

# AIC U-BOOT BLE WAKEUP USER-GUIDE

## 1、对应模组型号选择

在aicwf\_usb.h文件中，修改对应宏定义。

当使用的是8800D系列芯片模组时：

```
#define AIC_SUPPORT_8800D    1
#define AIC_SUPPORT_8800D80 0
```

当使用的是8800D80/D40系列芯片模组时：

```
#define AIC_SUPPORT_8800D    0
#define AIC_SUPPORT_8800D80 1
```

## 2、adv data filter配置

adv data filter的配置8800D与8800D40/D80系列配置方式一致，对应配置的代码位置不同。

8800D系列在aicwf\_usb.c文件内，aicfw\_download\_fw\_8800函数中。

8800D80/D40系列在aic\_compat\_8800d80.c文件内，

aicfw\_download\_fw\_8800d80函数中。

```
..wakeup_param->magic_num=-0x53454C42; // magic_num
..wakeup_param->delay_scan_to=-1000; // delay start scan time(ms)
..wakeup_param->reboot_to=-ble_scan_wakeup_reboot_time; // reboot time
..// =====
..// gpio_trigger_idx: 0: .. if wakeup_param->gpio_dft_lv[0]=0xfe, this idx will be invalid.
..wakeup_param->gpio_num[0]=gpio_num; // default select gpio2 for fw_wakeup_host
..wakeup_param->gpio_dft_lv[0]=gpio_dft_lv; // 0: default pull down, 1: default pull up
..// gpio_trigger_idx: 1: .. if wakeup_param->gpio_dft_lv[1]=0xfe, this idx will be invalid.
..wakeup_param->gpio_num[1]=-3; // default select gpio2 for fw_wakeup_host
..wakeup_param->gpio_dft_lv[1]=-1; // 0: default pull down, 1: default pull up
..// =====
..// MAX_AD_FILTER_NUM=5: num 0
{
    const uint8_t data[11] = {0x59, 0x4B, 0x32, 0x42, 0x41, 0x5F, 0x54, 0x45, 0x53, 0x54, 0x33};
    wakeup_param->ad_filter[0].ad_len = 12;
    wakeup_param->ad_filter[0].ad_type = 0x09;
    memcpy(wakeup_param->ad_filter[0].ad_data, data, wakeup_param->ad_filter[0].ad_len-1); // 1111 1111 1110 0000 0000 0000 0000 0000 // 0xfe000000
    wakeup_param->ad_filter[0].ad_data_mask = 0xffe00000;
    wakeup_param->ad_filter[0].ad_role = ROLE_COMBO1(COMBO_0<<4);
    wakeup_param->ad_filter[0].gpio_trigger_idx = TG_IDX_0; // 0: match for wakeup_param->gpio_num[0] ..... 1: match for wakeup_param->gpio_num[1]
}
..// =====
..// MAX_AD_FILTER_NUM=5: num 1
{
    const uint8_t data[2] = {0x12, 0x18};
    wakeup_param->ad_filter[1].ad_len = 3;
    wakeup_param->ad_filter[1].ad_type = 0x3;
    memcpy(wakeup_param->ad_filter[1].ad_data, data, wakeup_param->ad_filter[1].ad_len-1); // 1100 0000 0000 0000 0000 0000 0000 0000 // 0xc0000000
    wakeup_param->ad_filter[1].ad_data_mask = 0;
    wakeup_param->ad_filter[1].ad_role = ROLE_COMBO1(COMBO_0<<4);
    wakeup_param->ad_filter[1].gpio_trigger_idx = TG_IDX_0; // 0: match for wakeup_param->gpio_num[0] ..... 1: match for wakeup_param->gpio_num[1]
}
..// =====
..// MAX_AD_FILTER_NUM=5: num 2
{
    const uint8_t data[11] = {0x59, 0x4B, 0x32, 0x42, 0x41, 0x5F, 0x54, 0x45, 0x53, 0x54, 0x33};
    wakeup_param->ad_filter[2].ad_len = 0;
    wakeup_param->ad_filter[2].ad_type = 0;
    memcpy(wakeup_param->ad_filter[2].ad_data, data, wakeup_param->ad_filter[2].ad_len-1); // 1100 0000 0111 1111 1100 0000 0000 0000 // 0xc07fc000
    wakeup_param->ad_filter[2].ad_data_mask = 0;
    wakeup_param->ad_filter[2].ad_role = ROLE_ONLY;
    wakeup_param->ad_filter[2].gpio_trigger_idx = TG_IDX_0; // 0: match for wakeup_param->gpio_num[0] ..... 1: match for wakeup_param->gpio_num[1]
}
..// =====
..// MAX_AD_FILTER_NUM=5: num 3
{
    const uint8_t data[11] = {0x59, 0x4B, 0x32, 0x42, 0x41, 0x5F, 0x54, 0x45, 0x53, 0x54, 0x33};
    wakeup_param->ad_filter[3].ad_len = 0;
    wakeup_param->ad_filter[3].ad_type = 0;
    memcpy(wakeup_param->ad_filter[3].ad_data, data, wakeup_param->ad_filter[3].ad_len-1); // 1100 0000 0111 1111 1100 0000 0000 0000 // 0xc07fc000
    wakeup_param->ad_filter[3].ad_data_mask = 0;
    wakeup_param->ad_filter[3].ad_role = ROLE_COMBO1(COMBO_1<<4);
    wakeup_param->ad_filter[3].gpio_trigger_idx = TG_IDX_0; // 0: match for wakeup_param->gpio_num[0] ..... 1: match for wakeup_param->gpio_num[1]
}
..// =====
..// MAX_AD_FILTER_NUM=5: num 4
{
    const uint8_t data[11] = {0x59, 0x4B, 0x32, 0x42, 0x41, 0x5F, 0x54, 0x45, 0x53, 0x54, 0x33};
    wakeup_param->ad_filter[4].ad_len = 0;
    wakeup_param->ad_filter[4].ad_type = 0x09;
    memcpy(wakeup_param->ad_filter[4].ad_data, data, wakeup_param->ad_filter[4].ad_len-1); // 1111 1111 1110 0000 0000 0000 0000 0000 // 0xfe000000
    wakeup_param->ad_filter[4].ad_data_mask = 0xffe00000;
    wakeup_param->ad_filter[4].ad_role = ROLE_COMBO1(COMBO_1<<4);
    wakeup_param->ad_filter[4].gpio_trigger_idx = TG_IDX_0 | TG_IDX_1; // 0: match for wakeup_param->gpio_num[0] ..... 1: match for wakeup_param->gpio_num[1]
}
```

参数介绍：

### a) 初始化参数介绍

**wakeup\_param->delay\_scan\_to**: 下载好配置固件之后到真正开启BLE scan的delay时间, 默认1000ms

**wakeup\_param->reboot\_to**: 配置的adv data trigger后, 模组持续拉高或拉低的时间(依赖默认电平取反), 时间到了之后模组自动reboot, 恢复到boot阶段, 重新等待被usb枚举。

reboot\_to参数: 分为高2bit以及低30bit, 或者全0, 分别表征不同含义。

**高2bit**: 0: 时间到之后clean gpio到默认电平状态, 并关闭scan,相当于只能trigger一次gpio。

1: 时间到之后clean gpio到默认电平状态, 并关闭scan,相当于只能trigger一次gpio, 并且reboot IC。

2: 时间到之后clean gpio到默认电平状态, 不关闭scan, 可以重复接受匹配的ADV来进行trigger。

**低30bit**: 代表时间, 单位ms。

**全0**: 与高2bit值为2功能一致, 但时间默认为100ms。

```
/******
```

```
///gpio_trigger_idx : 0   if wakeup_param->gpio_dft_lvl[0]=0xfe,this idx will be invalid.
```

```
wakeup_param->gpio_num[0] = gpio_num;////default select gpiob2 for fw_wakeup_host
```

```
wakeup_param->gpio_dft_lvl[0] = gpio_dft_lvl;////0:default pull down, 1:default pull up
```

```
///gpio_trigger_idx : 1   if wakeup_param->gpio_dft_lvl[1]=0xfe,this idx will be invalid.
```

```
wakeup_param->gpio_num[1] = 3;////default select gpiob2 for fw_wakeup_host
```

```
wakeup_param->gpio_dft_lvl[1] = 1;////0:default pull down, 1:default pull up
```

模组默认支持两组GPIO作为trigger信号, 可分别单独使用, 或者同时使用。

**wakeup\_param->gpio\_num**: 代表芯片内模组的gpio号。

**wakeup\_param->gpio\_dft\_lvl**: 代表此gpio默认电平0为默认拉低, 1为默认拉高。

b) adv data filter参数介绍

模组默认支持最多5组标准adv type的过滤 (len | ad\_type | data) 。例如其中一组:

```
{
    const uint8_t data[11] =
{0x59,0x4B,0x32,0x42,0x41,0x5F,0x54,0x45,0x53,0x54,0x33};
    wakeup_param->ad_filter[0].ad_len = 12;
    wakeup_param->ad_filter[0].ad_type = 0x09;
```

```

memcpy(wakeup_param->ad_filter[0].ad_data, data, wakeup_param-
>ad_filter[0].ad_len-1); // 1111 1111 1110 0000 0000 0000 0000 0000 // 0xffe00000
wakeup_param->ad_filter[0].ad_data_mask = 0xffe00000;
wakeup_param->ad_filter[0].ad_role = ROLE_COMBO|(COMBO_0<<4);
wakeup_param->ad_filter[0].gpio_trigger_idx = TG_IDX_0; // 0: match for
wakeup_param->gpio_num[0]    1: match for wakeup_param->gpio_num[1]
}

```

**ad\_data**: 表示 (len | ad\_type | **data**) 中的data。

**ad\_len**: 表示 (len | ad\_type | data) 中的len。

**ad\_type**: 表示 (len | **ad\_type** | data) 中的ad\_type。

**ad\_data\_mask**: 表示对ad\_data有效数据的表示, mask最高位代表data数组第0type, 一次类推。

eg: {0x59,0x4B,0x32,0x42,0x41,0x5F,0x54,0x45,0x53,0x54,0x33}, 如果所有的type都是有效的唤醒词, 则需要将mask中对应的bit全部置为1, mask中低位补0。1111 1111 1110 0000 0000 0000 0000 0000 // 0xffe00000, 然后填写到ad\_data\_mask 中。如果data中仅第0,1,2,5,6,7,8,9,10byte是有效唤醒词, 则配置1110 0111 1110 0000 0000 0000 0000 0000其中byte 3 4对应的bit改为0。后面低位全部补0补齐32bit。转化为0xE7E00000填写到ad\_data\_mask 中。

**ad\_role**: 表示当前这组filter是作为独立匹配即可trigger gpio拉高或拉低还是需要与其他组filter组合匹配才能trigger gpio。

两组参数 **ROLE\_ONLY**: 表示独立即可trigger。

**ROLE\_COMBO**: **ad\_role** 的低4bit, 表示本组要与其他组ad\_role同为**ROLE\_COMBO**的filter组合, 同时匹配时才能trigger gpio。当配置为**ROLE\_COMBO**时, 需要在高4bit配置(**COMBO\_0**<<4)或者(**COMBO\_1**<<4) (暂时只支持**COMBO**状态下的最多两组不同数据的**COMBO**组合)。既配置**ROLE\_COMBO**|(COMBO\_0<<4)的这组与其他配置**ROLE\_COMBO**|(COMBO\_0<<4)的其他组或者多组需要同时匹配才能trigger, 配置**ROLE\_COMBO**|(COMBO\_1<<4)的这组需要与其他配置**ROLE\_COMBO**|(COMBO\_1<<4)的其他组或者多组需要同时匹配才能trigger。

**gpio\_trigger\_idx**: 代表trigger时拉低或拉高的两组gpio的其中idx, 可同时trigger。如果配置**TG\_IDX\_0**, 则对应初始化参数中的 wakeup\_param->gpio\_num[0], 如果配置**TG\_IDX\_0**|**TG\_IDX\_1**, 则对应初始化参数中的wakeup\_param->gpio\_num[0] 与 wakeup\_param->gpio\_num[1]同时被trigger取反。