基于 IAR 的 Contiki 系统在 CC2530 下的移植

摘要

本文简要介绍了Contiki 系统和6Lowpan IPv6 协议标准以及相对于zigbee 的技术优势，并介绍了 Contiki 系统的源代码目录结构以及各个目录代码的功能。之后逐步讲述了在IAR EW8051 v7.51 开发工具下移植Contiki 到 CC2530 芯片的准备工作、详细步骤、IAR 工程建立和设置工程选项等，并编写一个应用程序进行测试，实现两个LED 灯的交替闪烁。

前言

操作系统 Contiki 是比较适合于无线传感网络的操作系统。它不但有基本操作系统的线程管理、线程同步语等，而且具有电源管理、文件系统、动态加载、基于 uIPv6 的网络协议栈等更为全面的功能，是非常适用于物联网应用的操作系统。本文介绍将该系统移植到 TI 的 CC2530 芯片上，为了便于调试，开发环境选用 IAR for 8051 V7.51A 版本。本文将详细介绍从移植内核到移植网络协议栈，最终完成一整套应用的过程。

目前，大多数的无线自组织网络以 Zigbee 的应用为主，但是 Zigbee 协议栈基本都是由半导体企业垄断，不开放源代码，并且协议复杂，应用程序开发难。此外，zigbee 标准已经在新版本的 SEP2.0 智能电网标准中采纳 6Lowpan，这正式标志着 zigbee 开始走下坡路，zigbee 已成为昨日黄花。庆幸的是，Contiki 出现了，Contiki 是由瑞典计算机科学学院（Swedish Institute of Computer Science）的 Adam Dunkels 和他的团队开发的。它开源、免费，易于移植，并且已经在许多商业产品中获得了成功的应用。

本文将宣告 Zigbee 及 Zigbee Pro 的终结，操作系统 Contiki 的 uIPv6 网络协议基于 IEEE 802.15.4，实现了 6Lowpan、RPL、Coap 协议。由于采用了 IPv6 作为组网协议，具有海量的地址空间，易于与互联网相连接，实现端到端的物物互联网络，具有无法比拟的优越性，这将是物联网未来的发展方向和必然趋势。

1 Contiki 基本认识

1.1 下载源代码

操作系统 Contiki 完全开源、免费，目前，最新版本是 2.5，可以从网上下载到源代码，下载地址：

http://sourceforge.net/projects/contiki/files/Contiki/Contiki%202.5/

1.2 认识代码结构

- **整体源代码目录结构**

下载到源代码后，解压缩，可以看到代码架构如图 1.1 所示。

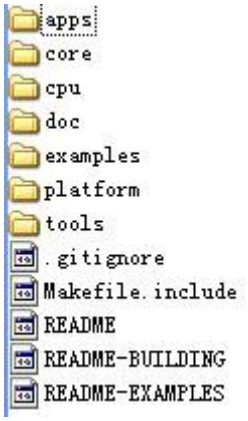

图 1.1 contiki 代码架构图

下面详细介绍下代码结构：

apps：测试应用程序，这是与硬件无关的部分，不需要移植，主要例程是基于 uIPv6 的网络测试，如 email、ftp、dhcp、ping6 等。

core： 内核代码，这其中有与内核相关的部分，也有与内核无关的部分，后面再做详细介绍。

cpu： 处理器代码，这是与处理器相关的部分。将 Contiki 移植到某一特定的处理器，仅仅需要修改该文件夹的内容即可，这部分与具体的硬件评估板无关。该文件夹内，每一个文件夹对应了一个处理器。

examples：测试例程，有一部分是与上位机相关的，这部分代码有是的基于 contiki 的，有的是基于 java 的。

platform：平台相关代码，针对不同的评估板。同样的处理器可能有不同的硬件评估板，这部分是针对特定评估板的应用程序入口。

doc：文档目录，该目录下存放了 Contiki 的介绍，以及帮助文档，读者在遇到困难的时候，可以来这里寻找解决方案。

tool：工具，包含了一些测试工具。大部分测试工具是基于 java 开发的，因此，与操作系统无关，在 windows、linux 或者是 mac os 上只要装上了 java 虚拟机，都可以运行。如测试基于 IPv6 的传感器测试工具 cooja 就在目录下，该工具可以在 java 环境下，模拟仿真出传感器网络的环境，对于开发调试有重要意义。具体，读者可以参考原厂的相关技术手册。

其它文件：是关于 Contiki 介绍以及相关说明的文档，读者可以根据需要阅读。

- **内核源代码目录结构**

本节主要介绍下内核结构。contiki 内核代码在 core 目录下，具体代码结构体如图 1.2 所示。下面详细介绍下每一个目录的功能结构：

图 1.2 Contiki 内核代码结构图

cfs：coffee 文件系统相关代码，这部分是与硬件无关的。需要移植的部分在 cpu 或者 platform 目录下。

ctk：GUI 以及 VNC 功能实现。Contiki 可以运行在 x86 平台下，可以当做一个桌面的操作系统使用。因此，GUI 是必不可少的部分，该目录实现的就是这部分功能。这部分也是与硬件无关的，具体移植的部分不再这里。

dev：该目录是一些外部设备驱动，是与硬件相关的部分，但是通常仅仅需要配置，但不需要做过多的源代码修改。如 DM9000A、CC2520 的驱动就这这部分。这些外设可以挂接在不同的处理器上，仅仅配置下地址，驱动实现的源代码都是相同的。

lib：一些基本操作的函数库实现，这也是与硬件无关的，不需要移植。如 CRC 校验、ringbuf 管理、内存分配管理等都是在这里实现的。

loader： 动态加载相关文件，相当于 windows 操作系统对动态连接口 dll 的处理。个人认为，这部分代码比较混乱。里面有与处理器相关的部分，也有与处理器无关的部分。由于不同处理器堆栈管理不同，以至于对动态加载有所不同，于是就需要针对不同处理器做一些接口移植工作。很遗憾的是这部分代码比较混乱。

net：网络相关代码，这部分也是与硬件无关的。具体的驱动是在 dev 或者 cpu 目录下实现的。这部分由为 mac 层、rime 层、rpl 层以及 uIPv6 组成，每一部分都是无线传感网络的有机组成部分，不可缺少。

sys：系统内核相关目录，这部分是与操作系统内核相关的部分，与具体的硬件

平台无关。

1.3 其它部分源代码

Contiki 是免费开源的操作系统，但是学习资料短缺，对于初学者是一个极大的挑战。庆幸的是，Internet 时代网络发达，很多的热心人在学习的过程中将他们的心得写成文档发表，对于后学者可以说是一大帮助。目前，关于系统更详细的介绍，应用程序设计更深一步的了解，读者可以到网站：

www.iotdev.net 以及

http://blog.chinaunix.net/uid/9112803/frmd/24079.html 学习了解。在此，非常感谢各位 Contiki 爱好者，你们提供的学习教程，对 Contiki 6Lowpan 的初学者提供了极大的帮助。

2 系统内核移植

本章通过将 Contiki 移植到 CC2530 的过程，逐步介绍 Contiki 操作系统内核的移植过程。Contiki 是一个非可剥夺性内核，移植非常简单。本章仅仅实现的是一个能够调度的内核，关于动态加载、文件系统、网络协议栈的移植都不在讨论范围内。

2.1 移植准备

官方的 Contiki 源码是在 Linux 开发环境进行的，移植到 IAR，首要解决的就是 GCC 与 IAR 之间的差异，这主要包括：

- GCC 是用 makefile 来管理工程编译，而 IAR 是 IDE，在 IAR 的 IDE 中需要添加那些文件，不需要添加那些文件，也是需要考虑的重点；
- 嵌入式汇编，GCC 内嵌汇编与 IAR 内嵌汇编格式差异甚大，需要全面修改 Contiki 出现内嵌汇编的地方；

- 文件差异，Linux 平台下的 Contiki 会调用 Linux 的头文件或者文件名不同；
- 增加 CC2530 的处理器移植代码，可以参考类似的处理器修改，如 CC2430。

了解了移植需要做的工作，首先就需要具备开发环境了，安装 **IAR for 8051 V7.51A** 版本，本手册采用这个版本，当然兼容更新的版本。如果读者可以找到更新的版本，当然建议使用最新的版本。

其次，读者还需要了解下 IAR 下建立工程，开发 8051 的基本应用程序的方法。这部分读者可以先了解下，编写一个最基本的 LED 灯闪烁实验来熟悉下开发环境。

2.2 建立 IAR 工程

本节首先介绍建立一个 CC2530 的 IAR 工程，然后逐步添加代码来完成 Contiki 的内核移植。

首先，逐步建立工程目录，在目录下拷贝 Contiki 源代码文件，另外再新建一个 User 目录，用来存放用户应用程序代码。 Contiki 目录存放文件名将 Contiki-2.5 后面的版本号去掉，为了保证以后更换版本方便。在建立一个 Linker 目录，将 IAR for 8051 安装目录下，8051\config 路径下的文件 lnk51ew_cc2530b.xcl 和 lnk51ew_cc2530.xcl 拷贝到 Linker 目录，以便不对原有的造成破坏。目录结构如图 2.1 所示。

图 2.1 工程目录结构

下一步，打开 IAR for 8051 IDE，建立工程，将工程保存到上一步建立好的工程目录下。如图 2.2 所示，建立好的 IAR 工程如图 2.3 所示。

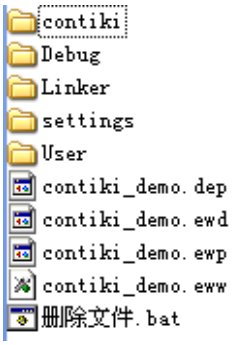

图 2.2 工程建立后目录结构

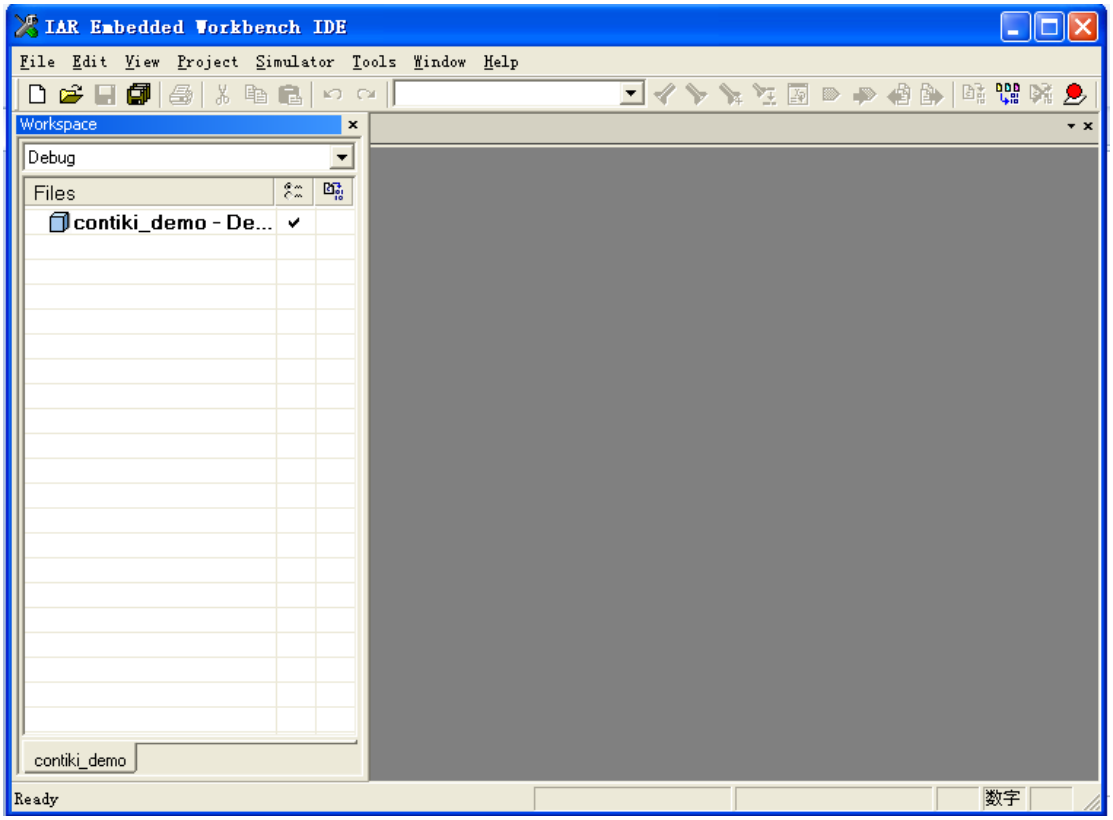

图 2.3 建立好的 IAR 工程图

下一步添加虚拟目录结构。为了便于工程阅读，按照工程文件夹下的目录结构在 IAR 工程中建立虚拟目录结构，如图 2.4 所示。在工程中只建立目录准备添加内核以及相关的代码，例程以及帮助文档不需要添加，因此，apps、examples 等目录不需要建立。platform 目录是硬件评估板平台相关的，这里用 User 目录将其替换，最终平台相关的都直接在 User 目录实现，因此，platform 也不需要添加了。cpu 目录直接添加所使用处理器的代码，不在区分具体的处理器目录。

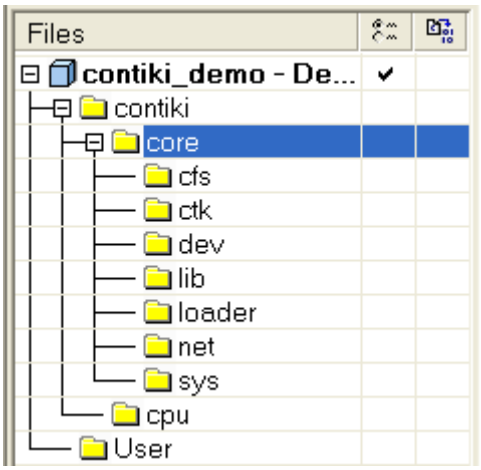

图 2.4 添加虚拟目录结构

下一步工程配置，右键点击工程按键，点击 Option 菜单，打开对话框，选择处理器以及处理器模式，如图 2.5 所示。

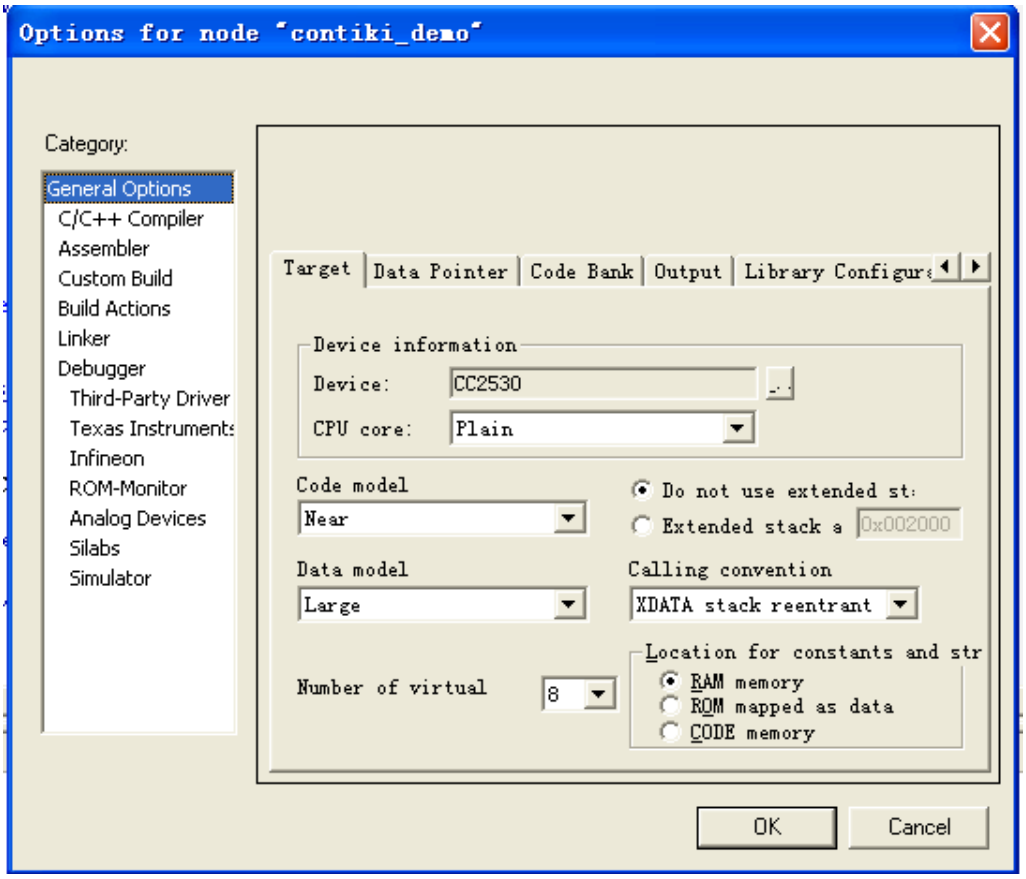

图 2.5 工程处理器配置

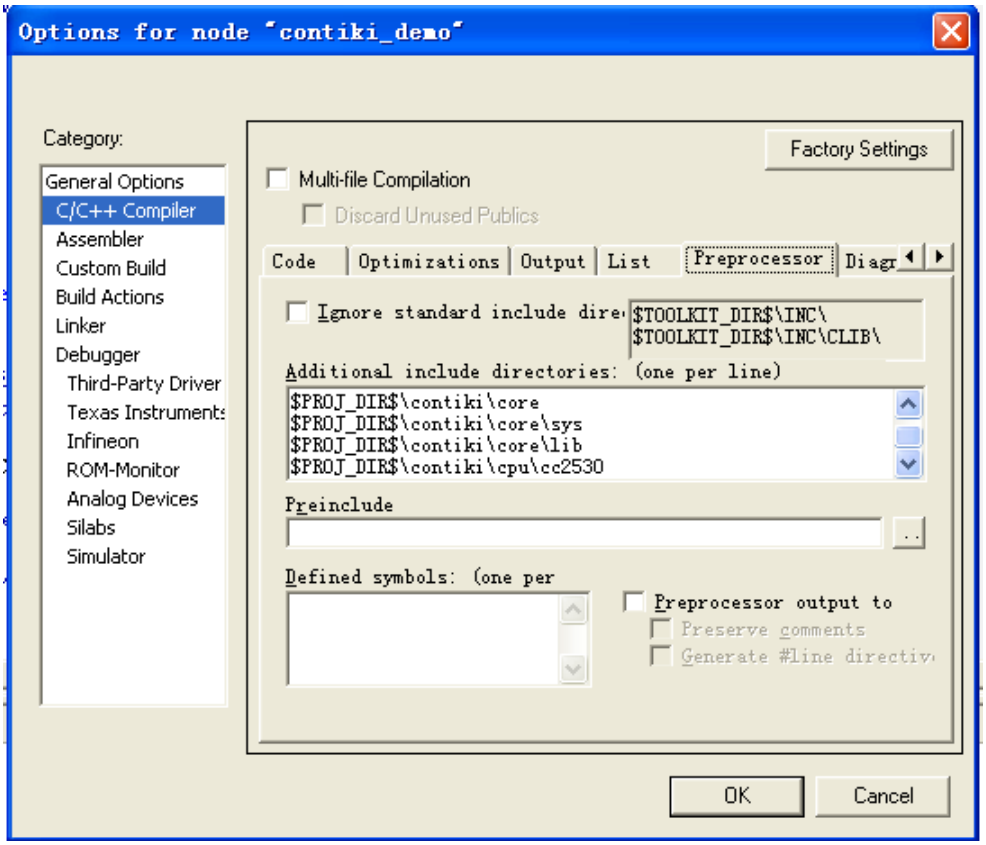

图 2.6 设置工程包含路径

下一步点击 C/C++选项卡，设置工程包含路径，以便 C/C++代码可以正确引用".h"文件，如图 2.6 所示。在 Preproessor 选项卡下，Additional include directories 输入路径如程序清单 2.1 所示。

程序清单 2.1 工程包含路径配置

```
$PROJ_DIR$\User
$PROJ_DIR$\contiki\core
$PROJ_DIR$\contiki\core\sys
$PROJ_DIR$\contiki\core\lib
$PROJ_DIR$\contiki\cpu\cc2530
```

下一步，配置工程链接，如图 2.1 所示。

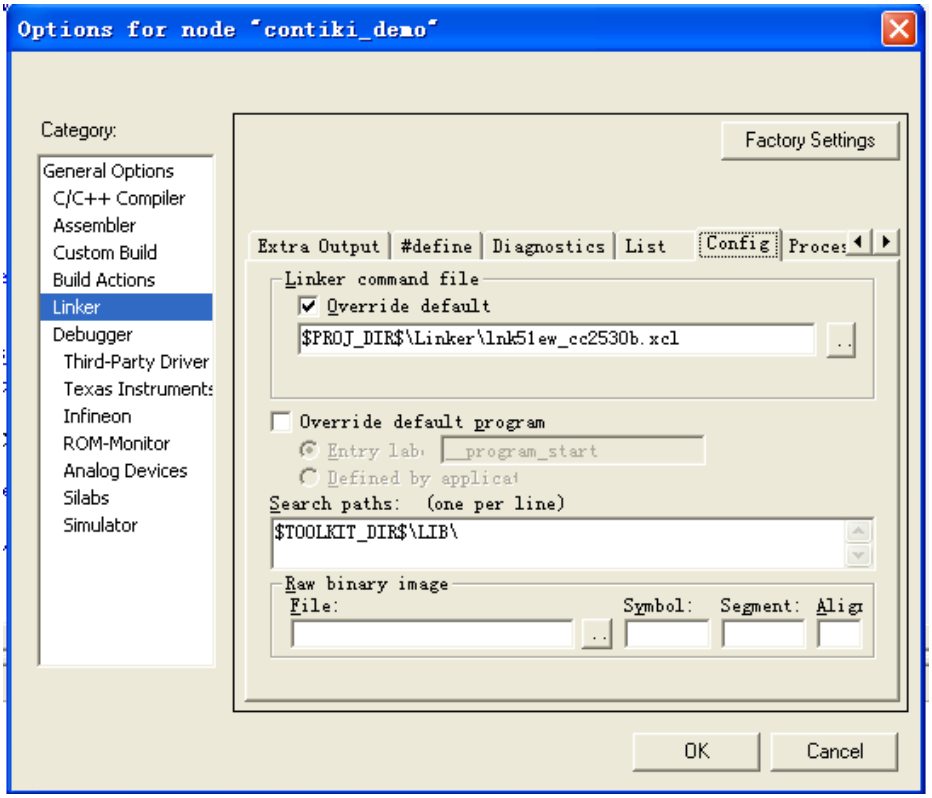

图 2.1 工程链接配置

为了便于仿真调试，在设置下调试器和下载器，如图 2.2 和图 2.3 所示。

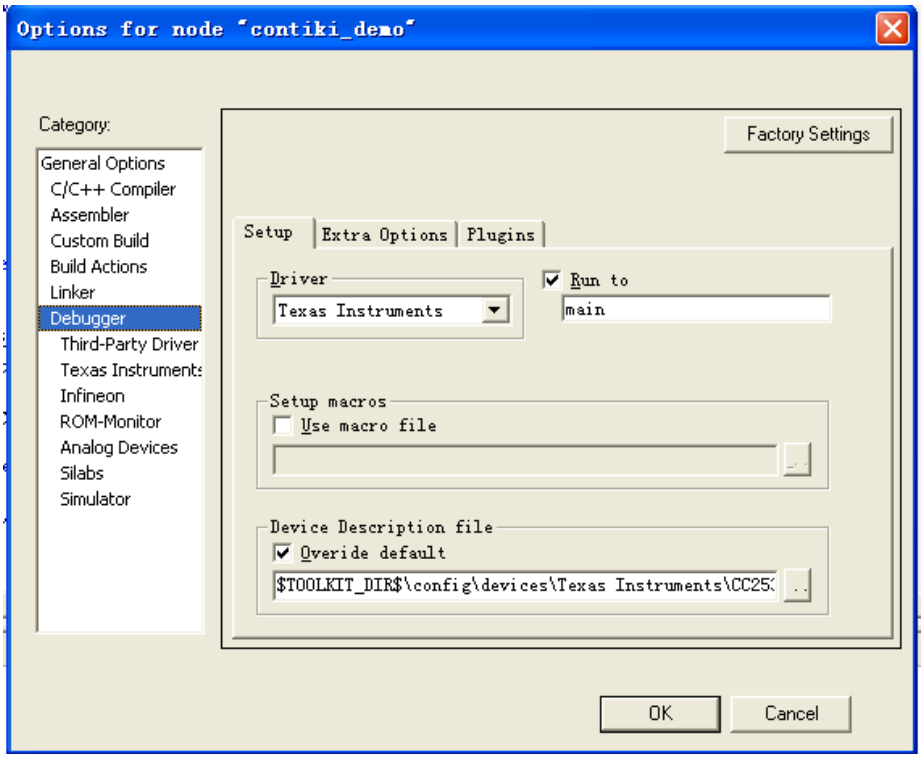

图 2.2 调试仿真器配置

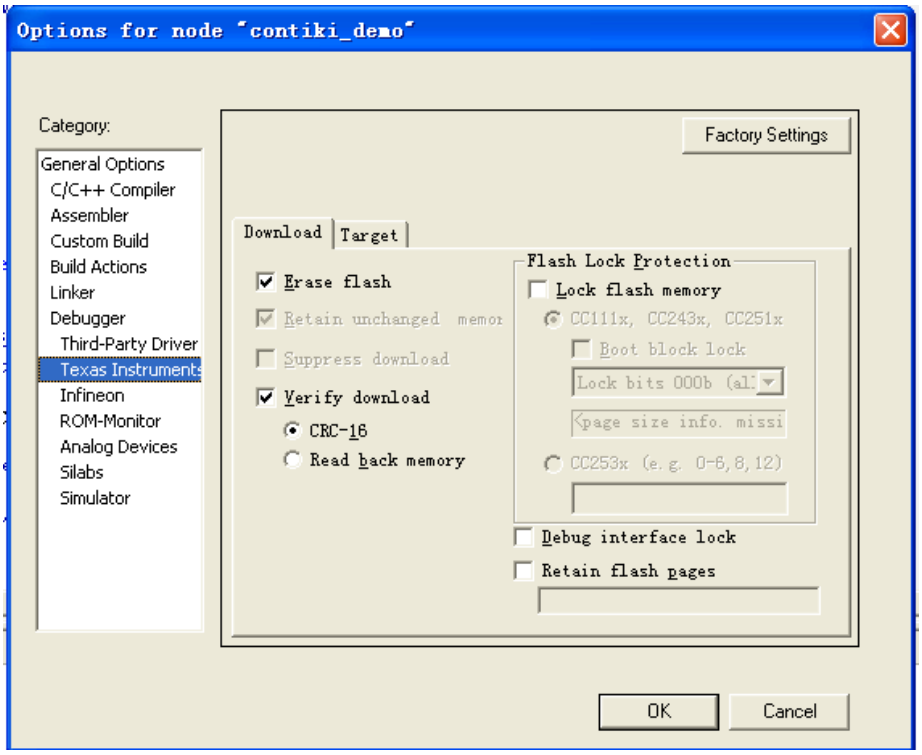

图 2.3 下载配置

其它工程选项都采用默认配置，暂时不做配置修改。

至此，工程基本结构建立完毕，下面逐步在工程中添加源代码。

2.3 向工程添加内核文件

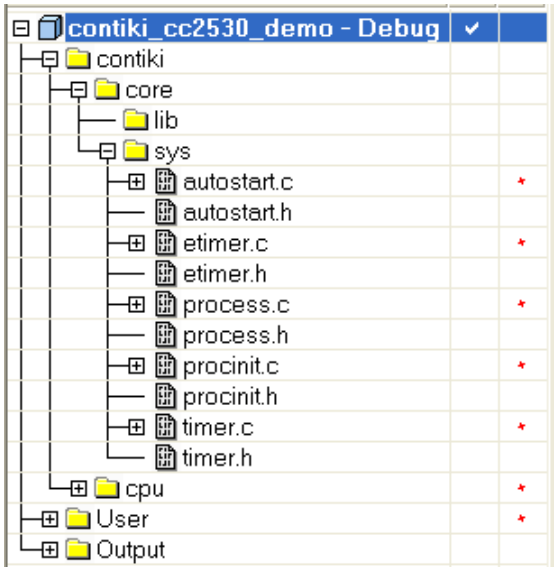

图 2.4 添加基本内核文件

本章要移植的是一个可以调度的基本内核，很多内核功能文件暂时不添加，这里暂时仅仅添加文件如图 2.4 所示。

2.4 编写移植部分代码

本节开始编写移植 CC2530 需要的移植代码。在 cpu 目录下，建立 cc2530 文件夹，移植代码主要在这个文件中完成。移植工作主要参考 cc2430 的移植代码，暂时仅仅移植内核，内核以外的部分(如网络协议栈)暂时不做考虑，后续章节再来考虑移植。

首先，将 IAR for 8051 安装目录 8051\inc 下的文件 ioCC2530.h 拷贝一份到目录 Contiki\cpu\cc2530 下，并修改文件名为 cc2530_sfr.h，避免和系统的头文件造成同名冲突。同时，修改下文件中防止文件重复包含的宏定义，如程序清单 2.2 所示。

程序清单 2.2 防止重复包含宏定义修改

```
将：
#ifndef IOCC2530_H
#define IOCC2530_H
修改为：
#ifndef __CC2530_SFR_H__
#define __CC2530_SFR_H__
```

下一步，参考 CC2430，将 cc2430 目录下的文件 8051def.h、io.h、rtimer-arch.h 拷贝到 cc2530 目录下，将里面文件内容所有关于 cc2430 的更换为 cc2530。整理下里面的代码，暂时不用做什么修改。在该目录下建立 clock.c、sys_handler.c、sys_handler.h 文件。文件 clock.c 用于完成系统 tick 的初始化以及操作系统的 tick 中断处理，该文件内容暂时为空，它的头文件声明在 sys/clock.h 下，是操作系统标准定义的，该文件可以作为移植 clock.c 的一个参考。sys_handler.c、sys_handler.h 文件对应，用于完成系统时钟初始化以及系统工作模式的选择等，在系统运行时函数 sys_init() 应该最先被调用，这两个文件也暂时为空，后面再逐步往里面添加代码。最后将这几个文件添加 IAR 的工程的 cpu 目录下，如图 2.5 所示。

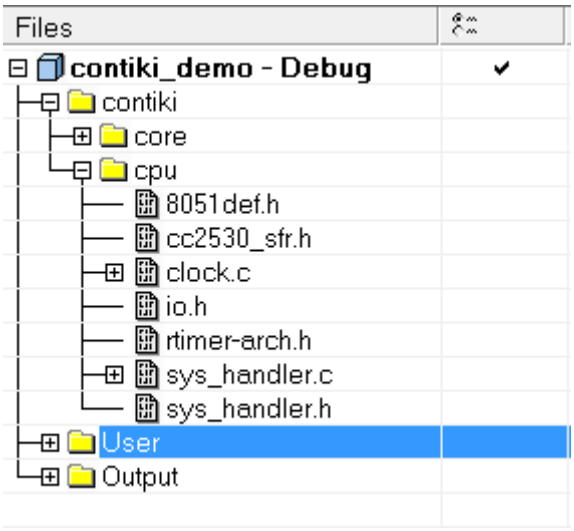

图 2.5 添加 cpu 内核移植文件

下一步，在 User 目录下，建立文件 contiki-conf.h、Main.h、Main.c 和 includes.h。下面逐一介绍着几个文件。

- contiki-conf.h：contiki 配置文件，对 Contiki 操作系统的所有配置都在这个文件中进行。
- Main.h：应用程序入口头文件声明，这里是用户程序开始的入口声明。
- Main.c：应用程序入口文件。该文件是用户编写应用程序的开始。
- includes.h：总头文件声明，这里会对应用程序应用到的文件做一个整体声明，这样在用到的地方只需要包含这一个头文件就行了。

本章仅仅是移植操作系统内核，contiki-conf.h 文件仅仅例举出基本的内核配置，该文件可以参考 platform 目录下的各种评估板下的同名文件来实现。具体内容，如程序清单 2.3 所示。

程序清单 2.3 contiki-conf.h 文件内容

```
#ifndef __CONTIKI_CONF_H__
#define __CONTIKI_CONF_H__
/*********************************************************************************************
** 头文件包含
*********************************************************************************************/
#include "8051def.h"
/*********************************************************************************************
** 使能启动函数
*********************************************************************************************/
#define AUTOSTART_ENABLE 1
```

```
/*********************************************************************************************
** 定义操作系统的 tick 频率
*********************************************************************************************/
#define CLOCK_CONF_SECOND              128

/*********************************************************************************************
** 定义文件系统 RAM 大小
*********************************************************************************************/
#define CFS_RAM_CONF_SIZE              512

/*********************************************************************************************
** 是否使能 log 日志功能
*********************************************************************************************/
#define LOG_CONF_ENABLED               0
#endif /* __CONTIKI_CONF_H__ */
```

Main.h 文件暂时为空，等待用户设计应用程序时添加。Main.c 文件主要有暂时只定义一个空的主函数，等待用户添加代码，下一节在做详细介绍，如何添加初始化代码等。

下面来逐步添加移植代码，首先，添加 cpu_init.c、cpu_init.h 的内容。文件 cpu_init.c 主要完成系统时钟的初始化话，它决定了系统运行的主频以及工作模式。实现代码如程序清单 2.4 所示。

程序清单 2.4 cpu_init. c 文件内容

```
#include <stdio.h>
#include "sys/clock.h"
#include "sys/etimer.h"
#include "cc2530_sfr.h"
#include "io.h"
/*********************************************************************************************
** Function name:       sys_init
** Descriptions:        初始化系统时钟，该函数应该在系统运行时最先调用
** Input parameters:    无
** Output parameters:   无
** Returned value:      无
** Created by:          任海波
** Created Date:        2012-06-24
*********************************************************************************************/
void sys_init(void)
```

```
{
    unsigned short i;
    unsigned char cclkconcmd;
    unsigned char cclkconsta;

    // Configure clock to use XOSC
    SLEEPCMD &= ~OSC_PD;                    /* turn on 16MHz RC and 32MHz XOSC   */
    while (!(SLEEPSTA & XOSC_STB));         /* wait for 32MHz XOSC stable    */
    __asm("nop");                           /* chip bug workaround */

    /* Require 63us delay for all revs */
    for (i=0; i<0x505; i++)
    {
        __asm("nop");
    }

    CLKCONCMD = (0x00 | OSC_32KHZ_XOSC);     /* 32MHz XOSC */
    while (CLKCONSTA != (0x00 | OSC_32KHZ_XOSC));
    SLEEPCMD |= OSC_PD;       /* turn off 16MHz RC */

    CLKCONCMD |= TICKSPD_B2 | TICKSPD_B1 | TICKSPD_B0;/*tickspeed 250 kHz*/

    cclkconcmd = CLKCONCMD;
    do{
        cclkconsta = CLKCONSTA;
    }while(cclkconcmd != cclkconsta);
    __asm("nop");
}
```

文件 sys_handler.h 中主要实现的是对 sys_handler.c 中函数的声明以及一些配置信息，这里就不列举其中的代码实现了。

最后，主要是为 cpu\cc2530\clock.c 文件添加处理代码，实现操作系统的移植。该文件主要完成系统时钟初始化、系统 tick 处理等功能。clock.c 的头文件声明在 core\sys\clock.h 下。clock.c 的具体代码实现如程序清单 2.5 所示。

程序清单 2.5 clock.c 的具体代码实现

```
/*********************************************************************************************
** 头文件包含
*********************************************************************************************/
#include <stdio.h>
#include "sys/clock.h"
```

```
#include "sys/etimer.h"
#include "cc2530_sfr.h"
#include "io.h"

/*---------------------------------------------------------------------------
** Sleep timer runs on the 32k RC osc, One clock tick is 7.8 ms
---------------------------------------------------------------------------*/
#define TICK_VAL      (32768/128)
#define MAX_TICKS     (~((clock_time_t)0) / 2)

/*---------------------------------------------------------------------------
** 全局变量定义
---------------------------------------------------------------------------*/
//   Used in sleep timer interrupt for calculating the next interrupt time
static unsigned long timer_value;
//   starts calculating the ticks right after reset
static volatile clock_time_t count = 0;
//   calculates seconds
static volatile clock_time_t seconds = 0;

/*---------------------------------------------------------------------------
** Function name:  clock_delay
** Descriptions:  时钟延时控制，One delay is about 0.6 us, so this function delays for len * 0.6 us
** Input parameters:      无
** Output parameters:    无
** Returned value: 无
---------------------------------------------------------------------------*/
void clock_delay(unsigned int len)
{
      unsigned int i;
      for(i = 0; i< len; i++)
      {
            __asm("nop");
      }
}

/*---------------------------------------------------------------------------
** Function name:  clock_wait
** Descriptions:          Wait for a multiple of ~8 ms (a tick)
** Input parameters:      无
** Output parameters:    无
** Returned value:        无
```

```
*********************************************************************************************/
void clock_wait(int i)
{
  clock_time_t start;

  start = clock_time();
  while(clock_time() - start < (clock_time_t)i);
}

/*********************************************************************************************
** Function name:  clock_time
** Descriptions:    获取时钟计数
** Input parameters:      无
** Output parameters:   无
** Returned value:  当前时钟计数值
*********************************************************************************************/
clock_time_t clock_time(void)
{
  return count;
}
/*********************************************************************************************
** Function name:  clock_seconds
** Descriptions:    获取当前时钟秒数
** Input parameters:      无
** Output parameters:   无
** Returned value:  当前时钟计数值，以秒为单位
*********************************************************************************************/
CCIF unsigned long clock_seconds(void)
{
  return seconds;
}
/*********************************************************************************************
** Function name:  clock_init
** Descriptions:    时钟初始化
** Input parameters:      无
** Output parameters:   无
** Returned value:        无
*********************************************************************************************/
void clock_init(void)
{
    //Initialize tick value
    timer_value = ST0;                          // sleep timer 0. low bits [7:0]
```

```
    timer_value += ((unsigned long int)ST1) << 8;          // middle bits [15:8]
    timer_value += ((unsigned long int)ST2) << 16;  // high bits [23:16]
    timer_value += TICK_VAL;                        // init value 256
    ST2 = (unsigned char) (timer_value >> 16);
    ST1 = (unsigned char) (timer_value >> 8);
    ST0 = (unsigned char) timer_value;

    IEN0 |= B_STIE;                                 // interrupt enable for sleep timers.
                                                        // STIE=Interrupt mask, CPU
    EA = 1;
}
/*********************************************************************************************
** Function name:  cc2530_clock_ISR
** Descriptions:     时钟 tick 中断服务函数
** Input parameters:      无
** Output parameters:    无
** Returned value: 无
** Created by:      任海波
** Created Date:    2012-06-21
*********************************************************************************************/
#pragma vector= ST_VECTOR
__interrupt void cc2530_clock_ISR( void )
{
  EA = 0;       /*interrupt disable*/
  /* When using the cooperative scheduler the timer 2 ISR is only
      required to increment the RTOS tick count. */

  /*Read value of the ST0,ST1,ST2 and then add TICK_VAL and write it back.
    Next interrupt occurs after the current time + TICK_VAL*/
  timer_value = ST0;
  timer_value += ((unsigned long int)ST1) << 8;
  timer_value += ((unsigned long int)ST2) << 16;
  timer_value += TICK_VAL;
  ST2 = (unsigned char) (timer_value >> 16);
  ST1 = (unsigned char) (timer_value >> 8);
  ST0 = (unsigned char) timer_value;

  ++count;

  /* Make sure the CLOCK_CONF_SECOND is a power of two, to ensure
      that the modulo operation below becomes a logical and and not
      an expensive divide. Algorithm from Wikipedia:
```

```
      http://en.wikipedia.org/wiki/Power_of_two */
#if (CLOCK_CONF_SECOND & (CLOCK_CONF_SECOND - 1)) != 0
#error CLOCK_CONF_SECOND must be a power of two (i.e., 1, 2, 4, 8, 16, 32, 64, ...).
#error Change CLOCK_CONF_SECOND in contiki-conf.h.
#endif
  if(count % CLOCK_CONF_SECOND == 0) {
    ++seconds;
  }

  if(etimer_pending() &&
    (etimer_next_expiration_time() - count - 1) > MAX_TICKS) {   /*core/sys/etimer.c*/
    etimer_request_poll();
  }
  /*IRCON.STIF = Sleep timer interrupt flag. This flag called this interrupt func, now reset it*/
  IRCON &= ~B_STIF;
  /*interrupt enable*/
  EA = 1;
}
```

2.5 重定向 printf 接口

函数 printf 是 C 语言编程的一个标准 IO 函数，在 PC 编程中他输出信息到控制台窗口，给用户提示应用程序的执行过程。在嵌入式编程中，可能没有 CRT 显示器，但是该函数支持重定向。例如，可以将该函数执行到从串口输出，通过串口可以查看相应的调试信息以及应用程序执行信息。这在开发调试应用程序的过程中是非常重要的。

2.6 编译工程

移植代码已经编写完毕，现在利用 IAR 的 IDE 编译工程，编译结果如图 2.6 所示。编译没有错误，但是有一些警告，该警告是由于一些系统兼容问题引起的，可以通过编译器屏蔽掉，屏蔽该编译警告的方法如图 2.7 所示。

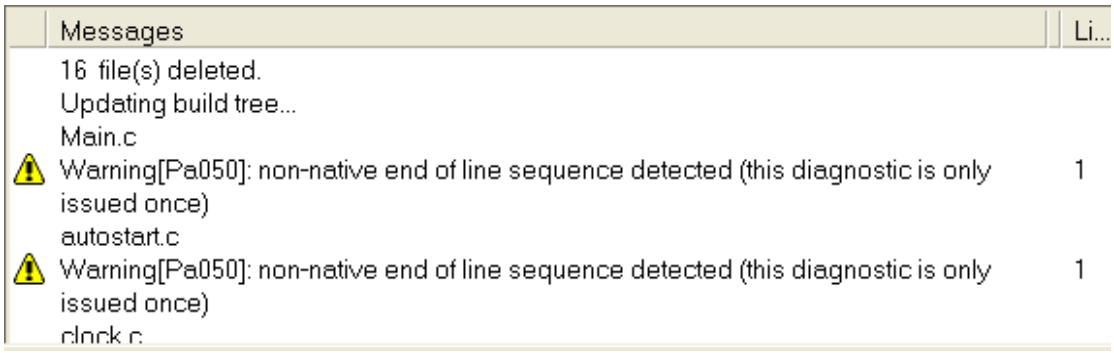

图 2.6 编译结果输出

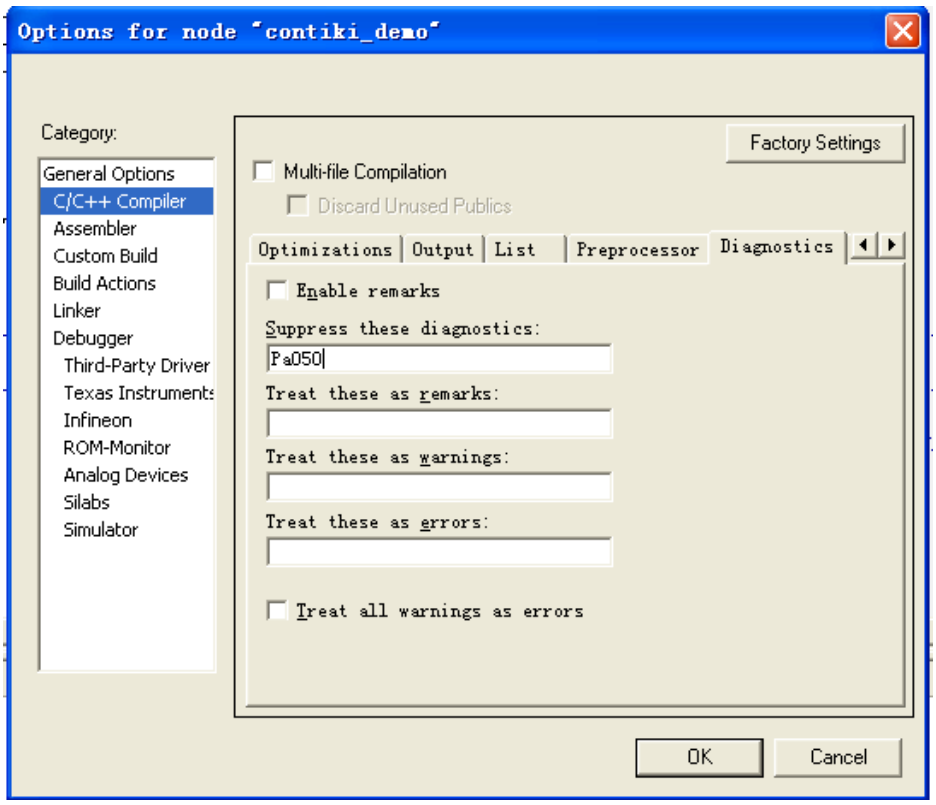

图 2.7 屏蔽编译警告

2.7 应用程序模板建立

- **includes.h 文件修改**

建立一个应用程序模板，测试移植结果。首先，includes.h 文件内容如程序清单 2.6 所示。

程序清单 2.6 includes.h 文件内容

```
#ifndef   __INCLUDES_H__
#define   __INCLUDES_H__

#ifdef __cplusplus
extern "C" {
#endif

/*********************************************************************************
  Standard header files  标准头文件
*********************************************************************************/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

/*********************************************************************************
  操作系统头文件
*********************************************************************************/
#include "contiki.h"
#include "sys_handler.h"

/*********************************************************************************
  处理器相关头文件
*********************************************************************************/
#include "cc2530_sfr.h"

/*********************************************************************************
  User's header files  用户头文件
*********************************************************************************/
#include "Main.h"
```

- **Main.c 测试例程**

在 Main.c 里面建立应测试例程，实现两个 LED 灯的交叉闪烁，具体实现代码如程序清单 2.7 所示。本节不讲解 Contiki 的具体编程方法，用户可以将其作为一个例程来运行就行了。

程序清单 2.7 测试例程代码

```
#include <includes.h>

/*********************************************************************************
**   全局变量定义
*********************************************************************************/
unsigned int idle_count = 0;

/*********************************************************************************
**   内部函数声明
*********************************************************************************/
//定义控制灯的端口
#define BLED P1_0      //蓝色 LED1 为 P10
#define RLED P1_1      //红色 LED2 为 P11

void led_init()
{
      P1DIR |= 0x03;   //P10、P11 定义为输出
      RLED = 1;
      BLED = 1;
}

PROCESS(led_process, "Blink1");
PROCESS(led_process2, "Blink2");
AUTOSTART_PROCESSES(&led_process,&led_process2);

PROCESS_THREAD(led_process, ev, data)
{
  PROCESS_BEGIN();
  while(1)
  {
     static struct etimer et;
     etimer_set(&et, CLOCK_SECOND);
     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

     //打开 LED
     RLED = !RLED;                  // LED1 灯闪一次
  }
   PROCESS_END();
}

PROCESS_THREAD(led_process2, ev, data)
{
```

```
  PROCESS_BEGIN();
  while(1)
  {
     static struct etimer et;
     etimer_set(&et, CLOCK_SECOND);
     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

     //打开 LED
     BLED = !BLED;                    // LED1 灯闪一次
  }
    PROCESS_END();
}


/*******************************************************************************
** Function name:  main
** Descriptions:     主函数的模板，一般不要随意更改
** input parameters:       无
** output parameters:     无
** Returned value:  0
*******************************************************************************/

int main (void)
{
     // 该函数必须被首先调用，它决定了系统时钟以及系统的工作模式
     sys_init();
     //dbg_setup_uart();
     led_init();
     //device_init();
     //printf("Initialising\r\n");
     // 系统 tick 初始化
     clock_init();
     process_init();
     process_start(&etimer_process, NULL);
     autostart_start(autostart_processes);
     //printf("Processes running\r\n");
     while(1) {
          do
          {
          } while(process_run() > 0);
          idle_count++;
```

```
        /* Idle! */
        /* Stop processor clock */
        /* asm("wfi"::); */
    }
    //return 0;
     }
```

3 网络协议栈移植

待续。

作者：www.iotdev.net 论坛任海波

编辑：www.iotdev.net 论坛文档组

日期： 2012-7-22