

基于 DE2 的 uClinux 移植及应用开发

任务要求:

- 1、首先在 DE2 上运用 SOPC 工具搭建一个运行 uClinux 所需的硬件环境,并下载到 FPGA;
- 2、下载 uClinux 包 (<ftp://ftp.altera.com/outgoing/uClinux-dist-20080131.tar>), 并解压缩;
- 3、移植 uClinux 到 NIOS II, 此处需要在 linux 环境下进行编译, 可以到 216 机房借用进行, 或者个人电脑上都可以, 建议使用 Fedora 或者 redhat As4 及以上版本。并测试; 此处可参考: <http://www.eetop.cn/blog/html/62/208662-8087.html>

建议步骤:

1.1 Nios II 硬件定制

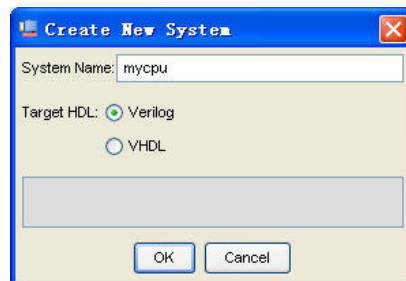
1.1.1 添加 SOPC 组件

首先, 在 Quartus II 新建工程 “NiosLinux”, 保存目录为 C:\uClinuxNios, 由于是采用 DE2 开发板, 器件选择 EP2C35F672C6, 其他保持默认。

在 Quartus II 中, 每一个 SOPC Builder 的系统对应一个 Quartus II 的工程, 可以用 Tools->SOPC Builder 菜单启动 SOPC Builder, 也可以用工具按钮启动 SOPC Builder。如果工程中没有 SOPC Builder 系统, Quartus II 会提示使用者输入 SOPC Builder 系统的名称, 并选择生成 VHDL 代码还是 Verilog HDL 代码。SOPC Builder 同时只能对一个系统进行操作, 但一个 Quartus II 工程可以包含多个系统, 在 SOPC Builder 中可以打开一个已有的系统, 也可以新建系统, 系统的文件与相关的 Quaturs II 工程存储在同一个目录中, 每个系统的信息都保存在一个名称为<系统模块名称>.ptf 的文件中。



(a) Quartus II 新建工程



(b) SOPC_Builder 新建工程

图 1 工程创建

在 SOPC Builder 中, 一个系统是由多个组件(component)组成的, 这些组件按照一定的规则设计, 能够被 SOPC Builder 识别并可以自动连接到系统中去。SOPC Builder 将多个组件连接在一起组成生成一个顶层设计模块称作“系统模块”(System Module)。活动组件列表区域中列出了当前系统中例化的组件, 在这个区域中, 可以修改组件的名称, 设定从组件的基地址、时钟源, 如果需要, 还可以设定每个组件的中断优先级, 所有外设系统中的管理都是按统一分配的地址进行的, 活动组件列表中可以设置每一个外设的基地址, 根据基地址和外设的特性, 自动生成结束地址。如果设计者不干预, SOPC Builder 则自动分配各个外设的地址。有效组件列表区域中, 分类列出了当前有效的组件, 包括 Altera 提供的以及自定义的组件库, 用户可以从这个列表中选择自己需要的组件添加到右侧的活动组件列表中, 建立用户自己的系统, 每个添加到系统中的组件都可能会启动一个配置向导, 指导用户配置这个

组件。连接面板区域显示了组件之间的连接，可以在这个区域中指定主组件与从组件之间的连接关系以及与多个主组件共享的从组件的总线仲裁器。时钟设置表用以设置系统的时钟。在做接下来一步前，为了用到 DE2 开发板上专门提供的硬件资源，如以太网控制器 DM9000A、音频编解码器 Audio_DAC_FIFO、VGA 控制器 Binary_VGA_Controller、USB OTG ISP1362、七段数码管控制器 SEG7_LUT_8 以及 SRAM 控制器 SRAM_16Bit_512K，需要把 DE2 光盘目录 demonstrations\SOPC_Builder\Component 上的所有内容，复制到硬盘上的 Altera 软件工具安装目录，默认的路径是 altera\72\ip\sopc_builder_ip。然后再来运行 SOPC Builder，可以在组件栏看到，以上组件全部被包含在 Terasic Technologies Inc 组，新建系统名字为“mycpu”，CPU 模块目标 HDL 文件为 Verilog 格式。

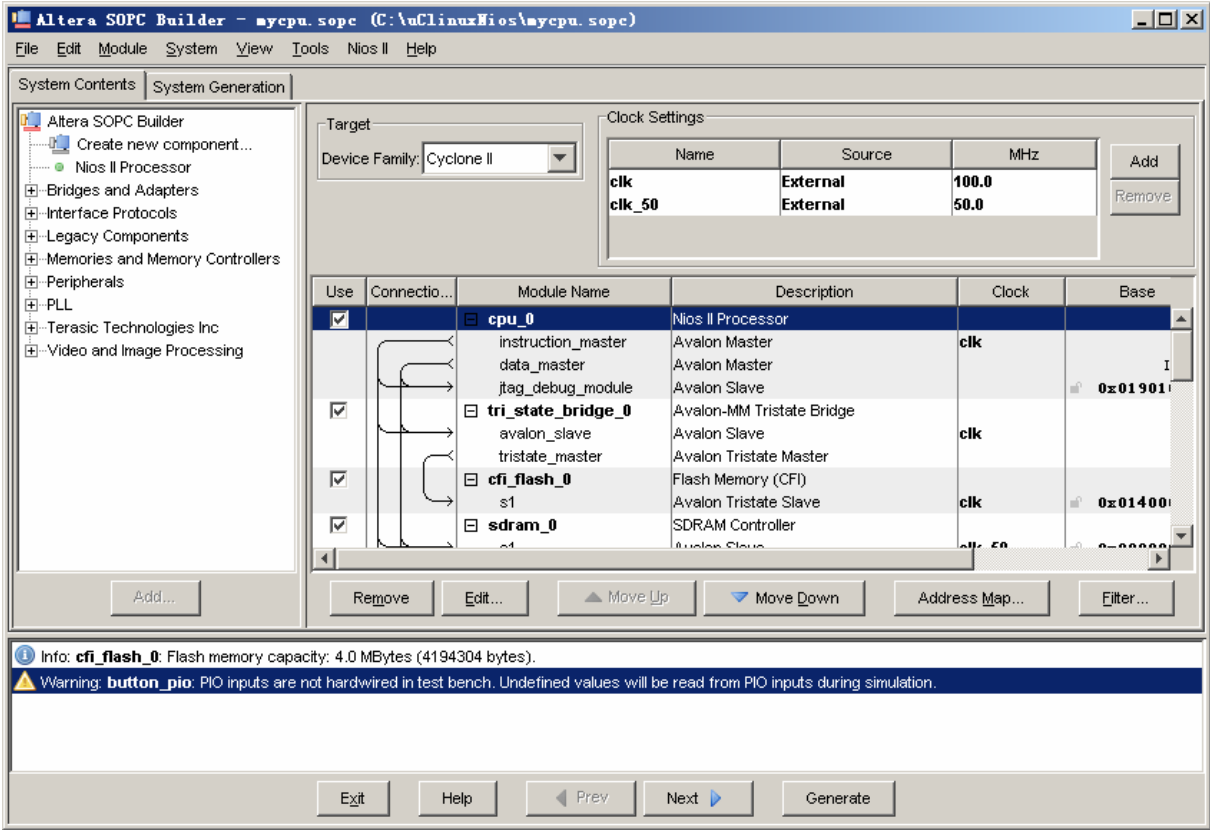


图 2 添加组件

如上图组建定制 Nios II 系统，用到的组件包括：
本文设计 uClinux 的硬件平台包括以下几个部分：cpu_0 (Nios II)、uart_0 串口组件、epcs_controller EPCS16 控制器、timer_0 定时器、Flash 存储器、sdram_0 存储器、jtag_uart_0 调试组件、led_green 绿 LED 灯、button_pio 4 位输入 PIO 和 DM9000 以太网控制器。这个平台就是后面移植 uClinux 的硬件基础。
下面是每个组件的具体添加过程，这里首先添加存储器，然后再添加 CPU 和其他组件：

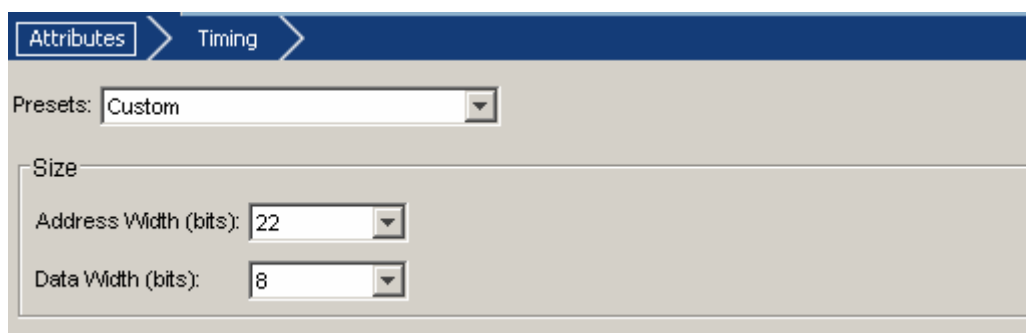


图 3 配置 Flash 存储器

● 添加 Flash Memory (CFI)

在 SOPC Builder 主界面左侧的组件列表中的 Memories and Memory Controllers 组中，点击 Flash，选中 Flash Memory (CFI)，在弹出菜单中，选择存储器地址宽度 22 位，数据位为 8 位，总内存尺寸 4M。改模块名称为 cfi_flash_0。

● 添加 EPCS Serial Flash Controller

在 SOPC Builder 主界面左侧的组件列表中的 Memories and Memory Controllers 组中，点击 Flash，选中 EPCS Serial Flash Controller，在弹出菜单中，直接单击 Finish 按钮完成默认配置，改模块名称为 epcs_controller。

● 添加 SDRAM Controller

在 SOPC Builder 主界面左侧的组件列表中的 Memories and Memory Controllers 组中，点击 SDRAM，选中 SDRAM Controller，在弹出菜单中，选择数据位为 16 位，总内存尺寸 8M。改模块名称为 sdram_0。

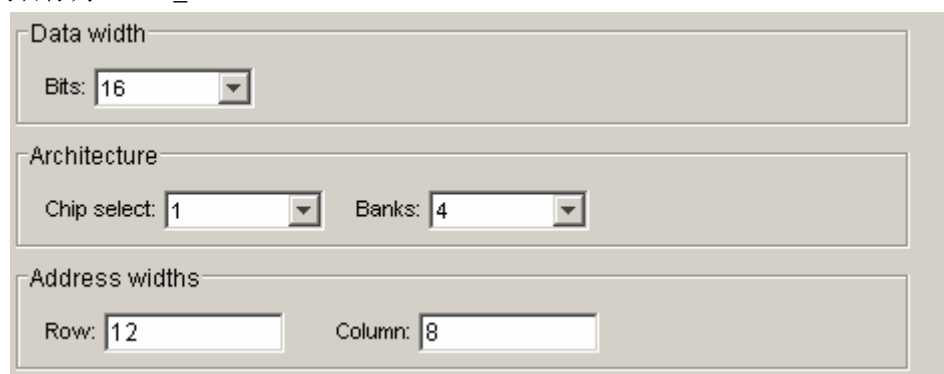


图 4 配置 Flash 存储器

● 向系统中添加 Nios II 处理器

选中 Nios II Processor，显示 Nios II 处理器配置界面。选择 Nios II/f 作为本设计的处理器，该模式最好性能可达 101MIPS，用以加强运行 uClinux 的系统性能和反应速度。

| | <input type="radio"/> Nios II/e | <input type="radio"/> Nios II/s | <input checked="" type="radio"/> Nios II/f |
|-------------------------------------------------------------------------------------------------------|---------------------------------|------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nios II Selector Guide Family: Cyclone II f_{system} : 100.0 MHz cpuid: 0 | RISC 32-bit | RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide | RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction |
| Performance at 100.0 MHz | Up to 9 DMIPS | Up to 50 DMIPS | Up to 101 DMIPS |
| Logic Usage | 600-700 LEs | 1200-1400 LEs | 1400-1800 LEs |

图 5 选择 Nios II 处理器

另外，在同一页面设置 Nios II 处理器的复位地址、异常地址及调试模块用的存储器类型和断点地址。

| | | | |
|-------------------|---------------------|--------------|------------|
| Reset Vector: | Memory: cfi_flash_0 | Offset: 0x0 | 0x01400000 |
| Exception Vector: | Memory: sdram_0 | Offset: 0x20 | 0x00800020 |

图 6 设置 Nios II 处理器的复位地址、异常地址及调试模块用的断点地址

然后，按 Next 按钮，设置处理器的指令缓存和紧密耦合指令存储器，选择指令缓存为 2K 字节，不使用紧密耦合指令存储器。

| Core Nios II | Caches and Memory Interfaces | Advanced Features | JTAG Debug Module | Custom Instruct |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-----------------|
| Caches and Memory Interfaces | | | | |
| Instruction Master Instruction Cache: 4 Kbytes <input type="checkbox"/> Enable Bursts (Burst Size: 32 bytes) Help <input type="checkbox"/> Include tightly coupled instruction master port(s). Number of ports: 1 | | Data Master Data Cache: 2 Kbytes <input type="checkbox"/> Omit data master port Data Cache Line Size: 4 Bytes <input type="checkbox"/> Enable Bursts Help <input type="checkbox"/> Include tightly coupled data master port(s). Number of ports: 1 | | |

图 7 配置 Nios II 处理器的指令缓存与紧密耦合存储器

按 Next 按钮，设置 JTAG 调试模块，JTAG 调试模块分为 4 个级别，每个级别的功能不同，占用的逻辑资源也不同，本设计选择占用逻辑资源最少的级别 Level 1。

| Select a debugging level: | | | | |
|-----------------------------------|---------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <input type="radio"/> No Debugger | <input checked="" type="radio"/> Level 1 | <input type="radio"/> Level 2 | <input type="radio"/> Level 3 | <input type="radio"/> Level 4 |
| | JTAG Target Connection Download Software Software Breakpoints | JTAG Target Connection Download Software Software Breakpoints 2 Hardware Breakpoints 2 Data Triggers | JTAG Target Connection Download Software Software Breakpoints 2 Hardware Breakpoints 2 Data Triggers Instruction Trace On-Chip Trace | JTAG Target Connection Download Software Software Breakpoints 4 Hardware Breakpoints 4 Data Triggers Instruction Trace Data Trace On-Chip Trace Off-Chip Trace |
| No LEs | 300-400 LEs | 800-900 LEs | 2400-2700 LEs | 3100-3700 LEs |
| No M4Ks | Two M4Ks | Two M4Ks | Four M4Ks | Four M4Ks |

图 8 设置 JTAG 调试模块

最后点击 Finish，完成 Nios II CPU 添加^[7]。

● 添加 Avalon-MM Tristate Bridge

在 SOPC Builder 主界面左侧的组件列表中的 Bridges and Adapters 组中，点击 Memory Mapped，选中 Avalon-MM Tristate Bridge，在弹出菜单选择 Registered，更改模块名为 tri_state_bridge_0。

● 添加 JTAG UART

在 SOPC Builder 主界面左侧的组件列表中的 Interface Protocols 组中，点击 Serial，选中 JTAG UART，在弹出菜单按照默认设置，配置不做改变，这样才能进行后面的在线命令窗口调试。

● 添加 DM9000 网络芯片

前面手动添加用户定义组件后，就可以在 SOPC Builder 的组件列表看到 Terasic Technologies Inc 组，选中其中的 DM9000A，经过一段时间的临时编译加载后，即可完成添加。

● 添加定时器

在 SOPC Builder 主界面左侧组件列表中的 Peripherals -> Microcontroller Peripherals 组中，选中 Interval Timer，按照默认设置，配置不做改变，生成一个初始周期为 1ms 的定时器。

● 自动设置基地址

至此已经添加了所有需要的组件，SOPC Builder 根据组件添加的顺序以及各组件需要的地址范围，自动为各组件分配了地址，可以手动修改这些地址，也可以自动设置地址。点击 System->Auto Assign Base Addresses 菜单，即可自动设置地址。

● 自动设置 IRQs

点击 System->Auto Assign IRQs 菜单，即可自动设置中断优先级，当然也可以手动修改。这里，只让计时器 timer 的优先级为 0，即优先级最高，其他组件的优先级让 SOPC Builder 自动分配。

● 生成系统

当完成组件添加，并且没有错误信息时，按 Generate 命令按钮开始生成 Nios II 系统。根据系统规模不同，生成系统所需的时间也不一样，在生成系统的过程中，屏幕上会提示相关的信息，生成系统的工作完成之后，显示 SYSTEM GENERATION COMPLETED。

● PTF 系统文件

到此，SOPC Builder 自动生成一个 PTF 文件。所有的设计信息都存储在该 PTF 文件里。PTF 不是字母缩写，不代表什么含义。用户可以在任何目录下创建任意多的 SOPC 系统，每一个系统都将生成一个唯一的 PTF 文件。当使用 SOPC Builder 重新打开一个已有的系统时，PTF

文件是 SOPC Builder 读取该系统具体设计信息的唯一来源。

系统 PTF 文件是图形用户界面和系统生成程序之间唯一的交互渠道。对大部分用户来说，他们仅仅了解 PTF 文件由图形用户界面产生，并且系统生成程序要读取 PTF 文件的内容就足够了。但高级用户可能希望手工编辑自己的设计，而不是使用 SOPC Builder 图形用户界面。

PTF 文件是一般可读文本，可以使用一个文本编辑器或其他任何生成文本文件的方式来创建它。PTF 文件的内容，一部分用于 SOPC Builder 图形用户界面（显示 IP 模块信息），一部分用于系统生成程序产生正确的互连逻辑（总线时序信息），还有一些是图形用户界面和系统生成程序共同使用的内容^[8]。

按 Exit 按钮可退出 SOPC Builder，切换到 Quartus II 软件继续后面的工作。

1.1.2 注意事项

1). 为了提高 CPU 的性能，Nios 类型选择 Fast(Nios II/f)，并带有 4k 或者 2K 的 Instruct cache 和 Data cache，复位应该有个延时控制，为此，需要在 HDL 顶层代码中增加一个 Reset_Delay (Reset_Delay.v) 实例 delay1；

2). 开发板运行 uClinux 至少要有 8M 的 SDRAM 容量，同时 SDRAM 的时钟与 Nios 的时钟要有适当相移，HDL 顶层代码应再补充一个锁相环 SDRAM_PLL (SDRAM_PLL.v) 实例 PLL1，以满足不同时钟要求。

例中：Nios II CPU 的时钟 CPU_CLK(100MHz)，PLL1 生成 SDRAM 的时钟 DRAM_CLK(3ns 移相后的 50MHz)

3). 在 Linux 当中，irq 0 意味着自动检测，除了计时器 timer，不能使用在任何器件。在选择完组件，点击自动分配地址和 IRQ 后，要回过头来检查

4). 无误后点击“Generate”，保存.sof 配置文件和生成.v 软核 verilog HDL 文件。

1.2 顶层文件

顶层文件采用 Verilog HDL 语言编写，在 Quartus II 用 HDL 语言编写顶层文件，比用原理图输入文件体积大大减少，而且条理清晰，方便修改组织管理，充分利用了当今电子技术计算机辅助设计的特点，可以参考 DE2 光盘上 DE2_NET 工程的顶层文件，并以默认的工程名字作为文件名，保存为 NiosLinux.v 文件。

在顶层文件中，添加了两个模块 Reset_Delay 和 SDRAM_PLL，分别用作 KEY[0]按键 CPU 手动复位和 SDRAM 的高速时钟 100MHz 分频输出。组件的输出信号引脚，与引脚锁定文件一一对应。

具体源码在附录 A。

1.3 编译系统

Quartus II 软件允许您在设计流程的每个阶段使用 Quartus II 图形用户界面、EDA 工具界面或命令行界面。可以在整个流程中只使用这些界面中的一个，也可以在设计流程的不同阶段使用不同的选项。

编译器包括以下模块（标有星号的模块表示在编译期间可选，具体视设置而定）：

- Analysis & Synthesis
- Fitter
- Assembler

- Timing Analyzer
- Design Assistant*
- EDA Netlist Writer*
- Compiler Database Interface*

可以在全编译过程中通过选择菜单 **Processing -> Start Compilation** 来运行所有的编译器模块。若要单独运行各个模块，可以通过选择 **Processing -> Start**，然后从 **Start** 子菜单中为模块选择相应的指令。

此外，还可以通过选择 **Tools -> Compiler Tool**，并在 **Compiler Tool** 窗口中运行该模块来启动编译器模块。在 **Compiler Tool** 窗口中，可以打开该模块的设置文件或报告文件，还可以打开其它相关窗口。

Quartus II 编译一个工程，主要步骤和完成的工作^[8]，可以参照下图。

- 使用 **Analysis & Synthesis** 对设计进行综合。
- 使用 **Fitter** 对设计执行布局布线。在对源代码进行少量更改之后，还可以使用增量布局布线。
- 使用 **Timing Analyzer** 对设计进行时序分析。
- 使用 **Assembler** 为设计建立编程文件。

编译前，需要为设计分配引脚，可以在引脚规划器 **Pin Planner** 中手动分配引脚，也可以导入 DE2 系统光盘中提供的引脚分配表。在 DE2 光盘中有个叫做 **DE2_pin_assignments.csv** 的文件，文件中包含了 DE2 平台上 **FPGA** 的引脚分配信息。为方便起见，我们从 **DE2_pin_assignments.csv** 中导入引脚分配信息。注意要使用导入的引脚分配，则顶层原理图设计中的引脚命名应该与引脚锁定文件里表示的名称完全一致。

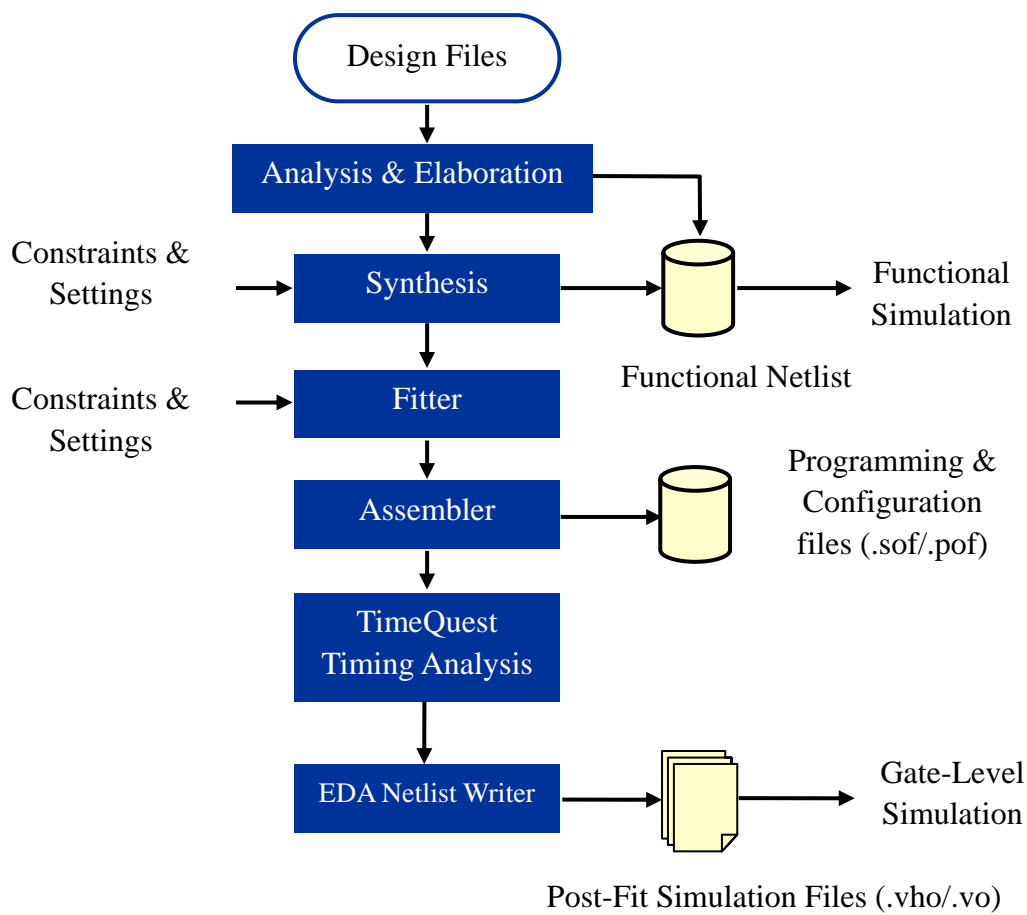


图 9 Quartus II 编译流程图

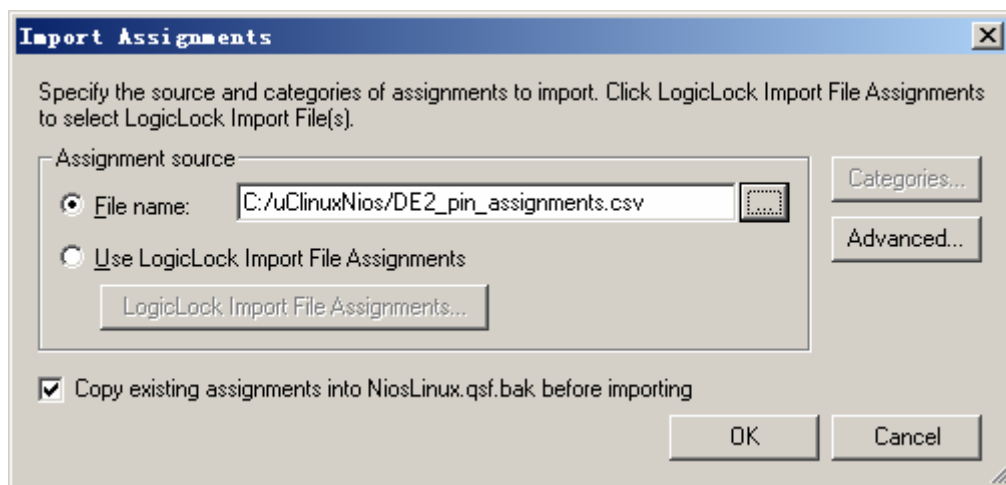


图 10 为工程添加引脚锁定文件添加

在 Quartus II 中，打开“NiosLinux”工程，点击菜单栏 Assignments -> Import Assignments，添加 DE2 开发板引脚锁定文件 DE2_pin_assignments.csv，最后编译“NiosLinux”工程，得到.sof（系统文件）^[11]。完成编译后，从返回的结果可以看出这个工程在 FPGA 上所占用的资源。将编译后的.pof 文件下载到 DE2 开发板上的 FPGA 中，就完成了硬件设计部分。

1.4 Git 的获取与安装

在 **Fedora Core 8** 下，用 `yum install git-core` 命令，就可以很方便地获取并安装 `git-core` 程序包，又或者到 <http://git.or.cz/> 下载最新的二进制源码包，本地编译安装。

然后，注册帐号和邮箱地址，命令如下

```
git config --global user.email "you@email.com"
```

```
git config --global user.name "Your Name"
```

到这里，我们转入正题。

1.5 获取 uClinux 源码

这里直接从下面的地址下载。

<http://ftp.altera.com/outgoing/uClinux-dist-20080131.tar>

下载回来的源码解压后，进入 `uClinux-dist` 目录，这时，看不到任何文件，原因是没有对分发源码解包。输入 `git branch -a` 命令查看有哪些分发版本，如图所示，接着用 `git checkout` 命令释放所需要的开发源码版本，这里提取 `nios2` 分支源码，命令是 `git checkout nios2`，这是一个稳定版本（`uClinux-dist-20080131 stable, kernel v2.6.23`）。提取过程，和一般文件解压缩有点类似，完成后，就可以看到文件了。

```
[root@localhost ~]# cd /mnt/sdb/uClinux-dist
[root@localhost uClinux-dist]# git branch -a
* master
* nios2
  v2.6.23-uc
  origin/HEAD
  origin/fixme
  origin/master
  origin/nios2
  origin/test
  origin/v2.6.23-uc
[root@localhost uClinux-dist]# git checkout nios2_
```

图 11 查看源代码分发版本

1.6 内核剪裁和编译

Fedora Core 8 默认使用的编译器是 GCC 4.2，用其执行 Nios II 交叉编译过程，在编译 `binutils` 时候会出错，所以需要使用 GCC 3，也就是要下载专门的 `BinaryToolchain —nios2gcc`，代码文件编译命令是 `nios2-linux-uclibc-gcc`，所使用的 `gcc` 版本是 3.4.6。

```
[root@localhost ~]# cat ~/.bash_profile
# ~/.bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH="$PATH":/opt/nios2/bin

export PATH
unset USERNAME
[root@localhost ~]# _
```

图 12 设置用户环境变量

设置用户环境变量，首先，用 `vi` 命令打开 `~/.bash_profile` 文件，在当中的 `PATH` 变量添加

/opt/nios2/bin, 再按 Esc 键, 输入 qw 命令, 保存并退出。上面所示是.bash_profile 文件修改后的输出信息。

在做内核剪裁编译工作之前, 这一步非常重要, 在 Linux 的环境变量直接配置在用户账号信息里。确保注销重新登陆帐户后, 输入 nios2-linux-gcc -v 命令, 会显示以下图示信息, 说明编译器已经安装配置好, 并找到正确的链接。

```
[root@localhost nios2]# nios2-linux-gcc -v
Reading specs from /opt/nios2/lib/gcc/nios2-linux-uclibc/3.4.6/specs
Configured with: /root/buildroot/toolchain_build_nios2/gcc-3.4.6/configure --pre
fix=/opt/nios2 --build=i386-pc-linux-gnu --host=i386-pc-linux-gnu --target=nios2
-linux-uclibc --enable-languages=c,c++ --disable-__cxa_atexit --enable-target-op
tspace --with-gnu-ld --disable-shared --disable-nls --enable-threads --enable-mu
ltilib
Thread model: posix
gcc version 3.4.6
[root@localhost nios2]# _
```

图 13 检查编译工具是否可用

来到 uClinux-dist 目录, 执行 make menuconfig 命令, 进入内核配置图形菜单:

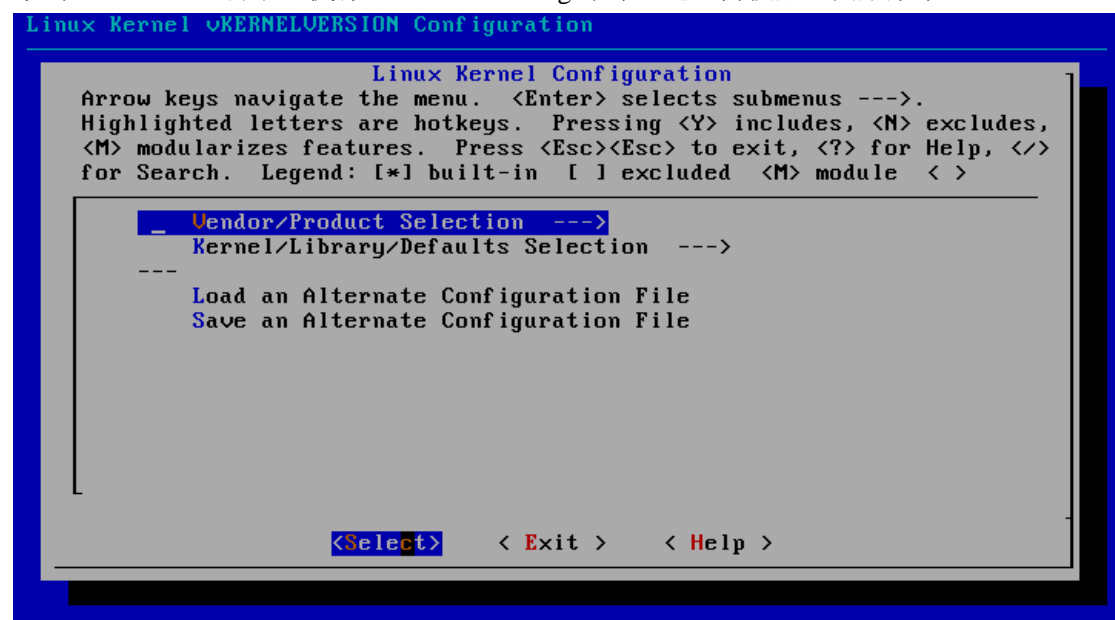


图 14 uClinux 内核配置图形菜单

按照下面选项进行定制

进入菜单

Vendor/Product Selection --->

--- Select the Vendor you wish to target

(Altera) Vendor # 选择厂商 Altera

--- Select the Product you wish to target

(nios2nommu) Altera Products # 自动选上 nios2

进入菜单

Kernel/Library/Defaults Selection --->

(linux-2.6.x) Kernel Version # 选择 Linux2.6 内核版本

(None) Libc Version # 不使用 uClinux 源码库

[*] Default all settings (lose changes) # 选上默认设置

[] Customize Kernel Settings # 首次编译不选上, 之后的自定义内核要选上

[] Customize Vendor/User Settings

[] Update Default Vendor Settings

按照以上设置后，输入以下命令

```
make vendor_hwselect SYSPTF=/mnt/hgfs/NiosDev1/system_0.ptf
```

联系硬件平台编译，这里用上一步生成的 ptf 文件

```
make
```

首次编译内核，需要花费一些时间

完成后，再次 make menuconfig

把 Kernel/Library/Defaults Selection --->[*] Customize Kernel Settings 选上，

然后 <exit> <exit> <yes> # 保存配置并退出

内核自定义菜单，自动运行

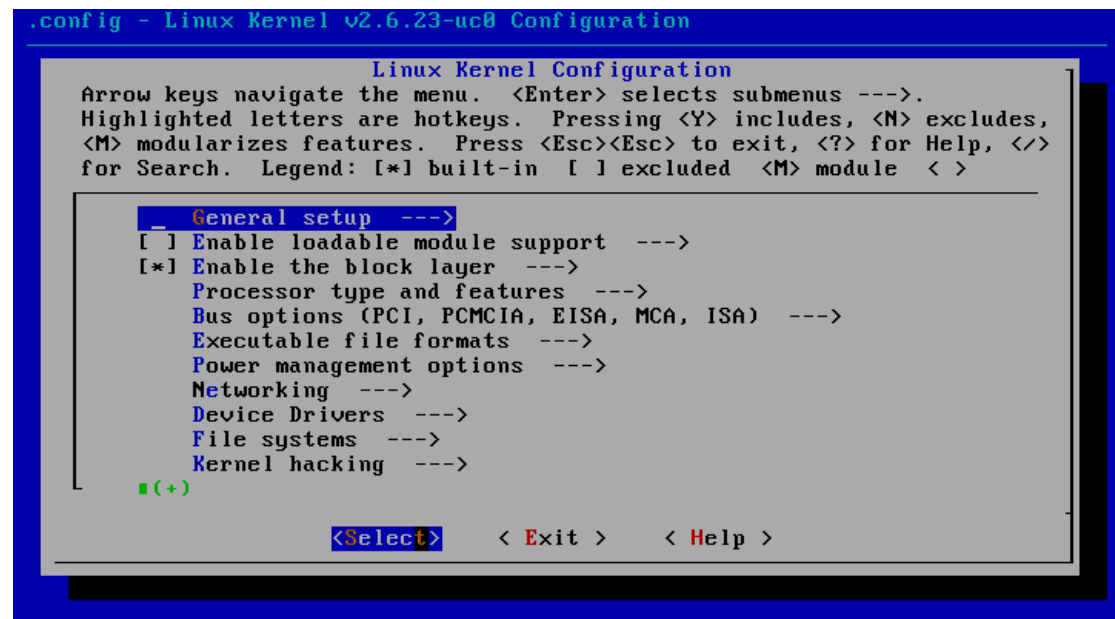


图 15 uClinux 2.6x 内核配置图形菜单

为了打开 DE2 开发板上的 DM9000A 网络芯片支持，要在内核配置菜单作以下选择：

进入菜单

Device Drivers -->Network device support -->

[*] Network device support # 开启网络设备支持

[*] Ethernet (10 or 100Mbit) # 使用网卡

进入菜单

[] SMC 91C9x/91C1xxx support

[] Opencores (Igor) Emac support

[] MoreThanIP 10_100_1000 Emac support

[*] DM9000 support # 开启 DE2 板载 DM9000A 芯片支持

[] DM9000A with checksum offloading # 修改后的 DM9000A 驱动，不选上

开启基于 nios2-terminal 终端的 DE2 开发板 JTAG UART (USB Blaster)端口调试支持

进入菜单

Device Drivers ---> Character devices ---> Serial drivers --->

[] Nios serial support

[*] Altera JTAG UART support

[*] Support for console on Altera JTAG UART

然后 <exit> <exit> <exit> # 自动保存配置并退出

make

再次编译内核

1.7 映像生成

成功完成以上剪裁和编译步骤后，就可以到 uClinux-dist dir 的 images 子目录获得 zImage 系统映像。为了接下来步骤的方便，把 zImage 映像复制到工程目录/mnt/hgfs/NiosDev1，即 Windows 下的 C:\uClinuxNios 目录。到此，内核定制完成，下一步是下载到 FPGA 芯片运行。

1.8 镜像下载

Nios 开发包中的 GNUPro 软件工具提供几个通用目的软件开发命令，包括 Nios SDK Shell 命令行。Nios SDK Shell 通过在 PC 平台上提供一个类 UNIX bash 环境来进行软件开发。熟悉 UNIX 使用起来会更方便一些，当中还包括程序生成和调试的专用命令^[12]。

在 Nios II EDS 的安装目录下，找到并运行 Nios II SDK Shell 程序。

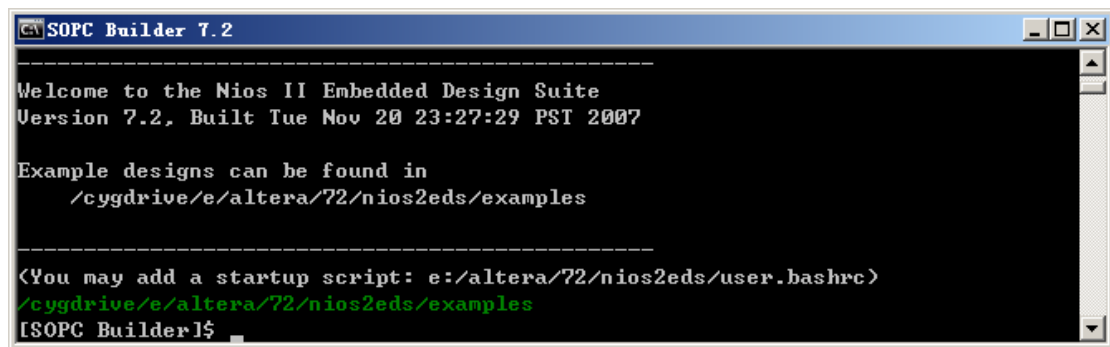


图 16 Nios II SDK Shell 启动等待界面

首先用命令 `cd /cygdrive/c/uClinuxNios` 进入工程所在文件夹，shell 调用的是 Cygwin 环境（Windows 平台下的 Linux 模拟环境，C 盘对应于/cygdrive/c）。

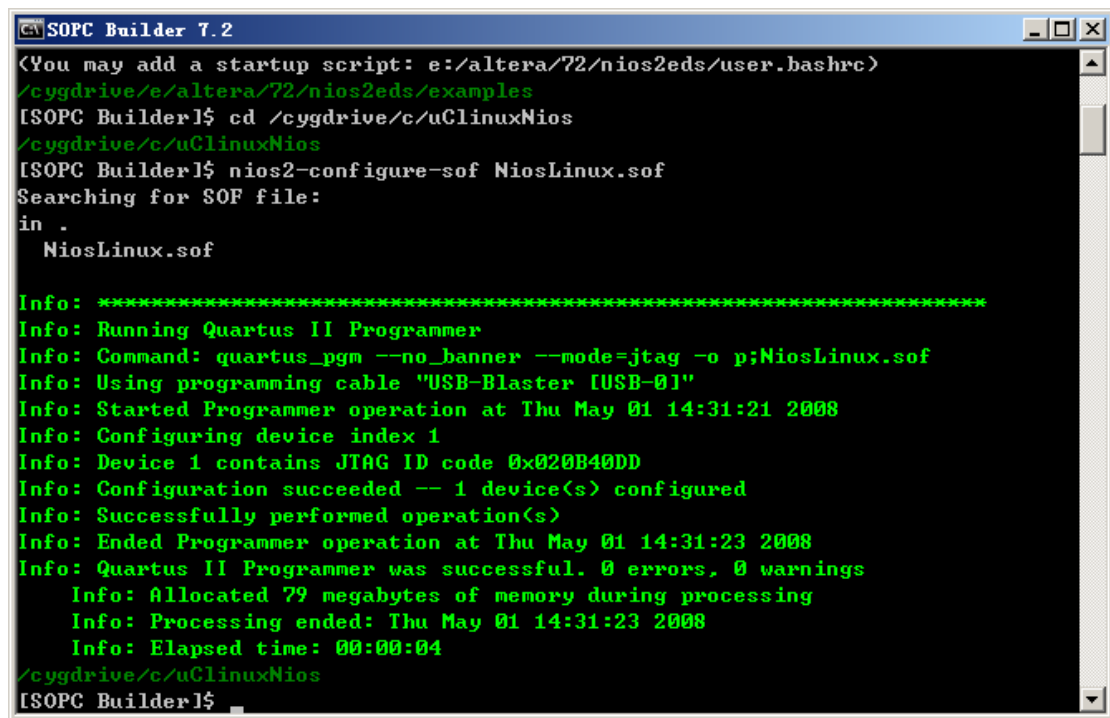
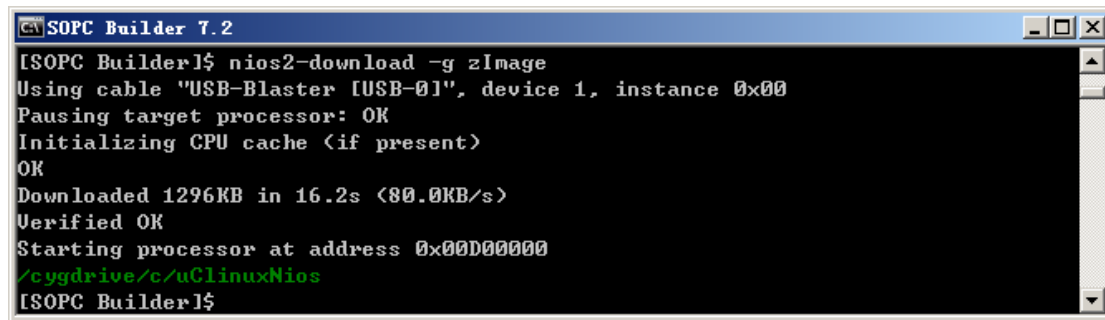


图 17 下载 sof 配置文件

然后用命令 `nios2-configure-sof NiosLinux.sof` 和 `nios2-download -g zImage`，分别下载 sof 配置文件(上图)和 zImage 镜像文件到 DE2 开发板(下图)

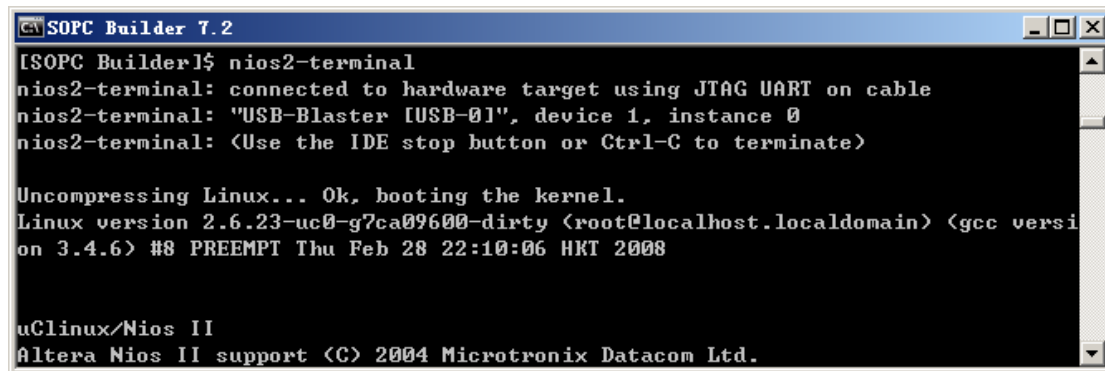


```
[SOPC Builder]$ nios2-download -g zImage
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Pausing target processor: OK
Initializing CPU cache <if present>
OK
Downloaded 1296KB in 16.2s <80.0KB/s>
Verified OK
Starting processor at address 0x00D00000
/cygdrive/c/uClinuxNios
[SOPC Builder]$
```

图 18 下载 zImage 镜像文件文件

1.9 终端调试

在 shell 输入 `nios2-terminal` 命令，进入终端调试模式，启动 uClinux 系统。

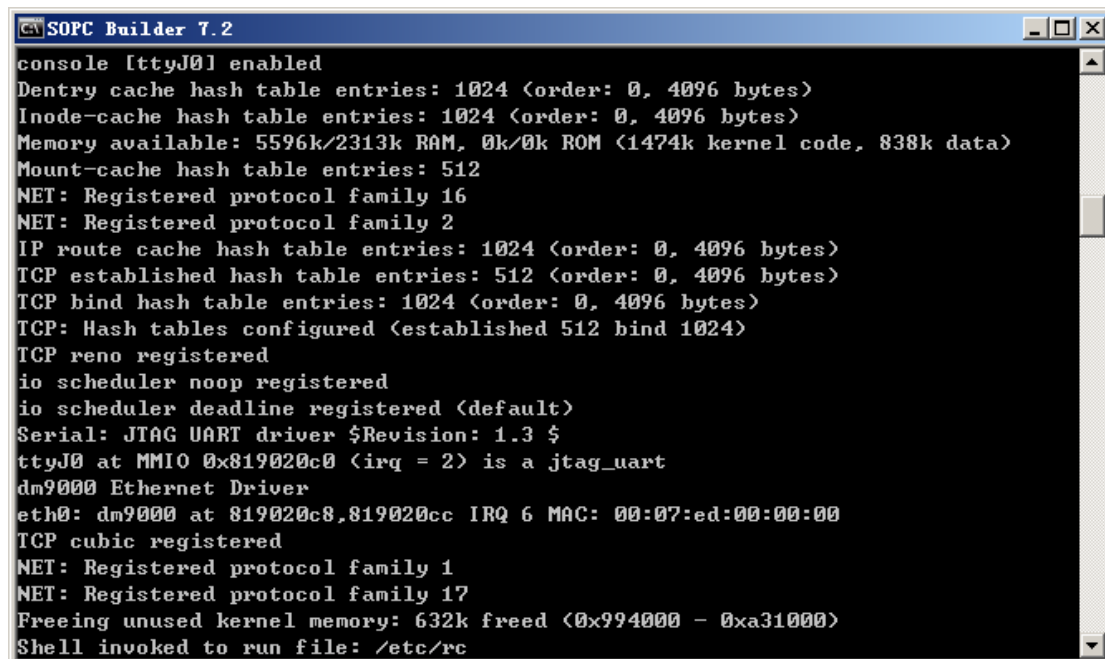


```
[SOPC Builder]$ nios2-terminal
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: <Use the IDE stop button or Ctrl-C to terminate>

Uncompressing Linux... Ok, booting the kernel.
Linux version 2.6.23-uc0-g7ca09600-dirty <root@localhost.localdomain> (gcc versi
on 3.4.6) #8 PREEMPT Thu Feb 28 22:10:06 HKT 2008

uClinux/Nios II
Altera Nios II support <C> 2004 Microtronix Datacom Ltd.
```

图 19 进入调试运行



```
[SOPC Builder]$ 
console [ttyJ0] enabled
Dentry cache hash table entries: 1024 <order: 0, 4096 bytes>
Inode-cache hash table entries: 1024 <order: 0, 4096 bytes>
Memory available: 5596k/2313k RAM, 0k/0k ROM <1474k kernel code, 838k data>
Mount-cache hash table entries: 512
NET: Registered protocol family 16
NET: Registered protocol family 2
IP route cache hash table entries: 1024 <order: 0, 4096 bytes>
TCP established hash table entries: 512 <order: 0, 4096 bytes>
TCP bind hash table entries: 1024 <order: 0, 4096 bytes>
TCP: Hash tables configured <established 512 bind 1024>
TCP reno registered
io scheduler noop registered
io scheduler deadline registered <default>
Serial: JTAG UART driver $Revision: 1.3 $
ttyJ0 at MMIO 0x819020c0 <irq = 2> is a jtag_uart
dm9000 Ethernet Driver
eth0: dm9000 at 819020c8,819020cc IRQ 6 MAC: 00:07:ed:00:00:00
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
Freeing unused kernel memory: 632k freed <0x994000 - 0xa31000>
Shell invoked to run file: /etc/rc
```

图 20 初始化设备

上图显示的是相关的设备模块注册信息，可以看到系统对 eth0(dm9000)网络设备作初始化。

系统加载完毕，等待输入操作命令

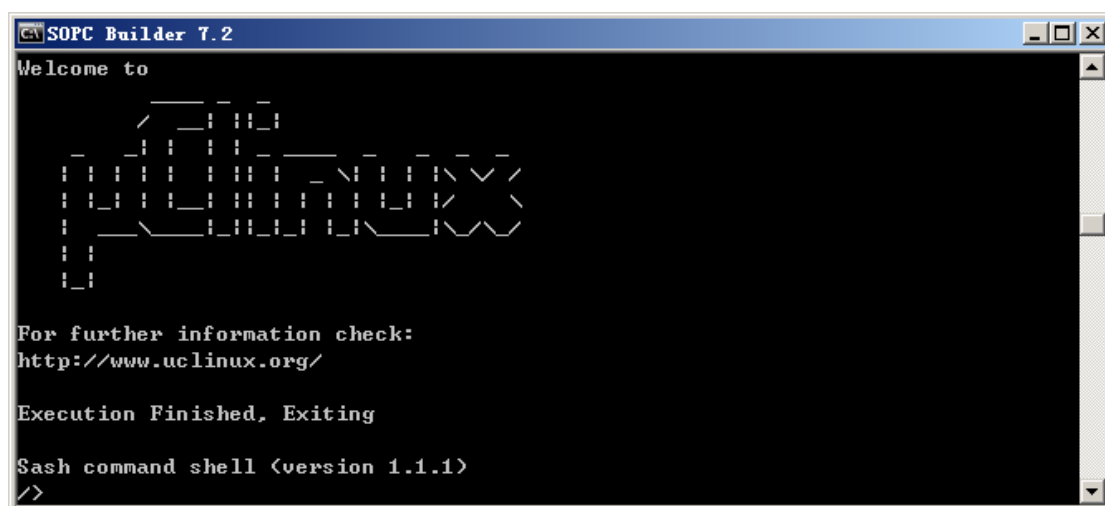


图 21 系统初始完成

1.10 执行 TCP/IP 网络命令

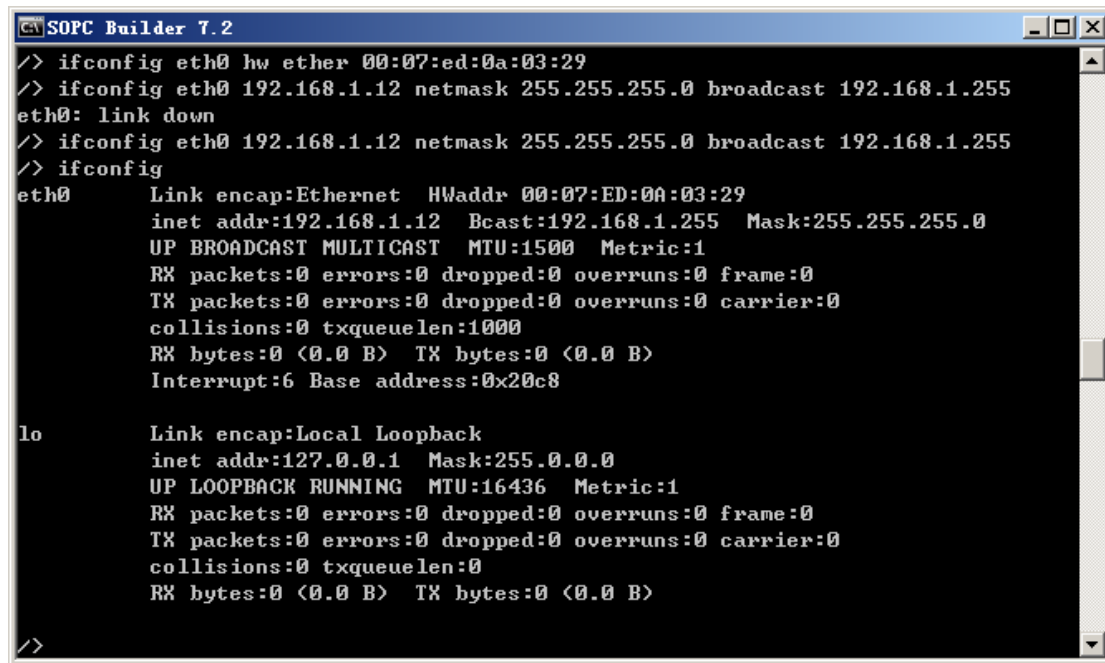
这里我们使用最简单的 ping 命令。

ping 的用途就是用来检测网络的连通情况和分析网络速度,通过一些参数和返回信息来显示判断是否连通,以及网络状况。

ping 的原理就是首先建立通道,然后发送一个 ICMP echo 请求的数据包,这个数据包会要求对方回应,对方接受后返回信息,这个包至少包括以下内容,发送的时候,送回 CMP echo reply 这样一个数据包来响应 ICMP echo 请求的数据包,包的内容包括对方的 ip 地址和自己的地址,还有序列数,回送的时候包括双方地址,还有时间等,主要是接受方在都是在操作系统内核里做好的,时刻在监听。

首先对系统进行网络设置,执行 MAC 地址修改和 IP 指定,如图,指定 DE2 的 IP 地址是 192.168.1.12,与 PC 设为同一子网内,下面是在 PC 上运行“Ping 192.168.1.2”后,执行的具体工作。

Ping 命令先构建一个固定格式的 ICMP 请求数据包,然后由 ICMP 协议将这个数据包连同地址“192.168.1.12”一起交给 IP 层协议(和 ICMP 一样,实际上是一组后台运行的进程),IP 层协议将以地址“192.168.1.12”作为目的地址,本机 IP 地址作为源地址,加上一些其他的控制信息,构建一个 IP 数据包,并在一个映射表中查找出 IP 地址 192.168.1.2 所对应的物理地址(即是 MAC 地址,这是数据链路层协议构建数据链路层的传输单元——帧所必需的),一并交给数据链路层。后者构建一个数据帧,目的地址是 IP 层传过来的物理地址,源地址则是本机的物理地址,还要附加上一些控制信息,依据以太网的介质访问规则,将它们传送出去。



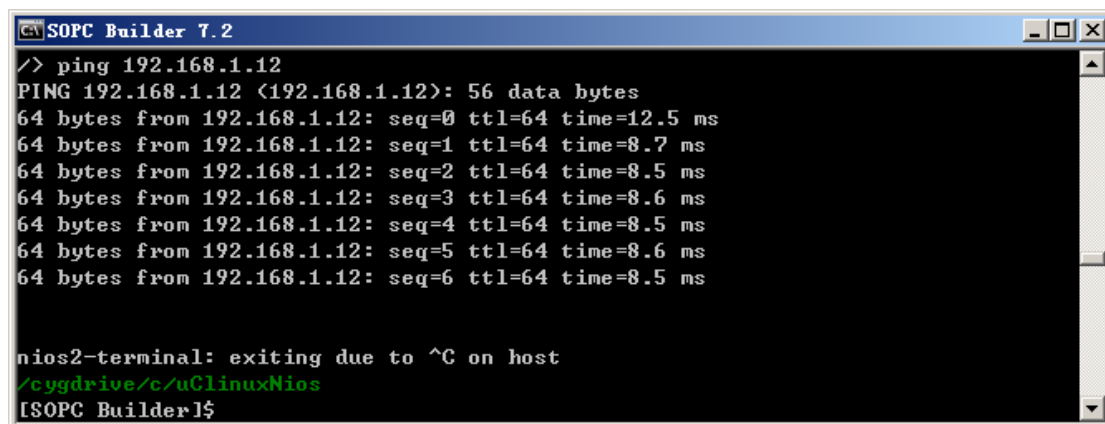
```
SOPC Builder 7.2
/> ifconfig eth0 hw ether 00:07:ed:0a:03:29
/> ifconfig eth0 192.168.1.12 netmask 255.255.255.0 broadcast 192.168.1.255
eth0: link down
/> ifconfig eth0 192.168.1.12 netmask 255.255.255.0 broadcast 192.168.1.255
/> ifconfig
eth0      Link encap:Ethernet  HWaddr 00:07:ED:0A:03:29
          inet addr:192.168.1.12  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:6 Base address:0x20c8

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/>
```

图 22 设置网络参数

DE2 收到这个数据帧后，先检查它的目的地址，并和本机的物理地址对比，如符合，则接收；否则丢弃。接收后检查该数据帧，将 IP 数据包从帧中提取出来，交给本机的 IP 层协议。同样，IP 层检查后，将有用的信息提取后交给 ICMP 协议，后者处理后，马上构建一个 ICMP 应答包，发送给 PC，其过程和 PC 发送 ICMP 请求包到 DE2 一样。



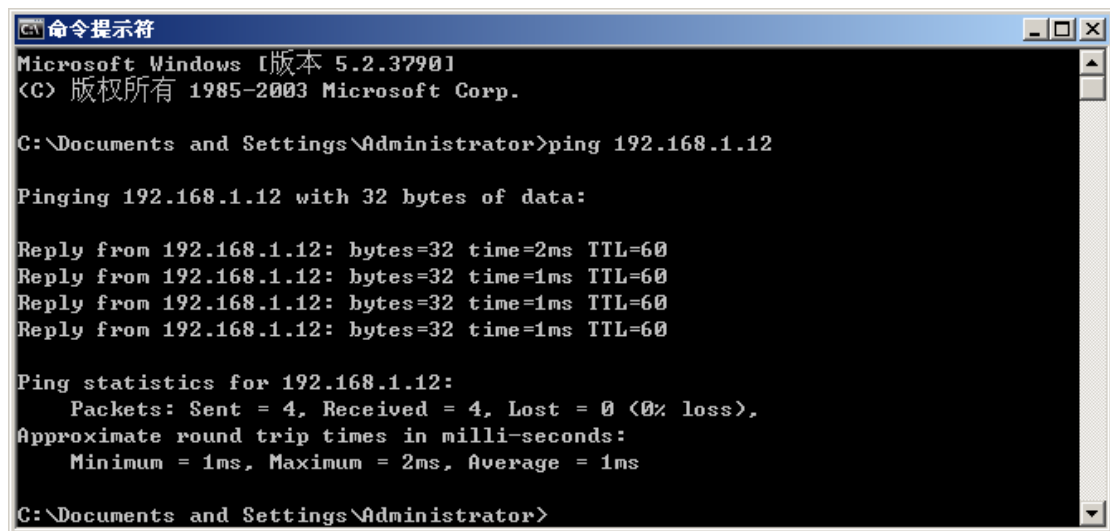
```
SOPC Builder 7.2
/> ping 192.168.1.12
PING 192.168.1.12 (192.168.1.12): 56 data bytes
64 bytes from 192.168.1.12: seq=0 ttl=64 time=12.5 ms
64 bytes from 192.168.1.12: seq=1 ttl=64 time=8.7 ms
64 bytes from 192.168.1.12: seq=2 ttl=64 time=8.5 ms
64 bytes from 192.168.1.12: seq=3 ttl=64 time=8.6 ms
64 bytes from 192.168.1.12: seq=4 ttl=64 time=8.5 ms
64 bytes from 192.168.1.12: seq=5 ttl=64 time=8.6 ms
64 bytes from 192.168.1.12: seq=6 ttl=64 time=8.5 ms

nios2-terminal: exiting due to ^C on host
/cygdrive/c/uClinuxNios
[SOPC Builder]$
```

图 23 检测自身网络

上图为板上的自行 ping 操作演示

下图为 Windows 下对 DE2 进行 ping 操作信息



```
命令提示符
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>ping 192.168.1.12

Pinging 192.168.1.12 with 32 bytes of data:

Reply from 192.168.1.12: bytes=32 time=2ms TTL=60
Reply from 192.168.1.12: bytes=32 time=1ms TTL=60
Reply from 192.168.1.12: bytes=32 time=1ms TTL=60
Reply from 192.168.1.12: bytes=32 time=1ms TTL=60

Ping statistics for 192.168.1.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\Documents and Settings\Administrator>
```

图 24 尝试 PC 连接开发板

通过简单网络命令的操作，我们可以看到，PC 与开发板之间可以互联互通，达到了 TCP/IP 通信的基本要求。到此，毕业设计的题目要求已经达到。至于，实现一些高级功能，例如 FTP 文件传送，Http 应用等，需要另外加载用户程序，在这就不详细列举。