

## 第四章 EDA 技术与可编程 ASIC 的设计实现

半个世纪以来，以微电子技术为核心的信息技术革命以迅雷不及掩耳之势推动着世界经济和社会发展。在一块只有指甲般大小的硅片上，竟能容纳数以亿计的电子器件，形成能完成不同功能的电路系统。由于数字技术在处理和传输信息方面的种种优势，数字技术与数字集成电路的广泛应用成了信息时代重要标志。

早期数字系统采用传统的搭积木式的方法进行设计，即用器件搭成电路板，再由电路板构成电子系统。随着半导体技术、集成技术和计算机技术的发展，特别是 EDA 技术的发展和普及给电子系统的设计带来了革命性的变化，也就是利用 EDA 工具，采用可编程器件，通过设计芯片来实现系统功能。本章将以 Altera、Xilinx 两大主流产品为主线，介绍这种只要拥有一台电脑、一套可编程器件芯片开发系统和配套的 EDA 软件系统即可实现数字系统设计的方法。

### 4.1 集成电路与 ASIC

集成电路经过半个世纪的演变、发展，目前品种已达 **5 万种**，年产量数以亿块计。关于集成电路的各种**新“名词”**虽不断花样翻新，也跟不上电子市场急剧变化和更新换代的需求。

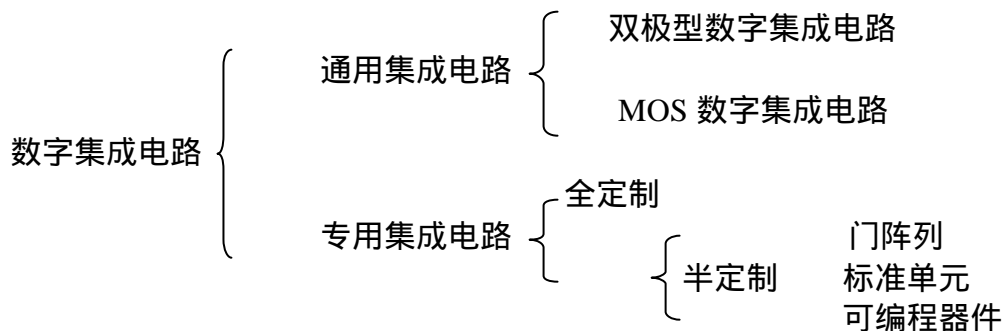
#### 4.1.1 集成电路及其分类

集成电路按集成度可分为小规模、中规模、大规模、超大规模(军用的称超高速)、极大规模和巨大规模集成电路。集成度是标志集成电路的一个重要指标，一般是指在一定尺寸的芯片上能做出多少个晶体管。通常将集成度少于 100 个元件的集成块称为小规模集成电路(简称 SSI)；把集成度在 100~1000 个元件的称为中规模集成电路(简称 MSI)；以此类推。

集成电路按制作工艺可分为：**膜集成电路**、**半导体集成电路**和**混合型集成电路**。其中，膜集成电路又分为薄膜集成电路和厚膜集成电路两类，薄膜集成电路制作方法主要采用淀积方法，如真空蒸发、溅射、电解氧化等方法，把需要的各种材料覆盖在陶瓷或玻璃片上，然后用光刻的方法获得电路；厚膜集成电路主要用丝网漏印的方法，像印刷画报那样把电路印制在陶瓷上。半导体集成电路是在半导体材料的晶圆片制作出电路。混合型集成电路是采用薄膜集成电路和半导体集成电路制作工艺联合制作出电路。

集成电路按功能可分为**数字集成电路**、**模拟集成电路**(线性集成电路和非线性集成电路)、**微波集成电路**和**专用集成电路(ASIC)**。所谓的数字集成电路就是传递、加工、处理数字信号的集成电路；数字集成电路可分为通用数字集成电路和专用集成电路；通用数字集成电路是指那些用户众多、使用领域广泛、标准型的电路，专用集成电路是指为特定的用户、某种专门或特别的用途而设计的电路。

具体分类如下。



其中，通用数字集成电路由于采用的晶体管不同，可分为**双极型集成电路**和**场效应集成电路**两种。这两大系列中主要由 TTL 和 CMOS 为代表。高阈值晶体管逻辑电路(HIL)、发射极耦合逻辑电路(ECL)、集成注入逻辑电路(IIL)、N 沟道场效应管逻辑电路(NMOS)和 P 沟道场效应管逻辑电路(PMOS)等系列，使用较少。反映数字集成电路的现状和应用

水平的是存储器、微处理器及微控制器和专用集成电路，存储器是典型的数字集成电路，也一直是集成电路的主要产品，其技术发展代表着集成电路发展水平。另外，数字集成电路的生涯是和计算机的命运紧紧拴在一起的，它应计算机的需要而诞生，并随电子技术的不断进展而发展，计算机的核心是微处理器和微控制器。数字集成电路发展到数字系统也与其和计算机、通信、网络等逐渐融合密不可分。

**模拟集成电路**是指能对电压、电流等模拟量进行放大与转换的集成电路，如果输入信号与输出信号呈线性关系的电路称为线性集成电路，输入信号与输出信号不成线性关系的电路称为非线性集成电路；**微波集成电路**是近些年迅速发展的高频集成电路，由于工作频率高(大于 300MHz)，导致其电路结构、元件类型、材料、工艺途径以及应用范围都大不相同，形成了单独的一大类型集成电路。

#### 4.1.2 ASIC 及其分类

ASIC(Application Specific Integrated Circuits)专用集成电路，相对于标准逻辑、通用存储器、通用微处理器等电路，它面向**专门用途**，通常根据**某一用户的特定要求**，能以低研制成本、短周期交货的**全定制或半定制**集成电路。

ASIC 从 20 世纪 60 年代提出概念，到 20 世纪 80 年代后期随半导体集成电路工艺技术、支持技术、设计技术和测试评价技术的发展，集成度的大大提高，电子整机、电子系统高速更新换代，从而得到充分发展。以其设计专用性、成本低、开发周期短、工具先进和对提高电子系统性能、可靠性、保密性、提高工作速度、降低功耗、减少芯片体积和重量等各方面的优势，很快形成了用 ASIC 取代中小规模集成电路来组成电子系统或整机的热潮，目前在集成电路市场中的占有率已达 1/3。

按照设计方法不同，ASIC 分为全定制和半定制。全定制是基于晶体管的设计方法，针对要求得到最高速、最低功耗和最省面积的芯片，它必须从晶体管的版图尺寸位置及连线开始亲自设计。通常设计成本高，周期长，只适用于性能要求很高或批量很大的芯片。**半定制则是一种约束性设计方法**，约束的主要目的是简化设计、缩短设计周期以及提

高芯片成品率，这时对芯片面积或性能做出牺牲，**尽可能采用已有的规则结构的版图**，用最短的时间设计出芯片，占领市场后再予以改进。

半定制方法分为**门阵列法、标准单元法和可编程 ASIC**。门阵列法又称母片法，是在半成品母片上将已有的规则单元相互连接实现电路要求，并有较高自动化的设计软件；门阵列法成本低、周期短，但门的利用率低，芯片性能不高。标准单元法是以精心设计的标准单元库为基础，调用库单元版图，利用自动布局布线完成电路到版图一一对应的设计，它的成本周期性能指标都较高。前几种 ASIC 方法都必须到集成电路厂家去加工流片才能完成，设计制造周期较长，而且一旦出错，需要重新修改设计和流片，周期成本必然大大增加。

可编程逻辑器件是一种已完成全部工艺制造可直接从市场购得的产品，用户**只需对它编程**就能实现电路功能，设计人员在实验室即可设计和制造出芯片，而且可以反复编程使硬件的功能像软件的功能一样通过编程来修改。可编程 ASIC 发展到现在，芯片上**包含的资源**越来越丰富，可实现的功能越来越强，已成为当今电子系统设计的重要手段。

目前，在电子系统开发阶段的硬件验证过程中，**一般**都采用可编程 ASIC，以期尽快开发产品，迅速占领市场。可编程 ASIC 尤其适合实验室中的科研和教学，不仅投入低，开发时间短，而且灵活性大，能够有效提高学生的设计能力。可编程逻辑器件的分类如图 4-1 所示。

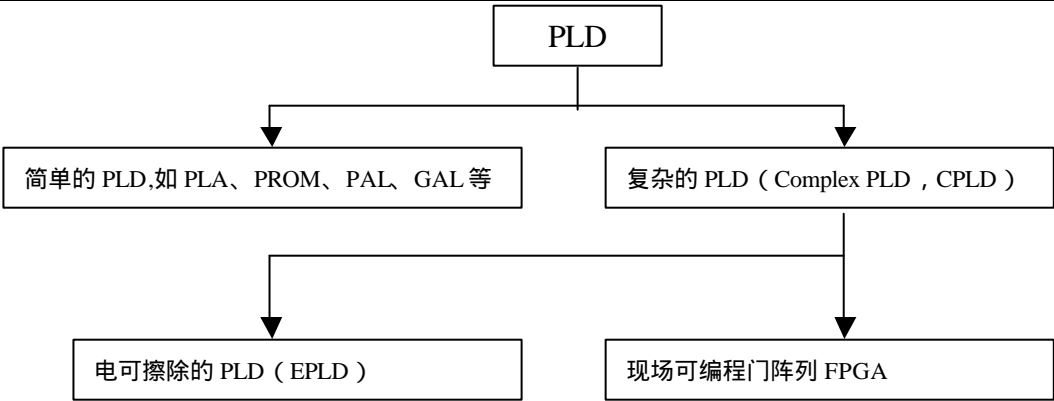


图 4-1 可编程逻辑器件分类示意图

可编程 ASIC 按复杂程度大致分为可编程逻辑器件(PLD, Programmable Logic Device), 复杂可编程逻辑器件(CPLD, Complex Programmable Logic Device), 现场可编程门阵列(FPGA, Field Programmable Gate Array)。PLD 由“与阵列”和“或阵列”组成, 可用来实现任何“以积之和”形式表示的各种布尔逻辑函数; CPLD 在 PLD 的基础上增加密度, 扩充功能, 以实现复杂逻辑; FPGA 则具有类似于半定制门阵列的通用结构, 即由逻辑功能块排列成阵列组成, 并由可编程的互连资源连接这些逻辑功能块来实现所需的设计。进入 20 世纪 90 年代后, 复杂可编程逻辑器件 CPLD 已经成为可编程 ASIC 的主流产品, 在 ASIC 市场占有了较大的份额。一般具有可重编程特性, 实现的**工艺**有 **EPROM、Flash EPROM 和 E<sup>2</sup>PROM**; 采用连续互连方式, 能够方便地预测设计时序同时保证 CPLD 的高速性能。

全定制电路芯片完全由设计者自己开发设计, 集成度高, 可以针对所设计的电子系统进行优化, 使性能达到最优, 但是其芯片设计、制作的成本都比较高, 并且设计开发的周期较长, 所以全定制电路芯片一般只用于大规模生产中, 在电子系统的设计开发过程中不使用全定制电路芯片。对于半定制电路芯片, 设计者不需从头设计, 只需将所需功能用开发系统“写”入半定制电路芯片, 如 EPLD、CPLD、FPGA 等。因而在性能上虽然不能达到最优, 但是设计制作的成本较低, 开发周期短, 比较适合于在电子系统的设计开发中使用。

一般在进行复杂数字系统全定制设计时，完成全定制电路芯片设计后，还是要进行 **FPGA 的仿真测试**，这样才能最大程度上保证电路的可靠性。因为电路设计完成后，仿真结果通过，并不等于说版图就能够一定通过(即和电路仿真结果一致)，所以须用 FPGA 验证流片。

## 4.2 可编程 ASIC 及其发展

可编程 ASIC 是利用 EDA 工具、可编程器件来实现数字系统功能。这种设计方法能够由设计者定义器件的内部逻辑和管脚，原来由电路板完成的大部分工作放在芯片的设计中进行，不仅可以通过芯片设计实现多种数字逻辑系统功能，而且由于管脚定义的灵活性，大大减轻电路图、电路板设计的工作量和难度，有效增强设计灵活性，提高工作效率，同时减小芯片数量，缩小系统体积，降低功耗，提高系统性能和可靠性。传统的“固定功能集成块+连线”的设计方法正逐步退出历史舞台，而基于芯片的可编程 ASIC 正在成为现代数字系统设计的主流，当今数字系统设计已经离不开可编程逻辑器件和 EDA 设计工具。

### 4.2.1 可编程逻辑器件的发展历程

随着微电子技术的发展，设计与制造集成电路的任务已不完全由半导体厂承担，系统设计师们更愿意通过 Fabless 模式来自行设计 ASIC 芯片，而且希望 ASIC 的设计周期尽可能短，最好在实验室就能设计出合格的 ASIC 芯片并立即投入使用，因而出现了现场可编程逻辑器件(FPLD)，其中应用最广泛的当属现场可编程门阵列(FPGA)和复杂可编程逻辑器件(CPLD)。

早期的可编程逻辑器件只有可编程只读存储器(PROM)、紫外线可擦除只读存储器(EPROM)和电可擦除只读存储器(E<sup>2</sup>PROM)三种，由于结构限制，只能完成简单的数字逻辑。其后，出现了一类结构上较复杂的可编程芯片，即可编程逻辑器件(PLD)，它能够完成各种数字逻辑功能。主要产品有可编程逻辑阵列(PLA)和通用逻辑阵列(GAL)。PLA 器件既有现场可编程的，也有掩膜可编程的。在 PLA 基础上，又发展了 GAL，它采用 E<sup>2</sup>PROM 工艺，实现了电可擦写，其输出结构

是可编程的逻辑宏单元，因而在当时用它做设计具有很强的灵活性。

这些早期的 PLD 器件的共同特点就是可以实现速度特性较好的逻辑功能，但其过于简单的结构也使它们只能实现规模较小的电路。为了弥补这一缺陷，20 世纪 80 年代中期，Altera 公司和 Xilinx 公司分别推出了类似于 PLA 结构的扩展型 EPLD (CPLD 的前身) 和类似于标准门阵列的 FPGA。

## 4.2.2 FPGA 和 CPLD 的比较

FPGA 和 CPLD 是半定制芯片的杰出代表。它们兼容了 PLD 和通用门阵列的优点，可实现较大规模的电路，编程也很灵活。与门阵列等其他 ASIC 相比，它们的规模随着集成电路的发展越来越大、并且可反复擦写，适合正向设计，对知识产权的保护也很有利。设计人员用它们做设计的开发周期更短，研发费用低，并且不需要具备专门的集成电路深层次的知识。

### 1. FPGA 与 ASSP 和 ASIC 竞争

可编程逻辑器件替代专用集成电路和专用标准电路(ASSP, Application specific standard products)的趋势主要是由 FPGA 的优点造成的，这些器件在逻辑密度、性能和特性上有惊人的增长，而同时在元件成本上有显著的下降。不久之前，FPGA 还是速度慢、密度小和价格贵的器件，目前，高容量的 FPGA 可以达到每提供 10 万系统门和运行在 100MHz 以上只需不到 10 美元。

(1)FPGA 的介入。设计服务者、知识产权 (IP) 产品开发者和代工厂 (Foundry) 正在使自身更能满足设计者的需求，随着深亚微米 ASIC 的掩膜版加工费用达到 50 万美元以上，设计者也不得不重新评估定制的固定逻辑的器件和其能够承受的一次性工程费用 (NRE)。

许多产品工程师发现 ASIC 只在数量或收入大到足以证明前期的投资值得时才适合，确实，PLD 已经明显地占领了以前由门阵列销售商专有的设计领域。同时，ASSP 的市场发展和变迁为 FPGA 与其在相应领域的竞争开辟了新的机遇。

(2)起相同作用的场合。与以前不同，FPGA 供应商已经采用了最先进的半导体工艺制造技术，现在，器件已由流行的 0.13 $\mu\text{m}$  工艺制造，而 Virtex-II Pro 等最新的 FPGA 还利用最新的铜互连工艺来进一步提高性能。事实上，FPGA 已经成为独立的代工厂的工艺驱动者，代替了存储器所起的作用。工艺技术的快速转换已经允许 FPGA 供应商提供焊盘受限的可编程逻辑方案，硅片成本的差距已经接近到如下程度：整个 FPGA 的成本可以在按每个引脚来计算的基础上与 ASIC 和 ASSP 进行竞争。

利用最前沿的工艺技术已经使 FPGA 供应商在每个晶圆上生产出更多的芯片，提供更多的逻辑和增加更好的性能。供应商也可以提供对片内和片外时钟管理的延时锁定环，丰富的片内 RAM 块，支持广泛种类的单端和差分 I/O 标准等特性。这些进展使 FPGA 和 ASSP 之间的差距大大地缩小。

由于利用知识产权，FPGA 供应商能提供 32 位 33MHz 的 PCI 主/从控制器，而价格只有同等功能 ASSP 的一半，伴随这些特性，近年来 FPGA 供应商推到市场的 64 位 66MHz PCI 控制器比大多数芯片组要早，价格要低。

PCI 方案的可编程特性进一步证明 FPGA 的价值，设计者可以选择利用 FPGA 供应商出售的 PCI 控制器，或者将它定制化来更好地适应其设计要求。

这个灵活性是可编程逻辑进军 ASSP 市场的资本，一个 ASSP 的主要价值是以相对低的成本使产品更快地投入市场。但是，ASSP 不允许产品定制化和个性化，例如，用户从 ASSP 销售商处选择标准的 PCI 器件，常常利用额外的 FPGA 定制化他们的产品。相反，FPGA/PCI 核的结合是一个单芯片的方案。

(3)有线的应用。在有线通信领域，可编程逻辑可以适应经常变化的技术条件。虽然，标准的 ASSP 供应商企图推出数据通信、通信和消费市场的方案，但是经常变化的标准，使他们存在服务上的问题。例如，互联网用户增加带宽的要求为数字用户环（DSL）方案创造了巨大的市场，但是，缺少一个统一的标准已经成为推迟采用 DSL 服务的关键因素。

DSL 采用的技术范围反映了要解决的问题的困难程度，至少七种不同的技术已经被不同的公司采用，但是，哪一个可能成为优胜者仍然不清楚。

冲突的技术条件和缺少明确的方向形成对可编程 ASSP 方案的需求，这些冲突中有一些可能永远得不到解决。由于不可能和成本禁止去符合所有的技术条件，ASSP 供应商冒险用只满足其中一个条件取得了成功，即可编程 ASSP 方案。

(4)从 ASSP 到 FPGA。

在 ASSP 市场的竞争是激烈的，对终端用户的服务晚一步，就意味着丧失机遇和失去市场的领先地位。开发的窗口是收缩得很快的，特别在新出现的市场中。在这个市场中 ASSP 已经象往常一样为成为首个可用方案而有效地控制了额外的费用。

其结果是 ASSP 供应商必须提供传统上 FPGA 供应商拥有的能力，如灵活性、产品定制化和减少开发时间等。网络处理器的最新冲击波是 ASSP 供应商如何面对市场的压力来适应产品定制化和发展窗口的最好例子。

FPGA 供应商现在服务于许多设计者的需要，也是其他方案所不可替代的。为广泛范围用户创立的 ASSP 很难满足每一个设计者的准确需要，它们是通用的器件，面向更广的应用。

设计者则需要利用可编程逻辑或 ASIC 围绕 ASSP 来增加他们自己的价值，以便与市场上的产品相互区分，这实际上增加了成本和印制板的面积，而且降低了性能。



另一个情况，设计者不需要将全部功能构造到标准的 ASSP 中，虽然标准的 ASSP 使设计者与最终的方案更接近，但它难以完成这些任务。

类似于 FPGA，一个可编程 ASSP 方案使得设计者选择正确的特性集和优化 ASSP 达到尽可能好的结果。设计者也可以靠定制化产品同时减少成本在同一硅片上集成他们的价值计划。

## 2. CPLD 的优势

FPGA 与 CPLD 虽然在**很大程度上具有类似**之处，但由于内部结构上的差异导致了它们功能与性能上的差别。主要表现在如下几个方面：

(1)**布线能力**。Altera CPLD 独特的内连线结构使其内连率很高，不需要人工布局布线来优化速度和面积。这与 Xilinx FPGA 有限的布线线段相比，更适合于电子系统设计自动化中芯片设计的可编程器件验证。

(2)**延迟可预测能力**。Altera CPLD 的连续式布线结构决定了它的时序延迟是均匀的和可预测的，这更方便学生做设计。

(3)**适用场合**。虽然 CPLD 和 FPGA 均可集成千门以上的数字逻辑电路。但相比较而言，CPLD 更适合于完成各类算法和组合逻辑及小规模时序逻辑。FPGA 则更适合于时序较多的时序逻辑电路设计。

### 4.2.3 PLD/FPGA 入门

CPLD、FPGA 两者的功能基本相同，只是实现原理有所不同，有时我们可以忽略这两者之间的区别，统称为 PLD 或 CPLD/FPGA，下面我们以 PLD 来统称它们。

PLD 能做什么呢？它们能完成几乎所有数字器件的功能，上至高性能 CPU，下至简单的 74 电路。用户可以通过传统的原理图输入法，或是硬件描述语言自由地设计一个数字系统。通过软件仿真，我们可以事先验证设计的正确性。在 PCB 完成以后，还可以利用 PLD 的在线调试能力，随时修改设计而不必改动硬件电路。

如何使用 PLD 呢？其实 PLD 的使用很简单，学习 PLD 比学习单片机要简单得多，有一定数字电路基础，会使用计算机，就可以进行相应的开发。开发 PLD 需要了解两个部分：一是开发软件，二是 PLD 本身。

由于 PLD 软件已发展的相当完善，用户甚至可以不用详细了解 PLD 的内部结构，也可以用自己熟悉的方法：如原理图输入或 VHDL 语言来完成相当优秀的 PLD 设计。所以对初学者，首先应了解 PLD 开发软件和开发流程。而进一步了解 PLD 的内部结构，将有助于提高我们设计的效率和可靠性。

如何获得 PLD 开发软件呢？许多 PLD 公司都提供免费试用版或演示版。例如：可以免费从 [www.Altera.com](http://www.Altera.com) 上下载 Altera 公司的 Maxplus2 (Baseline 版或 E+MAX 版)，或向其代理商索取这套软件。Xilinx 公司也提供免费软件 Foundation。Lattice、Actel 等公司也都有类似的免费软件提供。以上免费软件都需要在网上注册申请 License 文件。

对于 PLD 产品，一般分为：基于乘积项 (Product-Term) 技术，E2PROM (或 Flash) 工艺的中小规模 PLD，以及基于查找表 (Look-Up table) 技术，SRAM 工艺的大规模 PLD/FPGA。E2PROM 工艺的 PLD 密度小，多用于 5,000 门以下的小规模设计，适合做复杂的组合逻辑，如译码。SRAM 工艺的 FPGA，密度高，触发器多，多用于 10,000 门以上的大规模设计，适合做复杂的时序逻辑，如数字信号处理和各种算法。目前有多家公司生产 CPLD/FPGA，最大的几家是：Altera、Xilinx、Lattice 和 Actel。

在 PLD/FPGA 开发软件中完成设计以后，软件会产生一个最终的编程文件 (如 .pof)。如何将编程文件烧到 PLD 芯片中去呢？

(1)对于基于乘积项 (Product-Term) 技术，E2PROM(或 Flash)工艺的 PLD(如 Altera 的 MAX 系列、Lattice 的大部分产品、Xilinx 的 XC9500 系列)厂家提供编程电缆，如 Altera 叫：Byteblaster，电缆一端装在计算机的并行打印口上，另一端接在 PCB 板上的一个十芯插头，PLD 芯片有四个管脚(编程脚)与插头相连。它向系统板上的器件提供配置或编程数据，这就是所谓的在线可编程(ISP)。Byteblaster 使用户能够独立地配置 PLD 器件，而不需要编程器或任何其它编程硬件。编程电缆可以向代理商购买，也可以根据厂家提供的编程电缆的原理图自己制作，成本仅需一、二十元。目前的 PLD 都可以用 ISP 在线编程，也可用编程器编程。

(2)对于基于查找表技术 (Look-Up table) 技术，SRAM 工艺的 FPGA (如 Altera 的所有 FLEX、ACEX、APEX 系列，Xilinx 的 Sparten、Vertex)，由于 SRAM 工艺的特点，掉电后数据会消失，因此调试期间可以用下载电缆配置 PLD 器件，调试完成后，需要将数据固化在一个专用的 E2PROM 中 (用通用编程器烧写)，由这片配置 E2PROM 先对 PLD 加载数据，十几个毫秒后，PLD 即可正常工作。

(3)反熔丝(Anti-fuse)技术的 FPGA，如 Actel、Quicklogic 及 Lucent 的部分产品就采用这种工艺。用法与 EEPROM 的 PLD 一样，但这样的 PLD 是不能重复擦写，所以初期开发过程比较麻烦，费用也比较高昂。但反熔丝技术也有许多优点：布

线能力更强,系统速度更快,功耗更低,同时抗辐射能力强,耐高低温,可以加密,所以在一些有特殊要求的领域中运用较多,如军事及航空航天。

#### 4.2.4 可编程 ASIC 的发展趋势

可编程 ASIC 作为一种行业,经过了 20 世纪 80 年代的起步,现已发展到相当规模,成为当今世界上最富吸引力的半导体器件产品,在现代电子系统设计中扮演越来越重要的角色。过去几年里,可编程 ASIC 的市场增长主要来自大容量可编程逻辑器件 CPLD 和 FPGA。

##### 1. CPLD 和 FPGA 发展方向

未来的发展呈现出这样的几方面趋势:

(1)向高密度、超大规模的方向发展。目前高密度可编程 ASIC 已具备了片上系统 SOC 的集成能力,产品性能发生巨大飞跃。

(2)向系统内可重构方向发展。系统内可重构是指可编程 ASIC 在置入用户系统后仍具有改变其内部功能的能力,从而在电子系统中引入“软硬件”的全新概念,极大提高电子系统灵活性和适应性。

(3)向低电压,低功耗的方向发展。可编程 ASIC 作为电子系统重要组成部分,不可避免地向 3.3v/2.5v/1.8v 的标准靠拢,扩大应用范围,满足节能要求。

(4)向高速可预测延时器件的方向发展。可编程 ASIC 产品如果要在高速系统中占有一席之地,必然要求了器件高速可预测延时的发展。

(5)向混合可编程的技术方向发展。迄今为止,有关可编程 ASIC 的研究和开发的大部分工作基本上都集中在数字逻辑电路上,未来的局面将会有所改变,模拟电路和混合电路的可编程技术将得到发展。可编程模拟 ASIC 作为今后模拟电子电路设计的一个发展方向,必将翻开模拟电路设计的新篇章,使得模拟电子系统的设计也和数字系统设计一样简单易行。

可编程 ASIC 是一门正在发展的技术,未来发展的动力仍来自与实际应用的要求和制造商之间的竞争。可编程 ASIC 在结构、密度、功能、速度和灵活性方面将得到进一步发展,在现代电子系统设计中将起到越来越重要的作用。

##### 2. 可编程 SOC

近年来,可编程逻辑器件(PLD)一直呈现很好的发展态势,PLD 灵活方便,不仅性能、速度、连接具有优势,而且可以缩短上市时间,因此,应用领域不断拓展。随着通信设备、数据传输,以及计算技术的高速化和复杂化,对 PLD 的集

成度和性能提出了越来越高的要求，因而可编程片上系统或平台，也即可编程 SOC 就应运而生。

为了在增长潜力巨大的通信市场占有更大的份额，业界重要的可编程逻辑器件制造商都先后推出了自己的系统级产品。概括起来大致是采用两种方法来实现可编程 SOC，一种是在可编程器件 FPGA 中嵌入 CPU 内核，获得可编程系统平台；另一种是将可编程模块置入 ASIC 之中，得到具有可配置功能的 ASIC。

(1) Actel 公司的 VariCore 内核和 ProASIC Plus FPGA。Actel 公司是反熔丝（一次性烧写）PLD 的领导者，由于反熔丝 PLD 抗辐射，耐高低温，功耗低，速度快，所以在军品和宇航级上有较大优势。

Actel 的 VariCoreTM 是基于 SRAM 的嵌入式可编程门阵列 (EPGATM) IP 内核，采用 Chartered 半导体制造公司的  $0.18\mu\text{m}$  加工工艺制造。VariCore EPGA IP 具有小的片上可重编程 SOC 裸片面积，比标准 FPGA 的性能/裸片面积比更高，也比其他标准软件 IP 方案具有更佳的性能/功耗比，是完整的从前端至后端的嵌入式“软硬件”可重编程内核，可用于 ASIC 和 ASSP 的系统级芯片中，增强 ASIC 和 ASSP SOC 设计的灵活性，有效降低设计风险，加速产品投放市场。

Actel 公司的 ProASIC Plus 是基于闪存的微粒结构 FPGA 系列产品，为系统设计者提供一个可行的 ASIC 替代方案。该产品解决了困扰第一代 ProASIC 的制造产能率问题。

ProASIC Plus 比上代产品的功能更加齐全，但仍是价格低廉的主流产品，可满足几乎一半 ASIC 市场的需求。据估计，该产品最高达 100 万系统门（约 30 万 ASIC），足以满足 44% 的 ASIC 设计需要。新的 ProASIC Plus FPGA 器件兼具 ASIC 的密度和低成本，以及可编程逻辑器件的面市速度快等优势。

ProASIC Plus 器件采用  $0.22\mu\text{m}$  标准 CMOS 工艺，内置有闪存单元，因此具有可再编程及安全的非易失代码存储功能。Actel 称，其细粒结构比典型的粗粒可编程逻辑结构能更好地转换到 ASIC。

ProASIC Plus 器件工作电压为 2.5/3.3V，工作频率高达 100MHz。该系列产品包含 6 个型号，系统门数从 15 万到 100 万不等，包括嵌入式 RAM、高速可配置 I/O 和多个片上 PLL 等。有关 VariCore 和 ProASIC Plus 的更多信息可访问：<http://www.actel.com>。

(2) Altera 公司的 Excalibur 嵌入处理器方案和 Stratix 器件。Altera 的 StratixTM 器件构建在新的 MultiTrackTM 布线结构之上，是一种基于模块的设计，使用户能够方便地以一些小功能模块构建复杂的设计。其性能比 Altera 的 APEXTM II 器件提升 40%，内核尺寸比以前的体系小 35%，提供多达 10Mb 的 RAM 和 114 140 个逻辑单元，是相近产品存储容量的 3 倍，多 21 000 个逻辑单元。另外，Stratix 器件包括了专用 DSP 功能，能够实现比一般 PLD 快两倍的 DSP 和复杂计算应用，是容量大又速度快的可编程逻辑器件。

Altera 公司称在 PLD 中 Stratix 器件具有最大程度地集成内存、逻辑单元和 DSP 功能。它为客户提供和 ASIC 一样强大的产品，同时又具有可编程逻辑的全部优势。Altera 公司的方案中心的网址是 <http://www.Altera.com/solutioncenters>，它

可以为客户提供不同内存、DSP 和 I/O 设计需求的解决方案。

Altera 推出了多种 Excalibur<sup>TM</sup> 嵌入处理器方案, 交叉采用 ARM、MIPS 以及 Nios 等多种内核。新的基于 ARM 处理器的 Excalibur 嵌入处理器方案, 基于 APEX 器件体系, 集成了 ARM922T 处理器, 运行速度可达 200MHz。除了处理器内核及其相关缓冲和存储管理单元外, 这种基于 ARM 的 Excalibur 器件还包括外加的内部 SRAM 和 DPRAM 存储器、外设、外部存储控制器和软件调试的 JTAG 接口, 可编程逻辑可多达 38400 个逻辑单元。

这种 Excalibur 嵌入处理器解决方案在 PLD 结构中集成了处理器子系统, 使设计人员能够在处理器主导模式下引导系统和动态配置可编程逻辑。Nios 软核和基于 ARM 的嵌入处理器为可编程单芯片系统(SOPC)提供了灵活性。关于 Excalibur 嵌入处理器方案的更多信息, 请浏览 <http://www.Altera.com/products/devices/excalibur/exc-index.html>。

(3)ATMEL 公司的 FPSLIC 系列产品。Atmel 公司推出的 AT94K 和 AT94S 系列现场可编程系统级芯片(FPSLIC), 是将组成基本系统所需的逻辑、外设、存储器和微控制器等嵌入式系统模块集成在一片基于 SRAM 的现场可编辑器件上, 实现单芯片可编程 SOC。

FPSLIC 把 AT40K FPGA 和一个高性能 AVR 8 位 RISC MCU 结合在一起, 有两个 UART、一个时钟计数器、可编程 I/O、36K SRAM, 所有的这些都集成在一个芯片上, 能节约 70% 的芯片面积, 50% 的功耗, 同时性能也提高了 50%。FPSLIC 可在 FPGA 内对附加的外围设备和定制逻辑电路进行编程, 与使用分立元件的和其他类型方案相比, 可以加快产品面市时间。AVR 微控制器和 FPGA 逻辑电路的微码可以无数次重新配置, 从而使 FPSLIC 器件成为开发基于单集成电路的多端产品的理想平台。

AT94K 集成了 5000 ~ 40000 门的 FPAG, 1MIPS AVR RISC 微控制器, 36Kb SRAM, 以及 UARTS、定时器/计时器、可编程 I/O 等。

AT94S 集成了配置 EEPROM, 5000 ~ 40000 门的 FPAG, 20+MIPS AVR RISC 微控制器, 36Kb SRAM, 以及 UARTS、定时器/计时器、2 针串口等外设, 可以保证 IP 的安全。关于 AT94K 和 AT94S 的更多信息, 请浏览 <http://www.atmel.com>。

(4)Lattice 公司的 FPSC 和 ORCA FPGA。莱迪思半导体 (Lattice Semiconductor) 公司在集成 ASIC 宏单元和 FPGA 门于同一个硅片的方法方面处于领先地位。他们将该技术称之为单片现场可编程系统(FPSC)。与带有嵌入式 FPGA 门的 ASIC 相比, FPSC 器件是提供系统解决方案的器件。FPSC 器件将 ORCA Series 4 型 FPGA 可编程逻辑结构与总线接口、高速线路接口及高速收发器等内嵌的 IP 核组合起来, 形成优化的 ASIC 门。这意味着 FPSC 器件不仅是单芯片系统, 而且还是集使用灵活性和高性能于一身的针对复杂功能的强大的 IP 载体。嵌入式宏单元拥有工业标准 IP 核, 诸如 PCI、高速线接口和高速收发器。当这些宏单元与成千上万的可编程门结合起来时, 它们可应用在各种不同的高级系统设计中。

Lattice 公司的 ORCA 现场可编程门阵列(FPGA)是构建在所熟悉的优化重组单元阵列(FPGA)结构上的、新的现场可编

程门阵列(FPGA)系列。这种 FPGA 器件系列提供了许多新的特征和架构增强特性,早期的 FPGA 不具备这些特点。结合灵活的基于 SRAM 的可编程逻辑强有力的系统特征,丰富的布线层次和互连资源,以及融合多接口标准,FPGA 的 ORCA FPGA 系列可以适用于大多数复杂、高性能的设计。关于 FPSC 和 ORCA FPGA 的更多信息请浏览公司网站: [www.lattice.com](http://www.lattice.com), [www.latticesemi.com](http://www.latticesemi.com) 和 [www.latticesemi.com.cn](http://www.latticesemi.com.cn)。

(5)QuickLogic 公司的 QuickMIPS。QuickLogic 的优势是连接逻辑门的方法,该公司使用一种称作 ViaLink 的无晶体管连接方法,可以放入更多的信号绕线开关。该公司早在 1999 年就推出了在 FPGA 中集成了类似微处理器和高速串行接口功能的嵌入标准产品(ESP)。

QuickLogic 公司新推的 QuickMIPS 产品系列采用了嵌入式 MIPS 处理器以及其他专用内核。

FPGA 的灵活性的 QuickMIPS 架构和开发平台,关注 ASSP 的性能和经济性,并为用户增加了可配置灵活性,使工程师在获得高性能处理器同时可以缩短上市时间。

QL901M 基于 MIPS 科技公司的 MIPS32 4Kc 内核,采用  $0.25\mu\text{m}$  工艺时运行速度可达 133MHz,而采用  $0.15\mu\text{m}$  工艺则可达到 175MHz。该芯片集成了 45.7 万个系统门、两个 10/100 以太网控制器、一个 32 位 66/33MHz PCI 接口、多功能存储控制器和一个中断控制器。一个 32 位的高级外设总线可连接 4 个 32 位定时器和两个 UART。关于 QuickMIPS 的更多信息,请浏览 <http://www.quicklogic.com>。

(6)Xilinx 公司的 Virtex-II Pro FPGA。赛灵思(Xilinx)公司是 FPGA 的发明者,他们推出的 Virtex-II Pro<sup>TM</sup> FPGA 系列产品,采用  $0.13\mu\text{m}$  工艺,9 层金属结构,BGA 封装,是基于 Virtex-II 系列基础的高端 FPGA。主要特点是在 VirtexII 上增加了高速 I/O 接口能力和嵌入了 IBM 公司的 PowerPC 处理器,以解决高性能系统结构所面临的挑战。IBM 和 Xilinx 合作,利用 IP 植入(IP Immersion)技术,在 Virtex-II 结构中植入领先的嵌入式处理器结构—IBM PowerPC。Virtex-II Pro 系列支持最多达 4 个运行频率高达 300MHz 的 PowerPC 405 处理器。这种植入方法允许硬 IP 核心分布在 Virtex 结构中的任何位置,同时可保持与周围逻辑阵列的平滑集成。IP 植入技术将核心中的所有高速总线与可编程结构直接密切耦合,从而获得了比同样的分立处理器高得多的系统级性能。

Virtex-II Pro FPGA 还集成了 RocketIOT<sup>TM</sup> 技术,这是支持多端口的 3.125Gbps 串行接口的可编程解决方案。这一集成为高性能接口标准,如千兆位以太网、10G 以太网、3GIO、SerialATA、Infiniband 和 FibreChannel,提供了一个完全的解决方案。

Virtex-II 结构还包括了先进的主动互连(Active Interconnect)、块 RAM(BlockRAM)和时钟管理功能。关于 Virtex-II Pro FPGA 的更多信息请浏览 <http://www.Xilinx.com>。

综上所述,可编程 SoC 已经渐成可编程器件的发展趋势。ASIC 与 FPGA 融合的概念得到越来越多的可编程器件厂商

的认同，尽管 FPGA 和 ASIC 各具特点，但这些融合产品逐渐模糊了两者之间的界限。目前，虽然有许多这种融合的高性能 ASIC 替代产品进入市场，而且今后的发展方向也可能是二者进一步融合，但是，据预测，在今后一段时间内 ASIC 仍然会占据高端芯片市场。

## 器件比较与选择

### 一、逻辑单元

CPLD 基于乘积项的结构，一般可分为三块结构：宏单元(Macrocell)、可编程连线(PIA)和 I/O 控制块。宏单元是 PLD 的基本结构，由与或阵列和可编程 D 触发器构成，由它组合来实现基本的逻辑功能。由此可知 CPLD 分解组合逻辑的功能很强，一个宏单元就可以分解十几个甚至 20 - 30 多个组合逻辑输入。而 FPGA 的逻辑单元是小单元，一个 LUT 只能处理数个输入的组合逻辑，因此，CPLD 适合用于设计**译码**等复杂组合逻辑。

但 FPGA 的制造工艺决定了 FPGA 芯片的工艺结构占用芯片面积小，速度高（通常延时只有 1 ~ 2ns），每片芯片上集成的逻辑单元和触发器的数量非常多，如果用芯片价格除以逻辑单元数量，FPGA 的**平均逻辑单元成本大大低于 CPLD**。而且如果设计中使用到**大量触发器**，例如设计一个复杂的时序逻辑，那么使用 FPGA 就是一个很好选择。

### 二、编程工艺

从编程角度考虑，CPLD 采用的是可擦写的 Flash 或 EEPROM 结构，其擦写次数限制在**数万次**以内，擦写速度较慢，不太适合大规模反复擦写，并且**功耗较大**。而基于 SRAM 工艺的 FPGA 则可**反复无损擦写**，实现真正意义上的在线编程，且配置速度快，更加适合在 ISP 系统中用于验证开发阶段设计者的构思。同时降低了成本费用。

### 三、应用方向

目前可编程逻辑器件的发展趋势是系统集成化(SOC)，FPGA自身的特点决定它的规模便于不断增大，集成度可以不断提高，更适合在系统功能日益复杂，特别是片上系统出现的今天，向**嵌入IP内核**方向发展。

### 四、互联与延时

CPLD由于使用集总总线，总线上任意一对输入输出端之间的**延时相等**，且是**可预测**的。而FPGA由于单元小，互连复杂，使用的互连方式较多，有**分段总线、长线和直接互联**等。一对逻辑单元之间的互联通路可以有多种，其传输延时也是不同的。可见对于FPGA而言其延时是不确定的。实现同一个功能的不同方案延时不等。因此用**FPGA开发ASIC使出了逻辑设计外，还要进行延时设计**。

然而CPLD、FPGA之间的界限并非不可逾越，Altera公司的FLEX10K10系列就是介于二者之间的产品。它采用查找表结构的小单元，SRAM编程工艺，且每片含触发器较多，可以达到很大的集成规模，这些都和典型的FPGA相一致。但这种器件中又使用了集总总线的互联方式，速度高且pin-pin延时确定、可预知，又具有CPLD的特点。

### 五、小结

综上所述，考虑到器件及设计开发板的主要用途，为实现可在线编程，可重复配置，能实现一定程度的复杂系统。且便于学习和研究可编程逻辑器件的逻辑与延时的仿真与综合。

## 4.3 MAX+PLUS II 及其应用

MAX+ PLUS II(Multiple Array Matrix and Programmable Logic User Systems)开发工具是美国 Altera 公司设计的 EDA 工



具。利用该工具所提供的编辑、仿真、综合、编译、芯片编程等功能，将设计好的电路图或硬件描述语言程序转换成基本的逻辑单元写入到可编程的芯片中，做成 ASIC 芯片。在网上有许多中英文关于 MAX+ PLUS II 学习使用的文章、书籍，该软件具有完整的在线帮助系统，易于使用和掌握。本节将以交通灯控制器芯片生成的实例来讲解 MAX+ PLUS II 10.1 版本使用方法。

### 4.3.1 MAX+ PLUS II 软硬件环境

#### 1. MAX+PLUSII 简介

ALTERA 公司的 MAX+PLUSII 开发软件系统是一个完全集成化、易学易用的可编程逻辑设计环境，它可以从网上自由下载，并可去 Altera 公司申请非商用的版权许可证文件(license.dat)，基本满足万门级设计需求，该软件具有以下优点。

(1)开放的界面。Altera 的工作与 EDA 厂家紧密结合，使 MAX+PLUS II 可与其他工业标准的设计输入、综合与校验工具相连接。设计人员可使用 Altera 或标准 EDA 设计输入工具来建立逻辑设计，使用 MAX+PLUS II 编译器对 Altera 器件设计进行编译，并使用 Altera 或其他 EDA 校验工具进行器件或板级仿真。

(2)与结构无关。MAX+PLUS II 系统的核心 compile 支持 ALTERA 公司的 FLEX10K、FLEX8000、MAX7000 等可编程逻辑系列，提供了真正与结构无关的可编程逻辑设计环境。编译器还提供了强大的逻辑综合与优化功能，使用户比较容易的将其设计集成到器件中。

(3)多平台。可良好运行于普通 PC 机上，也可运行于工作站。

(4)完全集成化。MAX+PLUS II 的设计输入、处理与校验功能全部集成在统一的开发环境下，这样可以加快动态调试，缩短开发周期。

(5)丰富的设计库。提供丰富的库单元供设计者调用。

(6)OpenCore 特性。MAX+PLUS II 软件具有开放核特性，它允许设计人员添加自己认为有价值的宏函数。

#### 2. 如何选择合适的芯片

(1)一般情况下，尽可能选用速度等级最低的芯片，尽可能选用电压比较低的芯片(性价比较好)，尽可能选用贴片封装的芯片。

(2)如果设计中不需要使用容量较大的内嵌存储器，或超过 256 个宏单元的设计尽量选用 FLEX6000 系列的芯片，否则要用 FLEX10K 或 1K。

(3)如果设计中需要较大的存储器和比较简单的外围逻辑电路,而且对速度、总线宽度和 PCB 板面积无特殊要求的情况下,尽量选用一片 MAX7000 或 3000 系列的芯片和外接存储器。

(4)在速度较高的双向总线上尽量采用 MAX7000 或 3000 系列的芯片。

(5)如设计规模需要超过 10 万门或需要 PLL、LVDS、CAM 等新技术,则可以选择 APEX20KE。为保证及时供货和性价比,新设计应优先选择以下型号: MAX7032SLC44-10、7064SLC44-10、7128SLC84-15、7128STC100-15、7128AETC100-10、7128AETC144-10、FLEX6016AQC208-3、6016ATC144-3、10K20TC144-4、10K30EQC208-3、10K50EQC240-3 以及刚刚推出的 MAX3032ALC44-10、3064STC100-10、APEX20KE、ACEX1K 等等。最好是先和代理商沟通,再确认所需型号。

### 3. 交通灯芯片选择实例

(1)综合考虑上面的这些因素,以及开设其他 EDA 实验的需要,我们的所要设计交通灯控制器实验所用的芯片选用 Altera 公司的 MAX7000S 系列中的 EPM7128SLC84-15???

(2)MAX7000 系列是工业界中速度较快的高集成度可编程逻辑器件系列。它是在 Altera 公司的第二代 MAX 结构基础上,采用先进的 CMOS E<sup>2</sup>PROM 技术制造的,可 100%模仿 TTL。MAX7000 系列(包括 MAX7000E、MAX7000S、和 MAX7000A 器件)的集成度为 600~5000 可用门,使用 5V 或 3.3V 电源,有 32~256 个宏单元和 36~155 个用户 I/O 引脚。能够提供组合传输延迟快至 5.0ns,16 位计数器的频率为 178MHZ,遵守 PCI 规定,可编程宏单元触发器具有专用清除、置位、时钟和时钟使能控制。此外,它们输入寄存器的建立时间非常短,能过提供多个系统时钟且有可编程的速度/功率控制。另外还有**编程保密位**,全面保护专利设计。

(3)MAX7000S 是 MAX7000 的增强型,具有高集成度,还有 6 个由引脚或逻辑驱动的输出使能,2 个可选为反向工作的全局时钟信号。并且改善了布局布线,增加了连线资源,加快了从 I/O 引脚到宏单元寄存器的专用路径的建立时间。

EPM7128SLC84-15 有 **2500** 个可用门、**84** 个引脚,其中输入输出的引脚 **68** 个。MAX7000S 包括逻辑阵列块(LAB)宏单元(Macro Cells)、扩展乘积项(共享和并联)、可编程连线阵列(PIA)和 I/O 控制块五部分。另外,MAX7000 结构中还包括 4 个专用输入,它能用作通用输入或作为每个宏单元和 I/O 引脚的高速的、全局的控制信号,即时钟(Clock),清除(Clear)和输出使能(Output Enable)。器件方框图如图 4-2 所示

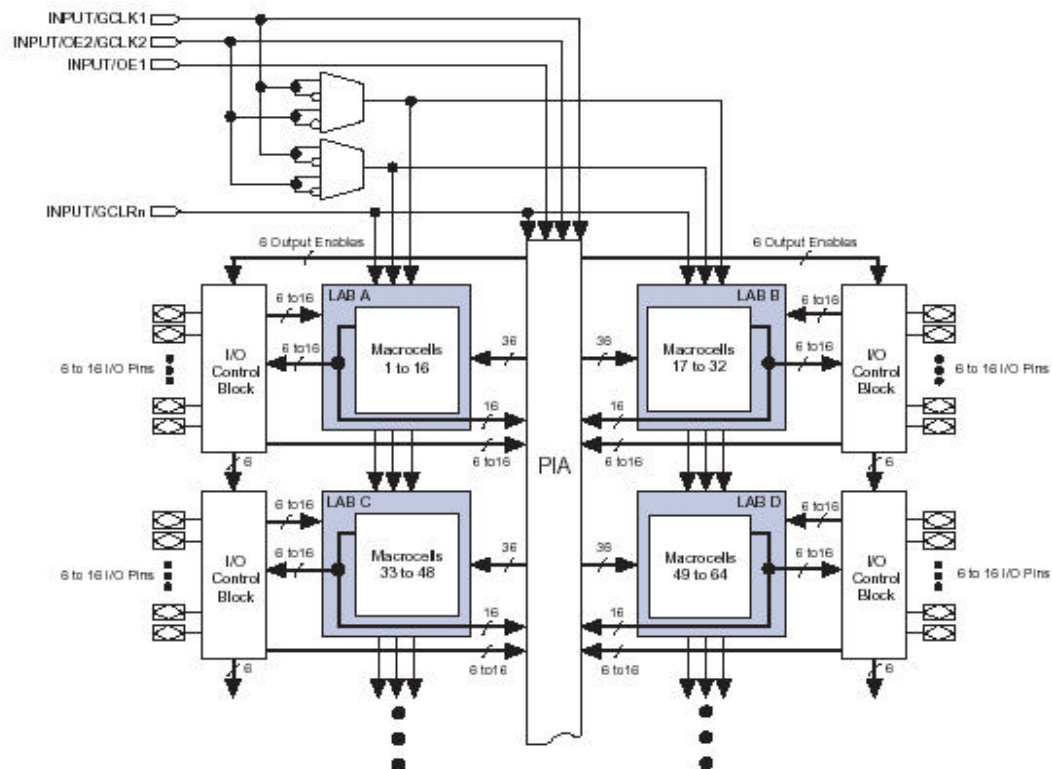


图 4-2 MAX7000S 的结构方框图

### 4.3.2 MAX+PLUSII 的设计过程

MAX+PLUS II 10.1 是由设计输入、设计处理、设计校验和器件编程四部分组成。如图 4-3 所示。

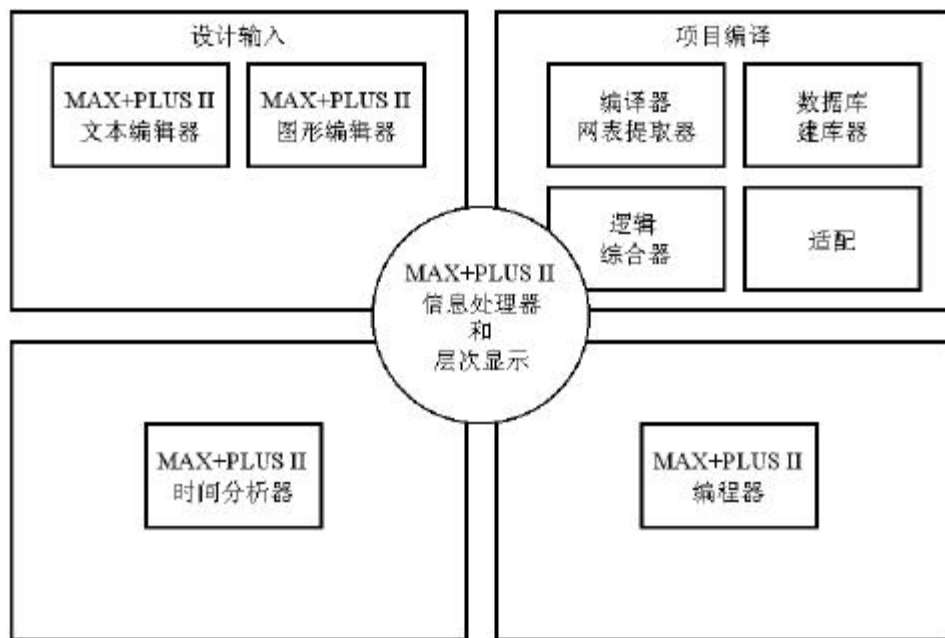


图 4-3 MAX+Plus II 的设计流程

### 1. 设计输入

MAX+PLUSII 设计输入的方法有多种，包括：原理图输入方式、文本设计输入方式、高级设计输入方式、波形设计输入方式、层次设计输入方式和底层设计输入方式。最常用的是原理图输入和文本输入两种方式。

(1)原理图输入是指用 MAX+PLUSII 提供的各种原理图库进行设计输入，是一种最直观的输出方式。但是输入效率低。

(2)文本设计输入是指设计者用 VHDL、Verilog HDL 或 AHDL ( ALTERA 自己开发的硬件描述语言 ) 编写 HDL 源程序进行输入。采用这种方法描述的优点是效率高，结果也较容易仿真，在不同设计库之间的转化非常方便。

### 2. 项目编译

MAX+PLUSII 处理一个设计时，Compiler 在设计文件中读取信息并产生编译文件和仿真文件，Timing Analyzer 可分析

设计定时, Message Processor 可自动定位错误。

(1)自动错误定位。Message Processor 与 MAX+PLUSII 的所有应用程序通信。可以给出信息(错误、警告等)。设计者可以利用它打开有错误源的文件,并以高亮显示。

(2)逻辑综合与试配。MAX+PLUSII Compiler 的 Logic Synthesize(逻辑综合)模块对设计方案进行逻辑综合并能看真正结果。Fitter(试配)模块应用试探法可把经过综合的设计最恰当的用一个或多个器件实现,使设计者得以从冗长的布局布线工作中解脱出来,生成报告文件(.rpf),该文件显示设计的具体实现以及器件中未使用的资源,并说明用户的定时要求是如何具体实现的。

(3)设计规则检查。Compiler 中的 Design Doctor 程序能检查每一个设计文件。用户可以选择预先定义好的三组检查规则中的一种,也可以建立自己的规则。

(4)编译文件的产生。Assemble(装配程序)模块为已编译的设计创建烧写文件(.pof)。

### 3.设计校验

设计校验包括设计仿真和定时分析两部分。其作用是测试逻辑操作和设计的内部定时。

(1)仿真。电路设计输入完之后,首先须检验输入是否正确。这是一项简单的逻辑检查,MAX+PLUSII 提供了功能编译选项。此时,只运行仿真网表的提取,而不做布局布线。所有延时为零延时。在仿真时需加入激励信号,激励信号可以用波形编译器直接编译成波形文件,也可以先用文本编辑器按软件给定的语法规则编辑成文本再转换成波形文件。如检查到错误则需修改原设计方案。

功能仿真无误后,需要进行后仿真。首先进行完全编译,每个设计项目都有一个配置文件(.acf),所有配置参数都存放在这里。如果需要修改配置,既可在菜单上修改,也可以直接修改此文件。

对于相对简单的设计。也可以只做一步仿真,即后仿真。

(2)定时分析。Timing Analyzer 可以计算点到点的器件延时矩阵。确定器件引脚上的建立时间与保持时间。还可以计算最高时钟频率。Message Processor 可以找出 Timing Analyzer 在设计文件中已证实的关键路径,并在适当的设计编辑器中加以现实。

### 4.器件编程

MAX+PLUS II Programmer 是使用 Compiler 生成的烧写文件(.pof)对 ALTERA 器件进行编程的。它可以用来对器件编程、校验、检查是否空白以及进行功能测试。

## 4.4 开发板设计原理及使用说明

**编者按：**不要认为学习好 MaxplusII 或 ISE 等 EDA 工具软件就能设计好 CPLD/FPGA 了，事实上数字电路、模拟电路知识（至少了解如何选用相应解决方案）和实际电路设计、分析**经验（书本上学不到的知识，作坊式的耳闻目染）**比熟练掌握这些 EDA 工具更重要。**重方法、轻流程。**

配《VHDL 及复杂数字系统设计》一书的开发板是基于 Altera 公司 EPM7128S 的实验板，电路板设计简明，易于理解、易于扩展。

### 1 . 电源电路

实验板采用普通的 6V 直流稳压电源供电，一般选用普通的新英电源 **XY303**，对于少数电脑，由于对 LPT 输出电流较敏感，其所需下载电流大于 **400mA**，我们可采用新英电源 **XY708**。DC 电压输入和一个 2.5mm×5.55mm 的插孔电源联接器相连。可接受的 DC 电压范围是 **6V~9V**，电流强度的提供最少要达到 **350mA**。

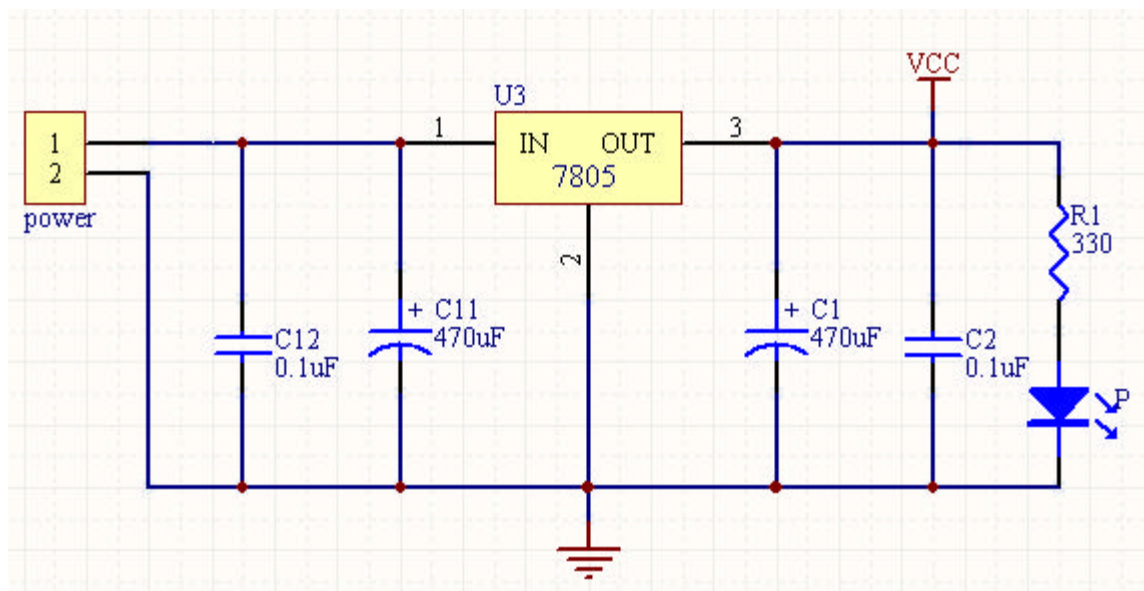


图 4-4 电源电路

在图 4-4 中，C1 和 C11 是防止电源干扰信号的**滤波电容**，我们选用的是  $470\ \mu\text{F}$  电解电容。C2 和 C12 是防止高频信号的干扰的**贴片电容**。

我们选用了 LDO(低压差线性稳压器)芯片 **LM7805** ,TO-220 封装，它可以在较宽的电压输入范围（5V 到 12V），稳定输出 5V。为保证芯片稳定工作，可采用大面积敷铜和裸露敷铜。一个标记有 P 的绿色发光二极管（Light-Emitting Diode，简称 LED）在 5V 整流电压源

有电流输出时被点亮。

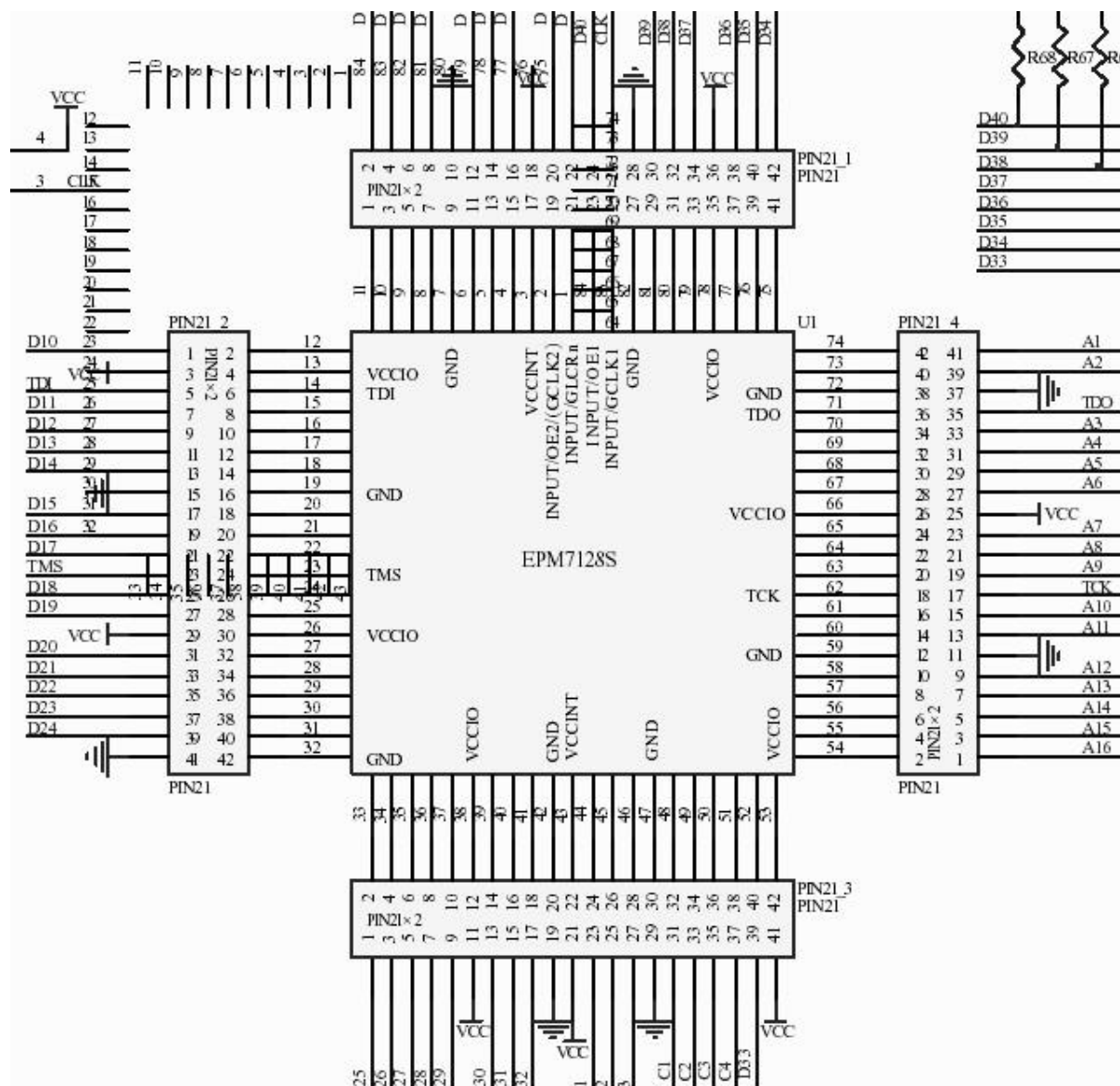
如果我们要输出 3.3V、2.5V、1.25V，我们可以选用芯片 **LM317** 等芯片，这也是我们不断关注相关的芯片参数的原因。

注意：开发板与计算机连接时，要在关机状态下，先接好两边并口的连线，然后再接开发板电源和开机；开发板与计算机分离时，先将开发板电源断开，再从并口卸下下载线。

## 2 . EPM7128S 芯片

EPM7128S 属于高密度高性能的 MAX7000S 系列，制造技术基于电可擦除可编程的只读存储器（Electrically Erasable Programmable Read Only Memory，简称 E<sup>2</sup>PROM）单元。选用带插座的 84 脚 PLCC 封装，该插座兼容以下芯片：**EPM7064SLC84**(5V)、**EPM7096SLC84**(5V)、**EPM7160SLC84**(5V)。参见 Schematic Prints.pdf，可以任意放大或缩小。





开发板上的 I/O 端口需要和实验资源连接时,将短路帽插上即可。如果需要把板子的 I/O 和实验板外的电路连接,把对应跳帽拔掉,自己连到外部,但要注意需要共地。

特别要注意以下全局输入引脚:

**1 脚 Gclear(1)**: 全局清零,如有寄存器清零,可由此脚控制。当然也可以由内部逻辑产生清零信号。Assign>-Device 中选择器件型号,再在 Assign>-Pin 中填入待分配的管脚号和类型,另一种方法是在原理图中选中 Input 或 Output,点击鼠标右键,选 Assign Pin,填入想分配的管脚号,编译一遍即可。

**2 脚**: 全局输出使能 / 全局时钟脚。该引脚也可作输入使用。

**83 脚**: 全局时钟脚,用作外部时钟输入,它到所有逻辑单元的延时基本相同。如果用其他脚作为时钟输入,必须将图 4-5 中 “ ” **点掉**。去掉后,则可以选择任意 I/O 引脚作时钟输入脚。如果怀疑时钟设计或引线有问题,可先测试组合逻辑,也可用信号发生器注入一个信号来测试,确认故障。

**84 脚**: **全局输出使能**,可用此脚控制三态输出(当然也可以由内部逻辑产生)。

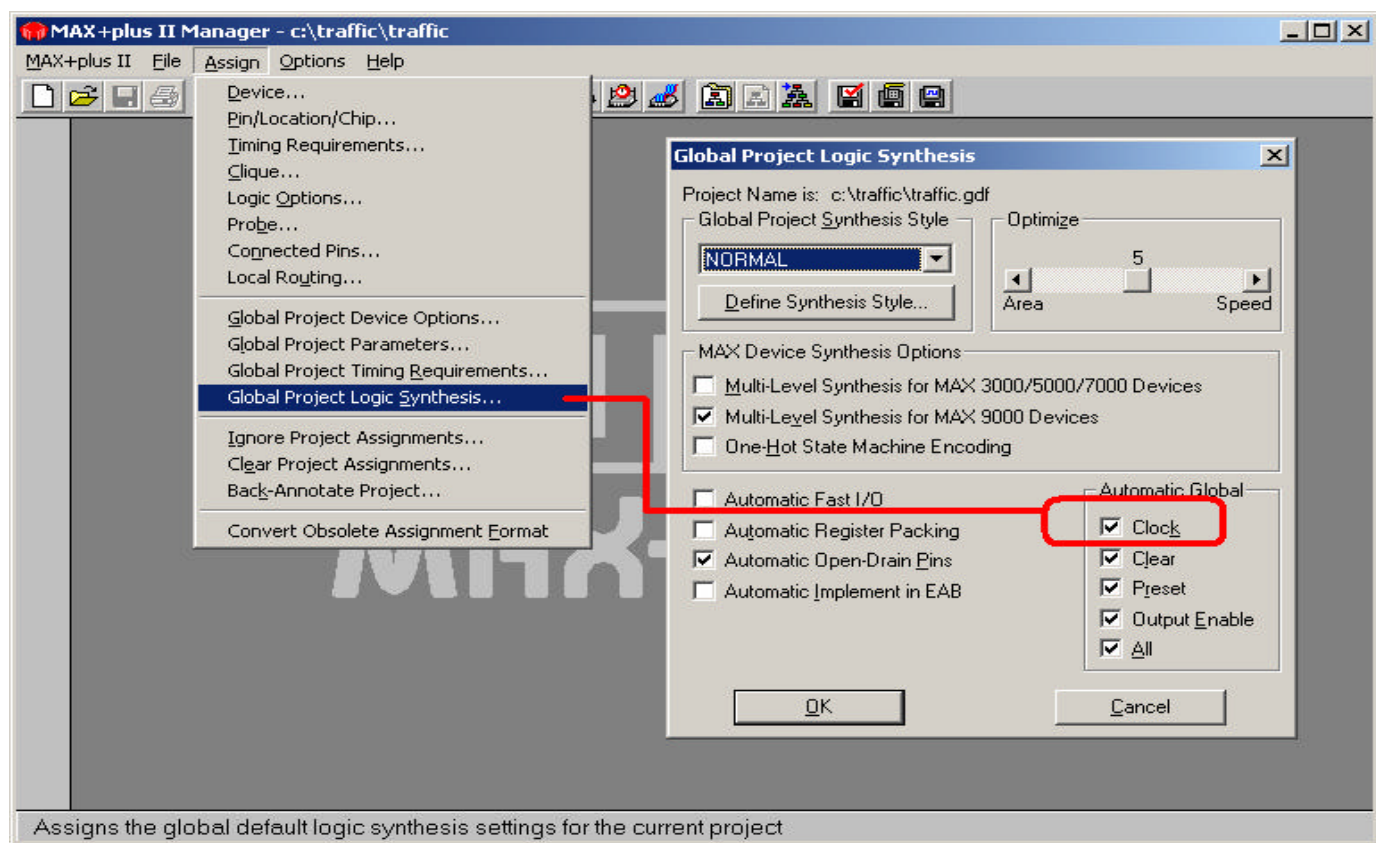


图 4-5 设置全局时钟

另外，EPM7128SLC84(5V)内含 128 个宏单元（macrocells）。每个宏单元含有一个可编程逻辑“与”和固定逻辑“或”的阵列，以及一个带独立可编程时钟的可配置寄存器。拥有

2500 个逻辑门和一个简单的结构，EPM7128S 非常适合入门级设计以及大型组合和时序逻辑功能设计。



提示：更多关于 MAX7000 器件的资料，请到公司网站参看《MAX 7000 系列可编程逻辑器件一览表》( MAX 7000 Programmable Logic Family Data Sheet )。

### ➡ 3. ByteBlasterMVTM 并行下载

逻辑设计编译结果可以便捷地通过 ByteBlasterMV 下载电缆下载到电路上。并口电缆的硬件接口就是一个标准并口。电缆将程序或者配置数据从 MAX+PLUS II 软件下载到电路板，同时可进行反复的逻辑修改，能够方便连续地进行。

该板即支持用市场上有售的 ByteBlasterMV 下载电缆下载，亦可通过板上的并口用并口线直接与计算机并口相连来下载，因为板上也有 ByteBlasterMV 下载电缆的电路，这样下载比较方便。

若用现成的 ByteBlasterMV 下载电缆下载，需将电缆上的 10 脚插孔插头和板上的 JTAG10 脚插针插头连接（注意：右上角第一个插针为管脚 1，对应 TCK 信号，下方为管脚

2, 依次类推, 不要插错位置)。由开发板提供电源和地给 ByteBlasterMV 下载电缆。数据从管脚 TDI 移位输入, 由管脚 TDO 一位输出。下表显示了当 ByteBlasterMV 工作在 JTAG (Joint Test Action Group : 联合测试行动小组) 模式下的对应 JTAG 的管脚名称。

表 JTAG 10 脚插头管脚定义

管脚	JTAG 信号
1	TCK
2	GND
3	TDO
4	VCC
5	TMS
6	没有连接
7	没有连接
8	没有连接
9	TDI
10	GND



提示：更多关于 ByteBlasterMV 下载电缆的资料，请到公司网站参看《ByteBlasterMV 并口下载电缆》(ByteBlasterMV Parallel Download Cable Data Sheet)。

## ➡ 4. CPLD 电路板预览

下面将分别描述该电路板的相关特点。下图中给出了电路板在 Protel DXP 预览下的仿真框图：

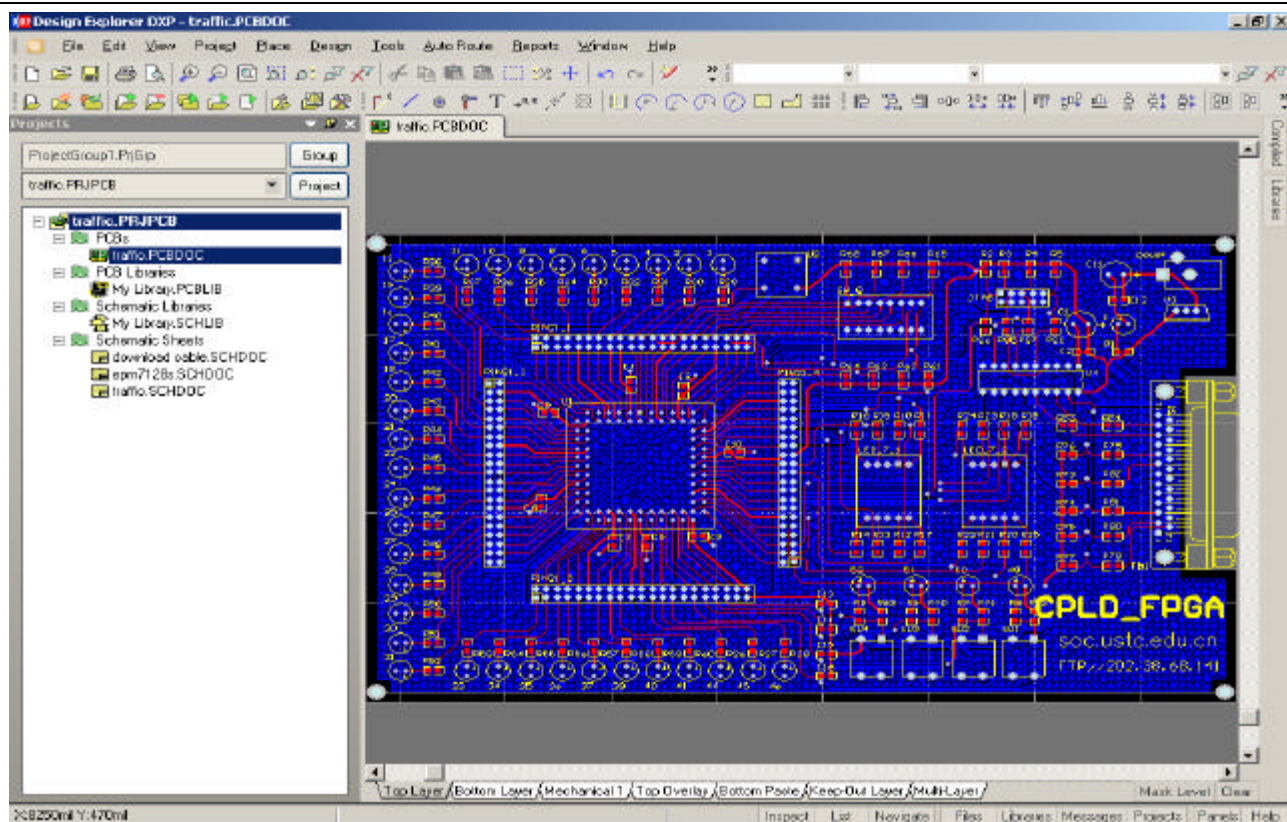


图 4-6 电路板在 Protel DXP 预览下的仿真图

电路板为 EPM7128S 提供了以下资源：

✓ 84 脚 PLCC 封装插座,可更换不同速度等级的芯片，以及兼容以下芯片：

**EPM7064SLC84(5V)、EPM7160SLC84(5V)。**

- ✓ 信号脚可通过插孔式插头连接，便于外接和调试。相关插头见电子市场中常见连接方式。
- ✓ 适用 ByteBlasterMV 电缆的 JTAG 链式连接。
- ✓ 带有 74HC244 的 ByteBlasterMV 电缆电路，，LPT 接口连接（简单一些，但电流需要大一些，占板子面积也大一些）。  
适用并口线直接下载。
- ✓ 四个瞬时按钮式开关。
- ✓ 一个八位的 DIP 封装开关。
- ✓ 所有 I/O 管脚均与发光二极管连接。
- ✓ 两个共阴七段数码管。
- ✓ 板载四角晶振（20MHz）。
- ✓ 专用的全局 CLR、OE1 和 OE2/GCLK2 管脚。

EPM7128S 的管脚并没有预先接到开关和发光二极管上，而是接到了插孔式插座上。通过对管脚的直接观测，可以更多地关注设计的基本原则和方法，从而更好地学习可编程 I/O 管脚和可编程逻辑器件。在 MAX+PLUS II 中成功地编译和验证了一个设计后，可以用插孔式插座的连接帽将指派好的 I/O 管脚和开关或发光二极管连接。然后就可以将它们的设计下载到器件中，比较设计仿真结果和实际硬件的执行结果。

## 5. 晶振

晶振在电子产品的设计有很重要的地位。如何选用无源晶体与有源晶振？无源晶体（价格低，量大生产时才考虑）相对于晶振而言其缺陷是信号质量较差，通常需要精确匹配外围电路（用于信号匹配的电容、电感、电阻等），更换不同频率的晶体时周边配置电路需要做相应的调整。建议采用精度较高的石英晶体，尽可能不要采用精度低的陶瓷晶体。



晶振在精密测量等领域需要把各种补偿技术集成在一起，从而减少了设计的复杂性。试想，如果采用晶体，然后自己设计波形整形、抗干扰、温度补偿，那样的话设计的复杂性将是什么样的呢？我们这里设计射频电路等对时钟要求高的场合，就是采用**高精度温补晶振**的，工业级的要几百元一个。特殊领域的应用如果找不到合适的晶振，就必须自己设计了，这种情况下就要选用特殊的高端晶体，如红宝石晶体等等。更高要求的领域情况更特殊，我们这里在高精度测试时采用的时钟甚至是**原子钟**、铷钟等设备提供的，通过专用的射频接插件连接，是个大型设备，相当笨重。

板载了一个 20MHz 的石英四脚有源晶振。有源晶振不需要内部振荡器，信号质量好，比较稳定，而且连接方式相对简单（主要是做好电源滤波，通常使用一个电容和电感构成的 PI 型滤波网络，输出端用一个小阻值的电阻过滤信号即可），不需要复杂的配置电路。20MHz 以下的晶体晶振基本上都是基频的器件，稳定度好，**20MHz 以上的大多是谐波的**（如 3 次谐波、5 次谐波等等），稳定度差，因此强烈建议使用低频的器件，毕竟倍频用的 PLL 电路需要的周边配置主要是电容、电阻、电感，其稳定度和价格方面远远好于晶体晶振器件晶振的输出驱动。

注意：时钟信号布线时，长度尽可能短，线宽尽可能大，与其它印制线间距尽可能大，紧靠器件布局布线，必要时可以走内层，以及用地线包围。

有源晶振通常的**用法**：一脚悬空，二脚接地，三脚接输出，EPM7128S 的全局时钟管脚（83 脚），四脚接电压。

## 6. EPM7128S 原型插头

EPM7128S 原型（prototyping）插头是环绕在器件信号管脚周围的插孔式接头。84 脚 PLCC 封装器件的每边上的 21 个管脚都链接到 21 脚、双排 0.1inch 的插孔式接头上。EPM7128S 的 I/O 管脚及专用的全局 CLR 和 OE2/GCLK2 管脚均由 LED 引出，管脚号被印制在对应的 LED 旁，其中管脚 1 和 2 分别为 CLR 和 OE2/GCLK2。



## 7. 瞬时按钮式开关

SW1、SW2、SW3 和 SW4 对应的芯片的管脚号为 48、49、50 和 51，这些开关被 10 K 的电阻下拉，每按一次有对应红色的 LED 闪烁一次，这四个开关都加有阻容消抖电路。当开关按下时为逻辑电平为 1，弹起时为逻辑电平为 0。

## 8. 八位的 DIP 封装开关

开关打开时逻辑电平为 1，关闭时为 0。下表从左到右依次列出了对应的芯片管脚号：

DIP 开关	1	2	3	4	5	6	7	8
相连管脚	84	81	80	79	77	76	75	52

## 9. 发光二极管 LED

电路板上每个 LED 均接有 330 的限流电阻上。所有的二极管在对应的插孔式插头被置为逻辑 1 时点亮。其中下排最右边两组黄、绿、红三个 LED 从作到右分别依次对应管脚 39、40、41 和 44、45、46，可用于交通灯程序的实验仿真。

## 10. 七段数码管（这部分内容来源网上，程序自编）

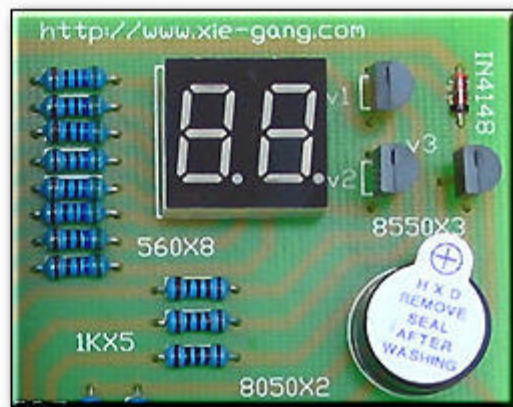
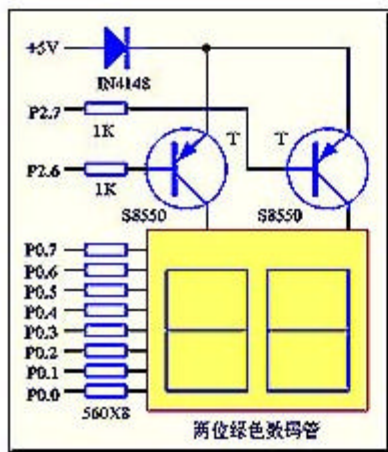
如何利用实验板上的两个数码管来做一个循环显示 00 ~ 99 数字、99 ~ 00 的实验。

数码管有共阴和共阳的区分，都可以加入我们进行驱动，但是驱动的方法却不同，并且相应的 0~9 的显示代码也正好相反。

**静态扫描接法：是指每个数码管的位码和字码都单独由不同信号控制，这样占用的 I/O 很多，比如 8 位数码管就要占用 64 个 I/O 口（这样通过 CPLD 电流也将很大）。**

**动态扫描接法：**动态扫描显示接口是最为广泛的一种显示方式之一。其接口电路是把所有显示器的 8 个笔划段 a-h 同名端连在一起，而每一个显示器的公共极 COM 是各自独立地受 I/O 线控制。CPU 向字段输出口送出字形码时，所有显示器接收到相同的字形码，但究竟是那个显示器亮，则取决于 COM 端，而这一端是由 I/O 控制的，所以我们就可以自行决定何时显示哪一位了。而所谓动态扫描就是指我们采用分时的方法，轮流控制各个显示器的 COM 端，使各个显示器轮流点亮。在轮流点亮扫描过程中，每位显示器的点亮时间是极为短暂的（约 1ms），但由于人的视觉暂留现象及发光二极管的余辉效应，尽管实际上各位显示器并非同时点亮，但只要扫描的速度足够快，给人的印象就是一组稳定的显示数据，不会有闪烁感。**（简单地说，一个时刻只有一个或部分数码管亮，另一个时刻只有另一个或或另一部分数码管亮）**

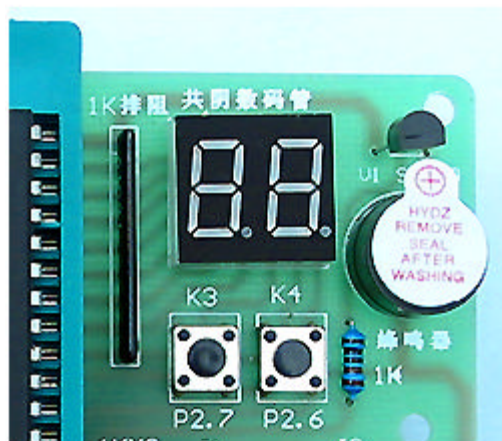
首先我们来介绍两位共阳数码管的单片机驱动方法，电路如下图：



P2.6 和 P2.7 端口分别控制数码管的十位和个位的供电，当相应的端口变成低电平时，驱动相应的三极管会导通，+5V 通过 IN4148 二极管和驱动三极管给数码管相应的位供电，这时只要 P0 口送出数字的显示代码，数码管就能正常显示数字。

因为要显示两位不同的数字，所以必须用动态扫描的方法来实现，就是先个位显示 1 毫秒，再十位显示 1 毫秒，不断循环，这样只要扫描时间小于 1/50 秒，就会因为人眼的视觉残留效应，看到两位不同的数字稳定显示。

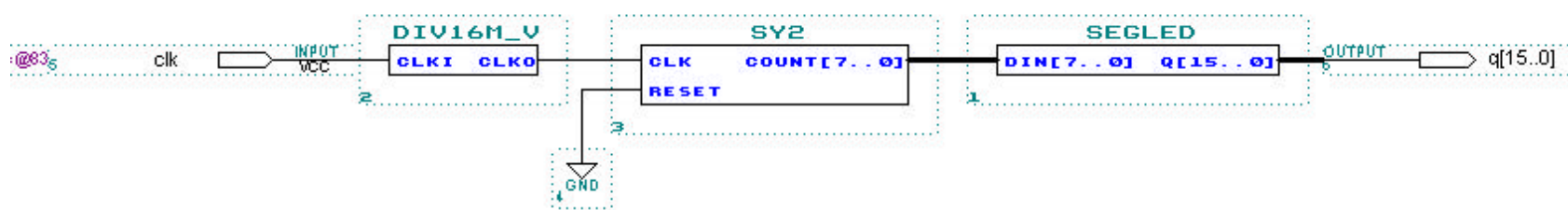
下面我们再介绍一种共阴数码管的单片机驱动方法，电路如下图：



可以看到 +5V 通过 1K 的排阻直接给数码管的 8 个段位供电，P2.6 和 P2.7 端口分别控制数码管的十位和个位的供电，当相应的端口变成低电平时，相应的位可以吸入电流。单片机的 P0 口输出的数据相当于将数码管不要显示的数字段对地短路，这样数码管就会显示需要的数字。

可以看到，共阴数码管的硬件更简单，所以在批量生产时，硬件开销小，节省 PCB 面积，减少焊接工作量，降低综合成本，所以采用共阴数码管更有利于批量生产，现在销售的试验板都是采用共阴数码管了。

程序：



```
library ieee;

    use ieee.std_logic_1164.all;

entity sy2 is
    port(
        clk,reset:in std_logic;
        count: out integer range 0 to 128);
end sy2;

architecture a of sy2 is
    signal con:std_logic;
begin
    process(clk,reset)
        variable coun:integer range 0 to 99;
    begin
        if reset='1' then
            coun:=0;
        elsif clk='1' and clk'event then
```

```
case con is
    when '0' =>
        if coun=99 then
            coun:=98;
            con<='1';
        else
            coun:=coun+1;
        end if;
    when '1' =>
        if coun=0 then
            coun:=1;
            con<='0';
        else
            coun:=coun-1;
        end if;
    when others => NULL;
end case;
end if;
```

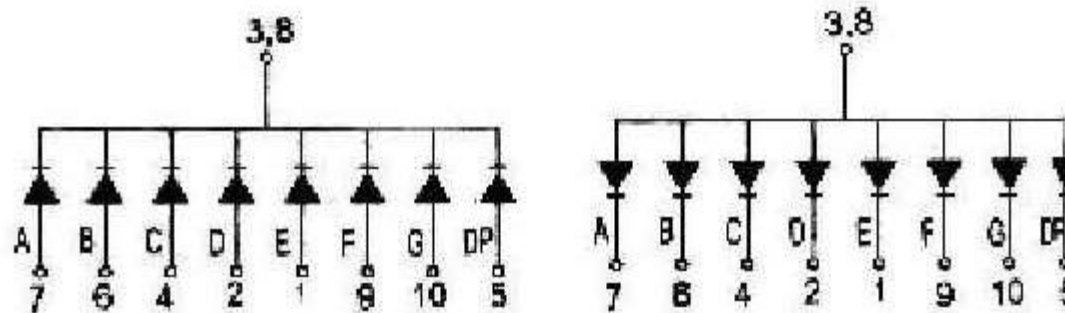
```

count<=count;
end process;
end a;

```

LED 只能显示几位的状态，要显示“0-9”、“A-F”和一些特殊字符，可选用数码管。数码管分共阳（4105）和共阴（4205）两种。段数码管的段排列和内结构见下图：

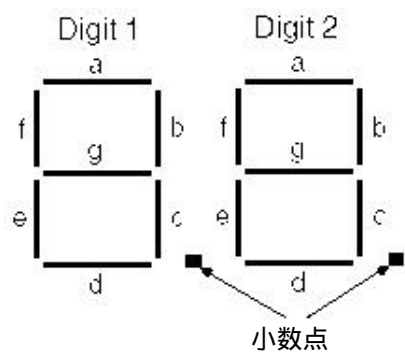
段排列：



**内部接线图**，上图中段数码管左为共阴，右为共阳。

两个共阴七段数码管，每段发光二极管都由 EPM7128S 芯片对应的 I/O 脚置零来点亮。下图中显示了每段发光二极管

的名称。



七段数码管中的发光二极管名称

下表中列出了数码管相连的 I/O 管脚号：**数码管与 I/O 管脚的连接**

代码管	LED_7_1 相连管脚	LED_7_2 相连管脚
a	74	63
b	73	61
c	70	60
d	69	58
e	68	57
f	67	56
g	65	55
小数点	64	54



## 11 . 配置 EPM7128S

源程序或原理图经过编译仿真无误后，指定好器件（EPM7128SLC84-15），分配好管脚，就可以按照下面的步骤采用 JTAG 链方式配置 EPM7128S，这里详细介绍了怎样使用 MAX+PLUS II 软件来进行此项操作，更多信息请参阅 MAX+PLUS II 的帮助：

- 选中“Multi-Device JTAG Chain”选项（JTAG 菜单下）；
- 选择“Multi-Device JTAG Chain Setup”命令（JTAG 菜单下）；
- 在“Multi-Device JTAG Chain Setup”对话框中的“Device Name”列表中选择 EPM7128S；
- 在“Programming File Name”框中输入要配置到 EPM7128S 芯片的文件名。可以使用按钮“Select Programming File”来浏览电脑中的目录结构，选择适当的配置文件；
- 点击“Add”来添加设备和相应的编译文件到“Device Names & Programming File Names”框中。芯片名称（Device Name）左边的数字表示了 JTAG 链中的芯片序号。配置用文件的文件名显示在芯片名称的同一行。如果芯片还没有制定配置文件，芯片名称旁边的响应位置显示“<none>”；
- 点击“Detect JTAG Chain Info”，通过 ByteBlasterMV 电缆检测芯片数目、JTAG ID 代码，JTAG 链中总共的指令长度。在按钮“Detect JTAG Chain Info”上方的一条消息报告这些被 ByteBlasterMV 电缆检测到的信息。如果和“Device Name & Programming File Names”框中的内容不一致，必须重新手动设置设备信息以保证一致；
- 在“Save JCF”对话框中点击“Save JCF”，在“File Name”框中填写上文件名，然后在“Directories”框中输入适当的目录名，保存现在的 JTAG 链设置到设置文件（JTAG Chain File：\*.jcf），以便将来使用。

最后点击“OK”；

- 点击“OK”来保存改动；
- 点击 MAX+PLUS II “Programmer” 对话框中的“Program”按钮，即可。

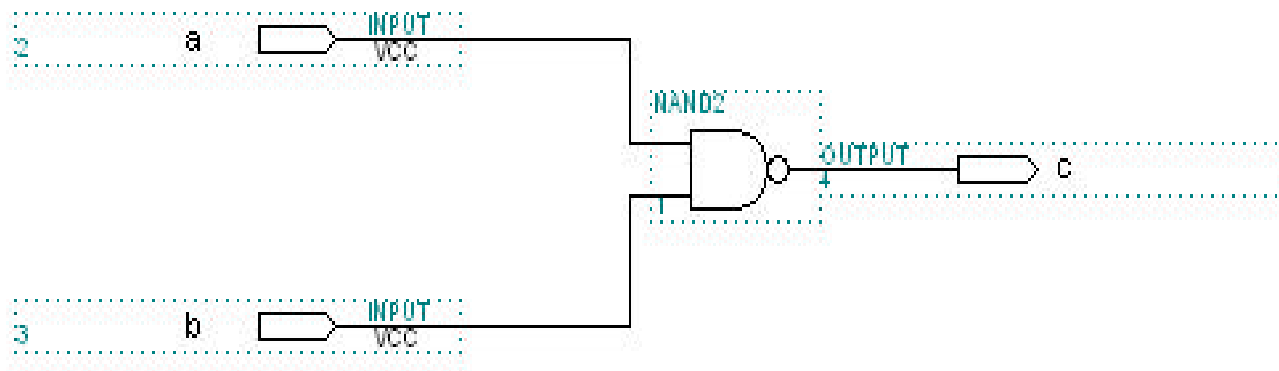
## ➡ 12. 范例演示



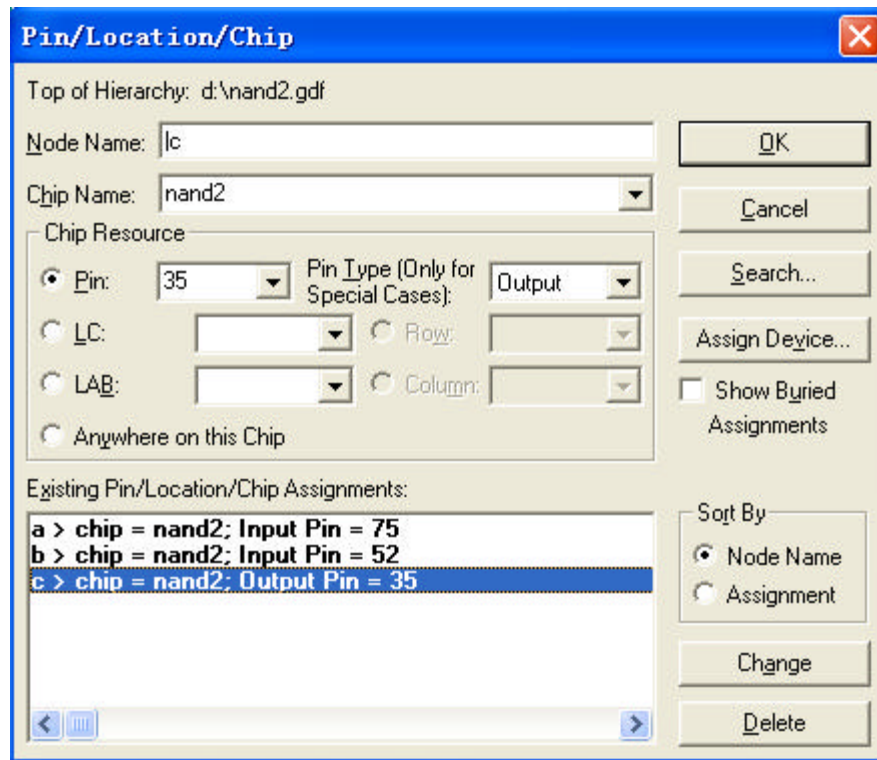
*提示：开发板的并口和电源必须要在计算机开机之前连接好，不要在开机后进行接口和电源的连接。*

### ● 组合逻辑


以简单的二输入与非门为例：此例采用原理图输入方式，见下图：

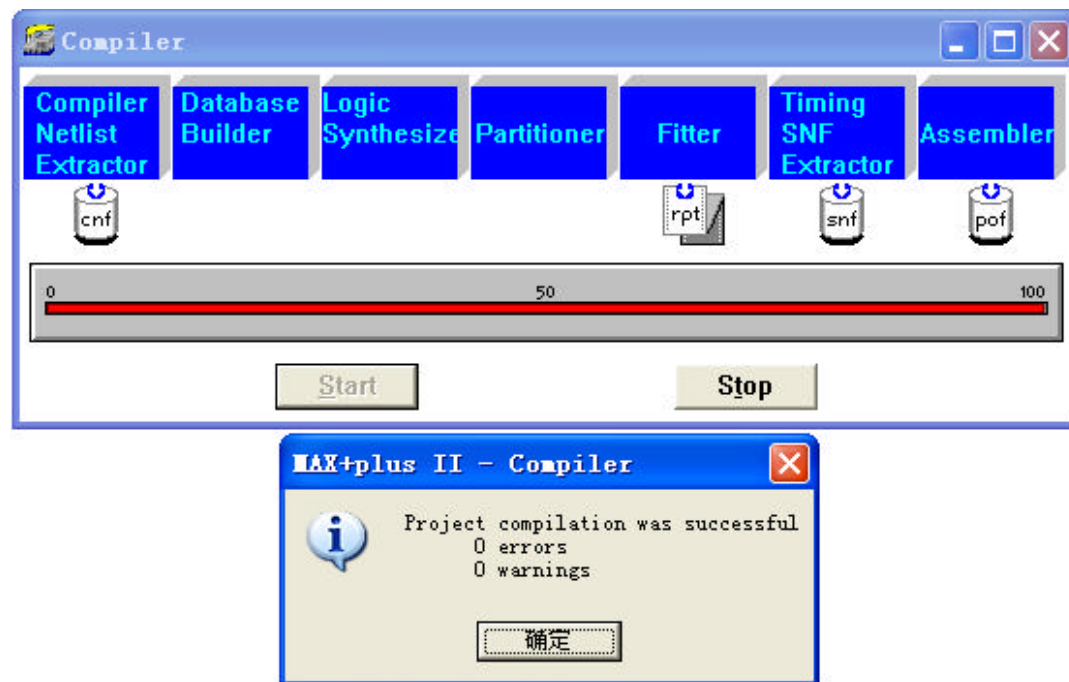


绘制完毕后,点击 Assign 菜单下的 Pin/Location/Chip,在弹出的窗口中点击 Assign Device 按钮,由于开发板上的 7128S 芯片的速度级别不高,需要将“Show Only Fastest Speed Grades”前面的钩给去掉,在“Device Family”中选择“MAX7000S”,再从下面的 Devices 中选择 EPM7128SLC84-15,点击 OK。

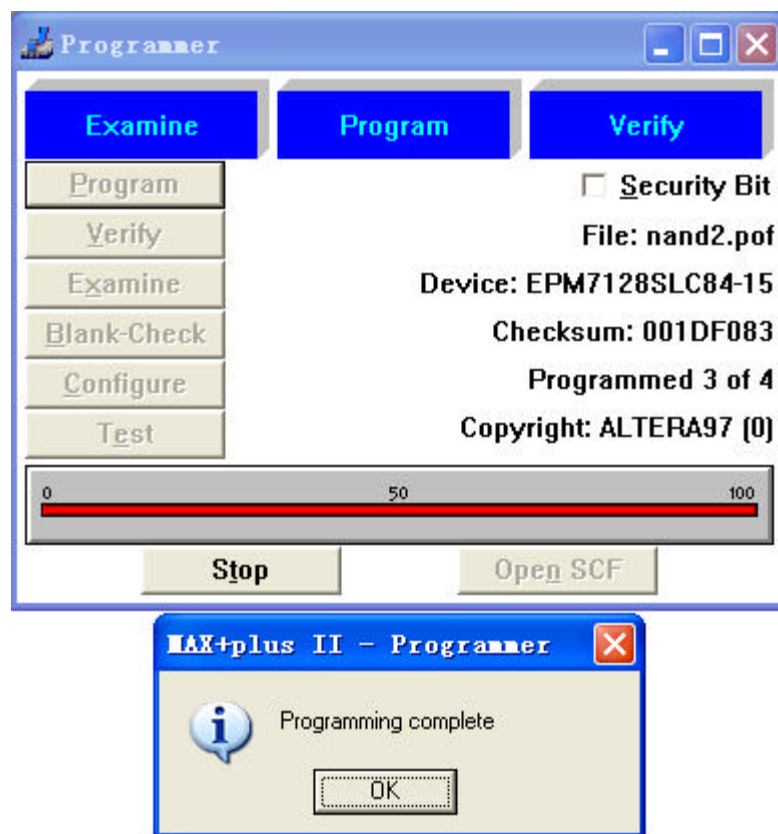


在 Node Name 中输入原理图管脚的名称,指定管脚和管脚的输入输出类型,如上图中将输出(Output)c 指定为 35 号管脚,指定完毕后,点击 OK 即可进行编译了。

在编译之前需将待编译的文件设置为当前文件，可以快捷键 Ctrl+Shift+J，也可以点击 File 菜单 Project 下的“Set Project to Current File”，这是一个很好的习惯，尤其在打开多个 project 后，这一步千万不能忽略。点击 ，进行编译，当然也可以通过选择 MAX+PLUSII 菜单下的 Compiler 进行编译，如下图：



确定编译无误，点击上图中的 ，此时可以通过点击 Options 菜单下的 Hardware Setup 来查看硬件类型的状态，这里



为 ByteBlasterMV 下载方式。点击 Program 即可进行下载，

下载完毕后，即可通过控制 a、b 对应的 75、52 管脚所连接的 DIP 开关的闭合进行逻辑验证。

## • 时序逻辑

以 D 触发器为例：此例采用 VHDL 文本输入方式，代码如下：

```
library ieee;
use ieee.std_logic_1164.all;

entity e_dff is
    port(
        d,clk: in std_logic;
        q: out std_logic);
end e_dff;

architecture e_dff of e_dff is
    begin
        process(clk,d)
        begin
            if clk'event and clk='1' then
                q<=d;
            end if;
        end process;
    end e_dff;
```

采用同上的方法进行编译、指定器件、分配管脚和下载，再进行逻辑验证，发现逻辑“不正确”，原因就在由于直接使用了板载时钟 20MHz，我们的肉眼是无法分辨 LED 的闪烁的，因此需要对板载时钟进行分频，具体方法见下面的“复杂数字系统”。

## • 复杂数字系统

以交通灯系统为例：

具体的 VHDL 代码请参见教材《VHDL 与复杂数字系统设计》一书。上面已经提到，需要对板载时钟分频，考虑到肉眼识别能力，设计一个 24 位的计数器，利用其最高位作为交通灯系统的时钟信号，具体内容和相关代码请参见教材。



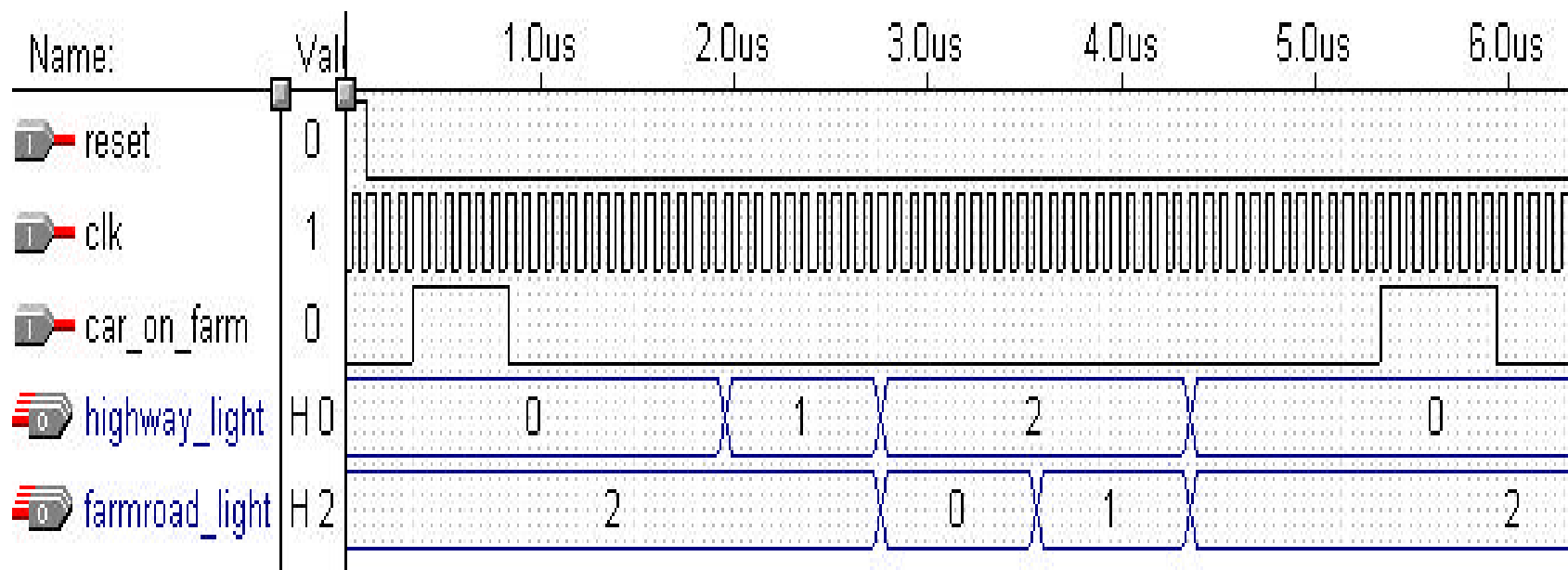
*提示：在仿真时切不可用计数器的第 24 位作为系统的时钟，要用第一位作为其时钟，因为若用第 24 位，仿真时间将会很长。*


因为系统比较复杂，我们先对没有进行时钟分频的交通灯控制系统进行编译、仿真，来观察一下各个逻辑状态。

编译通过后，点击 MAX+PLUSII 菜单下的 Waveform Editor，出现波形编辑窗口，在空白处，点击鼠标右键，选择 Enter Nodes from SNF，在弹出的窗口中点击 List，在

Available Nodes&Groups 中选择 input、output 或是 registered 这些输入输出信号，点击 OK 即可，然后保存该波形文件。

通过 File 菜单下的 End Time 设定仿真时间为 10us 通过 Options 菜单下的 Grid Size 设定为 50ns , 并且选择 Show Grid。



如上图设定好 reset、clk 和 car\_on\_farm 的值后保存文件，点击  按钮，将 Setup/Hold 选上，点击 Start 进行仿真，仿真完毕后即得上图输入输出的波形关系。



**提示：仿真时最好将 Setup/Hold 选上，即考虑信号的建立时间和保持时间。**

这里的 0、1、2 分别代表绿灯、黄灯和红灯，显然是满足我们的设计要求的。



下面就要进行下载验证了，前面已经说明，下载到开发板上的时钟要进行分频，此外还要把交通灯外围显示模块一同加入进行编译、下载，这些与显示有关的模块包括一个 5 进制的减法计数器、一个 10 进制的减法计数器和七段数码管显示的驱动程序，以及加入特殊情况（sp）下的交通灯控制。

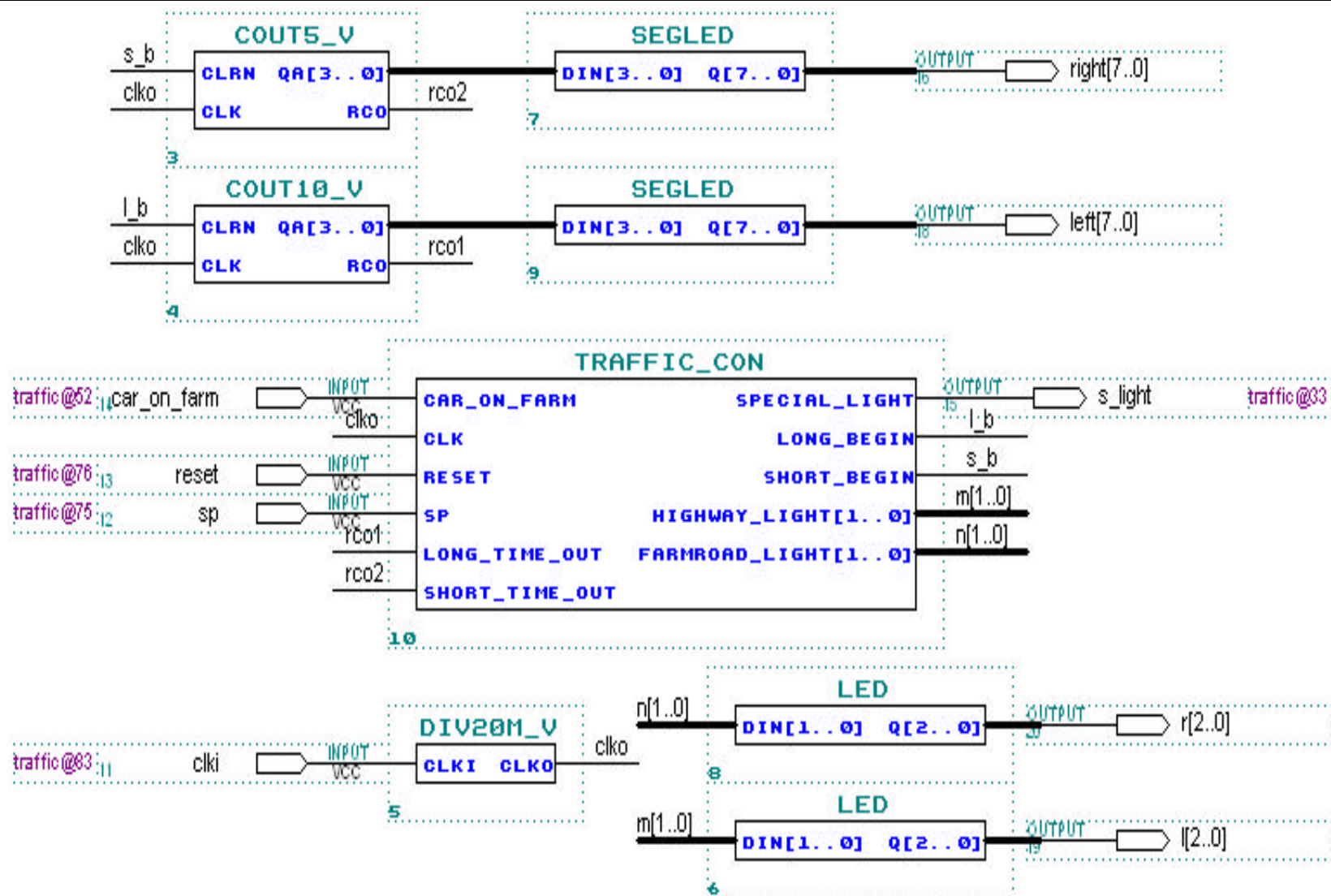


*提示：这里为了便于在开发板上验证逻辑，加之只有两个数码管，所有采用比较低的进制。*


将交通灯系统的控制模块、显示模块以及各个计数器编译仿真通过后，考虑到教材上是先建立一个 package，再利用 port map 进行设计输入的，在这里我们则用另一种输入的方法——原理图输入，这样可以清楚到看到各个端口的连接情况。

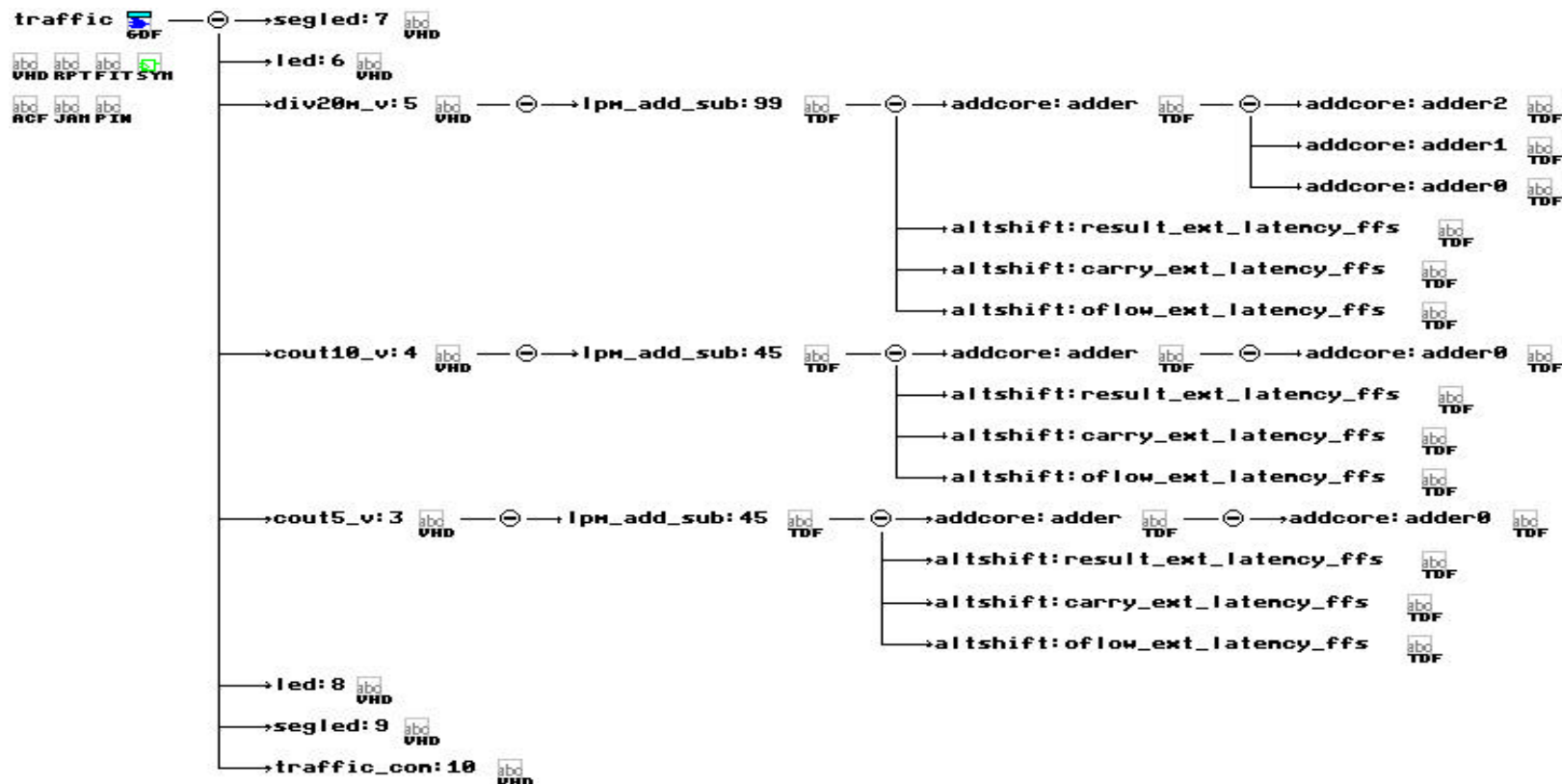
因为每个编译过的.vhd 完整的程序都会自动生成 Symbol 文件，新建一个.gdf 文档，双击空白处，在弹出的窗口中找到正确的路径后，即可在左边 Symbol Files 中选择需要的 Symbol。


电路原理图编辑完毕后，接下来就是选择器件和分配管脚了，在这个范例中电路原理图和管脚的分配如下所示：





[illegible]

编译通过后，按照前面所述的步骤进行下载即可。可以看出，high\_way 和 farm\_road 的三种颜色得灯的交替变化与数码管的计时显示都是正确的，这样，这个比较复杂的交通灯系统的设计就完成了。我们可以通过点击主菜单的  按钮，来查看此交通灯系统的层次显示：



双击左上角的  **rpt**，可以查看该设计的 report，reprot 中包含了很多信息，有管脚的分配情况、各个 microcell 的使用率、管脚的使用率、编译过程中各个步骤所用的时间以及整个器件的使用率，此交通灯系统对这个 7128S 芯片的使用率为 52%。

点击主菜单的  按钮，可以查看该设计的延迟矩阵，查看完毕后，再点击主菜单的  按钮，可以进行电路的性能分析。

至此，该交通灯系统的下载和分析就结束了，当然，还可以电路的性能进行优化，如采用 Clique 的打包功能等，这里就不再赘述了。

## ➡ ● 注意事项

- ✓ 开发板与计算机连接时，要在关机状态下，先接并口，再接开发板电源，再开机。
- ✓ 开发板与计算机分离时，先将开发板电源断开，再将并口卸下。
- ✓ 由于此开发板将所有的 I/O 都用 LED 引出，故不要将那些没有使用的 I/O 管脚所对应的插孔式接头用短路帽连接上，以避免其它 LED 的闪烁影响了逻辑的验证。
- ✓ 使用开发板时要注意将其放在绝缘、干燥的平面上。
- ✓ 开发板长期不使用时，要将其放在干燥阴凉处，防止器件老化。

## 4.5 基于 MAX+PLUS II 设计交通灯控制器

**编者按 :**在第二章的课堂上我们以自然语言描述开始 ,分 12 个要点讲述了 VHDL 设计的方法和流程。在本章主要以任务为驱动的教学方法来描述 ,它以完成具体的任务为目的 ,例如外设中 LED、数码管、LCD、触摸屏就是一个相似以显示为目的而不断提出的任务 ,我们可以根据它们各自不同原理进行设计。随着设计水平的不断提高就会积累相应的知识点。以一个相对较复杂的状态机起步 ,可以到达..... ? ? ?

在这小节中,我们将按照 Top-Down 的设计流程,利用 MAX+PLUS II,在 CPLD 器件 EPM7128SLC84-15 实现交通灯系统。由于交通灯系统是一个非常具有代表性的时序逻辑数字系统,时序明确,容易模块化,对提高系统设计能力有很大帮助。

### 1. 系统功能要求分析和设计方案论证

按照 Top-Down 设计流程,首先要进行系统的功能要求分析。

我们假设为某乡村公路(以下简称为 f)和高速公路(以下简称为 h)十字路口设置交通灯。要求两公路均有红黄绿三色灯和。保证 h 绝对优先,初状态 h 绿灯, f 红灯。正常状态下,当监测 f 上有车时,整个系统启动,经过 t1 时间以后, h 转为黄灯,再经过 t2 时间以后, h 转为红灯, f 转为绿灯。如在以上时间段内, f 上的车绕它道行驶,则直接跳变回初状态。在 h 红灯、f 绿灯状态下,如果监测到 f 上没车,或者已达到最长通行时间 t3,则 f 转为黄灯,再经过 t2 时间以后, h 转为绿灯、f 转为红灯;紧急状态下紧急状态显示灯开启,只允许 h 通车。

根据系统功能要求,确定系统的输入信号和输出信号。

输入信号有:系统时钟信号 clk,系统复位信号 reset,紧急状态信号 sp,乡村公路有车信号 car\_on\_farm。

输出信号有：乡村公路状态信号 farmroad\_light[1..0]，高速公路状态信号 highway\_light[1..0]以及紧急状态显示信号 special\_light。

根据以上设定的输入输出信号关系和对系统功能要求的分析，可建立如图4-4所示的系统的**状态转换图(ASM)**。

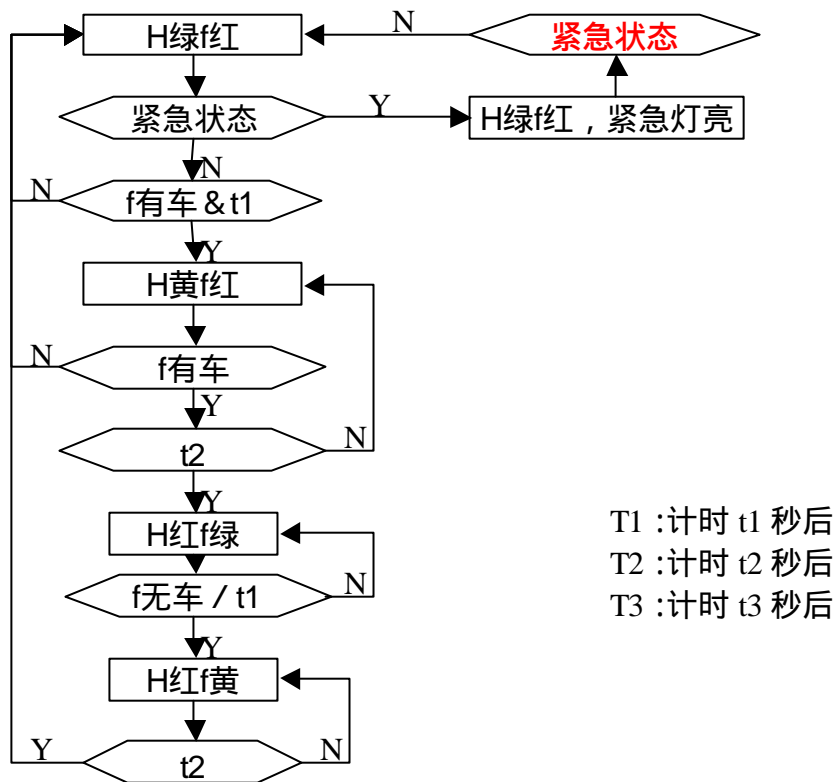


图 4-4 交通灯系统状态图

由系统的状态图可以看出，交通灯包含了四个正常状态，一个紧急状态。**在一定条件下实现状态转换。**

## 2.系统顶层的 VHDL 描述和仿真

状态图分析完毕以后，就可以根据它编写最顶层的 VHDL 源程序。并进行编译仿真。具体过程如下：

- (1)启动 MAX+PLUSII,建立新文件，进入文字编译。
- (2)保存文件：保存为 traffic\_con.vhd。
- (3)选定项目名与文件名相同（ Set Project To Current File ）。
- (4)输入 VHDL 源程序。源程序如下

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity traffic_con is
port (car_on_farm      :in      boolean;
      clk,reset,sp      :in      std_logic;
      highway_light    :out     std_logic_vector(1 downto 0); farmroad_light  :out     std_logic_vector(1 downto 0);
      special_light    :out     std_logic
      );
end traffic_con;

architecture a of traffic_con is
  signal high_state      :std_logic_vector(1 downto 0) ;
  signal farm_state      :std_logic_vector(1 downto 0) ;
  signal present_state   :std_logic_vector(1 downto 0) ;
begin
  process (clk)
    variable m ,g :integer range 0 to 30 :=30;
    variable n :integer range 0 to 10 :=10;          begin
    if reset='1' then
```



```
present_state<="00"; high_state<="00";
farm_state<="10";m:=30;n:=10;
elsif clk'event and clk='1' then
  case present_state is
    WHEN "00" =>
      if SP='1' then
        special_light<='1';
      else
        special_light<='0';
        if car_on_farm then
          m:=m-1;
        end if;
        if m=0 then
          present_state<="01";high_state<="01";m:=30;
        end if;
      end if;
      when "01"=>
        if not car_on_farm then
          present_state<="00";high_state<="00";n:=10;
        else
          n:=n-1;
        end if;
        if n=0 then
          present_state<="10";high_state<="10";
          farm_state<="00";n:=10;
        end if;
      end if;
    END IF;
```

```

    WHEN "10" =>
        m:=m-1;
    if not car_on_farm or m=0 then
        present_state<="11";farm_state<="01"; m:=30;
    end if;
    WHEN OTHERS =>
        n:=n-1;
    if n=0 then
        present_state<= "00";high_state<= "00";
        farm_state<="10";n:=10;
    end if;
    END CASE;
end if;
end process;
highway_light<=high_state;
farmroad_light<=farm_state;
end a;

```

源程序输入完毕之后就可进行编译了。

(5) **指定设计器件**：选取窗口菜单 Assign— Device，出现对话框，选择 MAX7000S 系列 EPM7128SLC84-15。

(6)保存并检查：选取窗口菜单 File— Project— Save&check，即可对电路设计文件进行检查。如有错误，会自动指示错误位置，以便修改。

(7)保存并编译：选取 File— Project— Save&compile，即可进行编译。图 4-5 中有“compile”对话框即为编译框。

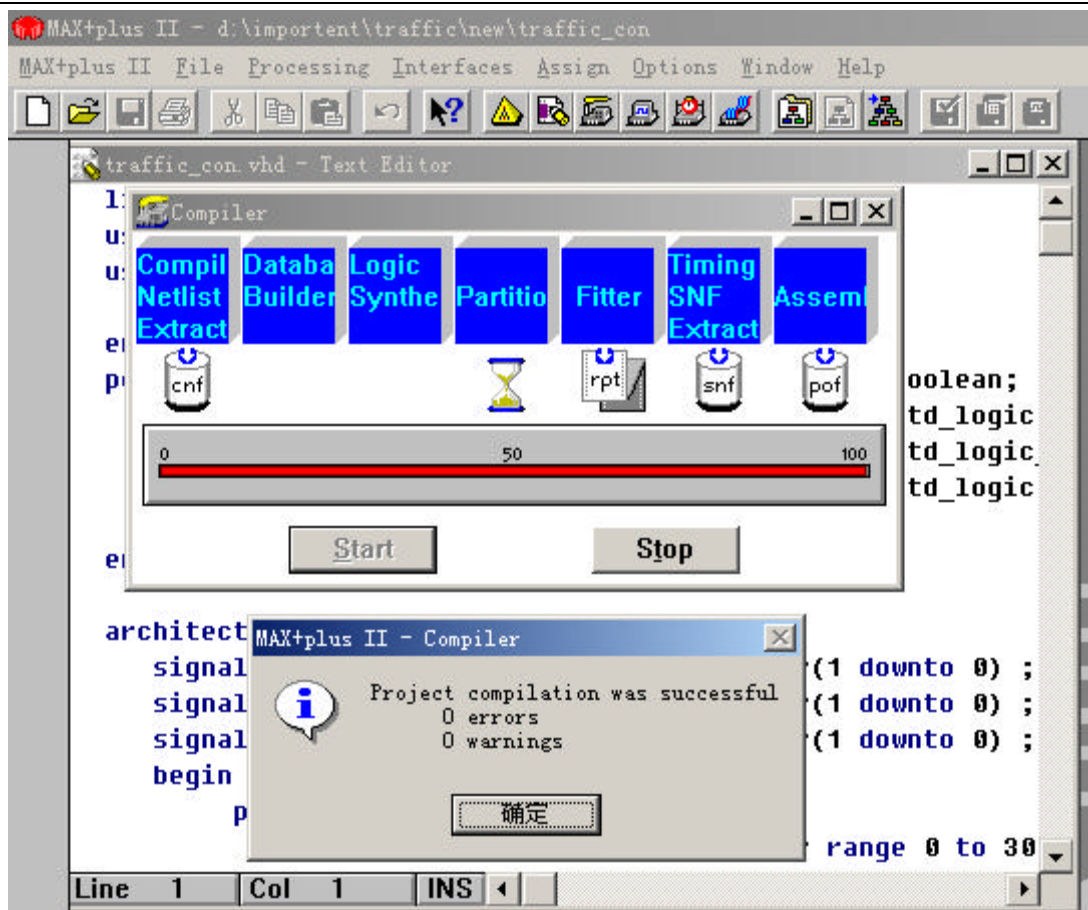


图 4-5 MAX+PLUSII 编译图示

(8)编译通过后，就可进行仿真了。从 Top-Down 设计方法来看，各个层次上的仿真波形分析必须保持一致。

### 3. 8M 分频程序

由于本次试验板上提供的时钟是 16M 的，所以我们还需要一个 8M 分频。其 VHDL 源程序如下：

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY div8M_v IS
    PORT(CLKI  : IN  STD_LOGIC;
         CLKO  : OUT STD_LOGIC
        );
END div8M_v ;
ARCHITECTURE a OF div8M_v IS
    SIGNAL cou :    STD_LOGIC_VECTOR(22 DOWNT0 0);
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL CLKI = '1';
        cou <= cou+1;
    END PROCESS;
    CLKO <= cou(22);
END a;

```

同样需要编译、仿真通过。

#### 4. 交通灯控制器的顶层模块

在主程序中调用上两个模块，就构成了交通灯控制器的顶层模块。主程序 VHDL 描述如下：

```

library ieee;
use ieee.std_logic_1164.all;

```

```

use ieee.std_logic_unsigned.all;

entity traffic is
port (car_on_farm           :in      boolean;
      clk,reset,sp         :in      std_logic;
      highway_light :out      std_logic_vector(1 downto 0);
      farmroad_light  : out      std_logic_vector(1 downto 0);
      special_light    :out      std_logic );
end traffic;

architecture a of traffic is
  signal clka :std_logic;
  COMPONENT div8m_v
    PORT(
      clki    : IN  STD_LOGIC;
      clko    : OUT  STD_LOGIC);
  END COMPONENT;
  component traffic_con
    port(car_on_farm :in  boolean;
         clk,reset,sp :in  std_logic;
         highway_light :out  std_logic_vector(1 downto 0);
         farmroad_light :out  std_logic_vector(1 downto 0); special_light :out  std_logic );
  end component;
begin
  n1 : div8m_v
    port map (clki=>clk,clko=>clka);
  n2 : traffic_con

```

```
port map (car_on_farm=>car_on_farm,clk=>clka,  
reset=>reset,sp=>sp,special_light=>special_light, highway_light=>highway_light,  
farmroad_light=>farmroad_light);  
end a;
```

然后再编译和仿真，生成顶层模块。图 4-6 为顶层模块示意图。



图 4-6 交通灯系统顶层模块示意图。

## 5. 系统的模块划分及实现

按照 Top-Down 层次化设计要求，我们需要对顶层系统进行模块划分。详见图 4-7。

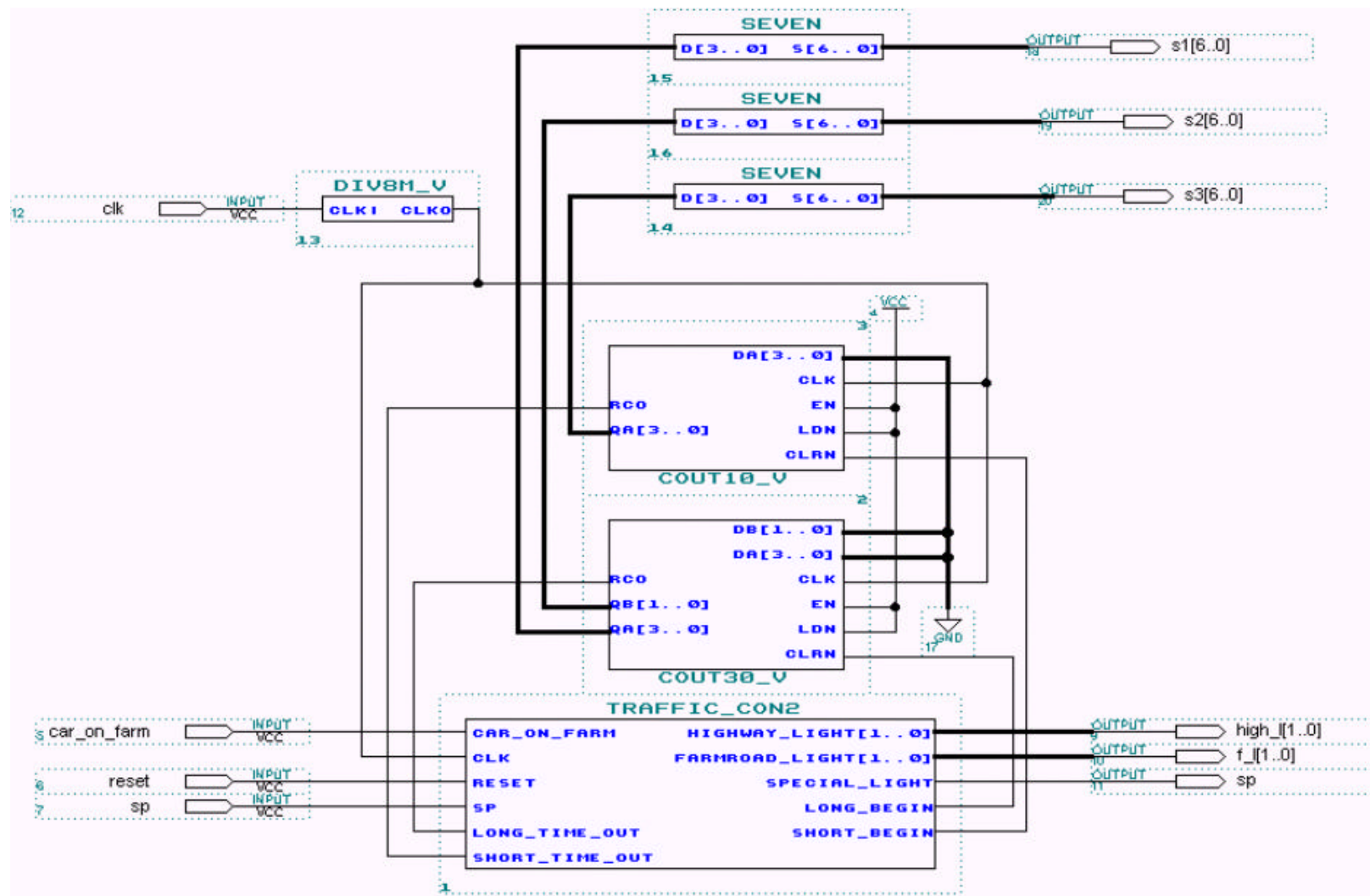


图 4-7 交通灯系统的模块划分

从图 4-7 可见，交通灯系统具体分为以下几个部分：交通灯**控制器**——TRAFFIC\_CON2，一个 8M**分频器**——DIV8M\_V，两个**减法计数器**（十、三十进制）COUNT10\_V,COUNT30\_V。划分好模块以后，我们就可以将各个具体模块分别的**编译仿真**，具体过程和上边顶层模块大同小异，在此不赘述。

#### 6. 主要部分的 VHDL 源程序

##### (1)十进制减法计数器 COUNT10\_V

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY cout10_v IS
PORT(CLRN,CLK : IN  STD_LOGIC;
      Qa      : OUT  STD_LOGIC_VECTOR(3 downto 0);
      RCO     : OUT  STD_LOGIC );
END cout10_v;

ARCHITECTURE a OF cout10_v IS
BEGIN
PROCESS (Clk)
VARIABLE tmpa ,tmpb:STD_LOGIC_VECTOR(3 downto 0);
BEGIN
    tmpb:=not tmpa;
    IF CLRN='0' THEN  tmpa := "1010";
    ELSIF (Clk'event AND Clk='1') THEN
        IF tmpa="0000" THEN
```



```

        tmpa:="1001";
    ELSE
        tmpa := tmpa-1;
    END IF;
END IF;
Qa <= tmpa;
RCO<= tmpb(0)and tmpb(1) and tmpb(2) and tmpb(3) ;
END PROCESS ;

```

END a;

(2)三十进制减法计数器(COUNT30\_V)

```

library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY cout30_v IS
    PORT(CLRN,CLK  : IN  STD_LOGIC;
          Qa      : OUT  STD_LOGIC_VECTOR(3 downto 0);
          Qb      : OUT  STD_LOGIC_VECTOR(1 downto 0);
          RCO     : OUT  STD_LOGIC  );
END cout30_v;
ARCHITECTURE a OF cout30_v IS
    BEGIN
        PROCESS (Clk)
            VARIABLE tmpa,tmpc:STD_LOGIC_VECTOR(3 downto 0);
            VARIABLE tmpb,tmpd:STD_LOGIC_VECTOR(1 downto 0);
        BEGIN
            tmpc:=not tmpa;      tmpd:= not tmpb;
            IF CLRN='0' THEN  tmpb := "11"; tmpa := "1010";

```

```

    ELSIF (Clk'event AND Clk='1') THEN
        IF tmpa="0000" THEN
            tmpa:="1001";
            IF tmpb="00" THEN tmpb:="10";
            ELSE tmpb:= tmpb-1;
            END IF;
        ELSE tmpa := tmpa-1;
        END IF;
    END IF;
    Qa <= tmpa; Qb <= tmpb;
    RCO<= tmpd(1) and tmpd(0) AND tmpc(0) AND tmpc(3) and tmpc(1)and tmpc(2) ;
    END PROCESS ;
END a;

```

### (3)交通灯控制模块程序

TRAFFIC\_CON2.VHD

```

use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity traffic_con2 is
    port (car_on_farm      :in      boolean;
          clk,reset,sp,long_time_out,short_time_out  :in  std_logic;
          highway_light    :out      std_logic_vector(1 downto 0); farmroad_light    :out      std_logic_vector(1 downto 0);
          special_light,long_begin,short_begin  :out      std_logic );
end traffic_con2;
architecture a of traffic_con2 is
    signal high_state,farm_state :std_logic_vector(1 downto 0) ;
    signal present_state :std_logic_vector(1 downto 0) ;
begin

```

```

process (clk)
begin
    if reset='1' then
        present_state<="00";high_state<="00";farm_state<="10";
    elsif clk'event and clk='1' then
        case present_state is
            WHEN "00" =>
                if SP='1' then
                    special_light<='1';
                else special_light<='0';
                    if car_on_farm then
                        long_begin<='1';
                    else long_begin<='0' ;
                        end if;
                    if long_time_out='1' then
                        present_state<="01"; high_state<="01"; long_begin<='0';
                    end if;
                end if;
            when "01"=>
                if not car_on_farm then
                    present_state<="00"; high_state<="00";
                else short_begin<='1';
                    end if;
                if short_time_out='1' then
                    present_state<="10";
                    high_state<="10";farm_state<="00";
                    short_begin<='0';
                END IF;
            end if;
        end case;
    end if;
end process;

```

```

    WHEN "10" =>
        long_begin<='1';
        if not car_on_farm or long_time_out='1' then
            present_state<="11";
            farm_state<="01"; long_begin<='0';
        end if;
    WHEN OTHERS =>
        short_begin<='1';
        if short_time_out='1' then
            present_state<= "00";
            high_state<= "00"; farm_state<="10";
            short_begin<='0';
        end if;
    END CASE;
end if;
end process;
highway_light<=high_state;
farmroad_light<=farm_state;
end a;

```

#### (4)包文件源程序

各模块编译完成之后，就需要在交通灯系统主程序中调用，所以需要创建一个**包文件**。源程序如下：

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
PACKAGE traffic_package1 IS
    COMPONENT cout30_v

```

```

    PORT(CLRN,CLK: IN  STD_LOGIC;
          RCO      : OUT STD_LOGIC );
END COMPONENT;
COMPONENT cout10_v
    PORT(CLRN,CLK  : IN  STD_LOGIC;
          RCO      : OUT STD_LOGIC );
END COMPONENT;
COMPONENT DIV8M_V
    PORT(CLKI  :IN   STD_LOGIC;
          CLKO  : OUT  STD_LOGIC);
END COMPONENT;
Component traffic_con2
port(car_on_farm      :in      boolean;
clk,reset,sp,long_time_out,short_time_out :in std_logic;
highway_light,farmroad_light
                                :out      std_logic_vector(1 downto 0); special_light,long_begin,short_begin :out std_logic );
end component;
END traffic_package1;

```

#### (5)主程序

主程序如下：

```

library ieee;use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;use work.traffic_package1.all;
entity traffic2 is
    port (car_on_farm      :in      boolean;
clka,reset,sp      :in      std_logic;
          highway_light,farmroad_light  :out      std_logic_vector(1 downto 0);
special_light      :out      std_logic      );

```

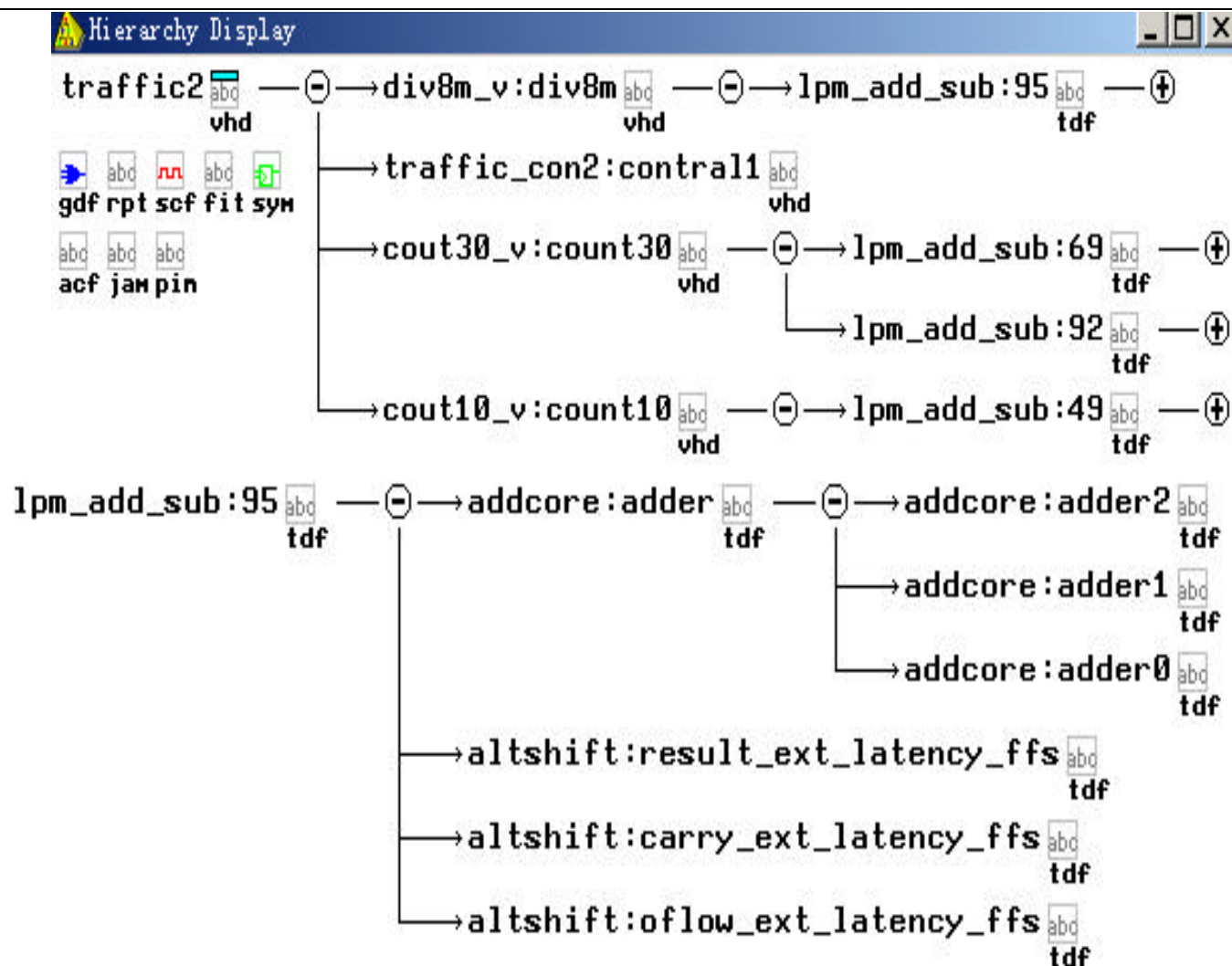
```

end traffic2;
architecture contain of traffic2 is
signal  lb,sb,lto,sto ,clk  :std_logic;
begin
    div8m :      div8m_v
        port map (clki=>clka,clko=>clk);
    contral1:      traffic_con2
port map (car_on_farm=>car_on_farm,clk=>clk, sp=>sp,
reset=>reset, long_time_out=>lto,short_time_out=>sto
highway_light=>highway_light,
farmroad_light=>farmroad_light, long_begin=>lb,
        special_light=>special_light, short_begin=>sb);
    count30:      cout30_v
port map (clrn=>lb,clk=>clk,rco=>lto);
    count10:      cout10_v
        port map (clrn=>sb,clk=>clk,rco=>sto);,
end contain;
7.烧写文件

```

将以上文件再次进行完全编译。产生烧写文件(.pof)、电路包含文件(.inc)等。

选择 MAXPLUS II 中的 Hierarchy Display 项，我们可以以层次树的方式显示整个项目和电路的设计文件。交通灯控制器的层次树如图 4-8 所示。



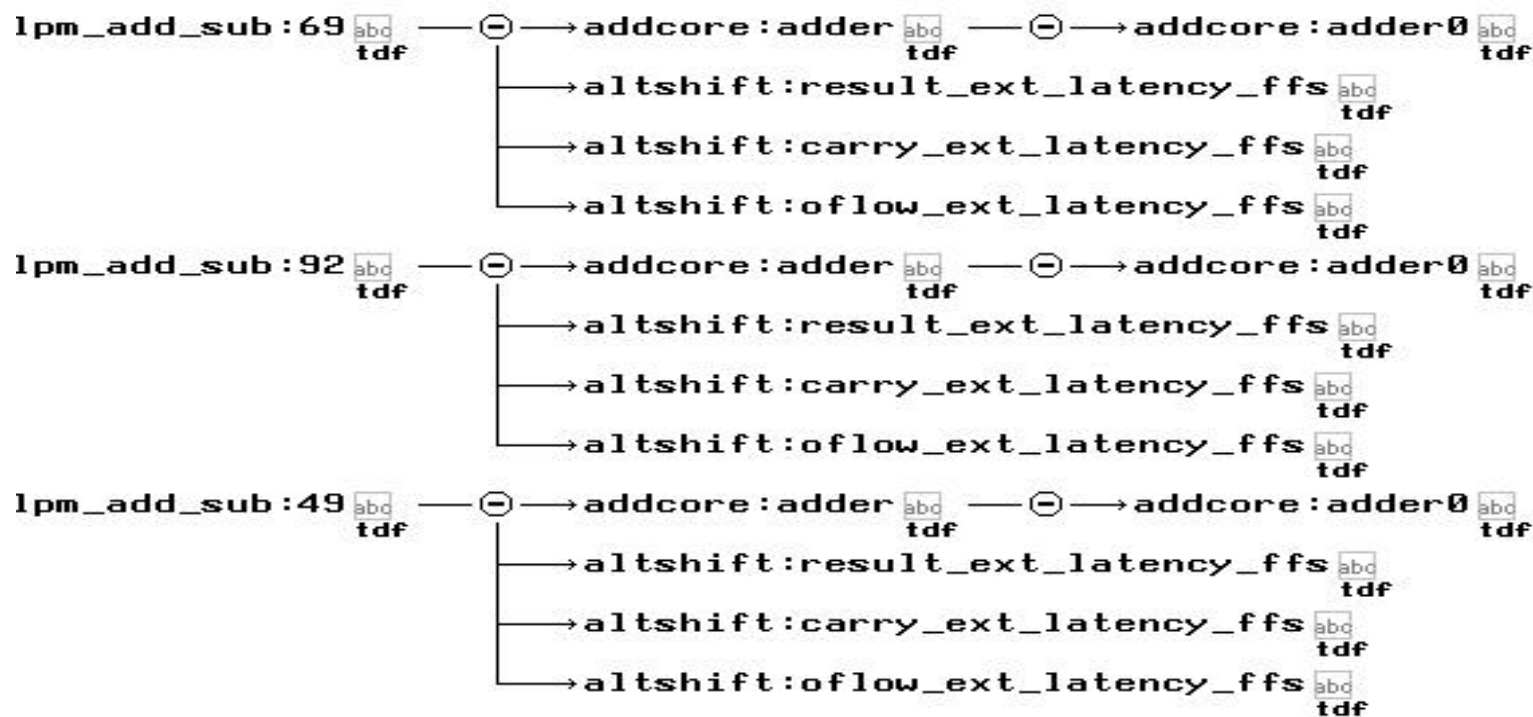


图 4-8 交通灯系统的层次显示

## 8. 系统的时间分析和仿真

编译无误之后就可进行系统的时间分析和仿真。

### (1) 电路的时间分析。

选取窗口菜单 Utilities→Analyze Timing,即可看到如图 4-9 所示的**延时矩阵**。



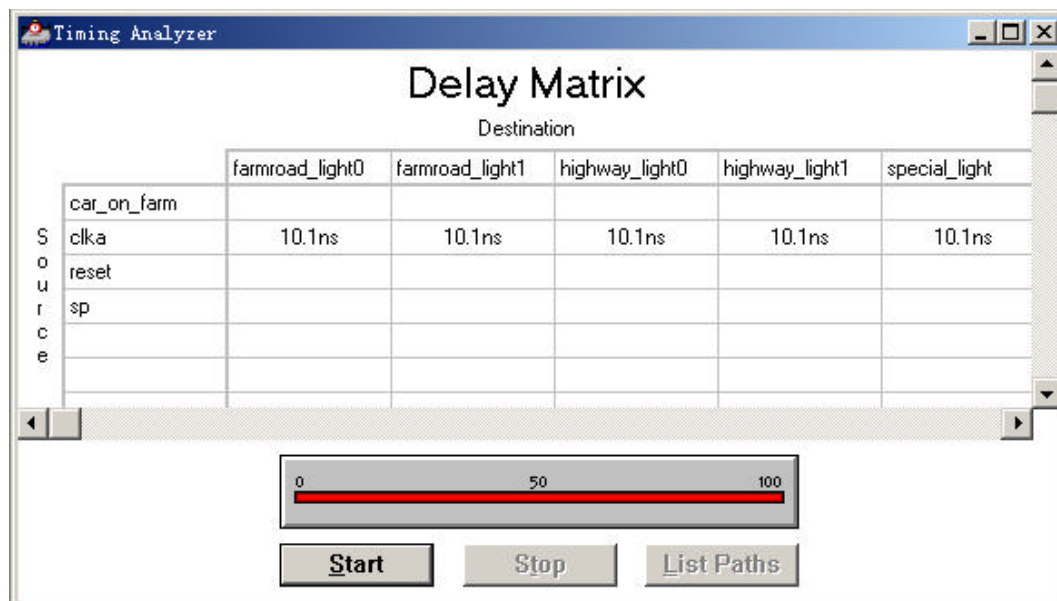


图 4-9 交通灯时间分析延时矩阵

从延时矩阵上看, 延时 10.1ns, 时间较长, 这是由于时钟先通过 8M 的分频器, 将会产生 4ns 的延时, 再通过后续模块, 又延时 4ns, 模块组合传输线路延时 2ns 左右。相加为 10ns。

在 Analysis 菜单中选择 Register Performance 项, 点击 start 即可进行**时序逻辑电路性能分析**, 如图 4-10 所示。

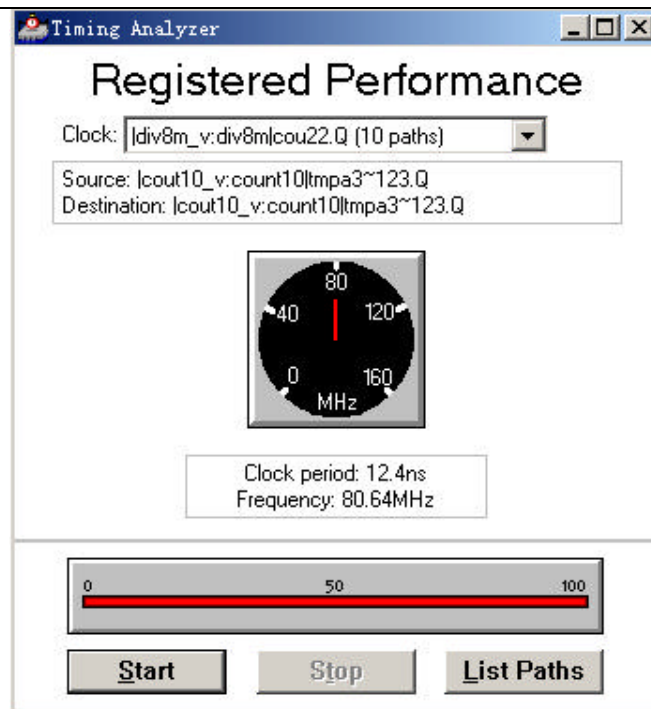


图 4-10 模块化后的交通灯系统性能分析

在图 4-10 中，Source 显示被分析的时钟信号的名称，Destination 显示制约性能的目标节点的名称，Clock period 显示给定时钟下时序逻辑电路要求的最小时钟信号，交通灯系统的最小周期是 12.4ns，Frequency 显示给定的时钟信号的最高频率，交通灯系统的最高频率是 80.64Mhz。List paths 可打开信息处理窗口并显示延迟路径。

实际上，我们再次观察图 4-5,可以发现，在顶层模块进行编译的过程中，已经有**适配**过程 Fitter，并产生了报告文件（.rpt）和烧写文件（.pof）。说明在顶层编译时已经由 MAX+PLUSII 自动配置出一个系统。但是用**时间分析器**可以

看出**性能上的差别**。图 4-11 为顶层模块的系统性能分析。

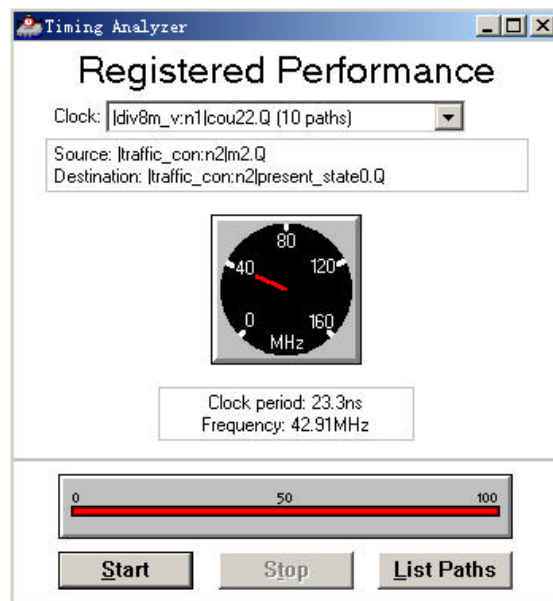


图 4-11 自动生成的交通灯系统性能分析

可以很清楚地看到，经过模块化设计后的交通灯系统，**在性能上有很大提升**。

(2)仿真过程。我们可以自己用 VHDL 编写仿真激励文件，也可用波形编辑器。后者更直观一些，下面加以重点介绍。

首先进入波形编辑窗口:选取窗口菜单 Max+plusII— Waveform editor.

引入输入输出脚：选取 Node— Enter Nodes From Snf,出现对话框，选 List 钮，将需要**观察的信号拖入右边**，并单击 OK 就可以了，如图 4-12 所示。

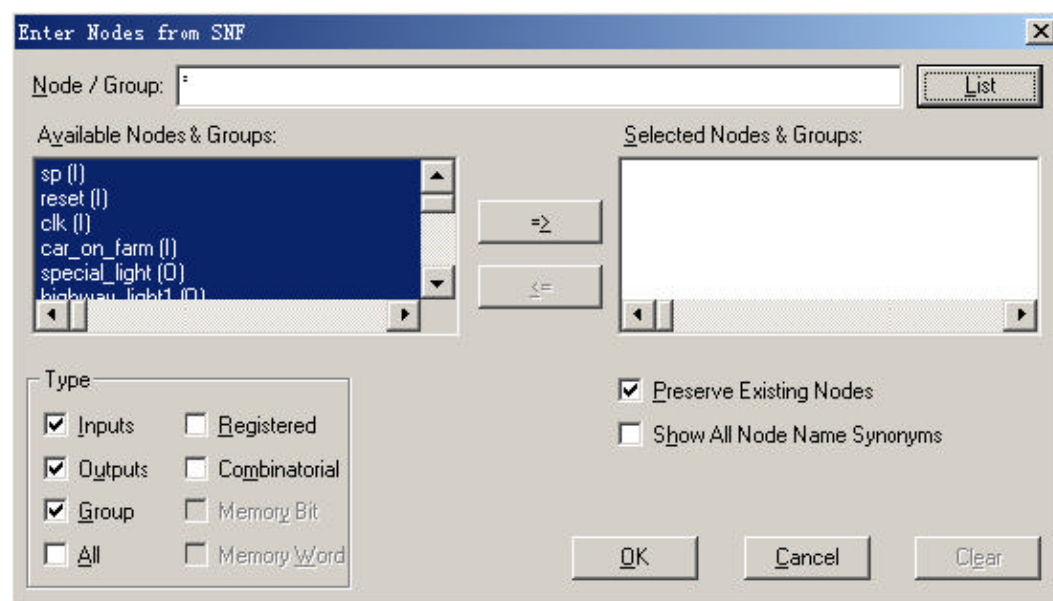


图 4-12 仿真管脚选择

管脚选定后就可仿真。我们可以在波形编辑器中设定输入的变化情况，设定好以后保存文件，选取 Max+plusII— Simulator，出现 Time Simulation 对话框，单击 start, 出现 Simulator 框，单击“确定”，即可完成仿真。仿真结果图 4-13 所示。

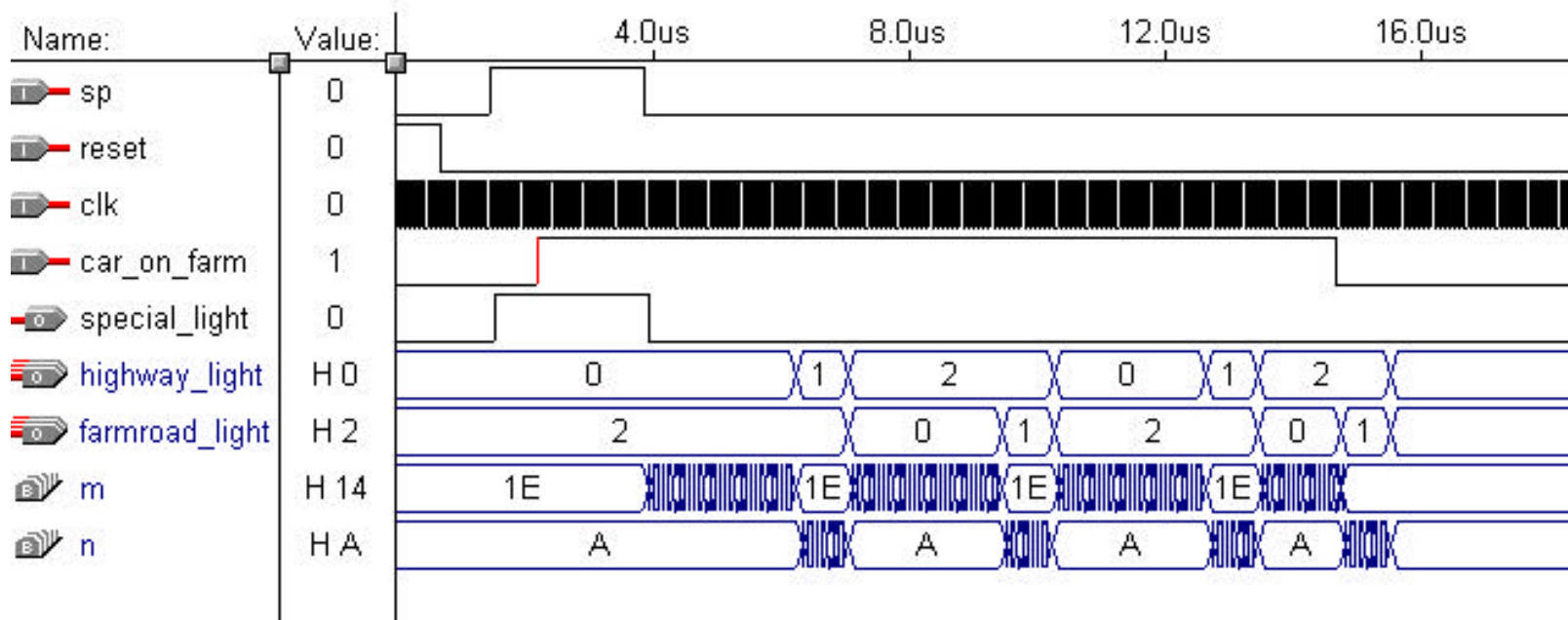


图 4-13 仿真结果示意图

在图 4-13 中, highway\_light 和 farmroad\_light 行上的数字 0 代表绿灯, 1 代表黄灯, 2 代表红灯。可以看到当 sp 有效时('1'电平), 不论小道上有没有车, 两路上的灯都不变。当 sp '0' 电平时, special\_light 变为 '0', m 开始计时, 过一段时间后灯会有变化。从图 4-13 中还可以看到 farmroad\_light 的第二个 '0' 值时间明显比第一个短, 这是因为第一个 '0' 值, 即第一个 farmroad 的绿灯期结束是因为超过一定时间, 而第二个绿灯期结束是因为 car\_on\_farm 变为 '0' 电平了。从上边分析可以看到, 系统的仿真是完全正确的。仿真正确以后, 就可以进行下一步的工作——系统烧写。

#### 9. 系统烧写和演示验证

系统烧写又叫系统下载, 是指将编写好的程序通过计算机并行口接到 Altera 专用编程电缆上, 再接到器件的编程接口, 利用应用软件提供的编程软件 Programmer 即可对器件进行配置。

具体过程如下：

(1)首先打开 MAX+PLUS II 菜单中的 Floorplan Editor，选择 layout 菜单中的 Device View，我们可以看到和实物器件相对应的管脚分配。选择 Current Assignments Floorplan 可以自定义管脚位置。将屏幕右上脚 Unassigned Nodes 框里的管脚拉到器件上再编译一次就可以了。如图 4-14 所示。

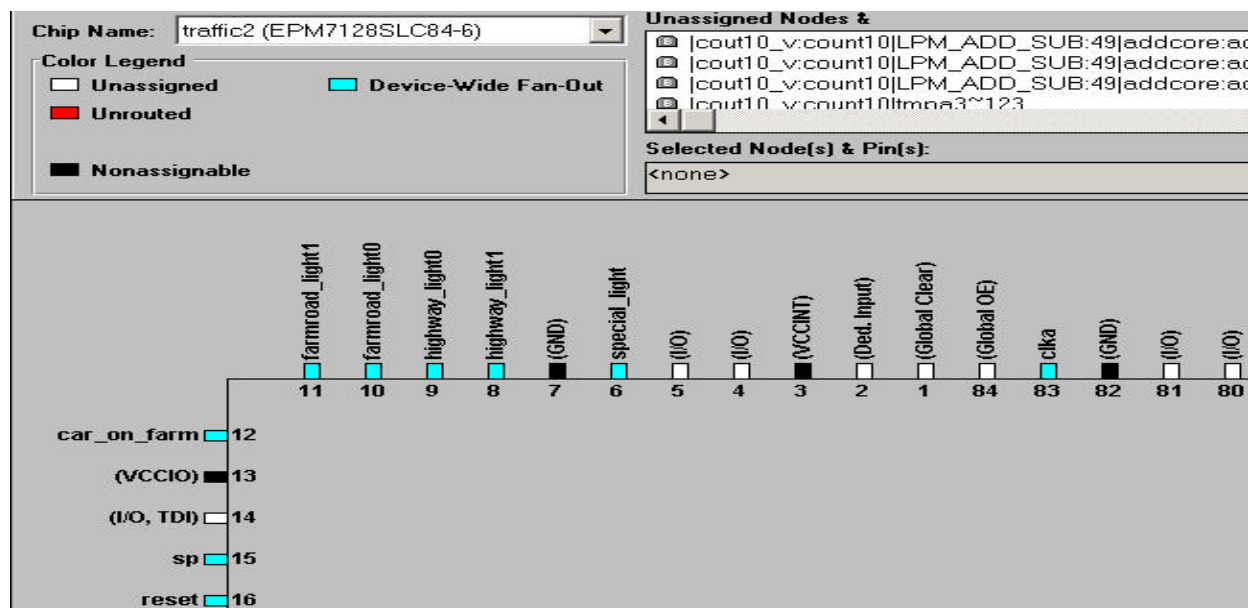


图 4-14 器件管脚配置示意图

如在 layout 中选择 Lab View 格式，我们还可以看到器件被利用的情况。

(2)器件管脚分配好并编译以后，就可以进行烧写了。首先，联接好并行电缆，给实验板上接 5v 电压。选择 Max+plusII 菜单中的 Programmer 选项。出现 Hardware Setup 框和 Programmer 框。分别如图 4-15、图 4-16 所示。

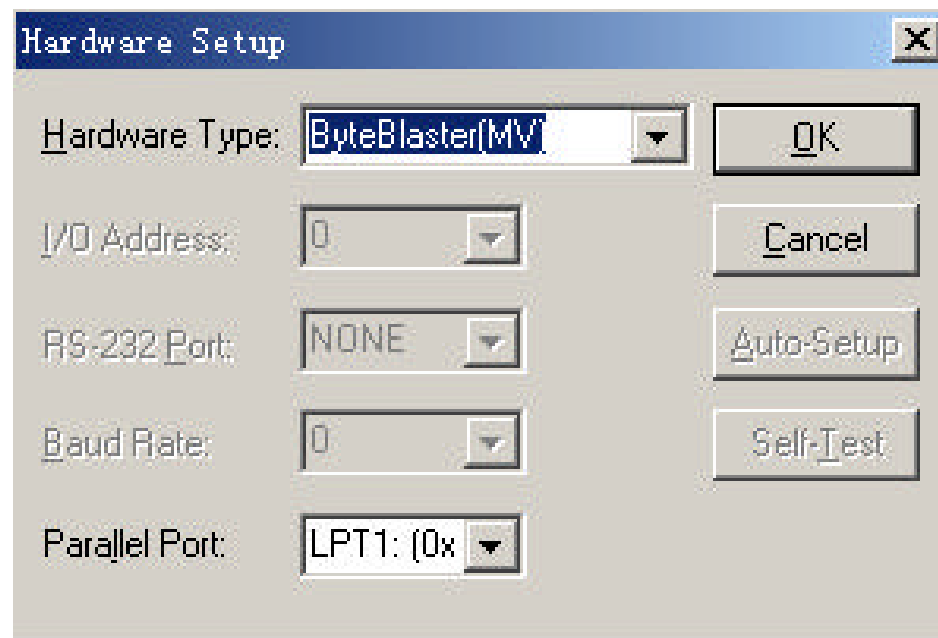


图 4-15 Hardware Setup 对话框

在 Hardware Type 中选择 ByteBlaster[mv]即可。

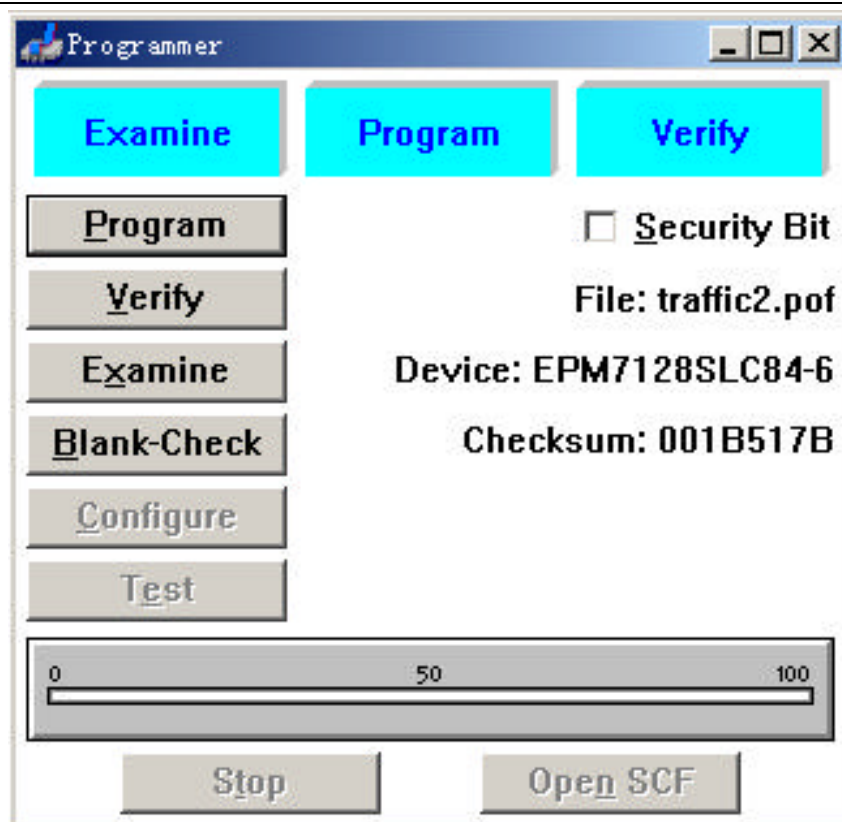


图 4-16 Programmer 对话框

选好以后，在 Programmer 对话框中，点击 Program 进行烧写。烧写成功后，即可施加外部信号，进行演示验证了。去掉电缆，保留电源。可以看到随着外部条件的变化，交通灯系统的输出信号也在进行相应变化。至此整个设计结束。



## 在 Win2000/XP 里安装 Altera 下载线的驱动程序

Windows 2000/XP 安全稳定，因此建议使用该操作系统。由于 Win2000 的保护机制使软件无法直接操作端口。因此，要在 Windows 2000 下使用下载线（ByteBlasterMV 或者 ByteBlaster 2），就必须在 WIN2000 下安装驱动程序。

下面介绍如何安装这个驱动程序。

- 1、打开控制面板（开始->设置->控制面板）
- 2、双击“添加/删除硬件”图标，启动添加/删除硬件向导，然后按“下一步”继续。
- 3、在“选择一个硬件任务”面板上，选择“添加/排除设备故障”，然后按“下一步”继续。WIN2000 将会在新的硬件检测窗口里搜索新的即插即用设备。
- 4、在“选择一个硬件设备”面板上，选择“添加新设备”。按“下一步”继续。
- 5、在“查找新硬件”面板上，选择“否，我想从列表选择硬件”。按“下一步”继续。
- 6、在“硬件类型”面板上，选择“声音、视频和游戏控制器”。按下一步继续。

- 7、在“选择一个设备驱动程序”窗口，点击“从磁盘安装”按钮。
- 8、指定 win2000.inf 文件的完整路径（例如<max+plus II 安装目录>\\drivers\\win2000），按下一步继续。
- 9、在“没有找到数字签名”窗口，点击“是”按钮。
- 10、在“选择一个设备驱动程序”窗口，如果您使用 **ByteblasterMV** 则选 **Altera** **ByteBlaster**，如果使用 Master Programming Unit（MPU）则选 Altera Programmer。
- 11、在“开始硬件安装”窗口，点击下一步继续。
- 12、在“没有找到数字签名”窗口，点击“是”按钮，继续安装。
- 13、在“完成添加/删除硬件向导”窗口，点击“完成”按钮
- 14、在弹出的系统对话选择是否重新启动计算机。驱动程序要在重新启动后才能使用。重新启动后，即可在 MaxplusII 里**进行下载** program 了。