

Telink

## 应用文档

# 泰凌 Zigbee SDK 开发指南

AN-19052901-C2

Ver.1.2.0

2021/05/08

## 简介

本文档为泰凌微电子 Zigbee SDK 的开发指南。

**Published by**

**Telink Semiconductor**

**Bldg 3, 1500 Zuchongzhi Rd,  
Zhangjiang Hi-Tech Park, Shanghai, China**

**© Telink Semiconductor**

**All Rights Reserved**

**Legal Disclaimer**

This document is provided as-is. Telink Semiconductor reserves the right to make improvements without further notice to this document or any products herein. This document may contain technical inaccuracies or typographical errors. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein.

Copyright © 2021 Telink Semiconductor (Shanghai) Co., Ltd.

**Information:**

For further information on the technology, product and business term, please contact Telink Semiconductor Company ([www.telink-semi.com](http://www.telink-semi.com)).

For sales or technical support, please send email to the address of:

[telinkcnsales@telink-semi.com](mailto:telinkcnsales@telink-semi.com)

[telinkcnsupport@telink-semi.com](mailto:telinkcnsupport@telink-semi.com)

# 版本历史

版本	主要改动	日期	作者
1.0.0	初始版本	2019/06	YB, WJZ, Cynthia
1.1.0	1、添加 2.2 TLSR9 RISC-V SDK 安装； 2、添加 2.3 App 版本管理； 3、更新 2.4.2 Flash 地址分配说明； 4、添加 4.3.4 网络安全； 5、添加 4.4.1 AF。	2020/10	YB, WJZ
1.1.1	1、将 8269 (b86) 、8258 (b85) 、8278 (b87) 归为 B85m，将 9518 (b91) 归为 B91m； 2、添加 4.3.11 NV 存储、4.3.12 异常处理； 3、更新 4.4 章节常用 APIs。	2021/01	YB, WJZ
1.2.0	1、新增 2.4 运行模式章节； 2、新增 4.1 运行模式选择和 4.2 芯片类型选择； 3、更新 4.5.8 软件定时任务和 4.5.9 睡眠和唤醒； 4、新增 4.7.4 OTA Image。	2021/04	YB, WJZ

# 目录

版本历史 .....	2
1. 概述 .....	11
1.1    Zigbee 概述 .....	11
1.1.1    设备类型 .....	11
1.1.2    网络类型 .....	11
1.1.3    基本概念 .....	12
1.2    泰凌 Zigbee SDK 概述 .....	14
1.2.1    泰凌 Zigbee SDK 软件开发环境 .....	14
1.2.2    泰凌 Zigbee SDK 支持的硬件平台 .....	15
2. 泰凌 Zigbee SDK .....	16
2.1    TLSR8 TC32 SDK 安装 .....	16
2.1.1    项目导入 .....	16
2.1.2    工程目录结构 .....	18
2.1.3    编译选项 .....	19
2.1.4    添加新项目 .....	20
2.1.5    项目配置说明 .....	22
2.2    TLSR9 RISC-V SDK 安装 .....	27
2.2.1    项目导入 .....	27
2.2.2    工程目录结构 .....	29
2.2.3    编译选项 .....	30
2.2.4    添加新项目 .....	31

2.2.5	项目配置说明 .....	34
2.3	App 版本管理 .....	38
2.3.1	制造商代码 .....	39
2.3.2	固件类型 .....	39
2.3.3	文件版本 .....	39
2.4	运行模式 .....	40
2.4.1	两种模式优缺点比较 .....	40
2.4.2	多地址启动模式 Flash 分布 .....	41
2.4.3	Bootloader 启动模式 Flash 分布 .....	43
2.5	Flash 分布说明 .....	44
2.6	固件烧录 .....	45
3.	软件架构 .....	48
3.1	目录说明 .....	48
3.1.1	硬件平台目录 .....	48
3.1.2	通用函数目录 .....	49
3.1.3	Zigbee 协议栈目录 .....	50
3.1.4	应用层目录 .....	51
3.1.5	工程编译目录 .....	52
3.2	抽象层驱动 .....	52
3.2.1	平台初始化 .....	52
3.2.2	射频 (RF) .....	52
3.2.3	GPIO .....	54
3.2.4	UART .....	55

---

3.2.5	ADC.....	56
3.2.6	PWM .....	56
3.2.7	TIMER.....	57
3.2.8	Watchdog.....	57
3.2.9	Systm Tick.....	58
3.2.10	电压检测 .....	58
3.2.11	睡眠和唤醒 .....	59
3.3	存储管理 .....	60
3.3.1	动态内存管理 .....	60
3.3.2	NV 管理.....	60
3.4	任务管理 .....	62
3.4.1	单次任务队列 .....	62
3.4.2	常驻任务队列 .....	63
3.4.3	软件定时任务 .....	63
4.	Zigbee SDK 应用开发 .....	66
4.1	运行模式选择 .....	66
4.2	硬件选择 .....	66
4.2.1	芯片型号确认 .....	66
4.2.2	目标板选择 .....	67
4.3	打印调试配置 .....	68
4.3.1	UART 打印 .....	68
4.3.2	USB 打印 .....	69
4.4	Zigbee 网络开发流程 .....	69
4.4.1	应用层初始化 (user_init) .....	69

4.4.2	BDB 流程 .....	71
4.4.3	网络安全 .....	73
4.4.4	数据交互 .....	76
4.4.5	网络参数配置 .....	78
4.4.6	系统异常处理 .....	79
5.	常用 APIs .....	80
5.1	应用框架 AF (Application Framework) .....	80
5.1.1	af_endpointRegister() .....	80
5.1.2	af_dataSend() .....	80
5.2	基础设备行为 BDB (Base Device Behaviour) .....	81
5.2.1	bdb_init() .....	81
5.2.2	bdb_networkFormationStart() .....	82
5.2.3	bdb_networkSteerStart() .....	82
5.2.4	bdb_networkTouchLinkStart() .....	82
5.2.5	bdb_findAndBindStart() .....	83
5.2.6	bdb_defaultReportingCfg() .....	83
5.3	网络管理 Network Management .....	84
5.3.1	zb_init() .....	84
5.3.2	zb_zdoCbRegister() .....	84
5.3.3	zb_setPollRate() .....	85
5.3.4	zb_rejoinReq() .....	85
5.3.5	zb_factoryReset() .....	86
5.3.6	zb_mgmtPermitJoinReq() .....	86
5.3.7	zb_mgmtLeaveReq() .....	87
5.3.8	zb_mgmtLqiReq() .....	88
5.3.9	zb_mgmtBindReq() .....	88
5.3.10	zb_mgmtNwkUpdateReq() .....	89
5.3.11	zb_zdoBindUnbindReq() .....	89

5.3.12	zb_zdoNwkAddrReq()	90
5.3.13	zb_zdoleeeAddrReq()	91
5.3.14	zb_zdoSimpleDescReq()	91
5.3.15	zb_zdoNodeDescReq()	92
5.3.16	zb_zdoPowerDescReq()	93
5.3.17	zb_zdoActiveEpReq()	93
5.3.18	zb_zdoMatchDescReq()	94
5.3.19	zb_isDeviceFactoryNew()	94
5.3.20	zb_isDeviceJoinedNwk()	95
5.3.21	zb_getMacAssocPermit()	95
5.3.22	zb_apsExtPanidSet()	95
5.4	ZCL	96
5.4.1	zcl_init()	96
5.4.2	zcl_register()	96
5.5	OTA	97
5.5.1	ota_init()	97
5.5.2	ota_queryStart()	98
6.	OTA	99
6.1	OTA 初始化	99
6.2	OTA Server	99
6.3	OTA Client	100
6.4	OTA Image	100
7.	扩展功能 HCI 接口	101
7.1	控制流程图	101
7.2	命令帧格式	102
7.3	应答格式 (Acknowledge format)	102
7.3.1	Message Type	102
7.3.2	Payload	103
7.4	BDB 命令	104

---

7.4.1	Message Type .....	104
7.4.2	Payload.....	104
7.5	网络管理命令 (Network management command) .....	107
7.5.1	Message Type (Host) .....	107
7.5.2	Payload (Host) .....	108
7.5.3	Message Type (Slave).....	117
7.5.4	Payload (Slave).....	118
7.6	ZCL Cluster 命令 .....	126
7.6.1	ZCL 命令 header 格式 .....	126
7.6.2	General cluster command .....	127
7.6.3	Basic cluster command.....	134
7.6.4	Group cluster command .....	135
7.6.5	Identify cluster command .....	140
7.6.6	On/Off cluster command .....	142
7.6.7	Level cluster command .....	143
7.6.8	Scene cluster command .....	147
7.6.9	OTA cluster command.....	155
8.	附录：Zigbee 联盟 Pro R21 认证证书 .....	157

# 图目录

图 1-1 Mesh 网络.....	12
图 1-2 Endpoint, Cluster 和 Node 关系示意图.....	14
图 2-1 选择导入工程文件 .....	16
图 2-2 完成工程导入 .....	17
图 2-3 工程目录结构 .....	18
图 2-4 选择编译 .....	19
图 2-5 输出文件 .....	20
图 2-6 添加新项目 .....	21
图 2-7 配置新项目 .....	21
图 2-8 符号定义 .....	23
图 2-9 库文件和路径 .....	24
图 2-10 link 文件预编译设置 .....	25
图 2-11 image 校验设置 .....	26
图 2-12 选择导入工程文件 .....	27
图 2-13 完成工程导入.....	28
图 2-14 工程目录结构.....	29
图 2-15 选择编译.....	30
图 2-16 输出文件.....	31
图 2-17 添加新项目.....	32
图 2-18 配置新项目.....	33

图 2-19 符号定义.....	34
图 2-20 库文件和路径 .....	36
图 2-21 link 文件预编译设置 .....	37
图 2-22 image 校验设置.....	38
图 2-23 二进制文件 .....	39
图 2-24 512k Flash 空间分配图 .....	41
图 2-25 1M Flash 空间分配图 .....	42
图 2-26 512k Flash 空间分配图 .....	43
图 2-27 1M Flash 空间分配图 .....	44
图 2-28 烧录连接 .....	46
图 2-29 烧录工具 .....	47
图 3-1 硬件平台目录 .....	48
图 3-2 通用函数目录 .....	49
图 3-3 Zigbee 协议栈目录 .....	50
图 3-4 应用层目录 .....	51
图 3-5 工程编译目录 .....	52
图 4-1 应用层初始化流程 .....	70
图 4-2 BDB 初始化流程 .....	72
图 4-3 直接入网 .....	74
图 4-4 间接入网 .....	75
图 7-1 控制流程图 .....	101

# 1. 概述

## 1.1 Zigbee 概述

### 1.1.1 设备类型

Zigbee 3.0 包含以下几种设备类型：

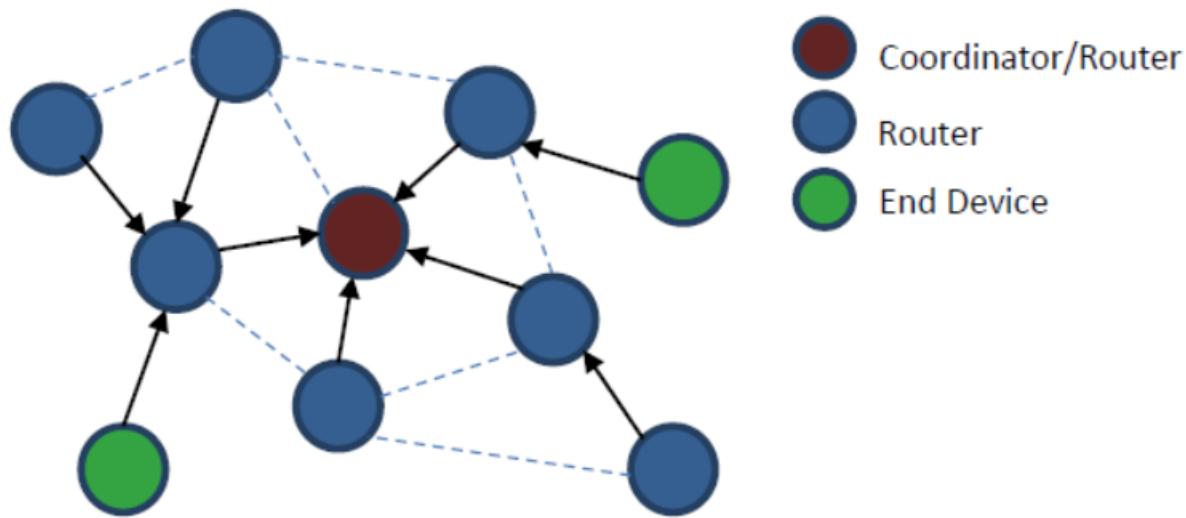
- ✧ 路由器 (Router)：具有路由功能的设备
- ✧ 协调器 (Coordinator)：具有网络管理能力的 Router
- ✧ 终端节点 (End Device)：可支持低功耗休眠模式的设备，不具有路由功能，必须通过其父节点与网络中其它节点进行信息交互。
- ✧ GPD (Green Power Device)：低功耗无源设备。本文不作介绍。

### 1.1.2 网络类型

Zigbee3.0 的网络属于 Mesh 网络，根据创建者的不同，可以分为集中式(Central)网络和分布式(Distribute)网络两种类型，这两种类型的网络在对网络中的 link key 以及 nwk key 的维护上有所区别。

- ✧ Central 网络：通常是由 Coordinator 作为 trust center (信任中心) 来组建和管理的网络。
- ✧ Distribute 网络：通过有 Router 功能的节点来组建网络。

图 1-1 Mesh 网络



### 1.1.3 基本概念

1) PAN ID: 个人局域网 (Personal Area Network) ID。

由 PAN ID 来标识所组建的网络。PAN ID 由负责组建该网络的节点分配，可以直接指定，也可以随机生成，但需要通过 active scan 来避免 PAN ID 冲突。

2) Channel: 频点/信道。

Zigbee 允许的工作频点(2.4G):  $2405 + (N-11)*5$  (mHz), (信道 N = 11~26)。

接入一个网络后采用单一频点的工作模式，不会主动跳频。

3) Node: 一个物理的设备。

具有唯一的 MAC 地址 (64Bits)，组建或者加入网络后具有该 PAN 中唯一的短地址 (16 Bits 的 network address)，从而可以在之后的数据交互中直接使用该短地址。

4) **Endpoint:** 端口， 对应具体的应用 (Apps) 。

一个 Node 可以包含多个应用 (多个端口) 。用户层的应用都是基于不同端口的操作。

端口号范围：0~255，对于端口号的使用，Zigbee 联盟有具体的规定。

- ✧ 0: 用于 ZDO (Zigbee Device Object)，是每个 Node 必须具有的。
- ✧ 1~240: 用户级应用使用，也就是应用开发中可以任意使用的端口。
- ✧ 241~254: Zigbee 联盟具体规定的一些应用，比如 242 仅用于 Green Power。
- ✧ 255: 广播端口，数据要广播给所有的应用端口。

5) **Clusters:** 簇。

一个具体的应用 (端口) 是由一组不同的簇组成，从而完成不同的行为。ZCL ( Zigbee Cluster Library) 定义了一系列 cluster 所遵循的行为规范。

6) **Attributes:** 属性。

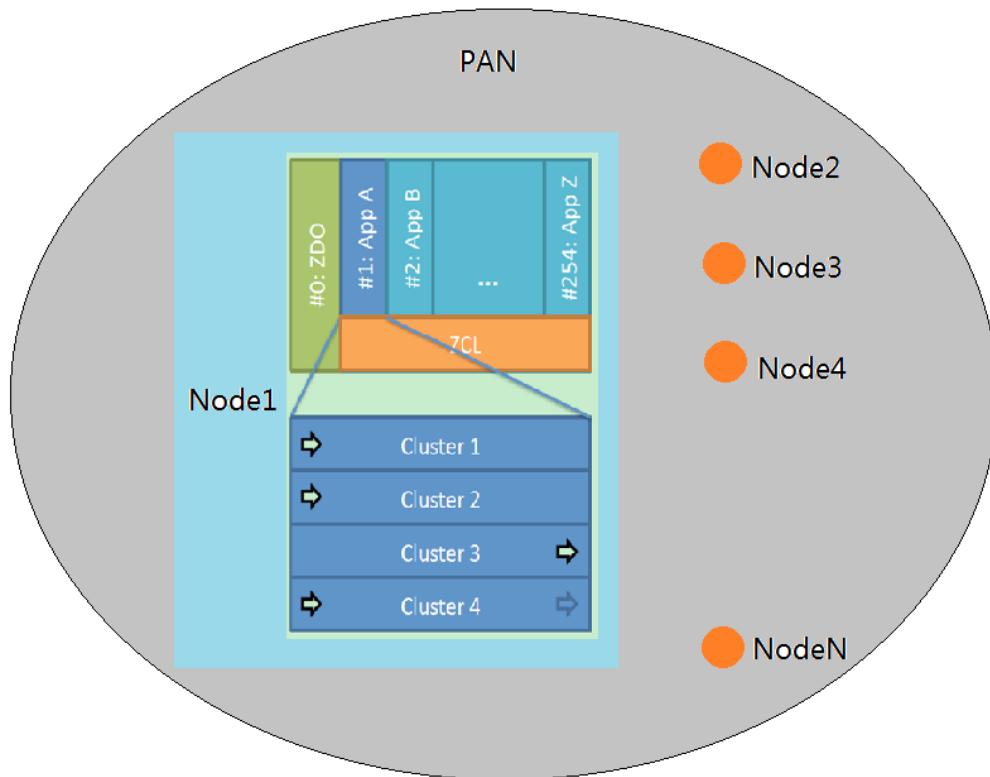
ZCL 定义了各个 cluster 所支持的属性，还定义了这些属性值的操作权限。

7) **BDB:** 基本设备行为规范 (Base Device Behavior) 。

规范了节点启动、组网、入网、密钥管理以及复位 (reset) 的行为。

从本质上说，应用层的操作对象就是某一端口(Endpoint)上的某个簇(Cluster)的某个属性(attribute)。它们之间的关系如下图：

图 1-2 Endpoint, Cluster 和 Node 关系示意图



## 1.2 泰凌 Zigbee SDK 概述

Telink Zigbee SDK 是基于 Zigbee PRO 规范开发的一套已通过联盟平台 Zigbee Pro R21 认证、支持 Pro R22 并符合 Zigbee3.0 应用规范的 Zigbee 协议栈。

### 1.2.1 泰凌 Zigbee SDK 软件开发环境

#### 1) 必备软件工具 (下载地址: <http://wiki.telink-semi.cn/>)

✧ 集成开发环境:

TLSR8 Chips: Telink IDE for TC32

TLSR9 Chips: Telink RDS IDE for RISC-V

✧ 下载调试工具: Telink Burning and Debugging Tools

- ✧ OTA 编码转换工具: tl\_ota\_tool
- 2) 抓包分析辅助工具 (自行下载或购买)
- ✧ TI Packet Sniffer
  - ✧ Ubiqua
- 3) 软件开发包 (下载地址: <http://wiki.telink-semi.cn/>)
- ✧ tl\_zigbee\_sdk\_v3.x.x.zip
- 4) PC 辅助控制软件 (ZGC)
- ✧ Zigbee\_gateway\_controller.exe

## 1.2.2 泰凌 Zigbee SDK 支持的硬件平台

- ✧ B85m (TC32 平台)
  - 8269: 8269 EVK Board 和 8269 USB Dongle
  - 8258: 8258 EVK Board 和 8258 USB Dongle
  - 8278: 8278 EVK Board 和 8278 USB Dongle
- ✧ B91m (RISC-V 平台)
  - 9518: 9518 EVK Board 和 9518 USB Dongle

## 2. 泰凌 Zigbee SDK

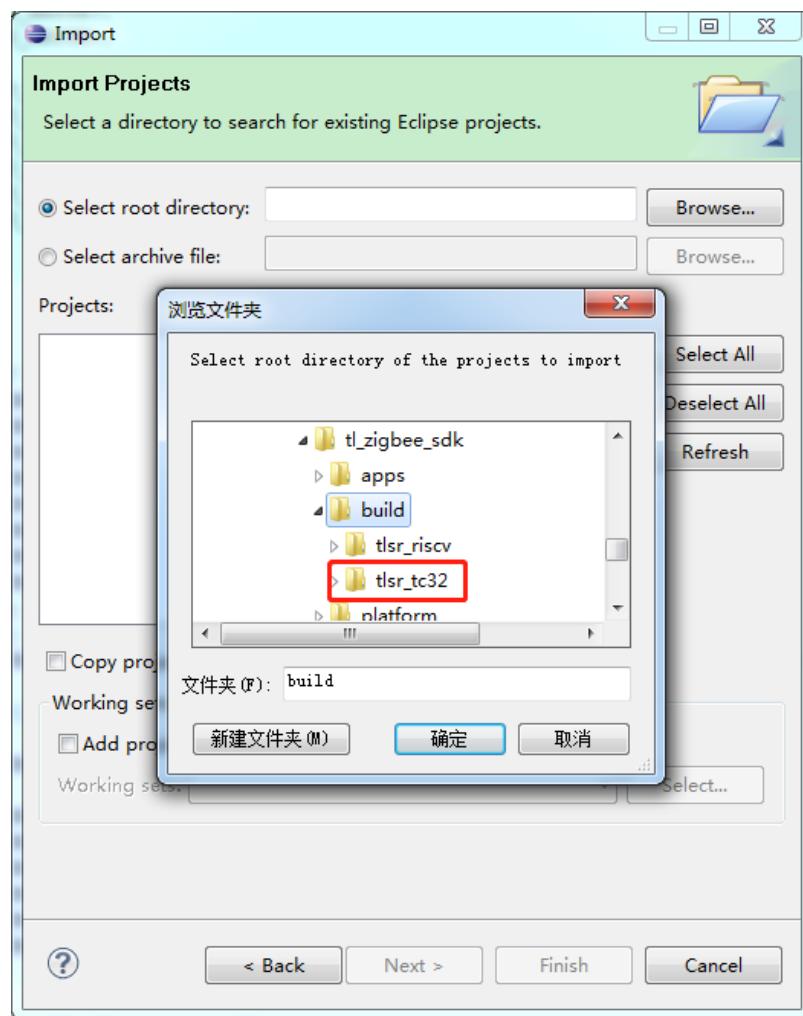
在安装 SDK 前, 请根据 1.2.1 章节安装相应的集成开发环境 Telink IDE for TC32 或 Telink RDS IDE for RISC-V。

### 2.1 TLSR8 TC32 SDK 安装

#### 2.1.1 项目导入

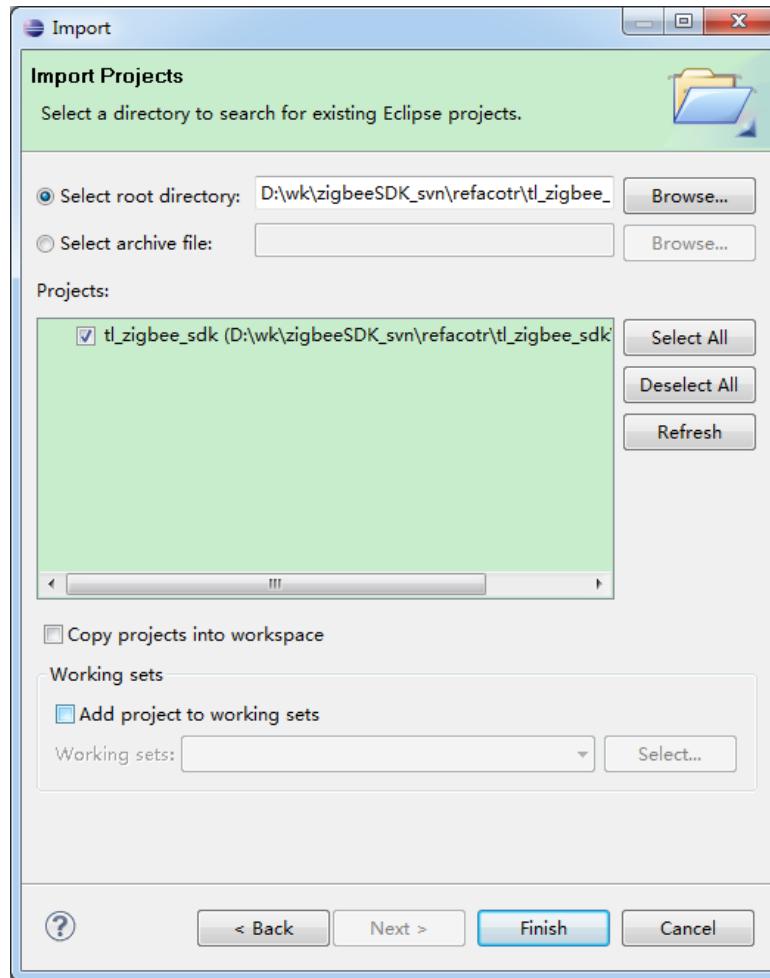
- 1) 打开 IDE, 依次进入界面 File -> Import -> Existing Projects into Workspace。
- 2) 选择 tl\_zigbee\_sdk/build 目录下的 tlsr\_tc32 -> 点击“确定”。

图 2-1 选择导入工程文件



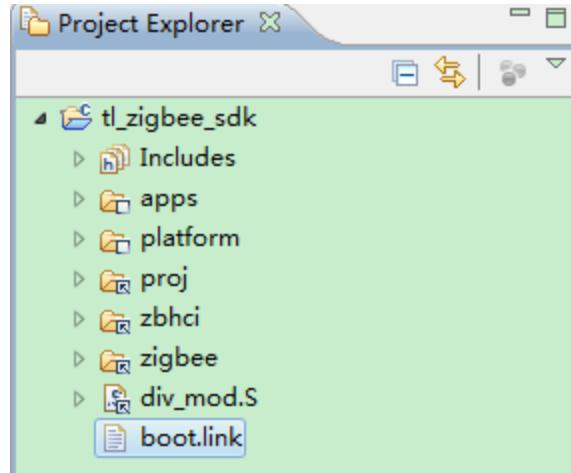
3) 点击“Finish”，完成工程导入。

图 2-2 完成工程导入



## 2.1.2 工程目录结构

图 2-3 工程目录结构



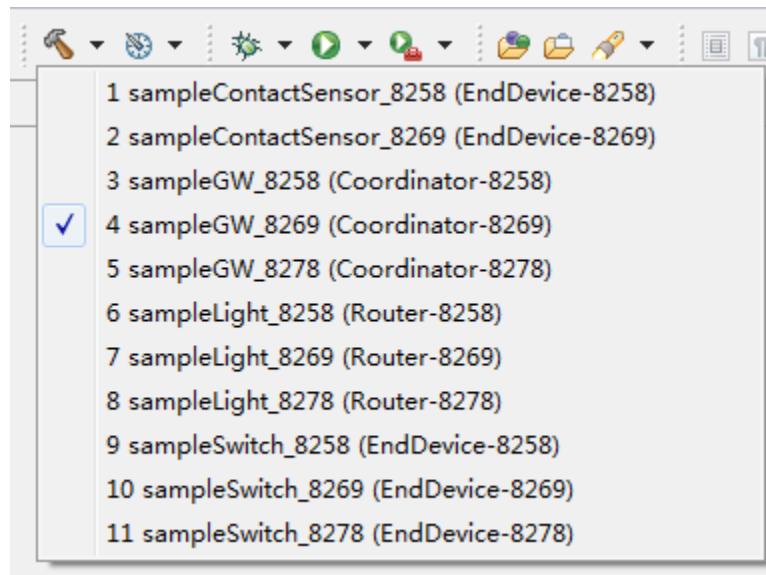
- ✧ /apps: 用户工程目录。
  - /common: 应用层公共代码目录，包含主函数 (main.c)、模块测试函数 (module\_test.c) 和版本及模式配置 (comm\_cfg.h) 等。
  - /sampleGW: 网关 (Coordinator) 例程。
  - /sampleLight: 灯 (Router) 例程。
  - /sampleSwitch: 开关 (End Device) 例程。
  - /bootLoader: 引导程序。
- ✧ /platform: 运行平台目录。
  - /boot: 启动和链接文件。
  - /chip\_xx: 芯片驱动文件。
  - /services: 中断服务函数文件。
- ✧ /proj: 工程代码目录。

- /common: 通用代码目录。
- /drivers: 抽象层驱动文件。
- /os: 任务事件、buffer 管理函数文件。
- ✧ /zigbee: 协议栈相关目录。
- ✧ /zbhci: hci 命令处理相关目录。
- ✧ div\_mod.S: 除法和取余相关的汇编函数。 (TLSR8 不支持硬件除法器)

### 2.1.3 编译选项

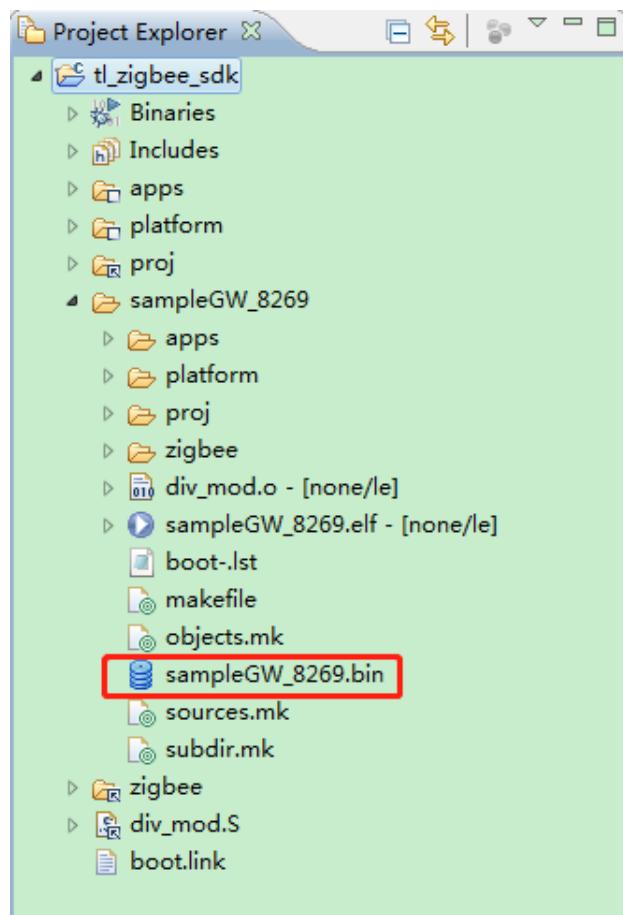
点击下拉图标 “

图 2-4 选择编译



编译完成后，在“Project Explorer”窗口会出现编译出来的文件夹，其中包含了编译出来的固件，如下图。

图 2-5 输出文件



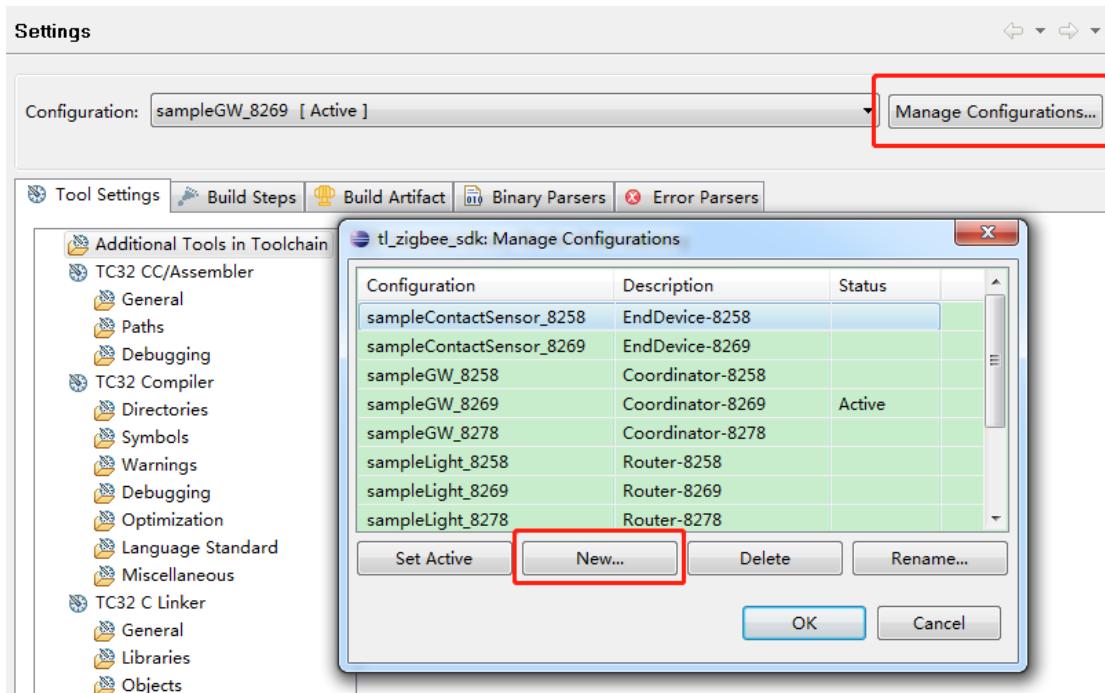
## 2.1.4 添加新项目

在提供的 SDK 中，仅仅给出了一些简单的用例。用户可以根据需求，自行添加应用工程以及编译选项。

具体步骤如下：

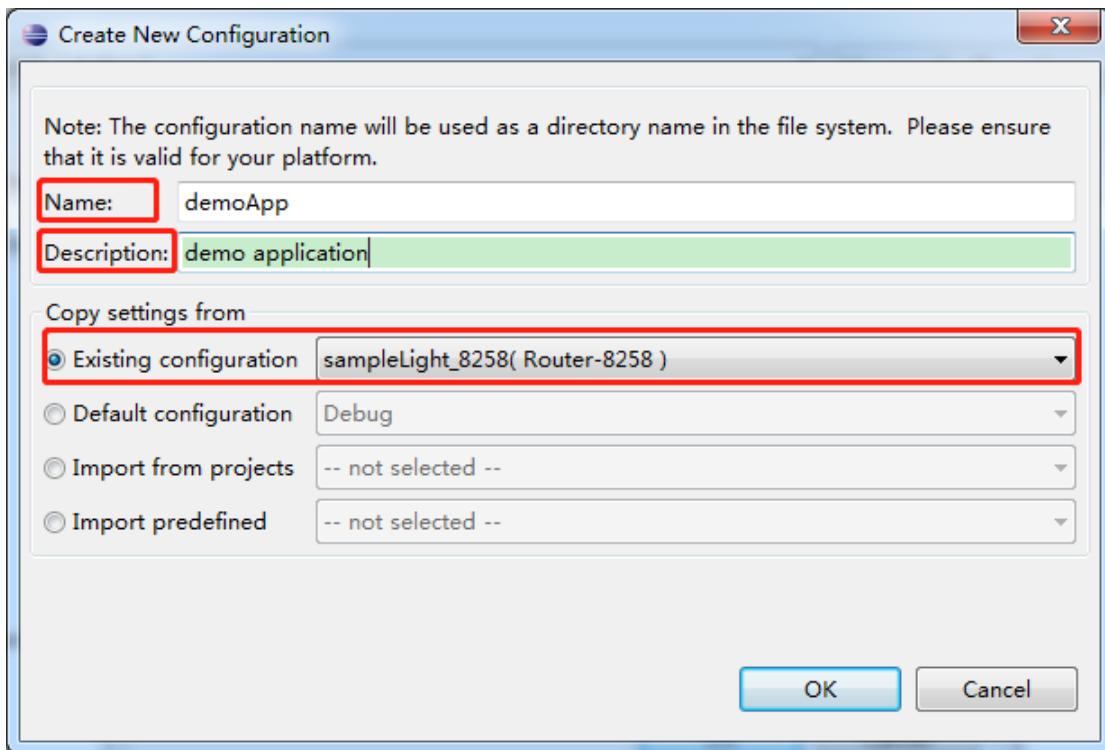
- 1) 步骤 1：Project Explorer -> tl\_zigbee\_sdk -> Properties -> Manage Configurations

图 2-6 添加新项目



2) 步骤 2: 单击 New, 弹出如下界面

图 2-7 配置新项目



- ✧ Name: 项目名称
- ✧ Description: 项目简单描述
- ✧ Copy settings from: 建议选用 Existing configuration, 可以减少一些配置过程。

选用 Existing configuration 的原则如下所示：

- ✧ 使用 8269 平台，选择 xxx\_8269；使用 8258 平台，选择 xxx\_8258。
- ✧ 项目用于 Gateway 的开发，选择 sampleGw\_xxxx；项目用于 Router 设备的开发，选择 sampleLight\_xxxx；项目用于 End Device 设备的开发，选择 sampleSwitch\_xxxx。

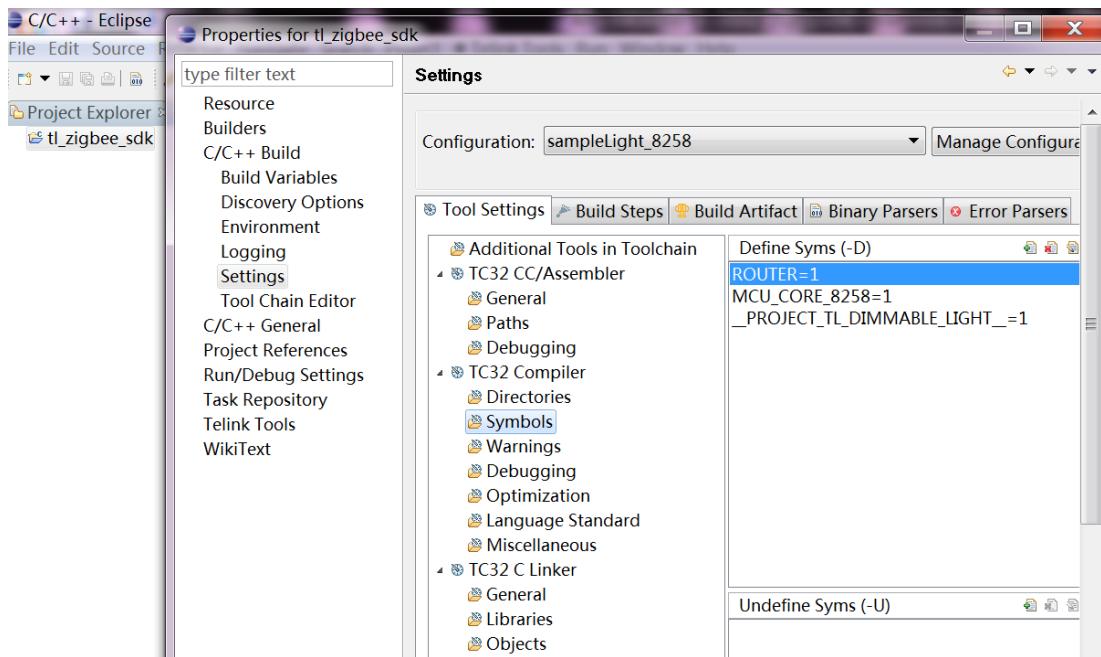
假设需要用 8258 开发一个具有路由功能的灯的项目，根据该原则，应选择项目"sampleLight\_8258"的配置作为这个新项目的配置。

如果需要修改配置，可按下节（2.1.5 项目配置说明），进行相应调整。

## 2.1.5 项目配置说明

依次进入：Project Explorer -> tl\_zigbee\_sdk -> Properties -> Settings （以项目 sampleLight\_8258 为例说明）

图 2-8 符号定义



在 Tool Settings 下，可以看到当前项目的一些预定义配置：

## 1) 设备类型预定义

-DROUTER=1：表示项目是个具有路由功能的设备

对于 coordinator 和 end device 的设备设置分别为：

-DENDE\_DEVICE=1：表示项目是个 end device 功能的设备

-DCOORDINATOR=1：表示项目是个 Coordinator 功能的设备

## 2) 平台选择

✧ 8269 平台: -DMCU\_CORE\_826x=1 以及-DCHIP\_8269=1

启动代码 cstartup\_826x.S 位于\tl\_zigbee\_sdk\platform\boot\826x 目录下。

✧ 8258 平台: -DMCU\_CORE\_8258=1

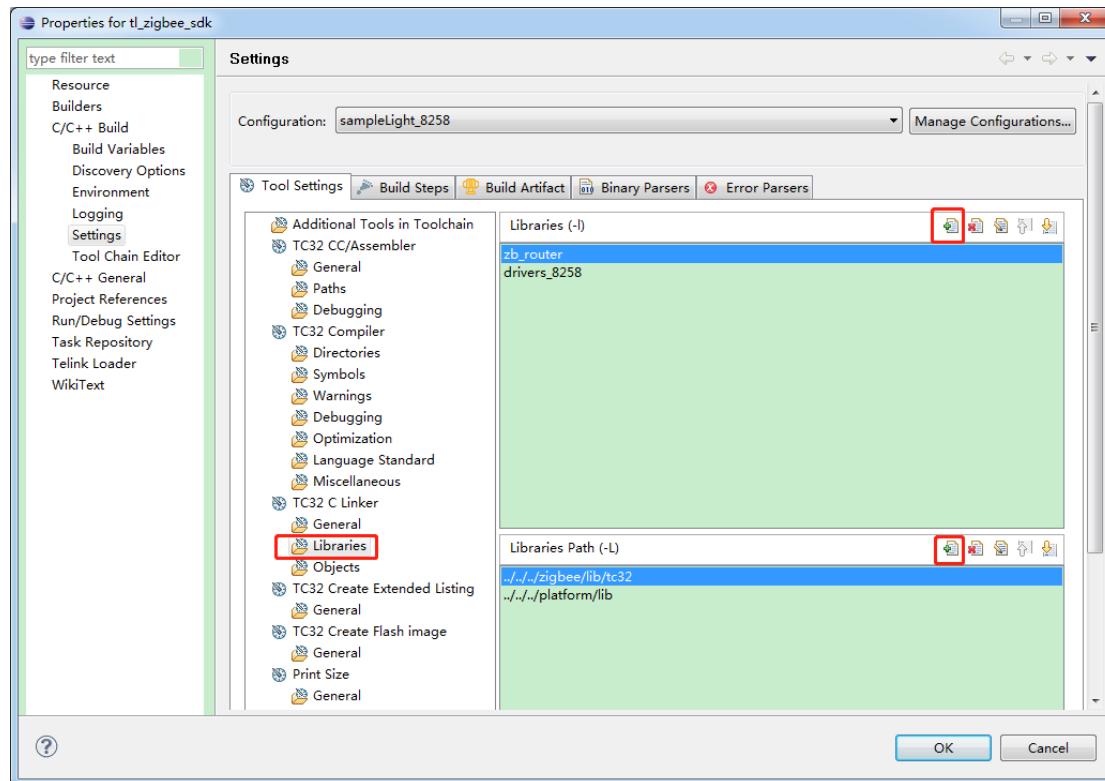
启动代码 cstartup\_8258.S 位于\tl\_zigbee\_sdk\platform\boot\8258 目录下。

- ✧ 8278 平台: -DMCU\_CORE\_8278=1

启动代码 cstartup\_8278.S 位于\tl\_zigbee\_sdk\platform\boot\8278 目录下。

### 3) lib 文件的链接

图 2-9 库文件和路径



当前 SDK 里包括 2 类库文件：zigbee stack 库和平台驱动库。

- ✧ zigbee stack 库: libzb\_router.a, libzb\_coordinator.a, libzb\_ed.a

位于\tl\_zigbee\_sdk\zigbee\lib\tc32 目录下。

平台驱动库: libdrivers\_826x.a, libdrivers\_8258.a, libdrivers\_8278.a 位于\tl\_zigbee\_sdk\platform\lib 目录下。

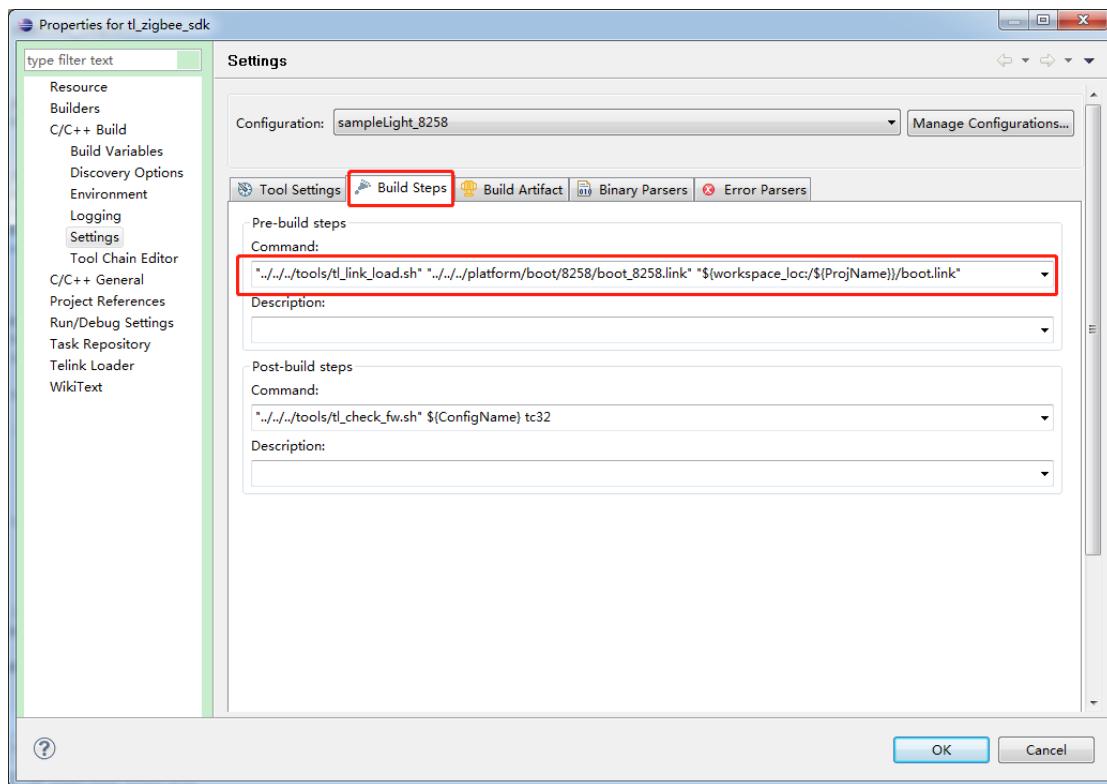
#### 4) 链接文件

用户可以依据实际应用需求, 根据所选用的不同平台以及内存要求, 调整合适的 link 文件。

当前 SDK 给出了 826x、8258 和 8278 平台的默认 link 文件 boot\_826x.link、boot\_8258.link 和 boot\_8278.link, 分别在\tl\_zigbee\_sdk\platform\boot\对应的目录下。

如下图所示, 通过预编译调用脚本 tl\_link\_load.sh (\tl\_zigbee\_sdk\tools 目录下) 来选择使用的 link 文件。

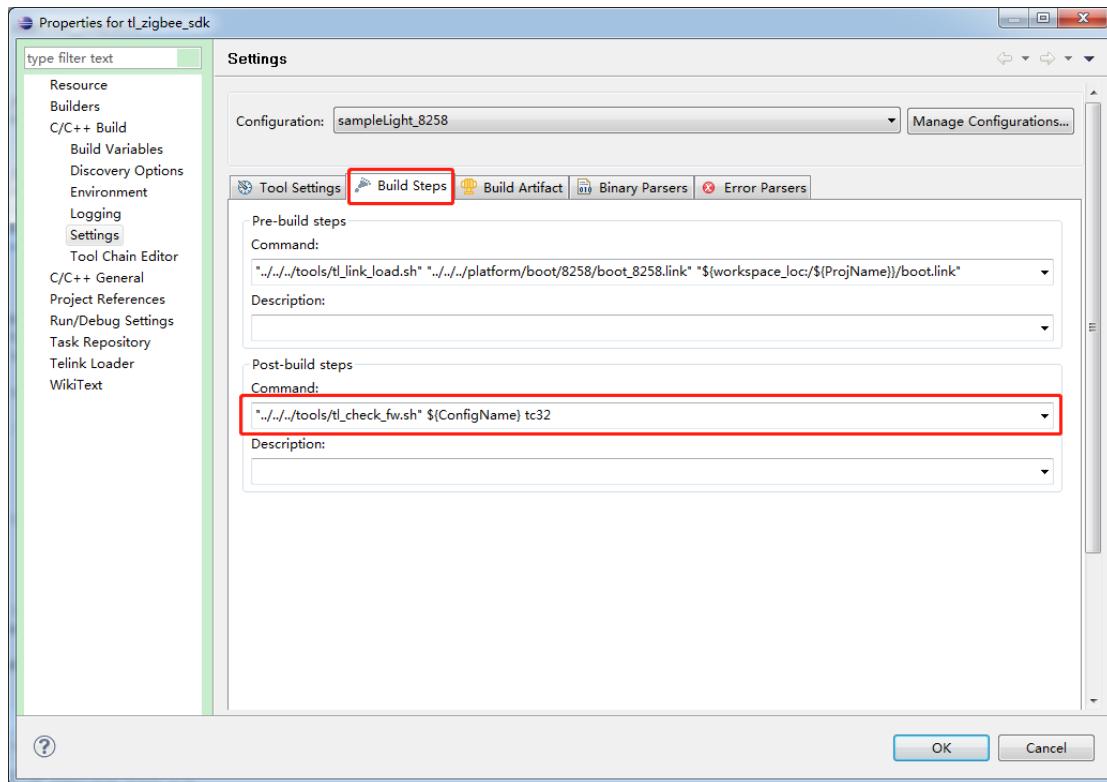
**图 2-10 link 文件预编译设置**



#### 5) .image 文件校验

为了保证下载文件的可靠性，通过脚本 tl\_check\_fw.sh (\tl\_zigbee\_sdk\tools 目录下) 在生成的 image 文件中加入校验字段，OTA 结束后通过检查校验字段是否匹配，来决定是否运行新的 image。

图 2-11 image 校验设置

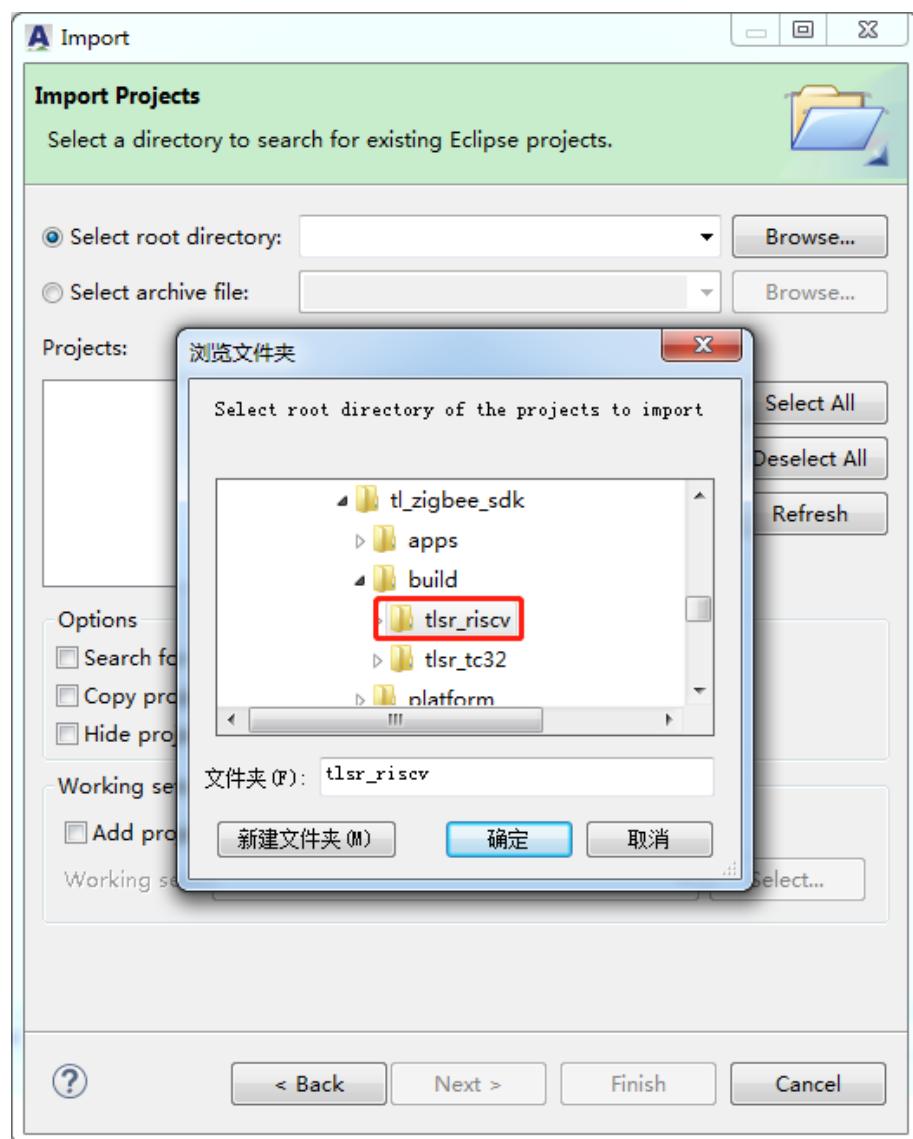


## 2.2 TLSR9 RISC-V SDK 安装

### 2.2.1 项目导入

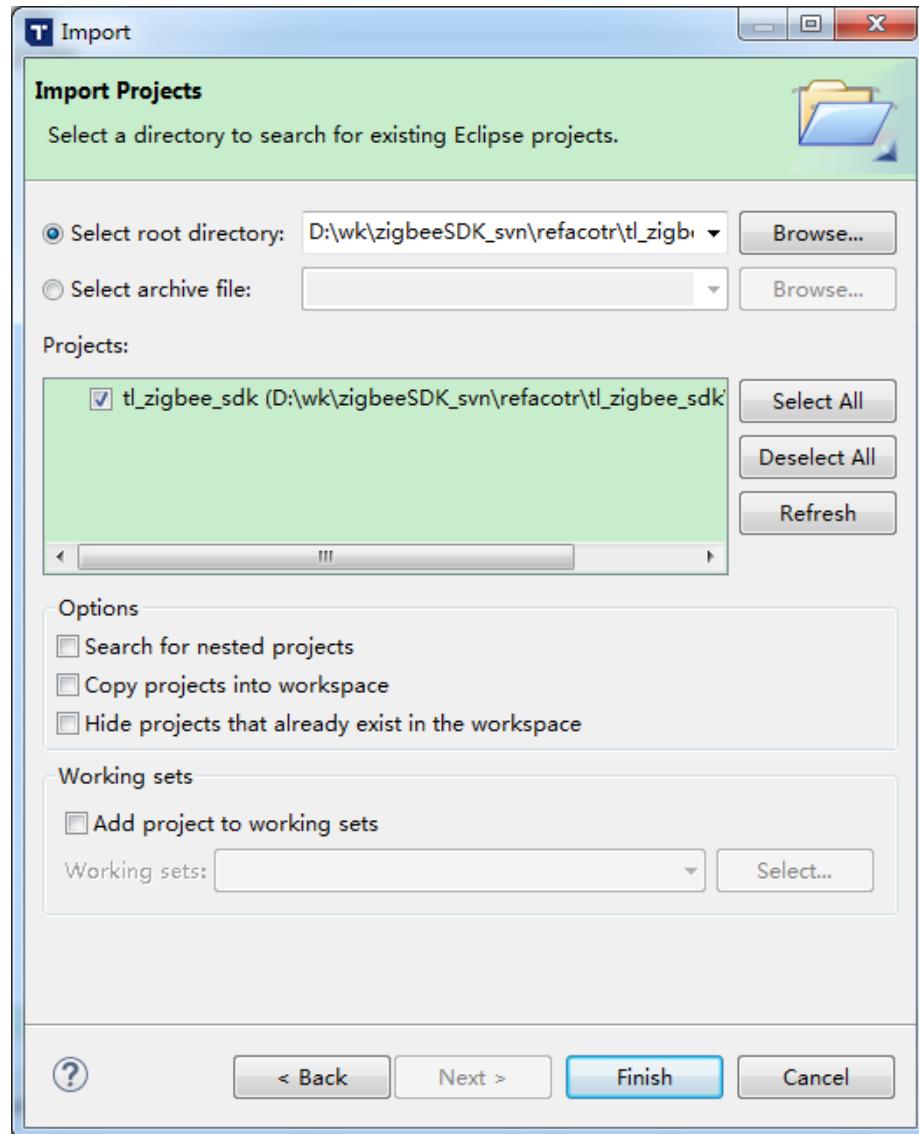
- 1) 打开 IDE，依次进入界面 File -> Import -> Existing Projects into Workspace。
- 2) 选择 tl\_zigbee\_sdk/build 目录下的 tlsr\_riscv -> 点击“确定”。

图 2-12 选择导入工程文件



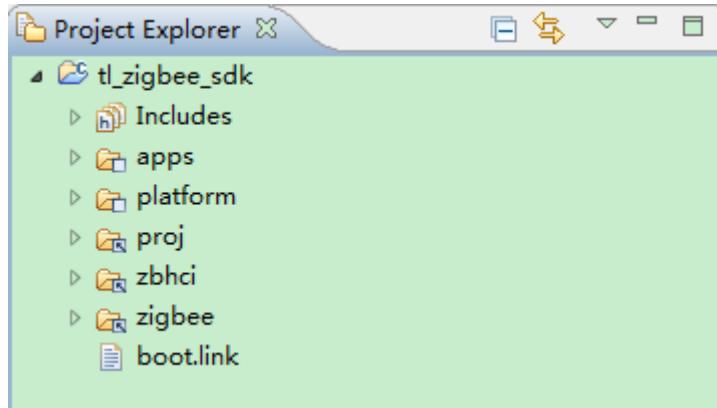
3) 点击“Finish”，完成工程导入。

图 2-13 完成工程导入



## 2.2.2 工程目录结构

图 2-14 工程目录结构



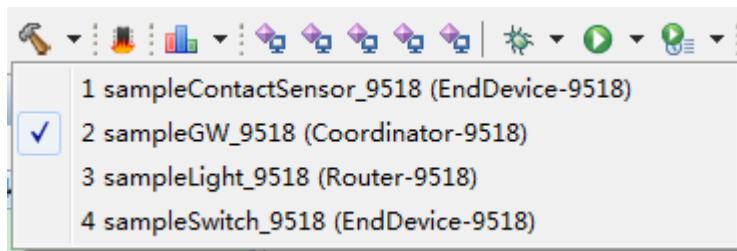
- ✧ /apps: 用户工程目录。
  - /common: 应用层公共代码目录，包含主函数 (main.c)、模块测试函数 (module\_test.c) 和版本及模式配置 (comm\_cfg.h) 等。
  - /sampleGW: 网关 (Coordinator) 例程。
  - /sampleLight: 灯 (Router) 例程。
  - /sampleSwitch: 开关 (End Device) 例程。
  - /bootLoader: 引导程序。
- ✧ /platform: 运行平台目录。
  - /boot: 启动和链接文件。
  - /chip\_xx: 芯片驱动文件。
  - /services: 中断服务函数文件。
- ✧ /proj: 工程代码目录。
  - /common: 通用代码目录。

- /drivers: 抽象层驱动文件。
  - /os: 任务事件、buffer 管理函数文件。
- ✧ /zigbee: 协议栈相关目录。
- ✧ /zbhci: hci 命令处理相关目录。

### 2.2.3 编译选项

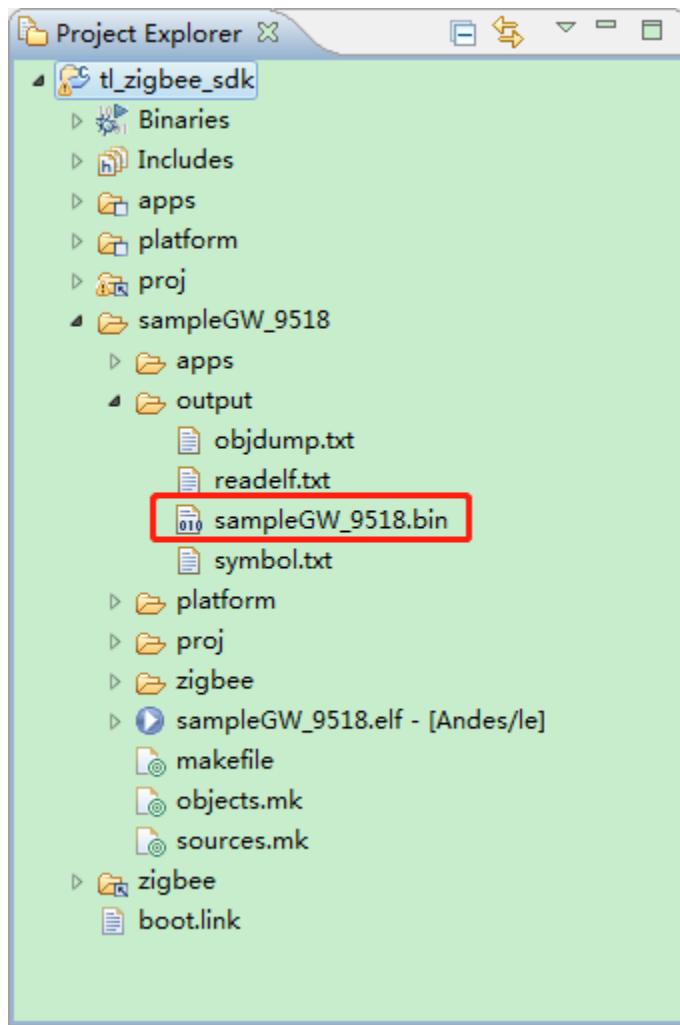
点击下拉图标 “”，可以看到当前所有的编译选项，选择需要编译的示例工程，如下图，等待编译完成。

图 2-15 选择编译



编译完成后，在“Project Explorer”窗口会出现编译出来的文件夹，其中包含了编译出来的固件，如下图。

图 2-16 输出文件



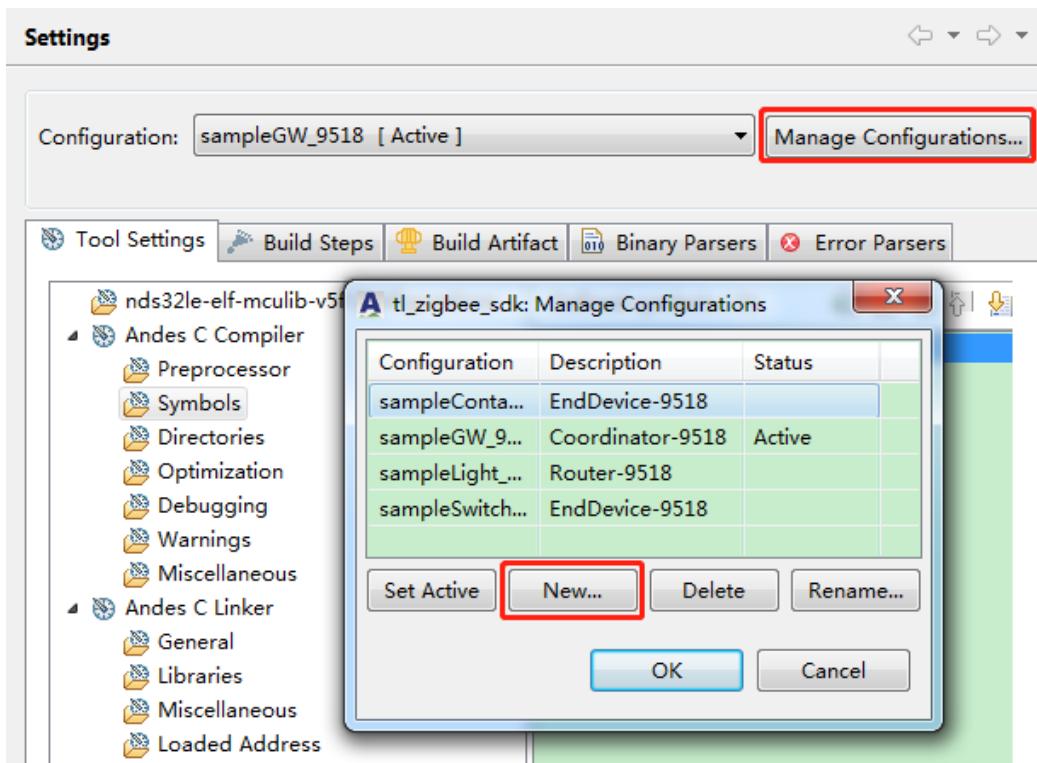
## 2.2.4 添加新项目

在提供的 SDK 中，仅仅给出了一些简单的用例。用户可以根据需求，自行添加应用工程以及编译选项。

具体步骤如下：

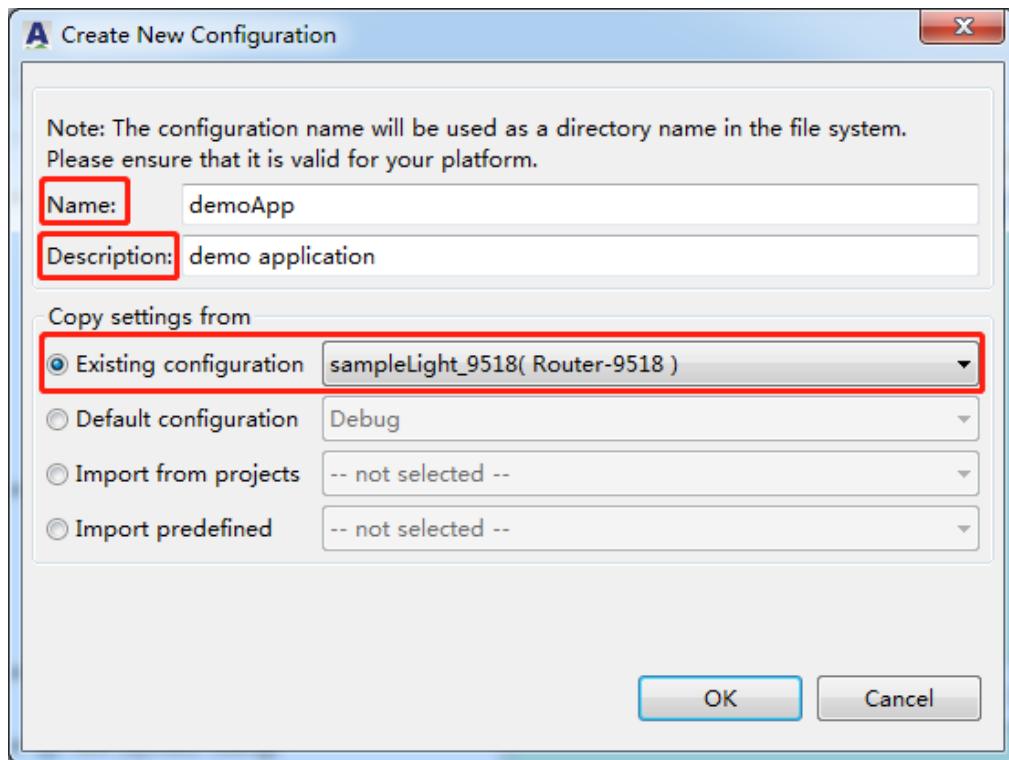
- 1) 步骤 1：Project Explorer -> tl\_zigbee\_sdk -> Properties -> Manage Configurations

图 2-17 添加新项目



2) 步骤 2：单击 New，弹出如下界面

图 2-18 配置新项目



- ✧ Name: 项目名称
- ✧ Description: 项目简单描述
- ✧ Copy settings from: 建议选用 Existing configuration，可以减少一些配置过程。

选用 Existing configuration 的原则如下所示：

- ✧ 使用 9518 芯片，选择 xxx\_9518。
- ✧ 项目用于 Gateway 的开发，选择 sampleGw\_xxxx；项目用于 Router 设备的开发，选择 sampleLight\_xxxx；项目用于 End Device 设备的开发，选择 sampleSwitch\_xxxx。

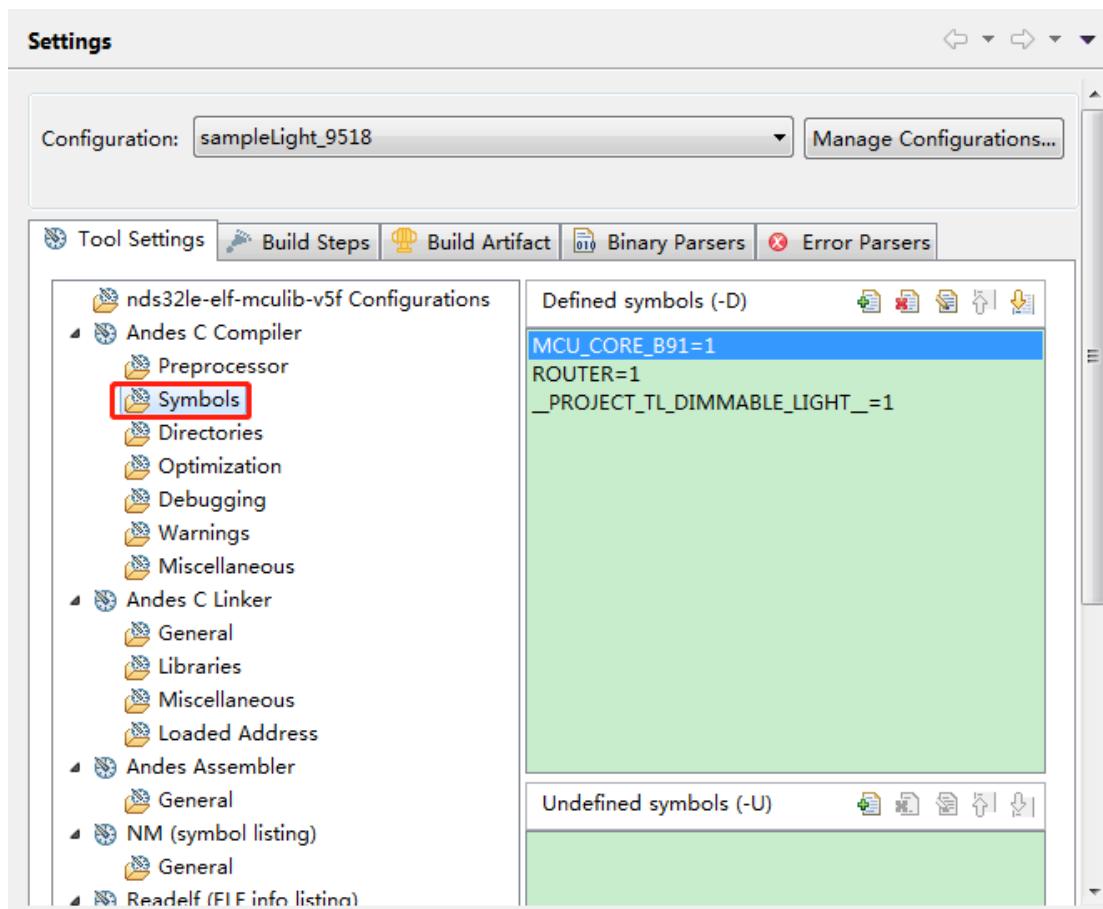
假设需要用 9518 开发一个具有路由功能的灯的项目，根据该原则，应选择项目"sampleLight\_9518"的配置作为这个新项目的配置。

如果需要修改配置，可按下节（2.2.5 项目配置说明），进行相应调整。

## 2.2.5 项目配置说明

依次进入：Project Explorer -> tl\_zigbee\_sdk -> Properties -> Settings （以项目 sampleLight\_9518 为例说明）

图 2-19 符号定义



在 Tool Settings 下，可以看到当前项目的一些预定义配置：

## 1) 设备类型预定义

-DROUTER=1: 表示项目是个具有路由功能的设备

对于 coordinator 和 end device 的设备设置分别为：

-DEND\_DEVICE=1: 表示项目是个 end device 功能的设备

-DCOORDINATOR=1: 表示项目是个 Coordinator 功能的设备

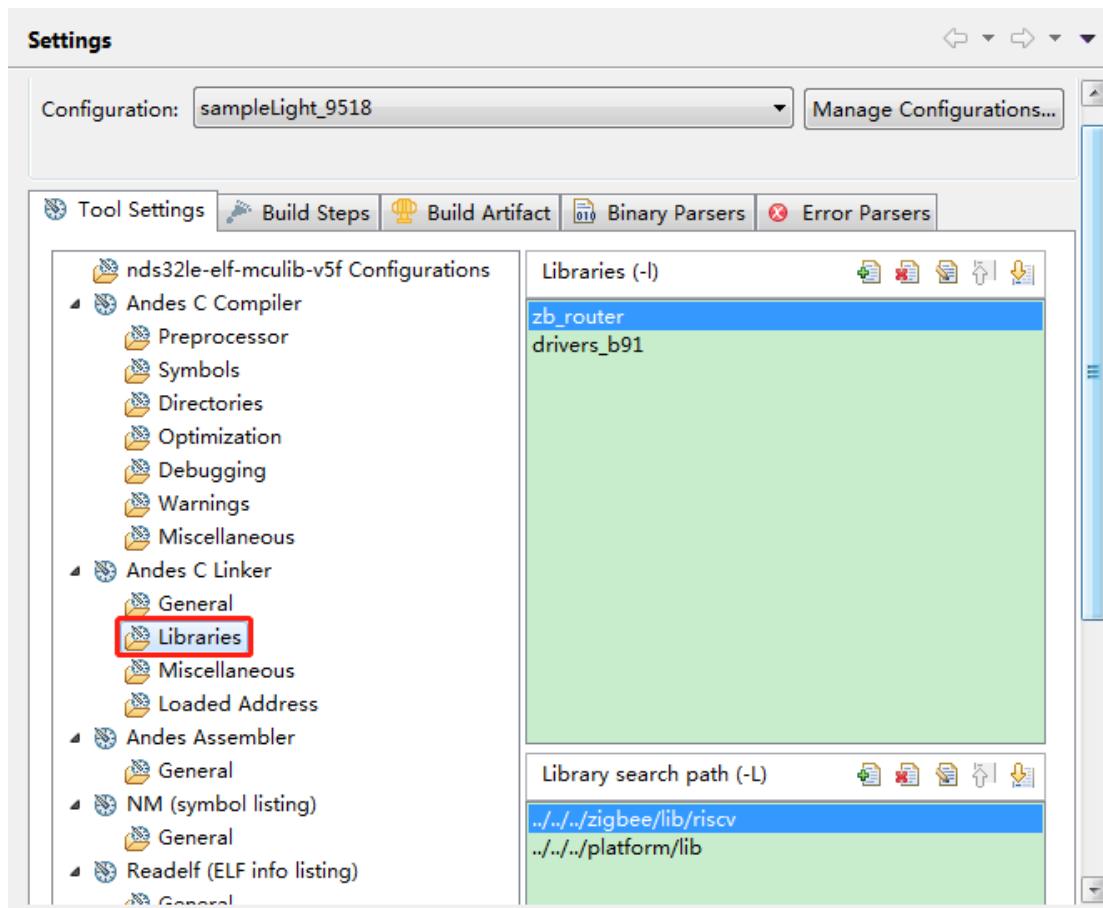
## 2) 平台选择

✧ b91 平台: -DMCU\_CORE\_B91=1

启动代码 cstartup\_b91.S 位于\tl\_zigbee\_sdk\platform\boot\b91 目录下。

## 3) lib 文件的链接

图 2-20 库文件和路径



当前 SDK 里包括 2 类库文件：zigbee stack 库和平台驱动库。

❖ zigbee stack 库： libzb\_router.a, libzb\_coordinator.a, libzb\_ed.a

位于\tl\_zigbee\_sdk\zigbee\lib\riscv 目录下。

平台驱动库： libdrivers\_b91.a 位于\tl\_zigbee\_sdk\platform\lib 目录下。

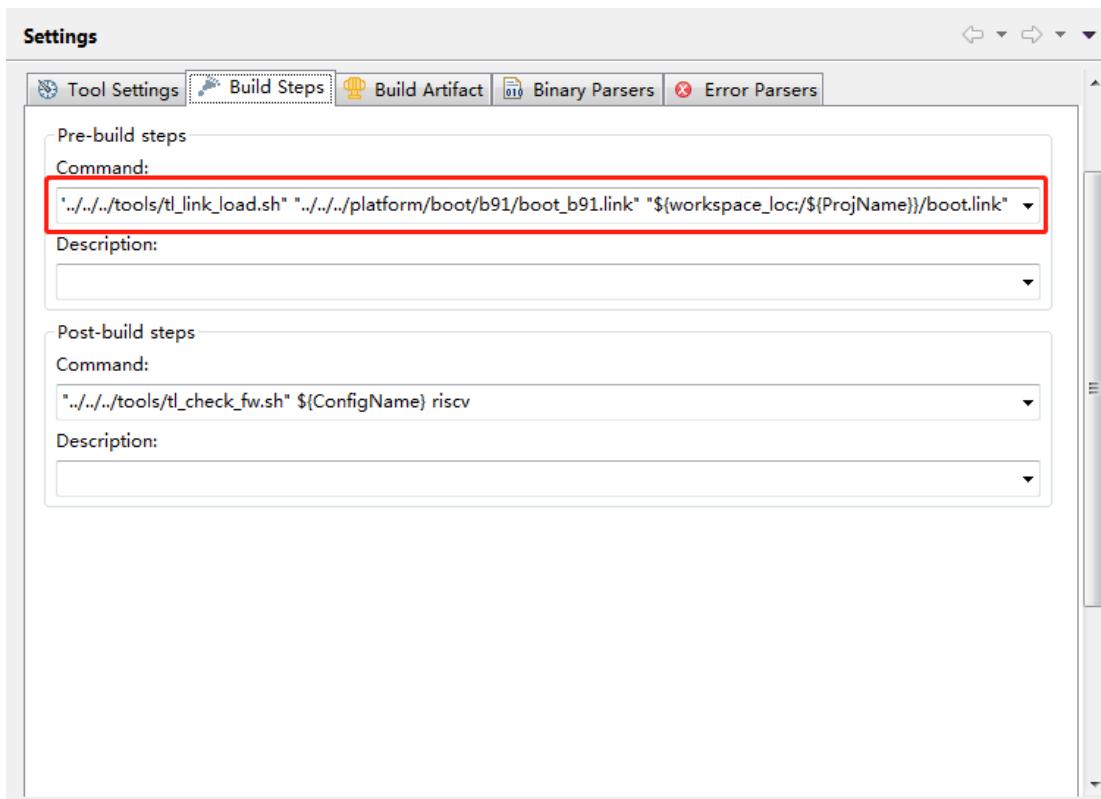
#### 4) 链接文件

用户可以依据实际应用需求，根据所选用的不同平台以及内存要求，调整合适的 link 文件。

当前 SDK 给出了 b91 平台的默认 link 文件 boot\_b91.link，在\tl\_zigbee\_sdk\platform\boot\对应的目录下。

如下图所示，通过预编译调用脚本 tl\_link\_load.sh (\tl\_zigbee\_sdk\tools 目录下) 来选择使用的 link 文件。

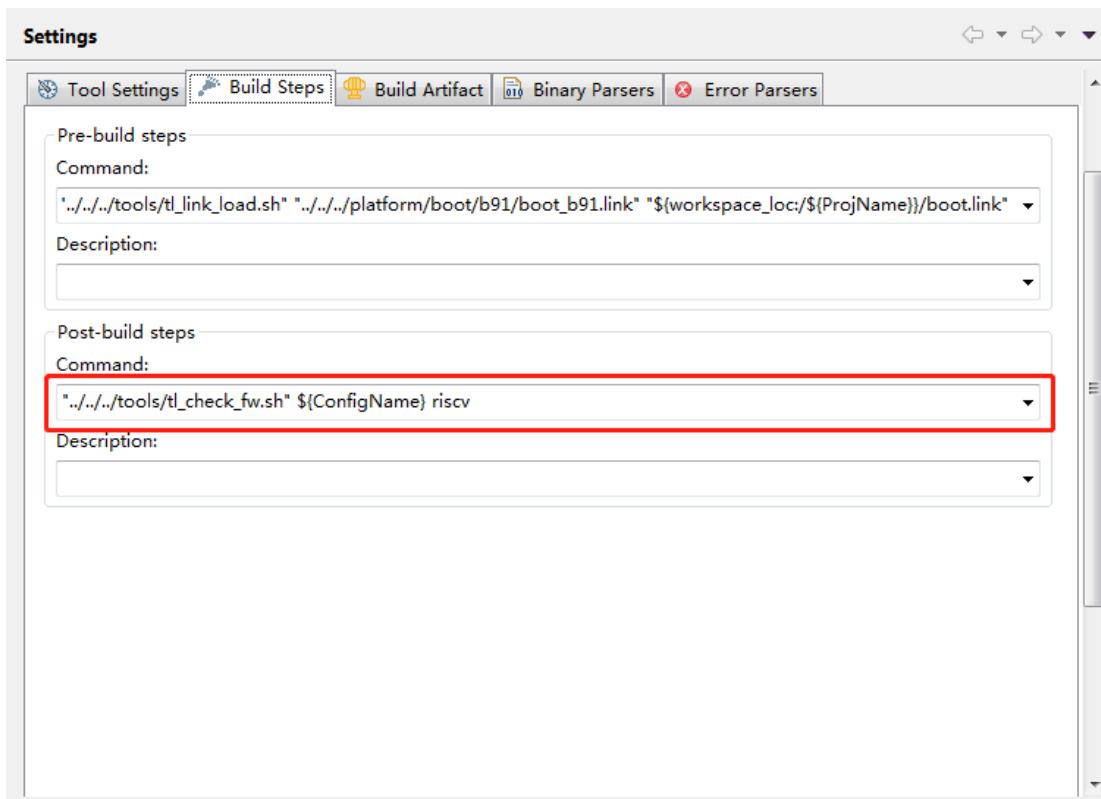
图 2-21 link 文件预编译设置



## 5) .image 文件校验

为了保证下载文件的可靠性，通过脚本 tl\_check\_fw.sh（\tl\_zigbee\_sdk\tools 目录下）在生成的 image 文件中加入校验字段，OTA 结束后会通过检查校验字段是否匹配，来决定是否运行新的 image。

图 2-22 image 校验设置

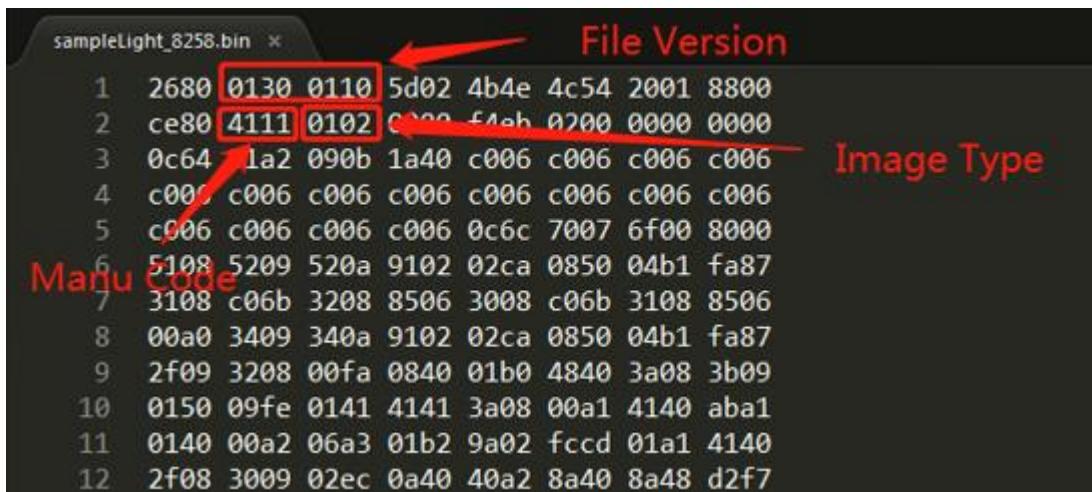


## 2.3 App 版本管理

从 SDK V3.6.0 开始，每个 demo 目录下都对应有一个 version\_cfg.h 文件用来管理 app 的版本，对 App 进行版本管理是非常有必要的，尤其是在 OTA 时可以有效的防止错误升级而导致变砖的风险。

App 版本是由三个关键字段组成的，分别是制造商代码（Manufacturer Code）、固件类型（Image Type）和文件版本（FileVersion），会以小端形式存放在固件的固定位置，如下图。（字段定义可以参考<<Zigbee Cluster Library Specification>>）。

图 2-23 二进制文件



### 2.3.1 制造商代码

MANUFACTURER\_CODE：制造商代码是 Zigbee 联盟为每个成员公司分配的一个 2 字节长度的标识符，比如 0x1141 是 Telink 的制造商代码。用户可以将其修改成自己的制造商代码。

### 2.3.2 固件类型

IMAGE\_TYPE：固件类型是一个 2 字节长度的常量，高字节代表 Chip 类型，低字节代表 Image 类型。Chip 类型和 Image 类型定义在 version\_comm.h 文件中，用户可以根据需求自行修改或添加。

### 2.3.3 文件版本

FILE\_VERSION：文件版本是反应固件发行和编译的版本号，是一个 4 字节长度的常量。文件版本必须以递增形式进行管理，例如当前的文件版本号必须大于之前发行的版本号，因为当 OTA 时，只有检查到新的版本号大于之前的版本号时才会进行升级。

## 2.4 运行模式

Telink Zigbee SDK 提供了两种运行模式：多地址启动模式（从 0x0 或 0x40000 地址启动）和 BootLoader 启动模式（从 BootLoader 启动后跳转到 App）。

用户可以通过修改 comm\_cfg.h 文件中的宏定义 BOOT\_LOADER\_MODE 来更改运行模式，SDK 默认使用多地址启动模式。

```
#define BOOT_LOADER_MODE 0//1
```

需要注意的是，在不同的运行模式和不同 Flash 大小容量的情况下，SDK 将会使用不同的 Flash 布局。

### 2.4.1 两种模式优缺点比较

#### 多地址启动模式

优点：启动速度快；OTA 结束后无需再次搬运 image，校验正确后可以快速启动。

缺点：image 只能位于地址 0x0 或 0x40000，会造成 flash 空间分配的不连续性；另外 image 的大小(如果支持 OTA 的话)只能小于 208KB。

#### Bootloader 启动模式

优点：通常 bootloader 位于地址 0x0，image 文件可以灵活选择地址，便于用户自行定义地址空间。

缺点：启动速度较地址启动模式慢；会额外增加 flash 消耗用以放置 bootloader 代码；另外在 OTA 时，需要增加搬运 image 的工作。

## 2.4.2 多地址启动模式 Flash 分布

图 2-24 512k Flash 空间分配图

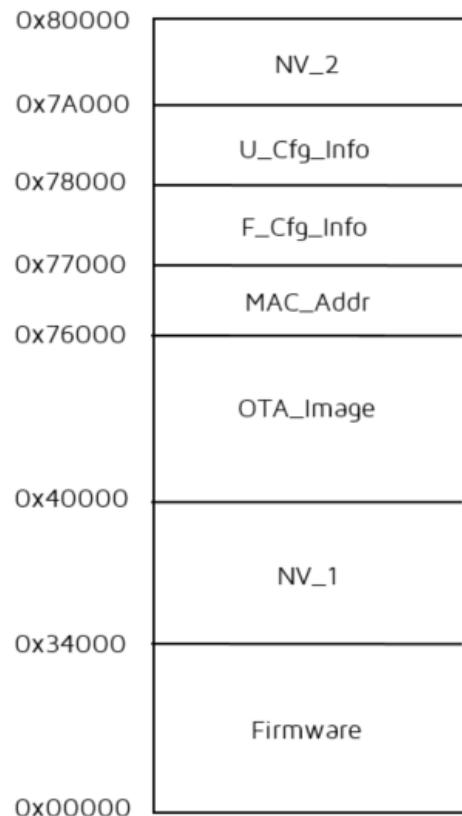
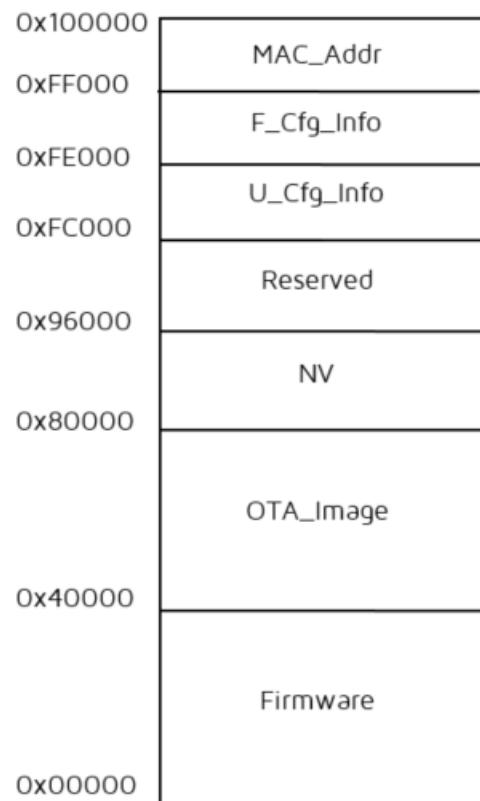


图 2-25 1M Flash 空间分配图



### 2.4.3 Bootloader 启动模式 Flash 分布

图 2-26 512k Flash 空间分配图

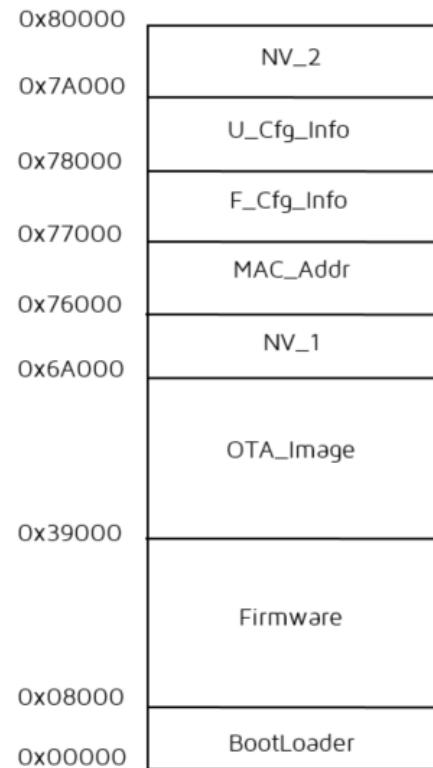
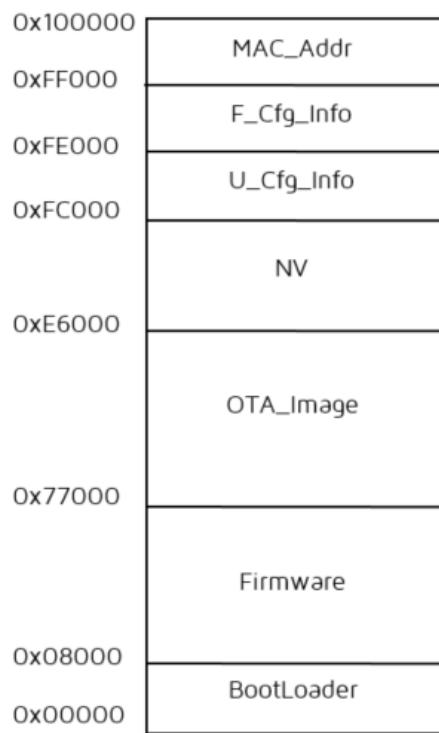


图 2-27 1M Flash 空间分配图



## 2.5 Flash 分布说明

### 1) MAC\_Addr:

MAC 地址信息存放在 Flash 的 0x76000 或 0xFF000 地址开始的 8 个 bytes 位置。系统启动后会检查 MAC 地址信息，如果发现是 0xFFFFFFFFFFFFFF，将会随机产生一个 MAC 地址。芯片出厂时会预先写入 MAC 地址，请谨慎擦除。

### 2) F\_Cfg\_Info:

出厂配置参数信息。芯片在出厂时会预先写入一些校准参数信息，比如射频频偏校准、ADC 校准等，请谨慎擦除。

### 3) U\_Cfg\_Info:

用户配置参数信息。例如 install code 和 factory reset 标志会存储在该区域。

4) NV(NV\_1, NV\_2):

节点入网后会将网络信息保存在 NV 区。512k Flash 网络信息分两部分保存，分别保存在 NV\_1 的 48kB 空间和 NV\_2 的 24kB 空间。1M Flash 网络信息保存在 NV 的 88kB 空间。

所以，如果只是更新固件或断电重启的话，网络信息不会丢失。

5) Firmware 和 OTA\_Image:

TLSR8 和 TLSR9 支持 Flash 多地址启动，即可以从 0x0 地址启动，也可以从 0x40000 地址启动。Telink Zigbee SDK 使用该特性，使用乒乓模式，利用 Firmware 和 OTA\_Image 相互切换来实现 OTA 功能。

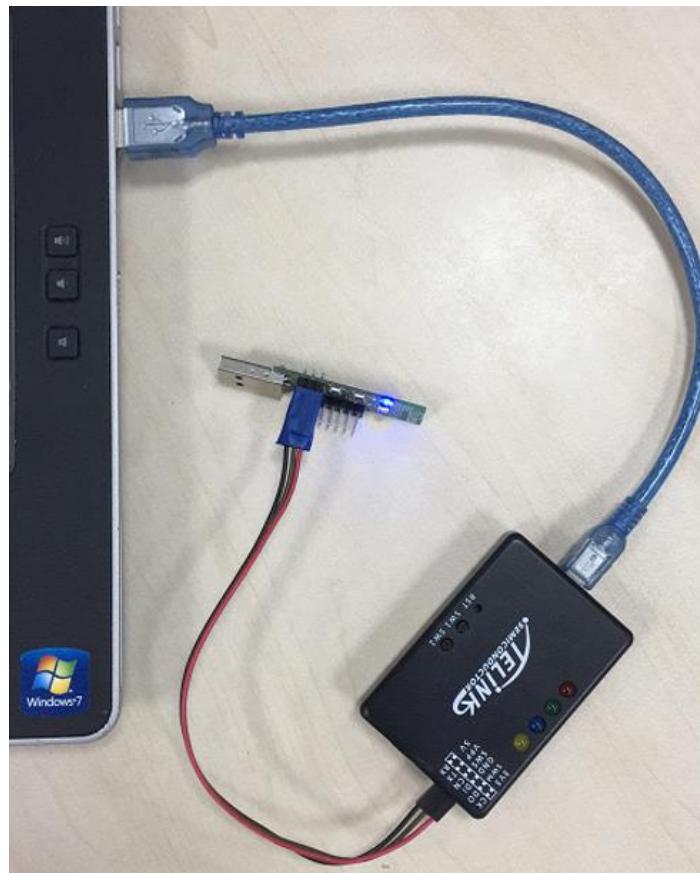
6) BootLoader:

引导程序，在 BootLoader 运行模式下使用。

## 2.6 固件烧录

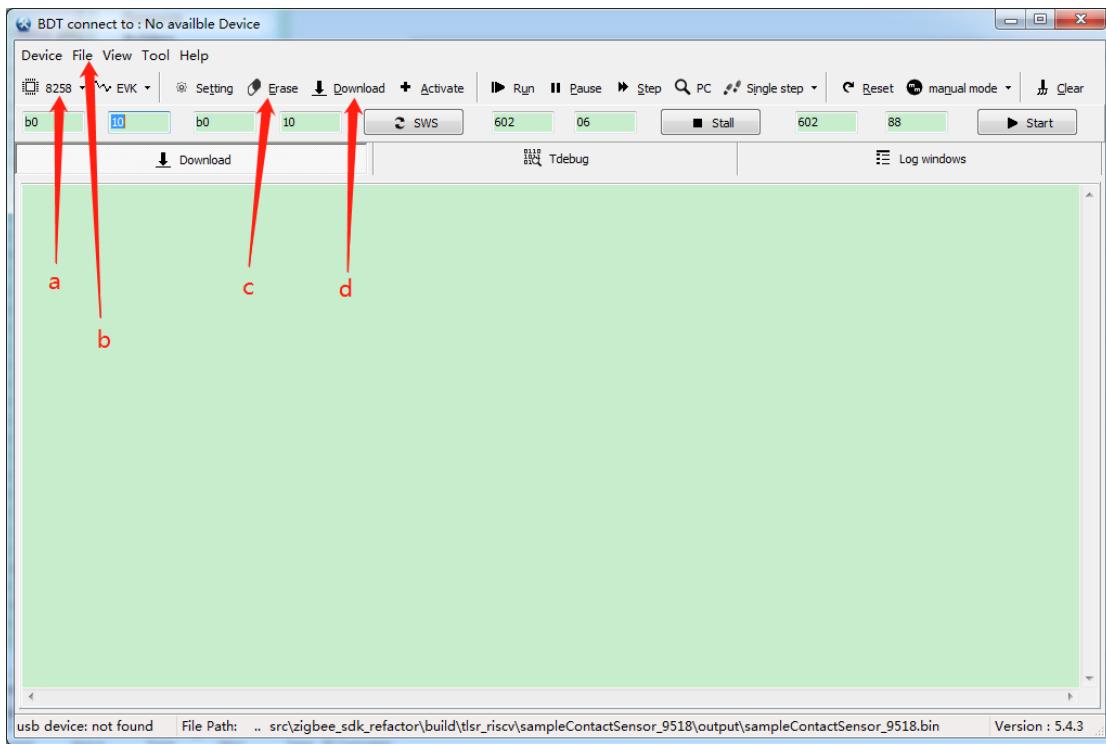
1) 通过 mini USB 线将 Telink 烧录 EVK 与 PC 相连，EVK 板指示灯会闪烁一次，表示 EVK 与 PC 连接正常；再使用三根杜邦线将 EVK 的 VCC、GND 和 SWM 分别与待烧录目标板的 VCC、GND 和 SWS 相连。

图 2-28 烧录连接



- 2) 硬件连接好之后，打开 Telink BDT.exe 烧录工具，准备烧录固件。
  - a) 选择“8258”芯片型号（本文档以 8258 为例）；
  - b) 选择“File”->“Open”，选择要烧录的固件；
  - c) 点击“Erase”擦除 512K Flash；
  - d) 点击“Download”下载固件；
  - e) 若使用 bootLoader 模式，请选择“setting”修改“Download Addr”，将 bootLoader 和 App 固件分别烧写到对应地址（参考 2.4.3）。  
(烧录工具详细的使用方法请参考烧录工具使用文档。)

图 2-29 烧录工具



# 3. 软件架构

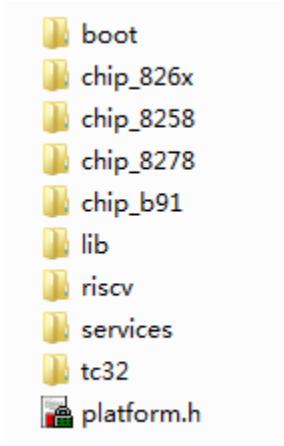
Telink Zigbee SDK 的目录结构已在 2.1.2 和 2.2.2 节中列出，本章节会对各个目录下的文件具体作下说明。

## 3.1 目录说明

### 3.1.1 硬件平台目录

当前 SDK 支持 b85m (826x、8258、8278) 和 b91m (9518) 多个平台 (\tl\_zigbee\_sdk\platform)，后续可以添加其他硬件平台。

图 3-1 硬件平台目录

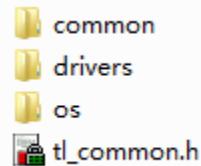


- ✧ \boot: 不同平台的.S 启动代码和.link 链接文件
- ✧ \chip\_826x: 826x 平台硬件模块驱动头文件
- ✧ \chip\_8258: 8258 平台硬件模块驱动头文件
- ✧ \chip\_8278: 8278 平台硬件模块驱动头文件
- ✧ \chip\_b91: b91 平台硬件模块驱动头文件
- ✧ \lib: 不同平台的驱动库文件
- ✧ \services: 不同平台中断处理文件

- ✧ \riscv: 支持 risc-v 平台的相关文件
- ✧ \tc32: 支持 tc32 平台的相关文件

### 3.1.2 通用函数目录

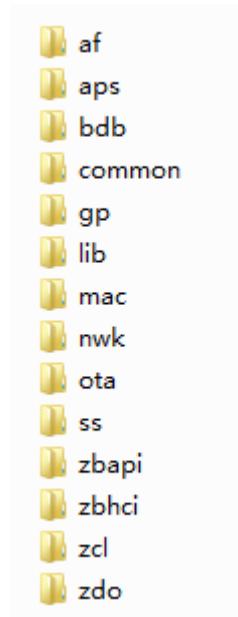
图 3-2 通用函数目录



- ✧ \common: 一些通用的功能函数，如字符串、链表、打印等处理函数文件
- ✧ \drivers: 抽象层驱动文件
- ✧ \os: 任务事件、buffer 管理函数文件

### 3.1.3 Zigbee 协议栈目录

图 3-3 Zigbee 协议栈目录



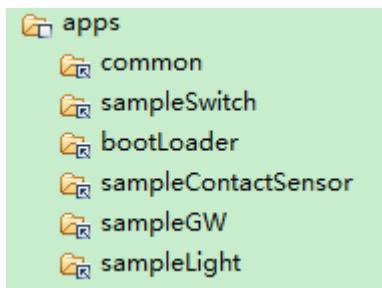
大多以.lib 文件的给出，其中 PHY 层以及与应用层密切相关的 ZCL 和 zbhci 以开源代码方式给出。

- ✧ \af: 应用框架层 (Application Framework) 相关文件
- ✧ \aps: 应用支持子层 (Application Support Sub-Layer) 相关文件
- ✧ \bdb: 基本设备行为 (Base Device Behavior) 规范相关文件
- ✧ \common: zigbee 协议栈配置相关文件
- ✧ \gp: 绿色能源 (Green Power) 规范相关文件
- ✧ \lib: zigbee 协议栈库文件
- ✧ \mac: 媒介控制层 (Media Access Control) 相关文件
- ✧ \nwk: 网络层 (Network) 相关文件
- ✧ \ota: 空中升级 (Over The Air) 相关文件
- ✧ \ss: 安全服务 (Security Services) 相关文件

- ✧ \zbapi: zigbee 常用接口函数文件
- ✧ \zcl: zigbee 簇 (Zigbee Cluster Library) 相关文件
- ✧ \zdo: zigbee 设备对象 (Zigbee Device Objects) 相关文件

### 3.1.4 应用层目录

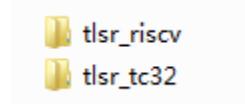
图 3-4 应用层目录



- ✧ \common: 应用层通用文件目录
  - factory\_reset.c/h: 通过断电操作来进行设备复位
  - firmwareEncryptChk.c/h: 软件加密防拷贝，利用 UID 的唯一性，防止固件被复制拷贝
  - module\_test.c: 模块测试文件，仅供开发过程中测试验证使用
  - main.c: 主函数入口
  - comm\_cfg.h: 版本及运行模式配置文件
- ✧ \sampleGW: 网关应用例程
- ✧ \sampleLight: 灯应用例程
- ✧ \sampleSwitch: 开关应用例程
- ✧ \sampleContactSensor: 门磁应用例程

### 3.1.5 工程编译目录

图 3-5 工程编译目录



- ✧ \build\tlsr\_riscv: risc-v 平台导入和编译的工程目录，TLSR9 系列芯片请选择使用该目录
- ✧ \build\tlsr\_tc32: tc32 平台导入和编译的工程目录，TLSR8 系列芯片请选择使用该目录

## 3.2 抽象层驱动

Telink Zigbee sdk 兼容多款 Telink 芯片，为了便于驱动的接口统一，添加了驱动抽象层，位于/proj/drvier/下，各驱动具体使用方法可参见 apps/module\_test.c。

### 3.2.1 平台初始化

- 初始化：

```
drv_platform_init(void)
```

完成芯片、系统 clock 的配置、gpio、射频（RF）、timer 等 Zigbee 应用开发中所需的一些模块的初始化

### 3.2.2 射频（RF）

- 初始化：

```
ZB_RADIO_INIT ()
```

- 模式切换：

```
ZB_RADIO_TRX_SWITCH(mode, chn)
```

mode: RF\_MODE\_TX = 0, 发送模式

RF\_MODE\_RX = 1, 接收模式

RF\_MODE\_AUTO = 2, zigbee 未使用

RF\_MODE\_OFF 关闭射频模块

Chn: 11-26(对应 802.15.4 中的 2.4G 的 16 个 channel 值)

➤ 发射功率:

**ZB\_RADIO\_TX\_POWER\_SET(level)**

设置 RF 模块发送功率, 不同芯片对应的功率值稍有不同, 具体指请参考 RF\_PowerIndexTypeDef 的定义  
(platform/chip/rf\_drv.h)

➤ 数据发送:

**ZB\_RADIO\_TX\_START(txBuf)**

参数 txBuf: 待发送数据的内存地址,

数据格式 dma length (4Bytes: payload 长度+1) + len (1Byte: payload  
长度+2) + payload

➤ 设置射频接收 buffer:

**ZB\_RADIO\_RX\_BUF\_SET(addr)**

在设置射频为 Rx 模式时, 必须确保将 Rx buffer 设置为有效安全的内存地址

➤ RSSI 获取:

**ZB\_RADIO\_RSSI\_GET ()**

调用该函数时, 请确保射频此时处于 Rx 模式

➤ Rssi 转化为 Lqi:

**ZB\_RADIO\_RSSI\_TO\_LQI(mode, rssi, lqi)**

mode: 仅对 8269 有效

```
typedef enum{
    RF_GAIN_MODE_AUTO,
    RF_GAIN_MODE_MANU_MAX,
}rf_rxGainMode_t;
```

➤ 接收中断:

**rf\_rx\_irq\_handler(void)**

➤ 发送中断:

**rf\_tx\_irq\_handler(void)**

- 注意： 射频中断函数 rf\_rx\_irq\_handler/rf\_tx\_irq\_handler 仅能被 zigbee stack 中使用，如果用户自行调用 ZB\_RADIO\_TX\_START 所产生的中断，需用户自行注册新的中断回调函数，以避免影响 Zigbee stack 的运行状态。

### 3.2.3 GPIO

- 初始化以及配置：

```
gpio_init()
```

此函数是通过将 platform/chip\_xxxx/gpio\_default.h 下的默认配置(应用层的 board\_xxxx.h 可以修改该默认配置)写入 gpio 寄存器，实现 gpio 初始化。

- IO 功能设置：

```
void drv_gpio_func_set(u32 pin)
```

设置 IO 具体功能

参数 pin： 参见 GPIO\_PinTypeDef 的定义

- GPIO 读：

```
void drv_gpio_read(u32 pin);
```

读取 gpio 的高低电平

- GPIO 写：

```
void drv_gpio_write(u32 pin, bool value);
```

设置 gpio 的高低电平

- 输出使能：

```
void drv_gpio_output_en(u32 pin, bool enable);
```

- 输入使能：

```
void drv_gpio_input_en(u32 pin, bool enable)
```

- GPIO 中断操作

芯片最多可以同时支持 3 路外部 GPIO 中断，中断模式分别为：GPIO\_IRQ\_MODE、GPIO\_IRQ\_RISC0\_MODE 和 GPIO\_IRQ\_RISC1\_MODE。

### 1) 注册中断服务函数

```
int drv_gpio_irq_conf(drv_gpio_irq_mode_t mode, u32 pin,
drv_gpioPoll_e polarity, irq_callback gpio_irq_callback);
```

### 2) 使能中断管脚

```
int drv_gpio_irq_en(u32 pin);
int drv_gpio_irq_risc0_en(u32 pin);
int drv_gpio_irq_risc1_en(u32 pin);
```

## 3.2.4 UART

### ➤ 管脚设置：

```
void drv_uart_pin_set(u32 txPin, u32 rxPin)
```

使用 uart 之前必须选择相应的 IO 作为 uart 的收发管脚

### ➤ 初始化：

```
void drv_uart_init(u32 baudrate, u8 *rxBuf, u16 rxBufLen, uart_irq_callback uart_recvCb)
```

Baudrate: 波特率

rxBuf: 接收 buffer

rxBufLen: 接收数据最大长度

uart\_recvCb: 接收中断时供应用层的回调函数

### ➤ 发送：

```
u8 drv_uart_tx_start(u8 *data, u32 len)
```

### ➤ 接收中断函数：

```
void drv_uart_rx_irq_handler(void)
```

### ➤ 发送中断函数：

```
void drv_uart_tx_irq_handler(void)
```

- 异常处理函数：

```
void drv_uart_exceptionProcess(void)
```

注意：当使用 uart 时，为避免通讯异常，main\_loop 必须轮询该异常处理函数。

### 3.2.5 ADC

- 初始化：

```
bool drv_adc_init(void);
```

- 配置：

```
void drv_adc_mode_pin_set(drv_adc_mode_t mode, GPIO_PinTypeDef pin)
```

参数 mode：参见 drv\_adc\_mode\_t 定义

参数 pin：所使用的管脚号，参见 GPIO\_PinTypeDef 定义

- 获取采样值：

```
u16 drv_get_adc_data(void)
```

### 3.2.6 PWM

- 初始化：

```
void drv_pwm_init(void)
```

- 配置：

```
void drv_pwm_cfg(u8 pwmid, u16 cmp_tick, u16 cycle_tick)
```

参数 pwmid：pwm 通道

参数 cmp\_tick：PWM 一个周期中处于高电平的 tick 数

参数 cycle\_tick：PWM 一个周期包含的 tick 数

### 3.2.7 TIMER

➤ 初始化:

```
void drv_hwTmr_init(u8 tmrIdx, u8 mode)
```

参数 tmrIdx:    TIMER\_IDX\_0, TIMER\_IDX\_1, TIMER\_IDX\_2, TIMER\_IDX\_3  
 参数 mode:     TIMER\_MODE\_SYSCLK, TIMER\_MODE\_GPIO\_TRIGGER,  
 TIMER\_MODE\_GPIO\_WIDTH, TIMER\_MODE\_TICK

➤ 设置:

```
void drv_hwTmr_set(u8 tmrIdx, u32 t_us, timerCb_t func, void *arg)
```

参数 tmrIdx:    需要使用的 timer  
 参数 t\_us:      定时间隔, 单位: us  
 参数 func:      该定时中断应用层回调函数  
 参数 arg:       应用层回调函数所需参数

➤ 注销:

```
void drv_hwTmr_cancel(u8 tmrIdx)
```

➤ 中断函数:

```
void drv_timer_irq0_handler(void)
void drv_timer_irq1_handler(void)
void drv_timer_irq3_handler(void)
```

其中 TIMER\_IDX\_2 默认作为 watchdog 使用, 建议用户不要使用

TIMER\_IDX\_3 用作 MAC-CSMA, 建议用户不要使用

### 3.2.8 Watchdog

➤ 初始化

```
void drv_wd_setInterval(u32 ms)
```

参数 ms: 超时时间

➤ 启动

```
void drv_wd_start(void)
```

➤ 喂狗

```
void drv_wd_clear(void)
```

### 3.2.9 System Tick

System Tick 计数器，它是一个 32bit 长度、每一个时钟周期自动加一的可读计数器。

由于 826x 和其他系列 IC 的 System Timer 的时钟源不同，8269 的 System Timer 是 32M，8258、8278 和 9518 的 System Timer 是 16M，所以在最大计数时间上存在区别。

- 8269 最大定时时间： $(1/32)\mu\text{s} \times (2^{32})$  约等于 134 秒，每过 134 秒 System Timer 转一圈。
- 8258、8278、9518 最大定时时间： $(1/16)\mu\text{s} \times (2^{32})$  约等于 268 秒，每过 268 秒 System Timer Ticker 转一圈。

用户可以使用 `clock_time()` 接口获取当前 tick。

### 3.2.10 电压检测

为了防止低压系统运行异常，SDK 提供了基于 ADC 驱动实现的电压检测函数，需要注意的是：

- 1、需要使用支持 ADC 功能的 I/O 口，且用于 ADC 检测的 I/O 口不可以做其他功能
- 2、使用 `DRV_ADC_VBAT_MODE` 模式时，I/O 口需要悬空
- 3、使用 `DRV_ADC_BASE_MODE` 模式时，I/O 口需要连接电压测试点
- 4、B91 只能使用 `DRV_ADC_BASE_MODE` 模式，且 I/O 口需要做外部分压处理

➤ 初始化

```
void voltage_detect_init(void)
```

➤ 电压检测

```
voltage_detect(void)
```

### 3.2.11 睡眠和唤醒

Telink Zigbee SDK 提供了相关的低功耗管理函数。sampleSwitch\_826x 使用 suspend 模式；sampleSwitch\_8258、sampleSwitch\_8278 和 sampleSwitch\_B91 使用 deep with retention 模式，即休眠时 RAM 数据可以保持（8258 和 8278 支持 32k RAM 保持，B91 支持 64k RAM 保持，详细请查阅数据手册）。

➤ 唤醒源类型

支持按键以及 timer 定时唤醒

➤ 配置唤醒引脚

如果需要按键唤醒功能，首先需要配置对应的唤醒管脚以及唤醒电平。

```
/**  
 * @brief Definition for wakeup source and level for PM  
 */  
drv_pm_pinCfg_t g_switchPmCfg[] =  
{  
{BUTTON1,    PM_WAKEUP_LEVEL},  
{BUTTON2,    PM_WAKEUP_LEVEL},  
};  
drv_pm_wakeupPinConfig(g_switchPmCfg, sizeof(g_switchPmCfg)/sizeof(drv_pm_pinCfg_t));
```

➤ 休眠函数

```
drv_pm_lowPowerEnter(void);
```

如果使能了按键唤醒，调用此函数时，若相应管脚当前的电平与唤醒电平一致，则系统不能有效进入低功耗状态。

➤ 休眠函数说明：

- 1) 调用 tl\_stackBusy() 和 zb\_isTaskDone() 检查是否符合休眠条件；

- 2) 遍历软件定时任务列表，检查是否存在定时任务，如果有执行 3），如果没有执行 4）；
- 3) 检索出临近任务的时间，以该时间作为休眠时间，进入 Suspend 或 Deep Retention 休眠模式，可以定时器唤醒和按键唤醒；
- 4) 进入 Deep 休眠模式，可以按键唤醒。

## 3.3 存储管理

### 3.3.1 动态内存管理

Telink Zigbee SDK 提供了动态内存分配和释放的接口，默认最大支持 142 字节长度的内存申请，用户可以在开发中直接使用，接口如下。

- 1) 申请内存：

```
u8 *ev_buf_allocate(u16 size);
```

- 2) 释放内存：

```
buf_sts_t ev_buf_free(u8 *pBuf);
```

- 3) 资源配置

默认由 4 组不同个数、不同大小的内存池构成，用户可以根据产品需求以及当前内存使用情况自行修改其个数以及大小。

```
MEMPOOL_DECLARE(size_x_pool, size_x_mem, BUFFER_GROUP_x, BUFFER_NUM_IN_GROUPx);
```

### 3.3.2 NV 管理

因为 Zigbee 需要保存各层的网络信息参数，在异常断电后能够恢复网络，所以 SDK 从 FLASH 中划分了一块区域专门用来存储此类信息，这块 FLASH 区域我们称它为 NV 区，即第 2.4.3 章节的 NV(NV\_1, NV\_2)区。

由于芯片 FLASH 仅支持 sector (1 sector = 4k) 擦除，所以 NV 区的信息存储模块的最小单位为 1 个 sector。

SDK 中使用了以下几个信息模块，其中 NV\_MODULE\_APP 是供用户使用，如需添加新的信息模块，请在末尾添加，不能改变现有的模块序号。

```
typedef enum {
    NV_MODULE_ZB_INFO = 0,
    NV_MODULE_ADDRESS_TABLE = 1,
    NV_MODULE_APS = 2,
    NV_MODULE_ZCL = 3,
    NV_MODULE_NWK_FRAME_COUNT = 4,
    NV_MODULE_OTA = 5,
    NV_MODULE_APP = 6,
    NV_MODULE_KEYPAIR = 7,
    //NV_MODULE_ADD,
    NV_MAX_MODULES
}nv_module_t;
```

信息模块又由多个条目信息组成，例如 NV\_MODULE\_ZCL 由一系列的 NV\_ITEM\_ZCL\_xx 组成。条目定义如下，如需添加新的条目信息，请在末尾添加，不能改变现有的条目序号。

```
typedef enum {
    NV_ITEM_ID_INVALID = 0,/* Item id 0 should not be used. */
    NV_ITEM_ZB_INFO = 1,
    NV_ITEM_ADDRESS_FOR_NEIGHBOR,
    NV_ITEM_ADDRESS_FOR_BIND,
    NV_ITEM_APS_SSIB,
    NV_ITEM_APS_GROUP_TABLE,
    NV_ITEM_APS_BINDING_TABLE,
    NV_ITEM_NWK_FRAME_COUNT,
    NV_ITEM_SS_KEY_PAIR,
    NV_ITEM_OTA_HDR_SERVERINFO,
    NV_ITEM_OTA_CODE,
    NV_ITEM_ZCL_REPORT = 0x20,
    NV_ITEM_ZCL_ON_OFF,
    NV_ITEM_ZCL_LEVEL,
    NV_ITEM_ZCL_COLOR_CTRL,
    NV_ITEM_ZCL_SCENE_TABLE,
    NV_ITEM_ZCL_GP_PROXY_TABLE,
    NV_ITEM_ZCL_GP_SINK_TABLE,
```

```

NV_ITEM_APP_SIMPLE_DESC,
NV_ITEM_APP_POWER_CNT,

NV_ITEM_ID_MAX           = 0xFF,/* Item id 0xFF should not be used. */

}nv_item_t;

```

- 每个 module 占用 2 个或(2\*n)sectors, 格式:

sector info + item 索引 + item 内容

其中 sector info 标识该块是否有效, 长度为 sizeof(nv\_sect\_info\_t), 之后紧跟待写入的 item 索引, item 内容起始于该 module 首地址偏移 512Byte 或 1024Byte 的位置

- 为了避免频繁进行擦除操作, NV 采用单 module 追加写入的方法, 直到 1 个 Sector 写满后, 将有效信息搬到另一个 sector, 再将之前的 sector 内容清除。
- NV 区读写操作接口如下。

```

nv_sts_t nv_flashWriteNew(u8 single, u16 id, u8 itemId, u16 len, u8 *buf);
nv_sts_t nv_flashReadNew(u8 single, u8 id, u8 itemId, u16 len, u8 *buf);

```

- 注意: 对于某个 item, 如果只存在一个有效值的话, 则 single 为 1, 那么当再次写这个 item 时, 会将之前有效的 item 清除; 否则的话, 需要根据内容检索相同的 item, 将其置为无效, 再写入新的 item。

## 3.4 任务管理

### 3.4.1 单次任务队列

- 接口函数:

```
TL_SCHEDULE_TASK(tl_zb_callback_t func, void *arg)
```

参数 func: 任务回调函数

参数 arg: 任务所需参数

- 只执行一次, 没有按优先级, 顺序依次处理

- 建议使用场合：
  - 1) 需要避免函数深度嵌套引起的堆栈溢出问题；
  - 2) 中断函数中，通过将数据、事件压入队列，避免在中断函数里消耗过多时间。
- 大小： 32 个（使用时避免一次性压入过多任务）

### 3.4.2 常驻任务队列

- 任务注册（任务注册后默认启动）：

```
ev_on_poll(ev_poll_e e, ev_poll_callback_t cb)
```

- 任务暂停

```
ev_disable_poll(ev_poll_e e, ev_poll_callback_t cb)
```

- 任务重启

```
ev_enable_poll(ev_poll_e e, ev_poll_callback_t cb)
```

- 此任务注册后，会在主循环中一直执行。

### 3.4.3 软件定时任务

为了方便用户实现对时间精度需求不高的定时任务，Telink Zigbee SDK 提供了一个软件定时任务管理机制。

需要注意的是，从 SDK V3.6.2 版本开始，软件定时任务管理的时间单位从原来的 ticker 改为毫秒，这解决了长时间定时任务需求的问题。

#### 3.4.3.1 接口函数

- 任务注册： TL\_ZB\_TIMER\_SCHEDULE(cb, arg, timeout)

```
/**  
 * @param[in]      func - the callback of the timer event
```

```

/*
 * @param[in]      arg - the parameter to the callback
 *
 * @param          cycle - the timer interval
 *
 * @return         the status
 */

```

- 任务取消: TL\_ZB\_TIMER\_CANCEL(evt)

```

/**
 * @param[in]      evt - the pointer to the timer event pointer
 *
 * @return         the status
 */

```

### 3.4.3.2 注意事项

- 1) 定时任务回调函数返回值的三种用法:

- 返回值小于 0, 则该任务执行后被自动删除, 任务不复存在。
- 返回值等于 0, 则一直使用之前启动时的 t\_ms 来周期定时触发回调函数。
- 返回值大于 0, 则使用该返回值作为新的周期定时触发回调函数, 单位 ms。
- 注意: 在中断函数里避免使用 TL\_ZB\_TIMER\_CANCEL()

### 3.4.3.3 使用实例

当 VK\_SW1 按键按下后, 启动或取消一个 10 秒的定时任务, 时间到达后执行一次广播 On/Off Toggle 的命令, 执行结束后退出。代码如下:

```

ev_timer_event_t *brc_toggleEvt = NULL;

s32 brc_toggleCb(void *arg)
{
    epInfo_t dstEpInfo;
    TL_SETSTRUCTCONTENT(dstEpInfo, 0);

```

```
dstEpInfo.dstAddrMode = APS_SHORT_DSTADDR_WITHEP;
dstEpInfo.dstEp = SAMPLE_GW_ENDPOINT;
dstEpInfo.dstAddr.shortAddr = 0xffff;
dstEpInfo.profileId = HA_PROFILE_ID;
dstEpInfo.txOptions = 0;
dstEpInfo.radius = 0;

zcl_onOff_toggleCmd(SAMPLE_GW_ENDPOINT, &dstEpInfo, FALSE);

brc_toggleEvt = NULL;
return -1;
}

void buttonShortPressed(u8 btNum)
{
    if(btNum == VK_SW1){
        if(!brc_toggleEvt){
            brc_toggleEvt = TL_ZB_TIMER_SCHEDULE(brc_toggleCb,
NULL,
10 * 1000);
        }else{
            TL_ZB_TIMER_CANCEL(&brc_toggleEvt);
        }
    }
}
```

# 4. Zigbee SDK 应用开发

## 4.1 运行模式选择

用户可以通过修改 comm\_cfg.h 文件中的宏定义 BOOT\_LOADER\_MODE 来更改运行模式，2 种模式区别详见 2.4 章节。

```
#define BOOT_LOADER_MODE      0//1
```

- sdk 默认采用多地址启动模式 (BOOT\_LOADER\_MODE 为 0)
- 如果采用 bootloader 方式，请将 BOOT\_LOADER\_MODE 设置为 1,
  - 1) BOOT\_LOADER\_IMAGE\_ADDR: bootloader image 放置的位置，通常为 0
  - 2) APP\_IMAGE\_ADDR: 用户 image 放置的位置，可根据实际应用自行修改，建议使用 0x8000

## 4.2 硬件选择

Telink Zigbee SDK demo 可以运行在不同系列芯片、不同硬件板子上，用户需要针对不同的硬件条件，做相应的配置

### 4.2.1 芯片型号确认

Telink Zigbee SDK 在 comm\_cfg.h 中列举了以下几种芯片类型，用户在编译工程实例前，请确认相应工程下 version\_cfg.h 中选择的芯片类型是否与实际使用的芯片类型一致。

#### 4.2.1.1 comm\_cfg.h 芯片类型定义

```
/* Chip IDs */
#define TLSR_8267      0x00
#define TLSR_8269      0x01
#define TLSR_8258_512K 0x02
```

```
#define TLSR_8258_1M      0x03
#define TLSR_8278          0x04
#define TLSR_9518          0x05
```

#### 4.2.1.2 version\_cfg.h 芯片类型选择

```
#if defined(MCU_CORE_826x)
    #if (CHIP_8269)
        #define CHIP_TYPE           TLSR_8269
    #else
        #define CHIP_TYPE           TLSR_8267
    #endif
# elif defined(MCU_CORE_8258)
    #define CHIP_TYPE           TLSR_8258_512K//TLSR_8258_1M
# elif defined(MCU_CORE_8278)
    #define CHIP_TYPE           TLSR_8278
# elif defined(MCU_CORE_B91)
    #define CHIP_TYPE           TLSR_9518
#endif
```

#### 4.2.2 目标板选择

Telink Zigbee SDK 提供的 Demo 可以运行在 EVK 板或 USB Dongle 上的，用户可以在相应 Demo 下的 app\_cfg.h 文件中选择更改目标板。

```
/* Board ID */
#define BOARD_826x_EVK          0
#define BOARD_826x_DONGLE        1
#define BOARD_826x_DONGLE_PA     2
#define BOARD_8258_EVK           3
#define BOARD_8258_EVK_V1P2       4//C1T139A30_V1.2
#define BOARD_8258_DONGLE         5
#define BOARD_8278_EVK            6
#define BOARD_8278_DONGLE         7
#define BOARD_9518_EVK             8
#define BOARD_9518_DONGLE          9
```

```

/* Board define */

#if defined(MCU_CORE_8258)
#if (CHIP_TYPE == TLSR_8258_1M)
    #define FLASH_CAP_SIZE_1M           1
#endif
#define BOARD                      BOARD_8258_DONGLE
#define CLOCK_SYS_CLOCK_HZ        48000000
#elif defined(MCU_CORE_8278)
    #define FLASH_CAP_SIZE_1M           1
#define BOARD                      BOARD_8278_DONGLE
#define CLOCK_SYS_CLOCK_HZ        48000000
#elif defined(MCU_CORE_B91)
    #define FLASH_CAP_SIZE_1M           1
#define BOARD                      BOARD_9518_DONGLE
#define CLOCK_SYS_CLOCK_HZ        48000000
#else
    #error "MCU is undefined!"
#endif

```

在预编译阶段会根据先前的芯片类型选择加载对应的板级配置，配置文件是各工程目录下的 `board_xx.h`，同时需要注意一下目标板的 `FLASH` 容量是否与板级配置一致，以防加载数据出错。

```
#define FLASH_CAP_SIZE_1M 1
```

## 4.3 打印调试配置

### 4.3.1 UART 打印

为避免硬件 `UART` 资源的浪费，我们通过 `GPIO` 模拟 `UART TX` 实现了 `UART` 打印功能（`printf()` 函数），用户可以任意更改合适的 `GPIO`。配置方法如下：

#### 1) 打印使能配置：

```
#define UART_PRINTF_MODE      1
```

#### 2) 打印口配置：

```
#define DEBUG_INFO_TX_PIN     GPIO_PC4//print
```

### 3) 波特率配置：

```
#define BAUDRATE 1000000//1M
```

注意：

- 1、826x 最大支持 2M 波特率，8258、8278 和 9518 最大支持 1M 波特率；
- 2、不要在中断处理函数中使用该打印，防止打印数据较多或函数嵌套过深导致中断栈溢出。

## 4.3.2 USB 打印

可以配合 Telink BDT 工具使用 USB 打印。配置方法如下：

### 1) 打印使能配置：

```
#define USB_PRINTF_MODE 1
```

### 2) 使能 USB 端口：

```
#define HW_USB_CFG() do{ \  
    usb_set_pin_en(); \  
}while(0)
```

注意：

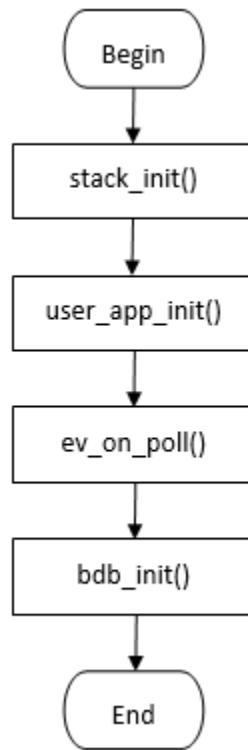
当使用 ZBHCI\_USB\_CDC 或 ZBHCI\_USB\_HID 功能时，USB 打印功能失效。

## 4.4 Zigbee 网络开发流程

### 4.4.1 应用层初始化 (user\_init)

应用层初始化主要由 zigbee stack 以及 zigbee 应用层初始化组成。

图 4-1 应用层初始化流程



### 1) stack\_init()

完成基本的协议栈初始化工作以及协议栈给出的应用层回调函数注册。

- `zb_init()`: 初始化 zigbee 协议栈，包括 `mac`、`nwk`、`aps`、`zdo` 等各层的初始化，对于 *not factory-new* 设备，会从 NV 中读取入网信息，恢复网络信息以及各层属性。

注意： 对于已经入网的设备，无法通过修改 `zb_config.c` 里的默认属性表配置达到改变属性的目的，如果想修改属性的话，必须通过 `g_zbMacPib/g_zbNIB/g_touchlinkAttr/g_bdbAttrs` 直接操作对应的 attribute.

- `zb_zdoCbRegister()`: 注册一些 zigbee stack 给出的回调函数，从而用以获取网络运行过程中的相关信息。比如：建网/入网结果、设备离网、设备加入网络、网络状态改变、父节点收到同步信息等。

## 2) user\_app\_init()

用户根据具体产品功能注册所要的端口以及 clusters 和 attributes。

- 端口注册：

```
af_endpoitRegister()
```

注册一个端口，用以明确该端口的具体功能 (profileId、deviceId、端口号、以及支持的输入 clusters(server)、输出 clusters (client))

- cluster 注册：

```
zcl_register()
```

注册某个端口需要支持的 Clusters、Attributes 相应的注册函数以及命令回调处理函数.

- cluster 初始化：

```
zcl_init()
```

用以注册 ZCL 层支持的基础命令处理回调函数，比如  
ZCL\_CMD\_WRITE/ZCL\_CMD\_WRITE\_RSP/ZCL\_CMD\_READ/ZCL\_CMD\_READ\_RSP 等

## 3) ev\_on\_poll()

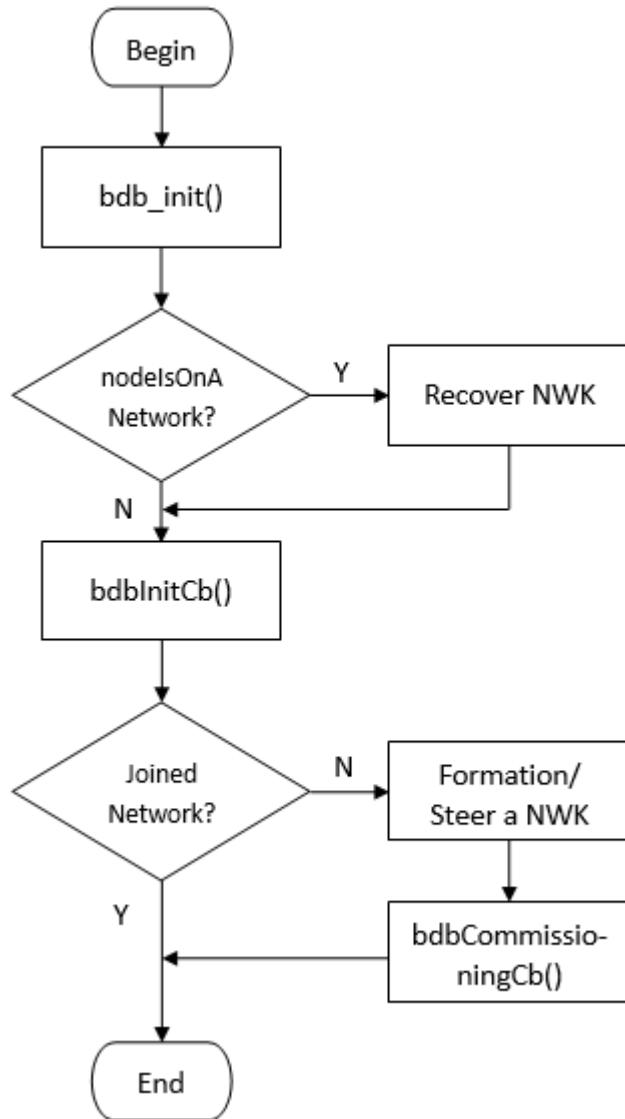
注册用户轮询事件。

### 4.4.2 BDB 流程

完成节点网络初始化、建网、入网的一系列操作

#### 4.4.2.1 BDB 初始化(bdb\_init)

图 4-2 BDB 初始化流程



**bdb\_init()**的功能：完成协议 ZigBee Base Device Behavior Specification 中的第 7 章节的相关功能，对于未入网设备，完成简单初始配置工作；对于已经入网的设备，重新接入到网络。

BDB 初始化后，会调用应用层注册的回调函数 **bdbInitCb**，再由用户调用相关函数决定接下来的 BDB Commissioning 行为。

#### 4.4.2.2 BDB Commissioning

完成协议 ZigBee Base Device Behavior Specification 中的第 8 章节的相关功能，BDB Commissioning 包括以下几种行为：

- Network steering：搜索发现并加入到网络。
- Network formation：新设备创建网络。Router 创建 Distribute 网络，Coordinator 创建 Central 网络。
- Finding & Binding：加入网络后搜索匹配的功能并绑定数据发送的方式。
- Touch Link：通过 touch link 方式接入网络。

在执行完 BDB Commissioning 之后，系统会回调 bdbCommissioningCb 函数，将 commissioning 的结果返回给用户，由用户决定下一步的动作。

例如：一个 End Device 在执行完 BDB Commissioning 后返回了一个 BDB\_COMMISSION\_STA\_NO\_NETWORK 的状态，那么用户可以决定是进入休眠或者继续尝试搜索加入网络。

#### 4.4.3 网络安全

当新设备首次入网时，会从信任中心（Trust Center，一般是指协调器网关）申请获取网络密钥，之后以该密钥加密空中数据进行消息传递。

如何保证网络密钥的安全性？通常使用的方法有：

- 1) 使用 Default TCLK 加密网络密钥，
- 2) 使用 Install code 衍生出来的 Link key 加密网络密钥，
- 3) 预设网络密钥。

#### 4.4.3.1 入网流程

新设备既可以接入信任中心入网（图 4-3 直接入网），也可以通过其他路由节点入网（图 4-4 间接入网），在入网过程中，信任中心会将加密过的网络密钥使用 APS Transport Key 命令下发给新设备，当新设备解析成功后便会广播一个 Device Announce 命令，将自己的地址信息通知到周围的设备。如果新设备是一个 zigbee 3.0 之前协议的设备，这时候就表示入网成功了。反之，如果新设备是一个符合 zigbee 3.0 协议的设备，它还会向信任中心发起 Node Descriptor Request 命令，查询信任中心是否支持 zigbee 3.0，如果支持，将会继续向信任中心申请新的 Trust Center Link Key。

图 4-3 直接入网

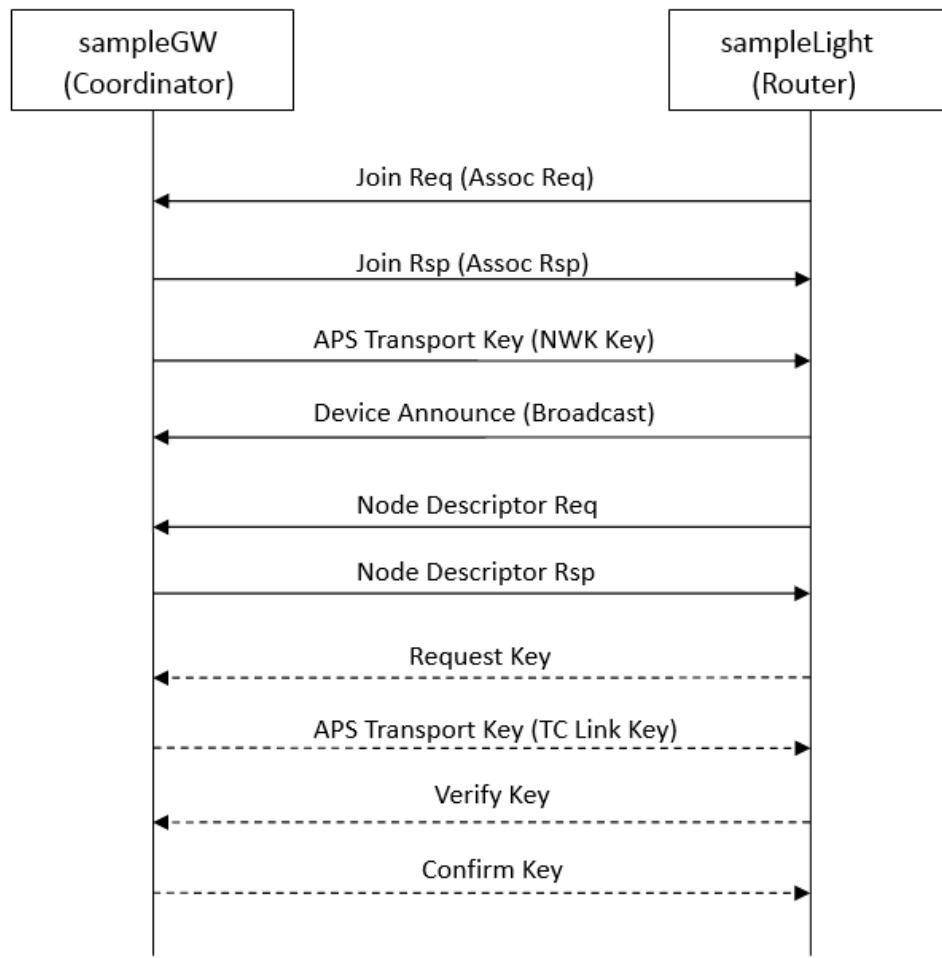
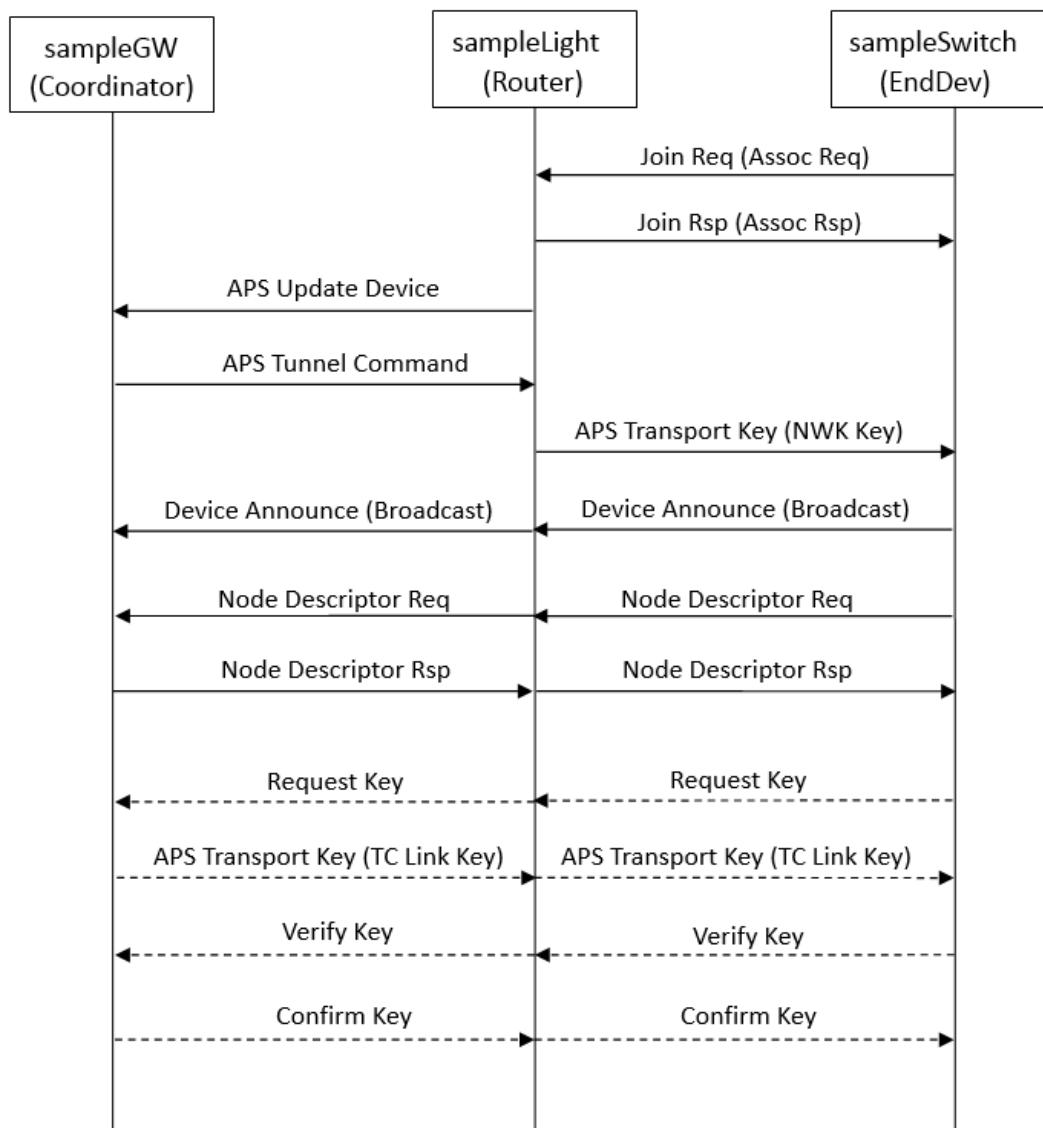


图 4-4 间接入网



#### 4.4.3.2 Default TCLK

Default TCLK，是联盟规定的一组公开的 Trust Center Link Key，它既保证了 zigbee 网络的安全性又保证了互联互通性。Telink Zigbee SDK 默认使用 Default TCLK 来加密网络密钥，但因为这是个公开的 link key，对于私有网络会存在安全隐患。

```

const u8 tcLinkKeyCentralDefault[] = {
    0x5a, 0x69, 0x67, 0x42, 0x65, 0x65, 0x41, 0x6c,
    0x6c, 0x69, 0x61, 0x6e, 0x63, 0x65, 0x30, 0x39
};
```

### 4.4.3.3 Install Code

Install Code 是联盟定义的一种通过 MMO 哈希算法来衍生 TCLK 的加密方法（请参考<<zigbee base device behavior specification>>），只有当信任中心和待入网的设备持有相同的 Install code，才能成功入网。

若使用 install code，通常情况下，每个节点都有一个唯一的 install code 存放在 Flash 的固定位置（见 2.4.3 章节，生产环节时预写进去）。当设备需要接入网关时，将此设备的 install code 和 IEEE 地址导入网关。

网关导入目标设备 install code 的接口如下：

```
void zb_pre_install_code_store(addrExt_t ieeeAddr, u8 *pInstallCode)
```

待入网设备从 Flash 加载使用 install code 的接口如下：

```
void zb_pre_install_code_load(app_linkkey_info_t *appLinkKey)
```

### 4.4.3.4 Pre-configured NWK Key

Pre-configured NWK Key 是提前将网络密钥配置到网关和待入网设备，在入网过程中网关给入网设备下发空的无效的网络密钥，入网节点忽略 transport key 命令中的 nwk key 字段。

接口如下，该函数需要在 bdb\_init()之后调用。

```
void zb_preConfigNwkKey(u8 *nwkKey, bool enTransKey)
```

## 4.4.4 数据交互

网络建立成功之后，同一网络的各个节点就可以相互收发数据了。

用户可以使用标准的 ZCL 来实现数据交互，也可以直接使用相关的 API 来自定义收发数据的处理。

### 4.4.4.1 使用 ZCL 层实现数据交互

Zigbee 联盟规范了 ZCL 标准的 Read/Write/Report/Configure Report 等基础命令，同时针对不同应用规范了一系列的 Cluster、Attribute 和 Command 的处理。

Telink Zigbee SDK 给出了 ZCL 的大部分实现方法，以源码方式提供给用户直接调用，也可以自行添加扩展。SDK 提供的几个 Demo 是使用的 ZCL 方法进行数据交互，具体请参考《ZigBee Cluster Library Specification》。

以 sampleLight 为例，实现方法：

- 1) 注册 AF 层端口描述和 ZCL 接收处理回调函数 zcl\_rx\_handler 以及发送确认回调函数 af\_dataSentConfirm。

```
/* Register endPoint */
af_endpointRegister(SAMPLE_LIGHT_ENDPOINT, (af_simple_descriptor_t*)&sampleLight_simpleDesc, zcl_rx_handler,
af_dataSentConfirm);
```

- 2) 注册 ZCL 基本命令处理回调函数 zclProcessIncomingMsg。

```
/* Register Incoming ZCL Foundation command/response messages */
zcl_init(sampleLight_zclProcessIncomingMsg);
```

- 3) 注册用户需要支持的 Cluster 相应的命令处理回调函数表，即 4.4.1 节中提到的注册表。

```
/* Register ZCL specific cluster information */
zcl_register (SAMPLE_LIGHT_ENDPOINT, SAMPLELIGHT_CB_CLUSTER_NUM,
g_sampleLightClusterList);
```

#### 4.4.4.2 使用 AF 层实现数据交互

除了直接使用 ZCL 规范的数据收发外，用户可以直接注册自定义的接收处理函数，也可以直接使用 AF 层提供的 af\_dataSend 函数往其他设备发送数据。实际上 ZCL 规范就是对 AF 层数据处理的二次封装。

如果不使用 sdk 里的 zcl，则不需要调用 zcl\_register() 以及 zcl\_init()

实现方法：

- 1) 注册数据接收处理回调函数 user\_rx\_handler 以及发送确认回调函数 af\_dataSentConfirm。

```
af_endpointRegister(SAMPLE_LIGHT_ENDPOINT, (af_simple_descriptor_t*)&sampleLight_simpleDesc, user_rx_handler,
NULL);
```

- 2) 调用 af\_dataSend 发送数据。

```
u8 af_dataSend( u8 srcEp, epInfo_t *pDstEpInfo, u16 clusterId,
u8 cmdPldLen, u8 *cmdPld, u8 *seqNo );
```

#### 4.4.5 网络参数配置

用户可以根据需求在/zigbee/common/zb\_config.c 中修改网络参数配置。

➤ TL\_ZB\_NWK\_ADDR\_MAP\_NUM

节点支持的最大地址映射表的数量，关系到网络最大容量。按照 SDK 默认配置，512k FLASH 最大支持 127，1M FLASH 最大支持 300。

➤ TL\_ZB\_NEIGHBOR\_TABLE\_NUM

节点支持的邻居表的个数。路由设备默认 26 个。

➤ ROUTING\_TABLE\_NUM

节点支持的 AODV 路由表个数。

➤ NWK\_ROUTE\_RECORD\_TABLE\_NUM

节点支持的 MTO 路由记录表个数。

➤ APS\_BINDING\_TABLE\_NUM

节点支持的绑定表个数。

➤ APS\_GROUP\_TABLE\_NUM

节点支持的分组个数。

➤ nwkKeyDefalut

网络密钥缺省值。当缺省值为 0 时，协调器将随机生成网络密钥分配给入网的设备；当缺省值为非 0 时，协调器将使用 nwkKeyDefault 作为网络密钥。

➤ macPibDefault

MAC 层的基本参数信息。

➤ nwkNibDefault

NWK 层的基本参数信息。

#### 4.4.6 系统异常处理

调用 `sys_exceptHandlerRegister()` 注册异常回调函数，当 stack 发生 buffer 泄露、状态异常时会触发该函数，在此函数里用户可以添加相应异常处理。

建议：开发阶段，回调函数直接 `while(1)`，以便定位问题，通过变量 `T_evtExcept[1]` 可以定位到发生了何种异常；产品阶段，最好做复位处理

例如：

```
Volatile u16 T_debug_except_code = 0;

static void sampleLightSysException(void){
    T_debug_except_code = T_evtExcept[1];
    while(1);      //or SYSTEM_RESET();
}

/* Register except handler for test */
sys_exceptHandlerRegister(sampleLightSysException);
```

## 5. 常用 APIs

### 5.1 应用框架 AF (Application Framework)

#### 5.1.1 af\_endpointRegister()

在应用框架层注册端点描述符和该端点的接收、发送结果回调处理函数。

原型

```
bool af_endpointRegister( u8 ep, af_simple_descriptor_t *simple_desc, af_endpoint_cb_t rx_cb,
                           af_dataCnf_cb_t cnfCb )
```

返回值

TRUE or FALSE.

Name	Type	Description
ep	u8	Endpoint.
simple_desc	af_simple_descriptor_t*	Pointer to simple descriptor.
rx_cb	af_endpoint_cb_t	Callback function to handle the APS data sent to this endpoint.
cnfCb	af_dataCnf_cb_t	Confirm callback function for APS data transmission.

#### 5.1.2 af\_dataSend()

使用 AF 层发送消息。

原型

```
u8 af_dataSend( u8 srcEp, eplInfo_t *pDstEplInfo, u16 clusterId, u16 cmdPlIdLen,
                  u8 *cmdPlId, u8 *apsCnt )
```

返回值

RET\_OK or Others.

Name	Type	Description
srcEp	u8	Source endpoint.
pDstEpiInfo	eplInfo_t*	Pointer to destination information.
clusterId	u16	Cluster identifier.
cmdPldLen	u16	Payload length.
cmdPld	u8*	Pointer to the payload.
apsCnt	u8*	Pointer to the APS Counter.

## 5.2 基础设备行为 BDB (Base Device Behaviour)

### 5.2.1 bdb\_init()

初始化 BDB 和注册回调处理函数。

原型

```
u8 bdb_init(af_simple_descriptor_t *simple_desc, bdb_commissionSetting_t *setting, bdb_appCb_t *cb,
            u8 repower)
```

返回值

TRUE

Name	Type	Description
simple_desc	af_simple_descriptor_t*	Pointer to the simple descriptor.
setting	bdb_commissionSetting_t*	Pointer to the BDB commissioning setting.
cb	bdb_appCb_t*	Callback function for BDB commissioning.

repower	u8	This function is called after power-on or sleep wakeup.
---------	----	---

### 5.2.2 bdb\_networkFormationStart()

启动创建网络，协调器调用的话将会创建一个集中式网络，路由设备调用将会创建一个分布式网络。

原型

```
u8 bdb_networkFormationStart(void)
```

返回值

BDB\_STATE

### 5.2.3 bdb\_networkSteerStart()

启动搜索网络。

原型

```
u8 bdb_networkSteerStart(void)
```

返回值

BDB\_STATE

### 5.2.4 bdb\_networkTouchLinkStart()

启动 Touch Link。

原型

```
u8 bdb_networkTouchLinkStart(u8 role)
```

返回值

BDB\_STATE

Name	Type	Description
role	u8	BDB_COMMISSIONING_ROLE_INITIATOR or BDB_COMMISSIONING_ROLE_TARGET.

### 5.2.5 bdb\_findAndBindStart()

启动发现和绑定。

原型

```
u8 bdb_findAndBindStart(u8 role)
```

返回值

BDB\_STATE

Name	Type	Description
role	u8	BDB_COMMISSIONING_ROLE_INITIATOR or BDB_COMMISSIONING_ROLE_TARGET.

### 5.2.6 bdb\_defaultReportingCfg()

配置默认上报消息，绑定后生效。

原型

```
status_t bdb_defaultReportingCfg(u8 endpoint, u16 profileID, u16 clusterID, u16 attrID,
                                  u16 minReportInt, u16 maxReportInt,
                                  u8 *reportableChange)
```

返回值

## ZCL\_STA

Name	Type	Description
endpoint	u8	Endpoint.
profileID	u16	Profile Identifier.
clusterID	u16	Cluster Identifier.
attrID	u16	Attribute Identifier.
minReportInt	u16	Minimum reporting interval, in seconds.
maxReportInt	u16	Maximum reporting interval, in seconds.
reportableChange	u8*	Reportable change (only applicable to analog data type).

## 5.3 网络管理 Network Management

### 5.3.1 zb\_init()

初始化 zigbee 协议栈各层。

原型

```
void zb_init(void)
```

返回值

None

### 5.3.2 zb\_zdoCbRegister()

注册协议栈回调函数。

原型

---

```
void zb_zdoCbRegister(zdo_appIndCb_t *cb)
```

返回值

None

Name	Type	Description
cb	zdo_appIndCb_t*	Callback function.

### 5.3.3 zb\_setPollRate()

设置 data request 的发送速率，单位：毫秒。仅入网后的终端设备有效。

原型

```
void zb_setPollRate(u32 newRate)
```

返回值

None

Name	Type	Description
newRate	u32	New poll rate, in millisecond.

### 5.3.4 zb\_rejoinReq()

回连请求，通常终端设备丢失父节点后调用。

原型

```
zdo_status_t zb_rejoinReq(rejoinNwk_method_t method, u32 chm)
```

返回值

zdo\_status\_t

Name	Type	Description
method	rejoinNwk_method_t	The method of joining the network.
chm	u32	Channel mask.

### 5.3.5 zb\_factoryReset()

恢复出厂设置。供路由设备和终端设备使用，恢复出厂前会广播离网通知。

原型

```
void zb_factoryReset(void)
```

返回值

None

### 5.3.6 zb\_mgmtPermitJoinReq()

允许入网控制命令。协调器或路由设备可以使用该命令控制允许其他设备入网的时间。

原型

```
zdo_status_t zb_mgmtPermitJoinReq( u16 dstNwkAddr, u8 permitJoinDuration,
                                    u8 tcSignificance, u8 *seqNo,
                                    zdo_callback indCb )
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
permitJoinDuration	u8	Time in seconds during which the device allows to join.
tcSignificance	u8	TC significance.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	NULL.

### 5.3.7 zb\_mgmtLeaveReq()

离网控制命令。

原型

```
zdo_status_t zb_mgmtLeaveReq( u16 dstNwkAddr, zdo_mgmt_leave_req_t *pReq, u8 *seqNo,
                               zdo_callback indCb )
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_leave_req_t*	Address information of the device that needs to be leaved.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Mgmt_Leave_rsp.

### 5.3.8 zb\_mgmtLqiReq()

获取目标设备的关联邻居信息。

原型

```
zdo_status_t zb_mgmtLqiReq( u16 dstNwkAddr, zdo_mgmt_lqi_req_t *pReq,
                            u8 *seqNo, zdo_callback indCb )
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_lqi_req_t*	Address information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Mgmt_Lqi_rsp.

### 5.3.9 zb\_mgmtBindReq()

获取目标设备的绑定表信息。

原型

```
zdo_status_t zb_mgmtBindReq(u16 dstNwkAddr, zdo_mgmt_bind_req_t *pReq,      u8 *seqNo,
                           zdo_callback indCb)
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_bind_req_t*	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Mgmt_Bind_rsp.

### 5.3.10zb\_mgmtNwkUpdateReq()

更新网络参数或请求网络环境信息。

原型

```
zdo_status_t zb_mgmtNwkUpdateReq(u16 dstNwkAddr,
                                  zdo_mgmt_nwk_update_req_t *pReq,
                                  u8 *seqNo)
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_nwk_update_req_t*	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.

### 5.3.11zb\_zdoBindUnbindReq()

发送绑定或解绑命令。

原型

```
zdo_status_t zb_zdoBindUnbindReq(bool isBinding, zdo_bind_req_t *pReq,
                                  u8 *seqNo, zdo_callback indCb)
```

返回值

zdo\_status\_t

Name	Type	Description
isBinding	bool	TRUE - bind, FALSE - unbind.
pReq	zdo_bind_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Bind_rsp/Unbind_rsp.

### 5.3.12 zb\_zdoNwkAddrReq()

请求目标短地址命令。

原型

```
zdo_status_t zb_zdoNwkAddrReq(u16 dstNwkAddr, zdo_nwk_addr_req_t *pReq,
                               u8 *seqNo, zdo_callback indCb)
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_nwk_addr_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.

Name	Type	Description
indCb	zdo_callback	Callback function for NWK_addr_rsp.

### 5.3.13zb\_zdoleeeAddrReq()

请求目标长地址命令。

原型

```
zdo_status_t zb_zdoleeeAddrReq(u16 dstNwkAddr, zdo_ieee_addr_req_t *pReq,
                                u8 *seqNo, zdo_callback indCb)
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_ieee_addr_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for IEEE_addr_rsp.

### 5.3.14zb\_zdoSimpleDescReq()

请求目标简单描述符信息。

原型

```
zdo_status_t zb_zdoSimpleDescReq(u16 dstNwkAddr,
                                 zdo_simple_descriptor_req_t *pReq,
                                 u8 *seqNo, zdo_callback indCb)
```

## 返回值

`zdo_status_t`

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	<code>zdo_simple_descriptor_req_t *</code>	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	<code>zdo_callback</code>	Callback function for Simple_Desc_rsp.

## 5.3.15zb\_zdoNodeDescReq()

请求目标节点描述符信息。

### 原型

```
zdo_status_t zb_zdoNodeDescReq(u16 dstNwkAddr,
                                zdo_node_descriptor_req_t *pReq,
                                u8 *seqNo, zdo_callback indCb)
```

## 返回值

`zdo_status_t`

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	<code>zdo_node_descriptor_req_t *</code>	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	<code>zdo_callback</code>	Callback function for Node_Desc_rsp.

### 5.3.16zb\_zdoPowerDescReq()

请求目标电源描述符信息。

原型

```
zdo_status_t zb_zdoPowerDescReq(u16 dstNwkAddr,
                                 zdo_power_descriptor_req_t *pReq,
                                 u8 *seqNo, zdo_callback indCb)
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_power_descriptor_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Power_Desc_rsp.

### 5.3.17zb\_zdoActiveEpReq()

请求目标活跃的端点信息。

原型

```
zdo_status_t zb_zdoActiveEpReq(u16 dstNwkAddr, zdo_active_ep_req_t *pReq,
                                u8 *seqNo, zdo_callback indCb)
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_active_ep_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Active_Ep_rsp.

### 5.3.18zb\_zdoMatchDescReq()

请求目标匹配描述符信息。

原型

```
zdo_status_t zb_zdoMatchDescReq(u16 dstNwkAddr,
                                 zdo_match_descriptor_req_t *pReq,
                                 u8 *seqNo, zdo_callback indCb)
```

返回值

zdo\_status\_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_match_descriptor_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Match_Desc_rsp.

### 5.3.19zb\_isDeviceFactoryNew()

检查节点是否是新设备。

原型

```
bool zb_isDeviceFactoryNew(void)
```

返回值

TRUE or FALSE

### 5.3.20 zb\_isDeviceJoinedNwk()

检查节点是否已经入网。

原型

```
bool zb_isDeviceJoinedNwk(void)
```

返回值

TRUE or FALSE

### 5.3.21 zb\_getMacAssocPermit()

获取本节点的入网允许状态。

原型

```
bool zb_getMacAssocPermit(void)
```

返回值

TRUE or FALSE

### 5.3.22 zb\_apsExtPanidSet()

设置 Extend PAN ID。

原型

---

```
void zb_apsExtPanidSet(extPANId_t panId)
```

返回值

None

Name	Type	Description
panId	extPANId_t	Extend PAN identifier.

## 5.4 ZCL

### 5.4.1 zcl\_init()

初始化 ZCL 层，注册基础命名处理函数。

原型

```
void zcl_init(zcl_hookFn_t fn)
```

返回值

None

Name	Type	Description
fn	zcl_hookFn_t	Callback function.

### 5.4.2 zcl\_register()

注册应用层端点支持的 cluster、attributes 和 command。

原型

```
void zcl_register(u8 endpoint, u8 clusterNum, zcl_specClusterInfo_t *info)
```

## 返回值

None

Name	Type	Description
endpoint	u8	Endpoint.
clusterNum	u8	The total number of clusters supported by the endpoint.
info	zcl_specClusterInfo_t*	Information of the clusters.

## 5.5 OTA

### 5.5.1 ota\_init()

初始化 OTA 需要的端点信息和回调函数。

#### 原型

```
void ota_init( ota_type_e type, af_simple_descriptor_t *simpleDesc,
               ota_preamble_t *otaPreamble, otaCallBack_t *cb)
```

#### 返回值

None

Name	Type	Description
type	ota_type_e	Client or server.
simpleDesc	af_simple_descriptor_t*	Simple descriptor.
otaPreamble	ota_preamble_t*	Information of the OTA preamble.
cb	otaCallBack_t*	Callback function for OTA message.

## 5.5.2 ota\_queryStart()

启动 OTA 查询功能，单位：秒。

原型

```
void ota_queryStart(u16 seconds)
```

返回值

None

Name	Type	Description
seconds	u8	Query cycle, in second.

# 6. OTA

TLSR8 和 TLSR9 系列芯片支持 Flash 多地址启动：除了 Flash 地址 0x00000，还支持从 0x40000 读取固件运行。Telink Zigbee SDK 使用了该特性来实现 OTA 的功能。

从 2.4.1 节可知，我们分配了两块固件区域 Firmware 和 OTA-Image，固件大小应不大于 208K。

假设当前正在运行 Firmware 的固件，当设备执行 OTA 升级时，新的固件数据将被存储到 OTA-Image，在 OTA 结束并且验证通过后，将重启并运行 OTA-Image 的固件。后续 OTA，将交替执行。

## 6.1 OTA 初始化

```
typedef enum{
    OTA_TYPE_CLIENT,
    OTA_TYPE_SERVER
}ota_type_e;

void ota_init( ota_type_e type, af_simple_descriptor_t *simpleDesc,
ota_preamble_t *otaPreamble, //current running fw info
otaCallBack_t *cb );
```

OTA 设备分为服务设备（Server）和终端设备（Client）。因此，在 OTA 初始化时需要注意 OTA 初始化的服务类型。

一般地，被升级设备为终端设备（Client），为被升级设备提供新固件的设备为服务设备（Server）。

## 6.2 OTA Server

OTA 服务端需要将被升级设备所需要的新固件写入 OTA-Image 区域供 OTA 使用。

例如：服务端自己运行的固件在 Firmware 区，那么可以将目标设备的 OTA image（包含 OTA Header 的新固件，参考 Zigbee Cluster Library Specification）暂存到 OTA-Image 区域。

当 OTA 服务端在调用 ota\_init 函数后，会自动装载 OTA image 的信息。

## 6.3 OTA Client

设置 OTA 终端设备的 OTA 查询周期，单位：秒。

```
void ota_queryStart(u16 seconds);
```

OTA 终端设备在调用 ota\_init 函数后，将会自动重载上一次的 OTA 进度信息，如果入网成功，将在设置的 seconds 秒后开始 OTA 请求。

## 6.4 OTA Image

OTA Image 是一个包含 OTA Header 供 OTA Client 端实现空中升级的 zigbee 文件。用户可以使用 Telink 提供的 zigbee\_ota\_tool 工具，将 Client 端的固件加上头信息转换成升级文件。SDK 中要求待升级固件的文件版本（请查阅 2.3.3 章节）必须大于当前的文件版本，否则，当 Client 端发起 OTA 查询时，Server 端将回复 NO\_IMAGE\_AVAILABLE。

OTA Image 文件生成，以 sampleLight\_8258 为例：

- 1) 将 sampleLight\_8258.bin 拷贝到 “zigbee\_ota\_tool\_v2.2.exe” 所在文件夹；
- 2) 双击 “zigbee\_ota\_tool\_v2.2.exe” 按提示操作；
- 3) 或使用命令行 “./ zigbee\_ota\_tool\_v2.2.exe [arg1] [arg2]”。

[arg1]: 文件名，需要转译的文件名，输入参数如：sampleLight\_8258.bin。

[arg2]: 如果不存在该参数，代表不加密。如果存在，代表使用 AES-128 加密，输入参数如：

00112233445566778899AABBCCDDEEFF。

## 7. 扩展功能 HCI 接口

Telink Zigbee SDK 提供了 HCI 接口方便用户做扩展应用开发，具体实现请参见..../zigbee/zbhci。

当前 HCI 接口实现提供了 3 种方式：UART、USB CDC 和 USB Print。相对应的宏如下：

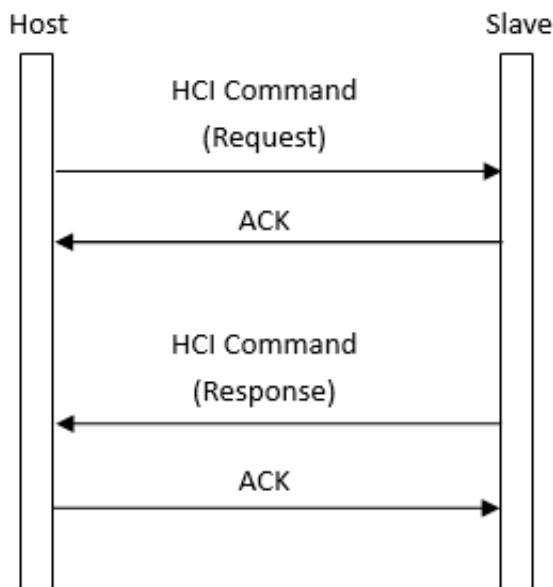
UART	->	#define ZBHCI_UART	1
USB CDC	->	#define ZBHCI_USB_CDC	1
USB Print	->	#define ZBHCI_USB_PRINT	1

如需使用某种方式，只需要将相应的宏设为 1 即可。

例如：sampleGW 例程要使用 UART 方式的 HCI 功能，请将 ZBHCI\_UART 设置为 1。

### 7.1 控制流程图

图 7-1 控制流程图



## 7.2 命令帧格式

Header	Message Type	Message Length	Checksum	Payload	Tail
1 Byte	2 Bytes	2 Bytes	1 Byte	Variable	1 Byte

Field	Description
<b>Header</b>	Header flag, shall be 0x55.
<b>Message Type</b>	Message Type, big-endian.
<b>Message Length</b>	Payload length, big-endian.
<b>Checksum</b>	The checksum.
<b>Payload</b>	Payload.
<b>Tail</b>	Tail flag, shall be 0xAA.

注：HCI 命令的所有字段都是大端模式（big-endian）。

## 7.3 应答格式 (Acknowledge format)

### 7.3.1 Message Type

Message Type	Value
ZBHCI_CMD_ACKNOWLEDGE	0x8000

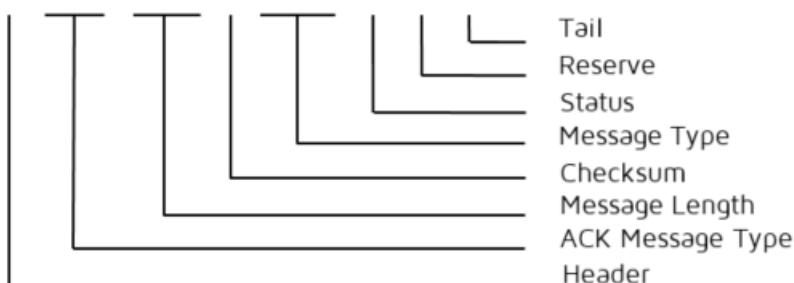
### 7.3.2 Payload

Message Type	Status	Reserved
2 Bytes	1 Byte	1 Byte

Name	Description
Message Type	HCI command message type.
Status	0 = Success; 1 = Wrong parameter; 2 = Unsupported command; 3 = Busy; 4 = No memory.
Reserved	0

示例：

55 80 00 00 04 00 XX XX XX XX AA



## 7.4 BDB 命令

### 7.4.1 Message Type

Message Type	Value
ZBHCI_CMD_BDB_COMMISSION_FORMATION	0x0001
ZBHCI_CMD_BDB_COMMISSION_STEER	0x0002
ZBHCI_CMD_BDB_COMMISSION_TOUCHLINK	0x0003
ZBHCI_CMD_BDB_COMMISSION_FINDBIND	0x0004
ZBHCI_CMD_BDB_FACTORY_RESET	0x0005
ZBHCI_CMD_BDB_PRE_INSTALL_CODE	0x0006
ZBHCI_CMD_BDB_CHANNEL_SET	0x0007

### 7.4.2 Payload

#### 1) ZBHCI\_CMD\_BDB\_COMMISSION\_FORMATION

无 payload。

示例：55 00 01 00 00 00 AA

#### 2) ZBHCI\_CMD\_BDB\_COMMISSION\_STEER

无 payload。

示例：55 00 02 00 00 00 AA

#### 3) ZBHCI\_CMD\_BDB\_COMMISSION\_TOUCHLINK

Role
1 Byte

Name	Description
Role	1 = Touch link initiator; 2 = Touch link target.

示例：55 00 03 00 01 00 XX AA

#### 4) ZBHCI\_CMD\_BDB\_COMMISSION\_FINDBIND

Role
1 Byte

Name	Description
Role	1 = Find & Bind initiator; 2 = Find & Bind target.

示例：55 00 04 00 01 00 XX AA

#### 5) ZBHCI\_CMD\_BDB\_FACTORY\_RESET

无 payload。

示例：55 00 05 00 00 00 AA

## 6) ZBHCI\_CMD\_BDB\_PRE\_INSTALL\_CODE

devAddr	uniqueLinkKey
8 Bytes	16 Bytes

Name	Description
devAddr	The IEEE address of the pre-configured device.
uniqueLinkKey	The link key for the device.

示例：55 00 06 00 18 00 XX[8 Bytes] XX[16 Bytes] AA

## 7) ZBHCI\_CMD\_BDB\_CHANNEL\_SET

channelIdx
1 Byte

Name	Description
channelIdx	11~26

示例：55 00 07 00 01 00 XX AA

## 7.5 网络管理命令 (Network management command)

### 7.5.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_DISCOVERY_NWK_ADDR_REQ	0x0010
ZBHCI_CMD_DISCOVERY_IEEE_ADDR_REQ	0x0011
ZBHCI_CMD_DISCOVERY_NODE_DESC_REQ	0x0012
ZBHCI_CMD_DISCOVERY_SIMPLE_DESC_REQ	0x0013
ZBHCI_CMD_DISCOVERY_MATCH_DESC_REQ	0x0014
ZBHCI_CMD_DISCOVERY_ACTIVE_EP_REQ	0x0015
ZBHCI_CMD_DISCOVERY_LEAVE_REQ	0x0016
ZBHCI_CMD_BIND_REQ	0x0020
ZBHCI_CMD_UNBIND_REQ	0x0021
ZBHCI_CMD_MGMT_LQI_REQ	0x0030
ZBHCI_CMD_MGMT_BIND_REQ	0x0031
ZBHCI_CMD_MGMT_LEAVE_REQ	0x0032
ZBHCI_CMD_MGMT_DIRECT_JOIN_REQ	0x0033
ZBHCI_CMD_MGMT_PERMIT_JOIN_REQ	0x0034
ZBHCI_CMD_MGMT_NWK_UPDATE_REQ	0x0035
ZBHCI_CMD_NODES_JOINED_GET_REQ	0x0040
ZBHCI_CMD_NODES_TOGGLE_TEST_REQ	0x0041

## 7.5.2 Payload (Host)

### 7.5.2.1 ZBHCI\_CMD\_DISCOVERY\_NWK\_ADDR\_REQ

dstAddr	ieeeAddr	reqType	startIdx
2 Bytes	8 Bytes	1 Byte	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.
ieeeAddr	The IEEE address to be matched by the remote device.
reqType	Request type for this command: 0x00 – Single device response; 0x01 – Extended response; 0x02 ~ 0xFF – Reserved.
startIdx	If the request type for this command is extended response, the startIdx provides the starting index for the requested elements of the associated device list.

示例：55 00 10 00 0C 00 XX[2 Bytes] XX[8 Bytes] XX XX AA

### 7.5.2.2 ZBHCI\_CMD\_DISCOVERY\_IEEE\_ADDR\_REQ

dstAddr	nwkAddrOfInterest	reqType	startIdx
2 Bytes	2 Bytes	1 Byte	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.

Name	Description
nwkAddrOfInterest	The NWK address that is used for IEEE address mapping.
reqType	Request type for this command: 0x00 – Single device response; 0x01 – Extended response; 0x02 ~ 0xFF – Reserved.
startIdx	If the request type for this command is extended response, the startIdx provides the starting index for the requested elements of the associated device list.

示例：55 00 11 00 06 00 XX[2 Bytes] XX[2 Bytes] XX XX AA

### 7.5.2.3 ZBHCI\_CMD\_DISCOVERY\_NODE\_DESC\_REQ

dstAddr	nwkAddrOfInterest
2 Bytes	2 Bytes

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.
nwkAddrOfInterest	NWK address of the target.

示例：55 00 12 00 04 00 XX[2 Bytes] XX[2 Bytes] AA

### 7.5.2.4 ZBHCI\_CMD\_DISCOVERY\_SIMPLE\_DESC\_REQ

dstAddr	nwkAddrOfInterest	endpoint
2 Bytes	2 Bytes	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.
nwkAddrOfInterest	NWK address of the target.
endpoint	The endpoint on the destination.

示例：55 00 13 00 05 00 XX[2 Bytes] XX[2 Bytes] XX AA

### 7.5.2.5 ZBHCI\_CMD\_DISCOVERY\_MATCH\_DESC\_REQ

dstAddr	nwkAddrOfInterest	profileID	numInClusters	numOutClusters	clusterList
2 Bytes	2 Bytes	2 Bytes	1 Byte	1 Byte	n Bytes

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.
nwkAddrOfInterest	NWK address of the target.
profileID	Profile ID to be matched at the destination.
numInClusters	The number of input clusters provided for matching within the inClusterList.
numOutClusters	The number of output clusters provided for matching within the outClusterList.

Name	Description
clusterList	<p>inClusterList + outClusterList.</p> <p>List of input cluster IDs to be used for matching, the inClusterList is the desired list to be matched by the Remote Device (the elements of the inClusterList are the supported output clusters of the Local Device).</p> <p>List of output cluster IDs to be used for matching, the outClusterList is the desired list to be matched by the Remote Device (the elements of the outClusterList are the supported input clusters of the Local Device).</p>

示例：55 00 14 msgLenH msgLenL 00 XX[2 Bytes] XX[2 Bytes] XX[2 Bytes] XX XX XX[n Bytes] AA

#### 7.5.2.6 ZBHCI\_CMD\_DISCOVERY\_ACTIVE\_EP\_REQ

dstAddr	nwkAddrOfInterest
2 Bytes	2 Bytes

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.
nwkAddrOfInterest	NWK address of the target.

示例：55 00 15 00 04 00 XX[2 Bytes] XX[2 Bytes] AA

#### 7.5.2.7 ZBHCI\_CMD\_DISCOVERY\_LEAVE\_REQ

devAddr	rejoin	removeChildren
8 Bytes	1 Byte	1 Byte

Name	Description
devAddr	The IEEE address of the device to be removed or NULL if the device removes itself.
rejoin	TRUE if the device is being asked to leave from the current parent and requested to rejoin the network. Otherwise, the parameter has a value of FALSE.
removeChildren	TRUE if the device being asked to leave the network is also being asked to remove its child device, if any. Otherwise, it has a value of FALSE.

示例：55 00 16 00 0A 00 XX[8 Bytes] XX XX AA

#### 7.5.2.8 ZBHCI\_CMD\_BIND\_REQ / ZBHCI\_CMD\_UNBIND\_REQ

srcIEEEAddr	srcEndpoint	clusterID	dstAddrMode	dstAddr	dstEndpoint
8 Bytes	1 Byte	2 Byte	1 Byte	2 or 8 Bytes	0 or 1 Byte

Name	Description
srcIEEEAddr	The IEEE address for the source.
srcEndpoint	The source endpoint for the binding entry.
clusterID	The identifier of the cluster on the source device that is bound to the destination.
dstAddrMode	The addressing mode for the destination address used in this command. 0x00 – Reserved; 0x01 – 16-bit group address for dstAddr and dstEndpoint not present; 0x02 – Reserved; 0x03 – 64-bit extended address for dstAddr and dstEndpoint present; 0x04 ~ 0xFF – Reserved.
dstAddr	The destination address for the binding entry.

Name	Description
dstEndpoint	Shall be present only if the dstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

示例：

ZBHCI\_CMD\_BIND\_REQ

55 00 20 msgLenH msgLenL 00 XX[8 Bytes] XX XX[2 Bytes] XX XX[2 or 8 Bytes] XX[0 or 1 Bytes] AA

ZBHCI\_CMD\_UNBIND\_REQ

55 00 21 msgLenH msgLenL 00 XX[8 Bytes] XX XX[2 Bytes] XX XX[2 or 8 Bytes] XX[0 or 1 Bytes] AA

### 7.5.2.9 ZBHCI\_CMD\_MGMT\_LQI\_REQ

dstAddr	startIdx
2 Bytes	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast.
startIdx	Starting index for the requested elements of the neighbor table.

示例：55 00 30 00 03 00 XX[2 Bytes] XX AA

### 7.5.2.10 ZBHCI\_CMD\_MGMT\_BIND\_REQ

dstAddr	startIdx
2 Bytes	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast.
startIndex	Starting index for the requested elements of the binding table.

示例：55 00 31 00 03 00 XX[2 Bytes] XX AA

#### 7.5.2.11 ZBHCI\_CMD\_MGMT\_LEAVE\_REQ

dstAddr	devAddr	rejoin	removeChildren
2 Bytes	8 Bytes	1 Byte	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast.
devAddr	The IEEE address of the device to be removed or NULL if the device removes itself.
rejoin	TRUE if the device is being asked to leave from the current parent and requested to rejoin the network. Otherwise, the parameter has a value of FALSE.
removeChildren	TRUE if the device being asked to leave the network is also being asked to remove its child device, if any. Otherwise, it has a value of FALSE.

示例：55 00 32 00 0C 00 XX[2 Bytes] XX[8 Bytes] XX XX AA

### 7.5.2.12 ZBHCI\_CMD\_MGMT\_PERMIT\_JOIN\_REQ

<b>dstAddr</b>	<b>permitDuration</b>	<b>TC_significance</b>
2 Bytes	1 Byte	1 Byte

<b>Name</b>	<b>Description</b>
dstAddr	The destination address to which this command will be sent.
permitDuration	The length of time in seconds during which the coordinator or router will allow associations. The value 0x00 and 0xff indicate that permission is disabled or enabled, respectively, without a specified limit.
TC_significance	This field shall always have a value of 1, indicating a request to change the Trust Center policy. If a frame is received with a value of 0, it shall be treated as having a value of 1.

示例：55 00 34 00 04 00 XX[2 Bytes] XX XX AA

### 7.5.2.13 ZBHCI\_CMD\_MGMT\_NWK\_UPDATE\_REQ

<b>dstAddr</b>	<b>nwkManagerAddr</b>	<b>scanChannels</b>	<b>scanDuration</b>	<b>scanCount/nwkUpdateId</b>
2 Bytes	2 Bytes	4 Bytes	1 Byte	1 Byte

<b>Name</b>	<b>Description</b>
dstAddr	The destination address to which this command will be sent.
nwkManagerAddr	This field shall be present only if the scanDuration is set to 0xff, and, when present, it indicates the NWK address for the device with the Network Manager bit set in its Node Descriptor.
scanChannels	Channel bit mask, 32-bit field structure.

Name	Description
scanDuration	A value used to calculate the length of time to spend on scanning each channel. 0x00 ~ 0x05 or 0xfe or 0xff.
scanCount	This field represents the number of energy scans to be conducted and reported. This field shall be present only if the scanDuration is within the range of 0x00 to 0x05.
nwkUpdateld	The value of the nwkUpdateld contained in this request. This value is set by the Network Channel Manager prior to sending the message. This field shall only be present if the scanDuration is 0xfe or 0xff. If the scanDuration is 0xff, then the value in the nwkUpdateld shall be ignored.

示例：55 00 35 00 0A 00 XX[2 Bytes] XX[2 Bytes] XX[4 Bytes] XX XX AA

#### 7.5.2.14 ZBHCI\_CMD\_NODES\_JOINED\_GET\_REQ

startIdx
1 Byte

Name	Description
startIdx	Starting index for the requested elements of the joined node list.

示例：55 00 40 00 01 00 XX AA

#### 7.5.2.15 ZBHCI\_CMD\_NODES\_TOGGLE\_TEST\_REQ

on/off	timerInterval
1 Byte	1 Byte

Name	Description
on/off	On/off command.
timerInterval	The timer interval of the data transmission. Unit: millisecond.

示例：55 00 41 00 02 00 XX XX AA

### 7.5.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_DISCOVERY_NWK_ADDR_RSP	0x8010
ZBHCI_CMD_DISCOVERY_IEEE_ADDR_RSP	0x8011
ZBHCI_CMD_DISCOVERY_NODE_DESC_RSP	0x8012
ZBHCI_CMD_DISCOVERY_SIMPLE_DESC_RSP	0x8013
ZBHCI_CMD_DISCOVERY_MATCH_DESC_RSP	0x8014
ZBHCI_CMD_DISCOVERY_ACTIVE_EP_RSP	0x8015
ZBHCI_CMD_BIND_RSP	0x8020
ZBHCI_CMD_UNBIND_RSP	0x8021
ZBHCI_CMD_MGMT_LQI_RSP	0x8030
ZBHCI_CMD_MGMT_BIND_RSP	0x8031
ZBHCI_CMD_MGMT_LEAVE_RSP	0x8032
ZBHCI_CMD_MGMT_DIRECT_JOIN_RSP	0x8033
ZBHCI_CMD_MGMT_PERMIT_JOIN_RSP	0x8034

Message Type	Value
ZBHCI_CMD_MGMT_NWK_UPDATE_RSP	0x8035
ZBHCI_CMD_NODES_JOINED_GET_RSP	0x8040
ZBHCI_CMD_NODES_TOGGLE_TEST_RSP	0x8041
ZBHCI_CMD_NODES_DEV_ANNCE_IND	0x8043

## 7.5.4 Payload (Slave)

### 7.5.4.1 ZBHCI\_CMD\_DISCOVERY\_NWK\_ADDR\_RSP

seqNum	status	ieeeAddr	nwkAddr	numAssocDev	startIndex	assocDevList
1 Byte	1 Byte	8 Bytes	2 Bytes	0 or 1 Byte	0 or 1 Byte	0 or n Bytes

Name	Description
seqNum	ZDP transaction sequence number.
status	The status of the request command.
ieeeAddr	64-bit address for the Remote Device.
nwkAddr	16-bit address for the Remote Device.
numAssocDev	Count of the number of 16-bit short address to follow.
startIndex	Starting index into the list of associated devices for this report.
assocDevList	The list of associated devices.

示例：55 80 10 msgLenH msgLenL 00 XX XX XX[8 Bytes] XX[2 Bytes]

{XX XX XX[n Bytes]} AA

### 7.5.4.2 ZBHCI\_CMD\_DISCOVERY\_IEEE\_ADDR\_RSP

<b>seqNum</b>	<b>status</b>	<b>ieeeAddr</b>	<b>nwkAddr</b>	<b>numAssocDev</b>	<b>startIdx</b>	<b>assocDevList</b>
1 Byte	1 Byte	8 Bytes	2 Bytes	0 or 1 Byte	0 or 1 Byte	0 or n Bytes

<b>Name</b>	<b>Description</b>
seqNum	ZDP transaction sequence number.
status	The status of the request command.
ieeeAddr	64-bit address for the Remote Device.
nwkAddr	16-bit address for the Remote Device.
numAssocDev	Count of the number of 16-bit short address to follow.
startIdx	Starting index into the list of associated devices for this report.
assocDevList	The list of associated devices.

示例：55 80 11 msgLenH msgLenL 00 XX XX XX[8 Bytes] XX[2 Bytes]

{XX XX XX[n Bytes]} AA

### 7.5.4.3 ZBHCI\_CMD\_DISCOVERY\_NODE\_DESC\_RSP

<b>seqNum</b>	<b>status</b>	<b>nwkAddrOfInterest</b>	<b>nodeDesc</b>
1 Byte	1 Byte	2 Bytes	0 or n Bytes

<b>Name</b>	<b>Description</b>
seqNum	ZDP transaction sequence number.

Name	Description
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.
nodeDesc	This field shall only be included in the frame if the status field is SUCCESS.

示例：55 80 12 msgLenH msgLenL 00 XX XX XX[2 Bytes] {XX[n Bytes]} AA

#### 7.5.4.4 ZBHCI\_CMD\_DISCOVERY\_SIMPLE\_DESC\_RSP

seqNum	status	nwkAddrOfInterest	length	simpleDesc
1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

Name	Description
seqNum	ZDP transaction sequence number.
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.
length	The length of simple description.
simpleDesc	This field shall only be included in the frame if the status field is SUCCESS.

示例：55 80 13 msgLenH msgLenL 00 XX XX XX[2 Bytes] XX {XX[n Bytes]} AA

#### 7.5.4.5 ZBHCI\_CMD\_DISCOVERY\_MATCH\_DESC\_RSP

<b>seqNum</b>	<b>status</b>	<b>nwkAddrOfInterest</b>	<b>matchLen</b>	<b>matchList</b>
1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

<b>Name</b>	<b>Description</b>
seqNum	ZDP transaction sequence number.
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.
matchLen	The count of endpoints on the Remote Device that match the request criteria.
matchList	List of bytes each of which represents an 8-bit endpoint.

示例：55 80 14 msgLenH msgLenL 00 XX XX XX[2 Bytes] XX {XX[n Bytes]} AA

#### 7.5.4.6 ZBHCI\_CMD\_DISCOVERY\_ACTIVE\_EP\_RSP

<b>seqNum</b>	<b>status</b>	<b>nwkAddrOfInterest</b>	<b>activeEpCount</b>	<b>epList</b>
1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

<b>Name</b>	<b>Description</b>
seqNum	ZDP transaction sequence number.
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.

Name	Description
activeEpCount	The count of active endpoints.
epList	List of active endpoints.

示例：55 80 15 msgLenH msgLenL 00 XX XX XX[2 Bytes] XX {XX[n Bytes]} AA

#### 7.5.4.7 ZBHCI\_CMD\_BIND\_RSP / ZBHCI\_CMD\_UNBIND\_RSP

seqNum	status
1 Byte	1 Byte

Name	Description
seqNum	ZDP transaction sequence number.
status	The status of the request command.

示例：

ZBHCI\_CMD\_BIND\_RSP

55 80 20 00 02 00 XX XX AA

ZBHCI\_CMD\_UNBIND\_RSP

55 80 21 00 02 00 XX XX AA

### 7.5.4.8 ZBHCI\_CMD\_MGMT\_LQI\_RSP

<b>seqNum</b>	<b>status</b>	<b>neighborTabEntries</b>	<b>startIdx</b>	<b>neighborTabListCount</b>	<b>neighborTabList</b>
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	n Bytes

<b>Name</b>	<b>Description</b>
seqNum	ZDP transaction sequence number.
status	The status of the request command.
neighborTabEntries	Total number of Neighbor Table entries within the Remote Device.
startIdx	Starting index within the Neighbor Table to begin reporting for the neighborTabList.
neighborTabListCount	Number of Neighbor Table entries included within neighborTabList.
neighborTabList	A list of descriptors, beginning with the startIdx element and continuing for neighborTabListCount.

示例：55 80 30 msgLenH msgLenL 00 XX XX XX XX XX {XX[n Bytes]} AA

### 7.5.4.9 ZBHCI\_CMD\_MGMT\_BIND\_RSP

<b>seqNum</b>	<b>status</b>	<b>bindingTabEntries</b>	<b>startIdx</b>	<b>bindingTabListCount</b>	<b>bindingTabList</b>
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	n Bytes

<b>Name</b>	<b>Description</b>
seqNum	ZDP transaction sequence number.

Name	Description
status	The status of the request command.
bindingTabEntries	Total number of Binding Table entries within the Remote Device.
startIdx	Starting index within the Binding Table to begin reporting for the bindingTabList.
bindingTabListCount	Number of Binding Table entries included within bindingTabList.
bindingTabList	A list of descriptors, beginning with the startIdx element and continuing for bindingTabListCount.

示例：55 80 31 msgLenH msgLenL 00 XX XX XX XX XX {XX[n Bytes]} AA

#### 7.5.4.10 ZBHCI\_CMD\_MGMT\_LEAVE\_RSP

seqNum	status
1 Byte	1 Byte

Name	Description
seqNum	ZDP transaction sequence number.
status	The status of the request command.

示例：55 80 32 00 02 00 XX XX AA

### 7.5.4.11 ZBHCI\_CMD\_MGMT\_PERMIT\_JOIN\_RSP

<b>seqNum</b>	<b>status</b>
1 Byte	1 Byte

<b>Name</b>	<b>Description</b>
seqNum	ZDP transaction sequence number.
status	The status of the request command.

示例：55 80 34 00 02 00 XX XX AA

### 7.5.4.12 ZBHCI\_CMD\_NODES\_JOINED\_GET\_RSP

<b>status</b>	<b>totalCnt</b>	<b>startIndex</b>	<b>listCount</b>	<b>macAddrList</b>
1 Byte	1 Byte	1 Byte	1 Byte	n Bytes

<b>Name</b>	<b>Description</b>
status	The status of the request command.
totalCnt	The total count of the joined nodes.
startIndex	Starting index within the mac address list.
listCount	The count of the MAC address list in the current packet.
macAddrList	The MAC address list in the current packet.

示例：55 80 40 msgLenH msgLenL 00 XX XX XX XX {XX[n Bytes]} AA

### 7.5.4.13 ZBHCI\_CMD\_NODES\_TOGGLE\_TEST\_RSP

无 payload。

示例：55 80 41 00 00 00 AA

### 7.5.4.14 ZBHCI\_CMD\_NODES\_DEV\_ANNCE\_RSP

nwkAddr	ieeeAddr	capability
2 Bytes	8 Bytes	1 Byte

Name	Description
nwkAddr	NWK address of the joined device.
ieeeAddr	IEEE address of the joined device.
capability	Capability of the joined device.

示例：55 80 43 00 0B 00 XX[2 Bytes] XX[8 Bytes] XX AA

## 7.6 ZCL Cluster 命令

### 7.6.1 ZCL 命令 header 格式

ZCLCmdHdr:

dstAddrMode	dstAddr	srcEp	dstEp
1 Byte	0/2/8 Bytes	1 Byte	1 Byte

Name	Description
dstAddrMode	Destination address mode: 0 – without destination address and endpoint, for binding; 1 – with group address; 2 – with destination short address and endpoint; 3 – with destination IEEE address and endpoint.
dstAddr	Destination address for this command.
srcEp	Source endpoint.
dstEp	Destination endpoint if dstAddrMode is 2 or 3.

## 7.6.2 General cluster command

### 7.6.2.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_ATTR_READ	0x0100
ZBHCI_CMD_ZCL_ATTR_WRITE	0x0101
ZBHCI_CMD_ZCL_CONFIG_REPORT	0x0102
ZBHCI_CMD_ZCL_READ_REPORT_CFG	0x0103

### 7.6.2.2 Payload (Host)

#### 1) ZBHCI\_CMD\_ZCL\_ATTR\_READ

ZCLCmdHdr	direction	clusterID	attrNum	attrList
n Bytes	1 Byte	2 Bytes	1 Byte	n Bytes

attrList:

<b>attrID[0]</b>	<b>attrID[1]</b>	...	<b>attrID[n]</b>
2 Bytes	2 Bytes	...	2 Bytes

<b>Name</b>	<b>Description</b>
ZCLCmdHdr	ZCL command header.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.
attrNum	The number of attributes to be read.
attrList	The list of the attribute IDs to be read.

示例： 55 01 00 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] XX {XX[n Bytes]} AA

## 2) ZBHCI\_CMD\_ZCL\_ATTR\_WRITE

<b>ZCLCmdHdr</b>	<b>direction</b>	<b>clusterID</b>	<b>attrNum</b>	<b>attrList[0]</b>	...	<b>attrList[n]</b>
n Bytes	1 Byte	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

<b>attrID</b>	<b>dataType</b>	<b>attrData</b>
2 Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.
attrNum	The number of attributes to be written.
attrList	The list of the attributes to be written.

示例： 55 01 01 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] XX {XX[n Bytes] ...} AA

### 3) ZBHCI\_CMD\_ZCL\_CONFIG\_REPORT

ZCLCmdHdr	direction	clusterID	attrNum	attrList[0]	...	attrList[n]
n Bytes	1 Byte	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

Report Direction	attrID	dataType	minRep Interval	maxRep Interval	reportable Change	timeout Period
1 Byte	2 Bytes	1 Byte	2 Bytes	2 Bytes	n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.

Name	Description
attrNum	The number of attributes to be configured.
attrList	The list of the attributes to be configured.

示例： 55 01 02 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] XX {XX[n Bytes] ...} AA

#### 4) ZBHCI\_CMD\_ZCL\_READ\_REPORT\_CFG

ZCLCmdHdr	direction	clusterID	attrNum	attrList[0]	...	attrList[n]
n Bytes	1 Byte	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

reportDirection	attrID
1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.
attrNum	The number of attributes' configuration to be read.
attrList	The list of the attributes to be read.

示例： 55 01 03 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] XX {XX[n Bytes] ...} AA

### 7.6.2.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_ZCL_ATTR_READ_RSP	0x8100
ZBHCI_CMD_ZCL_ATTR_WRITE_RSP	0x8101
ZBHCI_CMD_ZCL_CONFIG_REPORT_RSP	0x8102
ZBHCI_CMD_ZCL_READ_REPORT_CFG_RSP	0x8103
ZBHCI_CMD_ZCL_REPORT_MSG_RCV	0x8104

### 7.6.2.4 Payload (Slave)

- 1) ZBHCI\_CMD\_ZCL\_ATTR\_READ\_RSP

attrNum	attrList[0]	...	attrList[n]
1 Byte	n Bytes	...	n Bytes

attrList:

attrID	status	dataType	data
2 Bytes	1 Byte	1 Byte	n Bytes

Name	Description
attrNum	The number of attributes to be read.
attrList	The list of the attributes to be read.

示例： 55 81 00 msgLenH msgLenL 00 XX {XX[n Bytes] ...} AA

## 2) ZBHCI\_CMD\_ZCL\_ATTR\_WRITE\_RSP

<b>attrNum</b>	<b>attrList[0]</b>	...	<b>attrList[n]</b>
1 Byte	n Bytes	...	n Bytes

attrList:

<b>status</b>	<b>attrID</b>
1 Byte	2 Bytes

<b>Name</b>	<b>Description</b>
attrNum	The number of attributes to be written.
attrList	The list of the attributes to be written.

示例： 55 81 01 msgLenH msgLenL 00 XX {XX[n Bytes] ...} AA

## 3) ZBHCI\_CMD\_ZCL\_CONFIG\_REPORT\_RSP

<b>attrNum</b>	<b>attrList[0]</b>	...	<b>attrList[n]</b>
1 Byte	n Bytes	...	n Bytes

attrList:

<b>status</b>	<b>reportDirection</b>	<b>attrID</b>
1 Byte	1 Byte	2 Bytes

Name	Description
attrNum	The number of attributes' reporting to be configured.
attrList	The list of the attributes to be configured.

示例： 55 81 02 msgLenH msgLenL 00 XX {XX[n Bytes] ...} AA

#### 4) ZBHCI\_CMD\_ZCL\_READ\_REPORT\_CFG\_RSP

attrNum	attrList[0]	...	attrList[n]
1 Byte	n Bytes	...	n Bytes

attrList:

status	Report Direction	attrID	dataType	minRep Interval	maxRep Interval	Reportable Change	timeout Period
1 Byte	1 Byte	2 Bytes	1 Byte	2 Bytes	2 Bytes	n Bytes	2 Bytes

Name	Description
attrNum	The number of attributes' reporting to be read.
attrList	The list of the attributes to be read.

示例： 55 81 03 msgLenH msgLenL 00 XX {XX[n Bytes] ...} AA

#### 5) ZBHCI\_CMD\_ZCL\_REPORT\_MSG\_RCV

<b>srcAddr</b>	<b>srcEp</b>	<b>attrNum</b>	<b>attrList[0]</b>	...	<b>attrList[n]</b>
2 Bytes	1 Byte	1 Byte	n Bytes	...	n Bytes

attrList:

<b>attrID</b>	<b>dataType</b>	<b>data</b>
2 Bytes	1 Byte	n Bytes

<b>Name</b>	<b>Description</b>
srcAddr	The source address of the reporting message.
srcEp	The source endpoint of the reporting message.
attrNum	The number of attributes' reporting message to be received.
attrList	The list of the attributes' reporting message to be received.

示例： 55 81 04 msgLenH msgLenL 00 XX[2 Bytes] XX XX {XX[n Bytes] ...} AA

### 7.6.3 Basic cluster command

#### 7.6.3.1 Message Type (Host)

<b>Message Type</b>	<b>Value</b>
ZBHCI_CMD_ZCL_BASIC_RESET	0x0110

### 7.6.3.2 Payload (Host)



示例： 55 01 10 msgLenH msgLenL 00 XX[n Bytes] AA

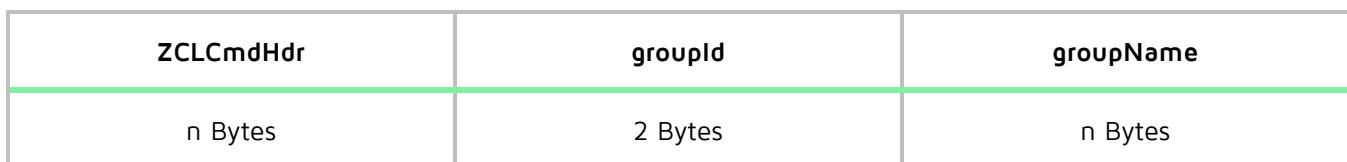
### 7.6.4 Group cluster command

#### 7.6.4.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_GROUP_ADD	0x0120
ZBHCI_CMD_ZCL_GROUP_VIEW	0x0121
ZBHCI_CMD_ZCL_GROUP_GET_MEMBERSHIP	0x0122
ZBHCI_CMD_ZCL_GROUP_REMOVE	0x0123
ZBHCI_CMD_ZCL_GROUP_REMOVE_ALL	0x0124
ZBHCI_CMD_ZCL_GROUP_ADD_IF_IDENTIFYING	0x0125

#### 7.6.4.2 Payload (Host)

##### 1) ZBHCI\_CMD\_ZCL\_GROUP\_ADD



Name	Description
ZCLCmdHdr	ZCL command header.
groupId	Group identifier.
groupName	Group name, character string.

示例： 55 01 20 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] {XX[n Bytes]} AA

## 2) ZBHCI\_CMD\_ZCL\_GROUP\_VIEW

ZCLCmdHdr	groupId
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	Group identifier.

示例： 55 01 21 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

## 3) ZBHCI\_CMD\_ZCL\_GROUP\_GET\_MEMBERSHIP

ZCLCmdHdr	groupCount	groupList
n Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupCount	Group count.
groupList	Group list.

示例： 55 01 22 msgLenH msgLenL 00 XX[n Bytes] XX XX[n Bytes] AA

#### 4) ZBHCI\_CMD\_ZCL\_GROUP\_REMOVE

ZCLCmdHdr	groupId
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	Group identifier.

示例： 55 01 23 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

#### 5) ZBHCI\_CMD\_ZCL\_GROUP\_REMOVE\_ALL

ZCLCmdHdr
n Bytes

示例： 55 01 24 msgLenH msgLenL 00 XX[n Bytes] AA

## 6) ZBHCI\_CMD\_ZCL\_GROUP\_ADD\_IF\_IDENTIFYING

ZCLCmdHdr	groupId	groupName
n Bytes	2 Bytes	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	Group identifier.
groupName	Group name, character string.

示例：55 01 25 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] {XX[n Bytes]} AA

## 7.6.4.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_ZCL_GROUP_ADD_RSP	0x8120
ZBHCI_CMD_ZCL_GROUP_VIEW_RSP	0x8121
ZBHCI_CMD_ZCL_GROUP_GET_MEMBERSHIP_RSP	0x8122
ZBHCI_CMD_ZCL_GROUP_REMOVE_RSP	0x8123

## 7.6.4.4 Payload (Slave)

## 1) ZBHCI\_CMD\_ZCL\_GROUP\_ADD\_RSP

status	groupId
1 Byte	2 Bytes

Name	Description
status	The status field is set to SUCCESS, DUPLICATE_EXISTS, or INSUFFICIENT_SPACE as appropriate.
groupId	Group identifier.

示例： 55 81 20 00 03 00 XX XX[2 Bytes] AA

## 2) ZBHCI\_CMD\_ZCL\_GROUP\_VIEW\_RSP

status	groupId	groupName
1 Byte	2 Bytes	n Bytes

Name	Description
status	The status field is set to SUCCESS, or NOT_FOUND as appropriate.
groupId	Group identifier.
groupName	Group name, character string.

示例： 55 81 21 msgLenH msgLenL 00 XX XX[2 Bytes] XX[n Bytes] AA

## 3) ZBHCI\_CMD\_ZCL\_GROUP\_GET\_MEMBERSHIP\_RSP

capability	groupCount	groupList
1 Byte	1 Byte	n Bytes

Name	Description
capability	The remaining capability of the group table of the device.
groupCount	The number of groups contained in the group list field.
groupName	The list of groupId in the group list field.

示例： 55 81 22 msgLenH msgLenL 00 XX XX XX[n Bytes] AA

#### 4) ZBHCI\_CMD\_ZCL\_GROUP\_REMOVE\_RSP

status	groupId
1 Byte	2 Bytes

Name	Description
status	The status field is set to SUCCESS, DUPLICATE_EXISTS, or INSUFFICIENT_SPACE as appropriate.
groupId	Group identifier.

示例： 55 81 23 00 03 00 XX XX[2 Bytes] AA

### 7.6.5 Identify cluster command

#### 7.6.5.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_IDENTIFY	0x0130
ZBHCI_CMD_ZCL_IDENTIFY_QUERY	0x0131

### 7.6.5.2 Payload (Host)

- 1) ZBHCI\_CMD\_ZCL\_IDENTIFY

ZCLCmdHdr	identifyTime
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
identifyTime	Unsigned 16-bit integer.

示例： 55 01 30 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

- 2) ZBHCI\_CMD\_ZCL\_IDENTIFY\_QUERY

ZCLCmdHdr
n Bytes

示例： 55 01 31 msgLenH msgLenL 00 XX[n Bytes] AA

### 7.6.5.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_ZCL_IDENTIFY_QUERY_RSP	0x8131

### 7.6.5.4 Payload (Slave)

- 1) ZBHCI\_CMD\_ZCL\_IDENTIFY\_QUERY\_RSP

<b>shortAddr</b>	<b>srcEp</b>	<b>timeout</b>
2 Bytes	1 Byte	2 Bytes

<b>Name</b>	<b>Description</b>
shortAddr	The short address of the device.
srcEp	The source endpoint of the device.
timeout	The remaining time.

示例： 55 81 31 00 05 00 XX[2 Bytes] XX XX[2 Bytes] AA

## 7.6.6 On/Off cluster command

### 7.6.6.1 Message Type (Host)

<b>Message Type</b>	<b>Value</b>
ZBHCI_CMD_ZCL_ONOFF_ON	0x0140
ZBHCI_CMD_ZCL_ONOFF_OFF	0x0141
ZBHCI_CMD_ZCL_ONOFF_TOGGLE	0x0142

### 7.6.6.2 Payload (Host)

<b>ZCLCmdHdr</b>
n Bytes

示例：

ZBHCI\_CMD\_ZCL\_ONOFF\_ON

55 01 40 msgLenH msgLenL 00 XX[n Bytes] AA

ZBHCI\_CMD\_ZCL\_ONOFF\_OFF

55 01 41 msgLenH msgLenL 00 XX[n Bytes] AA

ZBHCI\_CMD\_ZCL\_ONOFF\_TOGGLE

55 01 42 msgLenH msgLenL 00 XX[n Bytes] AA

## 7.6.7 Level cluster command

### 7.6.7.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_LEVEL_MOVE2LEVEL	0x0150
ZBHCI_CMD_ZCL_LEVEL_MOVE	0x0151
ZBHCI_CMD_ZCL_LEVEL_STEP	0x0152
ZBHCI_CMD_ZCL_LEVEL_STOP	0x0153
ZBHCI_CMD_ZCL_LEVEL_MOVE2LEVEL_WITHONOFF	0x0154
ZBHCI_CMD_ZCL_LEVEL_MOVE_WITHONOFF	0x0155
ZBHCI_CMD_ZCL_LEVEL_STEP_WITHONOFF	0x0156
ZBHCI_CMD_ZCL_LEVEL_STOP_WITHONOFF	0x0157

### 7.6.7.2 Payload (Host)

- 1) ZBHCI\_CMD\_ZCL\_LEVEL\_MOVE2LEVEL

ZCLCmdHdr	level	transTime
n Bytes	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
level	Level.
transTime	Transition time, 1/10ths of a second.

示例： 55 01 50 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] AA

## 2) ZBHCI\_CMD\_ZCL\_LEVEL\_MOVE

ZCLCmdHdr	mode	rate
n Bytes	1 Byte	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Move mode. 0x00 – Up; 0x01 – Down.
rate	The rate field specifies the rate of movement in units per second.

示例： 55 01 51 msgLenH msgLenL 00 XX[n Bytes] XX XX AA

## 3) ZBHCI\_CMD\_ZCL\_LEVEL\_STEP

ZCLCmdHdr	mode	stepSize	transTime
n Bytes	1 Byte	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Step mode. 0x00 – Up; 0x01 – Down.
stepSize	A step is a change in the current level of 'step size' units.
transTime	The transition time field specifies the time that shall be taken to perform the step, in 1/10ths of a second.

示例： 55 01 52 msgLenH msgLenL 00 XX[n Bytes] XX XX XX[2 Bytes] AA

#### 4) ZBHCI\_CMD\_ZCL\_LEVEL\_STOP

ZCLCmdHdr
n Bytes

示例： 55 01 53 msgLenH msgLenL 00 XX[n Bytes] AA

#### 5) ZBHCI\_CMD\_ZCL\_LEVEL\_MOVE2LEVEL\_WITHONOFF

ZCLCmdHdr	level	transTime
n Bytes	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
level	Level.

Name	Description
transTime	Transition time, 1/10ths of a second.

示例： 55 01 54 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] AA

#### 6) ZBHCI\_CMD\_ZCL\_LEVEL\_MOVE\_WITHONOFF

ZCLCmdHdr	mode	rate
n Bytes	1 Byte	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Move mode. 0x00 – Up; 0x01 – Down.
rate	The rate field specifies the rate of movement in units per second.

示例： 55 01 55 msgLenH msgLenL 00 XX[n Bytes] XX XX AA

#### 7) ZBHCI\_CMD\_ZCL\_LEVEL\_STEP\_WITHONOFF

ZCLCmdHdr	mode	stepSize	transTime
n Bytes	1 Byte	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Step mode. 0x00 – Up; 0x01 – Down.
stepSize	A step is a change in the current level of 'step size' units.
transTime	The transition time field specifies the time that shall be taken to perform the step, in 1/10ths of a second.

示例： 55 01 56 msgLenH msgLenL 00 XX[n Bytes] XX XX XX[2 Bytes] AA

#### 8) ZBHCI\_CMD\_ZCL\_LEVEL\_STOP\_WITHONOFF

ZCLCmdHdr
n Bytes

示例： 55 01 57 msgLenH msgLenL 00 XX[n Bytes] AA

### 7.6.8 Scene cluster command

#### 7.6.8.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_SCENE_ADD	0x0160
ZBHCI_CMD_ZCL_SCENE_VIEW	0x0161
ZBHCI_CMD_ZCL_SCENE_REMOVE	0x0162
ZBHCI_CMD_ZCL_SCENE_REMOVE_ALL	0x0163

Message Type	Value
ZBHCI_CMD_ZCL_SCENE_STORE	0x0164
ZBHCI_CMD_ZCL_SCENE_RECALL	0x0165
ZBHCI_CMD_ZCL_SCENE_GET_MEMBERSHIP	0x0166

### 7.6.8.2 Payload (Host)

#### 1) ZBHCI\_CMD\_ZCL\_SCENE\_ADD

ZCLCmdHdr	groupId	sceneld	transTime	scene NameLen	scene Name	extField Len	extField Sets
n Bytes	2 Bytes	1 Byte	2 Byte	1 Byte	n Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneld	The identifier, unique within this group, which is used to identify this scene.
transTime	The amount of time, in seconds, it will take for the device to change from its current state to the requested scene.
sceneNameLen	Length of scene name.
sceneName	Scene name, char string.
extFieldLen	Length of extFieldSets field.
extFieldSets	The sum of all such defines a scene.

示例：

55 01 60 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX XX[2 Bytes] XX  
 {XX[n Bytes]} XX {XX[n Bytes]} AA

## 2) ZBHCI\_CMD\_ZCL\_SCENE\_VIEW

ZCLCmdHdr	groupId	sceneld
n Bytes	2 Bytes	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneld	The identifier, unique within this group, which is used to identify this scene.

示例： 55 01 61 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

## 3) ZBHCI\_CMD\_ZCL\_SCENE\_REMOVE

ZCLCmdHdr	groupId	sceneld
n Bytes	2 Bytes	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneld	The identifier, unique within this group, which is used to identify this scene.

示例： 55 01 62 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

## 4) ZBHCI\_CMD\_ZCL\_SCENE\_REMOVE\_ALL

<b>ZCLCmdHdr</b>	<b>groupId</b>
n Bytes	2 Bytes

<b>Name</b>	<b>Description</b>
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.

示例： 55 01 63 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

#### 5) ZBHCI\_CMD\_ZCL\_SCENE\_STORE

<b>ZCLCmdHdr</b>	<b>groupId</b>	<b>scenId</b>
n Bytes	2 Bytes	1 Byte

<b>Name</b>	<b>Description</b>
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
scenId	The identifier, unique within this group, which is used to identify this scene.

示例： 55 01 64 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

#### 6) ZBHCI\_CMD\_ZCL\_SCENE\_RECALL

ZCLCmdHdr	groupId	sceneId
n Bytes	2 Bytes	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

示例： 55 01 65 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

#### 7) ZBHCI\_CMD\_ZCL\_SCENE\_GET\_MEMBERSHIP

ZCLCmdHdr	groupId
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.

示例： 55 01 66 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

### 7.6.8.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_ZCL_SCENE_ADD_RSP	0x8160
ZBHCI_CMD_ZCL_SCENE_VIEW_RSP	0x8161
ZBHCI_CMD_ZCL_SCENE_REMOVE_RSP	0x8162
ZBHCI_CMD_ZCL_SCENE_REMOVE_ALL_RSP	0x8163
ZBHCI_CMD_ZCL_SCENE_STORE_RSP	0x8164
ZBHCI_CMD_ZCL_SCENE_GET_MEMBERSHIP_RSP	0x8166

### 7.6.8.4 Payload (Slave)

- 1) ZBHCI\_CMD\_ZCL\_SCENE\_ADD\_RSP

status	groupId	sceneld
1 Byte	2 Bytes	1 Byte

Name	Description
status	SUCCESS, INSUFFICIENT_SPACE or INVALID_FIELD (the group is not present in the group table).
groupId	The group ID for which this scene applies.
sceneld	The identifier, unique within this group, which is used to identify this scene.

示例： 55 81 60 00 04 00 XX XX[2 Bytes] XX AA

- 2) ZBHCI\_CMD\_ZCL\_SCENE\_VIEW\_RSP

<b>status</b>	<b>groupId</b>	<b>sceneld</b>	<b>transTime</b>	<b>sceneName</b>	<b>extFieldSets</b>
1 Byte	2 Bytes	1 Byte	2 Bytes	n Bytes	n Bytes

<b>Name</b>	<b>Description</b>
status	SUCCESS, NOT_FOUND (the scene is not present in the scene table) or INVALID_FIELD (the group is not present in the group table).
groupId	The group ID for which this scene applies.
sceneld	The identifier, unique within this group, which is used to identify this scene.
transTime	Transition time copied from scene table entry.
sceneName	Scene name copied from scene table entry. First byte is the length of the scene name.
extFieldSets	Extension field sets copied from scene table entry. First byte is the length of the extension field sets.

示例：55 81 61 msgLenH msgLenL 00 XX XX[2 Bytes] XX XX[2 Bytes] XX[n Bytes] XX[n Bytes] AA

### 3) ZBHCI\_CMD\_ZCL\_SCENE\_REMOVE\_RSP

<b>status</b>	<b>groupId</b>	<b>sceneld</b>
1 Byte	2 Bytes	1 Byte

<b>Name</b>	<b>Description</b>
status	SUCCESS, NOT_FOUND (the scene is not present in the scene table) or INVALID_FIELD (the group is not present in the group table).
groupId	The group ID for which this scene applies.
sceneld	The identifier, unique within this group, which is used to identify this scene.

示例： 55 81 62 00 04 00 XX XX[2 Bytes] XX AA

4) ZBHCI\_CMD\_ZCL\_SCENE\_REMOVE\_ALL\_RSP

<b>status</b>	<b>groupId</b>
1 Byte	2 Bytes

<b>Name</b>	<b>Description</b>
status	SUCCESS or INVALID_FIELD (the group is not present in the group table).
groupId	The group ID for which this scene applies.

示例： 55 81 63 00 03 00 XX XX[2 Bytes] AA

5) ZBHCI\_CMD\_ZCL\_SCENE\_STORE\_RSP

<b>status</b>	<b>groupId</b>	<b>sceneld</b>
1 Byte	2 Bytes	1 Byte

<b>Name</b>	<b>Description</b>
status	SUCCESS, INSUFFICIENT_SPACE or INVALID_FIELD (the group is not present in the group table).
groupId	The group ID for which this scene applies.
sceneld	The identifier, unique within this group, which is used to identify this scene.

示例： 55 81 64 00 04 00 XX XX[2 Bytes] XX AA

## 6) ZBHCI\_CMD\_ZCL\_SCENE\_GET\_MEMBERSHIP\_RSP

<b>status</b>	<b>capability</b>	<b>groupId</b>	<b>sceneCnt</b>	<b>sceneList</b>
1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

<b>Name</b>	<b>Description</b>
status	SUCCESS or INVALID_FIELD (the group is not present in the group table).
capability	Contain the remaining capacity of the scene table of the device.
groupId	The group ID for which this scene applies.
sceneCnt	The number of scenes contained in the scene list field.
sceneList	Contain the identifiers of all the scenes in the scene table with the corresponding Group ID.

示例： 55 81 66 msgLenH msgLenL 00 XX XX XX[2 Bytes] XX XX[n Bytes] AA

## 7.6.9 OTA cluster command

## 7.6.9.1 Message Type (Host)

<b>Message Type</b>	<b>Value</b>
ZBHCI_CMD_ZCL_OTA_IMAGE_NOTIFY	0x0190

## 7.6.9.2 Payload (Host)

## 1) ZBHCI\_CMD\_ZCL\_OTA\_IMAGE\_NOTIFY

<b>ZCLCmdHdr</b>	<b>payloadType</b>	<b>queryJitter</b>
n Bytes	1 Byte	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
payloadType	0x00 – Query jitter; 0x01 – Query jitter and manufacturer code; 0x02 – Query jitter, manufacturer code, and image type; 0x03 – Query jitter, manufacturer code, image type, and new file version.
queryJitter	By using the parameter, it prevents a single notification of a new OTA upgrade image from flooding the upgrade server with requests from clients.

示例： 55 01 90 msgLenH msgLenL 00 XX[n Bytes] XX XX AA

## 8. 附录：Zigbee 联盟 Pro R21 认证证书

