**优微电子科技**

**CC2530 基础实验**

**模块化处理**

# 实验介绍

模块化编程是将大型程序按功能拆分为若干独立命名、接口清晰、可独立编译替换的模块，模块内部高内聚、模块之间低耦合，通过严格规定的接口协作完成整体功能，从而降低复杂度、提高可维护性和复用性。

# 实验目的

通过本实验了解对各个功能进行模块化处理。

# **三、**实验设备

## 3.1、硬件设备

（1）ZIGBEE开发板

（2）SmartRF04EB仿真器

## 3.2、软件环境

（1）IAR Embedded Workbench for 8051集成开发环境。

# **四、**实验原理

## 4.1、信息隐藏

每个模块把实现细节（数据结构、算法、内部状态）隐藏在自身内部，只向外部暴露一个最小化的接口。这样，模块内部的修改不会扩散到系统其他部分，降低了变更成本。

## 4.2、抽象与契约

模块之间通过严格定义的接口（函数原型、类型、协议）进行通信。接口就是“契约”，调用方只依赖契约而不依赖实现；实现方只要满足契约即可自由演化。

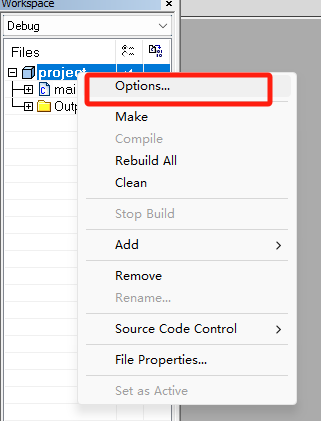
## 4.3、分治与组合

将系统功能递归分解为若干高内聚、低耦合的子功能（模块），再通过接口把这些模块按层次或网络结构组合起来。全局复杂度被分解为可独立理解、编译、测试、替换的局部复杂度，从而使整体系统易于构建、维护和扩展。

# **五、**实验演示

首先我们在我们的存放程序的文件夹内创建一个名为Code文件夹，用于放置我们即将创建的.c和.h文件。或者可以建立一个名为src的文件夹放置.c文件，创建一个Inc文件放置.h文件。这里直接创建一个名为Code文件放置所有文件。

打开IAR，右击工程，点击设置。



在这里我们要把我们新创建的文件夹在这里体现出来，这样编程时才可以找到.c和.h文件的路径，才能正常编程。

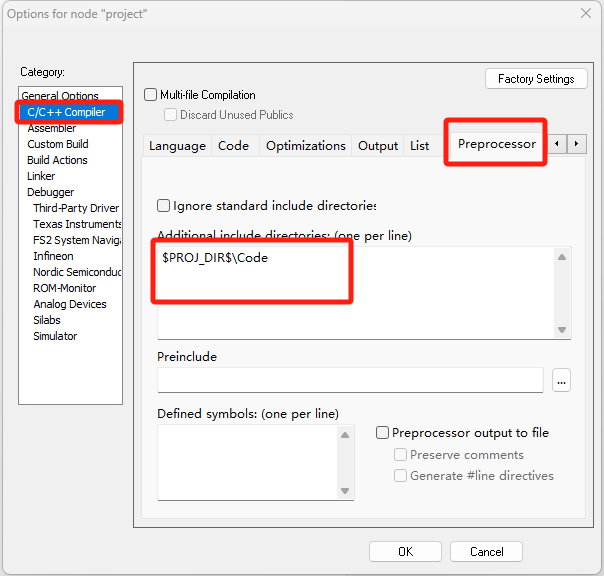
如果创建了Code文件的话就填写：

$PROJ\_DIR$\Code

如果是创建了src和Inc文件的话就填写：

$PROJ\_DIR$\src

$PROJ\_DIR$\Inc



填写完点击OK就可以开始实现我们的模块化处理了。

首先先创建一个名为headfile.h的文件，用于包含我们将创建的各个功能模块，这样我们就可以实现在主函数里包含headfile.h就可以调用其他功能。

创建完并保存完，输入以下：

#ifndef \_\_HEADFILE\_H\_\_

#define \_\_HEADFILE\_H\_\_

#include “iocc2530.h”

#endif

这样headfile.h文件才算完全创建，然后在我们的main.c内输入以下：

#include<headfile.h>

按照以上的方法，分别创建一个key.h和led.h文件，并分别输入以下：

#ifndef \_\_KEY\_H\_\_

#define \_\_KEY\_H\_\_

#include “headfile.h”

#endif

#ifndef \_\_LED\_H\_\_

#define \_\_LED\_H\_\_

#include “headfile.h”

#endif

将刚创建的key.h和led.h包含到headfile.h文件里。

#ifndef \_\_HEADFILE\_H\_\_

#define \_\_HEADFILE\_H\_\_

#include “iocc2530.h”

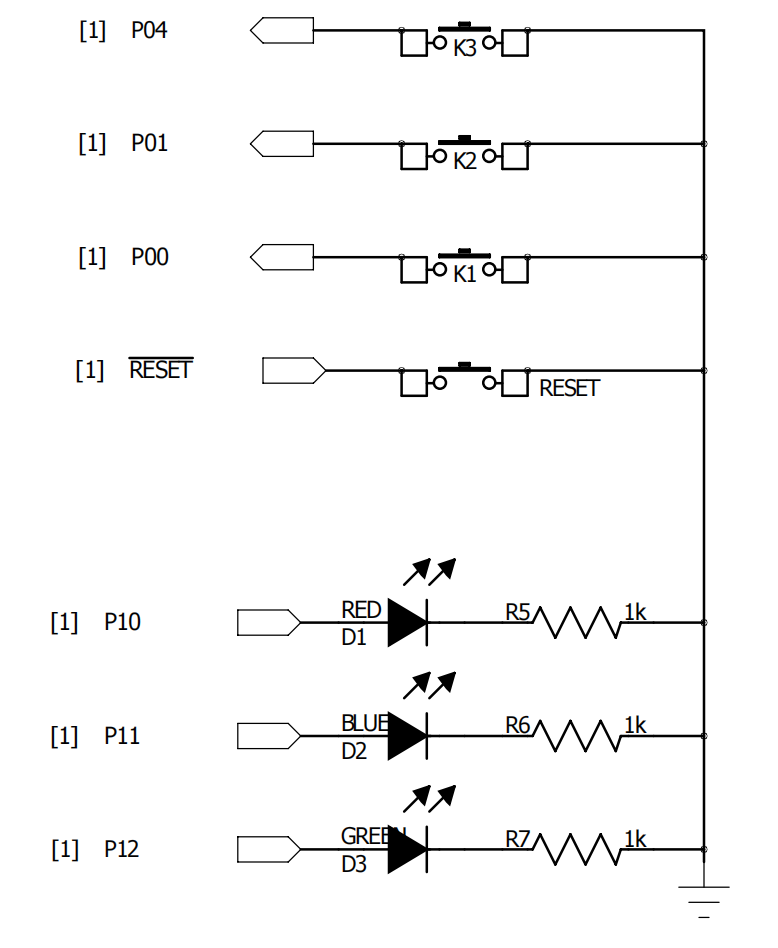
#include “key.h”

#include “led.h”

#endif

这样我们就可以在main.c内直接调用key.h和led.h内所有声明的函数。

接下来看一下原理图。



我们在headfile.h内把led和按键的引脚给进行一个宏定义。

//led

#define LED1 P1\_0

#define LED2 P1\_1

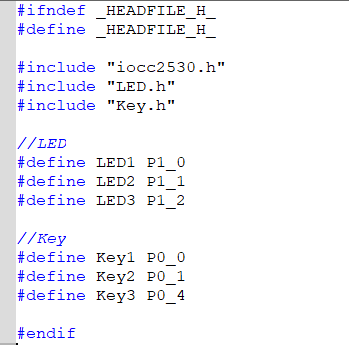
#define LED3 P1\_2

//key

#define KEY1 P0\_0

#define KEY2 P0\_1

#define KEY3 P0\_4



现在创建一个led.c文件，并输入以下：

#include "headfile.h"

//对led的三个引脚进行初始化

void led\_init(void)

{

P1SEL &= ~0x07;

P1DIR |= 0x07;

P1 = 0x00;

}

//led的亮灭模块，参数一为led的位数，参数二为led 的状态

void led\_proc(unsigned char led,unsigned char light)

{

switch(led)

{

case 1:LED1 = light;break;

case 2:LED2 = light;break;

case 3:LED3 = light;break;

default :break;

}

}



需要把对应的函数名在led.h文件内定义：

#ifndef \_LED\_H\_

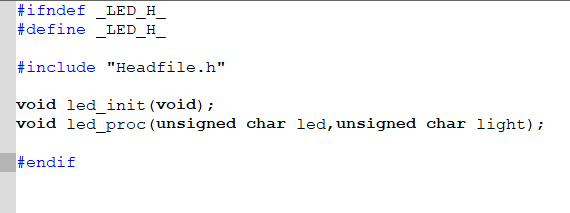
#define \_LED\_H\_

#include "headfile.h"

void led\_init(void);

void led\_proc(unsigned char led,unsigned char light);

#endif



接下来是创建key.c文件并输入以下：

#include "headfile.h"

//对三个按键进行初始化

void key\_init(void)

{

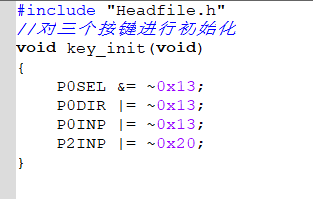
P0SEL &= ~0x13;

P0DIR |= ~0x13;

P0INP |= ~0x13;

P2INP |= ~0x20;

}



需要把对应的函数名在key.h文件内定义：

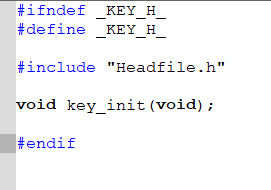
#ifndef \_KEY\_H\_

#define \_KEY\_H\_

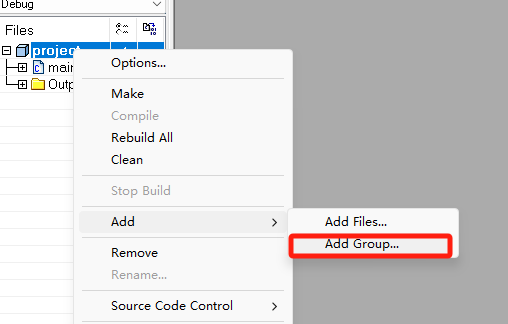
#include "headfile.h"

void key\_init(void);

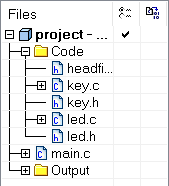
#endif



然后我们在工程内创建一个文件夹用于放置我们刚才创建的所有文件，这里命名为Code。



然后把我们刚才创建的文件都放置在这个文件夹里。



此时我们点击编译，信息窗口显示0错误和0警告就说明我们已经实现了模块化处理，接下来就是要演示该如何使用。

首先我们在主函数里分别对led和按键进行初始化，输入以下：

led\_init();

key\_init();

接下来我们实现按键1实现三个key亮的状态，按键2实现三个key灭的状态，记得使用按键时要对按键进行消抖处理。

if(KEY1 == 0)

{

delayms(10);

while(KEY1 == 0);

delayms(10);

key\_proc(1,1);

key\_proc(2,1);

key\_proc(3,1);

}

else if(KEY2 == 0)

{

delayms(10);

while(KEY2 == 0);

delayms(10);

key\_proc(1,0);

key\_proc(2,0);

key\_proc(3,0);

}

接下来是main.c内所有代码的整合：

#include<iocc2530.h> //头文件

#include<Headfile.h>

void delayms(unsigned int xms) //ms级别延时

{

for(;xms>0;xms--)

for(unsigned int j=587;j>0;j--);

}

void main(void)

{

led\_init();

key\_init();

while(1)

{

if(KEY1 == 0)

{

delayms(10);

while(KEY1 == 0);

delayms(10);

led\_proc(1,1);

led\_proc(2,1);

led\_proc(3,1);

}

else if(KEY2 == 0)

{

delayms(10);

while(KEY2 == 0);

delayms(10);

led\_proc(1,0);

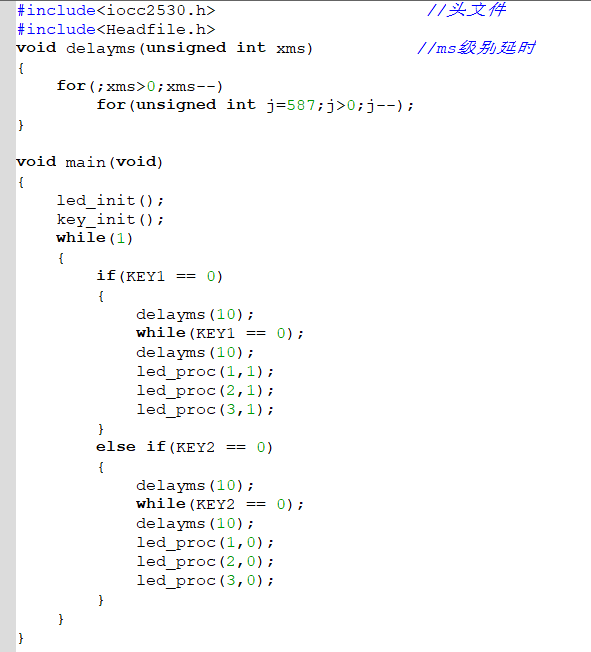
led\_proc(2,0);

led\_proc(3,0);

}

}

}



以下是我们的效果图。

这样我们就实现了模块化处理。