XR806 Button 应用

开发指南

版本号：1.0

发布时间：2021-01-14

版本历史

版本	日期	责任人	版本描述
1.0	2021-01-14	AWA 1451	创建文档。

目录

表格目录

1 前言

1.1 文档简介

本文档介绍了 XR806 Button 模块的使用方法。

1.2 目标读者

本文档适合于 XR806 Button 模块开发及维护的人员。

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

2 概述

2.1 背景说明

XR806 Button 模块用于需要用到按键且按键逻辑较复杂的场景中，Button 模块能有效降低不同模块对按键操作逻辑的耦合性，能够独立开发不同模块对按键操作的代码。

Button 模块支持的硬件按键类型有：AD 按键、GPIO 按键；支持的软件按键类型有：短按按键、长按按键、长短按键、重复按键、组合按键。Button 模块最多支持 32 个实体按键，不支持矩阵按键。

2.2 规格特性

XR806 Button 模块规格特性如下表所示。

表 2-1 XR806 Button 模块规格特性表

模块规格	规格描述	备注
硬件按键类型	AD 按键、GPIO 按键	
软件按键类型	短按按键、长按按键、长短按键、重复按键、组合按键	
实体按键	最多支持 32 个实体按键，不支持矩阵按键	

2.3 文件位置

以 SDK 包为根目录，本模块涉及到的主要文件位置如下。

表 2-2 Button 模块的文件位置

模块名	文件分类	文件位置
Button	源码文件	./project/common/apps/buttons/buttons.c ./project/common/apps/buttons/buttons_low_level.c
	头文件	./project/common/apps/buttons/buttons.h ./project/common/apps/buttons/buttons_low_level.h
	示例工程	./project/example/button/

3 技术说明

3.1 硬件按键说明

Button 模块支持的硬件按键类型有：AD 按键、GPIO 按键，两种按键的差异点有：

1. 对芯片 IO 口资源利用效率不同，AD 按键是一个 IO 口支持多按键输入，GPIO 是一个 IO 对应一个按键。

2. 防抖处理不同：虽然 Button 模块对两种按键都做了防抖处理，但实际上 AD 按键是监测模拟电压值对抖动不敏感，GPIO 按键对抖动较敏感。

Button 应用中需要在板级配置文件对选用的硬件按键进行参数配置，配置方法可参考 5.3.1 节。

3.2 软件按键说明

按照按键行为将软件按键进行分类，可分为五类：短按按键、长按按键、长短按键、重复按键、组合按键。

3.2.1 短按按键

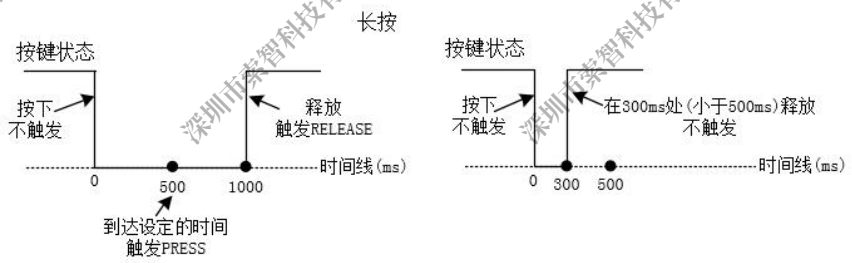
短按按键没有按下的状态，只有释放的状态（RELEASE）。如果一个硬件按键被设置为短按，当该按键被按下时，短按按键对象不会触发按下的状态，只有当该按键释放时，短按按键对象才会触发释放的状态（RELEASE）。该类型按键可用于控制音量加/减、上/下一首、暂停/播放等操作，即只有在按键释放时执行相应的操作。

图 3-1 短按按键

3.2.2 长按按键

长按按键有按下（PRESS）和释放（RELEASE）两种状态。如果一个硬件按键被设置为长按，当该按键被按下并且按下的时间超过设定的时间时（假设 500ms），长按按键对象会触发按下的状态（PRESS），当按键释放时（假设在 1000ms 处释放），长按按键对象会触发释放状态（RELEASE）。如果按键提前释放了（假设在 300ms 处释放了），则按键对象不会触发任何状态。该类型按键几乎可以应用于任何场景，其他类型的按键都可通过该类型按键和定时器来实现。

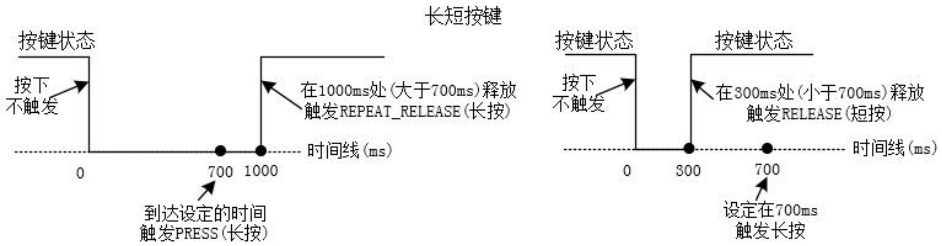
图 3-2 长按按键



3.2.3 长短按键

长短按键有按下（PRESS）、释放（RELEASE）、重复释放（REPEAT_RELEASE）三种状态，该类型按键将长按按键和短按按键相结合，同时具有长按和短按的特性。如果一个硬件按键被设置为长短按键，当该按键被按下并且按下的时间超过设定的时间时(假设 700ms)，长短按键对象会触发按下的状态(PRESS)，即长按的按下特性，当按键释放时（假设在 1000ms 处释放），长短按键对象会触发重复释放的状态（REPEAT_RELEASE），即长按的释放特性。如果按键提前释放了（假设在 300ms 处释放），则按键对象会触发释放状态（RELEASE），即短按的释放特性。该类型按键可用于长按录音场景，长按时间低于 500ms 时，播放“长按一会”提示音，长按时间超过 500ms 时，开始录音。

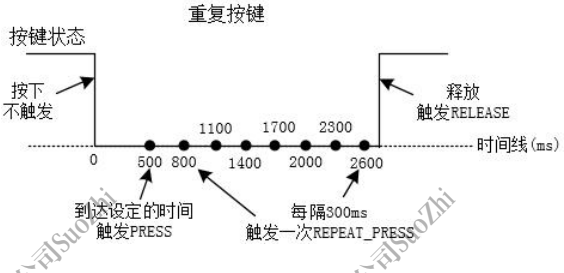
图 3-3 长短按键



3.2.4 重复按键

重复按键有按下（PRESS）、重复按下（REPEAT_PRESS）、释放（RELEASE）三种状态。如果一个硬件按键被设置为重复按键，当该按键被按下并且按下的时间超过设定的时间时（假设 500ms），重复按键对象会触发按下的状态（PRESS），后面每隔一定的时间（假设 300ms），按键对象会触发一次重复按下的状态（REPEAT_PRESS），当按键释放时，按键对象会触发释放状态（RELEASE）。如果按键提前释放了（假设在 300ms 处释放），则按键对象不会触发任何状态。该类型按键可用于，长按时需要重复执行某项操作的场景，比如长按音量键，音量持续增加或减小。

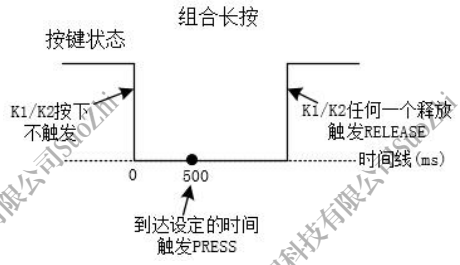
图 3-4 重复按键



3.2.5 组合按键

组合长按是长按按键的一种扩展，支持多个硬件按键，即多个硬件按键对应一个按键对象，其他类型的按键只支持单个硬件按键，即一个硬件按键对应一个按键对象，其有按下（PRESS）和释放（RELEASE）两种状态。如果 K1、K2 被设置为组合长按，当 K1、K2 同时按下并且按下的时间超过设定的时间时（假设 500ms），组合按键对象会触发按下的状态（PRESS），当任何一个按键释放了（假设在 1000ms 处释放），组合按键对象会触发释放状态（RELEASE）。如果按键提前释放了（假设在 300ms 处释放），则按键对象不会触发任何状态。该按键类型一般需要多个按键同时触发的场景，比如同时按上一首/下一首，触发配网按键对象，进入配网场景。

图 3-5 组合按键



4 应用说明

4.1 应用简述

Button 模块已经在 XR806 SDK 中实现了，用户通过 Button 模块提供的接口即可开发出常用的按键功能。

4.2 配置说明

表 4-1 XR806 Button 模块配置列表

配置项	配置说明
Button 模块管脚配置	设置说明： 此项配置用于在 SDK 中配置 AD 按键或 GPIO 按键的管脚。 设置位置： 工程对应的板级配置文件 board_config.c。 设置方式： 在 board_config.c 文件，根据实际硬件需要配置的硬件按键类型分别在 g_ad_buttons/g_gpio_button 添加按键的定义。

4.3 结构体说明

4.3.1 按键对象操作结构体

```
typedef struct button_handle {
    void (*start)(struct button_handle *handle);
    void (*stop)(struct button_handle *handle);
    int (*destroy)(struct button_handle *handle);
    int (*get_state)(struct button_handle *handle);
    void (*cb)(BUTTON_STATE sta, void *arg);
    void *arg;
} button_handle;
```

其结构体成员说明如下表所示。

表 4-2 button_handle 结构体成员说明

成员	说明
start	使能按键对象
stop	失能按键对象

成员	说明
destroy	销毁按键对象
get_state	获取按键对象状态
cb	按键对象被触发时的回调函数
arg	回调函数传入的参数

当一个按键对象被创建后，即可调用按键对象的接口来操作按键。如：short_button->start(short_button)、short_button->stop(short_button)、short_button->destroy(short_button)。

4.3.2 按键状态枚举

```
typedef enum {
    PRESS, //用于长按按键、长短按键、组合按键、重复按键
    RELEASE, //用于短按按键、长按按键、长短按键、组合按键、重复按键
    REPEAT_PRESS, //用于重复按键
    REPEAT_RELEASE, //用于长短按键
    INVALID_STA, //无效按键状态
} BUTTON_STATE;
```

4.3.3 按键类型枚举

```
typedef enum {
    SHORT_BUTTON, //短按按键
    LONG_BUTTON, //长按按键
    SHORT_LONG_BUTTON, //长短按键
    COMBINED_LONG_BUTTON, //组合按键
    REPEAT_LONG_BUTTON, //重复按键
} BUTTON_TYPE;
```

4.3.4 按键掩码

宏 KEY0~宏 KEY31 每个硬件按键对应一个掩码（对应 1bit），一一对应，不能重复。

4.3.5 底层按键注册接口

```
typedef struct {
    int (*low_level_init)(void);
    int (*low_level_get_state)(void);
    int (*low_level_wait_semaphore)(uint32_t waitms);
    int (*low_level_release_semaphore)(void);
    int (*low_level_deinit)(void);
```

```
} button_impl_t;
```

其结构体成员说明如下表所示。

表 4-3 button_impl_t 结构体成员说明

成员	说明
low_level_init	底层按键初始化
low_level_get_state	底层按键获取所有按键状态
low_level_wait_semaphore	底层按键等待一个信号量，即等待硬件按键的触发信号
low_level_release_semaphore	底层释放一个信号量，即释放硬件按键的触发信号
low_level_deinit	底层按键反初始化

4.4 接口说明

表 4-4 Button 模板接口简介

接口名	简要介绍
模块初始化接口	本接口用于按键模块的初始化，属于外部接口。 本接口在 buttons.c 文件中实现，并在 buttons.h 文件中声明。
模块反初始化接口	本接口用于按键模块的反初始化，属于外部接口。 本接口在 buttons.c 文件中实现，并在 buttons.h 文件中声明。
模块栈大小配置接口	本接口用于设置按键模块内部线程栈大小，属于外部接口。 本接口在 buttons.c 文件中实现，并在 buttons.h 文件中声明。
模块线程优先级配置接口	本接口用于设置按键模块内部线程优先级，属于外部接口。 本接口在 buttons.c 文件中实现，并在 buttons.h 文件中声明。
模块按键创建接口	本接口用于模块创建按键，如长按、短按，属于外部接口。 本接口在 buttons.c 文件中实现，并在 buttons.h 文件中声明。

4.4.1 模块初始化接口

4.4.1.1 buttons_init

表 4-5 buttons_init 接口函数说明

信息项	说明
原型	int buttons_init(button_impl_t* impl);
功能	注册底层按键接口，初始化按键模块。
参数	button_impl_t* impl 含义解释：底层按键注册接口

信息项	说明
	使用说明：N/A
返回值	0：成功 -1：失败

4.4.2 模块反初始化接口

4.4.2.1 buttons_deinit

表 4-6 buttons_deinit 接口函数说明

信息项	说明
原型	int buttons_deinit(void);
功能	反初始化按键模块。
参数	无
返回值	0：成功 -1：失败

4.4.3 模块栈大小配置接口

4.4.3.1 buttons_set_thread_stack_size

表 4-7 buttons_set_thread_stack_size 接口函数说明

信息项	说明
原型	void buttons_set_thread_stack_size(uint32_t size);
功能	设置按键模块内部线程栈大小，由于按键对象的回调函数都是在按键模块内部线程中被调用的，故线程栈大小需要根据应用代码来设置，默认栈大小为 512 字节。需在按键模块初始化函数前调用。
参数	uint32_t size 含义解释：要设置的栈大小，单位是字节 使用说明：N/A
返回值	无

4.4.4 模块线程优先级配置接口

4.4.4.1 buttons_set_thread_priority

表 4-8 buttons_set_thread_priority 接口函数说明

信息项	说明
原型	void buttons_set_thread_priority(uint32_t priority);
功能	设置按键模块内部线程优先级，默认优先级为 3。

信息项	说明
参数	uint32_t priority 含义解释：要配置的优先级 使用说明：取值只能是 0~6
返回值	无

4.4.5 模块按键创建接口

4.4.5.1 create_short_button

表 4-9 create_short_button 接口函数说明

信息项	说明
原型	button_handle* create_short_button(uint32_t id_mask);
功能	创建短按按键对象
参数	uint32_t id_mask 含义解释：按键掩码 KEY0~KEY31 使用说明：N/A
返回值	按键对象的操作接口：创建成功 NULL：创建失败

使用事例：

```
button_handle *short_button;
short_button = create_short_button(KEY0);
```

4.4.5.2 create_long_button

表 4-10 create_long_button 接口函数说明

信息项	说明
原型	button_handle* create_long_button(uint32_t id_mask, uint32_t timeout_ms);
功能	创建长按按键对象
参数	uint32_t id_mask 含义解释：按键掩码 KEY0~KEY31 使用说明：N/A uint32_t timeout_ms 含义解释：触发长按按键的时间，单位 ms。 使用说明：按键在小于 timeout_ms 时释放，操作无效；在 timeout_ms 时，按键对象的回调函数会被调用，并传递 PRESS 状态；在大于 timeout_ms 时释放，按键对象回调函数会被调用，并传递 RELEASE 状态。
返回值	按键对象的操作接口：创建成功

信息项	说明
	NULL：创建失败

使用事例：

```
button_handle *long_button;
long_button = create_long_button(KEY1, 50); //KEY1 按下时间超过 50ms，才算有效操作
```

4.4.5.3 create_short_long_button

表 4-11 create_short_long_button 接口函数说明

信息项	说明
原型	button_handle* create_short_long_button(uint32_t id_mask, uint32_t timeout_ms);
功能	创建长短按键对象
参数	uint32_t id_mask 含义解释：按键掩码 KEY0~KEY31 使用说明：N/A uint32_t timeout_ms 含义解释：长按和短按的时间分界点，单位 ms。 使用说明：按键在小于 timeout_ms 时释放，触发短按特性，按键对象回调函数会被调用，并传递 RELEASE 状态；在 timeout_ms 时，触发长按特性，按键对象回调函数会被调用，并传递 PRESS 状态；在大于 timeout_ms 时释放，按键回调函数会被调用，并传递 REPEAT_RELEASE 状态。
返回值	按键对象的操作接口：创建成功 NULL：创建失败

使用事例：

```
button_handle *short_long_button;
short_long_button = create_short_long_button(KEY2, 500); //500ms 为长按和短按的分界点
```

4.4.5.4 create_combined_long_button

表 4-12 create_combined_long_button 接口函数说明

信息项	说明
原型	button_handle* create_combined_long_button(uint32_t id_mask, uint32_t timeout_ms);
功能	创建重复按键对象
参数	uint32_t id_mask 含义解释：按键掩码 KEY0~KEY31 使用说明：N/A

信息项	说明
	uint32_t timeout_ms 含义解释：触发组合按键的时间，单位 ms。 使用说明：组合按键在小于 timeout_ms 时释放，操作无效；组合按键同时按下的时间达到 timeout_ms 时，按键对象回调函数会被调用，并传递 PRESS 状态；在大于 timeout_ms 时，任何一个按键释放了导致按键组合被打破，按键回调函数会被调用，并传递 RELEASE 状态。
返回值	按键对象的操作接口：创建成功 NULL：创建失败

组合按键的种类有同通道的 AD+AD、不同通道的 AD+AD、AD+GPIO、GPIO+GPIO 四种方式，只有同通道的 AD+AD 组合方式需要事先测量组合下的 AD 值并配置到 board_config.c 中（详见”示例说明“章节），其他三种方式都可以在创建对象时随意组合。

使用事例：

```
button_handle *combined_button;
combined_button = create_combined_long_button(KEY1 | KEY2, 50);
```

4.4.5.5 create_repeat_long_button

表 4-13 create_repeat_long_button 接口函数说明

信息项	说明
原型	button_handle* create_repeat_long_button(uint32_t id_mask, uint32_t timeout_ms, uint32_t repeat_timeout_ms);
功能	创建重复按键对象
参数	uint32_t id_mask 含义解释：按键掩码 KEY0~KEY31 使用说明：N/A uint32_t timeout_ms 含义解释：触发重复按键的时间，单位 ms。 使用说明：按键在小于 timeout_ms 时释放，操作无效；在 timeout_ms 时，按键对象回调函数会被调用，并传递 PRESS 状态；在大于 timeout_ms 时释放，按键回调函数会被调用，并传递 RELEASE 状态。 uint32_t repeat_timeout_ms 含义解释：重复触发的时间间隔，单位 ms。 使用说明：当按键按下的时间超过 timeout_ms 且一直按着时，每隔 repeat_timeout_ms 时间，按键对象会被调用，并触发 REPEAT_PRESS 状态。
返回值	按键对象的操作接口：创建成功 NULL：创建失败

使用事例：

```
button_handle *repeat_button;
repeat_button = create_repeat_long_button(KEY3, 500, 300); //在 500ms 时触发按键，之后每隔 300ms 触发一次
```

5 示例说明

Button 用于按键场景的功能开发，下面以 AD 按键为例，简要说明 Button 接口的使用。

5.1 示例简介

按键示例的演示的目的是简要介绍 Button 模块相关接口的基本使用方法，演示的功能为通过按不同按键触发不同的按键事件。

5.1.1 获取方式

按键示例有示例工程代码，位于 XR806 SDK 的/project/example/button 目录。

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

5.1.2 准备工作

按键示例工程的硬件准备有如下：

1. 评估板：运行示例工程代码。
2. 串口线：连接评估板的 Uart0 插针，用于 console 控制台的输入输出。
3. PC 机：用于镜像烧录和 console 控制的输入输出。
4. AD 按键模块：作为评估板外接按键硬件电路

烧写工具、代码编译和烧写操作，请参见《XR806_SDK_快速入门指南》。

5.1.3 操作步骤

1. 将 AD 按键模块连接到评估板的 ADC GPIO 管脚，注意 AD 按键模块的供电及与评估板的共地。
2. 编译 button 示例工程，评估板完成烧写，复位后，示例代码自动运行。

5.2 效果演示

在评估板中运行按键示例程序后，会控制台中会打印出以下的日志。

```
button demo started.
```

AD 按键模块按下不同按键会有不同的按键响应。

按下“PLAY（KEY1）”会触发短按按键响应：

```
short button for PLAY/PAUSE has release
```

如按下“PREV（KEY2）”会触发长按按键响应：

```
long button for PRE/VOL+ has press
long button for PRE/VOL+ has release
```

按下“NEXT（KEY3）”大于 500ms 会触发长按按键响应：

```
long button for NEXT/VOL- has press
long button for NEXT/VOL- has release
```

按下“MODE（KEY4）”会触发长按按键事件响应：

```
long button for MODE has press
long button for MODE has release
```

按下“AI（KEY5）”持续一会然后释放会触发重复按键响应：

```
repeat long button for AI has press
repeat long button for AI has repeat release
...
repeat long button for AI has repeat release
repeat long button for AI has release
```

同时按下“PREV（KEY2）”和“NEXT（KEY3）”会触发组合按键响应：

```
combined button for NEXT+PREV has press
combined button for NEXT+PREV has release
```

5.3 关键实现

5.3.1 模块使用流程

具体代码可参考 sdk/project/example/button/main.c。

5.3.1.1 添加按键的配置信息

```
#include "common/apps/buttons/buttons.h"
#include "common/apps/buttons/buttons_low_level.h"

/* do not set const */
static ad_button g_ad_buttons[] = {
    {.name = "play+pause",      .mask = KEY1, .channel = ADC_CHANNEL_6, .value =
704,   .debounce_time = 50},
    {.name = "pre+vol_up",      .mask = KEY2, .channel = ADC_CHANNEL_6, .value =
```

```
1343, .debounce_time = 50},
    {.name = "next+vol_down", .mask = KEY3, .channel = ADC_CHANNEL_6, .value =
1933, .debounce_time = 50},
    {.name = "mode",          .mask = KEY4, .channel = ADC_CHANNEL_6, .value =
2572, .debounce_time = 50},
    {.name = "ai",            .mask = KEY5, .channel = ADC_CHANNEL_6, .value =
3244, .debounce_time = 50},
    {.name = "pre+next",      .mask = KEY2 | KEY3, .channel = ADC_CHANNEL_6, .value =
940,   .debounce_time = 50},
};

__xip_rodata
static const gpio_button g_gpio_buttons[] = {
    {.name = "play", .mask = KEY5, .active_low = 1, { GPIO_PORT_A, GPIO_PIN_7,
{ GPIOA_P7_F6_EINTA7, GPIO_DRIVING_LEVEL_1, GPIO_PULL_UP } }, .debounce_time = 50},
};

static HAL_Status board_get_cfg(uint32_t major, uint32_t minor, uint32_t param)
{
    HAL_Status ret = HAL_OK;
    switch (major) {
    … //其他代码
    case HAL_DEV_MAJOR_AD_BUTTON:
        ((ad_button_info *)param)->ad_buttons_p = g_ad_buttons;
        ((ad_button_info *)param)->count = sizeof(g_ad_buttons) / sizeof(g_ad_buttons[0]);
        break;
    case HAL_DEV_MAJOR_GPIO_BUTTON:
        ((gpio_button_info *)param)->gpio_buttons_p = g_gpio_buttons;
        ((gpio_button_info *)param)->count = sizeof(g_gpio_buttons) / sizeof(g_gpio_buttons[0]);
        break;
    … //其他代码
    }
    return ret;
}
```

g_ad_buttons 为 AD 按键的配置信息。name 为按键的名称；mask 为按键的掩码，每个硬件按键对应一个掩码；channel 为 AD 按键的通道号，不同通道号的 AD 按键也可放于此数组；value 为 AD 按键按下后的 AD 值，需要通过实际测量并取平均；debounce_time 为按键消抖时间。

如果同一个通道下的 AD 按键需要组合按键的话，需要将组合按键的 AD 值添加到 g_ad_buttons 数组中，如组合按键 prev+next，由 KEY2 和 KEY3 组合，都在通道 6，故需要采集 KEY2 和 KEY3 同时按下时的 AD 值。不同 AD 通道的组合按键、GPIO 与 AD 的组合按键或者 GPIO 与 GPIO 的组合按键不需要添加配置信息，可在创建按键对象时直接组合。

g_gpio_buttons 为 GPIO 按键的配置信息。name 为按键名称；mask 为按键掩码；active_low 为低电平有效，如果为 1，则表示按键按下后，状态为低电平；{ GPIO_PORT_A, GPIO_PIN_7, { GPIOA_P7_F6_EINTA7, GPIO_DRIVING_LEVEL_1, GPIO_PULL_UP } }为 gpio 引脚参数；debounce_time 为消抖时间。

board_get_cfg 函数是外部模块获取 g_ad_buttons 和 g_gpio_buttons 的接口，需要添加 g_ad_buttons 和 g_gpio_buttons。

5.3.1.2 模块初始化

注册底层按键接口，设置优先级，栈大小，初始化按键模块。

```
#include "common/apps/buttons/buttons.h"
#include "common/apps/buttons/buttons_low_level.h"

int example_buttons_init(void)
{
    int ret;
    /* register the low level button interface */
    button_impl_t impl = {
        buttons_low_level_init,
        buttons_low_level_get_state,
        buttons_low_level_wait_semaphore,
        buttons_low_level_release_semaphore,
        buttons_low_level_deinit
    };
    /* set buttons thread priority and stack size */
    buttons_set_thread_priority(4);
    buttons_set_thread_stack_size(1024);
    /* init buttons, will init the low level buttons, ad buttons and gpio buttons */
    ret = buttons_init(&impl);
    return ret;
}
```

5.3.1.3 创建按键对象

```
#define MODE     KEY0
#define WECHAT KEY1
#define VOICE    KEY2
#define NEXT     KEY3
#define PREV     KEY4
//全局变量，其他场景需要用到按键
button_handle *short_button;
button_handle *long_button0;
button_handle *long_button1;
button_handle *short_long_button;
button_handle *combined_long_button0;
button_handle *repeat_long_button;

short_button = create_short_button(MODE);
```

```
long_button0 = create_long_button(WECHAT, 50);
long_button1 = create_long_button(VOICE, 500);
short_long_button = create_short_long_button(NEXT, 500);
combined_long_button0 = create_combined_long_button(NEXT | PREV, 50);
repeat_long_button = create_repeat_long_button(PREV, 500, 300);
```

5.3.1.4 设置按键回调函数

```
short_button->cb = short_button_cb;
long_button0->cb = long_button0_cb;
long_button1->cb = long_button1_cb;
short_long_button->cb = short_long_button_cb;
combined_long_button0->cb = combined_long_button0_cb;
repeat_long_button->cb = repeat_long_button_cb;
```

按键的回调函数不是固定的，在不同的场景下，可以设置不同的回调函数，回调函数的处理逻辑或代码在不同的场景中实现，从而实现不同场景下按键操作的解耦。当按键对象被触发后，会调用回调函数并传递相应的按键状态。

5.3.1.5 启动按键

只有按键对象被启动后，才会相应按键的操作。

```
short_button->start(short_button);
long_button0->start(long_button0);
long_button1->start(long_button1);
short_long_button->start(short_long_button);
combined_long_button0->start(combined_long_button0);
repeat_long_button->start(repeat_long_button);
```

5.3.1.6 停止按键

如果在某些场景下需要停止某些按键的操作，可停止按键，比如在 ota 升级场景下，要停止所有按键的操作。

```
short_button->stop(short_button);
long_button0->stop(long_button0);
long_button1->stop(long_button1);
short_long_button->stop(short_long_button);
combined_long_button0->stop(combined_long_button0);
repeat_long_button->stop(repeat_long_button);
```

5.3.1.7 销毁按键

在某些场景下，可能需要临时创建并销毁按键，比如在开机场景下，可临时创建一个厂测按键，多个特定的按键同时按下即进入自动测试场景，自动测试完成或者退出开机场景时，可销毁临时创建的按键。

```
short_button->destroy(short_button);
long_button0->destroy(long_button0);
long_button1->destroy(long_button1);
```

```
short_long_button->destroy(short_long_button);
combined_long_button0->destroy(combined_long_button0);
repeat_long_button->destroy(repeat_long_button);
```

6 常见问题

6.1 在按键初始化后，按键无响应

可能原因：

1. 硬件问题
2. 按键配置信息未配置对
3. 其他代码占用了或配置了按键的引脚

解决方法：

1. 测量 AD 按键按下后，AD 引脚的电压变化；测量 GPIO 按键按下后，GPIO 引脚的电平变化。可排除硬件问题。
2. 检查按键配置信息
3. 检查其他代码或其他模块是否占用了或重新配置了按键的引脚。

6.2 AD 按键个别失效

同一通道的 AD 按键有些能用，有些不能用；或者有时能用，有时不能用；或者会出现错误检测按键的现象。

可能原因：

1. 硬件中，AD 按键分压电阻精确度不高，或温飘太大。
2. 硬件中，各 AD 按键的 AD 值比价接近，导致容易出现按键误判现象。

解决方法：

1. 不同环境下，多次测量 AD 按键按下后的 AD 值，计算平均值。
2. 调整 project\common\apps\buttons\buttons_low_level.h 中的宏 AD_FLUCTUATION 到合适的值，该宏为按键 AD 值的波动区间，若某按键 AD 值的平均值为 1000，则该按键 AD 值的波动区为 1000±AD_FLUCTUATION。当出现有时能用，有时不能用时，说明区间太小，可增加该值；当出现错误检测按键现象时，说明区间太大，导致不同按键区间有重合，可减小该值。

著作权声明

商标声明

免责声明