**ZigBee通信串口实验——实验报告**

**（项目编号：07012024学时：2）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程** | 物联网传输技术 | **实验项目** | ZigBee通信串口实验 | **成绩** |  |
| **专业班级** | 15级物联网工程 | **学号** |  | **批阅日期** |  |
| **姓名** |  | **实验日期** | 2017/11/29 | **指导教师** | 邹正伟 |

**一【实验目的】**

1. 熟悉Zigbee协议栈Z-Stack
2. 掌握串口通信原理与方法

3、掌握编写协调器节点与上位机串口通信编程、串口设置方法

**二【实验内容】**

**1、**PC端串口调试助手向板子发送数据，板子接受到数据后，再把数据发送回给PC端串口调试助手。

**2、工具/原料**

* IAR Embedded Workbench for MCS-51
* CC2530 Zigbee开发套件
* CCDebuger调试器
* 串口调试助手

**3、方法/步骤**

**3.1新建工程**

新建工程，不知道如何配置的可以查看《IAR如何建立工程》学习如何建立、配置、编译、调试嵌入式系统。

**3.2相关知识**

串口发送接受数据的基本步骤：初始化串口（设置波特率、中断等）、向缓冲区发送数据或者从接受缓冲区读取数据。

       然而，上面的步骤都是以前不带操作系统单片机的步骤，而在OSAL中已经实现了串口的读取函数和写入函数。可以作为API一样使用。

       与串口相关的三个API函数：

      uint8    HalUARTOpen(uint8  port,halUARTCfg\_t \*    config);

      uint16  HalUARTRead(uint8 port,uint8 \*buf,uint16 len);

      uint16  HalUARTWrite(uint8 port,uint8\* buf,uint16  len);

**3.3完整代码**

**//Coordinator.c**

#include "OSAL.h"

#include "AF.h"

#include "ZDApp.h"

#include "ZDObject.h"

#include "ZDProfile.h"

#include <string.h>

#include "Coordinator.h"

#include "DebugTrace.h"

#if !defined(WIN32)

#include "OnBoard.h"

#endif

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

#include "hal\_uart.h"

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

{

  GENERICAPP\_CLUSTERID

};

//简单设备描述符（描述一个ZigBee设备节点）

const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc=

{

  GENERICAPP\_ENDPOINT,

  GENERICAPP\_PROFID,

  GENERICAPP\_DEVICEID,

  GENERICAPP\_DEVICE\_VERSION,

  GENERICAPP\_FLAGS,

  GENERICAPP\_MAX\_CLUSTERS,

 (cId\_t\*)GenericApp\_ClusterList, //

  0,

  (cId\_t \*)NULL

};

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

byte GenericApp\_TaskID;//任务优先级

byte GenericApp\_TransID;//数据发送序列号。

//unsigned char uartbuf[128];//串口接收发送数据缓冲单元

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

void GenericApp\_SendTheMessage(void);//数据发送函数

//static void rxCB(uint8 port,uint8 envent);//

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。

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

  //串口的初始化

  uartConfig.configured   =TRUE;

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

  uartConfig.flowControl  =FALSE;             //流控制

  //uartConfig.callBackFunc =rxCB;             //填的是回调函数 ，数的指针（即函数的地址）作为参数传递给另一个函数，

  //其实callBackFunc是一个函数指针，它的定义为halUARTCBack\_t callBackFunc;

  //而halUARTCBack\_t的定义为 typed void (\*halUARTCBack\_t)(uint8 port,uint8 envent) 定义的是一个函数指针

  uartConfig.callBackFunc =NULL; //本实验就不用回调函数了！！！

**HalUARTOpen(0,&uartConfig);**                 //串口是否打开

}

//下面这个是回调函数，回电函数就是一个通过函数指针（函数地址）调用的函数，如果把函数的指针（即函数的地址）作为参数传递给另一个函数，当通过该指正调用它锁指向的函数时，称为函数的回调。

//回调函数不是有该函数的实现方直接调用的，而是在特定的事件或条件时，由另一方调用的额，用于对该事件或条件进行响应。

//回调函数机制提供了系统对异步事件的处理能力。

/\*static void rxCB(uint8 port,uint8 envent)

{

       HalLedBlink(HAL\_LED\_2,0,50,500);    //LED1 闪烁

     HalUARTRead(0,uartbuf,10);   //从串口读取数据放在uartbuf缓冲区中

 if(osal\_memcmp(uartbuf,"abcdefghij",10))//判断接受到的数据是否是www.wlwmaker.com，如果是，函数返回TURE

 {

       HalLedBlink(HAL\_LED\_1,0,50,500);    //LED2 闪烁

       HalUARTWrite(0,uartbuf,10); //将接收到的数字输出到串口

 }

}\*/

//消息处理函数

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

                                       //接受到的是无线数据包

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

      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);

  }

   return 0;

}

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

{

  unsigned **char** buffer[10];

 switch(pkt->clusterId)

 {

 case GENERICAPP\_CLUSTERID:

   osal\_memcpy(buffer,pkt->cmd.Data,10);//把pkt->cmd.Data的数据复制到buffer

**HalUARTWrite(0,buffer,10);**

   break;

 }

}

//Coordinator.c

#include "OSAL.h"

#include "AF.h"

#include "ZDApp.h"

#include "ZDObject.h"

#include "ZDProfile.h"

#include <string.h>

#include "Coordinator.h"

#include "DebugTrace.h"

#if !defined(WIN32) //????

#include "OnBoard.h"

#endif

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

#include "hal\_uart.h"

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

{

GENERICAPP\_CLUSTERID

};

//简单设备描述符（描述一个ZigBee设备节点）

const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc=

{

GENERICAPP\_ENDPOINT,

GENERICAPP\_PROFID,

GENERICAPP\_DEVICEID,

GENERICAPP\_DEVICE\_VERSION,

GENERICAPP\_FLAGS,

GENERICAPP\_MAX\_CLUSTERS,

(cId\_t\*)GenericApp\_ClusterList, //?????

0,

(cId\_t \*)NULL

};

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

byte GenericApp\_TaskID;//任务优先级

byte GenericApp\_TransID;//数据发送序列号。

//unsigned char uartbuf[128];//串口接收发送数据缓冲单元

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

void GenericApp\_SendTheMessage(void);//数据发送函数

//static void rxCB(uint8 port,uint8 envent);//???????????

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。

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

//串口的初始化

uartConfig.configured =TRUE;

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

uartConfig.flowControl =FALSE; //流控制

//uartConfig.callBackFunc =rxCB; //填的是回调函数 ，数的指针（即函数的地址）作为参数传递给另一个函数，

//其实callBackFunc是一个函数指针，它的定义为halUARTCBack\_t callBackFunc;

//而halUARTCBack\_t的定义为 typed void (\*halUARTCBack\_t)(uint8 port,uint8 envent) 定义的是一个函数指针

uartConfig.callBackFunc =NULL; //本实验就不用回调函数了！！！

HalUARTOpen(0,&uartConfig); //串口是否打开

}

//下面这个是回调函数，回电函数就是一个通过函数指针（函数地址）调用的函数，如果把函数的指针（即函数的地址）作为参数传递给另一

//个函数，当通过该指正调用它锁指向的函数时，称为函数的回调。

//回调函数不是有该函数的实现方直接调用的，而是在特定的事件或条件时，由另一方调用的额，用于对该事件或条件进行响应。

//回调函数机制提供了系统对异步事件的处理能力。

/\*static void rxCB(uint8 port,uint8 envent)

{

HalLedBlink(HAL\_LED\_2,0,50,500); //LED1 闪烁

HalUARTRead(0,uartbuf,10); //从串口读取数据放在uartbuf缓冲区中

if(osal\_memcmp(uartbuf,"abcdefghij",10))//判断接受到的数据是否是www.wlwmaker.com，如果是，函数返回TURE

{

HalLedBlink(HAL\_LED\_1,0,50,500); //LED2 闪烁

HalUARTWrite(0,uartbuf,10); //将接收到的数字输出到串口

}

}\*/

//消息处理函数

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

//接受到的是无线数据包

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

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);

}

return 0;

}

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

{

unsigned char buffer[10];

switch(pkt->clusterId)

{

case GENERICAPP\_CLUSTERID:

osal\_memcpy(buffer,pkt->cmd.Data,10);//把pkt->cmd.Data的数据复制到buffer

HalUARTWrite(0,buffer,10);

break;

}

}

**终端节点代码：**在实验2的基础上修改。代码如下：

//Enddevice.c

#include "OSAL.h"

#include "AF.h"

#include "ZDApp.h"

#include "ZDObject.h"

#include "ZDProfile.h"

#include <string.h>

**#include "Coordinator.h"**

#include "DebugTrace.h"

#if !defined(WIN32)

#include "OnBoard.h"

#endif

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

#include "hal\_uart.h"

#define SEND\_DATA\_EVENT 0x01  //发送事件id

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

{

 GENERICAPP\_CLUSTERID

};

 //初始化端口描述符

const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc=

{

  GENERICAPP\_ENDPOINT,

  GENERICAPP\_PROFID,

  GENERICAPP\_DEVICEID,

  GENERICAPP\_DEVICE\_VERSION,

  GENERICAPP\_FLAGS,

  0,

  (cId\_t\*)NULL,

  GENERICAPP\_MAX\_CLUSTERS,

  (cId\_t\*)GenericApp\_ClusterList

};

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

byte GenericApp\_TaskID;          //任务优先级

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

devStates\_t GenericApp\_NwkState;//保存节点状态

void GenericApp\_MessageMSGCB(afIncomingMSGPacket\_t\* pckt);//消息处理函数的声明

void GenericApp\_SendTheMessage(void); //数据发送函数的声明

//任务初始化函数

void GenericApp\_Init(byte task\_id)

{

  GenericApp\_TaskID     = task\_id;//初始化任务优先级

  GenericApp\_NwkState   =DEV\_INIT; //初始化为DEV\_INIT,表节点没有连接到ZigBee网络

  GenericApp\_TransID    =0;        //发送数据包的序列号初始化为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()函数将节点描述符进行注册，注册后才可以使用OSAL提供的系统服务

**afRegister**(&GenericApp\_epDesc);

}

//消息处理函数

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

{

  afIncomingMSGPacket\_t\* MSGpkt;

  if(events&SYS\_EVENT\_MSG)

  {

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

    while(MSGpkt)

    {

      switch(MSGpkt->hdr.event)

      {

    case **ZDO\_STATE\_CHANGE**:

      GenericApp\_NwkState=(devStates\_t)(MSGpkt->hdr.status);//读取节点的设备类型

      if(GenericApp\_NwkState==DEV\_END\_DEVICE)

      {

        //当中断节点加入网络后使用osal\_set\_envent()函数设置SEND\_DATA\_EVENT事件，当事件发生时，执行事件处理函数

        osal\_set\_event(GenericApp\_TaskID,SEND\_DATA\_EVENT);//添加用户自定义任务事件（以任务号对应事件的形式添加）

        //GenericApp\_SendTheMessage(); //终端节点类型，执行无线数据发送

      }

      break;

    default:

      break;

      }

    osal\_msg\_deallocate((uint8\*)MSGpkt);

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

    }

    return (events^SYS\_EVENT\_MSG);

  }

if(events&SEND\_DATA\_EVENT)//这个函数为什么放在这里，好好想想？？（区分事件 ）

{

GenericApp\_SendTheMessage();

**osal\_start\_timerEx**(GenericApp\_TaskID,SEND\_DATA\_EVENT,1000);

//定时器，三个参数：

//一参：表定时事件到后，那个任务对其作出响应

//二参：表时间达到后，什么事件发生，该事件的处理函数中实现数据的发送（用户应用）。

//三参：定时的时间数量，单位毫秒。

**return (events^SEND\_DATA\_EVENT);**//清除事件标志（该事件已被处理）

}

  return 0;

}

void GenericApp\_SendTheMessage(void)

{

  unsigned **char** theMessageData[10]="EndDevice";//存放发送数据

  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                                                  长度  数据发送缓冲区

**AF\_DataRequest(**&my\_DstAddr,&GenericApp\_epDesc,GENERICAPP\_CLUSTERID,osal\_strlen("EndDevice")+1,theMessageData,&GenericApp\_TransID,AF\_DISCV\_ROUTE,AF\_DEFAULT\_RADIUS);

//Enddevice.c

#include "OSAL.h"

#include "AF.h"

#include "ZDApp.h"

#include "ZDObject.h"

#include "ZDProfile.h"

#include <string.h>

#include "Coordinator.h"

#include "DebugTrace.h"

#if !defined(WIN32)

#include "OnBoard.h"

#endif

#include "hal\_lcd.h"

#include "hal\_led.h"

#include "hal\_key.h"

#include "hal\_uart.h"

#define SEND\_DATA\_EVENT 0x01 //发送事件id

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

{

GENERICAPP\_CLUSTERID

};

//初始化端口描述符

const SimpleDescriptionFormat\_t GenericApp\_SimpleDesc=

{

GENERICAPP\_ENDPOINT,

GENERICAPP\_PROFID,

GENERICAPP\_DEVICEID,

GENERICAPP\_DEVICE\_VERSION,

GENERICAPP\_FLAGS,

0,

(cId\_t\*)NULL,

GENERICAPP\_MAX\_CLUSTERS,

(cId\_t\*)GenericApp\_ClusterList

};

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

byte GenericApp\_TaskID; //任务优先级

byte GenericApp\_TransID; //数据发送序列号

devStates\_t GenericApp\_NwkState;//保存节点状态

void GenericApp\_MessageMSGCB(afIncomingMSGPacket\_t\* pckt);//消息处理函数的声明

void GenericApp\_SendTheMessage(void); //数据发送函数的声明

//任务初始化函数

void GenericApp\_Init(byte task\_id)

{

GenericApp\_TaskID = task\_id;//初始化任务优先级

GenericApp\_NwkState =DEV\_INIT; //初始化为DEV\_INIT,表节点没有连接到ZigBee网络

GenericApp\_TransID =0; //发送数据包的序列号初始化为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()函数将节点描述符进行注册，注册后才可以使用OSAL提供的系统服务

afRegister(&GenericApp\_epDesc);

}

//消息处理函数

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

{

afIncomingMSGPacket\_t\* MSGpkt;

if(events&SYS\_EVENT\_MSG)

{

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

while(MSGpkt)

{

switch(MSGpkt->hdr.event)

{

case ZDO\_STATE\_CHANGE:

GenericApp\_NwkState=(devStates\_t)(MSGpkt->hdr.status);//读取节点的设备类型

if(GenericApp\_NwkState==DEV\_END\_DEVICE)

{

//当中断节点加入网络后使用osal\_set\_envent()函数设置SEND\_DATA\_EVENT事件，当事件发生时，执行事件处理函数

osal\_set\_event(GenericApp\_TaskID,SEND\_DATA\_EVENT);//??????????????????????????

//GenericApp\_SendTheMessage(); //终端节点类型，执行无线数据发送

}

break;

default:

break;

}

osal\_msg\_deallocate((uint8\*)MSGpkt);

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

}

return (events^SYS\_EVENT\_MSG);

}

if(events&SEND\_DATA\_EVENT)//这个函数为什么放在这里，好好想想才行？？？？

{

GenericApp\_SendTheMessage();

osal\_start\_timerEx(GenericApp\_TaskID,SEND\_DATA\_EVENT,1000);

//定时器，三个参数：一参：表定时事件到底后，那个任务对其作出响应

//二参：时间ID，表时间达到后，事件发生，该事件的处理函数中实现数据的发送。

//三参：定时的时间数量，单位毫秒。

return (events^SEND\_DATA\_EVENT);//清除事件标志

}

return 0;

}

void GenericApp\_SendTheMessage(void)

{

unsigned char theMessageData[10]="EndDevice";//存放发送数据

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 长度 数据发送缓冲区

AF\_DataRequest(&my\_DstAddr,&GenericApp\_epDesc,GENERICAPP\_CLUSTERID,\

osal\_strlen("EndDevice")+1,theMessageData,&GenericApp\_TransID,AF\_DISCV\_ROUTE,AF\_DEFAULT\_RADIUS);

//osal\_strlen("EndDevice")+1函数返回字符串实际长度，但发送的数据时，是要讲字符串的几位字符一起发送。所以这里要加1

      HalLedBlink(HAL\_LED\_1,0,50,500);    //LED2 闪烁

      HalLedBlink(HAL\_LED\_2,0,50,500);    //LED2 闪烁

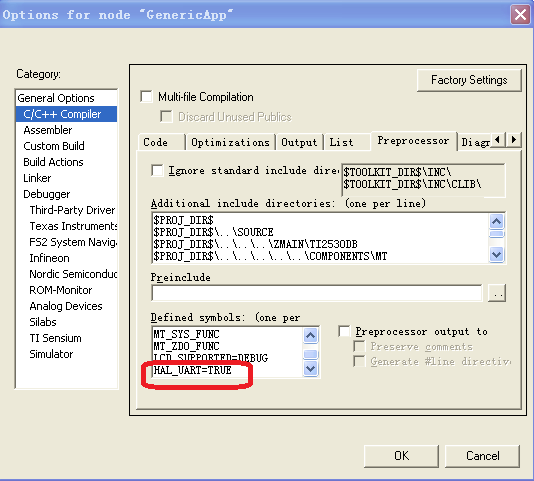
}

**2.4下载调试**

**注意：**项目配置

配置完成，选择对应的调试模块（按键选择），下载调试（调试器灯为红色，调试器工作正常，可按“reset”键重置；若调试器工作不正常，请查看电脑“设备管理”，卸载并重新安装调试器驱动）。

编译过程中别忘了设置下面一步（这种叫做条件编译，用来控制不同的模块是否参与编译，以达到最大程度地节省存储资源资源）



       PC端调试助手设置如下：

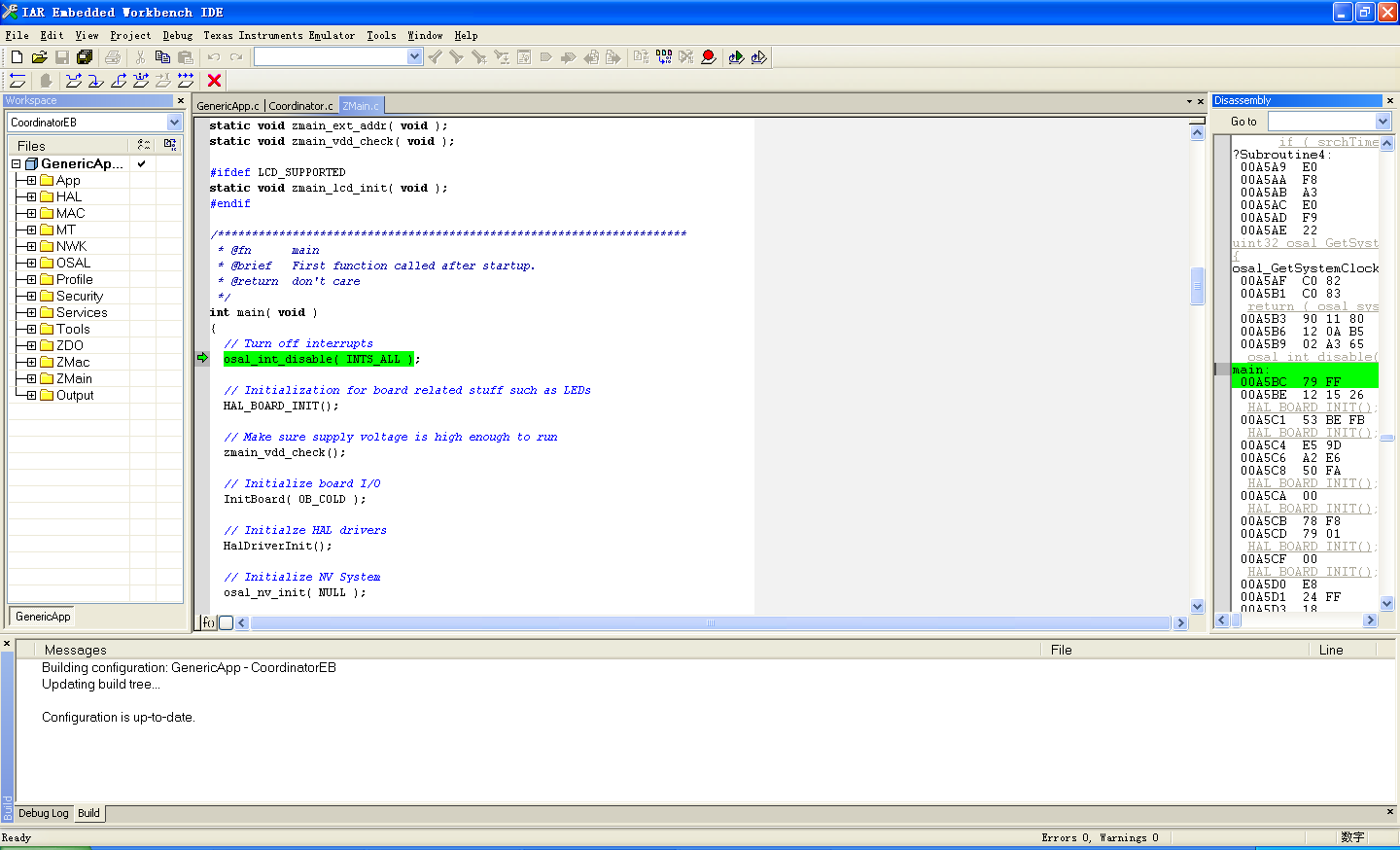
注意串口号和波特率与实际程序**匹配**



运行程序，完成实验要求。

**实验截图：**

**IAR截图：**



**上位机运行截图：**

**总结：**

通过以上实验，了解并熟悉CC2530开发板；熟悉了IAR集成开发环境；能掌握开发方法和步骤。

**思考：**