**可燃气体/烟雾传感器使用实验**

1. **实验目的**

该实验主要是让学生熟悉如何使用Zigbee协议监测可燃气体/烟雾传感器数据。

1. **实验设备**

硬件：华智Zigbee底板（含Zigbee模块）两块，可燃气体/烟雾传感器一个，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。

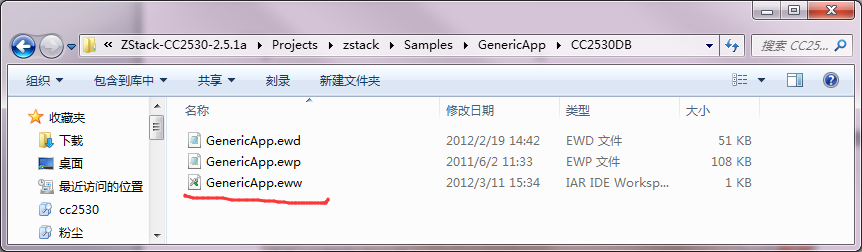
软件：IAR开发环境，HEX文件下载程序Flah Programmer， Zigbee协议栈（2.5.1a），相关实验例程，串口调试助手。

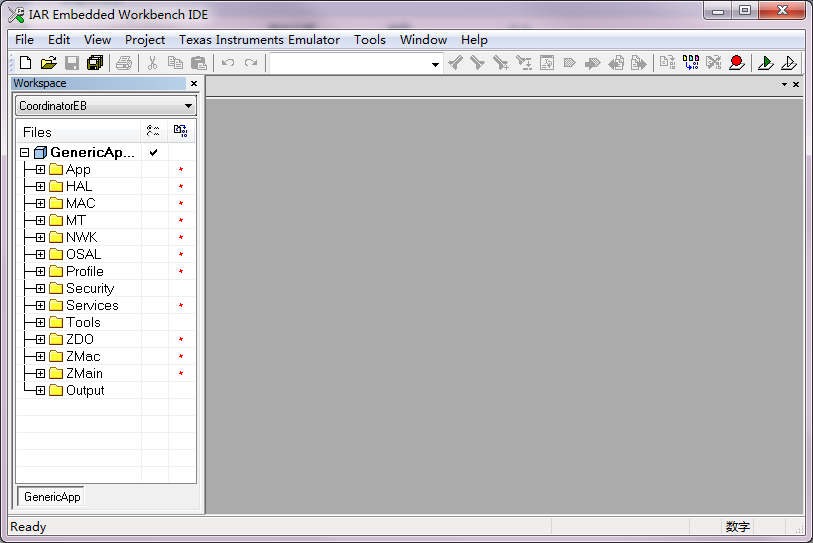
1. **实验原理**

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，终端节点实时监测可燃气体/烟雾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。
3. 终端节点实时监测可燃气体/烟雾传感器数据，并且通过串口输出到PC机的串口调试助手上显示。
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。
5. **实验步骤及相关代码**
6. Zigbee协议控制串口与PC机通信：

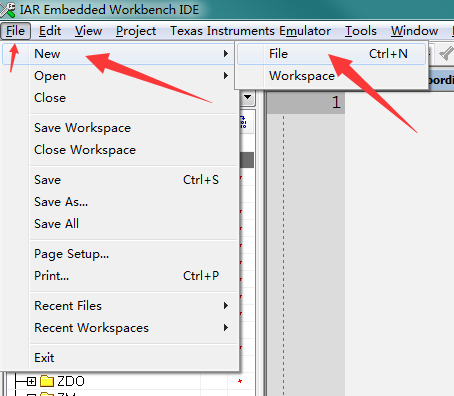
安装好协议栈并找到工程文件（详细步骤请参考《Zigbee无线网络实验指导书》相关章节），并打开：



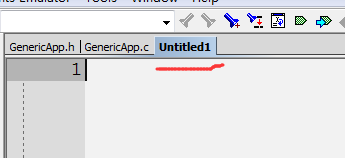


创建我们自己的程序文件，并且添加到App目录下，操作如下：

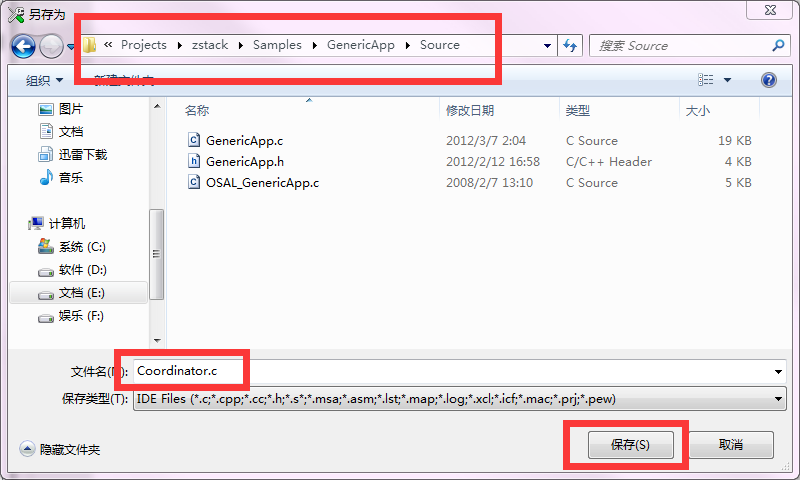
选择以下选项

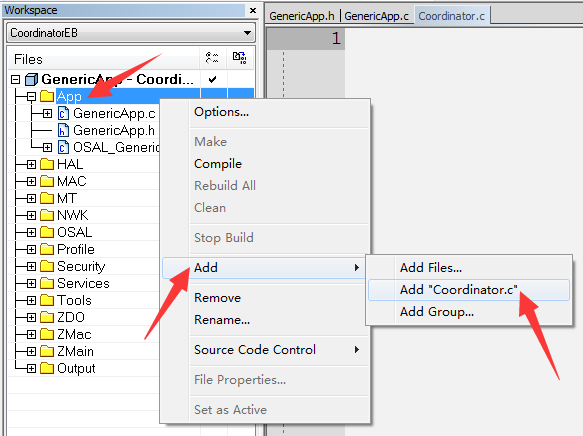


新建了一个文件

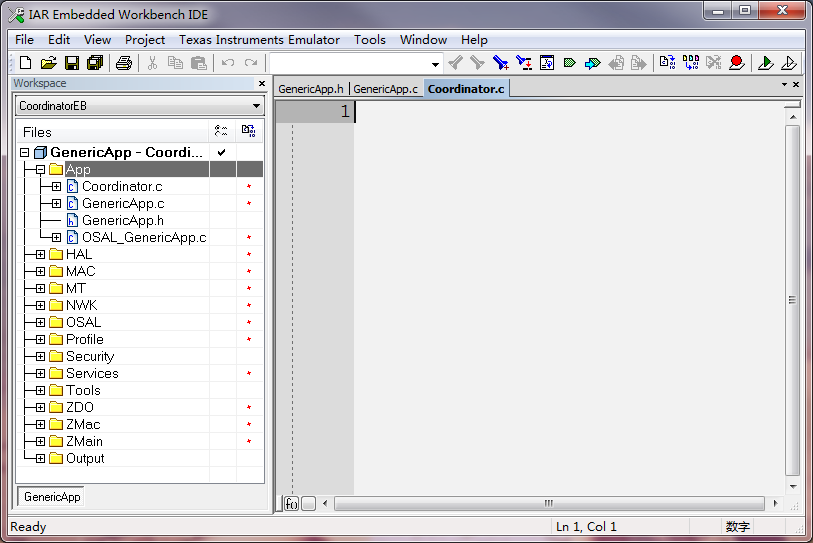


点击保存按钮，选择保存路径为以下路径，并且文件命名为Coordinator.c （协调器）：





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)

// {

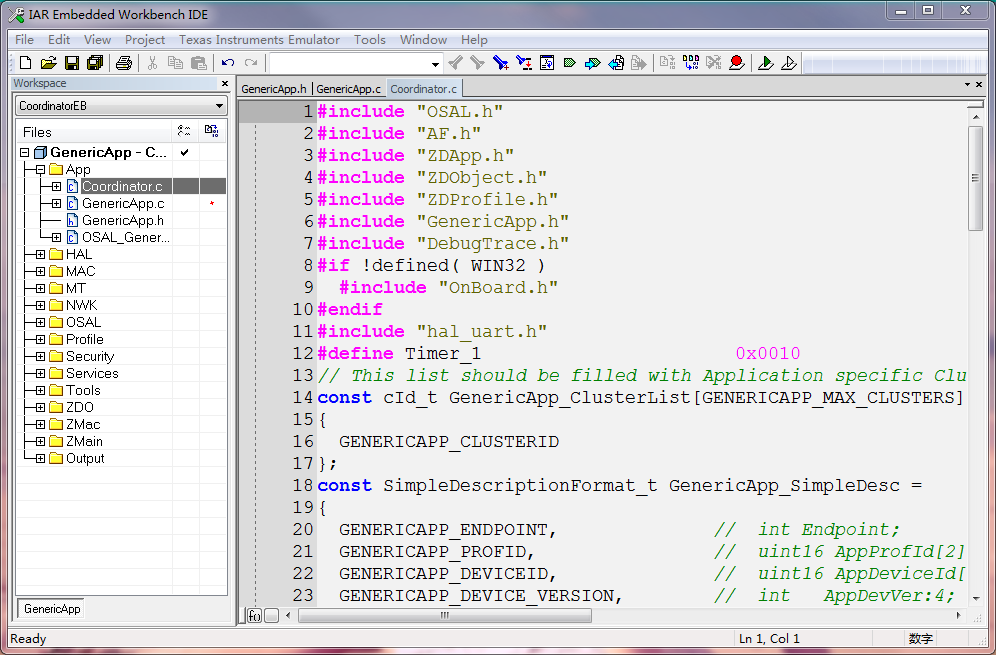
// HalUARTRead(0,uartbuf,rxlen); //从串口读取数据放在uartbuf缓冲区中

// 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

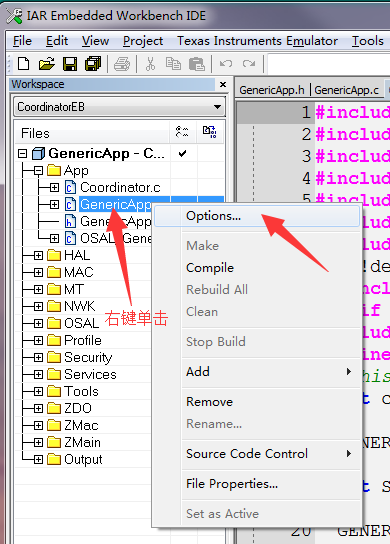
// }

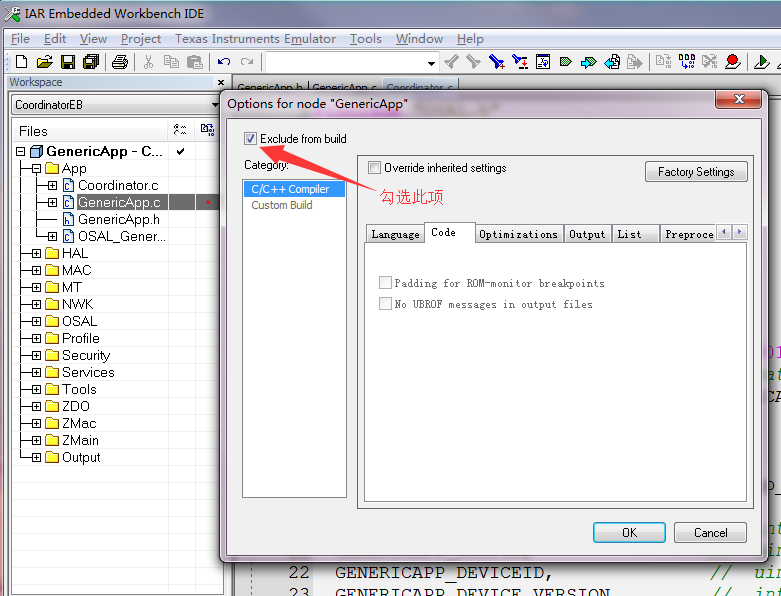
}

添加后如图：

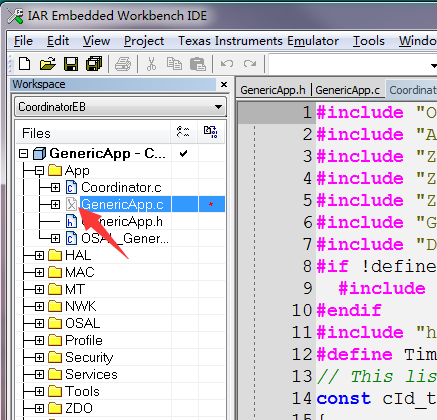


把GenericApp.c文件去掉编译：



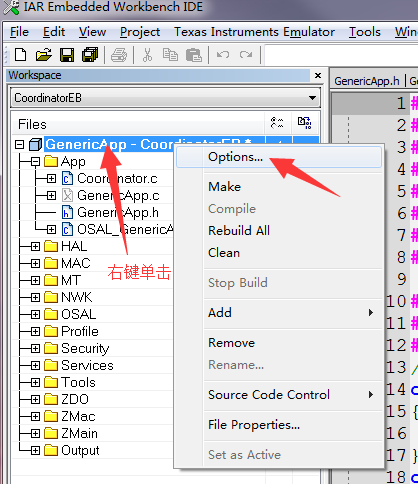


此时该文件的图标应变为失效图标：

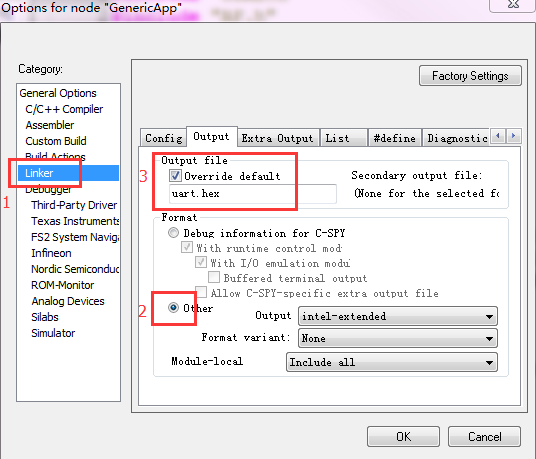


下来更改生成HEX文件选项。

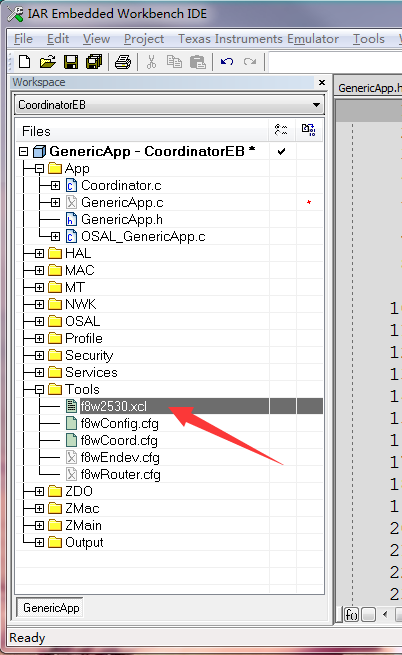
右键单击工程名，选择Options选项：



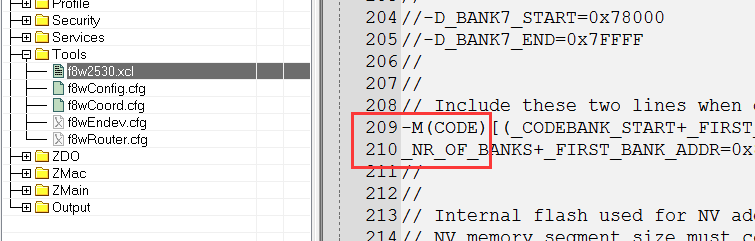
做如下修改后点击OK：



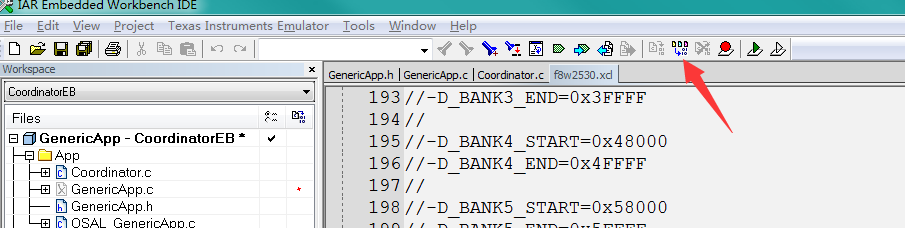
打开工程文件夹Tools下的f8w2530.xcl文件：



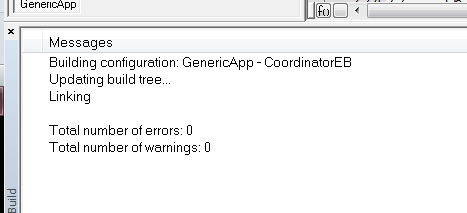
找到该文件第209、210行代码，并且把该两行代码前面的双斜杠去掉：



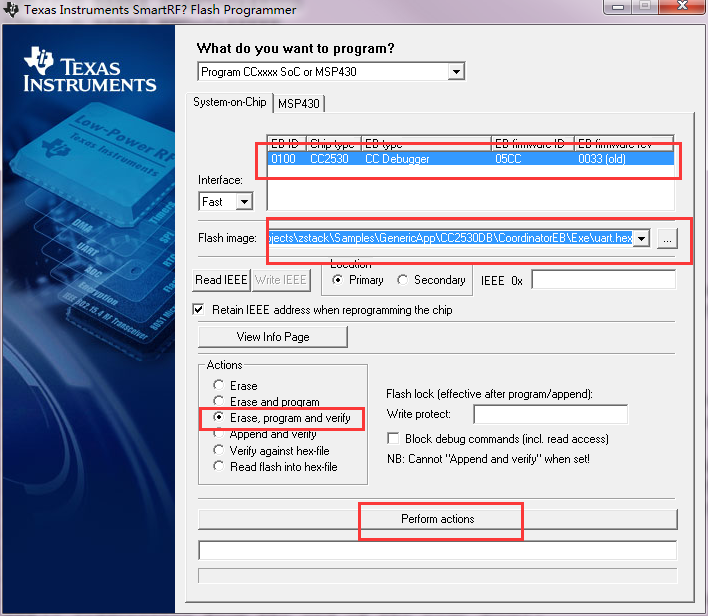
修改完成，点击编译按钮：



编译成功的提示为：

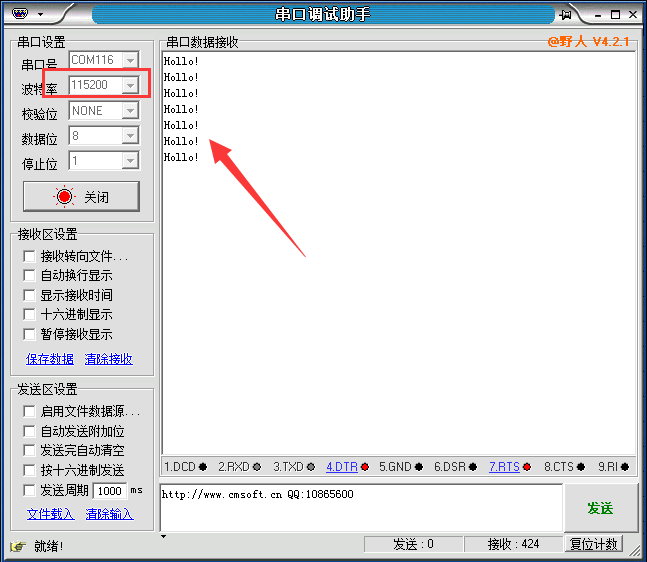


把CC2530模块插入下载底板，连上下载器，打开下载软件：



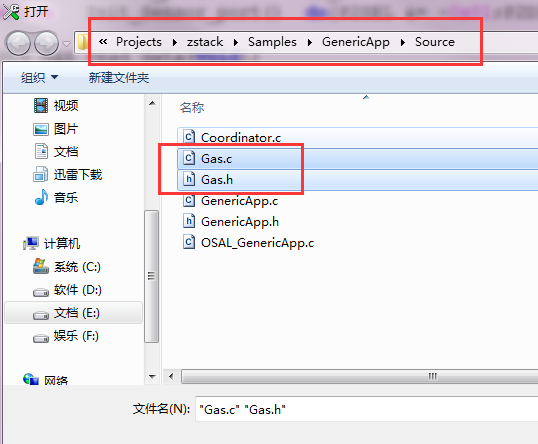
选择好后点击Perform actions 下载程序。

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：

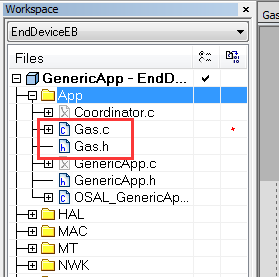


1. 终端节点监测可燃气体/烟雾传感器数据，并且通过串口输出到PC机的串口调试助手上显示。

新建两个文件，分别命名为Gas.c和Gas.h，保存到Source目录下：



并且把这两个文件添加到工程App目录下：



打开这两个文件，添加如下代码：

Gas.c的代码：

#include "Gas.h"

uint8 Gas\_read\_data(void)

{

Init\_Sensor\_port();//初始化传感器端口

return Sensor\_port;//返回传感器端口电平状态

}

Gas.h的代码：

#include <ioCC2530.h>

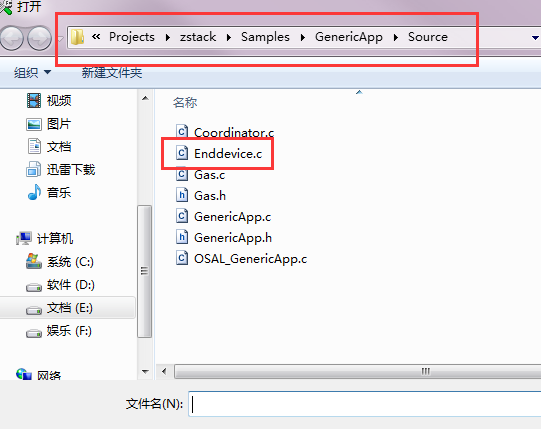
#include "GenericApp.h"

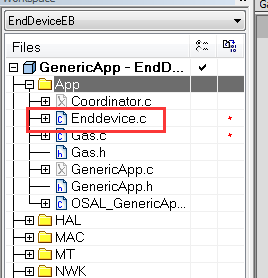
#define Sensor\_port P2\_0

#define Init\_Sensor\_port() do{P2SEL &= ~0x01;P2DIR &= ~0x01;}while(0) //P2\_0为普通io口并且设置为输入

uint8 Gas\_read\_data(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 "Gas.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\_1,1000); //1000MS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

{

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

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

}

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)

{

if(Gas\_read\_data() == 1)

{

HalUARTWrite(0,"瓦斯泄漏！",10);

}

osal\_start\_timerEx(GenericApp\_TaskID,Timer\_1,200); //每200MS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)

// {

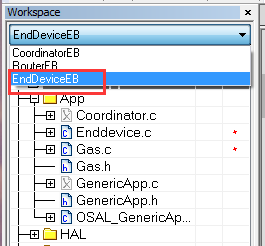
// HalUARTRead(0,uartbuf,rxlen); //从串口读取数据放在uartbuf缓冲区中

// 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

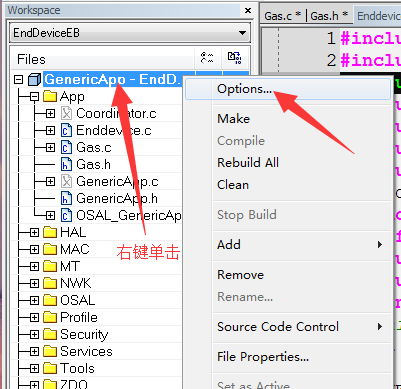
// }

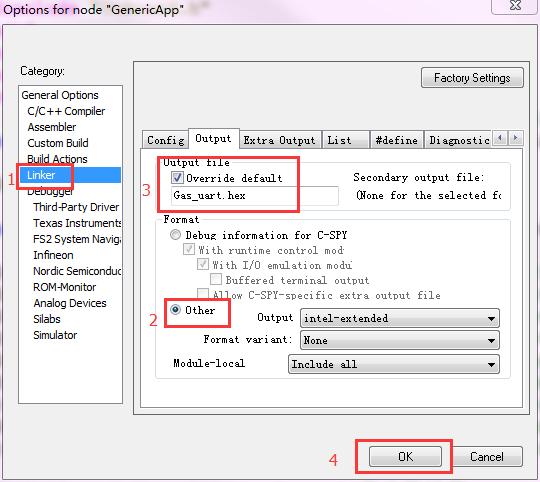
}

选择工程为EndDeviceEB选项：



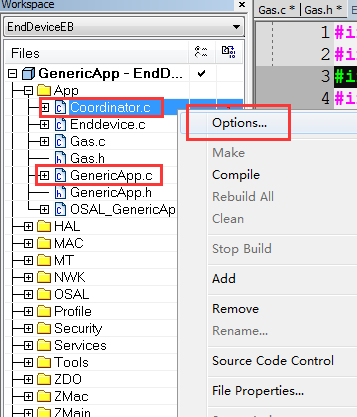
并且配置工程的编译输出选项：

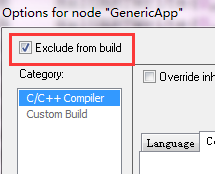


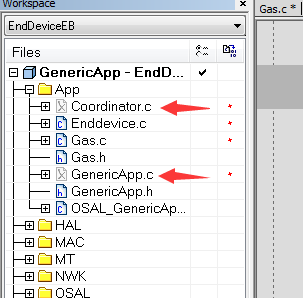


修改工程的可编译文件：

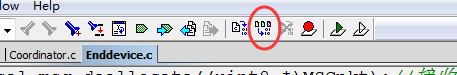
右键相关文件选择设置，勾选不参与编译选项：



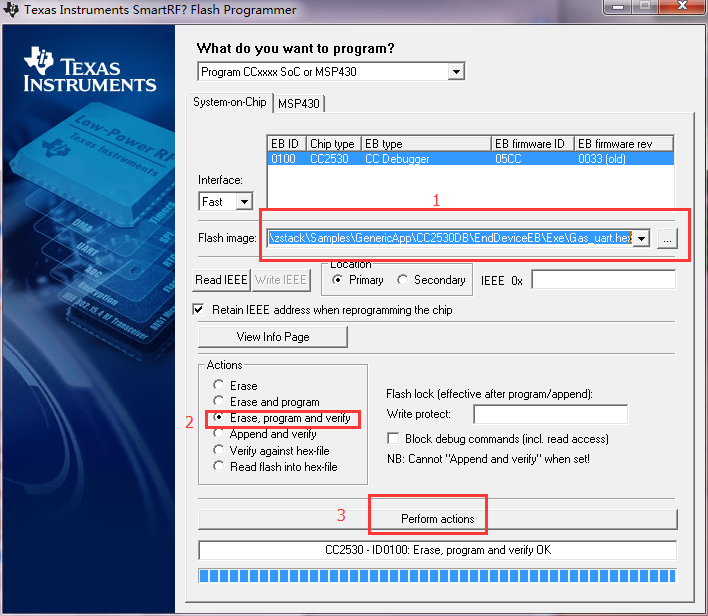




点击编译按钮：



编译完成后下载程序：

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，开始无数据，用一个打火机对着传感器探头喷气，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

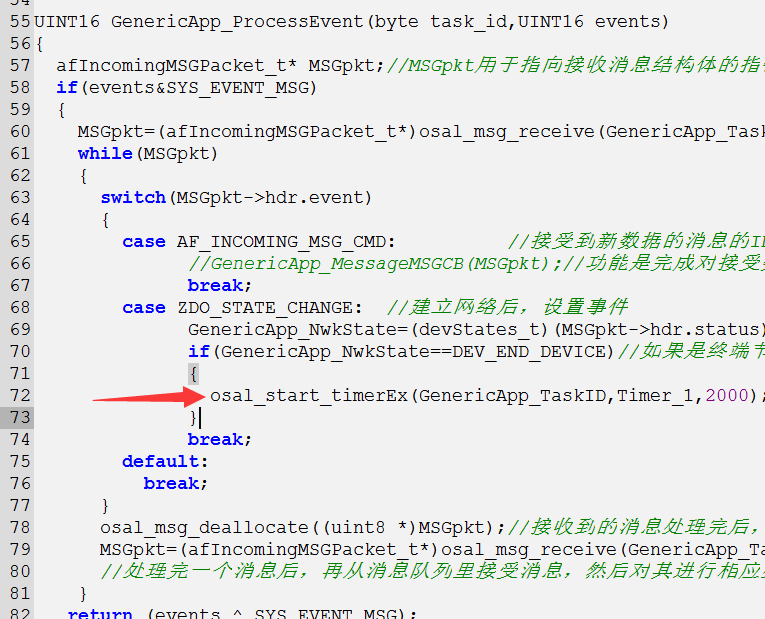
HalUARTOpen(0,&uartConfig); //串口打开

}

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)

{

if(Gas\_read\_data() == 1)

{

uint8 data[10] = {"瓦斯泄漏！"};

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);

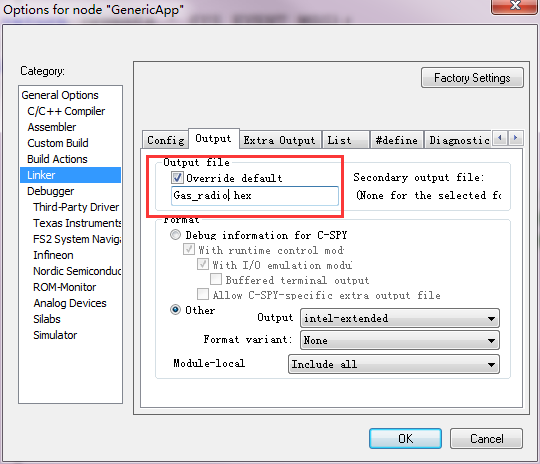
}

osal\_start\_timerEx(GenericApp\_TaskID,Timer\_1,200); //每200MS执行一次

return (events^Timer\_1);

}

修改生成的HEX文件名称：

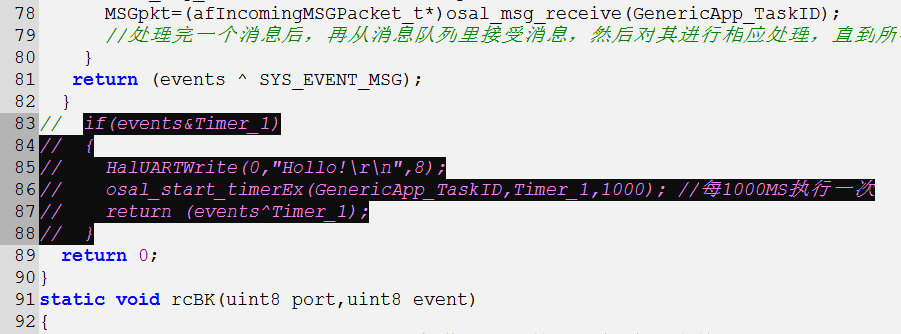


编译无错误即可。

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[10];

switch(pkt->clusterId)

{

case GENERICAPP\_CLUSTERID:

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

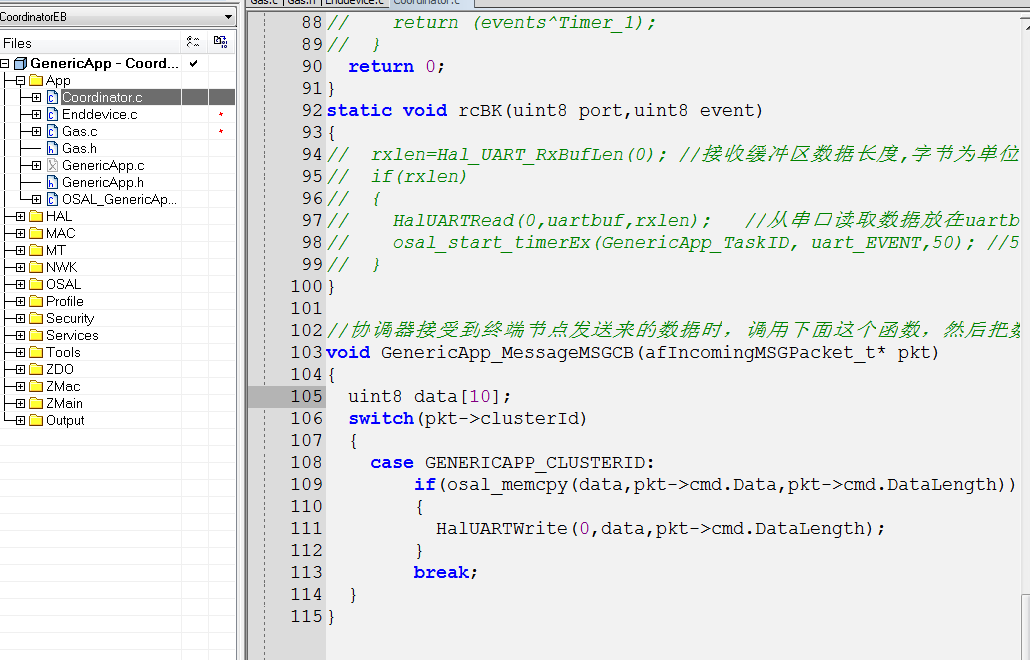
{

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

}

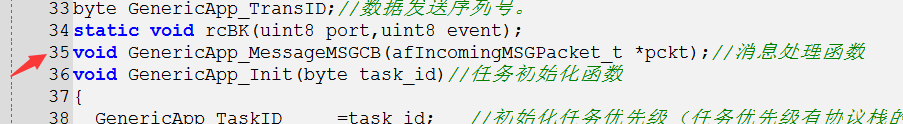
break;

}

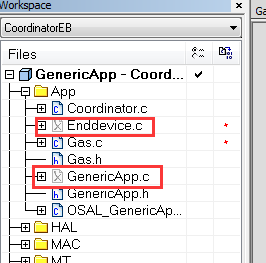
} } 

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：



编译无错误即可。

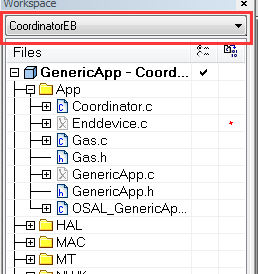
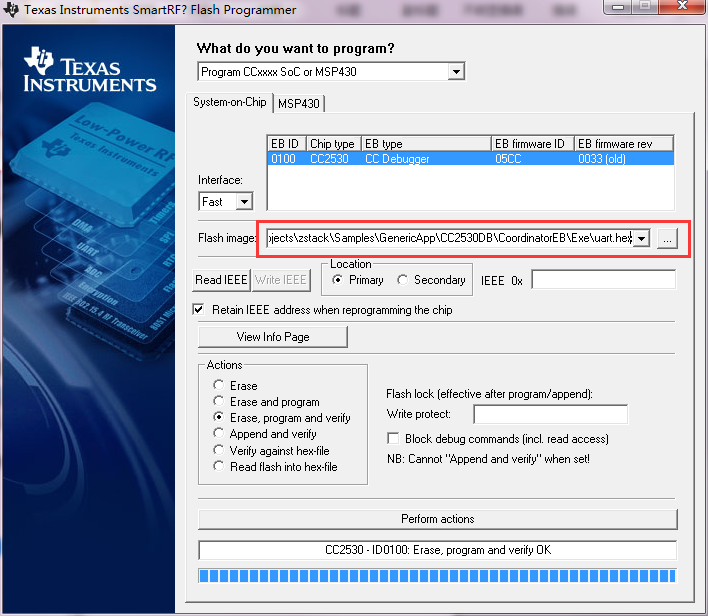
下载程序：

准备两个Zigbee模块和两块底板，一颗3.7V AAA电池；

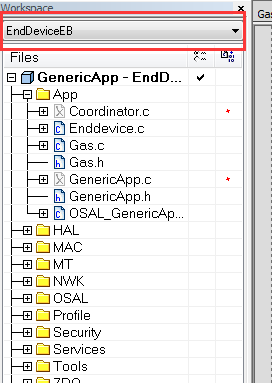
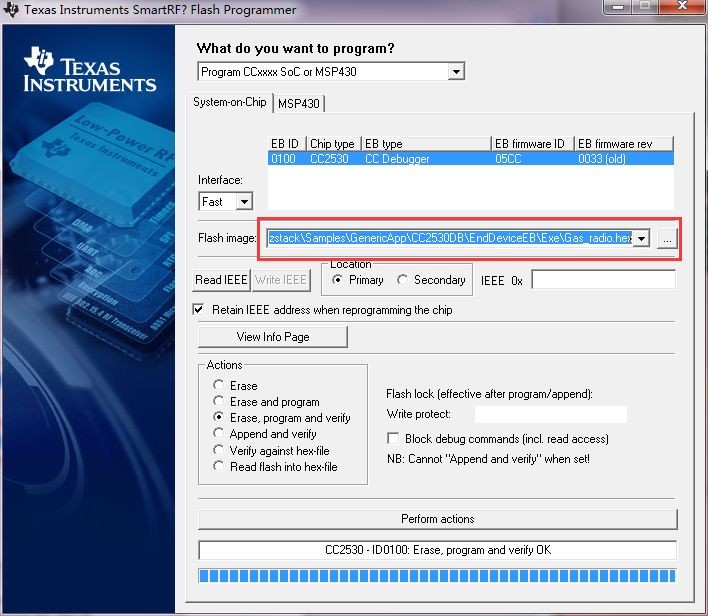
第一块底板插上一个Zigbee模块，并且用B型USB线把串口连接到电脑的USB口。

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，用一个打火机对着传感器探头喷气，即可看到输出的数据：

