**光照传感器BH1750FVI使用实验**

1. **实验目的**

**该实验主要是让学生熟悉如何使用Zigbee协议采集光照度传感器数据。**

1. **实验设备**

**硬件：华智Zigbee底板（含Zigbee模块）两块，BH1750FVI光照度传感器一个，Zigbee程序下载器一个，B型方口USB数据线一根，3.7V AAA电池一颗。**

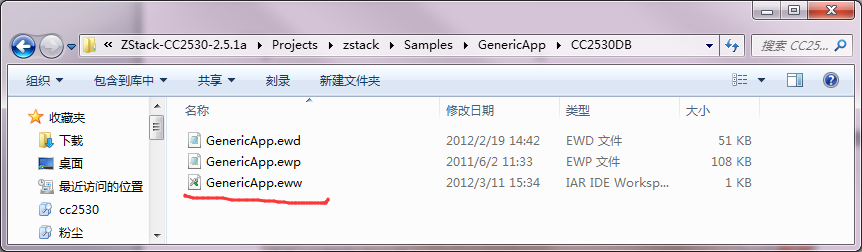
**软件：IAR开发环境，HEX文件下载程序Flah Programmer， Zigbee协议栈（2.5.1a），相关实验例程，串口调试助手。**

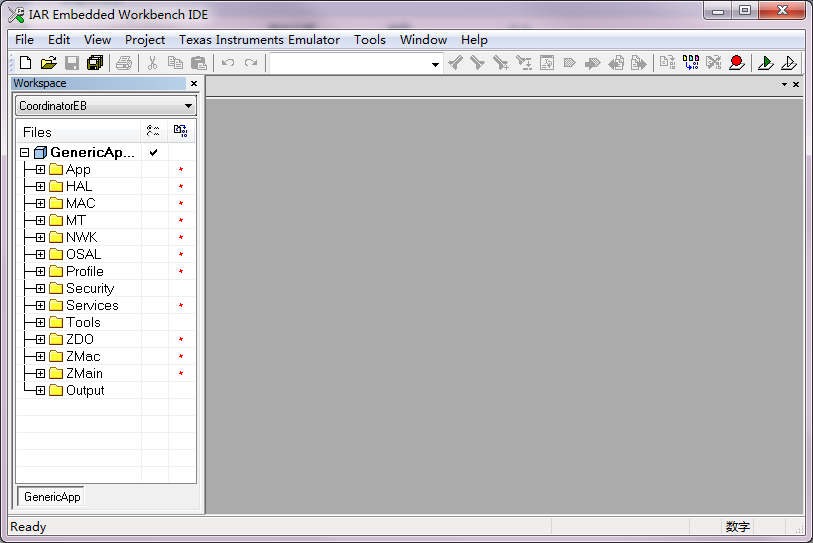
1. **实验原理**

**Zigbee协调器（无传感器）自动建立无线网络，Zigbee终端节点（带传感器）自动加入该网络，终端节点以一定周期循环采集光照度传感器数据并发送到协调器上，协调器再通过串口把数据输出到PC机端的串口调试助手上显示。**

1. **实验设计**
2. **Zigbee协议控制串口与PC 机通信。（使用下载线+延长线，B型方口线，输出“hollo”），其中：下载线+延长线连接传感器节点插槽与PC机USB口；B型方口线连接传感器节点所在单片机串口与PC机USB口。**
3. **终端节点读取BH1750FVI光照度传感器数据，并且通过串口输出到PC机的串口调试助手上显示。其中：下载线+延长线连接传感器节点插槽与PC机USB口；B型方口线连接传感器节点所在单片机串口与PC机USB口。**
4. **终端节点以Zigbee协议单播的方式把传感器数据发送到协调器上，协调器收到数据后再输出到PC机的串口调试助手上显示。（本环节要区分协调器节点和终端节点，注意在这两类节点中分别下载不同的程序）其中：B型方口线连接协调器节点串口与PC机USB口，下载线+延长线连接终端节点传感器插槽与PC机USB口。**
5. **实验步骤及相关代码**
6. **Zigbee协议控制串口与PC机通信：**

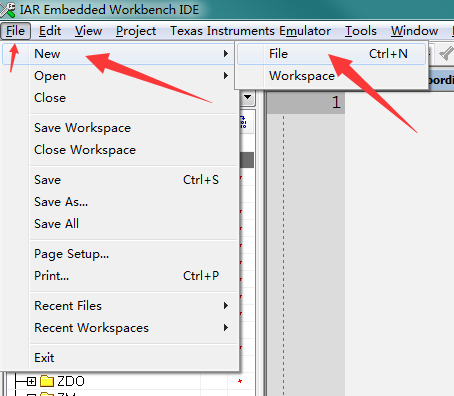
**安装好协议栈并找到工程文件（详细步骤请参考《Zigbee无线网络实验指导书》相关章节），并打开：**

****

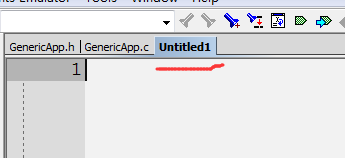
****

**创建我们自己的程序文件，并且添加到App目录下，操作如下：**

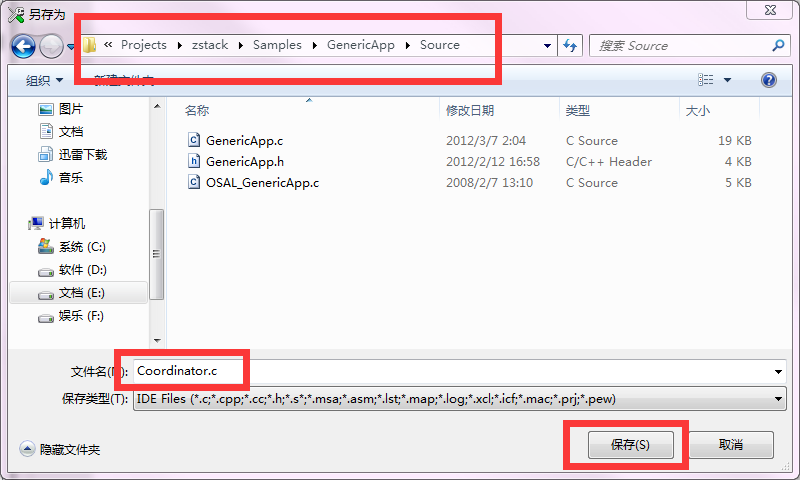
**选择以下选项**

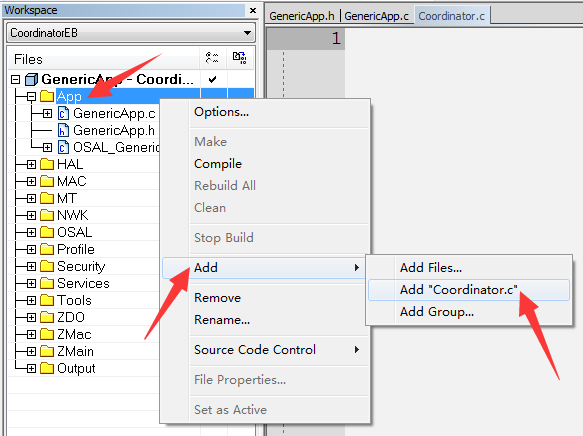
****

**新建了一个文件**

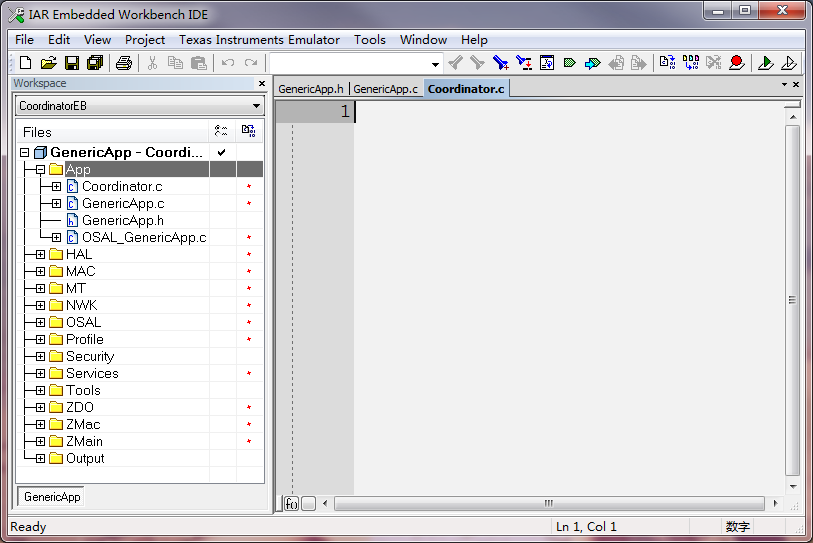
****

**点击保存按钮，选择保存路径为以下路径，并且文件命名为Coordinator.c （协调器）：**

****

****

**添加后如图：**

****

**把以下代码添加到Coordinator.c文件中：**

**#include "OSAL.h"**

**#include "AF.h"**

**#include "ZDApp.h"**

**#include "ZDObject.h"**

**#include "ZDProfile.h"**

**#include "GenericApp.h"**

**#include "DebugTrace.h"**

**#if !defined( WIN32 )**

**#include "OnBoard.h"**

**#endif**

**#include "hal\_uart.h"**

**#define Timer\_1 0x0010**

**// This list should be filled with Application specific Cluster IDs.**

**const cId\_t GenericApp\_ClusterList[GENERICAPP\_MAX\_CLUSTERS] =**

**{**

**GENERICAPP\_CLUSTERID**

**};**

**const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc =**

**{**

**GENERICAPP\_ENDPOINT, // int Endpoint;**

**GENERICAPP\_PROFID, // uint16 AppProfId[2];**

**GENERICAPP\_DEVICEID, // uint16 AppDeviceId[2];**

**GENERICAPP\_DEVICE\_VERSION, // int AppDevVer:4;**

**GENERICAPP\_FLAGS, // int AppFlags:4;**

**GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;**

**(cId\_t \*)GenericApp\_ClusterList, // byte \*pAppInClusterList;**

**GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;**

**(cId\_t \*)GenericApp\_ClusterList // byte \*pAppInClusterList;**

**};**

**endPointDesc\_t GenericApp\_epDesc;//节点描述符**

**devStates\_t GenericApp\_NwkState; //存储网络状态的变量**

**byte GenericApp\_TaskID;//任务优先级**

**byte GenericApp\_TransID;//数据发送序列号。**

**static void rcBK(uint8 port,uint8 event);**

**void GenericApp\_Init(byte task\_id)//任务初始化函数**

**{**

**GenericApp\_TaskID =task\_id; //初始化任务优先级（任务优先级有协议栈的操作系统OSAL分配）**

**GenericApp\_TransID =0; //发送数据包的序号初始化为0**

**//对节点描述符进行初始化**

**GenericApp\_epDesc.endPoint =GENERICAPP\_ENDPOINT;**

**GenericApp\_epDesc.task\_id =&GenericApp\_TaskID;**

**GenericApp\_epDesc.simpleDesc =(SimpleDescriptionFormat\_t\*)&GenericApp\_SimpleDesc;**

**GenericApp\_epDesc.latencyReq =noLatencyReqs;**

**afRegister(&GenericApp\_epDesc);//afRegister()对节点的描述符进行注册。注册后，才能使用OSAL提供的系统服务。**

**//串口的初始化**

**halUARTCfg\_t uartConfig; //该结构体变量是实现 串口的配置**

**uartConfig.configured =TRUE;**

**uartConfig.baudRate =HAL\_UART\_BR\_115200;//波特率**

**uartConfig.flowControl =FALSE; //流控制**

**uartConfig.callBackFunc = rcBK; //串口的回调函数**

**HalUARTOpen(0,&uartConfig); //串口打开**

**}**

**UINT16 GenericApp\_ProcessEvent(byte task\_id,UINT16 events)**

**{**

**afIncomingMSGPacket\_t\* MSGpkt;//MSGpkt用于指向接收消息结构体的指针**

**if(events&SYS\_EVENT\_MSG)**

**{**

**MSGpkt=(afIncomingMSGPacket\_t\*)osal\_msg\_receive(GenericApp\_TaskID);//osal\_msg\_receive（）从消息队列上接收消息**

**while(MSGpkt)**

**{**

**switch(MSGpkt->hdr.event)**

**{**

**case AF\_INCOMING\_MSG\_CMD: //接受到新数据的消息的ID是AF\_INCOMING\_MSG\_CMD,这个宏是在协议栈中定义好的值为0x1A**

**//GenericApp\_MessageMSGCB(MSGpkt);//功能是完成对接受数据的处理**

**break;**

**case ZDO\_STATE\_CHANGE: //建立网络后，设置事件**

**GenericApp\_NwkState=(devStates\_t)(MSGpkt->hdr.status);**

**if(GenericApp\_NwkState==DEV\_ZB\_COORD)//如果是协调器**

**{**

**osal\_set\_event(GenericApp\_TaskID,Timer\_1);//启动软件定时器1**

**}**

**break;**

**default:**

**break;**

**}**

**osal\_msg\_deallocate((uint8 \*)MSGpkt);//接收到的消息处理完后，释放消息所占的存储空间**

**MSGpkt=(afIncomingMSGPacket\_t\*)osal\_msg\_receive(GenericApp\_TaskID);**

**//处理完一个消息后，再从消息队列里接受消息，然后对其进行相应处理，直到所有消息处理完**

**}**

**return (events ^ SYS\_EVENT\_MSG);**

**}**

**if(events&Timer\_1)**

**{**

**HalUARTWrite(0,"Hollo!\r\n",8);**

**osal\_start\_timerEx(GenericApp\_TaskID,Timer\_1,1000); //每1000MS执行一次**

**return (events^Timer\_1);**

**}**

**return 0;**

**}**

**static void rcBK(uint8 port,uint8 event)**

**{**

**// rxlen=Hal\_UART\_RxBufLen(0); //接收缓冲区数据长度,字节为单位**

**// if(rxlen)**

**// {**

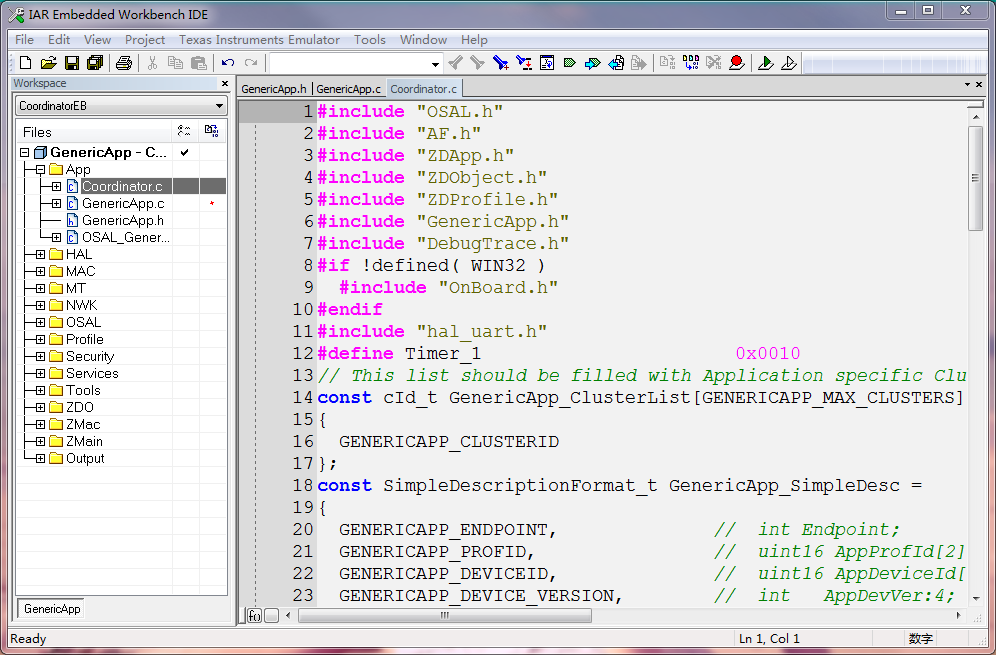
**// HalUARTRead(0,uartbuf,rxlen); //从串口读取数据放在uartbuf缓冲区中**

**// osal\_start\_timerEx(GenericApp\_TaskID, uart\_EVENT,50); //50MS后启动事件 如果再次进入则重新赋值 如果进入事件表示没有数据传送**

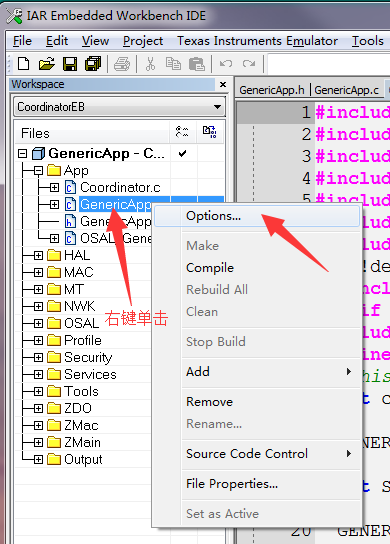
**// }**

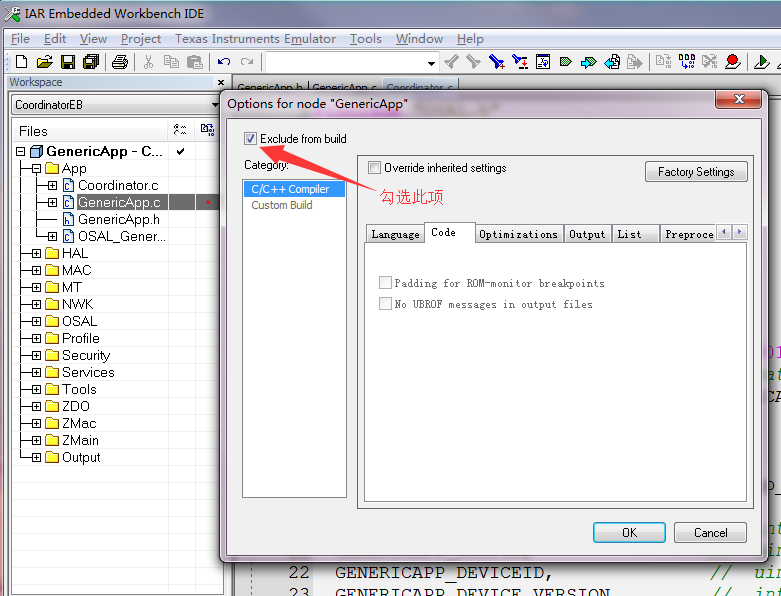
**}**

**添加后如图：**

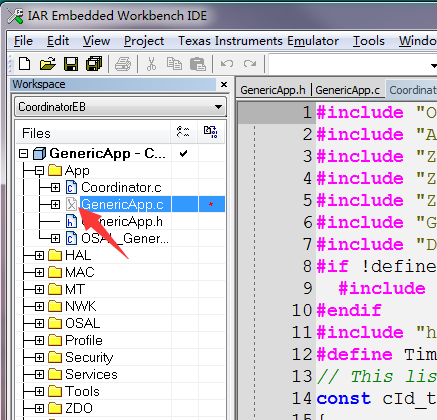
****

**把GenericApp.c文件去掉编译：**

****

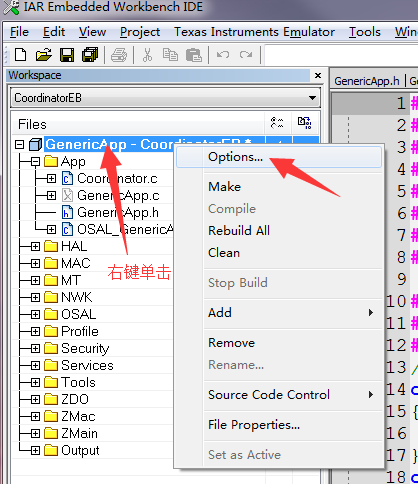
****

**此时该文件的图标应变为失效图标：**

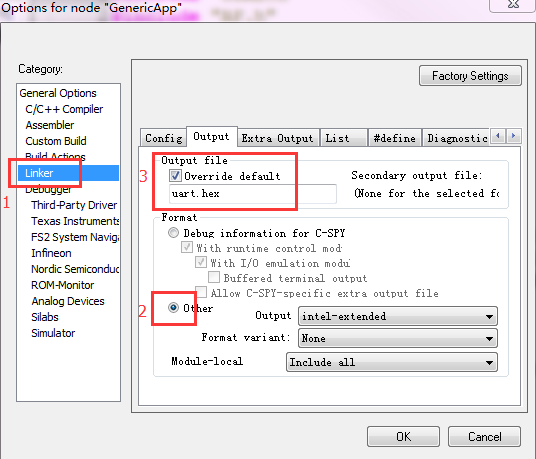
****

**下来更改生成HEX文件选项。**

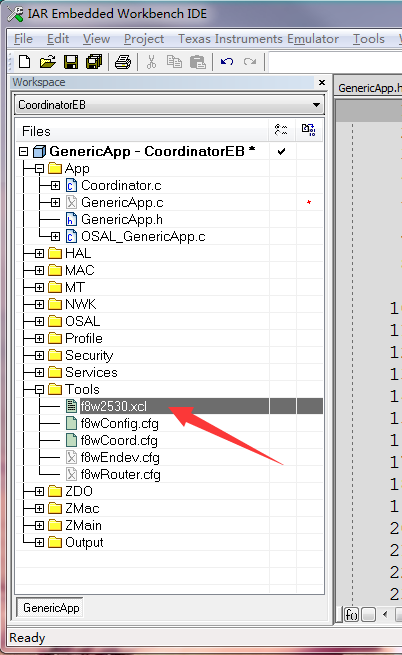
**右键单击工程名，选择Options选项：**

****

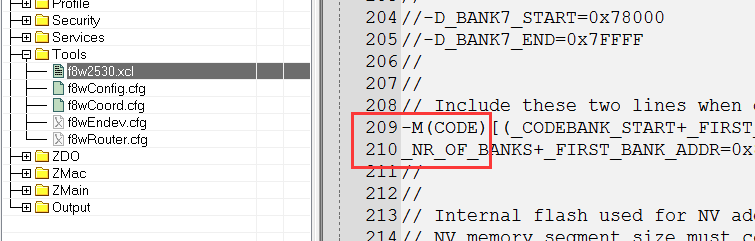
**做如下修改后点击OK：**

****

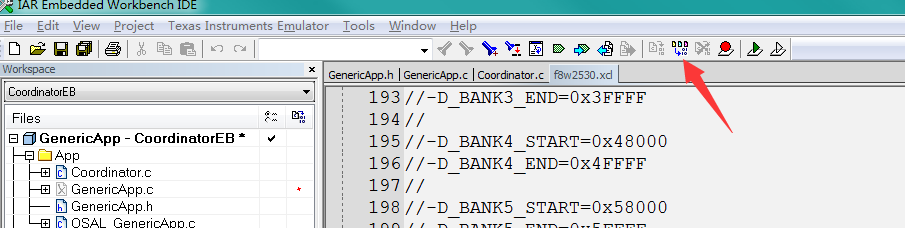
**打开工程文件夹Tools下的f8w2530.xcl文件：**

****

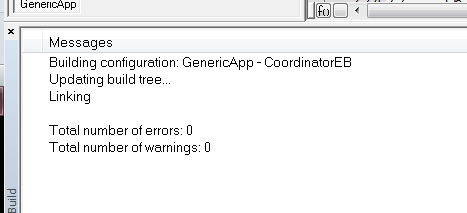
**找到该文件第209、210行代码，并且把该两行代码前面的双斜杠去掉：**

****

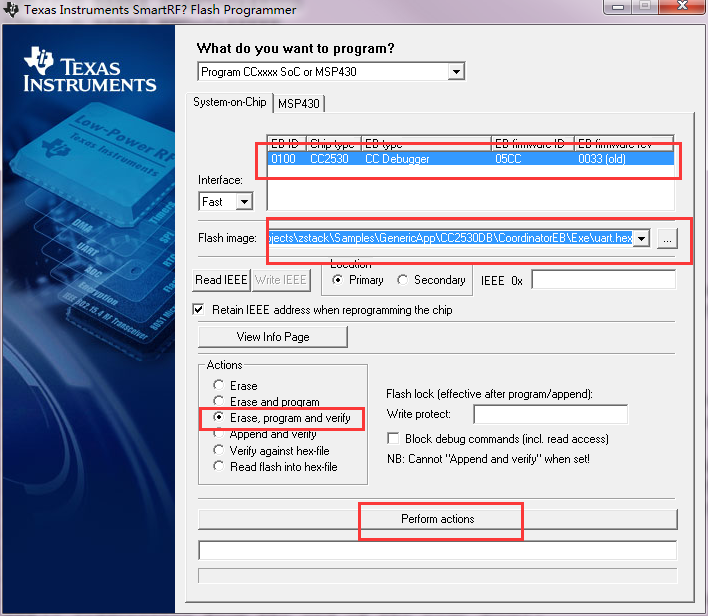
**修改完成，点击编译按钮：**

****

**编译成功的提示为：**

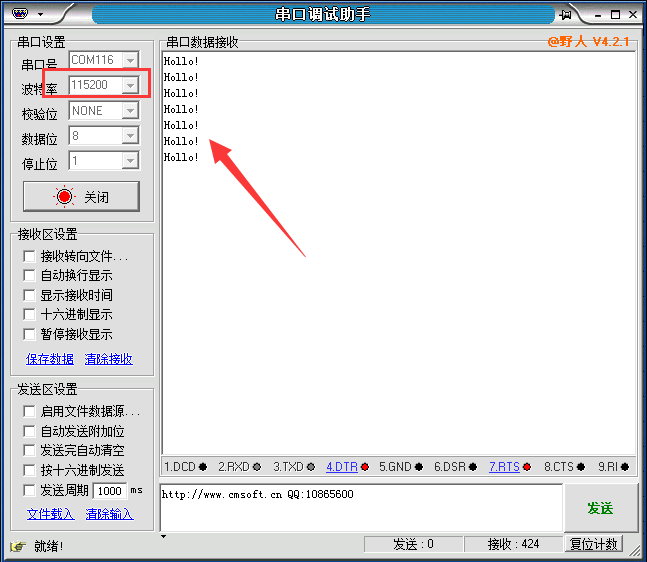
****

**把CC2530模块插入下载底板，连上下载器，打开下载软件：**

****

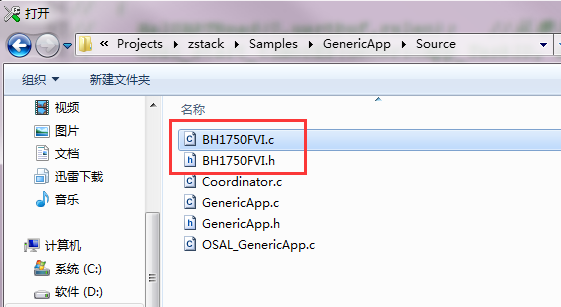
**选择好后点击Perform actions 下载程序。**

**下载完毕后，用B行USB先连接底板和电脑USB口，打开串口调试助手，选择对应的串口号和波特率，打开串口后应能接收到“Hollo!”字样：**

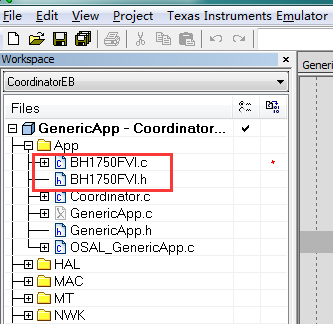
****

1. **终端节点读取BH1750FVI光照度传感器数据，并且通过串口输出到PC机的串口调试助手上显示。**

**新建两个文件，分别命名为BH1750FVI.c和BH1750FVI.h，保存到Source目录下，**

****

**并且添加到工程App目录下：**

****

**打开这两个文件，添加如下代码：**

**BH1750FVI.c的代码：**

**#include "BH1750FVI.h"**

**typedef unsigned char BYTE;**

**typedef unsigned short WORD;**

**BYTE BUF[8]; //接收数据缓存区**

**uint8 count\_light\_reset = 3;**

**#pragma optimize=none**

**void halMcuWaitUs(uint16 usec)**

**{**

**usec>>= 1;**

**while(usec--)**

**{**

**asm("NOP");asm("NOP");asm("NOP");asm("NOP");asm("NOP");**

**asm("NOP");asm("NOP");asm("NOP");asm("NOP");asm("NOP");**

**asm("NOP");asm("NOP");asm("NOP");asm("NOP");asm("NOP");**

**asm("NOP");asm("NOP");**

**}}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**延时5微秒(STC90C52RC@12M)**

**不同的工作环境,需要调整此函数，注意时钟过快时需要修改**

**当改用1T的MCU时,请调整此延时函数**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void Delay5us(void)**

**{**

**halMcuWaitUs(10);//为了稳定，多延时一点时间**

**}**

**#pragma optimize=none**

**void halMcuWaitMs(uint16 msec)**

**{**

**while(msec--)**

**halMcuWaitUs(1000);**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**延时5毫秒(STC90C52RC@12M)**

**不同的工作环境,需要调整此函数**

**当改用1T的MCU时,请调整此延时函数**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void Delay5ms(void)**

**{**

**halMcuWaitMs(10);//为了稳定，多延时一点时间**

**}**

**//毫秒延时\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void delay\_nms(unsigned int k)**

**{**

**halMcuWaitMs(k);**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**起始信号**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void BH1750\_Start(void)**

**{**

**Output\_model\_SDA();**

**Output\_model\_SCL();**

**SDA = 1; //拉高数据线**

**SCL = 1; //拉高时钟线**

**Delay5us(); //延时**

**SDA = 0; //产生下降沿**

**Delay5us(); //延时**

**SCL = 0; //拉低时钟线**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**停止信号**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void BH1750\_Stop(void)**

**{**

**Output\_model\_SDA();**

**Output\_model\_SCL();**

**SDA = 0; //拉低数据线**

**SCL = 1; //拉高时钟线**

**Delay5us(); //延时**

**SDA = 1; //产生上升沿**

**Delay5us(); //延时**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**发送应答信号**

**入口参数:ack (0:ACK 1:NAK)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void BH1750\_SendACK(uint8 ack)**

**{**

**Output\_model\_SDA();**

**Output\_model\_SCL();**

**SDA = ack; //写应答信号**

**SCL = 1; //拉高时钟线**

**Delay5us(); //延时**

**SCL = 0; //拉低时钟线**

**Delay5us(); //延时**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**接收应答信号**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**uint8 BH1750\_RecvACK(void)**

**{**

**Intput\_model\_SDA();**

**Output\_model\_SCL();**

**SCL = 1; //拉高时钟线**

**Delay5us(); //延时**

**CY = SDA; //读应答信号**

**SCL = 0; //拉低时钟线**

**Delay5us(); //延时**

**return CY;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**向IIC总线发送一个字节数据**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void BH1750\_SendByte(uint8 dat)**

**{**

**Output\_model\_SDA();**

**Output\_model\_SCL();**

**uint8 i;**

**for (i=0; i<8; i++) //8位计数器**

**{**

**if(dat & 0x80)**

**SDA = 1;**

**else**

**SDA = 0;**

**SCL = 1; //拉高时钟线**

**Delay5us(); //延时**

**SCL = 0; //拉低时钟线**

**Delay5us(); //延时**

**dat <<= 1; //移出数据的最高位**

**}**

**BH1750\_RecvACK();**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**从IIC总线接收一个字节数据**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**uint8 BH1750\_RecvByte(void)**

**{**

**Intput\_model\_SDA();**

**Output\_model\_SCL();**

**uint8 i;**

**uint8 dat = 0;**

**SDA = 1; //使能内部上拉,准备读取数据,**

**for (i=0; i<8; i++) //8位计数器**

**{**

**dat <<= 1;**

**SCL = 1; //拉高时钟线**

**Delay5us(); //延时**

**dat |= SDA; //读数据**

**SCL = 0; //拉低时钟线**

**Delay5us(); //延时**

**}**

**return dat;**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void Single\_Write\_BH1750(uint8 REG\_Address)**

**{**

**BH1750\_Start(); //起始信号**

**BH1750\_SendByte(SlaveAddress); //发送设备地址+写信号**

**BH1750\_SendByte(REG\_Address); //内部寄存器地址，**

**// BH1750\_SendByte(REG\_data); //内部寄存器数据，**

**BH1750\_Stop(); //发送停止信号**

**}**

**//\*\*\*\*\*\*\*\*单字节读取\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**/\***

**uchar Single\_Read\_BH1750(uchar REG\_Address)**

**{ uchar REG\_data;**

**BH1750\_Start(); //起始信号**

**BH1750\_SendByte(SlaveAddress); //发送设备地址+写信号**

**BH1750\_SendByte(REG\_Address); //发送存储单元地址，从0开始**

**BH1750\_Start(); //起始信号**

**BH1750\_SendByte(SlaveAddress+1); //发送设备地址+读信号**

**REG\_data=BH1750\_RecvByte(); //读出寄存器数据**

**BH1750\_SendACK(1);**

**BH1750\_Stop(); //停止信号**

**return REG\_data;**

**}**

**\*/**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//**

**//连续读出BH1750内部数据**

**//**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void Multiple\_read\_BH1750(void)**

**{**

**uint8 i;**

**BH1750\_Start(); //起始信号**

**BH1750\_SendByte(SlaveAddress+1); //发送设备地址+读信号**

**for (i=0; i<3; i++) //连续读取2个地址数据，存储中BUF**

**{**

**BUF[i] = BH1750\_RecvByte(); //BUF[0]存储0x32地址中的数据**

**if (i == 3)**

**{**

**BH1750\_SendACK(1); //最后一个数据需要回NOACK**

**}**

**else**

**{**

**BH1750\_SendACK(0); //回应ACK**

**}**

**}**

**BH1750\_Stop(); //停止信号**

**Delay5ms();**

**}**

**//初始化BH1750，根据需要请参考pdf进行修改\*\*\*\***

**void Init\_BH1750(void)**

**{**

**Single\_Write\_BH1750(0x01);**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//主程序\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**uint16 BH1750FVI\_read(void)**

**{**

**float temp;**

**uint16 dis\_data;**

**Multiple\_read\_BH1750();**

**dis\_data=BUF[0];**

**dis\_data=(dis\_data<<8)+BUF[1];//合成数据，即光照数据**

**temp=(float)dis\_data;**

**if(dis\_data == 0)//在中途拔掉传感器的情况下，重新插上传感器仍然能运行。**

**{**

**count\_light\_reset --;//开始上电，传感器前3次采集的数据为0，无效。**

**if(!count\_light\_reset)**

**{**

**Single\_Write\_BH1750(0x01); // power on**

**Single\_Write\_BH1750(0x10); // H- resolution mode2**

**delay\_nms(120); //延时120ms**

**Multiple\_read\_BH1750();**

**dis\_data=BUF[0];**

**dis\_data=(dis\_data<<8)+BUF[1];//合成数据，即光照数据**

**temp=(float)dis\_data;**

**}**

**}**

**return (uint16)temp;**

**}**

**BH1750FVI.h的代码：**

**#include "GenericApp.h"**

**#include "ioCC2530.h"**

**#include <math.h>**

**#include <stdio.h>**

**#define uchar unsigned char**

**#define uint unsigned int**

**#define SDA P0\_5**

**#define SCL P0\_4**

**#define Intput\_model\_SDA() do{P0SEL &= ~0X20;P0DIR &= ~0X20;} while(0)//设置SDA为输入模式**

**#define Output\_model\_SDA() do{P0SEL &= ~0X20;P0DIR |= 0X20;} while(0)//设置SDA为输出模式**

**#define Output\_model\_SCL() do{P0SEL &= ~0X10;P0DIR |= 0X10;} while(0)//设置SCL为输出模式**

**#define SlaveAddress 0x46 //定义器件在IIC总线中的从地址,根据ALT ADDRESS地址引脚不同修改**

**//ALT ADDRESS引脚接地时地址为0x46，接电源时地址为0xB8**

**void delay\_nms(unsigned int k);**

**void Init\_BH1750(void);**

**void Single\_Write\_BH1750(uchar REG\_Address); //单个写入数据**

**uchar Single\_Read\_BH1750(uchar REG\_Address); //单个读取内部寄存器数据**

**void Multiple\_read\_BH1750(void);**

**//------------------------------------**

**void halMcuWaitUs(uint16 usec);**

**void halMcuWaitMs(uint16 msec);**

**void Delay5us(void);**

**void Delay5ms(void);**

**void BH1750\_Start(void); //起始信号**

**void BH1750\_Stop(void); //停止信号**

**void BH1750\_SendACK(uint8 ack); //应答ACK**

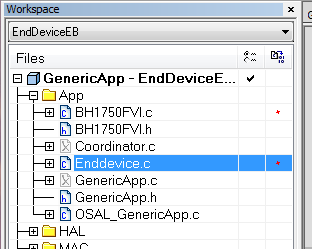
**uint8 BH1750\_RecvACK(void); //读ack**

**void BH1750\_SendByte(uint8 dat); //IIC单个字节写**

**uint8 BH1750\_RecvByte(void); //IIC单个字节读**

**uint16 BH1750FVI\_read(void);**

**再新建一个文件，命名为Enddevice.c（终端节点），保存在Source目录下，并且添加到App目录下：**

****

**添加如下代码到Enddevice.c文件：**

**#include "OSAL.h"**

**#include "AF.h"**

**#include "ZDApp.h"**

**#include "ZDObject.h"**

**#include "ZDProfile.h"**

**#include "GenericApp.h"**

**#include "DebugTrace.h"**

**#if !defined( WIN32 )**

**#include "OnBoard.h"**

**#endif**

**#include "hal\_uart.h"**

**#include "BH1750FVI.h"**

**#define Timer\_1 0x0010**

**// This list should be filled with Application specific Cluster IDs.**

**const cId\_t GenericApp\_ClusterList[GENERICAPP\_MAX\_CLUSTERS] =**

**{**

**GENERICAPP\_CLUSTERID**

**};**

**const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc =**

**{**

**GENERICAPP\_ENDPOINT, // int Endpoint;**

**GENERICAPP\_PROFID, // uint16 AppProfId[2];**

**GENERICAPP\_DEVICEID, // uint16 AppDeviceId[2];**

**GENERICAPP\_DEVICE\_VERSION, // int AppDevVer:4;**

**GENERICAPP\_FLAGS, // int AppFlags:4;**

**GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;**

**(cId\_t \*)GenericApp\_ClusterList, // byte \*pAppInClusterList;**

**GENERICAPP\_MAX\_CLUSTERS, // byte AppNumInClusters;**

**(cId\_t \*)GenericApp\_ClusterList // byte \*pAppInClusterList;**

**};**

**endPointDesc\_t GenericApp\_epDesc;//节点描述符**

**devStates\_t GenericApp\_NwkState; //存储网络状态的变量**

**byte GenericApp\_TaskID;//任务优先级**

**byte GenericApp\_TransID;//数据发送序列号。**

**static void rcBK(uint8 port,uint8 event);**

**void GenericApp\_Init(byte task\_id)//任务初始化函数**

**{**

**GenericApp\_TaskID =task\_id; //初始化任务优先级（任务优先级有协议栈的操作系统OSAL分配）**

**GenericApp\_TransID =0; //发送数据包的序号初始化为0**

**//对节点描述符进行初始化**

**GenericApp\_epDesc.endPoint =GENERICAPP\_ENDPOINT;**

**GenericApp\_epDesc.task\_id =&GenericApp\_TaskID;**

**GenericApp\_epDesc.simpleDesc =(SimpleDescriptionFormat\_t\*)&GenericApp\_SimpleDesc;**

**GenericApp\_epDesc.latencyReq =noLatencyReqs;**

**afRegister(&GenericApp\_epDesc);//afRegister()对节点的描述符进行注册。注册后，才能使用OSAL提供的系统服务。**

**//串口的初始化**

**halUARTCfg\_t uartConfig; //该结构体变量是实现 串口的配置**

**uartConfig.configured =TRUE;**

**uartConfig.baudRate =HAL\_UART\_BR\_115200;//波特率**

**uartConfig.flowControl =FALSE; //流控制**

**uartConfig.callBackFunc = rcBK; //串口的回调函数**

**HalUARTOpen(0,&uartConfig); //串口打开**

**osal\_start\_timerEx(GenericApp\_TaskID,Timer\_1,2000); //2000MS后启动定时任务1**

**HalUARTWrite(0,"\r\n",2);**

**}**

**UINT16 GenericApp\_ProcessEvent(byte task\_id,UINT16 events)**

**{**

**afIncomingMSGPacket\_t\* MSGpkt;//MSGpkt用于指向接收消息结构体的指针**

**if(events&SYS\_EVENT\_MSG)**

**{**

**MSGpkt=(afIncomingMSGPacket\_t\*)osal\_msg\_receive(GenericApp\_TaskID);//osal\_msg\_receive（）从消息队列上接收消息**

**while(MSGpkt)**

**{**

**switch(MSGpkt->hdr.event)**

**{**

**case AF\_INCOMING\_MSG\_CMD: //接受到新数据的消息的ID是AF\_INCOMING\_MSG\_CMD,这个宏是在协议栈中定义好的值为0x1A**

**//GenericApp\_MessageMSGCB(MSGpkt);//功能是完成对接受数据的处理**

**break;**

**case ZDO\_STATE\_CHANGE: //建立网络后，设置事件**

**GenericApp\_NwkState=(devStates\_t)(MSGpkt->hdr.status);**

**if(GenericApp\_NwkState==DEV\_END\_DEVICE)//如果是终端节点**

**{**

**}**

**break;**

**default:**

**break;**

**}**

**osal\_msg\_deallocate((uint8 \*)MSGpkt);//接收到的消息处理完后，释放消息所占的存储空间**

**MSGpkt=(afIncomingMSGPacket\_t\*)osal\_msg\_receive(GenericApp\_TaskID);**

**//处理完一个消息后，再从消息队列里接受消息，然后对其进行相应处理，直到所有消息处理完**

**}**

**return (events ^ SYS\_EVENT\_MSG);**

**}**

**if(events&Timer\_1)**

**{**

**uint16 LightValue;**

**uint8 dat;**

**LightValue = BH1750FVI\_read();**

**HalUARTWrite(0,"Light:",6);**

**dat = LightValue / 10000 + '0';**

**HalUARTWrite(0,&dat,1);**

**dat = LightValue % 10000 / 1000 + '0';**

**HalUARTWrite(0,&dat,1);**

**dat = LightValue % 1000 / 100 + '0';**

**HalUARTWrite(0,&dat,1);**

**dat = LightValue % 100 / 10 + '0';**

**HalUARTWrite(0,&dat,1);**

**dat = LightValue % 10 + '0';**

**HalUARTWrite(0,&dat,1);**

**HalUARTWrite(0," LUX",4);**

**HalUARTWrite(0,"\r\n",2);**

**osal\_start\_timerEx(GenericApp\_TaskID,Timer\_1,2000); //每2000MS执行一次**

**return (events^Timer\_1);**

**}**

**return 0;**

**}**

**static void rcBK(uint8 port,uint8 event)**

**{**

**// rxlen=Hal\_UART\_RxBufLen(0); //接收缓冲区数据长度,字节为单位**

**// if(rxlen)**

**// {**

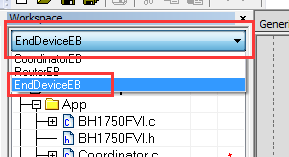
**// HalUARTRead(0,uartbuf,rxlen); //从串口读取数据放在uartbuf缓冲区中**

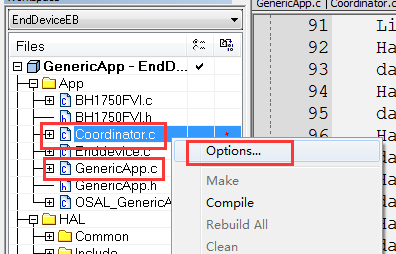
**// osal\_start\_timerEx(GenericApp\_TaskID, uart\_EVENT,50); //50MS后启动事件 如果再次进入则重新赋值 如果进入事件表示没有数据传送**

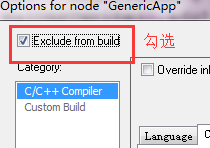
**// }**

**}**

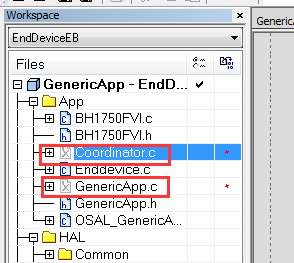
**选择工程为EndDeviceEB选项以及设置工程可编译的文件：**

****

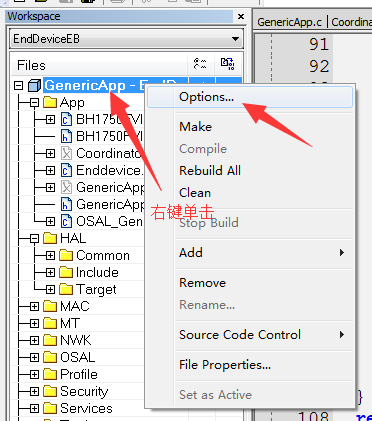
****

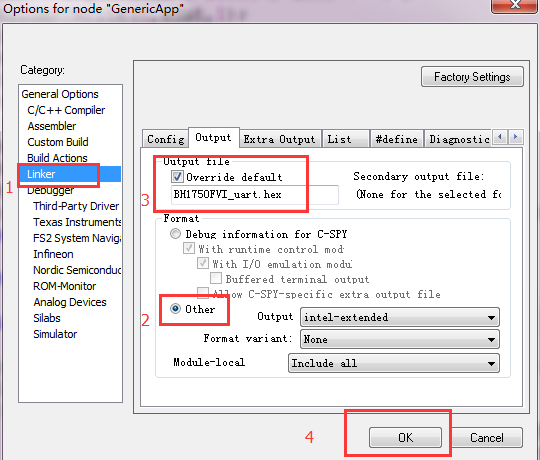
****

**修改完后的效果：**

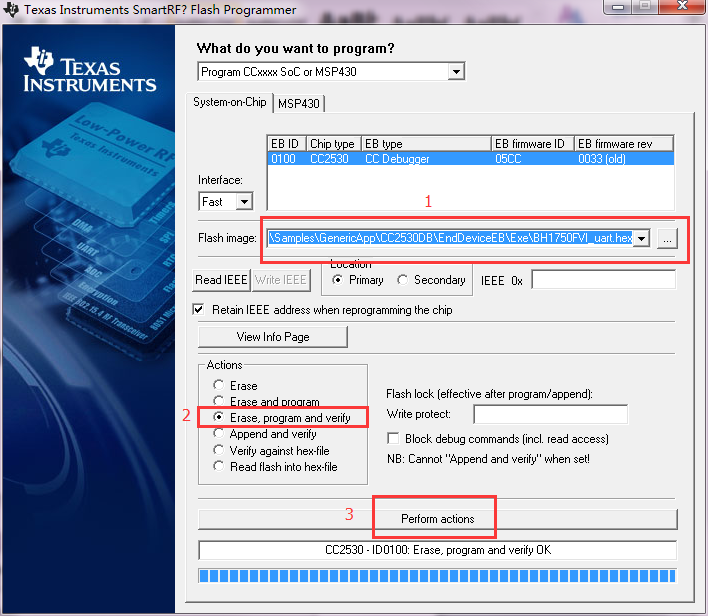
****

**配置工程的编译输出选项：**

****

****

**点击编译按钮，编译完成即可打开下载软件来下载程序：**

**下载完后，用串口先把板子的串口和电脑的USB口连接起来，打开串口调试助手，选择对应的串口参数，打开串口即可看到相关数据：**

****

**开始的数据为系统上电自检发出的数据，如不需要，可以用搜索功能找到相关代码，注释掉即可。**

1. **终端节点以Zigbee协议单播的方式把传感器数据发送到协调器上，协调器收到数据后再输出到PC机的串口调试助手上显示。**

**修改Enddevice.c文件：**

**删除函数“void GenericApp\_Init(byte task\_id)”的后两行，删除后为：**

**void GenericApp\_Init(byte task\_id)//任务初始化函数**

**{**

**GenericApp\_TaskID =task\_id; //初始化任务优先级（任务优先级有协议栈的操作系统OSAL分配）**

**GenericApp\_TransID =0; //发送数据包的序号初始化为0**

**//对节点描述符进行初始化**

**GenericApp\_epDesc.endPoint =GENERICAPP\_ENDPOINT;**

**GenericApp\_epDesc.task\_id =&GenericApp\_TaskID;**

**GenericApp\_epDesc.simpleDesc =(SimpleDescriptionFormat\_t\*)&GenericApp\_SimpleDesc;**

**GenericApp\_epDesc.latencyReq =noLatencyReqs;**

**afRegister(&GenericApp\_epDesc);//afRegister()对节点的描述符进行注册。注册后，才能使用OSAL提供的系统服务。**

**//串口的初始化**

**halUARTCfg\_t uartConfig; //该结构体变量是实现 串口的配置**

**uartConfig.configured =TRUE;**

**uartConfig.baudRate =HAL\_UART\_BR\_115200;//波特率**

**uartConfig.flowControl =FALSE; //流控制**

**uartConfig.callBackFunc = rcBK; //串口的回调函数**

**HalUARTOpen(0,&uartConfig); //串口打开**

**}**

**在此地方添加**

**“osal\_start\_timerEx(GenericApp\_TaskID,Timer\_1,2000); //2000MS后启动定时任务1”**

**这行代码：**

****

**修改“If(events&Timer\_1){}”函数为：**

**if(events&Timer\_1)**

**{**

**uint16 LightValue;**

**uint8 data[15] = {"Light:00000 LUX"};**

**LightValue = BH1750FVI\_read();**

**data[6] = LightValue / 10000 + '0';**

**data[7] = LightValue % 10000 / 1000 + '0';**

**data[8] = LightValue % 1000 / 100 + '0';**

**data[9] = LightValue % 100 / 10 + '0';**

**data[10] = LightValue % 10 + '0';**

**afAddrType\_t my\_DstAddr;**

**my\_DstAddr.addrMode=(afAddrMode\_t)Addr16Bit;//数据发送模式：可选 单播、广播、多播方式 这里选Addr16Bit表单播**

**my\_DstAddr.endPoint=GENERICAPP\_ENDPOINT; //初始化端口函**

**my\_DstAddr.addr.shortAddr=0x0000; //标志目的地址节点的网络地址 这里是协调器的地址**

**//下面是数据发送**

**AF\_DataRequest(&my\_DstAddr,**

**&GenericApp\_epDesc,**

**GENERICAPP\_CLUSTERID,**

**sizeof(data),**

**(uint8 \*)&data,**

**&GenericApp\_TransID,**

**AF\_DISCV\_ROUTE,**

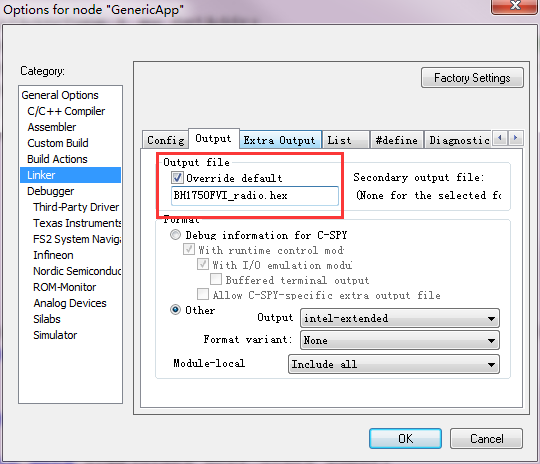
**AF\_DEFAULT\_RADIUS);**

**osal\_start\_timerEx(GenericApp\_TaskID,Timer\_1,2000); //每2000MS执行一次**

**return (events^Timer\_1);**

**}**

**修改生成的HEX文件名称：**

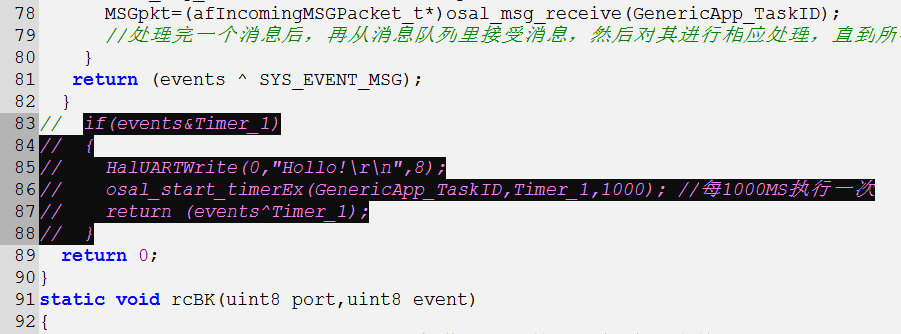
****

**编译无错误即可。**

**修改Coordinator.c文件：**

****

**65行去掉前面的双斜杠，注释掉71行代码；**

****

**注释掉这个函数（统一选定按Ctrl+k）；**

**在程序后面添加以下代码：**

**//协调器接受到终端节点发送来的数据时，调用下面这个函数，然后把数据发送到PC串口调试助手**

**void GenericApp\_MessageMSGCB(afIncomingMSGPacket\_t\* pkt)**

**{**

**uint8 data[15];**

**switch(pkt->clusterId)**

**{**

**case GENERICAPP\_CLUSTERID:**

**if(osal\_memcpy(data,pkt->cmd.Data,pkt->cmd.DataLength))**

**{**

**HalUARTWrite(0,data,pkt->cmd.DataLength);**

**HalUARTWrite(0,"\r\n",2);**

**}**

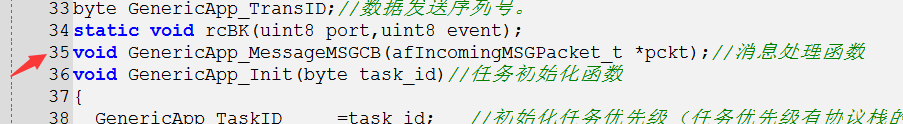
**break;**

**}**

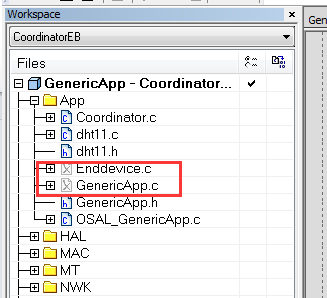
**} **

**在此处添加一行代码：**

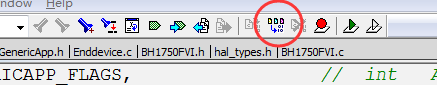
**“void GenericApp\_MessageMSGCB(afIncomingMSGPacket\_t \*pckt);//消息处理函数”**

****

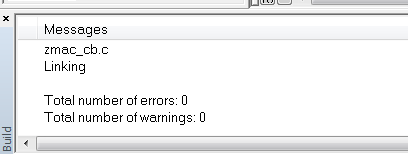
**记得修改参与工程编译的文件：**

****

**点击编译按钮：**

****

**无错误即可：**

****

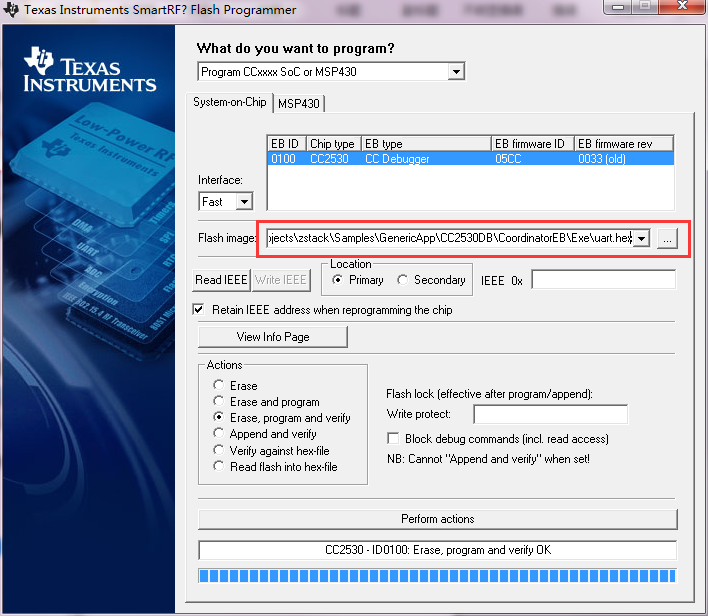
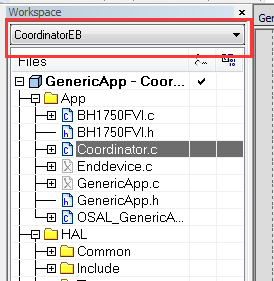
**下载程序：**

**准备两个Zigbee模块和两块底板，一颗3.7V AAA电池；**

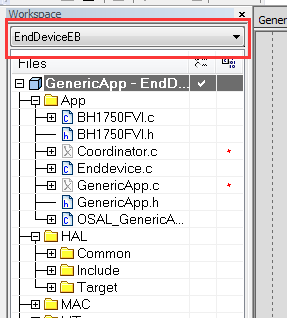
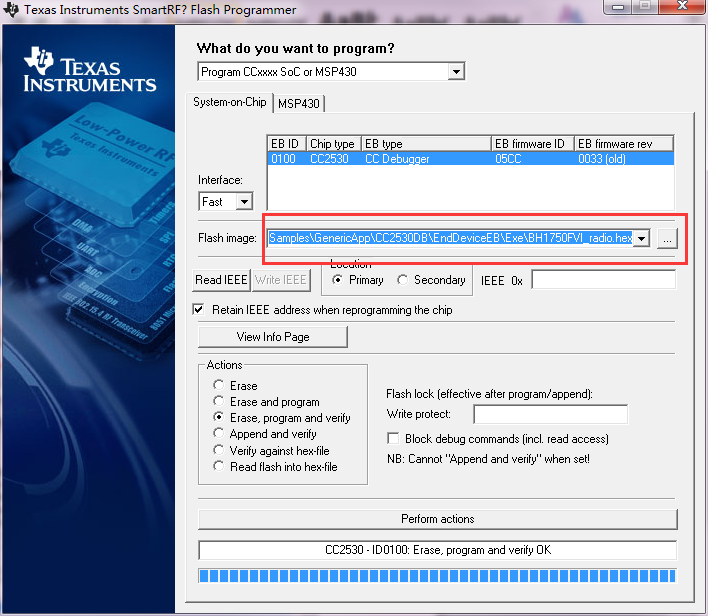
**第一块底板插上一个Zigbee模块，并且用B型USB线把串口连接到电脑的USB口。**

**第二块底板插上一个Zigbee模块，插上传感器模块，并且安装好一颗3.7V AAA电池。**

**第一块底板下载协调器所编译的程序:**

****

**第二块底板下载终端节点所编译的程序:**

** **

**下载完后两块板子上电运行，打开串口调试助手，设置串口参数后打开串口，即可看到输出的数据：**

****