特权和你一起学 NIOS2

欢迎加入 EDN FPGA/CPLD 助学小组共同交流学习：http://group.ednchina.com/1375/

本教程仅适用于特权制造的 SF-NIOS2 FPGA 开发板，转载请注明出处！

版本信息		
时间	说明	备注
2010-9-3	创建	V1.0

第三章 流程实践案例——手把手第一个工程

理清了一些基本概念，也对开发流程有了一个大体的了解，特权同学寻思着下一步该学点什么好呢？要达到循序渐进的效果，是不可以上来就一个大工程，虽然效果可能会比较炫，但是看设计细节估计要吓倒一片。但是，如果一开始就只谈理论，没有能够很好的让大家在实践中对整个开发的流程有一个理性上的认识，好像也不太合适。小时候语文老师教咱们写作文有种结构叫做“总—分—总”，还记得吧？特权同学就要继承这种风格，在大家学会了“写字认字”以后，在咱们的“文章”里开篇就要点题，让大家在总体上对今后要详细学习的知识点都“走马观花”式的实践一遍。

不过请放心，这个实践不会太难为大家，只要稍微有点悟性就能学会。并且后面的教程会如题所示的“手把手”，基本上一傻瓜教程，各位只要手握 SF-NIOS2 开发板，操着鼠标按部就搬即可，重在“体会”流程，不需要你太多的深入细节（也别担心，细节的东西以后会慢慢讲）。

硬件平台

都说到这里了，没有板板的初学者一定要郁闷了（都怪特权，板子没出来，教程就放出来了）。估计大家对新版 SF-NIOS2 学习板垂涎已久，那么特权同学也就先来几张靓照让各位饱饱眼福——

图 3.1 SF-NIOS2 板正面 1

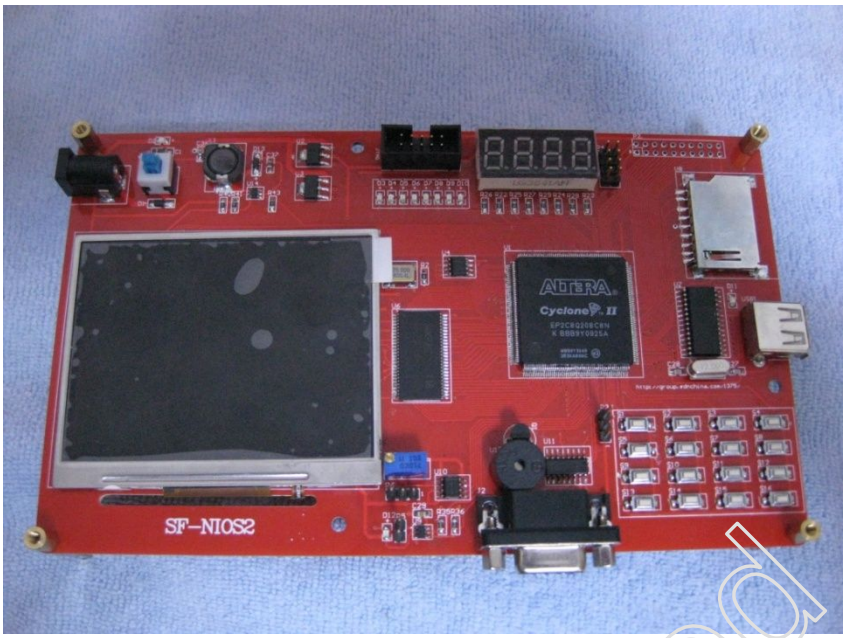


图 3.2 SF-NIOS2 板正面 2

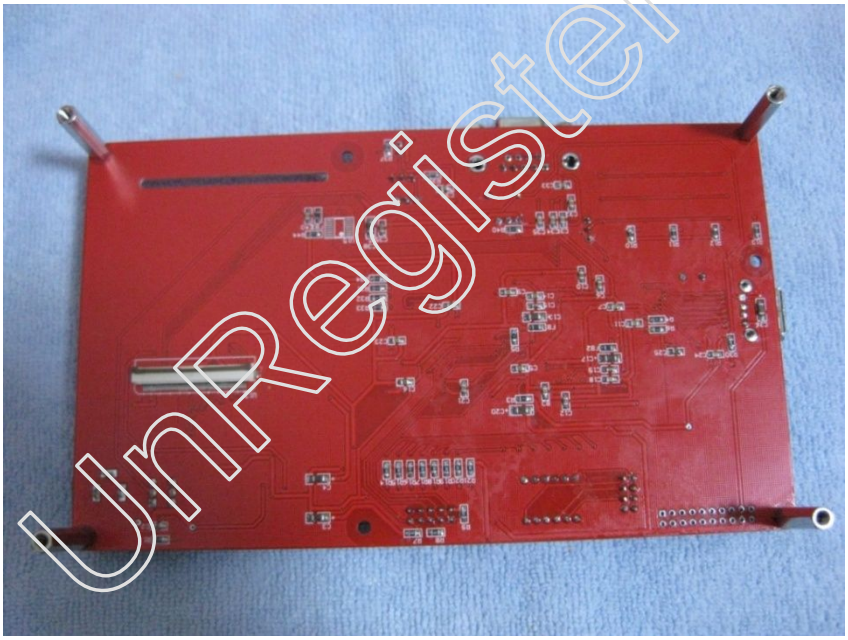


图 3.3 SF-NIOS2 板背面

若想得到此板，还需一段时日（唯一的办法是关注 END 的 FPGA/CPLD 助学小组：http://group.ednchina.com/1375/）。特权同学会就着教程进度对 SF-NIOS2 板进行各个实验例程编写和教程归档的工作。所以，您的期待和声援是对特权同学的最大支持和鼓励。

关于 SF-NIOS2 板的外设介绍不是本章的重点，在后续相关章节中会对相应例程所涉及的硬件外设做说明。

软件平台

光有一块板子，若没有 PC 机上的软件开发工具，哪怕它再神奇再伟大也是徒然。因此，download 一个 WEB 版 Quartus II 和 NIOS2 EDS 是当务之急。当然了，免费的 WEB 版本无论是功能和性能都要大打折扣。如果你或者应该说你们公司腰包够鼓，能买一个正版的，绝对是件值得庆幸的事。但是，买不起也很正常，特权同学给个建议：google 或者 baidu 上加个“骏龙破解”看看有没有什么惊喜。特权同学也很纳闷，骏龙可也是堂堂正正 Altera 在大陆的代理，居然……不过话说回来，不得不承认他们的“服务”做得很到位，比其它竞争对手都更“体恤”中国国情。好了，软件的事情大家自己花时间去琢磨吧，话说“工欲善其事，必先利其器”，特权同学使用的是 Quartus II 9.1 和 Nios II 9.1 Software Build Tools for Eclipse，二者都需要打上 SP1 的补丁，其它的组合或者版本都存在一些让用户很无可奈何的小 bug。v10 也上市了，但我们的口号是“不求最新的，但求最稳定的”。

图 3.4 Quartus II 安装

图 3.5 NIOS II EDS 安装

Quartus II 相信大家都很熟悉了，应该不太有人还在用 MAX+PLUS II 了，特权同学两年前

一开始接触 Q II 是 7.1 版本，压根就没有在 MAX+PLUS II 平台上玩过。这两个东西其实只是 Altera 的开发工具的前后版本而已，界面风格有点区别，其它基本是相通的。说到软件的升级，不光是版本，整个开发平台换代也是常有的事。NIOS2 的开发也在经历着从 IDE 升级到 EDS 的过程中。特权同学一年前玩 NIOS2 只知道 IDE，而且官方的文档也基本是基于 IDE 平台的。当把 IDE 用熟以后，这才发现不对劲，EDS 才是 altera 在主推的新平台，于是就一阵找资料熟悉这个玩意。其实平台的升级也还是换汤不换药的，对于电子工程师，毛主席教导我们要“活到老学到老”，多看资料多尝试咱们一样可以用得很前卫。好了，点到即止，希望每一章都能够给读者带来一些有营养的文字，虽然有时候不细谈，是要先让你知道有那么一回事，这个教程会滔滔不绝没完没了的写下去，特权同学也不确定哪一天会把你想知道的细节展开来发挥。所以，这份教程的一个特点就是：要多给读者留一份期待。哈哈，是不是觉得特权同学太可恶了 !教程就是要做得自由一点，如果还是条条框框的来，不如翻译官方文档算了。不是压力，也不是随意，特权同学是以一种近乎热爱的方式来写这些文字，所以，这些教程中流露出的知识点，对的不对的，好的不好的，大家都可以和我交流。可以发 EDN 站内信给我，可以在小组里留言（http://group.ednchina.com/1375/），当然也可以发 email 给我（wuhouhang@gmail.com），至于电话就不必了，特权同学尤其不喜欢工作的时候被打扰，思路被打断是件很糟糕的事。

在开始后面的手把手工程演示之前，和大家透露个秘密，我想有心的朋友其实都已经发现了这个不是秘密的秘密：但凡市面上的 altera 开发板做教程的第一个 NIOS2 工程，基本都是翻译 tt_nios2_hardware_tutorial.pdf 文档而已，哈哈，特权同学下面这个教程也不例外，虽然不是原本照抄，有一些改动和加工的成分在里面，但确也是换汤不换药的。提这件事并不是让大家以为没有什么原创的元素在里面，而是告诉大家好的东西大家都会去借鉴，并且如果要做得更好也只有站在“巨人”的肩膀上。

再说一件事，也正好是这几天遇上的，和这里要说的意思也沾得上边，所以决定和大家分享一下。有阵子由于项目上可能需要用到 BGA 封装的 FPGA，因此到 altera 官网一阵海找，发现了 http://www.altera.com.cn/literature/an/an114.pdf 这么个不错的 application note。然后昨天在 edn 瞎逛，无意中发现了 lattice 也有个 BGA 布局布线相关的中文资料（链接用搜索关键词“BGA”再去找的时候找不着了），好奇的下载一看，发现怎么内容和文档的结构怎么都那么眼熟啊，仔细一想，这不整个一 altera 的 application note 翻译嘛！话说得有些过激

了，其实谁抄谁，谁译谁，抑或追根问底文档的原创到底是谁，这都没有意义。好的东西就是好，很多流程、规范或知识点那是大家都认可的，所以大家都会以相同的方式向用户展示。我们所能做的，也是借鉴、学习，消化后就是自己的了。

手把手硬件工程

一、新建 Quartus II 工程

1. 打开 Quartus II v9.1，如图 3.6 所示，新建一个工程。

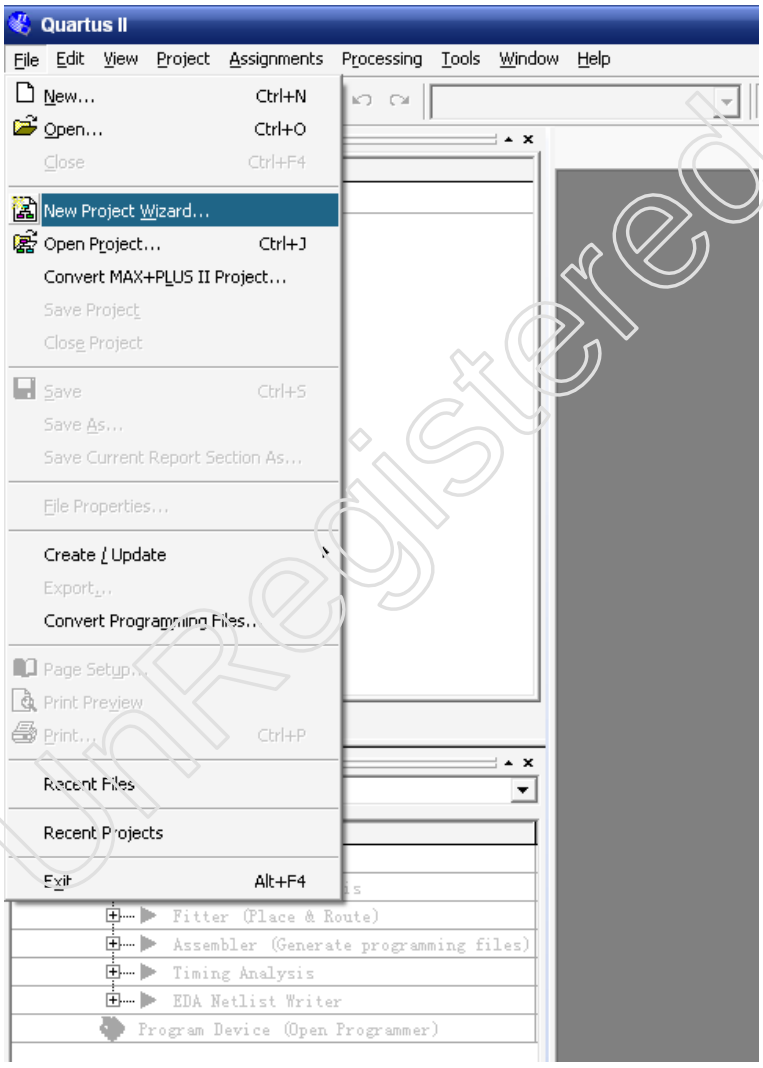


图 3.6 新建 Quartus II 工程

2. 如图 3.7 所示，进行工程目录与工程名配置。用户可以新建一个专用文件夹，该实例用 nios2ex1 文件夹来存放当前的工程文件。新建工程名为 first_q2_prj。在这里再提一下地球人都知道的规矩：工程名以及工程文件所在的相关路径均使用英文字母、阿拉伯数字或者下划线，不要使用其它的符号或汉字。虽然大多数时候好像没有问题，但是要记住 EDA 工具大

都是老外开发的，尤其对中文兼容性不好，特权同学的经验是 Modelsim 仿真中如果出现中文路径基本是编译不了的。所以，记住这个规则吧，会让你少很多郁闷的。

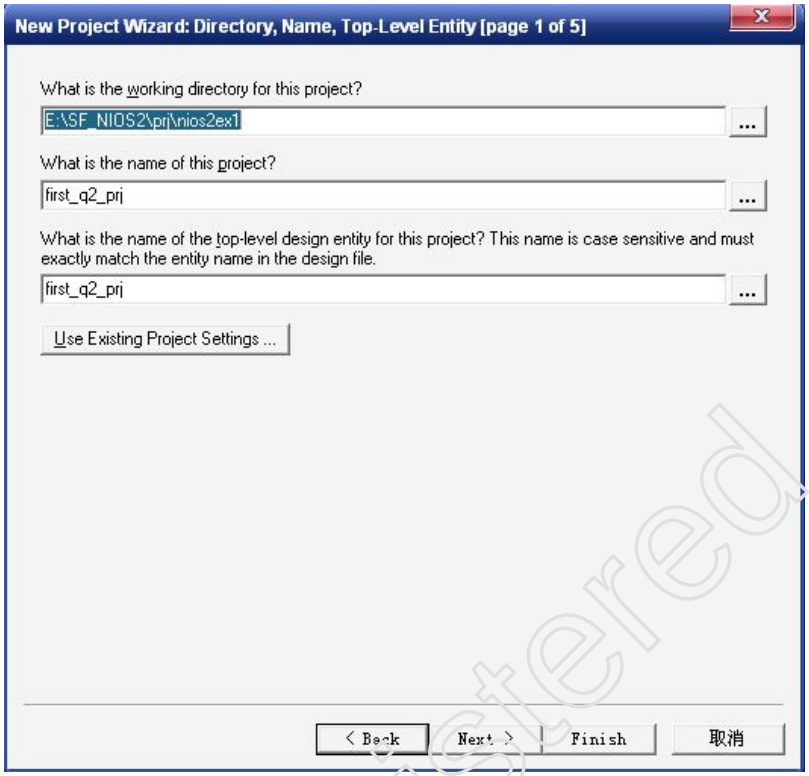


图 3.7 工程目录与工程名配置

3. 如图 3.8 所示，进行 FPGA 器件选择配置。该实例使用了特权打造的 SF-NIOS2 FPGA 板，其上的 FPGA 为 Cyclone II 家族 EP2C8Q208C8，其他地方使用默认配置。点击 Finish 完成 Quartus II 工程新建向导的设置。

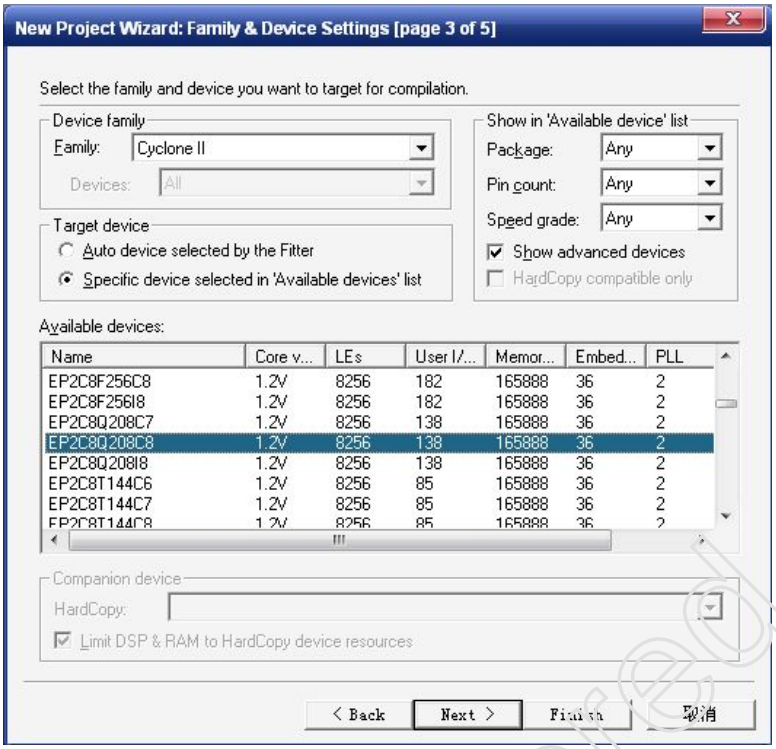


图 3.8 器件选择配置

既然这里说到了器件选择，那么就应该让大家对所使用的 EP2C8Q208C8 这个型号有一点了解，至少要明白它命名的原则。简单的说，EP2C 是 Cyclone II 系列器件的代号，整个 Cyclone（有 Cyclone/Cyclone Ⅱ/Cyclone Ⅲ/Cyclone Ⅳ）系列是面向低成本应用的高性价比解决方案，因此也可以说是普及率最高的一个系列了。紧跟 EP2C 后的 8 代表着逻辑资源数量，对于 Cyclone II 系列的 8 代表 8256 个 LEs。往后的 Q208 代表着封装和管脚信息，即 TQFP208。后面的 C（Commerce）表示器件等级，即商业级。最后的 8 表示速度等级，这个系列一般可以是-6/-7/-8。所谓速度等级，就是 FPGA 器件理论可以达到的一个频率水平。

二、SOPC Builder 配置

1. 创建好一个工程，我们接着就使用 SOPC Builder 给这个工程搭建一个硬件系统。如图 3.9 所示，点击菜单栏的 Tools→SOPC Builder 选项，进入 SOPC 配置页面。

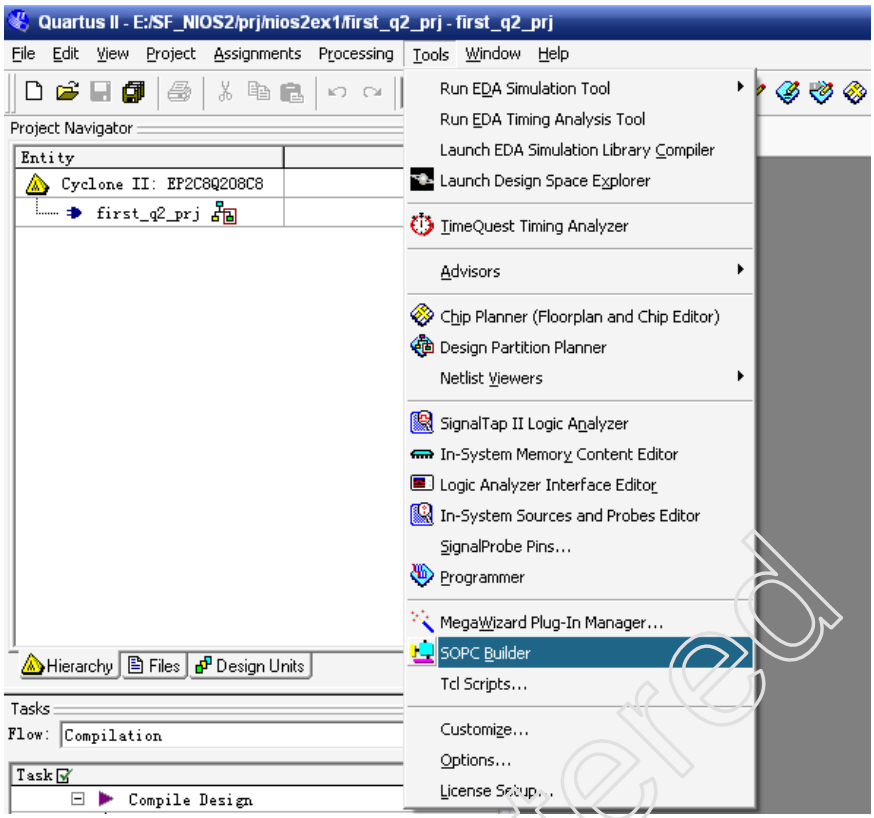


图 3.9 打开 SOPC Builder

2. 进入 SOPC Builder 配置页面后，首先映入眼帘的是如图 3.10 所示的创建新系统的界面。本实例输入系统名为 first_nios2_sys。点击 OK 完成系统新建。

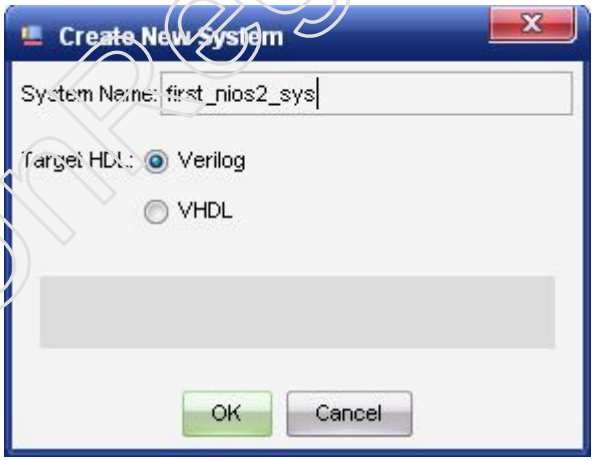


图 3.10 新建系统

三、添加和配置 NIOS2 处理器和通用外设

在配置这个硬件系统之前，我们对要创建的系统架构一定要心中有数，特权心里是有数了，各位同学可不能没数，呵呵。这个系统架构如图 3.11 所示，核心肯定是中中间那个 NIOS2 处理器，一个最小的嵌入式系统无非还要加上时钟（包括复位单元）单元、存储器（用于存

储代码的 ROM 和存放变量的 RAM）单元、还有电源也很重要，只不过我们的系统是在 FPGA 内部，电源系统已经是 FPGA 的一部分了，不需要我们在器件编程时考虑。出了这几项，还需要一个下载调试的接口，如该系统的 JTAG Uart，说白了也就是一个串口。另外有一个叫做 System ID 的组件，它能够帮助调试过程中确认系统是否正常工作。一般调试中会读读该系统的 ID，如果正确则说明系统握手成功，如果错误则系统握手失败，那么就没有必要接着调试下去了。定时器咱也不说了，GPIO 接着 8 个 LED 做演示，系统就这么简单，关键是要用心去体会这个“流程”。

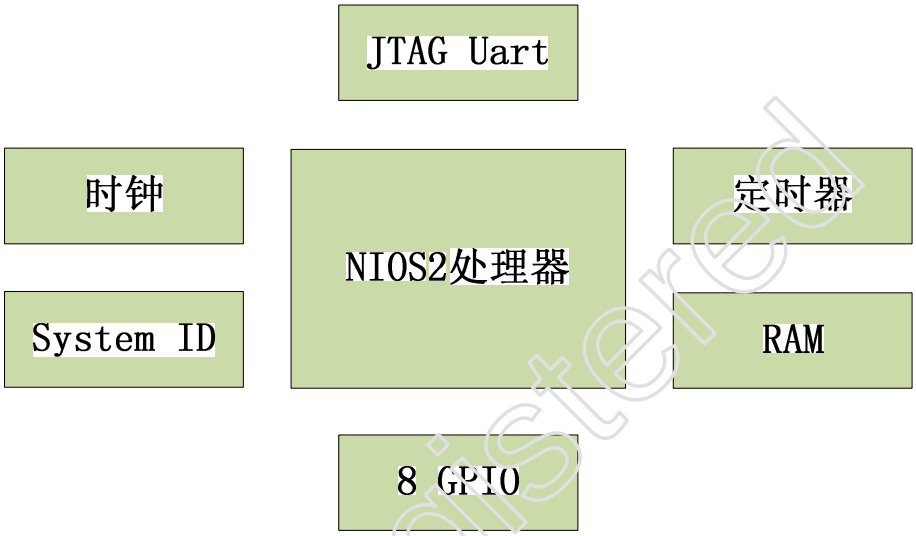


图 3.11 系统架构图

如图绝对图 3.11 有点山寨了，那么不妨看看官方给出的该实例的框图（如图 3.12）。

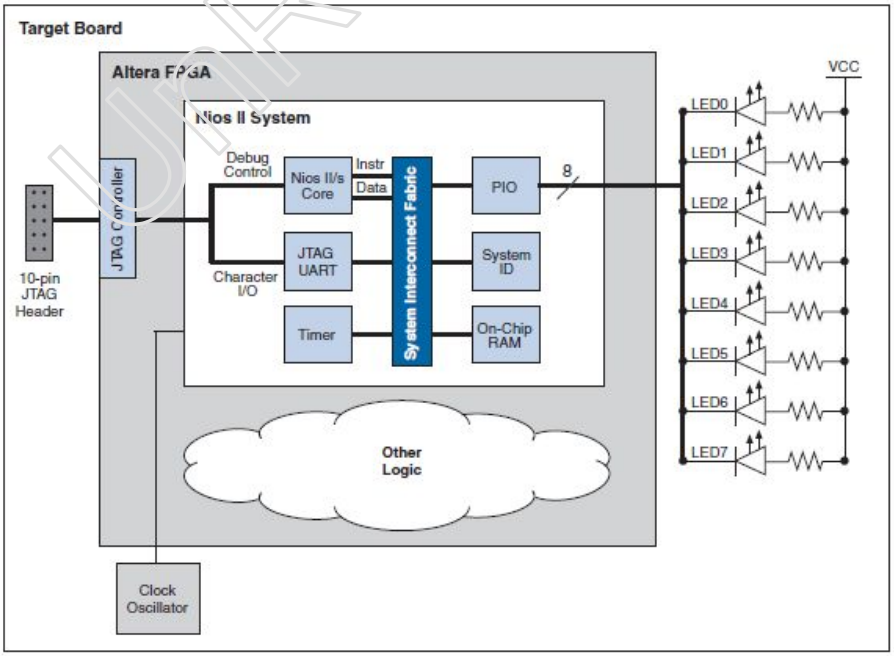


图 3.12 系统架构框图

1. 首先，如图 3.13 所示，进行系统时钟配置。双击时钟名称或时钟频率即可进行修改，这里修改时钟名称为 clk，时钟频率为 25MHz，与 SF-NIOS2 FPGA 板实际所外接的晶振频率一致。

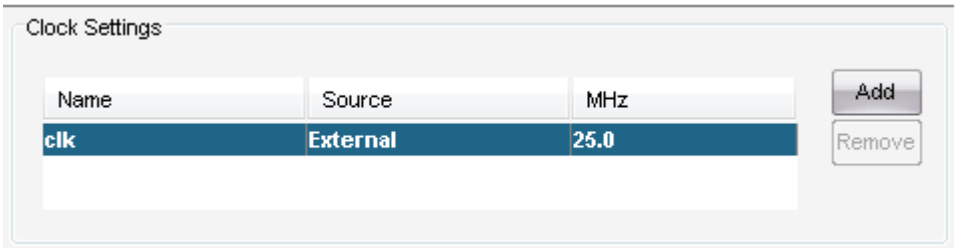


图 3.13 配置系统时钟

2. 添加配置 NIOS2 片内存储器。

如图 3.14 所示，双击组件库中的 On-Chip Memory(RAM or ROM)，添加片内存储器组件。

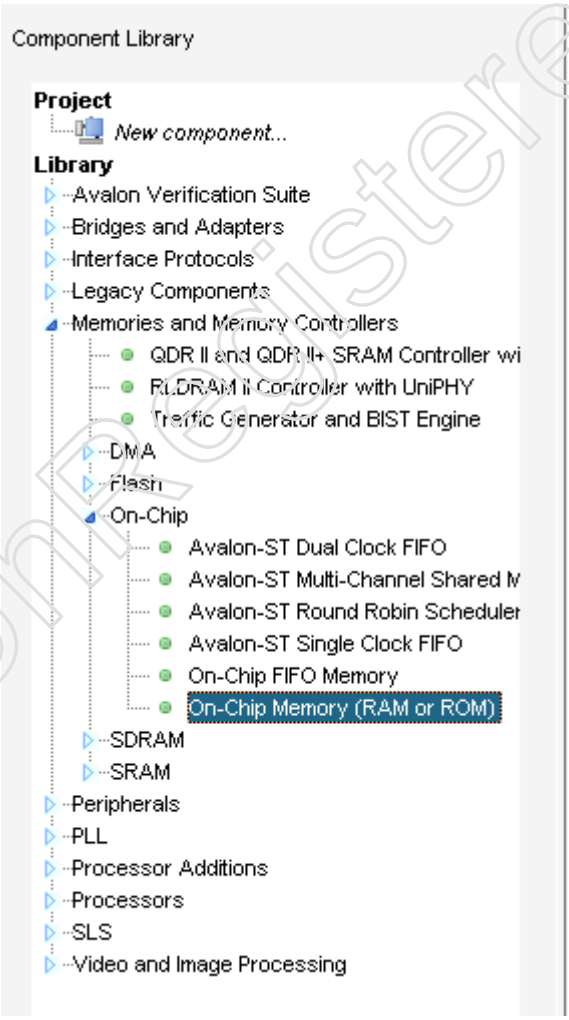


图 3.14 双击添加片内存储器组件

对片内存储器的配置如图 3.15 所示。选择片内存储器类型（Memory type）为 RAM(Writable)；使用的资源（Block type）为 M4K；数据位宽（Data Width）为 32；总存储

大小为（Total memory size）12Kbytes。其他使用默认设置即可，点击 Finish 完成配置。

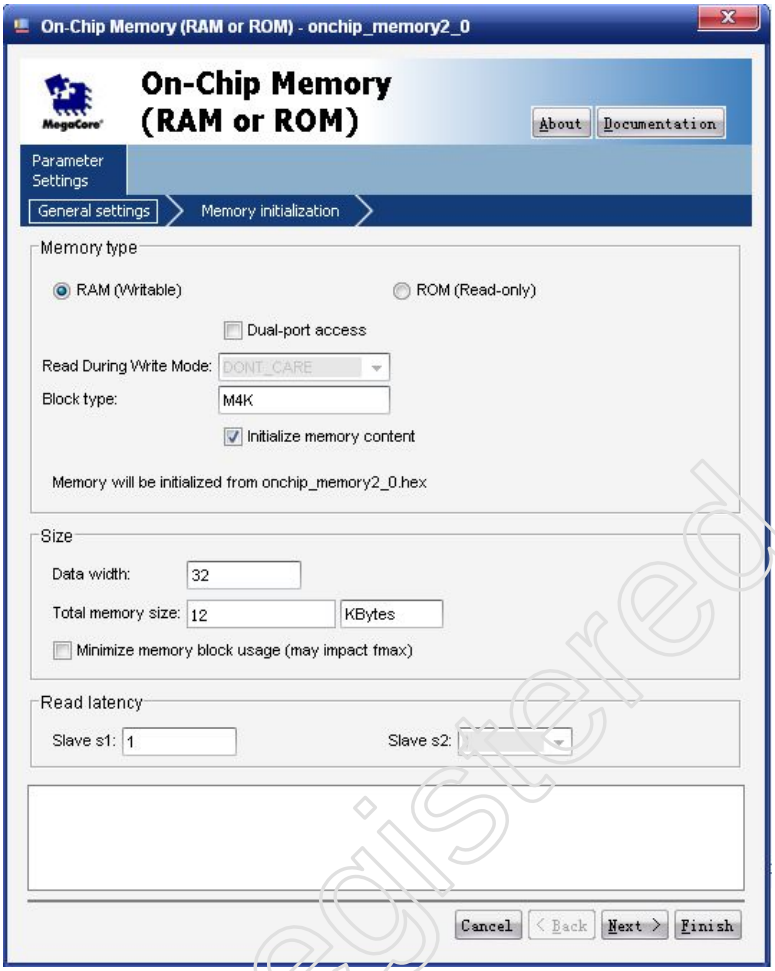


图 3.15 配置片内存储器

如图 3.16 所示，鼠标移动到片内存储器的名称 onchip_memonry2.0 上并点击右键，在弹出菜单中选择 Rename，然后更改名称为 onchip_mem。其它配置暂不做更改，默认即可。

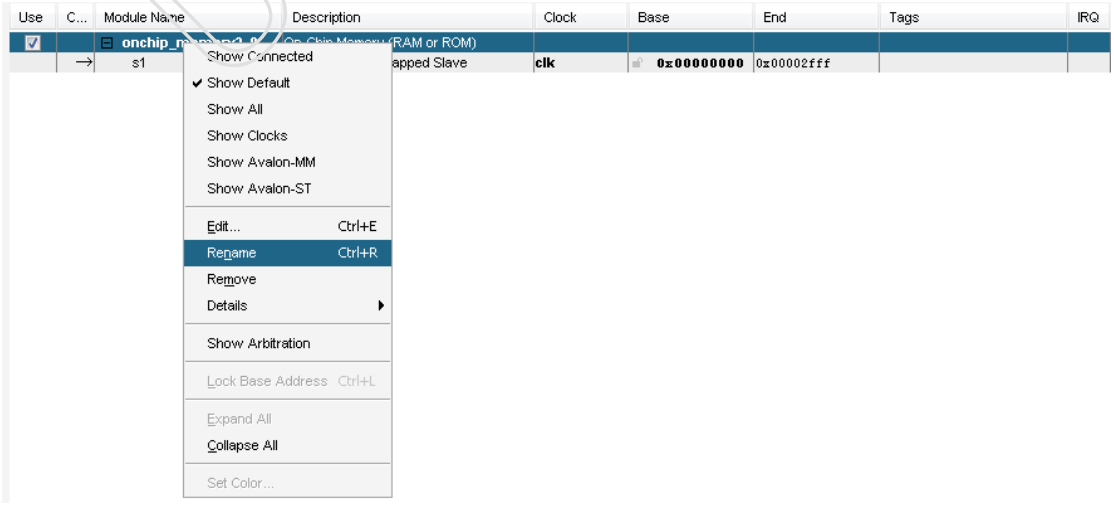


图 3.16 更改存储器名

3. 添加配置 NIOS2 处理器核。

如图 3.17 所示，双击组件库中的 NIOS II Processor，添加 NIOS II 处理器组件。

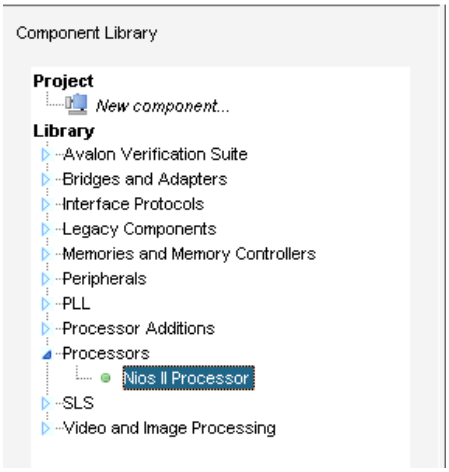


图 3.17 双击添加处理器组件

对 NIOS2 处理器进行图 3.18 所示的设置。处理器选择 NIOS II/s；硬件乘法器（Hardware Multiply）选择 None，即不需要硬件乘法器；复位向量和异常向量存储器（Memory）均选择前面刚刚添加的片内存储器 onchip_mem，此时二者的偏移量（Offset）自动设置为 0x0 和 ox20。复位向量是指整个系统软件复位后启动的程序地址，一般为非易失存储器。异常向量是软件的起始地址，一般是易失存储器，如 SDRAM 等。

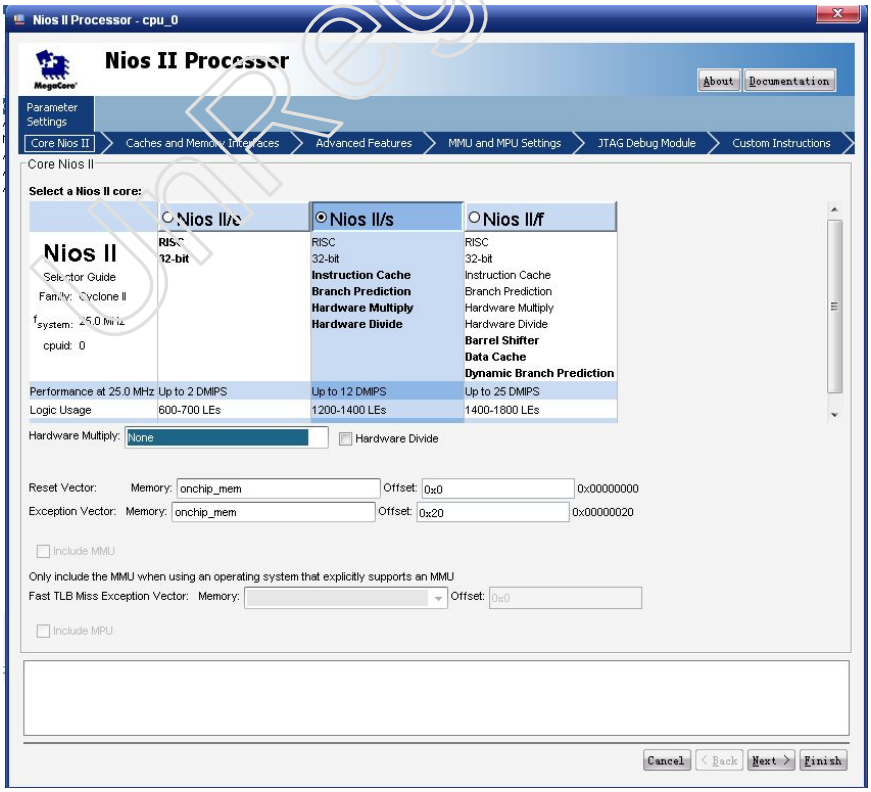


图 3.18 配置 NIOS2 处理器

如图 3.19 所示，更改指令缓存（Instruction Cache）为 2Kbyte，其他地方使用默认设置。最后点击 Finish 完成处理器设置。

图 3.19 配置指令缓存

更改 NIOS2 处理器组件名为 cpu。

4. 配置定时器。

如图 3.20 所示，双击组件库中的 Interval Timer，添加定时器组件。

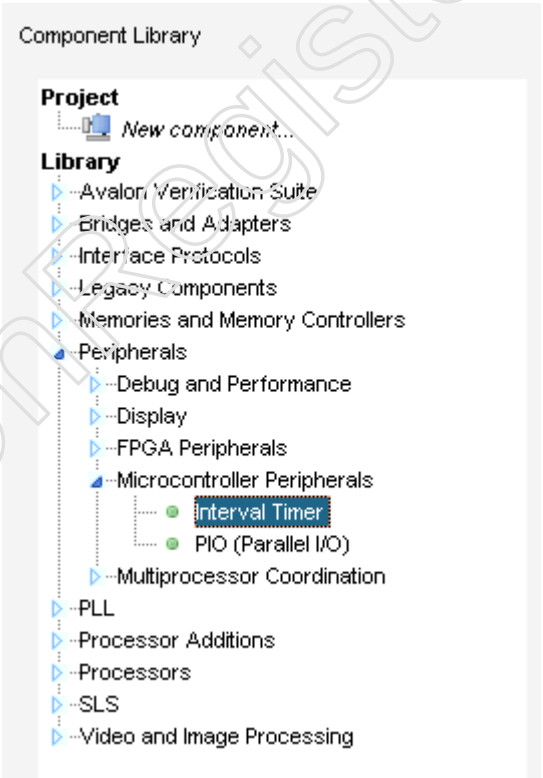


图 3.20 选择定时器组件

如图 3.21 所示，配置定时器组件。只要更改预置（Presets）选项为 Full-featured 即可，其他地方使用默认设置，最后点击 Finish 完成处理器设置。

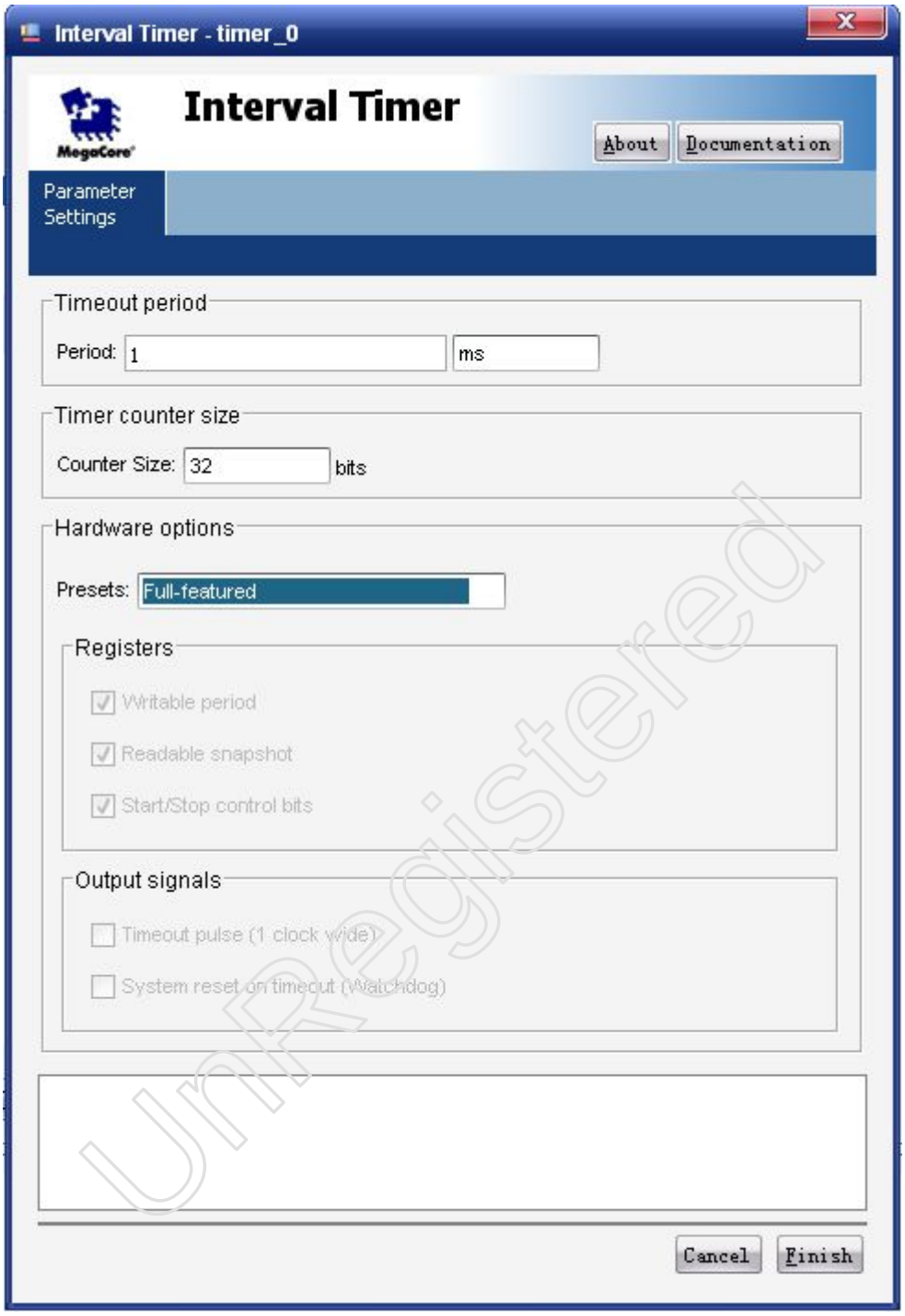


图 3.21 配置定时器

更改定时器组件名为 sys_clk_timer。

5. 配置 JTAG UART。

如图 3.22 所示，双击组件库中的 JTAG UART，添加 JTAG UART 组件。

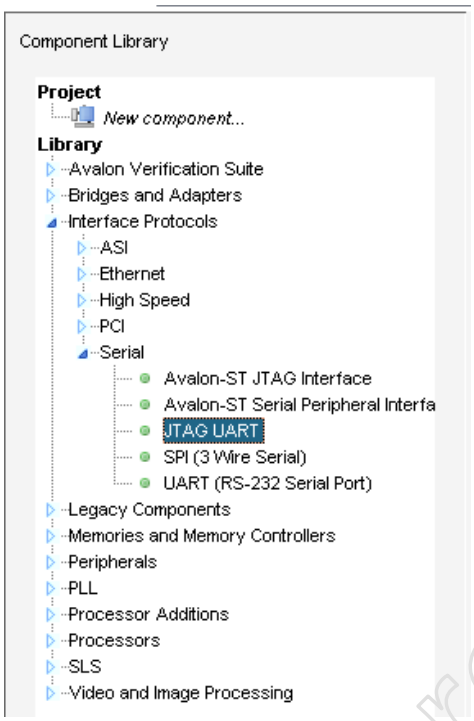


图 3.22 添加 JTAG UART 组件

如图 3.23 所示，JTAG UART 的配置均使用默认设置，点击 Finish 完成配置。

图 3.23 配置 JTAG UART

更改 JTAG UART 组件名为 jtag_uart。

6. 配置 8 位并行 I/O（PIO）。

如图 3.24 所示，双击组件库中的 PIO(Parallel I/O)，添加 PIO 组件。

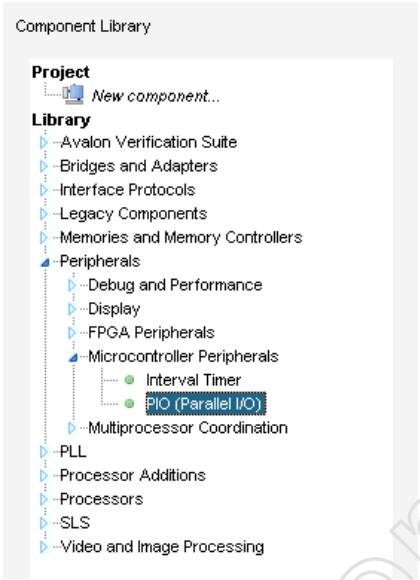


图 3.24 添加 PIO 组件

PIO 配置如图 3.25 所示，数据位宽（Width）选择 8；管脚方向（Direction）选择输出（Output ports only）；复位值（Reset value）为 0x0；其他选项为默认设置。点击 Finish 完成配置。

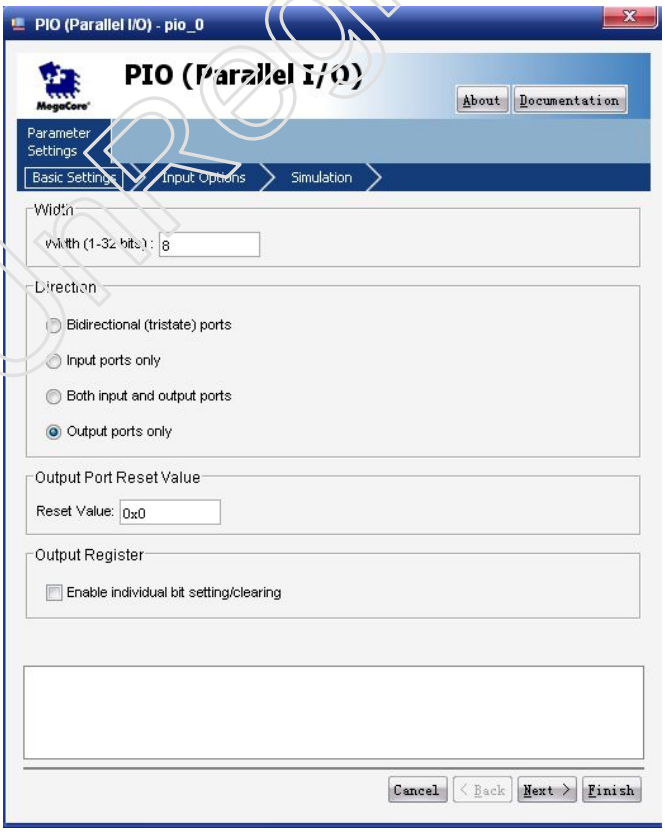


图 3.25 配置 PIO

更改 PIO 组件名为 led_pio。

7. 配置系统 ID。

如图 3.26 所示，双击组件库中的 System ID Peripheral，添加系统 ID 组件。

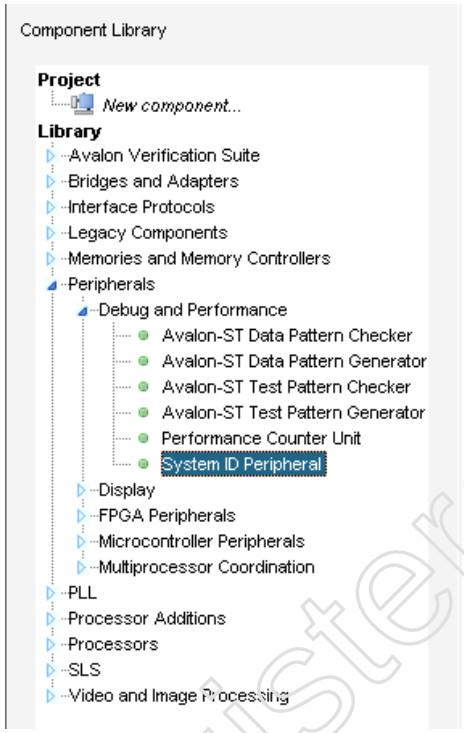


图 3.26 添加系统 ID 组件

如图 3.27 所示，无需做任何设置，直接点击系统 ID 配置窗口的 Finish 按钮即可。不过请注意其中的警告信息。是的，在配置完成后，务必更改系统 ID 名称为 sysid。

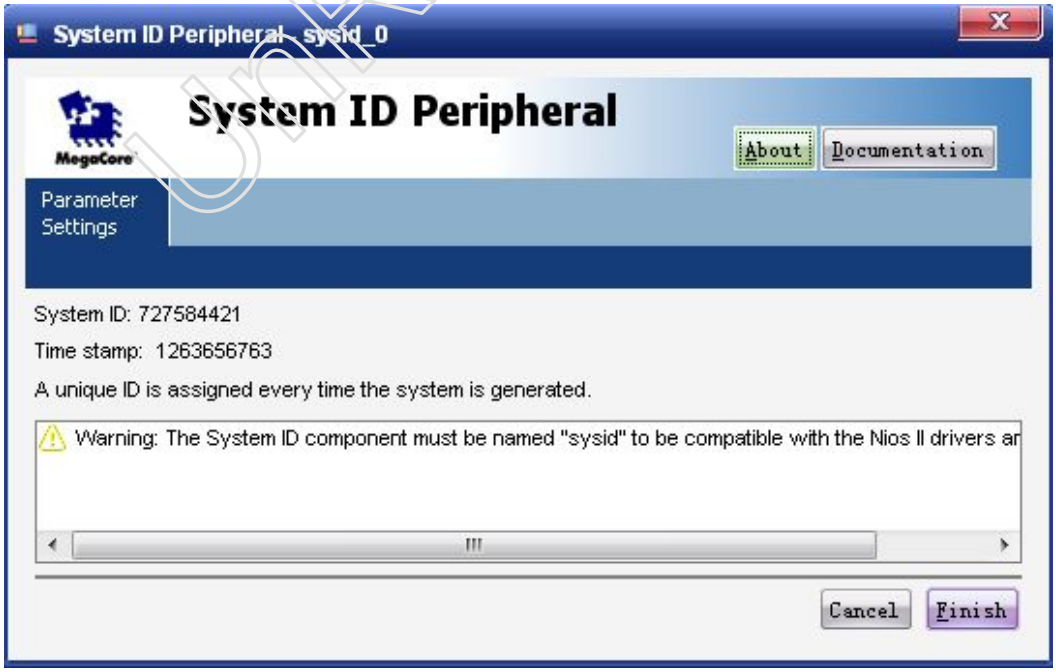


图 3.27 配置系统 ID

8. 自动分配地址

如图 3.28 所示，点击菜单栏的 System→Auto-Assign Base Addresses 选项，自动分配处理器和各个外设的地址。或者用户也可以直接点击地址栏进行更改。

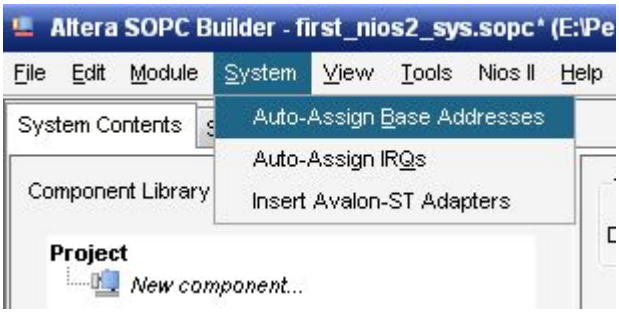


图 3.28 自动分配地址

9. 自动中断优先级设置

如图 3.29 所示，点击菜单栏的 System→Auto-Assign IRQs 选项，自动分配各个外设间的中断优先级（如果有中断）。或者用户也可以直接点击中断栏进行更改。

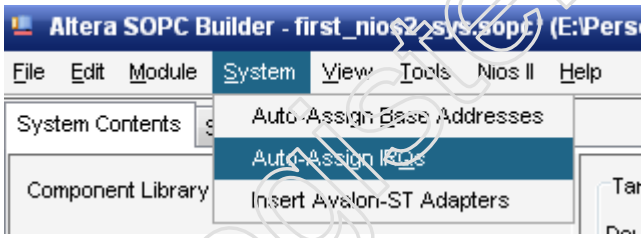


图 3.29 自动分配中断优先级

10. 生成系统

所有配置完成，有心的同学一定发现了当鼠标落在 Connection 一列时，如图 3.30 所示，系统的互联结构出现了，这些连接都是在我们添加组件后系统自动产生的。至于为什么如此连接已经连接的基本原理等，这里不过多讨论，后续教程会详细谈论。

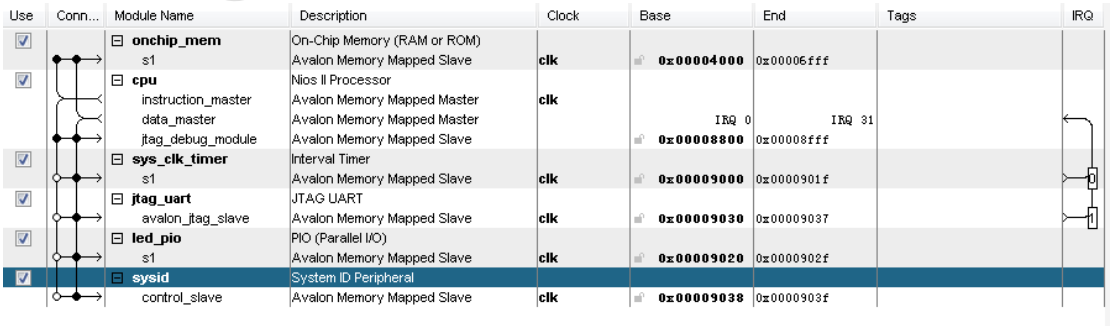


图 3.30 系统互联架构

如图 3.31 所示，最后点击配置页面的下方 Generate 按钮，需要等待几分钟，在 System Generation 的 Info 中若出现 System generation was successful 则说明系统成功生成。

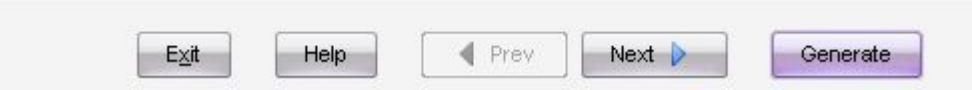


图 3.31 生成系统

四、例化 NIOS2 工程

1. 在 Quartus II 中新建一个 verilog 文档，作为工程的顶层文件，命名为 first_q2_prj.v。再新建一个 sys_ctrl.v 文档，在其中编写一个系统复位的设计逻辑，并将该模块例化到顶层文件中。复位模块的代码如下：

```
module sys_ctrl(
                clk,ext_rst_n,
                rst_n
            );

input clk;      //25MHz
input ext_rst_n;    //外部输入复位信号，低有效
output rst_n;   //低电平复位信号

//-------------------------------------------------
reg[9:0] rst_cnt;   //上电延时计数器
reg rst_nr; //复位寄存器，低有效

always @ (posedge clk or negedge ext_rst_n) //异步复位
    if(!ext_rst_n) begin
            rst_cnt <= 10'd0;
            rst_nr <= 1'b0; //复位中
        end
    else if(rst_cnt != 10'd1000) rst_cnt <= rst_cnt+1'b1;   //计时到 40us
    else rst_nr <= 1'b1;    //复位完成

assign rst_n = rst_nr;

endmodule
```

2. 如图 3.32 所示，打开一个已经生成的 NIOS2 系统的例化模板，将其内容复制到

first_q2_prj.v 中进行例化。

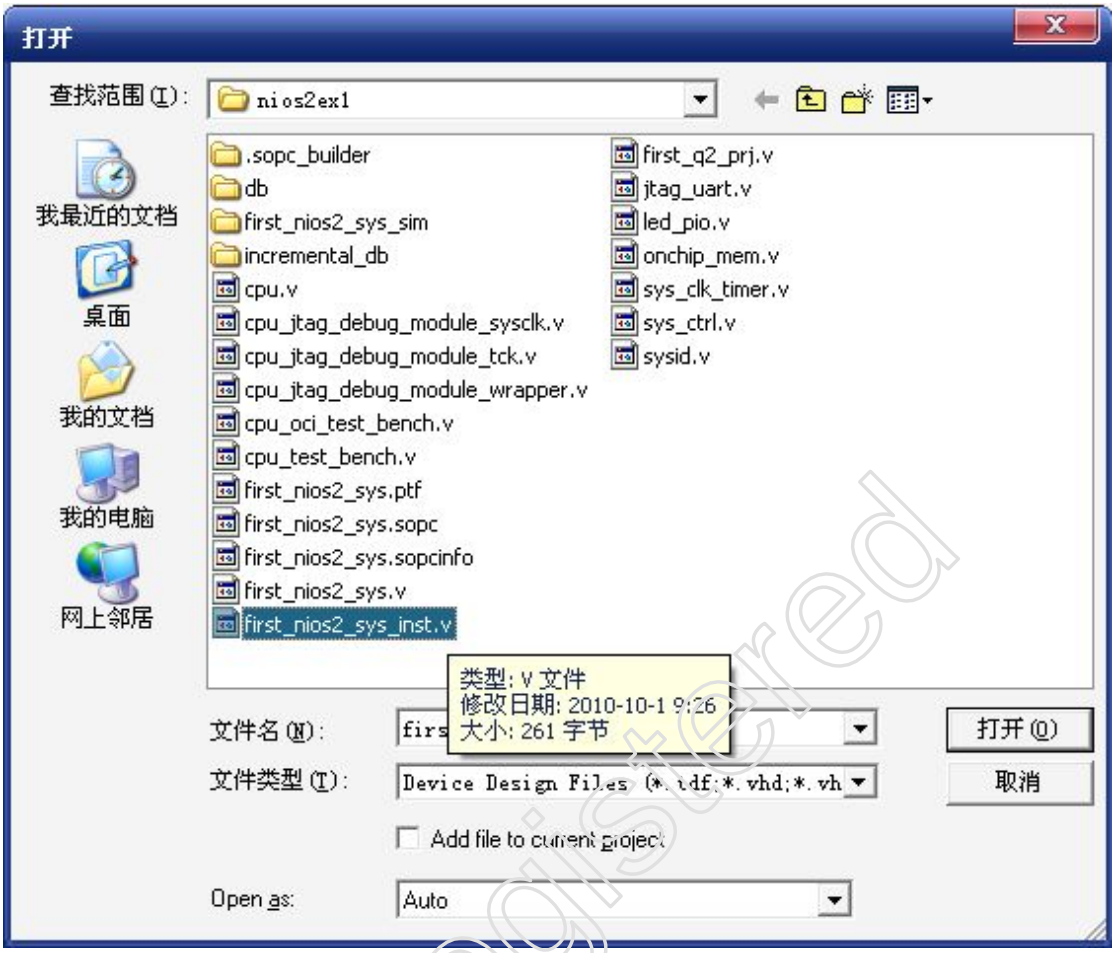


图 3.32 打开 NIOS2 系统例化模板

打开的 NIOS2 系统例化模板的代码如下：

```
//Example instantiation for system 'first_nios2_sys'
first_nios2_sys first_nios2_sys_inst
  (
    .clk                       (clk),
    .out_port_from_the_led_pio (out_port_from_the_led_pio),
    .reset_n                   (reset_n)
  );
```

对其进行相应的接口例化更改后如下：

```
//nios2 系统例化
first_nios2_sys first_nios2_sys_inst
  (
    .clk                       (clk),
    .out_port_from_the_led_pio (led_pio),
```

```
      .reset_n                    (rst_n)
    );
```

最终得到的工程顶层模块代码如下：

```
module first_q2_prj(
                clk,ext_rst_n,
                led_pio
            );

input clk;      //25MHz
input ext_rst_n;    //外部输入复位信号，低有效

output[7:0] led_pio;    //8 位流水灯，0--点亮

//-------------------------------------------------
wire rst_n; //低电平复位信号

//-------------------------------------------------
//复位产生模块
sys_ctrl        uut_sc(
                    .clk(clk),
                    .ext_rst_n(ext_rst_n),
                    .rst_n(rst_n)
                );

//-------------------------------------------------
//nios2 系统例化
first_nios2_sys first_nios2_sys_inst
    (
      .clk                      (clk),
      .out_port_from_the_led_pio (led_pio),
      .reset_n                  (rst_n)
    );

endmodule
```

五、分配管脚与编译下载

1. 对上述例化并编写好的代码进行综合，然后点击 Assignments→Pins 进行管脚分配。分配完管脚如图 3.33 所示。

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard
clk	Input	PIN_23	1	B1_N0	3.3-V LVTTL (default)
ext_rst_n	Input	PIN_206	2	B2_N1	3.3-V LVTTL (default)
led_pio[7]	Output	PIN_179	2	B2_N0	3.3-V LVTTL (default)
led_pio[6]	Output	PIN_176	2	B2_N0	3.3-V LVTTL (default)
led_pio[5]	Output	PIN_175	2	B2_N0	3.3-V LVTTL (default)
led_pio[4]	Output	PIN_173	2	B2_N0	3.3-V LVTTL (default)
led_pio[3]	Output	PIN_171	2	B2_N0	3.3-V LVTTL (default)
led_pio[2]	Output	PIN_170	2	B2_N0	3.3-V LVTTL (default)
led_pio[1]	Output	PIN_169	2	B2_N0	3.3-V LVTTL (default)
led_pio[0]	Output	PIN_168	2	B2_N0	3.3-V LVTTL (default)

图 3.33 管脚分配

2. 如图 3.34 所示，双击 Assembler(Generate programming files)编译整个工程，并产生下载用的*.pof 文件。当然了，在编译过一次工程后，需要做一些时序约束并达到时序收敛才可以继续往下走。这个步骤由于涉及的知识点比较深，在这个流程实例中就不多演示，希望大家能现有这样一个概念，后面的教程里应该会华专门的一些篇幅进行讨论。

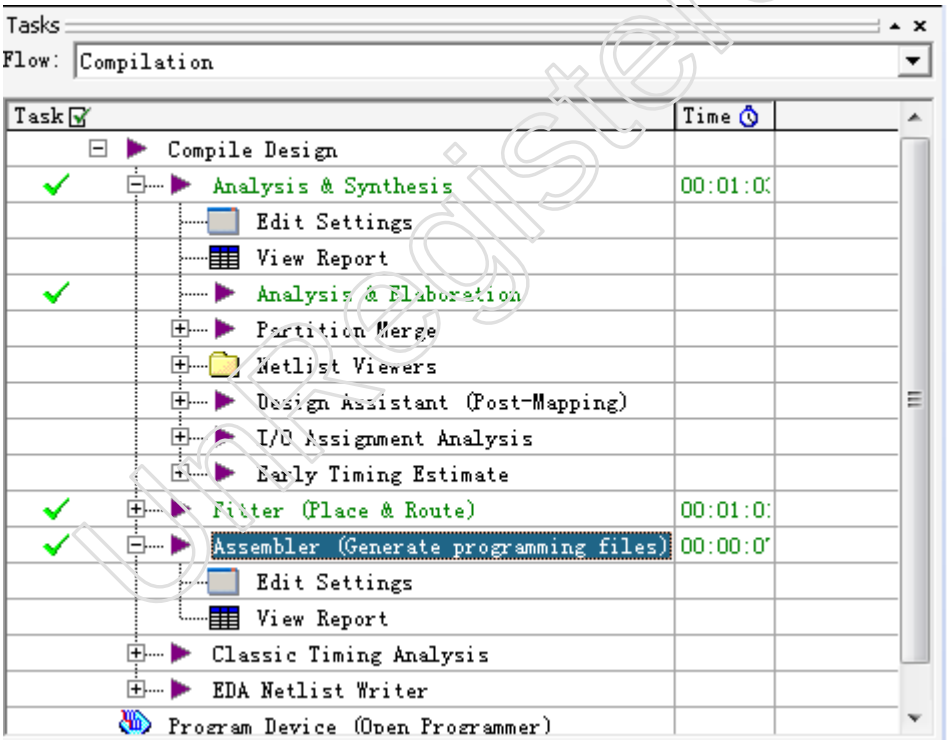


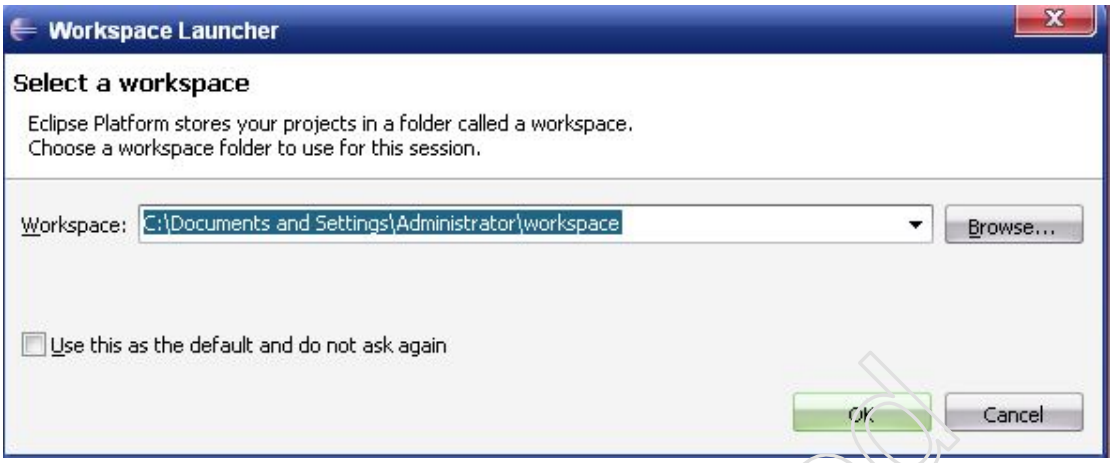
图 3.34 编译产生下载文件

3. 确保 Blaster II（并口）下载线或 USB Blaster 下载线与 SF-NIOS2 FPGA 板连接好，并给板子上电。同时下载刚刚生成的*.pof 文件到开发板中。

手把手软件工程

一、新建软件模板工程

1. 打开软件开发工具 Nios II 9.1 Software Build Tools for Eclipse（简称 EDS）。第一次打开 EDS 软件，会弹出如图 3.35 所示的 workspace 选择框，点击 OK 即可，不要对路径做任何更改。或者也可以勾选“Use this as the default and do not ask again”。



2. 如图 3.36 所示，新建一个 NIOS II Application and BSP from Template 工程。

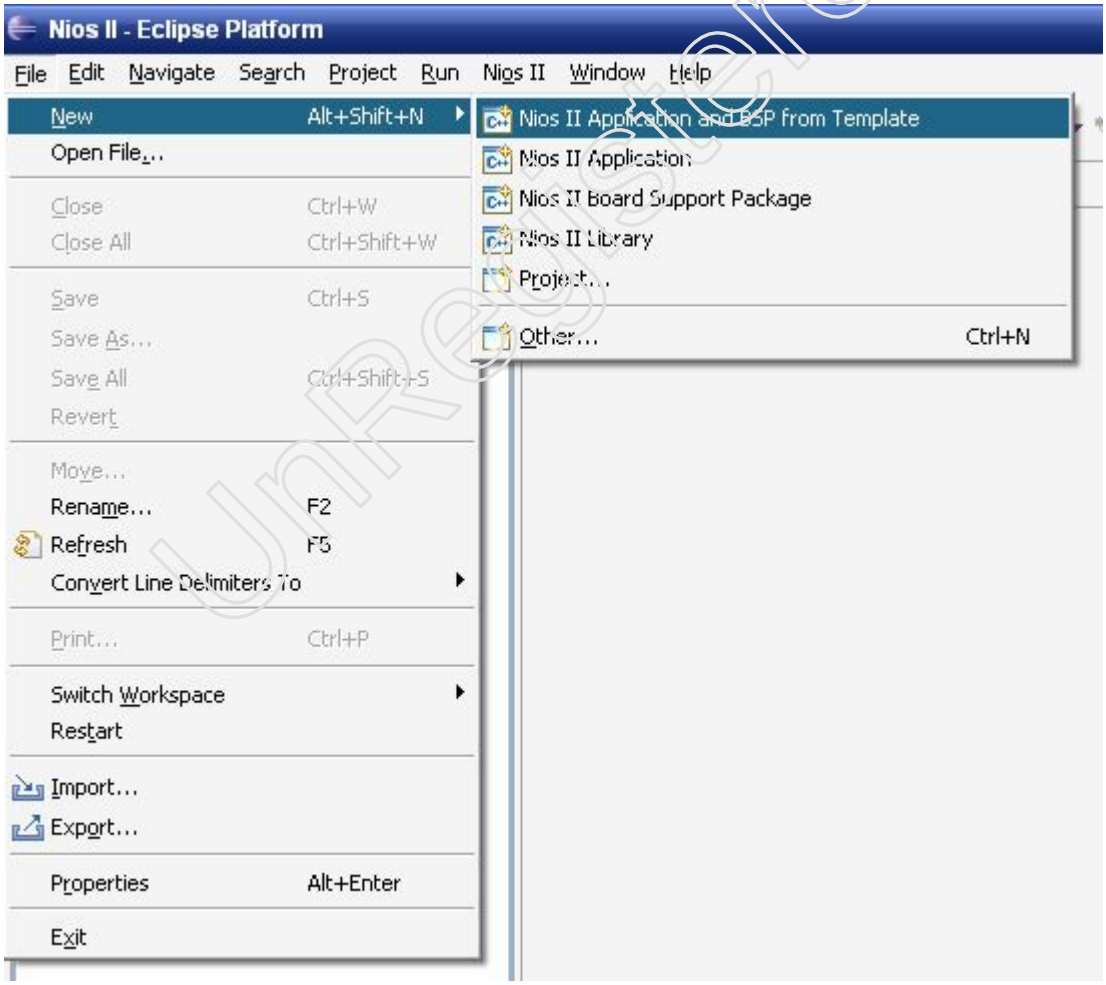


图 3.36 新建 NIOS2 工程

2. 如图 3.37 所示，对新建的 NIOS2 软件工程进行设置。点击 SOPC Information File name 一

栏后的按钮，找到硬件工程所在目录下的 first_nios2_sys.sopcinfo 文件，这里硬件和软件之间就是通过这个.sopcinfo 文件进行关联，就这么简单，个中奥秘还有待各位花精力自己去研究。指定了.sopcinfo 文件后，CPU name 自动显示为“cpu”。在 Project name 一栏输入软件工程名为“first_swprj”。使用默认的软件工程存放目录，即在硬件工程目录下生成一个名为 software 的目录用于存放软件工程。选择工程模板（Project Template）中的 Count Binary。然后点击 Finish 完成工程新建。

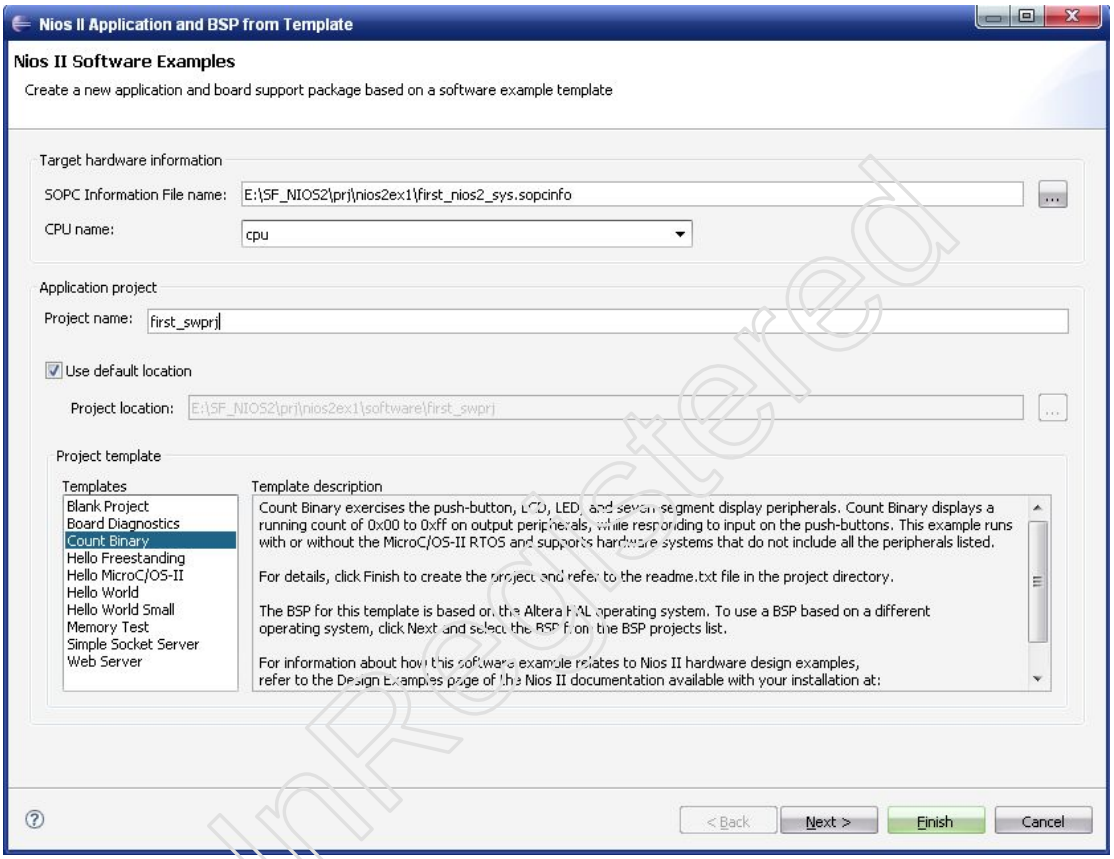


图 3.37 配置新建工程

3. 等待一会后，如图 3.38 所示，在工程目录窗口生成了两个工程，一个是软件应用工程，另一个是 bsp 工程。前者用于工程师编写程序；后者一般是系统自动产生，无需用户手动编辑，他主要是根据不同的硬件外设配置产生很多底层驱动相关的程序，在应用层只要调用这些程序就可以了，因此可以说 bsp 工程的主要作用就是做底层驱动，衔接应用层和硬件层。

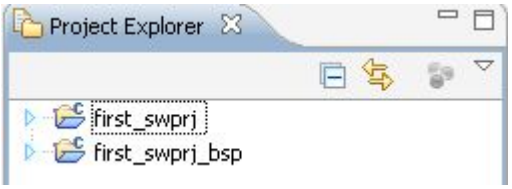


图 3.38 工程目录

七、设置软件编译属性

由于正常的工程模板软件 C 代码量比较大，而我们所分配的可用片内存储器（onchip_mem）容量也不大（12KBytes），因此需要在软件编译属性里做一些简单的设置，以裁剪代码量，否则编译将无法通过。

代码裁剪不是无依据的随便设置，在官方的 edh_ed_handbook.pdf→Section II→3.Debugging Nios II Designs→Reducing Code Size 一节有所描述。因此，根据文档中推荐的消减代码需要做如下设置。

HAL Setting	Value to Reduce BSP Library File Size
hal.max_file_descriptors	4
hal.enable_small_c_library	true
hal.sys_clk_timer	none
hal.timestamp_timer	none
hal.enable_exit	false
hal.enable_c_plus_plus	false
hal.enable_lightweight_device_driver_api	true
hal.enable_clean_exit	false
hal.enable_sim_optimize	false
hal.enable_reduced_device_drivers	true
hal.make.bsp_cflags_optimization	\"-Os\"

如图 3.39 所示，右键点击打开该工程的 BSP 编辑界面。

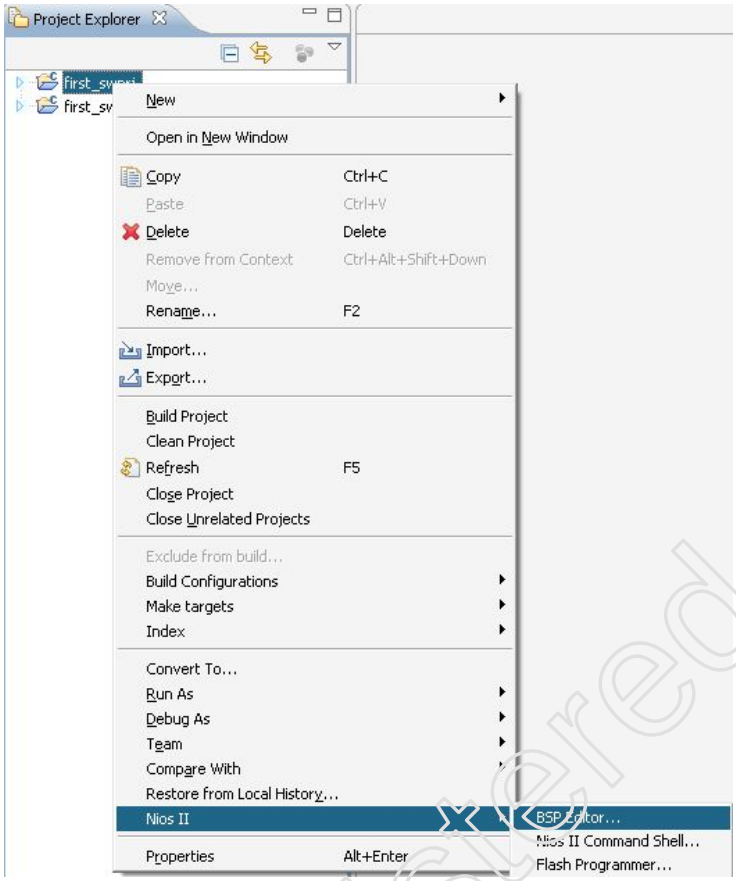


图 3.39 打开 BSP 编辑界面

然后弹出如图 3.40 所示的 BSP 编辑界面，在这个编辑界面中根据官方推荐进行设置。最后需要保存并点击右下角的“Generate”。完成后退出即可。

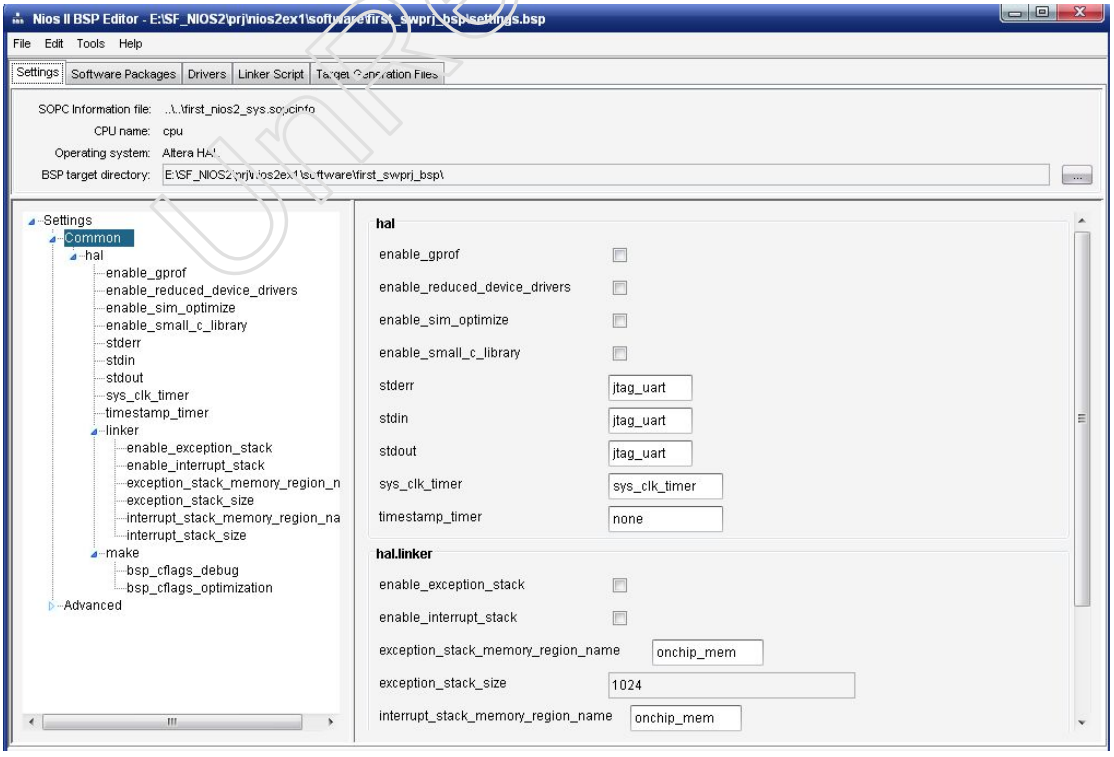


图 3.40 BSP 编辑界面

八、软件编译和下载

1. 如图 3.41 所示，右键点击 first_swprj 的 Build Project 选项进行软件工程编译。因为我们采用的是模板工程，因此软件程序都是编写好的，是可以直接驱动我们的 8 个 LED 外设进行跑马灯演示。感兴趣的朋友可以展开 first_swprj 的工程目录，打开 count_binary.c 进行查看。

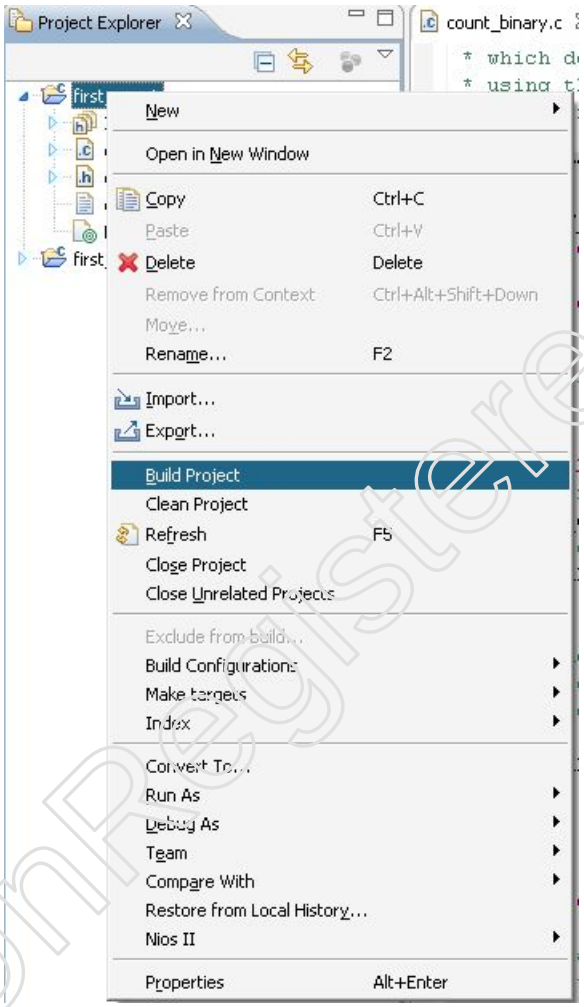


图 3.41 编译工程

2. 第一次编译时间稍微长一些。软件编译完成后，console 面板出现一些编译相关信息，其中有两行如下：

```
Info: (first_swprj.elf) 4452 Bytes program size (code + initialized data).
Info:                   7836 Bytes free for stack + heap.
```

说明编译的软件工程产生代码 4452 字节，12KB 总存储量中剩余 7836 字节可用。如果不做前面的代码消减优化，产生的代码量肯定要比这大几倍，甚至超出 12KB 使得编译无法通过。因为，默认情况下系统会将底层用户用到的不用到的所有相关驱动程序都编译进生成的代码里。优化的目的就是删除一些不使用的驱动程序从而减小代码量。

3. 如图 3.42 所示，右击 first_swprj 的 Run As→Nios II Hardware 选项。

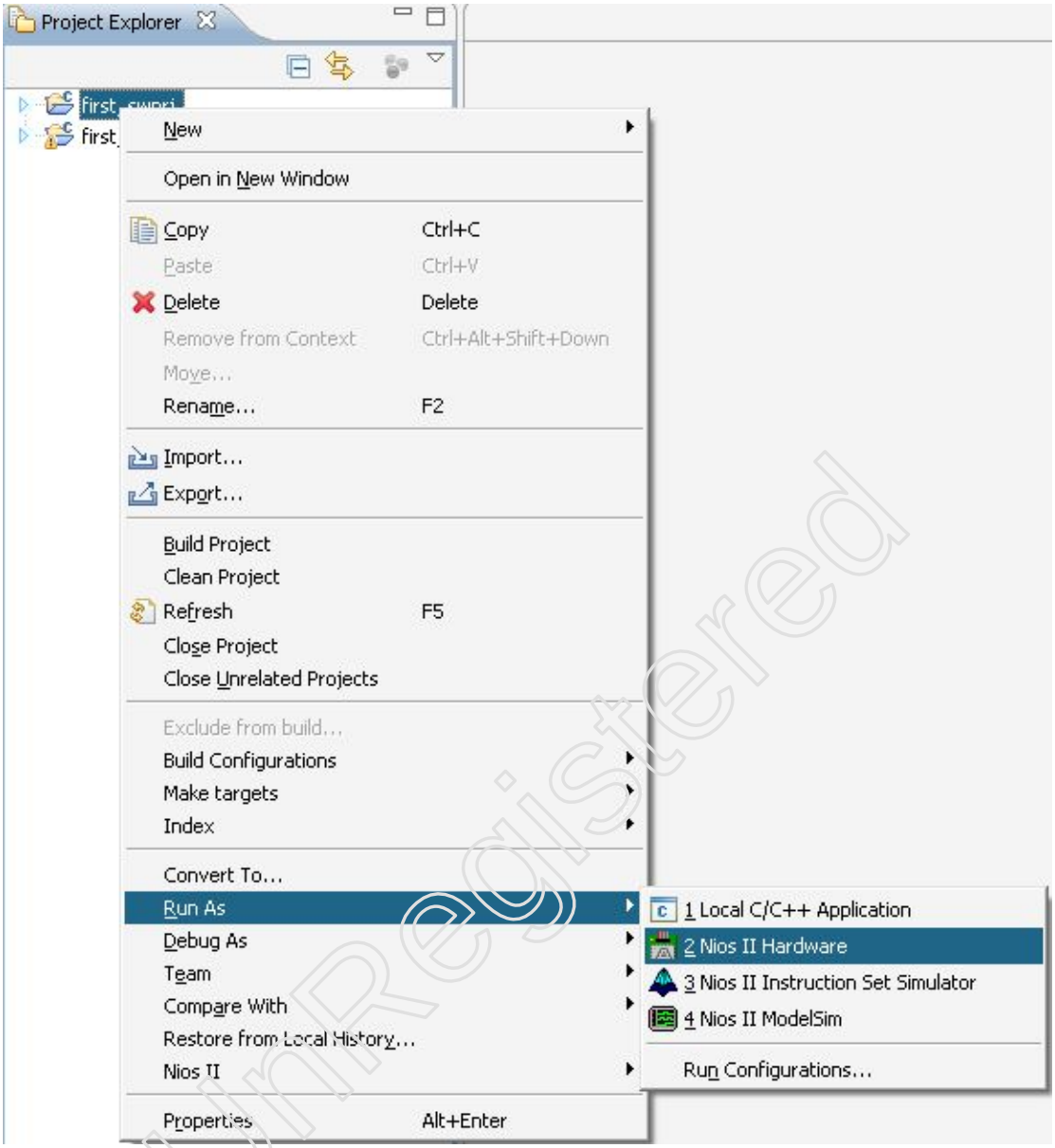


图 3.42 在目标板上运行软件

稍等片刻，此时一般会弹出如图 3.43 所示的运行配置窗口。在该窗口中提示下载线出现不匹配错误，因此需要用户进入 Target Connection 中刷新一下即可。

如图 3.44 所示，点击“Refresh Connections”，Cable 窗口下就出现了已经连接上的下载线名称，然后点击右下角的“Run”按钮即可。

片刻，我们可以看到 SF-NIOS2 FPGA 板上的 8 个 LED 闪烁的很“欢快”。

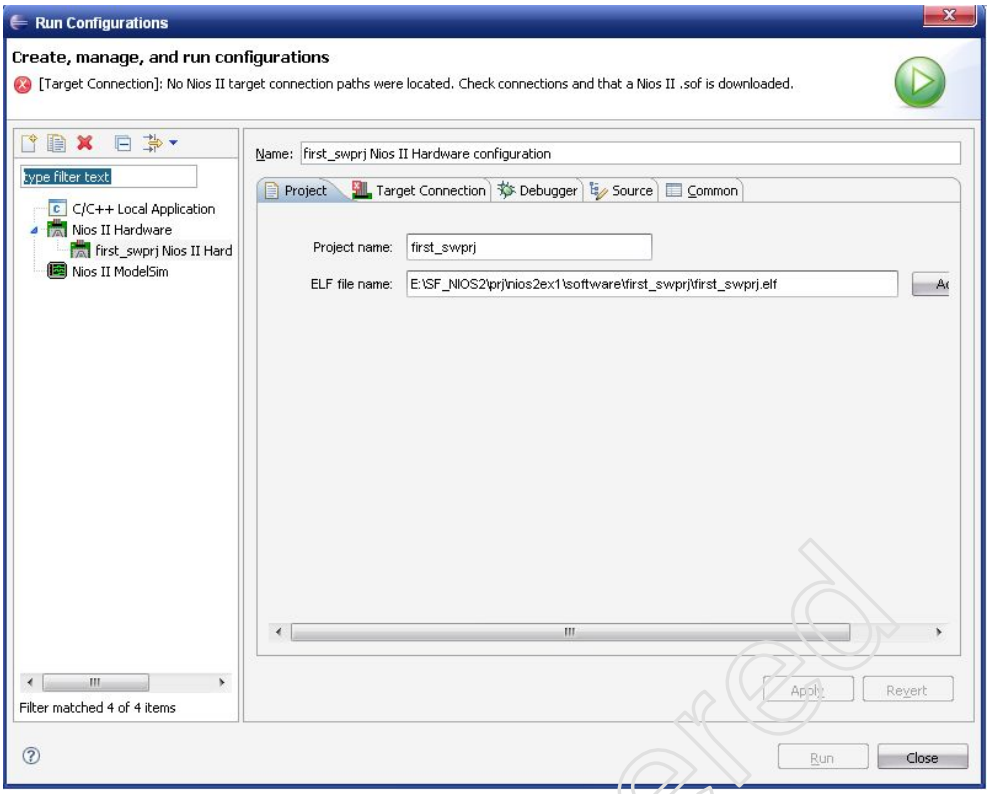


图 3.43 运行配置界面

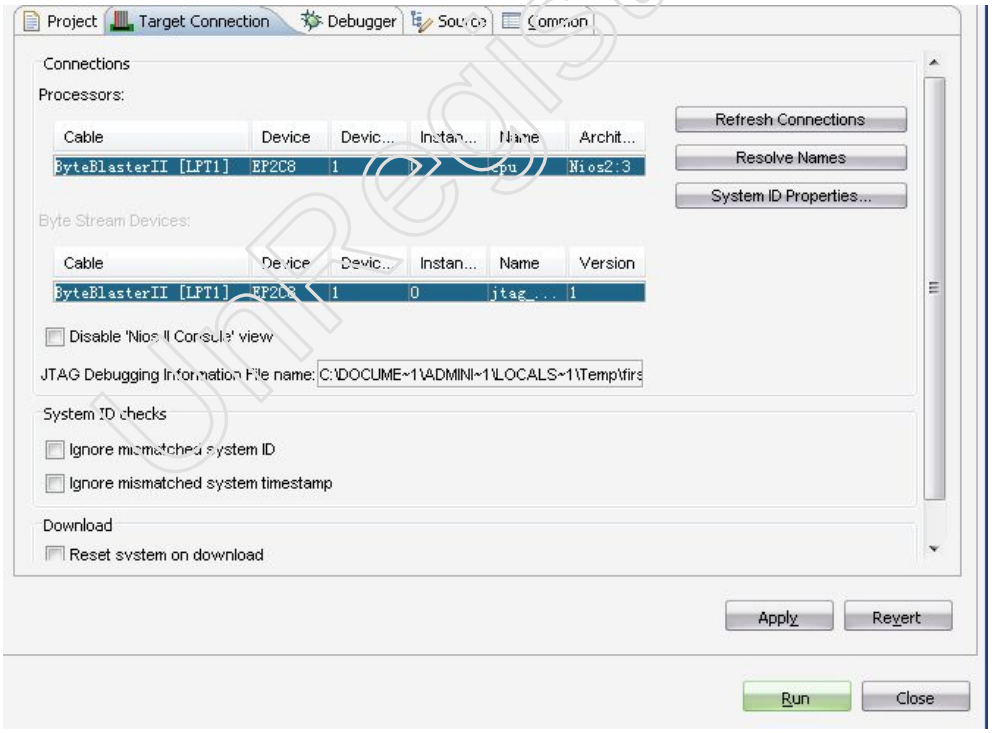


图 3.44 刷新下载线连接并运行

最后，恭喜您完成了第一个 NIOS2 工程的建立。希望第一个工程带您熟悉了 NIOS2 的开发流程。