**内部事项**

Wulian ZigBee产品

软件开发指南

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件编号： | XX-XX-XX-XX |
| 当前版本： | V0.5 |
| 作者： | 余峥峥 |
| 完成日期： | 2016/8/3 |

南京物联传感技术有限公司

# 版本变更记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修订内容 | 修订部门/修订人 | 修订日期 |
| V0.1 | 初始版本 | 余峥峥 | 2015/08/29 |
| V0.2 | 结构框架更新升级 | 余峥峥 | 2015/09/05 |
| V0.3 | 完善应用程序API接口和一般性示例编写 | 余峥峥 | 2015/12/15 |
| V0.4 | 完善常用API说明，  增加常用开发事项说明 | 余峥峥 | 2015/07/04 |
| V0.5 | 增加新的功能，修改设备入网的一下说明 | 余峥峥 | 2015/08/02 |

# 目录

[Wulian ZigBee产品 1](#_Toc455429190)

[软件开发指南 1](#_Toc455429191)

[版本变更记录 2](#_Toc455429192)

[目录 3](#_Toc455429193)

[1 概述 6](#_Toc455429194)

[1.1 背景 6](#_Toc455429195)

[1.2 目的 6](#_Toc455429196)

[1.3 内容 6](#_Toc455429197)

[1.4 封装协议栈 6](#_Toc455429198)

[1.5 术语和缩略词 6](#_Toc455429199)

[2 工程结构 7](#_Toc455429200)

[2.1 一对一的工程结构 7](#_Toc455429201)

[2.2 一对多的工程结构 7](#_Toc455429202)

[2.2.1 同一类产品不同配置的工程 7](#_Toc455429203)

[2.2.2 同一产品不同功能的工程结构 9](#_Toc455429204)

[3 工程文件组织和编译配置选项 10](#_Toc455429205)

[3.1 工程文件组织 10](#_Toc455429206)

[3.2 编译配置选项 10](#_Toc455429207)

[4 创建项目工程 12](#_Toc455429208)

[4.1 创建项目文件 12](#_Toc455429209)

[4.2 修改工程文件 13](#_Toc455429210)

[4.3 增加编译配置宏和头文件 16](#_Toc455429211)

[5 通用API接口的说明 19](#_Toc455429212)

[5.1.1 void UserApp\_Init( uint8 task\_id ) 19](#_Toc455429213)

[5.1.2 uint16 UserApp\_ProcessEvent( uint8 task\_id, uint16 events ) 19](#_Toc455429214)

[5.1.3 void UserAppHandleKeys( uint8 shift, uint8 keys ) 20](#_Toc455429215)

[5.1.4 void UserAppProcessKeyNwkAction(void) 20](#_Toc455429216)

[5.1.5 void UserAppProcessUartIncoming(uint8\* msg\_ptr) 20](#_Toc455429217)

[5.1.6 void UserAppProcessCommandCallBack(uint8 inMsgLen, uint8 \*inMsg, uint8 \*outMsgLen, uint8 \*outMsg) 21](#_Toc455429218)

[5.1.7 void UserAppProcessReportingCommand(uint8 cmd ,uint8\*inMsg) 21](#_Toc455429219)

[5.1.8 void UserAppProcessBindingMessage(afIncomingMSGPacket\_t \* pkt) 22](#_Toc455429220)

[5.1.9 void UserAppStartNwkJoiningCallBack(void) 23](#_Toc455429221)

[5.1.10 void UserAppProcessZDOMsgs( zdoIncomingMsg\_t \*inMsg ) 23](#_Toc455429222)

[5.1.11 void UserAppCoordinatorOfflineCallback(bool offline) 23](#_Toc455429223)

[5.1.12 void UserAppJoinedNetworkCallBack(void) 23](#_Toc455429224)

[5.1.13 void UserAppRssiBlinkLed(wzclDevRssiRsp\_t \*pRsp) 23](#_Toc455429225)

[5.1.14 void UserAppExtendLightBlink(uint8 numblinks) 23](#_Toc455429226)

[5.1.15 void UserAppPreSendRsMessageCallBack( uint8 \* rsMsg ) 24](#_Toc455429227)

[5.1.16 void UserAppSetDevInfo(wzclDevInfo\_t \*devInfo) 24](#_Toc455429228)

[5.1.17 uint8 UserAppAutoStartNwk(void) 24](#_Toc455429229)

[5.1.18 void UserAppSetPollRate(wzclPollControl\_t \*pollrate) 25](#_Toc455429230)

[5.1.19 uint32 ChannelSelect(void) 25](#_Toc455429231)

[5.1.20 void WZCL\_SendAsMessage(wzclAsMsg\_t \*iMsg) 25](#_Toc455429232)

[5.1.21 void WZCL\_FindAndJoinNetwork(void) 25](#_Toc455429233)

[5.1.22 void WZCL\_LeaveAndRestoreFactory(void) 25](#_Toc455429234)

[5.1.23 void WZCL\_SendRsMessage(void); 25](#_Toc455429235)

[5.1.24 void WZCL\_SendDsMessage(uint8 \* iMsg, uint8 len) 25](#_Toc455429236)

[5.1.25 void WZCL\_SendDsWithAckMessage(uint8 \* iMsg, uint8 len) 25](#_Toc455429237)

[5.1.26 void WZCL\_SendBindingMessage(endPointDesc\_t \* srcEP,uint16 cID, uint16 len, uint8 \*buf) 26](#_Toc455429238)

[5.1.27 void WZCL\_InBindingReqMessage(uint8 endpoint ,cId\_t\* bindingInClusters,uint8 clusterlists) 26](#_Toc455429239)

[5.1.28 void WZCL\_OutBindingReqMessage(uint8 endpoint ,cId\_t\* bindingOutClusters,uint8 clusterlists) 26](#_Toc455429240)

[6 基本功能开发 27](#_Toc455429241)

[6.1 设备加网退网功能 27](#_Toc455429242)

[6.1.1 老产品加网退网方式 27](#_Toc455429243)

[6.1.1.1 快按SET键四次入网，长按SET键10s退网方式 27](#_Toc455429244)

[6.1.1.2 通用加网退网操作 27](#_Toc455429245)

[6.1.1.3 上电自动加网操作 28](#_Toc455429246)

[6.1.1.4 串口命令入网退网 28](#_Toc455429247)

[6.1.1.5 其他动作退网 28](#_Toc455429248)

[6.1.2 Top47产品以及新研产品加网退网方式 28](#_Toc455429249)

[6.1.2.1 上电自动入网，长按SET按键6s退网，不在网状态下按一下SET按键或者快按SET键4次申请加入网络 28](#_Toc455429250)

[6.1.2.2 通用加网退网操作 28](#_Toc455429251)

[6.1.2.3 上电自动加网操作 29](#_Toc455429252)

[6.1.2.4 串口命令入网退网 29](#_Toc455429253)

[6.1.2.5 其他动作退网 29](#_Toc455429254)

[6.2 指示灯闪烁 29](#_Toc455429255)

[6.3 设备参数配置 29](#_Toc455429256)

[6.3.1 EndDevice设备参数配置 29](#_Toc455429257)

[6.3.2 Router设备参数配置 30](#_Toc455429258)

[6.4 PA配置问题 30](#_Toc455429259)

[6.5 32.768kHz晶振 30](#_Toc455429260)

[6.6 看门狗问题 30](#_Toc455429261)

[6.7 产品生产测试（产测） 31](#_Toc455429262)

[6.7.1 进入产测 31](#_Toc455429263)

[6.7.2 产测信道选择 31](#_Toc455429264)

[6.7.3 EndDevice设备的产测 31](#_Toc455429265)

[6.7.3.1 Poll Rate规律 31](#_Toc455429266)

[6.7.3.2 产测时禁止一些功能 32](#_Toc455429267)

[6.8 链接库使用 32](#_Toc455429268)

[7 固件生成以及输出 33](#_Toc455429269)

[7.1 固件输出配置 33](#_Toc455429270)

[7.1.1 设置固件生成路径 33](#_Toc455429271)

[7.1.2 设置固件输出格式以及名称 34](#_Toc455429272)

[7.1.3 配置固件生成构建选项 35](#_Toc455429273)

[7.2 固件的生成 36](#_Toc455429274)

# 概述

## 背景

为了向软件开发工程师介绍如何基于ZigBee/SmartRoom封装协议栈来开发一款我司的应用产品。

## 目的

指导软件开发工程师如何基于ZigBee/SmartRoom封装协议栈的软件项目。

## 内容

本软件开发指南（简称指南）包含了协议栈工程结构及配置、应用配置选择、工程结构修改、API接口说明等等。

## 封装协议栈

本指南主要适用于基于Zstack-2.6.3a协议栈的ZigBee/SmartRoom封装协议栈，目前可用的版本如下：

SmartRoomGlib-V1.0.xx

SmartRoomGlib-V1.0.xx适用于所有基于CC2530应用产品开发的软件工程师，软件开发工程师使用SmartRoomGlib-V1.0.xx基础库开发项目应用软件时需要使用Git软件版本控制软件将开发的可编译通过的软件提交到自已的本地版本库，待一个开发完成或者一天的开发结束之后将开发的结果推送到git服务器中。

## 术语和缩略词

|  |  |  |
| --- | --- | --- |
| 序号 | 术语或缩略语 | 说明性定义 |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |

# 

# 工程结构

ZigBee/SmartRoom封装协议栈的工程结构框架分为两类，一类是一个Workspace对应一个Project的结构框架，这种结构适用于单个产品项目；另一类是是一个Workspace对应多个Project的结构框架，适用于同类产品不配置的项目或者同一产品不同功能的项目。

## 一对一的工程结构

如下图2-1所示的红外入侵探测器的示例工程结构框架，一个红外入侵探测器的Workspace(InfraredIntrusionDetector.eww)中仅仅包含一个产品Project，对应大Project文件为InfraredIntrusionDetector.ewp。

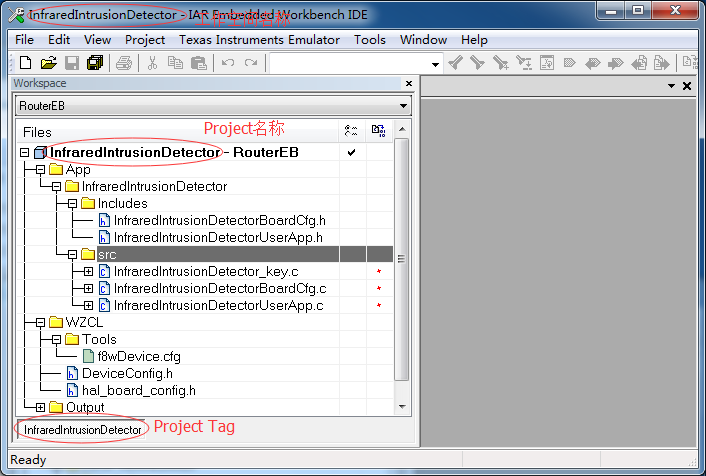


图2-1 一对一的工程结构

## 一对多的工程结构

### 同一类产品不同配置的工程

一对多的工程结构这里我们以如下图2-2所示的零火线开关1键、2键、3键的产品工程结构作为示例来说明，ZeroFirewireSwitch03M.eww的工作空间中包含N1Key、N2key、N3key三个Project分别是零火线1键、2键、3键的Project，采用此种结构的优势是在同一类产品的不同应用配置是可以统一开发和管理，可以使用同一套软件源码，只需要通过不同的宏定义选择是否编译某一段源码。

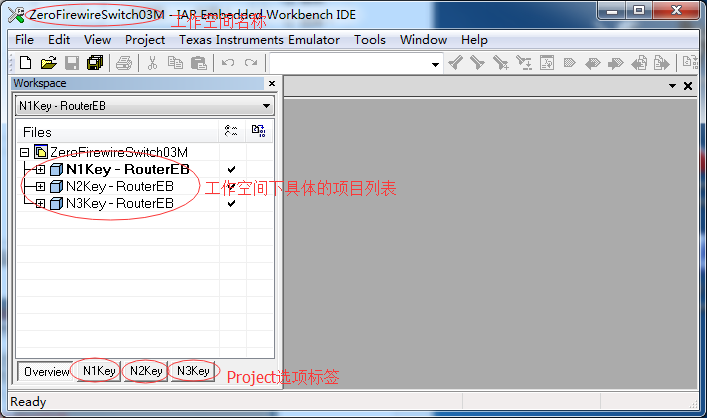


图2-2 一对多的工程结构

在一对多的工程结构下开发者选中一个Project的Tag时即可切换到相应的Project，然后用户即可根据实际的情况编写调试应用程序，如下图2-3所示。

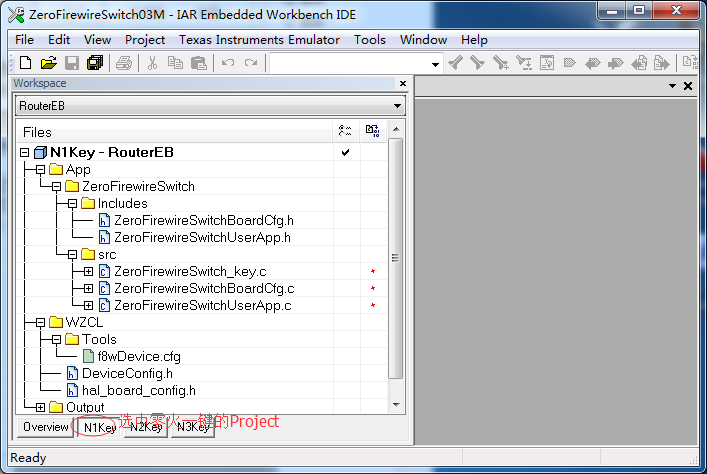


图2-3 一对多工程结构下选中一个Project

### 同一产品不同功能的工程结构

同一产品不同功能也可采用一对多的工程结构，例如下面的示例中的门窗磁探测器的工程结构，在老的门窗磁探测器中采用的中断I/O引脚和最新的门窗磁探测器不同，这里我们就通过两个Project来开发出程序源码同时兼容老的门窗磁探测器和新门的门窗磁探测器，通过不同的Project来编译出分别适用于新旧两种门窗磁探测器的固件，图2-4所示

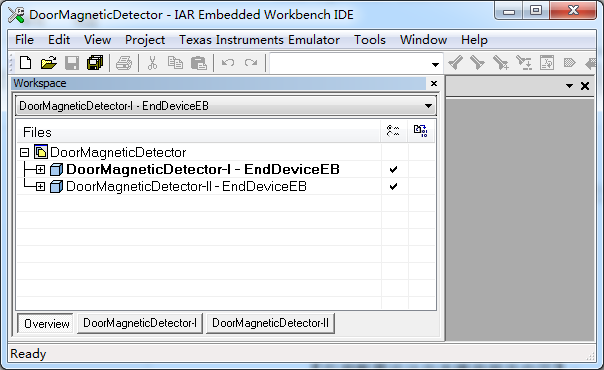


图2-4 门窗磁探测器工程

在门窗磁探测器的源码中，用新旧门窗磁的宏DOOR\_MAGNETIC\_DETECTOR\_V2和DOOR\_MAGNETIC\_DETECTOR\_V1分别将新门窗磁和旧门窗磁的源码用预编译的方式定义，在两个Project中的Defined symbols中分别定义上面两个宏，在程序编译时就会自动选择相应的部分的源码进行编译，如下图2-5所示。

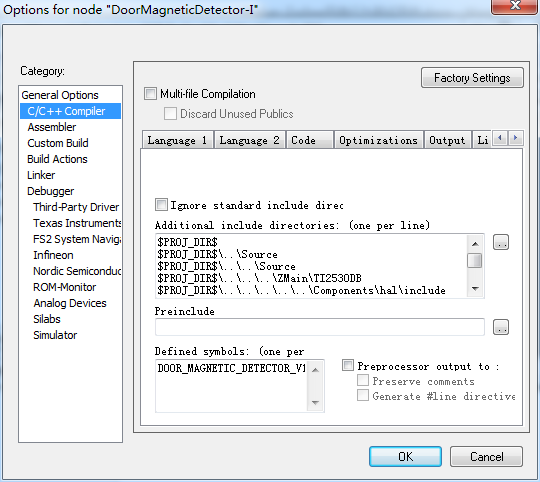


图2-4 宏预定义

# 

# 工程文件组织和编译配置选项

本节内容主要包含了Project的文件组织结构和编译配置选项。文件组织结构可参照示例文件组织结构来组织向产品应用开发者提供可灵活选择的编译配置选项来适应不同的设备类型和配置场合。

## 工程文件组织

在封装的协议栈中，默认只提供了应用相关程序源文件的结构组织。其他一些文件都被编译成lib，仅仅在编译程序的最后在link时才会将lib和整个应用程序文件连接成一个完整的应用程序固件。

封装协议栈中程序源文件结构组织如下图3-1所示。

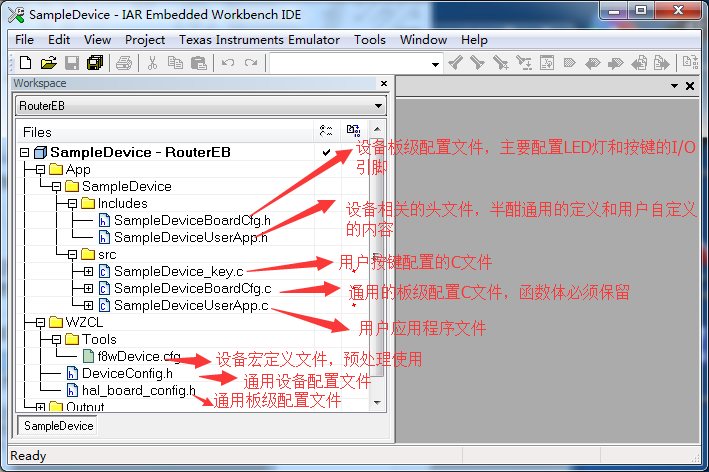


图3-1 工程文件组织结构

## 编译配置选项

在上述的两种结构中，每个具体的工程下包含了两种设备类型共四种Configurations，即Router和EndDevice两种设备类型分别带PA和不带PA的配置选项，分别为RouterEB、RouterEB-PA、EndDeviceEB和EndDeviceEB-PA，如下图3-2所示。

用户可根据实际的应用情况选择其中的一种配置作为产品配置编写程序，也可针对所有配置的编写通用的应用程序源码，这样在最后使用时，即可选择需要的配置来编译生成目标结果。

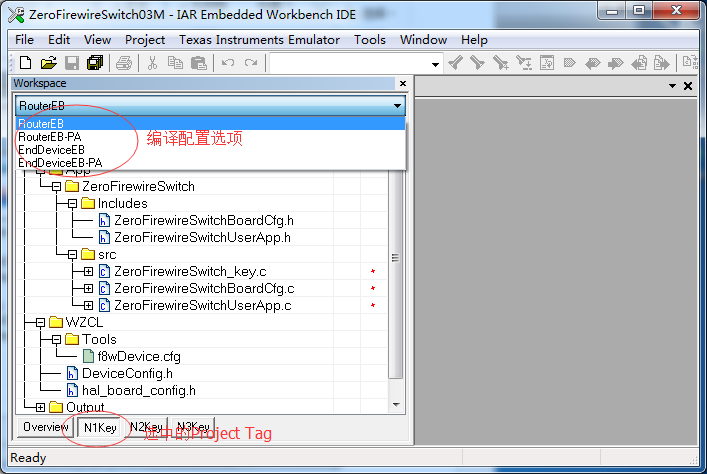


图3-2 Project的编译配置选项

如果开发者确定该产品只会选择其中一种或几种配置来编译生成目标固件，可以将其他不用的配置选项直接删除，删除方法Project🡪Edit Configurations，弹出如下对话框，如下图3-3所示。

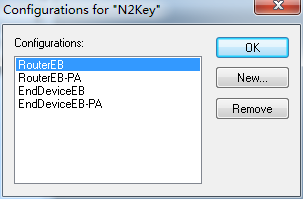


图3-3 配置选项

选择不用的配置选项，然后Remove即可删除不需要的配置选项。

# 

# 创建项目工程

项目应用程序开发者在开发一个新的项目产品时需要根据示例的项目工程创建一个基于该项目的新的项目工程，创建一个新的项目工程仅仅需要按照下面描述的步骤即可快速完成。

## 创建项目文件

应用产品开发者需要在路径\Projects\zstack\SmartRoom下创建一个文件夹，文件夹命名为该产品的软件包名称（命名应该遵循产品软件包命名规则）。例如，一个门窗磁探测器的产品，文件夹命名为DoorMagneticDetector。然后将SampleDevice文件夹下的所有文件复制到DoorMagneticDetector文件夹下，如下图4-1、图4-2所示：

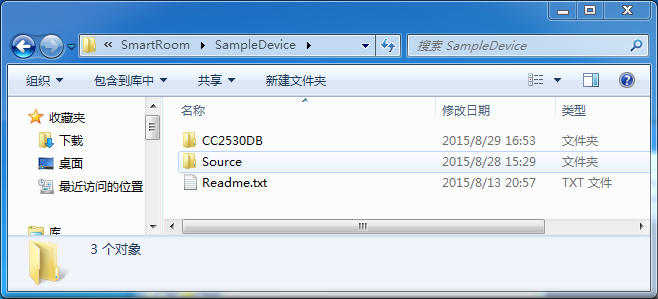


图4-1 SampleDevice项目文件内容

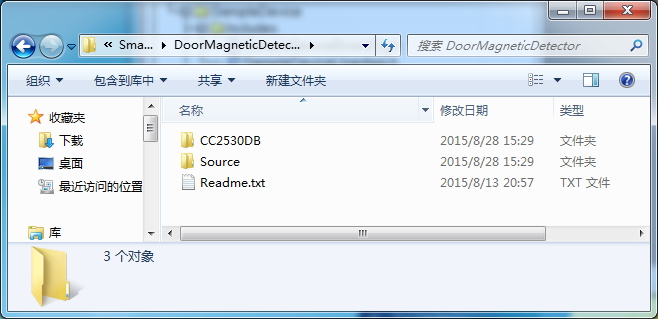


图4-2 DoorMagneticDetector项目文件内容

## 修改工程文件

进入DoorMagneticDetector\CC2530DB文件夹，将SampleDevice.ewd、SampleDevice.ewp、SampleDevice.eww三个文件分别重命名为DoorMagneticDetector.ewd、DoorMagneticDetector.ewp、DoorMagneticDetector.eww，进入DoorMagneticDetector\Source将该文件夹中的所有文件的文件名中的SampleDevice以及文件内部的SampleDevice统一都改为DoorMagneticDetector，文件名更改如下图4-3所示。

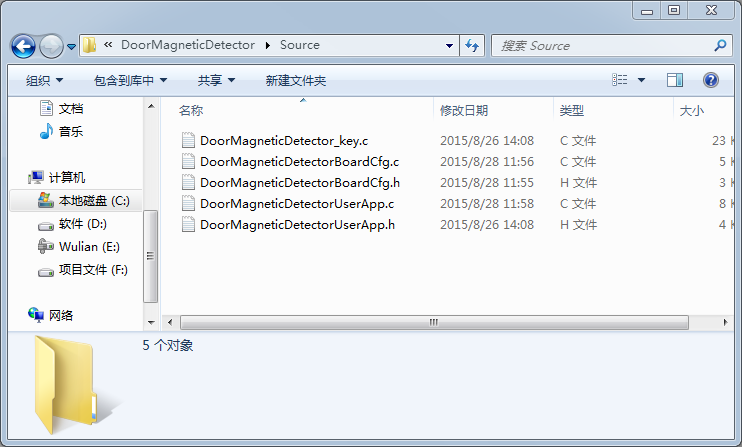


图4-3 重命名后的源文件

用IAR Embedded Workbench for 8051 9.10打开文件DoorMagneticDetector.eww此时会出现如下图4-4所示的警告提示，可以点击确认忽略。

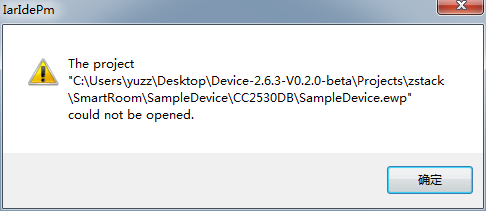


图4-4打开工程警告

然后在打开的IAR的菜单栏执行Project->Add Existing Project…弹出如下图4-4所示的对话框，然后选中DoorMagneticDetector.ewp加载该项目工程文件。打开后展开项目为文件组织结构如下图4-5所示：

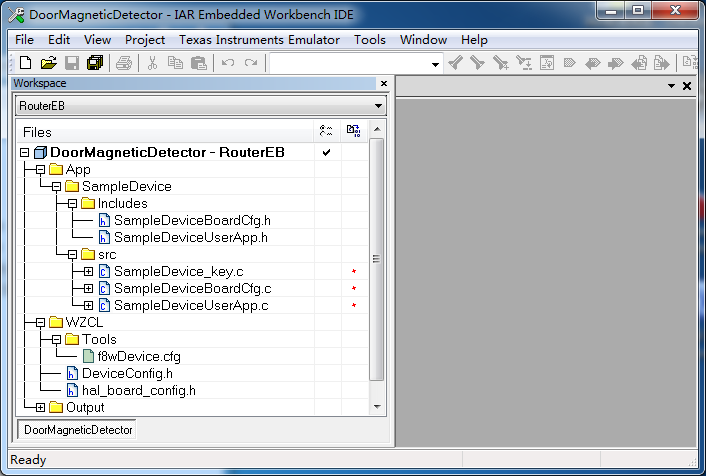


图4-5 重新加载Project后的文件组织结构

这里我们会发现项目工程的源文件中的文件结构还是SampleDevice的文件结构，需要开发者自己修改并添加正确的文件，修改结束后的文件结构如下图4-6所示。

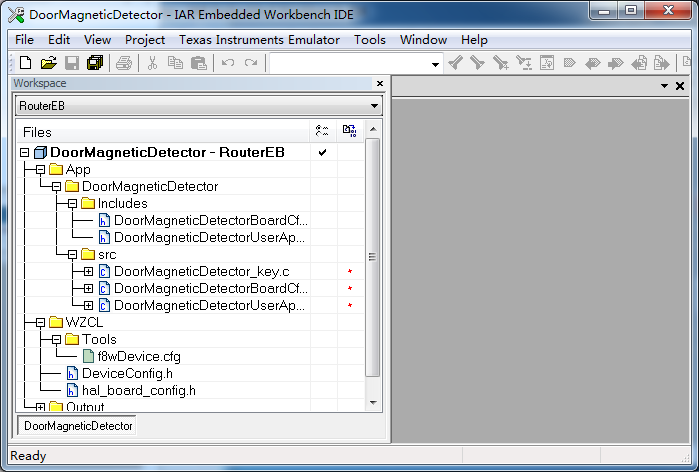


图4-6 修改后的文件组织结构

一对多的工程结构框架，开发者可以选择一单火开关或者零火开关的工程结构框架来修改为对应的应用项目的工程结构，文件的修改也可以选择对应的文件进行修改。

对于一对多的工程结构框架，这里建议使用同一套应用源码适用多种不同配置的Project，在同一Workspace中的Projects中，通过不同的配置预处理宏去选择编译同一套源码中的不同部分，按照这个方式的处理方法，可以同时解决这一类（系列）产品中的多个产品的程序代码，便于整个系列产品的源码的维护，下面以零火线开关1、2、3键共用同一套应用源码为例作为演示。如面所示的板级配置c文件的源码片段。

/\* -------------------------IO Configration------------------------------------\*/

void inline HAL\_IO\_CONFIG\_INIT(void)

{

LED1\_DDR |= LED1\_BV;

LED2\_DDR |= LED2\_BV;

HAL\_TURN\_OFF\_LED1();

HAL\_TURN\_OFF\_LED2();

/\* ------Configurations 1 for switch key 1------ \*/

#if defined(N1KEY) || defined(N2KEY) || defined(N3KEY)

LED3\_DDR |= LED3\_BV;

LED4\_DDR |= LED4\_BV;

HAL\_TURN\_OFF\_LED3();

HAL\_TURN\_ON\_LED4();

RELAY\_CTL1A\_DDR |= RELAY\_CTL1A\_BV;

RELAY\_CTL1A\_SBIT = 0;

#endif

/\* ------Configurations 2 for switch key 2------ \*/

#if defined(N2KEY) || defined(N3KEY)

LED5\_DDR |= LED5\_BV;

LED6\_DDR |= LED6\_BV;

HAL\_TURN\_OFF\_LED5();

HAL\_TURN\_ON\_LED6();

RELAY\_CTL2A\_DDR |= RELAY\_CTL2A\_BV;

RELAY\_CTL2A\_SBIT = 0;

#endif

/\* ------Configurations 3 for switch key 3------ \*/

#if defined(N3KEY)

LED7\_DDR |= LED7\_BV;

LED8\_DDR |= LED8\_BV;

HAL\_TURN\_OFF\_LED7();

HAL\_TURN\_ON\_LED8();

RELAY\_CTL3A\_DDR |= RELAY\_CTL3A\_BV;

RELAY\_CTL3A\_SBIT = 0;

#endif

}

通过N1KEY、N2KEY、N3KEY三个宏来区分零火1键、2键、3键开关需要的代码内容，在编译时根据上面的三个宏来选择性的编译。

## 增加编译配置宏和头文件

打开WZCL->DeviceConfig.h文件，针对开发应用定义一个全局的宏。例如这里我们定义门窗磁设备的宏为DOOR\_MAGNETIC\_DETECTOR，然后参照样例SampleDevice的配置头文件的包含方法，定义门磁的头文件包含方法。

在文件DeviceConfig.h中 如下面的代码段下增加门窗磁的头文件包含代码段，如红色字体部分所示。

/\* SmartRoom SampleDevice \*/

#if defined(SAMPLEDEVICE)

#include "SampleDeviceUserApp.h"

#endif

/\* SmartRoom门窗磁探测器-01型\*/

#if defined(DOOR\_MAGNETIC\_DETECTOR)

#include "DoorMagneticDetectorUserApp.h"

#endif

在文件WZCL->hal\_board\_config.h中增加对应用板级配置接口的头文件引用例如门磁设备板级配置的引用为

/\* SmartRoom门窗磁探测器-01型\*/

#if defined(DOOR\_MAGNETIC\_DETECTOR)

#include "DoorMagneticDetectorBoardCfg.h"

#endif

增加预处理宏定义，预处理宏定义在这里我们通过加载额外的文件的方式来定义预处理宏，如上图3-1的文件组织结构中的Tools目录中的f8wDevice.cfg的预处理配置文件，该文件在路径$PROJ\_DIR$\..\Source\Tools\f8wDevice.cfg下，在使用SampleDevice的样例项目创建新的应用工程项目时该文件已经默认在下图4-7所示的位置中加载了。

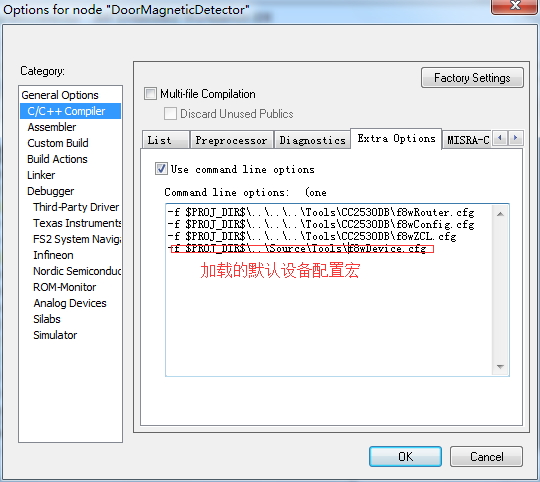


图4-7 加载设备配置宏的文件

加载上图4-7所示的配置宏之后，在编译预处理的宏定义的位置就无须再定义设备的配置宏了如下图4-8所示的位置，而是在文件f8wDevice.cfg中定义预处理宏的定义，如下图4-9 所示。

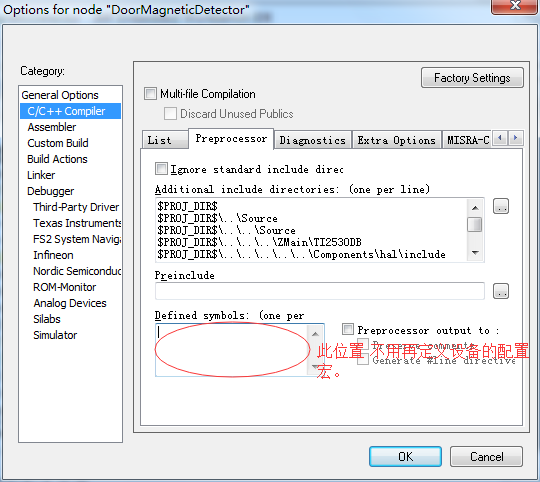


图4-8 取消的预定义宏配置

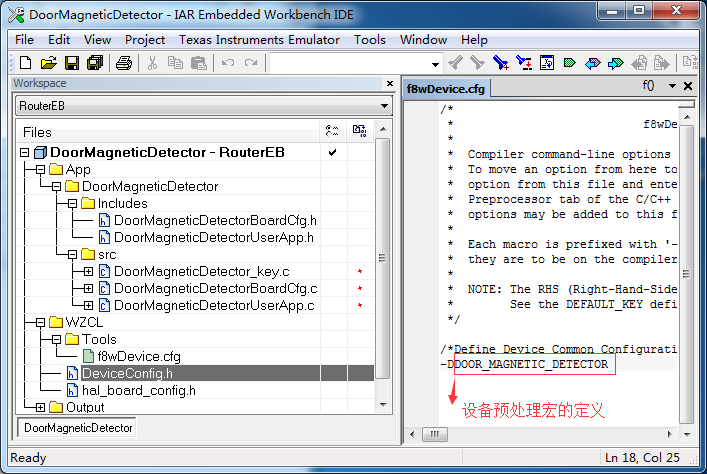


图4-9 设备预处理宏的定义

# 

# 通用API接口的说明

### void UserApp\_Init( uint8 task\_id )

该函数为UserApp层注册taskId函数，一般不需要开发者修改；但是，当开发者需要进行一些应用以及驱动程序的初始化时，需要在该函数内部增加驱动以及应用的初始化部分的应用代码，如下图5-1所示。

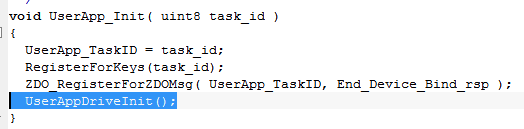


图5-1 设备应用初始化代码添加

### uint16 UserApp\_ProcessEvent( uint8 task\_id, uint16 events )

该函数为事件主处理函数，用户可以自己使用函数uint8 osal\_start\_timerEx( uint8 task\_id, uint16 event\_id, uint32 timeout\_value )启动一个定时事件或者使用函数uint8 osal\_set\_event( uint8 task\_id, uint16 event\_id )设置一个立即事件然后在事件主处理函数中的下面结构中进行对应立即事件和延时事件的处理,其中event\_id 为16bit数按位定义，且0x8000为系统使用，用户不可以使用，用户可以使用的事件号有15个，分别如下所示：

#define USERAPP\_EVENT\_1 0x0001

#define USERAPP\_EVENT\_2 0x0002

#define USERAPP\_EVENT\_3 0x0004

#define USERAPP\_EVENT\_4 0x0008

#define USERAPP\_EVENT\_5 0x0010

#define USERAPP\_EVENT\_6 0x0020

#define USERAPP\_EVENT\_7 0x0040

#define USERAPP\_EVENT\_8 0x0080

#define USERAPP\_EVENT\_9 0x0100

#define USERAPP\_EVENT\_10 0x0200

#define USERAPP\_EVENT\_11 0x0400

#define USERAPP\_EVENT\_12 0x0800

#define USERAPP\_EVENT\_13 0x1000

#define USERAPP\_EVENT\_14 0x2000

#define USERAPP\_EVENT\_15 0x4000

if(events & event\_id)

{

/\* ...... \*/

return events ^ event\_id;

}

### void UserAppHandleKeys( uint8 shift, uint8 keys )

用户按键识别处理函数，在这个函数里，HAL\_KEY\_SW\_1、HAL\_KEY\_SW\_2为固定的使用，分别为入网使用和检测功能使用，普通应用不可以使用这两个定义，***普通应用使用的按键定义应该从HAL\_KEY\_SW\_3开始。***

在一个按键的处理结构中，如果shift为1，则在该结构的下面添加帮banding功能处理程序，否则是具体应用处理程序。

else if(keys & HAL\_KEY\_SW\_4)

{

if(shift)

{

/\*按键binding使用\*/

}

else

{

/\*按键正常控制使用\*/

}

}

### void UserAppProcessKeyNwkAction(void)

处理设备网络行为相关的操作，其中变量KeyMultiClicks和KeyHoldPress分别表示SET按键的多击和长按，例如KeyMultiClicks == 2表示SET键被快按两次以此类推KeyMultiClicks == 3表示SET按键被快按三次；KeyHoldPress == 1表示SET长按达到预设的时间，否则KeyHoldPress==0。

### void UserAppProcessUartIncoming(uint8\* msg\_ptr)

接受串口数据的处理函数，开发者在配置好需要的串口后，在正常通讯过程中可以直接在本函数内填充接受串口数据的处理函【暂未实现】。

### void UserAppProcessCommandCallBack(uint8 inMsgLen, uint8 \*inMsg, uint8 \*outMsgLen, uint8 \*outMsg)

接受到网络命令后的数据处理函数，开发者直接可在本函数内处理网络应用命令，并且填充返回命令的应用数据，当收到网络传来的命令，且不需要通过本函数返回结果时需要设置\*outMsgLen = 0。

### void UserAppProcessReportingCommand(uint8 cmd ,uint8\*inMsg)

对于温湿度类环境检测设备，其一些环境状态值是可配置的周期性上报的，此时即可通过该函数接收周期性上报配置命令，然后配置可周期性上报的值。例如配置温湿度监测器的周期性上报。

*void UserAppProcessReportingCommand(uint8 cmd ,uint8\*inMsg)*

*{*

*switch(cmd)*

*{*

*case 'P':*

*PeriodTime = 100\*(inMsg[0]-'0') + 10\*(inMsg[1]-'0') + (inMsg[2]-'0');*

*PeriodTimeMinute = PeriodTime/60;*

*PeriodTimeSecond = PeriodTime%60;*

*if(PeriodTime > 0)*

*{*

*osal\_start\_timerEx(UserApp\_TaskID, COMMAND\_PERIODIC\_MSG\_EVT,100 );*

*}*

*break;*

*case 'S':*

*PeriodTime = 0;*

*PeriodTimeMinute = 0;*

*PeriodTimeSecond = 0;*

*osal\_stop\_timerEx(UserApp\_TaskID,COMMAND\_PERIODIC\_MSG\_EVT);*

*break;*

*case 'F':*

*TemperatureThrethodValue = 100\*(inMsg[0]-'0') + 10\*(inMsg[1]-'0') + (inMsg[2]-'0');*

*HumidityThrethodValue = 100\*(inMsg[3]-'0') + 10\*(inMsg[4]-'0') + (inMsg[5]-'0');*

*break;*

*case 'L':*

*BlinkCount = 100\*(inMsg[0]-'0') + 10\*(inMsg[1]-'0') + (inMsg[2]-'0');*

*HalLedBlink(HAL\_LED\_1, BlinkCount, 50, 800);*

*break;*

*default:*

*break;*

*}*

*}*

### void UserAppProcessBindingMessage(afIncomingMSGPacket\_t \* pkt)

处理绑定的Initiator设备发来的控制命令，例如移动插座收到绑定开关发来的控制命令然后执行开关动作：

*void UserAppProcessBindingMessage(afIncomingMSGPacket\_t \* pkt)*

*{*

*switch(pkt->endPoint)*

*{*

*case USERAPP\_ENDPOINT1:*

*if(pkt->cmd.Data[0]==1 && RelayLoad.load.loadstate == LOAD\_CLOSE)*

*{*

*RelayLoad.load.Loadopen();*

*}*

*else if(pkt->cmd.Data[0]==0)*

*{*

*RelayLoad.load.Loadclose();*

*}*

*else if(pkt->cmd.Data[0]==2)*

*{*

*if(RelayLoad.load.loadstate ==LOAD\_CLOSE)*

*RelayLoad.load.Loadopen();*

*else if(RelayLoad.load.loadstate ==LOAD\_OPEN)*

*RelayLoad.load.Loadclose();*

*}*

*LoadState.load = RelayLoad.load.loadstate;*

*break;*

*default:*

*break;*

*}*

*WZCL\_SendDsMessage((uint8\*)&LoadState,sizeof(LoadData\_t));*

*}*

### void UserAppStartNwkJoiningCallBack(void)

设备上电后如果启动加网，即会调用该函数(在版本V1.0.11和V1.0.12中失效，如果需要使用该功能需要使用最新的版本)。

在V1.0.13以及V1.1.0版本中正常调用

### void UserAppProcessZDOMsgs( zdoIncomingMsg\_t \*inMsg )

处理Zdo应用命令的函数，处理的内容取决于在函数void UserApp\_Init( uint8 task\_id )中注册的Zdo 命令，例如在示例应用中的UserApp\_Init函数中，注册的Zdo命令为ZDO\_RegisterForZDOMsg( UserApp\_TaskID, End\_Device\_Bind\_rsp );因此在UserApp层仅仅需要处理End\_Device\_Bind\_rsp命令的内容。

### void UserAppCoordinatorOfflineCallback(bool offline)

该函数仅对路由器设备有用，该函数传入参数 offline表示设备的网关是否处于联通状态，例如一个设备以路由器的角色加入到网关中，设备工作一段时间之后由于某种原因造成网关断电，这是路由器设备会在一段时间内检测到网关丢失，就调用该函数同时传入offline参数为true，表示网关丢失，当修复网关一段时间之后，路由设备又重新检测到网关在线，即会调用该函数并且传入参数offline 为false表示网关正常工作中。

### void UserAppJoinedNetworkCallBack(void)

入网成功后的回调处理函数，处理一些入网成功后需要处理的内容，例如设备入网成功之后需要闪灯，则可在该函数内添加闪灯代码，如下：

HalLedBlink ( HAL\_LED\_1, 1, 99, 2000 );

### void UserAppRssiBlinkLed(wzclDevRssiRsp\_t \*pRsp)

当设备收到命令/C/UUeXXXX或者主动调用函数WZCL\_InterRssiReqMessage()之后机会获取双向通讯信号值，并且调用该函数并传入设备的双向信号值，开发者可根据传入的双向信号值让设备做出适当的行为。

### void UserAppExtendLightBlink(uint8 numblinks)

用于调光灯具/内嵌式开关的/C/UULXXXX命令，取代默认的闪烁sys指示灯，如果需要启用该功能，在函数UserAppSetDevInfo()中设置devInfo->defaultBlinkEnable = FALSE。

### void UserAppPreSendRsMessageCallBack( uint8 \* rsMsg )

设置Rs应用数据的内容，Rs应用数据内容表示设备上线后上报的第一条应用数据；其中包含了设备的类型，设备的唯一ID号，以及设备的初始状态应用数据，在这里开发者仅仅需要填充设备的初始状态应用数据，其长度取决于void UserAppSetDevInfo(wzclDevInfo\_t \*devInfo)函数中devInfo->rsLen 设置的长度，例如某个设备的初始应用数据状态值长度为一个byte，那么仅需设置*devInfo->rsLen = 1;*

### void UserAppSetDevInfo(wzclDevInfo\_t \*devInfo)

设置该应用设备的一些相关信息，主要信息类容都在结构wzclDevInfo\_t中，可打开头文件wzcl.h文件查看，如下所示：

typedef struct

{

uint8 ctrlChar; //设备控制符 /C/UU\*XXXX

uint8 cmdIndCh; //设备应用数据指示符 DS：RS数据指示

uint8 devId; //设备的类型Id号

uint8 rsLen; //设备RS消息中应用数据的长度

uint8 extEpNums; //绑定控制中额外使用的端点，

uint8 defaultBlinkEnable; //默认闪灯使能，设备收到/C/UULXXXXYYY

//是执行闪灯还是UserAppExtendLightBlink(uint8)

#if !defined(RTR\_NWK)

uint8 maxPollFailureRetries; //EndDevice设备最大Poll rate失败次数后

//被认为丢失网络

#endif

LowBatteryAlarm\_t lowBatteryAlarm; //电池低压报警功能，默认不开启

uint8 MinutesSendHeartToCoord;//EndDevice直接到网关的心跳的周期/min

uint8 MinutesHeartWithApsAck;//设备指定固定周期心跳/min带有aps ack;

uint32 heartBeatIntval; //心跳周期配置默认60000ms（1min）

} wzclDevInfo\_t;

### uint8 UserAppAutoStartNwk(void)

设备上电自动加网启用功能，返回值为TRUE时设备上电自动加网，返回值为FALSE时设备上电后需要用户另外重新触发加网，该函数仅在老的加网方式中有效（在SmartRoomGlib-V1.1.0版本中返回值为TRUE时设备不管在新旧加网方式中都是上电自动入网，等于FALSE时上电不自动入网），在SmartRoomGlib-V1.0.11~ SmartRoomGlib-V1.0.13版本中，TOP47产品的开发使用XXX\_AtuoStart.lib，而不再使用该函数的返回值，老的产品的加网方式中该功能仍然有效。在SmartRoomGlib-V1.1.0及其以后版本中，该函数的返回值为TRUE时设备自动加网，等于FALSE时设备上电不自动加网，此时使用XXX\_AtuoStart.lib。

### void UserAppSetPollRate(wzclPollControl\_t \*pollrate)

设置EndDevice设备的PollRate机制，相关的内容在wzclPollControl\_t结构中有具体的体现。

该设置能够影响电池设备（安防设备、环境检测设备、锁具、红外转发等等）的命令响应速度和功耗，单火线开关设备的功耗和响应速度。具体产品的参数设置参考附件中的表格；表格中还包括上面函数中的几个参数。

### uint32 ChannelSelect(void)

产品生产测试测试信道选择，在SmartRoomGlib-V1.0.12及其以后版本中仅在TOP47以及新开发的产品中有效，老产品升级中该功能无效。

### void WZCL\_SendAsMessage(wzclAsMsg\_t \*iMsg)

设备向网关发送一个电池低压报警信息，或者发送一个设备防拆报警信息。

### void WZCL\_FindAndJoinNetwork(void)

如果设备未加入一个ZigBee网络，调用该API会使设备发生复位，然后启动请求加入一个ZigBee/SmartRoom网络进程。

### void WZCL\_LeaveAndRestoreFactory(void)

设备调用该API之后，不管设备是否在网，都会发生复位并且恢复出厂设置。

### void WZCL\_SendRsMessage(void);

启动一次发送RS消息，调用该API之后设备会主动发送一次RS消息给网关。

### void WZCL\_SendDsMessage(uint8 \* iMsg, uint8 len)

向网关发送一个DS应用消息，调用该API并且传入要发送的数据内容和长度，设备即会将要发送的应用消息内容发给网关。

### void WZCL\_SendDsWithAckMessage(uint8 \* iMsg, uint8 len)

同5.1.24，但是会根据是否搜到指定的aps ack进行应用层数据3次重发，确保网关能够准确收到应用数据。

### void WZCL\_SendBindingMessage(endPointDesc\_t \* srcEP,uint16 cID, uint16 len, uint8 \*buf)

发送绑定控制命令到被绑定的目标设备。

### void WZCL\_InBindingReqMessage(uint8 endpoint ,cId\_t\* bindingInClusters,uint8 clusterlists)

请求建立设备间的绑定，接收绑定控制命令的设备在建立设备间绑定时调用。

具体使用方法，参考六路场景开关SceneSwitch项目的应用程序。

### void WZCL\_OutBindingReqMessage(uint8 endpoint ,cId\_t\* bindingOutClusters,uint8 clusterlists)

请求建立设备间的绑定，发送绑定控制命令的设备在建立设备间绑定时调用。具体使用方法参考单/零火线开关项目的应用程序

# 基本功能开发

## 设备加网退网功能

设备加网退网功能的描述仅对SmartRoomGlib-V1.0.11之后的基础库有效，在这之前的基础库不在做详细的说明。

### 老产品加网退网方式

使用老产品加网退网方式的主要是库房升级产品，Top47产品以及新研发产品不适用该种设备加网退网方式。

#### 快按SET键四次入网，长按SET键10s退网方式

使用这种加网退网方式时需要在预处理中定义宏HOLD\_AUTO\_START，如下图6-1所示

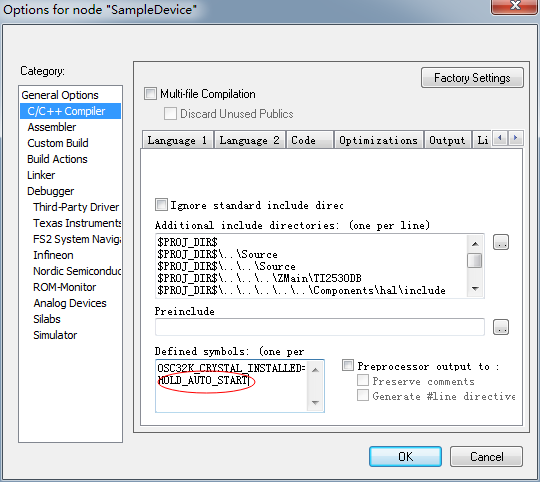


图6-1 加网退网方式HOLD\_AUTO\_START定义

#### 通用加网退网操作

针对我司大多数产品老而言，使用的是快按SET键四次入网，长按SET键10s退网的机制，因此对于一般产品而言使用在UserAppHandleKeys(uint8 shift, uint8 keys)函数中直接使用如下的代码段即可实现通用的加网退网操作，UserAppProcessKeyNwkAction()中已经集成了入网退网的操作，用户只需要在xxxx\_key.c和xxxxBoardCfg.h文件中写好SW\_1对应的底层检测驱动和引脚配置就可以了。

if(keys & HAL\_KEY\_SW\_1)

{

UserAppProcessKeyNwkAction ();

}

对于一些加网退网操作有变化的个别特定的产品，开发者可针对性的自行修改UserAppProcessKeyNwkAction()中的加网退网操作功能。

#### 上电自动加网操作

上电自动加网操作是一些灯具类设备，内嵌式开关类设备所用的加网方式，这类设备如需要实现上电自动加网，仅仅需要将UserAppAutoStartNwk()函数的返回值设定为TRUE设备就会在上电后自动进行加网

#### 串口命令入网退网

串口命令加网退网，适用于其他处理器作为主控制芯片，CC2530 ZigBee芯片作为从芯片仅仅用于通信方案，这类方案的设备在接受到串口入网命令时可调用函数WZCL\_FindAndJoinNetwork()来加网，接受到退网命令时可通过调用函数WZCL\_LeaveAndRestoreFactory()来使设备退出网络。

#### 其他动作退网

其他非常规退网时，可在动作触发后直接调用函数WZCL\_LeaveAndRestoreFactory()来实现退网。

### Top47产品以及新研产品加网退网方式

Top产品以及个别新研制的产品，加网退网方式主要使用的是上电自动加网，长按SET按键6s退网，退网成功后设备保持空闲不加网状态，按一下SET按键或者快按SET键4次即可重新请求加入网络。

#### 上电自动入网，长按SET按键6s退网，不在网状态下按一下SET按键或者快按SET键4次申请加入网络

使用这种加网方式，不需要在预处理中定义宏HOLD\_AUTO\_START，但是需要开发者更换使用Projects\zstack\Libraries\TI2530DB\bin\路径下XXXX-AtuoStart.lib的库文件，如图6-2所示。

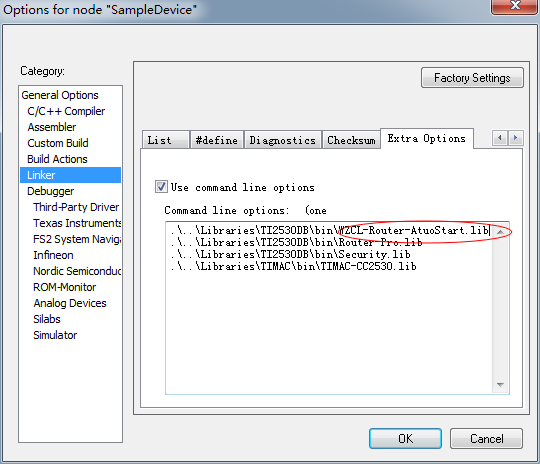


图6-2 使用XXXX-AtuoStart.lib的库文件

#### 通用加网退网操作

通用加网退网操作调用的API同[6.1.1.2](#_通用加网退网操作)开发者需要改动的地方仅仅是在调用的函数UserAppProcessKeyNwkAction()内部修改增加下面红色字体部分的代码段：

*if(KeyMultiClicks == 1)*

*{*

*if((devState == DEV\_END\_DEVICE) || (devState == DEV\_ROUTER))*

*HalLedBlink ( HAL\_LED\_1, 1, 50, 1000 );*

*#ifndef HOLD\_AUTO\_START*

*else*

*WZCL\_FindAndJoinNetwork();*

*#endif*

*}*

#### 上电自动加网操作

新产品使用的是上电自动加网，在SmartRoomGlib-V1.1.0及其以后的版本中同样需要和老版本的产品一样需要设定UserAppAutoStartNwk()的返回值为TRUE或者FALSE来决定设备是否上电自动入网。

#### 串口命令入网退网

同[6.1.1.4](#_串口命令入网退网)

#### 其他动作退网

同[6.1.1.5](#_其他动作退网)

## 指示灯闪烁

指示灯闪烁问题，通用的使用API void HalLedBlink( uint8 leds, uint8 cnt, uint8 duty, uint16 time )来实现，其他特定的指示灯闪烁问题，请开发者自行开发。

## 设备参数配置

### EndDevice设备参数配置

基于SmartRoomGlib-V1.0.11及其以后的版本中，EndDevice设备需要按照如下的配置表，配置设备工作过程中的参数。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 参数 | Default | 单火线开关 | 安防设备 | 温湿度 | 锁具 | 转发 | 机械手 |
| ctrlChar | ‘E’ | ‘v’ | --- | --- | --- | --- | --- |
| cmdIndCh | ‘/’ | ‘/’ | --- | --- | --- | --- | --- |
| devId | X | --- | --- | --- | --- | --- | --- |
| rsLen | X | --- | --- | --- | --- | --- | --- |
| extEpNums | 0 | 3 | 0 | 0 | 1 | 1 | 1 |
| defaultBlinkEnable | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE |
| lowBatteryAlarm | No Used | --- | --- | --- | --- | --- | --- |
| maxPollFailureRetries | 3 | 10 | 3 | 3 | 5 | 5 | 5 |
| MinutesSendHeartToCoord | 3 | 10 | 180 | 180 | 30 | 30 | 30 |
| MinutesHeartWithApsAck | 10 | 10 | 60 | 60 | 30 | 30 | 30 |
| heartBeatIntval (/s) | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| shortPollInterval (x)ms | 500ms | 500ms | 5 | 5 | 2 | 2 | 2 |
| enablelongPoll | FALSE | TRUE | TRUE | TRUE | FALSE | FALSE | FALSE |
| fastPollTimeout(x\*1000)ms | 60 | 25 | 60 | 60 | 60 | 60 | 60 |
| longPollInterval(x\*1000)ms | 18 | 800ms | 18 | 18 | 18 | 18 | 18 |

### Router设备参数配置

对于Router设备目前没有其他配置参数，可配置的参数主要是MinutesSendHeartToCoord 和MinutesHeartWithApsAck，两个参数主要影响路由设备对链路信息的识别和维护的灵敏度。

## PA配置问题

对于有PA的设备，在编译时选择配置选项xxxx-PA选项进行编译，对于没有PA的设备选择无PA扩展的配置选项进行编译生成结果。

## 32.768kHz晶振

对于使用的CC2530模块或者直接使用板上布线CC2530的设备没有外接32.768kHz的需要在预处理中定义OSC32K\_CRYSTAL\_INSTALLED=FALSE，否则在程序运行过程中会出现不能启动，或者启动后工作异常等问题。

## 看门狗问题

开发者在在线仿真调试程序时选择了Router角色的配置时，需要关掉看门狗初始化程序，否则在仿真调试的过程中会频繁出现复位的现象，关掉看门狗的方法是在XXXXBoardCfg.c文件中注释掉函数WatchDogEnableCB(void)的函数内部内容，如下所示

*void inline WatchDogEnableCB(void){/\*WatchDogEnable( WDTIMX );\*/}*

## 产品生产测试（产测）

产品生产测试主要用于产品生产时对硬件的功能，射频的指标进行一下测试，挑出产品生产过程中有问题的PCBA，保证产品的硬件的可靠性和一致性。产品产测方案参考《基于ZigBee SmartRoom-2.6.3a协议栈产测软件设计需求》

### 禁止产测

开发者通过在XXXX\_UserApp.c文件中通过设置uint8 DisableManufacture = TRUE来禁止产测，开发者设置禁止产测前需要根据钟工分配的license来设置填充License数组，该功能仅在SmartRoomGlib-V1.1.0版本中有效。

如果开发者设置DisableManufacture = FALSE表示默认设备会进入产测模式，如果设置DisableManufacture = TRUE，表示设备会禁止进入产测，此时开发者必须提供有效的License，具体License的内容需要向钟工索取。

### 进入产测

开发者配置关联好SET按键（在SmartRoomGlib中默认的HAL\_KEY\_SW\_1）之后，同时保证在XXXX\_key.c文件中的uint16 HalKeyRead ( void )函数内包含如下的代码段。

*if (HAL\_PUSH\_BUTTON1())*

*{*

*keys |= HAL\_KEY\_SW\_1;*

*}*

*else*

*{*

*keys &= ~HAL\_KEY\_SW\_1;*

*}*

烧录生成的固件之后设备按住SET按键上电即可进入产测功能代码，产测时设备通过API uint32 ChannelSelect(void)获取硬件中的信道配置并返回产测定义的信道掩码给网络层，用来指定产测时搜网信道，产测时设备仅在信道掩码指定的信道进行搜索加入网络。

### 产测信道选择

产测信道选择通过API uint32 ChannelSelect(void)的返回的32bit的信道掩码来确定，返回值通过获取产测信道引脚的状态值来确定。

### EndDevice设备的产测

#### Poll Rate规律

对于EndDevice设备，由于产测时需要保证快速性，以便提高生产效率，所以在EndDevice设备在进行产测时需要设置一个相对快速的Poll Rate，因此需要在产测的时候设定Poll Rate时间为200ms，设备正常工作的时候按照EndDevice设备类型去设定Poll Rate规律，如下所示红色部分是产测时设备工作Poll Rate

*if ( zdappHoldKeys & SW\_BYPASS\_START )*

*{*

*pollrate->shortPollInterval = 200;*

*}*

*else*

*{*

*pollrate->shortPollInterval = 5000;*

*pollrate->enablelongPoll = 1;*

*pollrate->longPollInterval = 18000;*

*}*

#### 产测时禁止一些功能

产品生产测试时一些应用功能需要禁止，或者和产品正常工作时区分对待，在这里如果 (zdappHoldKeys & SW\_BYPASS\_START) == TRUE 表示设备工作在产测模式，如果其值为FALSE表示产品工作在正常模式。

## 链接库使用

使用封装协议栈，开发者需要学会自己选用封装库，开发新项目可使用SimpleDevice示例程序创建，已经开发过的项目需要根据具体项目需求增加编译配置选项，并修改链接库，修改链接库如图6-3所示

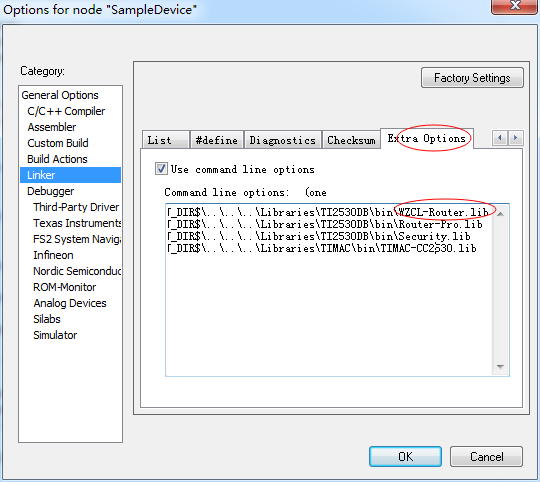


图6-3 修改程序链接库

# 固件生成以及输出

## 固件输出配置

### 设置固件生成路径

开发者使用SampleDevice示例工程创建一个新的项目工程后，必须设定好固件生成路径，固件生成路径需要在每个配置选项上都必须设置。

1. 选择配置选项如下图所示

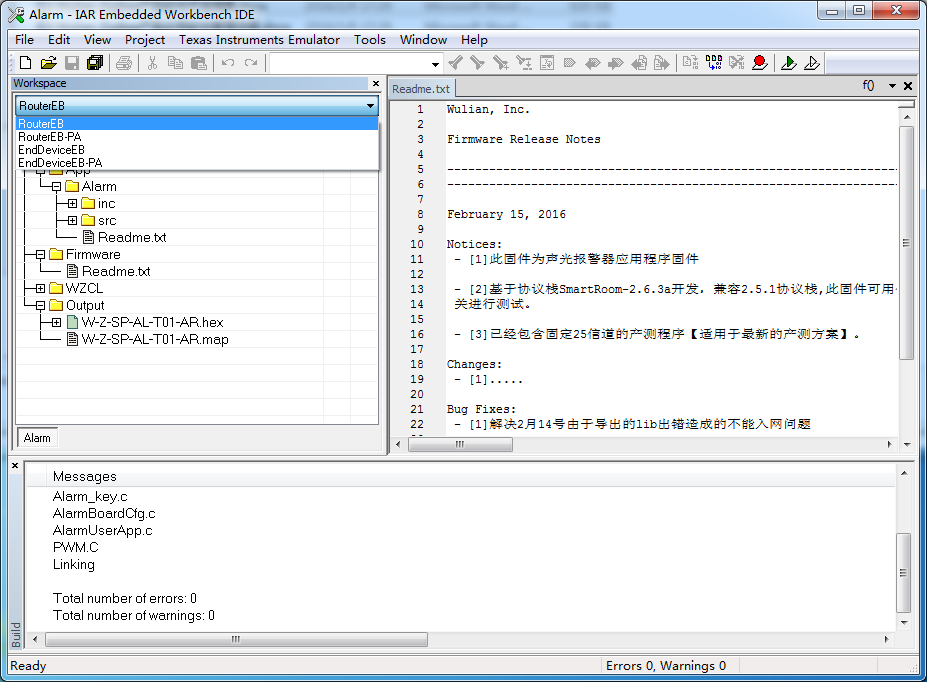


图7-1 编译配置选项

1. 选择菜单Project🡪Option….或者快捷键Alt+F7弹出如下图所示对话框

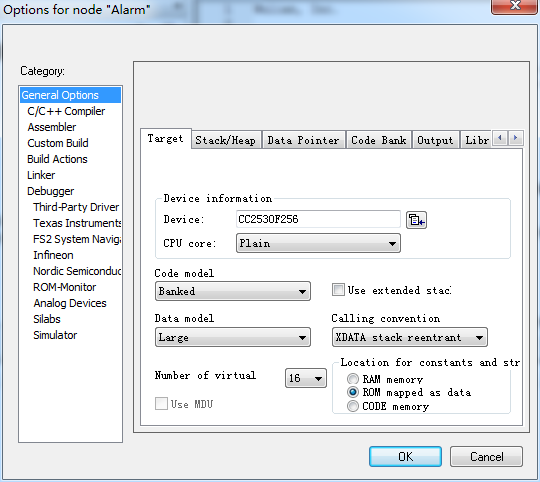


图7-2 配置对话框

1. 选择选项General Options下的Output选项卡输入D:\Firmware\SampleDevice \XXXX的固件生成路径，其中XXXX为固件的名称，同时配置编译生成的中间文件的路径为：

D:\tmp\SampleDevice\RouterEB\Obj

D:\tmp\SampleDevice\RouterEB\List

如下图所示

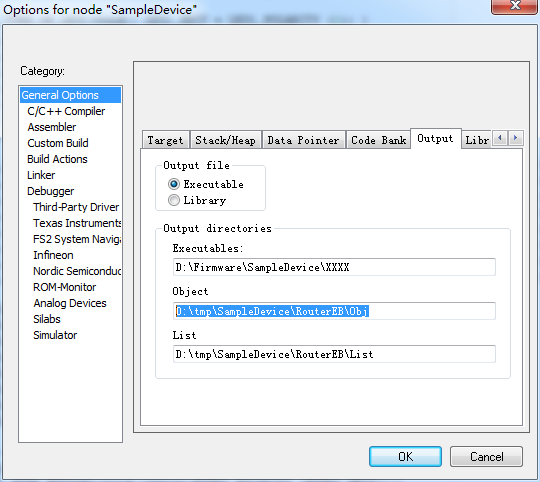


图7-3 中间文件生成路径以及固件生成路径

### 设置固件输出格式以及名称

在上图7-4中所示的对话框中，选择Linker选项的Output选项卡如下图所示

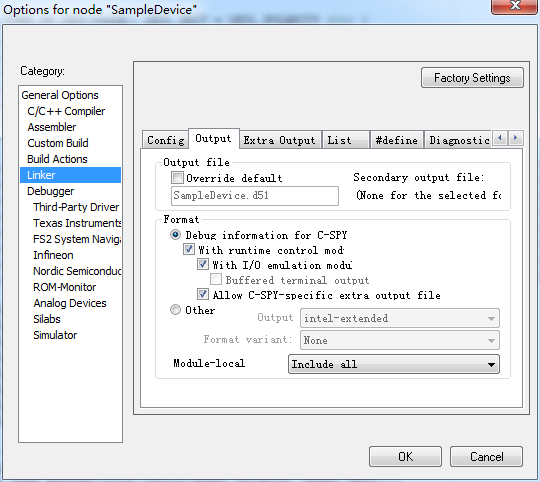


图7-4 Linker配置选项

设置如上图7-4所示的编译linker配置选项，然后切换到Extra Output选项卡，配置选项和固件生成格式如下图7-5所示

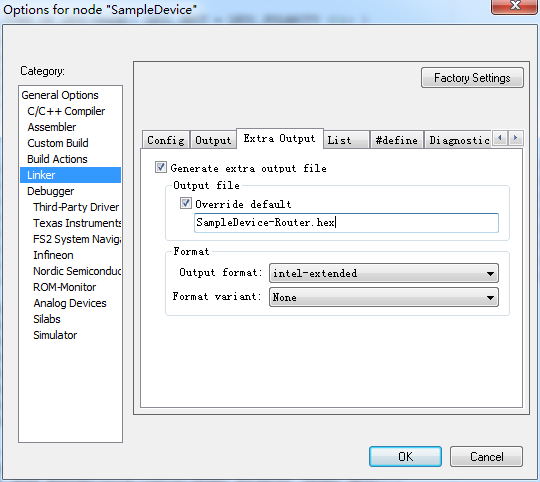


图7-5 固件生成格式配置

选中 Override default选项，然后在输入文本框中，输入目标输出固件的名称以及扩展名.hex，固件名称同固件路径设置时的XXXX，然后在Format中选择other输出格式，在output下拉列表框中选择Inter-extended输出格式。

开发者需要为每个配置选项设置固件生成路径和固件的输出格式以及名称。

### 配置固件生成构建选项

开发者提交项目源码以及工程时，必须选择合适的目标固件生成构建选项，配置生成Batch build构建选项，配置方式如下图：

1. 选择菜单命令 Project🡪Batch build…弹出如下的对话框

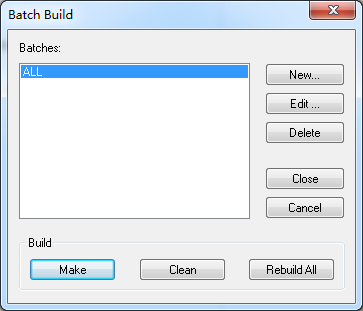


图 7-6

1. 如果对话框中已经有Batch build构建选项，则选择Edit…命令编辑一个Batch build构建选项，进入如下图所示的对话框然后编辑固件生成构建选项名称，然后从左边的列表框选择配置选项到右边的列表框中，如下图7-7所示；如果图7-6所示的对话框中没有Batch build构建选项则需要使用命令New…来创建一个新的固件生成构建选项。

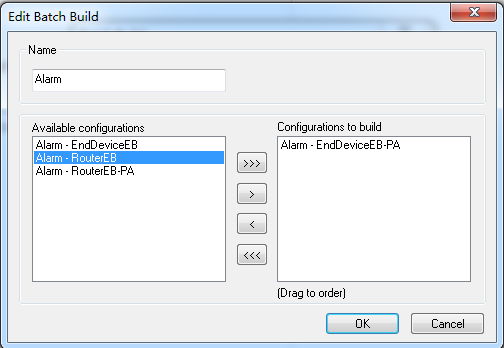


图 7-7

1. 单击OK后回到如下图7-8所示，然后单击Rebuild All来编译已经配置好的固件生成构建选项Alarm

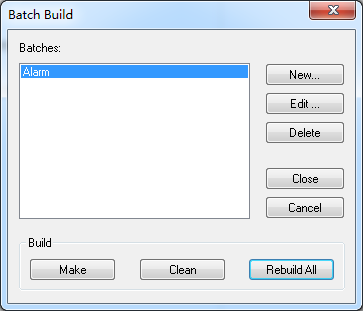


图 7-8

## 固件的生成

将已经设置好固件输出配置的项目工程文件以及源码，放入到适用的SmartRoom工程目录\Projects\zstack\SmartRoom\中，如Alarm项目的源码以及工程文件，然后打开该项目的工程文件，执行Project🡪Batch build…并且在弹出的对话框中选择构建选项，执行命令Rebuild All便可在已经设置好的路径下生成固件，选择工程目录Output下的固件右键选择Open Containing Folder…即可打开当前固件在硬盘上在的目录。