不断提高,目前可编程 ASIC 器件的门数已达到 25000 门之多。

从容量上对可编程 ASIC 器件分类是将复杂可编程逻辑器件(CPLD)和可编程门阵列(FPGA)统称为高容量可编程逻辑器件(HCPLD)。

§ 1.2.4 从结构的复杂程度上分类

从结构上对可编程 ASIC 进行分类不仅是最常用的方法,而且也是可编程 ASIC 器件设计实现和开发应用的需要,因为各类可编程 ASIC 器件的开发系统都是针对其结构来设计

的,了解和掌握可编程 ASIC 器件的结构,对合理和有效地利用开发软件来设计可编程 ASIC 器件就是重要的。

从结构上可将可编程 ASIC 器件分为以下三类:

1. 简单可编程逻辑器件(简单 PLD);
2. 复杂可编程逻辑器件(复杂 PLD);
3. 可编程门阵列(FPGA),也称为最复杂的 PLD。

在 § 1.3 中按此分类方法来进行专门的讨论。

§ 1.3 可编程 ASIC 的结构分类

可编程逻辑现在在种类繁多的器件中是可用的,为了方便,常从结构上将其分为简单 PLD,复杂 PLD 和 FPGA。但是这些分类之间的界限并不是很严格的,例如,在这里和第三章把 Altera 的 MAX5000 和 7000 系列作为 CPLD 来讨论,在第四章中又将其作为一类 FPGA 进行比较,而与它类似的 Xilinx 的 XC7000 系列仍列为 CPLD,但 XC7000 系列的前身是 Plus Logic 公司生产的,当时命名为 FPGA2000 系列,都说明了这一点。

§ 1.3.1 简单 PLD

简单 PLD 经典地是由与阵列及或阵列组成,能有效地实现以“积之和”为形式的布尔逻辑函数,当与及或两个阵列是可编程的,这个器件一般称为 PLA;其变型是 PROM 具有固定的与阵列,而 PAL 具有固定的或阵列。

PROM 包含所有 2^n 个可能的输入组合的乘积线,当器件变得很大时,逻辑应用仅需要这些乘积项小部份,更多的逻辑应用也仅需很少乘积项,,所以 PROM 至少现在仍广泛用于可编程逻辑应用。

由于并不需要将全部乘积项包含在每个输出表达式中,所以目前包含一个固定或阵列且其每个输出是和乘积项的子集合相连的 PAL 型式已经用得特别广泛。PAL 型式有两个有用的优点:第一有较少的需要被编程的连接,第二到输出线的连接数目较少意味着速度可以改善。

其它对 PLD 的一般选择包括寄存输出,阵列输出的内部反馈,可预置触发器,输出极性控制,具有全部阵列输出寄存的可编程时序对于“随机的”同步逻辑设计特别方便,最近 PLD 已达到大约 13 输入 $\times$ 92 乘积项 $\times$ 18 输出+16 内部反馈的规模。因此,除了地址译码和粘附逻辑(glue logic),简单 PLD 的规模现在已经大到足以实现很大部份同步逻辑和状态机的应用。

简单 PLD 基本结构的优点是没有布局布线的问题,性能完全是可以预测的,所以十分容易写出软件把设计放进器件或者在它们之间分割设计。因此软件相对便宜,执行快速。固定的资源意味着器件可以靠制造者对速度优化,保证最坏情况的延迟时间。另一方面制造者发现简单 PLD 要达到好的性能和器件很大时达到低功耗也增加了困难。

§ 1.3.2 复杂 PLD

为了增加电路密度而不蒙受性能或功耗的损失,PLD 器件结构已经引入了各种特性。

例如,在它们的 MAPL 结构中,为了保持规模高达 32 输入 $\times$ 128 乘积项 $\times$ 54 输出的 PLA 结构的灵活性,National Semiconductor 已引入分页系统(paging system),分页机构意味着仅仅阵列的一部份在任何给定的瞬间被加电。为了减少功耗的其它方案包括按备份模式放置阵列,或者靠变换检测自动地控制加电,或者利用外部指令。

在实现级上,与阵列及或阵列需要用缓冲器分开,因为这些一般是倒相器,在两个阵列中实际逻辑一般是相同的(例如 nMOS 或非逻辑),某些制造者已经引入折迭 PLA(folded PLA),仅用一个实际阵列,但可将乘积项反馈回阵列。这也允许在单个器件中实现多级(multi-level)逻辑。但是多次通过阵列将遭受更长的延迟。

折迭 PLA 的例子是 Signetics 的 PML 结构。

PLA 的性能最终是由与/或线的长度限制。此外,复杂设计常常象受组合逻辑表达式的大小限制一样受触发器可用的数目的限制。这些因素已经成为几个先进的 PLD 设计后面的驱动力,它们由更复杂的输入/输出逻辑块和由分层结构逻辑阵列来表现。

这类结构包括 Atmel(1991)的器件,AMD 的 MACH(1990),更大的可用器件来自 Plus Logic 和 Altera。Altera MAX7000 器件是最近这类中最大和最快的器件。最高的范围允许给出 1024 个宏单元(20000 门)和 70MHZ 性能。

基本上这些先进的 PLD 是由几个简单的 PLD 组成的,PLD 的输出进入包括一个触发器和一些控制逻辑的逻辑单元,单元输出然后通过一个纵横互连矩阵反馈到芯片的所有宏单元。这些被集成进 Atmel 设计的分层结构中,而在另一些器件中简单地是一个全连接的纵横开关。Plus Logic 和 AMD 利用了在一个宏单元中单元之间乘积项共享,使宏单元有效地把 PLA 和 PAL 的型式组合起来。而 Altera 和 Atmel 采用部份地折迭结构代替。Plus logic 在其逻辑单元中包括一个两输入查找表获得折迭连接提供的一些功能,而没有多次通过折迭阵列时间损失的风险。

显然,不可能获得象超出简单 PLD 一样超出分层 PLD 的最高性能,但是 Altera 已经达到十分接近,通过它们的互连矩阵仅有 2ns 的延迟。

同时,设计保持宽的逻辑门和高的 PLD 互连,而避免了 FPGA 的布局布线和多级实现问题。因此 CAD 支持仍然相对简单和运行快速。近乎固定的延迟使得预测系统性能变得简单。

因此先进的 PLD 对集成几个简单的 PLD 的功能和实现复杂状态机是有效的。

§ 1.3.3 FPGA

虽然 PLD 是十分灵活的,但许多设计一般并不是利用了它的全部功能。特别是随机逻辑,数据通道和信号处理设计常常不要求如此多的门,但是要求更多触发器或寄存器和快速的局部互连。为了达到这些要求,Xilinx 大约在 1984 年,而其它制造商最近做出由可编程逻辑块和可编程互连一起组成的现场可编程门阵列。最近可用的器件是来自 Xilinx, Actel(第二源自 Texas Instruments),GEC—Plessey Semiconductors 和 Algotronie Concurrent。

1. 逻辑块

Xilinx 已经选择了最复杂的逻辑块,如 XC3000 系列包括两个触发器和能实现五变量的任何函数的查找表。其它的 FPGA 器件是含有一个触发器和多路开关控制逻辑的较小逻辑块。

2. 编程技术

Actel 利用一次编程的反熔丝结构,它具有编程单元面积小和电阻低的优点,但不可再编程。其它的 FPGA 都是可再编程的,具有灵活,编程快速,有可再配制设计的能力等优点。此外,SRAM 的编程技术有比其它方法编程和再编程更快的优点,但存储是掉电丢失的。

3. 阵列形式

市场的领先者 Xilinx 和 Actel 利用通道结构,具有由布线通道分开的固有逻辑块。

Xilinx 阵列包含由局部,中等和长距离互连在行和列两个方向上分开的大逻辑块。

Actel 阵列包含由行互连分开的较小的行逻辑块。

GEC-Plessey Semiconductors 利用主要的局部互连和某些较长总线的门海阵列。

Algotronix 和 Concurrent Logic 利用毗连单元的阵列,前者仅有对接接触连接,而后者支持较长总线的连接。

1994 年 7 月 Xilinx 已经宣布其未来的 XC8000 系列采用全新的 MicroVia 工艺,它的反熔丝结构是金属到金属的反熔丝,并将三层金属的工艺和门海的结构结合在一起,有可能极大地减少 FPGA 的成本。

第二章和第三章从结构特点上对可编程逻辑器件(PLD)和现场可编程门阵列(FPGA)进行讨论。高复杂度的 PLD 是包含有宏单元的 PLD,结构主要类似于 PLD。由于通常是可擦除的,因此也称为 EPLD。

第二章 可编程逻辑器件(PLD)的结构

§ 2.1 概 述

可编程逻辑器件(PLD)是一个逻辑电路,它可以被用户编程来执行专门规定的逻辑功能。这些器件为逻辑设计者提供了一个十分吸引人的,对用标准 74 系列的离散逻辑芯片所作设计进行更新的手段。就象任何标准的 74 系列一样,PLD 器件也可以按标准产品目录在各地的销售市场上购买到。许多 PLD 的功能已经或多或少地变成工业标准,这表明它们可从许多销售厂家,按不同的速度——功耗——成本进行选择,有多种工艺的差别来选购。但是 PLD 还提供了标准 74 系列器件的离散逻辑从未提供的一些其本身固有的优点,即它们是用户可编程的,这表明用户买到的这种器件是处在未开发的状态,就像一个空白的 EPROM 那样没有被编程过的,然后经过类似于编程 EPROM 那样的过程,由用户对 PLD 编程来实现一个专门规定逻辑功能。这就极大地方便了用户,使用户可以将一些中小规模的器件的功能集成到一个或几个 PLD 中,因此 PLD 成为最早实现可编程 ASIC 的器件。

PLD 电路的两个基本部份是逻辑阵列的宏单元。逻辑阵列是 PLD 的用户可编程部份,它由与门,或门和反相器组成,输入信号是在逻辑阵列中布线通道上运行的。在第二代 PLD 上新增加的宏单元允许用户规定 PLD 的输出结构。最初的宏单元例子是能可编程输出极性的宏单元,这个宏单元是十分简单的:只要加一额外的熔丝到逻辑阵列的每个输出边就可实现,用户编程这个极性熔丝来配置输出信号的极性。下面首先讨论 PLD 的逻辑阵列的结构。然后再讨论带有宏单元的通常为擦除式的高密度和高性能的 PLD,即 EPLD。

§ 2.2 PLD 阵列结构

PLD 的主要部份是由两个逻辑门的阵列构成的,一个与阵列伴随一个或阵列。技术上,输入到 PLD 的信号必须首先通过一个与门阵列,在这里形成输入信号的组合,每组相与的组合被称为布尔代数的子项或 PLD 术语中的乘积线,这个乘积线在第二个或门阵列中被相加。这个逻辑信号现在对于输出宏单元是有用的,注意到输入信号的极性不是一个问题,因为真的输入信号和它的补都由输入缓冲器产生。亦就是影响门阵列的布局布线问题从不成为 PLD 的问题,因为布线是通过阵列来提供的。所以在 PLD 中 100% 的布通率总是保证的。

可编程逻辑器件中的与/或阵列结构在硅片上制造时是十分有效的,除此之外,它对于逻辑设计也是十分方便的,因为它直接实现“积之和”形式的逻辑表达式,逻辑表达式的积之和形式对逻辑设计者是十分熟悉的,因为在布尔代数中,与算子有高于或算子的层次,所以可以十分容易地利用逻辑设计技术来产生逻辑方程,如逻辑设计中常用的卡诺图和摩根定理。与/或逻辑的概念实际上在布尔代数中是有它的根据的,所有任意的逻辑函数都可以利用两级与或门来实现,对于十分复杂的逻辑这样得到的门电路可能是十分大的,对于离散逻辑的分立器件实现起来是一个问题,但是对阵列基础上的 PLD 不成为问题。

实际上,在与/或阵列的基础上有三种基本类型的 PLD,由哪个阵列是可编程的来区分

这三种类型：

- (1) 可编程只读存储器—PROM：与阵列固定，或阵列可编程。
- (2) 可编程阵列逻辑—PAL：与阵列可编程，或阵列固定。
- (3) 可编程逻辑阵列—PLA：与和或两个阵列都可编程。

§ 2.2.1 PROM

可编程只读存储器(PROM)的内部结构是由固定的与阵列及其输出加到一个可编程的或阵列组成的，如图 2.1 所示。因为与阵列是固定的，输入信号的每个可能组合是由连线连接好的，而不管此组合是否会被使用，所以在这方面 PROM 十分类似于一个查找表。

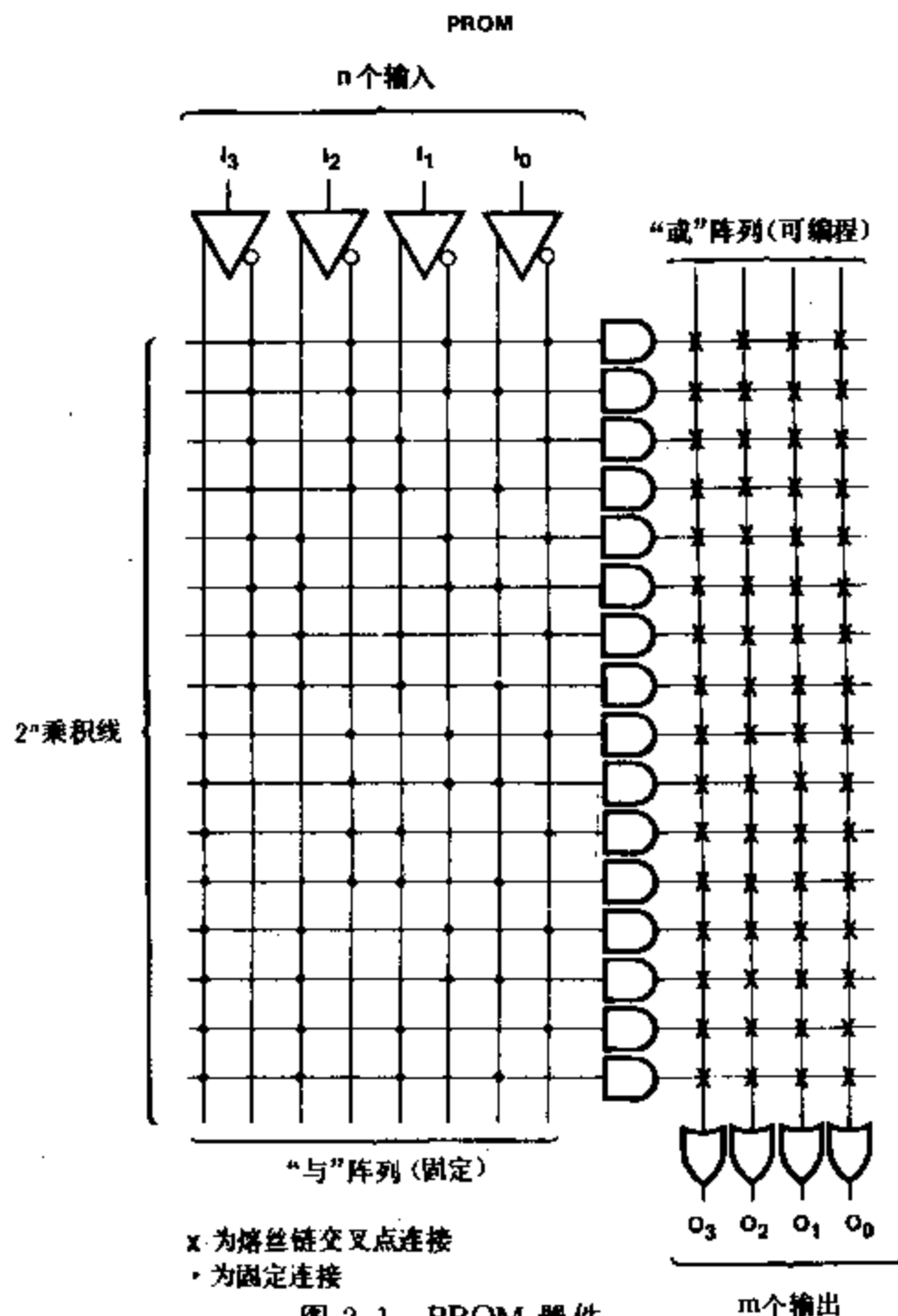


图 2.1 PROM 器件

因为每个输入信号组合都被译码，所以 PROM 的输入阵列结构可以为要求小数量的输入和许多组合项的逻辑应用很好地工作。因此对于如随机逻辑置换，译码器/编码器，错误检测和校正，查找表，和分布算法等小的逻辑应用 PROM 是一种速度快，成本低，编程容易的逻辑器件。在 PROM 中，随输入信号数量的增加，阵列会变得越来越大，因为 PROM 的规模随输入数量的增加按 2^n 指数地增长。所以当输入的数目变的太大时，变成大部件的 PROM 器件出现了许多缺点，如器件成本升高，功率消耗增加和可靠性降低等都成为问题。

§ 2.2.2 PLA(Programmable Logic Array)

在可编程逻辑阵列(PLA)中,与阵列和或阵列二者都是可编程的,图 2.2 给出 PLA 的结构。

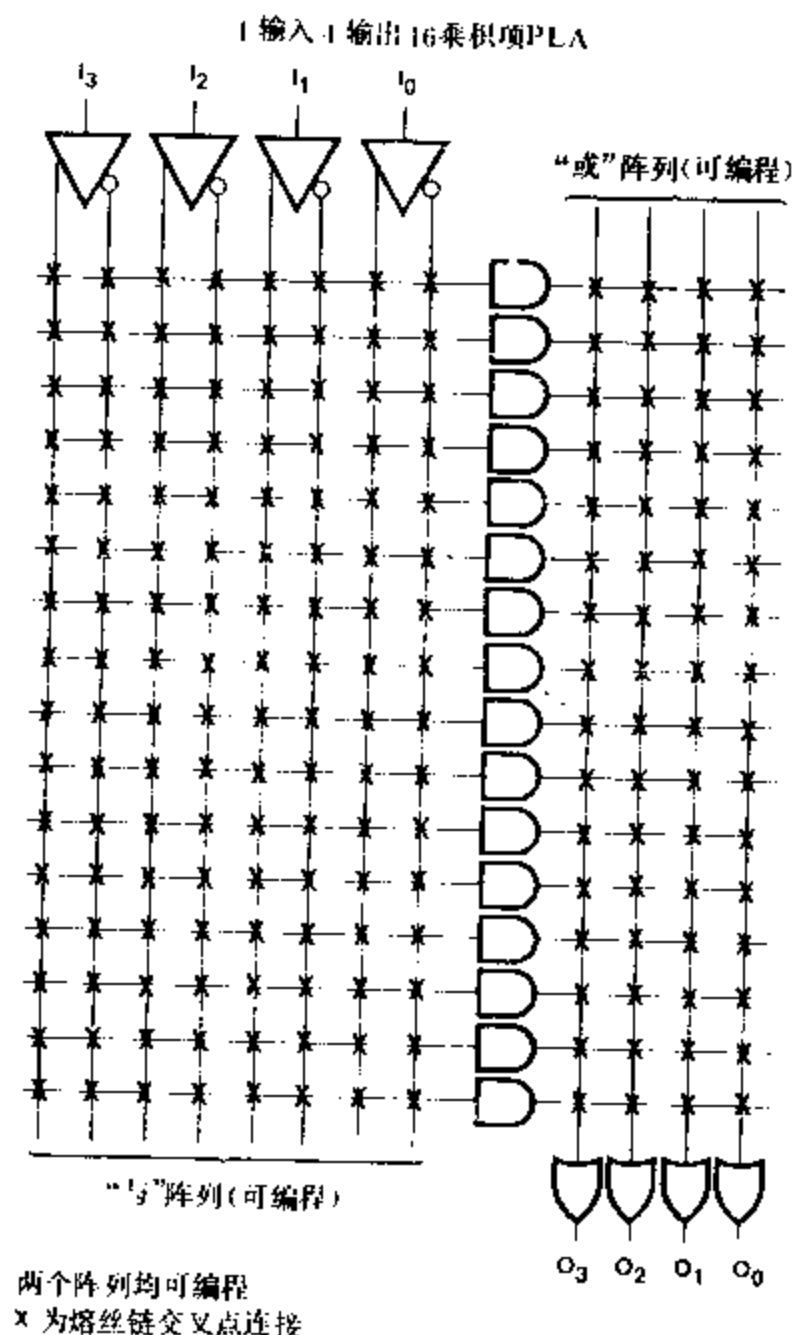


图 2.2 PLA 器件

PLA 的内部结构提供了在可编程逻辑器件中最高的灵活性。因为与阵列是可编程的,它不需要包含输入信号每个可能的组合,需要包含的组合仅仅是在逻辑功能中实际要求的那些组合。所以,在 PROM 中由于输入信号数量增加而使器件规模增大的问题在 PLA 中不存在。

PLA 对实现复杂的随机逻辑置换,如算术累加逻辑单元(ALU)和数据通道控制等是很理想的。但是,PLA 器件的灵活性被其很少称心的慢速特性和比仅有一个可编程阵列的器件有更多的耗费所抵销。利用 PLA 的另一缺点是在器件中有一个被限定的乘积项。

§ 2.2.3 PAL 和 GAL(Programmable Array Logic, Generic Array Logic)

PAL(可编程阵列逻辑)或 GAL(通用阵列逻辑)器件的结构在图 2.3 中被表示,它是由一个可编程与阵列及其输出加到一个固定的或阵列组成的。

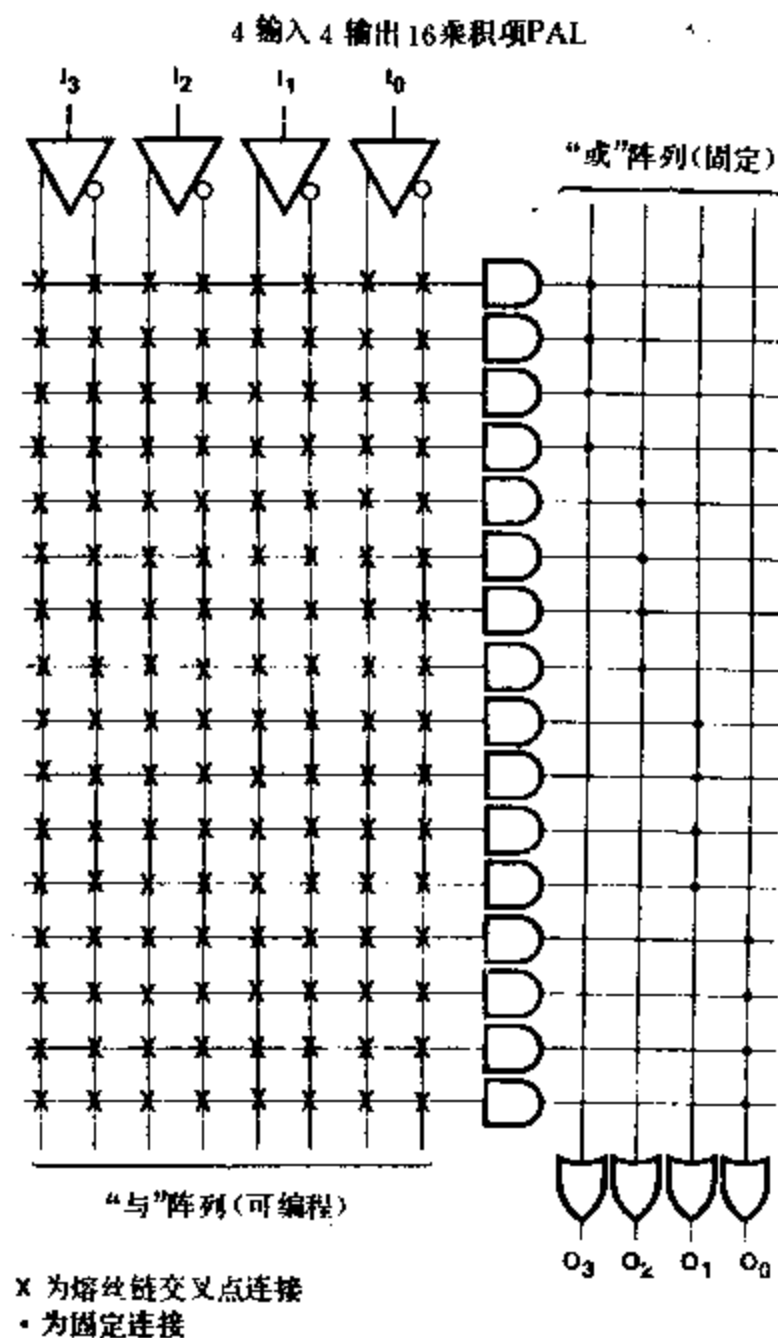


图 2.3 PAL 器件

所以 PAL 或 GAL 这类器件把 PROM 器件的成本低, 速度高, 编程容易的特性以及 PLA 器件的灵活性的优点结合在一起, 因此在早期成为实现可编程 ASIC 的主要器件, GAL 的可再编程特性, 更为开发提供了方便。

虽然所有的 PAL 和 GAL 器件有相同的内部结构, 这些类型的器件靠各种特性组合而被区别。器件的选择包括改变输入和输出腿的数目, 高有效或低有效的输出, 寄存的或组合的输出, 从输出反馈进阵列的反馈通道。

§ 2.2.4 EPLD

八十年代中期, Altera 推出 EPLD, 改进了 PLD 的结构, EPLD 为可擦除的可编程逻辑器件, 可擦 PLD 包括 EPLD(UV 可擦 PLD) 和 EEPLD(电可擦 PLD)。

EPLD 的结构特点是大量增加输出宏单元的数目, 提供更大的与阵列。随着与阵列的扩大和输出宏单元的增加而过渡到复杂可编程逻辑器件。

在说明了许多类型的 PLD 能力之后, 可擦 PLD 可以替代有相同出腿数目的几乎任何 PAL 或 PLA。

在表 2.1 中摘要给出各种可编程逻辑器件(PLD)所适合的应用范围。

表 2.1 可编程逻辑器件(PLD)应用摘要

器件	与阵列	或阵列	应用
EEPLD	可变	可变	大多数 PLA 和 PAL 的应用
PROM	固定	可编程	小的逻辑设计, 编码/译码, 查找表, 分布算法
PLA	可编程	可编程	状态机, 数据通道控制
PAL	可编程	固定	编码/译码
16L8			状态机
16R8			灵活的状态机, 混合输出应用
32V×10			嵌入状态寄存器设计

§ 2.3 复杂可编程逻辑器件的结构

前一节中, 对于所有被讨论的 PAL 器件的可编程能力被限制于与阵列, 换言之, 用户编程与阵列中的熔丝链来执行所需的逻辑功能。用户选择不同的输出结构要选用不同的 PAL 器件。

在这节中要讨论几种新的 PAL 器件, 它们包含熔丝来控制围绕输出结构的逻辑门, 用户可以选择一个器件并编程使输出成为寄存的, 或组合的, 或与 D 触发器相对的 J-K 触发器。选择这几个 PAL 器件是因为它们代表带有宏单元的新品种。

最小的 PAL 器件是十分简单的, 它们确实由可编程的与阵列伴随一个固定的或阵列所组成, 不带有寄存器或宏单元。每个器件类型由输入数目、输出数目和输出极性所区别。其优点就是它们的简单性, 直接用 PAL 器件代替小量的小规模器件(例如: 地址译码应用)。这些主要实现组合逻辑功能的早期 PLD 称为简单 PLD。

稍复杂一些的 PAL 提供了较大的与阵列和灵活的输出结构, 即包括下述输出结构:

1. 可编程 I/O: 允许用户对这些引脚编程或者作为输入或者作为输出只要在倒相的输出缓冲器之后经过反馈线连接实现。

2. 寄存器输出和反馈: 规则的状态序列(计数器和移位寄存器)随机状态序列。同步或流水线组合逻辑。

3. 异或门输出结构: —— 多功能计数应用。十分有效地建立大的计数器。

至少带有宏单元的 PAL 器件被称为复杂可编程逻辑器件, 有代表性的是最先带有宏单元的 PAL22V10, AMD 公司生产的 PAL22V10 是专为代替 24 出腿的 PAL 器件而设计的, 并且很快成为最畅销的 PAL 之一。目前 PAL22V10 已用它来作为划分 PLD 的参考, 可编程逻辑器件所包含的门数大于 PAL22V10 的门数, 就被认为是复杂或高级可编程逻辑器件。而且复杂可编程逻辑器件常常以等效为包含多少个 22V10 的功能块来决定它的规模。下面列举几个带有宏单元的 PAL 器件。

1. PAL22V10

包括熔丝链来控制围绕输出结构的逻辑门。

可编程输出极性: 由或阵列之后输出缓冲器为组合输出和寄存器(为寄存输出)之

前设置的异或门来实现的。

极性熔丝在熔丝接触时连接异或门的一个输入端到地依靠极性熔丝的通断,改变极性。

改变乘积项分配:每对宏单元有不同数量的乘积线。

在 PAL22V10 器件中,10 个出腿有输出宏单元,并有 12 个输入,10 个输出的任一个也可作为输入,每个宏单元有沿触发的 D 触发器和两个可编程的多路开关,两个新的乘积项可以预置或复位触发器。

2. PAL32VX10

可编程触发器(J-K,S-R,T,D)。

双的独立反馈。

3. PAL20RA10/PAL16RA8

扩充非同步能力寄存器 PAL 器件

可编程和硬连接三态输出

可编程时钟

非同步复位设置

可编程输出极性。

图 2.4 和图 2.5 分别是 PAL22V10 和 PAL32VX10 的宏单元。

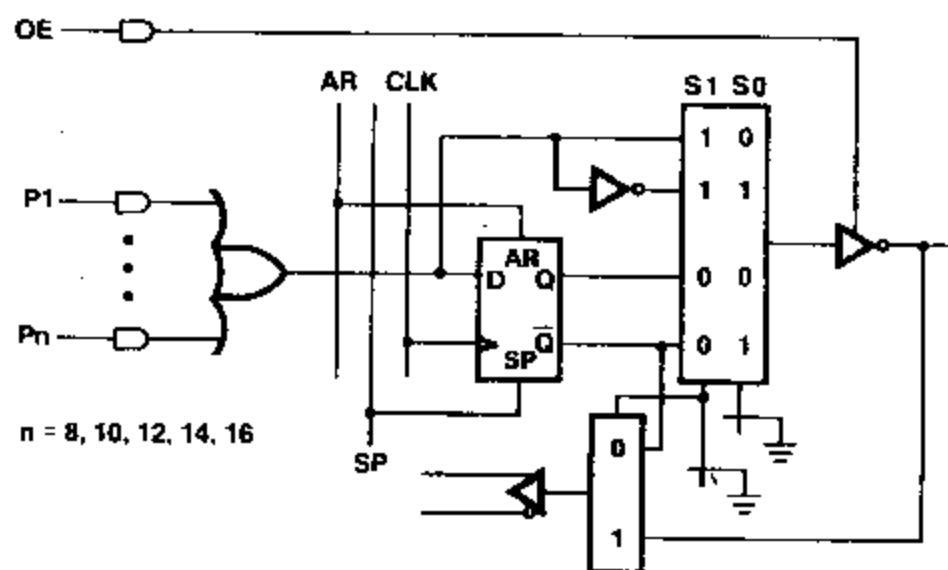


图 2.4 PAL22V10 的宏单元

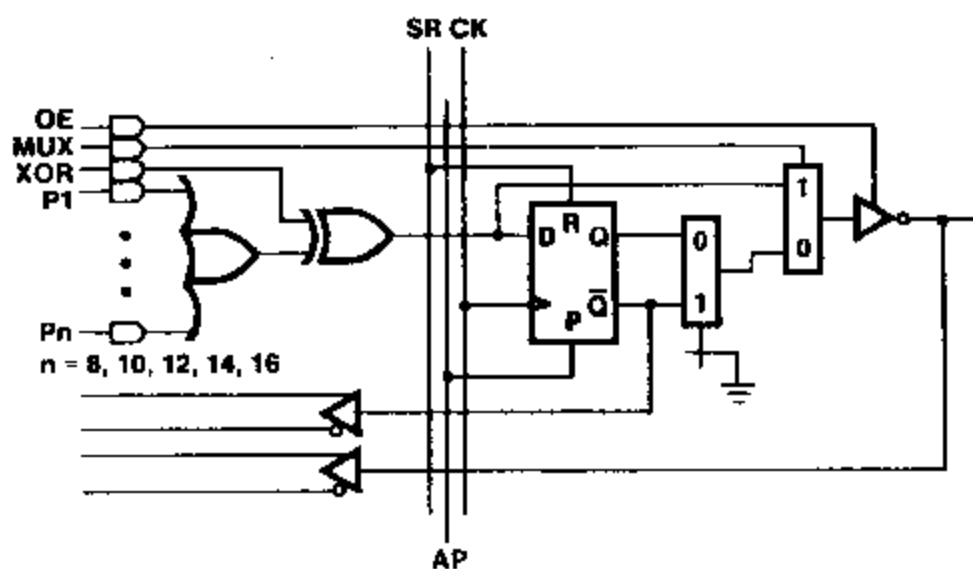


图 2.5 PAL32VX10 的宏单元

Lattice Semiconductor 公司推出了通用阵列逻辑 (GAL) 来取代 PAL, GAL 利用 E²PROM 单元来控制互连模式和作为开关的传输晶体管, 所以 GAL 是电可擦除的, 而且是可再编程及系统内可再编程的。八十年代 Altera 和 Plus Logic 公司推出 EPLD, 进一步发展了 PLD 的结构, EPLD 可用紫外光擦除, 以电方式编程, EPLD 的结构大量增加了输出宏单元的数目, 并提供了更大的与阵列。在第五章再详细讨论 Xilinx 的可擦除可编程逻辑器件, 由于 Xilinx 的 EPLD 相当于包含多个 PAL22V10, 所以也应是复杂的可编程逻辑器件。图 2.6 为 Altera 的 MAX EPM5128 的方框图。

九十年代 Lattice 公司又将其 UltraMOS 的 E²CMOS 工艺用于它的非丢失的系统内可编程高密 CPLD—ispLSI 系列。

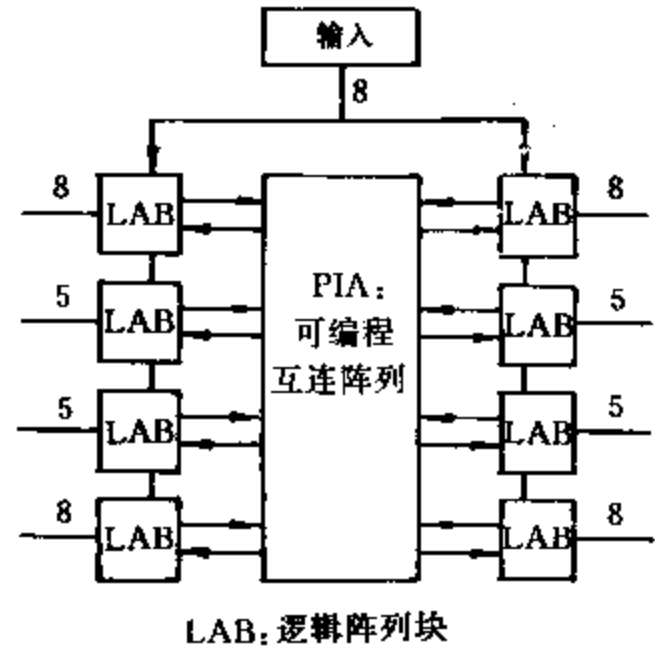


图 2.6 Altera MAX EPM5128 的方框图

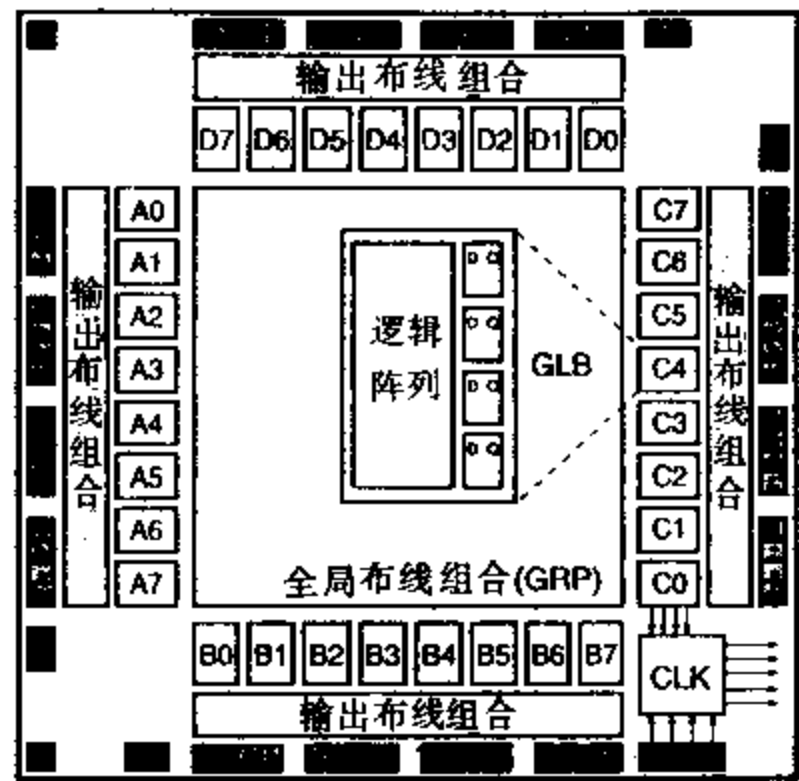


图 2.7 Lattice ispLSI 系列的方框图

因此,复杂可编程逻辑器件 CPLD 基本上是扩充原始的可编程逻辑器件 PAL,它常常由可编程逻辑的功能块围绕一个位于中心和延时固定的可编程互连矩阵构成,由于用固定长度的金属线实现逻辑单元之间的互连,而可编程逻辑单元又是类似 PAL 的与阵列,使得复杂可编程逻辑器件与 FPGA 相比很容易计算输入到输出的传播延时,但也限制了它的一些灵活性。CPLD 的设计也较 FPGA 简单。

虽然,CPLD 的结构看起来都很相似,而且一般都认为 CPLD 有100%的布通率,但是由于可编程互连矩阵的结构不同,实际上是有差别的。例如,Lattice 的全局布线组合(GRP)是一个多路开关的阵列,造成信号线互相连接有一定的限制,所以利用率不能达到100%。Altera 的 CPLD 在布通率上也有类似的问题。Xilinx 的通用互连矩阵(UIM)是纵横开关(Crossbar Switch)的矩阵,可以保证100%的布通率,只要逻辑电路放得进 Xilinx 的 CPLD,便一定能布得通,逻辑电路一定能实现,也就是说从器件输入或从一个功能块输出的一条信号线一定可以去到另外任一个功能块。Xilinx 的 UIM 除了可以连接不同的逻辑电路,还能在不增加任何额外逻辑资源的情况下实现线与功能,其开发系统软件会充分利用它的 UIM 资源,称为 Smart Switch,UIM 实现线与不增加延时,保证了设计的速度不会随设计复杂性的增加而下降。

第三章 可编程门阵列结构

§ 3.1 概 述

现场可编程门阵列(FPGA)是近几年加入到用户可编程电路行列的器件,可编程门阵列被设计成在器件的选择和内连上提供了更大的自由度。FPGA 的结构类似于掩膜可编程门阵列(MPGA),由逻辑功能块排列成阵列组成,并由可编程的内部连线连接这些逻辑功能块来实现不同的设计。FPGA 和 MPGA 的主要差别是 MPGA 利用集成电路制造过程进行编程来形成金属互连,而 FPGA 是编程电子开关实现逻辑功能和互连,类似于传统的可编程逻辑器件是由用户进行编程。

FPGA 可以达到比 PLD 更高的集成度,但有更复杂的布线结构和逻辑实现。PLD 与 FPGA 之间的主要差别是 PLD 通过修改具有固定内连电路的逻辑功能来编程的,而 FPGA 是通过修改一根或多根内连线的布线来编程的,内连线分隔了称为宏单元的固定功能块。所以 FPGA 不是建立在前面提到的可编程逻辑器件的结构上,而是在用户可编程的特性和它们的快速设计及诊断能力上类似于可编程逻辑器件。对于快速周转的样机这些特性使得 FPGA 是极其通常的选择,而且 FPGA 比 PLD 更适合于实现多级的逻辑功能。

通常 FPGA 由布线资源围绕的可编程逻辑单元(或宏单元)构成阵列,又由可编程 I/O 单元围绕阵列构成整个芯片,如图 3.1 所示,排成阵列的逻辑单元由布线通道中的可编程内连线连结起来实现一定的逻辑功能。一个 FPGA 可能包含静态存储单元,它们允许内连的模式在器件被制造以后再被加载和修改。

FPGA 是由掩膜可编程门阵列和可编程逻辑器件二者演变而来的,将它们的特性结合在一起,因此 FPGA 既有门阵列的高逻辑密度和通用性,又有可编程逻辑器件的用户可编程特性,对于设计 ASIC 来说,采用 FPGA 在实现小型化,集成化和高可靠性的同时,减少了风险,降低了成本,缩短了周期。

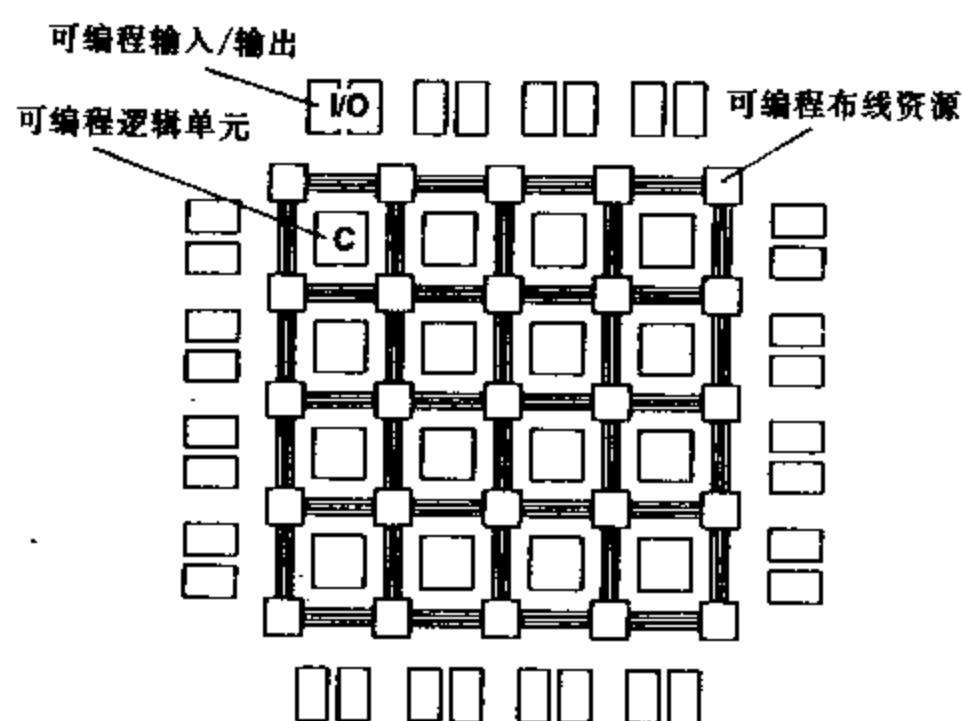


图3.1 FPGA 的基本结构

FPGA 的逻辑功能块在它们的规模和实现逻辑功能的能力上存在很大的差别,有的逻辑功能块规模十分小,仅包含只能实现倒相器的两个晶体管,有的逻辑功能块则规模比较大,是可以实现任何五输入逻辑函数的查找表结构,由于这些差别,可以按它们的粒度来划分逻辑功能块,但是粒度可以有各种方式来定义,使得划分的形式多种多样,例如按功能块可实现的布尔函数的数目,等效两输入与非门的数目,晶体管的总数,总的归一化面积或输入和输出的数目等。但是在一些结构中,逻辑和布线是紧密有关的,很难分出其对结构的影响,所以一般将逻辑功能块分为细粒度(fine-grain)和粗粒度(coarse-grain)两类。大多数细粒度逻辑块是与半定制门阵列的基本单元相同,由可以用可编程互连来连接的少数晶体管组成,所以细粒度逻辑块的规模一般都较小,它的主要优点是可用的功能块可以完全被利用,缺点是采用它通常要求大量的连线线段和可编程开关,使相对的速度变慢。而粗粒度功能块是规模较大和功能较强,其优越的性能使它们得到越来越广泛的应用。近年来,由于工艺的不断改进,芯片集成度不断提高,加上硬件描述语言(HDL)的设计方法得到广泛应用,而大多数逻辑综合工具是针对门阵列的结构开发的,因此为适应这方面的需要,许多厂家开发出有更高集成度的细粒度结构的 FPGA,如 Xilinx 公司的采用 MicroVia 技术的一次编程反熔丝结构的 XC8100 系列和 GateField 公司的采用 Flash-EPROM 控制开关元件的可再编程 GF100K 系列等,这些细粒度 FPGA 的逻辑功能块规模相对都较小。

作为细粒度 FPGA 结构的 XC8100 系列引入了最新的 MicroVia 工艺,图 3.2 为该系列的门海结构示意图, MicroVia 的结构在图 4.4 中给出,此工艺在制造过程中组合了三种元件:

1. 三层金属的工艺
2. 金属到金属的反熔丝
3. 细粒度门海(Sea-of-gate)结构

因为在 FPGA 中布线和可编程元件可以构造在逻辑上面的较高金属层内,这种结构可使 FPGA 将延伸的布线,小的管芯尺寸和高门利用率结合起来,因此 MicroVia 有潜力极大地降低 FPGA 成本,正因为其尺寸十分小,几乎不可能被观察,而称为“微型通孔”。

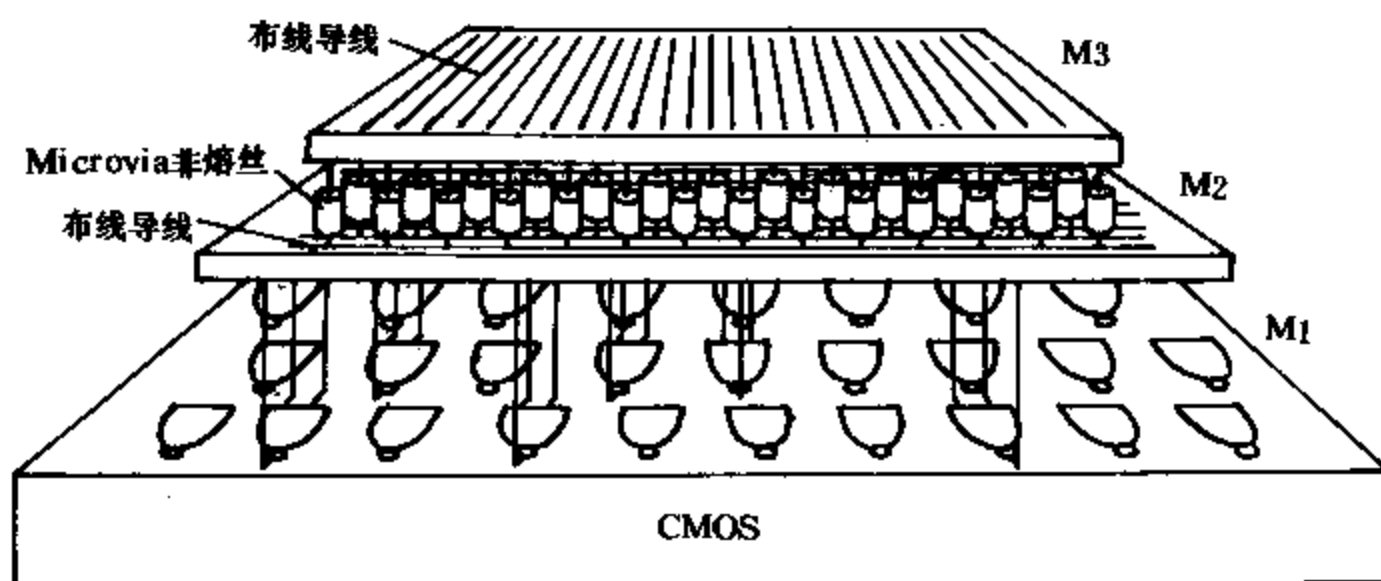


图 3.2 Xilinx 的门海 FPGA 技术

这一章下面的部分将着重讨论粗粒度结构 FPGA,从组成它的可编程逻辑块和可编程互连资源来看,主要有两类逻辑块的构造:

1. 查找表类型
2. 多路开关类型

因而有两种 FPGA 的结构:

1. 具有可编程内连线的通道型门阵列,采用分段互连线,利用不同长度的多种金属线经传输管将各种逻辑单元连接起来,布线延时是累加的、可变的和与通道有关的。

2. 具有类似 PLD 可编程逻辑块阵列的固定内连布线,采用连续互连线,利用相同长度的金属线实现逻辑单元之间的互连,布线延时是固定的和可预测的。

Xilinx 和 Actel 的 FPGA 属于具有分段式可编程内连线的通道型门阵列,两者的逻辑块构造,Xilinx 的属于查找表类型,而 Actel 的属于多路开关类型;Altera 的 FPGA 是由传统的 PLD 结构演变而来,因此属于具有类似 PLD 的可编程逻辑块阵列和连续互连布线这一类,逻辑块是基于与或门电路构成的。下面对这三种 FPGA 的结构分别作一介绍。

§ 3.2 查找表型 FPGA 结构

查找表型 FPGA 的可编程逻辑单元是由功能为查找表的静态存储器(SRAM)构成函数发生器,由它来控制执行 FPGA 应用函数的逻辑。 M 个输入的逻辑函数真值表存储在一个 $2^M \times 1$ 的 SRAM 中,SRAM 的地址线起输入的作用,SRAM 的输出为逻辑函数的值,由此输出状态控制传输门或多路开关信号的通断,实现与其它功能块的可编程连接。

查找表结构的优点是其呈现高的功能性, M 输入的查找表可以实现高达 M 个输入的任意函数,只要用相应于函数真值表的位模式预先加载存储器,这样的函数有 2^{2^M} 个,带来的问题是多于五个输入的查找表变得不能接收,因为 M 个输入查找表的存储单元数是 2^M ,而可实现的函数数目增加得太多,这些附加的函数在逻辑设计中通常用不到,也很难让逻辑综合工具去开发利用。目前有如下六种采用查找表结构的 FPGA 器件:

1. Xilinx 的 XC2000 系列:其逻辑块由四输入和一输出的查找表组成,可以生成任何四输入变量的逻辑函数。(参考图 6.20)

2. Xilinx 的 XC3000 系列和 AT&T 的 ATT3000 系列:它们逻辑块由五输入和一输出的查找表组成,且可以配置成两个四输入的查找表,限定在一起使用不超过五个输入的总数,增加这个灵活性,可使逻辑单元更好地利用,因为许多逻辑函数并不需要五个输入变量那样多。(参考图 6.12)

3. Xilinx 的 XC4000 系列:它采用两个四输入查找表,再接到一个三输入查找表的多成份构造来构成逻辑单元的,两种查找表之间采用非可编程的固定连接,尽管这样不具有灵活性,但由于不存在改变查找表型式的可编程连接而使速度显著加快。(参考图 6.1)

4. Xilinx 的 XC5000 系列:有四个四输入的查找表形成四个独立的逻辑单元,一对逻辑单元可组合成五输入的函数发生器。(参考图 6.23)

5. AT&T 的 ORCA 系列:它的可编程功能单元(PFU)由四个四输入查找表型式的函数发生器分成上下两对,每对函数发生器共享它们的三个输入,可以实现五输入的任何函数,PFU 的可编程控制逻辑可用来转换上或下函数发生器的输出,或与所有四个函数发生器相连实现六输入变量的函数。如图 3.3。

6. ALTERA 的 FLEX8000 系列:FLEX 的逻辑单元(LE)由单个四输入函数发生器和一

个寄存器组成,而逻辑阵列块(LAB)由八个逻辑单元,与其有关的进位及级联链,LAB 控制信号和 LAB 局部互连组成,如图3.4。

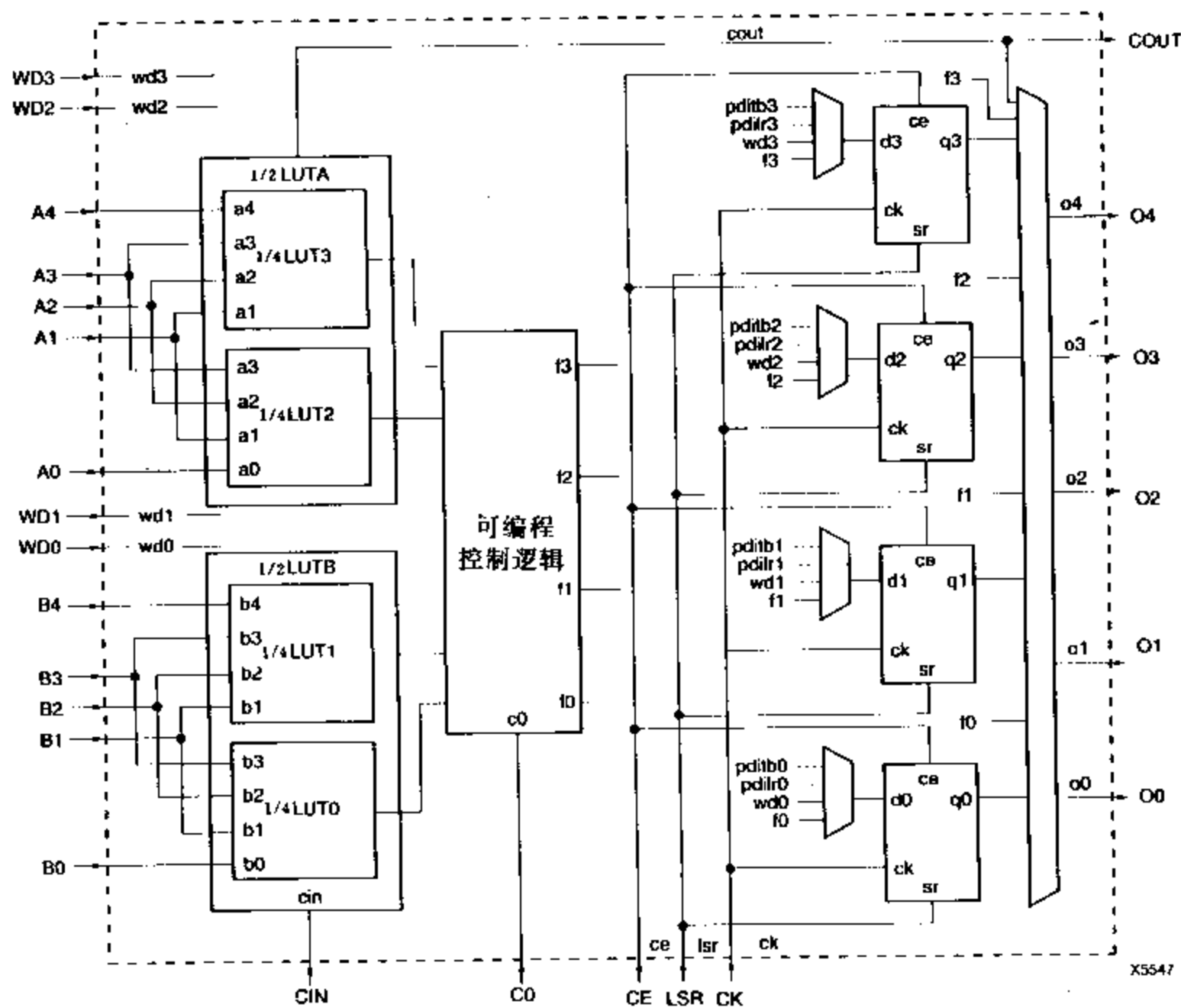


图3.3 ORCA 可编程功能块(PFU)方框图

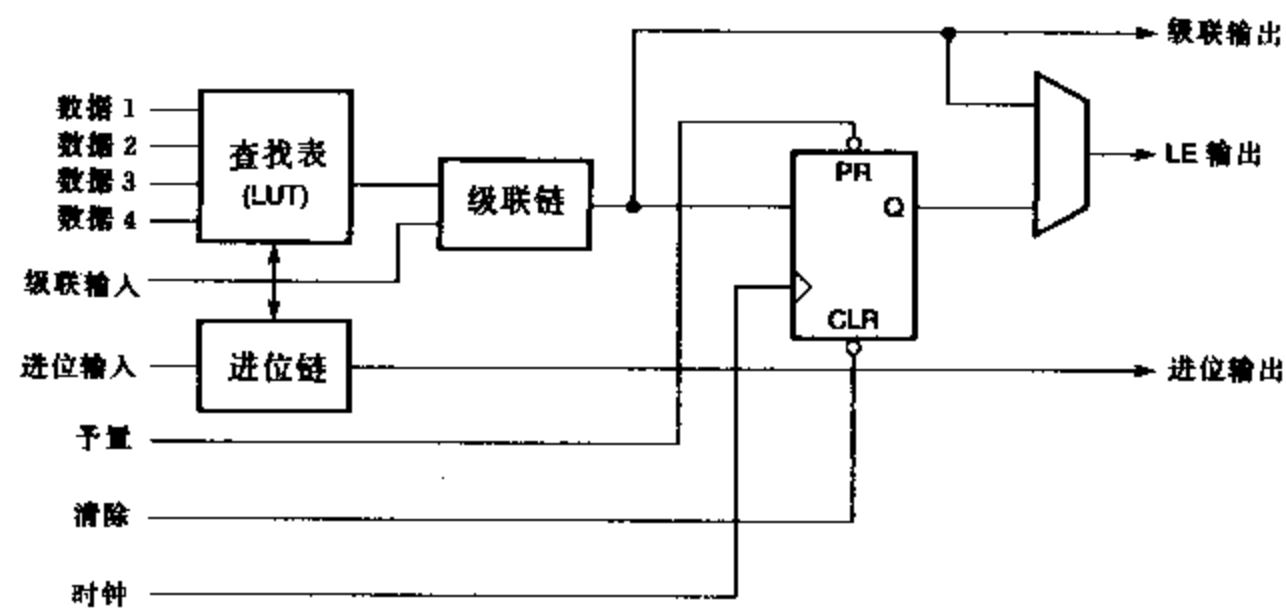


图3.4 FLEX 逻辑单元(LE)的方框图

比较上述六种结构可以看到 XC4000系列的 CLB 每个函数发生器有独立的输入,允许完全独立的函数在相同的功能块内实现,因此与其它几种功能块相比,它提供了逻辑功能块输入和输出更灵活和更有效的方案,最高可实现九个输入变量的任何函数。

图3.5表示了一个基本的 Xilinx 查找表宏单元如何实现逻辑函数的原理,两个 16×1 位的存储器表明它有五个通用的输入, $M=5$,可存储最多为五变量逻辑函数的真值表的 2^5 个可能的函数值,逻辑功能和内连关系是由一个“配置程序”来决定,此配置程序是存储在内部的静态存储器单元内,由于静态存储器中的配置程序在掉电时将丢失,因此在芯片每次加电时要进行一次配置程序的加载。

在查找表型结构中,变量数目是要求来实现的逻辑块数目的标志,它与函数中的文字数没有关系,例如, $Y_1=ABCDEF$ 有六个文字,它要求两个五输入的逻辑块,而 $Y_2=ABC+BD \sim E+\sim ACE+\sim CDE$ 有十二个文字,仅须要一个逻辑块,因为它仅仅由五个变量组成。

可以定义一个可行函数作为一个变量不比通用输入数 M 更多的任意函数,一个可行网络,它的每个中间结点实现一个可行函数,最好的 FPGA 之解是具有最小结点数目的一个可行布尔网络。所希望的解是减少要求的查找表数目的解,要获得最好地利用查找表的问题等效于数学上的装料归并(bin-packing)的问题。

图3.5中说明了用查找表结构实现一位全加器的方法,当输入信号为 A_0, B_0 和进位输入 C_1 时,全加器输出 S_0 和 C_0 的逻辑方程为:

$$S_0 = A_0 \oplus B_0 \oplus C_1$$

$$C_0 = (A_0 \oplus B_0)C_1 + A_0B_0 = A_0C_1 + B_0C_1 + A_0B_0$$

表3.1列出一位全加器输入和输出的真值表,在查找表中存储的是全加器真值表的输出数值,而输入变量相应为查找表的地址,由于有两个输出变量,对于 Xilinx 的 XC2000系列要采用它的如图6.22中所示的两个三变量函数的模式,四输入的 16×1 的 SRAM 分为两个三输入的 8×1 的存储器作为查找表,分别存入 S_0 和 C_0 的真值表数值。图3.5中也给出实现一位全加器的逻辑图,在 XC2000系列中恰好占用一个 CLB。对于 XC3000系列要采用它的如图6.13中所示的两个四变量输入函数的模式,原来五输入的 32×1 的 SRAM 分成两个 16×1 的存储器,每个存储器仅用一半存入真值表数值即可。所以图3.5表示的是五输入查找表实现一位全加器的原理。

表3.1 全加器输入和输出的真值表

输入			输出	
A0	B0	CI	S0	CO
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

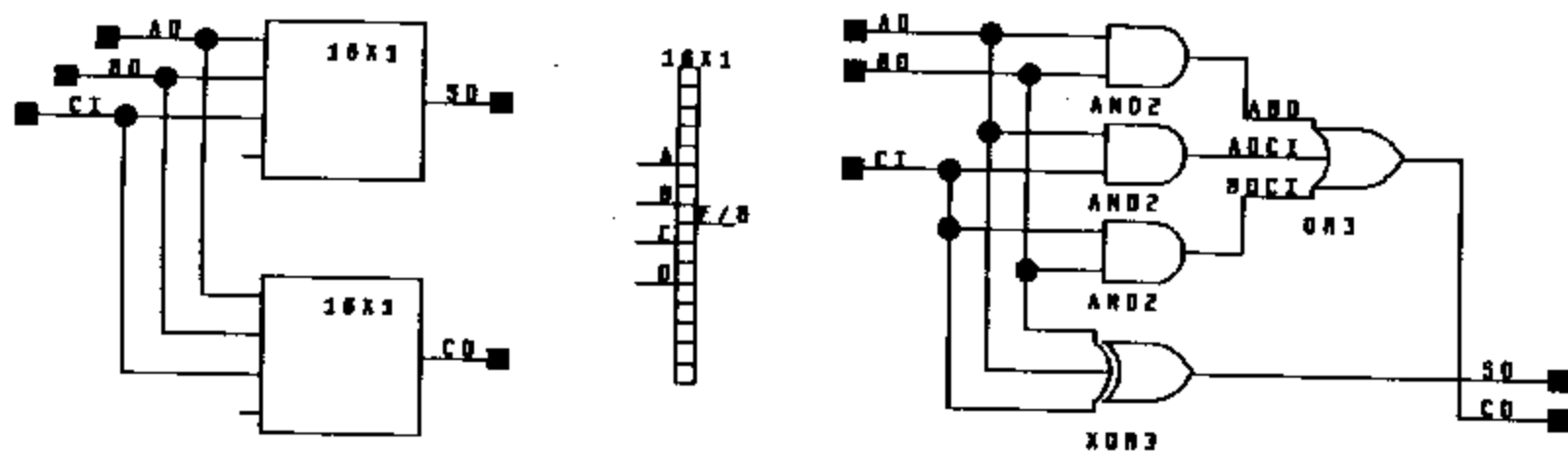


图3.5 基本的 Xilinx 查找表宏单元实现一位全加器

图3.6给出了 Xilinx FPGA 的逻辑单元阵列(LCA)的排列,它象一个门阵列通过布线通道利用编程好的内连图,把逻辑晶体管(按组称为可配置逻辑块—CLB)连接在一起综合阵列中的逻辑功能。它有三种可配置元件,核心的 CLB 是 LCA 的功能积木块,CLB 被用来构造逻辑功能和作为一个存贮单元,十分类似于 PLD 的宏单元;周围的可配置输入/输出块;所有这些 CLB 和 I/O 块利用三种互连资源连接在一起:直接连接线,通用内部连线和长线。在第六章再详细讨论 Xilinx 的 FPGA 的结构。

FPGA 在结构上还提供了如下所列的许多其它系统级的特性:

1. 片内 RAM:逻辑功能块的查找表可以用来实现片内 RAM 或 ROM,去除了对外部芯片的需要,由于去除了芯片通断的延时而提高了系统的性能。
 2. 全局时钟线:全局时钟线提供高速低扭曲的时钟,由于出现了更多的结构设计带有不同时钟要求的趋势,因此常常需要大量的全局时钟来实现复杂的设计。
 3. 专用的时钟使能:专用的时钟使能允许时钟控制不耗费有用的逻辑和布线,或者不引入附加的延时到电路中。
 4. 内部三态总线:这个特性增加面向总线设计的功能,容量和性能。线与能力在单根长线上安排多达24个输入的与函数,不需要其它的布线和逻辑资源。
 5. 宽的沿边译码器:以 XC4000系列为例,宽的沿边译码器安排多达72个输入的与函数,不耗费其它的布线和逻辑资源。
 6. 高的 I/O 驱动能力:这一能力允许器件直接接到总线,不需要外部的缓冲器,高的驱动能力增加高扇出应用的性能。
 7. 扭曲率控制:输出扭曲率控制允许设计者选择抗扰性和系统性能的最佳组合。
 8. 边缘扫描逻辑:边缘扫描逻辑增强了在板级或系统级的可测性。
- 在表3.2中,按照这些条件对列出的查找表型 FPGA 器件进行了比较。

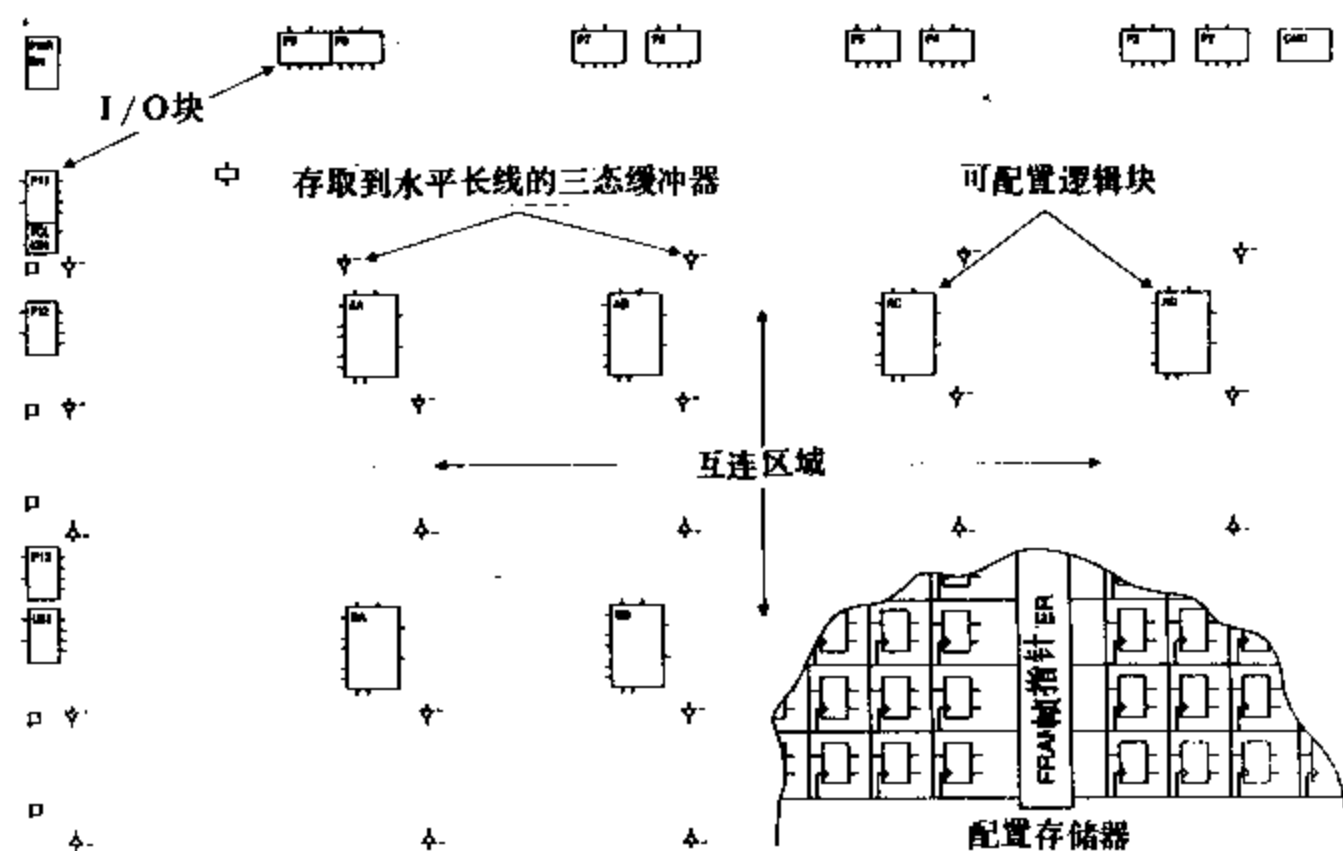


图3.6 Xilinx FPGA 的 LCA

表3.2 查找表型 FPGA 系统性能比较

特性	XILINX—XC				AT&T—ORCA		ALTERA
	2000	3000	4000	5000	1C	2C	8000
触发器时钟使能	无	有	有	有	有	有	无
专用进位逻辑	无	无	有	有	有	有	无
I/O 块寄存器	1	2	2	0	0	0	1
输出扭曲率选择	无	2	2/4		3	2	2
全局时钟缓冲器	2	2	8		0	0	2—4
全局复位	有	有	有	有	有	有	有
沿边宽译码器	无	无	有	无	无	无	无
内部三态总线	无	有	有	有	有	有	无
系统内可编程性	有	有	有	有	有	有	有
片内存储器	无	无	有	无	有	有	无
支持 JTAG	无	无	有	有	有	有	无
掉电模式	有	有	无		有	有	无
主机三态控制	无	无	有		有	有	无
输入上拉电阻	无	有	有		有	有	无
I/O 驱动(mA)	4	4/8	12/24		12	12	8

§ 3.3 多路开关型 FPGA 结构

在多路开关型结构中,基本的积木块是一个多路开关的配置。利用多路开关的特性,在多路开关的每个输入接到固定电平或输入信号时,可实现不同的逻辑功能,例如,具有选择输入 s 和输入 a 和 b 的二到一多路开关,输出为 $f=sa+\bar{s}b$,当置 b 为逻辑零时,多路开关实现与的功能; $f=sa$,当置 a 为逻辑1时,多路开关实现或的功能; $f=s+b$ 。大量的多路开关和逻辑门连接起来,可以构成实现大量函数的逻辑块。

Actel 的 FPGA ACT-1由三个两输入多路开关和一个或门组成的基本积木块,如图3.7所示,这个宏单元总共有八个输入和一个输出,可以实现的函数为:

$$f = (\overline{s_3 + s_4}) (\overline{s_1} w + s_1 x) + (s_3 + s_4) (\overline{s_2} y + s_2 z)$$

当设置每个变量为一个输入信号或一个固定电平时,可以实现702种逻辑函数,例如,当设置为: $w=A_0, x=\bar{A}_0, s_1=B_0, y=\bar{A}_0, z=A_0, s_2=B_0, s_3=C_1, s_4=0$ 时,可实现全加器输出 S_0 的逻辑函数:

$$\begin{aligned} S_0 &= (A_0 \oplus B_0) \oplus C_1 \\ &= (\overline{C_1 + 0}) (\overline{B_0} A_0 + B_0 \bar{A}_0) + (C_1 + 0) (\overline{B_0} \bar{A}_0 + B_0 A_0) \end{aligned}$$

当设置为: $w=0, x=C_1, s_1=B_0, y=C_1, z=1, s_2=B_0, s_3=A_0, s_4=0$ 时,可实现全加器输出 C_0 的逻辑函数:

$$\begin{aligned} C_0 &= (\overline{A_0 + 0}) (\overline{B_0} 0 + B_0 C_1) + (A_0 + 0) (\overline{B_0} C_1 + B_0 1) \\ &= \bar{A}_0 B_0 C_1 + A_0 \bar{B}_0 C_1 + A_0 B_0 \\ &= B_0 C_1 + A_0 C_1 + A_0 B_0 \end{aligned}$$

ACT-2逻辑块类似于 ACT-1,只是在第一行分开的多路开关被联起来接到一个两输入的与门,如图3.8所示,ACT-2的组合逻辑模块执行4到1线的多路开关作用,可实现766种函数。

Actel 的 FPGA 中,每行逻辑块由组成布线线路的布线通道和时钟分布网线分隔开。

给定一个多路开关型结构,必须选择一组2到1线的多路开关作为基本的函数,然后再相应地对它们进行编程,相同的函数可用不同的形式来实现,取决于输入选择控制和输入数据的选择。

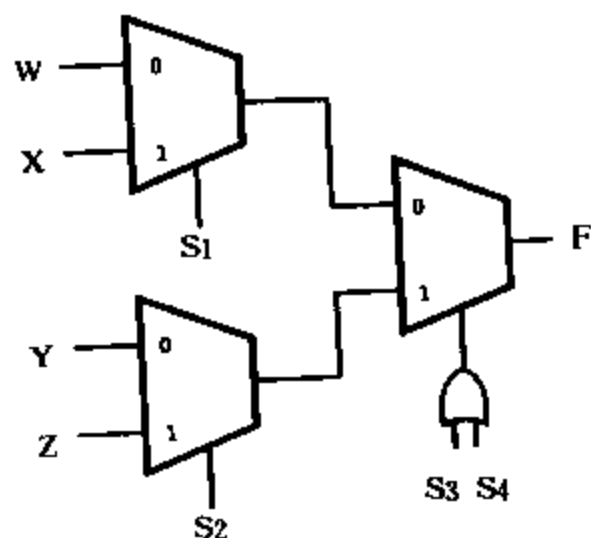


图3.7 基本的 Actel 多路开关型逻辑块

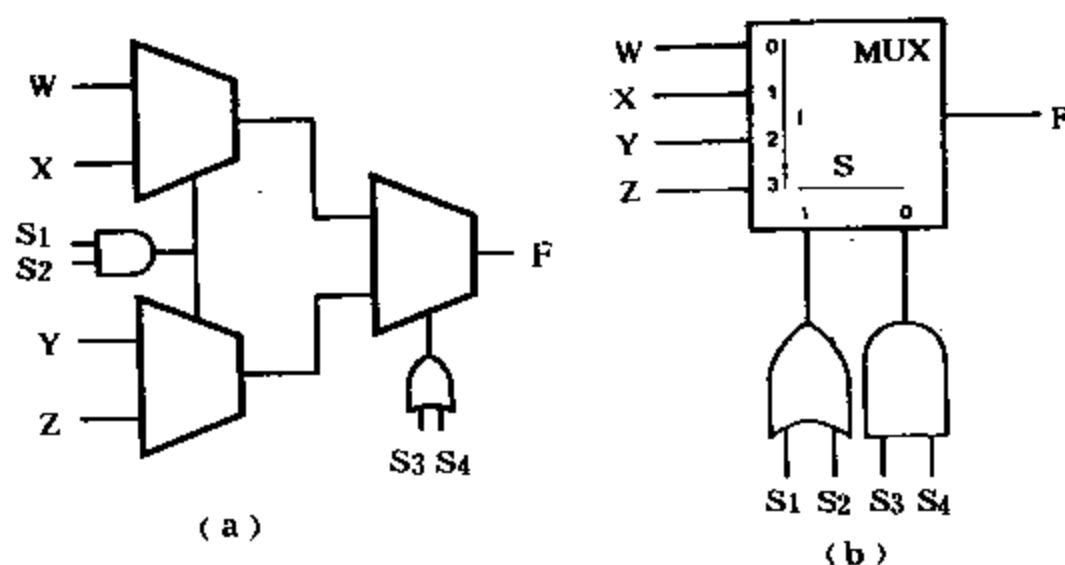


图3.8 第二类 Actel 多路开关型逻辑块

§ 3.4 多级与非门 FPGA 结构

Altera 的 FPGA 是从传统 PLD 的基于 PLA 的结构演变而来的,它的逻辑块是由从20到超过100个输入的宽扇入的与门馈送到带有3到8个输入的或门组成的。

Altera 的与非门结构是基于可以实现与—或逻辑的与—或电路,其输出馈送到一个异或门,如图3.9(a)所示,这个基本电路可以用一个触发器和一个多路开关来扩充,如图3.9(b)所示,此多路开关可选择锁存的输出信号,或者非锁存的输出信号。异或门可以用来获得可编程的非逻辑。如果一个异或门的输入端是分离的,它的作用是和或门相同,这允许或门和异或门形成更大的或函数,用来实现其它的算术功能,这个可编程的形式显著增加了逻辑功能块的功能性。

Altera FPGA 的功能块具有的优点是宽与门可以用较少的功能块来形成逻辑函数,但是很难做到有效地利用所有门的所有输入,导致芯片密度上的损失,但这个损失不象一开始那样严重,因为线与门有高的封装密度,以及逻辑连接也起布线功能的作用,而其它结构中布线和逻辑是分开的,未用的输入会导致同样的损失。

线与配置的缺点是利用上拉器件而消耗静态功率,充满这些上拉器件的阵列就会消耗很大的功率,为了减少这种功耗,MAX7000系列中每个门可以被编程,达到小于60%的功耗,但增加约40%的延时,在非临界通道中可利用这种办法来减少功耗。MAX7000的逻辑块与 MAX5000的类似,除了多提供两个乘积项,以及相邻块可以互相“借”乘积项而变得更灵活,这是在与门和或门之间利用小的布线结构,称为乘积项选择矩阵,来完成的。

用 MX5000系列的逻辑单元实现一位全加器的原理图如图3.10所示。

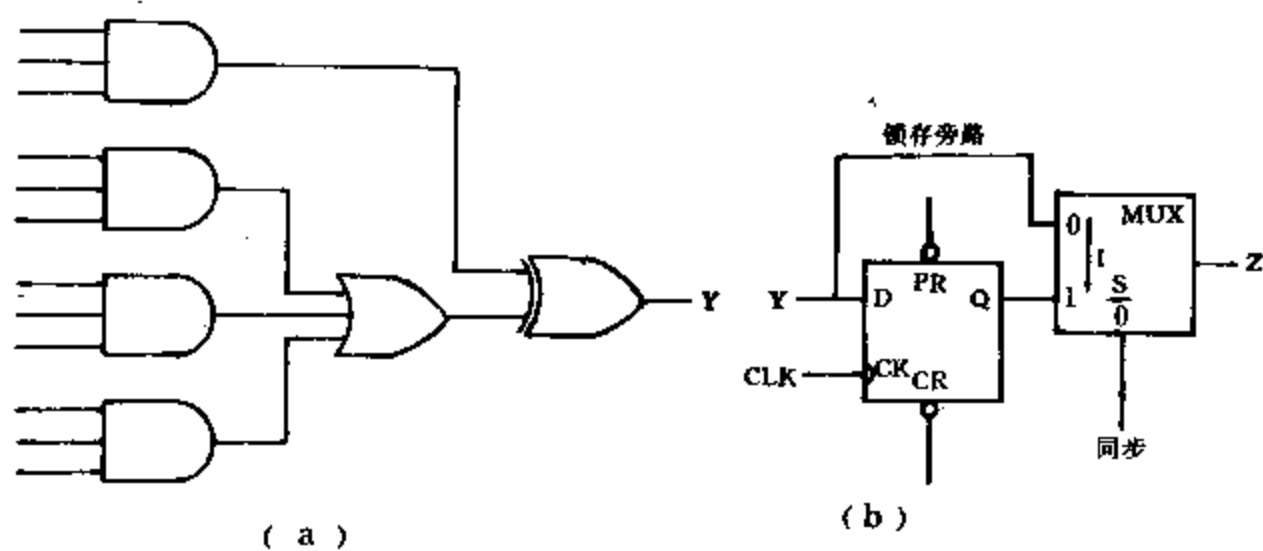


图3.8 (a)与或逻辑块 (b)同步/非同步输出电路

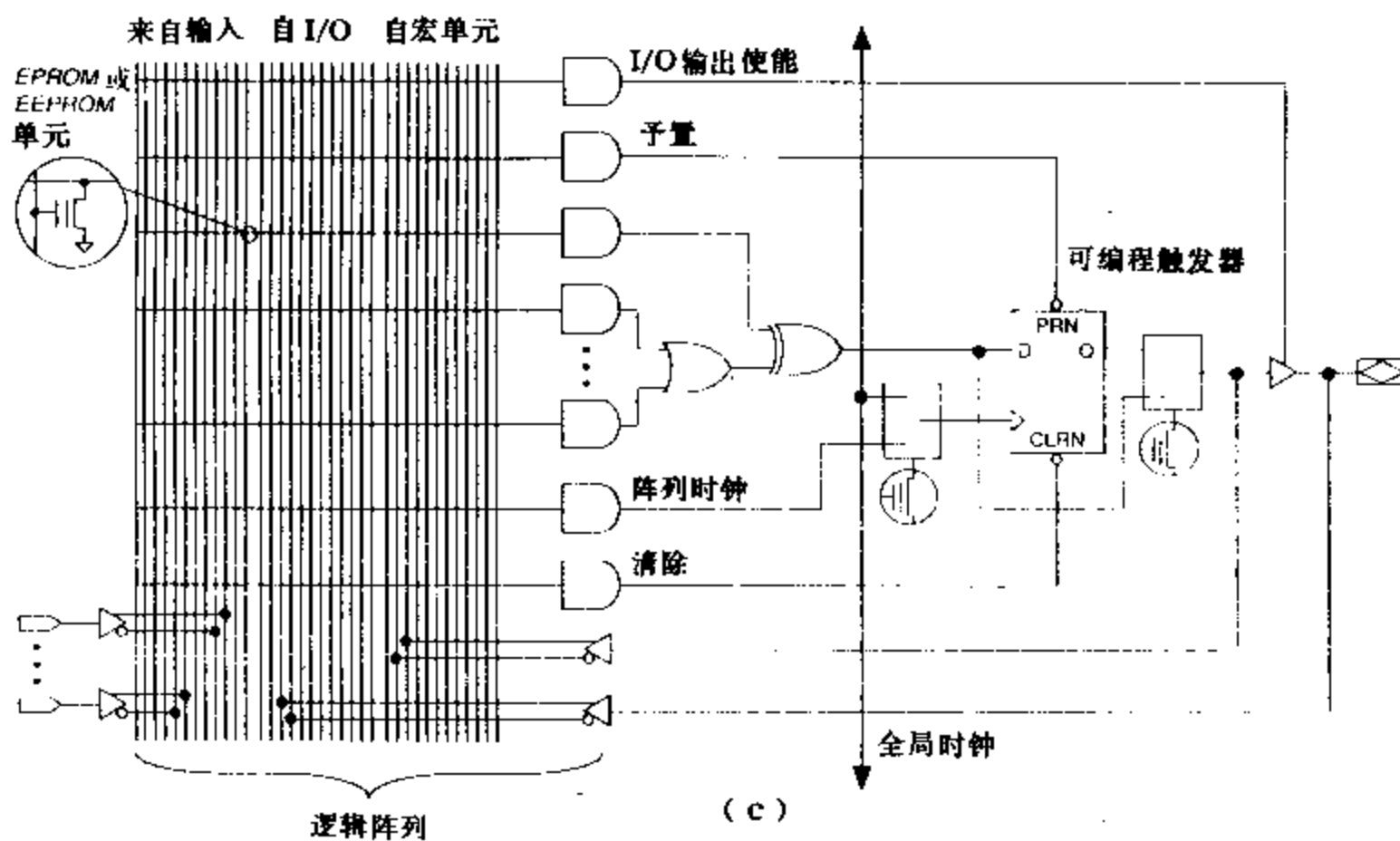


图3.9 (c)ALTERA 的 MAX7000系列逻辑单元

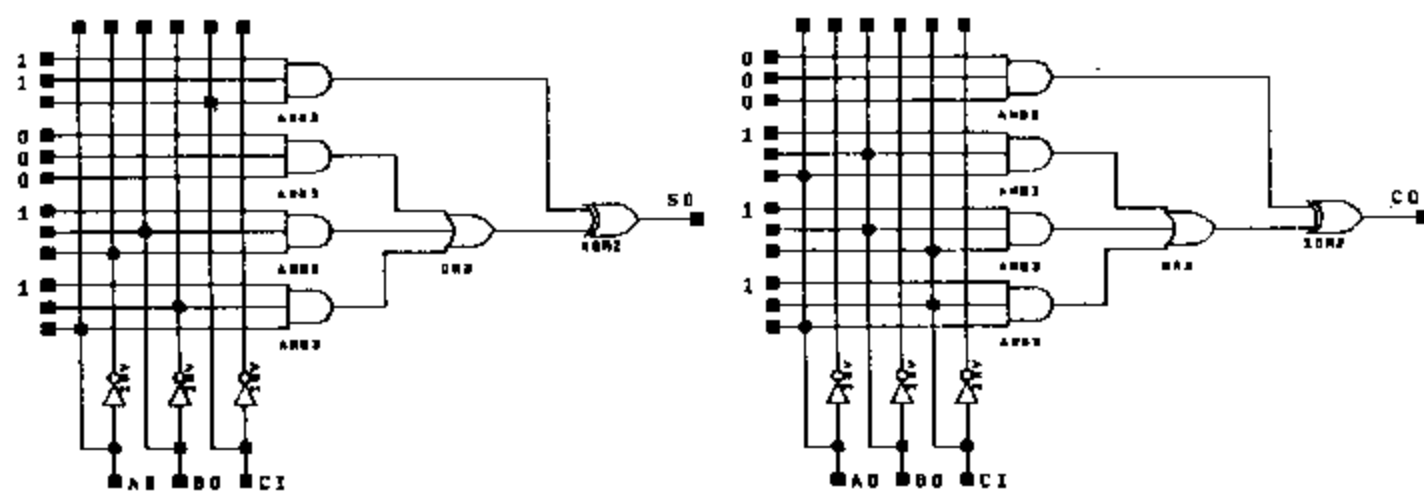


图3.10 MX5000系列的逻辑单元实现一位全加器的原理图

作为本章的结束,最后就上面介绍的几类 FPGA 列表做一比较,见表3.3。

表3.3 几种 FPGA 的性能比较(该系列最高指标)

ACTEL	ACT1	反熔丝,3个2-1多路开关,1个2输入逻辑门,702逻辑函数,2000门
	ACT2	反熔丝,3个2-1多路开关,2个2输入逻辑门,766逻辑函数,8000门
	A1200XL	反熔丝,2500-8000门,0.6 μ m 双层金属工艺,382-998触发器,140PIN
	A1280XL	8000门,150MHz 速率触发器998个,140I/O 线,10,12ns 速度档次
	ACT3	反熔丝,1500-10000门,0.6 μ m 双层金属工艺,312-1493触发器,228PIN
	A14100A-2	10000门,200MHz 触发器1493个,228I/O 线,Clock-to-out 为9.5ns,Chip-to-Chip 为90MHz
ALTERA	A3200DX	反熔丝,6500-40000门,0.6 μ m 双层金属工艺,片内 RAM2048-3584bit,292PIN
	A3265DX	510时序模块,475与和或控制四输入组合模块,20快速七输入译码模块,120I/O 模块,无片内 RAM
	MAX5000	积之和,EPROM,最多3750可用门,100PIN
	MAX7000	积之和,EPROM,EEPROM,5000可用门,208PIN
	MAX9000	积之和,EEPROM,6000-12000可用门,系统内可编程,3.3/5V I/O 线,80MHz 系统频率
	FLEX8000	查找表,SRAM,24000可用门,2252触发器304PIN
XILINX	EPX8160	FLASHLogic,配置元件 FLASH&SRAM,160宏单元,20480位最大片内 RAM,208PIN PQFP
	FLEX10K	6-26Kbit 片内 RAM
	EPF10K100	100000门,24567位 RAM,5392触发器,503PIN PGA BGA
	XC2000	SRAM,100四输入 CLB,174触发器,74 I/O
	XC30/3100	SRAM,484五输入 CLB,1320触发器,176 I/O,340Mc 翻转频率
	XC4000	SRAM,1024九输入 CLB,2560触发器,32768位 RAM,256I/O
	XC5200	低成本高容量 FPGA 0.6 μ m 三层金属,多用功能块,多用环
	XC5215	14000-18000门,22 $\times$ 22多用功能块阵列,484CLB,1936触发器,244I/O,每根水平长线24TBUF
	XC6200	可配置协处理器,8,16,32位操作的主处理器存储器接口,高速配置,典型数据总线速率传递数据
	XC8100	MicroVia 的三层金属门海结构金属-金属反熔丝,接通电阻<50 Ω ,10000门
	XC8109	8700门,2688可配置逻辑单元,1344触发器,208I/O

第四章 可编程 ASIC 的编程元件

可编程逻辑有许多不同的编辑技术是可用的。

易失元件:大多数 FPGA 为了灵活性和为了现代化的 IC 工艺兼容是用 SRAM 存贮。但是掉电后信息要丢失。此外,SRAM 方法比任何其它的编程方法有更快的可编程和可再编程的特性。

非易失元件:许多 PLD 是经过 EPROM 元件来编程。EPROM 可以取较小的电路面积,但是要用紫外光擦除。

一次编程元件:PROM 元件是由金属或多晶硅链路组成,靠高电流通过它们而被破坏,反熔丝开关也是一次编程元件。本章对以下几种编程技术进行讨论:

1. 熔丝型开关
2. 反熔丝型开关
3. 浮栅编程技术
4. 基于 SRAM 的编程元件

§ 4.1 熔丝型开关

PLD 最初的可编程开关是熔丝链路,与和它们互连的晶体管相比,熔丝是大的结构,虽然在每个可能的互连点上都需要熔丝,但仅仅小部份可能需要进行连接,所以必须烧断大部份的熔丝。

在一个 PLD 中与阵列及或阵列是熔丝连接的矩阵,当在乘积线上两个信号的熔丝被保留,而同一线上所有其它的熔丝被烧断时,这两个输入信号被组合,留在器件内的不烧断熔丝的模式产生相应的逻辑功能,如图 4.1 说明 PLD 中熔丝链路编程的概念。

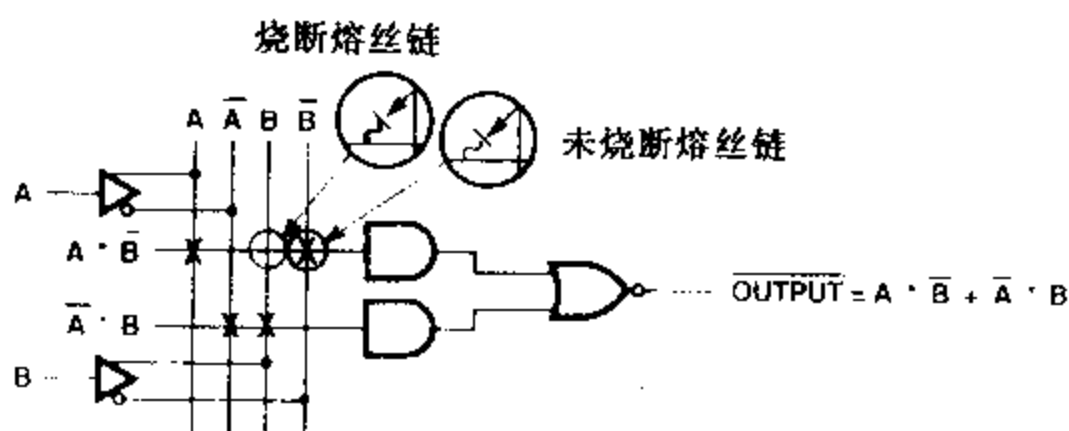


图 4.1 熔丝开关型

熔丝型开关很难测试可靠性。器件可能被编程,但是以后将会发现十分小的百分率的熔丝没有烧断,而一些不应该烧断的却烧断了。而且熔丝烧断就造成电路永远开路。

§ 4.2 反熔丝开关

熔丝链路作为可编程元件带来的问题是占的面积大,要求编程电流大,且熔丝链路属于三端器件。

可编程低阻电路元件(PLICE)是一个基于反熔丝的介质,它能提供比其它编程元件尺寸和性能上显著的改善。PLICE 反熔丝是在 n^+ 扩散和多晶硅之间的介质,是和 CMOS 以及其它如双极型, BiMOS 等工艺相兼容的。PLICE 元件在标准的12层掩膜的双金属孪生槽 CMOS 工艺中被集成。要求三个附加的掩膜: n^+ 反熔丝扩散,反熔丝多晶硅,和40nm 氧化物掩膜形成编程的高压晶体管,使掩膜总数达到15层。如图4.2,一薄层氧化物加热生长到 N^+ 的表面,接着低压化学气相沉积氮化物,再重新氧化顶层的氧化物,利用这个 ONO 结构紧缩电阻的分布,与简单氧化物反熔丝相比,也改善了产量和可靠性。

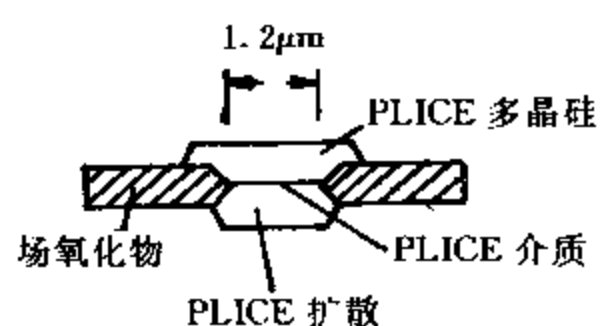


图4.2 反熔丝开关型

利用了四种晶体管类型:包括 N 型和 P 型的低压高速晶体管用于逻辑和信号通路,而高压晶体管用于编程路径。

在一个2000门的可编程门阵列器件中利用了186000PLICE 反熔丝达到每个逻辑门约100个反熔丝以达到高的互连性及门利用率。每个反熔丝占有等效于一个接触或通孔的面积。这个在两层导体之间夹着的介质,在电压加到此元件上时,介质击穿,而把两层导电材料连在一起。

表4.1 反熔丝工艺和性能

	晶体管	氧化物	L_{eff}	电压
工艺: 孪生槽双金属 CMOS 15层掩膜	N 低压	25nm	1.1μm	5V
	5V	ρ 低压	25nm	1.2μm
	N 高压	40nm	1.5μm	20V
	P 高压	40nm	1.8μm	20V
性能	编程电压:18		编程时间<1ms	
	编程电流<10mA			
	接通电阻 <1KΩ		断开电阻>100MΩ	

PLICE 反熔丝是一个两端,非丢失性一次可编程器件。在未编程状态,通常呈现十分高的阻抗($>100\text{M}\Omega$)。当18V 的编程电压加其上时,建立一个双向的低电阻,编程电流小于10mA。

除了 Actel 的反熔丝开关采用在 N^+ 扩散和多晶硅之间的多层氧—氮—氧(ONO)介质之外,也可以采用在多晶硅和第一层金属之间或金属层之间的非晶硅形成反熔丝开关。

Cypress 公司的 pASIC380 系列采用称为 ViaLink 的元件作为开关元件,ViaLink 元件是在标准的 CMOS 工艺的两层金属之间形成通孔的反熔丝。编程后产生的直接金属—金属链接达到低于 50Ω 电阻值的连接,它比 EPROM 或 SRAM 编程元件的电阻小5%,比介质反熔丝的电阻低10%。本章最后的表4.2给出了有关的比较。

另外,前面提到的 MicroVia 工艺与 Actel 的反熔丝结构也不一样,它是金属到金属的反熔丝,在第二层和第三层金属之间,逻辑电路之上,构造类似通孔,为不占管芯尺寸的可编程布线,它不仅是一个“反熔丝”,而且把三层金属工艺和门海结构等结合在一起,形成真正的门海 FPGA。图4.4给出普通反熔丝与金属—金属反熔丝的比较,可以比较清楚地看出二者的差别。

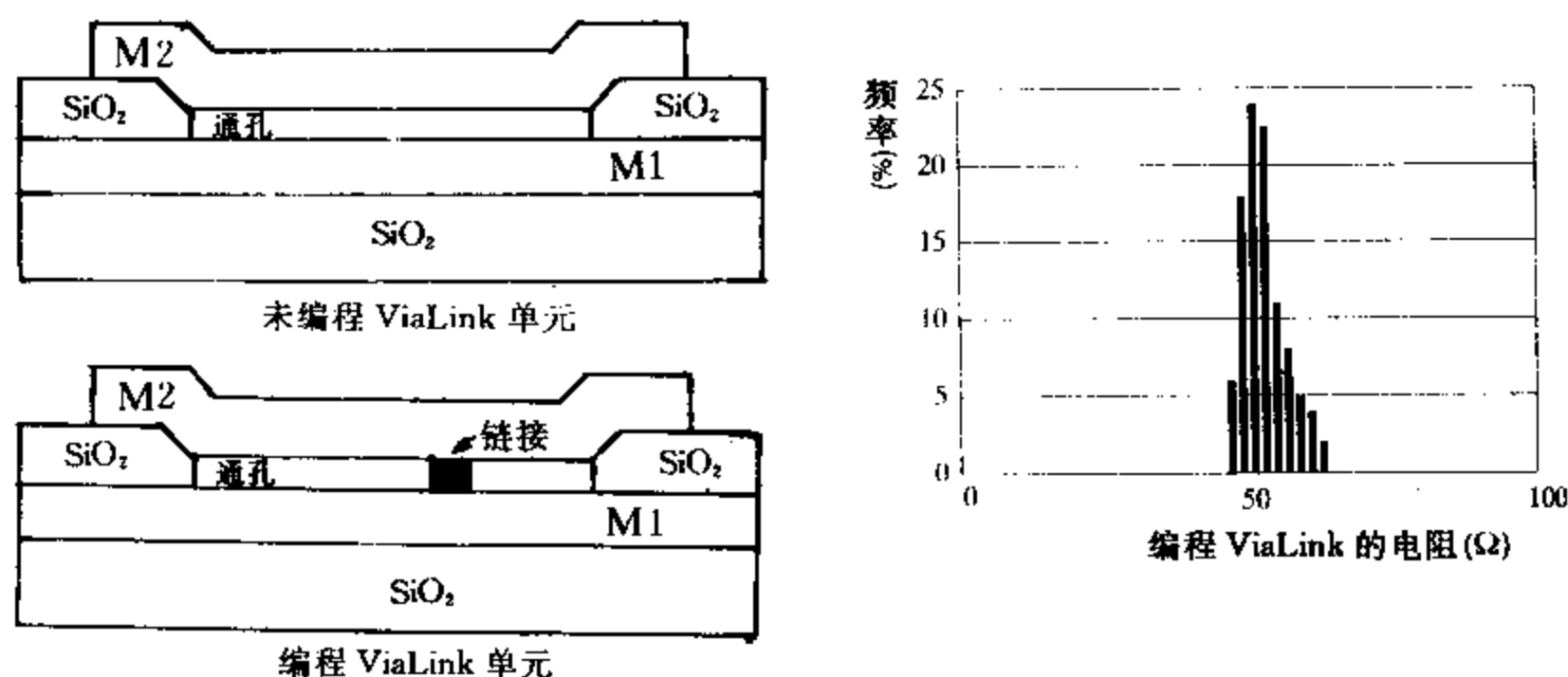


图4.3 金属—金属反熔丝 ViaLink

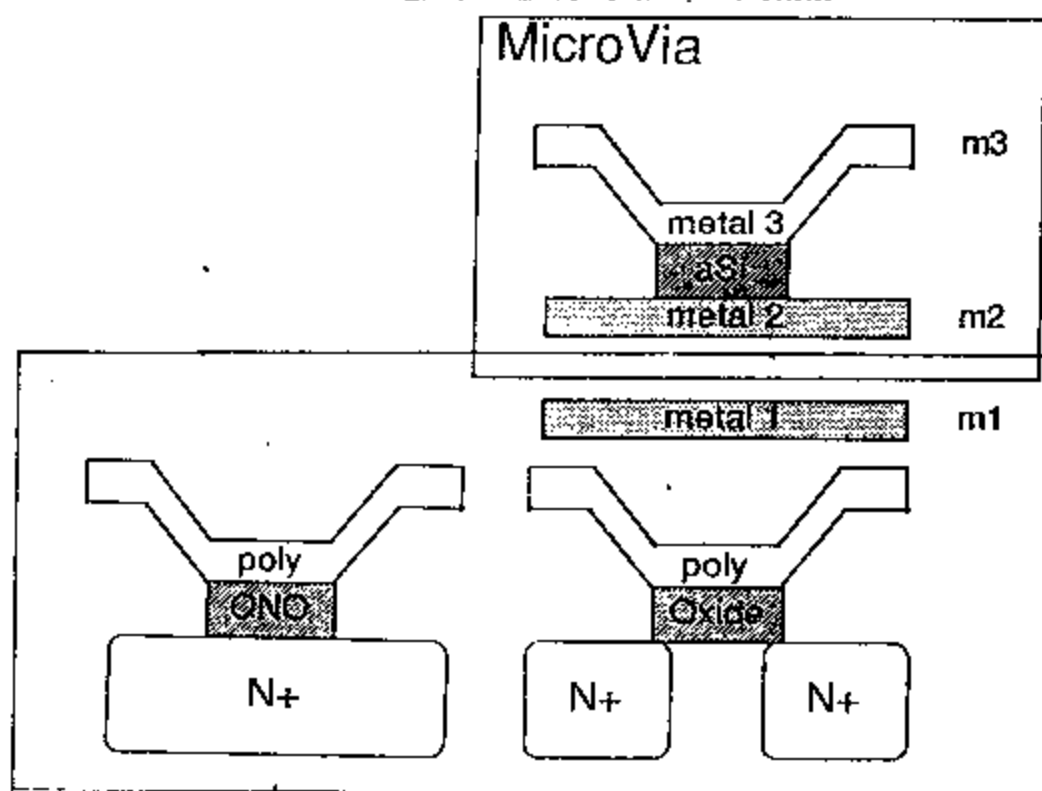


图4.4 普通反熔丝与金属—金属反熔丝的比较

§ 4.3 浮栅编程技术

浮栅编程技术包括紫外光擦除,电编程的 UV EPROM 及电擦除和电编程的 EEPROM 和闪速存储器,这三种存储器都是用悬浮栅存储电荷的方法来保存编程数据的,因此在断电时,存储的数据是不丢失的。

§ 4.3.1 EPROM

这类编程方法可用紫外光或 X—射线照射将编程内容一次全部擦除,然后再编入新的内容,为此要装上一个石英窗,否则是一次可编程器件。

采用 SIMOS(Stacked—gate injection MOS)结构的 EPROM 如图4.5所示,它有两个重叠的多晶硅栅,下面的栅与外界绝缘,称为浮栅,用来存储电荷,上面的栅 G 接到 X 译码器,称为控制栅或选择栅,起控制和选择的作用,所以 SIMOS 结构是单管的结构,它是双管的 FAMOS(Floating—gate Avalanche—injection MOS)结构的改进。SIMOS 是一个增强型管子,在浮栅中没有注入电子时,开启电压较低,称为“1”状态,在浮栅中注入电子后,开启电压较高,称为“0”状态。

浮栅中的电子注入是靠电子沟道注入完成的,为此要在漏与源之间加以足够高的电场,使电子加速,要达到高电场,沟道长度要足够短,例如在源漏电压为20V时,沟道长度要小于4 μm 。当“热电子”的能量超过 $\text{SiO}_2\text{—Si}$ 界面势垒3.2eV时,靠控制栅上的附加正电压,就可以把电子拉到浮栅中去。

在 SIMOS 结构的 EPROM 中,擦除信息时用紫外光照射多晶硅浮栅,其价带和导带中的电子吸收光子,此时受激电子便离开浮栅进入氧化层,在局部电场作用下,被扫到控制栅或衬底,在信息擦除过程中,控制栅,源,漏和衬底都应接近地电位。

一个 SIMOS 管就是一个存储单元,它的写入和读出是依靠 X 及 Y 地址选择线实现的。对于 N 沟 SIMOS 管,当浮栅中无注入电荷时,其控制栅的等效开启电压 V_{T0} 较低;当浮栅中注

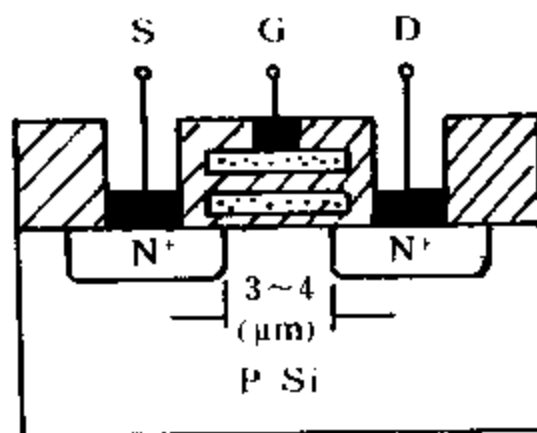


图4.5 构成 EPROM 的 N 沟 SIMOS 管

入电荷后,控制栅等效开启电压提高为 V_{T1} ,X 选择线加电压 V_R ,使 $V_{T0} < V_R < V_{T1}$,由管子是否导通可区别浮栅上是否有电子电荷。

§ 4.3.2 EEPROM

电可擦可编程只读存储器(EEPROM)是可以在电路中用电可擦除和可再编程的可编程只读存储器,对于 EEPROM 有各种工艺,最广泛被利用的工艺是基于浮栅隧道氧化物(FLOTOX),这是由浮栅 EPROM 工艺开发的一种工艺,可制造性和可靠性已被证明,与标准的 NMOS 和 CMOS 工艺也是可兼容的,允许制造与存储器标准有关的高性能的电路。

FLOTOX EEPROM 的基本电路如图4.6所示,浮栅由高质量绝缘物(硅氧化物)围绕,且电气上不与任何其它电极连接。

FLOTOX EEPROM 与 EPROM 之间的显著差别是厚度在100Å范围内的隧道氧化物,它在 EEPROM 中是关键元件。对一个足够薄的氧化层,在场强为 10^7V/cm 的电场作用下,将有一定数量的电子获得足够的能量,穿过二氧化硅的禁带进入到硅的导带,到达导带后,电子可自由地向正极—衬底移动,这就形成 Fowler—Nordheim 隧道效应。在编程中,高压被加到由多晶硅形成的顶栅上,由于源和漏被接地,部份高压被电容耦合到隧道氧化物,电子由漏通过

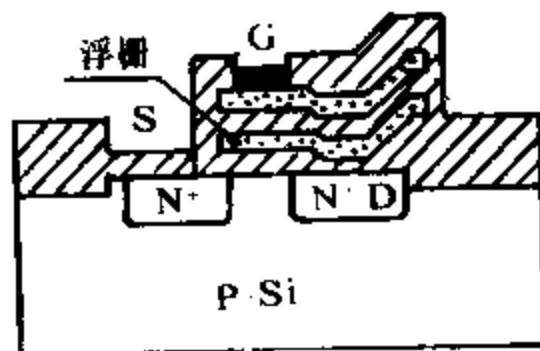


图4.6 构成 EEPROM 的 FLOTOX 管

F—N 隧道氧化物上的电场,减少了电子流,因此电子流是一个自身限制的过程。在擦除的逆过程中,顶栅被接地,源是浮起的,高压被加到漏上,部份电压又通过隧道氧化物被耦合,但方向相反,电子由栅流到漏,在浮栅上得到纯正电荷。浮栅也形成 MOSEFT 的栅极,可用于敏感电荷的状态,在编程状态,浮栅上有负电荷,门限电压向正压方向偏移;擦除之后有纯正电荷在浮栅上,门限电压向负值偏移。

对于 EEPROM 浮栅晶体管是基本存储单元,但是,当利用在阵列中,单个元件不足以适合存储器的功能,需要称为选择晶体管的第二个晶体管。这有两个原因:

1. 有扰动的问题:无选择晶体管,高压加到一个漏上,也加到相同的其它单元的漏上,对其它非选单元也被擦除,而选择晶体管使非选的漏不连接。
2. 浮栅晶体管在阵列中出现跨连,选择晶体管是 EEPROM 存储器的必要部件,限制对于高密度的存储单元的尺寸。

EEPROM 敏感晶体管的运行可以分析它的两个状态来理解:

1. 读出条件下:晶体管是门限电压由浮栅上电荷决定的浮栅 MOSFET,管子性能唯一主要差别是隧道氧化物给出高的漏耦合,因而管子门限电压对漏电压敏感,可以由门限电压随漏电压的变化测量漏耦合。
2. 编程和擦除条件下:此时在高电场区,晶体管的作用不再是重要的,存储单元实质上是接到浮栅上的许多电容,由电荷通过电容电压耦合动态地转移到浮栅可分析存储单元的性能。

§ 4.3.3 闪速存储器(Flash Memory)

从1984年提出闪速存储器的概念到生产出兆位级的器件仅五年的时间,闪速存储器又称为快擦写存储器,“闪速”的名字是为电可擦非易失存储器而创造的一个术语,这类存储器可以用一个单个的信号在几毫秒内完全擦除或擦除一个大的区段,而不象原始的 EEPROM 一次擦除一个字,因此闪速存储器可以有更简单和更有效的芯片结构,因为单个地址和控制线可不必布线到每个单元。闪速存储器是依据擦除式可编程 ROM(EPROM)和电擦除式可编程 ROM(EEPROM)的工艺,在价格和功能上处于这二者之间,适合于先高容量存储,后

能不移出系统再编程要求的任何应用,具有不丢失和快速读取的特性,与 EPROM 和 EEPROM 一样,都是悬浮栅存储电荷的方法保存数据。而且所有的闪速存储器都采用某种形式的分区段存储器阵列。

这个工业标准单晶体管异或型快擦写器件的存储单元,在设计和工艺上与成熟的 UVEPROM 的产品十分相同,Flash-EPROM 单元可以用十分类似于 UVEPROM 所用存储单元的工艺流程来制造,但有以下两点差别,如图4.7所示:

1. 快擦写单元在源区利用分级双扩散;
2. 有更薄的隧道氧化物层,约100Å,EPROM 此层约150Å。

源区的双扩散和更薄的隧道氧化物加强在擦除期间的费累涅尔隧道效应,大多数的闪速存储器用 EPROM 的热电子注入技术来编程的,每个存储单元的场效应晶体管(FET)靠在导电隧道上方浮栅上存在或不存在电荷来接通或关闭,电子能积累在浮栅上是因为在浮栅上方选择栅上大的正电压及当源接地时漏上类似的电压产生的电场,一旦在浮栅上电子被围绕的非导电氧化物截获,它们产生的电场则关闭 FET,逻辑0存储在该位的位置,当无过剩的电子在浮栅上截获,FET 通道可以导通电流,单元有逻辑1的值,因此,单元结构和编程技术二者上,闪速存储器与 EPROM 十分类似。所有的闪速存储器类似 EEPROM 在系统中1或2秒内电擦除,但在容积上类似 EPROM。

闪速存储器的特性非常接近于对随机存储器的理想要求,即系统设计者所期望的可以维持住存储其内的位模式,而又可以在任何时候以新的模式电再编程,闪速存储器存在的一个主要问题是在大多数情况下,其再编程的能力仅保证约十万次,虽然对许多应用这是足够的,但这不等于可以有 SRAM 和 DRAM 那样无限制写操作的能力,在最初引入闪速存储器时,要求两个供电电压,+5V 为逻辑供电,+12V 为编程存储器阵列,当前已趋向于仅需5V 的单电源供电,以简化系统设计,进一步将题供3.3V 的单电源供电,甚至用智能电压(SmartVoltage)于闪速存储器,即智能电压接口允许存储器在三种供电电压的任一种之下执行读操作和再编程。

在可编程器件朝系统内可再编程的方向发展的情况下,电擦除和电编程的 EEPROM 和闪速存储器作为编程元件就有其独特的优点,再加上其断电不丢失也是其特点之一,但还不能做到在系统操作中重新动态地编程那样灵活。

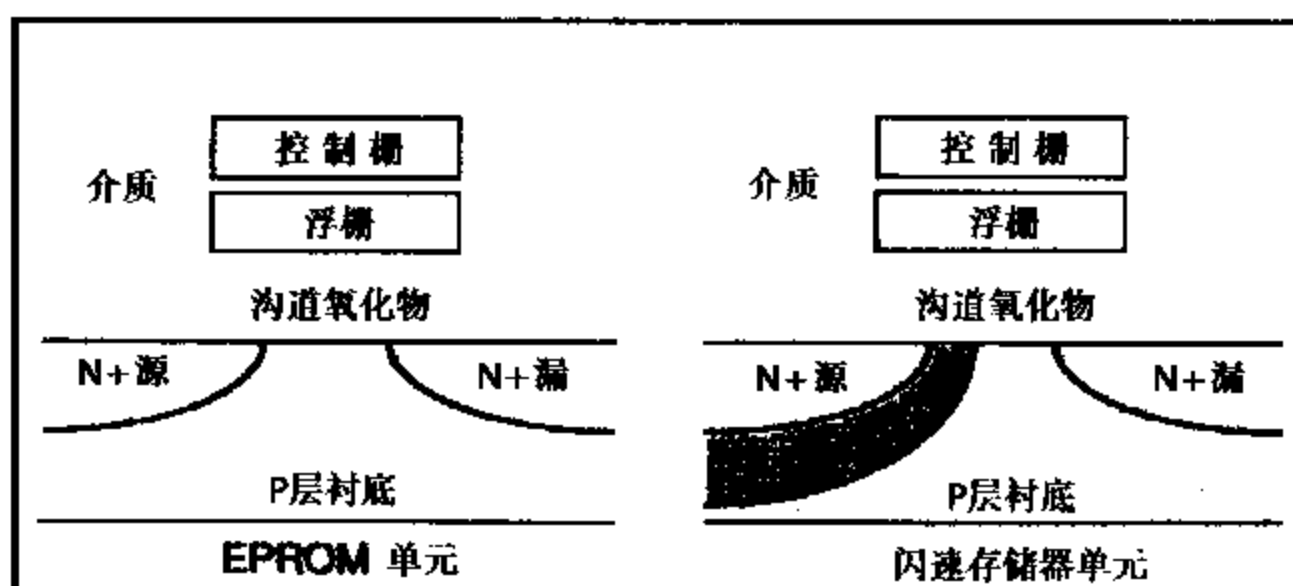


图4.7 Flash-EPROM 存储单元

§ 4.4 晶体管开关

80年代中期,Xilinx 和 Altera 推出了利用晶体管作为开关的器件。这个做法的目的是提供最高数量可用逻辑的晶体管开关,以利用较少的空间和用存储器单元来控制开关,而存储器单元是可以擦除和重新编程的,因此使得器件是可再利用的。Altera 利用 EPROM 单元作控制开关。而 Xilinx 利用静态存储器(SRAM)单元作控制开关。

但是 SRAM 单元必须让编程的模式保存在非易失性的存储器内,通常利用一个小的 EPROM 在邻近,在几个 EPROM 之间转换各种编程模式亦是可能的,目的是当用户从一种运行模式转换到另一种时改变 FPGA 的功能。

片上的静态存储器(SRAM)是配置存储器,用来存储逻辑单元阵(LCA)的配置数据。Xilinx FPGA 可靠性的重要方面是静态存储单元的稳健性。基本单元是单端五晶体管存储单元,省去了第六个晶体管达到高的电路密度。

Xilinx 专门的 FPGA 实现称为逻辑单元阵列(LCA)系列,由存贮在片内的静态存贮单元的配置程序的数据决定用户的逻辑功能和互连。这些静态存贮单元在用户系统加电时进行编程。这些程序的数据按照几种适合不同系统要求的模式的任一种而被加载到 LCA 中。程序数据本身驻存在外部的应用电路板上的 EPROM,EEPROM 或 ROM 中。或者在软盘或硬盘中,其片内的初始化逻辑提供了在加电时可选的自动加载程序数据。Xilinx 提供一个供选择的唯一的最小为八腿的串行配置 PROM 系列来存贮加载的程序。确定所要求的逻辑功能和互连状态的静态存贮单元在用户系统加电时被重新编程。

§ 4.4.1 五管静态存储单元

如上所述,Xilinx FPGA 可靠性的一个重要方面是用来存贮配置程序的静态存贮单元的稳固性。这个基本单元是一个单端五晶体管的存贮器元件。由于取消了第六个晶体管达到了较高的电路密度,配置程序和读回的时间由于没有额外的晶体管会变慢一些,但对器件正常运行时的性能没有关系。其结果是用户在每单位面积上获得了更多的功能和更高的单元稳定性和可靠性。

一个典型的 SRAM 单元是由六个晶体管组成,它包含第二个传输晶体管作为互补位线。因此五晶体管单元和六晶体管单元是相同的,除了附加的位线,它们利用 p 沟道和 n 沟道晶体管。这是与用于商业温度范围 SRAM 中的四晶体管单元完全不同,它只用 n 沟道晶体管。六晶体管单元与四晶体管单元相比具有固有的可靠性的优点。因此 Xilinx 的五晶体管单元具有六晶体管单元的优点,因为去掉的晶体管只是简单的互补位线,它允许更快的读/写时间。

Xilinx 五晶体管配置锁存(单元)是由两个具有有源下拉 n 沟道晶体管和有源上拉 p 沟道晶体管交互耦合的倒相器组成,见图 4.8。高和低电平是用具有分别到电源 V_{cc} 或地 GND 的低阻抗通道的有源器件定义的二个电平。四晶体管 SRAM 利用无源多晶硅上拉电阻具有约 5000 兆欧的阻抗值。1nA 的电流足以倒置 SRAM 单元,但是要翻转 Xilinx 的配置锁存器需大几百万倍的电流。因此五晶体管和六晶体管单元比四晶体管单元固有地更稳定。这不意味着典型的 SRAM 是不可靠的,而是单纯地强调 Xilinx 锁存配置的难以置信的稳定性。

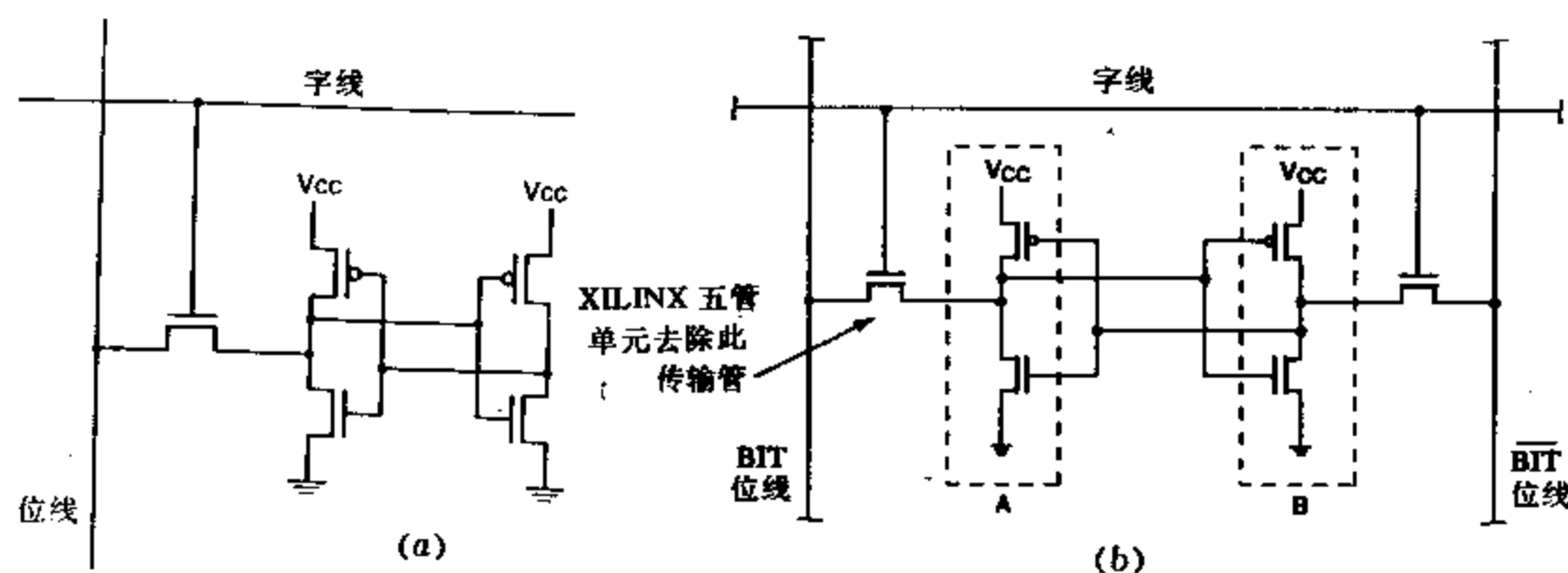


图4.8 五晶体管配制存储单元

与典型的 SRAM 器件比较,宇宙射线或 α 粒子等外界因素引起翻转要高出六个量级。在最初五年中1千万个 LCA 器件售出后 Xilinx 从未听到在配置存储器中自发地变化发生过。

在正常运行期间,Xilinx 五晶体管单元的输出是固定的,因为它们决定用户的配置。LCA 在噪声环境中确保高的数据完整性。在正常运行中,传输晶体管是“截止”的,不影响单元的稳定性。这和普通 SRAM 的存储器件是完全不同的,一般运用它们要连续地被读/写。Xilinx 单元的输出直接或通过缓冲器控制传输晶体管门,由于这些单元的结构,它们不受额外电源变化或十分高级的粒子辐射的影响。

因为这个保持数据的循环连接的倒相器是相邻的,供电晶体管只导致电压上相对小的差别。每个倒相器真正是一个互补的晶体管对。所以,不论输出是高或低,低阻通道存在于供电通道,导致额外的高的抗干扰性。

瞬间的几伏供电或接地对存储的数据没有影响。驱动位线的晶体管已经仔细地被设计以使任何时刻被写的数据和存储的数据是相反的,它可以取代反馈倒相器的输出。写操作的可靠性是在制造工艺的容错之内来保证的。在读模式,具有可观数值的寄生电容的位线被预充电到逻辑1。传输晶体管则靠驱动字线到高电平而被使能。如果存储的值是零值,字线则放电到地。存储器单元可靠地读是减少读期间字线的高电平到确保这个单元不被干扰的电平来达到的。

§ 4.4.2 考虑供电电压的起伏

因为 Xilinx LCA 是在静态锁存器中存储它们的配置,理解在非正常供电电压条件下配置程序的完整性是很重要的。

对于 XC3000系列,当 V_{cc} 开始供电,且仍低于3.0伏,器件在预初始化模式中唤醒。HDC 是高,INIT,LDC 和 D/P 是低,所有其它的输出是具有弱上拉电阻的三态。

当 V_{cc} 已经升到大约3伏的数值,1和0已经成功地写到配置存储器两个专门的单元,初始化的上电时间延迟开始,这个延迟补偿 V_{cc} 检测门限在菊花链中器件之间内部 CCLK 振荡器频率的差。初始化延迟计数芯片上振荡器(CCLK)的时钟周期。它有3:1的频率范围,取决于工艺,电压和温度。所以超时发生在对从属器件的11和33ms 之间,相对于主器件的四倍

长。四倍的因素为确保即使最快的主器件将总是比任何从属的发生更长。假设在33ms 和 $4 \times 11\text{ms}$ 之间的最坏情况的差足以补偿在菊花链中器件的门限差之间 V_{cc} 化费的上升时间, 仅仅在 V_{cc} 上升时间十分慢的情况, 用户必须保持 RESET 低, 直到 V_{cc} 达到适当的电平。在初始化时间超出的末尾, 每个器件在几分之一毫秒内清除它的配置存储器, 然后对不作用的 RESET 测试, 存储此模式输入, 开始配置过程。

在器件被配置之后, V_{cc} 可能降到大约3.5伏, 除了延迟增加(电路速度正比于 V_{cc}) 和输出激励减小, 而没有任何显著的影响。

如果 V_{cc} 降的更低, 有源输出变的更弱, 则器件停留在预初始化模式。当 V_{cc} 又一次升高, 正常的配置过程象上面所述那样地初始化。

用户不需要考虑供电电源的降低。XC3000系列 LCA 对于电压小的降落保留配置, 否则在主模式将自身重新配置, 如果在从模式把 INT 和 D/P 拉低来请求配置。一个 XC3000系列的器件不会锁死。因为用户可以在任何时候将 D/P 拉低, 或者当 D/P 是低时, 迫使 RESET 由高变低来重新配置。

Xilinx 可编程门阵列贮存它们的逻辑配置和片内互连模式在锁存器中, 当电源加到器件上或者任何时候重新配置被启动, 所有配置信息被写到这些锁存器中。这些过程的状态图如图4.9所示。显然, 所有配置信息当电源中断时将失去, 一些用户表示关心这一点。

V_{cc} 下降引起器件混乱也就是用错误的配置信息运行是不可能的:

①如果 V_{cc} 停留在转折点(大约三伏)器件功能正常, 只是象所有 CMOS 器件那样减少速度。

②如果 V_{cc} 下降到低于转折点, 器件所有输出为三态等待重新配置。

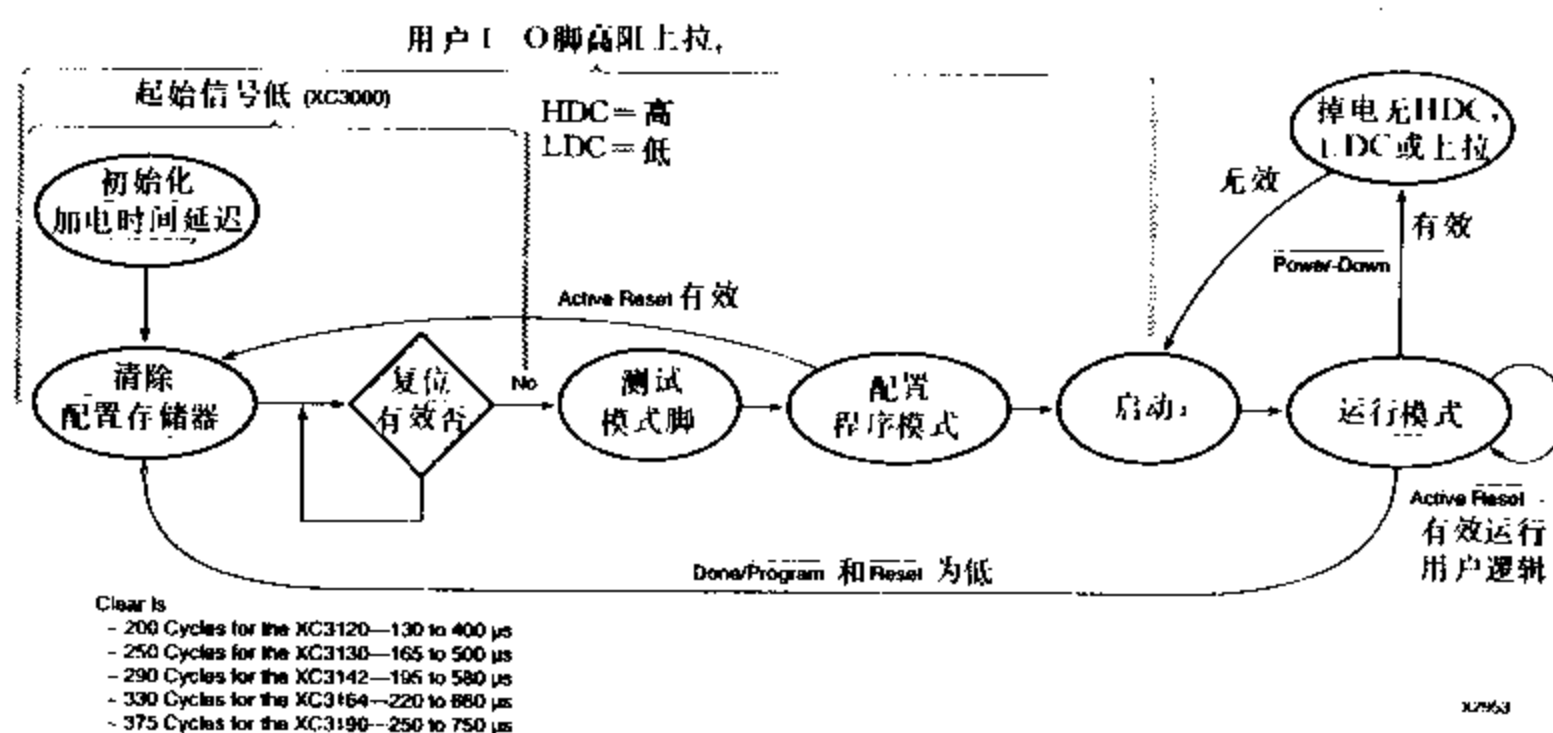


图4.9 加电和重编程时配制过程状态图

关于 Xilinx FPGA 的配置问题在第九章还要作专门讨论。

作为本章的结束,给出几种编程技术的比较,见表4.2。

表4.2 四种编程技术的技术性能的比较

1.2 μ mCMOS	反熔丝			UV EPROM		电擦除		SRAM
	ONO	非晶硅	金属—金属	OTP	窗口型	E ² PROM	闪存存储器	
可再编程能力	否			否	是	是		是
系统内编程	否			否		是		是
编程速度	慢			快		快		快
易失性	否			否		否		是
擦除时间	/			慢		快		快
测试效能	有限			有限		全部		全部
成品率	93—97%			98—99%		100%		100%
接通电阻(Ω)	300—500	50—100	<50	2K—4K		2K—4K		0.5K—2K
分布电容(fF)	5	1.1—1.3		10—20		10—20		10—20
增加工艺步骤	3			3		5		0
面积	熔丝小,编程管大			小		2EPROM 1EPROM		大
典型产品	ACTEL QUICKLOGIC CYPRESS			XILINX ALTERA		ALTERA CYPRESS		XILINX
	ACT1/2/3 pASIC380			XC72/7300 MAX5000		MAX7000 FLASH370		XC4/3/2000

第五章 可擦除式可编程逻辑器件(EPLD)

§ 5.1 概 述

本章主要以 Xilinx 的可擦除式可编程逻辑器件(EPLD)为主介绍复杂的可编程逻辑器件。从方块图来看,所有的 EPLD 器件的结构都很类似,这可以比较图2.6,图2.7和图5.6看出,即它们都有一些包含与-或结构及触发器的功能块,每一功能块都约等于一片 PAL22V10的密度,此外就是互相连接的布线资源和 I/O 块。

Xilinx 提供了两种系列的 EPLD, XC7200/A 系列和 XC7300 系列,这两个系列结合了芯片集成度高,尺寸小,成本低和可靠性高以及按用户需要产生专用集成电路的优点,而无掩膜可编程门阵列的成本,时延和风险等问题。

Xilinx 的 EPLD 有多个称为功能块的可编程逻辑结构,通过没有限制的通用互连矩阵(UIM)连接在一起。这些功能块,布线资源以及 I/O 块又与传统的 EPLD 不一样:每个功能块除包含九个由可编程与/或阵列驱动的宏单元外,还有一个算术运算单元(ALU)和专用进位逻辑,它们使得在复杂计数器和控制器的设计中可用较少的资源达到更高的速度,在应用中 ALU 不用于运算也可用于两输入的与,或,异或等逻辑功能;作为互连布线资源,UIM 是保证100%的布通率,即任何器件的输入和任何宏单元的输出可以接到任何其它宏单元的输入,这个无限制的可编程互连结构由于和传统的 PAL 结构的与/或逻辑结合而使得 EPLD 容易利用和容易理解,它完全消除了可布通率的问题,此外 UIM 还可用于线与等功能;I/O 块中包含有锁存器和寄存器等功能,可不占用宏单元资源实现输入锁存等;在 XC7300 系列中还有高速和高密两种功能块的结构。

通过 Xilinx EPLD 器件的延时不仅是可预测的,而且是常数,任何在一个功能块内可实现的函数将以器件规定的速度来运行。

Xilinx 的 EPLD 器件采用流行的 CMOS EPROM 工艺,所以是100%在出厂前被测试的。在电路中提供专门的运行测试模式,可加载任何位结构的配置到内部移位寄存器而不必编程 EPROM 阵列。亦即任何逻辑信号通道可以通过空的 EPROM 阵列建立以测试每个可能应用的功能和速度。

表5.1 Xilinx 的 EPLD 产品

	等效22V10	宏单元数	功能块数	触发器数	快输入数	信号腿数
XC7236A	4	36	4	68		36
XC7272A	8	72	8	126		72
XC7318	1.5~2	36	2	18	12	38
XC7336	3~4	36	4	36	12	38
XC7354	6	54	6	108	24	58
XC7372	8	72	8	126	30	84
XC73108	12	108	12	198	42	120
XC73144	16	144	16	234	54	156

EPROM 单元阵列在工厂对其石英窗封装器件的可编程性,数据保持和紫外线擦除能力作了100%的测试。所有上述测试方法的综合结果是100%缺陷复盖的,由半成品的工厂测试保证。器件是在全部交流和直流运行限制下被测试的,在门电路中的 MOS 晶体管作电气测试和装运前的受力测试。除了内在的自检和自受力特性外,Xilinx 的 EPLD 接收在线的产品高温烘烤以淘汰不合格产品。最后,全部器件进行在线动态加载应力测试,对不合格的产品作进一步淘汰。

§ 5.2 XC7200A 系列

§ 5.2.1 概述

XC7200A 系列是第二代高密度可编程逻辑器件,它结合了经典的类似 PAL 的 EPLD 结构和创新的面向系统的逻辑扩充的功能。这些支持快速状态机,大的同步计数器,快速算术运算以及多级通用逻辑。在可达到的系统时钟速率和临界的延时下测得的性能不仅是可预测的,而且与实际的逻辑映射,互连布线,资源利用无关。所以在设计反复时性能保持不变,对于在任何一个输出宏单元中实现的任何功能,它通过互连和逻辑的传播延迟是常数。

通过对每个宏单元中的算术逻辑单元(ALU)附加可用的门和控制功能来增强传统的可编程逻辑阵列结构功能的通用性。直接运行在相邻宏单元和功能块之间的专用快速算术进位线支持最高达36位的任何长度快速加法器,减法器和比较器。

在每个宏单元中这个附加的 ALU 可以产生任何两个积之和的组合功能,它可在相邻宏单元和功能块之间产生和传播算术进位信号。

通用的互连矩阵(UIM)方便于从所有器件输入端和宏单元输出端到任何功能块的与阵列输入端无约束固定延迟的互连。UIM 亦可以对接在这个通道上进入到任何功能块的任何数目信号执行逻辑与,没有附加延时地增加另一级的逻辑。这个支持任何高达36位大小的、运行在器件规定的最高频率上的双向可加载同步计数器。

作为这些逻辑扩充的结果,XC7236A 可以提供高的性能。即使在设计中每个输出要组合大量的乘积项,或者需要比与一或关系更多层的逻辑,或者在某些乘积项中需要更宽的与功能,或者执行更广的算术功能。

Xilinx 的开发软件支持自动设计映射,软件用第三方的原理图输入工具。PLD 编辑器或直接的文本式方程的文件作为设计起步的基础,设计映射在 PC 机上几分钟内可以完成。

§ 5.2.2 结构

图5.1表示 XC7236A 的结构。四个功能块(FB)由一个中央通用互连矩阵(UIM)全部被互连。每个 FB 从 UIM 接收21个信号,每个 FB 产生9个信号反馈到 UIM。全部器件的输入亦是经过 UIM 被布线到所有的功能块。每个功能块包括9个输出宏单元(MC),每个宏单元由来自 UIM 的21个信号形成一个可编程与阵列,大多数宏单元驱动一个三态的芯片输出,全部反馈到 UIM。

这个器件也包括六个专用的快速比较和快速译码的逻辑通道用于地址比较,译码或门功能。

1. 功能块和宏单元

XC7236A 包含相同结构的36个宏单元,分成四个功能块,每个功能块九个宏单元。每个宏单元由功能块中可编程与阵列形成的乘积项驱动。每个功能块中的与阵列接收来自 UIM 的21个信号和它们的补。在三个功能块中,与阵列接收三个直接来自 FastInput(FI)的附加输入和它们的补,因此提供了较快速的逻辑通道。

五乘积项是对每个宏单元专用的。一个附加的12乘积项是在每个功能块的九个宏单元之间共享的。专用乘积项的四个可以有选择地和高达四个共享乘积项相或。并驱动 D1 输入到算术逻辑单元。其它的输入和到 ALU 的 D2 由第五个专用乘积项及高达八个剩下的共享乘积项相或来驱动。

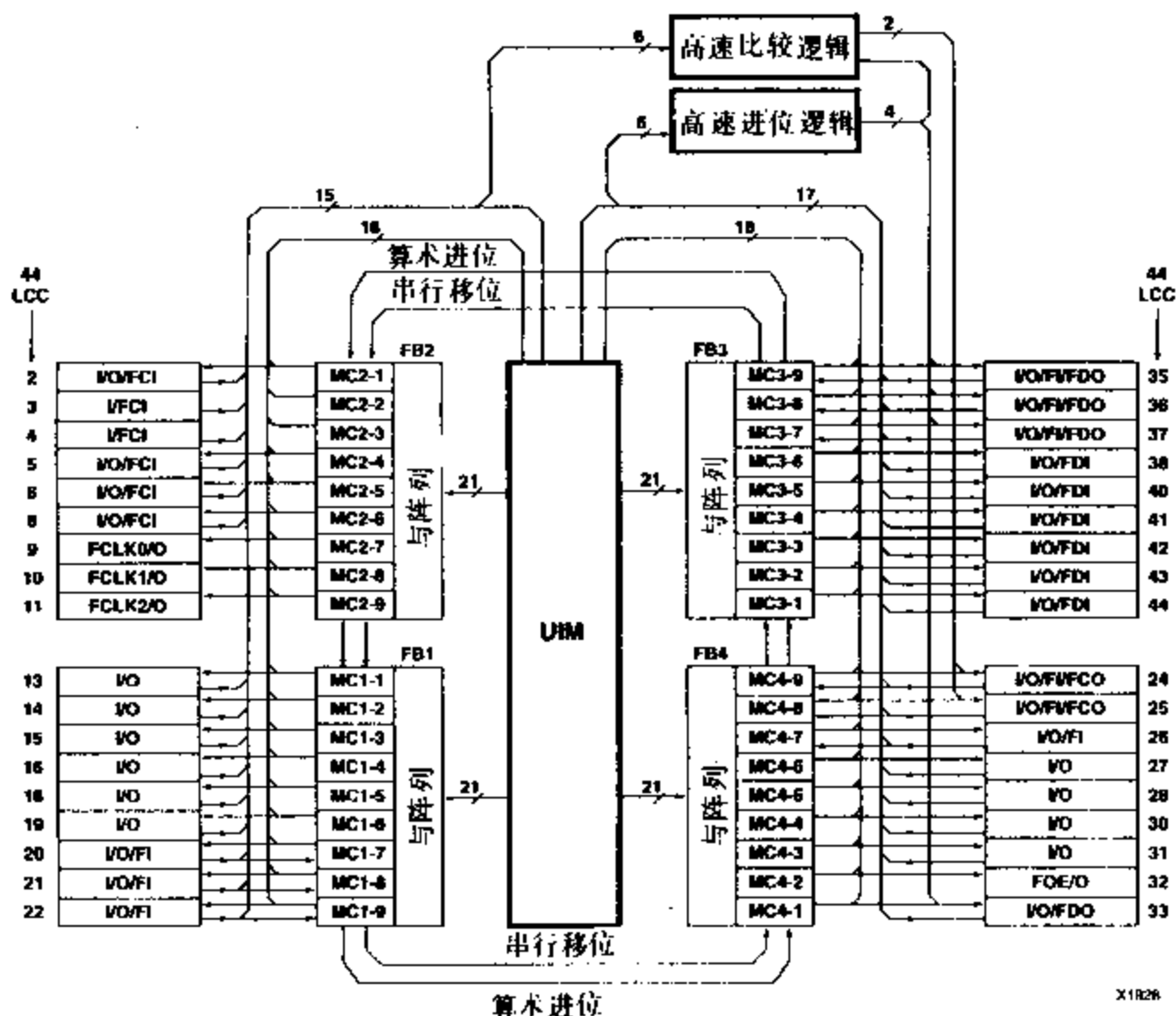


图5.1 XC7236 A 结构

作为一个可编程的选择,四个专用乘积项可以用于其它的目的,一个专用乘积项可以用来作为宏单元中触发器的专用时钟。(见下面描述的其它时钟选择。)专用乘积项的另一个可以是宏单元触发器的非同步高电平有效的复位端,再有一个可以是宏单元触发器非同步高电平有效的置位端。第三个可以是输出使能信号。

作为配置选择,宏单元输出可以被反馈,并和三个共享乘积项相与之后,进入 D2 输入端与 ALU 相或,以便实现计数器和触发器。

算术逻辑单元有两个可编程模式:在逻辑模式,它是一个两输入端函数发生器,一个四位查找表,它可以被编程来产生任何的两个输入变量的布尔函数。可以把它们相或,加宽这或函数到最大17个输入变量;可以把它们相与,这意味着一个积之和可以用来屏蔽另一个;也可以把它们异或,触发触发器或比较两个乘积项的和。任一到 ALU 的积之和的输入或两

者可以倒相,也可以不倒相。ALU 可以没有任何速度损失地实现一个附加的逻辑层。

在算术模式中,在每个宏单元中 ALU 块可以被编程来产生两个操作数的算术和或差,与从下一个低位宏单元表的进位信号组合;它也送出一个进位输出到另一个高位的宏单元,这个进位传送链跨过功能块之间的界限。这个专用的进位链克服了传统的 EPLD 结构在试图执行算术功能时固有的速度和密度问题。

ALU 输出激励宏单元触发器的 D 输入。每个触发器有几个可编程选择。一个选择是使它透明来取消触发器,它使得 Q 输出和 D 输入相同,与时钟无关。另外,在普通方式下运行触发器由它的输入时钟的上升沿触发。

时钟源是可编程的:它或者是以前提到的专门的乘积项,或者是两个全局快时钟信号 (FLCK0 或 FLCK1) 之一,它们是用最短的延迟和最小的扭曲分布在整个芯片。

非同步的置位和复位(清除)输入使时钟运行无效。当两个非同步是瞬时起作用的,复位使置位无效。一旦加电,每个宏单元触发器可以用 0 或 1 预加载。

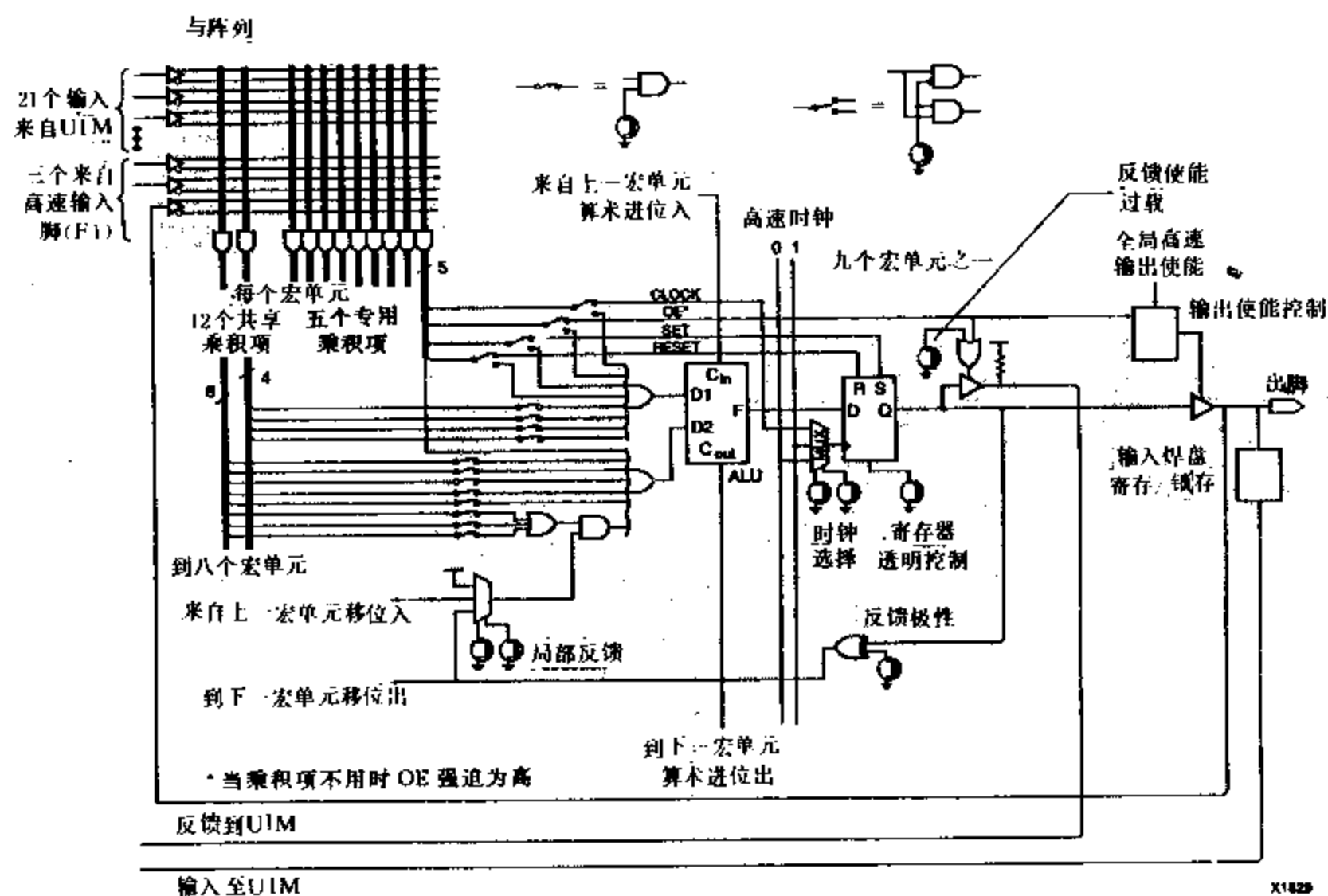


图 5.2 功能块和宏单元原理图

除了驱动一个芯片的输出腿,宏单元输出亦作为到 UIM 的一个输入布线返回。一个专用的乘积项可以配置来控制输出腿驱动器的输出使能和/或反馈到 UIM。如果配置成控制 UIM 反馈,则当 OE 乘积项是非确定时,UIM 反馈线被迫使变高而截止。

2. 输入和输出

36 个宏单元的 3/4 直接通过各个由三态输出缓冲器跟随的可编程倒相器驱动芯片的输

出;每个可以由上面提到的输出使能乘积项分别控制。一个附加的配置选择永久地使输出失效。一个专门的 FastOE 输出也可以配置成控制任何芯片的输出替代或者用一个单独的 OE 乘积项来连接。

每个到芯片的输入信号可以被编程为直接的,或锁存的,或在触发器中寄存的。锁存器和触发器可以用两个快时钟信号之一编程作为锁存使能或时钟。两个快时钟信号是 FCLK0 和 FCLK1 或 FCLK2 的任一全局选择。当快时钟是高时,和在快时钟上升沿时触发器时钟时锁存是透明的。寄存的输入允许高的系统的时钟,在它们承受器件的组合延迟之前流水地输入,假若一个时钟周期的流水延迟是可接收的。

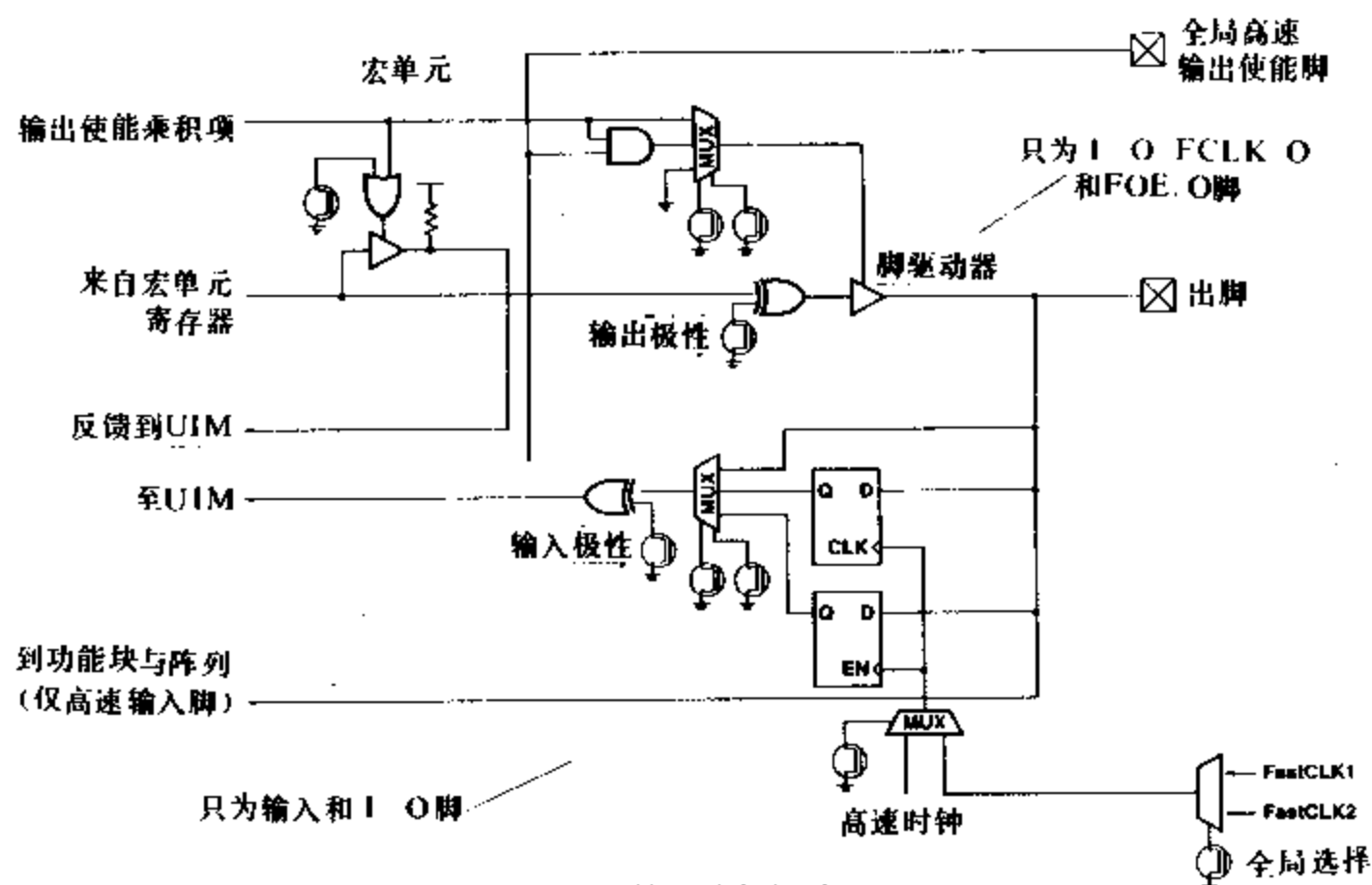


图5.3 输入/输出原理图

直通,锁存和寄存输入然后驱动 UIM。在纯输入和 I/O 输入之间没有传播延时的差别。

3. 通用互联矩阵

UIM 接收68个输入;36个来自宏单元反馈,30个来自双向 I/O 腿。2个来自专用输入腿。UIM 起着不受的纵横开关的作用,产生84个输出信号,21个到每个功能块。

68个输入任何一个可以被编程与84个输出的任一个连接。通过这个阵列的延迟是常数。与实际的布线距离扇出扇入或布线复杂性无关。布通率不是一个问题,任何 UIM 输入可以驱动任何 UIM 输出,甚至多个输出,延迟是常数。

当多个输出被编程连接到相同的输出。当这级被看作有效的高电平时,这个输出变成输入信号的相与。靠选择在输入,宏单元输出和功能块与阵列输入上的相应信号型式,这个与逻辑也可以被用来实现与非,或,或非功能,因此没有任何速度损失提供附加的逻辑级。

宏单元反馈信号用输出使用乘积项使其失效表示高电平输入到 UIM。

几个这样的宏单元输出编程到相同的 UIM 输出因此类似三态总线。如果宏单元输出的一个被使能。UIM 输出设置成相同的电平。

§ 5.2.3 应用特性

1. 编程和使用

按照或者由原理图输入或者由布尔方程表达的行为描述给出的技术条件,器件的上述特性和能力被 Xilinx 的开发软件利用来编程这类器件。

在器件被编程和校验之后,用户可以规定安全位来保护任何编程位映射的读出。此类器件是按 EPROM 相类似的方式用 intel 16进制格式被编程,来自许多编程器制造厂的编程支持是有效的。UIM 连接和功能块与阵列连接直接由非易失的 EPROM 单元来做的,其它控制位是在一加电后从 EPROM 阵列读出并存入锁存器中,通常在 EPLD 器件中,这个方法要求或者十分快的 V_{CC} 上升时间($<5\mu s$),或者应用主机复位信号延时至少到 V_{CC} 已达到要求的运行电压。后者可以在 MR 腿上利用电容和上拉电阻来达到,RC 周期应大于 V_{CC} 上升时间的两倍。加电和复位信号起动持续时间达 $350\mu s(t_{RESET})$ 自定时配置周期,在此期间所有器件输出保持禁止,被编程的预加载状态值送入宏单元寄存器。不用的输入和 I/O 腿应接地,或接 V_{CC} ,或某个有效的逻辑电平,对于所有 CMOS 器件这是通常的方法以避免消耗过多电流通过输入焊盘电路。在三个 V_{CC} 腿上去耦电容应该用数值达 $1\mu F$ 的高速陶瓷电容。

2. 定时和延时通道规定

延时通道由可以串联连接的三个功能块组成:输入缓冲器和相应的锁存器或寄存器;UIM,与阵列和宏单元等逻辑资源;三态输出缓冲器。

不管扇出或位置,所有的输入有相同的延时;不管逻辑的复杂性,互连拓扑或在芯片的位置,所有的逻辑资源有相同的延时;所有的输出有相同的延时。所以可达到的时钟率仅仅由输入方法,如直接输入,锁存输入或寄存输入,以及由信号通过组合逻辑的次数来确定。

§ 5.3 XC7200系列

Xilinx 的 EPLD 在 XC7200A 系列之前是 XC7200系列,这两个系列在结构上是完全相同的,只是 XC7200系列还包括快速译码器和快速比较器,所以这节仅介绍这两部分,其它部分可参考 § 5.2。

1. 快速译码器

快速译码器单元包括四个快速可编程6位译码器,带有通常的一个六输入组(FDI)。每个译码器把输入端数据与被编程的6位固定数值比较,并驱动被指定的芯片输出(FDO)。每个译码器可用"don't care"位编程,每个可以专门匹配高有效或低有效来作为可编程选择。

2. 快速比较器

两个快速比较器单元包括两个快速可编程,带有通常的一个六输入组(FCI),与 FDI 输入端分开。每个比较器输入端的数据与存储在其六个锁存器的模式比较,并驱动被指定的芯片输出(FCO)。数据可以从快速比较数据输入端被加载到这些锁存器,或者在芯片配置期间(加电或置位)预先加载,或者预先编程进器件内。

每个比较器用 Don't care 位可编程,且可以用快速译码器 FDO 输出一个或更多的结果调节。比较器可以被禁止(迫使出错),且可选匹配响应的极性。因为比较电路旁路 UIM 和宏单元,它是十分快的,也可以用作高速地址译码器。

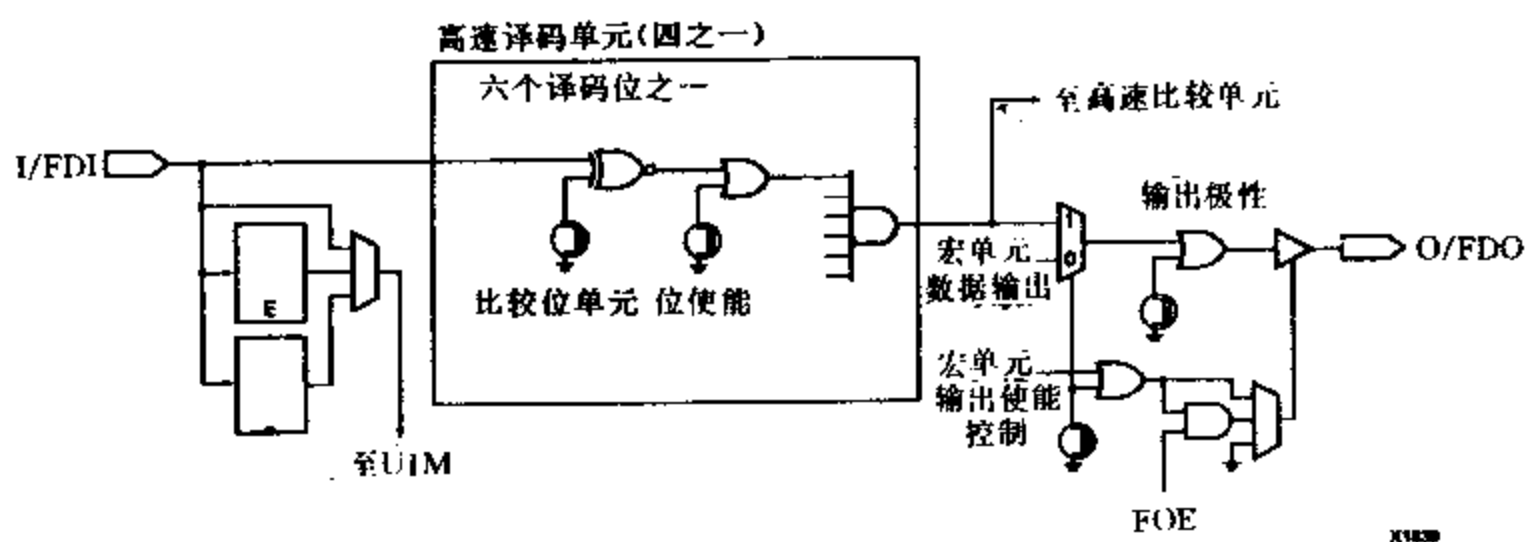


图5.4 快速译码器原理图

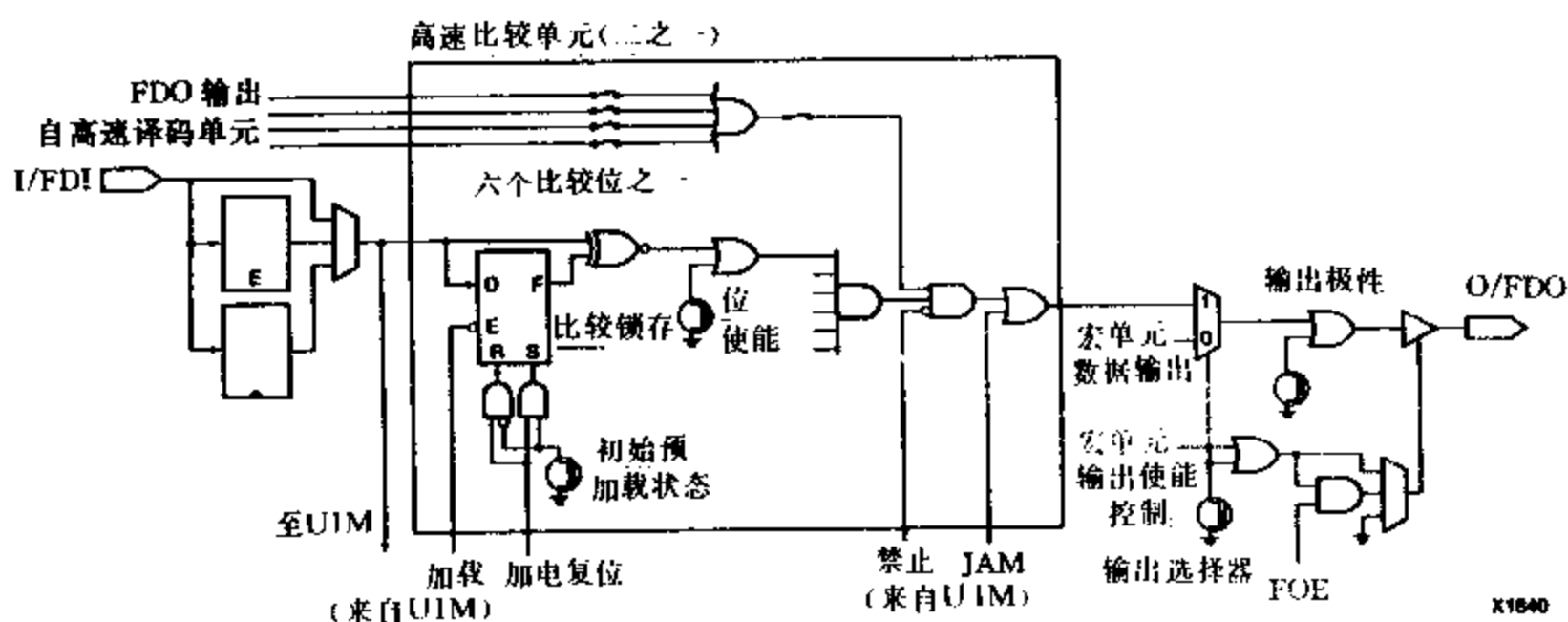


图5.5 快速比较器原理图

§ 5.4 XC7300系列

§ 5.4.1 概 述

XC7300系列采用了独特的双功能块结构,设计者在需要的时候,现在可以取高速通道的优点,而不会失去实现复杂功能的能力或丢弃定时的可预测性。

这个独特的能力是用组合两种功能块在相同的器件内来实现的,高速功能块(FFB)提供快速的腿到腿的速度和逻辑流量,用于临界的译码和极高速状态机应用。高密功能块(FB)提供最大的逻辑密度和系统级特性用可预测的定时实现复杂的功能,如加法器和累加器,要求大量乘积项的宽函数和状态机以及其它形式的复杂逻辑。

此外,XC7300结构采用通用互连矩阵(UIM)保证100%全部内部功能的互连,这个互连方式提供了对所有通过UIM的布线通道固定的短的互连延时,固定互连延时简化了器件的定时和保证了设计性能,而不管芯片内的逻辑布局。所有XC7300器件是用0.8 μ m CMOS E-PROM 工艺设计的,支持12ns 腿到腿延时和高达80MHZ 的系统时钟。

所有XC7300EPLD 包括可编程的功率管理特性,在单独的宏单元到宏单元的基础上规

定高性能或低功率运行,不利用的宏单元自动地关闭以使功耗最小,设计者可以在最大的性能上运行临界通道,而非临界通道消耗较小的功率。

Xilinx 开发软件支持 XC7300系列 EPLD 的设计,利用第三方的原理图输入工具,HDL 编译器或直接地基于方程的文本文件,利用 PC,工作站和这些设计输入工具之一,设计被自动地映射到 XC7300的 EPLD 仅需几分钟。

XC7300系列器件可用塑料和陶瓷的 FPGA 和 QFP 封装,封装选择包括为设计样机的陶瓷窗口封装和用于低成本产品的一次可编程塑装形式。

§ 5.4.2 结 构

Xilinx EPLD 的 XC7300系列结构如图5.6所示,它是用通用互连矩阵(UIM)互连的多个可编程功能块组成。双功能块结构包括两类功能块:高速功能块和高密功能块,两类功能块和 I/O 功能块再通过通用互连矩阵(UIM)相互连接起来的。

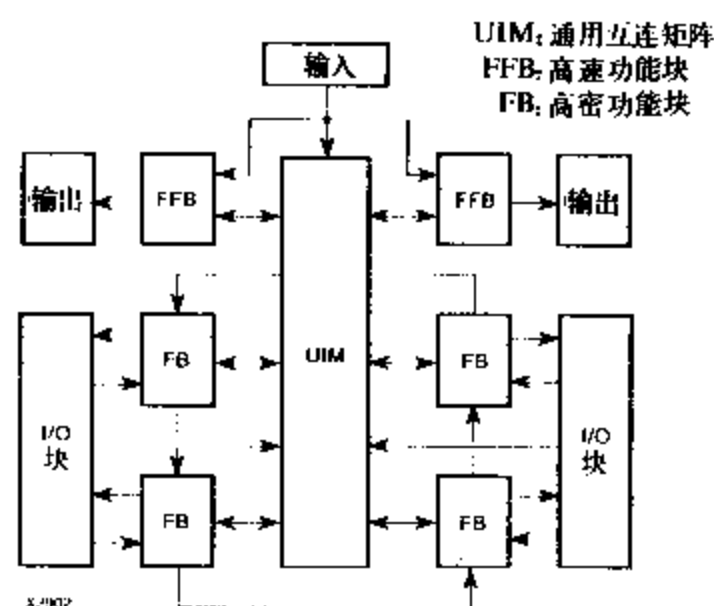


图5.6 XC7300器件方块图

1. 高速功能块

它接收来自 UIM 的24个信号及它们的反相信号。这24个输入可以从 UIM 中单独被选择,12个快速输入腿或九个来自高速功能块的宏单元反馈,在每个高速功能块中可编程与阵列产生45个乘积项来驱动每个高速功能块中的九个宏单元,每个宏单元可以配置成寄存的或组合的逻辑,见图5.7。

来自可编程与阵列的五个乘积项被分配到每个宏单元,这些乘积项中的四个被或在一起,并驱动可编程 D 触发器的输入,第五路乘积项驱动宏单元触发器的异步高有效置位输入,触发器可被配置成对组合输出是透明的。

可编程时钟源是两个全局高速 CLK 信号(FCLK0和 FCLK1)之一,它们以短延时和最小的畸变分布于整个芯片上。

高速功能块宏单元直接通过三态输出缓冲器驱动芯片输出,每个输出缓冲器可以用两个专用 FastOE 输入之一来单独控制,永久使能或永久禁止,宏单元输出也返回布线作为高速功能块的一个输入和 UIM 的一个输入。

2. 乘积项分配

XC7300系列利用乘积项分配方式提供乘积项的灵活性,而不使宏单元输出失去能力。

对于每个宏单元的积之和的或门可以利用高速功能块乘积项分配被扩充,乘积项分配按四个乘积项的增量把乘积项从一个宏单元转到另一个宏单元,要求高达36个乘积项的复杂逻辑函数可以用乘积项分配来实现。当乘积项被分配到相邻的宏单元,通常专用于置位功能的乘积项把D输入变成宏单元寄存,因而宏单元仍可利用,而乘积项被转移到相邻宏单元,见图5.8。

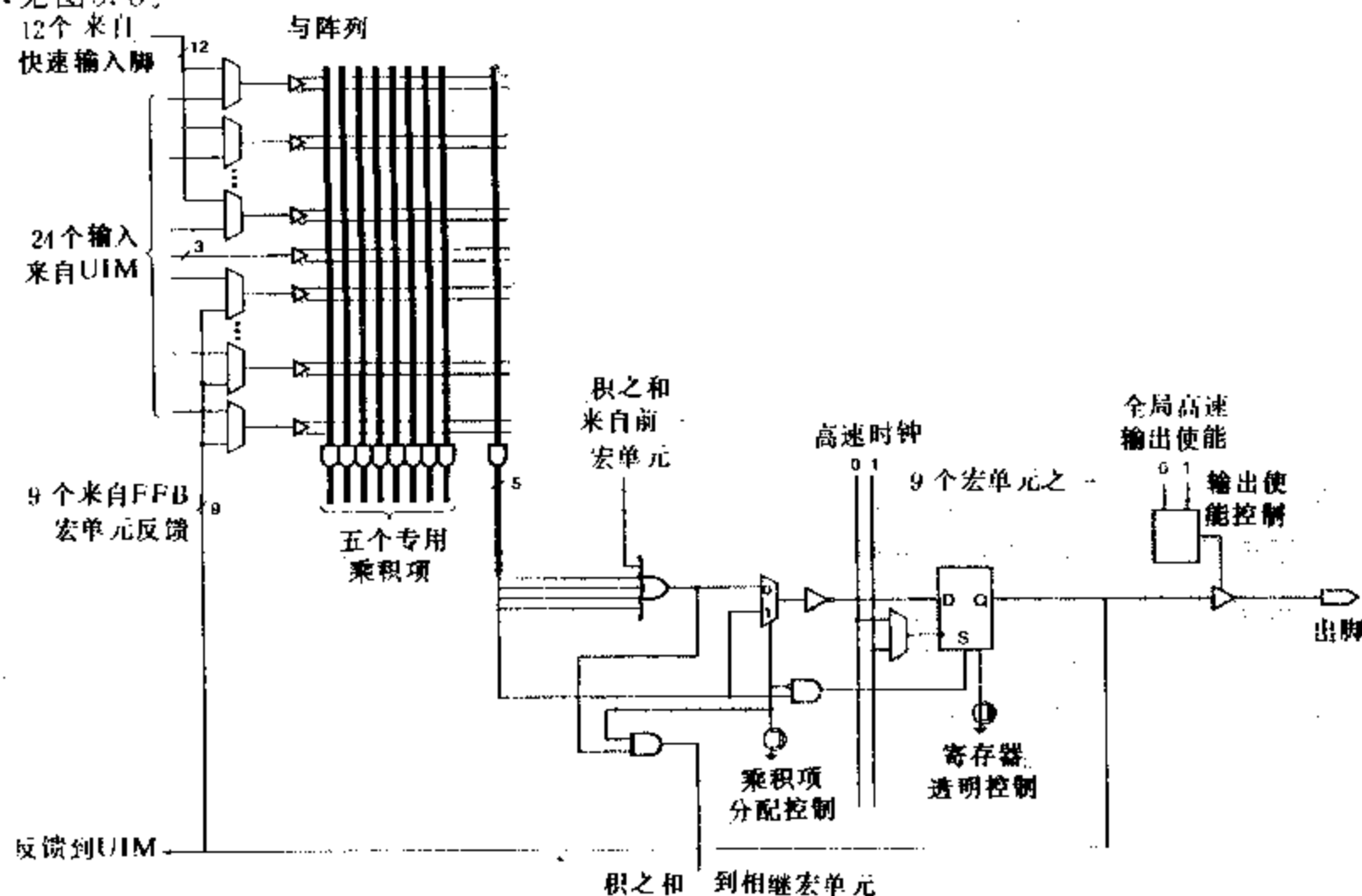


图5.7 高速功能块宏单元原理图

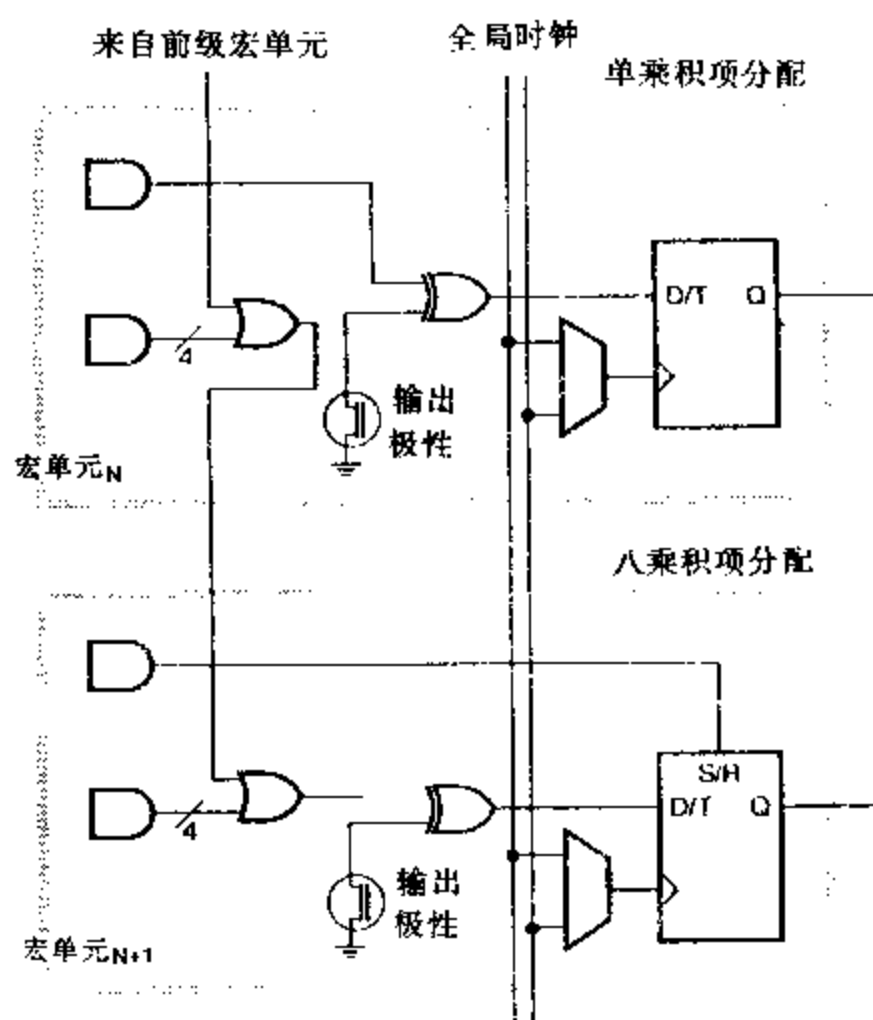


图5.8 高速功能块乘积项的分配

3. 高密功能块

XC7300的每类器件包含多个通过 UIM 连接的高密功能块,每个功能块包含九个宏单元,每个宏单元可以对或者寄存或者组合逻辑来配置,XC7300的高密功能块是和 XC7200系列的功能块相同的,所以详细功能见图5.2所示。每个功能块接收21个来自 UIM 的信号和它们的补,及来自快速输入(FI)腿的三个输入。

4. 共享和专用乘积项

每个宏单元包含五个专用乘积项,可以用来作为 ALU 中实现的组合函数的专用输入,或作为单个的复位,置位,输出使能和触发器的时钟逻辑函数,每个功能块也提供附加的12个共享的乘积项,在功能块内九个宏单元的任一个它是可用的独立的乘积项。

四个专用乘积项可以用高达四个共享的乘积项或在一起来驱动 ALU 的 D1输入端,D2输入端由第五个专用乘积项和高达八个余下的乘积项相或来驱动,共享乘积项相加无逻辑延时,每个共享乘积项可以接到功能块中的一个或全部九个宏单元。

5. 算术逻辑单元

每个宏单元的多功能性通过 ALU 中附加的门和控制功能是可用的而进一步增强,XC7300的 ALU 详细的方块图在图5.9中表示。

ALU 有两个可编程模式:逻辑的和算术的。在逻辑模式,ALU 功能作为两输入函数发生器利用四位查找表面可以被编程来产生它的两输入的任何布尔函数,函数发生器可以把它的输入相或,增大或函数到最大17个输入。可以把它们相与,这意味着一个积之和可以用来屏蔽另一个。也可以把它们异或起来,触发触发器或比较两个乘积项的和,任一个到 ALU 的积之和的输入或二者可以倒相或者弃之一边。所以 ALU 可以实现无任何速度损失的一个附加的逻辑层。在算术模式,ALU 方块可以被编程来作为 D1和 D2输入的算术和与差,与下一个低位的宏单元来的进位输入组合,ALU 作为一个全加器运行产生进位输出到另一个高位宏单元。相邻宏单元之间1的进位链传播也跨过功能块之间的界限,这个专用的进位链克服了传统的 EPLD 在试图执行算术功能时固有的速度与密度的问题。

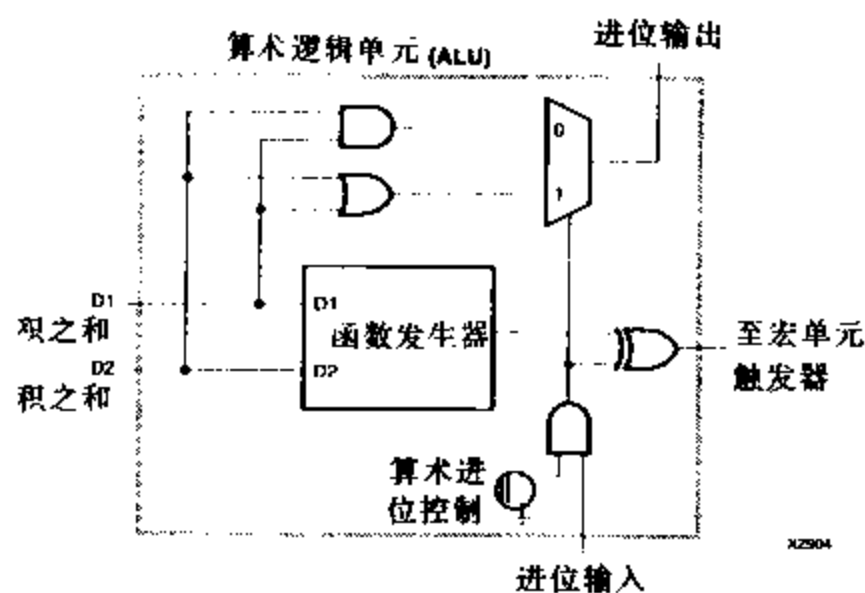


图5.9 ALU 原理图

6. 超前进位

每个功能块提供一个超前进位产生器,有预处理跨过所有九个宏单元进位的能力,超前进位产生器减少了宽位的算术函数如加、减等的脉动进位(ripple-carry)延时,与最初九位的进位幅度比较,加上高阶功能块的超前进位延时。

7. 宏单元触发器

由 ALU 块的输出驱动可编程 D 型触发器的输入, 触发器由时钟输入上升沿触发, 但是它可以配置为透明的, 使 Q 输出与 D 输入相同, 与时钟无关, 或者作为普通的触发器。

宏单元时钟源是可编程的, 可以是专用乘积项的一个或两个全局快时钟信号 (FCLK0 和 FCLK1) 的一个, 全局快时钟信号分布到每个宏单元触发器, 具有短的延时和小的畸变。

非同步置位和复位乘积项清除时钟运行, 如果非同步输入同时起作用, 复位代替置位。

除了驱动芯片的输出缓冲器, 宏单元输出被布线返回作为 UIM 的输入, 一个专用乘积项可以配置成输出缓冲器的输出使能, 和/或反馈到 UIM, 如果它被配置晨控置 UIM 反馈, 输出使能乘积项迫使 UIM 反馈线在宏单元输出失效时变高。

8. I/O 块

宏单元直接通过三态输出缓冲器驱动芯片输出, 每个由上面提到的输出乘积项单独控制, 宏单元输出可被倒相, 附加的配置选择允许输出永久失去能力, 两个专用的 FastOE 输入也可以配置来控制任何的芯片输出, 替代或与单独输出使能乘积项在一起, 见图 5.10。

输入到芯片的每个信号被接到一个作为直通、锁存或寄存来配置的可编程输入结构, 锁存和触发可以利用两个快时钟信号的一个作为锁存使能或时钟, 两个快时钟信号是 FCLK0 和 FCLK1 或 FCLK2 的全局选择, 锁存当快时钟为高时是透明的, 而触发时钟在快时钟的上升沿, 触发包括一个低有效时钟使能, 当它为高时, 保存触发的当前状态和对输入信号的固有响应, 时钟使能源是两个全局时钟使能信号 (CE0 和 CE1) 的一个, 附加的配置选择是每个输入信号的极性变化。

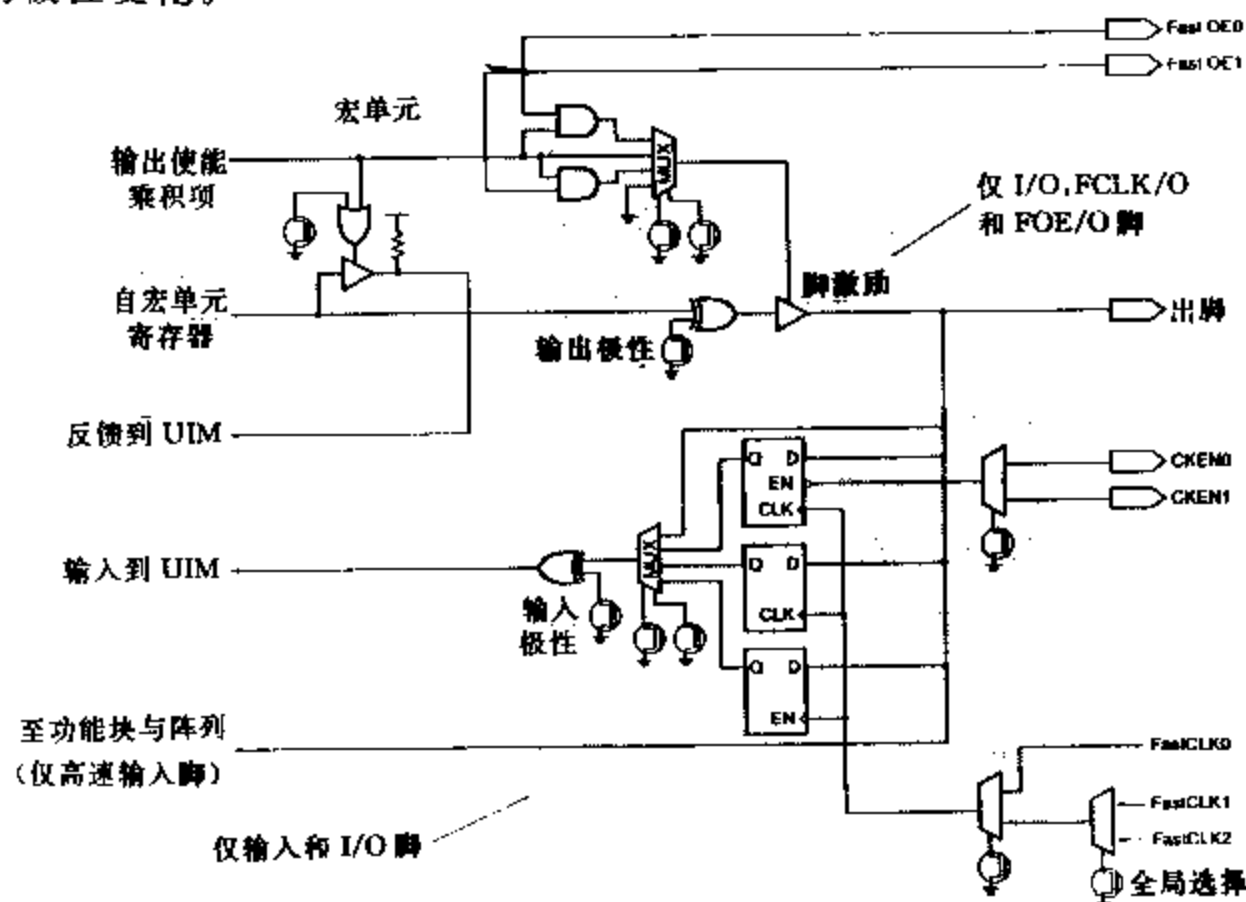


图 5.10 I/O 原理图

9. 通用互连矩阵 (UIM)

UIM 接收来自宏单元反馈线, 双向 I/O 腿和专用输入腿的输入, 起无限制的转换开关的作用, UIM 产生 21 个输出信号到每个高密功能块和 24 个输出信号到每个高速功能块。

任何 UIM 输入可以被编程接到任何 UIM 输出, 通过互连矩阵的延时是固定的, 不考虑

网线距离和复杂性,扇出或扇入,所以任何 UIM 输入可以驱动 UIM 输出,甚至多个输出,延时是固定的。

当多个输入被编程连接到相同的输出,这个输出产生输入信号的逻辑与,选择相应的在输入腿上,宏单元与阵列输入等的信号反相,这个与逻辑也可以被用来实现宽位与非,或,或非函数。

由输出使能乘积项禁止的宏单元反馈信号表示输入到 UIM 为高,编程几个这样的宏单元输出到相同的 UIM 输出,就可模拟一个三态总线,如果宏单元输出的一个被使能,UIM 输出假定它的电平。

§ 5.4.3 应用特性

1. 3.3V 或5V 接口配置

XC7300的器件可以用于两种不同供电电压:3.3V 和5V 的系统中。每个 XC7300器件有分别的 V_{CC}接到内部的逻辑和输入缓冲器(V_{CC}INT),以及 I/O 驱动器(V_{CC}IO)。V_{CC}INT必须总是接到5V 电源,但是 V_{CC}IO可以接到3.3V 或5V,取决于输出接口的要求。当 V_{CC}IO接到5V 时,输入门限是 TTL 电平,因此与3.3V 和5V 的逻辑兼容,输出高电平也是 TTL 兼容的。当 V_{CC}IO接到3.3V 时,输入门限仍是 TTL 电平,输出上拉到3.3V 上。这使得 XC7300 对直接接到3.3V 的元件是理想的。此外,输出结构设计成以使 I/O 可以安全地接到混合的 3.3V 和5V 总线。

2. 加电特性

和许多灵活性很高的 EPLD 一样,XC7300器件在器件加电时经历一个短的内部初始化的时序,在此期间,输出保持三态,而器件被配置形成它的内部 EPROM 阵列模式,并且所有的寄存器被初始化。注意到除了在器件初始化时短的延时外,这个运行过程对用户是完全透明的。初始化过程一般持续200 μ s,在所有情况下不超过300 μ s。

为了附加的灵活性,提供一个低有效的主复位腿(Master Reset),以使 EPLD 即使在加电后可以被重新初始化,它允许 EPLD 和系统内其它器件一起初始化。当它变为低时,所有的输出变成三态,并开始初始化的时序。当它返回到高时,输出变成被使能,器件准备运行。如果不需要此灵活性,把主复位腿简单地接到器件的 V_{CC}INT。

在初始化时序期间,所有的高速功能块宏单元寄存器和输入寄存器或锁存器被预加载到高电平,用缺省值,高密功能块宏单元寄存器被预加载到低电平。高密功能块宏单元寄存器预加载状态可以由用户来选择。注意到因为器件输入对初始化的局部可以起作用,关键的输入,如时钟,复位或置位应该在初始化期间保持不起作用,以确保预加载寄存器在运行前维持正确的状态。

3. 功率管理

当 EPLD 变得越复杂和系统的时钟频率上升时,片内功率耗散的控制变得越重要。XC7300功率管理方案允许一个设计的非速度临界部分以减小的功率来运行,整个功率耗散常常显著减少,因为在大多数系统中,仅有很少部分是速度临界的。宏单元可以单独地规定为高性能或低功耗运行,只要在逻辑原理图中加属性或加说明语句到行为描述中,为了进一步节省功率,无用的宏单元自动地被关掉。

4. 擦除特性

在带窗口的封装中, EPROM 阵列的内容可以在接近4000Å波长的紫外光照射下擦除掉。当器件放置在12000μW/cm²功率的紫外灯1英寸内, 推荐的擦除时间大约是1小时, 为防止无意识的擦除, 在器件的窗口上贴遮光的标签。当器件在高亮度的紫外灯照射太长时间, 可能出现永久的破坏, XC7300 EPLD 可被照射而无损坏的最大积累剂量是7000W/cm², 或在12000μW/cm²下大约一周时间。

6. 设计保密性

7. 大量产品的编程

8. 定时模式

定时模型是根据 XC7300结构的固定的内部延时,它由四部分组成:I/O 块,通用互连矩阵,高速功能块和高密功能块,定时模型识别内部的延时通道和它们与交流特性的关系,利用这个模型和交流特性,设计者可以容易地计算特定的 EPLD 的定时信息

图5.11 XC7300系列定时模式

第六章 可编程门阵列

§ 6.1 概述

Xilinx 的 FPGA 有 XC2000, XC3000 和 XC4000 三个系列。每个 Xilinx 的 FPGA 执行定制集成电路的功能, 类似于门阵列, 但是 Xilinx 的 FPGA 是用户可编程的, 甚至在系统中是可再编程的。

与传统的逻辑电路, PAL, EPLD, 甚至门阵列在结构上不同, 所有 Xilinx 的 FPGA 是在一个小的查找表 (16×1 ROM) 中实现组合逻辑的功能; 每个查找表的输出或者馈送到触发器的 D 输入端, 或者驱动其它的逻辑和 I/O 块。每个器件包括相同逻辑功能块的矩阵, 通常是方形的, 从 XC2064 的 8×8 矩阵到 XC4025 的 32×32 矩阵, 短的和长的金属线水平或垂直地在这些功能块之间穿过, 有选择地互连它们或把它们连接到 I/O 块上。这种功能块结构含有很多寄存器和能实现高达多个变量的任何函数的函数发生器, 从 XC2000 系列的四输入变量, XC3000/3100 系列的五输入变量, 到 XC4000 系列可实现最多九输入变量的任何逻辑函数, 这些功能块全都有相同的速度。丰富的片内缓冲使得块延时不敏感于互连结构的负载, 但全部互连延时是与布线有关的, 当设计处于临界性能时, 必须分析互连的延时。

时钟线有很好的缓冲, 从器件的一个角到另一个角, 甚至通过最大的器件可以小于 2ns 的畸变驱动所有触发器。当指定的时钟线被利用时, 用户不用担心时钟的负荷或时钟延时的平衡, 或片内保持时间的问题, 这些指定的时钟线在 XC4000 系列器件中有八条, 而在其它器件中是两条。除 XC2000 系列之外, Xilinx 的 FPGA 器件可以实现内部双向总线, XC4000 系列器件有专用的快速进位电路, 以改善加法器, 减法器, 比较器, 累加器和同步计数器的效率和速度, 每个出腿也支持边界扫描。

几乎所有器件的出腿可作为双向用户 I/O 来用, 只有 4—24 个供电连接 (V_{cc} 和 GND) 和少数专用于配置过程的出腿是例外。在每个系列内所有输入和输出有相同的电特性, 但输出电流能力随不同系列而改变, 见表 7.1。

XC2000/XC3000/XC3100 的输入可以整体地被编程为 TTL 输入门限或 CMOS 门限。XC4000 有固定的 TTL 输入门限。所有的输入有 $100\text{—}200\text{mV}$ 的滞后。

所有 Xilinx 的 FPGA 有全局非同步复位输入作用于所有器件的触发器。在 XC4000 系列器件中, 任何出腿可以配置成复位输出, 而其它系列复位是专用出腿。

因为所有 Xilinx 的 FPGA 利用了 CMOS—SRAM 的工艺, 它们的静态或备用的功耗是十分低的, 对于 XC2000/XC3000 器件仅为几 mW , 对 XC3100 最大为 25mW , 对 XC4000 最大为 50mW 。运行的功耗总的是动态的, 正比于输入, 输出和内部节点的转换速率, 典型的数值是在 100mW 到 2W 之间, 取决于器件的规模。

XC2000/XC3000 的器件允许供电不足, 其配置可以由一个电压大于 2.3V 的电池来保持, 电流消耗仅为几毫安, 器件使所有输出为三态, 所有输入不管, 且复位它的触发器, 可保持它的配置。XC3000/XC3100/XC4000 的器件连续地监视 V_{cc} , 并当检测到 V_{cc} 低于 3V 而失效, 然后器件使所有的输出为三态, 准备重新配置。XC2000 的器件需要外部监视是否在没有

一切方式接地时有任何 V_{CC} 显著降落的危险。

利用 FPGA 实现可编程 ASIC 时,设计通常用电子设计自动化软件从方框图和原理图画图开始,这些工具都有与 Xilinx 开发工具 XACT 的接口,可以在 PC 机或工作站上运行,在原理图输入或基于文本的输入之后,设计被自动转换成 Xilinx 的网表格式(XNF),XACT 软件首先把设计分割进逻辑功能块,对每个功能块找出接近最佳的布局,最后选择互连的布线,这个分割,布局和布线的过程(PPR)自动地进行,但用户也可以施加专门的限制,或者利用图形的设计编辑器(XDE)选择性地编辑设计的临界部份,因此用户在完全的自动实现和布图过程中详尽地介入之间可有很大范围的选择。

一旦设计完成,生成 LCA 的文件,由此文件可以产生串行的位流文件,然后用户从几种加载方法中选一种将此位流文件加载到 Xilinx 的 FPGA 器件中,它储存在排列成类似长移位寄存器的锁存器中,这些锁存器中的数据内容把 FPGA 配置成预定的数字功能,此配置位的位数随器件的不同而改变,由最小的器件 XC2064 的 12038 位到目前可用的最大器件 XC4025 的 422168 位。多个 LCA 器件可以连成菊花链,并用共同的串接起来的位流文件来配置,器件的应用不改变配置位的位数。

在器件内部,这些配置位控制或定义组合电路,触发器,互连结构和 I/O 的缓冲器。加电时器件等待 V_{CC} 达到可接受的电平,然后清除配置存储器,保持所有的内部触发器复位,除去其上拉电阻激活的输出,几乎所有的输出为三态,之后器件开始配置,或者作为主导器件,定时串行的 PROM 以接收串行的位流,或者作为从属器件,接收外部的时钟信号和来自外部源的串行或并行数据。

Xilinx 的串行 PROM 是最简单的配置器件的方式,仅用到四个器件出腿,一般的配置时间是每位约 $1\mu s$,但有方法加快十倍来减少配置时间,因此配置要化费几到几百毫秒的时间。Xilinx 的串行 PROM XC17000 系列有从 18K 到 128K 位的规模,PROM 也可连成菊花链来储存更长的位流。LCA 器件也可以字节数据来配置,或者由标准的 PROM,或者由微处理器提供数据。LCA 器件直接驱动 PROM 地址,或者与微处理器像一般的外设那样进行握手,字节数据立即变换成内部的串行位流,由内部配置时钟(CCLK)定时,所以并行配置模式并不比串行模式更快。

用户可以在任何时候重新配置器件,只要将 DONE 腿电平拉低来激励新的配置程序,在此过程中,所有不用于配置的输出都为三态,部分地重新配置是不可能的。器件被编程之后,配置的“移位寄存器”可以串行地读回,并不中断器件的运行,XC4000 器件包括同步地瞬时转移所有用户寄存器信息进配置寄存器,这就把电路内仿真能力加到读回功能上。

下面分别介绍 XC4000, XC3000 和 XC2000 三个系列。表 6.1 给出这三个系列的比较。§ 6.6 和 § 6.7 分别介绍最新的低成本高容量 XC5200 系列和细粒度的用于一次编程的 XC8100 系列。

§ 6.2 逻辑单元阵列—LCA

如前面第三章所介绍的,Xilinx 的 LCA 结构由三种可配置元素组成——周边的输入/输出块,核心的可配置逻辑块阵列及连线资源。IOB 提供内部逻辑阵列与器件封装管腿间的接口,CLB 实现用户指定的逻辑功能,连线资源在这些块间传送信号,贮存于内部静态存储单元中的配置程序决定逻辑功能及连接。配置数据在加电期间或允许重新编程时装入器件。

配置数据装入到内部存贮单元后,用户就能使用 LCA 器件了。LCA 器件既可以主动地从外部串行或并行 PROM 中读出它的配置数据(主模式),也可以把配置数据写入 LCA 器件(从属及外设模式)。逻辑单元阵列包含逻辑和控制信号以实现自动或无源的配置,配置的程序数据可以是以位串行的或以字节并行的,Xilinx 的 XACT 开发系统产生配置程序的位流用来配置 LCA 器件,存储器的加载过程是与用户的逻辑函数无关。

表6.1 Xilinx 三代 FPGA 的性能

		近似门数	最大 I/OB	触发器数	水平长线	配置数据位数	有效 CLB	方阵规模
XC2000	2064/L	1000	58	122	0	12038	64	8×8
	2018/L	2000	74	174	0	17878	100	10×10
XC3000/XC3100系列	30/3120/A	1800	64	256	16	14779	64	8×8
	30/3130/A	2700	80	360	20	22176	100	10×10
	30/3142/A	3700	96	480	24	30784	144	12×12
	30/3164/A	5500	120	688	32	46064	224	16×14
	30/3190/A	7500	144	928	40	64160	320	16×20
	3195	9000	176	1320	44	94984	484	22×22
XC4000系列	4002A	2000	64	256	2048	24	64	8×8
	4003A	3000	80	360	3200	30	100	10×10
	4003H	3000	160	200	3200	30	100	10×10
	4004A	4000	96	480	4608	36	144	12×12
	4005/A	5000	112	616	6272	42	196	14×14
	4005H	5000	192	392	6272	42	196	14×14
	4006	6000	128	768	8192	48	256	16×16
	4008	8000	144	936	10368	54	324	18×18
	4010/10D	10000	160	1120	12800	60	400	20×20
	4013	13000	192	1536	18432	72	576	24×24
	4020	20000	240	2280	28800	90	900	30×30
	4025	25000	256	2560	32768	96	1024	32×32

要说明的是逻辑单元阵列(LCA)是 Xilinx 为其发明的 FPGA 的注册商标,所以 LCA 仅仅应该用作形容词,如 LCA 器件,LCA 结构等。

Xilinx 的 FPGA 有以下三种可编程资源:

1. 可配置逻辑块(CLB)

在 LCA 器件的每一个系列(XC2000,XC3000,XC3100)中,可配置逻辑块阵列提供逻辑功能单元,LCA 器件中的逻辑就是在这些单元中构造的,CLB 以阵列方式布置在器件的中央。

在逻辑单元阵列中的 CLB 以它的水平行的位置和垂直行的位置来命名,作为其物理位

置名,如图3.3所示。

2. 输入/输出块(IOB)

LCA 中的输入/输出块(IOB)提供外部封装腿与内部逻辑块(CLB)间的接口。LCA 的布线资源用于 IOB 及 CLB 的输入和输出的连接。

每个 IOB 用它的封装腿数定义它的位置名,也如图3.3所示。

3. 连线资源

每个 LCA 器件都包括全局资源,这些资源可以以很小的畸变把时钟传送到 LCA 器件内的每一处。前面提到的 CLB 及 IOB 是用器件提供的布线资源连接起来的,当对设计布局布线时,Xilinx 软件(APR)选择最佳资源用于特殊信号类型。

在图3.3中也表示了在 CLB 的相邻的行和列之间留出的连线区域和 TBUF 的分布。

第一代 LCA 器件是 XC2000系列,在1985年引入,表征它的逻辑块是由能实现四输入布尔函数的组合函数发生器和一个存储单元组成,XC2000系列有两个型号,为800到1500门的复杂度。在1987年引入第二代的 XC3000系列,逻辑功能块扩展到实现更宽的布尔函数,并在每个逻辑块中增加第二个触发器,达到包含1300到10000个可用门,最高的可保证的触发频率达70到340MHz,等效的最大的系统时钟频率高达上百兆赫芝。第三代的 LCA 器件进一步以更强和更灵活的逻辑块扩展其结构,XC4000系列达到25000个可用门,支持高达50MHz 的系统时钟频率。

下面就 Xilinx 的三个系列,XC4000,XC3000/3100和 XC2000系列,分别对其 LCA 结构进行介绍,描述这些系列不同类型的可编程 LCB,可编程 I/O 和可编程布线资源。

§ 6.3 XC4000系列的 LCA

§ 6.3.1 XC4000系列的 CLB

图6.1表示 XC4000的可配置逻辑块(CLB)。可配逻辑块包含函数发生器,触发器和编程控制的多路开关。一个函数发生器是一个 N 输入 2^N 位存储器,实现它的输入的任何函数。用变址到存储器中的真值表可配置逻辑块包含的三个函数发生器,分别以 F,G 和 H 标志。两个第一级的输入函数发生器 F 和 G,每个实现四个变量的任何函数。这两个结果可以独立地从这个块输出。或者它们可以在 H 函数发生器组合成五输入的任何函数,或者高达九个输入的一些函数。这样允许某些高扇入函数。如九输入与,或及异或,而且在一个逻辑块中进行译码。可配置逻辑块亦包含两个触发器。这个触发器可以从函数发生器或者从外部信号取它们的输入,而它们的控制信号是共享的,即使在随机逻辑中,触发器也是十分普通的。靠直接地实现它们,而不是从函数发生器把它们构造出来,要求实现触发器的总面积减少,而改善了准稳性和控制的建立和保持时间。紧跟着函数发生器的触发器提供了强的流水能力,因为所有组合逻辑结果可以存贮在一寄存器中而没有引起附加的延迟。这个流水特性在高性能的数字信号处理应用中经常使用。

1. 快速进位逻辑

除了实现逻辑和提供存贮之外,XC4000包含专用硬件来加速加法器和计数器的进位通道,图6.2表示在算术模式中 CLB 的配置。利用这个特性,加法器和计数器变得十分快和有

效,占用最少数量的 CLB.CLB 的 F 和 G 函数发生器计算和,而专门的逻辑计算进位,专门的互连把进位信号布线到相邻的逻辑块。没有进位逻辑,单个位的算术可以在一个 CLB 中建立,用一个函数发生器为和,另一个为进位链。但是,算术运算的速度受进位链的速度制约。专门的进位逻辑实际上加速了运算,而加倍了算术计算的密度。这个进位逻辑可以配置来建立任意长度的高速序列发生器,计数器,相加器,相减器或加减器。

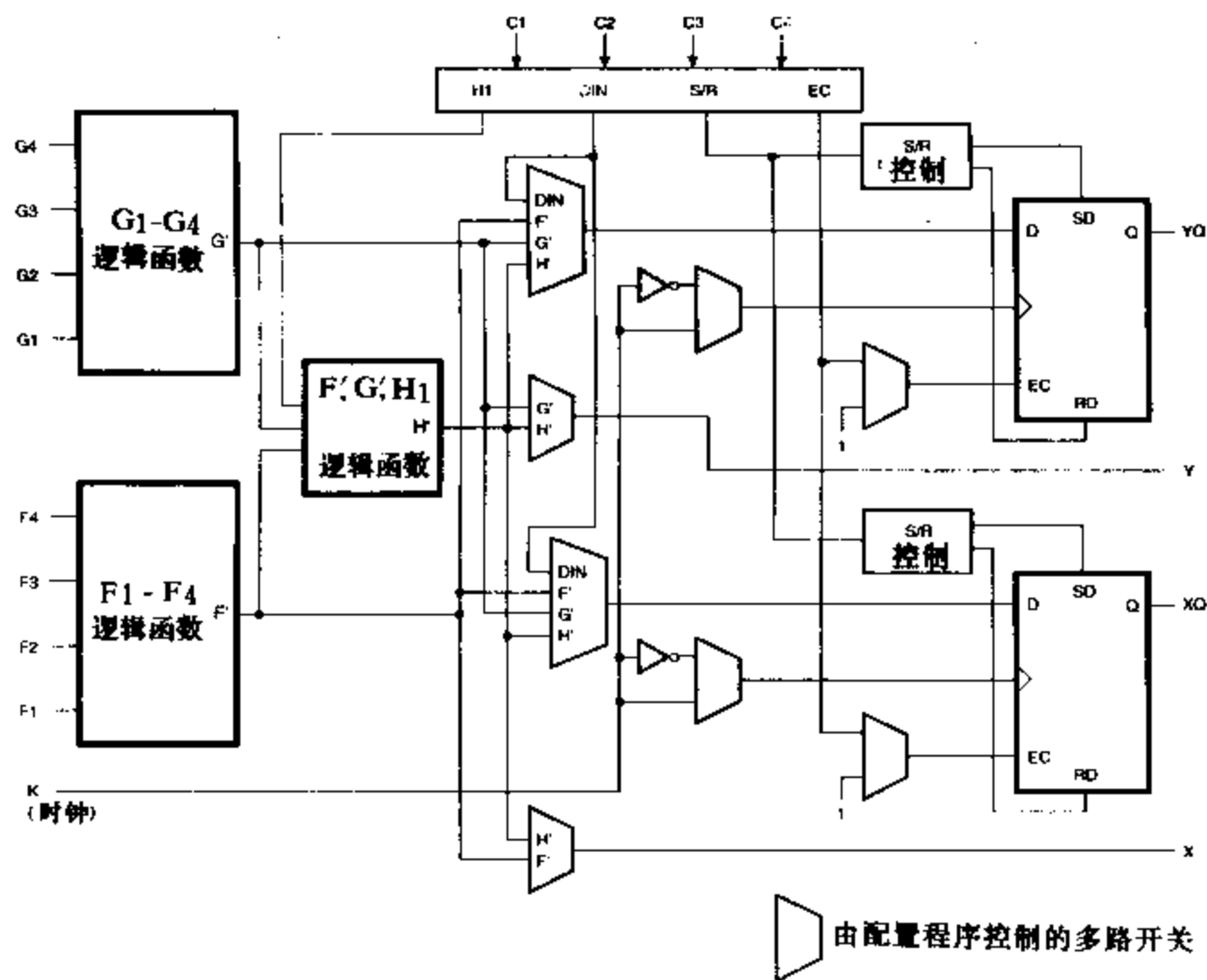


图6.1 XC4000系列 CLB 方块图

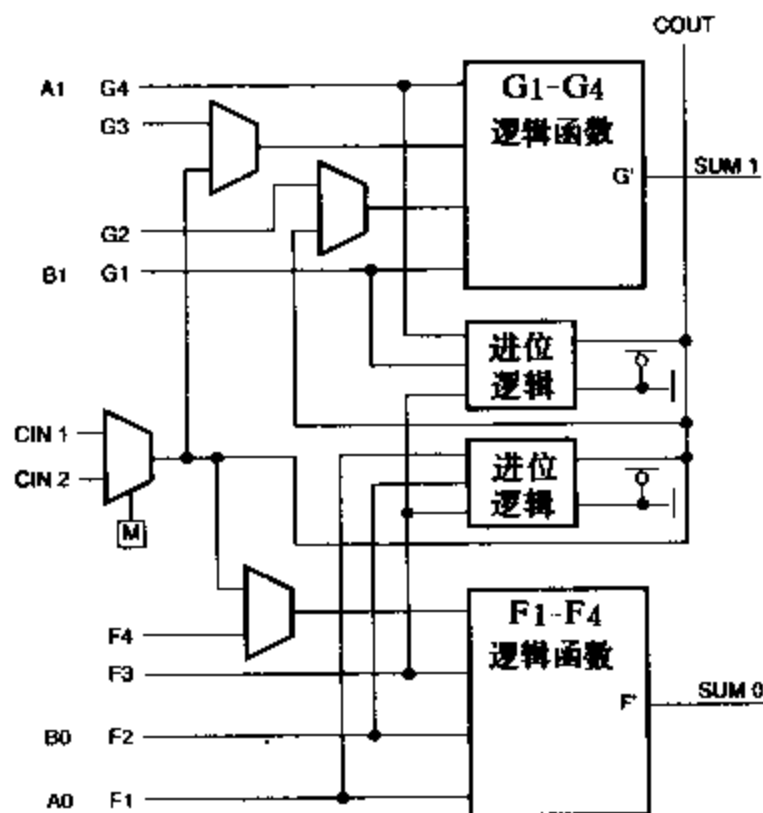


图6.2 XC4000 CLB 配置成 RAM

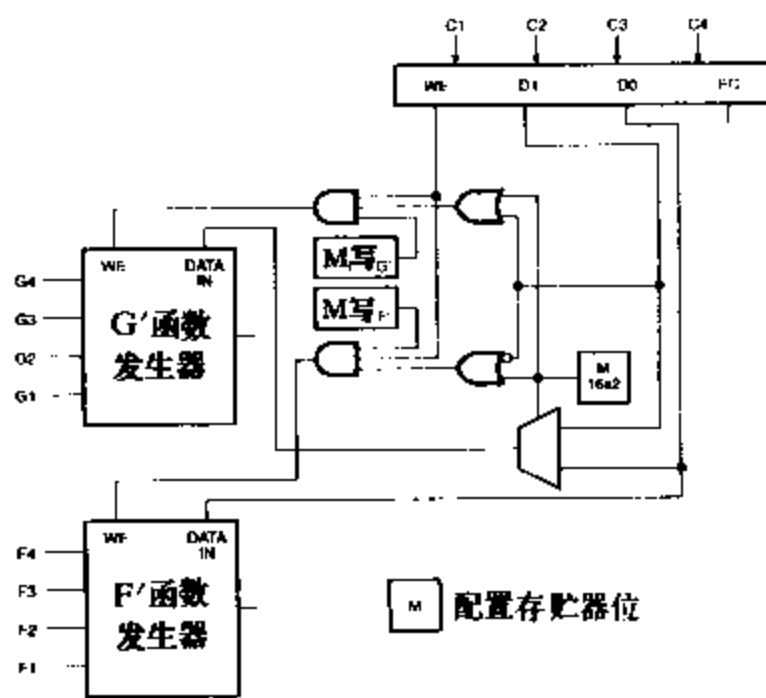


图6.3 XC4000 CLB 配置成算术单元

当专用逻辑和互连被用来优化进位通道,函数发生器用来形成由操作数和进位形成的和,按此方式,算术的两位(或当要求壹位时)可以在每个 CLB 中实现,因为专门的进位逻辑可以按接近40种不同的方式来配置,CLB 可以连成各种算术功能,进位传播从 CLB 的一列向上或向下。

专门的进位逻辑是通过硬宏单元来做到的,进位逻辑硬宏单元在 Xilinx 的库中是有效的,或者可以由用户用 HMGEN 程序来定义。HMGEN 程序包包括描述如何利用 HMGEN 程序和专门的进位逻辑的文档。

2. 片内 RAM

XC4000片内 RAM 显著地减少数据存储的成本,利用这个特性,高达64位的数据可以存储在单个 CLB 中,否则在它的触发器中仅仅存储两位。在 XC4000的 CLB 中,在芯片上 write enable 和 data-in 信号的控制下,写入 F 和 G 函数发生器的 RAM 中是可能的。如图6.3所示,一个 CLB 可以实现32位存储器,或者 32×1 或 16×2 的配置。这个存储器的能力对于寄存器,Cache(高速缓存)和缓冲器是有用的。

任何 CLB 可以作为 RAM 来配置,在 RAM 模式,F 和 G 函数发生器的查找表变成可写的,图6.3对于一个 16×2 位的 RAM,F 和 G 函数发生器被分别利用,对于 32×1 位的 RAM,它们在 H 函数发生器中组合。F 和 G 输入腿起存储器地址线的作用,如它们在非 RAM 模式中所做的那样,但是其它的 CLB 控制腿被重新定义。对于 16×2 bit 的 RAM,DIN 和 HI 变成两个数据输入,而在 32×1 bit 配置中,DIN 是单个数据输入,D1是第5个地址线,在两种情况下,S/R 写使能(WE)输入。

16×2 bit 模式,读数据在 F 和 G 函数发生器的输出端是有效的。

32×1 bit 模式,在 H 函数发生器输出端是有效的。

象在非 RAM 模式中一样,这些函数发生器输出驱动 X 和 Y 输出腿,或者它们可以在触发器中寄存。某些非 RAM 的功能在 CLB 作为 RAM 配置时仍保留,在 16×2 模式,H 函数发生器可以用来实现两个 RAM 输出的任意函数。在 32×1 模式,写和读数据可以在触发器中得到,因为触发器已经在 DIN 腿上得到 RAM 输入数据。

XC4000的 RAM 功能与通常为55ns 或更长周期时间的单片 SRAM 器件相比是极快的,在那些更慢的器件中,控制信号为1ns 是可以容忍的,但是在 XC4000RAM 的中不是这种情况,当周期时间少于10ns 时,短于1ns 的 WE 脉冲容易识别,设计好的控制电路是最重要的。用 XC4000的 SRAM 设计类似于很快的单片 SRAM(<25 ns)的设计,许多因素,如互连的延迟,在较慢的单片 SRAM 设计中可以放心的不管,但此时变得临界了。

§ 6.3.2 XC4000系列的 I/OB

图6.4表示了 XC4000输入/输出模块(IOB)的细节,从芯片输出的信号在输出之前可以被寄存和用分别的控制信号使能。输出可以选择上拉或下拉。在焊盘上的驱动可以配置成或者高速的模式或者低功率的模式。

来自焊盘的输入可送到芯片的内部以直接的或寄存的方式或者二者兼有之。由于允许二者兼有,I/O 模块可以分解多路开关的外部信号,如地址/数据总线,把地址存进 IO 触发器,而把数据直接馈送到网线上。

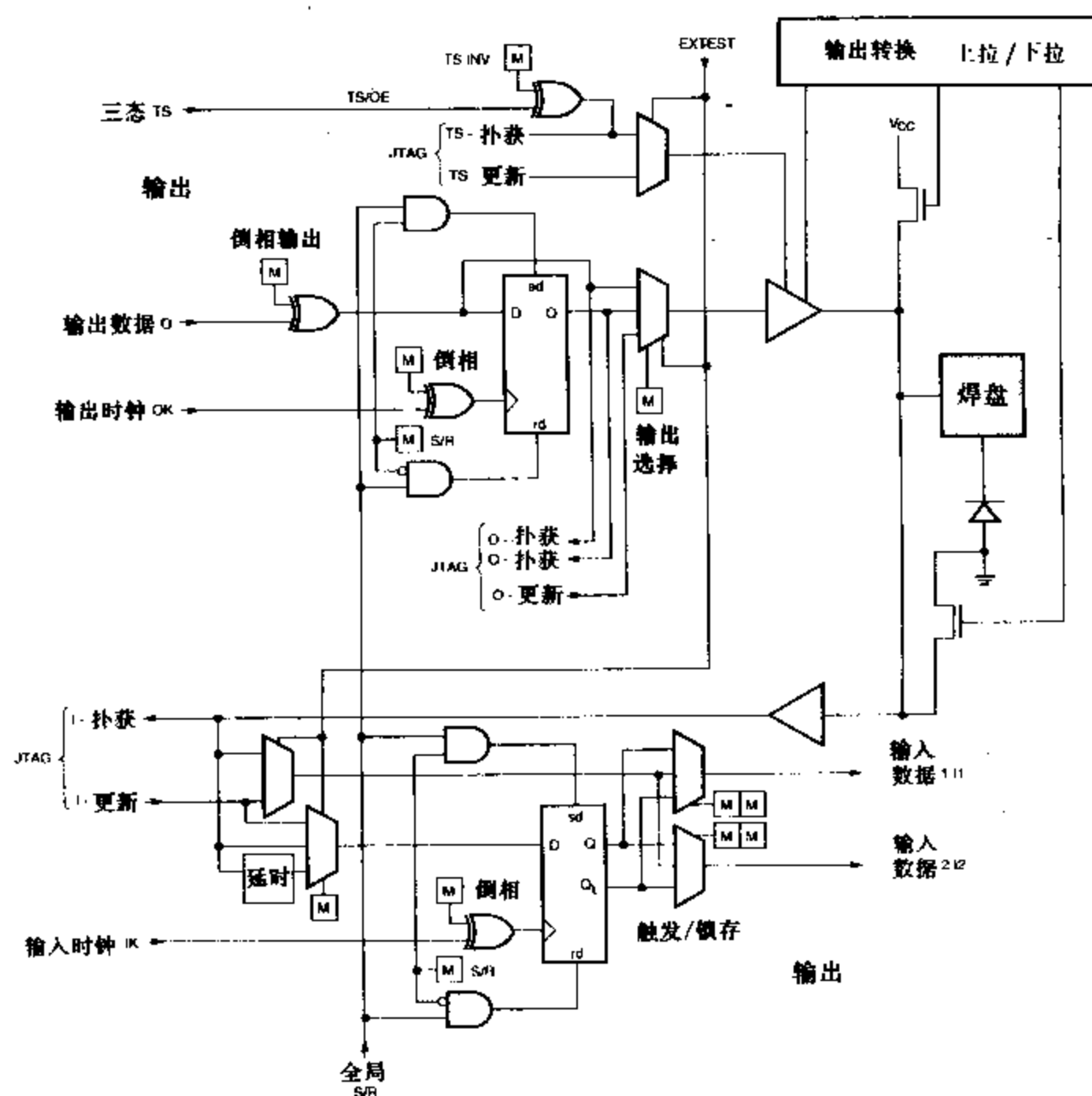


图6.4 XC4000的I/O原理图

1. 沿边宽译码器

为了进一步方便总线接口,输入可驱动在网线中构造的宽的译码器,由于构造十分有效,运行十分快,以便快速的识别地址。译码器允许类似地址的芯片使能,因为有些时候需要从大量的位数中译码特定的数值,例如,当译码规定的微处理器存储器的地址,为了便利这样的译码,沿着XC4000器件的每个边提供了专门的宽译码器的功能,如图6.5,这些沿边宽译码器由CLB分开,不消耗CLB的资源。

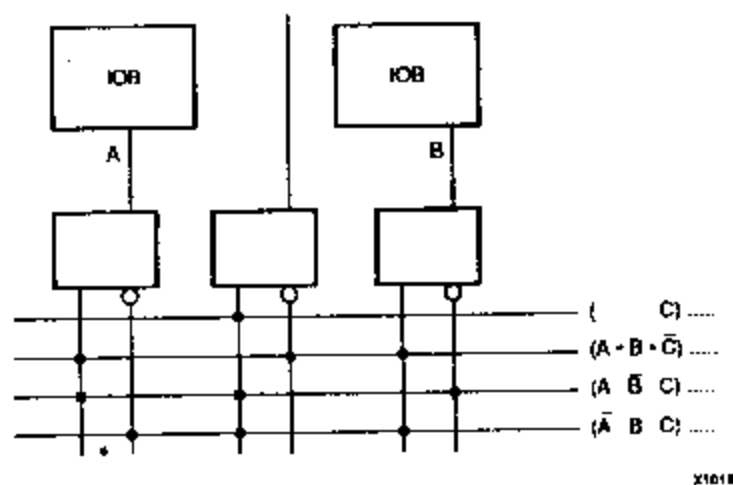


图6.5 XC4000系列的沿边宽译码器

在XC4000系列芯片的每一边,有四个译码器,共享公共的一组输入,对于每个CLB的行或列有三个译码器的输入,两相邻的I/O块每个有一个输入,再加局部互连的一个输入。当译码器共享输入时,在相同的边译码不同组的位数是可能的,因为所有的输入并不需要被

每个译码器利用。因此在相同的边对位数拆开的组译码也是可能的。例如,一个 XC4000 有 20×20 的 CLB 阵列,每个边有四个宽译码器共享 60 个输入。在一个边的译码器可以译码一个规定字节的地址和来自 32 位微处理器总线的规定的字的地址,并在相同的时间,译码来自内部的 16 位总线的两个规定的数值。

宽译码器作为线与门实现,在每端的电阻性上拉引起输出变为高,当所有的输入满足它们规定的条件,任何到译码器的输入破坏了满足它的规定的条件(高到低)引起输出被下拉。每个译码器可以在它的中心分裂成两个一半规模的译码器,所以有高达 32 个译码器是可用的,专门的译码器仅仅应该用于译码十个或更多的输入,因为九个或更少的输入可以利用单个 CLB 更有效地被译码。注意 XC4000A 的器件每边有两个宽译码器,利用符号 DECOOE4 DECOOE8 等在原理图中规定宽译码器的使用。

2. I/O 块的寄存器

位于 I/O 块中的寄存器和锁存器是一个常被忽视的资源,它们对同步输入的输出信号是理想的,而且可以对内部信号提供附加的存储,利用 I/O 块的寄存器可以显著地减少 CLB 触发器的利用,也可以减少布线的冲突。I/O 块寄存器或锁存器的利用在设计输入期间被规定,它们用原理图符号 OUTFF,INFF 和 INLAT 来表示。

§ 6.3.3 XC4000 系列的布线资源

XC4000 的设计有三种主要的为不同应用的布线资源,由它们的线段的长度来区分:

1. 通用单长度线,双长度线;2. 长线;3. 全局网

布线通道的数目是随阵列大小而改变的,图 6.6 和图 6.7 中的布线通道的数目仅为了图解所用。XC4000 芯片上的布线资源用不同数量的逻辑来改变,较大的芯片有较大的布线要求,所以有较大数量的导线。XC4005 有 8 根单长度线,4 根双长度线,6 根长线和四个附加时钟线。

1. 通用单长度线,双长度线

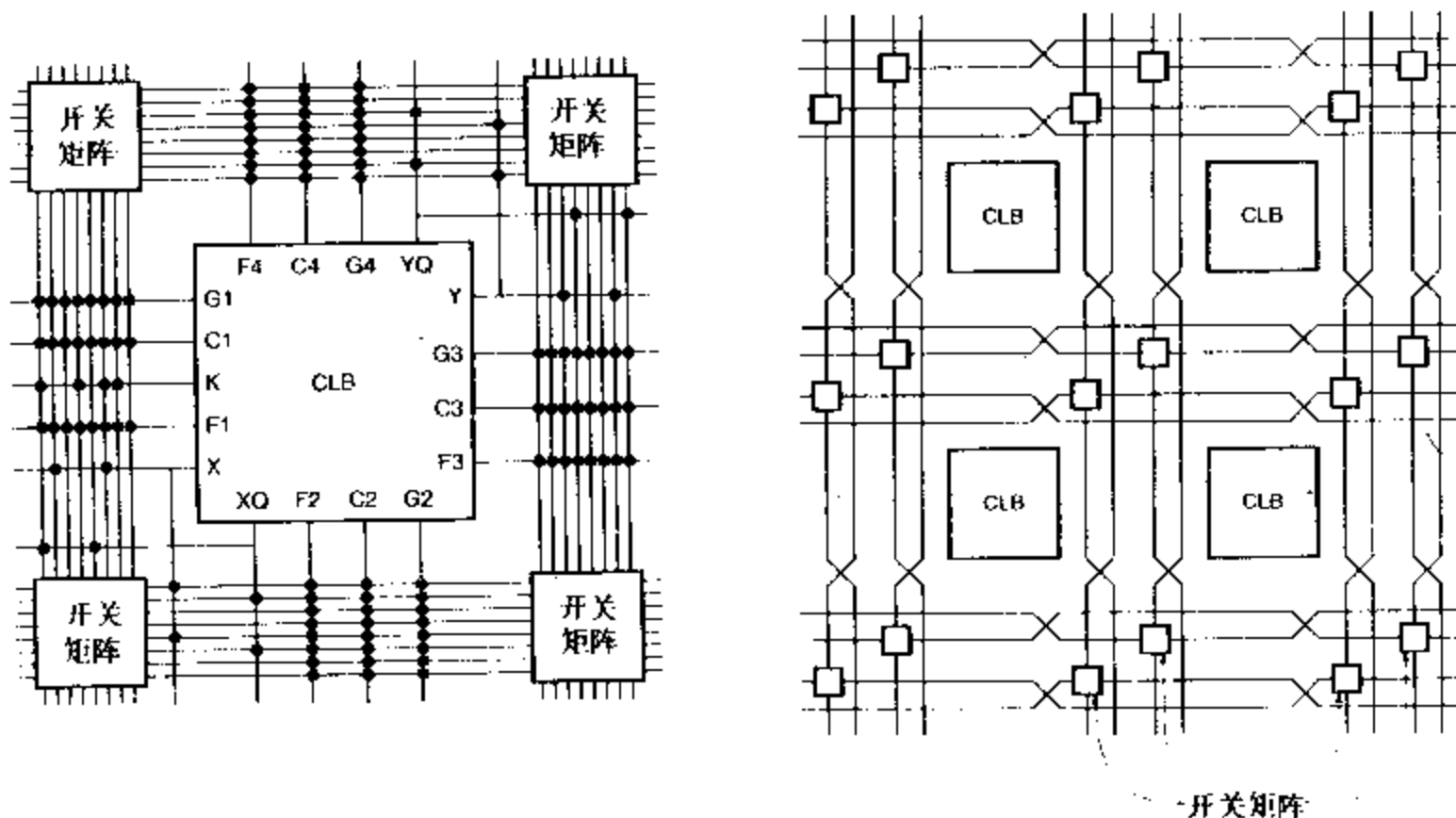


图 6.6 (a)CLB 接到相邻的单长度线 (b)双长度线

通用单长度线,双长度线提供相邻功能块之间的快速布线和复杂布线的最高灵活性,但是它们每通过一次开关矩阵增加一次延时。布线段被设计成平均布线通道的最小电阻和电容,因而显著改善了性能。图6.6表示了通用布线的全局图。

布线包括单长度线布线从一个开关盒连到下一个,在开关盒的每个线段的每一端上,有三个传输晶体管开关,控制这一线段从开关盒的其它三个方向连接到其它线段,如图6.6(a)所示。

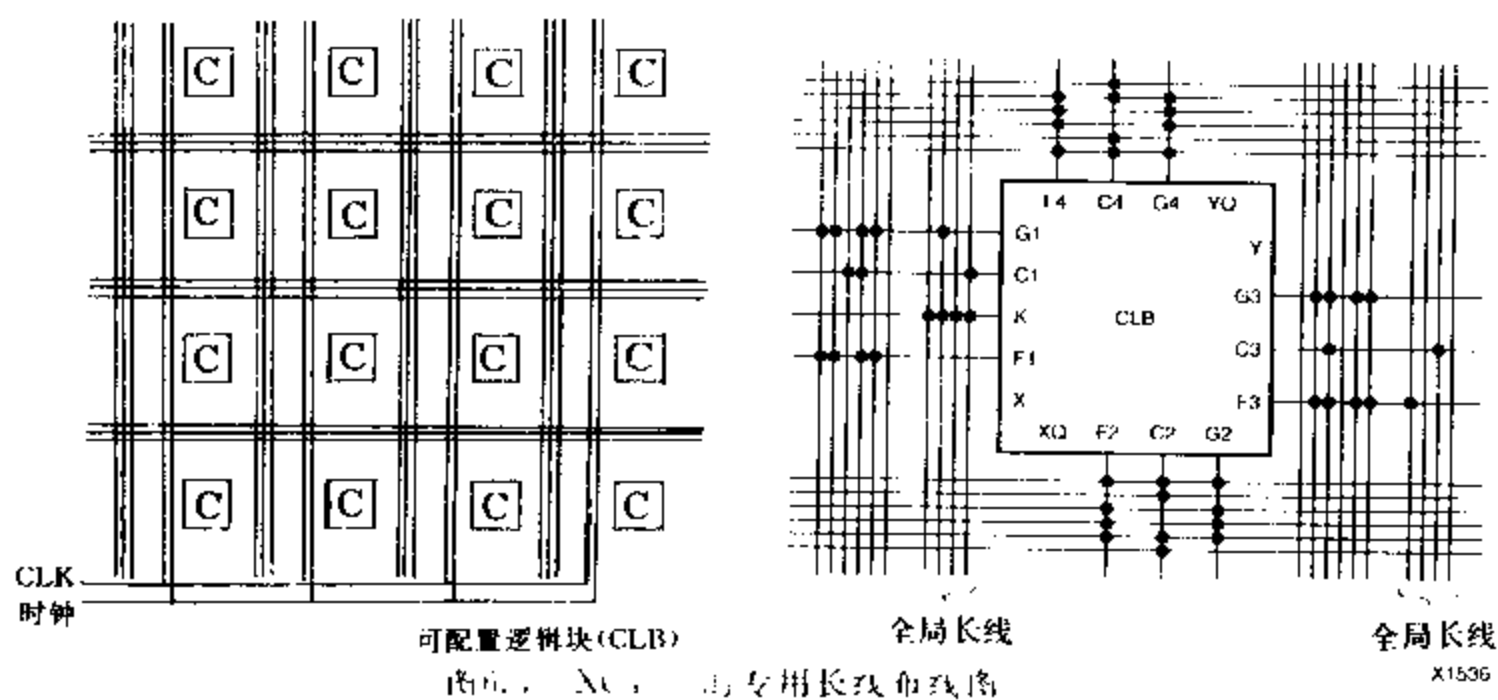
如果单长度线是相当的多,它们在连接芯片上所有逻辑块是足够的。但是提供专门的布线资源在性能上有显著的优点。图6.6(b)表示了一些旁路可选开关盒的布线段,即在进入开关矩阵之前通过两个 CLB 的双长度线。因为比起这些线段的长度,布线的延迟与信号通过线段的数目的关系更密切,这些双长度线允许一个信号在相同的时间内通行两倍的距离,或者等效地以一半的时间在给定的距离上通行。双长度线可为关键速度的通路使用。

2. 长线和三态缓冲器

一般的互连仍然导致时钟,寄存控制和其它高扇出信号的显著扭曲,布线结构包括附加长度的布线资源,它伸展到芯片的整个高度和宽度以支持这些信号(图6.7)。这些长线有两种类型,主要类型是在每一行逻辑块的通用高扇出/高速布线,两根水平长线可以通过三态缓冲器连接在芯片上提供三态总线。

图6.7中两根垂直长线是标志为时钟布线,它们由专门的高驱动能力的时钟缓冲器驱动,并通过芯片的核心被布线以消除时钟扭曲。XC4000包含8条可以从芯片上或芯片外出发的时钟线,并通过低扭曲专门布线通到所有的逻辑模块。专用时钟缓冲器和电路实际上消除了芯片上的时钟扭曲。

为了简化芯片内多路开关总线,XC4000结构包括可以由三态缓冲器(TBUF)驱动的长线,相邻于每个 CLB 放置的两个 TBUF 直接驱动 CLB 之上和之下的水平长线,如图6.8所示。



TBUF 数据输入可以方便地由关联的 CLB 的输出驱动,但是也可以由任何其它处来驱动,任何信号可以用来使能 TBUF,当把一个使能布线在垂直长线上,它可以用来选择列向函数来驱动水平总线。附加的 TBUF 位于阵列的左边和右边的 I/O 块的附近,这些允许 I/O 块包含在多路开关的总线中,因此可以扩充一个外部的双向总线到芯片上。

TBUF 长线也可以用来实现线与功能,当不被驱动时,在长线的每边上选择的电阻性上

拉使它变高,在它的输入有逻辑低使能任何 TBUF 将引起长线变为低,因此产生使能信号的线与。TBUF 长线是有用的布线资源,应该节省地使用,具有十六个或较少输入的多路开关和与门常常可以利用 CLB 更有效地实现,所以节省了长线资源,TBUF 不适合于具有四个或更少输入的多路开关。

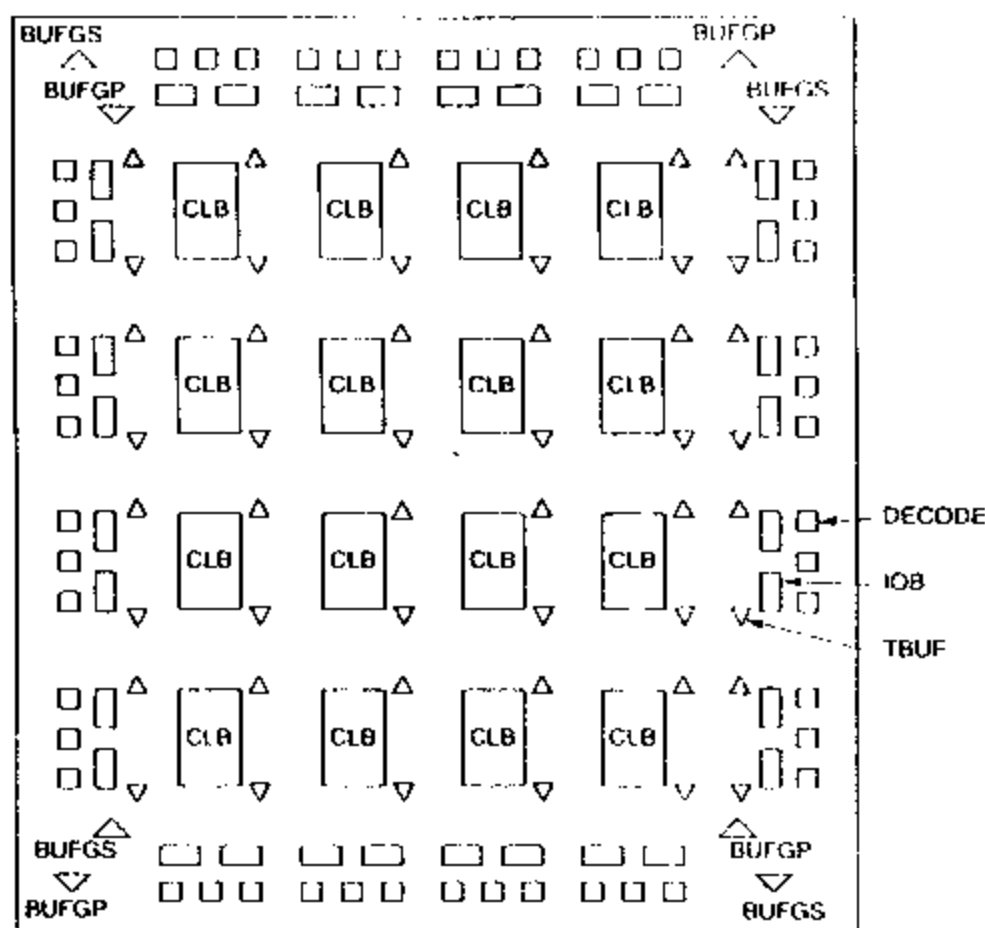


图6.8 XC4000的 TBUF 组织

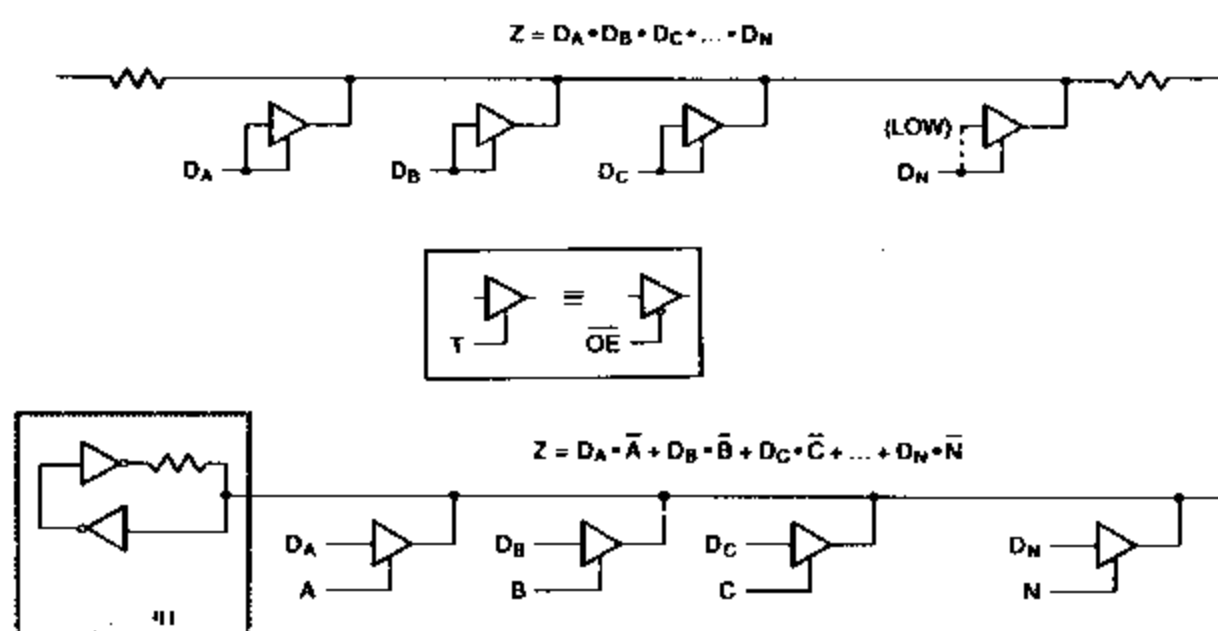


图6.9 TBUF 驱动实现线与和多路开关

3. 全局线和全局时钟缓冲器

八个全局线贯穿 XC4000的器件,可能达到每个 CLB,如图6.10所示,这些时钟线对于时钟和其它时间临界或高扇出的控制信号的分配是优化的。

八条线的四条是主要的全局线,提供最小的延迟产可忽略的扭曲,其它四条是次要的全局线,由于更重的加载,次要的全局线引入略微较长的延迟,约1ns 多,以及某些附加的扭

曲。主要的和次要的全局线是分别由 BUFG 和 BUFGS 缓冲器驱动,这两者可以直接连接到焊盘,在原理图中利用相应的缓冲器驱动所需的信号规定一个全局线的使用,BUFG 和 BUFGS 符号是在 Xilinx 的库中。

在每列的中心的开关矩阵连接八个全局缓冲器到用于分配在列内的全局信号的四条垂直线,图 6.10 表示了开关矩阵之一,每个垂直线可以用仅仅一个 BUFGP 来驱动,每个 BUFGP 驱动不同的垂直线,另一方面,BUFGS 可以每个驱动任何或所有的垂直线。因而,BUFGS 在它们的布线中是更多的灵活性,当布线非时钟的全局信号时,这个灵活性特别有用,因为垂直线已限于连接到非时钟的 CLB 腿,能够驱动任何或甚至由单个缓冲器来的多个垂直线是极大的优点。

布线非时钟的全局信号亦由于利用对时钟分配的 BUFGS 而变益,BUFGP 被限于时钟布局到每列中的相同垂直线,但用 BUFGS,任何垂直线可以用于时钟,可能释放一个垂直线为非时钟布线,无论何时定时和扭曲被允许,BUFGS 应该用于分配全局信号。在一个系统中时钟数目应该是最小的。因为仅仅四个时钟在一列 CLB 中可以做到有用,大量的时钟对 CLB 的布局会施加大量的影响,实际上,应该避免时钟,除非绝对必要,除了消耗全局线之外,选通逻辑引入不可控制的时钟扭曲,和潜在的时钟误操作,这两种情况可以引起系统的失效。当定时不要求时利用最小数目的时钟和不使能触发器是更好的,时钟使能信号常常可以在局部互连或规则长线上布线,因为它们不是关键的扭曲。

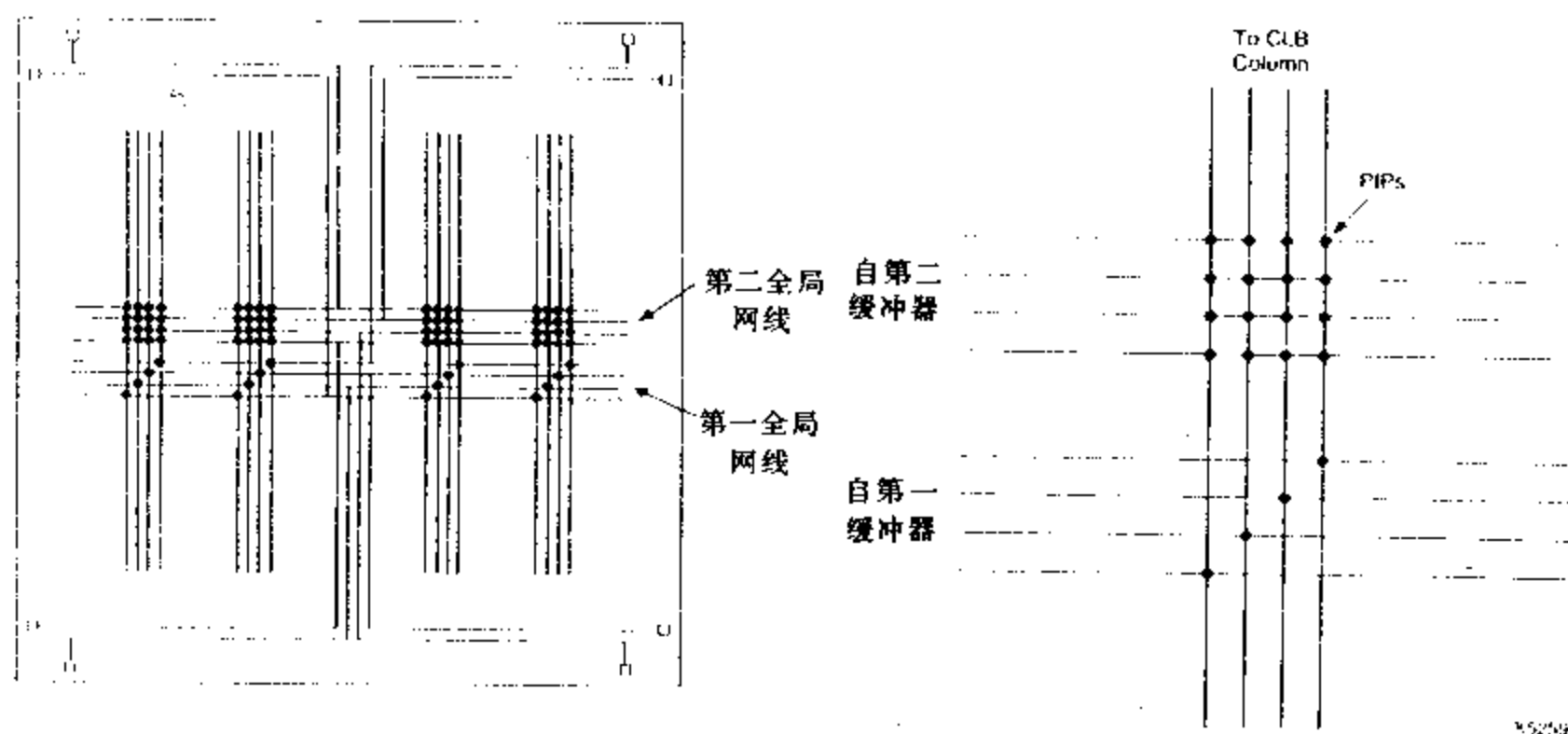


图 6.10 (a)XC4000 系列的全局网分布

图 6.10 (b)XC4000 系列全局网互连结构

4. 数据通道应用

FPGA 用通用逻辑结构实现逻辑,用可配置的通用网线实现连接。在最新一代 FPGA 中,这些特性被偏向于实现随机逻辑。但是大门数的设计不完全是由随机逻辑组成的,它们经常包含数据通道和存储器块。虽然瞄准一般随机逻辑的 FPGA 可以实现这些专门的功能,但是专门的逻辑和互连结构极大地改善了系统的规模和速度,以致可以集成进一个 FPGA 中。Xilinx 的 XC4000 系列包含这样的瞄准数据通道应用的专门结构。

模块阵列自然地数据通道运行分成两位的单片。CLB 包括进位逻辑支持每个模块两

位的加和减,两个 16×1 存储器和两触发器。在阵列中每行有两根三态总线的线。这些总线导线连到每行的两个 I/O 模块,I/O 模块可以配置持续三态总线到芯片外。到外部系统的接口用总线去耦多路开关,片上地址译码器和低扭曲片上时钟来简化。

结果是很容易构造面向总线的数据通道水平的位片。每行两位数据通道在一个函数发生器中可以包括算术、存储器或任何其它的功能实现。这些除了构成随机逻辑的能力。利用函数发生器和触发器连成需要的各种型式是有用的。

5. 主置位/复位

XC4000器件包含一个全局置位/复位(GSR)线,当 GSR 被要求,在 LCA 器件中的每个触发器同时被置位或复位,但无通用的布线资源被耗费,因为 GSR 有它自己专用的布线,以这种方式置位或复位所有的触发器远比利用单个 CLB 中 RD 腿更有效。每个触发器是置位或复位按照在原理图中所附着的属性,对此属性的缺省值是 $INIT=R$,它引起 GSR 去复位触发器,改变属性为 $INIT=S$ 引起触发器被置位,INIT 属性也决定在加电之后立即的触发器状态。为利用 GSR,在原理图中应该包含 STARTUP 原语,而且一个焊盘接到它的 GSR 腿,图9,任何用户 I/O 腿可以被利用,而且如果必要,此焊盘可以锁定到专门的器件腿;就象任何其它的焊盘,但是,如果可能,PPR 应该允许选择此位置,因为任何不必要的限制减少了 PPR 必须实现此设计的自由度,潜在地退化结果。

§ 6.3.4 边界扫描电路

边界扫描正在变成一个吸引人的特性来帮助富有经验的系统制造者更安全和更有效地测试其 PC 板,在生产中,印制板必须被测试以确保元件和连线二者的完善,但是,当集成电路变得更复杂和多层印刷电路板变得更密时,增加了测试装配的困难。

XC4000提供边界扫描测试设施帮助解决这个问题,所有用户 I/O 腿,完全是可测试的,而且测试协议完全和 IEEE 标准1149.1兼容,边界扫描不降低 XC4000器件的容量或能力,因为它仅利用专门的逻辑。

IEEE1149.1—1990边界扫描测试技术条件是联合测试行动组(JTAG)在八十年代开发的标准,边界扫描测试(BST)结构提供了有效地测试极紧引线空间的电路板上元件的能力。XC4000系列实现 IEEE1149.1兼容的 BYPASS,PRELOAD/SAMPLE 和 EXTEST 等边界扫描指令,当选择边界扫描配置时,三个通常的用户 I/O 腿变成这些指令的专用输入。设计和测试工程师可以插入标准的测试逻辑结构在其电子设计中,这个结构很容易用串行和/或并行连接一个四腿的接口到任何边界扫描兼容的 IC 上来实现。运用这些信号,用户可以串行加载指令和数据进这些器件来控制其输出的驱动和检查其输入,它是对针床测试的改进,避免了过激励器件输出的需要,减少用户接口到四个出腿,在标准中描述了可选的复位控制逻辑的第五个腿,在 Xilinx 的器件中没有用到。

实现 IEEE1149.1功能的专用片内逻辑包括一个16状态机构,一个指令寄存器和许多数据寄存器。寄存器由 CAPTURE 开始运行,一组数据并行地加载进指定的寄存器为移位输出,下一状态是 SHIFT,记录的数据移出,而需要的数据移入,为 WAITE 操作提供许多状态,一个寄存器序列的最后状态是 UPDATE,移位的寄存器内容加载进相应的指令寄存器或数据保持寄存器,或者为指令寄存器译码,或者为数据寄存器腿控制。

主要的数据寄存器是边界扫描寄存器,对于 LCA 中的每个 IO 块的出腿,它包含移位寄

存器和三个 update 锁存器的三个位:in,out 和三态的控制,非 IO 块的出腿有只为 in 和 out 的相应的单独的位,每个 Extest Capture 收集所有有效的输入出腿。其它的标准数据寄存器是单个触发器的旁路(BYPASS)寄存器,它重新同步正通过器件的数据,此器件不需要包含在当前的扫描操作中。CLA 器件提供两个用户网线(BSCAN. SEL1和 BSCAN. SEL2),它们是两个用户指令的译码。对于这些指令,两个相应的网线(BSCAN. TDO1和 BSCAN. TDO2)允许用户扫描数据移出到 TDO 上。数据寄存器时钟(BSCAN. DRCK)对控制测试逻辑是有效的,用户可能希望用 CLB 来实现。也提供 TCK 和 Run — test — idle 的与非(BSCAN. IDLE)输出。

XC4000边界扫描指令集也包括配置器件和读回配置数据的指令,如表6.2所示。

在每个 I/O 块内位序列是:in,out 和三态(3—state),由芯片的 XDE 片内视图,从芯片的右上角开始边界扫描数据寄存器的位有表6.3的顺序。数据寄存器也包括以下无出腿的位:TDO. T 和 TDO. 1,分别总是数据寄存器的位0和位1;BSCANT. UPD,它总是数据寄存器的末位。这三个边界扫描位是专用的 Xilinx 测试信号。PROGRAM,CCLK 和 DONE 不包含在边界扫描寄存器中。

表6.2 边界扫描指令

指令 I ₂ I ₁ I ₀	测试选择	TDO 源	I/O 数据源
0 0 0	Extest	DR	DR
0 0 1	Sample/Preload	DR	Pin/Logic
0 1 0	User1	TDO1	Pin/Logic
0 1 1	User2	TDO2	Pin/Logic
1 0 0	Readback	读回数据	Pin/Logic
1 0 1	Configure	DOUT	Disabled
1 1 0	reserved	—	—
1 1 1	Bypass	旁路寄存器	Pin?logic

表6.3 边界扫描顺序

Bit0(TDOcmd)	TDO. T
Bit1	TDO. 0
Bit2	顶边 I/O 块(右至左)
	左边 I/O 块(顶至底)
	MD
	MD1. t
	MD1. 0
	MD1. I
	MD0. I
	MD2. I
	底边 I/O 块(左至右)
	右边 I/O 块(底至顶)
(TDIend)	BSCANT. UPD

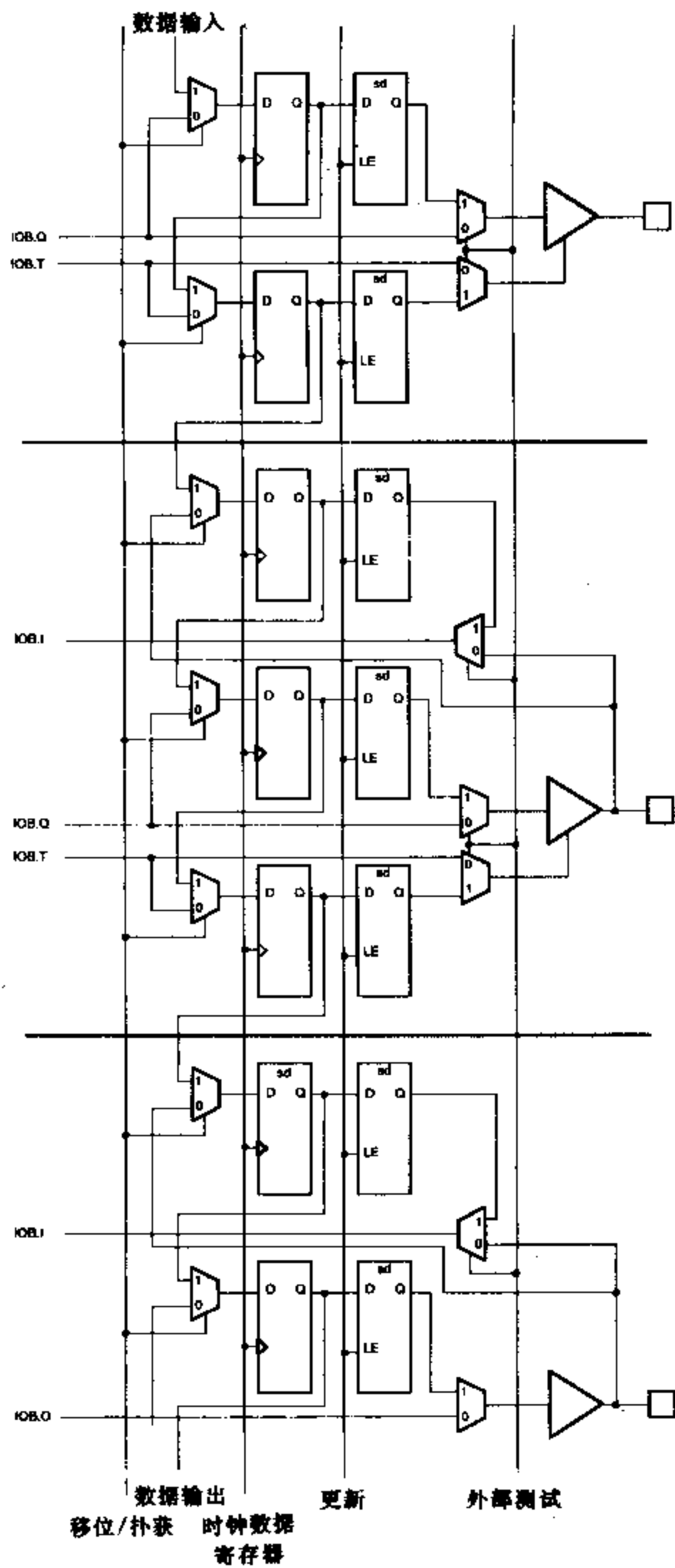
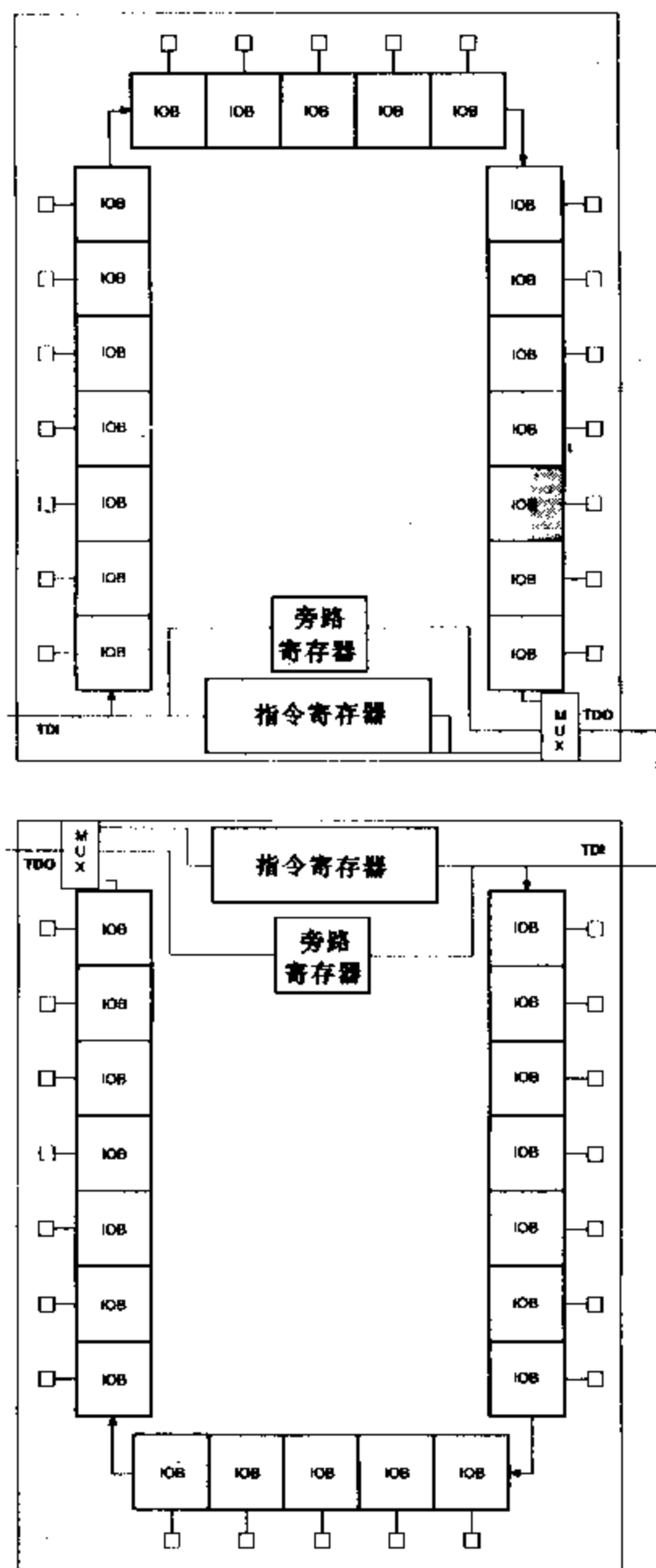


图6.11 XC4000边界扫描逻辑

§ 6.4 XC3000/3100系列的 LCA

§ 6.4.1 XC3000器件的 CLB

XC3000/XC3100中的每个 CLB 由一个组合的函数发生器和两个 D 触发器组成,图6.12给出了 XC3000系列 CLB 的结构,包括5个逻辑输入端(a、b、c、d、e),一个公共时钟输入 K,一个异步直接复位输入 RD,一个时钟使能 EC 及两个输出 X、Y,还提供一个 Data-in 输入以便对 CLB 中的触发器直接输入。两个输出可以由函数发生器或由触发器来驱动,两个触发器的输出可以不通过 CLB 的外部直接布线返回到函数发生器的输入

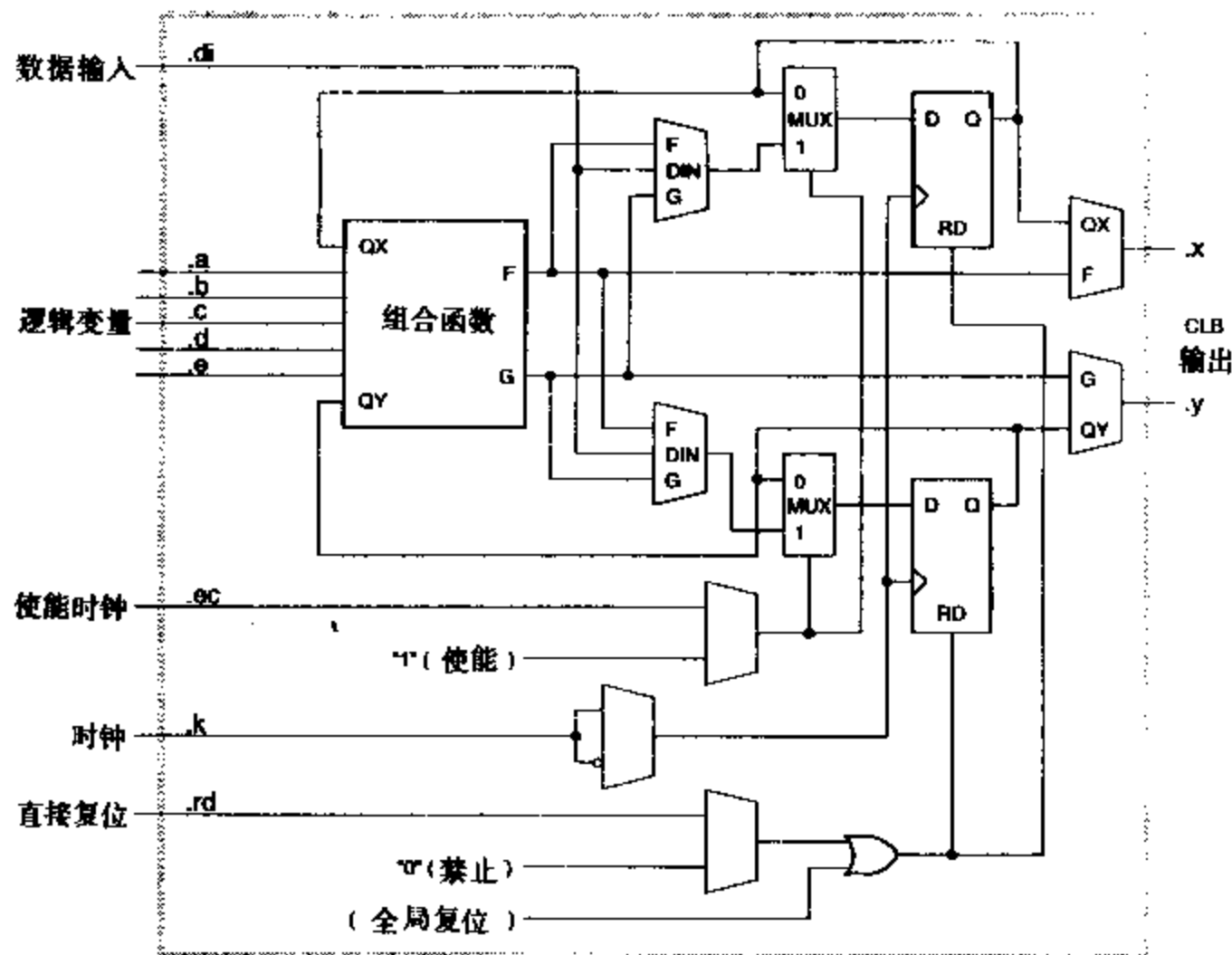


图6.12 XC3000系列 CLB 方块图

函数发生器由两个四输入的查找表组成,它们可以分别使用,也可以组合成一个单独的函数,图6.13表示了三种可用的选择,因为 CLB 仅有五个输入到函数发生器,所以这些输入必须在两个查找表之间共享。在 FG 模式中,函数发生器提供任何两个 a、b 和 c 加 d 或 e 的四输入函数,d 和 e 之间的选择由每个函数分别进行;在 F 模式,所有五个输入组合成单个五输入函数,任何的五输入函数都可以实现;FGM 模式是 F 模式的超级设置,两个 a、b、c 和 d 的四输入函数按第五个变量 E 复合在一起。在所有模式中,b 或 c 输入可选触发器输出 QX 或 QY 来代替。在 FG 模式中这个选择是对两个查找表分别进行的,功能上扩展到由七变量选出的任意两个四变量函数,只要这些变量的两个是存储在触发器中,在类似状态机的应用中这是特别有用的。在 F 模式中,函数发生器实现由七变量中选出的单个五变量函数,QX 和 QY 的选择对两个查找表被限定是相同的。而 FGM 模式与 F 模式不同,QX 和 QY 可以在两个查找表中分别选择,与 FG 模式相同。这个附加的灵活性允许仿真包括所有七个可能输入的被选函数。

在 XACT 开发系统中,自动逻辑分割软件仅利用了 FG 和 F 模式,但对人工分割来说,这三种模式都是有效的,可以在原理图中执行。当 FG 或 F 模式被要求时,简单地在原理图中包含定义 CLB 输入和输出的 CLBMAP。FGM 模式只是稍微复杂一些,CLBMAP 必须利用,在两个四输入函数之间转换的信号锁定在 E 腿上。如果所画的逻辑使得形成此多路开关的门无附加的逻辑合并进去而被明显地表示,CLB 将按 FGM 模式来配置。

两个 D 触发器共享一个公共的时钟,一个公共的时钟使能和一个公共的非同步复位信号。当数据以低有效形式来存储,利用非同步复位可以达到非同步预置;由复位产生的低对应于正被要求的位。这些触发器不可以用作锁存器。当输入到 CLB 触发器的数据由输入焊盘直接驱动而没有插入的触发器时,数据焊盘一时钟焊盘的保持时间一般并非为零。这个保持时间等于从时钟焊盘到此 CLB 的延时,但可以按照 70% 规则缩短,即保持时间按从数据焊盘到 CLB 的延时减少 70%,不考虑 CLB 的建立时间。按照 70% 规则的结果为负数时,此最小保持时间为零。

输出端选择,组合逻辑部份延时是固定的,与产生的逻辑函数复杂程度无关,组合逻辑的几种使用方法见图 6.13。

XC3000 的 CLB 内有两个 D 触发器,由 K 输入端的外部时钟驱动,时钟可以由用户规定有效极性(上升沿及下降沿),有复位端,由 RD 端或芯片内部全局复位驱动。数据端可由 DIN 和组合逻辑输出 F 及 G 三个信号中选择,Q 端既可直接驱动 CLB 输出端 X 及 Y,也可以作为组合逻辑的输入变量。

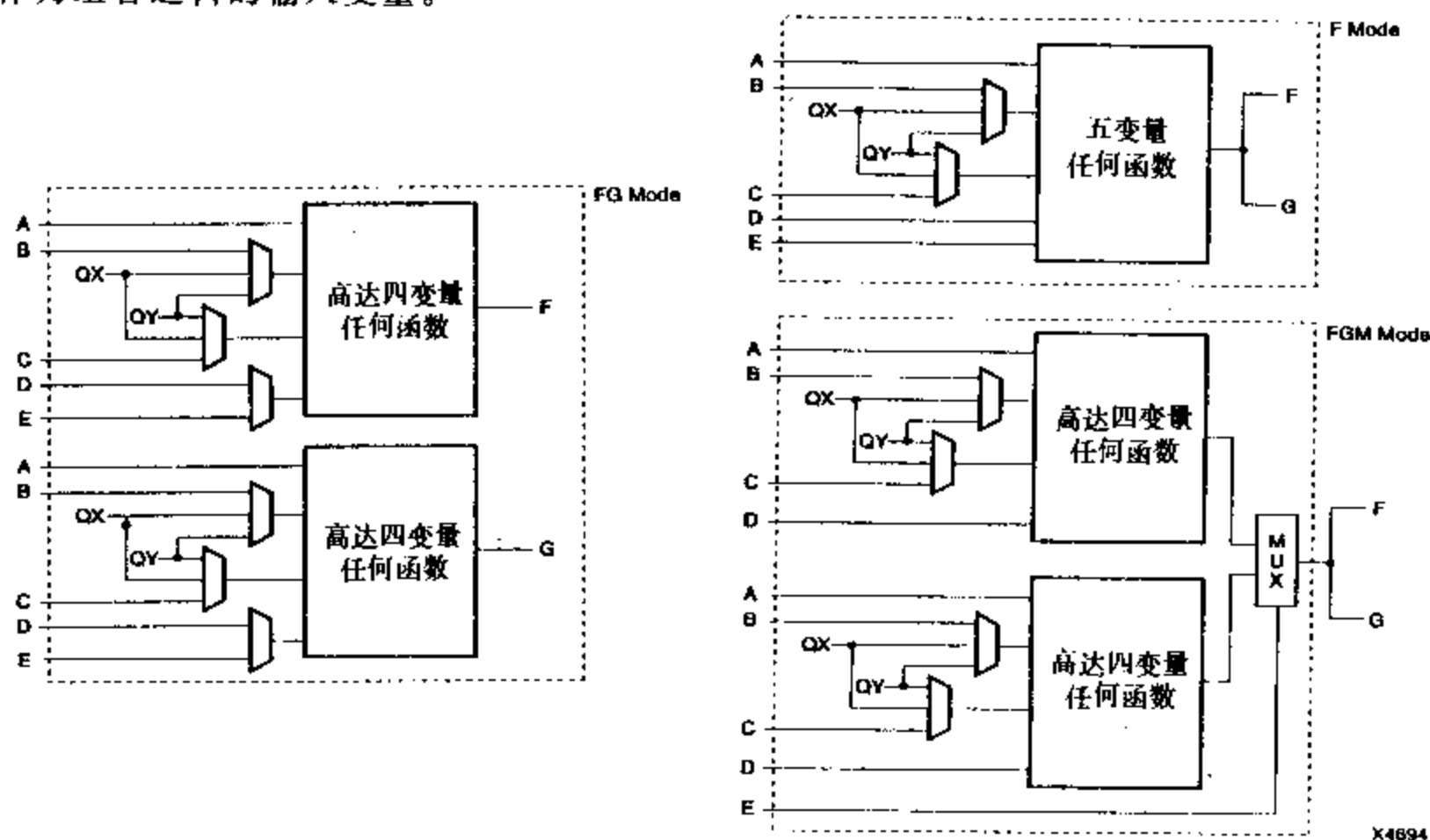


图 6.13 XC3000 系列组合逻辑选择

§ 6.4.2 XC3000 器件的 I/OB

XC3000 的 IOB 包括三部份:输入部份包括直接的及通过寄存器的输入路径,输入缓冲器按照用户编程将外部输入的 TTL 或 CMOS 信号转换成内部逻辑所要求的电平,其输出信号既可直接进入芯片内部,也可经一个 D 触发器寄存或 D 锁存器锁存之后再进入芯片内

部;输出部份包括一个寄存器,一个可选的反相器,一个可控制传递速度及三态控制可选相位的输出缓冲器。快速传递延时短 提高时间性能,慢速传递降低系统噪声,但延时长;公共部份有对存储单元在配置时或外部 RESET 时的全局复位(GLOBAL RESET),用户可编程的 PAD 端高阻值的上拉电阻,为输入和输出 D 触发器或锁存器提供时钟或使能的两路时钟资源。整个一边的 IOB 时钟可以1]时反相。XC3000结构图示于图6.14。

1. 输入部份

所有输入已限制滞后现象,一般对 TTL 输入门限超过200mV,对 CMOS 门限超过100mV。对此例外的是 PWRDWN 腿,和配置作为晶振输入的 XTL2腿。

实验证明输入上升和下降时间应不超过250ns,此数值通过最坏条件测试被确定的,利用内部环形振荡器驱动除两条腿之外的所有 I/O 腿,因此产生最大的片内噪声。保留的两腿之一被配置为输入,对单沿响应测试;另一 I/O 腿用作监视此响应的输出。这些测试条件多半要求苛刻,虽然已假定 PC 板有好的电源去耦和忽略接地噪声,保守看在大多数情况下最终的规定容易满足。

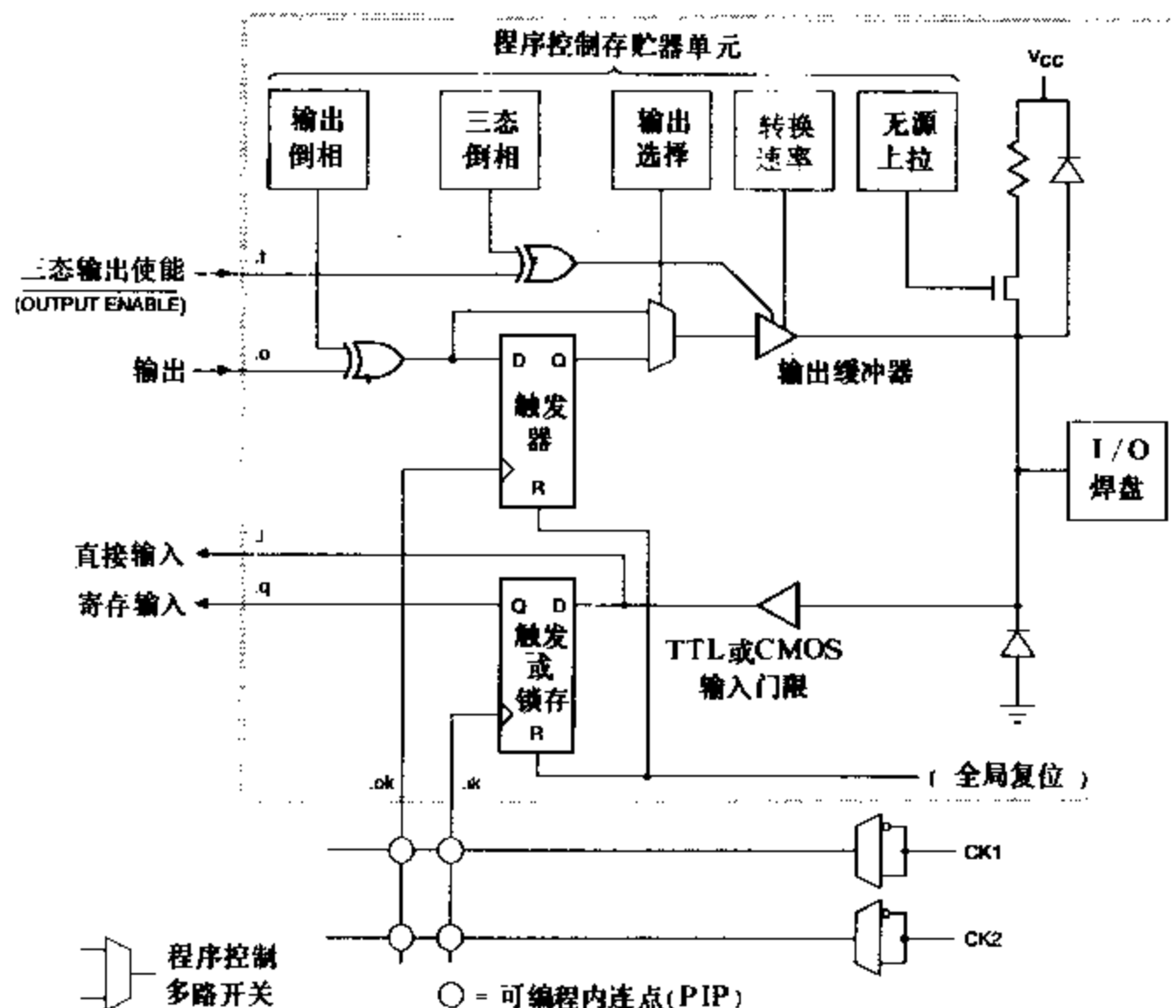


图6.14 XC3000的 I/O 原理图

只要利用专用的时钟输入焊盘和 GCLK 缓冲器,I/O 块的输入触发器保证正确运行而没有数据保持时间(相对于器件输入焊盘的时间)。使用 TTL 时钟或不同的时钟焊盘将导致要求数据保持时间,这个保持时间的长度等于从实际的时钟焊盘到 GCLK 缓冲器的延迟减去从专用 CMOS 时钟焊盘到 GCLK 缓冲器的延迟。为了保证输入触发器有零的保持时间,

延时包括进触发器的 D 输入端,造成它有相对长的建立时间。但是,在数据表上规定的建立时间是相对于到达此 I/O 块的时钟而言的。因为在时钟焊盘和此 I/O 块之间不可避免地存在延时,输入焊盘到时钟焊盘的建立时间实际上小于数据表上的数字。部分时钟延时可以从内部建立时间中减去,概念上可以减去所有时钟延时,但是当内部建立时间是其最大值时,时钟延时比其最大值小是可能的。因而在最坏条件设计中推荐只减去70%的时钟延时。如果内部建立时间的要求也比它的最大值小时,时钟延时只可能比它的70%的最大值小。在此情况下,实际要求的焊盘到焊盘的建立时间将比计算值小。例如,在 XC3000—125中,关于时钟到达 I/O 块的输入建立时间是16ns,如果从时钟焊盘到此 I/O 块的延时是6ns,则此延时的70%为4.2ns,可以减去它达到约12ns 的最大焊盘到焊盘建立时间。无论何时一个延时从另一个延时中减去必须运用70%的规则。但是建议延时校正只用于带输入保持时间的连接。在非同步电路中明确地建议不用延时校正。任何时候延时校正不应变成负值,如果70%的校正延时比从中减去的延时大时,最终的延时为零。70%规则决不定义由芯片到芯片可能遭受的绝对延时最小值,和随温度及电源的变化。它简单表示在整个运行条件范围上特定芯片内可找到的相对变化。一般说,所有的延时比它们的最大值要小,由于某些延时是不成比例地比其它要更快,70%规则描述了按比例因素的扩展;如果它已按减最小延时的建议缩减,减最大的延时将不比其本应有的延时的70%小。实际上,按最坏条件设计,假设任何延时不会一点也不缩减并保持在最大值,其它的延时将不比它们的最大值的70%小。

2. 输出部份

所有 XC3000/3100的 LCA 输出是 n 沟道的下拉晶体管和 p 沟道上拉晶体管的真正的 CMOS 电路,无负载时,这些输出从电源和地的干线拉到干线。一些附加的交流特性已列在表6.4中

表6.4 交流特性

交流特性	快速	慢速
无负载输出畸变率	2.8V/ns	0.5V/ns
无负载传输时间	1.45ns	7.9ns
812pF 附加上升时间	100ns	100ns
归一化值	0.12ns/pF	0.12ns/pFa
812pF 附加下降时间	50ns	64ns
归一化值	0.06ns/pF	0.08ns/pF

输出短路电流值只对指定的充电和放电电容负载能力给出。按其它逻辑器件一般的工业实践,在一次仅一个输出可以被短路,且短路到电源和地的持续期不可以超过一秒。Xilinx 不建议连续的输出或嵌位电流对任一输出腿超过20mA。数据表保证输出在320mV 时不大于4mA 以避免在许多输出同时地是沉降电流的问题。

高有效的三态控制(T)和低有效的输出使能($\sim$ OE)是相同的,换言之,在 OBUFZ 的 T 腿上高电平把输出设置在高阻状态,低电平使能此输出。相同名称的规定用于 LCA 器件内的 TBUF。

3. I/O 时钟部份

内部多达八个不同的 I/O 时钟可利用,在芯片的四边每边有两个。当 I/O 块没有提供可编程的时钟极性,适用于一个 I/O 块的两条时钟线可用作正的和反相的时钟,相应的极性接到此 I/O 块。但这不限制芯片这一边的所有 I/O 块只利用一个时钟的两个沿。I/O 块的锁存有低有效的锁存使能,当时钟输入为低时它们是透明的,而当时钟输入为高时它们是关闭的。锁存器取得数据在有效时钟沿,在其之前的半个时钟周期是透明的。

§ 6.4.3 XC3000/XC3100系列的布线资源

1. 水平和垂直长线

对于 XC2000/XC3000系列这些水平和垂直的长金属线段载送信号横越片子的长和宽,它们可用作双向或单向数据总线,并且最适用于高扇出低偏移控制或时钟信号。图6.12是水平和垂直长线的图示,在每个布线通道有四根垂直长线,两根为通用的,一根为全局时钟网线,一根为备用时钟网线。在 CLB 两行之间的两根水平长线,CLB 的每一列用一个 TBUF 驱动每个水平长线,在长线的左端增加一个附加的 TBUF,这个附加的 TBUF 对驱动 IOB 的数据到长线是方便的,一般,到 TBUF 的 T 和 I 腿的布线资源多少有点受限,水平长线可选择在任一端或两端被上拉,每个上拉电阻的值是3—10K Ω 。此外,水平长线固定由低功率的锁存器驱动,它们很容易用有效输出或上拉电阻取代,这些锁存器保持没有上拉或暂时没有驱动的水平长线的逻辑电平,被保持的逻辑电平是有效驱动到此长线的最后电平。当利用三态水平长线为多路开关时,用不到四个 TBUF 就是浪费资源,四个或少于四个输入的多路开关可以用 CLB 更有效地来实现。

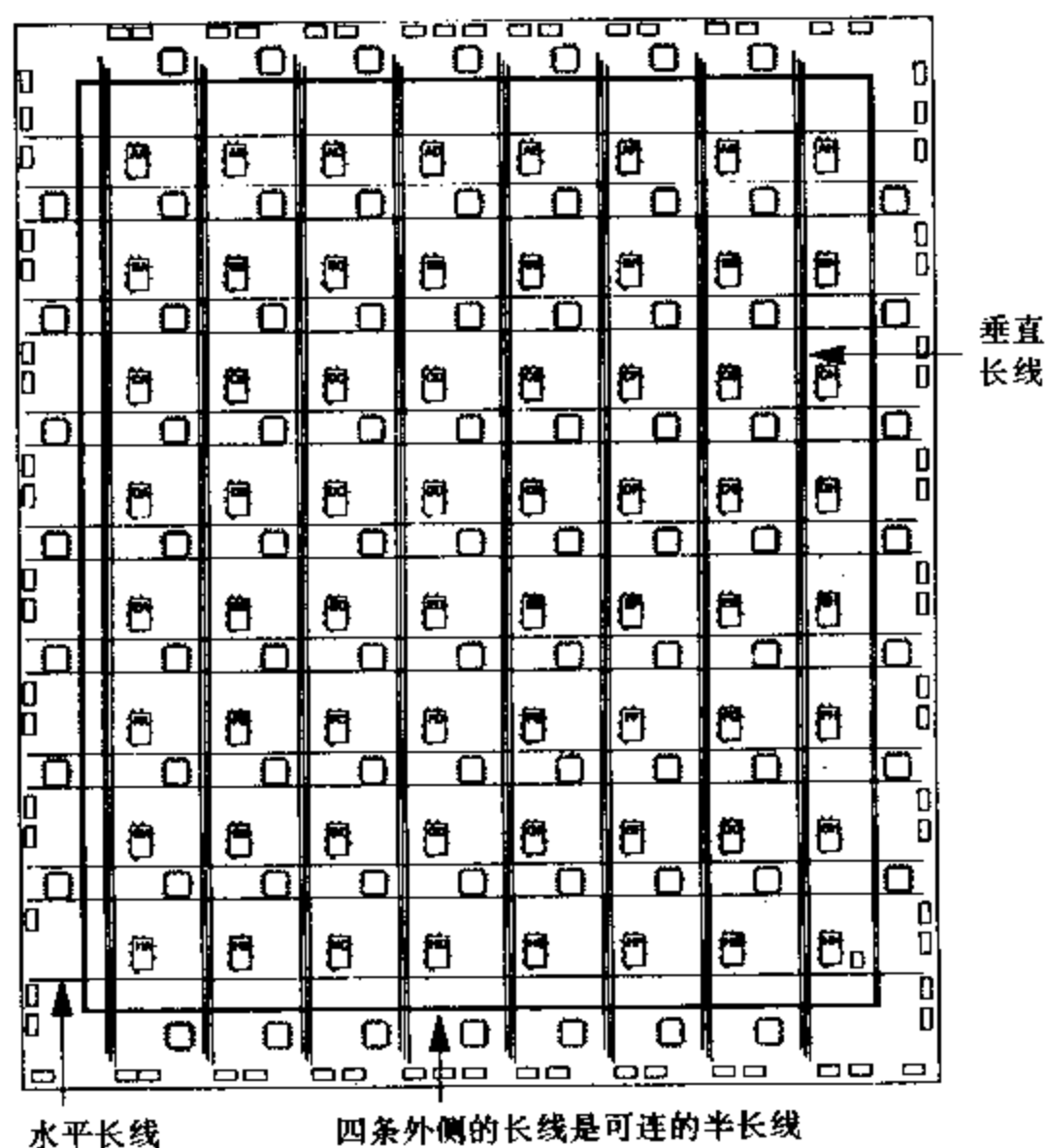


图6.15 水平和垂直长线

2. 时钟缓冲

每个器件包含两个高扇出低畸变的时钟分布网线,全局时钟网线起源于芯片左上角的 GCLK 缓冲器,而备用时钟网线起源于芯片右下角的 ACLK 缓冲器。所以 GCLK 及 ACLK 是与 XC2000/XC3000 器件中的全局时钟缓冲器及备选时钟缓冲器相对应的,在一个 LCA 设计中只能用一个 GCLK 和一个 ACLK,与全局缓冲器相连的是专门的长线,用它驱动时钟偏移特别小,并且扇出能力极大,能驱动所有的 CLB。备选缓冲器驱动一根水平长线,可转接驱动垂直长线,其特点也是偏移小,扇出能力强,适合驱动时钟或高扇出信号。

全局时钟网线及备选时钟网线每个有可选的快速 CMOS 输入,分别称为 TCLKIN 和 BCLKIN,利用这些输入提供从 PC 板到内部触发器和锁存器的快速通道因为信号旁路了输入缓冲器,定义好的 CMOS 电平必须保证在这些时钟腿。为了在电路图中规定利用 TCLKIN 或 BCLKIN,直接连接 IPAD 符号到 GCLK 或 ACLK 的符号,在 IPAD 和时钟缓冲器之间放一个 IBUF 将防止 TCLKIN 或 BCLKIN 被利用。时钟缓冲器输出网仅驱动 CLB 和 IOB 时钟腿。它们不驱动任何其它 CLB 输入。在需要把时钟接到逻辑输入或器件输出的罕见的情况下,信号应分接到时钟缓冲输入,并连线到逻辑输入,但利用 TCLKIN 或 BCLKIN 的时钟是不可能的。由局部连线对时钟布线而产生的时钟畸变使得安全的设计很难达到,因此实际上不推荐此法。通常,时钟利用的越少,设计越安全。高扇出的时钟总是应利用 GCLK 或 ACLK,如果要求的时钟多于两个,ACLK 网可以分段成单独的垂直长线,在每列的顶和底部由 PIP 来驱动。由局部连线对时钟信号布线仅应对单个的触发器来考虑。

3. 直接连接线(仅限于 XC2000和 XC3000系列)

这些直接的路径用于连接相邻的 CLB 或者相邻的 CLB 和 IOB,每个 CLB 的输出能通过直接连接线段和与之相邻的 CLB 或 IOB 的输入相连。这些资源最适用于相邻块间信号高速传送。图6.16是一个例子。

直接连接线布线短,延时很小。CLB 之间的直接连接规则是:X 输出端接到其上面的 CLB 的 C、D 端,其下面的 A、B 端,Y 输出端接到其右边的 CLB 的 B 端。

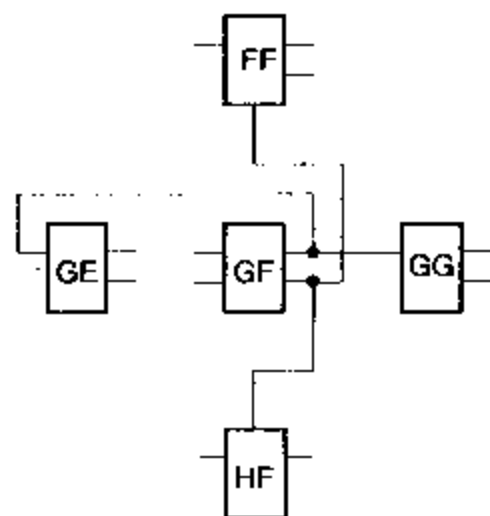


图6.16 直接连接线

4. 通用连接线

通用连接线由一个短的相邻的金属线段阵列组成,这些线段是在 CLB 行列间水平或垂直排列取向的,垂直线段长度为 CLB 的行距,水平线段长度为 CLB 的列距,CLB 的输出端

和邻近的通用内部连线相连,从而与开关矩阵相接,开关矩阵把通用内部连线的各线段连在一起形成网线开关矩阵和 CLB 输入端相连的线是单向的,相当于 CLB 的输出端。通用内部连线上还有一种双向缓冲器,用于对高扇出信号进行隔离和放大。LCA 内部 CLB 阵被分成九块,双向缓冲器分布在每块的边界上,块之间的连线用双向缓冲器放大,块内的连线则用不着。通用连接线示于图6.17。

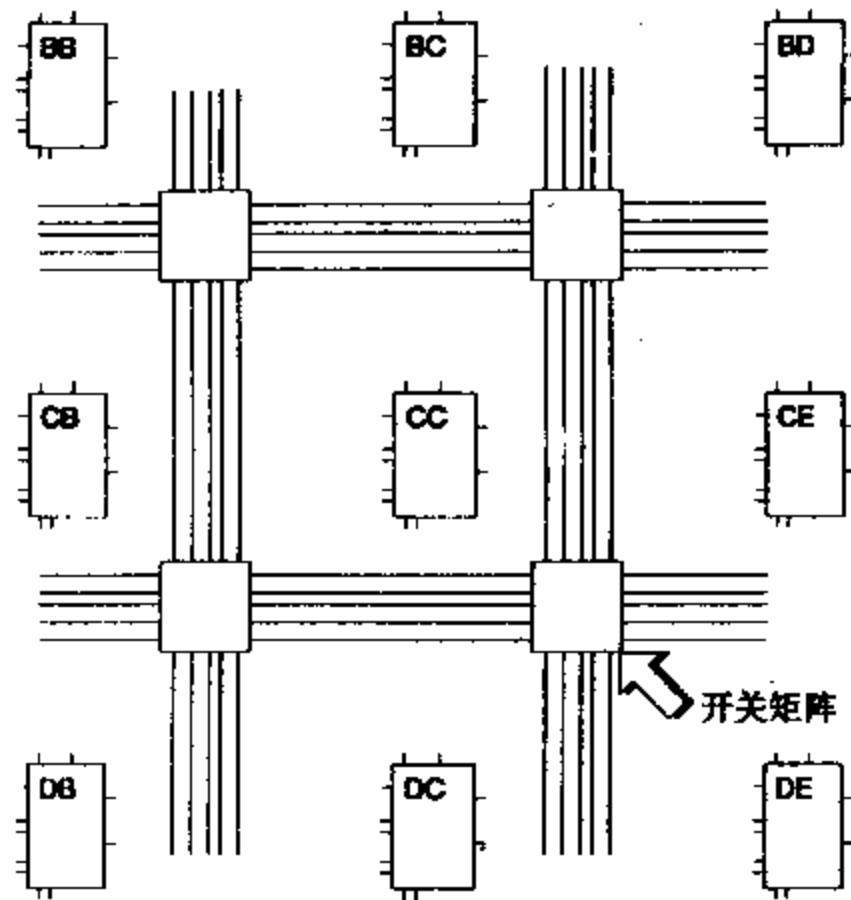


图6.17 通用连接线

5. 可编程连接点(PIP)

如图6.18,PIP 是一些独立的开关,用于相交布线线段或布线线段与 CLB 或 IOB 引脚间的连接。

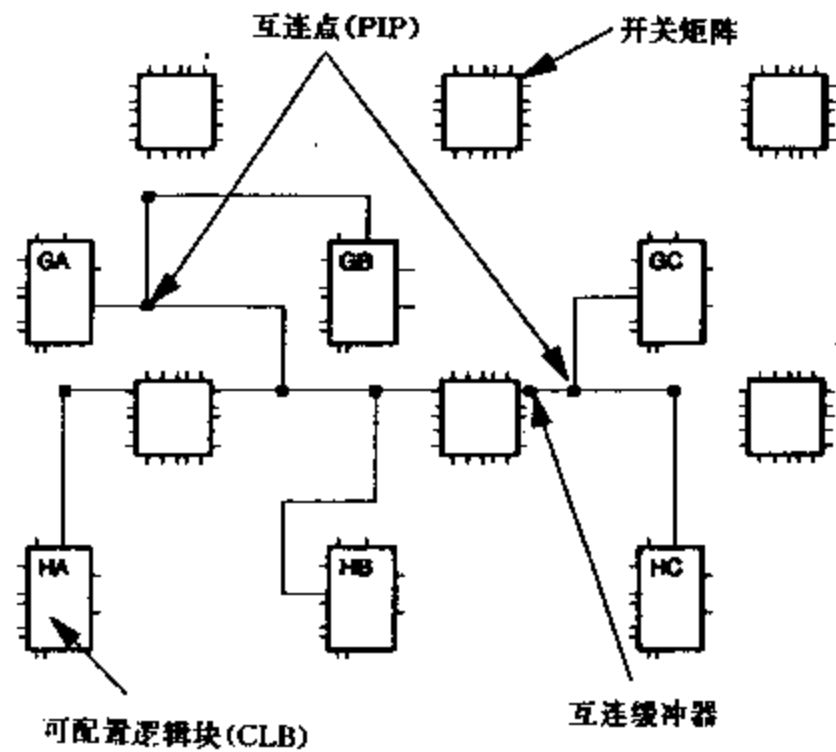


图6.18 可编程连接点(PIP)

6. 开关矩阵

开关矩阵位于水平和垂直通用连接线段의 相交处。这些矩阵也称为“魔术盒”。图6.19示出 XC3000开关矩阵腿之间可能的连接。

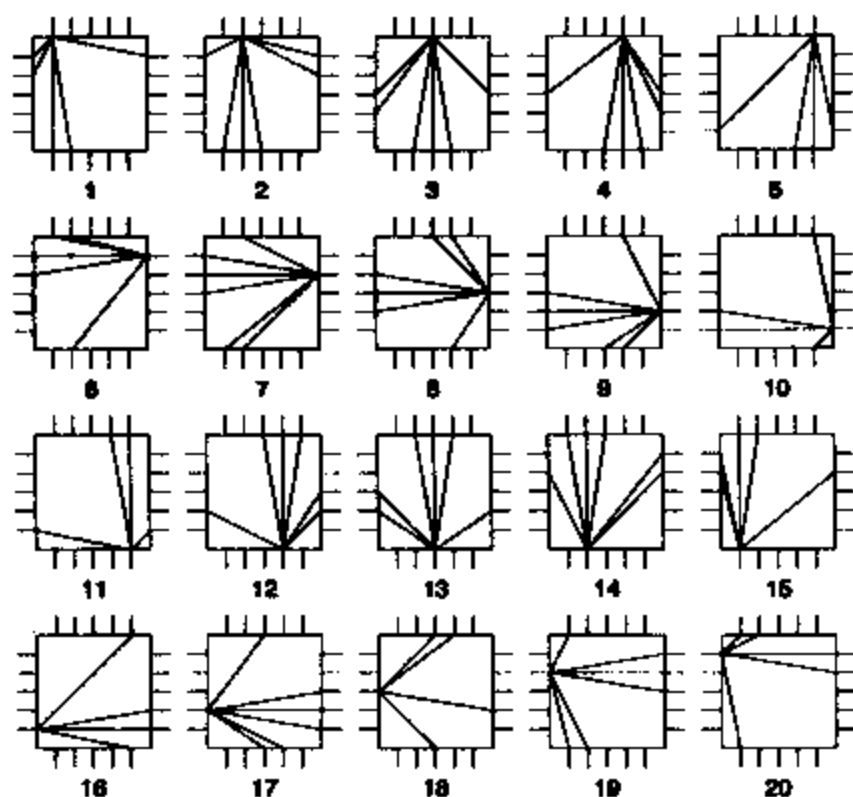


图6.19 开关矩阵

§ 6.5 XC2000系列的 LCA

§ 6.5.1 XC2000器件的 CLB

XC2000的每一个 CLB 都有三部份组成,一个组合逻辑部份,一个存贮单元及一个内部连线控制部分。另外,每个 CLB 还有四个通用输入 A、B、C 和 D,一个专用时钟输入 K,两个输出 X 及 Y,通过内部互连线可与其它的 CLB 或 IOB 相连,XC2000CLB 结构示于图6.20。

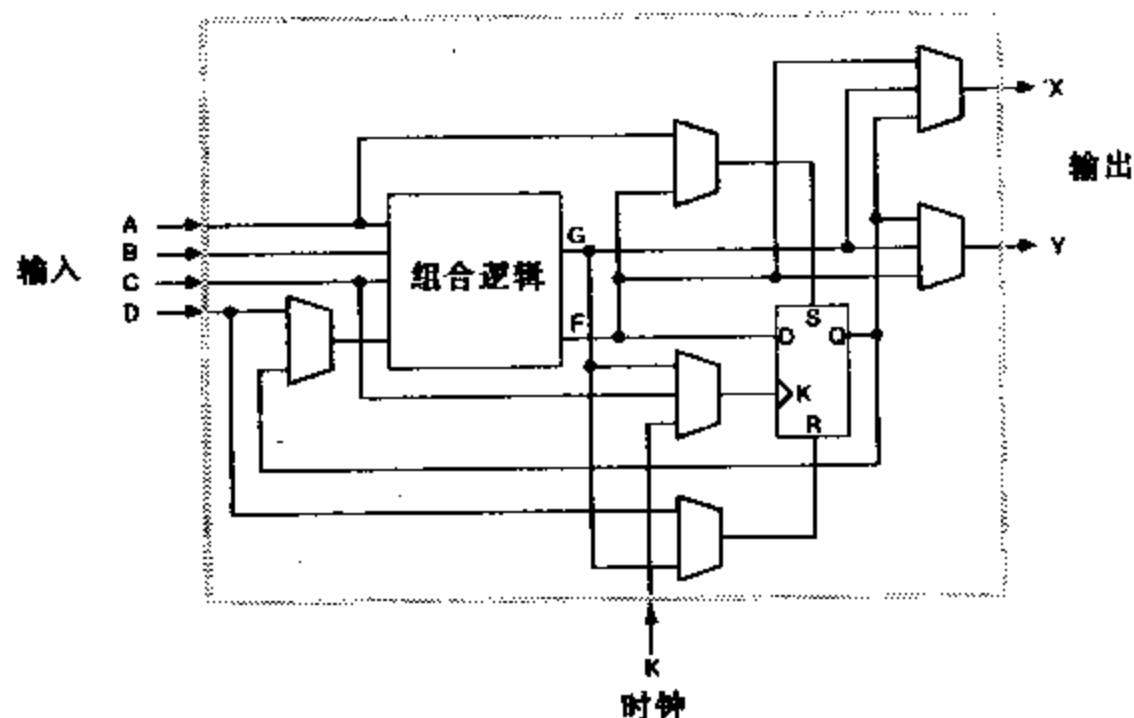


图6.20 XC2000系列 CLB 方块图

与 XC3000系列类似,图6.21给出 XC2000系列 CLB 实现组合逻辑的选择。每个组合逻辑

辑部份可以产生一个四变量任意逻辑函数或两个三变量任意逻辑函数。

CLB 中的存储单元可配置为边沿触发的 D 触发器或电平触发的 D 锁存器, D 触发器的时钟端或锁存器的使能端可从 K, C 和 G 三个信号中选择。

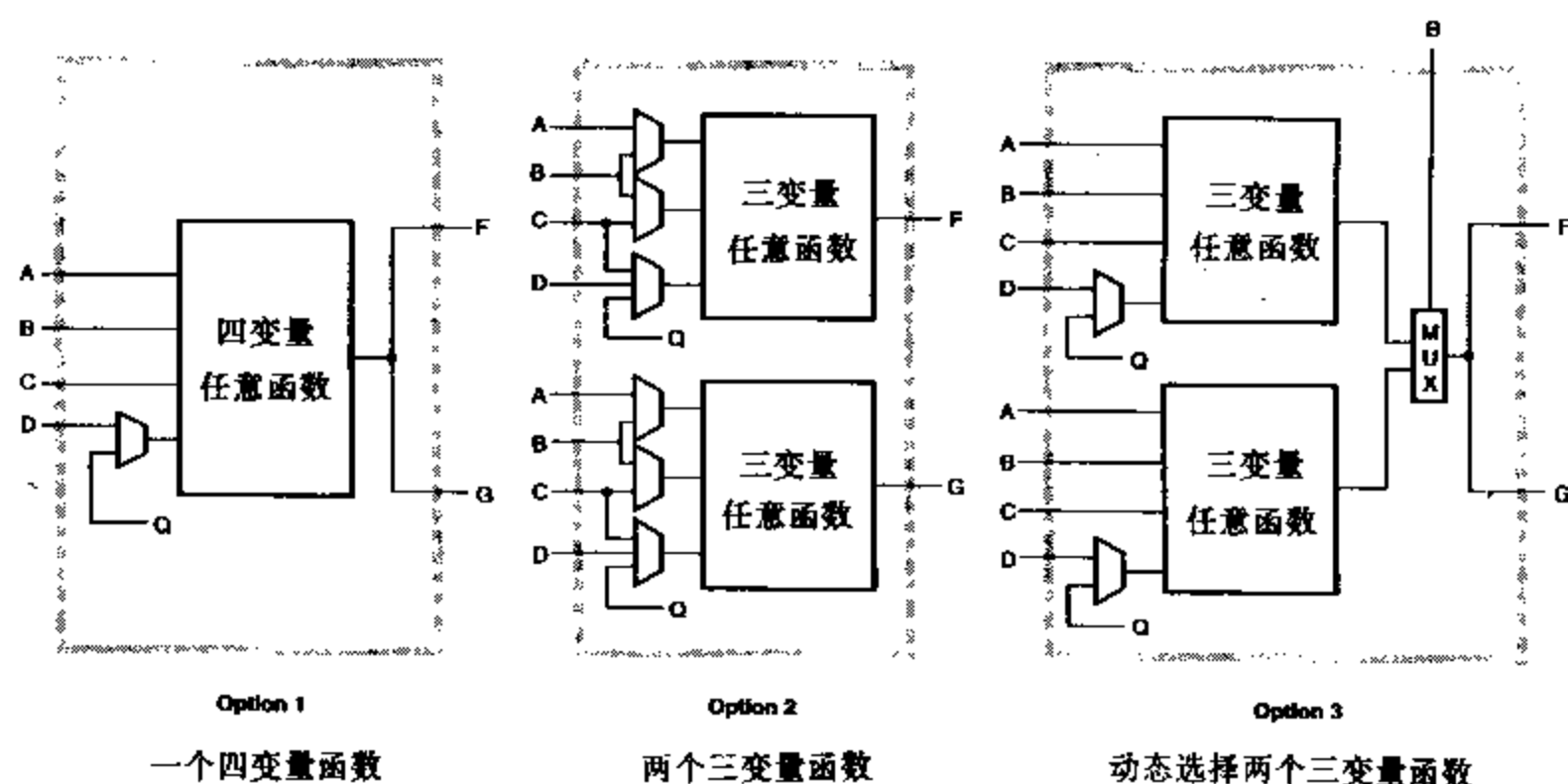


图6.21 XC2000系列组合逻辑选择

§ 6.5.2 XC2000器件的 I/OB

XC2000的 I/OB 包括一个直接的或通过寄存器的输入及一个三态输出。XC2000的 I/OB 结构示于图6.22。

XC2000的 I/OB 中输入缓冲器具有阈值检测功能,把封装腿上的电平转换成芯片内部电平,输入缓冲器的阈值分为 TTL(1.4V)和 CMOS(2.2V)两种,由用户编程来选择,其输出信号既可直接进入芯片内部,也可经一个触发器寄存之后再进入芯片内部,芯片每一边的 I/OB 共享一个 I/O 时钟,在配置过程和 RESET 有效的两种情况下, I/OB 的 D 触发器被清零。

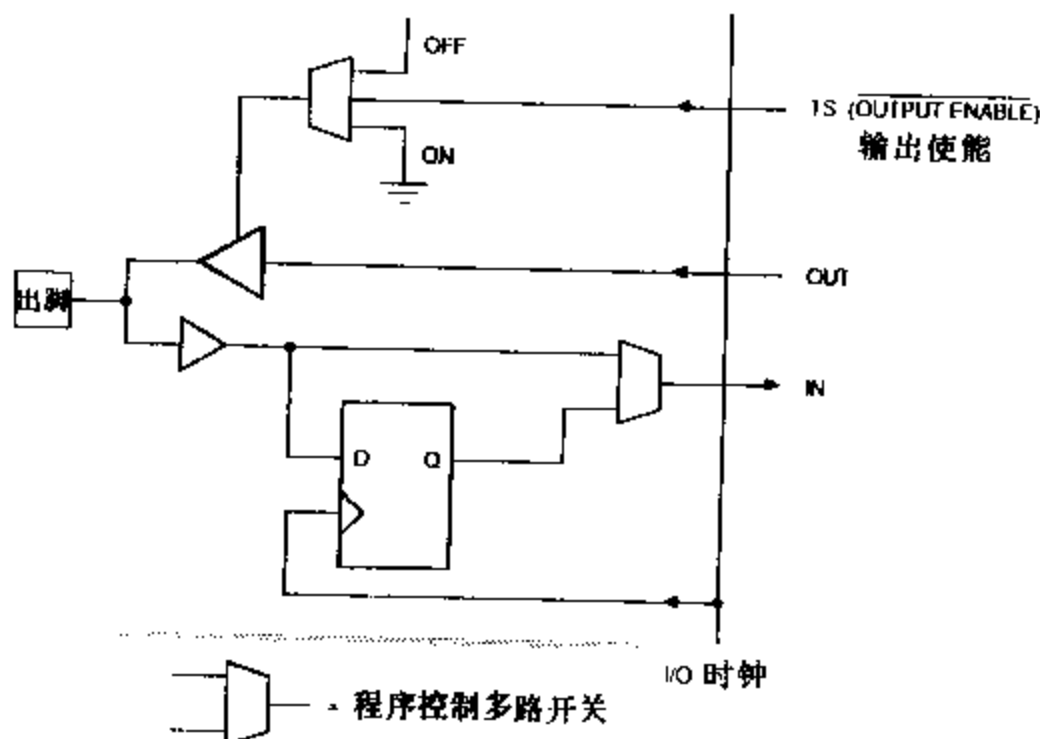


图6.22 XC2000的 I/O 原理图

输出缓冲器为高扇出的 CMOS 或 TTL 电平信号提供4mA 的驱动电流,其输入端(I/OB 的 O 端)由来自芯片的输出数据驱动,并有由用户编程控制的“允许”,“禁止”及“三态控制”三种工作状态。

§ 6.5.3 XC2000系列的布线资源

XC2000系列的布线资源与 XC3000/XC3100系列的布线资源相同,所以可参考 § 6.4 中 XC3000/XC3100系列的布线资源这一节。

§ 6.6 XC5200 FPGA 系列

XC5200 FPGA 系列是为提供最低成本的 FPGA 系列而设计的,新的 XC5200 系列的结构对三层金属和 $0.6\mu\text{m}$ CMOS SRAM 工艺进行优化,使硅片的效能已达到惊人的改进,这些改进使 XC5200 系列处于作为可替代门阵列的低成本大容量器件的地位。在已有的三种成功的 SRAM FPGA 系列所获得的经验的基础上,Xilinx 使 XC5200 系列有更优越的性能加到高密度可编程逻辑设计中。图 6.23(a)代表 XC5200 系列结构的一个简化的概念性的全貌,类似于普通的 FPGA,XC5200 系列由可编程 I/O 块,可编程逻辑块和可编程互连组成,但与其它 FPGA 不同,XC5200 系列的逻辑和局部布线资源组合成灵活的多用功能块 (VersaBlock),通用布线通过通用布线矩阵 (GRM) 接到多用功能块。

1. 多用功能块 (VersaBlock)

在每个多用功能块结构中的基本逻辑元件是如图 6.23(c) 中所示的逻辑单元 (LC),每个逻辑单元由一个四输入的函数发生器 (F),一个存储器件 (FD) 和控制逻辑组成,对每个逻辑单元有五个独立的输入和三个输出,输入和输出的独立性允许软件使每个逻辑单元内的资源利用率达到最大。每个逻辑单元也包含一个直接的反馈通道,它无损于函数发生器或寄存器的利用,这个特性对 FPGA 是基本的。存储器件可配置为 D 触发器或锁存器,控制逻辑由为快速实现算术函数的进位逻辑组成,它也可以配置为一个级联链,允许译码十分宽的输入函数。

图 6.23(c) 实际上表示由四个逻辑单元组成的 XC5200 系列的可配置逻辑块 (CLB),所以每个 CLB 有 20 个独立的输入和 12 个独立的输出,上面和下面逻辑单元可以配置成实现五输入函数。一般总是要求 FPGA 的实现软件能使资源的利用最大,XC5200 系列做到这一点靠围绕每个 CLB 的两类局部互连,即局部互连矩阵 (LIM) 和直接连接,这两个互连资源和 CLB 组合在一起形成多用功能块,如图 6.23(b) 所示。

局部互连矩阵 (LIM) 提供在一个给定的 CLB 中每个逻辑单元的输入和输出的 100% 的连通性,LIM 的好处是不要求通用布线资源来连接 CLB 中的反馈通道,LIM 经过 24 个双向结点连到通用布线矩阵 (GRM)。

直接连接允许立即接到相邻的 CLB,再一次不用任何通用的互连,这两层局部布线资源改善了这种结构的粒度,有效地使 XC5200 系列为“逻辑单元的海” (sea of logic cell)。每个多用功能块有四个共享一根公共使能线的三态缓冲器,并直接驱动水平长线,产生稳健的片内总线能力。多用功能块允许快速和局部地实现逻辑函数,有效地按层次型式实现用户的设计,这些资源也使局部布线的拥挤最小化,改善了用于连接更大的逻辑组的通用互连的效率,细粒度和粗粒度结构的这种组合使 XC5200 系列的逻辑利用率最大化,这个对称的结构得到第三金属层的所有优点,免去布局软件用最小的布线限制最优地打包用户逻辑。

2. 多用环 I/O 接口

在 XC5200 系列中 I/O 块和核心逻辑之间的接口已被重新设计,I/O 块与核心逻辑完全去耦,XC5200 I/O 块包括专用边界扫描逻辑以增加板级的可测性,但不包括输入或输出寄存器,这个方法允许最大数目的 I/O 块围绕器件放置,改善 I/O 与门的比率,并减少每个 I/O 的成本。围绕器件的互连单元的“自由方式”形成此多用环,它提供由 I/O 块到内部逻辑的连接,这些增量布线资源提供由每个 I/O 块到最近的多用功能块的连接,除长线围绕器件之

外。多用环消除了高逻辑利用率与出腿布局灵活性之间的历史性矛盾,这些增加的边沿资源给用户在完成其逻辑设计前预处理(即锁定)I/O 出腿能力,这个能力可加速产品的上市时间,因为 PCB 和其它系统元件可以和逻辑设计并行进行。

3. 通用布线矩阵(GRM)

GRM 功能上类似于其它结构中见到的开关矩阵,但是,它与包含在多用功能块中逻辑资源的紧耦合是新颖的。先进的仿真工具在 XC5200 结构开发期间被用来决定要求的布线资源的最优程度, XC5200 系列包括六级互连层次,一系列单长度线,双长度线和长线都通过 GRM 布线,直接连接, LIM 和逻辑单元反馈包含在每个多用功能块中,整个 XC5200 的互连,一个有效的多路开关方案和三层金属组合被用来改善整个硅片利用的效率。

由于许多门阵列不再支持多于 5000 门密度范围内的门阵列, XC5200 系列成为理想的替代产品,它提供高达 40MHz 的系统性能,用户现在可用 FPGA 的可再编程能力,短的研制周期和快的上市时间等优点用于以前利用门阵列的应用,这些应用包括网络适配器,打印机,复印机,扫描仪和大多数通信及计算机应用中遇到的数据通道的功能。

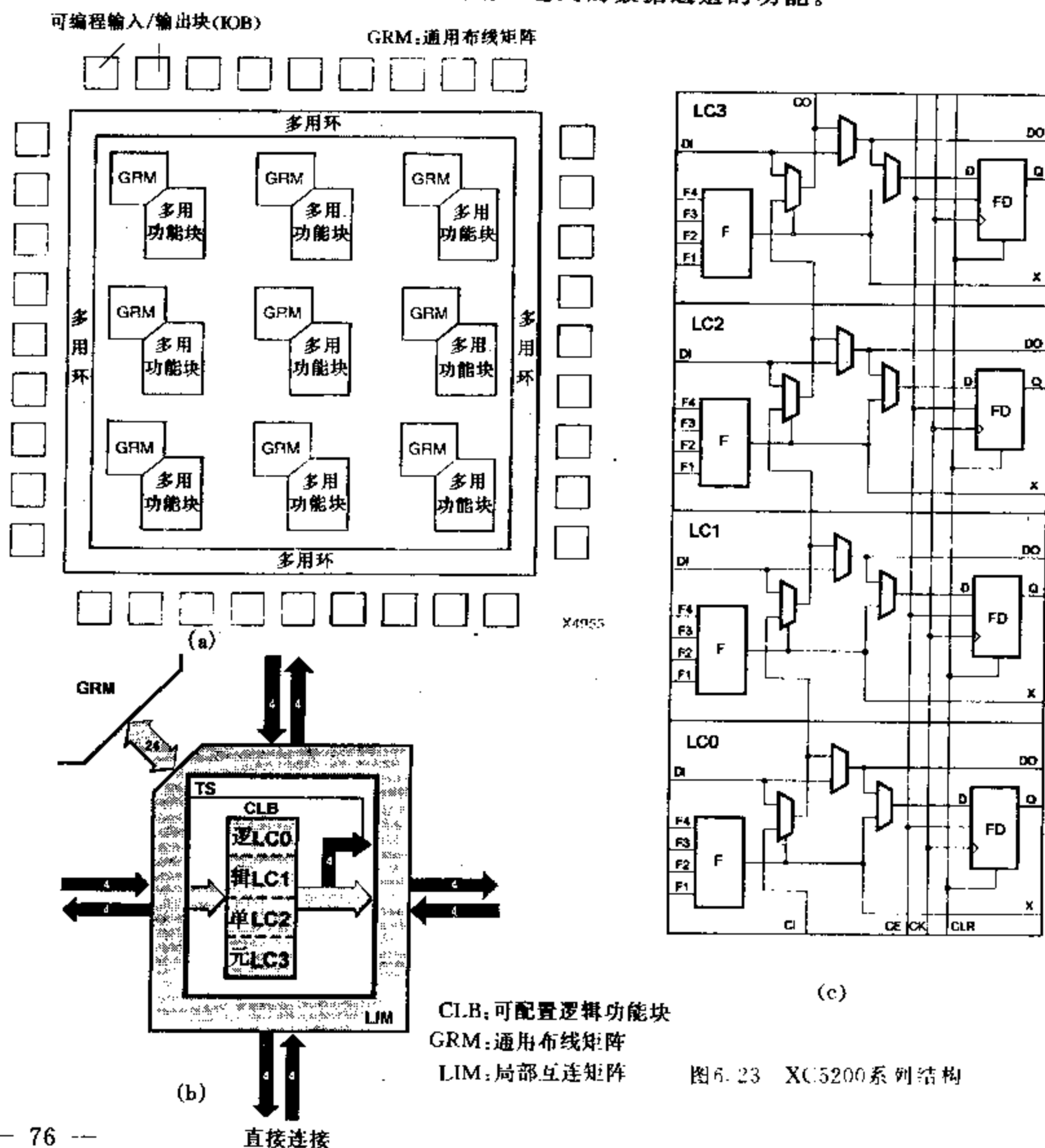


图 6.23 XC5200 系列结构

§ 6.7 XC8100 FPGA 系列

§ 6.7.1 特性概述

作为细粒度结构 FPGA 的 XC8100 系列引入了最新的 MicroVia 工艺,它提供了所有与其它 Xilinx 器件相同的优点:功耗低,加快上市时间,减少设计风险,以及标准的产品有效性和使用已有的设计方法。它将掩膜门阵列的密度和可编程逻辑的灵活性结合在一起。

由于在使用自顶向下与工艺无关的设计方法时设计可达到非常有效并节省成本, XC8100 系列采用新的门海 FPGA 结构,基本单元是小的和为工艺无关的设计专门构造的。由于布线完全被堆积在逻辑之上,占有比其它可编程逻辑技术要显著小的面积, XC8100 FPGA 在小的管芯上有大量的布线导线和互连资源,在 100% 单元利用率时达到高的布通率。MicroVia 的反熔丝有小于 50Ω 的典型的接通电阻,可在高速下运行。

与 Xilinx 的 SRAM 的 FPGA 可无限次再编程不同, XC8100 系列是一次可编程的 (OTP),编程是用 Xilinx 或第三方的编程器来实现,类似于一次编程的 PLD 器件,在实际应用中不需要配置的存储器,设计的保密性是十分高的。因为被编程的通孔几乎不能观察到,对阵列配置作反向工程的可能极小。在一个芯片上有许多 MicroVia,如等效 5616 可用门的 XC8106 上有 75 万个 MicroVia,但仅 2% 被编程,要找出被编程的 MicroVia 如同在大海捞针。此外, XACT8000 软件以压缩和加密的格式存贮设计信息,并提供一个可选密码的能力,它只允许设计文件用于编程而不能观察,写入和操纵。数据从与 JTAG 兼容的测试口送入。

与所有真正的 FPGA 一样, XC8100 系列由逻辑单元阵列和可编程布线资源组成,由环形的 I/O 连线围绕;与大多数 FPGA 企图提供“最好”的固定单元不同, XC8100 的基本逻辑单元是本身可编程的,称为“可配置逻辑单元”(CLC),它们可以实现同步,组合或三态功能,这意味着 XC8100 的软件有选择最好单元结构的灵活性,取决于要实现的逻辑。

与 Xilinx 的其它 FPGA 相比, XC8100 的结构有更多的单元,每个单元有较少的门,它也有较少的预配置的资源,例如,在逻辑单元或 I/O 单元中无触发器;布线资源是丰富的,到逻辑单元的输入和输出的互连有十分大的比例,此结构的目标是对多种多样的应用达到最高的门利用率和布通率。

用 XC8100 系列的设计类似于 ASIC 和其它 Xilinx 的 FPGA,设计用第三方的 CAE 工具输入和仿真,然后用 Xilinx 的工具布局 and 布线。XC8100 的结构使设计自动完成的可能性达到最大,人工控制是可能的。因为此结构和为它编写的新软件,当利用工艺无关的设计输入时,器件的利用率不降低,设计可以利用不涉及 IC 结构的高级设计语言 (HDL) 来输入,不同的逻辑实现可以在布局 and 布线之前精确地被估算。设计实现在 § 8.7 有简要说明。

XC8100 系列瞄准三个主要的应用:

1. 高级设计语言 (HDL) 和逻辑综合的设计

当用工艺无关方法输入设计时,所开发的结构给出十分高的门利用率和低的成本,今天,当利用 HDL 设计可编程逻辑时,利用率和速度常常要折衷考虑,大多数综合工具最初是为门阵列设计的,它们对 FPGA 和 CPLD 中所用的不同结构是次最佳的, XC8100 结构是专门设计以适合用综合算法产生的逻辑实现,在开发期间,结构仿真用实际的综合产物来运行,当利用逻辑综合时,选择 XC8100 可编程单元和设计库得到成本有效的结果, XC8100 的

软件是十分类似于 ASIC,在布局 and 布线之前综合和估计结构的舍取时给出精确的估算。

2. 通用的逻辑应用,特别是遥远通信和工业控制

XC8100提供高的和可预测的广泛范围逻辑功能的应用,它可以控制有大量同步或大量组合逻辑的设计,因为其结构提供不为专门应用的"可用门",设计可以很快输入,且结果是可预测的,例如,因为没有固定的触发器,没有为把设计放进有效数目的时序单元中的问题,软件只构造相应数目的锁存器或触发器,不需要对触发器,锁存器,三态等分析所做的设计,来决定如何使设计"适配"好。

3. 要求单片 FPGA 的应用

这些包括高保密和高速初始加电运行的应用。

§ 6.7.2 结 构

Xilinx 的可编程门阵列由三个主要的可配置元件组成:可配置单元,I/O 块和互连。可配置单元提供构造设计逻辑的功能元件,I/O 单元提供封装腿和内部信号线之间的接口,可编程互连资源提供连接可配置单元和 I/O 块的输入和输出到相应网线的布线通道。在 XC8100中,用户配置是用一次编程的 MicroVia 反熔丝来建立,以决定单元的逻辑功能,I/O 功能和互连。

1. 可编程单元—可配置逻辑单元(CLC)

XC8100靠配置和互连相同的可配置逻辑单元(CLC)实现组合和时序逻辑,每个 CLC 等效于3.25"LSI 逻辑门",此系数是综合大量的设计首先到 XC8100,然后再到 LSI 逻辑库所得到的。

为了允许将包括组合的、同步的和三态的最宽范围的逻辑能有效地映射到 XC8100的结构,CLC 是内部可编程的,尽管这对其它 Xilinx 的 FPGA 也是真实的,但大多数 FPGA 结构有固定的单元结构和唯一可编程的互连。一个可编程单元保证逻辑会更经常有效地放进一个单元,这个单元被编程为:输入倒相,组合函数选择,同步逻辑内部反馈通道,三态和级联使能,XC8100软件自动地根据用户的设计网表配置此单元,各种积木块意味着对大量的逻辑函数有更高的器件利用率。

每个 CLC 有四个逻辑输入加一个级联输入,一个逻辑输出加一个级联输出,逻辑输出有三态控制,每个 CLC 也连接到全局复位网络,四个输入的每一个可以配置成倒相或非倒相的,所以输入信号的倒相是自由的,和输出无关,单元不必作为倒相器而被浪费,这将降低门的利用率,任何输入可以硬线连接成0或1。

对组合逻辑,CLC 有两个可编程功能,与(AND)的功能或积之和(SOP)的功能,一对 CLC 的输入内部相与,这两个信号则在电路中被组合,可编程为或门或与门,SOP 也可被考虑为两输入多路开关或两输入异或和异或非。

对同步逻辑,配置单元内部反馈路径使任何 CLC 为带 D 和 G 输入的锁存器(ld 基本单元),当 G 为高,锁存器为通过状态,两个输入可配置为高有效或低有效,第二个两单元锁存器(ldc 基本单元)加一个非同步的清除(CLR)输入,所有的锁存器也用全局复位信号控制。

图6.24(a)表示单个 CLC 的不同可能的实现。

XC8100软件自动地通过级联(Cascade)连接组合相邻的两个锁存器产生一个 D 触发器,一个 CLC 是主锁存器,第二个是从锁存器,触发器有一个全局复位,它不牵涉用户的布

线,也有一个主要的非同步清除输入(fdc 基本单元),它也可以有一个取代 D 输入的非同步预置输入(fdp 基本单元),注意到既不是两个锁存器的反馈连接也不是主从连接利用通用布线单元,或有可编程单元在它们的通道中,这使触发器有极佳的速度和稳定的特性,XC8100“半专用”触发器结合了在使用中灵活的特性。

对于三态逻辑,CLC 输入的一个控制在输出通道中的三态缓冲器(bufe 基本单元),数据(Data)和使能(Enable)控制输入可以配置成高有效或低有效,把锁存器和三态功能结合起来得到寄存器堆的能力(lde4基本单元),它可以用来构造 FIFO。

全局复位(Global Reset)

全局复位信号自动地分布到每个 CLC 锁存器,因而到每个触发器,它以两种方式控制,自动地用外部主复位(Master Reset)腿,有较长的延时,和内部用 grst 库基本单元,它是自动在加电时有效的,当有效时,此网线预置所有非同步预置触发器(fdp 基本单元)和复位所有其它的触发器,在锁存器的情况下,清除仅当锁存器是在锁存状态时发生,换言之,复位在通过模式对输出无效。

级联(Cascade)

每个 CLC 有第五个输入为级联,级联输入和输出利用专用布线接到最近的单元,沿整个行扩展,组合两个相邻的 CLC 通过它们的级联连接扩展此功能,对于同步逻辑,级联自动地用于构造触发器;对于组合逻辑,软件级联 CLC 构造在基本单元库中更宽的与和积之和,级联也可以用(andcc 和 sopcc 基本单元)来构造如更宽的译码和快速移位寄存器等功能,级联连接比通常的通道布线有几个优点:

- 1)因为它不需要可编程元件,速度更快,且避免单元输出驱动的延时;
- 2)它有已知的时序;
- 3)它不消耗任何通用的布线资源。

地址译码器

级联也可以用在宽的译码功能,例如,每个配置为四输入与门的四个 CLC 可以馈到一个 CLC 配置为带级联的四输入与(andcx),这个有20个输入的译码器延时小于12ns,利用2CLC 的 and8基本单元产生40个输入译码器也有小于12ns 的延时,如果地址是有效的,闸门延时是约5ns。

2. 可编程互连

大多数可编程逻辑器件必须平衡布线/互连资源与晶片尺寸和成本,但是,XC8100的结构提供小的晶片尺寸有丰富的数量的布线,金属段连线和互连,布线软件有许多选择确保一个设计是完全可布通的。

图6.24(b)是互连的示例图,为一个设计实际利用的网线的走线。在水平通道中,有47根连线,每个单元可以与其连接,有五种分别的类型,其中三种也可水平地连接,以四个单元宽为一块,这些连线为:

- 1)单块长度连线,仅为一个块扩展,它们主要用于此块内单元的连接;
- 2)双块长度连线,向左和向右接到邻近的块;
- 3)四块长度连线,向左和向右为四个块长度的连接,它们可以连接在四个块长度内延伸的任何单元;
- 4)水平长线贯穿整个器件的长度,它们可以连接在一行内的任何单元,这些连线由行缓

冲器选择地使用；

5)对逻辑0和1输入使用常数0。

在垂直通道有两类连线：

1)双块长度连线，类似于水平的双块长度连线；

2)四块长度连线。

除了沿器件的所有行和列贯穿的通道外，在器件的每一边有12根垂直长线(VLLs)，它们可以和任何水平长线连接。

全局网线

全局网线是优化时序或为分布时钟，时间临界和/或高扇出控制信号的低扭曲需要的网线，XC8100有为此目的的专用硬件资源，四个封装腿，GCK1—4，接到四个专用的高驱动缓冲器(bufedge)，驱动上述之外在器件两边的四根垂直长线，如果GCK腿不使用，它们对用户I/O是有效的，CLC的每行也有四个专用缓冲器(bufedge)，为了限制扇出，XC8100软件可以利用CLC作为缓冲器(buflx, buf2x)，这些缓冲器可以由综合软件自动地选择。

3. 输入/输出单元(I/O)

I/O形成器件内部逻辑和I/O焊盘之间的接口，每个I/O由驱动焊盘的可编程输出部分和接收来自数据的数据的可编程输入部分组成，在图6.24(c)表示。除了被接到相同的焊盘，输入和输出部分没有共用的东西，通常，I/O的结构类似于XC4000H高I/O系列的I/O块。

输出部分从互连资源取数据和三态控制信息，它也可以从JTAG扫描电路取数据，输出数据可倒相或不倒相。输出驱动有三种选择：

1)可以永久地禁止，使焊盘为只输入的焊盘(ibuf, dbuf基本单元)；

2)可以永久地有效，使焊盘为只输出的焊盘(obuf, rbuf基本单元)；

3)可以由内部逻辑或JTAG电路控制的三态(obufe, rbufe基本单元)。

三态控制信号可以倒相，允许信号作为低有效输出使能或低有效三态通过，输出是CMOS兼容的，它导致无载干线到干线的信号摆动，每个输出可以单独地配置为两个摆动率选项的任一个，它仅影响下拉的运行。

当编程为“电阻性模式”(rbuf, rbufe基本单元)，下拉晶体管被硬驱动，导致实际上 $< 20\Omega$ 的固定的接通电阻，这个导致最快的由高到低的转换，和高达24mA的吸收电流的容量，电阻模式要求4—24mA直流吸收电流驱动端接传输线和高电容负载($> 200\text{pF}$)，当许多电阻模式输出瞬时地由高转到低，这个配置选项可能导致过量的接地逆跳，用户必须限制每个封装的接地腿的瞬时转换数目。

摆动率限制的配置是“电容模式”(obuf, obufe基本单元)，此模式利用专利未定的摆动率控制方法来减少接地逆跳，高到低的转换如上所述，但对下拉晶体管的驱动当输出电压一达到约1V就减少，这导致下拉晶体管中较高的电阻，一个慢的下降沿，并显著减少接地逆跳，在此模式，输出驱动器吸收4mA，对输出要求小于4mA或对小于200pF的电容负载推荐此模式。

输入部分接收来自焊盘或JTAG电路的数据，并把它传到互连结构上，输入可以全局编程为TTL或CMOS输入门限，有两个输入缓冲器的定时规定，利用ibuf库元件选较快的一个，而用dbuf库元件延时的时序是有效的，dbuf用于数据信号，以保证当数据和时钟从外部腿进到内部触发器时零的保持时间。

每个 I/O 焊盘可以编程为有或无上拉电阻,这个选择是与 IOC 用法无关的,当一个焊盘是未利用的,软件将自动编程上拉电阻防止一个悬浮的输入,上拉电阻是约50K Ω 。

XC8100有和 XC4000类似的 JTAG 的能力,三个差别是:

- 1)XC8100JTAG 的腿 TDI,TDO,TMS 和 TCK 是专用的;
- 2)XC8100支持 JTAG 指令 ID_CODE;
- 3)XC8100的指令字长是10位

◇可编程倒相

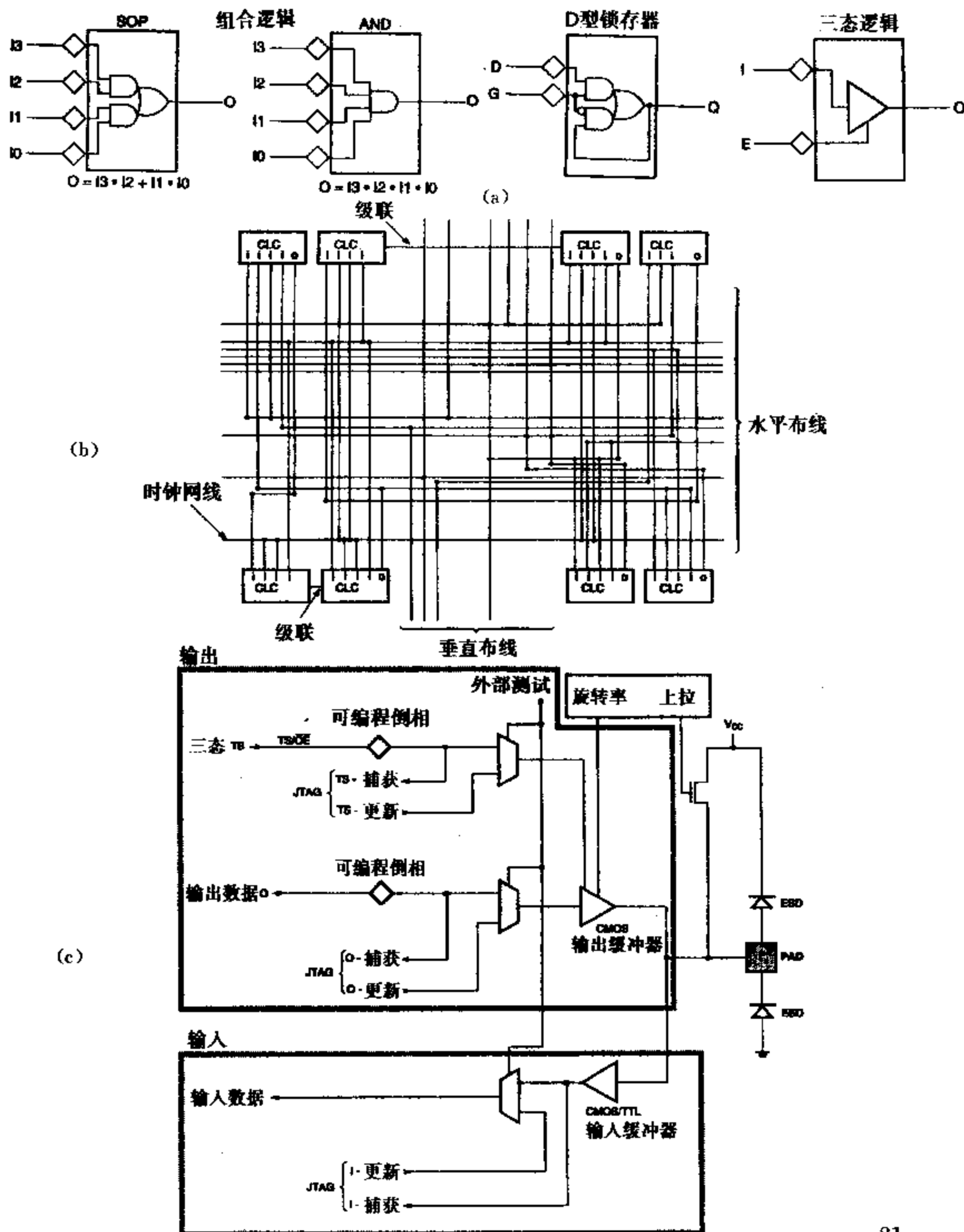


图6.24 XC8100系列三种可编程资源

第七章 Xilinx EPLD 和 FPGA 性能比较

在前面详细介绍了 Xilinx 的 EPLD 和 FPGA 基础上,这一章集中对它们的性能和特点进行多方面的比较,包括 Xilinx 的 EPLD 和 FPGA 之间在性能和应用方面的比较;FPGA 的四个系列包括最新的 XC5000 在性能上相互之间的比较;FPGA 的每个系列中不同型号产品之间的比较;以及从更广泛意义上将它们与 TTL 电路, PAL 及门阵列进行比较;Xilinx 的 FPGA 作为可编程 ASIC 与传统的 ASIC 方法之间从几个方面作了比较。由于可编程逻辑技术发展十分迅速,所有这些比较会有不全面的地方,但是通过比较可以对这个领域有更深入的了解,有助于读者在产品的开发和应用中更好地进行选择。

§ 7.1 FPGA 与 TTL 电路, PAL 及门阵列的比较

为了说明可编程 ASIC 在集成系统应用中优点,用数字系统设计者常用的三类逻辑电路,即 TTL, PAL 和门阵列与 FPGA 相比较。

一、FPGA 与 TTL 电路相比

与 TTL 电路相比, FPGA 有以下的优点:

1. 有更高的密度, FPGA 相当于把 20 至 500 个 TTL 电路封装进一个器件内。
2. 有更灵活的结构, 允许设计者以自己的方法设计具有所需逻辑的系统。
3. 设计集成之后, 可显著地改善系统的可靠性。
4. 可以更容易地进行设计和修改, 不再需要通过绕接板进行中间试验, 而可以通过电路图或逻辑方程输入的方式, 直接由软件来完成设计, 修改设计也是用软件来进行的, 而且一般只要改变芯片内的配置就可以了, 不需要对印制板进行修改。
5. 设计在集成到 FPGA 器件内之后, 只需要较少层的简化和缩小的印制板, 成本也相应减少。

二、FPGA 与 PAL 相比

PLA 或 PAL 作为可编程逻辑器件可以把少数 TTL 电路的逻辑集成起来, 尤其是 PAL 在一些应用中仍广泛使用, 但是 PAL 确有很多的限制, 而 FPGA 相比有如下的优点:

1. 有更多的 I/O 出腿, Xilinx 的 FPGA 最多可达 256 条 I/O 出腿, 不会出现 PLA 中逻辑没有用尽之前, I/O 出腿先用完了的情况。
2. 有更多的触发器, Xilinx 的 FPGA 有从几十个到上千个的触发器, 可以构造许多的寄存器和计数器。
3. 有更灵活的门阵列的结构, 包括有许多系统级的特性, 如在一个芯片内有内部的数据总线和输入寄存器。
4. 更高的集成度, 上述系统级的特性允许将许多 PAL 的逻辑集成进一个 FPGA 芯片内, 数量上可以高达 50 个 PAL 的规模。

5. 更低的成本,芯片集成度提高,使印制板的面积极大地减少,从而降低了系统的成本。
6. 更少的功耗,Xilinx 的 FPGA 运行在最高速度时,一般耗费不到100mW 的功率,何况还有降功率模式,愈加减少了功耗。
7. 更高的可靠性,系统级有较少的零件在印制板上,又有较低的功耗,直接提高了系统的可靠性。
8. 可再编程和可再利用,与一次编程的 PAL 不同,而与 GAL 相似,是可再编程的,可反复使用。

三、FPGA 与门阵列相比

Xilinx 的 FPGA 器件从掩膜门阵列借鉴了许多结构上的特点,但仍有许多超过门阵列的优点。

1. FPGA 有更低的设计和开发成本,当前,在实验室的办公桌上利用 PC 机或工作站,就可以进行整个设计,直到设计被配置进 FPGA 的器件内,而开发软件也是相对低的成本,能适合多种的器件和设计任务。

2. FPGA 是可编程的,不存在门阵列所需的不可重复的工程投资,门阵列每次修改设计用于定制掩模的费用可以达到几千到几万美元,FPGA 都不需要,因此风险就减少到最低限。

3. Xilinx 的 FPGA 是可再编程的,利用其功能很强的开发软件可以很快和很容易地改变设计,不受次数限制,不需支付费用,更省去了掩模门阵列投片制造带来的时间延误。

4. Xilinx FPGA 的可再编程特性,使其可以做到器件100%的出厂前测试,保证了器件的质量。

5. 对大量的产品,为降低成本,可从 FPGA 很容易转到固定连线的门阵列。

§ 7.2 Xilinx 的 EPLD 和 FPGA 性能比较

Xilinx 的可编程 ASIC 器件为设计者提供了用不同的方法来解决设计的问题,正确地选择器件是十分重要的,首先考虑选择 EPLD 和 FPGA 之间的一些取舍。

对于 EPLD 有可预测的性能,而 FPGA 的性能是和布线有关的。当一个设计放进一个 EPLD 时,此设计的速度和时序等性能是可预测的,但有一个上限;用 FPGA 进行设计时,可以对设计进行优化到十分快速和紧凑。从功耗的要求出发,如果系统要求低功耗,最好选择 FPGA。在有大量寄存器的应用中,则需要利用 FPGA,但是,EPLD 有相当数量的触发器来实现快速状态机和长的高速计数器。

EPLD 被设计成做宽的快速译码,并有宽的扇入,因为 XC4000 系列也有片内的专用译码能力,所以与设计的规模和性能等技术条件有关,可以选任一种。EPLD 有基本的与/或逻辑结构,适合译码和计数等,FPGA 实际上没有连接成任何特定的逻辑结构,提供了更大的灵活性。

EPLD 的软件更类似于 PAL 的软件,而 FPGA 的软件更接近于标准的 ASIC 工具。

许多应用可以用 EPLD,也可以用 FPGA,究竟选用哪一种,取决于系统的要求和特定的设计要求。表7.1给出 Xilinx 的 EPLD 和 FPGA 之间的比较。

表7.1 Xilinx EPLD 和 FPGA 之间的比较

		EPLD		FPGA	
		XC7200	XC7300	XC20/30/3100	XC4000/A/H
结构		类似 PAL 与或平面 宏单元和乘积项	先进 PLD 高速和高密度 功能块于一体	类似门阵列 含许多小功能块	类似门阵列 含许多小功能块
逻辑容量		36—72宏单元 集成4—8个 PAL22V10	36—144宏单元 集成4—16个 PAL22V10	600—7500门 集成 TTL MSI PLD	1600—25000门 集成 TTL MSI PLD RAM
设计时序		固定,类似 PAL 60MHZ 多数应用可预测	固定,类似 PAL 167MHZ 多数应用可预测	类似门阵列,与应用有关 可>100MHZ 一般:25—40MHZ(3000) 50—85MHZ(3100)	类似门阵列 与应用有关 可>100MHZ 一般:30—50MHZ
I/O 数		36—72	38—156	58—176类似门阵列	64—192类似门阵列
触发器数		72—144	18—234	122—1320	256—1536,18KRAM
功耗	静态 动态 典型	0.5—1.25W 0.75—1.5W	0.4—2.0W 0.5—2.25W	很低,mW 级 与应用有关 0.12—1.0mW	很低,mW 级 与应用有关 0.25—2.0mW
系统性能		算术进位逻辑 100%互连保证 ALU 每个宏单元	与 XC7200一样 增加: 超前进位 高输出驱动 双功能块	两个全局时钟缓冲 编程输出转换速率 内部三态总线 Power-down 模式 8mA 输出驱动	八个全局时钟缓冲 编程输出转换速率 内部三态总线 RAM 为 FIFO 寄存 快速进位逻辑为算术运算 宽位译码器 12mA 输出驱动 每路24mA
工艺		CMOS EPROM	CMOS EPROM	CMOS SRAM	CMO SRAM
编程方法		PROM 编程 OTP 或紫外线擦 配置在芯片上	PROM 编程 OTP 或紫外线擦 配置在芯片上	编程在电路中 四个模式 配置存储在外部	编程在电路中 六个模式 配置存储在外部
再编程性		紫外线擦除后	紫外线擦除后	在毫秒内可再编程 在电路中可再编程	在毫秒内可再编程 在电路中可再编程
厂内测试		是	是	是	是
主要应用		复杂状态机 复杂计数器 线和外设接口 存储器控制	高速图形 多口存储器控制 高速总线接口 50MHZ 16位累加	简单状态机 通用逻辑替代 可再编程应用 电池供电逻辑 3V 运行 超高速数器	简单状态机 复杂逻辑替代 板级集成;总线接口 可再编程应用 边界扫描可测性 加法,比较,计数 RAM 应用; FIFO,缓冲器

§ 7.3 XC4000与XC3000比较

下面以表7.2给出 XC4000系列主要的新特性

表7.2 XC4000系列主要的新特性

CLB	1. CLB 有两个独立的四输入函数发生器, 第三个函数发生器用第九个输入来组合前两个的输出, 所有函数输入是可交换和存取的, 无一相互排斥。 2. CLB 有十分快的算术进位能力。 3. CLB 的函数发生器的查找表也可以用来作为高速 RAM。 4. CLB 的触发器有非同步置位和复位。 5. CLB 有四个输出两个触发器的, 两个组合逻辑的。 6. CLB 的连线对称地位于四周。
IOB	1. IOB 有更多的通用时钟极性选择。 2. IOB 有可编程输入建立时间。 长建立时间避免潜在的保持时间的问题; 短建立时间改善了性能 3. IOB 有长线通过它拥有的 TBUF 存取 输出仅仅是 n 沟道的, 低的 V_{OH} 增加了速度, 输出不嵌位到 V_{CC} XC4000 可以对地加倍高达 24mA 的耗尽电流 XC4000A/H 每个可达 24mA, 成对可达 48mA
其它	在 I/O 中支持 IEEE1149-1 型边界扫描 LCA 的四周有宽位译码器 增加了互连资源的数目 所有 CLB 的输入和输出存取到大多数互连线 简化了开关矩阵来增加速度 八个全局连线可以用于时钟或分布逻辑信号 TBUF 输出配置是更少约束的通用和三态的控制 程序是单功能输入腿, 不与任何其它腿重迭 INIT 腿也起配置错误输出的作用
配置	已经增加了八位外设同步模式 外设非同步模式已经改善了握手 起动可以同步到任何用户时钟(这是一个配置选择) 无 Powerdown, 用一个不复位任何触发器的全局三态输入代替 无片内晶体振荡器用的放大器 配置的位流包括 CRC 错误校验 配置时钟可以增加到 >8MHz 配置时钟是全静态的, 不对最大的低时间作限制 Readback 或者不管触发器的内容(避免对掩膜的需求) 或者在开始 Readback 时刻对所有触发 Snapshot Readback 有和配置相同的极性, 可以被放弃不用

§ 7.4 XC4000H 与 XC4000 的比较:

1. XC4000H 的 I/OB 的数目大约是 XC4000 的两倍

2. XC4000H 显著地改善输出转换速率的控制:电阻负载意味着所有到地的通路有很强的下拉,有连续地耗散 24mA 的能力。如果许多输出瞬时地接通,最终的接地限制可能是不适合的。

电容负载或 Softedge 意味着更灵活的在到达地时减少强度的下拉。它可以在 V_{OL} 时仅仅耗散 4mA,与在驱动电容负载时不同,好处是在几个输出瞬时接通时减少了接地的限制。

在 XC4000 中,限制输出的转换速率减少了接地回跳,但也引入了较显著的附加延时。在 XC4000H 中,电容负载模式时的附加延时通常并不显著。

3. 在 XC4000H 系列中所有的输入和输出触发器已经被取消,用 CLB 中的触发器代替。

4. 在 $V_{OL} = 0.5V$ 时,保证输出可以耗散 24mA,相比之下,XC4000 系列在 0.4V 时耗散 12mA。

5. 沿边译码器输入的数目。

6. 每个输入可以作为以下之一单独地配置:

—TTL 兼容,利用 n 沟道晶体管为上拉和下拉二者

—用减少 V_{OH} 的推拉输出电路结构

—CMOS 兼容,与 XC2000 和 XC3000 相同,即 n 沟道下拉,p 沟道上拉, V_{OH} 接近于 V_{CC}

7. 每个输入可以单独配置,或者 TTL 兼容门限 (1.2V),或者 CMOS 兼容门限 (0.5 V_{CC})。每个输入可以配置成倒相或不倒相。

8. 可编程输入和输出电平在任何出腿上的任何组合是可能的,甚至在相同 I/O 腿上 TTL 输出和 CMOS 输入的双种组合。

9. 输出三态运行是由两输入多路开关控制。

10. 在配置过程结束之后,当它们从三态转到它们的实际电平时,输出的第一个活动总是 Softedge 模式,这样防止当所有输出瞬时接通时,潜在的接地限制问题,几个 ns 之后,每个输出假设由它的配置决定电流容量,这个软唤醒运行对用户是透明的。

结构上除了 I/O 接口之外,XC4000H 系列是与 XC4000 系列相同的,可配置逻辑块 (CLB) 的矩阵是通过一层灵活的布线资源被互连的。

XC4000 系列所具有的很强的系统集成特性,如芯片内 RAM,专用快速进位和宽的译码器等在 XC4000H 系列中被保留。

XC4000H 系列与 XC4000 系列相比几乎加倍了 I/O 出腿的数目,一个对 I/O 应用有吸引力的特性,输出驱动被重新设计,而使驱动能力更强和更灵活。

§ 7.5 XC3000A/L 及 XC3100 与 XC3000 的比较

一、XC3000A 与 XC3000 比较

XC3000A 提供了以下超过 XC3000 系列的扩充特性:

XC3000A 系列有附加的互连资源来驱动激励水平长线的 TBUF 的 I 输入。CLB 时钟使

能输入可以从第二条垂直长线驱动。这两个扩充在水平长线被用于数据总线时将导致更有效和更快的设计。

在配置期间,XC3000A 器件校验在相应位置中位流的停止位格式,出现任何错误就终止配置,并将 INIT 拉低。

当配置过程完成,器件启动成用户模式时,输出的第一个动作是自动地限制转换速率,这个特性称作软启动,避免了当所有的输出被瞬时接通时潜在的接地限制。启动之后,各个输出的转换速率与 XC3000系列一样是由各自的配置选择决定。

所以 XC3000A 与 XC3000相比,功耗下降约20%,速度增加约15%左右,而价格二者相当。

二、XC3000L 与 XC3000比较

XC3000L 系列对于由通常的3.3V 供电的运行是最优的。

在数据表中列出的除电气和定时参数外,XC3000L 系列在所有方面是与 XC3000A 系列是相同的,是 XC3000系列的高级集合。

Xilinx FPGA 的运行电源消耗几乎是与动态无关,它随供电电压的平方变化。对于已知的复杂性和时钟速度,XC3000L 仅仅为等效的 XC3000A 器件使用的功率的44%。当它用于电池供电装置中,XC3000L 系列被设计成最低可能的掉电和准电流消耗。

在混合供电系统中,由3.3V 规范电压供电的 XC3000L,可以直接用类似 TTL 的输入门限驱动任何器件。当5V 电源驱动 XC3000L 时,1千欧限流电阻或分压器要求来防止过多的输入电流。

和 XC3000A 系列一样,XC3000L 也具有和 XC3000A 相同的超过 XC3000系列的扩充特性。

三、XC3100与 XC3000比较

XC3100和 XC3100A 系列相对于 XC3000和 XC3000A 系列是性能上最优的,而两个系列是配置的位流和出腿兼容的,但 XC3100和 XC3100A 系列把系统性能扩充到80MHZ 甚至更高。表7.3提供了两个系列的比较:

表7.3 XC3100与 XC3000系列比较

	速度	T ₁ LO	最大触发频率
XC3100	-1	1.2	340 270 230 190
	-2	2.2	
	-3	2.7	
	-4	3.3	
	-5	4.1	
XC3000	-125	5.5	125
	-100	7.0	100
	-70	9.0	70

XC3100系列允许 XC4000速度档次,按照内部的逻辑块延时表示器件的性能。增加了22×22阵列规模的 XC3195。

XC3100A 与 XC3100相比时,有 XC3000A 超过 XC3000所具有的完全相同的扩充特性。

§ 7.6 XC5200与 XC4000的比较

XC5200系列是 Xilinx 推出的低成本,高容量的 FPGA 新系列,在已有的三种成功的 SRAM FPGA 系列所获得的经验的基础上,XC5200系列把更优越的性能加到高密度可编程逻辑设计中。新的 XC5200系列的结构对三层金属和 $0.6\mu\text{m}$ CMOS SRAM 工艺的优化,使硅片的效能已达到惊人的改进。与通常的 FPGA 相似,XC5200由可编程 I/O 块,可编程逻辑块和可编程互连组成,但与其它 FPGA 不同,XC5200的逻辑和局部布线资源组合成灵活的多用功能块(VersaBlock),多用功能块中的局部互连矩阵(LIM)可保证100%的局部连通性,并有24个双向结点连到通用布线矩阵(GRM),通用布线通过 GRM 接到多用功能块。其结构图如图6.23所示。

从表面上看,XC5200是完全类似于 XC4000的,两个系列都利用 CMOS SRAM 工艺,带有共享输入的四输入查找表,有专门的快速进位链,在 I/O 块中的专用边界扫描逻辑,两者的器件引脚是兼容的,其引脚名和引脚位置是相同的,XC5200器件提供与 XC4000相同的配置选择,它们可以和 XC4000器件在一个菊花链中混合在一起。

但是,这两个系列之间也有显著的差别:

1. XC5200系列的查找表不可以用作 RAM;

2. XC5200系列提供专门的进位逻辑,但与 XC4000系列不同,和是在相邻列的一个附加函数发生器中产生,因此一个 XC5200的器件利用两次作为加法器,减法器,累加器和一些计数器的许多函数发生器,但要注意,可加载的加/减计数器在两个系列中要求相同数量函数发生器;3. XC5200系列没有专门的沿边宽译码器,不像 XC4000系列,XC5200的进位逻辑可以用来级联函数发生器以实现诸如宽的与及或函数;

4. C5200系列包含逻辑和称为多用功能块(VersaBlock)的局部布线资源灵活的耦合,XC5200的多用功能块包含可编程逻辑块(LIB),局部互连矩阵(LIM)和到邻近多用功能块的直接连接;

5. XC5200系列的可配置逻辑功能块(CLB)大致等效于两个 XC4000系列的 CLB,每个 XC5200的 CLB 包括四个四输入的函数发生器和四个寄存器,它们作为四个独立的逻辑单元(LC),每个函数发生器的输出可以作为 CLB 的输出产生,和/或驱动触发器的 D 输入端,一对逻辑单元可以组合形成五输入的函数发生器;

6. 每个 CLB 有四个直接的连线通路,每个 LC 一个,这些通路可以提供额外的数据输入线或作为局部布线,不消耗任何逻辑资源;

7. XC5200系列有一个全局复位,但 XC4000系列有一个全局置位和一个全局复位;

8. 与 XC4000不同,每个寄存器可以配置为沿触发的 D 触发器,或配置为透明的电平锁存器;

9. 没有专用 IOB 触发器,但有到邻近 CLB 的快速直接连接;

10. 三层金属工艺允许布线资源显著地改进,每个 XC5200的多用功能块有完全的 CLB 内部布线,局部互连矩阵(LIM),并提供四个直接布线连接到四个邻近 CLB(东,西,南,北)的每一个,因此任何函数发生器或触发器有到其它19个函数发生器或触发器不受限的连通性;在自身 CLB 中的三个,和邻近 CLB 中的16个,这些直接连接不和通用布线资源竞争;

11. 每个 XC5200 的三态缓冲器 (TBUF) 可以驱动多达两条水平长线, 而每个 XC4000 的 TBUF 仅到达一条水平长线;

12. 在中心的 CLB 阵列的外圈和 IOB 环之间有一条专门的通道, 多用环 (VersaRing), 在克服由 I/O 腿提前锁定引起的问题时显然有用, 如图 7.2 所示;

13. 对 XC5000 系列的长线没有内部的上拉电阻。

§ 7.7 EPLD 与 FPGA 设计方法的比较

用可编程 ASIC 器件设计数字系统时, 通常要进行逻辑级和寄存器转移级的综合, 由于 EPLD 和 FPGA 二者在结构上的差别, 在这两级综合的方法也有所不同, 这一节就逻辑级综合中的逻辑优化和寄存器转移级中有限状态机设计两个方面二者的差别作一简单的比较。

§ 7.7.1 逻辑综合

有几种方法来实现逻辑优化, 最通常用的方法是把综合过程分成两个阶段: 工艺无关的综合阶段和跟随的工艺映射阶段, 工艺无关阶段力图产生逻辑电路的最优抽象表示, 工艺映射阶段从库中选一组门电路来实现此抽象的表示, 同时优化面积, 延时或二者的兼顾。

最常采用的两种逻辑综合的策略可分别用于 PLD 和 FPGA:

一、两级逻辑最小化:

对于可编程逻辑器件的两级与阵列和或阵列结构, 常采用两级逻辑最小化, 它的目标是瞄准规则的 PLD 的形式找出布尔函数的最小实现, 在最小化步骤之后, 两级逻辑可以直接在 PLD 器件中实现。

n 个变量的布尔函数可以用它的最小项的逻辑和来表示, 这些最小项使布尔函数的输出为 1, 对应的输入变量组合构成它的 on-set 集, (使输出为零的输入变量组合构成 off-set 集), 布尔函数的 on-set 集也可以表示为逻辑的积之和形式, 而且积之和的形式可能比最小项的和形式要紧凑得多, 如果把从不出现的输入变量组合所对应的值称为未定义的值, 即 don't care set 或 dc-set, 两级逻辑优化的过程是选择最小集合来复盖 on-set 集, 和部分可能的 dc-set 集。

二、多级逻辑优化:

由于 FPGA 器件可以实现相对大的多级逻辑模块, 常采用多级逻辑优化。多级逻辑实现已变得越来越重要, 因为有许多组合逻辑块不适合两级逻辑实现。多级逻辑电路常用布尔网络表示, 然后为优化网络结构要调整这些网络以满足给定的约束, 多级逻辑优化是要检测出对网络的几个结点为共同的逻辑部分, 将其提取出来, 并仅用一次实现它们, 此过程可看作对网络的每个结点产生一组适合的优选的表达式, 再检测对几个结点共同的那些优选者, 从而实现对逻辑的优化。

§ 7.7.2 状态机设计

状态机是用可编程逻辑实现的最通常的功能之一, 在各种数字应用中, 特别是控制器中被广泛采用, 在任何不是纯粹组合逻辑的数字系统中, 时序电路或有限状态机设计是基本

的,构成寄存器转移级的综合问题。有许多综合或编码状态机的方法,一种是符号状态机,它不参考存储在状态寄存器中对不同状态的值,把这些值应该是什么的解释留给软件去做,它能够对正采用的结构或正产生的状态机的规模以最有效的方案实现,在符号状态机中定义的一切是由输入信号影响其转换的状态,在每个状态期间的输出值和某些情况的初始状态之间的关系。另一种是编码状态机,它要求定义和符号状态机相同的信息,除此之外,还要求设计者定义为每个状态的状态寄存器的值。

由于PLD和FPGA两者在结构上的差别,需要用不同的状态机方案使状态机的结构匹配器件的结构,而PLD与FPGA在结构上的主要区别是CPLD有数量有限的触发器和乘积项丰富的结构,适合高编码状态序列的状态机;而高门数的FPGA有扇入有限和触发器丰富的结构,适合于每一状态用一个触发器来构造符号状态机的“一个有效”的编码方案。

一、二进制编码

二进制编码又称为最大编码,二进制编码方法利用二进制计数器以节省编码状态的触发器,例如,16状态的状态机将只需要四个触发器,因为二进制编码值在0000和1111之间。所以二进制编码是用最小数目的触发器来编码状态机,每个触发器代表二进制数的一位,要利用触发器的最大能力,但为了编码每个状态要求更多的组合逻辑,因此,当状态机的函数是最大编码的时候,PLD的结构对其最适合,因为CPLD有丰富的乘积项来构成复杂的组合逻辑,满足状态机的要求。

用高编码状态序列实现的状态机通常有许多宽输入逻辑函数来说明输入,并译码状态,所以它的性能当状态增加时下降,因为它要求更宽和更深的逻辑。标准的有限状态机的设计程序是将时序电路的要求作为状态图列出,分配变量来识别每个状态。由状态图则可画出由所用变量的组合逻辑及将要产生的下一状态矢量的状态转移表,然后写出下一状态矢量和输出的表达式,对触发器输入的简化方程式,最后用逻辑电路来实现。

一个好的状态机设计必须优化组合逻辑的数量,到寄存器的扇入和寄存器之间的传播延时,但是这三个因素是相互有关的,必须在它们之间进行取舍,例如,为了增加速度,必须减少逻辑级数,但较少的逻辑级数导致较宽的组合逻辑和较高的扇入;另一例子是为了减少门数必须分解因子,即提取共享项和利用分别的逻辑来实现,分解因子减少逻辑数,但增加寄存器之间的逻辑级数,并使电路速度降低。通常,高编码状态机的性能当状态数量增加时而降低,因为对每个增加的状态要求更宽和更深的编码。

二、一个有效的编码(one-hot encoding)

一个有效的编码采取与二进制编码相反的方法,在一个有效的编码中单个状态寄存器决定一个状态,在任何时间,仅仅一个触发器是有效的,或者所有的触发器可能都是无效的,所以对于相同的16状态的状态机要求16个触发器来实现,但一个有效的编码可以显著减少用于实现状态机的组合逻辑的数目。与最大编码设计趋于要求许多高扇入逻辑函数来说明输入不同,一个有效的编码利用较简单的译码过程,因为每个状态有它自己的寄存器或触发器,因此,状态机是已经“译码的”,机器的状态可以简单地找出哪个触发器是有效的来决定,这个过程减少组合逻辑的宽度,整个状态机要求寄存器之间较少的逻辑级,这样可减少它的复杂性,提高它的速度。

FPGA 是高密度可编程芯片,包含一个大的由用户可编程的互连资源围绕着的用户可配置的逻辑块阵列,组成阵列的 FPGA 逻辑块输入数量有限,所以用一个有效的编码可限制扇入到一个逻辑块的数量,而且用一个有效的编码可以产生要求资源较少和比二进制编码性能高的状态机,它只要求一个输入来译码一个状态,使得下一状态逻辑简单,很好地适合于有限扇入的 FPGA 结构,此外,最终汇集的触发器是类似于移位寄存器的结构,可以有效地在 FPGA 器件内布局 and 布线,一个有效编码状态机的速度即使当状态的数目增加时几乎保持常数,对一个有效编码状态机来说,为建立下一状态的逻辑是简单的。

如图 7.1 所示,简单的状态图常用由圆和弧的泡沫图来表示,在每个圆形的“泡沫”中,横线上的标记是状态名,横线下的标记是当此状态有效时要求的输出,反馈回同一状态的弧线是缺省的通道,仅当无其它条件通道为真时缺省通道才为真,每个条件通道是由在转移到下一状态之前必须存在的相应的逻辑条件标记,逻辑条件中的变量构成所有的逻辑输入。

在状态图中初始或加电条件必须仔细检查,在加电时,一个状态机总应进入一个初始的已知状态,对 Xilinx 的 FPGA 系列,在加电时自动复位所有的触发器,为了在加电时建立一个初始状态,初始状态的触发器的输出被倒相,为保持逻辑的一致,到触发器的输入也要倒相。所有其它的状态利用带异步复位输入的 D 型触发器,异步复位输入的的目的是为避免非法状态。

一旦初始条件建立,可以配置下一状态转移逻辑,为了做到这一步,首先检查各个状态,然后计算进入状态的条件通道数目,如果缺省条件将保留在相同的状态,则加一个额外的通道;其次,建立一个输入数等于在第一步决定的条件通道数的或门;第三,对每个或门的输入建立一个前一状态和它的条件逻辑的与门;最后,如果缺省条件将保持在相同的状态,建立一个当前状态和所有可能离开当前状态的条件通道取反的与门。对于图 7.1 所表示的状态图实现各个状态转移的逻辑如图 7.2 所示。

在定义了所有的状态转移逻辑之后,下一步骤是决定输出,如图 7.1 所示,三个输出信号 Single, Multi 和 Contig 对应三个主要的输出类型:最简单的情况是在一个状态期间要求的输出(Single),在多个邻接状态期间要求的输出(Contig)和在多个非邻接状态期间要求的输出(Multi)。一个有效编码方法使得定义输出容易,在许多情况下,状态触发器是输出,如 Single 的输出也是触发器的输出,有一些减少逻辑复杂性的技术来实现第二种输出,而第三种输出通常用有效状态的平滑译码来解决,对 XC3000 系列来说,译码五个状态不超出其逻辑块的输入能力。

在状态机结构中一个公共的问题是防止非法状态中断系统的运行,非法状态存在于状态机功能未定义或无效的场合,在一个有效编码的方法中,非法条件出现在两个或多个状态同时有效的时候,将状态机的所有输入与主时钟信号同步是防止非法状态的方法之一,当非同步输入变化十分接近于时钟沿时,不会出现非法的转换,额外的同步对有丰富触发器结构的 FPGA 是理想的。

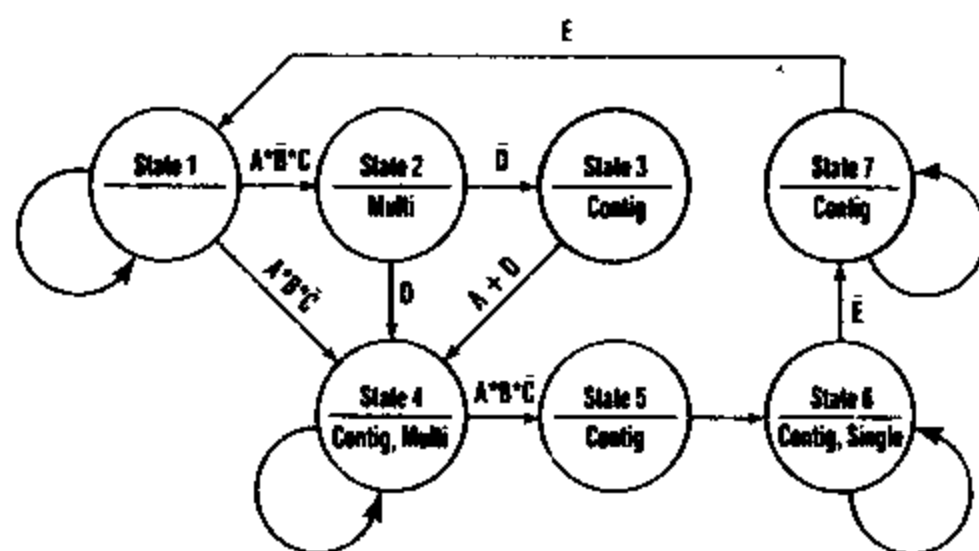


图 7.1 一个有效编码的状态机

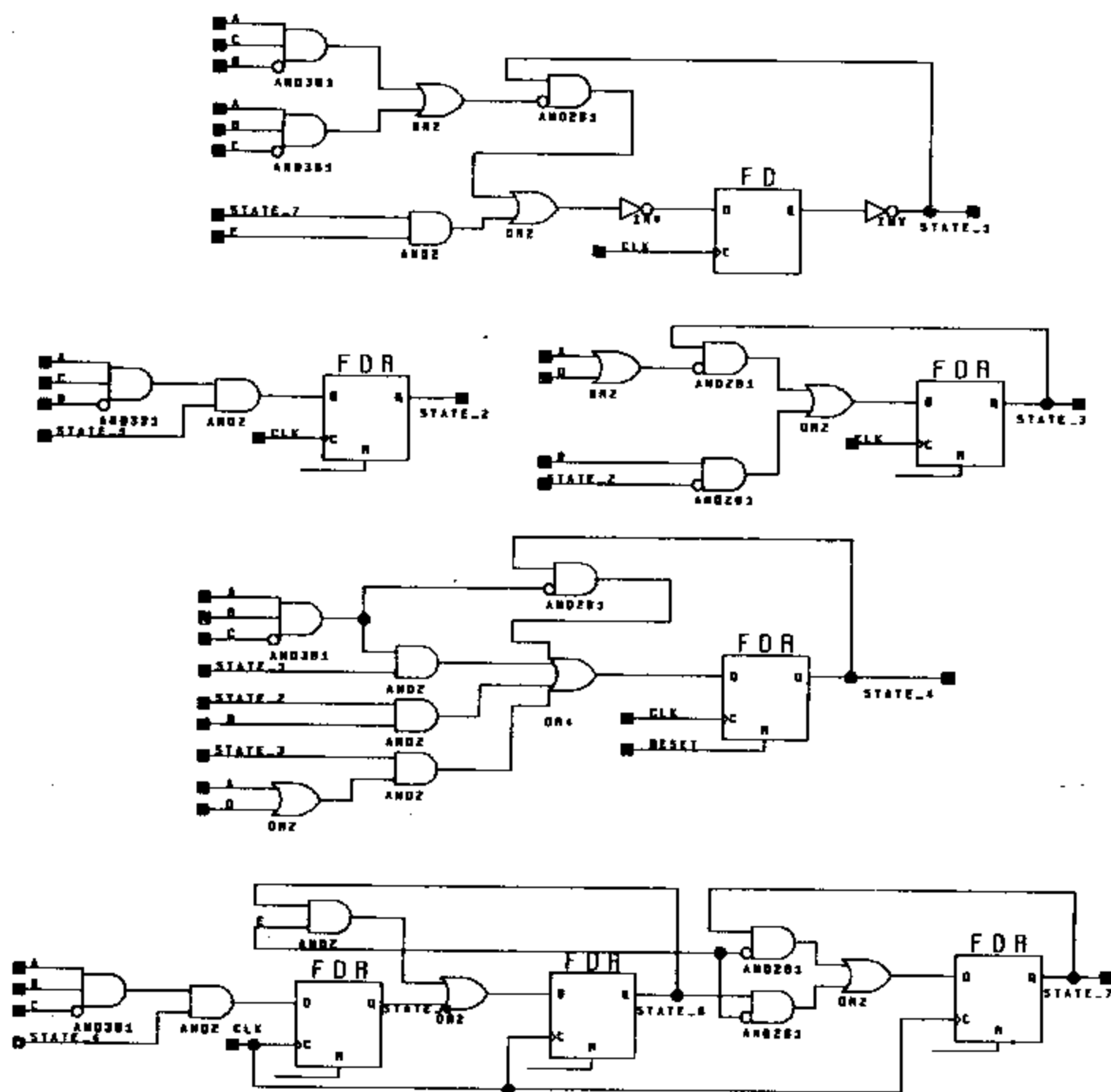


图 7.2 图 7.1 状态机的各状态转移逻辑

第八章 Xilinx 的 FPGA 开发系统

§ 8.1 设计流程概述

为了使用 XilinxTM自动 CAE(XACT[®])软件,本章描述 XACT 基本开发系统,有了开发系统能够在数小时内输入、实现、验证 LCA 设计。

在开始进行 LCA 设计之前,应先阅读一下有关 Xilinx 现场可编程门阵列(FPGA)产品的介绍,这节在前几章的基础上,首先介绍 Xilinx FPGA 的设计流程,对包括设计输入,设计实现及设计验证的 LCA 整个设计过程做个概述。

图8.1已表示出了 Xilinx FPGA 的设计流程。流程的三个组成部分是设计输入,设计实现及设计验证。表8.1中为设计流程中常用的术语以中文和英文对照的方式列出,以供参考。

表8.1 设计流程中常用的术语

术语	说明
Schematic Entry	用原理图符号输入
Text-based Entry	用设计语言输入
Optimization	把与器件无关的逻辑描述或行为逻辑描述转化为能在 Xilinx FPGA 上有效实现的形式
Mapping	把设计逻辑映射到 Xilinx FPAG 的资源中
Placement	把在映射中建立的设计块安排在 FPGA 中的特定位置
Routing	指定连接路径化为能装载到 Xilinx FPGA 中的位流文件
Back-annotation	把逻辑图与已布好的网线一一映射,使相同的输入激励能用作功能和时序仿真
Simulation	输入激励对设计逻辑和时序进行软件仿真
Static Timing Analysis	分析信号路径以保证延时是可接受的
In-circuit Verification	把设计装入 Xilinx FPAG 后在系统中进行测试

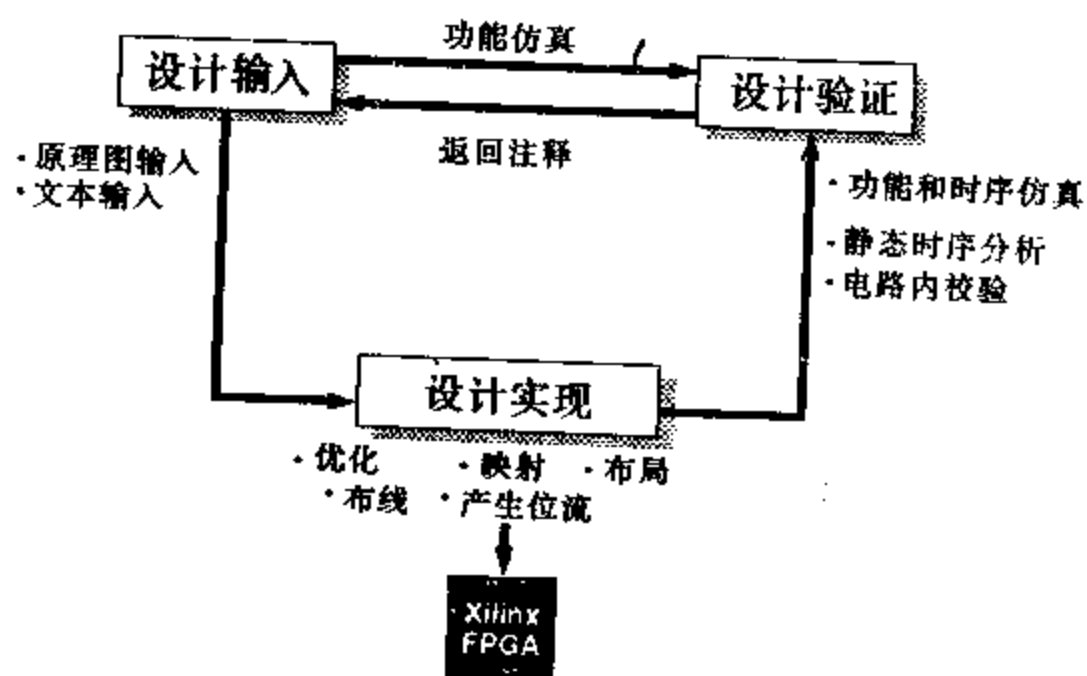


图8.1 Xilinx FPGA 设计流程

LCA 设计流程是由 Xilinx Design Manager(XDM)控制的三步过程,包括以下阶段:

- 设计输入——在设计流程的这一步,可以用 xilinx 支持的原理图编辑器或硬件描述语言(HDL)建立起设计。
- 设计实现——通过对设计分割、布局、布线,可以把在设计输入阶段建立的设计文件转换为 LCA 文件格式。然后可以从 LCA 文件建立位流(bitstream)文件,并且写到 PROM 或 EPROM 中以便接下来对 Xilinx 器件编程。
- 设计验证——使用仿真器、Xilinx XChecker™电缆线,或 Xilinx 加载线,确保设计以恰当的功能和时序工作。

整个流程是一个输入、实现、验证的递归过程,直到实现的设计正确及完整。XACT 基本开发系统能够做到使整个设计流程快速地递归,并且由于 LCA 器件可以无限次数的编程,因此在电路级调试设计时不会报废器件。图8.1是一个设计流程的图例。

一、设计输入

有多种方法可以用来输入 LCA 设计。最普遍的就是用 Xilinx 支持的第三方原理图编辑器。可以用的设计输入手段有以下两种:

- 原理图编辑器——这些能够做到原理图文件中使用常规门电路及宏单元来进行逻辑设计。在原理图编辑器内,也可以规定其它逻辑文件(例如 ABEL 状态机)。
- 硬件描述语言——除原理图编辑器外,也可以把整个设计从 Xilinx 支持的硬件描述语言(例如 ABEL—HDL)转换成 XNF(Xilinx Netlist Format)文件。

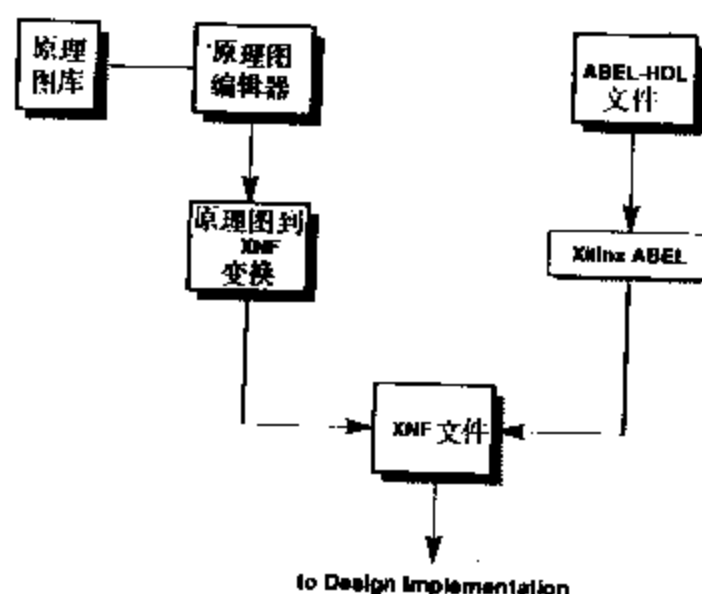


图8.2 设计输入说明

设计输入阶段的目的是产生一个 XNF 文件,它是设计实现阶段的开始。可以利用不同设计输入方法为组合多个逻辑源文件成一个平整的网表文件,以便进行设计实现。

二、设计实现

LCA 设计流程的这一部分是设计开发过程的核心部份,设计实现意味在所选的 FPGA 中物理地实现所需的逻辑,一般,设计实现的工具对给定的结构是唯一的,因为好的实现工具应开拓所选 FPGA 的所有结构资源。设计实现阶段根据设计的结构而定,要求一个或多个设计输入阶段产生的 XNF 文件,这些文件描述了逻辑功能和相互的连接。设计实现过程

由三个相互有关的任务组成：

- 映射——映射是把设计分割到构成 LCA 的逻辑功能块 (CLB), IOB 和其它资源中的过程。某些逻辑功能, 如 D 触发器和 I/O 缓冲器可直接映射到相应的 FPGA 资源中; 而组合逻辑等要利用映射算法在 CLB 中实现, 这些映射算法可以有很大的变化, 取决于 CLB 的结构, 用户可选最优密度映射或最优性能映射, 前者利用最少的 CLB 映射, 后者的映射使临界通道的逻辑级数最小, 有时映射算法也重视布通率的考虑。XACT 的 XNF Map 程序对 XC2000, XC3000 及 XC3100 系列设计进行逻辑分割。图 8.3 示出了 XC3000 系列设计进行逻辑分割的过程。

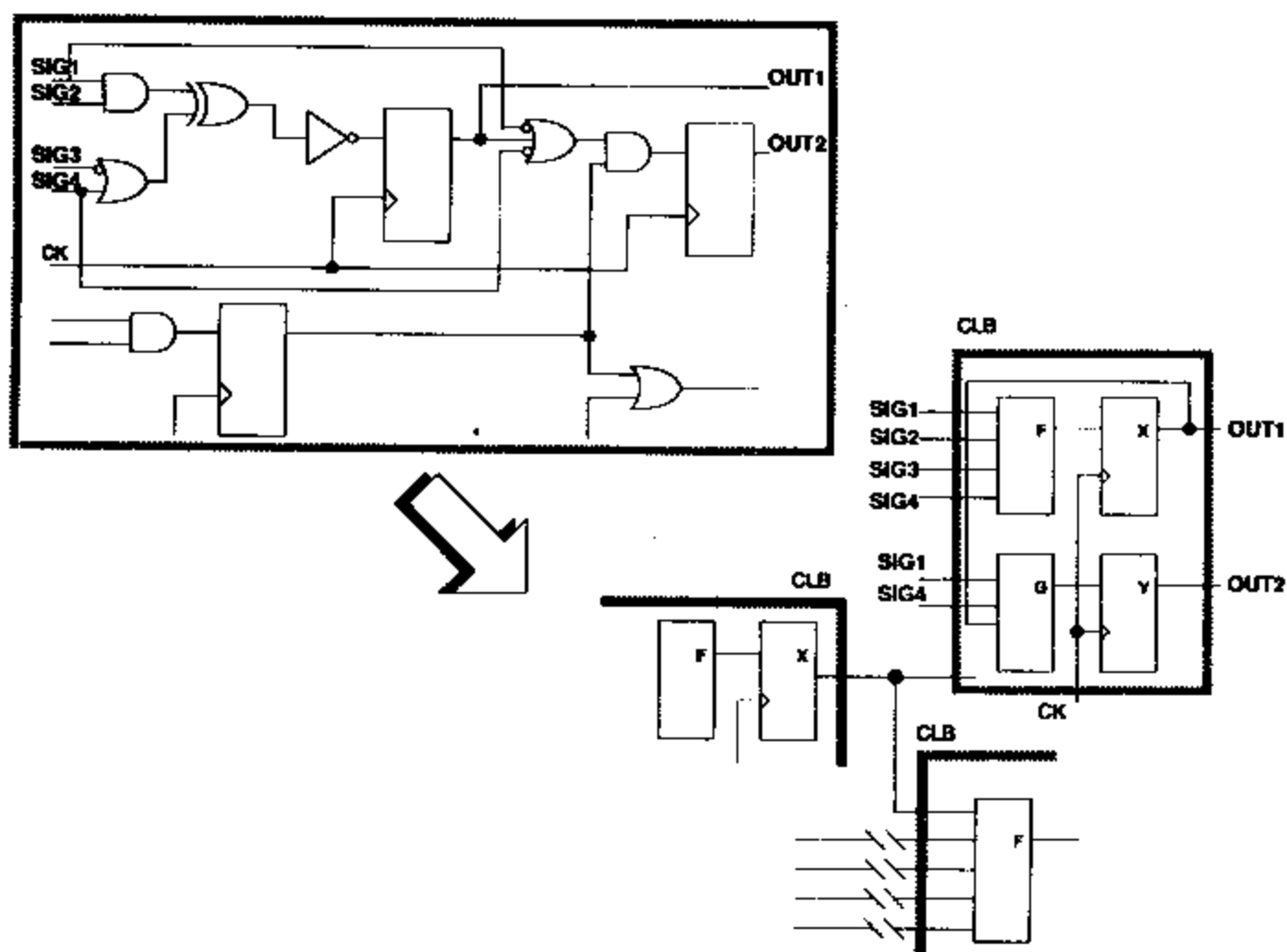


图 8.3 XC3000 设计实现说明

- 布局——布局包括从映射步骤取出定义的逻辑和 I/O 块, 把它们分配到 FPGA 内的物理位置。布局利用以下三个基本算法的组合: 极小截 (mincut), 模拟退火和一般的受力方向张弛 (GFDR), 例如, 对于 XC4000 的布局利用变更的极小截作初始的布局, 再用 GFDR 算法改善布局, 算法的目标是依据各种代价函数决定最优的布局, 如代价函数为最终的功能块互连的总网线长度或沿一组临界信号通道的延时。XACT 软件的自动布局布线 (APR) 程序对 XC2000, XC3000, XC3100 的功能块进行布局。
- 布线——在 LCA 器件内, XACT 自动布线软件用布线资源以全程最短的路径试着完成所有的逻辑连接。用户可以用约束条件来操纵自动布线软件。一般采用的迷宫布线算法试图找出在两点间仅沿水平和垂直布线的最短通道, 布线器还有拆开再试 (rip-up and retry) 的能力, 可移去已建立的布线, 并为另一信号留出空间而重新布线。APR

可以对 XC2000, XC3000, XC3100 系列的设计布线。

图8.4示出了设计实现的过程。

最新的设计实现工具是时序驱动的,换言之,在器件的布局和布线期间,对整个信号通道执行时序分析,因此,用户可以对设计规定性能的要求,例如,对一个状态机的时钟速度提出要求,这些性能要求变成布局和布线算法中利用的代价函数。设计实现工具的另一关键特性是支持递增设计,因为设计周期实际上是反复的,可能在设计接近末尾时需要作稍许更改,有递增设计的支持,设计的先前的实现可用作控制新的稍许更改反复的映射、布局和布线的指导,两个设计匹配的地方,新的反复模仿已有的旧的结果,保存它的物理实现,因而也保存它的时序特性,所以,设计中已证明不变的部分不需要重新仿真或重新测试,没有这个特性,设计的小改动可能导致物理实现的极大差别,从而否定先前的测试和仿真结果。

设计实现的结果是产生一个完整的已布局布线的 LCA 设计,及一个配置位流文件。一旦对设计完成了全部布局布线,就需要进行测试,这就引出了 LCA 开发过程的下一个阶段——设计验证。

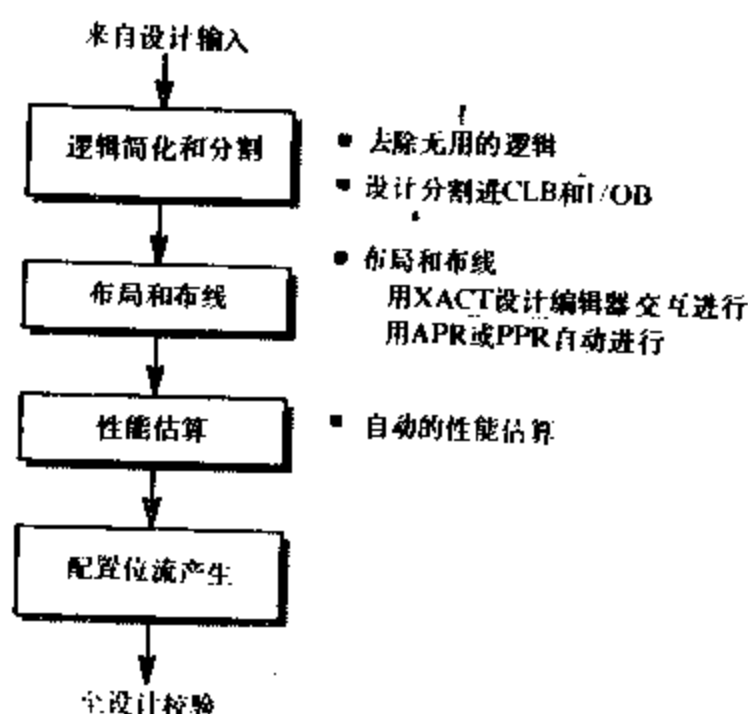


图8.4 设计实现流程图

三、设计验证

设计验证主要作用在于保证最坏情形下,设计在电路上能可靠地工作。

Xilinx 支持三种基本的设计验证方法,交互式验证,电路上验证及功能时序仿真。这三种验证方法对保证设计功能正确非常重要。

- 交互式验证——XDelay 可以交互式地进行路径时序分析。
- 电路上验证——可以在实际电路板上用 Xilinx 加载电缆加载并测试 LCA 设计。
- 仿真——完成设计实现后,应该用设计仿真软件包对设计做完整的测试。仿真能在最坏的工作情况下检查设计的逻辑功能,也能在设计实现之前检查设计的逻辑功能。

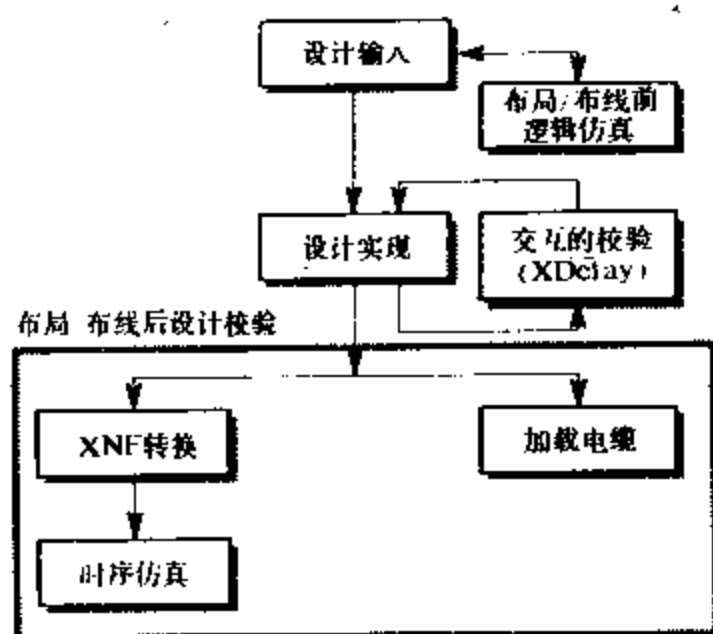


图8.5 设计校验流程图

§ 8.2 设计输入

设计输入是从概念到网表的方式来完成设计的。可以用原理图编辑器,文本工具,或两者结合来输入设计。这些设计方法需要支持 Xilinx 的第三方工具,这些工具以它们自己的网表格式产生设计文件。

这部分对用原理图或文本输入工具进行设计输入作些介绍。特别介绍层次设计,讨论在设计输入阶段对设计实现进行控制的方法。

§ 8.2.1 原理图输入

原理图输入工具提供设计输入的图形界面。可以用这些工具在设计中把代表逻辑元件的符号连接起来。可以用单个门电路建立设计,或用门电路来建立功能块。这部分集中介绍用库单元建立功能块的方法。

1. 库元件

a. 基本单元及宏单元

Xilinx FPGA 库除提供基本单元外还提供公用的高级宏单元功能块。基本单元是一些有唯一的库名、符号及描述的基本电路元素,例如与门(AND)及或门(OR),宏单元包含多个库单元,它可以包括基本单元及其它的宏单元。图8.6列出了几个库单元。

有两种宏单元可在 Xilinx FPGA 中使用:软宏单元和相关布局宏单元(RPM)。对 Xilinx 所有 FPGA 器件有效的软宏单元是预先定义功能的,但可以灵活地进行映射、布局、布线的宏单元。只对 XC4000/A/H 等器件有效的相关布局宏单元有固定的映射及预先定义的相对布局。

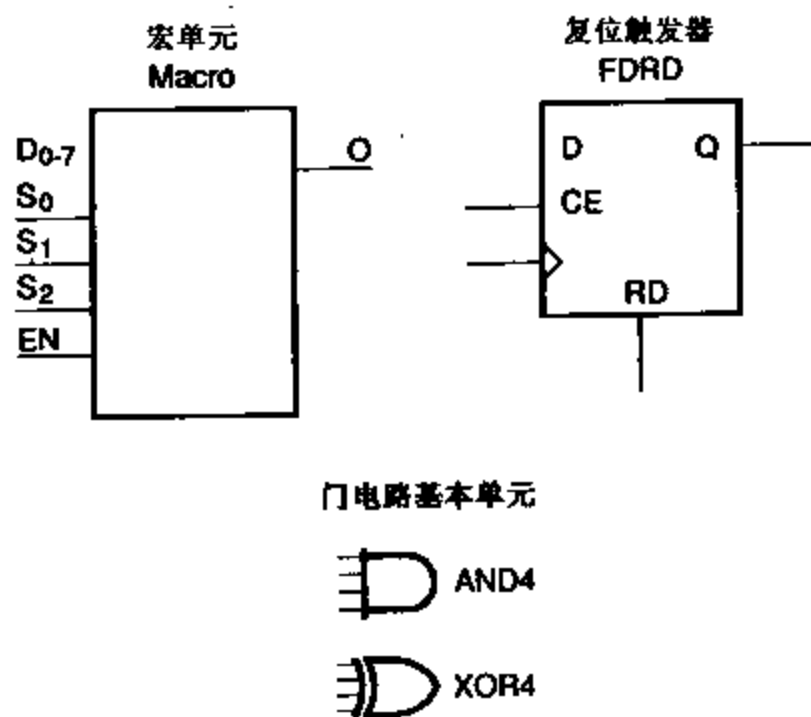


图8.6 库元件

b. 结构资源

对特定功能选择原理图符号是关键的。例如,可以用一个 CLB 的功能产生器来实现标准的 AND4 符号(示于图8.6)。若需要一个带异步复位的 D 触发器,则必须在原理图中指定 FDRD 基本单元。

原理图符号的特定选择也决定怎样实现寄存器,寄存器可用在 CLB 和 IOB 中,若要保证用 IOB 来实现寄存器,则必须使用相应的 I/O 基本库单元。

2. XC4000 的 MenGen

Xilinx 的存储器产生器 MenGen,对 XC4000/A/H 的 FPGA 产生 RAM 和 ROM 是方便的工具, MenGen 产生一个 XNF 文件,一个逻辑文件和原理图符号,它可以用作设计的部件。

MenGen 可以产生 32 位宽, 256 字大小的存储器,两个 16×1 的存储器或单个 32×1 存储器可以放进一个 XC4000 的 CLB。

3. X-BLOX

X-BLOX 工具提供可变尺寸的中小规模设计的积木块库,如加法器,计数器,译码器和移位寄存器。

这个库补充 XC3000A/L, XC3100A 和 XC4000A/H 的宏单元库,它包含较简单,固定尺寸的逻辑和门函数。X-BLOX 工具也包括 X-BLOX 软件,它把 X-BLOX 的库元件集成进设计中。

§ 8.2.2 文本输入

文本输入很适合包括状态机及译码器在内的很多设计。有许多硬件描述语言可用于设计 FPGA,这其中包括 ABEL—HDL, VHSIC HDL (VHDL), Verilog HDL, PALASM, MINC 及 CUPL,这部分着重点在 Xilinx ABEL 及 VHDL, Verilog HDL。

§ 8.2.3 层次设计

由于原理图经常包含有层次,如图8.7所示,有以下一系列理由使分层设计变得是很重要的设计方法,如

- 有助于帮助构思。
- 在设计中增加结构性。
- 易于设计调试
- 易于对设计的不同部分用不同的设计输入方法(原理图和文本)
- 易于递增式设计。递增式设计由对单个子块的设计、实现及验证组成。子块用于分阶段建造起一个设计。

有助于并行式设计,并行式设计是在许多开发人员中分配一个设计,以便并行地开发设计中的不同部分。

使用分层设计时要考虑的问题之一是层次名的问题。

一个规定的层次名表示每个库单元、唯一的功能块及产生的实例。例如,层次名
/Acc/alu-1/mute-4/8count-3/4bit-0/mux-1/or2

最后三项指明了一个4位计数器中一个多路选择器的第一个元件中的2输入或门。

注意:在设计中对器件及网线命名是非常重要的。在原理图编辑器里,设计验证工具例如XDelay 保存和使用器件名及网线名,这些名字也用于返回注释,并且出现在测试及分析工具中。若不对器件及网线命名,原理图编辑器则使用自动产生的名称,例如,上面的例子或许会命名成:

/ \$ 1a123/ \$ 1b942/ \$ 1c23/ \$ 1d235/ \$ 1e121/ \$ 1g123/ \$ 1h57

用自动产生的名称对电路进行分析是很困难的。

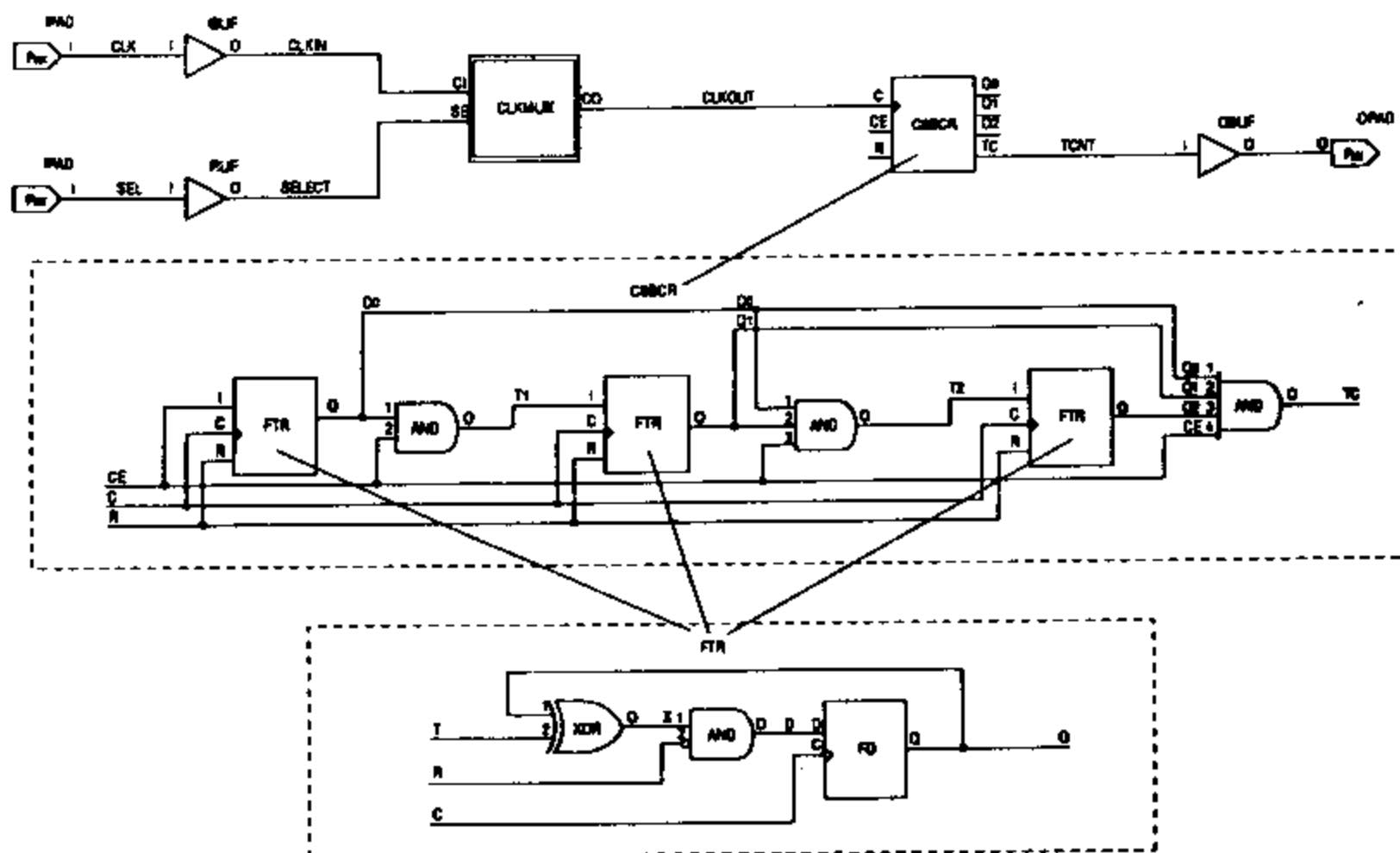


图8.7 原理图层次设计

§ 8.2.4 控制实现

若实际应用要求对设计做出约束,则要在设计输入阶段指定映射及块布局。

1. 映射

可以规定如何把一个特定的逻辑块映射到 CLB 中去,对 XC2000, XC3000/A/L 和 XC3100/A 系列 FPGA 利用 CLBMAP, 对 XC4000/A/H 系列的 FPGA 利用 FMAP 或 HMAP。这些映射符号可以在原理图中使用,但过多使用这些规定,会使设计的布线更困难。

2. 块布局

块布局可以约束到器件中的一个特定的位置,一个区域或沿着一条长线。所有这三者都可以在约束(CST)文件中指明。位置和区域可以在逻辑图中直接指定。不恰当的布局会对布局及布线有一定影响。块布局通常定义 IOB 的布局。

注意:设计实现软件可以给连线标上“关键”标志或给予很大的优先权。然而,标有“关键”标志及优先权大的连线太多将大大降低设计布局布线的布通率,导致差的性能。由于这个原因,Xilinx 特别奉劝用户只有在特别需要的地方才使用关键标志及优先权。

3. 时序规定

对于以下的器件:XC3000A/L, XC3100A 和 XC4000/A/H,可以在原理图中直接对设计中的通道规定时序要求,XACT 软件的 PPR 在对设计进行布局和布线时,利用这些时序规定来达到最佳的性能。

在已经输入设计之后,就要仿真或实现设计。功能仿真要测试设计中的逻辑以断定设计是否恰当地工作。若在设计流程的早期进行了功能仿真,则可以在以后的设计步骤中节省很多时间。

设计实现过程是把设计在所规定使用的 Xilinx FPGA 中进行实现,在下节作详细讨论。

§ 8.3 设计实现

设计实现接受设计网表并且产生适合于 Xilinx FPGA 的位流文件。这个过程可以用 XMake 程序来自动进行。XMake 程序是 XACT 基本开发系统的一部分。设计实现包括以下步骤,从设计网表到 Xilinx 网表格式(XNF)的转换,设计优化、合并、映射、布局、布线及位流文件产生。所以设计实现就是让设计做好装载到 Xilinx FPGA 中的准备的过程。这部分包括了在设计自动实现过程中 XMake 程序所包含的每一步骤。也着重介绍了成功地设计 Xilinx FPGA 的秘诀。

为了便于参考,设计实现的程序在表8.2中作了概括。尽管大多数程序自动执行,但了解每个程序在设计实现过程中所起的作用还是有益的。

§ 8.3.1 XNF 转换(XNF Translation)

XNF 文件在设计输入工具与 Xilinx 开发软件间提供一个公共的接口。是一个含有设计的逻辑及连接信息的 ASCII 文本网表格式,也支持设计的时序信息。

每个输入工具都有特定的输出类型。在把输入工具的输出转换到 XNF 文件时,必须使用转换器。也可以把含有时序信息的 XNF 文件转换成仿真器使用的格式。Xilinx 支持的每

个输入工具的软件文档详细说明了怎样进行 XNF 与别的格式的转换。

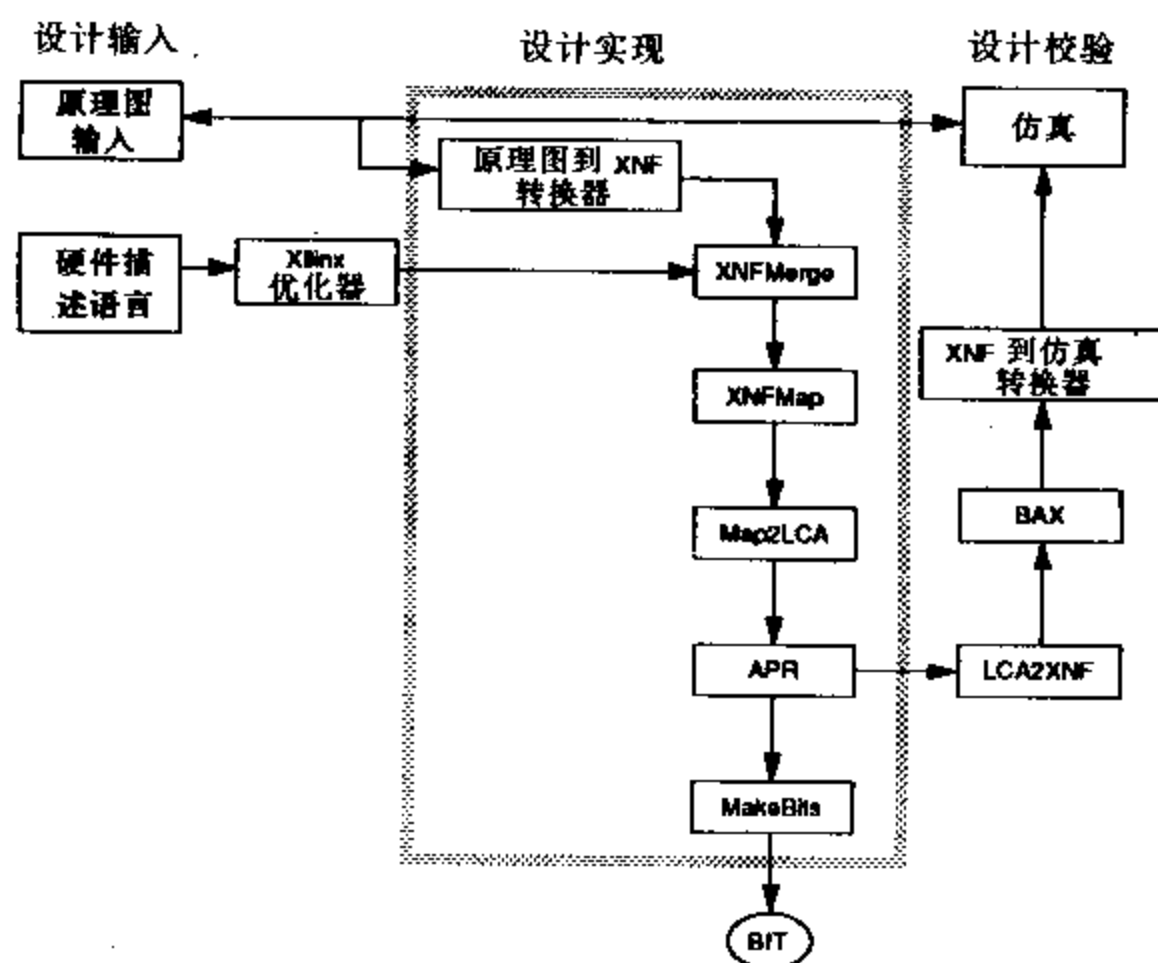


图8.8 各系列设计实现流程图

§ 8.3.2 优化(Optimization)

优化把与器件无关的或行为的逻辑变换成可以在 Xilinx FPGA 中最好地被实现的形式。这个过程由如 Xilinx ABEL 和 X-BLOX 的程序执行,设计优化的一个例子是利用 Xilinx ABEL 的符号状态机的编码。

在第七章的 § 7.7对 EPLD 和 FPGA 的逻辑优化作了简要的介绍和比较。

§ 8.3.3 合并(Merging)

分层设计及多设计输入源会产生多个文件,XMake 可以在进行后续的实现步骤前自动把它们合并成一个 XNF 文件.XMake 用 XNFMerge 程序来完成这项工作。

§ 8.3.4 映射(Mapping)

映射是把一个设计分成许多适合于 CLB,IOB 或其它 Xilinx FPGA 资源的小块的过程。这个过程已在图8.3中说明。这个过程中使用较多的另一术语是分割(Partitioning),映射和分割是可交换的术语,即映射也称为分割。由于不同 Xilinx FPGA 系列可用的资源不同,映射程序将按所使用的器件作不同抉择。

§ 8.3.5 布局(Placement)

XMake 使用自动布局软件 APR 或 PPR 把已映射的 CLB 及 IOB 放到 LCA 器件中最佳位置,以便使连接它们所需的布线数量最小。XMake 对 XC2000, XC3000 及 XC3100 器件执行自动布局(APR)程序。

XACT 开发系统为达到最优布局,不要求约束条件。实际上,诸如人工指定出腿的约束条件常极大地限制了有效的逻辑布局,约束条件会对布局算法产生大的限制,应尽量避免。

表8.2 在设计实现中使用的程序

程序	功能	输入	输出	适用系列
XMake	自动实现设计	原理图输入文件 或 XNF 文件	BIT 文件 LCA 文件	所有
XNFMerge	合并多个 XNF 文件	XNF 文件	单个平整的 XFF 文件	所有
XNFPrep	执行设计规则检验 移去无用多余逻辑	XNF 文件	XTF 文件或 XTG 文件 (X-BLOX)	XC2000/L XC3000/A/L XC3100/L XC4000
XNFMap	在器件的资源中 分配设计	XTF 文件	Map 文件及 CRF 文件	XC2000/L XC3000/A/L XC3100/A
MAP2LCA	把 MAP 文件 转换为 LCA 文件	MAP 文件	LCA 文件 SCP 文件	XC2000/L XC3000 XC3100
APR	自动布局和布线	LCA、SCP CST 文件	布线后 的 LCA 文件	XC2000/L XC3000 XC3100
PPR	映射,布局,布线	合并的 XNF 文件 和约束文件	布局布线后 的 LCA 文件	XC3000A/L XC3100A XC4000/A/H
MakeBits	产生 FPGA 位流	布线后 LCA 文件	BIT 文件	所有
XDE	人工设计编辑	LCA 文件	LCA 文件	所有

§ 8.3.6 布线(Routing)

XMake 使用 APR 或 PPR 对设计自动布线。布线算法设计成确定能达到最短延时的互连通道。

§ 8.3.7 产生位流文件(Bitstream Generation)

作为最后一个步骤,XMake 用 MakeBits 程序对 Xilinx FPGA 自动产生一个位流文件。

MakeBits 以 BIT 文件的方式产生特定的 FPGA 配置数据。

注意: MakeBits 对设计产生最后的位流文件之前,应运行设计规则检查(DRC)。然后再运行 MakeBits,必须用 Tie 选项以便把设计未使用的 Xilinx FPGA 资源定义成有效逻辑电平。关于 MakeBits 更多的说明,它的选项及配置模式在第九章中 § 9.1.1 有关 MakeBits 程序部分作进一步介绍。

§ 8.3.8 XACT 设计编辑器(XDE)

因为 XMake 可以执行自动设计实现的所有步骤,利用 XDE 的人工实现通常是不必要的。但是,可以利用 XDE 得到 XMake 如何实现设计的剖视。

关于 XACT 设计编辑器(XDE)在第十章再作详细讲述。

§ 8.3.9 设计的大小及性能

关于设计大小和性能的信息可以帮助优化设计。当布局 and 布线整个设计时,最终的报告文件列出 CLB, IOB 和其它所利用的器件资源。

1. 块延时

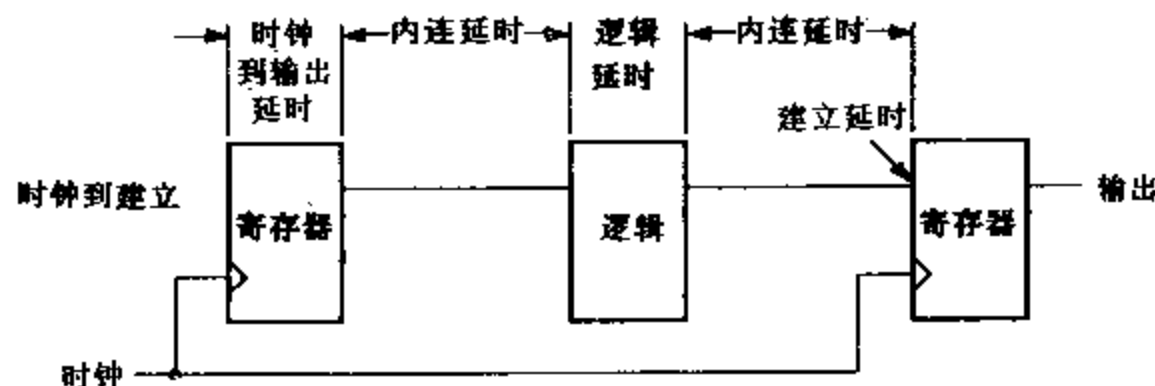


图8.9 块延时

三种类型的块延时示于图8.9。描述如下:

- clock-to-output 延时是时钟沿之后信号出现在触发器输出端所用的时间。
- logic 延时主要是由 CLB 中的配置逻辑决定的。一个给定的 CLB 中的组合延时是固定,只与查找表的速度有关,与逻辑函数无关。在使用了 FPGA 中的其它逻辑如 TBUF 时,延时也会增加。
- Set-up 延时出现在逻辑驱动寄存器的输入时候。建立时间是必需的,以保证触发时寄存器输入端的数据稳定。

这些块延时可以用《可编程逻辑数据手册》中的最坏情况值来估计。

2. 网线延时

网线延时由在对设计布线时使用的实际连接决定的。并且在进行布局布线之前是很难估计的。

§ 8.4 设计验证

设计验证包括仿真,静态时序分析及电路上验证。每一种设计验证方法所用的工具都集成到 XACT 基本开发系统中,并且仿真工具是作为选项提供的,在这节中,对“设计验证”部分的静态时序分析及电路上验证工具作较详细说明。

设计验证是测试设计的功能及性能的过程。XACT 基本开发系统提供三种互相补充的工具来进行设计验证:仿真,静态时序分析及电路上验证。这节讨论以下内容:

- 设计验证及设计流程——这部分说明用仿真,静态时序分析及电路上验证进行验证的设计流程。
- 仿真——这部分说明了仿真是如何对设计的功能及最坏情形时序进行验证的。这部分也讨论了设计仿真的一些重要特点,及在 Xilinx FPGA 设计流程中出现的位置。
- 静态时序分析——这部分描述静态时序分析是如何快速判定重要信号的传输延时的,在 XACT 基本开发系统也用了静态时序分析。

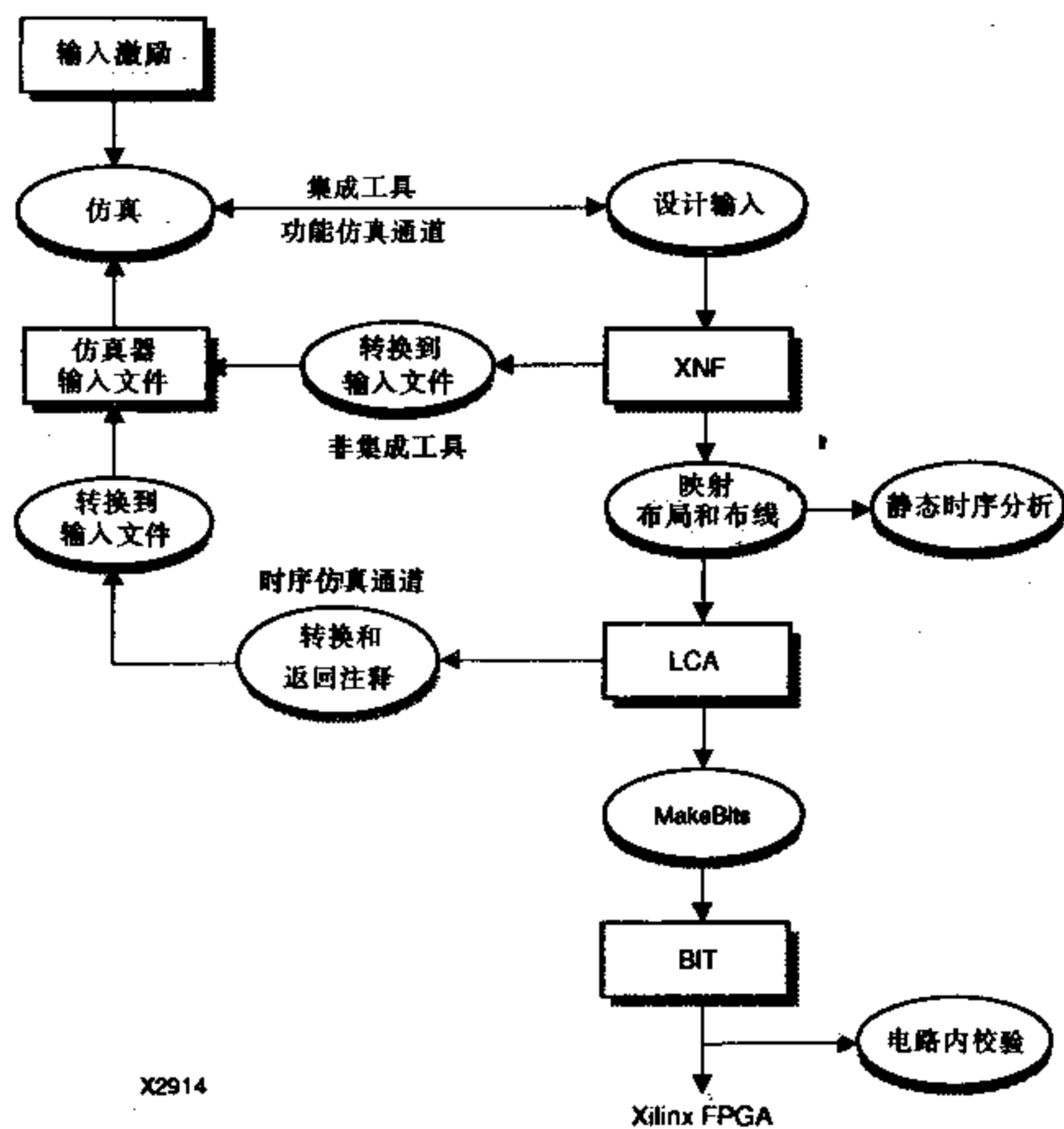


图8.10 校验如何安排在设计流程中

• 电路上验证——这部分说明对最终应用中的设计怎样进行电路上验证是最有价值的。

同时也介绍了 XACT 基本开发系统的加载、读回及探测功能。图8.10较详细地描述了设计流程,从图中可以看到有三种不同的验证 Xilinx FPGA 设计的方法、仿真静态时序分析及电路上验证。这节集中说明这三种验证方法。

§ 8.4.1 验证工具概述

图8.11概括了用于仿真,静态时序分析及电路上测试的工具。

1. 仿真工具

仿真工具提供的软件模块编制成执行设计的详尽特性。有两种仿真方式:功能仿真在用器件实现设计之前判断逻辑是否正确;时序仿真检验装载在指定器件的设计在最坏条件下以所需速度工作的情况,这是在对设计进行映射、布局、布线完成之后进行的,这时所有设计中的延时都是已知的。

2. 静态时序分析

使用 XACT 基本开发系统中的静态时序分析工具——XDelay,可以快速检查设计中的时序错误。XDelay 可以获取设计中所有路径的每一路径延时信息。

3. 电路上验证工具

作为最后的测试,可以在最终应用中对设计进行验证。因为可以多次地对 Xilinx FPGA 编程,所以能很容易地把设计装入器件并且对它在电路中进行测试。可以用 Xilinx 加载电缆把设计的位流文件装入器件以便对设计在电路上进行测试。

校验类型	工具
仿真 	第三方仿真器 (集成和非集成)
静态时序分析 MHz	- QueryNet - XDelay
电路内校验 	- 设计规则校验器 - XChecker or Download 电缆 - 探测器

图8.11 校验工具

§ 8.4.2 仿真

设计仿真包括使用软件模块对设计进行测试。在测试设计的功能和最坏条件下的性能时,这种方法是很有有效的。可以很容易用测试矢量或其它输入激励探测内部节点,并且可以

用这些结果在原理图中来修改设计。

仿真是用与 XACT 基本开发系统相连的第三方工具来执行的,每个仿真器都有许多相应的命令及过程,因而本节不详细讨论特定的仿真器。针对不同厂家的 CAE 工具的《接口用户手册》包括了 Xilinx 支持的仿真器的特定命令及特点,这些可以用作基本的参考。

1. 功能仿真

如图8.10,功能仿真可以在设计流程的最早阶段进行。由于没用到已实现的设计的时序信息,所以仿真器以单位延时来测试设计中的逻辑。如果在设计流程的早期进行功能仿真,则可以较快较早地纠正设计中的错误。

图8.10表示出了含有集成或非集成仿真工具在内的设计流程。集成工具把仿真器与原理图编辑器结合在一起,以便在它们之间无需进行转换。使用集成工具可以从输入软件直接进入仿真软件。非集成工具则没把仿真器与原理图编辑器结合在一起,转换程序把 XNF 文件转换成仿真输入文件,以使用非集成工具进行仿真。

2. 时序仿真

图8.10概括说明了在布局布线之后,一旦获取延时信息后如何进行时序仿真。时序仿真用于验证时序关系,及对关键路径在最坏情形下的测定。因此时序仿真是很有用的。也可用来判断设置或保持的设计内容是否有冲突。

为了对仿真器输入时序信息,必须利用 LCA2XNF 把已布线的 LCA 文件转换为 XNF 文件。对于 XC4000 的设计,在运行 LCA2XNF 之前必须以 -w 选项对 LCA 文件运行 MakeBits。返回注释 XNF(BAX)程序把 XNF 文件中的信息与包含在原始设计 XNF 文件中的网线名称结合起来。最终的 XNF 文件就可以转换成适合于仿真器的输入格式。

在原理图输入部份已指出,设计中给网线命名对功能和时序仿真来说都是很重要的。

3. 仿真特点

1) 数据表示

许多仿真器提供一个图形或表格化的界面用于显示分散的信号或总线信号的波形及进行它们的时序分析。图形仿真器能显示相对的及绝对的时序,用光标在信号边沿进行测量,表示于图8.12。

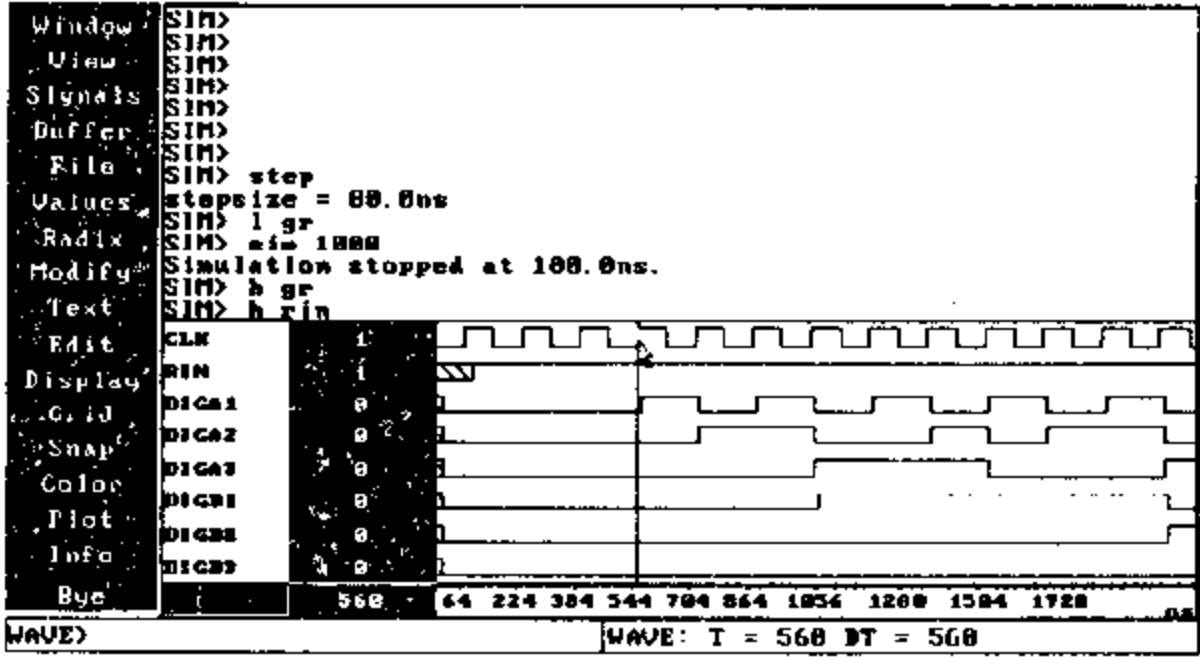


图8.12 仿真输出例子

除波形输出外,还可以用表格形式察看仿真数据。有两种常用方法可以跟踪所要测试的网线的输出与时间的关系。一种方法是每几个时间单位记录一次,但这种以时间为步长的输出有不利之处,常常在无输入变化又无输出信号时也按时间步长显示,另一种方法是每次输出电平有变化时记录。这种方法对以事件为工作对象的仿真是很合适的。

2) 返回注释

使用 BAX 的返回注释在时序仿真中是一个关键步骤。BAX 比较设计实现前后的 XNF 文件。这些比较是重要的,因为在实现过程中有一些网线和逻辑被删掉。BAX 修改将转换成仿真器格式的 XNF 文件,以便把原始网线名称与已实现设计的相应时序信息结合起来。

返回注释提供在功能和时序仿真中使用相同输入激励的方法。集成的仿真器及原理图输入软件包用返回注释提供了在设计图纸上直接查看仿真结果的方法。

图8.13表示出了一个对原理图编辑器返回注释的例子。

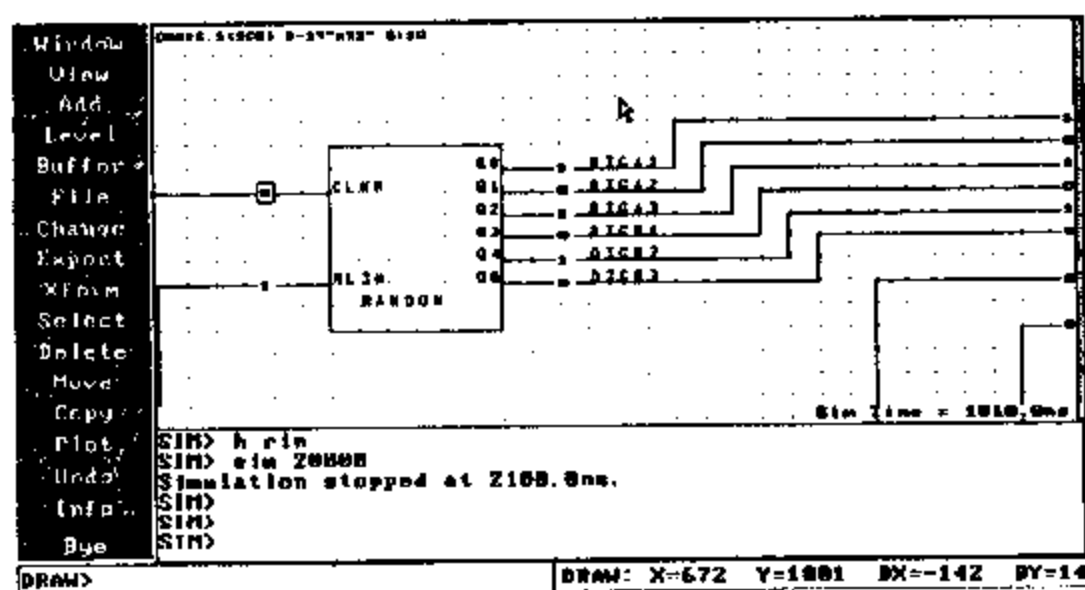


图8.13 返回注释到原理图

3) 输入激励

可以用激励文件或一系列测试矢量来仿真输入量。激励文件指定节点输入值。这些节点是希望在仿真期间控制的。可以指定希望改变的单个状态,也可以建立测试矢量。测试矢量指定选取的输入在特定时间的输入值。测试模式是应用于一系列连续时钟边缘的一组测试矢量。

§ 8.4.3 静态时序分析

静态时序分析最适合在布局布线完成后的快速时序检验。用 XDelay 可以确定设计中的时序问题,Xdelay 提供对设计中所有通道读取信号路径的延时信息,用它可确定设计中的路径延时。如图8.14所示,在 XDE 中可利用 XDelay 和 QueryNet 程序执行静态时序分析。

1. XDelay

XDelay 是一个单独的静态时序分析工具。XDelay 可以对设计性能做一个全面的分析。例如,XDelay 可以计算设计中每个时钟的最大时钟频率。图8.15中列出了一个 XDelay 输出文件的一部分。所以 Xdelay 命令向用户提供详细的延时信息,并可对芯片性能做整体上的分析,给出时钟的最大工作频率,用户可以通过选择项得到某一条或某一类路径的延时信

息,如果都缺省,则显示所有路径的延时信息。

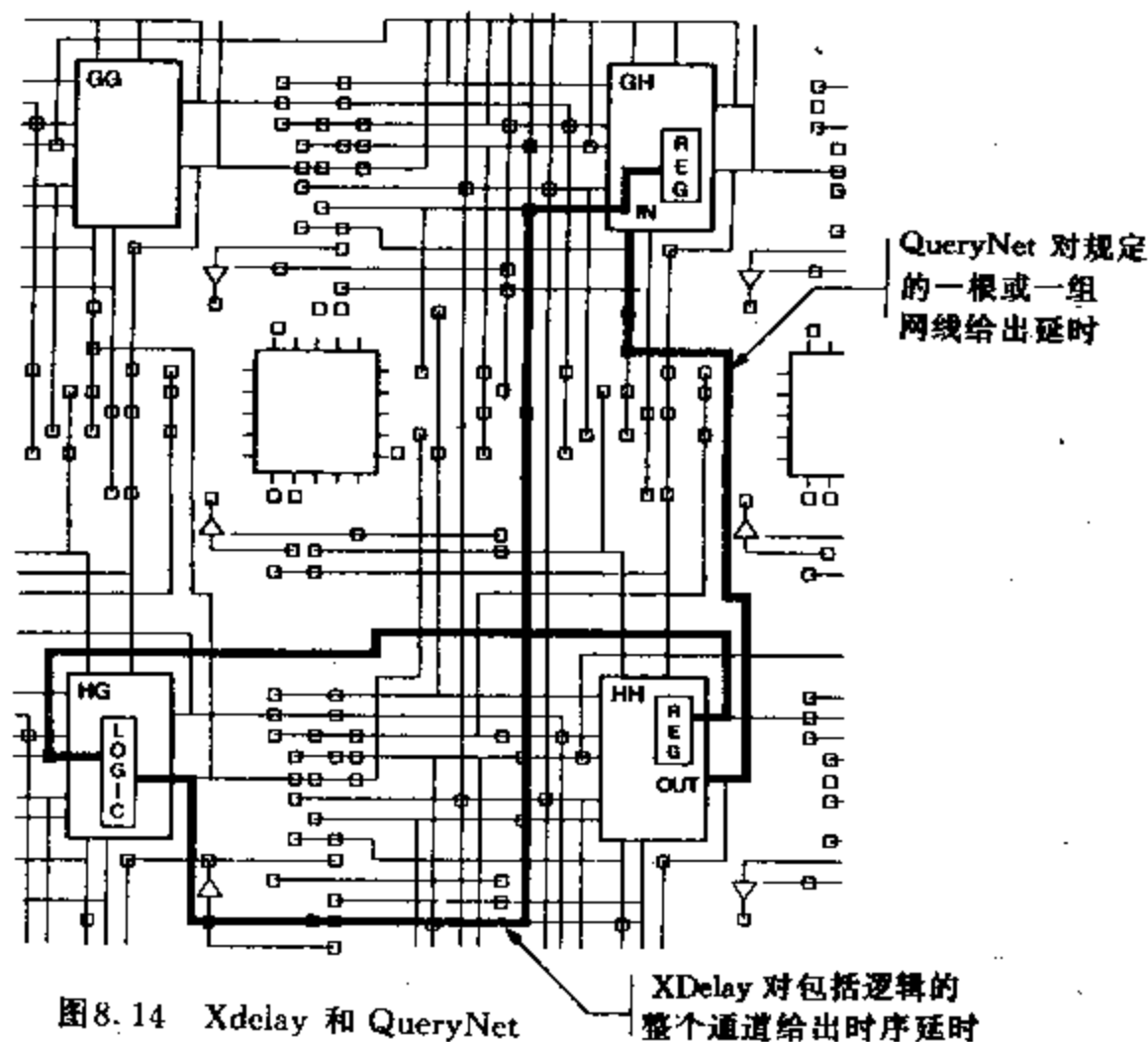


图8.14 Xdelay 和 QueryNet

下面列举几个可选择项:

- From 规定计算路径延时起始点,可以是 BLOCK,NET,PIN
- To 规定路径终点,可以是 BLOCK,NET,PIN
- PadtoPad 报告从输入焊盘开始到输出焊盘终止的组合逻辑路径延时
- PadtoSetup 报告从输入焊盘开始到 D 触发器数据端终止的路径延时,包括数据建立时间
- ClocktoPad 报告时钟驱动的触发器输出端到输出焊盘之间的路径延时,包括翻转时间
- ClocktoSetup 报告从触发器输出开始到触发器 D 端终止的路径延时,包括翻转和建立时间

从上面所列选项可看出,XDelay 是最适合于检查延时路径,有四种信号路径,焊盘到焊盘、焊盘到建立、时钟到建立(寄存器转移)及时钟到焊盘,可以进一步用 from 及 to 等选项指定要验证的路径。

2. QueryNet

图8.16示出了用 QueryNet 对一根网线检查后的结果。可以用 QueryNet 得到单根网线的延时,或者可以指定多根想要探测的网线。QueryNet 特别适用于快速判断网线是否有大的延时或偏移。图8.18是一个 QueryNet 输出文件的一部分。

§ 8.4.4 电路上验证

这节中讨论的验证方法可以在把设计装入到实际 Xilinx FPGA 之前进行。但在电路上验证也是可能的,以便可以在最终应用中测试实现的设计。电路上验证是在“典型”工作条件下测试电路。

Download 电缆及 XChecker 软件可用于把设计装载到应用中的 LCA 器件中。

也可用 XChecker 电缆探测已装载的设计的内部节点。探测可以很容易地指出任何设计问题的位置,因而是重要的。

XDelay: sample.lca (3020PC68-70), xact 4.29b, Mon Aug 31 17:51:04 1992

Xdelay path report options:

From all.

To all.

Output will be sorted by decreasing path delays.

Logical Path	Delay Cumulative
From: Blk CQL PAD to P14.I	: 6.0ns (6.0ns)
Thru: Net GOSCSET to BA.C	: 2.6ns (8.6ns)
Thru: Blk GOSC to BA.X	: 8.0ns (16.6ns)
Thru: Net CLKIN to BB.K	: 11.6ns (28.2ns)
To: Clock Input, Blk J01	: 0.0ns (28.2ns)
Target FFY drives output net "J1"	
From: Blk CQ PAD to P12.I	: 6.0ns (6.0ns)
Thru: Net GOSCRESET to BA.B	: 2.5ns (8.5ns)
Thru: Blk GOSC to BA.X	: 8.0ns (16.5ns)
Thru: Net CLKIN to CB.K	: 11.6ns (28.1ns)
To: Clock Input, Blk J23	: 0.0ns (28.1ns)
Target FFY drives output net "J3"	
From: Blk CQ PAD to P12.I	: 6.0ns (6.0ns)
Thru: Net GOSCRESET to BA.B	: 2.5ns (8.5ns)
Thru: Blk GOSC to BA.X	: 8.0ns (16.5ns)
Thru: Net CLKIN to BB.K	: 11.6ns (28.1ns)
To: Clock Input, Blk J01	: 0.0ns (28.1ns)
Target FFY drives output net "J1"	

图8.15 从 XDelay 输出的例子

Querynet: demo3020.lca (3020PC68-70), xact 4.29b, Mon Aug 31 17:50:10 1992

---- CLKIN	BA.X (GOSC) . . .	6.3 BA.A (GOSC)
		11.6 BB.K (J01)
		11.6 CB.K (J23)
		11.6 DB.K (J4)
		6.6 P12.O (CQ)
		7.6 P12.T (CQ)
---- GOSCQL	BA.Y (GOSC) . . .	4.6 P14.O (CQL)
		4.6 P14.T (CQL)
---- GOSCRESET . . .	P12.I (CQ) . . .	2.5 BA.B (GOSC)
---- GOSCSET	P14.I (CQL) . . .	2.6 BA.C (GOSC)
---- J1	BB.Y (J01) . . .	1.0 CB.A (J23)
---- J3	CB.Y (J23) . . .	1.0 DB.A (J4)
---- J4	DB.X (J4) . . .	9.7 BB.A (J01)
		9.6 HB.A (FLASH)
---- OUT7	HB.X (FLASH) . .	7.0 P28.O (O7A)
		6.3 P29.O (O7B)

图8.16 从 QueryNet 输出的例子

1. 设计规则检查器

在产生最终位流文件之前,用设计规则检查器(DRC)检查 LCA 文件中妨碍设计适当工作的问题是必要的。另外,在 XACT 设计管理器 (XDM)的 MakeBits 环境下执行 DRC。

2. Xilinx 加载电缆及 Xchecker 软件

为了装载设计,需要用 MakeBits 产生一个配置位流文件。Xchecker 软件支持并行加载电缆的操作。

加载电缆有两种:接在 PC 机并口的 DownLoad 电缆和接在 PC 机串口的 Xchecker 电

缆。Download 电缆用 V_{cc} , GND, CCLK, D/P 和 DIN 五根接线, 只能实现将位流文件加载到相应的器件, Xchecker 电缆除能将位流文件加载到相应的器件外, 还能在配置 FPGA 后, 与原设计比较, 校验它的配置, 以及探测 LCA 的内部逻辑以诊断设计。

用 Xchecker 软件执行以下步骤来完成对器件的加载:

1. 器件所在系统加电;
2. 用 PORT 指令规定加载电缆的连接口;
3. 用 LOAD 指令将 BIT 文件配置到芯片中。

Xchecker 软件也可以利用读回功能进行配置的校验和诊断, 在第十二章中的实验十四说明了如何使用读回功能。两种电缆的连接要求和有关说明见表 8.3。

表 8.3 加载电缆的说明

接线名称	功能	连接形式		
		XC2000	XC30/3100	XC4000
V_{cc}	供电线的电源, 一般为 +5V, 100mA	接目标系统 V_{cc}		
GND	供电线的电源地	接目标系统接地端		
CCLK	配置时钟—为目标系统提供配置和读回时钟	接目标系统配置时钟		
D/P	结束配置信号, 为 XC20/3000 提供重新配置脉冲	用 10—50K 上拉电阻 接 D/P 脚		用 10—50K 上拉电阻 接 DONE 脚
DIN	配置时提供配置数据到目标系统, 其它时间为三态	接目标系统 DIN 端		
PROG	对 XC4000 器件提供重新编程脉冲	不连接		用 10—50K 上拉电阻 接目标系统 PROG 脚
INIT	为 XC30/4000 指示配置开始的状态脚	不连接	用 10—50K 上拉电阻 接 INIT 脚	
RST	配置后此脚可为低复位目标 LCA 内部锁存和触发器	用 10—50K 上拉电阻 接目标 LCA RESET 脚		用户可编程连接 要求 10—50K 上拉电阻
RT	在目标 LCA 的 RTRIG 脚由低到高的瞬变起始读回	用 10—50K 上拉电阻 接目标 M0/RTRIG 脚		用户可编程连接 要求 10—50K 上拉电阻
RD	在此脚读目标 LCA 的读回数据	用 10—50K 上拉电阻接 M1/RDATA(串行从模式)		用户可编程连接 用 10—50K 上拉电阻
TRIG	此脚上由低到高的瞬变触发 Xchecker 起始读回	接目标系统读回触发器		
TDI TCK TMS	备用	不连接, 备用		
CLKI	传送 120KHz—10MHz 系统时钟到 Xchecker, 此脚接目标系统时钟, 可同步目标系统时钟读回触发	接目标系统时钟源, 同步读回和诊断		
CLKO	此脚是目标系统时钟, 此时钟可来自 CLKI 或由 Xchecker 电缆内部产生	接目标系统时钟源, 同步读回和诊断		

§ 8.4.5 读回功能

Xilinx 的器件是100%可测的,可以利用循环冗余校验(CRC)对配置位流的 XC4000系列器件校验加载进配置存储器的位流的完整性。配置位流包括一个对每个传输进 FPGA 器件的数据信息段的16位配置数据 CRC 的4位局部校验,利用这个技术,器件检测无效的数据位并中止配置过程,INIT 状态腿被拉低,标志配置存储器期间出现错误。因而,读回功能仅在以下少数情况起作用:

1. 在不稳定环境中校验配置;
2. 在 FPPA 开发状态期间,读回 RAM,CLB 和 IOB 的内部状态;
3. 测试要求系统内功能分析和校验的高可靠性应用。

读回运行并不与 FPGA 的运行冲突,在一个有效的读回请求后,内部节点的当前状态被拾取进专门的移位寄存器,利用用户定义的时钟信号,数据可以转移出器件,以下的内部配置数据和电路节点对读回是有效的:

1. 定义 CLB,IOB 和互连的逻辑配置的配置存储器位;
2. CLB 函数发生器的 X 和 Y 输出腿;
3. CLB 触发器的 XQ 和 YQ 输出腿
4. 在 IOB 输出通道中的触发器输出腿
5. IOB 的 I1 和 I2 输入腿

XC4000 的 CLB 包括 C1—C4 输入腿和 XQ 和 YQ 输出腿之间的专门连接,可以用来从 CLB 的一边到另一边的信号布线,虽然这些信号通过 XQ 和 YQ 腿离开 CLB,它们不拾取读回位流,仅拾取实际触发器的输出。

实验十四中计数器原理图 12.30 给出 READBACK 基本单元的用法;在 XDE 中,读回功能块是在器件的左下角和右下角,当连接 rdbk. TRIG 和 rdbk. DATA 信号时,激活此功能块,rclk. I 腿在没有接到别处时,接到 CCLK 腿。表 8.4 为读回基本单元的连接。

表 8.4 READBACK 基本单元的连接

读回腿	XDE 腿	描述
CLK	rclk. I	时钟输入可以接到任何器件输入腿,或任何 CLB 的输出,如它没有接到用户的网线,则接到器件 CCLK 输入腿,在 MakeBite 程序中选撤应的选项
TRIG	rdbk. TRIG	TRIG 输入低到高转换开始一个读回序列,要求最小脉宽是一个 rclk. I 周期,有效的触发引起确定节点的当前值被锁存到内部保持寄存器。 当 MakeBits 中 ReadAbort 选项被选,TRIG 输入的使读回流产高到低,在此情况必须提供附加时钟直到 rdbk. RIP 通知读回结束,在上述的读回正确终结之后,rdbk. TRIG 不可重确定直到最后三个时钟周期
RIP	rdbk. RIP (读回在进行)	此输出为高表示读回正在执行,在已出现有效的读回触发之后, RIP 使一个读回时钟周期有效,随最后数据位移出 FPGA 器件它变低,如果读回流产,RIP 保持有效直到读回序列正确有效
DATA	rdbk. DATA	在读回基本单元的 DATA 输出上读回数据是有效的, rclk. I 的每个上升沿从 LCA 内部保持寄存器移一个数据位到 DATA 输出 在读回位流中可以选:禁止用户数据

§ 8.5 先进的设计准则

Xilinx 的 FPGA 结构是十分规则,并类似于门阵列,但是它不是一个简单的门海,理解它提供的结构和资源可以使设计在速度和密度上变得更有效。本节着重于它的支持软件在改善设计有效性时的特性,描述从结构提取最大性能的先进设计技术。

§ 8.5.1 设计分割

设计实现过程的第一阶段是分割,LCA 器件仿效利用查找表的逻辑,必须把原理图分成能适合单个查找表的门的组合,无效的分割可能降低了设计的性能和造成利用比必须更多的 CLB 资源。无效地分割可以有两种原因。

(1)第一个原因:逻辑可能画成了普通分割边界不存在的方式,PPR 不把大的门分成可以集中到周围函数发生器未用部分的部件,替代之,它保护门的边界,并可能浪费函数发生器,图8.17表示相同的逻辑以两种方式画时,一种方式要求三个函数发生器级联,而另一种方式仅要求两个。

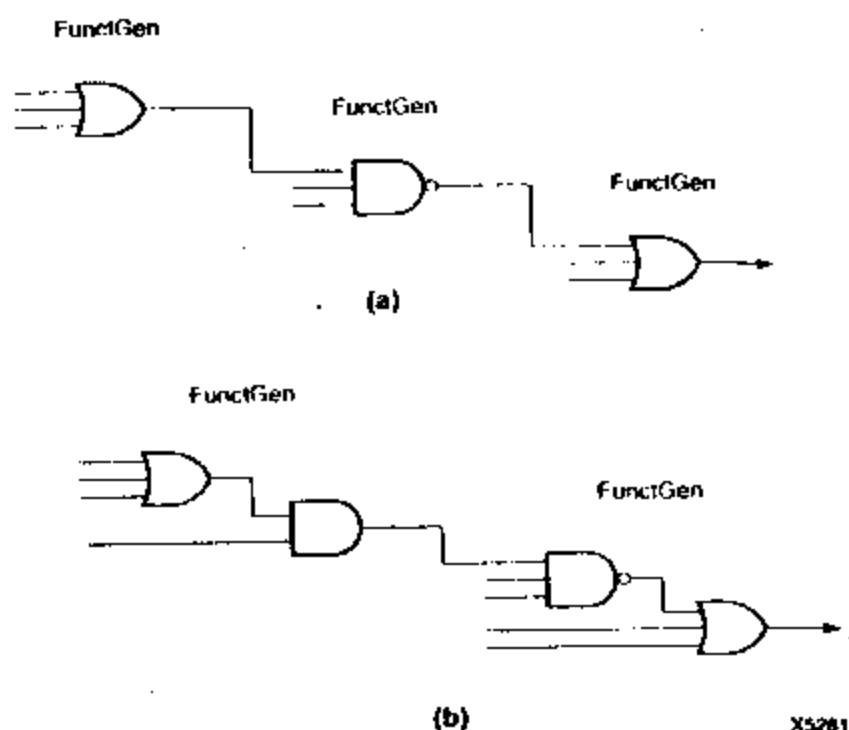


图8.17 用分解门电路改善分割

如果一个门有多于一个的扇出时,分割器是分配这个门的输出为一个函数发生器的输出,PPR 将不复制这个门,甚至这个拷贝可以集中到其它的函数发生器中,图8.18表示了第二个例子,不同地画出的逻辑可得到更少的 CLB 和 CLB 的更少的级。

重画一个已有设计的关键部分常常可以改善它的性能,但是最初画原理图时以保证普通的分割方式是更好的,这个似乎并不困难,也不要求门到门的分析或分割算法的知识。一般,设计过程从高层的方块图开始,象一个块分解成更小的块那样,通过一系列越来越详尽的方块图分层地进行,最终,最底层的块被转换成门以产生逻辑图,合适的分割之关键是以匹配 XC4000等 FPGA 结构的方式构造最底层的方块。

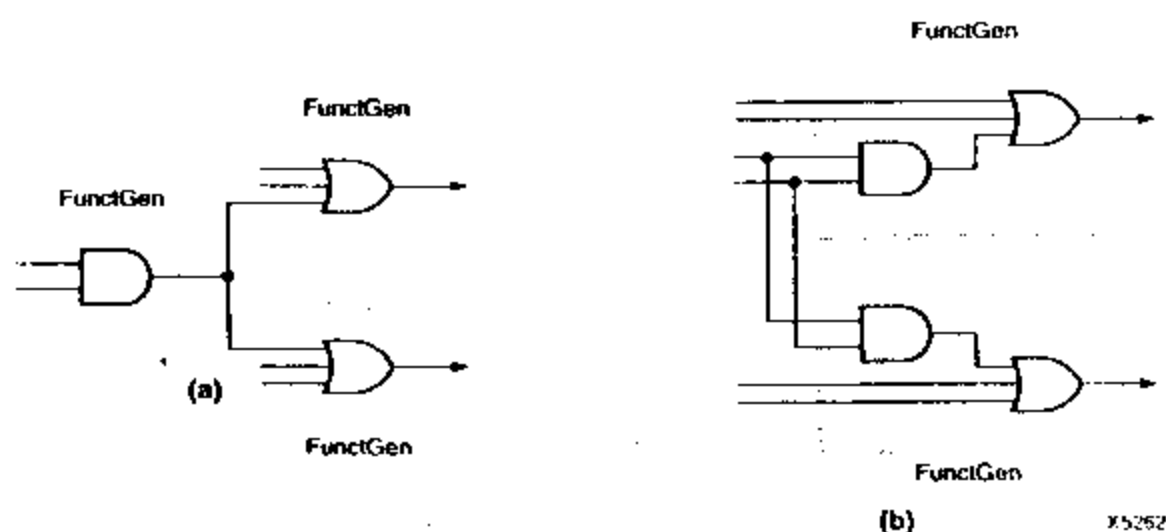


图8.18 用重复门电路改善逻辑分割

图8.19表示四个可以合适地在XC4000结构中实现的块结构,四输入块对应于F或G函数发生器,三输入块对应于H函数发生器,五输入块是F—G—H组合的特殊情况,这些块仅仅用它们的输入数目表征,因为一个查找表可以仿真它的输入的任意函数,它保证由每块要求的逻辑将安排进一个函数发生器,因此保证一个合适的分割存在。

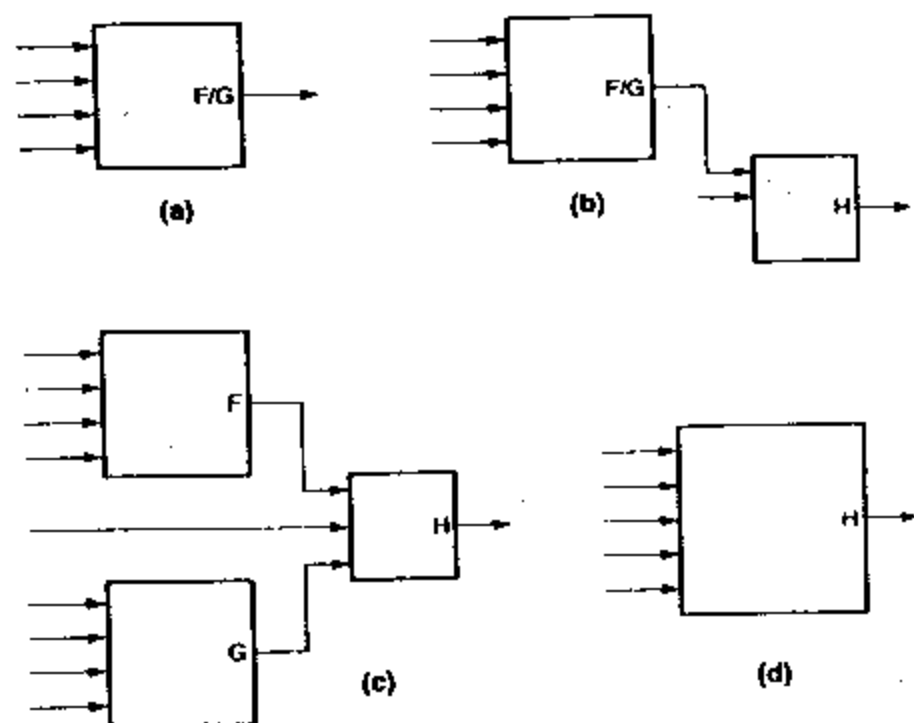


图8.19 可选的功能块大小

这个技术亦便利在实现过程中预先的性能估计,在方块图中的每个块对应于在XC4000数据表中规定的延迟,用一个简单的布线容差(加每个布线 TILO 的50%到100%),为估计设计的可行性是可能的,如果必要,结构设计的改变可以在输入原理图之前进行。

(2)第二个原因:出现无效分割的第二个原因是PPR在面积和速度之间的取舍,对于内在的规则要匹配每个设计的要求是不可能的,但是,分割可以在原理图中利用FMAP和HMAP容易地规定,如图8.20所示。当正在优化一个已有的设计,孤立的FMAP和HMAP

可以增加来改善关键的通道,但是,当利用上面描述的设计技术,对整个设计可以包含 FMAP 和 HMAP 来保证所预期的分割,而要求很少的附加工作,因为 FMAP 和 HMAP 精确地匹配在最底层方块图中的方块。

§ 8.5.2 流水线

即使用最佳的分割,每个逻辑函数有最小的可达到的延迟,如果这个最小延迟太长,函数必须利用流水线分裂成更小的函数。

假设一个设计需要运行在20MHz,它的最坏情况时钟到建立的延迟在表8.5中表示,明显地,这个没有满足它的目标,将仅仅运行高达10MHz。99.4ns 的延迟,54.5ns 被分配于块延迟,不可能靠布局或布线改善来减少,在所有的布线延迟可能减小到绝对的最小值(在 XC4000—5 中约1.3ns)的不相似的事件中,最坏情况时钟到建立的66.2ns 延迟仍将不可能允许运行15MHz 以上,利用更可靠的布线估计(每个布线 T_{ILO} 的50%到100%)设计可能运行在仅为10—13MHz。

为了达到20MHz,有四个选择:利用更快的器件;改善分割;重新构造逻辑使临界通道有更小的深度;或者增加一流水级把函数分裂成两个。当实际情况中这些选择中无一可能时,流水线常是最容易的选择,在一个 LCA 器件中,每个函数发生器有一个触发器在它的输出,如果这个触发器不会为其它的目的利用时,可以用最小的布图变化把它插入到逻辑通道中去,但是,困难是有的,用流水线时触发器引入的等待时间必须无论在哪里都匹配数据和控制信号。在此例中,流水线触发器在 BLOCK5 的输出可以是最有效的,最坏情况时钟到建立对此函数的两半是52.9ns,这仍不满足20MHz 的目标,但是这个接近于足以期望用布线变化来实现此工作。当然,预先考虑这样的问题比在设计几乎完成时解决它们要好得多,在前节中讨论的设计技术提供对检测整个问题更合适的延迟估计,流水线则可能建立进设计,任何时序的复杂性操纵要容易得多。

表8.5 最坏情况的 Clock-to-Set-Up 延时例子

逻辑通道	延时累加
Source clock net:	"CLK10MHZ" (上升沿)
From: Blk BLOCK1 CLOCK	to CLB-R13C12. YQ:
Thru Net NET1	to CLB-R13C11. C4:
Thru Blk BLOCK2	to CLB-R13C11. XQ:
Thru Net NET2	to CLB-R23C11. F3:
Thru Blk BLOCK3	to CLB-R23C11. X:
Thru Net NET3	to CLB-R18C9. C2:
Thru Blk BLOCK4	to CLB-R18C9. XQ:
Thru Net NET4	to CLB-R13C10. F1:
Thru Blk BLOCK5	to CLB-R13C10. Y:
Thru Net NET5	to CLB-R2C11. C4:
Thru Blk BLOCK6	to CLB-R2C11. YQ:
Thru Net NET6	to CLB-R2C9. F3:
Thru Blk BLOCK7	to CLB-R2C9. X:
Thru Net NET7	to CLB-R3C10. F4:
Thru Blk BLOCK8	to CLB-R3C10. X:
Thru Net NET8	to CLB-R10C10. F3:
Thru Blk BLOCK9	to CLB-R10C10. Y:
Thru Net NET9	to CLB-R9C11. G1:
To: FF Setup(D), Blk	BLOCK10
Target FFX drives output	net "NET 10"
Dest clock net:	"CLK10MHZ" (上升沿)

§ 8.5.3 平面布图

具有类似数据通道组织和固定数据宽度的结构设计可以从简单的平面布图得到很大的好处,利用多路开关的总线的设计也是候选的方案,在这节中,讨论与 LCA 器件内逻辑函数绝对布局有关的平面布图问题,相对布局的问题靠后在下节的硬宏单元内讨论。FPGA 的平面布图类似于布图一个原理图,一个允许门在原理图上方便连接的组织通常允许它们方便地在 FPGA 中被连接。但必须记住,总线必须通过逻辑而不是绕过它运行。在大多数情况下,设计本身建议这种平面布图,为了减少布线,具有相同加权的位应该在行中对准,函数应该对最好的数据流来组织,当要求多路开关的总线时,在行中对准位是重要的,因为水平的 TBUF 长线必须用于总线。如果无明显的结构可以被利用,平面布图应该避免。任意的或拙劣的选择布局限制可以妨碍 PPR,导致比 PPR 无限制所运行的结果更坏。在许多平面布图的例子中,仅仅需要考虑数位和函数的相对位置,但是,当布图总线时,绝对位置可能也是有意义的。

如前面描述的那样,每列 CLB 有两个 TBUF 长线,因而,最宽的总线可以安排有阵列中 CLB 列数两倍那样多的位数,这是假设总线要求全长的长线。如果总线可以利用半长总线来实现,中心分裂器可以打开,以安排两倍那样多的位数。即使有足够的 CLB 列可用全长的长线,任何时候有可能的话最好利用半长的线,因为较短的线有较小的电容,速度也就更快。此外,不用的半长线是有用的资源,可以用于其它目的。

垂直长线跨过 TBUF 长线运行,常常是用于控制信号,这些垂直长线也是可分裂的,而且任何时候有可能利用半长线是最好的。平面布图应该尝试完整数据通道到阵列的单个象限内。当一条总线的两位可以利用一条 TBUF 长线的两半段时,这个情况远非理想的,控制信号必须在两列中重复,因为此长线的一半没有重迭,跨过总线运行的算术进位等函数在阵列的两半之间被分裂时也被复杂化。在阵列的一半范围内大到足以包含所有的数据通道,这样利用器件是更好的方式。

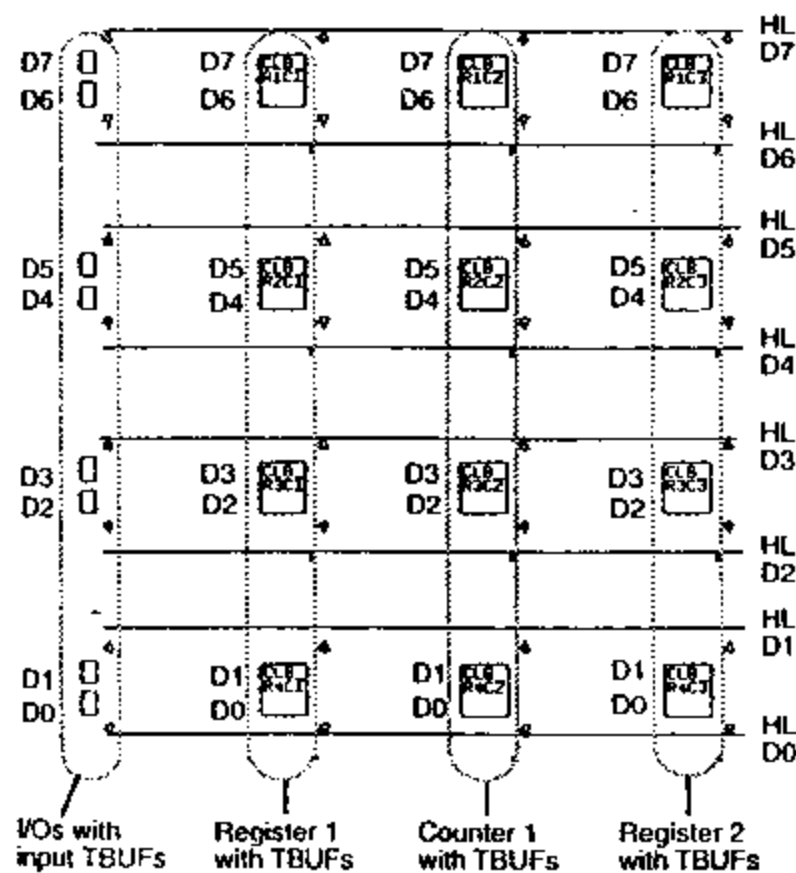


图8.20 三态总线的位对准

在每个 TBUF 长线的每端有一个 I/O 块,而且这些 I/O 块有到长线的好连通性,因而,需要连到内部总线的 I/O 腿应该放置在芯片的左边或右边而不是在会造成不必要的布线拥挤的顶边或底边。当信号被限制在专门的腿上,出腿的数位次序应该匹配总线的数位次序,如果位的次序不同,将又一次导致布线的拥挤,Xilinx 的硬宏单元有其 MSB 在顶部的数位次序,当库的宏单元被利用时,应该适应这个标准。

考虑到图8. 20中所表示的例子,两个八位寄存器,A[7:0]和 B[7:0]和两个八位计数器 C[7:0]和 D[7:0]被转换到一个八位的双向总线,X[7:0]并布线到 I/O 腿,寄存器每个取四个 CLB,并应该安排在一列中,软宏单元计数器亦用四个 CLB,一个 XC4003是10×10阵列,所以每个象限是5×5的阵列,可以包含此逻辑。

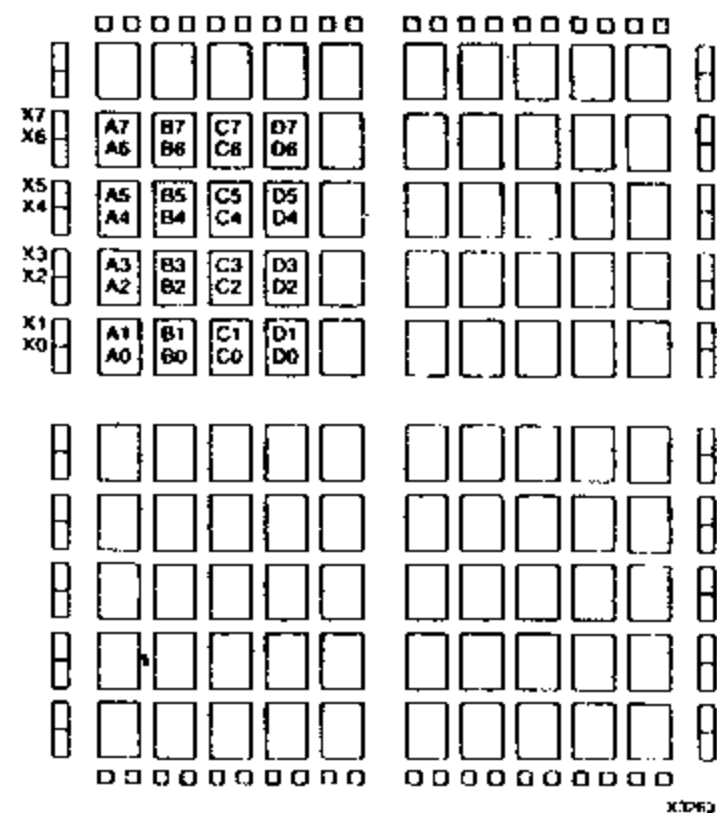


图8. 21 在 XC4003中有效的布局

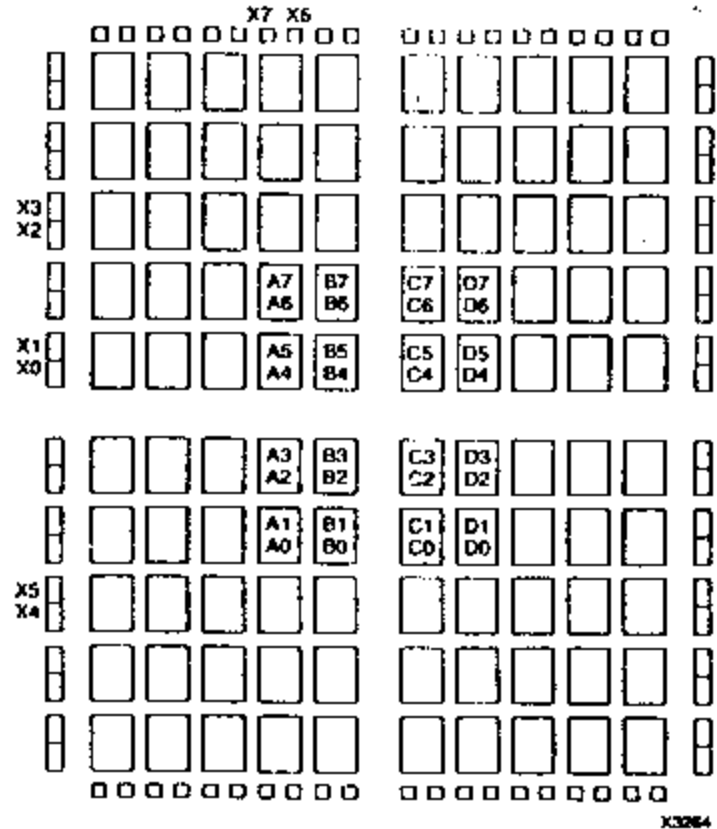


图8. 22 在 XC4003中无效的布局

图8. 21表示了一个好的布局,没有浪费长线资源,这个布局包含总线和它的多路转换功

能在一个象限内,所以它仅仅需要一组水平和垂直长线,总线的 I/O 方便地位于驱动它们的 TBUF 长线的左手边,I/O 布局的第二个选择应该是在 TBUF 长线的右手边,但是这个需要利用全长的 TBUF 长线。

图8.22表示了拙劣的布局例子,它利用水平长线的两个半段来构造三态总线,而且附加地可能必须利用垂直长线的两半来布线使能信号到 TBUF,当总线 I/O 位于图的顶部或底部(X6和 X7),对它们布线是困难的,X0—X5也没有和总线对准,而要求额外的布线资源,这个实现比前面的实现更慢且耗费更多资源。

§ 8.5.4 硬宏单元

在前节中,部分设计被分配到 CLB 阵列的固定位置,在这些固定位置被拙劣地选择时,对 PPR 实现设计时可能是更困难的。例如,可能有许多信号必须接到位于阵列顶部附近的逻辑的顶层,所以迫使 PPR 产生不希望的远离其目的地的信号。这个复杂性可以用产生硬宏单元来去除,硬宏单元让设计者规定临界逻辑单元相互有关的布局,而允许 PPR 自由地放置这个单元组到对实现此设计最方便的位置,除非需要规定 I/O 或长线资源,相关布局提供了没有过多限制 PPR 的绝对布局的好处。

但是在产生硬宏单元时有一些因素必须被考虑,首先,所有的硬宏单元是矩形的,非矩形的 CLB 组变换到矩形组要大到足以包含原始的 CLB 组。在此过程中,增加了对 PPR 无用的空的 CLB,而且将保持在最后的设计中不利用,在硬宏单元中的洞也将保持不用,留下太多的 CLB 不被利用将引起设计增长得比本应能适合它的器件要大。

其次,在硬宏单元中的函数发生器是关闭的,即没有附加的逻辑可以增加进其中。用一个规则的宏单元,驱动控制腿的倒相器将吸收进宏单元,改变所有控制腿所接到的函数发生器,但不要求附加的资源。用硬宏单元,这一点是不可能的,在硬宏单元的外边,一个附加的函数发生器可能是实现这个倒相器必须的。

第三个因素是腿的可交换性,PPR 常常交换函数发生器的腿来改善布通率,但是,当一个硬宏单元包含一个进位逻辑函数,所有的腿锁定在利用进位逻辑的 CLB 上,因为 PPR 不再可以交换腿来改善布通率,当产生宏单元时,出腿必须仔细选择。硬宏单元由包含刚好为所需 CLB 的未布线的 LCA 设计来产生。lca 文件利用程序 HMGEN 被变换到硬宏单元。

§ 8.5.5 锁定 I/O

PPR 允许用户锁定 I/O 信号到规定的腿,被锁定的腿某些时候必须容易挤到印制板上,匹配已有插座的输出腿,或者简单地允许印制电路板设计在 FPGA 设计完成之前开始。但是与任何其它的限制一样,锁定腿限制了 PPR,可能会降低性能,或甚至阻止设计完全地布线。任何 I/O 的锁定应该尽可能在设计过程的靠后进行,如果可能,在 I/O 被锁定之前设计的75—80%应该完成,在此阶段,对 FPGA 设计优选腿的位置应该知道,在 FPGA 的需要和印制板的需要之间可行的折衷应该做到。当较早地锁定腿可能方便的时候,存在的危险是腿的限制将阻止 FPGA 设计不能成功地完成,当此发生时,唯一的解决办法是移去对腿的限制,因此已经完成的印制板的设计也无效了。

§ 8.6 软件技术

§ 8.6.1 XACT 的“性能”

用 XACT 的“性能”(Performance),可以在 XC4000设计中规定时序的目标,这些目标主要由 PPR 优化它的布线资源的利用所用到,当一个临界线需要已经分配到非临界线的特定布线资源时,时序驱动布线器可以选择重布非临界的线,因此使得所需的资源有用,时序目标允许 PPR 做这样的智能舍取,告诉它哪条线是临界的,有多少自由度降低非临界的线。

XACT“性能”能做的事情包括:

1. 提供临界布线容易理解的控制
2. 允许 PPR 在临界区域更多的自由度
3. 任何时候可能的逻辑深度的补偿
4. 当期待的时序不可靠时提前给出警告

XACT“性能”不能做的事情包括:

1. 执行延迟的匹配
2. 允许任意深度逻辑
3. 保证时序规定将被满足

有效地利用 XACT“性能”要求理解两个基本的概念,利用前向追踪机构将线的属性施加于通道,和在多个属性用于相同的通道时采用仲裁机构。

1. 向前追踪

虽然 TIMESPEC 属性施加到连线,但是它们规定触发器之间的通道延迟或者触发器和 I/O 之间的延迟。每个 TIMESPEC 施加于一组触发器,它们是用向前跟踪从带有 TIME-SPEC 属性的连线出发来决定的, TIMESPEC 施加于带一个输入的任何触发器,可从 TIMESPEC 连线任一方向或通过任何深度的组合逻辑到达其输入。

2. 多个属性

在许多情况中,多于一个的 TIMESPEC 将向前追踪到相同的触发器,按照 TIMESPEC 到达的触发器的腿的类型来解决这样的矛盾,这个到达腿的属性列在表8.5中,对此 TIME-SPEC 属性选择相应的连线, TIMESPEC 的特性可以在原理图中设置,如果多于一个 TIMESPEC 有相同的特性,最快的 TIMESPEC 赢得,对 C2S, P2S 和 C2P 的规定分别进行仲裁。

表8.5 多属性的指标级别的规定方法

级别	原理图中规定方法	说明
1	不依附属性 DP2S, DC2S 或 DC2P 领先的 D 表示缺省属性	加到设计中所有触发器 可用2级或3级规定复盖
2	TS 属性 C2S, P2S 或 C2P 加标志到可向前追踪到触发器时钟腿的线上	加到由向前追踪达到时钟腿的所有触发器 可用3级规定复盖
3	TS 属性 C2S, P2S 或 C2P 加标志到可向前追踪到触发器输入腿的线上	加到由向前追踪达到非时钟腿的所有触发器 可用1级或2级规定复盖

一旦触发器已被分配 TIMESPEC, TIMESPEC 可以分配到通道, 对于一个通道的 TIMESPEC 是它的源和目的上最快的 TIMESPEC, 如果源或目的触发器任一个没有 TIMESPEC, 此通道没有 TIMESPEC, 当缺省的 TIMESPEC 已经设置成 IGNORE, 可以出现这种情况。如上所述, XACT"性能"主要靠按照网线的临界性分配布线资源来运行的, 赋予临界网线最大的存取布线资源的最简单策略是使非临界网线的临界性最小。但是, PPR 计算不带有用户的 TIMESPEC 的通道的缺省规定, 而且某些时候这些缺省是不必要要求的, 如果非临界通道的目标减少到匹配设计的要求, 临界通道可以受益。按照"最快者赢"的规则, 不附有属性的 DC2S, DP2S 和 DC2P 应该对每个通道类型置于最慢的要求, 其它的最快通道则被明确的规定。另一方面, 如果有"不考虑"的通道, 不依附属性可以置于 IGNORE。

PPR 产生缺省规定的另一种方式是修改一个触发器的 C2S 的规定以提供丢失的 P2S 或 C2P 的规定。另外, 这些缺省项可以不必要求, 这个问题的解决靠提供丢失的规定, 或置 EXTEND-C2S=FALSE 释放缺省机构。当 PPR 为一个设计产生缺省的规定时, 它根据设计中的逻辑深度估计典型通道的延迟。相同的估计可以用于任何 TIMESPEC, 只要置延迟到 AUTO, 而不是置 ns 或 MHz 的数。当可靠地估计可达到的延时无效时, AUTO 是有好处的。过于宽松的规定导致不必要慢的结果, 因为一旦已经满足规定的目标, PPR 停止改进此设计, 不可能的短延迟规定等于是无结果, 当优化器尝试执行不可能的任务, 它常常将产生极其长的通道延迟, 设计将比利用自动延迟更慢。无论何时 AUTO 在 TIMESPEC 中对网线调用, 典型的通道延迟根据其上加有特定 TIMESPEC 的逻辑来计算。每个自动的 TIMESPEC 对它控制逻辑产生不同的规定, 因而, 多个 AUTO 对此设计的不同部分导出不同的速度目标, 调查这些差别对包含必须是快的简单逻辑和可以较慢的复杂逻辑两种情况的设计是有好处的。自动的 TIMESPEC 应该放置在设计中, 以使每个 TIMESPEC 控制具有特殊复杂性和速度要求的逻辑, 这个安排产生简单逻辑比复杂逻辑更快的目标。对于整个设计单个自动的 TIMESPEC 将按照最复杂的逻辑设置目标, 并不企图做到使简单逻辑更快的目标。

经常要考虑的因素是 XACT"性能"的告警信息[pa;SPEC_DOES_NOT_APPLY], 这不一定意味着规定被错误地输入或已被 XACT 忽略。如前所述, 多个 TIMESPEC 常常追踪到单个触发器, 其中仅仅一个可被应用, 在去除掉多余的规定之后, 仲裁器用告警的形式记录它的活动。当相同的 TIMESPEC 应用到许多网线和其中的几个追踪到单个触发器, 告警的数目增加, 到达触发器的每个拷贝作为分别的规定被处理。因此增加了必须被消除的规定的数目和被产生的告警的数目, 仔细的放置 TIMESPEC 将减少告警信息的数目。

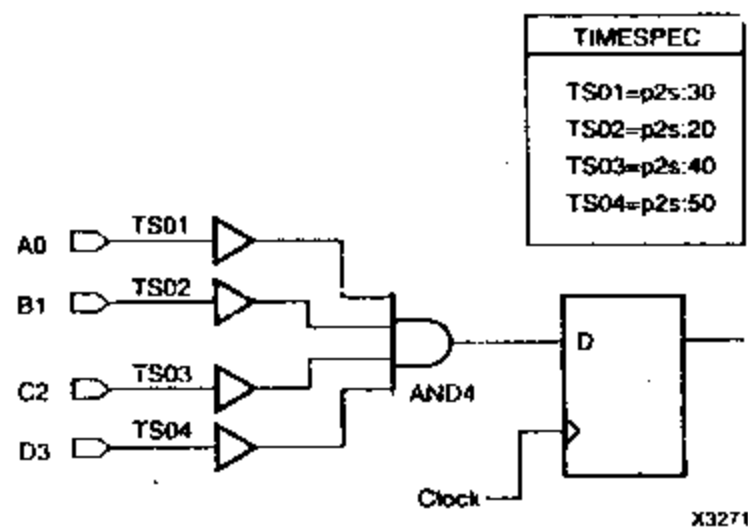


图8.23 规定被拒绝的 TIMESPEC 安排

但是,有时常会 P2S 或 C2P 的规定可被完全拒绝或忽略,例如,图 8. 23,所有四个 TIMESPEC 追踪到触发器的 D,并且最快的 TS02 赢得,TS01,TS03 和 TS04 被搁置,图 8. 24 表示一个甚至最坏的例子,TS02 又一次赢,但这被限制于用“B”开始的名字的焊盘到通道(通配符组 B*),因而,从 A0、C2 和 D3 的通道被作为不规定处理,不管它们的 TIMESPEC。

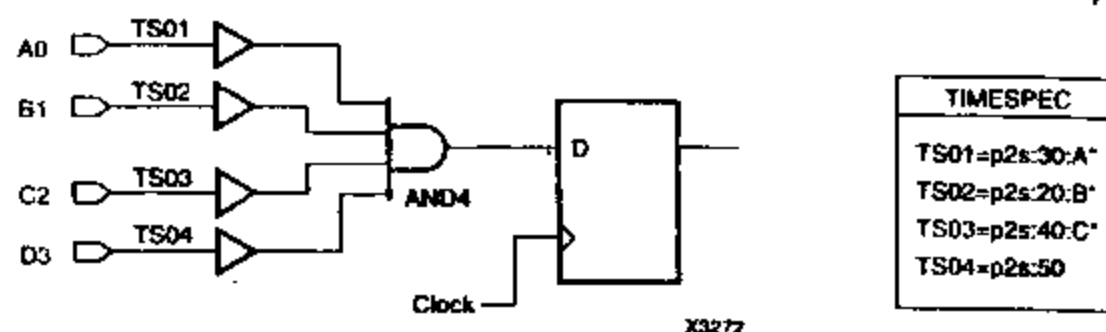


图 8. 24 规定被丢失的 TIMESPEC 安排

这个解表示在图 8. 25 中,当多个 P2S 的 TIMESPEC 放置于单个网线上,它们被确认用分别的焊盘运行,当此网线在仲裁中赢得时所有的规定被保留,全部规定则由 XACT“性能”利用,每个规定来自确认的焊盘组的通道。在此例子中,仅有一条网线用 TIMESPEC 配属,所以它必须赢,所有四个规定被保留,因为它们来自赢了的网线,TS01 是为 A0 所利用,因为它确认为 A* 的组,类似地,TS02 为 B2 所利用,TS03 为 C2 所利用,TS04 为所有不在 A*, B* 或 C* 组中的焊盘所利用,在此情况,仅仅 D3 剩下被 TS04 所控制,为了简化此原理图,四个 TIMESPEC 可以连接成单个的规定,TS05 这单个 TIMESPEC 依附于此网线代替其它四个。

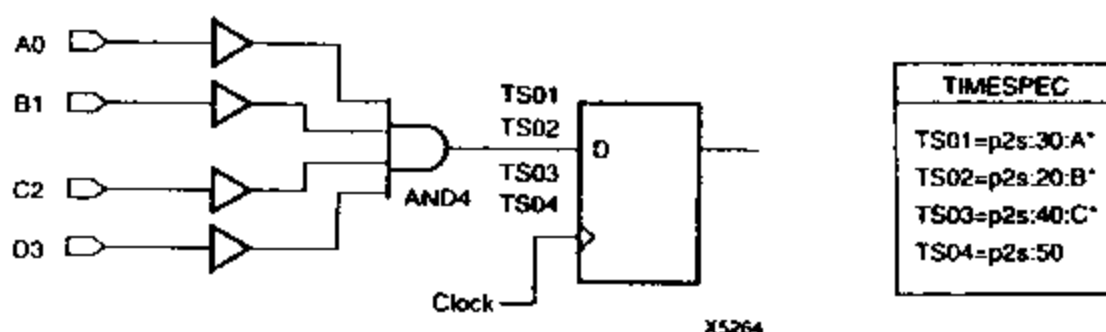


图 8. 25 所有规定被保留和应用的 TIMESPEC 安排

当产生定制硬宏单元供 XACT“性能”使用时,以下这些准则将帮助 XACT 更有效地分析此设计

- * 当有时钟腿在硬宏单元上,确认用 C 命名它,这允许 PPR 识别在宏单元中存在触发器,当无时钟腿在硬宏单元上,无其它腿被命名为 C。

- * 避免产生具有多于一个时钟的硬宏单元;因为 PPR 仅可识别每个硬宏单元一个时钟输出。

- * 避免组合定时的和非定时的输出在相同的宏单元,当 PPR 在一个硬宏单元上检测一个时钟腿,它假设硬宏单元的所有输出是用触发器为源。

当利用带有 XACT“性能”的硬宏单元时存在两个问题。第一,PPR 在实现的分割和布

局阶段期间内不可能充分地分析硬宏单元的时序,这个可能导致 PPR 过早地预测定时要求可能被满足,在布线期间被证明是不真实的。第二,分析通过进位逻辑的错误通道可能导致定时要求不可以被满足的错误标识。

在 PPR 的 V1.31 版本中已经解决了这两个问题。但是,在 PPR 和 XDELAY 的定时分析之间仍有某些差别。当分析时钟到建立的通道时,PPR 不管时钟的扭曲,当时钟分布在利用 BUFGP 或 BUFGS 缓冲器的全局网线时,这是合适的,但如果时钟是在局部网线上被布线,此扭曲可能是显著的。在设置 TIMESPEC 必须考虑。为确保正确运行,所有完成的设计应该用 XDELAY 仔细地分析,当时钟扭曲可能存在时这是特别确实的。因为 XDELAY 的分析选择没有考虑时钟扭曲,一个分别的时钟分析必须执行,如果 PPR 和 XDELAY 定时结果不同,XDELAY 的结果应该被利用。

§ 8.6.2 PPR 选项

像它的名字所表示的,PPR 执行三个主要的功能:分割(partitioning)布局(placement)和布线(routing)。在分割阶段,原理图中的门被分配到按大小分的函数发生器组中。然后在布局阶段这些组 and 任何附加的触发器被分配到规定的 CLB。首先,产生初始的布局,一般利用最小截(Mincut)算法,这个布局然后由分析连线距离来改善。布线是一个迭代的过程,早先的连线常常为以后布的连线释放临界的资源而重新布线。

下面介绍 PPR 较重要的选项,和这些选项推荐的设置,最后给出两个 PPR 设置的例子。

1. DC2P={ (number), "auto" "Ignore" }

DC2S={ (number), "auto" "Ignore" }

DP2S={ (number), "auto" "Ignore" }

DP2P={ (number), "auto" "Ignore" }

输入方法:指令行,参数文件或在 XACTIMAT.DAT 文件中/prdx2x。

参数是字符串分别代表 XACT"性能":

缺省的时钟到焊盘: Default—Clock—to—Pad (DC2P)

缺省的时钟到建立: Default—Clock—to—Set—up (DC2S)

缺省的焊盘到建立: Default—Clock—to—Set—up (DP2S)

缺省的焊盘到焊盘: Default—Clock—to—Pad (DP2P)

通道延迟的目标可以以十分之一纳秒来规定(0.1到3000.0),“auto”由 PPR 单独地自动设置,“Ignore”要求对特定的通道类型忽略时序(除了在原理图中定义显式的技术条件)。

例如,当无显式的 TIMESPEC 在 P2P 通道,设置 DP2P 为“Ignore”,将使 PPR 不管在设计中所有的 P2P 通道,如果设计中没有 P2P 要求,这个设置帮助 PPR 满足存在的 TIMESPEC 的要求,当 DP2 被置为“Ignore”,处理 P2P 通道的显式 TIMESPEC 被兑现,而且不仅带有 P2P 的 TIMESPEC 的通道被优化;在其它重要通道上的延迟可能增加,对这些参数的缺省设置是“自动的”。

推荐的设置:

Ignore: 当不考虑未规定的通道

(number): 当试图达到一个规定的延迟

auto: 让 PPR 决定相当的延迟去试

2. extend_c2s={True, False}

输入方法: 指令行, 参数文件, 或 XACTINIT.DAT 文件中/PPR/extend_c2s

当变量是 True, PPR 自动地对于每个无对应 P2S 或 C2P 的 C2S TIMESPEC 产生一个 P2S 和 C2S 的 TIMESPEC, 这个规定被分配相当的数值, 为忽略 P2S 和 C2P 通道, 除非它们有显式的 TIMESPEC 置变量于 False, PPR 则可以集中于 C2S 的 TIMESPEC, 而不必优化 P2S 和 C2P 通道, 这个选项的缺省设置是 True。

推荐的设置:

False: 避免不必要的要求, 自动地规定。

True: 扩展 C2S 的规定到 P2S 和 C2P 的通道。

3. ripup_allowance=(number)

输入方法: 在 XACTINIT.DAT 文件中/PPR/mxmazer/ripup_allowance=(number)

这个选项告诉布线器在分裂和重试过程中允许多少次回归, 较高的允许值将增加一个设计成功地被完成的概率, 因为布线器试验更长了, 例如, 当布线器到它的允许值关闭, 找出一个未布线的数目中最小的, 它可能停止, 用一个更大的允许值, 它可能持续下去达到一个更好的结果, 虽然未布线的数目可能暂时增加。

当运行 PPR 多次以获得更好的布局时, 置低的 ripup_allowance, 否则 PPR 可能化更长的时间来产生一个设计, 多少由于拙劣的布局没有完全的布线, 由于置参数低, PPR 可以更快地产生结果, 而且继续到循环的下一迭代。如果一个设计有带有小的 ripup_allowance 的几百条未布线, 增加允许数将可能不使它完成布线, 但当未布线的数目是 0—30, 重新运行具有较高的 ripup_allowance 的 PPR 可能将完全把设计布线完。对这个参数的缺省值是 2。

推荐的设置

—1 穷举地分裂和重新布线, 当设计将布线或接近于布线时用

5—10 当不确定设计是否将布线时用, 或当运行 PPR 循环时

4. improvecount=(number) PPRPPRPPR=3206—8

输入方法: 指令行/参数文件

这个选项设置使布局改善的 PPR 迭代数, 这个数愈高, PPR 企图改进布局的时间愈长, 当布局容易改进时, PPR 以它已获得的最好结果停止下来, 而不管设置的数值, 缺省值=3。

推荐的设置:

20: 当运行时间不是一个因素

6—8: 通常给出好的结果

5. seeds_to_try=(number)

输入方法: 指令行/参数文件

这个选项定义在布局改善阶段 PPR 试验多少次不同的种子, 对于规定的种子数的每一个, PPR 进行一个初始的布局并运行一个改善的迭代, 在这之后, 布局算法连续为 (improvecount—1) 迭代, 以有最好分数的结果开始, 缺省值=1。

推荐的设置:

10: 当运行时间不是一个因素

3—4: 通常给出好的结果

6. mincut_passes=(number)

输入方法:指令行/参数文件

这个选项设置每个分割尝试最大的改进遍数,当任何遍数表示无改善时,分割提前停止,缺省值=12。

推荐的设置:

20: 当运行时间不是一个因素

12: 相当的值

7. mincut_method=(number)

输入方法:指令行/参数文件

这个变量选择在起始布局期间利用的算法,合法的值是0—3,缺省的值是方法3,它已经证明是所有中最好的,但是,在特定的设计中,方法0—2可能产生更好的结果,这些其它的方法对于有低的 I/O 利用率和/或触发器和函数发生器的利用率之间差别大的设计是最有用的。(对利用的百分比检查 PPR. LOG 文件)。在布局用方法3达到时利用这些方法是不适合的,没有保证布局将改善,但是很少的试验可能是十分有价值的,缺省值=3。

推荐的设置:

3: 轮回因素通常产生好的结果

0,1,2 改善布局利用的试验

8. mincut_tries=(number)

输入方法:指令行/参数文件

这个选项设置每个分割步骤初始配置的最大数,当构成一个好的初始分割时,并改进它,在每个分割步骤做的试验次数,由这些试验的最好结果在此设计中被利用。缺省值=2。

推荐的设置:

10: 如果运行时间不是一个因素

2: 对于相当大的量

9. justflatten=(True, false)

输入方法:指令行/参数文件

当这个选项被设置为 True,设计被简单地整平进一个, LCA 文件;此设计既未布局又未布线,如果存在硬宏单元被合并进此设计。利用这个选项,一个设计为了返回注释和单位延迟仿真很快转换成一个, LCA 文件,当有硬宏单元在设计中时这仅仅是必须的,而且要求单位延迟仿真,缺省值是 False。

推荐的设置:

以下 PPR 选项的设置被推荐,为了比运行较长时间的缺省值设置得出更好的结果,利用这些值。

这指令行或参数文件中:

improvecount=7

seeds_to_try=3

extend_c2s=false

DP2S=ignore

DC2P=ignore (如不考虑未规定的通道)

DP2P=ignore

在 XACTINIT.DAT 文件中/PPR/mxmazer/ripup_allowance=8

为了比运行较长时间的缺省值设置得出好得多的结果,利用以下数值在指令行或参数文件中:

improvecount=20

seeds_to_try=10

extend_c2s=False

mincut_passes=20

mincut_tries=10

DP2S=ignore

DC2P=ignore (如不考虑未规定的通道)

DP2P=ignore

在 XACTINIT.DAT 文件中/PPR/mxmazer/ripup_allowance=-1

XDELAY 静态时序分析器

XDELAY 提供了 Xilinx FPGA 设计的静态时序分析,它分析通过一个设计的所有逻辑通道,并产生一个可以自动地对最坏情况性能分析报告,逐行观察,或存贮作为一个文件。因为,对单个通道的 XDELAY 报告可以要求许多行的文本,大的设计可以包含几千的通道,XDELAY 的报告文件可以是十分大,有些时候占有几兆字节的硬盘空间,这些报告的大小可以利用提供的滤波器显著地减少。

报告可以限于单个通道类型,例如对 Clock-to-set-up,而去除了不感兴趣的通道类型的报告,对于最坏情况的通道分析,设置 Delaygreater 变量可以去除太快而不被关心的通道,利用 From、to 或 Ignorenet 滤波器限制搜索范围可以执行局部化的分析。PPR 亦提供静态时序分析作为它的报告的一部分,有些时候,XDELAY 和 PPR 的时序结果有所不同,这些差别或者因为 PPR 没有追踪确定的通道,或者因为不正确地分析某些延迟。XDELAY 是最后正式的 Xilinx-FPGA 时序分析器,它的结果应该比由任何其它来源的结果更有权威性,XDELAY 分析应该是全部设计的一部分设计过程。

以下的情况可以导致 PPR 追踪与 XDELAY 不同的通道:

- * PPR 没有从直接复位或直接置位腿到触发器输出的非同步通道追踪,但是用缺省值,XDELAY 通过非同步通过非同步通道追踪,XDELAY 选择 Flagblk CLB_Disable_SR_Q 释放此追踪。

- * PPR 把 RAM 文件处理作为组合逻辑并连续追踪下一时钟单元或 I/O 腿

- * PPR 没有通过双向 I/O 块追踪,象 XDELAY 有时做的那样。

引起 PPR 分析不正确的情况如下:

- * PPR 总是利用 T_{IO1} 作为通过一个 TBUF 的延时, T_{IO1} 是从 TBUF 的 I 腿到输出的延时,因而,从 TBUF 的 T 腿的延时将和 T_{IO1} 延时不一致,在某些情况中, T_{IO1} 比正确的延时要小,WAND 和 WORAND 元件是用 TBUF 实现的,所以也受此问题的影响。

- * 如果 BUFGP 是来源于内部逻辑,PPR 必须通过专门的 IOB 布线信号输出,并返回进 BUFGP 的输入,在计算这个通道的延时中,PPR 假设输出驱动器是为快速扭曲率配置

的,而通常不是此情况,因此可能低估这个延时。

* 由于 PPR 和 XDELAY/LCA2XNF 之间的模型差别,一般少于1ns 的小的不符出现在各种地方。

§ 8.7 XC8100 FPGA 的设计实现

利用 XC8100开发系统(XDS)进行设计时,XDS 采用许多新的特性以利用新结构的优点:命令壳体和统一的数据库结构;新的布局布线软件;为综合类似 ASIC 的设计流程;输入网表和器件内 CLC 之间的一对一映射;保存整个设计流程的层次信息及原始网线和样例的名字;为 CAE 工具接口的 EDIF 和 SDF 格式。类似于 Xilinx 的 FPGA,XC8100 FPGA 的设计流程是根据三步的设计过程:

1. 利用标准的 CAE 工具做设计输入和功能校验;
2. 利用 XC8100开发工具做设计实现;
3. 利用标准的 CAE 工具做设计校验。

1. 设计输入

首先利用各种方法描述设计:Verilog,VHDL,原理图,方程和状态机等,在此阶段,设计者利用标准的 CAE 工具,Xilinx 提供为第三方 CAE 工具的设计包(综合模块和原理图库)。XC8100结构的灵活性意味着用户可以不考虑结构或实现的细节来设计,在大多数情况下,软件有前面所述的能力自动地选择。

2. 设计实现

设计过程第二步取设计输入阶段的 EDIF 网表输出,包括读设计网表;不改变任何逻辑把网表的基本单元放到专门的 CLC,这个保证设计和实现之间的一一对应;布线所有的网线,布线器利用割裂再试(rip-up-and-retry)算法以移去阻塞;对所有的网线计算实际的时序延时,这些信息用于时序分析和由第三方时序/仿真工具使用;显示物理层上设计的视图;产生为编程硬件用的熔丝映射和文件。

设计过程的关键是在逻辑综合期间进行工艺映射,称为 TrueMap 的一对一映射过程保留输入 EDIF 网表中的样例名,网线名及层次。在设计中的每个逻辑门对应 FPGA 中 CLC 的一个逻辑门,在设计中的每条网线对应 FPGA 中的一条网线,这个信息对返回注释、仿真和诊断是有用的。增量设计一般在综合环境已不见效,因为样例名,网线名每次运行都会改变,而大多数增量设计算法依靠示例名,网线名的匹配,8000系列软件采用独特的算法匹配两个设计的拓扑,而不是名字,这就允许在综合过程中利用 PowerGuide 的增量设计能力。

编程 XC8100 FPGA 类似于任何 OTP 的 FPGA,有几种方法提供编程:XC8100软件包含样品编程器卡;Xilinx HW130编程器;被选用的第三方编程器等。

在器件上仅五个腿为编程接口要求,JTAG 的 TDI,TDO,TCK 和 TMS 用来寻址被编程的元件,移入编程数据和校验,Vpp 腿提供编程时的高压,并用来测量被编程反熔丝的电阻,编程时间每一千门约一分钟,例如,XC8103约需三分钟。编程算法包括测量被编程 MicroVia 反熔丝的电阻以保证器件的速度和功能,因为这是在工厂测试期间不可校验的项目,它完成器件的100%功能测试。XC8100器件是100%可测的同时,在编程器硬件上做编程后测试是可能的,XC8100软件可取仿真矢量,并把它们加到器件和通过 JTAG 口读回。

3. 设计校验—第三步中设计者利用第三方 CAE 工具分析或仿真 XDS 输出的时序。

第九章 Xilinx FPGA 的配置

配置是加载设计规定的编程数据到一个或多个 LCA 器件中的运行过程,以定义器件内部功能块和其互连的功能,这个过程多少有点象加载可编程外设芯片的指令寄存器。Xilinx 的开发系统可以输入,布局,布线和校验 FPGA 的设计,在这些完成之后,可以利用程序 MakeBits 和 MakePROM 来产生配置文件,最终的配置数据被加载到 FPGA 的配置存储单元,它们控制可配置逻辑块(CLB)和 I/O 块的查找表函数,多路开关和互连资源,例如,XC4000 系列每个 CLB 大约利用了 350 位配置数据,每个配置位定义静态存储单元的状态,或者控制函数查找表的位,多路转换器的输入,或者是互连的传输晶体管。

§ 9.1 数据产生

XACT 开发系统把一个设计首先转换成网表文件,然后自动地分割,布局和布线来实现设计的逻辑,并按 PROM 格式产生配置的数据。在开发系统中,一旦已经为一个设计产生了一个 LCA 文件,可以利用程序 MakeBits 和 MakePROM 从 XACT 的设计编辑器(XDE)内产生配置的数据文件,这些文件然后可以用于配置 FPGA 器件。

§ 9.1.1 MakeBits

在 XDE 内运行的 MakeBits 程序利用 LAC 文件产生专门的 FPGA 配置数据。在 MakeBits 程序中,可以利用配置菜单的配置指令决定器件的选择,这些选项也可以在屏幕左下角的程序选项表中用鼠标直接设置。MakeBits 的图面(Profile)提供器件选择的缺省设置,可以执行 Saveprofile 指令用要求选的选项修改这些缺省值,为了以后的使用,保存的 MakeBits. pro 文件可以读入到 MakeBits 中,在配置菜单中用要求选的器件选项执行 MakeBits 指令,为现在加载的设计产生位流文件(.BIT 文件)。

MakeBits 首先用以下选项提示:

Tie, Norestore, Verbose, UseCriticalNetsLast, IgnoreCriticalNetFlags, MakeLL

如果这些选项中无一被选择,选择 Done 在存储器中产生一个位流,它可以直接由加载菜单用 Download 选项加载到器件,对室温试验板的快速逻辑校验可以利用“无约束”形式,其中增加的供电电流不是临界的。

下面讨论在 MakeBits 过程中的两个步骤:约束设计和产生位文件。

(1)约束设计:产生的设计要求利用 Tie 对设计中未利用的 FPGA 资源定义有效的逻辑电平。未利用的互连产生 CMOS 浮栅输入电平,它可以导致瞬时的 p- 和 n- 沟道导通,引起附加的供电电流,热和电噪声,利用 Tie 选项定义所有无用的 CLB 和 IOB 的输出为逻辑低,并利用它们驱动可使用的未用布线资源。

剩下的未用资源则被加到在 EditLCA 的 Net 菜单中用 Flagnet 指令未标志为临界的网线,这个可能导致与原始的 LCA 设计文件有关的时序变化,附加选择 Norestore 选项把约束设计的暂时型式保留在存储器中,允许利用具有 Tiechange 选项的 Querynet 指令以估计 Tie

(约束)操作对时序的影响,为保护原始的设计文件,设计的约束型式不保存在硬盘上,除非有特殊要求。在某些情况中,如果有太多的临界网线,设计不可能约束,在这个时候,可以利用 UseCriticalNetsLast 选项,这个选项使 MikeBits 在设计中利用临界网线的选择,但仅仅作最后的手段,如果 MakeBits 仍不能约束此设计,可以利用 IgnoreCriticalNetFlage 选项,这个选项告诉 MakeBits 忽略在设计中发现的临界标志。如果利用 UseCriticalNetsLast 或者 IgnoreCriticalNetFlags 选项,Xilinx 极力推荐对设计运行 XDelay 程序以确保设计临界网线的时序。未用和未约束的 I/O 块本来也要定义焊盘电压,在 XC3000 和 XC4000 系列中,这是用内部上拉电阻自动完成的,作为未配置 I/O 块的缺省值对于 XC2000 系列,焊盘电平可由外部信号定义,上拉电阻或输出缓冲器可以被使能,并由外部信号驱动。

(2)产生位文件:WriteBits 把产生的位流保存到文件中,这个二进制格式(BIT)文件可以被 Download 电缆或 Xchecker 电缆利用来配置电路内的 FPGA。一个用户可读的配置数据的 ASC II 文件也可以用在配置(Config)菜单中找到的 RawBits 指令来产生,这个文件对校验加载进 FPGA 的实际位次序是有用的。

§ 9.1.2 MakePROM

MakePROM 程序把 BIT 文件变换成编程器兼容的数据文件,然后,可以利用这些文件对 PROM 编程,提供用于定义 FPGA 运行的配置数据。MakePROM 也附加 BIT 文件,并产生一个合成的 PROM 文件,它包括对一个器件链的总长度计数和配置数据。这个链称为菊花链,在菊花链中,多个器件以串联的型式从相同的 PROM 文件中被配置,前面的器件首先被配置,每个相继的器件按它们在板上的排列的次序被编程。MakePROM 的缺省选项由 MakePROM 的图面来决定,如果缺省的 Promsize(PROM 规模)设置是不正确的,利用 Misc 菜单的 Set 指令,PROM 菜单的 Format 指令选希望的 PROM 编程器格式,由 MakePROM 写的格式是 Mcs86(Intel),Exormax(Motorola)和 Tekhex(Tektronix),大多数 PROM 编程器支持这些格式的一个或多个。

以下两个选择描述最通常的功能:

(1)加载>Loading):一旦设置格式和规模选项后,可以加载 BIT 文件进系统存储器,在 Prom 菜单中找到 Load 指令按希望的次序进行,Load 指令提示十六进制的 PROM 起始地址和地址顺序的方向(向上或向下),然后 Load 指令显示现在目录的 BIT 文件,为了配置多个 FPGA,按次序选择多个 BIT 文件——首先为前面的器件,当并行地配置有相同设计的多个文件,仅需选择 BIT 文件一次。也可以产生包含用各自的起始地址交替地配置的 PROM 文件,使得可以控制高位的 PROM 地址和在相同的 PROM 内从不同的段选不同的配置。

(2)保存>Saving):用 Save 指令产生一个 ASC II 文件,它可以转换到普通的 PROM 编程器,这个16进制字符文件包含标题,地址,PROM 数据和校验信息,标题信息识别记录类型,记录规模和记录的第一数据字节的地址,PROM 数据包含 FPGA 的配置数据。

§ 9.2 数据格式

位流的两个通常的类型是单个器件的位流和菊花链的位流,在两种情况下,数据位流由起始码和程序数据组成。起始码包含起始序列位和规定配置的器件需要的配置数据位数的

长度计数,程序数据包含配置器件的每个信息段需要的信息。程序数据也包含确保最后一个信息段数据被加载的尾位序列。

§ 9.2.1 产生位流

必须对每个 FPGA 设计产生位流,MakeBits 程序产生一个相应格式的文件,如果器件是菊花链的,MakePROM 为此链产生一个 PROM 的16进制文件,对此链中的第一个器件规定位流,(BIT 文件),然后规定第二个器件的位流,以此类推。

§ 9.2.2 单个器件位流

单个器件位流的40位起始码,以最小的八个1(伪程序位)开始,跟随0010预兆码,预兆码后跟着24位的长度计数,起始码以4位1的分隔字段结束,如表9.1所示。

表9.1 40位的起始码

11111111	0010	(MSB)···24位长度计数···(LSB)	1111	数据
----------	------	-------------------------	------	----

单个器件位流的程序数据包括配置数据的各个信息段,尾位和填充位。信息段数和构成信息段的位随不同的部件改变,但基本的分解十分类似。

配置数据的信息段被分裂成起始位,配置位和停止位,每个信息段由0起始位开始,跟随对器件的实际配置数据位,XC2000和 XC3000的部件有3个1作为它们的停止位,XC4000部件有一个4位的字段,或者是缺省的0110,或者计算的 CRC 校验,在 MakeBits 的选项中选择时,CRC 校验码是基于在当前器件中所有配置位的累加。XC2000和 XC3000的器件尾位是1111,对 XC4000器件尾位是01111,对每个器件合成的尾位数和信息段位变化数的差别意味着字节边界的填充位的数相同的变化,结束位的完全序列是尾位和填充位的组合,所以它的长度随不同的部件而改变。

§ 9.2.3 菊花链位流

菊花链位流也包含单个合成的40位起始码,起始码很象单个器件位流的起始码,因为它以最小的八位伪程序位跟随预兆码开始的,起始码包含一个24位长度计数,但是,长度计数对菊花链所有器件是公共的,菊花链起始码也用4位填充字段结束。

一个菊花链位流的程序数据包含每个器件的程序数据,跟随它的尾位,对整个菊花链以填充位结束,对每个器件的配置数据信息段以实际器件在板上排列的次序出现,前面的器件在程序数据的开始,在菊花链链中的最后器件是在程序数据的末尾,每个器件的一组数据包含器件的尾位(1111对 XC2000和 XC3000器件,01111对 XC4000的器件),在最后器件的程序数据和尾位之后,放置整个位流的填充位,根据在菊花链中器件的总数加上与字节边界有关的起动序列的控制来计算位流的填充位。

§ 9.2.4 PROM

一般,在通过仿真校验设计和实现实验板之后,可以制造最后的生产板,许多板利用 PROM 来配置 FPGA,串行 PROM 是最常用的。一旦位流已经产生,可以利用 MakePROM

程序产生必要的 PROM 文件, MakePROM 程序支持三种格式: MCS-86, EXORMAX, TEKHEX。在 XC17XXD 器件编程时, 必须设置 PROM 的 RESET/OE 的极性, 设置在器件中的极性位可以完成此任务。对 $\overline{RESET}/OE$ 为 0000 或 RESET/ $\overline{OE}$ 为 FFFF。

§ 9.3 配置模式

FPGA 以几种模式之一接收配置数据, 逻辑电平加于模式选择腿 M0、M1 和 M2, 在开始配置时确定利用的方法, 这些模式中的某些利用并行数据, 而其它利用串行数据, 某些模式是有源的, 即器件提供它自己的时钟, 其它是无源的, 即器件接收外部产生的时序。

§ 9.3.1 主模式与非主模式比较

有两种不同的 FPGA 配置方法, 主模式与非主模式。

在主模式, FPGA 从外部存储器如并行或串行 PROM 自己加载, FPGA 提供时钟和/或地址到 PROM, 并接收数据加载到它的内部配置存储器, 有三种不同主配置模式: 向上并行主模式, 向下并行主模式和串行主模式。

在非主模式中, FPGA 由另一器件加载, 如微处理器或另一 FPGA, 由它提供时钟和必须加载到非主模式的内部配置存储器的数据。

对于 XC2000 和 XC3000 系列有两种非主配置模式: 外设模式和串行从模式, 对 XC4000 系列器件支持三种非主配置模式: 异步外设模式, 同步外设模式, 串行从模式。注意要把从属器件的 $\overline{INIT}$ 腿线与起来, 把信号接到加载器件的 $\overline{INIT}$ 腿, 以便把配置推迟到所有的器件已清除它们的配置存储器才进行配置。 $\overline{INIT}$ 对 XC4000 的器件是输入/输出腿, XC4000 系列要求一个附加的时钟在加载配置数据之后完成起动; 所以它应该在混合器件系列的菊花链中用作主模式。

§ 9.3.2 XC2000/XC3000 配置模式

此节介绍 XC2000/XC3000 的配置模式: 向上/向下并行主模式, 串行主模式, 外设模式, 表 9.2 列出如何调整模式腿 M0、M1 和 M2 来设置器件的配置模式。

表 9.2 XC2000/XC3000 配置模式

M0	M1	M2	CCLK	模式	数据
0	0	0	输出	主模式	位串
0	0	1	输出	主模式	字节, 0000 上升
0	1	0		保留	
0	1	1	输出	主模式	字节, FFFF 下降
1	0	0		保留	
1	0	1	输出	外设模式	字节 (XC2000 为位串)
1	1	0		保留	
1	1	1	输出	从模式	位串

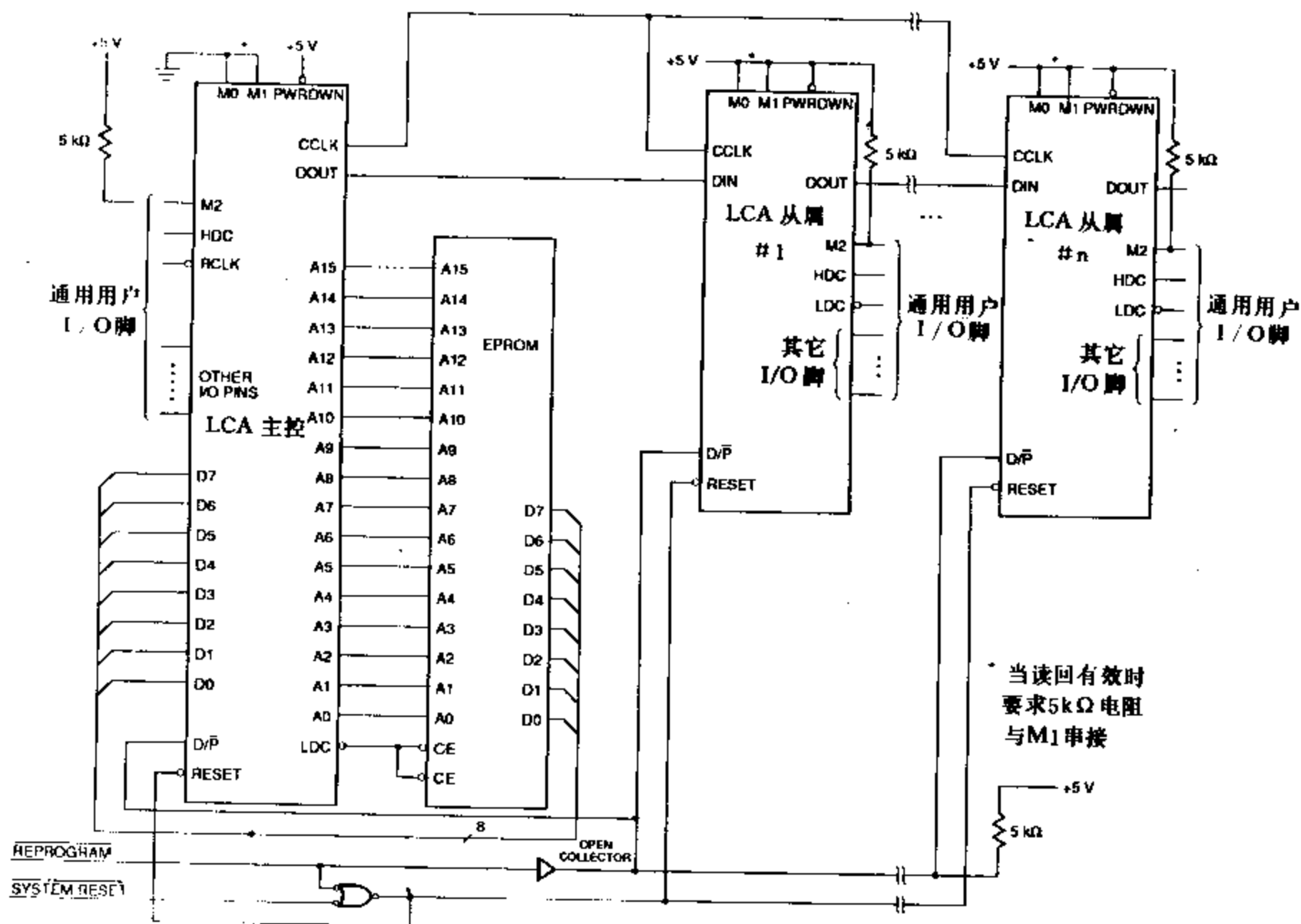


图9.1 XC2000器件的并行主模式

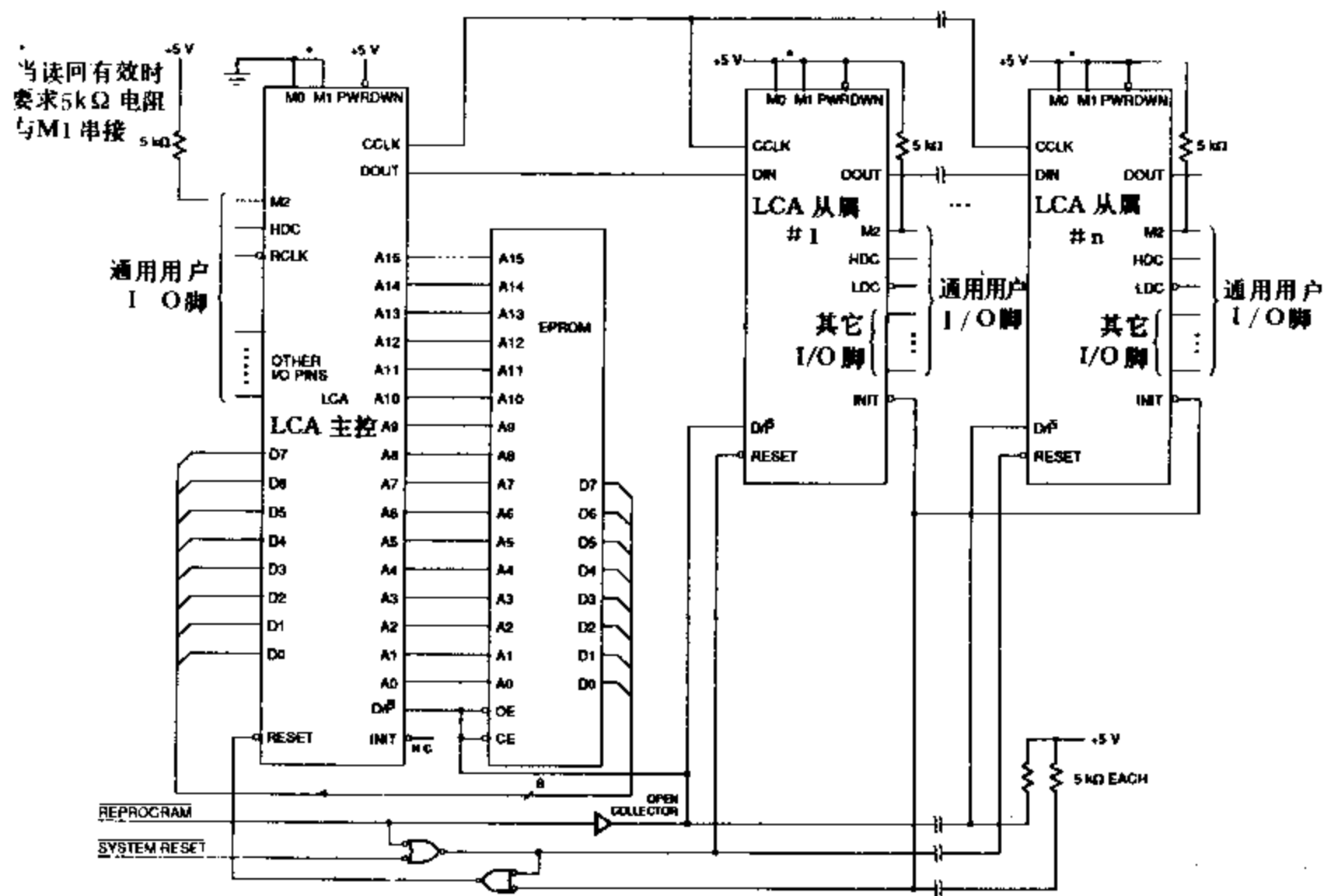


图9.2 XC3000/XC3100器件的并行主模式

1. 并行主模式(向上/向下)

在并行主模式中,器件直接寻址一个工业标准的字节型 EPROM,并在16位地址计数器加或减计数之前接收八个数据位。图9.1说明带有菊花链从模式器件的 XC2000系列器件的并行主模式。图9.2说明对 XC3000/XC3100器件的并行主模式。

八个数据位在器件中被串行化,并产生由主模式器件及并行或菊花链方式连接的任何从属器件内部使用的 CCLK。器件送出串行化的40位数据起始码到 DOUT 腿上,然后在接收它的配置数据信息段的同时加一高电平。有一个1.5CCLK 周期的内部延迟,在 CCLK 的上升沿接收一个字节的的数据,并改变 EPROM 的地址,直到 CCLK 的下降沿使此字节的 LSC(DO)出现在 DOUT,这意味着 DOUT 在 CCLK 的下降沿改变,在菊花链的下一个器件在其后的时钟上升沿接收数据。

2. 串行主模式

在此模式时,CCLK 的输出驱动一个串行 PROM,它馈电到 FPGA 的 DIN 输入,CCLK 输出的每个上升沿加计数串行 PROM 的内部地址计数器。CCLK 上的这个正的转换把下一数据位放到接在器件 DIN 腿的 PROM 数据输出端上,器件在其后的 CCLK 上升沿接收这个数据。图9.3和图9.4说明 XC2000/XC3000器件的串行主模式。

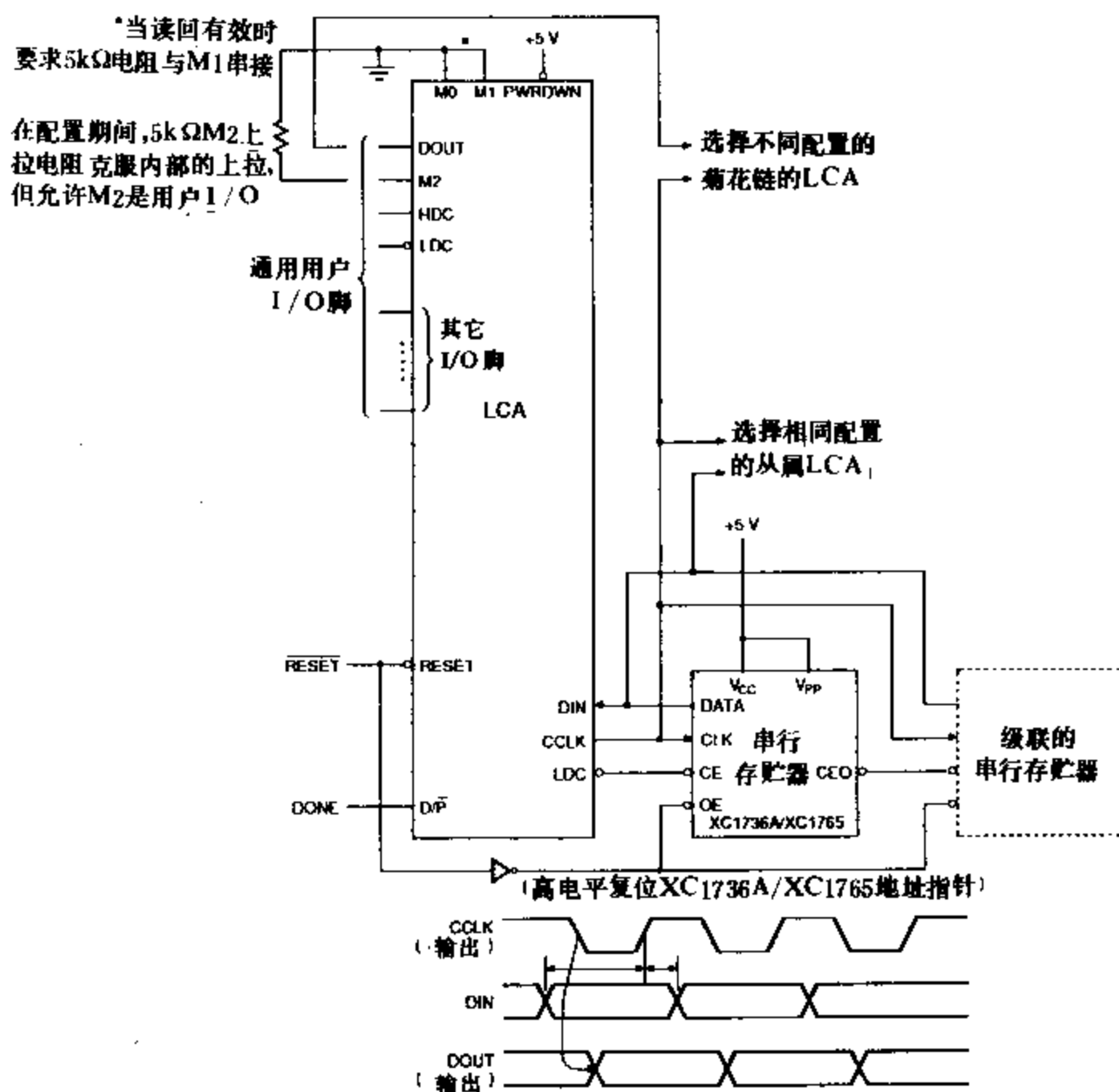


图9.3 XC2000器件的串行主模式

串行主模式器件送出40位的起始码和给出它的配置信息段的所有数据到 DOUT 腿上, 当在并行主模式时, 从 DIN 到 DOUT 有 $1.5CCLK$ 的内部延迟。串行 PROM 片选 (CE) 输入可以由 $\overline{LDC}$ 或 DONE 来驱动, 利用 $\overline{LDC}$ 避免可能的到 DIN 腿的连接, 如果这个腿配置作为用户的 I/O 腿, 但是 $\overline{LDC}$ 则被限制是固定高电平用户输出, 利用 DONE 也避免接到 DIN 上, 只要较早选择 DONW 的选项。

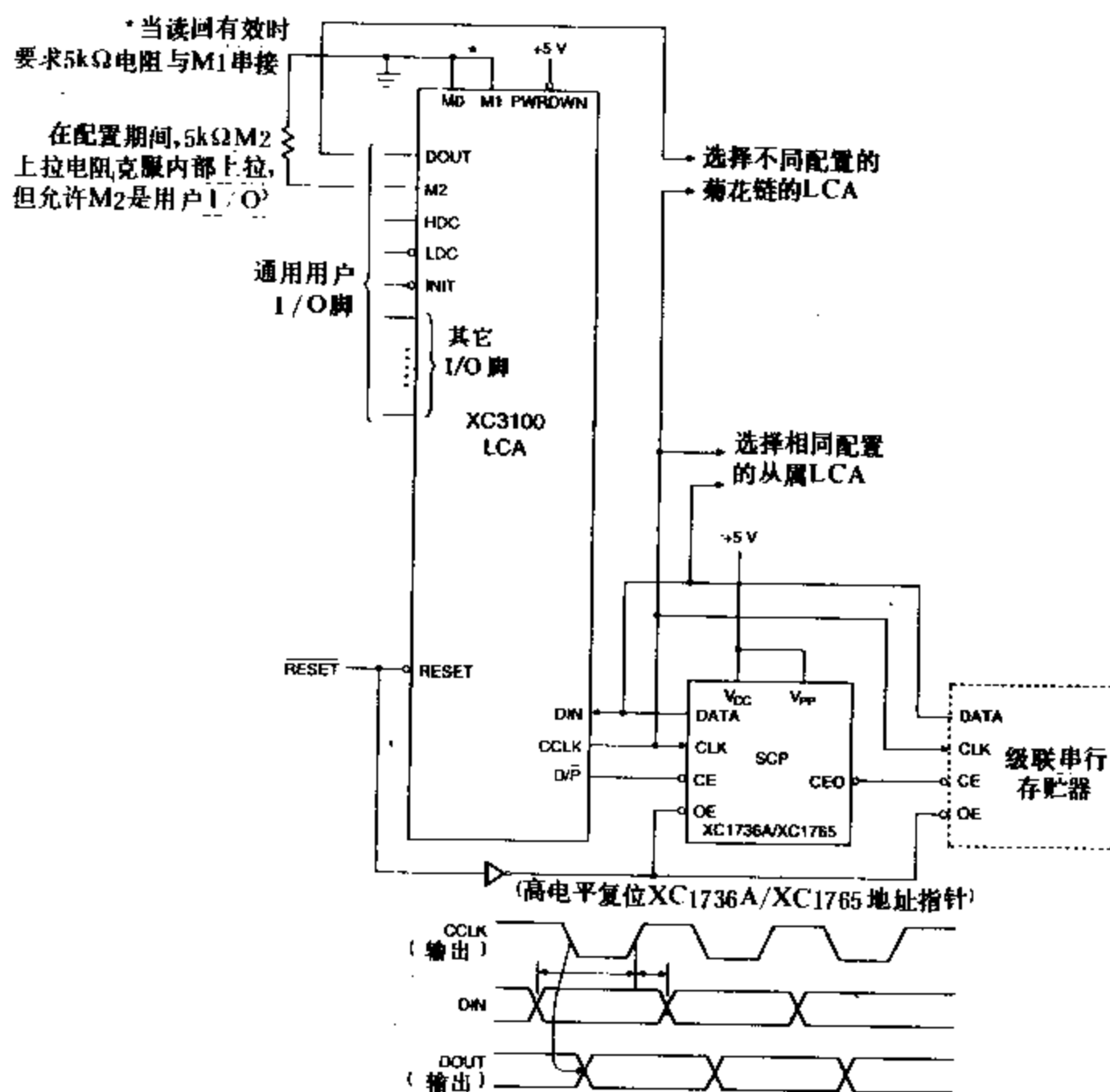


图9.4 XC3000/XC3100器件的串行主模式。

3. 外设模式

外设模式提供一个作为微处理器外设加载 FPGA 的接口, 图9.5表示 XC2000器件的串行数据外设模式的连接。图9.6表示 XC3000/XC3100器件的外设模式。

处理器的 Write 周期译码来自低有效写闸门 ($\overline{WS}$), 两个低有效和一个高有效的片选信号 ($\overline{CS0}$, $\overline{CS1}$, $CS2$) 的共同的要求, 如果所有这些信号不可用, 未用的输入必须驱动到它们的有效电平。XC2000的器件在每个被选的写周期接收一个配置数据的串行位到 DIN 腿, XC3000器件在每个被选的写周期接收一个并行的配置字节到 D0—07的输入。

XC3000配置数据的每个字节加载进一个寄存器, XC3000器件从内部时序发生器产生

一个配置时钟,并对内部的信息段把并行输入数据串行化,并送到菊花链从属器件的 DOUT 腿上。当 INIT 变高, RDY/ $\overline{BUSY}$ 信号表示器件的状态,先于 INIT 腿变高, RDY/ $\overline{BUSY}$ 信号通过上拉电阻变为三态,它给出器件准备接收数据的虚假指示, XC3000 的 RDY/ $\overline{BUSY}$ 输出腿上高电平表示当输入寄存器准备好时前一字节加载已完成。

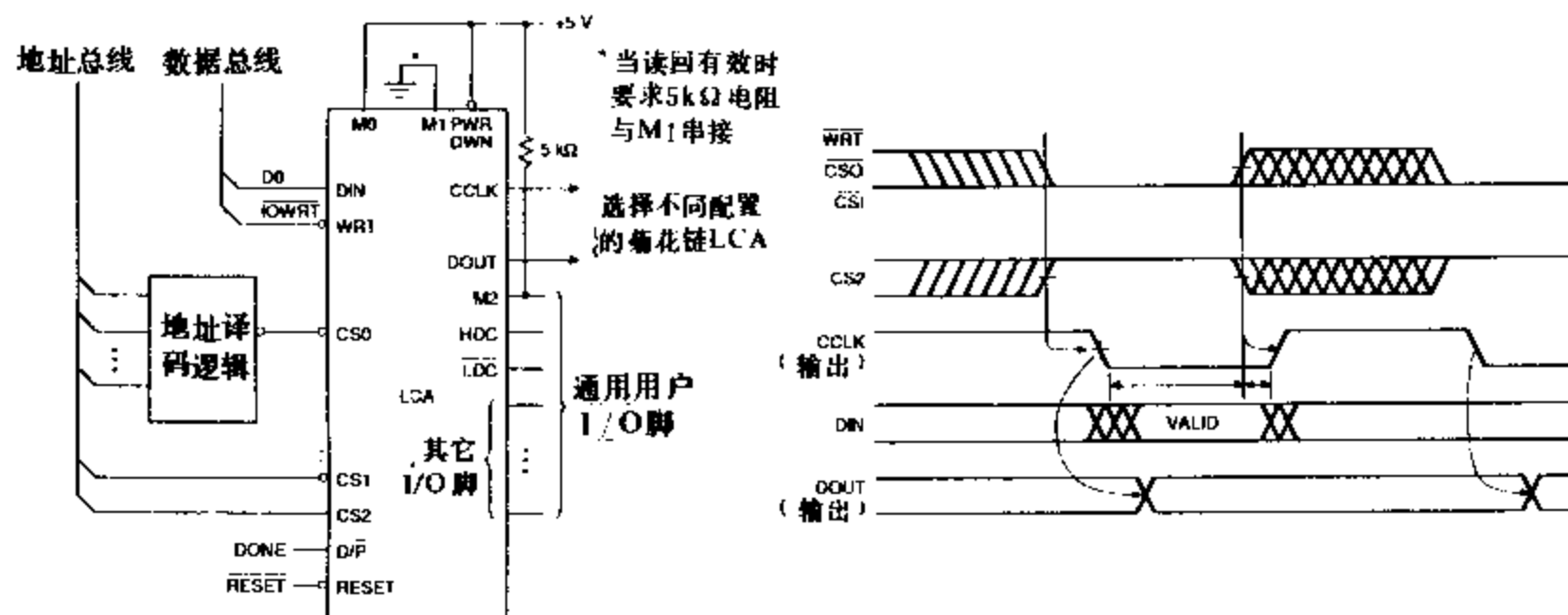


图9.5 XC2000器件的外设模式

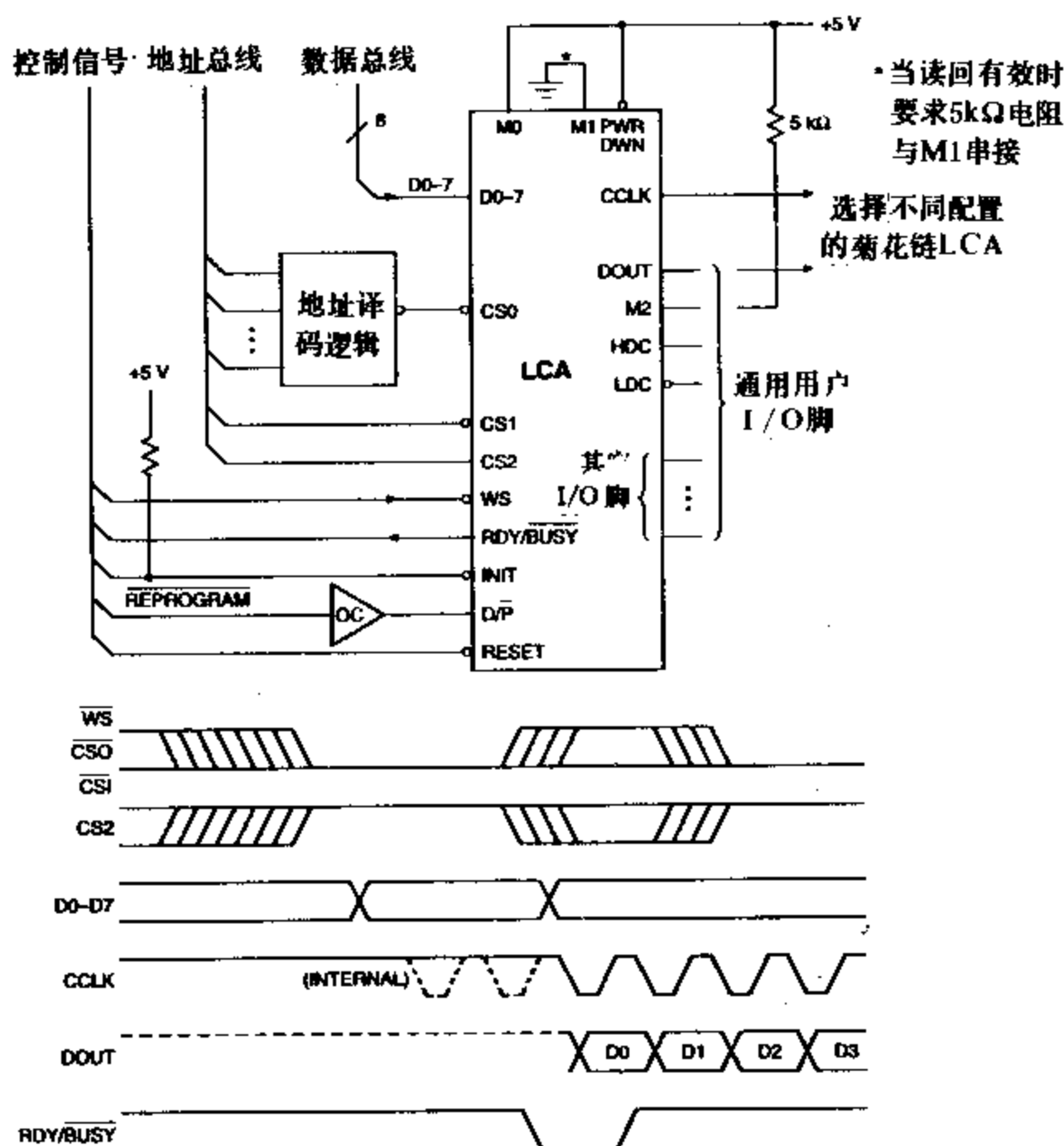


图9.6 XC3000/XC3100器件的外设模式

4. 从属模式

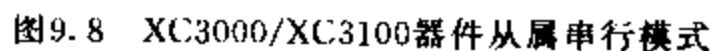
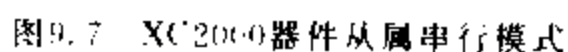


图9.7和图9.8说明 XC2000和 XC3000/XC3100器件的从属模式。这个模式为加载配置的数据提供简单的接口,这个模式利用串行数据和一个外部提供的同步配置时钟(CCLK),在 CCLK 上升沿读串行位的配置数据,在 CCLK 的下降沿提供 DOUT 上数据,CCLK 不必保持低电平多于 $5\mu\text{s}$ (在军用 B 档器件下为 $1\mu\text{s}$),多个从属模式的器件当它们利用相同的配置时可以并行地连接。大多数从属模式的应用是菊花链配置,其中数据的输入是由前面器件的数据输出来支持的,而公共的时钟是由在主模式或外设模式中领先的器件提供的,微处理器或其它专门的电路亦可以供给数据和时钟,XC2000器件要求一些从属数据的保持时间,而 XC3000的器件不要求。

§ 9.3.3 XC4000配置模式

XC4000系列有六个配置模式,用加于 MO、M1和 M2输入端的三位输入码来选择,有三个自动加载的主模式,两个外设模式和主要用于菊花链器件的串行从模式,如表9.3所示,配置期间,某些 I/O 腿被暂时用于配置过程。

1. 并行主模式

主模式利用一个内部振荡器产生 CCLK 以驱动可能的从属器件,并为包含配置数据的外部 PROM 产生地址和时序。(向上或向下)并行主模式利用来自 PROM 的字节宽的数据,响应下由 FPGA 配置逻辑产生的18位地址把数据加到 D0—D7腿上。图9.9说明了 XC4000器件的并行主模式

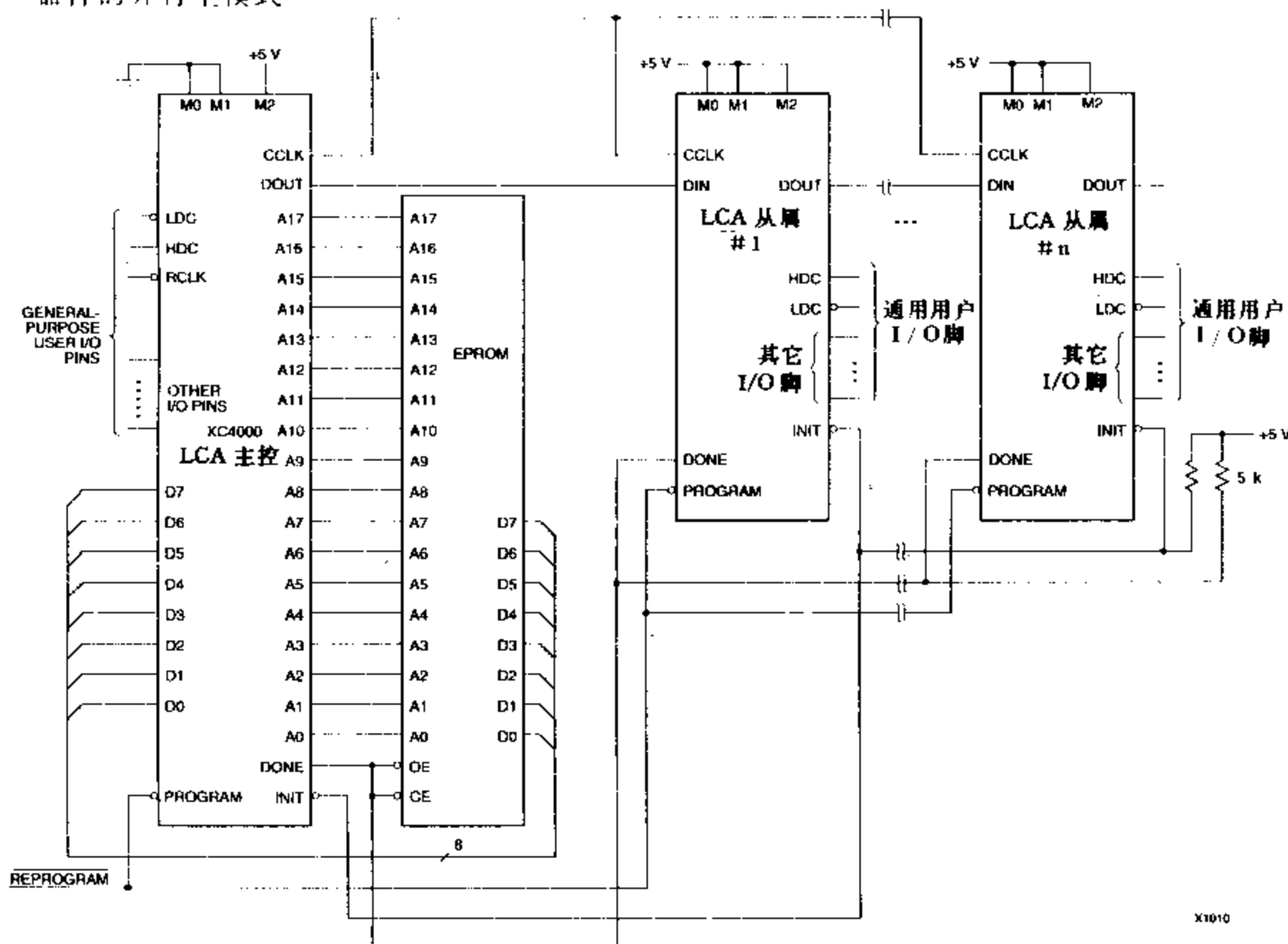


图9.9 XC4000器件并行主模式

(向上或向下)并行主模式产生 CCLK 信号和 PROM 地址,并接收并行的字节数据,向上和向下选择产生或为零或为3FFFF 的起始地址,以便和不同的微处理器寻址变换相兼容。对于(向上或向下)并行主模式,八位字节数据在每个读时钟(RCLK)被读出,在内部由配置时钟(CCLK)从最低有效位开始串行化。

对于在菊花链中的多个器件,主模式器件和并行 PROM 相接,并传递数据到其它的串行器件,从属器件利用领先器件产生的 CCLK,通过 DIN 移入数据。如果多个从属串行器件有相同的配置,它们的 DIN 腿可以并行连接,主模式器件由于没有 CCLK 同步而不可以并行连接。

2. 串行主模式

这个模式利用来自同步串行源,如串行 PROM,加到 DIN 腿的串行配置数据,如图9.10所示。器件自动从串行 PROM 在加电或重新编程时对自身加载。一个内部振荡器产生 CCLK,用于对数据定时和驱动从属串行器件。

3. 外设模式

两个外设模接收来自总线的字节宽的数据,READY/BUSY 状态,作为握手信号是有效的,在非同步模式内部振荡器产生 CCLK 脉冲信号,使字节宽度的数据串行化,在同步模式,外部提供的时钟信号送到 CCLK 使数据串行化。

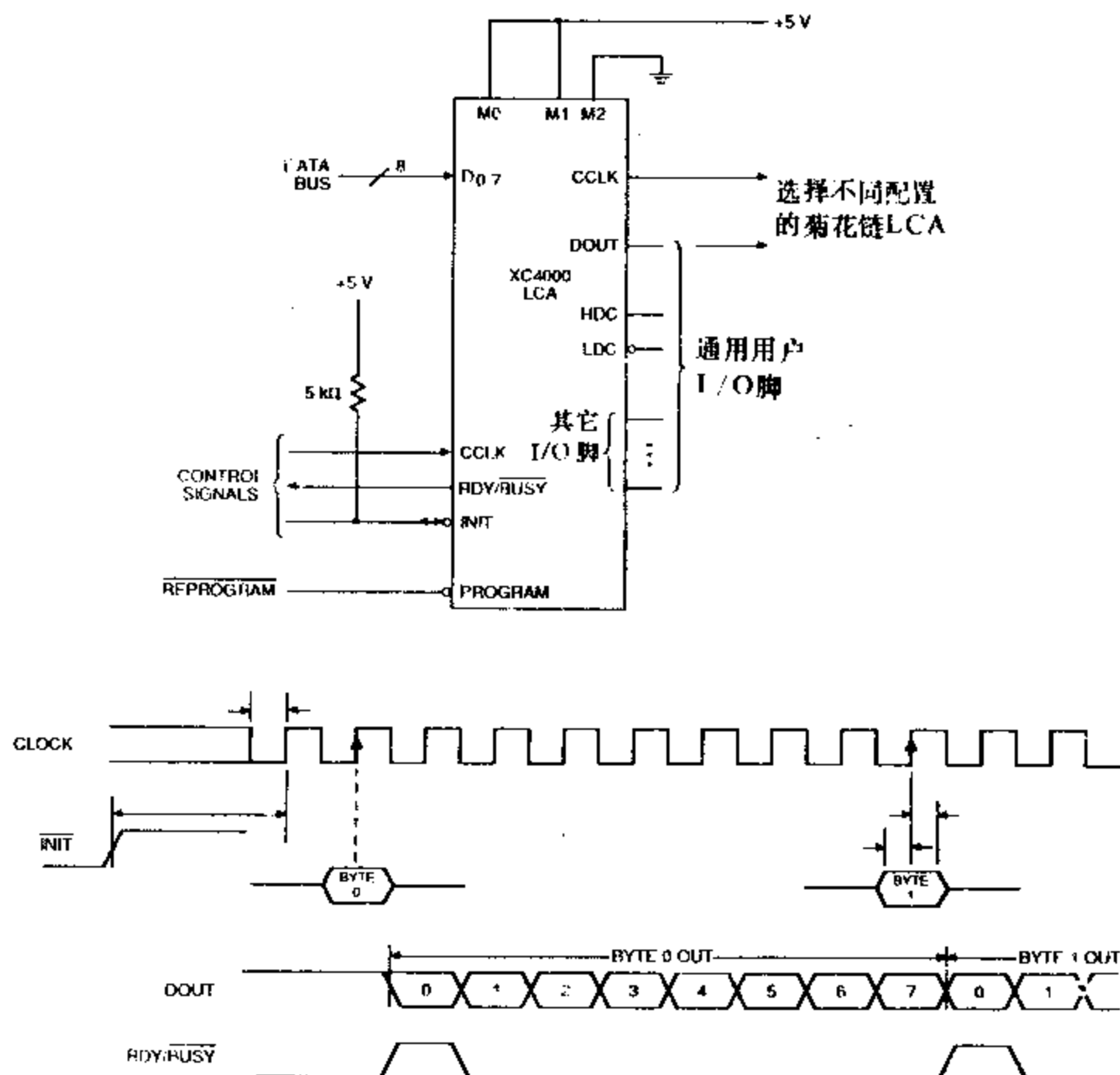


图9.11 XC4000器件同步外设模式

所以,功能上同步外设模式看起来象并行从模式,如图9.11所示。接收来自处理器总线或DMA 控制器字节型数据,在 $\overline{INIT}$ 变高之后,第二个用户产生的 CCLK 定时在数据的第一个字节 D0—D7上。其后的七个时钟 CCLK 内部地把数据串行化成配置的信息段。在数据的每个字节加载之后, $RDY/\overline{BUSY}$ 信号变高一个时钟周期,发出器件准备接收新字节的信号。数据的下一个字节可以在 $RDY/\overline{BUSY}$ 脉冲之后第一和第七个 CCLK 之间的任何时间放到 D0—D7数据总线上。

异步外设模式中,器件看起来象处理器总线上一个外设。通过字节型数据输入腿 D0—D7加载数据,在对器件写或读状态标志发生之前,芯片必须被选上, $CS1=1, \overline{CS0}=0$ 。

在写周期内, $\overline{WS}=0, \overline{CS0}=0, CS1=1$ 和 $\overline{RS}=1$ 的组合把数据的字节加载进并到串的移位寄存器的数据锁存器以加载成内部配置的信息段。当写的条件变为非时,它锁存 D0—D7腿上的最后数据到数据锁存器。在 $RDY/\overline{BUSY}$ 输出腿上的高电平表示数据锁存器是空的,另一个写周期可以开始。在某些保持时间之后的每个写周期后,数据的下一个字节可以存在于 D0—D7腿上。在读周期内, $\overline{RS}=0, \overline{CS0}=0, CS1=1$ 和 $\overline{WS}=1, RDY/\overline{BUSY}$ 的状态标志对 D7是有效的,以使可以从写上的相同总线来读。

注意:在外设模式,用片选腿作为用户的输出可以在配置的最后周期因用户 I/O 变为有效导致信号竞争。

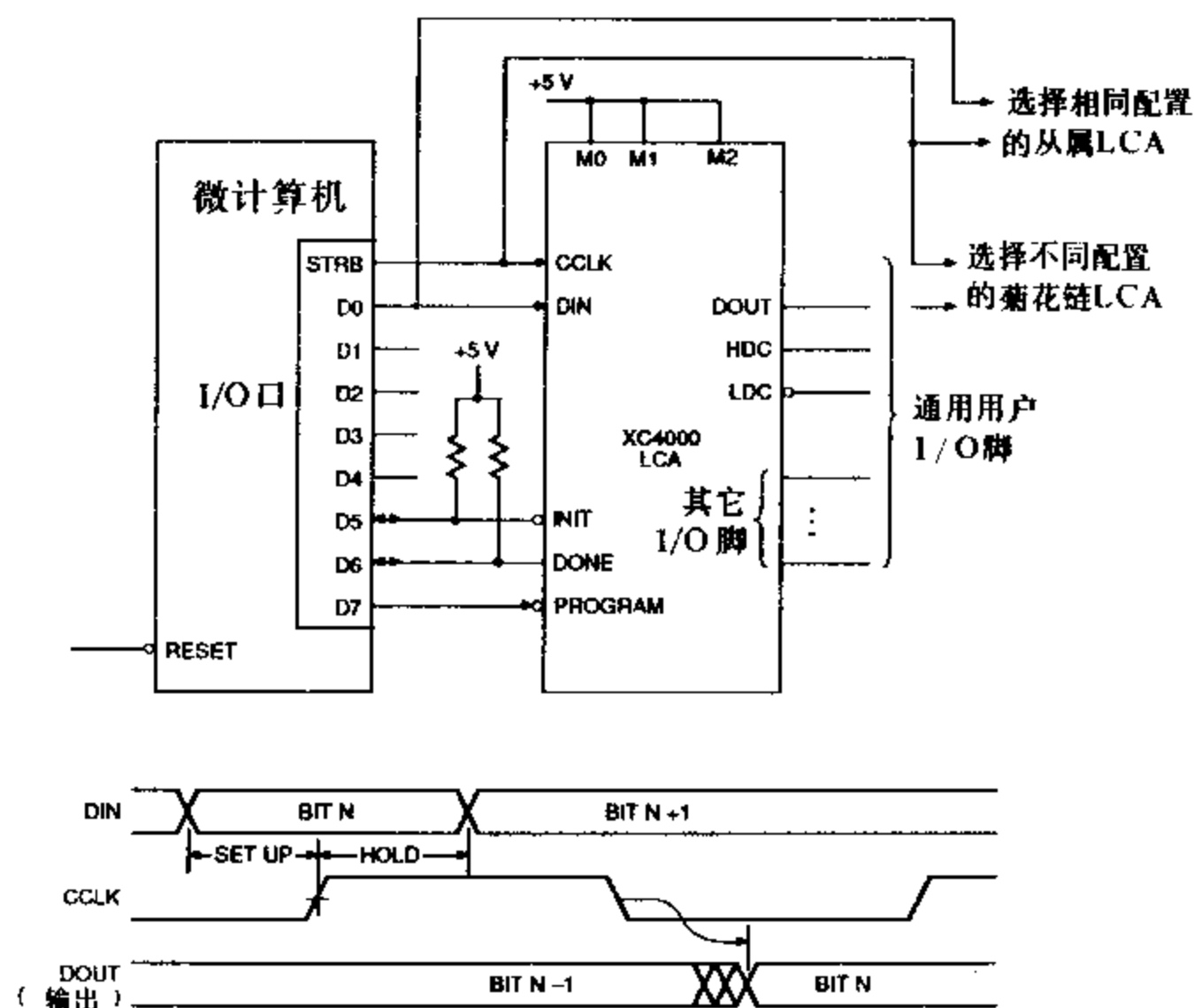


图9.12 XC4000器件串行从属模式

4. 串行从模式

串行从模式的器件用一个外部提供的配置时钟接收串行数据。LAC 器件在 CCLK 的上升沿接收串行配置数据,在加载它的配置之后,传递附加的数据输送出去,在 CCLK 的下一个下降沿重新同步,具有相同配置的多个从属器件可以用并行地连接 DIN 输入以使器件可以同时配置。大多数串行从模式应用是菊花链配置,其中由前面的器件 DOUT 提供数据,而公共时钟是由链头部的主模式或外设模式器件提供。处理器或其它逻辑也可以提供数据。

XC2000 器件要求串行从模式数据保持时间,而 XC3000 和 XC4000 器件不需要,图 9.12 表示 XC4000 器件的串行从模式。

§ 9.4 配置数据的加载和转成信息段

在加电时或重新编程时,器件清除它的存储器,如由 INIT 指示的那样,在器件清除它的存储器之后,配置数据从一个外部存储源,如 EPROM, RAM, 微处理或文件加载到 FPGA,对于串行配置模式, PROM 数据的每个字节被加载进 FPGA,最低有效位在先,对于并行配置模式,数据由此器件内部串行化以产生相同的配置结果。

图 9.13 表示在器件中时钟和数据通道的选择,这些决定配置时钟源,为 DOUT 和为内部配置存储器的信息段寄存器提供数据的串行化。

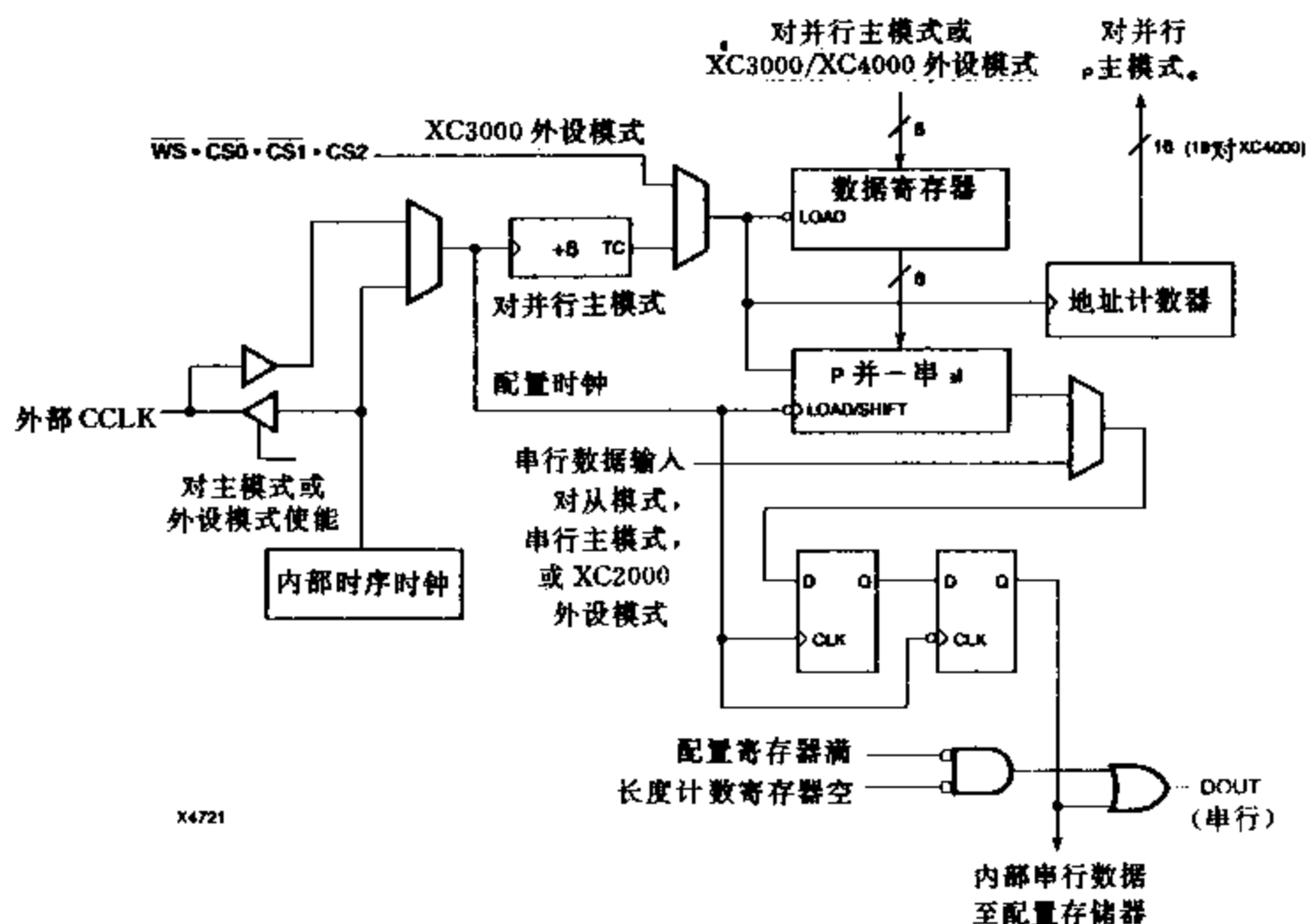


图 9.13 模式线逻辑电平输入到 FPGA 控制配置数据通道多路开关

预兆码/长度计数的串行数据和附加器件的配置数据是由 CCLK 的负沿定时,并为菊花链从属模式器件提供到 DOUT。

运行的长度计数器控制允许一个系统的多个不同规模的 FPGA 以非同步型式开始运行,如图 9.14 所示,信息段寄存器累加存储器数据的信息段。

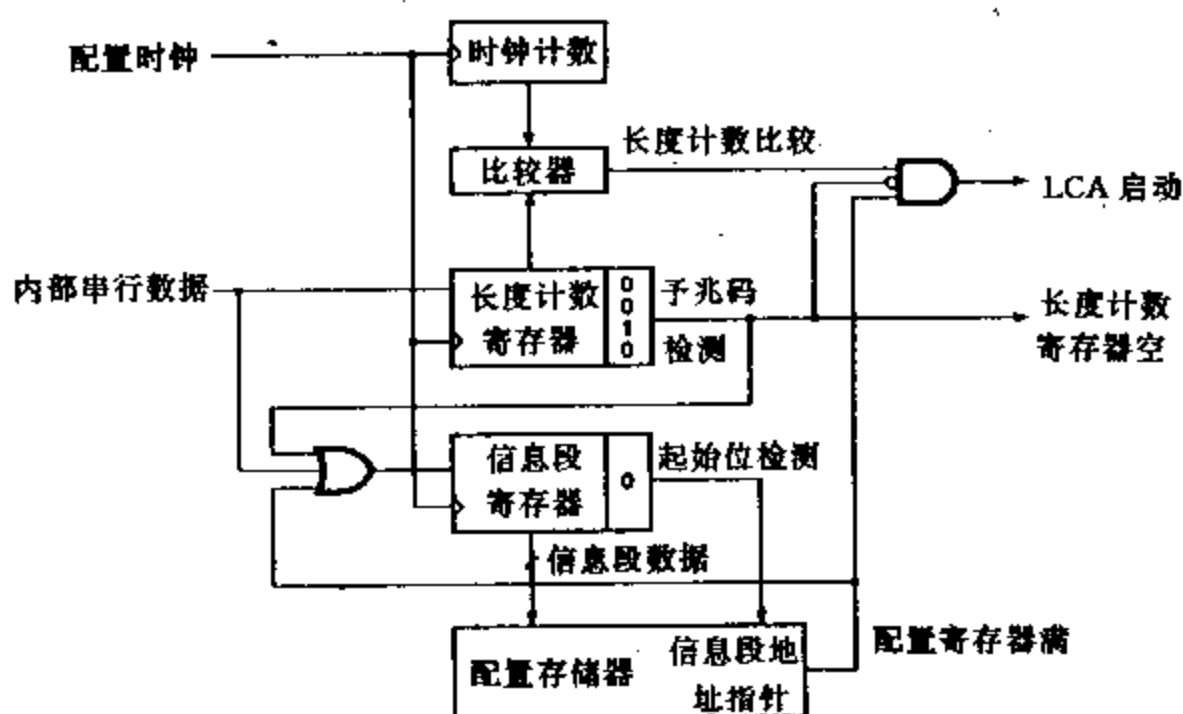


图9.14 信息段寄存器

用 MakeBits 程序产生的配置程序以预兆码111111110010开始,跟随一个24位长度计数代表为实现配置程序加载需要的配置时钟总数,在预兆码和长度计数已被加载之后,串行数据移进内部信息段寄存器。

领先的器件总是传递预兆码,长度计数和4位分隔字段到附加器件的 DOUT 腿,如果仅仅有一个器件在系统中,DOUT 腿仍反映此40位的起始码。跟随长度计数,器件保持高的 DOUT 信号直到它截获为填满它的配置存储器要求的信息段数,在此时器件开始直接通过它的 DOUT 腿传递数据。

这个流程允许菊花链器件接收公共的长度计数,并且禁止为后面器件的信息段数据起始位,直到前面器件的存储器被填满。

当内部时钟计数器匹配加载开始时的长度计数值,当它的程序存储器被填满时,器件开始启动,当此值不匹配时,则器件连续地从它的 DIN 腿移到它的 DOUT 腿直到它们完成。

§ 9.4.1 菊花链的串行加载

如果在设计中利用了几个器件,它们可以以菊花链被配置,在链中的第一个器件可按任意模式加载,称为领先器件,其余的器件按从属模式被加载。

对于菊花链配置,XACT 开发系统为所选的 FPGA 设计产生合成的配置位流,位流包括一个预兆码和总的位流的长度计数,跟随由器件尾部分开的一个或多个链数据组,具有一个24位长度寄存器,FPGA 可以寄存超过一千六百万的配置位或在单个菊花链中约700个 XC3030。

在加载和传递预兆码和长度计数到可能的菊花链之后,领先的器件加载它的配置数据信息段,而提供一个高的 DOUT 到可能的后面的器件。

当领先的器件已接收它的配置数据和当现存的时钟计数还没有达到全长计数值,附加数据被移出,以串行形式出现在 DOUT 腿,领先的器件也产生配置时钟同步后面器件的数据入和数据出。

从属器件利用 CCLK 的正沿读入数据, CCLK 的负沿移出数据。并行主模式是器件利用它的内部定时发生器产生比 EPROM 地址速率高八倍的连续 CCLK, XC3000 和 XC4000 外设模式器件为每个片选和写闸门周期产生八个 CCLK 的脉冲, 在 XC2000 器件每次写产生一个单个 CCLK。内部定时发生器包含对通用定时和所有模式中输入同步的操作。

§ 9.4.2 多个器件的并行加载

并行地配置几个 FPAG 是可能的, 当要用相同的设计加载几个器件时这是特别有用的, 有两个并行加载的方法, 仅仅在配置时钟的来源上有如下差别。

方法1: 领先器件提供 CCLK——以下步骤可执行方法1

1. 配置串行主模式的一个器件和串行从模式的其它器件;
2. 领先器件的 CCLK 连接到串行 PROM 的 CLK 腿和所有从属器件的 CCLK 腿
3. 串行的 PROM 的 DATA 腿接到所有器件的 DIN 腿。
4. 线与所有器件的 $\overline{DONE}/\overline{PROG}$ (XC4000 的 $\overline{DONE}$) 腿, 用来使能串行 PROM。

主模式器件提供时钟到串行 PROM, 它送出数据到主模式器件和所有从模式的器件, 主模式器件也定时进入从模式器件的数据, 当所有的 $\overline{DONE}/\overline{PROG}$ 和 $\overline{DONE}$ 腿变高时, PROM 被禁止。

方法2: 外部的 CCLK 来源——以下步骤执行方法2

1. 以串行从模式配置所有的器件
2. 把所有的 CCLK 腿接在一起, 由提供配置时钟的器件来驱动它们。
3. 把 DIN 腿接在一起, 把它们接到送出为数据的器件, $\overline{DONE}/\overline{PROG}$ 和 $\overline{DONE}$ 腿可以线与在一起产生一个信号监测所有器件已被配置的时间。
4. 当加载这些器件时, 利用 $\overline{INIT}$ 腿的线与来确认已允许它们有足够的时间来完成送数据前的清除状态。
5. 仔细地不要破坏任何 CCLK 的定时规定。

注意: 由于动态的电路, XC2000 和 XC3000 器件的 CCLK 低电平时间有 $5\mu\text{S}$ 最大的极限, 因为在 XC4000 部件中没有动态配置电路, 对配置的 CCLK 低电平时间没有限制。

§ 9.4.3 加载交替配置

用高位的 PROM 地址的外部控制或选 M1 的增或减地址可以完成交替配置的选择, 对主模式高或低, 八位数据字节被读入, 由内部的配置时钟从低有效位开始串行化, 一个主模式器件可以用于与 PROM 的连接, 传递附加的链配置数据到以菊花链方式的附加器件中去。

当所有的并行配置 PROM 的数据已经读出, 数据输出腿由来自 LDC 的高电平或一个较早的 $\overline{DONE}$ 去使能, 为从属器件产生配置时钟输出, 它利用由前面器件 DOUT 腿提供的串行化数据输出, 如果多个从属串行模式器件有相同的配置, 它们的 DIN 腿可以并行地连接。

用串行主模式配置, 附加的相同器件可以同时被配置, 每个是串行从模式。多个主模式器件不可以并行连接, 因为它们的时钟不同步。

§ 9.5 XC2000和XC3000器件的状态

在配置期间,FPGA 经历几个状态,在这些状态期间,FPGA 执行加电时间延迟,清除配置存储器,配置程序等几件事情,当器件已经加载配置位流的最后一个信息段,它进入起动,变成可操作的,当适当的信号被加入后,器件亦可以返回到重新配置状态。

对 XC2000和 XC3000系列配置过程的状态图已在第四章的图4.9表示。

§ 9.5.1 加电和初始化

当电源加上电,在 V_{cc} 达到器件开始运行的电压(近似2.5到3.0V)时触发内部加电复位电路。

注意:加电时, V_{cc} 必须在少于25msec 内从2.0V 上升到 V_{cc} 最小,如果不可能时,配置可以用保持 $\overline{RESET}$ 为低直到 V_{cc} 已达到4.0V(对 XC2000L 和 XC3000L 的器件为2.5V)来延迟。

大于100ms 的十分长的 V_{cc} 上升时间,或非单调的上升 V_{cc} 可能要求在 $\overline{RESET}$ 上的高电平持续大于6 μ S,在 V_{cc} 已达到4.0V 后,用一个最小6 μ S 的脉冲持续在 $\overline{RESET}$ 和 $\overline{DONE}/\overline{PROG}$ 上。

一旦稳定的电源已建立,用户可编程 I/O 输出缓冲器在配置中不用而被禁止,为 I/O 封装腿提供高阻的上拉电阻。配置的握手输出信号,如 HDC,LDC,DONE, $\overline{INIT}$ (仅 XC3000) 等等,设定它们的配置状态逻辑电平,内部超时延迟起动,以使电源供电电压可以稳定,非主模式器件的初始状态超时定时(11到33ms)由14位计数器来确定。

即使一个慢的菊花链串行从属器件必须允许在最快的主模式器件之前完成初始化,主模式器件扩展它们的初始化状态到非主模式延迟(43—130ms)的四位。

超时计数器由1MHz 的内部定时器定时,它随处理温度和供电电压从0.5—1.5MHz 可变,内部定时器也利用来同步输入和提供主模式中的 CCLK。

外部 $\overline{RESET}$ 腿当 V_{cc} 电源在不到25ms 内没有从2.0V 上升到 V_{cc} 最小时必须保持低,直到初始化和清除完成,在 XC3000 器件中 $\overline{PWRDWN}$ 腿被禁止,在 XC3000A 和 XC3000L 器件中在初始化和清除完成后,可以保持 $\overline{PWRDWN}$,但是,对于一个 XC2000, XC3000和 XC3100器件不应该保持 $\overline{PWRDWN}$ 直到配置完成之后。

§ 9.5.2 清除

在初始化后,FPGA 进入清除状态,它清除配置存储器,这个过程化费每个配置数据信息段的一个内部时钟周期,在清除状态之后,器件在它使模式线最后采样之前测试外部低有效的 $\overline{RESET}$ 不存在,则进入配置状态。当 $\overline{RESET}$ 是有效的,器件在进入配置状态之前等待,对于从属或外设的 XC2000器件,外部时间延迟必须保证在配置开始之前完成初始化和清除状态。

对 XC3000器件,在低有效漏开路的初始化信号 $\overline{INIT}$ 腿上高电平表示初始化和清除已完成,多个 XC3000的 $\overline{INIT}$ 腿的外部线与可以用来表示所有器件的清除已完成,这个线与信号可以用来控制配置开始和断定一个领先的主模式器件的低有效 $\overline{RESET}$ 而重新配置,或通知一个处理器所有连接的 XC3000器件已初始化来重新配置。

§ 9.5.3 配置

在进入配置周期之前,整个配置存储器被清除,在配置程序加载期间,组合的输入开始传播,因为器件变成配置状态,器件的触发器和锁存器保持复位,输出缓冲器保持高阻。配置期高(HDC)和配置期低($\overline{LDC}$)是两个用户的 I/O 腿,在器件的初始化,清除或配置状态时,它们定义有效的输出电平。HDC, $\overline{LDC}$ 和 $\overline{DONE}/\overline{PPOG}$ 提供外部逻辑函数控制的信号,如复位,总使能或配置期间的 PROM 使能,这些允许其它的配置腿和用户逻辑函数共享。

与配置模式有关,其它用户 I/O 腿有配置期间的暂时功能,包括地址腿,数据腿,片选/写闸门,数据入,数据出和 XC3000 的 $\overline{INIT}$ 腿。与配置功能共享的输出腿在 DONE 之前一个时钟周期变成有效,在 DONE 后编程的 XC2000 和 XC3000 的器件,有效器件输出和配置输入之间的连接必须避免,这个连接可以用隔离电阻隔离缓冲器,用来暂时禁止用户输出的外部信号,或内部 FPGA 逻辑用于匹配起始输出到外部的电平。

可以全局地编程用户输入,使其有 TTL 或 CMOS 兼容的门限,输入在加电和配置期间有 TTL 门限,门限在完成配置时变到 CMOS 电平,只要用户选择 MakeBits 选项,TTL—输入电平为接口所要求,因为它们的 2.4V 最小高电平,COMS 门限通常提供更好的系统噪声抑制。注意:TTL 门限选择仅仅影响输入,除了对 $\overline{PWRDWN}$ 之外,它是固定在 CMOS 电平的,所有 XC2000 和 XC3000 输出总是 CMOS 兼容的,即它们沿到沿转换。

当利用晶体振荡器时,它提前在配置中运行以提供在把 DONE 接到外部电路之前稳定的时间。在配置期间对 RESET 的判断在两个或三个内部时钟(2—6 μ S)后被识别,器件起始一个闸门,返回到清除状态,清除部分加载的配置存储器的字,然后器件测试无效的 RESET,在重新进入配置状态之前重新采样模式线。注意:清除超时对主模式重编程或流产没有四倍初始化状态的延迟,如果利用菊花链,外部 RESET 要求长到足以保证所有非主模式器件清除,在 XC3000 中,可以从从属器件 INIT 信号的线与达到超时延迟。

§ 9.5.4 起动

当起始的加载长度计数和内部时钟计数作比较,且配置存储器是满的时候,器件执行同步起动程序,而成为运行的器件。对 XC2000 和 XC3000 器件它包括释放 DONE 腿;激活用户的 I/O;失效内部全局复位信号,图 9.15 表示了起动的时序关系。

在长度计数和 CCLK 比较之后一个 CCLK,用户 I/O 变为有效,然后在用户 I/O 被释放后的一个时钟周期,DONE 腿被释放,内部的全局复位(RESET)失效,在同时 DONE 腿被释放,全局复位信号(RESET)失效,CCLK 变成带上拉电阻的输入。

XC3000 系列提供对 DONE 时序和失效与释放用户 I/O 有关的全局复位编程能力的灵活性,在一个 CCLK 之前或一个 CCLK 之后。在长度计数比较之后两个 CCLK 释放用户 I/O,DONE 信号的时序选择和失效内部全局复位在位流文件产生时用 MakeBits 的选项来编程。在 XC2000 和 XC3000 系列中,可以编程 $\overline{DONE}/\overline{PROG}$ 信号为漏开路的输出或为包含内部上拉电阻以适应外部的线与,利用内部上拉必须限制对任何线与少于 16 个器件以允许任何单个器件驱动组合负载为低,即使 $\overline{DONE}/\overline{PROG}$ 腿外部保持低,器件开始运行。

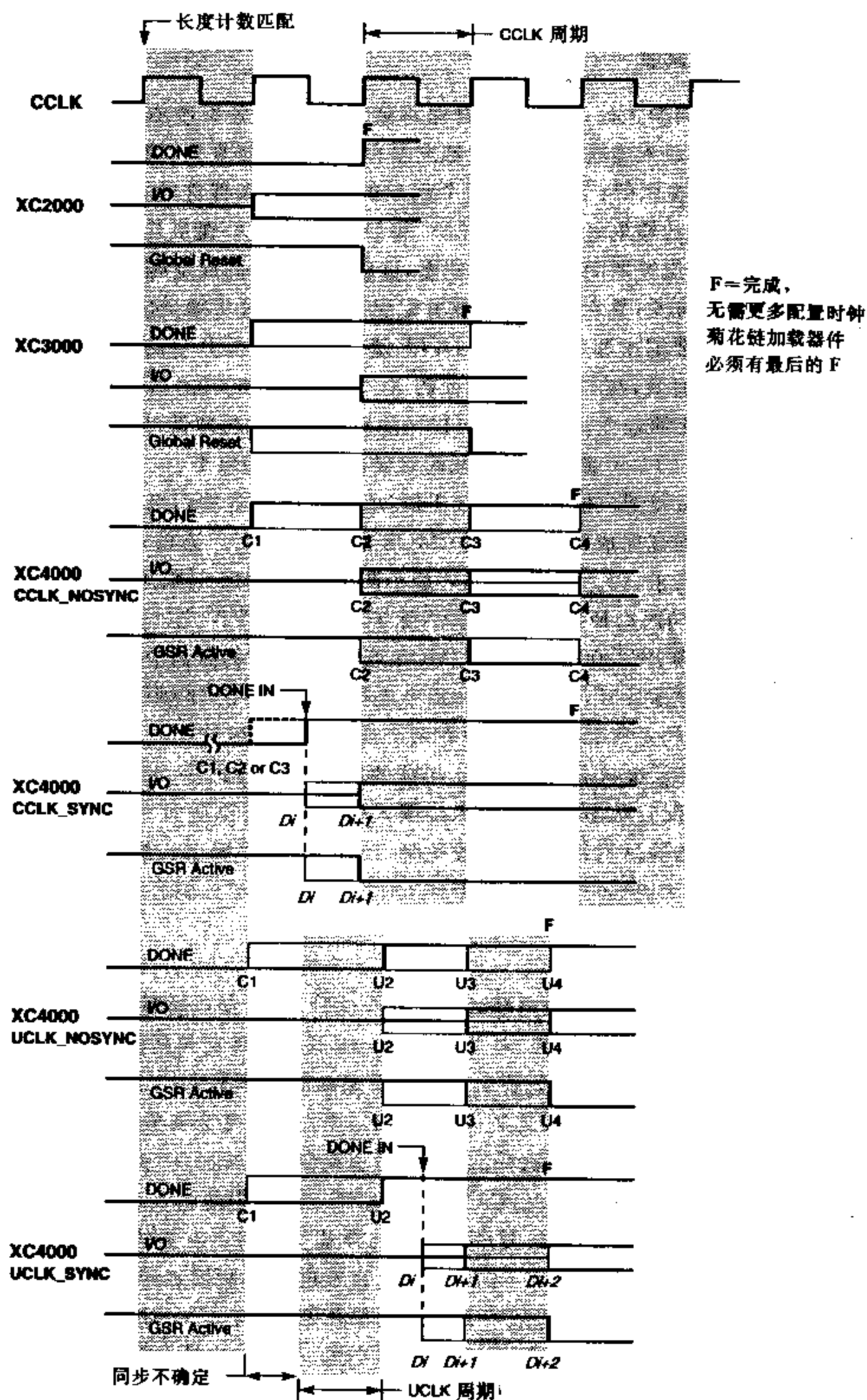


图9.15 Xilinx FPGA 的启动时序

§ 9.5.5 重新编程

器件在系统内运行的同时配置存储器可以重写,器件返回到清除(Clear)状态,此时清除配置存储器,禁止 I/O 腿,重新采样模式线。常利用外部集电极开路驱动器把 $\overline{DONE}/\overline{PROG}$ 拉低实现重新编程控制,一旦它识别一个稳定的请求,器件本身保持 $\overline{DONE}/\overline{PROG}$ 为低,直到新的配置已完成,即使 $\overline{DONE}/\overline{PROG}$ 腿超出配置周期外部仍保持为低,器件在完成配置时开始运行。为了减少噪声的影响,这些重新编程信号在内部时序发生器的2—3周期内(2—6 μ s)被滤波,注意到主模式重新编程或异常终止的清除超时没有初始化状态四倍的延时,当利用菊花链时,外部 $\overline{RESET}$ 必须足够长保证清除所有非主模式器件。对于 XC2000 系列,需要400 μ s 外部时间延时,而 XC3000 器件可达到800 μ s 的外部时间延时或用 $\overline{INIT}$ 腿的线与。

在某些应用中,系统供电电源可能有暂时的故障,它可以把器件的控制逻辑留在无效的状态,有两种方法从这种状态恢复:

1. V_{cc} 供电低于0.1V 并重新供电有效的 V_{cc} ;
2. 在 $\overline{RESET}$ 和 $\overline{DONE}/\overline{PROG}$ 上提供6 μ s 瞬时低电平,在 $\overline{RESET}$ 腿已为高电平6 μ s 之后跟着返回到有效的 V_{cc} 。

这样可以保证 FPGA 返回到清除状态,在不完全供电电压中断事故中也可用这些方法,但从断开到正常加电的应用中不需要用这些方法。

§ 9.6 XC4000 器件的状态

XC4000 系列的器件经历类似于 XC2000 和 XC3000 的器件的配置过程,即加电和清除存储器;初始化;配置;起动和重新编程。

§ 9.6.1 加电和清除存储器

当第一次或重新加电到 XC4000 的 FPGA 时,内部电路迫使初始化配置逻辑。在 V_{cc} 达到运行电平(接近2.5—3.0V),在电路通过一个配置位采样对的读写测试时,起始通常为85ms 的时间延时,在这个时间延时期间或只要插入 $\overline{PROGRAM}$ 输入,配置逻辑保持在配置存储器清除状态,在配置中无用的用户可编程 I/O 输出缓冲器被禁止,为 I/O 封装腿提供高阻抗上拉电阻,配置握手输出信号设定其配置状态逻辑电平; $\overline{HDC}$ = 高, $\overline{LDC}$ = 低和 $\overline{INIT}$ = 低, $\overline{CCLK}$ 腿变成高阻。利用内部振荡器,配置存储器信息段相继被初始化,配置清除每个信息段大约化费1.3 μ s,在每个完全通过信息段地址的末尾,加电超时延时电路和 $\overline{PROG}$ 腿的电平被测试,如果无一问题,逻辑起始一个附加的配置信息段的完全清除,然后释放 $\overline{INIT}$ 腿。

§ 9.6.2 初始化

在最后通过存储器清除之后,漏开路的 $\overline{INIT}$ 腿被释放,在 XC4000 系列中,外设或主模式器件识别无效的 $\overline{INIT}$ 之前,有一个40—160 μ s 的延时,为了在扩展的时间周期内脱离配置, $\overline{INIT}$ 信号可用集电极开路低来驱动,在 $\overline{INIT}$ 腿识别为高后,两个内部时钟,三个模式

线被采样确定为配置模式,相应的配置接口腿则被激活。

§ 9.6.3 配置

预兆码和长度计数配置数据传到菊花链的所有器件,24位长度计数加载到寄存器与对配置时钟(CCLK)计数的内部计数器比较,在预兆码和长度计数配置数据传到菊花链的所有器件之后,DOUT 保持高以防止信息段起始位到达任何其它的器件。

在主模式器件的第一信息段中专门配置位控制配置时钟速率,这个配置位可以在 MakeBits 选项中为 XC4000 器件设置,缺省值是较低的,约 0.75MHz,这个速率是与 XC2000 和 XC3000 兼容的,应该在混合菊花链中使用。

接近 6MHz 的快速率只可以对 XC4000 器件加速配置,XC4000 器件以 16 位 CRC 校验和的四位结束每一信息段,在每一信息段加载期间产生 16 位 CRC 校验和,数据信息段的最后四位以 16 位校验和被检验,如果在特定的信息段内发现有一个错误,器件停止加载,把漏开路的 $\overline{INIT}$ 腿拉低预示有错,如果出现信息段错误,器件必须将 $\overline{PROG}$ 腿拉低来重新配置。内部计数器对长度计数作校验,如果这些值匹配,且器件已填满它的配置存储器,它继续向起动阶段移动,否则器件继续从 DIN 腿送数据到 DOUT 腿。

§ 9.6.4 起动

当器件变成可运行时,起动状态有几个选项控制事件的顺序,图 9.15 中也说明了 XC4000 器件的起动时序。

由于来自长度计数的大量时钟要在 XC4000 中“完成”,所以不用 XC2000 和 XC3000 器件作为以 XC4000 为从属器件的主控器件。

在起动顺序中三个事件可以编程发生在不同的配置时钟沿,这个灵活性允许 XC4000 器件按用户的要求“唤醒”,按照 StartupClk 和 SynchToDone 选项如何来选择,DoneActive, GSRIinactive 和 OutputsActive 被设置于不同的缺省值。

1. DoneActive 确定漏极开路的 DONE 输出信号何时释放;

2. GSRIinactive 确定为所有 CLB 和 I/O 触发器的全局 SET/RESET 信号何时被释放,在配置末尾,有设置方向(FDSD)配置的触发器被置位,而所有其它的触发器被复位;

XC4000 器件没有降电压模式,提供全局三态网线保持所有输出在高阻状态以减少电源消耗。

3. OutputActive 确定有关配置腿转到用户定义功能的时间。

整个配置过程器件的输入腿是有效的,它意味着加到输入而不是配置腿的信号通过器件传播。

当为给定器件选择特定的起动方案时,首先决定用什么时钟,然后决定板上的系统是否需要用 DONE 信号同步。XC4000 系列允许用内部 CCLK 驱动起动顺序或外部用户时钟,MakeBits 缺省值表示利用内部 CCLK,但是,如果 FPGA 需要用系统的其余部分来起动时,可以用外部用户时钟,为利用用户时钟,必须在设计中把时钟接到起动符号的 CLK 腿上,并设置相应的 MakeBits 选项:

1. CCLK_NOSYNC—内部 CCLK 驱动起动,不与 DONE 同步;

2. CCLK_SYNC—内部 CCLK 驱动起动,与 DONE 同步;

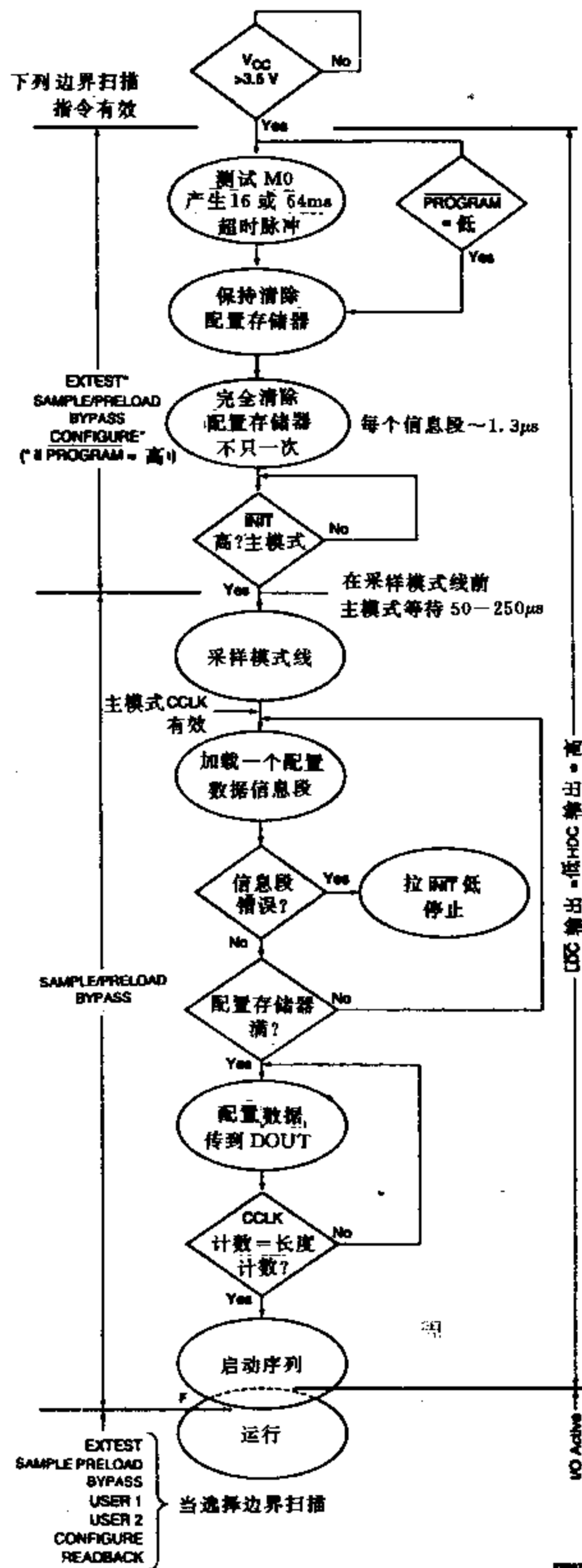


图9.16 XC4000配置过程状态图

3. UCLK_NOSYNC—外部用户时钟驱动起动,不与 DONE 同步;

4. UCLK_SYNC—外部用户时钟驱动起动,与 DONE 同步;

当并行地或在菊花链中配置多个 FPGA, DONE 信号可以连接在一起,以使所有的器件同时变高电平,最终的 DONE 信号可以用于对所有的器件同步 OutputsActive 和 GSRIinactive。可以在内部和外部时钟之间选择,是否选择与 DONE 信号同步。在数据错误的事件中,可以利用 Sync-To-DONE 防止正确配置领先的器件变为激活。

表9.4给出在 MakeBits 中可设置的四个选项。

表9.4 MakeBits 选项

MakeBits 选项	DoneActive	OutputsActive	GSRIinactive
CCLK_NOSYNC	C1,C2,C3,C4	C2,C3,C4	C2,C3,C4
CCLK_SYNC	C1,C2,C3	C2,C3,DI,DI-PLUS-1	C2,C3,DI,DI-PLUS-1
UCLK_NOSYNC	C1,U2,U3,U4	U2,U3,U4	U2,U3,U4
UCLK_SYNC	C1,U2	BU2,DI,DI-PLUS-1 DI-PLUS-2	C2,C3,DI,DI-PLUS-1

注: C1,C2,C3,C4分别为长度计数比较后 CCLK 的第一,二,三,四上升沿

DI, DONE 信号变高的时间

DI-PLUS-1, DONE 信号变高后第一个 CCLK 上升沿

U2,U3,U4: 在 C1之后用户时钟的第二,三,四有效上升沿

DI-PLUS-1, DI-PLUS-2: 在 DI 之后用户时钟的第一,二上升沿

当利用用户提供的起动时钟,在 C1之后的第一个用户时钟定时附加的寄存器,解决由 CCLK 和用户时钟之间不同步关系引起的潜在的似稳问题,对一个链中的多个 XC4000 器件,建议要求用户时钟选项时起动顺序与 DONE 同步。当利用 NOSYNC 模式,即使 DONE 腿外部保持为低,器件变为可运行的。

§ 9.6.5 重新编程

器件在系统内运行的时候配置存储器可以重写,重新编程可能要求加载不同的配置或在供电中断后重新加载配置。为了再编程,把 $\overline{PROG}$ 腿拉低,器件返回到清除(Clear)状态,清除配置存储器,禁止 I/O 腿,重新采样模式线,如前一节所述。

为了减少噪声的影响,这些重新编程信号在内部时序发生器的2—3周期内(2—6 μ s)被滤波,一旦器件识别一个稳定的请求,它保持 DONE 为低直到新的配置已被完成;即使 DONE 外部保持为低超出配置周期,器件在配置完时开始运行,除非利用 Sync-To-DONE。

在重新编程时,Xilinx 器件不提供超时延时,所以,必须提供确保靠后的器件可以清除它们本身配置存储器的时间。对主模式或外设模式重新编程清除超时没有在上电条件中的40—160 μ s 延时,如利用菊花链,把 $\overline{INIT}$ 腿连在一起,保证菊花链直到所有器件完成清除才开始配置。

第十章 XATC 设计编辑器(XDE)

XATC 设计编辑器(XDE)包含几个能使设计者对已有的设计进行修改或产生新设计的程序,这里说明 Edit LCA 程序,一个 FPGA 设计的器件级图形编辑器。要注意的是利用 XDE 所做的设计修改在原理图上得不到任何反映,所以应该把任何布局和逻辑的修改记录到原理图上,在原理图标志这些变化确保设计的完整性和保持所有的修改记录在设计中。通常,可以利用 XDE 看一下在 XDelay 程序中找到的时序信息或对已有的 FPGA 设计文件进行修改。

§ 10.1 XDE 的实现过程

为了使用 XDE,在 DOS 提示符下键入“XACT”,PC 机加载 XDE 程序,出现如图 10.1 所示的显示屏幕。

在 XDE 中,可以有许多不同的方式输入指令和信息,在 XDE 开始运行之前,必须执行以下的任务:

§ 10.1.1 设置模式

为了能够修改设计,必须在 Expert 模式下运行 XDE,在现在的模式上按动鼠标的左键,XDE 显示带有两个模式选项:Expert, Safe 的对话框。

在 Expert 上按动鼠标一次,所选项加亮。

选择 Done 关闭对话框,返回到主屏幕。

§ 10.1.2 设置目录

在现有设置部分的目录行上显示了现有目录的路径。为转入到设计文件,进行以下步骤,用鼠标的左键从菜单选 Designs→Directory,系统显示一个对话框使设计者不用退出程序而改变现在的目录。设置路径到已有的工作目录,C:\XACT\TUTORIAL\CORE\EDIT_LCA,然后选择 DONE 改变目录。

§ 10.1.3 选择设计文件

执行以下的步骤:用鼠标的左键从 XDE 主菜单选择 Designs→Design

可以用以下任一方法,指示设计文件的名字:

- * 从 Select File 指令行,键入:"edit" <Enter>

- * 从弹出的窗口,选择 edit_lca.lca,然后选 DONE。文件名字出现在现有设置部分的 Design 行上。

§ 10.1.4 加载设计

加载设计从菜单上用鼠标左键选 Programs→Editlca,XDE 加载称为“example”的设计,

读 Editlca.pro 的轮廓,显示 LCA 编辑器的屏幕。

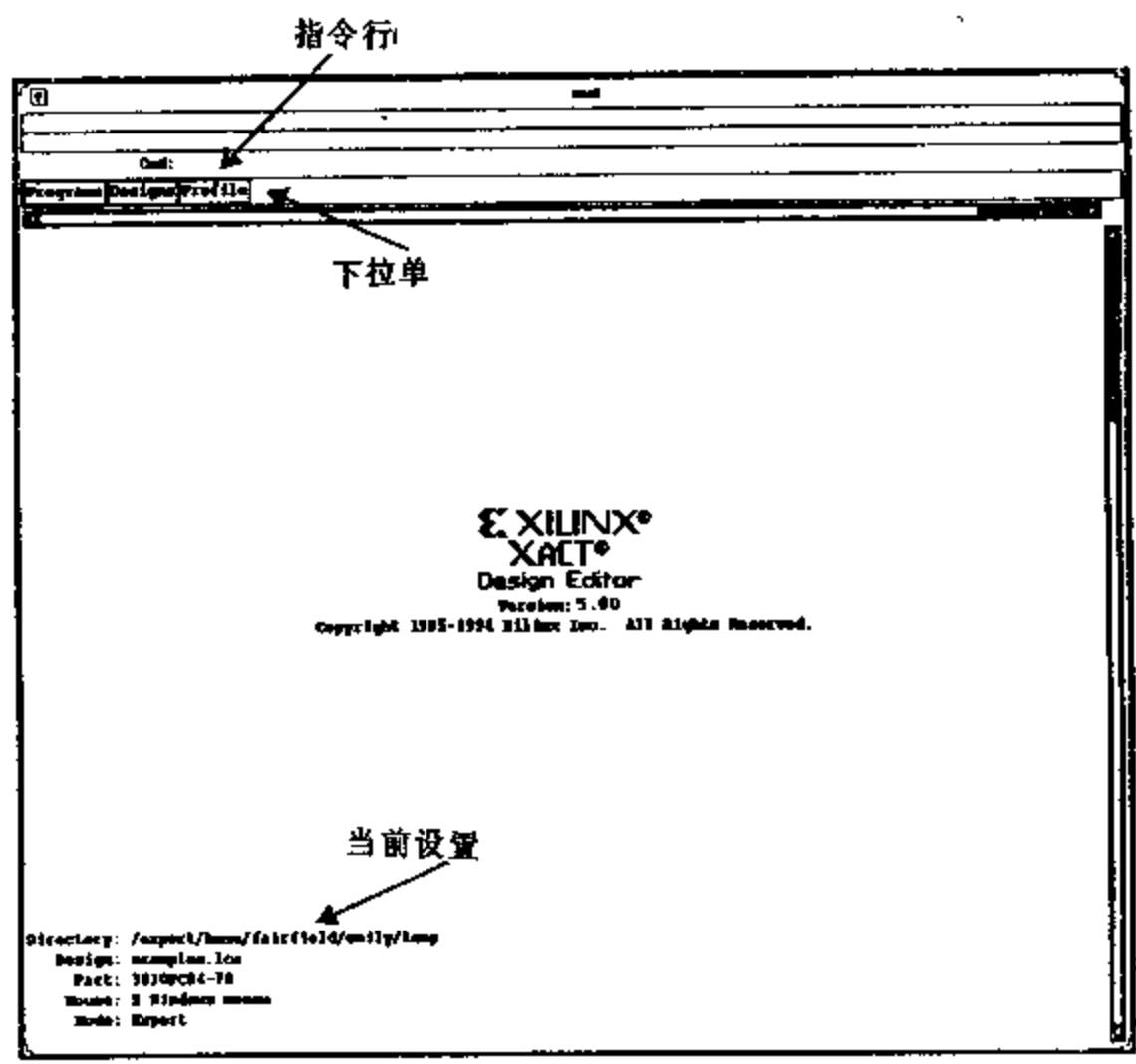


图10.1 PC 机的 XDE 屏幕

§ 10.1.5 剖视 FPGA

如图10.2的 EditLCA 屏幕是 FPGA 的图形表示,它由可配置逻辑块(CLB),可配置 I/O 块,三态缓冲器(TBUF),可编程互连线组成。

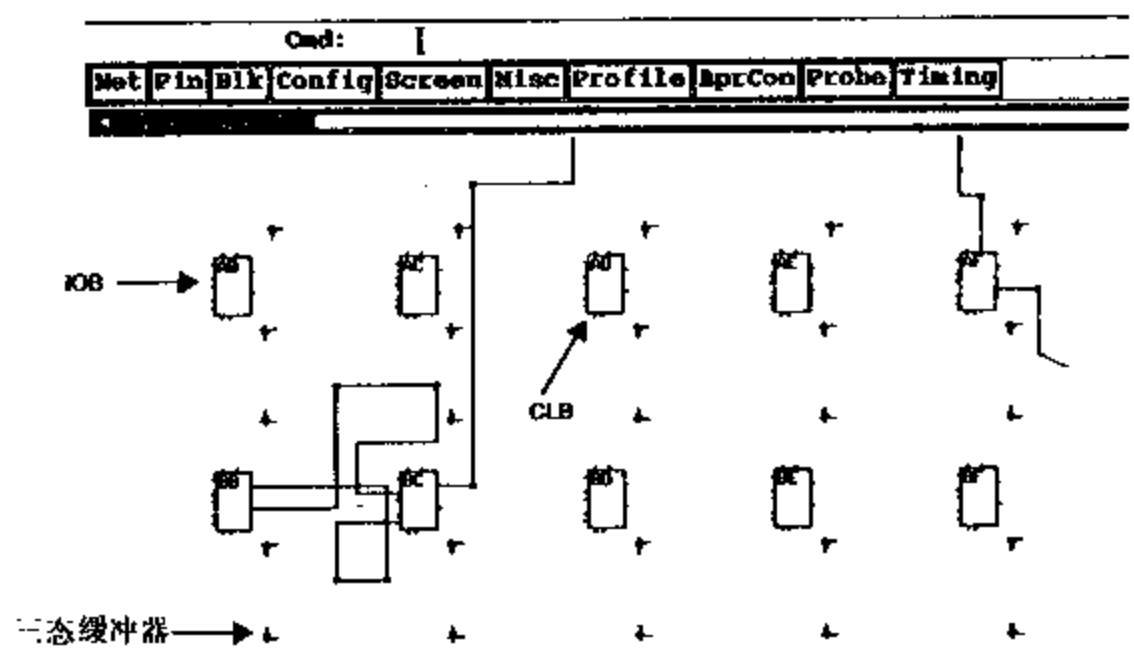


图10.2 EditLCA 的屏幕

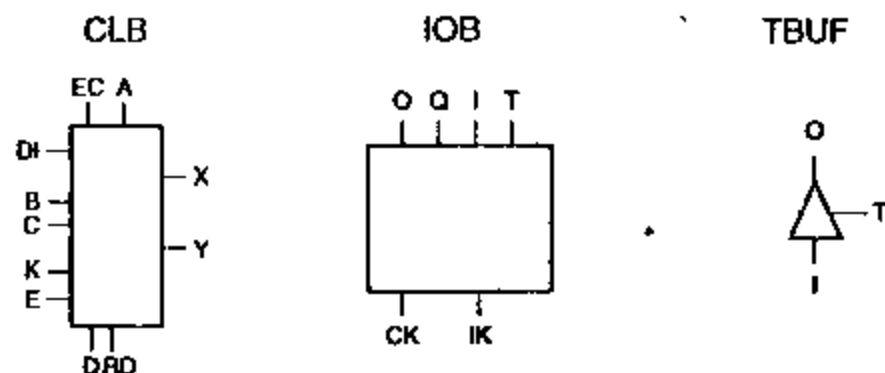


图10.3 功能块类型

图10.3说明了这些功能块类型的输入和输出腿。

1. 定义: Xilinx 利用了以下的定义:

功能块:设计的逻辑元件,例如,CLB 或 IOB

出 腿:一个功能块的任何输入或输出

网 线:一组在最后的设计中应该互连的腿

每个 CLB 有一物理位置名,用它的水平行的位置跟随它的垂直列位置来定义,例如 JC,每个 IOB 有它的封装腿数定义的位置名,例如 P36,对一个设计中利用的功能块,描述块的功能的“逻辑”名代替了这些位置的标志。

2. EditLCA 的屏幕:

可以执行以下的步骤来熟悉 EditLCA 屏幕

1. 按住鼠标的按钮移动,在屏幕的右下角出现了全视图,表示了整个芯片,红色的矩形把 FPGA 现在出现在屏幕上的部分包围起来。

2. 按住鼠标按钮连续移动鼠标扫视整个设计的其余部分,未用的逻辑单元资源用绿色表示,使用的功能块用黄色表示。

3. 释放鼠标按钮,全视图的窗口消失。

扫视整个设计一直到找到一组配置的逻辑块。把指针放到 CLB 的已利用的输入腿,注意指针的状态行显示关于此腿的有关信息。

§ 10.1.6 设置轮廓(profile)

当运行 EditLCA 程序时,XDE 读 Editlca.pro 文件,这个文件包括鼠标按键,功能键,屏幕设置,彩色设置等缺省的设置,因为有如此多的变量可选择,下面说明如何设置这些选项的某一些使得定制此编辑器以适合个人的口味。

1. 设置鼠标的按键:可执行以下的步骤来设置鼠标的按键

1) 用鼠标的左键,由 EditLCA 的菜单选 Profile→Mouse,出现一个上托窗口,列出鼠标的左,中,右按键,分别用 B1,B2,B3标志。

2) 为了设置鼠标的左按键,从上托窗口上选 B1,出现另一个上托窗口,列出了可以赋予鼠标按键的功能。

3) 选 Select,从现在起,当 XDE 允许选某项目时,把指针放到其上,按鼠标的左键。一旦在这个上托窗口中选择一个功能,XDE 返回到列出三个鼠标按键的上托窗口。

4) 为了设置鼠标的中间按键,选 B2。

5) 从上托窗口选 Done, 从现在起, 为完成一个指令, 或者从屏幕的顶部选 Done, 或者按鼠标的中键。

6) 为了设置鼠标的右按键, 从上托窗口选 B3。

7) 选 Switch, 从现在起, 鼠标右键执行 Switch 指令, 当利用 EditBlk 指令, 按动这个按键在 EditLCA 屏幕和 EditBlk 屏幕之间进行转换。

8) 现在已经完成所有三个鼠标按键的定义, 选择 Done 退出上托窗口。

2. 定义功能键: 下一步是设置功能键, 可以定义功能键来执行经常使用的指令, 代替经常的键入指令或通过菜单搜索, 可以定义功能键在按动一个按键时执行这些指令。

1) 由 EditLCA 的菜单选择 Profile→Keydef。系统显示一个上托窗口, 列出所有有效的功能键。F1 被保留为 Help, 所以从 F2 键开始定义。

2) 在上托窗口中, 选择规定的功能键, 例如 F2, 系统关闭上托窗口, 把指针放到 Enter key Definition 指令行上。

3) 键入想要映射到这个功能键的指令, 如

"editnet" <Enter>

这个过程定义 F2 执行 EditNet 指令。也可在指令行直接输入以下的语句来定义功能键

"keydef F3 routpin" <Enter>

为了方便地利用键盘产生功能键的映射, 执行以下的步骤:

(1) 利用向上箭头, 把最后的指令返回到指令行。

(2) 利用左箭头键, 沿指令行反向移动。

(3) 编辑指令, 以使它读出为: "keydef F4 unroutpin" <Enter>

这个方式可以输入类似的指令而不用重新键入整个指令行。

4) 按照表 10.1 设置剩下的功能键。

5) 当完成设置时, 保存此设置, 键入:

"Saveprofile" <Enter>

从现在起, 在这个目录中任何时候使用 XACT, 这些鼠标按键和功能键设置是缺省的。

表 10.1 功能键设置

功能键	定 义	功能键	定 义
F5	SwapBlk	F6	SwapSig
F7	MoueBlk	F8	EditBlk
F9	Route	F10	Unroute
F11	Query Net	F12	Dos

§ 10.1.7 利用 EditBlk 显示

这节描述 IOB 和 CLB 的内部图形表示, 当编辑一个功能块, 显示从 EditLCA 屏幕转换到 EditBlk 屏幕, 前者呈现所有功能块的外部, 后者显示单个功能块的内部, 为了实践利用 EditBlk 的显示, 执行以下的步骤:

1. 移动指针使得芯片的左下角显示在屏幕上。

2. 用鼠标左键, 由菜单选择 Blk→EditBlk, 或选相应的功能键, 此时为 F8。

这个选择允许观察现在建立的被选的功能块, 以编辑由这个功能块执行的逻辑功能, 指令行显示:

{Select block;}

3. 选择功能块 P36, 或者用鼠标选它, 或者用键盘输入块名, 屏幕现在表示为 I/O 块 P36 的 Block Editor 的显示, 如图 10.4 所示。

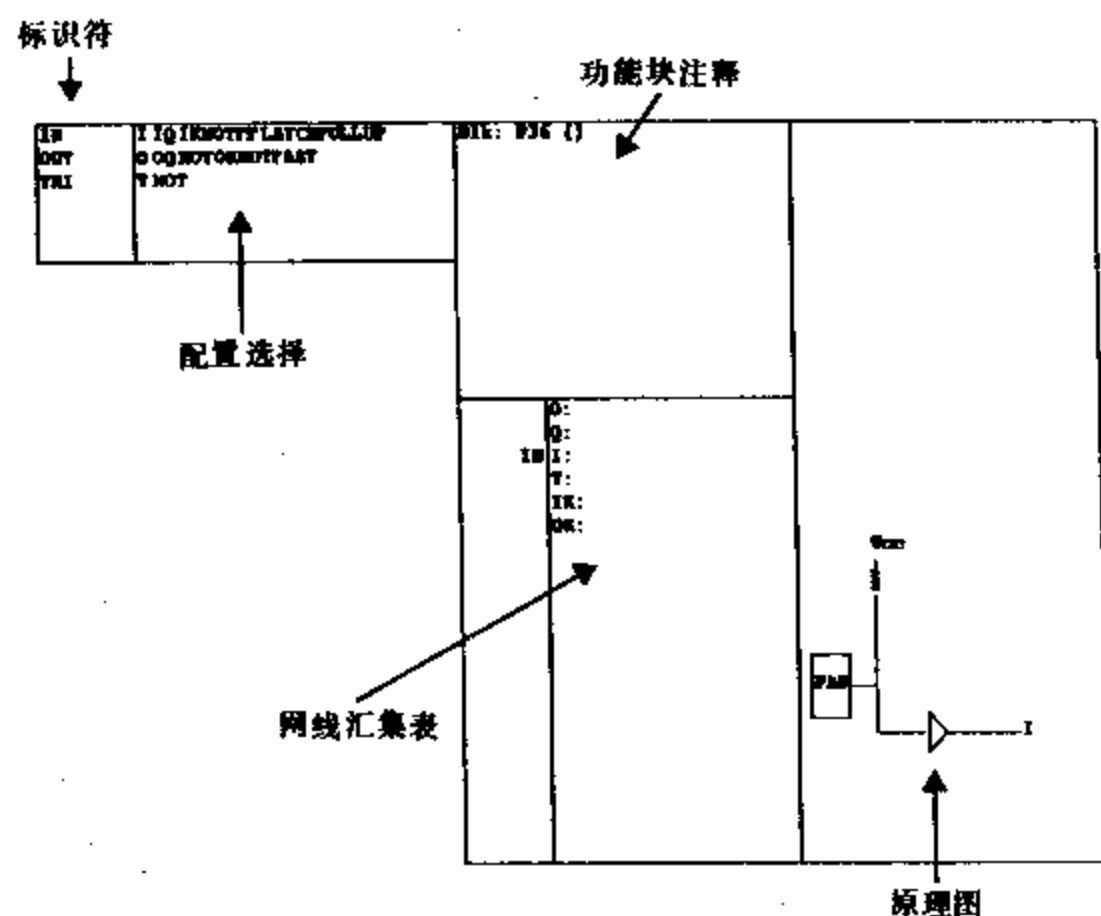


图 10.4 EditBlk 的 IOB 屏幕

可以在 EditBlk 屏幕和 EditLCA 的屏幕之间用以下方法进行转换

从 EditBlk 菜单中选 Screen→Switch

在指令行输入 Switch

按鼠标右键, 它已被定义为 Switch 按键。

4. 转换到 EditLCA 屏幕, P36 功能块现在是红色的, 这表示 P36 是激活的块, 在转换指示被执行时出现此功能块。

为了编辑另一功能块, 执行以下的步骤:

5. 用鼠标的左键, 从菜单选 Blk→EditBlk, 指令行显示: {Select block;}

6. 键入所要的功能块标志, 如 "JC" <Enter>

屏幕转到表示对功能块 JC 的 Block Editor 的显示, 如图 10.5。

7. 在每个配置选项上按动鼠标的左键来选择它, XDE 更改原理图的面积来反映此变化, 在配置区域, 系统以黄色加亮被使能的标记, 以绿色表示未利用的标记, 不仅可以改变布局 and 布线, 而且在 CLB 级的功能上编辑设计。

表 10.2 描述了这些标记的功能。

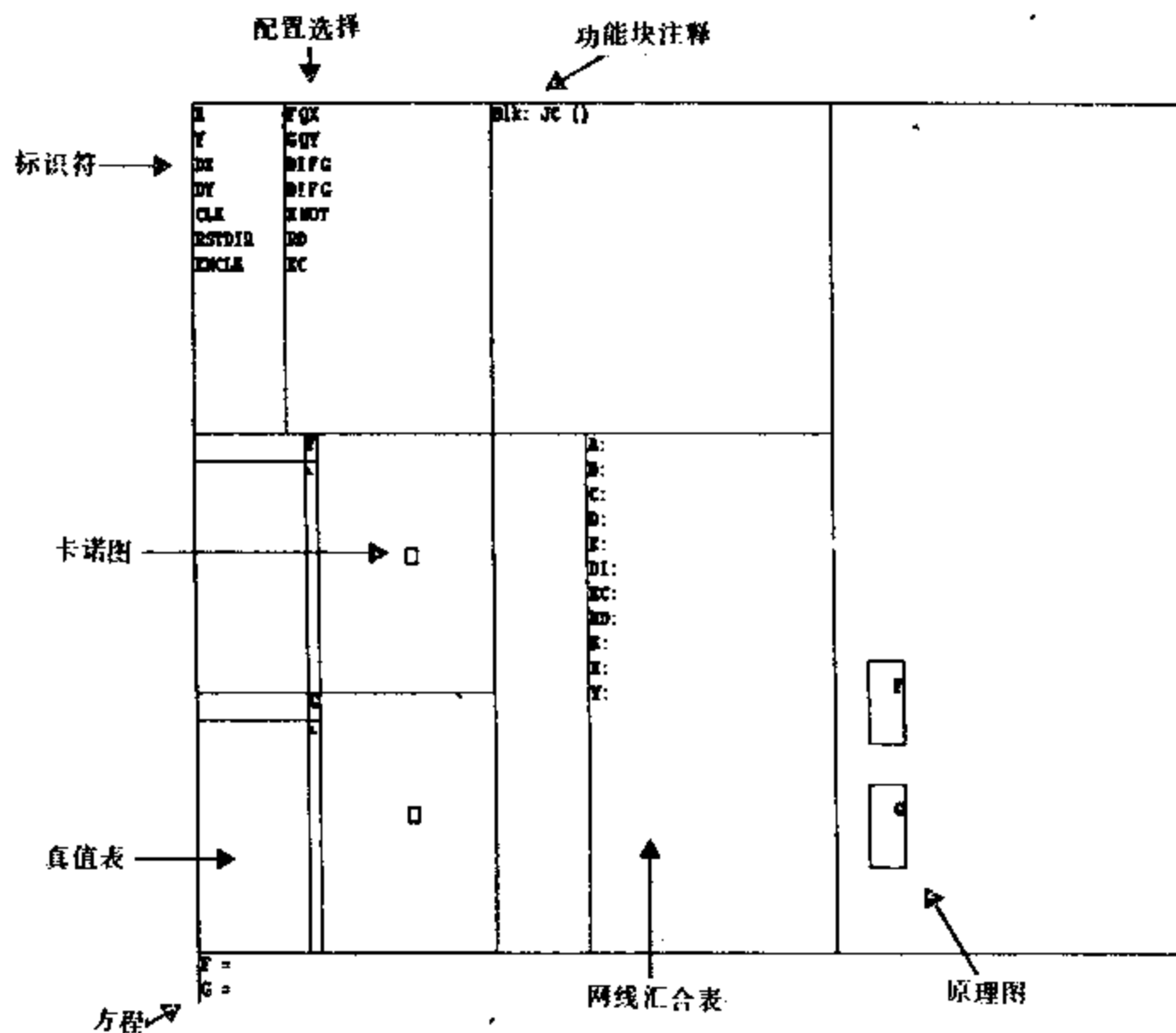


图10.5 EditBlk 的 CLB 屏幕

表10.2 标记定义

标记	选项	描 述
X	F	X 标记表示来自 CLB 顶部的输出(如图9.3)
		选 F 置 X 等于 F 函数发生器的输出
	QX	选 QX 置 X 等于顶部触发器的输出(QX)
Y	G	Y 标记表示来自 CLB 底部的输出
		选择 G 置 Y 等于 G 函数发生器的输出
	QY	选 QY 置 Y 等于底部触发器的输出(QY)
DX	DI	DX 标记表示顶部触发器的输入,
		选 DI 置 DX 等于 CLB 的直接输入腿
	F	选 F 置 DX 等于 F 函数发生器的输出
	G	选择 G 置 DX 等于 G 函数发生器的输出
DY	DI	DY 标记表示顶部触发器的输入,
		选 DI 置 DY 等于 CLB 的直接输入腿
	F	选 F 置 DY 等于 F 函数发生器的输出
	G	选择 G 置 DY 等于 G 函数发生器的输出
CLK	K	CLK 标记表示两个存储单元的时钟输入
		选 K 置触发器的时钟输入等于 CLB 的 K 腿,触发器在 K 的上升沿被定时。
	NOT	选 NOT 置触发器的时钟输入等于 K 的腿的补。
RSTDIR RD		RSTDIR 标记表示高有效复位两个触发器的直接输入,
		选 RD 置两个触发器的复位输入等于 CLB 的 RD 腿,当不选这个标记,RD 输入被去使能
ENCLK EC		ENCLK 标记表示高有效时钟使能两个触发器的输入,
		选择 EC 置两个触发器的使能时钟输入等于 CLB 的 EC 腿,当不选这个标记,EC 腿被去使能

图10.6表示一个已配置 CLB 的例子,带有用户赋予的块名,在顶部的中心方框内的注释和在显示器下部中心的出腿名。

为配置 CLB 的 JC,执行以下的步骤:

1. 在指令行键入以下的方程:

"f=a * b@d" <Enter>

真值表和卡诺图表示对方程 F 的说明。

2. 指针移到卡诺图,选择一个方块,在其上按动鼠标左键,注意方程反映了这个改变。

3. 终止 BlockEditor,返回到 LCAEditor 的图形显示,在指令行上键入

"endBlk" <Enter>

EndBlk 与 Switch 不同,EndBlk 结束 EditBlk 的程序,JC 块不留下红色,此时试图转换,不发生变化,因为不再有激活的功能块。

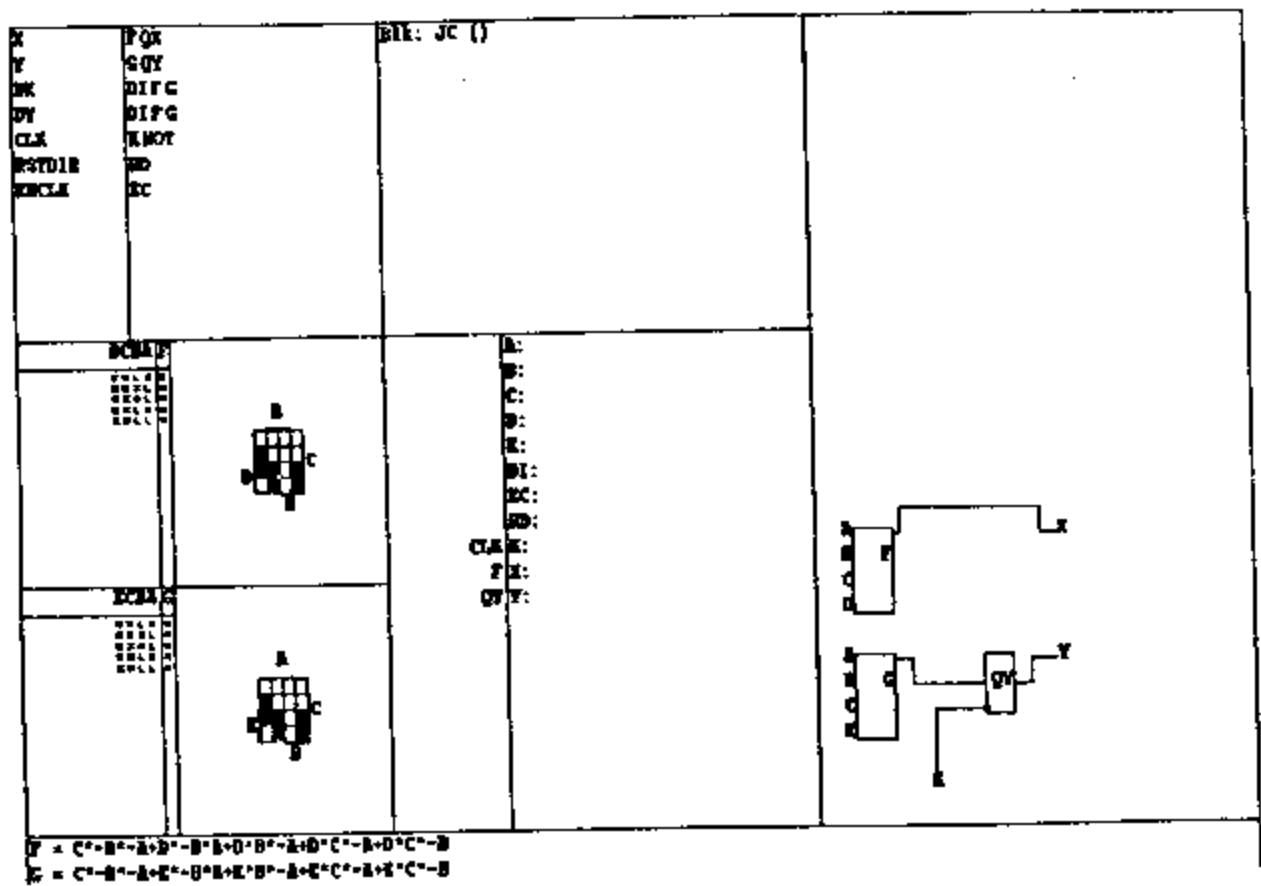


图10.6 已配置的 CLB 屏幕

§ 10.2 高层次编辑

在熟悉 EditLCA 和它的某些特性后,可以执行某些编辑,高层次编辑包括利用 XDE 的高级功能块和信号指令来构造或重新构造一个设计,高层次编辑要求以下指令的知识。

§ 10.2.1 SwapSig

SwapSig:指令交换两个腿的连线和块的功能,换言之,两个腿的属性被交换,但是设计

的功能保持不变。下面的图10.7和图10.8表示执行 SwapSig 前后的结果。

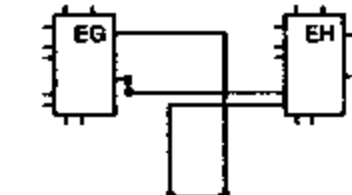


图10.7 SwapSig 之前

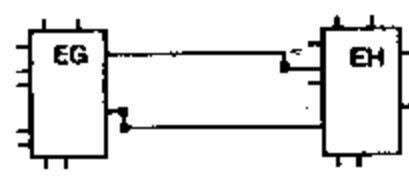


图10.8 SwapSig 之后

可以方便地利用这个指令重布网线,它可以减少定时延迟和释放互连资源,为了理解 SwapSig 的用途,执行以下的步骤:

- 1. 在指令行键入:
"find bb" <Enter>
系统把指针放到 CLB BB 上(也可以利用 Find 定位 IOB 和网线)
- 2. 从 EditLCA 菜单选 Net→QueryNet,出现 QueryNet 的上托窗口,指令行显示如下:
Select option(S) or net(S):
- 3. 为了利用鼠标选择网线,把指针放到连线的腿上,按动左键,在以下的腿上按动:
BB. X Selects net net0
BB. Y Selects net net1
- 4. 当完成后,选 Done
屏幕转换到文本模式,显示关于 net0和 net1的信息
- 5. 按任何键返回到图形屏幕

图10.9表示屏幕的显示,注意连线的时序

```
net0 ..... BB. X ..... 2.5BC. E
net1 ..... BB. Y ..... 1.0BC. B
```

图10.9 Querynet 显示

- 6. 现在由菜单选择 Pin→SwapSig 来交换两个信号,指令行显示:{From Pin:}
- 7. 在指令行键入:"BB. X" <Enter>
状态行显示:{pin BB. X ;Selected}
指令行显示:{To Pin:}
- 8. 选择要交换布线连接和块功能的腿,对这个例子,在指令行上输入 BB. Y 腿:
"BB. Y" <Enter>
两个腿的连接线和 CLB 功能被交换,出现以下三个信息:
{pin BB. X and BB. V: Signals Swapped}
{pin BB. X ;Routed}
{pin BB. X ;Routed}
- 9. 现在执行另一个 QueryNet,如图10.10说明对相同的两条连线,比较新的时序延迟和以前的延迟,在此情况中,交换腿导致 net1更短的布线通道,减少了它的延迟。

```
-net0 ..... BB. X ..... 2.5BC. E
-net1 ..... BB. Y ..... 1.0BC. B
```

图10.10 QueryNet 显示

除了时钟的 K 腿和直接输入的 DI 腿,可以交换大多数的出腿,但不可以交换带有输出腿的输入腿;取决于 CLB 是如何配置的,不可以直接互相交换一些通用的输入腿。

在第六章中图 6.13 和图 6.21 分别给出了 XC3000 和 XC2000 系列在一个 CLB 中所有可能的配置。

§ 10.2.2 SwapBlk

SwapBlk 指令交换两个功能块的配置和网线的连接;例如两个块交换位置,不影响设计的功能,因为某些确定的设计要求(如通过临界连线的延迟,布线资源的利用)可能要让某些功能块放在确定的有关位置。

可以利用 SwapBlk 指令改变功能块位置,为实践使用 SwapBlk,执行以下的步骤:

1. 在指令行键入:"find da" <Enter>
2. 由 EditLCA 菜单选择 Blk→SwapBlk,指令行显示:
{Select Block/;}
3. 在功能块 DA 上按动鼠标左键一次,或在指令行键入:"da" <Enter>
信息行显示:{Block DA;Selected}
指令行显示:{Select Block 2;}
4. 在功能块 DB 上按动鼠标左键一次,信息行显示:
{Block DA and DB;Swaped}

图 10.11 至图 10.14 分别说明在利用 SwapBlk 前和后的连线分配和 CLB 配置

X; Y; G F; G; B; E DX; DY; ENCLK; RSTDIR; CLK;

G = $\sim E + B$

A =

X =

B = net4

Y = net5

C =

D =

DI =

E = net3

EC =

RD =

K =

X; Y; G F; C; D G; D DX; DY; ENCLK; RSTDIR; CLK;

G = $\sim C + D$

A =

X = net3

B =

Y = net4

C = net2

D = net6

DI =

E =

EC =

RD =

K =

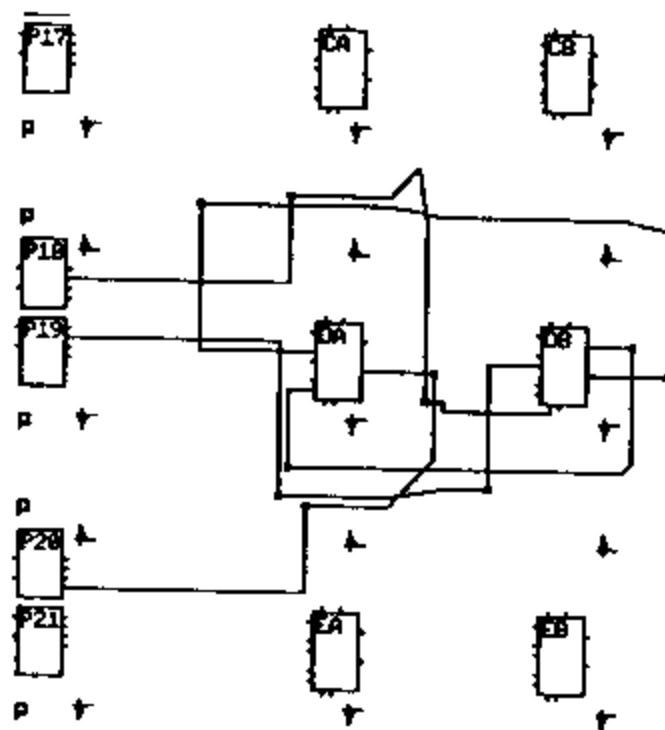


图 10.11 SwapBlk 前的 CLB 配置

图 10.12 SwapBlk 前的网线分配

```

X: Y:G F: C: D G: D: E DX: DY: ENCLK: RSTDIR: CLK:
G= ~C+D
G= ~D
A=                X=net3
B=                Y=net4
C=net2
D=net6
DI=
E=
EC=
RD=
K=
X: Y:G F: C: D G: D: E DX: DY: ENCLK: RSTDIR: CLK:
G= ~C+D
G= ~D
A=                X=net3
B=                Y=net4
C=net2
D=net6
DI=
E=
EC=
RD=
K=

```

图10.13 SwapBlk 后的 CLB 配置

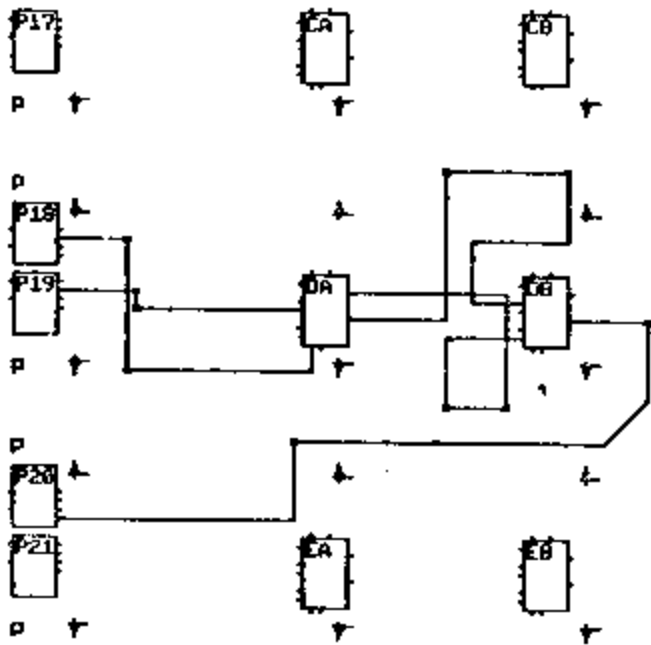


图10.14 SwapBlk 后的网线分配

§ 10.3 低层次编辑

低层次编辑是 XDE 十分强的特性,给予整个布线通道上的绝对控制,当试图满足时序限制时,或在封装的 FPGA 中试图布通最后的连线时这是十分有用的,这类编辑要求对 FPGA 结构和有效的布线资源有详尽的了解,这些资源由可编程互连点(PIP),开关矩阵等组成。

图10.15是带有 PIP 和所示矩阵的布线例子。
为了人工地在这个层次上布线,必须能看到 PIP 和开关矩阵,在 Screen 菜单的 Show 选项选择屏幕显示的各种选项。

第一组选项确定什么项目被显示,参考表10.3

表10.3 显示选项

选 项	描 述
Pips/Nopips	打开或关闭 PIP 和开关矩阵的显示
World/Noworld/NeverWorld	打开或关闭整图的显示
Used/AvailablePIP	显示所用的互连段(连到被编程的 PIP 或开关矩阵腿)
	显示现在未用的互连线
Bidis/Nobidis	打开或关闭为通用互连的双向缓冲器位置的显示
Matrix/NoMatrix	打开或关闭通过开关矩阵的布线选项的显示

第二组选项确定编辑屏幕的放缩级别:

XSmall/Small/Medium/Large/Xlrge

In 放大一级

Out 缩小一级

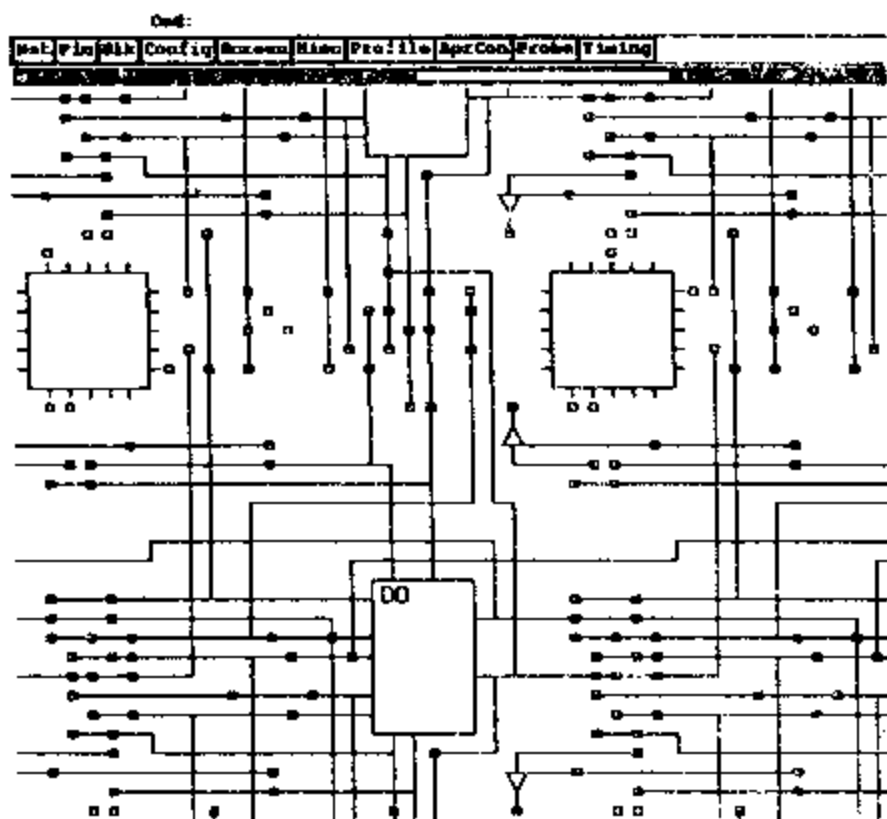


图10.15带有 PIP 的 EditLCA 屏幕

§ 10.3.1 屏幕选项

如下设置屏幕选项:

1. 由菜单选 Screen→Show 出现屏幕选项的上托窗口

2. 由上托窗口,使能以下的选项:

Pips Matrix In

3. 选 Done 关闭缩放级上托窗口,信息行显示: {Drawing Screen }

当重画屏幕完成时,除 CLB 和 IOB 外屏幕显示可编程互连点(PIP)和开关矩阵

4. 保存这个配置,在指令行键入:

"Saveprofile" <Enter>

§ 10.3.2 开关矩阵

开关矩阵连接金属段的末尾,并允许在相邻的互连的行和列之间相互连接,由自动布线器通过开关矩阵可以建立连接,或者利用 EditNet 指令人工地选择要求的几对矩阵腿。

把指针放到开关矩阵的腿上。

因为使能 Show Matrix 选项,XDE 加亮对此腿的所有合法的开关矩阵组合,图10.15已给出了在一个开关矩阵中所有腿的可能的连接。

§ 10.3.3 可编程互连点(PIP)

PIP 是唯一的可以连接固有的线段的地方,可以把它们想象为 on/off 开关,仅仅当被使能时传递一个信号,但是这些开关的某些是有方向性的——它们仅仅一个方向传导信号。

把指针放在任何的 PIP 上,状态行指示 PIP 的方向性,如表10.4

表10.4 PIP 的方向性

方向性	描 述
ND	无方向性的互连
D,H→V	从水平到垂直线驱动的 PIP
D,V→H	从垂直到水平线驱动的 PIP
D,C→T	从 T 的水平段到垂直段驱动的 PIP
D,T→C	从 T 的垂直段到水平段驱动的 PIP
D,CW	顺时针方向驱动角的 PIP
D,CCW	逆时针方向驱动角的 PIP
P0	非传导(OFF)的 PIP
P1	传导(ON)的 PIP

因为这些定义可能对新用户不清楚,XDE 有一个指令把方形的 PIP 改成表示方向性的箭头。在指令行,键入:

"Show directional" <Enter>

指到右边的 PIP 箭头是水平地驱动的 PIP(右和左两边),向上指的 PIP 箭头是垂直地驱动的 PIP,(上和下两边),仍作为方形出现的 PIP 是在任何方向驱动时无方向性 PIP。

§ 10.3.4 布线资源

为了有效地在较低的层次上布线,应该对不同布线资源的有效性有好的理解,和如何最好地利用它们,可编程互连由三个固有的类型组成。

1. 直接互连

直接互连表示在图10.16,提供最有效的方式连接相邻功能块之间的连线,利用直接互连布线的信号有十分短的延迟,且不利用通用互连资源,对于每个 CLB,.X 的输入可以直接连接到紧挨其右边的 CLB 的.B 的输入和紧挨其左边的 CLB 的.C 的输入,.Y 的输出可以直接连接到紧挨其上边的 CLB 的.D 的输入和紧挨其底下的 CLB 的.A 的输入。

以下操作表明了任何时候利用直接连线的好处:

1)在指令行键入:"querynet" <Enter>

系统显示一个带有选项的上托窗口,指令行显示:{select net(s) or option(s):}

2)指令行键入:"net3" <Enter> "net" <Enter> <Enter>

注意:最后的回车符表示已完成输入网线名到系统中。

3)对网线延迟加注释,然后交换块 DA 的输出

在指令行输入:"Swapsig" <Enter>

指令行显示:{From pin}

4)在指令行输入第一个出腿:da.x <Enter>

指令行显示:{To pin}

- 5)在指令行输入另一个出腿:da. y
 - 6)注意此网线的一条已利用直接互连布线,而另一条还没有
 - 7)由菜单选择 Net→QueryNet
 - 8)在指令行输入:da. x
 - 9)在指令行输入:da. y
 - 10)选择 Done,系统显示一个上托窗口,列出指定网线的延迟
- 利用直接互连的 Net4的延迟是1.0ns,而其它的延迟是2.5ns。

取直接互连的优点来布局功能块和选择出腿位置是有好处的,因为直接互连有最短的布线延迟,而没有利用通用互连资源,因此减少了布线密度和布线的拥挤。

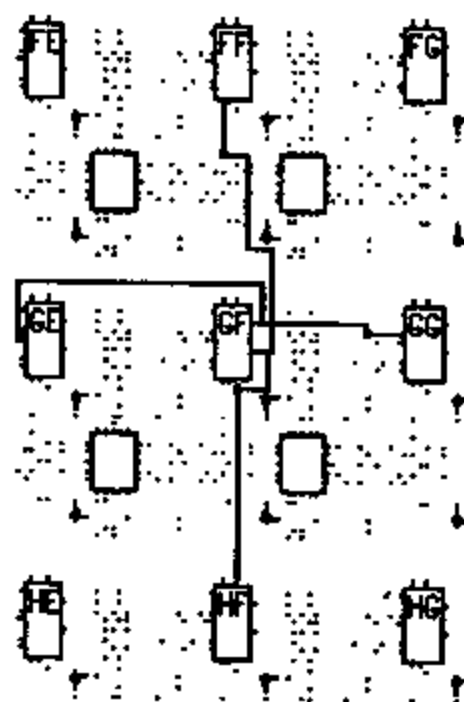


图10.16直接互连布线资源

2. 通用互连

通用互连是由栅格状的水平和垂直走线段组成,这些线段可以通过可编程互连点 PIP 及开关矩阵互连起来,如图10.17。

在通用互连区域内的专用缓冲器为改善长的网线性能提供信号的复原,互连缓冲器在给定的互连段上按要求的方向传递信号,这些双向缓冲器(BIDI)邻近于开关矩阵可找到,在使用 Show Bidis 指令时以红色加亮它们。

3. 长线:长线避开开关矩阵,主要是为必须走长距离的或在多个目标之间必须有最小扭曲的信号而设的。如图10.18所示,长线垂直地或水平地通过互连区域的高度或宽度,比XC3020大的器件,两根垂直长线是可连接的半长度线。

位于芯片左上角的全局时钟缓冲器(GCLK)驱动在每列中保留的一根长线,它连接到逻辑块的.K输入,利用时钟信号的全局缓冲器提供一个对所有 CLB 和 IOB 有效的无扭曲高扇出同步时钟,在芯片左边顶上数第二个 I/O 焊盘上直接存取到这个缓冲器是可行的。

记住在所有的设计中利用 GCLK,因为确定的长线是专用于全局时钟缓冲器,它们仅当 GCLK 被规定后才利用。备选时钟缓冲器(ACLK)类似于 GCLK,位于 FPGA 芯片的右下角。

在 XDE 中没有指令显示这些长线,所以必须间接对它们定位,垂直长线可以在 CLB 的左边找到,最左和最右的长线通过 PIP 连到 K 时钟腿,水平长线在开关矩阵的上和下来定位,这些长线交叉通过连接到 TBUF 输出端的 PIP。

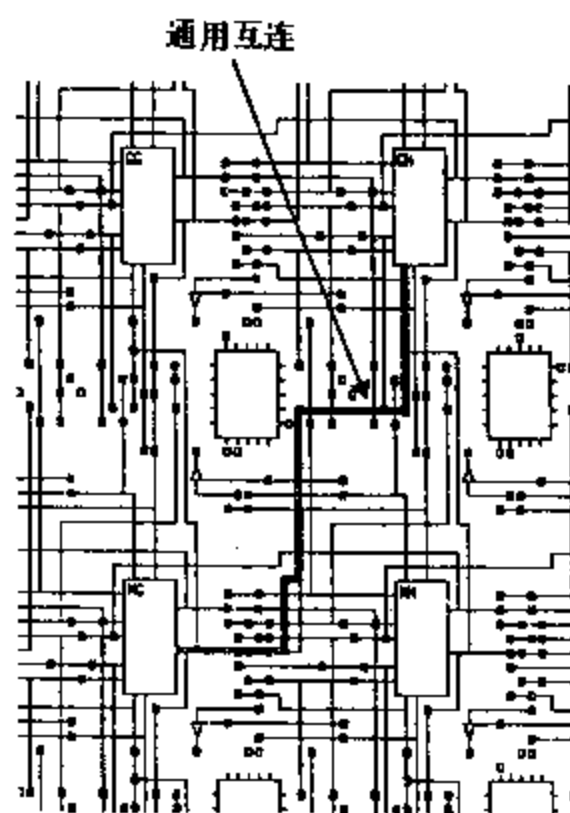


图10.17 通用布线资源

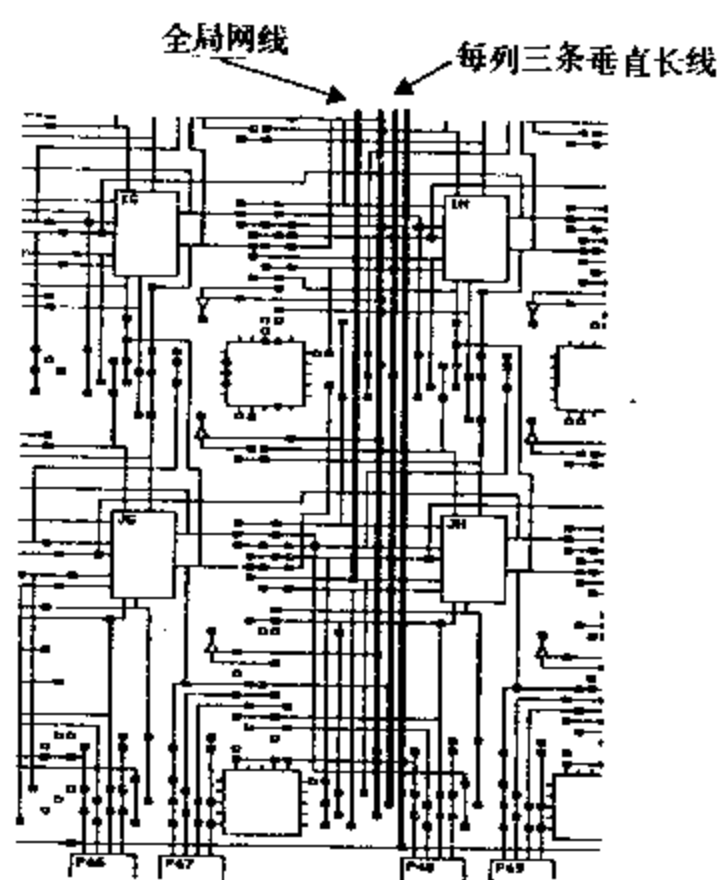


图10.18 XC3000系列布线资源

§ 10.3.5 人工布线

通常,当修改一条网线时,XDE 自动配置相应的互连,但是也可以人工布线,以下描述通过开关矩阵的布线。

1. 禁止自动布线选项

为进行人工布线,执行以下几个步骤关掉自动布线器:由 EditLCA 选择 Profile→Autoroute,系统显示带自动布线选项的上托窗口。用鼠标左键选择 off,则 XACT 的自动布线选项被禁止,现在,当产生网线或移动出腿时,必须人工地布线这些网线。

2. 人工布网线

为实现人工布网线,执行以下的步骤:

1) 在指令行键入:"find net8" <Enter>

指针移到网线起始的出腿上

2) 由 EditLCA 菜单,选 Net→Hiligh,系统显示带彩色选项的上托窗口

3) 从上托窗口选 Magenta,指令行显示:{Select net(s):}

4) 输入:"net8" <Enter>

5) 选择 Done

系统以品红色加亮与这些出腿有关的互连线

6) 由 EditLCA 菜单选 Net→EditNet,指令行显示:{Select net:}

7) 输入:"net8" <Enter>,指令行显示:{Select PIPs}

3. 连接起始腿和负载腿

现在利用菜单选择 PIP,以使布线连接起始腿和负载腿。

1) 在第二个 PIP 上把指针放到腿 FA.X 的右边,并按动鼠标左键,打开 PIP。

2) 沿直线向下移动指针直到 PIP 的顶部,即到另一条加亮线,在这个 PIP 上按动鼠标,这条网线被布好。

3) 从下拉菜单选择 Done,或按鼠标的中间键,结束 EditNet 指令,

当把指针放到网线的终点腿(腿 FB.D),信息行显示从起点到此腿的延迟,如果在 EditNet 中把指针放到 FB.D,网线延迟显示一个问号标志,因为网线延迟在 EditNet 结束后才被计算。

4. 通过开关矩阵布线

在下一例子中,起始腿和负载腿不可能直接被连接,所以,必须通过开关矩阵来布线。

1) 在指令行输入:"find net 10" <Enter>

2) 由 EditLCA 菜单选择 Net→Hiligh,系统显示带彩色选项的上托窗口。

3) 由上托窗口选 Magenta,指令行显示:{Select net(s):}

4) 输入:"net 10" <Enter> <Enter>

第二个回车符后才返回到主菜单,系统以品红色加亮接此腿的互连线

5) 由 EditLCA 菜单选 Net→EditNet,指令行显示

{Select net:}

6) 输入:"net 10" <Enter>,指令行显示

{Select PIPs:}

对此网线布线的最好方式是通过到 C 和 D 之间和行 F 之上的开关矩阵

7) 在第一个 PIP 上按鼠标, 如图 10.19 所示。

8) 在第一个腿上按鼠标, 如图 10.20 所示。

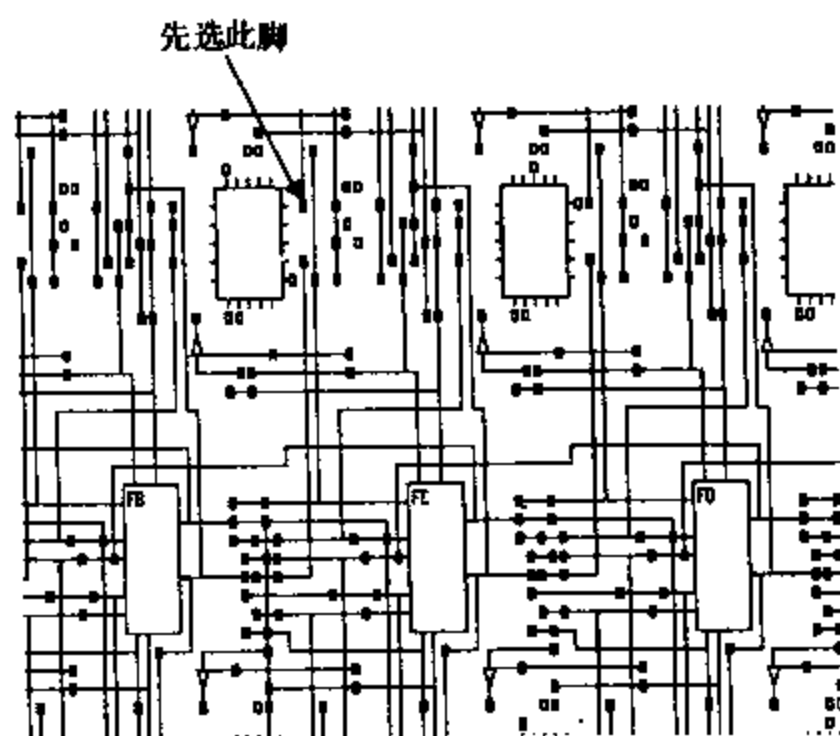


图 10.19 人工布线——步骤 1

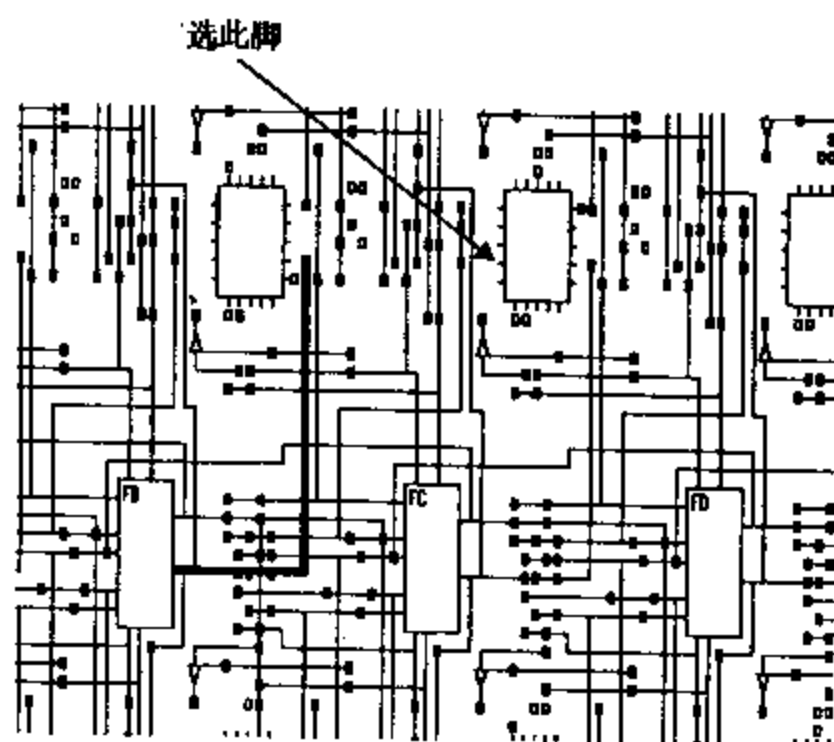


图 10.20 人工布线——步骤 2

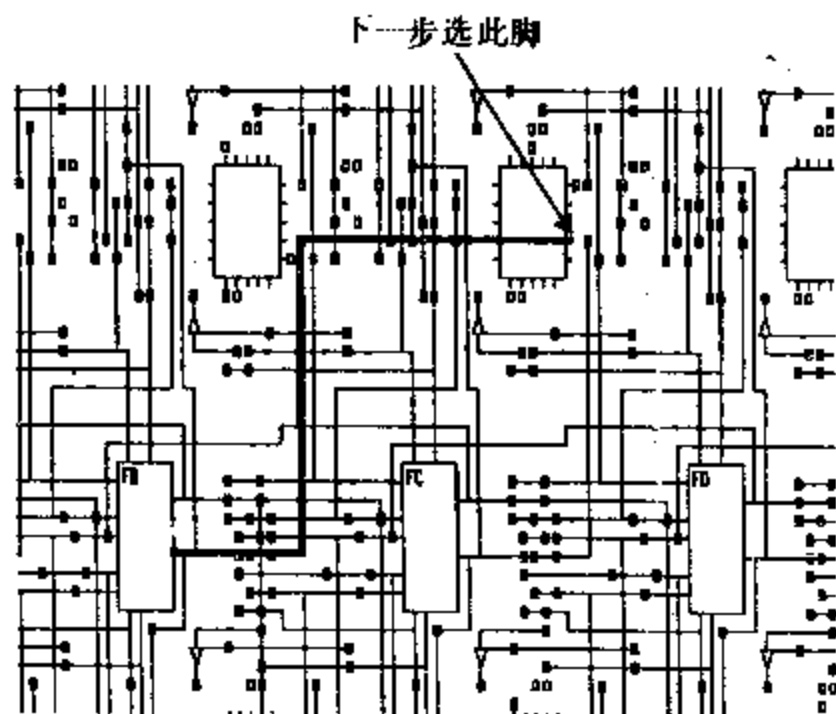


图10.21 人工布线——步骤3

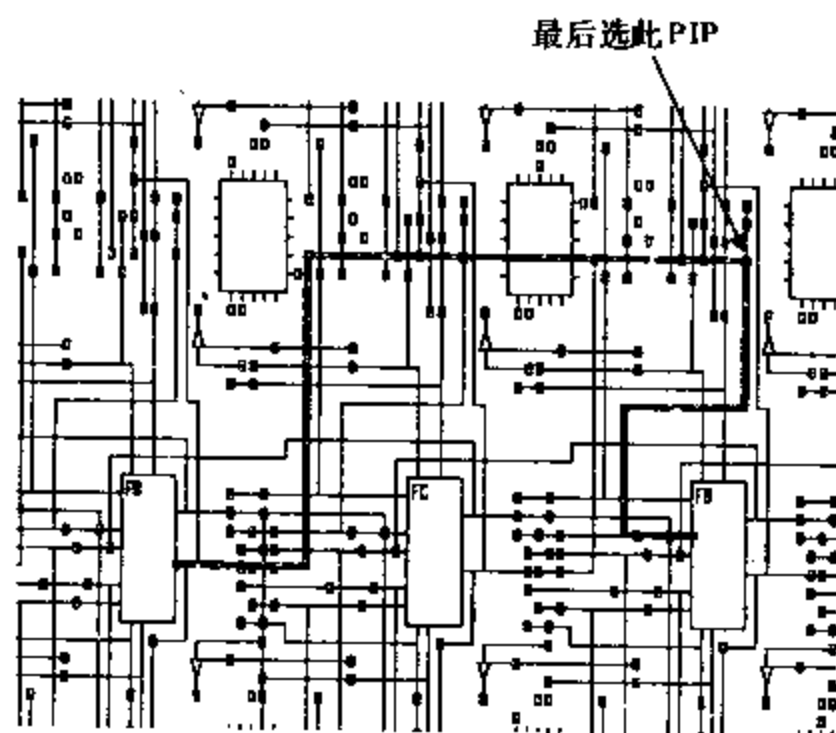


图10.22 人工布线——步骤4

9)检查指针的状态行,确认是在正确的出腿上,然后按鼠标左键,指令行显示:

{select other pin:}

10)选择开关矩阵腿9,如图10.21所示。

11)最后,在 PIP 上按鼠标,加亮线直接到开关矩阵出腿9的右边,如图10.22所示。

网线现在被布好,图10.22表示 net10的完全的布线通道。

§ 10.3.6 通过开关矩阵布线

当通过一个开关矩阵布线时,可能需要利用一个不是合法连接输入的输出腿,在此情况下,可以利用交织排(bank shot)通过中间腿,即输入和输出被有效连接,来连接这些腿,在以下的例子中,连接此被布线段的唯一方法是利用 交织排,因为没有方法直接连接网线的被布线段,如图10.23所示。

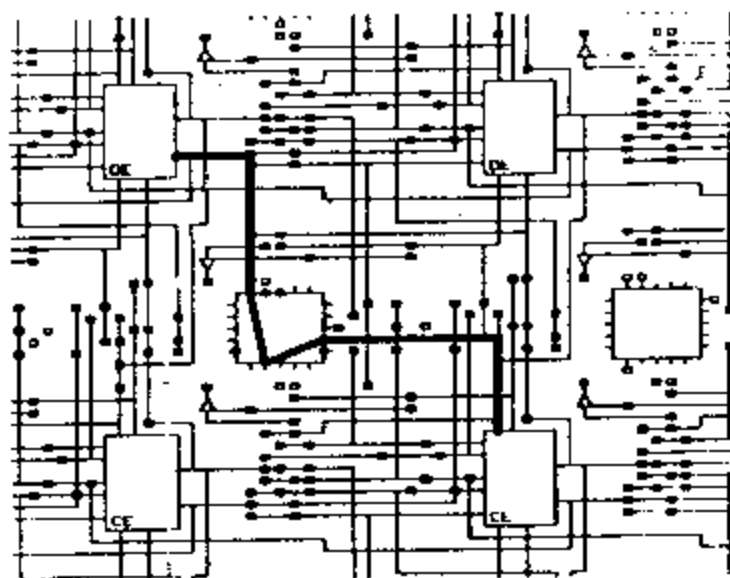


图10.23 开关矩阵交织排

- 1)重画一次屏幕,移去其它的加亮线,由主菜单选 Screen→Redraw
- 2)在指令行键入:"find af" <Enter>
- 3)加亮被编辑的文本:"hilight" <Enter>4)选择 Magenta,状态行显示 {Select net(s);}
- 5)输入:"net 13" <Enter>
- 6)选择 Done 返回到主菜单
- 7)由主菜单选择 Net→EditNet,状态行显示: {Select net;}
- 8)输入:"net 13" <Enter>
- 9)在开关矩阵.BH. 20,在腿19,14,14和2按顺序按动鼠标。
- 10)选择 Done 结束 EditNet 指令,网线现在通过交织排连接。

§ 10.3.7 利用长线布线

以下的例子说明如何利用长线对网线布线,为了有效地利用长线,CLB 在一列中排列,所以仅需要一根长线,利用 Swapsig 指令排列负载腿,以使在所有的 CLB 中利用相同的腿,按此方式,可以直接连接这些输入到长线上,不需要通用互连,.B 腿是移动负载腿到其上的最好选择,因为大多数的输入是来自这个腿。

1. 交换功能块:

- 1)在指令行键入:"find net 15" <Enter>
- 2)重排功能块,以使它们排成一列,由主菜单选择 Blk→SwapBlk
指令行显示: {Select block 1;}
- 3)输入:"bi" <Enter>,指令行显示: {Select block 2;}
- 4)输入:"bj" <Enter>

2. 交换出腿

- 1)移到功能块 CJ
- 2)为了利用单根长线,从 C 腿交换 net15到 B 腿,在指令行键入:

"Swapsig cj. c cj. b" <Enter>

3)为功能块 DJ 和 JJ 把 net15移到 B 腿,在指令行输入:

"Swapsig dj. c dj. b" <Enter>

"Swapsig jj. a jj. b" <Enter>

4)此网线末布线,所以可以在指令行输入以下指令用长线对它布线:

"unroute net 15" <Enter>

5)加亮此网线,可以容易地找到出腿。

"highlight magenta net 15" <Enter>

3. 对网线布线

为了利用长线对此网线布线,执行以下步骤:

1)利用 EditNet,如下对此网线布线:

"editnet net 15" <Enter>

指令行显示:{Select PIP(s): }

2)转到 I/O 腿 P76,在 P76.1底下第一个 PIP 上按鼠标,它是加亮的出腿。

3)向左移动,在第二个 PIP 上到此 PIP 的右边,对加亮线按动鼠标,PIP 驱动此长线。

4)向下移动,在加亮线段上选择每个 PIP,在到达 FJ 功能块时停止。

5)对 FJ. B 在长线的 PIP 上按鼠标,状态行显示以下信息:

{net net 15;No sources reachable}

这个警告信息表明刚刚打开的 PIP 没有连接到源腿上,如前所述,长线仅仅是芯片长度的一半,这个半路点在 E 和 F 两行之间,为了连接两半长度,必须接通分离器的 PIP。

在 CLB 的 EJ 和 FJ 之间,在开关矩阵顶部是没有线段与它们相接的一对相邻的 PIP,这些是 B 和 C 长线的分离器的 PIP。

6)按鼠标左键一次,两半的长线现在接通。

7)连续向下移动,把剩下来布线的负载腿连接好。

4. 执行 Query net

执行 Querynet 确保没有丢失任何负载腿。

1)在指令行,输入:"Querynet net 15" <Enter>

系统显示列出所有负载腿的上拉窗口,三个显号(* * *)表示未连的负载腿。

2)返回到原理图,连接丢失的腿。

§ 10.4 执行类似功能的指令

这一节解决执行类似功能的指令之间的差别。

§ 10.4.1 Unroute 和 UnroutePin

从网上移去所有的布线利用 Unroute,从被选的腿上移去布线,利用 UnroutePin,请执行以下的步骤:

1. 输入:"find net 15" <Enter>

2. 改变网线颜色:"Colornet" <Enter>

3. 由上托窗口选择颜色或在指令行直接输入:

"magenta" <Enter>

系统显示如下: {Select net (s):}

4. 在此情况下,输入指定的网线:"net 15" <Enter>

5. 练习 UnroutePin 指令,如下:

"UnroutePin" <Enter>

"BJ. B CJ. B" <Enter>

6. 试验 Unroute 指令如下:

"unroute net15" <Enter>

§ 10.4.2 Route 和 RoutePin

Route 配置整个网线的互连,RoutePin 仅仅连接被选腿到网线的源,执行以下步骤:

1. 输入:"find net 26" <Enter>

2. 改变网线的颜色:"hilight" <Enter>

3. 由上托窗口选择颜色,或在指令行直接输入。

"magenta" <Enter>

系统显示如下: {select net(s):}

4. 在此情况下,输入指定的网线:"net 26" <Enter>

5. 为了布线单个加载腿,如下利用 RoutePin 指令

"routePin cf. a" <Enter>

注意:在源腿上执行 RoutePin,布线所有加载的腿,它等效于规定"Route net26"

6. 为用规定源腿布线所有加载腿,如下输入:

"routePin cd. x" <Enter>

7. 布线所有加载腿,输入:route net 26 <Enter>

8. 为不布线此网线的的所有加载腿,输入:

"unroute net 26" <Enter>

§ 10.4.3 Route, EditNet 和 RoutePint

这三个指令之间的差别如下:

1. Route 指令利用自动布线器拾取整个布线通道,不对所取通道的控制

2. EditNet, 指令要求人工地选择整个通道,必须打开每一个 PIP,选择每个开关矩阵上的腿。

3. Route Point 允许选择中间点,自动布线器在这些点之间布线,这个指令提供与控制 EditNet 组合的 Route 指令的速度。

§ 10.4.4 Hilight 和 ColorNet

Hilight 不仅彩色地加亮被布线的互连,而且也加亮接到布线线段的未用短段。

ColorNet 仅改变包含布线的部分互连。

1. 在指令行输入:

"highlight magenta net 6" <Enter>

"Colornet magenta net 2" <Enter>

Hilight 画出整个一条网线(慢速和暂时),ColorNet 分配新的颜色于网线(快速和永久)。如果一条网线是未布线的,Hilight 加颜色于出腿段;但是,ColorNet 似乎不做什么事情,因为它只加颜色于已布线的网线。可是当网线确实被布线后,它有用 ColorNet 语句赋予它的颜色。

2. 输入:"unroute net2 net 6" <Enter>

3. 如下利用 Hilight 加颜色于已布线的互连线:

"highlight magenta net6" <Enter>

4. 利用 ColorNet 改变包含布线的互连线:

"colornet brightblue net2" <Enter>

5. 布线此网线:

"route net2" <Enter>

§ 10.5 建立四位乘法器

通过人工设计实现一个 4×4 位组合的乘法器,说明利用 XDE 的设计流程,如图10.24所示。

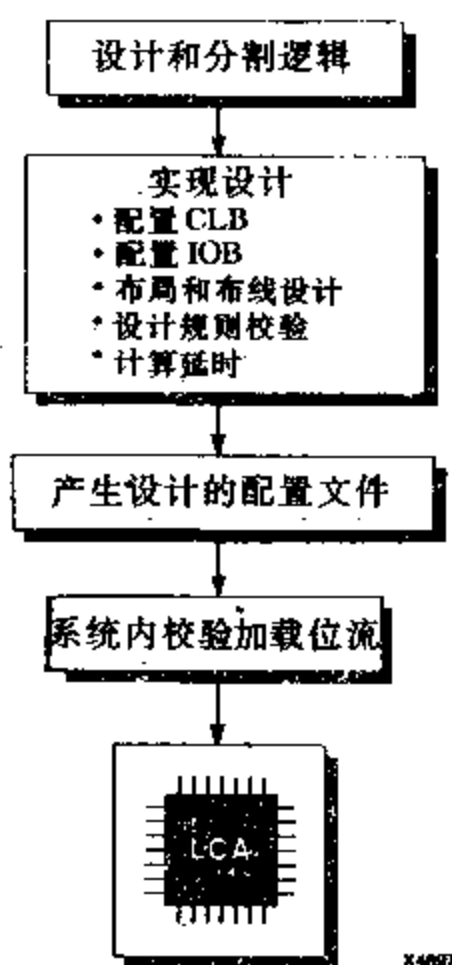


图10.24 用 XDE 设计 FPGA 的流程图

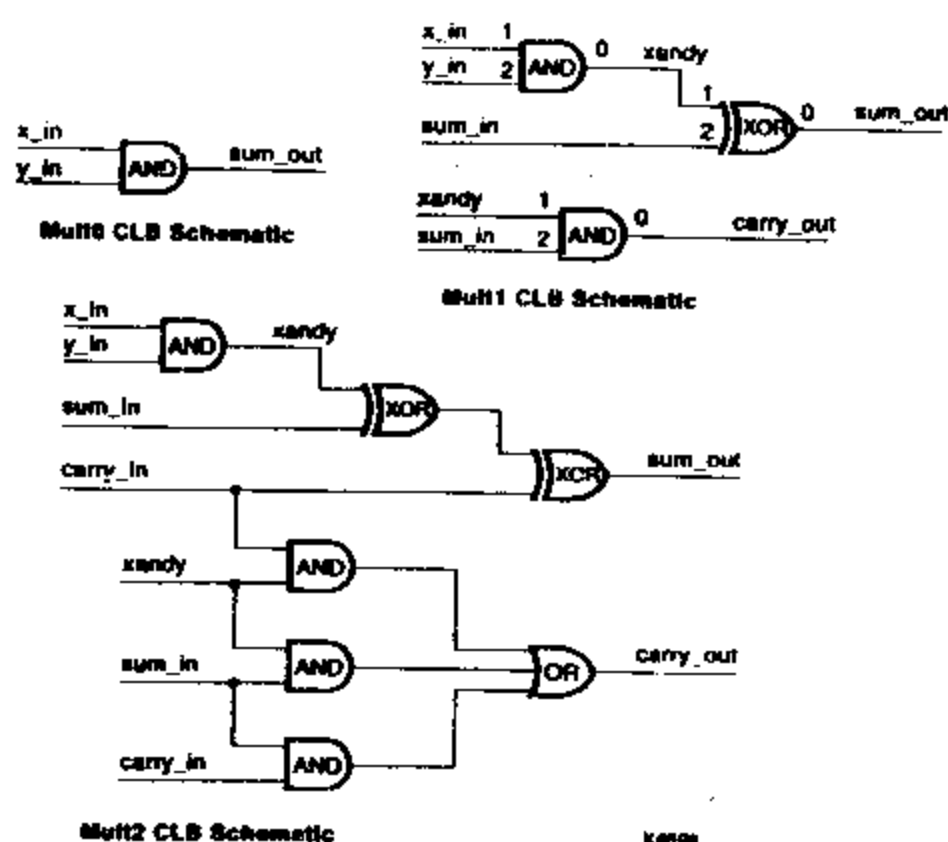


图10.25 乘法器单元的原理图

在图10.26组合乘法器的方框图中,表示了 CLB 的布局,它们的块名,每个 CLB 应被配置三种方式的哪一种,输入用 X 和 Y 标记,输出有 Z 的前缀。

组合乘法器利用一个 CLB 为每个部分积,一个两输入与门产生每个部分积,但要求相加电路把相等权重的所有部分积加在一起,图10.25给出 CLB 的三个不同类型的原理图,已

被命名为 mult0,mult1,mult2来构成乘法器,mult2的 CLB 产生部分积,把两个输入值相加,产生两个输出,Sum 和 Carry.Sum 作为部分积有相同的二进制权重,Carry 有下一级较高的权重,在左边和顶部的功能块有较少的输入,是较简单的,但执行相同的基本的功能。

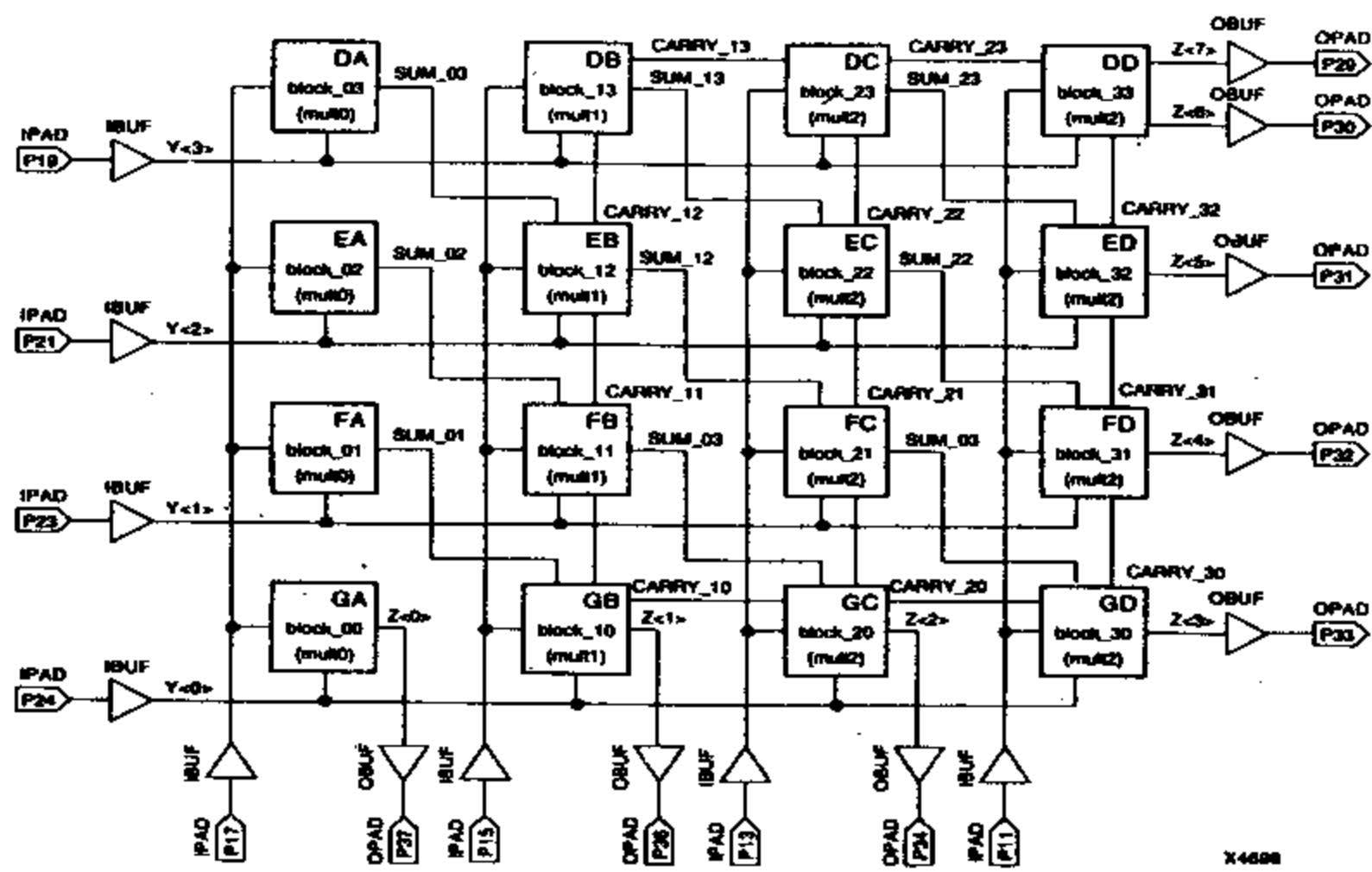


图10.26 组合乘法器的方框图

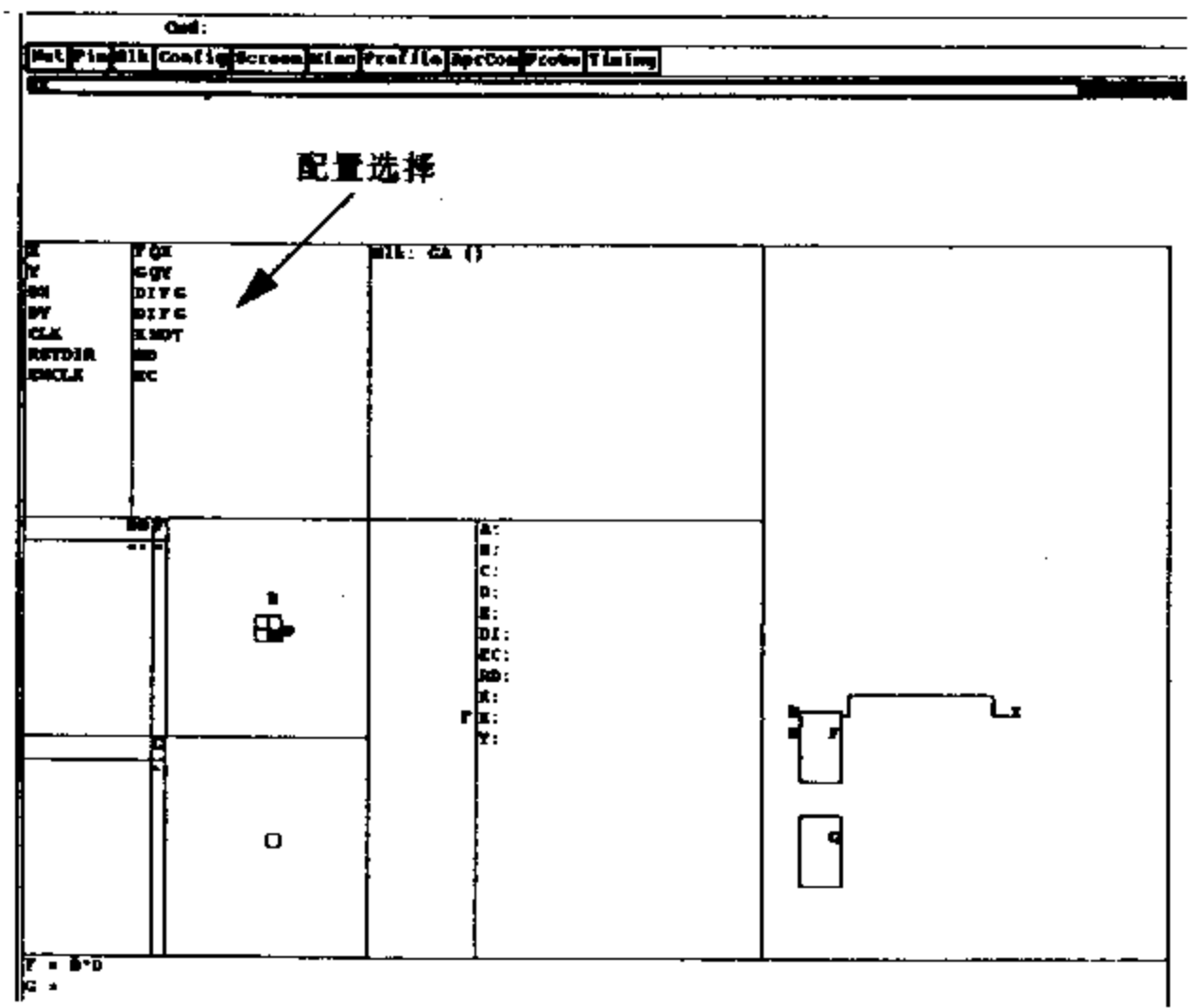


图10.27 CLB配置为mult0

一、产生新的设计

为退出 EditLCA,并准备编辑新的设计,执行以下的步骤:

1. 键入: "Exit" <Enter>, 返回 XDM 屏幕, 由主菜单选 Design→Part, 改变器件类型为 3020pc68;
2. 编辑称为 mult4 的新设计, 键入 "editlca new mult4" <Enter>;
3. 对上托菜单的提问, 选 Yes, 放弃已有的改变, 系统重画屏幕。

二、产生 mult0 CLB

1. 键入 "editblk ga" $\langle$ Enter $\rangle$, 以图 10.27 作指导配置功能块 GA;
2. 在指令行输入方程, "f=b*d" $\langle$ Enter $\rangle$;
3. 从配置选择区选 F 为 X, 功能块 GA 被配置, F 函数发生器把腿 B 和 D 输入的信号相与;
4. 选 "Switch", 从 editblk 屏幕返回到 editor 屏幕;
5. 由 Editblk 屏幕选 Blk $\rightarrow$ CopyBlk, 状态行显示 {From block:}, 输入要拷贝的功能块, 如 "GA" $\langle$ Enter $\rangle$;
6. 状态行显示 {To block(s):}, 在指令行键入: "fa" $\langle$ Enter $\rangle$, "ea" $\langle$ Enter $\rangle$, "da" $\langle$ Enter $\rangle$; 选 Done。

三、产生 mult1 CLB

1. 键入"editblk gb" <Enter>,在指令行输入以下方程:"f=c@b*d"和"g=c*b*d";
2. 从配置选择区选 F 为 X 和 G 为 Y,确认功能块类似于图10.28,

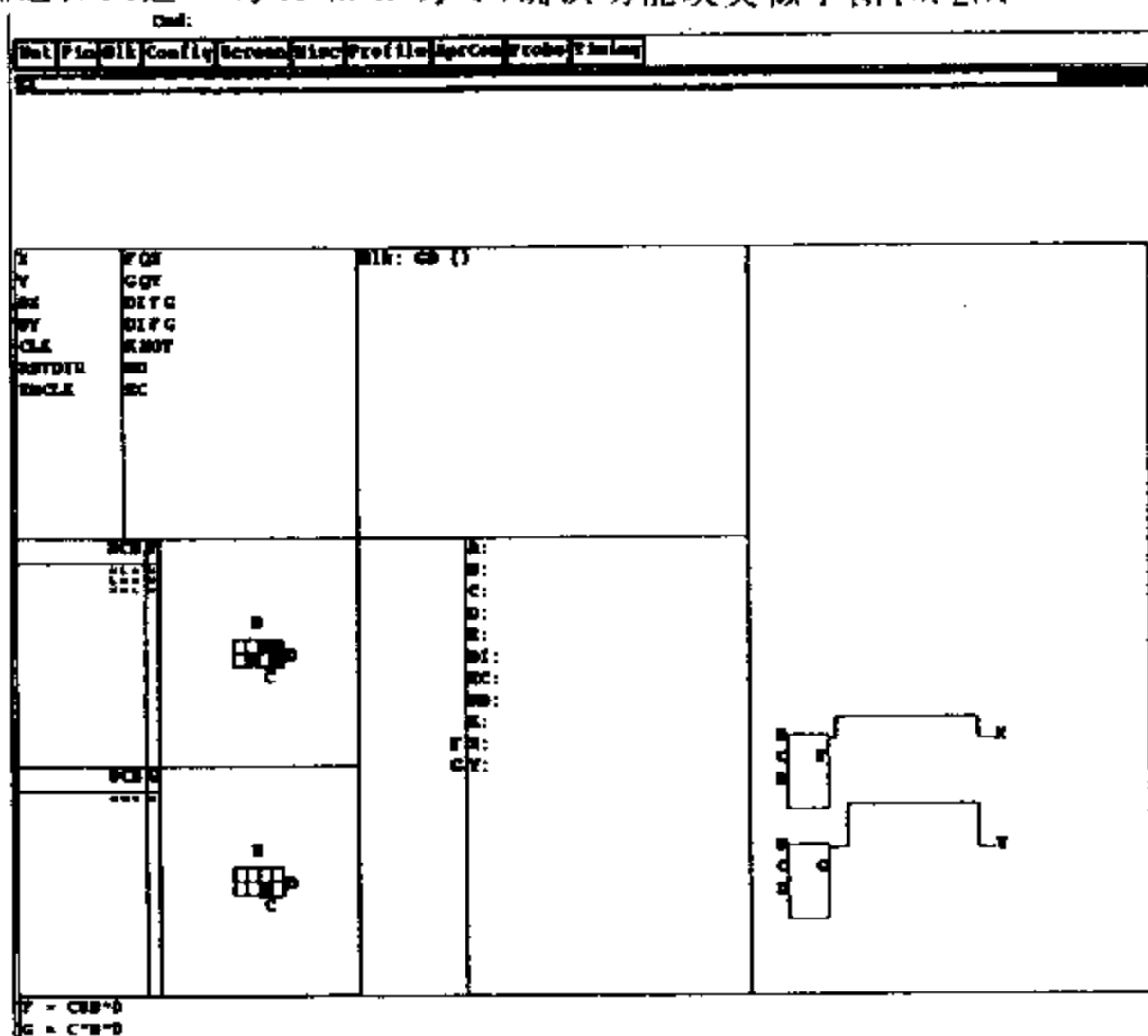


图10.28 CLB 配置为 mult]

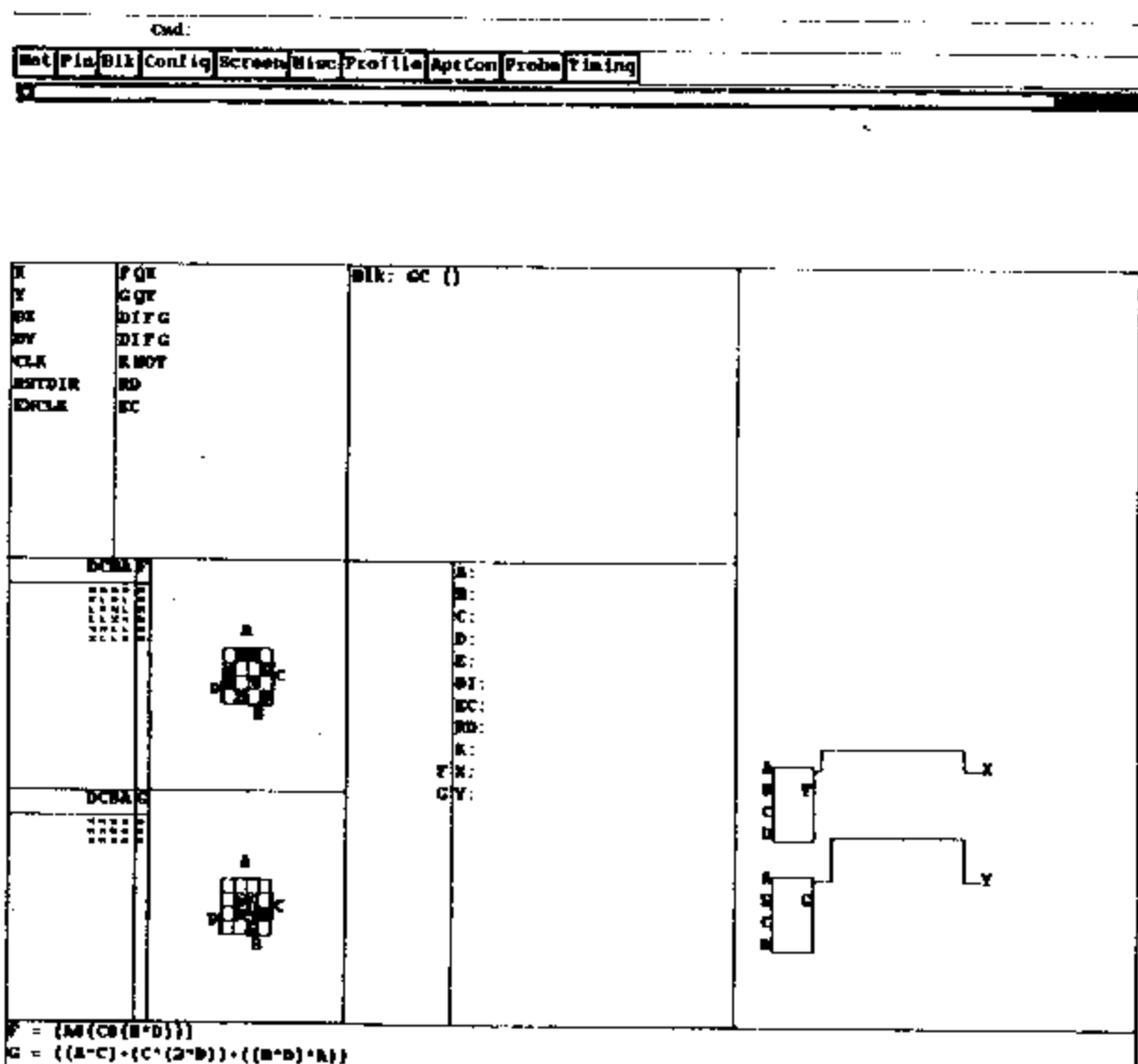


图10.29 CLB 配置为 mult2

3. 为了拷贝 mult1到其它利用它的位置,选 Switch,返回 EditLCA 屏幕,键入"copyblk" <Enter>,"gb" <Enter>,"fb" <Enter>,"eb" <Enter>,"db" <Enter>,选 Done;

四、产生 mult2 CLB

1. 键入"editblk gc" <Enter>,"f=(a@(c@(b*d)))"和"g=((a*c)(c*(b*d))+((b*d)*a));",
2. 从配置选择区选 F 为 X 和 G 为 Y,确认功能块类似于图10.29;
3. 选 Switch,返回 EditLCA 屏幕,拷贝 GC 到其它利用它的位置,键入"copyblk gc fc ec dc gd fd ed dd" <Enter>。

五、配置 IOB 为输入腿

1. 键入"editblk p24" <Enter>,配置功能块 P24为输入腿,在 LATCH 和 PULLUP 标记上按鼠标关闭它们,在 I 标记上按鼠标打开它;
2. 确认功能块 P24类似于图10.30,拷贝输入腿到其它利用的位置,键入"copyblk p24 p23 p21p19 p17 p15 p13 p11" <Enter>。

六、配置 IOB 为输出腿

1. 键入"editblk p37" <Enter>,配置功能块 P37为输出腿,在 I,LATCH 和 PULLUP 标记上按鼠标关闭它们,在 O 标记上按鼠标打开它;

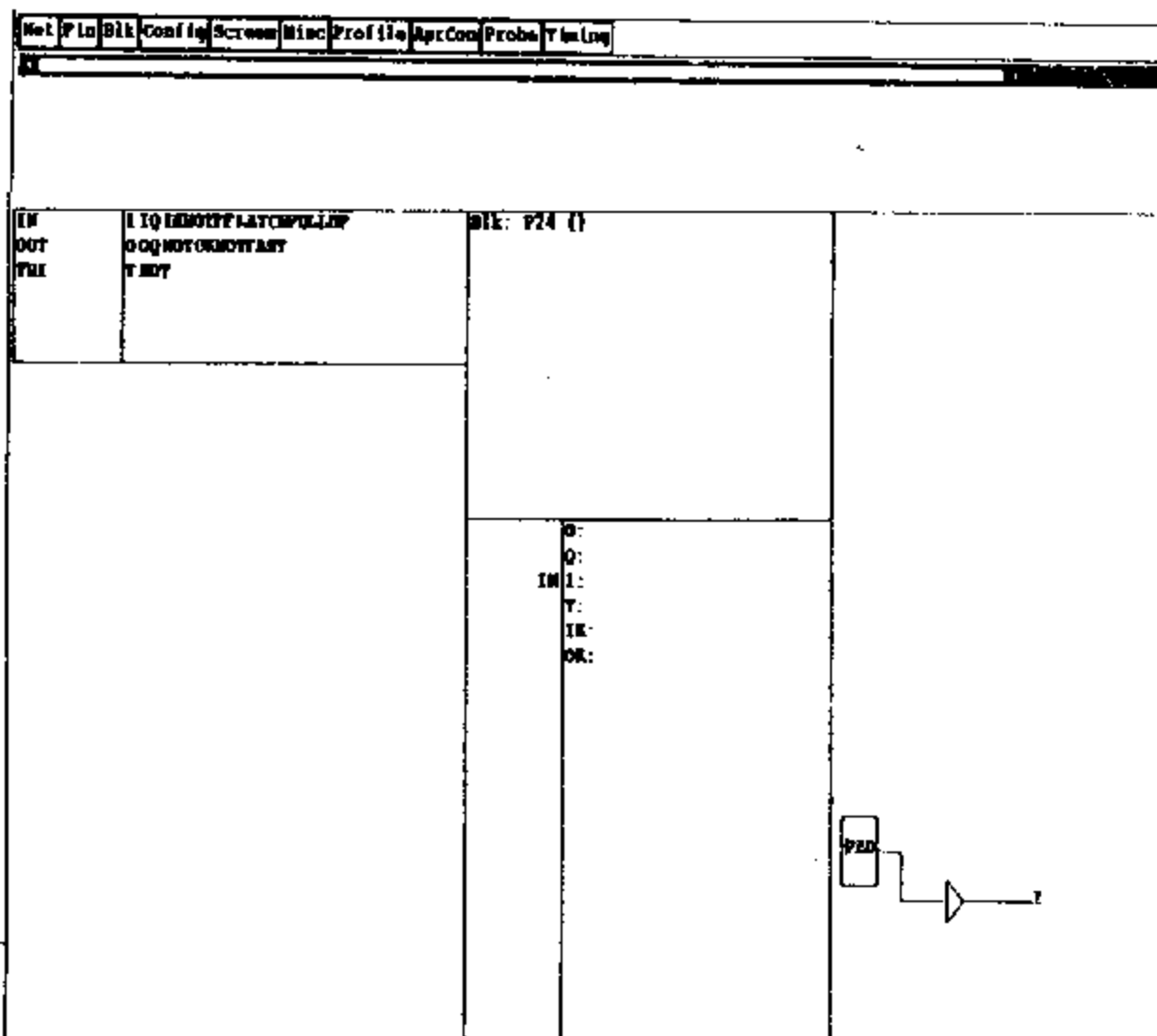


图10.30 IOB 配置为输入

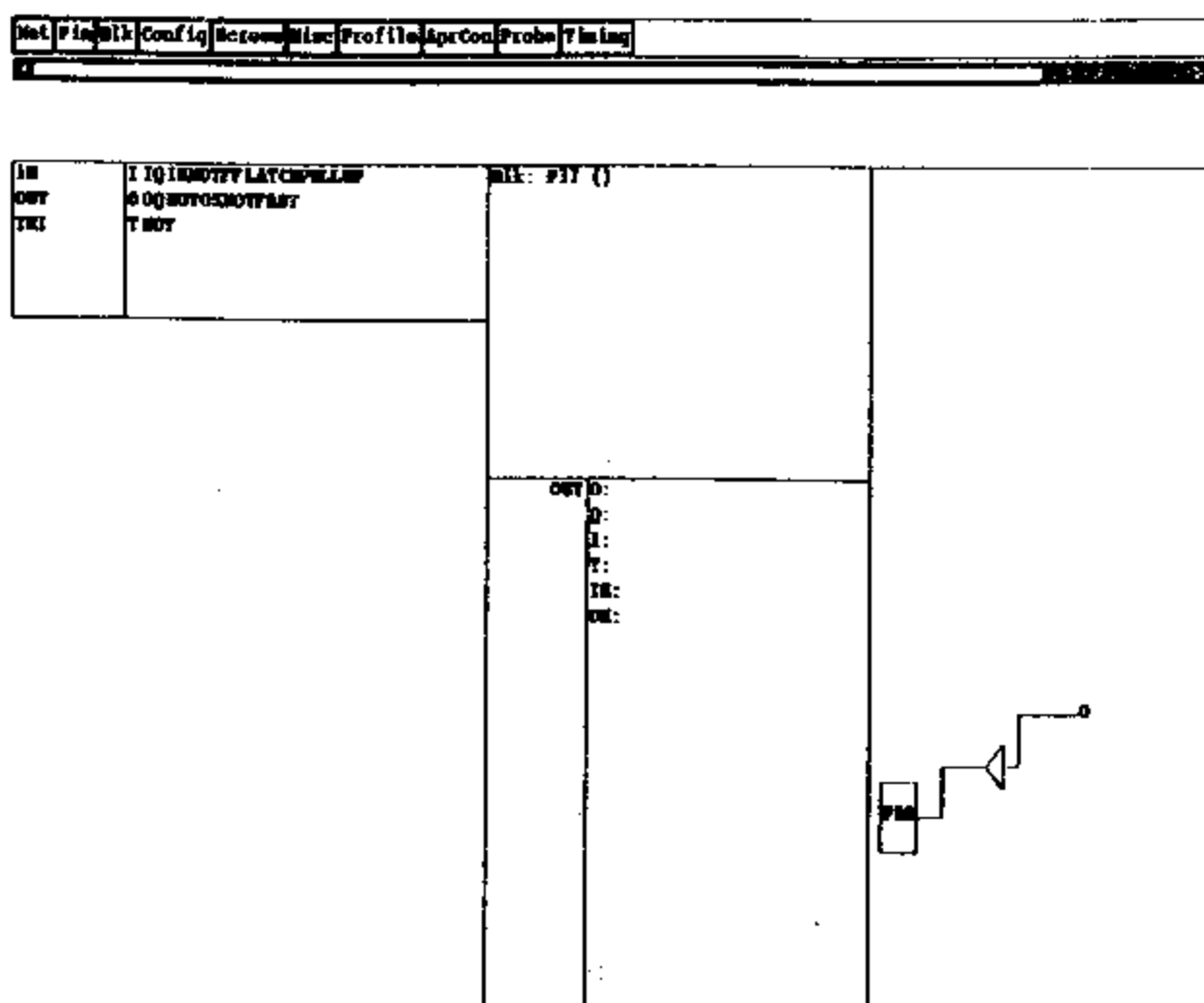


图10.31 IOB 配置为输出

2. 确认功能块 P37 类似于图 10.31, 拷贝输出腿到其它利用的位置, 键入 "copyblk p37 p36 p34 p33 p32 p31 p30 p29" <Enter>。

3. 保存设计的修改至缺省的 edit_lca.lca, 键入: "save" <Enter>。

七、给功能块逻辑名

1. 键入 "nameblk p24 yo_in" <Enter>, 或产生包含一组 NameBlk 语句的批作业文件;

2. 按照表10.4命名 I/O 块和 CLB, 在指令行键入 "exec names" <Enter>, 系统命名所有 I/OB 和 CLB。

可以把 XDE 指令的任何清单放进一个文件, 并用 Exec 指令执行它们, 可加快速度。

表10.4 IOB 和 CLB 的名字

IOB	NAME	CLB	NAME
P23	Y1-IN	GA	BLK-00
P21	Y2-IN	FA	BLK-01
P19	Y3-IN	EA	BLK-02
P17	X0-IN	DA	BLK-03
P15	X1-IN	GB	BLK-10
P13	X2-IN	FB	BLK-11
P11	X3-IN	EB	BLK-12
P37	Z7-OUT	DB	BLK-13
P36	Z6-OUT	GC	BLK-20
P34	Z5-OUT	FC	BLK-21
P33	Z4-OUT	EC	BLK-22
P32	Z3-OUT	DC	BLK-23
P31	Z2-OUT	GD	BLK-30
P30	Z1-OUT	FD	BLK-31
P29	Z0-OUT	ED	BLK-32
		DD	BLK-33

八、加网线于设计

1. 确认自动布线器是关闭的, 选 Profile → Autoroute, 由上托菜单选 off;

2. 如果仍在 EditBlk 模式下, 选 Switch 返回到 EditLCA 屏幕, 由菜单选 Net → AddNet;

3. XACT 提示输入新的网线名, 在指令行键入 "x<0>" <Enter>;

4. 移鼠标到以下每个出腿位置, 在其上按鼠标左键, 加网线到这些腿:

p17. i GA. d FA. d EA. d DA. d

5. 选 Done, 完成加网线到这些腿, 键入以下指令加网线 X<1>:

"addnet x<1> p15. i gb. d fb. d eb. d db. d" <Enter>

当此指令出现在图中, 用鼠标选单个出腿, 完成加网线, 或利用称为 net 的指令文件, 执行:

"exec nets" <Enter>

图10.31表示乘法器所有出腿的连线。

6. 选 Net → QueryNet, 在上托窗口选 — All → Done;

系统显示 queryNet 的结果, 将其列出的清单与图10.32的网线连接信息相比较, 如有错误或规则检验出现警告, 用 DelNet 指令去除有错的网线, 并重新加此网线。

NET NAME	SOURCE PIN	LOAD PIN(S)
X<0>	P17.I	GA.D FA.D EA.D DA.D
X<1>	15.I	GB.D FB.D EB.D DB.D
X<2>	13.I	GC.D FC.D EC.D KC.D
X<3>	11.I	GD.D FD.D ED.D DB.D
Y<0>	24.I	GA.B GB.B GC.B GD.B
Y<1>	23.I	FA.B FB.B FC.B FD.B
Y<2>	21.I	EA.B EB.B EC.B ED.B
Y<3>	19.I	DA.B DB.B DC.B DD.B
Z<0>	GA.X	29.O
Z<1>	GB.X	30.O
Z<2>	GC.X	31.O
Z<3>	GD.X	32.O
Z<4>	FD.X	33.O
Z<5>	ED.X	34.O
Z<6>	DD.X	36.O
Z<7>	DD.Y	37.O
SUM_01	FA.X	GB.C
SUM_02	EA.X	FB.C
SUM_03	DA.X	EB.C
SUM_11	FB.X	GC.C
SUM_12	EB.X	FC.C
SUM_13	DB.X	EC.C
SUM_21	FC.X	GD.C
SUM_22	EC.X	FD.C
SUM_23	DC.X	ED.C
CARRY_10	G.Y	GC.A
CARRY_11	F.Y	FC.A
CARRY_12	E.Y	DB.C
CARRY_13	D.Y	DC.C
CARRY_20	G.Y	GD.A
CARRY_21	F.Y	EC.A
CARRY_22	E.Y	DC.A
CARRY_23	D.Y	DD.C
CARRY_30	G.Y	FD.A
CARRY_31	F.Y	ED.A
CARRY_32	E.Y	DD.A

图10.32 网线连接信息

九、对网线布线

1. 到此,设计功能上已正确,唯一剩下的事情是网线的布线,在指令行键入"route *"<Enter>;

2. XDE 用自动布线器布线所有的网线,由菜单选 Misc→Drc→Done;

设计规则检验(DRC)检查所有的布线和功能块配置,确保设计中无致命的错误,出现错误或警告,返回去改正过来。

3. 选 Timing→XDelay,由上托窗口选 Analyze,得到最坏条件的时序延时。

十、改善布线

1. 键入"unroute *"<Enter>,出现对话框:

Unroute 36 nets, Are you sure?

2. 选 Yes,由 EditLCA 选 Pin→AlignSig;

3. 选第一条网线,在指令行键入"x<0>"<Enter>,指令行显示{Select pin?};
4. 键入"c"<Enter>,显示{Select Block(s):},键入"*a"<Enter>,选 Yes;
XDE 把网线 X<0>放在 A 列的所有 CLB 的 C 腿上的,一根长线可驱动这些 CLB。
5. 在指令行键入以下指令,为专门的网线交换腿:

"alignsig x<1> c * b" <Enter>

"alignsig x<2> c * c" <Enter>

"alignsig x<3> c * d" <Enter>

"alignsig y<0> a g * " <Enter>

"alignsig y<1> a f * " <Enter>

"alignsig y<2> a e * " <Enter>

"alignsig y<3> a d * " <Enter>

则 X<0:3>和 Y<0:3>布线到长线上,网线没有完全布完,打开负载接到长线的 PIP,不管出现的警告信息{No sources reachable},关闭 DRC,键入"autodrc off"<Enter>;

6. 参考图10.16,找出可用直接连线来改善布线的地方:

Carry_10用 X—to—B 直接连接; Carry_22用 Y—to—D 直接连接;

找出其它可以用直接连接的网线,用 SwapSig 把网线放到正确的腿,如果直接连线可以用而没有用,布线资源被浪费。

7. 键入"route * " <Enter>,对设计布线,找出最坏条件的延时,选 Timing→ XDelay,由上托窗口选 Analyze,由于布线改善,最坏条件的时延应降低。

十、产生位流文件加载芯片

1. 用缺省名保存设计. 键入"save" <Enter>, <Enter>;

2. 返回执行屏幕,键入"exit" <enter>;

3. 由 XDE 执行菜单选 Programs→ MakeBits;

4. 由 MakeBits 菜单选 Config→ MakeBits→ Done;

设计变换成可加载到芯片的位流文件;

5. 选 Config→ Writebits <Enter>,使用了缺省文件名;

6. 将加载的 Download 电缆接到演示板和 PC 机并口,演示板开关2,3,和4接到断开的位置,FPGA 为从属配置模式;

7. 初始化并口,由 Makebits 菜单选 Misc→ Port,由上托窗口选 LPT1;

8. 加载位流文件到芯片,由 Makebits 菜单选 Download→ Download;

9. 演示板上按 reset 键,然后 PC 机上按<Enter>键,屏幕出现:

'done' signal now high

指令行键入"quit"<Enter>,退出 Makebits 屏幕。再键入"quit"<Enter>,退出 XDE。

第十一章 XEPLD 的设计方法

§ 11.1 行为设计概述

本节说明将使读者很快理解如何利用 XEPLD 开发一个行为设计,包括一个简短的行为设计例子,说明自动 PAL 变换的过程。

§ 11.1.1 行为设计方法概述

行为设计利用文本式的语言而不是电路图定义逻辑电路的功能,利用 XEPLD,一个行为设计可以作为顶层文件被产生,它包括所有设计的控制信息和行为方程,亦可以用层次的格式开发设计,它包括在一个或多个包含文件(Include File)中的设计方程,通过顶层文件(Top-Level File)链接在一起。行为设计的流程如图11.1所示。

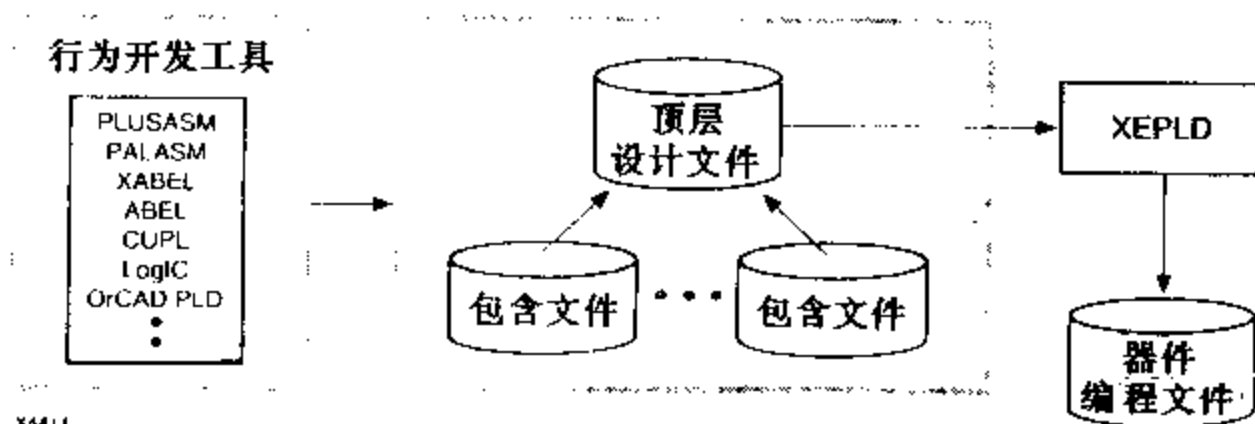


图11.1基本的行为设计流程

下面介绍产生行为设计的顶层文件和包含文件的方法。

变换已有的 PAL 设计：

XEPLD 允许自动地利用已有的 PAL 方程文件作为 XEPLD 行为设计的包含文件,这个设计方法有几个优点：

1. 可以利用已有的 PLD 编译工具,利用自己熟悉的编译工具来编辑自己的设计。
2. 可以把已有的 PAL 设计和用 PLUSASM 产生的设计或用 PLD 编译器产生的设计组合起来。

§ 11.1.2 利用 PLUSASM 产生新的设计

PLUSASM 是 Xilinx 的 XEPLD 开发语言,提供了对全部器件特性完全的存取,利用布尔方程输入,并提供以下的能力：

1. 控制 EPLD 分割, 逻辑布局, 出腿;
2. 存取到 EPLD 的结构和快速进位特性;
3. 存取到包括时钟使能(CE)的输入焊盘寄存器和锁存器;
4. 自动地支持十分宽的与门(支持无限数的宏单元反馈和器件腿的输入);
5. 支持快速输入(FastInput) 信号到高速功能块(旁路通用互连矩阵);
6. 自动支持高速输出使能(FOE)控制;
7. 支持 ALU 功能(如相加器);
8. 支持对器材件的速度和功率的控制。

§ 11.1.3 利用 Xilinx ABEL 产生新设计

Xilinx ABEL EPLD/FPGA 编译器提供开发行为设计的高级语言的能力, Xilinx ABEL 可以产生顶层文件和包含文件, 也可以产生为原理图设计方法所用的单独的文件, Xilinx ABEL 的特性:

1. 高层设计语言(HDL);
2. 状态机设计的能力;
3. 支持在原理图设计中的 PLD 文件;
4. 支持多模块设计;
5. 支持功能仿真。

§ 11.1.4 利用第三方的 PLD 编译器

任何能产生 PLUSASM 兼容的输出(或 PALASM)的 PLD 编译器可以用来产生 XE-PLD 设计的包含文件, 这些 PLD 编译器(如(ABEL, ORCAD PLD, CUPL, LOG/iC 等等)可以提供各种特性包括:

1. 支持较宽范围的器件类型(PAL 和 EPLD);
2. 高级的设计语言支持(HDL);
3. 状态机语言(SML);
4. 总线和矢量运算;
5. 功能仿真;
6. 真值表输入。

当利用 PLD 编译器(不是 Xilinx ABEL)时, 重要的是只要可以产生按器件无关格式的方程式文件或瞄准支持 PAL 类型(20V8或22V10)技术条件之一, 这些与器件无关的文件则被用作包含文件, 也即, 瞄准一个或多个 PAL 器件的已有设计和按 PLUSASM 兼容格式(如 PALASM)输出的已有设计可以用作包含文件。

§ 11.1.5 变换 JEDEC 文件

XEPLD 可以拆开为22V10和20V8PAL 的 JEDEC 文件, 可以利用最终的文件作为 XE-PLD 行为设计的包含文件或作为包含在原理图设计中的 PLD 文件, 但是, 如果可用原始设计的源文件通常更容易工作。

§ 11.1.6 设计例子—变换 PAL 文件

本节提供一个行为设计例子,表示如何自动地变换已有的 PAL 设计在 Xilinx 的 EPLD 使用,它由原始为利用 22V10PAL 而设计的四位计数器组成,方程式文件如图 11.2。在可以执行此设计例子中包含的指令前,必须相应地安装 XDM 软件,请参考 DS—550 的软件安装指令。

```
TITLE COUNTER EXAMPLE FILE
CHIP COUNTER 22V10;
;Pins (not in pin number order)
HCLK REGWR SELECT COUNTEN OUTPUTEN DD3 DD2 DD1 DD0 CARRY COL3
GND COL2 COL1 COL0
;Nodes
LOAD HOLD COUNT
;COUNTER
EQUATIONS
LOAD = (REGWR * ^ /COUNTEN);
HOLD = (REGWR * COUNTEN
+ /REGWR * /COUNTEN);
COUNT = ^ (/REGWR * COUNTEN);
COL0 := (DD0 * LOAD
+ HOLD * COL0
+ COUNT * /COL0);
COL0.CLKF = (HCLK);
COL0.TRST = (SELECT * /OUTPUTEN);
COL1 := (DD1 * LOAD
+ HOLD * COL1
+ COUNT * /COL1 * COL0
+ COUNT * COL1 * /COL0);
COL1.CLKF = (HCLK);
COL1.TRST = (SELECT * /OUTPUTEN);
COL2 := (DD2 * LOAD
+ HOLD * COL2
+ COUNT * COL2 * /COL1
+ COUNT * /COL2 * COL1 * COL0
+ COUNT * COL2 * /COL0);
COL2.CLKF = (HCLK);
COL2.TRST = (SELECT * /OUTPUTEN);
COL3 := (DD3 * LOAD
+ HOLD * COL3
+ COUNT * COL3 * /COL2
+ COUNT * COL3 * /COL1
+ COUNT * /COL3 * COL2 * COL1 * COL0
+ COUNT * COL3 * /COL0);
COL3.CLKF = (HCLK);
COL3.TRST = (SELECT * /OUTPUTEN);
CARRY := (DD3 * DD2 * DD1 * DD0 * LOAD
+ /LOAD * COL3 * COL2 * COL1 * COL0);
CARRY.CLKF = (HCLK);
```

图 11.2 四位计数器方程文件(Counter. PLD)

PAL 变换过程:

第一步:拷贝 PAL 文件

为了进行变换,在此例中利用的 PAL 方程文件已经产生,文件名是“Counter. PLD”,它在 Xilinx 软件的 XACT/Tutorial 目录中。

拷贝此文件到工作目录中:

第二步:运行 XDM

由 DOS 的提示,键入 XDM。

出现 XDM 的主菜单,利用鼠标加亮菜单的选项,用鼠标左键选择。

第三步,选器件系列,类型和速度

由 XDM 菜单,选 PROFILE→FAMILY,

XDM 显示器件系列的选择菜单,选“XC7300”;

然后 XDM 显示元件类型选择菜单,选“7354PC44”;

然后 XDM 显示速度类型选择菜单,选“-10”;

屏幕左下角的 XDM 主菜单“Family:”和“Part:”选项现在改变为新的选择。

可以直接在这些区域按动鼠标改变这些选择。

第四步:变换 PAL 方程文件

1) 利用鼠标选 FITTER→PALCONVT

屏幕出现输入设计名字的提示:{Enter design file name (pld):}。

2) 输入设计名字:“n_design”

此设计名字为整个这个例子使用,PALCONVT 显示在工作目录中所有 PAL 文件。

3) 选“counter. PLD”,在“DONE”上按鼠标键

屏幕出现“Cancel”和“DONE”选项和上托菜单。

4) 在“DONE”上按鼠标键

(缺省选择是“产生新的 PLD 和 PAL 互连报告”)。

PALCONVT 自动地产生设计的顶层文件,包含 PAL 方程式(Counter. PLD)作为包含文件。图11.2说明此顶层文件。

第五步:编译此设计

1) 利用鼠标选择 FITTER→FITEQN

XEPLD 显示 FITEQN—POPUP 菜单。

2) 选“n_design. PLD”,在“DONE”按鼠标键 XEPLD 显示选项选择的上托菜单。

3) 在“DONE”按鼠标键

缺省选项是“不管引脚分配”(ignore pin assignments)。XEPLD 处理设计文件,检验错误,产生设计报告,被编译的设计命名为“n_design. VMH”。

第六步:校验此设计

利用鼠标选择 UTILITIES→BROWSE。

可以浏览以下关键的报告,校验所做的设计。

1) n_design. ERR 由 XEPLD 产生的所有警告和错误信息在此报告中列出。

2) n_design. INT 给出每个信号名及其如何使用的 PAL 互连报告,利用此报告校验每个信号相应的连接。

3) n_design. RES 器件逻辑和出腿资源使用报告,利用此报告校验设计被放进具有合适数量资源的器件中。

第七步:编程器件

1) XDM 产生编程的位映射文件

VERIFY→MAKEPRG(为 Intel Hex 格式)。

VERIFY→MAKEJED(为 JEDEC 格式)。

2) 在上托菜单中选择“n_design. VMH”,在“DONE”项上按鼠标键,出现输入标志的提示。

3) 输入“testprg. a”

a 的扩展名为 ASC I 的识别标志,XEPLD 产生名为“n_design. JED”或“n_design. PRG”的位映射文件,包含把设计编程到所选器件所要求的信息。

在编程器上加载位映射文件。

§ 11.2 用 PLUSASM 产生设计

本节介绍如何利用 PLUSASM 产生一个行为设计,PLUSASM 是依据 PLUSASM2 方程句法的 Xilinx 的 EPLD 设计语言。

由 PLD 编辑器或原理图输入工具来的所有文件,在选配过程前用 XEPLD 软件变换到 PLUSASM,选配器可以优化 PLUSASM 的码,达到极优的结果,或者可以产生优化的 PLUSASM 源码,准确地规定选配器应该如何在高级设计文件中利用属性和特性语句分配资源。

§ 11.2.1 PLUSASM 概述

利用 ASC I 文本编辑器,在单个文件内可以产生完全的 PLUSASM 设计,它包括设计控制信息和行为方程两部分,但是常常为了方便,作为分别的包含文件(Include File)来开发设计,再把它在顶层文件(Top—Level File)中链接在一起形成完整的设计,如图11.3所示。

1. 文件结构

PLUSASM 顶层文件和包含文件的基本结构如图11.4所示。这个图表示层次设计由包括两个插入方程式和由 INCLUDE—EQN 语句引入的三个包含文件的顶层文件组成。

文件结构被识别为顶层文件和包含文件两种,但是包含文件有不同的 CHIP 语句句法,它们仅可包含一个说明语句的有限子集。

2. 标题段

标题段仅仅用于设计的文档,这些指令 XEPLD 是不管的,也不影响设计,标题可以按任何次序包含以下指令:

TITLE, AUTHOR, DATE, REVISION, TIME, COMPANY

3. 说明段:

说明语句规定器件的 I/O 腿,影响到行为方程如何映射到规定的器件中,在标题段之后第一个语句必须是由出腿清单直接跟随的 CHIP 语句,其它所有说明语句使用次序可任意。

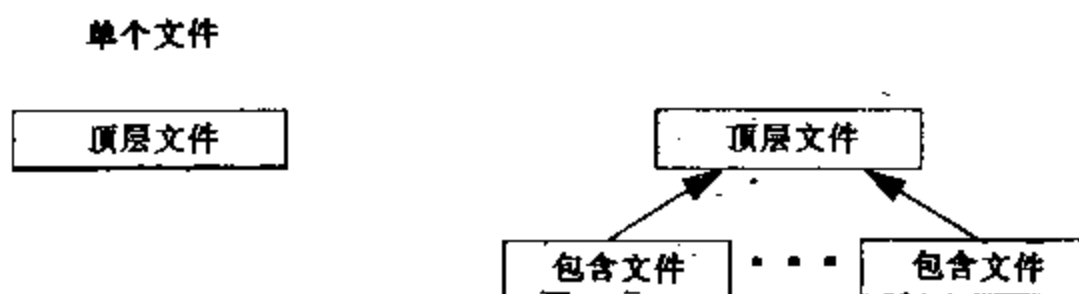


图11.3 PLUSASM 设计方法

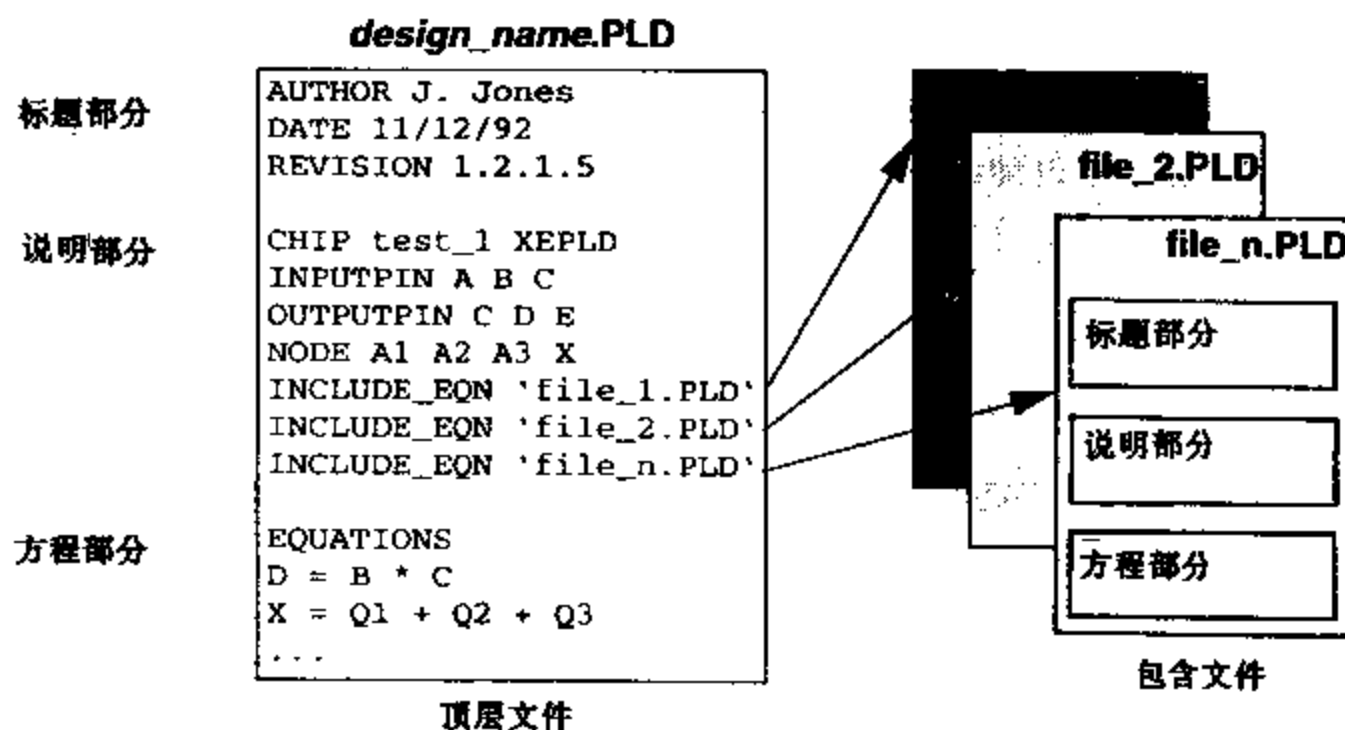


图11.4 PLUSASM 文件结构

说明语句	顶层	包含	功能
CEPIN	✓		规定全局时钟使能输入腿
CHIP	✓	✓	规定文件类型, 文件名字, 出腿清单
FASTCLOCK	✓	✓	规定全局 FastCLK 输入
FASTINPUT		✓	说明接到 FastInput 腿的信号
FOEPIN	✓		规定全局“快输出使能”输入腿
INCLUDE-EQN	✓		规定包含文件的名字
INPUTPIN	✓		规定器件输入腿
IOPEN	✓		规定器件 I/O 腿
LOGIC-OPT	✓		控制逻辑优化
MINIMIZE	✓	✓	控制逻辑最小化的使用
MRINPUT	✓		规定主复位输入
NODE	✓	✓	规定设计中的节点
OPTIONS	✓		控制器件资源的自动使用
OUTPUTPIN	✓		规定器件输出腿
PARTITION	✓	✓	规定成组方程的物理位置
PWR	✓		控制器件功率利用
STRING	✓	✓	规定全局文本字符串

4. 方程段

在方程段中规定行为设计方程, 它必须用关键词 EQUATIONS 开始, 可以利用任何有

效的 PLUSASM 方程的句法。

在顶层文件中,这一段是可选的,如果有在包含文件中规定的方程。但是,即使没有明显的方程在文件中,也必须包含 EQUATIONS 的关键词。

§ 11.2.2 产生基本的设计

Xilinx 的 EPLD 选配器自动地优化方程文件,它分析设计,并自动地映射函数到相应的器件资源,不用用户介入,选配器实现紧凑的高性能设计,用到了先进的 XC7000 结构特性的优点,如输入焊盘寄存器,通用互连矩阵,双功能块结构的功能块,全局控制信号等。

这节的内容是介绍 XC7000 结构的各种特性,并说明如何用 PLUSASM 控制器件资源的分配。

在典型的 EPLD 设计中使用五个基本的结构,这节将讨论如何对以下五个基本结构开发优化的方程:

输入焊盘;高密功能块;通用互连矩阵;高速功能块;输出焊盘。

1. 设计例子

以下的脉冲宽度调制器(PWM)的设计例子是利用来说明这节讨论的基本设计概念。

图11.5说明了这个设计的逻辑,以及它如何映射到基本的 XC7000 结构块中,对此原理图的方程式文件表示在图11.6。

PWM 的输出是可变高时间的固定频率,用可加载4位加减计数器控制,如果计数器用数值 P 加载,减计数周期的长度是 P 个时钟周期,加计数周期的长度是 $(2n+1)-P$,输出触发器的触发周期是 $2n+1$ (加计数和减计数周期的和),它是与 P 无关的,所以触发器输出有它的高时间正比于 P 的固定频率。

2. 利用输入焊盘结构

在 XC7000 系列器件中有两类输入焊盘:专用只输入焊盘和配置为输入的 I/O 焊盘。

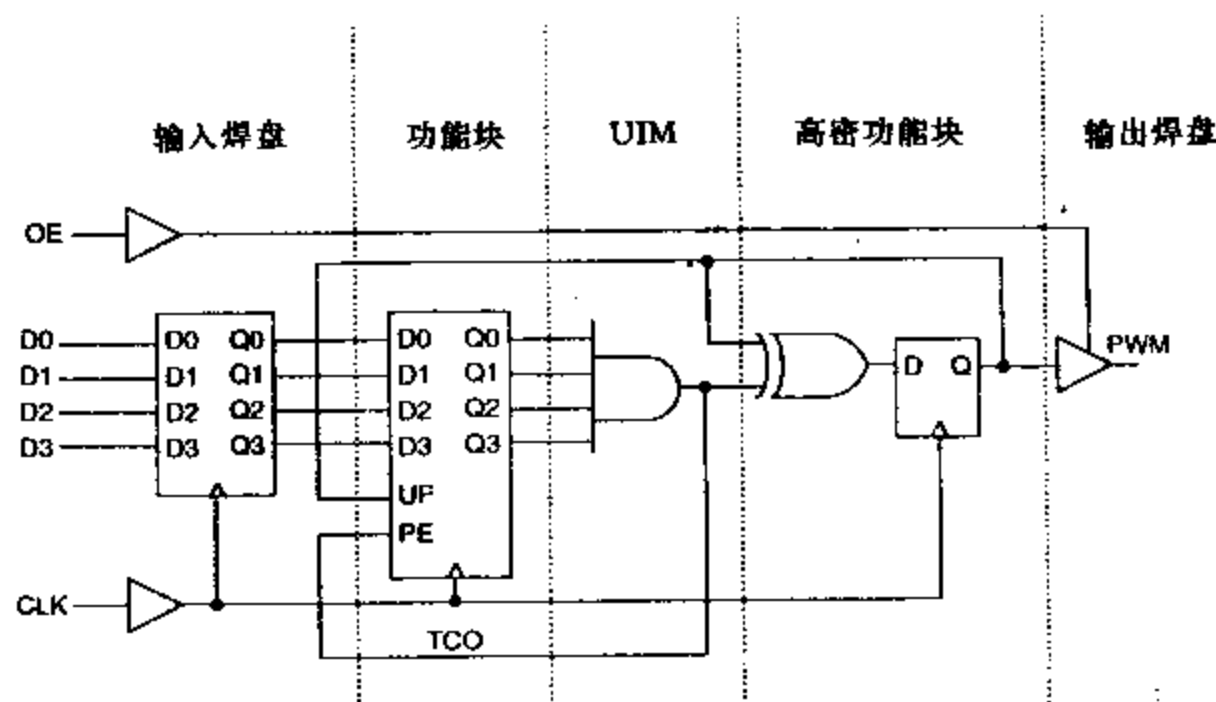


图11.5 四位脉宽调制器

```

TITLE    NEWPWM
CHIP     NEWPWM  XEPLD

; pulse width modulation design

OUTPUTPIN          q0 q1 q2 q3
OUTPUTPIN (FOE = OE)  pwm
INPUTPIN (RCLK=CLK)  d0 d1 d2 d3
NODE (UIM)          tco
FASTCLOCK           clk
FOEPIN              OE

PARTITION FFB          pwm

EQUATIONS

q3 := (q3 * /tco) :+: (d3 * tco
                      + pwm * q0 * q1 * q2 * /tco
                      + /pwm * /q0 * /q1 * /q2 * /tco)

q2 := (q2 * /tco) :+: (d2 * tco
                      + pwm * q0 * q1 * /tco
                      + /pwm * /q0 * /q1 * /tco)

q1 := (q1 * /tco) :+: (d1 * tco
                      + pwm * q0 * /tco
                      + /pwm * /q0 * /tco)

q0 := (q0 * /tco) :+: (d0
                      + /tco)
pwm := pwm :+: tco
tco = (q0 * q1 * q2 * q3)

```

图11.6 脉宽调制器文件

输入焊盘可以如下配置：

寄存的；由时钟使能寄存，锁存，组合的（无寄存，无锁存）和快输入。

1) 寄存输入：加一个时钟源修饰（RCLK=clock-source）到 INPUTPIN 语句来规定寄存输入。例如，对图11.6的输入焊盘部分在方程文件有：

```
INPUTPIN (RCLK=CLK) d0 d1 d2 d3
```

```
FASTCLOCK CLK
```

注意：输入焊盘寄存器仅可以用 FastCLK 输入来定时。

软件自动地把分配的单个输入寄存器方程接到输入焊盘寄存器，以减小可能的宏单元资源的使用。

2) 时钟使能寄存输入：加时钟使能（CE）修饰到 INPUTPIN 语句来规定带有时钟使能的寄存输入。

```
FASTCLOCK CLK
```

```
CEPIN/CKEN
```

INPUTPIN (RCLK=CLK CE=CKEN)D0, D1, D2, D3

3) 锁存输入:利用 INPUTPIN 语句中的锁存修饰(LE)配置输入焊盘为锁存运用。

FASTCLOCK ENAB

INPUTPIN (LE=ENAB)D0, D1, D2, D3.

4) 组合输入:在 INPUTPIN 语句中省去时钟源规定配置输入焊盘直接到 UIM 输入。

如 INPUTPIND0 D1 D2 D3

5) 快输入:每个 XC7300系列器件有专用的快输入腿,输入信号直接到高速功能块,旁路 UIM,所以要求很少延迟,例如 XC7300器件有12个专用 FastInput. 在 INPUTPIN 语句中利用(FI)修饰来规定 FastInput。

INPUTPIN (FI) D0 D1 D2 D3

上例中所有的 D0, D1, D2和 D3都用作 FastInput,也可以用.FI 扩展来规定要求用快输入通道的信号,而其它的信号通过 UIM 布线。

INPUTPIN D0.FI D1 D2 D3

6) 输入极性:在腿的指标前面加“/”配置或低有效输入。

例:INPUTPIN(RCLK=CLK) D0 /D1 D2 D3

在 INPUTPIN 语句中变换信号名字有效地变换在方程中输入的每个信号。

3. 利用高密功能块

XC7300系列器件有组合的高密功能块和高速功能块,被称为“双功能块”结构,高密功能块提供最大数量的设计可用的逻辑资源,它可以实现两种方程:寄存的方程,组合的方程。

1) 寄存的方程:规定寄存的方程用“:=”算子

例:EQUATIONS

q3:=(q3 */tco);+:(d3 * tco+PWM * q0 * q1 * q2 */tco+PWM * q0 * q1 * q2 */tco)

利用.CLKF 方程对寄存方程规定时钟源,时钟可以是设计中任何原始的输入或逻辑信号(宏单元反馈),它可以利用宏单元乘积项时钟来实现,也可以规定 FastCLK 信号,它是在 FASTCLOCK 语句中被说明的,如果不规定.CLKF 方程,软件利用在 FASTCLOCK 语句中命名的第一个信号。

注意:软件自动地分配单个上升沿时钟信号到可能的 FastCLK。

在高密功能块中的寄存器可以配置成上电后或为逻辑0,或为逻辑1,利用.PRLD 方程规定 GND 或 Vcc 作为电源的值。

Q1.PRLD=Vcc; preload the Q1 output to a logic 1.

如果不规定预加载值,软件通常假设为逻辑零(GND),但是当需要时将改变它允许寄存器优化成输入焊盘或 FFB。如果规定一个预加载值,可以禁止软件对方程执行优化。

用.RSTF 扩展名配置非同步清除的寄存方程。

Qn. SETF=CLEAR

用.RSTF 扩展名配置非同步清除的寄存方程。

Qn. SETF=SET

用.RSTF 扩展名产生(乘积项)三态控制信号

X. TRST=TRISTASE

也可以用全局 FUEPIN 信号控制三态信号,软件自动化分配单输入高有效.. TRST 方程到可能的全局 FOE 信号。

2) 组合的方程:利用“=”算子规定组合方程,例如:

$tco = q0 * q1 * q2 * q3$

利用 PARTITION 语句人工地放置一个或多个方程到任何高密功能块或规定的高密功能块,甚至规定的宏单元。

PARTITION FB Signal—name1 Signal—name2

PARTITION FB5 Signal—name1 Signal—name2

PARTITION FB5—3 Signal—name1 Signal—name2

注意:软件自动地分割设计,人工分割可以防止软件产生设计的最有效映射。

4. 利用通用互连矩阵(UIM)

UIM 提供允许任何输出驱动器件中任何输入的100%的互连,布线从不阻塞,UIM 的线与能力也允许它实现无附加延迟的逻辑函数。

1) 利用 UIM 互连

除去 FastInput,所有宏单元的输入都来自 UIM,不要求专门的句法来规定这些在方程式暗示的连接。

2) 利用线与功能

UIM 有能力执行线与功能,在可能改善资源利用率时它自动地被软件所使用,为了人工规定线与功能,利用 NODE 语句分配附加的结点到 UIM,在脉冲宽度调制的例子中,进位输出信号是由 UIM 中 Q0—Q3 输出相与产生的。

对图11.5的 UIM 部分方程文件为:

NODE (UIM) TCO

EQUATIONS

$tco = q0 * q1 * q2 * q3$

在大多数情况不需要在设计中人工地规定 UIM 的与门,如果软件允许选择进入 UIM 的函数,结果通常更好。

XEPLD 自动地利用 UIM,任何时候可能增加有效的宏单元资源,可以利用 OPTIONS 语句为整个设计,去掉 UIM 的优化功能。

OPTIONS OFF UIM—OPT; Turn off UIM optimization

变换 UIM 与函数的输入或输出可以产生摩根等效逻辑函数,(OR, NOR, NAND 等等)

例如在 UIM 中产生逻辑 OR 函数:

NODE(UIM) OR_GATE; place or_gate into the UIM

EQUATIONS

$/OR_GATE = /Q0 * /Q1 * /Q2 * /Q3$; a logic or function

注意:当试图放置一个由于缺少信号的倒相而不能实现的方程放进 UIM 中时,方程被自动地放进宏单元。

5. 利用高速功能块

如前所述,XC7000系列器件有组合的高速功能块和高速的功能块,放进高速功能块的逻辑执行起来比放进高密功能块的逻辑要快。

高速功能块可以执行组合的方程和寄存的方程两种。

1) 组合的方程:

利用“=”算符规定组合方程,这是与高密功能块相同的。例如:

$TCO = q0 * q1 * q2 * q3$

2) 寄存的方程

用“:=”算符规定寄存的方程,例如,对图11.5的高速功能块的寄存的方程为:

$pwm := pwm + tco$

注意:在高速功能块中对寄存器的预加载值是 V_{cc} (高),除非利用复位输入到寄存器,在利用复位输入时(仅在 XC7336中有效)预加载值是 GND(低)。

3) 方程的自动布局

在优化之后, XEPLD 自动地映射满足以下要求的函数到有效的高速功能块

① 所有时钟必须映射到全局 FastCLK 信号。

② 所有三态控制必须能映射到全局 FOE 信号。

③ 所有寄存器仅仅是非同步置位(无复位),除了 XC7336器件可选置位或复位。

④ 所有寄存器不必规定一个低的(GND)预加载值,除去 XC7336,如果在 XC7336中利用了复位输入到寄存器,预加载值必须是 GND。

⑤ 当用低有效输出实现时,所有逻辑必须利用4个或更少的乘积项,当用低有效输出实现时,XC7336器件例外,它支持低有效或高有效的输出。

注意:当方程要求多于4个的乘积项,XEPLD 软件自动地把方程分割成4的组中,并把输出给到利用 EXPORT 方程的相邻的宏单元,这个允许利用多于4个乘积项的方程在高速功能块中实现。

4) 方程的人工布局

利用 PARTITION 语句人工放置一个或多个方程到任意的或规定的高速功能块甚至到规定高速功能块的规定宏单元。

PARTITION FB1 signal_name1 signal_name2

PARTITION FB2 signal_name1 signal_name2

PARTITION FB2_3 signal_name1 signal_name2

注意:在 XC7300系列中,FB1和 FB2实际上是高速功能块,在 PARTITION 语句中规定 FB1或 FB2意味着利用一个高速功能块而不是高密功能块。

高速功能块没有 ALU 资源,所以,任何 D1或 D2方程当需要时,由软件变换到它们的积之和的形式,把方程选配到高速功能块中。

6. 利用输出焊盘结构

XC7300系列器件有两类输出焊盘,高密功能块输出有标准的驱动能力和高速功能块输出有高的驱动能力。这些输出可以配置为带有乘积项逻辑控制的三态;带有 FOEPIN 控制的三态;带有 FOEPIN 和乘积项控制的三态及直接输出。

1) 利用全局 FOE 的三态输出

在 OUTPUTPIN 语句中利用 FOE 修饰控制带有全局 FOEPIN 输入的三态输出:

OUTPUTPIN (FOE=OE) pwm

FOEPIN OE

2) 利用乘积项控制的三态

利用. TRST 扩展名控制带逻辑信号的三态输出。

OUTPUT; = tco

OUTPUT. TRST=X; X is the 3-state control

注意: 软件自动地分配单输入高有效的. TRST 函数到可能的全局 FOE 腿, 乘积项三态控制仅仅在高密功能块中有效; TRST 方程不可能对分配到高速功能块的输出作规定。

3) 直接输出: 利用 OUTPUTPIN 语句中不带有 FOE 修饰配置把输出配置为非三态的能力。

OUTPUTPIN D0 D1 D2 D3; outputs are not 3-state

4) 规定反馈通道: 有两类信号反馈到 UIM, 从器件腿反馈和从宏单元输出反馈。

在 IOPIN 语句中 PINFBK 选项可以用于全局控制反馈选项。

① 器件腿反馈: 利用 IOPIN 语句的 PINFBK 选项规定信号反馈总是来自器件腿。

IOPIN(PINFBK) A

Q; = A; A comes from the pin

② 宏单元输出反馈: 宏单元反馈通过 UIM 来布线, 并与有关的器件腿状态无关。

IOPIN A

Q0; = A; A comes from the macrocell—UIM feedback

Q1; = A. PIN; A comes from the pin—pin feedback

UIM 反馈除非其它的规定, 否则是缺省的。

5) 规定三态输出: 有两类乘积项三态控制, 三态腿和三态结点

① 三态腿: 当利用三态腿时, 乘积项仅控制腿的三态条件, 而不是到 UIM 的宏单元反馈。

OUTPUTPIN Q

Q; = A; Q is always fed back to the UIM

Q. TRST=X

除了 XC7272, 三态腿对所有器件是缺省的, 当上面的方程在 XC7272 中实现时, 当腿是在三态条件时, 到 UIM 的宏单元反馈失效。

② 三态结点: 利用三态结点来模仿 UIM 中三态总线, 但是, 实现多路开关代替三态总线通常要更好些, 因为当结点是在三态条件时, UIM 反馈是有损失的(在逻辑1状态)。

NODE (NODETRST) A B

NODE (UIM) AND_GATE

OUTPUTPIN C

EQUATIONS

A=SIGNA0; A=1 when x is not true

B=SIGNA1; B=1 when x is true

A. TRST=X

B. TRST=/X

AND_GATE=A * B; UIM AND gate

C=AND_GATE

③ 利用乘积项和 FOE 的三态控制

当对一个输出规定由乘积项和 FOEPIN 两个三态控制信号时,软件自动地把这些信号与在一起,两个控制信号必须是高有效来使能,当任一控制信号是无效的(为低),输出腿被迫在三态的条件下。

注意:这个三态控制与函数在器件中是固有的,不利用宏单元或 UIM-AND 资源。

§ 11.3 变换 PAL 设计

这一节说明如何利用 PAL 方程文件来开发 Xilinx 的 EPLD 设计,这是简单的过程,因为 Xilinx 的 EPLD 内部结构类似于 PAL,以及 XEPLD 软件自动地处理 PAL 方程文件,产生为编译器准备的顶层文件。图11.7说明 PAL 设计的变换过程。

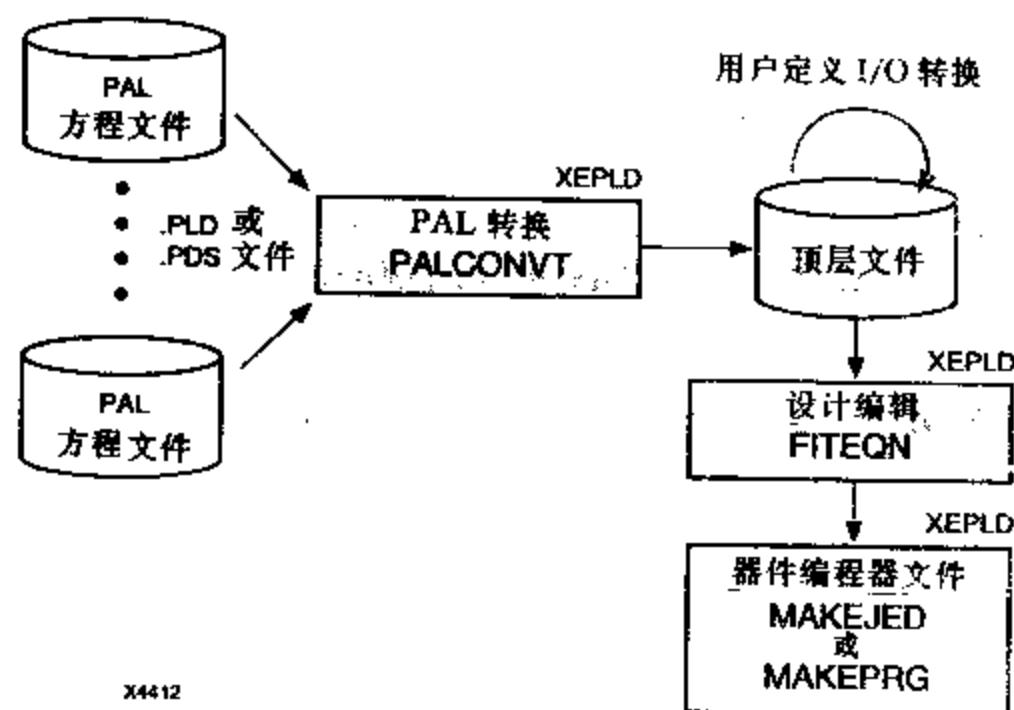


图11.7 PAL 设计的变换过程

因为 Xilinx EPLD 的自然语言是 PLUSASM,它是基于 PALASM2布尔方程句法的语言,许多通用 PAL 编译器,如 ABEL, CUPL, LOG/ic 和 PAISAM 可以产生 XEPLD 软件要求的 PALASM2布尔方程文件。利用 PAL 编译器内设的产生 .PDS 文件的能力(如 ABEL XFER 的使用或 CUPL-C 编译器选项),可以很容易产生 PLUSASM 兼容的方程文件,用 XEPLD 软件,这些方程文件则被映射到 Xilinx EPLD 的结构。

§ 11.3.1 PAL 变换方法概述

利用 FITTER→PALCONVT 指令把多个 PAL 方程组合进单个相关的 PLUSASM 设计,PALCONVT 程序读入 PAL 文件,分析其 I/O 信号配置,产生 PLUSASM 顶层文件,它规定了所有信号的注释,.PAL 方程文件变成在顶层文件中的包含文件。

PALCONVT 亦解决了 PAL 之间的任何极性冲突,解决任何 PAL 特定的功能(对 22V10和20V8),并产生一个 PAL 互连报告,摘要在设计中和在 EPLD 器件的 I/O 接口中所有信号的使用和互连,这个报告帮助选择匹配设计要求的目标器件。

1. 腿的结点的分配

PALCONVT 自动地根据 I/O 腿和结点在电路中的用途把它们的说明语句分配到 PAL 方程文件的信号上,这些说明被写到顶层文件。

- * 仅作为方程输入所用的信号被分配到 INPUTPIN 语句;
- * 仅作为方程输出所用的信号被分配到 OUTPUTPIN 语句;
- * 作为方程输入和输出二者用,又没有 .TRST 控制输入的信号分配到 NODE 语句;
- * 作为输出亦有 .TRST 控制输入所用的信号被分配到 OUTPUTPIN 语句,不管它们是否作为输入用。

在运行 PALCONVT 之后,修改在顶层文件中的说明语句可以改变任何这些分配的结点到输出或 I/O 腿,缺省分配在表11.1中说明。

表11.1 PAL 信号的自动分配

PAL 信号	缺省分配
仅出现在方程右边的信号	INPUTPIN
仅出现在方程左边的信号	OUTPUTPIN
出现在方程左右两边且无 .TRST 控制的信号	NODE
用作输出边有 .TRST 控制的信号	OUTPUTPIN

PALCONVT 亦处理并没有瞄准 PAL 的 PLUSASM 文件,这个包括在原理图设计中为 PLD 或定制符号准备的任何方程文件,如果方程文件包括 XEPLD 规定的说明语名,它们自动地分配为表11.2的项。

表11.2 XEPLD 说明语句分配

在包含文件中 XEPLD 特定的说明	分配
FASTCLOCK	FASTCLOCK
NODETRSTON	OUTPUTPIN(NODETRST)
NOD(UIM)	NODE(UIM)
FASTINPUT	INPUTPIN(F1)
PARTITION	PARTITION

2. FASTCLOCK 和 FOEPIN 分配

XEPLD 自动地把大多数经常利用上升沿的时钟输入分配到 FastCLK 网线,把大多数经常使用的高有效输出使能的输入分配到 FOE 网线。

注意:如果设计中时钟或输出使能控制被接到组合逻辑,它们不可以分配到 FastCLK 或 FOE 信号,这两信号在器件中利用专门的走线,对宏单元逻辑是不可存取的。

3. 信号极性的冲突解

PALCONVT 自动地修改方程来解决在信号极性说明中的冲突,软件自动地变换在 PAL 文件中一个信号的所有样例,如果此信号在一个 PAL 文件的出腿清单中用低有效极性来说明,但是在另一个 PAL 文件的出腿清单中被说明高有效。

§ 11.3.2 PAL 变换要求

可以利用任何符合 PLUSASM 句法(.PLD 或 .PDS)的 PAL 方程文件(或行为设计文件)。PLUSASMJ ADWFGF PALASM,大多数按 PALASM2布尔方程格式输出的 PAL 方

程文件是 PLUSASM 兼容的。

注意:可以用 Xilinx ABEL 重新编译基于 ABEL 的设计,最终的文件将是 PLUSASM 兼容的。

1. 利用 22V10 和 20V8 文件

PALCONVT 自动地处理包括隐含特性(如缺省的时钟,三态控制,全局置位/复位,等等)的 22V10 和 20V8 PAL 文件。

2. 利用通用的 PAL 文件

如果任何 PAL 方程文件的 CHIP 语句包括除“22V10”或“20V8”的 PAL 器件类型,或者当 CHIP 语句包括关键词,COMPONENT, PLFB9 或 PLFFB9 时,XEPLD 将假设方程文件是为通用 PAL,且将处理此方程作为一个行为设计,所有的方程必须在 PLUSASM 句法中明显地定义;通用文件不必包括任何隐含的特性(如三态控制,倒相等)。当器件无关的行为设计用 PLD 编译器处理时,通用方程也可以获得,当最终的 PLUSASM 兼容的方程文件不包含 CHIP 语句或者当 CHIP 语句省去目标的 PLD 类型,必须在文件中用不可识别的目标 PLD 类型规定一个 CHIP 语句。

CHIP my_pal 16R8; 16R8 is unrecognized and defaults to generic

注意:PAL 方程文件的腿 1 的信号假设是为任何寄存方程输出的缺省时钟,没有明显的时钟定义。

§ 11.3.3 PAL 变换顺序

准备一个或多个 PAL 方程文件在 Xilinx EPLD 中使用的顺序如图 11.5 所示。

第一步:产生 .PLD 或 .PDS 文件

对于在设计中的每个 PAL,从原始的 PAL 源码产生 PALASM2 布尔方程文件,例如,利用 ABEL 的“XFER”效用或 CUPL“-C”编译选项,所有 PAL 方程文件必须有 .PLD 或 .PDS 的扩展名。

第二步:输入 PAL 文件

拷贝 PAL 方程文件到设计目录,建议不用 /XACT 作设计目录。

注意:这是对设计校验每个 PAL 出腿清单中的信号名建立了适当信号连接的时机。

第三步:运行 XDM

XDM 是包括在 XEPLD 中的 Xilinx 设计管理程序,为了运行 XDM,在系统提示下键入:“XDM”,屏幕出现 XDM 主菜单。

第四步:选择器件类型和速度

根据设计要求选择 EPLD 器件的类型和速度

1) 由 XDM 主菜单选 PROFILE→FAMILY 指令打开器件系列清单,XDM 显示可用的各 Xilinx 器件系列。

2) 选择“XC7200”或“XC7300”,XDM 显示器件类型选择菜单。

3) 选择器件类型,例如选 7336PC144,XDM 显示速度选择菜单

4 选择一个器件速度档次,例如选“-7”。

注意:在 XDM 主菜单的左下角“Family:”和“Part:”部分也可改变选择。

第五步:产生顶层文件

PALCONVT 程序自动地产生顶层文件

1) 由 XDM 主菜单执行 FITTER→PALCONVT 指令运行 PALCONVT, 提示一个设计名字。

2) 输入设计名字, 如果文件名已经存在, 则会问是否要复盖此文件, XDM 然后在设计目录中列出所有 .PLD 或 .PDS 文件。

3) 选择一个或多个 PAL 文件包含到设计中, 在“NODE”上按键, 提示一个目标选择。

4) 如果选择选项1: “Create new PLD and PAL Interconnect Report”

PALCONVT 读方程文件, 并产生顶层文件, 每个被选的 PAL 文件名字是在 INCLUDE_EQN 语句中规定的, 来自每个 PAL 文件的出腿清单被合并在一起, 顶层文件被保存作为 design_name.PLD。图11.7为顶层文件的一个例子。

如果选择选项2: “Integrate new PLD using FITEQN”

XEPLD 运行 PALCONVT (象在选项1那样), 并利用 FITEQN 编译设计, 监视顶层文件校验由变换过程产生的结点和 I/O 腿分配是正确的, 然后到第八步“校验设计”。

当在 PAL 文件之间有矛盾时, PALCONVT 显示一个错误信息并暂停执行。PAL 互连报告 (design_name.INT) 将指出矛盾, 改变 PAL 源文件并运行 PALCONVT 直到设计无冲突为止。

第六步: 编辑顶层文件

在重新审看 PAL 互连报告之后, 可以利用文本编辑器编辑顶层文件 (design_name.PLD) 进行任何必要的改变到由 PALCONVT 进行自动分配。

1) 解决任何信号名的不符, 建立 PAL 文件之间的正确互连。

2) 改变由自动变换过程产生的缺省结点和 I/O 腿分配。

3) 规定任何要利用的全局控制信号 (Fast CLK, FOE)。

注意: 当完成编辑时, 在进行下一步过程前确信改变已保存。

第七步: 编译设计

当顶层文件完成时, 准备编译设计, 产生实际器件的版图。

1) 选择 FITTER→FITEQN 指令运行编译器, 在工作目录中显示所有的 .PLD 文件。

2) 选择设计文件 (design_name.PLD), XEPLD 产生 design_name.VMH 文件, 它包含逻辑设计的物理描述, 当它被映射到规定的 EPLD 中时, XEPLD 也产生为校验设计所用的报告。

第八步: 校验设计

利用由编译器产生的报告校验设计适合在规定的器件内, 利用文本编辑器选择要读的报告, 或选 UTILITIES→BROWSE 指令, 主要的报告是:

* design_name.RES 资源利用报告; * design_name.PIN 设计的出腿分配;

* design_name.ERR 警告和错误; * design_name.EQN 优化的方程文件;

第九步: 编程器件

利用 VERIFY→MAKEPRG 指令 (为 Intel Hex 格式) 或 VERIFY→MAKEJED 指令 (为 JEDEC) 格式产生器件编程的位映射文件。根据器件的编程器进行器件编程。

校验 PAL 变换

PALCONVT 产生 PAL 互连报告, 列出设计中每个 PAL 用的信号 (design_name.

INT),用它校验每个 PAL 包含文件引脚清单中的信号名建立的 PAL 文件间要求的互连。

§ 11.3.4 编辑顶层文件

PALCONVT 软件根据整个设计的假设产生顶层文件,在某些情况下需要人工地编辑顶层文件和重新分配某些赋予 NODE 语句的信号,这些信号将重新赋予 OUTPUTPIN 或 IOPIN 语句。

XEPLD 适配器读顶层文件和包含文件,优化设计,达到满意的结果。例如,适配器自动地拆开设计的逻辑,取全局时钟网线和全局输出使能控制网线的优点,即使在顶层文件中没有对它们作规定。但是当设计中有任何速度临界的通道或双向信号,必须规定适配器如何管理它们。

以下例子说明在要把速度临界信号映射到高速功能块和快速输入腿时,如何编辑顶层文件和如何管理双向信号。

1. PAL 变换例子

图11.8说明 PAL 设计的例子,PAL 方程如图11.9和图11.10所示,PAL 互连报告如图11.11所示,由 PALCONVT 产生的顶层文件如图11.12。

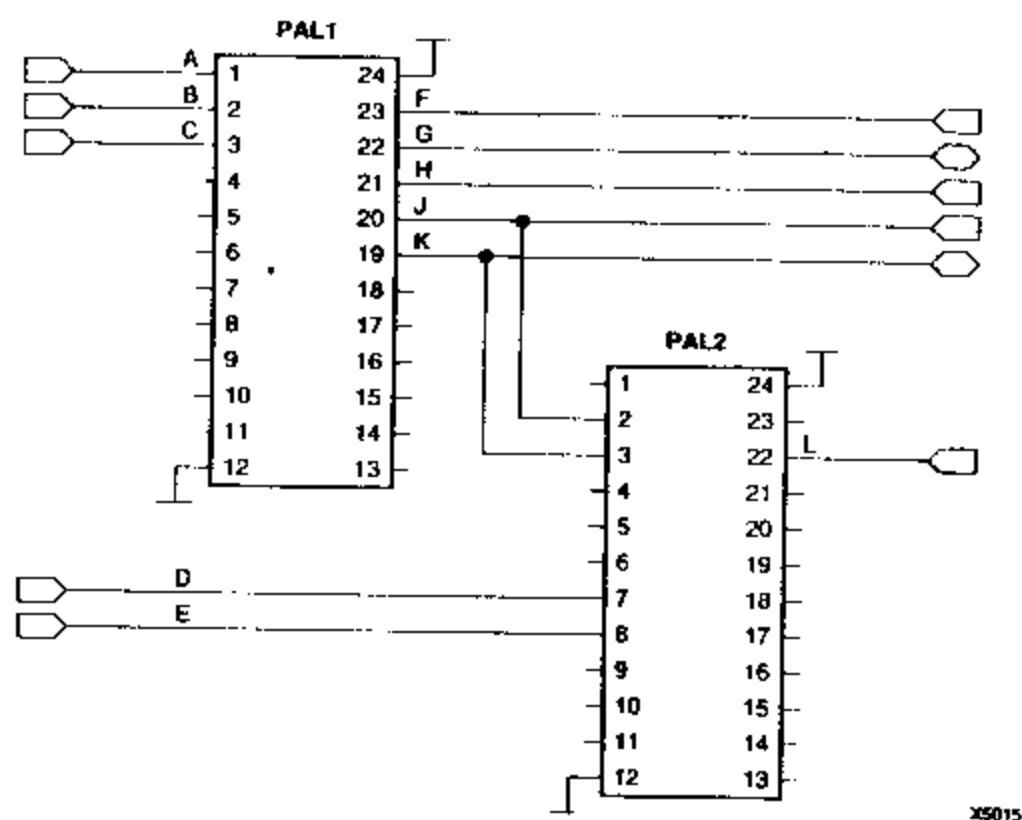


图11.8 原始 PAL 设计例子

```
TITLE PAL1
CHIP PAL1 P22V10
;PINLIST (Highest Pin Number = 24)
A B C NC NC NC NC NC NC NC NC NC NC NC NC NC NC NC NC K J H G F NC
; PALCONVT DESIGN EXAMPLE PAL1
EQUATIONS
F := (B);
G := (F);
G.TRST = C;
H := (G);
J := (B * K)
K := (B);
K.TRST = (C);
```

图11.9 PAL1 方程文件

```

TITLE PAL2
CHIP PAL2 P22V10
;PINLIST (Highest Pin Number = 24)
NC J K NC NC NC D E NC NC NC NC NC NC NC NC NC NC NC L NC NC
; PALCONVT DESIGN EXAMPLE PAL2
EQUATIONS
L = (J * K * E);
L.TRST = (D);

```

图11.10 PAL2方程文件

```

XEPLD, Version                               Xilinx Inc.
                                           PAL INTERCONNECT REPORT
Circuit name: EXAMPL1
Target Device: 7310884                      Integrated: 10-25-93, 6:26PM

PAL File: PAL2
+++++

PAL  Signal      PAL  CHIP
PIN  Name        Use  Use      Connectivity
 2   J           I   NODE      From PAL1:20
 3   K           I   NODE      From PAL1:19
 7   D           I   INPUTPIN   External Only
 8   E           I   INPUTPIN   External Only
22   L           O   OUTPUTPIN   External Only

Unconnected pins: 1 4 5 6 9 10 11 12 13 14 15 16 17 18 19 20 21

PAL File: PAL1
+++++

PAL  Signal      PAL  CHIP
PIN  Name        Use  Use      Connectivity
 1   A           I   INPUTPIN   External Only
 2   B           I   INPUTPIN   External Only
 3   C           I   INPUTPIN   External Only
19   K           O,FBK NODE      To PAL2:3
20   J           O   NODE        To PAL2:2
21   H           O   OUTPUTPIN   External Only
22   G           O,FBK NODE      Internal Only
23   F           O,FBK NODE      Internal Only

Unconnected pins: 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Total number of Output Equations specified = 6
Total number of device pins used for Input = 5
Total number of device pins used for Output = 2
Total number of device pins used for I/O = 0
Total number of Global Control pins used = 0

```

图11.11 PAL 互连报告的例子

```

PATTERN exampl1.pld - file made by PALCONVT command
CHIP exampl1 XEPLD
INCLUDE_EQN 'pal2.pds'
INCLUDE_EQN 'pal1.pds'
INPUTPIN D E A B C
OUTPUTPIN L H
NODE J K G F
EQUATIONS

```

图11.12(a) 由 PALCONVT 产生此例顶层文件

```

PATTERN exampl1a.pld - file made by PALCONVT command

CHIP exampl1a XEPLD

INCLUDE_EQN 'pal1.pds'
;INCLUDE_EQN 'pal2.pds' PAL2 equations copied into Top Level
;File to specify pin feedback only for K input of PAL2

FASTCLOCK          A          ; Changed from INPUTPIN
INPUTPIN           B C
FOEPIN             D          ; Changed from INPUTPIN
INPUTPIN (FI)      E          ; Changed from INPUTPIN to
                               ; fast input
OUTPUTPIN          F H J      ; Changed from NODE
IOPIN (PINFBK)     G          ; Changed from NODE
IOPIN              K          ; Changed from NODE
OUTPUTPIN (FOE=D)  L          ; Change from product term
                               ;to FOE enable

PARTITION FFB L          ; Place L in a fast function
                               ;block

EQUATIONS

L = (J * K.PIN * E)      ; Use pin feedback instead
                               ;of macrocell feedback
                               ;for K

```

图11.12(b) 此例的顶层文件(编辑的)

2. 分配结点到输出

PALCONVT 软件把出现在方程两边的所有信号注释为结点,并把它们分配到顶层文件中的结点语句,有四种情况需要改变这些结点说明:

1)驱动其它 PAL 输入也驱动 EPLD 输出的 PAL 输出必须转到 OUTPUTPIN 语句。

PAL 之间互连信号被假设是结点,在 PAL 变换例子中,PALCONVT 分配信号 J 到一个 NODE,因为它出现在方程的两边:

$J := (B * K);$ PAL1

$L = (j * K * B);$ PAL2

这个信号必须从 NODE 语句移去,并放进顶层文件中 OUTPUTPIN 语句,如图11.13所示。

2)在相同 PAL 中用作反馈的 PAL 输出,并也驱动 EPLD 输出必须转到 OUTPUTPIN

语句。

驱动内部逻辑反馈的 PAL 输出被假设是结点,在 PAL 变换例子中,PALCONVT 分配信号 F 到一个 NODE 替代一个出腿,因为信号 F 出现在方程的两边:

F := (B); PAL1

G := (F); PAL1

这个信号必须从 NODE 语句移去,并放进顶层文件中 OUTPUTPIN 语句,如图11.13所示。

3)利用出腿反馈的外部双向信号必须转到 IOPIN 语句。

双向信号总是假设为结点,因为它们出现在方程的两边。在 PAL 变换例子中,PALCONVT 分配信号 G 到一个 NODE 替代一个出腿,因为信号 G 用作输入和输出:

G := (F); PAL1

H := (G); PAL1

G. TRST = C; PAL1

信号 G 必须从 NODE 语句移到顶层文件中 IOPIN 语句,如图11.13所示。

在 IOPIN 语句中利用 PINFBK 选择规定反馈总是来自 XC7000的器件出腿,例如:

IOPIN (PINFBK) G

4)利用出腿反馈和内部宏单元反馈的外部双向信号必须转到 IOPIN 语句;利用出腿反馈的情况必须用.PIN 扩展被标识。

双向信号总是假设为结点。在 PAL 变换例子中,PALCONVT 分配信号 K 到一个 NODE 替代一个出腿,因为信号 G 用作输入和输出:

K := (B); PAL1

L := (J * K * E); PAL2

K. TRST = (C); PAL1

为改变信号 K 到 XC7000 I/O 信号,利用宏单元反馈(到 PAL1)和出腿反馈(到 PAL2),按下面进行:

(1)在顶层文件中把信号 K 作为 IOPIN 说明。

(2)把方程右边所有出现的 K 改变为 K. PIN(在 PAL2. PDS 中),把方程移进顶层文件。

一个包含文件必须至少有一个方程,在此例子中,当包含 K 的方程移到顶层文件,PAL2. PDS 不再包含文件,所以 PAL2不再被包含。

3. 分配输出使能信号到 FOE 网线

把任何三态输出使能信号从表示为利用 UIM 互连的. TRST 控制方程乘积项中改到全局快速输出使能(FOE)网线都导致减少资源要求和输出使能延时。

XEPLD 软件任何时候没有用户干预都可能自动地分配最常用的输出使能信号到 FOE 网线,但也可以编辑顶层文件人工分配 FOE 信号。

在 PAL 变换例子中,规定信号 D 为输出 L 的 FOE 控制,对顶层文件做以下的改变:

1)加"FOE=D"选项到要控制的输出(L)的 OUTPUTPIN 语句:

OUTPUTPIN (FOE=D) L

2)加语句: FOEPIN D

3)从 INPUTPIN 语句去掉信号 D:

INPUTPIN E A B C

4)在顶层文件中去掉. TRST 控制方程($L. TRST = D$)

不可以利用 FOEPIN 信号作为逻辑输入,FOEPIN 影响出腿反馈,但不影响 UIM 反馈只有用单个高有效出腿控制的输出使能信号可以分配到 FOE 网线,低有效输出使能信号必须来自乘积项(宏单元)。

4. 分配时钟信号到 FastCLK 网线

把时钟信号从表示为. CLKF 的控制方程乘积项中改到全局快速时钟(FastCLK)网线将会减少资源要求和时钟到输出的延时。

XEPLD 软件任何时候都可能自动地分配最常用的时钟信号到 FastCLK 网线,但也可以编辑顶层文件人工分配 FastCLK 网线。

在 PAL 变换例子中,规定信号 A 为 FastCLK,对顶层文件做以下的改变:

1)加语句: FASTCLOCK A

2)从 INPUTPIN 语句去掉信号 A

INPUTPIN E B C

XC7000系列的结构不允许相同的腿用于 FASTCLOCK 和逻辑输入。

5. 分配方程到高速功能块

高速功能块(FFB)比高密功能块速度快,并提供增加的输出驱动能力;软件在可能的時候自动地分配输出功能到 FFB。但也可以利用 PARTITION 语句人工分配输出信号和结点到 FFB。当信号出现在 FFB 方程右边时,利用 INPUTPIN(FI)语句或利用. FI 扩展分配高速输入信号到 FFB。

分配到 FFB 的寄存的功能仅可以利用 FastCLK 网线,且可以被非同步设置(XC7336例外,它可以被非同步复位)。

分配到 FFB 的任何功能仅可以利用 FOE 信号为三态输出控制。

所有的 FFB 输出(XC7336例外)在器件的出腿必须是低有效,如果它们不是低有效,只要可能软件将变换方程到低有效的输出。

在 PAL 变换例子中,分配信号 L 到 FFB 和信号 E 到高速输入腿,对顶层文件做以下的改变:

1)加语句: PARTITION FFB L

2)加语句: INPUTPIN (FI) E

因为在编辑过程(FITEQN)期间 L 被分配到 FFB,软件会自动地产生以下的语句来改变三态输出使能控制信号从乘积项控制到 FOE 控制:

FOEPIN D

OUTPUTPIN (FOE=D) L

软件也会从 PAL2方程文件去掉乘积项三态控制语句:

$L. TRST = (D);$

软件不修改原始的 PLD 文件,它产生一个由 FITEQN 编辑过的文件的修改型式,这个被修改的设计文件在 design_name. EQN 文件中给出,如图11. 13所示。

```

; This is the .eqn file produced by the partitioner. It shows
; how your equations were implemented in order to best utilize the
; resources available on the chip.
;
; This design was compiled for the 7310884

PATTERN exampl1a.eqn
DATE Mon Oct 25 19:03:21 1993
CHIP EXAMPL1A XEPLD
MINIMIZE OFF
PARTITION FB1_2 F H J L
PARTITION FB10_9 G
PARTITION FB6_1 K
INPUTPIN B
INPUTPIN ( FI ) E
OUTPUTPIN F H J
OUTPUTPIN ( FOE = D ) L
IOPIN ( FOE = C ) G K
FASTCLOCK A
FOEPIN C D
EQUATIONS
/L = /K.PIN
    + /J
    + /E
/F := /B
    F.CLKF = A
/G := /F
    G.CLKF = A
/H := /G.PIN
/J := /K
    + /B
    J.CLKF = A
/K := /B
    K.CLKF = A

```

图11.13 PAL 变换例子的方程报告

§ 11.3.5 利用方程报告校验设计时序

在不使用时序仿真器时,可以利用方程报告计算设计将运行多快。

在图11.13给出的 PAL 变换例子的方程报告中,PARTITION 语句表示优化软件把输出信号 F,H,J 和 L 放进高速功能块 FB1,I/O 信号 G 和 K 放进高密功能块,出腿说明语句表示两个输出使能信号 C 和 D 放进全局 FOE 网线,以下的时序计算根据图11.13。

1. 建立和保持—信号 B 和 G 至时钟

$$\text{Setup Time} = t_{\text{SUF}} + t_{\text{UIM}} = 6 + 10 = 16\text{ns}$$

$$\text{Hold Time} = t_{\text{HF}} + t_{\text{UIM}} = 0 + 10 = -10\text{ns}$$

2. 时钟到输出—由时钟到 F,G,H,J,K,L

信号 F,H 和 J 已放进高速功能块,所以高速功能块外部时钟到输出的延时(t_{COF})为:
 $\text{Clock-To-Output delay} = t_{\text{COF}} = 9\text{ns}$

信号 G 和 K 已放进高密功能块,这些信号的时钟用 FastCLK 实现,所以它们的时钟到输出的延时是高密功能块外部快时钟到输出的延时(t_{CO})为:
 $\text{FastCLK-To-Output delay} = t_{\text{CO}} = 12\text{ns}$

信号 L 已放进高速功能块,与放进高密功能块的寄存函数(K)的腿反馈无关,时钟到输出的延时通道由高密功能块外部快时钟到输出的延时(tCO)和高速功能块外部自 I/O 腿的腿—腿传播延时(tPDFU)之和来计算:

$$\text{Clock-To-Output} = t_{CO} + t_{PDFU} = 12 + 22 = 34\text{ns}$$

3. 输出使能/禁止—信号 G,K,L

在出腿说明语句中已说明这些信号由 FOE 腿控制,输出使能/禁止时间等于 I/O 焊盘输出使能/禁止延时(tFOEI):

$$\text{Output Enable/Disable time} = t_{FOEI} = 12\text{ns}$$

4. 腿—腿传播延时—信号 E 到 L

信号 L 已放进高速功能块,信号 E 已说明为快输入,腿—腿传播延时等于高速功能块外部传播延时(tPDFO):

$$\text{Pin-to-Pin Propagation Delay} = t_{PDFO} = 12\text{ns}$$

5. 最大频率—时钟 A

此设计最长的信号路径是由高密功能块寄存器(G)的输出,通过 I/O 缓冲器(信号 G 的腿反馈),通过 UIM,进入高速功能块寄存器(H):

$$\text{Ckcle Time} = t_{COI} + t_{OUT} + t_{IN} + t_{UIM} + t_{FLOGI} + t_{FSUI} = 1 + 8 + 4 + 10 + 2 + 3 = 28\text{ns}$$

§ 11.4 原理图中利用 PLD 文件

在原理图设计中包含行为方程是一个容易的过程,因为 XEPLD 自动地处理原理图,装配任何 PLD 方程文件,自动地集成最后的设计。虽然设计者可以产生完全的 Xilinx EPLD 设计,利用预先定义的原理图库元件,如门电路,触发器和计数器等,也可以产生由行为设计或 PLD 文件表达的定制元件,来完全控制器件结构。

§ 11.4.1 选择库元件

为了在原理图设计中包括一个行为方程文件,必须利用专门的 PLD 库元件,或产生定制的元件,在 XEPLD 库中每一类 PLD 元件的特性和使用它们的要求在下面进行讨论。表 11.3 表示规定 PLD 文件到原理图符号的各种方法。

表11.3 规定的库元件

器件类型	CHIP 语句	库元件
PAL—22V10	22V10	PL22V10
PAL—20V8 GAL—20V8	20V8	PL20V8
其它 PAL	未标识 *	PL20PIN PL24PIN PL48PIN
定制	COMPONENT	定制符号
高密功能块	PLFB9	PLFB9
高速功能块	PLFFB9	PLFFB9

注:除22V10或20V8外的任何 PAL 器件。

在 CHIP 语句中的 PLD 类型不被识别,它将对 GENERIC 缺省。

1. 利用 PL22V10或 PL20V8

XEPLD 通过专门的 PAL 库元件支持22V10和20V8的 PAL 器件,如果已经有针对它们的 PAL 设计或有为它们写方程文件的经验,通常会选择这些元件。XEPLD 支持这些器件所有隐含的特性,提供自动的分割和方程分割。

如果不能得到这些器件的 PALASM 源文件,XEPLD 提供自动的 JEDEC 文件变换

2. 利用 PL20PIN, PL24PIN 和 PL48PIN

除22V10和20V8以外的 PAL 由通用的20腿,24腿和48腿 PAL 库元件支持,通用 PAL 方程文件必须以与结构无关的形式显式地定义所有的逻辑方程;任何隐含的功能,如三态控制,信号变换,时钟等(那些对原始选定的 PAL 器件作的规定)不可能被 XEPLD 识别。但是,XEPLD 不提供对这些器件的自动分割和方程分割。

在 PAL 文件 CHIP 语句中利用"GENERIC"一词(或除22V10或20V8之外任何 PAL 器件类型)把 PAL 文件转到通用 PAL 元件,在这些通用元件符号中提供的每个出腿可以用于任何在 PAL 方程文件中的输入,输出或 I/O 功能,也可以产生专门的定制符号表示 PAL 方程,在"产生定制元件"部分中描述。

XEPL 没有为通用 PAL 的 JEDEC 变换应用,要求带有 .PLD 或 .PDS 文件扩展的 PALASM 布尔方程文件。

3. 利用 PLFB9和 PLFFB9

这些库元件是依据在 XC7000系列的高密功能块(PLFB9)或高速功能块(PLFFB9)所包含的逻辑把定制逻辑转到这些元件之一逻辑映射到单个功能块;分割和方程分割将不被编译器执行。库符号有足够的 I/O 和控制腿以支持所选功能块所有可能的应用,PLFB9符号包括算术进位入和进位出的出腿。

如果熟悉 PLUSASM 和希望利用 EPLD 器件结构的专门特性,一般会选择这些元件,这些允许有经验的用户控制器件中方程的布局产生最致密的设计。

在方程文件 CHIP 语句中利用关键词"PLFB9"或"PLFFB9"把 PAL 文件转到 PLFB9或 PLFFB9元件。

4. 产生定制元件符号

可以定义带有其专有的唯一的符号和出腿的定制元件,把它们放进原理图。定制符号可以有出腿,在检查原理图时,较容易识别它们。当定制功能重复地用于整个设计,它们是特别有用。

在方程文件 CHIP 语句中利用关键词"COMPONENT"把 PAL 文件转到定制符号,即通知软件符号上的出腿有方程文件中信号相同的名字。对这些元件的逻辑利用任何行为设计方法来产生,与任何通用 PLD 方程文件一样,所有逻辑功能必须用方程式显式地定义。

不像表示主要原理图的结构符号,定制符号由 XEPLD 软件处理作为基本元件,可能需要一个属性给符号表示它是一个基本元件,并不被扩展到主要的原理图中。

§ 11.4.2 选择 PLD 开发方法

各种 PLD 开发方法的完整描述在 § 11.1 和 § 11.2 中已给出,下面介绍每个方法如何应

用于原理图。

1. 利用 JEDEC 文件

利用下面的指令可以把每个 JEDEC 文件变换成 PLUSASM 方程文件

XDM→TRANSLATE→JED2PLD

然后可以按设计流图的第七步描述的那样利用 TRANSLATE→XEMAKE 处理原理图设计。

JEDEC 文件对早先输入的 22V10 或 20V8PAL 的已有文件是有用的,总是利用 PL22V10和 PL20V8库符号表示 JEDEC 文件。

2. 利用 PLUSASM

在要使用如高速进位,时钟反馈等专门的结构特性时,利用 PLUSASM 开发 PLD 方程文件,产生器件的终止输出通道。PLUSASM 是 Xilinx EPLD 依据 PALASM2布尔方程语法的自然语言。此外,此语言包含允许使用 XC7000系列的先进结构特性的语法结构。

可以对任何原理图中的 PLD 或定制元件在 PLUSASM 方程文件的 CHIP 语句放上 PL22V10,PL20V8,PLFB9,PLFFB9,GENERIC 或 COMPONENT。

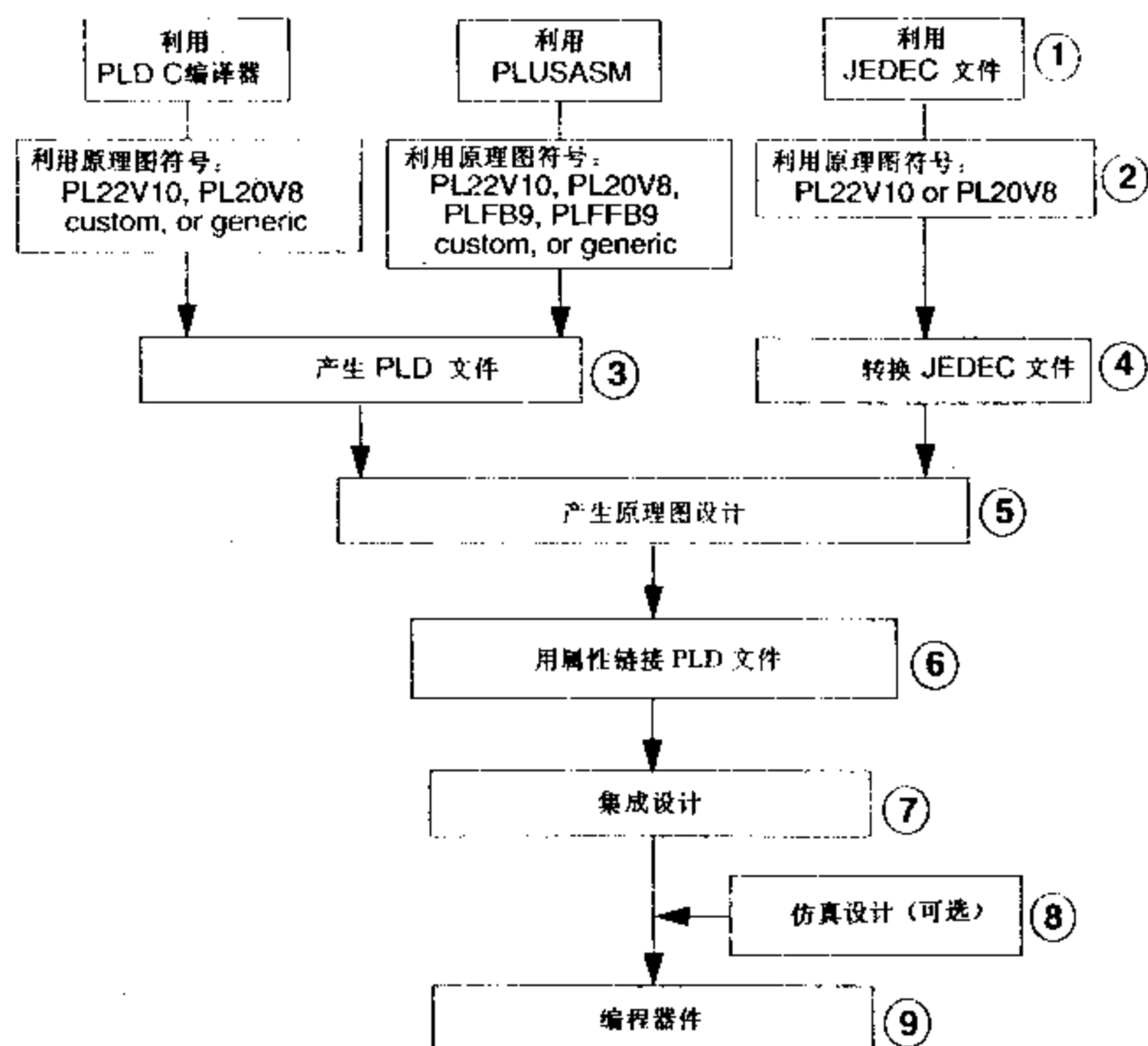


图11.14 在原理图中利用 PLD 的设计流程

3. 利用 PLD 编译器

在要使用编译器的高级语言能力的优点,但不需要如高速进位,时钟反馈等器件的专门特性时,利用 PLD 编译器开发 PLD 方程文件,产生器件的终止输出通道。这个方法对于输入已有的 PAL 文件也是有用的。

如上所述,PLUSASM 是 Xilinx EPLD 依据 PALASM2 布尔方程语法(.PDS)的自然语言,许多通用的 PAL 编译器,如 ABEL,CUPL,LOG/iC 和 PALASM 可以产生 XEPLD 软件要求的 PALASM2 布尔方程文件。

利用 PAL 编译器的内装的产生 .PDS 文件的能力(如 ABEL XFER 用途或 CUPL -c 的编译器的选择),可以产生 PLUSASM 兼容的方程文件。

这些方程文件可以针对 PL22V10,PL20V8,通用 PAL 或定制原理图符号。

§ 11.4.3 设计流程

产生包含 PLD 文件的原理图设计如图 11.14 所示。整个设计步骤包括:选择 PLD 设计方法;选择 PLD 元件;产生 PLD 文件;变换 JEDEC 文件;产生原理图设计文件;链接文件到原理图;集成设计;仿真设计;编程器件等。

下节介绍利用 OrCAD 软件设计 EPLD 的方法,原理图输入和仿真等方法同样可用于设计 FPGA;同样地,下一章中介绍的 ViewLogic 软件的设计方法也可用于设计 EPLD。因为 OrCAD 和 ViewLogic 软件都是 Xilinx 采用的 PC 机上的第三方的设计工具。

§ 11.5 OrCAD 软件设计 Xilinx EPLD

这一节以通用非同步收发器(UART)为例说明如何利用 OrCAD 设计 EPLD。

UART 电路变换串行数据流为并行字节,并提供握手和错误检测信号到主机系统。图 11.15 是 UART 接收机的功能性逻辑图,其功能如下:

1. 一个串行到并行的移位寄存器,Deserializer,变换串行数据流为并行数据,然后锁存到数据输出寄存器,Data Output Reg。

2. 一个简单的状态机,FFrame Detector,控制此接收机,一旦起始位被检测,计数器,Frequency Divider,开始顺序计数,由四倍的波特率时钟定时。

3. 由准备信号,Rcvr Ready,通知主机,由接收机输出总线使能信号确认来读数据。

4. 计数器的输出被译码产生移位寄存器,数据锁存器和错误检测电路的控制信号,有以下信号产生:

- 1) 当字节奇偶性为奇数时产生奇偶错误。
- 2) 当任何停止位为低时产生信息段错误。
- 3) 在 CPU 读前一数据之前,新的数据准备锁存进输出寄存器产生溢出错误。

§ 11.5.1 利用 XEPLD 软件

一、系统准备

由 XDM 菜单系统,可以访问到 XEPLD 软件,在起动 XDM 之前,必须装配了 XEPLD

软件和 OrCAD 库,此外,校验 DOS 环境配置。

1. 要求 config.sys 文件包含如下内容:

```
files=25
```

```
buffers=25
```

```
rem make extra space to accommodate more environmental variables
```

```
shell=c:\dos\command.com c:\dos\ /e:1024 /p
```

2. 要求 autoexec.bat 文件包含如下内容:

```
set XACT=c:\xact
```

```
set ORCADEXE=c:\orcadexe\
```

```
set ORCADESP=c:\orcadesp\
```

```
set ORCADPROJ=c:\orcad\
```

```
set ORCADUSER=c:\orcadesp\
```

```
path ...;c:\xact;c:\orcadexe;.....
```

3. 包含 XDM 和 OrCAD 软件目录的 XACT 软件目录必须包含在路径内。在执行任何其它准备步骤前,必须设置 OrCAD 变量和 XACT 变量。

4. 所有 OrCAD 的环境变量必须按照 OrCAD 的要求设置。

例如:ORCADPROJ 变量设置在 c:\orcad\ 之下。

5. XACT 变量必须按照 XACT 装配要求设置。

例如:XACT 变量设置在 c:\xact 之下。

6. 为了规定 XDM 下所用的文本编辑器,设置 EDITOR 环境变量。

二、起动 XDM

在 DOS 提示下键入“XDM”,显示器显示 XDM 的菜单。

三、配置 XEPLD 的环境

由屏幕左下角,选 Family,规定所用的 Xilinx 器件等。

§ 11.5.2 在 DRAFT 中画设计原理图

一、产生新的设计

开始新的设计之前,必须产生新的设计目录

1. 产生设计目录

1) 选 DesignEntry→ ORCAD,出现 OrCAD ESP 主菜单

2) 选 Design→ Manegement→ Tools→ Execute

3) 一个对话框出现,目录清单列在左边,设计文件清单列在右边,一组按钮在底部。

4) 一个 TEMPLATE 目录通常是加亮的,如果没有被加亮,在左边的清单中选择 TEMPLATE 目录。

5) 在 Create Design 按钮上按动鼠标左键。

出现产生设计的对话框,在新设计名字(New Design Name)的框上按动鼠标左键。

6) 当小的白色方形出现在框内时, 键入 "uart".

7) 在 OK 按钮上按动鼠标左键。在左边的目录清单内选择新的 UART 目录。

2. 拷贝设计文件

由 Xilinx OrCAD 库提供的完全的 UART 设计按装在 \xact 目录下, 拷贝此文件到新的设计目录下。

在设计管理工具 (Design Management Tools) 对话框内, 在 Suspend to System 上按动鼠标左键。DOS 壳体打开新的设计目录作为当前目录。

拷贝示教目录的整个内容:

```
"copy c:\xact\tutorial\orcad\uart" <Enter>
```

3. 配置 UART 目录

在开始 Xilinx EPLD 设计之前, 必须利用 XDrafter 程序配置设计目录以存取 \xact\ xc7000 的库。

在 DOS 提示符下, 键入: "xdrafter 7" <Enter>

键入: "exit" <Enter>, 返回到 OrCAD。

在 OK 按钮上按动鼠标左键, 返回到主菜单。

二、打开和观察设计

在 OrCAD 原理图编辑器 Draft 中打开 UART 的原理图。

1. 打开设计

1) 选 Schematic Design Tools → Execute

2) 选 Draft → Execute

出现如图 11.15 所示的 UART 原理图。当 UART 原理图没有打开, 选 Quit → Initialize, 键入: "uart" <Enter> 来打开此原理图。

在任何时候要退出原理图输入, 返回到主菜单, 选 Quit → Abandon Edits。

2. 在 OrCAD 中利用鼠标

在 OrCAD 中鼠标执行以下功能:

左按钮: 选择; 右按钮: 取消 (与 Escape 键相同)。

3. 由 SDT 菜单选择

为编辑原理图的指令被组织在一系列菜单中, 可以用两种方法选择菜单指令:

1) 按动鼠标左按钮一直到所要的指令出现在菜单中, 然后选择此指令。在选择指令时, 原理图上指针的放置没有影响, 指针释放, 鼠标只影响指令的选择。

2) 键入指令名字对应的第一个字母键, 如果知道所要指令的命字, 这常常是较容易的方法。

为了取消指令, 并返回到前面的菜单, 按鼠标右键或按 Escape (Esc) 键。

Draft 提供几个改变设计的幅面等级的缩放指令。

放大: 用鼠标 zoom → in → 1, 2, 5, 10, 20; 用键盘 z → i → 1, 2, 5, 10, 20

缩小: 用鼠标 zoom → out → 20, 10, 5, 2, 1; 用键盘 z → o → 20, 10, 5, 2, 1

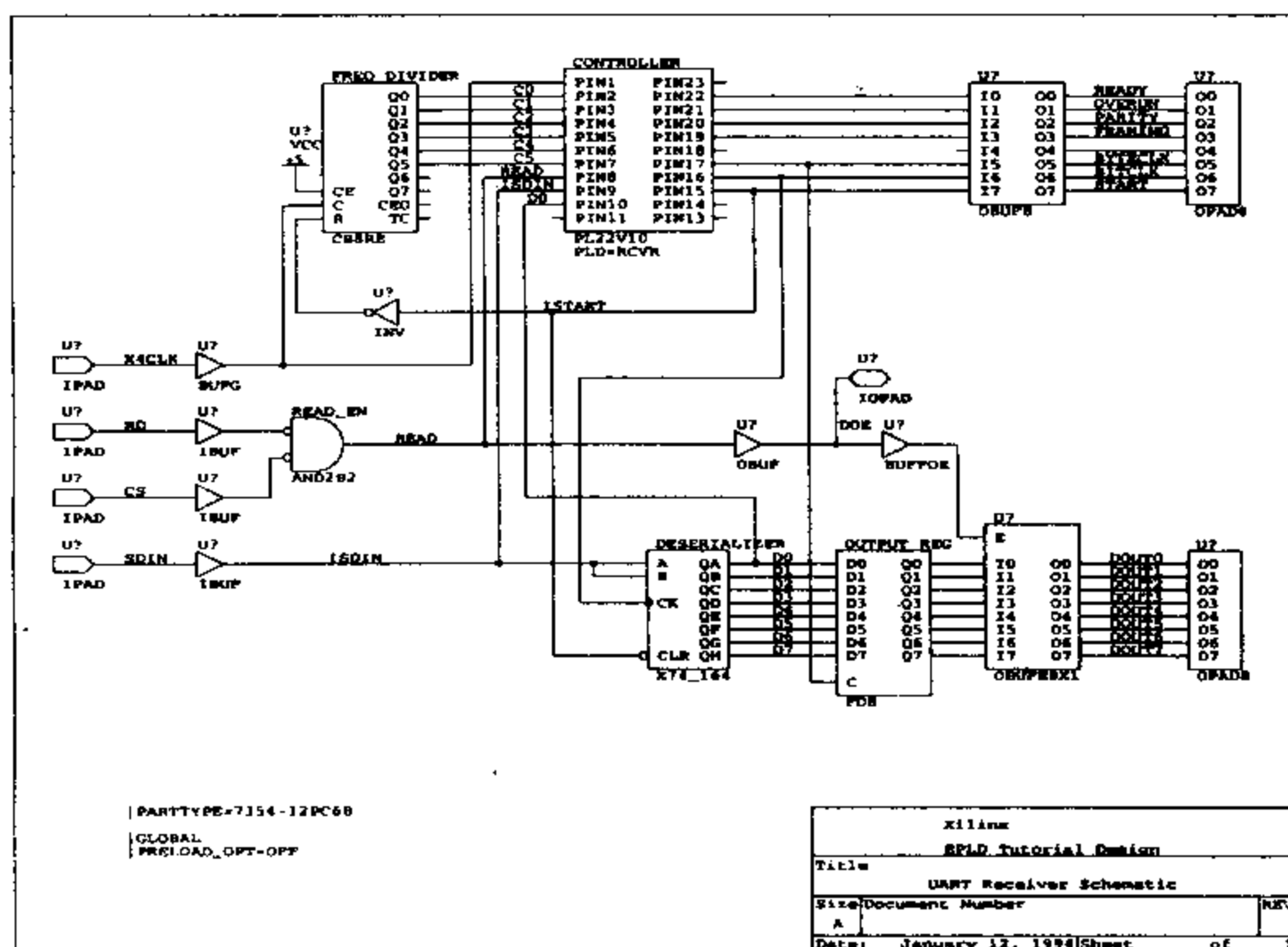


图 11.15 完全的 UART 原理图

§ 11.5.3 产生新的原理图

按动鼠标左键后,由显示的菜单选 Quit→Initialize(或 q→i)

此指令退出当前的原理图,允许产生新的原理图。

键入"uart2" <Enter>,出现空白的原理图,在左上角提示信息:<<<New Worksheet>>>,此原理图命名为 uart2. sch。

一、输入元件

1. 按鼠标左键, 选 get (g), 提示出现 {Get ?}
2. 键入 "ibuf" <Enter>, 一个 IBUF 元件的图形出现; 或者响应 {Get ?} 键入 <Enter>, 从菜单选 XC7000.LIB, 在元件名字清单上移动指针, 选择 IBUF。
3. 移动鼠标, 直到元件图形是在屏幕左下角。
4. 由 Get 菜单选 Place 指令, 元件被放置, 第二个图形出现在它的上面。
5. 移动第二个图形在第一个上面1英寸。
6. 又一次选 Place, 第三个图形出现。
7. 移动第三个图形在第二个上面1英寸。
8. 又一次选 Place, 第四个图形出现。
9. 按鼠标右键或<Esc>键取消 Get 指令, 第四个图形和 Get 菜单消失。

二、排列元件

在元件输到图中后,必须能移动它们,移动元件有以下步骤。

1. 由菜单选 Block→ Move。
2. 把指针放到顶部的 IBUF 元件附近。
3. 选 Begin 指令起始要移动目标的选择。
4. 移动指针,指针的第一个位置对应所选框架的一个角,当前位置是相反的角度。
5. 移动选择框架的角直到框架跟踪到顶部 IBUF 元件的位置。
6. 选 end。
7. 移动鼠标来移动元件图形。
8. 当图形在所要的位置,选 Place 指令。

三、取消元件

1. 选 Delete→ Object。
 2. 移动指针到顶部的 IBUF 元件。
 3. 选 Delete 子菜单的指令,元件消失。
 4. 按鼠标右键或<Esc>键结束 Delete 指令。
- 利用 Delete Undo 指令恢复消错的元件。

四、放置旋转的元件

1. 选 Get,在提示处键入"inv"<Enter>。
2. 在 Get 菜单选 Rotate 两次,INV 元件现在反相了。
3. 在 Get 菜单选 Place,放置 INV 元件在适当的位置。
4. 按鼠标右键或<Esc>键结束 Get 指令。

五、输入其它元件

1. 利用 Get 指令增加以下元件:PL22V10,CB8RE,BUFG,IBUF,IPAD 和 AND2B2。
BUFG 元件代表 FastCLK 输入。
2. 利用 Get 指令增加 VCC 符号,不要利用 Add Power 指令来加这个符号。

六、画连线

1. 选 Place→ Wire,移指针到 CB8RE 的 Q0腿的末端;
2. 选 Begin,移指针到 PL22V10的腿2,选 End。

连线要注意,仔细地在准确的元件腿末端的栅格点上开始和结束连线,否则 OrCAD 不认为连线被接通。

为节省时间可以利用 Repeat 指令画并行的与刚画的线相同长度的连线,利用 Set Repeat Parameters 指令设置 X Repeat Step 为0;Y repeat Step 为+1。

4. 选 Repeat 五次,CB8RE 的 Q0到 Q5有连线到 PL22V10的腿2到7。

七、移动一个块

1. 选 Block→ Drag, 移动指针到 PL22V10 内或周围的一点。
2. 选 Begin, 移动框架的角到 PL22V10 内或周围的另一点, 以致没有其它元件在框内。
3. 选 End, 移 PL22V10 的符号, 直到连线是直的为止, 选 Place。

八、简便法画线

1. 选 Place→ wire, 把指针放到较上面的 IBUF 元件的输出端, 选 Begin;
2. 移指针到 AND2B2 的上面的输入端, 选 new;
3. 移指针到低一个 IBUF 的输出端, 选 Begin;
4. 移指针到 AND2B2 下面的输入端, 合适地放置中间段的连线;
5. 再一次选 Begin, 移指针到 AND2B2 的下面的输入端, 选 new;
6. 连续地加线直到全部连线完成, 按鼠标右键或 <Esc> 键结束加连线。

九、加连结点(Junction)

1. 选 Place→ junction, 移指针到两线交点处, 选 Place;
2. 全部加完后, 按鼠标右键或 <Esc> 键结束加连结点。

十、标注元件

1. 把指针放到 PL22V10 元件上, 选 Edit→ Edit→ Reference→ Name;
2. 去掉 Reference? 提示后面的 U?, 键入 "Controller" <Enter>;
3. 按鼠标右键或 <Esc> 键。
4. 把指针放到 CB8RE 元件上, 选 Edit→ Reference→ Name;
5. 去掉 Reference? 提示后面的 U?, 键入 "Freq_Divider" <Enter>;
6. 按鼠标右键或 <Esc> 键。
7. 按 4—6 的步骤给 AND2B2 加标注 "Read_En", 在标注中只用字母字符和下划字符。

十一、标注连线

1. 选 Place→ Label, 出现 Label? 提示;
2. 键入 "READ" <Enter>, 移动鼠标指针, 使出现的标注移到连结 AND2B2 元件输出的连线上;
3. 选 Place, 出现 Label? 提示, 重复步骤 2 放置以下标注:
C0—C5, X4CLK, RD, CS, SDIN, START, ISTART, ISDIN4. 按鼠标右键或 <Esc> 键, 结束加标注。

十二、加 PLD 属性

1. 移指针到 PL22V10 元件上, 选 Edit→ Edit→ 1st Part Field→ Name, 出现 1st Part Feild? 的提示;
如果第一个元件字段已用于其它元件, 选取其它七个元件字段的任一个。XEPLD 适配

器检验所有元件字段,提取它发现的所有 XEPLD 的属性。

2. 键入 "PLD=RCVR" <Enter>, 按鼠标右键两次或 <Esc> 键两次, 结束 Edit 指令。

十三、加 PARTTYPE 属性

1. 选 Place→Text, 出现 Text=? 的提示;
2. 键入 "PARTTYPE=7354-12PC68" <Enter>;
3. 把指针移到设计图的左下角空白处, 选 Place, 放置此属性。

十四、加全局属性

1. 如13项, 利用 Place Text 指令加以下全局属性的正文:

|GLOBAL

|PRELOAD_OPT=OFF

起始的管道字符必须对齐, 用 GLOBAL 字符串开始。

2. 按鼠标右键或 <Esc> 键, 结束 Place Text 指令。

利用上述各项 Draft 的指令, 输入剩下的元件, 连线, 标注, 完成 UART 的原理图输入, 并仔细地进行校对, 改正各种错误。

十五、分配信号到专门的腿

1. 把指针移到与 SDIN 信号相连的 IPAD 元件上;
2. 选 Edit→Edit→2nd Part Field→name, 出现 2nd Part Field? 的提示;
3. 键入 "LOC=P18" <Enter>;
4. 按鼠标右键两次或 <Esc> 键两次, 结束 Edit 指令。
5. 分配其它的信号到所希望的出腿上, 参考相应的器件手册上有关出腿信息。

十六、保存设计

1. 选 Quit→Write to File, 并键入 "uart" <Enter>;
2. 选 Quit→Abandon Edit→Yes。

十七、退出 OrCAD

1. 由原理图设计工具菜单选 To Main→Execute
2. 选 Exit ESP→Execute

按任何键, 返回到 XDM, 出现 XDM 屏幕。

§ 11.5.4 定义 PLD 方程

一、定义说明语句

1. 在 XDM 的目录(Directory)字段上按鼠标, 或选 Utilities→Directory;
2. 在指令提示后键入 "c:\orcad\uart" <Enter>, 或由目录清单选 UART;
3. 选 Done, 在 XDM 目录字段显示 UART 的整个路径。

4. 类似地, 确认 Family 和 Part 字段被置于 XC7300 和 Indesign。

```

TITLE      UART Receiver Controller
AUTHOR     Applications
COMPANY    Xilinx
DATE       February 1993

CHIP       RCVR      PL22V10

; 1      2  3  4  5  6  7  8      9      10 11 12
x4clk c0 c1 c2 c3 c4 c5 read sdin d0 nc gnd

; 13 14 15      16      17      18 19      20      21      22      23 24
nc nc start bitclk byteclk par framing parity overrun ready nc vcc

EQUATIONS

/start := /start * sdin
        + c5*/c4*c3*/c2*/c1*c0
        ; start goes high when sdin goes low
        ; and stays high until count=41.

bitclk := /c0 * /c1 * start
        ; bitclk pulses every 4 clock cycles
        ; to strobe deserializer.

byteclk := c5*/c4*/c3*/c2*c1*/c0 * /ready
        ; byteclk strobes output_reg at count=34
        ; only if ready not still active.

overrun := c5*/c4*/c3*/c2*c1*/c0 * ready
        + overrun * /read
        ; overrun error at count=34 if ready still
        ; active; stays on until read.

par := par * /sdin * bitclk * start
      + /par * sdin * bitclk * start
      + par * /bitclk * start
        ; accumulate parity on sdin on each bitclk;
        ; reset while start=0.

parity := c5*/c4*/c3*/c2*c1*/c0 * par
        + parity * /read
        ; parity error at count=34 if par odd (1);
        ; stays on until read.

framing := c5*/c4*c3*/c2*/c1*/c0 * /sdin

```

图11.16 Rcvr.pld 文件

5. 选 Utilities→ Browse 或 Utilities→ Edit;

6. 由文件清单选 RCVR. PLD, 检查 PLD 文件的说明部分, 如图 11. 16。

二、产生布尔方程

1. 检查 RCVR. PLD 文件的方程部分
2. 退出编辑器, 返回 XDM 执行屏幕。

§ 11. 5. 5 选配设计

一、校验设计

1. 选 Translate→Cleanup, 选 UART. SCH, 然后选 Done;
2. 为执行电气规则检查(ERC), 选 Translate→Annotate→UART. SCH→Done;
3. 为运行规则校验, 选 Translate→INET→UART. SCH→/W→Done, 运行规则校验, 或在 XDM 的指令行键入:

"dos inet uart. sch uart. erc /w" <Enter>

二、自动实现设计

1. 选 Translate→XEMAKE→Done→UART. SCH

产生以下报告文件:

uart. err	选配错误报告
rcvr. err	汇编错误报告
rcvr. lga	汇编逻辑报告
uart. res	资源报告
uart. map	映射报告
uart. pin	引脚清单报告
uart. lgc	逻辑优化报告
uart. par	分割报告

2. 保存引脚分配, 选 Translate→PINSAVE→UART. VMH
3. 生编程文件, 选 Verify→AMKEPRG→UART. VMH

§ 11. 5. 6 仿真设计

一、原理图放置激励和追踪数据

1. 由 XDM 菜单, 选 DesignEntry→ORCAD;
2. 选 Schematic Design Tools→Execute→Draft→Execute, 显示 uart 原理图;
3. 选 Place→Stimulus, 在提示 Stimulus?后键入:"0;0 1500;1 2000;G;0" <Enter>;
4. 移指针到时钟输入 X4CLK 的网线处, 选 Place, 小的波形标记出现在网线上。
5. 选 Place→Trace Name, 在提示 Trace Name?后键入"X4CLK";
6. 指针移到激励数据附近, 选 Place;
7. 用 Esc 键退出 Place Trace 指令, 选 Quit→Update File→Abandon Edits;
8. 在 To Main 上按鼠标两次, 在 Exit ESP 上按鼠标两次, 按任意键返回 XDM。

二、自动产生时序仿真网表

1. 选 Verify→XSIMMAKE, 选 -F 选项, orcad_Epld_Timing 作为运行流;
2. 然后选 Done, 从显示的可用输入文件清单中选 UART.VMH。

三、进入 OrCAD 仿真器

1. 由 XDM 选 DesignEntry→ORCAD;
2. 由 OrCAD 主菜单选 Design Management Tools→Execute;
3. 在左边目录清单选 UART, 然后在 OK 按钮按动鼠标;
4. 由 OrCAD 主菜单选 Digital Simulation Tools→Execute。

四、配置 OrCAD 仿真器

1. 选 Simulate, 选 Local Configuration→Configure SIMULATE;
2. 在文件选项的 Connectivity database 字段按鼠标, 再对话框键入 "UART.VST" <Enter>;

出现如图11.17所示的文件选项。

File Options	
Connectivity Database	UART.VST
Stimulus File	UART.STM
Trace File	UART.TRC

图11.17 配置仿真器的文件选项

3. 在处理选项的 Use Delay Annotation 上按鼠标, 在 OK 按钮上按鼠标。

五、使用激励编辑器(设已编辑 uart.stm)

1. 选 Simulate→Execute 开始仿真对话期间, 出现除左边的信号清单外的空白波形显示。
2. 为观看激励设置文件 uart.stm, 选 Edit Stimulus, 出现提示:
Editing Stimulus will INITIALIZE simulator. Continue?
3. 选 Yes(出激励编辑屏幕)→SDIN→Edit(出激励细节编辑屏幕)→Return;
4. 选 X4CLK→Edit(如图11.20)→Return;
5. 选 Use Stimulus, 返回波形屏幕。

六、使用追踪编辑器(设已编辑 uart.trc)

1. 选 Trace→Trace Edit, 观看追踪设置文件 uart.trc;
2. 观看信号细节, 选该信号, 再选 Edit, 显示 uart.trc 文件中此信号细节;
3. 选 Use Trace, 返回波形的屏幕。

七、运行仿真

1. 为了在一次看更多的信息,选 Trace→ Change View;
2. 在提示 Trace Delta Time?后键入"1100"<Enter>,波形图将显示1100ns;
3. 选 Run Simulation,在提示 Simulation Length?后键入"110000"<Enter>,将仿真1100ns。
4. 观看仿真波形,校正矢量错误。

八、编辑激励

1. 选 Edit Stimulus,出提示,选 Yes,仿真屏幕上选 SDIN,选 Edit;
2. 在激励细节编辑屏幕上 Time 的任一处,选 Add. 出提示,键入"80500"<Enter>,出提示,选1插入一上升沿;
3. 重复此程序,在时间83500 时 SDIN 返回到0,选 Return→ Use Stimulus,返回波形图;
4. 选 Run Simulation,键入"110000"<Enter>,此过程消除仿真的极性错误,但仍有信息段错误,用上述方法在时间95500处置 S. DATA. IN 的值为0来消除。

九、加信号到波形显示

1. 选 Trace→ Trace Edit→ Add→ Edit→ "C0"<Enter>;
2. 在 Signal Name 字段选 Browse,列出所有信号清单,用 PageDown 键找到 C0_1信号;
3. 选 C0_1→ Return→ Use Trace→ Initialize→ Yes;
4. 选 Run Simulation,键入"110000"<Enter>,检查仿真结果。
5. 选 Quit→ Abandon→ Simulation→ Yes,返回数字仿真屏幕;
6. 选 To Main→ Execute,返回 OrCAD ESP 屏幕。

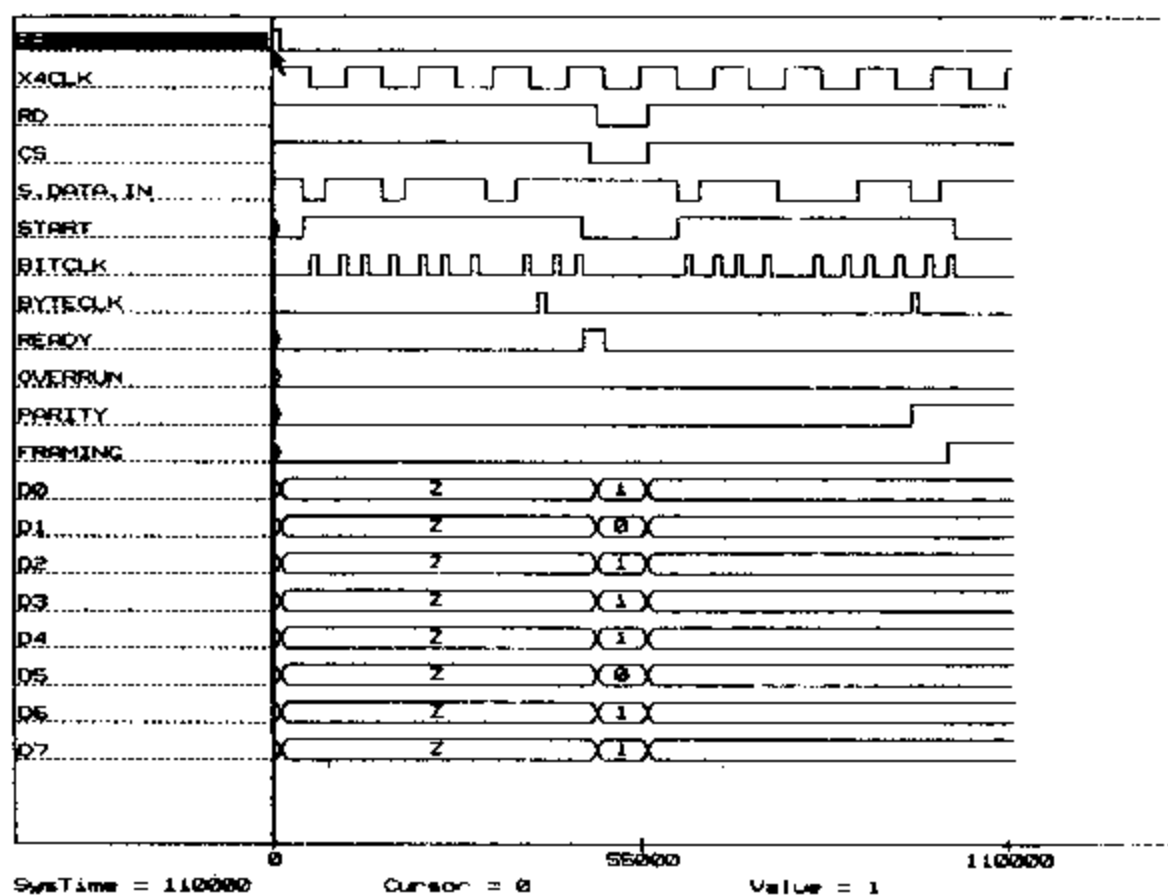


图11.18 仿真结果的波形

第十二章 FPGA 设计应用

§ 12.1 FPGA 设计应用实例

一、多数逻辑电路和奇偶检测电路

如图12.1(a)表示了一个五输入的多数电路,此函数的输出当五个输入中的任意三个变为高时将变为高,当用积之和的形式写出此函数时,它有16个乘积项。在大多数 PAL 中宏单元仅仅有八个乘积项,因此将需要多级的 PAL 逻辑来构造这个函数。此外,不管这个函数是如何地复杂,当它有五个输入和一个输出时,它总是可放进一个逻辑块,并且这个函数对 Xilinx 最快的器件将有2ns 的延时。

图12.1(b)表示了一个七输入的多数逻辑电路,这个电路有64个乘积项,但被放进两层的 Xilinx 逻辑中。

图12.1(c)是九输入的奇偶检测电路,奇偶检测电路是与正在检测的奇或偶极性有关的大型“异或”或“异或非”电路。当为 PAL 或 EPLD 写出这个函数的积之和形式,它需要128个乘积项,将需要多层的 PAL 逻辑,也就会浪费许多宏单元,但同样的函数可放进两个 FPGA 的逻辑块,在 Xilinx 的两层逻辑块中可安排高达25个输入的逻辑函数。

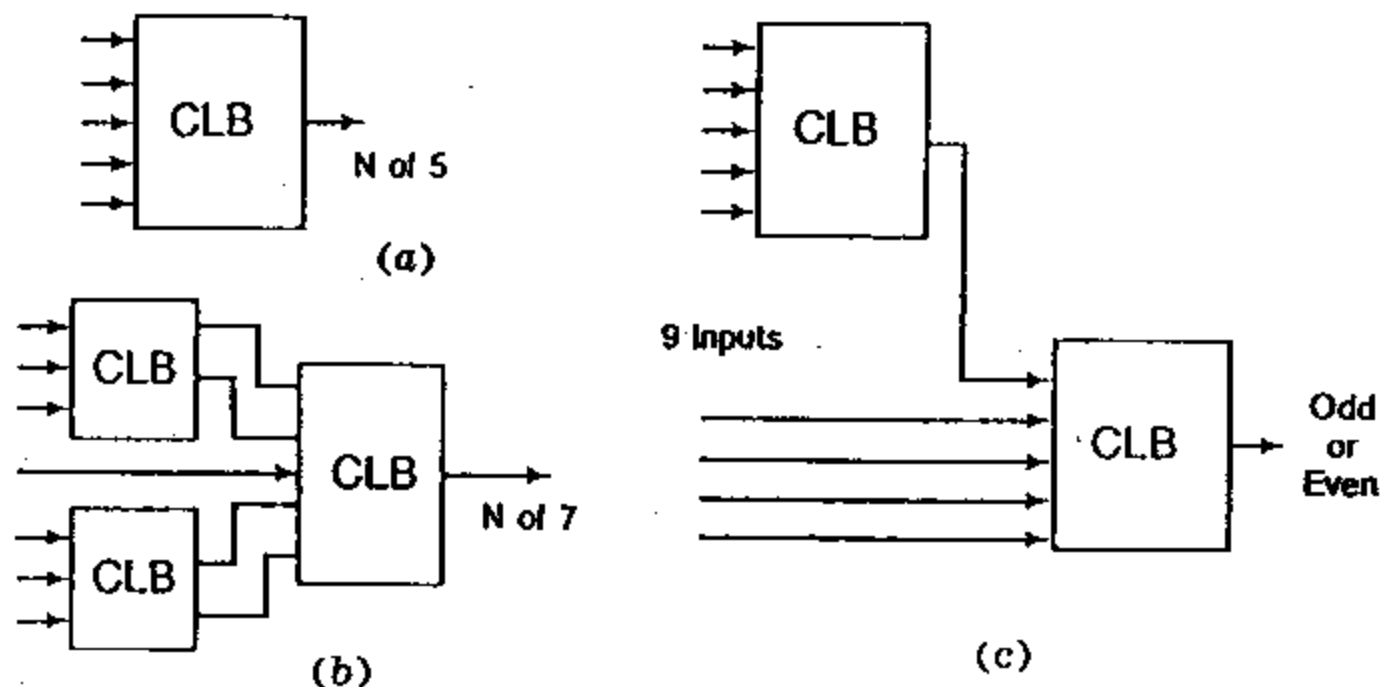


图12.1 多数逻辑电路(a)(b)和奇偶检测电路(c)

二、相加器

图12.2表示一些简单的相加器电路,图12.2(a)中简单的两位相加器有五个输入,和三个输出,因为可以把任何五输入的函数放进一个逻辑功能块,这整个函数将需要三个逻辑功能块,每个相加器输出要一个逻辑功能块。

图12.2(b)为一个四位加法器,因为这个函数有多于五个的输入,它要求多于一层的逻辑块,所以加法器的前两位和它的进位输入在第一层中计算,后两位和进位输出在第二层逻辑块中计算,当每个 CLB 的延时为5ns 时,不包括 I/O 的延时,此函数的延时约15ns。

多位的加法器要求更多层的逻辑块和需要更多的延时,有一些技巧可用于构造极其快速的多位的加法器。在 LCA 器件中有许多方法实现二进制的加法器,减法器 and 累加器,各种方法提供了大小和速度之间的取舍,最紧凑但最慢的是串行位加法器,美疙时钟运行1或2位,产生和及进位,和加到输出移位寄存器,进位存储到下一位时使用;最紧凑的并行组合加法器是由级联的 CLB 组成,每个 CLB 实现一个全加器,接收每个操作数的一位和输入进位,产生和及输出进位,16位加法器要求16个 CLB,在16个 CLB 延时中完成;用 XC3000系列的五输入函数发生器,一个 CLB 一次实现两位加,三个 CLB 可管理每个操作数的两个输入位及一个输入进位,产生两个和及输出进位,16位加法器要求24个 CLB,但在8个 CLB 延时中实现。XC3000系列中实现16位超前进位加法器,用30个 CLB 仅5个 CLB 延时实现;而16位条件和加法器为41个 CLB,3个 CLB 延时。

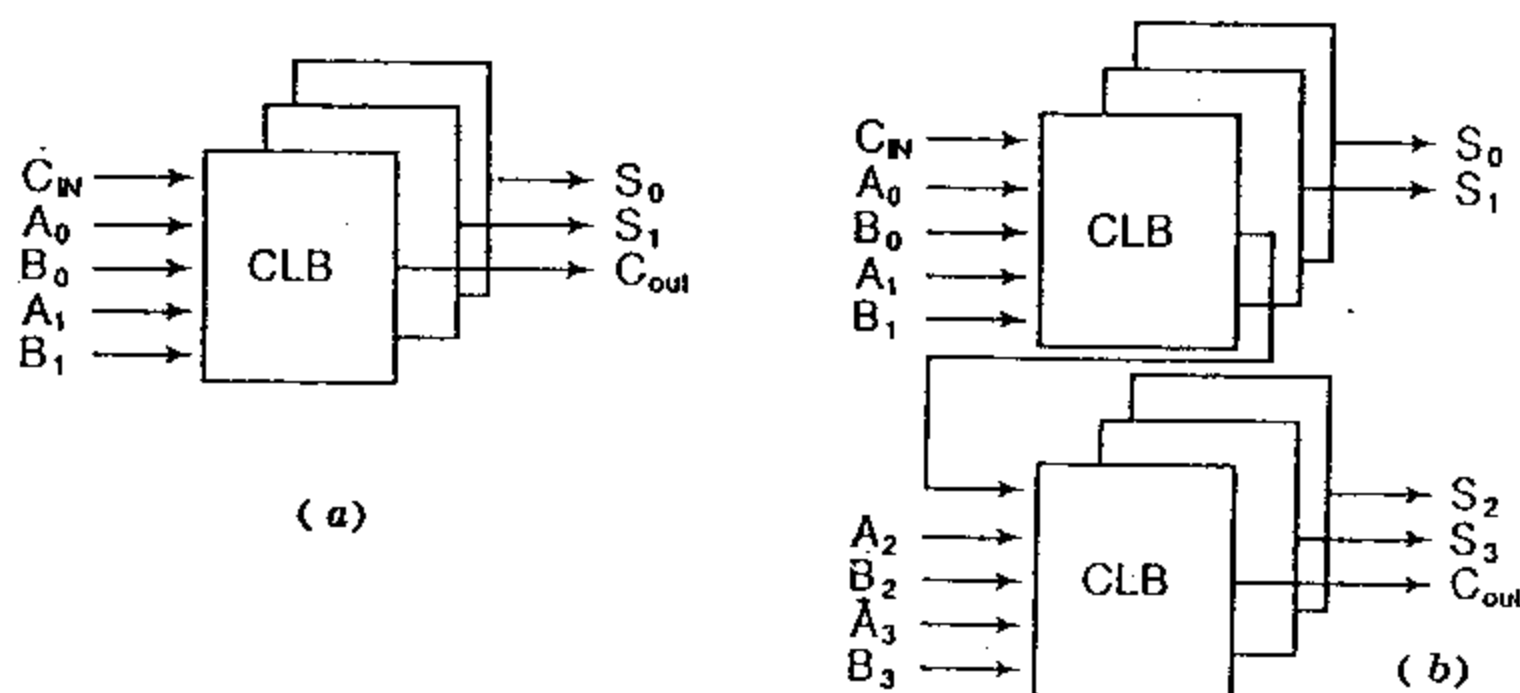


图12.2 相加器:(a)两位加法器(b)四位加法器

上面两个例子是组合逻辑的应用,下面介绍包括触发器的应用例子。

三、带有保持的左/右移位寄存器

图12.3表示了仅仅两位的带有保持的左/右移位寄存器,但是它可以扩展到任意长度的移位寄存器。这个移位寄存器的两位刚好可以放进一个逻辑功能块(CLB)中,以此为例,在 XC3020器件中,可以构造一个128位的移位寄存器。

XC3000系列的 CLB 可以分成两部份,每部份可以构造任何四输入的函数,图12.3所表示的逻辑控制哪一个数据加载进触发器,数据或者来自右边的触发器,或者来自左边的触发器,LFT/RT 信号控制移位的方向,当保持信号(Hold)为低时,移位寄存器的值被保持。

XC3000系列—125档次的器件类似这样的移位寄存器可以运行在超过90MHz,—150档次的器件可达到115MHz。

四、四位计数器

图12.4表示四位任何可能形式的计数器,利用 XC3000系列的 CLB 可以实现两个四变量的函数的灵活性,用两个 CLB 可以实现四位二进制计数器,四位 BCD 计数器,四位格林码计数器等,—150档次的器件可达到75—80MHz。

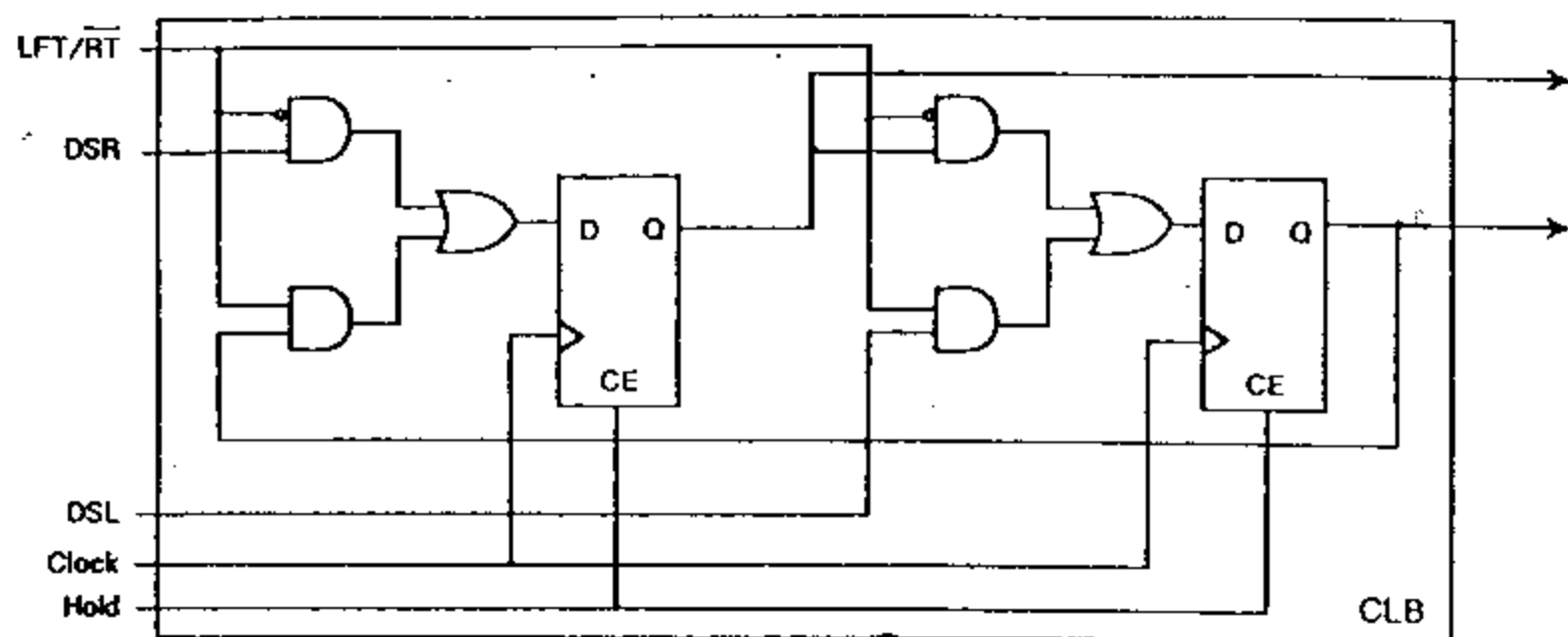


图12.3 带有保持的左/右移位寄存器(二位/CLB)

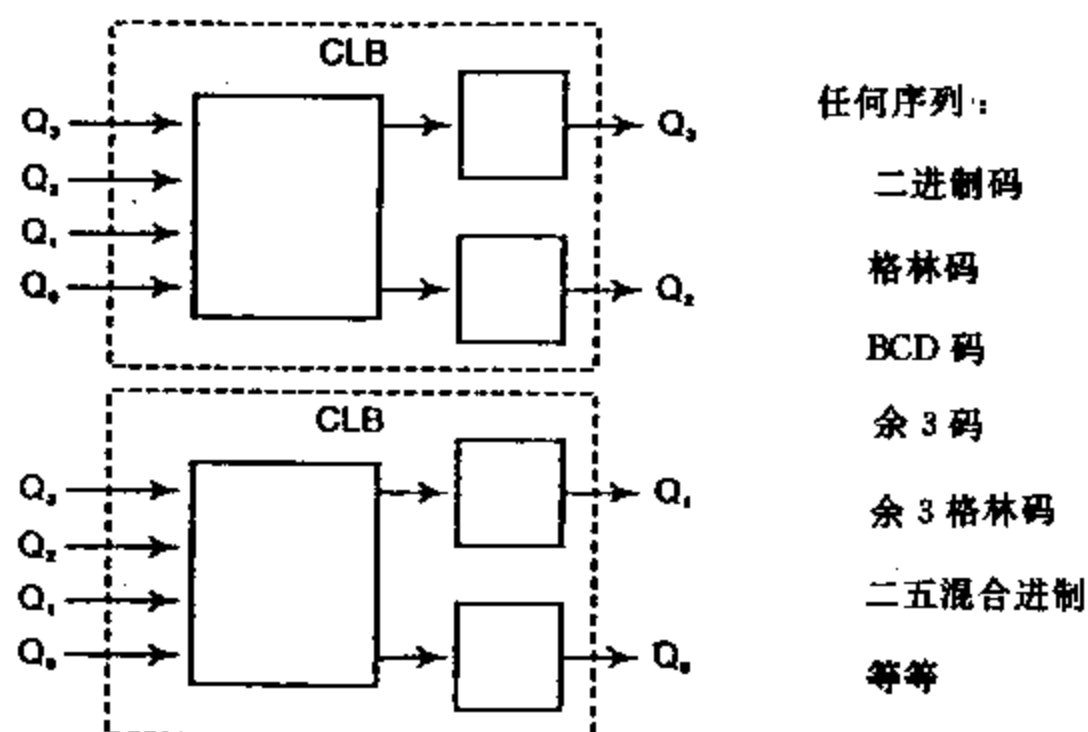


图12.4 四位计数器的应用例子

五、三态缓冲器

Xilinx FPGA 器件的可编程资源中,长线通常用于时钟信号,内部三态数据总线和任何高扇出信号。实际结果表明水平长线可以用来构造这些总线,内部总线可以减少要求的 I/O 腿的数目,因此可以把整个印制板上的逻辑放进单个器件内,带有内部总线能力是 Xilinx FPGA 的特点之一,不同规模的器件有不同可能的内部数据总线的尺寸,例如,XC3020可达到16位内部总线,而 XC3090可达到40位的内部总线。

总线的作用就像一个多路开关,如图12.5所示,当要把信号 A 放到总线上,就要使选择信号 Sel A 处于低电平,它会打开三态缓冲器,驱动信号 A 到总线上。如图12.5(b)所示,软件自动地加一个弱的保持电路(Weak Keeper),因为在无任何信号驱动总线时,则 CMOS 处于悬浮状态,这就可能引起额外的功耗,甚至出现自由振荡,弱保位电路可以避免总线处于悬浮状态,而维持在总线的最后数值上。

利用内部三态数据总线也可以实现“线与”,或者“线或”。当把三态缓冲器的输入连接到地,并驱动使能信号,可以使三态缓冲器看作漏极开路的缓冲器,再把漏极开路的缓冲器连接在一起,并接到每条线的上拉电阻上,就可实现线与的功能。在图12.5中,当 A,B 和 C 都为高时,三态缓冲器呈现高阻,上拉电阻把长线拉到高电平;当 A,B 和 C 被驱动为低时,则相应的缓冲器被打开,驱动整个长线为低,所以这个电路实现线与的功能。

必须记住,Xilinx 所用的所有三态缓冲器是低有效输出使能的,尽管符号上看起来像高有效,输出使能总是低有效的。

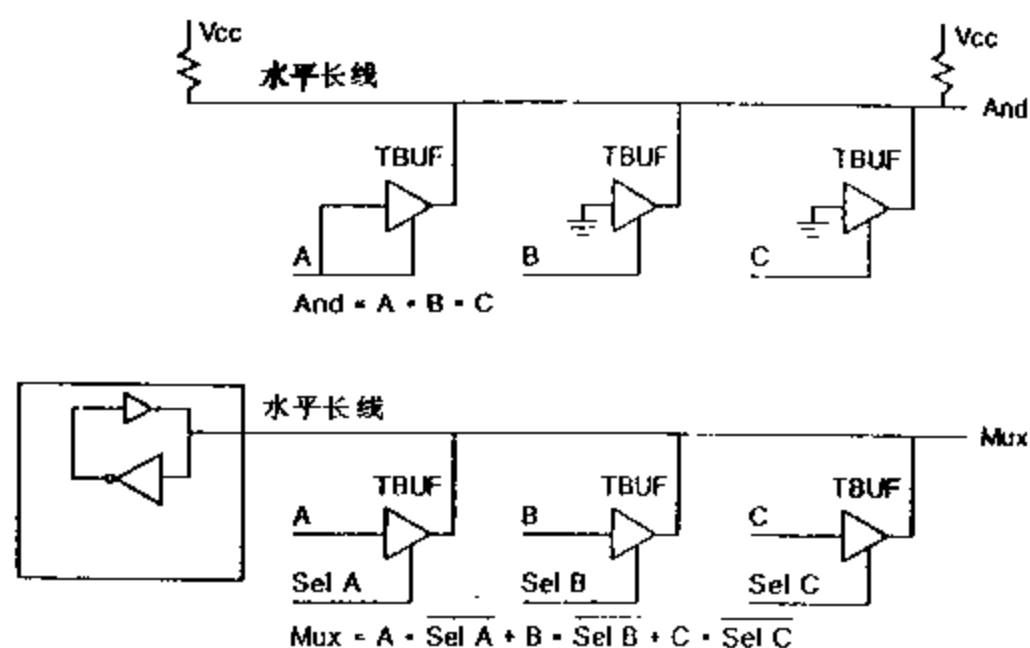


图12.5 三态缓冲器(a)线与(b)多路转换器

六、同步电路设计

Xilinx 的 FPGA 结构对同步电路设计是最适合的,严格的同步设计保证所有的寄存器由无时钟畸变的相同时基驱动,这节提供几个产生高性能同步设计的要点。

1. 全局时钟的分配

Xilinx 的 FPGA 时钟网络保证极小的时钟畸变,表12.1为各系列的全局时钟资源。

2. 其它的同步设计考虑

利用时钟使能代替门时钟来控制数据锁存到寄存器,如图12.7所示。

如果设计需要比全局分配时钟网络更多的时钟,应重新设计来减少时钟数,否则,把有最低扇出的时钟放到通常的布线网线上,给它们一个高的网线加权,但要记住用通常的网线布线时钟网线有畸变。

3. 数据反馈和时钟使能

在图12.6(a)的电路中,一个选通的时钟时序图表示这个实现可以导致能引起触发器在

错误时间计时钟的时钟干扰。

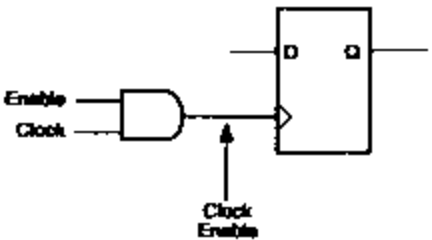
表12.1 全局时钟资源

FPGA 系列	资源	数量	指定出腿
XC2000	ACLK	1	时钟或控制
XC2000L	GCLK	1	时钟或控制
XC3000/A/L	ACLK	1	时钟
XC3100/A	GCLK	1	时钟
XC4000	BUFGP	4	时钟
XC4000A/H	BUFGS	4	时钟或控制

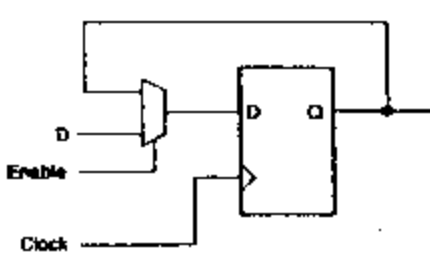
注, XC4000的 BUFGP 和 BUFGS 也接到控制腿和逻辑输入,但限定的布线资源可增加额外的延时到这些负载

图12.6(b)表示用数据通道同步替代选通的时钟,触发器在每个时钟周期被定时,数据通道由一个使能信号控制,当使能信号为低时,寄存器输出反馈到多路开关自身;当使能信号为高时,新数据进到触发器,寄存器改变其状态,这个电路保证最小的时钟脉冲宽度,它不给时钟加畸变,如图所说明的, XC3000和 XC4000触发器有内部时钟使能。

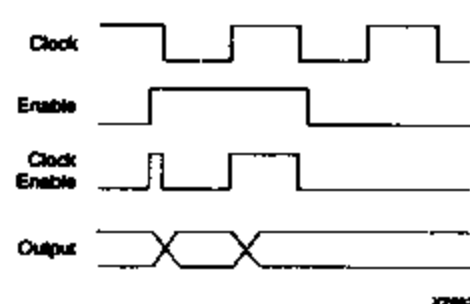
选通的时钟



利用反馈通道



相应的时序图



相应的时序图

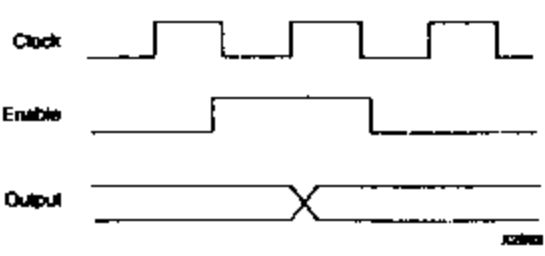
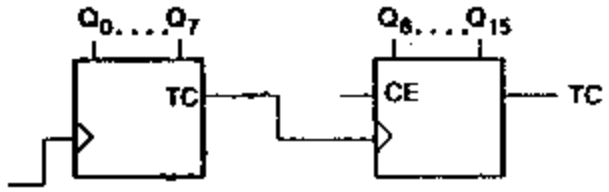


图12.6 (a) 选通的时钟

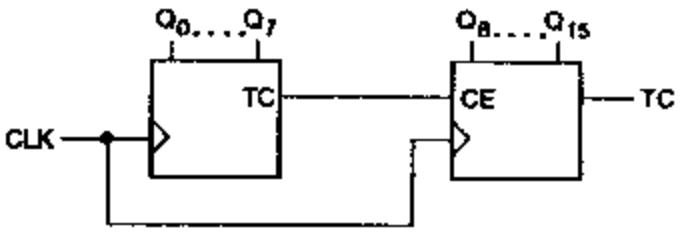
(b) 利用数据反馈的同步设计

4. 计数器

级联几个小的计数器产生大的计数器类似于选通的时钟。例如,当两个八位计数器被连接,第一个计数器的 TC 是一个大的与函数选通第二个时钟输入,利用 CE 的输入,可以产生一个如图12.7所示同步设计,在此情况,第一级的 TC 被直接接到第二级的 CE。



TC 接 CLK 的 16 位计数器(为不适当方法)



TC 接 CE(时钟使能)的 16 位计数器(推荐方法)

图12.7 两个八位计数器连接成一个16位计数器

§ 12.2 Viewlogic 软件设计 Xilinx FPGA

在介绍 Viewlogic 软件之前,先对本节中所使用的各有关符号作一说明:

例: Window→Open→Viewdraw→Schematic: →为 Viewlogic 菜单的依次选项

例: 加框名(text): ()内为 Viewlogic 在指令窗口的键盘输入指令,可代替菜单选项

例: {Schematic name[]}: {}内为 Viewlogic 指令窗口的提示信息

例: "andblk2" <Enter>: " "内为键盘输入内容

§ 12.2.1 Viewlogic 软件概述

ViewLogic 软件包括电路图输入软件—Viewdraw (DS390)和仿真软件—ViewSim (DS290),下面先给出 ViewDraw 的一般要求。

1. Xilinx FPGA 的设计流程如图12.8所示。

2. 软件要求: Workview4.13版本或更高

Xilinx/ViewDraw 接口, WIR2XNF V5.00 (DS390)

XDM V5.00 (DS502)

3. PC 机上装配完软件后,校验以下文件:

Autoexec.bat 文件应包括:

```
set XACT=c:\XACT
```

```
set WDIR=c:\workview\standard
```

```
set SYSPLT=c:\workview\standard
```

```
PATH . . . . ;c:\xact;c:\workview;. . . .
```

Config.sys 文件应包括:

```
files=25
```

```
buffers=25
```

```
rem make extra space to accommodate more environmental variables
```

```
shell= c:\dos\command.com c:\dos /e:720 /p
```

```
device=c:\qemm\qemm386.sys ram
```

4. ViewDraw 的目录结构如图12.9:

sch 目录包括原理图的图形描述文件;

sym 目录包括产生和保存的符号图形描述文件;

wir 目录原理图的网表描述文件。

5. Viewdraw.ini 文件:

建议在每个设计目录中保存初始化 ViewDraw 的文件 Viewdraw.ini。

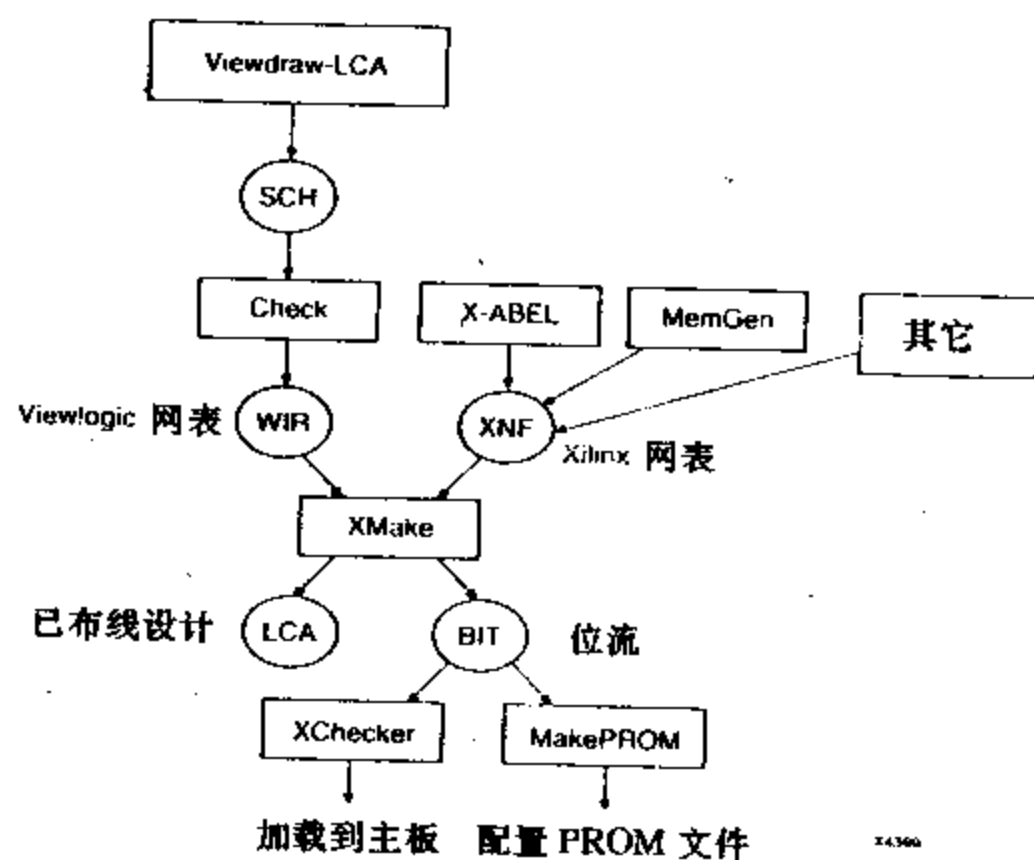


图12.8 FPGA 设计流程

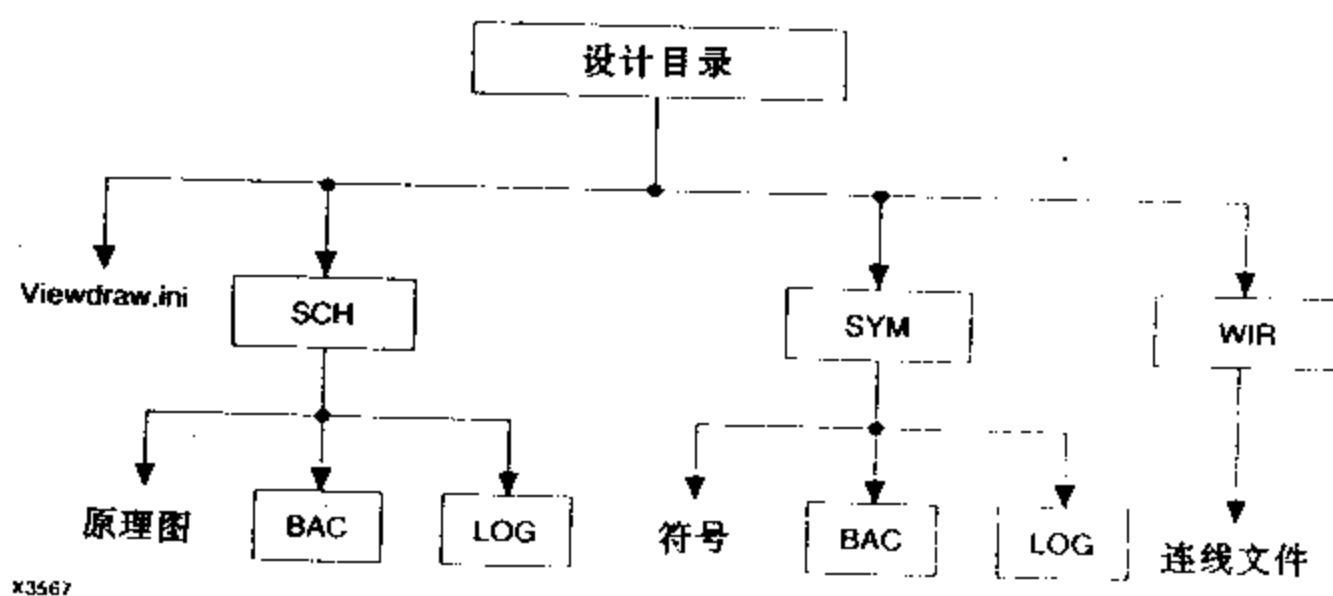


图12.9 ViewDraw 目录结构

表12.2 Viewdraw 库搜索次序

Type	Directory	Directory
P	/project_directory	
m	/workview/unified/shm4000	shm4000
m	/workview/unified/xc2000	xc2000
m	/workview/unified/xc3000	xc3000
m	/workview/unified/xc3000o	xc3000o
m	/workview/unified/xc4000	xc4000
m	/workview/unified/xblox	xblox
m	/workview/unified/builtin	builtin

Type	Directory	Directory
p	/project_directory	
m	/workview/unified/xc7000	xc7000
m	/workview/unified/builtin	builtin

```

| ViewDraw initialization file for Version 5.0
|
| Copyright (c) 1985 1986 1987 1988 1989 Viewlogic
| Systems, Inc.
| Set the default display characteristics for each object
| Object Color Fillstyle Linestyle
| -----
NET          4      1      0 | RED   SOLID SOLID
COMPONENT    15      0      0 | WHITE HOLLOW SOLID
ATTRIBUTE    14      0      0 | YELLOW HOLLOW SOLID
LABEL        15      0      0 | WHITE HOLLOW SOLID
PIN          11      0      0 | LT. CYAN HOLLOW SOLID

BOX          2      0      0 | GREEN HOLLOW SOLID
LINE         2      0      0 | GREEN HOLLOW SOLID
CIRCLE       2      0      0 | GREEN HOLLOW SOLID
ARC          2      0      0 | GREEN HOLLOW SOLID
TEXT        10      0      0 | LT. GREEN HOLLOW SOLID

| set the display characteristics for the selection layer
SELECTION_LAYER 15      0      0 | WHITE HOLLOW SOLID

| set the display characteristics for the border layer
BORDER_LAYER   15      0      0 | WHITE HOLLOW SOLID

| set the display characteristics for the value layer
VALUE_LAYER    7      0      0 | LT. GREY HOLLOW SOLID

| set the display characteristics for the annotation layer
ANNO_LAYER     7      0      0

| Default values
GRID           10      | Grid spacing of 10 units
DOTSIZE        5      | Solder dot radius of 5 units
BUS_DOTSIZE    12      | Bus solder dot radius of
                    | 12 units
BOXSIZE        5      | Dangling joint box radius of
                    | 5 units
TEXTSIZE       10      | Text size of 10 units high
TEXTORIGIN     3      | Text origin of lower left
BUSWIDTH       4      | Bus half-width of 4 units
BUBBLESIZE     5      | Bubble size on pins of 5 units
SHEETSIZE      1      | BSIZE sheet

| SHEETSIZE 10 200 200 | ZSIZE sheet 200 by 200
AUTOLOG        10      | Autolog value of 10
SDISTANCE      10      | Selection distance of 10 pixels
ADISTANCE      20      | Avoidance distance of 20 units
|ROUTE 0       | Straight net routing
|ROUTE 1       | Orthogonal net routing

```

图12.10 Viewdraw.ini 实例

ROUTE	2	Avoidance net routing
SCOPE	1	Local scope for labels
SCOPE 1		Global scope for labels
TEXT_THRESHOLD	3	Threshold of 3 pixels high
NEW_ATTR_VIS 2		Upon creation, new attr's name
		will be visible.
NEW_ATTR_VIS 3		Upon creation, new attr's value
		will be visible.
NEW_ATTR_VIS	1	Upon creation, new attr's will
		be totally visible.
BLOCKTYPE	0	default block type is composite
BLOCKTYPE 1		default block type is module
BLOCKTYPE 3		default block type is annotate
BLOCKTYPE 4		default block type is pin
Default modes		
GRIDON	1	Grid display on
BORDERON	1	Border display on
HEADERON	1	Header display on
COMPTXTON	1	Component text, labels and
		attributes on
TEXTON	1	Text on
ATTRON	1	Attributes on
LABELON	1	labels on
DETAIL	0	detail off
BELL	0	bell off
SNAPTOPIN	1	snap to pin on
UNIQUE_LABEL	0	create unique labels on copy off
VALUESON	1	display of simulation values off
CONTEXT_WINDOW	0	display of context window off
NAMESON	0	display of internal names off
SORTON	1	alphabetically sort dialog
		entries
PNUMSON	1	display of pin numbers on
RNUMSON	1	display of refdes on
DEFSHEET	0	sheet template for new sheets
		off
XTRAERRS	1	print out all errors in checker
DBOXON	0	pick names from dialog boxes off
NETNAME VDD		Upon creation, VDD will have
		global scope.
NETNAME GND		Upon creation, GND will have
		global scope.
SYMBOL_QUALIFIERS \D		Demorgan
SYMBOL_QUALIFIERS \C		Conventional
SYMBOL_QUALIFIERS \A		Ansi
SYMBOL_QUALIFIERS \P		Package
DIR [p] .		(primary)
dir [m] \workview\xc4000		(xc4000)
dir [m] \workview\xc4000o		(xc4000o)

图12.10 Viewdraw.ini 实例续

§ 12.2.2 由 XDM 到 WORKVIEW

1. 启动 XACT 设计管理软件(XDM)

cd workview\calc→XDM

2. 进入 Workview—Viewlogic 的设计环境

DesignEntry→Workview

3. 如何利用鼠标:

运行 Viewlogic 必须有三按钮串行鼠标

左键:选择目标,一次一个

中间键:执行指令

右键:取消指令,选择多个目标

4. 如何利用功能键

F1:与鼠标左键功能相同(选择)

F6:Pan(View→Pan)

F2:与鼠标中间键功能相同(执行)

F7:放大二倍(View→In)

F3:与鼠标右键功能相同(取消,选多目标)

F8:缩小二倍(View→Out)

F4:Scroll(View→Scroll)

F9:放大一个区域(View→Zoom)

F5:Refresh

F10:全图显示(View→Full)

5. 如何从菜单选择指令:鼠标指针移到最左边,菜单会依次出现

6. 在 Viewfile 中产生一个设计项目

Window→Open→Viewfile

{Specify area for opening window. },用鼠标开窗口

Project→Create

键入完全的路径名,例:"workview\CALC"<Enter>

7. 检查 Project 的路径和单元库

project→search→viewdraw

出现一个对话框,选正确的目录,指针移到 ACCEPT 处,按鼠标左键

出现类似表12.2的库搜索清单,如果路径不正确,应在 DOS 下修改 viewdraw.ini 文件;缺少单元库可在空行中按表12.2的形式输入,直到路径和库正确后,移指针到 ACCEPT 处,按鼠标左键。

8. 设置 Project 目录

set→Project

出现一个对话框,选正确的目录,指针移到 ACCEPT 处,按鼠标左键

{current project:xxxxxx}; Window→Close

9. 打开原理图窗口

Window→Open→Viewdraw→schematic

{Schematic name:},"CALC"<Enter>,等待提示后按鼠标两次打开窗口,出现顶层原理图。

10. 退出原理图

Window→Close

§ 12.2.3 产生一个符号

1. Window→Open→Viewdraw→Symbol

2. {Symbol name[]:}, "andblk2" <Enter> (省略第一步, 直接键入 sym andblk2)

3. 改变图纸尺寸

Change→Block→Sheet→Z-WxH

{Block width[100]}, "150" <Enter>, 改变宽;

{Block height[100]}, "120" <Enter>, 改变高。

4. 产生边框符号(box)

Add→Box, 用鼠标画出边框

5. 加出腿到符号(pin)

Add→Pin, 指针指到内部边框线要放置出腿处, 按动鼠标两次, 即加上出腿。

6. 标记符号的出腿(lab)

Add→Label, 用鼠标左键选要标记的出腿, 用中键执行 Label 指令后, 键入相应的标记。允许后键入 Y, 标记出现在屏幕上。例: andblk2 的标记有 A[3:0], B[3:0], O[3:0]

7. 加属性(cat)

Change→Attr→Dialog→All, 一个对话框出现在屏幕上, 在 Attr name 下键入 pin-type; 在 Attr Value 下键入 in, 在 Accept 处按鼠标后, 符号上出现 "PINTYPE=IN"。

8. 改变属性尺寸(siz)

Change→Text→size

9. 关闭属性(ai)

Change→Attr→Invisible, 用 F5 刷新屏幕。

10. 加框名(text)

Add→Text, 按鼠标中键(执行 Text 指令), {Text string:}, "ANDBLK2" <Enter>, 放置到合适位置上, 按鼠标中键。

Change→Text→Size 改变框名大小

Change→Text→String 修改框名。

11. 保存符号(wri)

File→Write

12. 由 ANDBLK2 产生 ORBLK2

由于两者类似, 利用 ANDBLK2, 仅修改其框名, 用鼠标左键选框名字符串 "ANDBLK2", 出现加亮框

Change→Text→String, {Text String:ANDBLK2}, 删除 ANDBLK2, "ORBLK2" <Enter>

File→writeto, {symbol name[primary:ANDBLK2.1]:}, "orblk2" <Enter>, 修改的符号作为 ORBLK2 被保存。

Window→Close, {Are you sure(y/n)? DRAW: Symbol ANDBLK2 modified. }, "Y" <Enter>

§ 12.2.4 产生原理图

1. 产生低一级原理图(sc andblk2)

Window→Open→Viewdraw→Schematic, {Schematic name[]:}, "andblk2" <Enter>
> 用鼠标中键定区域, 打开 ViewDraw 原理图窗口

2. 加元件到原理图(com and2)

Add→Comp, 按鼠标中键, {Symbol name:}, "and2" <Enter>

一个矩形框出现在指针处, 用鼠标左键看到符号, 移到合适位置, 用鼠标中键确认。

用鼠标中键重新激活, {Symbol name:}, <Enter>, 又出现 AND2的框, 移到新位, 用鼠标中键放置。

3. 拷贝元件(cop)

用鼠标左键选要拷贝的元件, 出现加亮框, 选 Copy, 按鼠标中键, 移位按中键放置。

4. 移动元件(m)

用鼠标左键选要拷贝的元件, 出现加亮框, 选 Move, 按鼠标中键, 移位按中键放置。

5. 加连线(net)

Add→Net, 移指针到起点位置, 按鼠标中键, 再移指针到终点位置, 按鼠标中键, 再按鼠标右键。

* 连接两点: 按中键(起点), 按中键(终点), 连线自动找平直。

* 产生悬挂线: 按中键(起点), 按中键(终点), 按右键(出现小方块)。

6. 删除连线(del)

用鼠标左键选要删除的连线使此线加亮, 选 Delete; 对连线每一段重复以上步骤。

7. 加总线到原理图(bus)

Add→Bus, 移指针到起点位置, 按鼠标中键, 再移指针到终点位置, 按鼠标中键, 再按鼠标右键, 终止总线。

8. 标记连线

用鼠标左键选要标记的连线, 出现加亮线, 选 add→Label, 按鼠标中键, {Label text string:}, 例: "A0" <Enter>。

一个空框出现在指针处, 用鼠标左键看到标记, 移到合适位置, 用鼠标中键确认。

9. 标记总线

接到总线的所有连线和总线, 都需加标记, 以下为合法总线名的例子:

Q[0:7]; 八位总线, MSB: Q0—LSB: Q7

Q[7:0]; 八位总线, MSB: Q7—LSB: Q0

Q[7:0], SET, CLK; 十位总线, MSB: Q7—LSB: Q0, SET 和 CLK

A[7:0], B[7:0]; 十六位总线, A7—A0, B7—B0

DATA[0:7:2]; 四位总线, DATA0, DATA2, DATA4, DATA6

DATA[0:F/H]十六位总线, 按16进制, 也可以10, 8或2进制。

10. 拷贝原理图

作为新图保存来实现拷贝, File→Writeto, {Schematic name [Primary: ANDBLK2]:}, "orblk2" <Enter>

Window→Close

11. 完成 ORBLK2设计

将原图中 AND2改为 OR2,用鼠标左键选要改的 AND2,出现加亮线

选 Change→Comp, {Symbol name [xc3000:and2.1]}, "or2" <Enter>, OR2代替 AND2

重复以上步骤,完成修改。

File→Write

Window→Close

12. 完成 ALU 原理图:

Window→Open→Viewdraw→Schematic, {Schematic name:}, "alu" <Enter>

等待提示后按鼠标两次打开窗口,出现顶层原理图。

Add→Comp, {Draw:component}, 按中键, {Symbol name:}, "andblk2" <Enter>

用鼠标左键看到符号,移到合适位置,用鼠标中键确认。

类似上述步骤加 ORBLK, fd4ce

参考 ALU 原理图,加连线,总线和标记

File→Write

用鼠标左键选要看的图(符号,宏单元), Level→Push→Schematic

Level→pop, 返回到 ALU

由库单元组成的图可下拉看, Level→Push→Symbol, Level→Pop 返回。

§ 12.2.5 VIEWSIM 使用说明

1. "XDM" <Enter>; DesignEntry→Workview

2. Window→Open→Viewdraw→Schematic, {Schematic name:} "calc" <Enter>

3. 在屏幕的上半部开窗口显示原理图

4. 修改 WIR 网表文件(mex chkprj)

Window→Open→Utility→Workview→Check→Project

加载的原理图(例:CALC)自动作为输入,给出其校验输出

5. 用 VSM 产生 ViewSim 网表清单(mex vsm)

Export→Wirelist→Viewsim

VSM 完成后,给出信息 {0 error and 0 warnings. Press any key to continue. }

6. 为仿真选连线

用 F9放大原理图被选部份,指针指到标有 CLK 的连线,按左键,连线和标记变亮

重复上述步骤选其它连线

用右键选以下连线:WE,RST

用右键选以下标记:EXC_P, SW[6:0], ALU[3:0], STACK[3:0]

7. 打开 ViewSim 窗口(simu calc)

Window→Open→Viewsim s/d

{Network file name[CALC]:}, <Enter>

{Specify Viewsim window}, 在屏幕下半部开 Viewsim 窗口

8. 设置 ViewWave 流(s_wave)

Setup→Vwave, {Viewwave stream name[CALC]:}; <Enter>

{Node(s)/Vector(s)[selected nodes]:}; <Enter>

{Viewsim active...SIM; wave calc. wfm CLK WE RST EXc_P SW[6,0]ALU[3,0]STACK[3;0]}, 信息后半部份可能看不到

9. 打开 ViewWave 窗口(wvsim calc)

Window→Open→Viewwave→Viewsim

{Enter stream name[CALC. WFM]:}; <Enter>

{Specify area for opening window }, 在 Viewdraw 窗口按中键两次, 出现波形窗口

10. 分配时钟数值

指针移到 Viewsim 窗口, 按左键重新激活它, {viewsim active}

Viewsim 窗口中键入 "clock CLK 0 1" <Enter>

"step" > Enter >, {Stepsize=100ns. }, 此时时钟周期为200ns

"clock newclk 0 0 1 1" (周期为400ns), step 50ns, (不给定 ns, 缺省值为0.1ns)

11. 定义矢量, 分配初始值

"vector sw sw[6:0]" ; "assign sw 000000" (置数据全部为零);

"h exc_p" (置执行开关为高)

12. 置全局复位

对 XC2000/XC3000: "l gr" <Enter>, "sim 1000" <Enter>

{Simulation stopped at 100.0ns}, "h gr" <Enter>

对 XC4000: "h gsr" <Enter>, "sim 1000" <Enter>

{Simulation stopped at 100.0ns}, "l gsr" <Enter>

13. 仿真此电路

加载11111到 ALU 的寄存器

"assign sw 1101111" <Enter>

"l exc_p" <Enter> "cycle 2" <Enter>

"h exc_p" <Enter> "cycle 3" <Enter>

把寄存器值放到堆栈

"assign sw 1111011" <Enter>

"l exc_p" <Enter> "cycle 2" <Enter>

"h exc_p" <Enter> "cycle 4" <Enter>

指针移到 Viewwave 窗口, 按左键激活它, {viewwave active}, 显示仿真波形。

14. 原理图中检查仿真值

Window→Push, 下拉 ViewWave, 上托 ViewDraw 窗口

指针移到 Viewdraw 窗口, 按左键重新激活它, {viewdraw active}

指针移到 Viewdraw 窗口空白处, 按左键清除选择的连线

F10(显示全图), F9, 放大 ALU 符号,

Change→Display→Value→On, 原理图上显示返回的在1200ns 处的仿真值

按左键选 ALU, Level→Push→Schematic, F9, 原理图的各层有返回的仿真值

Level→Pop, Level-Pop, Window→Push, 看到 ViewSim 和 ViewWave 的窗口

指针移到移到 X 轴的 200 处, F9, 指针右移到 400 处 (X 轴上出现红色区段)

按中键,仅200—400ns间的波形被显示;按F10,显示全部波形

17. 返回 XDM: Window→Close; Bye→quit

当设计包括 X-BLOX, Xilinx ABEL 元件或有用 FILE 属性的任何功能块为网表所参考时, 在利用 XSimMake 作时序仿真之前必须运行带有功能流选项的 XSimMake, 利用 XSimMake 进行功能和时序仿真可参考下一节实验十一。

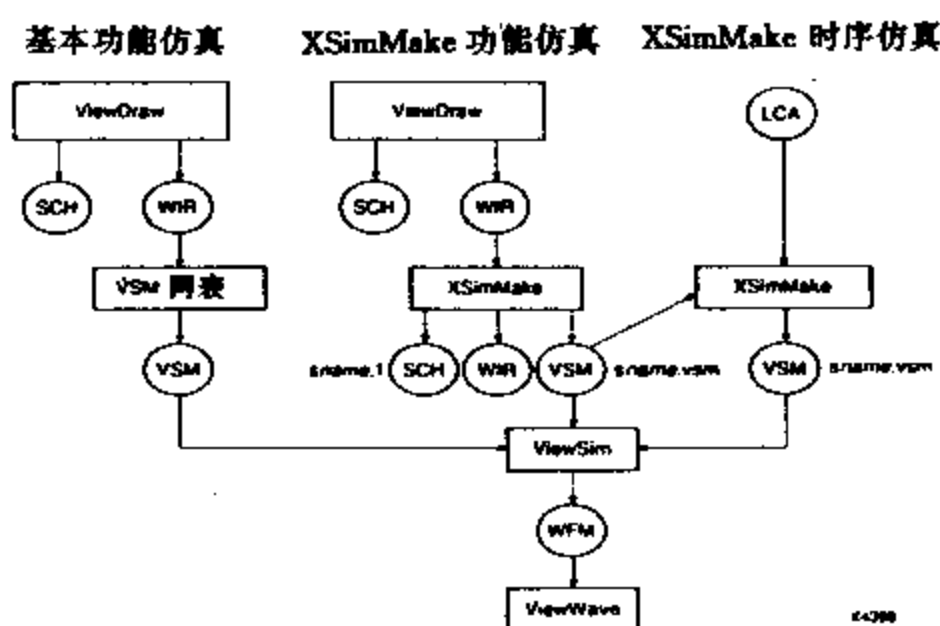


图12.14 ViewSim 的 FPGA 仿真流程

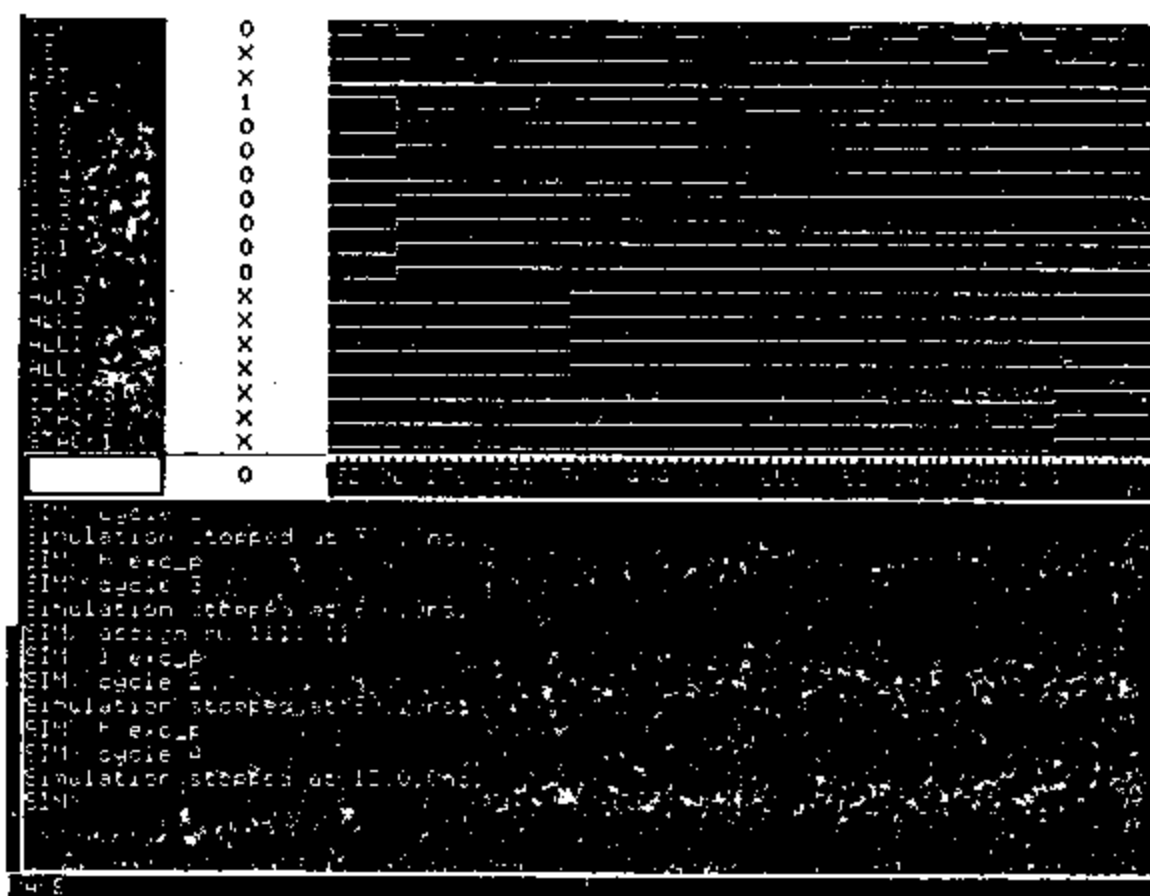


图12.15 ViewWave 窗口仿真波形

§ 12.2.6 原理图中可加属性表(表12.3)

	XC2000/L	XC3000/A/L	XC3100/A	XC4000/A/D	XC4000H
XII.INX 一般属性					
FILE	X	X	X	X	X
BLKNM	X	X	X	X	X
HBLKNM	X	X	X	X	X
M&P	X	X	X	X	X
DOUBLE		XC3000/A	X	X	X
DECODE				X	X
INIT				X	X
位置约束					
LOC	X	X	X	X	X
RLOC				X	X
USER—RLOC				X	X
U—SET				X	X
HU—SET				X	X
RLOC—ORIGIN				X	X
RLOCRAHFE				X	X
输入/输出属性					
NODELAY				X	
FAST 或 LOW		X	X	X	
MEDFAST/MEDSLOW				XC4000A	
RES/CAP					X
CMOS/TTL					X
CLB/IOB 属性					
BASE	X	X			
CONFIG	X	X			
EQUATE—F EQUATE—G	X	X			
时序规定属性					
TSidentifier		XC3000A/L	XC3100A	X	X
TNM		XC3000A/L	XC3100A	X	X
主要的网线属性					
外部网线属性—X	X	X	X	X	X
临界属性—C	X	X	X		
非临界属性—N	X	X	X	X	X
保存属性—S	X	X	X		
加权属性—W	X	X	X		

§ 12.3 XACT 5.0版本 FPGA 实验实例

§ 12.3.1 实验一:计数器设计

此实验可在等待培训班开始的时候完成,实验者不必有任何使用 Xilinx 软件设计的经验。

- 目的:
1. 引导实验者进入 Xilinx 开发系统。
 2. 利用 XACT 设计管理软件(XDM)的基本训练。
 3. 利用自动转换处理预先输入的原理图。

设计描述:

这是利用标准的 Xilinx 宏单元库的八位计数器,它针对 XC3000系列介绍,原理图如图 12.16所示,对于 XC4000系列仅出腿的位置需要改变。

实验步骤:

- A1. 如果计算机的屏幕已显示 XDM,则转到步骤 B1。
- A2. 进入到目录 C:\COUNT\XMAKE\3K
键入 C:\COUNT\XMAKE\3K <返回>
- A3. 调用 XDM
键入 XDM <返回>
- B1. 如果 XDM 的屏幕已经显示
Directory:C:\COUNT\XMAKE\3K,则转到步骤 D1。
- C1. 调用 XDM 目录指令
把光标指到"Directory"上,在屏幕上此项加亮
用鼠标左键选
- C2. 转到目录 C:\COUNT\XMAKE\3K
键入 C:\COUNT\XMAKE\3K <返回>
- D1. 设置 XDM 的选择:
用鼠标的左键选菜单的 "Profile"
再选 "Read Profile"
- E1. 利用自动设计工具转换设计—XMake
用鼠标左键选菜单的 "Translate"
再选 "XMake"
不改变任何选择,选 "Done"
指定原理图,选 "MACRO.1"
做整个转换,选 "Done"
- E2. 对此设计的处理要化费不到十分钟的时间,处理完成后屏幕显示:
"MACRO.BIT has been made, XMake succeeded" 敲任何键继续。

E3. 设置演示板

确认 XChecker 电缆和 +5V 电源已接到 XC3000 演示板, 七段数码管的十进位点应点亮, 指示电源已加到 LCA。

E4. 加载设计

选菜单的 "Verify"

选 "XChecker" → 选 "port" → 选 "com2" (因 com1 接鼠标, 并注意指令行提示)

选 "Done"

选 "MACRO. bit"

软件将告知在哪里找到加载线, 屏幕提示允许时, 敲 <返回> 键, 七段数码管十进位点不再亮, 软件应告知加载成功, 或提示错误。

E5. 控制计数器校验设计

用以下的开关 (演示板底下的八个按钮开关)

UP 增计数 PIN 11 开关1

LOAD 加载 PIN 13 开关2

时钟使能 PIN 15 开关3

置 UP 和 CLOCK EABLE 为 1, LOAD 为 0, 计数器应运行, 在 LED 上显示, 用 CLOCK EABLE 来停止计数, 用 UP 来改变计数方向, 用 LOAD 和开关的 5—8 加载计数器的四个最低位, 四个最高位总是置 1。

F1. 检查在 XMake 运行时产生的输出文件, 用以下方法打开一个文件:

选择菜单的 Utilities

选择 Browse 调用 DOS EDIT 指令

选择一个文件名

用上和下箭头或 PgUp 和 PgDn 移动文件, 用 <Alt> 及 F 键, 然后 X 键退出 EDIT 指令

最感兴趣的文件为: MACRO. PRP—提供可能改善设计的建议

MACRO. RPT—描述利用的资源 and 布局, 布线

MACRO. MRG—合成进此文件的原理图清单

MACRO. MAK—运行 XMake 转换此原理图程序清单

此设计利用器件多少资源?

§ 12.3.2 实验二: 计算器设计

通过比计数器设计更先进的计算器设计, 介绍 XACT 设计管理软件的能力。在完成计数器设计的转换之后再进行此实验, 此实验的指令不再象实验一那样详细。

目的: 1. 增强对 XDM 的使用。

2. 对预先输入的原理图进行自动转换处理。

设计描述:

此设计是具有操作码和数值输入及输出的简单计算器,包括可以被显示的堆栈,器件可以选 XC3000或 XC4000系列。

Calc 设计由一个具有堆栈的四位处理器组成,处理器执行一个内部寄存器和堆栈顶或外部开关来的输入数据之间的函数,各种操作的结果被储存进寄存器,并在七段数码管上以16进制显示,堆栈顶部的值由 LED 按二进制显示。

计数器(CALC)的设计如图12.17所示,它由九个基本功能块组成:

ALU:算术运算,如图12.18所示。组成 ALU 的 XORBLK2,ORBLK2和 ANDBLK2结构基本相同,如图12.19仅为 ANDBLK2的原理图。

CONTROL:对堆栈和 ALU 的操作码译码进入控制线,如图12.20所示。控制器中的 STAMACH 原理图如图12.21所示。

STACK:四位的存储器件,原始用触发器实现,然后把触发器转换成 RAM,如图12.22所示。

OSC_3K/OSC_4K:产生时钟信号,XC3000演示板利用外部 RC 振荡器,XC4000演示板用内部振荡器。如图12.23为 OSC_4K 的原理图。

DEBOUNCE:去掉“执行”开关的反弹,提供一次接通的输出,如图12.24所示。

SW7:按钮开关连接操作码和数据输入,如图12.25所示。

7SEGDEC:对 ALU 的输出译码,送到七段数码管上显示,如图12.26所示。7SEG_TRU/7SEG_INV:在演示板上将 ALU 的输出连接到七段数码管,XC3000和 XC4000有不同的极性和出腿位置。如图12.27为7SEG_INV 的原理图。

LED_TRU/LED_INV:由堆栈的顶接到 LED,XC3000和 XC4000有不同的极性和出腿位置。如图12.28为 LED_INV 的原理图。

演示板的原理图如图12.31所示。

实验步骤:

A1. 进到目录 C:\CALC\XMAKE\3K(或4K)

A2. 如果需要改变系列类型到正确的器件

A3. 改变“Profile”的选择,利用缺省的设置

 Profilr→ReadProfile

A4. 在顶级原理图 CALC.1上运行 XMake

 Translate→XMake→Done→CALC.1→Done

A5. 加载设计到演示板,在演示板加电后,

 Verify→XChecker→Done→CALC.BIT

设计检验:

CALC 设计是一个带有堆栈的四位处理器,必须提供三类输入:操作码,数据和执行指令,按钮开关最左的按钮是“执行”指令,拨动此钮两次,它才起作用。接着的三个开关(2—4)选择操作码,最右边四个开关输入数据。当操作码开关置于111,数据开关变成操作码的一部份,形成扩展的指令集,如表12.4所示。

表12.4 CALC 操作码表

[illegible]

§ 12.3.3 实验三: XACT"性能"实验—COUNT

此实验利用 XACT"性能"改善 COUNT 设计的结果

目的: 1. 利用 XACT"性能"的基本训练

2. 允许容易地把 XACT"性能"规定加到其它试验

实验步骤:

A1. 进入目录 C:\COUNT\XMAKE\4K (或3K), 并已经运行过 XMake

A2. 运行 XDelay 确定最大的时钟频率

选择 Verify→XDelay→Done→MACRO. LCA

(或选 Verify→XDelay→*→Done→MACRO. LCA→Timing→XDelay→Clock-ToPad→Done)

A3. 记录最大的时钟至设置路径延时报告: _____ns

这个路径对于时钟周期比限制的路径更短, 但其它限制的路径是由于 I/O 腿的锁定。软件将在计数器本身的反馈路径上有更多的控制。

以下可以分 A4—A9或 B4—B6的两种方法来进行:

A4. 调用 Workview, 打开原理图

选择 DesignEntry→Workview

打开菜单将指针移到屏幕的左边; 关闭菜单取消功能只要按鼠标的右键。

选择 Window→Open→Viewdraw→Schematic→MACRO

A5. 增加 TIMESPEC 原语来定义 XACT"性能"

选择 Add→Comp→TIMESPEC

用鼠标的左键将其放在原理图的空白处

A6. 加"性能"数到原语

选择 TIMESPEC 原语 选择 Change→Attr→Dialog→All

在 TIMESPEC 原语行 COMPONENT VALUE 处按动鼠标右键

键入"TS!=FROM;FFS;TO;FFS=xxx"

xxx 为填入的比最大 clock-to-setup 延时值更好的数值, 缺省的单位为 ns

在 ACCEPT 处按动鼠标右键

A7. 保存新的原理图

File→Write

Bye→Quit

A8. 利用 XMake 转换此设计, 确认 PPR 利用确省的选择

A9. 设计完成后, 与以前的结果进行比较

B4. 选 Misc→Exit, 退出 XDelay;

利用此 Clock-to-Pad 通道 15ns(XC4000)或 14ns(XC3000)的 XACT"性能"规定对设计重新运行 PPR, 如果设计已经满足此要求, 用一个比现在的值快 1ns 的数来做:

PlaceRoute→PPR→MACRO. XTF (MACRO. MAP)→HelpAll→dc2p=→"15"→
Done

B5. 完成之后,看 MACRO. RPT 文件,确定此 XACT"性能"数是否被满足

B6. 运行 XDelaay—TimeSpec 校验 XACT"性能"数

Verify→XDelay→TimeSpec→Done→MACRO>LCA

§ 12. 3. 4 实验四:沿边译码器实验—CALC

CALC 设计的 XC4000型式可以用沿边译码器实现 CONTROL 的功能部份。虽然这些门不是十分宽,它们是相同组输入信号的与,利用译码器可以节省 CLB 的资源。

目的:说明在原理图中如何包含译码器。

实验步骤:

A. 这个设计利用 GUIDE 选择可以更快地转换,可以编辑此原理图,利用 GUIDE 文件由 XMake 实现 CALC 来做到这一点。如果不想编辑原理图,带有译码器的设计可以在目录 CALC\DECODE\4K 中找到。由上述目录拷贝已布局 and 布线的 LCA 文件,然后跳到步骤 A6。

A1. 在 XDM 中进入目录 C:\CALC\XMAKE\4K。(在 XC3000系列中沿边译码器是无效的。)

A2. 对 CALC 的设计打开 WorkView,选 Level→Push→Schematic 下拉到 CONTROL 的原理图,然后进入到 STATMACH 的原理图。

A3. 选 Level→push→Schematic,下拉到 STATMACH,检查在 CLB 中用标准门的实现。

A4. 选 Level→Pop,上托返回到 CONTROL,利用 Change→Component,按鼠标中键后用 STATA_4K 代替 STATMACH

A5. 下拉到这个原理图观察它利用的沿边译码器,仅仅两个沿边译码器被利用,整个设计很容易装进 XC4000A 的结构。

A6. 上托返回到 CALC 的顶层,选择 File→Write 保存这个变化,利用以下选择:

Window→Open→Utility→Workview→Check→project→CALC. 1

来修改所有的 Viewlogic 网表文件(netlist)。

A7. 对此设计运行 XMake

如果已经在以前转换过此设计,可以利用 PPR 的 GUIDE 选择来加速实现,首先把原布线文件拷贝到新的文件名:

键入"DOS COPY CALC. LCA GUIDE. LCA"

然后对 PPR 设置 GUIDE 选择:

Profile→Opitions→PPR→HelpAll→Guide=→GUIDE. LCA→Done→Done

对 CALC. 1运行 XMake

B. 此实验也可利用 CALC\DECODE\4K 目录下的沿边译码器来进行:

在 XDM 中进入到 CALC\DECODE\4K 目录下,选 ReadProfile 以确保正确的选项被设置,对 CALC.1 运行 XMake

C. 在加载位流文件到器件中验证其有同样的功能,并不带选项运行 XDelay 检查时序:

Verify→XDlay→*→Done→CALC. LCA; Timing→XDlay→Done

检查最坏条件通道的延时,它是否包括译码器,通过译码器的延时是多少?与 GUIDE. LCA 的相同信息作比较,此通道变快还是变慢了?

选择练习:在 XC4000 中,用 WAND4 符号代替 DECODE4 来改变元件,比较何种资源是最有效。

§ 12.3.5 实验五:MemGen 实验

这个实验提供运行 MemGen 的实践,它描述 MemGen 的选择和步骤

目的:说明如何运行 MemGen

实验步骤:

A. 利用 MemGen 提示产生一个存储器

A1. 进入目录 C:\CALC\XMAKE\4K (RAM 仅可在 XC4000 系列实现)。

A2. 由 XDM,选 Translate→MemGen→NewFile→"new1"→<Enter>

A3. MemGen 允许由选择来定义 RAM 类型,在第一次仅仅选 -v 来产生一个 Viewlogic 的符号,然后选 Done。

A4. MemGen 允许它需要的其它信息,在有效的范围内做出所需的选择。

A5. 察看生成的 new.MEM 和 new.XNF 文件

B. 选择选项重新运行 MemGen

B1. 选 Translate→MemGen→New File,并键入"New2"。

B2. 首先选择 memory_depth。MemGen 仅支持最多256个字,键入:"16"<Enter>。

B3. MemGen 返回到选择清单,memory_depth 被加亮,并在 XDM 的信息行上显示数值。选择 word_width,并键入:"4"<Enter>。

B4. 从两个选项选择 type=和 RAM,因为选项内容的长度,XDM 的信息行出现重迭,但不是问题。

B5. 选 -b,在 Viewlogic 的符号中为一总线。

B6. 选 Done,以产生 Memory 文件。

B7. 查阅 memory 定义的文件 New2.MEM。注意到对 RAM 缺少 DATA 的关键字。

C. 在 ViewDrw 中检查生成的符号

C1. 调用 Viewdraw,并打开新的电路图

由 DesignEntry→Workview,再 Window→Open→Viewdraw→Schematic→"New"

C2. 插入第一个存储器符号,Add→Component→(按鼠标中键)"New1"

C3. 注意到符号有 FILE= 的语句, 以使 Viewdraw 知道在其下无原理图。

C4. 插入第二个存储器符号, Add→Component→(按鼠标中键)"New2", 注意总线输入和输出的不同。

§ 12.3.6 实验六: RAM 实验—CALC

CALC 的 XC4000 型式设计可以用 RAM 的形式来实现堆栈, 这样做使 CLB 从八个减少到仅两个, 因为在 RAM, 而不是在触发器中每个 CLB 可以存储更多的位。

目的: 1. 说明在原理图中如何包含 RAM。

2. 理解 MemGen 如何工作的。

实验步骤:

A1. 进入目录 C:\CALC\XMAKE\4K

A2. 对 CALC 设计打开 Workview, 下拉到 STACK 的原理图, 检查它按触发器的实现

A3. 上托返回到 CALC 的顶层, 利用 Change→Component 用 STACK_4K 代替 STACK, 下拉到这个原理图观察它的 RAM 实现。

作为一个替换, 可以在目录 C:\CALC\XMAKE\4K 中看到一个被修改的原理图。

A4. 上托返回到 CALC 的顶层, 选择 File→Write 保存这个修改, 选择

Window→Open→Utility→Workview→Check→Project→CALC. 1

来修改所有的网表文件。

A5. 对此设计运行 XMake, 如果以前已经转换过此设计, 利用 GUIDE 选择来加速实现, 否则, 在 PPR 利用 Effort 的级别为 1。

B1. 此实验也可利用 CALC\RAM\4K 目录下的 RAM 来进行:

在 XDM 中进入到 CALC\RAM\4K 目录下, 选 ReadProfile 以确保正确的选项被设置, 对 CALC. 1 运行 XMake

C1. 在加载位流文件到器件中验证其有同样的功能, 并不带选项运行 XDelay 检查时序:

Verify→XDelay→*→Done→CALC. LCA; Timing→XDelay→→ToIOB→"LED*"→Done

检查最坏条件通道的延时, 它是否包括 RM, 通过 RAM 的延时是多少?

C2. 与 GUIDE. LCA 的相同信息作比较:

用 Control—Break 结束 XDelay 的报告, 在屏幕选 Design; 转到 GUIDE. LCA

选 Timing→XDelay→→ToIOB→"LED*"→Done

最坏条件通道的延时是多少, 它是从哪一点开始的?

C3. 因为通道从寄存器堆栈开始, 代替了地址产生器逻辑, 也需要包含地址产生器逻辑的延时:

Timing→XDelay→—ClearOptions→FremFF→"ADDR"→Done→ToFF→"STACK*"→Done→Done

当利用 RAM 时通道如何改变?有和没有 RAM 的设计之间 CLB 数目有多少变化?

选择练习:

运行 MemGen 产生新的 RAM 符号,选择 4x4 作为 RAM 的规模,以便不必连接无用的地址线,插入新的符号进 CALC 原理图。

§ 12.3.7 实验七:X-BLOX 实验—COUNT

这个实验用 X-BLOX 的模块转换八位计数器,有 XC3000 和 XC4000 两种演示板型式。

目的:1. 说明运行 X-BLOX 的基本点。

2. 一个设计的 X-BLOX 实现与其它的实现进行比较。

设计描述:

此设计是可在 LED 上显示的简单八位计数器,可选择 XC3000 或 XC4000 的器件实现。

实验步骤:

A1. 进入目录 C:\COUNT\XMAKE\3K(或 4K)

选 Directory

向上选目录,一直到根目录 C:

选 COUNT

选 XBLOX

选最后的目录:用 XC3000 系列选 3K;用 XC4000 系列选 4K。

A2. 如果必要变更器件系列到正确的选用器件:

选 Family

选 XC3100A(或 XC4000)

选 XC3120APC68(或 XC4003APC84),包括正确的封装

选 -3(或 -4)

A3. 检查 Profile 的选择:

选 Profile

选 Setting(或键入 set),特别注意 XMake 和 PPR 的选项,应利用缺省的选择。

A4. 运行 XMake

选 Translate

选 XMake,按鼠标左键

选 XBLOX.1,按鼠标左键

A5. 加载此设计以便校验它

选 Verify→XChecker→Done→XBOLX.BIT

当允许时,键入 <Enter>

选择练习:

B1. 检查 XBOLX.BLX 文件看一下 X-BLOX 为此设计做了什么, X-BLOX 利用了什么专门的资源, 而标准的宏单元不能利用?

C1. 比较 X-BLOX 设计比其它实现的有效性, 检查 .RPT 文件确定资源的利用率, 运行 XDelay 确定性能。

§ 12.3.8 实验八: X-BLOX 实验—CALC

这个实验用 X-BLOX 的符号代替 CALC 设计的 ALU, X-BLOX 的符号是较易利用的, 并自动对准目标结构的资源。

目的: 1. 说明运行 X-BLOX 的基本点。

2. 一个设计的 X-BLOX 实现与其它的实现进行比较。

实验步骤:

A. 在 CALC\XMAKE\3K(或4K)目录下进行

A1. 进入目录 C:\CALC\XMAKE\3K(或4K)

A2. 如果必要变更系列到正确的目标:

A3. 打开 CALC 的原理图

A4. 利用 X-BLOX 的模块代替 ALU 原理图

选宏单元 ALU

选 Change→Component→ALU_BLOX

选 File→Write

此 X-BLOX 型式可以在 C:\CALC\XMAKE\3K(或4K)中找到

A5. 下拉到 ALU_BLOX 原理图, 观察此 X-BLOX 的模块, 如图12.29所示。

选择 ALU_BLOX, 再选 select→Push→Schematic 下拉到 ALU_BLOX 原理图
做完后, 键入 Q 退出 Workview。

A6. 运行 XMake, 转换此设计

Translate→XMake→Done→CALC.1→Done

A7. 加载此设计以便校验它

选 Verify→XChecker→Done→CALC.BIT

当允许时, 键入 <Enter>

B. 此实验也可利用 CALC\XBLOX\4K(3K)目录下的原理图来进行。

选择练习:

B1. 比较 X-BLOX 设计比其它实现的有效性, 检查 .RPT 文件确定资源的利用率, 运行 XDelay 确定性能。

```

module count
title 'LCA, with symbolic state machine entry'
'clocks
    clock                pin;
'control inputs
    en                    pin;
'outputs
    Q0, Q1, Q2, Q3        pin;
'state diagram declaration and assignment
    sbit                  STATE_REGISTER istype 'reg_D';
    s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15
STATE;
xilinx property 'Initialstate s0';
Equations
    sbit.clk = clock;
    Q0=(s1 # s3 # s5 # s7 # s9 # s11 # s13 # s15);
    Q1=(s2 # s3 # s6 # s7 # s10 # s11 # s14 # s15);
    Q2=(s4 # s5 # s6 # s7 # s12 # s13 # s14 # s15);
    Q3=(s8 # s9 # s10 # s11 # s12 # s13 # s14 # s15);

State_Diagram sbit
    State s0: if (en) then s1
              else s0;
    State s1: if (en) then s2
              else s1;
    State s2: if (en) then s3
              else s2;
    State s3: if (en) then s4
              else s3;
    State s4: if (en) then s5
              else s4;
    State s5: if (en) then s6
              else s5;
    State s6: if (en) then s7
              else s6;
    State s7: if (en) then s8
              else s7;
    State s8: if (en) then s9
              else s8;
    State s9: if (en) then s10
              else s9;
    State s10: if (en) then s11
              else s10;
    State s11: if (en) then s12
              else s11;
    State s12: if (en) then s13
              else s12;
    State s13: if (en) then s14
              else s13;
    State s14: if (en) then s15
              else s14;
    State s15: if (en) then s0
              else s15;
end

```

§ 12.3.9 实验九: Xilinx—ABEL 实验—COUNT

这个实验利用 Xilinx—ABEL 实现一个四位计数器,它利用定义状态的符号法,留下判
决于软件,只利用二进制或 one-hot 编码,但是,输出被定义来产生二进制文件。

- 目的: 1. 说明运行 Xilinx—ABEL 的基本点。
2. 说明替换用原理图定义计数器。
3. 用 Xilinx—ABLE 的计数器实现与原理图输入工具的实现进行比较。

设计描述:

这个称为 XABLE 的设计是利用 Xilinx—ABEL 的符号状态机定义的一个四位计数器,
它有用最左边按钮开关控制它的使能输入。

实验步骤:

A1. 进入目录 C:\COUNT\XABEL\3K(或4K)

选 Directory

向上选目录,一直到根目录 C:

选 XABEL

选最后的目录:用 XC3000系列选3K;用 XC4000系列选4K。

A2. 如果必要变更系列到正确的目标:

选 Family

选 XC3100A(或 XC4000)

选 XC3120APC68(或 XC4003APC84),包括正确的封装

选—3(或—4)

A3. 首先检查 ABEL 的文件,如下调用 Xilinx—ABEL:

选 DesignEntry

选 XABEL

选 COUNT.ABL

A4. 此设计可以被转换或者在 Xilinx—ABEL 中,或者在 XDM 中用 ABL2XNF 本身或
作为 XMake 的一部份。此处作为 XMake 的一部份来转换,先退出 Xilinx—ABEL:
键入<ALT>及 F,然后 X

A5. 在顶级原理图 XABEL.1上运行 XMake,此图把 XABEL 文件连接到演示板的输入
和输出。

选 Translate

选 XMake,按鼠标左键

选 XABEL.1,按鼠标左键

A6. 当完成后,加载此设计到演示板。

演示板最左边按钮开关控制计数器的使能输入。

以下是 COUNT.ABL 文件的内容:

```
module count
title 'LCA, with symbolic state machine entry'
*clocks
    clock                                pin;
*control inputs
    en                                    pin;
*outputs
    Q0, Q1, Q2, Q3                       pin;
*state diagram declaration and assignment
    sbit                                STATE_REGISTER istype 'reg_D';
    s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15
STATE;
xilinx property 'Initialstate s0';
Equations
    sbit.clk = clock;
    Q0=(s1 # s3 # s5 # s7 # s9 # s11 # s13 # s15);
    Q1=(s2 # s3 # s6 # s7 # s10 # s11 # s14 # s15);
    Q2=(s4 # s5 # s6 # s7 # s12 # s13 # s14 # s15);
    Q3=(s8 # s9 # s10 # s11 # s12 # s13 # s14 # s15);
```

State_Diagram sbit

```
State s0: if (en) then s1
           else s0;
State s1: if (en) then s2
           else s1;
State s2: if (en) then s3
           else s2;
State s3: if (en) then s4
           else s3;
State s4: if (en) then s5
           else s4;
State s5: if (en) then s6
           else s5;
State s6: if (en) then s7
           else s6;
State s7: if (en) then s8
           else s7;
State s8: if (en) then s9
           else s8;
State s9: if (en) then s10
           else s9;
State s10: if (en) then s11
           else s10;
State s11: if (en) then s12
           else s11;
State s12: if (en) then s13
           else s12;
State s13: if (en) then s14
           else s13;
State s14: if (en) then s15
           else s14;
State s15: if (en) then s0
           else s15;
```

end
- 242 -

§ 12.3.10 实验十: Xilinx—ABEL 实验—CALC

这个实验利用 Xilinx—ABEL 的文件代替在 CALC 设计中的状态机,功能上是相同的。在已经完成计数器设计的转换后做此实验。此实验也将建立在以后实验中演示 XSimMake 的好处的设计;XSimMake 允许 Xilinx—ABEL 的符号返注到原理图中。

目的:1. 说明运行 Xilinx—ABEL 的基本点。

2. 说明如何集成 Xilinx—ABEL 的设计到原理图中。

设计描述:

编辑已经转换的 CALC 设计的一个,将一个块用 Xilinx—ABEL 的块代替。

实验步骤:

A1. 进入目录 C:\CALC\XABEL\3K(或4K)

A2. 如果必要变更系列到正确的目标:

A3. 首先检查 ABEL 的文件,如下调用 Xilinx—ABEL:

选 DesignEntry

选 XABEL

选 STAT_ABL.ABL

A4. 仿真此设计,按动<ESC>键在 Xilinx—ABEL 中打开菜单,然后利用左和右键进入到正确的菜单,或者键入<ALT>+菜单的字母

键入<ALT>+C 或<ESC>后移到 Compile 菜单

键入 S,仿真此设计

当处理暂停时,按动任何键继续

确认所有的测试矢量都通过

A5. 此设计可以被转换或者在 Xilinx—ABEL 中,或者在 XDM 中用 ABL2XNF 本身或作为 XMake 的一部份。在运行 XMake 之前,需做转换产生在原理图中可被代替的符号。此处从 XDM 做转换,先退出 Xilinx—ABEL:

键入<ALT>及 F,然后 X

A6. 运行:

选 Translate→ABL2XNF

选 STAT_ABL.ABL 文件,可以或者选—area 对密度优化,或者选—speed 对性能优化。注意由 Xilinx—ABEL 报告的资源要求:

Number of CLBs:—————

Max LC2S:—————

Max LC2P:—————

Max LP2S:—————

Max LP2P:—————

A7. 在由 ABL2XNF 产生的 STAT_ABL.XSF 上运行 SemGen

选 DesignEntry→SymGen→STAT_ABL.XSF

对 Viewlogic 选—v, 然后选 Done

由 SymGen 产生这个符号, 并放进 SYM 子目录, 此符号也已放进此目录的原理图中。

A8. 运行 XMake

Translte→XMake→Done→CALC. 1→Done

A9. 在完成后, 加载此设计到演示板。

以下是 STAT_ABL.ABL 文件的内容:

```
module stat_abl

title 'State machine for Calc design'
*This state machine has 3 states which control the functions
*of the ALU and the stack. The states are as follows:
*      SPUSH  -- increment stack pointer
*      SWE    -- write value into stack
*      SOTHER -- do neither (initial state)

*This is a one-hot state machine, which means that only
* one of the states is active at any given time. This method
*is particularly suited for use with Xilinx ABEL and Xilinx
*FPGAs, which are rich in flip-flop resources.
*This file also generates control signals from equations.
*For an equivalent schematic, see statmach.1.

declarations

*inputs
      OP5, OP4, OP3, OP2, OP1, OP0, EXC      pin;

*clock
      CLK                                     pin;

*outputs
      CTL3, CTL2, CTL1, CTL0                 pin;
      UP_DN, WE, RST, ADD_SUB, CE_ALU, CE_ADDR pin;

*state diagram declarations and assignments
      XABELSM      state_register istype 'reg_d';
      SPUSH, SWE, SOTHER      state;

*vector definitions
      OP  = [OP5,OP4,OP3,OP2,OP1,OP0];
      HOP = [OP5,OP4,OP3];
      CTL = [CTL3,CTL2,CTL1,CTL0];

*declare internal nodes
      SEL_OP, OP_CTL2, OP_CTL1, OP_CTL0      node;

*node declarations for simulation only, can't use state names
*in simulation vectors
      PUSH, OTHER                             node;

*define clock & don't-care values for test vectors
      C, X = .C., .X.;

Xilinx property 'initialstate SOTHER';
```

equations

```

XABELSM.CLK = CLK;
RST          = (HOP == ^b101) & EXC;
ADD_SUB      = !OP_CTL2;
SEL_OP       = (HOP == ^b111);
CE_ALU       = !(SEL_OP & OP2 & OP0) & SOTHER & EXC;
CE_ADDR      = !(OP2 & OP1 & OP0) & SEL_OP & EXC;
OP_CTL2      = (OP5 & !SEL_OP) # (OP2 & SEL_OP);
OP_CTL1      = (OP4 & !SEL_OP) # (OP1 & SEL_OP);
OP_CTL0      = (OP3 & !SEL_OP) # (OP0 & SEL_OP);
CTL3         = SEL_OP;
CTL2         = OP_CTL2 & OP_CTL1;
CTL1         = OP_CTL1 & !OP_CTL2;
CTL0         = !OP_CTL2 & OP_CTL0;
UP_DN        = OP2 & !OP1 & OP0 & SEL_OP & EXC;
PUSH         = SPUSH;
WE           = SWE;
OTHER        = SOTHER;

```

*always optimize out don't-cares`
@DCSET

state_diagram XABELSM

```

state SPUSH: goto          SWE;
state SWE:   goto         SOTHER;
state SOTHER: if (UP_DN) then SPUSH
               else       SOTHER;

```

test_vectors

*begin in initial state, each line is one clock cycle

((CLK,EXC,OP)->[PUSH,WE,OTHER,ADD_SUB,RST,CE_ALU,CE_ADDR,CTL])

*quick check to test the state machine

```

[ C, 0, X ]->[ 0, 0, 1, X, X, X, X, X ];
[ C, 1, ^h3F ]->[ 0, 0, 1, X, X, X, X, X ];
[ C, 0, X ]->[ 0, 0, 1, X, X, X, X, X ];
[ C, 1, ^h3D ]->[ 1, 0, 0, X, X, X, X, X ];
[ C, 0, X ]->[ 0, 1, 0, X, X, X, X, X ];
[ C, 0, X ]->[ 0, 0, 1, X, X, X, X, X ];
[ C, 1, ^h38 ]->[ 0, 0, 1, X, X, X, X, X ];

```

*test the control logic, EXC low

```

[ C, 0, ^h0 ]->[ 0, 0, 1, 1, 0, 0, 0, ^h0 ];
[ C, 0, ^h8 ]->[ 0, 0, 1, 1, 0, 0, 0, ^h1 ];
[ C, 0, ^h10 ]->[ 0, 0, 1, 1, 0, 0, 0, ^h2 ];
[ C, 0, ^h18 ]->[ 0, 0, 1, 1, 0, 0, 0, ^h3 ];
[ C, 0, ^h20 ]->[ 0, 0, 1, 0, 0, 0, 0, ^h0 ];
[ C, 0, ^h28 ]->[ 0, 0, 1, 0, 0, 0, 0, ^h0 ];
[ C, 0, ^h30 ]->[ 0, 0, 1, 0, 0, 0, 0, ^h4 ];

```

*extended instruction set

```
[ C, 0, ^h38]->[ 0, 0, 1, 1, 0, 0, 0, ^h8 ];
[ C, 0, ^h39]->[ 0, 0, 1, 1, 0, 0, 0, ^h9 ];
[ C, 0, ^h3A]->[ 0, 0, 1, 1, 0, 0, 0, ^hA ];
[ C, 0, ^h3B]->[ 0, 0, 1, 1, 0, 0, 0, ^hB ];
[ C, 0, ^h3C]->[ 0, 0, 1, 0, 0, 0, 0, ^h8 ];
[ C, 0, ^h3D]->[ 0, 0, 1, 0, 0, 0, 0, ^h8 ];
[ C, 0, ^h3E]->[ 0, 0, 1, 0, 0, 0, 0, ^hC ];
[ C, 0, ^h3F]->[ 0, 0, 1, 0, 0, 0, 0, ^hC ];
```

*test the control logic, EXC high

```
[ C, 1, ^h0 ]->[ 0, 0, 1, 1, 0, 1, 0, ^h0 ];
[ C, 1, ^h8 ]->[ 0, 0, 1, 1, 0, 1, 0, ^h1 ];
[ C, 1, ^h10]->[ 0, 0, 1, 1, 0, 1, 0, ^h2 ];
[ C, 1, ^h18]->[ 0, 0, 1, 1, 0, 1, 0, ^h3 ];
[ C, 1, ^h20]->[ 0, 0, 1, 0, 0, 1, 0, ^h0 ];
[ C, 1, ^h28]->[ 0, 0, 1, 0, 1, 1, 0, ^h0 ];
[ C, 1, ^h30]->[ 0, 0, 1, 0, 0, 1, 0, ^h4 ];
```

*extended instruction set

```
[ C, 1, ^h38]->[ 0, 0, 1, 1, 0, 1, 1, ^h8 ];
[ C, 1, ^h39]->[ 0, 0, 1, 1, 0, 1, 1, ^h9 ];
[ C, 1, ^h3A]->[ 0, 0, 1, 1, 0, 1, 1, ^hA ];
[ C, 1, ^h3B]->[ 0, 0, 1, 1, 0, 1, 1, ^hB ];
[ C, 1, ^h3C]->[ 0, 0, 1, 0, 0, 1, 1, ^h8 ];
[ C, 1, ^h3D]->[ 1, 0, 0, 0, 0, 0, 1, ^h8 ];
```

*insert two clocks to return to initial state

```
[ C, 0, ^h3D]->[ 0, 1, 0, 0, 0, 0, 0, ^h8 ];
[ C, 0, ^h3D]->[ 0, 0, 1, 0, 0, 0, 0, ^h8 ];
[ C, 1, ^h3E]->[ 0, 0, 1, 0, 0, 1, 1, ^hC ];
[ C, 1, ^h3F]->[ 0, 0, 1, 0, 0, 0, 0, ^hC ];
```

end stat_abl

§ 12.3.11 实验十一:仿真实验—CALC

这个实验利用 XSimMake 来执行 CALC 设计的功能和定时仿真。任何已有原理图型式中将能完全仿真此设计,即使用 MemGen 宏单元,X-BLOX 模块和 Xilinx-ABEL 宏单元。

目的:1. 说明运行 XSimMake 的基本点。

2. 观察 CALC 设计的性能。

实验步骤:

A1. 进入目录 C:\CALC\XMAKE\3K(或4K),或 CALC\XBLOX\3K(或4K)

为了看到返回注释到 XBLOX 的总线,需利用已编辑过包括 XBLOX 模块的设计。

A2. 为在 Viewsim 作功能仿真,先运行 XSimMake

Verify→XSimMake→—F→Viewlogic_Fpga_Func→Done→CALC. 1

A3. 打开 Workview,检查新的 SCALC 的原理图

注意原理图 SCALC 看起来和原始的 CALC 几乎一样,用到 ALU-BLOX 时将有 XBLOX 的总线代替 Viewlogic 的总线。

A4. 返回到 XDM,为 Viewsim 的时序仿真运行 XSimMake

Verify→XSimMake→—F→Viewlogic_Fpga_Timing→Done→CALC. LCA

A5. 打开 Workview 和 SCALC 原理图,利用屏幕的上面3/5

A6. 如下打开 Viewsim:

Window→Open→Viewsim S/D→<Enter>

在屏幕底部,按动鼠标两次,打开窗口

A7. 取决于所在的设计和目录,键入或者 CALC_3K,或者 CALC_4K,以运行 Viewsim 的指令文件。

A8. 在 Viewwave 中观察结果

选 Window→Open→Viewwave→Viewsim→<Enter>

在 Schematic 窗口,按动鼠标两次,用 Viewwave 窗口复盖它

A9. 在原理图上观察结果,选 Window→Push→Schmatic,下拉在 Viewdraw 窗口后的 Viewwave 窗口,下拉到原理图中,观察整个信号值。

§ 12.3.12 实验十二:RPM 实验—COUNT

这个实验利用相关布局宏单元(RPM)转换八位计数器,因为 RPM 仅支持 XC4000 系列,设计必须对 XC4000 系列进行转换。

目的:1. 说明在原理图中如何包含 RPM。

2. 一个设计的 RPM 实现与其它的实现进行比较。

设计描述:

这个设计是可以在 LED 上显示的一个简单八位计数器。

实验步骤:

A1. 进入目录 C:\COUNT\RPM\4K

A2. 如果必要变更系列到正确的目标,选 Profile→ReadProfile

A3. 运行 XMake

选 Translate→XMake→(按鼠标右键)→RPM.1→按鼠标右键

A4. 加载此设计以便校验它

选 Verify→XChecker→Done→RPM.BIT

当允许时,键入<Enter>

B1. 比较 X-BLOX 设计比其它实现的有效性,检查.RPT 文件确定资源的利用率,运行 XDelay 确定性能。

§ 12.3.13 实验十三:RPM 实验—CALC

在不利用 X-BLOX 的 CALC 设计的 XC4000型包含一个 RPM 在 ALU 中,命名为 ADSU4,这个实验将观察原理图中的 RPM,并检查最终的结果。

目的:1. 说明在原理图中如何包含 RPM。

2. 理解 RPM 如何工作。

实验步骤:

A1. 进入目录 C:\CALC\RPM\4K

A2. 如果以前已经转换过此设计,利用 GUIDE 和 Timing=ignored 选择运行 XMake 以加速实现,RPM 仍将按它描述的那样被实现。

A3. 在原理图中检查 RPM,如果已把 ALU 转到 ALU_BLOX,用 Change→Component 把它改回到 ALU,下拉进 ALU,然后进入到 ADSU4宏单元,记录被描述的相关布局,它是一行,一列或方形?

A4. 检查实现的结果,在 XDelay 中 QueryNet 可能是一个有用的工具,不仅对检查布局,而且与网线延时有关的布局质量。

§ 12.3.14 实验十四:读回实验—COUNT

这个实验由 COUNT 设计执行读回,图12.30表示有 READBACK 基本单元的原理图。

目的:1. 说明在 XC4000中如何使用读回。

设计描述:

对设计的此型式加读回符号到原理图,读回由 PIN67触发,数据被接到 PIN81。时钟由芯片外引入,以使 Xchecker 电缆可以接通和控制时钟,它由 PIN83出,由 PIN81进入 LCA。

实验步骤:

- A1. 进入目录 C:\COUNT\RDBK\4K
- A2. 设计已转换,但需产生位流文件和其它有关文件,运行 MakeBits 选相应的选项以能读回,选 Verify→MakeBits
 - F ReadCapture→Enabled
 - L→Done→READ. LCA
- A3. 选 XChecker,进入交互模式
 - Verif→XChecker→Done→Done
- A4. Xchecker 电缆的相应插针插入演示板上器件的相应出腿:
 - RT: Pin 67 RD: Pin 69
 - CLKI;Pin 83 CLKO;Pin 81
- A5. 按以下指令加载此设计后,校验此设计正在正确地运行
 - "Load READ",当允许时,键入<Enter>
- A6. 校验配置:
 - "Verify"
- A7. 由顶级原理图拾取信号来观察
 - "PICK Q00 Q01 Q02 Q03 Q04 Q05 Q06 Q07"
- A8. 执行读回
 - "READ"
- A9. 显示结果
 - "SHOW"
- A10. 停止时钟
 - "CLOCK —STOP"
- A11. 利用内部的 XChecker 时钟
 - "CLOCK —INT"
- A12. 八倍时钟周期
 - "CLOCK 8"
- A13. 恢复时钟
 - "CLOCK —RESUME"
- A14. 每个时钟触发读回
 - "TRIG —CLOCK 0 1"
- A15. 读回16次
 - "READ 16"
- A16. 显示结果
 - "SHOW"

§ 12.4 FPGA 设计变换到 EPLD

Xilinx 开发系统 XACT5.0 采用的统一库包括所有 Xilinx 元件的符号, 这些符号的某一些是为一个器件系列规定的, 某一些是为两个或更多系列共用的, 而有一些是为所有的系列所共有的。当相同名字的一个元件存在于多个系列的库中, 它有相同的功能, 图形符号和相似的命名腿, 但是, 无论这个元件是行为模块的基本单元或原理图模块的宏单元, 元件的实现可能在系列之间改变。

为了使设计与器件无关, 只要利用对所有器件系列共用的符号, XACT 软件自动地映射设计中的符号到所选用的器件, 产生与器件无关的设计允许用不同的 Xilinx 器件方便地测试设计。

对于 EPLD 的设计, 通常公共的库元件就很见效, 只是在以下的特殊条件下应利用 XC7000 的专用元件:

1. 在级联算术元件时, 应利用 EPLD 专用算术元件, 因为元件之间走的进位线被映射到进位链。

2. 在级联加减计数器时, 应利用 EPLD 专用计数器, 因为分开的加减计数端可为更高的速度和密度映射到 UIM 中。

3. 当需要带时钟使能的输入焊盘寄存器, 必须利用 EPLD 专用的元件, 如 IFDX1 等。

如果希望设计是完全与器件无关的, 就不要利用原理图的属性, 对 FPGA 和 EPLD 共用的属性只有以下两个:

PART=device_name(或 OrCAD 中的 PARTTYPE)

LOC=device_pin

即使以上属性也不完全是与器件无关的, 当改变器件时, 必须改变属性值。

§ 12.4.1 变换 FPGA 设计到 EPLD 的一般顺序

为说明方便, 假设由 XC3000 系列开始作转换:

1. 转换到为 XC7000 系列的工作目录, 并配置相应的库:

对 Viewlogic 软件, 编辑 viewdraw.ini 文件, 除加进 XC7000 系列的库之外, 保留 XC3000 的库, 因为下一步用的 Altran 指令要求它。

对 OrCAD 用户, 进入工作目录, 在 DOS 提示下键入: "xdraft 7" <Enter>

2. 利用任何必要的变换程序使原理图中的符号参照重新配置的库, 利用 OrCAD 软件时跳过这一步; 利用 Viewlogic 软件则用 Altran 指令, 其中 N=2, 3 或 4:

altran -1 primary xcN000=xc7000

3. 进入设计工具, 打开设计;

4. 在原理图中找出所有不是 EPLD 兼容的库元件, 并利用 XACT 库手册找出等效的 EPLD 兼容的库元件;

如果对 Viewlogic 的原理图已运行过 Altran, 不兼容的元件以白框出现在原理图中;

如果对 OrCAD 的原理图已运行 XDRAFT7, 不兼容元件在 OrCAD 中产生错误信息, 不出现在原理图中;

5. 如果必要, 利用专用的 EPLD 元件来得到更有效的设计, 在大多数情况下, 与器件无

关的元件有效地被映射到 EPLD 结构上,但有例外;

6. 移去所有的属性,但 PART=device_name 和 LOC=device_pin 除外,可以改变器件名为选用的新器件的名字,当不要重新分配引脚时,移去 LOC=device_pin 的属性;

7. 可能要查阅 EPLD 属性的清单,特别是优化的属性,利用它们来精细地调整设计,软件用缺省值优化它可以优化的每一件事,在某些情况下可能要防止优化,例如,要确保内部节点在时序仿真期间是可见的;

8. 进入 XDM,运行 XEMake 指令来适配设计,如果要执行功能和时序仿真,则运行 XSimMake;

9. 当执行功能和时序仿真时,必须设置预加载信号 PRLD 先高再低,代替置全局复位 GR 先低后高;

10. 如要执行时序仿真,可能必须改变驱动和监视的节点,EPLD 适配器优化逻辑,使得设计中的许多内部节点消失,但所有外部信号总是可见的。

如果 FPGA 设计有 RAM,ROM 或其它 EPLD 不能等效的元件时,除非重新设计这些元件,否则不可以转换设计。

§ 12.4.2 变换行为描述模块

如果设计包含 Xilinx-ABEL 或 XBOLX 模块,在运行 XEMake 之前,必须执行以下步骤:

1. 改变状态机模块的编码,不必重写逻辑,只要重作状态分配,对有丰富寄存器的 FPGA,一个有效编码(一个符号状态机)是最有效的,对有丰富乘积项的 EPLD 最大编码(一个编码状态机)是最有效的。对于十分简单的状态机可能是不必要的。

2. 利用 ABL2PLD 指令把所有 Xilinx-ABEL 模块下的文件变换到 PLUSASM,利用存在的 Xilinx-ABEL 符号,但移去 DEF=XABEL 和 FILE=file_name 的属性,代之以 PLD=file_name,为了保证软件不处理旧文件,删除在 xnf 目录下的 file_name.xnf 文件。

3. 变换 XBLOX 模块,不能直接地变换,可以利用行为输入工具(PLUSASM,Xilinx-ABEL,ABEL)重写逻辑,或产生与 EPLD 兼容的低层原理图。

§ 12.4.3 变换 FPGA 设计到 EPLD 的例子

利用 FPGA 实验中的计算器设计-CALC 作为例子来说明变换到 EPLD 的步骤。

1. 重新配置库和原理图符号

1) 进入 CALC 的目录,calc/soln_3K;

2) 编辑 viewdraw.ini 文件,除加进 XC7000 系列的库之外,保留 XC3000 的库,因为下一步用的 Altran 指令要求它。

3) 在工作目录下,在 DOS 提示下执行以下指令:

```
altran -l primary xc3000=xc7000
```

2. 编辑原理图

为了编辑原理图以使所有原理图的符号与 XC7000 的器件相兼容,仿照以下的步骤。

1) 启动 XDM,由设计输入菜单选 Workview 选项;

2) 打开 CALC 原理图,选择 OSC_3K 符号,执行 Level→Push→Schematic;

- 3) 编辑原理图以使它如图12.32所示;
- 4) 确认标记新的 IOPAD 网线"XCLK";
- 5) 选 File→Write, 再选 Level→Pop;

可以利用 File→Write to 保存这个原理图为 OSC_7K, 如原理图存于新的名字, 要确保相应的符号文件存于相同的名字, 此外要确保新的符号加到并存进顶层原理图中。

6) 搜索顶层原理图和每个子层原理图的包括 LOC=pin 赋值的属性, 取消除 PART=device_name 之外的所有属性, 属性以黄色显示以区别白色的标记, 利用 File→Write 保存改变的每个原理图;

- 7) 利用 Chang→Text 改变器件名: PART=7372-10PC68;
- 8) 例用 File→Write 保存顶层7原理图。

3. 执行功能仿真

- 1) 选 Export→Wirelist→ViewSim;
 - 2) 按 § 12.2 的 ViewSim 使用说明运行仿真程序, 但复位信号例外;
- 原执行: l gr 改为: h prld

.....	
h gr	l prld

4. 运行适配器指令

- 1) 退出 Workview;
- 2) 在 XDM 的 Translate 菜单中选 XEMake;
- 3) 选 Done, (即选锁有缺省选项), 选 calc. 1;
- 4) 选 Make design database.

5. 执行时序仿真

- 1) 选 XSimMake, 再选 Viewlogic→EPLD→Timing, 由清单选 calc. vmh 文件;
- 2) 按 § 12.2 的 ViewSim 使用说明运行仿真, 但要改变矢量定义, 仿真输出定义, 时钟定义和全局复位及初始输入值的有关行如以下文件所示。

必须作如下的改变是因为 EPLD 适配器优化逻辑, 去掉了许多内部节点, 这使得设计更有效, 但更难仿真。

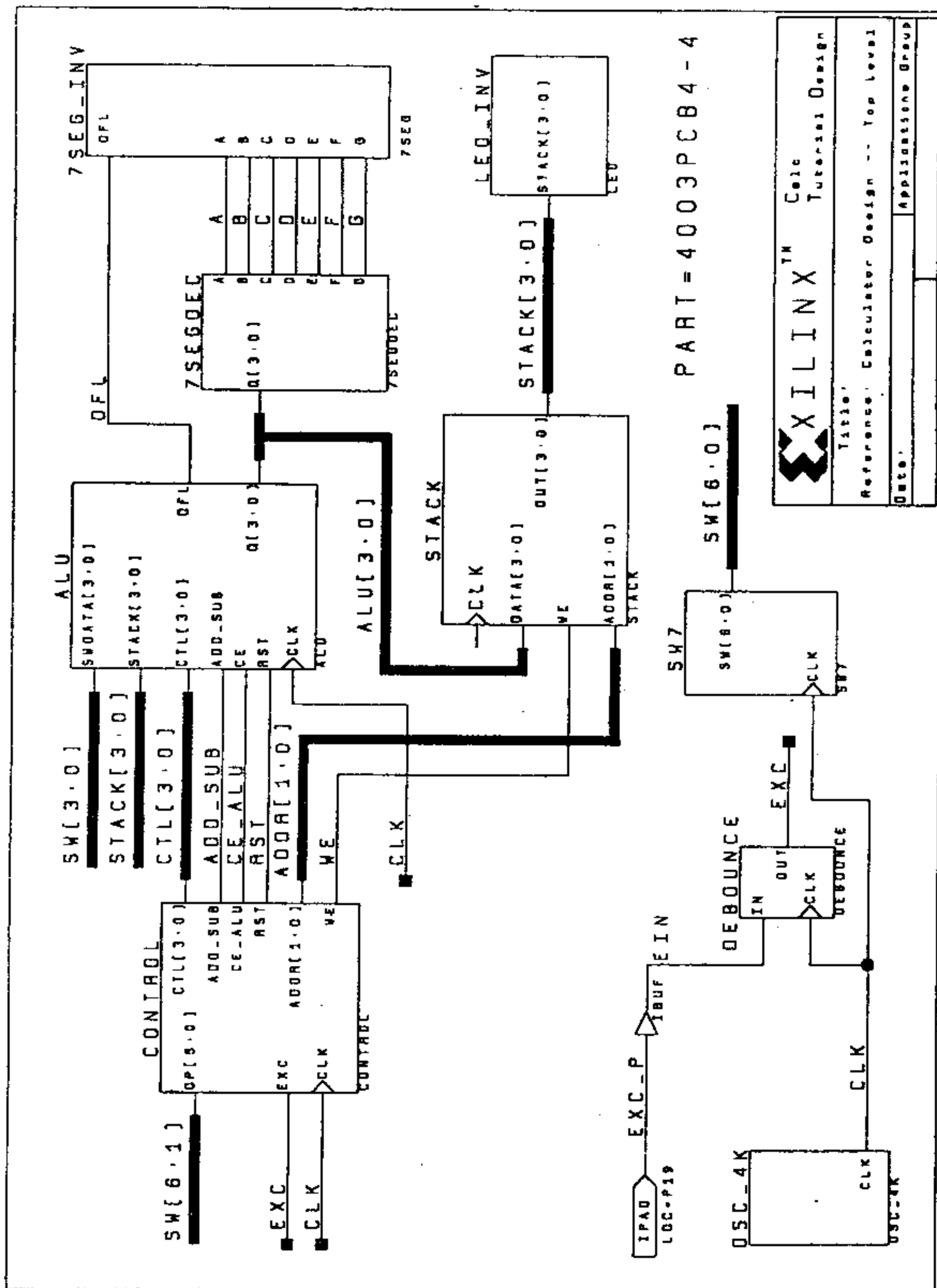


图 12.17

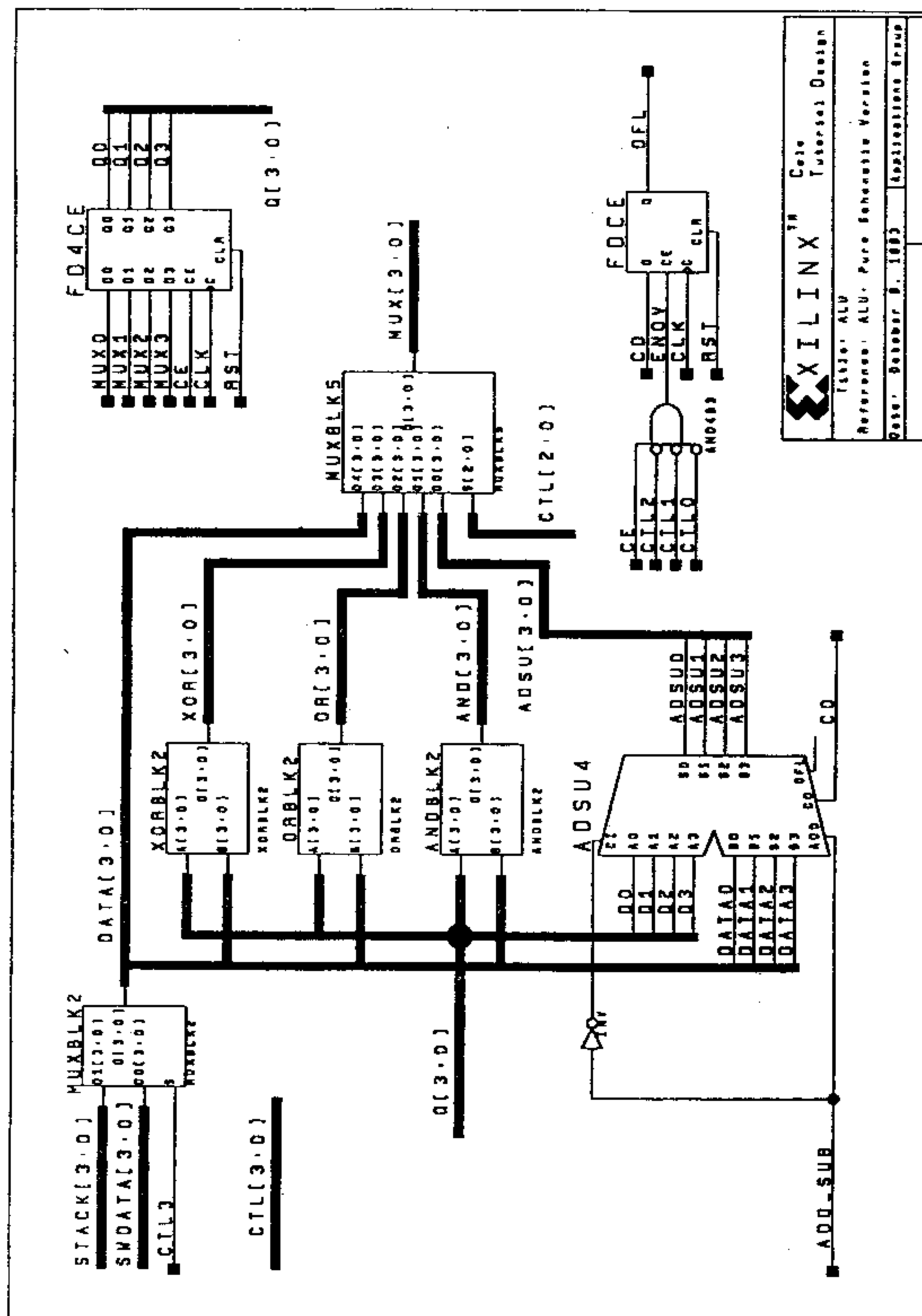
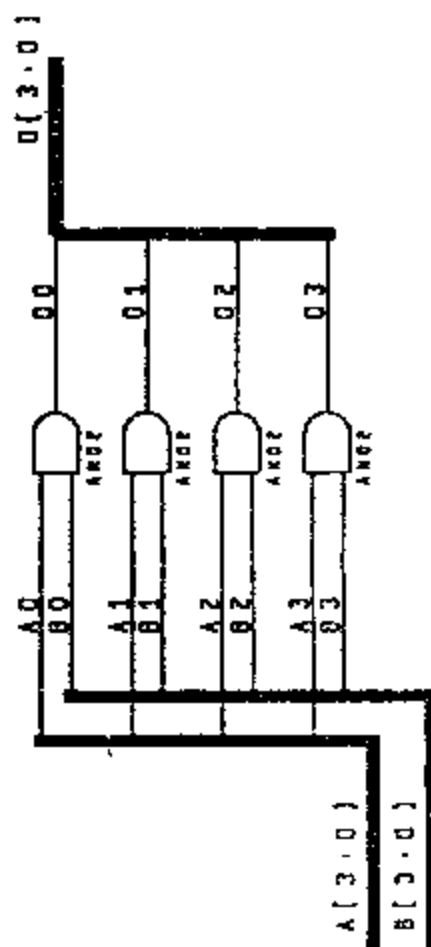


图 12.18



XILINX™	Case
Title: AND01E2	Tutorial Design
Reference: AND for Highpin	
Date: August 23, 1993	Application Group

图 12.19

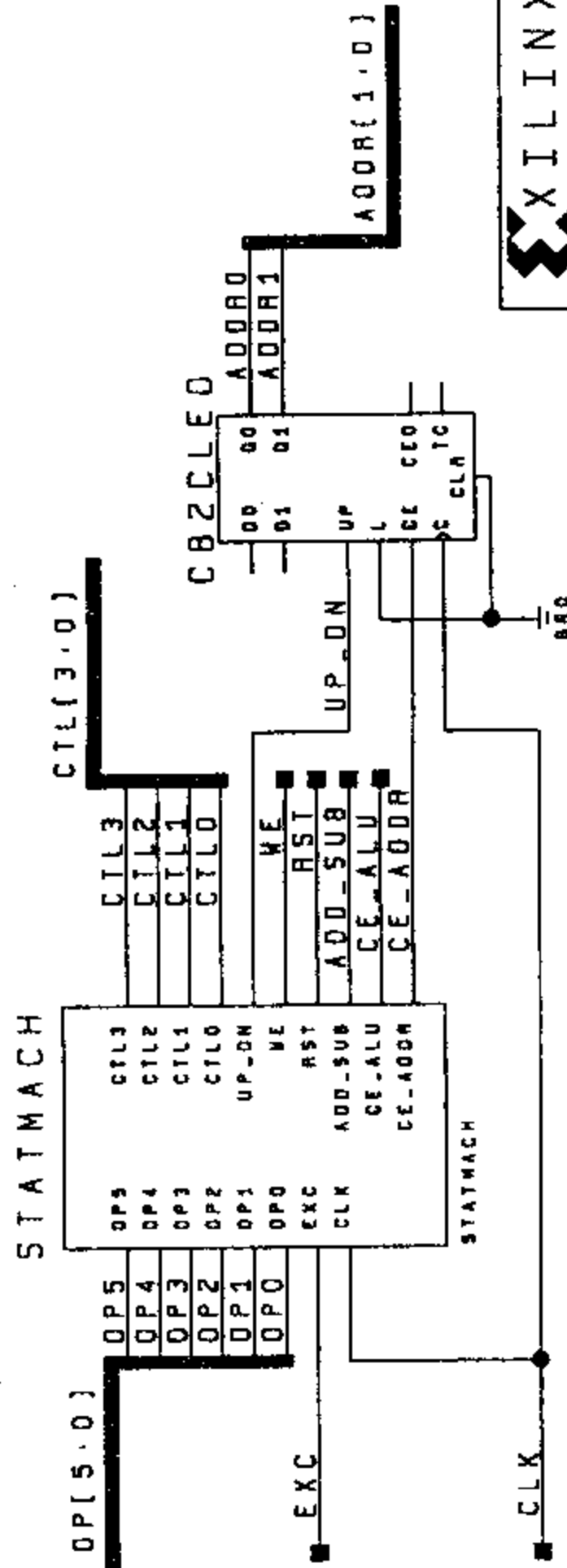
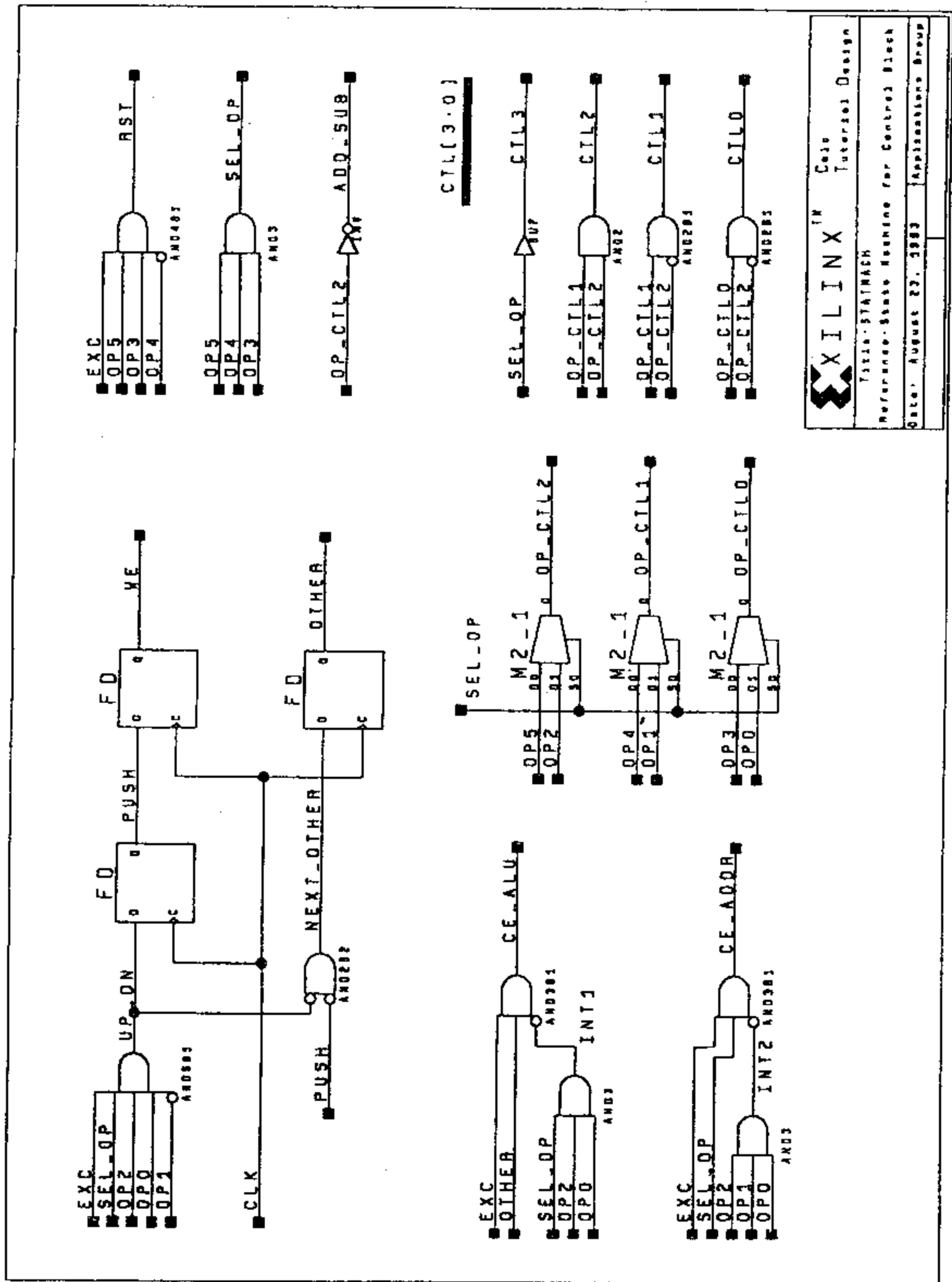


图 12.20

XILINX™	Case
Title: CONTROL	Tutorial Design
Reference: Control Logic	
Date:	Application Group



XILINX ™		Cell
		Interpol Design
Title: STATMACH		
Reference: State Machine for Control Block		
Date: August 27, 1983	Applications Group	

图 12.21

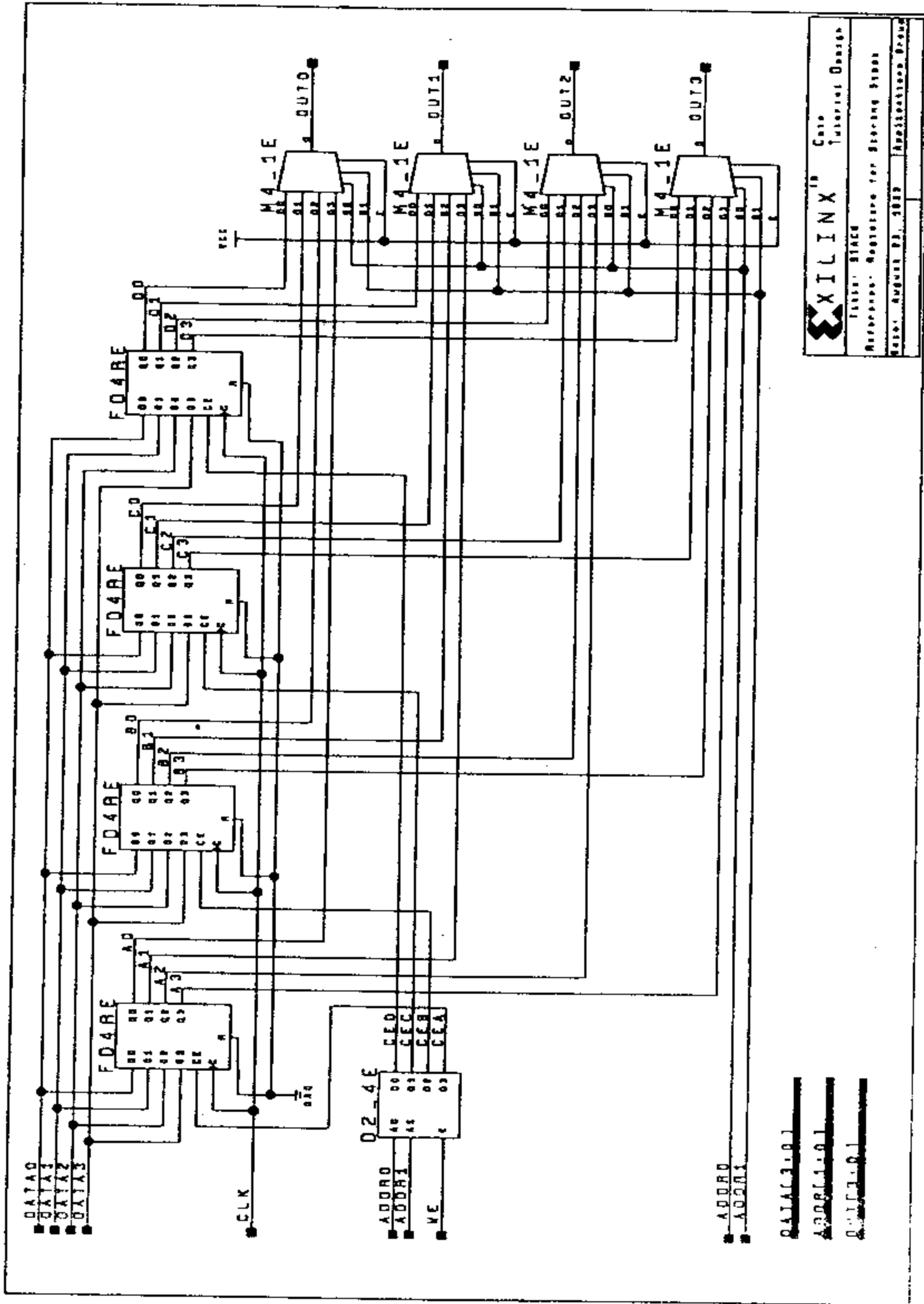


图 12.22

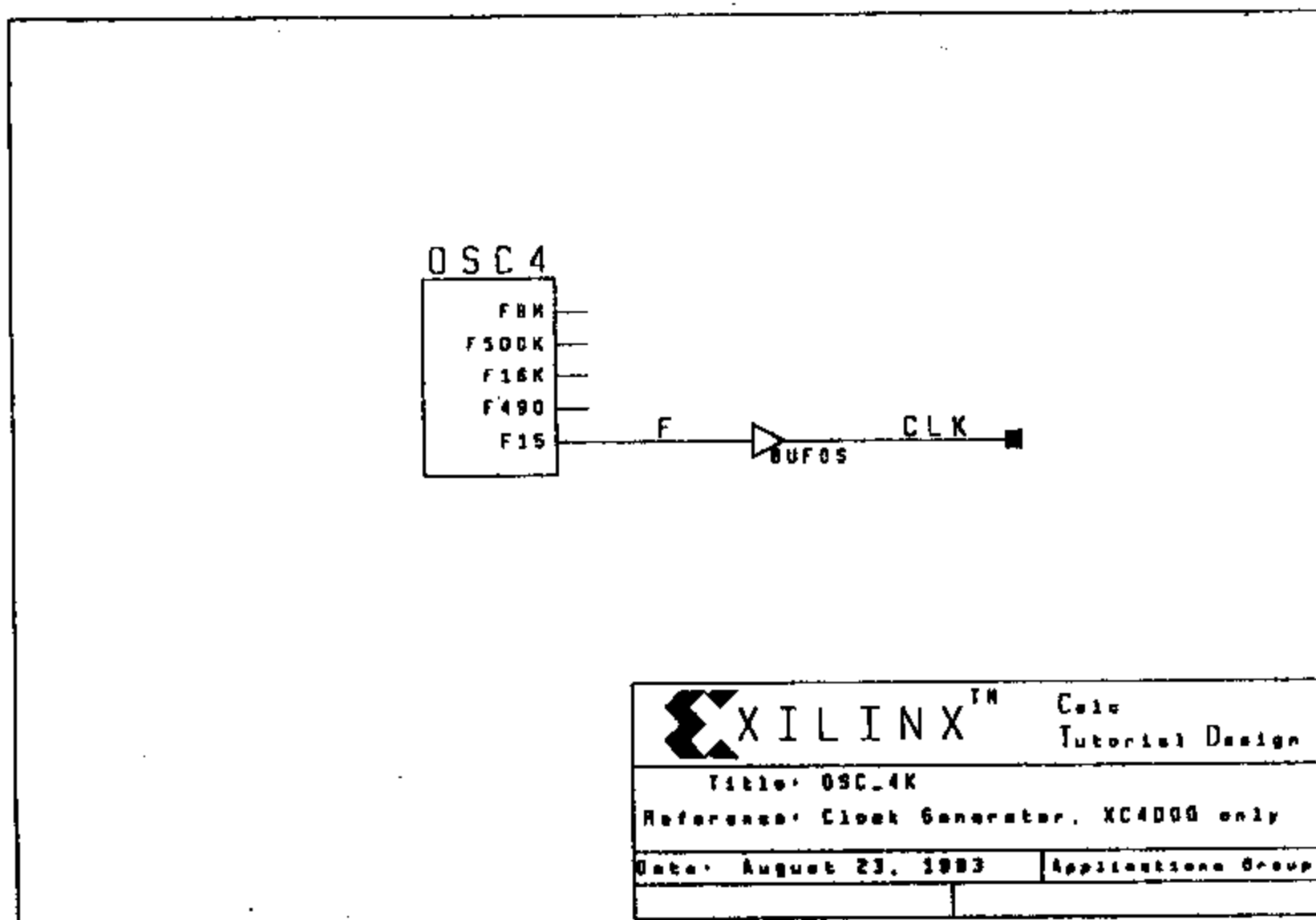


图 12.23

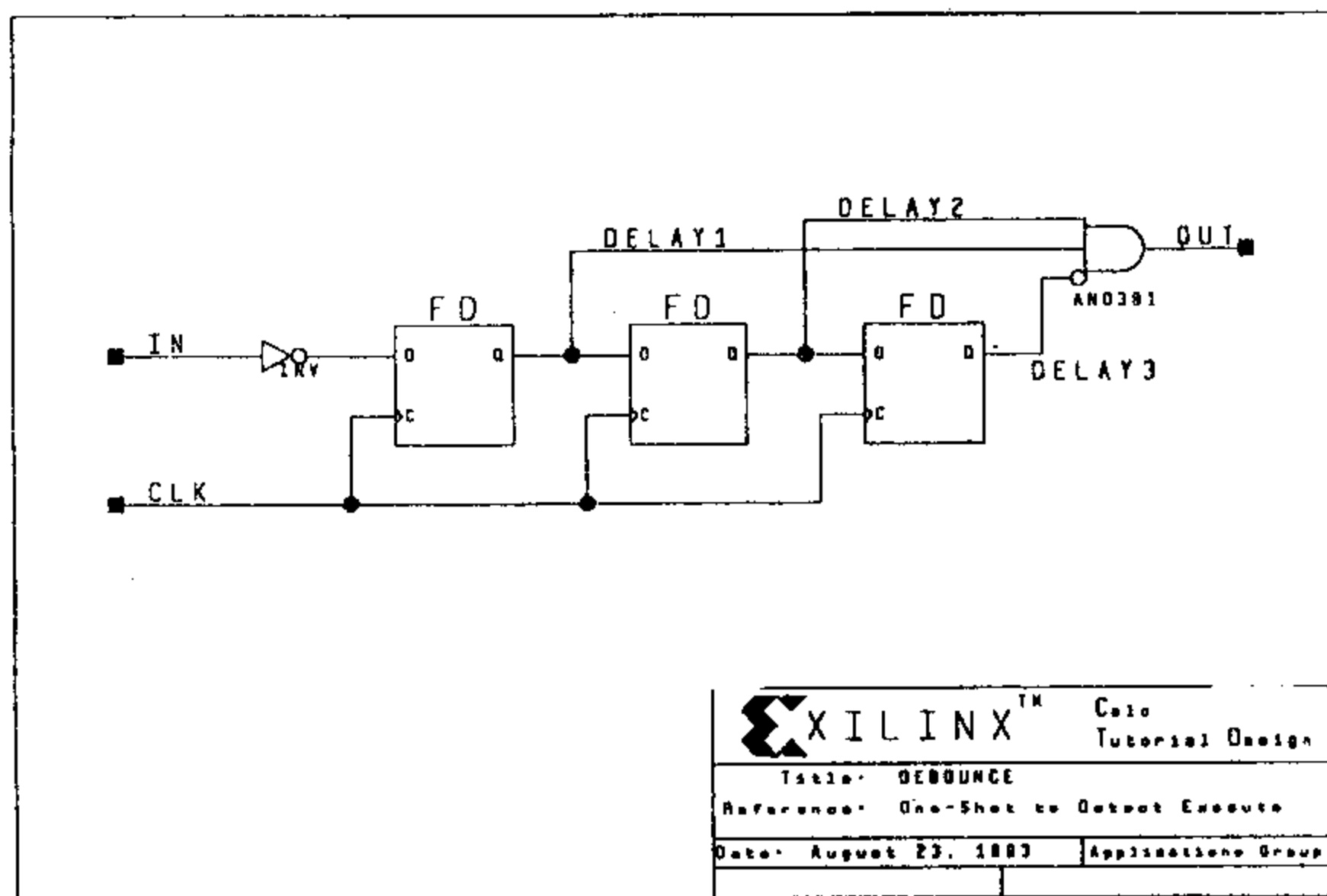


图 12.24

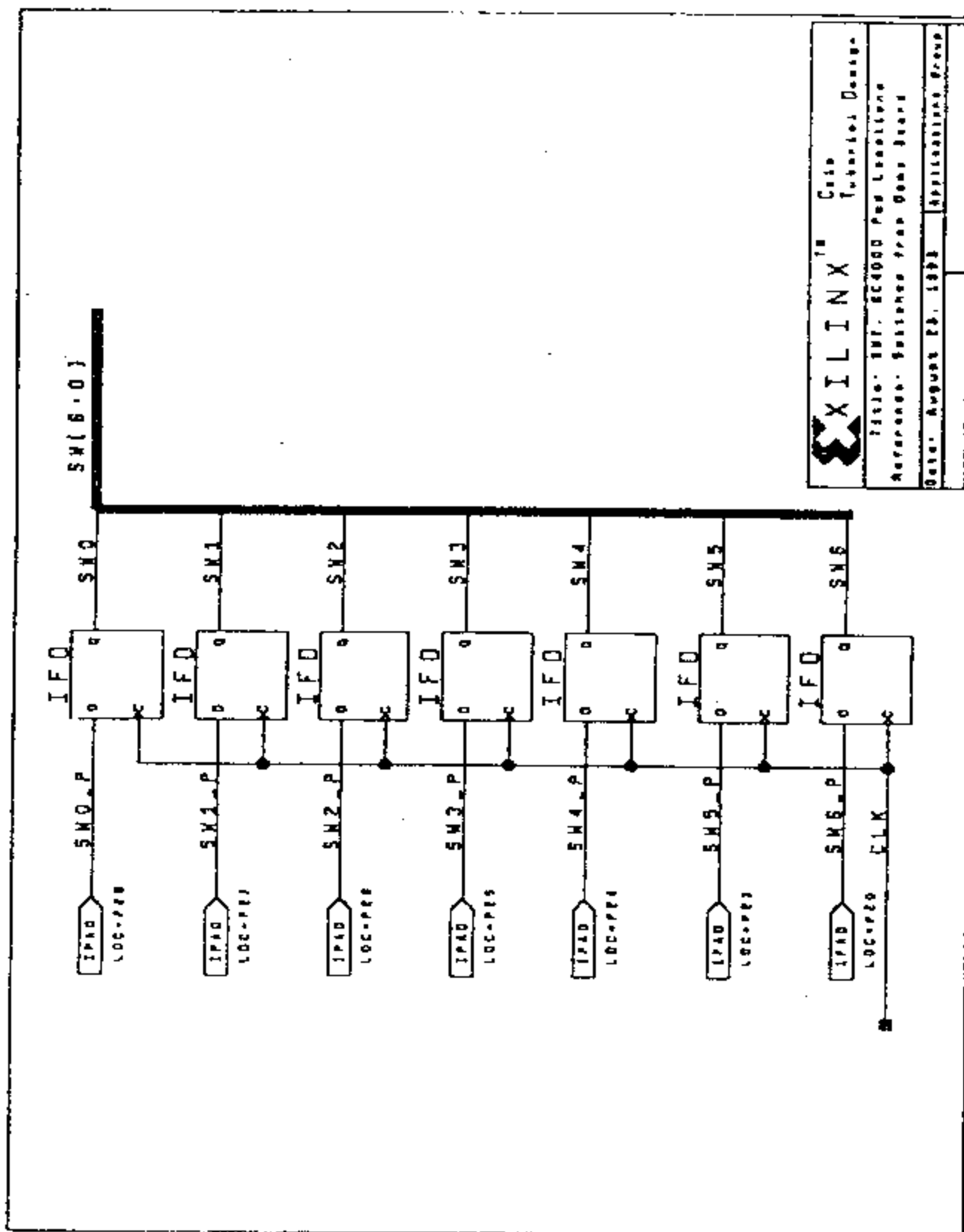


图 12.25

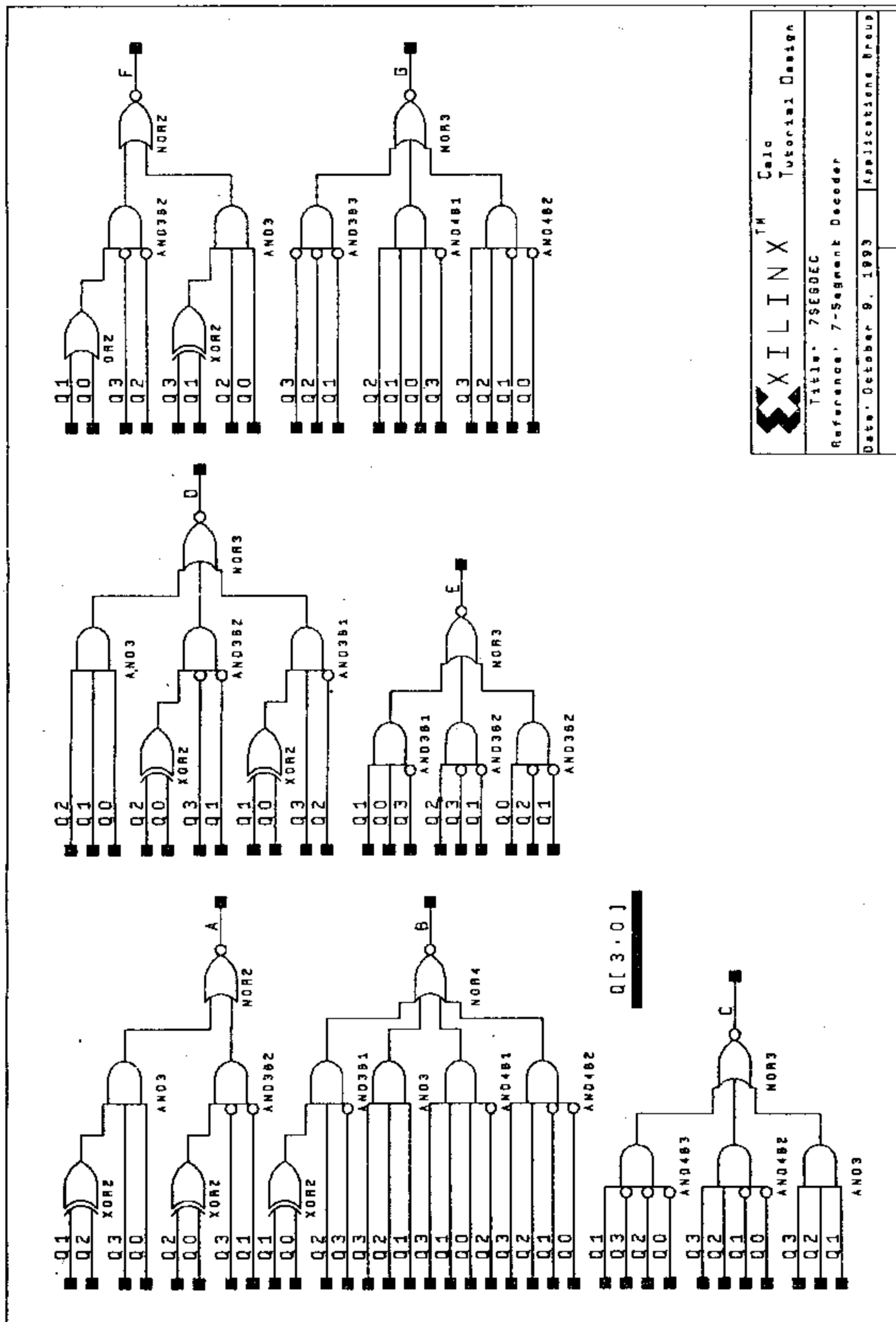


图 12.26

 XILINX™		Calc Tutorial Design	
Title: 7960DEC		Reference: 7-Segment Decoder	
Date: October 9, 1993		Applications Group	

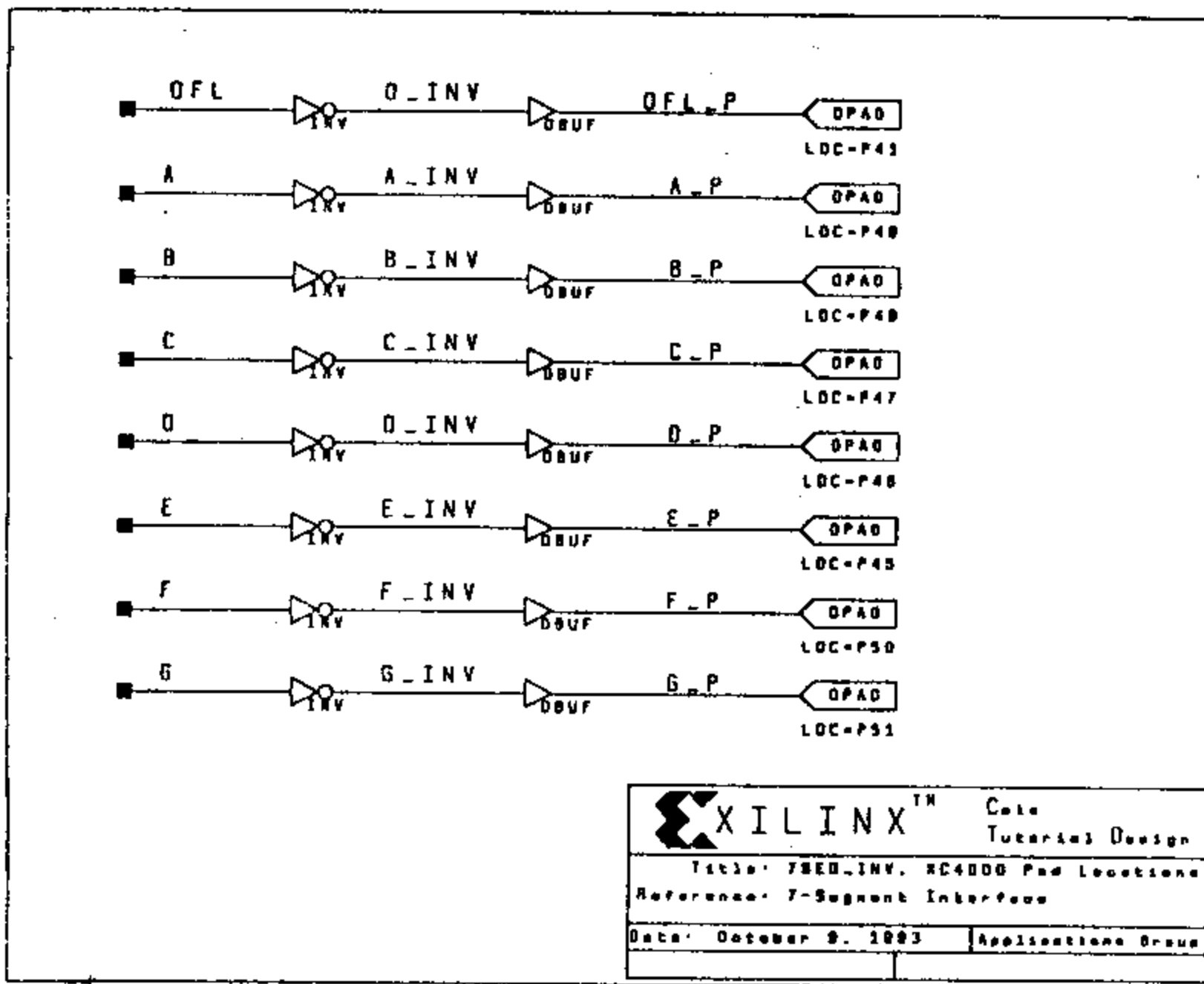


图 12.27

STACK[3:0]

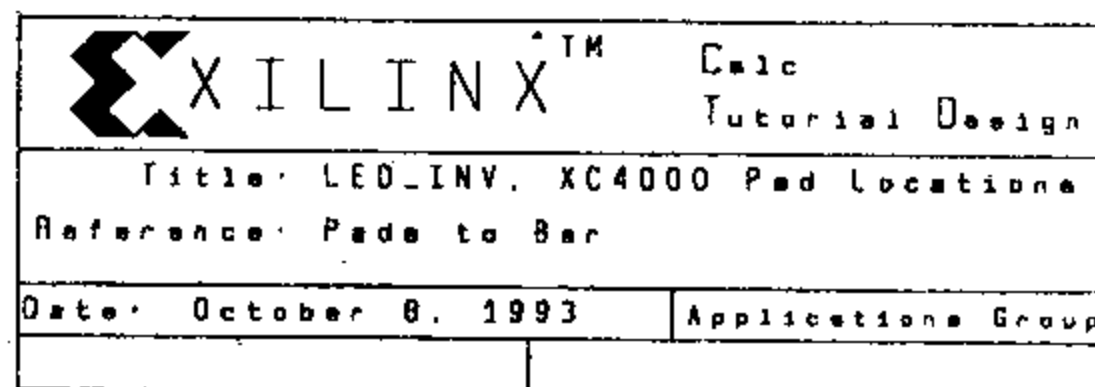
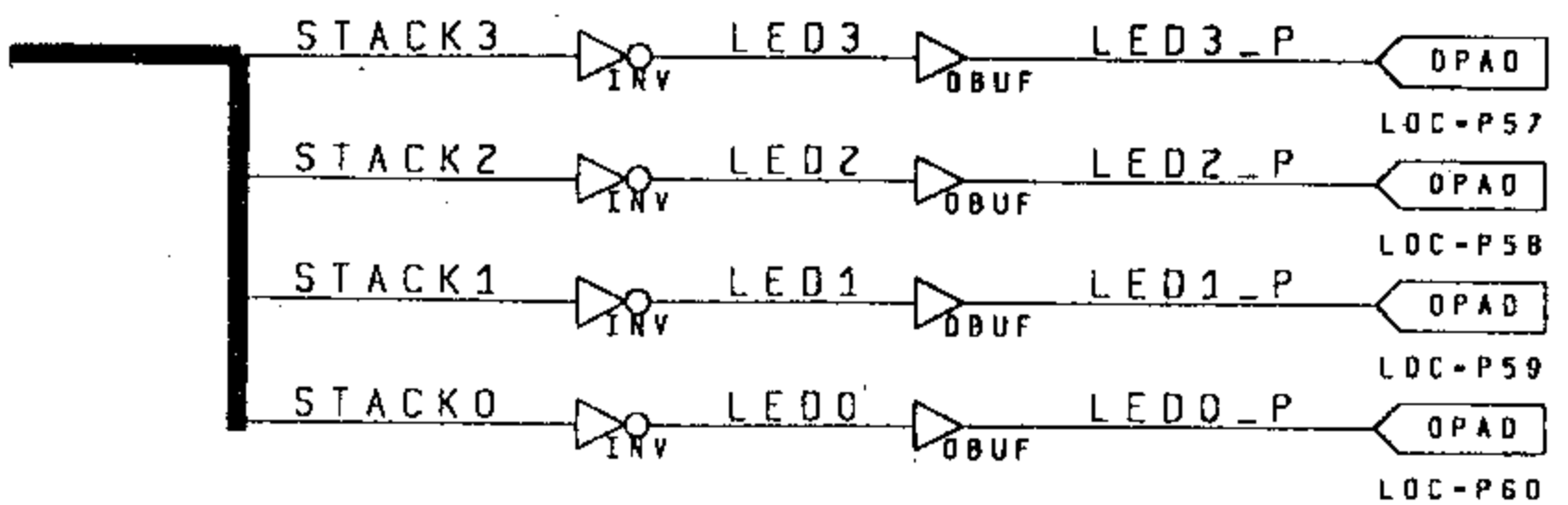


图 12.28

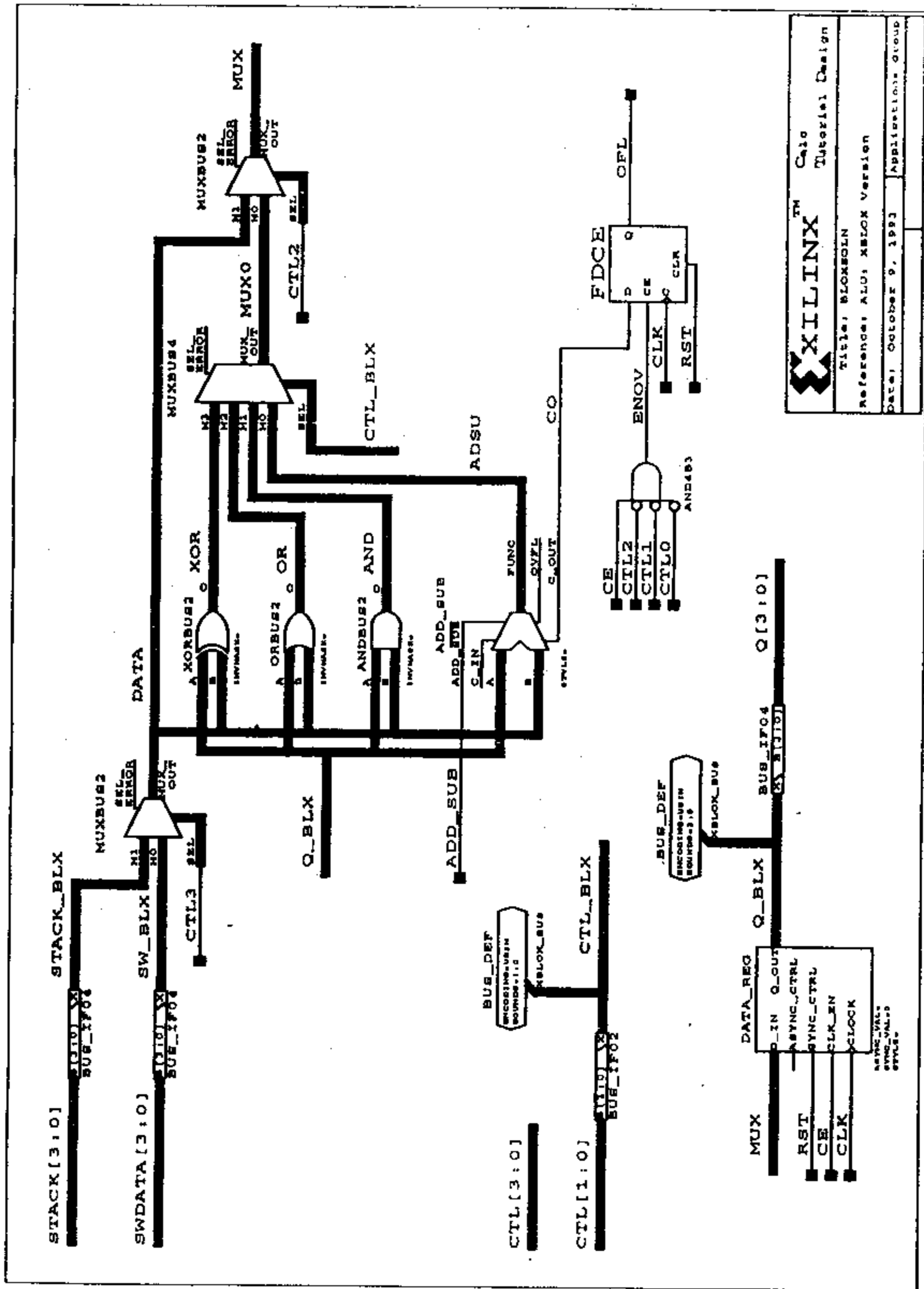
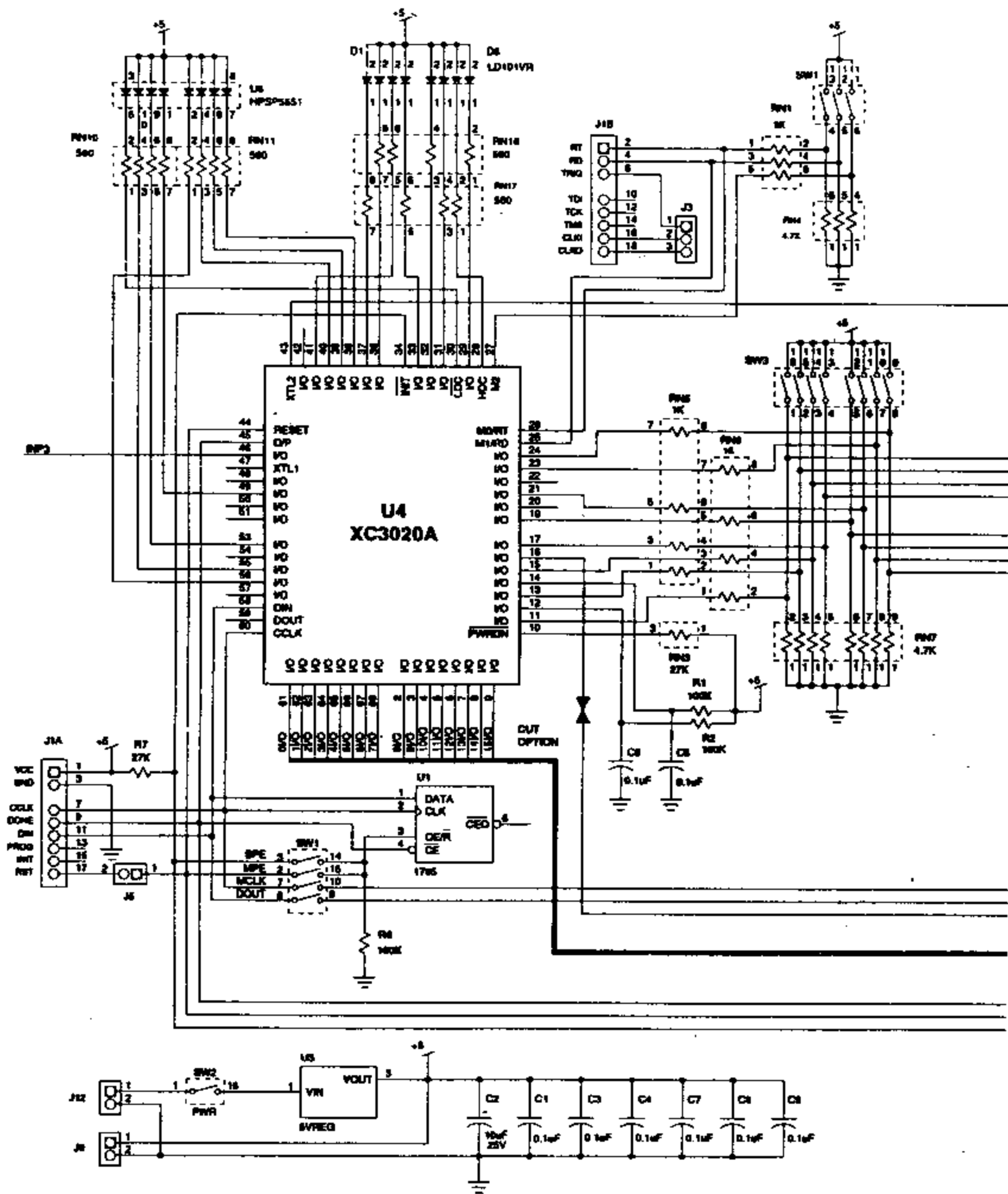


FIG 12.29



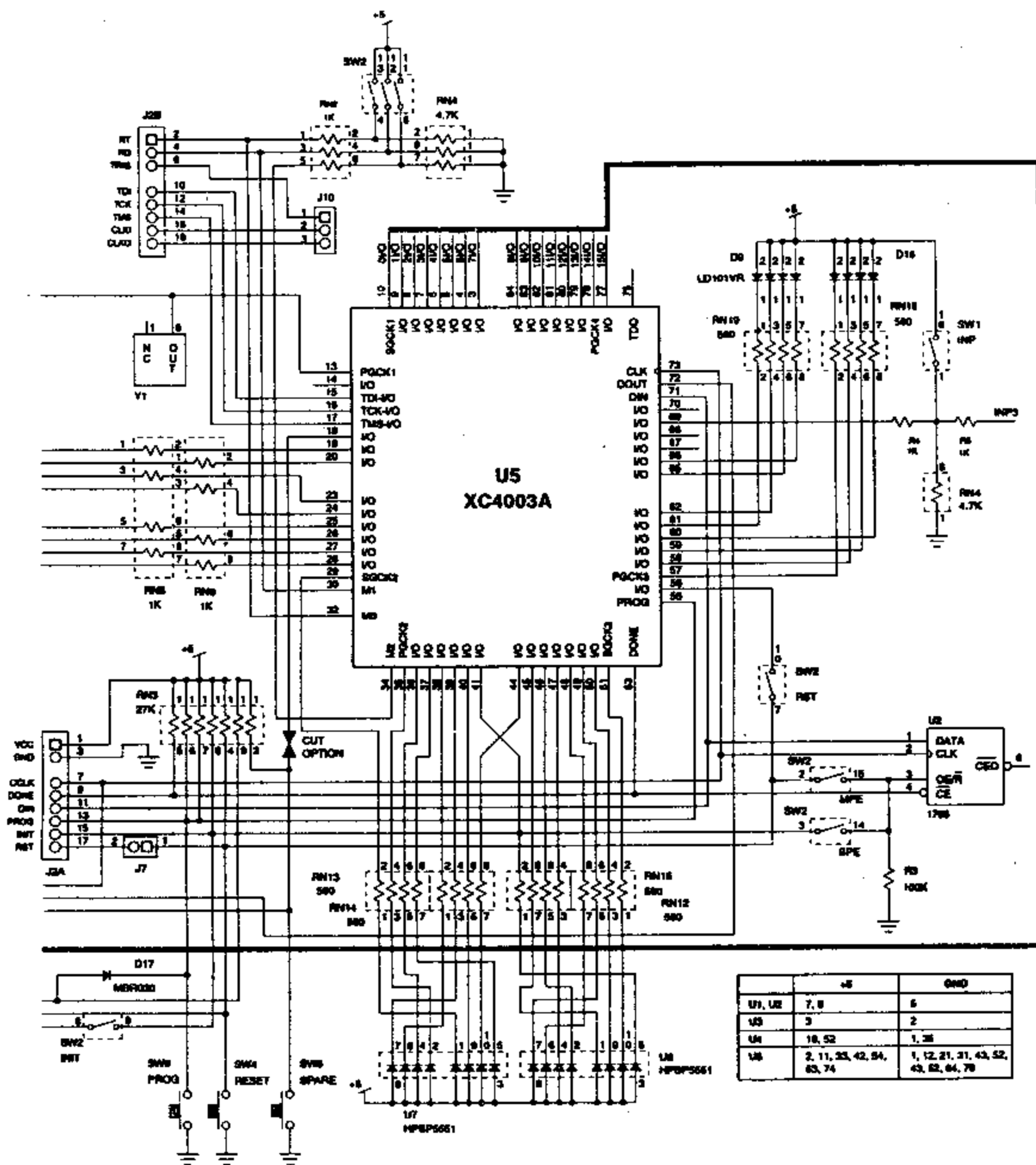


图 12.31

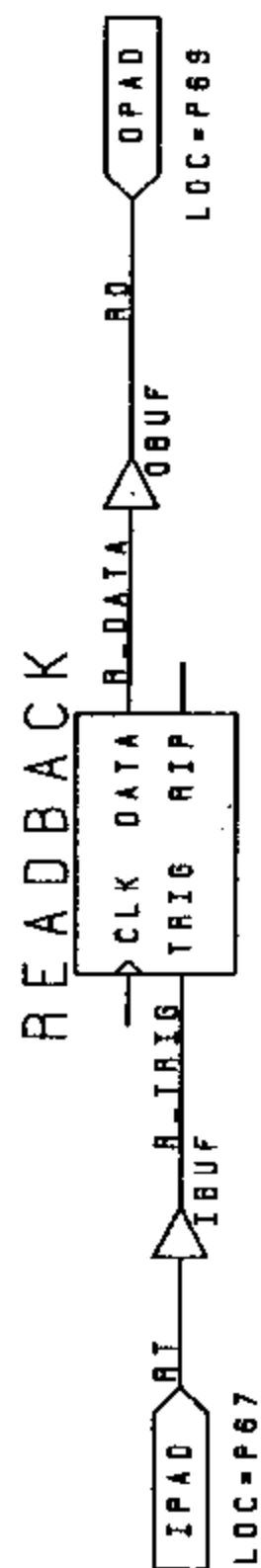
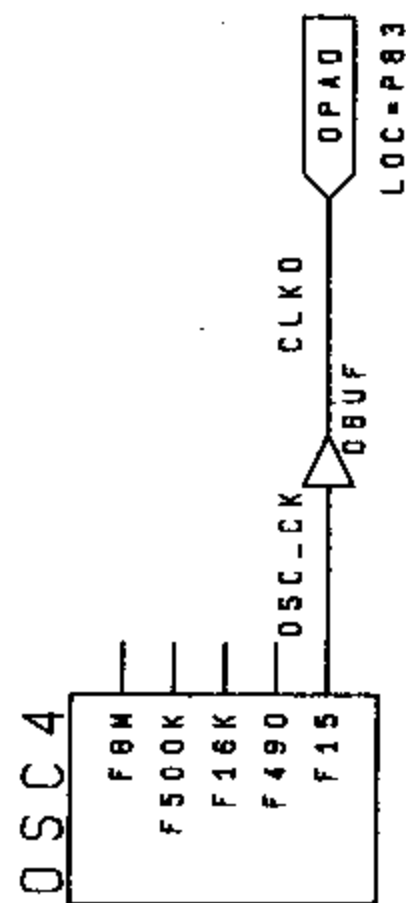


图 12.30

附录

附录 A XBLOX

对 Xilinx 优化的逻辑块 X-BLOX (Blocks of Logic optimized for Xilinx) 综合工具由模块库组成, 可以藉助高级函数代替门级基本单元用来描述一个系统。

由 X-BLOX 库提供的复杂函数补充 Xilinx 设计输入接口的宏单元和门级单元, X-BLOX 支持 XC4000/A/D/H, XC3000A/L 和 XC3100A 等 FPGA 系列。因为 X-BLOX 模块是可定制的, 每个模块可以描述上千个唯一的功能, 利用属性和把总线及网线接到模块的相应引脚可以定制这些模块。

§ A.1 X-BLOX 的特性

X-BLOX 提供以下四方面的特性, 可以简化和加速设计输入和设计修改的过程。

一、方框图设计输入

X-BLOX 允许在方框图级完成系统设计的主要部分, 利用中规模 (MSI) 和大规模 (LSI) 逻辑函数, 如加法器, 计数器, 比较器, 时钟分频器, 译码器, 通用移位寄存器, SRAM 和 PROM 等。

二、通用数据通道的大小和编码

X-BLOX 通过它的定义总线结构和输据格式的唯一方法简化设计输入和设计修改, 事实上, 每个数据通道的总线结构仅需要定义一次, 即使数据通道可能包含许多 X-BLOX 逻辑模块。总线的宽度和其上所载数据类型作为一个模块的部件被定义, 并自动地传播到整个设计和通过各个层次, 这使得仅在原理图的少数区域做修改而改变整个设计的总线尺寸。

三、优化实现

由 X-BLOX 综合工具配置的逻辑函数适合所需总线的宽度和芯片系列, X-BLOX 模块产生器使逻辑实现适合每个模块的特定需要和特定的 Xilinx 器件系列。例如, 比较器的实现方式取决于加到比较器的数据尺寸和要比较什么——比大, 比小或这些输出函数的组合。

X-BLOX 设计系统利用专为 Xilinx 优化技术增强综合设计实现的性能和密度, X-BLOX 软件利用芯片资源的专门知识与智能的逻辑实现技术结合产生快速和有效的电路设计, 因此, X-BLOX 模块产生器不仅节省了宝贵的设计时间, 而且产生优化的电路实现。

四、兼容性

X-BLOX 支持通用的原理图编辑器, 如 Viewlogic, OrCAD, Mentor 和 Cadence 等, 它也用于 Synopsys 的 FPG 编译器和其它第三方的支持 X-BLOX 的综合工具。

§ A.2 产生 X-BLOX 的设计

用 X-BLOX 进行设计要求把一个模块加到用第三方设计工具的原理图,当缺省的模式不适合时,把属性加到模块来规定模块;规定 X-BLOX 的总线并把总线接到 X-BLOX 的模块;产生层次符号简化顶层的设计;把 X-BLOX 的总线接到非 X-BLOX 的总线。

一、加 X-BLOX 模块到原理图

X-BLOX 的模块用原理图符号表示,X-BLOX 的符号分组在原理图输入接口提供的库中,X-BLOX 的库必须在原理图输入程序包的库搜索路径以便存取它。

每个 X-BLOX 的模块可以考虑作为一个有一种功能的属性单元(Template)。这些功能包括算术的,总线的,逻辑的,I/O 的,时序的和存储的功能等,表 A.1 列出这些 X-BLOX 的模块。

表 A.1 X-BLOX 模块的清单

计数器,寄存器和算术模块	
ACCUM	通用累加器
ADD-SUB	加法器和/或减法器
BUS-IFxx	总线接口,用于连接 X-BLOX 模块到非 X-BLOX 元件
COMPARE	比较两个数值的大小和/或相等性
COUNTER	通用计数器
DATA-REG	通用寄存器
DECODE	把任意编码的数据变换到一个有效的编码
INC-DEC	增和/或减一个常数
SHIFT	寄存器并行或串行地加载和/或移位数据,并移出数据
总线和连接性	
BIDIR-IO	定义一个或更多的双向 I/O 焊盘和缓冲器
BUS-DEF	规定总线的精度和编码
CAST	使能总线数据类型(精度和编码)的多种解释
ELEMENT	从一条总线提取一条网线(一位)
FORCE	设置总线为一个固定值
IUPUTS	定义一个或更多的输入焊盘和输入缓冲器
MUXBUS	选择输入总线的一个信号
MUXBUS2	选择两个输入总线的一个
MUXBUS4	选择四个输入总线的一个
MUXBUS8	选择八个输入总线的一个
OUTPUTS	定义一个或更多的输出焊盘和输出缓冲器
SLICE	从一条总线提取一子集网线

门级逻辑	
ANDBUS	与总线的所有输入网线产生一个单位的结果
ANDBUS1	一条总线的每一位和单个位相与
ANDBUS2	两条总线相与产生一总线输出
INVBUS	倒相一条输入总线被选位的数值
ORBUS	或总线的所有输入网线产生一个单位的结果
ORBUS1	一条总线的每一位和单个位相或产生一总线
ORBUS2	两条总线相或产生一总线输出
TRISTATE	宽三态总线
XORBUS	异或总线的所有输入网线
XORBUS1	一条总线的每一位和单个位异或产生一总线
XORBUS2	两条总线异或产生一总线输出
存储器	
PROM	通用可编程只读存储器
SRAM	通用静态随机存储器
时钟分频器	
CLK_DIV	时钟分频器

二、定制 X-BLOX 模块

X-BLOX 的部分能力是能够定制每个模块来表示许多函数,只要连接 X-BLOX 模块所需的控制腿和在缺省模式不是所需的时候规定模块属性做到这一点,利用属性和对某些属性赋予数据值可以定制模块。

1. 实现方式和运行模式

有几种方式在 Xilinx 的结构中实现 X-BLOX 的某些模块,称这些方法为格式(Style),某些格式利用较少的 CLB,但损失速度,而其它的格式利用较多的 CLB 达到较快的性能,可以对特定的模块规定格式,在不规定格式时,X-BLOX 利用缺省的最快的格式,一个格式属性可以赋予表 A.2 所列的 X-BLOX 模块

表 A.2 对模块的格式规定

模块	可能的格式
ACCUM ADD SUB INC DEC	ALIGEND UNALIGNED RIPPLE FAST3KA
ANDBUS	WAND DECODE
COMPARE	ARITH RIPPLE WIRED TREE
COUNTER	BINARY JOHNSON LFSR ONE_HOT
DATA_REG	CLB IOB ILD IFD OFD
SHIFT	LOGICAL CIRCULAR ARITH

如果不赋予模块一个格式,X-BLOX 会选择合适的格式,所以推荐不规定格式,而让 X-BLOX 选择合适的实现格式,因为它能选择最好的可能格式。

某些模块利用 STYLE 属性为模块规定运行模式,例如,SHIFT 模块的运行模式可以用属性 STYLE=ARITH,STYLE=CIRCULAR 或 STYLE=LOGICAL 来规定,得到算术的、循环的或逻辑的移位运行。

2. 数据值

某些模块的属性赋予数字值,数据值由算术基跟随一个所选基的数字数据值的表示式组成的,基是在2到36之间递归的十进制数,十进制是缺省表示的,不需要规定。作为例子,十进制数17可以表示为:

二进制 2#10001# 八进制 8#21# 十进制 17
十六进制 16#11# 十八进制 18#H#

数字数据值必需包含唯一的对规定基有效的字符一个不考虑的数(?)可以跟2,4,8,16和32的基使用;代表不考虑位与每个数字位通常在一起的数,例如,4#?2#=2#??10#;负数据值作为2的补码赖对待,用一个负号在数据值之前来表示,例如,-2#0011#=2#1101#=-3,但是不考虑位不允许在2的补码基中被规定;为了表示小数数据,可以利用基数点,例如,2#100.11#=4.75;可以利用下横线表示数的可读性,下横线上无数据,软件就不处理,例如,

2#00010010011100100110011101101001#
2#0001_0010_0111_0010_0110_0111_0110_1001#

表 A. 3 为可以作为数据值规定的属性。

表 A. 3 可作为数据值规定的属性

模块	要求一个数值的属性
ACCUM SHIFT DATA-REG	ASYNC-VAL SYYNC-VAL
COUNTER	ASYNC-VAL SYNC-VAL COUNT-TO
CLK-DIV	VALUE
DIVIED-BY DUTY-CYCLE	
FORCE	VALUE
ANDBUS/1/2 ORBUS/1/2 XORBUS/1/2 INVBUS	INVMASK DECODEMASK
SHIFT COUNTER OUTPUTS INPUTS CAST FORCE BIDIR-IO	BOUNDS
TRISTATE BIDIR-IO OUTPUTS INPUTS	FLOAT-VAL
PROM	DEPTH MEMFILE
SRAM	DEPTH

表 A. 4 利用各种基本数值的有效字符

基类型	基	有效的数据值字符
二进制	2	0 1 _ ?
八进制	8	0-7 _ ?
十进制	10	0-9 _
十六进制	16	0-9 A-F a-f _ ?
36	36	0-9 A-Z a-z _

3. 同步和非同步控制

同步和非同步控制出腿和属性决定如何初始化,设置或复位包含带触发器的模块,当同步控制被建立时,触发器在时钟上升沿被复位或置位,当非同步控制被建立时,触发器的复位或置位与时钟无关,X-BLOX 模块允许对相同的模块用每类控制的不同数值规定两类控制,这意味着 X-BLOX 允许设置整个寄存器非同步于一个数值,和设置此寄存器同步于一个不同的数值,这些数值用 ASYNC_VAL 和 SYNC_VAL 属性对 X-BLOX 的模块规定的常数,且相互独立

有同步控制(SYNC_CTRL)和非同步控制(ASYNC_CTRL)出腿的模块包括:ACCUM,CLK_DIV,COUNTER,DATA_REG 和 SHIFTB,除了 CLK_DIV 之外,可以对以上模块规定 SYNC_VAL 和 ASYNC_VAL 的属性值,在模块的 SYNC_CTRL 和 ASYNC_CTRL 输入的控制下这些数值被加载进寄存器和计数器。

ASYNC_VAL 属性值在芯片加电期间和 X-BLOX 模块的 ASYNC_CTRL 口上升沿被加载,这个加载有超过任何时钟有效加载的特权,ASYNC_VAL 属性值复盖所有其它输入,并与时钟使能和同步控制数值无关地被加载;SYNC_VAL 的常数当 X-BLOX 模块的 SYNC_CTRL 输入在时钟上升沿和时钟使能期间为高时加载进寄存器 SYNC_CTRL 通常有超过在相同模块上其它同步功能的特权,如果 ASYNC_CTRL 和 SYNC_CTRL 输入没有被连接,这些功能不被综合。

加电复位和初始化:可以利用 SYNC_VAL 属性定义一个在加电时加载的常数,如果不规定 ASYNC_VAL,除了线性反馈移位寄存器的计数器(LFSR)之外,所有的寄存器和计数器被置位于零,LFSR 计数器加载电时置位于它们的起始计数状态。

4. 总线选通功能的倒相和译码时标

INVMASK 和 DECODEMASK 属性单个地对带有总线输入的某些 X-BLOX 模块规定倒相或时标的输入(高有效或低有效)。

INVMASK 和 DECODEMASK 属性

INVMASK 和 DECODEMASK 属性对执行总线选通功能的 X-BLOX 模块是有效的,这些模块包括:ANDBUS/1/2,ORBUS/1/2,XORBUS/1/2,INVBUS,对它们的输入 INVMASK=1 或 DECODEMASK=0 将是低有效,而对 INVMASK=0 或 DECODEMASK=1 的输入将是高有效,利用前节的数据数值中规定的任何基规定倒相或译码时标。

缺省值的选择是直觉的,所以对某些模块是不同的,对与,或,异或的总线,缺省值是 INVMASK=0 或 DECODEMASK=1,所以无倒相在输入端执行,对 INVBUS 模块,INVMASK 表示总线的哪一位将被倒相,缺省值是倒相总线的所有位,这是从宽总线倒相器所期望的。

单输入总线模块

作为例子,五位 ANDBUS 有 INVMASK=2#10010#,则位1和位4被倒相,相同的是 DECODEMASK 为真,当 DECODEMASK=2#01101#,位1和位4被倒相。

双输入总线模块

当加 INVMASK 或 DECODEMASK 属性到一符号,这个特性是加到此符号的所有输入端,例如,对于两输入的模块,相同的 INVMASK 加到两者的输入,这个规则应用于 ANDBUS2,ORBUS2 和 XORBUS2。

数学上,加这些时标的一个到 XORBUS2的两个输入与任一个不加其上是相同的,当要把 INVMASK 或 DECODEMASK 仅加到两总线输入的一个,INVBUS 符号在相应的输入之前;当对任一输入需要任何唯一的时标将带有 INVBUS 的总线符号的 INVMASK 与它本身的时标组合起来。

单总线选通模块

对一个 ANDBUS1,仅输入总线受 INVMASK 的影响。

INVBUS 模块

X-BLOX 的 INVBUS 模块有对 INVMASK 属性相反的缺省值,当不定义 INVMASK 时,所有连接到 INVBUS 模块的信号被倒相,INVBUS 的输出由 INVMASK 的位模式决定,对于 INVMASK 昨为零的每一位,在总线中相应的位不倒相,对于 INVMASK 昨为1的每一位,在总线中相应的位被倒相,可以用任何基规定 INVMASK 的值。

5. I/O 焊盘的上拉和下拉电阻—FLOAT_VAL 属性

FLOAT_VAL 属性决定是否在 I/O 和三态的 TRISTATE 模块中利用上拉或下拉电阻,可以用任何允许的基规定此值,这个值可以包含不考虑的数,例如,带有 ENCODING=UBIN,BOUNDS=7:0和 FLOAT_VAL=16#?3#规定焊盘7到4不接电阻,焊盘3和2接下拉电阻,焊盘1和0接上拉电阻。也可以把可以分别规定一个上拉电阻,FLOAT_VAL=PULLUP,两个上拉电阻,FLOAT_VAL=PULLUP_D,或一个下拉电阻,FLOAT_VAL=PULLDOWN 把一个符号的所有焊盘接到上拉或下拉电阻,两个上拉电阻比一个电阻驱动更大的功率,但支持更快的转换时间。

6. 超范围指示

选择或寻址由 X-BLOX 综合的逻辑的模块是 DECODE,MUXBUS,PROM 和 SRAM 模块,这些模块有构造在它们符号内的超范围指示,在 DECODE 模块和所有 MUXBUS 模块中,这个输入称为 SEL_ERROR,在 PROM 和 SRAM 模块中它称为 ADDR_ERROR,以下表格给出这些指示器是如何表现的,X-BLOX 也考虑到 SEL 输入可能在哪里标记未连的 MUXBUS 输入或 DECODE 输出。

表 A.5 DECODE 超范围指示

DECODE;D_OUT[n;m]上的边介		
SEL 输入值	SEL_ERROR	D_OUT
不在 n 和 m 之间的值	高	0
标记一个未连 D_OUT 元件的值	高	0
表示所有其它情况的值	低	在序数的 SEL 上为高

表 A.6 MUXBUS 超范围指示

DECODE;D_OUT[n;m]上的边介		
SEL 输入值	SEL_ERROR	MUX_OUT
不在 n 和 m 之间的值	高	0
标记一个未连 D_OUT 元件的值	高	0
表示所有其它情况的值	低	在序数的 SEL 上为高

表 A.7 MUXBUS_n 超范围指示

MUXBUS, MUX-IN[n:m] 上的边介		
SEL 输入值	SEL_ERROR	MUX_OUT
比 n-1 大或 < 0 的值	高	0
标记一个未连 MUX-IN 元件的值	高	0
表示所有其它情况的值	低	在 MUX-IN 的序数的 SEL 上为高

表 A.8 PROM, SRAM 超范围指示

PROM, SRM, DEPTH=n		
ADDR 输入值	ADDR_ERROR	D_OUT
比 n-1 大的值	高	0
表示所有其它情况的值	低	ADDR 的内容

三、表示 X-BLOX 总线

X-BLOX 如何管理总线是它的另一个特点, X-BLOX 总线包括以下特殊性。

1. 规定总线和总线标记

所有 X-BLOX 模块和总线在尺寸上是通用的, X-BLOX 表示所有总线尺寸, 或者用无范围的总线, 或者用单位网线编码, 取决于原理图输入软件包, 例如, Viewlogic 和 Mentor 利用无范围总线, OrCAD 利用网线。为了标记接到 X-BLOX 模块的 X-BLOX 的总线, 利用一个标记命名总线, 在标记上没有规定总线范围, 对比以下的 X-BLOX 和非 X-BLOX 总线的规定:

X-BLOX 总线的规定: DATA 非 X-BLOX 总线的规定: DATA[15:0]

在标记 X-BLOX 的总线之后, 在任何相同的数据通道上规定边介和为总线编码, 当运行 X-BLOX 时, 它综合总线把它们扩展到适合的尺寸, 产生仿真模型, 并执行芯片级的实现, 通用尺寸总线允许原理图快速地被重新尺寸化, 因为一条总线的 BOUNDS 属性对每个数据通道仅需要规定一次, 利用对总线的编码达到模块的适合的实现, 如果需要改变第二条总线的编码, 利用 CAST 模块连接两条总线。

2. 总线数据类型

在 X-BLOX 中, 一条总线不只是连线的集合, 一条总线定义通过这条总线传输的这种数据, 因为在构造的计算机语言中, 这个数据有一个数据类型, 一条总线的数据类型由编码方案和总线精度来定义, 必须分配两个属性, BOUNDS 和 ENCODING, 来建立数据类型, ENCODING 属性定义由 X-BLOX 支持的数据编码方案, BOUNDS 属性定义总线上左和右的位的指示, 例如, 一个八位无符号二进制总线有八个位, 可以从 7 降到 0 来表示, 在此情况, BOUNDS 属性为 BOUNDS=7:0。

必须沿每个 X-BLOX 数据通道规定至少一个输入, 输出或 BUS_DEF 分量的数据类型以传播此数据类型到数据道的其它模块。

编码

对一条数据的编码方案用称为 ENCODING 属性规定,此编码使能 X-BLOX 处理不同于2的补码的无符号数,或按照所用的编码处理一个有效的编码数,X-BLOX 软件用这个信息确保像加法器,比较器,多路开关和译码器等模块对所用数据通道正确地综合。

X-BLOX 现在支持三种有 ENCODING 属性的编码方案,如表 A. 9所示。

表 A. 9 X-BLOX 编码方案

编码	描述
BIT 或 UBIN	无符号二进制编码
ONE_HOT	n 个之一编码——一次仅一条线被置于逻辑1
TWO_COMP	2的补码

X-BLOX 不校验由 X-BLOX 总线环境外部来的 ONE_HOT 数据的有效性,X-BLOX 模块在需要时产生有效的 ONE_HOT 数值,除非此模块被禁止,例如,当使能为低时,DECODE 模块被禁止,在此情况,输出为零,它不是有效的 ONE_HOT 数值。

用 UBIN,ONE_HOT 或 TWO_COMP 编码,BOUNDS 属性必须被规定,对 BOUNDS 没有缺省值,但是,对 BIT 编码,属性 BOUNDS 是可选的,当 BOUNDS 用 BIT 编码规定,它等效于 UBIN 编码,如果属性 BOUNDS 不用 BIT 编码规定,结果是标量网线,等效于规定 ENCODING=BIT。

边界

用 BOUNDS 属性规定数据类型的精度(总线边界),边界由一对用冒号分开的整数表示,第一个数是左边界,为一个数的最高有效位(MSB),第二数是右边界,为一个数的最低有效位(LSB),一个八位口通常有 BOUNDS=7:0,其最左位标记7,最右位标记0。

大端与小端的比较

当一条总线的 MSB 标记比 LSB 标记要大时,例如,BOUNDS=9:0,被称为大端,因为大的端(最大位标记)是在左端,当 MSB 比 LSB 小时,例如,BOUNDS=0:9,称为小端,因为小的端是在左边,注意到 MSB 总是在左边,LSB 总是在右边。

边界有一个重要的解释总线上数值的规则,在大端总线,位的标记规定位的权,例如,位4为权 $2^4=16$,0位(权为 $2^0=1$)是刚好在二进制点的左边,位-3为权 $2^{-3}=1/8=0.125$,在小端总线,对二进制点没有标准的位置,所以 X-BLOX 把二进制点放到小端总线最右位的右边,用作选择器的总线上的数值决定 X-BLOX 多路开关和译码器上 SEL 口的作用,选择器的值取决于选择器的编码和边界。

一位阵列作为大端来处理,BIT0:0为权 $2^0=1$,BIT4:4为权 $2^4=16$,BIT-3:-3为权 $2^{-3}=1/8=0.125$,如果需要带权为1的单个位,或 BOUNDS 区域为空,或置它为0:0。

大端和小端的 ONE_HOT 数据总有对应于位标记的权,例如,ONE_HOT(3;-1)的位值 $10000=3$,而 ONE_HOT(-1,3)位值 $00001=3$ 。

3. 数据类型的传播

X-BLOX 利用数据类型传播使在一个设计中必须规定的编码方案和精度的次数和位置最小,X-BLOX 传播赋予一个模块总线上的 ENCODING 和 BOUNDS 属性到在数据通

道中所有其它的被连的模块和总线,这个保证总线尺寸和编码方案的一致性,使得全局修改容易得多。例如,一个八位无符号二进制总线接到一个加法器,加法器输出接到另一个八位总线,这条总线接到一个八位寄存器,则必须在八个地方规定数据类型,在加法器的三个口上,寄存器的两个口上,三个总线的尺寸,改变这些数据类型的一个而不改变其它的可能导致错误,用 X-BLOX 模块完成相同的设计,要求在数据通道规定 ENCODING=UBIN 和 BOUNDS=7,0 的属性。

在 X-BLOX 设计系统中对芯片的外设或功能规定 ENCODING 和 BOUNDS 属性,两属性传播到其它的模块和总线,利用 ENCODING 和 BOUNDS 属性的 X-BLOX 模块有: INPUTS, OUTPUTS, BIDIR_IO, BUS_DEF, CAST, DATA_REG, COUNTER, PROM, SHIFT 和 SRAM。INPUTS, OUTPUTS 和 BIDIR_IO 模块把电路连接到焊盘和器件,大多数赋予 ENCODING 和 BOUNDS 属性的相应的位置是在其它芯片的总线接口处; BUS_DEF 模块用于定义内部总线的数据类型和不接到 I/O 网络的模块口。

4. 产生分层的符号

这部分说明如何产生一个分层符号,它包括一个或更多接到 X-BLOX 总线的出腿。

总线尺寸

就像规定 X-BLOX 总线标记不规定范围一样,当产生一个分层符号时,规定 X-BLOX 的总线出腿不规定范围。

用 Viewlogic 编辑器,视觉上帮助识别这些是通用的总线,用以下的指令可以画这些接口出腿像总线接口出腿一样宽:

- 1) 加 X-BLOX 出腿到现有的符号;
- 2) 围绕出腿画一个框;
- 3) 填满这个框,参考任何 X-BLOX 的符号。

数据类型

在单张原理图纸中,必须对每个利用 BOUNDS 和 ENCODING 属性的数据通道规定至少一次的数据类型,在分层设计中是相同的;规定每个数据通道至少一次数据类型无论数据通道是不是通过层次的边界。

如果底层原理图的总线尺寸不改变,可以定义数据通道边界和在子电路图级编码,但是在底层原理图中不规定数据类型,这个原理图变成通用的,用主原理图中规定的数据类型的定义可以重新定尺寸。

数据类型传播

当在一级上而不是在子电路级上定义 X-BLOX 总线的数据类型,这个 X-BLOX 总线的边界或编码的任何变化传播到整个数据通道,包括子电路。

5. 信号混淆现象

在一个 X-BLOX 的设计中,所有的信号线和总线有唯一的标记或命字,大多数总线代表唯一的数据通道,但是当一条或多条连线用 CAST 符号相等或藉助 ELEMENT 或 SLICE 模块从总线中提取时,将有不同的名字赋予相同的逻辑连线:

- 1) 对接到 CAST 模块的两条不同命名的总线,在总线中对应信号的每一个是与另一条相等的;
- 2) 用 ELEMENT 模块从一条总线中提取信号线将有多多个名字,一个是单个线,另一个

是在总线中相等的网线；

3)包含在切分器(SLICE)中的每一条连线,或用 SLICE 模块从较大的总线中选出的连线子集合也将有多个名字,来自主总线的名字,来自于子总线的名字。

一个混淆是一条网线通常为设计中所有相等网线所参考,利用混淆,因为 XNF 文件仅支持一个名字为设计中的任何给定的网线。SLICE,ELEMENT 和 CAST 模块仅当总线和网线没接任何逻辑或缓冲器时来实现,软件组合从这些元件得到的网线名字,仅维持为最大容纳总线支持的名字,除非 KN(KEEPNAME)属性加到网线或总线,所有其它的名字将不在 XNF 文件中,但将有信号的混淆。

X-BLOX 决定信号的混淆靠把包含信号的最大总线的标记赋予较小的总线或信号,只要可能,移去赋予这些总线的原始标记。没有保留的信号名字由"Signal Aliases"混淆,原始名字和较小总线的混淆在.blx 的文件中报告。在.blx 文件中给出最终的信号定义和混淆,没有被混淆的或与其它网线有混淆的网线可以直接在仿真期间被参考。一些原理图输入软件可以利用信号混淆的信息,如果支持这一点,在原始设计中没有吸收进 CLB 或 IOB 的所有网线名在仿真期间可以被存取;如果软件不支持此特性,必须参考它的"信号混淆"的混淆网线,因此,当仿真期间不可以参考 X-BLOX 总线,因为不在参考一个混淆,或者网线已被吸收进 CLB,参考.blx 文件决定在仿真激励文件的混淆名字。

为了保持一个名字,而不是由软件选择的的名字,规定可选的 KN(KEEPNAME)属性到所需的总线和信号上;仿真指令文件应参考最大等效的 X-BLOX 总线的标记,否则,指令文件在时序仿真时不工作。

6. 总线操纵

利用以下专门的模块来操纵总线。

CAST 符号

利用 CAST 模块连接总线,总线有以下情况:

- 1)相同的尺寸,但不同的编码;
- 2)相同的尺寸,但不同的边界(范围);
- 3)相同的尺寸,但不同的编码和边界;

表 A. 10给出如何按所要做的任务对腿 A 和腿 B 规定 BOUNDS 和 ENCODING 属性。

表 A. 10 规定 BOUNDS 和 ENCODING 属性

功效	边界和编码	
	A 腿	B 腿
改变边界	7:0	11:4
	7:0	4:11
	7:0	0:7
	7:0	2:9
改变编码	BIT UBIN TWO_COMP ONE-HOT	BIT UBIN TWO_COMP ONE-HOT

此外,可以用 CAST 模块按三种不同方式对模块上的腿(A 和 B),仅一个腿(A 或 B)规定数据类型,或两个腿的任一个不规定此属性。

ELEMENT 符号

ELEMENT 符号用来从一条总线提取单根连线或把单根连线接到一条总线,为了利用它,用 ELEMENT 属性规定从总线被连接或提取的连线的标记。

FORCE 符号

FORCE 符号对一条总线强加一个值,用它来规定一个固定输入值到计数器模块的并行 D-IN 上,或涉置比较器模块的一个输入为一个固定值,当这个值总是存在总线上,可以利用 TRISTATE 模块控制存取 FORCE 值的部分时间,这个符号也可以用来定义此值的数据类型和附加 FORCE 符号的总线。

SLICE 符号

SLICE 符号提取较大总线的一部分,或把较小总线汇集到大的总线,为了方便,它重新标记一组连线注意 MAIN 和 SUB 代表 X-BLOX 的总线,在这个符号上的属性是互相补充的,不需要同时利用。

MUXBUS 符号

MUXBUS 符号是用于在 Select 输入口控制下将 n 个输入的一个皆到输出的多路开关。

7. 位置属性

靠对 X-BLOX 模块放置 LOC 或 LOC[i] 属性可以规定由 X-BLOX 综合的逻辑的布局,LOC 属性应用于所有为 X-BLOX 模块综合的逻辑,LOC[i] 属性仅应用于 X-BLOX 第 i 个元件仅当模块的数据类型是一个阵列(不是 BIT)时可以利用,赋予 LOC 和 LOC[i] 属性的数值是 FPGA 的 CLB, TBUF, IOB 或沿边译码器的位置。

LOC 属性可以用数据类型 BIT(不是边界)规定一个模块的位置,例如,LOC=AA(CLB 的位置 AA),LOC=TL(I/O, 译码逻辑或全局缓冲的左上边)。

LOC 属性也可对 X-BLOX 模块放置,其数据类型包括 BOUNDS,在此情况,属性必须规定一个位置的范围,例如,LOC=TBUF_R * CO. 1(第 0 列中任意一行的 TBUF)。

LOC[i] 属性与对 X-BLOX 模块的第 i 位综合的逻辑有关的 LOC 值,不带 BOUNDS 的 ENCODING=BIT 没有一个标记 i,所以 LOC(i) 是不适合的,几个 LOC[i] 属性可以放到一个 X-BLOX 模块,每个为一个不同的 i,例如,LOC[1]=AA,LOC[-1]=AC 规定 BIT1 放在 CLB 位置 AA,BIT-1 放在 CLB 位置的 AC。

8. 用 BUS_IFxx 连接 X-BLOX 总线到非 X-BLOX 逻辑

有时候,可能要连接 X-BLOX 电路到非 X-BLOX 的电路,问题是,对比于 X-BLOX 总线,X-BLOX 总线可以修改 ENCODING 和 BOUNDS 属性重定尺寸,在非 X-BLOX 垫路的总线有一个规定的尺寸,不可以直接连接到通用的 X-BLOX 总线上,为此原因,必须利用一个由 X-BLOX 库提供的总线接口宏单元 BUS_IFxx 来连接 X-BLOX 总线和原理图编辑器的总线,这些原理图符号呈现如同无逻辑和方向控制的连线。

Xilinx 提供总线尺寸在 2—32 位宽的总线接口符号,命名为 BUS_IF02—BUS_IF#@ 因为这些宏单元不规定编码和边界,一个分开的 X-BLOX 的 BUS_DEF 符号必须接到 BUS_IFxx 符号的 X-BLOX 总线一边,在设计阶段,必须利用符号,并连接它们,以使能 X-BLOX 在综合过程中完成它的数据类型的传播。

要确信对接到 XBLOX_BUS 的总线已规定数据类型(编码和边界),或者通过从其它模块数据类型的传播,或者用一个 BUS_DEF 模块。

附录 B Xilinx FPGA 器件的出腿

Xilinx 生产超过45种不同类型的 FPGA 器件,多于30种不同的封装是有效的,几种不同的器件通常是按相同出腿的相同封装可用,用户可以从以下两种不同的方式由此得到好处:

1. 当一个设计不改变 I/O 要求增加特性作过修改时,一个有相同封装的更大的器件避免重新布图印刷电路板;

2. 当一个设计按较大的部件出的样机,此设计以后可以利用相同封装的较小器件来降低成本,一般,不要求修改印刷电路板。

在不同子系列中相同规模的器件总是100%兼容的,如:

XC2000, XC2000L

或 XC3000, XC3000A, XC3100, XC3100A, XC3000L

或 XC4000, XC4000A, XC4000D, (XC4000和 XC4000A 位流文件是不兼容的)

为保持印刷电路板的布局,任何设计装入到不同规模的器件要求相同的出腿,在大多数情况下, Xilinx 提供100%的出腿兼容,如表 B.1所示。

当不同的器件利用相同的封装,较大的器件通常有比封装腿更多的芯片接点(焊盘),所以一些 IOB 没有带出芯片,在这些器件(表 D.1中黑体字)之间任何方向移植是没有问题的,因为封装是共同受限,出腿是相同的;对较小的器件,封装有比器件的焊盘更多的出腿,以致一些封装出腿维持不接,移植到右边较大的器件没有问题,因为较大的器件总是接到较小器件所用的全部出腿,但移植到左边较小的器件有限制,仅当设计者预知这样的移植,并只利用在较小器件上被连接的出腿。

按上面提到的, XC4000的器件与所有其它 XC4000或 XC4000A 器件在相同封装时100%兼容; XC2000的器件与所有其它 XC2000或 XC2000L 器件在相同封装时100%兼容;在 XC3000中,有少数附加的例外:对 PC84,较小和较大器件间存在差别,需要两个附加的 GND 和 VCC 连接。移去 INIT,一些地址,数据和片选腿,对所有配置模式影响五条出腿,对主并行或外设模式加到七条腿。

在系列之间的出腿兼容受到限制。

XC2000和 XC3000有少些差别:两个系列晶振输出 XTL1的位置和并行主配置模式所用的地址和数据腿的位置不同, XC2000的外设模式是位串,而 XC3000的是字节并行。

XC3000和 XC4000有几个原因是不兼容的: XC4000没有 PWRDWN 输入和专门的 RESET 输入; XC4000没有在 XC3000中有来实现晶振的 XTL1和 XTL2出腿; XC4000把 XC3000的 DONE/PROG 腿的功能分成 DONE 和 PROGRAM 两条出腿; XC3000不支持边界扫描,仅有两条全局时钟网线输入,而 XC4000有八条;在 PC84中, XC4000器件有八个 GND 的连接,而 XC3000仅有2个或4个,把大多数专用功能移到不兼容的腿。

XC7300EPLD 的器件和 FPGA 的出腿不兼容。

表 B.1 Xilinx FPGA EPLD 的出腿兼容性

封装	每一行中的器件是出腿兼容的
PC44	XC7318 XC7336 XC2064 XC2018
PC68	XC2068 XC2018 XC3020 XC3030
PC84	XC7372 XC73108 XC3020 XC3030 XC3042 XC3064 XC3090 XC3195 XC4002 XC4003/3A XC4004A XC4005/5A XC4006 XC4008 XC4010/10D
PG84	XC3020 XC3030 XC3042
PQ100	XC3020 XC3030 XC4002 XC4003/3A
TQ100	XC3030 XC3042
VQ100	XC3030 XC3042 XC4002A XC4003AA
PG120	XC4002A XC4003/3A XC4004A
PG/PP132	XC3042 XC3064
TQ144	XC3042 XC3064 XC4004A XC4005A
PG156	XC4005/5A XC4006
PQ160	XC3064 XC3090 XC3195A XC3090 XC3195A XC4004A XC4005/5A XC4006 XC4008 XC4010/10D XC4013/13D
PG/PP175	XC3090 XC3195
PG191	XC4003H XC4008 XC4010
PG191	XC4003H XC4008 XC4010
PQ208	XC3090 XC3195 XC4003H XC4005/5A XC4006 XC4008 XC4010 XC4013 XC4020
PG223	XC4005H XC4013 XC4020 XC4025
BG225	XC4010/10D XC4013/13D
PQ/MQ240	XC4005H XC4013 XC4020 XC4025
PG299	XC4020 XC4025

参考资料

1. Xilinx: The Programmable Logic Data Book 1992, 1993, 1994
2. Altera: Data Book 1993
3. Actel: Actel's Family Fied Programmable Gate Array Data Book 1990
4. Cypress: Programmable Logic Data Book 1994/1995
5. Littice: Data Book 1994
6. AMD: MACH 1,2,3,4 Family Data Book 1994
7. AT&T, ORCA 1994
8. Xilinx: XCELL Issue 16,14,13,12,11
9. Special Section on Filed Programmable Gate Arrays
Proceedings of the IEEE Vol 81 No. 7 July 1993
10. Joseph Di Giacomo: VLSI Handbook 1989
11. Xilinx: User Guide
12. Xilinx: XEPLD User Guide
13. Xilinx: XBlox User Guide
14. Xilinx: XABEL User Guide
15. I. Moore, Will R. :FPGAs 1991
16. Xilinx: XC5000 Logic Cell Array Family 1995. 2
17. Xilinx: XC8100 FPGA Family 1994. 12
18. Electronic Design: 1993,1994,January 23,May 1,May 30,June 26,July 10, 1995

[G e n e r a l I n f o r m a t i o n]

书名=可编程专用集成电路原理设计和应用

作者= B E X P

页数= 2 8 0

下载位置= <http://book7.ssreader.com/diskjsj/04/1b52/!00001.pdg>