Proteus 仿真 ARM7 实验手册

作者：陈拓
chentuo@ms.xab.ac.cn
2007 年 6 月 27 日 最后修改日期：2008 年 8 月 22 日

摘要

仿真软件 Proteus 是英国 Labcenter electronics 公司的 EDA 工具软件，Proteus 已有十五年的历史，在全球广泛使用，除了其具有和其它 EDA 工具一样的原理布图、PCB 自动或人工布线及电路仿真的功能外，其革命性的功能是，他的电路仿真是互动的，针对微处理器的应用，可以直接在基于原理图的虚拟原型上编程，并实现软件源码级的实时调试，如有显示及输出，还能看到运行后输入输出的效果，配合系统配置的虚拟仪器如示波器、逻辑分析仪等，您不需要别的，Proteus 为您建立了完备的电子设计开发环境！Proteus 产品系列也包含了革命性的 VSM 技术，用户可以对基于微控制器的设计连同所有的周围电子器件一起仿真。Proteus 可以仿真 8051、ARM、AVR、PIC 单片机，不愧为一款非常优秀的嵌入式仿真软件。简而言之，proteus 是个很好的东西，几乎没有他不能干的！

Proteus 嵌入式系统仿真与设计平台使你真正在 PC 上就可完成从原理图设计、电路仿真、PCB 设计到软件代码调试、实时仿真、测试和验证的整个开发过程。

Proteus 可以和 Keil uVision 配合使用，在 Proteus 里面画原理图，在 Keil 里面编写 C 语言程序，协同仿真调试。

名词

- SPICE 是一种用于电路描述与仿真的语言与仿真器软件，英文全称 Simulation Program with Integrated Circuits Emphasis。用于检测电路的连接和功能的完整性，以及用于预测电路的行为。主要用于模拟电路和混合信号电路的仿真。SPICE 是 1975 年由加里福尼亚大学伯克利分校的 Donald Pederson 在电子研究实验室首先建立的。第一版和第二版都是用 Fortran 语言编写的，从第三版开始用 C 语言编写。
- EDA 是电子设计自动化（Electronic Design Automation）的缩写。
- Proteus VSM 虚拟系统模型(Proteus Virtual System Modelling (VSM))组合了混合模式的 SPICE 电路仿真、动态器件和微控制器模型，实现了完整的基于微控制器设计的协同仿真，第一次真正使在物理原型出来之前对这类设计的开发和测试成为可能。

预备知识

本文的读者应该具有：

- 模拟电路和数字电路基础。
- 嵌入式开发概念。
- 嵌入式操作系统概念。
- 实时式操作系统概念。
- ARM 技术基础。

准备工作

先安装 Proteus 7.12。程序在本手册的配套光盘中，先看使用说明，再运行 setup71.exe。

目录(上册)

一、 初探 Proteus

在用 Proteus 进行电路设计和仿真之前，我们先装好 Proteus，并看看 Proteus 自带的例子，熟悉 Proteus 的基本用法和功能。

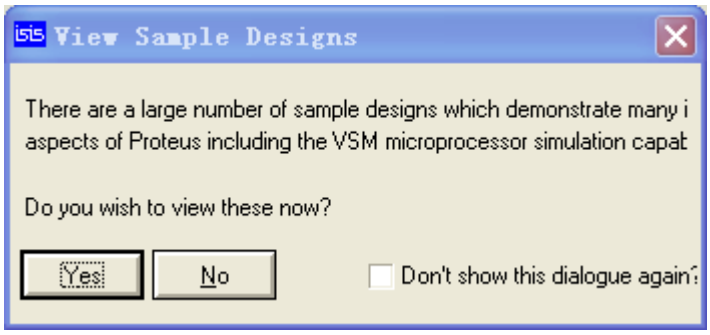
安装 Proteus

详细说明见 Proteus 安装光盘的“使用说明.txt”。

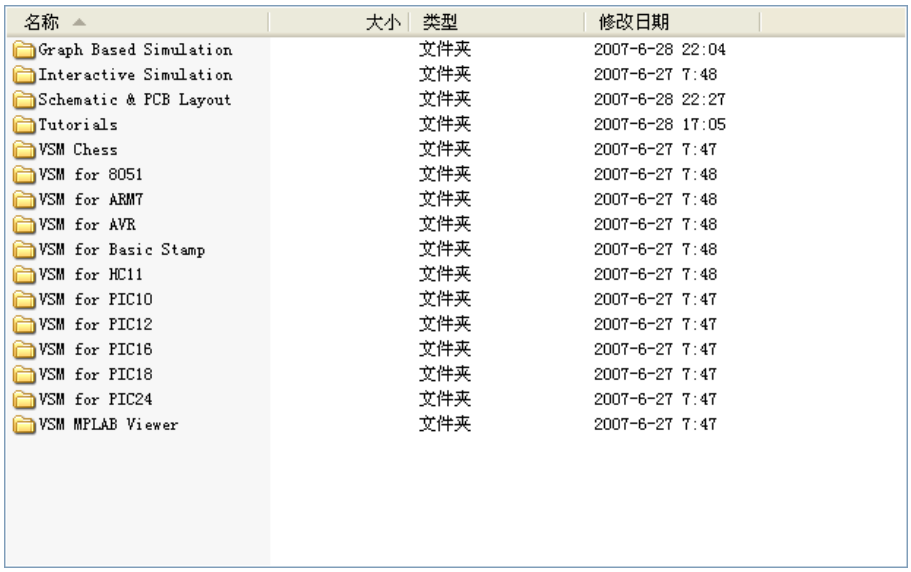
启动 Proteus 原理图设计

开始 > 所有程序 > Protues 7 Professional > ISIS 7 Professional

在 Proteus 启动时弹出的 View Sample Designs 窗口中选择 Yes，查看 Proteus 自带的例子。



Protues 7 自带的例子有 16 组，如下图所示。



详细说明见附录 1。

仿真

Proteus 原理图设计 ISIS 7 Professional 启动后就可以使用仿真按钮对电路进行仿真，仿真按钮  有四个，功能分别是启动仿真、单步运行调试仿真、暂停仿真和停止仿真。

体验 Proteus 自带的范例

详细内容见附录 1。

二、 Proteus 虚拟仪器的使用

学习光盘附带资料：

- Proteus 介绍.pdf
- PROTEUS 特点.pdf
- ProteusVSM.pdf
- PROTEUS 应用.pdf
- Proteus 教程 1 之入门.pdf
- Proteus 教程 2 之修改元件.pdf
- VSMSHOW.ppt
- Proteus 中文入门教程.doc

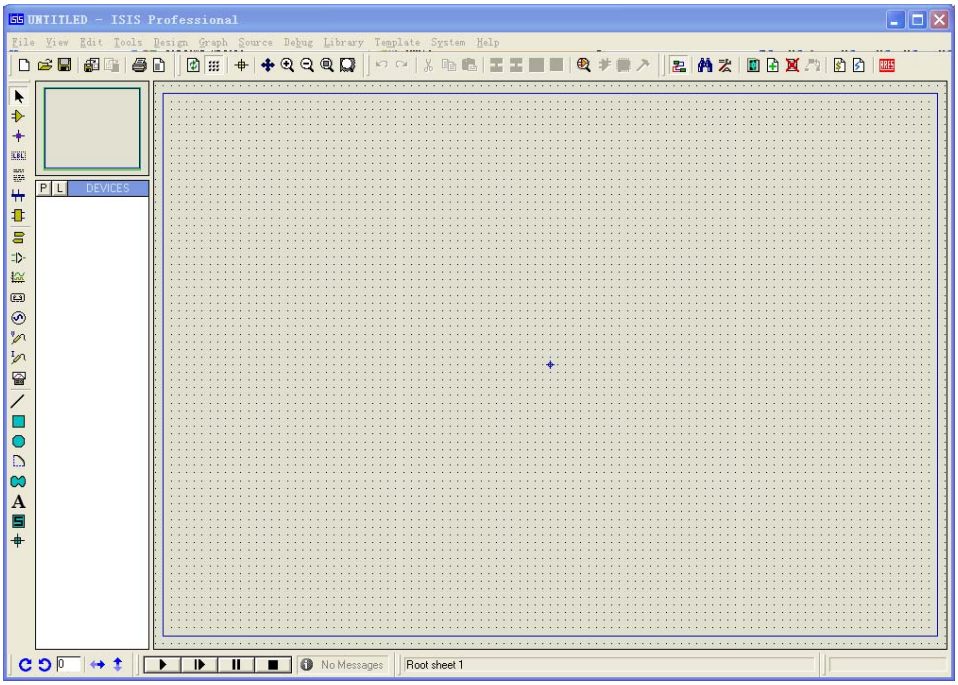
三、 Proteus 电路设计

使用 Proteus 仿真的基础是要准确绘制原理图，并进行合理的设置。绘制原理图使用 ISIS 原理图输入系统。下面以一个实际的 ARM 仿真为例，介绍如何使用 Proteus 进行电路原理图设计。

启动 Proteus 原理图设计

- 开始 > 所有程序 > Protues 7 Professional > ISIS 7 Professional

启动画面

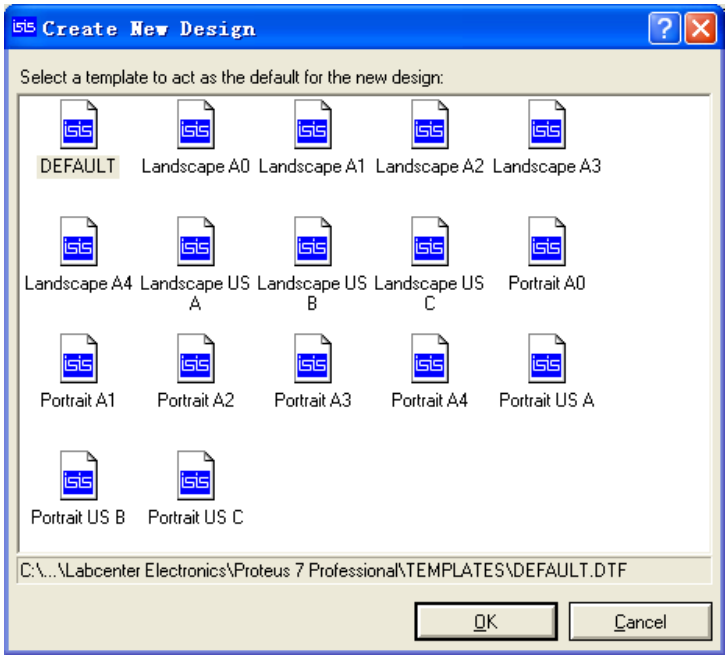


原理图编辑界面

默认的绘图格点为 100 th (1 th = 0.001 inch)。如果希望使用标准纸张绘图可选择模板。

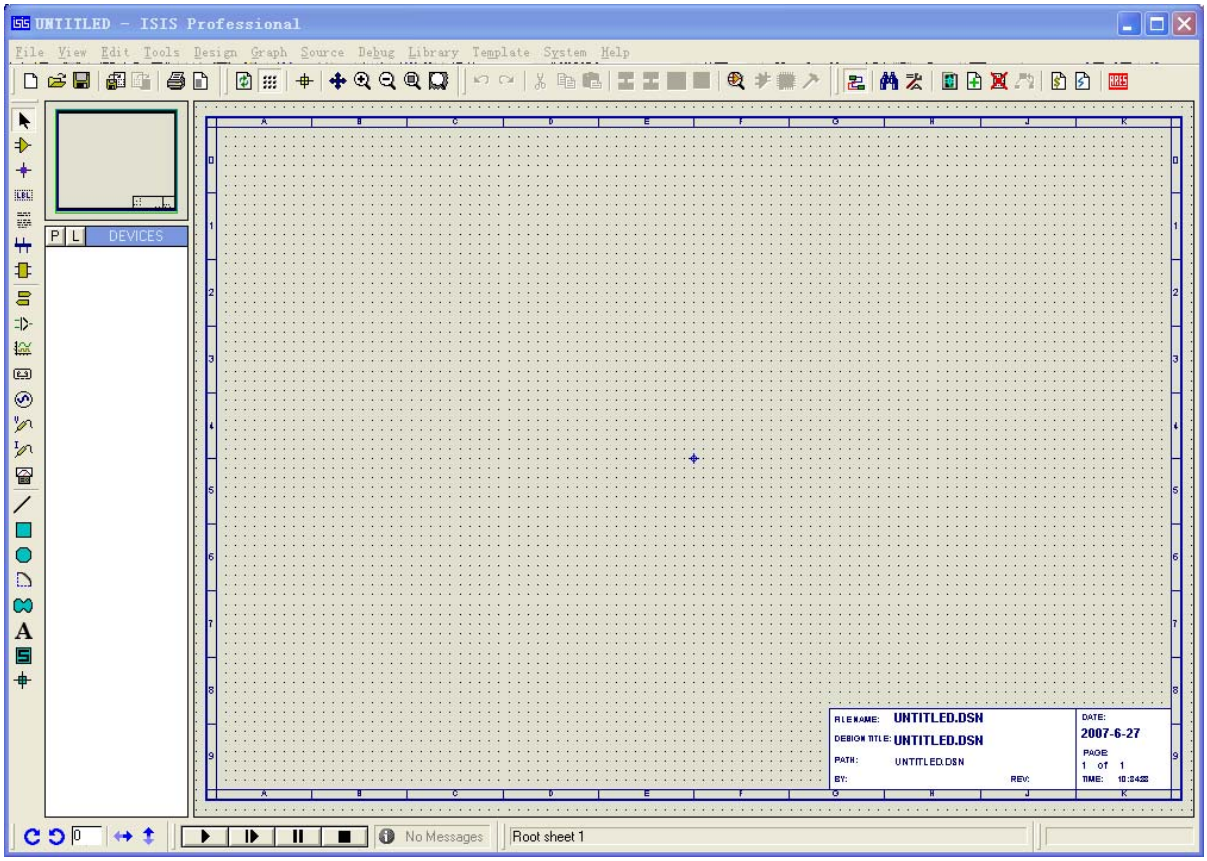
选择模板

- File > New Design



选择模板

- 选择 Landscape A4 模板 > OK。

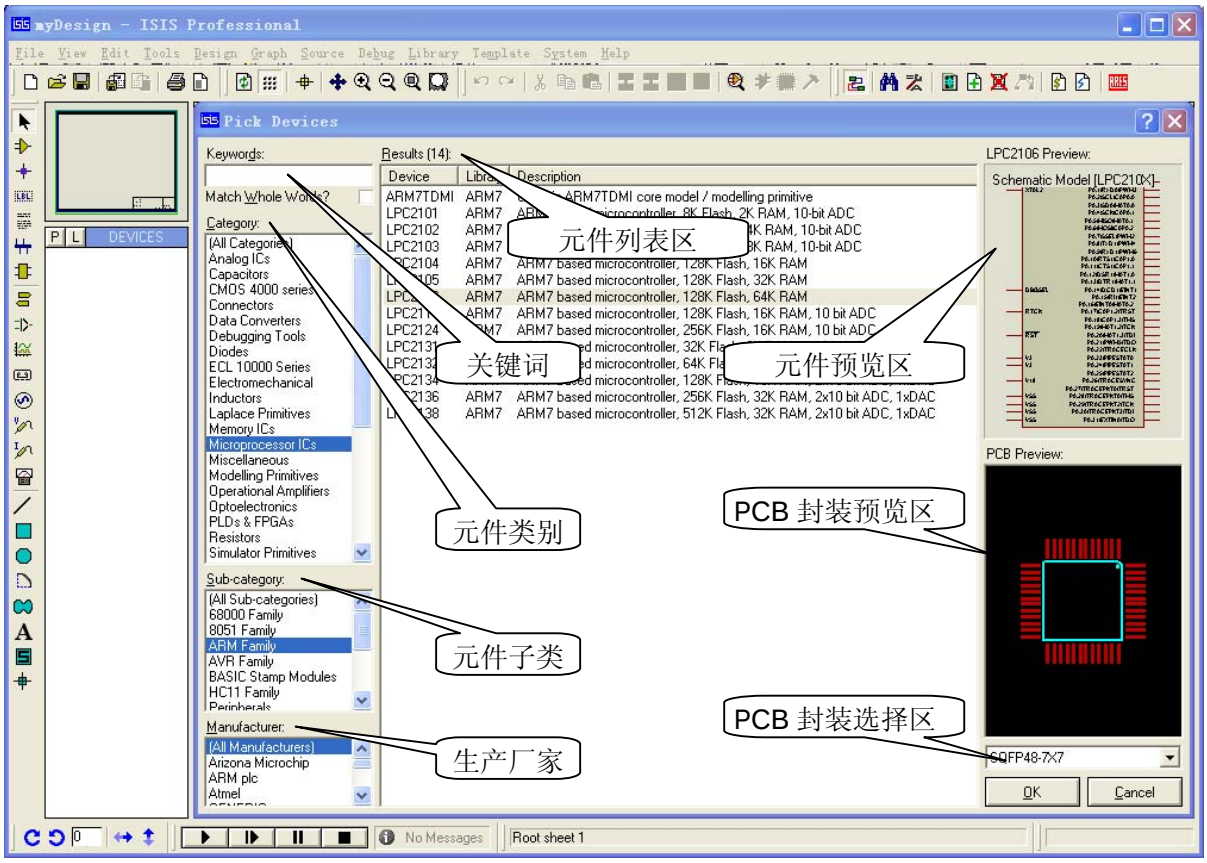


添加模板

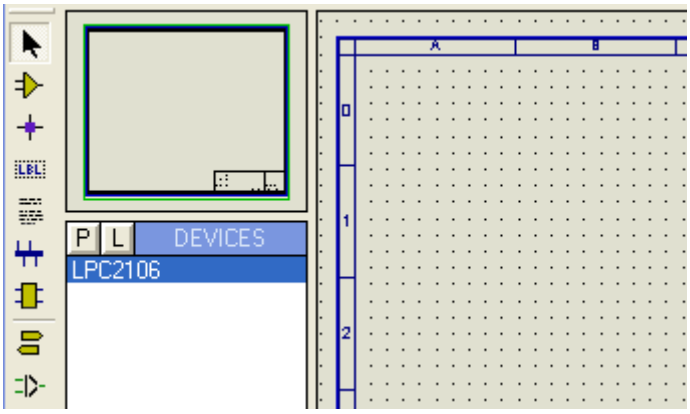
- 保存(Save Design)：counter

选取元件

- Library > Pick Devices/Symbol...。

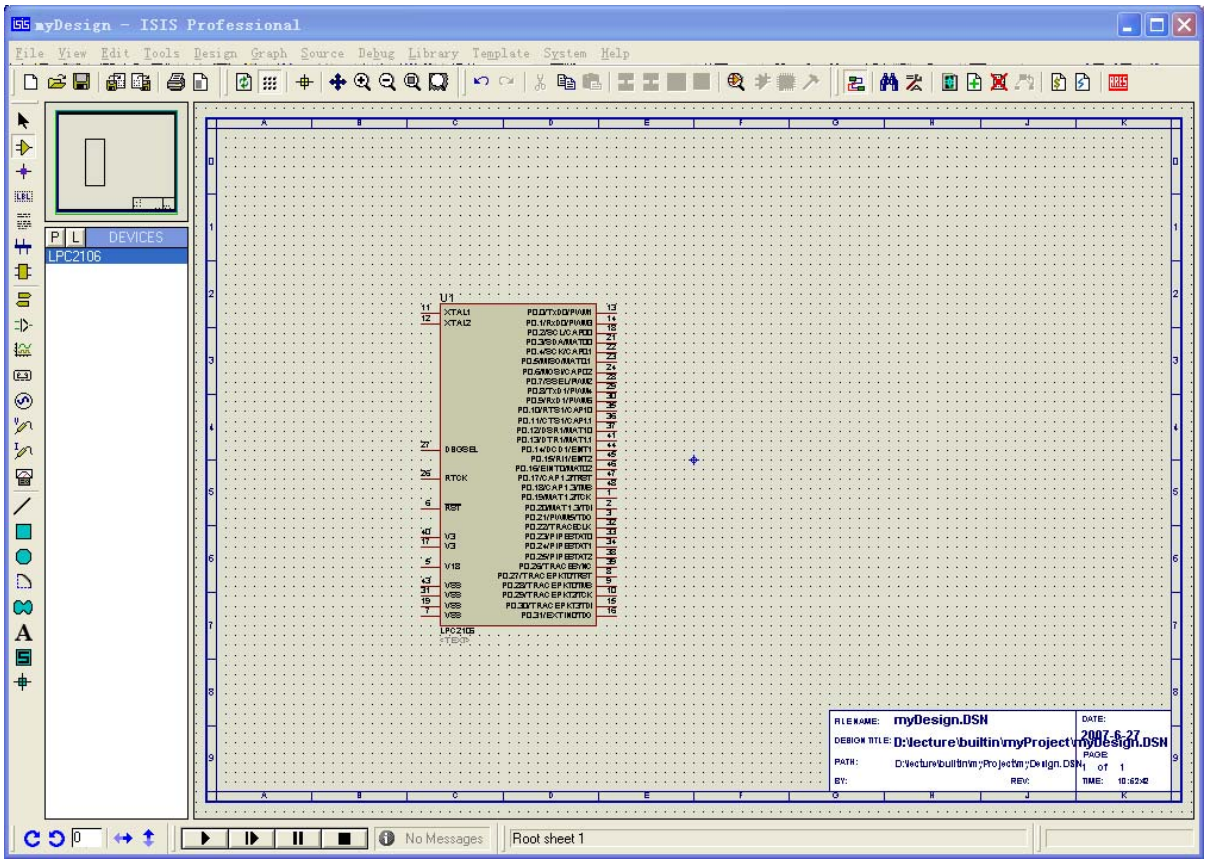


选取元件

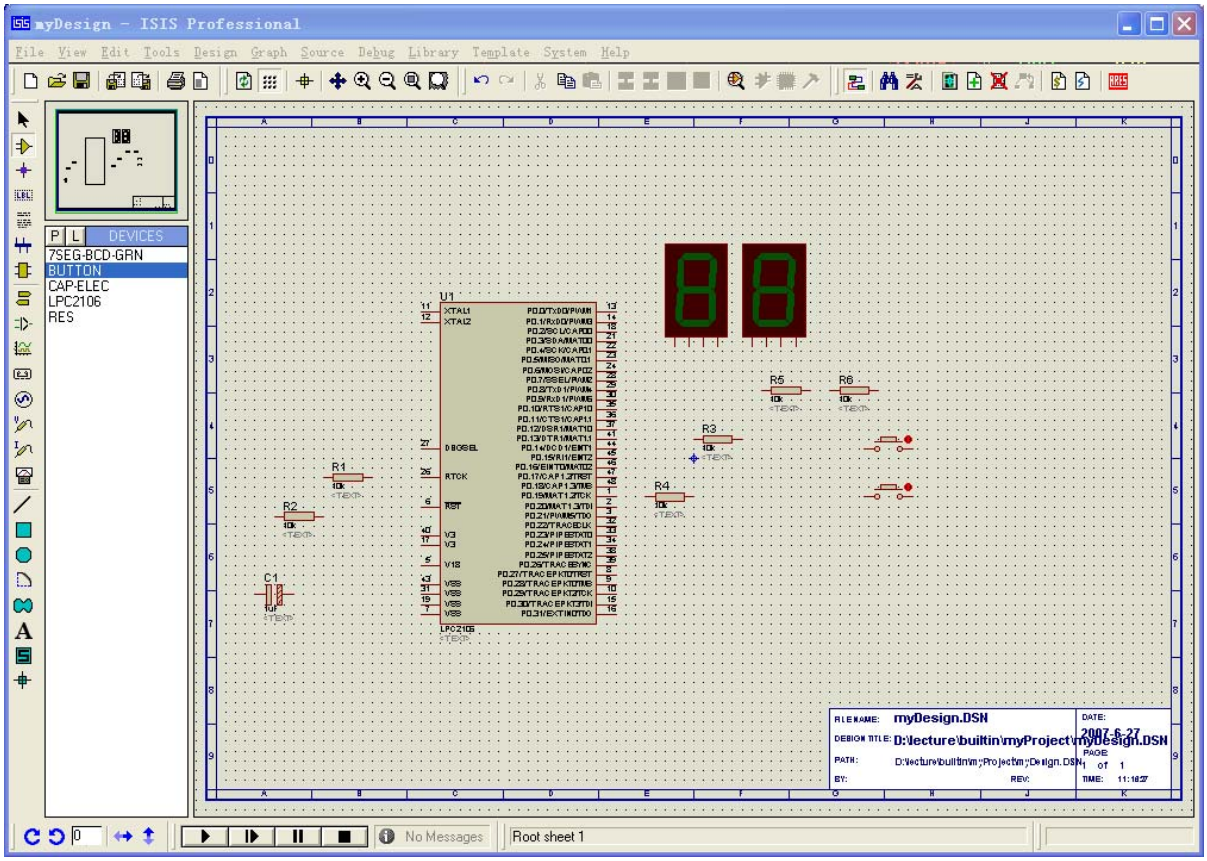


选择的元件

- 右击设计图纸 > Place > Component > LPC2106 > 放在合适的位置。
- 摆放其他元件。
- 元件以默认的方向摆放，可以使用左下角（或右击元件）的旋转与翻转命令，改变元件的方向。
- 双击元件可以改变元件的参数，如电阻的阻值等。
- 鼠标滚轮用来缩放图纸。
- 改变图纸大小：System > Set Sheet Sizes...。
- 通过 View 菜单改变点格大小。
- 窗口最左侧是工具箱。

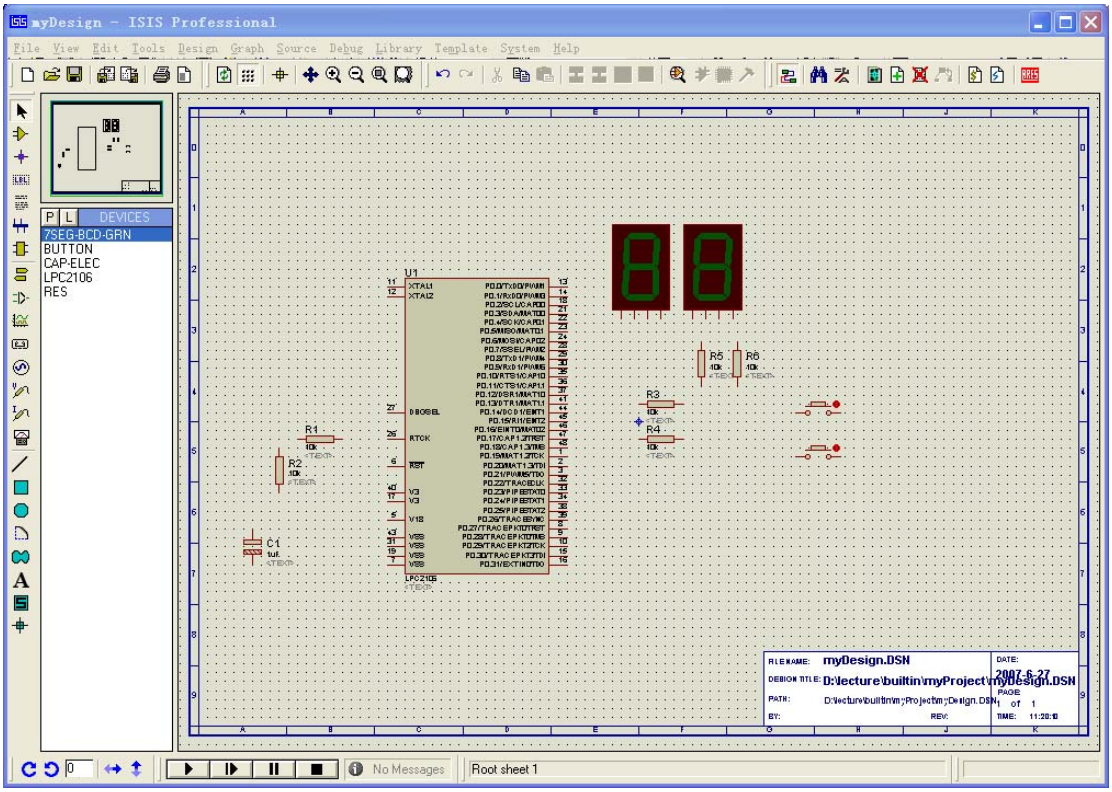


摆放元件 LPC2106



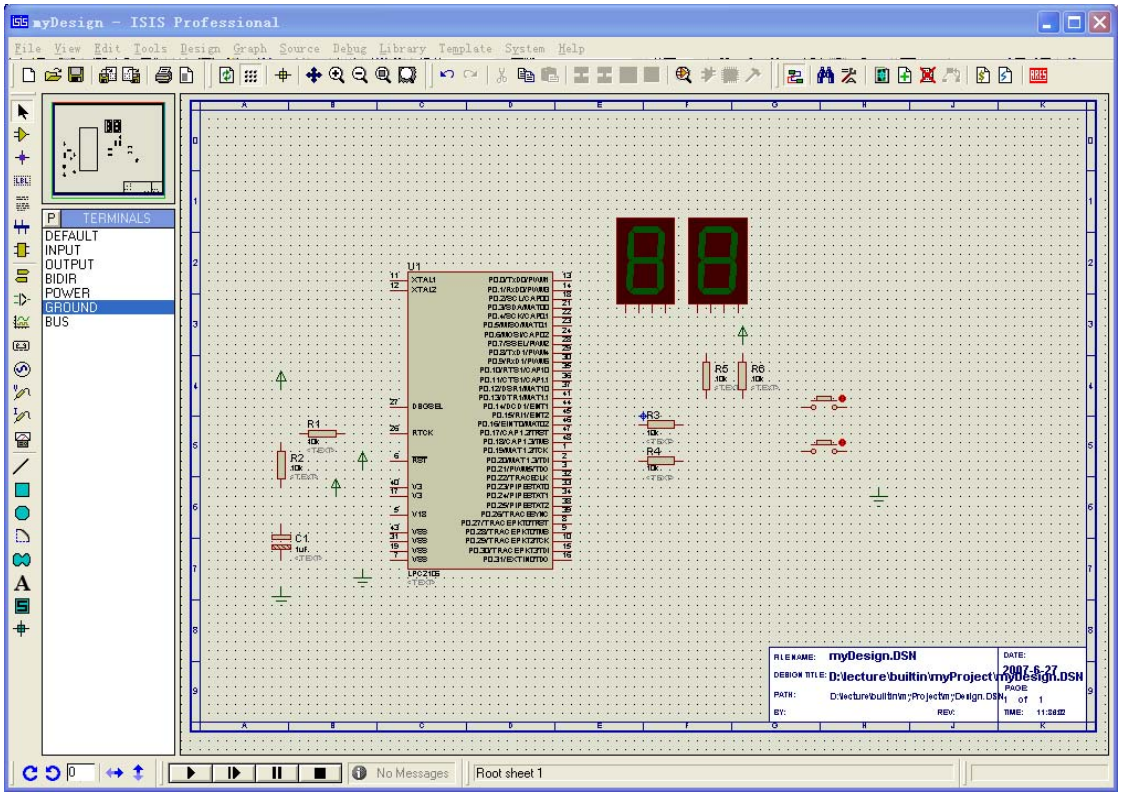
摆放其他元件

- 右击目标元件 > Rotate。



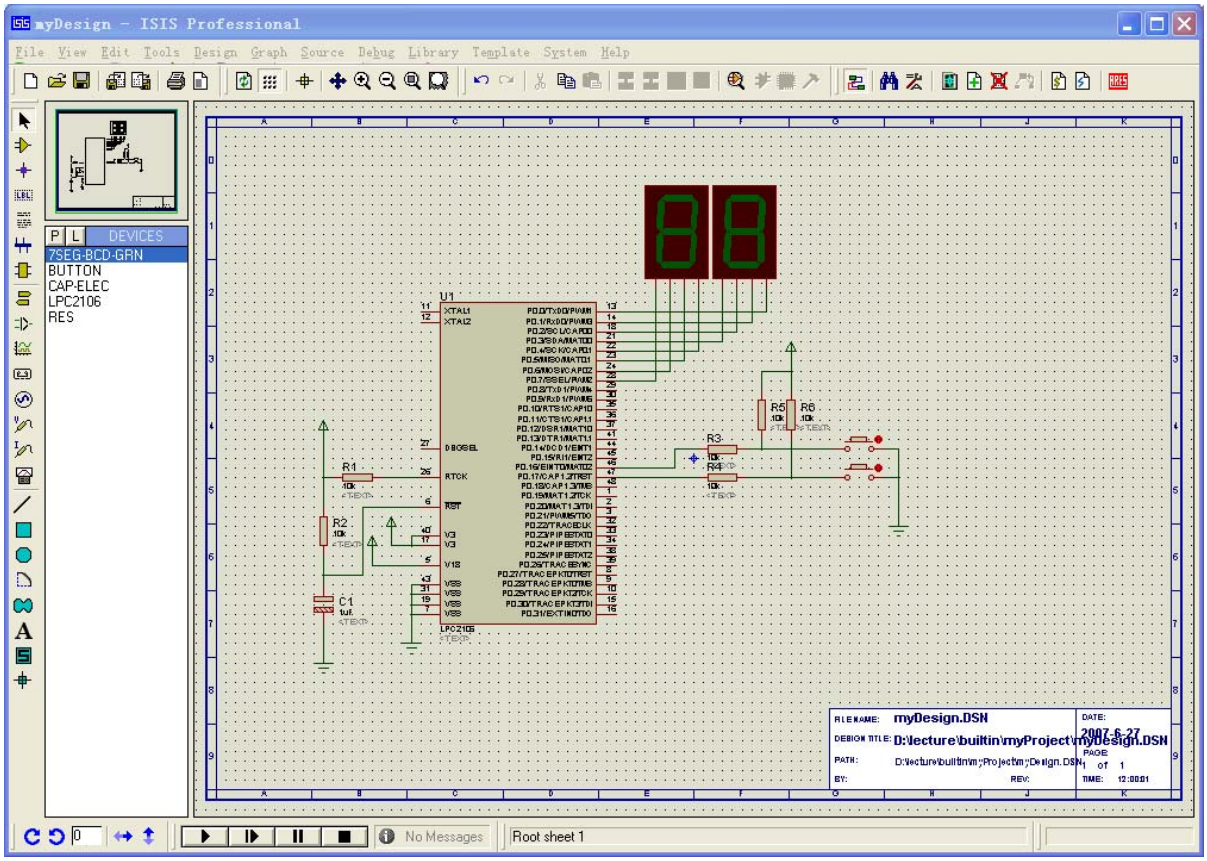
改变元件方向

- 添加电源：单击工具栏中的图标，或右击设计图纸 > Place > Terminal > POWER。
- 添加接地终端：右击设计图纸 > Place > Terminal > GROUND。



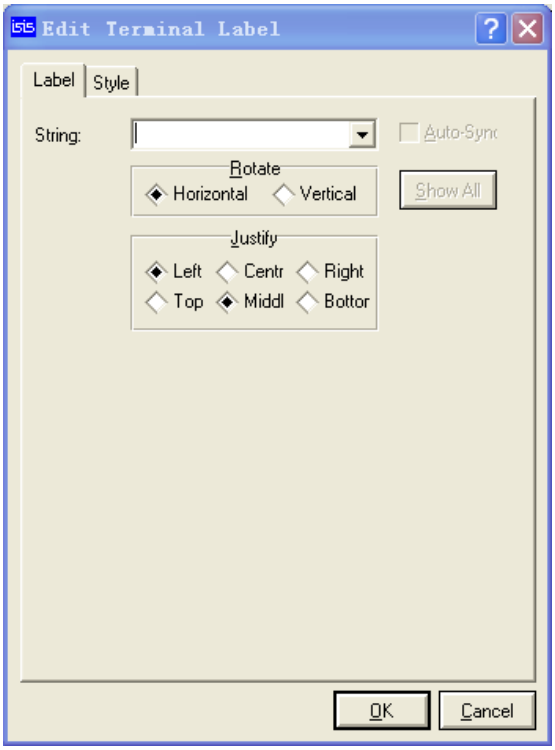
添加电源和接地终端

- Proteus 支持自动布线。分别单击两个引脚（注意对准小红框），两个引脚之间会自动添加走线，也可以手动走线，或手动修正走线。



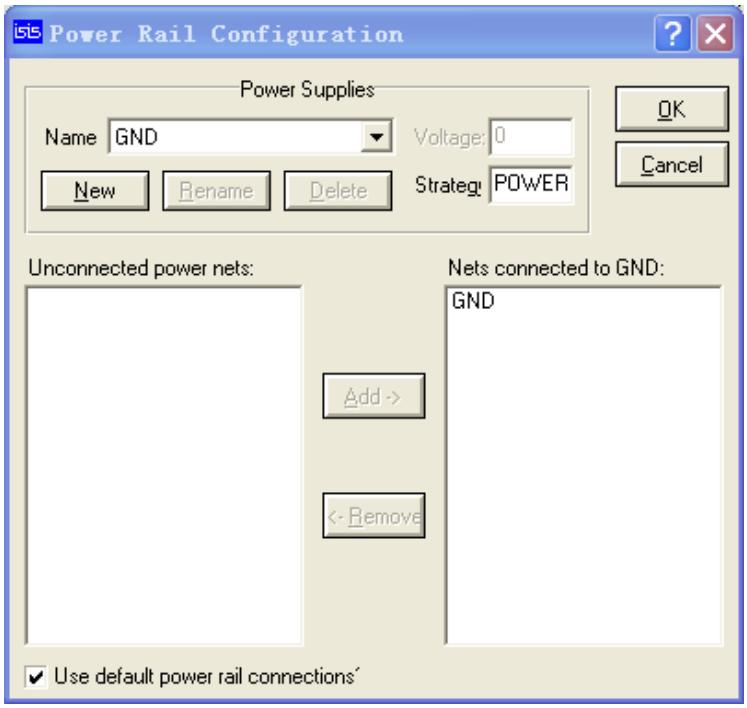
连接走线后的电路图

- 添加电源电压标签。双击电源终端，在“Edit Terminal Label”对话窗口中输入电压值。



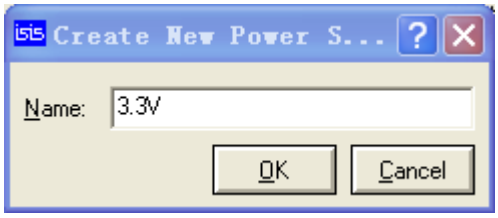
编辑终端标签

- **配置电源电压**。Design > Configure Power Rails ...。**注意：**配置电源对仿真很重要。



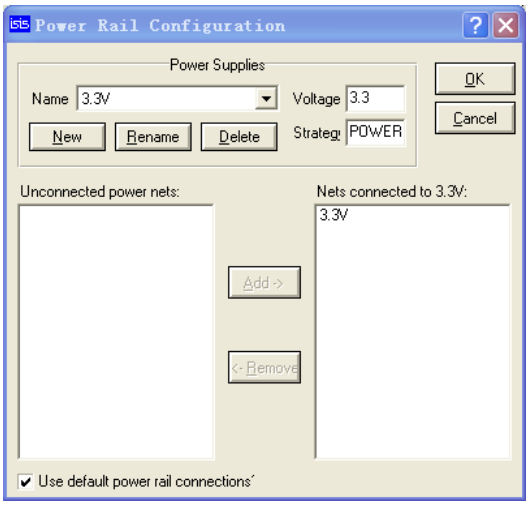
配置电源

- 输入电源电压值。单击 New，在弹出的对话框中输入电压值 3.3V，添加电源供给。
  提示：如果我们在定义电源时带上符号，例如+3.3V，软件会自动加上这个电源，免得麻烦。

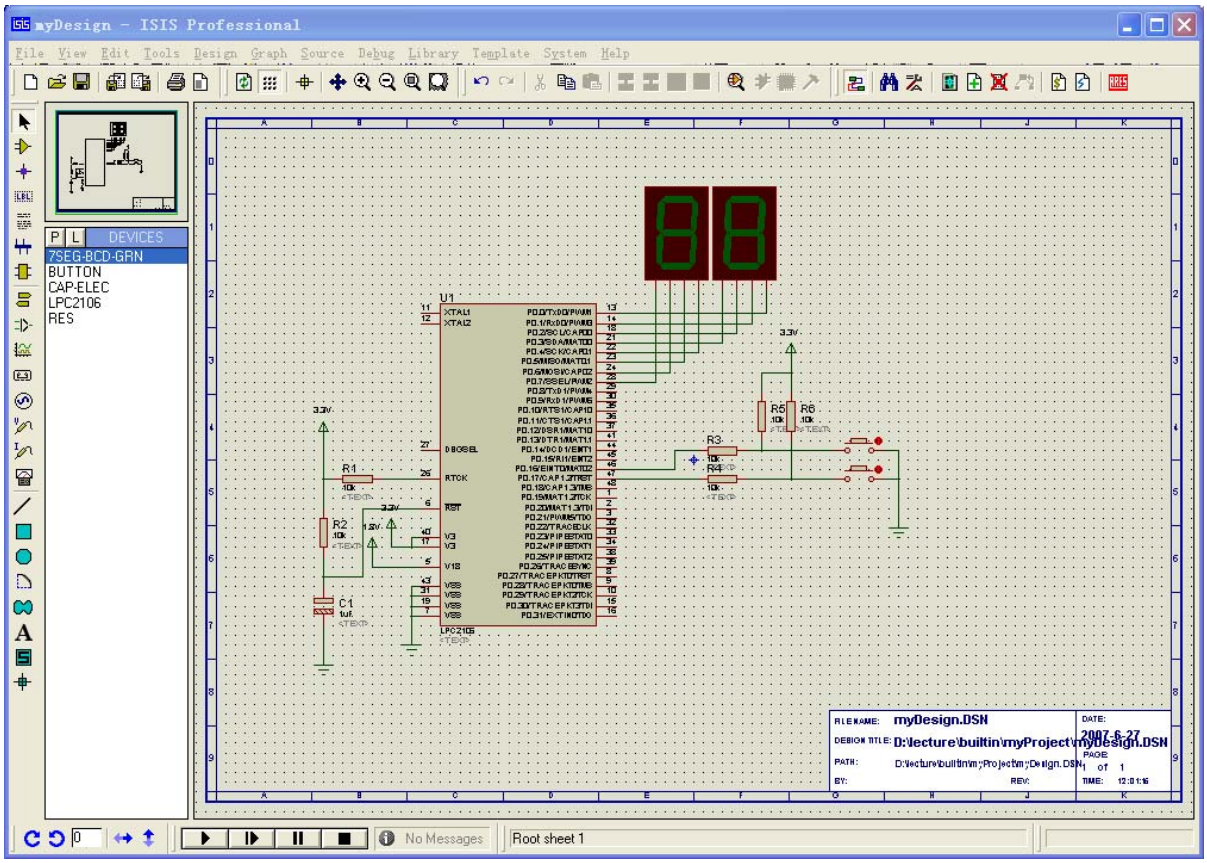


输入电压值

单击 OK，选择 Unconnected power nets 列表框中的电压值 3.3V > 单击 Add 按钮，右侧列表框显示 3.3V。用同样的方法添加 1.8V 电源。

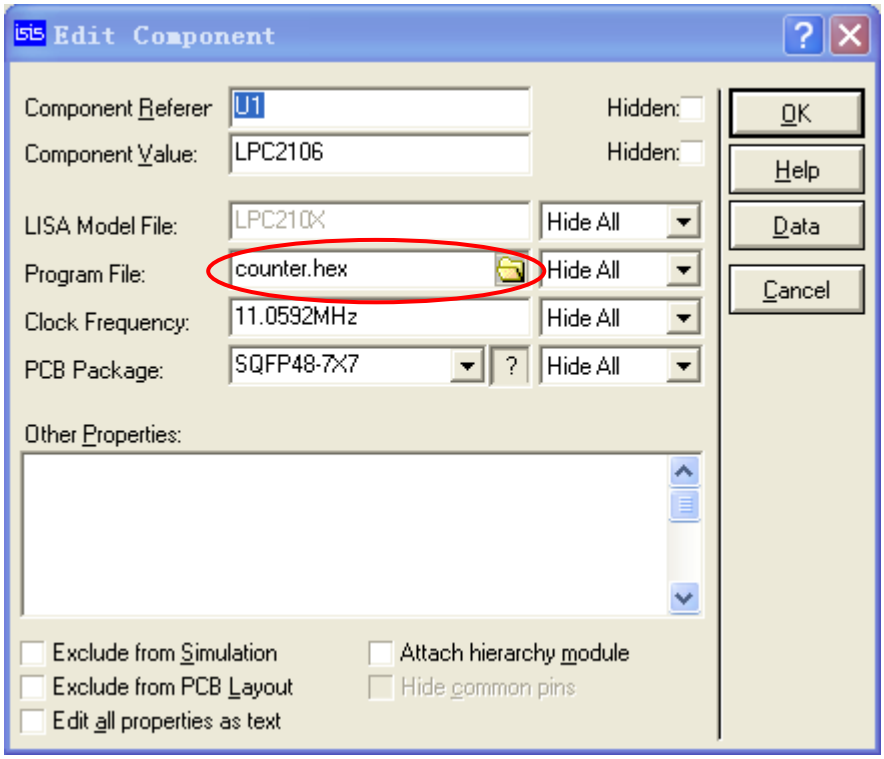


配置好的电源



完整电路图

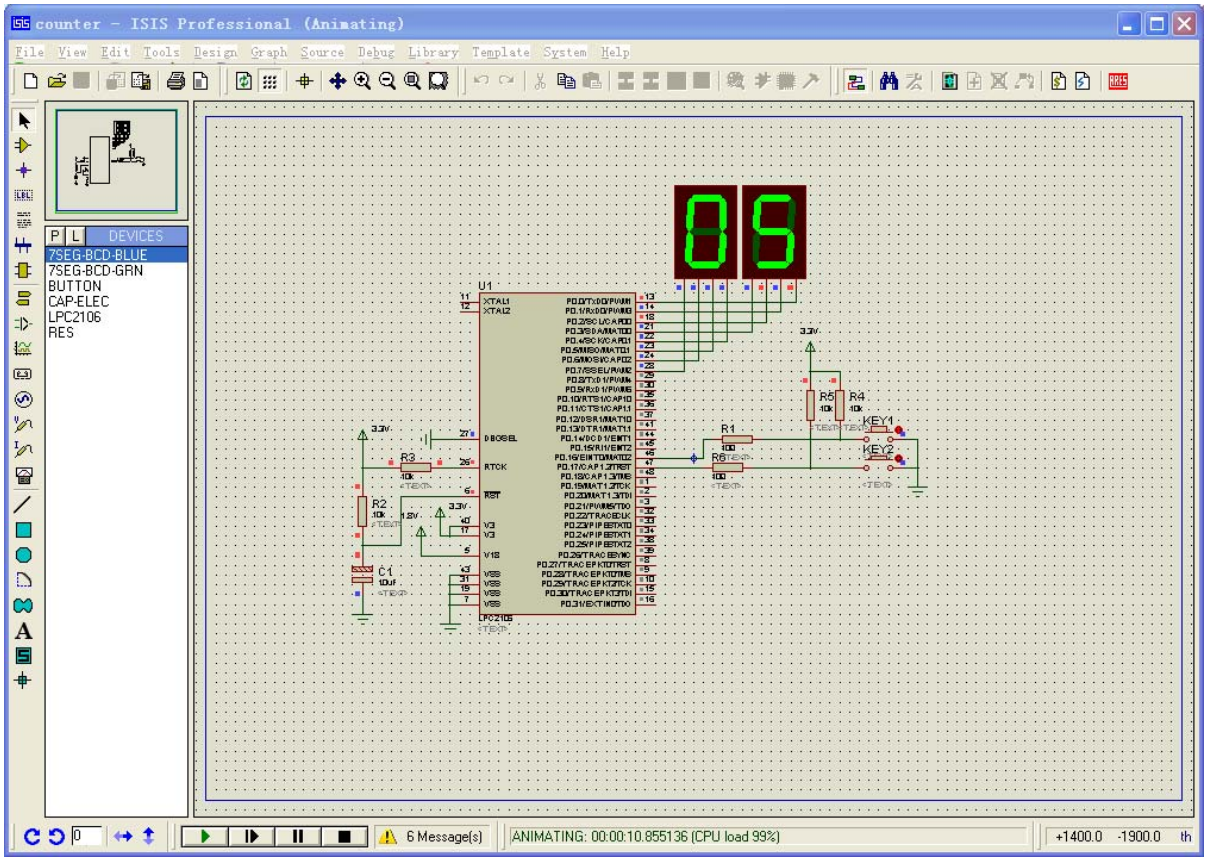
- 双击 LPC2106 芯片，弹出 Edit Component 对话框。



Edit Component 对话框

- 单击 Program File 的浏览按钮，添加已经编译好的目标程序文件 counter.hex，OK。

- 单击启动仿真按钮  ，运行仿真，单击电路图中按键可观察到数码管数字增大或减小。



仿真结果

这 4 个按钮的功能分别是：启动仿真、单步运行仿真、暂停仿真和停止仿真。

单击停止仿真按钮，停止运行。

注意：在仿真过程中按 KEY2 时 Proteus 下方的状态栏中可以看到如下的提示：

这是因为 KEY1 和 KEY2 靠的太近了，可以通过按 KEY2 右旁的小红点进行操作，或者把 KEY2 向下拉远一些。

四、 Keil for ARM 安装

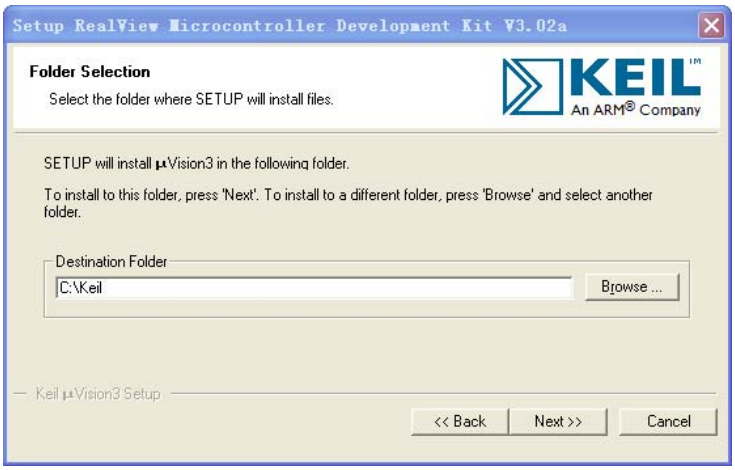
安装 Keil for ARM

- 在本手册的配套光盘中找到 mdk302a.exe，双击开始安装。

注意：安装完成后 Help > About 中的版本与安装文件的版本 V3.02a 不同。

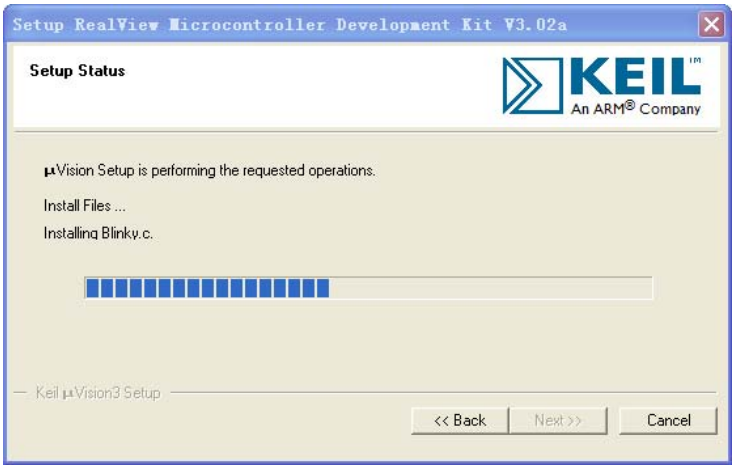
- Next

- 接受许可协议，Next

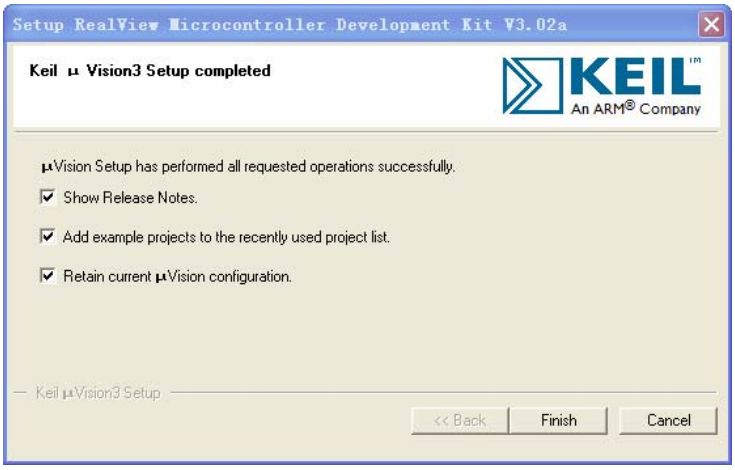


- 选择安装目录，下一步

- 填写客户信息，下一步



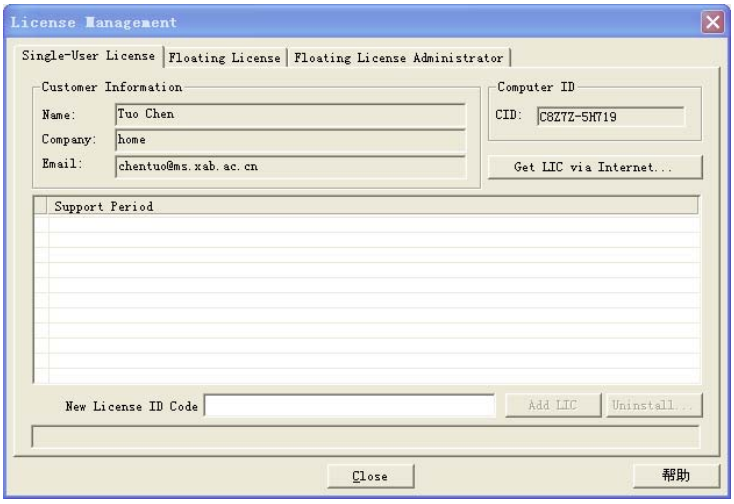
- 等待安装过程结束，下一步



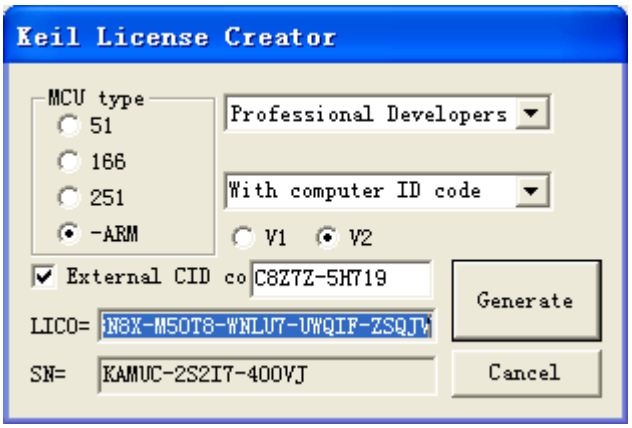
- 结束 Finish

注册 Keil for ARM

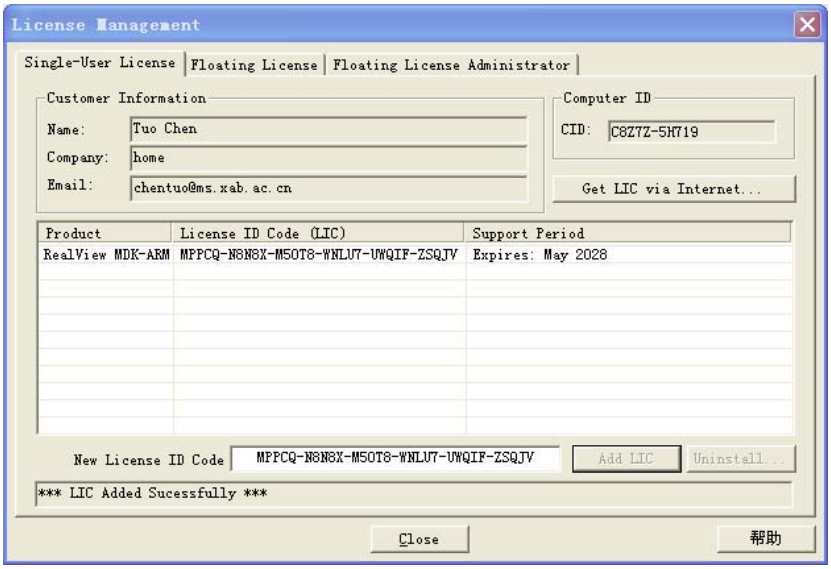
- 启动 Keil uVision3，File > lisence management



- 在本手册的配套光盘中找到 Keil_lic_v3.2.exe，双击运行



将 Keil License Management 窗口中的 CID (这里是 C8Z7Z-5HL1Y)复制到 Keil License Creator 的 External CID co 中，点击 Generate 按钮，生成 LICO 和 SN。将 LICO 复制到 Keil License Management 窗口中的 New License ID Code 输入框中，单击 Add LIC 按钮，如图。



*** LIC Added Sucessfully *** 表示 License ID Code 添加成功。

- Close

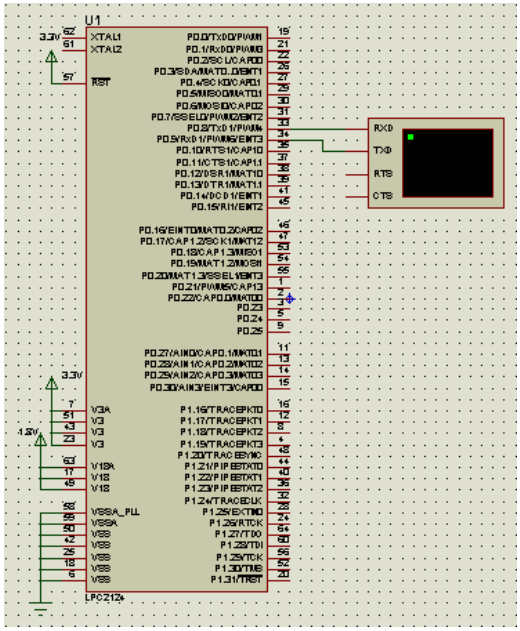
五、 Keil for ARM 实例 1：UART 程序设计与电路仿真(使用 Keil CARM 编译器)

使用 UART1 来输出字符“Hello World”。

注释：UART (Universal Asynchronous Receiver/Transmitter) 通用异步收发器。UART0 也称串口 1；UART1 也称串口 2。

电路设计

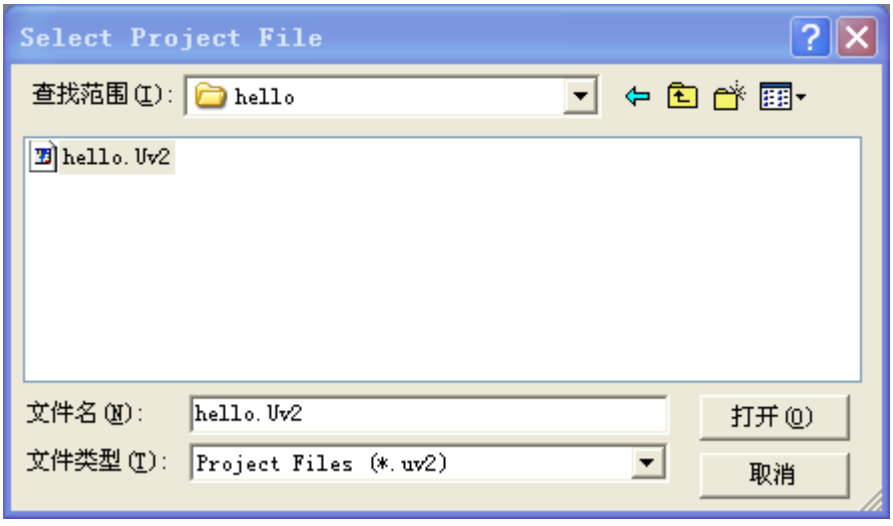
在 Proteus 的 ISIS 中设计使用 UART1 的电原理图如下。



电路原理图

打开 Keil 项目

打开已有的项目：Project > Open Project...

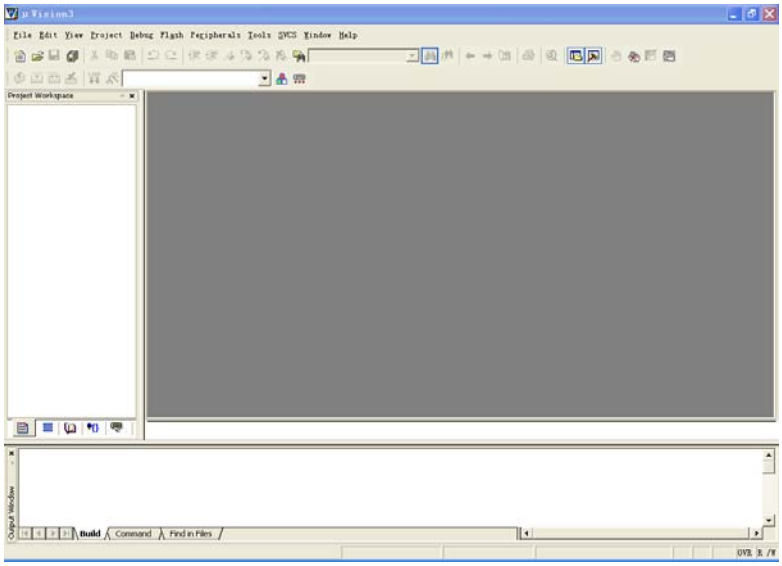


选择项目

如果要新建项目，请看下面 - 创建 Keil 项目。

创建 Keil 项目

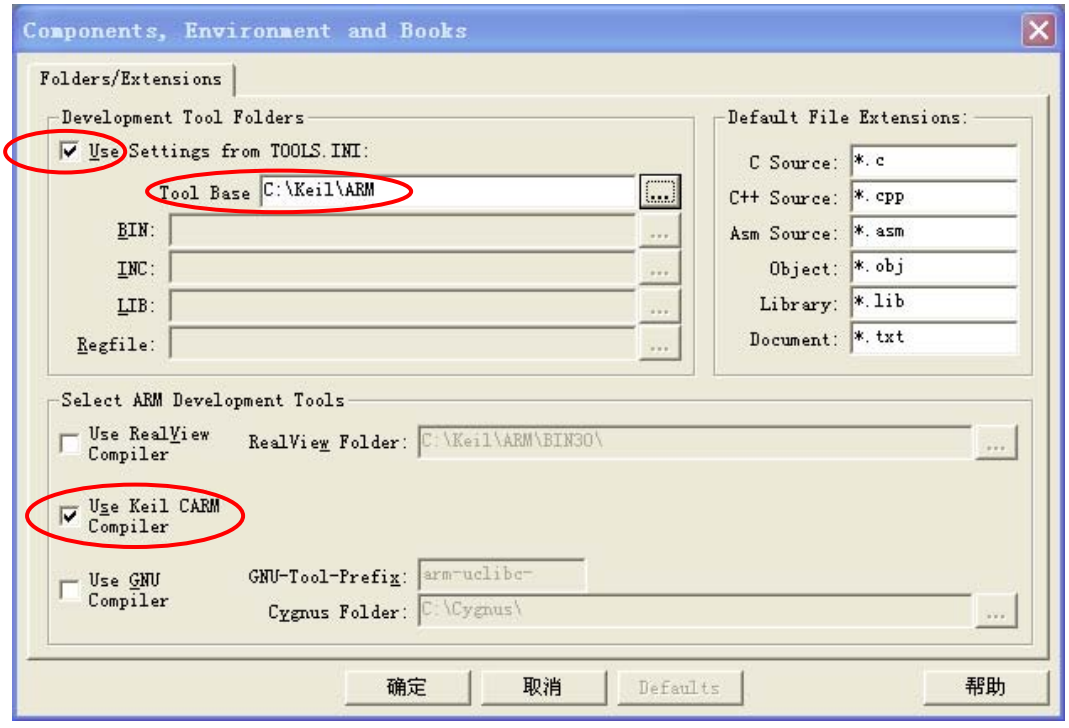
- 启动 Keil uVision3



Keil uVision3 界面

- 设定编译器 Keil 将依据不同的编译器生成不同的启动代码。

(1) Project > Components, Environment, Books...或单击图标，弹出 Components, Environment and Books 对话框。



设置编译器

在 Environment and Books 对话框中可以选择编译器、设置工具、库文件和 INC 文件等的路径及默认文件扩展名。

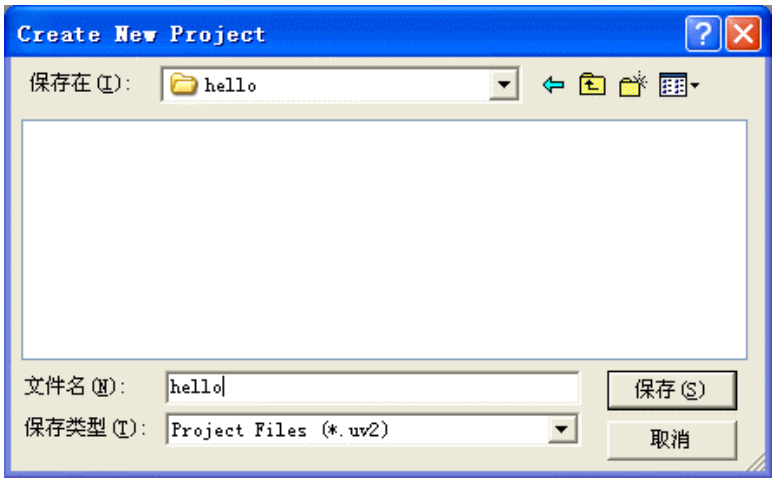
(2) 如图选中 Use Keil CARM Compiler 和 Use Setting from TOOLS.INI 复选框，更改编

译器。

(3) 单击“确定”按钮保存设置。**注意：**破解的 mdk302a 以上版本在这里会出问题。

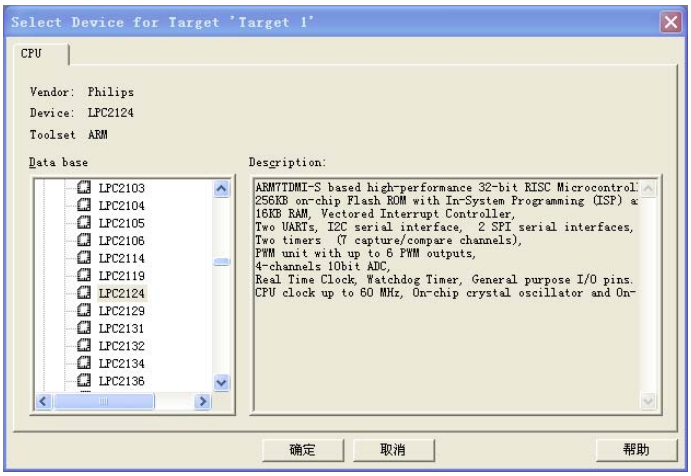
- 建立 Keil 项目

(1) Project > New Project，显示 Create New Project 对话框，指定项目路径并输入文件名 hello。



建立新项目

(2) 单击“保存”按钮，显示 Select Device for Target ‘Target 1’ 对话框，如图。



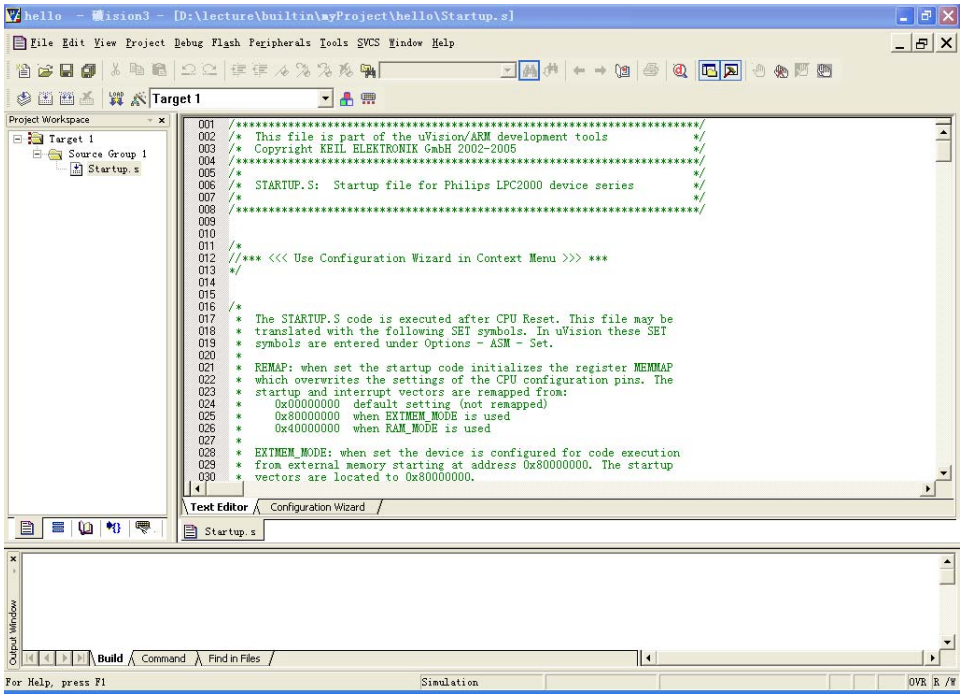
选择 CPU 类型

在 Data base 窗格中选择 CPU 类型(Philips 公司的 LPC2124 芯片)，Description 窗格中显示该芯片的资源描述。

(3) 单击“确定”按钮，弹出提示信息，确认是否复制 LPC2100 启动代码到项目文件夹并添加文件到项目中。这里的启动代码是依据前面设置的编译器生成的，现在生成的启动代码与设置为 GNU 编译器时生成的启动代码不同。所以我们要先选定编译器。

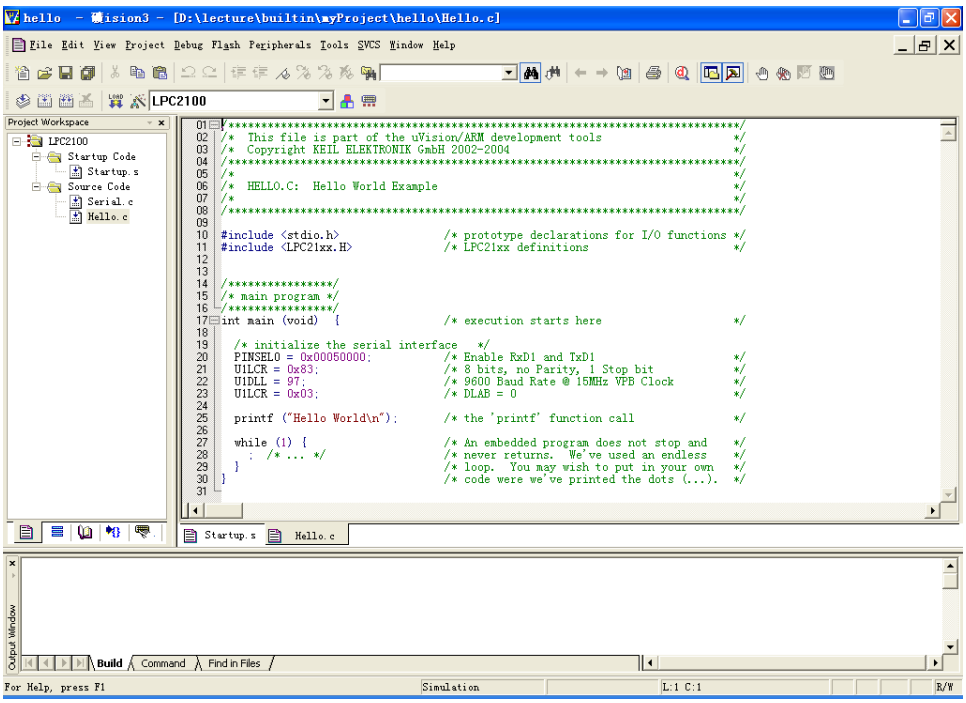
提示信息

(4) 单击“是”，添加启动代码（如果选择“否”，可以添加自己编写的启动代码）。这样将启动代码会添加到项目中，在 Project Workspace 窗格中显示出已添加的启动代码文件，双击 Startup.s 可查看该代码，如图。**注意：**KEIL MDK 3.03 以上就不带 KEIL 本身的 CARM 编译器了，所以我们要使用 mdk302a。



启动代码

(5) 在工作空间 Project Workspace 中选择 Target 1，再单击该文本，修改为 LPC2100，并将 Source Group 1 修改为 Startup Code。在 Project Workspace 的空白处右击并选择 New Group 命令，将其名称修改为 Source Code；选中 Source Code，右击并选择 Add Files to Group ‘Source Code’命令，将 Serial.c 和 Hello.c 文件添加到项目中。



添加用户代码

相关的基础知识

请看 Hello 目录下《UART 程序设计》。

启动文件

附录 6：Keil GCC 启动文件 Startup.s

C 程序代码

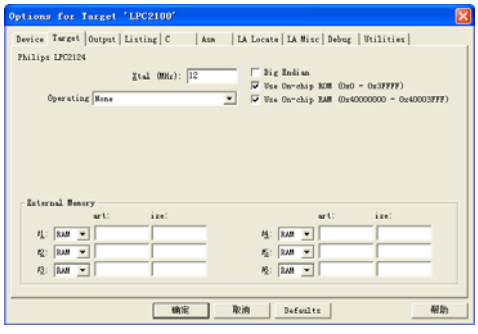
```
/*****************************************************************************/
/*  该文件是 uVision/ARM 开发工具的一部分，在 Keil 安装目录下的 Examples\Hello 中       */
/*  Copyright KEIL ELEKTRONIK GmbH 2002-2004                                 */
/*****************************************************************************/
/*  HELLO.C:  Hello World Example。                                          */
/*****************************************************************************/
#include <stdio.h>                          /* I/O 原型描述 */
#include <LPC21xx.H>                        /* LPC21xx 定义，在 Keil 安装目录下可以找到*/
```

注意：通常在 Keil 安装目录下的头文件放在<>中，在用户目录下的头文件放在""中。

```
/*****************/
/* 主程序        */
/*****************/
int main (void)  {                          /* 程序从这儿开始执行 */

  /* 初始化串口 */
  PINSEL0 = 0x00050000;                     /* 设定 UART1 (RxD1 和 TxD1) */
  U1LCR = 0x83;                             /* 8 位数据位，无奇偶校验，1 位停止位，DLAB = 1 */
  U1DLL = 97;                               /* 在 12MHz 时钟下，波特率为 9600 bps */
  /* U1DLM = 0x00; */                       /* 除数的高 8 位，因为是 0，可以省略 */
  U1LCR = 0x03;                             /* DLAB = 0，进行其他操作所必须 */
  printf ("Hello World\n");                 /* 调用 'printf' 函数输出字符 */
  while (1) {                               /* 一个式的程序永远不会停止和返回。 */
    ;  /* ... */                            /* 我们使用一个无限循环。*/
  }                                         /* 你可以将自己的代码放在这里 */
}
```

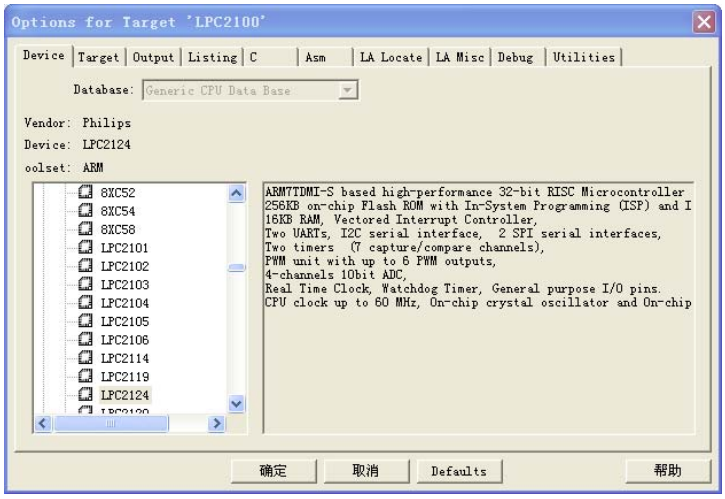
项目配置

(1) 在 Project Workspace 中选择 LPC2100，然后右击并选择 Options for Target ‘LPC2100’命令，在弹出的对话框中设置 Xtal 为 12MHz，并选中 Use On–chip ROM 和 Use On-chip RAM 复选框。



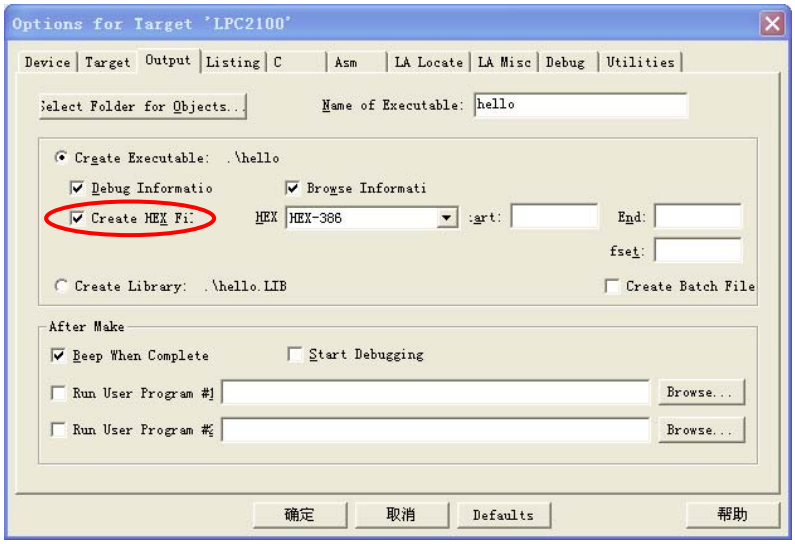
设置晶振频率和存储器

(2) 在上图中选择 Device 选项卡，显示 CPU 的类型及其相关的片资源，这是前面已经设定的 CPU 的类型，如下图所示。



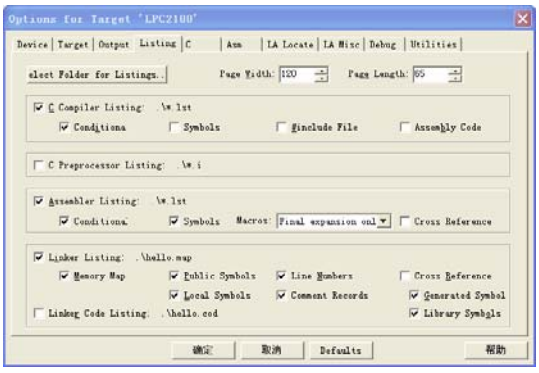
显示 CPU 信息

(3) 选择 Output 标签，选中 Create HEX File 复选框，如图，可生成 hex 格式的可执行文件。单击 Select Folder for Objects 按钮，指定输出文件的路径。



设置输出文件

(4) 选择 Listing 标签，再单击 Select Folder for Listings 按钮，指定生成的 list 文件的输出路径；然后选中 Assembler Listing 和 linker Listings 复选框。这些选项生成查看编译和链接的信息，如果你不关心它们，也可以不管这些设置。

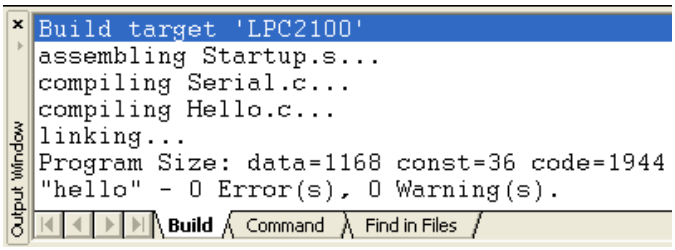


输出列表文件

其他标签保持默认设置，单击“确定”按钮保存设置。

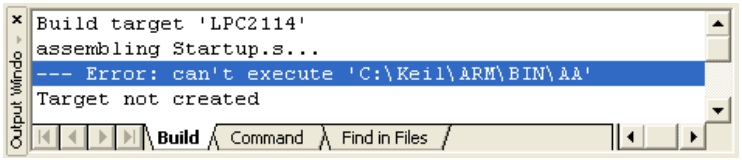
Build 目标

在 Project Workspace 中选择 LPC2100，然后右击并选择 Build Target (F7) 命令，或单击快捷图标  ，编译并链接项目，输出窗口如图。



显示编译结果

注意：KEIL MDK 3.03 以上就不带 KEIL 本身的 CARM 编译器了，执行编译出现错误如下：

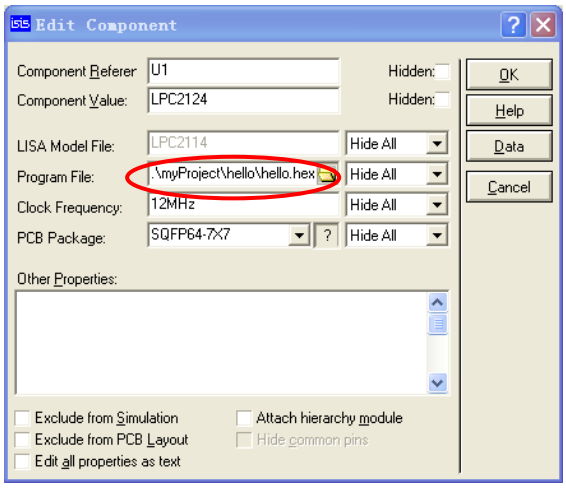


装回 KEIL MDK3.02 就可以正常编译了。

在 Protsus 中仿真

(1) 回到 Proteus 的 ISIS。

(2) 在电原理图中双击 LPC2124，弹出 Edit Component 对话框，指定程序文件 hello.hex 的路径，如图。(**注意：**用 Keil CARM 生成的带调试信息的二进制文件 hello.elf 仿真有问题)

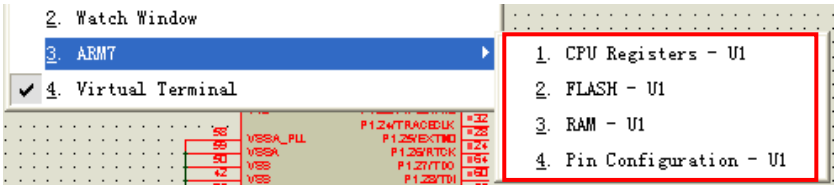


指定可执行文件

(3) 单击 Proteus 的仿真按钮  运行仿真，虚拟终端上显示字符 Hello World。

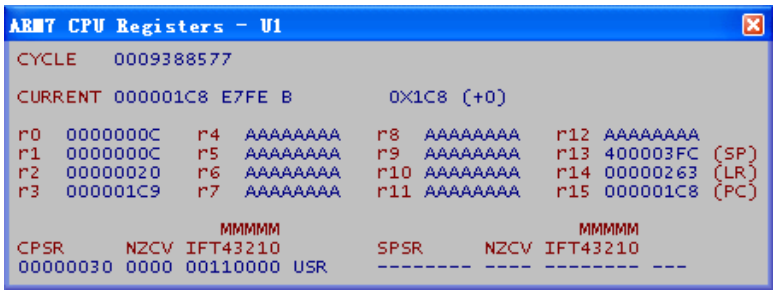
在虚拟终端上显示结果

(4) 单击 ⏸ 按钮，暂停仿真，在 Debug 菜单中用 ARM7 命令可以查看 CPU 寄存器、Flash、RAM 和引脚配置等信息。



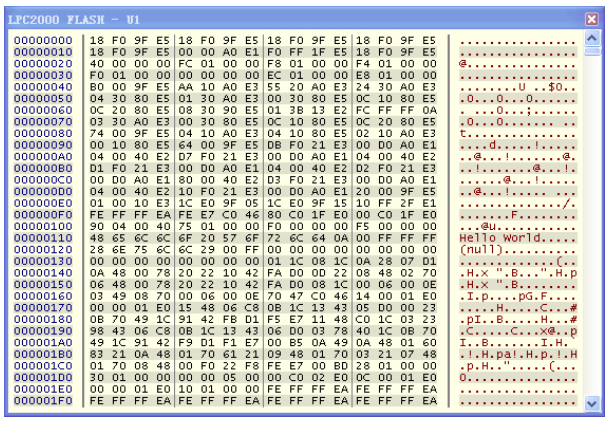
Debug > ARM7 命令

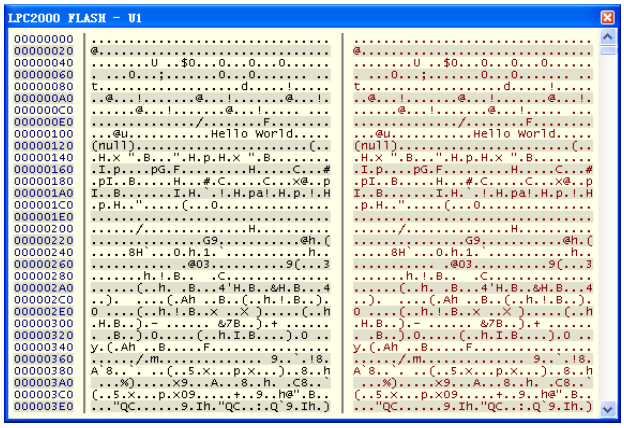
- 查看 CPU Registers



查看 CPU 寄存器

- 查看 Flash

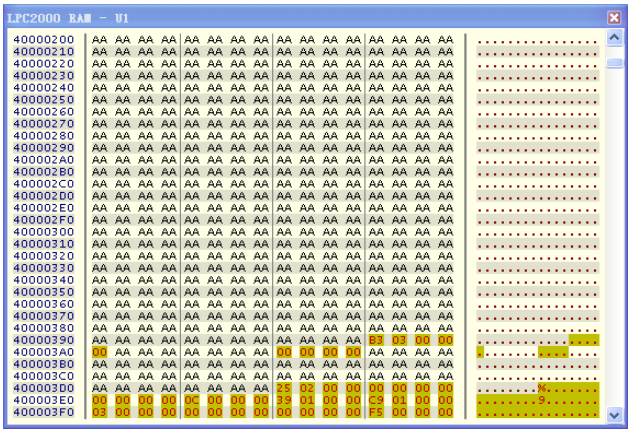




查看 Flash　　更改显示的数据类型为 Text

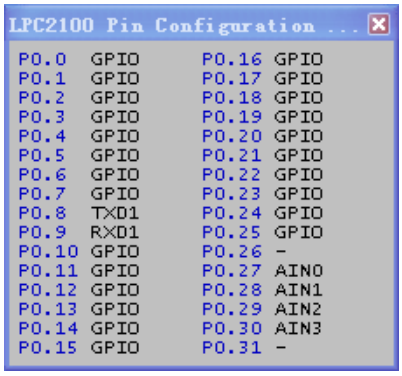
还可以右击信息显示窗口，从弹出菜单中更改显示方式。更改显示的数据类型为 Text。

- 查看 RAM



查看 RAM

- 查看 Pin Configuration



查看 Pin Configuration

(5) 单击  按钮，停止仿真。

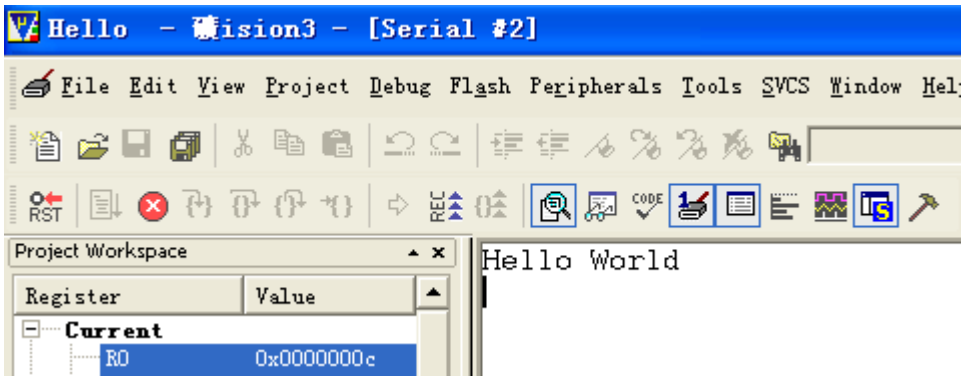
注意：用 Keil CARM 生成的带调试信息的二进制文件 hello.elf 仿真有问题，用 GCC 编译生成的 elf 文件可仿真，见例子“Keil for ARM 实例 2：A/D 程序设计与电路仿真(使用 GCC 编译器)”。

在 Keil 中仿真

我们也可以在 Keil 中仿真，步骤如下：

- 启动 Keil 的调试功能 Debug > Start/Stop Debug Session
- 打开串口 1 的仿真窗口 View > Serial Window #2
- 运行程序 Debug > Run

仿真结果如下：



使用 UART0 的问题

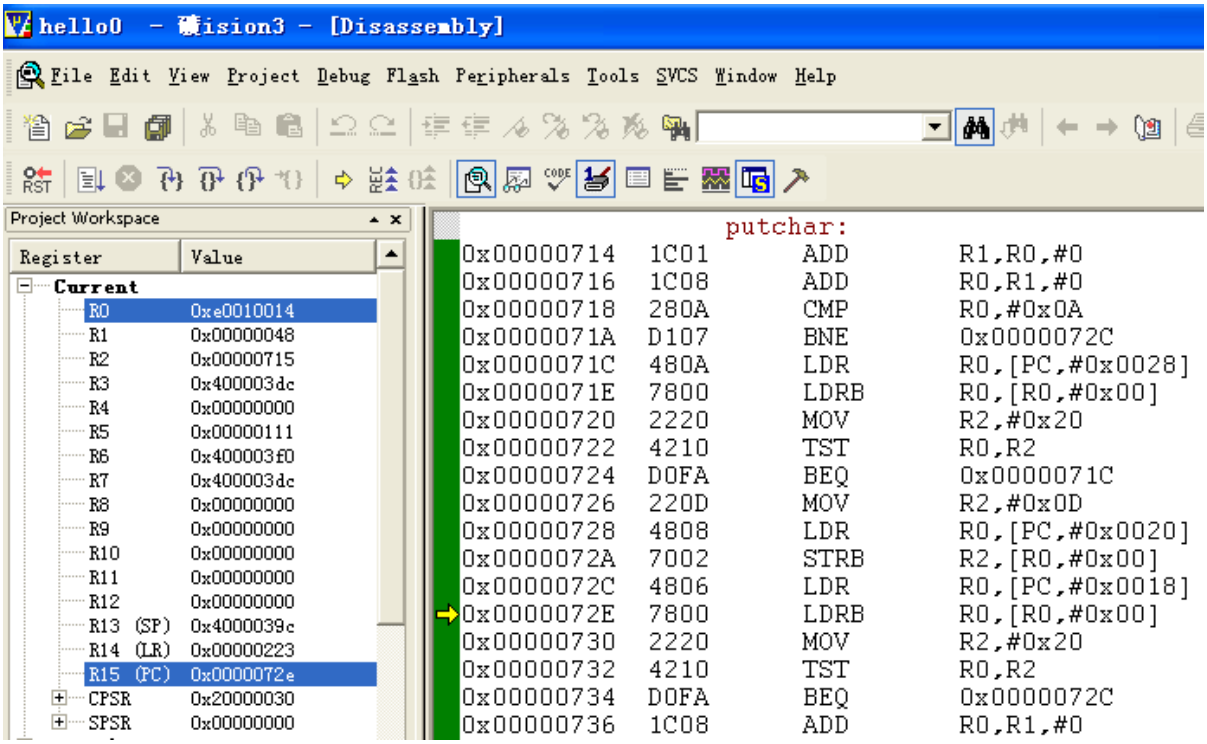
我们将代码中的 main 函数修改如下：

```
int main (void)  {                  /* execution starts here                  */

  /* initialize the serial interface   */
  PINSEL0 = 0x00000005;             /* Enable RxD0 and TxD0                   */
  U0LCR = 0x83;                     /* 8 bits, no Parity, 1 Stop bit          */
  U0DLL = 97;                       /* 9600 Baud Rate @ 15MHz VPB Clock       */
  U0LCR = 0x03;                     /* DLAB = 0                               */

  printf ("Hello World\n");         /* the 'printf' function call             */

  while (1) {                       /* An embedded program does not stop and  */
    ; /* ... */                     /* never returns.  We've used an endless  */
  }                                 /* loop.  You may wish to put in your own */
}                                   /* code were we've printed the dots (...). */
```

这样应该可以把字符输出到 UART0，可是你试一下就会发现不行，下面我们通过 Keil 的调试

功能查找原因。

启动 Keil 的调试功能 Debug > Start/Stop Debug Session，跟踪程序运行至 putchar：



当执行完 0x0000072C 后 R0 中是 UART1 的线状态 U1LSR 的地址 0xE0010014；执行完 0x0000073E 之后 R1 为 0xE0010000，而它正是 UART1 的接收缓冲寄存器 U1RBR(DLAB=0，只读)；UART1 的发送保持寄存器 U1THR(DLAB=0，只写)及 UART1 的除数缩存 LSB 寄存器 U1DLL(DLAB=1)。由此可知 printf 函数通过 putchar 函数向 UART1 发送数据，所以我们不能使用 printf 函数向 UART0 发送数据。putchar 函数是由 CARM 的 C 库提供的，我们修改较为困难，所以一般我们只使用 printf 函数向 UART0 发送数据。

使用 GCC 编译器的问题

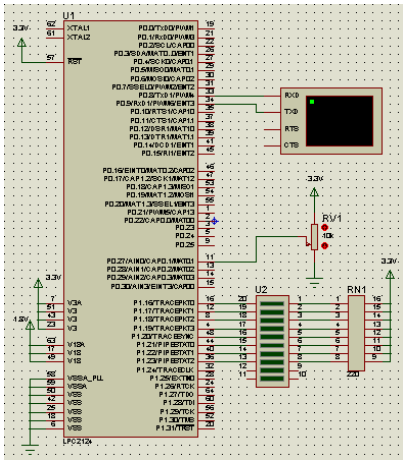
学过下面的实例 2 之后，你可能想回过头来使用 GCC 编译器编译实例 1，试过后你会发现，用 printf 函数，不论向 UART0 还是向 UART1 发送数据，都不会成功，因为你用上面介绍的方法跟踪一下程序的运行，你会发现在 GCC 中使用 printf 函数根本不会涉及到任何 UART 寄存器，当然也就不会在 UART 窗口中看到结果。

六、 Keil for ARM 实例 2：A/D 程序设计与电路仿真(使用 GCC 编译器)

向串口发送十六进制的 A/D 值，并控制 LED 闪烁。当 A/D 采集值大时，相邻两个 LED 闪烁间隔时间长；反之，相邻两个 LED 闪烁间隔时间短。

电路设计

在 Proteus 的 ISIS 中设计使用 ADC 的电原理图如下。

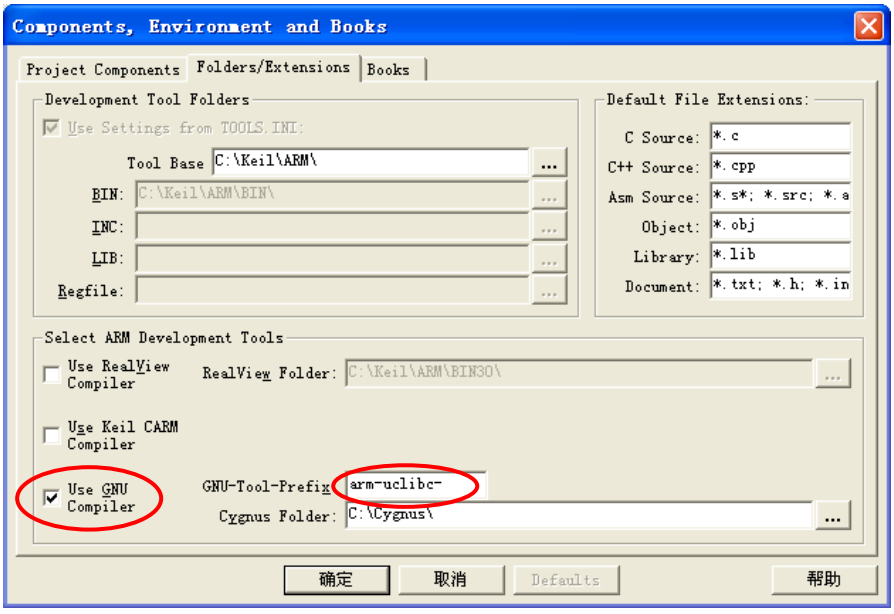


电路原理图

创建 Keil 项目

- 启动 Keil uVision3
- 设定编译器

(1) 在 Project Workspace 窗格的空白处右击并选择 Manage Components 命令，弹出 Components, Environment and Books 对话框。



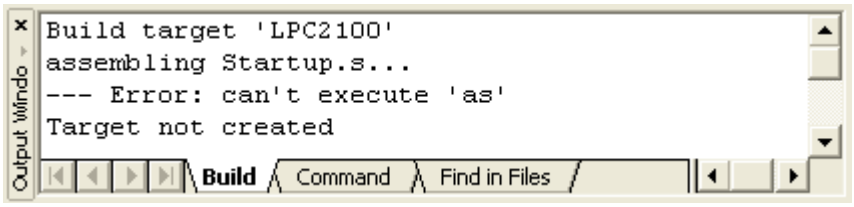
选择 GNU 编译器

在 Environment and Books 对话框中可以选择编译器、设置工具、库文件和 INC 文件等的路径及默认文件扩展名。

(2) 如图，选中 Use GNU Compiler 复选框，在 GNU-Tool-Prefix 文本框中输入 arm-uclibc-。

注意：C:\Cygnus\是 GCC for ARM 编译器的安装目录。

注意：如果不在 GNU-Tool-Prefix 文本框中输入 arm-uclibc-，编译时将出现如下错误信息：



编译时将出现错误信息

(3) 单击“确定”按钮保存设置。

- 建立 Keil 项目

(1) Project > New Project，显示 Create New Project 对话框，指定项目路径并在“文件名”文本框中输入文件名 adc。

(2) 单击“保存”按钮，显示 Select Device for Target ‘Target 1’ 对话框。在 Data base 窗格中选择 CPU 类型(Philips 公司的 LPC2124 芯片)，Description 窗格中显示该芯片的资源描述。

(3) 单击“确定”按钮，弹出提示信息，确认是否复制 LPC2100 启动代码到项目文件夹并添加文件到项目中。这里的启动代码是依据前面设置的编译器生成的。

(4) 单击“是”，添加启动代码（如果选择“否”，可以添加自己编写的启动代码）。这样将启动代码会添加到项目中，在 Project Workspace 窗格中显示出已添加的启动代码文件，双击 Startup.s 可查看该代码。

(5) 在工作空间 Project Workspace 中选择 Target 1，再单击该文本，修改为 LPC2100，并将 Source Group 1 修改为 Startup Code。在 Project Workspace 的空白处右击并选择 New Group 命令，将其名称修改为 Source Code；选中 Source Code，右击并选择 Add Files to Group ‘Source Code’命令，将 Serial.c 和 adc.c 文件添加到项目中。

相关的基础知识

请看 adc 目录下《A / D 程序设计》。

主文件代码

```
/*****************************************************************************/
/*  该文件是 uVision/ARM 开发工具的一部分                                    */
/*  Copyright KEIL ELEKTRONIK GmbH 2002-2004                                 */
/*****************************************************************************/
/*  ADC.C:  LED Flasher & Write A/D Conversion Value to Serial Interface     */
/*****************************************************************************/
#include <LPC21xx.H>                        /* LPC21xx 定义 */

extern void init_serial (void);             /* 初始化串口 */
extern int  putchar (int ch);               /* 写字符到串口 */
extern int  getchar (void);                 /* 从串口读字符 */

void puthex (int hex) {                     /* 发送十六进制数到串口 */
  if (hex > 9) putchar('A' + (hex - 10));   /*把大于 9 的十六进制数转换为 ASCII 码 */
  else putchar('0' +  hex);                 /* 把十六进制数转换为 ASCII 码 */
}
```

```
void putstr (char *p) {                     /* 发送字符串到串口 */
  while (*p) {
    putchar (*p++);
  }
}

void delay (void) {                         /* 延时 */
  unsigned int cnt;
  unsigned int val;

  ADCR |= 0x01000000;                       /* 启动 A/D 转换器 */
  do {
    val = ADDR;                             /* 读 A/D 数据寄存器的值 */
  } while ((val & 0x80000000) == 0);        /* 等待 A/D 转换结束 */
  ADCR &= ~0x01000000;                      /* 停止 A/D 转换。&=按位与，~ 1 的补(NOT) */
  val = (val >> 6) & 0x03FF;                /* 提取 AIN0 的值。 >> 右移，& 与(AND) */
                    /* 1111 1111 1100 0000 */
                    /* 0000 0011 1111 1111 */  /* AIN0 模拟信号输入 0 */
  putstr ("\nAIN0 Result = 0x");            /* 输出 A/D 转换结果 */
  puthex((val >> 8) & 0x0F);                /* 写最高位十六进制数 Write 1. Hex Digit */
                    /* 0000 1111 */
  puthex((val >> 4) & 0x0F);                /* 写中间位十六进制数 Write 2. Hex Digit */
  puthex (val & 0x0F);                      /* 写最低位十六进制数 Write 3. Hex Digit */

  val = (val >> 2) << 12;                   /* 调整延时值 */
  for (cnt = 0; cnt < val; cnt++);          /* 延时 Delay */
}

int main (void) {
  unsigned int n;

  IODIR1 = 0x00FF0000;          /* 定义 P1.16..23 为输出。这些是带内部上拉的标准 I/O 口。*/
           /* 0000 0000 1111 1111 0000 0000 0000 0000
  ADCR   = 0x002E0401;                      /* 设置 A/D: 10-bit AIN0 @ 3MHz */
  init_serial();                            /* 初始化串口 */
  IOSET1=0x00ff0000;
  while (1) {                               /* 无限循环 */
    for (n = 0x00010000; n <= 0x00800000; n <<= 1) { // 00000001 左移 8 次等于 10000000
      /* 0000 0000 1000 0000 0000 0000 0000 0000 */
      /* Blink LED 0, 1, 2, 3, 4, 5, 6, 7 */         // 以 16 进制看 01 左移 8 次等于 80
      IOCLR1 = n;                           /* 点亮 LED */
      delay();                              /* 延时 */
      IOSET1 = 0x00FF0000;                  /* 关闭 LEDs */
    }
  }
}
```

串口程序代码

```
/*****************************************************************************/
/*  该文件是 uVision/ARM 开发工具的一部分                                    */
/*  Copyright KEIL ELEKTRONIK GmbH 2002-2004                                 */
/*****************************************************************************/
/*  SERIAL.C:  Low Level Serial Routines                                     */
/*****************************************************************************/
#include <LPC21xx.H>                        /* LPC21xx 定义 */

#define CR     0x0D
```

```
void init_serial (void)  {                    /* 初始化串口 */
  PINSEL0 = 0x00050000;                       /* UART1 为输出(Enable RDX1 and TDX1) */
  U1LCR = 0x83;                               /* 8 位数据, 无奇偶校验, 1 位停止位 */
  U1DLL = 97;                                 /*  在 12 MHz VPB 时钟下，波特率为 9600 */
  U1LCR = 0x03;                               /* DLAB = 0 */
}
int putchar (int ch)  {                       /* 写字符到串口 */
  if (ch == '\n')  {
    while (!(U1LSR & 0x20));                  /* while 条件: 1 为 ture，&运算相等为 0。
    U1THR = CR;                               /* 输出 CR */
  }
  while (!(U1LSR & 0x20));
  return (U1THR = ch);
}
int getchar (void)  {                         /* 从串口读字符 */

  while (!(U1LSR & 0x01));

  return (U1RBR);
}
```

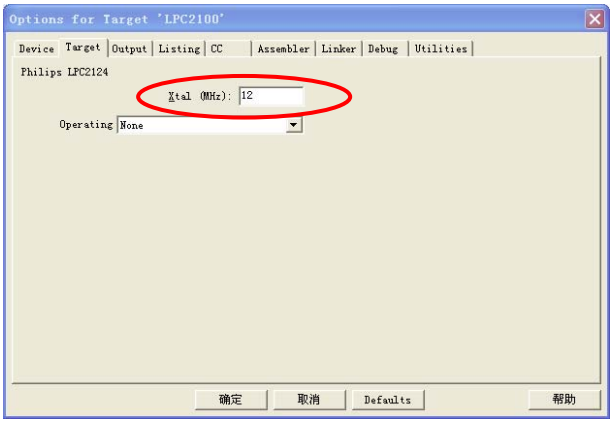
上面两段程序示范了怎样将一段程序分开写在两个文件中。要点如下：

- 调用另一个文件中的函数，需要先用 extern 关键字声明，例如

```
extern void init_serial (void);               /* 初始化串口 */
```

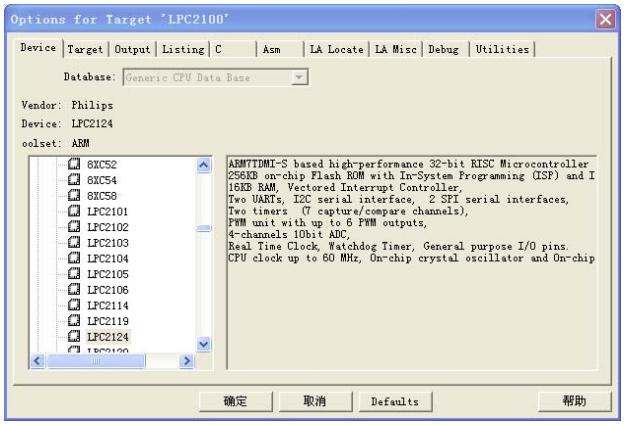
项目配置

(1) 在 Project Workspace 中选择 LPC2100，然后右击并选择 Options for Target ‘LPC2100’命令，在弹出的对话框中设置 Xtal 为 12MHz。



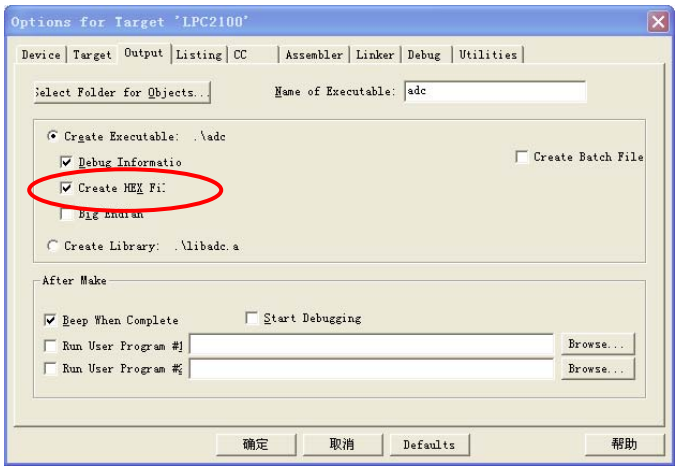
设置晶振频率

(2) 在上图中选择 Device 选项卡，显示 CPU 的类型及其相关的片资源，这是前面已经设定的 CPU 的类型，如下图所示。



显示 CPU 信息

(3) 选择 Output 选项卡，选中 Create HEX File 复选框，如图，可生成 hex 格式的可执行文件。单击 Select Folder for Objects 按钮，指定输出文件的路径。

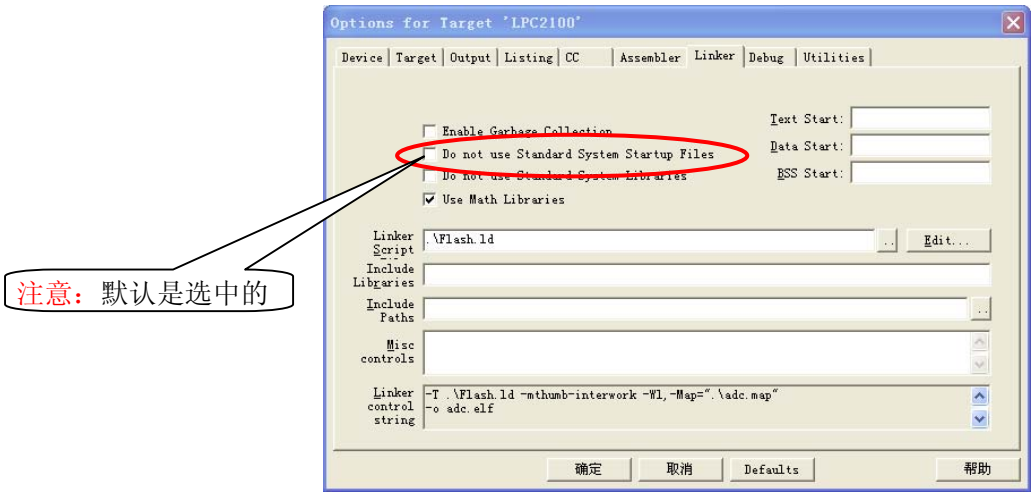


设置输出文件

(4) 选择 Listing 选项卡，再单击 Select Folder for Listings 按钮，指定生成的 list 文件的输出路径；然后选中 Assembler Listing 和 linker Listings 复选框。

输出列表文件

(5) 选择 Linker 选项卡，设定链接参数，指定 Linker Script 文件的路径，如图。单击 Edit 按钮可对 Flash.ld 文件进行编辑，这是一个文本文件。



设置链接文件

(6) 取消对 Do not use Standard System Startup Files 的默认选择。

(7) 其他标签保持默认设置，单击“确定”按钮保存设置。

Build 目标

在 Project Workspace 中选择 LPC2100，然后右击并选择 Build Target (F7) 命令，或单击快捷图标  ，编译并链接项目，输出窗口如图。

```
Build target 'LPC2100'
assembling Startup.s...
--- Error: can't execute 'arm-uclibc-as'
Target not created
```

显示编译结果

uVision3 支持 3 种不同的 ARM 开发工具(编译器)：Keil CARM，Keil RealView 和 GNU。如果编译中出现错误：CAN'T execute 'arm uclibc as'，说明你已经选择了使用 GNU 工具，而 GUN 工具没有安装。解决的方法有两个：

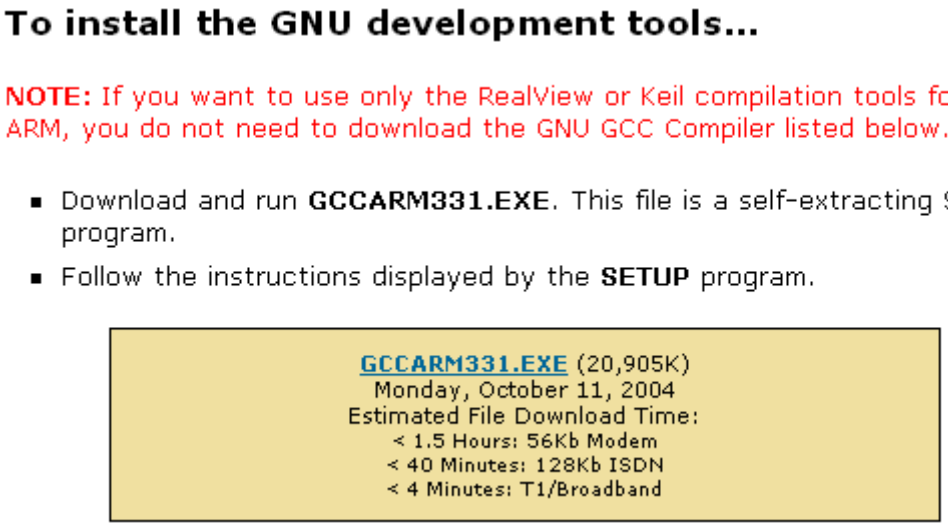
(1) 选择其他的开发工具，如 Keil CARM 编译器。从 Project - Components, Environment and Books 对话框中选择 Use Keil CARM Compiler，并且用 Keil CARM 版本的 STARTUP.S 文件代替 GNU 特定的 STARTUP.S 文件 (位于 \KEIL\ARM\STARTUP\ 文件夹中)。

(2) 如果要使用 GNU 工具集，你就必须安装 GNU 工具系列(toolchain)。可以从 www.keil.com/demo 下载 Keil ARM 评估工具。

我们采用第二中方法：

(1) 下载 GNU 工具

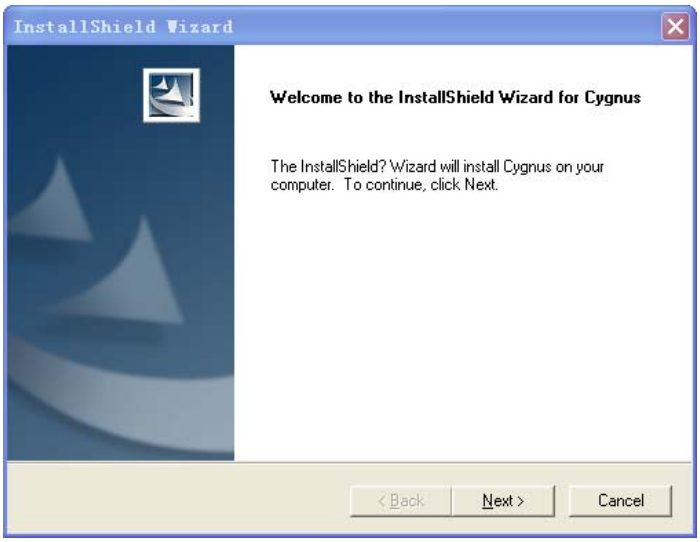
https://www.keil.com/demo/eval/arm.htm



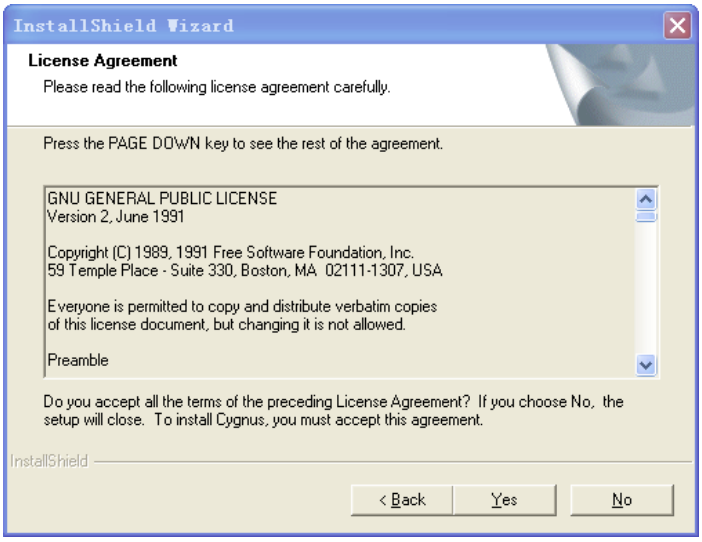
下载 GNU 开发工具

(2) 安装 GNU 开发工具

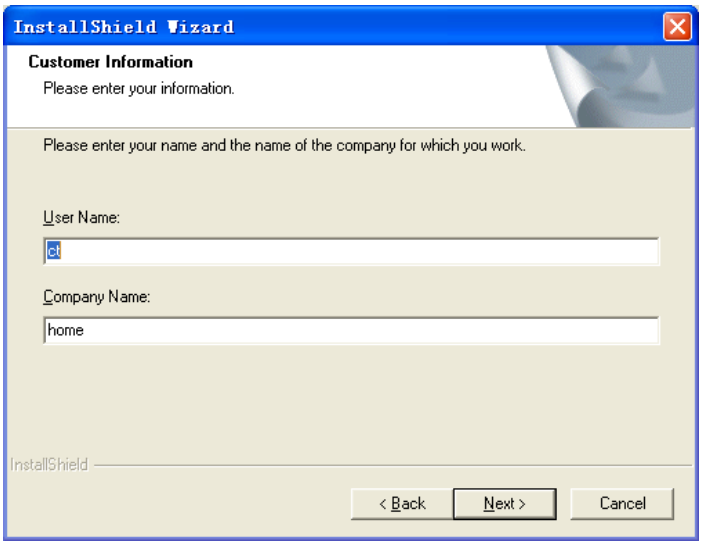
- 在本手册的配套光盘中找到 gccarm331.exe，双击，显示欢迎界面



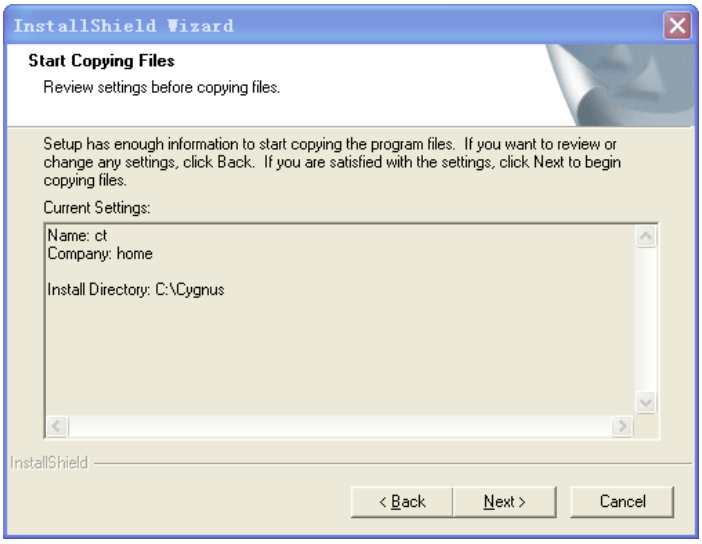
- Next，显示授权许可



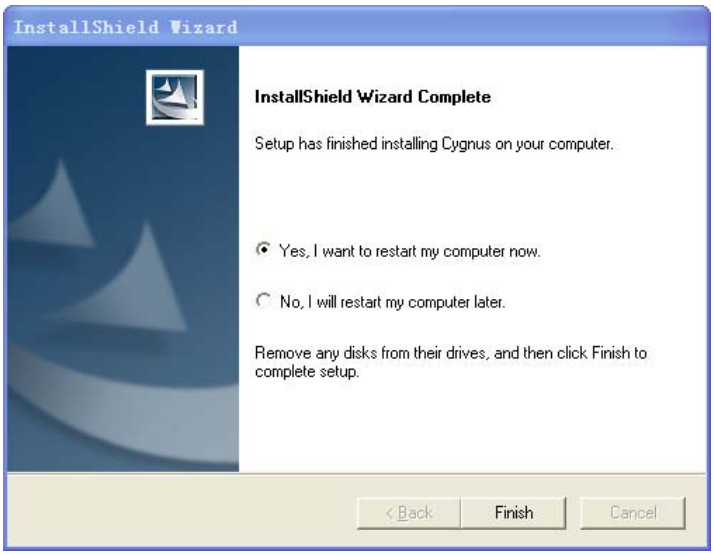
- 许可确认，Yes



- 输入用户名和公司名，Next



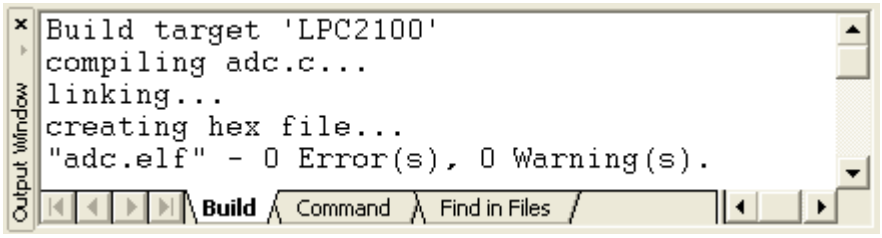
- Next，开始复制文件



- Finish，结束安装，重新启动计算机。

(3) 重新 Build 目标

在 Project Workspace 中选择 LPC2100，然后右击并选择 Build Target (F7) 命令，或单击快捷图标  ，编译并链接项目，输出窗口如图。



显示编译结果

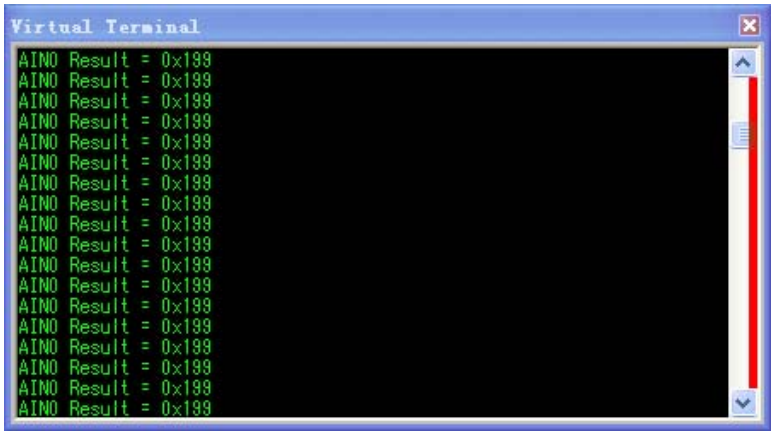
如果出现下面的错误：

请检查前面“设置链接文件”是否正确。

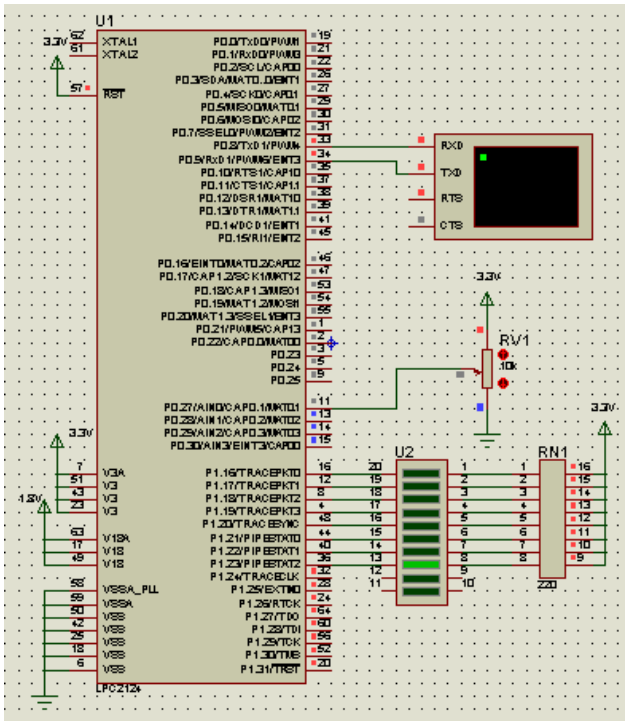
在 Proteus 中仿真

(1) 在电路图上双击 LPC2124，在弹出的 Edit Component 对话框中指定程序文件 adc.elf 的路径 (elf 文件包含调试信息和 bin 文件内容，所以 elf 文件比 hex 文件大许多，在正式场合使用 hex 文件)。

单击 Proteus 的仿真按钮  启动仿真，仿真结果如图。

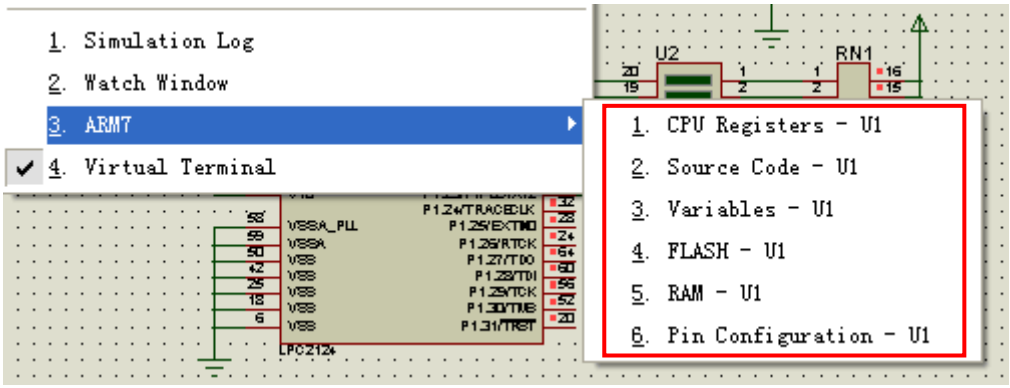


虚拟终端上的显示结果



仿真结果

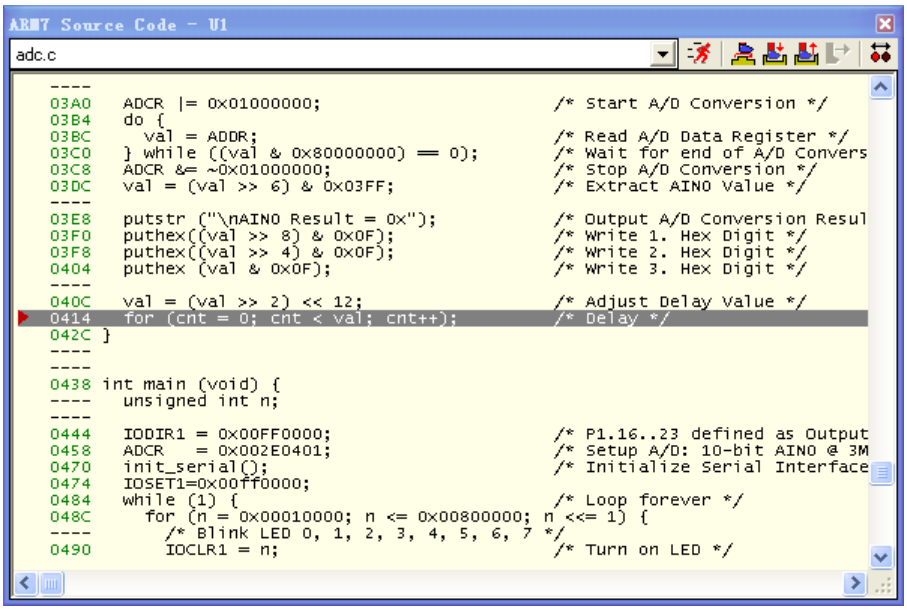
(2) 单击 ⏸ 按钮，暂停仿真，在 Debug 菜单中用 ARM7 命令可以查看 CPU 寄存器、源代码、变量、Flash、RAM 和引脚配置等信息，比用 CARM 编译器丰富多了。



Debug > ARM7 命令

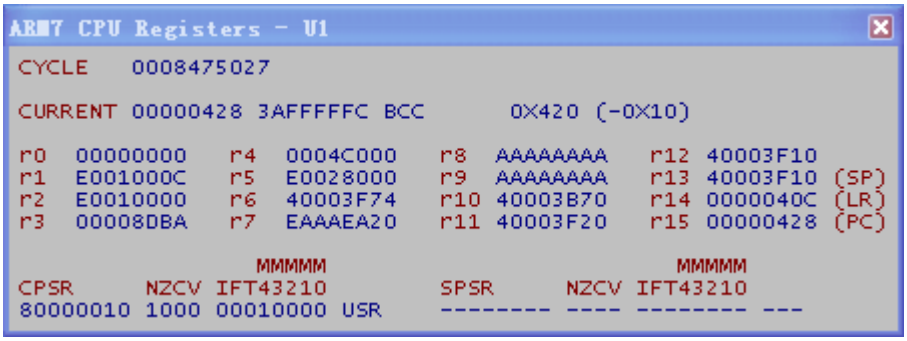
(3) 单击仿真按钮 ▶ 继续仿真，向下调整电位器使 A/D 采样值减小，LED 闪烁加速。

(4) 单击 ⏯ 按钮单步运行，在 ARM7 Source Code 窗口中可以查看程序执行的位置；右上角的按钮也可以控制仿真。仿真时，在 ARM7 Variables 窗口可以实时查看变量值和 CPU 寄存器的变化。

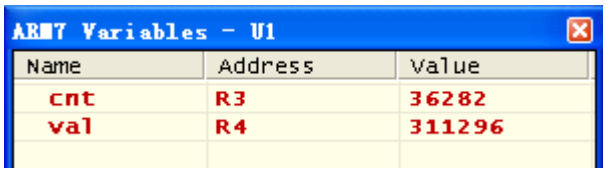


显示源代码的单步调试

注意：单步调试功能只能用 GCC 编译才能实现。



寄存器信息



变量信息

(5) 单击  按钮，停止仿真。

单步断点跟踪调试

在 ARM7 Source Code 窗口中使用下面几个按钮就可以进行单步断点跟踪调试。

按钮的功能按顺序分别是：全速、越过、步入、步出、全速执行到下一断点处。

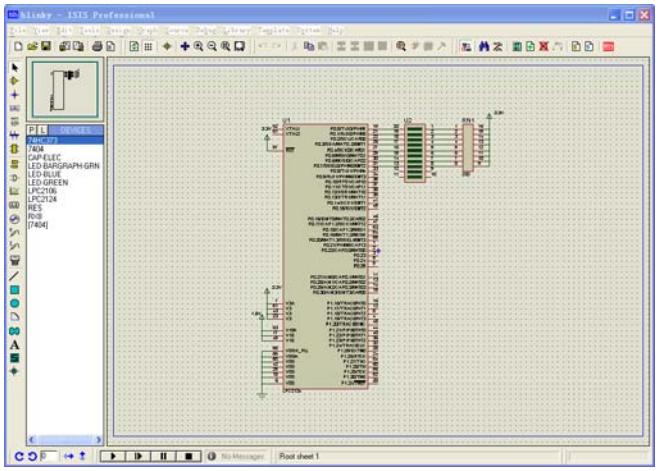
单步断点跟踪调试功能与其他软件类似，学过 C 语言的同学应该会做。

七、 Keil for ARM 实例 3：GPIO 程序设计与电路仿真(使用 GCC 编译器)

使用 P0.0~P0.7 的输出功能来控制 LED 闪烁。采用灌电流的方式驱动 LED 即输出低电平时 LED 点亮。首先进行 IO0DIR 寄存器设置，使 P0.0~P0.7 为输出模式，通过对 IO0SET 和 IO0CLR 寄存器进行口线置 1 或清 0 来控制 LED 闪烁。延时程序采用定时器中断方式。

电路设计

在 Proteus 的 ISIS 中设计使用 GPIO 的电原理图如下。

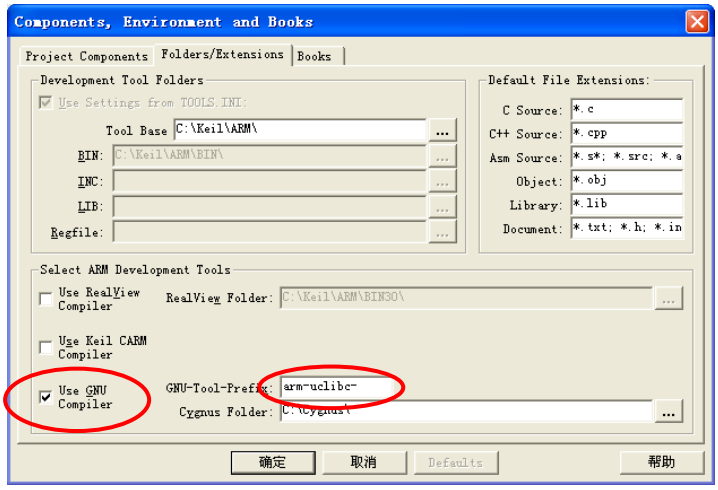


电路原理图

创建 Keil 项目

- 启动 Keil uVision3
- 设定编译器。Keil 将依据不同的编译器生成不同的启动代码。

(1) Project > Manage > Components, Environment, Books...或单击图标，弹出 Components, Environment and Books 对话框。



选择 GNU 编译器

在 Environment and Books 对话框中可以选择编译器、设置工具、库文件和 INC 文件等的路径及默认文件扩展名。

(2) 如图，选中 Use GNU Compiler 复选框，在 GNU-Tool-Prefix 文本框中输入 arm-uclibc-(早期的版本需要手工输入)。

(3) 单击“确定”按钮保存设置。

- 建立 Keil 项目

(1) Project > New Project，显示 Create New Project 对话框，指定项目路径并在“文件名”文本框中输入文件名 Blinky。

(2) 单击“保存”按钮，弹出 Select Device for Target ‘Target 1’ 对话框。在 Data base 窗格中选择 CPU 类型(Philips 公司的 LPC2124 芯片)，Description 窗格中显示该芯片的资源描述。

(3) 单击“确定”按钮，弹出提示信息，确认是否复制 LPC2100 启动代码到项目文件夹并添加文件到项目中。启动代码 Startup.s 是依据前面设置的编译器生成的。

(4) 单击“是”，添加启动代码（如果选择“否”，可以添加自己编写的启动代码）。这样将启动代码会添加到项目中，在 Project Workspace 窗格中显示出已添加的启动代码文件，双击 Startup.s 可查看该代码。

(5) 在工作空间 Project Workspace 中选择 Target 1，再单击该文本，修改为 LPC2100，并将 Source Group 1 修改为 Startup Code。右击 LPC2100 并选择 New Group 命令，将其名称修改为 Source Code；选中 Source Code，右击并选择 Add Files to Group ‘Source Code’ 命令，将 Time.c 和 Blinky.c 文件添加到项目中。

注意：项目文件夹中应包含头文件 Timer.h

相关的基础知识

请看 blink 目录下《GPIO 程序设计》。

主文件代码

```
/*****************************************************************************/
/*  该文件是 uVision/ARM 开发工具的一部分                                    */
/*  Copyright KEIL ELEKTRONIK GmbH 2002-2004                                 */
/*****************************************************************************/
/*  BLINKY.C:  LED Flasher                                                   */
/*****************************************************************************/
#include <LPC21xx.H>                        /* LPC21xx 定义 */
#include "Timer.h"

extern long volatile timeval;  // extern 是对外的一个引用。变量 timeval 在 TIME.C 中声明
/*****************************************************************************
**函数名称: wait()
**描述: 延时
*****************************************************************************/
void wait (void)  {                         /* 延时函数 */
  unsigned long i;

  i = timeval;
  while ((i + 10) != timeval);              /* 延时 100ms */
}
/*****************************************************************************
**函数名称: main()
**描述: 主程序
*****************************************************************************/
int main (void) {
```

```
  unsigned int j;                              /* LED 变量 */
  IO0DIR = 0x0000FF;                           /*P0.0..P0.7 设为输出 */
  init_timer();
  IO0SET=0x0000FF;
  while (1)  {                                 /* 无限循环 */
    for (j = 0x000001; j < 0x000080; j <<= 1) { /* 闪烁 LED 0,1,2,3,4,5,6 */
      IO0CLR = j;                              /* 点亮 LED */ // 01 左移 8 次等于 80
      wait ();                                 /* 调用延时函数 */
      IO0SET = j;                              /* 关闭 LED */
    }
    for (j = 0x000080; j > 0x000001; j >>=1 ) { /* 闪烁 LED 7,6,5,4,3,2,1 */
      IO0CLR = j;                              /* 点亮 LED */
      wait ();                                 /* 延时 */
      IO0SET = j;                              /* 关闭 LED */
    }
  }
}
```

定时程序代码

```
/*****************************************************************************/
/*  该文件是 uVision/ARM 开发工具的一部分                                    */
/*  Copyright KEIL ELEKTRONIK GmbH 2002-2004                                 */
/*****************************************************************************/
/*  TIME.C:  Time Functions for 100Hz Clock Tick                              */
/*****************************************************************************/
#include <LPC21XX.H>                          // LPC21XX 外设寄存器
#include "Timer.h"
long timeval;
void tc0 (void) __attribute__ ((interrupt));    // 产生中断
/* 设置定时/计数器(Timer Counter) 0 中断 */
void init_timer (void) {                  // 定时器 0 初始化
   T0MR0 = 149999;                        // 匹配值为 10ms (10mSec = 150.000-1 counts)
   T0MCR = 3;                             // 在 MR0 匹配(计数满)时产生中断并复位 T/C
   T0TCR = 1;                             // 定时器 0 使能；既启动定时器
   VICVectAddr0 = (unsigned long)tc0;          // 设置中断服务程序的地址
   VICVectCntl0 = 0x20 | 0x04;                 // 设置定时器 0 中断控制
   VICIntEnable = 0x00000010;                  // 使能定时器 0 中断
}
/* 定时/计数器(Timer Counter - TC) 0 中断每 10ms 执行一次 (在 CPU 时钟频率为 60MHz) */
void tc0 (void) {
   timeval++;
   T0IR = 0x01;                                    // 清除中断标志
   VICVectAddr = 0x00;                             // 中断应答
}
```

项目配置

(1) 在 Project Workspace 中选择 LPC2100，然后右击并选择 Options for Target ‘LPC2100’命令，在弹出的对话框中设置 Xtal 为 12MHz。

(2) 在对话框中选择 Device 选项卡，显示 CPU 的类型及其相关的片资源，这是前面已经设定的 CPU 的类型。

(3) 选择 Output 选项卡，选中 Create HEX File 复选框，可生成 hex 格式的可执行文件。单击 Select Folder for Objects 按钮，指定输出文件的路径。

(4) 选择 Listing 选项卡，再单击 Select Folder for Listings 按钮，指定生成的 list 文件的输出路径；然后选中 Assembler Listing 和 linker Listings 复选框。

(5) 选择 Linker 选项卡，设定链接参数，指定 Linker Script 文件 Flash.ld 的路径。单击 Edit

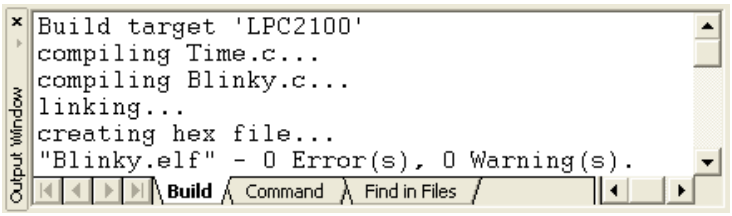
按钮可对 Flash.ld 文件进行编辑，这是一个文本文件。

注意：去掉 Do not use Standard System Startup Files 复选项。

(6) 其他标签保持默认设置，单击“确定”按钮保存设置。

Build 目标

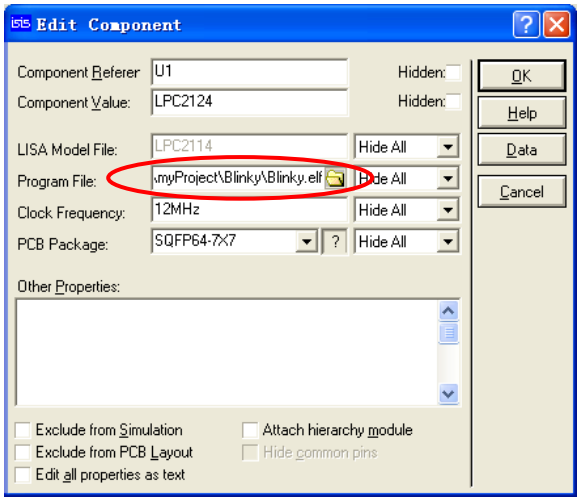
在 Project Workspace 中选择 LPC2100，然后右击并选择 Build Target (F7) 命令，或单击快捷图标  ，编译并链接项目，输出窗口如图。



显示编译结果

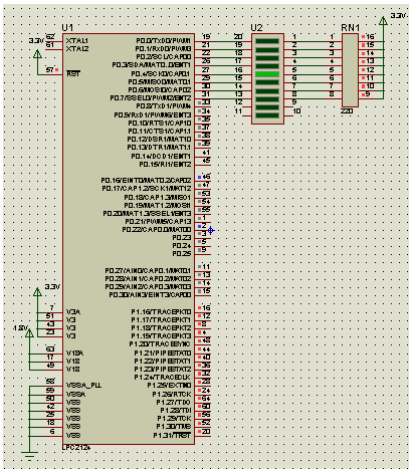
Protues 仿真

(1) 右击电路图中 U1：LPC2124，然后单击弹出 Edit Component 对话框，在 Program File 文本框中指定程序文件 Blinky.elf 的路径，如下图所示。



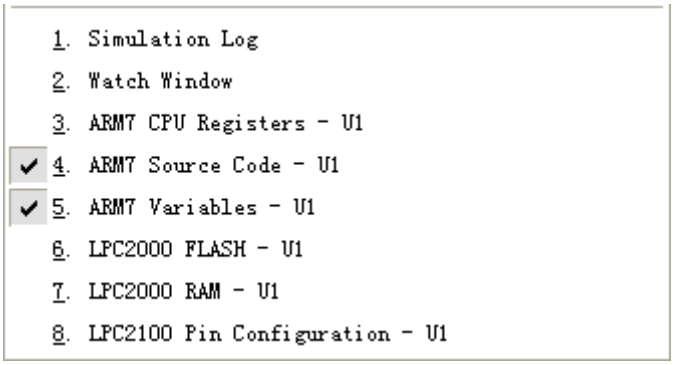
指定可执行文件

(2) 单击  按钮运行仿真，仿真结果如下图所示，LED 循环显示。

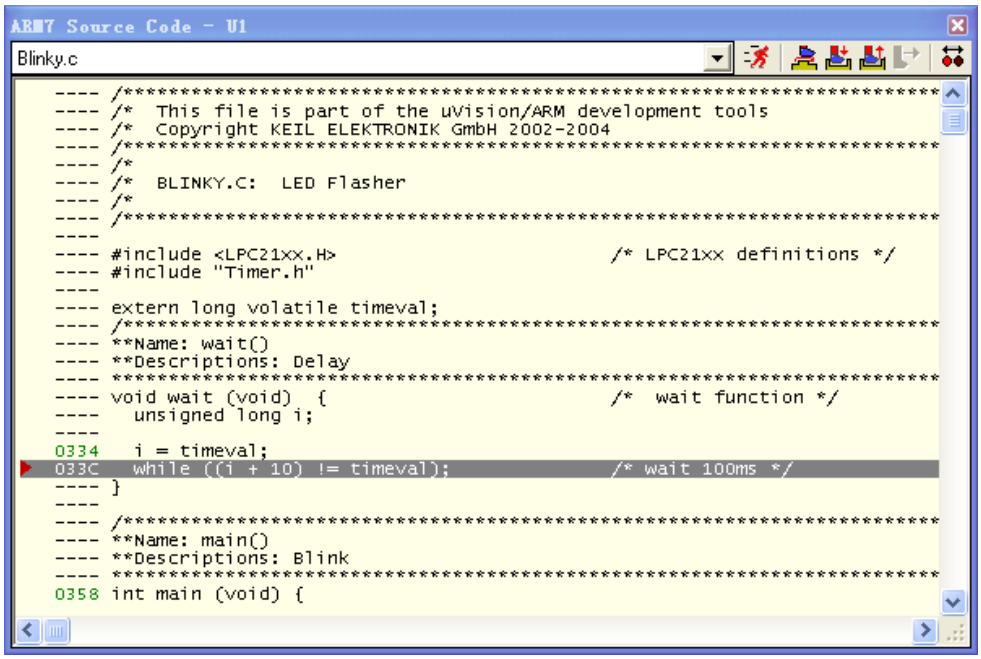


仿真结果

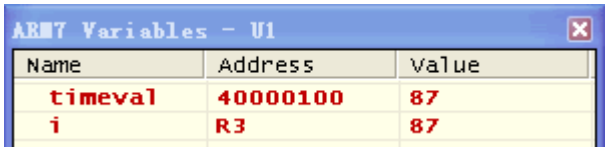
(3) 单击 ⏸ 按钮，暂停仿真，在 Debug 菜单中可以选择查看 CPU 寄存器、源代码、变量、Flash、RAM 和引脚配置等信息。



Debug 菜单下可查看的信息

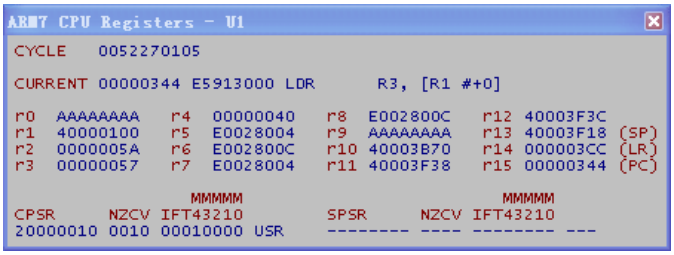


ARM7 Source Code 窗口：显示源代码



变量信息

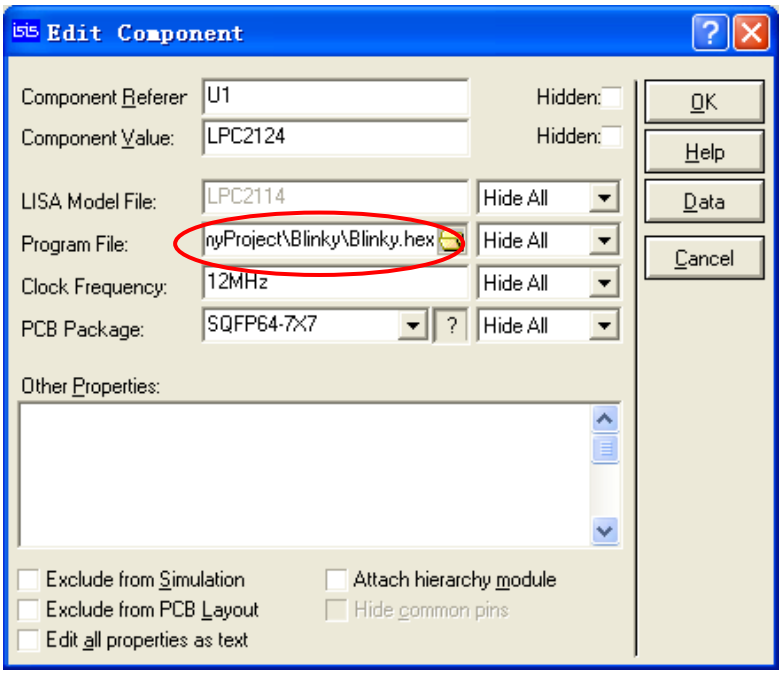
(4) 单击 ⏯ 按钮单步运行，在 ARM7 Source Code 窗口中可以查看程序执行的位置；右上角的按钮也可以控制仿真。仿真时，在 ARM7 Variables 窗口可以实时查看变量值，选择 3. ARM7 CUP Reisters - U1 命令，可以查看 CPU 寄存器及反汇编代码等信息。



CPU 寄存器及反汇编代码等信息

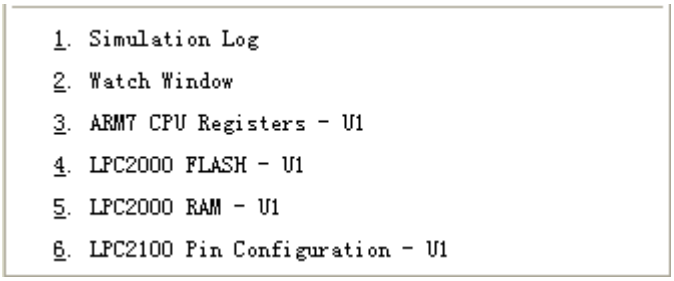
(5) 单击 ■ 按钮，停止仿真。

(6) 右击 U1：LPC2124，然后单击弹出 Edit Component 对话框，在 Program File 文本框中重新指定程序文件 Blinky.hex 的路径，如下图所示。



指定可执行文件

(7) 单击 ▶ 运行仿真程序；单击 ⏸ 按钮，暂停仿真，在 Debug 菜单中可以选择查看 CPU 寄存器、Flash、RAM 和引脚配置等信息。



Debug 菜单下可查看的信息

比使用 elf 文件少了源代码查看和变量查看。这是因为 elf 文件包含调试信息和 hex 文件内容。

(8) 单击 ■ 按钮，停止仿真。

第三方元件

这里介绍一下如何利用别人已做好的元件。其实很简单，仿真模型提供者一般会给出三样东西：模型文件（一般为 dll 文件）、例子、库文件。我们要做的工作是：先把 dll 文件拷贝到 Proteus 安装目录下的 MODELS 文件夹里，这样附带的例子就可以运行了！如果还附带库文件的话，就把 lib 文件拷贝到 Proteus 安装目录下的 LIBRARY 文件夹里，这样你就可以在 Proteus 的库管理器中看到该库文件。如果没有附带库文件，你就要把它添加到你自己的库里面，方法下面介绍。

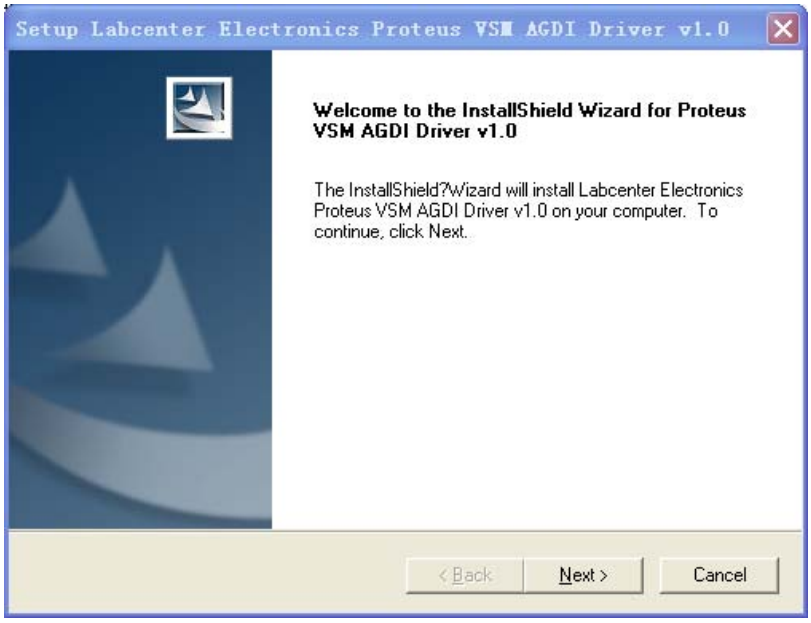
先到 http://www.callbus.ru/models.html 下载 I2C Spy 和 DS1621，我们的目标是 I2C Spy，但它的例子中要用到 DS1621，所以也把 DS1621 下了。于是，我们得到下面文件：I2CSPY.dll、DS1621.dll 和 test_i2c.DSN 等文件。先把 I2CSPY.dll 和 DS1621.dll 文件拷贝到 Proteus 安装目录下的 MODELS 文件夹里，运行 test_i2c.DSN。

八、 Keil 与 Proteus 整合的电路仿真

Proteus VSM (Virtual System Modelling) AGDI 驱动程序支持将 Proteus VSM 作为 Keil 集成开发环境的一个调试器插件。驱动程序必须安装在运行 Keil IDE 的机器上，而 Proteus 可以安装在 Keil IDE 运行的机器上，也可以安装在另一台机器上。

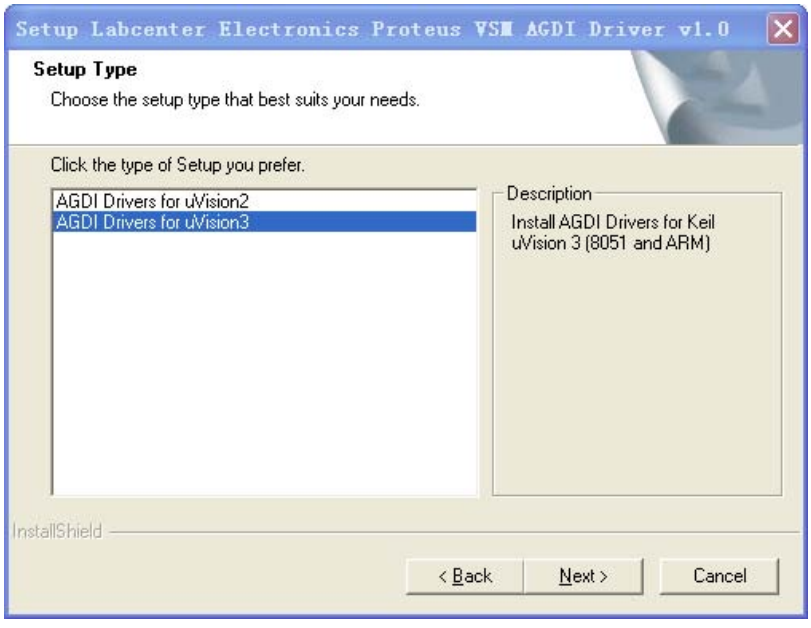
安装 Proteus VSM AGDI 驱动程序

(1) 在本手册的配套光盘中找到 vdmagdi.exe，双击运行，显示欢迎窗口

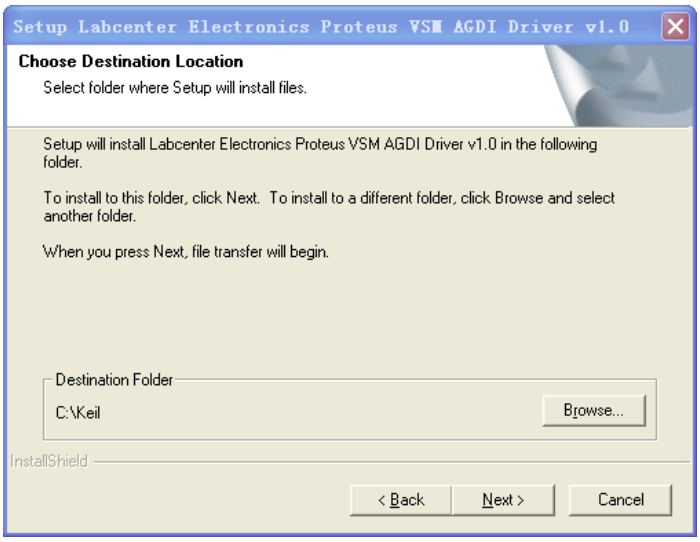


(2) Next

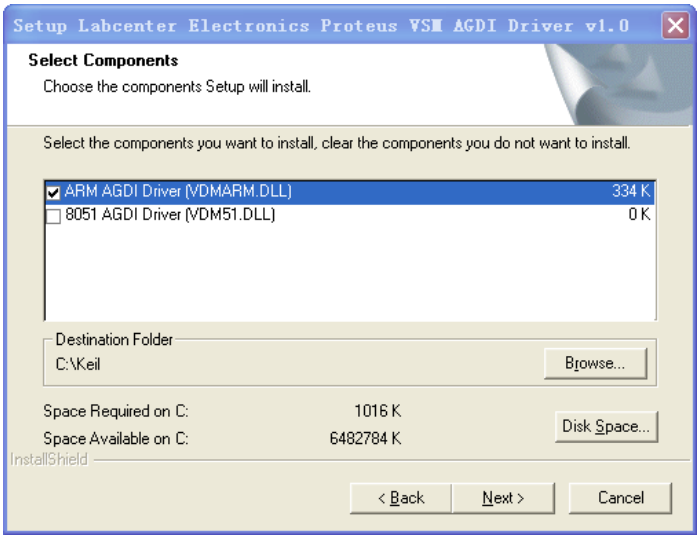
Keil 定义了一个接口叫做 AGDI 接口。 用户通过该接口的 dll 来连接你的仿真机和 keil 的 IDE。从下面的安装说明可知，AGDI Drivers for uVision2 只能用于仿真 8051；而 AGDI Drivers for uVision3 是用于仿真 8051 和 ARM 的。



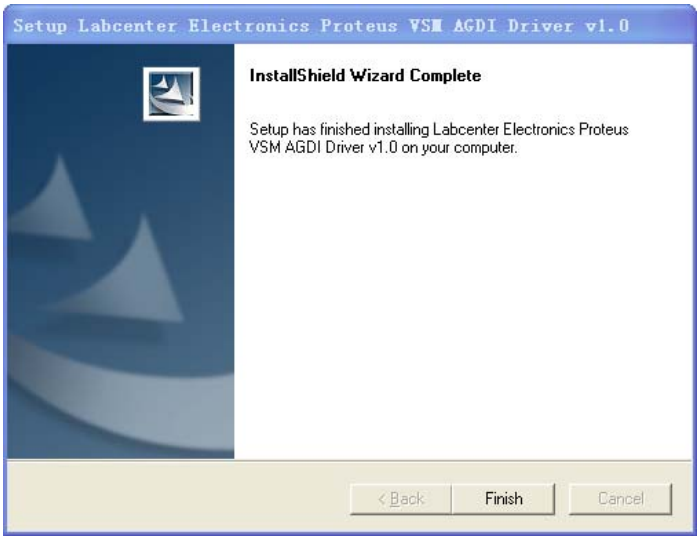
(3) 选择版本，Next



(4) 选择目的文件夹，Next



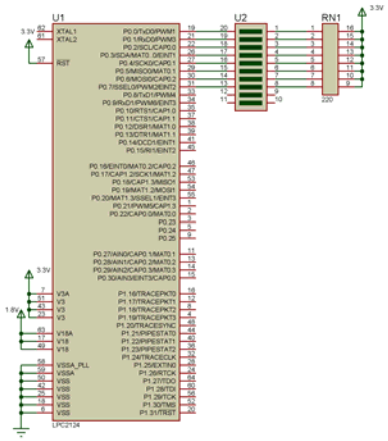
(5) 选择组件，Next



(6) 等待安装完成，Finish

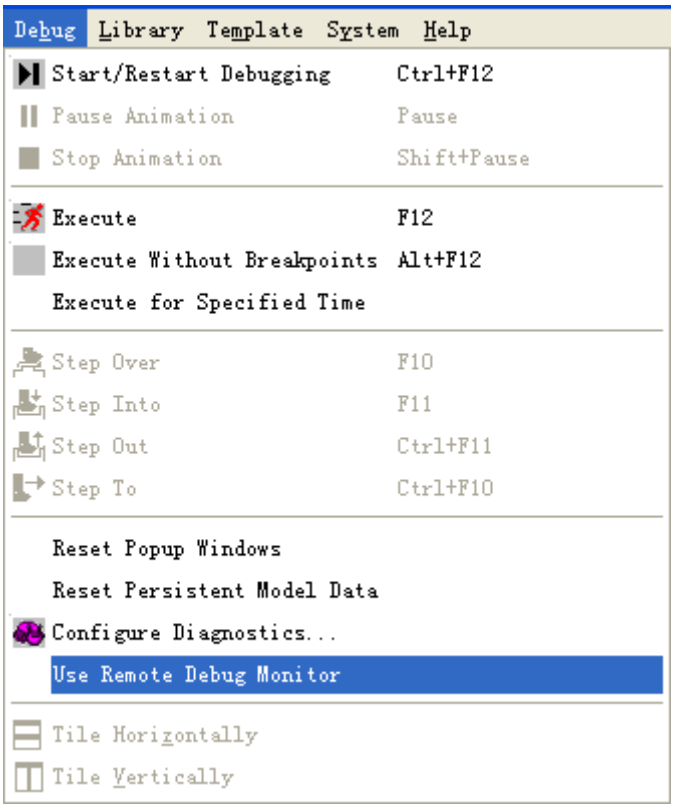
单机上整合 Kiel 与 Proteus

(1) 在 Proteus ISIS 中打开上一章的 Blinky 电路图。



仿真电路

(2) 在 ISIS 中 Debug > Use Remote Debug Monitor 命令。

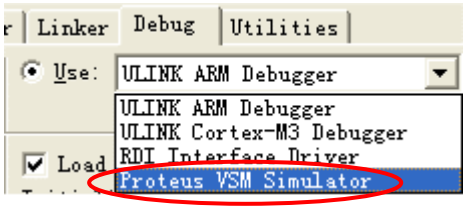


设置远程调试

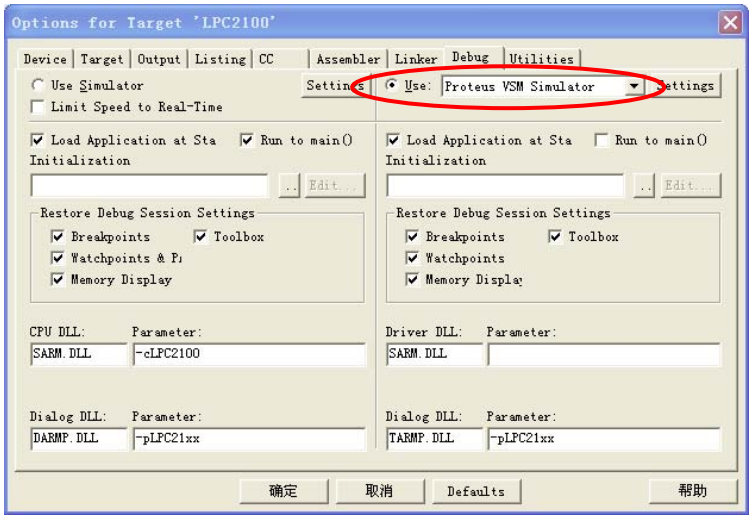
注意：通常 Use Remote Debug Monitor 功能应该关闭。

(3) 重新启动 Keil，在 Keil IDE 中打开项目 Blinky.uv2；在工作空间 (Project Workspace) 中选择 LPC2100，右击并选择 Options for Target 'LPC2100'命令，在弹出的 Options for Target 'LPC2100'对话框中选择 Debug 选项卡，在右侧选中 Use 复选框，并在其后的下拉列表框中选择 Proteus VSM Simulator。

注意：选项 Proteus VSM Simulator 只有在安装了“Proteus VSM AGDI 驱动程序”后才会出现。



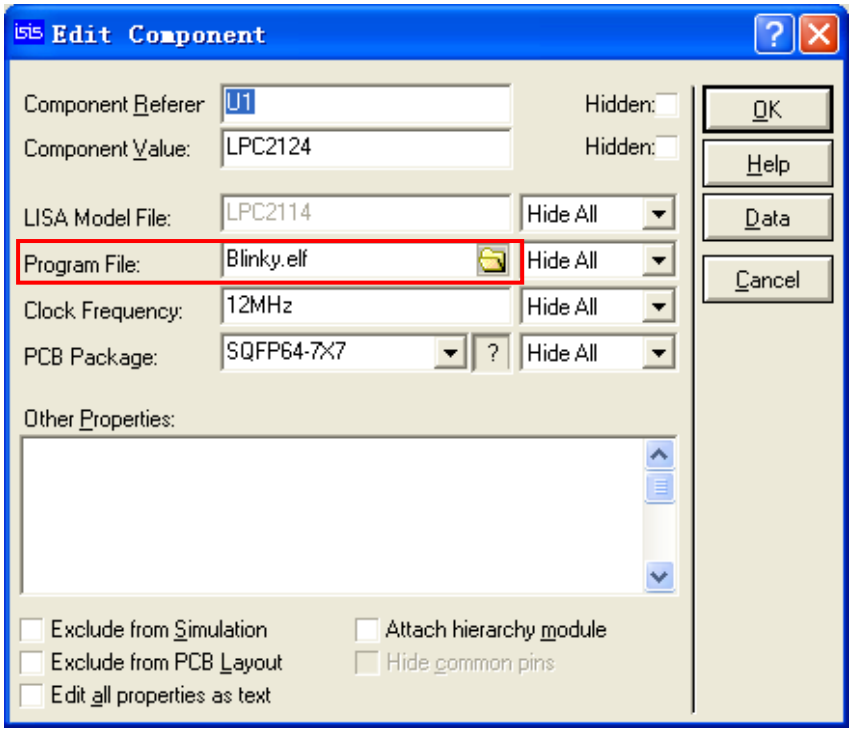
选择 roteus VSM Simulator



设置调试器

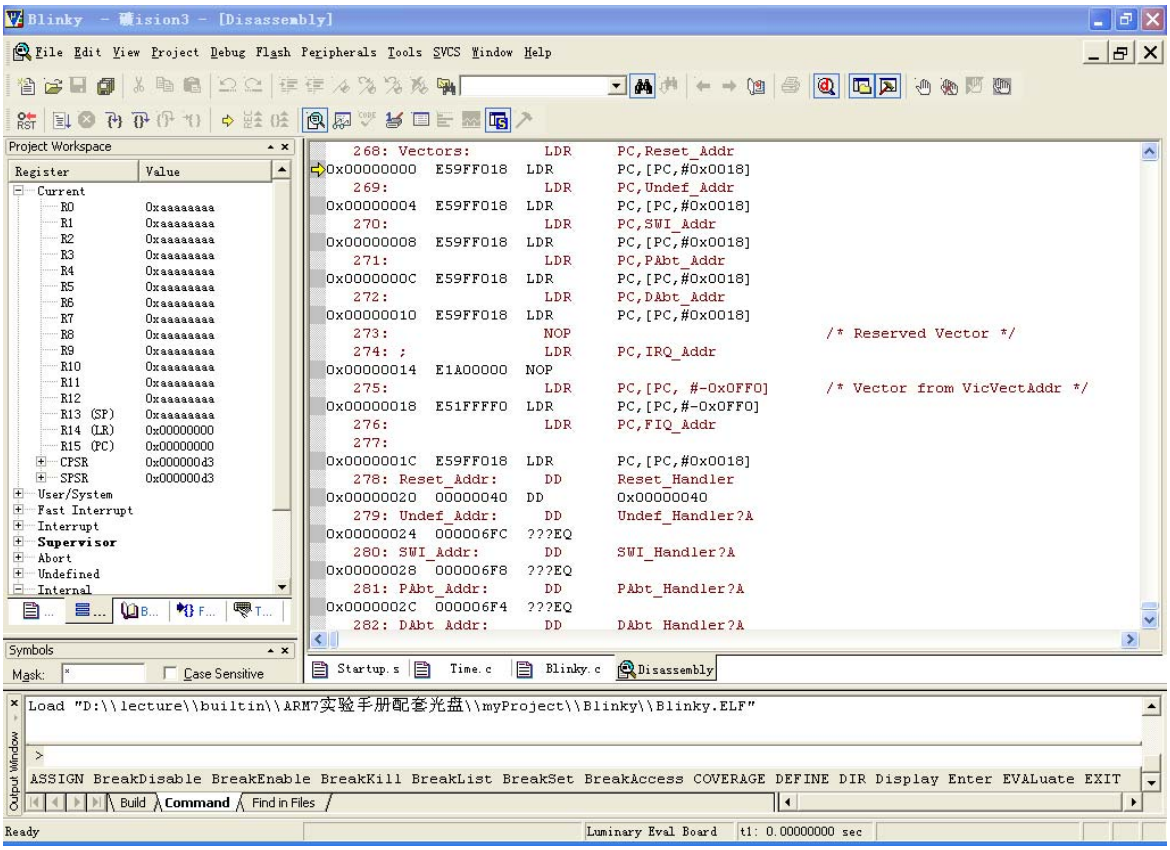
(4) 单击“确定”按钮保存设置。

(5) 回到 ISIS，在电路图中双击 LPC2124，弹出 Edit Component 对话框，在 Program File 文本框中指定程序文件 Blinky.elf 的路径，如下图所示。点 OK。

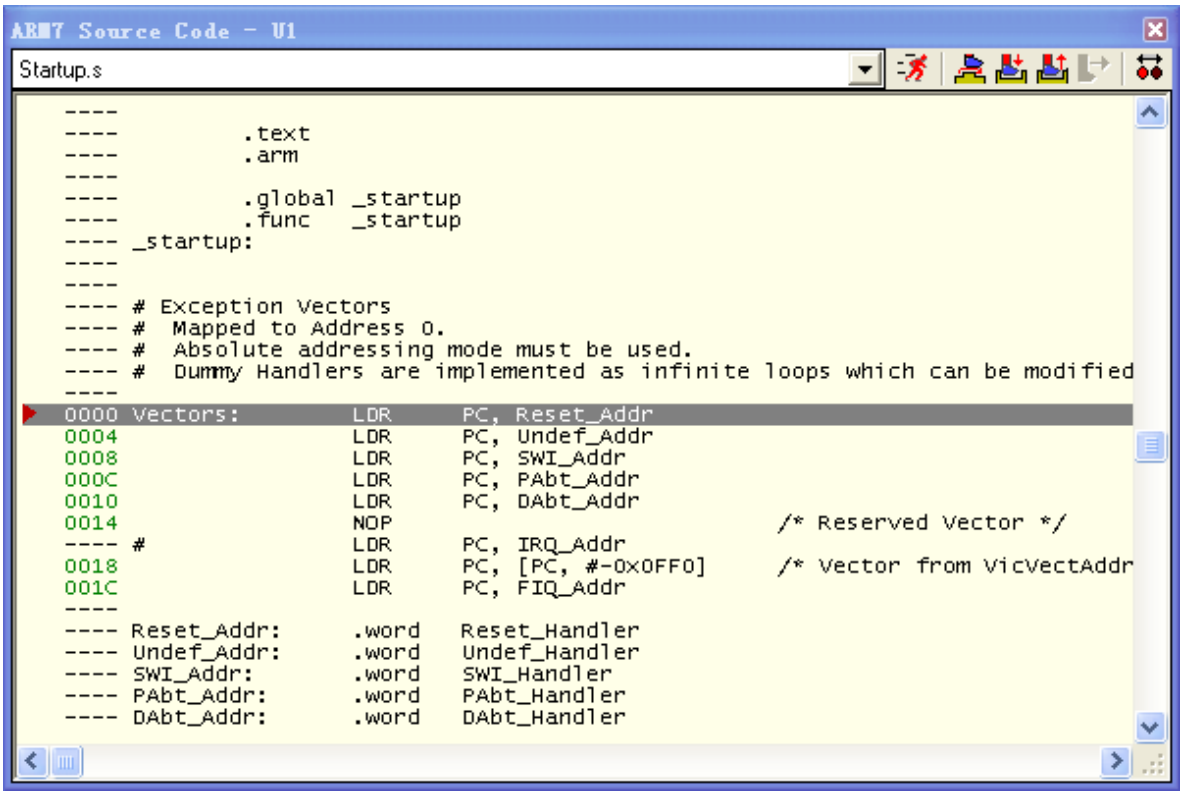


指定目标程序文件

(6) 再到 Keil 中选择 Debug > Start/Stop Debug Seccsion 命令，加载程序到 LPC2124 中，调试界面如下。Proteus 中的 ARM7 Source Code 窗口显示启动代码。

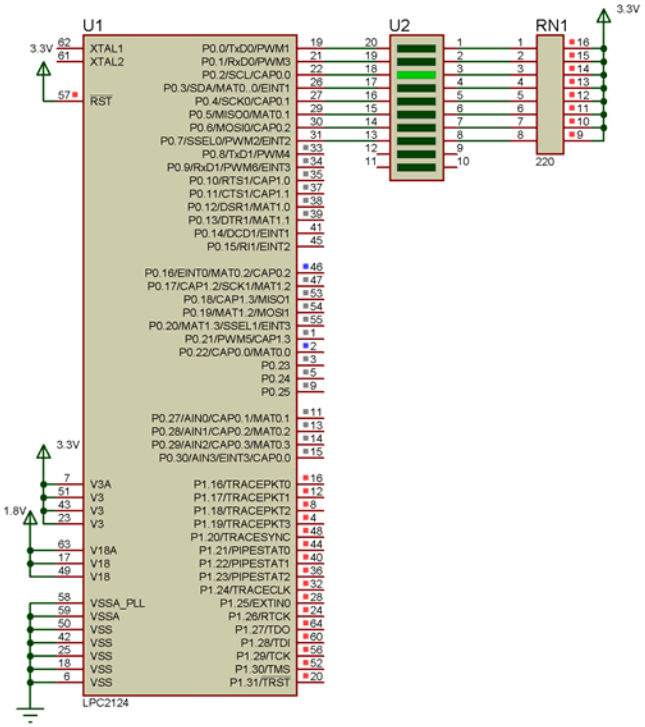


调试界面



显示启动代码

(7) 在 Keil 的调试界面下，选择 Debug > Run 命令或按下 F5 键，全速运行仿真。结果如图。

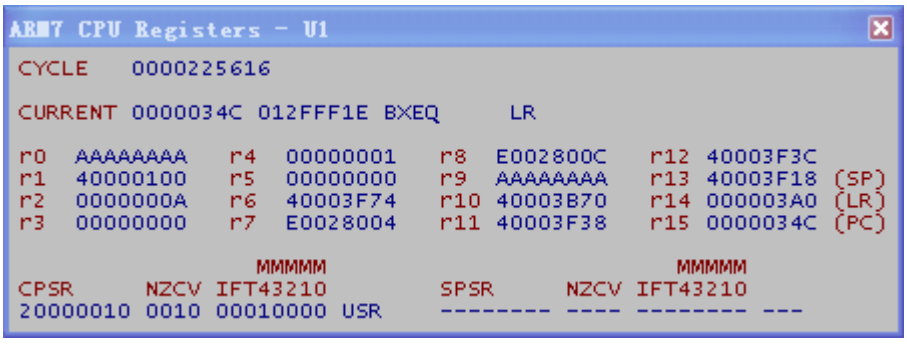


仿真结果

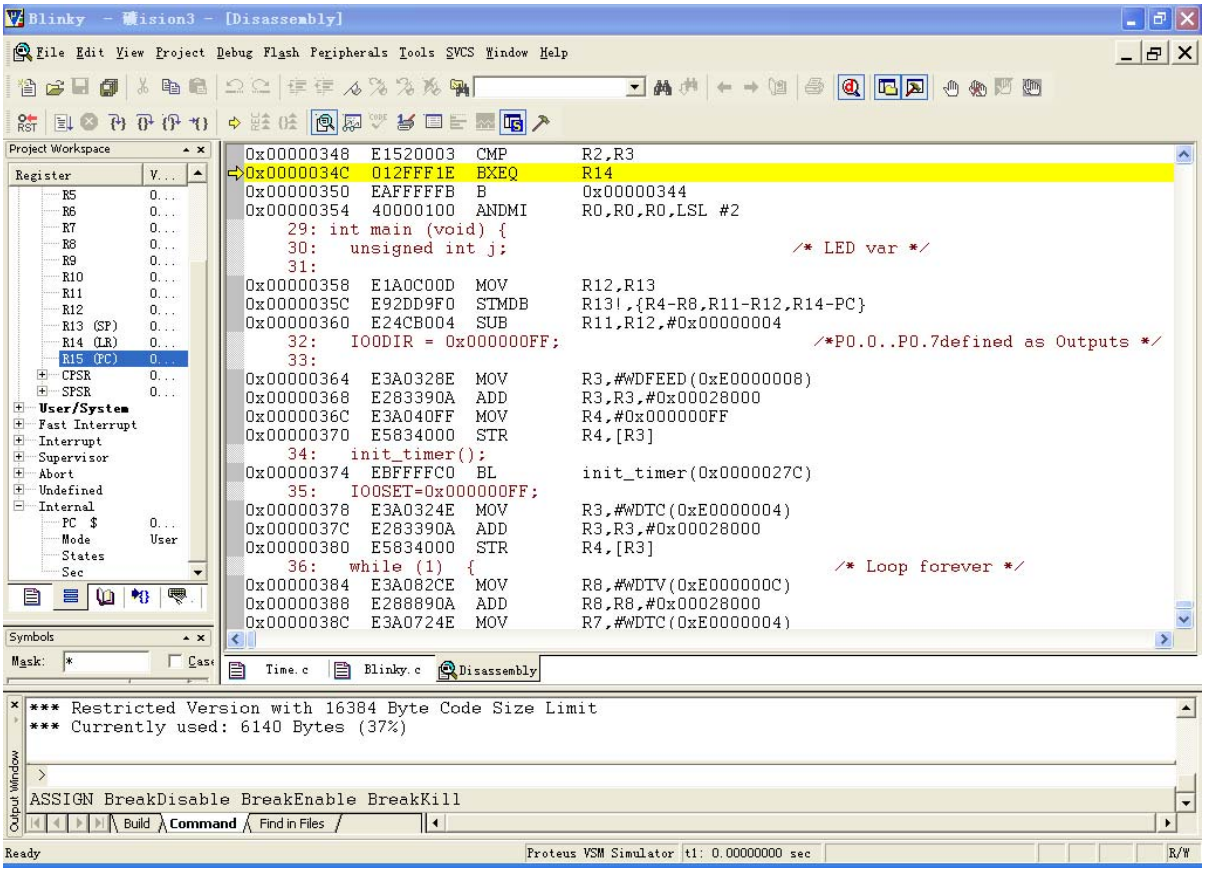
(8) 选择 Debug > Stop Running 命令，停止仿真。

(9) 先选择 Debug > Start/Stop Debug Seccsion 命令，再选择 Debug > Step 命令或按下 F11 键，进行单步调试。查看 Proteus 软件中电路也执行单步运行，并且出现 ARM7 Source Code 窗口和 ARM7 Variables 窗口，显示程序运行的位置。

(10) 在 Proteus 中选择 Debug > 3.ARM7 CPU Registers – U1 命令，弹出 CPU 寄存器窗口。可以看到现在运行的反汇编代码为 BXEQ R1，而 Keil 中也显示相同的反汇编代码，如图。



CPU 寄存器窗口



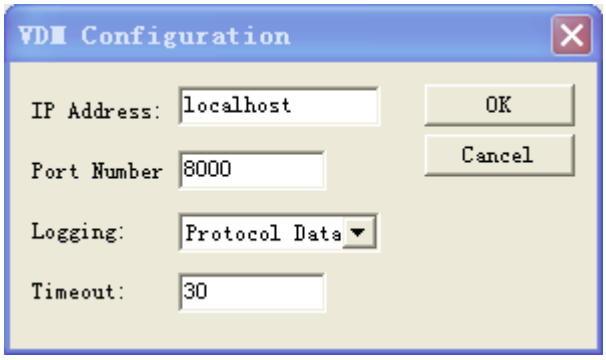
反汇编代码

(11) 选择 Debug > Start/Stop Debug Session 命令，停止仿真。

网络上整合 keil 与 Protues

通过 TCP/IP 协议。Proteus VSM AGDI 驱动程序能够与 ISIS 通信，这样 Keil 就能够控制运行于不同计算机上的 Proteus 仿真。

网络上整合方法于单机整和类似，只须单击 Option for Target ‘LPC2100’ 的 Debug 选项卡上的 Setting 按钮，设置运行 Proteus 软件机器的 IP 地址，如图。



配置 IP 地址

九、 ARM 的开发步骤

1) 做个最小系统板：如果你从没有做过 ARM 的开发，建议你一开始不要贪大求全，把所有的应用都做好，因为 ARM 的启动方式和 dsp 或单片机有所不同，往往会遇到各种问题，所以建议先布一个仅有 Flash、SRAM 或 SDRAM、CPU、JTAG、和复位信号的小系统板，留出扩展接口。使最小系统能够正常运行，你的任务就完成了一半，好在 ARM 的外围接口基本都是标准接口，如果你已有这些硬件的布线经验，这对你来讲是一件很容易的事情。

2) 写启动代码，根据硬件地址先写一个能够启动的小代码，包括以下部分：
初始化端口，屏蔽中断，把程序拷贝到 SRAM 中；完成代码的重映射；配置中断向量，连接到 C 语言入口。也许你看到给你的一些示例程序当中，bootloader 会有很多东西，但是不要被这些复杂的程序所困扰，因为你不是做开发板的，你的任务就是做段小程序，让你的应用程序能够运行下去。
3) 仔细研究你所用的芯片的资料，尽管 ARM 在内核上兼容，但每家芯片都有自己的特色，编写程序时必须考虑这些问题。在这儿千万别有依赖心理，总想拿别人的示例程序修改，却越改越乱。
4) 多看一些操作系统程序，在 ARM 的应用开放源代码的程序很多，要想提高自己，就要多看别人的程序，linux,uc/os-II 等等都是很好的原码。
5) 如果你是作硬件，每个厂家基本上都有针对该芯片的 DEMO 板原理图。先将原理图消化。这样你以后做设计时，对资源的分配心中有数。器件的 DATSHEET 一定要好好消化。
6) 如果做软件最好对操作系统的机理要有所了解。
7) 四层板和 33 欧电阻：
选用四层板不仅是电源和地的问题，高速数字电路对走线的阻抗有要求，二层板不好控制阻抗。33 欧电阻一般加在驱动器端，也是起阻抗匹配作用的；布线时要先布数据地址线，和需要保证的高速线；
8) 在高频的时候，PCB 板上的走线都要看成传输线。传输线有其特征阻抗，学过传输线理论的都知道，当传输线上某处出现阻抗突变(不匹配)时，信号通过就会发生反射，反射对原信号造成干扰，严重时就会影响电路的正常工作。采用四层板时，通常外层走信号线，中间两层分别为电源和地平面，这样一方面隔离了两个信号层，更重要的是外层的走线与它们所靠近的平面形成称为“微带”(microstrip) 的传输线，它的阻抗比较固定，而且可以计算。对于两层板就比较难以做到这样。这种传输线阻抗主要于走线的宽度、到参考平面

的距离、敷铜的厚度以及介电材料的特性有关，有许多现成的公式和程序可供计算。

9) 33 欧电阻通常串连放在驱动的一端(其实不一定 33 欧，从几欧到五、六十欧都有，视电路具体情况) ，其作用是与发送器的输出阻抗串连后与走线的阻抗匹配，使反射回来(假设解收端阻抗没有匹配) 的信号不会再次反射回去(吸收掉)，这样接收端的信号就不会受到影响。接收端也可以作匹配，例如采用电阻并联，但在数字系统比较少用，因为比较麻烦，而且很多时候是一发多收，如地址总线，不如源端匹配易做。

这里所说的高频，不一定是时钟频率很高的电路，是不是高频不止看频率，更重要是看信号的上升下降时间。通常可以用上升(或下降) 时间估计电路的频率，一般取上升时间倒数的一半，比如如果上升时间是 1ns，那么它的倒数是 1000MHz，也就是说在设计电路是要按 500MHz 的频带来考虑。有时候要故意减慢边缘时间，许多高速 IC 其驱动器的输出斜率是可调的。

[参考文献]

1. 基于 Proteus 的 ARM 虚拟开发技术，周润景 袁伟亭，北京航空航天大学出版社，2007.1
2. http://www.embedworld.com/forum_view.asp?forum_id=4&view_id=2002