台湾友晶公司 DE2-70 实验平台

FPGA/SOPC 入门级实验指导书

适用 Altera 公司 FPGA 芯片 EP2C70F896C6
适用 Altera 公司软件工具 Quartus II　V7.2 / V8.0

1.5 版

南京大学计算机系整理改编

2009 年 6 月 29 日星期一

目　录

第 1 章　DE2-70 开发板驱动安装

1.1 DE2-70 介绍

Altera DE2-70（以下简称 DE2-70）实验/开发两用板是台湾友晶公司为高等院校学生学习 FPGA 编程和学习基于 Nios 软核处理器应用项目编程而开发的实验平台。DE2-70 多媒体开发平台安装了一片拥有 70，000 个逻辑单元的 Cyclone® II 系列 2C70 型 FPGA 芯片，该芯片是 Altera 公司出产的。它的全称是：EP2C70F896C6。

DE2-70 与 Altera DE2-70（以下简称 DE2-70）相兼容，也具有 DE2-70 多媒体平台丰富的多媒体、储存及网络等应用接口的优点。以下是 DE2-70 实验开发板的平面视图。参看图 1-1。

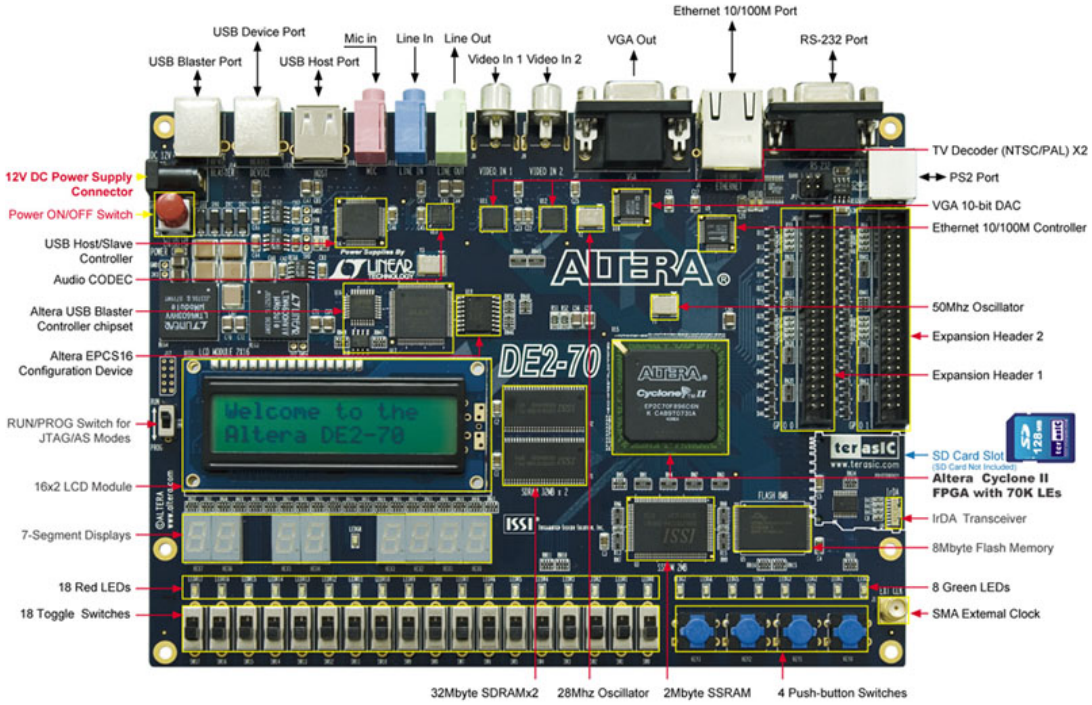


图 1-1　DE2-70 开发板/教学实验板俯视图

Altera 公司网站提供了 DE2-70 开发板/教学实验板的教学资源。其网址如下列出。

http://www.altera.com/education/univ/materials/unv-overview.html

DE2-70 实验平台具有较强的多媒体功能。特别是它具有 VGA 输出功能，可以把视频数据输出到 VGA 视频显示器上。用户可以通过演示程序来体验 DE2-70 实验平台多媒体功能。表 1-1 给出了这些演示程序的列表。

DE2-70 控制面板工具程序	卡拉 OK 机功能展示
DE2-70 影像工具程序	以太网络封包传递功能展示
电视盒	SD 卡音乐播放程序功能演示
电视盒 Picture in Picture 子母画面	音乐合成器功能演示
USB Paintbrush 展示	音乐录制及播放功能

USB 装置功能展示	

表 1-1　DE2-70 实验平台的演示程序

DE2-70 开发板/教学实验板的主要器件技术特点如表 1-1 所示。

器件	描　　述
FPGA	Cyclone II EP2C70F896C6 with EPCS16 16-Mbit serial configuration device
I/O Devices	Built-in USB-Blaster cable for FPGA configuration
	10/100 Ethernet
	RS232
	Video Out (VGA 10-bit DAC)
	Video In (NTSC/PAL/Multi-format)
	USB 2.0 (type A and type B)
	PS/2 mouse or keyboard port
	Line In/Out， Microphone In (24-bit Audio CODEC)
	Expansion headers (76 signal pins)
	Infrared port
Memory	8-MBytes SDRAM， 512K SRAM， 4-MBytes Flash
	SD memory card slot
Displays	16 x 2 LCD display
	Eight 7-segment displays
Switches and LEDs	18 toggle switches
	18 red LEDs
	9 green LEDs
	Four debounced pushbutton switches
Clocks	50 MHz crystal for FPGA clock input
	27 MHz crystal for video applications
	External SMA clock input

表 1-2　DE2-70 开发板/实验板的器件技术特点

1.2 USB-Blaster 的驱动安装

除了加电之外，为了让 DE2-70 开发板/教学实验板正常进行开发工作，还需要在主机上安装 USB-Blaster 电缆的驱动程序以支持 PC 端的开发软件，如 Quartus II、Nios II IDE 等。

安装环境相关说明：

开发软件：Quartus II 7.2/8.0

开发平台：Windows XP SP3

以下是 USB-Blaster 驱动程序在 Windows XP 下的安装步骤。

1.　将 DE2-70 实验平台的 Blaster 接口(开发板上部最左边)接好 USB 连接线，插头插入主机的 USB 接口，Windows XP 发现新硬件后会弹出一个对话框。参看图 1-2。

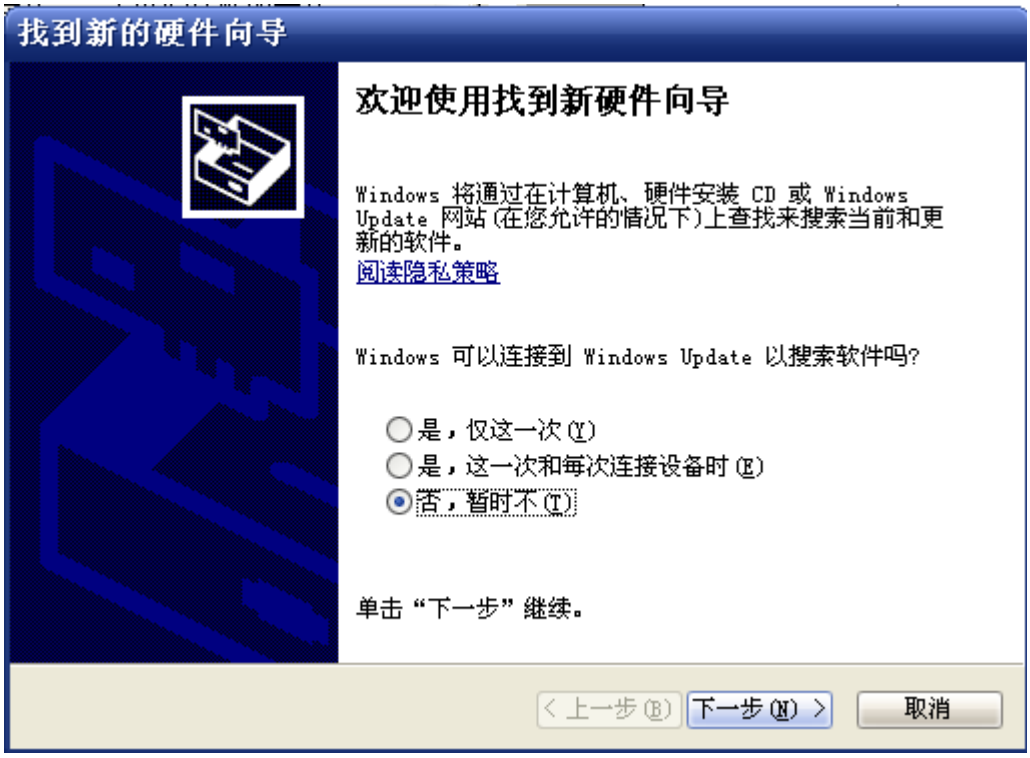


图 1-2　找到新的硬件向导（未识别）

注意：只有计算机没有识别出 USB-Blaster 才会出现图 1-2，如果是卸载后重新安装或者预装载一些驱动信息，则会从图 1-3 开始。

2.　硬件向导

图 1-3　找到新的硬件向导（识别）

3.　选择搜索和安装选项，如图 1-4。

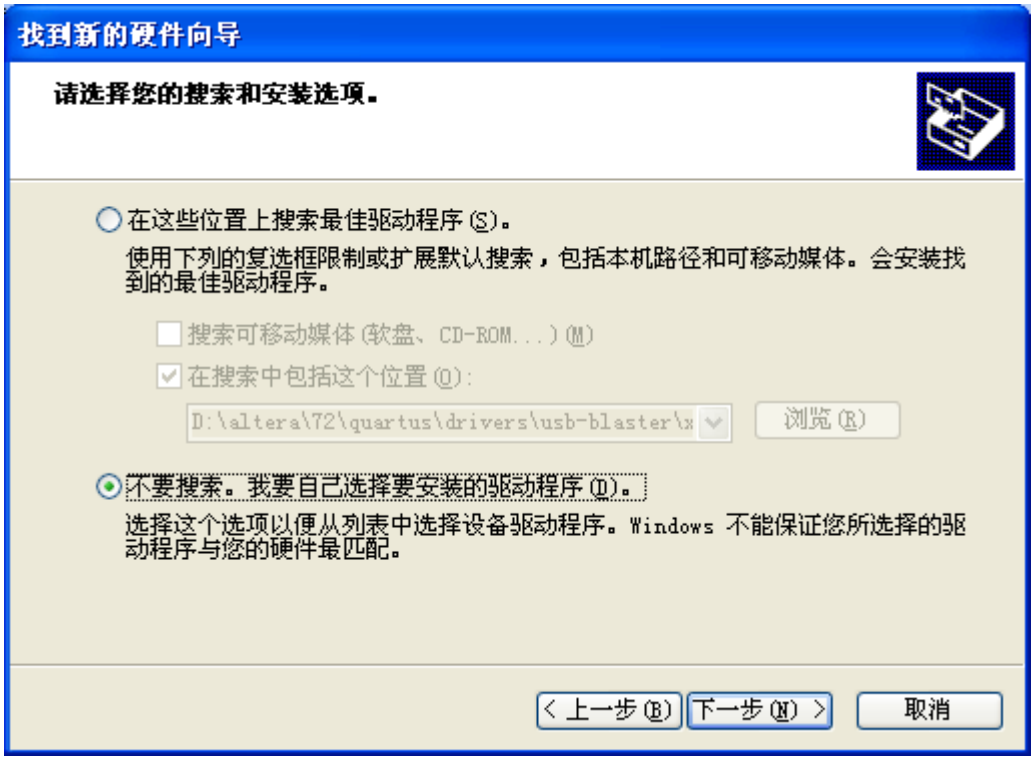


图 1-4　选择搜索和安装选项

4.　选择硬件类型

图 1-5　选择硬件类型

注意：只有从来没有安装过 USB-Blaster 才会出现图 1-5，如果是卸载后重新安装，则会直接转到图 1-6。

5.　选择从磁盘安装

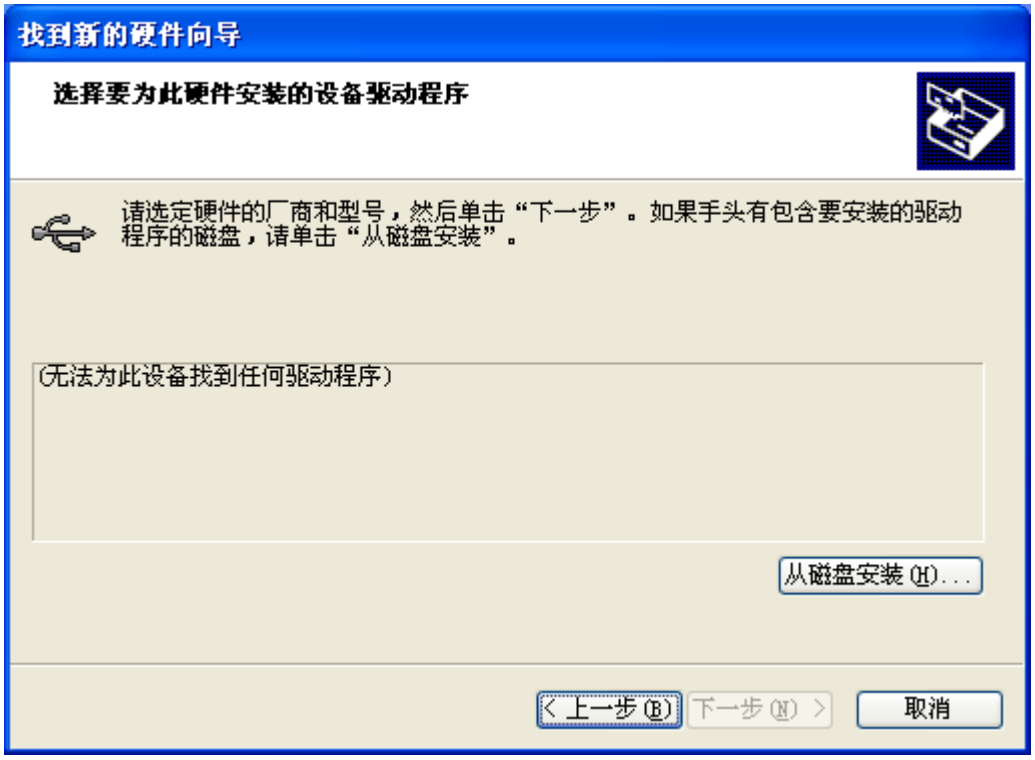


图 1-6A　选择驱动程序（未发现）

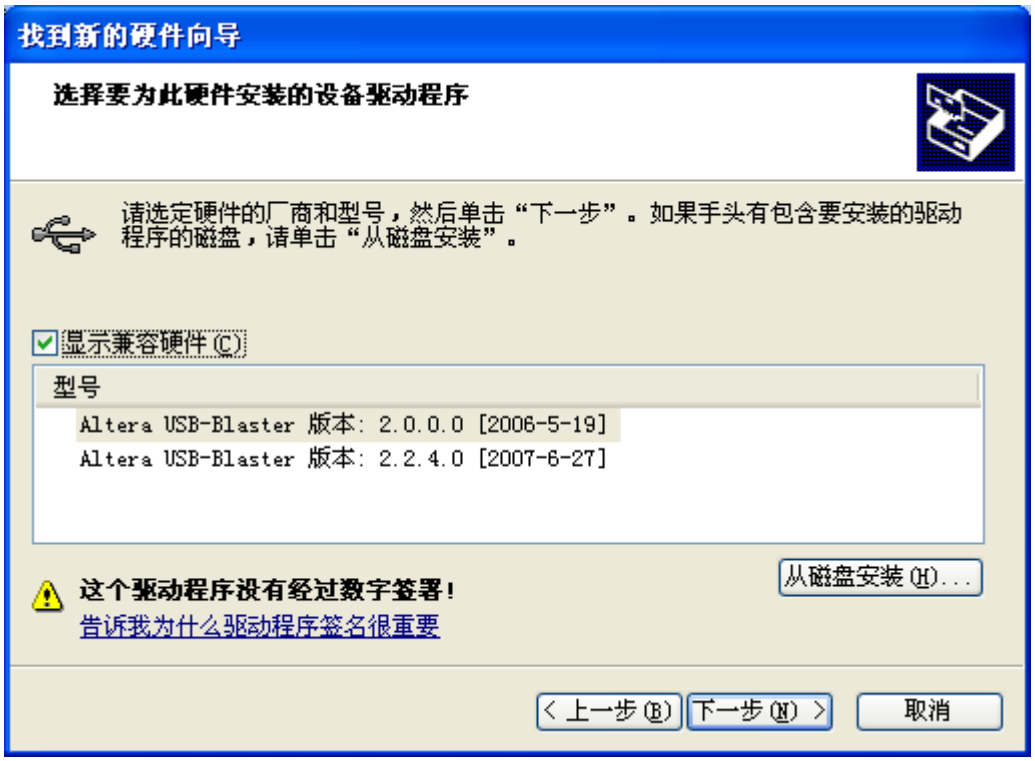


图 1-6B　选择驱动程序（发现）

如果从没装过，则如图 1-6A 所示；如果以前安装过 USB-Blaster 驱动，这时会显示兼容列表，如图 1-6B。选择从磁盘安装。

6.　选择路径及文件，如图 1-7。

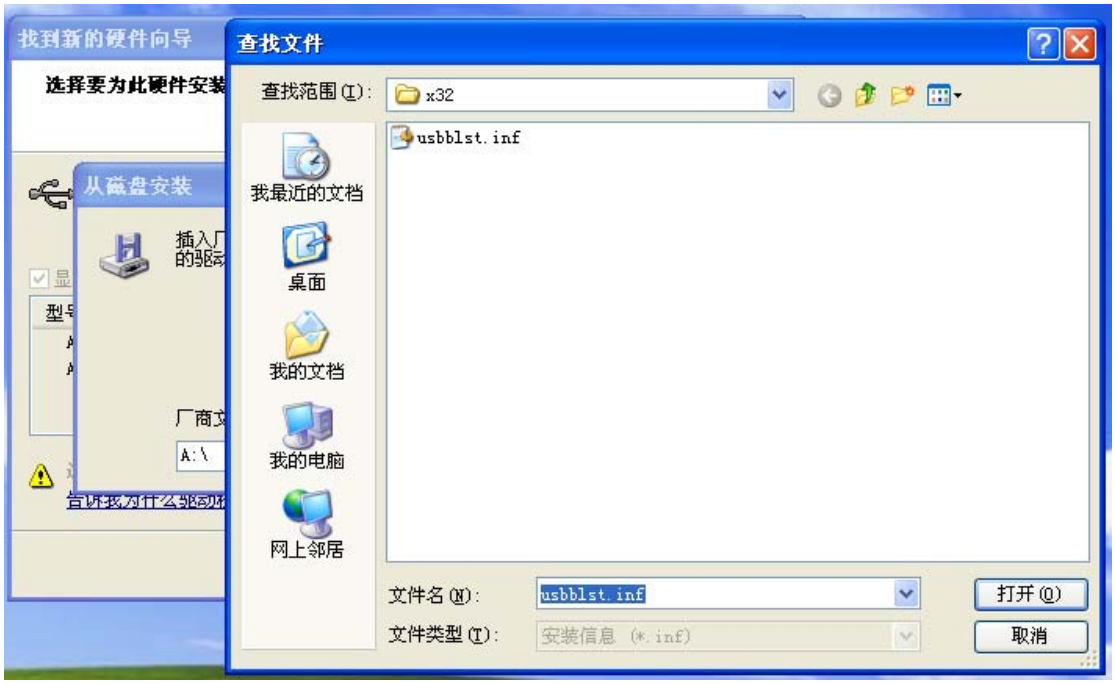


图 1-7　选择驱动程序文件路径

注意：7.2 的驱动文件路径在 D:\altera\72\quartus\drivers\usb-blaster\x32\usbblst.inf

8.0 的驱动文件路径在 D:\altera\80\quartus\drivers\usb-blaster\usbblst.inf

8.0 的 x32 与 x64 共用一个 inf 文件请注意！！

7.　选择驱动，如图 1-8。

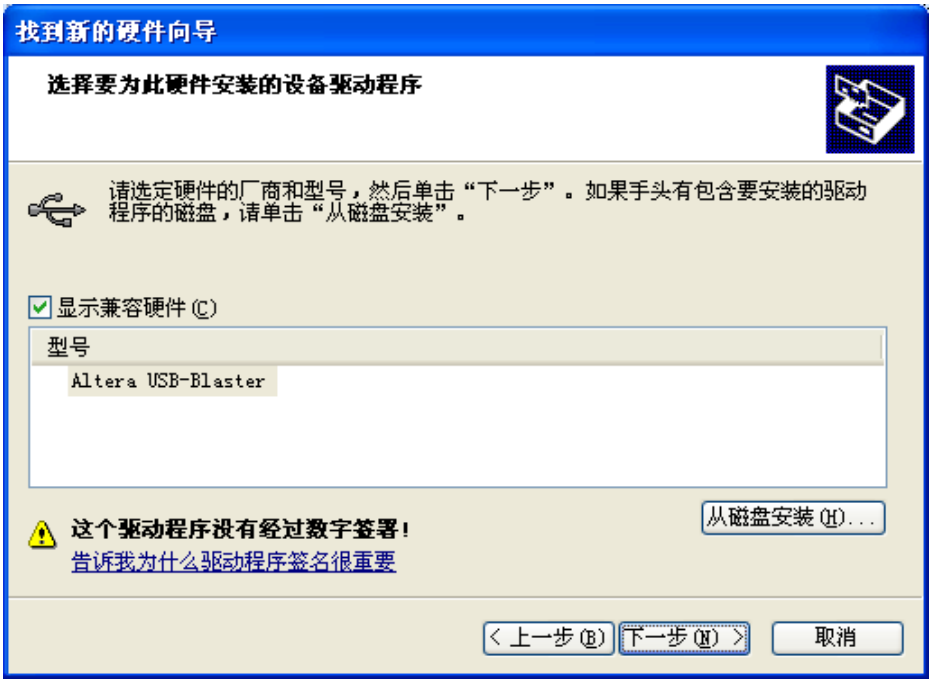


图 1-8　选择驱动程序(手动指定完毕)

8.　单击“下一步”及“仍然继续”，如图 1-9。

图 1-9　驱动程序未签名提示

9.　安装进程，如图 1-10。

图 1-10　驱动程序安装中

10.　完成安装的提示对话框如图 1-11。

图 1-11　驱动程序安装完成

11. 右键单击我的电脑，进入属性页，再进入“硬件”标签页，单击“设备管理器”对话框，单击“通用串行总线控制器”图标，查看安装是否成功。如图 1-12 所示。

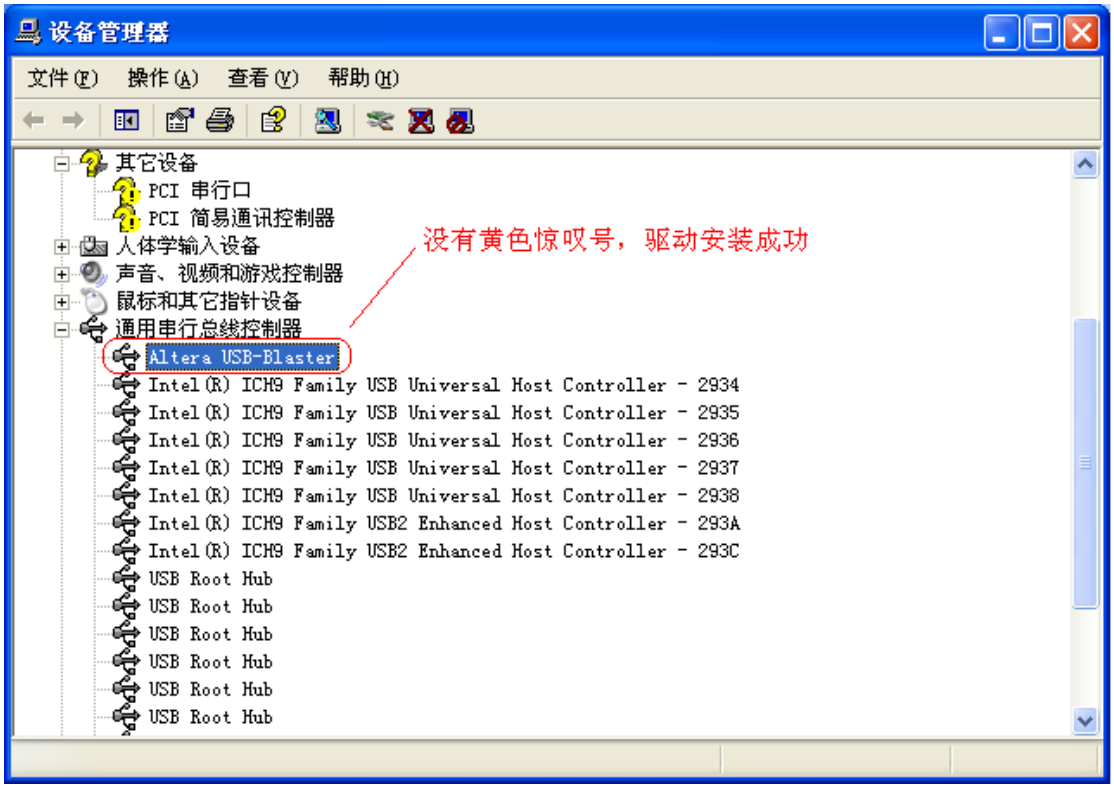


图 1-12　驱动程序安装成功与否检查

以上就是 DE2-70 实验平台的驱动安装过程。

1.3 USB-Blaster 驱动之疑难解答

问 1：USB-Blaster 8.0 的驱动在 Quartus II 7.2 下无法正常工作？
答：是的，如果想同时安装 Quartus II 7.2 与 8.0，必须使用 7.2 的 Altera USB-Blaster 驱动。

问 2：驱动无法在 X64 环境下使用?
答：这是因为 Altera 公司的驱动未加数字签名，而 Windows Vista x64 不允许加载未加数字签名的驱动；如果使用的是 Windows XP x64 环境，可以尝试使用 USB-Blaster 7.2 的驱动。

问 3：更换 Quartus 版本无法安装 USB-Blaster Driver 驱动，提示“名称已用作服务名或服务显示名”?
答：windows 设备管理器中的驱动卸载操作卸不干净，必须手动在注册表中删除服务，之后重新安装。

问 4：更换 Quartus 版本无法安装 USB-Blaster Driver 驱动，提示“找不到文件”?
答：很可能是原有驱动文件被破坏但注册表中信息未同步删除。手工 copy 驱动文件到如下位置，重启再次安装驱动。
windows\system32\drivers\ftdibus.sys
windows\system32\ftbussui.dll
windows\system32\ftd2xx.dll
windows\system32\drivers\usbblstr.sys (8.0 专有)
windows\system32\usbblstr32.dll(8.0 专有)
windows\system32\usbblstrui.dll(8.0 专有)

1.4 DE2-70 引脚分配的一般性指导

1. **LED** 灯：有两组，**LEDR[17:0]**和 **LEDG[7:0]**

   这两组 LED 灯用于简单输出。一般用于二进制结果输出，如果是较大的十进制数，采用 HEX 或者 LCD 输出较好。**oLEDR** 与 **oLEDG** 除了数量与颜色不同外，用法基本一致。
2. **HEX** 发光管 **HEX[7:0]**，用于数值的输出。

   一般用于十进制或十六进制结果的输出，有时也可用来显示英文字符。
3. 开关 **SW[17:0]**：用于简单的输入。

   拥有输入并保持同一电平信号的优势，一般用于数据信号或者功能控制信号。相对于按钮来说，可以用开关手工模拟低速的方波信号。
4. 按钮 **KEY[3:0]**：用于简单的输入。

   平时状态是高电平，按下时低电平，与开关各有优劣，一般用于复位信号与单步调试时的时钟信号。
5. 额外时钟：用于额外的时钟输入。

   位于开发板的右下角，直径为 8mm 左右，一个带螺纹的黄铜接口。当实验工程中含有额外时钟信号发生器的时候，可以用引线外接到 **DE2-70** 上进行时钟输入。使用场合较少。

第 2 章　实验一　3-8 译码器实验

- 实验说明

Quartus II 设计工具支持多种设计输入模型，本次实验使用 Verilog 硬件描述语言在 DE2-70 开发平台上设计一个基本组合逻辑电路——3-8 译码器。通过这个实验，读者可以了解使用 Quartus 工具设计硬件的基本流程。

- 实验步骤

2.1 建立 Quartus 工程

1. 打开 Quartus II 工作环境，如图 2-1 所示。

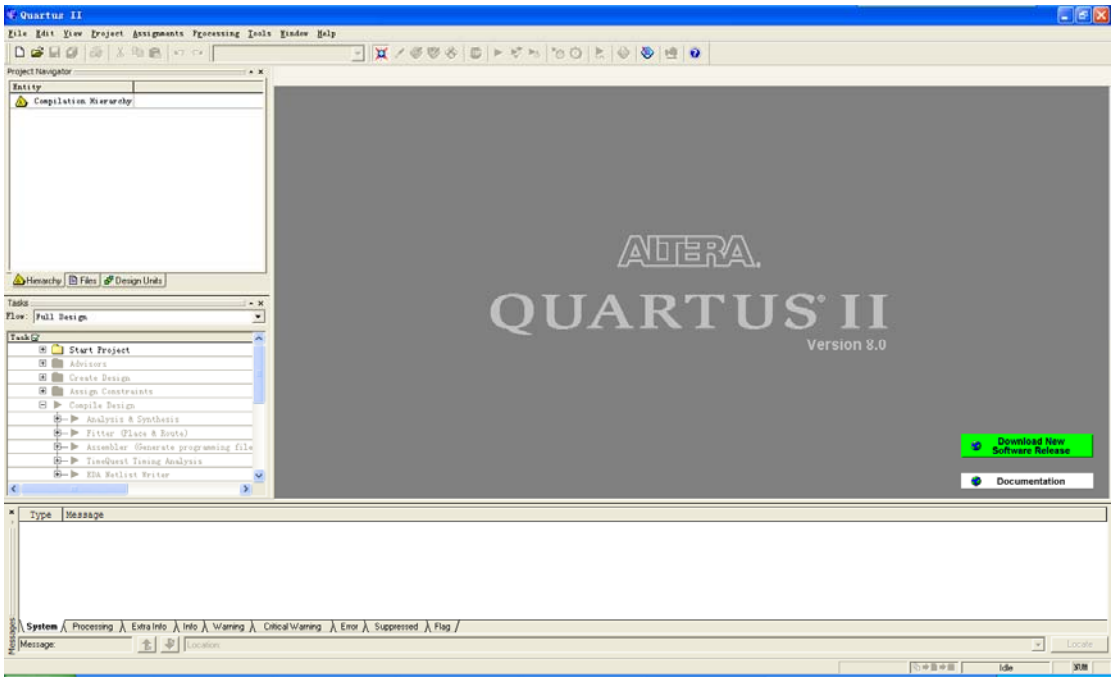


图 2-1　Quartus II 工作环境界面

2. 点击菜单项 File->New Project Wizard 帮助新建工程。参看图 2-2。

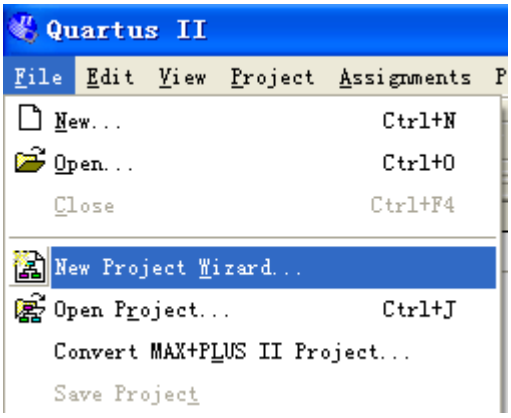


图 2-2　选择 New Project Wizard

打开 Wizard 之后，界面如图 2-3 所示。点击 Next，

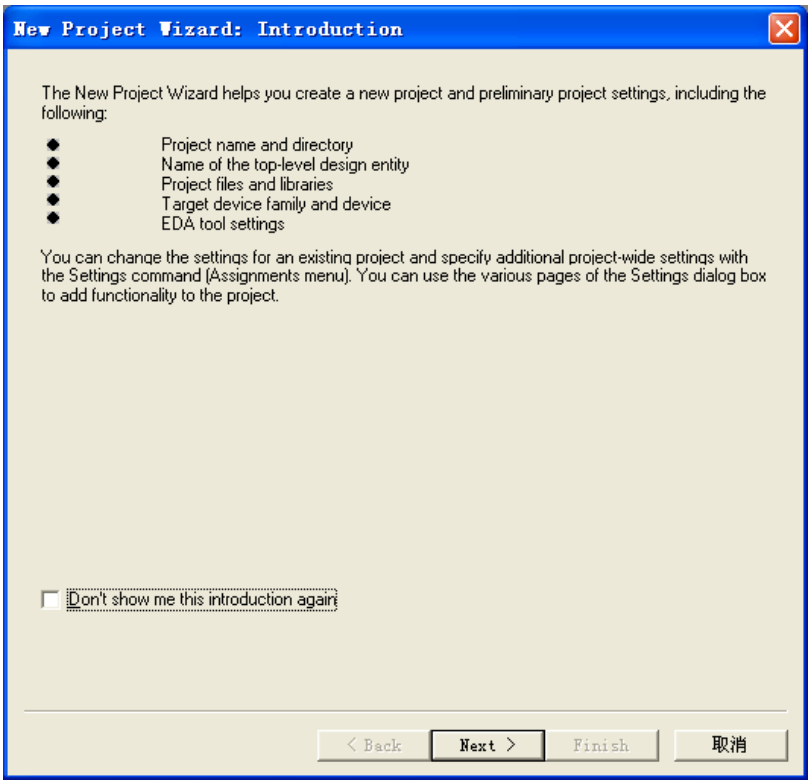


图 2-3　New Project Wizard 界面

3.　输入工程工作路径、工程文件名以及顶层实体名。

注意：这里输入的顶层实体名必须与之后设计文件的顶层实体名相同，默认的顶层实体名与工程文件名相同，本次实验采用这种命名方法，用户也可以根据需要输入不同的顶层实体名。输入结束后，如图 2-4 所示。点击 Next。

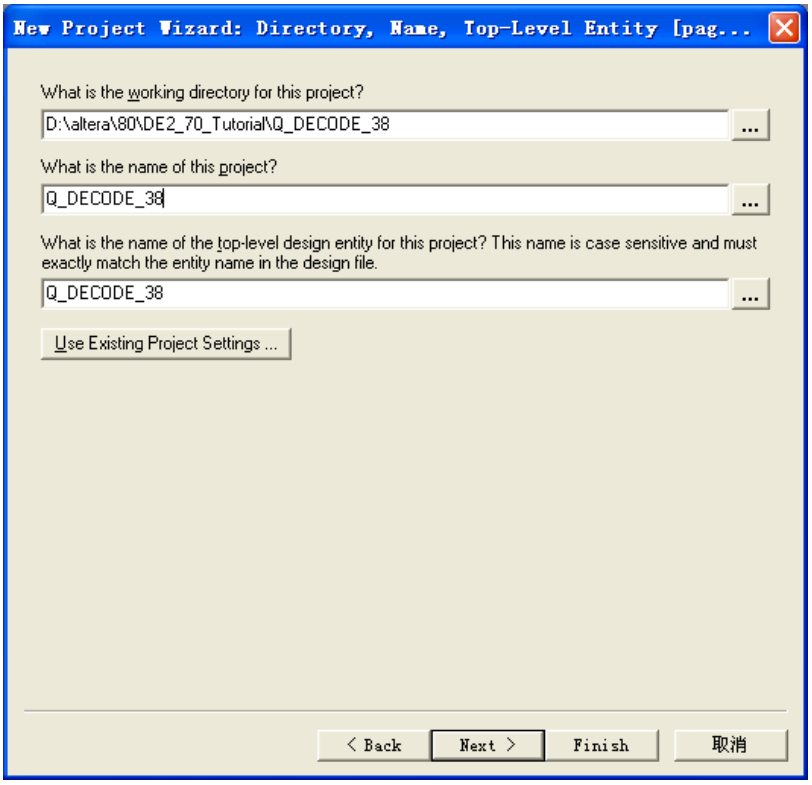


图 2-4　输入设计工程信息

4.　添加设计文件。界面如图 2-5 所示。如果用户之前已经有设计文件（比如.v 文件）。那么再次添加相应文件，如果没有完成的设计文件，点击 Next 之后添加并且编辑设计文件。

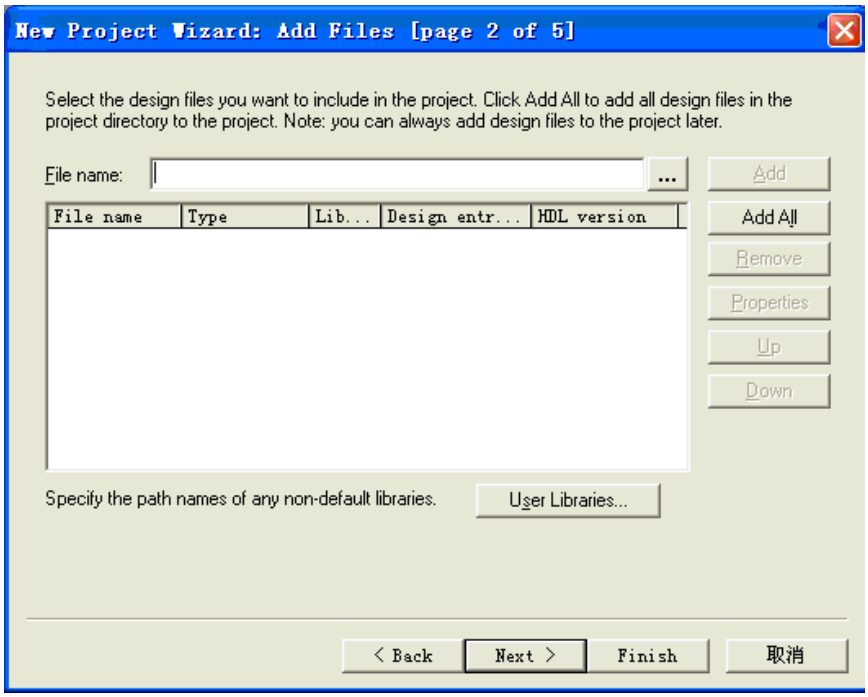


图 2-5　添加设计文件

5.　选择设计所用器件。由于本次实验使用 Altera 公司提供的 DE2-70 开发板，用户必须选择与 DE2-70 开发板相对应的 FPGA 器件型号。

在 Family 菜单中选择 Cyclone II，Package 选 FBGA，Pin Count 选 896，Speed grade 选 6，确认 Available devices 中选中 EP2C70F896C6，如图 2-6 所示。

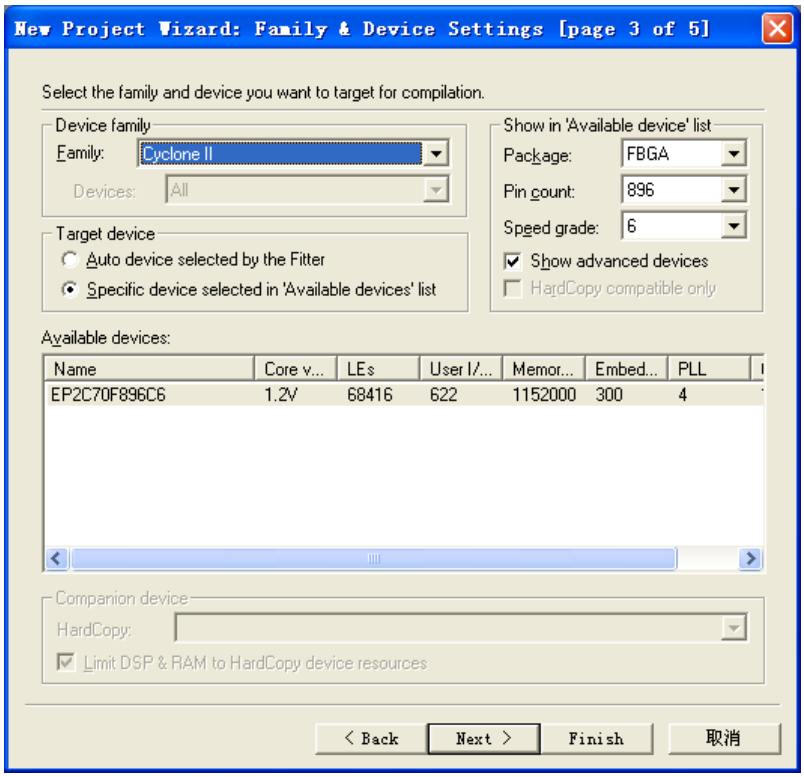


图 2-6　选择相应器件

6.　设置 EDA 工具。设计中可能会用到的 EDA 工具有综合工具、仿真工具以及时序分析工具。本次实验中不使用这些工具，因此点击 Next 直接跳过设置，如图 2-7 所示。

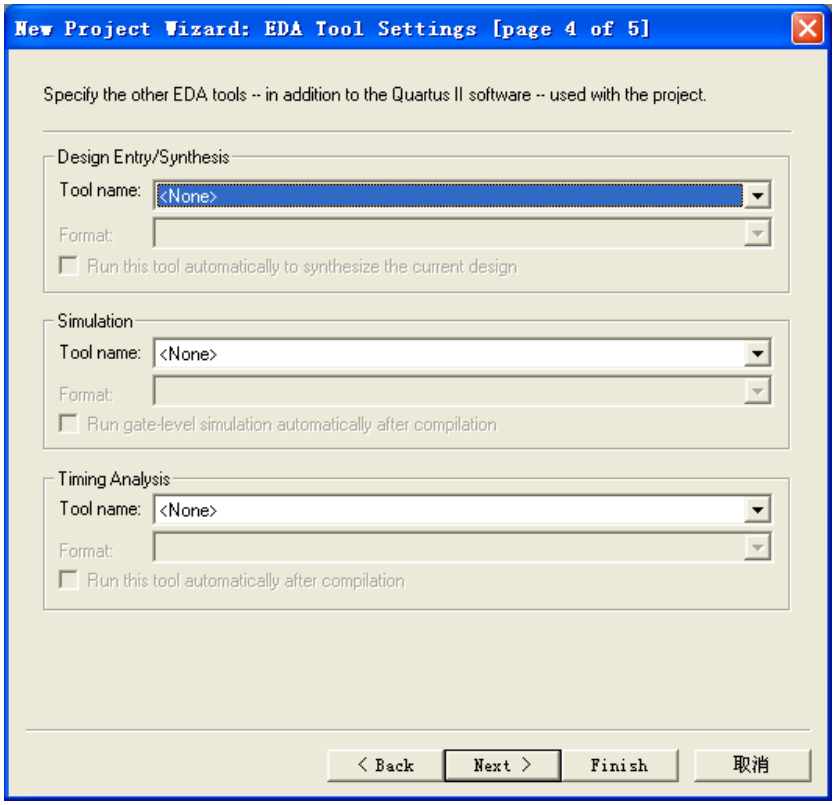


图 2-7　设置 EDA 工具

7.　查看新建工程总结。在基本设计完成后，Quartus II 会自动生成一个总结让用户核对之前的设计，如图 2-8 所示，确认后点击 Finish 完成新建。

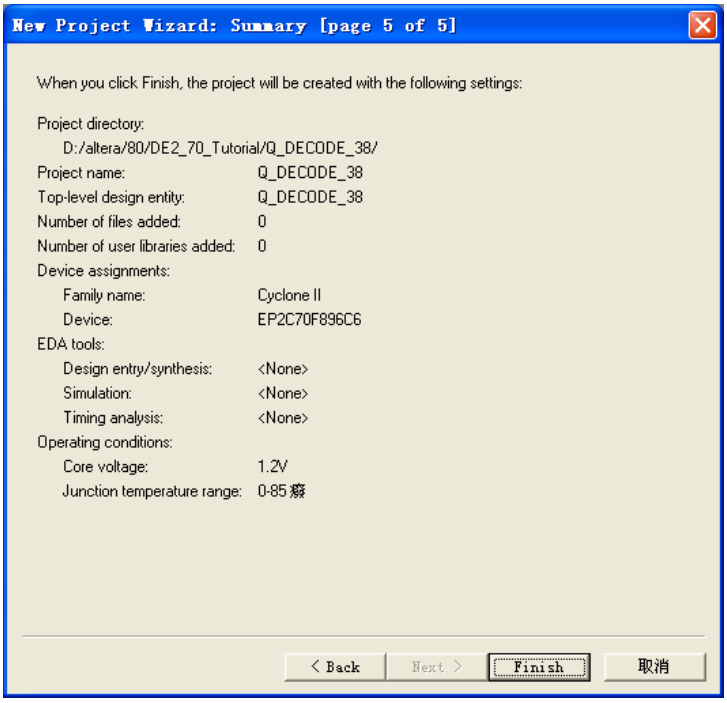


图 2-8　新建工程总结

在完成新建后，Quartus II 界面中 Project Navigator 的 Hierarchy 标签栏中会出现用户正

在设计的工程名以及所选用的器件型号，如图 2-9 所示。

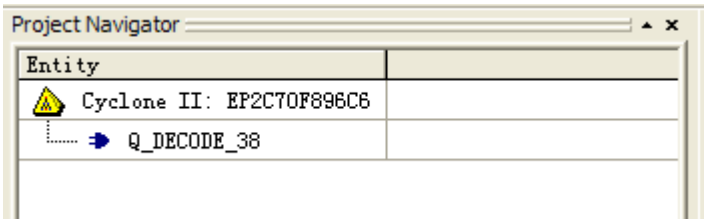


图 2-9　观察正在设计的工程

8.　培养良好的文件布局。Quartus II 默认把所有编译结果放在工程根目录，为了让 Quartus II 像 Visual Studio 等 IDE 一样把编译结果放在一个单独的目录中，需要指定编译结果输出路径。

点击菜单项 Assignments->Device，选中 Compilation Process Settings 选项卡，勾上右边的 Save Project output files in specified directory，输入路径(一般为 debug 或者 release)，如图 2-10 所示。

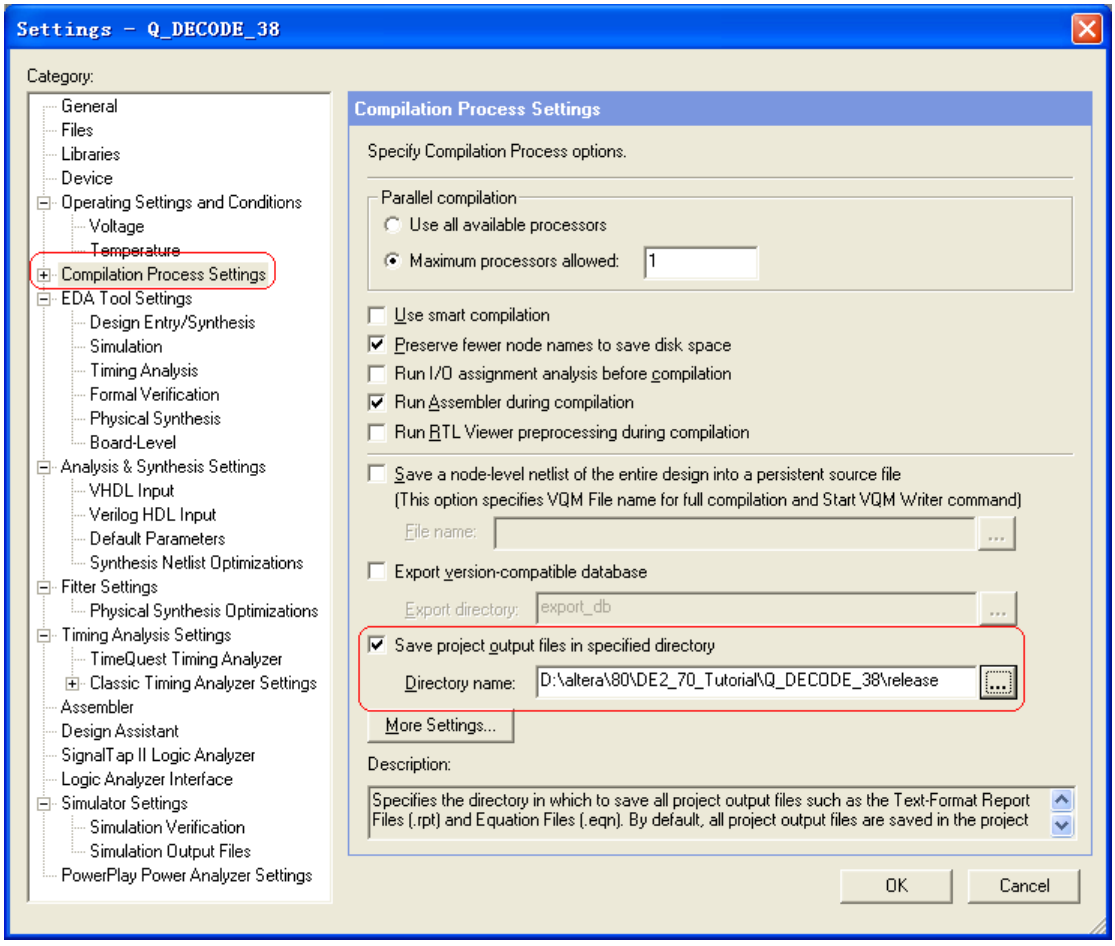


图 2-10　指定单独的编译结果文件目录

2.2 使用硬件描述语言完成硬件描述设计

9.　添加所需设计文件。本次实验通过 Verilog HDL 来描述所设计的硬件，因此要添加 Verilog 设计文件到工程文件中去。

点击菜单项 File->New、点击图标或者使用快捷键 Ctrl+N 新建一个设计文件，选

择 Verilog HDL File，如图 2-11 所示，点击 OK。

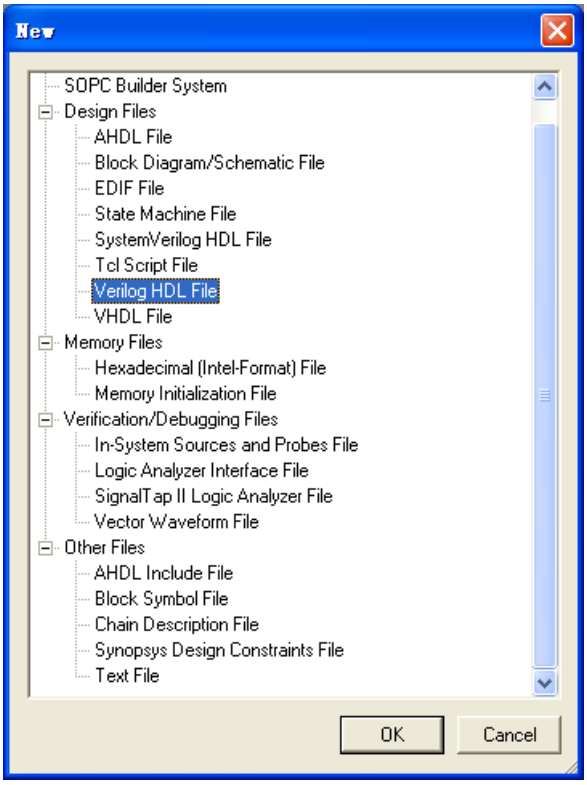


图 2-11　选择设计文件类型

10. 输入硬件描述。在 Quartus II 环境提供的文本编辑器中输入用户设计的硬件描述语言，在本次实验设计的是一个 3-8 译码器，输入代码如图 2-12 所示。

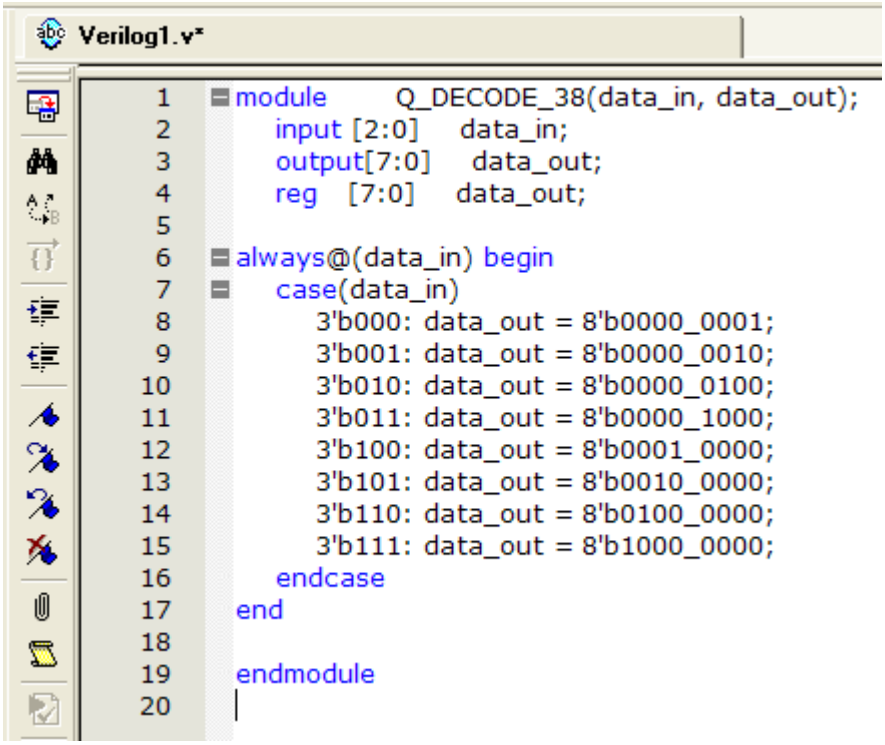


图 2-12　输入设计代码

11. 保存设计。点击菜单项 File->Save、点击图标或者使用快捷键 Ctrl+S 保存设计，

如图 2-13 所示。给设计文件命名 Q_DECODE_38，与 3-8 译码器的模块名相同，点击保存。

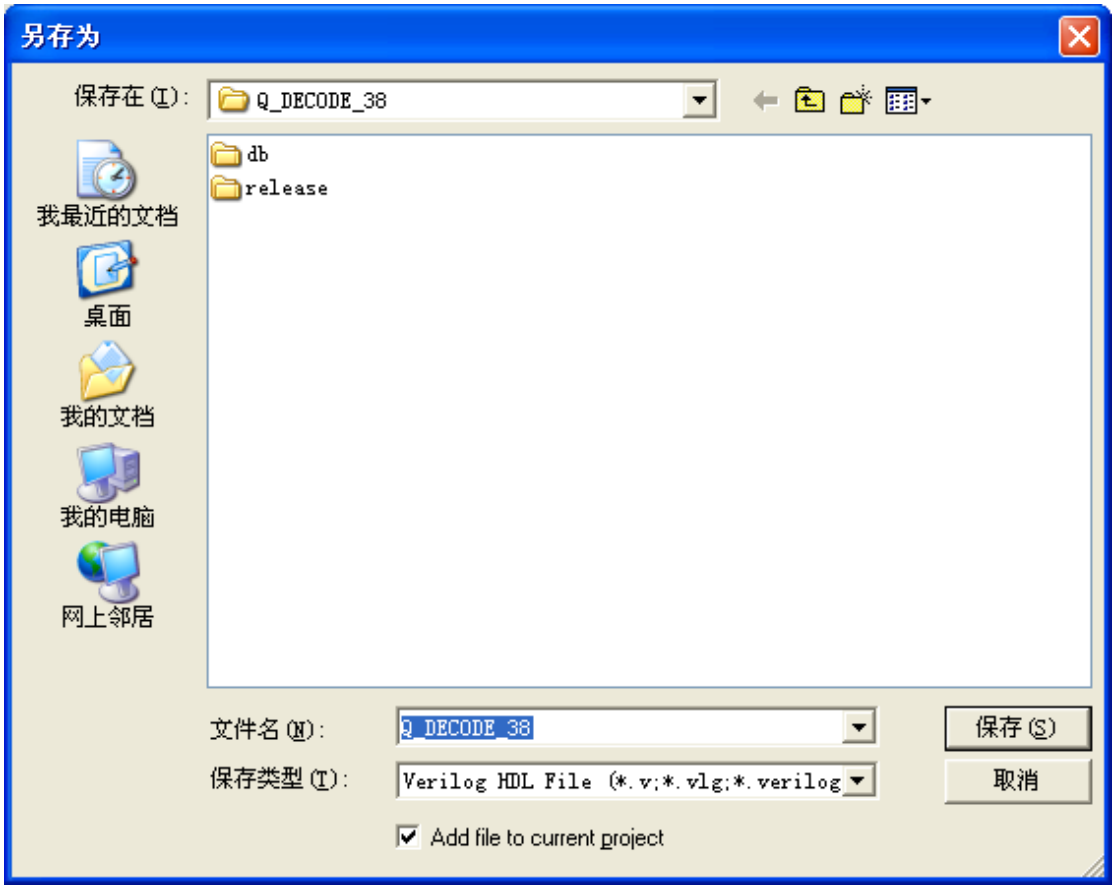


图 2-13　保存设计文件

12. 分析与综合。点击菜单项 Processing->start->Start Analysis & Synthesis、点击图标 或者使用快捷键 Ctrl+K 执行分析与综合。参看图 2-14。

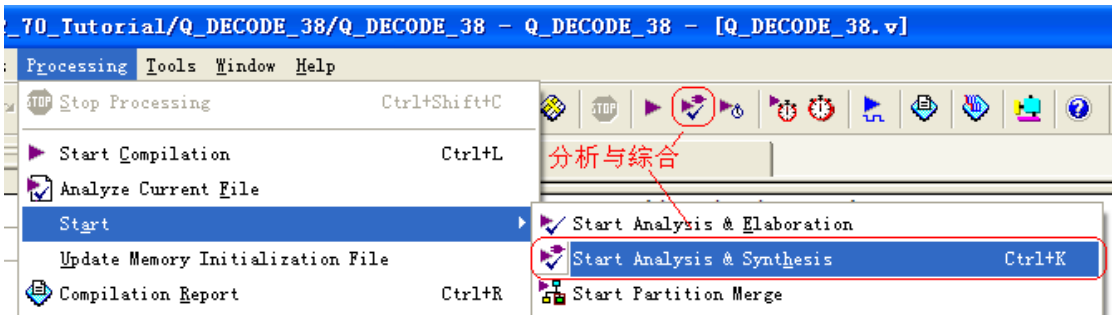


图 2-14　执行 start Analysis & Synthesis（开始分析与综合）

注意：Start Analysis & Synthesis(分析与综合) = Start Analysis & Elaboration(分析与解析)+ Mapping(映射)。

如果仅仅需要检查语法，那么执行 Analysis & Elaboration 即可，但是这一步生成的数据库并不对应 FPGA 器件的物理结构，生成的网表中结点的名称也不与 FPGA 器件的 Cell 名称对应。而且这一操作没有快捷键支持，更多的情况下直接执行 Start Analysis & Synthesis。

Start Analysis & Synthesis 后，生成的数据库已经对应了 FPGA 器件的物理结构，“映射”后的数据库包含了 FPGA 底层 Cell 的位置信息和 Cell 本身的时序信息。

分析与综合完成后，状态窗口如图 2-15 所示。

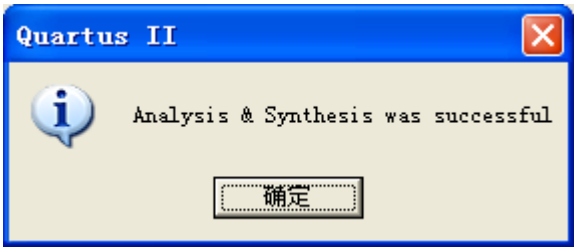


图 2-15　执行 start Analysis & Synthesis 后

13. 全编译文件。点击菜单项 Processing->start compilation、点击图标 或使用 CTRL+L 执行全编译，如图 2-16。

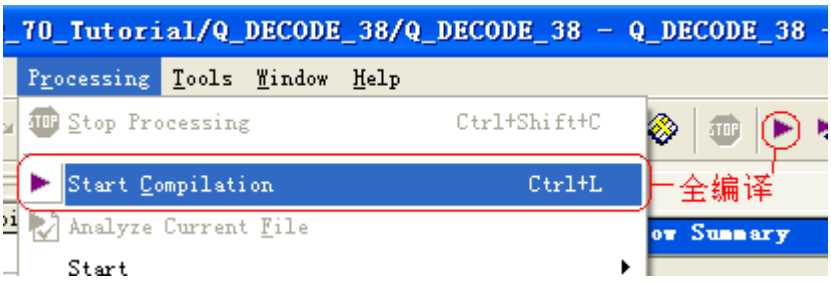


图 2-16　执行 start compilation

编译结果如图 2-17 所示。

图 2-17　全编译结果显示

注意：进行这次全编译仅仅是为了利用 Assignments->Pins 来手工分配引脚，分配完成后需要再次全编译。如使用 qsf 文件分配引脚则只需全编译一次即可。

14. 为 DE2-70 运行 3-8 译码器配置引脚。配置引脚有 3 种方法，分别是手工指定、使用 csv 文件导入与使用 qsf 文件，这里使用第一种，剩下两种后面的实验中会提到。点击菜单项 Assignments->Pins，如图 2-18。

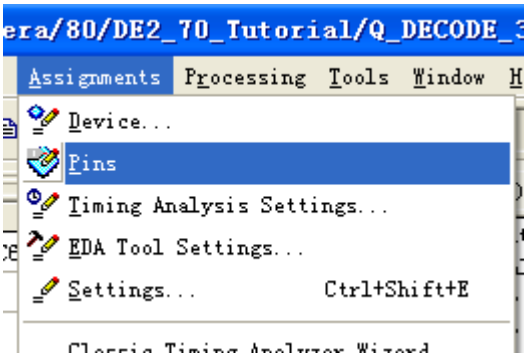


图 2-18　配置 DE2-70 的引脚操作

15. Pins 菜单项执行之后，会出现一个引脚配置窗口。参看图 2-19。我们只用该窗口的下部的列表进行具体信号的引脚指定，参看图 2-20。

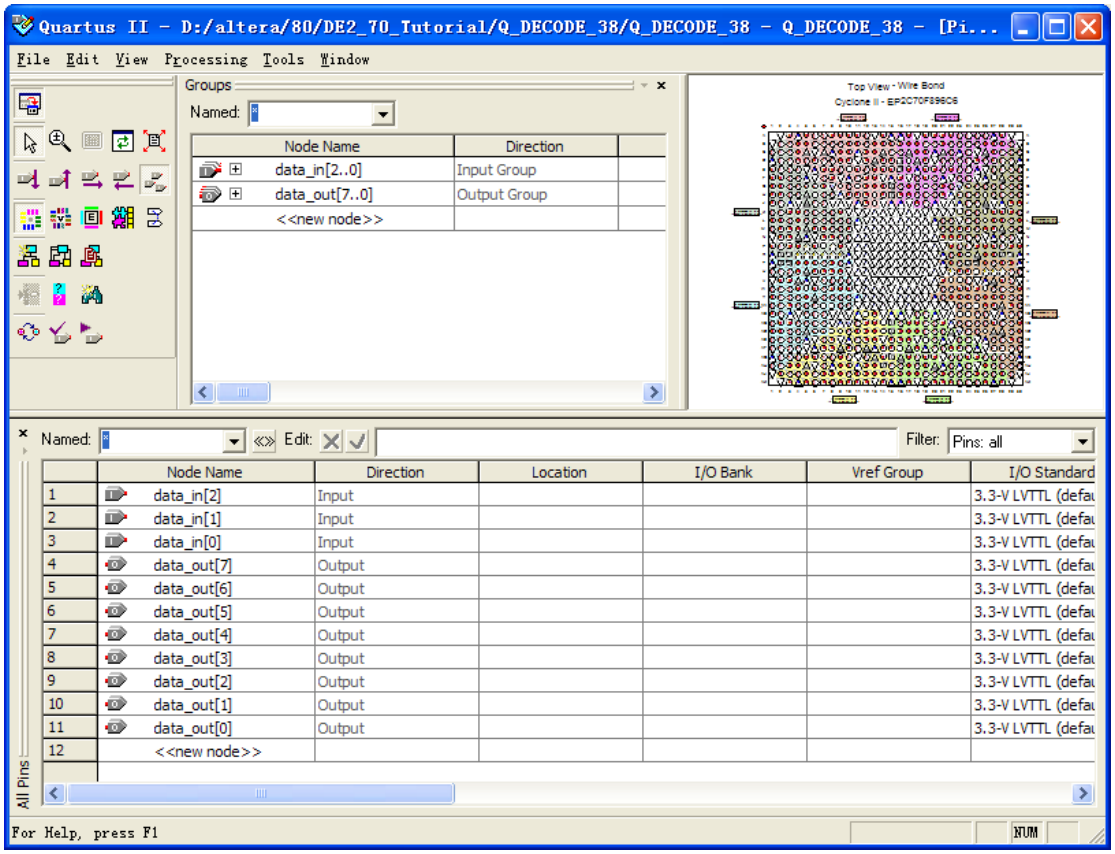


图 2-19　引脚配置窗口

为了将逻辑分配到 FPGA 外围引脚上，必须根据所用的 FPGA 型号配置输出引脚。根据所提供的 DE2-70 用户指导手册，将 3-8 译码器的输入与输出分别配置到 DE2-70 开发板的 3 个选择开关（SW2，SW1，SW0）以及 8 个 LED（LEDR7-LEDR0）上。

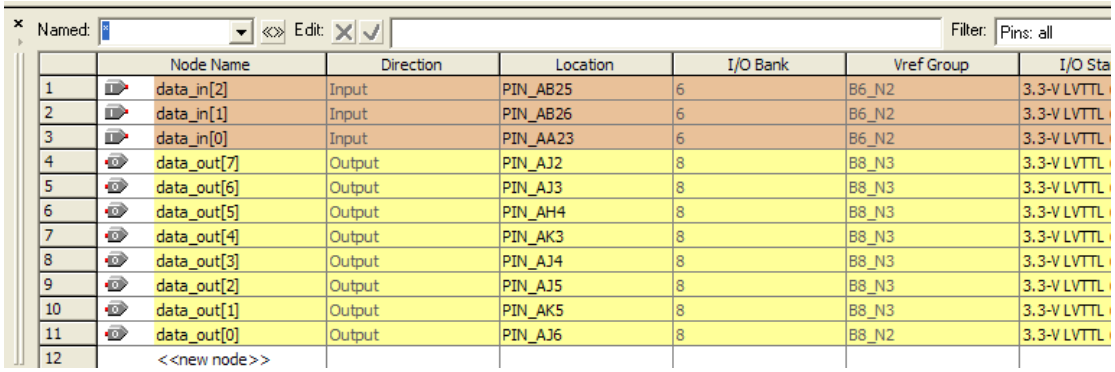


图 2-20　输入引脚编号

16. 全编译文件。完成分配引脚后，点击菜单项 Processing->start compilation、点击图标  或使用 CTRL+L 执行全编译，生成 sof 目标文件。结果如图 2-21 所示，可见引脚未指定的 warning 已经去掉。

图 2-21　编译结果窗口

17. 将设计下载在 FPGA 中。点击菜单项 Tools->Programmer 或者点击图标  打开程序下载环境，如图 2-22 所示。

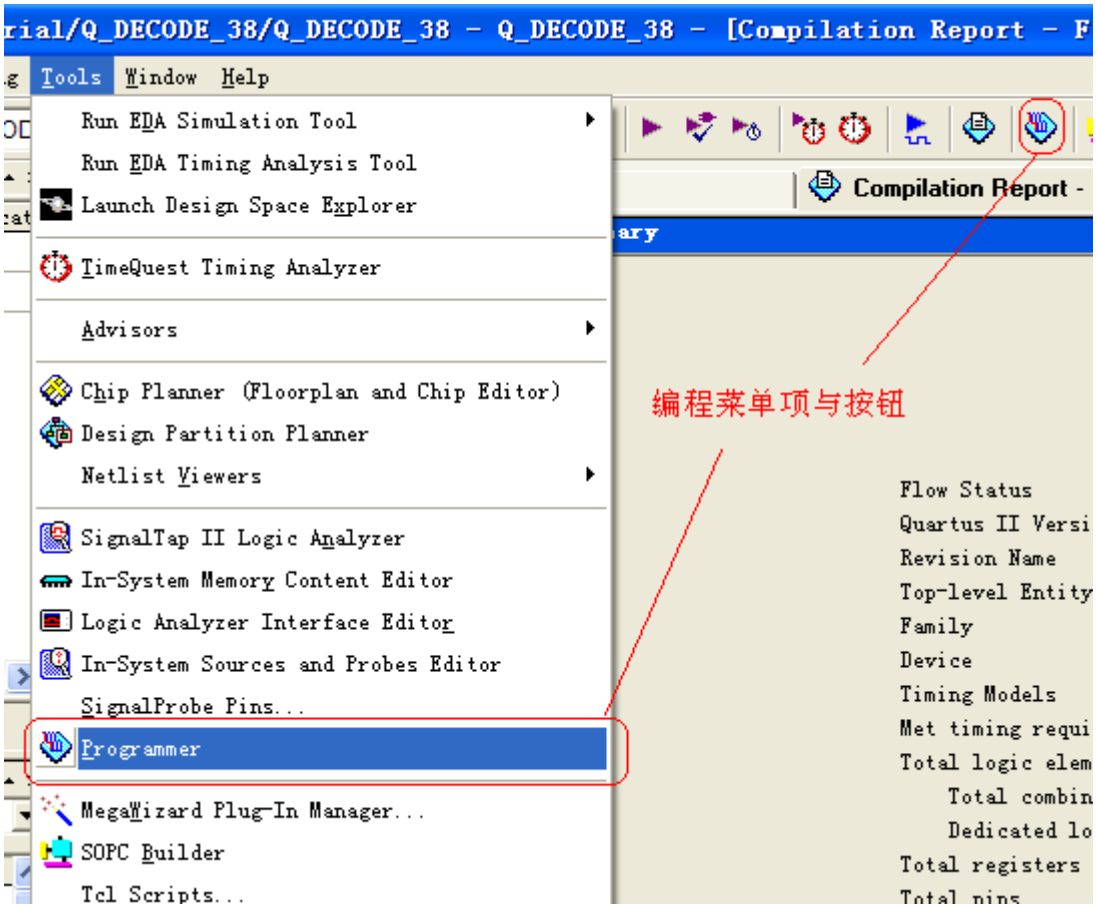


图 2-22　编程菜单项和编程按钮

18. 之后的输出画面如图 2-23 所示。

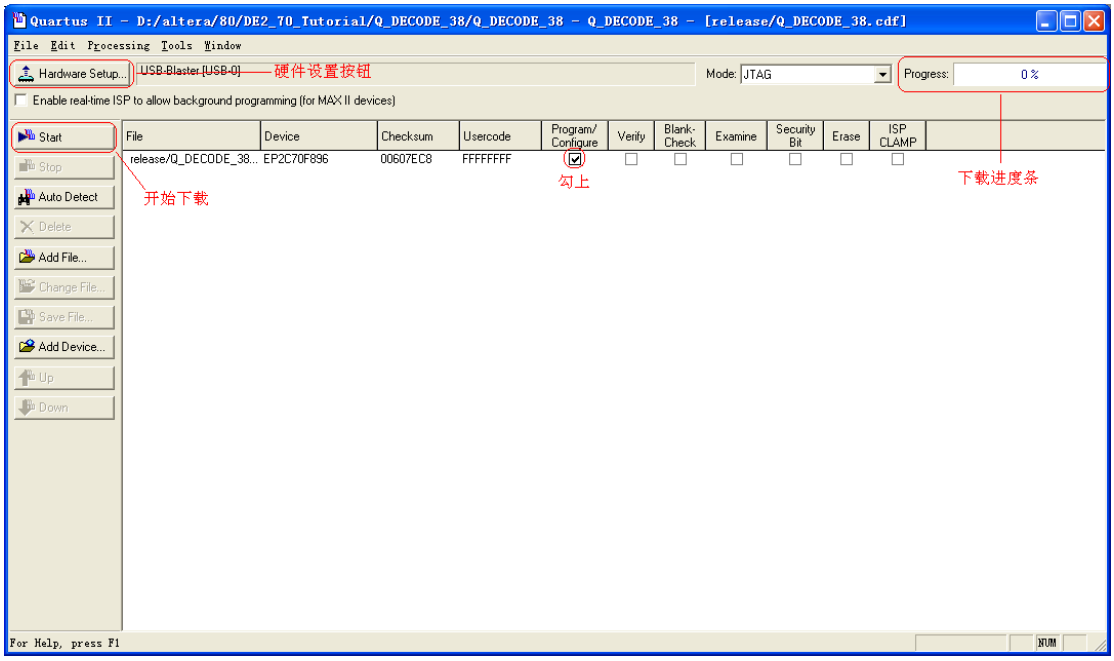


图 2-23　程序下载界面

点击 Hardware Setup 按钮下载时使用的硬件。如图 2-24 所示。

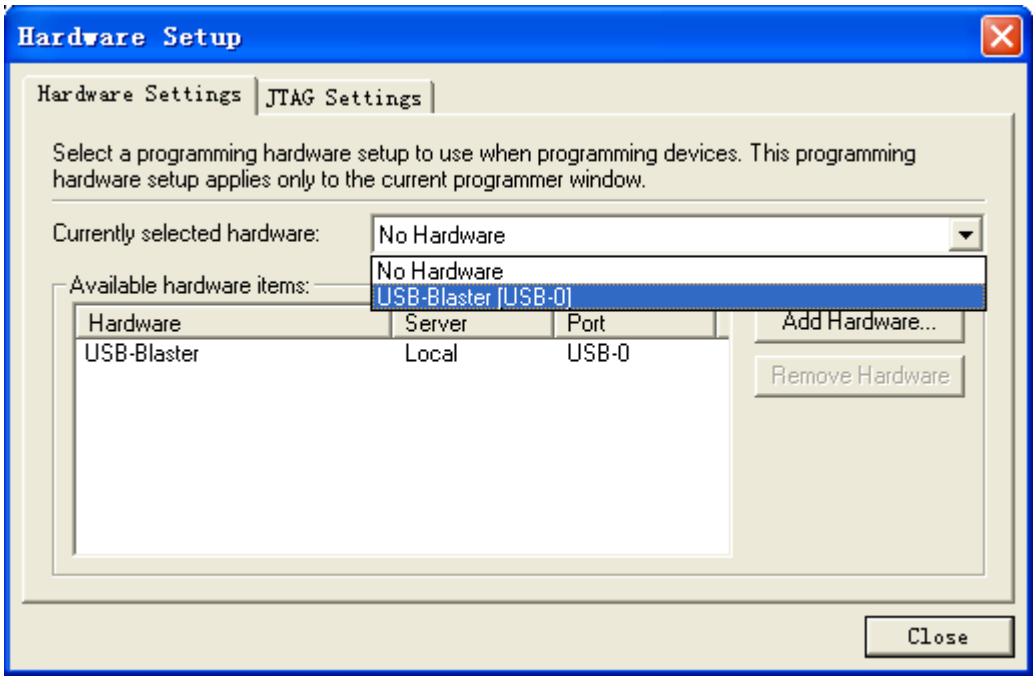


图 2-24　Hardware Setup 界面

点击 Close 确认设置。

19. 下载程序。在 Programmer 界面中，将 Q_DECODE_38.sof 文件列表中 Program/Configure 属性勾上，如图 2-25。

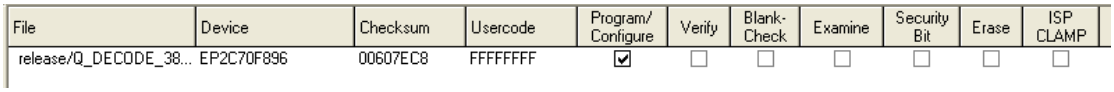


图 2-25　选择 Program/Configure 属性

再在图 2-23 点击 Start 按钮，开始下载程序。

完成后，下载程序显示为 100%，如图 2-26。

图 2-26 下载进度显示

20. 最终调试，在 DE2-70 实验板上，扳动 SW2，SW1 和 SW0 开关，可以看到译码之后的 LEDR7-LEDR0 红色 LED 发光输出。

第 3 章　实验二　十进制计数器实验

- 实验说明

该实验将使用 Verilog 硬件描述语言在 DE2-70 开发平台上设计一个基本时序逻辑电路——1 位十进制计数器。通过这个实验，读者可以了解使用 Quartus 工具设计硬件的基本流程以及使用 Quartus II 内置的工具进行仿真的基本方法和使用 SignalTap II 实际观察电路运行输出情况。SignalTap II 是 Quartus 工具的一个组件，是一个片上的逻辑分析仪，可以通过 JTAG 电缆将电路运行的实际输出传回 Quartus 进行观察，从而省去了外界逻辑分析仪时的很多麻烦。

- 实验步骤

3.1 建立工程并完成硬件描述设计

1. 打开 Quartus II 工作环境，如图 3-1 所示。

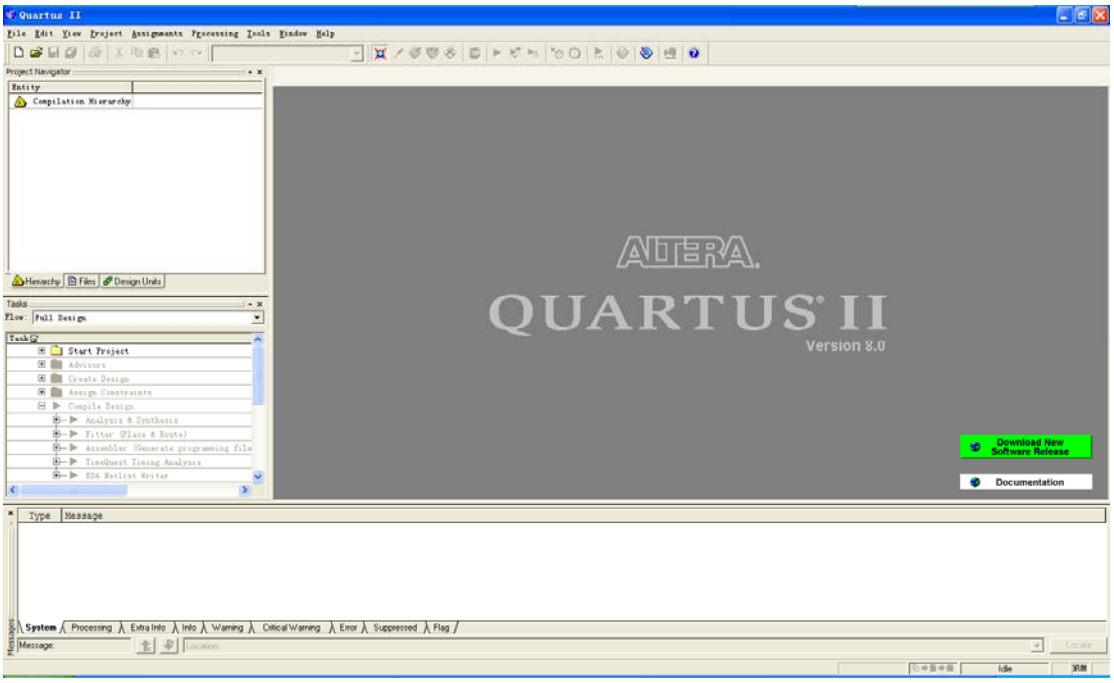


图 3-1　Quartus II 工作环境界面

2. 点击菜单项 File->New Project Wizard 帮助新建工程。参看图 3-2。

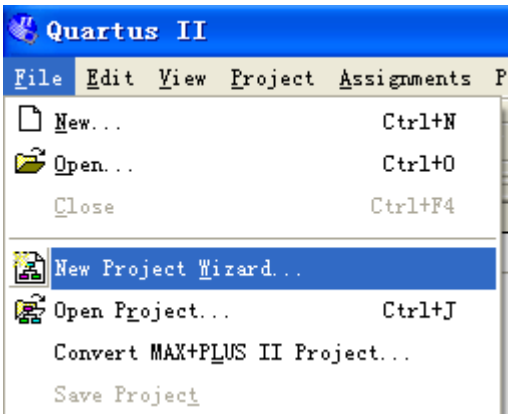


图 3-2　选择 New Project Wizard

打开 Wizard 之后，界面如图 3-3 所示。点击 Next，如图 3-3。

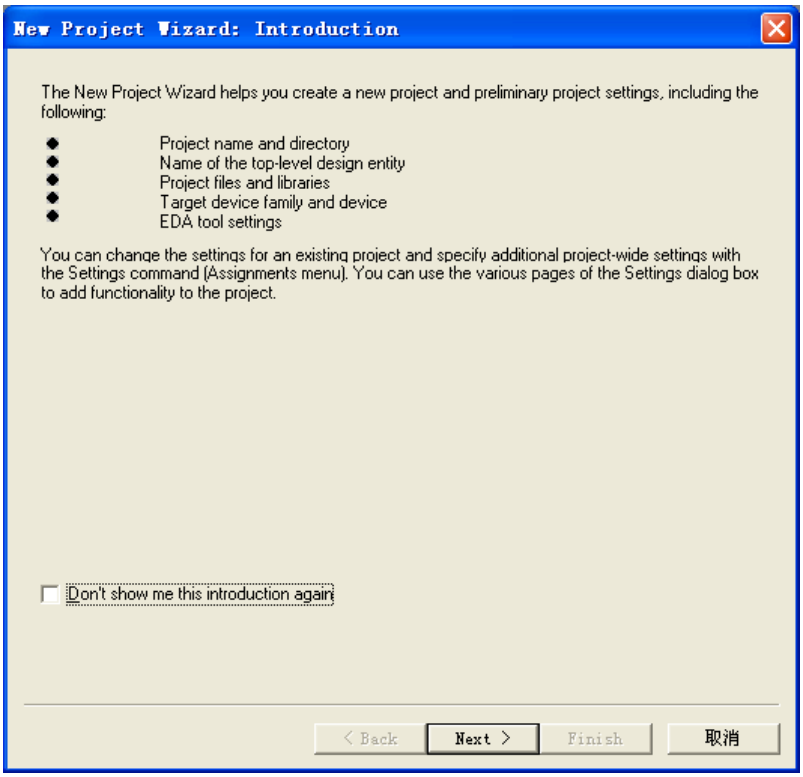


图 3-3 New Project Wizard 界面

3.　输入工程工作路径、工程文件名以及顶层实体名。

这次实验会帮助读者理解顶层实体名和工程名的关系，记住目前指定的工程名与顶层实体名都是 Counter10，输入结束后，如图 3-4 所示。点击 Next。

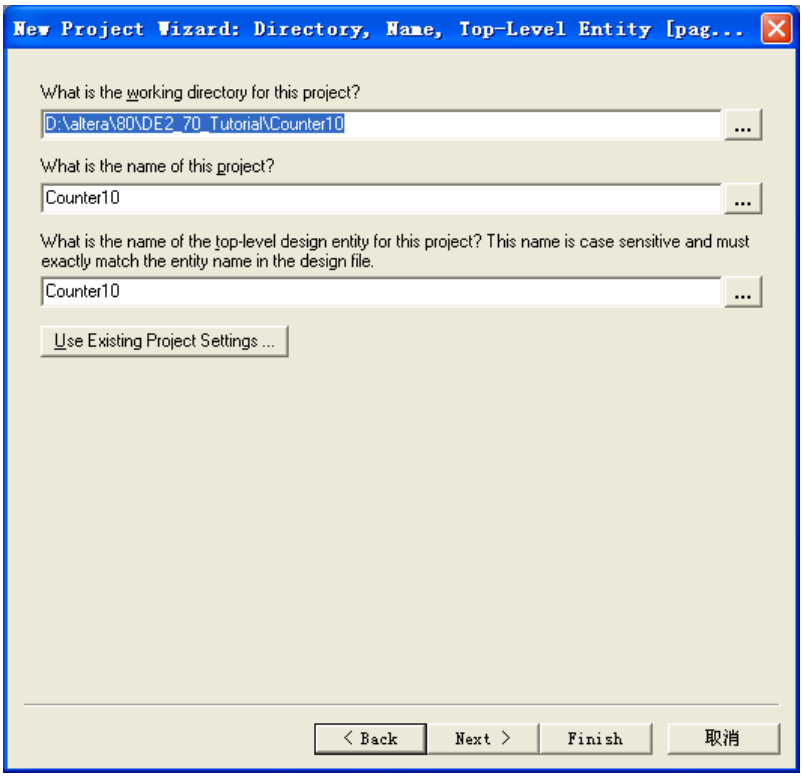


图 3-4 输入设计工程信息

4.　添加设计文件。界面如图 3-5 所示。如果用户之前已经有设计文件（比如.v 文件）。那么再次添加相应文件，如果没有完成的设计文件，点击 Next 之后添加并且编辑设计文件。

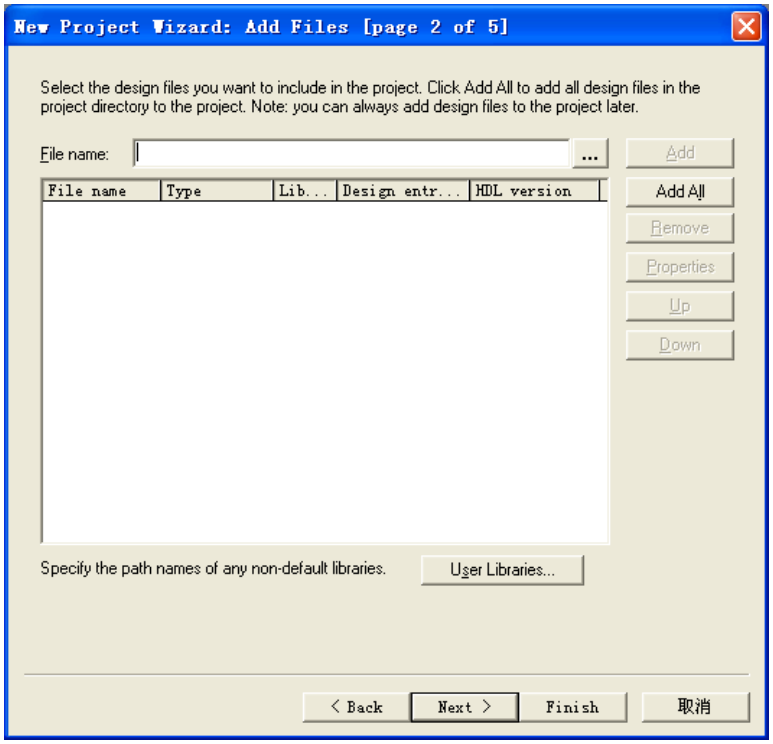


图 3-5 添加设计文件

5.　选择设计所用器件。由于本次实验使用 Altera 公司提供的 DE2-70 开发板，用户必须选择与 DE2-70 开发板相对应的 FPGA 器件型号。

在 Family 菜单中选择 Cyclone II，Package 选 FBGA，Pin Count 选 896，Speed grade 选 6，确认 Available devices 中选中 EP2C70F896C6，如图 3-6。

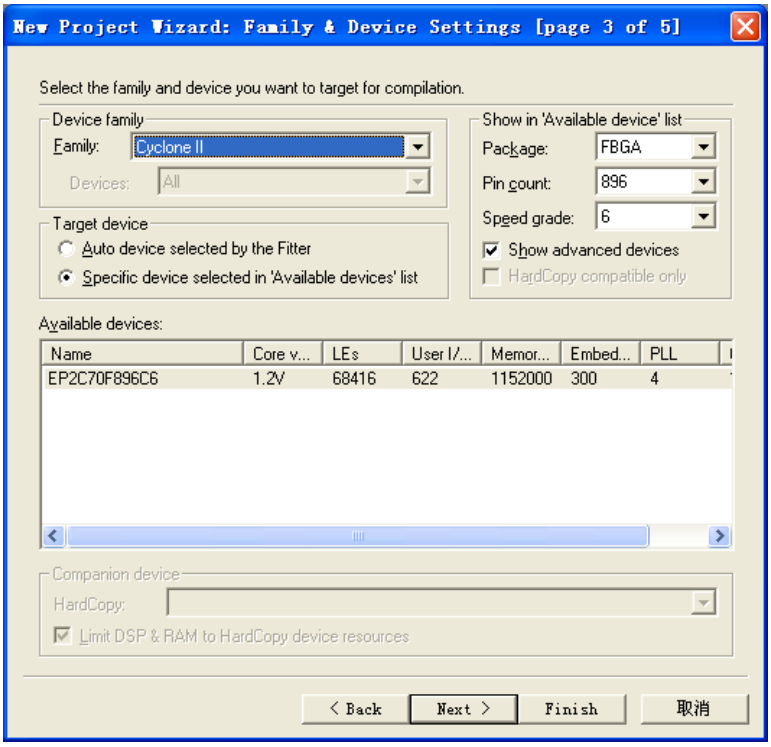


图 3-6 选择相应器件

6.　设置 EDA 工具。设计中可能会用到的 EDA 工具有综合工具、仿真工具以及时序分析工具。本次实验中不使用这些工具，因此点击 Next 直接跳过设置。如图 3-7。

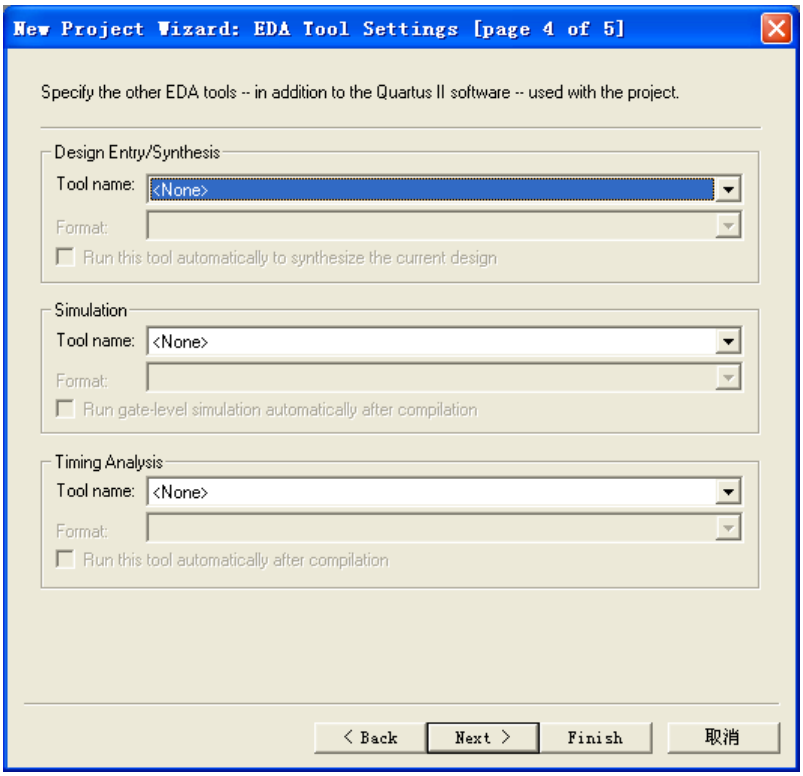


图 3-7 设置 EDA 工具

7.　查看新建工程总结。在基本设计完成后，Quartus II 会自动生成一个总结让用户核对之前的设计，如图 3-8 所示，确认后点击 Finish 完成新建。

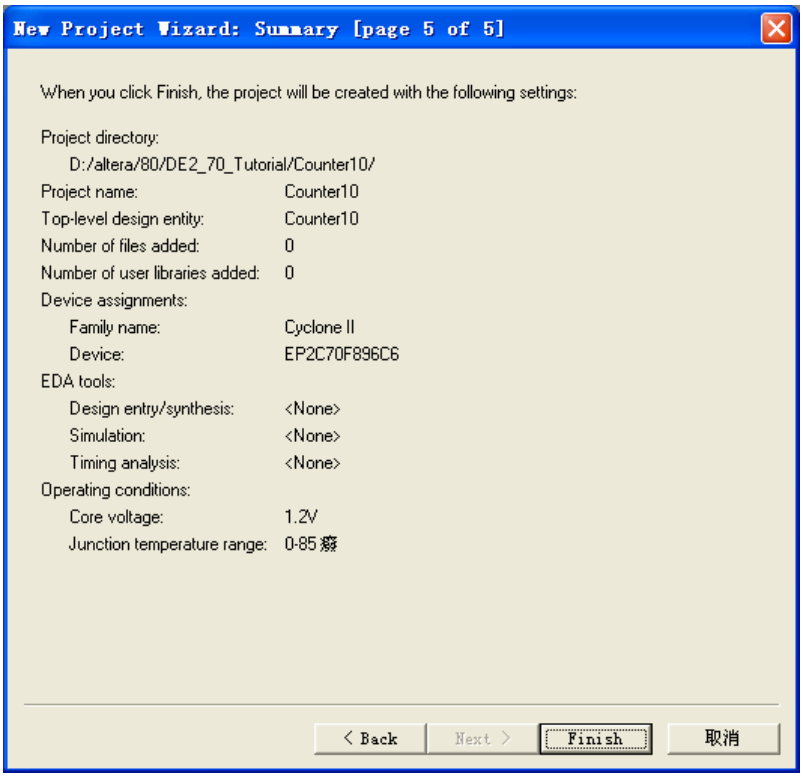


图 3-8 新建工程总结

在完成新建后，Quartus II 界面中 Project Navigator 的 Hierarchy 标签栏中会出现用户正在设计的工程名以及所选用的器件型号，如图 3-9 所示。

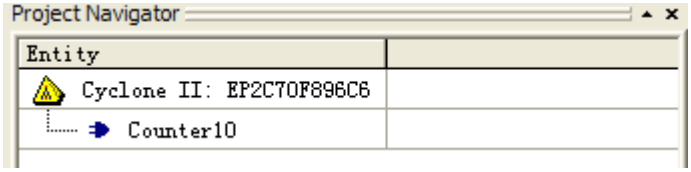


图 3-9 观察正在设计的工程

8. 培养良好的文件布局。

点击菜单项 Assignments->Device，选中 Compilation Process Settings 选项卡，勾上右边的 Save Project output files in specified directory，输入路径(一般为 debug 或者 release)，如图 3-10 所示。

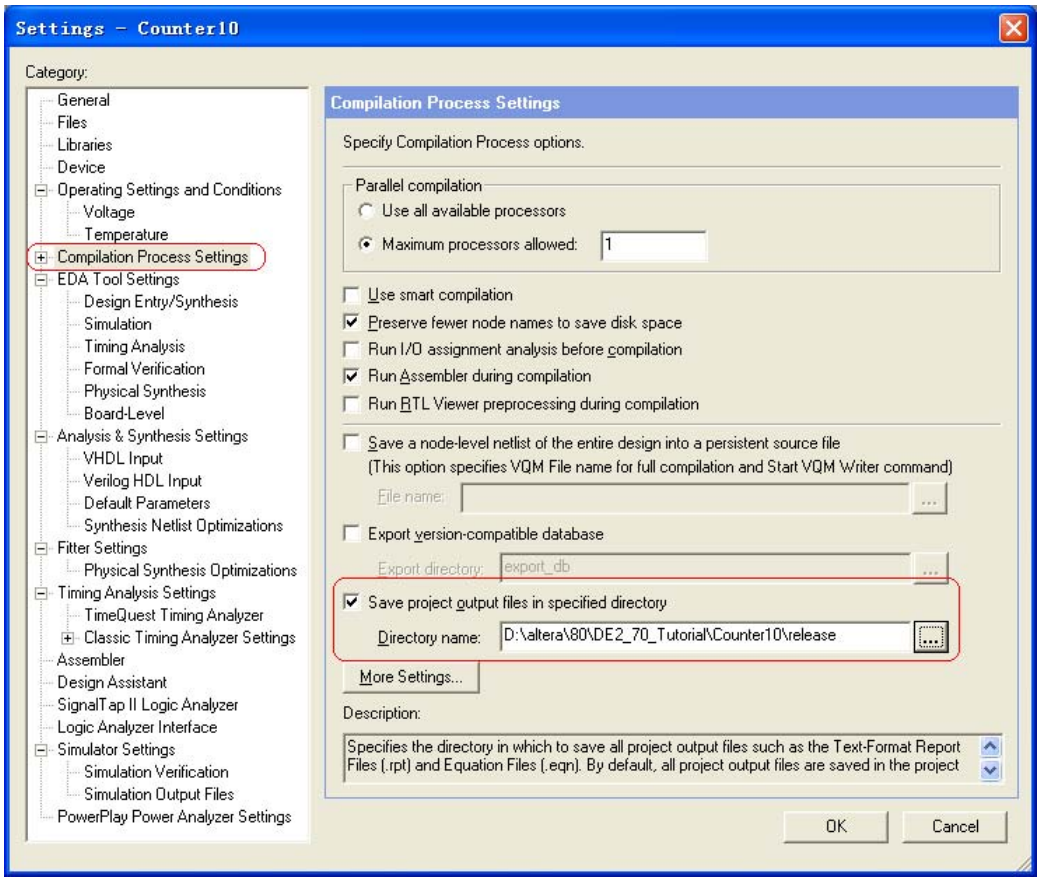


图 3-10　指定单独的编译结果文件目录

9. 添加所需设计文件。

点击菜单项 File->New 或者点击图标 新建一个设计文件，选择 Verilog HDL File，如图 3-11 所示，点击 OK。建立 Verilog 源代码文件。

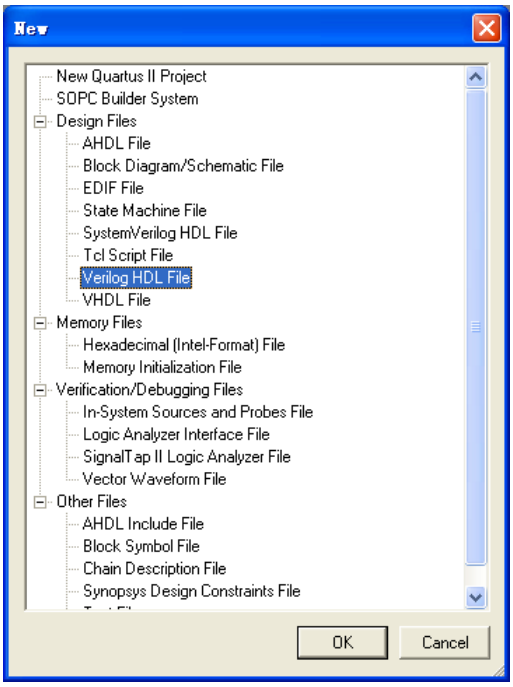


图 3-11　选择设计文件类型

输入如下设计代码：

```
module Counter
(
iclk，
rst_n，
q，
overflow
);
input iclk;
input rst_n;
output reg [3:0] q;
output overflow;

always @(posedge iclk or negedge rst_n)
begin
    if(~rst_n) q <= 4'h0;
    else
    begin
        if(4'h9 == q) q <= 4'h0;
        else q <= q + 4'h1;
    end
end

assign overflow = 4'h9 == q;

endmodule
```

10. 保存设计。点击菜单项 File->Save、点击图标  或者使用快捷键 Ctrl+S 保存设计，如图 3 -12 所示。给设计文件命名 Counter，与模块名相同，注意不是 Counter10，点击保存。

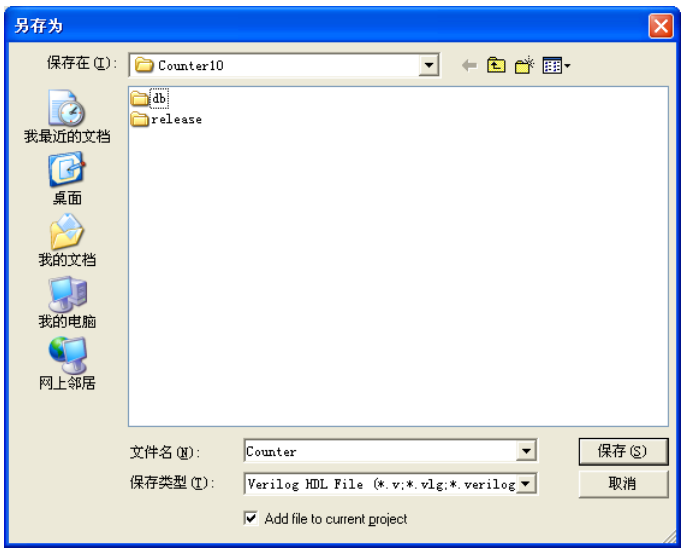


图 3-12　保存设计文件

11. 分析与综合。点击菜单项 Processing->start->Start Analysis & Synthesis、点击图标  或者使用快捷键 Ctrl+K 执行分析与综合。参看图 3-13。

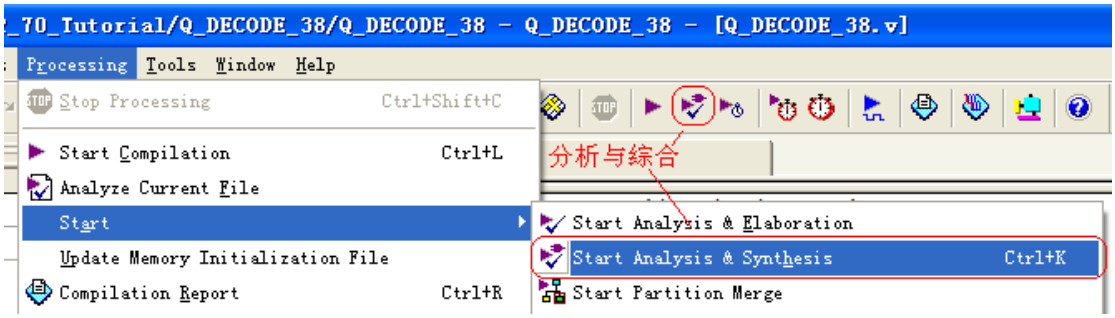


图 3-13　执行 start Analysis & Synthesis（开始分析与综合）

分析与综合完成后，编译出错，错误原因如图 3-14 所示。

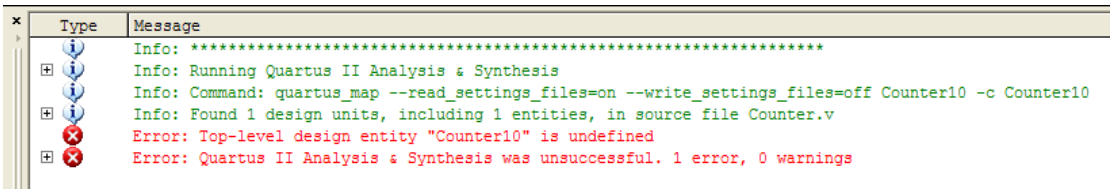


图 3-14　分析与综合错误原因

顶层实体 Counter10 未在源码中定义，必须更改顶层实体为 Counter，这在多文件的工程中经常需要用到。

将左侧的 Project Navigator 切到 Files 标签，对着 Counter.v 文件右击，选择 Set as Top-Level Entity，如图 3-15。

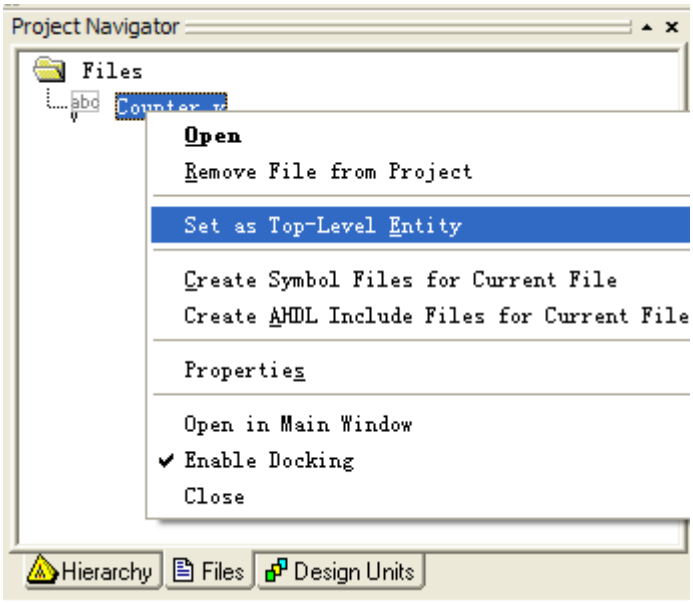


图 3-15　重新指定顶层实体

12. 重新执行分析与综合，结果如图 3-16，出现了 12 个警告，这是因为 qsf 文件中记录的顶层实体在这一步执行时还未更新。

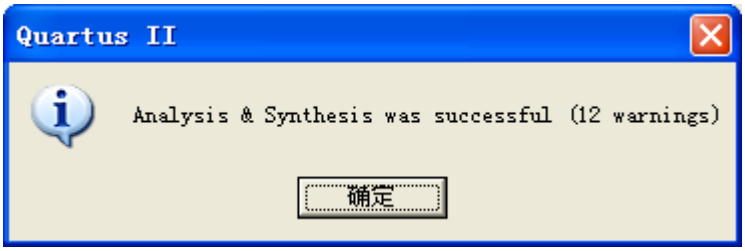


图 3-16　分析与综合结果(第二次执行)

如果再次执行分析与综合，无论你是否删掉原先的编译结果，都会完全成功，如图 3-17。

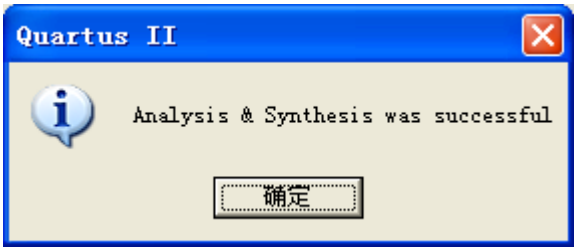


图 3-17　分析与综合结果(第三次执行)

3.2 电路仿真

13. 功能仿真。它是为了检查设计是否在理论上达到预期功能，该仿真不考虑期间实际物理特性。首先创建仿真输入波形文件。仿真时需要对顶层实体的输入管脚提供激励信号，在 Quartus 软件中可以通过波形文件方便的输入。点击菜单项 File->New->Vector Waveform File，如图 3-18 所示。

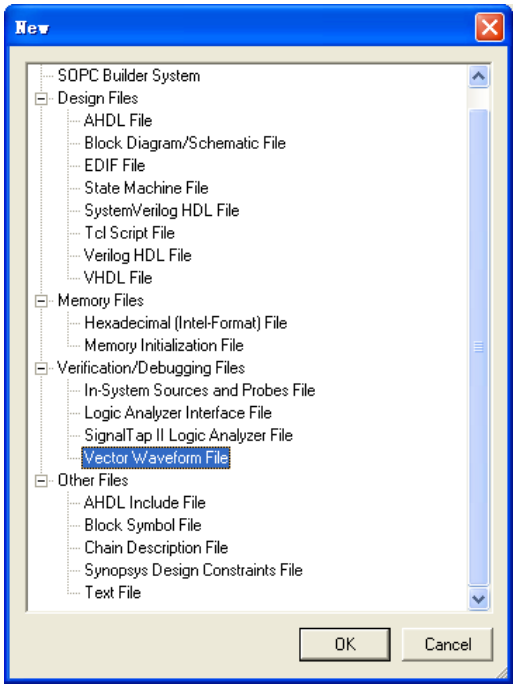


图 3-18　创建波形文件

14. 添加信号结点。在空波形文件中点击右键，如图 3-19 进行选择（或者直接双击）。

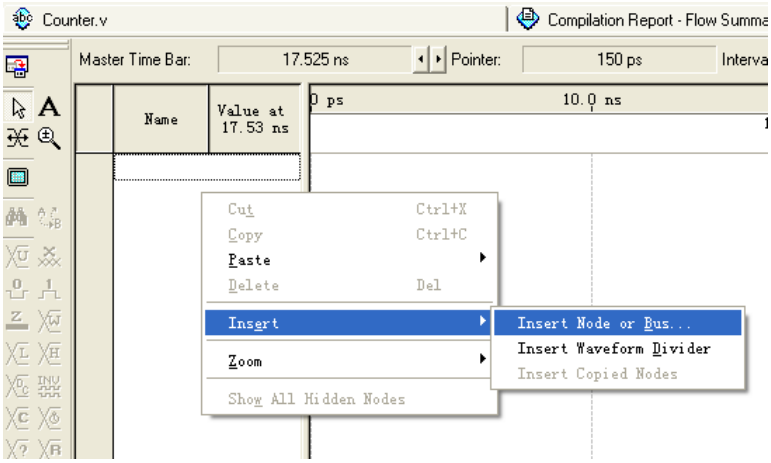


图 3-19　添加结点右键菜单

单击 Insert Node or Bus 后，出现如图 3-20 所示对话框。

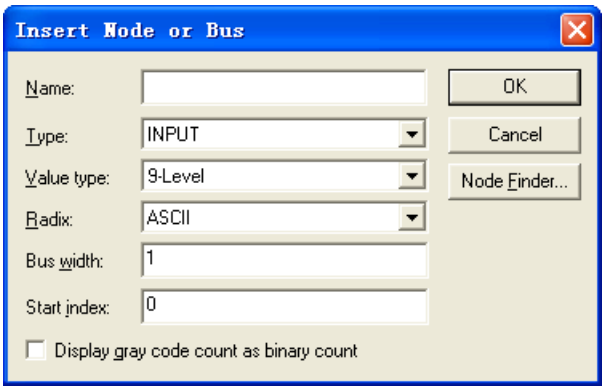


图 3-20　添加结点对话框

选择 Node Finder 按钮可以从结点列表中选择我们需要的，而避免一个一个输入结点

的麻烦。

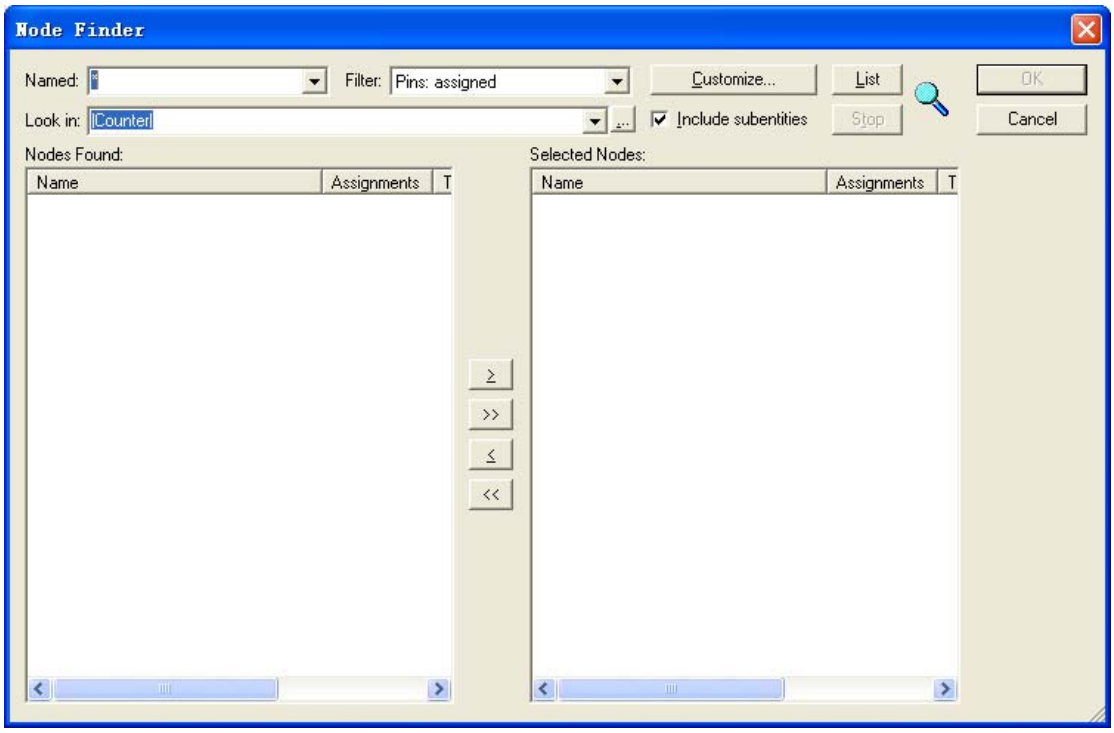


图 3-21　Node Finder 对话框

Fitter 选择 Pin:all，点击 List 按钮。出现如图 3-22 所示的结点列表。

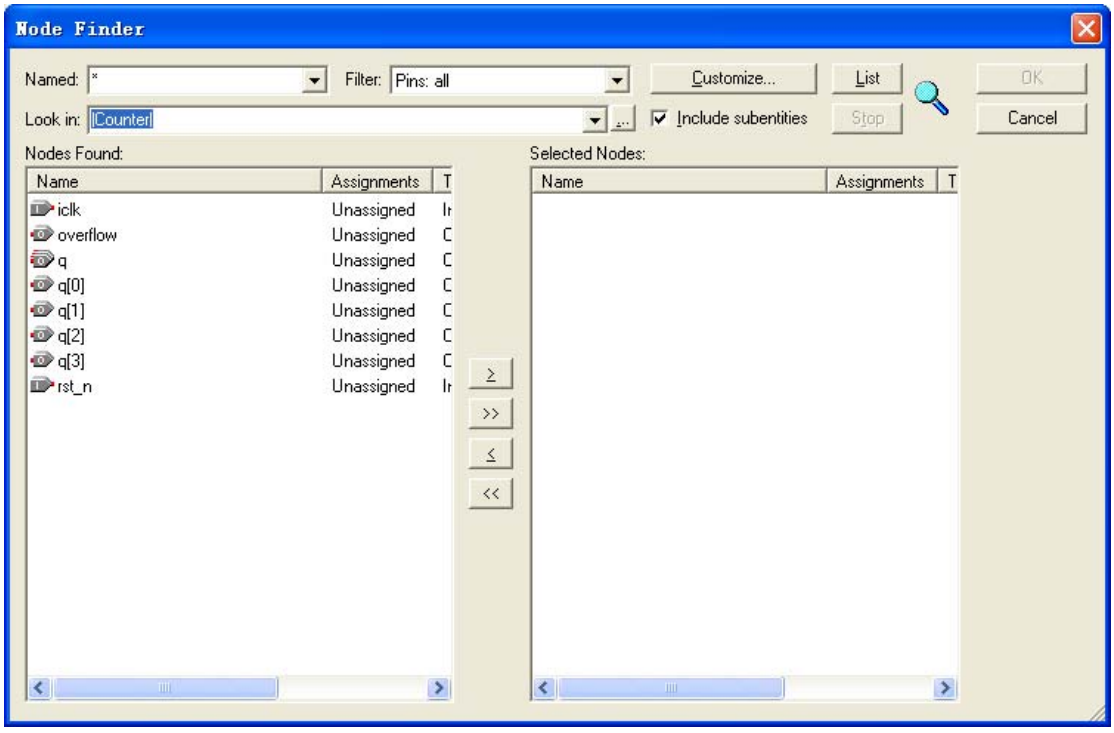


图 3-22　结点列表

简单起见，可以直接点>>按钮，将所有结点加入右侧 Select Nodes 栏中。完成后如图 3-23 所示。点击 OK 按钮确认。

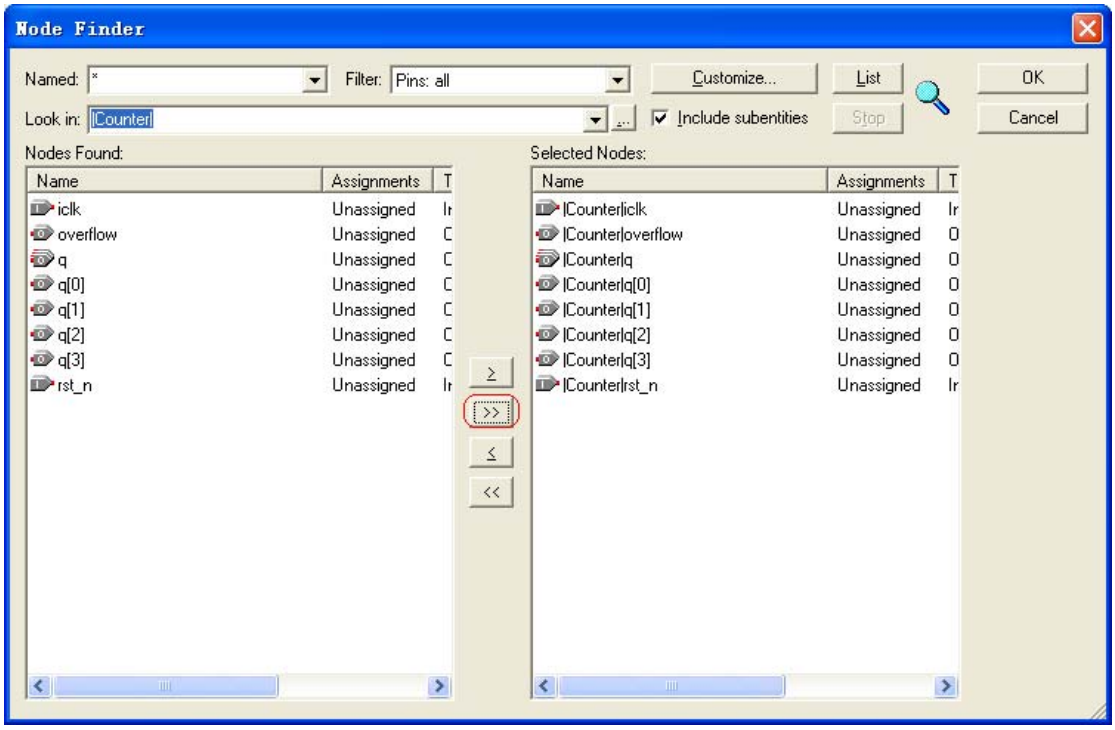


图 3-23　添加结点到右侧

点击 OK 后返回添加结点对话框。如图 3-24 所示。

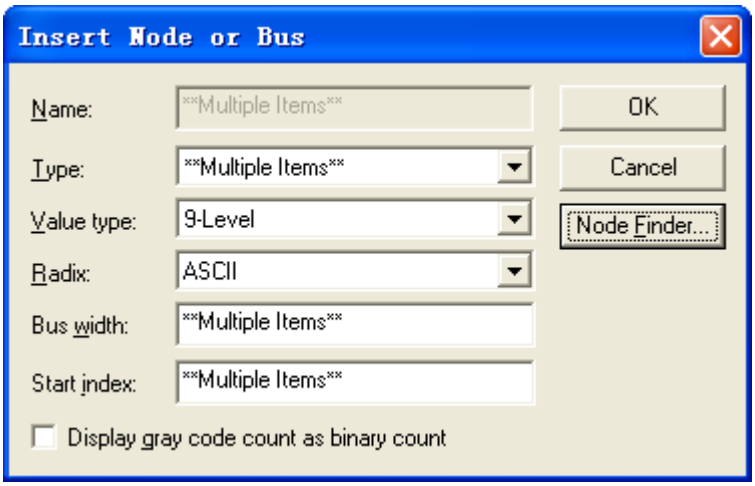


图 3-24　添加结点后的对话框

点击 OK 确定，波形文件将如图 3-25 所示。

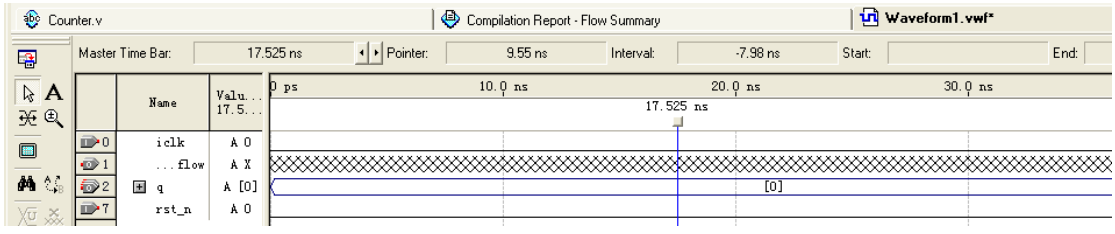


图 3-25　波形文件

15. 将 iclk 设为方波。右击 iclk 信号，选择 value->clock..，如图 3-26 所示。

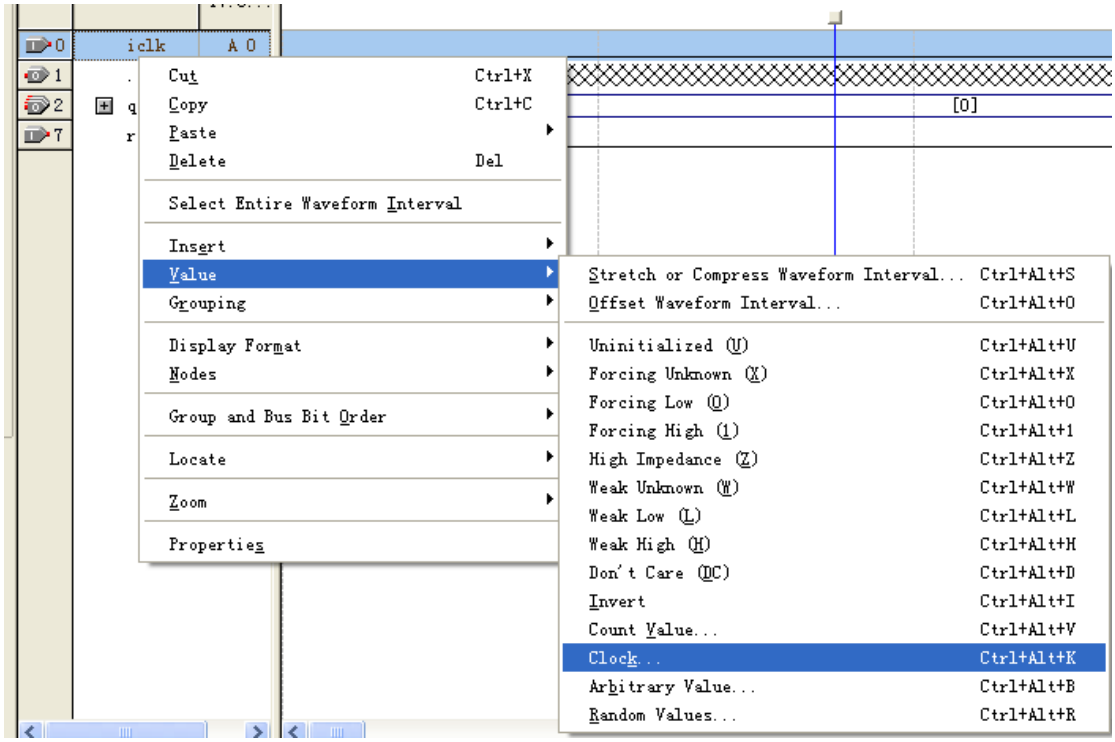


图 3-26　将 iclk 改为方波

在弹出的 clock 设定对话框中把周期调整为 20ns，如图 3-27。Duty cycle 的意思是占空比，即是指高电平在一个周期之内所占的时间比率。

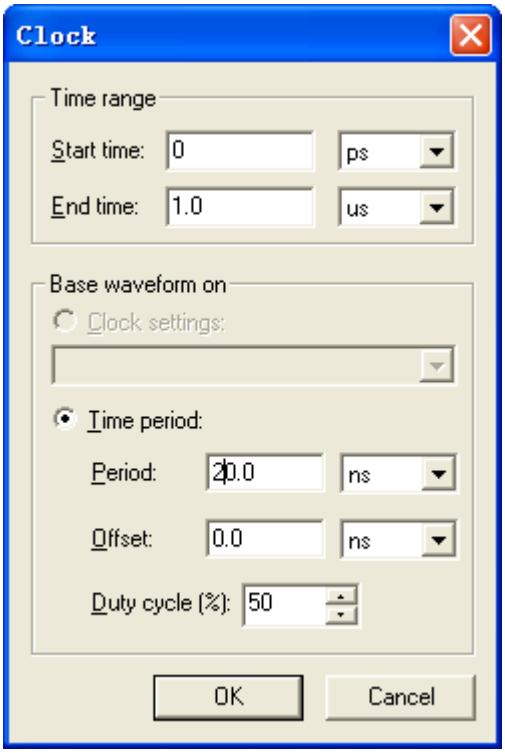


图 3-27　时钟的周期设置

16. 将 rst_n 改成低 20ns 后持续高电平。选中 rst_n 信号，单击左侧图标  强制设为高电平。在波形上拖动鼠标选中前 20ns，单击左侧图标  强制设为低电平。

完成后波形如图 3-28 所示。输出波形可不管。

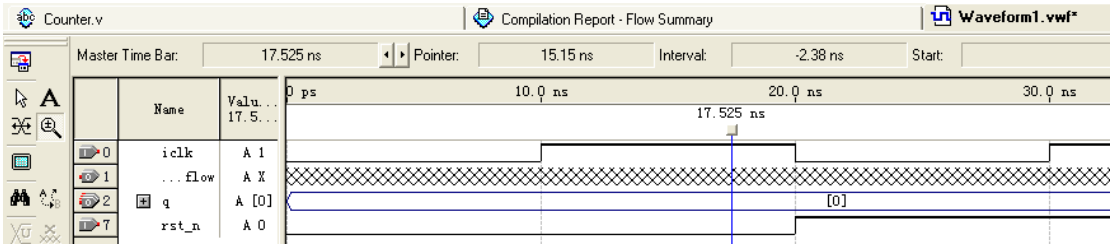


图 3-28 波形文件

17. 保存波形文件 counter.vwf，如图 3-29，这里的命名可以随意。

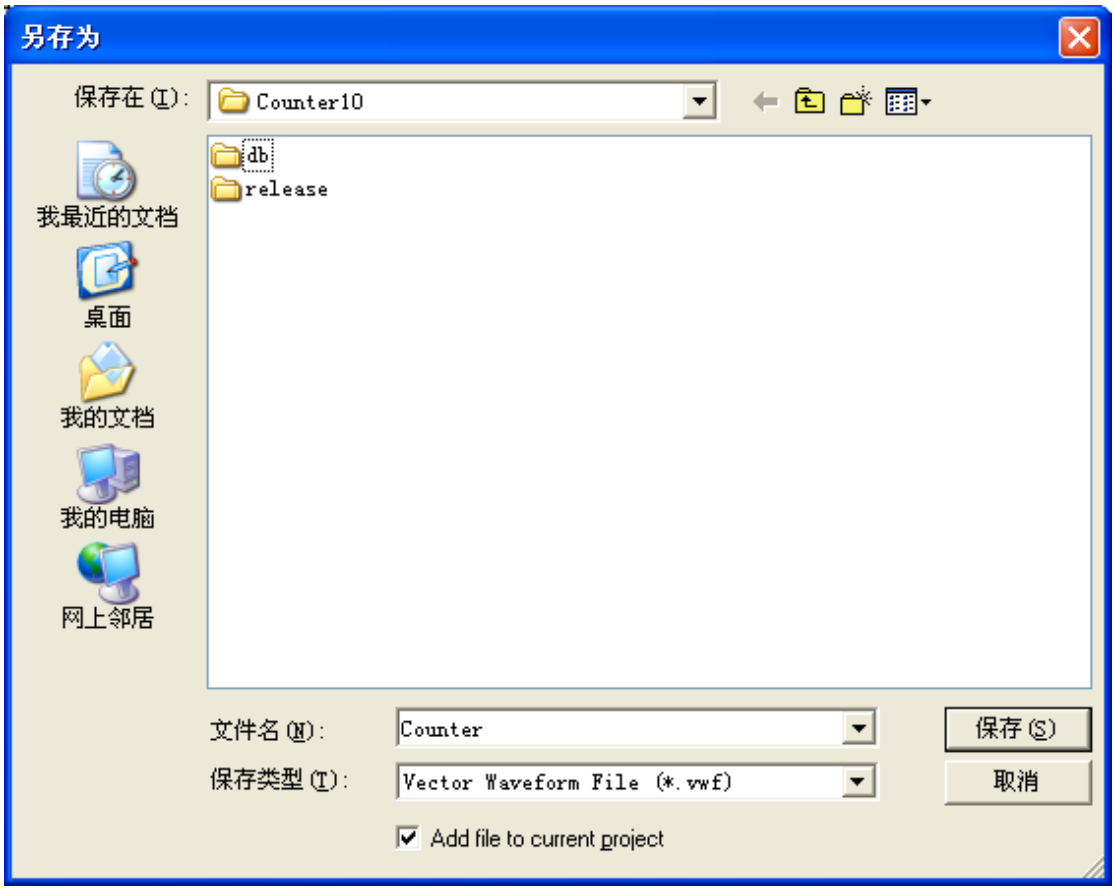


图 3-29 保存波形文件

18. 波形文件生成后，直接点击仿真按钮会提示错误，见图 3-30，这是因为没有先产生功能仿真网表。

图 3-30 未生成网表错误

19. 要生成功能仿真网表，首先设置仿真模式。点击菜单项 Assignment->Settings，选中 Simulator Settings 选项卡，出现图 3-31 所示对话框。在 Simulation mode 中选择 Functional，Simulation input 选择刚才建立的波形文件，完成后点击 OK。

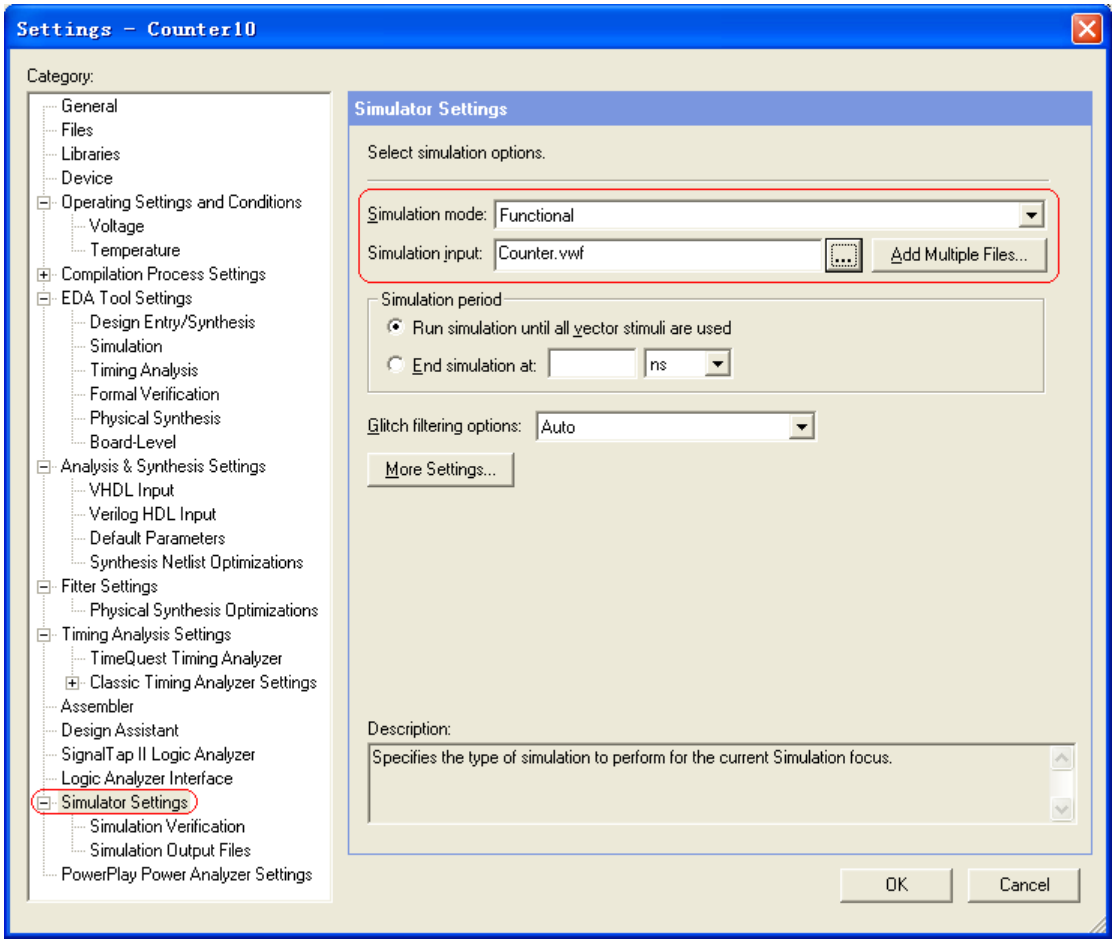


图 3-31 仿真模式设置对话框

点击菜单项 Processing->Generate Functional Simulation Netlist，产生功能仿真所需的网表，参看图 3-32。完成后结果显示如图 3-33。

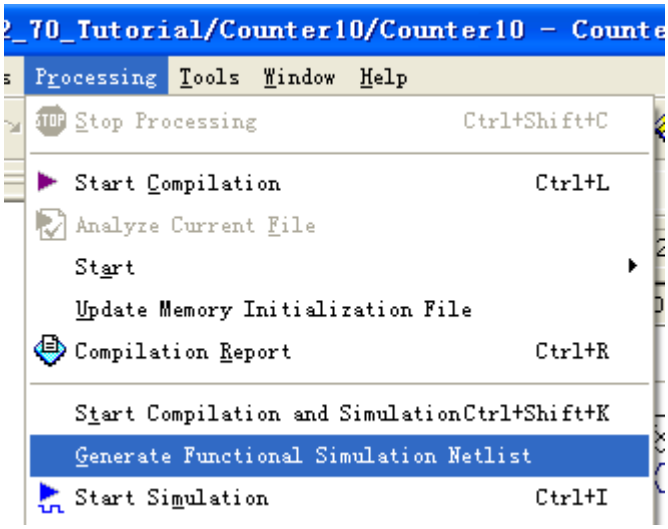


图 3-32　生成功能仿真网表的操作菜单项

图 3-33　功能仿真网表产生结果显示图

20. 点击菜单项 Processing->Start Simulation 或 工具按钮启动功能仿真。如图 3-34，完成后结果显示如图 3-35。

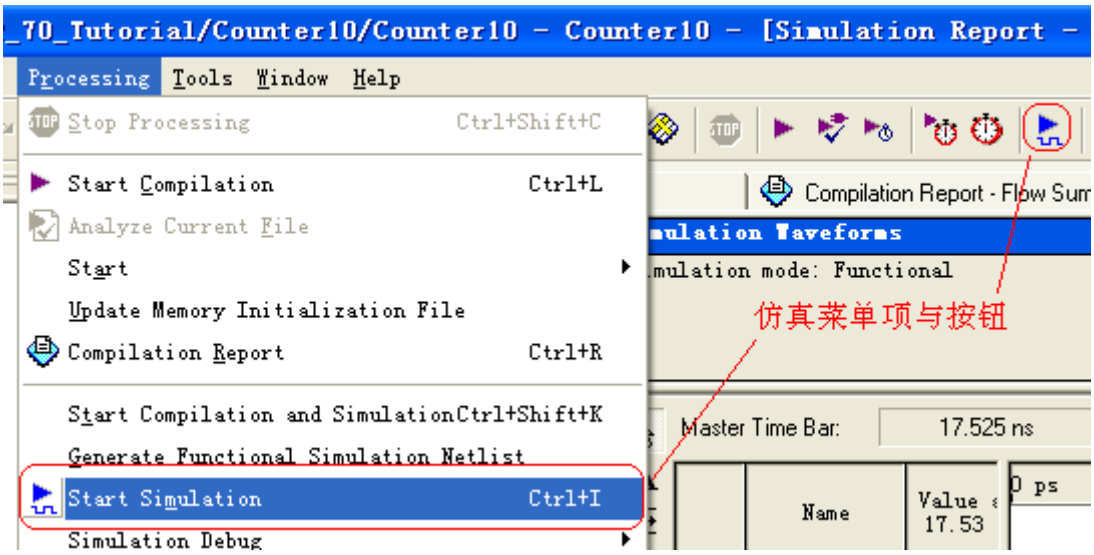


图 3-34　仿真菜单项与按钮

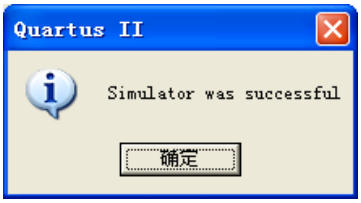
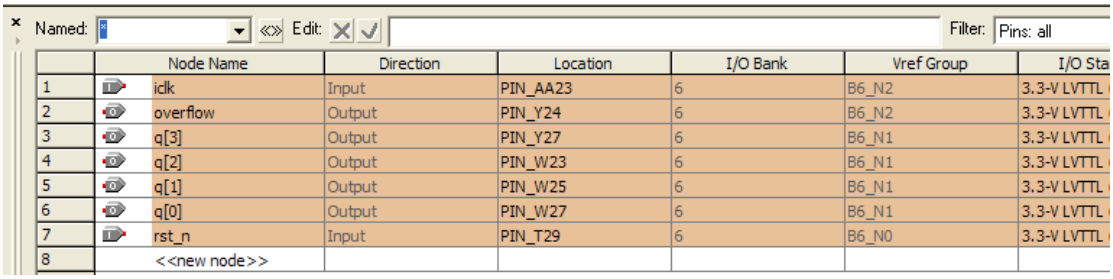


图 3-35　仿真结果

21. 配置引脚。仿真完成后，确认功能正确后，可以进行分配引脚的操作。根据所提供的 DE2-70 用户指导手册，将计数器的 q 输出配置到 DE2-70 开发板的 4 个绿 LED（LEDG[3]-LEDG[0]）上，overflow 接 LEDG[4]，rst_n 接 KEY[0]，clk 接开关 SW[0]。（参考实验一）参考图 3-36，注意 Y24 不是 V24。



	Node Name	Direction	Location	I/O Bank	Vref Group	I/O Sta
1	iclk	Input	PIN_AA23	6	B6_N2	3.3-V LVTTL
2	overflow	Output	PIN_Y24	6	B6_N2	3.3-V LVTTL
3	q[3]	Output	PIN_Y27	6	B6_N1	3.3-V LVTTL
4	q[2]	Output	PIN_W23	6	B6_N1	3.3-V LVTTL
5	q[1]	Output	PIN_W25	6	B6_N1	3.3-V LVTTL
6	q[0]	Output	PIN_W27	6	B6_N1	3.3-V LVTTL
7	rst_n	Input	PIN_T29	6	B6_N0	3.3-V LVTTL
8	<<new node>>					

图 3-36 分配引脚图

注意：clock 相关：DE2_70 开发板没有办法直接输出低频方波，使用开关手动控制。

22. 完成引脚分配后，全编译文件。点击菜单项 Processing->start compilation、点击图标或使用 CTRL+L 执行全编译，如图 3-37 所示。

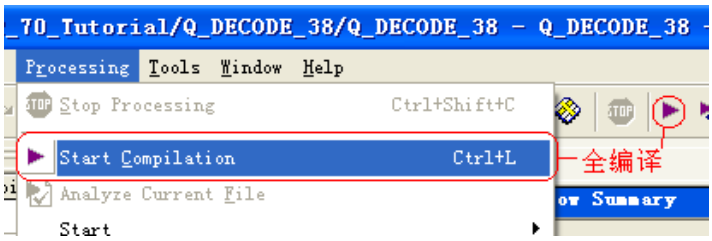


图 3-37　执行 start compilation

编译结果如图 3-38 所示。

图 3-38　全编译结果显示

23. 时序仿真。其主要用途是查看实际设计的电路运行时是否满足延时要求，时序仿真考虑了电路实际运行的延时等因素。

单击菜单中 Assignment->Settings，选中 Simulator Settings 选项卡，在 Simulation mode 中选择 Timing，Simulation input 选择刚才建立的波形文件，完成后点击 OK，如图 3-39。

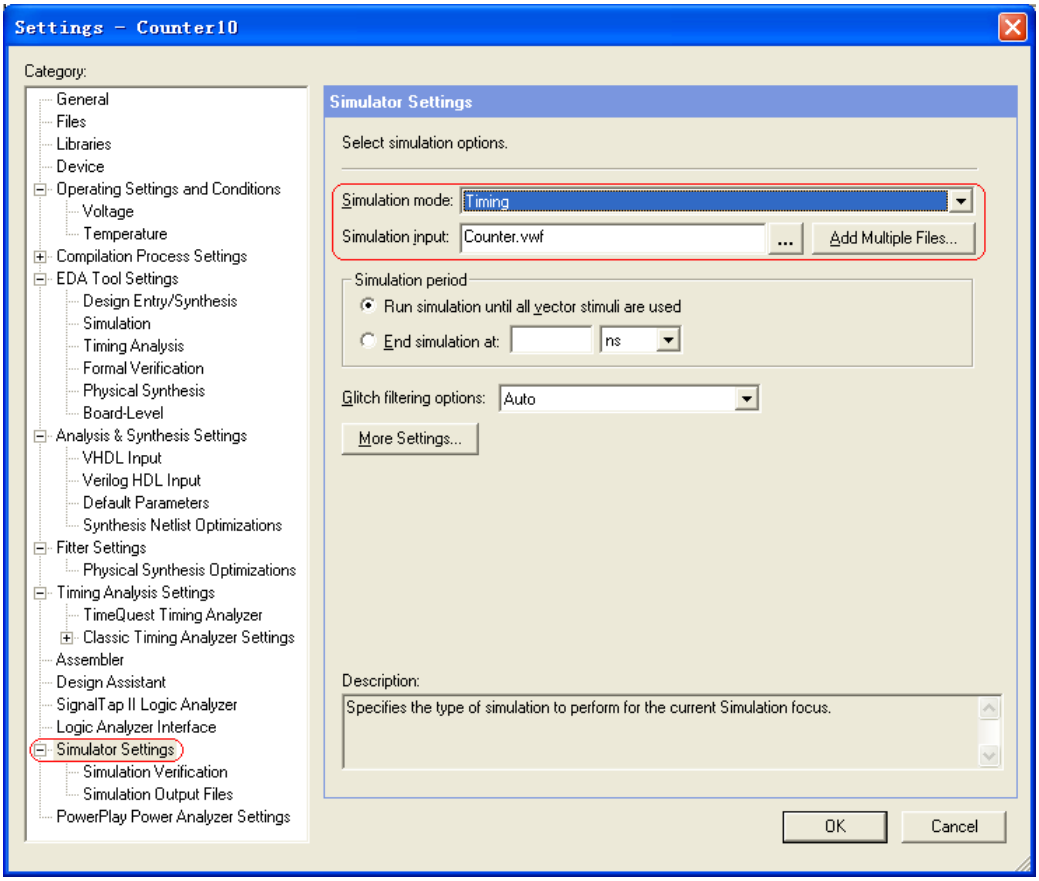


图 3-39 仿真模式设置对话框

如果是 8.0 版，在左侧带问号的 Quartus II Simulator (Timing)处右击 start，启用时序仿真，如图 3-40A.

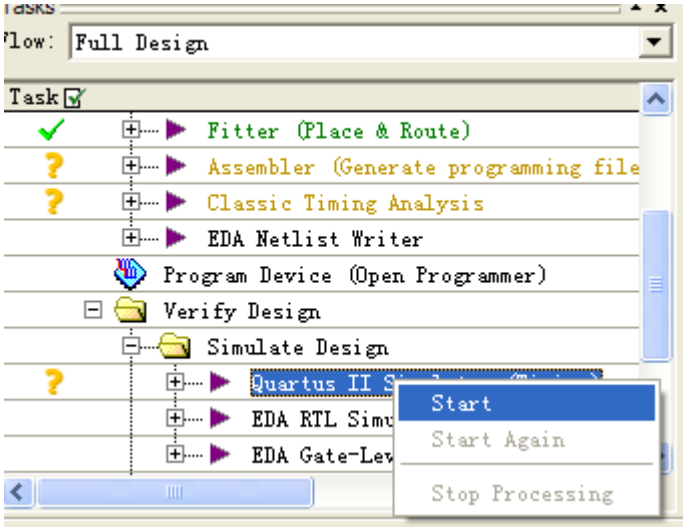


图 3-40A 启用时序仿真

如果是 7.2 版，由于没有 Tasks 窗口，需要在 Processing->Start 菜单按照 **A—E** 的步骤执行。如图 3-40B 所示。每一步骤完成会弹出一个对话框，单击 OK 或者确定。

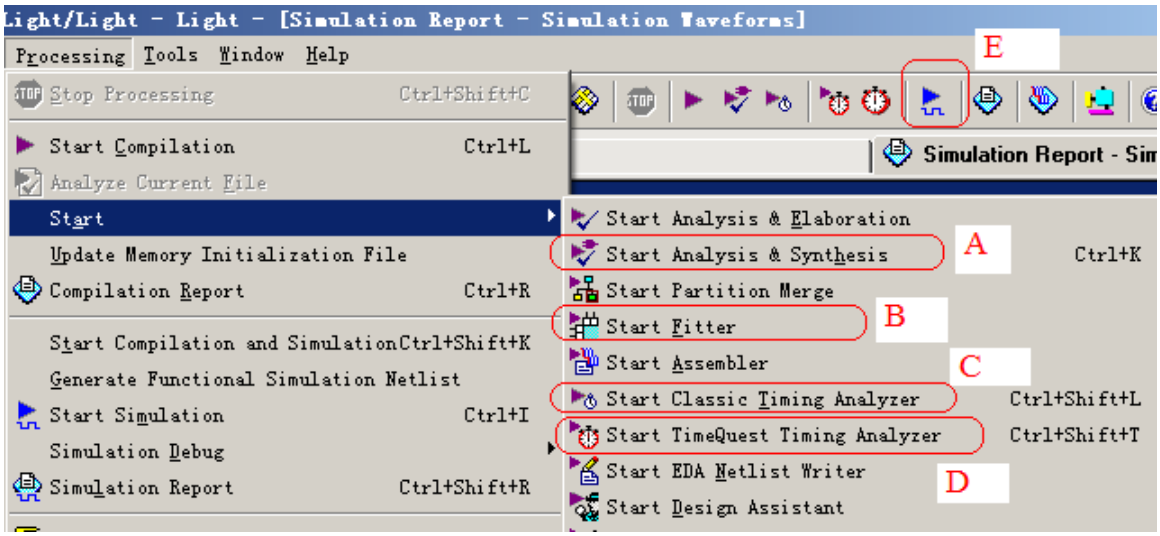


图 3-40B　时序仿真的后五步操作图解

仿真结果如图 3-41 与 3-42 所示。

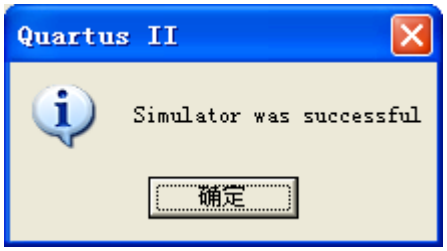


图 3-41 仿真结果图

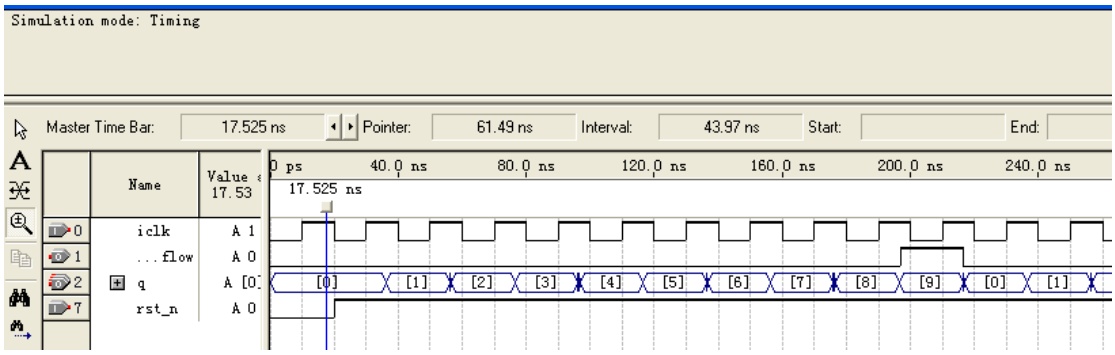


图 3-42 时序仿真波形

24. 将设计下载在 FPGA 中。完成设计后就可以下载到板上实际运行，点击菜单项 Tools->Programmer 或点击图标  打开程序下载环境。点击 start 开始下载。(参考实验一)

25. 手工拨动 SW[0]，测试实验结果。

3.3 逻辑分析仪 SignalTap II 的使用

26. **首先将手工开关时钟换回 50Mhz 的时钟，否则由于时钟过于低速，SignalTap II 抓取不到波形。**方法是在引脚配置中将 iclk 指定 AD15，之后全编译工程，并且**下载运行**！

可以看到绿灯有 5 个在亮，最左边的暗一点，如图 3-43A 所示。否则，很可能是引脚分配出错，如图 3-43B 中出现了 Y27 设成了 V27 的错误。

图 3-43A　5 个灯都亮，正确。

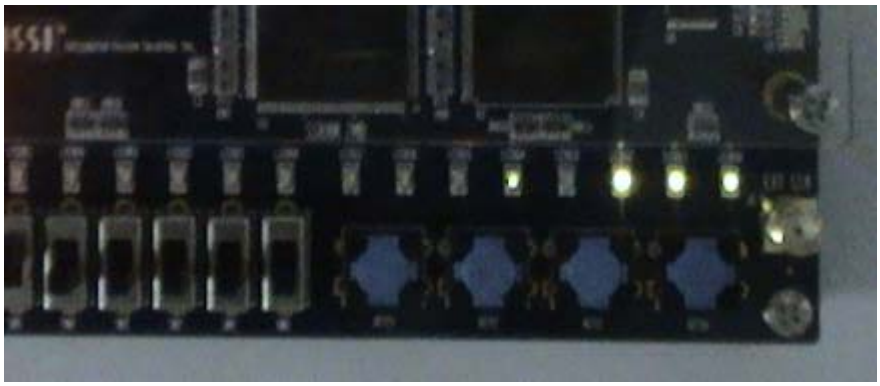


图 3-43B　只有 4 个灯亮，错误。

27. 新建 SignalTap II 文件。点击菜单项 File->SignalTap II Logic Analyzer File

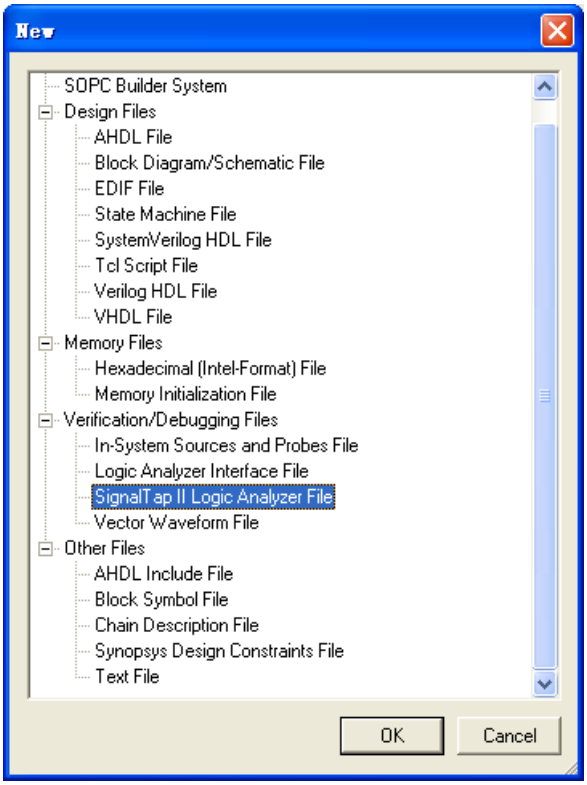


图 3-44　新建逻辑分析仪文件

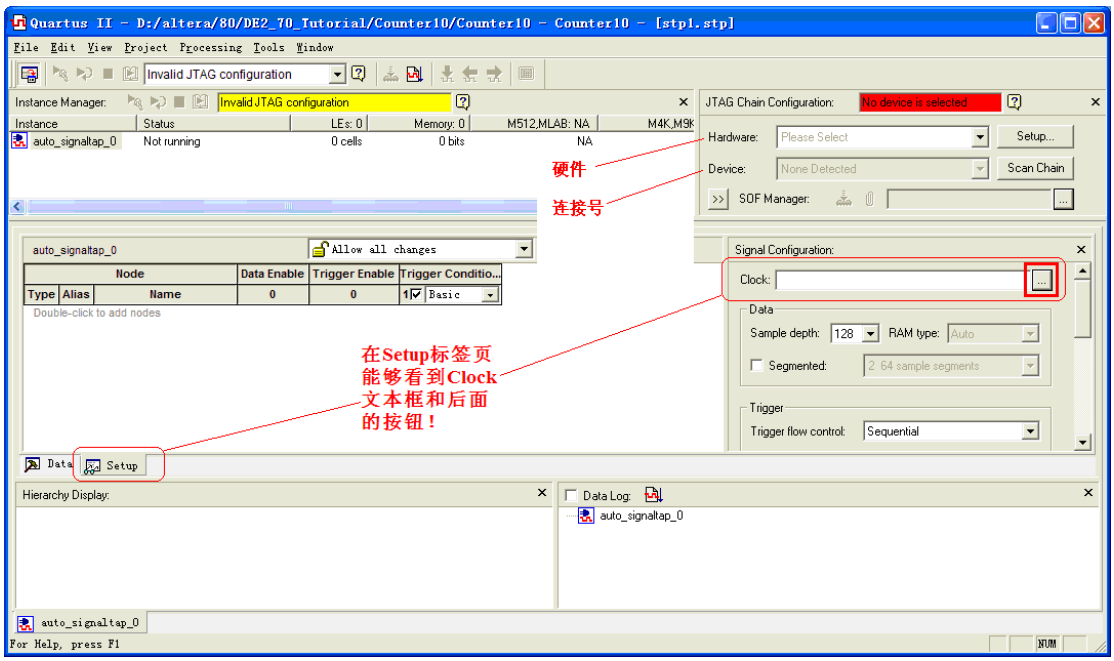


图 3-45　逻辑分析仪文件

由于窗口界面面积较小，可以通过文件左上角的  按钮将文件子窗口与主窗口分离。

28. 选择硬件，首先连接号 DE2-70，然后在文件右上的 Hardware 下拉菜单中选择 USB-Blaster，选好后应能自动识别出 Device 是 EP2C70。选择后的情况如图 3-46 所示。

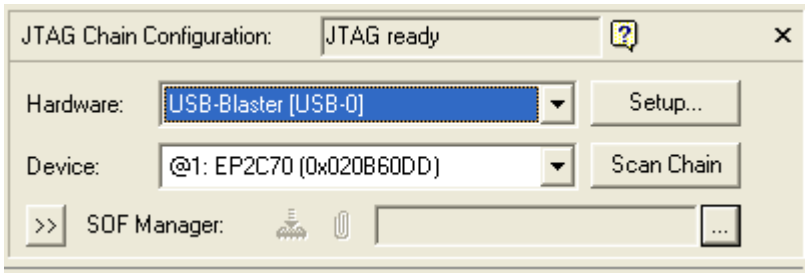


图 3-46　选择硬件环境

29. 选择逻辑分析仪时钟，本实验中就以计数器时钟作为逻辑分析仪时钟。确认左下角的标签页是 setup，然后点击右下侧 SignalConfiguration 中的 Clock 栏后的按钮。出现如图 3-47 所示。

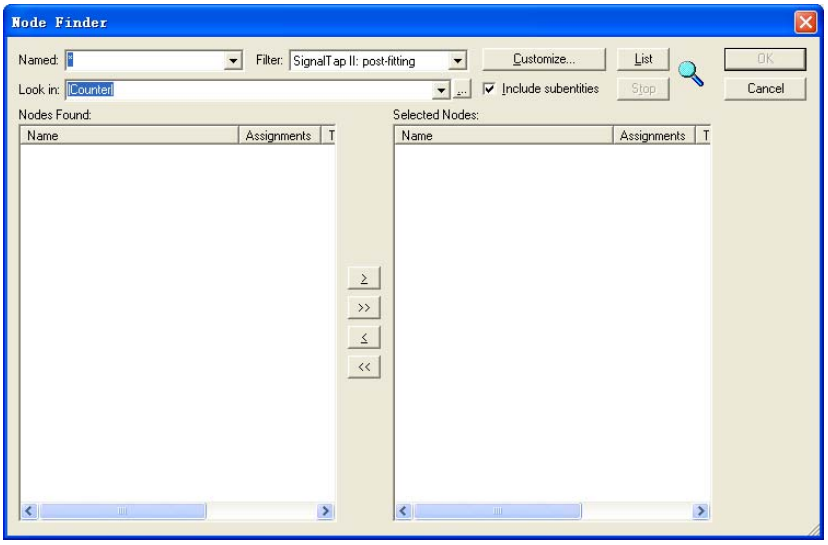


图 3-47　结点查找对话框

Fitter 选择 SignalTap II：post-fitting，点击 List 按钮，左侧出现可选结点，选择其中的 iclk，点击中间的≥按钮。完成后如图 3-48 所示。

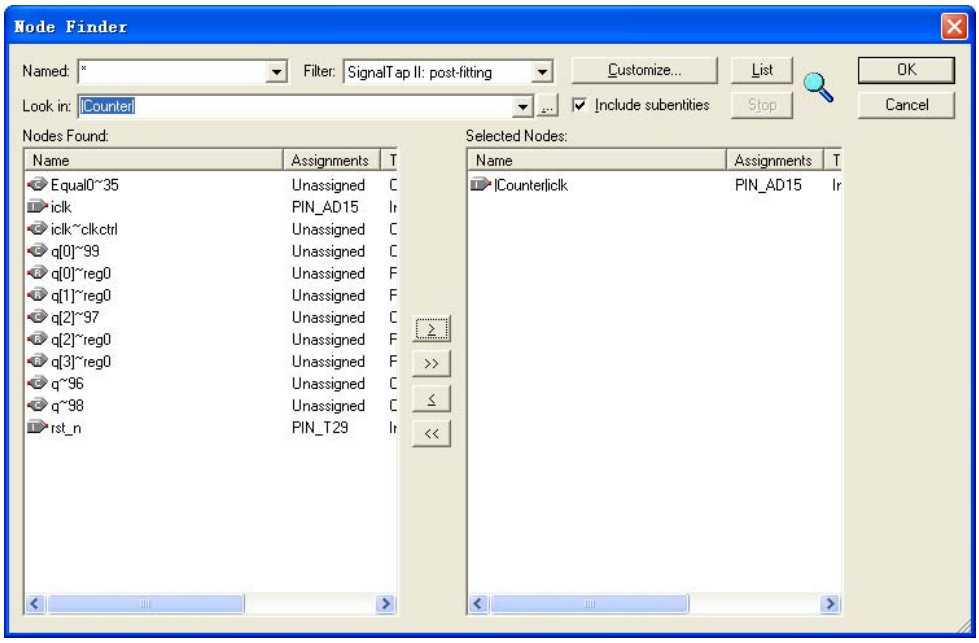


图 3-48　选择时钟结点

完成后出现确认对话框。点击 OK 按钮。参看图 3-49

图 3-49　确认对话框

而后出现时钟结点选择界面，如图 3-50 所示。

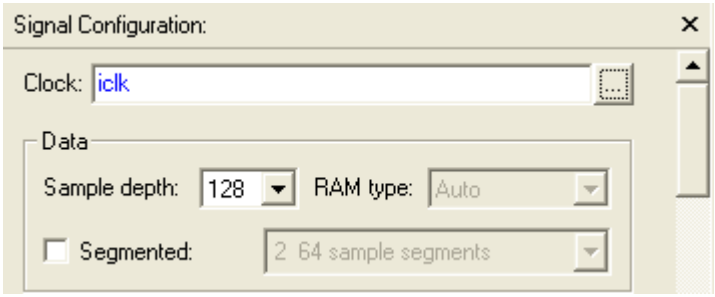


图 3-50　完成时钟结点选择

30.　完成后可在栏中选择存储深度、触发级等选项，这里采用默认设置。下面加入需要观测的结点。在左侧空白区域双击，再次出现选择结点对话框。点击 List 列出所有可选结点。将关心的结点选择好，选择 Pins:all 列出所有引脚，除 iclk 外全部导入。完成后如图 3-51 所示。点 OK 确定。

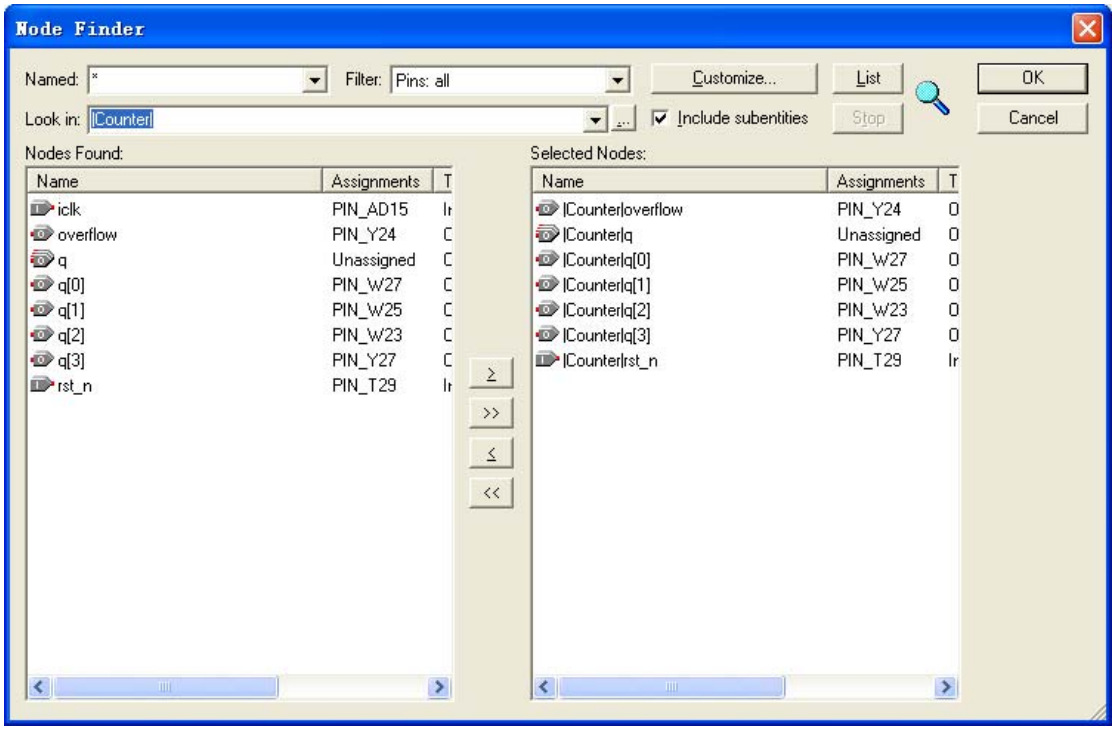


图 3-51 选择观察结点

完成后如图 3-52 所示。

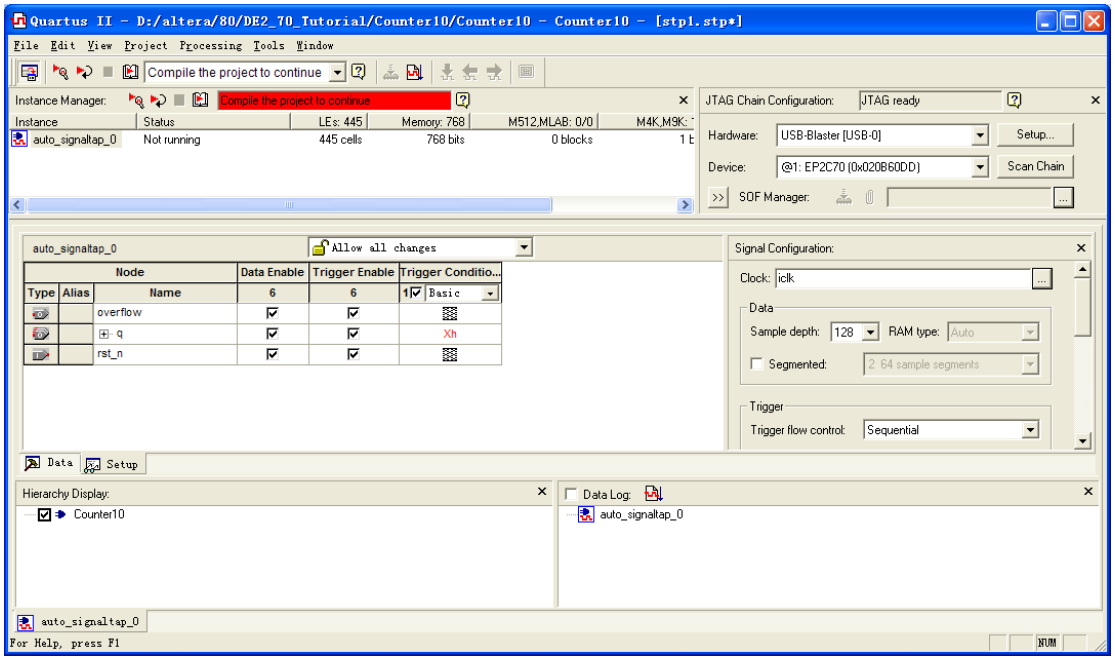


图 3-52 结点完成后界面

31. 完成以上步骤后保存 SignalTap II 文件，可取名为 Counter.stp，参看图 3-53。并在询问是否设置为当前工程的 SignalTap 时选“确定”。再到 Quartus 主界面中执行全编译。编译完成后下载文件到 FPGA。

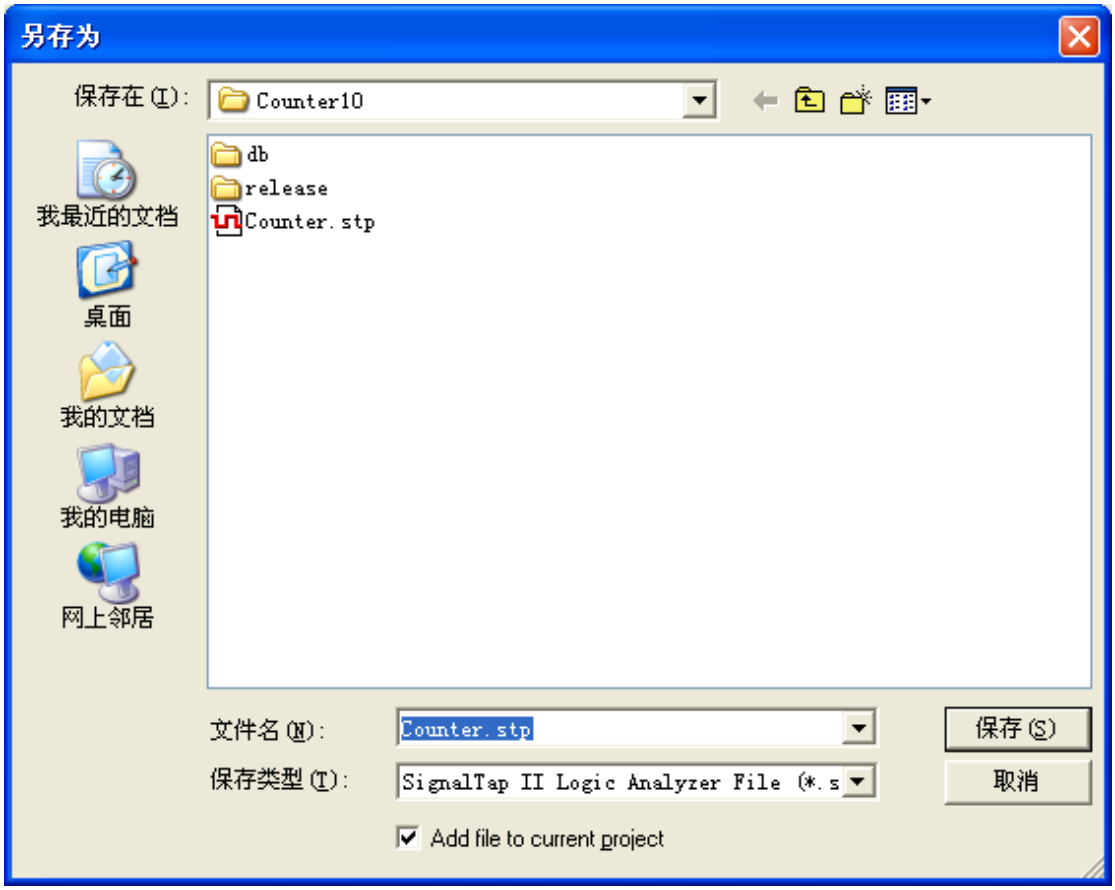


图 3-53 保存 SignalTap II 文件

32. 完成后，再次回到逻辑分析仪文件，点击左上 按钮开始分析。就可以观察到

实际捕获的波形。如图 3-54 所示。

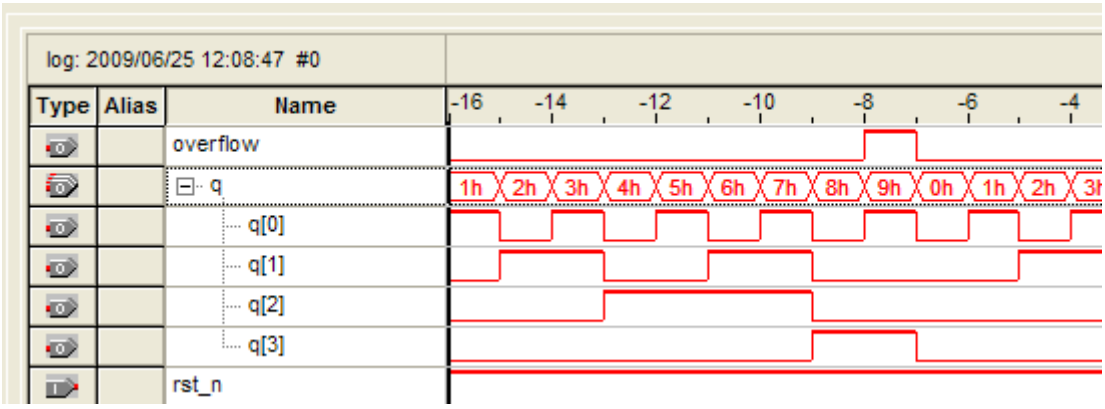


图 3-54 SignalTap II 抓取到的波形

至此，整个实验结束。读者应该体会到了从设计->仿真->下载->波形抓取的四个经典步骤。

第 4 章　实验三　灯光控制实验

- 实验说明

该实验将使用符号框图设计在 DE2-70 开发平台上设计一个非常基本的组合逻辑电路。通过该实验，让读者学会不同于硬件描述语言的一种基于符号框图的设计方案。实验中，还将介绍两种新的引脚配置方法并且简单示意 Quartus 对电路的优化。

- 实验步骤

4.1 建立 Quartus 工程

1. 新建 Quartus 工程 Light，顶层实体名 Light
2. 重新设置编译输出目录为../Light/release。

4.2 使用符号框图描述完成硬件描述设计

3. 选择菜单项 File -> New…，在弹出的对话框中选择 Block Diagram/Schematic File，点击 OK。如图 4-1。

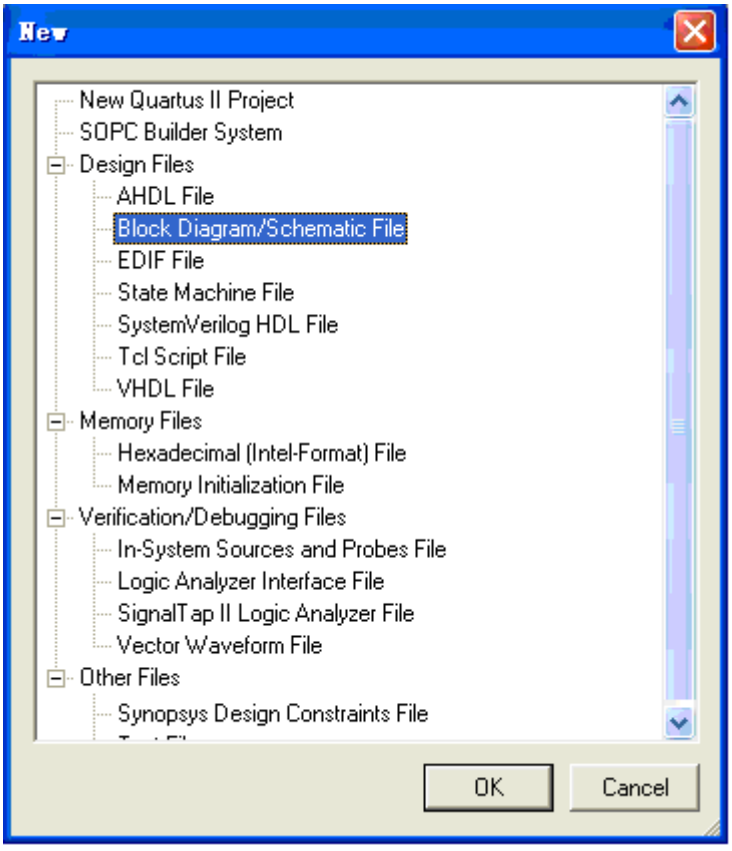


图 4-1　选择设计文件

4. 导入逻辑门电路符号

用鼠标双击图形编辑器窗口的空白处或者短纳期图像编辑器窗口左侧工具条中


图标，选择门电路符号，进行添加。

展开左侧树状目录到…/primitives/logic/and2，找到二输入与门，点击 OK 后在白板上单击放置即可。参看图 4-2。

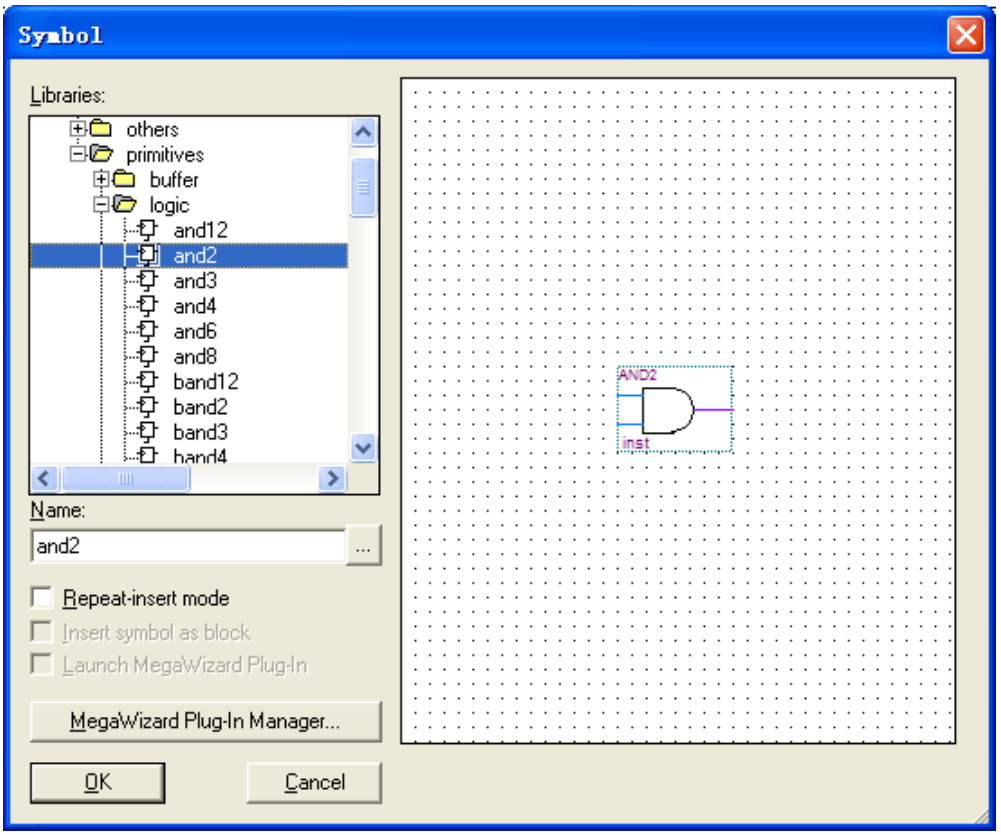


图 4-2　选择二输入与门

展开左侧树状目录到…/primitives/logic/or2，添加二输入或门。参看图 4-3。

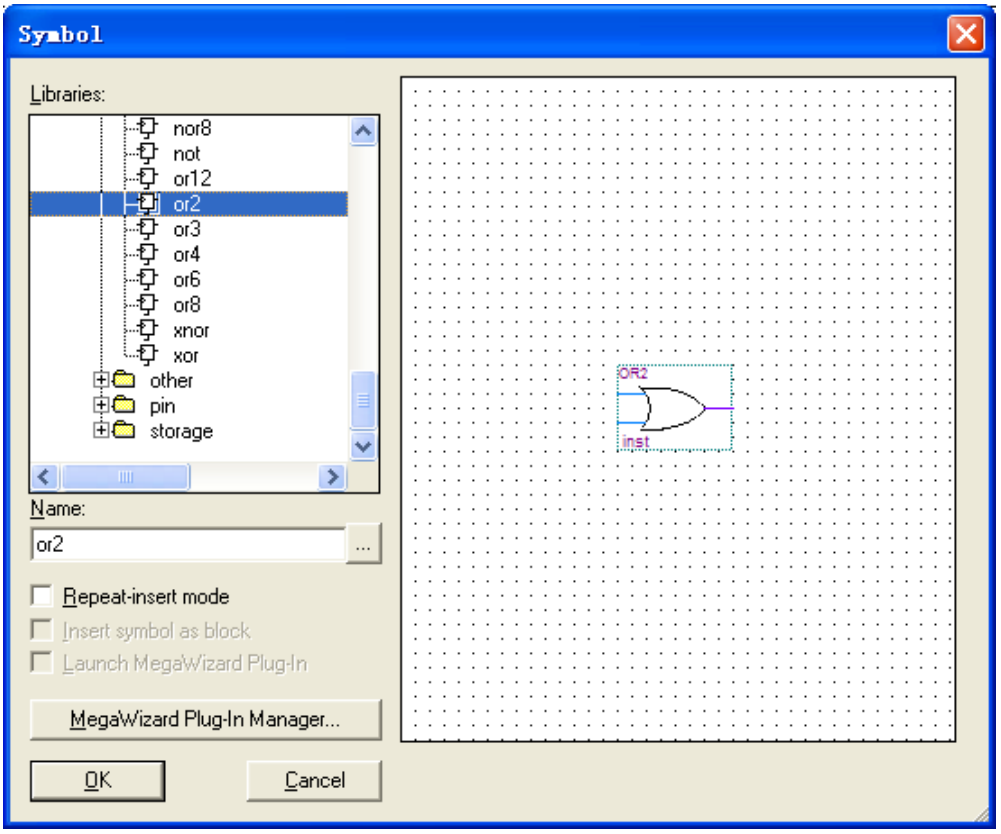


图 4-3　选择二输入或门

依上再添加两个取反门，元件整体布局如图 4-4 所示。

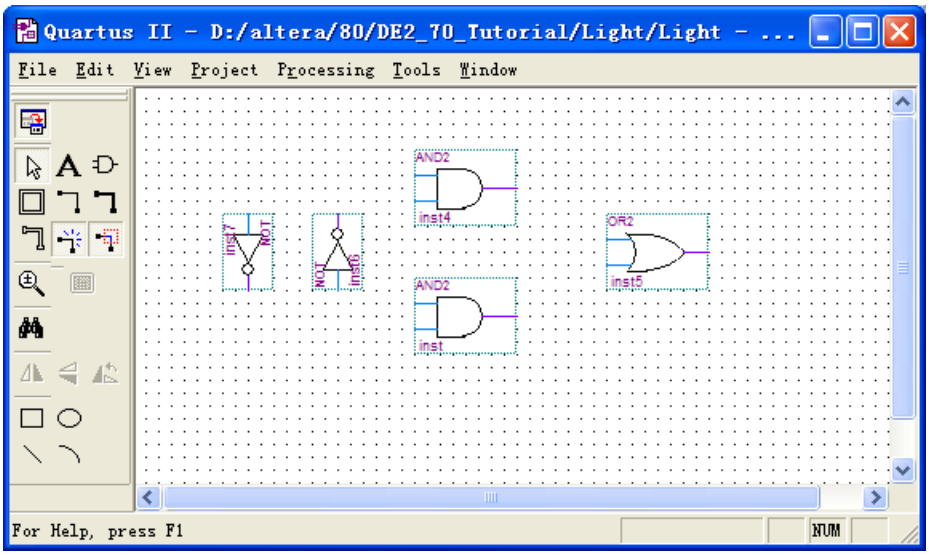


图 4-4　放置门电路符号

5. 放置输入/输出引脚，位置在../primitives/pin/input 与../primitives/pin/output，参看图 4-5 与图 4-6。

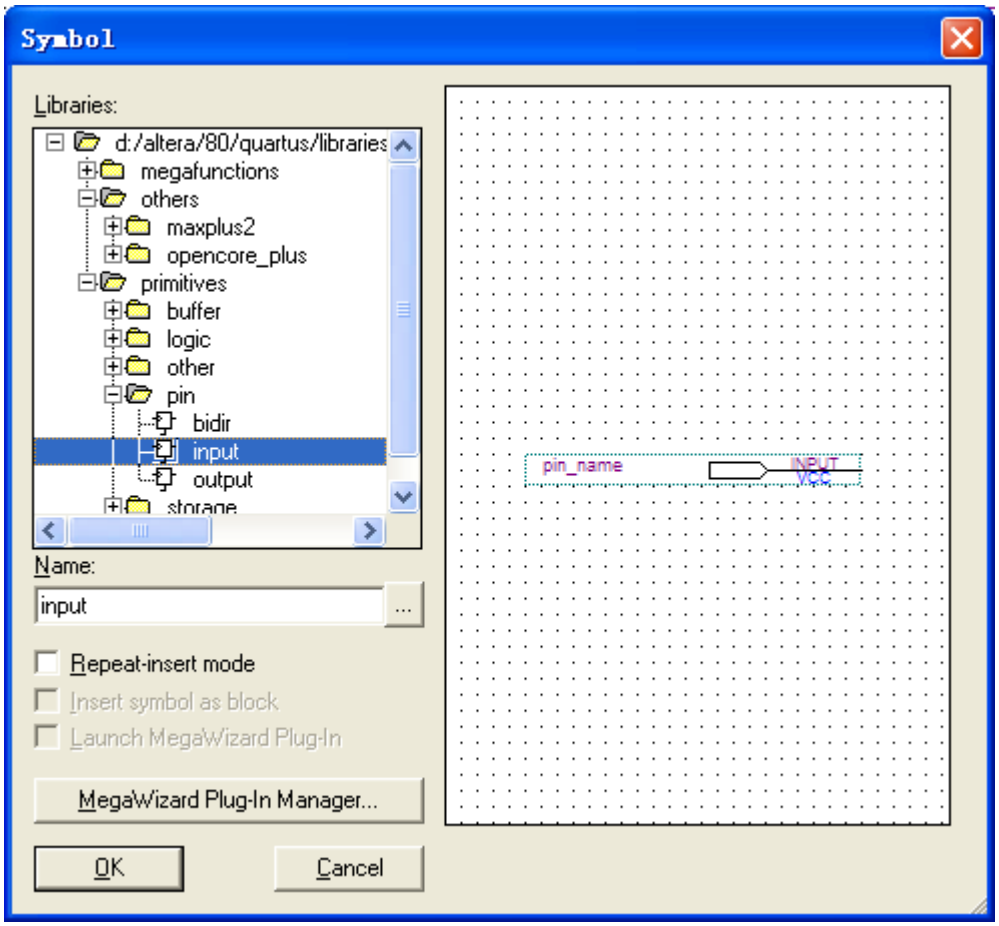


图 4-5　添加输入引脚

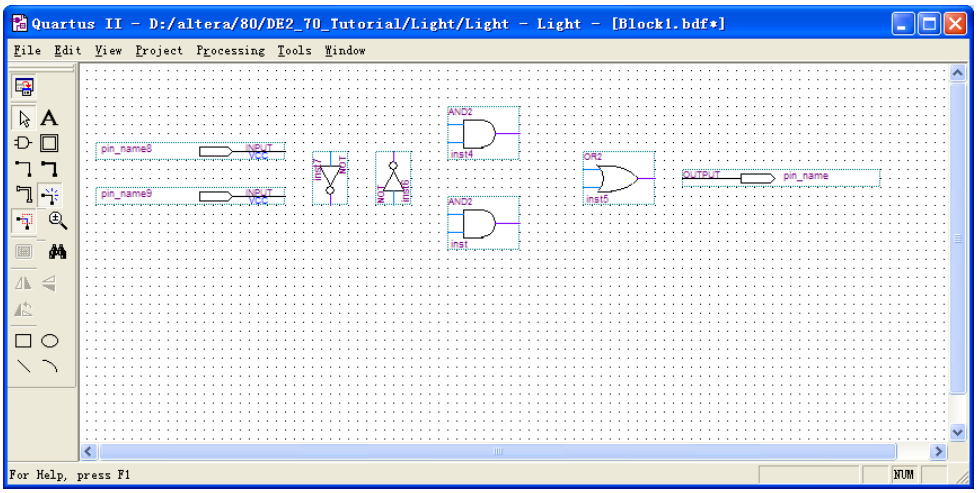


图 4-6 放置输入/输出引脚

6. 电路图连接，用鼠标拖动引线即可连接元件，完成后如图 4-7。

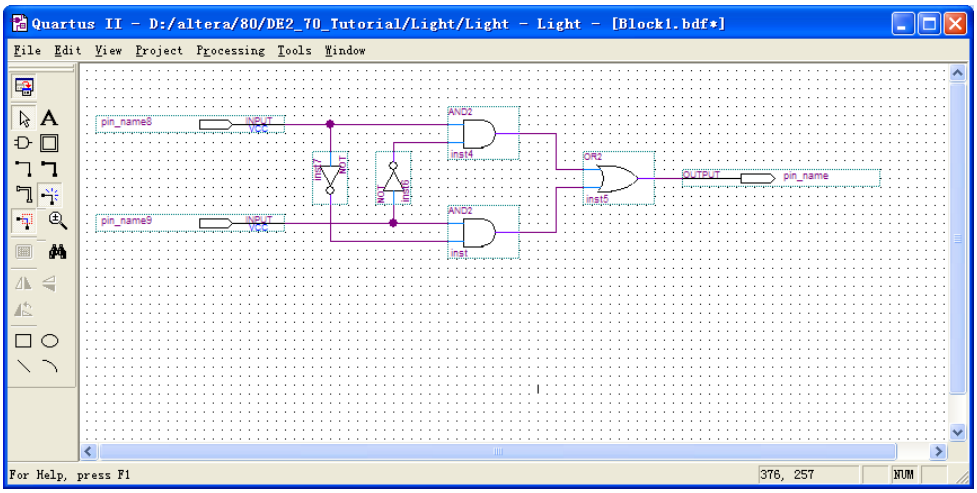


图 4-7　连接电路

7. 修改电路结点的名称，双击元件即可修改，将输出结点名称改为 oLEDG[0]，将两个输入分别改为 iSW[1]和 iSW[0]，并保存设计文件 Light.bdf。如图 4-8。

注意：使用符号框图与使用 Verilog 语言实现完全等价。

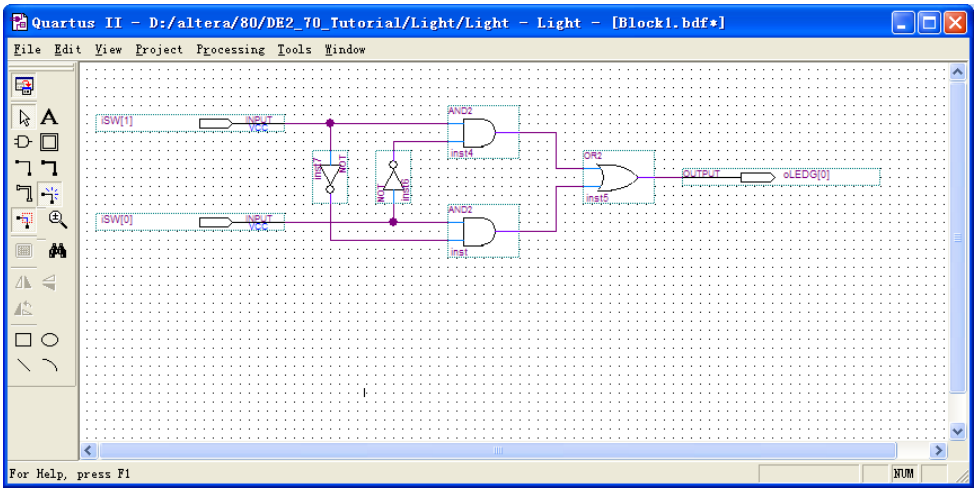


图 4-8　修改引脚名称

8. 分配引脚。前面讲过手工分配引脚，这次使用自动导入。用 Assignments->Import

Assignments…菜单.如图 4-9 所示。

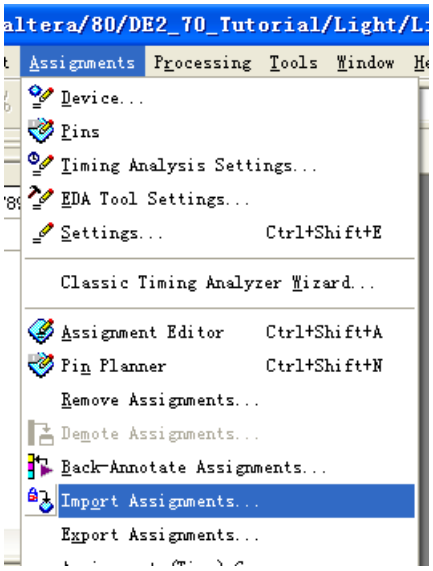


图 4-9　导入引脚配置菜单项

选择文件名，导入 DE2_70_pin_assignments.csv 中的引脚配置。如图 4-10。

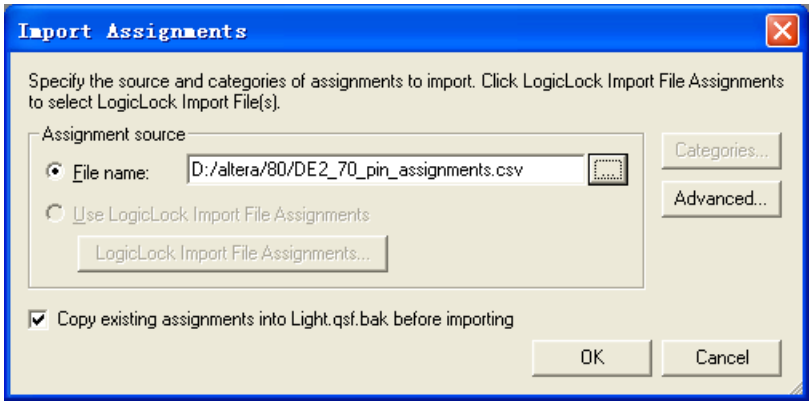


图 4-10　导入引脚配置文件

9.　编译电路，如图 4-11。

图 4-11　编译结果

10.　结果出现 530 个警告，这主要是因为引脚分配文件中含有大量引脚本实验没有用到。如图 4-12。

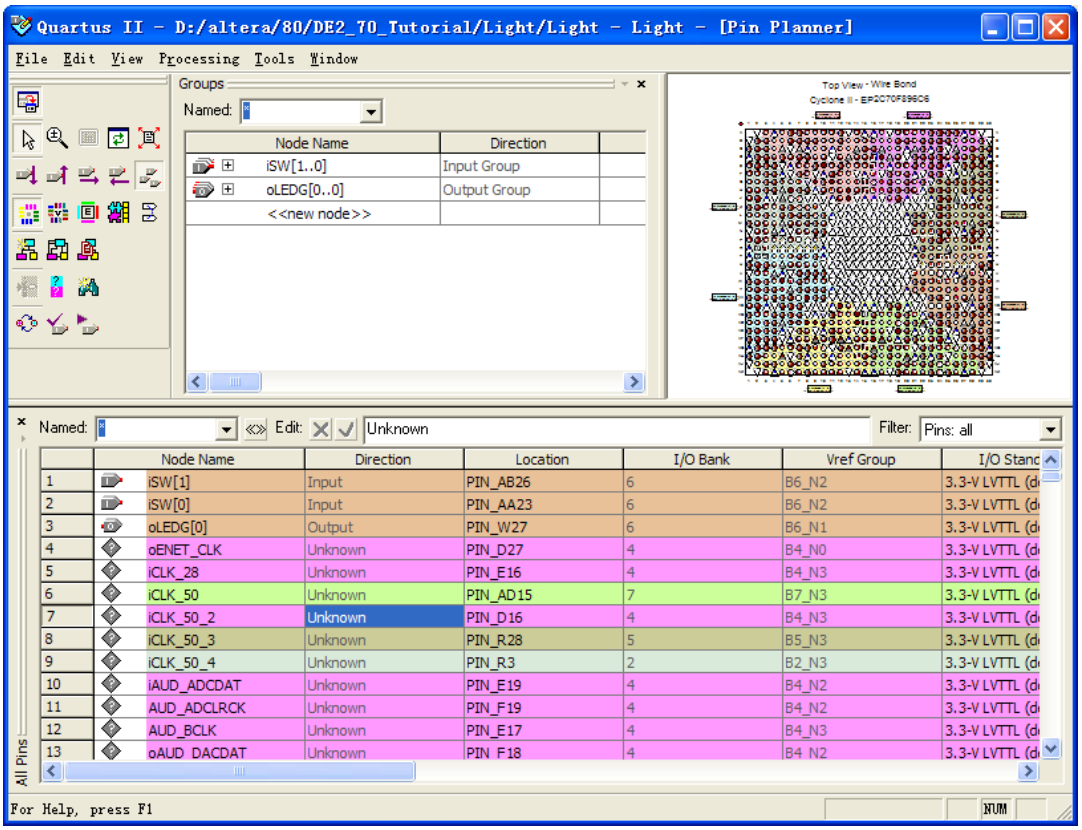


图 4-12　csv 文件自动导入的引脚分配图

11. 删除多余的引脚。用记事本等外部编辑器打开 Light.qsf 文件。发现大量的 set_location_assignment PIN_XXX –to 信息。将不用的信息删除，留下

set_location_assignment PIN_AA23 -to iSW[0]

set_location_assignment PIN_AB26 -to iSW[1]

set_location_assignment PIN_W27 -to oLEDG[0]

三行，保存。回到 Quartus 重新观察引脚。如图 4-13。

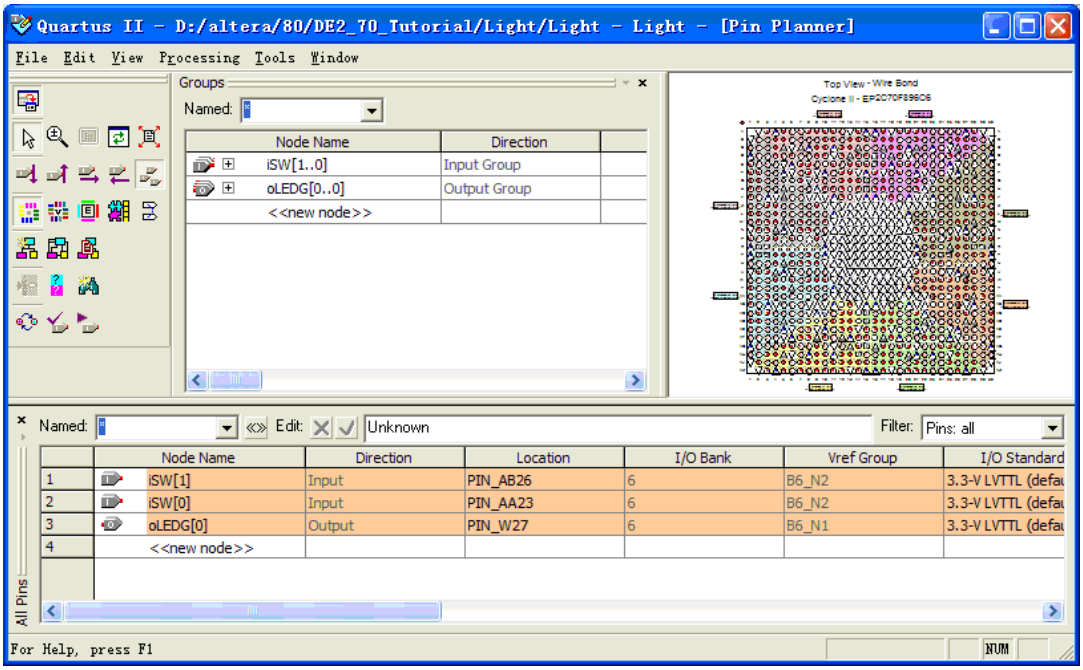


图 4-13　修改 qsf 文件后的引脚分配图

12. 执行全编译，发现警告信息恢复正常。如图 4-14。

图 4-14　编译结果

注意：以后分配引脚，都可使用标准文件名(参考 DE2_70_pin_assignments.csv 文件)加上手工编辑 qsf 文件的方案。

13. 观察电路图，不难发现电路功能仅仅是求一个异或：A^B=A(~B)+(~A)B= A+B-AB

使用一个逻辑单元即可实现，点击菜单项 Tools->Netlist Viewers->Technology Map Viewer，如图 4-15，可以看到只使用了一个逻辑单元，如图 4-16。

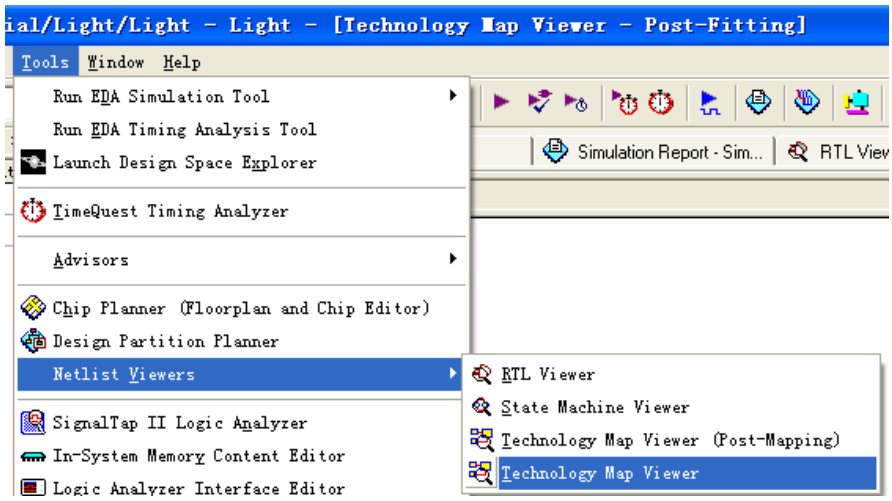


图 4-15　查看 Technology Map 菜单项

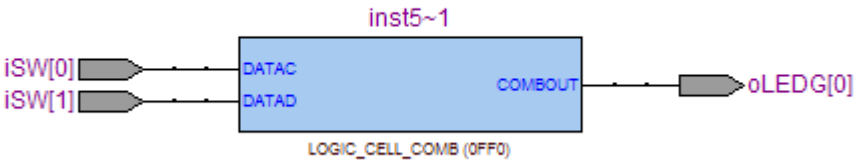


图 4-16　查看 Technology Map

双击进入，可以看到实际只有一个异或门。如图 4-17。这是 Quartus 对电路自动进行的优化。

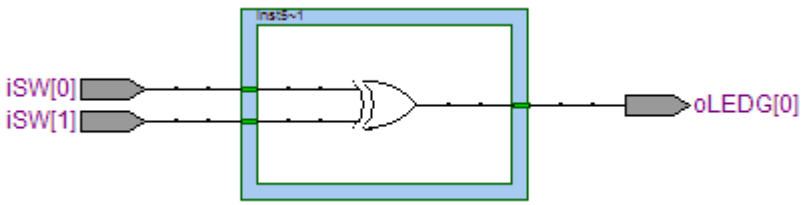


图 4-17　查看 Technology Map 内部

4.3 电路仿真

在 DE2-70 平台上实现该电路之前，我们先在 Quartus II 软件中对电路进行功能仿真，以测试设计的正确性。

14. 点击菜单项 File->New，打开如图所示对话框，并选择 Verification/Debugging Files 中的 Vector Waveform File，单击 OK。

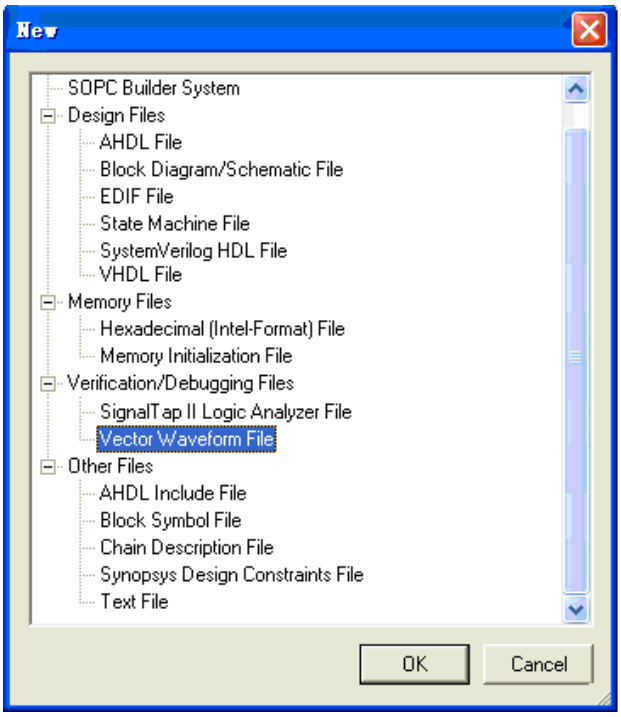


图 4-18　建立矢量波形文件

15. 点击菜单项 Edit->End Time，设定仿真终止时间为 200ns。如图 4-19 与图 4-20。

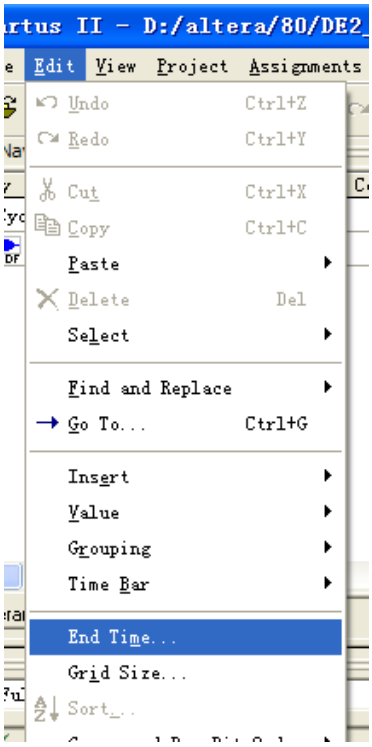


图 4-19 设定仿真结束时间菜单项

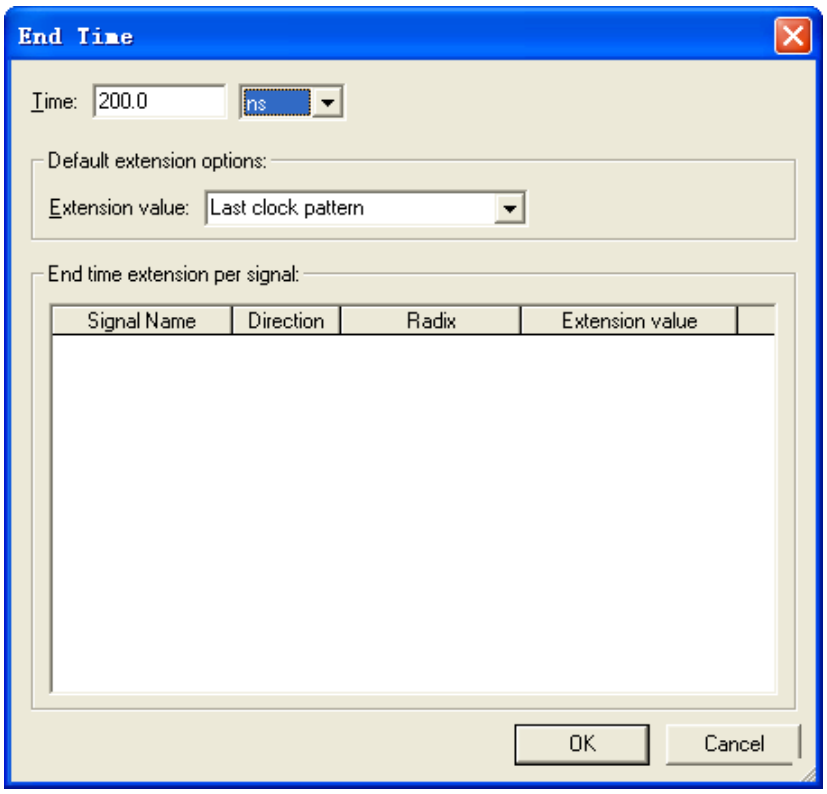


图 4-20　设定仿真结束时间

16. 将要仿真的输入/输出等电路结点加入到波形中来。用 Edit->Insert->Insert Node or Bus 菜单如图所示。如图 4-21。

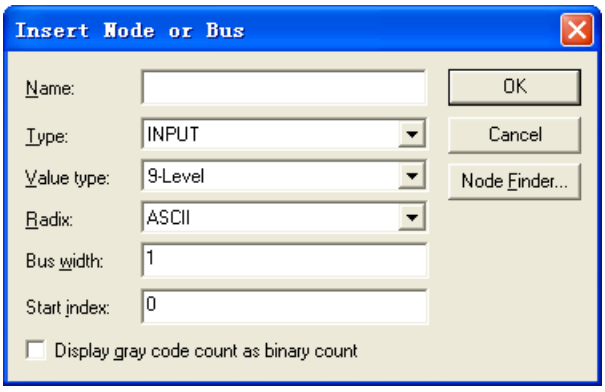


图 4-21　添加结点到波形图

17. 单击 Node Finder..按钮，添加相关的结点 iSW[0]、iSW[1]、oLEDG[0]到波形图。将 iSW[0]设为周期 200ns 的方波，iSW[1]设为周期 100ns 的方波。如图 4-22。

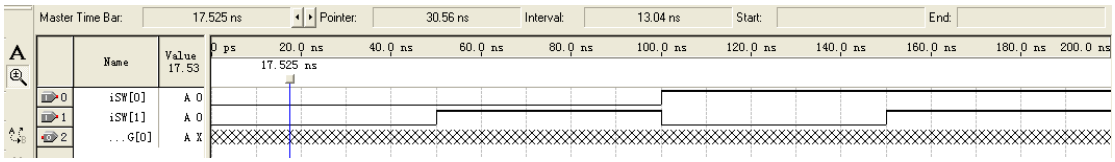


图 4-22　测试用的矢量波形

18. 功能仿真

点击菜单项 Assignment->Settings，选中 Simulator Settings 选项卡，出现图 4-23 所示对话框。在 Simulation mode 中选择 Functional，Simulation input 选择刚才建立的波形文件，完成后点击 OK。

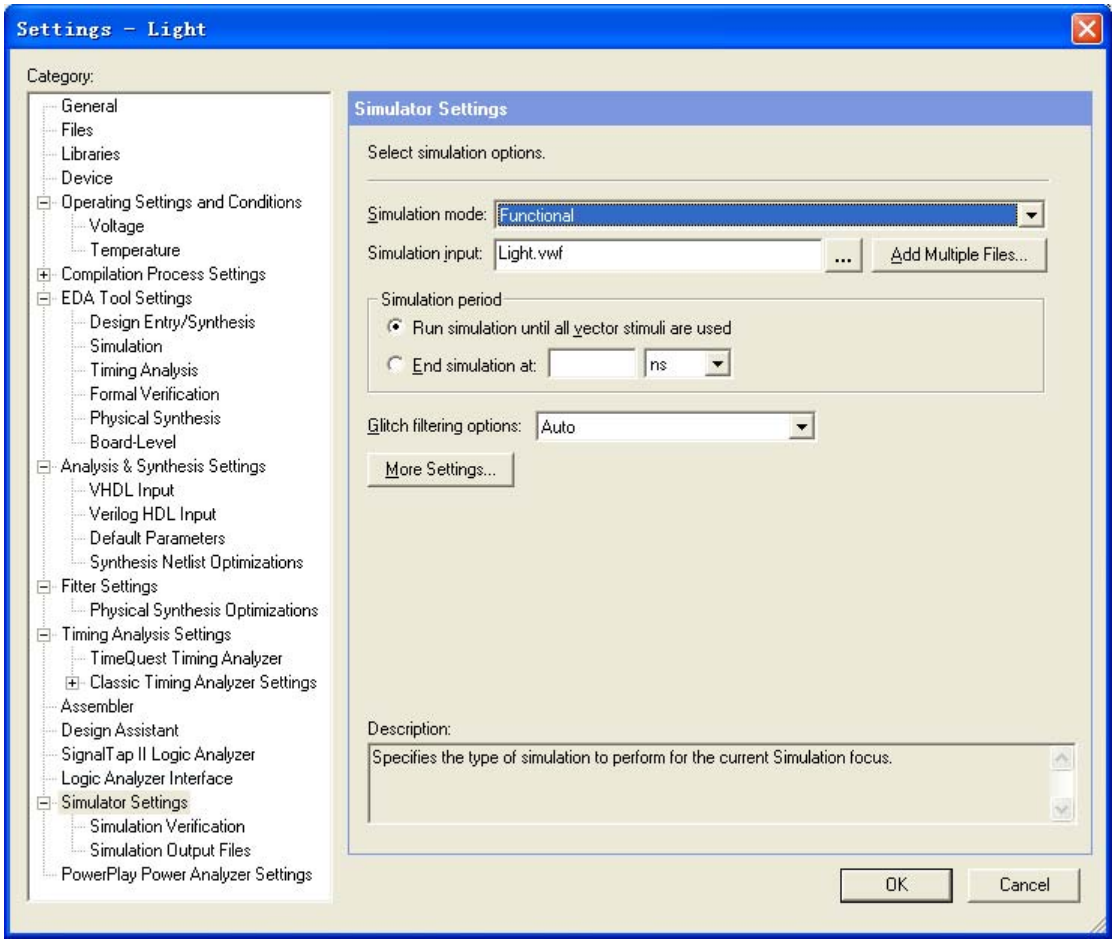


图 4-23　设置仿真模式

点击菜单项 Processing->Generate Functional Simulation Netlist 产生功能仿真所需的网表。点击菜单项 Processing->Start Simulation 或 工具按钮启动功能仿真。结果如图 4-24。

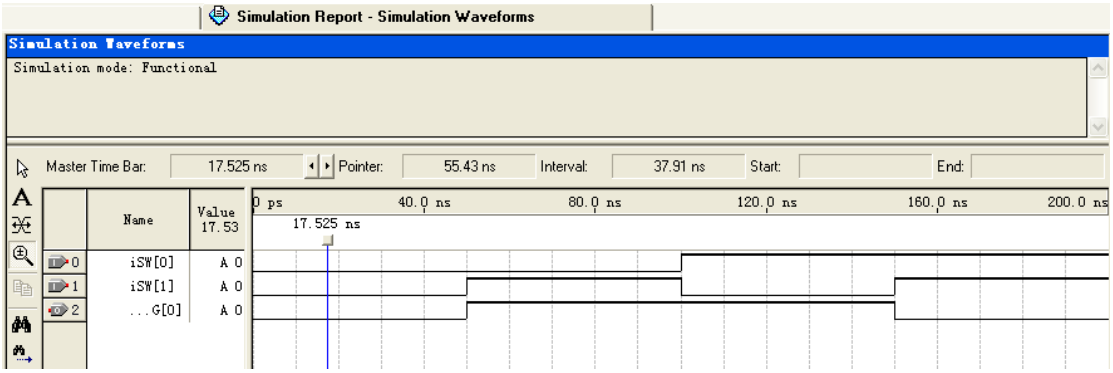


图 4-24　功能仿真结果图

19. 时序仿真

点击菜单项 Assignment->Settings 菜单打开 Settings 窗口，在 Simulation mode 中选择 Timing，Simulation input 选择刚才建立的波形文件，单击 OK。

8.0 版在左侧 Tasks 窗口中 Quartus II Simulator (Timing)处右击 start，执行时序仿真。

7.2 版需要依次执行 Start Analysis&Synthesis、Start Fitter、Start Classic Timing Analyzer、Start TimeQuest Timing Abalyzer、Start Simulation(参看实验二)。

结果如图 4-25 与图 4-26

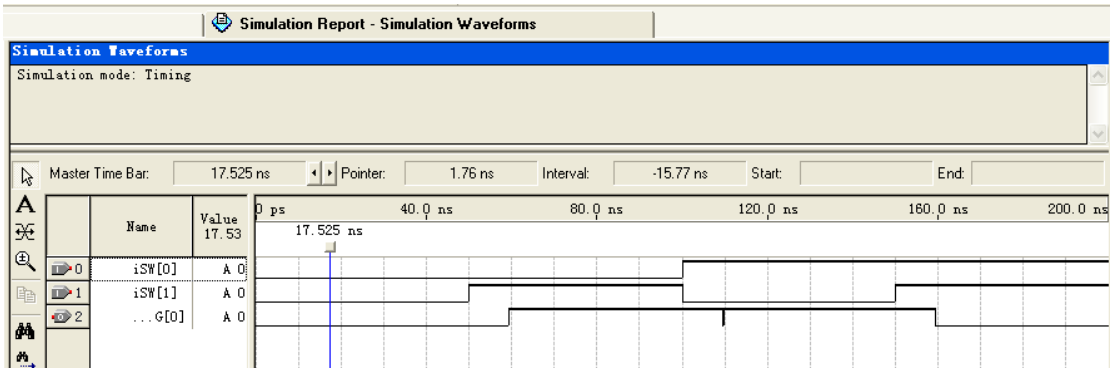


图 4-25　时序仿真结果图

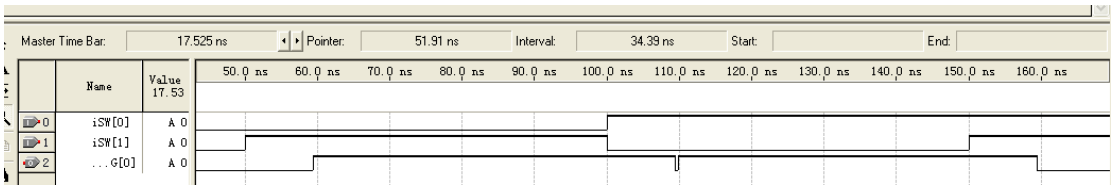


图 4-26　时序仿真结果图放大

注意：图中从 iSW[1]状态变化到输出 oLEDG[0]状态发生相应的变化之间有 9.44-9.45ns 的延时，而从输入 iSW[0]状态变化到输出 oLEDG[0]状态发生相应的变化之间约有 9.84ns 的延时。该延时为 FPGA 布线所决定，不同开发板均不同。

20. 下载到开发板上，拨动开关查看结果，应与异或操作结果一致。

第 5 章　实验四　移位寄存器实验

- 实验说明

该实验使用两种方法在 DE2-70 平台上设计一个移位寄存器。通过这个实验，读者可以了解使用 Quartus 工具中的 MegaFunction 功能的基本流程，MegaFunction 中有大量的 IP Core 可以使用，可以加快设计周期，实验中还通过 Verilog 实现了同样功能的移位寄存器作为对比。实验中的 MegaFunction 模块通过符号框图方法使用，这是 Quartus 的另一种设计输入方法。

- 实验步骤

5.1 建立 Quartus 工程

1. 新建 Quartus 工程 ShiftRegCore，顶层实体名 ShiftRegCore
2. 重新设置编译输出目录为../ ShiftRegCore/release。

5.2 使用 MegaFunction+符号框图描述完成硬件描述设计

3. 用 MegaFunction 创建移位寄存器模块，首先点击菜单项 Tools->MegaWizard Plug-In Manager…。如图 5-1。

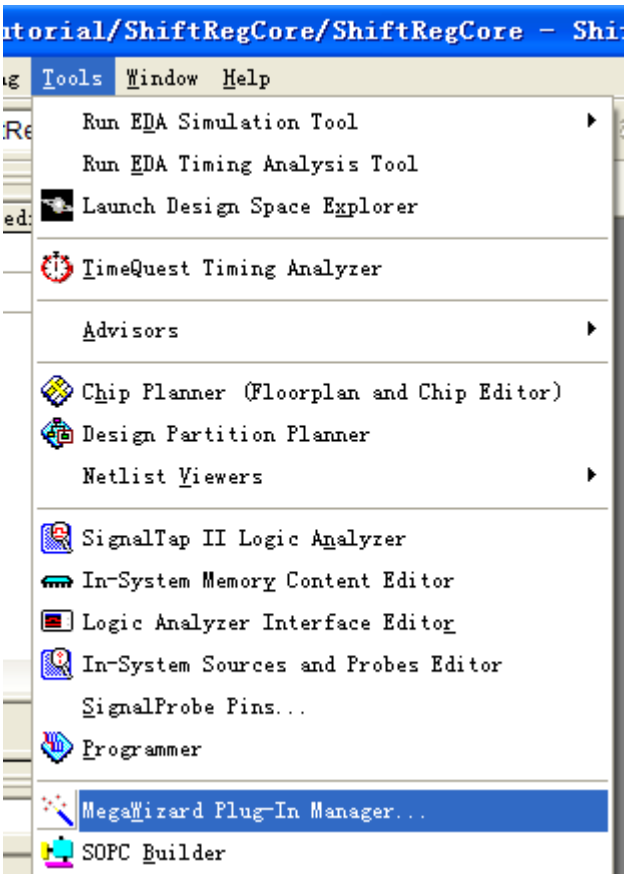


图 5-1　MegaFunction 菜单

打开后的界面如图 5-2 所示。选择其中的第一项，新建一个模块。选好后点 Next。

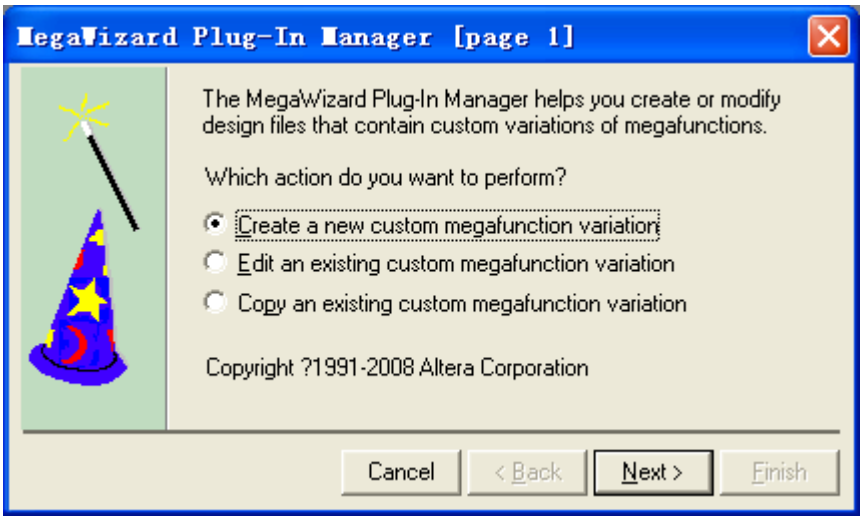


图 5-2　MegaFunction 第一步

接下来出现如图 5-3 所示的界面。如图在左侧选择 Installed Plug-Ins->Storage->LPM_SHIFTREG。并在右侧输出文件名一栏中输入所要的文件名。本例中输入 ShiftReg。完成后点 Next。

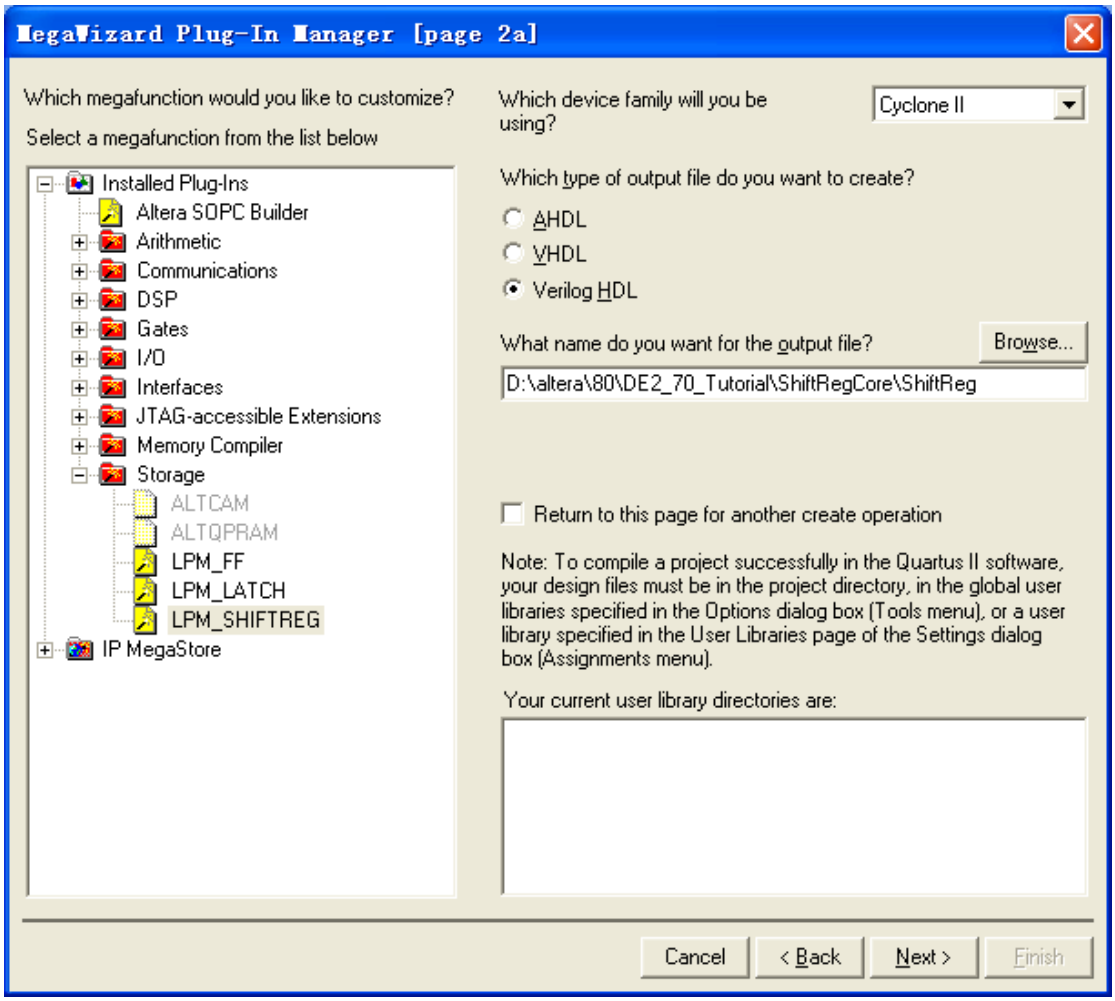


图 5-3 选择所需的 MegaFunction

完成后出现图 5-4 所示界面。按图选择选项，创建一个带有 load 端、串行输入输出、并行输出功能的右移寄存器。点 Next 进入下一页。

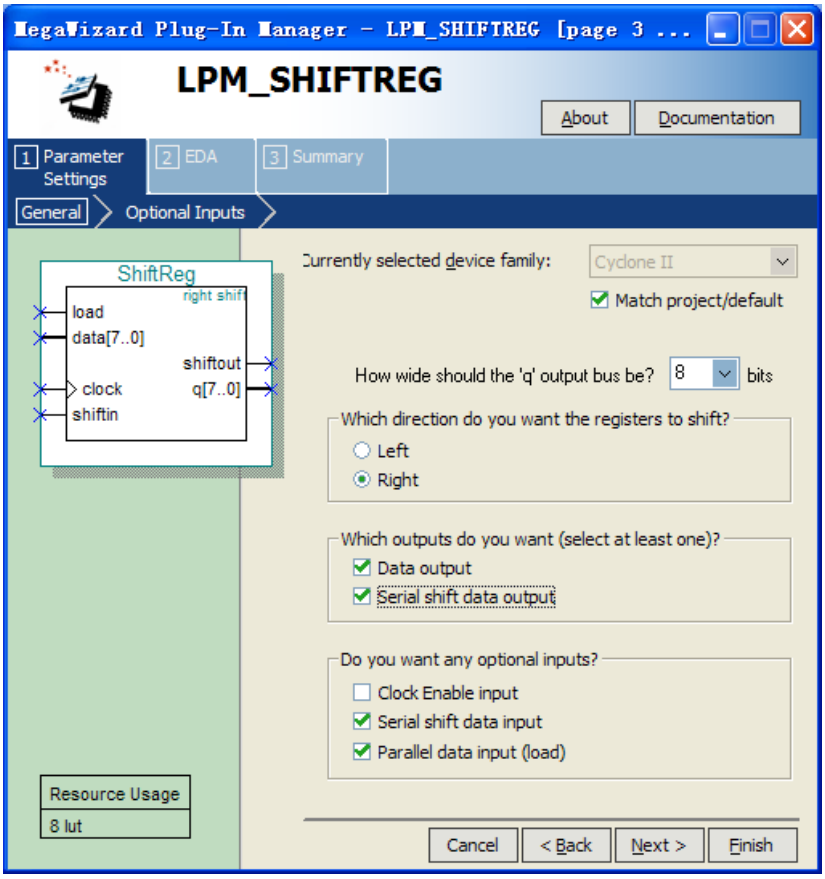


图 5-4　选择移位寄存器功能

完成后如图 5-5 所示。可进一步选择其功能。为移位寄存器增加异步清零端。完成后点击 Next 进入下一页。

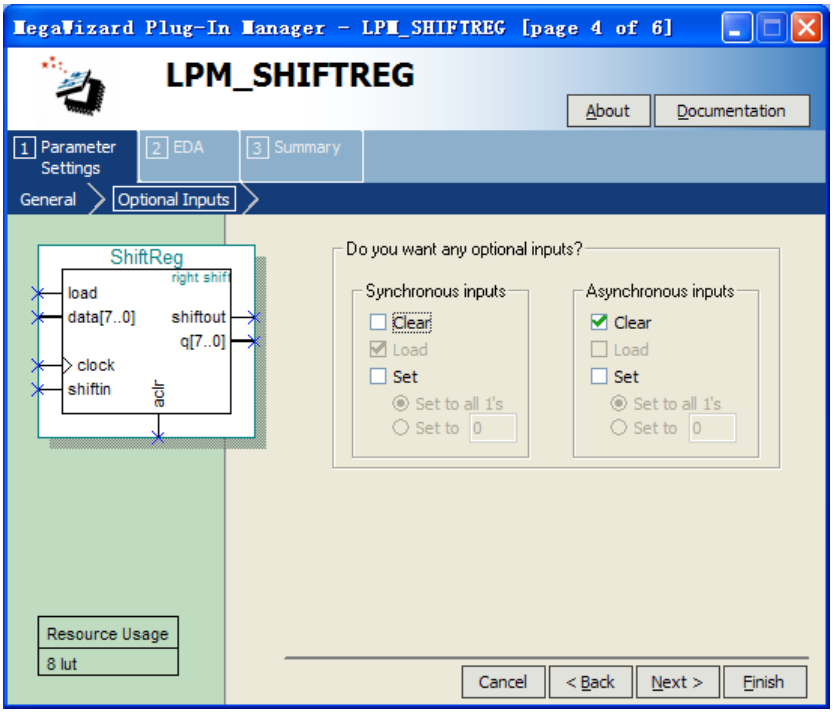


图 5-5 进一步选择

完成后如图 5-6 所示。可以选择生成相应的仿真库。本实验采用默认操作。点 Next 进入下一页。

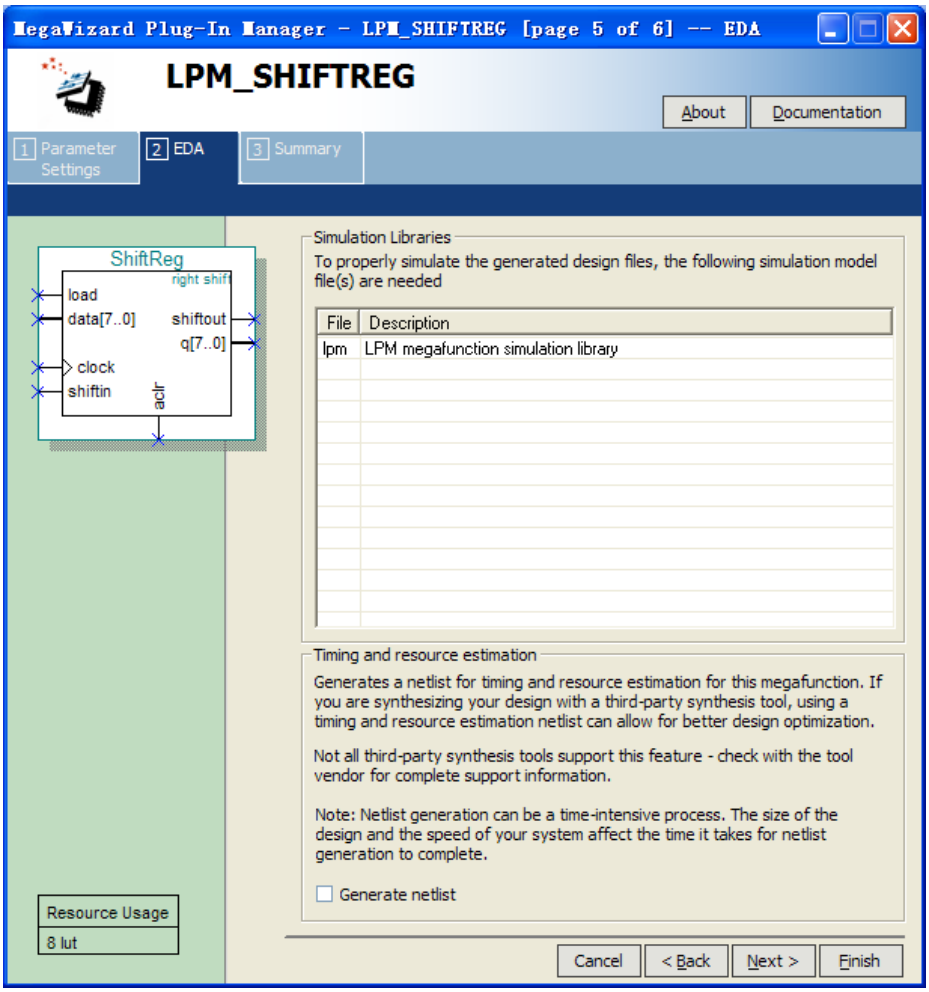


图 5-6 与仿真工具接口

完成后如图 5-7 所示。该页给出了可选的生成文件。注意选择上 ShiftReg.bsf 文件以生成符号文件，为符号框图输入提供源文件。完成此步后直接点 Finish 完成 MegaFunction 设计。完成后如果有对话框问你是否添加此 ip，选否。

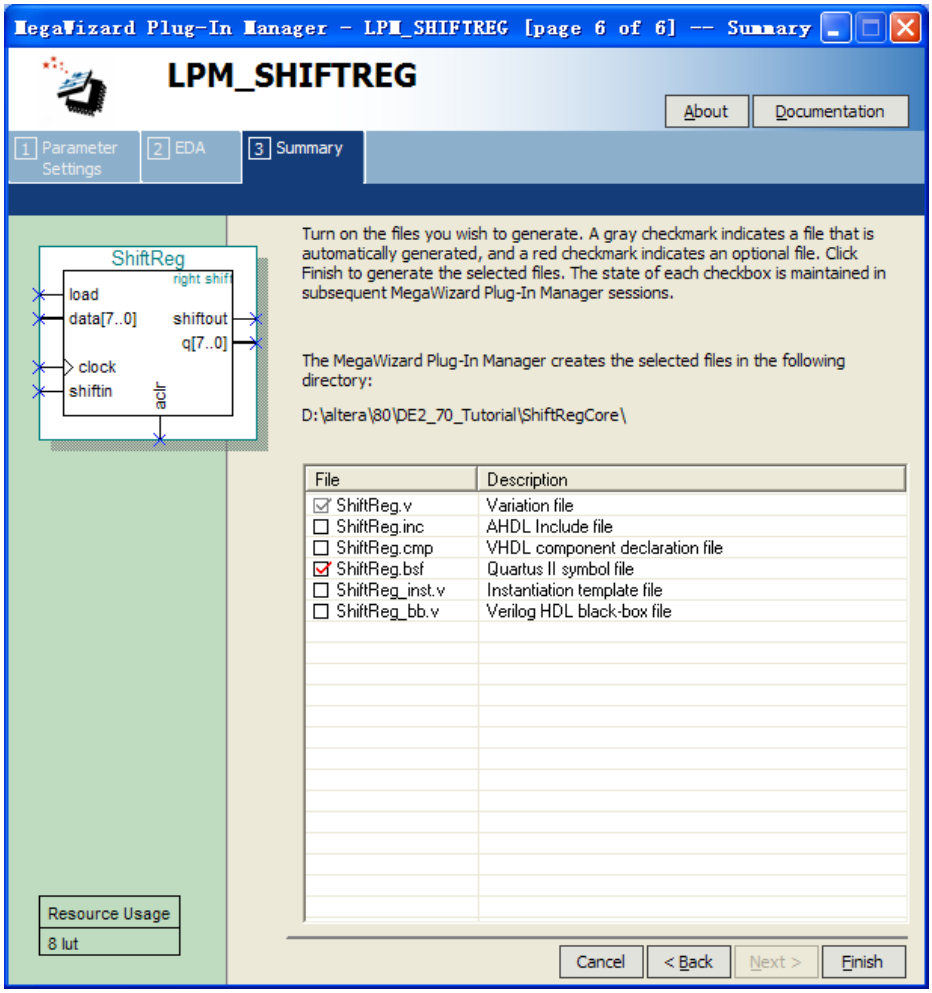


图 5-7 输出文件列表

4. 创建顶层实体描述文件。本次实验使用符号框图来完成这一步。实际上这与使用 Verilog 语言实现顶层实体完全等价。首先点击新建文件，选择其中的 Block Diagram/Schematic File。文件如图 5-8 所示。

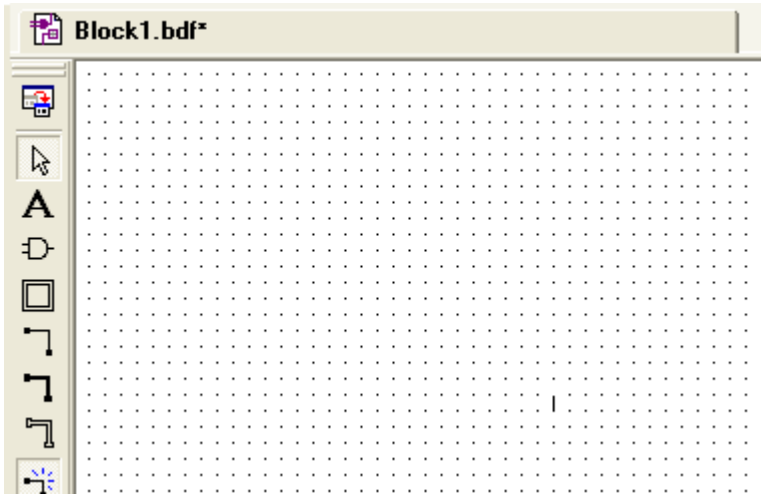


图 5-8 符号框图文件

5. 加入移位寄存器模块。在文件空白处双击进入选择模块对话框。在左侧选择 Project->ShiftReg，即刚才创建的模块，点 OK 可以选择插入点，在适当位置点击左键插入。插入后如图 5-9、5-10 所示。

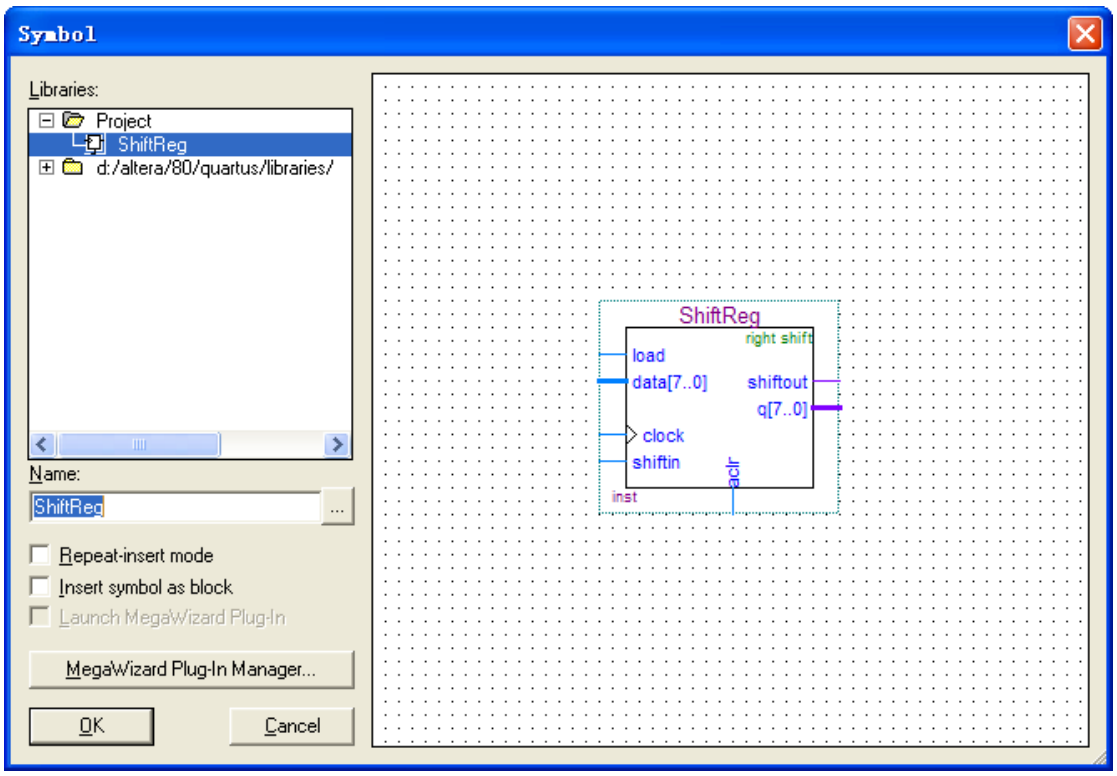


图 5-9 选择模块对话框

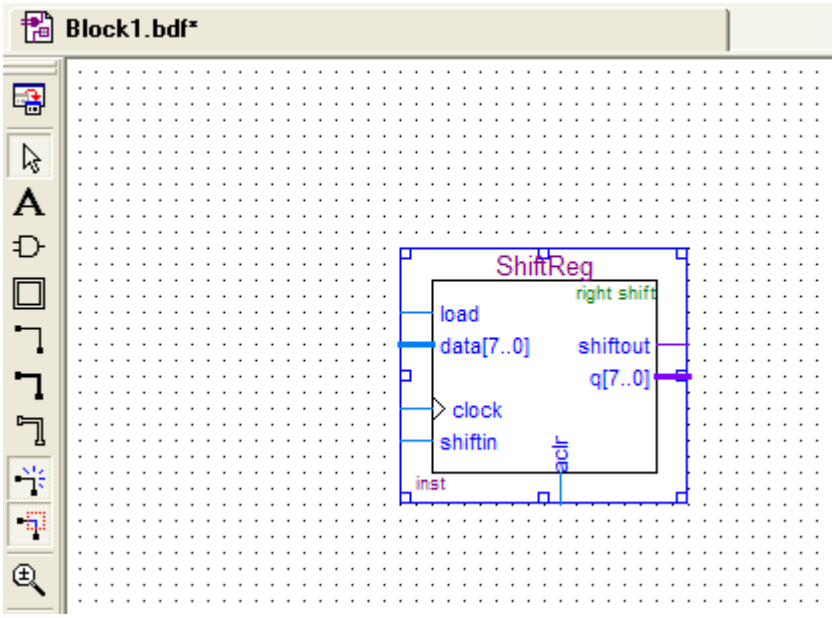


图 5-10 插入移位寄存器

6. 加入输入输出引脚。依照上一步，选择输入输出引脚，插入到符号框图中。如图 5-11 所示。

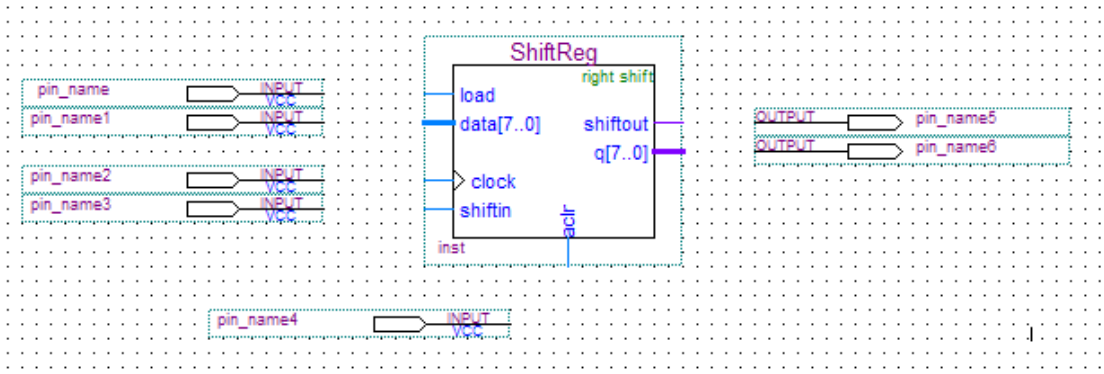


图 5-11 放置输入输出引脚

连接并修改引脚名称。注意数组的写法 iSW[7..0]，这与硬件描述语言中的 iSW[7:0]不同。如图 5-12。

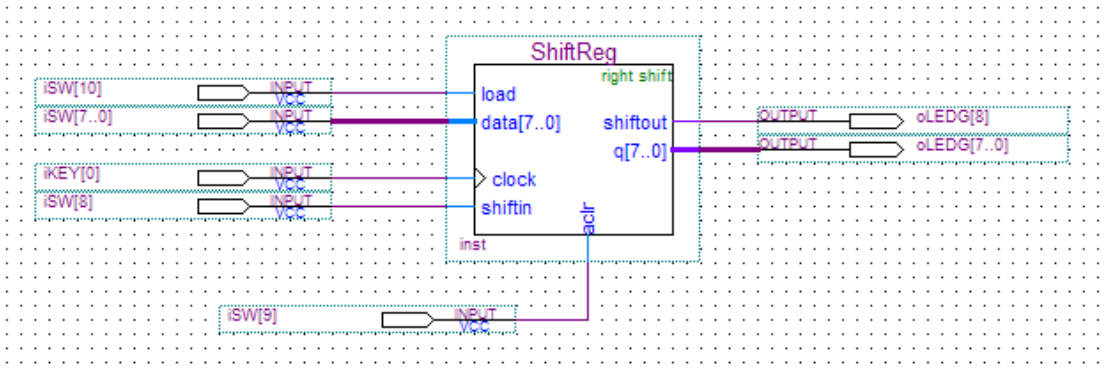


图 5-12 连接并修改引脚名称

完成后保存该文件。

7. 分析与综合，这一步除了检查电路图是否出错外，尚有另一个目的，就是为了将已有配置信息写入 qsf 文件。以免分配引脚时造成 qsf 文件的修改时间比内存更新，丢失以往配置信息(如编译输出路径)。

8. 配置引脚。直接编译 qsf 文件添加引脚。（参见实验三）结果如图 5-13。

set_location_assignment PIN_AA23 -to iSW[0]
set_location_assignment PIN_AB26 -to iSW[1]
set_location_assignment PIN_AB25 -to iSW[2]
set_location_assignment PIN_AC27 -to iSW[3]
set_location_assignment PIN_AC26 -to iSW[4]
set_location_assignment PIN_AC24 -to iSW[5]
set_location_assignment PIN_AC23 -to iSW[6]
set_location_assignment PIN_AD25 -to iSW[7]
set_location_assignment PIN_AD24 -to iSW[8]
set_location_assignment PIN_AE27 -to iSW[9]
set_location_assignment PIN_W5 -to iSW[10]
set_location_assignment PIN_W27 -to oLEDG[0]
set_location_assignment PIN_W25 -to oLEDG[1]
set_location_assignment PIN_W23 -to oLEDG[2]
set_location_assignment PIN_Y27 -to oLEDG[3]
set_location_assignment PIN_Y24 -to oLEDG[4]

set_location_assignment PIN_Y23 -to oLEDG[5]
set_location_assignment PIN_AA27 -to oLEDG[6]
set_location_assignment PIN_AA24 -to oLEDG[7]
set_location_assignment PIN_T29 -to iKEY[0]
set_location_assignment PIN_AC14 -to oLEDG[8]

Named: Edit:

	Node Name	Direction	Location	I/O Bank	Vr
1	iKEY[0]	Input	PIN_T29	6	B6_N0
2	iSW[10]	Input	PIN_W5	1	B1_N1
3	iSW[9]	Input	PIN_AE27	6	B6_N2
4	iSW[8]	Input	PIN_AD24	6	B6_N3
5	iSW[7]	Input	PIN_AD25	6	B6_N3
6	iSW[6]	Input	PIN_AC23	6	B6_N3
7	iSW[5]	Input	PIN_AC24	6	B6_N3
8	iSW[4]	Input	PIN_AC26	6	B6_N3
9	iSW[3]	Input	PIN_AC27	6	B6_N2
10	iSW[2]	Input	PIN_AB25	6	B6_N2
11	iSW[1]	Input	PIN_AB26	6	B6_N2
12	iSW[0]	Input	PIN_AA23	6	B6_N2
13	oLEDG[8]	Output	PIN_AC14	8	B8_N0
14	oLEDG[7]	Output	PIN_AA24	6	B6_N2
15	oLEDG[6]	Output	PIN_AA27	6	B6_N1
16	oLEDG[5]	Output	PIN_Y23	6	B6_N2
17	oLEDG[4]	Output	PIN_Y24	6	B6_N2
18	oLEDG[3]	Output	PIN_Y27	6	B6_N1
19	oLEDG[2]	Output	PIN_W23	6	B6_N1
20	oLEDG[1]	Output	PIN_W25	6	B6_N1
21	oLEDG[0]	Output	PIN_W27	6	B6_N1

图 5-13　分配引脚

9.　编译，一般情况下你会遇到Quartus II与DE2-70的一个重大兼容性bug，如图5-14。

```
Info: Device migration not selected. If you intend to use device migration la
Info: Fitter converted 3 user pins into dedicated programming pins
Error: Can't place pins assigned to pin location Pin_AD25 (IOC_X95_Y2_N1)
Info: Fitter preparation operations ending: elapsed time is 00:00:00
Error: Can't fit design in device
Error: Quartus II Fitter was unsuccessful. 2 errors, 0 warnings
Error: Quartus II Full Compilation was unsuccessful. 4 errors, 2 warnings
```

图 5-14　Quartus II 引脚无法分配的问题

出现的条件似乎是使用了包含 iSW[7]的多数开关，也就是 AD25 对应的引脚。（即使没有遇到，也请看一下，只要是大量使用开关引脚，将来一定会遇到。）

解决方法如下：点击菜单项 Assignments->device..选择 Device and Pin Options，如图5-15。

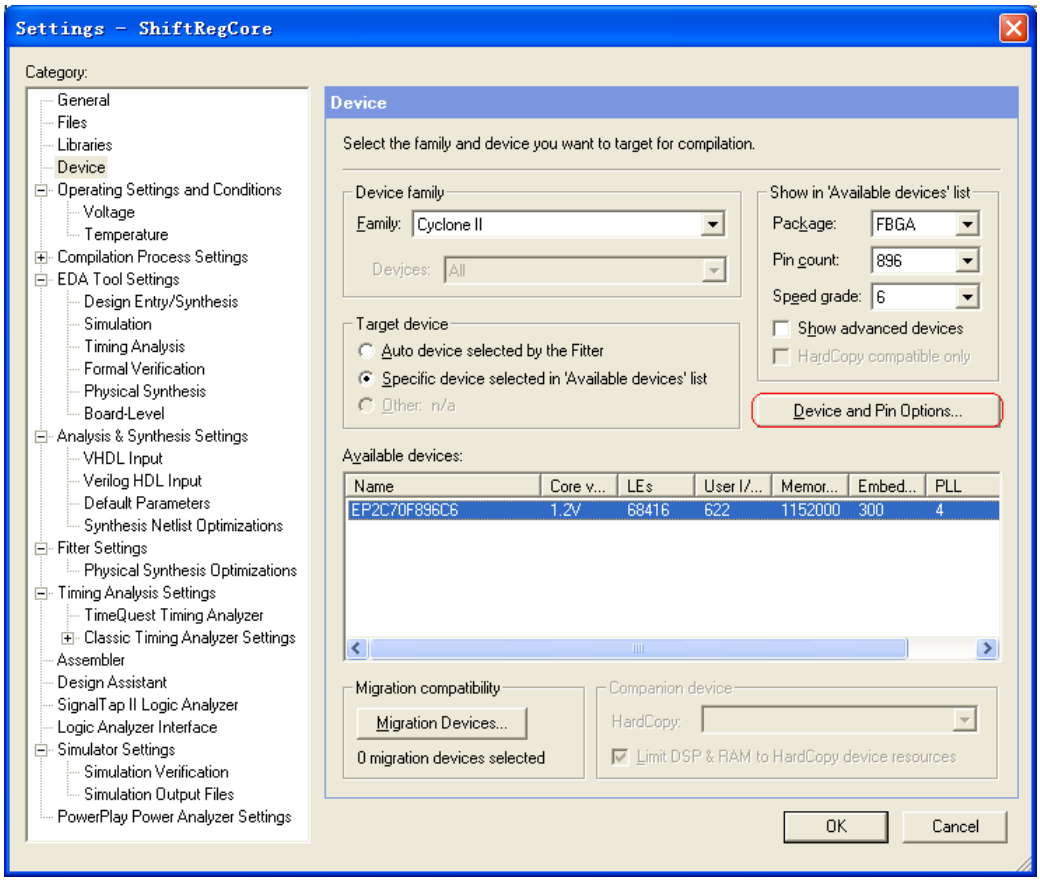


图 5-15　Device 设置对话框

选择 Dual-Purpose Pins 标签，然后把 nCEO 那一项改为 Used as regular I/O，如图 5-16。

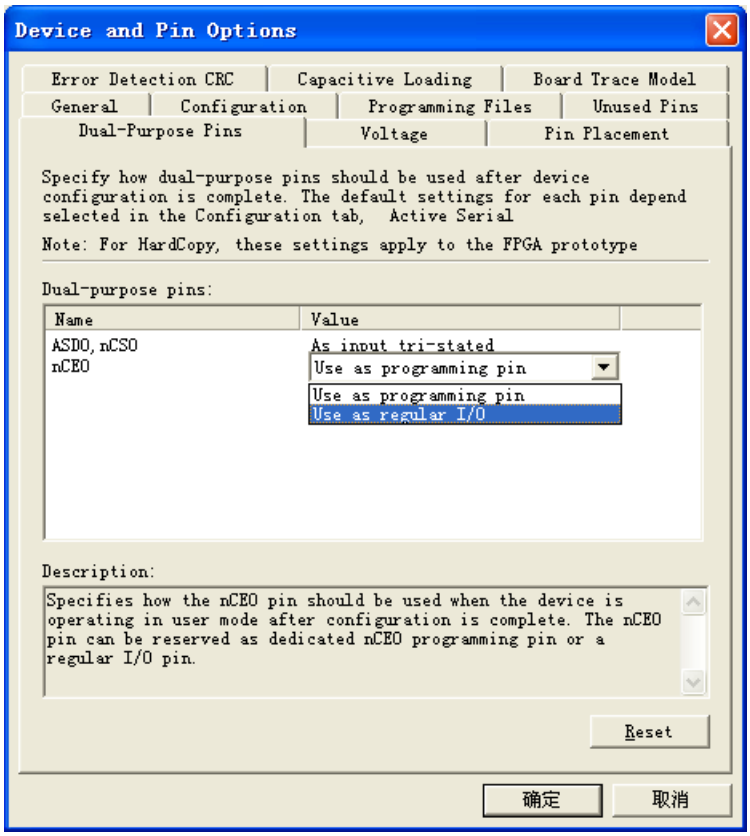


图 5-16　Dual-Purpose 引脚设置对话框

10. 重新编译，错误已经消失。

11. 下载运行。

运行操作非常繁琐，故特别说明：首先将低八位开关定为你想要的数据初始值，比如01101101，iSW[8](shiftin)置 1，然后将 iSW[10](load)拨上，按 KEY[0]载入数据，再将iSW[10](load)拨下，按 KEY[0]数据右移一位。反复按，数据不断右移。iSW[9](aclr)按钮是重置按键。oLEDG[8](shiftout)与数据最右边一位是始终一致的。

下面是一个实验原理图的说明：

图 5-17 是一个信号逻辑概略图，其中不包括功能切换信号 iSW[10]。如果想要了解详细的逻辑信号布局，请使用菜单项 Tools->Netlist Viewers->Technology Map Viewer(Post Mapping)，如图 5-18。

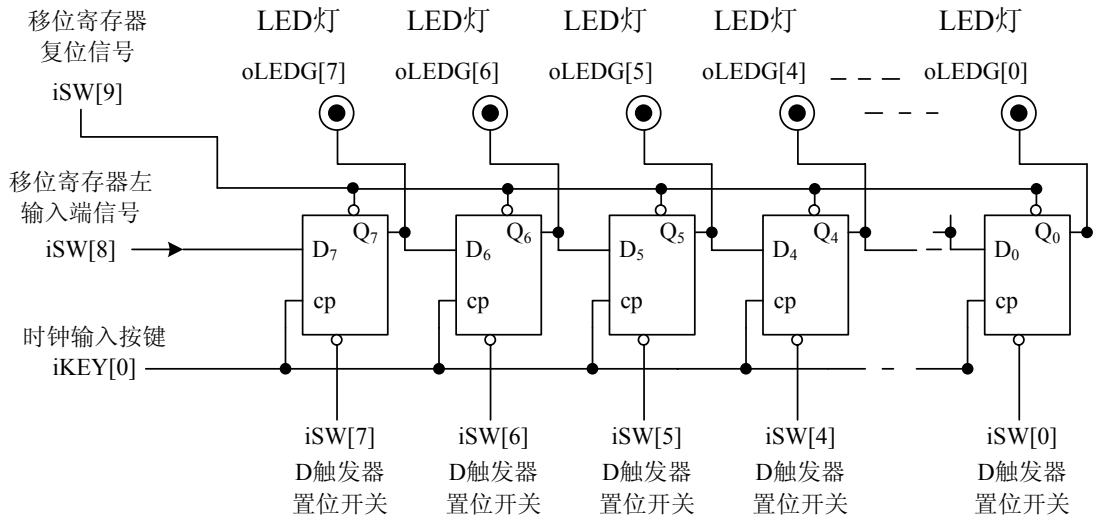


图 5-17　移位寄存器实验的信号逻辑概略图

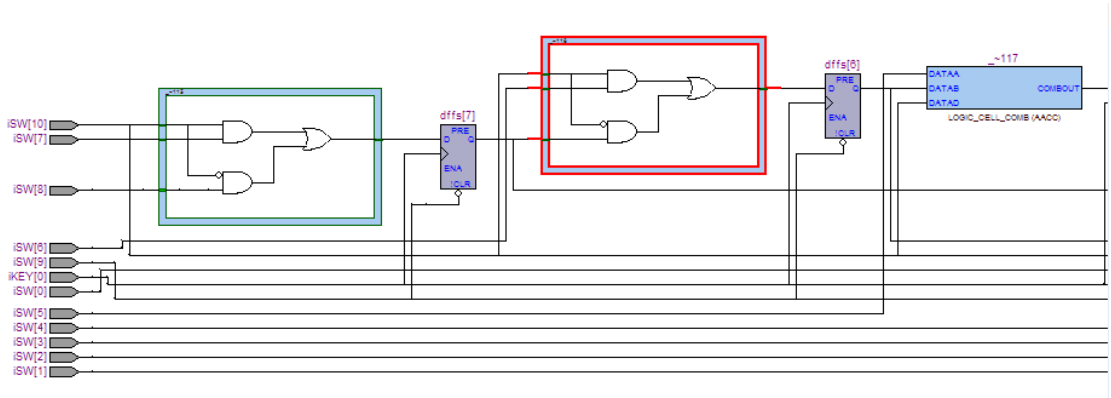


图 5-18　Technology Map Viewer(Post Mapping)（展开了左侧两个逻辑单元）

实验执行参考图 5-19、5-20 与图 5-21。

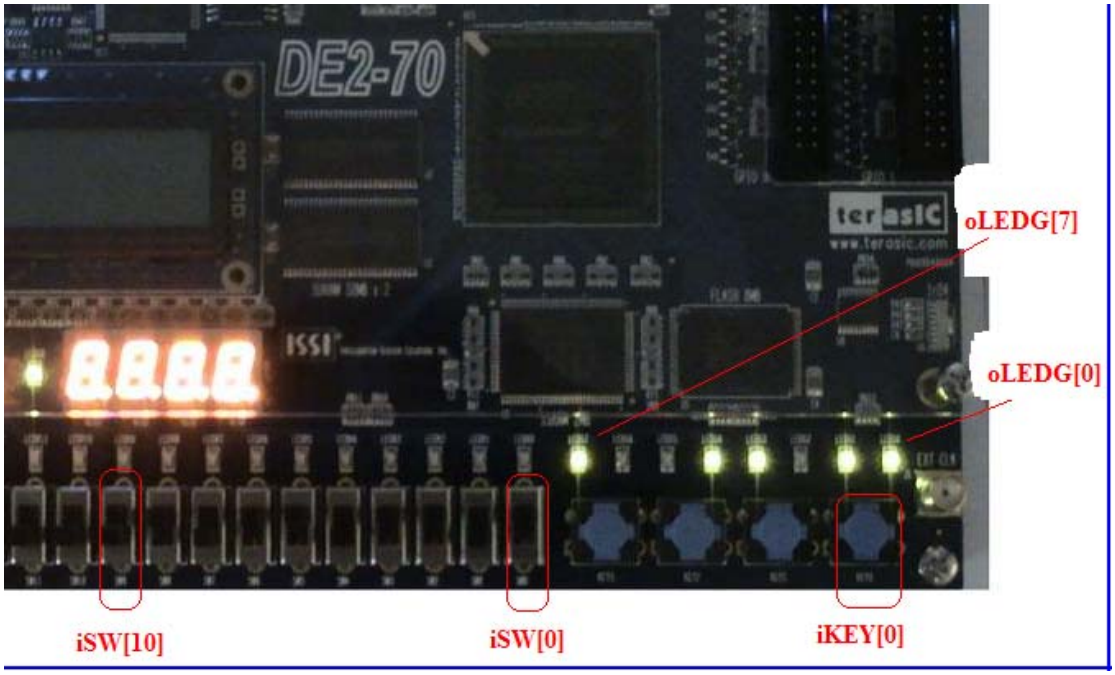


图 5-19　移位寄存器实验的运行指导图

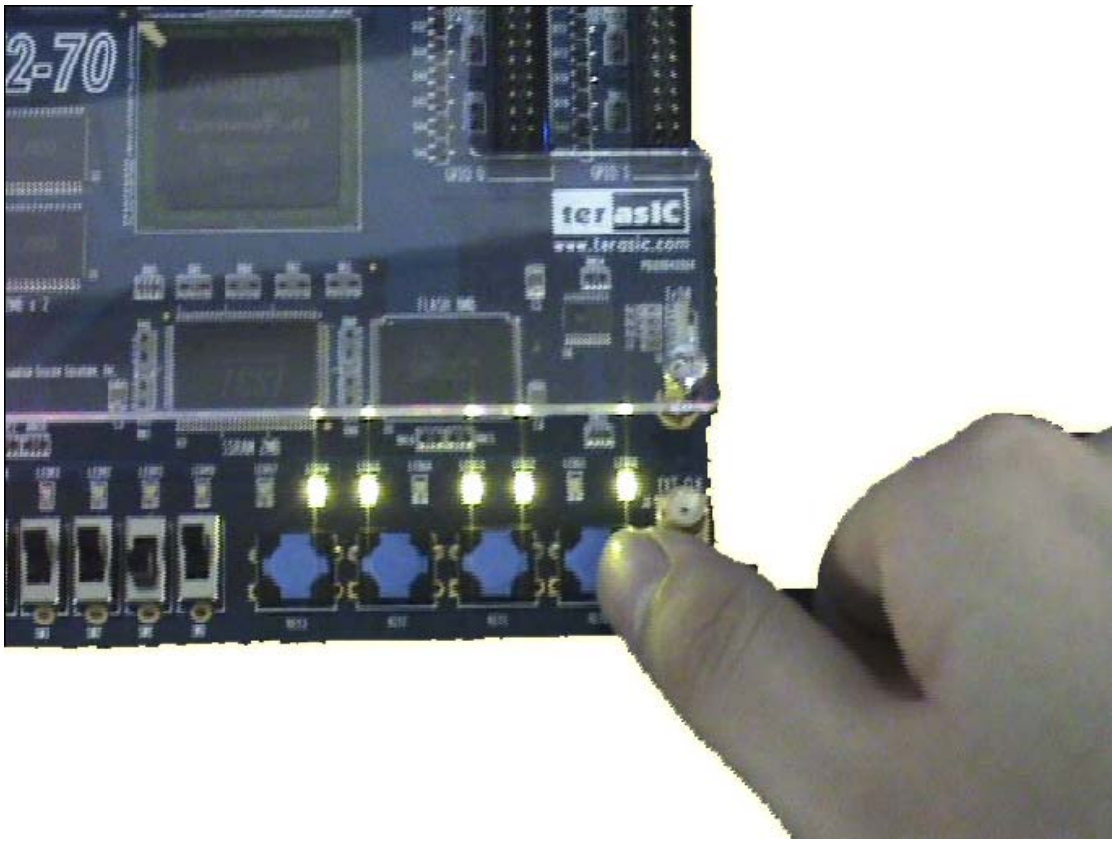


图 5-20　移位寄存器实验的运行效果图

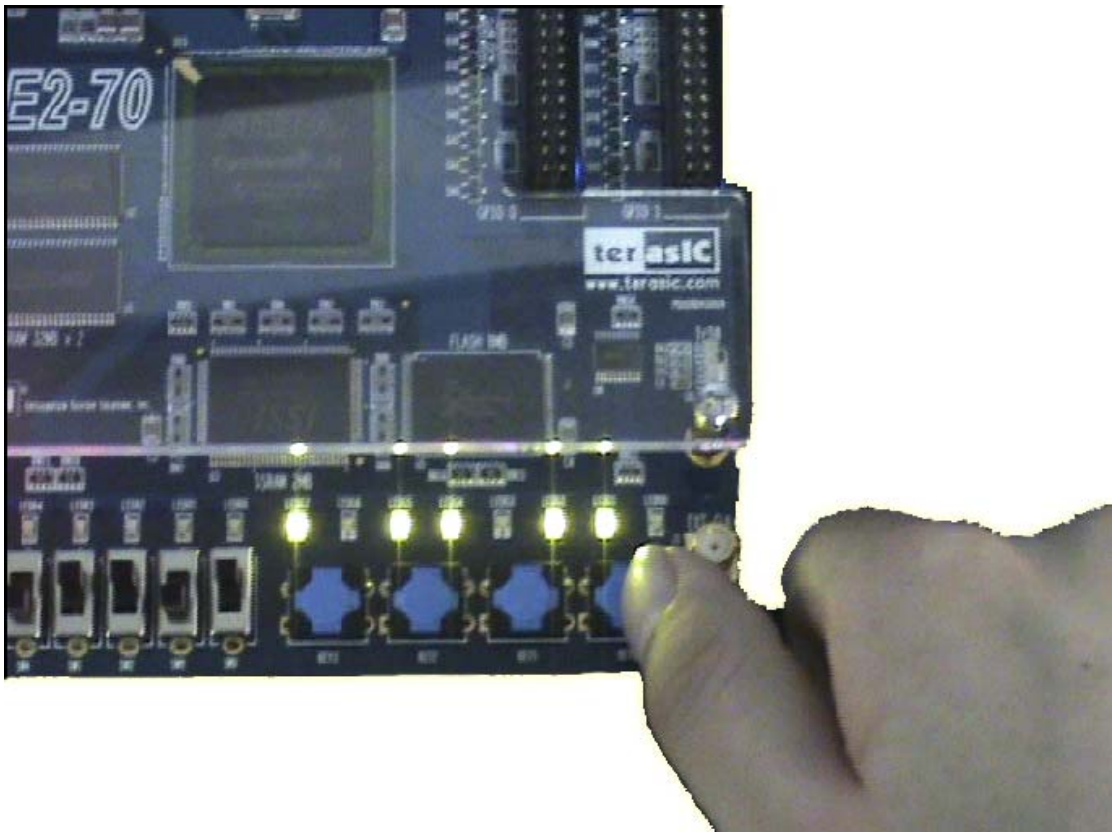


图 5-21　移位寄存器实验的运行效果图（右移一位，高位移入 1）

5.3 使用 Verilog 语言完成硬件描述设计

12. 下面通过 Verilog 代码实现同样功能的设计。

在同一个工程下新建一个 Verilog 源文件，输入如下代码，并保存 ShiftRegCore_2.v。

```
module ShiftRegCore_2(
    aclr，
    clock，
    data，
    load，
    shiftin，
    q，
    shiftout);

    input        aclr;
    input        clock;
    input    [7:0]   data;
    input        load;
    input        shiftin;
    output   reg [7:0]   q;
    output       shiftout;
always@(posedge clock or posedge aclr)
begin
if(aclr)q = 8'h0;
```

```
else
    begin
    if(load)q = data;
    else
    q = {shiftin，q[7:1]};
    end
end

assign shiftout = q[0];

endmodule
```

13. 在左侧 Project Navigator 处右击 ShiftReg_2.v 选择 Create Symbol Files for Current File。之后就可以像 ShiftReg 一样在顶层实体的符号框图里使用了，如图 5-22 与图 5-23。不过本实验不使用这种方法。

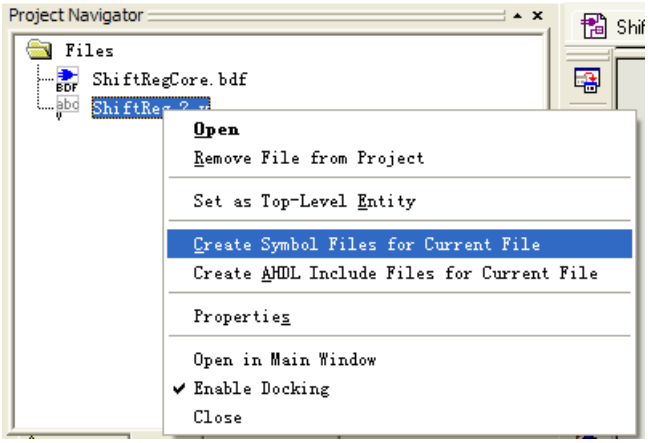


图 5-22　创建符号文件

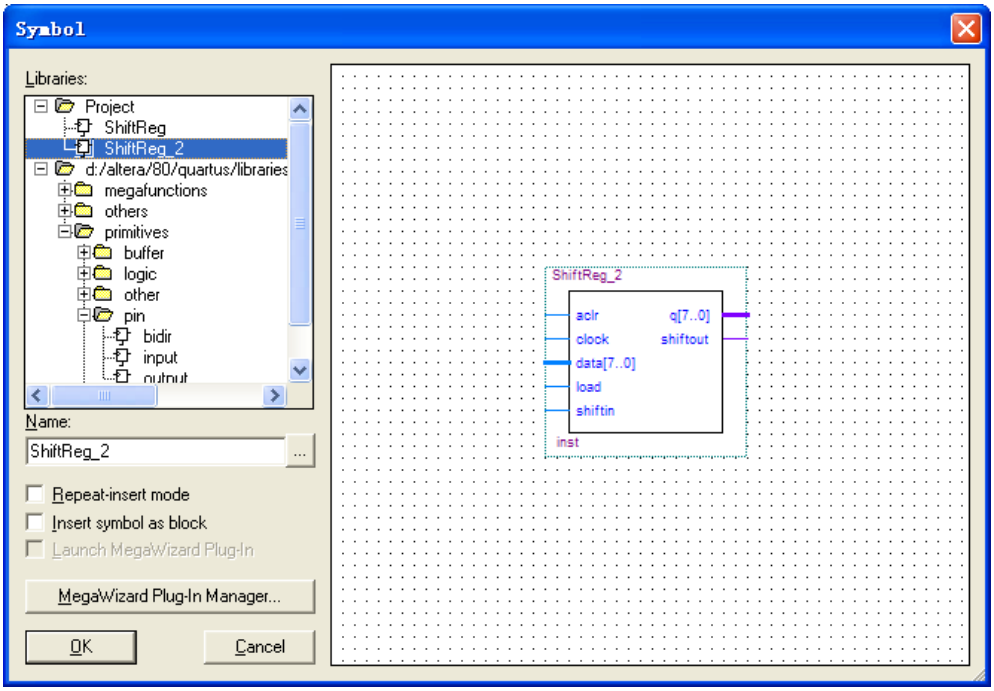


图 5-23　用 Verilog 语言创立的移位寄存器

14. 使用 Verilog 语言代替符号框图完成顶层实体。

新建一个 Verilog 文件添加以下代码，并设置其为顶层实体：

```
module ShiftRegCore_2
(
input [10:0] iSW，
input [0:0] iKEY，
output [8:0] oLEDG
);

ShiftReg_2 (
    .aclr(iSW[9])，
    .clock(iKEY[0])，
    .data(iSW[7:0])，
    .load(iSW[10])，
    .shiftin(iSW[8])，
    .q(oLEDG[7:0])，
    .shiftout(oLEDG[8])
    );
endmodule
```

15. 由于引脚名与符号框图一致，无须重新分配引脚，直接编译下载运行，结果应与使用符号文件设计的结果一致。

16. Tools->Netlist Viewers 观察，可以发现二者在 RTL Viewer 中的结构不同，但在 Technology Map Viewer (post-mapping)中的结构是一致的。

第 6 章　实验五　LCD 显示实验

- 实验说明

该实验在 LCD 上滚动显示“Hello from Nios II”。实验简单介绍了 SOPC Builder 与 Nios II IDE 的使用。通过该实验，使读者大致了解 SOPC 的建立过程，明白如何在 DE2-70 上跑简单的 C 程序。

- 实验步骤

6.1 建立 Quartus 工程

1. 新建 Quartus 工程 HelloWorld，顶层实体名 HelloWorld
2. 重新设置编译输出目录为../ HelloWorld/release。

6.2 建立 SOPC 系统

3. 点击 Quartus II 软件右上方图标打开 SOPC Builder，创建一个 SOPC 系统。填写系统名称为 HelloWorld_System，并指定 Verilog 为描述系统的语言（注意：在添加系统名称时不可以与工程名相同）如图 6-1。

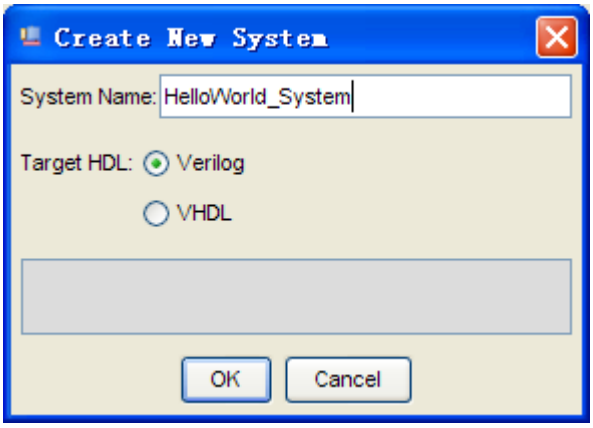


图 6-1　添加系统名称并指定语言

4. 在系统上添加 On-Chip Memory。

在程序左侧列表中选择 Memory and Memory Controllers -> On-Chip -> On-Chip Memory (RAM or ROM)，双击添加至系统中。如图 6-2 所示。

注意：左侧的组件均可以根据需要添加，8.0 以上的 SOPC Builder 具有过滤器，可以快速定位。

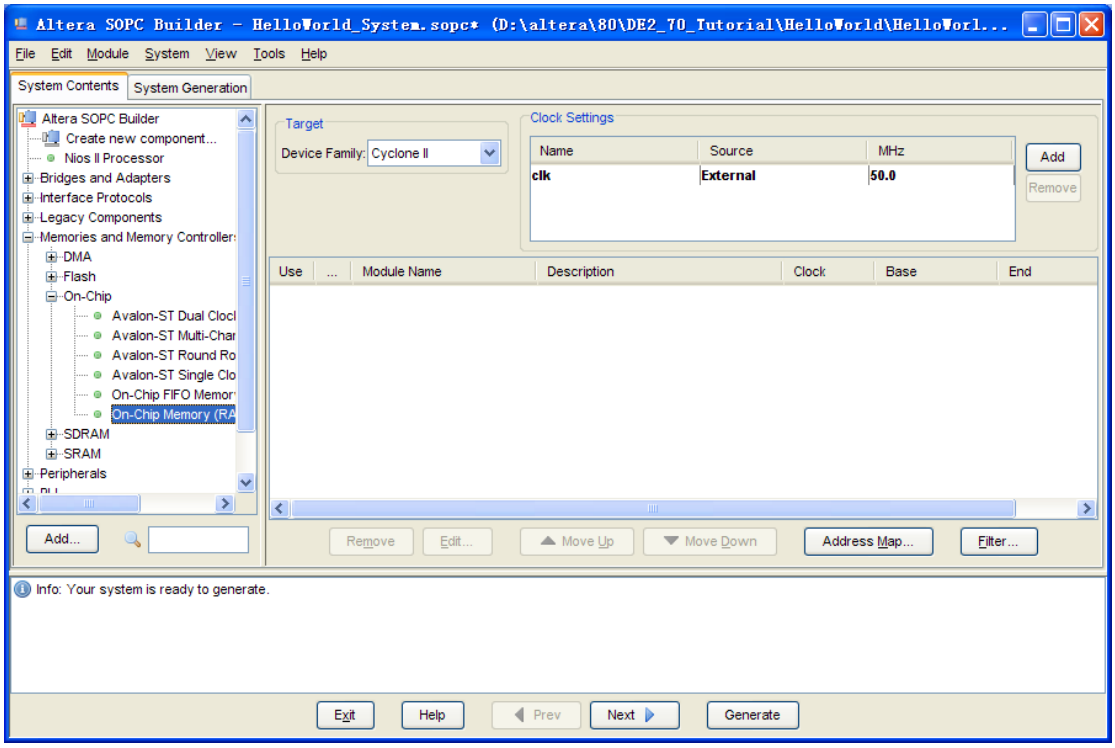


图 6-2 添加 On-Chip Memory

在弹出的对话框中指定片上 RAM 的属性。**主要是片上存储器的大小问题，太小的情况下甚至无法编译 HelloWorld。**对于 Quartus II 8.0 自带的 HelloWorld 模板编译结果官方给出的大小是~69K。如果用编译器参数-Os 优化，最终可以优化到 22K，这里把 On-Chip Memory 指定到 30K，如图 6-3，配置好后点击 Finish.。

注意：如果实在执着于空间限制，请在 C 语言编程时使用 alt 库。

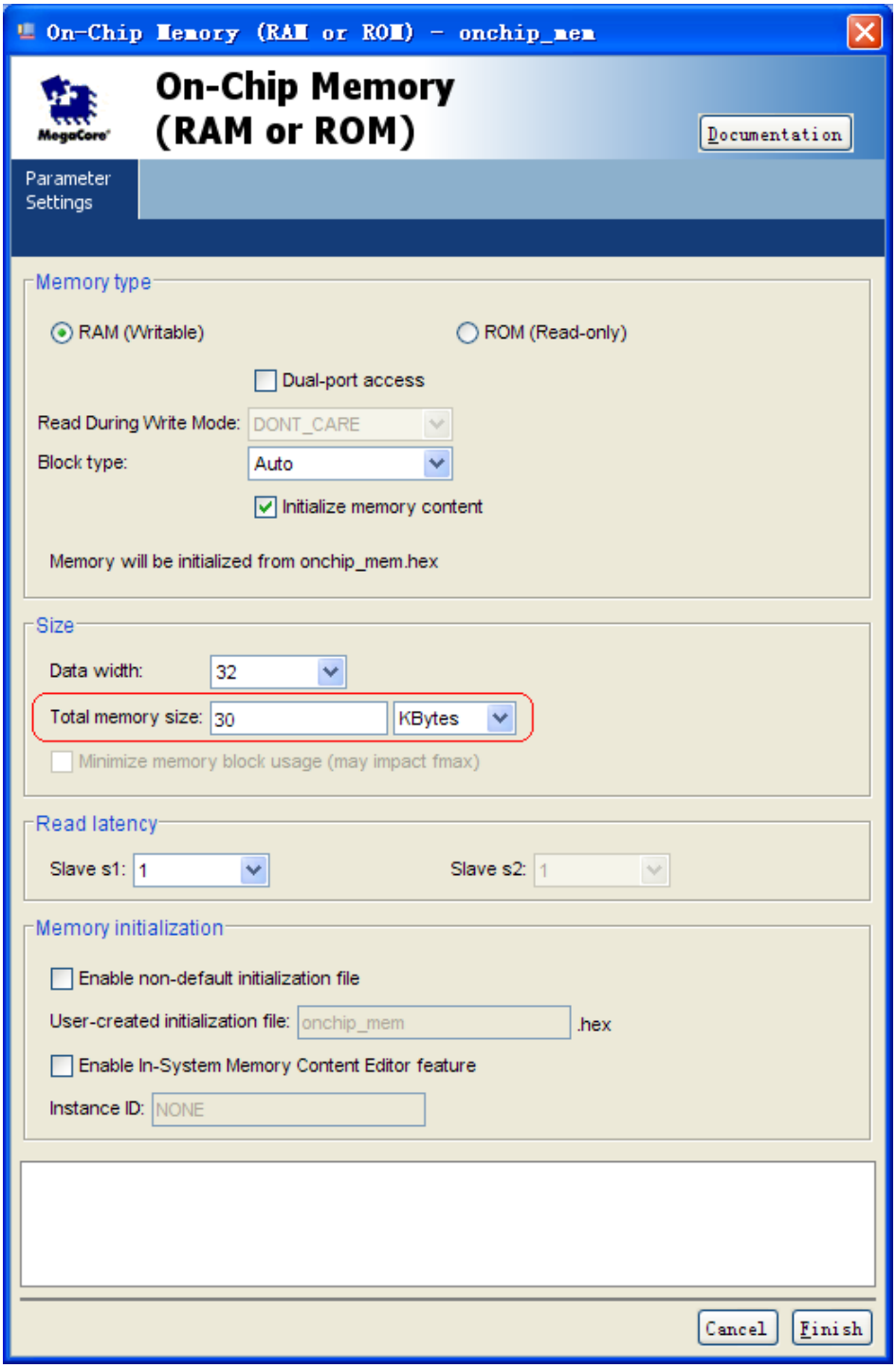


图 6-3 指定 On-Chip Memory 属性

5. 添加 Nios II Processor。

双击 Altera SOPC Builder -> Nios II Processor，在弹出的对话框中间选择第一个 Nios II/e，表示 economy，最小的 NIOS II 核心。下面的 Reset Vector 和 Exception Vector 都选择 onchip_men，即刚才添加的片上 RAM 的名称。其它的都保留默认设置即可。点击 Finish 添加 CPU 核。如图 6-4。

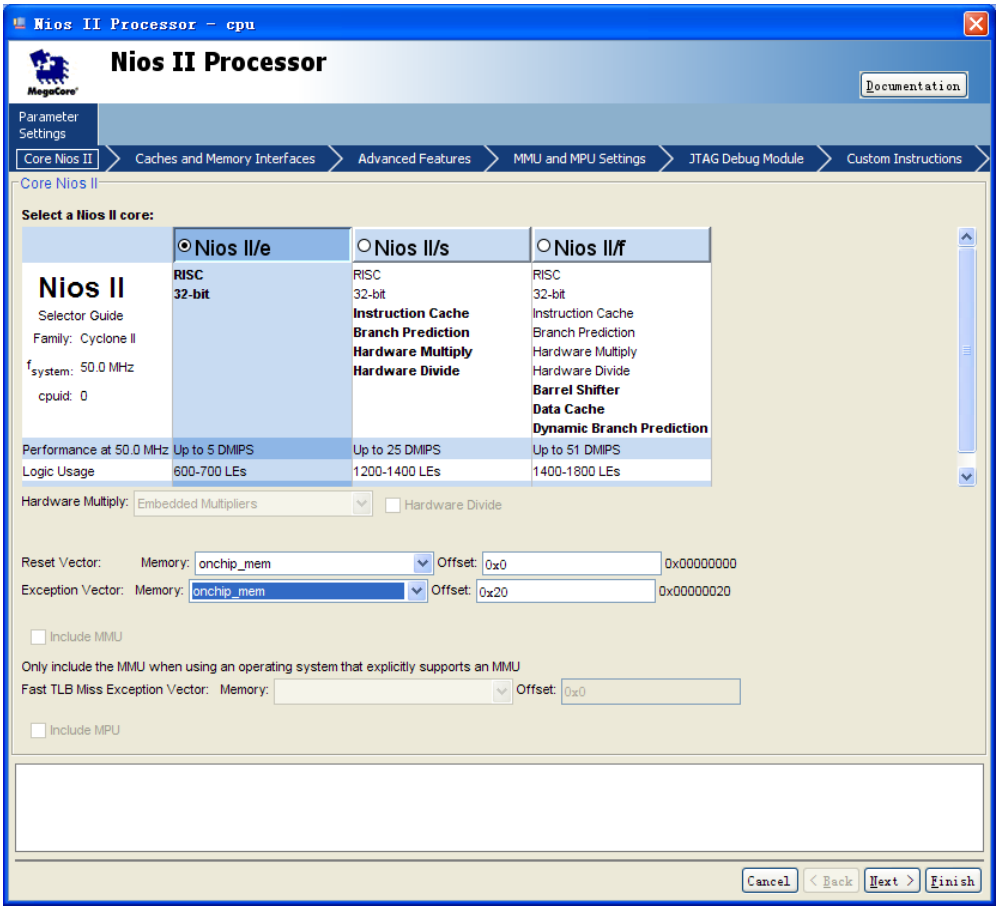


图 6-4　添加 CPU 并设置参数

6.　添加 JTAG UART，默认即可。如图 6-5。

图 6-5　添加 JTAG UART 并设置参数

7.　添加 LCD，保持默认。如图 6-6。

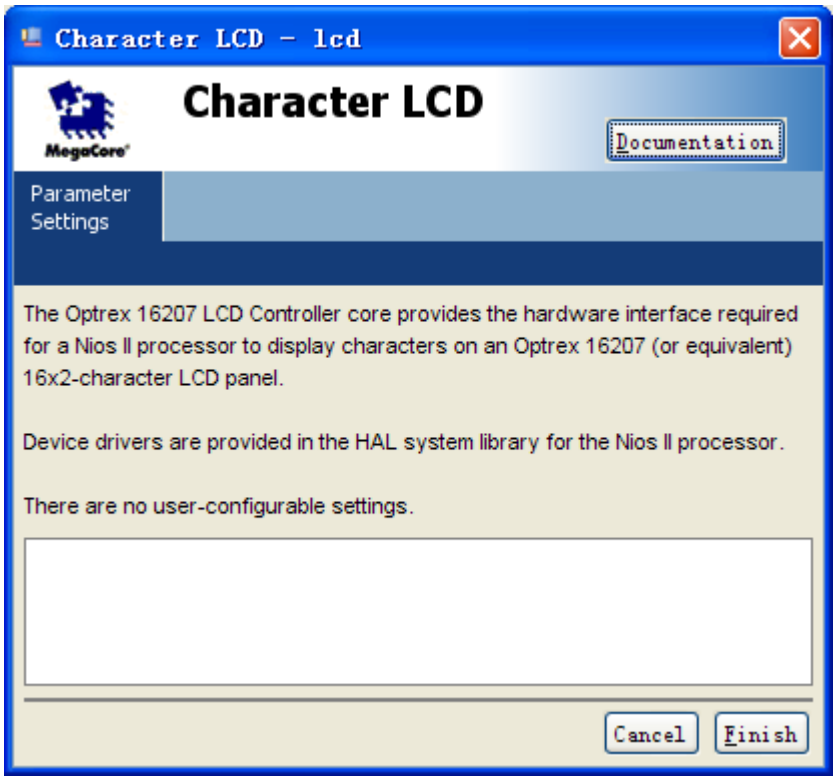


图 6-6 添加 LCD

8. 完成 SOPC 工程设计，如图 6-7。

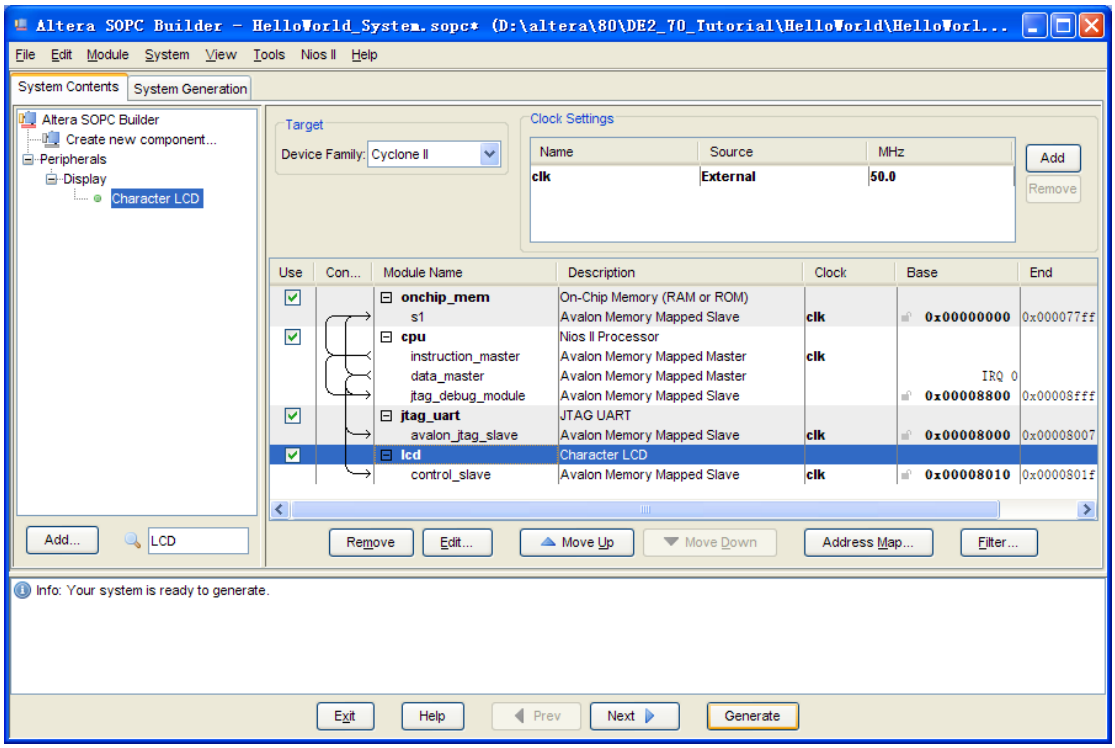


图 6-7 完成的 SOPC 工程

9. 生成系统。通过点击下方 Generate 完成，如图 6-8。

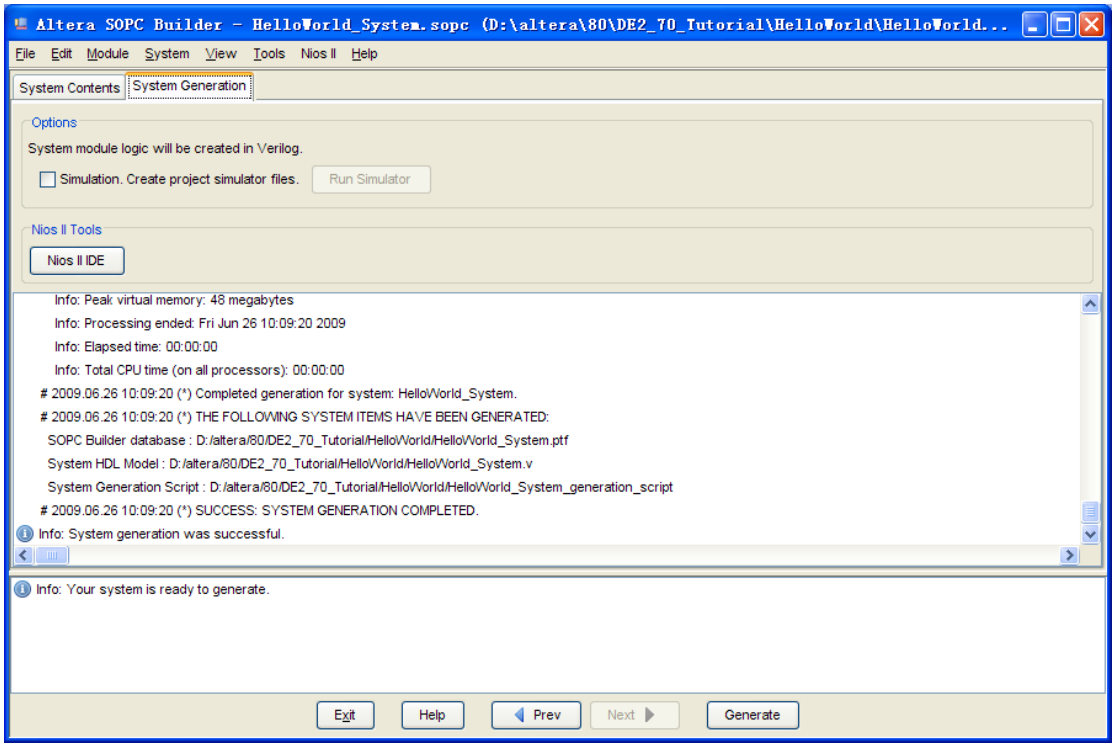


图 6-8 生成系统

6.3 用 Verilog 语言完成顶层实体

10. 大多数情况下，都选择使用语言完成顶层实体，因为引脚太多，使用符号框图容易出错。退出 SOPC Builder。将如下代码保存至 HelloWorld.v：

```
module HelloWorld
(
iCLK_50，
oLCD_ON，
oLCD_BLON，
oLCD_RW，
oLCD_EN，
oLCD_RS，
LCD_D
);

input iCLK_50;
inout [7:0] LCD_D;
output oLCD_ON;
output oLCD_BLON;
output oLCD_RW;
output oLCD_EN;
output oLCD_RS;

assign oLCD_ON = 1'b1;
assign oLCD_BLON = 1'b1;
```

```
HelloWorld_System u1(
.clk(iCLK_50)，
.reset_n(1)，
.LCD_E_from_the_lcd(oLCD_EN)，
.LCD_RS_from_the_lcd(oLCD_RS)，
.LCD_RW_from_the_lcd(oLCD_RW)，
.LCD_data_to_and_from_the_lcd(LCD_D)
);
endmodule
```

11. 分析与综合，这一步除了检查顶层实体是否出错外，尚有另一个目的，就是为了将已有配置信息写入 qsf 文件。以免分配引脚时造成 qsf 文件的修改时间比内存更新，丢失以往配置信息(如编译输出路径)。

12. 引脚分配，编辑 qsf 文件添加引脚，结果如图 6-9。

```
set_location_assignment PIN_B2 -to LCD_D[7]
set_location_assignment PIN_C3 -to LCD_D[6]
set_location_assignment PIN_C2 -to LCD_D[5]
set_location_assignment PIN_C1 -to LCD_D[4]
set_location_assignment PIN_D3 -to LCD_D[3]
set_location_assignment PIN_D2 -to LCD_D[2]
set_location_assignment PIN_E3 -to LCD_D[1]
set_location_assignment PIN_E1 -to LCD_D[0]
set_location_assignment PIN_AD15 -to iCLK_50
set_location_assignment PIN_G3 -to oLCD_BLON
set_location_assignment PIN_E2 -to oLCD_EN
set_location_assignment PIN_F1 -to oLCD_ON
set_location_assignment PIN_F2 -to oLCD_RS
set_location_assignment PIN_F3 -to oLCD_RW
```

	Node Name	Direction	Location	I/O Bank	Vref Group	I/O Standard
1	LCD_D[7]	Unknown	PIN_B2	2	B2_N0	3.3-V LVTTL (default)
2	LCD_D[6]	Unknown	PIN_C3	2	B2_N0	3.3-V LVTTL (default)
3	LCD_D[5]	Unknown	PIN_C2	2	B2_N0	3.3-V LVTTL (default)
4	LCD_D[4]	Unknown	PIN_C1	2	B2_N0	3.3-V LVTTL (default)
5	LCD_D[3]	Unknown	PIN_D3	2	B2_N1	3.3-V LVTTL (default)
6	LCD_D[2]	Unknown	PIN_D2	2	B2_N1	3.3-V LVTTL (default)
7	LCD_D[1]	Unknown	PIN_E3	2	B2_N0	3.3-V LVTTL (default)
8	LCD_D[0]	Unknown	PIN_E1	2	B2_N1	3.3-V LVTTL (default)
9	iCLK_50	Unknown	PIN_AD15	7	B7_N3	3.3-V LVTTL (default)
10	oLCD_BLON	Unknown	PIN_G3	2	B2_N0	3.3-V LVTTL (default)
11	oLCD_EN	Unknown	PIN_E2	2	B2_N1	3.3-V LVTTL (default)
12	oLCD_ON	Unknown	PIN_F1	2	B2_N2	3.3-V LVTTL (default)
13	oLCD_RS	Unknown	PIN_F2	2	B2_N2	3.3-V LVTTL (default)
14	oLCD_RW	Unknown	PIN_F3	2	B2_N0	3.3-V LVTTL (default)
15	<<new node>>					

图 6-9 引脚分配图

13. 编译下载。

6.4 软件设计

14. 打开 NIOS II IDE 软件。第一次打开的时候会提示选择工作空间。也可在程序打开后选择菜单栏 File -> Switch Workspace…以选择一个合适的工作空间。如果仅仅在一台电脑上做开发，选择默认工作空间即可，但如果需要经常移动开发平台，最好还是保存到工程目录下，和软件源码放在一起。这里我们选择..\HelloWorld\Software，如图 6-10 所示。

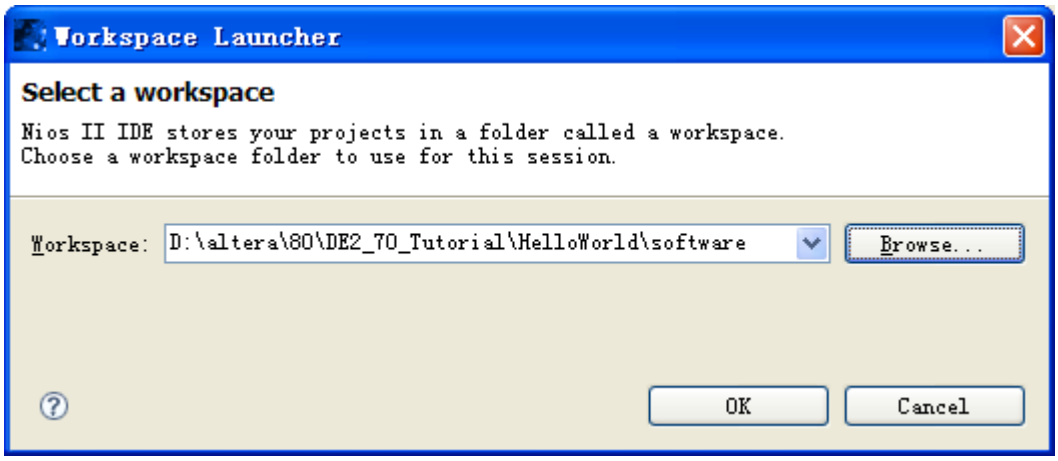


图 6-10　工作区选择

15. 自动重启 Nios II IDE 后，会进入 Nios II IDE 的主界面，如图 6-11。

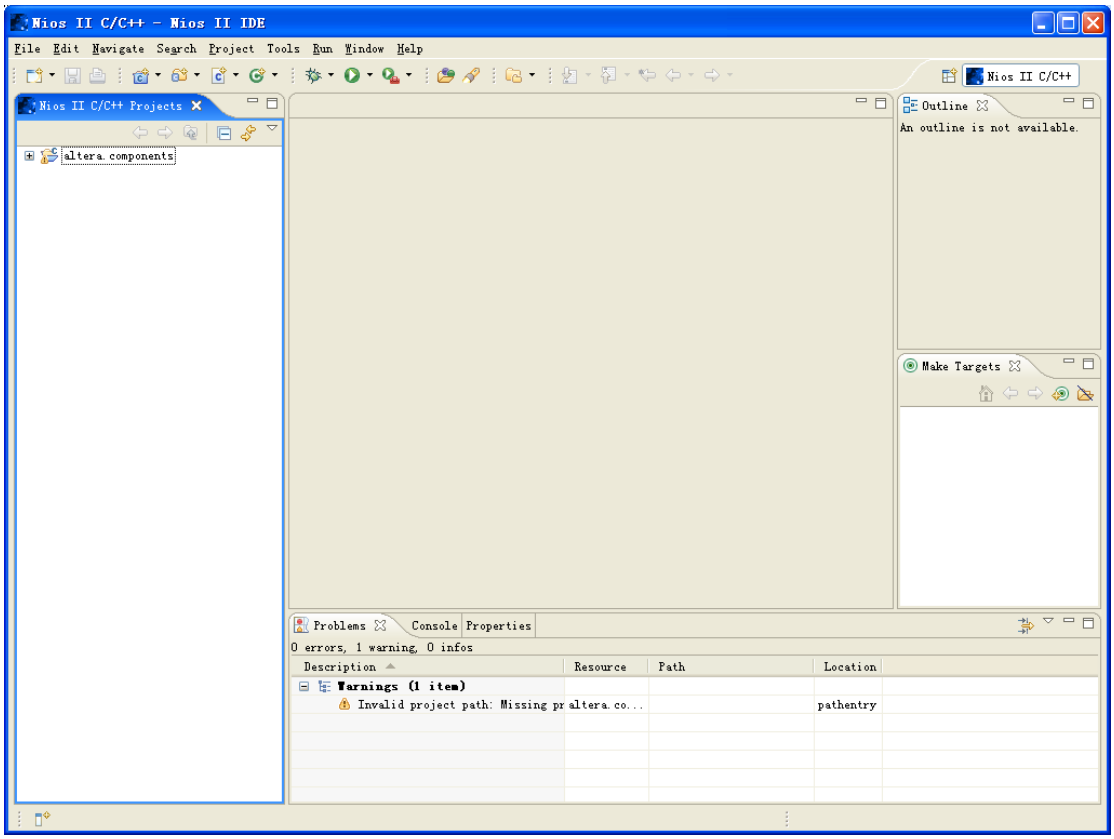


图 6-11　Nios II IDE 的主界面

16. 选择 “File->New->Nios II C/C++ Application”，左侧选择 HelloWorld 模板，右侧选择好 SOPC 系统对应的 ptf 文件，Finish，参看图 6-12。

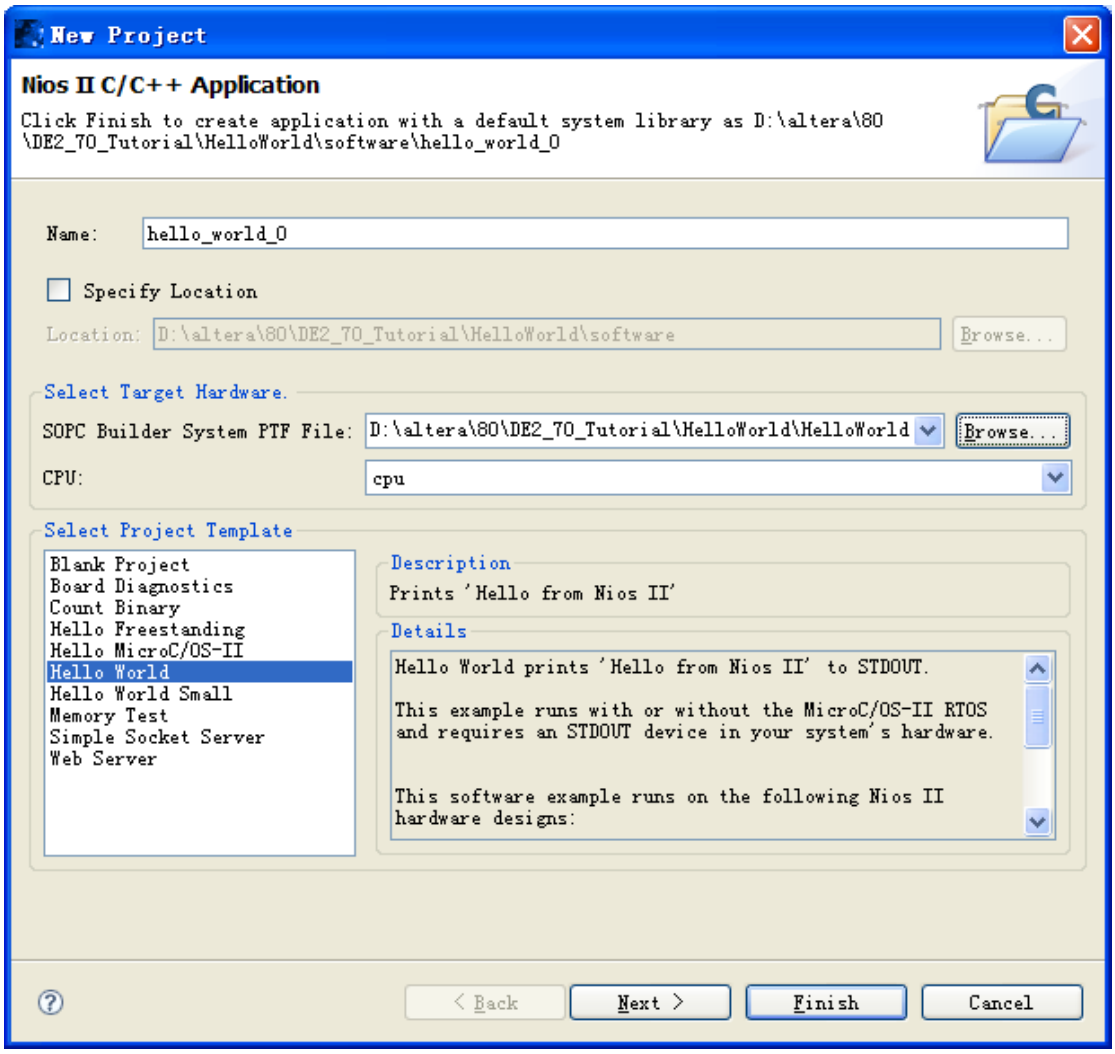


图 6-12 新建工程

17. 在左侧 hello_world_0 工程处，右击选择 System Library Properties，如图 6-13。

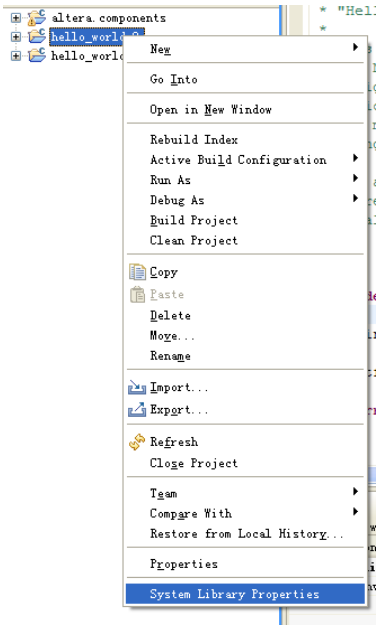


图 6-13 选择 System Library Properties

配置 System Library Properties，将 stdin，stdout，stderr 三项全部设为 lcd，在下图所示界面中，取消掉 Clean Exit (Flush Buffer)和 Support C++前的勾，因为我们的程序不会退出，也不包含 C++的函数和库。选中 Program never exits，Reduce device drivers 和 Small C library 以减小程序体积。其他保持默认设置即可，如图 6-14。

注意 Reduced device drivers 的勾绝对不能勾，否则 lcd 无响应。

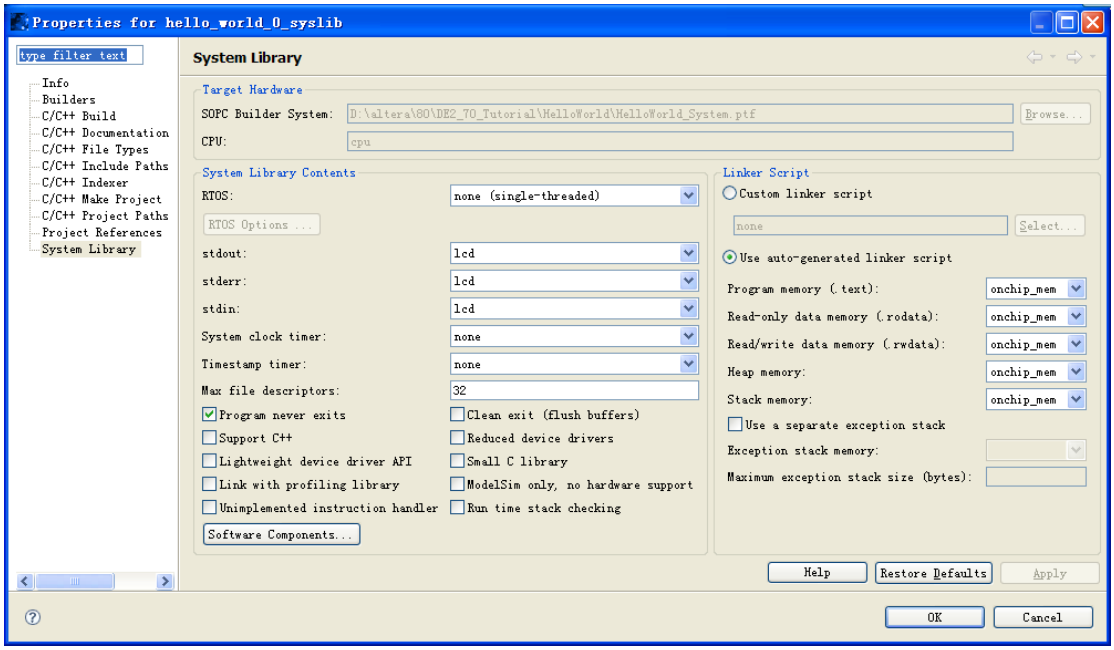


图 6-14 配置 System Library Properties

18. Project->Build All，很遗憾得到了一个内存已满的结果，如图 6-15.

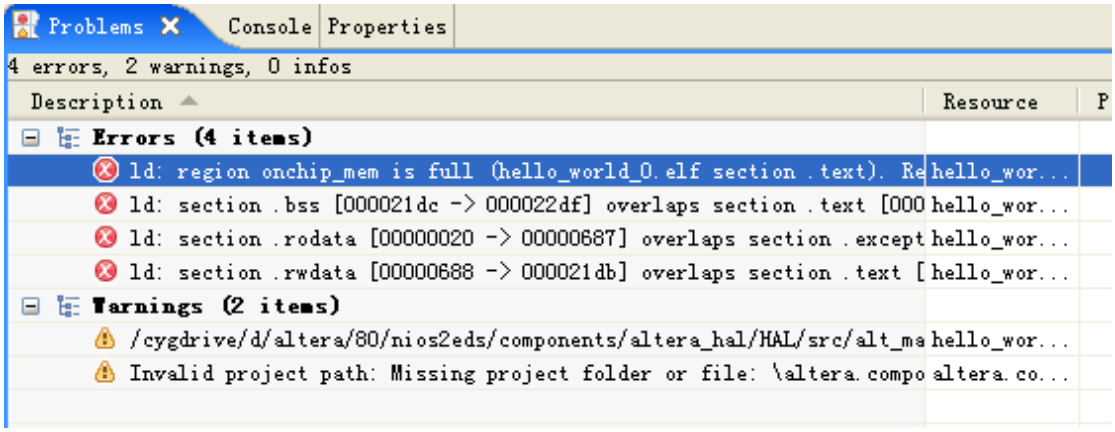


图 6-15 编译失败

19. Clean 掉刚编译的工程，Project->clean，去掉 Start a build immediately 的勾防止立即再 build。

20. 依次打开两个工程(一个应用工程与一个系统库工程)的 Properties，选中 C/C++ Build 选项卡，将 Configuration 设为 Release，编译器参数设为-Os，如图 6-16。

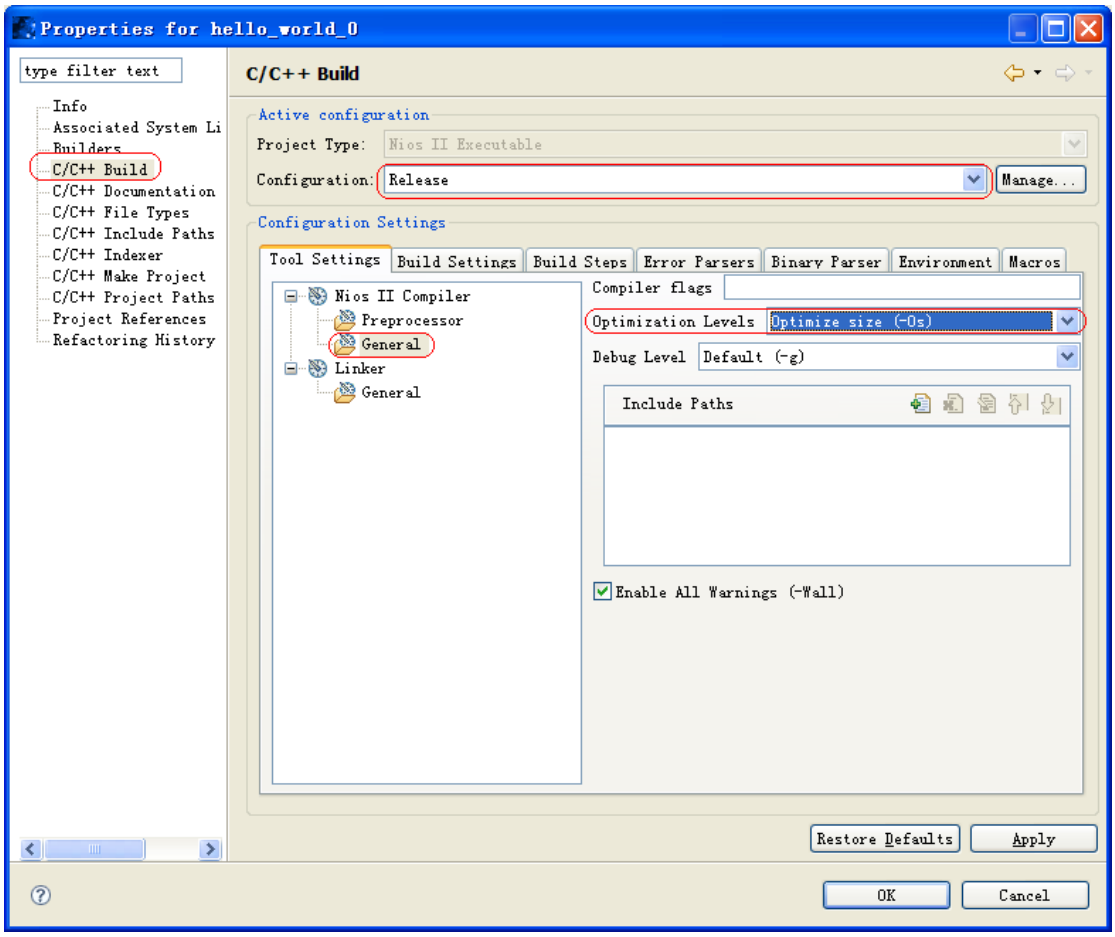


图 6-16　编译器参数

21. Project->Build All，编译结果为 22K

22. 去 System Library Properties 勾上 Small C Library，再次编译，程序只有 12K 了。

23. 在 hello_world_0 工程上右击，选择 Run As ->Nios II Hardware，如图 6-17。

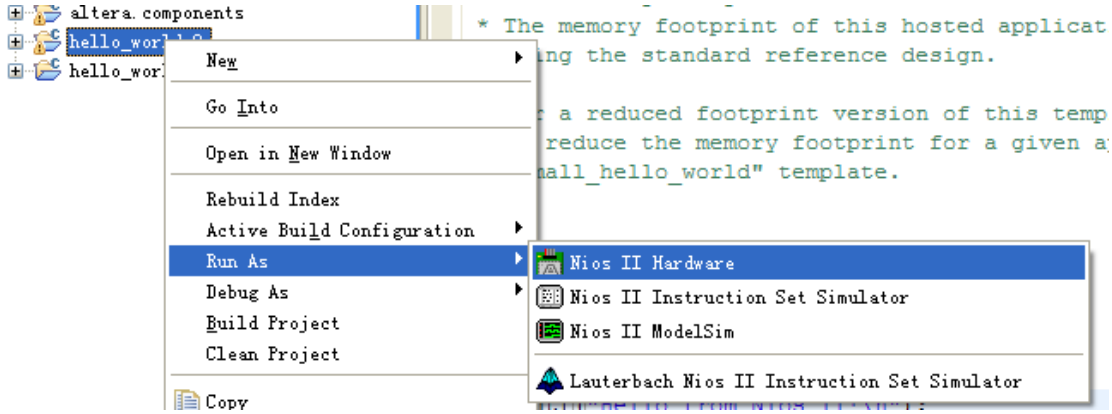


图 6-17 运行 C 代码

24. 查看实验板上的结果

运行结果为在 LCD 上显示：Hello from Nios，后面的 II 不见了

25. 对于较长字符串，DE2-70 的常见做法是在 lcd 屏上滚动显示。方法是加入一个 Interval timer。

26. 首先在 SOPC 系统中添加 Interval Timer，配置保持默认，如图 6-18。之后 generate。

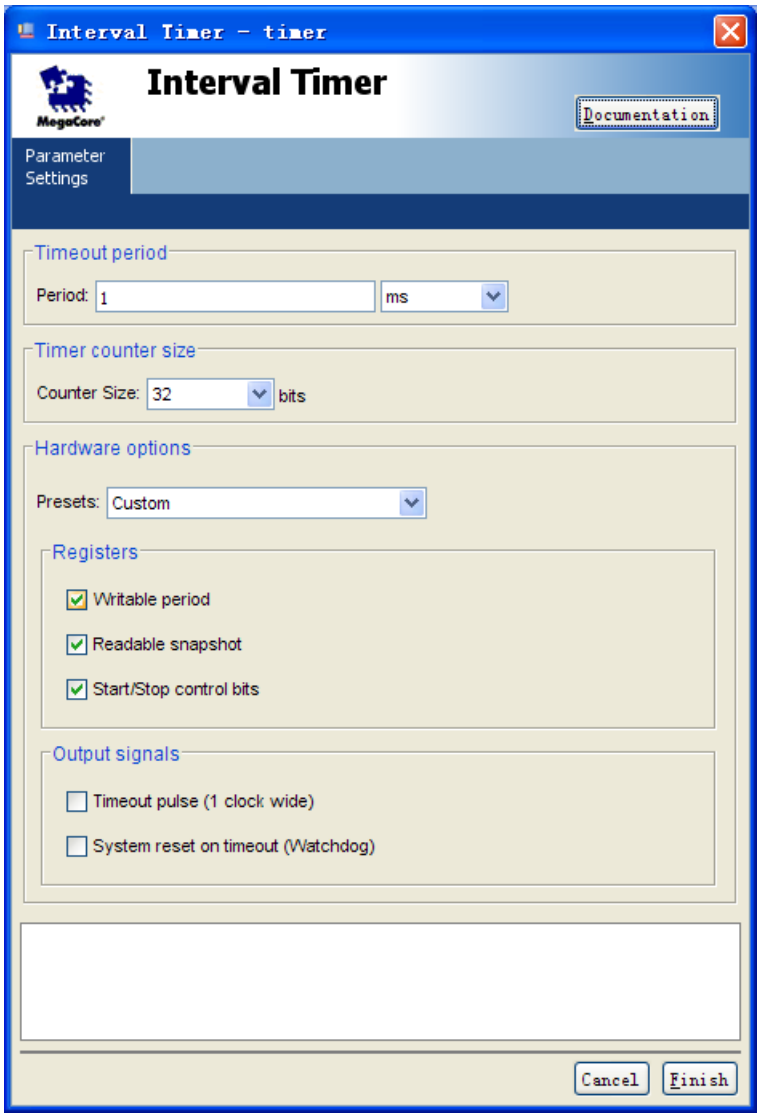


图 6-18 Interval Timer

27. 然后在工程的 System Library Properties 中选中那个 timer 作为 System clock timer，如图 6-19。

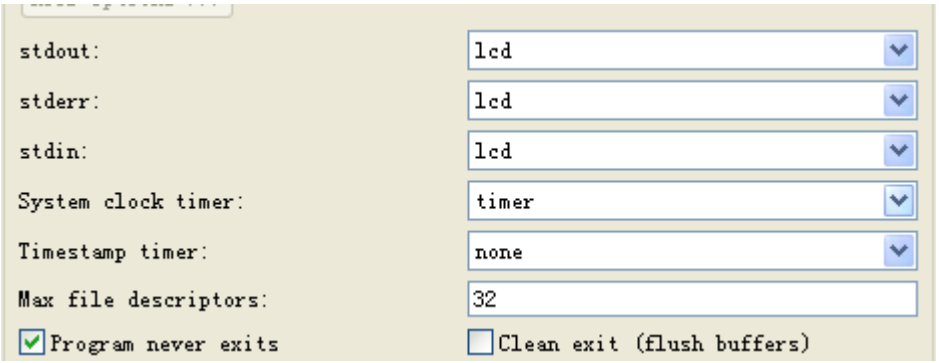


图 6-19 设定 Timer

28. 编译运行，即可发现滚动的 Hello from Nios II。

第 7 章　实验六　跑马灯实验

- 实验说明

  该实验完成的是跑马灯的设计。
- 实验步骤

7.1 建立 Quartus 工程

1. 新建 Quartus 工程 RunningLED，顶层实体名 RunningLED
2. 重新设置编译输出目录为../ RunningLED/release。

7.2 建立 SOPC 系统

3. 点击 Quartus II 软件右上方图标打开 SOPC Builder，创建一个 SOPC 系统。填写系统名称为 RinningLED_System，并指定 Verilog 为描述系统的语言，如图 7-1。

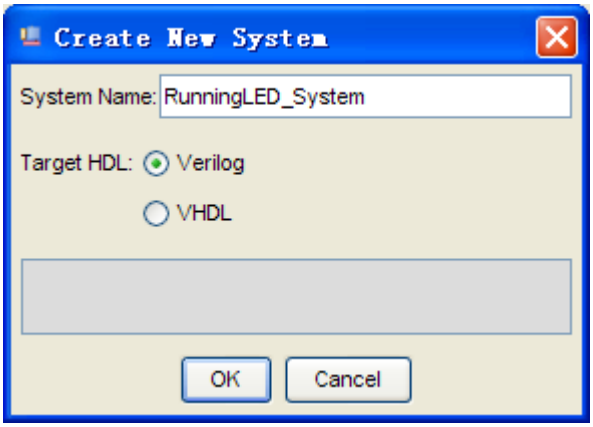


图 7-1 添加系统名称并指定语言

4. 在系统上添加 On-Chip Memory。

在程序左侧列表中选择 Memory and Memory Controllers -> On-Chip -> On-Chip Memory (RAM or ROM)，双击添加至系统中。

在弹出的对话框中指定片上 RAM 的属性，因为不需要显示，编译结果很小，保持默认即可。

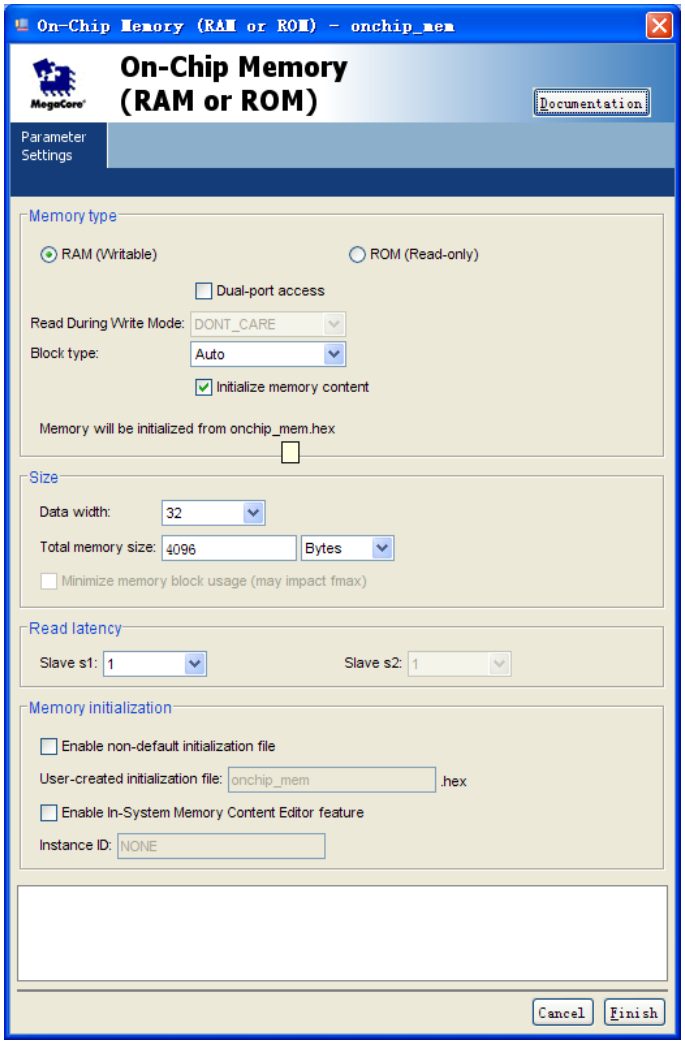


图 7-2 指定 On-Chip Memory 属性

5. 添加 Nios II Processor。

双击 Altera SOPC Builder -> Nios II Processor，在弹出的对话框中间选择第一个 Nios II/e，表示 economy，最小的 NIOS II 核心。下面的 Reset Vector 和 Exception Vector 都选择 onchip_men，即刚才添加的片上 RAM 的名称。其它的都保留默认设置即可。点击 Finish 添加 CPU 核。如图 7-3。

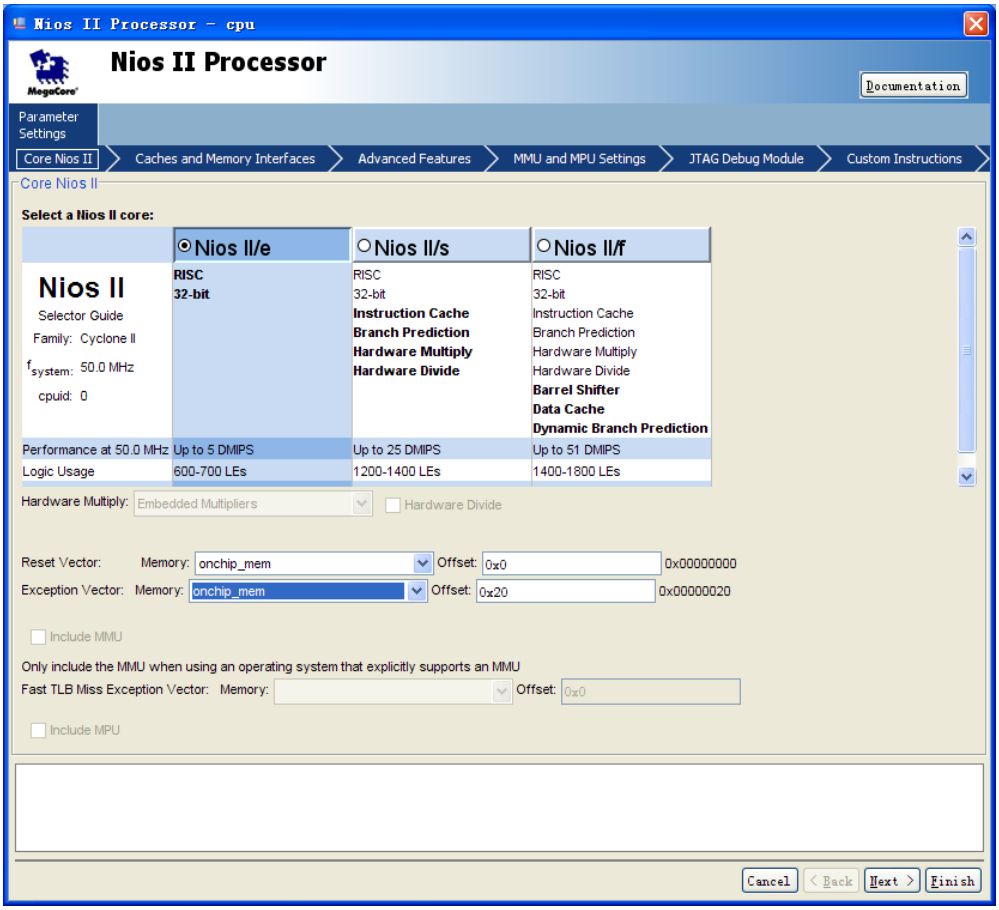


图 7-3 添加 CPU 并设置参数

6. 添加定时器。

在列表中选择 Peripherals -> Microcontroller Peripherals -> Interval Timer，弹出如下对话框。定时器在本系统中主要作用是产生一个固定间隔的中断信号，让 CPU 改变 LED 灯的状态。因此在 Period 中选择 500ms，表示灯的状态每 500ms 改变一次。Preset 选择 Simple periodic interrupt 即可。如图 7-4。

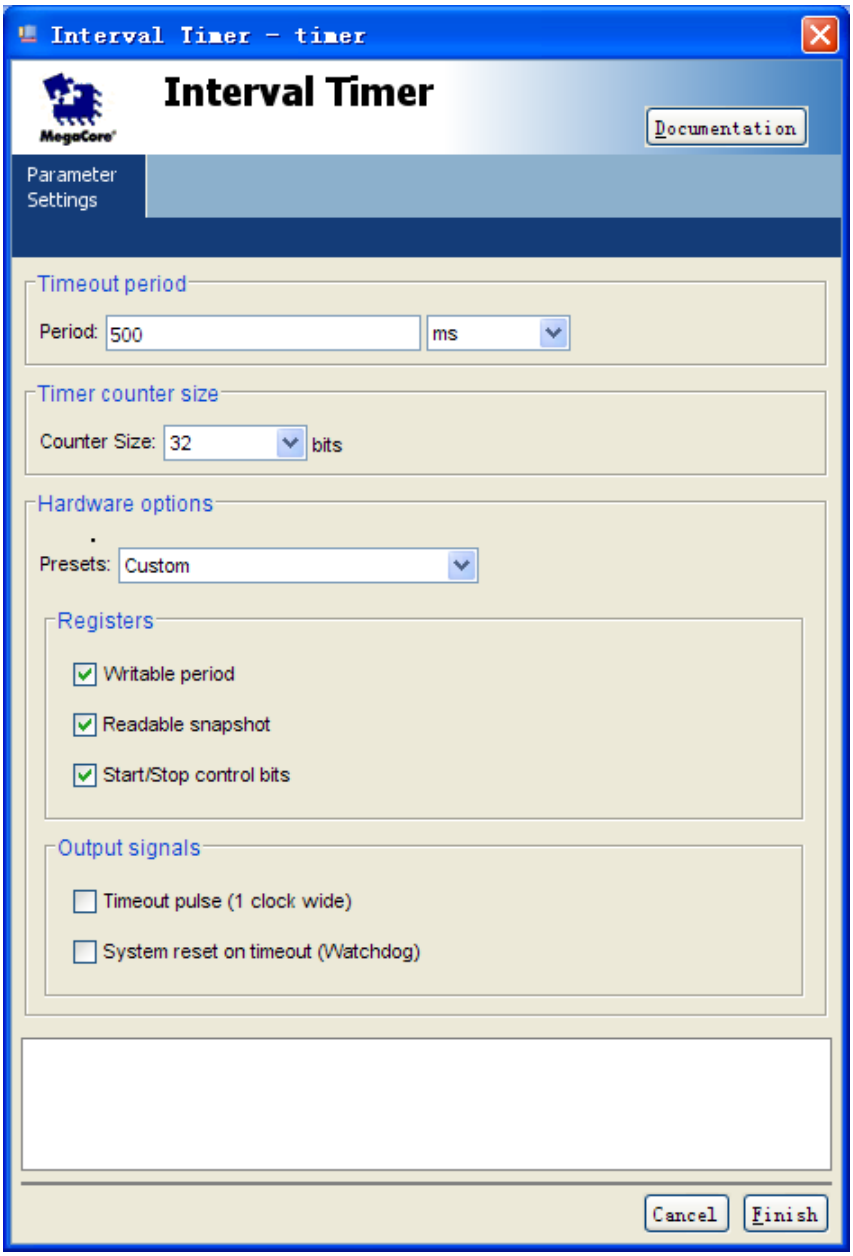


图 7-4 添加定时器并设置参数

7. 添加 IO 控制器。

双击 Peripherals -> Microcontroller Peripherals -> PIO (Parallel I/O)，保持默认设置即可，表示有 8 个输出用 IO 口，分别控制开发板上的 8 个绿色 LED 灯（LEDG[7..0]）。如图 7-5。

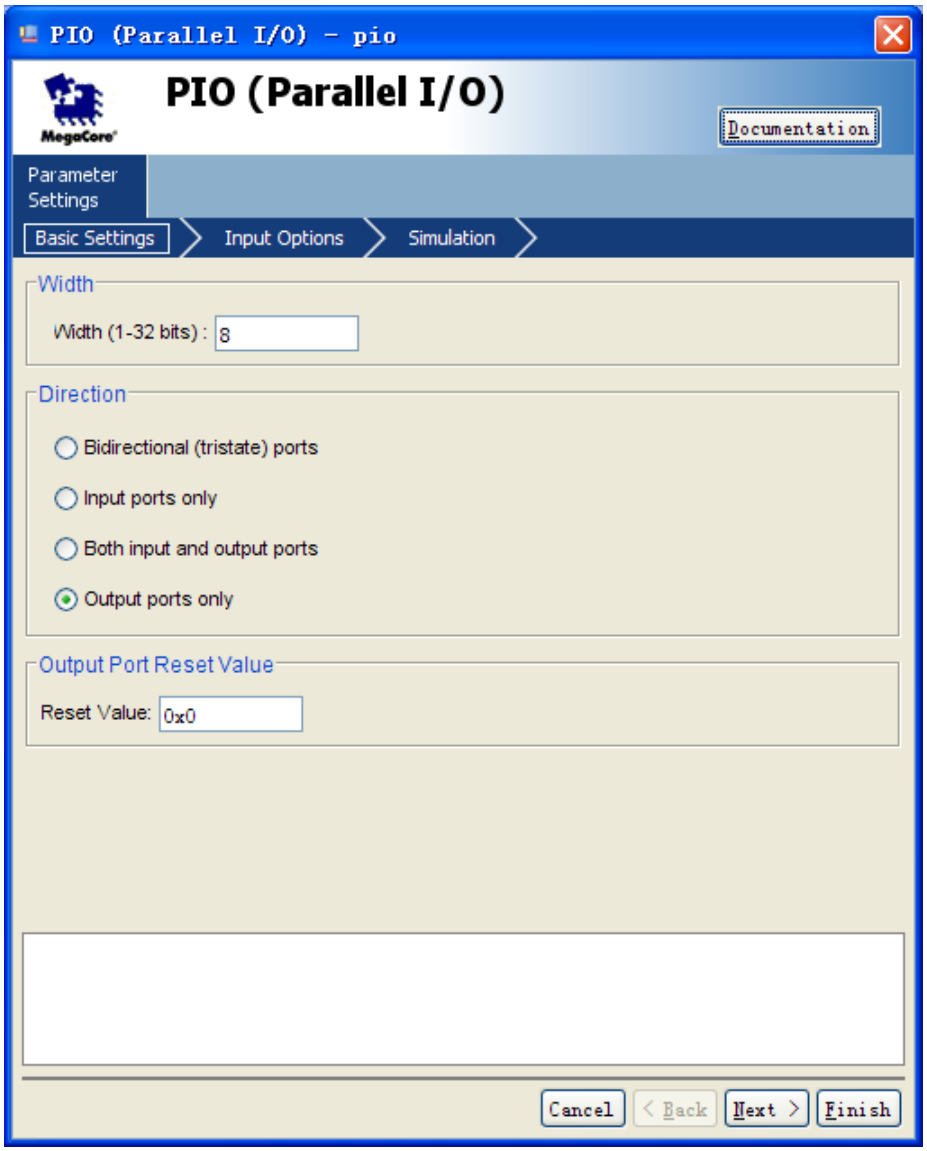


图 7-5 添加 IO 控制器并设置参数

8.　完成 SOPC 工程设计，为了方便将 PIO 的名称改为 pio_ledg。在 pio 上点击右键 -> rename，将名称改为 pio_ledg。如图 7-6。

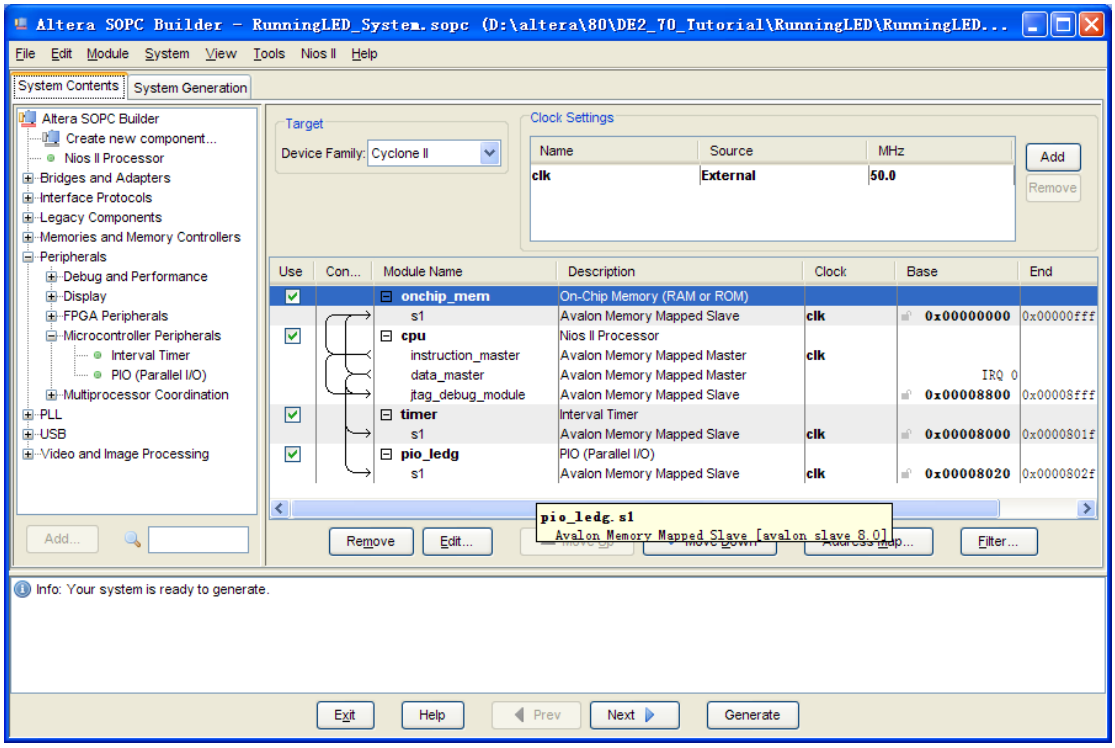


图 7-6 完成的 SOPC 工程

注意：系统的每个组件都需要一个地址才能正常工作。某些组件，如定时器（Interval Timer）还需要分配一个 IRQ 号。如果发现各组件的地址或者 IRQ 号出现冲突，可以选择菜单栏上 System -> Auto-Assign Base Addresses 以及 System -> Auto-Assign IRQs 自动设定地址和 IRQ。系统 IRQ 可以是从 0 到 31 的整数，数值越小优先级越高。

9. 生成系统。通过点击下方 Generate 完成。如图 7-7。

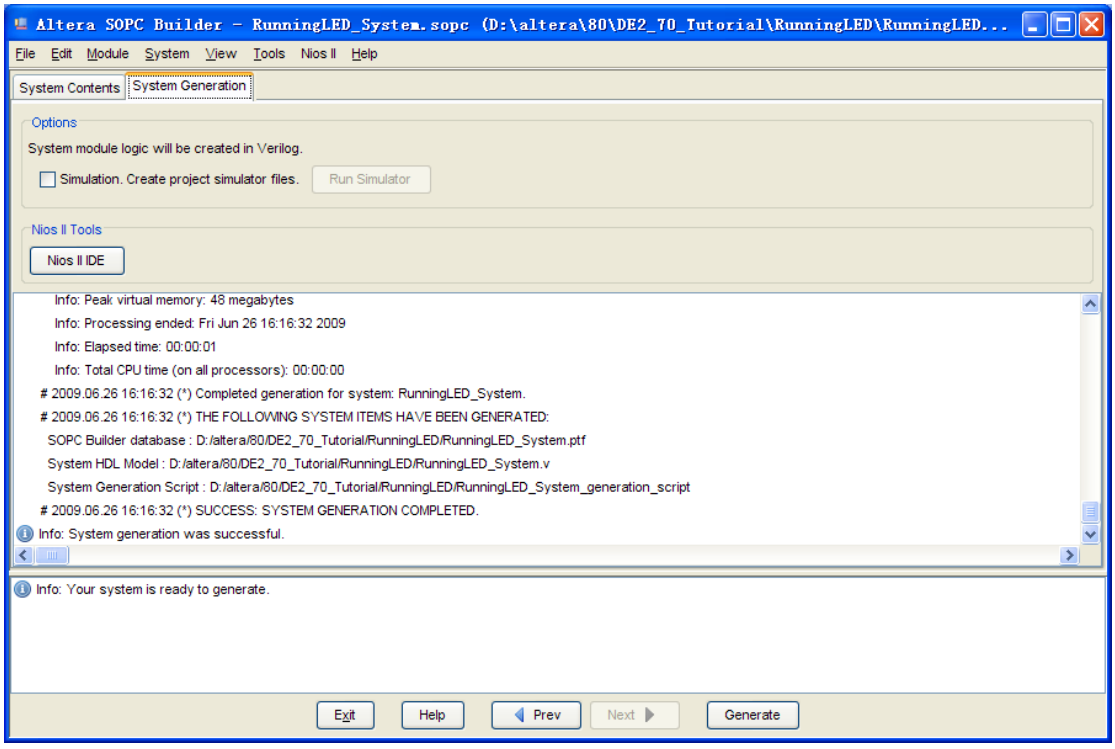


图 7-7 生成系统

7.3 用符号框图完成顶层实体

10. 这次使用符号框图完成顶层实体。新建一个符号文件，添加刚才建立的 SOPC 系统。如图 7-8。

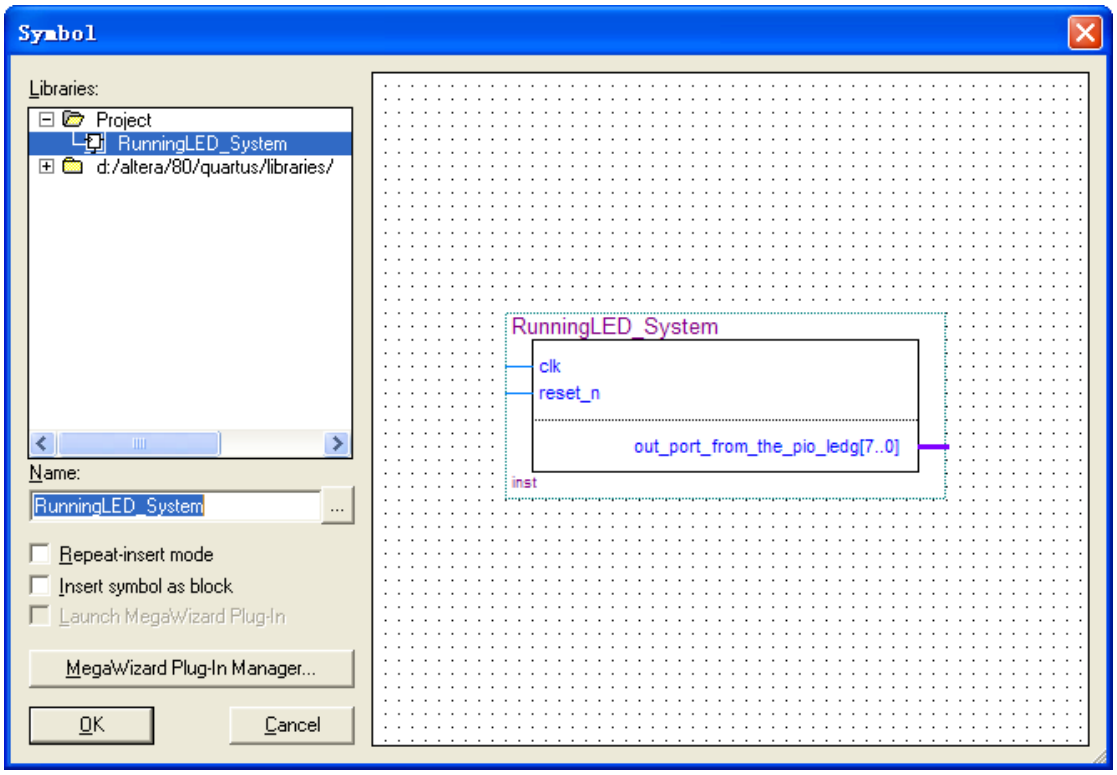


图 7-8 添加 SOPC 系统

11. 添加输入与输出端口。在空白部分双击，在 Name 框内输入 input 可以快速定位，添加输入端口。一共需要两个。然后使用同样步骤添加一个 ouput 输出端口。结果应如图 7-9 所示。

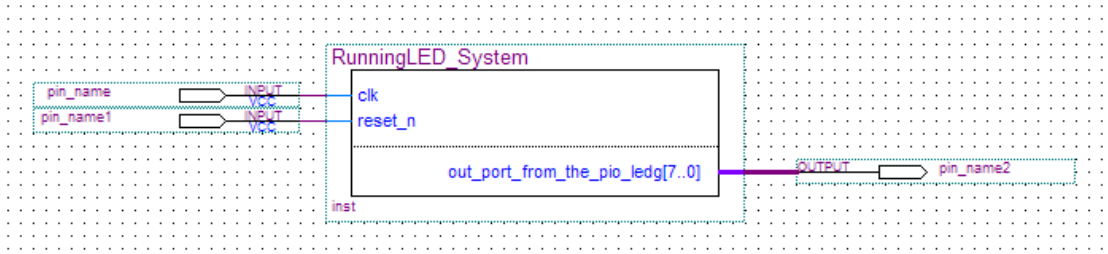


图 7-9 添加结果图

将两个输入端分别改名为 iCLK_50 及 iKEY[0]，代表开发板上的 50MHz 晶振和 KEY0 按钮。将输出端改名为 oLEDG[7..0]，代表开发板上的 oLEDG7 到 oLEDG0 共 8 个绿色 LED 灯。需要注意的是 SOPC Builder 生成的系统的重启信号为低电平有效，开发板上的按键按下后代表低电平，弹起代表高电平。然后将这几个元件连接起来，硬件电路部分设计完毕。电路应如下图所示。(注意：此处的名称修改应该与 DE2-70 引脚的配置相一致)

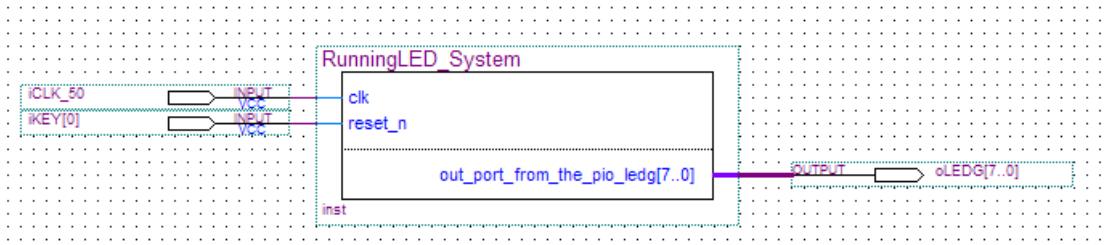


图 7-10　电路图

12. 保存 bdf 文件，然后执行分析与综合

13. 分配引脚

```
set_location_assignment PIN_AD15 -to iCLK_50
set_location_assignment PIN_AC14 -to oLEDG[8]
set_location_assignment PIN_W27 -to oLEDG[0]
set_location_assignment PIN_W25 -to oLEDG[1]
set_location_assignment PIN_W23 -to oLEDG[2]
set_location_assignment PIN_Y27 -to oLEDG[3]
set_location_assignment PIN_Y24 -to oLEDG[4]
set_location_assignment PIN_Y23 -to oLEDG[5]
set_location_assignment PIN_AA27 -to oLEDG[6]
set_location_assignment PIN_AA24 -to oLEDG[7]
set_location_assignment PIN_T29 -to iKEY[0]
```

14. 编译下载。编译完成后将程序烧写至 FPGA 开发板。由于目前还没有编写软件，因此开发板上不会有什么现象。

7.4 软件设计

15. 打开 Nios II IDE，首先选择一个合适的工作空间，依旧设置在<工程所在目录>\softawre。如图 7-11。

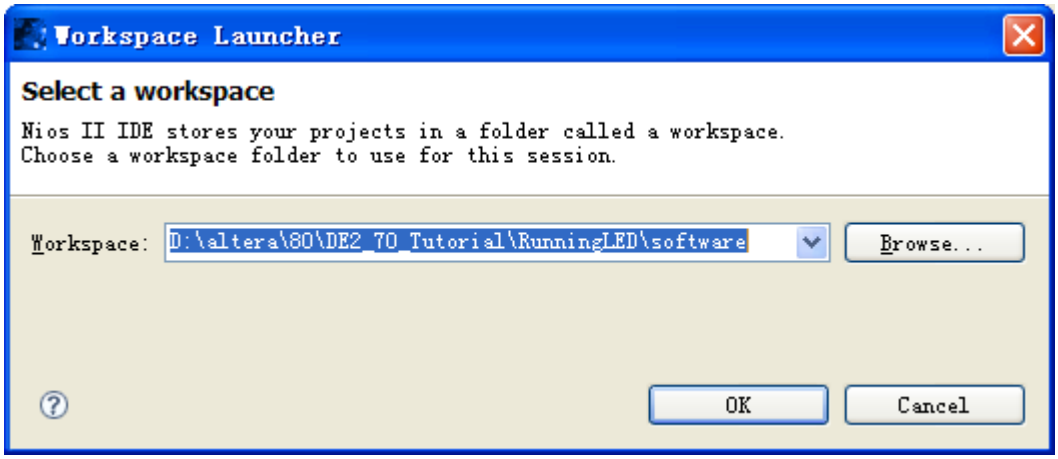


图 7-11　选择工作空间

确认以后软件会重新启动。在欢迎界面中选择 Workbench，进入主界面

16. 选择 File -> New -> Nios II C/C++ Application

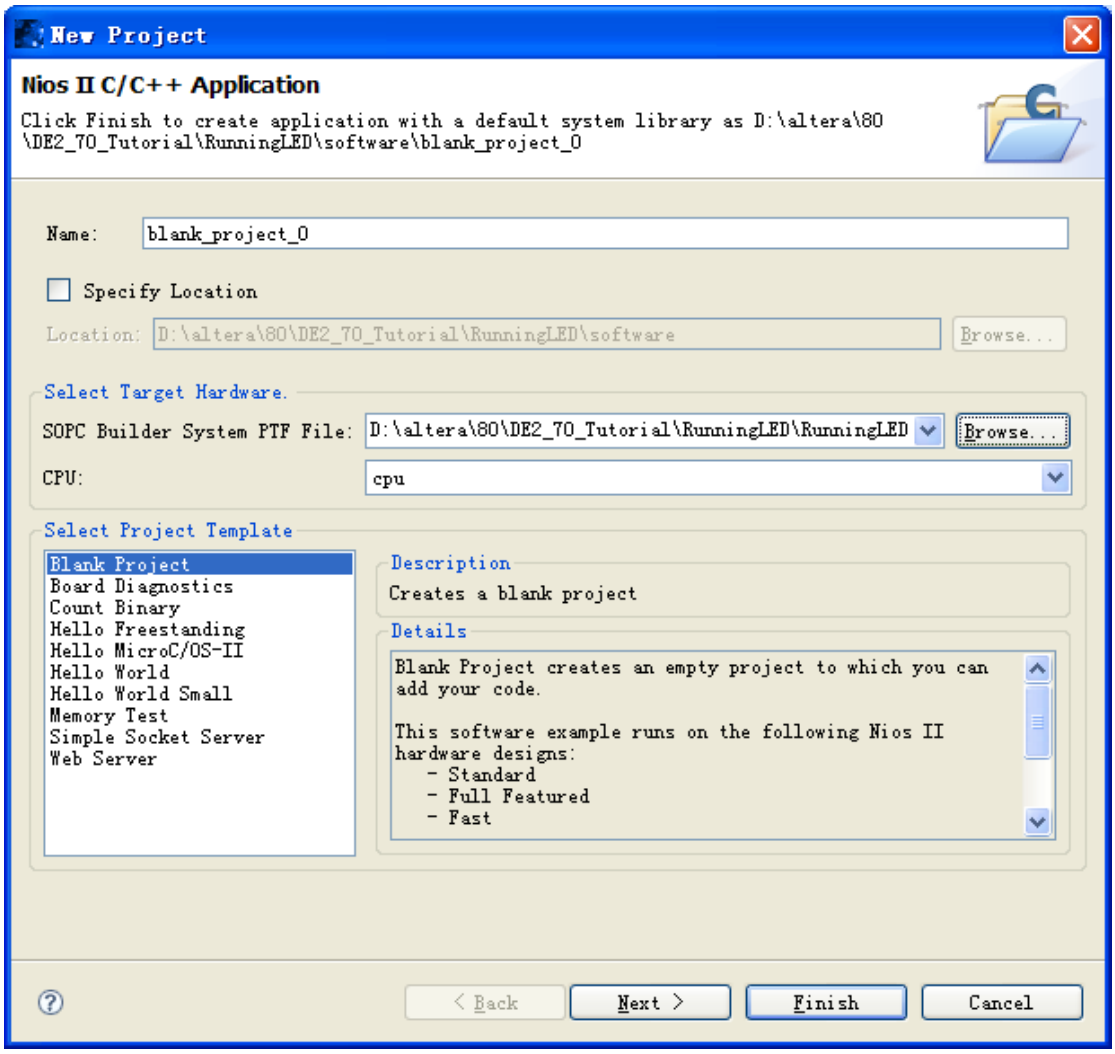


图 7-12　选择工程模板

在 Select Project Template 内选择第一项 Blank Project，Name 使用默认 blank_project_0，SOPC Builder System PTF File 使用默认设置，即刚才生成的 SOPC 系统，点击 Finish 完成。如图 7-12。

17.　选择工程 blank_project_0，右键单击，选择 Properties，在 C/C++ build 选项卡中配置编译器参数，跟上次一样依旧是-DALT_RELEASE -Os -g –Wall，如图 7-13。

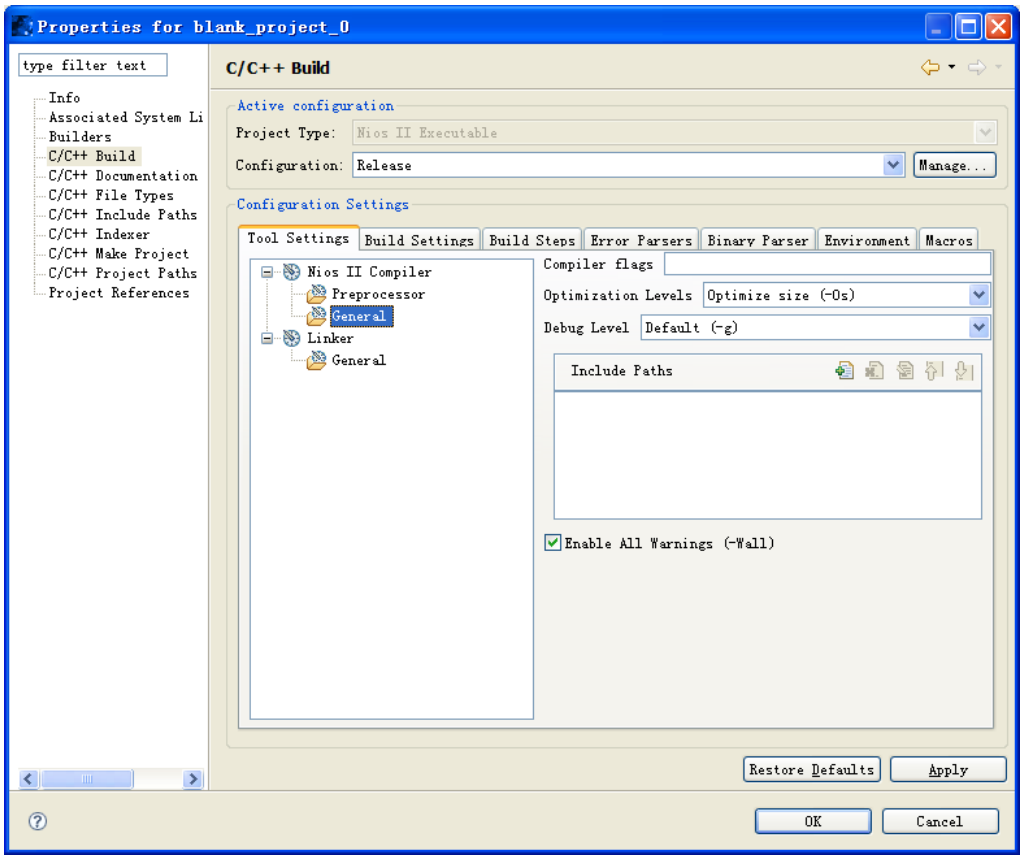


图 7-13　配置工程编译选项

18. 配置 blank_project_0_syslib 的编译器参数。跟 blank_project_0 一样。

19. System Library Properties 设置。

右击工程 blank_project_0_syslib -> Properties，选中 System Library 选项卡，在下图所示界面中，取消掉 Clean Exit (Flush Buffer)和 Support C++前的勾，因为我们的程序不会退出，也不包含 C++的函数和库。选中 Program never exits，Reduce device drivers 和 Small C library 以减小程序体积。其他保持默认设置即可。如图 7-14。**再次声明如果使用 lcd 输出，则不能勾选 Reduce device drivers。**

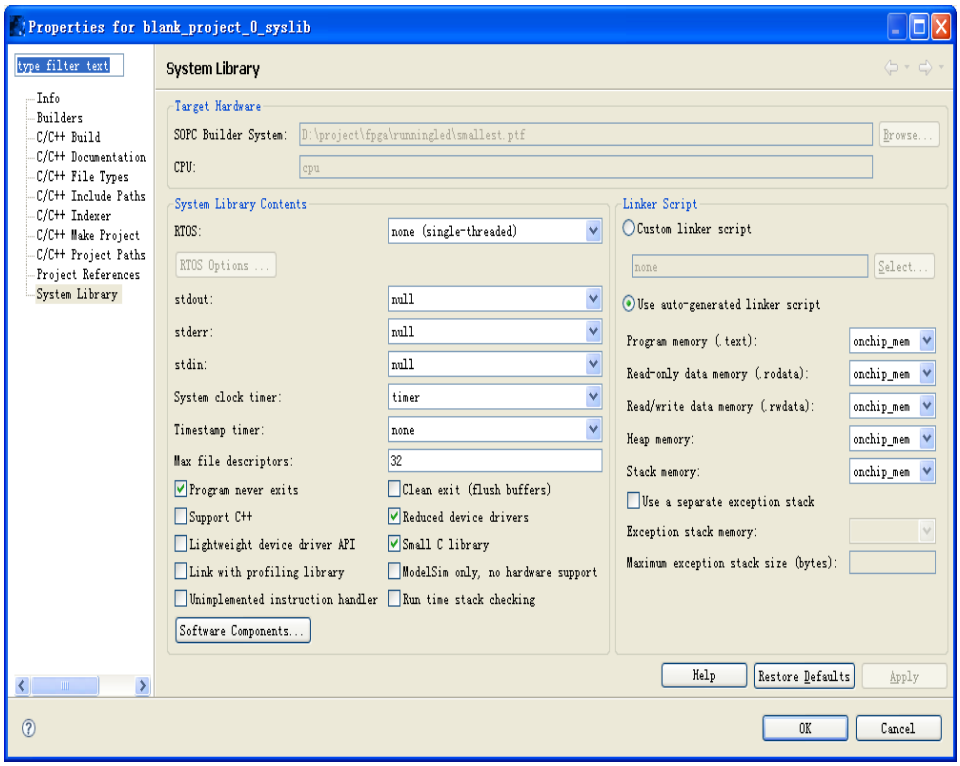


图 7-14　System Library Properties 设置

20.　右键点击工程 blank_project_0，选择 New -> Source File。如图 7-15。

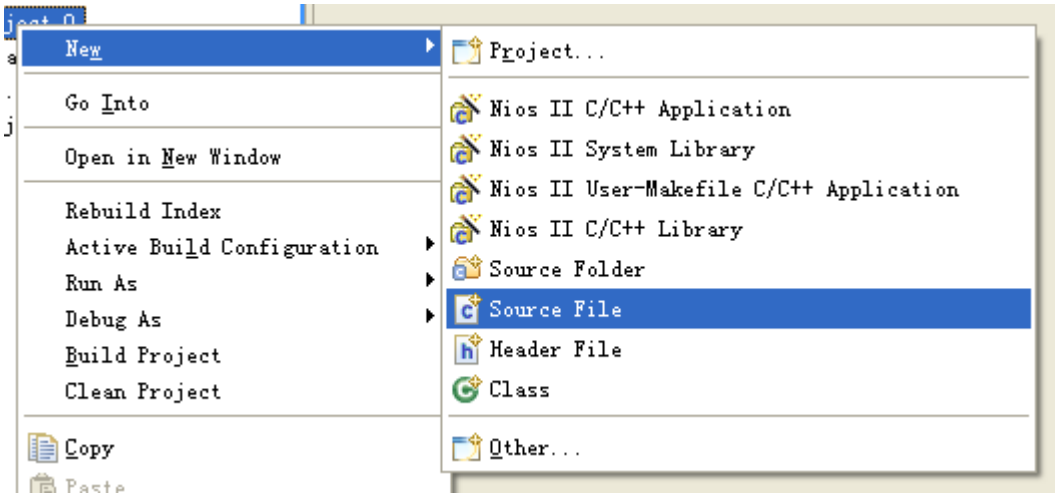


图 7-15 添加源文件图

在弹出的对话框中指定文件名为 main.c，并将以下代码复制进去，并保存。

```
#include "system.h"
#include <sys/alt_irq.h>
#include "alt_types.h"
#include <io.h>

// Internal Timer Overflow interrupt
static void timer_overflow(void* context，　alt_u32 id)
{
    IOWR(TIMER_BASE，0，0);
```

```
    if (*(alt_u8 *)context & 0x80)
    {
        *(alt_u8 *)context = 0x01;
    }
    else
    {
        *(alt_u8 *)context = *(alt_u8 *)context << 1;
    }
    IOWR(PIO_LEDG_BASE，0，*(alt_u8 *)context);
    return;
}

int main()
{
    alt_u8 led = 0x01;
    // Register Interrupt Service Routine (ISR)
    alt_irq_register(TIMER_IRQ，  (void*)&led，  timer_overflow);
    while(1);
}
```

21. Project->Build All，编译，结果只用了 2K。
22. 右击 blank_project_0，选择 Run As ->Nios II Hardware 如图 7-16。

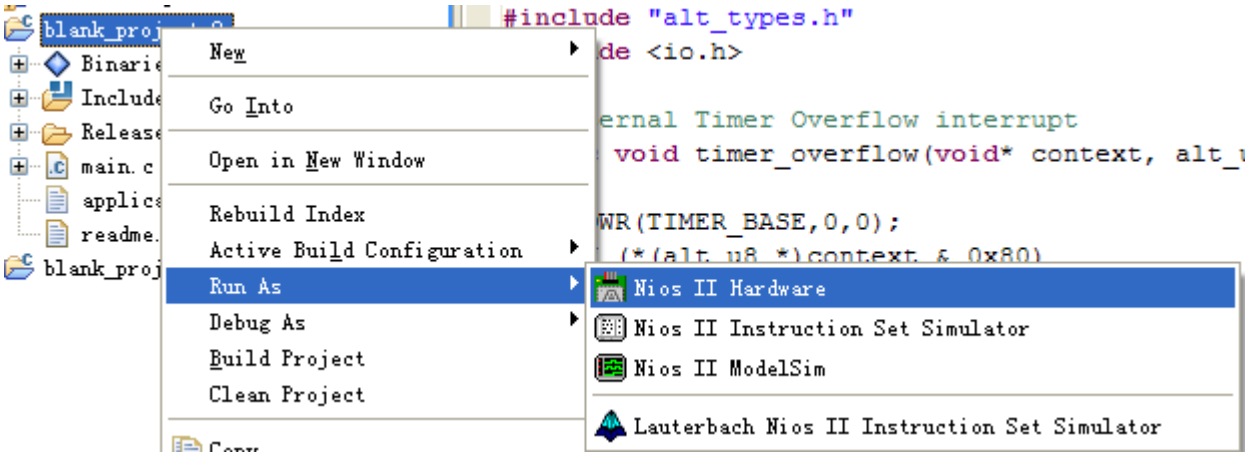


图 7-16 下载运行 C 程序

23. NIOS II IDE 将程序下载到开发板上，之后就能看到 8 个 LEDG 灯轮流点亮了。

第 8 章　实验七　C2H 编译器实验

- 实验说明

该实验将演示在 Nios II 处理器编写程序的时候，Nios II IDE 中所带的 C2H Compiler 工具的作用。什么是 C2H?它是(C to Hardware)的缩写，能将你原本的软件 C 语言程序代码变成硬件实现。

- 实验步骤

8.1 建立 Quartus 工程

1. 新建 Quartus 工程 C2H，顶层实体名 C2H
2. 重新设置编译输出目录为../ C2H/release。

8.2 建立 SOPC 系统

3. 打开 SOPC Builder，建立一个名为 C2H_System 的 SOPC 系统，并指定 Verilog 为描述系统的语言。
4. 在系统上添加 On-Chip Memory。保持默认设置即可。
5. 添加 Nios II Processor。依旧选择 E 型。
6. 添加串行 I/O，将 Width 改为 18，对应 18 个红色 LED 灯。如图 8-1。

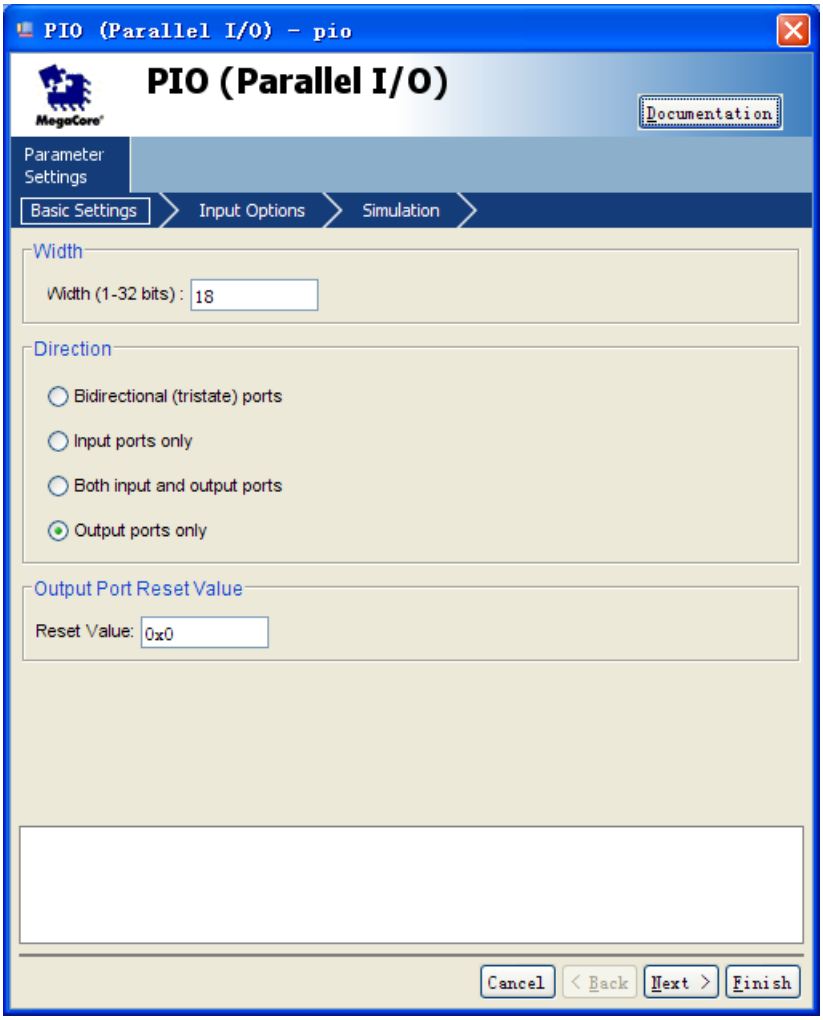


图 8-1　PIO 参数设置

在系统组件列表中，右键单击新添加的 pio，选择 Rename，重命名为 pio_ledr。

7. 选择菜单项 System -> Auto-Assign Base Addresses 去除基址错误。如图 8-2。

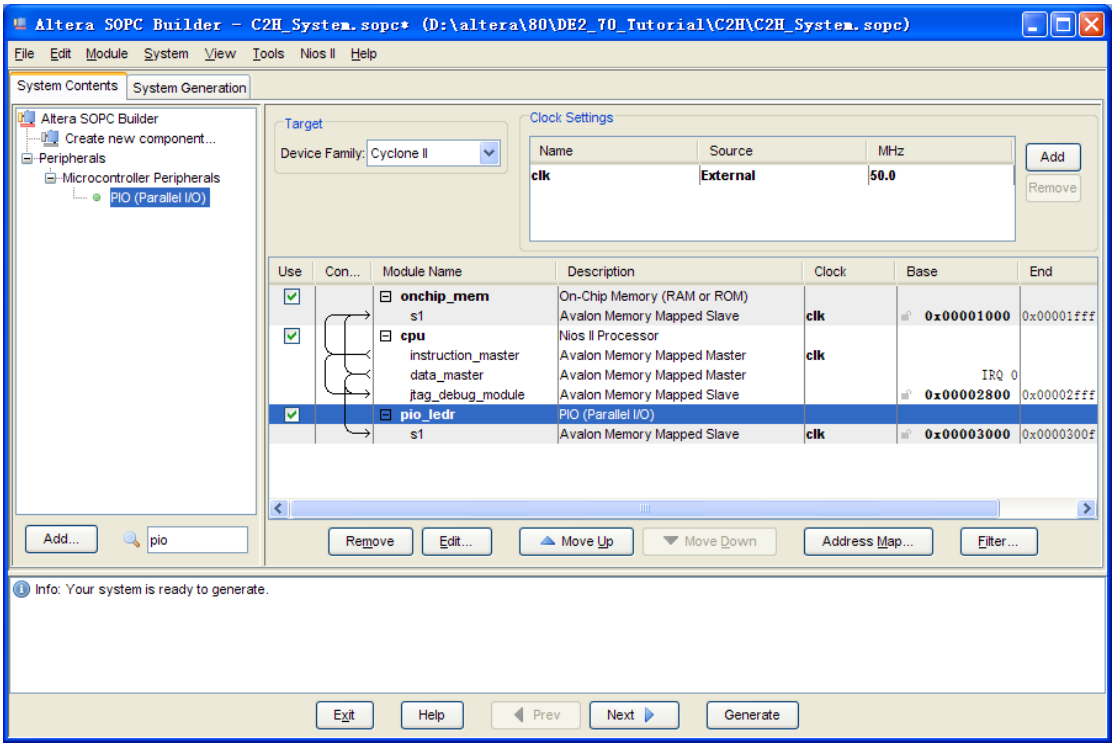


图 8-2　完成的 SOPC 工程

8. 生成系统。通过点击下方 Generate 完成。

8.3 如何用 Verilog 语言完成顶层实体

9. 这次具体讲解如何建立一个使用 Verilog 语言描述的顶层实体文件，之前所做的都是直接给代码样例，现在具体讲解这些顶层实体代码是怎么来的。

10. 顶层实体只要干两件事：

（1）决定最顶层的输入输出使用哪个器件。

（2）实例化之前建立的模块。

11. 这次实验的输出只用到了 oLEDR，输入由于用到 SOPC，需要一个时钟信号与复位信号，时钟信号 DE2-70 开发板上常用的是 iCLK_50，复位信号常用 iKEY[0]。

12. 可以写一个顶层实体头如下：

```
module C2H (
input iCLK_50,
input [0:0] iKEY,
output [17:0] oLEDR
);
```

13. 打开之前 SOPC 建立的创建 C2H_System.v 文件，查找 module C2H_System，发现

```
module C2H_System (
// 1) global signals:
clk,
reset_n,

// the_pio_ledr
```

```
out_port_from_the_pio_ledr
)
;
```

这就是需要在顶层实例化的模块了，

14. 将以上代码的 module 关键字去掉，随便起个实例名 u0，每个参数变成“.参数名(输入输出信号)”的样式，例如 clk 就改为.clk(iCLK_50)，修改代码如下

```
C2H_System u0 (
.clk(iCLK_50)，
.reset_n(iKEY[0])，
.out_port_from_the_pio_ledr(oLEDR[17:0])
);
```

15. 加上 endmodule，如此一来顶层实体就完成了。
16. 分析与综合
17. 分配引脚

```
set_location_assignment PIN_AD15 -to iCLK_50
set_location_assignment PIN_T29 -to iKEY[0]
set_location_assignment PIN_AJ6 -to oLEDR[0]
set_location_assignment PIN_AK5 -to oLEDR[1]
set_location_assignment PIN_AC13 -to oLEDR[10]
set_location_assignment PIN_AB13 -to oLEDR[11]
set_location_assignment PIN_AC12 -to oLEDR[12]
set_location_assignment PIN_AB12 -to oLEDR[13]
set_location_assignment PIN_AC11 -to oLEDR[14]
set_location_assignment PIN_AD9 -to oLEDR[15]
set_location_assignment PIN_AD8 -to oLEDR[16]
set_location_assignment PIN_AJ7 -to oLEDR[17]
set_location_assignment PIN_AJ5 -to oLEDR[2]
set_location_assignment PIN_AJ4 -to oLEDR[3]
set_location_assignment PIN_AK3 -to oLEDR[4]
set_location_assignment PIN_AH4 -to oLEDR[5]
set_location_assignment PIN_AJ3 -to oLEDR[6]
set_location_assignment PIN_AJ2 -to oLEDR[7]
set_location_assignment PIN_AH3 -to oLEDR[8]
set_location_assignment PIN_AD14 -to oLEDR[9]
```

18. 编译下载

8.4 软件设计

19. 打开 Nios II IDE，选择好合适的工作空间，新建空白工程。如图 8-3。

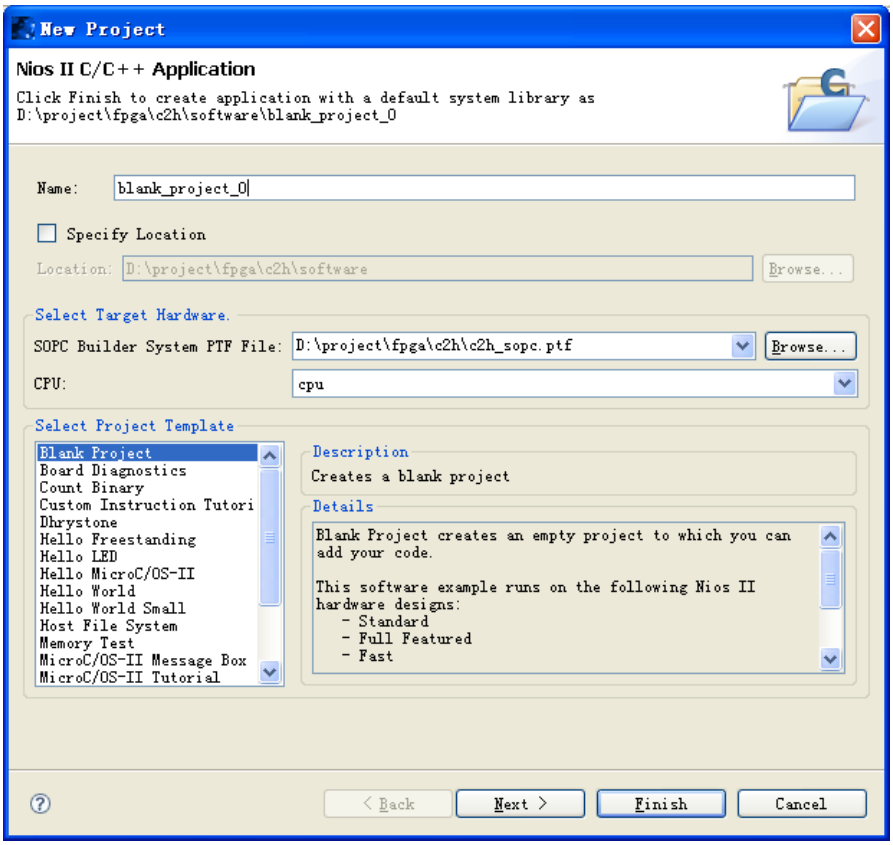


图 8-3 选择工程模板

20. 配置编译器参数为-DALT_RELEASE -Os -g –Wall
21. 配置 System Library Properties。
22. 新建 main.c，添加如下代码。

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"


int main (void) __attribute__ ((weak，  alias ("alt_main")));

void delay (void)
{
      alt_u32 i=0，  j=0;
      while (i<100000)
            i++;
      while (j<100000)
            j++;
      return;
}

int alt_main (void)
{
   alt_u32 led = 0x2;
   alt_u8 dir = 0;
   /*
     * Infinitely shift a variable with one bit set back and forth，  and write
     * it to the LED PIO.    Software loop provides delay element.
     */
```

```
while (1)
{
    if (led & 0x00020001)
    {
        dir = (dir ^ 0x1);
    }

    if (dir)
    {
        led = led >> 1;
    }
    else
    {
        led = led << 1;
    }
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_LEDR_BASE,  led);

    delay();
    }

    return 0;
}
```

23. Project->Build All

24. 右击工程，Run As->Nios II Hardware，可以看到红色 LED 灯轮流闪烁。如果是以 Release 方式编译，那么闪烁将相对较快，如果以 Debug 方式编译，那么闪烁和使用 500ms 的 Timer 速度差不多。

8.5 C2H 编译器

这才是本实验的重点，前面一切都跟实验六跑马灯基本一致。

25. 指定 function 用硬件实现

现在将 delay()操作改为硬件实现，右击 delay()选择 Accelerate with the Nios II C2H Compiler。如图 8-4。

图 8-4 开启 C2H 编译器加速

26. 选了之后会在下方多出 C2H 的设定，选择“Build Software and generate SOPC Builder system”和“Use hardware accelerator in place of software implementation. Flush data cache before each call”。如图 8-5。

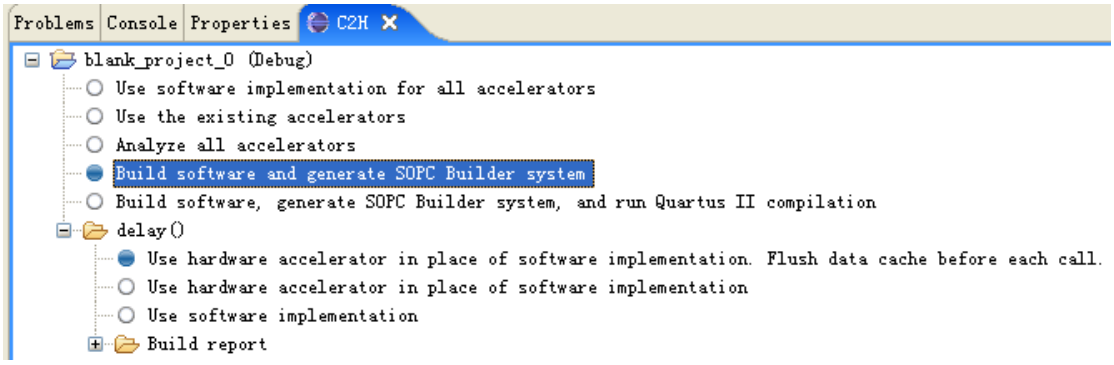


图 8-5 C2H 编译器加速选项（硬件加速）

27. Project->Build All，比纯编译软件要久，结果 576B，非常小。

28. 回到 Quartus II 查看 SOPC，发现确实生成了新的硬件模块，如图 8-6

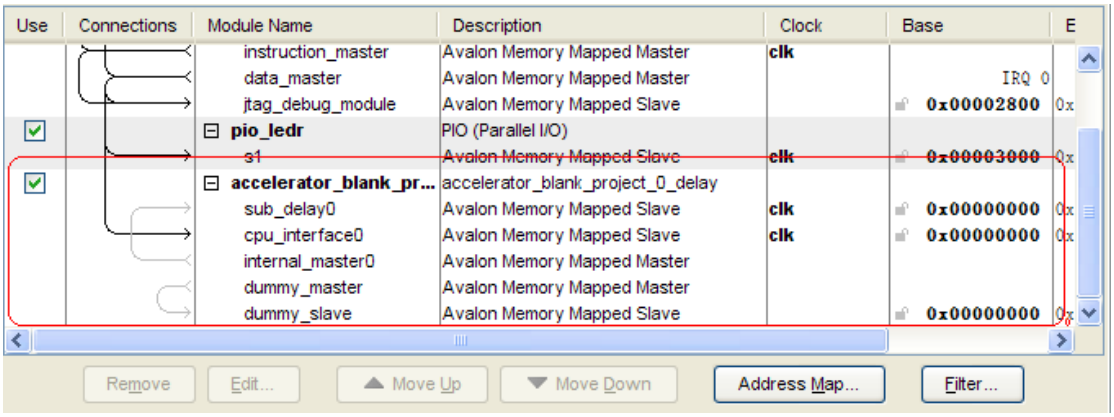


图 8-6 C2H 编译结果在 SOPC 中

29. 退出 SOPC，Quartus II 全编译，很遗憾生成的 sof 文件被受限，不是我们想要的 C2H.sof，而是 C2H_time_limited.sof，即表示下在到板上后无法长期使用。

这是因为我们的 Quartus 没有使用完全的 license，是破解版的。这也是之前为什么不选择“Build Software and generate SOPC Builder system，and run Quartus II compilation”而选择“Build Software and generate SOPC Builder system”的原因。

不过没有关系，这只是实验，不需要长期运行，像平时一样用 programmer 将其烧入 DE2-70 上运行即可。

30. 点击 Programmer，弹出警告对话框，如图 8-7。

图 8-7 使用 C2H 编译器后 Quartus II 编译结果受限警告

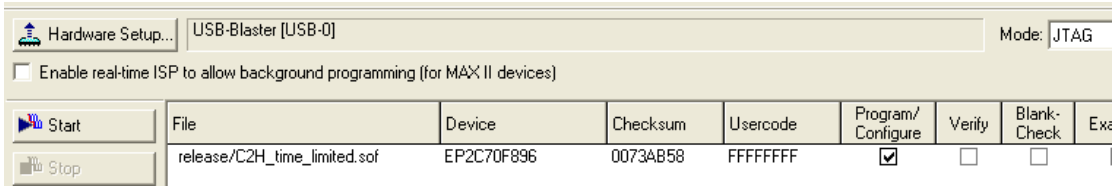


图 8-8　使用 C2H 编译器后 Quartus II 编译结果下载窗口

31. 参看图 8-8，点击 start 下载，出现 Error: The OpenCore Plus IP in device 1 is not responding.可以无视。

32. 回到 Nios II IDE 运行，发现灯全面点亮，这是因为原本每次加一都是 CPU 指令执行，现在是硬件自己完成，硬件实现比软件快很多。

33. 如欲关闭 C2H 硬件加速，在 C2H 的 View 里选择对应函数下面的 Use Software implementation 即可，如图 8-9。

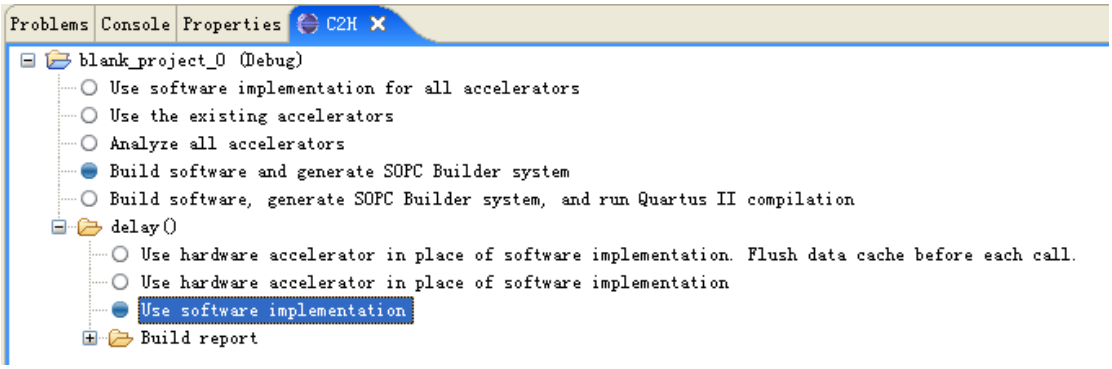


图 8-9 C2H 编译器加速选项（软件实现）

34. 小结：使用软件 Debug 方式，灯闪烁速度较慢；使用软件 Release 方式，灯闪烁速度较快，但肉眼可识别；使用 C2H 硬件方式，灯闪烁速度肉眼不可识别。

第 9 章　实验八 上电自动加载软硬件程序

- 实验说明

实际应用中往往需要上电时加载 FPGA 配置文件，然后自动加载软件运行，FPGA 配置文件可以通过 Quartus II Programmer 烧写在 EPCS 芯片中，软件同样可以通过 Nios II IDE Flash Programmer 烧写进板上带有的 Flash 或 EPCS 中。本实验以上电加载 Flash 中的跑马灯程序为例。本实验还演示了如何从已有的工程上再加工。

- 实验步骤

9.1 建立 Quartus 工程

1. 复制原来的 RunningLED 文件夹为 RunningLED_Flash。
2. 打开 RunningLED.qpf 文件进入工程
3. 重新设置编译输出目录为../ RunningLED_Flash/release
4. 删除原本编译结果

9.2 建立 SOPC 系统

5. 打开 SOPC Builder。
6. 添加 Bridges and Adapters ->Memory Mapped-> Avalon-MM Tristate Bridge：属性第一页 Incoming Signals 选择 Registered。如果是 8.0 可以直接 Finish，7.2 在第二页 Shared Signals 中将 data，address，read_n 都选上。如图 9-1。

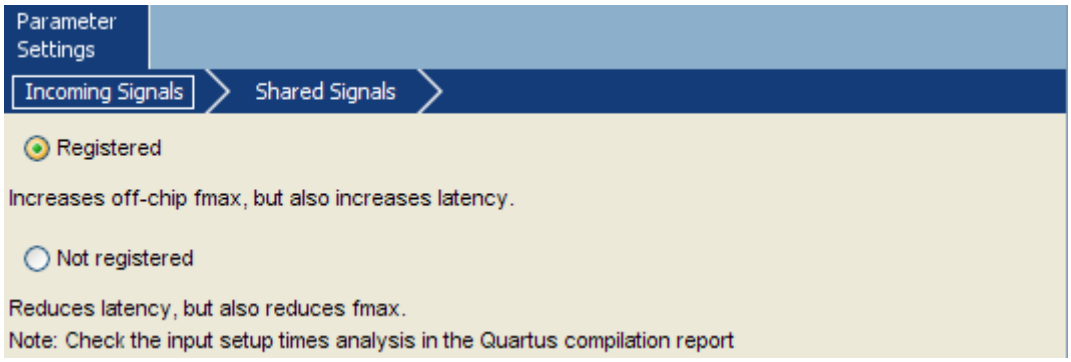


图 9-1 三态桥属性页

7. Memories and Memory Controller->Flash->Flash Memory(CFI)：属性第一页 Attributes 中的 Presets 选择 Custom。Size 中地址线宽度选择 22 位，数据线选择 16 位。第二页 Timing 中 Setup=0，Wait=100，Hold=0，Unit=ns。如图 9-2 与图 9-3。

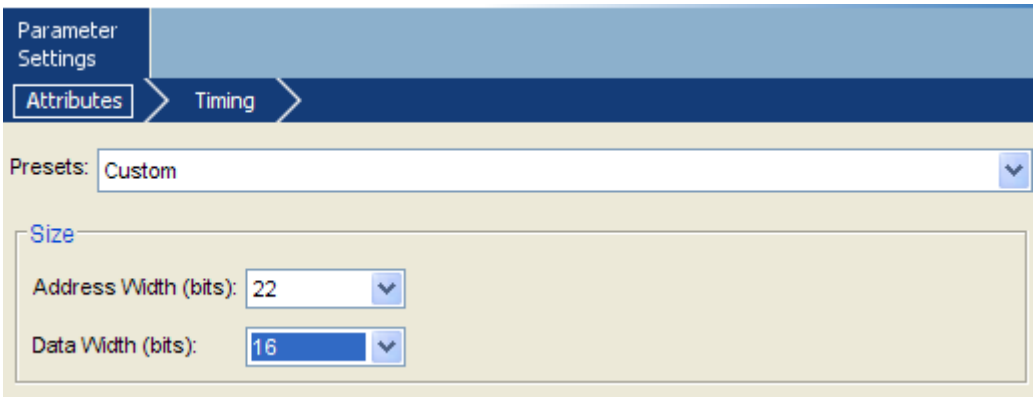


图 9-2 Flash 属性页（1）

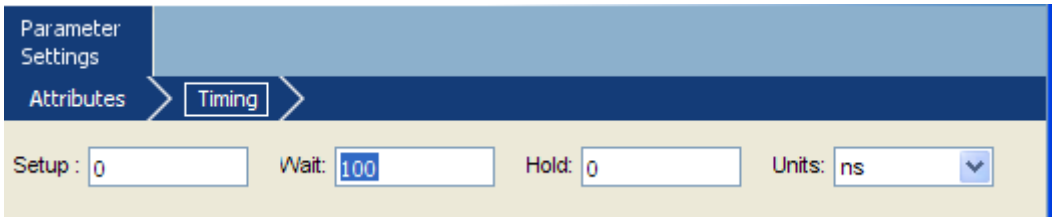


图 9-3 Flash 属性页（2）

注意：这两页的配置数据出自友晶 DE2-70 的光盘上的样例。

8. 将 Avalon-MM Tristate Bridge 与 Flash 连起来，点击图 9-4 中红圈圈住的位置

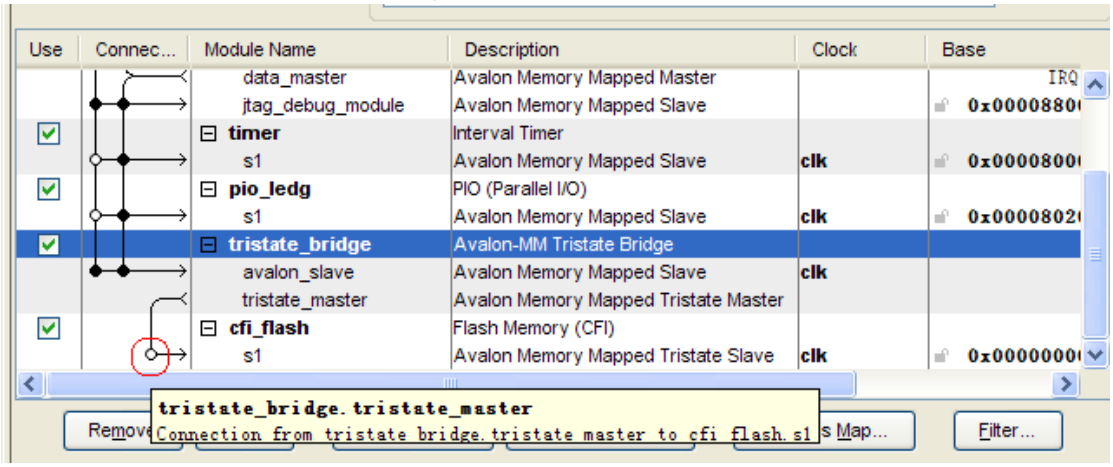


图 9-4　Flash 属性页

9. 完成后双击加入的 CPU，选择其 Reset Vector 在 cfi_flash 中。如图 9-5。

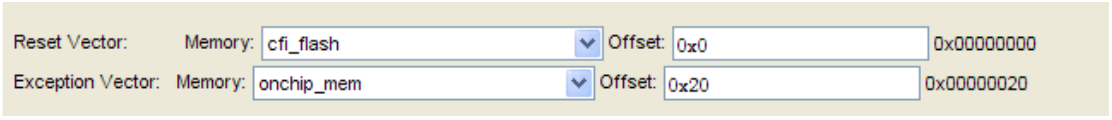


图 9-5　CPU 属性设置

10. 选择菜单栏上 System -> Auto-Assign Base Addresses 重新分配基址，去除基址错误，如图 9-6。

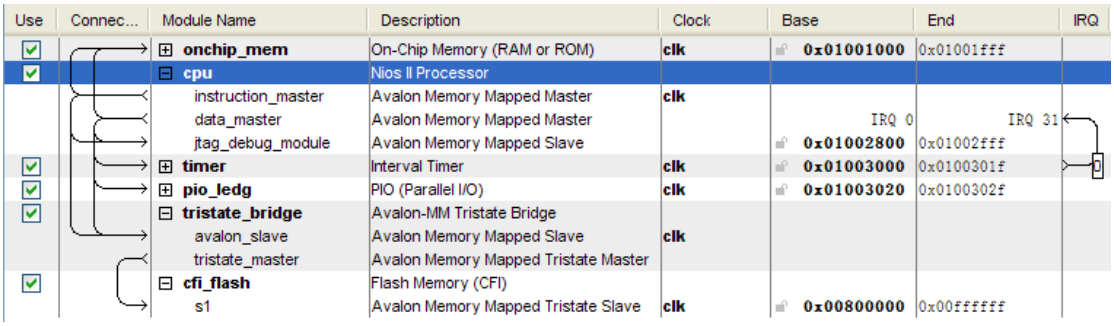


图 9-6 完成的 SOPC 工程

11. 生成系统。点击 Generate 按钮生成系统。

9.3 完成顶层实体

12. 回到 Quartus，把原来的顶层 bdf 文件移除，右击选择 Remove File from project

13. 建立顶层实体如下：

```
module RunningLED_Flash(
input [0:0] iKEY，
```

```
input iCLK_50，
output [7:0] oLEDG，
inout   [14:0] FLASH_DQ，              // FLASH Data bus 15 Bits (0 to 14)
inout          FLASH_DQ15_AM1，   // FLASH Data bus Bit 15 or Address A-1
output [21:0] oFLASH_A，             // FLASH Address bus 22 Bits
output         oFLASH_WE_N，         // FLASH Write Enable
output         oFLASH_RST_N，        // FLASH Reset
output         oFLASH_WP_N，         // FLASH Write Protect /Programming Acceleration
input          iFLASH_RY_N，         // FLASH Ready/Busy output
output         oFLASH_BYTE_N，      // FLASH Byte/Word Mode Configuration
output         oFLASH_OE_N，         // FLASH Output Enable
output         oFLASH_CE_N       // FLASH Chip Enable
);

wire FLASH_16BIT_IP_A0;
assign oFLASH_BYTE_N = 1'b1; // FLASH Byte/Word Mode Configuration
assign oFLASH_RST_N    = 1'b1; // FLASH Reset
assign oFLASH_WP_N     = 1'b1; // FLASH Write Protect /Programming Acceleration

RunningLED_System u0(
.clk(iCLK_50)，
.reset_n(iKEY[0])，
.out_port_from_the_pio_ledg(oLEDG[7:0])，
//the_tristate_bridge_avalon_slave
.address_to_the_cfi_flash({oFLASH_A[21:0]，  FLASH_16BIT_IP_A0})，  // FLASH Address
bus 26 Bits
.data_to_and_from_the_cfi_flash({FLASH_DQ15_AM1，  FLASH_DQ})，       // FLASH Data
bus 15 Bits (0 to 14)
.read_n_to_the_cfi_flash(oFLASH_OE_N)，       // FLASH Output Enable
.select_n_to_the_cfi_flash(oFLASH_CE_N)，   // FLASH Chip Enable
.write_n_to_the_cfi_flash(oFLASH_WE_N)，     // FLASH Write Enable
);
Endmodule
```

注意：该顶层实体的建立牵扯到 Flash 的使用，简单使用的话，按照本例的代码用就可以了，如果想了解细节，请参考 DE2-70 光盘上关于 Flash 的 Datasheet。

14. 将其设为顶层实体并分配引脚

除原有引脚外，新加入 Flash 的引脚

```
set_location_assignment PIN_AF24 -to oFLASH_A[0]
set_location_assignment PIN_AG24 -to oFLASH_A[1]
set_location_assignment PIN_AH26 -to oFLASH_A[10]
set_location_assignment PIN_AJ26 -to oFLASH_A[11]
set_location_assignment PIN_AK26 -to oFLASH_A[12]
set_location_assignment PIN_AJ25 -to oFLASH_A[13]
set_location_assignment PIN_AK25 -to oFLASH_A[14]
```

```
set_location_assignment PIN_AH24 -to oFLASH_A[15]
set_location_assignment PIN_AG25 -to oFLASH_A[16]
set_location_assignment PIN_AF21 -to oFLASH_A[17]
set_location_assignment PIN_AD21 -to oFLASH_A[18]
set_location_assignment PIN_AK28 -to oFLASH_A[19]
set_location_assignment PIN_AE23 -to oFLASH_A[2]
set_location_assignment PIN_AJ28 -to oFLASH_A[20]
set_location_assignment PIN_AE20 -to oFLASH_A[21]
set_location_assignment PIN_AG23 -to oFLASH_A[3]
set_location_assignment PIN_AF23 -to oFLASH_A[4]
set_location_assignment PIN_AG22 -to oFLASH_A[5]
set_location_assignment PIN_AH22 -to oFLASH_A[6]
set_location_assignment PIN_AF22 -to oFLASH_A[7]
set_location_assignment PIN_AH27 -to oFLASH_A[8]
set_location_assignment PIN_AJ27 -to oFLASH_A[9]
set_location_assignment PIN_Y29 -to oFLASH_BYTE_N
set_location_assignment PIN_AG28 -to oFLASH_CE_N
set_location_assignment PIN_AF29 -to FLASH_DQ[0]
set_location_assignment PIN_AE28 -to FLASH_DQ[1]
set_location_assignment PIN_AD29 -to FLASH_DQ[10]
set_location_assignment PIN_AC28 -to FLASH_DQ[11]
set_location_assignment PIN_AC30 -to FLASH_DQ[12]
set_location_assignment PIN_AB30 -to FLASH_DQ[13]
set_location_assignment PIN_AA30 -to FLASH_DQ[14]
set_location_assignment PIN_AE24 -to FLASH_DQ15_AM1
set_location_assignment PIN_AE30 -to FLASH_DQ[2]
set_location_assignment PIN_AD30 -to FLASH_DQ[3]
set_location_assignment PIN_AC29 -to FLASH_DQ[4]
set_location_assignment PIN_AB29 -to FLASH_DQ[5]
set_location_assignment PIN_AA29 -to FLASH_DQ[6]
set_location_assignment PIN_Y28 -to FLASH_DQ[7]
set_location_assignment PIN_AF30 -to FLASH_DQ[8]
set_location_assignment PIN_AE29 -to FLASH_DQ[9]
set_location_assignment PIN_AG29 -to oFLASH_OE_N
set_location_assignment PIN_AH28 -to oFLASH_RST_N
set_location_assignment PIN_AH30 -to iFLASH_RY_N
set_location_assignment PIN_AJ29 -to oFLASH_WE_N
set_location_assignment PIN_AH29 -to oFLASH_WP_N
```

15. 进行全编译。编译完成后将程序烧写到 FPGA。

9.4 软件设计

16. 打开 Nios II IDE 进行软件设计。将工作空间切换到../RunningLED_Flash /software。
17. 检查编译器参数，编译工程并运行测试之

9.5 软件固化

18. 打开 Tool -> Flash Programmer。如图 9-7 所示。

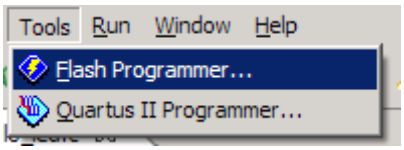


图 9-7 Flash Programmer

19. 打开后，如图 9-8 所示。

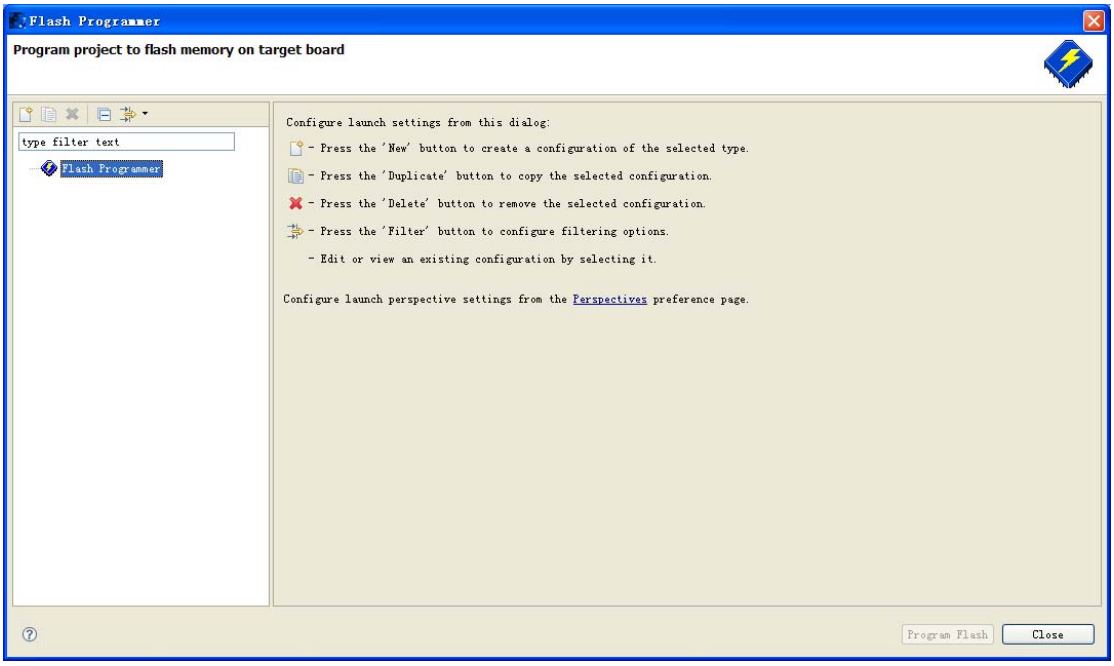


图 9-8 Flash Programmer 界面

20. 左侧 new 一个项目，随便填写个名称比如 RunningLED，Apply 刷新，如图 9-9

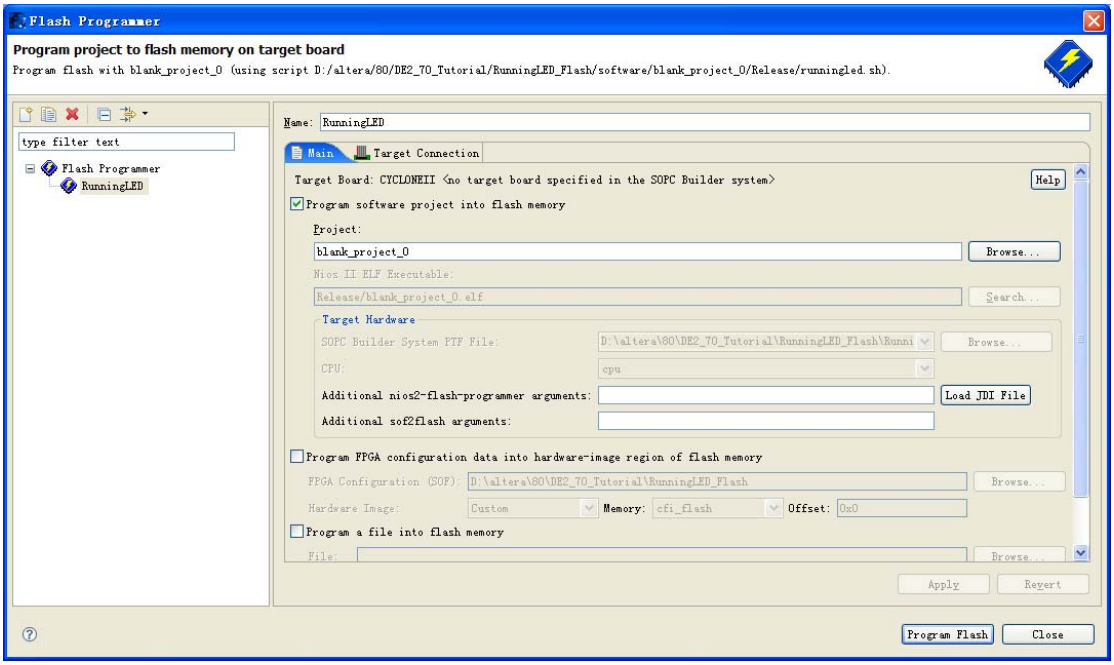


图 9-9　Flash Programmer 新建任务

21. 点击 Program Flash 就可以将程序烧入 Flash。完成后如图 9-10。

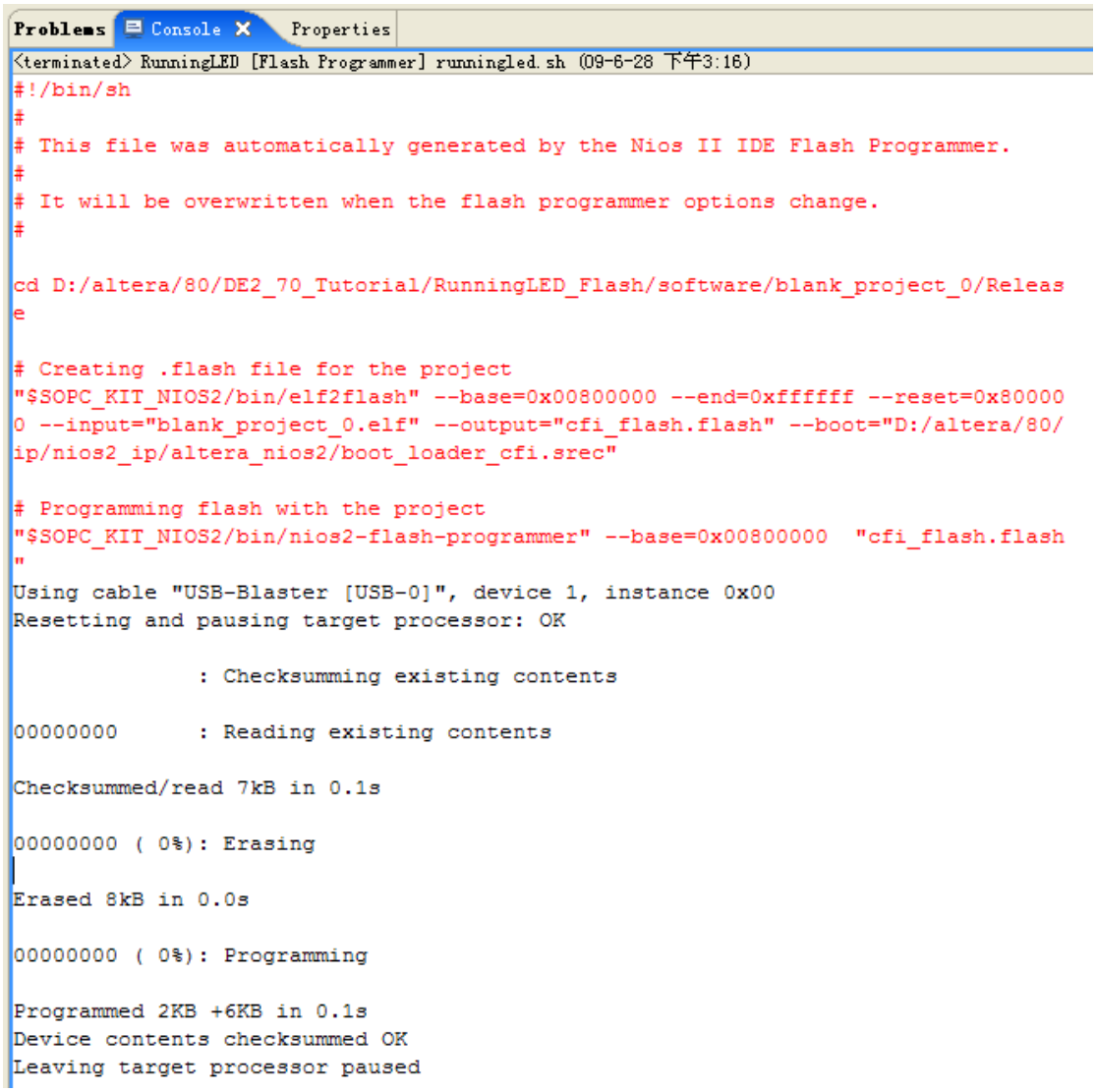


图 9-10 烧写程序到 Flash 图

22. 关闭 Nios II IDE，断电后再打开，烧入 FPGA 的配置程序就可以看到程序直接运行了。

9.6 FPGA 配置固化

23. 现在加电启动后依旧需要烧入 FPGA 配置文件方可引导软件程序，如果把 sof 文件也固化掉，那么加电启动后就可以脱离开发主机直接运行你所设计的程序。

方法归纳为三步：

（1）把 DE2-70 开发板左侧的开关从 RUN 拨到 PROG

（2）在平时经常使用的 Programmer 窗口中，把 mode 从 JTAG 改为 Active Serial Programming，遇到一个提示窗口选是确认。如图 9-11。

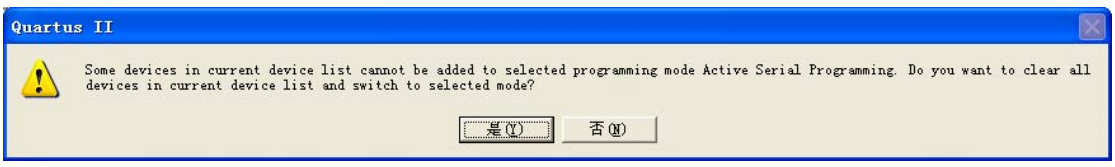


图 9-11 烧写 FPGA 配置提示窗口

（3）重新添加 pof 文件，不是 sof 文件，勾上 Program/Configure，start。如图 9-12

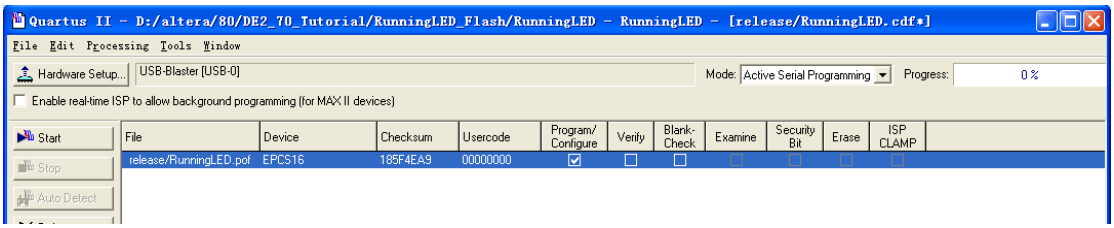


图 9-12 烧写 FPGA 配置到 Flash 图

24. 等待烧写完毕，直到右上角进度条 100%，状态栏显示 Ended 信息，如图 9-13。

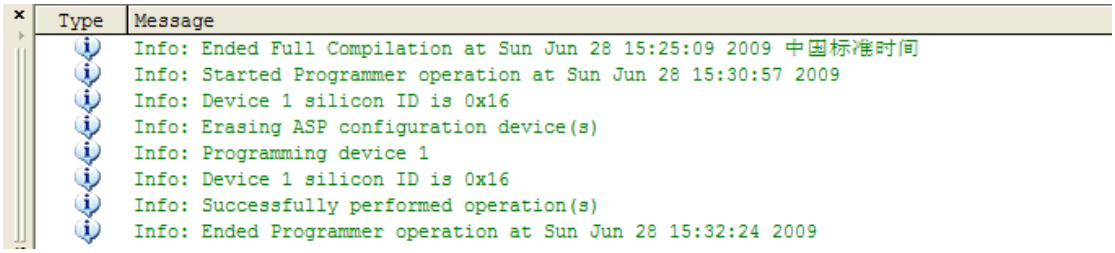


图 9-13 烧写 FPGA 配置到 Flash 完成

25. 把 DE2-70 开发板上的开关从 PROG 拨回 RUN。
26. 重新加电测试，如果忘了上一步，那么将会看到一排很暗的灯。
27. 如想恢复初始设置，烧入 DE2-70 光盘上的 DE2_70_Default 即可。

第 10 章　实验九　SDRAM 读写测试实验

- 实验说明

该实验主要完成对 SDRAM 读写的测试。主要讲解如何使用 SDRAM，由于 DE2-70 上的 SDRAM 是两片，所以比使用 Flash 稍微复杂一点点。通过本实验，读者应该了解不同器件对时钟的需求不同，并学会如何创建新的时钟。两片 SDRAM 的用法可以是统一使用，即只建立一个 SOPC 的 SDRAM 模块，数据宽 32 位，也可以分开使用，即建立两个 SOPC 的模块，数据宽度 16 位，本例是读写测试，需要知道两片各自独立的基址，故使用后一种用法。

- 实验步骤

10.1 建立 Quartus 工程

1. 建立一个新的工程 SDRAMTest。
2. 重新设置编译输出目录为../ SDRAMTest/release

10.2 建立 SOPC 系统

3. 打开 SOPC Builder，建立一个名为 SDRAMTest_System 的 SOPC 系统，并指定 Verilog 为描述系统的语言。
4. 在系统上添加 On-Chip Memory。大小设置 20k。
5. 添加 Nios II Processor。依旧选择 E 型。
6. 添加左侧的 Memories and Memory Controllers->SDRAM->SDRAM Controller：

配置第一页中，presets 选择 Custom，Data Width 选择 16，Chip Select 选 1，Bank4，Row 选 13，Column 选 9，确认 Memories size=32MB。第二页中，Issue one refresh command every 填 7.8125us，Delay after powerup，before initialization 填 200us。如图 10-1 与图 10-2。

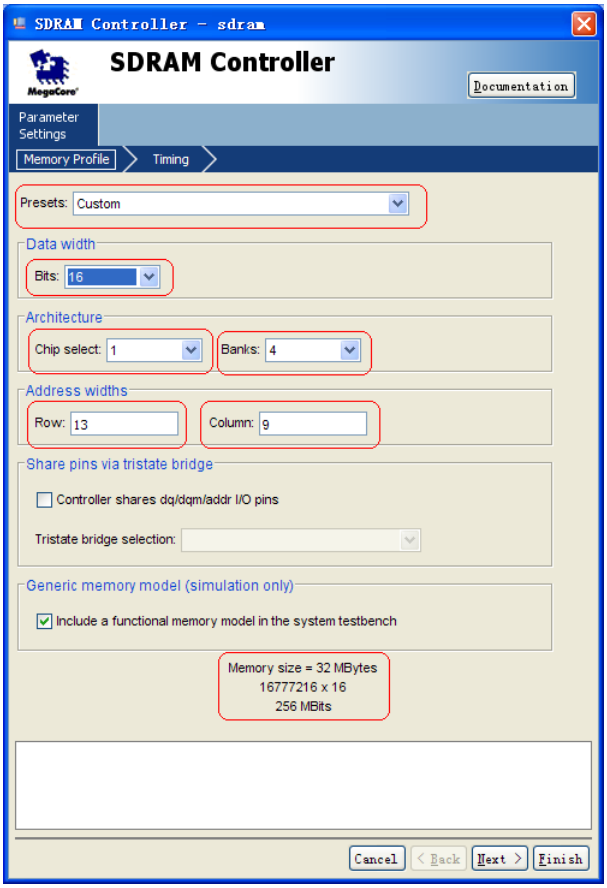


图 10-1 SDRAM Controller 设置第一页

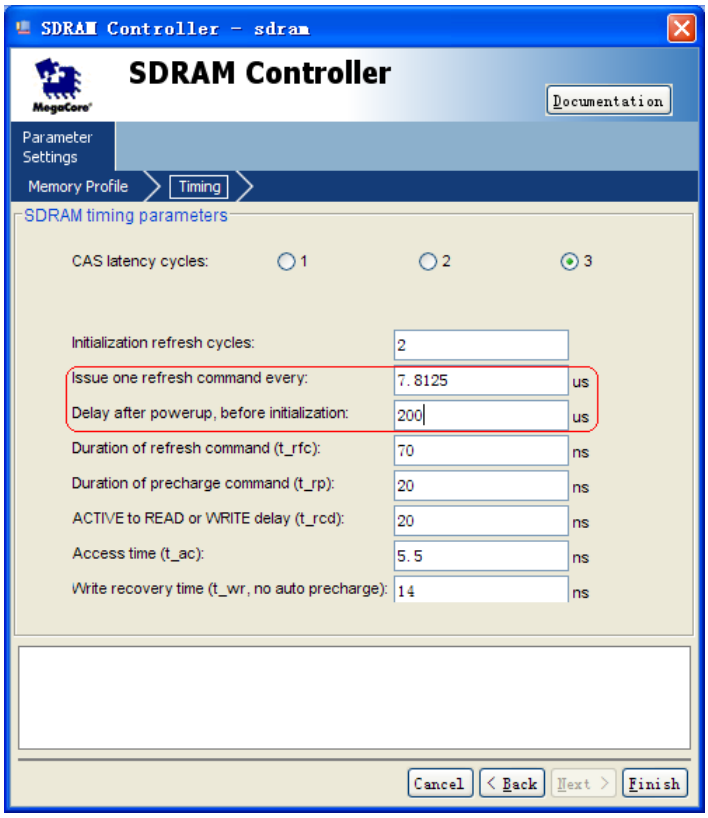


图 10-2 SDRAM Controller 设置第二页

注意：这两页的配置数据出自友晶 DE2-70 的光盘上的样例

7. 如此再添加一块 SDRAM，分别命名为 sdram_u1 与 sdram_u2。

8. 添加 jtag_uart，这个系统的输出连到 Nios II IDE 的 Console，需要 jtag_uart 支持。

9. 添加 pll，在左侧 PLL->PLL，在弹出的窗口中选择 Launch Altera's ALTPLL MegaWizard，如图 10-3。

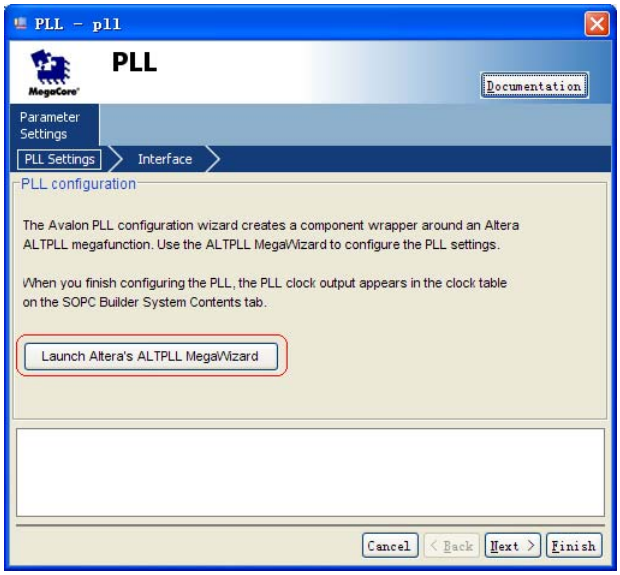


图 10-3　PLL 设置第一页

10. 弹出窗口如图 10-4，next

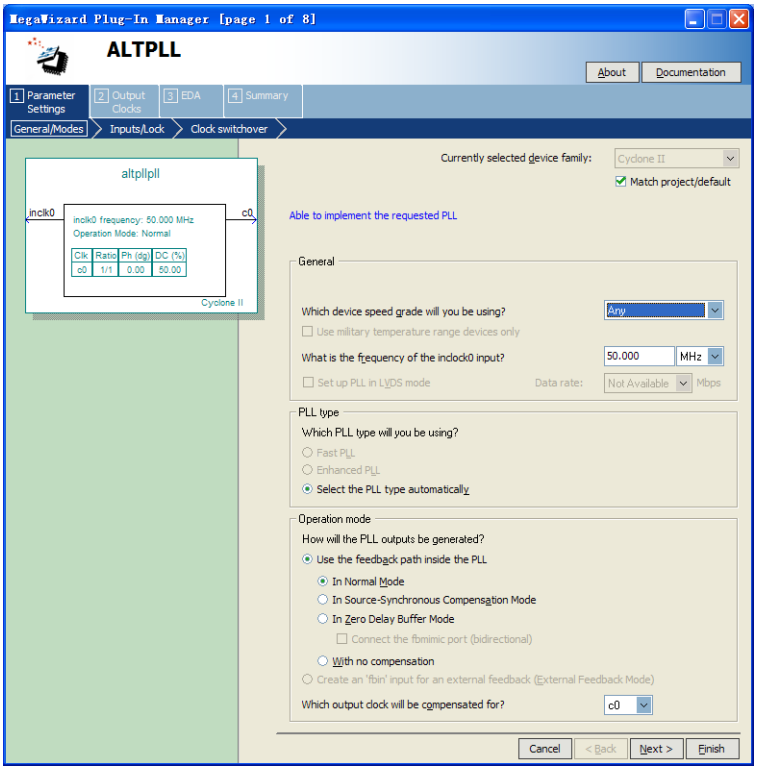


图 10-4　ALTPLL 设置第一页

11. 如图 10-5，继续 next

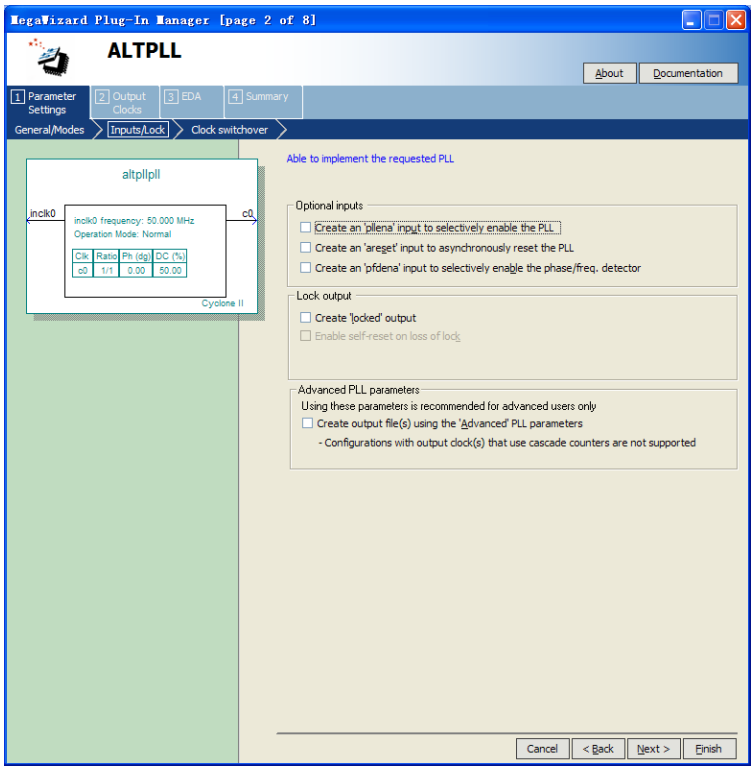


图 10-5　ALTPLL 设置第二页

12. 如图 10-6，继续 next

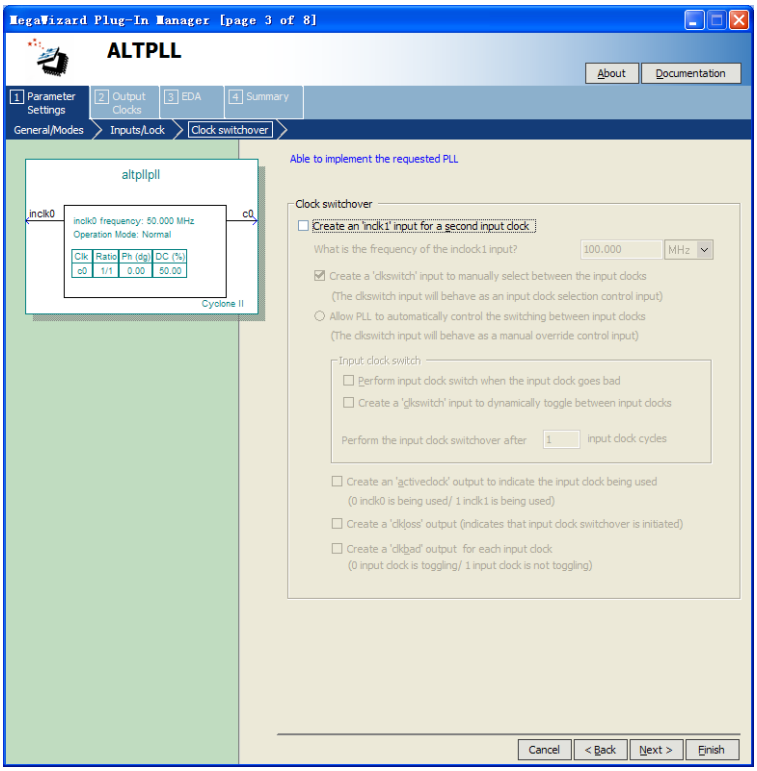


图 10-6　ALTPLL 设置第三页

13. 来到第一个输出时钟的设置，倍频选 2，即给系统时钟 100MHz，如图 10-7。第二输出时钟设为一个负 65 度相位的 100MHz 时钟，给 SDRAM，如图 10-8。

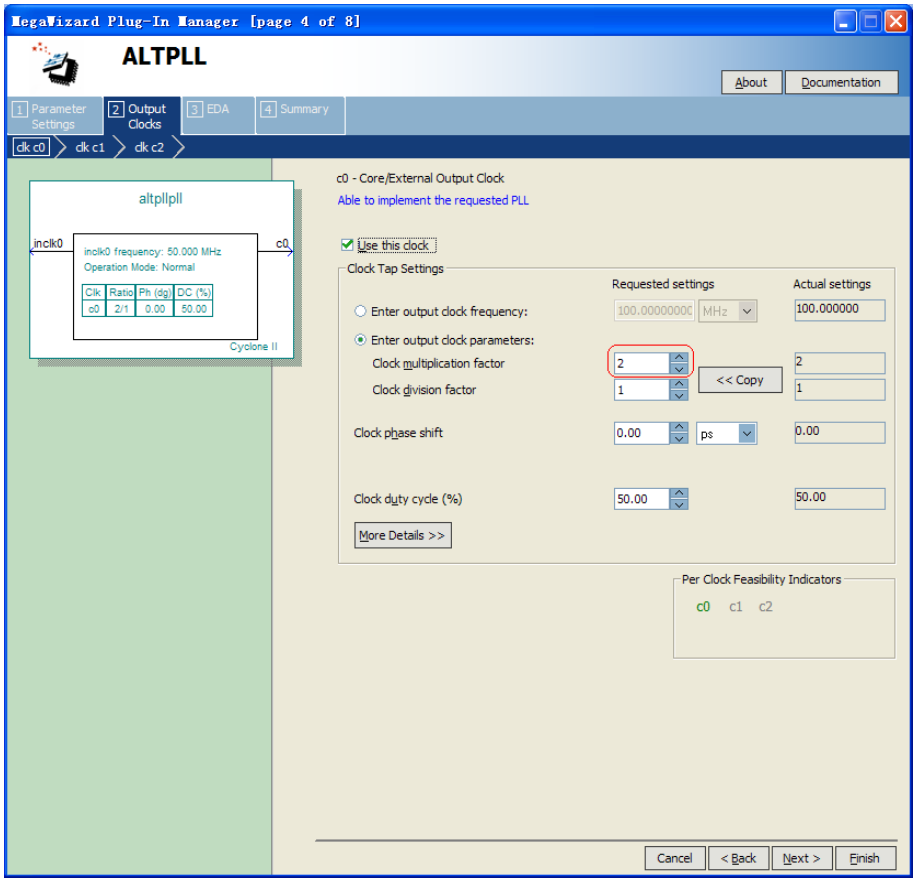


图 10-7　ALTPLL 设置第四页

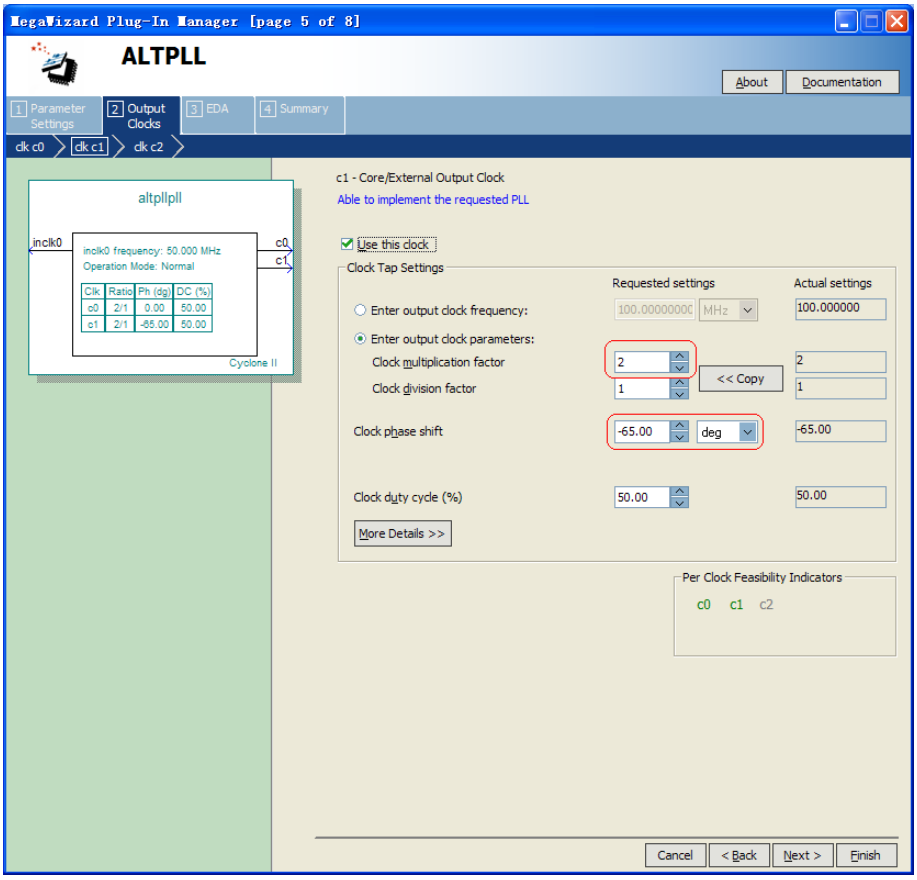


图 10-8　ALTPLL 设置第五页

14. 之后一路 next 直到 MegaWizard finish，回到 PLL 添加页面，如图 10-9，Finish。

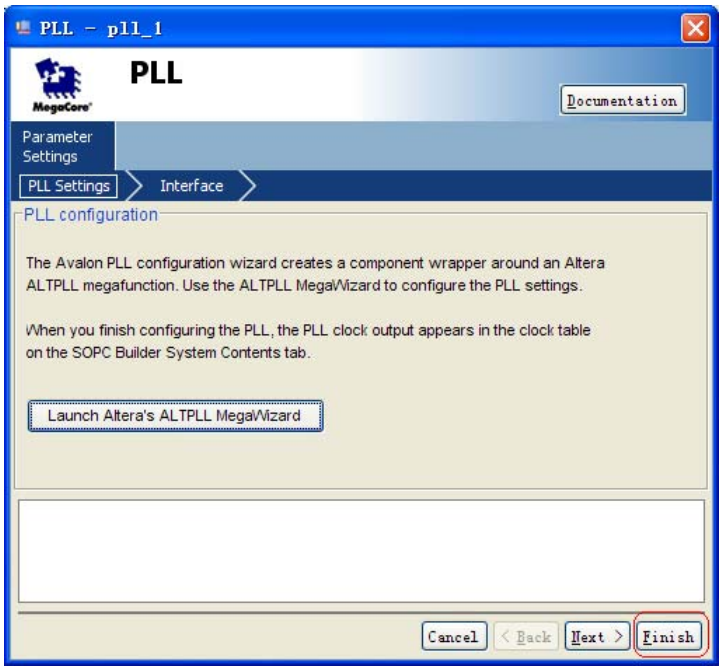


图 10-9　ALTPLL 设置第五页

15. 回到 SOPC 系统视图，在右上方时钟视图中右击对应时钟信号选择 Rename 修改时钟名字，clk 改为 clk_50，pll.c0 改为 pll.c0_system，pll.c1 改为 pll.c1_memory，并将除 pll 外各器件的时钟用下拉框选择系统时钟，如图 10-10.

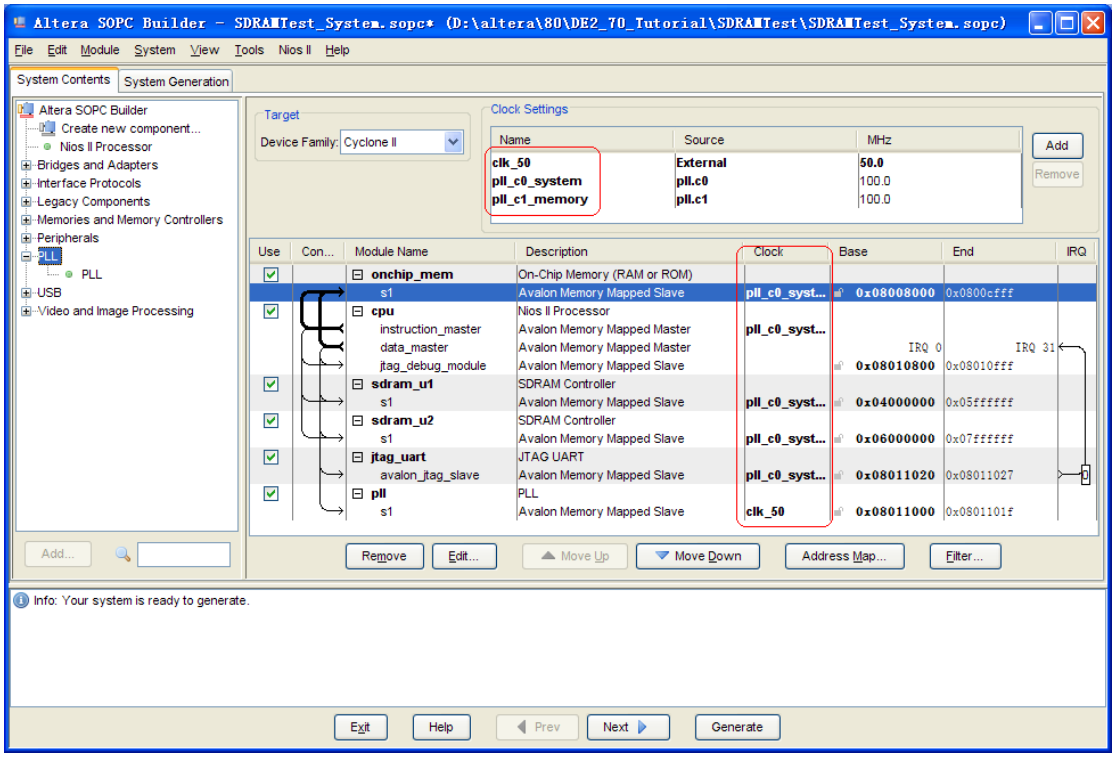


图 10-10　ALTPLL 设置第五页

16. System Auto-Assign Address 分配地址，generate 生成系统。

10.3 完成顶层实体

17. 添加以下代码到 SDRAMTest.v：

```
module SDRAMTest (
   input    iCLK_50，
   input    [0:0]    iKEY，
   inout    [31:0] DRAM_DQ，              // SDRAM Data bus 32 Bits
   output [12:0] oDRAM0_A，             // SDRAM0 Address bus 12 Bits
   output [12:0] oDRAM1_A，             // SDRAM1 Address bus 12 Bits
   output             oDRAM0_LDQM0，       // SDRAM0 Low-byte Data Mask
   output             oDRAM1_LDQM0，       // SDRAM1 Low-byte Data Mask
   output             oDRAM0_UDQM1，       // SDRAM0 High-byte Data Mask
   output             oDRAM1_UDQM1，       // SDRAM1 High-byte Data Mask
   output             oDRAM0_WE_N，        // SDRAM0 Write Enable
   output             oDRAM1_WE_N，        // SDRAM1 Write Enable
   output             oDRAM0_CAS_N，       // SDRAM0 Column Address Strobe
   output             oDRAM1_CAS_N，       // SDRAM1 Column Address Strobe
   output             oDRAM0_RAS_N，       // SDRAM0 Row Address Strobe
   output             oDRAM1_RAS_N，       // SDRAM1 Row Address Strobe
   output             oDRAM0_CS_N，        // SDRAM0 Chip Select
   output             oDRAM1_CS_N，        // SDRAM1 Chip Select
   output [1:0]    oDRAM0_BA，           // SDRAM0 Bank Address
   output [1:0]    oDRAM1_BA，           // SDRAM1 Bank Address
   output             oDRAM0_CLK，         // SDRAM0 Clock
   output             oDRAM1_CLK，         // SDRAM0 Clock
   output             oDRAM0_CKE，         // SDRAM0 Clock Enable
   output             oDRAM1_CKE        // SDRAM1 Clock Enable
);
wire CPU_CLK;

// the sdram is shahred with rtl and nios
assign     oDRAM1_CLK = oDRAM0_CLK;

SDRAMTest_System u0(
    //globalsignals:
    .clk_50(iCLK_50)，
    .pll_c0_system(CPU_CLK)，
    .pll_c1_memory(oDRAM0_CLK)，
    .reset_n(iKEY[0])，

    // the_sdram (u1)
    .zs_addr_from_the_sdram_u1(oDRAM0_A)，
    .zs_ba_from_the_sdram_u1(oDRAM0_BA)，
    .zs_cas_n_from_the_sdram_u1(oDRAM0_CAS_N)，
    .zs_cke_from_the_sdram_u1(oDRAM0_CKE)，
    .zs_cs_n_from_the_sdram_u1(oDRAM0_CS_N)，
    .zs_dq_to_and_from_the_sdram_u1(DRAM_DQ[15:0])，
```

```
    .zs_dqm_from_the_sdram_u1({oDRAM0_UDQM1，oDRAM0_LDQM0})，
    .zs_ras_n_from_the_sdram_u1(oDRAM0_RAS_N)，
    .zs_we_n_from_the_sdram_u1(oDRAM0_WE_N)，

    // the_sdram (u2)
    .zs_addr_from_the_sdram_u2(oDRAM1_A)，
    .zs_ba_from_the_sdram_u2(oDRAM1_BA)，
    .zs_cas_n_from_the_sdram_u2(oDRAM1_CAS_N)，
    .zs_cke_from_the_sdram_u2(oDRAM1_CKE)，
    .zs_cs_n_from_the_sdram_u2(oDRAM1_CS_N)，
    .zs_dq_to_and_from_the_sdram_u2(DRAM_DQ[31:16])，
    .zs_dqm_from_the_sdram_u2({oDRAM1_UDQM1，oDRAM1_LDQM0})，
    .zs_ras_n_from_the_sdram_u2(oDRAM1_RAS_N)，
    .zs_we_n_from_the_sdram_u2(oDRAM1_WE_N)
);
endmodule
```

注意：这种 SDRAM 用法属于两片分开使用，联合一起使用时顶层实体与此不同。

18. 分配引脚

```
set_location_assignment PIN_AD15 -to iCLK_50
set_location_assignment PIN_AC1 -to DRAM_DQ[0]
set_location_assignment PIN_AC2 -to DRAM_DQ[1]
set_location_assignment PIN_AF2 -to DRAM_DQ[10]
set_location_assignment PIN_AF3 -to DRAM_DQ[11]
set_location_assignment PIN_AG2 -to DRAM_DQ[12]
set_location_assignment PIN_AG3 -to DRAM_DQ[13]
set_location_assignment PIN_AH1 -to DRAM_DQ[14]
set_location_assignment PIN_AH2 -to DRAM_DQ[15]
set_location_assignment PIN_U1 -to DRAM_DQ[16]
set_location_assignment PIN_U2 -to DRAM_DQ[17]
set_location_assignment PIN_U3 -to DRAM_DQ[18]
set_location_assignment PIN_V2 -to DRAM_DQ[19]
set_location_assignment PIN_AC3 -to DRAM_DQ[2]
set_location_assignment PIN_V3 -to DRAM_DQ[20]
set_location_assignment PIN_W1 -to DRAM_DQ[21]
set_location_assignment PIN_W2 -to DRAM_DQ[22]
set_location_assignment PIN_W3 -to DRAM_DQ[23]
set_location_assignment PIN_Y1 -to DRAM_DQ[24]
set_location_assignment PIN_Y2 -to DRAM_DQ[25]
set_location_assignment PIN_Y3 -to DRAM_DQ[26]
set_location_assignment PIN_AA1 -to DRAM_DQ[27]
set_location_assignment PIN_AA2 -to DRAM_DQ[28]
set_location_assignment PIN_AA3 -to DRAM_DQ[29]
set_location_assignment PIN_AD1 -to DRAM_DQ[3]
set_location_assignment PIN_AB1 -to DRAM_DQ[30]
```

```
set_location_assignment PIN_AB2 -to DRAM_DQ[31]
set_location_assignment PIN_AD2 -to DRAM_DQ[4]
set_location_assignment PIN_AD3 -to DRAM_DQ[5]
set_location_assignment PIN_AE1 -to DRAM_DQ[6]
set_location_assignment PIN_AE2 -to DRAM_DQ[7]
set_location_assignment PIN_AE3 -to DRAM_DQ[8]
set_location_assignment PIN_AF1 -to DRAM_DQ[9]
set_location_assignment PIN_AA4 -to oDRAM0_A[0]
set_location_assignment PIN_AA5 -to oDRAM0_A[1]
set_location_assignment PIN_Y8 -to oDRAM0_A[10]
set_location_assignment PIN_AE4 -to oDRAM0_A[11]
set_location_assignment PIN_AF4 -to oDRAM0_A[12]
set_location_assignment PIN_AA6 -to oDRAM0_A[2]
set_location_assignment PIN_AB5 -to oDRAM0_A[3]
set_location_assignment PIN_AB7 -to oDRAM0_A[4]
set_location_assignment PIN_AC4 -to oDRAM0_A[5]
set_location_assignment PIN_AC5 -to oDRAM0_A[6]
set_location_assignment PIN_AC6 -to oDRAM0_A[7]
set_location_assignment PIN_AD4 -to oDRAM0_A[8]
set_location_assignment PIN_AC7 -to oDRAM0_A[9]
set_location_assignment PIN_AA9 -to oDRAM0_BA[0]
set_location_assignment PIN_AA10 -to oDRAM0_BA[1]
set_location_assignment PIN_W10 -to oDRAM0_CAS_N
set_location_assignment PIN_AA8 -to oDRAM0_CKE
set_location_assignment PIN_AD6 -to oDRAM0_CLK
set_location_assignment PIN_Y10 -to oDRAM0_CS_N
set_location_assignment PIN_V9 -to oDRAM0_LDQM0
set_location_assignment PIN_Y9 -to oDRAM0_RAS_N
set_location_assignment PIN_AB6 -to oDRAM0_UDQM1
set_location_assignment PIN_W9 -to oDRAM0_WE_N
set_location_assignment PIN_T5 -to oDRAM1_A[0]
set_location_assignment PIN_T6 -to oDRAM1_A[1]
set_location_assignment PIN_T4 -to oDRAM1_A[10]
set_location_assignment PIN_Y4 -to oDRAM1_A[11]
set_location_assignment PIN_Y7 -to oDRAM1_A[12]
set_location_assignment PIN_U4 -to oDRAM1_A[2]
set_location_assignment PIN_U6 -to oDRAM1_A[3]
set_location_assignment PIN_U7 -to oDRAM1_A[4]
set_location_assignment PIN_V7 -to oDRAM1_A[5]
set_location_assignment PIN_V8 -to oDRAM1_A[6]
set_location_assignment PIN_W4 -to oDRAM1_A[7]
set_location_assignment PIN_W7 -to oDRAM1_A[8]
set_location_assignment PIN_W8 -to oDRAM1_A[9]
set_location_assignment PIN_T7 -to oDRAM1_BA[0]
```

```
set_location_assignment PIN_T8 -to oDRAM1_BA[1]
set_location_assignment PIN_N8 -to oDRAM1_CAS_N
set_location_assignment PIN_L10 -to oDRAM1_CKE
set_location_assignment PIN_G5 -to oDRAM1_CLK
set_location_assignment PIN_P9 -to oDRAM1_CS_N
set_location_assignment PIN_M10 -to oDRAM1_LDQM0
set_location_assignment PIN_N9 -to oDRAM1_RAS_N
set_location_assignment PIN_U8 -to oDRAM1_UDQM1
set_location_assignment PIN_M9 -to oDRAM1_WE_N
set_location_assignment PIN_T29 -to iKEY[0]
```

19. 编译下载

10.4 软件设计

20. 打开 Nios II IDE 进行软件设计。将工作空间切换到../SDRAMTest /software。
21. 新建空白工程，添加以下 C 代码到 main.c

```
#include "stdio.h"
#include "system.h"

int main()
{
    short *sdram1=(short*)SDRAM_U1_BASE;
    short *sdram2=(short*)SDRAM_U2_BASE;
    int i;
    int right;
    short temp;

    printf("testing sdram1... \n");
    for(i=0;i<0x1000000;i++)
    {
        *(sdram1+i)=0x55aa;
    }
    right=1;
    for(i=0;i<0x1000000;i++)
    {
        temp=*(sdram1+i);
        if(temp!=0x55aa)
        {
            right=0;
            printf("sdram1 test failed at %d"，i);
            break;
        }
    }
    if(right)
    printf("sdram1 test ok!\n");
```

```
  printf("testing sdram2... \n");
  for(i=0;i<0x1000000;i++)
{
      *(sdram2+i)=0x55aa;
}
right=1;
for(i=0;i<0x1000000;i++)
{
      temp=*(sdram2+i);
      if(temp!=0x55aa)
      {
            right=0;
            printf("sdram2 test failed at %d"，i);
            break;
      }
}
if(right)
printf("sdram2 test ok!\n");

return 0;

}
```

22. 配置编译器参数为-DALT_RELEASE -Os -g –Wall。

23. 配置 System Library Properties。，去掉 Support C++，勾上 Small C library，确认右边使用的是 onchip_mem，没有使用两块 SDRAM，输入输出使用 jtag_uart，如图 10-11 所示

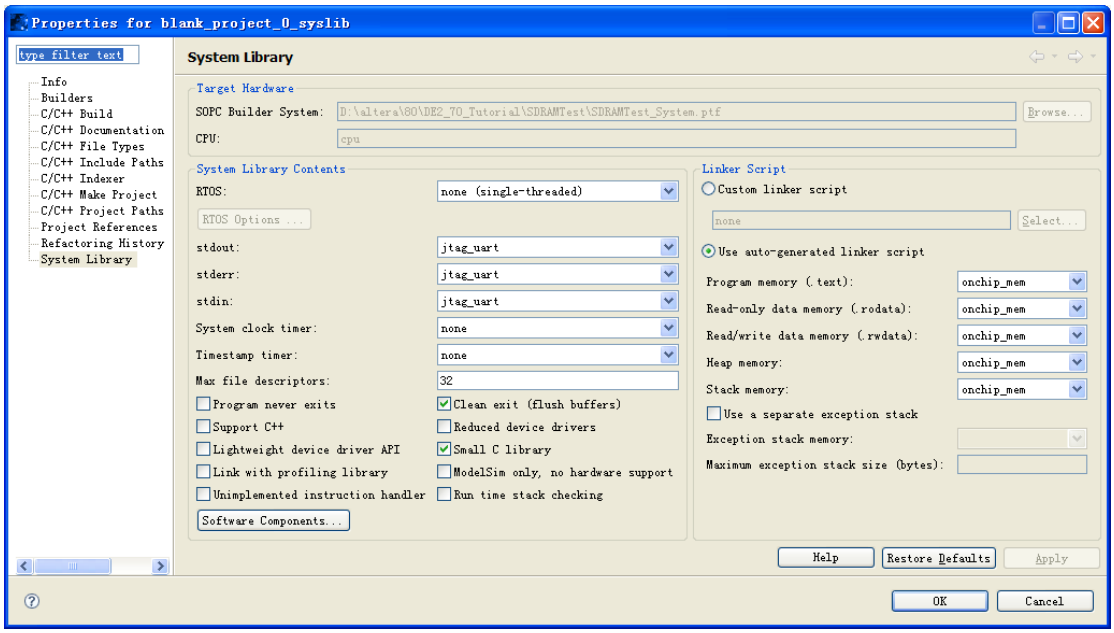


图 10-11　System Library 配置

24. 编译工程并运行之，结果正确。参看图 10-12

```
testing sdram1...
sdram1 test ok!
testing sdram2...
sdram2 test ok!
```

图 10-12　100MHz 下的测试结果

25. 如图 10-13 所示。如果没有按照前文中指示的那样，修改时钟为 100MHz，而使用默认的 50MHz，则会出现下面这种情况：

```
testing sdram1...
sdram1 test ok!
testing sdram2...
sdram2 test failed at 5165
```

图 10-13　50MHz 下的测试结果

这主要是器件切换的时延和 CPU 指令执行的时延不匹配造成的。

附录 1　实验操作快速索引

附录 2　实验各种常见文件格式说明

文件后缀名	用途说明
qpf	Quartus 工程文件，通常只记录 Quartus 版本号、开发日期以及各版本工程名。
qsf	Quartus 工程配置文件，记录各种配置信息，常用来直接编辑添加引脚分配信息。
qws	Quartus 工程工作空间文件，记录当前工作空间与操作系统的一些交互信息，例如强文件名
v	Verilog 源代码文件
vhd	VHDL 源代码文件
bdf	符号框图文件，等效于源代码文件
Vwf	仿真波形文件，记录仿真波形信息
stp	Signal Tap II 逻辑分析仪文件，记录逻辑分析信息
sof	编译结果文件，用于下载到 FPGA 上执行
pof	FPGA 配置文件，用于修改 FPGA 加电启动项
ptf	SOPC Builder 对 Nios II IDE 的接口文件，用于生成 System.h
sopc	SOPC Builder 配置文件，记录 SOPC 系统中各器件的配置信息
qif	SOPC 路径指定文件，主要用于记录自定义 SOPC 模块的路径

本实验指导书编辑说明

本 DE2-70 入门级实验指导书的编写由多位老师和助教参与。最初的电子版实验指导文档来源于上海交通大学电子信息与电气工程学院孙广跃老师主持的 2008 年秋季 FPGA 短期培训班。后来在 2009 年春季，由南京大学计算机系的齐敬先助教、姜孟冯助教和俞建新老师等人整理修改，完成了本实验指导书的第 1.5 版。

由于时间仓促，水平有限，本入门级实验指导书内难免存在一些不足之处。为了更好地让学生在 DE2-70 实验平台上开展实习，欢迎使用者在使用过程中，及时将发现的错误和不妥之处反馈给整理者和编写者。

■