AP应用接口文档

前言说明:

1）AP_8288只提供镜像文件

库接口

1. 初始化

- 目的

  wiota系统初始化

- 语法

```
void uc_wiota_init(void);
```

- **描述**

  初始化wiota资源，初始化线程，内存等。

- 返回值

  无

- 参数

  无

2. 启动wiota

- 目的

  启动wiota系统

- 语法

```
void uc_wiota_start(void);
```

- 描述

  启动wiota系统，进入NULL状态。

- 返回值

  无

- 参数

  无

3. 关闭wiota

- 目的

  关闭wiota系统，ap8288也将停止

- 语法

```
void uc_wiota_exit(void);
```

- 描述

  关闭wiota系统，回收所有wiota系统资源。

- 返回值

  无

- 参数

  无

3. 设置动态参数

3.1 设置全部动态参数

- 目的

  设置动态参数

- 语法

```
uc_result_e uc_wiota_set_all_dynamic_parameter(dynamic_para_t *dyan_para);
```

- 描述

  wiota系统开始之前配置参数

- 返回值(下同)

```
typedef enum
{
    UC_SUCCESS = 0,
    UC_TIMEOUT = 1,
    UC_FAILED  = 2,
}uc_result_e;
```

- 参数

```
typedef struct
{
    u32_t system_id;          // 系统id
    u32_t subsystem_id;       // 子系统id
    u8_t id_len;              // id长度，取值0,1,2,3代表2,4,6,8字节
    u8_t pn_num;              // 固定为1，暂时不提供修改
    u8_t symbol_length;       // 帧配置，取值0,1,2,3代表128,256,512,1024
    u8_t dlul_ratio;          // 帧配置，下上行比例，取值0,1代表1:1和1:2
    u8_t bt_value;            // 和调制信号的滤波器带宽对应，BT越大，信号带宽越大，取值0,1代表1.2和0.3，BT=1.2的数据率比BT=0.3的高
    u8_t group_number;        // 帧配置，取值0,1,2,3代表1,2,4,8个上行group数量
    u8_t ap_max_power;        // ap最大功率，默认为21db。范围 0 - 31db.
    u8_t spectrum_idx;        // 频谱，默认值3（470M-510M）
    u8_t na[48];
}dynamic_para_t;
```

频谱idx	低频 MHz	高频 MHz	中心频率 MHz	带宽 MHz	频点step MHz	频点 idx	频点 个数
0（other1）	223	235	229	12	0.2	0~60	61
1（other2）	430	432	431	2	0.2	0~10	11
2（EU433）	433.05	434.79	433.92	1.74	0.2	0~8	9
3（CN470-510）	470	510	490	40	0.2	0~200	201
4（CN779-787）	779	787	783	8	0.2	0~40	41
5（other3）	840	845	842.5	5	0.2	0~25	26
6（EU863-870）	863	870	866.5	7	0.2	0~35	36
7（US902-928）	902	928	915	26	0.2	0~130	131

3.2 设置system id

- 目的

  设置系统id

- 语法

```
uc_result_e uc_wiota_set_system_id(u32_t system_id);
```

- 描述

  wiota系统开始单独配置系统id

- 返回值

```
uc_result_e
```

- 参数

```
system_id                    // 系统id
```

3.3 设置subsystem id

- 目的

  设置子系统id

- 语法

```
uc_result_e uc_wiota_set_subsystem_id(u32_t subsystem_id);
```

- 描述

  wiota系统开始单独配置子系统id

- 返回值

```
uc_result_e
```

- 参数

```
subsystem_id                //子系统id
```

3.4 设置user id length

- 目的

  设置user id length

- 语法

```
uc_result_e uc_wiota_set_user_id_len(u8_t user_id_len);
```

- 描述

  wiota系统开始单独配置user id length

- 返回值

```
uc_result_e
```

- 参数

```
user_id_len                 //user id的长度
```

3.5 设置symbol length

- 目的

  设置symbol length

- 语法

```
uc_result_e uc_wiota_set_symbol_length(symbol_length_e symbol_length);
```

- 描述

  wiota系统开始单独配置symbol length

- 返回值

```
uc_result_e
```

- 参数

```
typedef enum
{
    SYMBOL_LENGTH_128  = 0,      //symbol length is 128
    SYMBOL_LENGTH_256  = 1,      //symbol length is 256
    SYMBOL_LENGTH_512  = 2,      //symbol length is 512
    SYMBOL_LENGTH_1024 = 3,      //symbol length is 1024
    SYMBOL_LENGTH_INVALID,
}symbol_length_e;
```

3.6 设置下行和上行的比例

- 目的

  设置上下行比例

- 语法

```
uc_result_e uc_wiota_set_dlul_ratio(dlul_ratio_e dlul_ratio);
```

- 描述

  wiota系统开始单独配置上下行比例

- 返回值

```
uc_result_e
```

- 参数

```
typedef enum
{
    DL_UL_ONE_TO_ONE  = 0,      //1:1
    DL_UL_ONE_TO_TWO  = 1,      //1:2
    DL_UL_INVALID,
}dlul_ratio_e;
```

3.7设置bt value

- 目的

  设置bt value

- 语法

```
uc_result_e uc_wiota_set_bt_value(bt_value_e bt_value);
```

- 描述

  wiota系统开始单独配置bt value

- 返回值

```
uc_result_e
```

- 参数

```
typedef enum
{
    BT_VALUE_0_POINT_3 = 0,      //0.3:支持所有的symbol length
    BT_VALUE_1_POINT_2 = 1,      //1.2：只支持symbol length 为128和256
    BT_VALUE_INVALID,
}bt_value_e;
```

3.8 设置group number

- 目的

  设置group number

- 语法

```
uc_result_e uc_wiota_set_group_numbr(group_number_e group_number);
```

- 描述

  wiota系统开始单独配置group number

- 返回值

```
uc_result_e
```

- 参数

```
typedef enum
{
    GROUP_NUMBER_1 = 0,      //group number is 1
    GROUP_NUMBER_2 = 1,      //group number is 2
    GROUP_NUMBER_4 = 2,      //group number is 4
    GROUP_NUMBER_8 = 3,      //group number is 8
    GROUP_NUMBER_INVALID,
}group_number_e;
```

3.9 设置dcxo

- 目的

  设置dcxo，有源晶体的ap8088芯片不需要设置dcxo

- 语法

```
uc_result_e uc_wiota_set_dcxo(u32_t dcxo);
```

- 描述

  wiota系统开始单独配置dcxo，必须设置正确，否则可能无法同步

- 返回值

```
uc_result_e
```

- 参数

```
dcxo                    // 校准后的dcxo值
```

3.10设置pn number（暂不支持设置）

- 目的

  设置pn number

- 语法

```
uc_result_e uc_wiota_set_pn_number(u8_t pn_num);
```

- 描述

  wiota系统开始单独配置pn number

- 返回值

```
uc_result_e
```

- 参数

```
pn_num                  // pn number
```

3.11 查询dcxo

- 目的

  查询dcxo

- 语法

```
u32_t uc_wiota_get_dcxo(void);
```

- 描述

  查询设置的dcxo

- 返回值

```
dcxo的值
```

- 参数

```
NULL
```

3.12 获取频点

- 目的

  获取当前设置的频点

- 语法

```
u32_t uc_wiota_get_frequency_point(void);
```

- 描述

  获取设置的频点

- 返回值

```
freq
```

- 参数

```
NULL
```

3.13 设置射频功率

- 目的

  设置ap8288射频功率

- 语法

```
uc_result_e uc_wiota_set_rf_power(s8_t rf_power);
```

- 描述

  设置ap8288射频功率

- 返回值

  uc_result_e

- 参数

  rf_power：功率值，取值范围为：-1~34

3.13 设置active态的终端数量

- 目的

  设置同一个子帧上的最大的active态终端的数量

- 语法

```
uc_result_e uc_wiota_set_max_active_iote_num_in_the_same_subframe(u8_t
max_iote_num);
```

- 描述

  用于设置同一个子帧位置上最大的active态终端数量

- 返回值

  uc_result_e

- 参数

  max_iote_num：（默认为1，最大为3）

4. 扫频和锁频

4.1 扫描频点集合

- 目的

  扫描频点集合（见例子test_handle_scan_freq()）

- 语法

```
uc_result_e uc_wiota_scan_freq(u8_t *freq,
                               u8_t freq_num,
                               s32_t timeout,
                               uc_scan_callback callback,
                               uc_scan_recv_t *scan_result
                               );
```

- 描述

  扫描频点集合，返回各频点的详细结果，包括snr、rssi、is_synced

- 返回值

  uc_result_e

- 结构体

```
typedef struct
{
    u16_t data_len;      //扫频后的结果数据总长度
    u8_t  *data;         //扫频后的结果数据，将该数据强制转化为uc_scan_freq_info_t类型，即可得到各频点的详细信息
    u8_t  result;        //扫频操作执行的结果是否成功
}uc_scan_recv_t;

//频点的信息
typedef struct
{
    u8_t freq_idx;
    n8_t snr;
    s8_t rssi;
    u8_t is_synced;
}uc_scan_freq_info_t;
```

- 参数

```
freq                    //频点集合
freq_num                //频点数量
timeout                 //超时时间
callback                //执行结果回调
scan_result             //扫频结果
注：如果freq==NULL && freq_num == 0 && timeout == -1时，为全扫，全扫大约需要4分钟
```

4.2 设置默认频点

- 目的

  设置默认频点

- 语法

```
uc_result_e uc_wiota_set_frequency_point(u32_t frequency_point);
```

- 描述

  设置默认频点，频点范围470M-510M，每200K一个频点

- 返回值

```
uc_result_e
```

- 参数

```
frequency_poin          //范围0 ~ 200，代表频点（470 + 0.2 * frequency_point）
```

4.3 设置跳频频点

- 目的

  设置跳频频点

- 语法

```
uc_result_e uc_wiota_set_hopping_freq(u8_t hopping_freq);
```

- 描述

  设置跳频频点，频点范围470M-510M，每200K一个频点

- 返回值

```
uc_result_e
```

- 参数

```
hopping_freq            //范围0 ~ 200，代表频点（470 + 0.2 * hopping_freq）
```

4. 设置跳频模式

- 目的

  设置跳频模式

- 语法

```
uc_result_e uc_wiota_set_hopping_mode(u8_t hopping_mode);
```

- 描述

  设置跳频模式，默认为模式0，不跳频

- 返回值

```
uc_result_e
```

- 参数

```
hopping_mode：0代表这一帧不跳，1代表这一帧工作在设置的跳频频点上
typedef enum
{
    HOPPING_MODE_0 = 0,     //default mode, not hopping
    HOPPING_MODE_1 = 1,     //010101...
    HOPPING_MODE_2 = 2,     //00110011...
    HOPPING_MODE_3 = 3,     //000111000111...
    HOPPING_MODE_INVALID,
}hopping_mode_e;
```

5. 黑名单

5.1 获取黑名单

- 目的

  获取已设置的黑名单信息

- 语法

```
blacklist_t *uc_wiota_get_blacklist(u16_t *blacklist_num);
```

- 描述

  获取已设置的黑名单链表头

- 返回值

```
blacklist_t             //黑名单链表头（使用完后需要释放该头结点空间）
```

- 参数

```
blacklist_num           //返回总共的黑名单数量
```

5.2 打印黑名单内容

- 目的

  打印已获取到的黑名单内容

- 语法

```
uc_result_e uc_wiota_print_blacklist(blacklist_t *head_node, u16_t
blacklist_num);
```

- 描述

  根据获取到的黑名单链表头打印所有节点信息

- 返回值

```
uc_result_e
```

- 参数

```
head_node               //获取到的黑名单链表头
blacklist_num           //获取到的黑名单总个数
```

5.3 添加iote到黑名单

- 目的

  添加一个或多个iote到黑名单（可用于删除指定id的iote，将该iote的id添加到黑名单即可）

- 语法

```
uc_result_e uc_wiota_add_iote_to_blacklist(u32_t *user_id, u16_t
user_id_num);
```

- 描述

  根据传入的user id和数量，将该组userid添加到黑名单，黑名单中的userid将不再处理

- 返回值

```
uc_result_e
```

- 参数

```
user_id                 //user id数组首地址
user_id_num             //数组有效id数量
```

5.4 从黑名单中移除iote

- 目的

  将一个或多个iote从黑名单中移除

- 语法

```
uc_result_e uc_wiota_remove_iote_from_blacklist(u32_t *user_id, u16_t
user_id_num);
```

- 描述

  根据传入的user id和数量，将该组userid从黑名单中移除

- 返回值

```
uc_result_e
```

- 参数

```
user_id                 //user id数组首地址
user_id_num             //数组有效id数量
```

6. 查询

6.1 查询当前已连接iote的信息

- 目的

  查询当前连接态的iote信息

- 语法

```
ioteInfo_t *uc_wiota_query_active_iotes(u16_t *iote_num);
```

- 描述

  查询当前已连接iote的信息，返回信息链表头和总个数

- 返回值

```
ioteInfo_t              //信息链表头（使用完后需要释放该头结点空间)
```

- 参数

```
iote_num                //传出当前iote的总个数
```

6.2 打印查询到的iote信息

- 目的

  打印当前已连接iote的信息

- 语法

```
uc_result_e uc_wiota_print_iote_info(IoteInfo_T *head_node, u16_t iote_num);
```

- 描述

  根据查询到的结果，打印所有iote信息

- 返回值

```
uc_result_e
```

- 参数

```
head_node               //获取到的信息链表头
iote_num                //获取到的iote数量
```

7. 回调注册

7.1 iote接入提示

- 目的

  iote接入提示回调注册

- 语法

```
uc_result_e uc_wiota_register_iote_access_callback(uc_iote_access callback);
```

- 描述

  当有iote接入时主动上报哪一个userid的iote接入，可在初始化之后或者wiota start之后注册

- 返回值

```
uc_result_e
```

- 参数

```
typedef void (*uc_iote_access)(u32_t user_id);
callback                  //回调函数函数指针(参数可增加，目前只有user_id)
```

7.2 iote掉线提示

- 目的

  iote掉线提示回调注册

- 语法

```
uc_result_e uc_wiota_register_iote_dropped_callback(uc_iote_drop callback);
```

- 描述

  当有iote掉线时主动上报哪一个userid的iote掉线，可在初始化之后或者wiota start之后注册

- 返回值

```
uc_result_e
```

- 参数

```
typedef void (*uc_iota_drop)(u32_t user_id);
callback                  //回调函数函数指针(参数可增加，目前只有user_id)
```

7.3 接受数据

- 目的

  数据被动上报回调注册

- 语法

```
uc_result_e uc_wiota_register_report_ul_data_callback(uc_report_data
callback);
```

- 描述

  当有数据时上报完成数据，可在初始化之后或者wiota start之后注册

- 返回值

```
uc_result_e
```

- 参数

```
typedef void (*uc_report_data)(u32_t user_id, u8_t *report_data, u32_t
report_data_len);
callback                  //回调函数函数指针
```

8.发送数据

8.1 发送广播数据

- 目的

  发送广播数据给所有iote，现在发送广播（ota或普通广播）时可同时进行上下行业务

- 语法

```
uc_result_e uc_wiota_send_broadcast_data(u8_t *send_data,
                                         u16_t send_data_len,
                                         broadcast_mode_e mode,
                                         s32_t timeout,
                                         uc_send_callback callback
                                         );
```

- 描述

  发送广播数据给所有iote，有两种模式，设置mode的值决定为哪种模式

  如果callback为NULL，为阻塞调用，发送的数据大于1k需要等到函数返回值为UC_SUCCESS才能发送下一个包

  如果callback不NULL，为非阻塞调用，发送的数据大于1k需要等到注册的回调返回UC_SUCCESS才能发送下一个包

  详见：uc_wiota_interface_test.c中test_send_broadcast_data();的例子

- 返回值

```
uc_result_e              //函数执行结果
当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

```
send_data                //要发送的数据
send_data_len            //要发送的数据长度，最大为1024byte
mode:
    typedef enum
    {
        NORMAL_BROADCAST = 0,       //模式0：数据量小，速率相对较低
        OTA_BROADCAST    = 1,       //模式1：数据量大，速率相对较高
        INVALID_BROADCAST,
    }broadcast_mode_e;
timeout                  //超时时间，发送1k数据的时间大约为4s，若要发送大量数据请将数据分段并控制发送频率
callback                 //执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用
```

8.2 设置广播的mcs

- 目的

  设置广播的mcs

- 语法

```
uc_result_e uc_wiota_set_broadcast_mcs(uc_mcs_level_e mcs)
```

- 描述

  设置广播的传输速率，分为7个等级，ota默认等级2，等级越高每个包可携带的数据量越大

- 返回值

```
uc_result_e              //函数执行结果
```

- 参数

```
mcs：mcs等级
    typedef enum
    {
        UC_MCS_LEVEL_0 = 0,
        UC_MCS_LEVEL_1 = 1,
        UC_MCS_LEVEL_2 = 2,
        UC_MCS_LEVEL_3 = 3,
        UC_MCS_LEVEL_4 = 4,
        UC_MCS_LEVEL_5 = 5,
        UC_MCS_LEVEL_6 = 6,
        UC_MCS_LEVEL_7 = 7
    }broadcast_mode_e;
```

8.3 指定iote发送数据

- 目的

  指定iote发送数据，paging功能已支持，只要iote连接过，就可以调用该接口发送数据，不管该iote是不是连接态

- 语法

```
uc_result_e uc_wiota_paging_and_send_normal_data(u8_t *send_data,
                                                 u16_t send_data_len,
                                                 u32_t *user_id,
                                                 u32_t user_id_num,
                                                 s32_t timeout,
                                                 uc_send_callback callback
                                                 );
```

- 描述

  可向一个或多个iote发送数据

  如果回调函数不为NULL，则非阻塞模式，成功发送数据或者超时后会调用callback返回结果

  如果回调函数为NULL，则为阻塞模式，成功发送数据或者超时该函数才会返回结果

  目前只支持单个寻呼不支持组播

- 返回值

```
uc_result_e              //函数执行结果
当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

```
send_data                //要发送的数据
send_data_len            //要发送的数据长度
user_id                  //要发送数据的iote的userid数据首地址
user_id_num              //iote的个数
timeout                  //超时时间
callback                 //执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用
```

9.连接超时时间设置与查询

9.1 查询超时时间

- 目的

  查询连接态的连接态保持时间

- 语法

```
u32_t uc_wiota_get_active_time(void);
```

- 描述

  查询idle态的连接超时时间，单位：秒

- 返回值

```
active_time              //连接态生存时间
```

- 参数

  无

9.2 设置超时时间

- 目的

  设置连接态的连接态保持时间

- 语法

```
uc_result_e uc_wiota_set_active_time(u32_t active_time);
```

- 描述

  设置idle态的连接态保持时间（需要与iote保持一致）

  终端在接入后，即进入连接态，当无数据发送或者接收时，会保持一段时间的连接态状态，在此期间ap和终端双方如果有数据需要发送则不需要再进行接入操 作，一旦传输数据就会重置连接时间，而在时间到期后，终端自动退出连接态，ap同时删除该终端连接态信息。正常流程是终端接入后发完上行数据，ap再开始发送下行数据，显然，这段时间不能太短，否则会底层自动丢掉终端的信息，导致下行无法发送成功。默认连接时间是3秒，也就是说ap侧应用层在收到终端接入后，需要在3秒内下发下行数据。

- 返回值

```
uc_result_e
```

- 参数

```
active_time      //单位：秒，系统默认值为3秒
```

10.读取ap8288芯片实时温度

- 目的

  可实时获取到ap8288芯片的温度

- 语法

```
uc_result_e uc_wiota_read_temperature(uc_temp_callback callback,
uc_temp_recv_t *read_temp, s32_t timeout);
```

- 描述

  调用该接口可读取ap8288芯片的实时温度，读取温度需要两帧左右，需要在没有任务的时候读取，有任务时会直接返回读取失败

  如果回调函数不为NULL，则非阻塞模式，成功执行或者超时后会调用callback返回结果

  如果回调函数为NULL，则为阻塞模式，成功执行或者超时该函数才会返回结果

- 返回值

```
uc_result_e             //函数执行结果
当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

  callback：函数执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用

  read_temp：出参，返回读取的温度和执行结果

  timeout：函数执行超时时间

11.获取当前版本信息和构建时间

- 目的

  获取当前版本信息和构建时间

- 语法

```
void uc_wiota_get_version(u8_t *version, u8_t *time);
```

- 描述

  获取版本信息和构建时间，需自行开辟空间或使用数组接收出参，version为24个字节，time为20个字节

- 返回值

  无

- 参数

  version：当前版本号

  time：当前版本构建时间