

Tuya Beacon SDK AK80x IoTOS Guide

目录

1、SDK 概述.....	2
1.1 名词解释.....	2
1.2 SDK 架构.....	3
1.3 SDK 目录结构.....	4
1.4 主要业务流程.....	6
2、快速体验.....	9
2.1 环境构建.....	9
2.2 创建产品.....	9
2.3 运行调试.....	14
3、开发步骤.....	15
4、应用开发指导.....	16
4.1 初始化.....	16
4.2 Loop.....	17
4.3 DP 下发数据处理.....	18
5、API 详细介绍.....	19
5.1 beacon 协议栈初始化.....	19
5.2 beacon 接收处理函数.....	20
5.3 beacon 发送函数.....	21
5.4 beacon loop 函数.....	22
5.5 beacon 重置配网函数.....	23
5.6 beacon 获取配网状态函数.....	24
6、量产指导.....	25
6.1 购买授权码.....	25
6.2 芯片烧录授权.....	26
7、其它.....	27
7.1 涂鸦平台芯片烧录授权方式介绍.....	27
7.2 Tuya Beacon 协议栈芯片移植指导.....	27
7.2.1 移植概述.....	27
7.2.2 环境搭建.....	27
7.2.3 移植说明.....	28
8、FAQ.....	35

1、SDK 概述

可接入涂鸦平台的蓝牙协议包括：Tuya BLE 协议（简称：单点（单节点连接控制）、SIG Mesh 协议、Tuya Beacon 协议。

涂鸦蓝牙协议分类	通信方式	典型特点	典型应用
Tuya BLE	与手机或网关建立 BLE 连接进行通信	直连、低功耗、传输速率快	手环、手表、智能牙刷、智能体脂秤
SIG Mesh	遵循 SIG 蓝牙 MESH 标准的蓝牙组网通信	网状网络、大规模群控	MESH 灯具、商照、智能家居套件
Tuya Beacon	与手机或网关之间通过 BLE 广播进行通信	小数据包、近距离、超低速率、超低成本	低成本门磁、PIR、温湿度传感器、低成本灯带、低成本墙面贴、遥控器、场景开关

其中 Tuya Beacon 协议相对于 Tuya BLE 和 SIG Mesh 非常精简，其 SDK 特别适合直接移植到各种 BLE Beacon 芯片上，实现 SoC 级别的低成本应用。

Tuya Beacon SDK 特点：

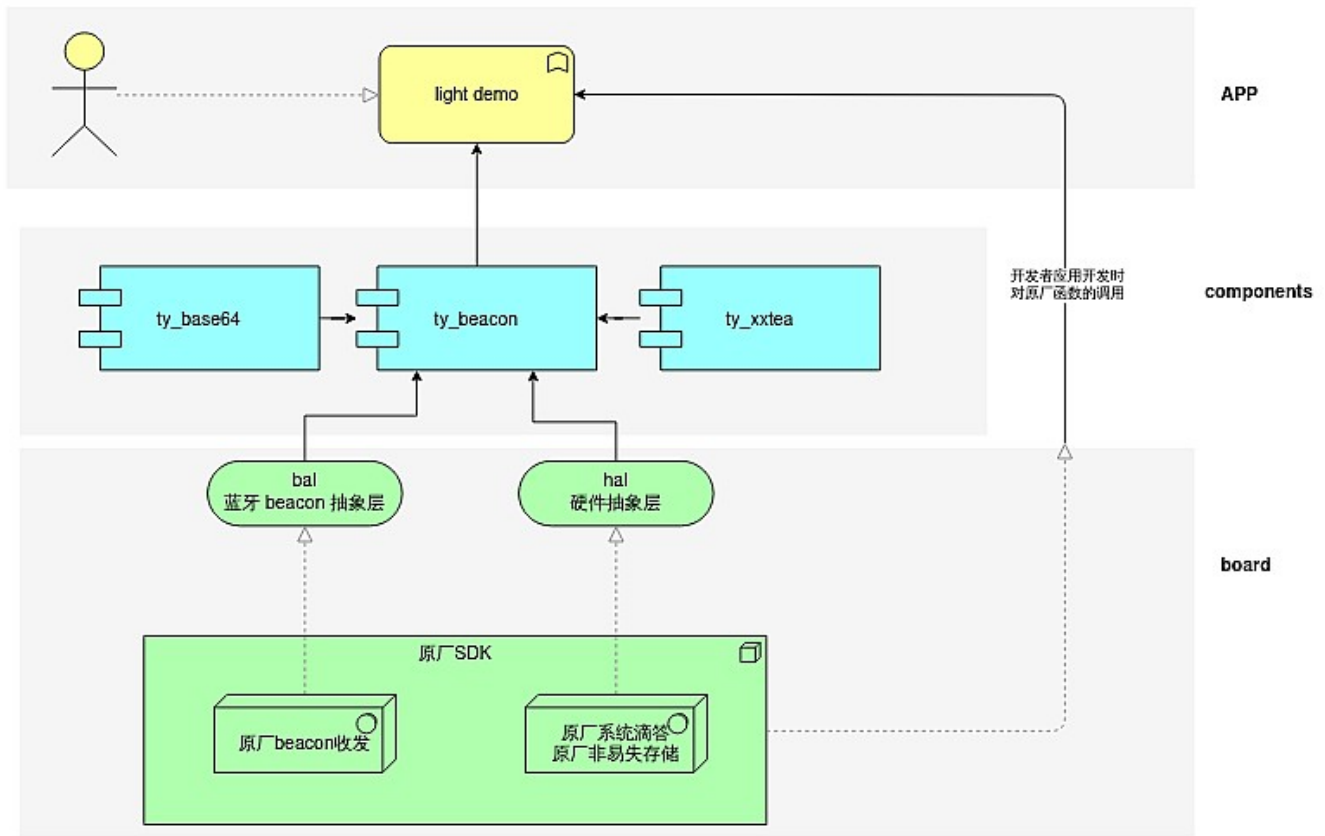
- 代码极简：总共约 1000 行 C 语言代码，可运行在 2K RAM + 16K ROM 的芯片上
- 跨平台：设计层面支持了跨硬件平台，只要实现 10 个以内的硬件和蓝牙广播相关函数，即可实现移植
- 功能完备：SDK 实现配对解绑、组包拆包、群组管理、加密解密、安全策略等丰富的 IoT 通信能力

1.1 名词解释

详细名词解释请参照：[涂鸦智能开发者名词解释](#)。

1.2 SDK 架构

SDK 架构图如下：



从底层向上：

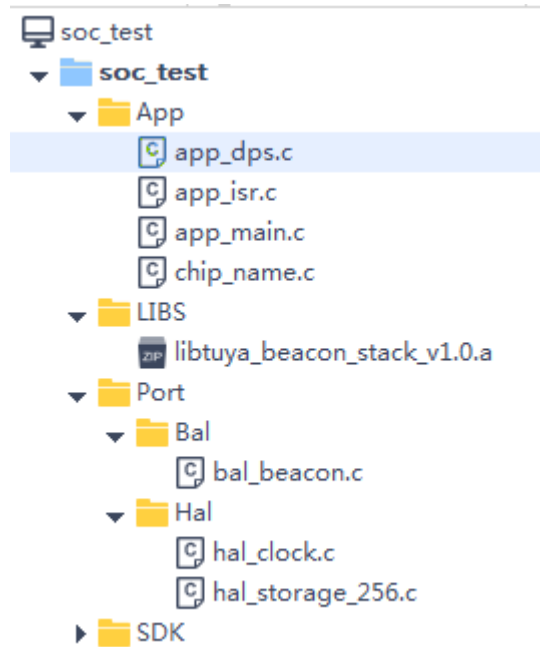
- board 层：包含芯片原厂提供的 SDK，以及为了对接涂鸦体系的适配文件（蓝牙抽象层 bal，硬件抽象层 hal）
- components 层：包含 Tuya Beacon 协议栈核心组件
 - ty_xtea 组件负责加密
 - ty_base64 组件负责 base64 解码
 - ty_beacon 组件负责核心 beacon 通信协议，其中 ty_xtea 和 ty_base64 仅仅服务于 ty_beacon
- app 层：包含实现应用逻辑的全部文件

1.3 SDK 目录结构

SDK 文件目录结构如下：

```
→ SDK git:(master) X tree -L 3
.
├── app #应用层
│   └── app_bluebeacon_light_ak80x
│       ├── app_cfg.h
│       ├── app_common.h
│       ├── app_dps.c
│       ├── app_dps.h
│       ├── app_ir.h
│       ├── app_isr.c
│       ├── app_main.c
│       ├── app_select.h
│       ├── _build
│       ├── config.json
│       └── readme.md
├── sdk #将 components 打包成库
│   ├── include
│   │   ├── ty_base64.h
│   │   ├── ty_beacon.h
│   │   └── ty_xtea.h
│   └── lib
│       └── libtuya_beacon_stack_v1.0.a
├── platforms #board层
│   └── ak80x_adv
│       ├── bal
│       ├── board.h
│       ├── hal
│       └── sdk
├── readme.md
└── CHANGE_LOG.md
```

某一示例在 AK801 专用 IDE 中打开目录结构如下：

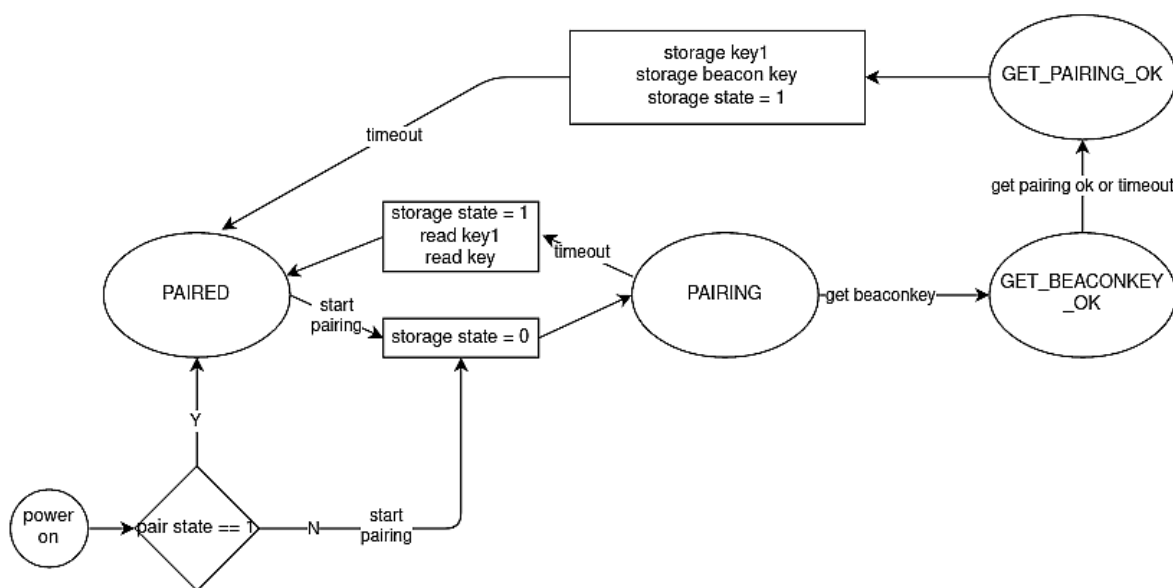


APP 对应文件目录结构中的 app_bluebeacon_light_ak80x

- LIBS 对应文件目录结构中的 sdk/lib
- Port 对应文件目录结构中的 platforms/ak80x_adv 中的 bal 和 hal
- SDK 对应文件目录结构中的 platforms/ak80x_adv 中的 sdk

1.4 主要业务流程

1) 设备配网状态状态机



协议栈级别的设备状态分为 4 个：

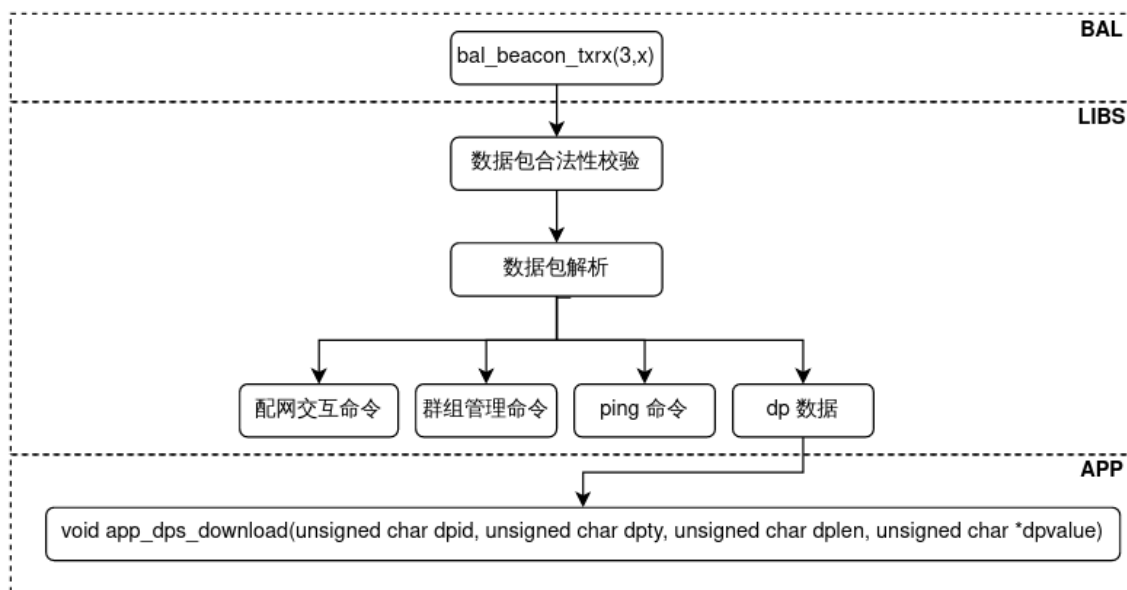
- PAIRED 已经配网
- PAIRING 正在配网
- GET_BEACONKEY_OK 正在配网过程中获取到云端分配的 BEACON_KEY
- GET_PAIRING_OK 正在配网过程中向手机/网关回复配网成功

协议栈中集成了误重置恢复上一次配网状态的逻辑，四种状态切换逻辑为：

- 上电从非易失存储上读取设备配网状态，如果已经配网，则切换到已经配网状态；如果没有配网，直接启动开始配网
- 每次调用 `ty_beacon_start_pairing(void)` 会立即存储设备配网状态为未配网，然后进入正在配网流程
- 如果在正在配网流程超时，会存储设备配网状态为已经配网，并从非易失存储读取 beacon 通信所需要的密钥，进行配网超时恢复
- 正常的配网流程会与手机或网关交互获取 BEACON_KEY，并在配网成功后设备返回配网成功命令

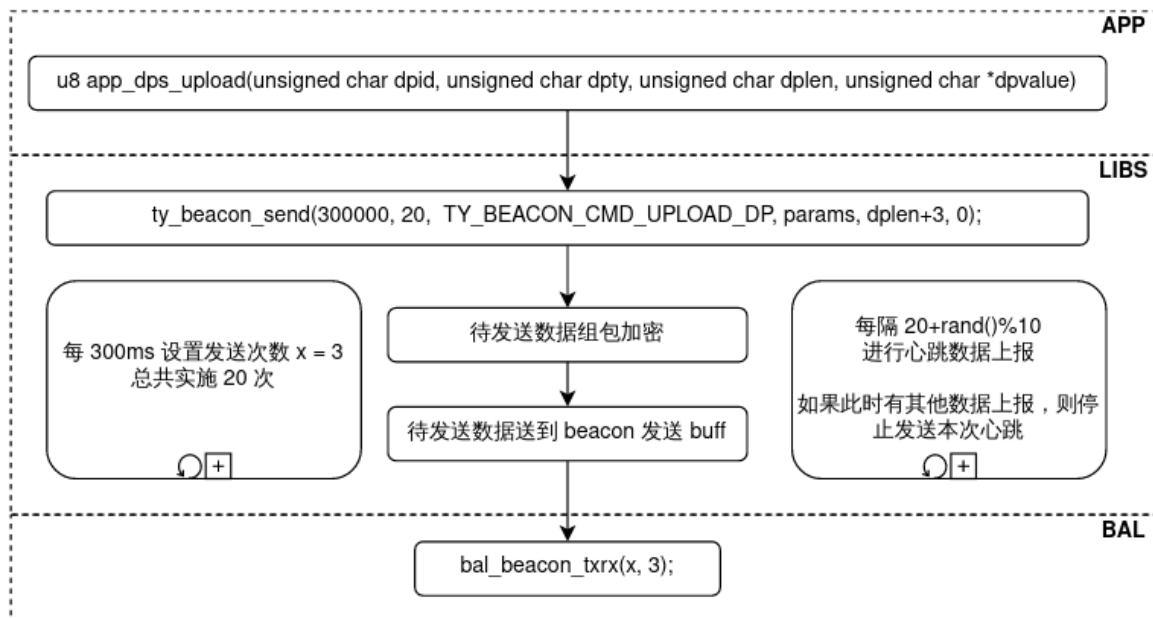
注：应用层只需要关注：PAIRED 和 PAIRING

2) 数据接收流程



系统循环调用 `bal_beacon_trx(x,3)` 持续搜索周围蓝牙广播包，收到的数据包到 beacon 协议栈层进行合法性校验、数据包解析，最终推到应用层收到的 DP 数据信息。

3) 数据发送流程



应用层调用 `u8 app_dps_upload(unsigned char dpid, unsigned char dpty, unsigned char dplen, unsigned char *dpvalue)` 设置要发送的 DP 数据，以及发送的频率和时间。

协议栈层对待发送数据进行组包和加密，送入 beacon 发送 buff，之后一旦调用 `bal_beacon_txrx(x,3)` 就会将 buff 中的数据发出去。

注：上述 300000（300ms）是为了保证 `bal_beacon_txrx(3,3)`，20 是 `txrx(3,3)` 重复 20 次，保证接收方能收到。

2、快速体验

2.1 环境构建

根据《Tuya_Beacon_SDK_AK80x_IoTOS_开发包_V1.0/hardware/芯片手册/开发环境搭建V1.0.pdf》搭建开发环境。

注：

1) IDE 在 Tuya_Beacon_SDK_AK80x_IoTOS_开发包_V1.0/pc/IDE/cdk-windows-V2.2.2-20200515-1730.zip

2) 开发板购买链接：后续会上架，目前可以联系兆焯微公司进行采购

2.2 创建产品

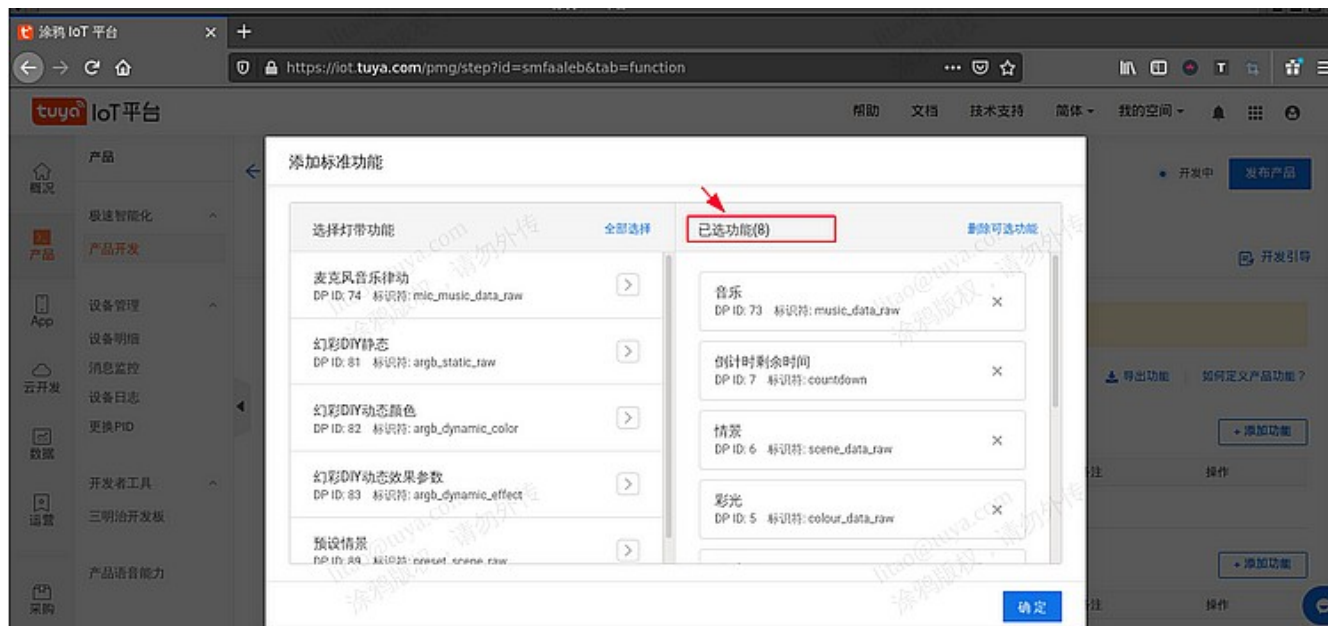
登录涂鸦开发者平台：<https://iot.tuya.com/>

STEP1：创建一个自定义开发的蓝牙 beacon 类型的灯带方案

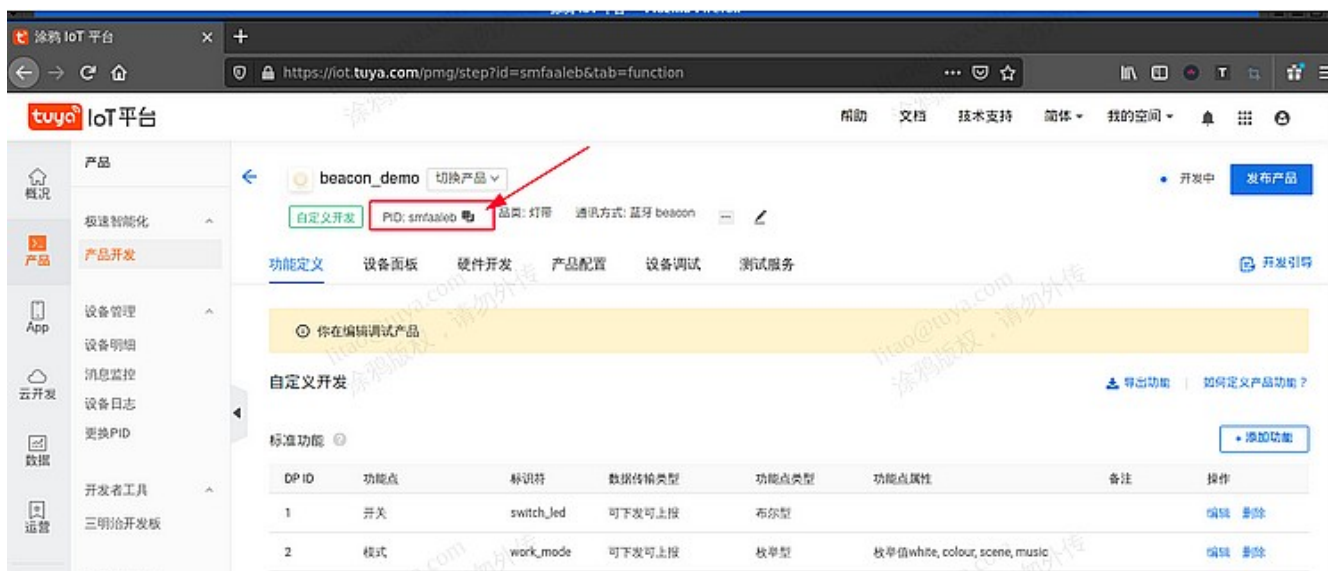




STEP2: 选择开关、模式、亮度值、冷暖值、彩光、情景、倒计时剩余时间、音乐功能选中

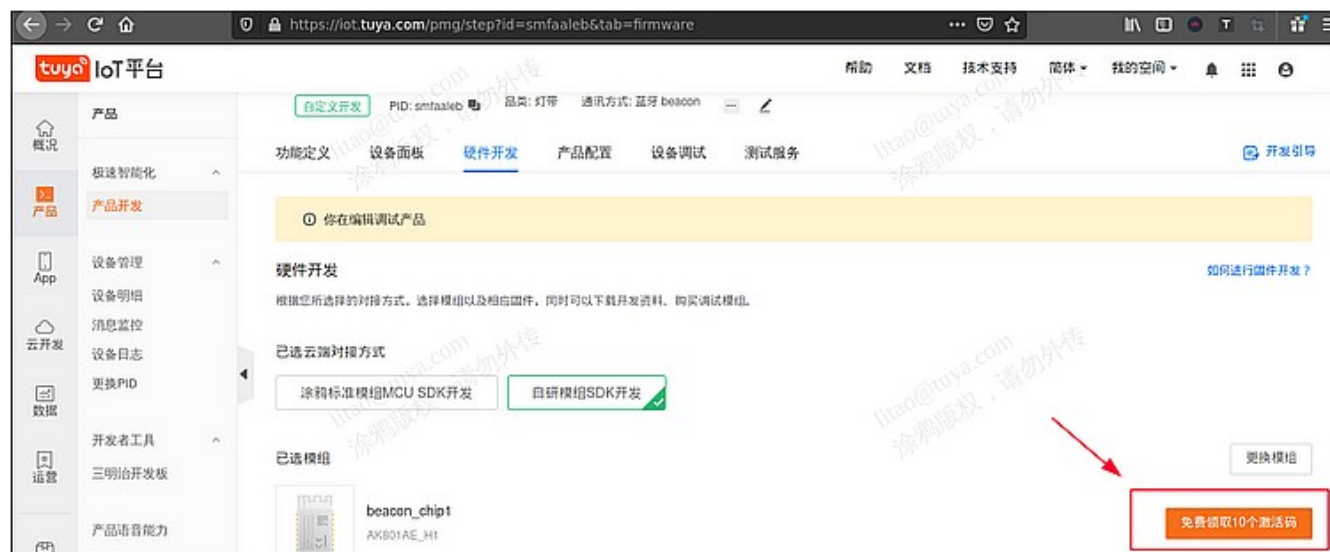
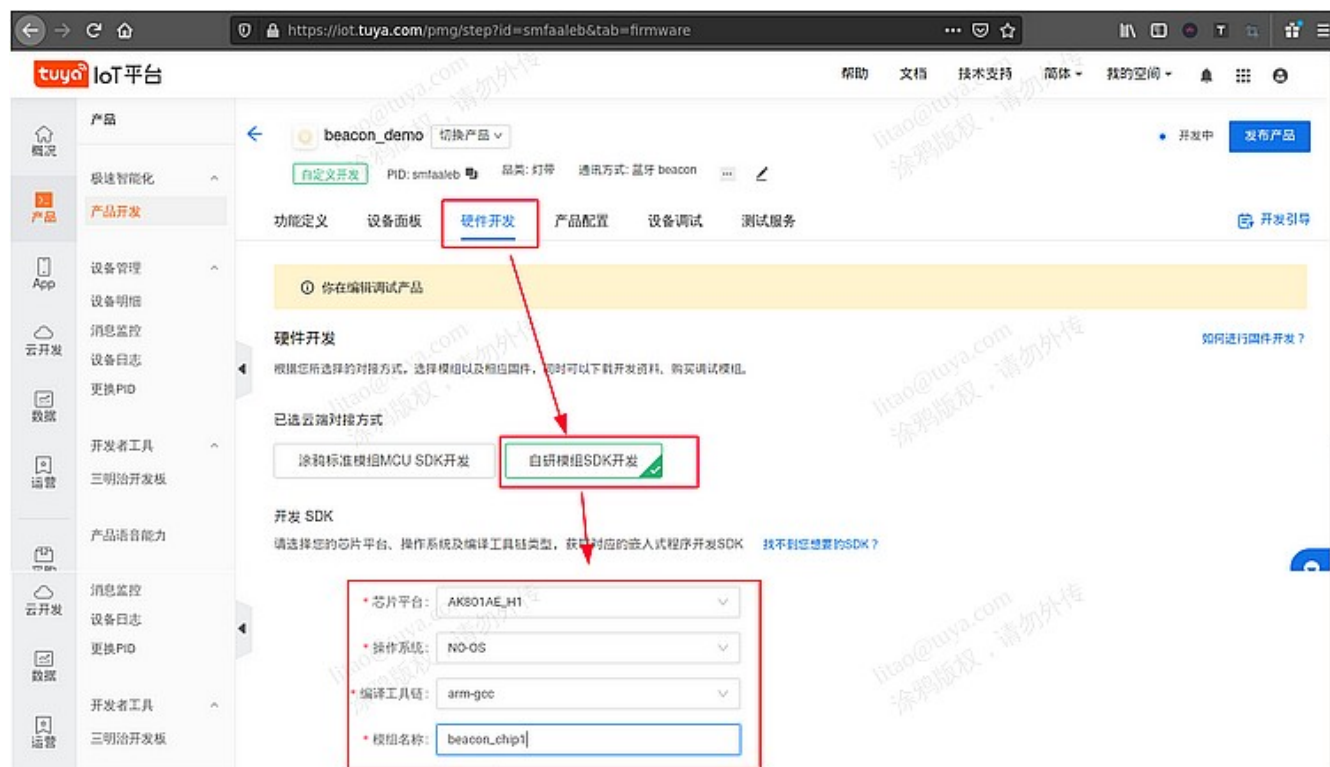


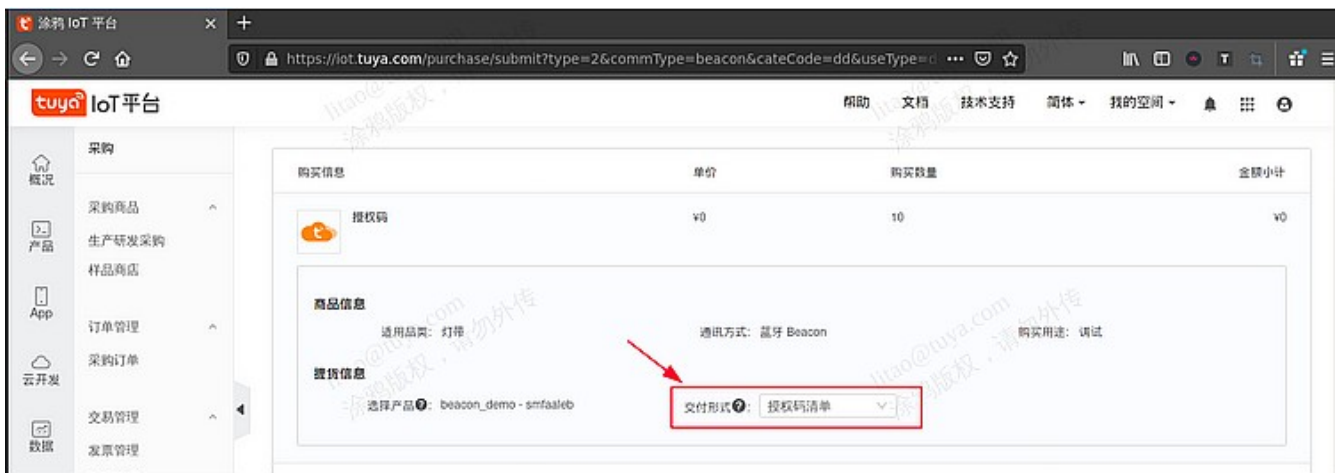
STEP3: 获取产品 ID (PID)



STEP4: 获取免费 10 个激活码

选择自研模组 SDK 开发，芯片平台参考如下进行填写：





等待大约 1 ~ 2 分钟，授权码清单生成，点击下载即可：



清单格式如下：

uuid	key	mac	
tuyae15aa4acb29dc46d	RAjczGyAK4lvmPIIXGpKIKwHiyRayKsg	DC234D28EBEB	举例，不是正确的数据

2.3 运行调试

1) 三色灯连接:

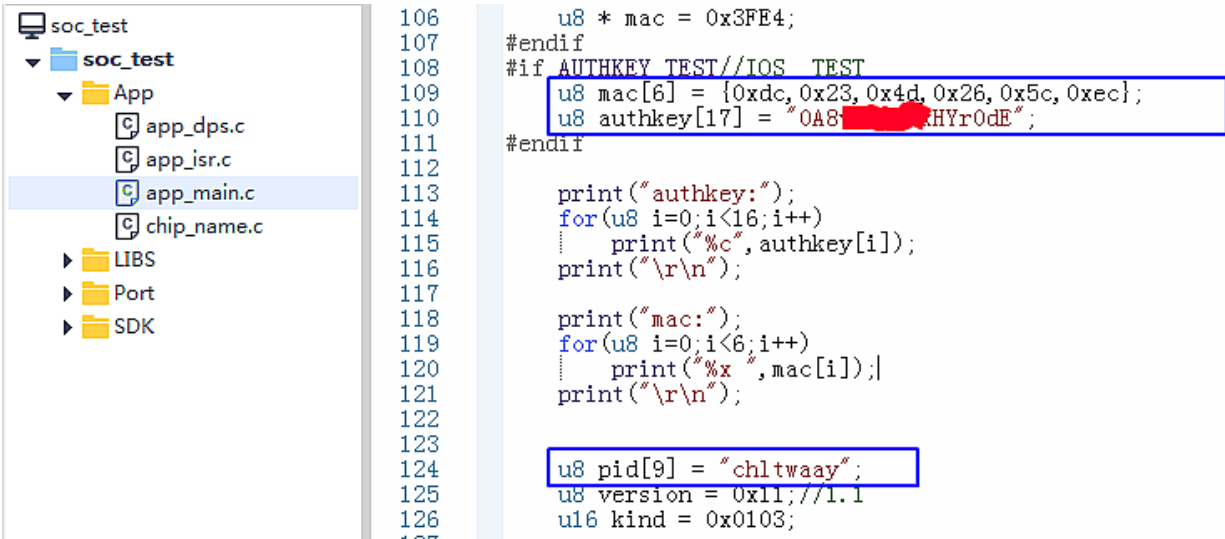
开发板 IO 丝印	代号	PIN
6	PWMG	IO3/PWM0
7	PWMR	IO4/PWM1
12	PWMB	IO8/PWM5
15	UART_TX	IO15/TXD (115200)

2) 三元组填写与开启调试模式

- 由于在开发板上看门狗、EEPROM 无法模拟，同时可以在调试状态时将 log 打开，因此需要先将 board/ak80x_adv/sdk/ak80x_adv_sdk/drv/config.h 中的 `//#define DEBUG` 去掉注释。

注：实际生产时需要关闭 DEBUG，节省出 ROM 资源并使能看门狗和 EEPROM。

- 修改 main.c 中的 pid 为你自己创建产品的 ID，authkey 取免费 10 个激活码中的任意一个（只取前 16 字节），mac 地址一定要和所取 authkey 对应：



```
106     u8 * mac = 0x3FE4;
107 #endif
108 #if AUTHKEY_TEST//IOS_TEST
109     u8 mac[6] = {0xdc, 0x23, 0x4d, 0x26, 0x5c, 0xec};
110     u8 authkey[17] = "0A8" + "xHYr0dE";
111 #endif
112
113     print("authkey:");
114     for(u8 i=0;i<16;i++)
115         print("%c", authkey[i]);
116     print("\r\n");
117
118     print("mac:");
119     for(u8 i=0;i<6;i++)
120         print("%x ", mac[i]);
121     print("\r\n");
122
123
124     u8 pid[9] = "chl'twaay";
125     u8 version = 0x11;//1.1
126     u16 kind = 0x0103;
```

启动调试后，RGB 灯闪烁，表示进入配网状态（30S 超时），此时使用涂鸦智能便可搜索、添加、控制设备，串口也有一些 log 帮助理解程序。

3、开发步骤

SDK+DEMO 获取 -> 开发环境搭建 -> 在 IoT 平台创建产品 -> 为产品设计合理的 DPs -> 为产品设计/选择合适的控制面板 -> 开发并测试 -> 发布产品进行小批量

注：

- 1) beacon 适合短数据，单包交互，不适合长数据，多包交互；
- 2) DP 有效数据最长支持 6 字节（bool 类型是 1 字节；value 类型是 4 字节；raw 和 string 类型请务必控制在 6 字节以内）；
- 3) Tuya Beacon 协议的优劣、限制、以及适用场景，见《涂鸦 BEACON 介绍.pdf》

4、应用开发指导

以三色灯《app/app_bluebeacon_light_ak80x》 DEMO 为例（每个 DEMO 的应用开发指导存放在其对应应用文件夹下）。

4.1 初始化

```
void app_light_init(void){  
    app_dps_read_from_storage();  
}
```

初始化函数主要实现：从非易失存储中读取上次设备的状态，并进行初始化设置，来实现设备的断电记忆功能。

4.2 Loop

```
void app_light_run(void){
    static u32 T = 0;

    if(ty_beacon_get_state() == DEVICE_STATE_PAIRING){//正在配网时，让灯 1S toggle 闪烁
        if(hal_clock_time_exceed(T,1000000)){
            light_value.scene_mode = SCENE_BLINK;
            light_value.onoff = 1;
            music_h = 0;
            music_s = 0;
            music_v = music_v == 0?1000:0;
            T = hal_clock_get_system_tick();
        }
    }else{//已经配网
        if(light_value.scene_mode == SCENE_BLINK)//停止配网闪烁
            light_value.scene_mode = SCENE_NORMAL;

        if(hal_clock_time_exceed(T,8000000)){//每隔 8S 对灯当前状态进行存储
            app_dps_write_to_storage();
            T = hal_clock_get_system_tick();
        }

        if(countdown != 0){//倒计时逻辑实现
            if(hal_clock_time_exceed(T,1000000)){
                countdown--;
                print("countdown=%d\r\n", countdown);
                if(countdown == 0){
                    set_onoff(countdown_target_onoff);
                }
                if(countdown%60 == 0){//1min upload
                    u8 dpvalue[4];
                    dpvalue[0] = (countdown >> 24) & 0xFF;
                    dpvalue[1] = (countdown >> 16) & 0xFF;
                    dpvalue[2] = (countdown >> 8) & 0xFF;
                    dpvalue[3] = (countdown >> 0) & 0xFF;
                    app_dps_upload(0x07,0x02,4,dpvalue);
                }
                T = hal_clock_get_system_tick();
            }
        }
    }
    scene_run();//灯的场景模式实现
    app_light_reset_run();//灯的三次上下电重置逻辑实现
}
```

灯的主循环函数主要实现：正在配网时灯的闪烁，已经配网时周期性存储灯的状态数据、倒计时、三次上下电重置设备等逻辑。

注：由于 AK801 资源有限，开启 DEBUG 模式后，也将应用层的一些逻辑注释掉了（如场景模式、倒计时、音乐模式，参见：`#ifndef DEBUG`）

4.3 DP 下发数据处理

```
void app_dps_download(unsigned char dpid, unsigned char dpty, unsigned char
dplen, unsigned char *dpvalue){
    print("dp:%d [%d]\r\n", dpid, dpvalue[0]);
    switch(dpid){
        case 0x01://onoff
            light_value.scene_mode = SCENE_NORMAL;
            light_value.onoff = dpvalue[0];
            app_dps_upload(dpid, dpty, dplen, dpvalue);
            break;
        case 12:
            light_value.S=1000, light_value.V=1000;
            light_value.scene_mode = dpvalue[0]+1;
            app_dps_upload(dpid, dpty, dplen, dpvalue);
            break;
        case 11:{//HSV
            light_value.H = ((dpvalue[0]<<8) | dpvalue[1]); //360
            light_value.S = dpvalue[2] * 10; //1000
            light_value.V = dpvalue[3] * 10; //1000
            light_value.scene_mode = SCENE_NORMAL;
            app_dps_upload(dpid, dpty, dplen, dpvalue);
        }
        break;
#ifdef DEBUG
        case 0x07:{//count down
            //uint32_t count
            countdown = (dpvalue[0]<<24) | (dpvalue[1]<<16) | (dpvalue[2]<<8) |
(dpvalue[3]);
            countdown_target_onoff = !light_value.onoff;
            app_dps_upload(dpid, dpty, dplen, dpvalue);
        }
        break;
        case 13://music
            music_h = ((dpvalue[1]<<8) | dpvalue[2]); //360
            music_s = dpvalue[3] * 10; //1000
            music_v = dpvalue[4] * 10; //1000
            light_value.scene_mode = SCENE_MUSIC;
            break;
#endif
        default:
            break;
    }
}
```

DP 下发处理函数，对应着平台上创建的 DPs，开发者自定义开发时，自己新增的 DPs 的处理可以放到这里。

5、API 详细介绍

5.1 beacon 协议栈初始化

函数	u8 ty_beacon_init(u8 pmac, u8 pauthkey, u8 *ppid, u8 version, u16 kind);	
作用	初始化 tuya beacon 协议栈	
入参	u8 *pmac	mac 6 bytes
	u8 *pauthkey	pid 8 bytes
	u8 *ppid	authkey 16 bytes (authkey[0~16],authkey total 32 bytes)
	u8 version	version 1 byte(0.0~9.9)
	u16 kind	kind (大小类, 0x0103 表示 3 路灯, 01 表示灯大类, 03 表示灯小类三路)
返回		

示例:

往往在程序初始化的地方调用:

```
u8 mac[6] = {0xdc, 0x23, 0x4d, 0x26, 0x5c, 0xe6};
u8 authkey[17] = "jmF7Ju8Jxxxxxxxxx";
u8 pid[9] = "smfaaleb";
u8 version = 0x10; //1.0
u16 kind = 0x0103;

ty_beacon_init(mac, authkey, pid, version, kind);
```

5.2 beacon 接收处理函数

函数	u8 ty_beacon_download(u8 *rx_buf, u8 len_pdu, u8 rssi);	
作用	beacon 接收处理函数，从扫描回调来的 beacon 原始数据，送入该函数，进行过滤解析	
入参	u8 *rx_buf	beacon 接收原始数据：header(2B) +mac(6B) +data(max31B)
	u8 len_pdu	beacon 接收的原数据长度
	u8 *ppid	authkey 16 bytes (authkey[0~16],authkey total 32 bytes)
	u8 rssi	接收该条 beacon 时信号强度
返回		

示例：

往往在 beacon 接收回调地方调用：

```
void rx_callback(...){
    ...
    rx_rssi = rf_get_rssi();
    ty_beacon_download(rx_buffer,buffer_len,rx_rssi);
}
```

5.3 beacon 发送函数

函数	u8 ty_beacon_send(u32 cycle, u8 send_times, u8 cmd, u8* params, u8 params_len, u8 tp);	
作用	能够将一包数据，按照一定周期，分 send_times 次发送 注：底层对每一包 beacon 数据会在 37，38，39 上轮流发数次，我们称之为底层发包 1 次 这里每个 send_times 对应一次底层发包，假设底层发包一次时间为 40ms，cycle 设置为 100ms，则会出现每 100ms 的前 40ms 在进行底层发包 注：再次调用会直接打断上一次的发包，进行全新一次的发包	
入参	u32 cycle	发包周期，us
	u8 send_times	发包次数
	u8 cmd	发包的命令
	u8* params	发包的参数
	u8 params_len	发包参数的长度
	u8 tp	填 0
返回		

示例：

dp 上报函数：

```
u8 app_dps_upload(unsigned char dpid, unsigned char dpty, unsigned char dplen, unsigned char *dpvalue){
    if(dplen > 7)return 0;
    u8 params[10];
    params[0] = dpid;
    params[1] = dpty;
    params[2] = dplen;
    memcpy(&params[3], dpvalue, dplen);
    ty_beacon_send(300000, 20, TY_BEACON_CMD_UPLOAD_DP, params, dplen+3, 0); //300ms send 20 times, again use ty_beacon_send will break pre send
    return 1;
}
```

注：为什么这么设计？主要考虑 beacon 传输速率低、不稳定，因此对于同一个命令持续发送多次，来保证接收端能收到。如上面示例，上报一个 dp 数据以 300 ms 周期，分 20 次尝试发送（底层会更高频发送），也就是说如果没有被新上报命令打断，该上报会持续 300ms * 20 = 6S。

5.4 beacon loop 函数

函数	u8 ty_beacon_run(void);	
作用	需要被高频周期性调用，来驱动 beacon 协议栈	
入参		
返回		

示例：

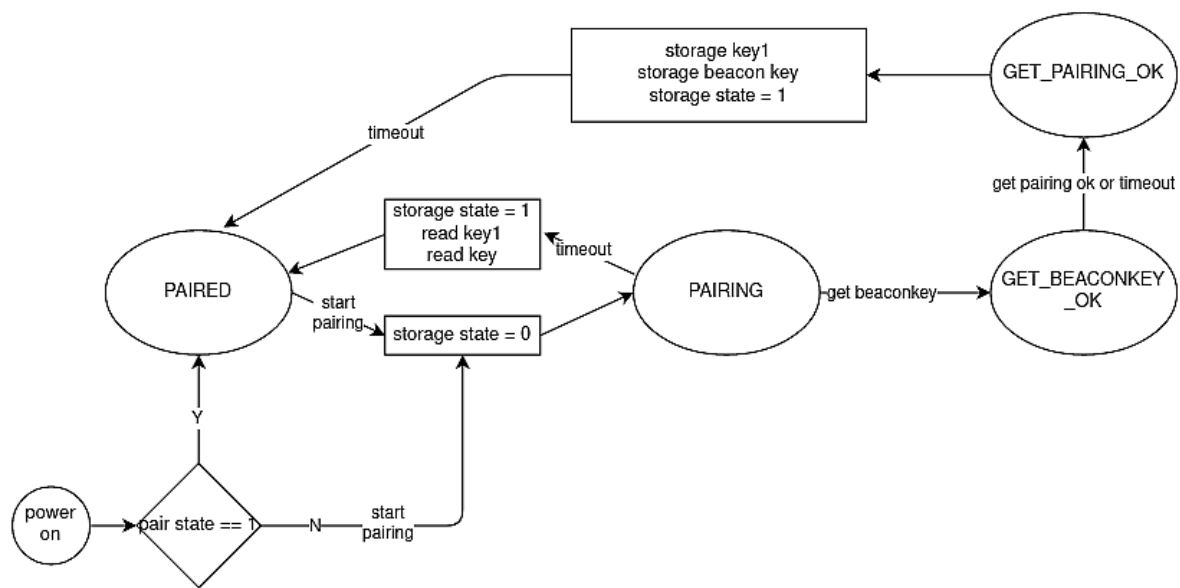
```
int main( void ){
    ...
    while(1){
        ...
        ty_beacon_run();
    }
}
```

5.5 beacon 重置配网函数

函数	u8 ty_beacon_start_pairing(void);	
作用	一旦调用，整个系统进入配网状态 40 S	
入参		
返回		

一般在满足配网条件下调用该函数，如：复位按键按动

tuya beacon 协议栈状态机如下，带有自动恢复功能：



5.6 beacon 获取配网状态函数

函数	state_machine_e ty_beacon_get_state(void);	
作用	获取配网状态	
入参		
返回	typedef enum{ DEVICE_STATE_PAIRING = 0x00, DEVICE_STATE_PAIRING, //← 已经配网，其他都是未配网 DEVICE_STATE_NOT_PAIRING, DEVICE_STATE_REP_GET_BEACONKEY, DEVICE_STATE_REP_GET_PAIRING_OK }state_machine_e;	

示例：

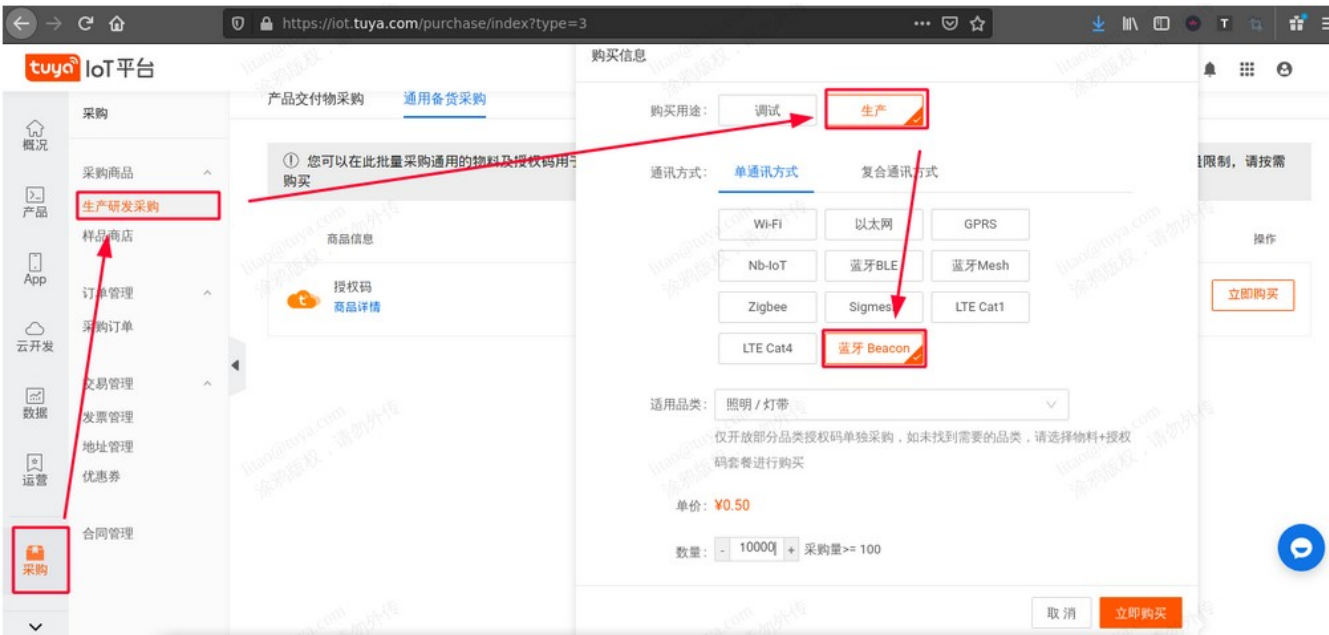
上电判断是否时未配网状态（从 flash 读取上次存储信息），如果处于未配网状态，则直接开启配网时间窗，等待配网：

```
if(ty_beacon_get_state() == DEVICE_STATE_NOT_PAIRING)
    ty_beacon_start_pairing();
```

6、量产指导

6.1 购买授权码

当开发的产品完成功能测试、稳定性测试、老化测试、拉距测试后，对产品进行发布，然后可在涂鸦 IoT 平台采购授权码：（选择授权码清单）

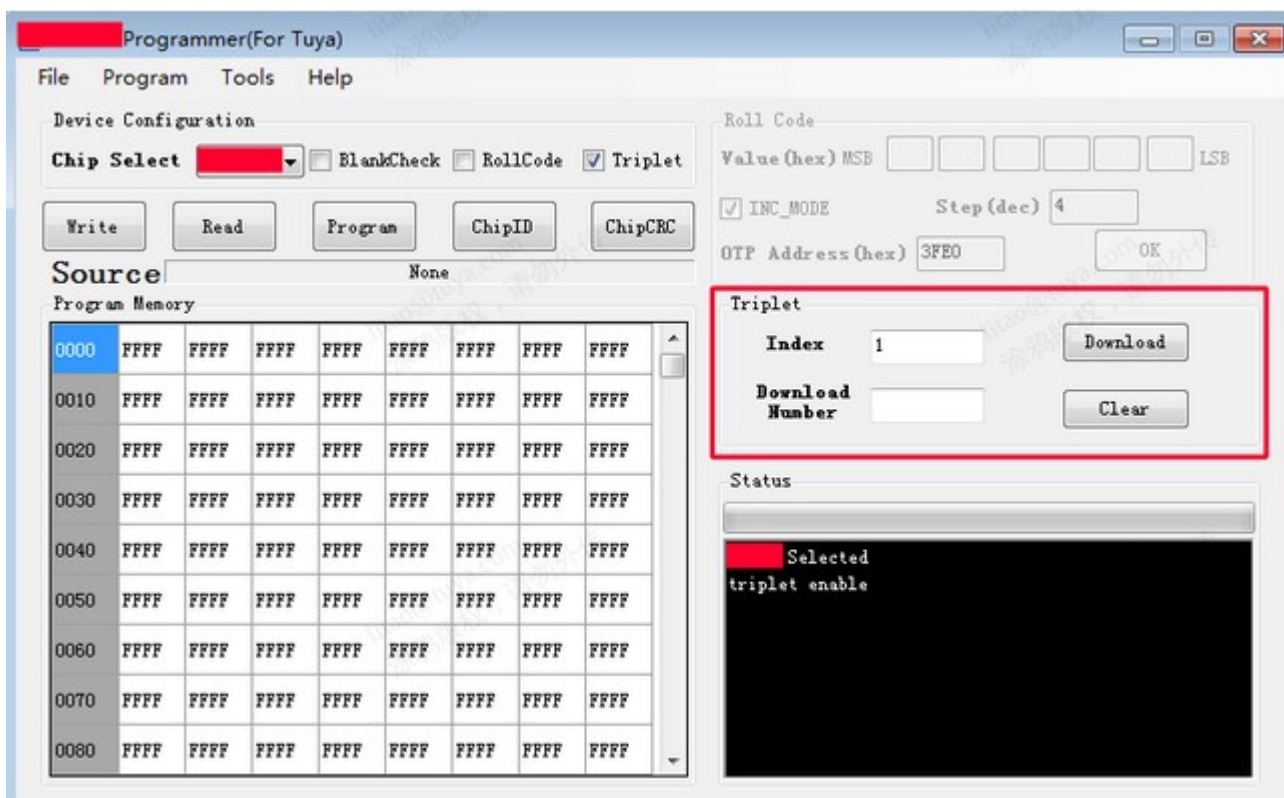


订单成功后，在邮箱中会收到授权码清单（一个表格，将表格另存为 xlsx 格式，表格中内容格式如下：）

uuid	key	mac	
tuyae15aa4acb29dc46d	RAjczGyAK4lvmPIIXGpKIKwHiyRayKsg	DC234D28EBEB	举例，不是正确的数据

6.2 芯片烧录授权

参考《Tuya_Beacon_SDK_AK80x_IoTOS_开发包_V1.0/pc/烧录授权软件/AK801 芯片烧录器使用说明.pdf》，使用绿色免安装软件《Tuya_Beacon_SDK_AK80x_IoTOS_开发包_V1.0/pc/烧录授权软件/烧录授权 V1.0.rar》，采用 Triplet 三元组表格授权模式，对烧写器进行写入 bin 文件和 n 条三元组信息：



- 1) Chip Select 选择 AK801，模式选择 Triplet
- 2) 烧写器连接好 AK80X Programmer(For Tuya) 后，点击 Triplet 面板中的 Clear 按钮，清空烧写器中原有授权三元组残留（擦除需要一些时间）
- 3) File -> Import Bin (选择待烧录固件) -> Write 实现待烧录固件写入离线烧写器
- 4) 在 Triplet 面板，Index 填写 1，Download Number 填写需要写入到离线烧写器中的三元组数量，然后点击 Downlod 按钮，选择 6.2 节申请的授权码清单（xlsx），实现将授权码清单写入离线烧写器
- 5) 给离线烧写器重新上电，将待烧写芯片放入卡槽中，按烧写器中间的按钮，如果烧录成功，烧录指示灯会变绿

注：代码里 authkey 和 mac 不能直接写死，而是读取对应 flash/OTP 中的内容，对其进行初始化。量产固件一定要将 DEBUG 模式关闭，使能看门狗和 EEPROM！！！！

7、其它

7.1 涂鸦平台芯片烧录授权方式介绍

- 采用基于串口通信的涂鸦标准产测授权协议，实现将授权信息写入芯片
- 直接授权码清单的方式，客户需要自己制作上位机，将 MAC\AUTHKEY\UUID 等信息直接写入芯片（AK801 就是采用这种方式，这种方式如果客户直接购买授权码清单进行大批量生产，需要经过涂鸦内部审核）

7.2 Tuya Beacon 协议栈芯片移植指导

7.2.1 移植概述

目前针对如下平台已经做好移植：

芯片	厂家	指令集类别	IDE	SDK+DEMO 地址
AK801	兆焯微	平头哥	CSKY	https://github.com/tuya/tuya-iotos-beacon-sdk-ak80x
MS1656	巨微	32 位 ARM Cortex-M0+	KEIL	待上架
BK3431S	BK	ARM7	KEIL	待上架

如果没有所选芯片的 Demo Project，您可以下载 Tuya Beacon SDK For Linux 并根据 SDK 中的文档（或本文）说明自行移植（下载地址：[Tuya Beacon SDK For Linux](#)）。

注：如果所选芯片指令集有和已经移植好的 Demo Project 一样的，推荐参考相同指令集的 Demo Project。

7.2.2 环境搭建

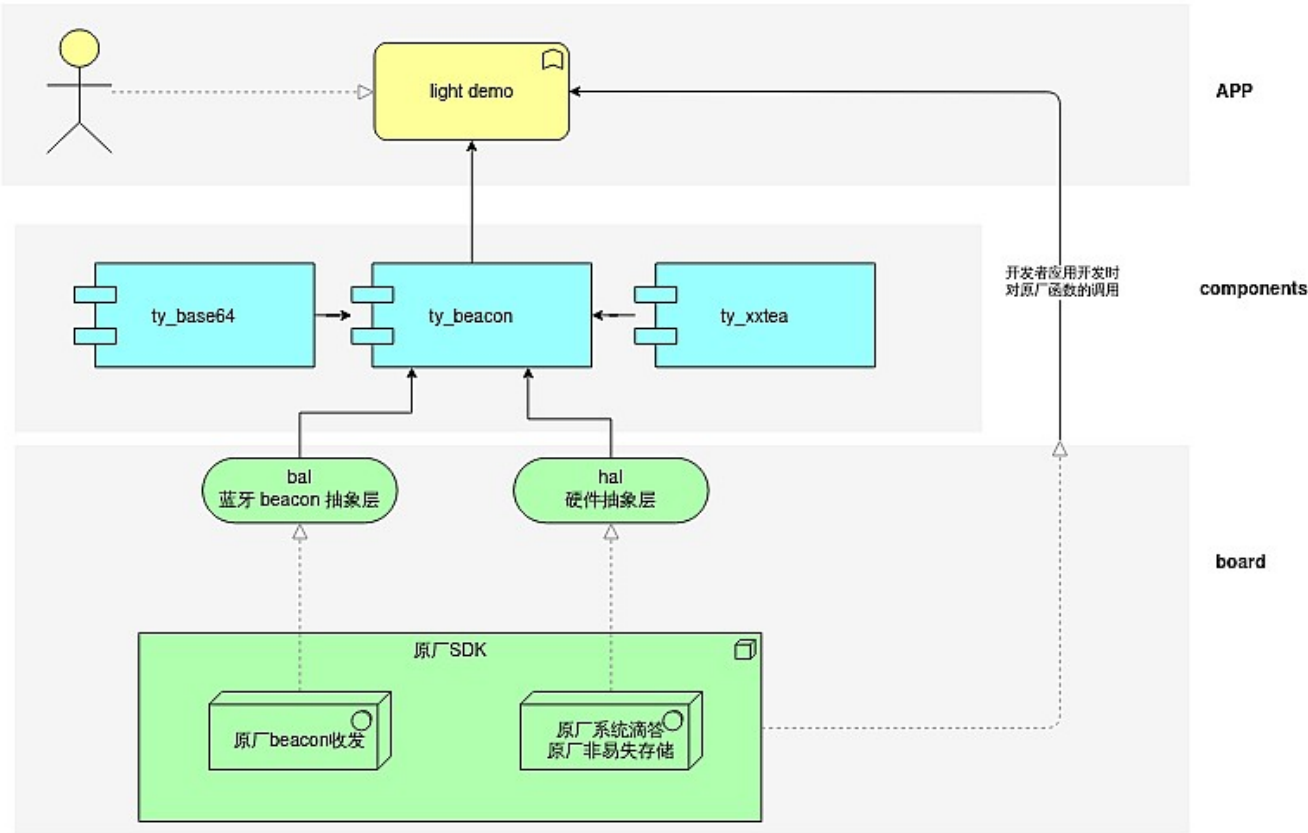
软件环境

- IDE 根据芯片原厂 SDK 要求进行安装。
- 硬件环境

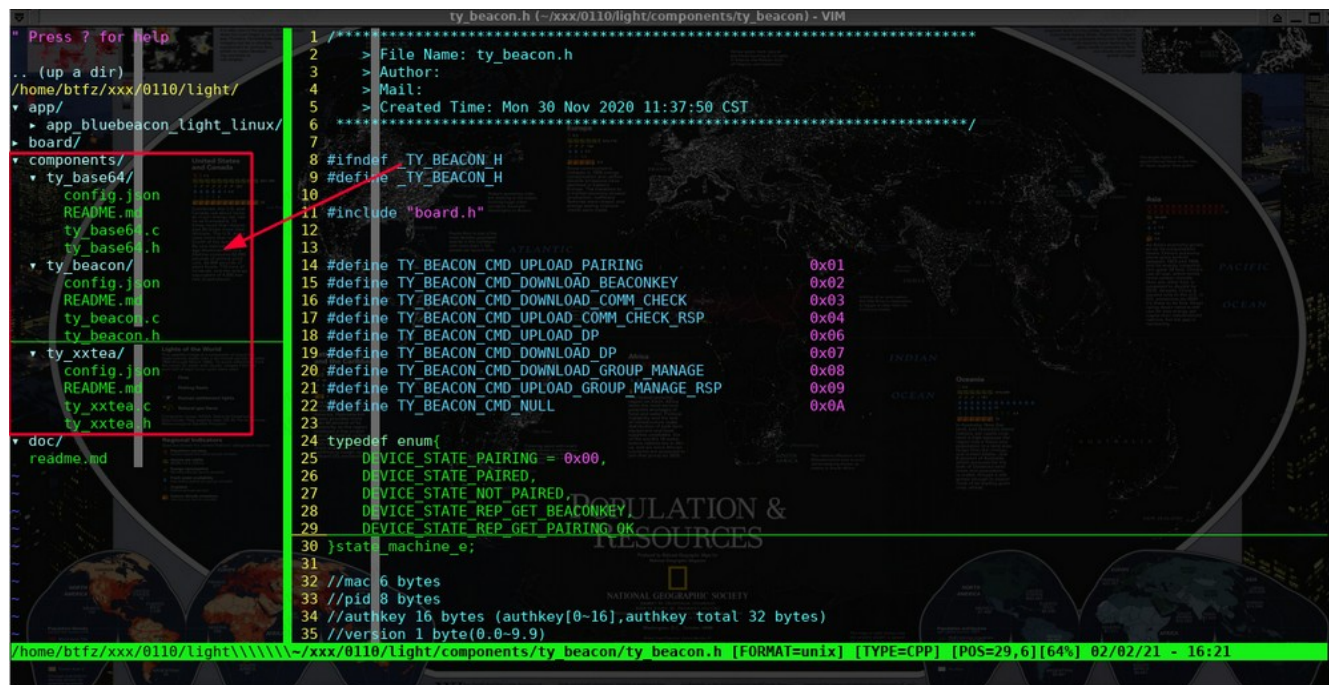
硬件平台可以选用芯片原厂对应的开发板，或者使用自研 PCBA，Tuya BLE SDK Demo Project 是基于芯片原厂开发板开发测试。

7.2.3 移植说明

整体架构如下：



STEP1: 核心组件移植



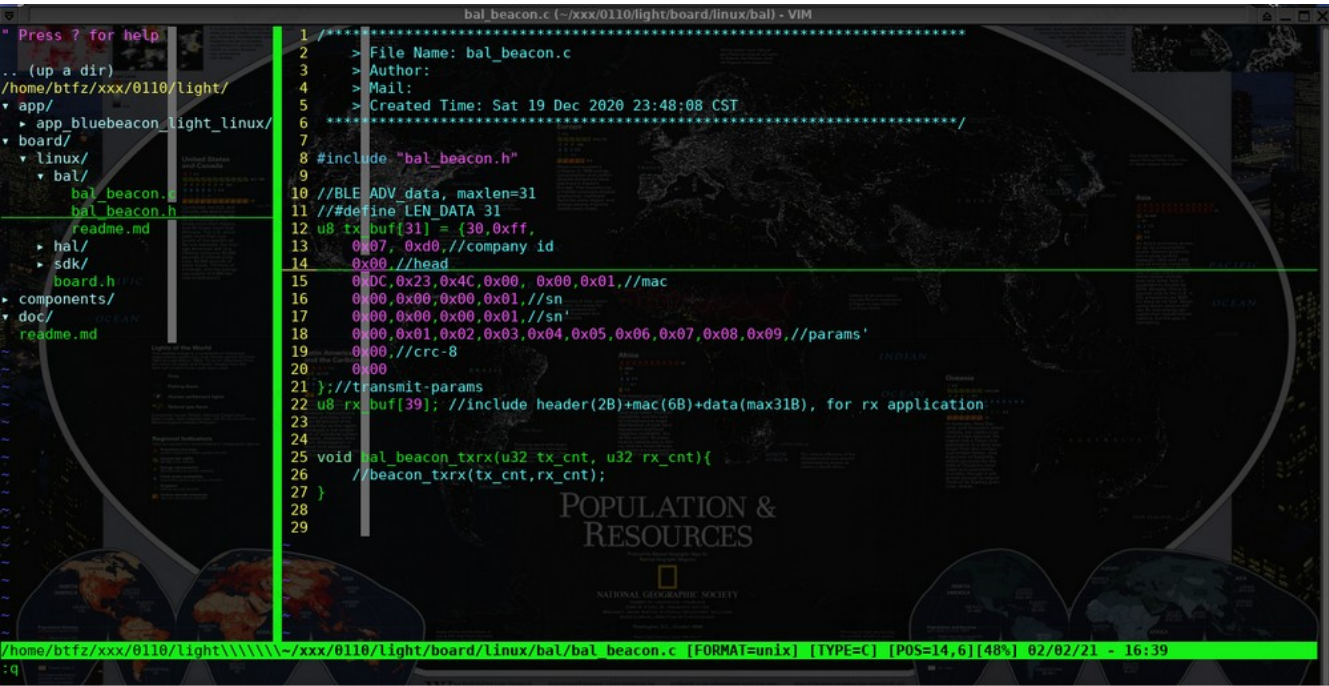
```
ty_beacon.h (-/xxx/0110/light/components/ty_beacon) - VIM
1  /*****
2  > File Name: ty_beacon.h
3  > Author:
4  > Mail:
5  > Created Time: Mon 30 Nov 2020 11:37:50 CST
6  *****/
7
8  #ifndef TY_BEACON_H
9  #define _TY_BEACON_H
10
11 #include "board.h"
12
13
14 #define TY_BEACON_CMD_UPLOAD_PAIRING 0x01
15 #define TY_BEACON_CMD_DOWNLOAD_BEACONKEY 0x02
16 #define TY_BEACON_CMD_DOWNLOAD_COMM_CHECK 0x03
17 #define TY_BEACON_CMD_UPLOAD_COMM_CHECK_RSP 0x04
18 #define TY_BEACON_CMD_UPLOAD_DP 0x06
19 #define TY_BEACON_CMD_DOWNLOAD_DP 0x07
20 #define TY_BEACON_CMD_DOWNLOAD_GROUP_MANAGE 0x08
21 #define TY_BEACON_CMD_UPLOAD_GROUP_MANAGE_RSP 0x09
22 #define TY_BEACON_CMD_NULL 0x0A
23
24 typedef enum{
25     DEVICE_STATE_PAIRING = 0x00,
26     DEVICE_STATE_PAIRING,
27     DEVICE_STATE_NOT_PAIRING,
28     DEVICE_STATE_REP_GET_BEACONKEY,
29     DEVICE_STATE_REP_GET_PAIRING_OK,
30 }state_machine_e;
31
32 //mac 6 bytes
33 //pid 8 bytes
34 //authkey 16 bytes (authkey[0-16],authkey total 32 bytes)
35 //version 1 byte(0.0-9.9)
```

ty_xxtea 组件负责加密

- ty_base64 组件负责 base64 解码
- ty_beacon 组件负责核心 beacon 通信协议，其中 ty_xxtea 和 ty_base64 仅仅服务于 ty_beacon

移植时直接将 components 中的所有 .c .h 加入目标工程。

STEP2：蓝牙适配层实现（ beacon 收发）



```
bal_beacon.c (-:/xxx/0110/light/board/linux/bal) - VIM
1  /*****
2  > File Name: bal_beacon.c
3  > Author:
4  > Mail:
5  > Created Time: Sat 19 Dec 2020 23:48:08 CST
6  *****/
7
8  #include "bal_beacon.h"
9
10 //BLE ADV_data, maxlen=31
11 //define LEN_DATA 31
12 u8 tx_buf[31] = {30, 0xff,
13 0x07, 0xd0, //company id
14 0x00, //head
15 0xDC, 0x23, 0x4C, 0x00, 0x00, 0x01, //mac
16 0x00, 0x00, 0x00, 0x01, //sn
17 0x00, 0x00, 0x00, 0x01, //sn'
18 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, //params'
19 0x00, //crc-8
20 0x00
21 }; //transmit-params
22 u8 rx_buf[39]; //include header(2B)+mac(6B)+data(max31B), for rx application
23
24
25 void bal_beacon_ttrx(u32 tx_cnt, u32 rx_cnt){
26 //beacon_ttrx(tx_cnt, rx_cnt);
27 }
28
29
```

蓝牙 beacon 适配层，封装了蓝牙 beacon 收发函数：void bal_beacon_ttrx(u32 tx_cnt, u32 rx_cnt);

该函数定义如下：

参数名称	说明	备注
tx_cnt	37, 38, 39 信道轮流发送 beaconn 数据的总次（需要为 3 的倍数）	待发送 beacon 数据放在：tx_buf[31] 中（参考代码中直接用，不要改动）
rx_cnt	37, 38, 39 信道轮流接收 beaconn 数据的总次（需要为 3 的倍数）	接收到的 beacon 数据放在：rx_buf[39] 中（参考代码中直接用，不要改动）

注意：

- 1) 如果收发不能并行，调用上述函数，先发送，再接收
- 2) 如果条件允许，可以一直开接收，在接收回调函数中放置：

```
void rx_callback(...){
    ...
    rx_rssi = rf_get_rssi();
    ty_beacon_download(rx_buffer, buffer_len, rx_rssi);
}
```

3) 如果条件允许，发送按照最小时间间隔发送

移植时直接将 bal 中的所有 .c.h 加入目标工程，基于目标平台实现 bal_beacon_txrx 函数。

STEP3: 核心硬件抽象层实现 (us 级别系统滴答 + 存储)

Tuya beacon 协议栈中的防重放序列号，加密 key 等信息需要借助一个 256 字节任意位置读写的非易失存储驱动，和一个用来给协议栈提供定时逻辑的 us 级别的系统滴答驱动。

其中非易失存储驱动要求如下：

hal_storage_256.h

```
/* *****
> File Name: hal_storage_256.h
> Author:
> Mail:
> Created Time: Tue 26 Jan 2021 12:26:46 CST
> 256 bytes fast storage(eeprom or flash ...)
***** */

#ifndef _HAL_STORAGE_256_H
#define _HAL_STORAGE_256_H

#include "hal_sys.h"

u8 hal_storage_256_init(void);
u8 hal_storage_256_read_bytes(u16 offset, u8* pdata, u8 len); //offset 0~0xFF
u8 hal_storage_256_write_bytes(u16 offset, u8* pdata, u8 len);

#endif
```

协议栈不会高频调用读写，但是建议实现读写时采用双备份方式。

其中 us 级别的系统滴答驱动要求如下：

hal_clock.h

```
/* *****
> File Name: hal_clock.h
> Author:
> Mail:
> Created Time: Wed 03 Jul 2019 22:09:57 CST
***** */

#ifndef _HAL_CLOCK_H
#define _HAL_CLOCK_H

#include "hal_sys.h"

#define HAL_CLOCK_HZ 1000000    //tick 增加的频率（某些系统直接等于系统主频）

enum{//对 1S\1MS\1US 等于多少 tick 的定义
    //how much 1us = ? ticks
    HAL_CLOCK_1S_TICKS = HAL_CLOCK_HZ,
    HAL_CLOCK_1MS_TICKS = (HAL_CLOCK_1S_TICKS / 1000),
    HAL_CLOCK_1US_TICKS = (HAL_CLOCK_1MS_TICKS / 1000),
};

extern u32 hal_clock_get_system_tick(void);//获取当前 tick
extern u32 hal_clock_time_exceed(u32 ref, u32 span_us);//ref 填写 old_tick，该函数能够计算当前 tick-old_tick 是否超过了设定时间 span_us，超过返回 true（非阻塞）

#endif
```

另外三个辅助.h

- hal_sys.h：放置所有原厂 SDK 相关的头文件（组件层或其他 hal 层文件通过引用 hal_sys.h 来包含原厂头文件），例如：

```
#include<stdio.h>
#include<string.h>
#include<time.h>
#include<unistd.h>
#include<pthread.h>
#include<stdlib.h>

#include <unistd.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/stat.h>
```

- hal_types.h：定义 u8\u16\u32 s8\s16\s32 等


```
typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;

typedef char s8;
typedef short s16;
typedef int s32;
```

□ board.h: 放置:

```
#include "hal_clock.h"

#include "hal_storage_256.h"

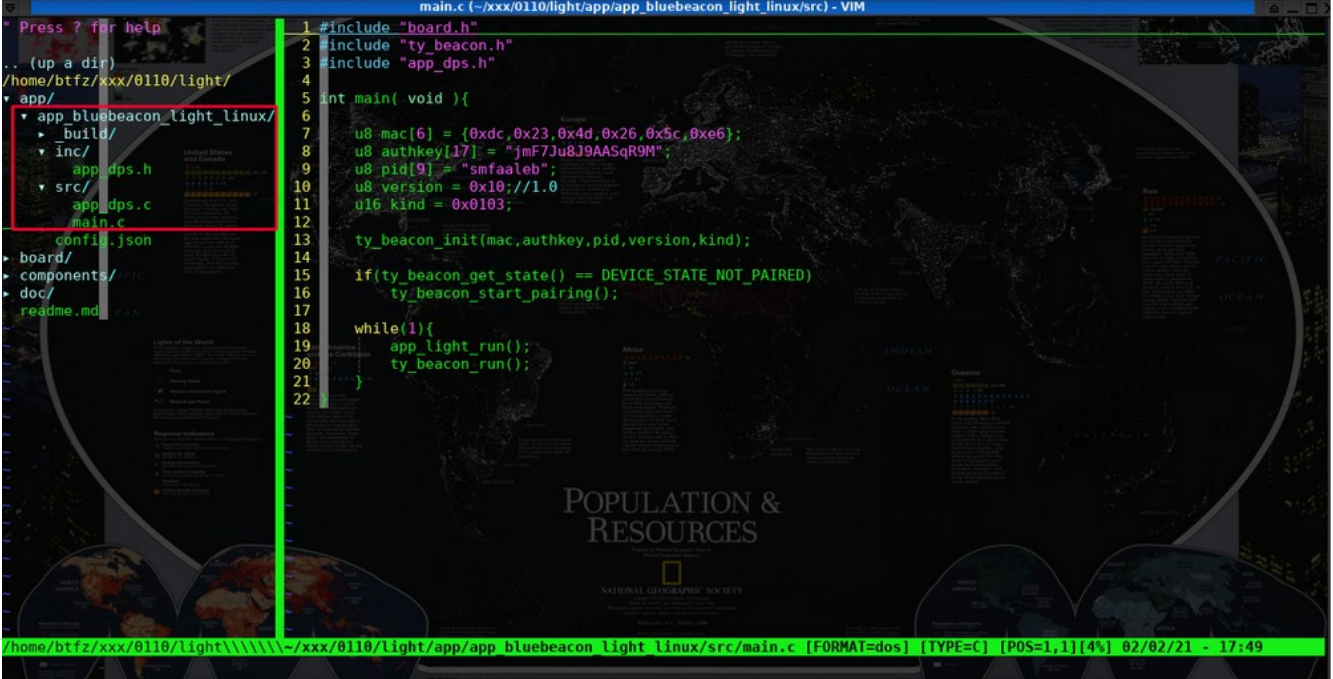
#include "hal_sys.h"

#include "hal_types.h"
```

最终将下面适配好的文件全部加入工程:

适配文件名	作用
hal_storage_256.h	存储
hal_storage_256.c	
hal_clock.h	定时
hal_clock.c	
hal_sys.h	系统头文件汇总
hal_types.h	重定义 u8\u16\u32\s8\s16\s32
board.h	hal 层头文件汇总

STEP4: 应用层最小 DEMO



```
main.c (~/xxx/0110/light/app/app_bluebeacon_light_linux/src) - VIM
" Press ? for help
.. (up a dir)
/home/btfz/xxx/0110/light/
app/
  app_bluebeacon_light_linux/
    build/
    inc/
    app_dps.h
    src/
      app_dps.c
      main.c
    config.json
  board/
  components/
  doc/
  readme.md

1 #include "board.h"
2 #include "ty_beacon.h"
3 #include "app_dps.h"
4
5 int main( void ){
6
7     u8_mac[6] = {0xdc,0x23,0x4d,0x26,0x5c,0xe6};
8     u8_authkey[17] = "jmF7Ju8J9AASqR9M";
9     u8_pid[9] = "smfaaleb";
10    u8_version = 0x10;//1.0
11    u16_kind = 0x0103;
12
13    ty_beacon_init(mac,authkey,pid,version,kind);
14
15    if(ty_beacon_get_state() == DEVICE_STATE_NOT_PAIED)
16        ty_beacon_start_pairing();
17
18    while(1){
19        app_light_run();
20        ty_beacon_run();
21    }
22}
```

POPULATION & RESOURCES
NATIONAL GEOGRAPHIC SOCIETY

/home/btfz/xxx/0110/light/../../../../xxx/0110/light/app/app_bluebeacon_light_linux/src/main.c [FORMAT=dos] [TYPE=C] [POS=1,1][4%] 02/02/21 - 17:49

直接将 app_bluebeacon_light_linux 中的 main.c、app_dps.c、app_dps.h 加入工程即可（跨平台）

经过上面四步便可实现移植。后续可以用涂鸦智能 APP 去测试各项功能是否齐全、是否稳定、拉距测试。

8、FAQ