

FPGA/SOPC开发快速入门教程¹

V2.03
2005-9-1

www.21control.com

前言

FPGA 在复杂逻辑电路以及数字信号处理领域中扮演者越来越重要的角色，SOC（片上系统）以其低功耗，高性能，低成本，高可靠性等优点成为嵌入式系统的发展趋势。作为一个简明的教程，主要宗旨是让初学者快速地了解 FPGA/SOPC（可编程片上系统）开发的流程。目前 IT 技术的发展可以说是一日千里，以本人的观点来讲，如果希望在电子设计领域有所作为，则必须具备快速掌握新技术的能力。电子设计最重要的是实践的积累，我们只要具备了一定的基础，应当马上投入实践，否则很多概念都无法真正理解。有不少人包括我，当下决心要成为一个合格的电子设计工程师的时候，总是想如果把有关电路方面的理论都掌握了才能所向披靡，有底气参加实际项目设计。当然如果能做到“把有关理论都掌握了”这样的境界，我想应该是很理想的，但经验发现这并不实际。据我所知，我所认识的不少电子设计牛人，他们的理论知识可能都比不上我们的本科生，但很多不错的产品都是从他们的手中开发出来的，有了实践的经验后，他们掌握新技术的速度相当惊人。有人跟我说：“新技术是拿来用的，不是拿来学的。”他们认为掌握新的设计技术应当尽快掌握它的设计流程。因此，我参考朋友给我的意见，写了这个简易的教程，以非常详细的实例来让初学者了解基于 QuartusII 和 NiosII IDE 的 FPGA/SOPC 开发的基本流程，目的是为了初学者尽快上手 FPGA/SOPC 的开发流程，尽快投入到实践中。为了易于说明问题，本教程中的一些概念并不是很严谨，如果读者对某些提法有异议，请参考相关资料和教材，并以相关资料和教材为准。通过该简明教程，初学者能快速了解 FPGA/SOPC 的基本开发流程，很多技巧和深入理解都靠长期的经验积累，因此初学者应该在了解了基本流程以后，思维不能局限于此，应在实践中提高水平，并参考更全面和权威的资料。

本教程配套 CT-SOPCx 系列 FPGA/SOPC 学习套件（对于该套件的相关内容请参考附录。）以实践为基础，适合具备基本的数字电路设计基础的初学者。第一章是 CPLD/FPGA 的基本知识，这部分内容摘自互联网（www.fpga.com.cn）并稍加删改，对于 CPLD/FPGA 知识为零的初学者应先了解这部分内容；对于已经有了一定基础的同学可以跳过这部分内容。第二章以两个例子来让初学者了解 FPGA 的基本开发流程，并熟悉 QuartusII 软件的使用。第三章以一个例子来让初学者了解基于 NiosII 软 CPU 核的 SOPC 设计流程，并熟识 SOPC Builder 和 NiosII IDE 的基本使用。附录是本文所涉及的例子的学习板相关的内容。

由于本人水平有限，错漏和不严谨之处在所难免，欢迎大家批评指正。

冯寿廷

2005 年 9 月 25 日，于华南理工大学

¹ 版权归 www.21control.com 所有，未经同意不得翻印、公开传播等，互联网转载请注明出处：www.21control.com。

第一章 CPLD/FPGA 的基本知识

(一) 可编程逻辑器件的历史和概述

随着数字电路应用越来越广泛,传统通用的数字集成电路已经难以满足系统的功能要求,而且随着系统复杂程度的提高,所需通用集成电路的数量呈爆炸性增值,使得电路的体积膨大,可靠性难以保证。此外,现代产品的生命周期都很短,一个电路可能需要在很短的周期内作改动以满足新的功能需求,对于采用通用的数字集成电路设计的电路系统来说即意味着重新设计和重新布线。因此,系统设计师们希望自己设计专用集成电路(ASIC)芯片,而且希望 ASIC 的设计周期尽可能短,最好是在实验室里就能设计出合适的 ASIC 芯片,并且立即投入实际应用之中,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(EEPROM)三种。由于结构的限制,它们只能完成简单的数字逻辑功能。

其后,出现了一类结构上稍复杂的可编程芯片,即可编程逻辑器件(PLD),它能够完成各种数字逻辑功能。典型的 PLD 由一个“与”门和一个“或”门阵列组成,而任意一个组合逻辑都可以用“与-或”表达式来描述,所以,PLD 能以乘积和的形式完成大量的组合逻辑功能。

这一阶段的产品主要有 PAL(可编程阵列逻辑)和 GAL(通用阵列逻辑)。PAL 由一个可编程的“与”平面和一个固定的“或”平面构成,或门的输出可以通过触发器有选择地被置为寄存状态。PAL 器件是现场可编程的,它的实现工艺有反熔丝技术、EPROM 技术和 EEPROM 技术。还有一类结构更为灵活的逻辑器件是可编程逻辑阵列(PLA),它也由一个“与”平面和一个“或”平面构成,但是这两个平面的连接关系是可编程的。PLA 器件既有现场可编程的,也有掩膜可编程的。在 PAL 的基础上,又发展了一种通用阵列逻辑 GAL (Generic Array Logic),如 GAL16V8,GAL22V10 等。它采用了 EEPROM 工艺,实现了电可擦除、电可改写,其输出结构是可编程的逻辑宏单元,因而它的设计具有很强的灵活性,至今仍有许多人使用。这些早期的 PLD 器件的一个共同特点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 和 Xilinx 分别推出了类似于 PAL 结构的扩展型 CPLD(Complex Programmable Logic Device)和与标准门阵列类似的 FPGA(Field Programmable Gate Array),它们都具有体系结构和逻辑单元灵活、集成度高以及适用范围宽等特点。这两种器件兼容了 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。与门阵列等其它 ASIC(Application Specific IC)相比,它们又具有设计开发周期短、设计制造

成本低、开发工具先进、标准产品无需测试、质量稳定以及可实时在线检验等优点，因此被广泛应用于产品的原型设计和小批量产品生产(一般在 10,000 件以下)之中。几乎所有应用门阵列、PLD 和中小规模通用数字集成电路的场合均可应用 FPGA 和 CPLD 器件。

(二) FPGA / CPLD 概述

FPGA(现场可编程门阵列)与 CPLD(复杂可编程逻辑器件)都是可编程逻辑器件，它们是在 PAL,GAL 等逻辑器件的基础之上发展起来的。同以往的 PAL,GAL 等相比较 ,FPGA / CPLD 的规模比较大 ,它可以替代几十甚至几千块通用 IC 芯片。这样的 FPGA / CPLD 实际上就是一个子系统部件。这种芯片受到世界范围内电子工程设计人员的广泛关注和普遍欢迎。经过了十几年的发展，许多公司都开发出了多种可编程逻辑器件。比较典型的就 Altera 公司和 Xilinx 公司的 CPLD 器件系列和 FPGA 器件系列，它们开发较早，占用了较大的 PLD 市场。通常来说，在欧洲用 Xilinx 的人多，在日本和亚太地区用 ALTERA 的人多，在美国则是平分秋色。全球 PLD/FPGA 产品 60%以上是由 Altera 和 Xilinx 提供的。可以讲 Altera 和 Xilinx 共同决定了 PLD 技术的发展方向。当然还有许多其它类型器件，如：Lattice，Vantis，Actel，Quicklogic，Lucent 等。（99 年 Lattice 收购了 Vantis，成为第三大 PLD 供应商。

表 1.2.1 1998 年世界十大 PLD 公司

排名	公司	销售额(亿美金)	市场占有率
1	Altera	5.96	30.1
2	Xilinx	5.74	29.0
3	Vantis	2.20	11.1
4	Lattice	2.18	11.0
5	Actel	1.39	7.0
6	Lucent	0.85	4.3
7	Cypress	0.44	2.2
8	Atmel	0.42	2.1
9	Philips	0.28	1.4
10	Quicklogic	0.24	1.2

资料来源：99 年 4 月《电子产品世界》

尽管 FPGA,CPLD 和其它类型 PLD 的结构各有其特点和长处，但概括起来，它们是由三大部分组成的：(1) 一个二维的逻辑块阵列，构成了 PLD 器件的逻辑组成核心；(2) 输入 / 输出块；(3) 连接逻辑块的互连资源，由各种长度的连

线线段组成，其中也有一些可编程的连接开关，它们用于逻辑块之间、逻辑块与输入/输出块之间的连接。

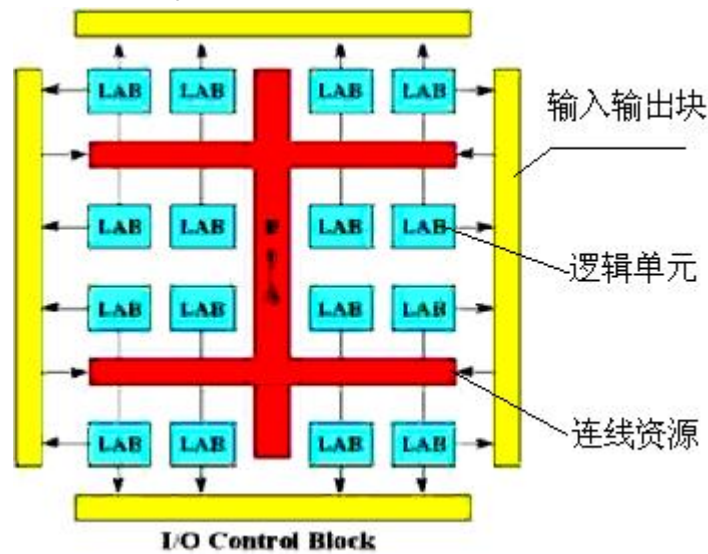


图 1.2.1 PLD 的结构

对用户而言，虽然 CPLD 与 FPGA 的内部结构稍有不同，但其用法都一样，所以多数情况下，不加以区分。FPGA / CPLD 芯片都是特殊的 ASIC 芯片，它们除了具有 ASIC 的特点之外，还具有以下几个优点：

- (1) 随着 VLSI (Very Large Scale IC, 超大规模集成电路) 工艺的不断提高单一芯片内部可以容纳上百万个晶体管，FPGA / CPLD 芯片的规模也越来越大，其单片逻辑门数已达到上百万门，它所能实现的功能也越来越强，同时也可以实现系统集成，即片上系统 SOC。
- (2) FPGA / CPLD 芯片在出厂之前都做过百分之百的测试，不需要设计人员承担投片风险和费用，设计人员只需在自己的实验室里就可以通过相关的软硬件环境来完成芯片的最终功能设计。所以，FPGA / CPLD 的资金投入小，节省了许多潜在的花费。
- (3) 用户可以反复地编程、擦除、使用或者在外围电路不动的情况下用不同软件就可实现不同的功能。所以，用 FPGA / PLD 试制样片，能以最快的速度占领市场。FPGA / CPLD 软件包中有各种输入工具和仿真工具，及版图设计工具和编程器等全线产品，电路设计人员在很短的时间内就可完成电路的输入、编译、优化、仿真，直至最后芯片的制作。当电路有少量改动时，更能显示出 FPGA / CPLD 的优势。电路设计人员使用 FPGA / CPLD 进行电路设计时，不需要具备专门的 IC (集成电路) 深层次的知识，FPGA / CPLD 软件易学易用，可以使设计人员更能集中精力进行电路设计，快速将产品推向市场。
- (4) 在线可编程技术 (ISP) 使得使用 CPLD/FPGA 的产品可以做到远程升级。

(以上内容参照西电《CPLD 技术及其应用》，有改动)

(三) PLD/FPGA 结构与原理初步

一. 基于乘积项 (Product-Term) 的 PLD 结构

采用这种结构的 PLD 芯片有：Altera 的 MAX7000，MAX3000 系列（EEPROM 工艺），Xilinx 的 XC9500 系列（Flash 工艺）和 Lattice, Cypress 的大部分产品（EEPROM 工艺）。我们先看一下这种 PLD 的总体结构（以 MAX7000 为例，其他型号的结构与此都非常相似）：

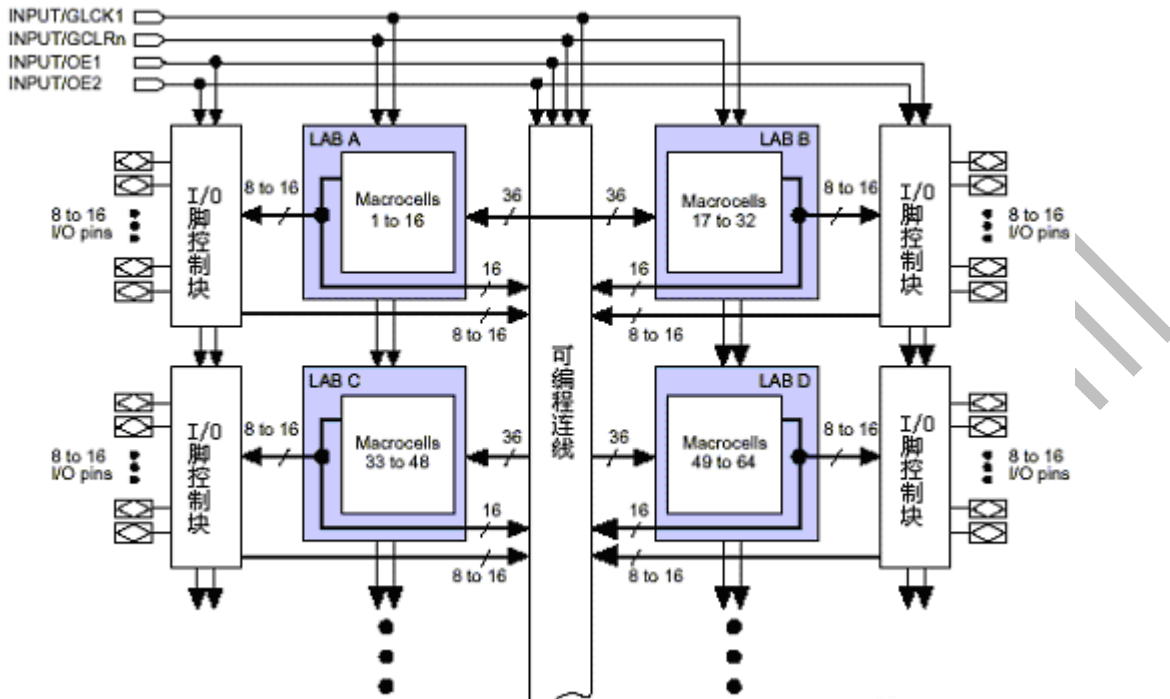


图 1.3.1 基于乘积项的 PLD 内部结构

这种 PLD 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，由它来实现基本的逻辑功能。图 1.3.1 中阴影部分是多个宏单元的集合（因为宏单元较多，没有一一画出）。可编程连线负责信号传递，连接所有的宏单元。I/O 控制块负责输入输出的电气特性控制，比如可以设定集电极开路输出，摆率控制，三态输出等。图 1.3.1 左上的 INPUT/GLCK1，INPUT/GCLRn，INPUT/OE1，INPUT/OE2 是全局时钟，清零和输出使能信号，这几个信号有专用连线与 PLD 中每个宏单元相连，信号到每个宏单元的延时相同并且延时最短。宏单元的具体结构见下图：

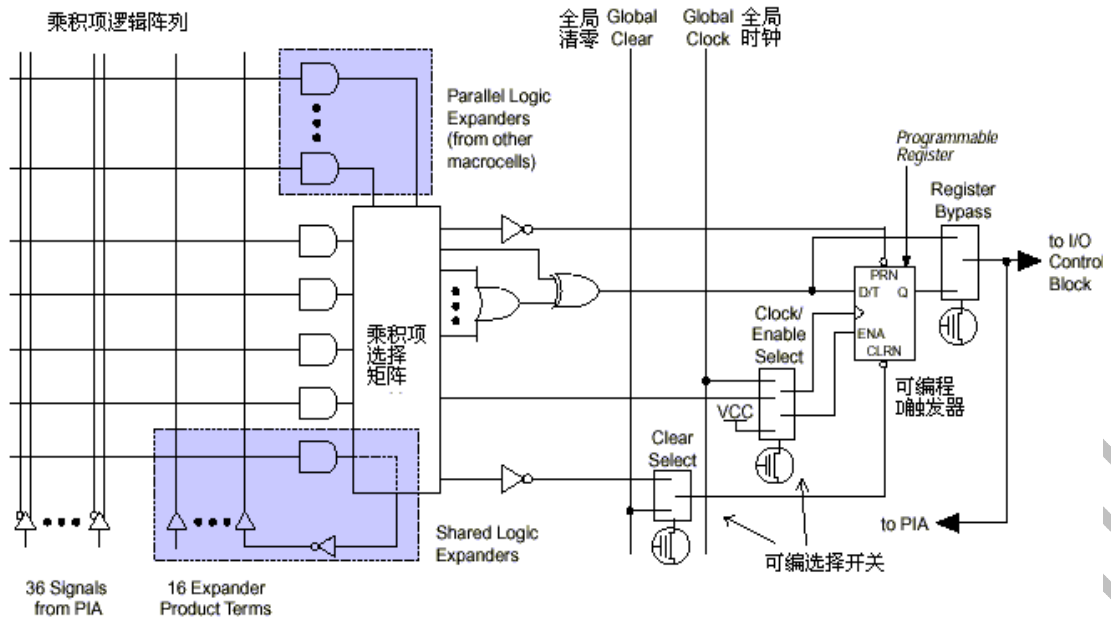


图 1.3.2 宏单元结构

左侧是乘积项阵列，实际就是一个与或阵列，每一个交叉点都是一个可编程熔丝，如果导通就是实现“与”逻辑。后面的乘积项选择矩阵是一个“或”阵列。两者一起完成组合逻辑。图右侧是一个可编程 D 触发器，它的时钟，清零输入都可以编程选择，可以使用专用的全局清零和全局时钟，也可以使用内部逻辑（乘积项阵列）产生的时钟和清零。如果不需要触发器，也可以将此触发器旁路，信号直接输给 PIA 或输出到 I/O 脚。

二. 乘积项结构 PLD 的逻辑实现原理

下面我们以一个简单的电路为例，具体说明 PLD 是如何利用以上结构实现逻辑的，电路如下图：

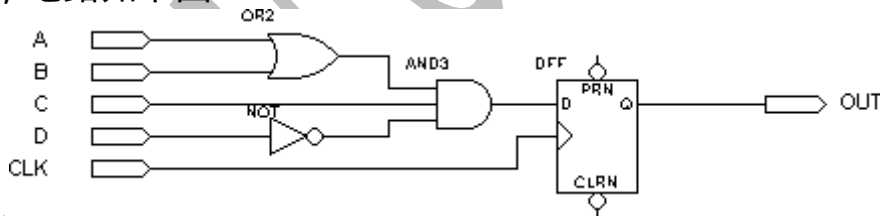


图 1.3.3

假设组合逻辑的输出(AND3 的输出)为 f 则 $f=(A+B)*C*(!D)=A*C*!D + B*C*!D$ (我们以 $!D$ 表示 D 的“非”)

PLD 将以下面的方式来实现组合逻辑 f ：

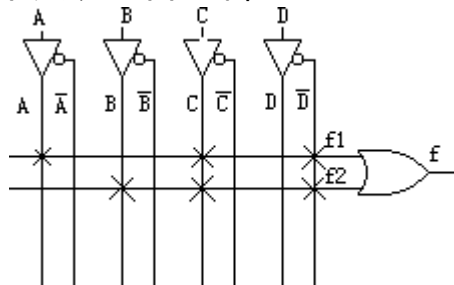


图 1.3.4

A, B, C, D 由 PLD 芯片的管脚输入后进入可编程连线阵列 (PIA)，在内部会产生 A, A 反, B, B 反, C, C 反, D, D 反 8 个输出。图中每一个叉表示相连

(可编程熔丝导通), 所以得到: $f = f_1 + f_2 = (A * C * !D) + (B * C * !D)$ 。这样组合逻辑就实现了。图 3 电路中 D 触发器的实现比较简单, 直接利用宏单元中的可编程 D 触发器来实现。时钟信号 CLK 由 I/O 脚输入后进入芯片内部的全局时钟专用通道, 直接连接到可编程触发器的时钟端。可编程触发器的输出与 I/O 脚相连, 把结果输出到芯片管脚。这样 PLD 就完成了图 1.3.3 所示电路的功能。(以上这些步骤都是由软件自动完成的, 不需要人为干预)。

图 1.3.3 的电路是一个很简单的例子, 只需要一个宏单元就可以完成。但对于一个复杂的电路, 一个宏单元是不能实现的, 这时就需要通过并联扩展项和共享扩展项将多个宏单元相连, 宏单元的输出也可以连接到可编程连线阵列, 再做为另一个宏单元的输入。这样 PLD 就可以实现更复杂逻辑。

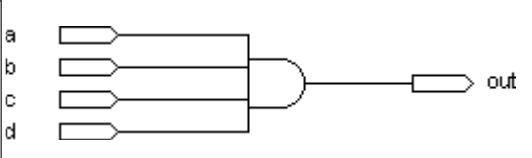
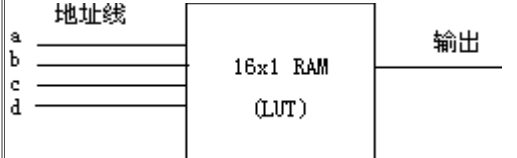
这种基于乘积项的 PLD 基本都是由 EEPROM 和 Flash 工艺制造的, 一上电就可以工作, 无需其他芯片配合。

三. 表 (Look-Up-Table) 的原理与结构

采用这种结构的 PLD 芯片我们也可以称之为 FPGA: 如 altera 的 ACEX, APEX 系列, xilinx 的 Spartan, Virtex 系列等。

查找表 (Look-Up-Table) 简称为 LUT, LUT 本质上就是一个 RAM。目前 FPGA 中多使用 4 输入的 LUT, 所以每一个 LUT 可以看成是一个有 4 位地址线的 16×1 的 RAM。当用户通过原理图或 HDL 语言描述了一个逻辑电路以后, PLD/FPGA 开发软件会自动计算逻辑电路的所有可能的结果, 并把结果事先写入 RAM, 这样, 每输入一个信号进行逻辑运算就等于输入一个地址进行查表, 找出地址对应的内容, 然后输出即可。

下面是一个 4 输入与门的例子:

实际逻辑电路		LUT 的实现方式	
			
a, b, c, d 输入	逻辑输出	地址	RAM 中存储的内容
0000	0	0000	0
0001	0	0001	0
....	0	...	0
1111	1	1111	1

四. 基于查找表 (LUT) 的 FPGA 的结构

altera 的 FLEX/ACEX 等芯片的结构如下图:

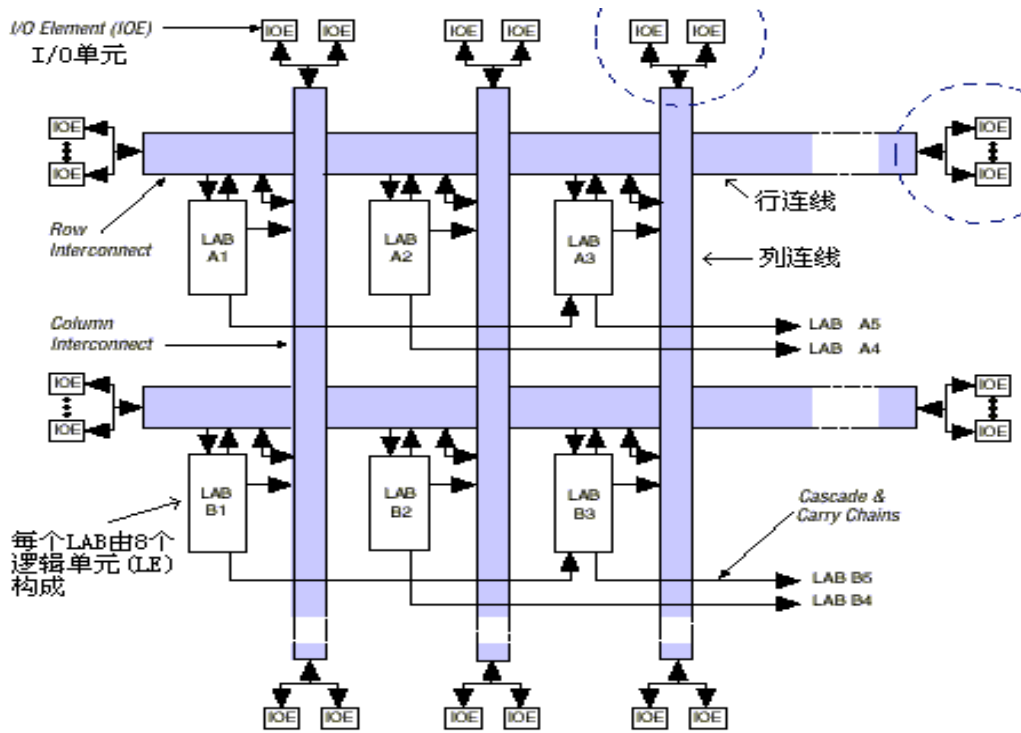
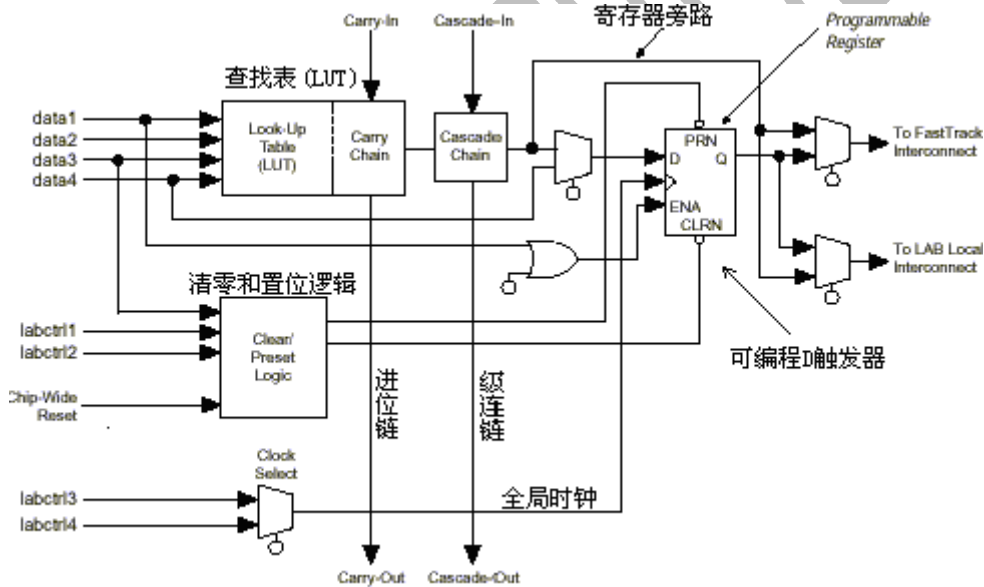


图 1.3.5 altera FLEX/ACEX 芯片的内部结构

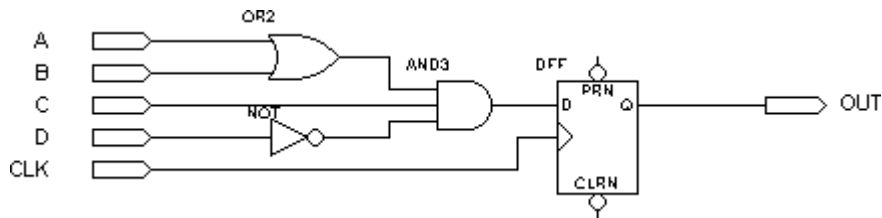


逻辑单元 (LE) 内部结构

FLEX/ACEX 的结构主要包括 LAB, I/O 块, RAM 块 (未表示出) 和可编程行/列连线。在 FLEX/ACEX 中, 一个 LAB 包括 8 个逻辑单元 (LE), 每个 LE 包括一个 LUT, 一个触发器和相关的相关逻辑。LE 是 FLEX/ACEX 芯片实现逻辑的最基本结构(altera 其他系列, 如 APEX、CYCLONE 等的结构与此基本相同, 具体请参阅数据手册)。

五. 查找表结构的FPGA逻辑实现原理

我们还是以这个电路的为例：



A, B, C, D 由 FPGA 芯片的管脚输入后进入可编程连线, 然后作为地址线连到 LUT, LUT 中已经事先写入了所有可能的逻辑结果, 通过地址查找到的数据然后输出, 这样组合逻辑就实现了。该电路中 D 触发器是直接利用 LUT 后面 D 触发器来实现。时钟信号 CLK 由 I/O 脚输入后进入芯片内部的时钟专用通道, 直接连接到触发器的时钟端。触发器的输出与 I/O 脚相连, 把结果输出到芯片管脚。这样 PLD 就完成了图 1.3.3 所示电路的功能。(以上这些步骤都是由软件自动完成的, 不需要人为干预)。

这个电路是一个很简单的例子, 只需要一个 LUT 加上一个触发器就可以完成。对于一个 LUT 无法完成的电路, 就需要通过进位逻辑将多个单元相连, 这样 FPGA 就可以实现复杂的逻辑。

由于 LUT 主要适合 SRAM 工艺生产, 所以目前大部分 FPGA 都是基于 SRAM 工艺的, 而 SRAM 工艺的芯片在掉电后信息就会丢失, 一定需要外加一片专用配置芯片, 在上电的时候, 由这个专用配置芯片把数据加载到 FPGA 中, 然后 FPGA 就可以正常工作, 由于配置时间很短, 不会影响系统正常工作。也有少数 FPGA 采用反熔丝或 Flash 工艺, 对这种 FPGA, 就不需要外加专用的配置芯片。

六. 选择 PLD 还是 FPGA ?

根据上述 PLD 的结构和原理可以知道, PLD 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 的一个 LUT 只能处理 4 输入的组合逻辑, 因此, PLD 适合用于设计译码等复杂组合逻辑。但 FPGA 的制造工艺确定了 FPGA 芯片中包含的 LUT 和触发器的数量非常多, 往往都是几千上万, PLD 一般只能做到 512 个逻辑单元, 而且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 PLD。所以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。

第二章 FPGA 基本教程

第一节 FPGA 的基本开发流程

PLD 是可编程逻辑器件 (Programmable Logic Device) 的简称, FPGA 是现场可编程门阵列 (Field Programmable Gate Array) 的简称, 两者的功能基本相同, 只是实现原理略有不同, 所以我们有时可以忽略这两者的区别, 统称为可编程逻辑器件或 PLD/FPGA。

PLD 是电子设计领域中最具活力和发展前途的一项技术, 它的影响丝毫不亚于 70 年代单片机的发明和使用。

PLD 能做什么呢? 可以毫不夸张的讲, PLD 能完成任何数字器件的功能, 上

至高性能 CPU, 下至简单的 74 电路, 都可以用 PLD 来实现。PLD 如同一张白纸或是一堆积木, 工程师可以通过传统的原理图输入法, 或是硬件描述语言自由的设计一个数字系统。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 的在线修改能力, 随时修改设计而不必改动硬件电路。使用 PLD 来开发数字电路, 可以大大缩短设计时间, 减少 PCB 面积, 提高系统的可靠性。PLD 的这些优点使得 PLD 技术在 90 年代以后得到飞速的发展, 同时也大大推动了 EDA 软件和硬件描述语言 (HDL) 的进步。

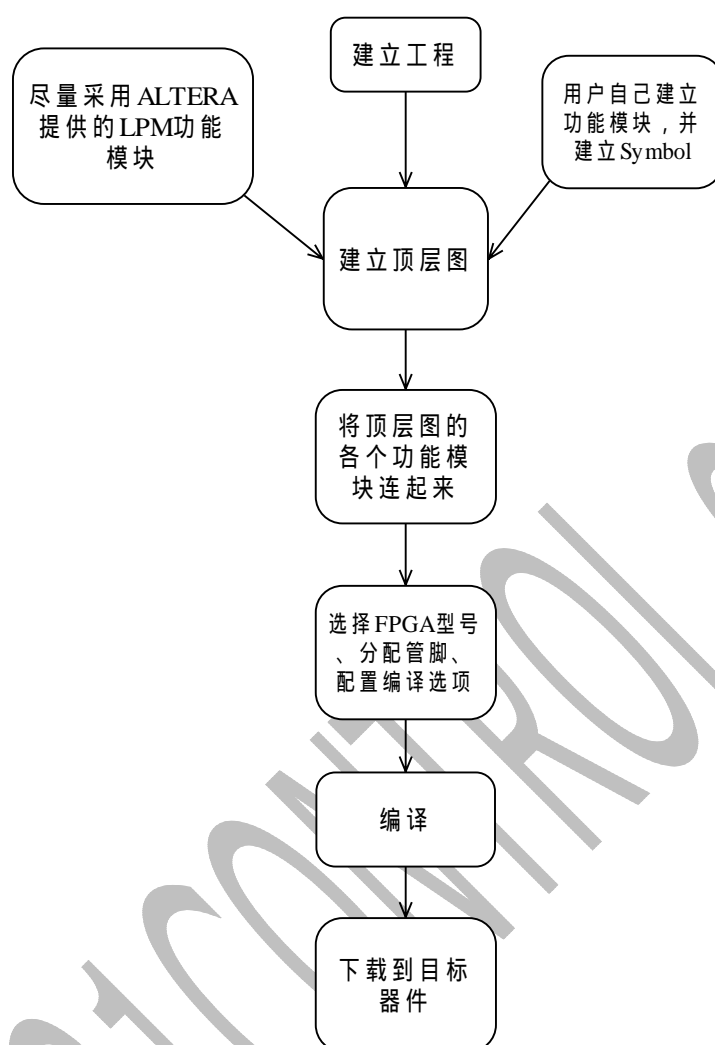
如何使用 PLD 呢? 其实 PLD 的使用很简单, 学习 PLD 比学习单片机要简单的多, 有数字电路基础, 会使用计算机, 就可以进行 PLD 的开发。

开发 PLD 需要了解两个部分: 1. PLD 开发软件 2. PLD 本身

由于 PLD 软件已经发展的相当完善, 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 HDL 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。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。

下面我们以基于 Altera 公司的 QuantusII 软件来简单说明一下 FPGA 的开发流程。

下图是一个典型的基于 QuantusII 的 FPGA 开发流程。



- (1) 建立工程是每个开发过程的开始，QuartusII（以下简称 Q2）以工程为单元对设计过程进行管理。
- (2) 建立顶层图。可以这样理解，顶层图是一个容器，将整个工程的各个模块包容在里面，编译的时候就将这些模块整合在一起。也可以理解为它是一个大元件，包含各个模块，编译的时候就是生成一个这样的大元件。
- (3) 采用 ALTERA 公司提供的 LPM 功能模块。Q2 软件环境里包含了大量的常用功能模块，例如计数器、累加器、比较器、译码器等等；如果不懂得在工程中采用这些现有的功能模块真是太浪费了。以本人的经验，一个设计中一般只有极少部分的模块需要自己从零设计。
- (4) 自己建立功能模块。当然，有些设计中现有的模块功能不能满足具体设计的要求，那就只能自己设计啦。可以用硬件描述语言也可以用原理图的输入方法。可以把它们独立地当作一个工程来设计，并生成模块符号（Symbol），然后在顶层图中使用这个模块的符号，并将源文件（实现该模块的原理图或 HDL 文件）拷到顶层图所在的工程目录下。这个过程好比你要做一个电路，现在市面上没有你想要的某个芯片，你就只能自己做一块这样的一块芯片，然后添加到你的电路板上。

- (5) 将顶层图的各个功能模块用连线连起来。这个过程类似电路图设计，把各个芯片连接起来，组成电路系统。
- (6) 系统的功能原理图至此已经基本出炉了，下一步要为该设计选择芯片载体，才能真正在物理上实现系统的功能。这一步的主要工作是：(1) 选择芯片型号；(2) 为顶层图的各个输入输出信号分配芯片的管脚；(3) 设置编译选项，目的是让编译器知道更多的信息。
- (7) 编译。这个过程类似软件开发里的编译，但实际上这个过程比软件的编译要复杂得多，因为它毕竟最终要实现硬件里的物理结构，包含了优化逻辑的组合、综合逻辑以及布线等步骤。在类似 Q2 这样的集成环境里面，这些过程都可以一气呵成，集成环境帮你自动完成了几个步骤的工作。当然，你也可以用其它工具来实现各个步骤的工作，这些内容超出了本教程陈述的范围。
- (8) 编译后会生成*.sof 或*.pof 文件，前者可以通过 JTAG 下载到 FPGA 内部，设计无误的话即能实现预期的功能，但断电后 FPGA 里的这些信息会丢失；后者可以下载到 FPGA 的配置芯片（EEPROM 或 FLASH 芯片），掉电后这些配置信息不会丢失，重新上电以后通过该配置芯片对 FPGA 的内部 RAM 进行配置。
- (9) 对于复杂的设计，工程编译了以后可以采用 Q2 的仿真功能或其它仿真软件（如 ModelSim）对设计反复进行仿真和验证，直到满足要求。

第二节 基于 QuartusII 的实例

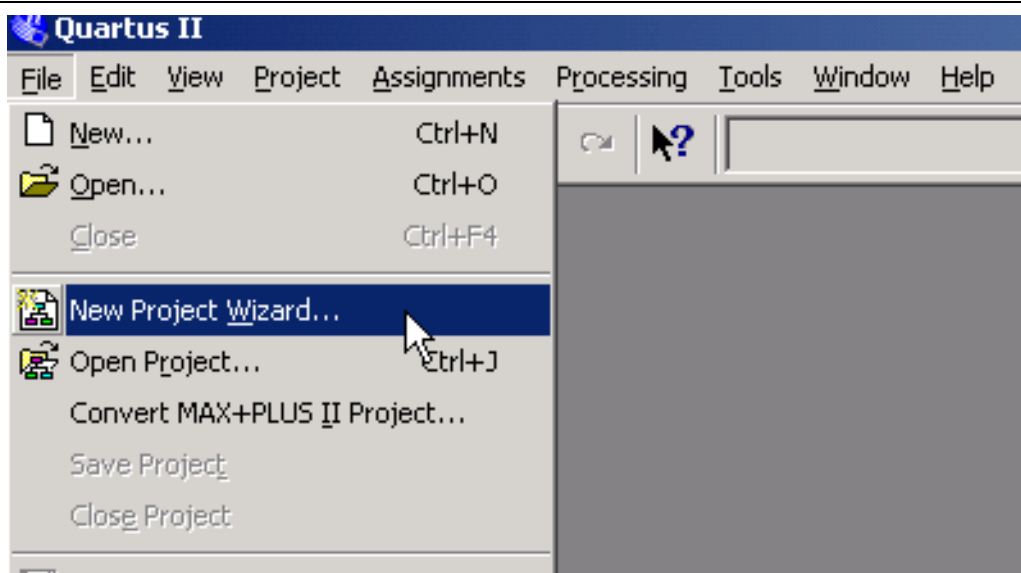
（一）实验一：实验板上的 KEY1 按钮控制 FPGA 核心板上的第一个 LED 灯。

目的：通过该实例学习，可以了解 FPGA 的基本开发流程，熟识 quartusII 软件基本功能的使用。

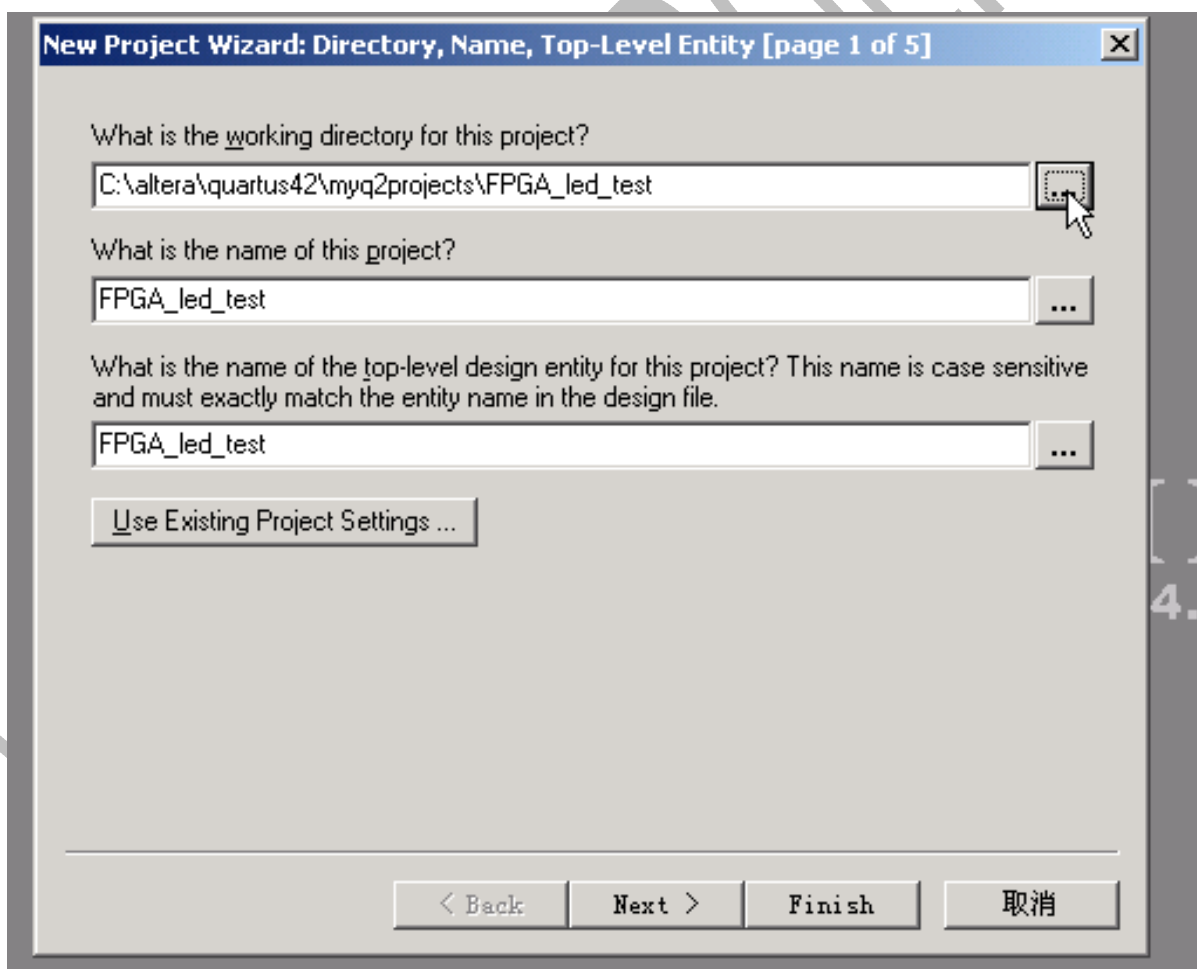
原理：利用一个常开按钮（实验板上的 KEY1）作为输入（常开时输入 1，闭合时输入 0），经过一个反相器后输出到核心板的第一个 LED。KEY1 常开时，LED 亮，按下（闭合）实验板上的 KEY1，该 LED 熄灭。

1. 建立工程

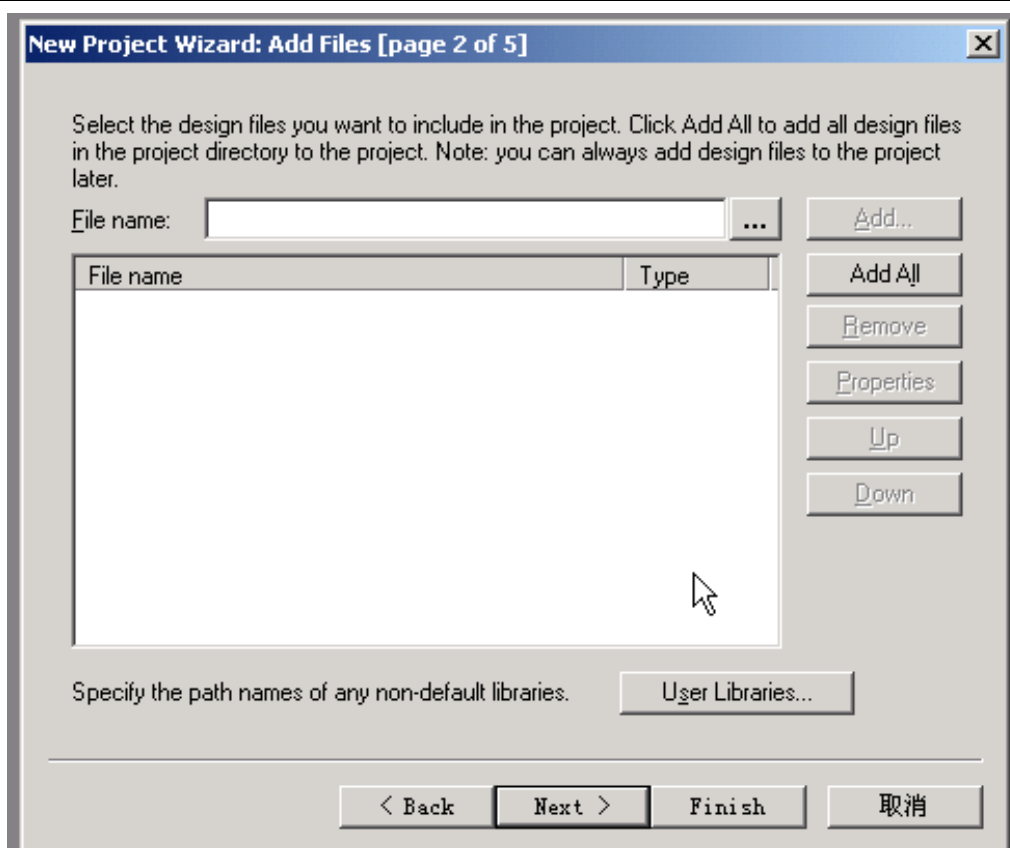
运行 QuatrusII 软件（以下简称 Q2），建立工程，File→New Project Wizard 如下图



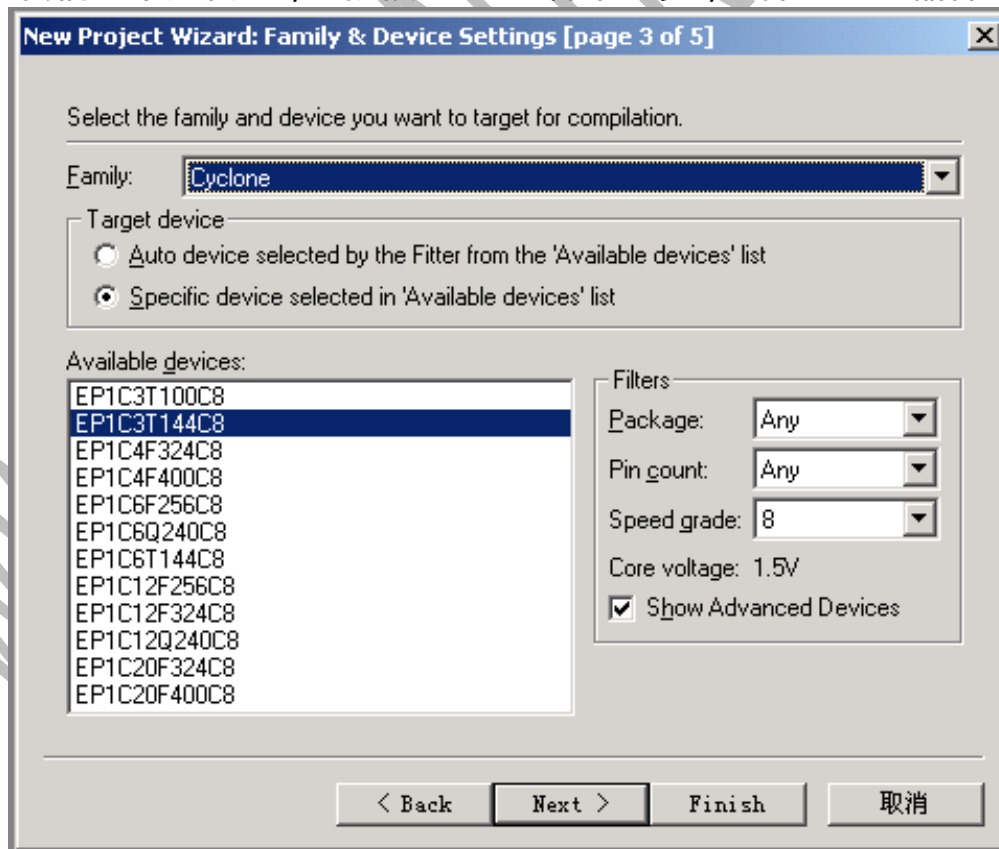
点击 New Project Wizard 后弹出指定工程名的对话框,在 Directory, Name, Top-Level Entity 中如下图填写:



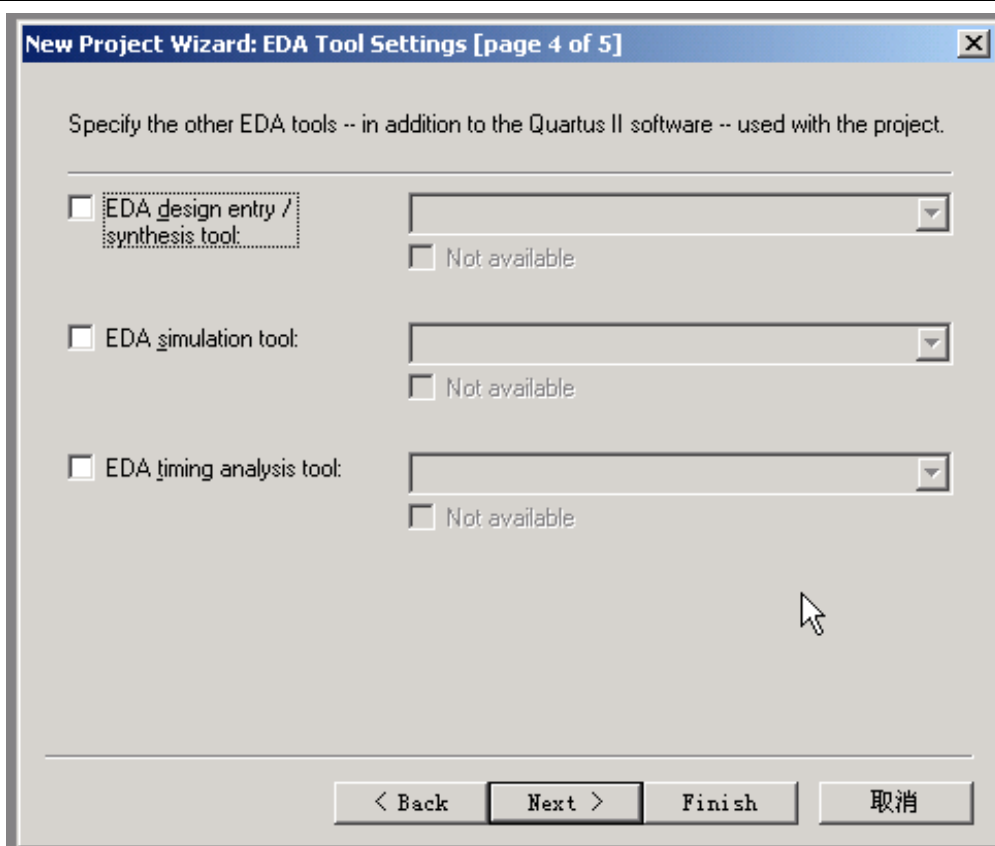
按 Next 按钮, 出现添加工程文件的对话框:



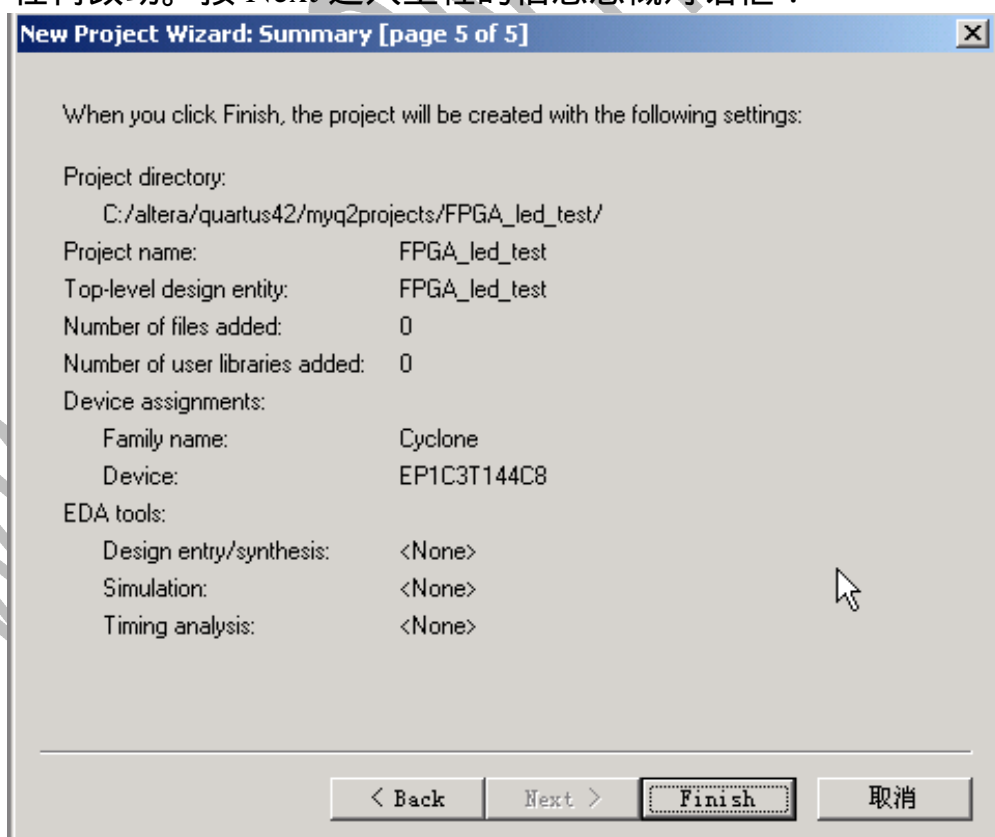
在这里我们先不用管它，直接按 Next 进行下一步，选择 FPGA 器件的型号：



在 Family 下拉筐中，我们选择 Cyclone 系列 FPGA，然后在“Available devices:”中根据核心板的 FPGA 型号选择 FPGA 型号，注意在 Filters 一栏选上“Show Advanced Devices”以显示所有的器件型号。执行下一步出现对话框：



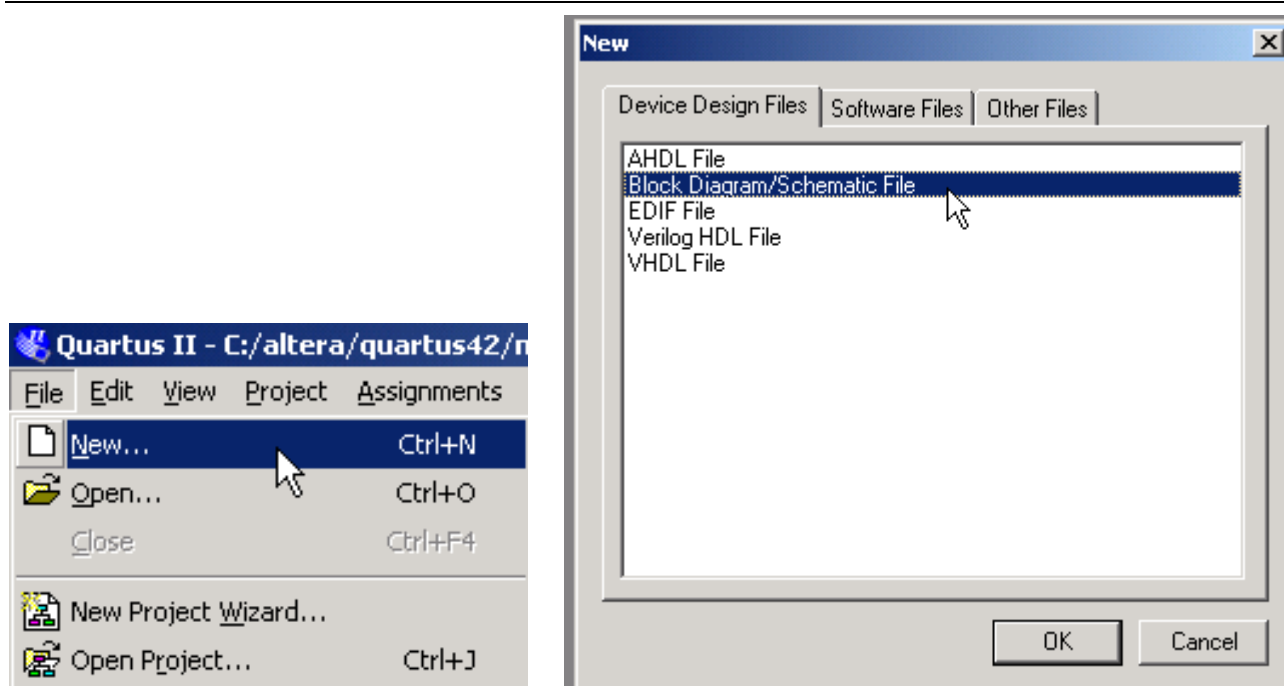
这里是选择其它 EDA 工具的对话框，我们用 Q2 的集成环境进行开发，因此这里不作任何改动。按 Next 进入工程的信息总概对话框：



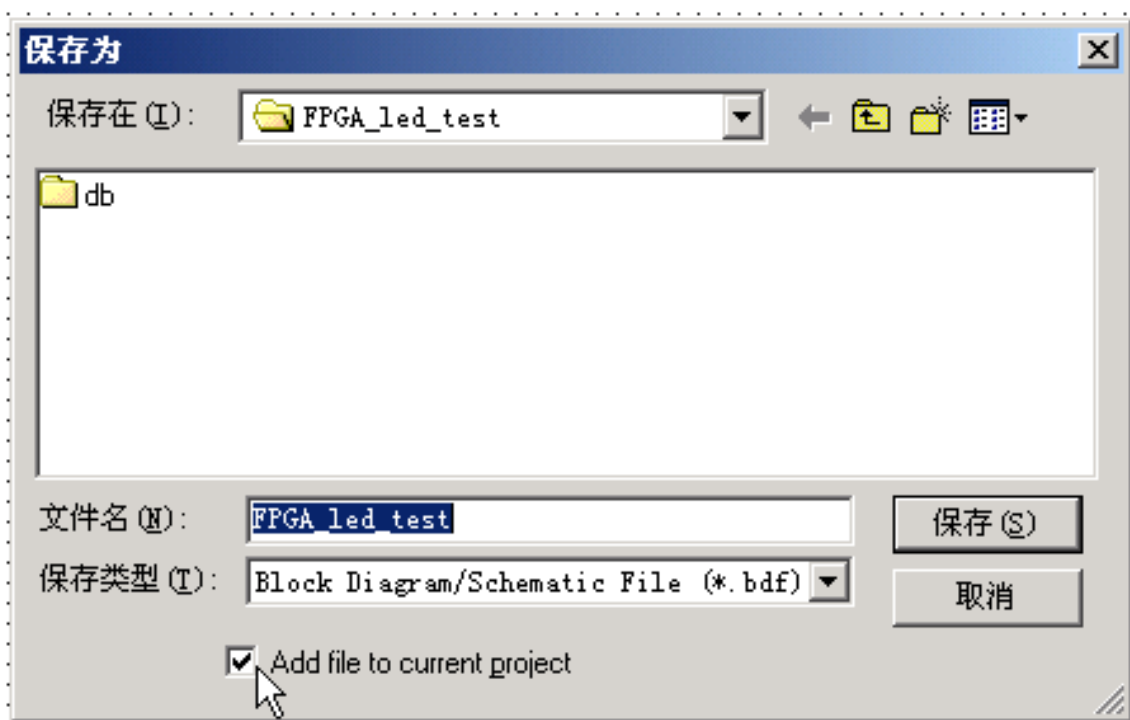
按 Finish 按钮即建立一个空项目。

2. 建立顶层图

执行 File→New，弹出新建文件对话框：

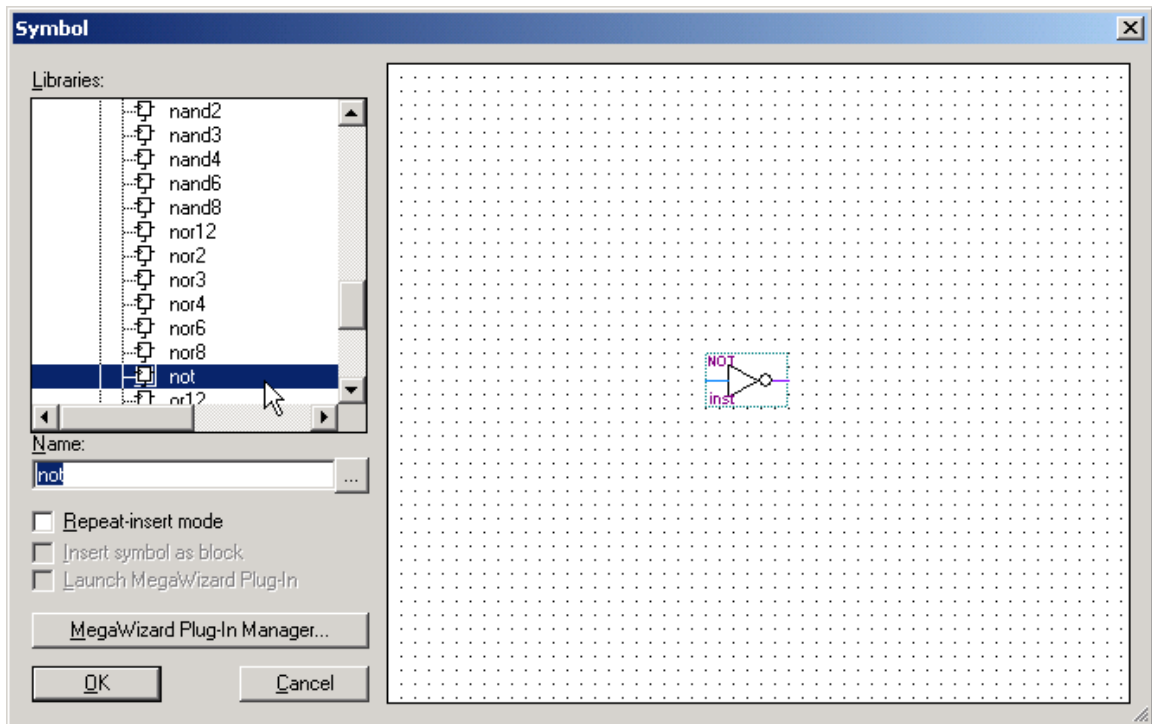


选择“Block Diagram Schematic File”按 OK 即建立一个空的顶层图，缺省名为“Block1.bdf”，我们把它另存为（File→Save as），接受默认的文件名，并将“Add file to current project”选项选上，以使该文件添加到工程中去。如图所示：



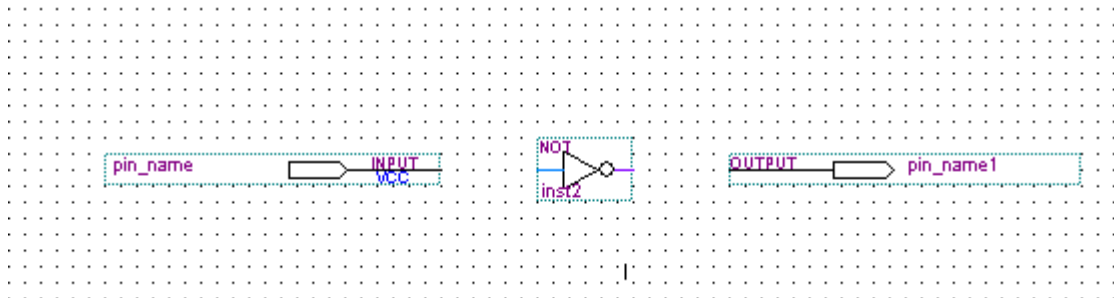
3. 添加逻辑元件（Symbol）

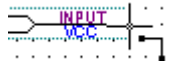
双击顶层图图纸的空白处，弹出添加元件的对话框：

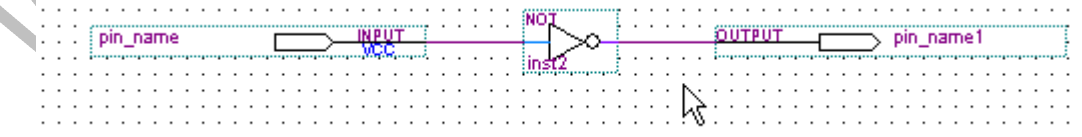


在 Libraries 里寻找所需要的逻辑元件，如果知道逻辑元件的名称的话，也可以直接在 Name 一栏敲入名字，右边的预览图即可显示元件的外观，按 OK 后鼠标旁边即拖着一个元件符号，在图纸上点击左键，元件即安放在图纸上。

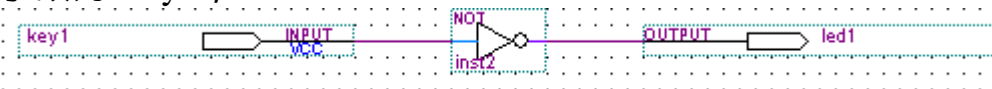
在图纸上分别添加非门（not）、输入（input）、输出（output）三个 symbol，如图所示：



连线，将鼠标移到 symbol 连线端口的那里，鼠标变成图示模样：，按下左键拖动鼠标到另一个 symbol 的连线端。本例中，这三个 symbol 的连线如下图所示：



分别双击 input 和 output symbol 的名字“pin_name”、“pin_name1”，将它们的名字改为 Key1，LED1：

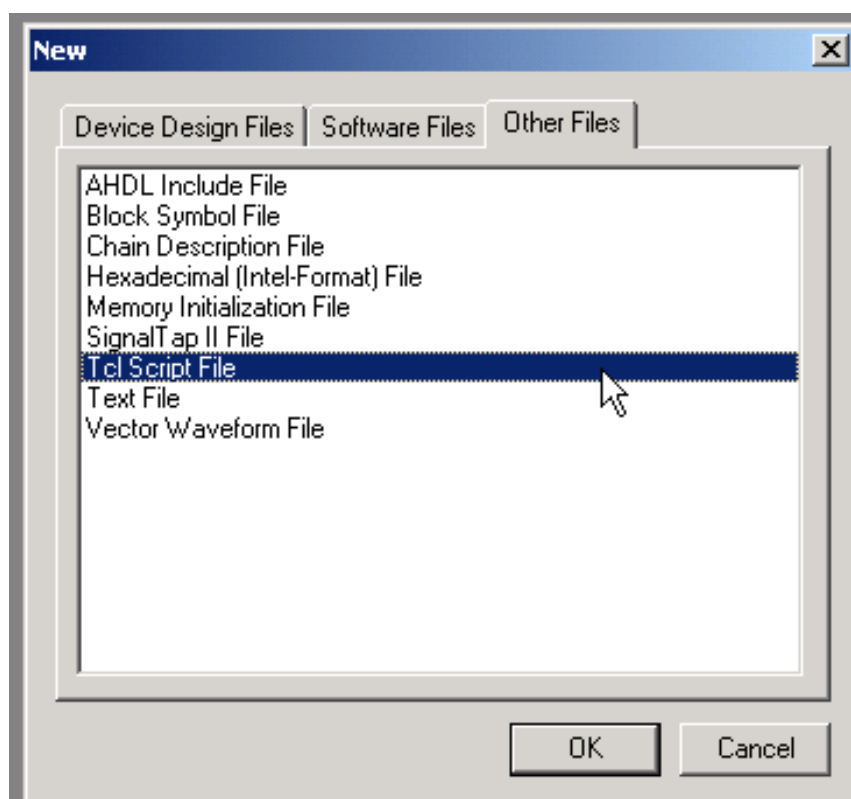


4. 分配管脚

为芯片分配管脚可以用 QuartusII 软件里的“Assignments→Pins”菜单，也

可以用 tcl 脚本文件。用 Tcl 文件进行配置可重用性好，易于管理，因此本文介绍用 tcl 的方法。对于另一种方法，可以参考 QuartusII 软件的帮助文档。

在工程目录下建立一个 name 为 Setup.tcl 的 file。File→New，选择 other files 页面：



有关 tcl 文件的更详尽内容可参考 QuartusII 的帮助文档，对于我们所选用的学习套件来说，由于不同型号的 FPGA 核心板的管脚与实验板上的引脚也不同，因此不同的核心板对应的.tcl 文件也不同（关于核心板引脚与实验板引脚对应的详细情况请参照“CT-SOPCx 学习套件用户手册”或相关电路原理图）。在实际项目中，该文件也可以根据具体管脚分配要求来改写。

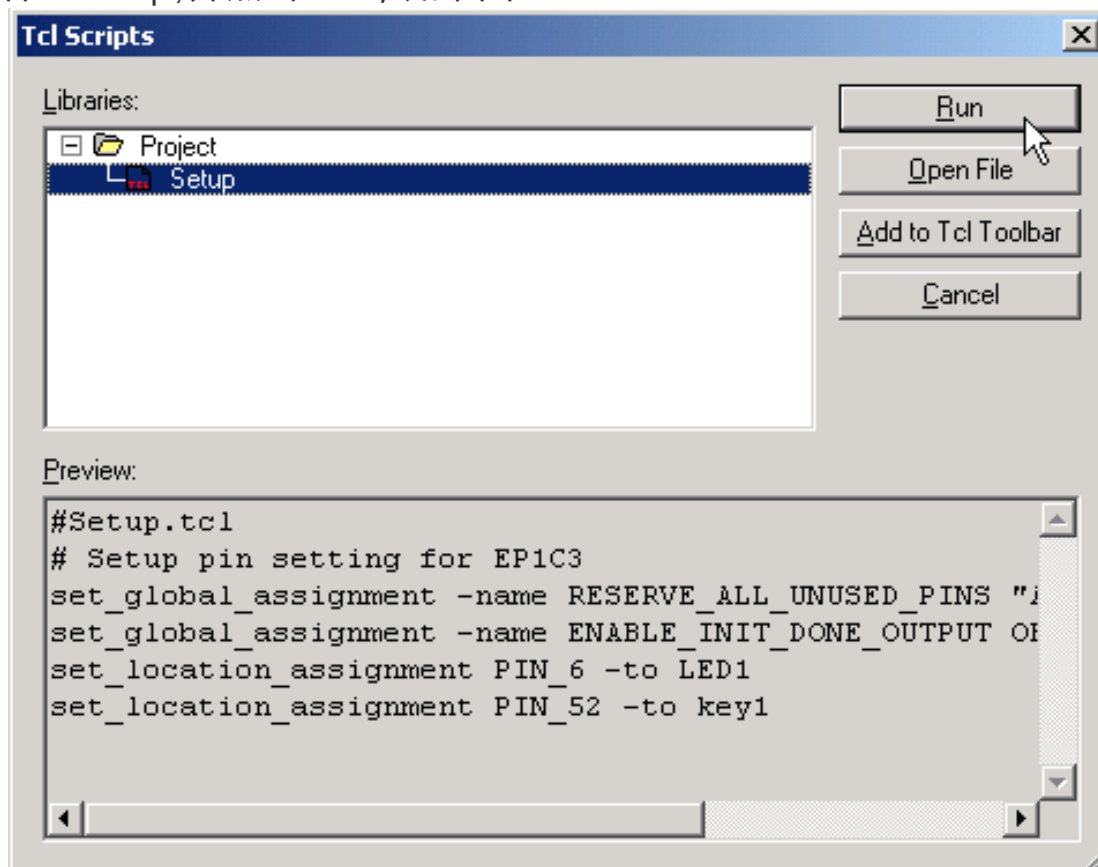
（1）对应于 EP1C3 核心板：

```
#Setup.tcl
# Setup pin setting
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT TRI-STATE"
set_global_assignment -name ENABLE_INIT_DONE_OUTPUT OFF
set_location_assignment PIN_6 -to LED1
set_location_assignment PIN_52 -to key1
```

（2）对应于 EP1C6 核心板：

```
#Setup.tcl
# Setup pin setting
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT TRI-STATE"
set_global_assignment -name ENABLE_INIT_DONE_OUTPUT OFF
set_location_assignment PIN_1 -to led1
set_location_assignment PIN_156 -to key1
```

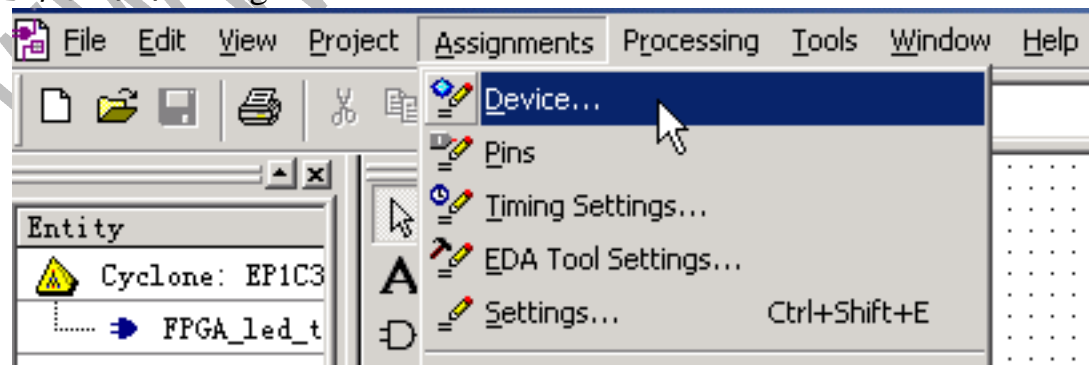
保存到工程目录下，并注意在保存对话框选上“Add file to current project”选项。然后打开 Tools -> Tcl Scripts,选中刚才编辑的 Script 文件：Setup,并点击 Run，如下图：



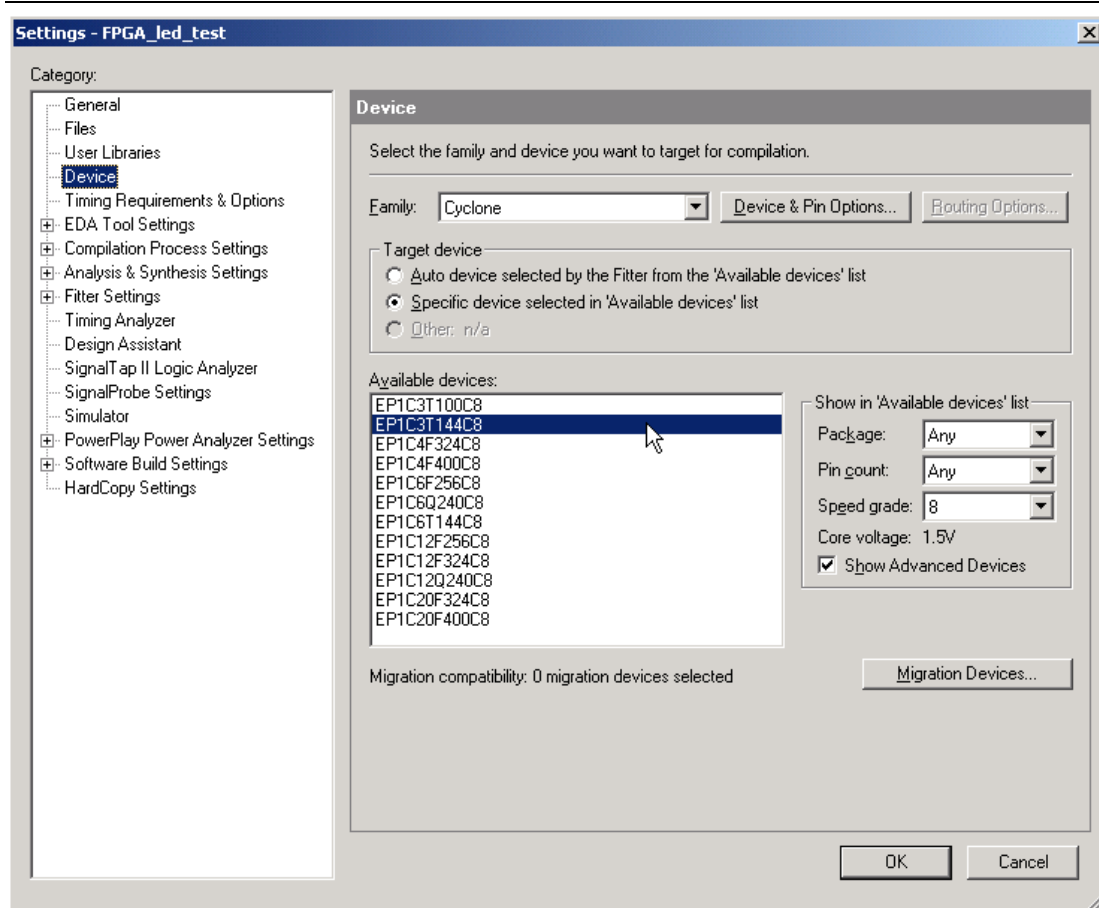
注意：建立工程时如果路径名有中文字符或者路径名有空格字符，则 tcl Script 文件将运行不了。比如，本例建立的工程“FPGA_led_test”目录是：C:\altera\quartus42\myq2projects\FPGA_led_test 如果是：

C:\altera\quartus42\my q2projects\FPGA_led_test\
在该目录下运行工程里的 setup.tcl 就会出错。

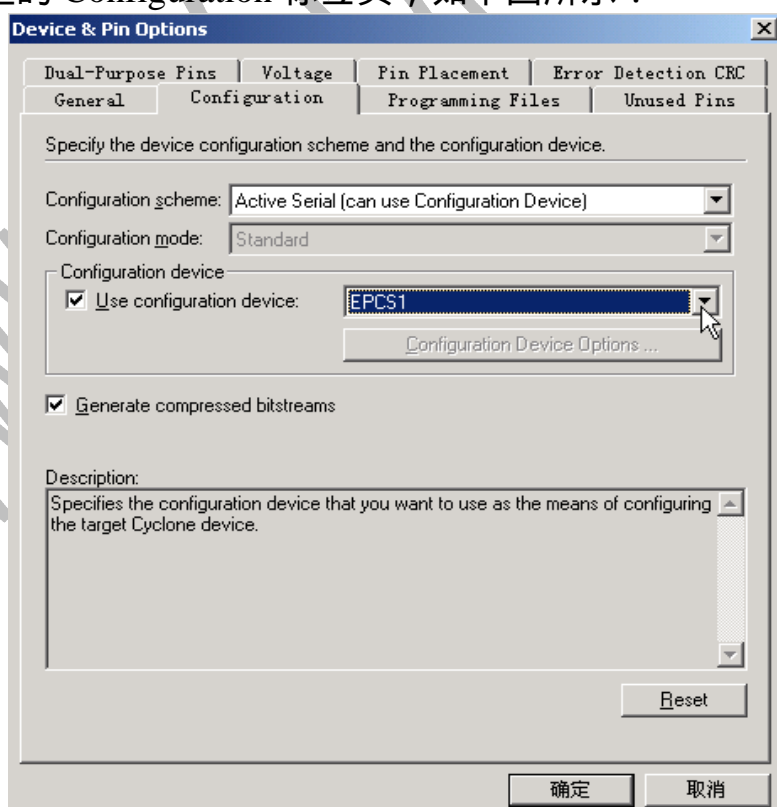
5. 设置。在建立工程的时候我们选定了芯片型号，其实也可以在这一步设定芯片型号，在菜单 Assignments→Device：



弹出设置对话框：



根据核心板的 FPGA 选择芯片型号，如上图所示选择 EP1C3T144C8。点击设置对话框的“Device & Pin Options”按钮弹出 Device & Pin Options 对话框，并选择该对话框的 Configuration 标签页，如下图所示：



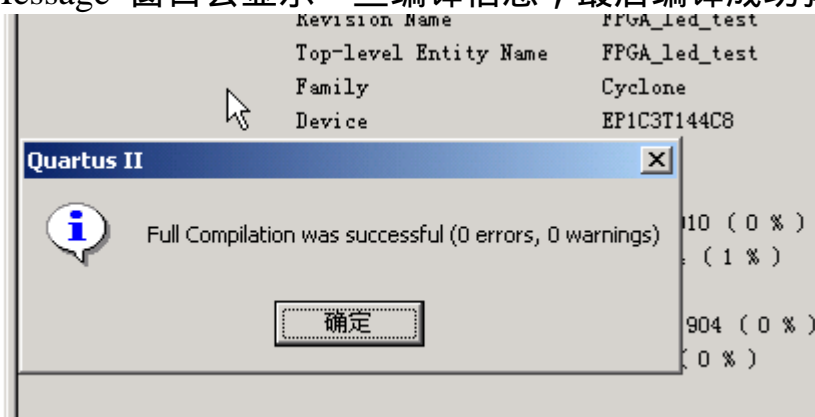
在 Configuration 标签页上，选择如上图所示内容。其余留缺省设置即可。

按确定退出该对话框，按 OK 退出设置对话框，返回到顶层图界面。

6. 编译。按主工具栏上的编译按钮即开始编译：

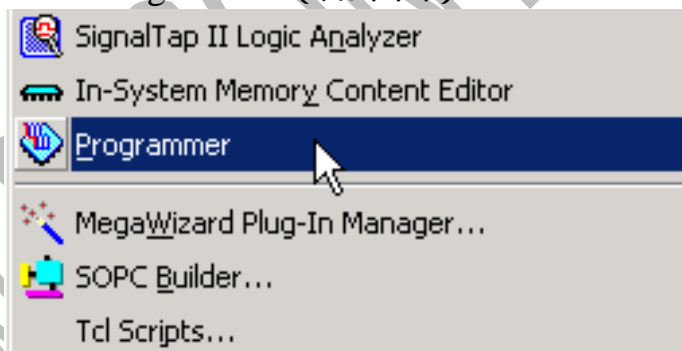


QuartusII 下面的 Message 窗口会显示一些编译信息，最后编译成功弹出提示：

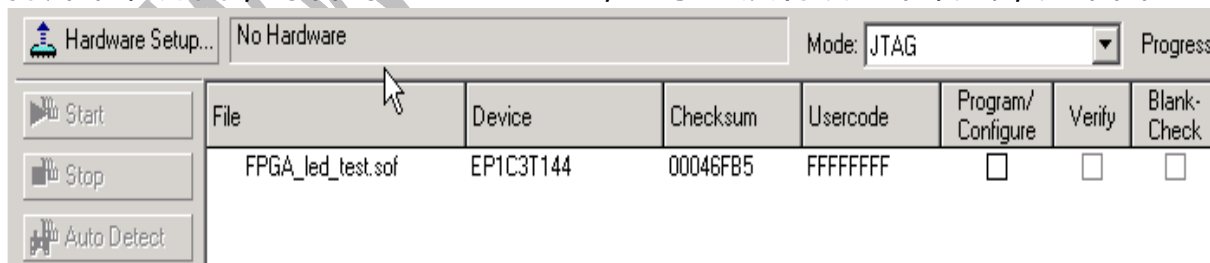


7. 下载。

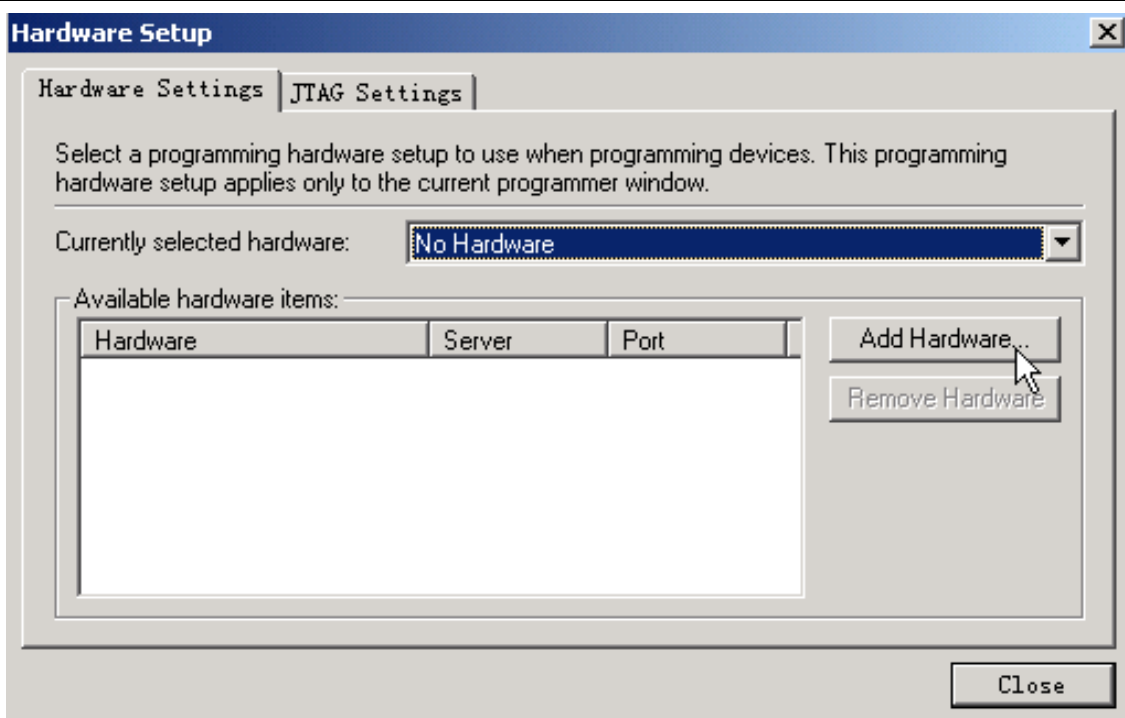
(1) 下载设置：如果第一次使用下载线下载配置文件到 FPGA，则需要在 Q2 软件设置下载线的型号等信息。先将 ByteBlasterII 下载线的一头接到 PC 的并口，执行菜单 Tools→Programmer（如下图）



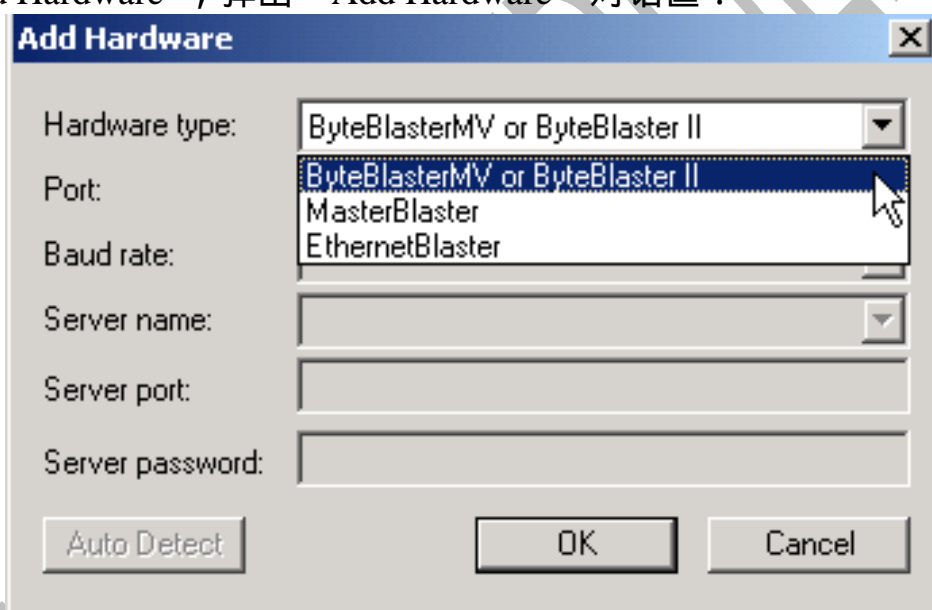
打开下载界面，可看到 No Hardware，表示还没有设置下载线，如下图：



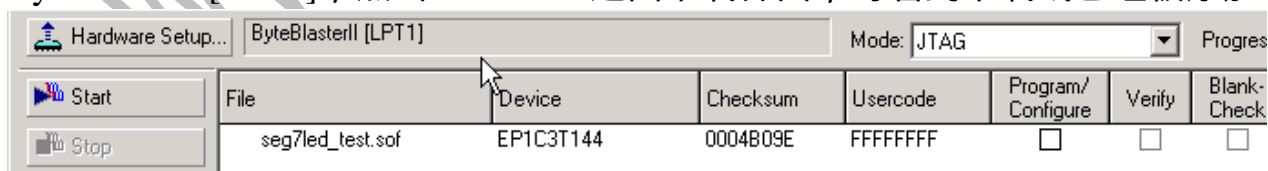
点击 Hardware Setup，弹出“Hardware Setup”对话框：



点击“Add Hardware”，弹出“Add Hardware”对话框：



在 Hardware type 下拉列表选择“ByteBlasterMV or ByteBlasterII”，点 OK 返回“Hardware Setup”对话框，从“Currently selected hardware:”下拉列表选择 ByteBlasterII[LPT1]，点击“Close”返回下载界面，可看到下载线已经被添加：



(2) 将ByteBlasterII下载线一头与PC连接，另一头插入到JTAG口或EPCS1的下载口**注意：JTAG口或EPCS1下载口的Pin1必须与ByteBlasterII的10针插头的Pin1对应。**

(3) FPGA 核心板接上 5V 电源。

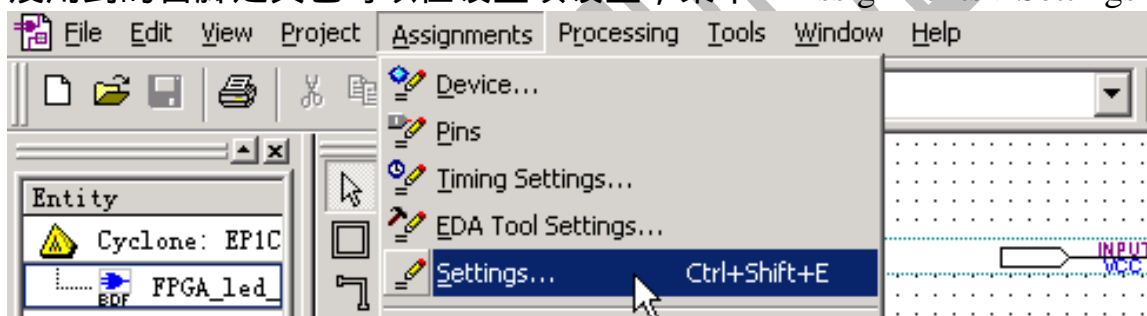
(4) 如果只是调试一下设计是否成功，可通过 JTAG 口把芯片的配置信息

下载到 FPGA 芯片内，掉电后配置信息丢失。此时，下载界面的“ Mode :” 下拉列表应选择“ JTAG ”，并选择工程中.sof 后缀的文件进行下载。（注意记得在“ Program/Configure ”那个方框那里打上“ ”，其它“ Verify ”、“ Blank Check ”等可根据需要选择。）

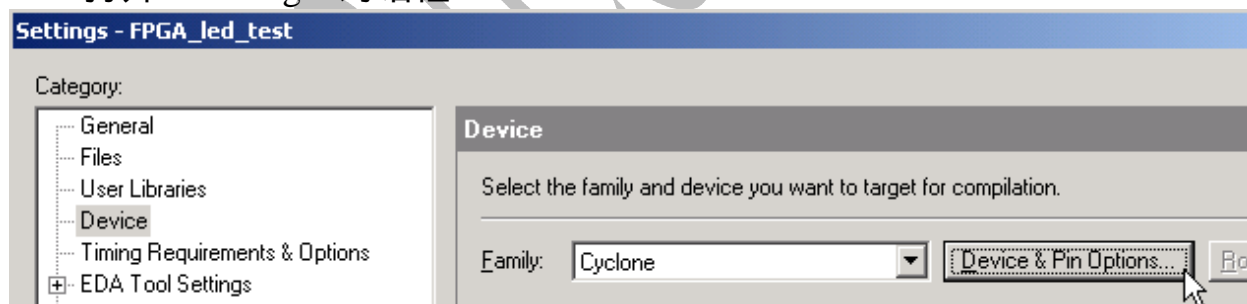
(5) 如果要下载到非易失 Flash 配置芯片（掉电后配置信息不丢失）里去，则将 ByteBlasterII 插到 EPCS1 芯片的下载口。并在下载界面的“ Mode :” 下拉列表应选择“ Active Serial Programming ”，并选择工程中.pof 后缀的文件进行下载。

下载完后可以发现核心板上只有第一个LED (D1) 点亮，原因是我们在 Setup.tcl文件中指定没用的管脚为输入&三态 (set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT TRI-STATED"), (**注意，如果是 EP1C6 核心板V2.0，请务必将未用管脚定义为“ As inputs, tri-stated”，因为必须确保该核心板上当前没用到的Flash芯片与FPGA之间以高阻态相隔，否则会发生损坏芯片的危险！**)核心板上的LED是低电平点亮，第二个发光二极管 (D2) 对应的FPGA管脚因为没有用而设为输入&三态，因此不亮；而由于D1 的是由内部非门控制，所以按钮 1 (KEY1) 没按下时是亮的。

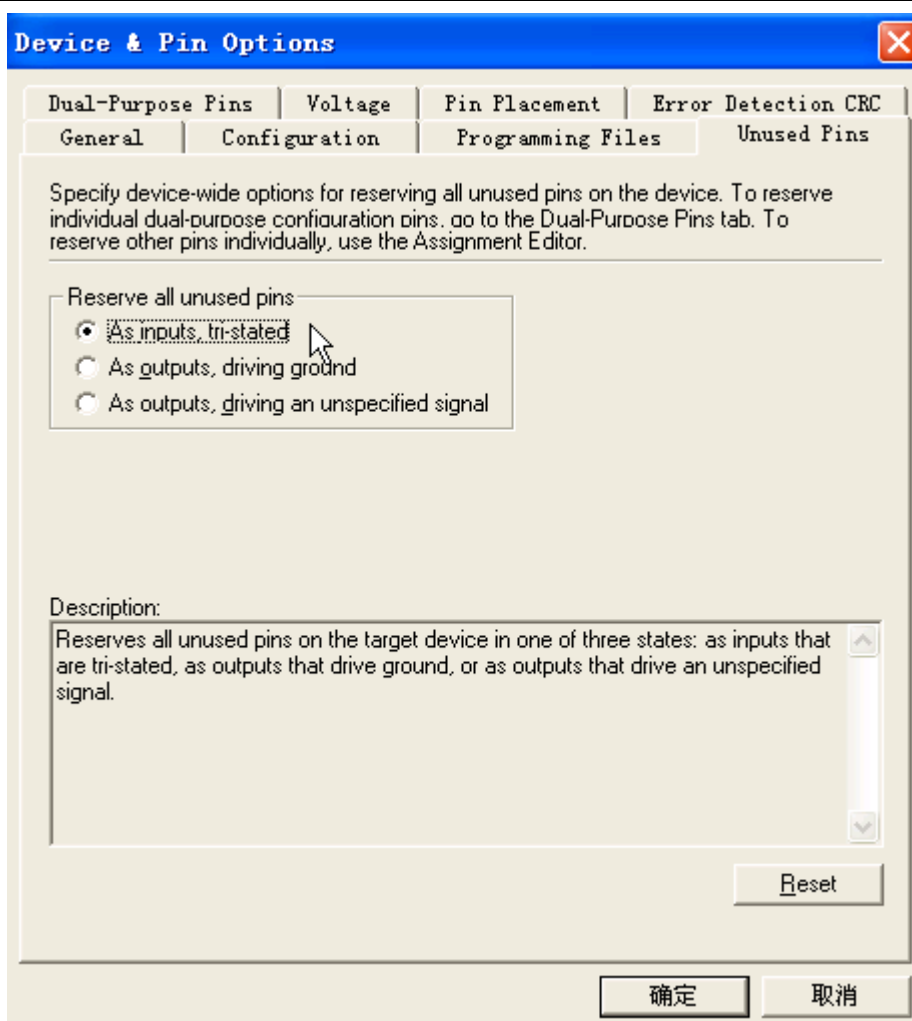
没用到的管脚定义也可以在设置项设置，菜单“ Assignments->Settings ”：



打开“ Setting ”对话框：



再点击“ Device & Pin Options ”，弹出“ Device & Pin Options ”对话框：



在“Unused Pins”标签页，有未用到管脚的设置选项（“Reserve all unused pins”），用户可以根据需要选择。“Device & Pin Options”对话框还有其它设置的标签页，大家可以看看其它的吧，多熟识一下 QuartusII 的界面。

8. 结果。在本例的设计中，我们利用一个常开按钮（实验板上的 KEY1）作为输入（常开时输入 1，闭合时输入 0），经过一个反向器后输出到核心板的第一个 LED；因此按下（闭合）实验板上的 KEY1，该 LED 灭。该设计非常简单，并没有实际的用处，然而我们就是需要简单，目的就是为了解更容易理解 FPGA 的设计流程和 QuartusII 软件的基本用法。虽然简单，也用去了不少篇幅啊，可能有人会略嫌罗嗦啦，呵呵，耐心点，我相信还是有不少刚接触这个软件或刚接触 FPGA 设计的吧，看到自己居然能用按钮控制这个 LED 的亮和灭了，说不定会激动良久呢！呵呵，两年前我就是这样啊。

（二）实验二：实验板上的按钮控制数码管

目的：

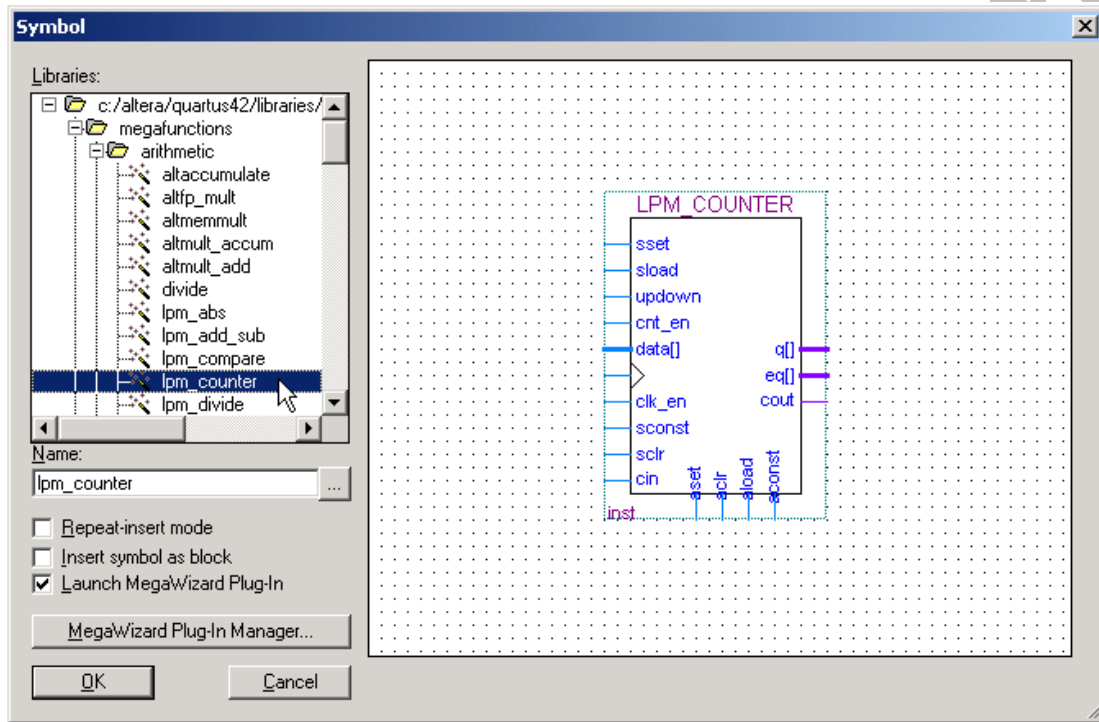
- ✧ 进一步熟识利用 QuartusII 软件进行 FPGA 设计的流程；
- ✧ 掌握利用现有的宏功能模块进行设计；
- ✧ 熟识自行设计功能模块的方法。

原理：

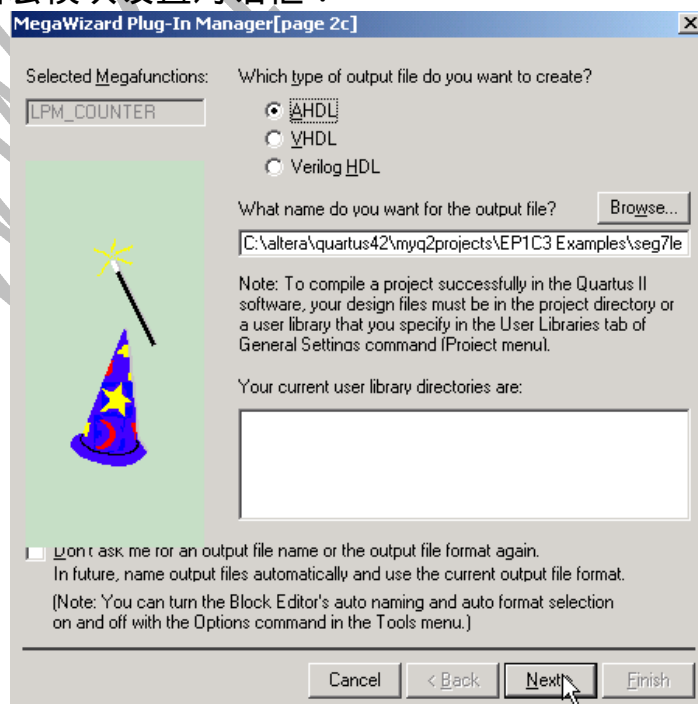
设计一个 8 位的可逆计数器，利用实验板上的三个按钮开关分别作为“减少计数值”、“增加计数值”和“清零计数值”的输入。计数器的 8 位二进制输出用两个七段数码管来显示，其中七段数码管的译码模块由用户添加，本例中我们用 VHDL 语言来设计该功能模块，并添加到设计中。

1. 建立工程。参照实验一的步骤建立一个名为“seg7led_test”的工程，并建立顶层图。

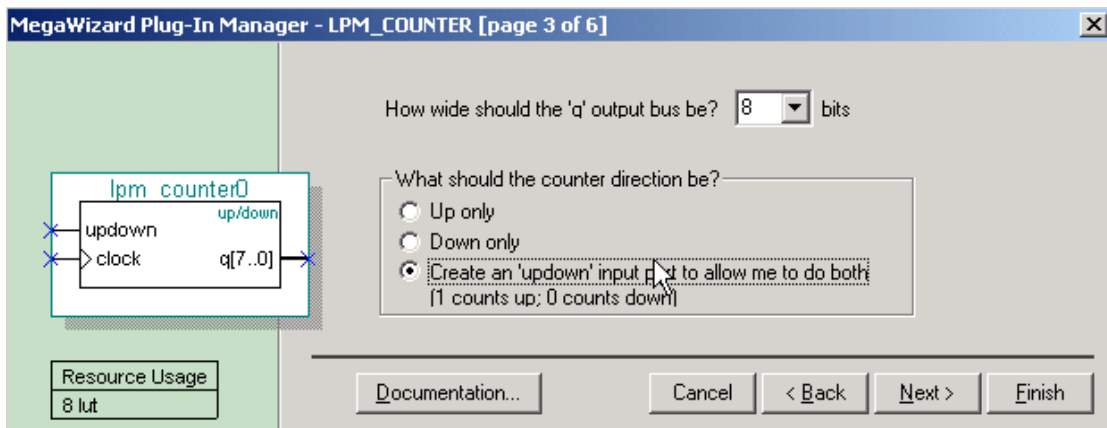
2. 使用宏功能模块。使用 Altera 公司提供的宏功能模块 (LPM)，本例中我们要利用计数器 LPM。双击顶层图空白处，弹出 symbol 对话框，展开 Libraries，找到 lpm_counter，如下图所示：



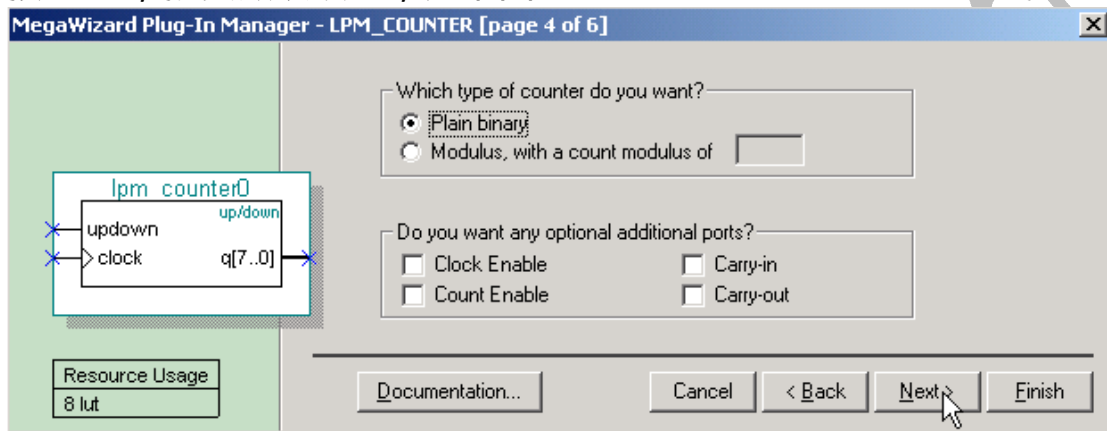
按 OK，弹出宏模块设置对话框：



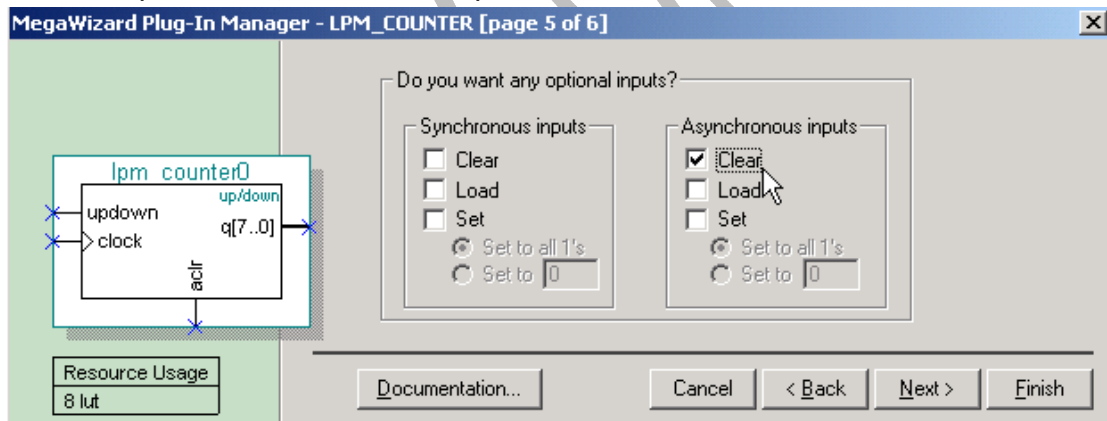
接受默认设置，并按 Next，计数器的宽度设为 8 位，选上可逆计数选项，如下图：



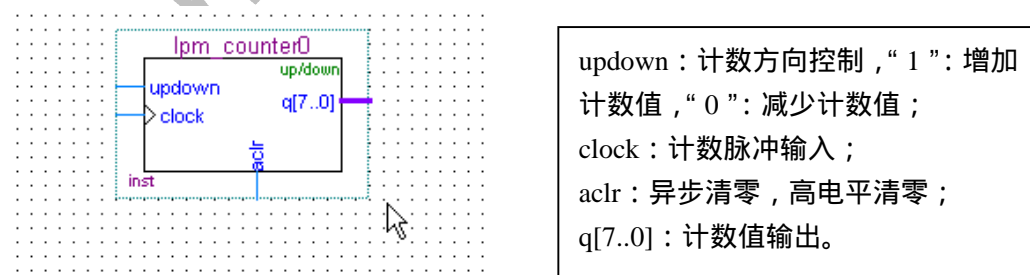
按 Next，接受默认设置，如下图：



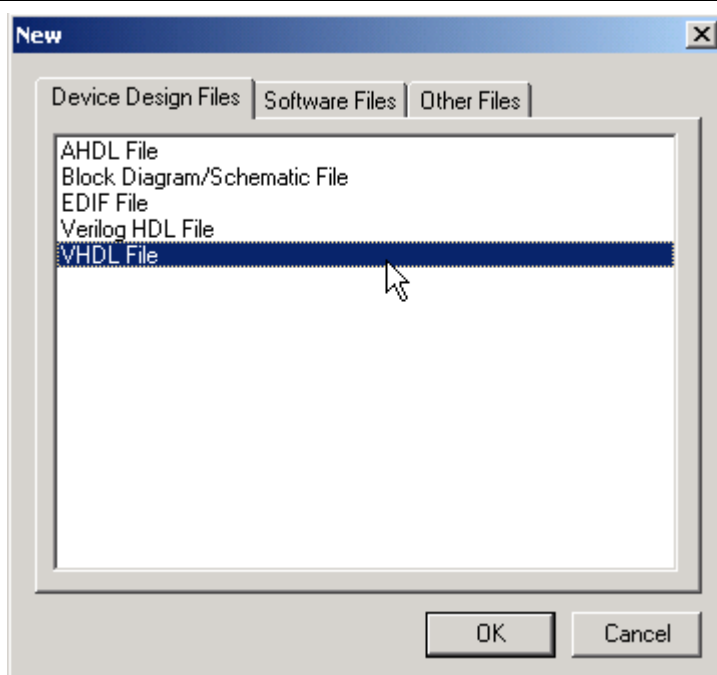
按 Next，选上异步清零输入，如下图：



按 Next，接受默认设置，再按 Finish 即完成 LPM_Counter 宏功能模块的设置。在顶层图上点击鼠标即可将该计数器模块加入到工程中，如下图：



3. 设计七段数码管的译码模块。打开 File→New 如图：



点击OK后建立一个VHDL文件，另存为bin27seg.vhd。**注意：QuartusII要求文件名必须与代码中的实例名（entity）相同。**编写七段译码模块的VHDL代码（该文件在该例子目录中能找到）：

```
library IEEE;
use IEEE.std_logic_1164.all;
entity bin27seg is
    port (
        data_in : in std_logic_vector (3 downto 0);
        data_out : out std_logic_vector (6 downto 0)
    );
end entity;
architecture bin27seg_arch of bin27seg is
begin
    process(data_in)
    begin
        case data_in is
            when "0000" => data_out <= "0111111"; -- 0
            when "0001" => data_out <= "0000110"; -- 1
            when "0010" => data_out <= "1011011"; -- 2
            when "0011" => data_out <= "1001111"; -- 3
            when "0100" => data_out <= "1100110"; -- 4
            when "0101" => data_out <= "1101101"; -- 5
            when "0110" => data_out <= "1111100"; -- 6
            when "0111" => data_out <= "0000111"; -- 7
            when "1000" => data_out <= "1111111"; -- 8
            when "1001" => data_out <= "1100111"; -- 9
            when "1010" => data_out <= "1110111"; -- A
            when "1011" => data_out <= "1111100"; -- b
            when "1100" => data_out <= "1011000"; -- c
```

```

when "1101" => data_out <= "1011110"; -- d
when "1110" => data_out <= "1111001"; -- E
when "1111" => data_out <= "1110001"; -- F
when others => NULL;

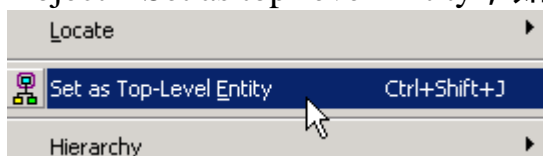
```

```
end case;
```

```
end process;
```

```
end architecture;
```

输入完成后，将该文件设为顶层实体（因为 Q2 分析、综合和编译等都针对顶层实体），该命令在 Project→Set as top-level Entity，如下图：



分析该 vhd 设计文件：执行工具栏处的“Start Analysis & Synthesis”命令按钮，开始分析综合，这个步骤在这里用于检查设计错误。

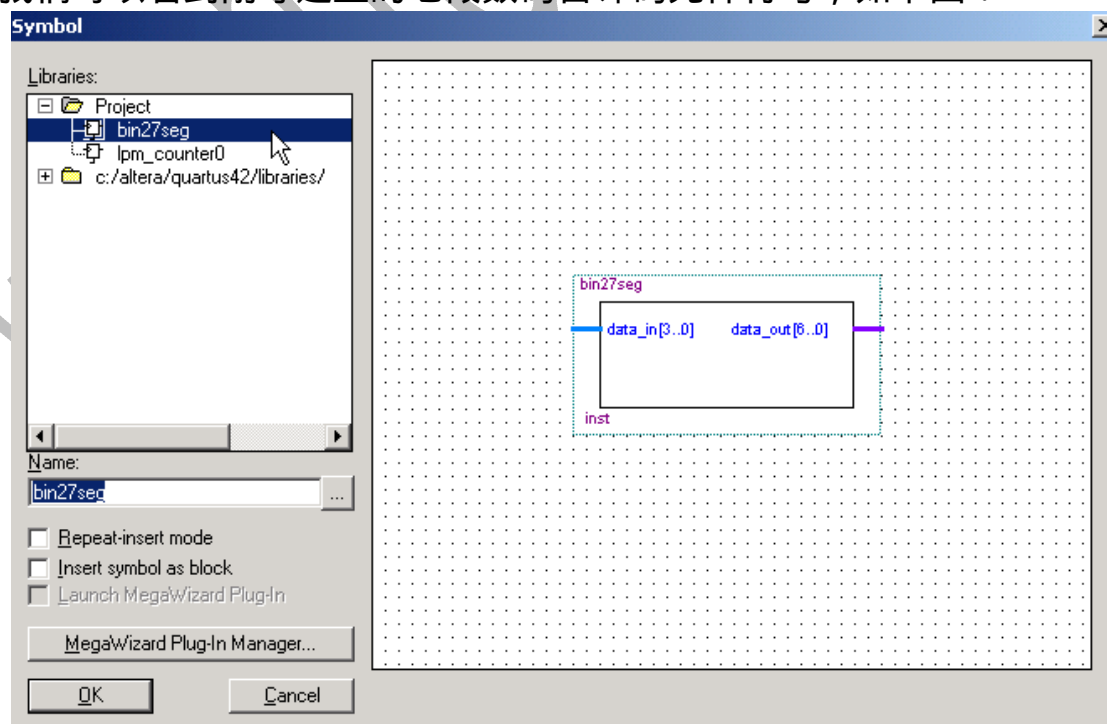


分析成功后我们要生成一个七段数码管的译码元件符号（Symbol），执行 File→Create/Update→Create Symbol files for current file，开始建立该文件的元件符号。

4. 设计完整的顶层图。

我们返回顶层原理图，并 **注意重新将顶层原理图设为顶层实体(Project→Set as top-level Entity)**，可以这样理解，编译操作都是针对顶层实体进行的。

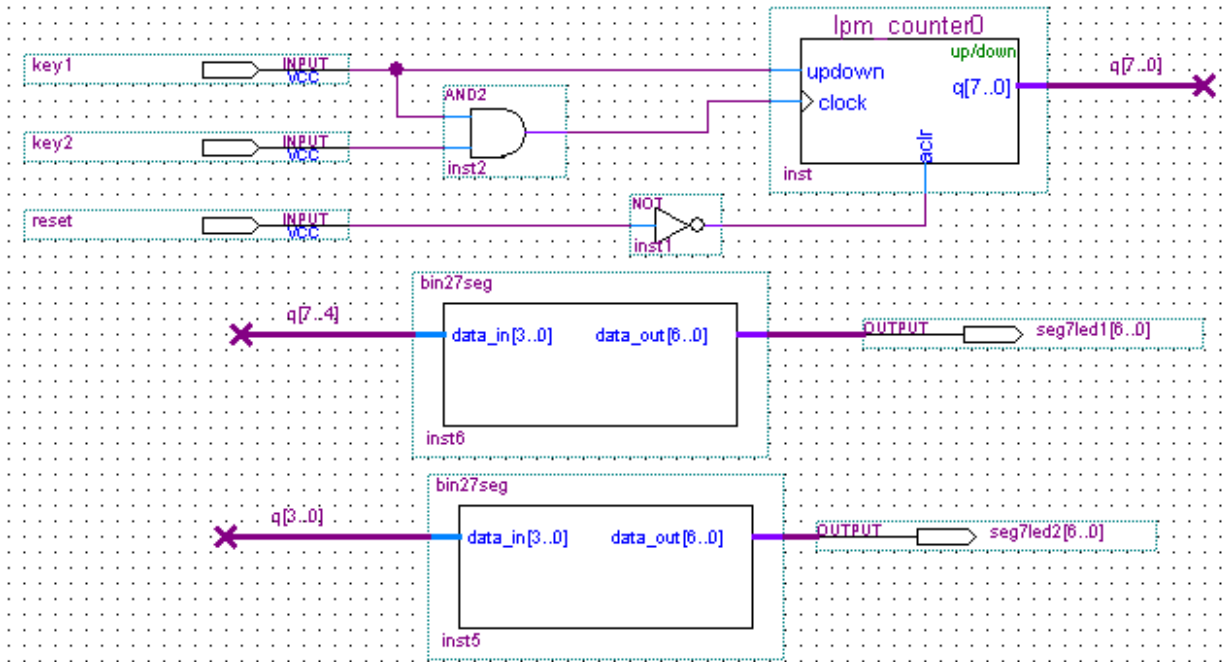
双击顶层图空白的地方，弹出 symbol 对话框，展开 Libraries 栏的 Project 库，我们可以看到刚才建立的七段数码管译码元件符号，如下图：




按 OK，在图纸上空白地方点击即可输入七段数码管译码元件，共需在图纸

上输入两个七段数码管译码元件。

添加其它的元件，并完成如下图所示连接（如果不明白如何添加，请参考本教程实例一）：



从上图可以看出，基于原理图的设计输入类似于在 protel 软件中绘制电路图，上图中粗线属于总线类型，它的宽度（例如 $q[3..0]$ 标识从 $q3$ 到 $q0$ 共 4 条电气连线）必须与元件端口的宽度一致，网络标号相同的线表示相互连接。例如上图中， $q[7..4]$ 网络标号表示 inst6 元件（七段数码管）的 data_in[3..0] 与 inst（计数器）的高 4 位线相连，即 $q[7..4]$ 。

总线与网络标号的编辑：将鼠标移到元件的端口处，变成  形状，按住左键，拖动到所需的长度，松开鼠标，拉出来的线呈被选择的状态，此时直接从键盘敲入网络标号即可；如果线没有呈被选择的状态，则左键点击线选上。

5. 设置芯片和管脚。参照实验一的第 6 点设置芯片型号、配置芯片型号等内容。

参照以下 tcl script 文件配置芯片管脚，并运行该 tcl 脚本。

1 对于 EP1C3 核心板，tcl 管脚配置文件为（关于核心板引脚与实验板引脚对应的详细情况请参照“CT-SOPCx 学习套件用户手册”或相关电路原理图）：

```
#Setup.tcl
# Setup pin setting for EP1C3 main board
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT TRI-STATE"
set_global_assignment -name ENABLE_INIT_DONE_OUTPUT OFF
set_location_assignment PIN_52 -to key1
set_location_assignment PIN_53 -to key2
set_location_assignment PIN_54 -to reset
set_location_assignment PIN_68 -to seg7led1\[0\]
set_location_assignment PIN_61 -to seg7led1\[1\]
set_location_assignment PIN_56 -to seg7led1\[2\]
```

```
set_location_assignment PIN_55 -to seg7led1\[3\  
set_location_assignment PIN_59 -to seg7led1\[4\  
set_location_assignment PIN_67 -to seg7led1\[5\  
set_location_assignment PIN_62 -to seg7led1\[6\  
set_location_assignment PIN_72 -to seg7led2\[0\  
set_location_assignment PIN_69 -to seg7led2\[1\  
set_location_assignment PIN_58 -to seg7led2\[2\  
set_location_assignment PIN_60 -to seg7led2\[3\  
set_location_assignment PIN_57 -to seg7led2\[4\  
set_location_assignment PIN_71 -to seg7led2\[5\  
set_location_assignment PIN_70 -to seg7led2\[6\  
2 对于 EP1C6 核心板，tcl 管脚配置文件为：
```

```
#Setup.tcl
```

```
# Setup pin setting for EP1C6 main board
```

```
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT  
TRI-STATE"
```

```
set_global_assignment -name ENABLE_INIT_DONE_OUTPUT OFF
```

```
set_location_assignment PIN_156 -to key1
```

```
set_location_assignment PIN_158 -to key2
```

```
set_location_assignment PIN_159 -to reset
```

```
set_location_assignment PIN_169 -to seg7led1\[0\  
set_location_assignment PIN_166 -to seg7led1\[1\  
set_location_assignment PIN_161 -to seg7led1\[2\  
set_location_assignment PIN_160 -to seg7led1\[3\  
set_location_assignment PIN_164 -to seg7led1\[4\  
set_location_assignment PIN_168 -to seg7led1\[5\  
set_location_assignment PIN_167 -to seg7led1\[6\  
set_location_assignment PIN_175 -to seg7led2\[0\  
set_location_assignment PIN_170 -to seg7led2\[1\  
set_location_assignment PIN_163 -to seg7led2\[2\  
set_location_assignment PIN_165 -to seg7led2\[3\  
set_location_assignment PIN_162 -to seg7led2\[4\  
set_location_assignment PIN_174 -to seg7led2\[5\  
set_location_assignment PIN_173 -to seg7led2\[6\  
6. 编译。 注意当前顶层实体是否顶层图，否的话必须执行菜单命令Project → Set  
as top-level Entity 将顶层图设为当前顶层实体。
```

```
7. 下载。下载完毕会看到两个七段数码管显示当前计数值“00”，按实验板上的  
KEY1 按钮，计数值增加；按 KEY2 则减小；按 RESET 按钮则清零。
```

第三章 SOPC 的基本开发流程

第一节 SOPC vs MCU、DSP和FPGA²

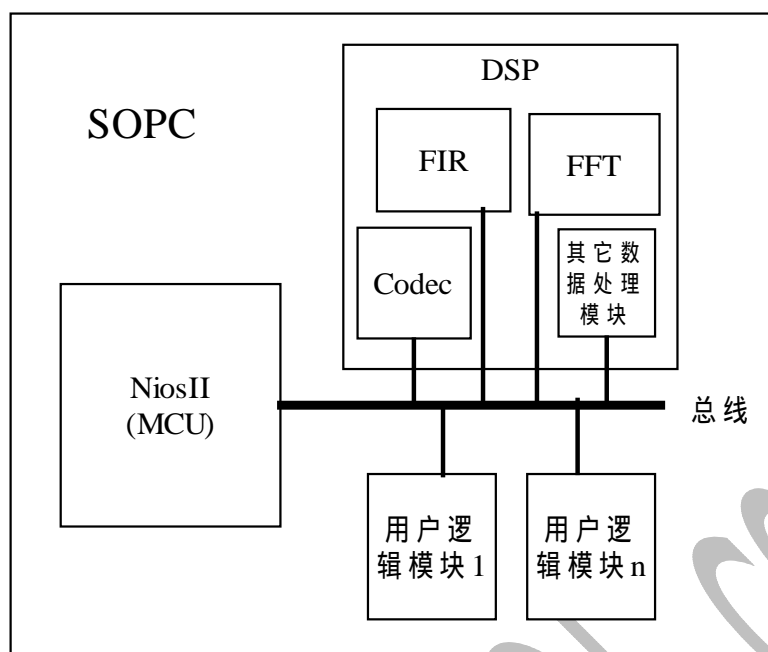
嵌入式系统发展朝着小体积、低功耗、高性能的趋势发展。MCU、DSP 和 FPGA 三种处理器在现代嵌入式系统中扮演的角色呈现三分天下的局面,它们各自具有独特的优势而在某方面又略显不足。以 51 系列单片机和 ARM 微处理器为代表的 MCU 家族因丰富的软件系统支持在控制和处理人机接口领域占据绝对的领先地位;然而在海量数据处理方面却被 DSP 占尽了风头;FPGA 在高速复杂逻辑处理方面独占风骚,并且最近异军突起,凭借其超大规模的单芯片容量和硬件电路的高速并行运算能力,在信号处理方面也显示出突出的优势。因而,MCU、DSP、FPGA 的结合将是未来嵌入式系统发展的趋势。

SOPC = MCU + DSP + FPGA, SOPC 可以将 MCU、DSP 和 FPGA 完美结合!

- (1) SOPC \supseteq MCU: 目前,在大容量 FPGA 中可以嵌入 16 位或 32 位以上的 MCU。如 Altera 公司的 FPGA 可嵌入一个或多个软核 CPU(Nios 或 NiosII), 或预嵌入 ARM 等微处理器;
- (2) SOPC \supseteq DSP: DSP 对海量数据快速处理的优异性能主要在于它的流水线计算技术,只有规律的加减乘除等运算才容易实现流水线的计算方式。然而,这种运算方式也较容易用 FPGA 的硬件门电路来实现。目前,实现各种 DSP 算法的 IP 核已经相当丰富和成熟,例如,FFT、IIR、FIR、Codec 等等。有实践证明,用 FPGA 实现的 MPEG4 压缩/解压速度比通用 DSP 实现快 10 倍以上。利用相关工具(例如 DSP Builder)可以很方便地把现有的数字信号处理 IP 添加到工程中去。
- (3) SOPC \supseteq FPGA: SOPC 一般采用大容量 FPGA 作为载体,除了在一片 FPGA 中定制 MCU 处理器和 DSP 功能模块外,还可以设计其它逻辑功能模块,实现 MCU+DSP+FPGA 在一片芯片上集成。如可采用 ALTERA 公司的 Cyclone、Stratix、StratixII 等大容量 FPGA 实现片上系统。

下图是一个典型的基于 Altera 公司大容量 FPGA 的 SOPC 结构图。其中 NiosII 可以采用 Altera 公司的 SOPC Builder 来定制,DSP 采用 DSP Builder 来定制。

² 本节内容部分属作者个人观点,并不代表任何权威。



相对于单片机、ARM 等，SOPC 的应用还不多见；然而，数年后（甚至更短），也许 SOPC 的身影就像今天的单片机一样随处可见。因此，谁掌握了 SOPC 技术，谁就能成为未来嵌入式系统设计的先锋。

Altera 推出的 Nios II 系列嵌入式处理器扩展了目前世界上最流行的软核嵌入式处理器的性能，把 Nios II 嵌入到 Altera 的所有 FPGA 中，例如 StratixII、Stratix、Cyclone 等系列器件中，用户可以获得超过 200 DMIPS 的性能，用户可以从三种处理器以及超过 60 个的 IP 核中选择所需要的，Nios II 系统为用户提供了最基本的多功能性，设计师可以以此来创建一个最适合他们需求的嵌入式系统。

Nios II 处理器的优点和特性：

使用 Nios II 处理器的用户可以根据他们的需要来调整嵌入式系统的特性、性能以及成本，快速将产品推向市场，扩展产品的生命周期，并且避免由于处理器的更新换代带来的损失。

提高系统性能：

- 具有一系列的处理器核可供选择，其中包括了超过 200 DMIPS 性能的核；
- 实现任何数量的处理器或将不同的处理器核组合在一起；
- 增加了已有的处理器，在 FPGA 中添加一个或更多的 Nios II 软核处理器；
- 更低的系统成本；
- 通过将处理器、外设、存储器和 I/O 接口集成到一个单一的 FPGA 中，从而降低了系统成本、复杂性和功耗；
- 通过将 NiosII 处理器嵌入到低成本的 Cyclone FPGA 中只需花费 35 美分或者更低（编者：对大多数用户而言，NiosII 所占逻辑资源的成本大约是 10 - 20 元人民币，具体取决于所选 FPGA 的类型和 NiosII 的配置）。

应对产品的生命周期：

- 提供易用的设计工具从而快速将产品推向市场。
- 提供永久的，免费的许可从而使基于 Nios II 处理器的产品避免了处理器

的更新换代而带来的损失。

功能强大、易用的开发工具：

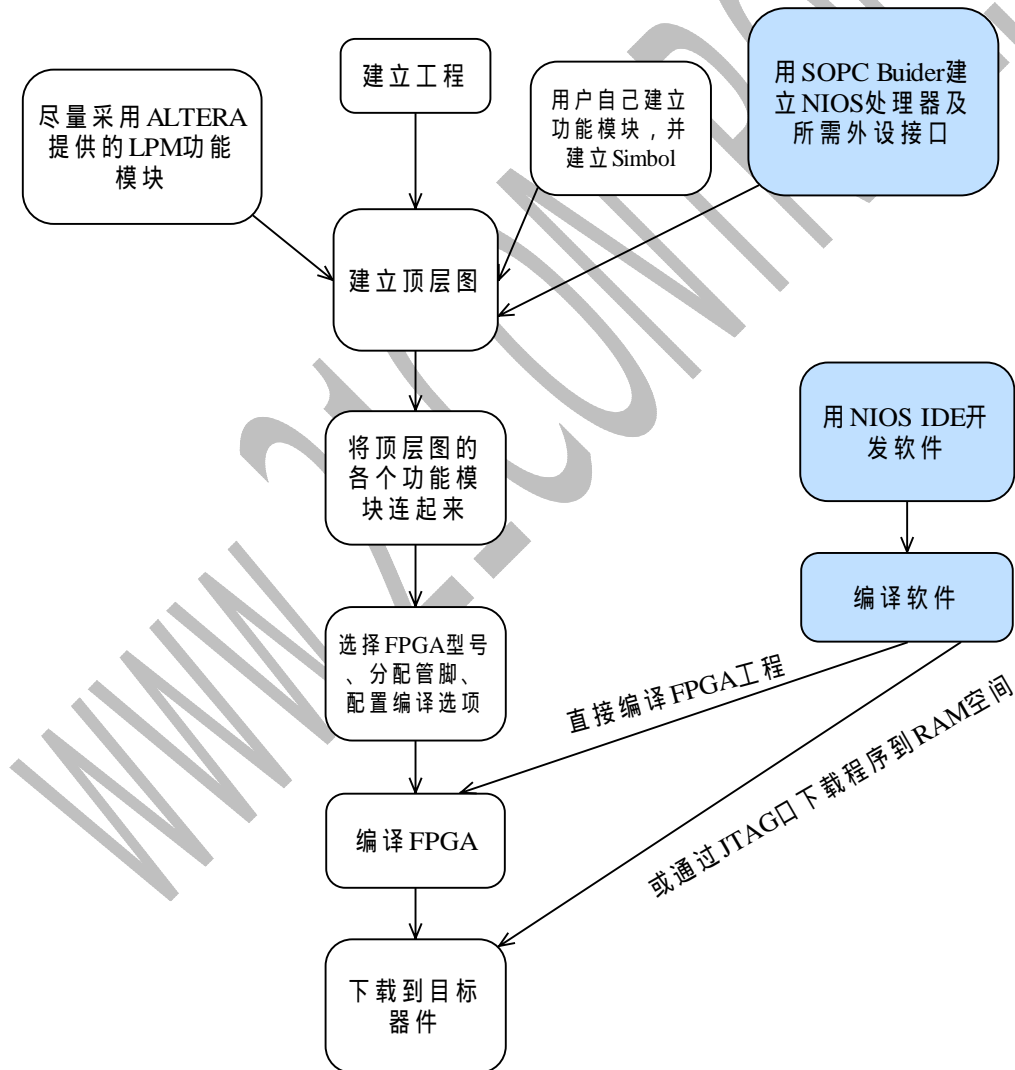
通过使用 Nios II 集成开发环境 (IDE)，从而加速了软件的开发；

利用 Altera 的强大的 SOPC Builder 系统开发工具和 Quartus II 设计软件可以在几分钟内设计一个系统。

第二节 基于 QuartusII 和 NiosII 的 SOPC 基本开发流程

下图是一个简化的基于 QuartusII 和 NiosII 的 SOPC 开发流程。从图中可见，SOPC 开发流程比 FPGA 的开发流程增加了处理器及其外设接口的定制步骤以及软件开发的步骤（阴影筐）。然而，这些新增加的步骤在 SOPC Builder（定制处理器和外设接口）、NiosII IDE（软件集成开发环境）工具的协助下显得相当轻松。

基于 QuartusII 和 NIOS 的 SOPC 设计流程



第三节 基于 QuartusII 和 NiosII 的 SOPC 实例

在第二章中，我们学习了基于 QuartusII 的 FPGA 设计基本流程，在掌握了基本的 FPGA 设计基础上，我们在本节中将学习基于 QuartusII 和 NiosII 的 SOPC 设计流程。

实验一：核心板上的两个 LED 交替闪烁

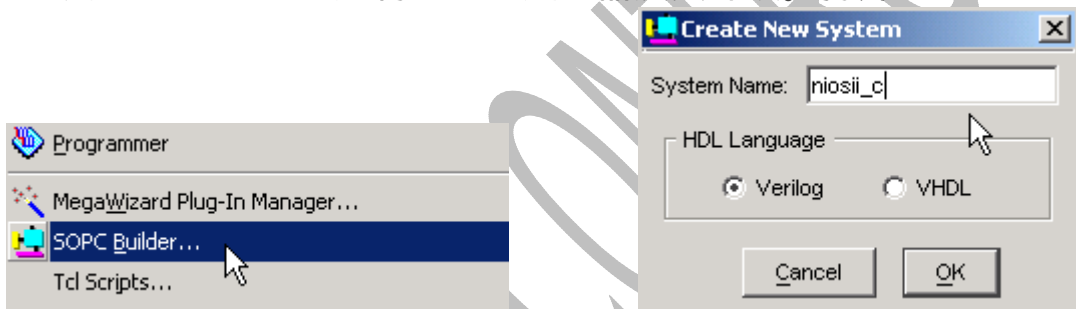
目的：

- (1) 掌握 NiosII 软核的定制流程；
- (2) 掌握 NiosII 软件开发流程；
- (3) 熟识 NiosII IDE 开发环境的使用；
- (4) 掌握基本的软件调试方法；
- (5) 学会使用 Cyclone 内部的 PLL 的使用方法。

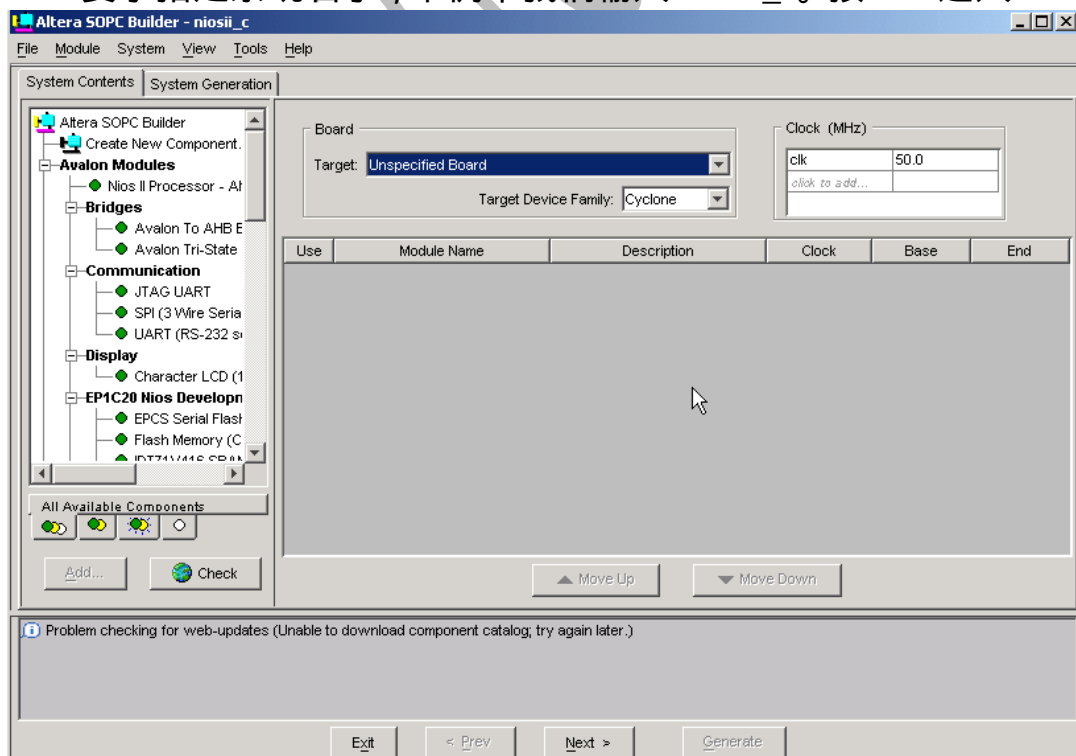
原理：在本实验中，我们用软件来控制核心板上的两个 LED 交替闪烁。

2. 新建工程。新建一个工程目录“sopc_led”，在此目录下建立一个名为“sopc_led”的 QuartusII 工程，并新建一个顶层图，保存于工程中。

3. 用 SOPC Builder 定制 NiosII 处理器及其外设。打开 Tools→SOPC Builder，



要求指定系统名字，本例中我们输入 niosii_c。按 OK 进入 SOPC 定制界面：



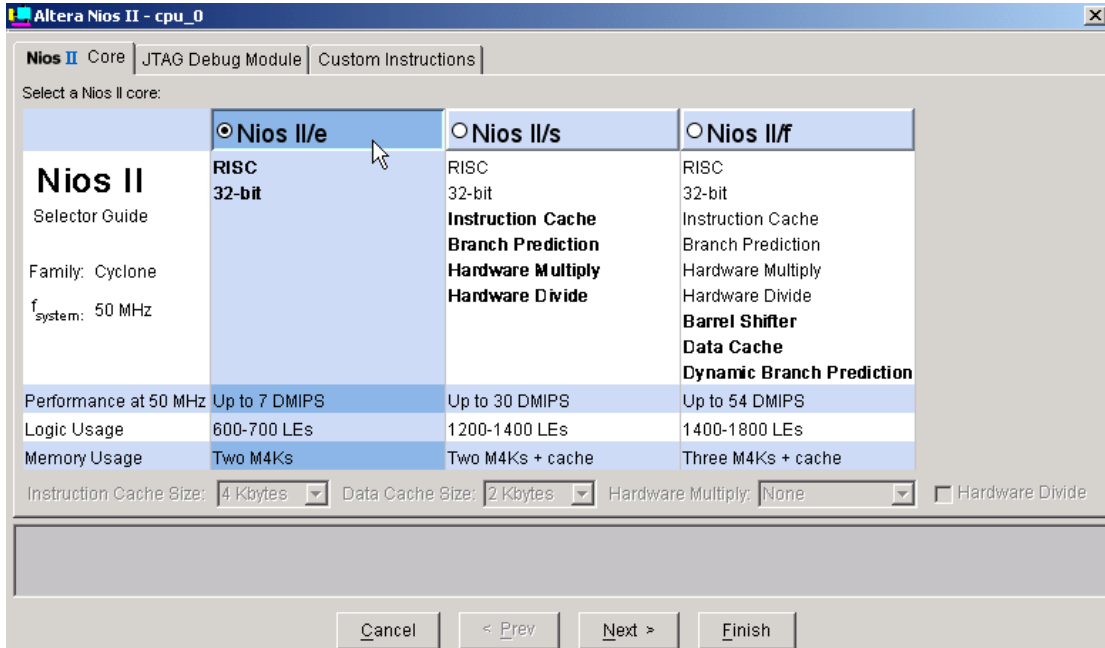
选择 Target，本例选择 Unspecified Board；

选择时钟频率，Clock (MHz)：本例选 50.0；

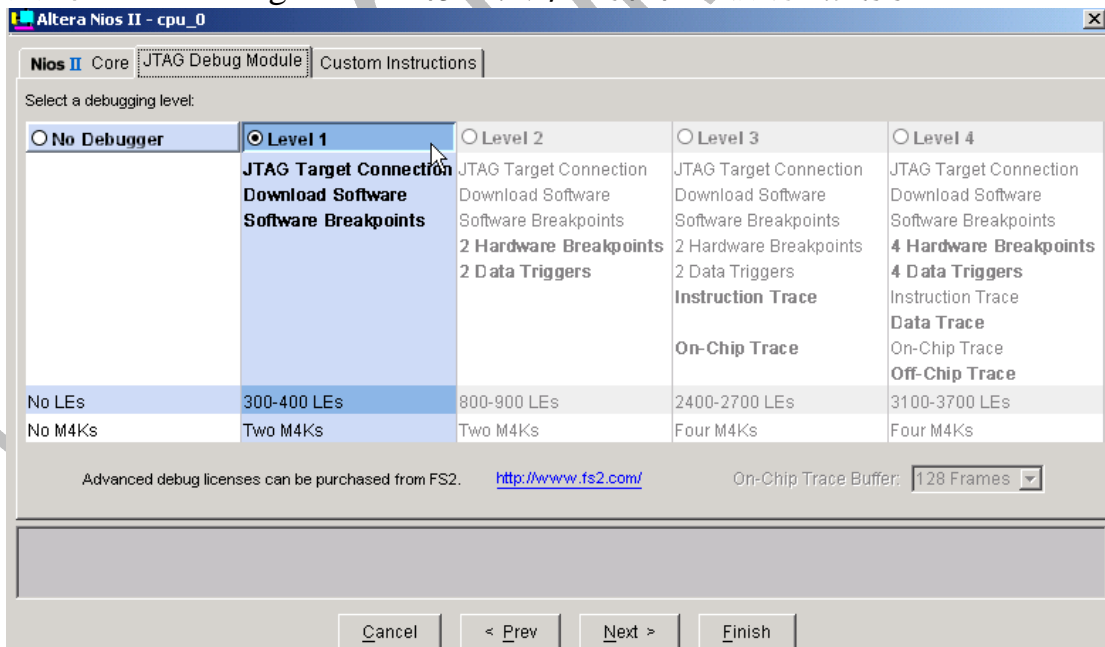
选择目标器件系列，Target Device Family：本例选 Cyclone。

在 SOPC 定制界面的左边，我们可以看到有很多功能模块，这些功能模块用户可以按照需要添加到所设计的系统中。

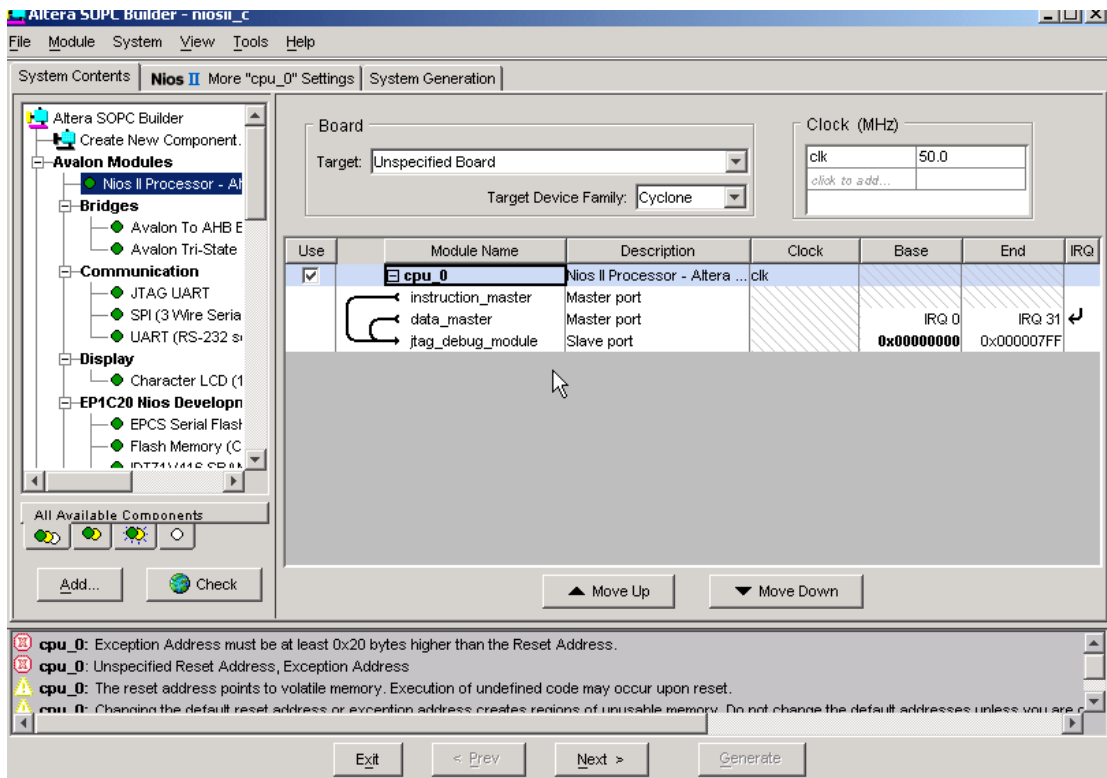
首先，我们需要一个 CPU，双击 NiosII Processor – Altera Corporation 弹出 Altera NiosII 对话框，我们选择一个经济型的 CPU 核，即 NiosII/e，如下图所示：



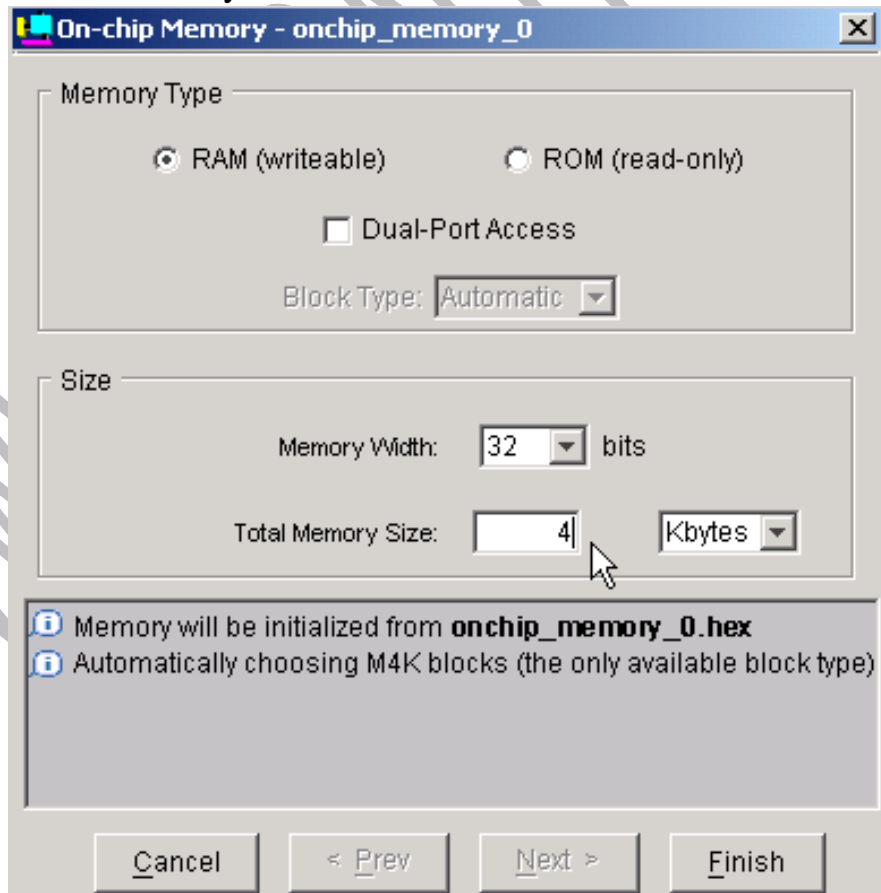
点击 JTAG Debug Module 标签页，选择第一级调试支持 Level1：



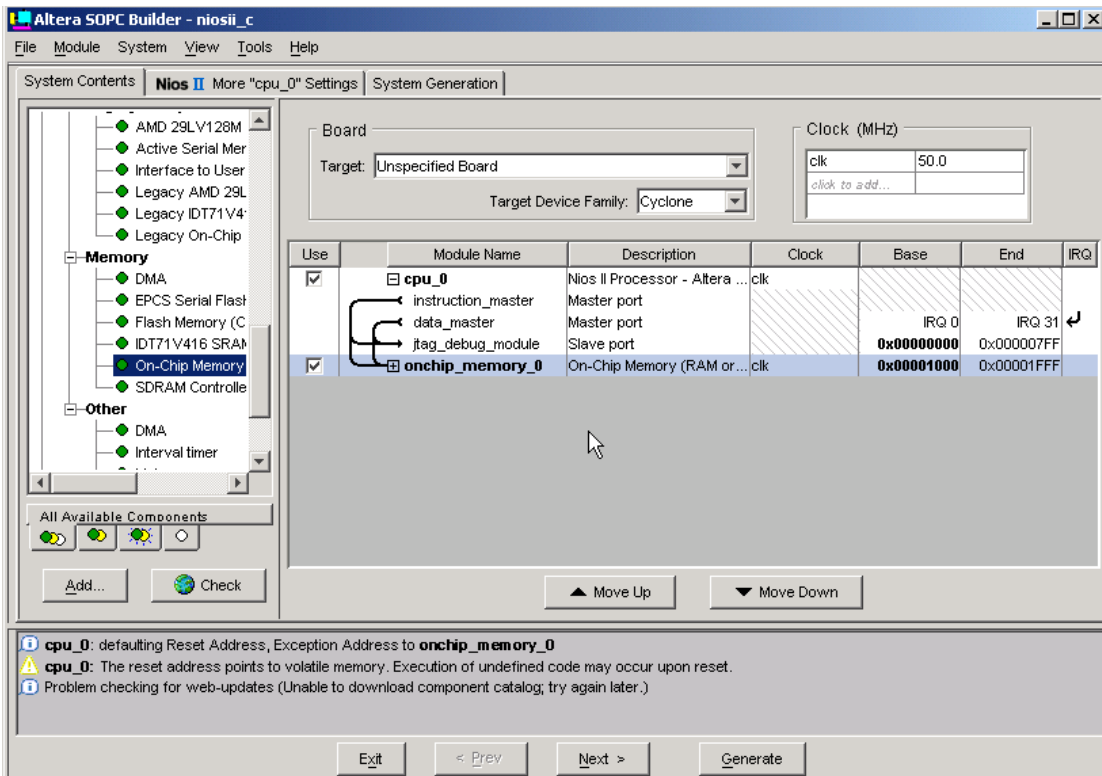
点击 Finish 完成 NiosII CPU 的配置工作。项目中会增加一个 niosII 处理器，名字为 cpu_0,为了简便起见，没有将它改名。改名的方法是:右键→ReName，输入名字后回车。如下图：



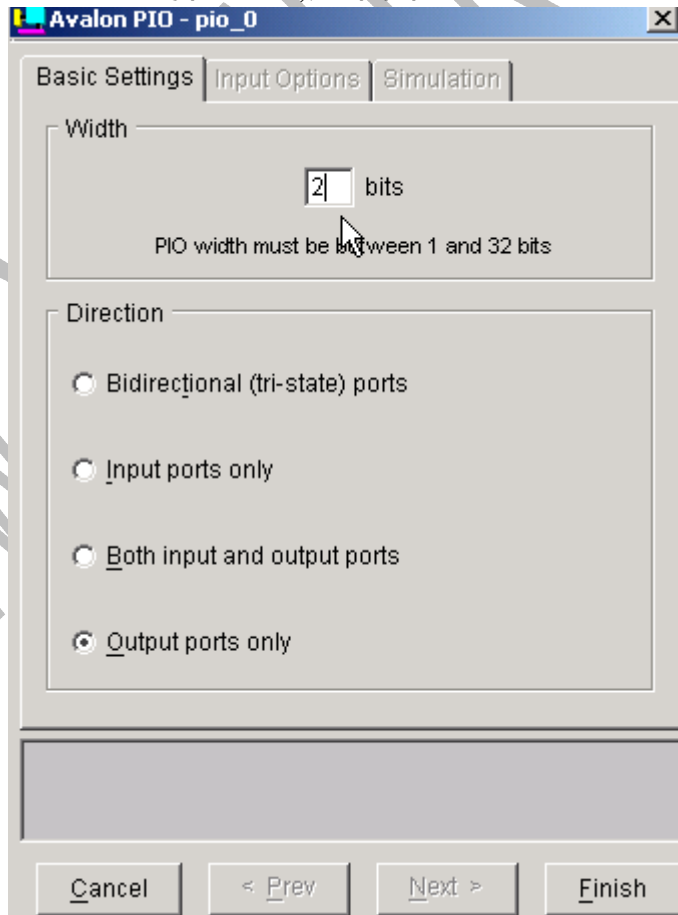
双击 On-Chip Memory(RAM or ROM),(在 Avalon Modules -> Memory -> 下), 为系统添加 RAM. Memory Type 选择 RAM; Data Width 选择 32bits, Total Memory Size 可以选择 4K bytes, 如下图:



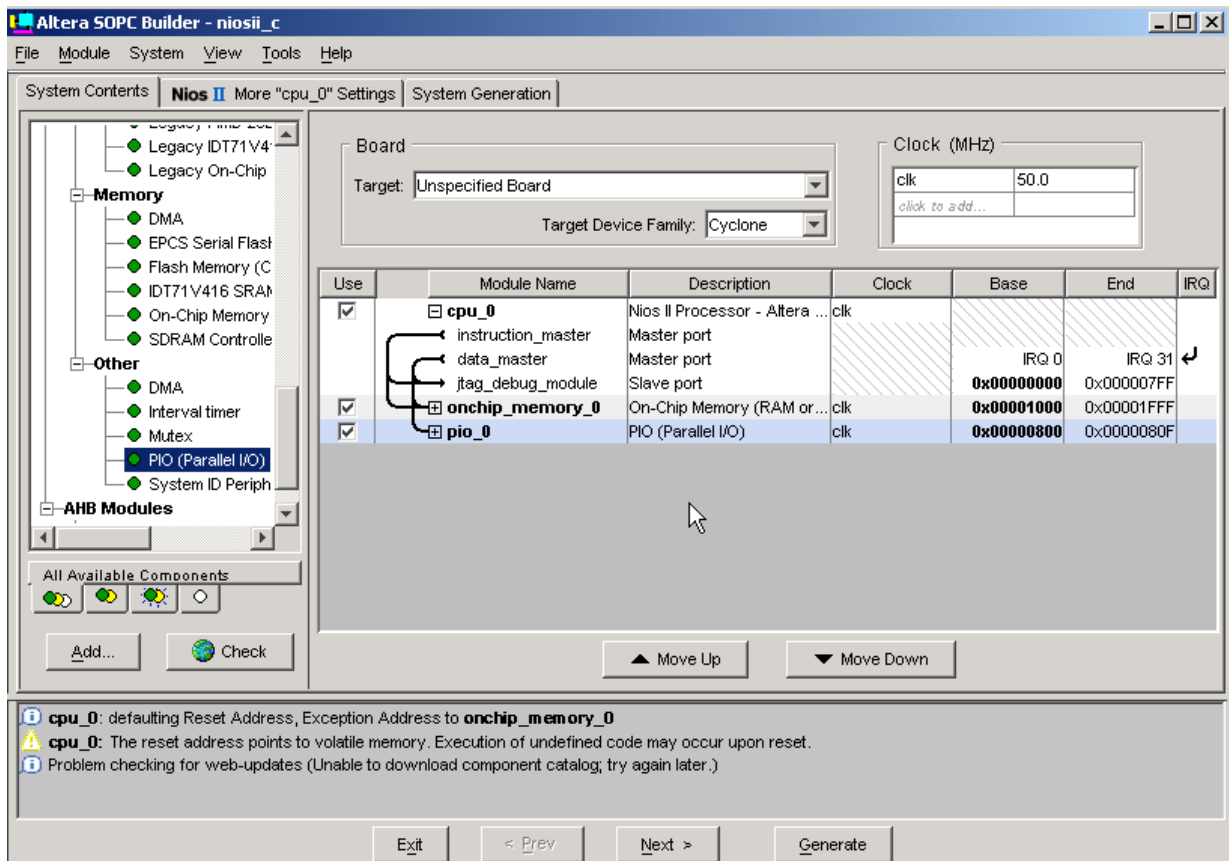
按 Finish 确认, 返回 SOPC Builder 界面:



双击 PIO(在 Parallel I/O)(在 Avalon Modules -> Other 下), 为系统添加输出到 LED 的输出接口。Width 选择 2 bits, 如下图:



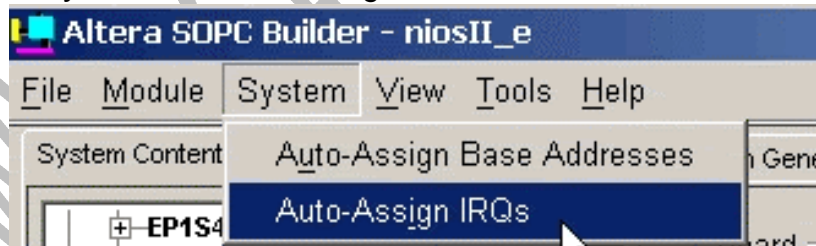
然后点击 Finish, 返回 SOPC Builder 界面:



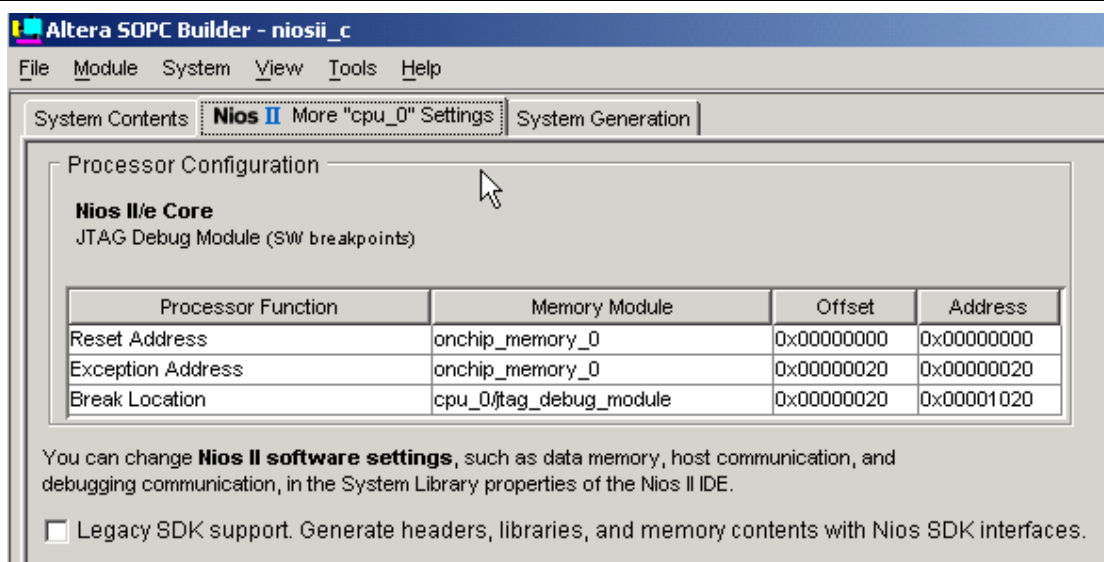
然后，选择 System ->Auto-Assign Base Addresses,让系统自动分配基地址。如下图：



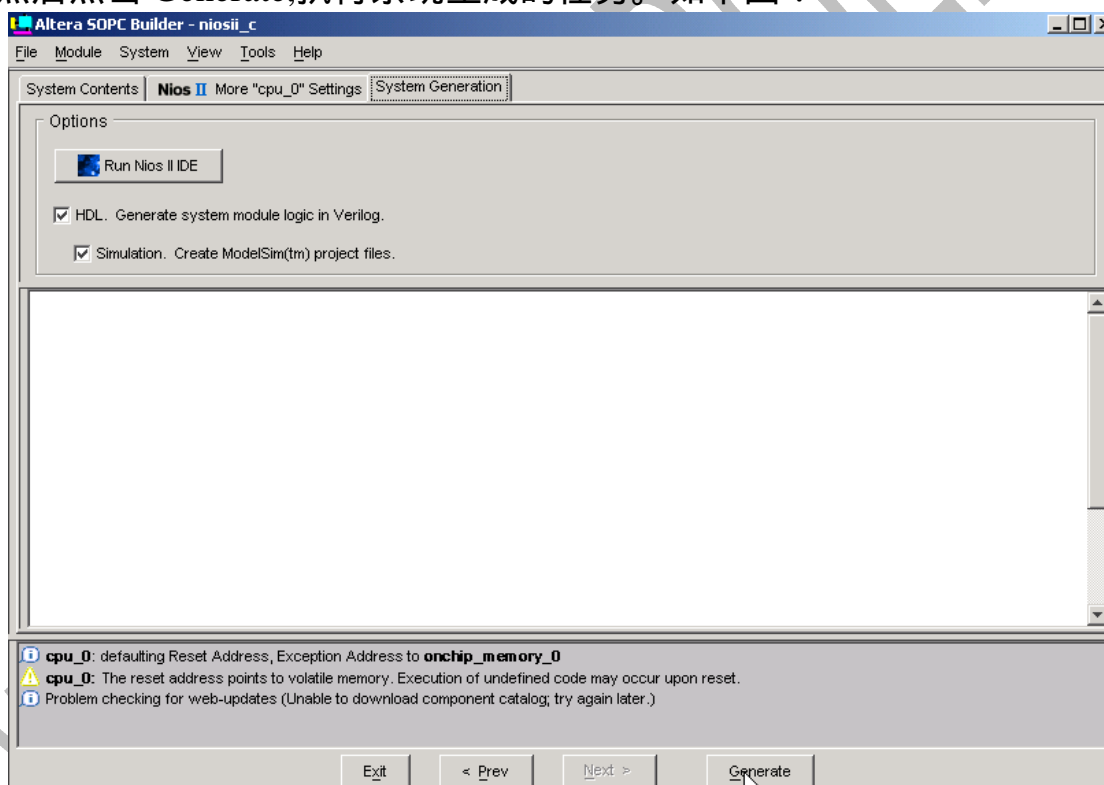
然后，选择 System->Auto-Assign IRQs,让系统自动分配中断。如下图：



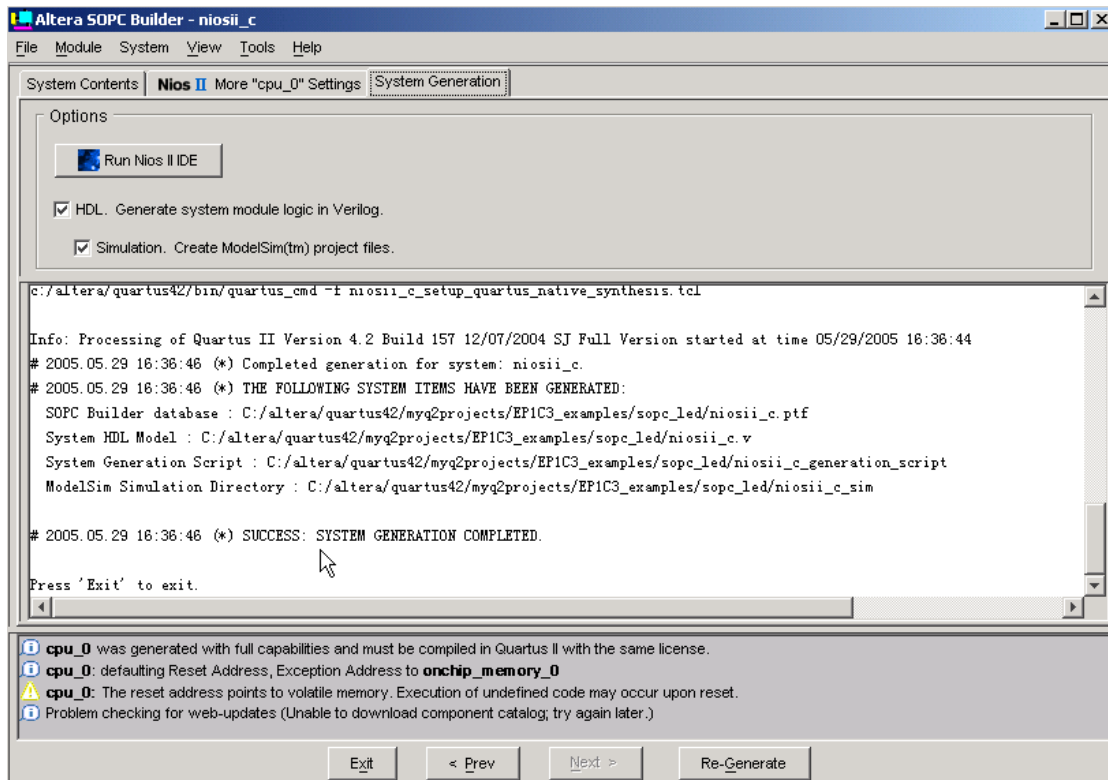
点击 Nios II More "cpu_0" settings 选项卡，进行处理器设定。在该例中，无需做任何更改；Reset Address、Exceptioning Address、Break Location 默认值如下图所示：



点击 System Generation 选项卡，进行最后的设定并生成系统。选中 HDL.Generate system module logic in Verilog, 如果需要仿真，也请选中 Simulation.Create ModelSim(tm) project files 然后点击 Generate,执行系统生成的任务。如下图：



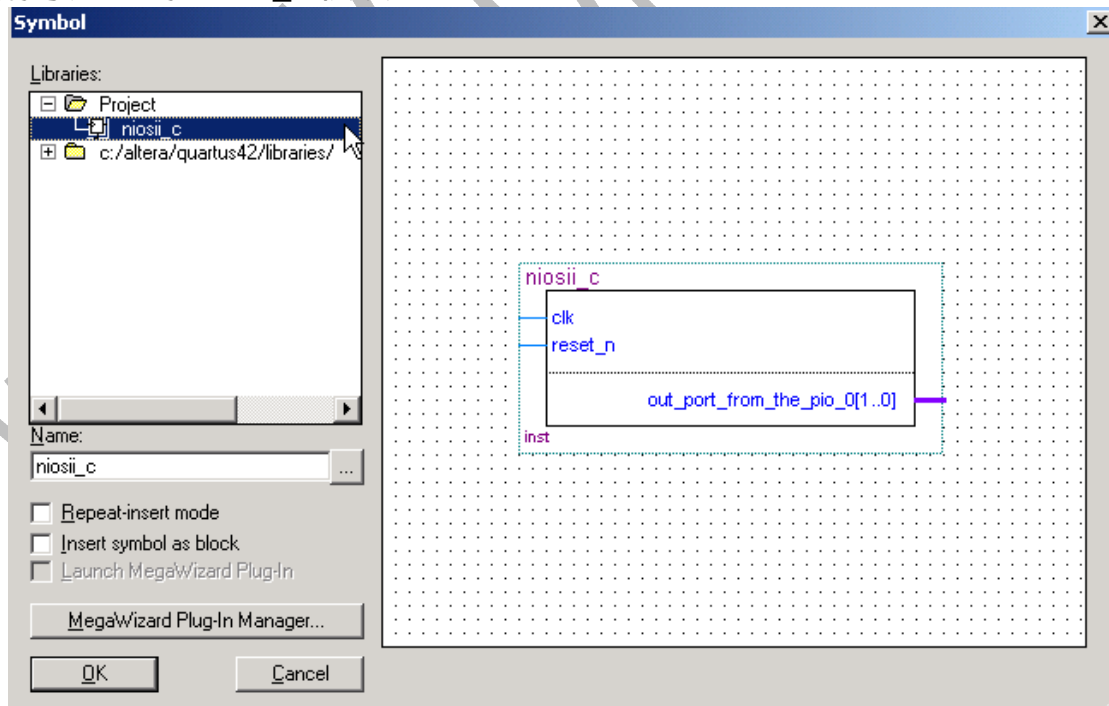
然后，耐心地等待系统的生成。一般没有问题的话，可以看到系统提示：SUCCESS: SYSTEM GENERATION COMPLETED. 如果看到此信息，恭喜恭喜，系统被正确生成了。如果失败，请返回并检查、修改。系统成功生成如下图所示：



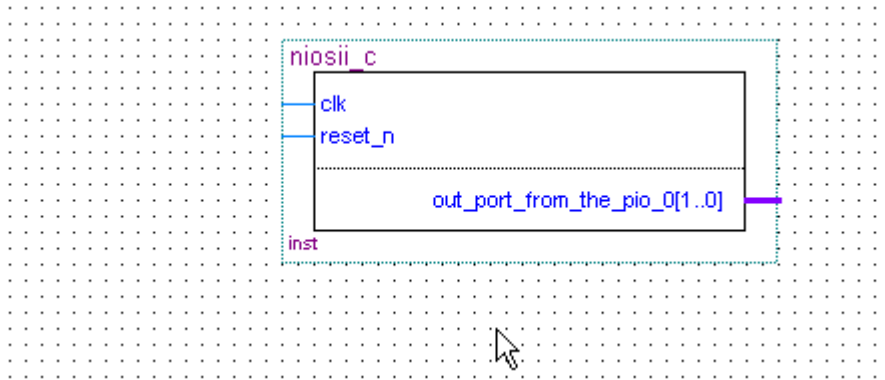
点击 Exit 退出 SOPC Builder。

4. 在 QuartusII 工程中添加上述 niosII 系统。

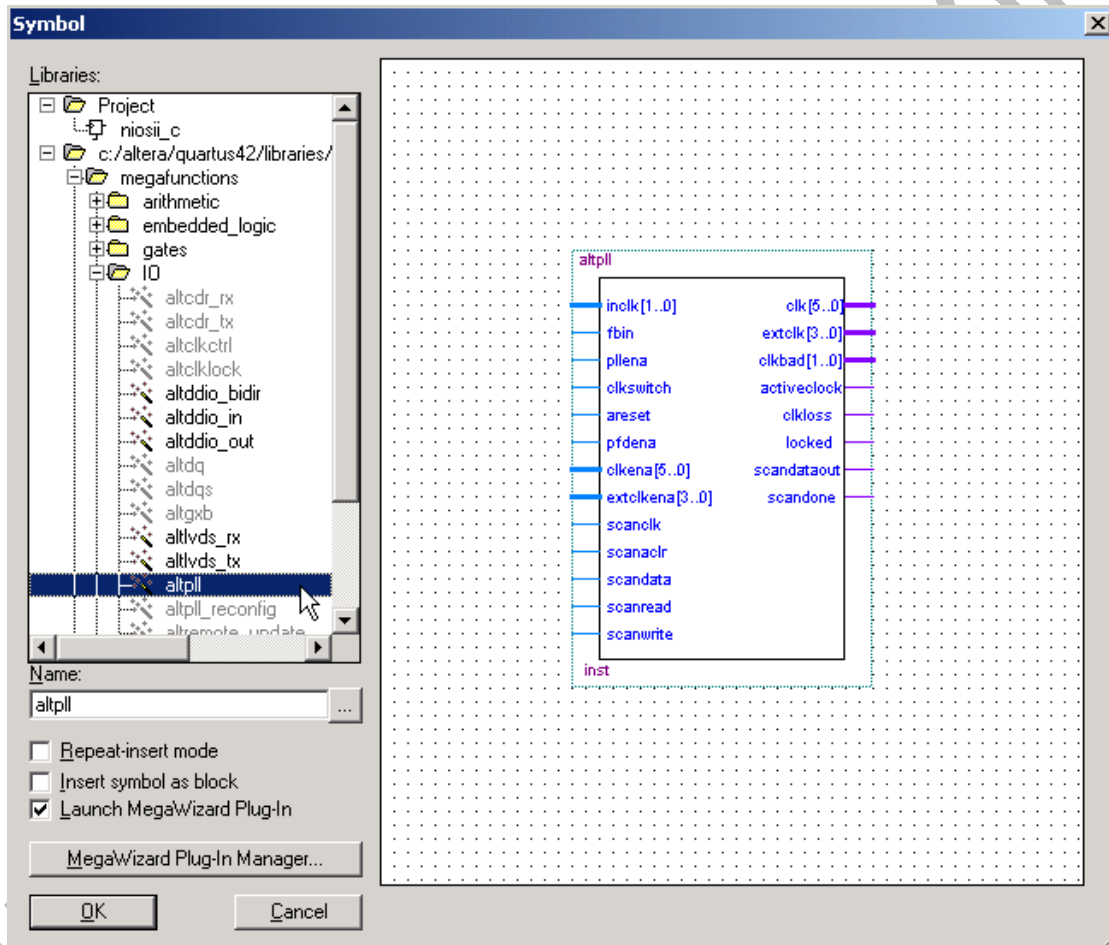
类似往 QuartusII 工程添加功能模块,将所定制的 NiosII 软核 CPU 添加到本例工程中,即双击工程顶层图空白处,弹出 Symbol 对话框,在 Project 下面可以找到刚才建立的 niosii_c 模块:



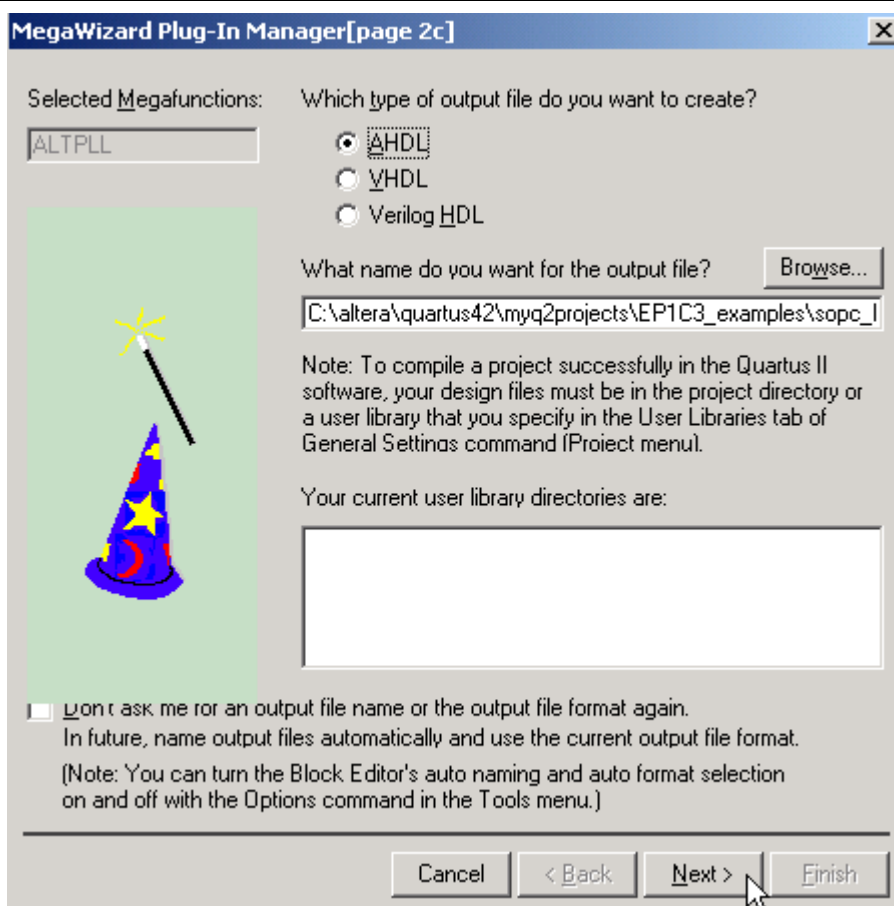
按 OK 将其 niosii_c 添加到顶层图中:



5. 添加 PLL 模块。从 QuartusII 的库中添加一个锁相环 (PLL) 模块，以将外部时钟倍频后输送给 CPU。打开 Symbol 对话框，在下图所示的目录下找到 altpll 模块。



按 OK 进入 altpll 模块的设置对话框：



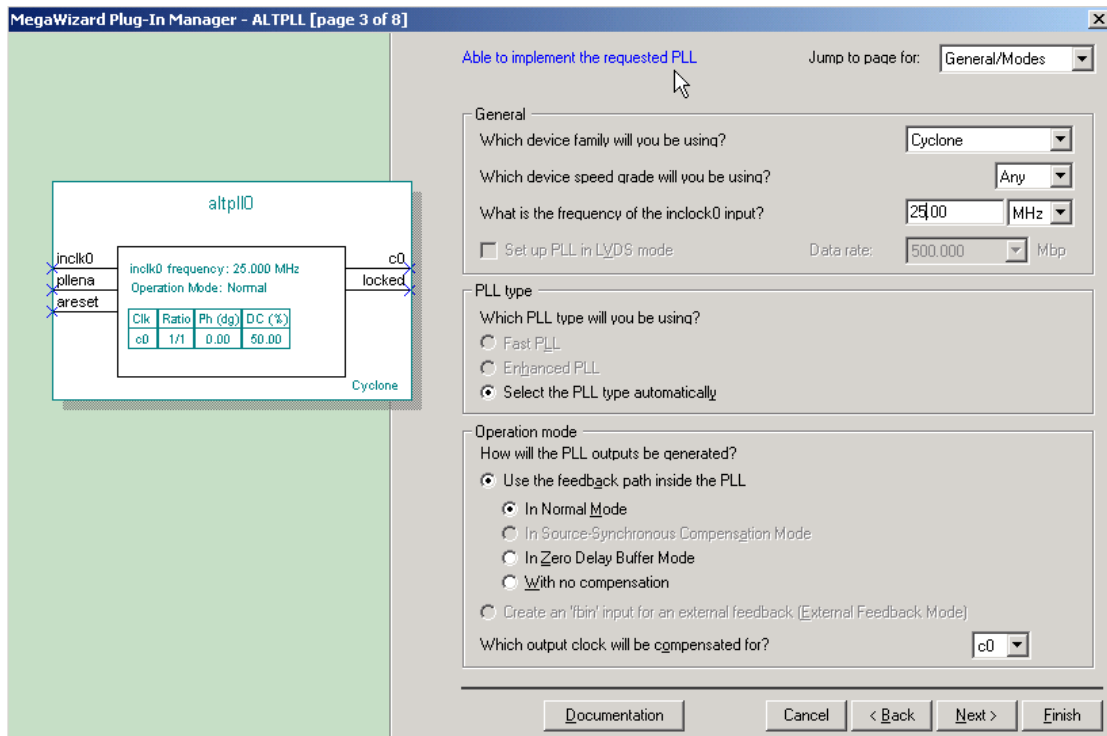
按 Next，分别设置下列内容：

“ Which Device family will you be using? ”： Cyclone ；

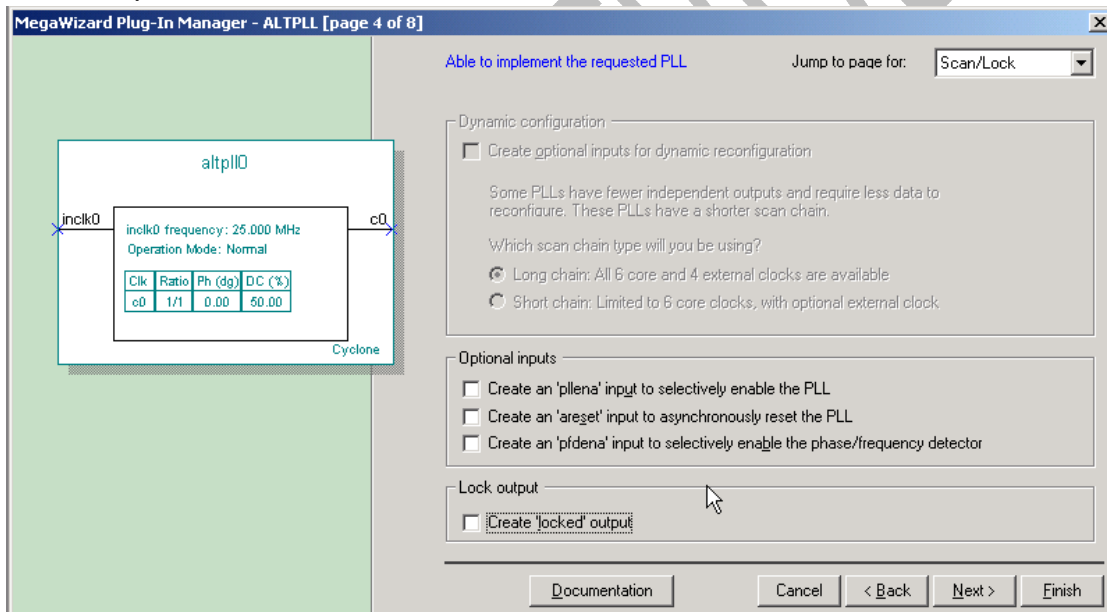
“ Which device speed grade will you be using ”： any ；

“ What is the frequency of the inclk0 input? ”： 25.00MHz ；（ 注意，该项内容最小必须填 16MHz 以上，否则下图鼠标所指的地方会提示错误信息；如果板上的有源时钟芯片的标称值小于该值，则在该栏填入大于 16MHz 的值，直到不会提示错误信息。）；

其它接受缺省设置，如下图所示：



按 Next 按钮，在进入的页面设置如下图：

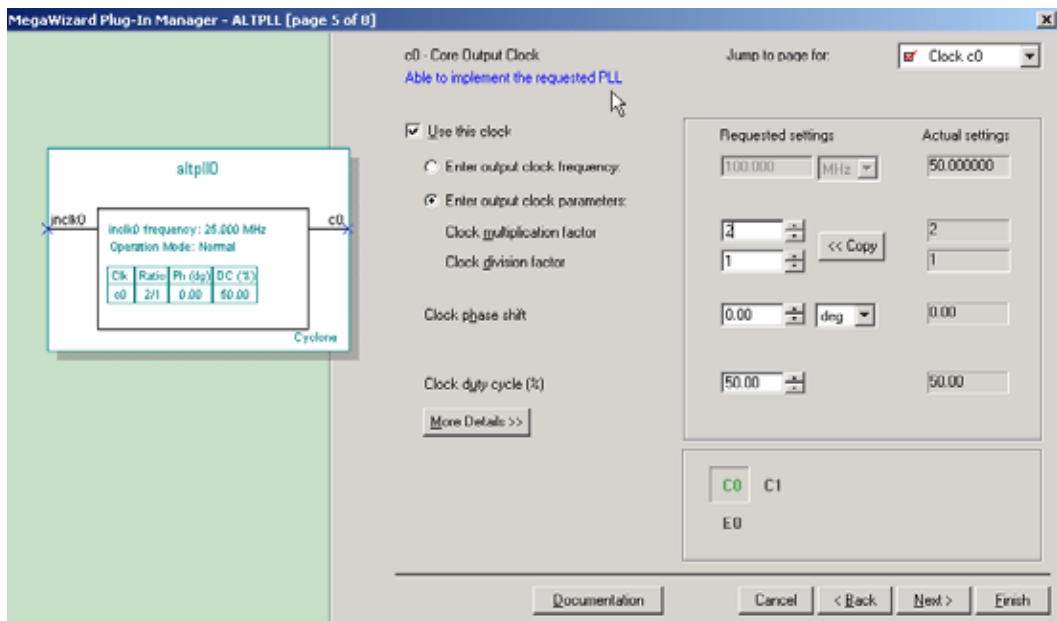


按 Next，进入 c0 输出设置界面：

选上“Use this clock”选项；

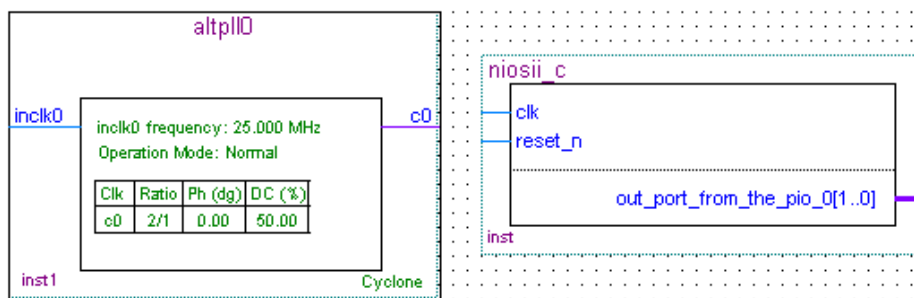
选择“Enter output clock parameters”；

在“Clock multiplication factor”与“Clock division”中填入合适的倍频因子和分频因子，倍频因子和分频因子的选择参照 c0 口所需要输出的频率而设置，例如本例中我们定义 CPU 的时钟为 50MHz，因此如果外部时钟的实际输入为 25MHz，则可以选择倍频因子为 2，分频因子为 1；如果实际输入时钟为 10MHz，则可以选择倍频因子为 5，分频因子为 1。注意：如果倍频因子和分频因子超出 altpll 所能实现的范围，则会提示错误。

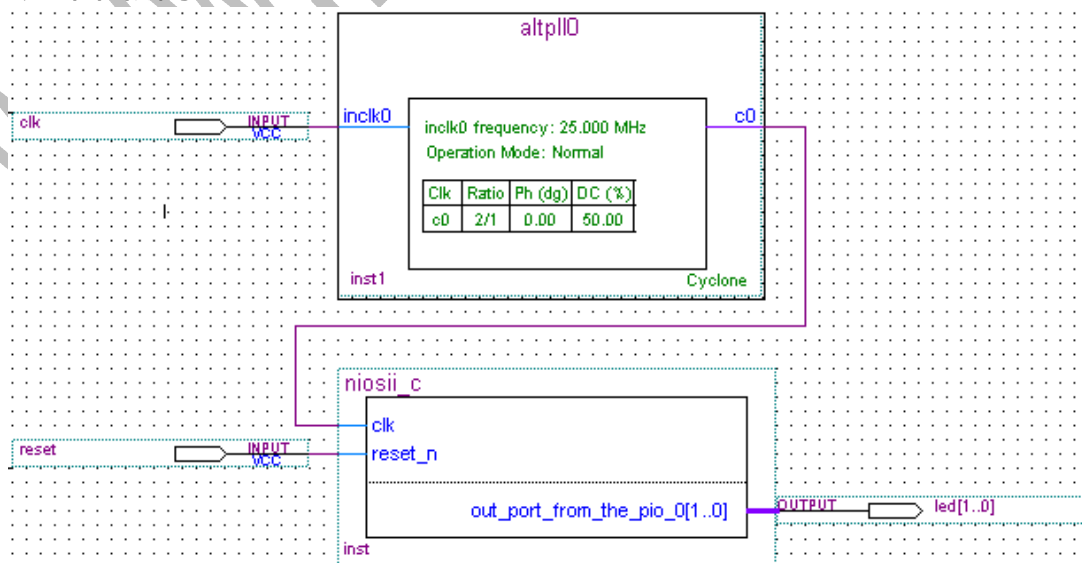


本例中我们没有用到 c1 和 e0 输出, 因此不用对它们进行配置, 直接按 Finish 完成 pll 的配置。如果需要用到它们的话则必须对它们进行配置, 过程类似于 c0 输出的配置。

我们按 Finish 返回到 QuartusII 的顶层图界面, 将建立好的 PLL 模块添加进去, 如图所示:



6. 添加其它元件模块。本例分别添加 clk 和 reset 输入、led[1..0]输出。并相互连接。如下图所示:



7. 设置编译选项。建立并运行 tcl 脚本语言定义管脚。

1 对于 EP1C3 核心板，tcl 管脚配置文件为（关于核心板引脚与实验板引脚对应的详细情况请参照“CT-SOPCx 学习套件用户手册”或相关电路原理图）：

```
#Setup.tcl
```

```
# Setup pin setting for EP1C3 main board
```

```
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT  
TRI-STATE"
```

```
set_global_assignment -name ENABLE_INIT_DONE_OUTPUT OFF
```

```
set_location_assignment PIN_16 -to clk
```

```
set_location_assignment PIN_54 -to reset
```

```
set_location_assignment PIN_6 -to led\[0\]
```

```
set_location_assignment PIN_7 -to led\[1\]
```

2 对于 EP1C6 核心板，tcl 管脚配置文件为：

```
#Setup.tcl
```

```
# Setup pin setting for EP1C6 main board
```

```
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT  
TRI-STATE"
```

```
set_global_assignment -name ENABLE_INIT_DONE_OUTPUT OFF
```

```
set_location_assignment PIN_28 -to clk
```

```
set_location_assignment PIN_159 -to reset
```

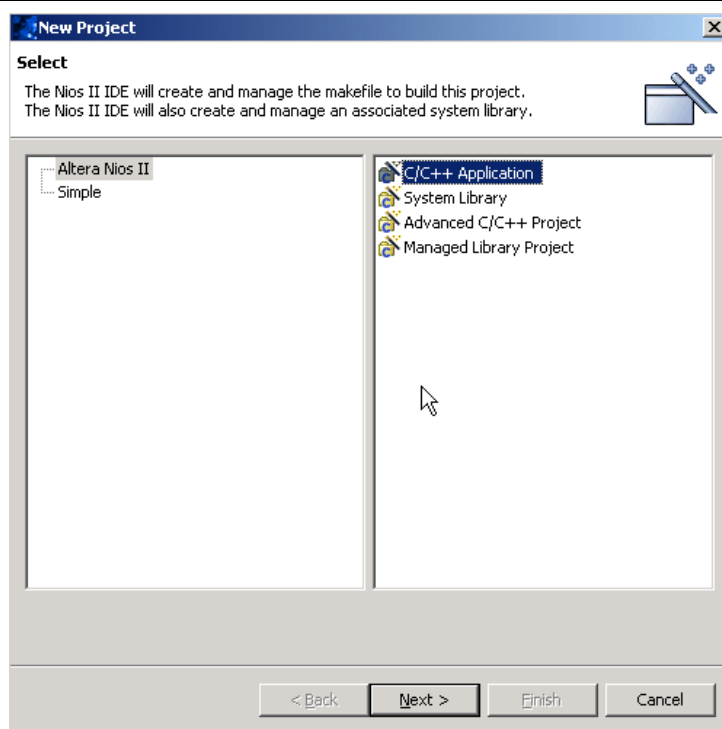
```
set_location_assignment PIN_1 -to led\[0\]
```

```
set_location_assignment PIN_2 -to led\[1\]
```

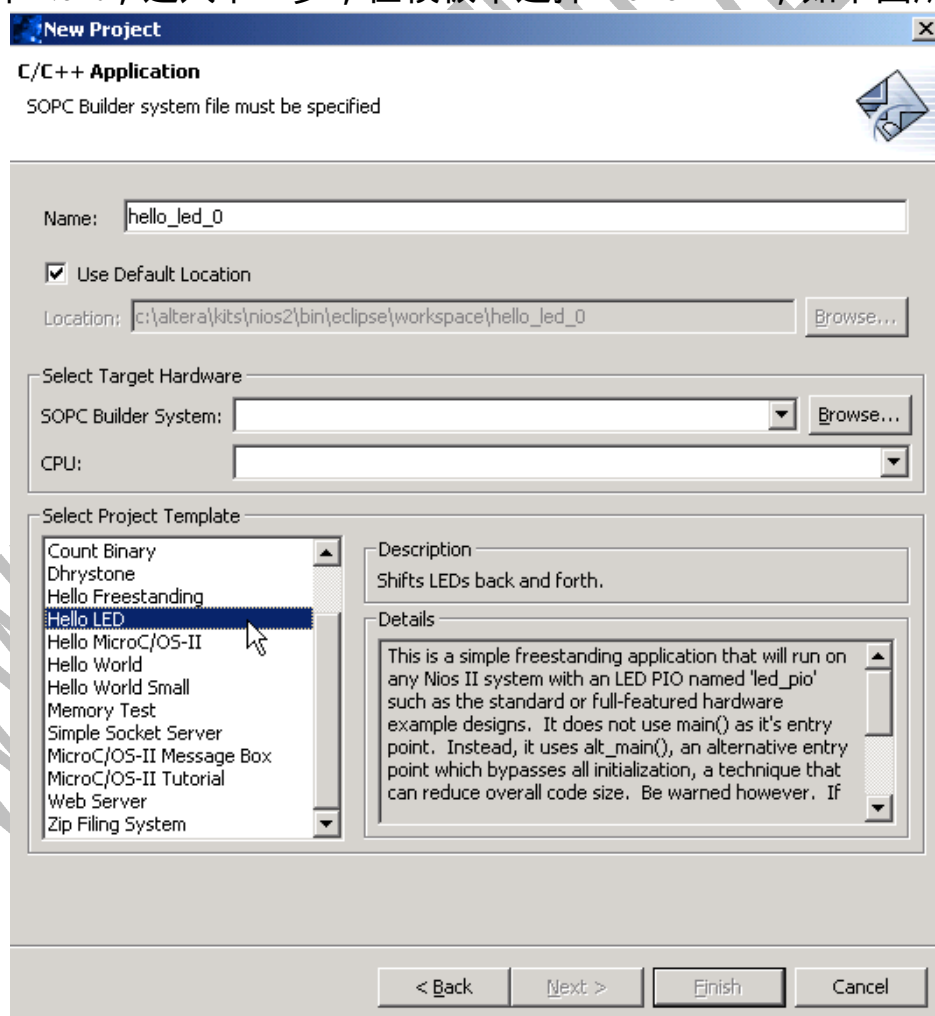
8. 编译该工程并通过Flash下载口下载到EPCS1 芯片中。注意是下载到EPCS1 芯片，目的是在FPGA芯片中建立硬件系统，为后来的软件调试做好准备。

9. 设计该工程的软件。

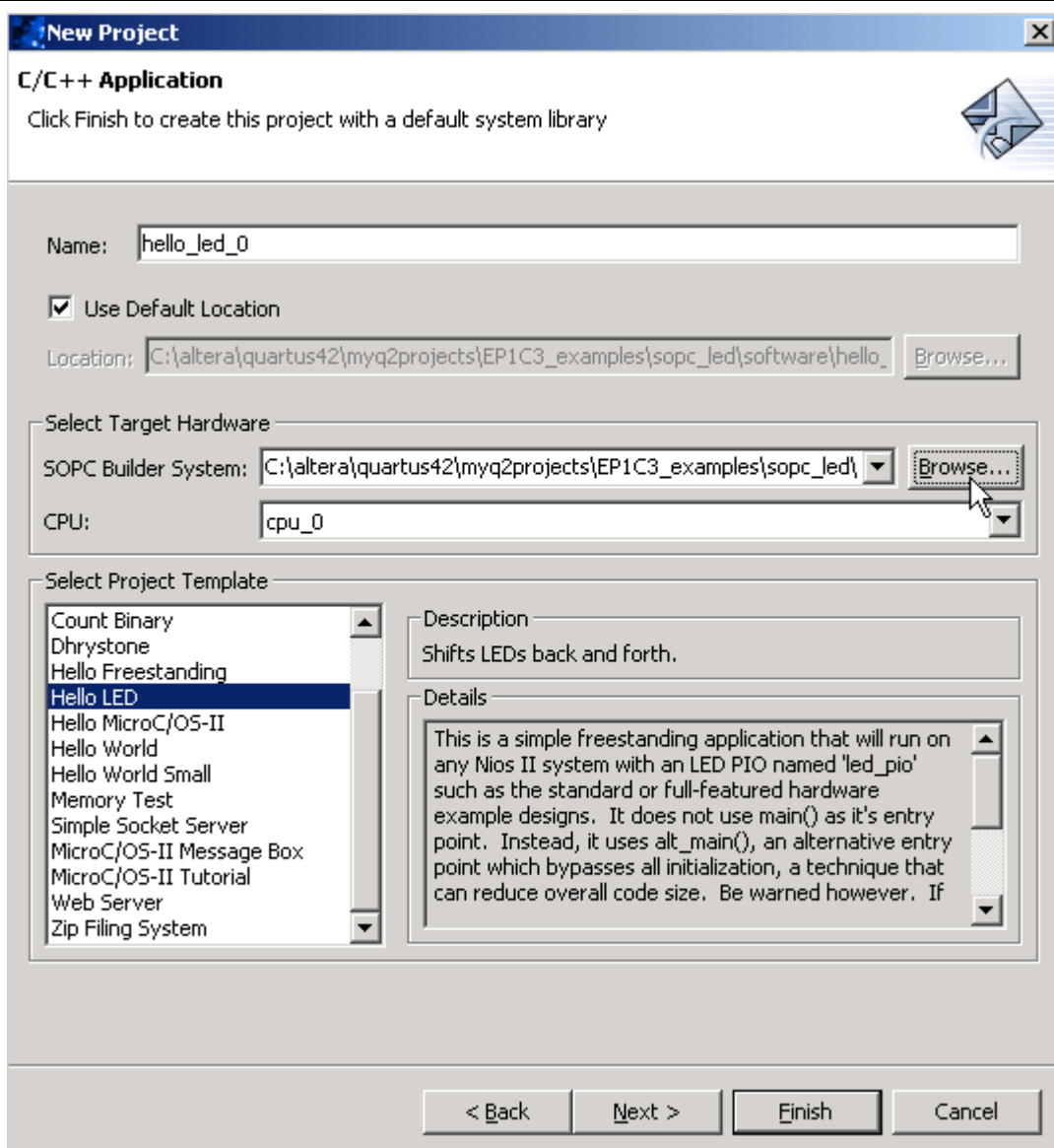
接下来我们要设计该工程的软件。运行 Nios IDE，在 Nios II IDE 中，选择 File -> New -> Project，开启 New Project 对话框，选择 C/C++ Application，如下图所示：



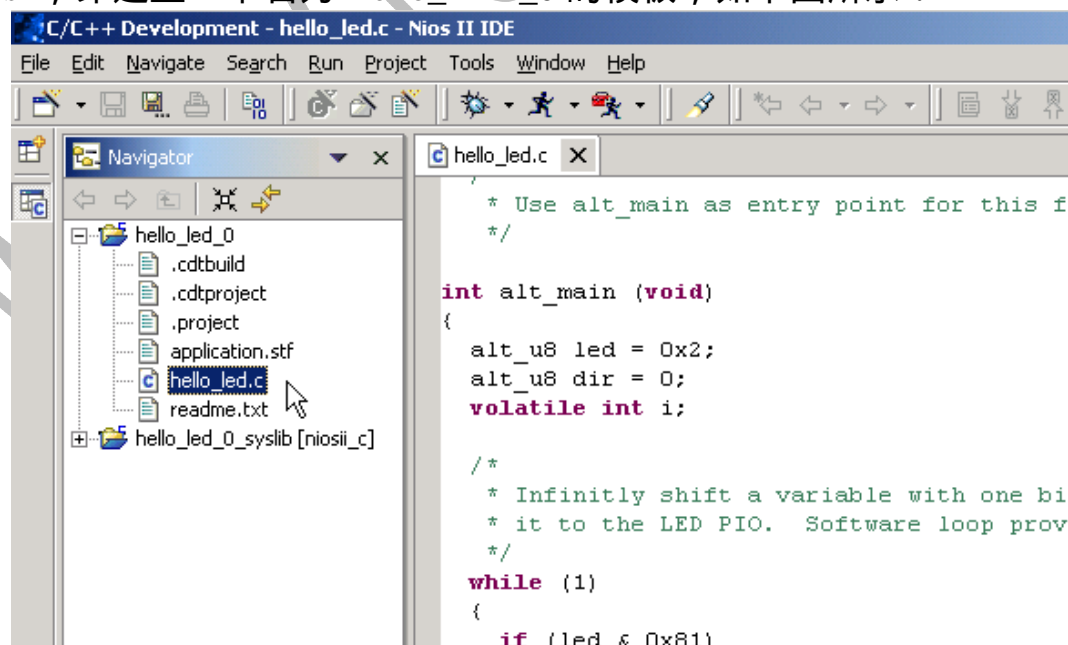
然后选择 Next，进入下一步，在模板中选择 Hello LED，如下图所示：



点击“SOPC Builder System”右边的“Browse”按钮打开目录对话框，从该工程目录下找到“niosii_c.ptf”，因为 NiosII IDE 必须从这个文件获取该系统的相关信息。打开该文件后，界面显示如下图：



按 Finish，即建立一个名为 Hello_LED_0 的模板，如下图所示：

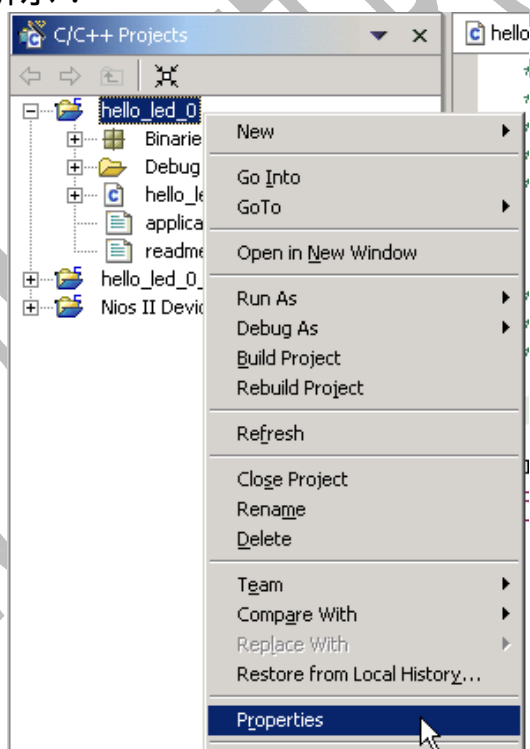


打开项目中的文件：hello_led.c，根据需要进行修改，在本例中，我们模板的程序改写为以下内容：

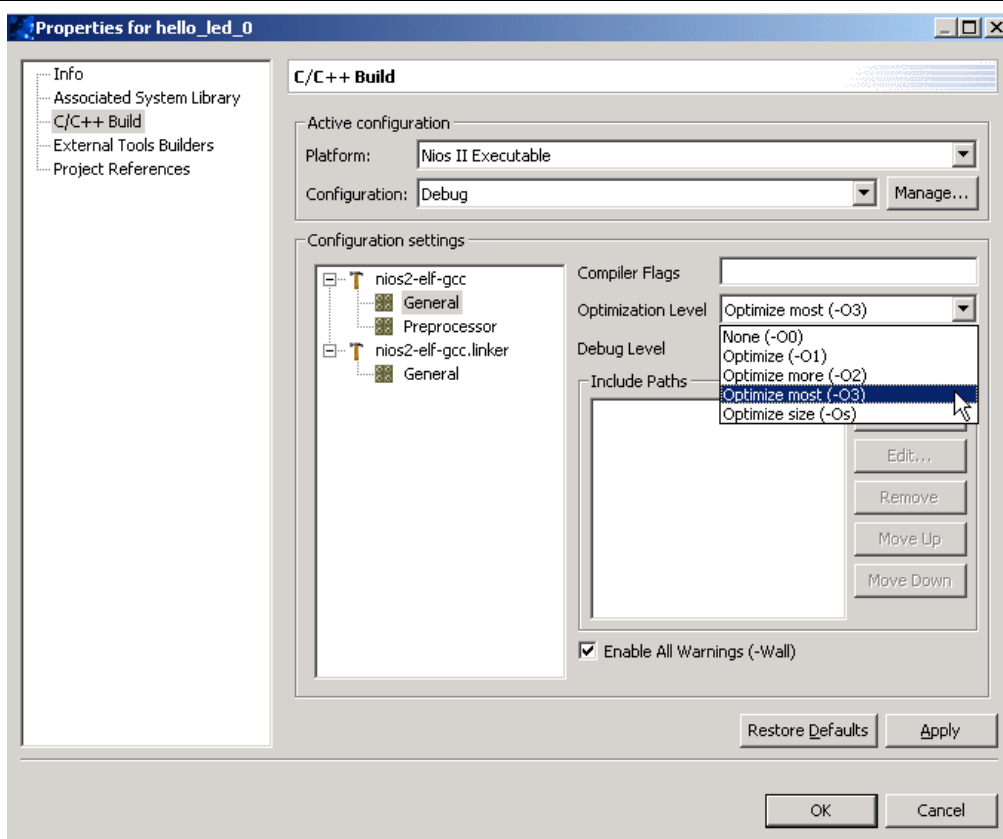
```
int alt_main (void)
{
    alt_u8 led = 0x2;
    volatile int i;
    while (1)
    {
        for(i=0;i<300000;i++) ;
        led = 0x1;
        *(unsigned int *)PIO_0_BASE = led;
        for(i=0;i<300000;i++) ;
        led = 0x2;
        *(unsigned int *)PIO_0_BASE = led;
    }
    return 0;
}
```

10. 编译设置。

编译之前我们先对项目进行一些设置，以使编译器编译出更高效、占用空间更小的代码。右键点击 `hello_led_0` 工程名称，在弹出的菜单中选择“Properties”，如下图所示：

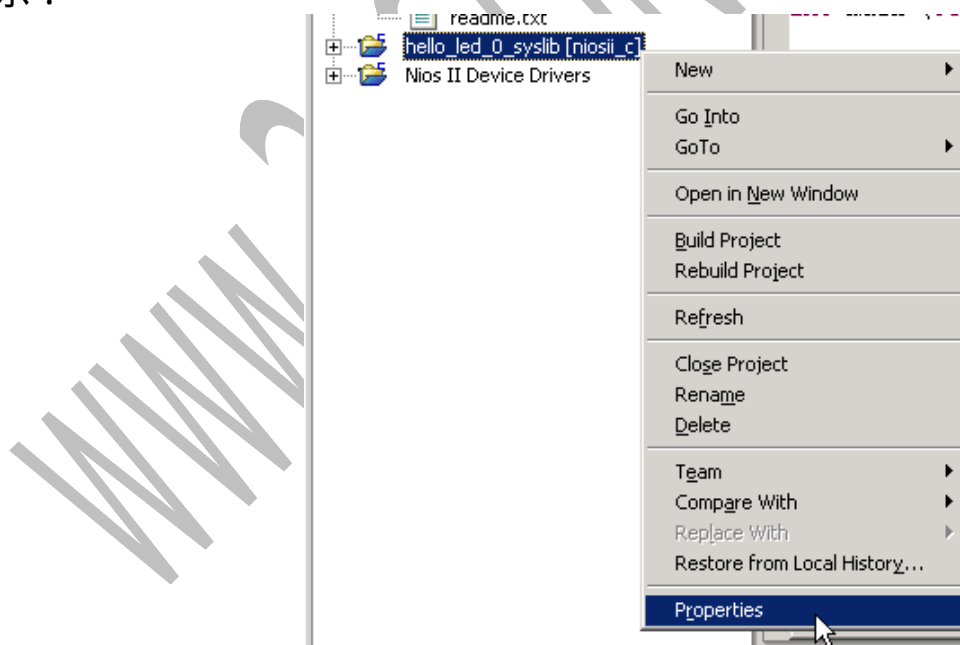


点击后打开工程属性 (Properties for hello_led_0) 对话框，在“Configuration Settings”点击“General”页面，选择“”在“Optimization Level”中选择“Optimize size (-Os)”，如下图：

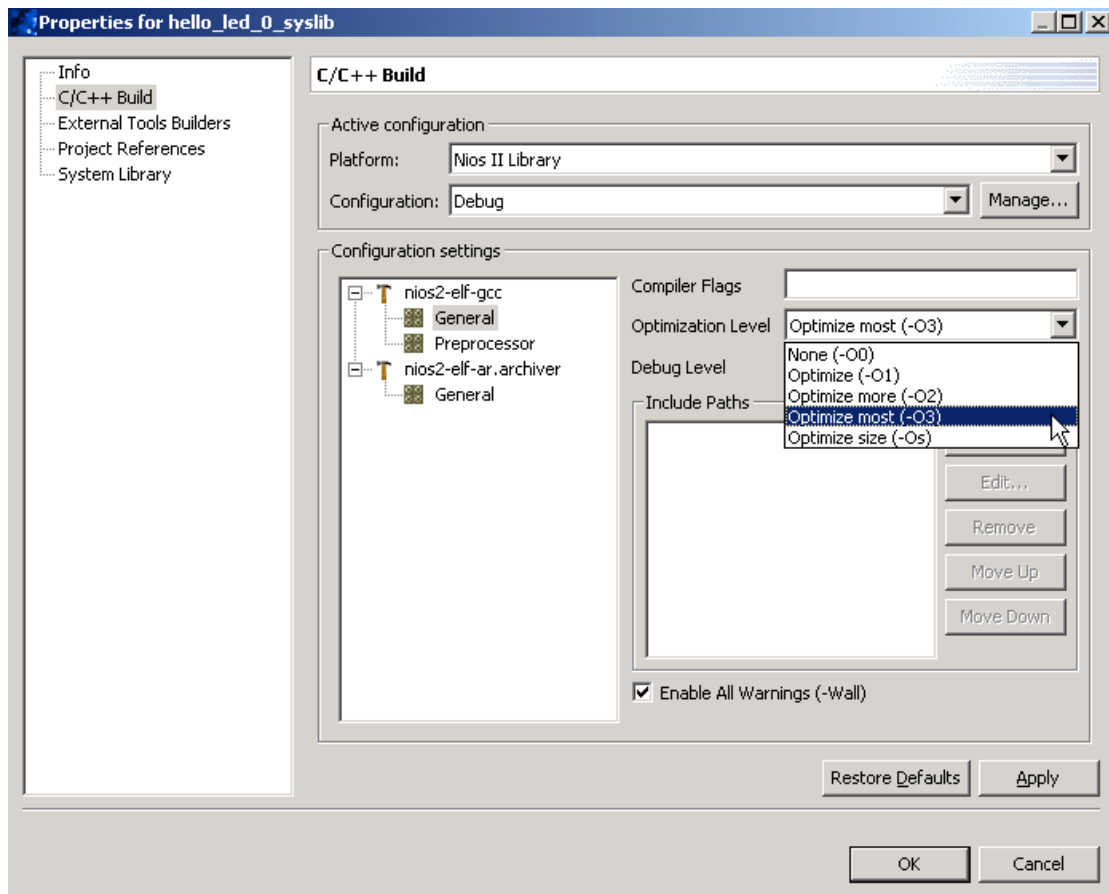


点击 OK 退出对 hello_led_0 工程属性的设置。

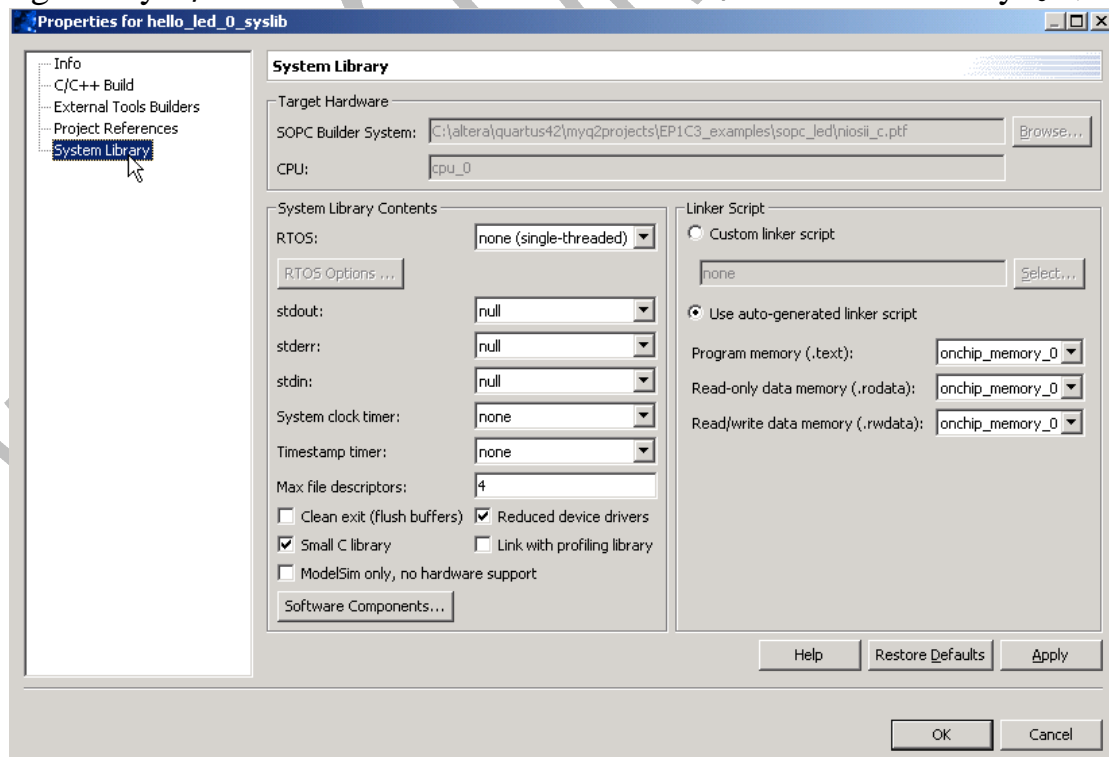
然后，再设置 hello_led_0_syslib[niosii_c] 工程。右键点击 hello_led_0_syslib[niosii_c] 工程名称，在弹出的菜单中选择“Properties”，如下图所示：



点击后打开工程属性（Properties for hello_led_0_syslib）对话框，在“Configuration Settings”点击“General”页面，在“Optimization Level”中选择“Optimize size (-Os)”，如下图：



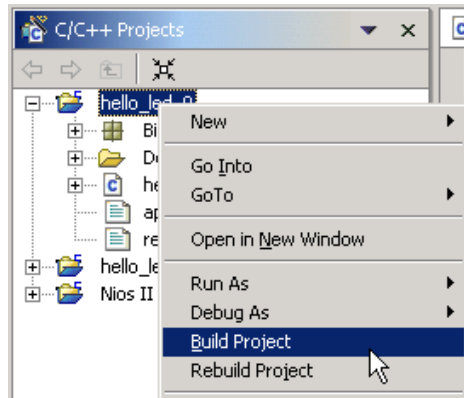
然后，点击该对话框的“System Library”打开 System library 属性页面，在“Max file descriptors:”栏改为 4，清除“Clean exit (flush buffers)”和“Link with profiling library”，选上“Reduced device drivers”和“Small C library”。如下图：



点击OK完成设置。以上设置主要目的是为了优化程序，并减少程序占用内存空间。

11 . 编译。右键点击“hello_led_0”，在弹出的菜单中选择“Build Project”。（为

使编译过程更加顺利，此时最好关闭杀毒软件和其它占用电脑资源较大的软件。)

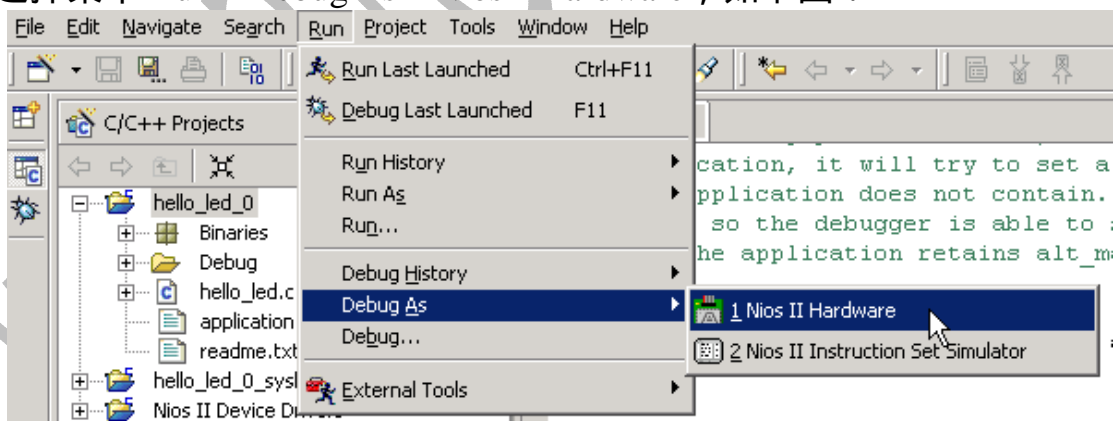


如果无误，可以看到以下信息：Build completed.如果看到了该条提示，表示软件编译成功，可以看到程序占用空间等信息，例如本例子程序占用 760 Bytes 内存空间，剩余 3336 Bytes（我们建立 niosii_c 系统的时候定义的是 4K 内存）。

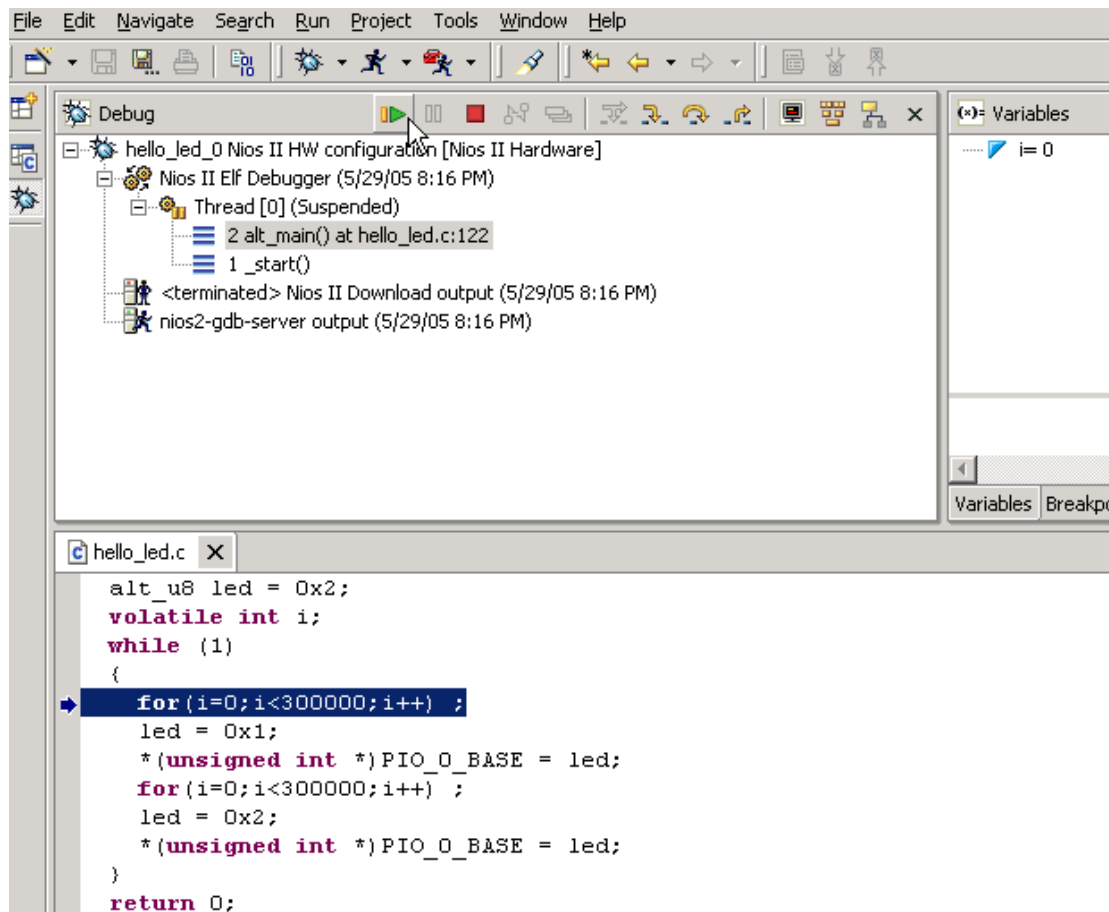
```




C-Build [hello_led_0]
Compiling hello_led.c...
Linking hello_led_0.elf...
Info: (hello_led_0.elf) 760 Bytes program size (code + initialized data).
Info:          3336 Bytes free for stack + heap.
Post-processing to create onchip_memory_0.dat
Post-processing to create onchip_memory_0.hex
WARNING: Default charset GBK not supported, using ISO-8859-1 instead
Post-processing to create onchip_memory_0.sym
Build completed
  
```

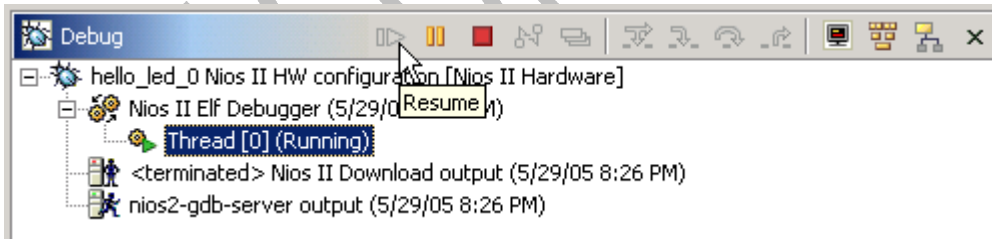
12. 调试。将 ByteBlasterII 下载线接到 FPGA 核心板的 JTAG 口，接上 5V 电源。选择菜单 Run→Debug As→NiosII Hardware，如下图：





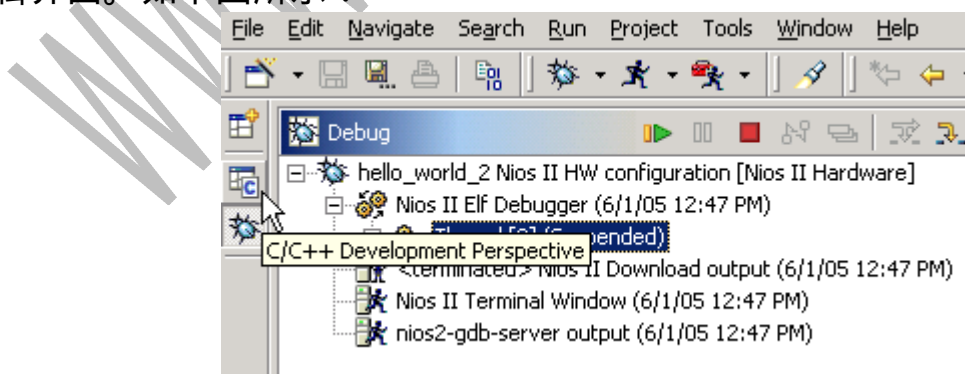
NiosII IDE 会打开调试界面（Debug Perspective），并在程序中设置断点，运行停止在断点处（下图的箭头处），如下图所示：



点击继续执行 (Resume) 按钮 ，程序就会继续运行，此时可以看到核心板上的两个 led 灯交替闪烁。点击  按钮，程序暂停运行，点击  按钮退出运行，如下图所示：



如需对程序进行修改，点击  按钮退出运行，然后点击  按钮回到 C/C++ 编辑界面。如下图所示：

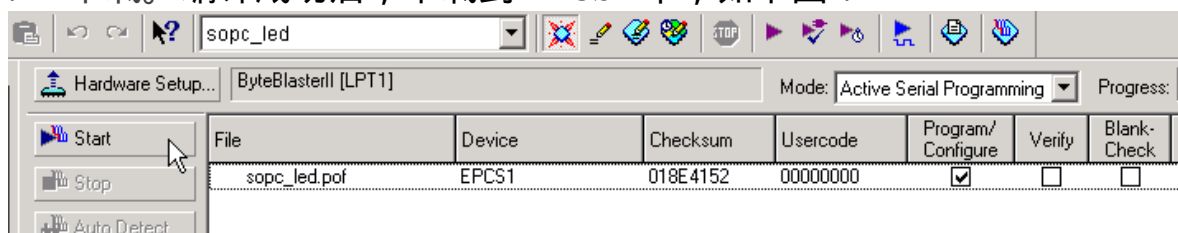


可返回程序编辑界面修改代码，然后后再重复上述调试过程。

13. 编译整个项目。软件设计完成并调试成功后，可以在 QuartusII 中将整个项目进行编译，将硬件配置信息和软件初始化信息编译在一起，并固化到 EPCS1

芯片中。在 QuartusII 中,选择 Processing -> Start Compilation,开始编译整个项目。

14. 下载。编译成功后,下载到 EPCS1 中,如下图:



15. 结果。下载完毕可以看到核心板上的两个 LED 灯交替闪烁。按实验板上的 RESET 键复位程序。

(二) 实验二: LED、数码管、LCD 和串口

通过实验(一)的学习,同学们已经可以基本掌握基于 QuartusII 和 NiosII IDE 的 SOPC 开发流程。学习贵在实践,对于嵌入式系统设计更是如此,因此,如果同学们想在这个领域有所作为,请多参加实践。利用我们提供的 CT-SOPCx 系列 FPGA/SOPC 开发套件,同学们可以自行设计有趣的实验,也可以自己制作实验板,插到我们提供的 FPGA 核心板上,实现更丰富有趣的功能。同时,我们还提供了比实验一稍复杂的实验例子,同时控制核心板上的 LED、实验板上的数码管、按键开关、LCD 和串口,实现的功能是:核心板上的两个 LED 等交替闪烁,实验板上的 KEY1 按键按下,数码管显示值减 1,KEY2 按下,显示值加 1;同时 LCD 显示两行字符,串口不断往 PC 机发生一行字符。限于篇幅,我们在此没有写出该实验的详细过程。下面列出该实验所定制的 NiosII 的处理器以及外设的一些配置。在以上实验的基础上,我们相信同学们参照我们随光碟给出的例子代码,很快就可以理解该例子的设计方法。

说明:该例子程序可以在学习套件附带光碟的“学习套件例子”找到。对应于 EP1C6 型号和 EP1C3 型号的核心板,该例子的所有文件分别保存在“EP1C6_Examples”或“EP1C3_Examples”的“all_test”子目录中。

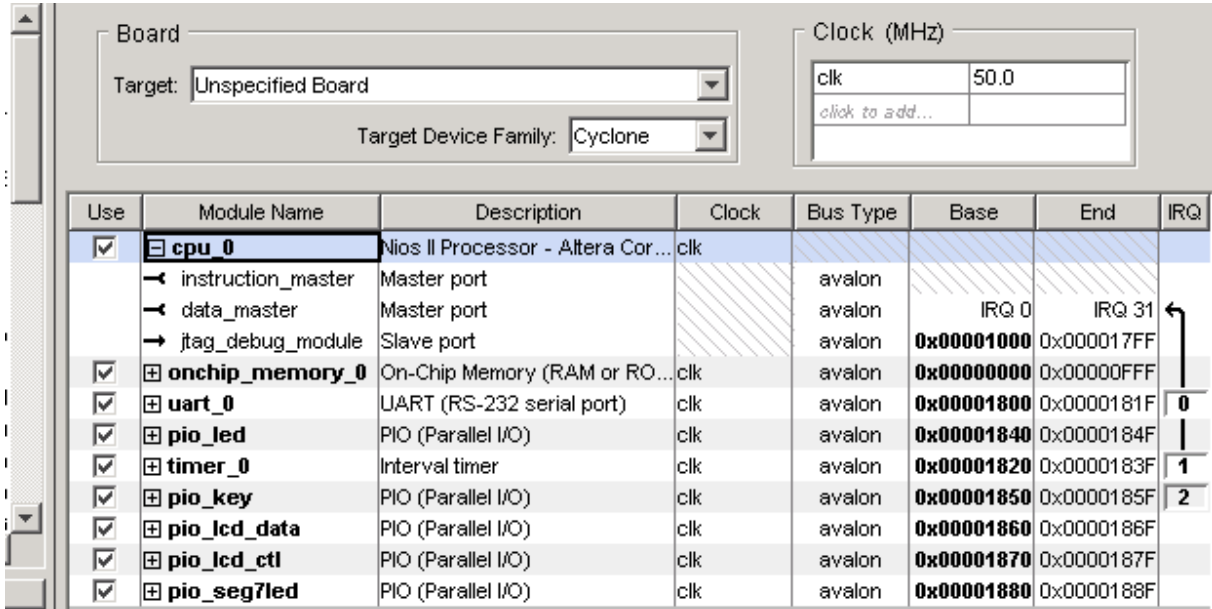
该例子分别定制了:

1. 32 位的 NiosII 处理器“CPU0”:简约型处理器“NiosII/e”,一级调试支持“Level1”;
2. 片上存储器“onchip_memory_0”:32 位宽,4Kbyte;
3. 串口“uart_0”:波特率设为 9600,无校验位,8 位数据位,1 位停止位;
4. 控制 LED 灯的 IO“pio_led”:2 位,“Output ports only”;
5. 定时器“Timer0”;
6. 作为按键输入的 IO“pio_key”:2bits,“Input ports only”;
7. LCD 液晶的数据线“pio_led_data”:8bits,“Bidirection (tri-state) ports”;
8. LCD 液晶的控制线“pio_led_ctl”:3bits,“Output ports only”;
9. 控制 7 段数码管的 IO“pio_seg7led”:8bits,“Output ports only”。

其中,LCD 液晶的接口我们没有采用 SOPC Builder 已有的 Character LCD (16X2 Optrex 15207) 接口,因为市面上很难买得到这个型号的 LCD,而 1602A 型号的 LCD 比较常用,所以我们自己定义了它的接口,即定义 8 位的

三态 IO 作为 LCD 的数据总线，而另外定义 3 个输出 IO 作为 LCD 的三个控制信号输出 (lcd_e, lcd_rs, lcd_rw)，在实际项目设计中，大家也可以自己定义其它方式，或者修改 Altera 提供的 Character LCD (16X2 Optrex 15207) 的驱动，实现 1602A 型号 LCD 的驱动。

下图为本例子所定制的 Nios 系统的配置情况：



Use	Module Name	Description	Clock	Bus Type	Base	End	IRQ
<input checked="" type="checkbox"/>	cpu_0	Nios II Processor - Altera Cor...	clk				
	← instruction_master	Master port		avalon			
	← data_master	Master port		avalon			
	→ jtag_debug_module	Slave port		avalon			
<input checked="" type="checkbox"/>	onchip_memory_0	On-Chip Memory (RAM or RO...	clk	avalon	0x00000000	0x00000FFF	
<input checked="" type="checkbox"/>	uart_0	UART (RS-232 serial port)	clk	avalon	0x00001800	0x0000181F	0
<input checked="" type="checkbox"/>	pio_led	PIO (Parallel I/O)	clk	avalon	0x00001840	0x0000184F	1
<input checked="" type="checkbox"/>	timer_0	Interval timer	clk	avalon	0x00001820	0x0000183F	2
<input checked="" type="checkbox"/>	pio_key	PIO (Parallel I/O)	clk	avalon	0x00001850	0x0000185F	
<input checked="" type="checkbox"/>	pio_lcd_data	PIO (Parallel I/O)	clk	avalon	0x00001860	0x0000186F	
<input checked="" type="checkbox"/>	pio_lcd_ctl	PIO (Parallel I/O)	clk	avalon	0x00001870	0x0000187F	
<input checked="" type="checkbox"/>	pio_seg7led	PIO (Parallel I/O)	clk	avalon	0x00001880	0x0000188F	

说明：该例子程序的串口程序的调试可利用随光碟的串口调试工具来调试，对于实验板 V1.0，该学习套件出厂的时候已经将该例子程序灌在 EPCS1 中，因此连上串口线，打开串口调试工具并正确设置 COM 信息（波特率设为 9600，无校验位，8 位数据位，1 位停止位），即可收到学习套件不断发送过来的字符串。

（三） 实验三——关于实验板 V2.0 版本增加的 USB 功能的例子

实验板 V2.0 在 V1.0 版本的基础上，新增加了 USB 至 UART 桥接器 CP2102，串行 E²PROM AT24C32（焊接位，EP1C3 核心板对应的实验板不含该功能），4 Pins IO 引出插座，用户可配置为三线制 SPI 通信口或作普通 IO 用（EP1C3 核心板对应的实验板不含该功能）。

其中串行 E²PROM 常用于控制系统中保存系统的状态参数以及用户设定参数等数据；4 Pins IO 引出插座，用户可配置为三线制 SPI 通信口或作普通 IO 用。

USB 至 UART 桥接器的设计添加，更突显了 SOPC 系统的精简实用精神。我们认为，用 Cyclone 组建的片上系统，更适用于需要大量的逻辑电路、程序代码不很复杂的系统中（特别是控制系统、仪器仪表）。这类系统通常需要与上位机进行通讯，例如 RS232 串口、RS485、USB 等。虽然 RS232 之类的通讯在工控、仪器仪表系统中还占据主导地位，但由于当今 USB 接口的流行，很多设备也慢慢趋向于有必要增添 USB 接口的功能。因此，实验板 V2.0 增添了 USB 接口的功能。

实验板 V2.0 采用 USB 至 UART 桥接器设计，一方面使得 USB 接口设

计比较简单,开发者无需懂得 USB 协议和设计驱动程序,只需类似串口一样对通讯进行操作即可,因而使得开发周期短,风险小,成本低;另一方面,体积小,工作稳定可靠,正常工作温度范围为: $-40 \sim +85$, 特别适合工业控制或仪器仪表使用。该芯片的通讯速度可达 1Mbits/s,兼容 USB1.1 和 USB2.0,满足绝大多数工业控制或仪器仪表的数据传输需求。

随学习套件的光碟中,附带了该芯片的数据手册、驱动程序,以及例子程序 (“\EP1C6_Examples\all_test_USB”)。

1. 安装 USB 驱动

驱动的安装方法请参照《CT-SOPCx 系列 FPGA/SOPC 学习套件用户手册》。

2. 添加功能模块

在 SOPC 系统中添加一个 UART 功能模块,例如我们在本节的实验二的例子的基础上添加一个“uart_usb”的 UART 功能模块,波特率设为 115200,无校验位,8 位数据位,1 位停止位;如下图:

Use	Module Name	Description	Clock	Bus Type	Base	End	IRQ
<input checked="" type="checkbox"/>	<input type="checkbox"/> cpu_0	Nios II Processor - Alt...	clk				
	<input type="checkbox"/> instruction_master	Master port		avalon			
	<input type="checkbox"/> data_master	Master port		avalon		IRQ 0	IRQ 31
	<input type="checkbox"/> jtag_debug_module	Slave port		avalon	0x00001000	0x000017FF	
<input checked="" type="checkbox"/>	<input type="checkbox"/> onchip_memory_0	On-Chip Memory (RA...	clk	avalon	0x00000000	0x00000FFF	
<input checked="" type="checkbox"/>	<input type="checkbox"/> uart_0	UART (RS-232 serial ...	clk	avalon	0x00001800	0x0000181F	0
<input checked="" type="checkbox"/>	<input type="checkbox"/> pio_led	PIO (Parallel I/O)	clk	avalon	0x00001860	0x0000186F	1
<input checked="" type="checkbox"/>	<input type="checkbox"/> timer_0	Interval timer	clk	avalon	0x00001820	0x0000183F	1
<input checked="" type="checkbox"/>	<input type="checkbox"/> pio_key	PIO (Parallel I/O)	clk	avalon	0x00001870	0x0000187F	2
<input checked="" type="checkbox"/>	<input type="checkbox"/> pio_lcd_data	PIO (Parallel I/O)	clk	avalon	0x00001880	0x0000188F	
<input checked="" type="checkbox"/>	<input type="checkbox"/> pio_lcd_ctl	PIO (Parallel I/O)	clk	avalon	0x00001890	0x0000189F	
<input checked="" type="checkbox"/>	<input type="checkbox"/> pio_seg7led	PIO (Parallel I/O)	clk	avalon	0x000018A0	0x000018AF	
<input checked="" type="checkbox"/>	<input type="checkbox"/> uart_usb	UART (RS-232 serial ...	clk	avalon	0x00001840	0x0000185F	3

▲ Move Up

uart_usb: 8-bit UART with 115200 baud, 1 stop bits and N parity (avalon)

3. 编译调试

分配好相应的管脚,以及设置好编译等信息。编译,并下载到 EPCS1 中去,即可开始调试 Nios 的软件。

参照以上教程在 NiosII IDE 中建立该实验的软件工程。

将光盘上“SSCOM32.EXE”和“sscom.ini”拷贝到硬盘某个目录,去掉只读属性。双击“SSCOM32.EXE”打开串口调试工具即可对该 USB 进行调试。



注：由于 CP2102 是一个 USB 到 UART 的桥接器，因此安装驱动后，实际是在操作系统内会增加一个虚拟的 COM 口，该 USB 口的通讯就是通过这个虚拟的 COM 口进行，所以无论是在实际应用中还是调试过程中，都是对该虚拟 COM 口操作即可，非常简单易用。通讯参数设置为：波特率为 115200，无校验位，8 位数据位，1 位停止位。

说明：该例子程序可以在学习套件附带光碟的“学习套件例子”找到。对应于 EP1C6 型号和 EP1C3 型号的核心板，该例子的所有文件分别保存在“EP1C6_Examples”或“EP1C3_Examples”的“all_test_USB”子目录中。

4. 结论

在嵌入式系统中，常常少不了与上位机的通讯，其中串口、USB 等是使用最广泛的通讯方式，当数据吞吐量需求不是很高的情况下，可以采用本实验的设计方案，简单、可靠，开发时间和风险都大大降低。

实验四——EP1C6 核心板 V2.0 的 Flash 烧写例子

EP1C6 核心板 V2.0 增加了 8Mbyte SDRAM 和 2Mbyte Flash，可容纳更多的用户程序和数据。如果用户程序和数据比较大，超出了 EPCS1 的容量，则程序和数据可以保存在普通 Flash 中；如果用户程序较大，超出了 EP1C6 所能定制的最大内部 RAM 容量，则也可以将程序放在 SDRAM 中运行。Flash 的烧写可以采用 NiosII IDE 的“Flash Programmer”来烧写，具体操作过程请参考

/altera/kits/nios2/documents 目录下的“ug_nios2_flash_programmer.pdf”或我们编写的《Nios IDE 下 Flash 烧写教程》(该教程包含在印刷版的用户手册中)。

“\EP1C6_Examples\”目录下的“burn_flash_ep1c6”和“ucosII_test”例子都是运行于 SDRAM 中的例子，同学们可以参照例子来实现更多有趣的实验。

实验五 关于 uc/OS-II 实时操作系统在 Nios II 中的运行

uc/OS-II 已经在世界范围内得到广泛的使用，包括手机、路由器、集线器、不间断电源、飞行器、医疗设备及工业控制等。实际上，uc/OS-II 已经通过了非常严格的测试，并且得到了美国航空航天管理局 (FAA) 的安全认证，可以用于飞机、航天器等与人生命攸关的控制系统中³。因此，uc/OS-II 在工业等实时性需求比较强的领域应用非常广泛，我们的学习套件提供了在 NiosII 系统中运行 uc/OS-II 的例子 (“\EP1C6_Examples\ucosII_test”目录下)，该例子定义了两个任务：Task1 和 Task2，两个任务交替执行。例子虽然简单，但可以作为 uc/OS-II 在 NiosII 系统中运行的演示。同学们可以参考 NiosII 的软件开发手册以及 uc/OS-II 的发明人——Jean J. Labrosse 的著作“MicroC/OS-II The Real-Time Kernel” (Second Edition) 的中译本《嵌入式实时操作系统 uc/OS-II》(第二版，邵贝贝等译)，研究和设计出功能更强、更符合实际应用的程序。

³ 《嵌入式实时操作系统 uc/OS-II》(第二版)

附录 CT-SOPC_x 系列 FPGA/SOPC 学习套件简介

关于 CT-SOPC_x 学习套件

CT-SOPC_x 学习套件，针对 Altera 的低成本高性能 Cyclone 系列 FPGA，助学习者快速掌握 FPGA/SOPC 设计技术。

- (1) 学习 FPGA 的设计技术；
- (2) 了解 NiosII 可编程片上系统的设计流程，学习在 FPGA 上包括定制 32 位的处理器、DSP 处理模块、逻辑功能模块等组成的系统；学习利用 ALTERA 公司提供的软件集成开发环境 (NIOS IDE) 设计、调试片上系统的软件；
- (3) FPGA 核心板可用于系统设计前期快速评估设计方案。

CT-SOPC_x 学习套件有以下主要特点：

- (一) 设计独特，既适合学生学习使用，也适合于实际项目设计中快速搭建系统原型以验证设计方案。核心板 (EP1C6 型 V2.0) 实际上是一块独立的 SOPC 最小系统板，SDRAM 和 Flash 都集成在核心板上，用户甚至可以用面包板制作特定项目所需的外围电路，并插上该核心板就构成了一个完整的项目系统原型。另外，如果项目试验中只需用到 FPGA 功能 (即无需定制 Nios 系统)，则可以将 SDRAM 和 Flash 卸下，核心板就成了一个将所有 FPGA 的 IO 管脚引出的 FPGA 核心板，同样可用于快速搭建项目系统原型。
- (二) 模块化结构，简单明了，有详尽文档、教程 (快速入门的教程，新版本为印刷版)，真正适合初学者。经验表明，很多学生在学习 FPGA、ARM、DSP 等设计技术的过程中，虽然刚开始学习热情很高，但真正能坚持下来一直到“学会”的却只有寥寥几个。除了学生个人的毅力因素的原因外，另外一个更主要的原因是由于所选用的学习板不适合初学者学习使用。对于初学者来说，理解和消化开发板并不容易 (事实上如果能达到这种程度的话，也就没必要借助学习板了)，再加上这些开发板在文档资料、教程上都非常欠缺，有些几乎连说明文档都没有，这就给学生们的学习增添了更多的困难，很多学生会因为找不到入门的口子而慢慢失去兴趣和信心。我们从初学者的角度出发，针对学习套件编写详尽的教程，从最简单的例子出发，希望学生从简单到深入地理解开发的过程，并能快速地跨入这个门槛，建立起学习的信息和兴趣。
- (三) 具有可持续学习性。传统的学习板将 FPGA 与实验电路集成一体，虽然预留了一些 FPGA 的 I/O 给学生扩展用，但预留的 I/O 有限，当学生学习到一定程度并初步具备自主设计能力的时候，这样的学习板显然已经不能满足进一步学习的要求。我们采用 FPGA 核心板与实验板相分离的结构，核心板主要由 FPGA 芯片和电源、配置芯片和时钟源组成，即 FPGA 的最小系统组成，FPGA 的所有 I/O 都以核心板的插针引出。当学生初步具备自主设计能力的时候，

可以自主设计实现更复杂功能的实验板，并将核心板插接到自主设计的实验板的插座上，即可利用 FPGA 的所有 I/O。

(四) 价格低，真正面向广大学习者，特别是在校学生。



CT-SOPC_x 学习套件实物照片



配套教程



配套用户手册

CT-SOPC_x 系列 FPGA/SOPC 学习套件包括 (实物照片如上图所示):

- (1) FPGA 核心板 ,使用高性价比 Cyclone 系列 FPGA ,EP1C3 相当于 5 万门左右 , EP1C6 相当于 12 万门左右 ;——EP1C6 核心板现已升级到 2.0 版本 ,增加 8Mbyte SDRAM 和 2Mbyte Flash。
- (2) 实验板V1.0——现已升级至实验板V2.0 版本 ,新增加USB、串行E²PROM等功能 ;
- (3) ByteBlasterII 下载线 ,使用于 ALTERA 公司全系列 CPLD/FPGA 芯片和 NiosII 软件调试 ;
- (4) USB 线 ;
- (5) 5V 电源 ;
- (6) 用户手册以及 FPGA/SOPC 快速教程 ;
- (7) 相关软件资料等 ;
- (8) 详尽电路原理图 ;
- (9) 1602A 型号 LCD ,(选配)。

如需了解更多关于该学习套件 , 请登陆www.21control.com , 或联系 :

电话 : 020-35511160 , 13760811768

QQ : 26128396

Email : rouder@163.com

www.21control.com

注 : 目前我们已发现某公司盗版我们的 CT-SOPC_x 学习套件 , 该公司对该产品及相应资料只是直接抄袭和拷贝 , 因此无论是功能的升级还是资料的更新 , 都远跟不上我们的步伐。希望大家购买时认准 “ CT-SOPC_x ” , 并通过我们或我们的代理商 (可咨询我们来确认代理商) 购买。同时 , 对于每套新版本的 CT-SOPC_x 学习套件 , 我们都编制了唯一的序列号 , 所有的售后服务以及技术支持都以该序列号为凭证。

CT-SOPC_x 系列 FPGA/SOPC 学习套件的例子

1. FPGA 例子

- (1) FPGA_led_test ——最简单的 FPGA 例子，一个按钮控制一个 LED 灯的亮或灭，带领同学们跨入 FPGA 设计的门槛；
- (2) seg7led_test ——进一步熟悉 QuartusII 软件，了解用硬件描述语言设计用户功能模块的流程；

2. SOPC (Nios II) 例子

- (1) soc_led ——最简单的 NiosII 系统，运行于 FPGA 内部的软件控制两个 LED 灯交替闪烁，让同学们初尝让人激动的 SOPC 设计；
- (2) all_test ——学习套件 V1.0 的所有模块的测试程序，测试内容包括：LED、串口、数码管、按键开关、LCD；
- (3) all_test_USB ——学习套件 V2.0 所有模块的测试程序，测试内容包括：LED、USB 口、数码管、按键开关、LCD；
- (4) burn_flash_ep1c6 ——用 NiosII IDE 的 Flash Programmer 烧写 EP1C6 核心板 V2.0 上的 Flash 例子（EP1C3 型核心板不支持该例子）；
- (5) my_board ——用 NiosII IDE 的 Flash Programmer 烧写 EP1C6 核心板 V2.0 上的 Flash 时制作“Target Board”的例子（EP1C3 型核心板不支持该例子）；
- (6) ucosII_test ——在 NiosII 系统中运行 uc/OS-II 的例子（EP1C3 型核心板不支持该例子）。

我们将不断改善和升级我们的设计，努力为广大嵌入式系统学习者和工业控制系统设计者提供更完美的方案，请关注、支持我们，QQ 或 E-mail 给我们提出您的宝贵建议，谢谢！

此外，由于我们的水平有限，本教程存在的错漏之处在所难免，请各位朋友批评指出，并希望与我联系以便更正，谢谢。

我的 E-mail：rouder@163.com ,QQ:26128396