珠海市杰理科技股份有限公司
ZhuHai JieLi Technology Co.,Ltd

AC630N_bt_data_transfer_sdk_Introduction

AC630N_bt_data_transfer_sdk_Introduction User Manual

Rev **2.1.0** - **March** 14 , **2022**

Table of contents

change log

Version	date	describe
0.1.0	2019 / 12 / 03 User Manual	
renew:	ÿ Build the initial version	
	ÿ Define the document format	
	ÿ Describe SDK functions	
0.2.0	2020 / 06 / 02	
	ÿ Integrate the SDK project of HID and SPP_AND_BLE	
	ÿ Added description of AT+data transmission (SPP+BLE)	
	ÿ Added APP upgrade (BLE) description	
	ÿ Added BLE host client function description	
0.3.0	2020 / 07 / 09	
	ÿ Modified and updated some documentation descriptions	
0.4.0	2020 / 09 / 11	
	ÿ Added MOUSE and DONGLE Bluetooth and 2.4G mode CASE support for AC636N series	
	ÿ Add AT command to control BLE host	
	ÿ HID adds Selfie, page turner and standard keyboard CASE	
	ÿ Added support chip AC635N series	
0.5.0	2020 / 11 / 05	
	ÿ Added support for AC637N series chips	
	ÿ Add pairing and identify the other system interface	
0.6.0	2020 / 12 / 15	
	ÿ Add audio part.	
0.7.0	2021 / 1 / 8	
	ÿ Added audio instructions.	
	ÿ Keypage failed to add mobile phone system, switch descriptor	
0.8.0	2021 / 3 / 1	
	ÿ Added support for AC632N series chips	
	ÿ Add GameBox example	
0.9.0	2021 / 4 / 2	
	ÿ Added Bluetooth multi-connection example	

	ÿ Add the IBEACON broadcast example of BLE ÿ Add AT COM string mode control
0.9.1	2021 / 4 / 23
0.9.2	2021 / 5 / 14
1.0.0	ÿ Example of non-connected 2.4G data transmission and reception ÿ Tencent connection application description ÿ Added AUDIO MIDI instructions
2.0.0	2021 / 9 / 13 ÿ Adjust the code structure and fix the description of apps ÿ Add Tuya protocol support ÿ Add the example of voice remote control hid support encoding ÿ INI configuration file generation ÿ Added the description of GATT common module
2.1.0	2022 / 3 / 10 ÿ Added lighting handshake charging function ÿ Added online serial port upgrade function ÿ OTA adds an upgrade method that can jump to MASKROM ÿ MESH project supports AUDIO function

Chapter 1 SDK Quick Instructions

1.1 Purpose of writing

This document mainly describes the use method of the AC630N_SDK development kit and some problems that should be paid attention to in the development. provided references, including:

Installation toolchain, installation package management tool, configuration tool usage, INI configuration file generation, etc.

1.2 Install the toolchain

1.2.1 Installation package

For SDK development, the following two tools need to be installed:

1. Integrated Development Environment Codeblocks

(1) Download address: http://jl-update.oss-cn-shenzhen.aliyuncs.com/codeblocks-latest.exe

2. Compiler

(2) Download address: http://jl-update.oss-cn-shenzhen.aliyuncs.com/jieli-windows-toolchains-latest.exe

Note: You need to install Codeblocks first, and then install the compiler; follow the prompts to install it.

1.3 Install package management tools

ÿ Package management tool installation

The package management tool is used to download the tools that Jerry SDK needs to use, and it is also responsible for opening the files with the suffix of jlproxj. which is,

The package management tool is responsible for downloading the latest tools required by the SDK to the current computer from the Jerry server.

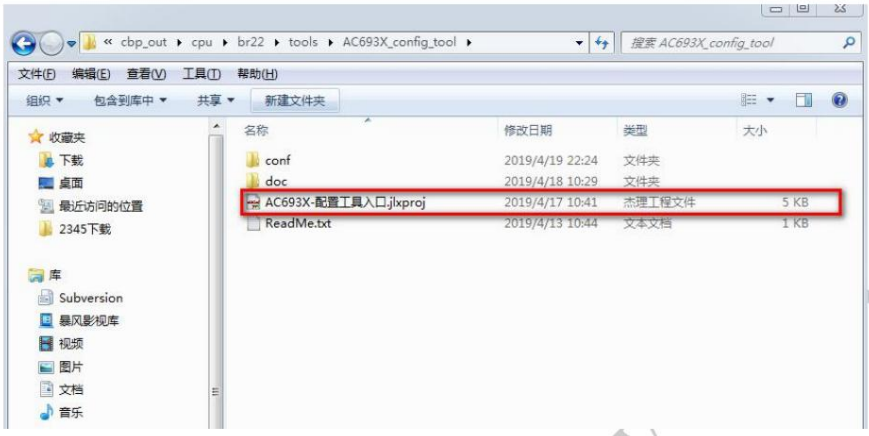
(Package management tool download path: http://jl-update.oss-cn-shenzhen.aliyuncs.com/jieli-pkgman-setup-latest.exe)

Double-click *.exe to install, follow the prompts to complete the installation. After the installation is complete, find the file directory to open, double-click to open the suffix directly

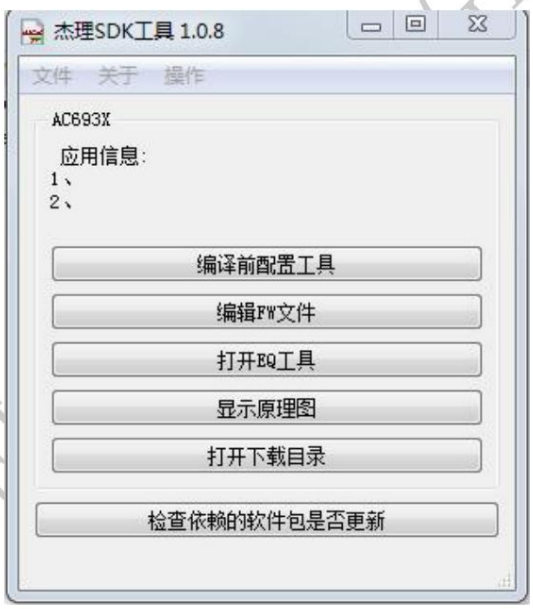
file for .jlxproj. As shown in the figure below, the configuration path: sdk\cpu\bd19\tools\AC632N_config_tool

1.4 Use of configuration tools

1. Open the sdk project directory and enter the cbp_out\cpu\BD29\tools\AC630N_config_tool directory;



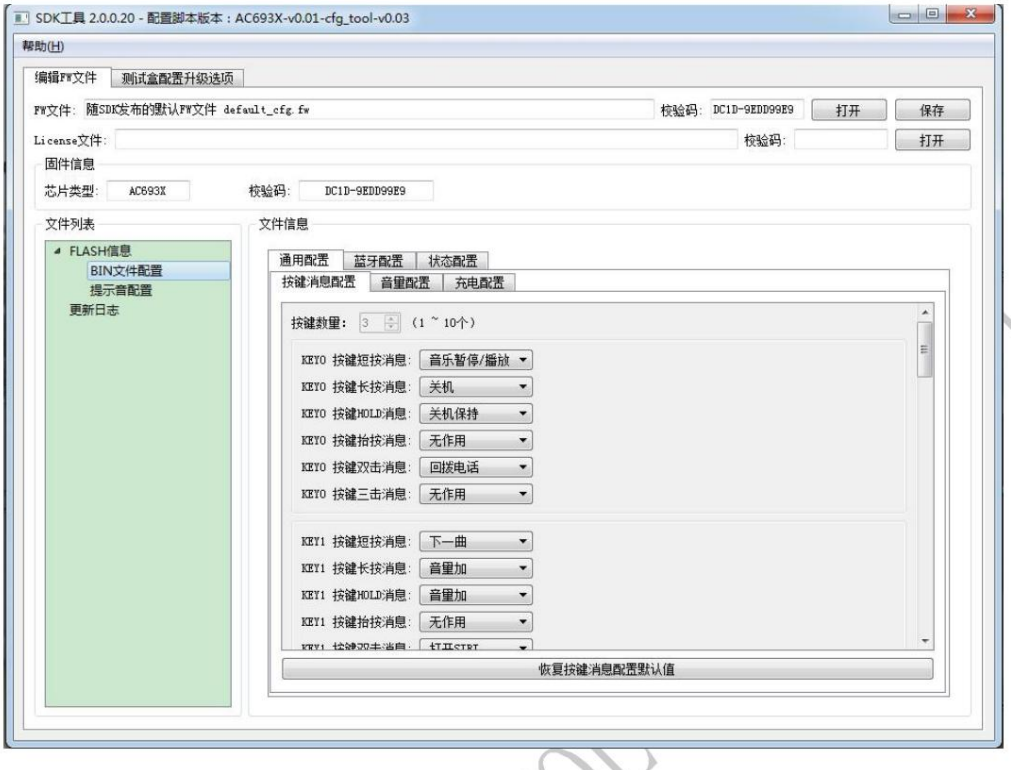
2. Double-click [AC630N-Configuration Tool Entry.jlxproj] to open the [Jerry SDK Tool]



ÿJerry SDK configuration toolÿInterface description:

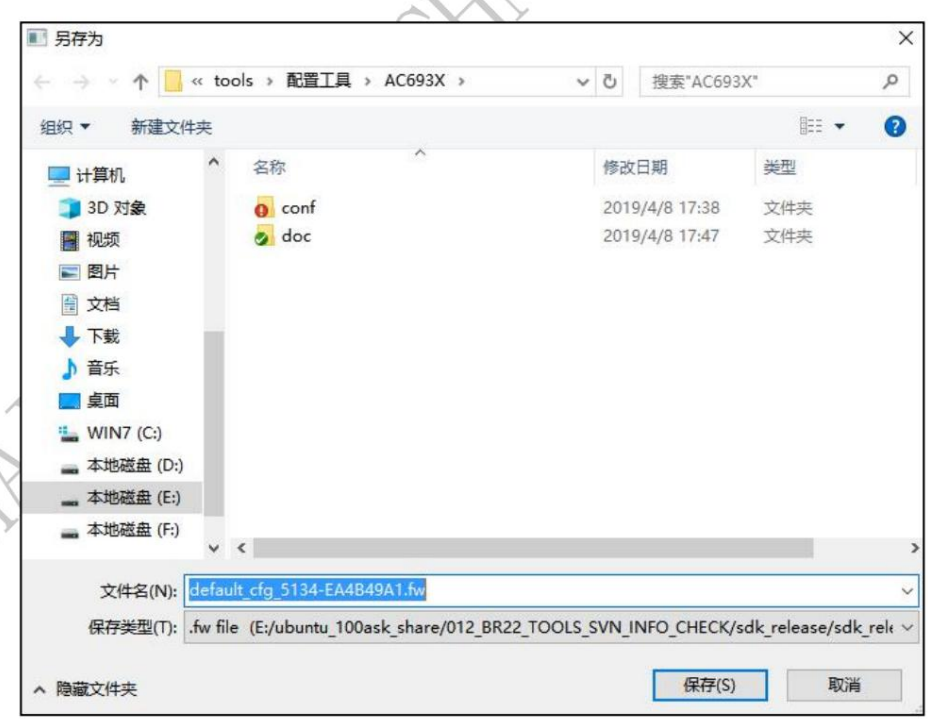
1) Edit FW file: Click [Edit FW file] to edit a FW file and modify the board level configuration of the FW file

Settings, Bluetooth configuration, status configuration and prompt tone configuration, the interface is as follows:



Click [Restore Defaults] to restore the original configuration when the sdk was released. After the configuration is complete, click [Save] to select the save path.

Save the modified fw and ufw files in the path;



There are a few things to note when editing fw files:

(1) The optional options of the board-level configuration depend on the cfg_tool.bin file, that is to say, what is selected in the configuration before compilation, editing the fw file will have

What configuration, for example, the configuration before compilation is to select 3 keys, then there are only 3 key options when editing the fw file.

(2) fw version number control, the corresponding fw file can be edited only if the version number of the tool and fw match, and the version number is in

Modify the user_cfg.lua file in the AC630N_config_tool\conf\entry directory, as shown below:

```
1   设置fw版本信息
2   cfg:addKeyInfo("script_version", "AC693X-v0.0.3");
3
4   -------------------- 设置应用名称 --------------------
5   product_name = "AC693X"; --此名称将显示于配置工具入口界面
6
7   -------------------- 设置配置工具开发状态 --------------
8       -- develop: 开发状态，用于开发SDK使用
9       -- release: 发布状态，用于发布上传使用
```

If the version number prompt is incorrect, please confirm the version number.

(3) fw file production: when there is no problem in the fw file test, when you want to publish an editable fw file, put the generated fw file

Just replace the default_cfg.fw file in the AC630N_config_tool\conf\output\default directory

(4) Production of default values: After configuring with the pre-compile options, click Save and it will be generated in AC630N_config_tool\conf\output

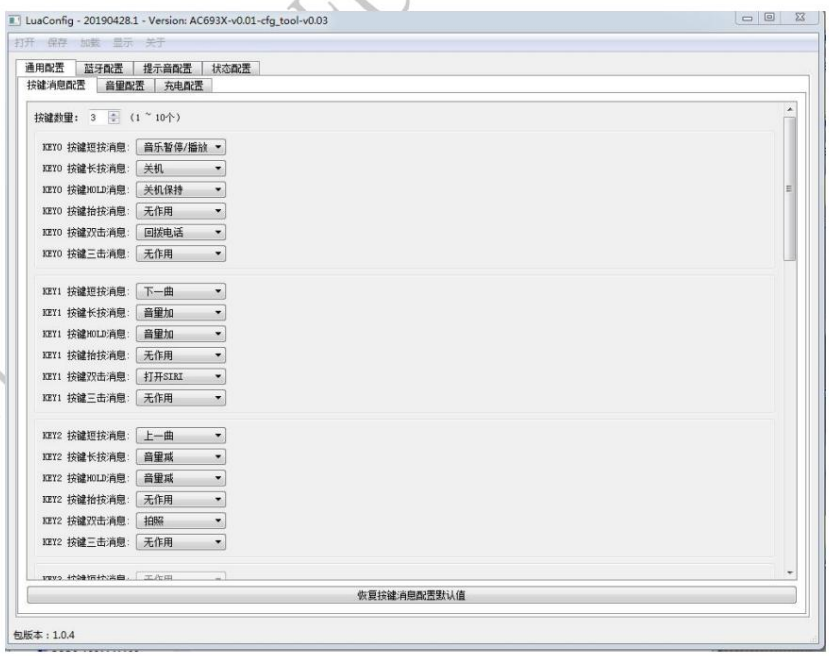
A default_cfg.lua file, overwrite the file in the AC630N_config_tool\conf\output\default directory

default_cfg.lua file, click restore default settings when editing the fw file, it will restore the previously set value.

2) Show schematic diagram: Click [Show schematic diagram], you can open a doc folder, and you can store the schematic diagram in this folder

and other related documents;

3) Configuration before compilation: Click [Configure before compilation], you can configure related configuration items before sdk code compilation, and open the interface as follows:



You can perform general configuration, Bluetooth configuration, prompt tone configuration and status configuration in the pre-compilation configuration tool, click [Restore Defaults]

You can restore the original configuration when the sdk was released. After the configuration is complete, click [Save] to save the configuration, and cfg_tools.bin will be output to the following

In the download directory, recompile the sdk code to apply the latest configuration;

4) Open the download directory: Click [Open Download Directory], the tools download directory will be opened, and cfg_tools.bin will be output to this directory.

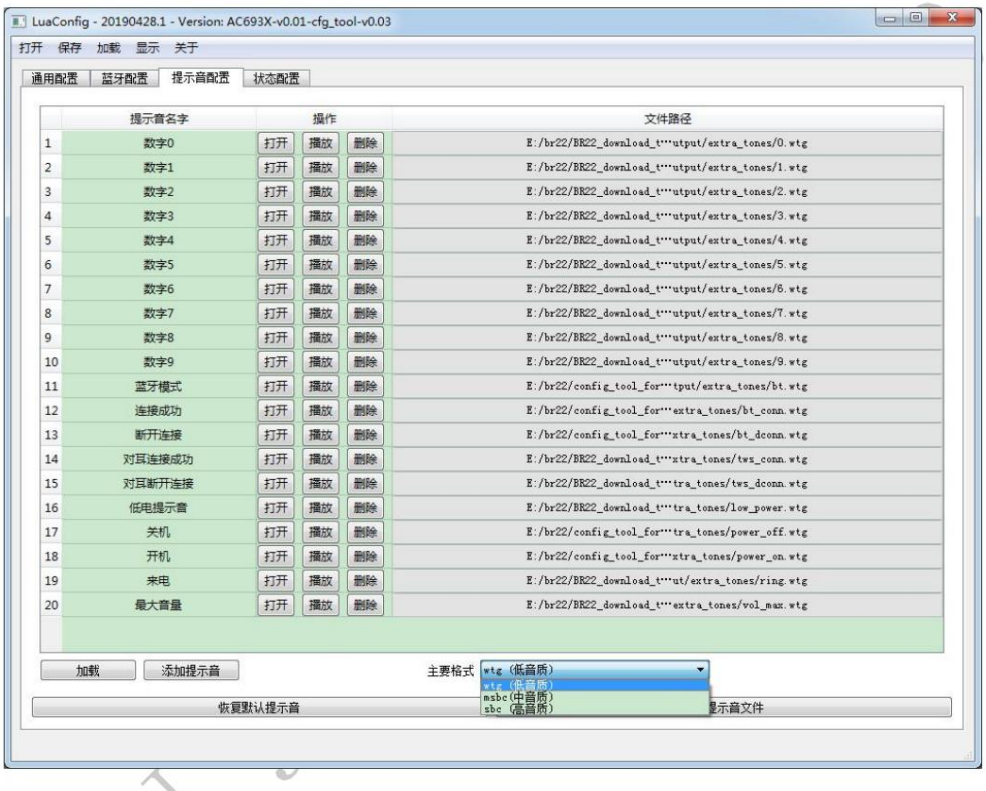
Mark down;

5) Check whether the dependent software packages are updated: Click [Check whether the dependent software packages are updated], and the related configuration tools will be checked.

The version update situation, if there is an update, just download the corresponding update;

3. Prompt sound configuration

The prompt tone is set in the "Prompt Tone Configuration" option of the tool, click it as shown below:



The default is to load the tone under conf\output\extra_tones. If there is a new tone.cfg file, click the load button to

Load the sound file into it. If you want to change the prompt tone, you can directly pull the corresponding prompt tone file to the corresponding option. hint

The audio tool provides 3 kinds of sound quality prompt tone formats, each prompt tone can be individually specified in different formats, mainly depending on the flash space.

For example, if the sound quality of the "Bluetooth mode" sound is better, you can select sbc first, and then pull the corresponding sound file. Other formats are also available.

It is possible to operate in this way. The format of each prompt tone depends on which format was selected when the prompt tone file was pulled in the past.

Note: The digital prompt tone format of the number report must be selected in the same format. The original audio data should be converted with a sampling rate of 32k for sbc and 16k for msbc.

The sample rate conversion can guarantee the sound quality.

1.5 Download directory INI configuration file generation

Mainly describe the configuration and generation of the isd_config.ini configuration file in the tools directory of the new SDK (v2.0.0 and later).

The SDK configuration file isd_config.ini is configured by isd_config_rule.c according to the corresponding board level board_xxx_global_build_cfg.h generate. It should be noted that if you want to modify isd_config.ini, you must modify board_xxx_global_build_cfg.h or isd_config_rule.c, if you modify isd_config.ini directly, the modification will be overwritten every time you compile.

1.4.1 Reset source

in board_xxx_global_build_cfg.h

```
1. //config long-press reset io pin,time,trigger level
2. #if CONFIG_LP_TOUCH_KEY_EN
3. #define CONFIG_RESET_PIN LDO // i pin
4. #define CONFIG_RESET_TIME 04 //unit:second
5. #define CONFIG_RESET_LEVEL 1 //tigger level(0/1)
6. #else
7. #define CONFIG_RESET_PIN PB01 //io pin
8. #define CONFIG_RESET_TIME 08 //unit:second
9. #define CONFIG_RESET_LEVEL 0 //tigger level(0/1)
10. #endif
```

1.4.2 OTA upgrade

in board_xxx_global_build_cfg.h

```
1. #define CONFIG_DOUBLE_BANK_ENABLE 0 // Single and double backup options (
   open macro ,FLASH The structure becomes a dual backup structure, which is suitable for accessing third-party protocols. OTA PS: JL-OTA same
   Also supports dual backup upgrade need according to the actual FLASH Configure the size at the same time CONFIG_FLASH_SIZE)
2. #define CONFIG_APP_OTA_ENABLE 0 // support RCSP Lift
   class (JL-OTA)
```

1.4.3 PID and VID

in board_xxx_global_build_cfg.h

```
1. //DON'T MODIFY THIS CONFIG EXCEPT SDK PUBLISHER
2. #define CONFIG_CHIP_NAME                AC637N          //  remove
   span SDK  Publisher please do not modify
3. //it can be modified before first programming,but keep the same as the o
   riginal version
4. #define CONFIG_PID                      AC637N          // burn
   It can be modified before writing or forced upgrade, and the upgrade must be consistent afterward
5. //it can be modified before first programming,but keep the same as the o
   riginal version
6. #define CONFIG_VID                      0.01 //          Flash or forced upgrade
   It can be modified before and after the upgrade, it should be consistent
```

It should be noted here that all firmwares of the same product need to ensure that CHIP, PID, and VID are consistent, otherwise they will not be able to be programmed and upgraded.

1.4.4 VM Size

```
1. //with single-bank mode,actual vm size should larger this VM_LEAST_SIZE,
   and dual bank mode,actual vm size equals this;
2. #define CONFIG_VM_LEAST_SIZE            8K
```

1.4.5 uboot_debug, ota_debug.bin print port configuration

in board_xxx_global_build_cfg.h

```
1. //isd_download loader/ uboot/ update_loader debug io config
2. //#define CONFIG_UBOOT_DEBUG_PIN                PA05
3. //#define CONFIG_UBOOT_DEBUG_BAUD_RATE          1000000
```

1.4.6 Other Configurations

For other configurations, please refer to the comments in board_xxx_global_build_cfg.h for configuration

Chapter 2 APP Instructions

2.1 APP overview

This chapter mainly introduces the common application solutions of general Bluetooth, so that users can start development from the APP that is closest to the target product.

It has become a Bluetooth application App, rich board-level configuration, peripheral drivers, and Bluetooth libraries, and will continue to integrate more general Bluetooth methods in the future.

case.

2.2 APP - Bluetooth Dual-Mode SPP+BLE

2.2.1 Overview

Support Bluetooth dual-mode transparent transmission function. CLASSIC Bluetooth uses standard serial port SPP profile protocol, BLE Bluetooth uses custom

The defined profile protocol provides ATT's WRITE, WRITE_WITHOUT_RESPONSE, NOTIFY and INDICATE

and other attributes to transmit and receive data.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.2.2 Project configuration

Code project: apps\spp_and_le\board\bd29\AC631N_spp_and_le.cbp

(1) First configure the board-level board_config.h and the Bluetooth dual-mode enable in the corresponding configuration file

```
/*
    * Board-level configuration options
    */
#define CONFIG_BOARD_AC631N_DEMO
// #define CONFIG_BOARD_AC6311_DEMO

#include "board_ac631n_demo_cfg.h"
#include "board_ac6311_demo_cfg.h"
```

//Configure whether to open edr and ble modules in board_ac631n_demo_cfg.h

```
#define TCFG_USER_BLE_ENABLE                1 // BLE function enable
#define TCFG_USER_EDR_ENABLE                1 //EDR function enable
```

(2) Configure app selection: "app_config.h"

```
//apps example choice only 1 to configure the corresponding board_config.h
#define CONFIG_APP_SPP_LE                   1 //SPP + LE or LE's client
#define CONFIG_APP_MULTI                    0 // Bluetooth LE multi-link + spp
#define CONFIG_APP_DONGLE                   0 //usb + Bluetooth became host ),PC hid equipment
```

```
5. #define CONFIG_APP_CENTRAL                0 //ble client,          Central equipment
6. #define CONFIG_APP_LL_SYNC                0 //    Tencent Lianlian
7. #define CONFIG_APP_BEACON                 0 //    BluetoothBLE ibeacon
8. #define CONFIG_APP_NONCONN_24G            0 //2.4G    Connectionless Transceiver
9. #define CONFIG_APP_TUYA                   0 //    Graffiti Protocol
10. #define CONFIG_APP_AT_COM                 0 //AT com HEX    format command
11. #define CONFIG_APP_AT_CHAR_COM            0 //AT com    String format command
12. #define CONFIG_APP_IDLE                   0 //    idle task
```

(3) Configure the corresponding case requirements: "app_config.h"

```
1. #if CONFIG_APP_SPP_LE
2. //    Configure dual mode with the same name and address
3. #define DOUBLE_BT_SAME_NAME               0 //    same name
4. #define DOUBLE_BT_SAME_MAC                0 //    same address
5. #define CONFIG_APP_SPP_LE_TO_IDLE         0 //SPP_AND_LE To IDLE Use
```

(3) BLE configuration of Bluetooth to protect the configuration of GATT and SM

```
1. //    BluetoothBECAME configure
2. #define CONFIG_BT_GATT_COMMON_ENABLE      1 //    Configure using gatt    public module
3. #define CONFIG_BT_SM_SUPPORT_ENABLE       0 //    Configure whether to support encryption
4. #define CONFIG_BT_GATT_CLIENT_NUM         0 //    Configure the host client    number    (app not support)
5. #define CONFIG_BT_GATT_SERVER_NUM         1 //    Configure the slave server    number
6. #define CONFIG_BT_GATT_CONNECTION_NUM     (CONFIG_BT_GATT_SERVER_NUM + CONFIG_BT_GA
   TT_CLIENT_NUM) //    Configure the number of connections
```

2.2.3 SPP data communication

SPP ---Recommended to use mobile phone test tool: "Bluetooth serial port"

1. The code file spp_trans.c

2. Interface description: "spp_trans.c"

(1) SPP module initialization

```
1. void transport_spp_init(void)
2. {
3.         log_info("trans_spp_init\n");
4.         log_info("spp_file: %s", __FILE__);
5. #if (USER_SUPPORT_PROFILE_SPP==1)
6.         spp_state = 0;
7.         spp_get_operation_table(&spp_api);
8.         spp_api->regist_recieve_cbk(0, transport_spp_recieve_cbk);
9.         spp_api->regist_state_cbk(0, transport_spp_state_cbk);
10.        spp_api->regist_wakeup_send(NULL, transport_spp_send_wakeup);
11. #endif
12.
13. #if TEST_SPP_DATA_RATE
14.        spp_timer_handle = sys_timer_add(NULL, test_timer_handler, SPP_TIMER_MS);
15. #endif
16.
17. }
```

(2) SPP connection and disconnection event processing

```
1. static void transport_spp_state_cbk(u8 state)
2. {
3.         spp_state = state;
4.         switch (state) {
5.         case SPP_USER_ST_CONNECT:
6.             log_info("SPP_USER_ST_CONNECT ~~~\n");
7.
8.             break;
9.
10.        case SPP_USER_ST_DISCONN:
11.            log_info("SPP_USER_ST_DISCONN ~~~\n");
```

```
12.             spp_channel = 0;
13.
14.             break;
15.
16.         default:
17.             break;
18.         }
19.
20. }
```

(3) SPP send data interface, call interface transport_spp_send_data_check to check before sending

```
1. int transport_spp_send_data(u8 *data, u16 len)
2. {
3.         if (spp_api) {
4.             log_info("spp_api_tx(%d) \n", len);
5.             /* log_info_hexdump(data, len); */
6.             /* clear_sniff_cnt(); */
7.             bt_comm_edr_sniff_clean();
8.             return spp_api->send_data(NULL, data, len);
9.         }
10.        return SPP_USER_ERR_SEND_FAIL;
11. }
```

(4) SPP checks whether data can be sent to the protocol stack

```
1. int transport_spp_send_data_check(u16 len)
2. {
3.         if (spp_api) {
4.             if (spp_api->busy_state()) {
5.                 return 0;
6.             }
7.         }
8.         return 1;
```

```
9. }
```

(5) SPP send completion callback, indicating that it can continue to send data to the protocol stack, which is used to trigger continued sending of data

```
1. static void transport_spp_send_wakeup(void)
2. {
3.         putchar('W');
4. }
```

(6) SPP receiving data interface

```
1. static void transport_spp_recieve_cbk(void *priv, u8 *buf, u16 len)
2. {
3.         spp_channel = (u16)priv;
4.         log_info("spp_api_rx(%d) \n", len);
5.         log_info_hexdump(buf, len);
6.         clear_sniff_cnt();
7.         ......
```

3. Transceiver test: "spp_trans.c"

The code has realized that after receiving the SPP data of the mobile phone, it will actively send the data back and test the data transmission and reception.

```
1. //loop send data for test
2. if (transport_spp_send_data_check(len)) {
3.         log_info("-loop send\n");
4.         transport_spp_send_data(buf, len);
5. }
```

4. UUID of serial port: "lib_profile_config.c"

The UUID of the serial port is 16bit 0x1101 by default. To modify the customizable 16bit or 128bit UUID, you can modify the S of SDP Information structure sdp_spp_service_data, see the example of filling in 16it and 128bit UUID for details. The primary channel id defaults to 1, Can not be modified.

```
1. #if (USER_SUPPORT_PROFILE_SPP==1)
2. u8 spp_profile_support = 1;
3. SDP_RECORD_HANDLER_REGISTER(spp_sdp_record_item) = {
```

```
4.          .service_record = (u8 *)sdp_spp_service_data,
5.          .service_record_handle = 0x00010004,
6. };
7. #endif
```

Examples of 16it and 128bit UUID filling are as follows:

```
1. /*128 bit uid: 11223344-5566-7788-aabb-8899aabbccdd */
2. const u8 sdp_test_spp_service_data[96] = {
3.          0x36, 0x00, 0x5B, 0x09, 0x00, 0x00, 0x0A, 0x00, 0x01, 0x00, 0x04, 0x09, 0x00, 0x01, 0x36, 0x00,
4.          0x11, 0x1C,
5.
6.          0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0xaa, 0xbb, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, /
    / uuid128
7.
8.          0x09, 0x00, 0x04, 0x36, 0x00, 0x0E, 0x36, 0x00, 0x03, 0x19, 0x01, 0x00, 0x36, 0x00,
9.          0x05, 0x19, 0x00, 0x03, 0x08, 0x01, 0x09, 0x00, 0x09, 0x36, 0x00, 0x17, 0x36, 0x00, 0x14, 0x1C,
10.
11.         0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0xaa, 0xbb, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, /
    / uuid128
12.
13.         0x09, 0x01, 0x00, 0x09, 0x01, 0x00, 0x25, 0x06, 0x4A, 0x4C, 0x5F, 0x53, 0x50, 0x50, 0x00, 0x00,
14. };
15.
16. //spp 16bit uuid 11 01
17. const u8 sdp_spp_update_service_data[70] = {
18.         0x36, 0x00, 0x42, 0x09, 0x00, 0x00, 0x0A, 0x00, 0x01, 0x00, 0x0B, 0x09, 0x00, 0x01, 0x36, 0x00,
19.         0x03, 0x19,
20.
21.         0x11, 0x01, //uuid16
22.
23.         0x09, 0x00, 0x04, 0x36, 0x00, 0x0E, 0x36, 0x00, 0x03, 0x19, 0x01, 0x00,
24.         0x36, 0x00, 0x05, 0x19, 0x00, 0x03, 0x08, 0x01, 0x09, 0x00, 0x09, 0x36, 0x00, 0x09, 0x36, 0x00,
```

```
    0x06, 0x19,

    0x11, 0x01, //uuid16

    0x09, 0x01, 0x00, 0x09, 0x01, 0x00, 0x25, 0x09, 0x4A, 0x4C, 0x5F, 0x53,
    0x50, 0x50, 0x5F, 0x55, 0x50, 0x00,
};
```

2.2.4 BLE data communication

 ---Recommended to use mobile phone test tool: "nRF Connect"

1. The code file ble_trans.c

2. The trans_profile_data data table generated by Profile is placed in ble_trans_profile.h. User available tools make_gatt_services (under the tools directory of the sdk) Customize and modify according to "make_gatt_services tool description.pdf", reconfigure the GATT service and properties, etc.

```
////////////////////////////////////////////////////////////
//
// 0x0001 PRIMARY_SERVICE 1800
//
////////////////////////////////////////////////////////////
0x0a, 0x00, 0x02, 0x00, 0x01, 0x00, 0x00, 0x28, 0x00, 0x18,

/* CHARACTERISTIC, 2a00, READ | WRITE | DYNAMIC, */
// 0x0002 CHARACTERISTIC 2a00 READ | WRITE | DYNAMIC
0x0d, 0x00, 0x02, 0x00, 0x02, 0x00, 0x03, 0x28, 0x0a, 0x03, 0x00, 0x00, 0x2a,
// 0x0003 VALUE 2a00 READ | WRITE | DYNAMIC
0x08, 0x00, 0x0a, 0x01, 0x03, 0x00, 0x00, 0x2a,

```

```
14. ////////////////////////////////////////////////////////
15. //
16. // 0x0004 PRIMARY_SERVICE ae30
17. //
18. ////////////////////////////////////////////////////////
19. 0x0a, 0x00, 0x02, 0x00, 0x04, 0x00, 0x00, 0x28, 0x30, 0xae,
20.
21. /* CHARACTERISTIC, ae01, WRITE_WITHOUT_RESPONSE | DYNAMIC, */
22. // 0x0005 CHARACTERISTIC ae01 WRITE_WITHOUT_RESPONSE | DYNAMIC
23. 0x0d, 0x00, 0x02, 0x00, 0x05, 0x00, 0x03, 0x28, 0x04, 0x06, 0x00, 0x01, 0xae,
24. // 0x0006 VALUE ae01 WRITE_WITHOUT_RESPONSE | DYNAMIC
25. 0x08, 0x00, 0x04, 0x01, 0x06, 0x00, 0x01, 0xae,
```

3. Interface description: "ble_trans.c"

(1) Configure broadcast ADV data

```
1. static int trans_make_set_adv_data(void)
2. {
```

(2) Configure broadcast RESPONE data

```
1. static int trans_make_set_rsp_data(void)
2. {
```

(3) Configure the send buffer size

```
1. #define ATT_LOCAL_PAYLOAD_SIZE      (64*2) //note: need >= 20
2. #define ATT_SEND_CBUF_SIZE          (512) //note: need >= 20,
```

Cache size can be modified

(4) Protocol stack event callback processing, mainly events such as connection and disconnection: "le_gatt_server.c"

```
1. static int trans_event_packet_handler(int event, u8 *packet, u16 size, u8 *ext_param
   )
2. {
3.         /* log_info("event: %02x,size= %d\n",event,size); */
4.         switch (event) {
```

```
5.          case GATT_COMM_EVENT_CONNECTION_COMPLETE:
6.              log_info("connection_handle:%04x\n", little_endian_read_16(packet, 0));
7.              log_info("peer_address_info:");
8.              put_buf(&ext_param[7], 7);
9.
10.             trans_con_handle = little_endian_read_16(packet, 0);
11.             trans_connection_update_enable = 1;
12.
13.             log_info("con_interval = %d\n", little_endian_read_16(ext_param, 14 + 0));
14.             log_info("con_latency = %d\n", little_endian_read_16(ext_param, 14 + 2));
15.             log_info("cnn_timeout = %d\n", little_endian_read_16(ext_param, 14 + 4));
16.             break;
17.
18.         case GATT_COMM_EVENT_DISCONNECT_COMPLETE:
```

(5) ATT read event processing: "ble_trans.c"

```
1. static uint16_t trans_att_read_callback(hci_con_handle_t connection_handle, uint16_t
       att_handle, uint16_t offset, uint8_t *buffer, uint16_t buffer_size)
2. {
3.          uint16_t att_value_len = 0;
4.          uint16_t handle = att_handle;
5.
6.          log_info("read_callback,conn_handle =%04x, handle=%04x,buffer=%08x\n", connectio
    n_handle, handle, (u32)buffer);
7.
8.          switch (handle) {
9.          case ATT_CHARACTERISTIC_2a00_01_VALUE_HANDLE: {
10.             char *gap_name = ble_comm_get_gap_name();
11.             att_value_len = strlen(gap_name);
12.
13.             if ((offset> = att_value_len) || (offset + buffer_size)> att_value_len) {
14.                 break;
```

```
15.             }
16.
17.             if (buffer) {
18.                 memcpy(buffer, &gap_name[offset], buffer_size);
19.                 att_value_len = buffer_size;
```

(6) ATT write event processing: "ble_trans.c"

```
1. static int trans_att_write_callback(hci_con_handle_t connection_handle, uint16_t att
   _handle, uint16_t transaction_mode, uint16_t offset, uint8_t *buffer, uint16_t buffe
   r_size)
2. {
3.     int result = 0;
4.     u16 tmp16;
5.
6.     u16 handle = att_handle;
7.
8.     log_info("write_callback,conn_handle =%04x, handle =%04x,size =%d\n", connection
   _handle, handle, buffer_size);
9.
10.    switch (handle) {
11.
12.    case ATT_CHARACTERISTIC_2a00_01_VALUE_HANDLE:
13.        break;
```

(7) NOTIFY and INDICATE send interface, send pre-call interface app_send_user_data_check check

```
1. static int app_send_user_data(u16 handle, u8 *data, u16 len, u8 handle_type)
2. {
3.     u32 ret = APP_BLE_NO_ERROR;
4.
5.     if (!con_handle) {
6.         return APP_BLE_OPERATION_ERROR;
7.     }
```

```
8.
9.          if (!att_get_ccc_config(handle + 1)) {
10.             log_info("fail,no write ccc!!!,%04x\n", handle + 1);
11.             return APP_BLE_NO_WRITE_CCC;
12.         }
13.
14.         ret = ble_op_multi_att_send_data (con_handle, handle, data, len, handle_type);
```

(8) Check whether data can be sent to the protocol stack

```
1. //       Transceiver test, automatically send received data, for test
2. if (ble_comm_att_check_send(connection_handle, buffer_size)) {
```

(9) Send completion callback, indicating that you can continue to send data to the protocol stack, which is used to trigger continued sending of data

```
1.          case GATT_COMM_EVENT_CAN_SEND_NOW:
2. #if TEST_AUDIO_DATA_UPLOAD
3.              trans_test_send_audio_data(0);
4. #endif
5.              break;
```

4. Transceiver test: "ble_trans.c"

Using the mobile phone NRF software, after connecting the device; after enabling the notification function of the UUID (AE02 and AE05) of notify and indicate;

You can send data to the UUID (AE01 or AE03) of write; test whether the UUID (AE02 or AE05) receives data.

```
1.          case ATT_CHARACTERISTIC_ae01_01_VALUE_HANDLE:
2.              log_info("\n-ae01_rx(%d):", buffer_size);
3.              put_buf(buffer, buffer_size);
4.
5.              // Transceiver test, automatically send received data, for test
6.              if (ble_comm_att_check_send(connection_handle, buffer_size)) {
7.                  log_info("-loop send1\n");
8.                  ble_comm_att_send_data (connection_handle, ATT_CHARACTERISTIC_ae02_01_VAL
    UE_HANDLE, buffer, buffer_size, ATT_OP_AUTO_READ_CCC);
9.              }
```

```
10.             break;
11.
12.         case ATT_CHARACTERISTIC_ae03_01_VALUE_HANDLE:
13.             log_info("\n-ae_rx(%d):", buffer_size);
14.             put_buf(buffer, buffer_size);
15.
16.             // Transceiver test, automatically send received data, for test
17.             if (ble_comm_att_check_send(connection_handle, buffer_size)) {
18.                 log_info("-loop send2\n");
19.                 ble_comm_att_send_data (connection_handle, ATT_CHARACTERISTIC_ae05_01_VAL
    UE_HANDLE, buffer, buffer_size, ATT_OP_AUTO_READ_CCC);
20.             }
21.             break;
```

2.3 APP - Bluetooth Dual-Mode HID

2.3.1 Overview

Standard bluetooth mouse, support bluetooth CLASSIC, bluetooth BLE and 2.4G mode.

Bluetooth mouse supports windows system, mac system, android system, ios system connection.

Supported board levels: bd29, br25, br30, bd19

Supported chips: AC631N, AC6363F, AC6369F, AC6379B, AC6379B

2.3.2 Project configuration

Code project: apps\hid\board\bd29\AC631N_hid.cbp

1. Configuration description

(1) First configure the board-level board_config.h and the Bluetooth dual-mode enable in the corresponding configuration file

```
/*
 * Board-level configuration options
 */
#define CONFIG_BOARD_AC631N_DEMO                // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNER

#include "board_ac631n_demo_cfg.h"
```

//Configure whether to open the edr and ble modules

```
#define TCFG_USER_BLE_ENABLE                    1 //BLE or 2.4G function enable
#define TCFG_USER_EDR_ENABLE                    1 //EDR function enable
```

(2) Configure app selection

```
///app case selection, only choose 1, to configure the corresponding board_config.h
#define CONFIG_APP_MOUSE_SINGLE                 1//Single mode switching #de
```

BLE configuration for Bluetooth, protects GATT and SM configuration

```
//  Bluetooth BECAME configure
#define CONFIG_BT_GATT_COMMON_ENABLE            1 //  Configure using gatt public module
#define CONFIG_BT_SM_SUPPORT_ENABLE             1 //  Configure whether to support encryption
#define CONFIG_BT_GATT_CLIENT_NUM               0 //  Configure the host client number (app not support)
```

```
5. #define CONFIG_BT_GATT_SERVER_NUM                   1 //  Configure the slave server number
6. #define CONFIG_BT_GATT_CONNECTION_NUM               (CONFIG_BT_GATT_SERVER_NUM + CONFIG_BT_GA
    TT_CLIENT_NUM) //  Configure the number of connections
```

(3) Mode selection, configure BLE or 2.4G mode; if you choose 2.4G, the pairing code must be the same as the pairing code of the other party

```
1. //2.4G mode: 0---ble, not 0---2.4G pairing code
2. #define CFG_RF_24G_CODE_ID                          (0) //<=24bits
```

(4) Support dual-mode HID device switching processing

```
1.  static void app_select_btmode(u8 mode)
2. {
3.      if (mode != HID_MODE_INIT) {
4.          if (bt_hid_mode == mode) {
5.              return;
6.          }
7.          bt_hid_mode = mode;
8.      } else {
9.          //init start
10.     }
```

(5) Support entering power saving and low power consumption Sleep

```
1. //*********************************************************************************//
2. //                          Low power configuration                                //
3. //*********************************************************************************//
4. // #define TCFG_LOWPOWER_POWER_SEL                  PWR_DCDC15//
5. #define TCFG_LOWPOWER_POWER_SEL                     PWR_LDO15//
6. #define TCFG_LOWPOWER_BTOSC_DISABLE 0
7. #define TCFG_LOWPOWER_LOWPOWER_SEL SLEEP_EN
8. #define TCFG_LOWPOWER_VDDIOM_LEVEL VDDIOM_VOL_30V
9. #define TCFG_LOWPOWER_VDDIOW_LEVEL VDDIOW_VOL_24V
10. #define TCFG_LOWPOWER_OSC_TYPE                     OSC_TYPE_LRC
```

(6) Support entering soft shutdown, which can be triggered by IO to wake up

```
1. struct port_wakeup port0 = {
2.     .pullup_down_enable = ENABLE,                    //Configure whether the internal pull-up and pull-down of I/O is enabled
3.     .edge               = FALLING_EDGE, //Wakeup mode selection, optional: rising edge\falling edge
4.     .attribute          = BLUETOOTH_RESUME, //Reserve parameters
5.     .iomap              = IO_PORTB_01, //Wakeup port selection
6.     .filter_enable = ENABLE,
7. };
```

```
1.  static void hid_set_soft_poweroff(void)
2. {
3.     log_info("hid_set_soft_poweroff\n");
4.     is_hid_active = 1;
```

(7) System event handling function

```
1.  static int event_handler(struct application *app, struct sys_event *event)
2. {
```

(8) Bluetooth event handler

```
1. static int bt_hci_event_handler(struct bt_event *bt)
2. {
```

(9) Add pairing and binding management

```
1. //2.4 Open pairing management, you can modify the button method yourself
2. #define DOUBLE_KEY_HOLD_PAIR (1 & CFG_RF_24G_CODE_ID) //Press the middle button + right button for a few seconds to enter 2.4G
       pairing mode
3. #define DOUBLE_KEY_HOLD_CNT                (4) //Long press the middle button to count the number of times >= 4s
```

(10) In and out of low-power functions

```
1. void sleep_enter_callback(u8 step)
2. {
3.         /*    This function prohibits adding printing   */
```

```
4.          if (step == 1) {
5.              putchar('<');
6.              APP_IO_DEBUG_1(A, 5);
7.              /*dac_power_off();*/
8.          } else {
9.              close_gpio();
10.         }
11. }
12.
13. void sleep_exit_callback(u32 usec)
14. {
15.         putchar('>');
16.         APP_IO_DEBUG_0(A, 5);
17. }
```

Can be used to count the time of low power sleep.

2.3.3 Directory structure

Taking the mouse APP_MOUSE as an example, the directory structure of the SDK is shown in Figure 1.2.

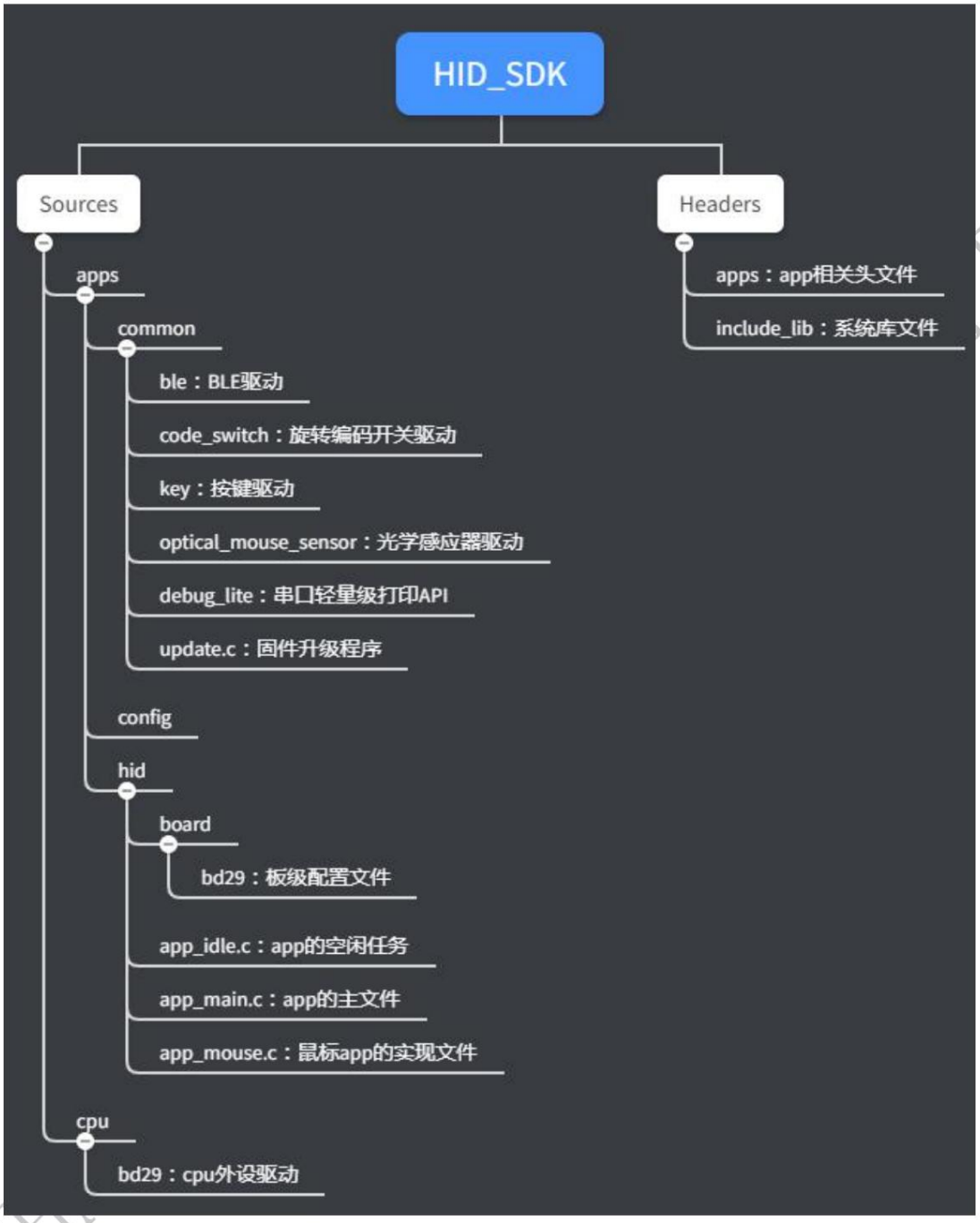


Figure 1.1 HID_SDK directory structure

2.3.3 Board Level Configuration

2.3.3.1 Board-level solution configuration

In order to improve the flexibility of the development process, HID_SDK provides users with several different CPU configurations and corresponding board-level solutions.

Users can choose the corresponding scheme according to their specific development needs. SDK adds standard keyboard application based on BR23 on the basis of the previous version

use.

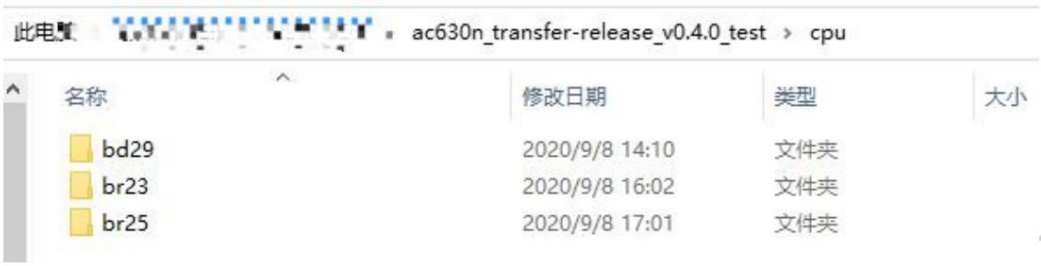


Figure 1.2 CPU Configuration

The path of the board-level scheme configuration file: apps/hid/board/BD29/board_config.h (hid can be replaced with the corresponding app name).

The user only needs to add the corresponding macro definition in the board-level scheme configuration file board_config.h and include the corresponding header file to complete the operation.

Configuration into a board-level solution. An example configuration is shown in the figure.

```
1. /*
2.      * Board-level configuration options
3.      */
4. #define CONFIG_BOARD_AC631N_DEMO                    // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNER
5.
6. #include "board_ac631n_demo_cfg.h"
```

Figure 1.3 Example of board-level solution configuration

2.3.3.2 Board Level Configuration File

The function of the board-level configuration file is to realize the configuration scheme of the same series of different packages, and its storage path is: apps/hid/board/BD29(hid with the appropriate app name). The board level configuration file corresponds to a C file and an H file.

(1) H file

The H file of board-level configuration contains the configuration information of all onboard devices, which is convenient for users to modify the specific device configuration information.

(2) C file

The function of the board-level configuration C file is to initialize the onboard device according to the onboard configuration information contained in the H file.

2.3.3.3 Board level initialization

The system will call the board_init() function in the C file to initialize the onboard device. The board-level initialization process is shown in Figure 1.3. use

Users can add the initialization function of the onboard device in the board_devices_init() function according to the development requirements.

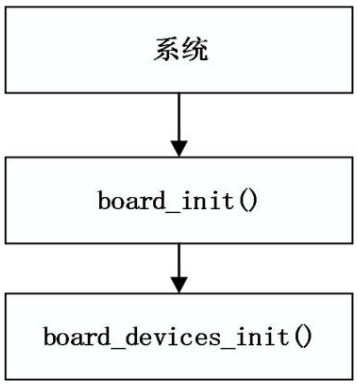


Figure 1.4 Board-level initialization process

2.3.4 APP development framework

2.3.4.1 Overall framework of APP

HID_SDK provides users with an APP development framework based on an event processing mechanism, and users only need to add

The corresponding events and event processing functions can be completed according to the application requirements. The overall framework of the APP is shown in Figure 1.4.

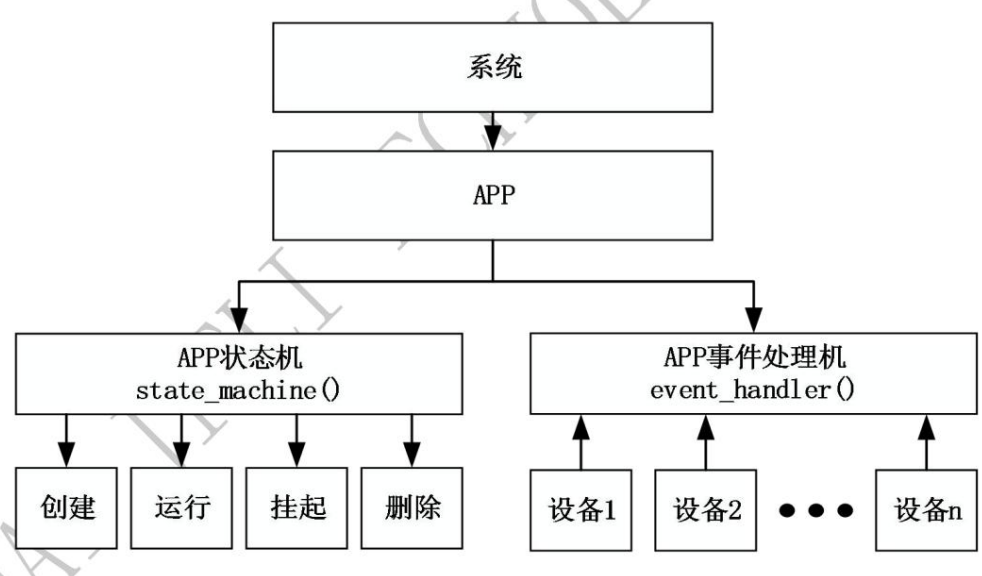


Figure 1.5 Overall framework of APP

(1) APP state machine

During the running process of the system, the state can be switched through the APP state machine, and its state includes creation, running, suspension, and deletion.

(2) APP event handler

APP is based on the event processing mechanism to run. During the operation of the system, the data generated by the hardware device will be in the form of events.

Feedback to the global event list of the system, the system will dispatch the event handler of the APP to run the corresponding event processing function to process it.

APP's event handler implementation function apps/hid/app_mouse.c->event_handler().

2.3.4.2 Events and event handling

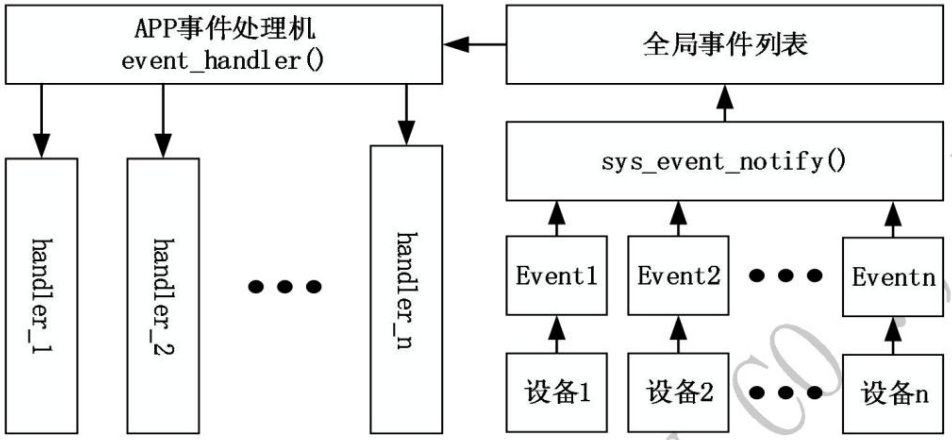


Figure 1.6 Event processing

(1) Definition of event

HID_SDK defines the basic types of events in include_lib/system/event.h, as shown in Figure 1.6. Users can go here

On the basis, add custom event types according to application requirements.

```
1. struct sys_event {
2.     u16 type;
3.     u8 consumed;
4.     void *arg;
5.     union {
6.         struct key_event key;
7.         struct axis_event axis;
8.         struct codesw_event codesw;
```

Figure 1.7 Definition of Events

(2) The occurrence of events

The data generated by the hardware device will be packaged as events. HID_SDK provides an API for sending events. Users can

The APP calls this API to send events to the global event list. The usage of this API is shown in Table 1-1.

Table 1-1 Event sending API

function prototype	void sys_event_notify(struct sys_event *event)
Function	Send data to the global event list
parameter	struct sys_event *event: the event that needs to be sent
return value	none

(3) Handling of events

The system sends messages to the APP by calling the APP's event handler, that is, the apps/hid/app_mouse.c->event_handler() function.

Generated events are processed. This function adopts the switch selection structure, and the user adds corresponding event handlers under different event cases

Just count. An example of the use of the mouse_event_handler() function is shown in Figure 1.7.

```
1.  static int mouse_event_handler(struct application *app, struct sys_event *event)
2. {
3. #if (TCFG_HID_AUTO_SHUTDOWN_TIME)
4. //Reset the no-operation timing count
5.      sys_timer_modify(g_auto_shutdown_timer, TCFG_HID_AUTO_SHUTDOWN_TIME * 1000);
6. #endif
7.
8.      bt_sniff_ready_clean();
9.
10. /* log_info("event: %s", event->arg); */
11. switch (event->type) {
12. case SYS_KEY_EVENT:
13.         /* log_info("Sys Key : %s", event->arg); */
14.         app_key_event_handler(event);
15.         return 0;
16.
17. case SYS_BT_EVENT:
18.         if ((u32)event->arg == SYS_BT_EVENT_TYPE_CON_STATUS) {
19.             bt_connction_status_event_handler(&event->u.bt);
20.         } else if ((u32)event->arg == SYS_BT_EVENT_TYPE_HCI_STATUS) {
21.             bt_hci_event_handler(&event->u.bt);
```

```
22.     }
23.     return 0;
```

Figure 1.8 Example of usage of event_handler() function

2.3.5 Use of keys

2.3.5.1 Use of IOKEY

(1) Configuration instructions

IOKEY parameters are configured in the board configuration file (C file and H file), and can be opened in the H file TCFG_IOKEY_ENABLE Macro and structure configuration (IO port and key connection method) related parameters, the configuration structure parameter description is as follows shown in Table 2-1.

Table 2-1 IOKEY configuration structure description

```
1. struct iokey_port {
2.     union key_type key_type;
3.     u8 connect_way;
4.     u8 key_value;
5. };
```

parameter name	describe
key_type	The key_type structure depends on connect_way, if connect_way is configured as ONE_PORT_TO_LOW and ONE_PORT_TO_HIGH only needs to configure one IO; and if connect_way is configured as DOUBLE_PORT_TO_IO, two IOs need to be configured;
connect_way	Three connection methods are supported: ÿ ONE_PORT_TO_LOW: One port of the button is grounded, and the other port is connected to IO; ÿ ONE_PORT_TO_HIGH: One port of the button is connected to 3.3v, and the other port is connected to IO; ÿ DOUBLE_PORT_TO_IO: The two ports of the button are connected to IO and push each other.
key_value	key_value refers to the key value, the key is detected to be pressed will send the key_value value configured by the key.

(2) Configuration example

The IOKEY parameters are configured in the board-level configuration files (c file and h file). The configuration example is shown in Table 2-2.

Table 2-2 IOKEY configuration example

Hardware configuration

```
1. //iokey configuration
2. #define TCFG_IOKEY_ENABLE                       ENABLE_THIS_MOUDLE
3.
4. #define TCFG_IOKEY_POWER_ENABLE                 ENABLE_THIS_MOUDLE
5. #define TCFG_IOKEY_POWER_CONNECT_WAY ONE_PORT_TO_LOW //One end of the button is connected to a low level
   Terminate IO
6. #define TCFG_IOKEY_POWER_ONE_PORT               IO_PORTB_01
7. #define TCFG_IOKEY_POWER_IN_PORT                NO_CONFIG_PORT
8. #define TCFG_IOKEY_POWER_OUT_PORT               NO_CONFIG_PORT
9.
10. #define TCFG_IOKEY_PREV_ENABLE                  ENABLE_THIS_MOUDLE
11. #define TCFG_IOKEY_PREV_CONNECT_WAY ONE_PORT_TO_LOW //One end of the button is connected to the low end
    ÿ I.
12. #define TCFG_IOKEY_PREV_ONE_PORT                IO_PORTB_00
13. #define TCFG_IOKEY_PREV_IN_PORT                 NO_CONFIG_PORT
14. #define TCFG_IOKEY_PREV_OUT_PORT                NO_CONFIG_PORT
15.
16. #define TCFG_IOKEY_NEXT_ENABLE                  ENABLE_THIS_MOUDLE
17. #define TCFG_IOKEY_NEXT_CONNECT_WAY ONE_PORT_TO_LOW //One end of the button is connected to the low end
    ÿ I.
18. #define TCFG_IOKEY_NEXT_ONE_PORT                IO_PORTB_02
19. #define TCFG_IOKEY_NEXT_IN_PORT                 NO_CONFIG_PORT
20. #define TCFG_IOKEY_NEXT_OUT_PORT                NO_CONFIG_PORT
```

Hardware configuration

```
1. #if TCFG_IOKEY_ENABLE
2. const struct iokey_port iokey_list[] = {
3.     {
4.         .connect_way = ONE_PORT_TO_LOW,                  //Connection method of IO button
5.         .key_type.one_io.port = TCFG_IOKEY_MOUSE_LK_PORT, //The pin corresponding to the IO key
6.         .key_value = KEY_LK_VAL,                         //key value
7.     },
```

```
8.
9.  {
10.     .connect_way = ONE_PORT_TO_LOW,                 //Connection method of IO button
11.     .key_type.one_io.port = TCFG_IOKEY_MOUSE_RK_PORT, //The pin corresponding to the IO key
12.     .key_value = KEY_RK_VAL,                        //key value
13. },
14.
15. {
16.     .connect_way = ONE_PORT_TO_LOW,                 //Connection method of IO button
17.     .key_type.one_io.port = TCFG_IOKEY_MOUSE_HK_PORT, //The pin corresponding to the IO key
18.     .key_value = KEY_HK_VAL,                        //key value
19. },
```

2.3.5.2 Use of ADKEY

(1) Configuration instructions

ADKEY parameters are configured in board configuration files (c and h files), such as board_ac6xxx_mouse.c and board_ac6xxx_mouse_cfg.h, you can open the TCFG_ADKEY_ENABLE macro and structure configuration (IO port and Key connection mode) related parameters, configuration structure parameter description is shown in Table 2-3.

Table 2-3 ADKEY configuration structure description

```
1. struct adkey_platform_data {
2.     u8 enable;
3.     u8 adkey_pin;
4.     u8 extern_up_en; //Whether to use external pull-up, 1: use external pull-up, 0: use internal pull-up 10K
5.     u32 ad_channel;
6.     u16 ad_value[ADKEY_MAX_NUM];
7.     u8 key_value[ADKEY_MAX_NUM];
8. };
```

parameter name	describe
enable	ADKEY enable selection, 1 means enable, 0 means disable

adkey_pin	IO number of AD module
ad_channel	AD channel number
external_up_en	Whether to use external pull-up, 1 means to use external pull-up, 0 means to use internal pull-up (10K)

ad_value[ADKEY_MAX_NUM] The AD value corresponding to the key

key_value[ADKEY_MAX_NUM] key value, when the key is detected to be pressed, it will send the value of the key_value configured by the key.

(2) Configuration example

ADKEY parameters are configured in the board-level configuration files (c file and h file). The configuration example is shown in Table 2-4.

Table 2-4 ADKEY configuration example

H

```
#define TCFG_ADKEY_ENABLE                       DISABLE_THIS_MOUDLE //Whether the AD button is enabled
#define TCFG_ADKEY_PORT                         IO_PORTB_01         //AD button port (note that the selected IO port is
    No support for AD function)
/*AD channel selection, it needs to correspond to the port of the AD button:
AD_CH_PA1 AD_CH_PA3 AD_CH_PA4 AD_CH_PA5
AD_CH_PA9 AD_CH_PA1 AD_CH_PB1 AD_CH_PB4
AD_CH_PB6 AD_CH_PB7 AD_CH_DP AD_CH_DM
AD_CH_PB2
*/
#define TCFG_ADKEY_AD_CHANNEL                   AD_CH_PB1
#define TCFG_ADKEY_EXTERN_UP_ENABLE ENABLE_THIS_MOUDLE //Whether to use external pull-up
```

C

```
#if TCFG_ADKEY_ENABLE
const struct adkey_platform_data adkey_data = {
    .enable = TCFG_ADKEY_ENABLE,                //AD button enable
    .adkey_pin = TCFG_ADKEY_PORT,               //AD button corresponding pin
    .ad_channel = TCFG_ADKEY_AD_CHANNEL, //AD channel value
    .extern_up_en = TCFG_ADKEY_EXTERN_UP_ENABLE, //whether to use an external pull-up resistor
    .ad_value = {                               //The voltage value calculated from the resistance
        TCFG_ADKEY_VOLTAGE0,
        TCFG_ADKEY_VOLTAGE1,
        TCFG_ADKEY_VOLTAGE2,
        TCFG_ADKEY_VOLTAGE3,
        TCFG_ADKEY_VOLTAGE4,
```

```
13.     TCFG_ADKEY_VOLTAGE5,
14.     TCFG_ADKEY_VOLTAGE6,
15.     TCFG_ADKEY_VOLTAGE7,
16.     TCFG_ADKEY_VOLTAGE8,
17.     TCFG_ADKEY_VOLTAGE9,
18. },
19. .key_value = {                  //The key value of each button of the AD button
20.     TCFG_ADKEY_VALUE0,
21.     TCFG_ADKEY_VALUE1,
22.     TCFG_ADKEY_VALUE2,
```

2.3.5.3 Button scan parameter configuration

After IOKEY or ADKEY is enabled, the key scan code will register the timer to regularly scan whether the key is pressed.

Scan parameters can be configured in the file apps/common/key/iokey.c or adkey.c, and the parameters available for configuration are shown in Table 2-5.

Table 2-5 Description of key scan parameter configuration

```
1. //Key-driven scan parameter list
2. struct key_driver_para adkey_scan_para = {
3.     .scan_time = 10, //Key scan frequency, unit: ms
4.     .last_key      = NO_KEY, //The last get_value key value, initialized to NO_KEY;
5.     .filter_time = 2, //Key debounce delay;
6.     .long_time = 75, //The button determines the number of long presses
7.     .hold_time = (75 + 15), //Press the button to determine the number of HOLD
8.     .click_delay_time = 20, //The number of delays to wait for the combo after the button is lifted
9.     .key_type      = KEY_DRIVER_TYPE_AD,
10.    .get_value     = ad_get_key_value,
11. };
```

parameter name	describe
scan_time	The key scan frequency, in ms, the timer will scan the IOKEY regularly according to the set time or ADKEY

last_key	Default initialized to NO_KEY
filter_time	Button debounce time, calculation method: fileter_time * scan_time (ms)
long_time	Button long press event judgment time, calculation method: long_time * scan_time (ms)
hold_time	Press and hold the key and hold the event judgment time, the calculation method is: hold_time * scan_time (ms)
click_delay_time	The delay time of pressing the button to wait for the combo operation, the calculation method is: hold_time * scan_time (ms), note that This parameter configuration will affect the button sensitivity, and also affect the time interval of the combo operation, so in the During the debugging process, you need to select an appropriate parameter value according to your needs;

2.3.5.4 Key event processing

Some key general events implemented in HID_SDK are shown in Table 2-6.

Table 2-6 General event table of keys

key event	illustrate
KEY_EVENT_CLICK	Click event, release after the key is pressed for filter_time and click_delay_time If it is not pressed a second time after the time, the key scan function will be determined as a key click event concurrent cloth out.
KEY_EVENT_LONG	Long press event, when the button is pressed after the filter_time time has elapsed and has been pressed, after the filter_time After the long_time time, the key scan function will determine the key long press event and publish it.
KEY_EVENT_HOLD	Press hold event, when the button is pressed after the filter_time time and has been pressed, after the filter_time After the hold_time time, the key scan function will determine the key press hold event and publish it. After the cloth is finished, if the button is found to be pressed, it will be delayed after hold_time - long_time. Publish the hold down event.
KEY_EVENT_UP	Lift press event, after sending long press event (KEY_EVENT_LONG) and press hold event (KEY_EVENT_HOLD), if the key is released, the key scan function will issue a Press event.
KEY_EVENT_DOUBLE_CLICK	Double-click event, release after the key is pressed for filter_time time and release at click_delay_time The same button was pressed again before, and the button was not pressed again after click_delay_time , the key scan function publishes a double-click event.
KEY_EVENT_TRIPLE_CLICK	Triple-click event, released after the key is pressed for filter_time time and released at click_delay_time The same button was pressed again before, and after click_delay_time, the button was pressed again, If the above-mentioned combo operation is repeated in the subsequent operations, the key scan function will be judged as a triple-click event concurrent cloth out.

After pressing the button to release the message, the APP will receive the message, and the APP can perform related processing according to the button message.

The data format of the key message received by event_handler is shown in Table 2-7. The user can perform related processing according to the received key message operate.

Table 2-7 Key message data format

Data Format	illustrate
event->type key message type is SYS_KEY_EVENT, marked as key message.	
event->u.key.value key value, the value is configured in the configuration IOKEY or ADKEY, marking a certain key is pressed.	
event->u.key.event Key event type, corresponding to the key events listed in the table above.	

2.3.5.5 Button expansion function

HID_SDK provides some general key configuration and message processing methods, if these general mechanisms can not meet the needs of users If required, the user can use the extended function of the button by modifying the configuration.

(1) Combination key function

The IOKEY of HID_SDK only supports the detection of a single button by default. If users need to support combination buttons, they can modify the Change the configuration item of IOKEY to achieve, the specific implementation is as follows:

1) Open the MULT_KEY_ENABLE macro in the H file of the configuration file, and add the combined key value;

2) Configure the key remapping data structure in the C file of the configuration file.

Configuration examples are shown in Table 2-8.

Table 2-8 IOKEY key combination configuration example

H arts piece match set

```
1. #define MOUSE_KEY_SCAN_MODE                 ENABLE_THIS_MOUDLE
2. #define MULT_KEY_ENABLE                     ENABLE
3.
4. #define KEY_LK_VAL                  BIT(0)
5. #define KEY_RK_VAL                  BIT(1)
6. #define KEY_HK_VAL BIT(2)
7. #define KEY_LK_RK_VAL BIT(0) | BIT(1)
8. #define KEY_LK_HK_VAL BIT(0) | BIT(2)
9. #define KEY_RK_HK_VAL BIT(1) | BIT(2)
10. #define KEY_LK_RK_HK_VAL BIT(0) |BIT(1) | BIT(2)
11.
12. #define TCFG_IOKEY_MOUSE_LK_PORT IO_PORTB_03
```

```
#define TCFG_IOKEY_MOUSE_RK_PORT IO_PORTB_02
#define TCFG_IOKEY_MOUSE_HK_PORT IO_PORTB_01
#define TCFG_IOKEY_MOUSE_MK_IN_PORT NO_CONFIG_PORT
#define TCFG_IOKEY_MOUSE_MK_OUT_PORT NO_CONFIG_PORT
```

C arts piece match set

```
const struct key_remap key_remap_table[] = {
    {
        .bit_value = BIT(KEY_LK_VAL) | BIT(KEY_RK_VAL),
        .remap_value = KEY_LK_RK_VAL,
    },

    {
        .bit_value = BIT(KEY_LK_VAL) | BIT(KEY_HK_VAL),
        .remap_value = KEY_LK_HK_VAL,
    },

    {
        .bit_value = BIT(KEY_RK_VAL) | BIT(KEY_HK_VAL),
        .remap_value = KEY_RK_HK_VAL,
    },

    {
        .bit_value = BIT(KEY_LK_VAL) | BIT(KEY_RK_VAL) | BIT(KEY_HK_VAL),
        .remap_value = KEY_LK_RK_HK_VAL,
    },
};

const struct key_remap_data iokey_remap_data = {
    .remap_num = ARRAY_SIZE(key_remap_table),
    .table = key_remap_table,
```

```
26. };
```

(2) button multi-click event

HID_SDK supports single-click, double-click and triple-click events by default. If users need to support more click events, they can modify such as File below:

1) Add a new key event type definition in the event.h file, as shown in Figure 2.1, other multi-clicks can be added at the arrow position event.

```
1. enum {
2.     KEY_EVENT_CLICK,
3. KEY_EVENT_LONG,
4. KEY_EVENT_HOLD,
5.     KEY_EVENT_UP,
6.     KEY_EVENT_DOUBLE_CLICK,
7.     KEY_EVENT_TRIPLE_CLICK,
8.     KEY_EVENT_FOURTH_CLICK,
9.     KEY_EVENT_FIRTH_CLICK,
10. KEY_EVENT_USER,
11. KEY_EVENT_MAX,
12. };
```

Figure 2.2.1 Definition of key event type

2) The judgment of any multi-click event is added to the key_driver_scan() function in key_driver.c, and the addition position is shown in Figure 2.2.

```
1.  if (scan_para->click_delay_cnt > scan_para->click_delay_time) //delay until the button is lifted
2. {
3.     //TODO: You can add any multi-click event here
4.     if (scan_para->click_cnt >= 5)
5.     {
6.         key_event = KEY_EVENT_FIRTH_CLICK; //Five clicks
7.     }
8.     else if (scan_para->click_cnt >= 4)
9.     {
```

```
10.             key_event = KEY_EVENT_FOURTH_CLICK; //4 ÿ
11.         }
12. else if (scan_para->click_cnt >= 3)
13.         {
14.             key_event = KEY_EVENT_TRIPLE_CLICK; //ÿÿ
15.         }
16.     else if (scan_para->click_cnt >= 2)
17.         {
18.             key_event = KEY_EVENT_DOUBLE_CLICK; //ÿÿ
19.         }
20. else
21. {
22.             key_event = KEY_EVENT_CLICK; //click
23. }
24. key_value = scan_para->notify_value;
25. goto _notify;
26. }
27. else //After the button is lifted, wait for the next delay time
28. {
29. scan_para->click_delay_cnt++;
30. goto _scan_end; //The delay time has not expired after the button is lifted, return
31. }
```

Figure 2.2.2 Example of adding multi-click event judgment

3) Finally, the multi-click event is received at the APP layer and processed.

(3) Some keys only respond to click events

This function can be specially processed through a certain bit of the key value. Since the key value (key_value) is currently represented by 1byte, It can support 0 ~ 255 keys, but not so many keys are used in most applications, so the key value is currently used in HID_SDK

The 7th bit of (key_value) is marked. When the key is scanned, the multi-click judgment processing is not performed on the key marked with bit (7).

To apply this function, you only need to mark the key value in the board-level configuration file, as shown in Figure 2.3.

```
1.     {
2.         .connect_way = ONE_PORT_TO_LOW,                    //Connection method of IO button
```



```
3.      .key_type.one_io.port = TCFG_IOKEY_MOUSE_LK_PORT, //The pin corresponding to the IO key
4.      .key_value = KEY_LK_VAL,                          //key value
5. },
```

Figure 2.2.3 Example of adding multi-click event judgment

2.3.6 Use of serial port

The initialization parameters of the serial port are configured in the board configuration files (c file and h file), such as board_ac6xxx_mouse.c and board_ac6xxx_mouse_cfg.h, enable TCFG_UART0_ENABLE macro and structure configuration related parameters in h file, in C Add the initialization data structure to the file, and the configuration example is shown in Table 2-9. After the serial port initialization is completed, the user can call apps/debug.c The functions in the file perform serial port printing operations.

Table 2-9 Serial port configuration example

H arts piece match set

```
1. //*****************************************************************************//
2. //                          UART configuration                                 //
3. //*****************************************************************************//
4. #define TCFG_UART0_ENABLE                ENABLE_THIS_MOUDLE
5. #define TCFG_UART0_RX_PORT               NO_CONFIG_PORT
6. #define TCFG_UART0_TX_PORT               IO_PORT_DP // IO_PORTA_05
7. #define TCFG_UART0_BAUDRATE              1000000
```

C arts piece match set

```
1. /************************** UART config*****************************/
2. #if TCFG_UART0_ENABLE
3. UART0_PLATFORM_DATA_BEGIN(uart0_data)
4. .tx_pin = TCFG_UART0_TX_PORT,                //The serial port prints the TX pin selection
5. .rx_pin = TCFG_UART0_RX_PORT,                //Serial port print RX pin selection
6.      .baudrate = TCFG_UART0_BAUDRATE,        // serial port baud rate
7.      .flags = UART_DEBUG,                    //The serial port is used to print and the parameter needs to be set to UART_DEBUG
8. UART0_PLATFORM_DATA_END()
9. #endif //TCFG_UART0_ENABLE
```

2.3.7 Mouse Report Map

The Mouse Report Map is defined in the apps/common/ble/le_hogp.c file, as shown in Figure 2.3.1.

```
1. 0x05, 0x01,          // Usage Page (Generic Desktop Ctrls)
2.     0x09, 0x02,      // Usage (Mouse)
3.     0xA1, 0x01,      // Collection (Application)
4.     0x85, 0x01,      // Report ID (1)
5.     0x09, 0x01,      // Usage (Pointer)
6.     0xA1, 0x00,      // Collection (Physical)
7.     0x95, 0x05,      // Report Count (5)
8.     0x75, 0x01,      // Report Size (1)
9.     0x05, 0x09,      // Usage Page (Button)
10.    0x19, 0x01,      // Usage Minimum (0x01)
11.    0x29, 0x05,      // Usage Maximum (0x05)
12.    0x15, 0x00,      // Logical Minimum (0)
13.    0x25, 0x01,      // Logical Maximum (1)
14.    0x81, 0x02,      // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
15.    0x95, 0x01,      // Report Count (1)
16.    0x75, 0x03,      // Report Size (3)
17.    0x81, 0x01,      // Input (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
18.    0x75, 0x08,      // Report Size (8)
19.    0x95, 0x01,      // Report Count (1)
20. 0x05, 0x01,         // Usage Page (Generic Desktop Ctrls)
21. 0x09, 0x38,         // Usage (Wheel)
22. 0x15, 0x81,         // Logical Minimum (-127)
23. 0x25, 0x7F,         // Logical Maximum (127)
24. 0x81, 0x06,         // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
25. 0x05, 0x0C,         // Usage Page (Consumer)
26. 0x0A, 0x38, 0x02, // Usage (AC Pan)
27. 0x95, 0x01,         // Report Count (1)
28. 0x81, 0x06,         // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
29. 0xC0,               // End Collection
```

```
30. 0x85, 0x02,     // Report ID (2)
31. 0x09, 0x01,     // Usage (Consumer Control)
32. 0xA1, 0x00,     // Collection (Physical)
33. 0x75, 0x0C,     // Report Size (12)
34. 0x95, 0x02,     // Report Count (2)
35. 0x05, 0x01,     // Usage Page (Generic Desktop Ctrls)
36. 0x09, 0x30,     // Usage (X)
37. 0x09, 0x31,     // Usage (Y)
38. 0x16, 0x01, 0xF8, // Logical Minimum (-2047)
39. 0x26, 0xFF, 0x07, // Logical Maximum (2047)
40. 0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
41. 0xC0,           // End Collection
42. 0xC0,           // End Collection
43. 0x05, 0x0C,     // Usage Page (Consumer)
44. 0x09, 0x01,     // Usage (Consumer Control)
45. 0xA1, 0x01,     // Collection (Application)
46. 0x85, 0x03,     // Report ID (3)
47. 0x15, 0x00,     // Logical Minimum (0)
48. 0x25, 0x01,     // Logical Maximum (1)
49. 0x75, 0x01,     // Report Size (1)
50. 0x95, 0x01,     // Report Count (1)
51. 0x09, 0xCD,     // Usage (Play/Pause)
52. 0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
53. 0x0A, 0x83, 0x01, // Usage (AL Consumer Control Configuration)
54. 0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
55. 0x09, 0xB5,     // Usage (Scan Next Track)
56. 0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
57. 0x09, 0xB6,     // Usage (Scan Previous Track)
58. 0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
59. 0x09, 0xEA,     // Usage (Volume Decrement)
60. 0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
61. 0x09, 0xE9,     // Usage (Volume Increment)
```

```
0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
0x0A, 0x25, 0x02, // Usage (AC Forward)
0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
0x0A, 0x24, 0x02, // Usage (AC Back)
0x81, 0x06,     // Input (Data,Var,Rel,No Wrap,Linear,Preferred State,No Null Position)
0x09, 0x05,     // Usage (Headphone)
0x15, 0x00,     // Logical Minimum (0)
0x26, 0xFF, 0x00, // Logical Maximum (255)
0x75, 0x08,     // Report Size (8)
0x95, 0x02,     // Report Count (2)
0xB1, 0x02,     // Feature (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position,Non-volatile)
0xC0,           // End Collection
```

Figure 2.3.1 Mouse Report Map

The analysis of Mouse Report Map can be realized by online analysis tools, and users can modify the Report Map according to their needs.

Report Map online parsing tool address: http://eleccelerator.com/usbdescreqparser/.

2.3.8 Overall Framework of Bluetooth Mouse **APP**

The overall framework of the Bluetooth mouse APP is shown in Figure 2.3.2.

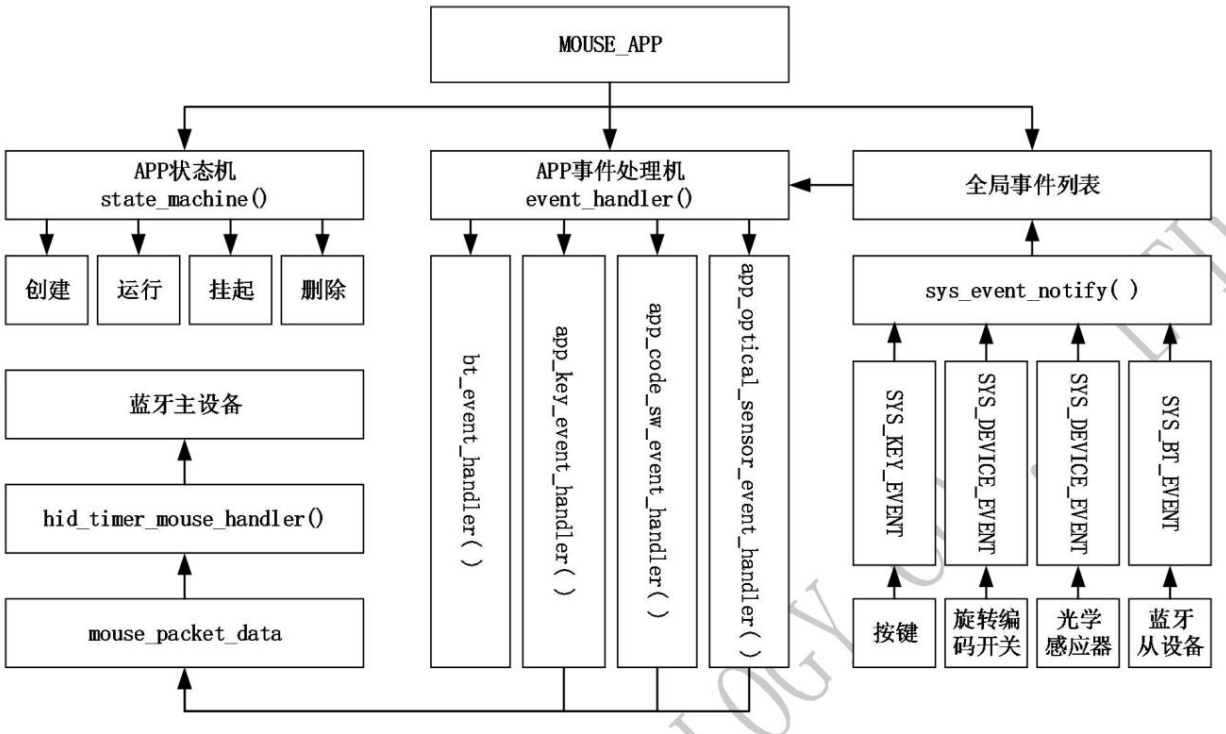


Figure 2.3.2 Overall frame diagram of Bluetooth mouse APP

(1) APP registration and operation

The app's registration information is included in the apps/hid/app_mouse.c file, as shown in Figure 3.3, the system will root

Based on this information, the registration of the APP is completed.

```
1. /*
2. * Register AT Module mode
3.      */
4. REGISTER_APPLICATION(app_mouse) = {
5.          .name = "mouse",
6.          .action = ACTION_MOUSE_MAIN,
7.          .ops = &app_mouse_ops,
8.          .state = APP_STA_DESTROY,
9. };
```

Figure 2.3.3 APP registration information

After the system is initialized, the system will schedule the app_task task, which calls the apps/hid/app_main.c->app_main() function number, start running app_mouse.

(2) The occurrence of events

The data collection of mouse buttons, rotary coding switches, and optical sensors is completed in the interrupt service function of the system software timer. The collected data will be packaged as corresponding events and reported to the global event list periodically.

(3) Handling of events

The system will schedule the APP's event handler (app_mouse.c->event_handler()) to call the corresponding event according to the event type. file handler function.

app_key_event_handler()ÿapp_code_sw_event_handler()ÿapp_optical_sensor_event_handler()ÿÿ

apps/hid/app_mouse.c files are used to handle key events, rotary encoder switch events, and optical sensor events, respectively. The contained data will be filled in mouse_packet_data and saved.

(4) Sending of data

When the Bluetooth device is initialized, a system software timer is set to periodically send the Bluetooth master device mouse_packet_data data, the interrupt service function of the system timer is:

app / common / ble / le_hogp.c-> hid_timer_mouse_handler ()ÿ

2.3.9 Bluetooth mouse power consumption

(1) Optical sensor data used

PAW3205DB-TJ3T

Power Supply	**Operating voltage** **2.1V ~ 3.6V (VDD)**
Typical Operating Current (without I/O toggling)	**1.5mA @ Mouse moving (Normal)** **100uA @ Mouse not moving (Sleep1)** **15uA @ Mouse not moving (Sleep2)** **10uA @ Power down mode** *not including LED, typical value

available, normal mode and sleep1 mode. After 256 ms (typical) not moving during normal mode, the mouse sensor will enter sleep1 mode, and keep on sleep1 mode until motion detected or

After 20 sec (typical) not moving during sleep1 mode, the mouse sensor will enter sleep2 mode

(2) Test conditions

1. Interval in ble connection state: 6*1.25 ms = 7.5ms, lantency: 100;

2ÿRadio TX: 7.2 dBm

3ÿDCDCÿVDDIOM 3.0VÿVDDIOW 2.4V

4. Short **circuit between VDDIO** and **VBAT**

(3) Chip power consumption

(1. The hardware is not connected to the module; 2. The software closes all modules)			
		no action	soft shutdown
2.1V	Maximum current	1,338 mA	1 uA
	Average current	139 uA	1 uA
	Minimum current	30 uA	1 uA
2.6V	Maximum current	852 uA	2 uA
	Average current	110 uA	2 uA
	Minimum current	33 uA	1 uA
3.0V	Maximum current	965 uA	2 uA
	Average current	110 uA	2 uA
	Minimum current	33 uA	2 uA
3.3V	Maximum current	813 uA	3 uA
	Average current	113 uA	3 uA
	Minimum current	35 uA	2 uA

(4) Power consumption of the whole machine

(1. The hardware is connected to all modules; 2. The software enables all modules)				
		sleep1 (after 256 ms of inactivity)	sleep2 (after 20.48 seconds of inactivity)	soft shutdown (after 1 minute of inactivity)
2.1V	Maximum current	2,894 mA	1,253 mA	65 uA
	Average current	230 uA	140 uA	12 uA

	Minimum current	95 uA	29 uA	1 uA
2.6V	Maximum current	1,008 mA	1,062 mA	63 uA
	Average current	193 uA	125 uA	13 uA
	Minimum current	111 uA	31 uA	3 uA
3.0V	Maximum current	864 uA	864 uA	55 uA
	Average current	190 uA	120 uA	13 uA
	Minimum current	109 uA	37 uA	6 uA
3.3V	Maximum current	858 uA	783 uA	53 uA
	Average current	185 uA	139 uA	14 uA
	Minimum current	114 uA	39 uA	7 uA

2.4 APP - Bluetooth Dual-Mode AT Moudle

2.4.1 Overview

The main function is that on the basis of ordinary data transmission BLE and EDR, the host computer or other MCU can communicate with each other through UART. Connect to the Bluetooth chip for basic configuration, status acquisition, control scanning, disconnection, and data transmission and reception.

AT control transparent transmission supports slave mode and master mode. You can only choose one of the two when compiling. Slave mode supports dual mode and master mode. Only BLE is supported.

Define a set of serial port control protocol, please refer to the protocol document "Bluetooth AT Protocol" for details.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.4.2 Project Configuration

Code project: apps\spp_and_le\board\bd29\AC631N_spp_and_le.cbp

(1) First configure the board-level board_config.h and the Bluetooth dual-mode enable in the corresponding configuration file

```
1. /*
2.      * Board-level configuration options
3.      */
4. #define CONFIG_BOARD_AC631N_DEMO                    // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNER
5.
6. #include "board_ac631n_demo_cfg.h"
```

(2) Configure the corresponding board_acxxx_demo_cfg.h file to enable BLE or EDR (the host does not support EDR), to board_ac630x_demo_cfg.h as an example

```
3. #define TCFG_USER_BLE_ENABLE                1 //BLE function enable
4. #define TCFG_USER_EDR_ENABLE                1 //EDR function enable
```

(3) Configure app_config.h, enable AT

```
3. //app case selection, only one can be selected, and the corresponding board_config.h must be configured
4. #define CONFIG_APP_SPP_LE                   0
5. #define CONFIG_APP_AT_COM                   1
```



```
6. #define CONFIG_APP_DONGLE                    0
```

(4) Configure app_config.h, select AT master or AT slave (choose one of two)

```
7. //app case selection, only one can be selected, and the corresponding board_config.h needs to be configured
8. #define CONFIG_APP_AT_COM                    01//AT com HEX format command
9.
10.
11. #define TRANS_AT_COM                         0
12.     #define TRANS_AT_CLIENT                  1 //Select host AT
```

2.4.3 Main description code

code file	Description
app_at_com.c	The main realization of the task, the process
at_uart.c	Serial port configuration, data sending and receiving
at_cmds.c	AT protocol parsing and processing
ble_at_com.c	Slave ble control implementation
spp_at_com.c	spp control implementation
ble_at_client.c	Host ble control implementation

2.5 APP - Bluetooth Dual-Mode Central

2.5.1 Overview

The central device uses the client role of GATT, which is implemented as a host master in the SDK.

to search and connect to other BLE devices. After the connection is successful, traverse the Services information data of the slave GATT. Supports up to 16 Sevices traverse.

The example of the SDK is to search for Services in the BLE device of Jerry's data transmission SDK.

Run a search to filter other Services.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.5.2 Project Configuration

Code project: apps\spp_and_le\board\bd29\AC631N_spp_and_le.cbp

(1) First configure the board-level board_config.h and the Bluetooth ble enable in the corresponding configuration file

```
1. /*
2.      * Board-level configuration options
3.      */
4. #define CONFIG_BOARD_AC631N_DEMO                    // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNER
5.
6. #include "board_ac631n_demo_cfg.h"
```

// The configuration only supports the ble module

```
5. #define TCFG_USER_BLE_ENABLE                1 //BLE function enable
6. #define TCFG_USER_EDR_ENABLE                0 //EDR function enable
```

(2) Configure app selection

```
13. //app case selection, only one can be selected, the corresponding board_config.h must be configured
14. #define CONFIG_APP_CENTRAL                  1 //ble client, central device
```

2.5.3 Main code description

The BLE implementation file ble_central.c is responsible for module initialization, processing GATT module events, and command and data control sending.

1. Configure the basic elements of the GATT client module.

```
1. static const sm_cfg_t cetl_sm_init_config = {
2.
3. static gatt_ctrl_t cetl_gatt_control_block = {
4.
5. static const gatt_client_cfg_t central_client_init_cfg = {
6.              .event_packet_handler = cetl_client_event_packet_handler,
7. };
```

2. Configure the searched GATT service and record the searched information, support 16bit and 128bit UUID.

(1) Configure scan matching devices

```
1. //      Configuring Multiple Scan Match Devices
2. static const u8 cetl_test_remoter_name1[] = "AC897N_MX(BLE)";//
3. static client_match_cfg_t cetl_match_device_table[] = {
4. #if MATCH_CONFIG_NAME
```

Searching for matching devices will issue event handling

```
1. case GATT_COMM_EVENT_SCAN_DEV_MATCH: {
2.                  log_info("match_dev:addr_type= %d\n", packet[0]);
3.                  put_buf(&packet[1], 6);
4.                  if (packet[8] == 2) {
5.                      log_info("is TEST_BOX\n");
6.                  }
7.                  client_match_cfg_t *match_cfg = ext_param;
8.                  if (match_cfg) {
9.                      log_info("match_mode: %d\n", match_cfg->create_conn_mode);
10.                     if (match_cfg->compare_data_len) {
11.                         put_buf(match_cfg->compare_data, match_cfg->compare_data_len);
12.                     }
```

```
            }
        }
        break;
```

(2) Configure the uuid searched after connecting

```
// specified search uuid
// specified search uuid
static const target_uuid_t jl_cetl_search_uuid_table[] = {

    // for uuid16
    // PRIMARY_SERVICE, ae30
    // CHARACTERISTIC, ae01, WRITE_WITHOUT_RESPONSE | DYNAMIC,
    // CHARACTERISTIC, ae02, NOTIFY,

    {
        .services_uuid16 = 0xae30,
        .characteristic_uuid16 = 0xae01,
        .opt_type = ATT_PROPERTY_WRITE_WITHOUT_RESPONSE,
    },

    {
        .services_uuid16 = 0xae30,
        .characteristic_uuid16 = 0xae02,
        .opt_type = ATT_PROPERTY_NOTIFY,
    },
```

A matching UUID and handle is found, the event is handled, and the operation handle can be logged.

```
case GATT_COMM_EVENT_GATT_SEARCH_MATCH_UUID: {
        opt_handle_t *opt_hdl = packet;
        log_info("match:server_uuid= %04x,charactc_uuid= %04x,value_handle= %04x\n",
\
```

```
4.                          opt_hdl->search_uuid->services_uuid16, opt_hdl->search_uuid->charac
    teristic_uuid16, opt_hdl->value_handle);
5. #if cetl_TEST_WRITE_SEND_DATA
6.             // for test
7.             if (opt_hdl->search_uuid->characteristic_uuid16 == 0xae01) {
8.                 cetl_ble_client_write_handle = opt_hdl->value_handle;
9.             }
10. #endif
11.         }
12.         break;
```

2. Configure scan and connection parameters

The scan parameters are set in units of 0.625ms, as shown in the figure below:

```
1. #define SET_SCAN_TYPE SCAN_ACTIVE
2. #define SET_SCAN_INTERVAL 48
3. #define SET_SCAN_WINDOW 16
```

The connection parameter interval is in units of 1.25ms, and the timeout is in units of 10ms, as shown in the following figure:

```
1. #define SET_CONN_INTERVAL 0x30
2. #define SET_CONN_LATENCY 0
3. #define SET_CONN_TIMEOUT 0x180
```

Please modify the above two sets of parameters carefully, and must be defined and modified according to the Bluetooth protocol specification.

2.6 APP - Bluetooth Dual-Mode Dongle

2.6.1 Overview

The Bluetooth dongle complies with USB and BLE or EDR transmission standards, and has the characteristics of plug and play, convenience and practicality. it can be used for blue

The data transmission between the dental devices enables the computer to perform wireless connection and data communication with the surrounding Bluetooth devices, and automatically discover and manage the data.

It manages remote Bluetooth devices, resources and services, and realizes binding and automatic connection between Bluetooth devices.

The Bluetooth dongle supports two connection modes: BLE and 2.4G; supports connection to specify the Bluetooth name or mac address; the application example is to connect Pick up Jerry's mouse.

The bluetooth dongle supports EDR, supports connecting to the specified bluetooth name or mac address, and connects to Jerry's keyboard device.

The Bluetooth dual-mode dongle cannot be turned on at the same time, which will reduce the search efficiency.

Supported board levels: br25, br30, bd19, br34

Supported chips: AC636N, AC637N, AC632N, AC638N

2.6.2 Project Configuration

Code project: apps\spp_and_le\board\br25\AC636N_spp_and_le.cbp

(1) APP selection, configure app_config.h

```
15. //app case selection, only one can be selected, and the corresponding board_config.h must be configured
16. #define CONFIG_APP_DONGLE                    1//usb + bluetooth (ble host), PC hid device
```

Using EDR mode, the following host search enabled must also be printed

```
17. #define EDR_EMITTER_EN                      1 //Bluetooth (edr host)
```

(2) Board level selection, configure board_config.h. Currently only AC6368B_DONGLE board supports Bluetooth dongle

```
18. //#define CONFIG_BOARD_AC636N_DEMO
19. #define CONFIG_BOARD_AC6368B_DONGLE //CONFIG_APP_DONGLE
20. // #define CONFIG_BOARD_AC6363F_DEMO
```

(3) To enable USB and BLE, you need to configure board_ac6368b_dongle_cfg.h

```
21. #define TCFG_PC_ENABLE                      ENABLE_THIS_MOUDLE//PC module enable
22. #define TCFG_UDISK_ENABLE                   DISABLE_THIS_MOUDLE//U disk module enable
```

```
23. #define TCFG_OTG_USB_DEV_EN                    BIT(0)//USB0 = BIT(0) USB1 = BIT(1)
```

```
24. #define TCFG_USER_BLE_ENABLE                   1 //BLE or 2.4G function enable
25. #define TCFG_USER_EDR_ENABLE                   0 //EDR function enable
```

(4) Mode selection, configure BLE or 2.4G mode; if 2.4G pairing code is selected, the pairing code must be the same as that of the other party

```
26. //2.4G mode: 0---ble, not 0---2.4G pairing code
27. #define CFG_RF_24G_CODE_ID                     (0) //<=24bits
```

(5) If the BLE mode is selected, the Bluetooth dongle defaults to connecting the slave by the Bluetooth name, and the Bluetooth name of the connected slave needs to be configured.

```
1. static const u8 dongle_remoter_name1[] = "AC696X_1(BLE)";//
2. static const u8 dongle_remoter_name2[] = "AC630N_HID123(BLE)";// Automatically connect the slave with the same name
```

(6) By default, power on for 10 seconds and pair the device with a short distance according to the signal strength rssi. If the pairing fails, stop searching. link back to an existing configuration to the device.

```
1. //The dongle is powered on to enable pairing management. If the pairing fails and there is no paired device, stop searching.
2. #define POWER_ON_PAIR_START                    (1)//
3. #define POWER_ON_PAIR_TIME                     (10000)//unit ms, continuous pairing search time
4. #define DEVICE_RSSI_LEVEL                      (-50)
5. #define POWER_ON_KEEP_SCAN                     (0)//Pairing failed, keep searching for pairing
```

2.6.3 Main code description

Bluetooth dongle implementation files app_dongle.c and ble_dg_central, responsible for module initialization, processing protocol stack events and commands Data control sending, etc.

(1) HID descriptor, described as a mouse

```
1. static const u8 sHIDReportDesc[] = {
2.     0x05, 0x01,                    // Usage Page (Generic Desktop Ctrls)
3.     0x09, 0x02,                    // Usage (Mouse)
4.     0xA1, 0x01,                    // Collection (Application)
5.     0x85, 0x01,                    // Report ID (1)
6.     0x09, 0x01,                    // Usage (Pointer)
```

```
7.
```

(2) Use the specified uuid to communicate with the slave, which needs to cooperate with the slave, saving the time of searching for uuid

```
8. static const target_uuid_t dongle_search_ble_uuid_table[] = {
9.          {
10.             .services_uuid16 = 0x1812,
11.             .characteristic_uuid16 = 0x2a4d,
12.             .opt_type = ATT_PROPERTY_NOTIFY,
13.         },
14.
15.         {
16.             .services_uuid16 = 0x1812,
17.             .characteristic_uuid16 = 0x2a33,
18.             .opt_type = ATT_PROPERTY_NOTIFY,
19.         },
20.
21.         {
22.             .services_uuid16 = 0x1801,
23.             .characteristic_uuid16 = 0x2a05,
24.             .opt_type = ATT_PROPERTY_INDICATE,
25.         },
26.
27. };
```

```
1. /*
2. Determine the 3 notify handles left for the slave to send data
3. */
4. static const u16 mouse_notify_handle[3] = {0x0027, 0x002b, 0x002f};
```

(3) Used to monitor Bluetooth data reporting, and forward Bluetooth data to PC through USB

```
1. //edr          Receive device data
```

```
2. static void dongle_edr_hid_input_handler(u8 *packet, u16 size, u16 channel)
3. {
4.         log_info("hid_data_input:chl=%d,size=%d", channel, size);
5.         put_buf(packet, size);
```

```
1. // ble      Receive device data
2. void dongle_ble_hid_input_handler(u8 *packet, u16 size)
3. {
4. #if TCFG_PC_ENABLE
5.         hid_send_data(packet, size);
6. #endif
7. }
```

(4) Configure the connection mode configuration of BLE, you can choose to connect by address or device name, etc.

```
1. static const client_match_cfg_t dg_match_device_table[] = {
2.         {
3.             .create_conn_mode = BIT(CLI_CREAT_BY_NAME),
4.             .compare_data_len = sizeof(dg_test_remoter_name1) - 1, //      go to terminator
5.             .compare_data = dg_test_remoter_name1,
6.         },
```

2.7 APP - Bluetooth DualMode Keyboard

2.7.1 Overview

This case is a keyboard device based on HID, which can be used for media playback, volume control for up and down songs, and supports Android and IOS The dual system, and supports BLE and EDR two working modes.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.7.2 Project Configuration

Code project: apps\hid\board\br23\AC635N_hid.cbp

(1) First perform APP selection configuration (apps\hid\include\app_config.h)

```
1. //app case selection, only choose 1, to configure the corresponding board_config.h
2. #define CONFIG_APP_KEYBOARD                1//hid button, default case
3. #define CONFIG_APP_KEYFOB                  0//Selfie, board_ac6368a,board_6318
4. #define CONFIG_APP_MOUSE                   0//mouse, board_mouse
5. #define CONFIG_APP_STANDARD_KEYBOARD       0//Standard HID keyboard, board_ac6351d
6. #define CONFIG_APP_KEYPAGE                 0//pager
```

(2) Configure the board level board_config.h (apps\hid\board\bd29\board_config.h), the following figure shows the selection of the AC631n board level, or to select the ac6313 board level.

```
1. /*
2. * Board configuration options
3.      */
4. #define CONFIG_BOARD_AC631N_DEMO // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNER

5. // #define CONFIG_BOARD_AC6313_DEMO // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNER

6. #include "board_ac631n_demo_cfg.h"
7. #include "board_ac6302a_mouse_cfg.h"
8. #include "board_ac6319a_mouse_cfg.h"
```

```
#include "board_ac6313_demo_cfg.h"
#include "board_ac6318_demo_cfg.h"

#endif
```

//Configure whether to open the edr and ble modules

```
#define TCFG_USER_BLE_ENABLE        1 //BLE function enable
#define TCFG_USER_EDR_ENABLE        1 //EDR function enable
```

After selecting the corresponding board level and APP, other settings and initialization are based on the default settings, and the APP can be run.

2.7.3 Main code description

(1) APP registration and operation

```
#if (CONFIG_APP_MOUSE)
    it.name = "mouse";
    it.action = ACTION_MOUSE_MAIN;
#elif(CONFIG_APP_KEYFOB)
    it.name = "keyfob";
    it.action = ACTION_KEYFOB;
#elif(CONFIG_APP_KEYBOARD)
    it.name = "hid_key";


    start_app(&it);
}
```

First add the hid_key application branch in the app_main.c function, and then register the application.

```
REGISTER_APPLICATION(app_hid) = {
    .name = "hid_key",
    .action = ACTION_HID_MAIN,
    .ops = &app_hid_ops,
    .state = APP_STA_DESTROY,
};
```

Follow the code above to register the app and execute the configured app. Then enter APP_state_machine, according to the state machine

Different states execute different branches, enter the APP_STA_CREATE branch at the first execution, and execute the corresponding app_start(). open

Start executing app_start() in this function to initialize the clock, select the Bluetooth mode, enable key messages and other initialization operations.

The key enabling enables the system to respond in time when an external key event occurs, and perform event processing.

```
1. REGISTER_LP_TARGET(app_hidkey_lp_target) = {
2.         .name = "app_hidkey_deal",
3.         .is_idle = hidkey_app_idle_query,
4. };
5. static const struct application_operation app_hidkey_ops = {
6.         .state_machine = hidkey_state_machine,
7.         .event_handler = hidkey_event_handler,
8. };
```

After the keyboard application is registered, the above app_hid_ops is processed. Divided into two modules hidkey_state_machine

and hidkey_event_handler. The execution process is roughly as follows, and the corresponding function is located in the app_keyboard.c file:

hidkey_state_machine()--->app_start()--->sys_key_event_enable(). The clock is initialized mainly according to the state of the application

Configuration, Bluetooth name setting, reading configuration information, message button enable and other configurations.

hidkey_event_handler(struct application *app, struct sys_event

*event)--->hidkey_key_event_handler(sys_event

*event)--->hidkey_key_deal_test(key_type,key_value)---->hidkey_key_deal_test(key_msg). event processing flow

The process is roughly as shown above. hidkey_event_handler() selects the corresponding processing branch according to the incoming second parameter event type, this

Select to execute the key event, and then call the key event handler to perform the corresponding event processing according to the key value and key type of the event.

reason.

(2) APP event processing mechanism

1 Generation and definition of events

The data collection of external events is completed in the interrupt service function of the system software timer, and the collected data will be packaged into corresponding

Events are periodically reported to the global event list. .

void sys_event_notify(struct sys_event *e)ÿ

This function is an event notification function, which is called when an event occurs in the system.

2 Handling of events

The main event processing in this case includes connection event processing and button event processing. The common entry of the event processing function is

After event_handler(), different functions are called to implement response processing of different types of events.

1. Bluetooth connection event processing

After the APP runs, the first Bluetooth connection event processing, Bluetooth initialization, HID descriptor interpretation, Bluetooth mode formula selection, etc. Call hidkey_event_handler(), hidkey_bt_connction_status_event_handler() function to realize Bluetooth connection Wait for the event.

2. Button event processing

The key event processing is realized by calling the hidkey_key_event_handler() function in the hidkey_event_handler() function.

(3) Data transmission

The custom keyboard descriptor looks like this:

```
1. static const u8 hidkey_report_map[] = {
2.         0x05, 0x0C,         // Usage Page (Consumer)
3.         0x09, 0x01,         // Usage (Consumer Control)
4.         0xA1, 0x01,         // Collection (Application)
5.         0x85, 0x01,         // Report ID (1)
6.         0x09, 0xE9,         // Usage (Volume Increment)
7.         0x09, 0xEA,         // Usage (Volume Decrement)
8.         0x09, 0xCD,         // Usage (Play/Pause)
9.         0x09, 0xE2,         // Usage (Mute)
10.        0x09, 0xB6,         // Usage (Scan Previous Track)
11.        0x09, 0xB5,         // Usage (Scan Next Track)
12.        0x09, 0xB3,         // Usage (Fast Forward)
13.        0x09, 0xB4,         // Usage (Rewind)
14.        0x15, 0x00,         // Logical Minimum (0)
15.        0x25, 0x01,         // Logical Maximum (1)
16.        0x75, 0x01,         // Report Size (1)
17.        0x95, 0x10,         // Report Count (16)
18.        0x81, 0x02,         // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Nu
       ll Position)
19.        0xC0,               // End Collection
20.        // 35 bytes
21. };
```

Data is sent according to different key values and key types. Part of the send function is entered as follows:

```
if (key_msg) {
    printf("key_msg = %02x\n", key_msg);
    if (bt_hid_mode == HID_MODE_EDR) {
        edr_hid_key_deal_test(key_msg);
        bt_sniff_ready_clean();
    } else {
#if TCFG_USER_BLE_ENABLE
        ble_hid_key_deal_test(key_msg);
#endif
    }
    return;
```

2.8 APP - Bluetooth DualMode Keyfob

2.8.1 Overview

This case is mainly used for the realization of bluetooth Selfie. After the following configuration, open the mobile phone bluetooth to connect the device to take corresponding pictures operate. Since the use of the Selfie uses LEDs, the LEDs should also be set correspondingly in this case. After the Selfie device is powered on, Before bluetooth is not connected, the LED flashes at a certain frequency until it is connected or goes into sleep mode. After bluetooth connection The LED is off, only when the button is pressed, the LED will be connected at the same time, and the working status of the Selfie can be judged by the status of the LED.

Support board level: bd29, br25, br30, bd19

Support chip: AC6318, AC6368A, AC6379B, AC6328A

2.8.2 Project Configuration

Code project: apps\hid\board\bd19\AC632N_hid.cbp

(1) Configure the app Select (apps\hid\include\app_config.h), select the corresponding Selfie application as shown below.

```
1.  //app case selection, only choose 1, to configure the corresponding board_config.h
2. #define CONFIG_APP_KEYFOB                       1//Selfie, board_ac6368a,board_6318
```

(2) First configure the board-level board_config.h (apps\hid\board\bd29\board_config.h), and select the corresponding development board.

```
1. //#define CONFIG_BOARD_AC631N_DEMO // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURN
   IS
2. // #define CONFIG_BOARD_AC6319A_MOUSE // CONFIG_APP_MOUSE
3.  #define CONFIG_BOARD_AC6318_DEMO // CONFIG_APP_KEYFOB
4.
5. #include "board_ac631n_demo_cfg.h"
6. #include "board_ac6318_demo_cfg.h"
```

(3) Board configuration (board_ac6318_demo_cfg.h)

```
1. //*********************************************************************************//
2. //                                LED configuration                                //
3. //*********************************************************************************//
```

```
4. #define TCFG_PWMLED_ENABLE          ENABLE_THIS_MOUDLE //Whether IO push light module is supported, bd29
    no PWM module
5. #define TCFG_PWMLED_IO_PUSH         ENABLE             //The IO push light used by the LED
6. #define TCFG_PWMLED_IOMODE          LED_ONE_IO_MODE    //LED mode, single IO or two IO
7. #define TCFG_PWMLED_PIN             IO_PORTB_01        //IO port used by LED
```

//Configure whether to open the edr and ble modules

```
9. #define TCFG_USER_BLE_ENABLE        0 //BLE function enable
10. #define TCFG_USER_EDR_ENABLE        1 //EDR function enable
```

2.8.3 Main code description

(1) APP registration and operation

```
1. REGISTER_LP_TARGET(app_hid_lp_target) = {
2.      .name = "app_hid_deal",
3.      .is_idle = app_hid_idle_query,
4. };
5.
6. static const struct application_operation app_hid_ops = {
7.      .state_machine = state_machine,
8.      .event_handler = event_handler,
9. };
10. /*
11. * Register AT Module mode
12. */
13. REGISTER_APPLICATION(app_hid) = {
14. .name = "keyfob",
15.     .action = ACTION_KEYFOB,
16.     .ops = &app_hid_ops,
17. .state = APP_STA_DESTROY,
18. };
```

Follow the code above to register the app and execute the configured app. Then enter APP_state_machine, according to the state machine

Different states execute different branches, enter the APP_STA_CREATE branch at the first execution, and execute the corresponding app_start(). open

Start executing app_start() in this function to initialize the clock, select the Bluetooth mode, enable key messages and other initialization operations.

The key enabling enables the system to respond in time when an external key event occurs, and perform event processing.

(2) APP event processing mechanism

1. Generation and definition of events

The data collection of external events is completed in the interrupt service function of the system software timer, and the collected data will be packaged into corresponding

Events are periodically reported to the global event list.

void sys_event_notify(struct sys_event *e)ÿ

This function is an event notification function, which is called when an event occurs in the system.

2. Handling of events

The main event processing in this case includes connection event processing, button event processing and LED event processing.

The common entry is event_handler(). After that, different functions are called to realize the response processing of different types of events.

2.1 Bluetooth connection event processing

After the APP runs, the first Bluetooth connection event processing, Bluetooth initialization, HID descriptor interpretation, Bluetooth mode

Formula selection, etc. The second parameter of the function, according to the different events, passes in different event types and executes different branches, as shown in the following figure:

```
1.  static int keyfob_event_handler(struct application *app, struct sys_event *event)
2. {
3.
4. #if (TCFG_HID_AUTO_SHUTDOWN_TIME)
5. //Reset the no-operation timing count
6.      sys_timer_modify(g_auto_shutdown_timer, TCFG_HID_AUTO_SHUTDOWN_TIME * 1000);
7. #endif
8.
9.      bt_sniff_ready_clean();
10.
11. /* log_info("event: %s", event->arg); */
12. switch (event->type) {
13.      case SYS_KEY_EVENT:
14.          /* log_info("Sys Key : %s", event->arg); */
15.          app_key_event_handler(event);
16.          return 0;
```

The following figure shows the Bluetooth connection event processing function, which performs Bluetooth initialization and mode selection.

```
1.  static int bt_connction_status_event_handler(struct bt_event *bt)
```



```
2. {
3.
4.      log_info("----------------------bt_connction_status_event_handler %d", bt->event);
5.
6.      switch (bt->event) {
7.      case BT_STATUS_INIT_OK:
8.          /*
9.           * Bluetooth initialization completed
10.          */
11.         ı
```

Call keyfob_event_handler(), bt_connction_status_event_handler() functions to implement events such as Bluetooth connection.

2.2 Button event processing and **LED** event processing

Enter the key event processing flow by calling the app_key_event_handler() function, and enter the key event processing flow according to the key type and key value.

Enter the app_key_deal_test() and key_value_send() functions for event processing.

```
1.  static void app_key_event_handler(struct sys_event *event)
2. {
3.      /* u16 cpi = 0; */
4.      u8 event_type = 0;
5.      u8 key_value = 0;
6.
7.      if (event->arg == (void *)DEVICE_EVENT_FROM_KEY) {
8.          event_type = event->u.key.event;
9.          key_value = event->u.key.value;
10.         printf("app_key_evnet: %d,%d\n", event_type, key_value);
11.         app_key_deal_test(event_type, key_value);
12.     }
13. }
```

```
1.  static void app_key_deal_test(u8 key_type, u8 key_value)
2. {
3.      u16 key_msg = 0;
4.
```

```
#if TCFG_USER_EDR_ENABLE
    if (!edr_hid_is_connected()) {
        if (bt_connect_phone_back_start (1)) { // Cycle
            return;
        }
    }
```

The figure below shows the LED working state part of the realization function.

```
static void led_on_off(u8 state, u8 res)
{
    /* if(led_state != state || (state == LED_KEY_HOLD)){ */
    if (1) { //The same state also needs to update the time
        u8 prev_state = led_state;
        log_info("led_state: %d>>>%d", led_state, state);
        led_state = state;
        led_io_flash = 0;

    }
```

3. Data transmission

KEYFOB belongs to the category of HID devices. The definition and transmission of data should be determined according to the content of the HID device descriptor.

The descriptor of the figure shows that the descriptor is a user-defined descriptor, which can be combined to achieve various required functions. There are two

Input entity descriptor. Each function button corresponds to a bit, a total of 11 bits, and the remaining 13 bits of constant input entities, so

The packet length of the custom descriptor is 3 bytes. If the user needs to add events of different key types on the basis of the Selfie, they can

To add this function in the following descriptor, and then set the corresponding key value and key type in the key processing function branch,

to achieve the corresponding function.

The user-defined descriptor constitutes the KEYFOB descriptor of this case, and implements the corresponding button function.

```
static const u8 keyfob_report_map[] = {
    //General button
    0x05, 0x0C,         // Usage Page (Consumer)
    0x09, 0x01,         // Usage (Consumer Control)
    0xA1, 0x01,         // Collection (Application)
    0x85, 0x03,         // Report ID (3)
    0x15, 0x00,         // Logical Minimum (0)
```

```
0x25, 0x01,                 // Logical Maximum (1)
0x75, 0x01,                 // Report Size (1)
0x95, 0x0B,                 // Report Count (11)
0x0A, 0x23, 0x02, // Usage (AC Home)
0x0A, 0x21, 0x02, // Usage (AC Search)
0x0A, 0xB1, 0x01, // Usage (AL Screen Saver)
0x09, 0xB8,                 // Usage (Eject)
0x09, 0xB6,                 // Usage (Scan Previous Track)
0x09, 0xCD,                 // Usage (Play/Pause)
0x09, 0xB5,                 // Usage (Scan Next Track)
0x09, 0xE2,                 // Usage (Mute)
0x09, 0xEA,                 // Usage (Volume Decrement)
0x09, 0xE9,                 // Usage (Volume Increment)
0x09, 0x30,                 // Usage (Power)
0x0A, 0xAE, 0x01, // Usage (AL Keyboard Layout)
0x81, 0x02,                 // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Nu
ll Position)
0x95, 0x01,                 // Report Count (1)
0x75, 0x0D,                 // Report Size (13)
0x81, 0x03,                 // Input (Const,Var,Abs,No Wrap,Linear,Preferred State,No N
ull Position)
0xC0,                       // End Collection

// 119 bytes
};
```

The key_big_press/null in the figure represents the data packet of the button press and lift in the custom descriptor that realizes the volume increase.

```
static void key_value_send(u8 key_value, u8 mode) {
    void (*hid_data_send_pt)(u8 report_id, u8 * data, u16 len) = NULL;

    if (bt_hid_mode == HID_MODE_EDR) {
```

```
5. #if TCFG_USER_EDR_ENABLE
6.          hid_data_send_pt = edr_hid_data_send;
7. #endif
8.      } else {
9. #if TCFG_USER_BLE_ENABLE
10.         hid_data_send_pt = ble_hid_data_send;
11. #endif
12.     }
13.
14. if (!hid_data_send_pt) {
15.         return;
16. }
```

2.9 APP - Bluetooth DualMode KeyPage

2.9.1 Overview

This APP is developed based on HID. It is mainly used to browse the current popular small videos such as Douyin, page up and down, left and right menu switching, temporary Stop and wait. First select the application that needs to be used for this case selection, and then select the corresponding support board level. For details, please refer to the following step. Compile and download the software to the corresponding development board, turn on the Bluetooth of the mobile phone to connect, and enter the video browsing interface to operate the corresponding press key.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC638N

2.9.2 Project Configuration

Code project: apps\hid\board\bd19\AC632N_hid.cbp

(1) app configuration

Find the corresponding file (apps\hid\include\app_config.h) in the project code to select the APP. In this case, select the pager.

The result is shown below:

```
1. //app case selection, only choose 1, to configure the corresponding board_config.h
2. #define CONFIG_APP_KEYPAGE                    1//Page turner
```

(2) Board level selection

Then in the file (apps\hid\board\bd29\board_config.h), make the corresponding board level selection as follows:

```
1. /*
2. * Board configuration options
3.       */
4. #define CONFIG_BOARD_AC631N_DEMO // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNE
       R
5. // #define CONFIG_BOARD_AC6313_DEMO // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURN
       IS
6.
7. #include "board_ac631n_demo_cfg.h"
8. #include "board_ac6302a_mouse_cfg.h"
9. #include "board_ac6319a_mouse_cfg.h"
```

```
#include "board_ac6313_demo_cfg.h"
#include "board_ac6318_demo_cfg.h"

#endif
```

The configuration is to select the corresponding board_ac631n_demo board level.

//Configure whether to open the edr and ble modules

```
#define TCFG_USER_BLE_ENABLE        0 //BLE function enable
#define TCFG_USER_EDR_ENABLE        1 //EDR function enable
```

2.9.3 Main code description

(1) APP registration (the function is located in apps/hid/app_keypage.c)

During system initialization, APP registration is performed according to the following information. The general process of execution is:

REGISTER_APPLICATION--->state_machine--->app_start()--->sys_key_event_enable();This process is mainly carried out

The initial settings of the device and some functions are enabled.

REGISTER_APPLICATION--->event_handler--->app_key_event_handler()--->app_key_deal_test();This process

There are multiple cases under event_handler. The above-mentioned processing flow for selecting key events is described in the code flow, mainly showing key events.

When the event occurs, the processing flow of the program.

```
REGISTER_LP_TARGET(app_hid_lp_target) = {
    .name = "app_keypage",
    .is_idle = app_hid_idle_query,
};
static const struct application_operation app_hid_ops = {
    .state_machine = state_machine,
    .event_handler = event_handler,
};
* Registration mode
REGISTER_APPLICATION(app_hid) = {
.name = "keypage",
.action = ACTION_KEYPAGE,
    .ops = &app_hid_ops,
```

```
14. .state = APP_STA_DESTROY,
15. };
```

(2) APP state machine

The state machine has create, start, pause, resume, stop, destroy states, and executes corresponding branches according to different states.

After the APP is registered, perform the initial operation, enter the APP_STA_START branch, and start the APP operation.

```
1.  static int state_machine(struct application *app, enum app_state state, struct intent *it)
2. { switch (state) {
3.      case APP_STA_CREATE:
4.          break;
5.      case APP_STA_START:
6.          if (!it) {
7.              break;          }
8.          switch (it->action) {
9.          case ACTION_TOUCHSCREEN:
10.             app_start();
```

After entering the app_start() function, perform corresponding initialization, clock initialization, mode selection, low-power initialization, and external event enable can.

```
1.  static void app_start()
2. {
3.      log_info("======================================");
4.      log_info("----------------------------KEYPAGE----------------------");
5.      log_info("======================================");
6.
```

(3) APP event processing mechanism

1. The definition of the event (the code is located in Headers\include_lib\system\even.h)

```
1. struct sys_event {
2.      u16 type;
3.      u8 consumed;
4.      void *arg;
5.      union {
6.          struct key_event key;
7.          struct axis_event axis;
```

```
8.          struct codesw_event codesw;
```

(4) Event generation (include_lib\system\event.h)

```
void sys_event_notify(struct sys_event *e);
```

Event notification function, the system calls this function when an event occurs.

(5) Event processing (app_keypage.c)

The general flow of function execution is: evevt_handler()--->app_key_event_handler()--->app_key_deal_test().

```
1.  static int event_handler(struct application *app, struct sys_event *event)
2. {
3. #if (TCFG_HID_AUTO_SHUTDOWN_TIME)
4. //Reset the no-operation timing count
5.      sys_timer_modify(g_auto_shutdown_timer, TCFG_HID_AUTO_SHUTDOWN_TIME * 1000);
6. #endif
7.
```

```
1.  static void app_key_event_handler(struct sys_event *event)
2. {
3.      /* u16 cpi = 0; */
4.      u8 event_type = 0;
5.      u8 key_value = 0;
6.
7.      if (event->arg == (void *)DEVICE_EVENT_FROM_KEY) {
8.          event_type = event->u.key.event;
9.          key_value = event->u.key.value;
10.         printf("app_key_evnet: %d,%d\n", event_type, key_value);
11.         app_key_deal_test(event_type, key_value);
12.     }
13. }
```

```
1.  static void app_key_deal_test(u8 key_type, u8 key_value)
2. {
```

```
3.      u16 key_msg = 0;
4.      void (*hid_data_send_pt)(u8 report_id, u8 * data, u16 len) = NULL;
5.
6.      log_info("app_key_evnet: %d,%d\n", key_type, key_value);
7.
8.      }
```

(6) Sending of APP data

After the APP is registered and running, when a key event occurs, the corresponding data will be sent. Since it is a HID device, the data will be sent.

The send form is generated from the descriptor of the corresponding HID device. If users need to customize the functions of the device, they can combine the HID official documents.

file to modify the following descriptors. Some of the descriptors are as follows:

```
1.  static const u8 hid_report_map[] = {
2.      // 119 bytes
3.      0x05, 0x0D,         // Usage Page (Digitizer)
4.      0x09, 0x02,         // Usage (Pen)
5.      0xA1, 0x01,         // Collection (Application)
6.      0x85, 0x01,         // Report ID (1)
7.      0x09, 0x22,         // Usage (Finger)
8.      0xA1, 0x02,         // Collection (Logical)
9.      0x09, 0x42,         // Usage (Tip Switch)
10.     0x15, 0x00,         // Logical Minimum (0)
11.     0x25, 0x01,         // Logical Maximum (1)
12.     0x75, 0x01,         // Report Size (1)
13.     0x95, 0x01,         // Report Count (1)
14.     0x81, 0x02,         // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
15.     0x09, 0x32,         // Usage (In Range)
16.     0x81, 0x02,         // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
17.     0x95, 0x06,         // Report Count (6)
18.     0x81, 0x03,         // Input (Const,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
19.     0x75, 0x08,         // Report Size (8)
20. 0x09, 0x51,             // Usage (0x51)
21. 0x95, 0x01,             // Report Count (1)
22. 0x81, 0x02,             // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
```

```
23. 0x05, 0x01,                    // Usage Page (Generic Desktop Ctrls)
```

```
1.    static const u8 pp_press[7] = {0x07,0x07,0x70,0x07,0x70,0x07,0x01};
2. static const u8 pp_null[7]= {0x00,0x07,0x70,0x07,0x70,0x07,0x00};
```

The above figure shows that the HID device corresponding to the pause button sends data packets, and the data is transmitted through hid_data_send_pt() in the figure below.

```
1. log_info("point: %d,%d", point_cnt, point_len);
2.         if (point_cnt) {
3.             for (int cnt = 0; cnt < point_cnt; cnt++) {
4.                 hid_data_send_pt(1, key_data, point_len);
5.                 key_data += point_len;
6.                 KEY_DELAY_TIME();
7.             }
8.         }
```

It can be seen from the descriptor that the device has a total of 5 input entities Input, which constitute a total of 7 bytes of data, so the corresponding pause button

The data packet consists of 7bytes of data, the first 2bytes indicate whether there is touch input, and the middle 2bytes indicate the y coordinate and

x coordinate, the last 1byte represents the contact count, different key events correspond to different data packets, and the data passes hid_data_send_dt

The function is sent to the device. The corresponding key event realizes the corresponding function through the event processing mechanism and data transmission.

(6) Increase the processing of public messages

Follow up to identify different mobile phone systems to switch descriptors.

```
1. static int app_common_event_handler(struct bt_event *bt)
2. {
3.             log_info("--------app_common_event_handler: %02x %02x", bt->event, bt->value);
4.
5.             switch (bt->event) {
6.             case COMMON_EVENT_EDR_REMOTE_TYPE:
7.                 log_info(" COMMON_EVENT_EDR_REMOTE_TYPE \n");
8.                 connect_remote_type = bt->value;
9.                 if (connect_remote_type == REMOTE_DEV_IOS) {
10.                     user_hid_set_ReportMap(hid_report_map_ios, sizeof(hid_report_map_ios));
11.                 } else {
12.                     user_hid_set_ReportMap(hid_report_map, sizeof(hid_report_map));
```

```
13.             }
14.             break;
15.
16.         case COMMON_EVENT_BLE_REMOTE_TYPE:
17.             log_info(" COMMON_EVENT_BLE_REMOTE_TYPE \n");
18.             connect_remote_type = bt->value;
```

2.10 APP - Bluetooth DualMode Standard Keyboard

2.10.1 Overview

This case is mainly used for the realization of a standard dual-mode Bluetooth keyboard. After the device is turned on, it enters the pairing state. Bluetooth 3.0 and Bluetooth 4.0 devices are connected. After the connection is successful, the other one will disappear. The SDK supports 4 device connections by default Keyboard (only one can be connected at the same time), switch devices by pressing keys.

Support board level: br23, bd19

Support chip: AC6351D, AC6321A

AC6321A keyboard solution, a new key scanning module AD15N is added, except that the key scanning is slightly different, other parts The implementation method is the same as that of AC6351D. In addition, download the packaged ex_mcu.bin file, and you can modify the batch processing. tools\download\data_trans\download.batÿ

amend as below:

..\..\isd_download.exe ..\..\isd_config.ini -tonorflash -dev bd19 -boot 0x2000 -div8 -wait 300 -uboot ..\..\uboot.boot -app ..\..\app.bin ..\..\cfg_tool.bin -res ..\..\p11_code.bin ..\..\ex_mcu.bin -uboot_compress

2.10.2 Project Configuration

Code project: apps\hid\board\br23\AC635N_hid.cbp

(1) Configure app selection (apps\hid\include\app_config.h), select the corresponding standard keyboard application as shown in the figure below.

```
1.  //app case selection, only choose 1, to configure the corresponding board_config.h
2. #define CONFIG_APP_STANDARD_KEYBOARD 1//Standard HID keyboard,board_ac6351d
```

(2) First configure the board-level board_config.h (apps\hid\board\br23\board_config.h), and select the corresponding development board.

```
7. #define CONFIG_BOARD_AC635N_DEMO
8. //#define CONFIG_BOARD_AC6351D_KEYBOARD
9.
10. #include "board_ac635n_demo_cfg.h"
```

```
11. #include "board_ac6351d_keyboard_cfg.h"
```

(3) Function configuration (board_ac6351d_keyboard_cfg.h)

//Configure to open edr or ble module

```
13. #define TCFG_USER_BLE_ENABLE                    0 //BLE function enable
14. #define TCFG_USER_EDR_ENABLE                    1 //EDR function enable
```

```
1. //*********************************************************************//
2. //                          Matrix button configuration                //
3. //*********************************************************************//
4. #define TCFG_MATRIX_KEY_ENABLE                  ENABLE_THIS_MOUDLE
5.
6. //*********************************************************************//
7. //                          touchpad configuration                     //
8. //*********************************************************************//
9. #define TCFG_TOUCHPAD_ENABLE                    ENABLE_THIS_MOUDLE
```

(4) IO configuration (board_ac6351d_keyboard.c)

Configure matrix scan row and column IO

```
1. static u32 key_row [] = {IO_PORTB_06, IO_PORTB_07, IO_PORTB_08, IO_PORTB_09, IO_PORTB_10, IO_PORTB_11,
       IO_PORTC_06, IO_PORTC_07};
2. static u32 key_col [] = {I_PORT_00, I_PORT_01, I_PORT_02, I_PORT_03,
3. IO_PORTA_04, IO_PORTA_05, IO_PORTA_06, IO_PORTA_07, \
4. IO_PORTA_08, IO_PORTA_09, IO_PORTA_10, IO_PORTA_11, IO_PORTA_12, IO_PORTA_13, IO_PORTC_00, IO_PORTA_
       14, IO_PORTC_01, IO_PORTA_15, IO_PORTB_05,
5. };
```

Configure the touchpad IIC communication interface

```
1. const struct soft_iic_config soft_iic_cfg[] = {
2.     //iic0 data
3.     {
4.         .scl = TCFG_SW_I2C0_CLK_PORT,                   // IIC CLK leg
5.         .sda = TCFG_SW_I2C0_DAT_PORT,                   // IIC DAT ÿ
6.         .delay = TCFG_SW_I2C0_DELAY_CNT,                //Software IIC delay parameter, affects the communication clock frequency
```

```
7.          .io_pu = 1,                                    //Whether to open the pull-up resistor, if the external circuit does not have a pull-up resistor welded
            to be set to 1
8.      },
9. };
```

(5) Wakeup port configuration (board_ac6351d_keyboard.c)

After the keyboard enters low power consumption, it needs to wake up the cpu by pressing the button. The 635N supports 8 common IOs, LVD wake-up, and LDOIN wake-up.

Normal IO wakeup:

```
1. struct port_wakeup port0 = {
2.          .pullup_down_enable = ENABLE,                  //Configure whether the internal pull-up and pull-down of I/O is enabled
3.          .edge               = FALLING_EDGE,            //Wakeup mode selection, optional: rising edge \ falling edge
4.          .attribute          = BLUETOOTH_RESUME,        // keep parameters
5.          .iomap              = IO_PORTB_06,             // wakeup port selection
6.          .filter_enable      = ENABLE,
7. };
8. const struct wakeup_param wk_param = {
9.          .port[0] = &port0,
10.         ...
11.         ...
12.         .port[7] = &port7,
13.         .sub = & sub_wkup,
14.         .charge = &charge_wkup,
15. };
```

LVD wakeup:

lvd_extern_wakeup_enable();//It is necessary to choose whether to use LVD wakeup according to the package, 6531C package LVD is PB4

LDOIN wake up

LDOIN wake-up is charging wake-up

(6) Configuration of key values (app_standard_keyboard.c)

The keyboard's key-value table matrix_key_table and fn key remapping key-value table are defined in the app_standard_keyboard.c file

fn_remap_event also needs other key events other_key_map

ÿ matrix_key_table defines the key values of the standard Keyboard, such as RCTRL, LCTRL, A, B, etc..., the user according to the scheme

Select the key core to modify the keyboard key value table, the corresponding key value function is defined in `apps/common/usb/host/usb_hid_keys.h`

ÿ fn_remap_event is divided into two types, one is for system control, such as volume up key, search and search, etc., and the other is for customer

Custom functions, such as Bluetooth switching, etc., is_user_key is 0 means that the key is used for system control, and the key value can be found in COUSTOM_CONTROL page.

ÿ When Is_user_key is 1, it means that it is used for automatic definition of keys, which has nothing to do with standard HID. The processing of related keys is in user_key_deal deal with.

2.10.3 Main code description

(1) APP registration and operation

```
1. REGISTER_LP_TARGET(app_hid_lp_target) = {
2.         .name = "app_hid_deal",
3.         .is_idle = app_hid_idle_query,
4. };
5.
6. static const struct application_operation app_hid_ops = {
7.         .state_machine = state_machine,
8.         .event_handler = event_handler,
9. };
10. /*
11. * Register AT Module mode
12. */
13. REGISTER_APPLICATION(app_hid) = {
14. .name = "hid_key",
15. .action = ACTION_KEYFOB,
16.         .ops = &app_hid_ops,
17. .state = APP_STA_DESTROY,
18. };
```

Follow the code above to register the app and execute the configured app. Then enter APP_state_machine, according to the state machine Different states execute different branches, enter the APP_STA_CREATE branch at the first execution, and execute the corresponding app_start(). open Start executing app_start() in this function to initialize the clock, select the Bluetooth mode, enable key messages and other initialization operations. The key enabling enables the system to respond in time when an external key event occurs, and perform event processing.

(2) APP event processing mechanism

1. Generation and definition of events

The data collection of external events is completed in the interrupt service function of the system software timer, and the collected data will be packaged into corresponding Events are periodically reported to the global event list.

void sys_event_notify(struct sys_event *e)ÿ

This function is an event notification function, which is called when an event occurs in the system.

2. Handling of events

The main event processing in this case includes connection event processing, button event processing and LED event processing. The common entry is event_handler(). After that, different functions are called to realize the response processing of different types of events.

2.1 Bluetooth connection event processing

After the APP runs, the first Bluetooth connection event processing, Bluetooth initialization, HID descriptor interpretation, Bluetooth mode Formula selection, etc. The second parameter of the function, according to the different events, passes in different event types and executes different branches, as shown in the following figure:

```
1.  static int event_handler(struct application *app, struct sys_event *event)
2. {
3.
4. #if (TCFG_HID_AUTO_SHUTDOWN_TIME)
5. //Reset the no-operation timing count
6.      sys_timer_modify(g_auto_shutdown_timer, TCFG_HID_AUTO_SHUTDOWN_TIME * 1000);
7. #endif
8.
9.      bt_sniff_ready_clean();
10.
11. /* log_info("event: %s", event->arg); */
12. switch (event->type) {
13. case SYS_KEY_EVENT:
14.      /* log_info("Sys Key : %s", event->arg); */
15.      app_key_event_handler(event);
16.      return 0;
```

The following figure shows the Bluetooth connection event processing function, which performs Bluetooth initialization and mode selection.

```
1.  static int bt_connction_status_event_handler(struct bt_event *bt)
2. {
3.
4.      log_info("----------------------bt_connction_status_event_handler %d", bt->event);
```

```
5.
6.      switch (bt->event) {
7.      case BT_STATUS_INIT_OK:
8.          /*
9.           * Bluetooth initialization completed
10.          */
11.         ı
```

Call event_handler(), bt_connction_status_event_handler() functions to implement events such as Bluetooth connection.

2.2 Key event processing and touchpad event processing

Enter the key event processing flow by calling the app_key_event_handler() function, and enter according to the key type and key value.

The app_key_deal_test() and key_value_send() functions perform event processing.

```
1. void matrix_key_map_deal(u8 *map)
2. {
3.          u8 row, col, i = 0;
4.          static u8 fn_press = 0;
5.
6.          if (special_key_deal(map, fn_remap_key, sizeof(fn_remap_key) / sizeof(special_ke
    y), fn_remap_event, 1)) {
7.              return;
8.          }
9.
10.         if (special_key_deal(map, other_key, sizeof(other_key) / sizeof(special_key), ot
    her_key_map, 0)) {
11.             return;
12.         }
13.
14.         for (col = 0; col < COL_MAX; col++) {
15.             for (row = 0; row < ROW_MAX; row++) {
16.                 if (map[col] & BIT(row)) {
17.                     full_key_array(row, col, MATRIX_KEY_SHORT);
18.                 } else {
19.                     full_key_array(row, col, MATRIX_KEY_UP);
```

```
                }
            }
        }
        Phantomkey_process();
        send_matrix_key_report(key_status_array);
}
```

```
void touch_pad_event_deal(struct sys_event *event)
{
    u8 mouse_report[8] = {0};
    if ((event->u).touchpad.gesture_event) {
        //g_printf("touchpad gesture_event:0x%x\n", (event->u).touchpad.gesture_even
t);
        switch ((event->u).touchpad.gesture_event) {
        case 0x1:
            mouse_report[0] |= _KEY_MOD_LMETA;
            mouse_report[2] = _KEY_EQUAL;
            hid_report_send(KEYBOARD_REPORT_ID, mouse_report, 8);
            memset(mouse_report, 0x0, 8);
            hid_report_send(KEYBOARD_REPORT_ID, mouse_report, 8);
            return;
        case 0x2:
            mouse_report[0] |= _KEY_MOD_LMETA;
            mouse_report[2] = _KEY_MINUS;
            hid_report_send(KEYBOARD_REPORT_ID, mouse_report, 8);
            memset(mouse_report, 0x0, 8);
            hid_report_send(KEYBOARD_REPORT_ID, mouse_report, 8);
            return;
        case 0x3:
            mouse_report[0] |= BIT(0);                    //left mouse button
            break;
        case 0x4:
```

```
                mouse_report[0] |= BIT(1);
                break;
            }
        }
        if ((event->u).touchpad.x || (event->u).touchpad.y) {
            mouse_report[1] = gradient_acceleration((event->u).touchpad.x);
            mouse_report[2] = gradient_acceleration((event->u).touchpad.y);
        }
        hid_report_send(MOUSE_POINT_REPORT_ID, mouse_report, 3);
}
```

3. Data transmission

KEYBOARD belongs to the category of HID devices. The definition and transmission of data should be determined according to the content of the HID device descriptor.

As can be seen from the descriptor in the following figure, the descriptor is a user-defined descriptor, which is composed of KeyBoard, Consumer Control and Mouse group.

The Keyboard mainly realizes the function of ordinary keys, Consumer Control realizes the multimedia system control, and Mouse realizes the touchpad function.

```
    0x05, 0x01,             // Usage Page (Generic Desktop Ctrls)
    0x09, 0x06,             // Usage (Keyboard)
    0xA1, 0x01,              // Collection (Application)
    0x85, KEYBOARD_REPORT_ID,// Report ID (1)
    0x05, 0x07,             // Usage Page (Kbrd/Keypad)
    0x19, 0xE0,              // Usage Minimum (0xE0)
    0x29, 0xE7,              // Usage Maximum (0xE7)
    0x15, 0x00,             // Logical Minimum (0)
    0x25, 0x01,             // Logical Maximum (1)
    0x75, 0x01,             // Report Size (1)
    0x95, 0x08,             // Report Count (8)
    0x81, 0x02,             // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
    0x95, 0x01,             // Report Count (1)
    0x75, 0x08,             // Report Size (8)
    0x81, 0x01,             // Input (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
    0x95, 0x03,             // Report Count (3)
```

```
    0x75, 0x01,             // Report Size (1)
    0x05, 0x08,             // Usage Page (LEDs)
    0x19, 0x01,             // Usage Minimum (Num Lock)
    0x29, 0x03,             // Usage Maximum (Scroll Lock)
    0x91, 0x02,             // Output (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position,Non-volatile)
    0x95, 0x05,             // Report Count (5)
    0x75, 0x01,             // Report Size (1)
    0x91, 0x01,             // Output (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null Position,Non-volatile)
    0x95, 0x06,             // Report Count (6)
    0x75, 0x08,             // Report Size (8)
    0x15, 0x00,             // Logical Minimum (0)
    0x26, 0xFF, 0x00, // Logical Maximum (255)
    0x05, 0x07,             // Usage Page (Kbrd/Keypad)
    0x19, 0x00,             // Usage Minimum (0x00)
    0x2A, 0xFF, 0x00, // Usage Maximum (0xFF)
    0x81, 0x00,             // Input (Data,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
    0xC0,                   // End Collection
    0x05, 0x0C,             // Usage Page (Consumer)
    0x09, 0x01,             // Usage (Consumer Control)
    0xA1, 0x01,             // Collection (Application)
    0x85, COUSTOM_CONTROL_REPORT_ID,// Report ID (3)
    0x75, 0x10,             // Report Size (16)
    0x95, 0x01,             // Report Count (1)
    0x15, 0x00,             // Logical Minimum (0)
    0x26, 0x8C, 0x02, // Logical Maximum (652)
    0x19, 0x00,             // Usage Minimum (Unassigned)
    0x2A, 0x8C, 0x02, // Usage Maximum (AC Send)
    0x81, 0x00,             // Input (Data,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
    0xC0,                   // End Collection
//
    // Dummy mouse collection starts here
    //
```

```
0x05, 0x01,                 // USAGE_PAGE (Generic Desktop)
0x09, 0x02,                 // USAGE (Mouse)
0xa1, 0x01,                 // COLLECTION (Application)
0x85, MOUSE_POINT_REPORT_ID,            // REPORT_ID (Mouse)
0x09, 0x01,                 // USAGE (Pointer)
0xa1, 0x00,                 // COLLECTION (Physical)
0x05, 0x09,                 //      USAGE_PAGE (Button)
0x19, 0x01,                 // USAGE_MINIMUM (Button 1)
0x29, 0x02,                 // USAGE_MAXIMUM (Button 2)
0x15, 0x00,                 // LOGICAL_MINIMUM (0)
0x25, 0x01,                 // LOGICAL_MAXIMUM (1)
0x75, 0x01,                 //      REPORT_SIZE (1)
0x95, 0x02,                 // REPORT_COUNT (2)
0x81, 0x02,                 //      INPUT (Data,Var,Abs)
0x95, 0x06,                 // REPORT_COUNT (6)
0x81, 0x03,                 //      INPUT (Cnst,Var,Abs)
0x05, 0x01,                 //      USAGE_PAGE (Generic Desktop)
0x09, 0x30,                 // USAGE (X)
0x09, 0x31,                 // USAGE (Y)
0x15, 0x81,                 // LOGICAL_MINIMUM (-127)
0x25, 0x7f,                 // LOGICAL_MAXIMUM (127)
0x75, 0x08,                 //      REPORT_SIZE (8)
0x95, 0x02,                 // REPORT_COUNT (2)
0x81, 0x06,                 //      INPUT (Data,Var,Rel)
0xc0,                       // END_COLLECTION
0xc0                        // END_COLLECTION
```

HID data sending interface, send HID report according to the current connection mode.

```
1. void hid_report_send(u8 report_id, u8 *data, u16 len)
```

```
2. {
3.          if (bt_hid_mode == HID_MODE_EDR) {
4. #if TCFG_USER_EDR_ENABLE
5.              edr_hid_data_send(report_id, data, len);
6. #endif
7.          } else {
8. #if TCFG_USER_BLE_ENABLE
9.              ble_hid_data_send(report_id, data, len);
10. #endif
11.         }
12. }
```

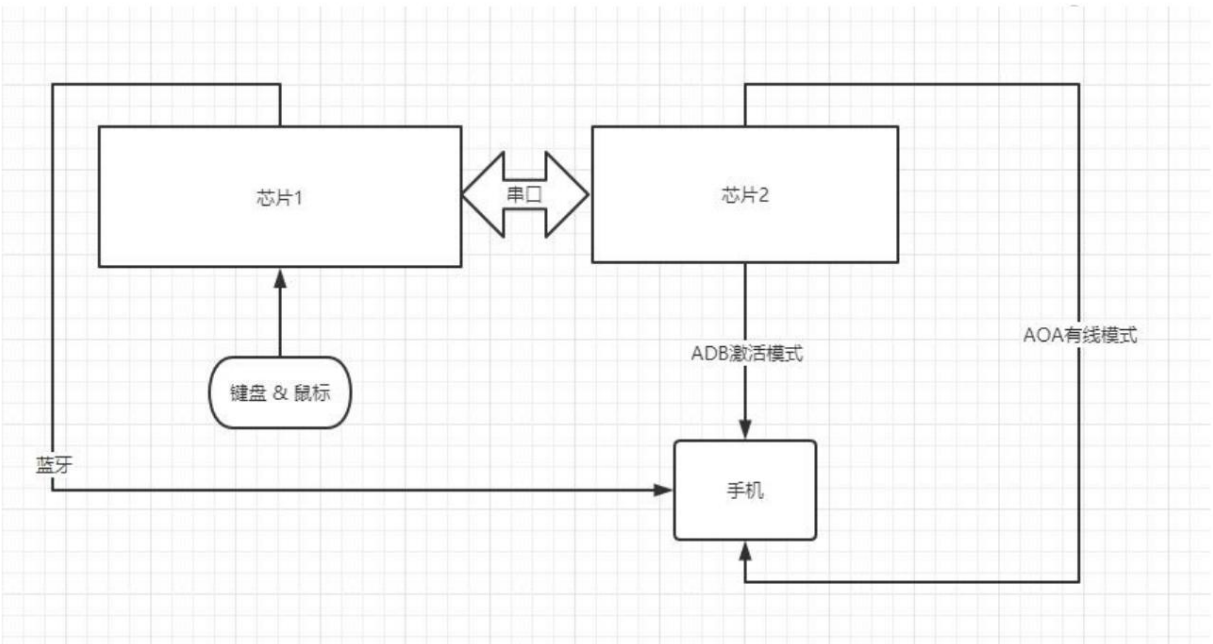
2.11 APP - Gamebox eat chicken throne mode

2.11.1 Overview

This case is mainly used for the realization of eating chicken throne, open the mobile phone bluetooth to connect the device, and connect the keyboard and mouse to usb 0 and usb1 respectively. pieces

After the lift is successful, the light of the mouse and keyboard will light up. You can also use the AOA wired mode to connect to the Android phone. In this case, two chips are required, and one dual chip is used.

The usb chip parses the keyboard and mouse, and sends the keyboard and mouse data to another chip through the serial port. The system block diagram is as follows



If you don't need wired mode, you can omit chip 2 and add an analog switch to activate the mobile phone of the MTK platform.

Eat Chicken Throne SDK. Peace Elite game that supports default layout.

The keys are mapped as follows:

F1 burst mode switch

F2 pressure gun switch

F3 to speed up shooting

F4 reduces the firing speed

F5 increases the pressure of the gun

F6 to reduce pressure gun power

Support board level: bd19, bd29, br23

Support chip: AC6311, AC6321A

2.11.2 Project Configuration

Code project: apps\hid\board\bd19\AC632N_hid.cbp

(1) Configure the app Select (apps\hid\include\app_config.h), select the corresponding standard keyboard application as shown below.

```
3.  //app case selection, only choose 1, to configure the corresponding board_config.h
4. #define CONFIG_APP_KEYBOARD                0//hid button, default case
5. #define CONFIG_APP_KEYFOB                  0//Selfie, board_ac6368a,board_6318
6. #define CONFIG_APP_MOUSE                   0//mouse, board_mouse
7. #define CONFIG_APP_STANDARD_KEYBOARD 0//Standard HID keyboard,board_ac6351d
8. #define CONFIG_APP_KEYPAGE                 0//pager
9. #define CONFIG_APP_GAMEBOX                 1//Chicken Throne
```

(2) First configure the board-level board_config.h (apps\hid\board\bd19\board_config.h), and select the corresponding development board.

```
12. #define CONFIG_BOARD_AC635N_DEMO
13. //#define CONFIG_BOARD_AC6351D_KEYBOARD
14.
```

```
15. #include "board_ac635n_demo_cfg.h"
16. #include "board_ac6351d_keyboard_cfg.h"
```

(3) Function configuration (board_ac632n_demo_cfg.h)

```
10. //*********************************************************************//
11. USB Configuration
12. #define TCFG_PC_ENABLE                  DISABLE_THIS_MOUDLE //PC module enable
13. #define TCFG_UDISK_ENABLE               DISABLE_THIS_MOUDLE //U disk module enable
14. #define TCFG_HID_HOST_ENABLE            ENABLE_THIS_MOUDLE //Game box mode
15. #define TCFG_ADB_ENABLE                 ENABLE_THIS_MOUDLE
16. #define TCFG_AOA_ENABLE                 ENABLE_THIS_MOUDLE
```

//Configure to open the ble module

```
15. #define TCFG_USER_BLE_ENABLE            1 //BLE function enable
16. #define TCFG_USER_EDR_ENABLE            0 //EDR function enable
```

2.11.3 Main code description

1. Handling of events

The main event processing in this case includes usb unplugging, MTK ordinary adb activation mode mobile phone unplugging and plugging events. mouse movement click, keyboard keys

2.1 usb event handling

```
1. static void usb_event_handler(struct sys_event *event, void *priv)
2. {
3.     const char *usb_msg;
4.     usb_dev usb_id;
5.
6.     switch ((u32)event->arg) {
7.     case DEVICE_EVENT_FROM_OTG:
8.         usb_msg = (const char *)event->u.dev.value;
9.         usb_id = usb_msg[2] - '0';
```

```
10.
11.             log_debug("usb event : %d DEVICE_EVENT_FROM_OTG %s",
12.                       event->u.dev.event, usb_msg);
13.             if (usb_msg[0] == 'h') {
14.                 if (event->u.dev.event == DEVICE_EVENT_IN) {
15.                     log_info("usb %c online", usb_msg[2]);
16.                     if (usb_host_mount(usb_id, 3, 20, 250)) {
17.                         usb_h_force_reset(usb_id);
18.                         usb_otg_suspend(usb_id, OTG_UNINSTALL);
19.                         usb_otg_resume(usb_id);
20.                     }
21.                 } else if (event->u.dev.event == DEVICE_EVENT_OUT) {
22.                     log_info("usb %c offline", usb_msg[2]);
23.                     set_phone_connect_status(0);
24.                     usb_host_unmount(usb_id);
25.                 }
26.             } else if (usb_msg[0] == 's') {
27. #if TCFG_PC_ENABLE
28.                 if (event->u.dev.event == DEVICE_EVENT_IN) {
29.                     usb_start(usb_id);
30.                 } else {
31.                     usb_stop(usb_id);
32.                 }
33. #endif
34.             }
35.             break;
36.         case DEVICE_EVENT_FROM_USB_HOST:
37.             log_debug("host_event %x", event->u.dev.event);
38.             if ((event->u.dev.event == DEVICE_EVENT_IN) ||
39.                 (event->u.dev.event == DEVICE_EVENT_CHANGE)) {
40.                     int err = os_taskq_post_msg(TASK_NAME, 2, DEVICE_EVENT_IN,
    event->u.dev.value);
```

```
41.                         if (err) {
42.                             r_printf("err %x ", err);
43.                         }
44.                     } else if (event->u.dev.event == DEVICE_EVENT_OUT) {
45.                         log_error("device out %x", event->u.dev.value);
46.                     }
47.                     break;
48.             }
49. }
```

The following mouse handler functions,

```
1. void mouse_route(const struct mouse_data_t *p)
2. {
3.     if (get_run_mode() != UART_MODE) {
4.         if (mouse_filter((void *)p) == 0) {
5.             return;
6.         }
7.     }
8.     /* log_info("btn: %x x-y %d %d wheel %d ac_pan %d", */
9.     /*             p-> btn, p-> x, p-> y, p-> wheel, p-> ac_pan); * /
10.    switch (get_run_mode()) {
11.    case UART_MODE :// in USB interrupt function call
12.        send2uart(MOUSE_POINT_MODE, p);
13.        break;
14.    case BT_MODE ://Interrupt function in uart or usb
15.    case USB_MODE:// is called on the serial port interrupt
16.        if (is_mouse_point_mode) {
17.            send2phone(MOUSE_POINT_MODE, p);
18.        } else {
19.            mouse_mapping(p);
20.            send2phone(TOUCH_SCREEN_MODE, p);
21.        }
```

```
22.             break;
23.         case MAPPING_MODE:
24.             send2phone(MOUSE_POINT_MODE + 1, p);
25.             break;
26.         default :
27.             log_info("btn: %x x-y %d %d wheel %d ac_pan %d",
28.                         p-> btn, p-> x, p-> y, p-> wheel, p-> ac_pan);
29.             break;
30.     }
31. }
```

Handle middle key messages, pointer mode, touch mode switching.

2.2 Keyboard data processing

```
1. void keyboard_route(const u8 *p)
2. {
3.     /* log_info("keyboard:"); */
4.     /* printf_buf(p, 8); */
5.     if (keyboard_filter((struct keyboard_data_t *)p) == 0) {
6.         return;
7.     }
8.
9.     switch (get_run_mode()) {
10.     case UART_MODE :// in USB interrupt function call
11.         send2uart(KEYBOARD_MODE, p);
12.         break;
13.     case BT_MODE ://Receive event in uart or call usb interrupt function
14.     case USB_MODE:// is called on the serial port event
15.         key_mapping((const void *)p);
16.         send2phone(TOUCH_SCREEN_MODE, p);
17.         break;
18.     case MAPPING_MODE:
```

```
19.             send2phone(KEYBOARD_MODE, p);
20.             break;
21.         default :
22.             printf_buf((u8 *)p, 8);
23.             break;
24.         }
25. }
```

3. **Adb** activate **MTK** phone

is the script that runs the activation process, active.bash

APP_ACTIVITY_PATH Android activation page

```
1. #define     APP_ACTIVITY_PATH "com.zh-jieli.gmaeCenter/com.zh-jieli.gameCenter.act
   ivity.guide.SplashActivity\n"
2. #define     APP_WEBSITE       "http://www.zh-jieli.com\n"
3. #define     APP_BASH_IN_PATH  "/sdcard/jilei/active.bash"
4. #define     APP_BASH_OUT_PATH "/data/local/tmp/active.bash"
5.
6. u32 adb_game_active()
7. {
8.     log_info("%s() %d\n", __func__, __LINE__);
9.     u32 max_len = adb.max_len;;
10.    u8 *adb_buffer = adb.buffer;
11.    //1, start the app
12.    adb_ex_cmd("am start -n  APP_ACTIVITY_PATH, adb_buffer, max_len);
13.    puts((char *)adb_buffer);
14.    //Look for the Error string, if you find the jump page to download the app, otherwise execute the adb command
15.    if (strstr((const char *)adb_buffer, "Error") != NULL) {
16.        adb_ex_cmd("am start -a android.intent.action.VIEW -d "  APP_WEBSITE, adb_buf
   fer, max_len);
17.        puts((char *)adb_buffer);
18.    } else {
```

```
19.                 adb_ex_cmd("dd if=" APP_BASH_IN_PATH      " of=" APP_BASH_OUT_PATH "\n", adb_buff
    is, max_len);
20.                 puts((char *)adb_buffer);
21.                 adb_ex_cmd("chown shell " APP_BASH_OUT_PATH";chmod 777 "APP_BASH_OUT_PATH "\
    n", adb_buffer, max_len);
22.                 puts((char *)adb_buffer);
23.                 adb_ex_cmd("trap \"\" HUP;sh "APP_BASH_OUT_PATH "&\n", adb_buffer, max_len);

24.                 puts((char *)adb_buffer);
25.         }
26.
27.         return 0;
28. }
```

4. Button mapping

Handle key mappings in this function.

void key_mapping(const struct keyboard_data_t *k)

2.12 APP - IBEACON

2.12.1 Overview

Bluetooth beacon is a Bluetooth beacon, which broadcasts a broadcast packet of a specific data format through BLE, and the receiving terminal obtains BLE by scanning. Broadcast packet information, and then parse it according to the protocol. The communication between the receiving terminal and the bluetooth beacon does not need to establish a bluetooth connection. target Previous applications: 1. Bluetooth beacon indoor positioning, with the advantages of simplicity, low power consumption, and good compatibility with mobile phones. 2. News push, see in the big Marketing activities in shopping malls, etc. 3. Attendance punch card, identification.

Support board level: bd19, br23, br25, bd29, br30

Support chip: AC632N, AC635N, AC636N, AC631N, AC637N

2.12.2 Project Configuration

Code project: apps\spp_and_le\board\bdxx\AC63xN_spp_and_le.cbp

1. Configure the IBEACON application in the app_config. file.

```
29. //app case selection, only one can be selected, the corresponding board_config.h should be configured
30. #define CONFIG_APP_BEACON                    1 //Bluetooth BLE ibeacon
```

2.12.3 Main code description

1. After configuring the application case, according to the defined data format, make the beacon data package, this function is located in the le_beacon.c file.

```
31. static u8 make_beacon_packet(u8 *buf, void *packet, u8 packet_type, u8 *web)
32. {
33.     switch (packet_type) {
34.     case IBEACON_PACKET:
35.     case EDDYSTONE_UID_PACKET:
36.     case EDDYSTONE_TLM_PACKET:
37.         memcpy(buf, (u8 *)packet, packet_type);
38.         break;
39.     case EDDYSTONE_EID_PACKET:
40.         memcpy(buf, (u8 *)packet, packet_type);
41.         break;
42.     case EDDYSTONE_ETLM_PACKET:
```

```
43.             memcpy(buf, (u8 *)packet, packet_type);
44.             break;
45.         case EDDYSTONE_URL_PACKET:
46.             packet_type = make_eddystone_url_adv_packet(buf, packet, web);
47.             break;
48.         }
49.         return packet_type;
50. }
```

2. For example, the following is the data format of eddystone_etlm. After making the data into a data packet, use make_set_adv_data().

According to the sent, this function is also located in the le_beacon.c file, and the broadcast type of the beacon should be broadcast disconnected

ADV_NONCONN_IND type.

```
51. static const EDDYSTONE_ETLM eddystone_etlm_adv_packet = {
52.         .length = 0x03,
53.         .ad_type1 = 0x03,
54.         .complete_list_uuid = 0xabcd,
55.         .length_last = 0x15,
56.         .ad_type2 = 0x16,
57.         .eddystone_uuid = 0xfeaa,
58.         .frametype = 0x20,
59.         .tlm_version = 0x01ÿ
60.         .etml = {
61.             0, 0, 0, 0,
62.             0, 0, 0, 0,
63.             0, 0, 0, 0
64.         },                  //12 bytes encrypted data
65.         .random = 1,                        //random number, the same as the random number used in encryption
66.         .check = 2ÿ                         //Checksum calculated by AES-EAX
67. };
```

3. The data is sent in the form of broadcast packets, and the client obtains the data sent by the beacon through scanning.

```
68. static int make_set_adv_data(void)
```



```
69. {
70.         u8 offset = 0;
71.         u8 *buf = adv_data;
72. /* offset += make_eir_packet_val(&buf[offset], offset, HCI_EIR_DATATYPE_FLAGS, 0x06, 1);
    */offset+=make_beacon_packet(&buf[offset],&eddystone_etlm_adv_packet,EDDYSTONE_ETLM_PACKET,
    NULL);
73. offset+=make_beacon_packet(&buf[offset],&eddystone_eid_adv_packet,EDDYSTONE_EID_PACKET,
    NULL); */
74.         //offset+=make_beacon_packet(&buf[offset],&eddystone_url_adv_packet,EDDYSTONE_URL_PACKET,
    "https://fanyi.baidu.com/");
75.         //offset+=make_beacon_packet(&buf[offset],&eddystone_tlm_adv_packet,EDDYSTONE_TLM_PACKET,
    NULL);
76.         //offset+=make_beacon_packet(&buf[offset],&eddystone_uid_adv_packet,EDDYSTONE_UID_PACKET,
    NULL);
77. // offset += make_beacon_packet(&buf[offset], &ibeacon_adv_packet, IBEACON_PACKET, NULL);
78.         if (offset > ADV_RSP_PACKET_MAX) {
79.             puts("***adv_data overflow!!!!!\n");
80.             return -1;
81.         }
```

2.13 APP - Bluetooth Multi connections

2.13.1 Overview

Support Bluetooth dual-mode transparent transmission and data transmission, and support Bluetooth LE multi-connection function. CLASSIC Bluetooth uses standard serial port SPP profile

Protocol that supports discoverable search connections. Bluetooth LE currently supports applications such as GAP 1 master 1 slave, or 2 master roles.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

Note that there are differences in the amount of RAM available for different chips, which may affect performance.

2.13.2 Project Configuration

Code project: apps\spp_and_le\board\br30\AC637N_spp_and_le.cbp

(1) app configuration

Find the corresponding file (apps\hid\include\app_config.h) in the project code to select the APP. In this case, select the pager.

The result is shown below:

```
1. //app case        choice only    1  to configure the corresponding  board_config.h
2. #define CONFIG_APP_MULTI                    0 //Bluetooth LE multi-connect + spp
```

Configure the number of multiple machines and encryption

```
1. //      Bluetooth BECAME configure
2. #define CONFIG_BT_GATT_COMMON_ENABLE        1 //   Configure using gatt    public module
3. #define CONFIG_BT_SM_SUPPORT_ENABLE         0 //   Configure whether to support encryption
4. #define CONFIG_BT_GATT_CLIENT_NUM           1 //   Configure the host client    indivual
      number  (app not support)
5. #define CONFIG_BT_GATT_SERVER_NUM           1 //   Configure the slave server    number
6. #define CONFIG_BT_GATT_CONNECTION_NUM       (CONFIG_BT_GATT_SERVER_NUM +
      CONFIG_BT_GATT_CLIENT_NUM) //    Configure the number of connections
```

(2) Board level selection

Then in the file (apps\hid\board\bd30\board_config.h), make the corresponding board-level selection as follows:


```
/*
 * Board-level configuration options
 */

#define CONFIG_BOARD_AC637N_DEMO
// #define CONFIG_BOARD_AC6373B_DEMO
// #define CONFIG_BOARD_AC6376F_DEMO
// #define CONFIG_BOARD_AC6379B_DEMO

#include "board_ac637n_demo_cfg.h"
#include "board_ac6373b_demo_cfg.h"
#include "board_ac6376f_demo_cfg.h"
#include "board_ac6379b_demo_cfg.h"
#endif
```

//Corresponding board-level header file, configure whether to open the edr and ble modules

```
#define TCFG_USER_BLE_ENABLE        1 //BLE function enable
#define TCFG_USER_EDR_ENABLE        0 //EDR function enable
```

2.13.3 Main code description

1. The main task processes the file apps/hid/app_multi_conn.c

(1) APP registration processing (the function is located in apps/hid/app_multi_conn.c)

During system initialization, APP registration is performed according to the following information. The general process of execution is:

REGISTER_APPLICATION--->state_machine--->app_start()--->sys_key_event_enable();This process is mainly carried out

The initial settings of the device and some functions are enabled.

REGISTER_APPLICATION--->event_handler--->app_key_event_handler()--->app_key_deal_test();This process

There are multiple cases under event_handler. The above-mentioned processing flow for selecting key events is described in the code flow, mainly showing key events.

When the event occurs, the processing flow of the program.

```
static const struct application_operation app_multi_ops = {
    .state_machine = state_machine,
    .event_handler = event_handler,
```

```
4. };
5.
6. /*
7.      * Register AT Module mode
8.      */
9. REGISTER_APPLICATION(app_multi) = {
10.         .name = "multi_conn",
11.         .action = ACTION_MULTI_MAIN,
12.         .ops = &app_multi_ops,
13.         .state = APP_STA_DESTROY,
14. };
15.
16. //-----------------------
17. //system check go sleep is ok
18. static u8 app_state_idle_query(void)
19. {
20.         return !is_app_active;
21. }
22.
23. REGISTER_LP_TARGET(app_state_lp_target) = {
24.         .name = "app_state_deal",
25.         .is_idle = app_state_idle_query,
26. };
```

(2) APP state machine

The state machine has create, start, pause, resume, stop, destroy states, and executes corresponding branches according to different states.

After the APP is registered, perform the initial operation, enter the APP_STA_START branch, and start the APP operation.

```
1.      static int state_machine(struct application *app, enum app_state state, struct intent *it)
2. { switch (state) {
3.      case APP_STA_CREATE:
4.          break;
5.      case APP_STA_START:
```

```
6.          if (!it) {
7.              break;          }
8.          switch (it->action) {
9.          case ACTION_MULTI_MAIN:
10.             app_start();
```

After entering the app_start() function, perform the corresponding initialization, clock initialization, mode selection, Bluetooth initialization, low-power initialization, and

External event enable.

```
1.  static void app_start()
2. {
3.      log_info("======================================");
4.      log_info("----------------------multi_conn demo----------------------");
5.      log_info("======================================");
6.
```

(3) APP event processing mechanism

1. The definition of the event (the code is located in Headers\include_lib\system\even.h)

```
1. struct sys_event {
2.      u16 type;
3.      u8 consumed;
4.      void *arg;
5.      union {
6.          struct key_event key;
7.          struct axis_event axis;
8.          struct codesw_event codesw;
```

(4) Event generation (include_lib\system\event.h)

```
void sys_event_notify(struct sys_event *e);
```

Event notification function, the system calls this function when an event occurs.

(5) Event processing (app_mutil.c)

The general flow of function execution is: evevt_handler()--->app_key_event_handler()--->app_key_deal_test().

```
1.  static int event_handler(struct application *app, struct sys_event *event)
2.
```

```
1.  static void app_key_event_handler(struct sys_event *event)
```

```
1.  static void app_key_deal_test(u8 key_type, u8 key_value)
```

2. LE public processing ble_multi_conn.c

Configure Gatt common module initialization, Bluetooth initialization and other operations.

3. LE 's **GATT server** implements ble_multi_peripheral.c

Configure the broadcast, connection, event processing, etc. on the Gatt Server side.

4. LE 's **GATT client** implements ble_multi_central.c

Configure search, connection, event handling, etc. on the Gatt Cleint side.

2.14 APP - Bluetooth Dual-Mode AT Moudle (char)

2.14.1 Overview

The main function is that on the basis of ordinary data transmission BLE and EDR, the host computer or other MCU can communicate with each other through UART. Connect to the Bluetooth chip for basic configuration, status acquisition, control scanning, disconnection, and data transmission and reception.

AT control transparent transmission master-slave multi-machine mode.

Define a set of serial port control protocol, please refer to the protocol document "Bluetooth AT_CHAR Protocol" for details.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

Note that there are differences in the amount of RAM available for different chips, which may affect performance.

2.14.2 Project Configuration

Code project: apps\spp_and_le\board\bd29\AC631N_spp_and_le.cbp

(1) First configure the board-level board_config.h and the Bluetooth dual-mode enable in the corresponding configuration file

```
1. /*
2.      * Board-level configuration options
3.      */
4. #define CONFIG_BOARD_AC631N_DEMO                // CONFIG_APP_KEYBOARD,CONFIG_APP_PAGE_TURNER
5.
6. #include "board_ac631n_demo_cfg.h"
```

(2) Configure the corresponding board_acxxx_demo_cfg.h file to enable BLE. Take board_ac630x_demo_cfg.h as an example

```
19. #define TCFG_USER_BLE_ENABLE                    1 //BLE function enable
20. #define TCFG_USER_EDR_ENABLE                    0 //EDR function enable
```

(3) Configure app_config.h, select CONFIG_APP_AT_CHAR_COM

```
28. #define CONFIG_APP_AT_CHAR_COM                  1//AT com string format command
```

2.14.3 Main description code <at_char_cmds.c>

1. Command header

```
29. static const char at_head_at_cmd[]        = "AT+";
30. static const char at_head_at_chl[]        = "AT>";
31.
32. static const str_info_t at_head_str_table[] = {
33.         INPUT_STR_INFO(STR_ID_HEAD_AT_CMD, at_head_at_cmd),
34.         INPUT_STR_INFO(STR_ID_HEAD_AT_CHL, at_head_at_chl),
35. };
36.
```

2. Command type:

```
37. static const char at_str_gver []           = "GOVERN";
38. static const char at_str_gcfgver[]         = "GCFGVER";
39. static const char at_str_name[]            = "NAME";
40. static const char at_str_lbdaddr[]         = "LBDADDR";
41. static const char at_str_baud[]            = "BAUD";
42.
43. static const char at_str_adv[]             = "ADV";
44. static const char at_str_advparam[]        = "ADVPARAM";
45. static const char at_str_advdata[]         = "ADVDATA";
46. static const char at_str_srdata[]          = "SRDATA";
47. static const char at_str_connparam[] = "CONNPARAM";
48.
49. static const char at_str_scan[]            = "SCAN";
50. static const char at_str_targetuuid[] = "TARGETUUID";
51. static const char at_str_conn[]            = "CONN";
52. static const char at_str_disc[]            = "DISC";
53. static const char at_str_ota[]             = "OTA";
54.
55. static const str_info_t at_cmd_str_table[] = {
56.         INPUT_STR_INFO (STR_ID_GVER, at_str_gver),
57.         INPUT_STR_INFO(STR_ID_GCFGVER, at_str_gcfgver),
58.         INPUT_STR_INFO(STR_ID_NAME, at_str_name),
```

```
59.     INPUT_STR_INFO(STR_ID_LBDADDR, at_str_lbdaddr),
60.     INPUT_STR_INFO(STR_ID_BAUD, at_str_baud),
61.
62.     INPUT_STR_INFO(STR_ID_ADV, at_str_adv),
63.     INPUT_STR_INFO(STR_ID_ADVPARAM, at_str_advparam),
64.     INPUT_STR_INFO(STR_ID_ADVDATA, at_str_advdata),
65.     INPUT_STR_INFO(STR_ID_SRDATA, at_str_srdata),
66.     INPUT_STR_INFO(STR_ID_CONNPARAM, at_str_connparam),
67.
68.     INPUT_STR_INFO(STR_ID_SCAN, at_str_scan),
69.     INPUT_STR_INFO(STR_ID_TARGETUUID, at_str_targetuuid),
70.     INPUT_STR_INFO(STR_ID_CONN, at_str_conn),
71.     INPUT_STR_INFO(STR_ID_DISC, at_str_disc),
72.     INPUT_STR_INFO(STR_ID_OTA, at_str_ota),
73.
74. // INPUT_STR_INFO(, ),
75. // INPUT_STR_INFO(, ),
76. };
```

3. Serial port data reception is interrupted,

A preliminary check will be performed before entering the data receiving interrupt. If the end of the data is not '\r', the interrupt cannot be woken up.

```
77. static void at_packet_handler(u8 *packet, int size)
```

4. AT command parsing

```
78.             /* Compare packet headers */
79.         str_p=at_check_match_string(parse_pt,
    parse_size,at_head_str_table,sizeof(at_head_str_table));
80.         if (!str_p)
81.         {
82.             log_info("###1unknow at_head:%s", packet);
83.             AT_STRING_SEND(at_str_err);
```

```
84.                 return;
85.             }
86.             parse_pt += str_p->str_len;
87.             parse_size -= str_p->str_len;
88.
89. /*Ordinary commands*/
90.             if(str_p->str_id == STR_ID_HEAD_AT_CMD)
91.             {
92.                 /*Compare command*/
93.                 str_p=at_check_match_string(parse_pt,
    parse_size,at_cmd_str_table,sizeof(at_cmd_str_table));
94.                 if (!str_p)
95.                 {
96.                     log_info("###2unknow at_cmd:%s", packet);
97.                     AT_STRING_SEND(at_str_err);
98.                     return;
99.                 }
100.
101.                parse_pt += str_p->str_len;
102.                parse_size -= str_p->str_len;
103.                /* Determine the current command type, query or set command */
104.                if(parse_pt[0] == '=')
105.                {
106.                    operator_type = AT_CMD_OPT_SET;
107.                }
108.                else if(parse_pt[0] == '?')
109.                {
110.                    operator_type = AT_CMD_OPT_GET;
111.                }
112.                parse_pt++;
113.            }
114.
```

```
115. /*Channel switching command*/
116.         else if(str_p->str_id == STR_ID_HEAD_AT_CHL)
117.         {
118.             operator_type = AT_CMD_OPT_SET;
119.         }
120.
121. //        if(operator_type == AT_CMD_OPT_NULL)
122. //        {
123. //            AT_STRING_SEND(at_str_err);
124. //            log_info("###3unknow operator_type:%s", packet);
125. //            return;
126. //        }
127.
128.
129.         log_info("str_id:%d", str_p->str_id);
130.
131. /*Parse and return the command parameter par*/
132.         par = parse_param_split(parse_pt,',','\r');
133.
134.         log_info("\n par->data: %s",par->data);
135.
136. /*Command processing and response*/
137.         switch (str_p->str_id)
138.         {
139.             case STR_ID_HEAD_AT_CHL:
140.                 log_info("STR_ID_HEAD_AT_CHL\n");
141.             break;
142.         .....
```

5. Command parameter acquisition and traversal

```
143. par = parse_param_split(parse_pt,',','\r');
144. /*parameter
```

```
145. *packet: parameter pointer
146. split_char: The separator between parameters, usually ','
147. end_char: The end character of the parameter, usually '\r'
148. */
149. static at_param_t *parse_param_split(const u8 *packet,u8 split_char,u8 end_char)
150. {
151.     u8 char1;
152.     int i = 0;
153.     at_param_t *par = parse_buffer;
154.
155.     if (*packet == end_char) {
156.         return NULL;
157.     }
158.
159.     log_info("%s:%s",__FUNCTION__,packet);
160.
161.     par-> len = 0;
162.
163.     while (1) {
164.         char1 = packet[i++];
165.         if (char1 == end_char) {
166.             par-> data [par-> len] = 0;
167.             par->next_offset = 0;
168.             break;
169.         } else if (char1 == split_char) {
170.             par-> data [par-> len] = 0;
171.             par-> len ++;
172.             par->next_offset = &par->data[par->len] - parse_buffer;
173.
174.             //init next par
175.             par = &par->data[par->len];
176.             par-> len = 0;
```

```
177.                } else {
178.                    par->data[par->len++] = char1;
179.                }
180.
181.                if (&par->data[par->len] - parse_buffer >= PARSE_BUFFER_SIZE) {
182.                    log_error("parse_buffer over");
183.                    par->next_offset = 0;
184.                    break;
185.                }
186.            }
187.            return (void *)parse_buffer;
188. }
```

When there are multiple parameters, it needs to be traversed and obtained, taking connection parameters as an example

```
189.            while (par) { // loop through all parameters
190.                log_info ("len =% d, par:% s", par-> len, par-> data);
191.                conn_param[i] = func_char_to_dec(par->data, '\0'); // Traverse to get connection parameters
192.                if (par->next_offset) {
193.                    par = AT_PARAM_NEXT_P(par);
194.                } else {
195.                    break;
196.                }
197.                i++;
198.            }
```

6. Default information configuration

```
199. #define G_VERSION "BR30_2021_01_31"
200. #define CONFIG_VERSION "2021_02_04" /*version information*/
201. u32 uart_baud = 115200; /*Default baud rate*/
202. char dev_name_default[] = "jl_test"; /*default dev name*/
```

7. Add a new command to the code (take query and set baud rate as an example)

(1) Add baud rate enumeration member

```
203. enum {
204.     STR_ID_NULL = 0,
205.     STR_ID_HEAD_AT_CMD,
206.     STR_ID_HEAD_AT_CHL,
207.
208.     STR_ID_OK = 0x10,
209.     STR_ID_ERROR,
210.
211.     STR_ID_GVER = 0x20,
212.     STR_ID_GCFGVER,
213.     STR_ID_NAME,
214.     STR_ID_LBDADDR,
215.     STR_ID_BAUD, /*Baud rate member*/
216.
217.     STR_ID_ADV,
218.     STR_ID_ADVPARAM,
219.     STR_ID_ADVDATA,
220.     STR_ID_SRDATA,
221.     STR_ID_CONNPARAM,
222.
223.     STR_ID_SCAN,
224.     STR_ID_TARGETUUID,
225.     STR_ID_CONN,
226.     STR_ID_DISC,
227.     STR_ID_OTA,
228. //    STR_ID_,
229. //    STR_ID_,
230. };
```

(2) Add baud rate command type

```
231. static const char at_str_gver []              = "GOVERN";
232. static const char at_str_gcfgver[]           = "GCFGVER";
233. static const char at_str_name[]               = "NAME";
234. static const char at_str_lbdaddr[]           = "LBDADDR";
235. static const char at_str_baud[]               = "BAUD"; /*Baud rate command type*/
236.
237. static const char at_str_adv[]                = "ADV";
238. static const char at_str_advparam[]          = "ADVPARAM";
239. static const char at_str_advdata[]           = "ADVDATA";
240. static const char at_str_srdata[]            = "SRDATA";
241. static const char at_str_connparam[] = "CONNPARAM";
242.
243. static const char at_str_scan[]               = "SCAN";
244. static const char at_str_targetuuid[] = "TARGETUUID";
245. static const char at_str_conn[]               = "CONN";
246. static const char at_str_disc[]               = "DISC";
247. static const char at_str_ota[]                = "OTA";
```

(3) Add command to command list

```
248. static const str_info_t at_cmd_str_table[] = {
249.         INPUT_STR_INFO (STR_ID_GVER, at_str_gver),
250.         INPUT_STR_INFO(STR_ID_GCFGVER, at_str_gcfgver),
251.         INPUT_STR_INFO(STR_ID_NAME, at_str_name),
252.         INPUT_STR_INFO(STR_ID_LBDADDR, at_str_lbdaddr),
253.         INPUT_STR_INFO(STR_ID_BAUD, at_str_baud), /*Baud rate command*/
254.
255.         INPUT_STR_INFO(STR_ID_ADV, at_str_adv),
256.         INPUT_STR_INFO(STR_ID_ADVPARAM, at_str_advparam),
257.         INPUT_STR_INFO(STR_ID_ADVDATA, at_str_advdata),
258.         INPUT_STR_INFO(STR_ID_SRDATA, at_str_srdata),
259.         INPUT_STR_INFO(STR_ID_CONNPARAM, at_str_connparam),
260.
```

```
261.        INPUT_STR_INFO(STR_ID_SCAN, at_str_scan),
262.        INPUT_STR_INFO(STR_ID_TARGETUUID, at_str_targetuid),
263.        INPUT_STR_INFO(STR_ID_CONN, at_str_conn),
264.        INPUT_STR_INFO(STR_ID_DISC, at_str_disc),
265.        INPUT_STR_INFO(STR_ID_OTA, at_str_ota),
266.
267. // INPUT_STR_INFO(, ),
268. // INPUT_STR_INFO(, ),
269. };
```

(4) Add command processing and response in the at_packet_handler function

```
270.            case STR_ID_BAUD:
271.                log_info("STR_ID_BAUD\n");
272.                {
273.                    if(operator_type == AT_CMD_OPT_SET) //Set the baud rate
274.                    {
275.                        uart_baud = func_char_to_dec(par->data, '\0');
276.                        if(uart_baud==9600||uart_baud==19200||uart_baud==38400||uart_baud==115200||
277.                            uart_baud==230400||uart_baud==460800||uart_baud==921600)
278.                        {
279.                            AT_STRING_SEND("OK"); /*return response*/
280.                            ct_uart_init(uart_baud);
281.                        }
282.                        else{ //TODO returns error code
283.
284.
285.                        }
286.                    }
287.                    else{                                   //read baud rate
288.
289.                        sprintf( buf, "+BAUD:%d", uart_baud);
290.                        at_cmd_send(buf, strlen(buf)); /*Return baud rate data*/
```

```
291.                                AT_STRING_SEND("OK"); /*return response*/
292.                            }
293.                        }
294.                    break;
```

8. Serial port data api, used to send response information

```
295. /*
296. parameter
297. packet: data packet
298. size: data length
299. */
300. at_uart_send_packet(const u8 *packet, int size);
```

Used to reply to a response with "\r\n"

```
301. void at_cmd_send(const u8 *packet, int size)
302. {
303.         at_uart_send_packet(at_str_enter, 2);
304.         at_uart_send_packet(packet, size);
305.         at_uart_send_packet(at_str_enter, 2);
306. }
```

2.15 APP - Nonconn_24G

2.15.1 Overview

The main function is to customize the invisible 2.4G non-connection mode data transmission example based on the Bluetooth BLE architecture.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.15.2 Project Configuration

Code project: apps\spp_and_le\board\bdxx\AC63xN_spp_and_le.cbp

(1) Configure app selection (apps\spp_and_le\include\app_config.h), select the corresponding application example as shown below

```
1. //app case          choice only    1  to configure the corresponding  board_config.h
1. #define CONFIG_APP_NONCONN_24G                    1 //2.4G non-connected transceiver
```

(2) First configure the board-level board_config.h (apps\spp_and_le\board\brxx\board_config.h), select the corresponding development board, you can to use the default board level

```
1. #define CONFIG_BOARD_AC632N_DEMO
2. // #define CONFIG_BOARD_AC6321A_DEMO
```

Just enable BLE

```
1. #define TCFG_USER_BLE_ENABLE          1 // BLE      function enable
2. #define TCFG_USER_EDR_ENABLE          0 //EDR      function enable
```

2.15.3 Data transceiver module

The implementation code file is in ble_24g_deal.c

(1) The main configuration macros are as follows:

```
1. //-----------------------------------------------------
2. #define CFG_RF_24G_CODE_ID      0x13 // 24g      Identifier(24bit),   Both sending and receiving must match
3. //-----------------------------------------------------
4. //      Configure Transceiver Roles
5. #define CONFIG_TX_MODE_ENABLE      1 //    launcher
6. #define CONFIG_RX_MODE_ENABLE      0 //    receiver
```

```
7. //-----------------------------------------------------
8. //TX     send configuration
9. #define TX_DATA_COUNT                3 //  The number of timps to send  os_time_dly    how long
10. #define TX_DATA_INTERVAL             20 //  send interval>=20ms
11.
12. #define ADV_INTERVAL_VAL             ADV_SCAN_MS(TX_DATA_INTERVAL)//
13. #define RSP_TX_HEAD                  0xff
14. //-----------------------------------------------------
15. //RX     receive configuration
16. //  search type
17. #define SET_SCAN_TYPE                SCAN_ACTIVE
18. //  search cycle size
19. #define SET_SCAN_INTERVAL ADV_SCAN_MS(200)//
20. //  search window size
21. #define SET_SCAN_WINDOW              ADV_SCAN_MS(200)//
```

(2) Transmitter sending interface

```
1. //  send data ,  len support max is 60
2. int ble_tx_send_data(const u8 *data, u8 len)
```

(3) Receiver receiving interface

```
1. void ble_rx_data_handle(const u8 *data, u8 len)
```

2.16 APP - CONN_24G

2.16.1 Overview

The main function is to customize the invisible 2.4G connection mode data transmission example based on the Bluetooth BLE architecture.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.16.2 Project Configuration

Code project: apps\spp_and_le\board\bdxx\AC63xN_spp_and_le.cbp

(1) Configure app selection (apps\spp_and_le\include\app_config.h), select the corresponding application example as shown below

```
2. //app case          choice only    1  to configure the corresponding  board_config.h
2. #define CONFIG_APP_CONN_24G                    1 //Based on BLE 2.4g, the board only needs to open BLE
```

(2) First configure the board-level board_config.h (apps\spp_and_le\board\brxx\board_config.h), select the corresponding development board, you can to use the default board level

```
1. #define CONFIG_BOARD_AC632N_DEMO
2. // #define CONFIG_BOARD_AC6321A_DEMO
3. // #define CONFIG_BOARD_AC6323A_DEMO
4. // #define CONFIG_BOARD_AC6328A_DEMO
5. // #define CONFIG_BOARD_AC6328B_DONGLE
6. // #define CONFIG_BOARD_AC6329B_DEMO
7. // #define CONFIG_BOARD_AC6329C_DEMO
8. // #define CONFIG_BOARD_AC6329E_DEMO
9. // #define CONFIG_BOARD_AC6329F_DEMO
```

Just enable BLE

```
3. #define TCFG_USER_BLE_ENABLE          1 // BLE     function enable
4. #define TCFG_USER_EDR_ENABLE          0 //EDR     function enable
```

(3) Set 2.4G ID (apps\spp_and_le\examples\conn_24g\app_conn_24g.c): modify CFG_RF_24G_CODE_ID is 0x23

```
1. //2.4G     model 0 --- ble,     No 0---2.4G     pairing code
```

```
2. /* #define CFG_RF_24G_CODE_ID        (0) // <=24bits */
3. #define CFG_RF_24G_CODE_ID          (0x23) //<=24bits
```

2.16.3 Setting the **2.4G** physical layer

The transmission rate can be set by setting the physical layer. At present, the physical layer included in the Bluetooth protocol is supported by our company (including 1M, 2M, CODED S2, CODED S8), the specific settings are as follows:

Set to 1M physical layer transceiver (1M does not need to set S2/S8, so SELECT_CODED_S2_OR_S8 can be set at will):

```
1. //  Select physical layer
2. #define SELECT_PHY                  CONN_SET_1M_PHY// 1M:CONN_SET_1M_PHY 2M:CONN_SET_
   2M_PHY CODED:CONN_SET_CODED_PHY
3. //  choose CODED type  :S2 or S8
4. #define SELECT_CODED_S2_OR_S8       CONN_SET_PHY_OPTIONS_S2// S2:CONN_SET_PHY_OPTIONS
   _S2 S8:CONN_SET_PHY_OPTIONS_S8
```

Set to 2M physical layer transceiver (2M does not need to set S2/S8, so SELECT_CODED_S2_OR_S8 can be set at will):

```
1. //  Select physical layer
2. #define SELECT_PHY                  CONN_SET_2M_PHY// 1M:CONN_SET_1M_PHY 2M:CONN_SET_
   2M_PHY CODED:CONN_SET_CODED_PHY
3. //  choose CODED type  :S2 or S8
4. #define SELECT_CODED_S2_OR_S8       CONN_SET_PHY_OPTIONS_S2// S2:CONN_SET_PHY_OPTIONS
   _S2 S8:CONN_SET_PHY_OPTIONS_S8
```

Set to CODED S2 physical layer transceiver:

```
1. //  Select physical layer
2. #define SELECT_PHY                  CONN_SET_CODED_PHY// 1M:CONN_SET_1M_PHY 2M:CONN_S
   ET_2M_PHY CODED:CONN_SET_CODED_PHY
3. //  choose CODED type  :S2 or S8
4. #define SELECT_CODED_S2_OR_S8       CONN_SET_PHY_OPTIONS_S2// S2:CONN_SET_PHY_OPTIONS
   _S2 S8:CONN_SET_PHY_OPTIONS_S8
```

Set to CODED S8 physical layer transceiver:

```
1. //  Select physical layer
2. #define SELECT_PHY                  CONN_SET_CODED_PHY// 1M:CONN_SET_1M_PHY 2M:CONN_S
   ET_2M_PHY CODED:CONN_SET_CODED_PHY
```

```
3. //      chooseCODED type :S2 or S8
4. #define SELECT_CODED_S2_OR_S8                CONN_SET_PHY_OPTIONS_S8// S2:CONN_SET_PHY_OPTIONS
    _S2 S8:CONN_SET_PHY_OPTIONS_S8
```

2.16.4 Data sending module

(1) Turn on the CONN_24G_KEEP_SEND_EN enable (data sending enable) in (app_conn_24g.c):

```
1. #define CONN_24G_KEEP_SEND_EN                1 //just for 2.4gtest keep data
```

The code will open the send test after connecting and close the send test after disconnecting:

```
1.          case BLE_ST_CREATE_CONN:
2.              /*
3.                * The bluetooth device has been connected to allow the host to send data
4.                */
5. #if CONN_24G_KEEP_SEND_EN && CONFIG_BT_GATT_CLIENT_NUM
6.              if (conn_24g_phy_test_timer_id == 0) {
7.                  log_info("OPEN CONN_24G_KEEP_SEND_EN\n");
8.                  conn_24g_phy_test_timer_id = sys_timer_add(NULL, conn_24g_phy_test, 200)
    ;
9.              }
```

```
1.          case BLE_ST_DISCONN:
2.              /*
3.                * Bluetooth is disconnected, and the host is set to turn off data transmission
4.                */
5. #if CONN_24G_KEEP_SEND_EN && CONFIG_BT_GATT_CLIENT_NUM
6.              log_info("CLOSE CONN_24G_KEEP_SEND_EN\n");
7.              sys_timeout_del(conn_24g_phy_test_timer_id);
8.              conn_24g_phy_test_timer_id = 0;
9. #endif
```

2.16.5 Binding and Unbinding Use

(1) Binding:

Principle: Ordinary Bluetooth device, after the slave connects to a device, it will remember the address of the peer, but it can still communicate with the new one next time.

The device is connected; this is because there is no binding. Since customers may need to bind functions, they are added;

After the slave is connected to a device, the pairing information is saved. When re-broadcasting, if the slave has pairing information, the slave will set the broadcast to directional

Broadcast (directed broadcast has directed connectivity). After the host connects to a slave device, it saves the slave pairing information, and when rescanning

If the host has pairing information, the host only scans the slaves with directional broadcast, and only connects with the same pairing information recorded by the host.

the slave

Slave settings (ble_24g_server.c): There is pairing information set as directional broadcast.

```
1. //  Configure directed broadcast information
2. conn_24g_server_adv_config.adv_type = ADV_DIRECT_IND_LOW;
3. conn_24g_server_adv_config.adv_interval = PAIR_DIRECT_LOW_ADV_INTERVAL;
4. log_info("===DIRECT_ADV address:");
5. put_buf(conn_24g_server_adv_config.direct_address_info, 7);
```

Host settings (ble_24g_client.c): After connecting, only directional broadcasts are recognized, and the search configuration is re-passed

```
1. //  Identify directional broadcast after connection, otherwise it will cause the slave to be disconnected and the host to connect again immediately
2. ble_gatt_client_set_search_config(&conn_24g_central_bond_config);
```

(2) Unbinding:

Principle: If there is binding, there must be a need for unbinding. This design sets AD key 0 to long press and lift to unbind. Delete the pairing information of the master and slave

The unbinding can be completed by removing the binding. After unbinding, turn on the scanning and broadcasting of the master and slave to prepare for the next binding.

Host (app_conn_24g.c): delete pairing information and set scan:

```
1. memset(&client_pair_bond_info[0], 0, 8);
2. conn_24g_pair_vm_do(client_pair_bond_info, sizeof(client_pair_bond_info), 1);
3. log_info("clear client pear info!");
4. put_buf(&client_pair_bond_info, 8);
5. ble_gatt_client_scan_enable(0);
6. ble_gatt_client_disconnect_all();
7. conn_24g_central_init();
8. ble_gatt_client_scan_enable(1);
```

Slave (app_conn_24g.c): delete pairing information and set adv:

```
1. memset(&pair_bond_info[0], 0, 8);
2. conn_24g_pair_vm_do(pair_bond_info, sizeof(pair_bond_info), 1);
3. log_info("clear server pear info!");
4. put_buf(&pair_bond_info, 8);
5. //      Close the module and reopen it
6. ble_gatt_server_adv_enable(0);
7. ble_gatt_server_disconnect_all();
8. /* ble_comm_disconnect(0x50); */
9. conn_24g_server_init();
10. ble_gatt_server_adv_enable(1);
```

2.17 APP - Tecent LL

2.17.1 Overview

This case is used to implement the Tencent Lianlian protocol, and the device can be controlled after the Tencent Lianlian WeChat applet is connected to the device.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.17.2 Project Configuration

Code project: apps\spp_and_le\board\bdxx\AC63xN_spp_and_le.cbp

(2) Configure app selection (apps\spp_and_le\include\app_config.h), select the corresponding application example as shown in the figure below.

```
1. //app case          choice only    1  to configure the corresponding  board_config.h
2. #define CONFIG_APP_LL_SYNC                    0 //Tencent Lianlian/
```

(3) Configure the board-level Bluetooth settings (apps\spp_and_le\board\brxx\board_acxxxx_demo.cfg), only open BLE but not EDR

```
1. //*************************************************************//
2. //                        Bluetooth configuration                          //
3. //*************************************************************//
4. #define TCFG_USER_TWS_ENABLE                  0 //tws function enable
5. #define TCFG_USER_BLE_ENABLE                  1 //BLE function enable
6. #define TCFG_USER_EDR_ENABLE                  0 //EDR function enable
```

2.17.3 Module Development

For details, refer to "Tencent Lianlian Development Documentation".

2.18 APP - YOURS

2.18.1 Overview

This case is used to implement the Tuya protocol, and the device can be controlled after connecting to the device using Tuya Smart or Tuya Cloud APP.

Supported board levels: bd29, br25, br23, br30, bd19, br34

Supported chips: AC631N, AC636N, AC635N, AC637N, AC632N, AC638N

2.18.2 Project Configuration

(1) Code project: apps\spp_and_le\board\bdxx\AC63xN_spp_and_le.cbp

(2) Configure app selection (apps\spp_and_le\include\app_config.h), select the corresponding application example as shown in the figure below.

```
3. //app case          choice only    1  to configure the corresponding  board_config.h
4. #define CONFIG_APP_TUYA                    0 //Tuya Protocol/
```

(3) Configure the board-level Bluetooth settings (apps\spp_and_le\board\brxx\board_acxxxx_demo.cfg), only open BLE but not EDR

```
7. //*************************************************************//
8. //                          Bluetooth configuration                          //
9. //*************************************************************//
10. #define TCFG_USER_TWS_ENABLE                0 //tws function enable
11. #define TCFG_USER_BLE_ENABLE                1 //BLE function enable
12. #define TCFG_USER_EDR_ENABLE                0 //EDR function enable
```

2.18.3 Module Development

For details, please refer to "Tuya Protocol Development Documentation".

2.19APP - Voice remote control

2.19.1 Overview

This case is used for the hid voice remote control function transmission function, and the supported encoding format.

IMAADPCMÿSpeexÿOpusÿSBCÿmSBCÿLC3

Supported board levels: br30, br34

Supported chips: AC637N, AC638N

2.19.2 Project Configuration

(1) Code project: apps\hid\board\bdxx\AC63xN_hid.cbp

(2) Configure app selection (apps\spp_and_le\include\app_config.h), select the corresponding application example as shown in the figure below.

```
5. //app case          choice only    1  to configure the corresponding  board_config.h
6. #define CONFIG_APP_REMOTE_CONTROL                    0//Voice remote control
```

(3) Configure the board-level Bluetooth settings (apps\spp_and_le\board\brxx\board_acxxxx_demo.cfg), only open BLE but not EDR

```
13. //*************************************************************//
14. //                          Bluetooth configuration                                //
15. //*************************************************************//
16. #define TCFG_USER_TWS_ENABLE                    0 //tws function enable
17. #define TCFG_USER_BLE_ENABLE                    1 //BLE function enable
18. #define TCFG_USER_EDR_ENABLE                    0 //EDR function enable
```

2.19.3 Module Development

The example is based on the original app_keyboard example, adding support for coding applications. The encoding format supports IMA ADPCM, Speex, Opus, SBC, mSBC, LC3 and other formats.

The general encoding interface is in the audio_codec_demo.c file, the configuration of the Audio encoding function is used, refer to (5.2 Audio use Use) in the <Using of General Coding Interface> chapter.

2.20 GATT COMMON

2.20.1 Overview

Mainly describe the new version of SDK (v2.0.0 and later) that adds a GATT public module in the COMMON directory, mainly dealing with the GATT layer and blue

The functional information exchange of the tooth protocol stack, GATT and the functional information exchange of the apps layer; effectively make the apps do not need to rely too much on the protocol stack interface

You only need to pay attention to the configuration and control of the GATT interface module.

2.20.2 Code Description

Directory path: apps\common\third_party_profile\jieli\gatt_common

File information:

1. le_gatt_client.c
2. le_gatt_common.c
3. le_gatt_common.h
4. le_gatt_server.c

The module is mainly managed by the gatt configuration in app_config.h, as follows:

```
1. //  BluetoothBECAME configure
2. #define CONFIG_BT_GATT_COMMON_ENABLE      1 //  Configure using gatt public module
3. #define CONFIG_BT_SM_SUPPORT_ENABLE       0 //  Configure whether to support encryption
4. #define CONFIG_BT_GATT_CLIENT_NUM         0 //  Configure the host client number (app not support)
5. #define CONFIG_BT_GATT_SERVER_NUM         1 //  Configure the slave server number
6. #define CONFIG_BT_GATT_CONNECTION_NUM     (CONFIG_BT_GATT_SERVER_NUM + CONFIG_BT_GA
   TT_CLIENT_NUM) //  Configure the number of connections
```

GATT master-slave roles independently support up to 8, if used at the same time, the connection supports at most link adjustment.

le_gatt_common.c

------Mainly execute the initialization driver of the Bluetooth protocol stack, including configuration initialization such as GATT, ATT and SM, and drive the GATT server and

The GATT client module is implemented, distributes the message processing of the two modules, and supports multi-machine GATT multi-machine connection management. Provide some GATT pass

Called through the interface, such as ATT data transmission, Bluetooth link disconnection, etc.

For the list of provided interfaces, see the code comments for specific interface definitions:

```
1. //common
2. u32 ble_comm_cbuffer_vaild_len(u16 conn_handle);
3. int ble_comm_att_send_data ( u16 conn_handle, u16 att_handle, u8 * data, u16 len, att_o
       p_type_e op_type);
4. bool ble_comm_att_check_send(u16 conn_handle, u16 pre_send_len);
5. const char *ble_comm_get_gap_name(void);
6. int ble_comm_disconnect(u16 conn_handle);
7. u8 ble_comm_dev_get_handle_state(u16 handle, u8 role);
8. void ble_comm_dev_set_handle_state(u16 handle, u8 role, u8 state);
9. void ble_comm_register_state_cbk(void (*cbk)(u16 handle, u8 state));
10. s8 ble_comm_dev_get_index(u16 handle, u8 role);
11. s8 ble_comm_dev_get_idle_index(u8 role);
12. u8 ble_comm_dev_get_handle_role(u16 handle);
13. u16 ble_comm_dev_get_handle(u8 index, u8 role);
14. void ble_comm_set_config_name(const char *name_p, u8 add_ext_name);
15. void ble_comm_init(const gatt_ctrl_t *control_blk);
16. void ble_comm_exit(void);
17.void ble_comm_module_enable(u8 en);
18. int ble_comm_set_connection_data_length(u16 conn_handle, u16 tx_octets, u16 tx_time)
       ;
19. int ble_comm_set_connection_data_phy(u16 conn_handle, u8 tx_phy, u8 rx_phy);
```

le_gatt_server.c

------Mainly perform GATT server role operations, such as profile configuration, broadcast, connection parameter update process, ota general operations

Operation, protocol stack event processing, multi-machine mechanism management, etc. The apps layer only needs to perform configuration operations to implement the driver module implementation base.

this function.

For the list of provided interfaces, see the code comments for specific interface definitions:

```
1. //server
2. void ble_gatt_server_init(gatt_server_cfg_t *server_cfg);
3. void ble_gatt_server_exit(void);
4. ble_state_e ble_gatt_server_get_work_state(void);
5. ble_state_e ble_gatt_server_get_connect_state(u16 conn_handle);
```

```
6. int ble_gatt_server_adv_enable(u32 en);
7. void ble_gatt_server_module_enable (u8 en);
8. void ble_gatt_server_disconnect_all(void);
9. int ble_gatt_server_connetion_update_request(u16 conn_handle, const struct conn_upd
    ate_param_t *update_table, u16 table_count);
10. int ble_gatt_server_characteristic_ccc_set(u16 conn_handle, u16 att_handle, u16 ccc_
    config);
11. u16 ble_gatt_server_characteristic_ccc_get (u16 conn_handle, u16 att_handle);
12. void ble_gatt_server_set_update_send ( u16 conn_handle, u16 att_handle, u8 att_handle_
    type);
13. void ble_gatt_server_receive_update_data(void *priv, void *buf, u16 len);
14. void ble_gatt_server_set_adv_config(adv_cfg_t *adv_cfg);
15. void ble_gatt_server_set_profile(const u8 *profile_table, u16 size);
```

le_gatt_client.c

------Mainly perform GATT client role operations, such as configuring scan parameters, matching device scan connections, matching profile search connections

Connection, event processing of the protocol stack; multi-machine mechanism management, etc. The apps layer only needs to perform configuration operations to implement the driver module implementation base.

this function.

For the list of provided interfaces, see the code comments for specific interface definitions:

```
1. //client
2. void ble_gatt_client_init(gatt_client_cfg_t *client_cfg);
3. void ble_gatt_client_exit(void);
4. void ble_gatt_client_set_scan_config(scan_conn_cfg_t *scan_conn_cfg);
5. void ble_gatt_client_set_search_config(gatt_search_cfg_t *gatt_search_cfg);
6. ble_state_e ble_gatt_client_get_work_state(void);
7. ble_state_e ble_gatt_client_get_connect_state(u16 conn_handle);
8. int ble_gatt_client_create_connection_request(u8 *address, u8 addr_type, int mode);
9. int ble_gatt_client_create_connection_cannel(void);
10. int ble_gatt_client_scan_enable(u32 en);
11.void ble_gatt_client_module_enable (u8 en);
12. void ble_gatt_client_disconnect_all(void);
```

le_gatt_common.1

------ Module configuration structure definition, providing interface definition.

Chapter 3 SIG Mesh Instructions for Use

3.1 Overview

Comply with the Bluetooth SIG Mesh protocol and realize the communication between nodes in the network based on Bluetooth 5 ble. The specific functions are as follows:

ÿFull node type support (Relay/Proxy/Friend/Low Power);

ÿ Support network access by PB-GATT (mobile phone APP distribution network, support "nRF Mesh" Android and Apple latest versions);

ÿ Support network access by PB-ADV (“Tmall Genie” distribution network);

ÿ Support device power-on self-configuration network (devices can be in the same network without a gateway, support encryption key customization);

ÿ Node relay/beacon function can be modified;

ÿ The node address can be customized;

ÿ The node information is saved after power failure;

ÿ Node publish (Publish) and subscription (Subscribe) addresses can be modified;

ÿ Support node Reset as unconfigured device;

ÿ Support Bluetooth SIG existing Models and user-defined Vendor Models.

Supported board levels: bd29, br25, br30, bd19, br34, br23

Supported chips: AC631N, AC636N, AC637N, AC632N, AC638N

3.2 Engineering configuration

Code project: apps\mesh\board\bd29\AC631N_mesh.cbp

```
ÿ mesh/
  ÿ api/
    feature_correct.h
    mesh_config_common.c
    model_api.c model_api.h

  ÿ board/ ÿ
    bd29/
      board_ac630x_demo.c
      board_ac630x_demo_cfg.h
      board_ac6311_demo.c
      board_ac6311_demo_cfg.h
      board_ac6313_demo.c
      board_ac6313_demo_cfg.h
      board_ac6318_demo.c
      board_ac6318_demo_cfg.h
      board_ac6319_demo.c
      board_ac6319_demo_cfg.h
      board_config.h
  ÿ examples/
    generic_onoff_client.c
    generic_onoff_server.c
    vendor_client.c
    vendor_server.c
    AliGenie_socket.c
```

Under api/model_api.h , select the corresponding example by configuring CONFIG_MESH_MODEL . The SDK provides 5 application examples.

SIG_MESH_GENERIC_ONOFF_CLIENT is selected by default , which is the example located under examples/generic_onoff_client.c .

```
4. //< Detail in "MshMDLv1.0.1" 5.
#define SIG_MESH_GENERIC_ONOFF_CLIENT 0 // examples/ generic_onof _client.c 6. #define
SIG_MESH_GENERIC_ONOFF_SERVER 1 // examples/ generic_onof _server.c 7. #define
SIG_MESH_VENDOR_CLIENT 8. #define SIG_MESH_VENDOR_SERVER 9. #define
SIG_MESH_ALIGENIE_SOCKET 10. // more... 11.
                                                    2 // examples/ generic_onof _client.c
                                                    3 // examples/ vendor_server.c
                                                    4 // examples/ AliGenie_socket.c


12. //< Config which example will use in <examples>
13. #define CONFIG_MESH_MODEL                    SIG_MESH_GENERIC_ONOFF_CLIENT
```

3.2.2 **Mesh** configuration

Under api/mesh_config_common.c , you can freely configure network and node characteristics, such as LPN/Friend node characteristics, Proxy Broadcast interval before and after the distribution network , broadcast interval and duration when node information is transmitted, etc.

The following example shows the configuration of broadcast interval and duration when node information is transmitted, and broadcast interval before and after network configuration under PB-GATT Configuration.

```
/**
* @brief Config adv bearer hardware param when node send messages */
/*-------------------------------------------------------------*/ const u16
config_bt_mesh_node_msg_adv_interval = ADV_SCAN_UNIT(10); // unit: ms const u16
config_bt_mesh_node_msg_adv_duration = 100; // unit: ms

/**
* @brief Config proxy connectable adv hardware param
*/
/*-------------------------------------------------------------*/ const u16
config_bt_mesh_proxy_unprovision_adv_interval = ADV_SCAN_UNIT(30); // unit: ms const u16
config_bt_mesh_proxy_pre_node_adv_interval = ADV_SCAN_UNIT(10); // unit: ms
_WEAK_
const u16 config_bt_mesh_proxy_node_adv_interval = ADV_SCAN_UNIT(300); // unit: ms
```

Note: The declaration of "_WEAK_" before the constant means that the constant can be redefined in other files. if you want a If the configuration is privatized to an instance, the "_WEAK_" statement should be added before the configuration under the file, and in the instance file where it is located. Redefine this constant configuration.

3.2.3 **board** configuration

Under board/xxxx/board_config.h , you can select different boards according to different packages . Take AC63X as an example, the default selection is CONFIG_BOARD_AC630X_DEMO as the target board.

```
4. /* 5. *
Board configuration selection
6. */
7. #define CONFIG_BOARD_AC630X_DEMO
8. // #define CONFIG_BOARD_AC6311_DEMO
9. // #define CONFIG_BOARD_AC6313_DEMO
10. // #define CONFIG_BOARD_AC6318_DEMO
11. // #define CONFIG_BOARD_AC6319_DEMO
12.
13. #include "board_ac630x_demo_cfg.h" 14.
#include "board_ac6311_demo_cfg.h" 15. #include
"board_ac6313_demo_cfg.h" 16. #include
"board_ac6318_demo_cfg.h" 17. #include
"board_ac6319_demo_cfg.h"
```

3.3.1 SIG Generic OnOff Client

1 Introduction

The instance is configured through the mobile phone "nRF Mesh"

```
-> Device name: OnOff_cli
-> Node FeaturesÿProxy + Relay
-> Authentication method: NO OOB
-> Number of Elements: 1
-> ModelÿConfiguration Server + Generic On Off Client
```

2. Actual operation

1) .Basic configuration

Use USB DP as serial port debug pin, baud rate is 1000000

Use PA3 as AD button

The device name is "OnOff_cli" and the MAC address is "11:22:33:44:55:66"

```
-> api/model_api.h
#define CONFIG_MESH_MODEL -> board/                SIG_MESH_GENERIC_ONOFF_CLIENT
xxxx/board_xxxx_demo_cfg.h
#define TCFG_UART0_TX_PORT #define                  IO_PORT_DP
TCFG_UART0_BAUDRATE #define                        1000000
TCFG_ADKEY_ENABLE ENABLE_THIS_MOUDLE //#define TCFG_ADKEY_PORT        Whether to enable the button AD
IO_PORTA_03 //IO #define TCFG_ADKEY_AD_CHANNEL AD_CH_PA3   Pay attention to whether the selected port supports the function AD

-> examples/generic_onoff_client.c #define
BLE_DEV_NAME #define                                'O', 'n', 'O', 'f', 'f', '_', 'c', 'l', 'i'
CUR_DEVICE_MAC_ADDR                                0x112233445566
```

For the MAC address, if you want different devices to use random values when they are powered on for the first time, you can follow the steps below to pass NULL into the bt_mac_addr_set function

If you want to configure the MAC address with the configuration tool , you should not call the bt_mac_addr_set function

```
-> examples/generic_onoff_client.c void
bt_ble_init(void)
{
    u8 bt_addr[6] = {MAC_TO_LITTLE_ENDIAN(CUR_DEVICE_MAC_ADDR)};

    bt_mac_addr_set(NULL);

    mesh_setup(mesh_init);
}
```

2) .Compile the project and download it to the target board, connect the serial port, connect the AD button, power on or reset the device

3) .Use the mobile phone APP "nRF Mesh" for network distribution, please click here for detailed operation- >click here<-

(The animation is located in the same level directory of the document, if the click is invalid, please open "Generic_On_Off_Client.gif" manually)

After the distribution network is completed, the node structure is as follows:

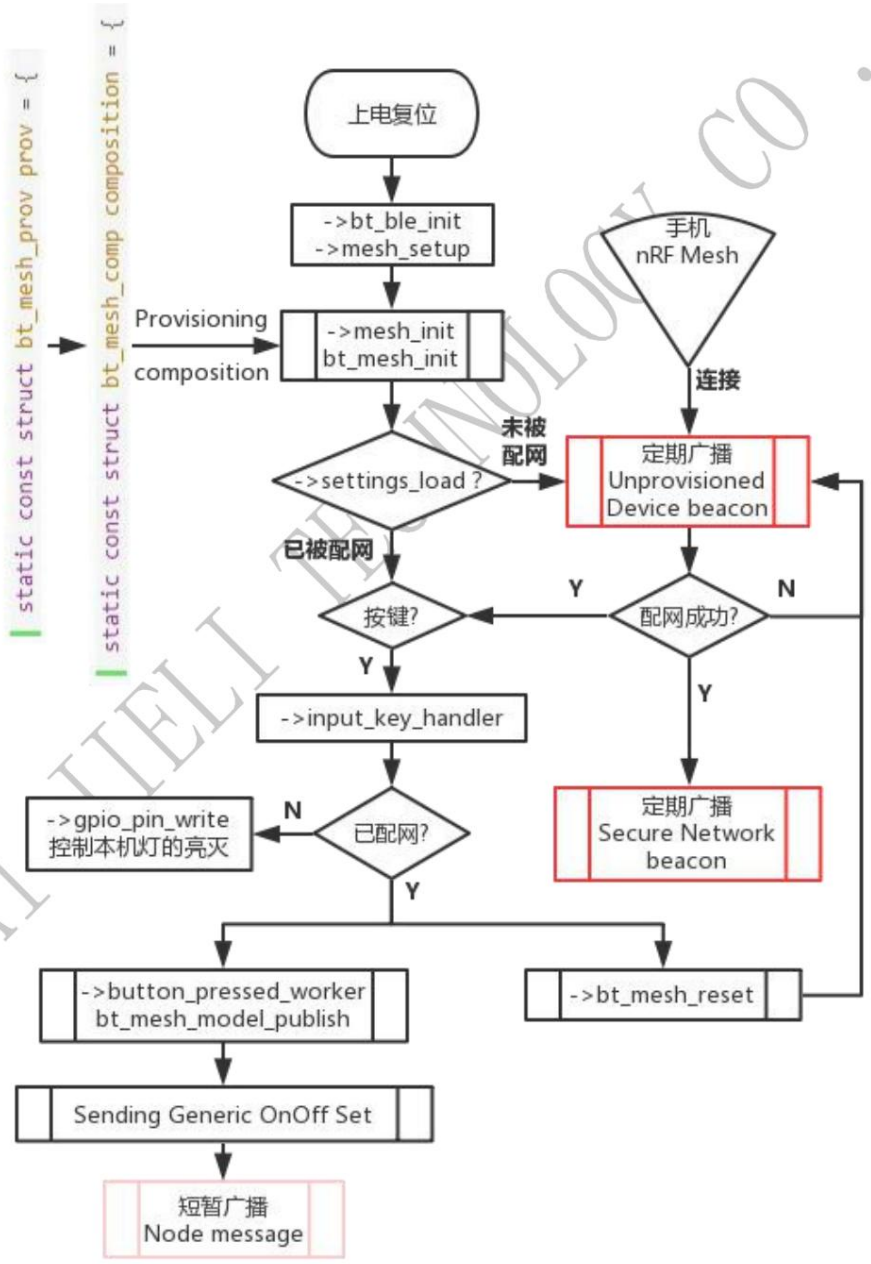
```
ÿ Elements ÿ
  Element
    Configuration Server #SIG Model ID: 0x0000
    Generic On Off Client #SIG Model ID: 0x1001
```

4). Press the button at this time to publish the switch information to the Group address 0xC000 . If combined with the next section

SIG Generic OnOff Server, you can control the on and off of the led light of this server device

3. Code interpretation

1) .Client operation flow chart



3.3.2 SIG Generic OnOff Server

1 Introduction

The instance is configured through the mobile phone "nRF Mesh"

```
-> Device name: OnOff_srv
-> Node FeaturesÿProxy + Relay
-> Authentication method: NO OOB
-> Number of Elements: 1
-> ModelÿConfiguration Server + Generic On Off Server
```

2. Actual operation

1) .Basic configuration

Use USB DP as serial port debug pin, baud rate is 1000000

Use PA1 to control LED lights

The device name is "OnOff_srv" and the MAC address is "22:22:33:44:55:66"

```
-> api/model_api.h
#define CONFIG_MESH_MODEL -> board/          SIG_MESH_GENERIC_ONOFF_SERVER
xxxx/board_xxxx_demo_cfg.h
#define TCFG_UART0_TX_PORT #define           IO_PORT_DP
TCFG_UART0_BAUDRATE -> examples/             1000000
generic_onoff_server.c #define
BLE_DEV_NAME #define                          'O', 'n', 'O', 'f', 'f', '_', 's', 'r', 'v'
CUR_DEVICE_MAC_ADDR const u8                 0x222233445566
led_use_port[] = {
    IO_PORTA_01,};
```

For the MAC address, if you want different devices to use random values when they are powered on for the first time, you can follow the steps below to pass NULL into the bt_mac_addr_set function

If you want to configure the MAC address with the configuration tool, you should not call the bt_mac_addr_set function

```
-> examples/generic_onoff_server.c void
bt_ble_init(void)
{
        u8 bt_addr[6] = {MAC_TO_LITTLE_ENDIAN(CUR_DEVICE_MAC_ADDR)};

        bt_mac_addr_set(NULL);

        mesh_setup(mesh_init);
}
```

2). Compile the project and download it to the target board, connect the serial port, connect the LED light for demonstration , power on or reset the device

3) .Use the mobile phone APP "nRF Mesh" for network distribution, please click here for detailed operation- >click here<-

(The animation is located in the same level directory of the document, if the click is invalid, please manually open "Generic_On_Off_Server.gif")

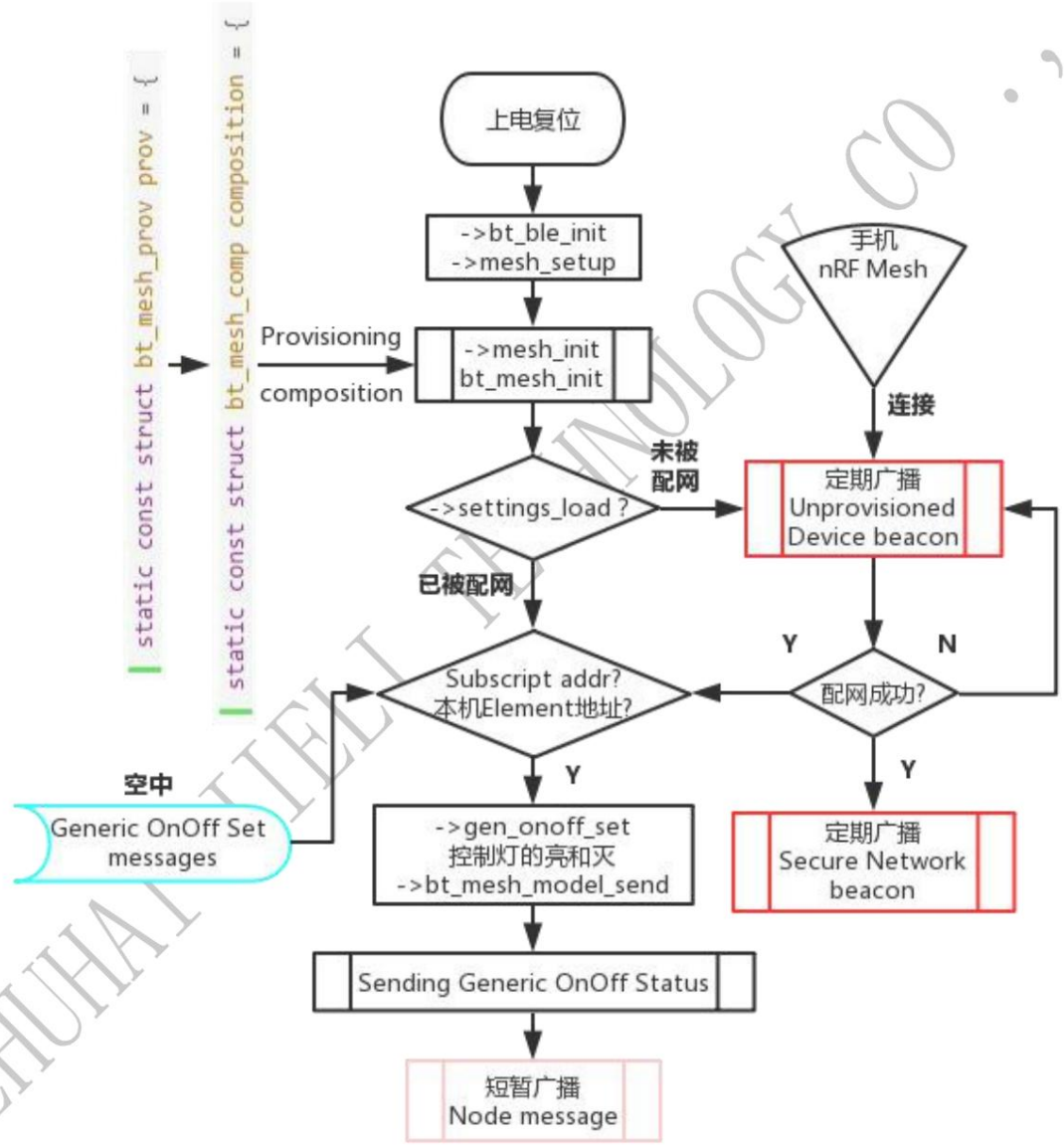
After the distribution network is completed, the node structure is as follows:

```
ÿ Elements ÿ
  Element
      Configuration Server #SIG Model ID: 0x0000
      Generic On Off Server #SIG Model ID: 0x1000
```

4) .Combined with the previous section SIG Generic OnOff Client, if the Client device presses the button, then the LED light of the machine will turn on will turn on or off

3. Code interpretation

1) .Server operation flow chart



3.3.3 SIG AliGenie Socket

1 Introduction

This example is based on Alibaba's "IoT open platform" "about" Tmall Genie Bluetooth *mesh* Software Basic Specifications", according to" hardware Category Specifications" Describe yourself as a" socket ", through the "Tmall Genie" voice input to discover the connection (distribution network) and control the device ready.

2. Actual operation

1) .Basic configuration

Use USB DP as serial port debug pin, baud rate is 1000000

Use PA1 to control the LED light (simulate the on and off operation of the socket)

The device name is "AG-Socket"

The triplet (MAC address, ProductID, Secret) is applied on the Tmall Genie developer website

```
-> api/model_api.h
#define CONFIG_MESH_MODEL -> board/                SIG_MESH_ALIGENIE_SOCKET
xxxx/board_xxxx_demo_cfg.h
#define TCFG_UART0_TX_PORT #define                  IO_PORT_DP
TCFG_UART0_BAUDRATE -> examples/                   1000000
AliGenie_socket.c #define BLE_DEV_NAME //
<                                                  'A', 'G', '-', 'S', 'o', 'c', 'k', 'e', 't'
    Triplet In this (example, the socket-type triplet applied in the name of the individual)
#define CUR_DEVICE_MAC_ADDR #define                0x28fa7a42bf0d
PRODUCT_ID #define DEVICE_SECRET                   12623
const u8 led_use_port[] = {                        "753053e923f30c9f0bc4405cf13ebda6"

    IO_PORTA_01,};
```

For the MAC address, in this example, it must be passed into the bt_mac_addr_set function according to the MAC address in the triplet

```
-> examples/AliGenie_socket.c void
bt_ble_init(void)
{
        u8 bt_addr[6] = {MAC_TO_LITTLE_ENDIAN(CUR_DEVICE_MAC_ADDR)};

        bt_mac_addr_set(bt_addr);

        mesh_setup(mesh_init);
}
```

2). Compile the project and download it to the target board, connect the serial port, connect the LED light for demonstration , power on or reset the device

3) .Tmall Genie is connected to the Internet

ÿ. Power on the "Tmall Genie", and press and hold the voice button on the device to make the device enter the state of waiting for connection

ÿ. Download the "Tmall Genie" APP from the mobile application store, and log in to the personal center on the APP

ÿ. Turn on the "WLAN" of the mobile phone, and connect the "Tmall Genie" to the Internet through the mobile phone hotspot

For details, please->click here<-

(The animation is located in the same level directory of the document, if the click is invalid, please open "AliGenie_connect.gif" manually)

4) .Network distribution and control through Tmall Genie

ÿ. Distribution Network Dialogue

User: "Tmall Genie, search device"

Tmall Genie: "Found a smart socket, whether it is connected"

User: "connect"

Tmall Genie: "The connection is successful..."

ÿ. Voice control socket commands (available via "IoT open platform "Add custom voice commands)

Command: "Tmall Genie, open the socket" Effect: The LED light on the development board is turned on

Command: "Tmall Genie, close the socket" Effect: The LED light on the development board is turned off

3. Code interpretation

1). Distribution network

The key lies in how to set the triplet applied for on the Tmall Genie developer website

ÿ. Apply for triples on the Tmall Genie developer website and fill in the corresponding macro definitions in the following files

For example, the applied triples are as follows:

Product ID (decimal)	**Device Secret**	**Mac** address
12623	753053e923f30c9f0bc4405cf13ebda6	28fa7a42bf0d

Then fill in according to the following rules, add 0x before the MAC, and wrap the Secret in double quotation marks

```
-> examples/AliGenie_socket.c
// < Triplet In this example, the socket-type triplet applied in the name of the individual)
#define CUR_DEVICE_MAC_ADDR #define 0x28fa7a42bf0d
PRODUCT_ID #define DEVICE_SECRET 12623
                                  "753053e923f30c9f0bc4405cf13ebda6"
```

ÿ. Build Element and Model

according to Socket Software Specification, to create a element, two models

Element	**Model**	property name
Primary	Generic On/Off Server 0x1000	switch
Primary	Vendor Model 0x01A80000	fault report/ timing control switch

The corresponding code operation is as follows:

ÿ The structure elements register a primary element = SIG root_models + Vendor_server_models

ÿ Structure root_models = Cfg_Server + Generic_OnOff_Server

ÿ ÿÿÿ vendor_server_models = Vendor_Client_Model + Vendor_Server_Model

```
// < Basic_Cfg_Server + Generic_OnOf _Server
static struct bt_mesh_model root_models[] = {
    BT_MESH_MODEL_CFG_SRV(&cfg_srv),
    BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_ONOFF_SRV, gen_onoff_srv_op, &gen_onoff_pub_srv, &onof
f_state[0]), };


// < Vendor_Client + Vendor_Server
static struct bt_mesh_model vendor_server_models[] = {
    BT_MESH_MODEL_VND(BT_COMP_ID_LF, BT_MESH_VENDOR_MODEL_ID_CLI, NULL, NULL, NULL),
    BT_MESH_MODEL_VND(BT_COMP_ID_LF, BT_MESH_VENDOR_MODEL_ID_SRV, vendor_srv_op, NULL,
&onoff_state[0]), };


// < Only primary element
static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, root_models, vendor_server_models), // primary element
    // second element
    // ... ...
};
```

2) .User data processing

ÿ. SIG Generic OnOff Server callback

Generic_OnOff_Server in the structure root_models registers the callback gen_onoff_srv_op to process user data

When a registration message such as BT_MESH_MODEL_OP_GEN_ONOFF_GET is received , the corresponding response such as gen_onoff_get will be called.

Call function for user data processing

```
static const struct bt_mesh_model_op gen_onoff_srv_op[] = {
    { BT_MESH_MODEL_OP_GEN_ONOFF_GET, 0, gen_onoff_get },
    { BT_MESH_MODEL_OP_GEN_ONOFF_SET, 2, gen_onoff_set },
    { BT_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK, 2, gen_onoff_set_unack },
    BT_MESH_MODEL_OP_END, };
```

ÿ. Vendor Model callback

The Vendor_Server_Model in the structure vendor_srv_op registers the callback vendor_srv_op to process user data

When a registration message such as VENDOR_MSG_ATTR_GET is received , the corresponding callback function such as vendor_attr_get will be called to perform

User data processing

```
static const struct bt_mesh_model_op vendor_srv_op[] = {
    { VENDOR_MSG_ATTR_GET, ACCESS_OP_SIZE, vendor_attr_get },
    { VENDOR_MSG_ATTR_SET, ACCESS_OP_SIZE, vendor_attr_set },
    BT_MESH_MODEL_OP_END, };
```

3.3.4 SIG Vendor Client

1 Introduction

The instance will automatically configure the network

```
1. -> Device name: Vd_cli
2. -> Node Features:Proxy
3. -> Authentication method: NO OOB
4. -> Number of Elements: 1
5. ->Root modelÿConfiguration Server + Configuration Client
6. ->Vendor model: Vendor_Client_Model
```

2. Actual operation

1) .Basic configuration

Use USB DP as serial port debug pin, baud rate is 1000000

Use PA3 as AD key port, connect PA3 with ADKEY

The device name is "Vd_cli", the "Node" address is "0x0001", and the "Group" address is "0xc000"

The device name is "OnOff_cli" and the MAC address is "11:22:33:44:55:66"

```
1. -> api/model_api.h
2. #define CONFIG_MESH_MODEL                SIG_MESH_VENDOR_CLIENT
3. > board/xxxx/board_xxxx_demo_cfg.h
4. #define TCFG_UART0_TX_PORT               IO_PORT_DP
5. #define TCFG_UART0_BAUDRATE              1000000
6. -> examples/generic_onoff_server.c
7. #define BLE_DEV_NAME                     'V', 'd', '_', 'c', 'l', 'i'
8. #define CUR_DEVICE_MAC_ADDR              0x222233445566
9. const u8 led_use_port[] = {
10. IO_PORTA_01,
11. };
```

2) .Compile the project and download it to the target board, connect the serial port, connect the AD button, power on or reset the device

3). Press the button at this time to publish the switch information to the Group address 0xC000, combined with the next section SIG Vendor

Client, you can control the LED lights on the server device to turn on and off

3. Code interpretation

1) .Distribution process

1. Configure node information and element composition

```
1. int bt_mesh_init(const struct bt_mesh_prov *prov,
2.                   const struct bt_mesh_comp *comp);
3.         parameters:
4.             Configuration information of the prov node
5.                 prov in this example: static const struct bt_mesh_prov prov = {
6.                         .uuid = dev_uuid,
7.                         .output_size = 0,
8.                         .output_actions = 0,
9.                         .output_number = 0,
10.                        .complete = prov_complete,
11.                        .reset = prov_reset,
12.                        };
13.            The element composition of the comp node
14.                comp in this example: static const struct bt_mesh_comp composition = {
15.                        .cid = BT_COMP_ID_LF,
16.                        .elem = elements,
17.                        .elem_count = ARRAY_SIZE(elements),
18.                        };
19.        return value:
20.            int type, return 0 for success, other for error
```

2. Configure the node address and network communication key

```
1. int bt_mesh_provision(const u8_t net_key[16], u16_t net_idx,
2.                       u8_t flags, u32_t iv_index, u16_t addr,
3.                       const u8_t dev_key[16]);
4.         parameters:
5.             addr          Node address
6.             net_key       Network keys, used to secure communications at the network layer
7.             net_idx       net_key the index of the network key
8.             dev_key       Device key, used to secure communication between the node and the configuration client.
```

```
9.      return_value:
10.         int type, return 0 for success, other for error
```

3. Add app_key to the node

The added AppKey must be used in pairs with the NetKey, and the AppKey is used to authenticate and encrypt received and sent messages.

dense

```
1. int bt_mesh_cfg_app_key_add(u16_t net_idx, u16_t addr, u16_t key_net_idx,
2.                             u16_t key_app_idx, const u8_t app_key[16],
3.                             u8_t *status);
4.      parameters:
5.          addr            Node address
6.          app_key         Application key, used to protect the communication of the upper transport layer
7.          key_app_idx     index of app_key
8.      return_value:
9.          int type
```

4. Summary

The net_key/dev_key/app_key configured in this instance must be the same as that of the server, otherwise normal communication cannot be performed

2) .model configuration

1. Bind the model to the app_key added to the node

Bind the app key to the element's model

```
1. int bt_mesh_cfg_mod_app_bind_vnd(u16_t net_idx, u16_t addr, u16_t elem_addr,
2.                                  u16_t mod_app_idx, u16_t mod_id, u16_t cid,
3.                                  u8_t *status);
4.      parameters:
5.          addr            Node address
6.          elem_addr is configured as the element address of this model
7.          mod_id          ID of the model
8.                          mod_id in this example: BT_MESH_VENDOR_MODEL_ID_CLI
9.          key_app_idx     index of app_key
10.         cid             company identifier
11.     return_value:
12.         int type
```

2. Add publish behavior to model

Configure the release status of a model

```
int bt_mesh_cfg_mod_pub_set_vnd(u16_t net_idx, u16_t addr, u16_t elem_addr,
                                u16_t mod_id, u16_t cid,
                                struct bt_mesh_cfg_mod_pub *pub, u8_t *status);
        parameters:
            addr The address of this node, namely node_addr
            elem_addr is configured as the element address of the send attribute in pub
            pub     The pub structure stores the relevant properties of the publish behavior
                    The pub structure in this example:
                    struct bt_mesh_cfg_mod_pub pub;
                    pub.addr = dst_addr;            //publish destination address
                    pub.app_idx = app_idx;          //index of app_key
                    pub.cred_flag = 0;              //friendship's credential flag
                    pub.ttl = 7;                    //Life cycle, determines the number of times it can be relayed
                    pub.period = 0;                 // Period of periodic status release
                    pub.transmit = 0;               //The number of retransmissions of each publish msg (higher 3 bits) + 50ms between retransmissions
    number of steps (lower 5 digits)

            status The status of the request message
```

3) .Node behavior

1. Publish behavior of client

When the button is pressed, enter the input_key_handle function, this function will obtain the key value of the button and the state of the button, and press the button.
deal with

```
void input_key_handler(u8 key_status, u8 key_number)
```

According to the different states of the key, the input_key_handle function will pass the corresponding state information (click is 1, long press is 0) through the knot
The structure struct _switch *sw is passed to the client_publish function

```
static void client_publish(struct switch *sw)
```

The client_publish function will process the msg to be sent. Next, the main functions in client_publish will be introduced in detail.

Function and composition of msg

First, the client_publish function obtains the corresponding vendor_model stored in the mod_cli_sw structure according to the key_number,

The sending attribute of this vendor_model has been configured in the bt_mesh_cfg_mod_pub_set_vnd function in the previous section

Then use the bt_mesh_model_msg_init function to initialize a structure msg and store the 3-byte opcode

```
1. void bt_mesh_model_msg_init(struct net_buf_simple *msg, u32_t opcode);
```

Next, the buffer_add_u8_at_tail function will be used to add a status value to the tail of msg for server-side control of LEDs

state

```
1. u8 *buffer_add_u8_at_tail(void *buf, u8 val);
```

Then use the buffer_memset function to fill the remaining empty part of msg with 0x02, and the remaining empty part of msg can be

Content can also be customized by the user

```
1. void *buffer_memset(struct net_buf_simple *buf, u8 val, u32 len);
```

Then use bt_mesh_model_publish to send the information containing msg

```
1. int bt_mesh_model_publish(struct bt_mesh_model *model);
```

2. Client callback function

BT_MESH_VENDOR_MODEL_ID_CLI in the structure vendor_client_models registers vendor_cli_op

for data processing

Called when the opposite ack msg is received and matches BT_MESH_VENDOR_MODEL_OP_STATUS

vendor_status callback function to process data

```
1.  static const struct bt_mesh_model_op vendor_cli_op[] = {
2. {
3. BT_MESH_VENDOR_MODEL_OP_STATUS, ACCESS_OP_SIZE, vendor_status },
4. BT_MESH_MODEL_OP_END,
5. };
```

The function of the vendor_status function: display the address of the sender server as the ack msg and the address controlled by the client

Happening

```
1.  static void vendor_status(struct bt_mesh_model *model,
2.                           struct bt_mesh_msg_ctx *ctx,
3.                           struct net_buf_simple *buf)
```

3.3.5 SIG Vendor Server

1 Introduction

The instance will automatically configure the network

```
1. -> Device name: Vd_srv
2. ->Node Features:Proxy
3. ->Authentication method: NO OOB
4. ->Number of Elements: 1
5. ->Root modelÿConfiguration Server + Configuration Client
6. ->Vendor model: Vendor_Client_Model
```

2. Actual operation

1) .Basic configuration

Use USB DP as serial port debug pin, baud rate is 1000000

Use PA3 as AD button port, connect PA1 with LED

The device name is "Vd_srv", the "Node" address is "0x0002", and the "Group" address is "0xc000"

```
1. -> api/model_api.h
2. #define CONFIG_MESH_MODEL                SIG_MESH_VENDOR_SERVER
3. > board/xxxx/board_xxxx_demo_cfg.h
4. #define TCFG_UART0_TX_PORT               IO_PORT_DP
5. #define TCFG_UART0_BAUDRATE              1000000
6. -> examples/generic_onoff_server.c
7. #define BLE_DEV_NAME                     'V', 'd', '_', 's', r', 'v'
8. #define CUR_DEVICE_MAC_ADDR              0x222233445566
9. const u8 led_use_port[] = {
10. IO_PORTA_01,
11. };
```

2) .Compile the project and download it to the target board, connect the serial port, connect the LED light, power on or reset the device

3). At this time, if the client side presses the button, the LED light on the server device will light up according to whether the button is clicked or long-pressed or extinguished

3. Code interpretation

1) .Distribution process

1. Configure the information and element composition of the node

```
1. int bt_mesh_init(const struct bt_mesh_prov *prov,
2.                  const struct bt_mesh_comp *comp);
3. parameters:
4.         Configuration information of the prov node
5.             prov in this example: static const struct bt_mesh_prov prov = {
6.                                 .uuid = dev_uuid,
7.                                 .output_size = 0,
8.                                 .output_actions = 0,
9.                                 .output_number = 0,
10.                                .complete = prov_complete,
11.                                .reset = prov_reset,
12.                                };
13.        The element composition of the comp node
14.            comp in this example: static const struct bt_mesh_comp composition = {
15.                                .cid = BT_COMP_ID_LF,
16.                                .elem = elements,
17.                                .elem_count = ARRAY_SIZE(elements),
18.                                };
19. return value:
20.        int type, return 0 for success, other for error
```

2. Configure the node address and network communication key

```
1. int bt_mesh_provision(const u8_t net_key[16], u16_t net_idx,
2.                       u8_t flags, u32_t iv_index, u16_t addr,
3.                       const u8_t dev_key[16]);
4. parameters:
5.         addr node address
6.         net_key network key used to secure communication at the network layer
7.         net_idx          index of net_key
8.         dev_key Device key used to secure communication between the node and the config client.
9. return_value:
10.        int type, return 0 for success, other for error
```

3. Add app_key to the node

The added AppKey must be used in pairs with the NetKey, and the AppKey is used to authenticate and encrypt received and sent messages.

dense

```
1. int bt_mesh_cfg_app_key_add(u16_t net_idx, u16_t addr, u16_t key_net_idx,
2.                             u16_t key_app_idx, const u8_t app_key[16],
3.                             u8_t *status);
4. parameters:
5.      addr            Node address
6.      key_app_idx     index of app_key
7.      app_key         Application key, used to protect the communication of the upper transport layer
8.      status          the status of the request message
9. return_value:
10.     int type
```

4. Summary

The net_key/dev_key/app_key configured in this instance must be the same as that of the server, otherwise normal communication cannot be performed

2) .model configuration

1. Bind the model to the app_key added to the node

Bind the app key to the element's schema

```
1. int bt_mesh_cfg_mod_app_bind_vnd(u16_t net_idx, u16_t addr, u16_t elem_addr,
2.                                  u16_t mod_app_idx, u16_t mod_id, u16_t cid,
3.                                  u8_t *status);
4. parameters:
5.      addr            Node address
6.      elem_addr is configured as the element address of this model
7.      mod_id          ID of the model
8.                      mod_id in this example: BT_MESH_VENDOR_MODEL_ID_CLI
9.      index of key_app_idx app_key
10.     cid             company identifier
11.     status          the status of the request message
12. return_value:
13.     int type
```

2. Add a Subscription address to the model

The subscription address of the configuration model, which is used to receive the msg published on the client side

```
1. int bt_mesh_cfg_mod_sub_add_vnd(u16_t net_idx, u16_t addr, u16_t elem_addr,
2.                                 u16_t sub_addr, u16_t mod_id, u16_t cid,
3.                                 u8_t *status);
4. parameters:
5.         addr            the address of this node
6.         elem_addr element address, fill in the address of the element corresponding to the subscription control information here
7.         Sub_addr Subscription address
8.         mod_id          ID of the model
9.         status          the status of the request message
10. return_value:
11.        int type
```

3) .Node behavior

1. The callback function of the server

BT_MESH_VENDOR_MODEL_ID_CLI in the structure vendor_client_models is registered

vendor_cli_op for data processing

Called when the opposite ack is received and BT_MESH_VENDOR_MODEL_OP_STATUS is matched

vendor_set callback function to process data

```
1.  static void vendor_set(struct bt_mesh_model *model,
2.                         struct bt_mesh_msg_ctx *ctx,
3.                         struct net_buf_simple *buf)
```

In the vendor_set function, use the buffer_pull_u8_from_head function to extract the led status information stored in msg

```
1. u8 buffer_pull_u8_from_head(void *buf);
```

Then use the gpio_pin_write function to turn on or off the lights

```
1. void gpio_pin_write(u8_t led_index, u8_t onoff)
2. parameters:
3.         led_index GPIO port index
4.         Onoff status of the onoff LED
```

After the lighting operation, the information should be organized as ack feedback to the client, first use the bt_mesh_model_msg_init function

Initialize a net_buf_simple type object ack_msg and add a 3-byte opcode to it

```
1. void bt_mesh_model_msg_init(struct net_buf_simple *msg, u32_t opcode)
```

Then use the buffer_add_u8_at_tail function to add the status of the LED at the end of the ack_msg function

```
1. u8 *buffer_add_u8_at_tail(void *buf, u8 val);
```

Then use the buffer_memset function to fill up ack_msg

```
1. void *buffer_memset(struct net_buf_simple *buf, u8 val, u32 len);
```

Finally, use the bt_mesh_model_send function to send the feedback information to the client

```
1. int bt_mesh_model_send(struct bt_mesh_model *model,
2.                         struct bt_mesh_msg_ctx *ctx,
3.                         struct net_buf_simple *msg,
4.                         const struct bt_mesh_send_cb *cb,
5.                         void *cb_data);
6. parameters:
7.          model the model to which the sent message belongs
8.          Environment information of ctx message, including communication key, life cycle, remote address, etc.
9.          msg feedback information ack_msg
10.         cb          optional callback for message sending, this instance is empty
11.         cb_data User data to pass to the callback, this instance is empty
```

3.3.6 SIG AliGenie Light

1 Introduction

This example is based on Alibaba's "IoT open platform "about" Tmall Genie Bluetooth *mesh* Software Basic Specifications",according to" hardware Category Specifications"Describe yourself as a" lamp', through the "Tmall Genie" voice input to discover the connection (distribution network) and control the device.

2. Actual operation

2) .Basic configuration

Use USB DP as serial port debug pin, baud rate is 1000000

Use PA1 to control the LED light (simulate the on and off operation of the socket)

The device name is "AG-Socket"

The triplet (MAC address, ProductID, Secret) is applied on the Tmall Genie developer website

```
-> api/model_api.h
#define CONFIG_MESH_MODEL -> board/          SIG_MESH_ALIGENIE_SOCKET
xxxx/board_xxxx_demo_cfg.h
#define TCFG_UART0_TX_PORT #define          IO_PORT_DP
TCFG_UART0_BAUDRATE                         1000000
```



```
-> examples/AliGenie_socket.c
#define BLE_DEV_NAME //<                    'A', 'G', '-', 'L', 'i', 'g', 'h', 't'
    this example{Socket class triplet triplet applied in the name of an individual in}
#define CUR_DEVICE_MAC_ADDR #define        0x18146c110001
PRODUCT_ID #define DEVICE_SECRET            7218909
const u8 led_use_port[] = {                 "aab00b61998063e62f98ff04c9a787d4"

   IO_PORTA_01,};
```

For the MAC address, in this example, it must be passed into the bt_mac_addr_set function according to the MAC address in the triplet

```
-> examples/AliGenie_socket.c
void bt_ble_init(void)
{
      u8 bt_addr[6] = {MAC_TO_LITTLE_ENDIAN(CUR_DEVICE_MAC_ADDR)};

      bt_mac_addr_set(bt_addr);

     mesh_setup(mesh_init);
}
```

4) .Compile the project and download it to the target board, connect the serial port, connect the LED light for demonstration, power on or reset the device

5) .Tmall Genie is connected to the Internet

ÿ. Power on the "Tmall Genie", and press and hold the voice button on the device to make the device enter the state of waiting for connection

ÿ. Download the "Tmall Genie" APP from the mobile application store, and log in to the personal center on the APP

ÿ. Turn on the "WLAN" of the mobile phone, and connect the "Tmall Genie" to the Internet through the mobile phone hotspot

For details, please->click here<-

(The animation is located in the same level directory of the document, if the click is invalid, please open "AliGenie_connect.gif" manually)

5) .Network distribution and control through Tmall Genie

ÿ. Distribution Network Dialogue

User: "Tmall Genie, search device"

Tmall Genie: "Found a smart socket, whether it is connected"

User: "connect"

Tmall Genie: "The connection is successful..."

ÿ. Voice control socket commands (available via "IoT open platform" Add custom voice commands)

Command: "Tmall Genie, turn on the lights" Effect: The LED light on the development board is turned on

Command: "Tmall Genie, adjust the brightness of the light to 50" Effect: The brightness of the LED light on the development board is 50%

Command: "Tmall Genie, turn off the lights" Effect: The LED light on the development board is turned off

3. Code interpretation

2). Distribution network

The key lies in how to set the triplet applied for on the Tmall Genie developer website

ÿ. Apply for triples on the Tmall Genie developer website and fill in the corresponding macro definitions in the following files

For example, the applied triples are as follows:

Product ID (decimal)	**Device Secret**	**Mac** address
7218909	aab00b61998063e62f98ff04c9a787d4	18146c110001

Then fill in according to the following rules, add 0x before the MAC, and wrap the Secret in double quotation marks

```
-> examples/AliGenie_socket.c
// < Triplet In this example, the socket-type triplet applied in the name of the individual)
#define CUR_DEVICE_MAC_ADDR 0x18146c110001
#define PRODUCT_ID #define 7218909
DEVICE_SECRET "aab00b61998063e62f98ff04c9a787d4"
```

ÿ. Build Element and Model

according to Specification for lighting fixtures, to create a element, three models

Element	**Model**	property name
Primary	Generic On/Off Server 0x1000	switch
Primary	Light_Lightness_Server 0x1300	brightness
Primary	Vendor Model 0x01A80000	fault report/ timing control switch

The corresponding code operation is as follows:

ÿ The structure elements register a primary element = SIG root_models + Vendor_server_models

ÿ ÿÿÿ root_models = Cfg_Server + Generic_OnOff_Server + Light_Lightness_Server

ÿ ÿÿÿ vendor_server_models = Vendor_Client_Model + Vendor_Server_Model

```
// < Basic_Cfg_Server + Generic_OnOf _Server
static struct bt_mesh_model root_models[] = {
    BT_MESH_MODEL_CFG_SRV(&cfg_srv),
    BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_ONOFF_SRV, gen_onoff_srv_op, &gen_onoff_pub_srv, &onof
f_state[0]),
BT_MESH_MODEL(BT_MESH_MODEL_ID_LIGHT_LIGHTNESS_SRV, light_lightness_srv_op,
&gen_onoff_pub_srv, &light), };


// < Vendor_Client + Vendor_Server
static struct bt_mesh_model vendor_server_models[] = {
    BT_MESH_MODEL_VND(BT_COMP_ID_LF, BT_MESH_VENDOR_MODEL_ID_CLI, NULL, NULL, NULL),
    BT_MESH_MODEL_VND(BT_COMP_ID_LF, BT_MESH_VENDOR_MODEL_ID_SRV, vendor_srv_op, NULL,
&onoff_state[0]), };
```

```
// < Only primary element
static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, root_models, vendor_server_models), // primary element
    // second element
    // ... ...
};
```

3) .User data processing

ÿ. SIG Generic OnOff Server callback

Generic_OnOff_Server in the structure root_models registers the callback gen_onoff_srv_op to process user data

When a registration message such as BT_MESH_MODEL_OP_GEN_ONOFF_GET is received , the corresponding response such as gen_onoff_get will be called.

Call function for user data processing

```
static const struct bt_mesh_model_op gen_onoff_srv_op[] = {
    { BT_MESH_MODEL_OP_GEN_ONOFF_GET, 0, gen_onoff_get },
    { BT_MESH_MODEL_OP_GEN_ONOFF_SET, 2, gen_onoff_set },
    { BT_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK, 2, gen_onoff_set_unack },
    BT_MESH_MODEL_OP_END, };
```

ÿ. Vendor Model callback

Vendor_Server_Model in the structure vendor_server_models registers the callback vendor_srv_op to process user data

reason

When a registration message such as VENDOR_MSG_ATTR_GET is received , the corresponding callback function such as vendor_attr_get will be called to perform

User data processing

```
static const struct bt_mesh_model_op vendor_srv_op[] = {
    { VENDOR_MSG_ATTR_GET, ACCESS_OP_SIZE, vendor_attr_get },
    { VENDOR_MSG_ATTR_SET, ACCESS_OP_SIZE, vendor_attr_set },
    BT_MESH_MODEL_OP_END, };
```

ÿ. Light_Lightness_Server callback

Light_Lightness_Server in the structure vendor_server_models registers the callback vendor_srv_op to process user data

reason

When receiving registration messages such as BT_MESH_MODEL_OP_LIGHT_LIGHTNESS_GET , it will call vendor_attr_get and other peers

The corresponding callback function for user data processing

```
static const struct bt_mesh_model_op light_lightness_srv_op[] = {
    { BT_MESH_MODEL_OP_LIGHT_LIGHTNESS_GET, 0 ,lightness_get },
    { BT_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET, 0, lightness_set },
    { BT_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET_UNACK, 0, lightness_set_unack },
    BT_MESH_MODEL_OP_END, };
```

3.3.7 SIG AliGenie Fan

1 Introduction

This example is based on Alibaba's "IoT open platform"about" Tmall Genie Bluetooth mesh Software Basic Specifications",according to" hardware Category Specifications"Describe yourself as a" fan ", through the "Tmall Genie" voice input to discover the connection (distribution network) and control the device ready.

2. Actual operation

3) .Basic configuration

Use USB DP as serial port debug pin, baud rate is 1000000

Use PA1 to control the LED light (simulate the on and off operation of the socket)

The device name is "AG-Socket"

The triplet (MAC address, ProductID, Secret) is applied on the Tmall Genie developer website

```
-> api/model_api.h
#define CONFIG_MESH_MODEL ->                SIG_MESH_ALIGENIE_SOCKET
board/xxxx/board_xxxx_demo_cfg.h
#define TCFG_UART0_TX_PORT                  IO_PORT_DP
#define TCFG_UART0_BAUDRATE ->              1000000
examples/AliGenie_socket.c #define
BLE_DEV_NAME //< #define                     'A', 'G', '-', 'F', 'a', 'n
CUR_DEVICE_MAC_ADDR #define                 Input in this example, the socket type triplet applied in the name of the individual
PRODUCT_ID #define DEVICE_SECRET             0x27fa7af002a0
const u8 led_use_port[] = {                 7809508
                                            "d2729d5f3898079fa7b697c76a7bfe8e"

    IO_PORTA_01,};
```

For the MAC address, in this example, it must be passed into the bt_mac_addr_set function according to the MAC address in the triplet

```
-> examples/AliGenie_socket.c void
bt_ble_init(void)
{
        u8 bt_addr[6] = {MAC_TO_LITTLE_ENDIAN(CUR_DEVICE_MAC_ADDR)};

        bt_mac_addr_set(bt_addr);

       mesh_setup(mesh_init);
}
```

6). Compile the project and download it to the target board, connect the serial port, and connect the LED light for demonstration (used to indicate the fan switch status and wind speed),

Power on or reset the device

7) .Tmall Genie is connected to the Internet

ÿ. Power on the "Tmall Genie", and press and hold the voice button on the device to make the device enter the state of waiting for connection

ÿ. Download the "Tmall Genie" APP from the mobile application store, and log in to the personal center on the APP

ÿ. Turn on the "WLAN" of the mobile phone, and connect the "Tmall Genie" to the Internet through the mobile phone hotspot

For details, please->click here<-

(The animation is located in the same level directory of the document, if the click is invalid, please open "AliGenie_connect.gif" manually)

6) .Network distribution and control through Tmall Genie

ÿ. Distribution Network Dialogue

User: "Tmall Genie, search device"

Tmall Genie: "Found a fan, is it connected?"

User: "connect"

Tmall Genie: "The connection is successful..."

ÿ. Voice control socket commands (available via "IoT open platform "Add custom voice commands)

Command: "Tmall Genie, turn on the fan" Effect: The LED light on the development board is turned on

Command: "Tmall Genie, adjust the fan to 2" Effect: The brightness of the LED light on the development board changes

Command: "Tmall Genie, turn off the fan" Effect: The LED light on the development board is turned off

3. Code interpretation

3) . Distribution network

The key lies in how to set the triplet applied for on the Tmall Genie developer website

ÿ. Apply for triples on the Tmall Genie developer website and fill in the corresponding macro definitions in the following files

For example, the applied triples are as follows:

Product ID (decimal)	**Device Secret**	**Mac** address
7809508	d2729d5f3898079fa7b697c76a7bfe8e	27fa7af002a0

Then fill in according to the following rules, add 0x before the MAC, and wrap the Secret in double quotation marks

```
-> examples/AliGenie_socket.c
// < Triplet In this example, the socket-type triplet applied in the name of the individual)
#define CUR_DEVICE_MAC_ADDR 0x27fa7af002a0
#define PRODUCT_ID #define 7809508
DEVICE_SECRET "d2729d5f3898079fa7b697c76a7bfe8e"
```

ÿ. Build Element and Model

according to Socket Software Specification, to create a element, two models

Element	**Model**	property name
Primary	Generic On/Off Server 0x1000	switch

Primary	Vendor Model 0x01A80000	Adjust the fan speed gear/ timing control switch

The corresponding code operation is as follows:

ÿ The structure elements register a primary element = SIG root_models + Vendor_server_models

ÿ Structure root_models = Cfg_Server + Generic_OnOff_Server

ÿ ÿÿÿ vendor_server_models = Vendor_Client_Model + Vendor_Server_Model

```
// < Basic_Cfg_Server + Generic_OnOf _Server
static struct bt_mesh_model root_models[] = {
    BT_MESH_MODEL_CFG_SRV(&cfg_srv),
    BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_ONOFF_SRV, gen_onoff_srv_op, &gen_onoff_pub_srv, &onof
f_state[0]), };


// < Vendor_Client + Vendor_Server
static struct bt_mesh_model vendor_server_models[] = {
    BT_MESH_MODEL_VND(BT_COMP_ID_LF, BT_MESH_VENDOR_MODEL_ID_CLI, NULL, NULL, NULL),
    BT_MESH_MODEL_VND(BT_COMP_ID_LF, BT_MESH_VENDOR_MODEL_ID_SRV, vendor_srv_op, NULL, &onoff_state[0]), };


// < Only primary element
static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, root_models, vendor_server_models), // primary element
    // second element
    // ... ...
};
```

4) .User data processing

ÿ. SIG Generic OnOff Server callback

Generic_OnOff_Server in the structure root_models registers the callback gen_onoff_srv_op to process user data

When a registration message such as BT_MESH_MODEL_OP_GEN_ONOFF_GET is received , the corresponding response such as gen_onoff_get will be called.

Call function for user data processing

```
static const struct bt_mesh_model_op gen_onoff_srv_op[] = {
    { BT_MESH_MODEL_OP_GEN_ONOFF_GET, 0, gen_onoff_get },
    { BT_MESH_MODEL_OP_GEN_ONOFF_SET, 2, gen_onoff_set },
    { BT_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK, 2, gen_onoff_set_unack },
    BT_MESH_MODEL_OP_END, };
```

ÿ. Vendor Model callback

The Vendor_Server_Model in the structure vendor_srv_op registers the callback vendor_srv_op to process user data

When a registration message such as VENDOR_MSG_ATTR_GET is received , the corresponding callback function such as vendor_attr_get will be called to perform

User data processing

```
static const struct bt_mesh_model_op vendor_srv_op[] = {
```

```
    { VENDOR_MSG_ATTR_GET, ACCESS_OP_SIZE, vendor_attr_get },
    { VENDOR_MSG_ATTR_SET, ACCESS_OP_SIZE, vendor_attr_set },
    BT_MESH_MODEL_OP_END, };
```

Chapter 4 OTA Instructions for Use

4.1 Overview

1. Test box **OTA** upgrade introduction

AC630N supports OTA upgrade of BLE or EDR link through Jerry's Bluetooth test box by default, which is convenient for customers to develop Firmware update for prototypes that are inconvenient for wired upgrades in the stage, or batch upgrades in the mass production stage. About Jerry Bluetooth Test Box Please refer to the document "AC690x_1T2 Test Box Instructions for Use.pdf" for the usage and related upgrade operation instructions.

In addition, the upgrade method can be directly jumped to the MASK interface to avoid the change of the IO state when the chip is powered off. It is suitable for the chip power supply. In the scheme of keeping the IO port KEEP, it is necessary to pay attention to check whether all hardware modules using DMA are closed before jumping.

Open the CONFIG_UPDATE_JUMP_TO_MASK configuration in the corresponding board configuration file xxx_global_build_cfg.h set:

```
2. #define CONFIG_UPDATE_JUMP_TO_MASK 1 // Configuration upgrade to loader The way0 for direct reset,1 For jumps applied to the chip power supply by IO mouth KEEP The live scheme needs to pay attention to check whether it will be used before jumping DMA the hardware module close all )
```

2. APP OTA upgrade introduction

AC630N can optionally support APP OTA upgrade, and the SDK provides a demo to complete OTA by interacting with APP through JL_RCSP protocol process. Customers can directly refer to the related documents of the JL_RCSP protocol and the description of the mobile APP OTA add-in library, and combine the APP OTA function set into the customer's own APP. The APP OTA function facilitates remote firmware push upgrades for products already on the market to fix known problems issues or support new features.

In addition, the upgrade method can be directly jumped to the MASKROM interface to avoid the change of the IO state when the chip is powered off. It is suitable for the chip. In the scheme where the power is kept by the IO port KEEP, it is necessary to check whether all the hardware modules using DMA are turned off before the jump. at the corresponding board level Open the CONFIG_UPDATE_JUMP_TO_MASK configuration in the configuration file xxx_global_build_cfg.h.

Note: At present, the APP upgrade supports BLE mode, but EDR does not currently support it.

3. Introduction of custom single/dual-line serial port upgrade

AC630N can optionally support the upgrade function of custom single/dual-line serial port. Set a single IO port to the sending state when the data is sent, and set it back to the receiving state after the data is sent to achieve a single-line Serial communication. When the SDK receives the data sent by the host computer that conforms to the Jerry serial port upgrade specification protocol, it will enter the upgrade process. most The new version of the writer can be used as the host computer for this kind of upgrade. The currently supported host computer is a 1-to-8 burning tool.

4.2 OTA - APP upgrade (BLE)

1. SDK project related configuration

1.1 Open CONFIG_APP_OTA_ENABLE in the corresponding board configuration file xxx_global_build_cfg.h

configure

```
/* Following Macros Affect Periods Of Both Code Compiling And Post-build */

#define CONFIG_DOUBLE_BANK_ENABLE 0 // If the single and double backup selection is turned on, change the macro FLASH The structure becomes a dual backup structure, which is suitable for accessing third-party protocols. OTA , PS: JL-OTA Also supports dual backup upgrade, need According to actual FLASH Configure the size at the same time CONFIG_FLASH_SIZE)

#define CONFIG_APP_OTA_ENABLE 1 // support RCSP upgrade(JL-OTA)

#define CONFIG_UPDATE_JUMP_TO_MASK 0 // Configuration upgrade to loader way of direct reset,1 For jumps applied to the chip power supply by IO mouth KEEP The live scheme needs to pay attention to check whether it will be used before jumping DMA the hardware module close all )
```

1.2 The generated upgrade file is update.ufw, put it in the file directory corresponding to the mobile APP, connect to Bluetooth,

After selecting the file, click Start to upgrade.

2. Mobile phone tools

2.1 Android side development instructions: For details, please refer to the Android_Jerry OTA external library development instructions in the tools directory

2.2 IOS side development instructions: For details, please refer to the IOS_Jili OTA external library development instructions in the tools directory

1. SDK project related configuration

1.1 Open the USER_UART_UPDATE_ENABLE configuration in app_config.h

```
1. #define USER_UART_UPDATE_ENABLE                    0 //Whether to support custom serial port upgrade
```

1.2 In app_config.h, configure the receiving/transmitting IO. If both receiving/transmitting are configured as the same IO port, the current

It is a single-wire serial port upgrade.

```
2. #define UART_UPDATE_RX_PORT          I_PORTA_02      //Set PA2 as the receiving port
3. #define UART_UPDATE_TX_PORT          IO_PORTA_03     //Set PA3 as the receiving port
```

1.3 If serial port upgrade is required between prototypes, you need to configure the current role, that is, configure master/slave,

If it is configured as a host, it needs to send data that conforms to the "Jerry Serial Port Upgrade Specification" protocol for the slave to perform the upgrade stream.

Procedure. AC63N is configured as a slave by default.

```
#define UART_UPDATE_ROLE          UART_UPDATE_SLAVE          //The macro corresponding to the host is UART_UPDATE_MASTER
```

1.4 The host computer currently supported by the SDK is the 1-to-8 programming tool (version 3.1.8 and above).

Chapter 5 AUDIO function

5.1 Overview

HID, SPP_AND_LE and MESH newly added the implementation sample code of AUDIO, need to use the AUDIO function to be in

The board-level configuration enables TCFG_AUDIO_ENABLE, as shown below:

```
1. //Support Audio function to enable DAC/ADC module
2. #ifdef CONFIG_LITE_AUDIO
3. #define TCFG_AUDIO_ENABLE                       ENABLE
4. #if TCFG_AUDIO_ENABLE
5. #undef TCFG_AUDIO_ADC_ENABLE
6. #undef TCFG_AUDIO_DAC_ENABLE
7. #define TCFG_AUDIO_ADC_ENABLE                   ENABLE_THIS_MOUDLE
8. #define TCFG_AUDIO_DAC_ENABLE                   ENABLE_THIS_MOUDLE
```

Support board level: br23, br25, br30, br34

Support chip: AC635N, AC636N, AC673N, AC638N

5.2 Use of **Audio**

1. DAC hardware output parameter configuration

There are the following configurations in the board-level configuration file,

```
1. /*
2. The connection method on the DAC hardware, optional configuration:
3.          DAC_OUTPUT_MONO_L                   left channel
4.          DAC_OUTPUT_MONO_R                   right channel
5.          DAC_OUTPUT_LR                       stereo
6.          DAC_OUTPUT_MONO_LR_DIFF             Mono differential output
7. */
8. #define TCFG_AUDIO_DAC_CONNECT_MODE                 DAC_OUTPUT_MONO_LR_DIFF
```

The TCFG_AUDIO_DAC_CONNECT_MODE macro needs to be configured according to the specific hardware connection method.

2. MIC configuration and use

2.1 Configuration Instructions

In each board.c file, there is a structure for configuring mic parameters, as shown in the following figure:

```
1. struct adc_platform_data adc_data = {
2.          .mic_channel            = TCFG_AUDIO_ADC_MIC_CHA,                   //MIC channel selection, for
    693x, MIC has only one channel, fixed selection of right channel
3. /*MIC LDO current gear setting:
4.          0:0.625ua       1:1.25ua        2:1.875ua       3: 2.5ua*/
5.          .mic_ldo_low = TCFG_AUDIO_ADC_LDO_SEL,
6. /*Whether the MIC saves the DC blocking capacitor:
7.          0: No capacitor saving 1: Capacitor saving*/
8. #if ((TCFG_AUDIO_DAC_CONNECT_MODE == DAC_OUTPUT_FRONT_LR_REAR_LR) || (TCFG_AUDIO_DAC
    _CONNECT_MODE == DAC_OUTPUT_DUAL_LR_DIFF))
9.          .mic_capless            = 0,//Four-channel and two-channel differential use, no need for capacitor connection
10. #else
11.         .mic_capless            = 0,
12. #endif
```

```
13. /*The MIC capacitor-free scheme needs to be set, which affects the bias voltage of the MIC
14.         21:1.18K    20:1.42K    19:1.55K    18:1.99K    17:2.2K    16:2.4K    15:2.6K
            14:2.91K    13: 3.05K   12:3.5K     11:3.73K
15.         10:3.91K    9:4.41K     8:5.0K      7:5.6K      6:6K       5:6.5K     4:7K
            3:7.6K      2:8.0K      1:8.5K                  */
16.         .mic_bias_res = 16,
17. /*MIC LDO voltage gear setting will also affect the MIC bias voltage
18.         0:2.3v 1:2.5v 2:2.7v 3:3.0v */
19.         .mic_ldo_vsel = 2,
20. /*MIC capacitor DC blocking mode uses internal mic bias*/
21.         .mic_bias_inside = 1,
22. /*Keep the internal mic bias output*/
23.         .mic_bias_keep = 0,
24.
25.         // ladc channel
26.         .ladc_num = ARRAY_SIZE(ladc_list),
27.         .ladc = ladc_list,
28. };
```

The main focus is on the following variables:

1) mic_capless: 0: select the capacitor-saving mode 1: select the capacitor-saving mode

2) mic_bias_res: It is only valid when the capacitor-saving mode is selected, the pull-up bias resistor of mic, the selection range is:

1:16K 2:7.5K 3:5.1K 4:6.8K 5:4.7K 6:3.5K 7:2.9K 8:3K 9:2.5K 10:2.1K 11:1.9K 12:2K 13:1.8K
14:1.6K 15:1.5K 16:1K 31:0.6K

3) mic_ldo_vsel: the bias voltage of mic_ldo, which determines the bias of mic together with the bias resistor. The selection range is: 0:2.3v 1:2.5v 2:2.7v 3:3.0v

4) mic_bias_inside: The mic external capacitor blocks DC, and the chip provides bias voltage. When mic_bias_inside=1, you can Use mic_bias_res and mic_ldo_vsel normally

2.2 Automatically calibrate the MIC bias voltage

When using the capacitor saving mode, you can configure TCFG_MC_BIAS_AUTO_ADJUST in app_config.h, select the MIC Auto calibration mode, automatically select the corresponding MIC bias resistor and bias voltage. Note: Insensitive capacitors cannot be calibrated. The configuration is as follows:

```
1. /*
```

```
2. *Capacitor-saving mic bias voltage automatic adjustment (because calibration takes time, so there are different ways)
3. *1. After burning the program (completely updated, including the configuration area), start the calibration once
4. *2. It is calibrated during power-on reset, that is, if the power is off and then powered on again, it will calibrate whether there is any deviation (default)
5. *3. It is calibrated every time it is turned on, regardless of whether the power is cut off or not, that is, the calibration process runs every time.
6.      */
7. #define MC_BIAS_ADJUST_DISABLE          0 //Capacitor-saving mic offset calibration is disabled
8. #define MC_BIAS_ADJUST_ONE              1 //The capacitor-saving mic offset is calibrated only once (same as dac trim)
9. #define MC_BIAS_ADJUST_POWER_ON         2   // Capacitor-saving mic bias is calibrated every power-on reset
       (Power_On_Reset)
10. #define MC_BIAS_ADJUST_ALWAYS           3 //The capacitor-saving mic bias is calibrated every time it is powered on (including power-on reset and its
       he resets)
11. #define TCFG_MC_BIAS_AUTO_ADJUST        MC_BIAS_ADJUST_POWER_ON
12. #define TCFG_MC_CONVERGE_TRACE          0 //Capacitor-saving mic convergence value tracking
```

2.3 Example of using Mic

You can call the audio_adc_open_demo(void) function to output the sound of mic. The example is as follows:

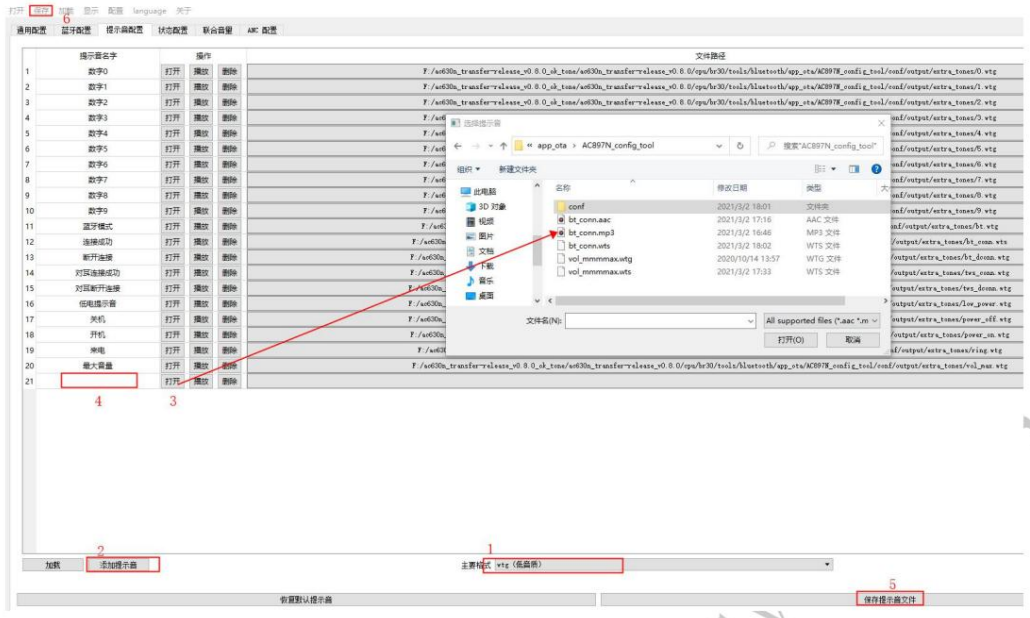
```
1. if (key_type == KEY_EVENT_LONG && key_value == TCFG_ADKEY_VALUE0) {
2.          printf(">>>key0:open mic\n");
3.          //br23/25 mic test
4.          extern int audio_adc_open_demo(void);
5.          audio_adc_open_demo();
6.          //br30 mic test
7.          /* extern void audio_adc_mic_demo(u8 mic_idx, u8 gain, u8 mic_2_dac); */
8.          /* audio_adc_mic_demo(1, 1, 1); */
9. }
```

3. The use of prompt tone

3.1 Prompt sound file configuration

1. Open the cpu\brxx\tools\ACxxxN_config_tool corresponding to the SDK, enter the configuration tool entry ---> select the configuration before compilation

Tools--->Prompt sound configuration.



2. Open the above interface and follow the steps to add the source file in *.mp3 format you need, and convert it into the main format you need. Pay attention to the file Path, the default path in the SDK may be different from the path saved locally, and it should be changed to the current absolute path of the SDK.

3. Add the tone.cfg configuration option in the download.bat download item of the ota directory.

4. To play the sin\wtg prompt, enable TCFG_DEC_G729_ENABLE and TCFG_DEC_PCM_ENABLE two macros, as shown below:

```
1. #if TCFG_AUDIO_ENABLE
2. #undef TCFG_AUDIO_ADC_ENABLE
3. #undef TCFG_AUDIO_DAC_ENABLE
4. #define TCFG_AUDIO_ADC_ENABLE                ENABLE_THIS_MOUDLE
5. #define TCFG_AUDIO_DAC_ENABLE                ENABLE_THIS_MOUDLE
6. #define TCFG_DEC_G729_ENABLE                 ENABLE
7. #define TCFG_DEC_PCM_ENABLE                  ENABLE
8. #define TCFG_ENC_OPUS_ENABLE                 DISABLE
9. #define TCFG_ENC_SPEEX_ENABLE                DISABLE
10. #define TCFG_LINEIN_LR_CH                   AUDIO_LIN0_LR
11. #else
12. #define TCFG_DEC_PCM_ENABLE                 DISABLE
13. #endif/*TCFG_AUDIO_ENABLE*/
```

And define the file name and path in the tone_player.h file: the file name and the name of the tone added in step 4 in the configuration tool should be the same. To.

```
14.#define TONE_NUM_0                      TONE_RES_ROOT_PATH"tone/0.*"
```

3.2 Example of use of prompt tone

You can call tone_play() to play the prompt tone. The usage example is as follows:

```
1. if (key_type == KEY_EVENT_LONG && key_value == TCFG_ADKEY_VALUE2) {
2.         printf(">>>key1:tone_play_test\n");
3.         //br23/25 tone play test
4.         /* tone_play_by_path(TONE_NORMAL, 1); */
5.         /* tone_play_by_path(TONE_BT_CONN, 1); */
6.         //br30 tone play test
7.         tone_play(TONE_NUM_0, 1);
8.         /* tone_play(TONE_SIN_NORMAL, 1); */
9.
10. }
```

4. Use of opus\speex encoding

4.1 Configuration Instructions

The opus\speex encoding module encodes the mic data. To use this function, it needs to be enabled in the board-level configuration file.

The two macros, TCFG_ENC_OPUS_ENABLE and TCFG_ENC_SPEEX_ENABLE, are configured as shown in the following figure:

```
1. #if TCFG_AUDIO_ENABLE
2. #undef TCFG_AUDIO_ADC_ENABLE
3. #undef TCFG_AUDIO_DAC_ENABLE
4. #define TCFG_AUDIO_ADC_ENABLE          ENABLE_THIS_MOUDLE
5. #define TCFG_AUDIO_DAC_ENABLE          ENABLE_THIS_MOUDLE
6. #define TCFG_DEC_G729_ENABLE           ENABLE
7. #define TCFG_DEC_PCM_ENABLE            ENABLE
8. #define TCFG_ENC_OPUS_ENABLE           ENABLE
9. #define TCFG_ENC_SPEEX_ENABLE          ENABLE
10. #define TCFG_LINEIN_LR_CH             AUDIO_LIN0_LR
11. #else
12. #define TCFG_DEC_PCM_ENABLE           DISABLE
13. #endif/*TCFG_AUDIO_ENABLE*/
```

4.2 opus\speex encoding example

```
int audio_mic_enc_open(int (*mic_output)(void *priv, void *buf, int len), u32 code_type);
```

Use the audio_mic_enc_open() function to encode the mic data with opus\speex , and the parameter mic_output is the encoded data

The output function, code_type is the type to be encoded, AUDIO_CODING_OPUS and AUDIO_CODING_SPEEX are optional, for example

The following figure:

```
1. /*encode test*/
2. extern int audio_mic_enc_open(int (*mic_output)(void *priv, void *buf, int len), u32
        code_type);
3. audio_mic_enc_open(mic_enc_output_data, AUDIO_CODING_OPUS);//opus encode test
4. /* audio_mic_enc_open(mic_enc_output_data, AUDIO_CODING_SPEEX);//speex encode test *
```

5. The use of general coding interface

5.1 Configuration Instructions

The general encoding interface is in the audio_codec_demo.c file. To use this interface, it needs to be enabled in the board-level configuration file.

The ENC_DEMO_EN macro is configured as shown below:

```
#define ENC_DEMO_EN                                    ENABLE;
```

5.2 To create an encoding just call the following audio_demo_enc_open function

```
int audio_demo_enc_open(int (*demo_output)(void *priv, void *buf, int len), u32 code_type, u8
ai_type)
```

The first parameter is the externally registered encoding output callback, register this callback, and you can get the encoded post-data in this callback.

The second parameter is the encoding type, and the corresponding encoding is selected according to the incoming parameters. The optional encodings include OPUS encoding, SPEEX encoding, AD

PCM encoding, LC3 encoding, SBC encoding and MSBC encoding, creating corresponding encoders need to be enabled in the board file respectively

TCFG_ENC_OPUS_ENABLE,TCFG_ENC_SPEEX_ENABLE,TCFG_ENC_ADPCM_ENABLE

TCFG_ENC_SBC_ENABLE, TCFG_ENC_SBC_ENABLE, MSBC encoding is enabled by default, no macro control

```
#define TCFG_ENC_OPUS_ENABLE                    ENABLE
#define TCFG_ENC_SPEEX_ENABLE                   ENABLE
#define TCFG_ENC_LC3_ENABLE                     ENABLE
#define TCFG_ENC_ADPCM_ENABLE                   ENABLE
#define TCFG_ENC_SBC_ENABLE                     ENABLE
```

The third parameter is the parameter of speex encoding. You can pass 0 or 1 as needed. Different encoding parameters are modified in

You can modify the value of the fmt structure inside the audio_demo_enc_open function. Take the adpcm parameter as an example, as follows

```
case AUDIO_CODING_WAV:
fmt.sample_rate = 16000;
fmt.bit_rate = 1024; //blockSize, can be configured as 256/512/1024/2048
fmt.channel = 2;
fmt.coding_type = AUDIO_CODING_WAV;
break;
```

Modify the value of the fmt structure member according to the actual encoding parameters

A timer is opened by default in the audio_demo_enc_open function. The timer writes the source data to be encoded to the decoder, and the timer

The device executes the following functions:

```
static void demo_frame_test_time_fundc(void *parm)
```

Just write the source data in this function, the code writes the sine wave data to encode by default

5.3 To close the encoding, just call the following audio_demo_enc_close function

```
int audio_demo_enc_close()
```


5.3 Use of **Audio_MIDI**

5.3.1 File Download Configuration

1. Add the corresponding midi.bin file under SDK\cpu\br23\tools, and add the download item under download.bat in the same level directory:

```
isd_download.exe -tonorflash -dev br23 -boot 0x12000 -div8 -wait 300 -uboot uboot.boot -app app.bin

cfg_tool.bin -res midi.bin %1
```

Add to the isd_config.ini file in the same directory as download.bat:

```
INTERNAL_DIR_ALIGN=0X2; //The starting address of the file in the flash directory is 4 aligned
```

2 ÿ In the corresponding SDK\apps\spp_and_le\board\br23\board_ac635n_demo_cfg.h file, enable AUDIO

Function:

```
#define TCFG_AUDIO_ENABLE                       1//DISABLE

#define AUDIO_MIDI_CTRL_CONFIG                  1 //Midi keyboard interface is enabled, open this macro to turn off the low power enable
```

Turn off low power mode:

```
#define TCFG_LOWPOWER_LOWPOWER_SEL              0//SLEEP_EN                    //SNIFF status

Whether the next chip enters powerdown#if TCFG_USER_BLE_ENABLE
```

3. Make the following settings in SDK/cpu/br23/audio_dec/audio_dec_midi_ctrl.c: The specific function content is subject to the SDK.

```
    void midi_paly_test(u32 key)

{

        static u8 open_close = 0;

        static u8 change_prog = 0;

        static u8 note_on_off = 0;

        switch (key) {

        case KEY_IR_NUM_0:

                if (!open_close) {

                        /* midi_ctrl_dec_open(16000);//Start midi key */

                        //midi_ctrl_dec_open(16000, "storage/sd0/C/MIDI.bin\0");//Start midi key SD card

                        midi_ctrl_dec_open(16000,SDFILE_RES_ROOT_PATH"MIDI.bin\0");//Start midi key system



                } else {midi_ctrl_dec_close();//ÿÿ midi key
```

```
            }open_close = !open_close;
            break;
        case KEY_IR_NUM_1:
            if (!change_prog) {
                midi_ctrl_set_porg(0, 0);//Set instrument 0, track 0

            } else {midi_ctrl_set_porg(22, 0);//Set instrument 22, track 0
            }
            change_prog = !change_prog;
            break;
        case KEY_IR_NUM_2:
            if (!note_on_off) {
                //Simulate buttons 57, 58, 59, 60, 61, 62, with force 127, channel 0, press the test
                ...
            } else
                //simulate button 57, 58, 59, 60, 61, 62 release test
               ...

            }note_on_off = !note_on_off;
            break;
        default:
            break;
        }
}
```

4. Call the midi_play_test() function in app_spp_and_le.c to play the corresponding file:

```
static void app_key_event_handler(struct sys_event *event)
{
        ......
#if TCFG_AUDIO_ENABLE
if (event_type == KEY_EVENT_CLICK && key_value == TCFG_ADKEY_VALUE0) {
```



```
            /*midi test*/
        printf(">>>key0:open midi\n");
          midi_paly_test(KEY_IR_NUM_0);
            }
if (event_type == KEY_EVENT_CLICK && key_value == TCFG_ADKEY_VALUE1) {


                printf(">>>key0:set midi\n");
                midi_paly_test(KEY_IR_NUM_1);


            }
            if (event_type == KEY_EVENT_CLICK && key_value == TCFG_ADKEY_VALUE2)
                printf(">>>key2:play midi\n");
                midi_paly_test(KEY_IR_NUM_2);
            }
```

Chapter 6 lighting handshake charging function

6.1 Overview

Support can use the lighting cable to charge the device. When the lighting is inserted, first shake hands with the lighting line to make it Output strong electricity, and then turn on the built-in charging function to charge the device. Lighting charging supports ordinary Apple cable charging

It can be charged with electricity and Apple fast charging cable, and it supports positive and negative plug-in power supply. The specific amount of mA power supply is related to different board levels.

1. SDK project related configuration

1.1 In the board-level configuration file board_xxx_demo_cfg.h, enable the charging configuration and whether it supports lighting grip

In addition, the IO port configuration of protocol communication can be customized. The enabling configuration is as follows:

```
1. //Whether the chip supports built-in charging
2. #define TCFG_CHARGE_ENABLE                    ENABLE_THIS_MOUDLE
3. //Whether it supports power-on charging
4. #define TCFG_CHARGE_POWERON_ENABLE DISABLE
5. //Whether it supports the automatic power-on function of unplugging and charging
6. #define TCFG_CHARGE_OFF_POWERON_NE DISABLE
7. //Whether to support lighting handshake protocol
8. #define TCFG_HANDSHAKE_ENABLE                 ENABLE
9. #define TCFG_HANDSHAKE_IO_DATA1               IO_PORTB_02//Handshake IO near the middle of the lighting seat
10. #define TCFG_HANDSHAKE_IO_DATA2               IO_PORTB_07//Handshake IO on the side of the lighting seat
```

1.2 Main code description:

1.2.1 lighting charging initialization (handshake_app_start) , and judging whether to charge

(get_charge_online_flag()) Set different power supply methods in board_init of board_xxx_demo_cfg.c

Inside, the corresponding code is as follows:

```
1. void board_init()
2. {
3. #if TCFG_CHARGE_ENABLE && TCFG_HANDSHAKE_ENABLE
4.         if(get_charge_online_flag()){
5.             handshake_app_start(0, NULL);
6.         }
7. #endif
8.
9. if(get_charge_online_flag()) {
10. power_set_mode(PWR_LDO15);
11. } else {
```

```
12. power_set_mode(TCFG_LOWPOWER_POWER_SEL);
13. }
14. }
```