Tuya Zigbee 开发指南（TLSR8258 平台）

版本	内容	作者	日期
1.0	初版		2022.7.15
1.1	增加虚拟打印等说明		2022.11.4
1.2	增加管脚对应关系说明 增加向前兼容 OTA 说明		2023.02.28
1.3	增加 appconfig.json 配置项的说明		2023.07.07
1.4	增加 app_config.yaml 替换 appconfig.json 说明		2023.12.22

目录

一、固件烧录和调试 3
1.1 软件和硬件 3
1.2 工具配置 5
1.3 连接芯片 5
1.4 设置地址信息 6
1.5 配置默认参数 6
1.6 烧录固件 7
1.7 固件调试 9
1.8 读取 FLASH 和寄存器数据 13
1.9 异常查看 14
1.10 BDT 工具更新自带 unlock 按钮 17
二、功能介绍 18
2.0 打印调试 18
2.1 管脚对应关系 19
2.2 向前兼容 OTA 说明 19
2.3 新增支持自定义设备信息属性(厂商名称，设备型号) 20
若有任何开发框架与产品开发包使用问题，请在涂鸦开发者论坛留言提问 23

一、固件烧录和调试

1.1 软件和硬件

工具下载地址： http://wiki.telink-semi.cn/wiki/IDE-and-Tools/Burning-and-Debugging-Tools-for-all-Series/

安装成功后图标

需要泰凌官方烧写器:

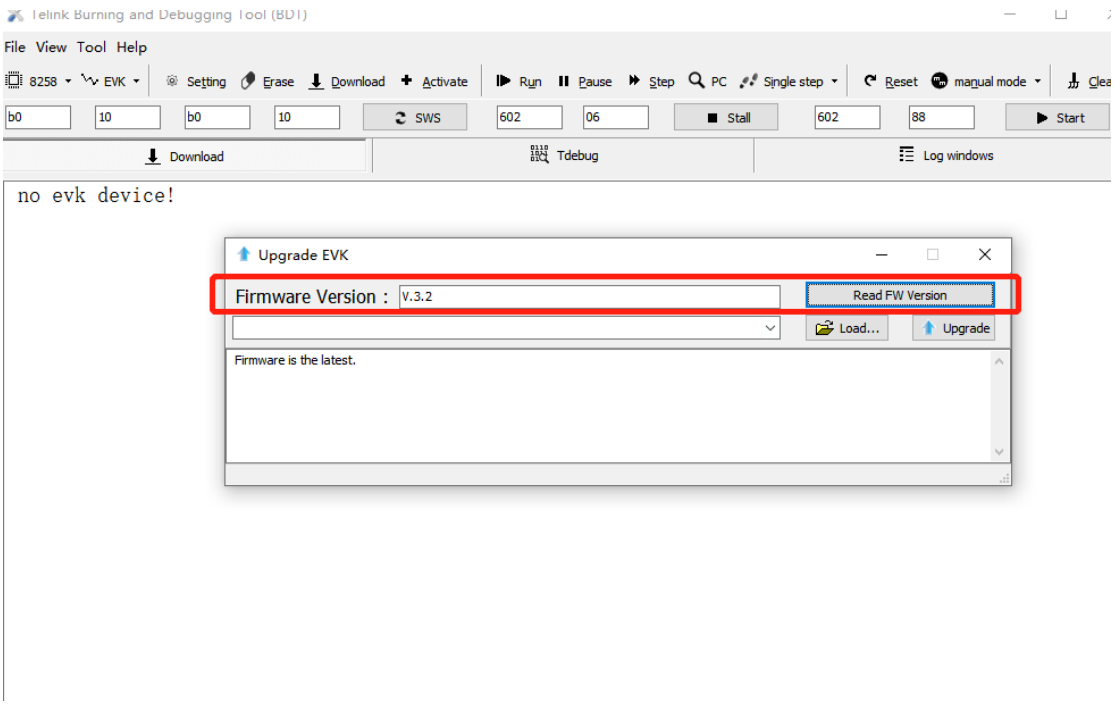
烧写器与模组之间的管脚连接，通常情况下只需要连接 2 个管脚即可。

烧写器	模组
SWM	SWS

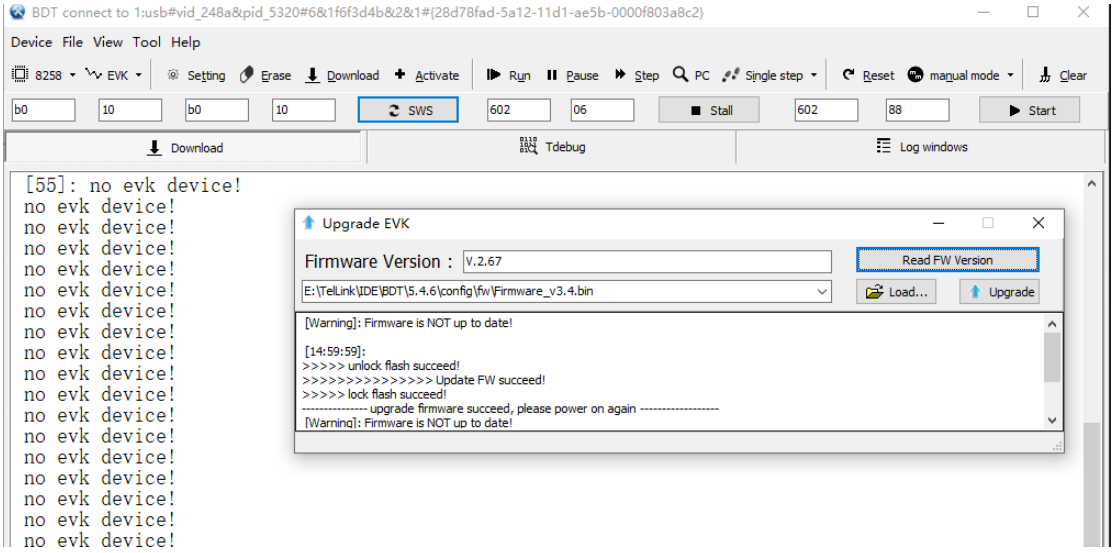
GND	GND

BDT 升级：3.2 版本以下的有 bug，会导致前 4K flash 区域被异常檫除,使用前请注意检查 Telink 固件版本（每换一个 Telink 烧录器均需要检查一遍固件版本）

找到 BDT 安装路径：XXXX\BDT\release_v5.4.4\config\fw



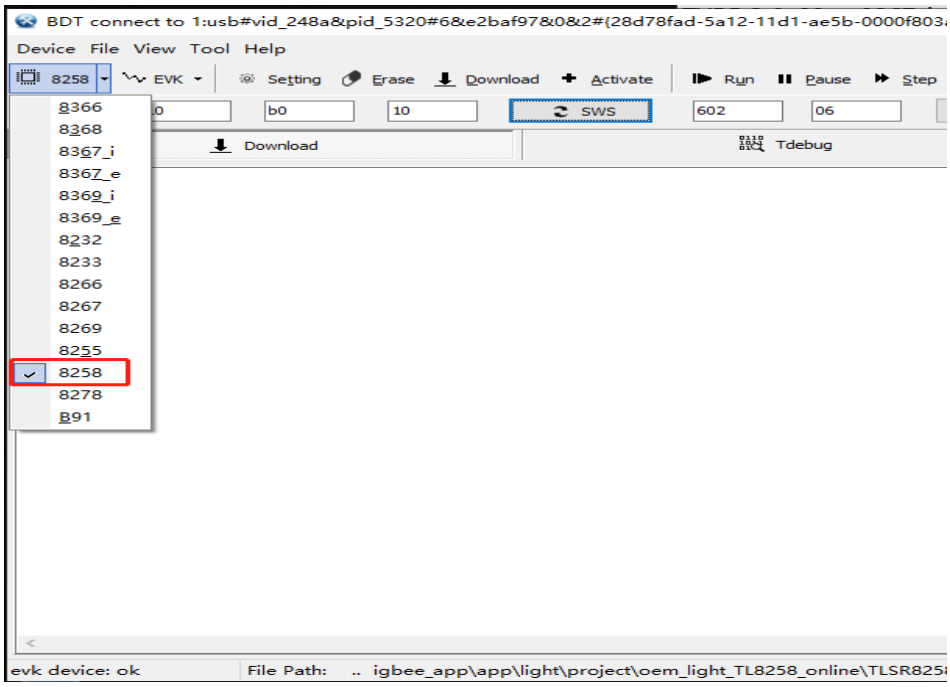
点击 load 选择新版本固件，点击 upgrade 进行升级



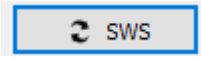
升级成功后将 Telink 烧录器插拔，重新上下电即可

1.2 工具配置

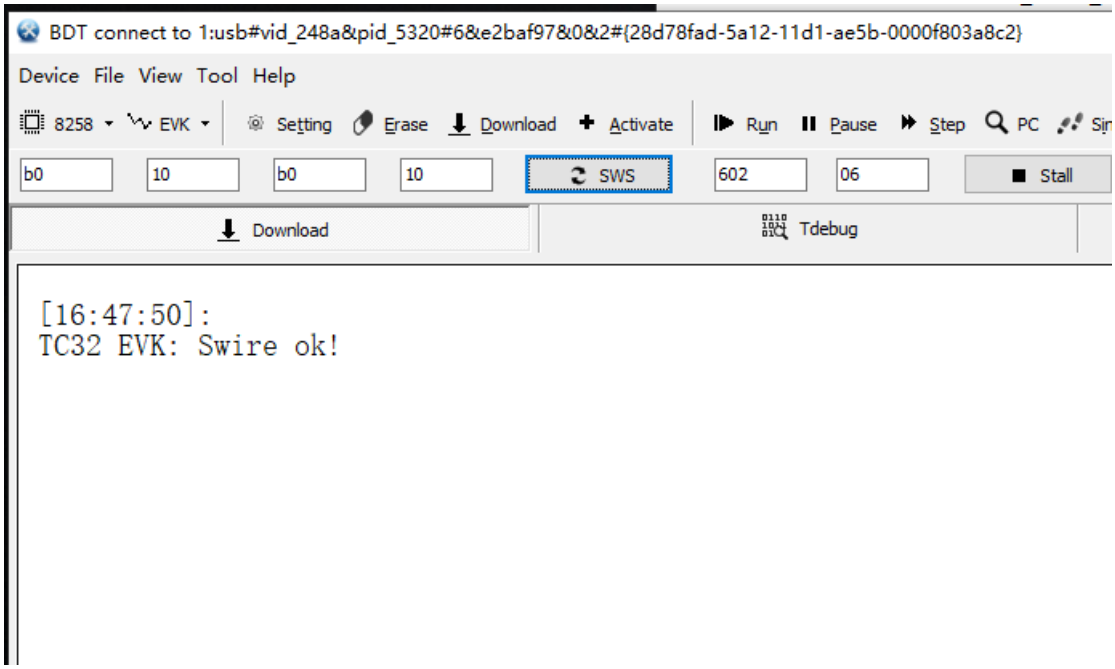
1. 芯片选择 8258,



1.3 连接芯片

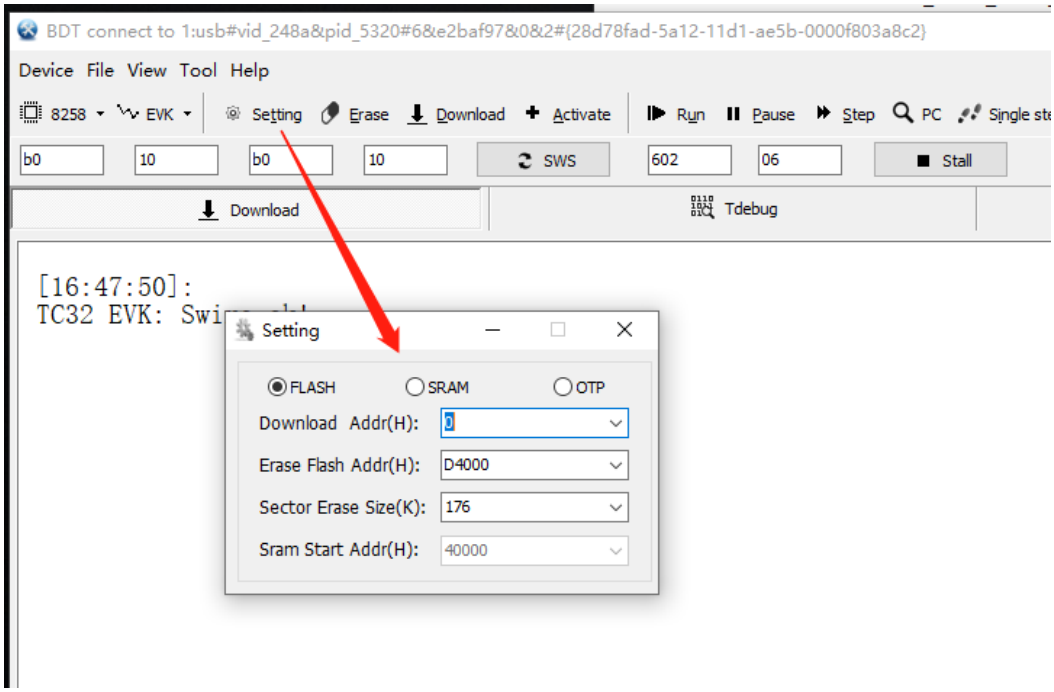
点击 

图标，连接芯片，连接成功会提示 TC32 EVK: Swire ok! ，如果连接失败，可以尝试重新插一下烧录器。



1.4 设置地址信息

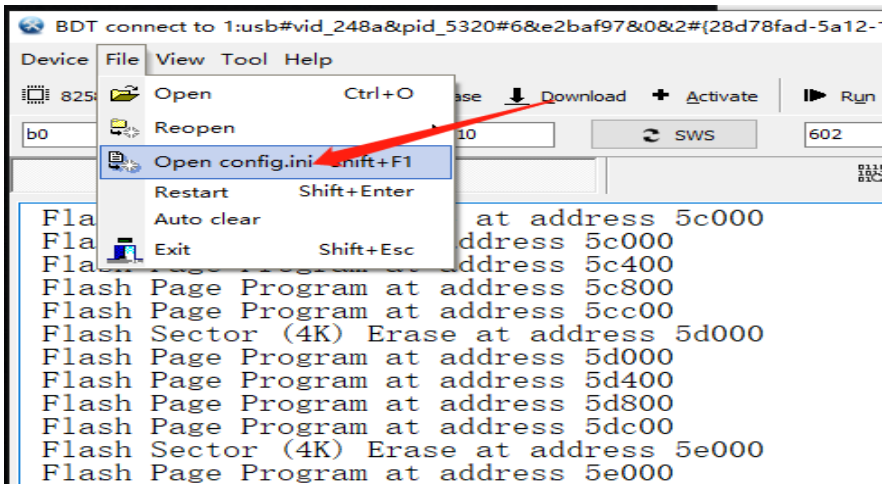
如下图所示，1M flash 的总大小是 0~0x100000,下载地址采用默认地址 0，擦除起始地址可以根据需求填写，如果希望连同固件一并擦除，Erase Flash Addr 可以填 0.



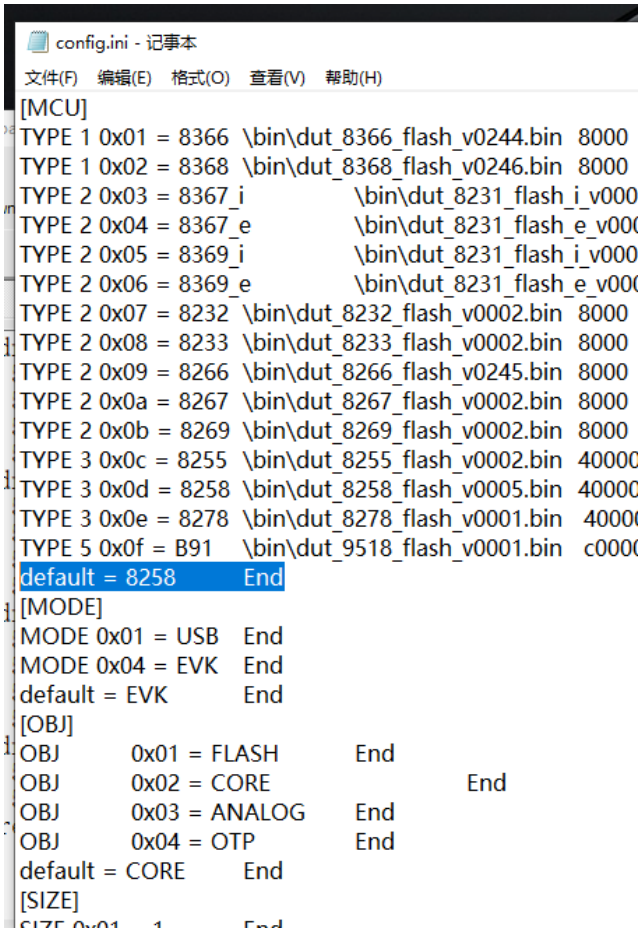
注意：0xFF000 的前 8 个字节的是 zigbee 的 mac 地址，

1.5 配置默认参数

File > Open config.ini 可以打开软件的配置文件，软件在启动的时候会加载配置文件，这样可以省去每次打开软件后都需要进行一次配置的麻烦



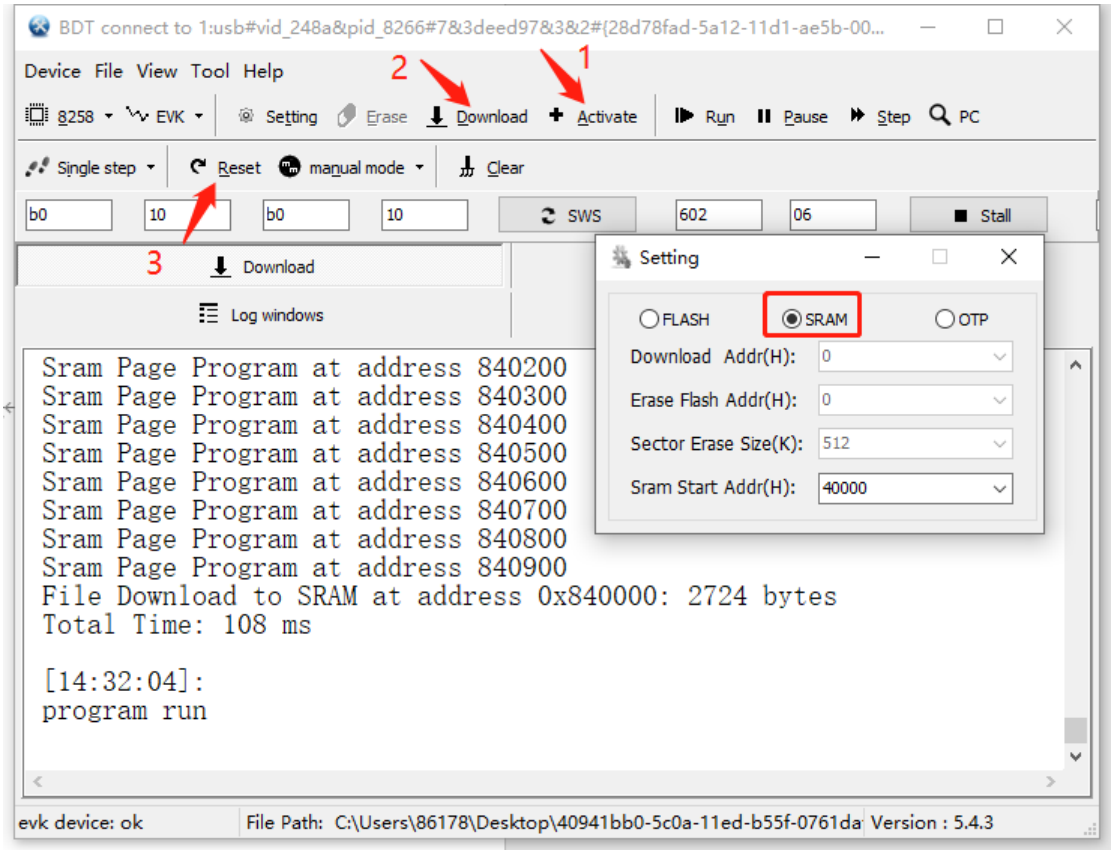
例如，MCU 将 default 改为 8258，这样下次打开软件后就不用在配置 MCU 了，其他参数类似。



1.6 烧录固件

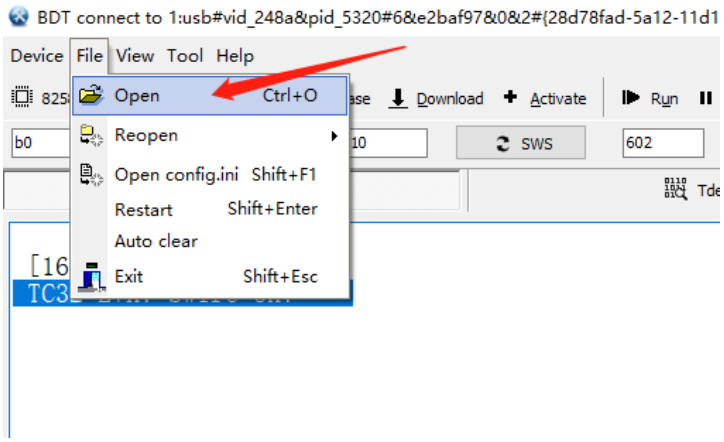
需要注意的是固件 flash 被锁定，再次烧录时需要先解锁模块

解锁固件获取

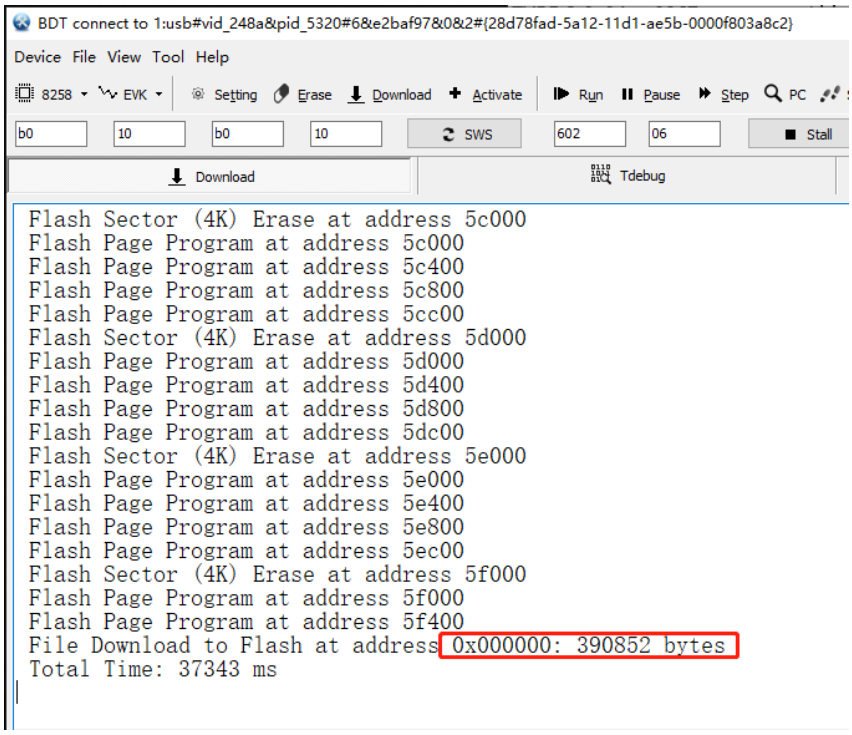


选择该文件后通过 setting 配置为 sram 运行，依次点击 active，download，reset 解锁成功后再按照下列步骤进行固件烧录

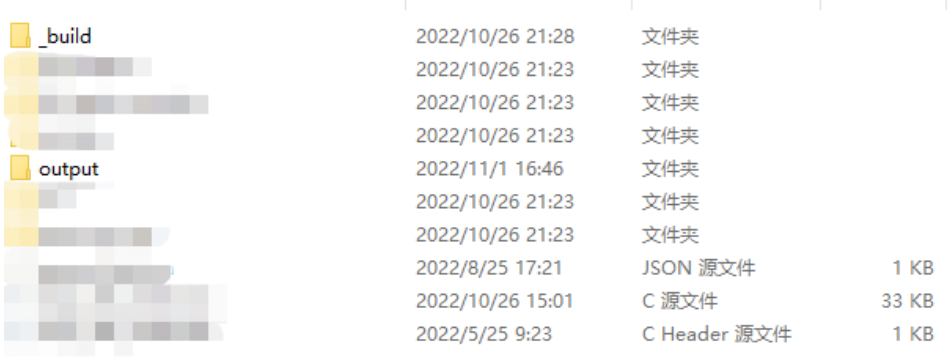
1. 选中要烧录的固件 File -> open，找到要烧录的 Bin 文件



2. 点击 Download 图标开始下载，下载完成后会显示所用时间和地址空间



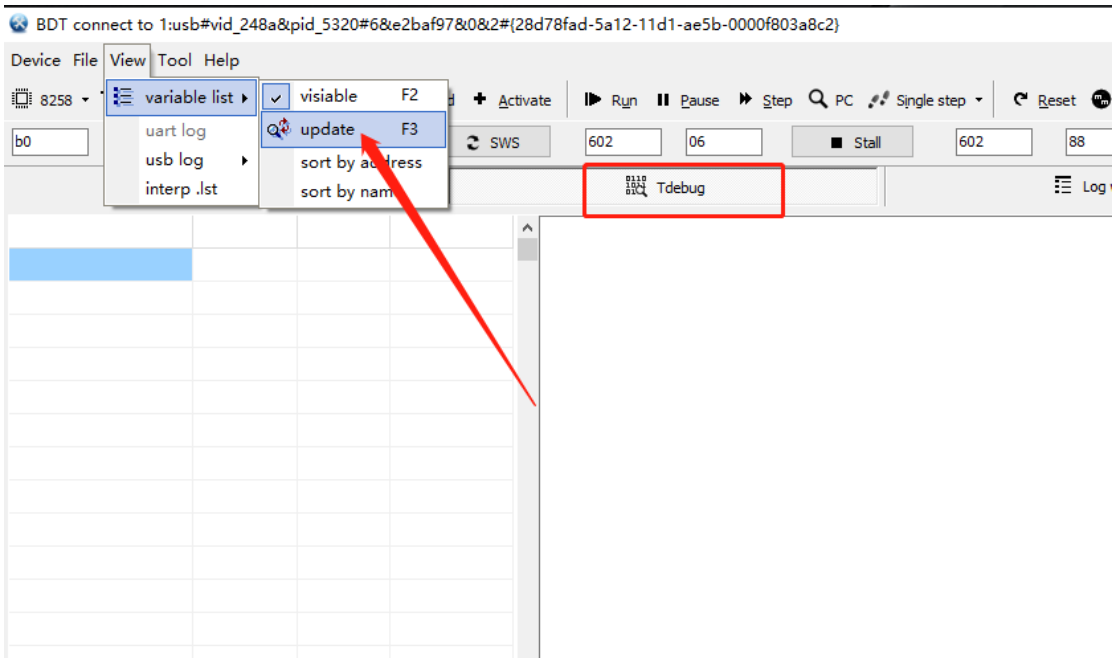
1.7 固件调试



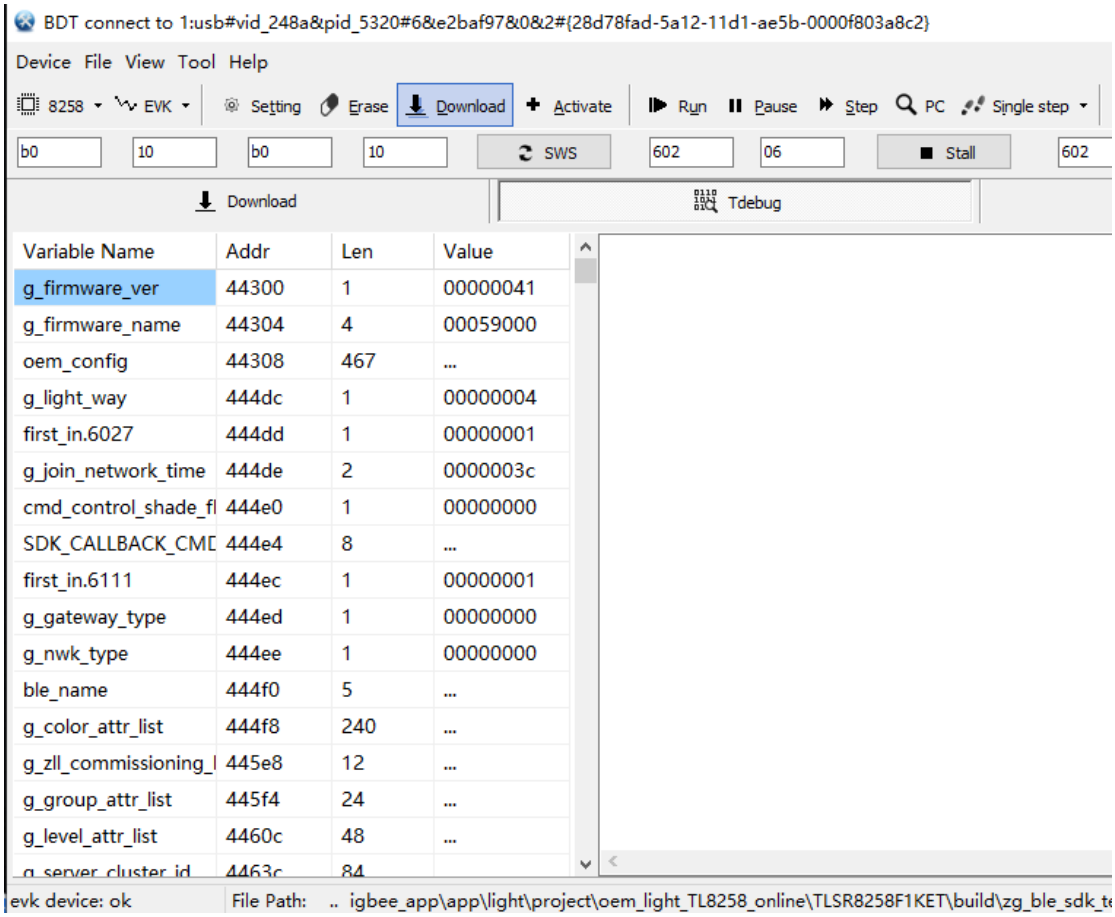
[redacted]	2022/10/26 21:24	文件夹	
[redacted]	2022/10/26 21:24	文件夹	
[redacted]	2022/10/26 21:24	文件夹	
[redacted]	2022/10/26 21:25	文件夹	
vendor	2022/10/26 21:24	文件夹	
app.mk	2022/10/26 21:28	Makefile 源文件	79 KB
boot-.lst	2022/10/26 21:28	LST 文件	5,773 KB
build.log	2022/10/26 21:28	文本文档	8 KB
components.mk	2022/10/26 21:28	Makefile 源文件	19 KB
include.mk	2022/10/26 21:28	Makefile 源文件	1 KB
libgcc_s_dw2-1.dll	2022/10/26 21:24	应用程序扩展	123 KB
libiconv-2.dll	2022/10/26 21:24	应用程序扩展	1,041 KB
libintl-8.dll	2022/10/26 21:24	应用程序扩展	120 KB
libs.mk	2022/10/26 21:28	Makefile 源文件	8 KB
libwinpthread-1.dll	2022/10/26 21:24	应用程序扩展	67 KB
make.exe	2022/10/26 21:24	应用程序	234 KB
makefile	2022/10/26 21:28	文件	2 KB
objects.mk	2022/10/26 21:28	Makefile 源文件	1 KB
sources.mk	2022/10/26 21:28	Makefile 源文件	1 KB
tuyaos_8258_1M_cw_light.elf	2022/10/26 21:28	ELF 文件	485 KB
vendor.mk	2022/10/26 21:28	Makefile 源文件	84 KB

点击进入目录，找到 boot.lst 文件，该文件是最近一次编译产物，每次编译后 QIO 和 UG 文件放在 output 路径下，请注意 boo.lst 每次编译成功都会覆盖，如需要进行调试，请将该 boot.lst 文件复制到 output 路径下

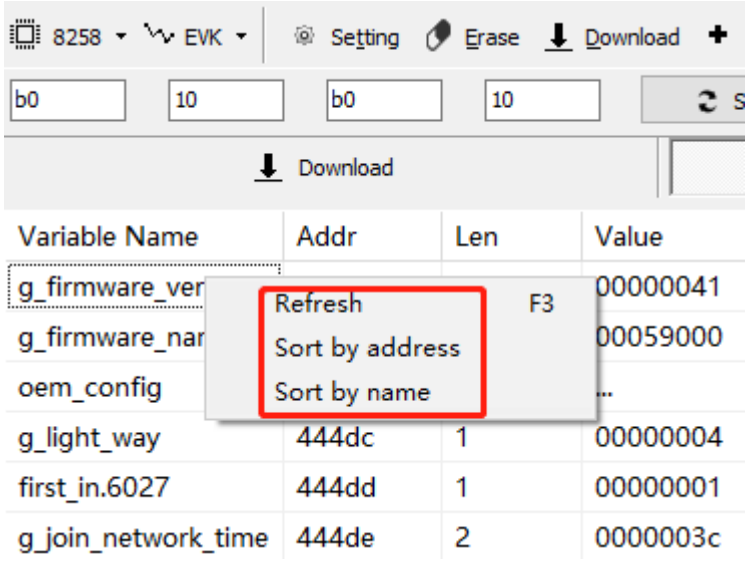
1.点击 Tdebug 按钮，进入调试窗口
2.点击 View > variable list > update 刷新变量的值



刷新后如下图所示，会在左侧窗口中展示全部变量值，



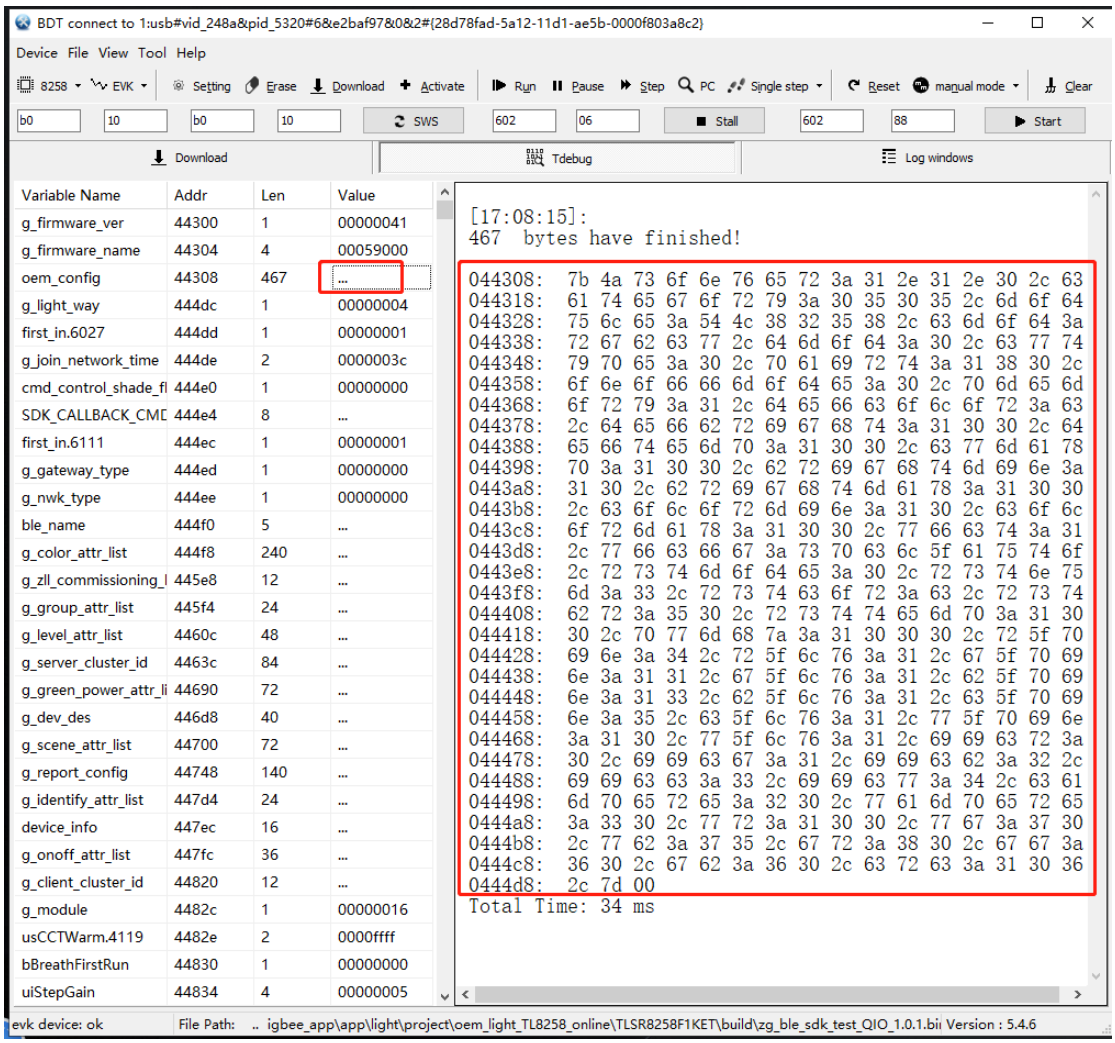
4. 在左侧窗体中右键，可以选择变量的排序方式



之前我们已经介绍了 boot.lst 文件，也可以根据变量的名称找到变量的地址，然后在这个变

量展示窗体中找到对应的变量

对于数据长度超过 4 字节的，变量值会显示为 …， 只需双击三个点，即可在右侧窗体中读取变量的值，如下所示。

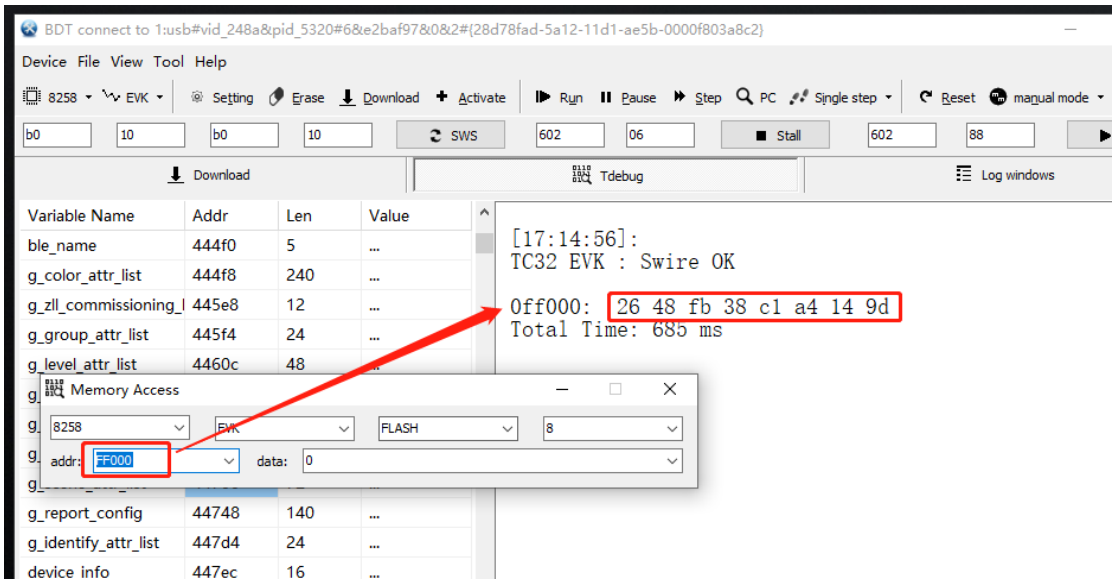


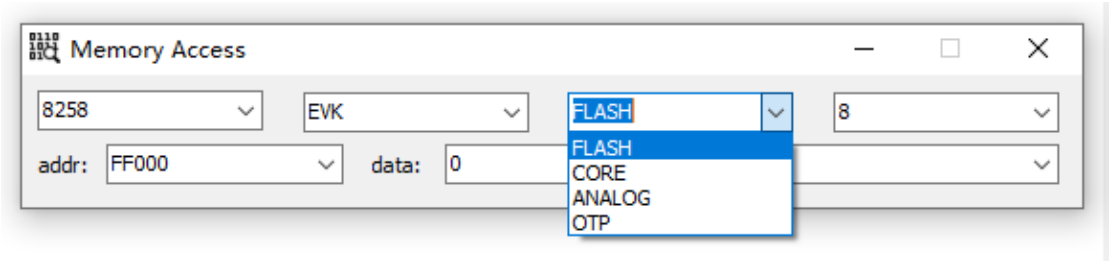
5. 程序运行 点击 Reset 按钮，固件开始运行

固件运行过程中，可以通过按 F3 刷新变量的值(注意此时需要左侧窗体处于被选中状态，鼠标点击左侧窗体任意位置即可)

1.8 读取 FLASH 和寄存器数据

1. 点击 Tool > Memery Access，出现地址访问窗口，如下图所示。
2. 假如我们此时要访问地址 0xFF000（Zigbee mac 地址），则选中 Flash,；选择要读取的数据长度 8；在 addr 输入要读取的地址 FF000,(不需要填 0x),按键盘 Tab 按键
3. 地址 0xFF000 开始的 8 字节数据会展现在右侧窗体，如图 26 48 fb c1 a4 14 9d
   注意：程序中会进行转换，zigbee 实际地址是 a4 c1 38 fb 48 26 9d 14， 且前 6 字节用作 BLE mac 地址。





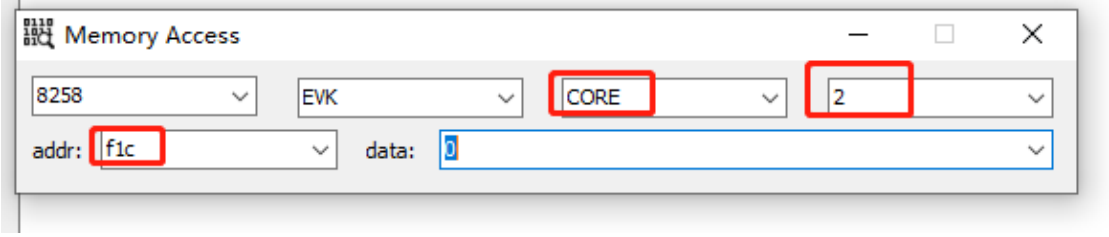
4. 读取寄存器值

通过查询芯片手册，或者直接从原厂代码中可以了解到寄存器地址，例如 register_8258.h 头文件中，有 IRQ_mask 寄存器的地址：0xf1c

```
709
710   #define reg_rf_irq_mask          REG_ADDR16(0xf1c)
711   #define reg_rf_irq_status        REG_ADDR16(0xf20)
712   #define reg_rf_fsm_timeout       REG_ADDR32(0xf2c)
713
```

则可以在地址访问窗体中，选中 core (寄存器)，长度 2，addr: f1c，再按键盘 Tab 键，即可读到此地址的值，03 00.

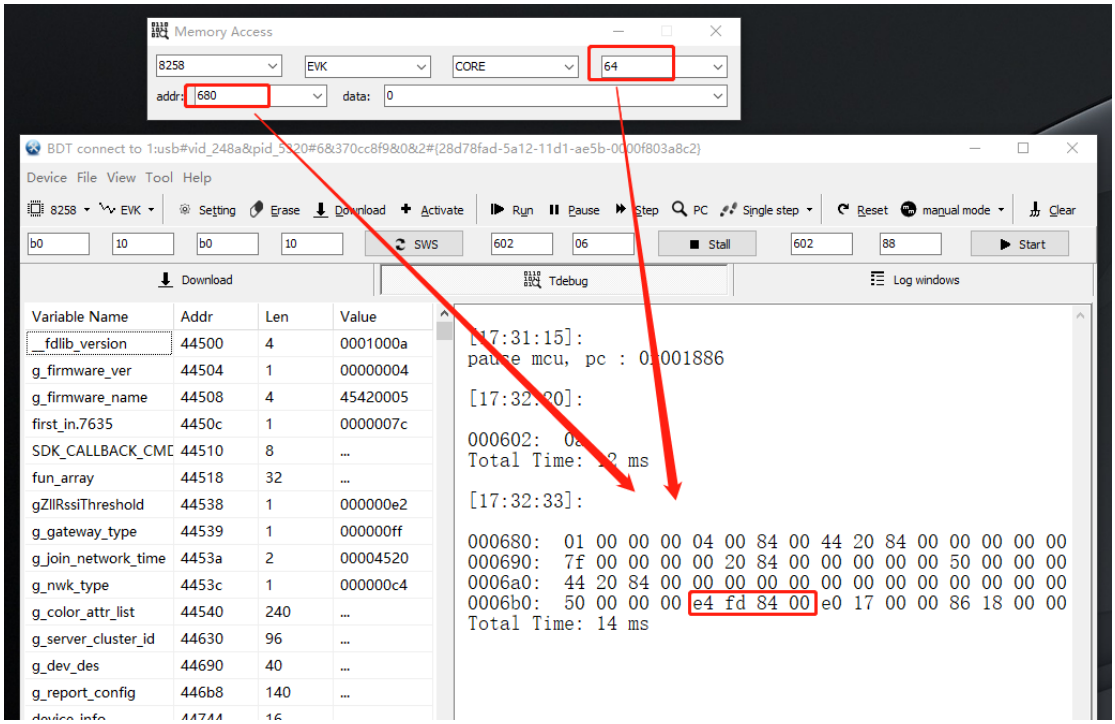
```
000f1c:  03 00
Total Time: 13 ms
```



1.9 异常查看

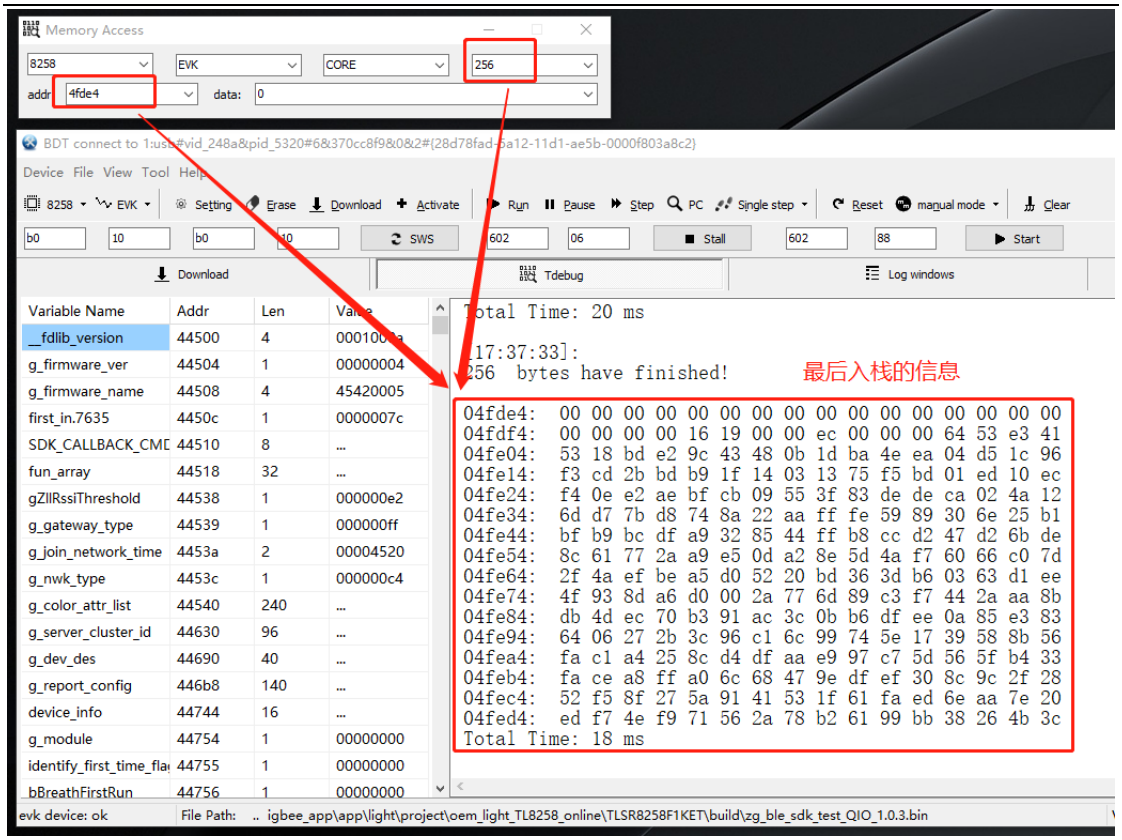
当程序因为某些原因复位了，或者因为某些原因跑飞了，可以通过 BDT 工具查看栈的情况来初步调查程序产生异常的位置。在进行问题定位时，可以临时将看门狗关闭。

1. 点击 Pause，让程序暂停。
2. 依次点击 Tool -》Memory Access，根据第 2.8 节的方法读取 680 位置，64 字节的数据并定位到如下位置，这个地址就是当前栈位置，可以通过这个地址再读取其附近的数据来分析栈的情况。



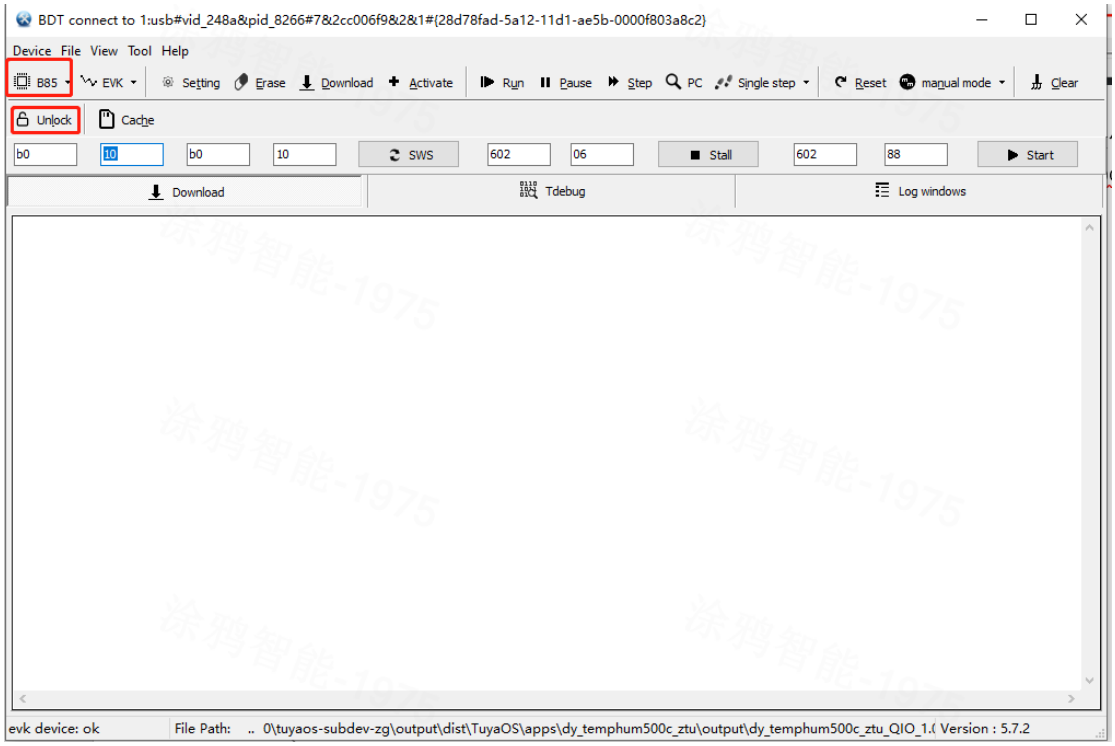
3. 读取最后入栈的数据，栈数据里面是函数参数，函数返回地址，以及一些局部变量等，可以通过每 4 字节，找出一个个入栈的函数地址，结合 Boot.lst 文件进行查找地址，定位到具体的函数。

Tips: 栈中的数据都是小端格式的，并且函数一般存在 Flash 中（部分在 RAM 中），故通常其地址比较小，因此想要找到一个函数地址，可以每隔 4 字节找一个地址比较小的数据，再到 boot.lst 文件中进行查找。



如上图所示，栈中信息包含了很多数据，每 4 字节，可能是函数地址也可能是变量值，通常 00 84 开头的是变量，较小的数据开头的是函数地址。提取函数地址后，可以到 boot.lst 文件中进行检索，这样一个一个的函数地址可以分析出整个 callstack.

1.10 BDT 工具更新自带 unlock 按钮



芯片型号选择 B85

有需要可下载使用

二、功能介绍

2.0 打印调试

在 tuya_init_first 与 tuya_init_last 都需要添加 tal_uart_init 函数
推荐 telink 使用 gpio 模拟串口打印

```
#include "tlprintf.h"
#include "tkl_platform_types.h"

OPERATE_RET tal_uart_init(TUYA_UART_NUM_E port_id, TAL_UART_CFG_T
*cfg);

VOID_T app_uart_init(VOID_T)
{
        TAL_UART_CFG_T cfg = {
        .rx_buffer_size = 128,/*rx 接受 buf 大小*/
        .base_cfg.baudrate = 115200,/*波特率*/
        .base_cfg.parity = TUYA_UART_PARITY_TYPE_NONE,/*奇偶校验位*/
        .base_cfg.databits = TUYA_UART_DATA_LEN_8BIT,/*数据位*/
        .base_cfg.stopbits = TUYA_UART_STOP_LEN_1BIT,/*停止位*/
        .base_cfg.flowctrl = TUYA_UART_FLOWCTRL_NONE,/*流控标志位*/
    };
    tal_uart_init(TUYA_UART_NUM_0, &cfg);

    tal_log_create_manage_and_init(TAL_LOG_LEVEL_DEBUG, 128, Tl_printf);
}

BOOL_T app_print_get_cfg(UINT8_T *print_type, UINT8_T *disable_irq,
GPIO_PORT_PIN_T *gpio)
{
    gpio->port = PORT_C;/* 模拟串口打印的引脚*/
    gpio->pin = PIN_0;
    *print_type = 0;/*0 表示 gpio 模拟打印，1 表示串口打印*/
    *disable_irq = 1;/*默认使用 gpio 模拟打印时关闭中断*/
    #ifdef UART_PRINTF_MODE
      return 1;/*开始模拟串口打印*/
    #else
      return 0;/*关闭模拟串口打印*/
    #endif
}
```

2.1 管脚对应关系

TLSR8258 平台对于 tal_gpio.h 文件中包含的函数中，TUYA_GPIO_NUM_E pin_id 是通过映射如下 list 进行管脚操作的

```
43  STATIC CONST UINT32_T tkl_gpio_list[] = {
44      GPIO_PA0, GPIO_PA1, GPIO_PA2, GPIO_PA3, GPIO_PA4, GPIO_PA5, GPIO_PA6, GPIO_PA7, // 0 - 7 //
45      GPIO_PB0, GPIO_PB1, GPIO_PB2, GPIO_PB3, GPIO_PB4, GPIO_PB5, GPIO_PB6, GPIO_PB7, // 8 - 15 //
46      GPIO_PC0, GPIO_PC1, GPIO_PC2, GPIO_PC3, GPIO_PC4, GPIO_PC5, GPIO_PC6, GPIO_PC7, // 16 - 23 //
47      GPIO_PD0, GPIO_PD1, GPIO_PD2, GPIO_PD3, GPIO_PD4, GPIO_PD5, GPIO_PD6, GPIO_PD7, // 24 - 31 //
48      GPIO_PE0, GPIO_PE1, GPIO_PE2, GPIO_PE3, // 32 - 35//
49  };
```

比如 TUYA_GPIO_NUM_0 对应的是 PA0，以此类推

2.2 向前兼容 OTA 说明

客户通过 涂鸦 IOT 平台 创建的以及基于 ty_iot_zigbee_sdk_tlsr8258 开发的 Zigbee ZT 系列固件，支持直接 OTA 到 TuyaOS-Zigbee-TLSR8258 3.6.0 及以上版本

TUYAOS					TELINK SDK			
0x100000	内容	size	说明		0x100000	内容	size	说明
0xFF000	mac	4K			0xFF000	mac	4K	
0xFE000	factory pre cfg	4K			0xFE000	factory pre cfg	4K	
0xFD000	install code	4K			0xFD000	install code	4K	
0xFC000	factory rst cnt	4K			0xFC000	factory rst cnt	4K	
0xFB400	reserve	3K			0xFB400	reserve	3K	
0xFB000	user flash1	1K	用来存储产测数据		0xFB000	mf test	1K	用来存储产测数据
0xF9000	reserve	8K	应用层存储数据用		0xF9000	user flash	8K	应用层存储数据用
0xF8000	user flash	4K	用来存储免开发配置项		0xF8000	oem cfg	4K	用来存储免开发配置项
0xF7000	ota cfg	4K	ota结束时写入boot loader会进行解析		0xF7000	ota cfg	4K	ota结束时写入boot loader会进行解析
					0xF6000	ble info(secure)	4K	ble 配对信息
0xF4000	reserve	12K			0xF4000	tuya ble info	8K	TUYA ble信息
0xF3000	TYOS_INFO	4K	3.6.X升级兼容信息区域					
0xEE000	reserve	16K						
0xE8000	tal_nv	24K			0xF0000	reserve	16K	
0xD8000	sdk_nv	64K			0xD8000	nv	96K	
0x70000	ota image	416K			0x70000	ota image	416K	
0x08000	firmware	416K			0x08000	firmware	416K	
0x00000	bootloader	32K			0x00000	bootloader	32K	

sdk_nv	内容	size	说明
0xEC000	MODULE_TABLE	8K	
0xEA000	MODULE_ATTR	8K	
0xE8000	MODULE_USER	8K	
0xE6000	KEYPAIR	8K	
0xE4000	APP	8K	
0xE2000	OTA	8K	
0xE0000	NWK_FRAME_COUNT	8K	
0xDE000	ZCL	8K	
0xDC000	APS	8K	
0xDA000	ADDRESS_TABLE	8K	
0xD8000	ZB_INFO	8K	

nv	内容	size	说明	
0xF0000	reserve	16K		
0xEE000	SDK	8K	net_info,base_info,error_code	
0xEC000	ATTR	8K		
0xEA000	FLASH	8K	user_flash+raw_flash	250+N
0xE8000	SCENE	8K		
0xE6000	KEYPAIR	8K		
0xE4000	APP	8K		
0xE2000	OTA	8K		
0xE0000	NWK_FRAME_COUNT	8K		
0xDE000	ZCL	8K		
0xDC000	APS	8K		
0xDA000	ADDRESS_TABLE	8K		
0xD8000	ZB_INFO	8K		

FLASH 划分对比

1. OTA 后固件网络信息保留，仍在网
2. 占用 0xF3000~0xF4000 的 4K 数据区域用于版本标志等，请勿复写
3. 0xE8000~0xEE000 的 TAL_NV 数据是数据格式转换后的数据，可以通过 tal_nv_flash_inner.h 中相关函数进行数据读取
4. Telink sdk 开发的固件中 user_flash（user_flash_data_write，user_flash_data_read 函数读写区域），raw_flash（flash_block_raw_write，flash_block_raw_read 函数读写区域）统一需要通过 tal_nv_flash.h 中的函数进行数据读取，user_flash 对应的 item id 是 0xFE，BLOCK_0~BLOCK_16 对应的 item id 是 0xFD~0xED
5. 涂鸦体系中产测信息，自恢复功能等均可以正常生效，其中免开发配置项保留在相同区域格式不变，仍可以通过原有接口进行解析

2.3 新增支持自定义设备信息属性(厂商名称，设备型号)

1.TLSR8258 平台 3.8.0 版本开始支持用户配置 basic 下的 0004 属性 *ManufacturerName* 和 0005 属性 *ModelIdentifier* 以满足开发者需要接入第三方网关时自定义厂商名称，设备型号的需求。

```
{
    "firmwareInfo": {
        "description": "this is a demon project",
        "dev_role":"router",
        "image_type":"0xD3A3",
        "manufacture_id":"0x1141",
        "model_id":"custom model id",
        "manufacture_name": "custom manufacture name",
        "tuya_product_id": "hxmtspzj",
        "tuya_capacity": "_TZ3210_",
        "tuya_product_type": "TS0502B",
        "module_name":"ZT3L",
        "chip_id":"TLSR8258F1KET"
    }
}
```

2.这种方式 "model_id" 和 "manufacture_name" 关键字长度上限是 32 字节。在这种配置方案下，涂鸦相关内容比如 pid，能力值和设备类型会迁入 basic 下 FFC0 属性，高版本涂鸦网关入网时会读取 FFC0 属性，获取相应设备面板，实现涂鸦网关和第三方网关接入兼容。

3.但是需要注意的是如果接入涂鸦网关暂时不支持读取 FFC0 属性或者是没有自定义厂商号设备型号的需求，则上述属性建议填入 涂鸦信息（默认做法）,如下图所示：
"manufacture_name" = "tuya_capacity" + "tuya_product_id"
"model_id" = "tuya_product_type"

```
{
    "firmwareInfo": {
        "description": "this is a demon project",
        "dev_role":"router",
        "image_type":"0xD3A3",
        "manufacture_id":"0x1141",
        "model_id":"TS0502B",
        "manufacture_name": "_TZ3210_hxmtspzj",
        "tuya_product_id": "hxmtspzj",
        "tuya_capacity": "_TZ3210_",
        "tuya_product_type": "TS0502B",
        "module_name":"ZT3L",
        "chip_id":"TLSR8258F1KET"
    }
}
```

4. 脚本向前兼容没有"tuya_product_id" "tuya_capacity" "tuya_product_type" 的 json 配置。认为是不需要自定义厂商号和设备型号，会按照（3）中提到的规则自动生成正确的 app_config.h

```
{
    "firmwareInfo": {
        "description": "this is a demon project",
        "dev_role":"sleep_end_dev",
        "module_name":"ZT3L",
        "chip_id":"TLSR8258F1KET",
        "image_type":"0xD3A3",
        "manufacture_id":"0x1141",
        "model_id":"TS0225",
        "pid": "revfnetb",
        "manufacture_name": "_TZ3210_"
    }
}
```

5. TLSR8258 TuyaOS 3.8.0 版本底层新增了 basic 下 FFC0 属性用于迁入涂鸦 pid，能力值和设备类型这些信息；比如开发者使用 3.7.1 版本开发了一个固件，OTA 到 3.8.0 开发的固件后，底层会自动将烧录授权后的信息更新到 FFC0 属性，保证烧录授权信息不丢失。同时会

按照 app_config.h 中 APP_MANUFACTURE_NAME 宏是否以 "_TZ" 开头同时第 7 个字节是‘_‘ 。如果是的话那么是按照涂鸦网关接入方式，将烧录授权后的信息更新到 0004 属性 *ManufacturerName* 和 0005 属性 *ModelIdentifier* 中。否则是按照自定义设备信息属性方式将 config.h 中内容同步到 0004 属性 *ManufacturerName* 和 0005 属性 *ModelIdentifier* 中。

2.4 3.9.0 版本用 app_config.yaml 文件替换 appconfig.json 的说明

yaml 相比 json 格式文件，具有可以添加注释，格式更加简洁等优势，因此我们在 3.9.0 版本后将使用 app_config.yaml 文件，替换 appconfig.json 文件

同时在 泰凌平台，将 3.9.0 之前的工程放在 app 路径下进行编译时，脚本会自动进行检测，如果工程中只有 appconfig.json，没有 app_config.yaml 文件，则会自动将 appconfig.json 转换为 app_config.yaml 文件，实现向前兼容。

但是由于脚本处理 yaml 格式不够完善，转换之后的 app_config.yaml 缺乏可读性，建议使用者还是在理解的基础上进行确认，避免造成问题。

注意 manufactured_name 整合了原来 appconfig.json 的 pid 关键字

```
Firmware_Information:
  description: "this is a demo project"
  device_role: "router"
  image_type: 0xD3A3
  manufacture_id: 0x1141
  model_id: "TS0502B"
  manufacture_name: "_TZ3210_hxmtspzj"
  module_name: "ZT3L"
  chip_id: "TLSR8258F1KET"
```

```
Firmware_Information:
  description: "this is a demo project"
  device_role: "router"
  image_type: 0xD3A3
  manufacture_id: 0x1141
  model_id: "custom model id"
  manufacture_name: "custom manufacture name"
  product_id: "hxmtspzj"
  capacity: "_TZ3210_"
  product_type: "TS0502B"
  module_name: "ZT3L"
  chip_id: "TLSR8258F1KET"
```

product_id 为涂鸦设备 pid；capacity 为涂鸦设备能力值；product_type 为涂鸦设备类型

若有任何开发框架与产品开发包使用问题，请在涂鸦开发者论坛留言提问