Telink

# 泰凌微电子（上海）股份有限公司

物联网无线连接解决方案

# Telink Zigbee概述

主讲：杨斌（bin.yang@telink-semi.com）

# 一、芯片介绍

# Telink Zigbee芯片集

## TLSR8

- 8258 - Multimode ULP: Zigbee 3.0 + Bluetooth 5.0
  - 64K SRAM(32K with retention); 512K/1M FLASH
  - TX: up to 10dBm; RX: -99.5dBm@802.15.4 250kbps
  - RX: 5.3mA; TX: 4.8mA@0dBm
- 8278 - Multimode ULP: Zigbee 3.0 + Bluetooth 5.1
  - 64K SRAM(32K with retention); 1M FLASH
  - TX: up to 10dBm; RX: -99.5dBm@802.15.4 250kbps
  - HW accelerator for ECC
  - RX: 4.6mA(DCDC), 9.1mA(LDO)

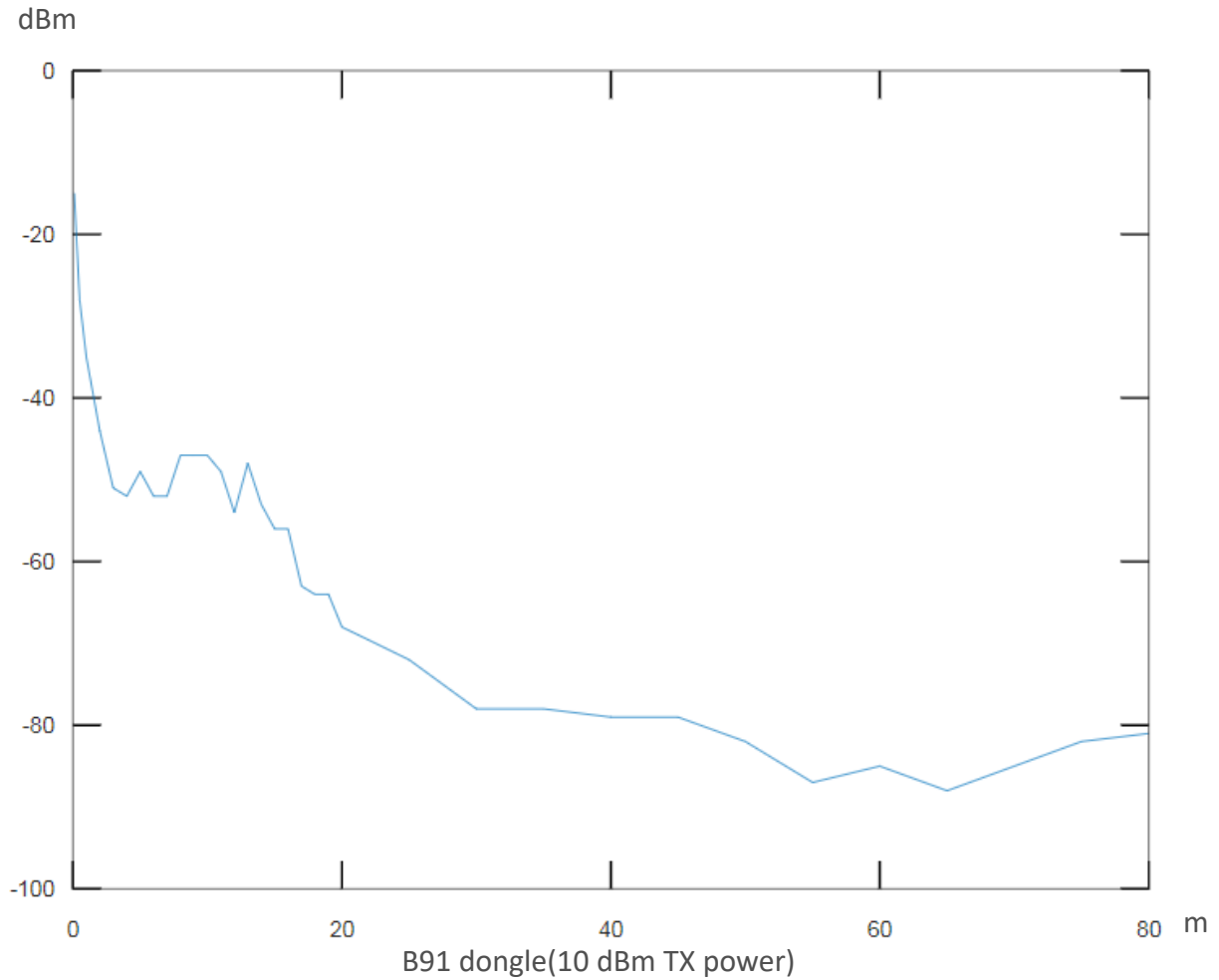
    TX: 4.9mA(DCDC), 9.5mA(LDO) @0dBm

## TLSR9

- 9518 – Multimode ULP, RISC-V IoT: Zigbee 3.0 + Bluetooth 5.2
  - 32-bit RISC-V MCU
  - 256K SRAM(64K with retention)
  - 1M/2M Flash
  - TX: up to 10dBm; RX: -99.5dBm@802.15.4 250kbps
  - Max. 96M operating frequency
  - DSP instruction set, floating-point unit
  - 1.8 - 5.5v
  - RX: 5.2mA(DCDC); TX: 5.3mA(DCDC)@0dBm

# Telink Zigbee距离与RSSI的关系

- 距离与RSSI的关系
  - 11 channel
  - 办公室环境

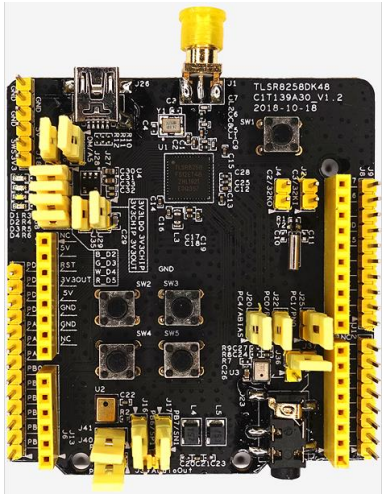


# 二、开发环境

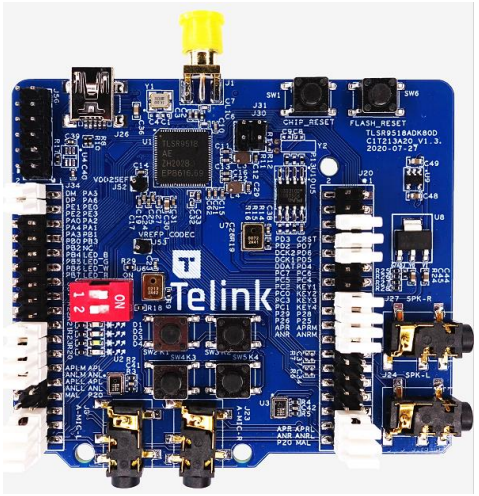
# 硬件

## ➢ 开发板

- EVK Board




8258 Development Board




B91 Development Board

- USB Dongle

8258 Dongle

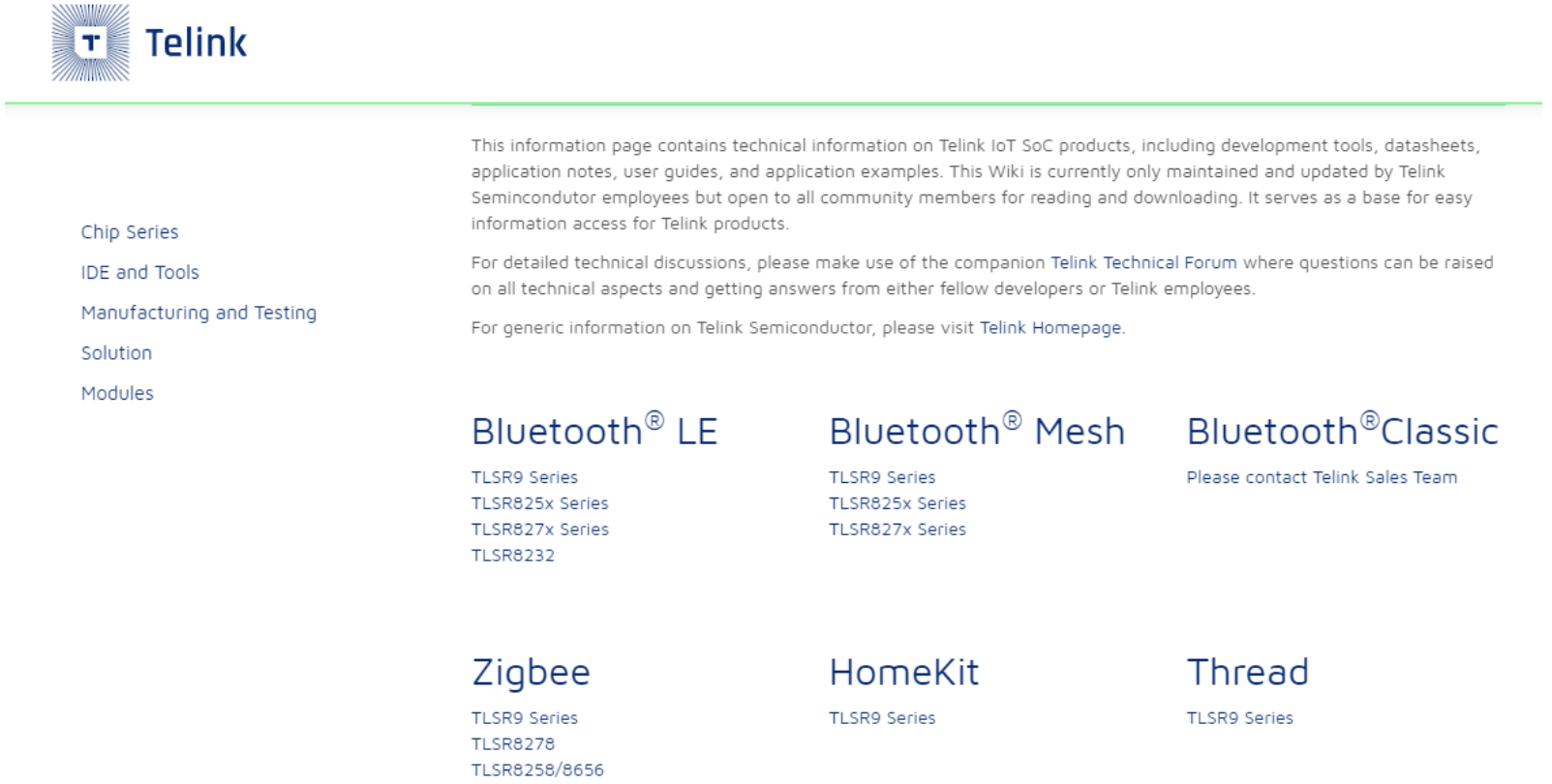
B91 Dongle

# 硬件

- 烧录器(Burning EVK)
  - USB连接PC，LED灯灭后连接成功
    - 免驱动
  - 杜邦线连接DUT
    - 3V3 -> 3V3(DUT)
    - SWM -> SWS(DUT)
    - GND -> GND(DUT)

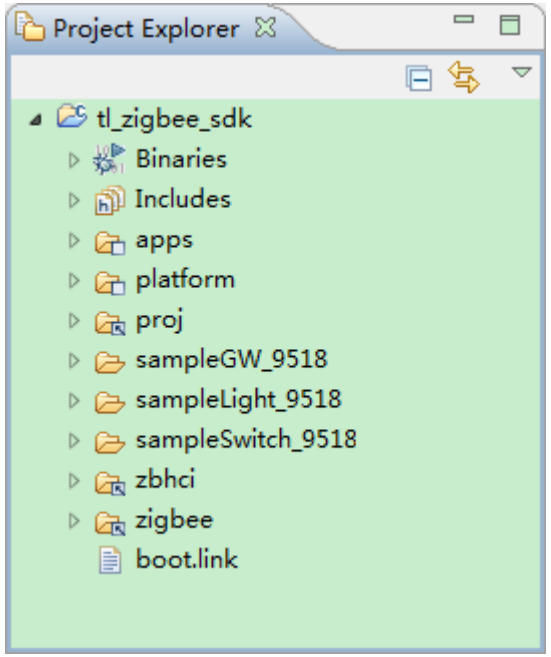
# 软件

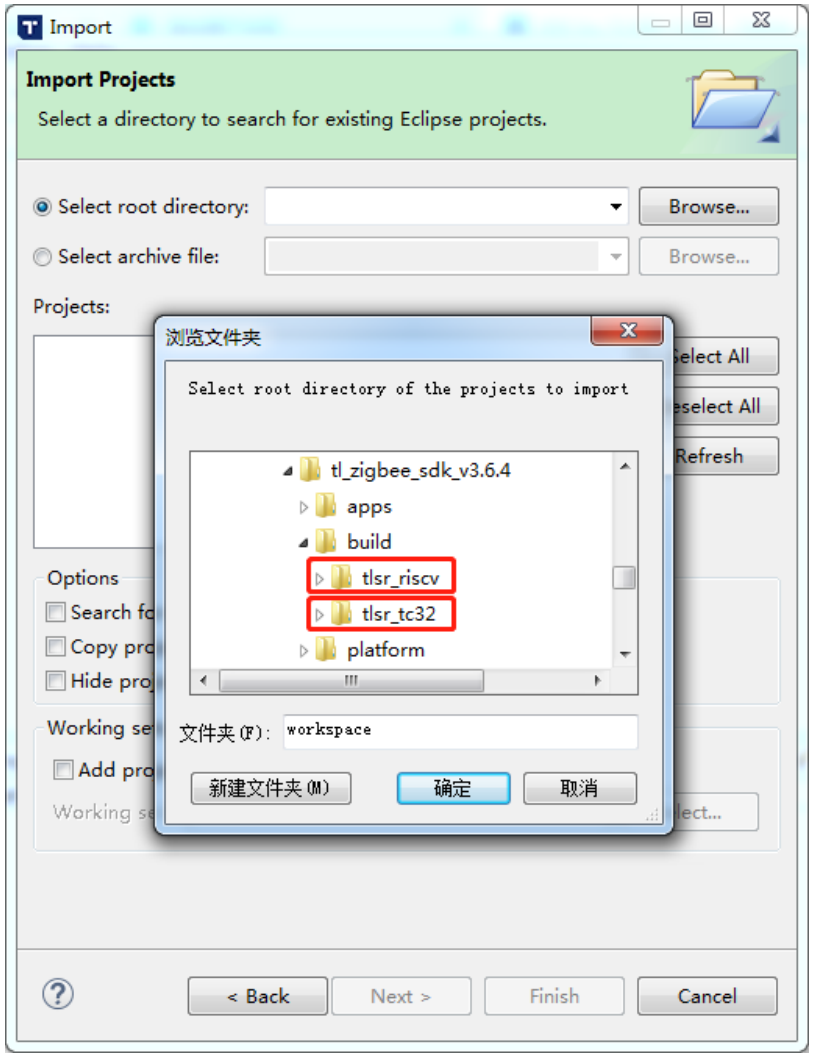
- 下载地址（wiki.telink-semi.cn）
  - 集成开发环境（IDE）
    - IDE for TLSR8 Chips
    - IDE  for TLSR9 Chips
  - 软件开发包（SDK）
    - V3.6.x
  - 烧入调试工具（BDT）
    - Burning and Debugging Tool



# Telink Zigbee SDK

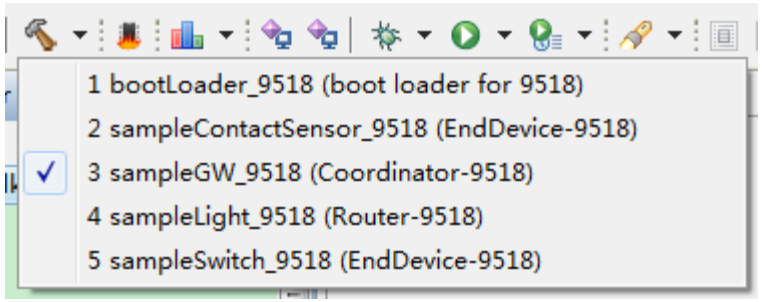
- 工程导入
  - 打开IDE
  - 菜单界面File->Import->Existing Projects into Workspace
  - 选择tl_zigbee_sdk/build
    - TLSR8选择tlsr_tc32
    - TLSR9选择tlsr_riscv
- 目录结构
  - /apps：应用程序目录
  - /platform：平台目录
  - /proj：工程目录
  - /zigbee：协议栈目录
  - /zbhci：hci命令处理目录

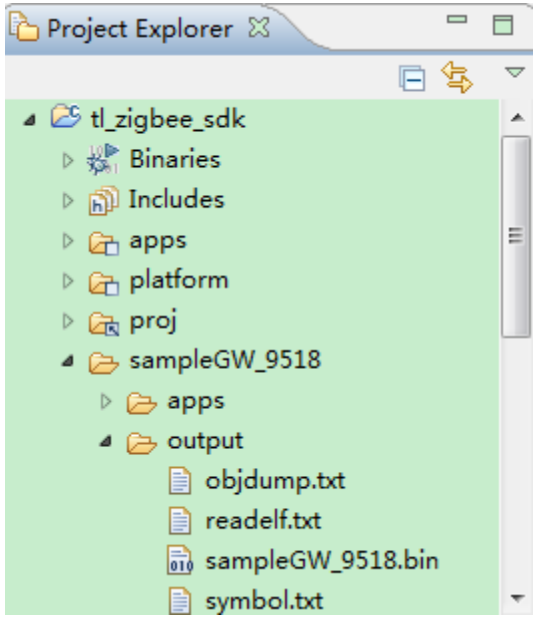


# Telink Zigbee SDK

- 工程编译
  - sampleGW_xx
  - sampleLight_xx
  - sampleSwitch_xx
  - sampleContactSensor_xx
- 编译输出文件
  - 烧录文件（xx.bin）
  - mapping文件（xx.lst / objdump.txt）

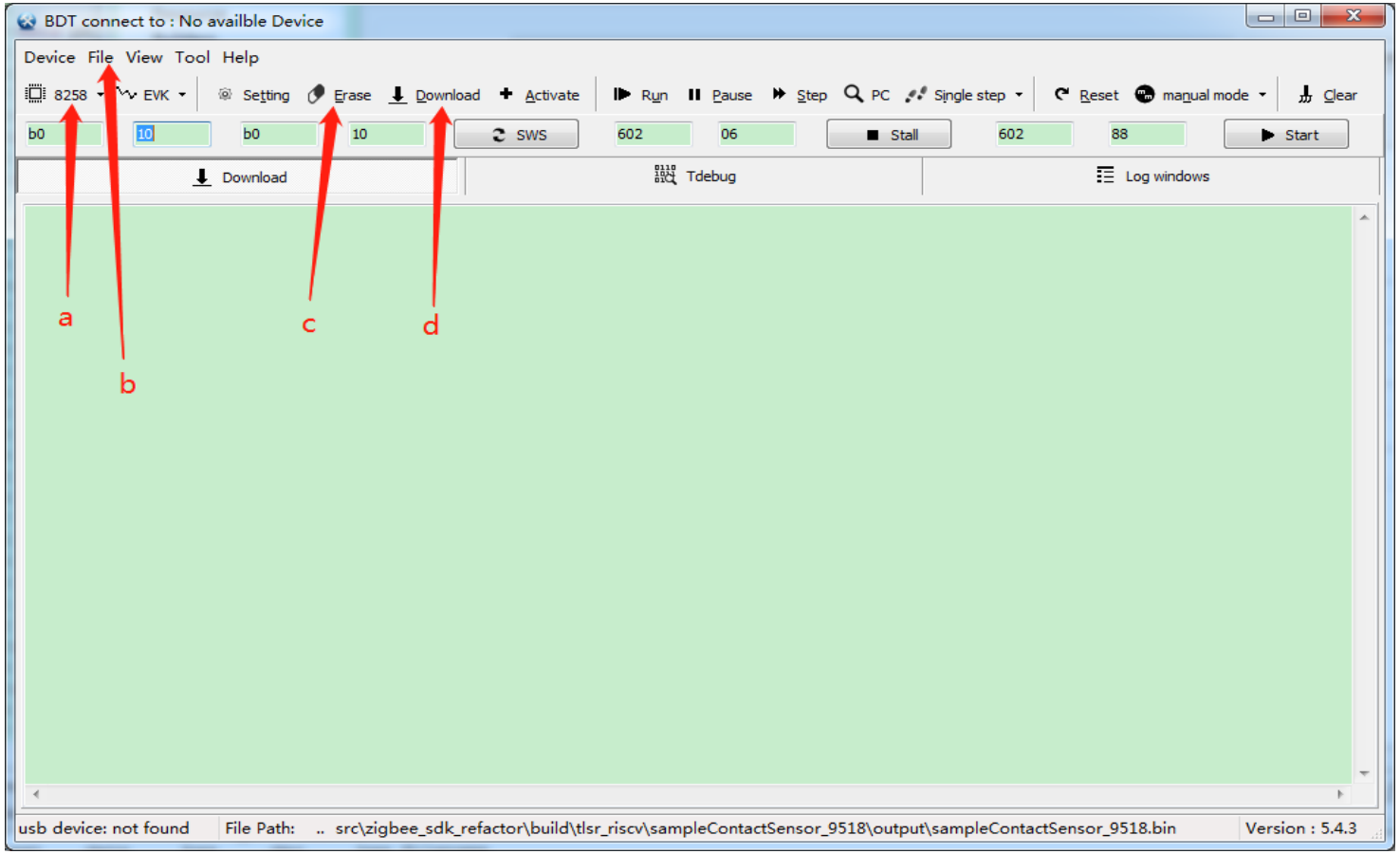


# Telink Zigbee SDK

## ➢ 固件烧录

- Single wire连接
- 使用BDT工具烧写固件
  - a.选择芯片类型
  - b.选择固件
  - c.擦除FLASH
  - d.下载固件



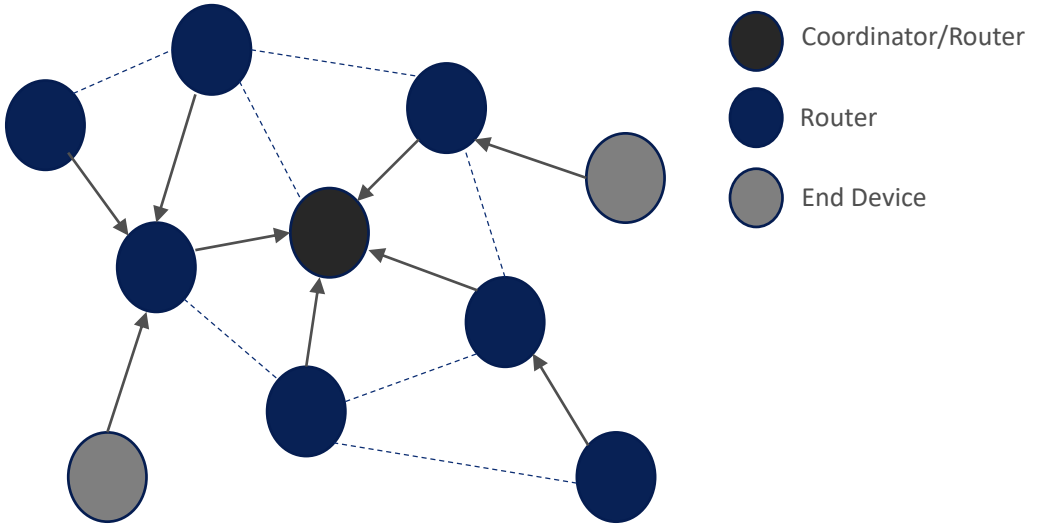
# 三、SDK开发

# 基本概念 – 设备类型

- 协调器（Coordinator）
  - 创建集中式网络（Central network）
  - 信任中心（Trust Center）
  - 网络管理中心（Network Manager）
  - 路由功能
- 路由节点（Router）
  - 创建分布式网络（Distribute network）
  - 路由功能
- 终端节点（End Device）
  - 低功耗

# 基本概念 – 网络类型

- 网络拓扑
  - Mesh网络
  - 终端设备收发数据都要经过父节点
- 网络类型
  - 集中式网络
    - 协调器创建的网络
  - 分布式网络
    - 路由节点创建的网络
    - TouchLink方式创建的网络



# 基本概念 – 协议规范

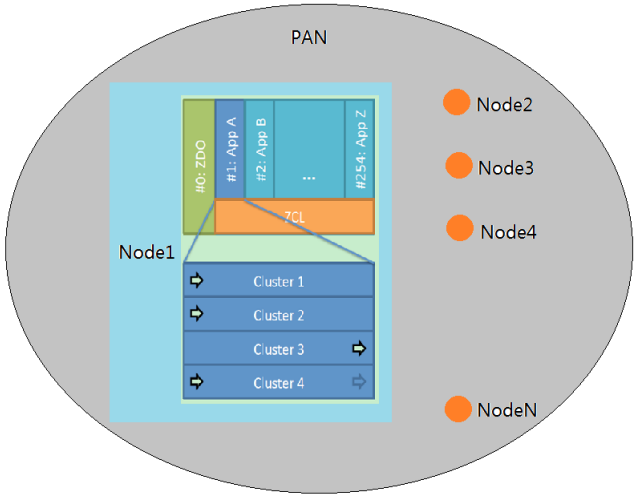
- **PAN ID**：个域网标识，一个Zigbee网络有且只有一个PAN ID。
- **Channel**：Zigbee的工作信道N为11~26，工作频点与信道的关系为：2405+(N-11)*5 MHz。
- **Node**：一个物理设备，具有全球唯一的IEEE地址，入网后在该PAN ID网络中具有唯一的短地址。
- **Endpoint**：应用端口，一个Node可以包含多个Endpoint。

  0 ：ZDO端口。

  1~240 ：APP端口。

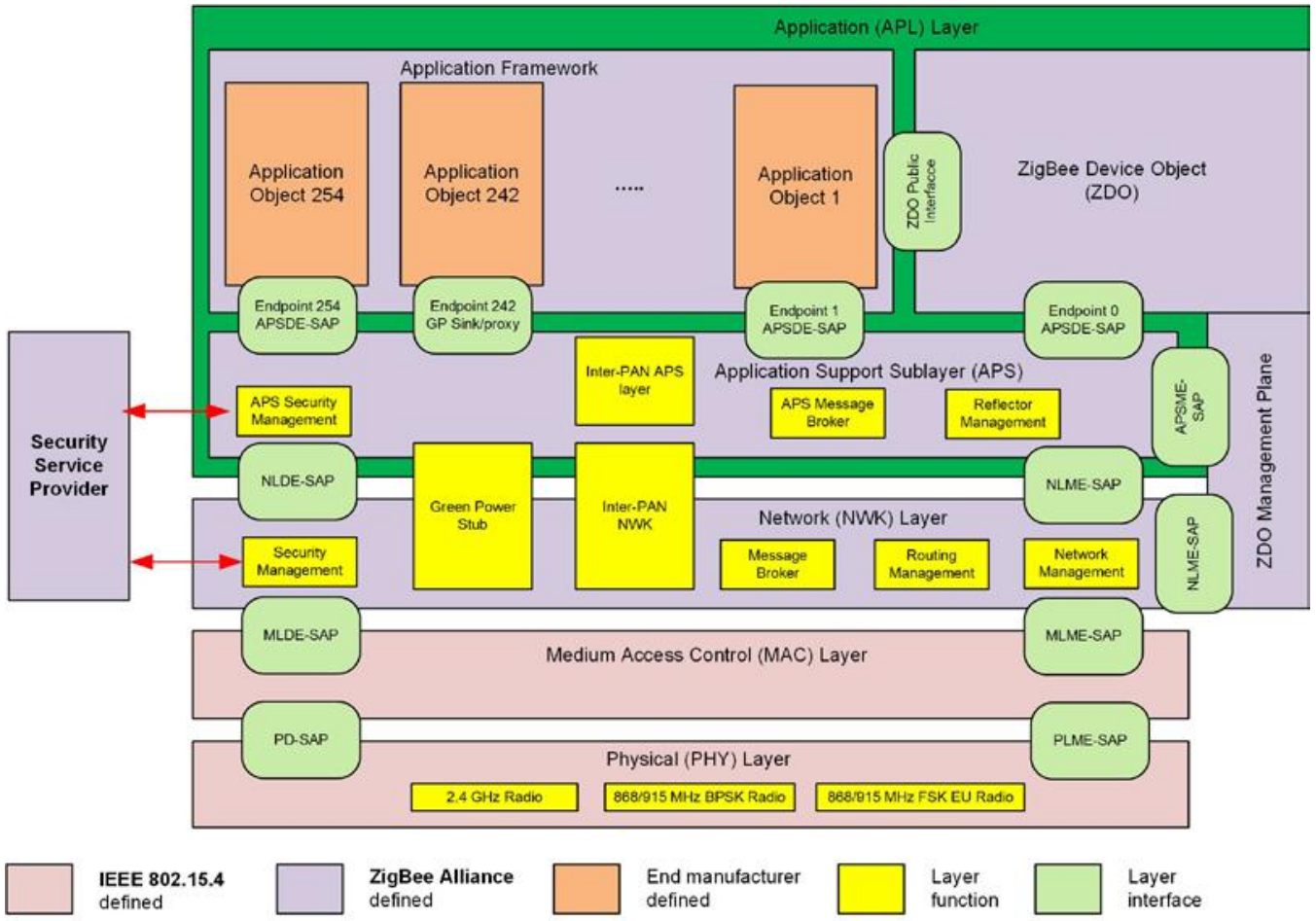
  240~254：协议使用的端口，APP不可用。如242用于Green Power。

  255 ：广播端口。
- **Cluster**：一个具体应用是由多个不同cluster组成，从而完成不同的行为。ZCL定义了一系列cluster需要遵循的行为规范。
- **Attribute**：ZCL除了定义了属性的值，还定义了属性值的操作权限。



# 基本概念 – 协议框架

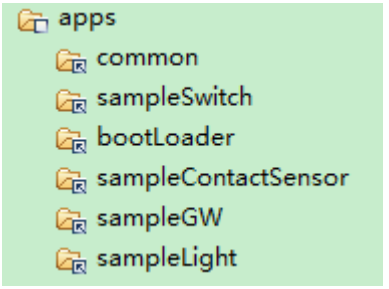
- 协议框架
  - AF
  - APS
  - ZDO
  - NWK
  - MAC
  - PHY



# 目录结构

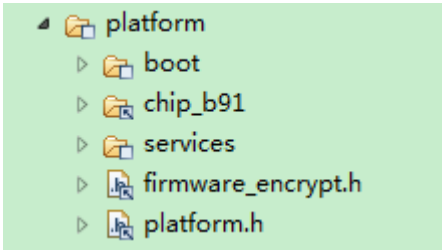
## ➢ 应用程序目录

- /apps
  - /common 通用函数目录
    - ◆ main.c 主函数
    - ◆ comm_cfg.h 版本定义和运行模式配置文件
  - /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
  - /bootLoader 引导程序



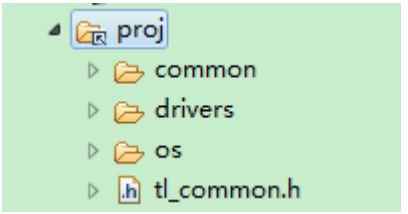
# 目录结构

- 平台驱动目录




  - /platform
    - /boot 启动文件
    - /chip_xxxx 芯片驱动文件
    - /services 中断处理文件
    - platform.h 头文件
    - firmware_encrypt.h 基于FLASH UID的加密方案

- 工程目录



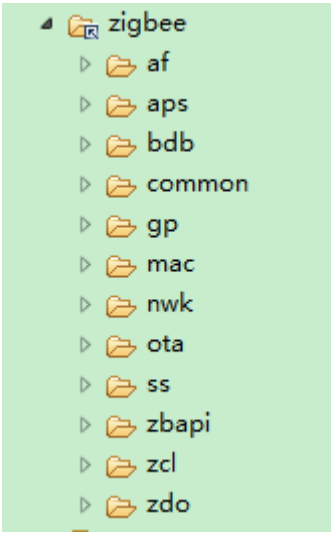

  - /proj
    - /common 通用函数文件

      如字符串、断言等
    - /drivers 抽象层驱动文件
    - /os 任务管理文件
    - tl_common.h 头文件

# 目录结构

## Zigbee协议栈目录

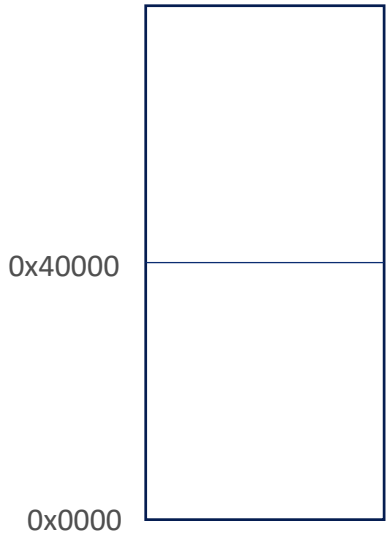
- /zigbee
  - /af 应用框架层文件
  - /aps 应用支持子层文件
  - /bdb 基本设备行为文件
  - /common 协议配置文件
  - /gp 绿色能源文件
  - /mac 媒介控制层文件
  - /nwk 网络层文件
  - /ota 空中升级处理文件
  - /ss 安全服务文件
  - /zbapi 常用接口文件
  - /zcl ZCL层文件
  - /zdo 设备对象层文件



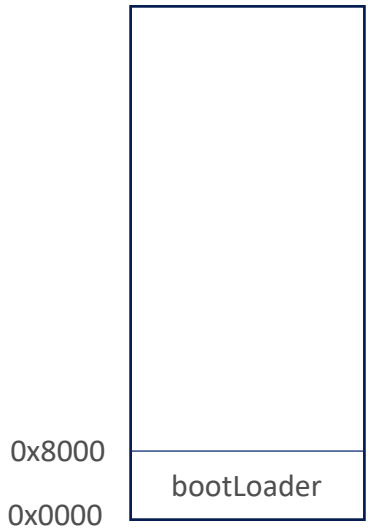
# 开发指南 – 启动模式

## ➢ 运行模式选择

- 支持两种运行模式
  - 多地址启动模式（0x0、0x40000）
    - ◆ 优点：启动快，OTA成功后不需要搬运image
    - ◆ 缺点：固件只能放在0x0或0x40000地址，造成flash空间分配的不连续性
  - BootLoader启动模式
    - ◆ 优点：固件位置可以随意指定，灵活性强
    - ◆ 缺点：消耗bootloader代码空间；启动慢，OTA成功后需要搬运image
- 修改方法（comm_cfg.h）
  - #define BOOT_LOADER_MODE　0//1



多地址启动模式



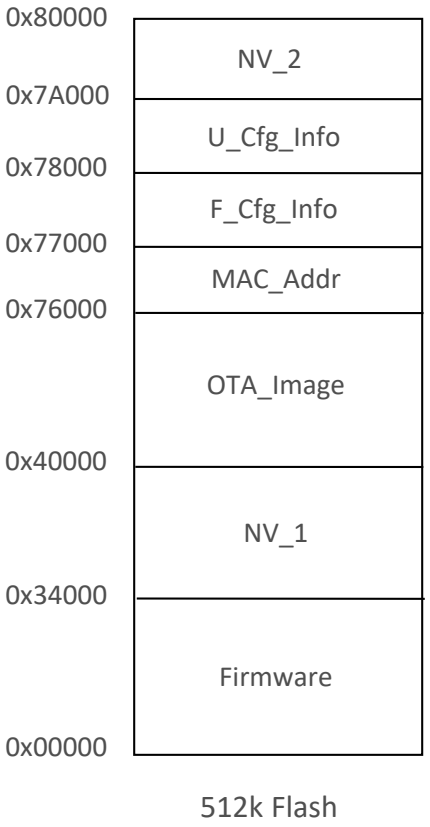
BootLoader启动模式

# 开发指南 – 硬件选择

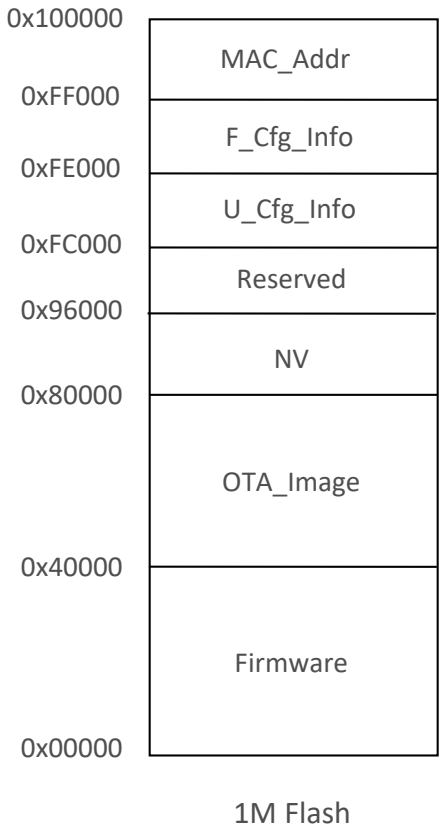
- 芯片类型选择
  - 修改方法（version_cfg.h）
    - #define CHIP_TYPE　TLSR_9518
- 目标板选择
  - 修改方法（app_cfg.h）
    - #define BOARD　BOARD_9518_DONGLE// BOARD_9518_EVK

# 开发指南 – Flash分布

- 多地址启动模式Flash分布



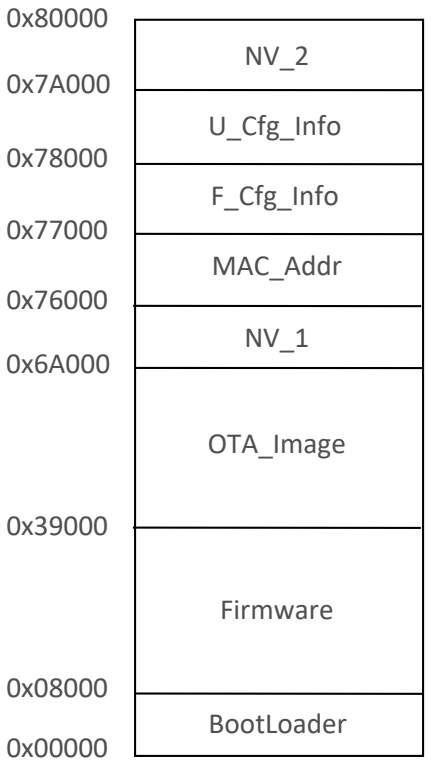

512k Flash



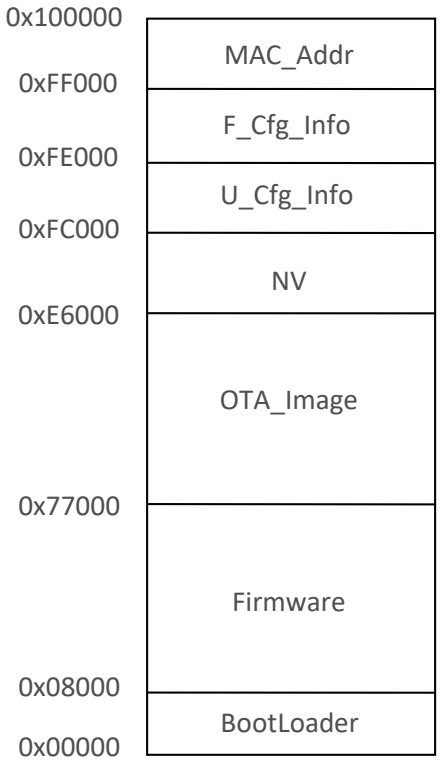

1M Flash

# 开发指南 – Flash分布

- bootLoader启动模式Flash分布




512k Flash




1M Flash

# 开发指南 – Flash分布

## Flash分布说明

- MAC_Addr：该区域起始的8个Bytes存放着MAC地址，芯片出厂时会预写，请谨慎擦除。

  系统启动后会检查MAC地址信息，如果发现8个Bytes是全0xFF，将会随机生成一个MAC地址回填到该区域。

- 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，比如频偏校准、ADC校准等，请谨慎擦除。

- 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。
- NV(NV_1,NV_2)：网络信息存储区，重启后，从该区域读取并恢复网络参数。
- Firmware和OTA_Image：固件存放区域。
- BootLoader：使用bootLoader启动模式时，存放引导程序代码。

# 开发指南 – 打印输出

- 打印调试

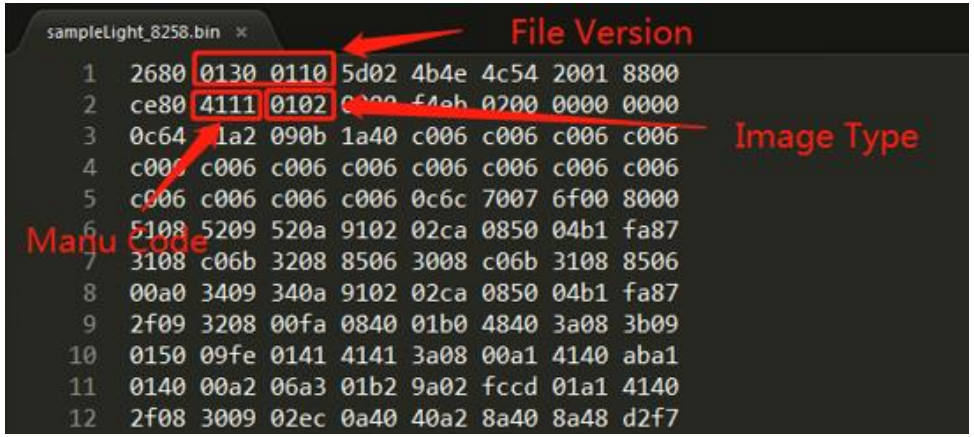
GPIO模拟串口TX功能，使用printf()函数实现打印输出。用户可以任意更改成合适的I/O口，不占用硬件串口资源。

- 打印使能（app_cfg.h）
  - #define UART_PRINTF_MODE　1
- 打印口配置（board_xx.h）
  - #define DEBUG_INFO_TX_PIN　GPIO_PC4
  - #define BAUDRATE　1000000//1M

# 开发指南 – 版本管理

➢ APP版本管理

每个demo目录下都对应有一个version_cfg.h文件用来管理App应用的版本，在OTA时可以有效的防止因固件与芯片不匹配而造成升级变砖的问题。




- 制造商代码
  - #define MANUFACTURER_CODE_TELINK 0x1141
- 固件类型
  - #define IMAGE_TYPE ((CHIP_TYPE << 8) | IMAGE_TYPE_SWITCH)
- 文件版本
  - #define FILE_VERSION ((APP_RELEASE << 24) | (APP_BUILD << 16) | (STACK_RELEASE << 8) | STACK_BUILD)

# 开发指南 – 存储管理

- 动态内存管理
  - 内存申请
    - ev_buf_allocate()
  - 内存释放
    - ev_buf_free()
  - 使用数组实现的内存池管理，默认最大支持142字节长度的内存申请。

    用户可以自行修改内存池的相关配置（ev_buffer.h）。

# 开发指南 – 存储管理

## ➢ NV管理

- NV_MODULE
  - 每个模块占用2个或（2*n）个sectors（1 sector = 4k flash）
  - 采用追加写入的方法，直到1个sector写满后，将有效的信息搬到另一个sector后，再将之前的sector擦除
- NV_ITEM
  - 每个条目就是一个数据存储单元
  - 一个模块可以由很多个条目组成
- 格式
  - sector info + item索引 + item内容
- API接口
  - nv_flashWriteNew()
  - nv_flashReadNew()

```
/**********************************************************************************************
 * Store zigbee information in flash.
 *      Module ID                |           512K Flash                |           1M Flash                |
 * -----------------------------|-------------------------------------|-----------------------------------|
 * NV_MODULE_ZB_INFO             |       0x34000 - 0x36000             |       0x80000 - 0x82000           |
 * NV_MODULE_ADDRESS_TABLE       |       0x36000 - 0x38000             |       0x82000 - 0x84000           |
 * NV_MODULE_APS                 |       0x38000 - 0x3a000             |       0x84000 - 0x86000           |
 * NV_MODULE_ZCL                 |       0x3a000 - 0x3c000             |       0x86000 - 0x88000           |
 * NV_MODULE_NWK_FRAME_COUNT     |       0x3c000 - 0x3e000             |       0x88000 - 0x8a000           |
 * NV_MODULE_OTA                 |       0x3e000 - 0x40000             |       0x8a000 - 0x8c000           |
 * NV_MODULE_APP                 |       0x7a000 - 0x7c000             |       0x8c000 - 0x8e000           |
 * NV_MODULE_KEYPAIR             |       0x7c000 - 0x80000             |       0x8e000 - 0x96000           |
 *                               |   *16K - can store 127 nodes        |   *32K - can store 302 nodes      |
 * NV_MAX_MODULS
 */
```

# 开发指南 – 任务管理

- 单次任务
  - API接口
    - 注册 TL_SCHEDULE_TASK(tl_zb_callback_t func, void *arg)
  - 使用注意
    - 只执行一次，没有优先级
    - 避免一次性压入过多任务导致任务队列溢出
- 常驻任务
  - API接口
    - 注册 ev_on_poll(ev_poll_e e, ev_poll_callback_t cb)
    - 启动 ev_enable_poll(ev_poll_e e, ev_poll_callback_t cb)
    - 暂停 ev_disable_poll(ev_poll_e e, ev_poll_callback_t cb)
  - 启动后，在主循环中一直执行

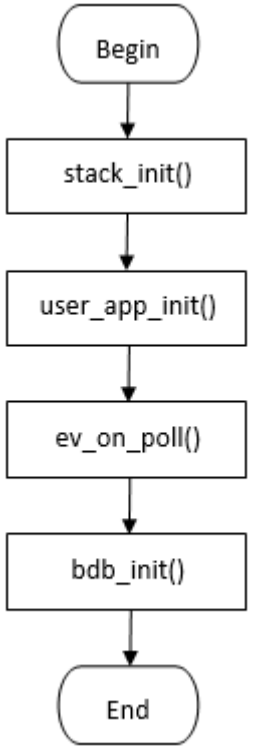
# 开发指南 – 任务管理

## ➢ 软件定时任务

- API接口
  - 注册 TL_ZB_TIMER_SCHEDULE(ev_timer_callback_t func, void *arg, u32 t_ms)
  - 取消 TL_ZB_TIMER_CANCEL(ev_timer_event_t **evt)
- 使用注意
  - 在中断函数中避免使用TL_ZB_TIMER_CANCEL()
  - 在回调函数中
    - 不可以使用TL_ZB_TIMER_CANCEL()取消自身的任务
    - 返回值小于0，表示该任务只执行一次，执行后自动注销
    - 返回值等于0，表示该任务执行后，仍然使用注册时的时间参数来启动定时任务
    - 返回值大于0，表示该任务执行后，使用该返回值作为新的时间参数来启动定时任务

# 开发指南 – 初始化流程

- user_init() 应用层初始化流程
  - stack_init()
    - zb_init() 协议栈初始化
    - zb_zdoCbRegister() 注册协议栈回调函数
  - user_app_init()
    - af_endpointRegister() 注册端口描述符信息和消息处理回调
    - zcl_init() 初始化ZCL基础命令
    - zcl_register() 注册应用层需要的cluster处理函数
  - ev_on_poll() 注册用户轮询事件
  - bdb_init() 初始化并启动BDB流程



# 开发指南 – 初始化流程

- void zb_zdoCbRegister(zdo_appIndCb_t *cb)
  注册协议栈回调函数
  - 回调函数
    - zdpStartDevCnfCb
    - zdpResetCnfCb
    - zdpDevAnnounceIndCb
    - zdpLeaveIndCb
    - zdpLeaveCnfCb
    - zdpNwkUpdateIndCb
    - zdpPermitJoinIndCb
    - zdoNlmeSyncCnfCb
    - zdoTcJoinIndCb

```
const zdo_appIndCb_t appCbLst = {
    bdb_zdoStartDevCnf,
    NULL,
    sampleGW_devAnnHandler,
    sampleGW_leaveIndHandler,
    sampleGW_leaveCnfHandler,
    sampleGW_nwkUpdateIndicateHandler,
    NULL,
    NULL,
    sampleGW_tcJoinIndHandler,
};
```

# 开发指南 – 初始化流程

- af_endpointRegister() 注册端口描述符信息和消息处理回调
  - u8 ep，应用端口
  - af_simple_descriptor_t *simple_desc，端口描述符信息
  - af_endpoint_cb_t rx_cb，接收消息的回调函数，通常注册的是zcl_rx_handler()
  - af_dataCnf_cb_t cnf_cb，发送消息的确认回调函数

```
typedef struct{
    u16 app_profile_id;          //APP profile ID specifies the profile which supported on this EP.
    u16 app_dev_id;              //APP DEV ID specifies the device description supported on this EP.

    u8  endpoint;                //end-point num of the simple descriptor 1 ~ 240
    u8  app_dev_ver:4;           //APP DEV version specifies the version of the device description supported
    u8  reserved:4;              //Reserved
    u8  app_in_cluster_count;    //The number of input clusters supported on this EP
    u8  app_out_cluster_count;   //The number of output clusters supported on this EP

    u16 *app_in_cluster_lst;     //Input cluster list address
    u16 *app_out_cluster_lst;    //Output cluster list address
}af_simple_descriptor_t;
```

# 开发指南 – 初始化流程

- zcl_register() 注册应用层需要的cluster处理函数
  - u8 endpoint，应用端口
  - u8 clusterNum，支持的cluster数量
  - zcl_specClusterInfo_t *info，clusters和attributes注册表
    - u16 clusterId，cluster ID
    - u16 manuCode，厂商代码
    - u16 attrNum，该cluster所支持的属性数量
    - const zclAttrInfo *attrTbl，属性表
    - cluster_registerFunc_t func，cluster注册函数指针
    - cluster_forAppCb_t appCb，cluster消息回调函数

```
typedef struct {
    u16 clusterId;
    u16 manuCode;
    u16 attrNum;
    const zclAttrInfo_t      *attrTbl;
    cluster_registerFunc_t   clusterRegisterFunc;
    cluster_forAppCb_t       clusterAppCb;
} zcl_specClusterInfo_t;
```

# 开发指南 – 初始化流程

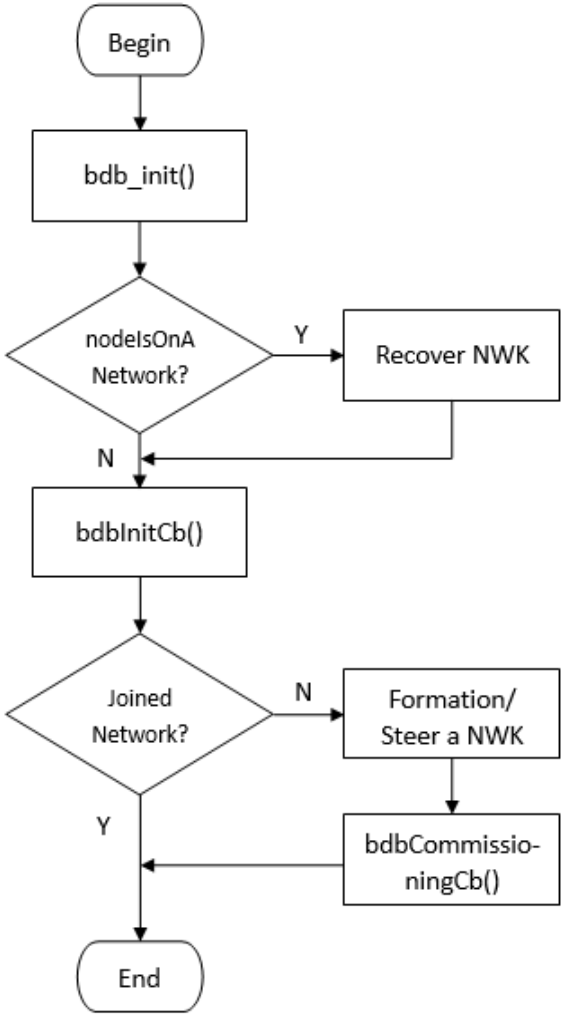
- zcl_specClusterInfo_t *info，clusters和attributes注册表

```
/**
 *  @brief Definition for simple GW ZCL specific cluster
 */
zcl_specClusterInfo_t g_sampleGwClusterList[] =
{
    {ZCL_CLUSTER_GEN_BASIC,                 ZCL_BASIC_ATTR_NUM,     basic_attrTbl,      zcl_basic_register,     sampleGW_basicCb},
    {ZCL_CLUSTER_GEN_IDENTIFY,              ZCL_IDENTIFY_ATTR_NUM,  identify_attrTbl,   zcl_identify_register,  sampleGW_identifyCb},
    {ZCL_CLUSTER_GEN_GROUPS,                0,                      NULL,               zcl_group_register,     sampleGW_groupCb},
    {ZCL_CLUSTER_GEN_SCENES,                0,                      NULL,               zcl_scene_register,     sampleGW_sceneCb},
#ifdef ZCL_DOOR_LOCK
    {ZCL_CLUSTER_CLOSURES_DOOR_LOCK,        0,                      NULL,               zcl_doorLock_register,  &sampleGW_doorLockCb},
#endif
#ifdef ZCL_TEMPERATURE_MEASUREMENT
    {ZCL_CLUSTER_MS_TEMPERATURE_MEASUREMENT,    0,                  NULL,               zcl_temperature_measurement_register,   NULL},
#endif
#ifdef ZCL_IAS_ZONE
    {ZCL_CLUSTER_SS_IAS_ZONE,               0,                      NULL,               zcl_iasZone_register,   &sampleGW_iasZoneCb},
#endif
#ifdef ZCL_POLL_CTRL
    {ZCL_CLUSTER_GEN_POLL_CONTROL,          0,                      NULL,               zcl_pollCtrl_register,  &sampleGW_pollCtrlCb},
#endif
};
```

# 开发指南 – BDB流程

## ➢ BDB流程

- bdb_init() 初始化会检查是否已经加入过网络
  - 如果是，首先恢复网络，然后回调到bdbInitCb()
  - 如果不是，将回调到bdbInitCb()，由用户决定下一步的动作
- bdbInitCb() 用户通过该函数可以获取到bdb初始化的结果，并决定下一步的行为，例如创建网络或加入网络。
  - bdb_networkFormationStart() 创建一个网络
  - bdb_networkSteerStart() 搜索并加入一个网络
- bdbCommissioningCb() 用户通过该函数获取到commissioning的结果。



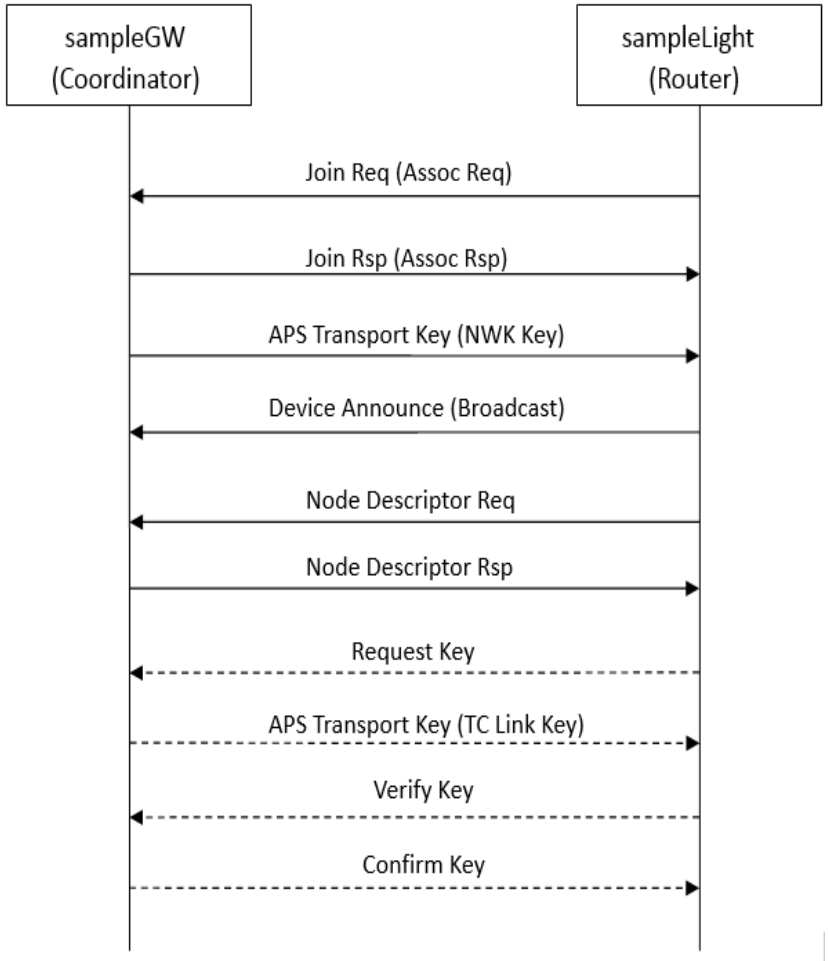
# 开发指南 – 网络安全

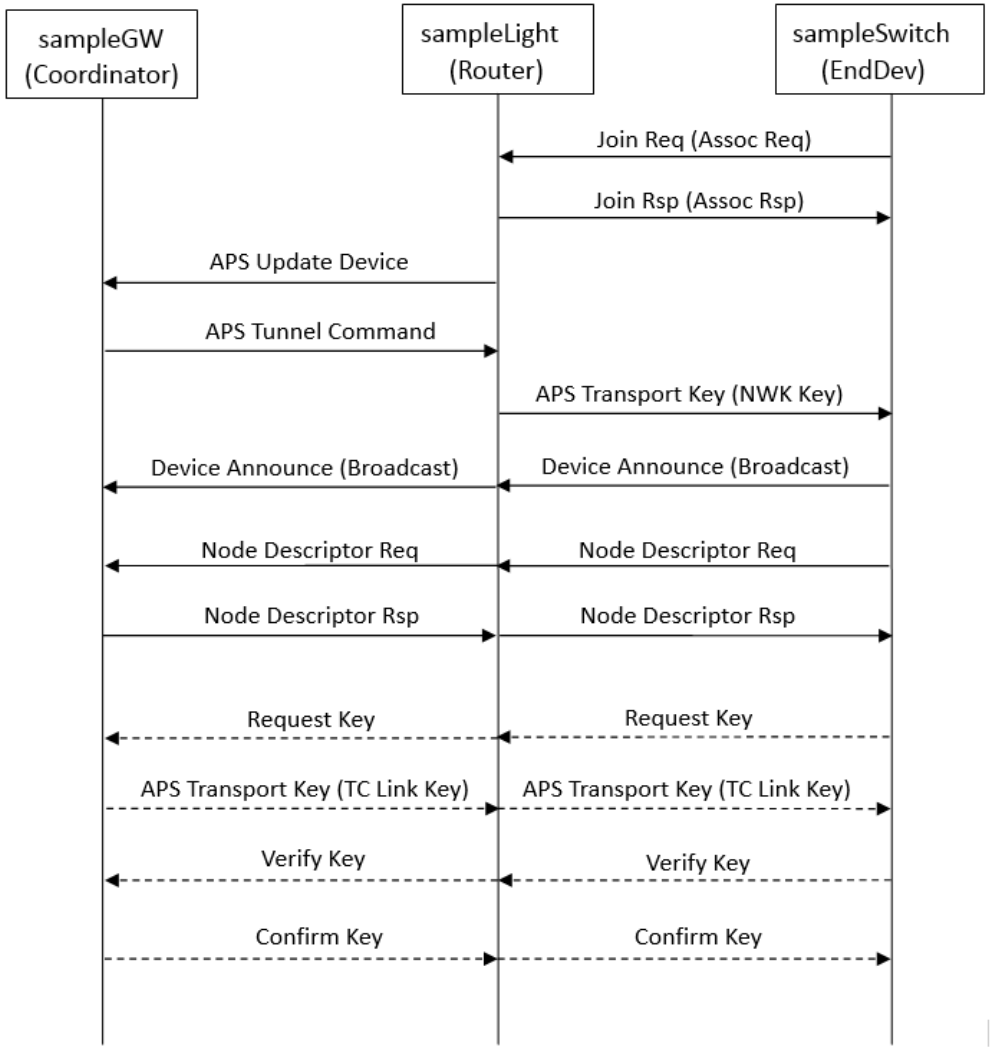
## ➢ 网络安全

- Default TCLK 联盟定义的Global Trust Center Link Key
  - {5a6967426565416c6c69616e63653039}
- Install Code 使用MMO-Hash算法衍生出Unique Trust Center Link Key
  - zb_pre_install_code_load()，节点从FLASH读取install code
  - zb_pre_install_code_store()，网关调用该接口将节点的MAC地址和install code写入网关
- Pre-configured NWK Key 预设网络密钥
  - zb_preConfigNwkKey()

# 开发指南 – 入网流程

- 入网流程图



# 开发指南 – 数据收发

- 数据发送
  - 使用ZCL命令接口
    - read/write/report等基础命令
    - cluster命令
  - 使用AF Data发送接口
    - af_dataSend()
      - u8 srcEp，源端口
      - epInfo_t *pDstEpInfo，目标端口信息
      - u16 clusterId，cluster ID
      - u16 payloadLen，数据长度
      - u8 *payload，消息地址
      - u8 *apsCnt，当前消息的aps counter
- 数据接收
  - 使用af_endpointRegister()注册消息接收处理函数
    - zcl_rx_handler(void *pData)

```
typedef struct apsdeDataInd_s{
    aps_data_ind_t indInfo;
    u16 asduLen;
    u8  asdu[];
}apsdeDataInd_t;
```

# 开发指南 – 低功耗管理

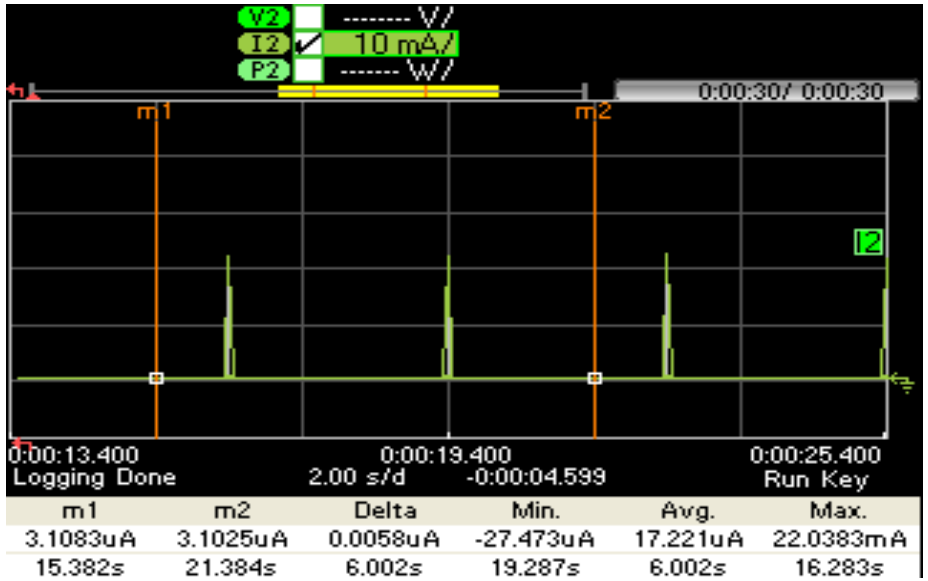
## ➢ 低功耗管理

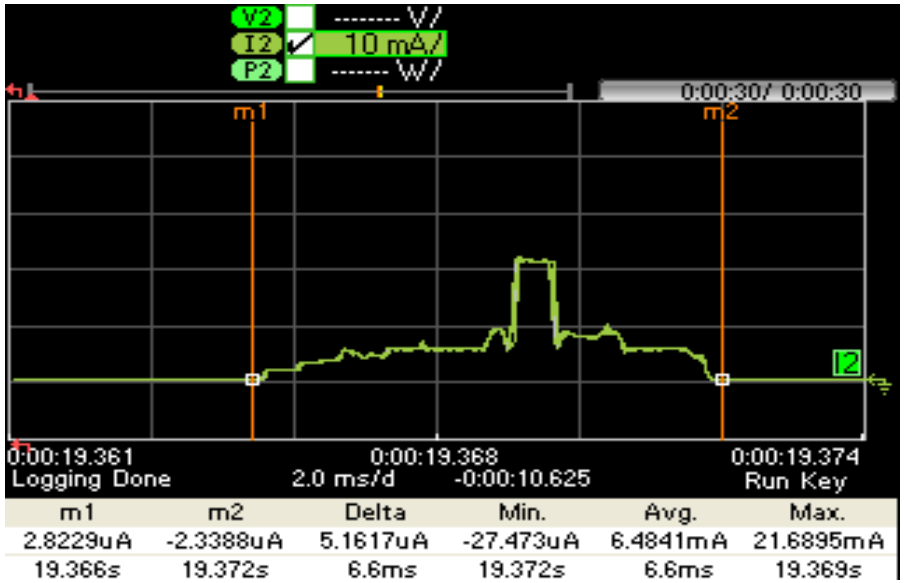
- 休眠接口
  - drv_pm_lowPowerEnter()
- 两级休眠
  - suspend/deep with retention模式（有定时任务）
  - deep模式（无定时任务）
- 唤醒方式
  - Timer唤醒
    - ◆ 配合软件定时任务使用
  - 按键唤醒
    - ◆ drv_pm_wakeupPinConfig()

# 开发指南 – 低功耗管理

- 8258 dongle，deep sleep with 32k ram retention

| poll rate(s) | 1 | 3 | 5 | 10 |
|---|---|---|---|---|
| sampling time(s) | 6 | 6 | 15 | 20 |
| wakeup time(ms) | 6.9 | 6.6 | 6.6 | 7 |
| avg current(uA) | 48.45 | 17.22 | 12.04 | 7.64 |

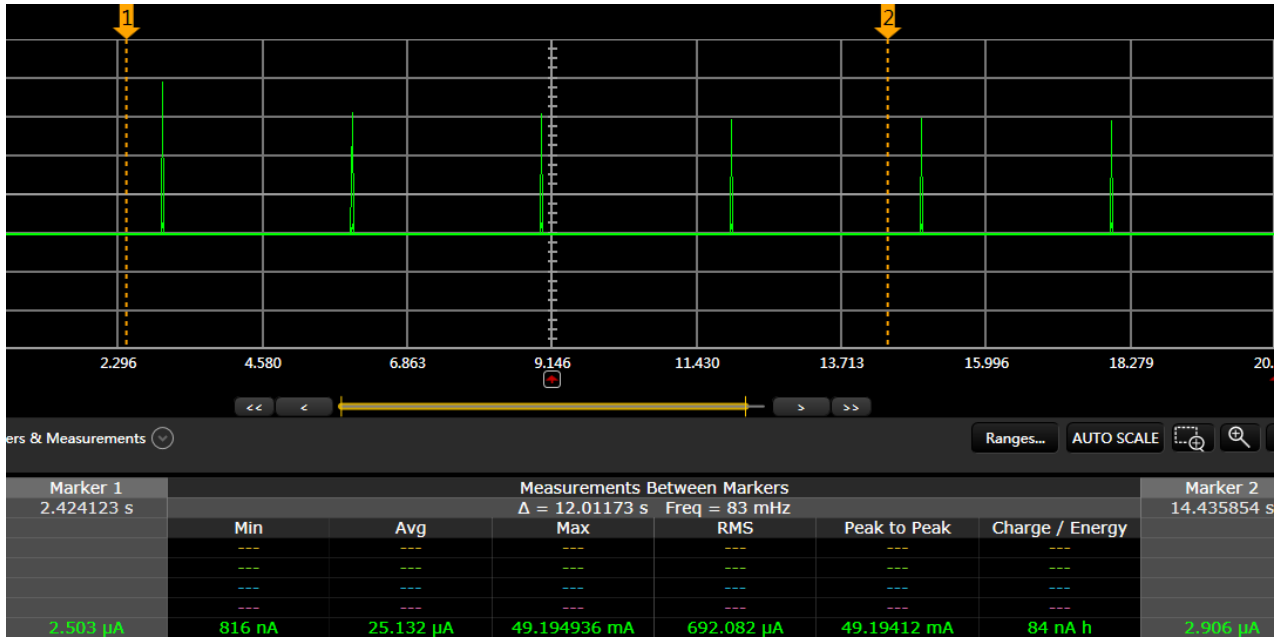


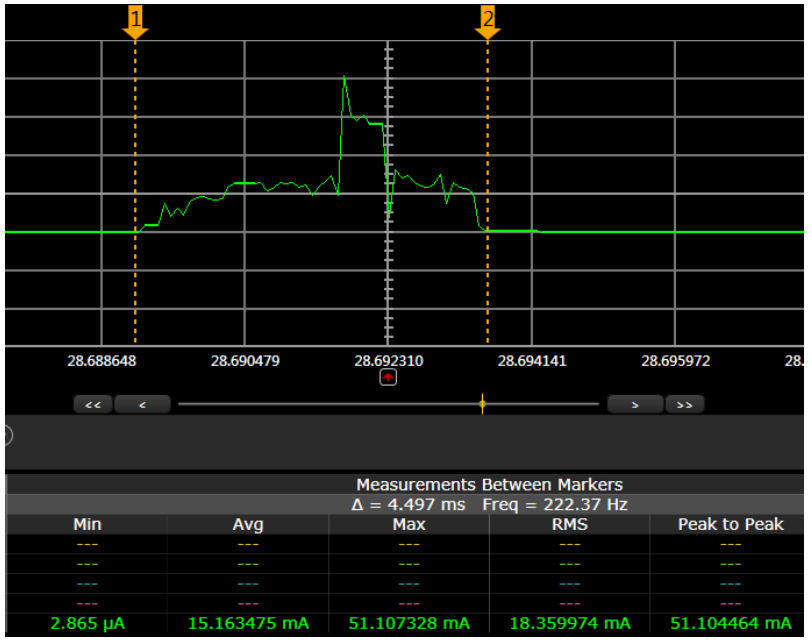



# 开发指南 – 低功耗管理

- B91 dongle，deep sleep with 64k ram retention

| poll rate(s) | 1 | 3 | 10 |
|---|---|---|---|
| sampling time(s) | 6 | 12 | 20 |
| wakeup time(ms) | 4.2 | 4.5 | 4.5 |
| avg current(uA) | 64.8 | 25.1 | 10.1 |



# 开发指南 – 网络参数配置

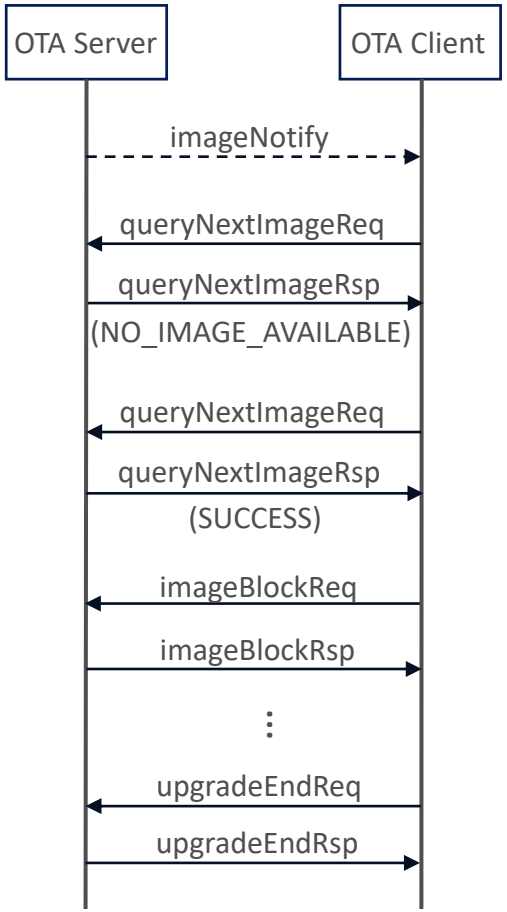
## ➢ 网络参数配置

- APS_BINDING_TABLE_NUM，绑定表个数
- APS_GROUP_TABLE_NUM，分组表个数
- TL_ZB_NWK_ADDR_MAP_NUM，地址映射表个数
- TL_ZB_NEIGHBOR_TABLE_NUM，邻居表个数
- ROUTING_TABLE_NUM，路由表个数
- NWK_ROUTE_RECORD_TABLE_NUM，路由记录表个数
- NWK_BRC_TRANSTBL_NUM，广播表个数
- NWK_ENDDEV_TIMEOUT_DEFAULT，终端设备保活超时时间
- ZB_MAC_RX_ON_WHEN_IDLE，终端设备空闲时RF接收机的状态
- ZB_NWK_LINK_STATUS_PEROID_DEFAULT，link status命令的周期时间

# 开发指南 -OTA

## ➢ OTA升级

- OTA初始化
  - OTA设备类型：Server/Client
  - ota_init()
- Client启动OTA定时查询
  - ota_queryStart(u16 seconds)
- Server发起OTA通知
  - zcl_ota_imageNotifyCmdSend()
- OTA条件检查
  - Manufacturer Code
  - Image Type
  - File Version



# 开发指南 - OTA

- OTA Image
  - 包含OTA Header，符合ZCL OTA规范，供远程设备实现空中升级的zigbee文件
  - 扩展：AES-128加密
- 生成方法
  - 将”sampleLight_9518.bin”拷贝到”zigbee_ota_tool_v2.2.exe”所在文件夹
  - 编码转换
    - ◆ 双击”zigbee_ota_tool_v2.2.exe”，按提示操作
    - ◆ 或使用命令行”./zigbee_ota_tool_v2.2.exe [arg1] [arg2]”

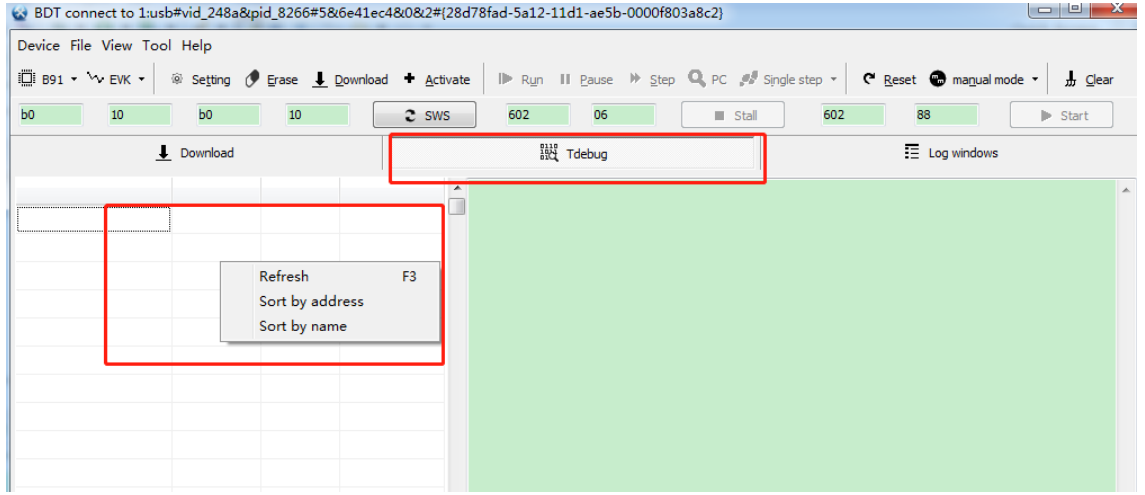
    [arg1]：文件名，如”sampleLight_9518.bin”

    [arg2]：不输入参数表示不加密；输入该参数代表使用该参数加密，输入格式如”00112233445566778899AABBCCDDEEFF”
- 输出文件
  - 1141-0201-10013001-sampleLight_9518.zigbee

# 四、调试方法

# 调试方法

- 使用printf()函数打印
  - printf(“test: %d, %x\r\n”, val_1, val_2)
- 使用BDT工具，直接查看RAM中的数据
  - 点击“Tdebug”
  - 确保当前的file文件与芯片所运行的固件一致
  - 在左边的显示栏中鼠标右键->”Refresh”
  - BDT工具将会根据mapping文件（xx.lst / objdump.txt）自动解析所有的全局变量，并将变量名和值显示到显示栏中
  - 需要注意的是，当设备在休眠时，是无法获取RAM中数据的




| Variable Name | Addr | Len | Value |
|---|---|---|---|
| g_bdbCommissionSet | 02308 | 18 | ... |
| g_gpCtx | 0231c | 23 | ... |
| g_zbDemoBdbCb | 02334 | 16 | ... |
| g_zcl_basicAttrs | 02344 | 54 | ... |
| gpEpCbs | 0237c | 12 | ... |
| myUartDriver | 02388 | 9 | ... |
| sampleGW_otaInfo | 02394 | 12 | ... |
| zclGpAttr_gpLinkKey | 023a0 | 16 | ... |
| g_zllTouchLink | 023b0 | 73 | ... |
| ndt.lto_priv.185 | 023fc | 13 | ... |
| pmcd | 0240c | 41 | ... |
| dGpStubHandle | 02435 | 1 | 00000000 |
| flash_cnt | 02436 | 1 | 00000000 |

# 五、Demo测试

# Demo测试

- 前期准备

  这里以TLSR9518为例。

  若使用其他型号的芯片，可自行修改CHIP_TYPE和BOARD定义，编译生成相应的固件。

  - 开发板
    - USB Dongle，作为Gateway（Coordinator）
    - EVK Board 1，作为Light（Router）
    - EVK Board 2，作为Switch（End Device）
  - 固件
    - sampleGW_9518_dongle.bin
    - sampleLight_9518_evk.bin
    - sampleSwitch_9518_evk.bin

# Demo测试

## ➢ 烧录连接

- USB Dongle
  - J56
    - ◆ 3V3
    - ◆ SWS
    - ◆ GND
- EVK Board
  - J51
    - ◆ VBAT（跳线帽：烧录时移除，烧录后插回）
  - J56
    - ◆ SWS
    - ◆ GND

# Demo测试

- 创建网络
  - 上电Gateway节点（ USB Dongle ），红色LED灯常亮。
    - 如果是新设备，将创建网络，并开启permit join on（180s）允许设备入网，绿色LED灯亮起。
    - 如果是已经建立过网络的设备，将恢复网络，可以通过按键SW7开启或关闭permit join的状态（绿色LED灯亮表示开启，绿色LED灯灭表示关闭）。
- 加入网络（在Gateway的permit join打开的条件下，也即是Gateway的绿色LED灯亮时）
  - 上电Light节点（EVK Board 1），红色LED灯亮起。
    - 如果是新设备，自动启动入网，入网成功后绿色LED灯将会亮起。
    - 如果是入过网的设备，将恢复网络。
  - 上电Switch（EVK Board 2）。
    - 如果是新设备，自动启动入网，入网成功后绿色LED灯闪烁。
    - 如果是入过网的设备，将恢复网络。

# Demo测试

## ➢ 按键操控功能

- Gateway
  - SW2：开启或关闭广播toggle命令，周期1秒。
    （如果Light节点已经入网成功，其红色LED灯将随之亮灭）
  - SW7：开启或关闭允许设备入网。
    （permit join，绿色LED灯亮表示开启，绿色LED灯灭表示关闭）
- Light
  - SW2：短按，切换灯（红色LED）的开关状态；
    长按5秒，离网，离网成功后红色LED闪烁三次。
  - SW3：开启或关闭允许设备入网。
    （permit join， 绿色LED灯亮表示开启，绿色LED灯灭表示关闭）
- Switch
  - SW2：短按， 广播toggle命令。
    （如果Light节点已经入网成功，其红色LED灯将随之亮灭）
    长按5秒，离网，离网成功后红色LED闪烁三次。
  - SW3：广播move to level命令。
    （如果Light节点已经入网成功，其红色LED灯亮度随之发生变化）

# 应用扩展

## ➢ 双模应用

- Zigbee + BLE（concurrent模式）
  - 分时复用射频模块，实现2种模式实时切换
  - 支持Zigbee Coordinator、Router和End Device（低功耗设备）
  - 支持BLE Master和Slave
- Zigbee + SIG Mesh（switch模式）
  - 未入网设备，主动检测当前有效网络（Zigbee或SIG Mesh）后自动接入网络
  - 已入网设备，保持原先网络状态直到factory new reset

## ➢ OS支持

- freeRTOS