单元一　EDA 技术与可编程逻辑器件

进入 20 世纪以来，电子技术得到了飞速的发展，在其推动下，现代电子产品几乎渗透到了社会的各个领域，有力地推动了社会生产力的发展和社会信息化程度的提高，同时也使现代电子产品的性能进一步提高，更新换代的速度加快。特别是进入 20 世纪 90 年代以来，随着微电子和计算机技术的迅速发展，出现了 EDA 技术，它为电子系统的设计带来了革命性的变化，并已渗透到电子系统的各个领域。

一、EDA 技术

(一)EDA 技术的概念

EDA(electronic design automation，电子设计自动化)技术是微电子和计算机技术飞速发展的产物，它融多学科于一体，是一门综合性的学科。EDA 技术是以计算机硬件和系统软件为基本工作平台，继承和借鉴前人在电路和系统、数据库、图形学、图论和拓扑逻辑、计算数学、优化理论等多学科的最新科技成果而研制成的商品化通用支撑软件和应用软件包，其目的在于帮助电子设计工程师在计算机上完成电路的功能设计、逻辑设计、性能分析、时序测试直至 PCB(印制电路板)的自动设计等。

与早期的 CAD 软件相比，EDA 软件的自动化程度更高，功能更完善，运行速度更快，而且操作界面友好，有良好的数据开放性和互换性，即不同厂商的 EDA 软件可以相互兼容。因此，EDA 技术一经出现，就很快在世界各大公司、企业和科研单位得到了广泛的应用，并已成为衡量一个国家电子技术发展水平的重要标志。

EDA 技术的范畴应包括电子工程师进行电子系统设计的全过程中期望由计算机完成的各种辅助工作，包括计算机辅助设计(CAD)、计算机辅助制造(CAM)、计算机辅助测试(CAT)、计算机辅助工程(CAE)等。因此，利用 EDA 技术，电子工程师在计算机上即可完成电子系统设计的全过程。

(二)EDA 技术的基本特征

现代 EDA 技术的基本特征是采用高级语言描述，具有系统级仿真和综合能力。下面介

绍与这些基本特征有关的几个新概念。

1.“自顶向下”的设计方法

“自顶向下”的设计方法首先从系统级设计入手，在顶层对整机电路系统进行功能方框图的划分和结构设计；在方框图一级进行仿真、纠错，并用硬件描述语言对高层次的系统行为进行描述；在功能一级进行验证，然后用逻辑综合优化工具生成具体的门级逻辑电路网表，其对应的物理实现级可以是印制电路板或专用集成电路。

这种设计方法有利于在设计的早期发现结构设计中的错误，提高设计的一次成功率，因而在现代电子系统设计中被广泛采用。

2. 硬件描述语言

用硬件描述语言（hardware description language，HDL）进行电路系统的设计是当前EDA 技术的一个重要特征。与传统的原理图设计方法相比，硬件描述语言更适合规模日益增大的电子系统，它还是进行逻辑综合优化的重要工具。

硬件描述语言能使设计者在比较抽象的层次上描述设计的结构和内部特征。它的突出优点是语言的公开性、设计与工艺的无关性、宽范围的描述能力、便于组织大规模系统的设计、便于设计的复用和继承等。

3. 逻辑综合和优化

逻辑综合功能能将高层次的系统行为设计自动翻译成门级逻辑的电路描述，做到了设计与工艺的独立。优化则是根据布尔方程等效的原则，对上述综合生成的电路网表进行化简，用更小更快的综合结果替代一些复杂的逻辑电路单元，根据指定的目标库映射成新的网表。

4. 开放性和标准化

框架是一种软件平台结构，它提供了与硬件平台无关的图形用户界面及工具之间的通信、设计数据和设计流程的管理等。对于任何一个 EDA 系统来讲，只要建立了一个符合标准的开放式框架结构，就可以接纳其他厂商的 EDA 工具一起进行设计，这样可以实现不同厂商 EDA 工具间的优化组合，并且可以把它们集成在一个易于管理的同一环境下，实现资源共享。

近年来，随着硬件描述语言等设计数据格式的逐步标准化，不同设计风格和应用的要求导致各具特色的 EDA 工具被集成在同一工作站上，从而使 EDA 框架标准化。新的 EDA 系统不仅能够实现高层次的自动逻辑综合、版图综合和测试码生成，而且可以使各个仿真器对同一设计进行协同仿真，进一步提高了 EDA 系统的工作效率和设计的正确性。

5. 库的引入

EDA 工具之所以能够完成各种自动设计过程，关键是有各类库的支持，如逻辑模拟时的模拟库、逻辑综合时的综合库、版图综合时的版图库、测试综合时的测试库等，这些库都是EDA 设计公司与半导体生产厂商紧密合作、共同开发的。

二、可编程逻辑器件

可编程逻辑器件(programmable logic device,PLD)是20世纪70年代发展起来的一种新型逻辑器件,最初是用来解决数字系统的存储问题,后来转为各种逻辑应用,现在已成为实现数字系统的重要手段。这种器件在出厂时内部已集成了各种硬件资源,如逻辑单元、互联线等,但其内部的连线不一定需要制造厂完成,用户可以借助强大的EDA软件与编程器,自己在实验室等一般场所改变PLD的内部连线,以实现自己所期望的数字系统。

PLD的出现是现代数字系统向着超高集成度、超低功耗、超小封装和专用化方向发展的重要基础,它的应用和发展不仅简化了电路的设计,降低了成本,提高了系统的可靠性和保密性,而且给数字系统的设计带来了革命性的变化。

(一)可编程逻辑器件的分类

利用可编程逻辑器件,用户通过编程即可实现所需的功能。按照结构复杂程度的不同,可将PLD大致分为简单可编程逻辑器件、复杂可编程逻辑器件和现场可编程门阵列。

1. 简单可编程逻辑器件

简单可编程逻辑器件主要指早期开发的PLD,它通常由与阵列和或阵列组成,能够用来实现任何以“积之和”形式表示的各种布尔逻辑函数。当与和或两个阵列都可编程时,这个器件就称为PLA,其变形是PROM、PAL和GAL,前者具有固定的与阵列和可编程的或阵列,后两者具有可编程的与阵列和固定的或阵列。

拓展

常用名词缩写

PAL和GAL是早期得到广泛应用的可编程逻辑器件,通常一片PAL或GAL可用来代替4～10片中、小规模集成电路。从编程工艺上看,PAL器件一般用熔丝链路作为编程开关,是一次性可编程的;GAL器件则可反复编程,它采用了E^2CMOS工艺,实现了电可擦除与电可改写,为设计和修改提供了极大的方便。

2. 复杂可编程逻辑器件

复杂可编程逻辑器件(complex programmable logic device,CPLD)是20世纪80年代后期得到迅速发展的新一代可编程逻辑器件。早期的PLD结构简单,具有成本低、速度快、设计简便等优点,但其规模较小,通常只有几百个等效逻辑门,难以实现复杂的逻辑。为了增加PLD的密度,扩充其功能,一些厂家对PLD的结构进行了改进。例如,在两个逻辑阵列的基础上大量增加输出宏单元、提供更大的与阵列及采用分层结构逻辑阵列等,使PLD逐渐向复杂可编程逻辑器件过渡。

进入20世纪90年代后,CPLD已经成为可编程逻辑器件的主流产品,在整个PLD市场上占有较大的份额。它们一般都具有可重复编程的特性,能方便预测设计的时序,具有高速性能。CPLD的集成度一般可达数千甚至数万门,能够实现较大规模的电路集成。

3. 现场可编程门阵列

现场可编程门阵列(field programmable gate array,FPGA)是与传统PLD不同的一类可编程逻辑器件。它采用了类似门阵列的通用结构,即由逻辑功能块排列成阵列组成,并由可编程的互联资源连接这些功能块来实现所需的设计。FPGA可由用户现场编程来完成逻

辑功能块之间的连接，因此，它是一种将门阵列的通用结构和 PLD 的现场可编程特性结合于一体的新型器件，具有集成度高、通用性好、设计灵活、编程方便、产品上市快捷等多方面的优点。

与传统的 PLD 相比，FPGA 由于采用了类似门阵列的通用结构，规模可以做得较大，可实现的功能更强，设计的灵活性也更大。在问世的前 10 年里，其单片可用门数以年平均 42%的速度增长，目前已突破 4 000 万门。FPGA 中包含丰富的触发器资源，有些还具有诸如片上 RAM、内部总线等许多系统级的功能，因而完全可以实现片上系统的集成。就互联结构而言，典型的 FPGA 通常采用分段互联式结构，具有走线灵活、便于复杂功能的多级实现等优点，但与此同时也带来了布线复杂度增加、输入至输出的延时变大及总的性能估计较困难等问题。随着用户对 FPGA 性能要求的多样化，出现了各种改进结构的 FPGA，如 Altera公司 20 世纪 90 年代推出的 FLEX8000 系列产品，这些产品将 CPLD 中的互联结构引入 FPGA 中，采用功能块内局部互联和功能块间 fast track（快速通道）互联相结合的方法，较好地解决了高集成度下芯片的走线规则性和延时可预测性问题。

目前，FPGA 的生产厂商已由最初的一家增加到十多家，产品种类日益丰富，性能不断完善，成为最受欢迎的器件之一。

（二）可编程逻辑器件的主要特点

前面介绍的三种 PLD，尽管其结构和性能不尽相同，但共同点是都由用户通过编程来决定芯片的最终功能。随着微电子工艺和技术的进步，它们在现代电子系统中所占的地位也越来越重要。与传统的集成电路相比，采用 PLD 进行电子系统的设计具有以下特点。

1. 缩短研制周期

相对于用户而言，PLD 可像通用器件一样按一定的规格型号在市场上购买，其功能的实现完全独立于 IC（集成电路）工厂，由用户在实验室或办公室即可完成，因此不必像传统 IC 那样花费样片制作等待时间。由于采用先进的 EDA 技术，PLD 的设计和编程均十分方便与有效，整个设计通常只需几天便可完成，缩短了产品研制周期，有利于产品的快速上市。

2. 降低设计成本

制作传统 IC 的前期投资费用较高，动辄数万元，只有在生产批量很大的情况下才有价值。这种设计方法还需要承担很大的风险，因为一旦设计中存在错误或设计不完善，则全套样片便不能使用，巨额的设计费用将付之东流。采用 PLD 为降低投资风险提供了合理的选择途径，它不需要样片制作费用，在设计的初期或小批量的试制阶段，其平均单片成本很低。如果要转入大批量生产，由于已用 PLD 进行了原型验证，也比直接设计 IC 费用少、成功率高。

3. 提高设计灵活性

PLD 由用户编程实现芯片的功能，与传统的 IC 相比，具有更好的设计灵活性。首先，PLD 在设计完成后可立即进行验证，有利于及早发现设计中的问题，完善设计；其次，大多数 PLD 可反复多次编程，为设计修改和产品升级带来了方便；最后，基于 SRAM 开关的 FPGA 具有动态重构特性，在系统设计中引入了“软硬件”的全新概念，使得电子系统具有更好的灵

活性和自适应性。

(三)可编程逻辑器件的基本结构

PLD的种类较多,不同厂家的PLD结构差别较大,这里只选择一些具有代表性的结构来说明。

图1-1所示为PLD的基本结构框图,它由输入缓冲电路、与阵列、或阵列、输出缓冲电路四部分组成。其中,与阵列和或阵列是PLD的主体,可以实现任何以“积之和”形式表示的逻辑函数;输入缓冲电路主要是对输入信号进行预处理;输出缓冲电路可提供所需的寄存器或触发器,并可根据需要选择各种灵活的输出方式。

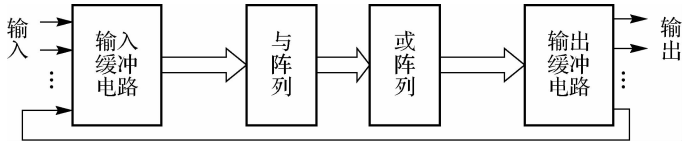
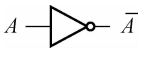
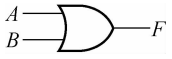
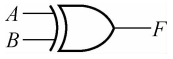
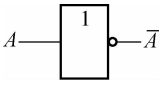
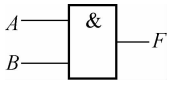
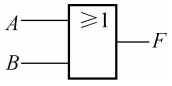
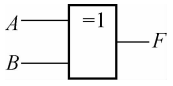


图1-1 PLD的基本结构框图

1. 电路符号表示

在常用的EDA软件中,原理图一般是用器件符号来表示的。由于PLD阵列规模一般远大于普通电路,用传统的器件符号已不能满足PLD原理图的需要,因此在PLD中,用一种约定的符号来简化表示原理图。常用的逻辑门符号见表1-1。

表1-1 常用的逻辑门符号

表达形式	非　门	与　门	或　门	异 或 门
常用符号				
国标符号				
逻辑表达式	$F=\overline{A}$	$F=A\cdot B$	$F=A+B$	$F=A\oplus B$

为了使输入信号具有足够的驱动能力并产生原码和反码两个互补的信号,PLD的输入缓冲器和反馈缓冲器都采用互补的输出结构,如图1-2所示。图中,$B=A$,$C=\overline{A}$。

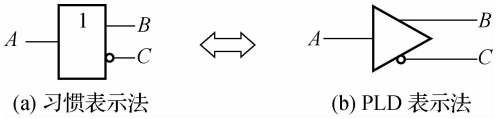


图1-2 PLD的输入缓冲器电路

图 1-3 所示是 PLD 中与阵列的简化图形，表示可以选择 A、B、C 和 D 四个信号中的任何一个或全部作为与门的输入。图 1-4 所示是 PLD 中或阵列的简化图形。

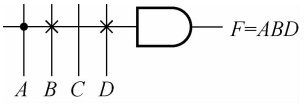


图 1-3　PLD 中与阵列的简化图形

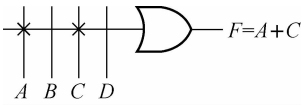


图 1-4　PLD 中或阵列的简化图形

图 1-5 所示是在阵列中连接关系的表示。十字交叉线表示此二线未连接；交叉线的交点上打黑点，表示固定连接，即在 PLD 出厂时已连接；交叉线的交点上打叉，表示该点可编程，即在 PLD 出厂后通过编程可随时改变其连接。

图 1-5　在阵列中连接关系的表示

2. 与-或阵列

与-或阵列是 PLD 中最基本的结构，通过改变与阵列和或阵列的内部连接，就可以实现不同的逻辑功能。依据可编程的部位不同，可将简单可编程逻辑器件分为可编程只读存储器（PROM）、可编程逻辑阵列（PLA）、可编程阵列逻辑（PAL）、通用阵列逻辑（GAL）四种最基本的类型，见表 1-2。

表 1-2　四种简单可编程逻辑器件的比较

器　件　名	与　阵　列	或　阵　列	输出电路
PROM	固定	可编程	固定
PLA	可编程	可编程	固定
PAL	可编程	固定	固定
GAL	可编程	固定	可组态

PROM 中包含一个固定连接的与阵列和一个可编程连接的或阵列，其结构示意图如图 1-6 所示。图中的 PROM 有 4 个输入端、16 个乘积项和 4 个输出端。

PLA 中包含一个可编程连接的与阵列和一个可编程连接的或阵列，其结构示意图如图 1-7 所示。

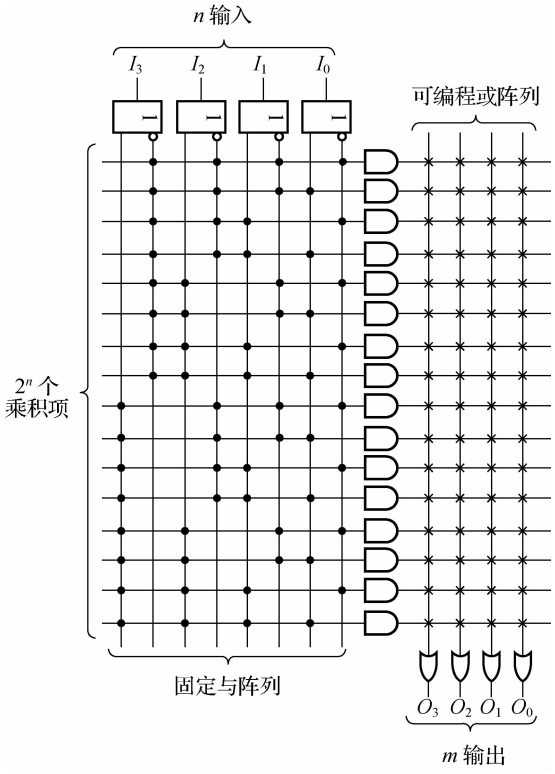


图 1-6　PROM 的结构示意图

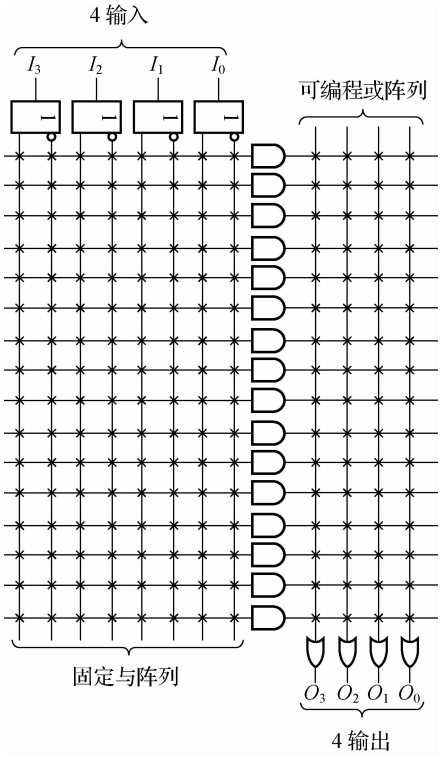


图 1-7　PLA 的结构示意图

PAL 和 GAL 的基本门阵列结构是相同的，即与阵列可编程，或阵列固定连接。两者之间的差异除了表现在输出结构上，还表现在 PAL 只能编程一次，而 GAL 器件则可多次编程，正是因为这一点使得 GAL 器件更受到用户的欢迎。

3. 宏单元

与或阵列在 PLD 中只能实现组合电路的功能，而时序电路的功能则由包含触发器或寄存器的逻辑宏单元来实现，宏单元也是 PLD 中的一个重要的基本结构。

尽管不同厂商在 PLD 产品的宏单元设计上有着各自的特点，但总体来说，逻辑宏单元具有以下几个作用。

(1)提供时序电路所需的寄存器和触发器。

(2)提供多种形式的输入/输出方式。

(3)提供内部信号反馈，控制输出逻辑极性。

(4)分配控制信号，如寄存器的时钟和复位信号、三态门的输出使能信号等。

上面介绍了简单 PLD 的结构和可编程的基本原理。由于这些简单 PLD 具有可编程资源较少、编程不便等缺点，在现代电子系统设计中已基本被淘汰，只有 GAL 还在某些场合被使用。现在的可编程逻辑器件以大规模、超大规模集成电路工艺制造的 CPLD、FPGA 为主。

拓展

Xilinx 公司简介

(四)Altera 公司的可编程逻辑器件

Altera 公司是 20 世纪 90 年代以来发展较快的 PLD 生产厂商。在激烈的市场竞争中，Altera 公司凭借其雄厚的技术实力、独特的设计构思和功能齐全的芯片系列，成为世界最大的可编程逻辑器件供应商之一。有资料显示，目前该公司的 PLD 产品与 Xilinx 公司的 PLD 产品占 PLD 市场的 60%左右。

1. Altera 公司的产品简介

1)PLD 产品的结构特点

Altera 产品的基本构造块是逻辑单元。在 Classic、MAX3000A、MAX5000、MAX7000、MAX9000 系列中，逻辑单元又称为宏单元，它由可编程的与阵列和固定的或阵列构成；在 FLEX8000、FLEX6000、FLEX10K、APEX20K、ACEX1K 等系列中，逻辑单元采用查找表(look-up table，LUT)结构构成。所谓的查找表结构，就是在 RAM 中预先存入所要实现函数的真值表数值，然后以输入变量作为地址，从 RAM 中选择相应的数值作为逻辑函数的输出值，这样就可以实现输入变量的所有可能的逻辑函数。

不同结构的 PLD 侧重的应用场合不同，如 MAX 系列的宏单元分解组合逻辑的功能很强，一个宏单元可以分解数十个组合逻辑输入，因此 MAX 系列的产品较适合设计组合逻辑电路；而 FLEX 系列的制造工艺允许它拥有较多的查找表和触发器，从逻辑单元的数量来看，后者远高于前者，因此采用查找表结构的产品更适合用来设计需要用到大量触发器的复杂时序逻辑电路。

由于 PLD 一般具有可重复编程的特点，所以其内部必须采用一定的工艺来实现这种功能。FLEX、APEX、ACEX 等系列一般采用 SRAM(静态随机存储器)工艺；MAX 系列则一

般采用 E^2PROM(电可擦除可编程存储器)工艺;而早期的 Classic、MAX5000 系列则大多采用 EPROM(可擦除可编程存储器)工艺。

2)Altera 的开发工具

Altera 的开发工具已经经历了四代,从最初的基于 PC DOS 的 A+plus,发展到MAX+plus,又于 1991 年推出性能更加完善的基于 Windows 的开发工具 MAX+plus Ⅱ,之后,Altera又推出了它的第四代开发工具 Quartus。随着器件性能的不断提高和集成度的不断扩大,Altera 始终能够同步推出与之相适应的开发工具。

Quartus Ⅱ是一个功能强大、易学易用的 EDA 开发工具,它提供了一种与结构无关的设计环境。设计者不需要精通器件内部的复杂结构,而只需要用自己熟悉的设计输入工具(如原理图或高级语言)把自己的设计输入计算机中,Quartus Ⅱ就能够自动把这些设计转换成最终所需的数据格式,用户只要把最后的数据通过下载电缆下载到芯片中,就能完成所有设计工作。

2. CPLD 的基本结构

下面以 MAX7000 系列产品为例介绍 CPLD 的基本结构。

MAX7000 属于高性能、高密度的 CPLD,其制造工艺采用了先进的 CMOS E^2PROM 技术,在结构上包括逻辑阵列块(logic array blocks,LAB)、宏单元(macrocells)、扩展乘积项(expender product terms)、可编程连线阵列(programmable interconnect array,PIA)和 I/O 控制块(I/O control blocks)。

1)逻辑阵列块

MAX7000 系列产品主要由逻辑阵列块(LAB)及它们之间的连线构成,如图 1-8 所示。每个 LAB 由 16 个宏单元组成,多个 LAB 通过可编程连线阵列(PIA)和全局总线连接在一起,全局总线从所有的专用输入、I/O 引脚和宏单元馈入信号。

每个 LAB 的输入信号包括来自作为通用逻辑输入的 PIA 的 36 个信号、全局控制信号和从 I/O 引脚到寄存器的直接输入信号。

2)宏单元

MAX7000 的宏单元由三个功能块组成,即逻辑阵列、乘积项选择矩阵和可编程触发器,它们可以被单独地配置为组合逻辑和时序逻辑工作方式。宏单元的结构框图如图 1-9 所示。

图中的逻辑阵列实现组合逻辑功能,它可以给每个宏单元提供 5 个乘积项。乘积项选择矩阵分配这些乘积项作为到或门和异或门的主要逻辑输入,以实现组合逻辑函数;或者把这些乘积项作为宏单元中寄存器的辅助输入(清零、置位、时钟、时钟使能控制等)。

每个宏单元中有一个“共享逻辑扩展项”经非门后回馈到逻辑阵列中。另外,宏单元中的可配置寄存器可以单独地被配置为带有可编程时钟控制的 D、T、JK 或 RS 触发器工作方式,也可以将寄存器旁路去掉,以实现组合逻辑工作方式。

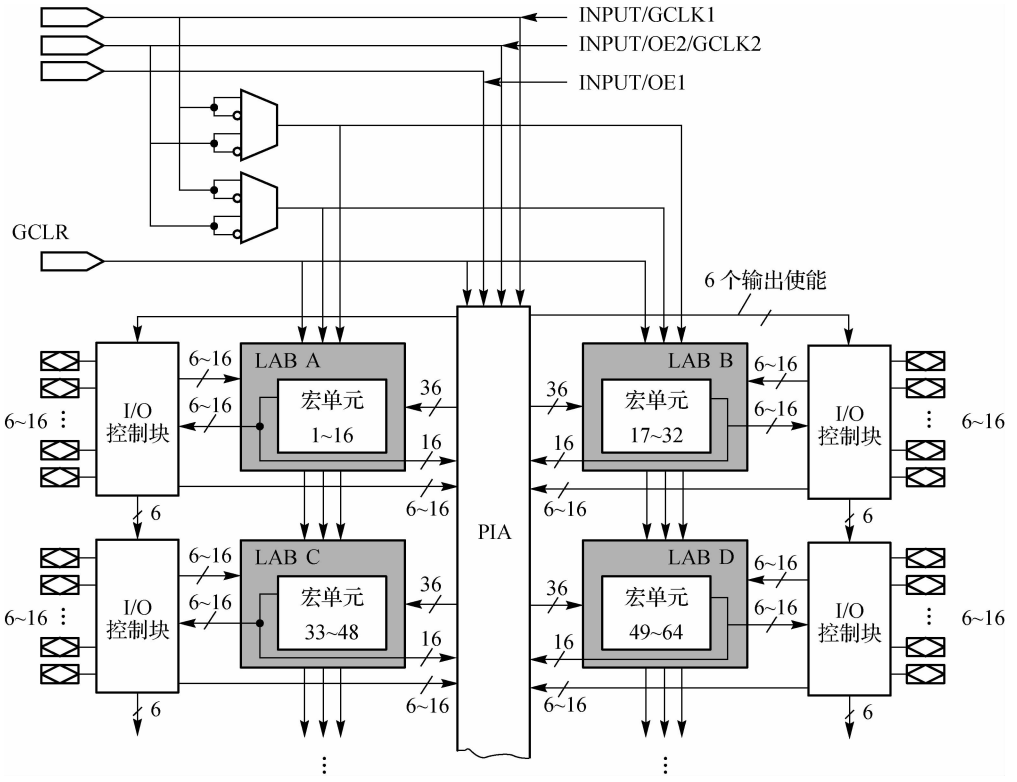


图 1-8　MAX7000E 和 MAX7000S 的结构图

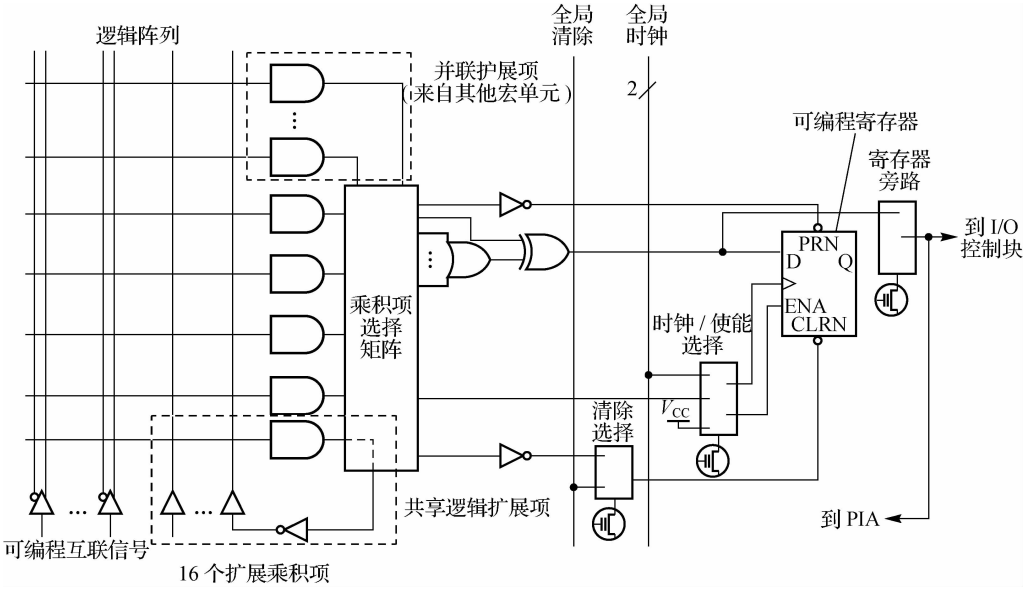


图 1-9　宏单元的结构框图

3)扩展乘积项

虽然大多数逻辑函数能够用每个宏单元中的5个乘积项实现,但某些逻辑函数比较复杂,要实现它们需要附加乘积项。这时,可以利用其他宏单元以提供所需的逻辑资源。对于MAX7000系列产品,还可以利用其结构中具有的共享和并联扩展乘积项,这两种扩展项作为附加的乘积项直接送到该LAB的任意一个宏单元中。利用扩展项可保证在实现逻辑综合时,用尽可能少的逻辑资源,得到尽可能快的工作速度。

4)可编程连线阵列

可编程连线阵列(PIA)用于将各个LAB相互联接,构成所需的逻辑布线通道。这个全局总线是一种可编程的通道,可以把器件中任何信号连接到其目的地。所有MAX7000器件的专用输入、I/O引脚和宏单元输出都连接到PIA,而PIA可把这些信号送到整个器件内的各个地方。

5)I/O控制块

I/O控制块允许每个I/O引脚单独地配置为输入、输出和双向三种工作方式。所有I/O引脚都有一个三态缓冲器,它能由全局输出使能信号中的一个控制,或者把使能端直接连接到地或电源上。I/O控制块有两个全局输出使能信号,它们由两个专用的、低电平有效的输出使能引脚OE1和OE2来驱动。图1-10所示为I/O控制块的结构图。

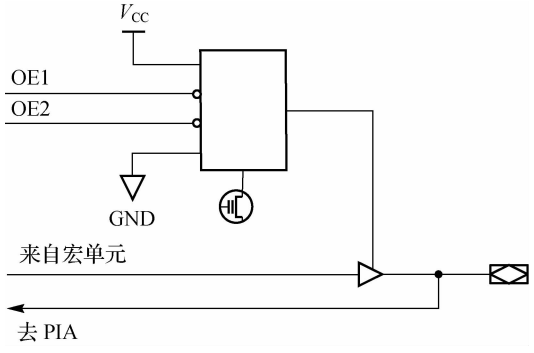


图1-10 I/O控制块的结构图

为降低CPLD的功耗,减少其工作时的发热量,MAX7000系列产品还提供可编程的速度或功率优化,使得在应用设计中,让影响速度的关键部分工作在高速或全功率状态,而其余部分则工作在低速或低功率状态。允许用户配置一个或多个宏单元工作在50%或更低的功率下,而仅增加一个微小的延时。对于I/O工作电压,MAX7000器件有多种不同特性的系列,其中,E、S系列为5.0 V工作电压;A和AE系列为3.3 V混合工作电压;B系列为2.5 V混合工作电压。

3. FPGA的基本结构

下面以FLEX10K系列产品为例介绍FPGA的基本结构。

FLEX(flexible logic element matrix,灵活逻辑单元矩阵)系列是Altera公司推出的主流FPGA产品,它具有高密度、可在线配置、高速度与连续式布线结构等特点。

FLEX10K系列是Altera于1998年推出的,它的集成度达到了10万门级。它还是业界

首次集成了嵌入式阵列块(EAB)的芯片。所谓 EAB,实际上是一种大规模的 SRAM 资源,它可以被方便地设置为 RAM、ROM、FIFO 及双口 RAM 等存储器。EAB 的出现极大地拓展了 PLD 芯片的应用领域。

FLEX10K 在结构上包括嵌入式阵列块(EAB)、逻辑阵列块(LAB)、快速通道(fast track)互联和 I/O 单元,其基本结构如图 1-11 所示。图中,由一组逻辑单元 LE 组成一个 LAB,LAB 按行和列排成一个矩阵,并且在每一行中放置了一个 EAB。在器件内部,信号的互联及信号与器件引脚的连接由快速通道提供,在每行或每列快速通道互联线的两端连接着若干个 IOE。

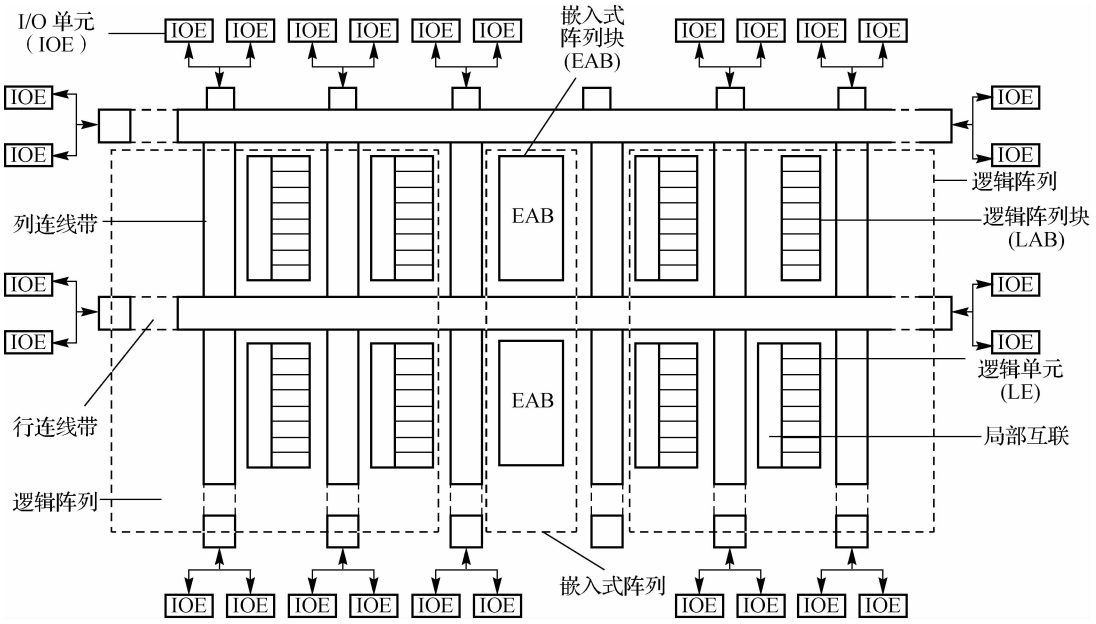


图 1-11　FLEX10K 的基本结构

1)嵌入式阵列块

EAB 是一种输入端和输出端带有寄存器的 RAM,它既可以作为存储器使用,也可以用来实现逻辑功能。

当作为存储器使用时,每个 EAB 可提供 2 048 个比特位,用来构成 RAM、ROM、FIFO RAM 或双口 RAM。每个 EAB 单独使用时,可配置成以下几种尺寸:256×8,512×4,1 024×2 和 2 048×1。多个 EAB 可组合成一个规模更大的 RAM 或 ROM 使用。例如,两个 256×8 的 RAM 可组合成一个 256×16 的 RAM;两个 512×4 的 RAM 可组合成一个 512×8 的 RAM。

EAB 的另一个应用是用来实现复杂的逻辑功能。每个 EAB 可相当于 100～300 个等效门,能方便地构成乘法器、加法器、纠错电路等模块,并由这些模块进一步构成诸如数字滤波器、微控制器等系统。具体实现时,逻辑功能是通过配置时编程 EAB 为只读模型,生成一个大的 LUT 来实现的。在 LUT 中,逻辑功能是通过查表的方式实现的,因而其速度较用常规逻辑运算实现时更快。加上 EAB 的大容量,使得设计者能够在一个逻辑级上完成复杂

的功能，避免了多个 LE 连接带来的连线延时。因此，与用 LE 实现组合逻辑功能相比，EAB 不但大大提高了器件的效率与性能，也极大地减小了器件的占用面积。

2）逻辑单元

LE 是 FLEX10K 结构中的最小单元，它能有效地实现逻辑功能。每个 LE 包含一个 4 输入的 LUT、1 个带有同步使能的可编程触发器、1 个进位链和 1 个级联链。每个 LE 有两个输出分别可以驱动局部互联和快速通道互联。LE 的结构框图如图 1-12 所示。

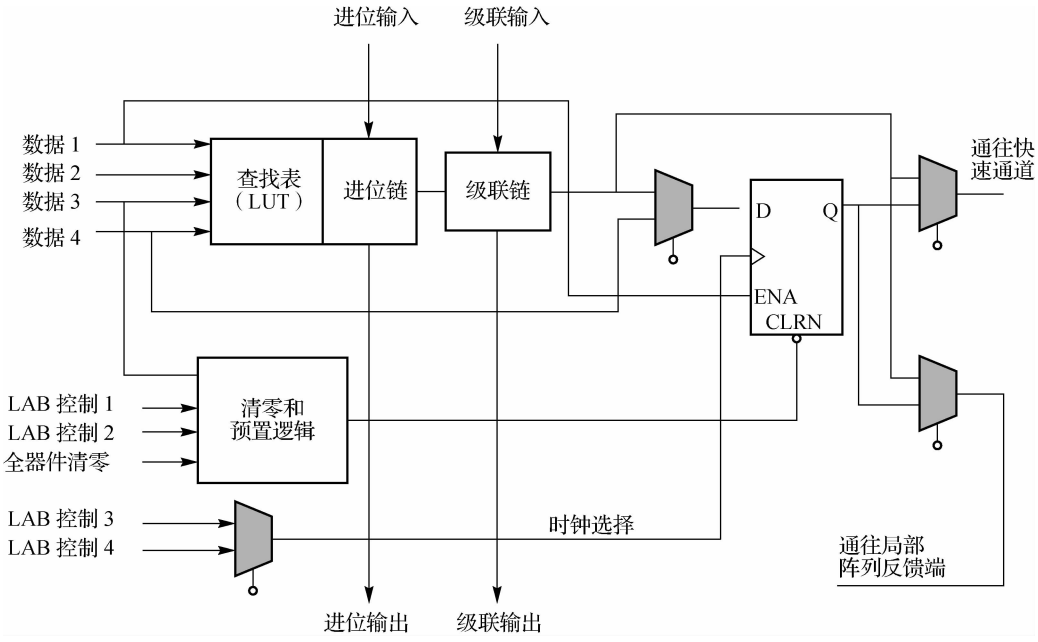


图 1-12　LE 的结构框图

图中的 LUT 是一种函数发生器，它能实现 4 输入 1 输出的任意逻辑函数。LE 中的可编程触发器可设置成 D、T、JK 或 RS 触发器。该触发器的时钟、清零和置位信号可由全局信号通用 I/O 引脚或任何内部逻辑驱动。当用于实现组合逻辑时，可将该触发器旁路，LUT 的输出作为 LE 的输出。

LE 有两个输出驱动内部互联，一个驱动局部互联，另一个驱动行或列的快速通道互联输出。这两个输出可以单独控制，可以实现在同一 LE 中，LUT 驱动一个输出，而寄存器驱动另一个输出。因而在一个 LE 中的触发器和 LUT 能够用来完成不相关的功能，能够提高 LE 的资源利用率。

在 FLEX10K 结构中还提供了两种专用高速数据通道，用于连接相邻的 LE，但不占用局部互联通路，它们是进位链和级联链。进位链用来支持高速计数器和加法器；级联链可以实现多输入逻辑函数，而且延时很小。级联链和进位链可以连接同一个 LAB 中的所有 LE 和同一行中的所有 LAB，但是级联链和进位链的大量使用会限制逻辑布线的灵活性，导致资源的浪费。因此，建议只在设计中要求速度的部分才使用它们。

3)逻辑阵列块

LAB 是由一系列相邻的 LE 构成的。每个 LAB 包含 8 个 LE、相连的进位链和级联链、LAB 控制信号和 LAB 局部互联线。LAB 构成了 FLEX10K 的主体部分。

每个 LAB 提供 4 个可供 8 个 LE 使用的控制信号，其中 2 个可用作时钟，另外 2 个用作清除/置位逻辑控制。LAB 的控制信号可由专用输入引脚、I/O 引脚或借助 LAB 局部互联的任何内部信号直接驱动，专用输入端一般用作公共的时钟、清除或置位信号。

FLEX10K 的 LAB 结构框图如图 1-13 所示。

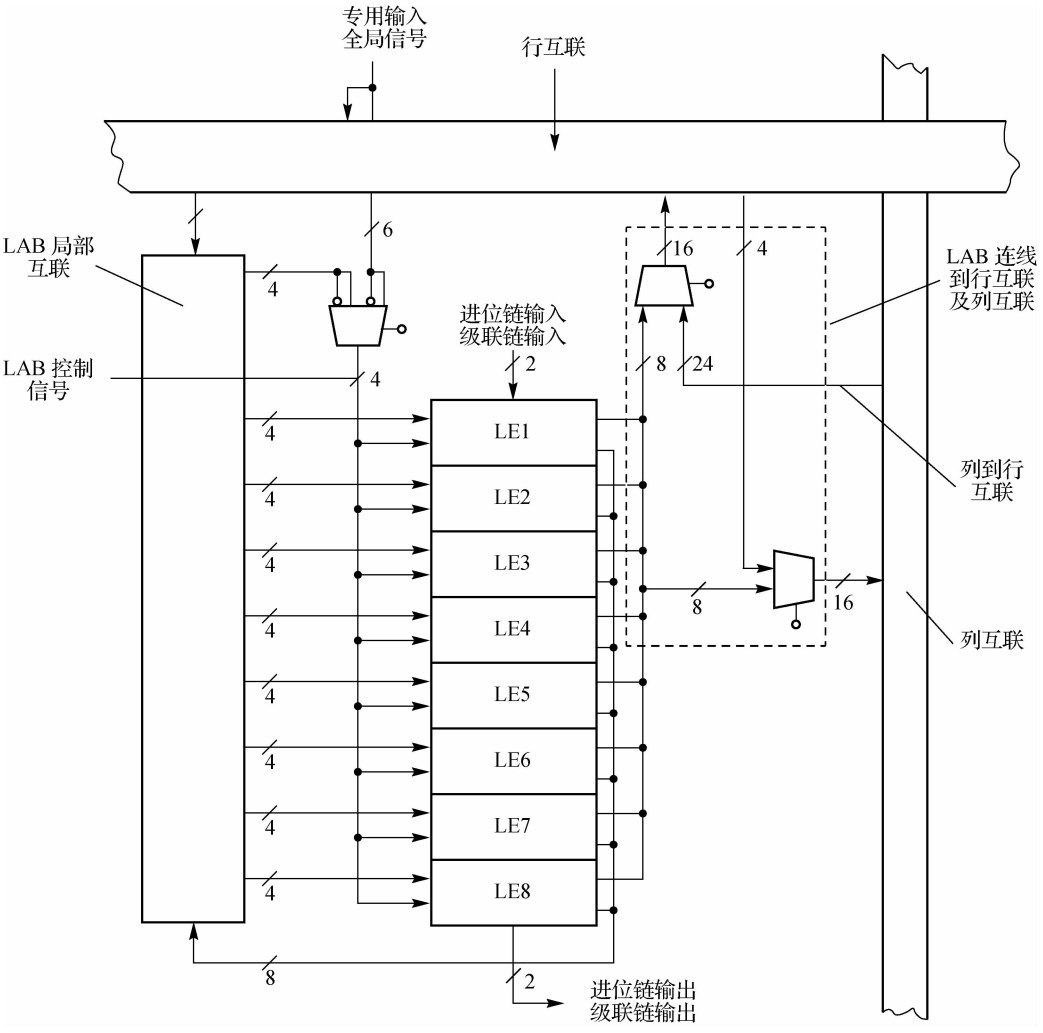


图 1-13　FLEX10K 的 LAB 结构框图

4)快速通道

在 FLEX10K 中，不同 LAB 中的 LE 与器件 I/O 引脚之间的连接是通过快速通道互联实现的。快速通道是贯穿整个器件长和宽的一系列水平和垂直的连续式布线通道，由若干组行连线和列连线组成。每一组行连线视器件大小的不同可以有 144 根、216 根或 312 根，

每一组列连线均为24根。

快速通道由行连线带和列连线带组成。采用这种布线结构，即使对于复杂的设计也可预测其性能。相反，采用其他连线结构（如分段式连线结构）会增加逻辑资源之间的延时，从而使性能下降。

5）I/O单元

FLEX10K器件的I/O引脚是由一些I/O单元（IOE）驱动的。IOE位于快速通道和行与列的末端，包含一个双向I/O缓冲器和一个寄存器。这个寄存器可以用作需要快速建立时间的外部数据的输入寄存器，也可作为要求快速“时钟到输出”性能的输出寄存器。IOE可以被配置成输入、输出或双向口。

FLEX10K的IOE具有许多有用的特性，如JTAG编程支持、摆率控制、三态缓冲和漏极开路输出等。FLEX10K还提供了6个专用输入引脚，这些引脚用来驱动IOE存储器的控制端，使用了专用的布线通道，以便具有比快速通道更短的延时和更小的偏移。专用输入中的4个输入引脚可用来驱动全局信号，内部逻辑也可以驱动这4个全局信号。另外，每个IOE中输出缓冲器输出信号的电压摆率可调，可通过配置达到低噪声或高速度的要求。电压摆率的加快能使速度提高，但这会在器件工作时引入较大的噪声。

4. FPGA的数据配置

由于FPGA使用的是SRAM工艺的器件，其内部逻辑功能和连线由芯片内SRAM所存储的数据决定，而SRAM是数据掉电易失性存储器。因此，在利用FPGA构成独立的数字系统时，必须外接掉电非易失性存储器（如E^2PROM）存储配置数据。系统加电时，通过存储在芯片外部的串行E^2PROM所提供的数据对FPGA进行配置。

Altera提供了专门的配置器件来存储基于SRAM工艺的FPGA器件的配置数据，提供的配置器件支持ISP和多重端口电压标准。例如，EPC1441和EPC1是专门供FLEX10K器件配置用的E^2PROM，它们借助串行数据流配置FLEX10K。配置完成后，还可以通过复位进行在线重配置，装入新数据，实现新功能。由于重新配置所需的时间少于100 ms，所以在系统工作过程中可以实时地改变配置。

在数字系统的调试阶段，为了避免多次对配置器件的擦写而引起器件的损坏，也可以通过Altera的Bit Blaster串行下载电缆直接将配置数据下载到FPGA中的SRAM。待系统调试成功后，再将配置数据写入配置器件中。

表1-3给出了Altera公司部分用于配置SRAM工艺的E^2PROM。

表1-3　Altera公司部分用于配置SRAM工艺的E^2PROM

E^2PROM型号	容　量	适用器件型号	电　压	常用封装形式
EPC1441（不可擦写）	441 KB	所有FLEX系列器件	3.3/5 V自动选择	8脚DIP
EPC1（不可擦写）	1 MB	所有APEX、FLEX系列器件	3.3/5 V自动选择	8脚DIP

续表

E^2PROM 型号	容　量	适用器件型号	电　压	常用封装形式
EPC2(可重复擦写)	2 MB	所有 APEX、FLEX 系列器件	3.3/5 V 引脚控制	20 脚 PLCC
EPC8(可重复擦写)	8 MB			100 脚 PQFP
EPC16(可重复擦写)	16 MB			88 脚 BGA

(五)可编程逻辑器件的发展趋势

可编程逻辑器件尽管起步较晚,但由于其具有可编程性和设计方便性,所以其发展的速度很快。特别是大规模 PLD 已经成为当今世界上最富吸引力的半导体器件,在现代电子系统设计中扮演着越来越重要的角色,其未来的发展将呈现以下几个方面的趋势。

1. 向高密度、大规模的方向发展

电子系统的发展必须以电子器件为基础,但并不与之同步,往往系统的设计需求更快。因而随着电子系统复杂度的提高,PLD 的规模不断地扩大,从最初的几百门到现在的上百万门。目前,高密度的 PLD 产品已经成为主流器件,PLD 已具备了片上系统(system on chip,SoC)集成的能力,产品性能发生了巨大的飞跃,这也促使着工艺的不断进步,而每次工艺的改进,PLD 的规模都将有很大的扩展。由于看好高密度 PLD 的市场前景,世界各大公司纷纷推出自己功能强大的 CPLD 和 FPGA 产品。这些高密度、大容量的 PLD 的出现,给现代电子系统(复杂系统)的设计与实现带来了巨大的帮助。

2. 向系统内可重构的方向发展

系统内可重构是指 PLD 在置入用户系统后仍具有改变其内部功能的能力。采用系统内可重构技术,使得系统内硬件的功能可以像软件那样通过编程来配置,从而在电子系统中引入“软硬件”的全新概念。它不仅使电子系统的设计和产品性能的改进与扩充变得十分简便,还使新一代电子系统具有极强的灵活性和适应性,为许多复杂信号的处理和信息加工的实现提供了新的思路与方法。

按照实现的途径不同,系统内重构可分为静态重构和动态重构两类。对于基于 E^2PROM 或快速擦写技术的 PLD,系统内重构是通过在系统编程技术实现的,是一种静态逻辑重构。ISP 可编程逻辑器件的工作电压和编程电压是相同的,编程数据可通过一根编程电缆从 PC 或工作站写入芯片,设计者无须把芯片从电路板上取下就能完成芯片功能的重构,给设计修改、系统调试及安装带来了极大的方便。另一类系统重构即动态重构,是指在系统运行期间,根据需要适时地对芯片重新配置以改变系统的功能,可由基于 SRAM 技术的 FPGA 实现。这类器件可以无限次地被重新编程,利用它可以 1 s 几次或者 1 s 数百次地改变器件执行的功能,甚至可以只对器件的部分区域进行重构,此时芯片的其他部分仍可正常工作。PLD 的系统内可重构特性有着极其广泛的应用前景,近年来在通信、航天、计算机硬件系统、程序控制、数字系统的测试诊断等方面获得了较好的应用。

3. 向低电压、低功耗的方向发展

集成技术的飞速发展，工艺水平的不断提高，节能潮流在世界的兴起，也为半导体工业提出了降低工作电压的发展方向。在全球环保声日益强烈和国际环保标准 ISO 14000 的推动下，半导体制造商纷纷研发能够节能的绿色元件。Philips 的 XPLA1 系列 CPLD 就是一个代表。该绿色 CPLD 产品家族由 22V10、32MC、64MC 和 128MC 等型号产品组成，是在 Philips 第二代 CPLD 基础上发展起来的，其功耗是一般 CPLD 产品的 1/1 000。

4. 向高速可预测延时器件的方向发展

PLD 产品能够得以广泛应用，与其灵活的可编程性分不开；同时，时间特性也是一个重要的原因。作为延时可预测的器件，PLD 的速度在系统中的作用巨大。当前的系统中，由于数据处理量的激增，要求数字系统有大的数据吞吐量，加之多媒体技术的迅速发展，更多的是图像的处理，相应地要有高速的硬件系统，而高速的系统时钟是必不可少的。PLD 若要在高速系统中占有一席之地，也必然向高速发展。另外，为了保证高速系统的稳定性，PLD 的延时预测也是十分重要的。用户在进行系统重构时，担心的是延时特性是否会因重新布线的改变而改变。延时特性的改变将导致系统重构的不稳定性，这对庞大而高速的系统而言是不可想象的，其带来的损失将是巨大的。因此，为了适应未来复杂高速电子系统的要求，PLD 的高速可预测延时也是一个发展趋势。

5. 向混合可编程技术方向发展

可编程器件特有的产品上市快及硬件可重构特性为电子产品的开发带来了极大的方便，它的广泛应用使得电子系统的构成和设计方法均发生了很大的变化。过去几年中，有关可编程器件的研究与开发的大部分工作基本上都集中在数字逻辑电路上。近年来，这一局面已有所改变，可编程模拟器件（programmable analog device，PAD）成为一类新型集成电路。它既属于模拟集成电路，又同可编程逻辑器件一样，可由用户通过现场编程和配置来改变其内部连接与元件参数，从而获得所需要的电路功能。

目前，国外已有几家公司开展了这方面的研究，并且推出了各自的模拟与数模混合型的可编程器件，如美国加州 International Microelectronic Products 公司开发的 EPAC（可编程模拟电路），美国 Lattice 公司推出的一种基于在系统编程技术的可编程模拟电路（in-system programmability programmable analog circuits，ispPAC）。

可编程模拟器件已在数据采集、信号处理、仪器仪表、控制与监测、人工神经网络、电路实验等重要领域得到应用，其典型应用包括信号调理、模拟计算、中高频应用、人工神经网络、电路进化设计等。尽管可编程模拟器件的技术与产品仍显稚嫩，但其内在的便利性和经济性，使我们有理由相信可编程模拟器件是今后模拟电子电路设计的一个发展方向，可编程模拟器件的技术将日益成熟，器件品种必将日益丰富，从而为模拟电路的设计提供一种新的途径，也为电子设计自动化技术的应用开拓更广阔的前景。

思考与练习

1. 什么是 EDA 技术？EDA 技术的基本特征是什么？

本单元练习题

2. 可编程逻辑器件有什么特点？有哪些可编程资源？

3. 写出 Altera 器件中的下列英文缩写的中文含义。

LE　LAB　PIA　EAB

4. 简述可编程逻辑器件的发展趋势。

单元二　可编程逻辑器件的设计与开发

利用 EDA 技术设计电子系统是当代电子工程师必须掌握的内容之一。本单元介绍 EDA 技术的设计流程、设计的开发环境和基本的设计方法。

一、可编程逻辑器件的设计流程

CPLD/FPGA 器件的设计流程一般分为设计输入、设计实现、设计校验和编程下载等几个步骤，如图 2-1 所示。

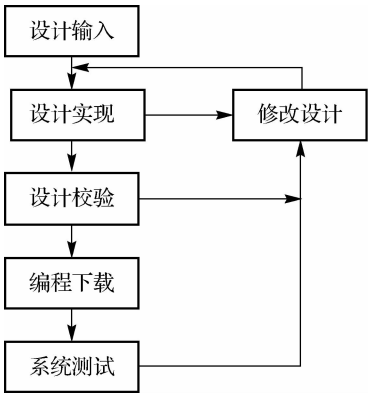


图 2-1　可编程逻辑器件的设计流程

(一)设计输入

设计输入就是将设计者所设计的电路以开发软件所要求的某种形式表示出来，并输入相应的软件中。设计输入有多种表达方式，主要包括原理图输入、硬件描述语言输入、网表输入和波形输入四种，最常用的是原理图输入和硬件描述语言输入。

1. 原理图输入

原理图是图形化的表达方式，它利用软件中所提供的元件符号和连线来描述设计。其特点是比较直观，便于进行接口设计和引脚锁定，容易实现仿真，便于信号的观察和电路的调整，系统运行速率较高，但当描述复杂电路时则比较烦琐。为提高这种输入方式的效率，

应采用“自顶向下”的逻辑分块设计方法。

一般而言，若对系统很了解，并且要求系统的工作速率较高，或在大系统中对时间特性要求较高的部分可采用这种输入方法。

2. 硬件描述语言输入

硬件描述语言输入采用文本方式进行描述设计，这种方式的描述范围较宽，从简单的门电路到复杂的数字系统均可描述。特别是在描述复杂设计时，非常简便。但这种描述方式不适合描述接口和连接关系，并且该输入方式必须依赖综合器，只有好的综合器才能把语言综合成优化的电路。

拓展

VerilogHDL 与 VHDL 简介

对于大量规范的、易于语言描述、易于综合、速率较低的电路，可采用这种输入方式。常用的硬件描述语言有 VerilogHDL、VHDL。

(二)设计实现

设计实现主要是由 EDA 开发工具依据设计输入文件自动生成用于器件编程、波形仿真及延时分析等所需的数据文件。此部分对开发系统来讲是核心部分，但对于用户来说并不用关心它的实现过程，当然设计者也可通过设置设计实现策略等参数来控制设计实现过程。EDA 开发工具进行设计实现时主要完成以下相关任务。

1. 优化和合并

优化是指进行逻辑化简，把逻辑描述转变为最适合在器件中实现的形式；合并是将模块化设计产生的多个文件合并成一个网表文件，并使层次设计平面化。

2. 映射

映射是根据所选择的 PLD 型号，把设计分割为多个适合器件内部逻辑资源实现的逻辑小块形式。

3. 布局和布线

布局是将已分割的逻辑小块放到器件内部逻辑资源的具体位置，并使它们易于连线，且连线最少；布线是利用器件内的布线资源完成各功能块之间和反馈信号的连接。

4. 产生编程文件

设计实现的最后一步是产生可供器件编程使用的数据文件。对 CPLD 而言，产生的是熔丝图文件(*.JEDEC)；对 FPGA 器件，则产生位数据流文件。

(三)设计校验

设计校验就是让计算机根据编译器所产生的数据文件对 EDA 设计进行模拟，以验证用户的设计是否正确。设计校验包括仿真和定时分析两部分，这两部分可分别通过仿真器和延时分析器来完成。在仿真文件中加载不同的激励，可以观察中间结果及输出波形。必要时，可以返回设计输入阶段，修改设计输入，最终达到设计要求。

设计校验可对设计中的各个模块乃至整个系统进行，若有错误，可以很方便地修改，而不必对硬件进行改动，极大地节约了成本。规模越大的设计，越需要设计仿真。仿真不消耗器件内的资源，仅消耗少许时间，但从节约成本的角度考虑，这种时间的消耗是完全值得的。

可以认为仿真是 EDA 的精髓。

(四)编程下载

编程下载是将设计实现阶段所产生的熔丝图文件或位数据流文件装入可编程逻辑器件中,以便硬件调试和验证。编程下载需要满足一定的条件,如编程电压、编程时序和编程算法等。在编程下载时需注意以下几方面的问题。

(1)对于不能进行在系统编程的 CPLD 和不能再重配置的 FPGA 器件,需要编程专用设备(编程器)完成器件编程。

(2)对于使用 LUT 技术和基于 SRAM 的 FPGA 器件,下载的编程数据将存入 SRAM 中,而 SRAM 掉电后所存的数据将丢失,为此须将编程数据固化到 E^2PROM 中。器件上电时,由器件本身或微处理器控制 E^2PROM 将数据配置到 FPGA 中。

(3)对于使用乘积项逻辑、基于 E^2PROM 或 Flash 工艺的 CPLD 进行编程下载时,使用器件厂商提供的专用下载电缆,一端与计算机的并行口相接,另一端接到 CPLD 所在 PCB 上的 10 芯插头上(PLD 只有 4 个引脚与该插头相连)。编程数据通过该电缆下载到 CPLD 中,这个过程称为在系统编程(ISP)。

二、可编程逻辑器件的开发环境

拓展

Viewlogic 与 Cadence 简介

如前所述,EDA 技术在当代迅猛发展,同时各种 EDA 软件也如雨后春笋般呈现在用户面前。它们一般分为两种,一种是 PLD 芯片制造商为推广自己的芯片而开发的专业 EDA 软件,本书所使用的 Altera 公司推出的 Quartus Ⅱ 就属于此类;另一种是 EDA 软件商提供的第三方软件,如知名的 Synplify、Synopsys、Viewlogic、Cadence 等,这种软件可以支持大部分芯片公司的 PLD。

(一)常用的 EDA 工具软件

下面介绍几种应用最广泛的 EDA 工具软件。

1. Synplify

该软件是由 Synplicity 公司专为 FPGA 和 CPLD 开发设计的逻辑综合工具。它在综合优化方面的优点非常突出,得到了广大用户的好评。它支持用 Verilog 和 VHDL 硬件描述语言描述的系统级设计,具有强大的行为及综合能力。综合后,能生成 Verilog 或 VHDL 网表,以进行功能级仿真。

Synplify 的综合过程分为三步:首先是语言综合,将硬件描述语言的设计编译成结构单元;接下来采用优化算法对设计进行优化,除去冗余项,提高可靠性与速度;最后是工艺映射,将设计映射为相应 PLD 的网表文件。

2. Synopsys

该软件是另一种系统综合软件,它因综合功能强大而被广泛使用。Synopsys 综合器的综合效果比较理想,系统速度快,消耗资源少。对系统的优化过程大致分为两步:第一步是设计规则,提出必须满足的设计要求,如最大延时、最大功耗、最大扇出数目、驱动强度等;第二步是提出各种设计约束,一般有反应时间约束、芯片面积约束等。综合器根据设计要求,

采用相应算法，力争使综合效果达到最佳。

Synopsys 支持完整的 VHDL 和 Verilog 语言子集，另外它的元件库中包含许多现成的实现方案，调用非常方便。正是因为这些突出的优点，Synopsys 逐渐成为设计人员普遍接受的标准工具。

3. ispDesignEXPERT

该软件是 Lattice 公司专为本公司的 PLD 芯片开发设计的软件，它的前身是该公司的 Synario、ispEXPERT。ispDesignEXPERT 是完备的 EDA 软件，支持系统开发的全过程，包括设计输入、设计实现、仿真与时序分析、编程下载等。

ispDesignEXPERT 包括三个版本，其中 Starter 版适合初学者学习，可以免费下载，Base 版为试用版，它们的设计规模都低于 600 个宏单元；Advanced 版是专业设计版，支持该公司的各种系列器件，功能全面。

4. MAX＋plus Ⅱ

该软件是 Altera 公司专为本公司的 PLD 芯片开发设计的软件。其功能齐全，使用方便，易懂好学，是最广为接受的 EDA 工具之一。

5. Quartus Ⅱ

该软件也是 Altera 公司为本公司的 PLD 芯片开发设计的软件。它比 MAX＋plus Ⅱ 支持的器件更全面，特别包括 Altera 公司的超高密度的芯片系列——APEX 系列器件。Quartus Ⅱ 可开发的单器件门数达到了 260 万门，特别适合高集成的大型系统的开发设计。

(二)Quartus Ⅱ 软件介绍

Quartus Ⅱ 软件是 Altera 的综合开发工具，它集成了 Altera 的 FPGA/CPLD 开发流程中所涉及的所有工具和第三方软件接口。通过使用此综合开发工具，设计者可以创建、组织和管理自己的设计。

1. Quartus Ⅱ 软件的特点

(1)广泛的适用范围。Quartus Ⅱ 软件支持的器件种类众多，主要有 Stratix^{G}X 和 Stratix Ⅱ、Stratix、Cyclone、Cyclone Ⅱ、APEX Ⅱ 系列、APEX20KC、APEX20KE、Mercury 系列、FLEX10K 系列、FLEX6000 系列、MAX Ⅱ 系列、MAX3000A 系列、MAX7000 系列、MAX9000 系列等。

(2)支持 Windows、Solaris、HP-UX 和 Linux 等多种操作系统。

(3)支持多时钟定时分析、LogicLock™基于块的设计、SOPC(单芯片可编程系统)，内嵌 SignalTap Ⅱ 逻辑分析器、功率估计器等高级工具。

(4)Quartus Ⅱ 是全集成化的设计平台，支持多种输入方式，具有逻辑综合、布局布线、模拟、时序分析、器件编程等功能。

(5)易学易用。Quartus Ⅱ 软件提供丰富的图形用户接口，软件界面新颖友好，通过短期学习就能熟练掌握。

2. Quartus Ⅱ 软件的用户界面

启动 Quartus Ⅱ 软件后的默认界面如图 2-2 所示，由标题栏、菜单栏、工具栏、资源管理窗、工程工作区、信息显示窗等部分组成。

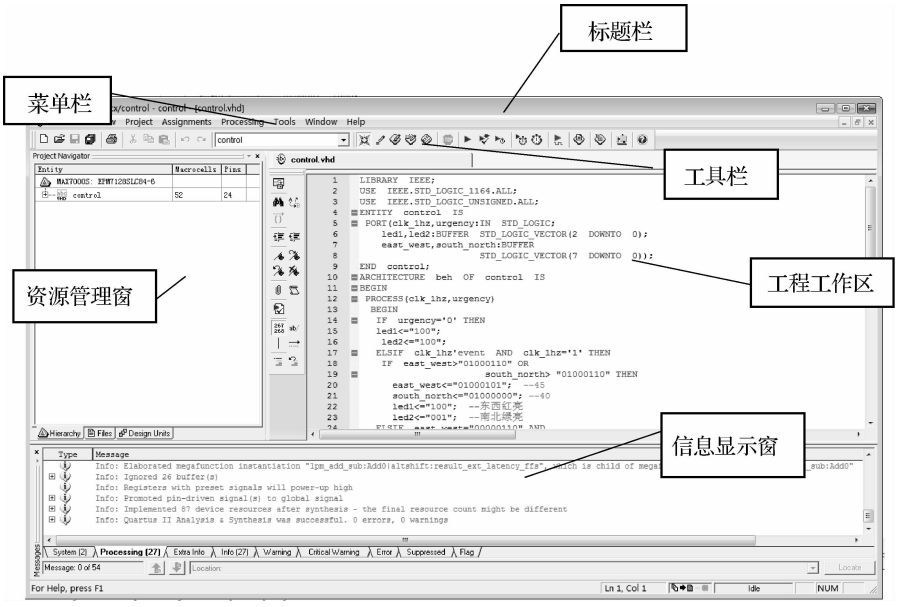


图 2-2　Quartus Ⅱ 软件的用户界面

1)标题栏

标题栏显示当前工程的路径和程序的名称。

2)菜单栏

菜单栏主要由文件(File)、编辑(Edit)、视图(View)、工程(Project)、资源分配(Assignments)、操作(Processing)、工具(Tools)、窗口(Window)和帮助(Help)共 9 个菜单项组成。其中，Project、Assignments、Processing、Tools 集中了 Quartus Ⅱ 软件较为核心的全部操作命令。

(1)Project 菜单主要是对工程的一些操作。其中，Add/Remove Files in Project 用于添加或新建某种资源文件，Set as Top-Level Entity 用于把工程工作区打开的文件设定为顶层文件。

(2)Assignments 菜单的主要功能是对工程的参数进行配置，如引脚分配、时序约束、参数设置等。

(3)Processing 菜单包含了对当前工程执行的各种设计流程，如开始综合、开始布局布线、开始时序分析等。

(4)Tools 菜单用于调用 Quartus Ⅱ 软件中集成的一些工具，如 MegaWizard Plug-In manager(用于生成 IP 核和宏功能模块)、Chip Editor、RTLViewer、Programmer 等。

3)工具栏

工具栏中包含了常用命令的快捷图标，如设置(Settings)、编译(Compile)等。将鼠标指

针移到相应的图标上时，就会出现此图标对应的含义，而且每种图标在菜单栏均能找到相应的命令菜单。

4)资源管理窗

资源管理窗用于显示当前工程中所有相关的资源文件。资源管理窗左下角有 3 个标签，分别是结构层次(Hierarchy)、文件(Files)和设计单元(Design Units)。结构层次窗口在工程编译之前只显示了顶层模块名，工程编译一次后，此窗口按层次列出工程中的所有模块，并列出每个源文件所用资源的具体情况。顶层可以是用户产生的文本文件，也可以是图形编辑文件。

5)工程工作区

器件设置、定时约束设置、底层编辑器和编译报告等均显示在工程工作区中，当 Quartus Ⅱ 实现不同功能时，此区域将打开相应的操作窗口，显示不同的内容，进行不同的操作。

6)信息显示窗

信息显示窗显示 Quartus Ⅱ 软件综合、布局布线过程中的信息，如开始综合时调用源文件、库文件、综合布局布线过程中的定时、警告、错误等，如果是警告和错误，则会给出具体的引起警告和错误的原因，方便设计者查找和修改错误。

三、设计实例

本节将通过一个简单的实例具体说明使用 Quartus Ⅱ 软件进行数字系统设计的基本过程。我们将设计一个 1 位全加器，按照设计输入、编译、分配 I/O 引脚、仿真、定时分析和编程下载共 6 个阶段来完成设计。

本设计是以原理图的方式进行的设计输入，目的是让读者对设计过程有一个整体的了解。对于硬件描述语言的设计输入，只是设计输入方式的不同，其他过程完全相同。

由数字电路知识可知，1 位全加器(见图 2-3)可以由 2 个半加器及 1 个或门组成。因此，需要首先完成半加器(见图 2-4)的设计。以下将给出使用原理图输入的方法进行底层元件设计和层次化设计的完整步骤，其主要流程与数字系统设计的一般流程基本一致。

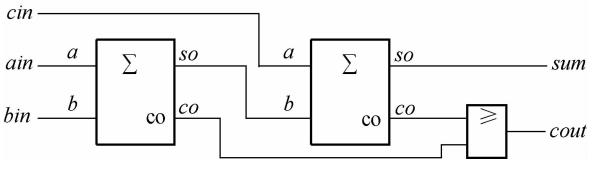


图 2-3　1 位全加器

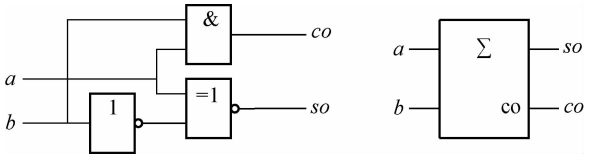


图 2-4　半加器的逻辑图与符号

1. 为本项工程设计建立文件夹

任何一项设计都是一项工程，都必须首先为此工程建立一个文件夹，以放置与此工程相关的文件，此文件夹将被 EDA 软件默认为工作库(Work Library)。一般不同的设计项目最好放在相应的文件夹中。文件夹的命名不能用中文，且不可带空格。

假设本项目设计的文件夹取名为 EDAjc，路径为 E:\EDAjc。

2. 建立工程

(1)打开软件，显示 Quartus Ⅱ 的主界面，如图 2-5 所示。

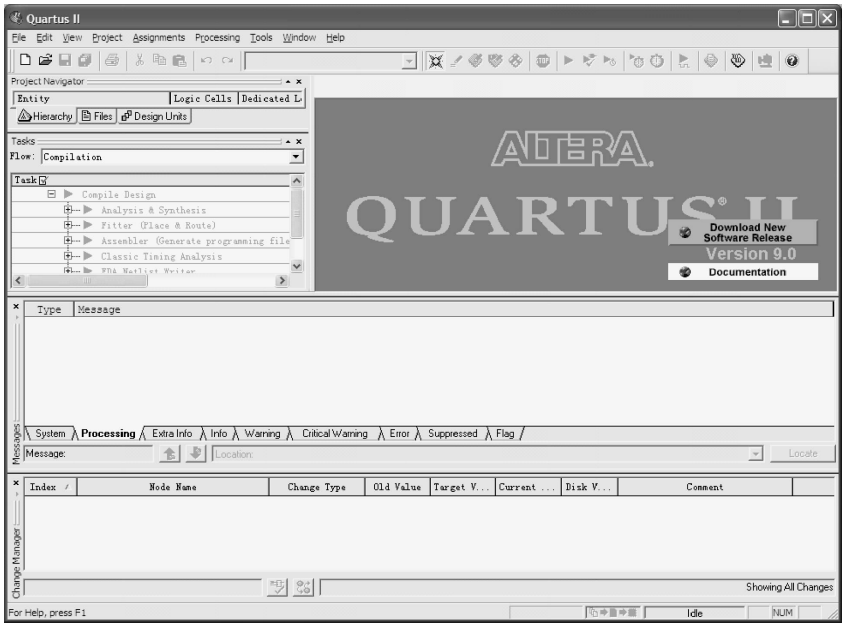


图 2-5　Quartus Ⅱ 软件的主界面

(2)新建工程。执行 File→New Project Wizard 菜单命令，弹出图 2-6 所示的对话框，在“工程路径”文本框选择之前为工程建立的目录 E:\EDAjc，在“工程名称”文本框中输入所建工程的名称，在“顶层模块名”文本框中输入模块名，要求模块名与工程名相同。然后单击 Next 按钮，弹出图 2-7 所示的对话框，添加已有文件。

(3)添加已有文件。没有已有文件的直接跳过，单击 Next 按钮，弹出图 2-8 所示的对话框，选择芯片型号。

(4)选择芯片型号。在 Family 下拉列表框中选择器件系列。首先应该选定目标器件对应的系列名，如 EPM7128SLC84-6 对应的是 MAX7000S 系列。在 Available devices 列表框

中选择具体的器件型号，如图 2-8 所示。完成选择后，单击 Next 按钮，弹出图 2-9 所示对话框。

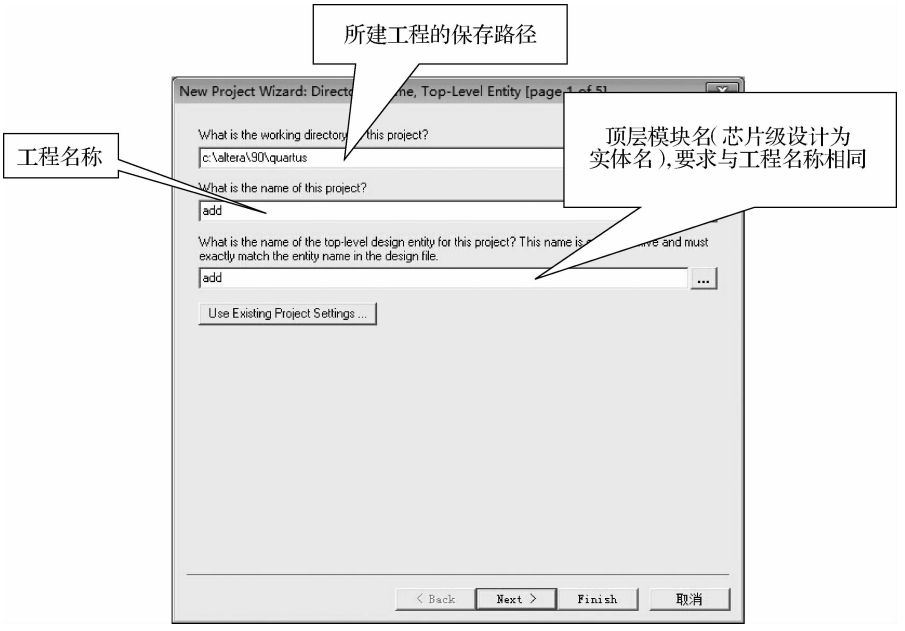


图 2-6　新建工程

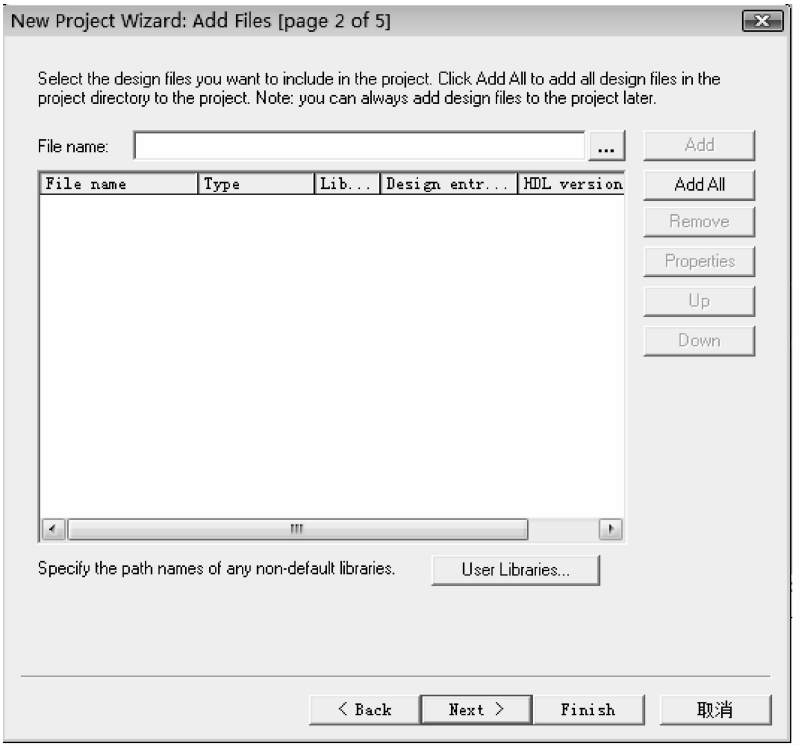


图 2-7　添加已有文件

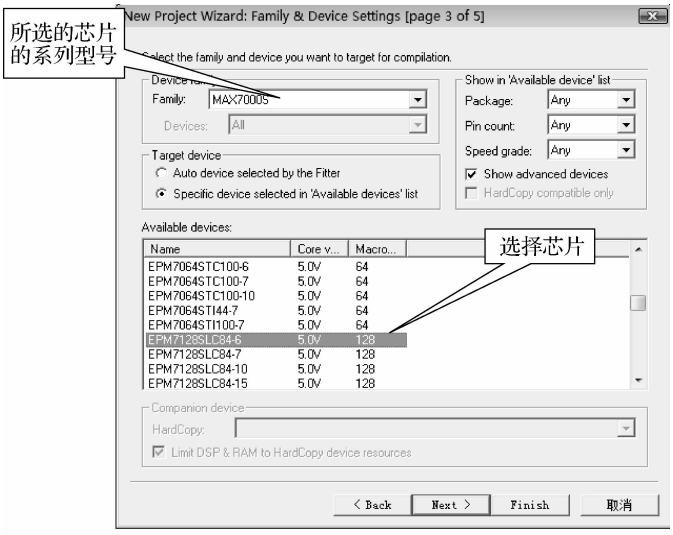


图 2-8　选择芯片型号

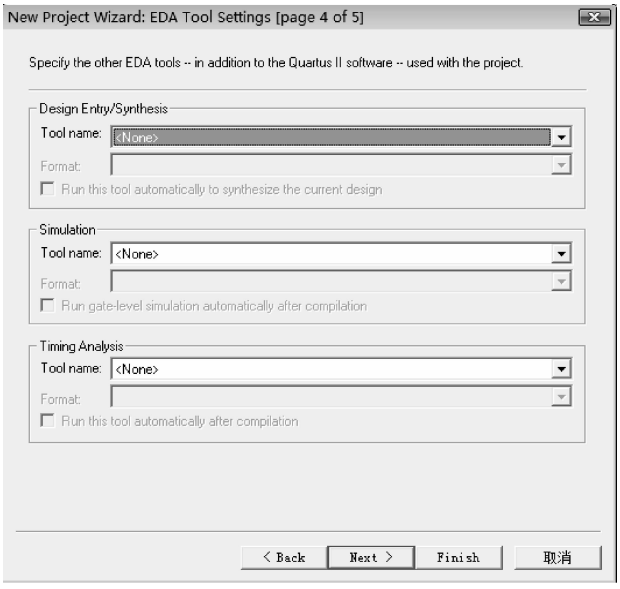


图 2-9　选择综合、仿真、时序分析工具

(5)选择综合、仿真、时序分析工具，如果只利用 Quartus Ⅱ，则在 Design Entry/Synthesis、Simulation 和 Timing Analysis 三个选项区的 Tool name 下拉列表框中都选 None，然后单击 Next 按钮，弹出图 2-10 所示对话框。

(6)工程建立完成。如图 2-10 所示，该对话框显示所建立工程的所有芯片、其他第三方 EDA 工具选择情况及模块名等信息，单击 Finish 按钮，完成工程的建立。

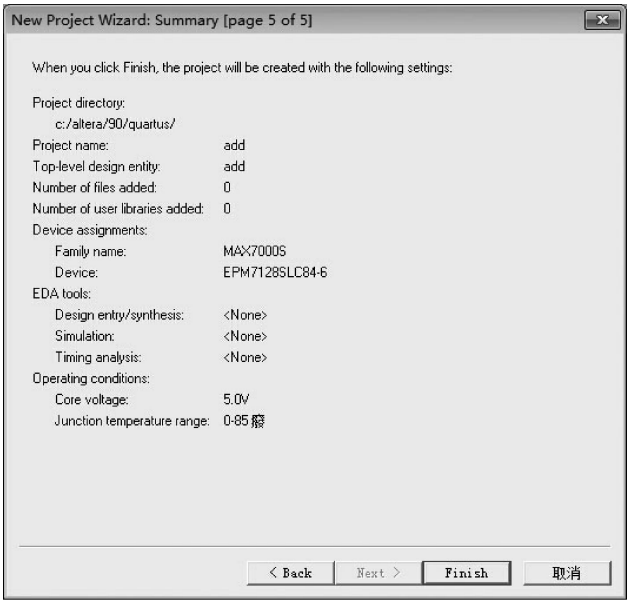


图 2-10　工程建立完成

3. 添加设计文件并保存

(1)执行 File→New 菜单命令，如图 2-11 所示，在弹出的 New 对话框中选择 Block Diagram/Schematic File(建立原理图编辑文件)，如果采用 VHDL 文本输入形式，则选择 VHDL File。然后单击 OK 按钮，进入原理图编辑界面。

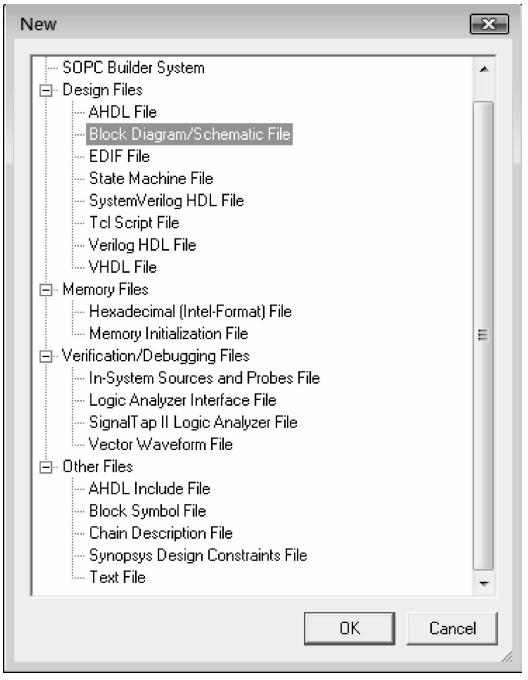


图 2-11　建立新的设计文件

(2)在原理图编辑界面的任意编辑位置双击,弹出图 2-12 所示的 Symbol 对话框。双击元件库 Libraries 中的 c:/altera/91/quartus/libraries/,元件库目录下面即显示几个库,常用的基本元件库为 primitives 库,双击打开 primitives 库,然后双击 logic 打开逻辑库查找基本逻辑元件,双击 pin 打开引脚库查找输入输出引脚。然后单击 OK 按钮,即可将元件调入原理图编辑窗口中。分别调入 and2、not、xnor、input 和 output,并连好线,如图 2-13 所示。然后双击 input 和 output 上的 pin_name,修改引脚名。

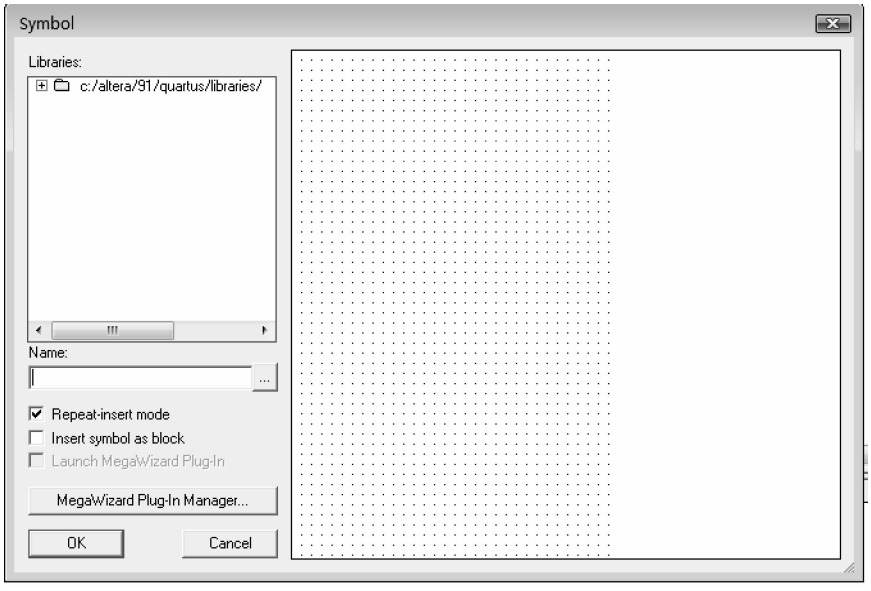


图 2-12 Symbol 对话框

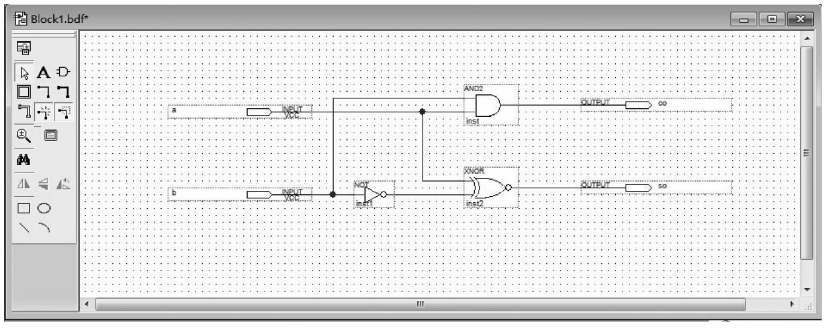


图 2-13 原理图编辑

(3)执行 File→Save As 菜单命令,选择目录 E:\EDAjc,将已设计好的原理图取名为 add.bdf(扩展名是默认的),并存盘在此目录内,如图 2-14 所示。

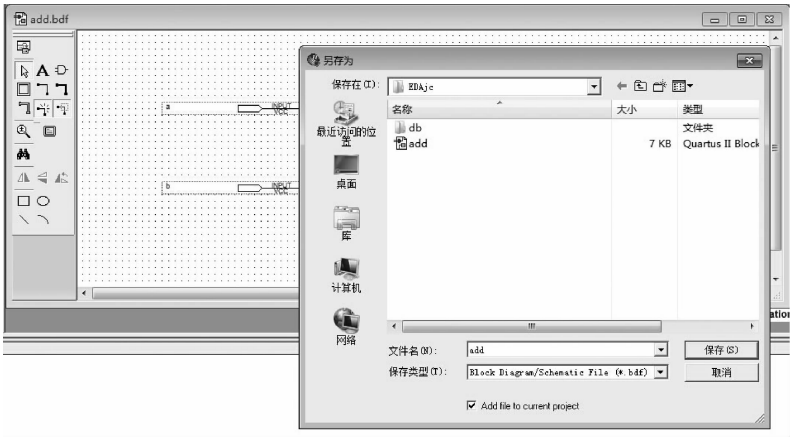


图 2-14　原理图文件存盘

4. 检查设计

单击工具栏中的按钮进行检查，如果出错，需要检查并修改设计直至成功为止，如图 2-15 所示，单击“确定”按钮，完成设计检查。

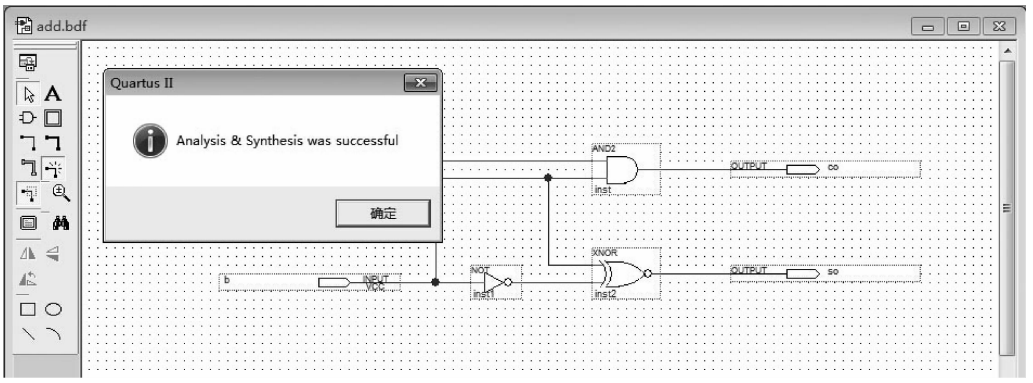


图 2-15　设计检查

5. 锁定引脚并编译

(1)单击工具栏中的按钮，弹出图 2-16 所示的对话框，进行引脚配置。双击 Location 为设计的输入输出配置引脚。

(2)整体编译。单击工具栏中的按钮，编译成功，弹出 Flow Summary 窗口，该窗口给出综合后代码的资源使用情况及芯片型号等信息，如图 2-17 所示。

6. 功能仿真

(1)将仿真类型设置为功能仿真，执行 Assignments→Settings 菜单命令，弹出图 2-18 所示对话框。在 Simulation mode 下拉列表框中选择 Functional。Functional 表示功能仿真，即不包括时序信息；Timing 表示时序仿真；Timing using Fast Timing Model 表示快速时序仿真。然后单击 OK 按钮。

(2)建立一个波形文件。执行 File→New 菜单命令,弹出 New 对话框,如图 2-19 所示,选择 Vector Waveform File,然后单击 OK 按钮,完成波形文件的建立。波形文件作为信号输出文件主要通过它观察信号的输出情况。

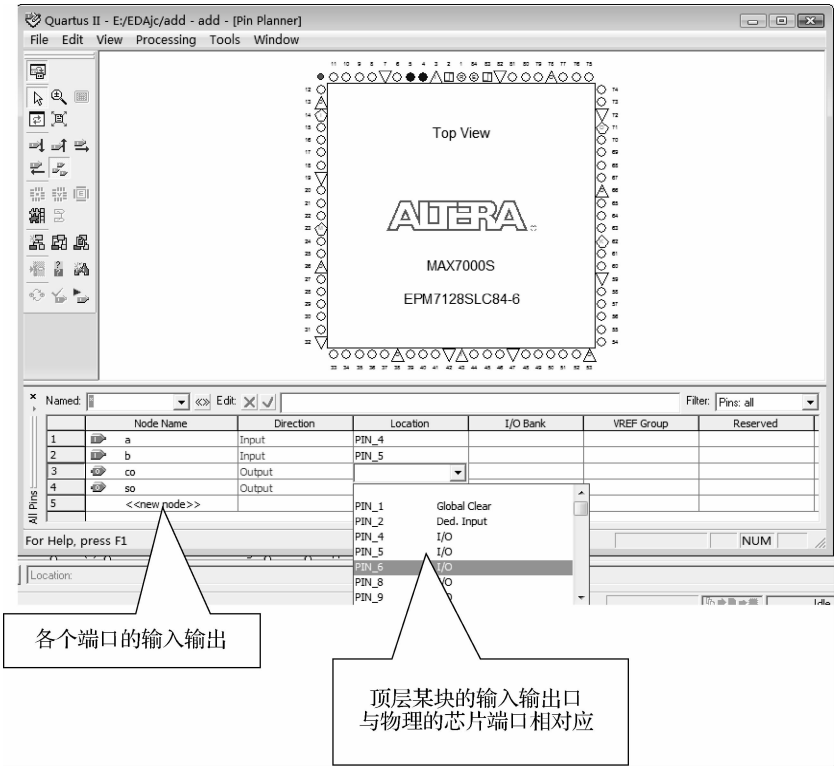


图 2-16　配置引脚

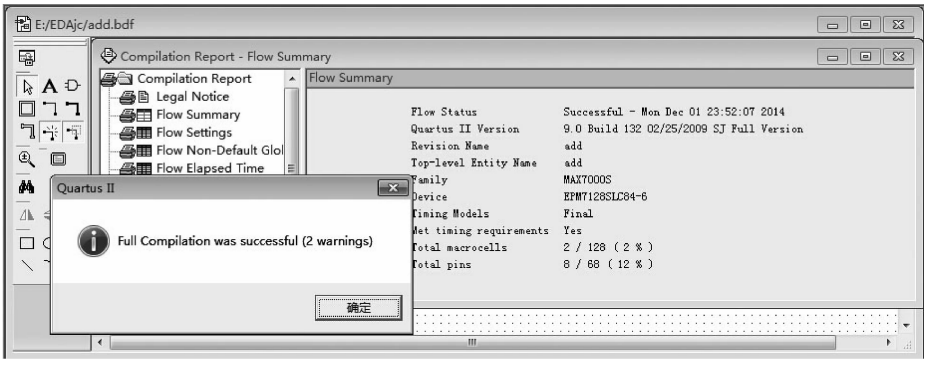


图 2-17　整体编译

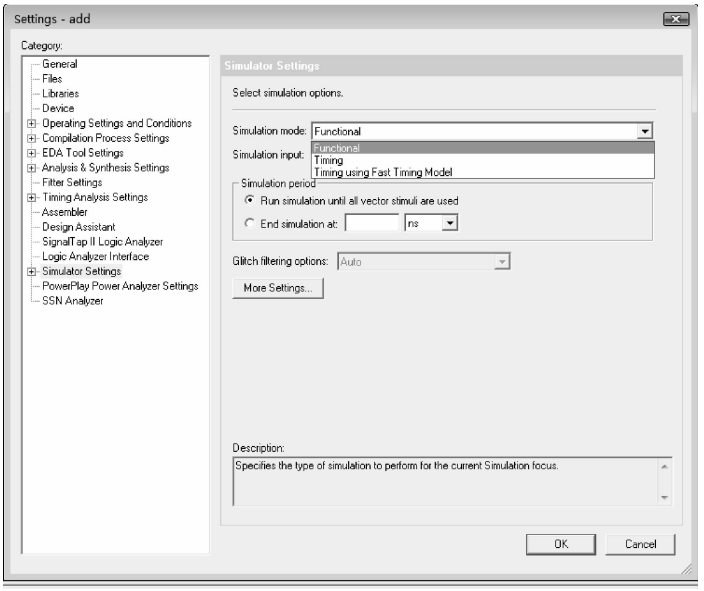


图 2-18　功能仿真设定

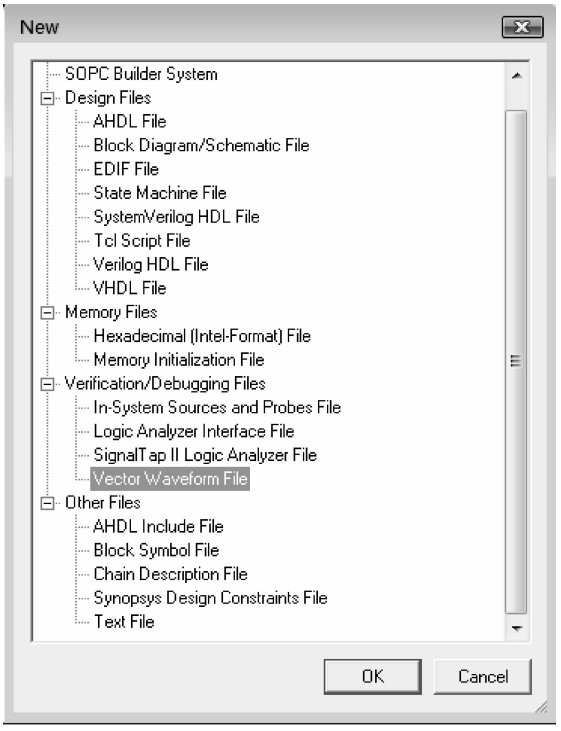


图 2-19　新建波形文件

(3)导入引脚。如图 2-20 所示,双击 Name 下面的空白区域,弹出 Insert Node or Bus 对话框。单击 Node Finder 按钮,弹出 Node Finder 对话框,单击 List 按钮显示引脚信息,单击 >> 按钮选择节点,产生端口列表。设置完成后单击 OK 按钮。

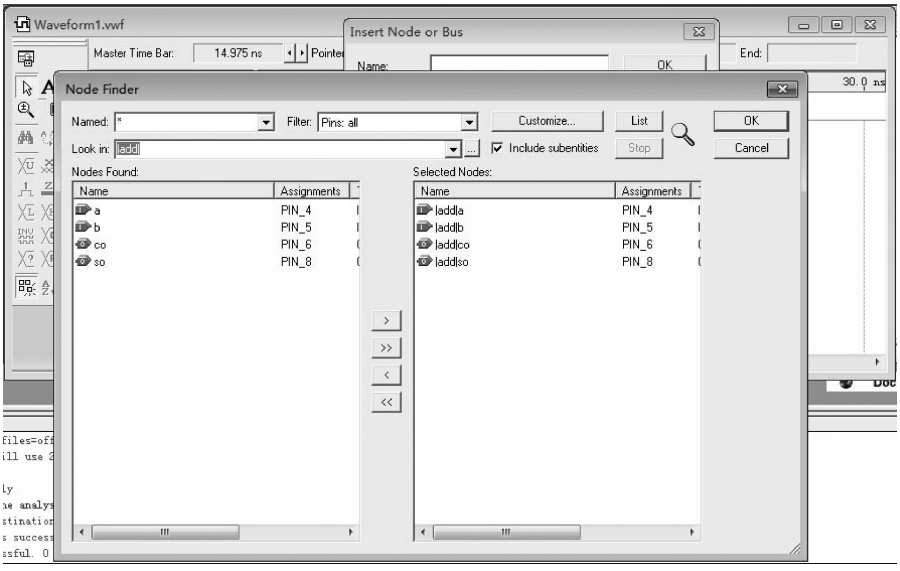


图 2-20　列出并选择需要观察的节点信号

(4)设置激励信号。单击输入引脚 0 a A 0 ,用鼠标拖黑相应的段,在左侧选择赋值信号,如图 2-21 所示。

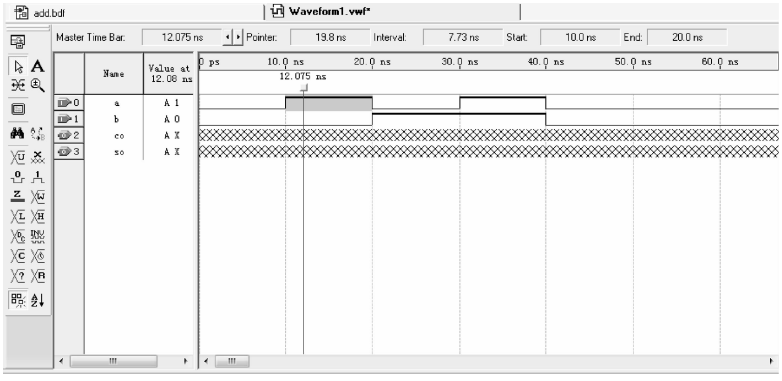


图 2-21　为输入信号设定必要的测试电平或数据

(5)保存建立的波形图文件,文件名和路径为默认,执行 Processing→Generate Functional Simulation Netlist 菜单命令,生成仿真需要的网表。

(6)开始仿真。执行 Processing→Start Simulation 菜单命令,或者单击工具栏中的 按钮。观察波形,如果仿真波形符合逻辑,则功能仿真通过,如图 2-23 所示。

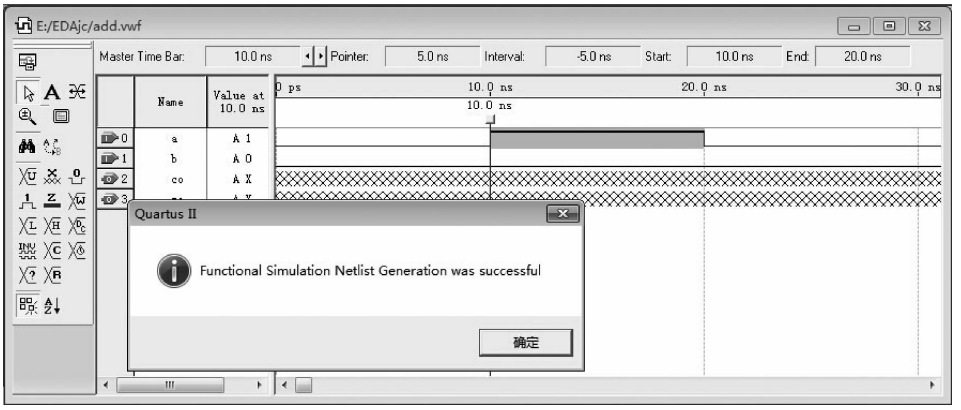


图 2-22　生成网表

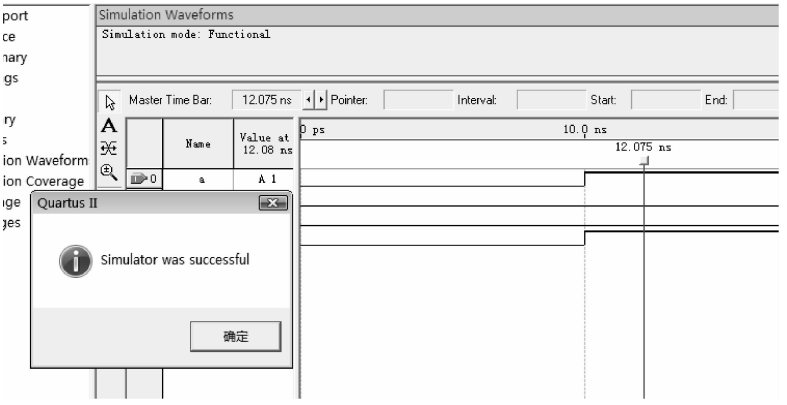


图 2-23　仿真完成

7. 下载

单击工具栏中的按钮，再单击 Hardware Setup 按钮配置下载电缆。单击弹出对话框中的 Add Hardware 按钮，在 Hardware type 下拉列表框中选择 ByteBlasterMV or ByteBlasterMV Ⅱ（并口下载），单击 Close 按钮完成设置。CPLD 生成的下载文件的扩展名为. pof，在图 2-24 中选中下载文件，然后直接单击 Start 按钮开始下载。

8. 顶层文件设计

（1）底层元件封装入库。打开已设计好的半加器，执行 File→Open 菜单命令，在弹出的对话框中选择前面设计好的半加器设计文件并打开，如图 2-25 所示。然后执行 File→Creat/Update→Creat Symbol Files for Current File 菜单命令，生成元件封装，如图 2-26 所示。

（2）添加新的设计文件并保存，在图 2-27 所示的元件输入窗口找到本工程目录中已封装好的半加器元件，并将它调入原理图编辑窗中。

（3）如图 2-28 所示，完成全加器原理图设计，并以文件名 qadd. bdf 存在同一目录下。

（4）选择目标器件 EPM7128SLC84-6，并编译。

(5)建立波形仿真文件,设定输入信号电平,启动仿真,观察输出波形,如图 2-29 所示。

(6)手动分配 I/O 引脚,编译并编程下载,硬件实测此全加器的逻辑功能。

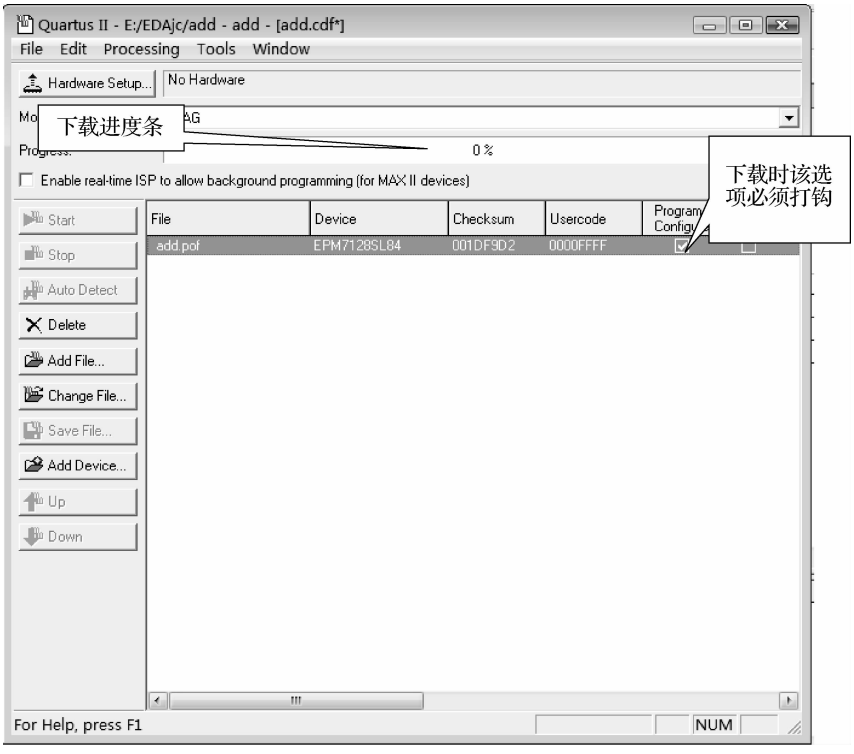


图 2-24　下载文件

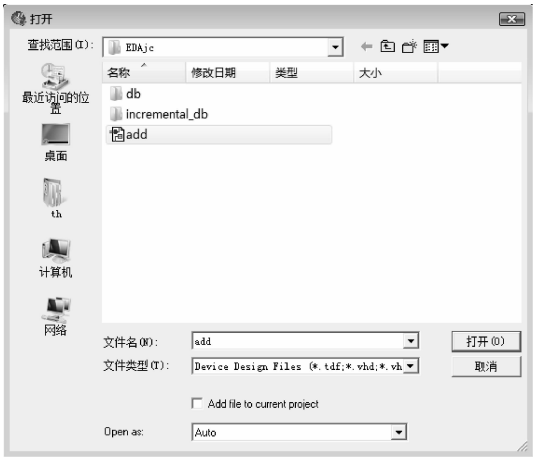


图 2-25　选择并打开文件

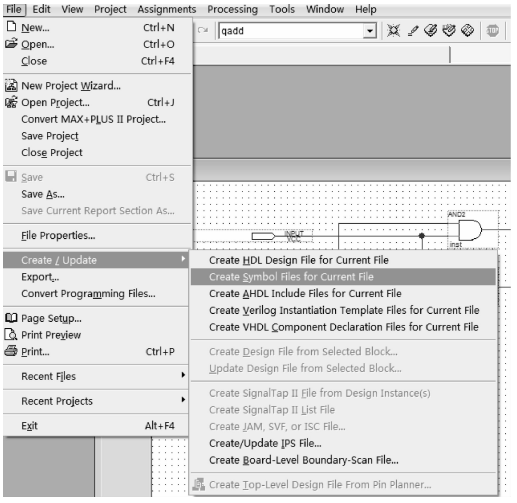


图 2-26　封装入库

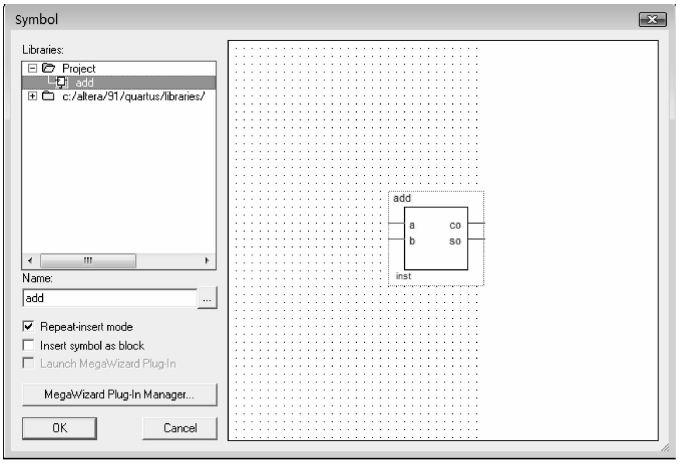


图 2-27　调用半加器元件

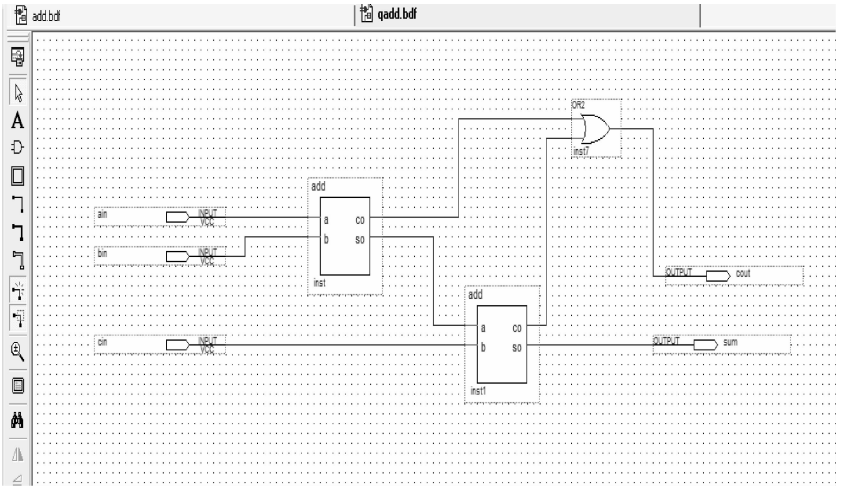


图 2-28　全加器原理图设计

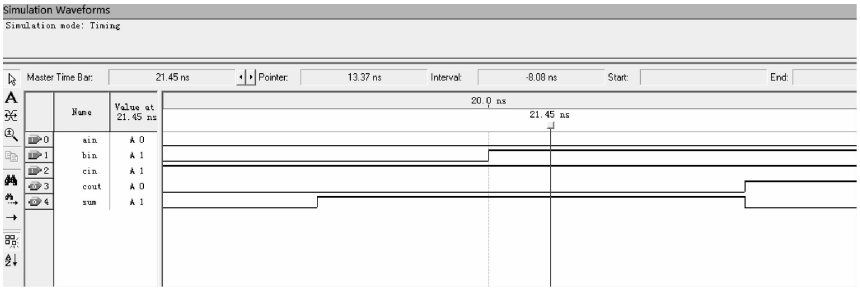


图 2-29　全加器仿真波形图

思考与练习

1. 简述可编程逻辑器件的一般设计过程。
2. 试比较可编程逻辑器件设计的常用输入方式。
3. 列举常用的 EDA 工具软件，并比较各有什么特点。
4. EDA 开发工具在设计实现过程中主要完成哪些工作？
5. 简述 Quartus Ⅱ 进行 EDA 设计的一般步骤。

本单元练习题

单元三　硬件描述语言

随着电子技术的发展，人们对电子产品性能的要求越来越高，电子产品功能越来越多，相应的电子线路也越来越复杂，而采用传统的电子线路设计方法（如原理图设计）显得越来越困难。EDA 技术的出现为复杂电子线路的设计提供了新途径。硬件描述语言（hardware description language，HDL）是 EDA 工具最常用的一种设计输入方式，它可采用语言化的方式来描述电路的行为，为复杂的电子线路设计提供了一条捷径。

一、硬件描述语言概述

目前，国际上越来越多的 EDA 工具都接收 HDL 作为设计输入，因此世界各大公司也都相继开发了自己的 HDL。进入 20 世纪 80 年代后期，硬件描述语言向着标准化、集成化的方向发展，最终，VHDL 和 VerilogHDL 适应了这种趋势的要求，先后成为 IEEE 标准。因此，目前应用最广泛的硬件描述语言有 VHDL 和 VerilogHDL 两种。

（一）VerilogHDL 的发展和功能

VerilogHDL（Verilog hardware description language，Verilog 硬件描述语言）是 1983 年由 GDA 公司的 Phil Moorby 首创的。Phil Moorby 设计出第一个关于 Verilog-XL 的仿真器，并提出了用于快速门级仿真的 XL 算法。随着 Verilog-XL 算法的成功，VerilogHDL 得到了迅速的发展。1989 年，Cadence 公司收购了 GDA 公司，并公开了 VerilogHDL。基于 VerilogHDL 的优越性，IEEE 于 1995 年制定了 VerilogHDL 的 IEEE 标准，即 VerilogHDL 1364-1995。

VerilogHDL 是专为 ASIC（application specific intergrated circuits，专用集成电路）设计而开发的。在亚微米和深亚微米 ASIC 已成为电子设计主流的今天，VerilogHDL 的发展前景是非常远大的。VerilogHDL 较为适合算法级、寄存器传输级、逻辑级和门级的设计，可以很容易地把完成的设计移植到不同厂家的不同芯片中，并且很容易修改设计。采用 VerilogHDL 输入法的最大优点是其与工艺的无关性，这使得设计者在功能设计、逻辑验证阶段可不必过多考虑门级及其工艺实现的具体细节，只需利用系统设计时对芯片的要求，施加不同的约束条件，即可设计出实际电路。

VerilogHDL 完全集成于 MAX+plusⅡ中，它可以包含 MAX+plusⅡ支持结构的任何形式的组合，也可以包含 Altera 提供的元器件、兆功能模块和宏功能模块。利用 VerilogHDL 可以进行功能描述、仿真验证、时序分析、逻辑综合等。因此，它是目前应用较为广泛的一种硬件描述语言。

(二)VHDL 的发展和功能

VHDL(very high speed integrated circuit hardware description language，超高速集成电路硬件描述语言)是美国国防部于 20 世纪 80 年代后期出于军事工业的需要开发的。1984 年，VHDL 被 IEEE 确定为标准化的硬件描述语言。1994 年，IEEE 对 VHDL 进行了修订，增加了部分新的 VHDL 命令与属性，增强了系统的描述能力，并公布了新版本的 VHDL，即 IEEE 标准版本 1046-1994。VHDL 已经成为系统描述的国际公认标准，得到众多 EDA 公司的支持，越来越多的硬件设计者使用 VHDL 描述系统的行为。

VHDL 涵盖面广，抽象描述能力强，支持硬件的设计、验证、综合与测试。VHDL 能在多个级别上对同一逻辑功能进行描述，如可以在寄存器级别上对电路的组成结构进行描述，也可以在行为描述级别上对电路的功能与性能进行描述。无论哪种级别的描述，都有赖于优良的综合器将其转化为具体的硬件结构。

利用 VHDL 描述设计，设计者可以不懂硬件结构，也不必考虑最终实现的目标器件是什么，只需用正确的语言描述系统的行为即可。正因为 VHDL 的硬件描述与具体的工艺技术和硬件结构无关，使得 VHDL 设计程序的硬件实现目标器件有广阔的选择范围，其中包括各系列的 CPLD、FPGA 及各种门阵列器件。

在各种硬件描述语言中，VHDL 的行为抽象描述能力是最强的，从而决定了它成为系统设计领域最佳的硬件描述语言。

(三)VerilogHDL 与 VHDL 的比较

一般的硬件描述语言可在三个层次上进行电路描述，其层次由高到低依次可分为行为级、RTL 级和门电路级。VHDL 通常更适合行为级和 RTL 级的描述；VerilogHDL 通常只适合 RTL 级和门电路级的描述。因此，与 VerilogHDL 相比，VHDL 是一种高级描述语言，适用于电路的高级建模，最适合描述电路的行为，即描述电路的功能，但它几乎不能直接控制门电路，即控制电路的资源；VerilogHDL 则是一种低级的描述语言，最适合于描述门级电路，易于控制电路资源。

显然，VHDL 和 VerilogHDL 主要区别在于逻辑表达的描述级别。VHDL 虽然也可以直接描述门电路，但这方面的能力却不如 VerilogHDL；相反，VerilogHDL 在高级建模描述方面不如 VHDL。VerilogHDL 的描述风格接近于电路原理图，从某种意义上说，它是电路原理图的高级文本表示方式。

任何一种语言源程序，最终都要转换成门电路级才能被布线器或适配器所接收。VHDL 的综合通常要经过“行为级—RTL 级—门电路级”的转化；而 VerilogHDL 的综合过程要简单些，即经过“RTL 级—门电路级”的转化。

由于 VHDL 和 VerilogHDL 各有所长，市场占有量也相差不多。VHDL 描述语言层次较高，不易控制底层电路，因而对综合器的综合性能要求较高。但当设计者积累一定经验后会发现，每种综合器一般将一定描述风格的语言综合成确定的电路，只要熟悉基本单元电路的描述风格，综合后的电路还是易于控制的。VHDL 入门相对稍难，但在熟悉以后，设计效率明显高于 VerilogHDL，生成的电路性能也与 VerilogHDL 的不相上下。在 VHDL 设计中，综合器完成的工作量是巨大的，设计者所做的工作就相对减少了；而在 VerilogHDL 设计中，工作量通常比较大，因为设计者需要搞清楚具体电路结构的细节。本书的内容以 VHDL 为主。

二、VHDL 的程序结构

小到一个元件、一个电路，大到一个系统，都可以用 VHDL 来描述其结构、行为、功能和接口。本节介绍 VHDL 的程序结构、常用语句及其使用方法。

(一)一般结构

VHDL 程序结构的一个显著特点是，任何一个完整的设计项目(或称为设计实体，简称实体)都可以分为两部分。第一部分主要用于描述电路的外部端口，包括器件名称、端口名称、数据类型等，称为“实体说明”；第二部分主要用于描述电路的内部结构、功能及其实现的算法，称为“结构体”。以上两部分是一个 VHDL 程序必备的两部分，缺一不可。当一个设计实体的内、外两部分都设计完成后，其他设计实体就可以像调用普通元件一样直接调用它。

下面通过一个简单的例子来说明 VHDL 程序的一般结构。

【例 3-1】 试用 VHDL 描述一个 2 输入端的与非门。

```
--第一部分:实体说明
ENTITY nand2 IS                    --实体说明描述,实体名称为 nand2
  PORT(a,b:IN BIT;                 --a、b 为 2 个输入引脚,数据类型为 BIT
       y:OUT BIT);                 --y 为输出引脚,数据类型为 BIT
END nand2;                         --结束实体说明
--第二部分:结构体
ARCHITECTURE a OF nand2 IS         --结构体描述,名称为 a,是 nand2 的结构体
BEGIN
  y<=a NAND b;                     --a 和 b 进行与非运算,结果赋给 y
END a;                             --结束结构体描述
```

该例子包括了一个 VHDL 程序必备的两部分：实体说明和结构体。

(1)实体说明部分给出了器件 nand2 的外部引脚(PORT)，如图 3-1 所示。A、B 为输入引脚，Y 为输出引脚，数据类型均为 BIT。BIT 指的是 1 位二进制数，只有两种逻辑取值，即 0 和 1。

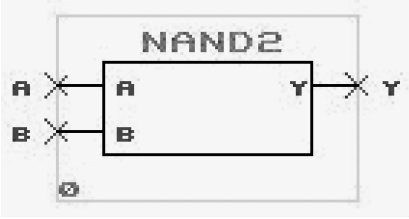


图 3-1　实体 nand2 的元件符号

(2)结构体部分给出了 nand2 的内部功能信息。其中,NAND 是实现与非运算的运算符;“<=”是赋值运算符,从电路的角度来看就是表示信号的传输,即将 A 和 B 与非运算后的结果信号传输给 Y。

从上述例子还可看出,VHDL 的所有语句都是以“;”结束的,语句后面的“--”部分表示程序的注释。

(二)实体

实体就是设计对象或设计项目。实体可代表任何电路,从一个门电路、一个芯片、一块电路板,到一个复杂系统都可看成一个实体。如果在设计时采用的是“自顶向下分层、划分模块”的设计方法,那么各层的设计模块都可看成一个实体。顶层的系统模块称为顶层实体,底层的设计模块称为底层实体。在用 VHDL 描述时,顶层的实体可将比它低的底层实体当作元件来调用。至于底层实体的具体结构和功能,在底层实体中描述。

(三)实体说明

实体说明是实体的一个必备组成部分,其功能是对实体与外部电路的接口进行描述,是实体的表层设计单元,它规定了实体的输入输出接口信号,是实体对外的一个通信界面。

实体说明常用的语句结构如下。

```
ENTITY 实体名 IS
  [GENERIC(类属参数说明语句);]
  [PORT(端口说明);]
END 实体名;
```

实体说明部分必须按照这一结构编写,编写时应注意以下几个问题。

(1)实体名可由设计者自己规定,可采用英文字母 A～Z、a～z,阿拉伯数字 0～9 或下划线符号“_”,但字符数不能超过 32 个,不能以数字或“_”开头。不能连续使用下划线符号(如“_ _”),也不能以“_”结束。

(2)中间方括号内的语句描述并非必需。后续内容规定相同。

(3)对于 VHDL 的编译器和综合器来说,程序字母的大小写是不加区分的,但为了便于阅读和分辨,建议将 VHDL 的表示符或基本语句的关键词以大写方式表示,而由设计者添加的内容以小写方式来表示。

【例 3-2】 试编写一个 *D* 触发器的实体说明。*D* 触发器的符号如图 3-2 所示。

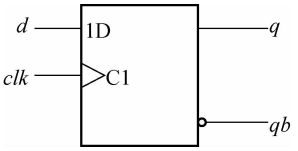


图 3-2 *D* 触发器的符号

```
ENTITY dff IS
  PORT(clk,d:IN BIT;
    q,qb:OUT BIT);
END dff;
```

1. 类属参数说明语句

类属参数说明语句(GENERIC)必须放在端口说明语句之前,类属参数的值可由实体外部提供。它通常用于设定元件内部电路的结构和规模,设计者可从外面通过重新设定类属参数的值而改变元件内部电路的结构和规模。

类属参数说明语句的一般书写格式如下。

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

其中的常数名由设计者定义,数据类型通常是 INTEGER(整数)或 TIME 等,设定值即为常数名所代表的值。但需要注意,综合器只支持整数数据类型。

【例 3-3】 试编写一个 12 位二进制计数器的实体说明。

```
ENTITY counter_x IS
  GENERIC(n:INTEGER:=12);
  PORT(clk:IN STD_LOGIC;
    q:OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END counter_x;
```

在例 3-3 中,首先定义了类属参数 n 的设定值为 12,在后续语句中,凡是用到 n 的地方均可用 12 代替。因此,例 3-3 的第 4 行语句也可写成

```
q:OUT STD_LOGIC_VECTOR(11 DOWNTO 0));
```

采用类属参数语句的优点是,当在某个实体内大量使用某个参数时,就可以把该参数定义成类属参数。当设计者需要改变该参数的值时,只需在类属参数语句中改写一次即可,从

而避免改写多处所带来的麻烦。对设计者来说，改变一个类属参数的值看似很容易，但从综合的结果来看，将大大地影响设计结果的硬件规模，由此可看出应用 VHDL 进行 EDA 设计的优越性。

2. 端口说明

端口说明(PORT)对设计实体与外部电路的接口通道进行了说明，其中包括对每一接口的输入输出模式和数据类型进行了定义。

端口说明的一般书写格式如下。

```
PORT(端口信号名,端口信号名:端口模式 数据类型;
                ⋮
     端口信号名,端口信号名:端口模式 数据类型);
```

1)端口信号名

端口信号名是设计者赋给每个对外通道的名字，它与原理图中元件符号上的引脚名相类似。对端口的命名可采用英文字母 A～Z、a～z，阿拉伯数字 0～9 或下划线符号“_”，但字符数不能超过 32 个，不能以数字或“_”开头，不能连续使用下划线符号，也不能以“_”结束。每个端口名在同一实体中必须是唯一的，不能有重复现象。

2)端口模式

端口模式用来说明端口上的数据流动方向。端口模式有以下几种。

(1)IN。定义的端口为输入端口，并规定为单向只读模式，可以通过此端口将外部的其他信号读入设计实体中。

(2)OUT。定义的端口为输出端口，并规定为单向输出模式，通过此端口可将信号从设计实体输出到外部。

(3)INOUT。定义的端口为输入输出双向端口，即实体既可以通过该端口输入外部信号，又可以通过该端口把实体内的信号输出到外部。

(4)BUFFER。定义的端口为具有数据读入功能的输出端口。它与 INOUT 的区别是，INOUT 是双向端口，既可以输入信号，也可以输出信号；而 BUFFER 也可以输出实体信号到外部，但作为输入时，信号不是由外部输入，而是在实体内由输出信号反馈得到，即 BUFFER 模式的端口在信号输出实体的同时，也可以被实体本身读入。

以上各种端口模式的区别可参考图 3-3。

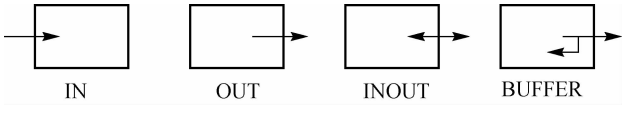


图 3-3　端口模式说明

3)数据类型

数据类型是指端口信号的取值类型。常见的数据类型有以下几种。

(1)BIT。二进制位类型，其取值是一个 1 位的二进制数，只能是 0 或 1。

(2)BIT_VECTOR。位向量数据类型，其取值是一组二进制数，常用来描述总线等端

口。例如，data：IN BIT_VECTOR(3 DOWNTO 0)定义了一个具有 4 位位宽的输入数据总线端口。

(3)STD_LOGIC。工业标准逻辑类型，取值有 0、1、Z(高阻)、X(未知)等 9 种，该数据类型由 IEEE 库中的 STD_LOGIC_1164 程序包定义。

(4)STD_LOGIC_VECTOR。工业标准逻辑向量类型，是多位 STD_LOGIC 数据类型的组合，也常用来描述总线等端口。

(5)INTEGER。整数类型，可用作循环的指针或常数，通常不用作 I/O 信号。

(6)BOOLEAN。布尔类型，取值有 FALSE(假)、TRUE(真)两种。

(四)结构体

对一个设计实体而言，实体说明部分描述的是实体的对外接口，并不考虑实体内部的具体细节。实体的内部结构与行为由结构体来描述。

结构体是实体的一个重要组成部分，主要用来描述实体内的硬件结构、元件之间的连接、实体所完成的逻辑功能以及数据的传输和变换等方面的内容。一个实体可以有一个或多个结构体，每个结构体可分别描述该实体功能的不同实现方案。

1. 结构体的一般书写格式

结构体的一般书写格式如下。

```
ARCHITECTURE 结构体名 OF 实体名 IS
[说明语句]
BEGIN
  [功能描述语句]
END 结构体名;
```

在书写格式上需要注意，结构体中的“实体名”必须与实体说明中的“实体名”相一致，而“结构体名”可由设计者自己选择。但当一个实体具有多个结构体时，各结构体名不可重复。说明语句必须放在关键词 ARCHITECTURE 和 BEGIN 之间，结构体必须以“END 结构体名;”作为结束句。

2. 结构体说明语句

结构体中的说明语句是对结构体的功能描述语句中要用到的信号(SIGNAL)、数据类型(TYPE)、常数(CONSTANT)、元件(COMPONENT)、函数(FUNCTION)和过程(PROCEDURE)等加以说明。需要注意的是，在一个结构体中说明和定义的数据类型、常数、元件、函数和过程只能用于该结构体中。如果希望这些定义也能用于其他实体或结构体中，需要将其作为程序包来处理。

3. 功能描述语句

功能描述语句含有 5 种不同类型且以并行方式工作的语句结构，这可以看成是结构体的 5 个子结构，而在每一子结构的内部也可能含有顺序运行的逻辑描述语句。这就是说，虽然 5 个子结构本身是并行语句，但每个子结构内所包含的语句并不一定是并行

语句。

图 3-4 给出了结构体的一般构造图。有关常用的子结构基本组成和语句描述将在后续内容中介绍。

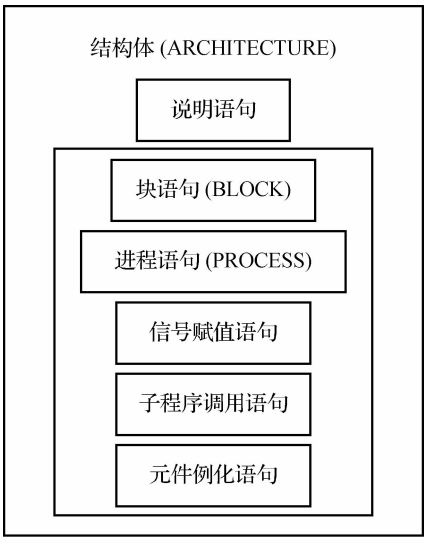


图 3-4　结构体的一般构造图

【例 3-4】 试用 VHDL 描述图 3-5 所示电路原理图。

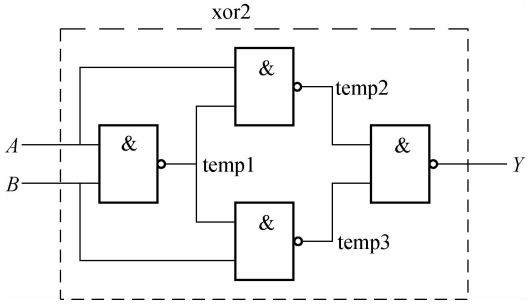


图 3-5　例 3-4 电路原理图

描述如下。

```
--实体说明
ENTITY xor2 IS          --实体名为 xor2
  PORT(A,B:IN BIT;  --A、B 为 2 个输入引脚,数据类型为 BIT
       Y:OUT BIT); --Y 为输出引脚,数据类型为 BIT
END xor2;
```

```
--结构体
ARCHITECTURE a OF xor2 IS          --结构体名为 a
SIGNAL temp1,temp2,temp3:BIT;      --类属参数说明语句。定义 3 个内部信号。数据
                                     类型为 BIT
BEGIN                              --开始结构体描述
  temp1<=A NAND B;                 --A 和 B 与非运算,结果赋给 temp1
  temp2<=A NAND temp1;             --A 和 temp1 与非运算,结果赋给 temp2
  temp3<=B NAND temp1;             --B 和 temp1 与非运算,结果赋给 temp3
  Y<=temp2 NAND temp3;             --temp2 和 temp3 与非运算,结果赋给 Y
END a;                             --结束结构体描述
```

在上述例子中,temp1、temp2、temp3 和 Y 这四条赋值语句之间是并行运行的关系,即它们的执行是同步的。只要某个信号发生变化,都会立即引起相应的语句被执行,产生相应的输出,而不管该语句书写的先后顺序。这一点和传统的程序设计语言的顺序执行情况是不同的,但它和硬件电路的工作情况相一致,这种并行的执行方式是 VHDL 与传统程序语言最显著的区别。

(五)库和程序包

对于一个设计实体而言,除了包括前面介绍的实体说明和结构体外,还经常要用到库和程序包。

1. 库

在利用 VHDL 进行工程设计时,为了提高设计效率以及使设计遵循某些统一的语言标准或数据格式,有必要将一些有用的信息汇集在一个或几个库(LIBRARY)中,以供 VHDL 调用。这些信息可以是预先定义好的数据类型,也可以是已编译过的设计单元(包括实体说明、结构体、程序包等)。因此,可以把库看成是用于存放预先完成的数据类型和源设计单元的仓库。如果在一项 VHDL 设计中要用到库中的信息,就必须在这项设计中预先打开这个库。在综合过程中,每当综合器遇到库语言,就可以将库指定的源文件读入,并参与综合。

库的语句书写格式如下。

```
LIBRARY 库名;
```

常用的库有 IEEE 库、STD 库和 WORK 库。

1)IEEE 库

IEEE 库是 VHDL 设计中最常用的库,它包含了 IEEE 标准的程序包和其他一些支持工业标准的程序包。IEEE 库中主要包括 STD_LOGIC_1164、STD_LOGIC_UNSIGNED、STD_LOGIC_SIGNED 等标准程序包,其中的 STD_LOGIC_1164 是最重要和最常用的程序包。使用 IEEE 库时必须先用语句“LIBRARY IEEE;”声明。

2)STD 库

STD 库是 VHDL 的标准库。在利用 VHDL 进行设计时,可以自动使用这个库,而不必像 IEEE 库那样须首先声明。因此,类似“LIBRARY STD;”的语句是不必要的。

3)WORK 库

WORK 库是 VHDL 设计的现行工作库,用于存放用户设计和定义的一些设计单元与程序包。因此,它自动满足 VHDL 标准,在实际调用时不必预先声明,所以像“LIBRARY WORK;”的语句也是不必要的。

在一个库中往往有很多可使用的资源,这些资源通常存放在不同的程序包中。因此,在使用这些资源时,除了声明所在的库外,还要说明使用的是该库中的哪个程序包。说明书写格式如下。

```
LIBRARY 库名;
USE 库名.程序包名.使用的范围;
```

例如,要使用 IEEE 库的 STD_LOGIC_1164 程序包中的所有内容,可书写如下。

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

2.程序包

在实体中定义的各种数据类型、数据对象等信息只能局限在该实体内调用,其他实体是不可用的。为了使这些信息能够被其他设计实体所使用,VHDL 提供了程序包机制。只要把这些信息收集在一个 VHDL 程序包中并入库,这些信息就成为公共信息,其他设计实体就可以使用这些公共信息。

程序包一般由程序包首和程序包体两部分组成,其语句书写格式如下。

```
--程序包首
PACKAGE 程序包名 IS
[说明语句]
END 程序包名;
--程序包体
PACKAGE BODY 程序包名 IS
[程序包体说明语句]
END 程序包名;
```

为了方便设计,VHDL 提供了一些标准的程序包,如 STANDARD,它定义了若干数据类型、子类型和函数。另外,在 IEEE 库中还有一些常用的程序包,这些程序包已编译过,在使用时,只需声明库和使用的程序包即可。

常用程序包的内容见表 3-1。

表 3-1　常用程序包的内容

库名	程序包名	包中预定义的内容
STD	STANDARD	VHDL 类型，如 BIT、BIT_VECTOR 等
IEEE	STD_LOGIC_1164	定义了 STD_LOGIC、STD_LOGIC_VECTOR 等数据类型
IEEE	STD_LOGIC_UNSIGNED	定义了基于 STD_LOGIC 与 STD_LOGIC_VECTOR 类型上的无符号的算术运算
IEEE	STD_LOGIC_SIGNED	定义了基于 STD_LOGIC 与 STD_LOGIC_VECTOR 类型上的有符号的算术运算
IEEE	STD_LOGIC_ARITH	定义了有符号与无符号类型及基于这些类型的算术运算

(六)VHDL 的常用语句

并行语句和顺序语句是程序设计中的两类基本描述语句。在逻辑系统的设计中，这些语句能够从多方面完整地描述数字系统的硬件结构和基本逻辑功能，其中包括通信的方式、信号的赋值、多层次的元件例化及系统的行为等。

1. 并行语句

相对于一般的软件描述语言，并行语句结构是最具 VHDL 特色的，是与一般软件程序最大的区别所在。在 VHDL 中，并行语句有多种语句格式，各种并行语句在结构体中的执行都是同步进行的，或者说是并行运行的，其执行的方式与书写的顺序无关。这种并行性是由硬件本身的并行性决定的，即一旦电路接通电源，它的各部分就会按照事先设计好的方案同时工作。并行语句在执行时，各并行语句之间可以有信息来往，也可以互为独立、互不相关。另外，每一并行语句内部的语句可以有两种不同的运行方式，即并行执行方式（如块语句）和顺序执行方式（如进程语句）。

图 3-6 给出了常用的并行语句及在同一结构体中各种并行语句运行的示意图。这些语句不必同时使用，每一语句模块都可以独立异步运行，模块之间并行运行，并通过信号交换信息。

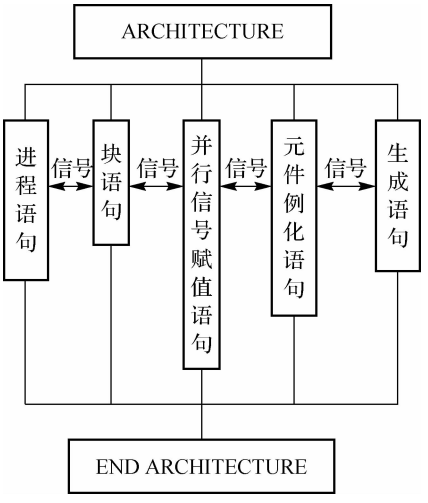


图 3-6　结构体中的并行语句模块

1)进程语句

进程语句(PROCESS)是 VHDL 程序中使用最频繁和最能体现 VHDL 特点的一种语句,其原因是它的并行和顺序行为的双重性,以及其行为描述风格的特殊性。一个结构体中可以包括多条进程语句,各进程语句之间或各进程语句与其他并行语句之间的通信是依靠信号(SIGNAL)来传递的。

进程语句通常由一段程序构成。虽然进程语句本身是并行语句,但其内部的语句是由顺序语句构成的。因此,在编写进程语句的内部语句时要特别注意各语句书写的先后顺序,不同的语句书写顺序将导致不同的硬件设计结果。

进程语句的书写格式如下。

```
[标号:]PROCESS(信号 1,信号 2,…)
  [进程说明语句]
  BEGIN
    [顺序语句]
  END PROCESS [标号];
```

书写格式说明如下。

(1)标号。标号是进程的名称,是为了区别同一结构体中的不同进程而设置的,但它并不是必需的。

(2)敏感信号。小括号中的信号是进程的敏感信号,进程的敏感信号可以由一个或多个信号组成,它是进程内所用到的一些信号。当任何一个敏感信号发生变化时,该进程才能被执行(或激活),其内部的顺序语句才能被执行一遍。当进程最后一条语句执行完成后,将返回到进程的第一条语句,以等待下一次敏感信号的变化。

(3)进程说明语句。进程说明语句用于对该进程内所用到的局部数据进行定义(如常数、变量、信号等),这里所说的局部数据就是该数据只对本进程有效,只能用于本进程内,不可用于其他进程或并行语句中。若想把局部数据带出进程,则必须把该数据传递给全局信号(在结构体说明语句或实体说明部分所定义的信号),由全局信号带出该进程并可为其他进程或并行语句所使用。

【例 3-5】

```
ARCHITECTURE a OF states_mach IS
BEGIN
  P1:PROCESS(clk)
     BEGIN
        IF (clk'EVENT AND clk='1')THEN
          current_state<=next_state;
        END IF;
```

```
        END PROCESS;
    ⋮
END a;
```

本例中，P1 为进程标号，时钟 clk 为敏感信号。每当 clk 发生一次变化时，BEGIN 和 END PROCESS 之间的顺序语句就会运行一次。由于时钟 clk 变化包括上升沿和下降沿，为了准确描述，在顺序语句中采用一个条件判断语句“IF(clk'EVENT AND clk='1') THEN”来判断 clk 的上升沿。若要判断 clk 的下降沿，可用“IF(clk'EVENT AND clk='0') THEN”语句。

【例 3-6】

```
ARCHITECTURE one OF gate IS
SIGNAL m:BIT;                    --m 为全局信号
BEGIN
  P1:PROCESS(a,b)                --进程 P1，敏感信号为 a 和 b
     VARIABLE n:BIT;             --局部变量
     BEGIN
        n<=a NAND b;
        m<=n;                    --将局部变量传递给全局信号
     END PROCESS;
  P2:PROCESS(c,m)                --进程 P2，敏感信号为 c 和 m
     BEGIN
     y<=c NAND m;                --该进程利用了 P1 进程中的 n 值
     END PROCESS;
   ⋮
END one;
```

本例中，m 为全局信号，它可应用于进程 P1 和 P2 中。在进程 P1 的说明语句中定义了局部变量 n，该变量只适用于本进程，若要把其值带出 P1，必须把 n 赋给全局信号 m。进程 P1 和 P2 为并行关系。

2)块语句

使用过 Protel 软件的读者都知道，画一个较大的电路原理图时，通常把它分成几个子模块来绘制，而其中的每个子模块都可以是一个具体的电路原理图；倘若子模块仍很大，还可以往下再分子模块。VHDL 中的块语句(BLOCK)结构的应用类似于此。

事实上，块语句的应用只是一种将结构体中的并行描述语句进行组合的方法，其目的是改善并行语句的可读性，一般应用于较复杂的 VHDL 程序中。块语句对复杂并行结构的划分仅限于形式上的，从综合的角度来看，它不会改变电路的结构和逻辑功能。

块语句的书写格式如下。

```
块标号名:BLOCK[块保护表达式]
    [接口说明语句]
    [类属参数说明语句]
    BEGIN
      并行语句
    END BLOCK 块标号名;
```

书写格式说明如下。

(1)在关键词 BLOCK 的前面必须设置一个块标号,并在结尾语句“END BLOCK”右侧也写上此标号(此处的块标号不是必需的)。

(2)“接口说明语句”和“类属参数说明语句”类似于实体的定义部分,它可包含由关键词 PORT、GENERIC、PORT MAP 和 GENERIC MAP 引导的接口说明语句,对 BLOCK 的接口设置及与外界信号的连接情况加以说明。“接口说明语句”和“类属参数说明语句”的适用范围仅限于当前 BLOCK 内部,对于块外来说是不透明的,即不能适合于外部环境或由外部环境所调用。

(3)“并行语句”可包含任何并行语句结构。既然 BLOCK 语句本身属于并行语句,当然也可以作为这里的并行语句,即可以实现 BLOCK 语句的嵌套。

【例 3-7】 利用 BLOCK 语句描述图 2-3 所示的全加器。

描述如下。

```
ENTITY add IS
  PORT(ain,bin,cin:IN BIT;
    sum,cout:OUT BIT);
END add;
ARCHITECTURE a OF add IS
SIGNAL so1,co1,co2:BIT;          --全局信号
BEGIN
P1:BLOCK
   SIGNAL a,b:BIT;               --局部信号,只适用于块 P1
   BEGIN
--半加器 1 的描述语句
  END BLOCK P1;
```

```
P2:BLOCK
   BEGIN
--半加器 2 的描述语句
  END BLOCK P2;
P3:BLOCK
   BEGIN
--将 2 个半加器组合成全加器的连接部分描述语句
   END BLOCK P3;
END a;
```

【例 3-8】 本例是一个 BLOCK 语句嵌套的例子。

```
…
B1:BLOCK
  SIGNAL s1:BIT;                --对块 B1、B2、B3 都适用
  BEGIN
    s1<=a AND b;
  B2:BLOCK
    SIGNAL s2:BIT;              --对块 B2、B3 适用
    BEGIN
      s2<=c AND d;
    B3:BLOCK
      BEGIN
        z<=s2;
    END BLOCK B3;
  END BLOCK B2;
  y<=s1;
END BLOCK B1;
…
```

3)并行信号赋值语句

并行信号赋值语句有三种形式:简单信号赋值语句、选择信号赋值语句和条件信号赋值语句。这三种信号赋值语句的共同特点是赋值目标必须都是信号,所有赋值语句与其他并行语句一样,在结构体中的执行是同时发生的。

(1)简单信号赋值语句。简单信号赋值语句是并行语句结构的最基本单元,它的语句书写格式如下。

```
赋值目标<=表达式;
```

应用该类赋值语句时一定要注意，赋值目标的数据对象必须是信号，赋值运算符“<=”两边的数据类型必须一致。

【例 3-9】 本例的结构体中描述了 3 个基本的逻辑门。

```
ENTITY gate IS
  PORT(a,b:IN BIT;
       y1,y2,y3:OUT BIT);
END gate;
ARCHITECTURE one OF gate IS
BEGIN
   y1<=a AND b;     --与门
   y2<=a OR b;      --或门
   y3<=NOT a;       --非门
END one;
```

(2)选择信号赋值语句。选择信号赋值语句的书写格式如下。

```
WITH 选择表达式 SELECT
    赋值目标信号<=表达式 WHEN 选择值,
                  表达式 WHEN 选择值,
                  ⋮
                  表达式 WHEN 选择值;
```

书写格式说明如下。

①每当“选择表达式”的值发生变化时，将启动此语句对各子句的“选择值”进行测试对比。当发现有满足条件的子句时，就将此子句中的“表达式”值赋给“赋值目标信号”。

②每条子句应以“,”结束，最后一条子句以“;”结束。

③“选择值”不能有重复，且“选择值”应包含“选择表达式”的所有取值，不允许存在选择值涵盖不全的情况。

【例 3-10】 用选择信号赋值语句设计一个 4 选 1 数据选择器，其符号如图 3-7 所示。其中，SEL[1..0]为 2 位地址输入端，A、B、C、D 为数据输入端，Y 为数据输出端。当 SEL[1..0]为“00”时，Y＝A；当 SEL[1..0]为“01”时，Y＝B；当 SEL[1..0]为“10”时，Y＝C；当 SEL[1..0]为“11”时，Y＝D。

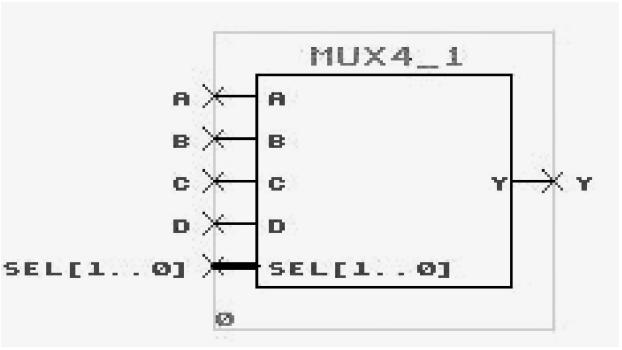


图 3-7　4 选 1 数据选择器符号

描述如下。

```
ENTITY mux4_1 IS
   PORT(a,b,c,d:IN BIT;
        sel:IN BIT_VECTOR(1 DOWNTO 0);
        y:OUT BIT);
END mux4_1;
ARCHITECTURE one OF mux4_1 IS
BEGIN
   WITH sel SELECT
      y<=a WHEN "00",
           b WHEN "01",
           c WHEN "10",
           d WHEN "11";
END one;
```

(3)条件信号赋值语句。条件信号赋值语句的书写格式如下。

```
赋值目标信号<=表达式 WHEN 赋值条件 ELSE
              表达式 WHEN 赋值条件 ELSE
               ⋮
              表达式;
```

书写格式说明如下。

①条件信号赋值语句与选择信号赋值语句的最大区别在于后者的各个“选择值”之间处于同一优先级,而前者的各个“赋值条件”具有优先顺序,优先级由高到低的顺序与语句书写顺序一致。

②当某个“赋值条件”得到满足(其值为“真”)时,立即将该条件 WHEN 前的“表达式”值

赋给“赋值目标信号”；当几个“赋值条件”都得到满足时，将优先级高的那个条件 WHEN 前的“表达式”值赋给“赋值目标信号”；当所有的“赋值条件”都得不到满足时，将最后一个 ELSE 关键词后的“表达式”值赋给“赋值目标信号”。

③每行语句后面没有标点符号，最后一行“表达式”用“；”结束。

【例 3-11】 用条件信号赋值语句设计例 3-10 中的 4 选 1 数据选择器。

描述如下。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux4_1 IS
   PORT(a,b,c,d:IN BIT;
        sel:IN BIT_VECTOR(1 DOWNTO 0);
        y:OUT BIT);
END mux4_1;
ARCHITECTURE one OF mux4_1 IS
BEGIN
   y<=a WHEN sel="00" ELSE
      b WHEN sel="01" ELSE
      c WHEN sel="10" ELSE
      d;
END one;
```

通常，当某个设计实体既可以用选择信号赋值语句描述，也可以用条件信号赋值语句描述时，应尽量采用选择信号赋值语句。

4)元件例化语句

元件例化语句引入的是一种连接关系，即将预先设计好的设计实体定义成一个元件，然后利用例化语句将此元件与当前的设计实体中的指定端口相连接，从而为当前设计实体引入了一个低一级的设计层次。也可以这样来理解例化语句，当前设计实体相当于一个较大的电路系统，预先设计好的设计实体相当于一个要插在这个电路系统板上的芯片，而当前设计实体中的例化语句则相当于这块电路板上准备接收此芯片的一个插座。

元件例化语句通常由元件声明和元件例化两部分组成，语句的书写格式如下。

```
--元件声明部分
COMPONENT 元件名
   GENERIC(参数表);
   PORT(端口信息);
```