珠海市杰理科技有限公司
ZHUHAI JIELI TECHNOLOGY CO.,LTD

AC692X_SDK_Introduction

AC692X_SDK_Introduction User Manual

Rev 1.5 —— September 5, 2018

Table of contents

Changelog

Version	date	describe
1.5	2018 / 9 /05	Added BLE transparent transmission function description
1.4	2018 / 8 /22 Added recording instructions	
1.3	2018 / 8 /11 Added serial port upgrade protocol description	
1.2	2018 / 5 /18 Added Reverb and TWS chapters	
1.1	2018 / 4 /08	Add VM usage notes
1.0	2018 / 3 / 30	AC692x User Manual
renew:	ÿ Create the initial version ÿ Define document format	

Chapter 1 SDK Development Kit Quick Start Guide

1.1 Purpose

This document mainly describes the usage of AC692x_SDK development kit and some issues to pay attention to during development, and provides users with secondary development References are provided, including:

Introduction to development environment setup, IDE installation steps, authentication methods, and other pre-development preparations;

Directory structure, project structure introduction, files that users are allowed to modify and files that users are not allowed to modify;

1.2 Installation of IDE development tools

ÿ Compilation environment installation

ÿ Toolchain installation

To develop AC692x, you must reinstall the toolchain:

1. codeblocks-16.01mingw-setup, same as AC690x

2ÿjl_toolchain_pi32v2_lto_2.1.6.exe

ÿ Installation tutorial:

jl_toolchain_pi32v2_lto_2.1.6

The tool needs to be registered, otherwise an error will occur in the linking step, reporting "No Such File"

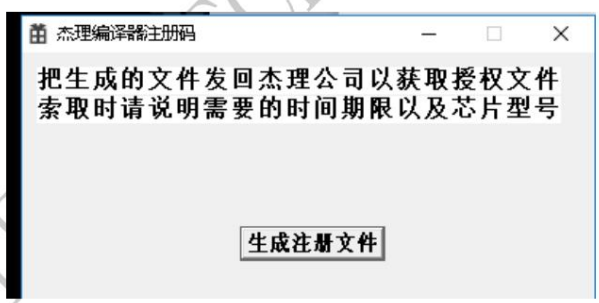
ÿ Registration tools:

ÿ Generate key file

In order to register, you first need to generate a key file and feed it back to Jie Li Technology

Start --> Programs --> JL toolchain --> Generate License Key File --> "Generate Registration File"

Then save the key file and send it to Jieli Technology, and explain the project you need to use (AC54, AC691X, AC692X etc.) or backend (PI32, PI32V2, Q32S)



ÿ Import lic file

Start --> Programs --> JL toolchain --> Import License file --> "Open License file"

Then select the obtained lic file

1.3 Opening the AC692x_SDK project

1. Before opening the project, make sure you have installed the latest IDE and toolchain released by Jerry Technology

2. Double-click the ac69_app.cbp project file in the apps/ directory, or drag the ac69_app.cbp file to the CodeBlocks shortcut.

Open in shortcut mode, the AC692x SDK project will be opened

Note: If CodeBlocks fails to compile after installing it for the first time, you may need to specify a compiler (if it compiles normally,

No need to specify), method: right-click the project name -> build options -> Compiler settings, as shown below:

1. Before opening the project, make sure you have installed the latest IDE and toolchain released by Jerry Technology, because AC692x has

Some new compilation features are added, so you must first install the compiler version after (jl_toolchain_pi32v2_lto_2.1.6.exe)

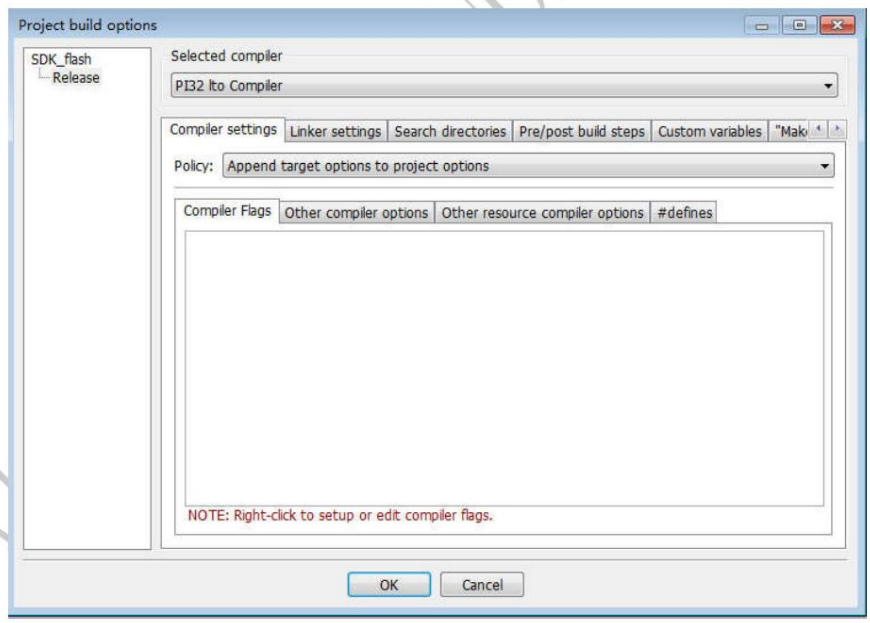
Can be used normally, otherwise the SDK can be compiled normally but will execute incorrectly.

2. Double-click the sdk.cbp project file in the apps/ directory, or drag the sdk.cbp file to the CodeBlocks shortcut.

Open in the SDK project mode, the SDK project will be opened

Note: If you install CodeBlocks for the first time and it fails to compile, you may need to specify a compiler (

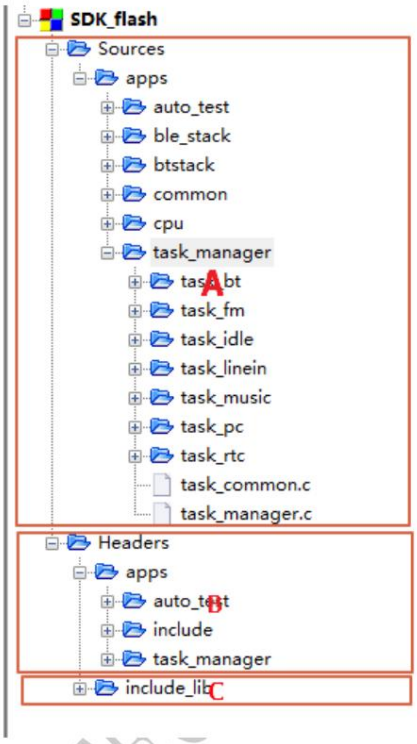
Normal, no need to specify), method: right-click the project name -> build options -> Compiler settings, as shown below:



After making your selection, recompile.

1.4 Project Directory Introduction

After the project is opened, the project directory is as follows:



Screenshot A area: some .c files that users can see and modify, and users can also add their own c files

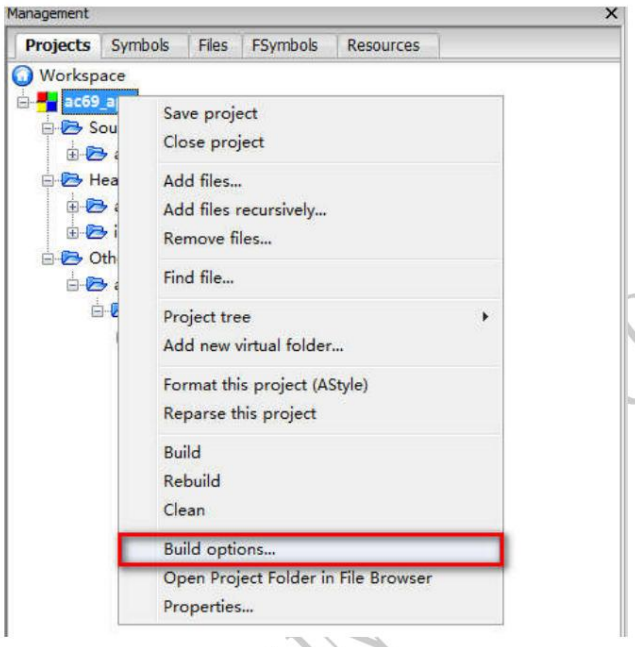
Screenshot B area: The apps directory provides header files for external applications, which users can modify at will

Screenshot C area: include_lib provides the header files used by the library, which is closely related to the compilation of the library. Customers should try not to modify the header files in it. content

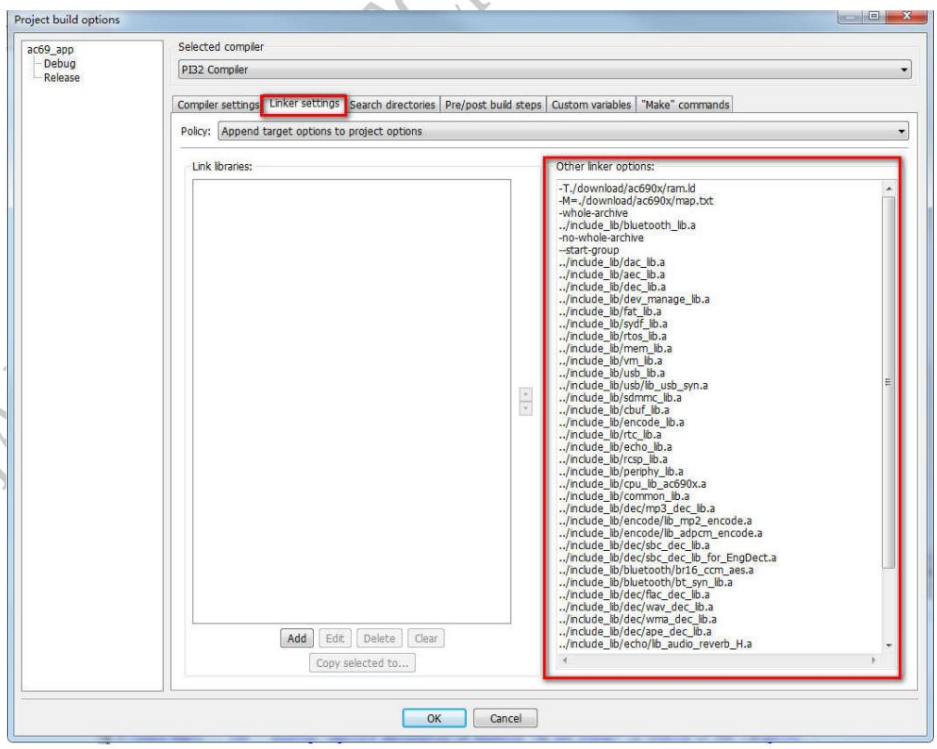
The Map.txt file in the apps\download\post_build\flash directory is updated every time the compilation is completed. The content is mainly the project letter The mapping relationship between numbers and some variables, sdk.ld mainly shows the usage of ram

1.5 Some common settings instructions

Select the project and right-click to set it as shown below:



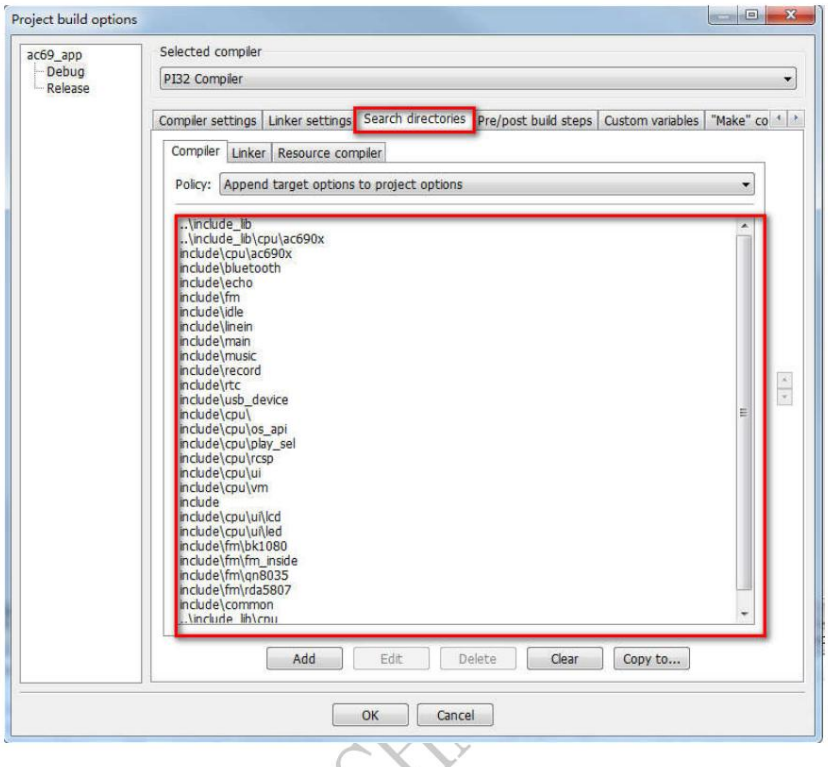
After clicking the red border, the following setting interface will appear. Select Linker settings to set the library file:



Note: When adding library files (changing library files), you need to rebuild. Just modify the .c or .h files and execute the build.

The red box on the right is the added library file

Select Search directories to set the path:



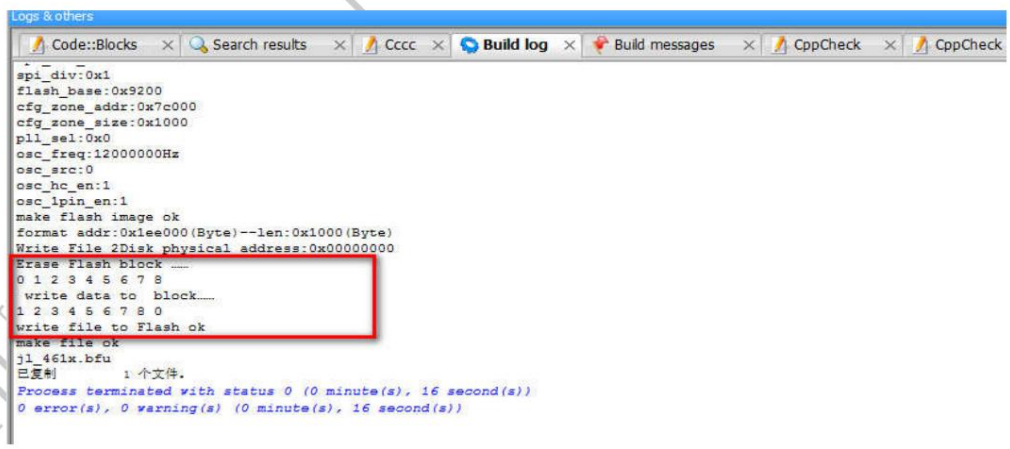
1.6 Program Download Instructions

The download.bat file is a batch file for downloading, corresponding to the configuration, (Part A is responsible for downloading the program to flash, and Part B is

Responsible for generating burning files and upgrade files (bfu):

```
cd %~dp0

if exist uboot.boot del uboot.boot
type uboot.bin > uboot.boot

cd tone_resource
copy *.mp3 ..\
cd ..

isd_download.exe -tonorflash -dev br21 -boot 0x2000 -div6 -wait 300  -f uboot.boot sdk.app bt_cfg.b
in  bt.mp3 music.mp3  linein.mp3 radio.mp3 pc.mp3 connect.mp3 disconnect.mp3 ring.mp3 warning.mp3
0.mp3 1.mp3 2.mp3 3.mp3 4.mp3 5.mp3 6.mp3 7.mp3 8.mp3 9.mp3


::-format cfg

:: -read flash_r.bin 0-2M

if exist *.mp3 del *.mp3
if exist *.PIX del *.PIX
if exist *.TAB del *.TAB
if exist *.res del *.res
if exist *.sty del *.sty
if exist jl_692x.bin del jl_692x.bin


rename jl_isd.bin jl_692x.bin
bfumake.exe -fi jl_692x.bin -ld 0x0000 -rd 0x0000 -fo updata.bfu


IF EXIST no_isd_file del jl_692x.bin
del no_isd_file

@rem format vm        //擦除VM 68K区域
@rem format cfg       //擦除BT CFG 4K区域
@rem format 0x3f0-2  //表示从第 0x3f0 个 sector 开始连续擦除 2 个 sector(第一个参数为16进制或10进制
都可，第二个参数必须是10进制)

ping /n 2 127.1>null
::pause
```

Finally, it will show "write file to Flash ok", indicating the download is successful.



Note: After the compiler is compiled, you can download directly. Before downloading, you need to reset the flash and power it on (press the flash reset button on the development board).

power on, or press the flash reset button and then press the Reset button to reset), the drive letter pops up on the PC before downloading. After the download is complete, please

Power off restart

Chapter2 Upgrade Instructions

2.1 Download and upgrade instructions

692x series download update download program, including: PC online upgrade, U disk/TF card upgrade, PC forced upgrade,

Test box Bluetooth wireless upgrade. [Note] Flash partition: code area, configuration area, vm area.

ÿ Generate the upgrade code upgrade file updata.bfu. As shown in the figure below, AC692x\flash\download.bat generates the file to be upgraded.

updata.bfu, this upgrade file can update the relevant code area.

```
isd_download.exe -tonorflash -dev br21 -boot 0x2000 -div6 -wait 300  -

::-format cfg

:: -read flash_r.bin 0-2M

if exist *.mp3 del *.mp3
if exist *.PIX del *.PIX
if exist *.TAB del *.TAB
if exist *.res del *.res
if exist *.sty del *.sty
if exist jl_692x.bin del jl_692x.bin


rename jl_isd.bin jl_692x.bin
ufumake.exe -fi jl_692x.bin -ld 0x0000 -rd 0x0000 -fo updata.bfu
```

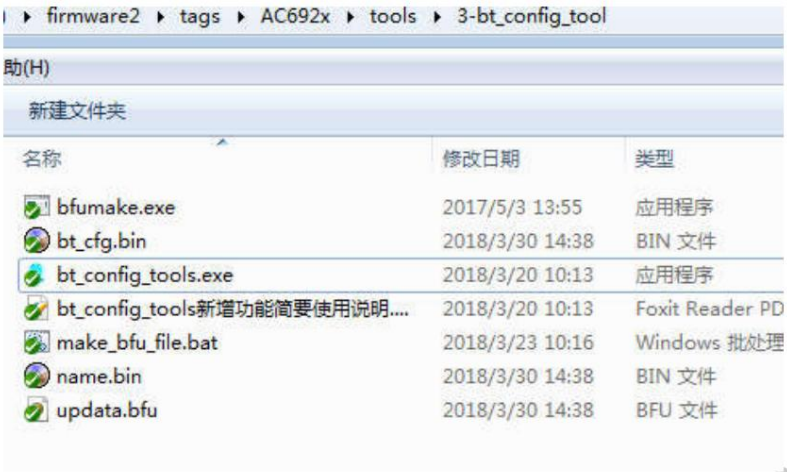
Both burning bin files and upgrading update.bfu files must be generated when the prototype or development board is not connected to the computer.

That is, offline generation!

ÿ Generate updata.bfu to upgrade the Bluetooth name separately. As shown in the figure below, use the tool bt_config_tool.exe to modify the Bluetooth that needs to be upgraded.

Then double-click the batch process make_bfu_file.bat to generate the upgrade file updata.bfu for upgrading the Bluetooth name.

(This upgrade file is upgraded by erasing the flash configuration area and then writing the relevant information into the configuration area)



ÿ Upgrading requires erasing the flash configuration area and VM. For example, erasing the flash configuration area requires updating the configuration information. . Revise AC692x\flash\isd_tools.cfg is shown in Figure 1. The last generated updata.bfu will contain the information of erasing the flash configuration area.

Erase when upgrading.

```
########flash空间使用配置区域#####################################################
#PDCTNAME:    产品名，对应此代码，用于标识产品，升级时可以选择匹配产品名
#BOOT_FIRST:  1=代码更新后，提示APP是第一次启动；0=代码更新后，不提示
#UPVR_CTL:    0：不允许高版本升级低版本   1：允许高版本升级低版本
#XXXX_ADR:    区域起始地址       AUTO：由工具自动分配起始地址
#XXXX_LEN:    区域长度           CODE_LEN：代码长度
#XXXX_OPT:    区域操作属性
#操作符说明  OPT:
#       0:  下载代码时擦除指定区域
#       1:  下载代码时不操作指定区域
#       2:  下载代码时给指定区域加上保护
#################################################################################

SPECIAL_AREA_START;
{
PDCTNAME=jl_692X;
BOOT_FIRST=1;
UPVR_CTL=0;
PRCT_ADR=0;
PRCT_LEN=CODE_LEN;
PRCT_OPT=2;
BTIF_ADR=AUTO;
BTIF_LEN=0x1000;
BTIF_OPT=0;
VMIF_ADR=AUTO;
VMIF_LEN=0x10000;
VMIF_OPT=1;
}
```

#Operator Description OPT:

0: Erase the specified area when downloading code

1: Do not operate the specified area when downloading code

2: Add protection to the specified area when downloading code

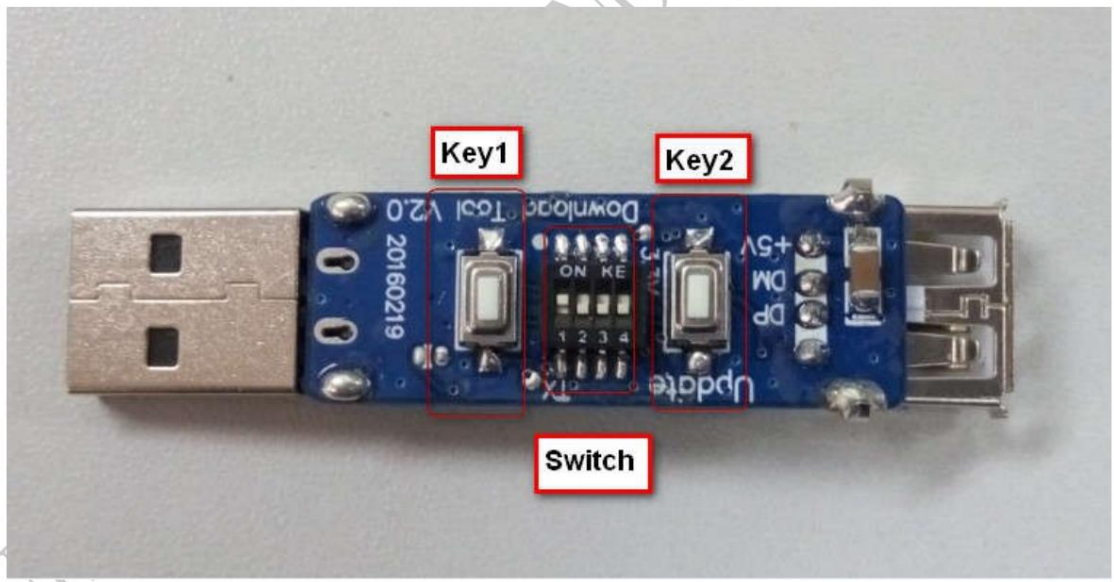
ÿ **PC** mandatory upgrade

ÿ Use our customized USB tool, press the "update" button, the blue light is on, and insert the USB upgrade tool, the blue light goes out, you can enter the upgrade mode, click batch process AC692x\flash\download.bat Download. This upgrade mode is mainly used when the chip cannot run or the wrong program is burned!

ÿ The upgrade tool uses a USB interface. During normal use, the female port is connected to the computer and the male port is connected to the USB port of the prototype. Version V2.0 adds a dip switch to facilitate subsequent upgrades and compatibility with older versions.

Dip switch	Working Mode	illustrate
0-0-0-0	AC460X	Used to upgrade AC460X series chips
1-0-0-0	AC69XX_USB slave upgrade	Used to upgrade AC9XX series chips
Other	reserve	reserve

As shown in the following figure:



Upgrade Tool

ÿ After the tool is connected to the USB cable, it defaults to normal mode and does not perform any operation, which is equivalent to a USB extension cable.

ÿ Upgrade steps

1. First, completely power off the prototype.
2. Connect the tool's female port to the computer via a USB cable and wait for LED1 to light up.
3. Press key2, and the upgrade indicator LED2 will light up. (If LED2 does not light up after pressing key2, please check the hardware or

Retry step 2 after power failure)

4. Insert the upgrade tool male port into the USB of the prototype, turn on the prototype, and the prototype will enter the upgrade mode .

If the computer boots normally but cannot enter the upgrade mode, please try step 1 again.

5. After successfully entering the upgrade mode, recompile the code and it will be automatically upgraded and downloaded to the prototype.

ÿ **PC** online upgrade

ÿ First, the prototype program needs to have the PC slave function, then when it is turned on, connect it to the PC and enter the PC upgrade mode.

Click batch download AC692x\flasht\download.bat. Upgrading requires erasing the flash configuration area and VM. Refer to the description above.

to configure.

ÿ **U disk/TF** card upgrade

ÿ First, the prototype program needs to have the function of inserting a USB disk or a TF card. Then, when the machine is turned on, enter the music mode.

Detecting the update.bfu file in the USB disk or TF card, enter the upgrade mode. The upgrade requires erasing the flash configuration area and VM reference

Configure as described above.

ÿ Test box Bluetooth wireless upgrade

ÿ First, the prototype program needs to be in Bluetooth mode to connect, and then turn on the machine.

Insert the TF card of update.bfu into the Jerry one-to-two test box, then use the test box to connect the prototype and enter the upgrade mode for upgrade.

(The test box needs to be updated to AC690x_1T2 test box V1.0.9 to support opening updata.bfu for upgrading).

For reference, please refer to (AC690x_1T2 Test Box User Manual.pdf)

U disk/TF card/Bluetooth wireless upgrade The upgrade is completed. The user can modify **the SDK** app/ cpu /**updata.c, as shown below**

As shown, the upgrade is complete, the light will turn on, and there will be a key tone. If the upgrade fails, there will be an alarm.

It is necessary to test relevant upgraded functions to prevent irreparable program errors during mass production.


```
void update_result_deal()
{
    u8 key_voice_cnt = 0;
    u16 result = 0;
    result = (g_updata_flag & 0xffff);
    printf("<---------update_result_deal=0x%x--------->\n", result);
    if (result == UPDATA_NON) {
        return;
    }
#ifdef UPDATE_VOICE_REMIND
    set_sys_vol(30, 30, FADE_ON);
#endif
#ifdef UPDATE_LED_REMIND
    if (result == UPDATA_SUCC) {
        led_update_finish();
    }
#endif
    while (1) {
        clear_wdt();
        key_voice_cnt++;
#ifdef UPDATE_VOICE_REMIND
        if (result == UPDATA_SUCC) {
            puts("<<<<<<UPDATA_SUCC");
            sin_tone_play(500);
            delay_2ms(500);
            puts(">>>>>>>>>>>\n");
        } else {
            printf("!!!!!!!!!!!!!!!updata waring !!!!!!!!!!=0x%x\n", result);
            sin_tone_play(1000);
            delay_2ms(500);
        }
#endif
        if (key_voice_cnt > 5) {
            key_voice_cnt = 0;
            delay_2ms(500);
            puts("enter_sys_soft_poweroff\n");
            enter_sys_soft_poweroff();
        }
    }
}
```

Chapter3 VM Usage Instructions

3.1 VM Overview

ÿ Overview:

VM (virtual memory) is a virtual memory system, mainly used to save information to flash devices.

The memory device has a limited number of read times, and the system can balance the wear of the memory device and extend the life of the device.

The basic use of VM is mainly divided into the following four steps:

ÿ Initialize VM: For any VM operation, it is necessary to ensure that the VM has been initialized.

ÿ Apply for VM: After applying for VM, obtain VM handle, which provides VM read and write functions.

ÿ Read VM: Operate through VM handle.

ÿ Write VM: Operate through VM handle.

The minimum total capacity of a VM is 8K, and the maximum total capacity is 128K. This is configured by modifying the isd_tools.cfg file.

3.2 Basic Use of VM

ÿ Apply for **VM:**

function prototype	void vm_open_all(void)
Function describe	Apply for vm storage space of length data_len
parameter illustrate	index: the serial number of the VM area to be applied for data_len: the size of the requested VM area
Returns the	handle of the VM area requested by the operation

void vm_open_all(void);

To use VM storage, you must first open it, using vm_open or vm_open_all. Note that:

When opening the new version of VM, there is no need to pass a length parameter.

```
enum {
    VM_REMOTE_DB = 1,
    VM_REMOTE_DB_END = (VM_REMOTE_DB + 20),

    //vm_start index
    VM_SYS_VOL,
    VM_SYS_EQ,
    VM_DEV0_BREAKPOINT,
    VM_DEV1_BREAKPOINT,
    VM_DEV2_BREAKPOINT,
    VM_DEV3_BREAKPOINT,
    VM_MUSIC_DEVICE,
```



```
    VM_PC_VOL,
    VM_FM_INFO,
    VM_PHONE_VOL,
    VM_BT_STEREO_INFO,
    VM_BT_OSC_INT_R,
    VM_BT_OSC_INT_L,

    //------Please add a new VM item under the sub-dividing line-----//
    VM_TEST,

    VM_MAX_INDEX,
};
```

ÿ **VM** read:

function prototype	vm_err vm_read(u8 index, void * data_bufÿu16 len)
Function describe	Read VM
parameter illustrate	index: When applying for vm, the corresponding index enumeration variable table *data_buf: read buff pointer
Return len: Success ret < 0: error	

Usage examples:

```
err = vm_read(vm_hdl_vol,&sys_info_var.vol, 1);
```

ÿ **VM** writes:

function prototype	vm_err vm_write(u8 index ,const void *data_buf, u16 len)
Function describe	Write VM
parameter illustrate	index: When applying for vm, the corresponding index enumeration variable table *data_buf: pointer to the data to be written
Return len: Success ret < 0: error	

Usage examples:

```
err = vm_write(vm_hdl_vol,&sys_info_var.vol);
```

Note about VM read and write operations on AC692X:

This version of VM requires a length parameter for reading and writing. Each VM only stores the last valid data.

If the read/write length is greater than the internal data length, only the valid length is returned. If the read/write length is less than the internal access length, the read/write length is returned.

VM and DAC:

DAC can continue to work while the VM is reading/defragmenting

In the vm_init_api function, the parameter is 0: DAC is not allowed to work during vm operation, and the parameter is 1: DAC is allowed during vm operation.

Work

It should be noted that if DAC is allowed to work, all functions or constants called in dac_isr_cb need to be placed in audio_text

Segment (use AT_AUDIO definition)

Chapter4 AUX Mode

4.1 Overall Design

ÿ System:

This description is mainly based on the SDK development kit to achieve the AUX function.

The main functions of the AUX application include:

ÿ Aux input channel selection

aux_adc conversion

ÿ Overall architecture design:

The AUX task is mainly divided into two functional modules:

ÿ Input channel selection: Aux input includes amux0, amux1, amux2 and dac_amux. Each channel has

Only one input of the channel can be used, which is suitable for IC packages with fewer IOs.

One of the two output channels is used as input and the other as output.

ÿaux_adc conversion: This function is used to realize the digital conversion of aux analog input. It can realize energy value sampling.

Sample, and sample the aux input for ad, and then do digital sound processing.

ÿApplication startup

There are two ways to enter aux mode:

ÿInsert the aux data cable and jump into aux mode, provided that the aux insertion detection function is turned on

ÿ Switch the mode button to enter the aux mode directly, provided that the aux insertion detection function is turned off

ÿ Exit of application:

When the Mode key is pressed, the aux data cable is unplugged, or other tasks need to be activated, the main thread will forcefully exit aux.

mode, it will release Aux task resources and turn off the prompt sound

Chapter5 Bluetooth Certification Description

5.1 FCC Certification Description

ÿ **FCC** certification configuration introduction:

ÿ The macro for opening the Bluetooth test mode is introduced in the file Bluetooth_api.h as shown in Figure 1 below

```
#define NORMAL_MODE          0
#define TEST_BQB_MODE        1       ///<测试bqb认证
#define TEST_FCC_MODE        2       ///<测试fcc认证
#define TEST_FRE_OFF_MODE    3       ///<测试频偏(使用频谱分析仪-手提测试仪-中心频率默认2422M)
#define TEST_PERFOR_MODE     4       ///<指标性能测试(使用MT8852A仪器测试,测试芯片性能的时候使用)
#define TEST_BOX_MODE        5       ///<测试盒测试
```

Figure 1

ÿ To enable FCC certification, you need to configure the BT_MODE macro to TEST_FCC_MODE in sdk_cfg.h, as shown below . (Note that after enabling this function, you need to pull the USB port high to start the PC online upgrade, and download.bat needs to use the -erase configuration to erase all flash.)

```
#define BT_MODE     TEST_FCC_MODE
```

Figure 2

ÿ The fcc certification of SDK uses USB port (DP is TX \DM is RX) as the UART serial port and power supply by default. Computer communication (configured in uart.c as shown in Figure 3 below). In order to ensure normal communication between the serial port and the computer, all programs using the USB port need to be shielded in sdk_cfg.h (as shown in Figure 4 below). Pay attention to the packaging of each chip. If the USB port is tied to other pins, set the pin to high impedance, otherwise it may Affects the communication between the serial port and the computer. If the user wants to use other hardware serial ports, he can modify the corresponding hardware settings in the void fcc_uart_init() function of the uart.c file.


```c
void fcc_uart_init()
{
    u32 status;
    puts("-----fcc_uart_init\n");
    fcc_uart_handle=NULL;
    status=uart_module_open(&uart1_handle,UART1_HARDWARE_NAME);
    if(!status)
    {
        __uart_param fcc_param;
        memset(&fcc_param,0,sizeof(__uart_param));
        fcc_param.baud_rate=9600;
        fcc_param.io= UART_USBP_USBM;
        fcc_param.workmode=UART_WORKMODE_NORMAL;
        fcc_param.custom |= (BIT(14)|BIT(3));
        status=uart_module_init(&uart1_handle,&fcc_param);
        uart_reg_isr_callback_fun(&uart1_handle,5,fcc_uart_isr_callback);
        if(status)
        {
            puts("uart_module_init err\n");
        }
        if(!status)
        {
            status=uart_module_start(&uart1_handle);
            if(!status)
                fcc_uart_handle=&uart1_handle;
        }

    }
}
```

Figure 3

```
*/
#ifdef MINI_BT
    #define SDMMC0_EN       1
    #define SDMMC1_EN       0
    #define USB_DISK_EN     0
    #define USB_PC_EN       0
#else
    #define SDMMC0_EN       1
    #define SDMMC1_EN       1
    #define USB_DISK_EN     0
    #define USB_PC_EN       0
#endif
```

Figure 4

ÿ **FCC** certification **PC** tool **FCCAssist_1.5.exe** introduction:

ÿ According to the system type of the computer, first open the REG_WINDOWS_UART folder, click register_uart_WIN7.bat or register_uart_WINXP.bat batch processing to follow the library file; ÿ Open the software FCCAssist_1.5.exe, click the serialport selection drop-down box, select the correct serial port, if it is normal, the red mark next to it will turn green, as shown in Figure 5;

ÿ During the test, set the corresponding configuration items as needed, and then press the send configuration button. If the sending is successful, the box in the lower left corner indicates that the sending is successful. Otherwise, the sending is unsuccessful (as shown in Figure 5). Show).

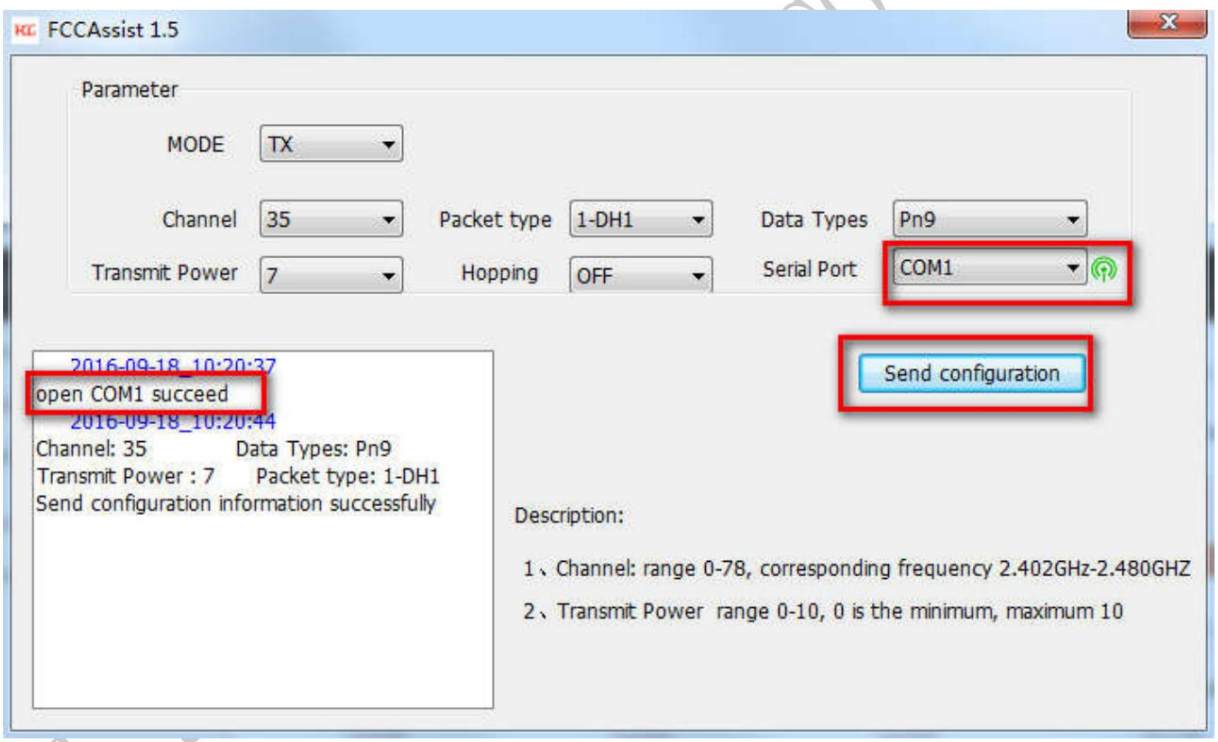


Figure 5

Note: By default, the SDK uses the USB port as the serial port to communicate with the computer. If the port is disconnected from the serial port board, The system will detect the USB online status when it starts up) which will cause the machine to not start normally, so you need to check if the machine starts normally. Then connect the USB port to the serial port board to communicate with the computer.

5.2 BQB

ÿ **BQB** configuration introduction

ÿTo enable BQB authentication, you need to configure the BT_MODE macro to TEST_BQB_MODE in sdk_cfg.h, as shown in Figure 6 shown.

```
#define BT_MODE    TEST_BQB_MODE
```

Figure 6

ÿ **BQB RF** test

ÿ Open TEST_BQB_MODE macro, the prototype can enter the dut test, and the mobile phone can search for the configured bluetooth after power on.
Tooth name, the mobile phone cannot connect to it, it is only for BQB test connection.

ÿ The antenna end of the prototype needs to be welded with a shielding wire, but the material of the antenna end does not need to be modified for now.

ÿ Go to the laboratory and connect the prototype and test instrument with shielded cables, then power on and conduct the connection test.

ÿ There is a setting item center frequency is 2M.

ÿ When testing RF, some items may require special processing before they can pass. Try to pass all test items first and then
Check the report to see which items have problems and then see how to deal with them.

ÿ **BQB profile** test

ÿ Turn on NORMAL_MODE mode, the prototype antenna does not need to be soldered with shielding wire;

ÿ Select the test items according to the customer's prototype functions. This test mainly tests functions such as up and down songs, pause, answer, hang up,
Backlinks, etc.

Chapter6 Bluetooth Development Instructions

6.1 Terms and abbreviations

Tip: List the definitions of specialized terms used in this document and the original phrases of foreign initials.

Abbreviations and terms	explain
AC692x	Jerry Technology AC692x series chips
1 to 1	Bluetooth only supports one mobile phone connection

ÿ Open the **AC692X** project:

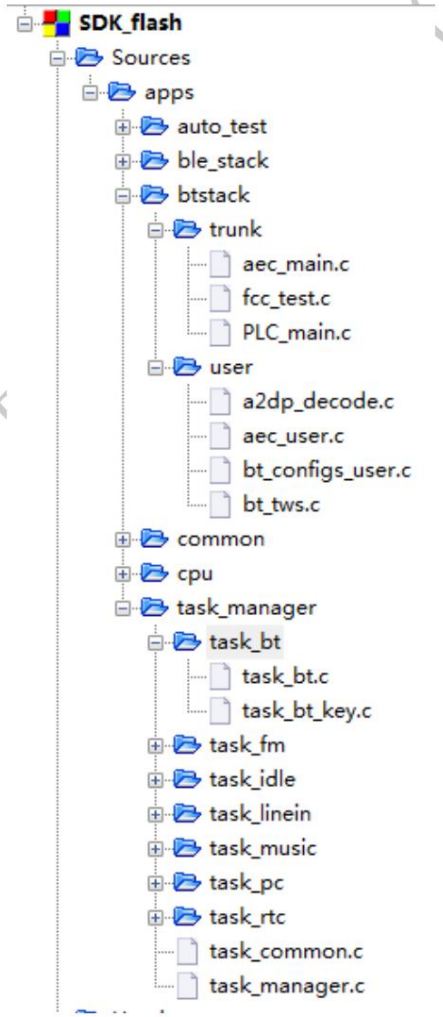
ÿ Before opening the project, make sure you have installed the latest jl_toolchain and CodeBlocks released by Jier Technology.

ÿ Double-click the sdk.cbp file in the apps directory, or drag the sdk.cbp file to the CodeBlocks shortcut

Open, the project will be opened.

ÿ Project Catalog Introduction:

ÿ After the project is opened, the project directory is as follows



Sources: some .c files that users can modify, and users can also add their own c files

Headers: used to store header files. The header files under apps can be modified by users, but the header files under include_lib

This is the header file corresponding to the library. Users should be careful not to modify it at will. If you want to modify the library, update the header file at the same time.

ÿ Main C file for Bluetooth part

1. File aec_main.c

The following process code for call echo cancellation, including initialization and addition of calculation module

2. File fcc_test.c

Parameter settings for FCC certification mode

3. File PLC_main.c

The packet loss repair module code of the call does not need to be modified in general

4. File a2dp_decode.c

Some decoding processes of Bluetooth music, mainly open and close the music module outside the library. Generally, it is not necessary

To change

5. File aec_user.c

It is mainly used to set the echo cancellation parameters. To adjust the echo cancellation, please refer to this file.

6. File bt_configs_user.c

Mainly used for some Bluetooth parameter settings and callback function registration, this file will be modified frequently

Protocol selection, the following macros are in bt_configs_user.c, configure these macros to select the protocol supported by Bluetooth

Discuss.

```
///---sdp service record profile-user selects supported protocol---///
#define USER_SUPPORT_PROFILE_SPP                1
#define USER_SUPPORT_PROFILE_HFP                1
#define USER_SUPPORT_PROFILE_A2DP 1
#define USER_SUPPORT_PROFILE_AVCTP 1
#define USER_SUPPORT_PROFILE_HID                0
```

Description of the main functions in bt_configs_user.c :

ÿFunction static void bt_setup_init(u8 *adr, char *name, u8 idx, char *pin_code); is to realize reading

Flow for Classic Bluetooth Profile.

ÿFunction static void bt_function_select_init(); is a function for parameter configuration. This function concentrates most of

This function is used for secondary development settings. Sometimes new configurations added by the patch should also be placed in this function.

ÿThe function static void ble_config_select_init(void); implements the process of reading the BLE Bluetooth configuration file.

ÿThe function static void bredr_handle_register() is used to register some callback functions of the Bluetooth library.

The function library entered will be used in the corresponding process.

ÿThe function void bt_mode_init() is used to initialize the Bluetooth hardware and Bluetooth protocol stack.

Please do not adjust the initialization position at will.

7. File bt_tws.c

A process file used for box matching. Some SDK versions do not support box matching.

8. File task_bt.c

Process processing, message processing, this file will be modified frequently

Main function description:

ÿFunction void bt_work_state_control(u8 enable); controls the discoverability and connectability of Bluetooth.

ÿFunction void hook_hfp_incoming_phone_number(char *number, u16 length); if there is

When a call comes in, the library will call this callback function to feedback the phone number to the upper layer.

ÿFunction int btstack_status_update_deal(u8 *info, u16 len), this function belongs to the library

A callback function, which is called directly by the Bluetooth protocol stack process. Be careful not to add too many

Delay or waiting operation will affect the analysis of the Bluetooth command data. It is recommended to reset the Bluetooth command after determining certain status.

The complex process pushes the message to the main loop function void task_bt_deal(void *hdl) for processing. Other callbacks

The same thing should be noted for functions

ÿFunction void bt_discon_complete_handle(u8 *addr, int reason) handles the same as before

Some messages about Bluetooth connection and disconnection.

ÿThe function static void *task_bt_init(void *priv) will be called when entering the mode.

ÿFunction static void task_bt_exit(void **hdl) mode exit will call

ÿFunction void task_bt_deal(void *hdl) is the main loop function, which processes key messages or messages generated by other situations.

Note that some of the functions in this file are background management functions, which are reserved for future expansion. AC692x does not support background management.

9. File task_bt_key.c

Key message table

10. File bt_ui.c

Display processing for screen version

ÿ The most important header file of Bluetooth----avctp_user.h

This header file defines many Bluetooth library states and commands supported by Bluetooth.

Multiple query command interfaces for changing files. In many cases, when updating a library, you need to pay attention to whether there is a file to be updated in the patch.

Chapter7 Music Development Instructions

7.1 Overall Design

ÿ Functional Overview:

The functions implemented by the Music task are as follows:

ÿ Support playback of MP3, WMA, WAV, FLAC, and APE files.

ÿ Support playback of SD card, U disk, and FLASH.

ÿ Support single channel decoding.

ÿ Supports obtaining song playing time, total time, song name, etc.

ÿ Support breakpoint playback of songs.

ÿ Support fast forward, fast rewind, repeat, pause and continue playing.

ÿ Support variable sampling, channel control, EQ/spectrum, volume/auto mute and other sound effects.

ÿ Support multiple sound effects turned on at the same time.

7.2 Overall Architecture Design

ÿ The overall software flow chart of the music mode is shown in Figure 1.2.1:

ÿ Music playback related initialization mainly includes file playback mode, device selection mode, supported file suffixes Settings such as decoding error handling and registration of other functions.

ÿ Before decoding, the device to be operated will be checked online, and the file format of the file to be played will be checked. If the device is not online or the file does not exist in the device, the file is damaged, the file format is not supported, etc., the corresponding Error handling.

ÿ The decoding-related function calls are performed in the interrupt service function of the soft interrupt. By registering the soft interrupt, In conjunction with the data output from the DAC, the interrupt flag of the soft interrupt is set from time to time. In the soft interrupt service function Perform decoding and output operations, and handle errors generated during the decoding process.

ÿ Because decoding and other operations are performed in the soft interrupt service function, it supports up and down tracks and pause during playback. Responses to stop\play, plug and unplug devices, etc.

ÿ This system supports opening multiple sound effects at the same time, and the decoded data is output after being superimposed through sound effect processing.

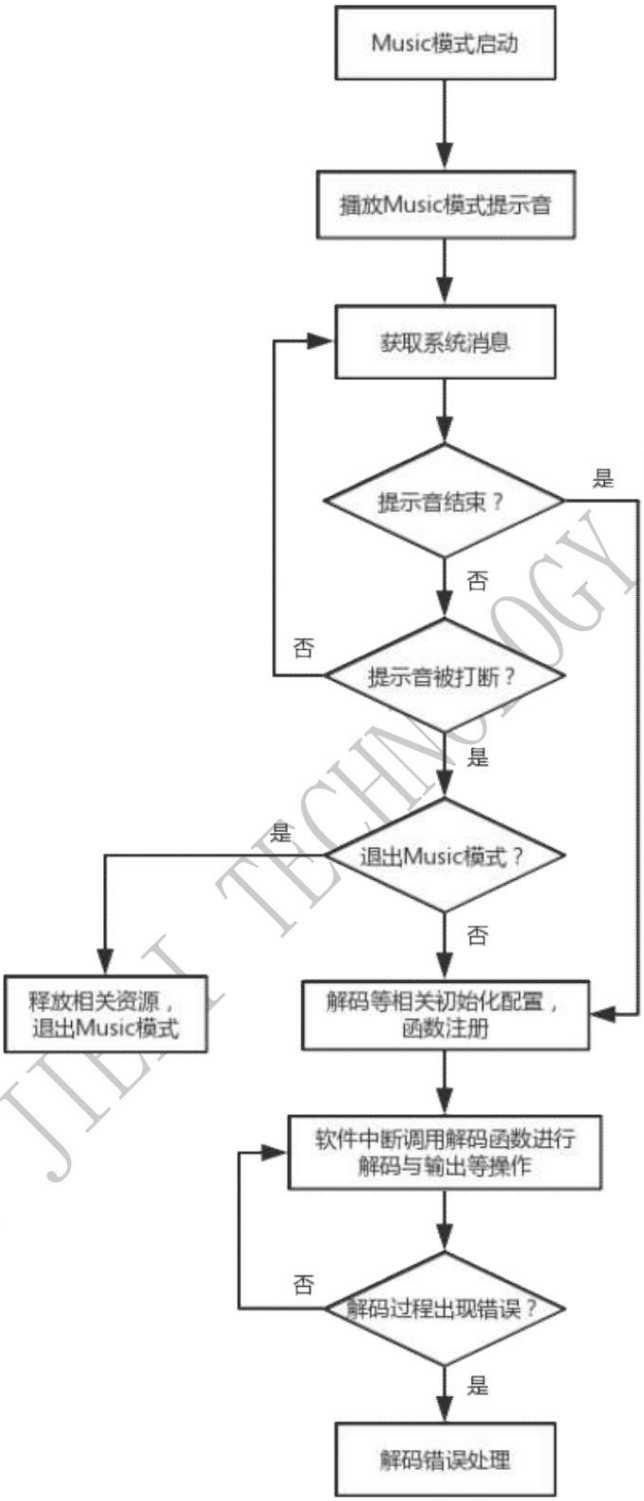


Figure 7.2.1 Music mode flow chart

7.3 Decoding Channel Description

ÿ The decoding related program flow is as follows:

ÿ Initialize decoding system parameters and enable decoding libraries of different formats.

ÿ Check the file format and call the corresponding decoding function.

ÿ Perform decoding and decoding output.

ÿ Perform relevant error handling when a decoding error occurs.

ÿ During playback, the system receives messages for processing. Depending on the message, it can control songs, switch songs, or switch devices.
Switch, sound control and other operations.

ÿ After a song is finished playing, it will automatically search for the next song to play.

ÿ If a new device is plugged in, it will enter the most recently plugged in device and search for songs to play.

ÿ If the device being played is unplugged, send a SYS_EVENT_DEC_DEVICE_ERR message and search for the next device.
If there is no device left, wait to exit music mode.

7.4 Partial API Function Description

Function prototype:

MUSIC_PLAYER *music_player_creat(void)

Function description: Create and set the music playback control handle

Parameter Description: None

return: NULL: creation failed

Others: Create successfully and return the corresponding handle address

tbool music_player_play(MUSIC_PLAYER *obj, MUSIC_PLAYER_BP *bp_info, u8 is_auto)

Function description: Music playback process control

Parameter description: *obj Music playback control handle

*bp_info Breakpoint information

is_auto auxiliary parameter

return: falseÿ Playback failed

TrueÿPlayback successful

Chapter8 Radio Development Instructions

8.1 Overall Design

ÿ System:

This function is mainly based on the SDK development kit to realize the function of radio reception.

The main functions of FM applications include:

ÿ Automatic channel search mode, manual channel search mode, semi-automatic channel search.

ÿ Support pause and play radio stations.

ÿSupport breakpoint memory, can remember the frequency of last playback.

ÿ Overall architecture design:

The radio is mainly divided into two functional modules:

ÿ Radio main mode module: Initialize FM module, play the current channel, and use the up and down buttons to select the channel to play.

ÿAutomatic channel search, semi-automatic channel search, manual channel search.

ÿApplication launch and application exit

The radio mode can be entered in the following ways:

Press the Mode key to switch modes and enter FM mode. Press the Mode key again to exit FM mode.

8.2 Radio Design Description

ÿ Module Description

In the radio process, a unified interface is used to be compatible with multiple radio modules. Different radio modules only need to provide the following functions

The function can be added to the sound reception process:

ÿ Get module ID function

ÿ Startup/initialization function

ÿ Close function

ÿ Set volume function

ÿ Set frequency function

ÿ Process Function

The radio process has added basic radio functions. The main function is to connect the basic module driver to achieve the following functions:

ÿ Full frequency channel search

ÿ Search for the previous/next frequency

ÿ Move to the next/upper frequency point

ÿ Mute, unmute

8.3 Built-in radio search parameter description

ÿ Introduction to built-in FM station search parameters

ÿ FMSCAN_SEEK_CNT_MIN: Zero-crossing counting to determine the minimum value.

ÿ FMSCAN_SEEK_CNT_MAX: Zero-crossing count to determine the maximum value.

ÿ FMSCAN_CNR: signal-to-noise ratio threshold.

These three parameters are set in the function fm_inside_io_ctrl(SET_FM_INSIDE_SCAN_ARG1 as follows:

fm_inside_io_ctrl(SET_FM_INSIDE_SCAN_ARG1,FMSCAN_SEEK_CNT_MIN,
FMSCAN_SEEK_CNT_MAX, FMSCAN_CNR);

ÿ True station judgment: When scanning the radio station, the signal zero crossing statistical value seek_cnt is in [FMSCAN_SEEK_CNT_MIN,
If the signal-to-noise ratio of the current radio station is within the range of FMSCAN_SEEK_CNT_MAX, and the signal-to-noise ratio of the current radio station is cnr>= FMSCAN_CNR, it is considered to be a real station.
It is judged as a false platform.

ÿClear and unclear radio station characteristics:

ÿ Characteristics of a clear and normal radio station: Within a fixed time, the seek_cnt statistics of the signal zero crossing point are within a relatively small range.
The signal-to-noise ratio cnr is relatively large.

ÿ No station (white noise) or unclear radio station characteristics: seek_cnt is generally large, cnr is relatively small.

ÿ General debugging method of built-in FM:

1. Configure the search channel according to the default parameters. If the search channel does not meet the requirements, first check the hardware. If there is no problem with the hardware, continue to step 2.
step.

2. Perform a station search (press message MSG_FM_SCAN_ALL_INIT), open the print function in the program to view all radio stations.
seek_cnt, cnr0, cnr1. Preliminary statistics of the zero crossing points of each radio station seek_cnt, signal-to-noise ratio cnr0, cnr1.

Typical search printing parameters are as follows:

[freq: 875 seek_cnt: 554 cnr: 22]

[freq: 876 seek_cnt: 385 cnr:-21]

[freq: 877 seek_cnt: 545 cnr: 0]

Adjust the signal-to-noise ratio threshold FMSCAN_CNR and the zero-crossing value range [FMSCAN_SEEK_CNT_MIN,
FMSCAN_SEEK_CNT_MAX], when

(cnr>=FMSCAN_CNR))&&(FMSCAN_SEEK_CNT_MIN<=seek_cnt)&&(seek_cnt<=FMSCAN_SEEK_CNT_MA

X), the current radio station is judged as a real station, otherwise it is judged as a fake station.

Note: To print seek_cnt/cnr information while searching channels, you need to call the following function to open the related printing in the library.

fm_inside_io_ctrl (SET_FM_INSIDE_PRINTF, 1); //1 turns on printing in the library. 0 turns off printing in the library.

If you suspect that printing is turned on in the library during channel search, it will affect the channel search parameters. You can turn off printing during channel search. After searching, turn on the following

The function prints out the search parameters uniformly (when searching, the parameters are stored in the internal RAM first)

fm_insice_scan_info_printf(875,1080); //Print the search parameters from 87.5 to 108.0.

ÿ Common channel search problem handling:

1) Insufficient number of receiving channels: first lower FMSCAN_CNR, if that still doesn't work . Then relax [FMSCAN_SEEK_CNT_MIN, FMSCAN_SEEK_CNT_MAX]

2) Too many fake stations: first increase FMSCAN_CNR, then narrow [FMSCAN_SEEK_CNT_MIN, FMSCAN_SEEK_CNT_MAX].

ÿ Other API functions:

1) void fm_inside_set_stereo(u8 set); //Dual channel (stereo) effect setting. The value range of set is [0,127].

When the set value is 0, it is equivalent to complete mono.

The closer the set value is to 127, the more obvious the dual-channel effect is.

When the set value is 127, it is dual channel.

2) void fm_inside_set_abw(u8 set); //audio bandwidth setting, set value range [0,128].

The larger the set value is, the wider the audio bandwidth is. The adjustable bandwidth range is [2k, 16k]

3) void fm_inside_deempasis_set(u8 set); //de-emphasis parameter setting. set can only be set to 0 or 1.

When the set value is 0, the de-emphasis time parameter is 50us.

When the set value is 1, the de-emphasis time is 75us.

4) s16 fm_inside_rssi_read(void); //Get the RSSI value of the receiver. The unit is dB.

Chapter9 Clock Development Instructions

9.1 Chapter Introduction

This chapter provides design and development documents for developers of AC692x_SDK applications, and also provides a tool for testing RTC applications.

The document provides a reference for testers. It defines in detail the overall functions of the RTC application, the system interface and data attributes;

The basic structure, functional modules and names of each program are divided to facilitate the detailed design and coding of RTC applications.

9.2 Overall Design

ÿ Requirements Overview

This module is mainly based on the AC692x_SDK system development kit to implement the RTC function. The main functions implemented by the RTC application include:

(1) Support system time adjustment.

(2) Support alarm setting processing.

The development and implementation of this module is based on the Jerry AC692x_SDK software development kit and the corresponding hardware platform.

ÿ Overall architecture design

RTC is mainly divided into three functional modules:

ÿ Display module: mainly displays the current time

ÿ Setting module: set system time and alarm time

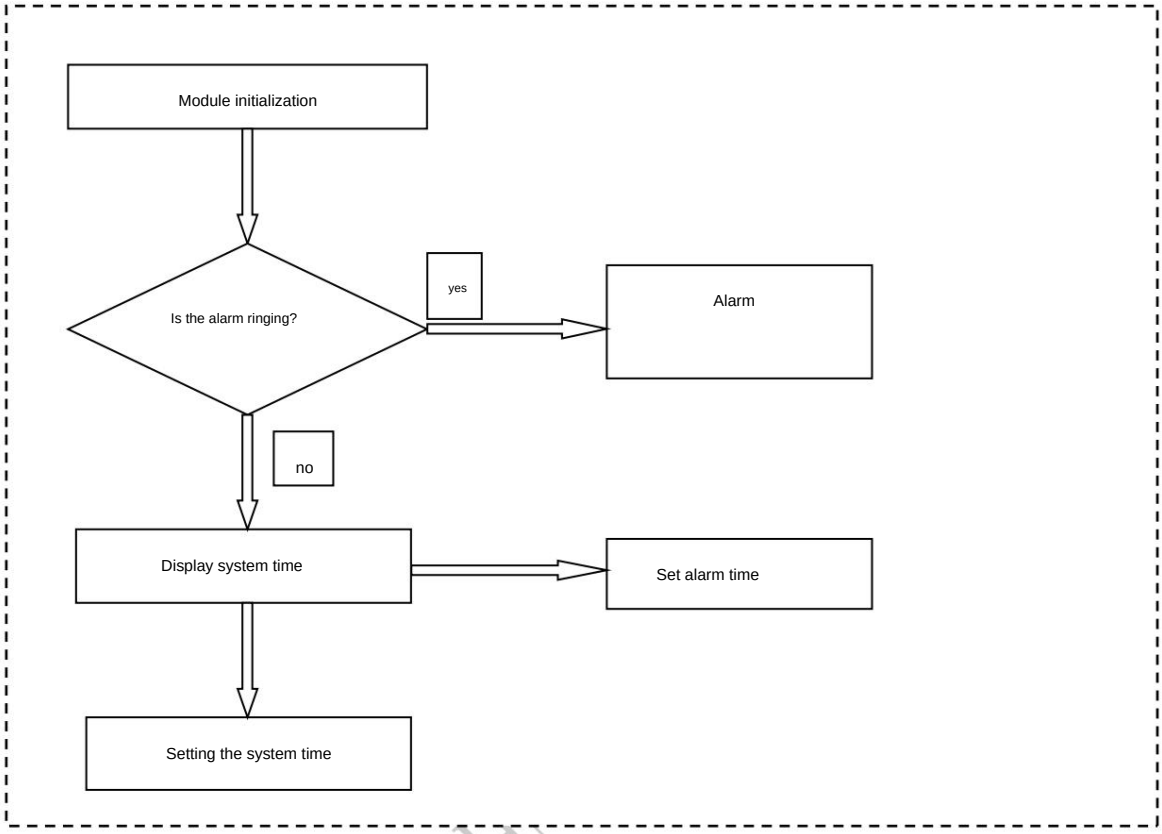
ÿ Alarm response module: mainly responds to the alarm

ÿ Overall architecture diagram

Figure 10.2.1 RTC flow chart

ÿ Functional module division

Module Name	Functional Description	Corresponding files
System entry module alarm response	Set system related information and respond to alarm operations	rtc.c
Display Module	Display system time, set system time and alarm time rtc_ui.c	
Clock Setting Module	Setting time and alarm operation process	rtc_setting.c



ÿ Functional module division

Module Name	Functional Description	Corresponding files
System entry module alarm response	Set system related information and respond to alarm operations	task_rtc.c
Display Module	Display system time, set system time and alarm time	rtc_ui.c
Clock Setting Module	Setting time and alarm operation process	rtc_setting.c

ÿ Application life cycle

ÿ Press the Mode key to switch modes, enter RTC mode, and start running. Press the Mode key again to exit RTC mode.

End of life cycle.

ÿ Application startup

When switching to the RTC task, the void rtc_info_init(void) function is called to create and initialize

Resources and hardware modules required for RTC.

The activation process mainly consists of the following steps:

(1) Initialize the resources required by RTC and configure the initial values

(2) Initialize RTC information and check whether the clock and alarm range are normal.

ÿ Exit of application

When there is a message to switch tasks or insert a device, task_common will select the task to switch and exit RTC model.

9.3 System Entry Module Design Description

ÿ Module Description:

The corresponding file is task_rtc.c, which is the entry point of the RTC application and is responsible for system initialization when entering. It is responsible for closing when exiting. Resources and hardware modules.

ÿ Module Function

This module is the entry module of the application, and is mainly responsible for system initialization, screen initialization, DAC initialization, etc.

ÿ Module interface design

Function description: Register the callback function for RTC interrupt. The flag used in the callback distinguishes the interrupt situation. The processing to be done when triggered is in the if(RTC_ISR_ALARM_ON == flag) branch.

Function prototype: void rtc_isr_user_handler(u8 flag)

Other notes:

1) if (RTC_ISR_PCNT==flag) interrupt generated by count overflow

2) if(RTC_ISR_LDO5V == flag) is the interrupt generated by LDO5V detection

9.4 Setting time module design description

ÿ Module Description

This module allows customers to adjust time and alarm.

ÿ Module Function

This module allows customers to adjust time and alarm.

ÿ Module interface design

Function description: Set the time or alarm in RTC mode according to the setting mode

Function prototype: void rtc_setting(int msg)

For more RTC function setting interfaces, please refer to the rtc_api.h file.

ÿ Exception handling

Timeout, main interface message, device insertion will exit setup mode.

9.5 Alarm Module Design Description

ÿ Module Description

This module responds to the alarm.

ÿ Module Function

This module responds to the alarm, displays the alarm interface and sounds the alarm.

ÿ Module interface design

Function description: alarm, clock setting

Function prototype: void rtc_setting(int msg)

Function description: Alarm switch

Function prototype: void rtc_sw(u8 flag)

For more RTC function setting interfaces, please refer to the rtc_api.h file.

ÿ Exception handling

Timeout, main interface message, device insertion will exit setup mode.

9.6 RTC module special function description

ÿ Ordinary IO port

RTC has independent IO ports, PORTR0~3, PORTR1 outputs 0 by default, and PORTR0 multiplexes 32.768KHz Crystal oscillator pin, for RTC internal clock module. PORTR3 multiplexes 12-26MHz crystal oscillator, can connect external crystal oscillator is the system clock.

PORTR0~3 can wake up the RTC low power mode.

The function to set the IO port is as follows. Only one IO port can be set at a time:

```
void PORTR_DIR(u8 port, u8 val);
void PORTR_OUT(u8 port, u8 val);
void PORTR_HD(u8 port, u8 val);
void PORTR_PU(u8 port, u8 val);
void PORTR_PD(u8 port, u8 val);
u8 PORTR_IN(u8 port);
```

PortÿPORTR0ÿPORTR1ÿPORTR2ÿPORTR3 Val: 0 or 1

NOTEÿ

1. PORT3 pin must be connected to a 32.768KHz crystal oscillator for the clock function to work properly.
2. PORTR3 cannot be set to output if a crystal oscillator is used, otherwise the clock will stop
3. After entering low power consumption, PORTRIO can still work normally, and other common ports are in high impedance state IO

ÿ PORTR1 and PORTR2 can be used as ADkey. The following code example is provided

```
/*1: PR1 port voltage drive ADC                    0: no this function*/
//if you want to use port1 to be ADKEY io, you can set as below
PORTR1_ADCEN_CTL(1);
PORTR_PD(PORTR1 ,      0);
PORTR_PU(PORTR1 ,      0);
PORTR_DIR(PORTR1 ,      1);
```

PORTR_DIE(PORTR1 , 0);

ÿ The RTC module also supports counter overflow wake-up and generates corresponding interrupts.

Function description: Set the related configuration of count wakeup.

Function prototype: void pcnt_init(u8 port, u8 edge);

Function description: Set the count value of the count register.

Function prototype: void set_pcnt_value(u8 value);

Function description: Get the count value of the count register.

Function prototype: int get_pcnt_value();

ÿ PORTR port long press reset function interface

Function description: Set the configuration of button reset

Function prototype: int rtc_port_reset(u8 mode, u8 port, u8 enable, u8 edge);

ÿ Low power consumption

RTC supports independent low power mode. When entering low power mode, the main control power will be turned off.

Only the RTC module is working, please refer to the relevant chapters on low power consumption.

Chapter10 PC slave development instructions

10.1 Overall Design

ÿ System:

This description is mainly based on the SDK development kit to realize the functions of PC.

The main functions of PC application include:

ÿ Support card reader function.

ÿ Support sound card function.

ÿ Support HID function

ÿ Overall architecture design:

PC tasks are mainly divided into three functional modules:

ÿ Card reader function module: In slave mode, the PC can operate the storage device in the device. ÿ Sound card function module: In slave mode, it acts as the speaker of the PC music player. ÿ HID operation module: It mainly acts as a sound card from the slave to play the music played by the PC music player.

Can control the corresponding operation of PC player by controlling the previous/next song and play on the slave machine.

ÿApplication startup

There are two ways to enter PC mode:

ÿAfter the USB slave cable is inserted into the device, the device manager detects it and sends a SYS_EVENT_PC_IN, the main thread will forcefully exit the current mode and activate the PC mode.

ÿ By switching the mode button, the main thread will forcefully exit the current mode and activate the PC mode.

ÿ Exit of application:

When the Mode key is pressed, the PC cable is unplugged, or other tasks need to be activated, the main thread will force exit PC mode.

Free up PC task resources

Turn off the beep

ÿ Application dependency library and its interface description

usb_lib.a	--USB module device library
lib_usb_syn.a	-- PC Mode AUDIO Synchronous Library

10.2 System Entry Module Design Description

ÿ Module Description

The corresponding file is task_pc.c, which is the entry point of PC application. When entering, it is responsible for the variable application and hardware initialization of PC mode.

Responsible for releasing the variable resources occupied by PC mode.

ÿ Module Function

This module is the entry module of the application, which is mainly responsible for the initialization of the system when entering, such as the initialization of the card reader function, AUDIO

Functions include DAC initialization, HID initialization, etc.

10.3 Card Reader Module Design Description

ÿ Module Description

This module mainly realizes the reading of large-capacity storage devices. card_reader_io.c provides the IO connection between the card reader and the device.

ÿ Module Function

The card reader function.

ÿ Module interface design

Card reader function realization function description:

ÿ Function prototype: s32 app_usb_slave_card_reader(u32 cmd)

ÿ Function description: PC card reader execution process function

ÿ Parameter description: cmd command

ÿ Return value description: execution status

ÿ Function prototype: sUSB_DEV_IO *get_card_read_io(DEV_TYPE dev_type)

ÿ Function description: Get the IO of the card reader operation device

ÿ Parameter Description: dev_type device type

ÿ Return value description: device operation IO

10.4HID Operation Module Design Description

ÿ Module Description

This module mainly realizes the corresponding operation of the player on the PC by using the device connected to the PC line.

ÿ Module Function

This module implements HID functionality.

ÿ Module interface design

HID Function Description:

- ÿ Function prototype: void usb_slave_hid(u32 key)
- ÿ Function description: PC HID function
- ÿ Parameter description: key: PC HID key command
- ÿ Return value description: None

10.5USB_SPK Module Design Description

ÿ Module Description

This module mainly realizes the function of using this device as a PC sound card.

ÿ Module Function

This module implements the USB_AUDIO functionality.

ÿ Module interface design

PC_AUDIO Sound Card

Functional description:

ÿ Function prototype: u8 pc_set_speaker_vol(u32 pc_mute_status)

ÿ Function description: PC AUDIO volume setting function

ÿ Parameter description: pc_mute_status: mute status

ÿ Return value description: current sound card volume value

ÿ Function prototype: void pc_dac_mute(bool mute_status, u8 fade_en)

ÿ Function description: PC AUDIO mute setting function

ÿ Parameter description: mute_status: mute status; fade_en: fade in and fade out settings

ÿ Return value description: None

10.6 PC detection function

In each mode, it will automatically jump to PC mode when the PC cable is inserted, and it can also switch to PC mode through the Mode key.

ÿ Function prototype: void pc_check_api(void)

ÿ Function description: PC online detection function

ÿ Parameter description: void

ÿ Return value description: void

Add a PC detection function to the device detection task. If a PC cable is detected, the device detection task will issue a EVENT_PC_IN event, switched to PC mode controlled by task_common.

Chapter11 F1A prompt sound file

11.1 Overview of F1A Prompt Tone

F1A prompt sound file is an audio format specially customized by our company based on the characteristics of prompt sound file.

Improve the compression ratio to make the prompt tone take up as little code space as possible.

Turn on the function.

The corresponding function macro is in the sdk_cfg.h file

```
#define DEC_TYPE_F1A_ENABLE                ///<30K_code_space
```

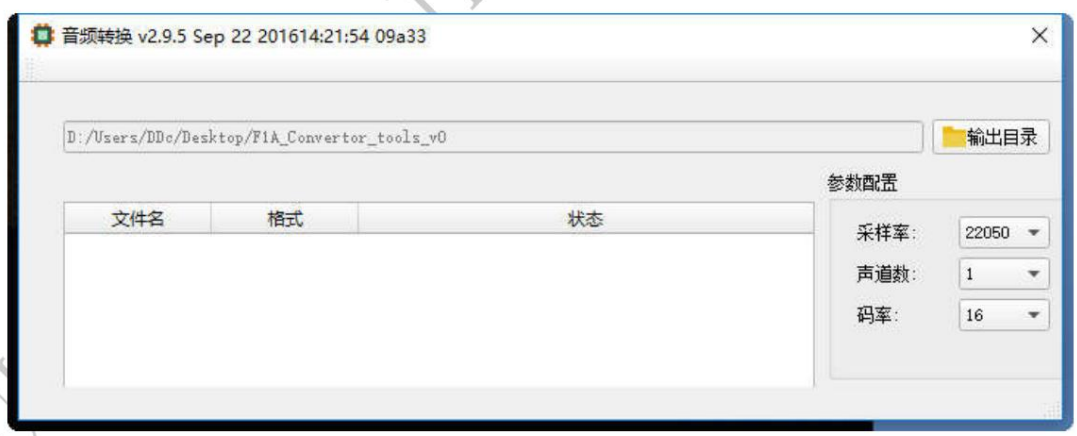
Download the corresponding prompt tone file.

apps\download\ac692x\post_build\download.bat file

isd_download.exe …. power_off.f1a bt.f1a music.f1a record.f1a linein.f1a radio.f1a pc.f1a wait.f1a

connect.f1a disconnect.f1a ring.f1a ….

F1A_Convertor_tools_v1.0 can convert ordinary audio files to F1A files.



picture

Chapter 12 Reverb mode and reverb function

12.1 Overall Design

ÿ System:

This description is mainly based on the SDK development kit to achieve the reverberation function.

The main functions of the reverberation application include:

ÿ Depth and intensity customization.

ÿ Support enabling reverberation function in multiple modes.

ÿ Overall architecture design:

The reverberation task is mainly divided into two functional modules:

ÿ Audio source module: Reverb source input only supports mic channel.

ÿ Reverberation processing module: After obtaining the sound source, it performs reverberation effect processing and then outputs it to the callback function.

ÿApplication startup

Reverb mode can be entered in the following ways:

ÿ Turn on the reverb mode by pressing the button.

ÿ When the Mode button is pressed, the sound switches to Reverb mode.

ÿ Exit of application:

Reverb mode can be entered in the following ways:

ÿ Turn off the reverb mode by pressing the button.

ÿ When the Mode key is pressed, the reverb mode is exited.

ÿ Application dependency library and its interface description

ÿ lib_audio_reverb_H.a ——Reverb processing library

ÿ Parameter Description

```
// ECHO 效果参数初始化.
ECHO_CONTROL_VAR  echo_var = {
    .music_vol = 13384,           //0 ~ 16384   //背景音乐音量大小. 16384(0 db)
    .mic_vol_digital = 13384,     //0 ~ 16384   //MIC数字音量大小.  16384(0 db)
    .deep = 900,                  //0 ~ 1024    //值越大, 混响深度越深
    .mic_vol = 50,                //0 ~ 63      //MIC模拟音量(放大)大小
    .pitch = -2,                  //-127 ~ 127  //变调, 0为正常声音, 负值越小频率越低, 正值越大频率越高. (howlingsupression)
    .decay = 110,                 //0 ~ 128     //混响强度衰减: 值越大, 混响的后一声比前一声衰减越多.
    .run_flag = 0,
    .first_echo_sel = FIRST_ECHO_ANALOG,
};

// ECHO 效果参数初始化.
REVERB_PARM_SET rev_parm = {
    /*-------------------------------------------------*/
    /* 以下参数用于调试使用, 请勿修改 */

    /* reserved */

    /*-------------------------------------------------*/
    .ef_reverb_parm_2.deepval       = 1024,     /* 混响深度, 范围:(0-1024) , (1024->max_ms)) */
    .ef_reverb_parm_2.decayval      = 128,      /* 混响强度, 范围:(0-128) */
    .ef_reverb_parm_2.gainval       = 4000,     /* 音量增益, 范围:(0-4096) */
    .ef_reverb_parm_2.rs_mode       = 0x100,    /* reserved */
    /*-------------------------------------------------*/
};
```

Parameter Description (reserved is only used for debugging, modification is not recommended)

Note: Due to RAM resource issues on 692X, the maximum reverberation depth configured by the SDK by default is 200ms, which requires approximately 8K of space.

12.2 System Entry Module Design Description

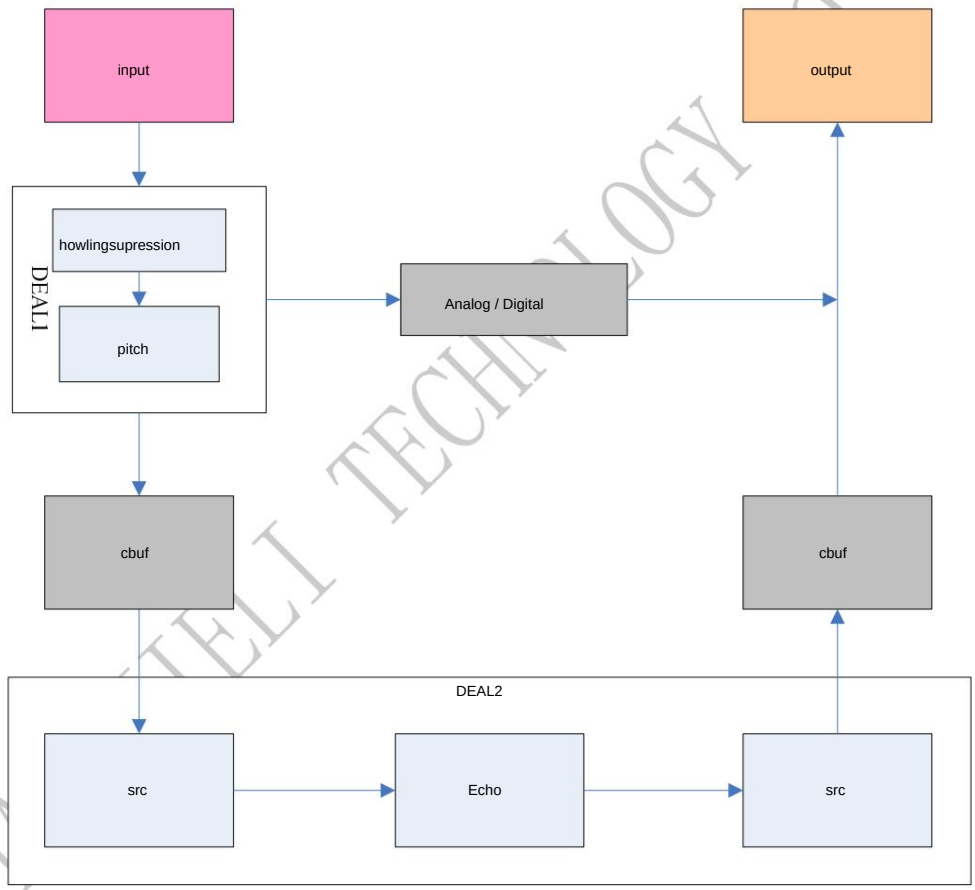
ÿ Module Description

Reverb comprehensive message processing, support turning on/off reverb.

ÿ Module Function

In different modes, press the Reverb On/Off button to turn the reverb function on/off.

flow chart:



Flowchart 1

Chapter13 Bluetooth Box Development and Use Instructions

13.1 Purpose

This document mainly describes the AC692x TWS development method and some issues that need to be paid attention to during development, and provides users with

For reference only.

TWS development is compatible with ordinary single Bluetooth speaker development, and other development documents and tool instructions are common.

Explain the development and use of Bluetooth TWS.

13.2 Terms and abbreviations

Tip: List the definitions of specialized terms used in this document and the original phrases of foreign initials.

Abbreviations and terms	explain
AC692x	Jerry Technology AC692x series chips
TWS (turewireless) Bluetooth pairing, based on master and slave Bluetooth devices	
TWS_ROLE_MASTER Bluetooth pairing host	
TWS_ROLE_SLAVE Bluetooth pairing slave	
Inquiry	Discovery Device Status
Inquiry scan/i_s	Discoverable status
page	Connected device status
page scan/p_s	Connectable status

13.3 TWS Development Instructions

1. TWS Configuration Options

1. Enable configuration for the box

```
///Optional configuration: 0 (normal speaker)/BT_TWS_TRANSMIT (enable speaker)
#define BT_TWS                          BT_TWS_TRANSMIT
```

2. Operation settings **tws** option configuration

1. Press the button to search for the device: **MSG_BT_SEARCH_DEVICE**

For the first pairing, you need to use this message to search and pair with the device.

user_send_cmd_prepare(USER_CTRL_SEARCH_DEVICE,0,NULL);

```
case MSG_BT_SEARCH_DEVICE:
    puts("MSG_BT_INQIRY_DEVICE\n");
    if (get_tws_device_role() == 0) {
        user_val->inquiry_flag = 1;
        bt_work_state_control(0);
        user_send_cmd_prepare(USER_CTRL_SEARCH_DEVICE, 0, NULL);
    } else {
        puts("TWS_connected,ignore INQUIRY\n");
    }
    break;
```

Note: By default, when connecting to a box, the device search information will be automatically ignored.

//Set the Bluetooth search time, actual time = N*1.28s

__set_search_timeout_value(0x06);/*Search device time, Range: 0x01-0x30, Time = N * 1.28s*/

2. Open and discoverable connection

case MSG_BT_PAGE_SCAN:

puts("MSG_BT_PAGE_SCAN\n");

bt_work_state_control(1);

break;

3. Whether to connect to **the TWS** slave after booting

__set_tws_start_conn(1); /*Reconnect to the box, 0: do not reconnect, 1: reconnect*/

4. Set the box search mark, and pairing connection is allowed only when the corresponding mark is found.

/*Set the search mark for the box, used when inquiring, and the connection is allowed only when the corresponding mark is found*/

char tws_device_indicate[2] = {'J', 'L'};/*Only two characters, you can customize the content*/

5. After the TWS connection is successful, the master device performs call processing and control, and transmits the audio stream to the slave device. The master and slave devices cooperate

Realize wireless channel synchronous playback. Master and slave devices play audio synchronously, and can play left and right channels separately.

Synthesize master-slave synchronized playback.

When connected to a mobile phone, it will automatically play in stereo.

ÿÿÿaudio_sync/sync_tws.c:void*get_tws_sync_param()

```
tws_sync.tws_sync_parm.set_user_ch = 0;                    //0:master left 1:master right
tws_sync.tws_sync_parm.set_ch_mode = 0;                    //0: Channel independence 1: opL = opR = (L+R)/2
tws_sync.tws_sync_parm.sbc_bitpool_value = 31; // Audio compression rate for the box
```

6. When the Bluetooth phone call is played on the master device, the slave device does not establish a call connection. That is, the call is only realized on the master device.

7. Display the current device search pairing status (for debugging)

void bt_baseband_state_debug()ÿ

This interface can show whether the device is currently discoverable and connectable, whether it is pairing or searching for a device.

8. avctp_user.h declares some commonly used **tws** interfaces

9. The box is connected successfully and the current device role (host or slave) is saved

```
/**
 *保存对箱连接角色
 *用来判断下次开机是否发起回连
 */
void bt_tws_info_save(u8 info)
{
    u8 tws_info = 0;
    vm_read(VM_TWS_INFO, &tws_info, VM_TWS_INFO_LEN);
    if (tws_info == info) {
        puts("same tws_info,return\n");
    }
    tws_info = info;
    update_bt_tws_info(info);
    log_printf("TWS_info W:%x\n", info);
    vm_write(VM_TWS_INFO, &tws_info, VM_TWS_INFO_LEN);
}
```

10. When the device is turned on, the corresponding action is performed by reading the device's memory device role.

```
/**
  *获取对箱上次连接角色，用来判断开机是否发起回连
  *主机发起回连，从机等代连接
  */
u8 get_tws_mem_role(void)
{
    u8 info;
    vm_read(VM_TWS_INFO, &info, VM_TWS_INFO_LEN);
    log_printf("TWS_info R:%x\n", info);
    //return 0;
    return info;
}
```

3. After TWS is successfully connected, the master and slave devices can be controlled by key association

1. After TWS is connected, key messages will be associated through the following interfaces

```
#if BT_TWS
        if (tws_key_cmd_send(msg[0], 0)) {
            continue;
        }
#endif
```

Note: Some messages are implemented by the master and do not need to be implemented synchronously by the slave and will be filtered out.

2. After TWS is connected, information is synchronized

```
/**
  *对箱连接成功，同步一些必要的配置，如eq模式，音量级数等
  *如果不想同步，不注册该接口或者直接返回即可
  */
static void bt_tws_info_sync(void)
{
    tws_cmd_send(MSG_BT_TWS_EQ_SYNC, sound.eq_mode);
    tws_cmd_send(MSG_BT_TWS_VOL_SYNC, sound.vol.sys_vol_l);
}
```

Here, more information can be synchronized according to actual needs.

3. TWS host and slave upper layer communication interface

void tws_cmd_send(int msg, u8 value)

For example, for synchronous shutdown, you can send a shutdown command to the other party before shutting down.

4. Status acquisition and processing after TWS connection is successful

1. Get what device is currently connected or disconnected

```
    case BT_STATUS_FIRST_CONNECTED:
    case BT_STATUS_SECOND_CONNECTED:
#if BT_TWS
        /*判断当前连接的设备类型:对箱或者手机*/
        newest_bd = get_cur_conn_bd(1);
        log_printf("BT_CONNTCTED:%d\n", newest_bd);
        if (newest_bd == BT_CUR_CONN_PHONE) {
            puts("[connect_dev]phone\n");
        } else if (newest_bd == BT_CUR_CONN_TWS_MASTER) {
            puts("[connect_dev]tws_master\n");
        } else if (newest_bd == BT_CUR_CONN_TWS_SLAVE) {
            puts("[connect_dev]tws_slave\n");
        }
#endif
        led_fre_set(C_BLED_FAST_ONE_5S_MODE);
        /* led_fre_set(C_BLED_FAST_DOBLE_5S_MODE); */
        task_post_msg(NULL, 1, MSG_BT_TONE_CONN);
        break;
    case BT_STATUS_FIRST_DISCONNECT:
    case BT_STATUS_SECOND_DISCONNECT:
#if BT_TWS
        /*判断当前断开的设备类型: 对箱或者手机*/
        newest_bd = get_cur_conn_bd(0);
        log_printf("BT_DISCONN:%d\n", newest_bd);
        if (newest_bd == BT_CUR_CONN_PHONE) {
            puts("[disconn_dev]phone\n");
        } else if (newest_bd == BT_CUR_CONN_TWS_MASTER) {
            puts("[disconn_dev]tws_master\n");
        } else if (newest_bd == BT_CUR_CONN_TWS_SLAVE) {
            puts("[disconn_dev]tws_slave\n");
        }
#endif
        task_post_msg(NULL, 1, MSG_BT_TONE_DISCONN);
        break;
```

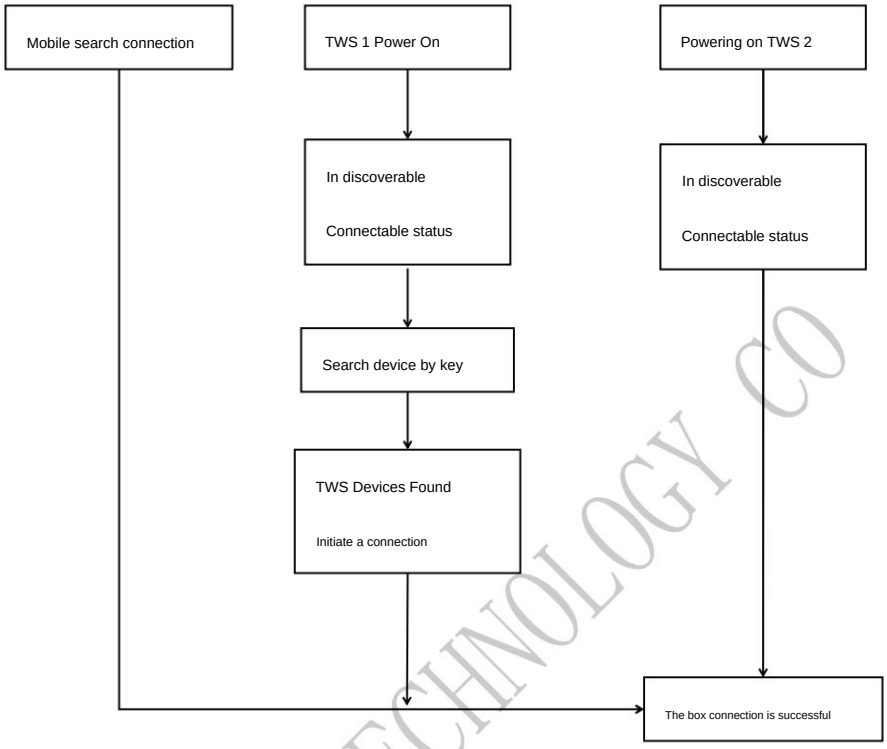
2. Manually connect and disconnect the box

```
    /*对箱连接控制*/
    case MSG_BT_TWS_TEST:
    case MSG_BT_TWS_CONNECT_CTL:
        if ((get_tws_connect_status() == BT_STATUS_CONNECTING)    ||
            (get_tws_connect_status() == BT_STATUS_TAKEING_PHONE) ||
            (get_tws_connect_status() == BT_STATUS_PLAYING_MUSIC)) {
            //puts("disconn_tws\n");
            user_send_cmd_prepare(USER_CTRL_TWS_DISCONNECTION_HCI, 0, NULL);
        } else {
            //puts("conn_tws\n");
            user_send_cmd_prepare(USER_CTRL_TWS_START_CONNECTION, 0, NULL);
        }
        break;
```

1.4 Bluetooth **TWS** communication process

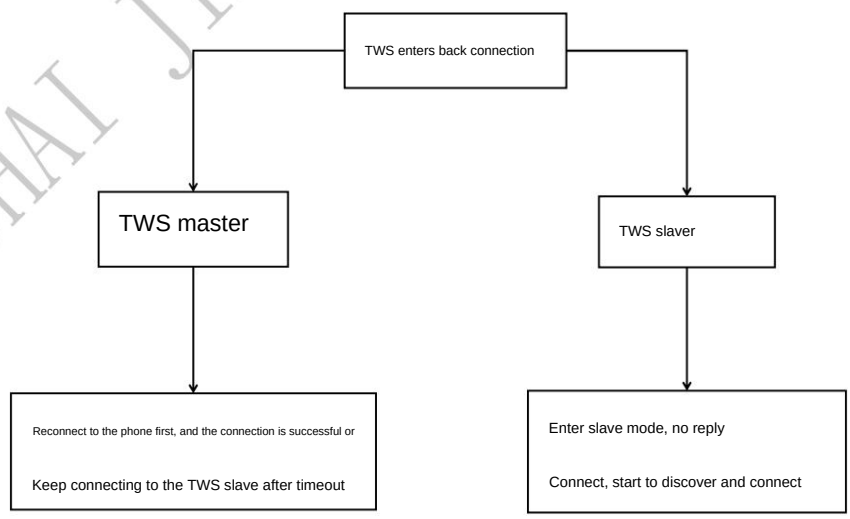
ÿ **TWS** connection process

The actual connection process depends on the configuration options.



ÿ TWS boot reconnection process

According to the last connected role, the TWS host starts to initiate a reconnection



Chapter14 AC692X serial port upgrade protocol

14.1 Purpose

This document mainly describes the method of developing serial port upgrade for AC692x and the communication protocol and communication commands developed.

This function requires the support of uboot.

It can be developed in versions later than AC692x_SDK_release_V2.3.1.

14.1 Configuration

1) Open the configuration macro:

#define UART_UPDATA_EN 2)

Configuration structure:

```
UPDATA_UART updata_uart = {
    .control_io = UART0_TXPA5_RXPA6, //Only UART0 has DMA, so only UART0 IO can be selected
    .control_baud = 460800, // baud rate
    .control_timeout = 300, //Timeout exit and resend, unit 10ms
};
```

14.2 File Usage

The file required for the upgrade is a .bfu file, such as the update.bfu file.

14.3 Agreement

Order	Outgoing	Receiver
Upgrade start tool side (12byte): UPDATE_STAR! Enter		Mcu(12byte):RECEIVE_CMDÿ
upgrade tool side (12byte): UART_UPDATE! Read file		Mcu(12byte):RECEIVE_CMDÿ
command Mcu (12byte): READ+addr+len Complete upgrade		Tool side: crc(4byte)+data
Mcu (12byte): UPDATE_SUCC! Upgrade failed Mcu		Tool side (12 bytes): RECEIVE_CMD!
(12byte): UPDATE_FAIL!		Tool side (12 bytes): RECEIVE_CMD!

Protocol explanation:

When upgrading, the MCU (AC692x) is upgraded, so it is the slave end, and the end that initiates the upgrade is the host end, such as the tool,

For other chips, the following contents all use the host to refer to the end that initiates the upgrade.

1. When an upgrade is required, the command is sent from the host to the slave (MCU: AC692x): the host sends: UPDATE_STAR! A total of 12 bytes, the slave responds: RECEIVE_CMD! A total of 12 bytes, confirming that the communication is normal;

2. After confirming the communication, enter the upgrade. The host sends UART_UPDATE! A total of 12 bytes to the slave (mcu: AC692x), enter uboot to upgrade;

3. After entering the upgrade, the command is sent from the slave (MCU: AC69) to the host: Slave sends: READ + file read address (4byte) + read length (4byte) total 12byte, length len is aligned to 4byte, and the alignment part of the data is 0 (for example When the required length is 22 bytes, it needs to be padded to 24 bytes, and the last two bytes are 0). The host sends: 16 bits of the data area CRC checksum + data; usually the data is 512 bytes, but it needs to be adapted according to the read command len. CRC is placed in the data header. Use u16 crc, which takes up 4 bytes, that is, the total reply length is 4+len, and the data is in little-endian mode (that is, when the data is 0x87654321, the data to be sent should be 0x21436587). If a packet of data is read more than 5 times, it is considered an upgrade failure, and the upgrade will be exited and the MCU will restart.

For example: read the 34 (0x22) bytes of data after 0x00 09 D8 E0, and the binary representation of the data is: 4C 15 BD 6E 55 E0 EB 75 CA 24 78 05 B2 DA 46 05 28 6B F9 AE 1B 55 7B E9 4B CD E4 69 2C 82 5B 55 B3 A9

The process is as follows:

Read file data command (READ+addr little endian format+len little endian format): 52 45 41 44 E0 D8 09 00 22 00 00 00

Reply command (crc little endian format + 34 bytes of data + data alignment and 0): C3 5F 00 00 4C 15 BD 6E 55 E0 EB 75 CA 24 78 05 B2 DA 46 05 28 6B F9 AE 1B 55 7B E9 4B CD E4 69 2C 82 5B 55 B3 A9 00 00

4. When the upgrade is completed, the command is sent from the slave (mcu: AC69) to the host: the slave sends: UPDATE_SUCC! A total of 12 bytes, the host responds: RECEIVE_CMD! A total of 12 bytes, the slave starts, and the upgrade is completed; when the upgrade fails, the command is sent from the slave (mcu: AC69) sent to the host: The slave sends: UPDATE_FAIL! A total of 12 bytes, the host responds: RECEIVE_CMD! A total of 12 bytes,

Start from the machine,

Chapter 15 Recording Mode and Recording Function

15.1 Overall Design

ÿ System:

This description is mainly based on the SDK development kit to achieve the recording function.

The main functions of the recording application include:

ÿ Support WAV and MP3 format recording.

ÿ Support USB flash drive and SD/TF card recording.

ÿ Supports up to 9999 recording file numbers.

ÿ Support BT recording, FM recording, aux recording, and mic recording in recording mode.

ÿ Supports getting data directly from DAC or directly from data source.

ÿ Support recording playback.

ÿ Overall architecture design:

The recording task is mainly divided into three functional modules:

ÿ Audio input source: In different modes, the audio source is different, but the input data is all PCM data.

ÿ Encoding module: WAV and MP3 use different encoders.

ÿ Data output: The encoded data flows out from the output module and is stored in a USB flash drive or SD/TF card by default. Only fat File system. The output destination can be modified according to needs.

ÿApplication startup

ÿ In the mode that supports recording, if the storage device is online, press the record button to start recording.

ÿ Switch through the mode button, if the storage device is online, you can enter the recording mode.

ÿ Exit of application:

When the Mode key is pressed, the device is unplugged, or other tasks need to be activated, the main thread will force the recording to close or Exit recording mode

ÿ Application dependency library and its interface description

ÿ lib_adpcm_encode.a --WAV encoding library

ÿ lib_mp2_encode.a --MP3 encoding library

15.2 System Entry Module Design Description

ÿ Module Description

The recording function mode can be added to the message processing in any mode. When a recorded message is received, you can enter the recording mode.

The corresponding message processing.

ÿ Module Function

In different modes, this module is the entry module of the application. You only need to change the input audio source to support different recordings.

The recording mode supports the following processing:

ÿ Start recording.

ÿ Pause/resume recording.

ÿ End recording.

15.3 Recording Parameter Configuration Instructions

ÿ Sampling rate parameter description

This module mainly realizes the reading of large-capacity storage devices.

(44100), (48000), (32000) The sampling rate can be configured with the following bit rates:

32,48,56,64,80,96,112,128

(44.1K/2,/4), (48k/2,/4), (32K/2,/4) The sampling rate can be configured with the following bit rates:

8,16,24,32,40,48,56,64,80,96,112,128

ÿ Recording folder

Recording folder, only supports one level of directory:

```
const char rec_folder_name[] = "/JL_REC";
```

ÿ Recording files

const char rec_file_name[] = "/JL_REC/FILE0000.MP3"; //MP3 recording file name (including path)

const char rec_file_name[] = "/JL_REC/FILE0000.WAV"; //ADPCM recording file name (including path)

ÿ Recording resource configuration

Since AC692N does not have a memory allocation function, a block of memory needs to be transferred for recording. Calling interface void rec_alloc_init(void *alloc_buffer, u32 buffer_len); By default, MP3 encoding requires 16K resources, WAV encoding 8K RAM is required. But please note that the WAV encoded data is larger and requires a faster device writing speed. Currently Bluetooth recording only supports WAV recording, because the resources under Bluetooth are not enough for MP3 recording.

ÿ Optimize recording frame loss

Due to resource constraints, recording is currently run with minimal resources.

The following methods can be used to optimize the current frame loss phenomenon, provided that resources are available.

1. Increase input and output buffers. void encode_input_buf_len(u32 len); ÿ void encode_output_buf_len(u32 len); is the interface for increasing the input and output buffers respectively. It must be set before initializing the encoder. The resources used for recording must be increased accordingly.

2. Reduce the bit rate of MP3 encoding to reduce the data written to the device. However, you must select an appropriate bit rate.

3. Use two soft interrupts to run the encoding process and the encoding output process separately. However, the priority of the encoding process soft interrupt must be higher than that of the encoding process. Output, thereby reducing the time of running the recording process and improving efficiency. This approach needs to fully consider the interrupt priority of the entire system to prevent Stop other interrupt priority issues,

Chapter 16 Bluetooth BLE Transparent Transmission Function Description

16.1 Purpose

This document mainly describes the AC692x BLE data transparent transmission development method and some issues to pay attention to during development, and provides users with a

This article provides a reference for the development.

BLE is an independent Bluetooth low energy Bluetooth that can be used alone or with a single Bluetooth speaker.

Other development documents and tool usage instructions are common. The following is an explanation of Bluetooth BLE development and use.

16.2 Terms and abbreviations

Tip: List the definitions of specialized terms used in this document and the original phrases of foreign initials.

Abbreviations and terms	explain
BLE (Bluetooth Low Energy)	
GAP (Generic Access Profile)	Universal access app for controlling device connections and broadcasts.
GATT (Generic Attribute Profile)	A general protocol for sending and receiving very short data segments over a Bluetooth connection. In general, these short data segments are called attributes.
ATT(Attribute Protocol)	The attribute layer is the basis of GATT and GAP, which defines the upper layer of the BLE protocol stack. Data structure and organization.
ADV(Advertising)	Discoverable status
SCAN(Scanning)	Discovery Device Status
CONNECT	Device connection status
MASTER	Host, actively initiates Bluetooth connection
SALVE	Slave, in the role of passive Bluetooth connection
SERVER	The server role defined by GATT is the data center; it can be read and written by the client. There is also a notification client function, the SDK corresponding to SLAVE
CLIENT	The client role defined by GATT is to access data; it can read and write to the server. Also receive server notification function, SDK corresponding to MASTER
UUID	It is used to uniquely identify an ID of a feature value.
HANDLE	It is a handle of the corresponding attribute

16.3 BLE Configuration Instructions

1. BLE enabling configuration

```
// optional configurationÿBT_BREDR_EN/BT_BLE_EN/(BT_BREDR_EN|BT_BLE_EN)
#define BLE_BREDR_MODE                  (BT_BREDR_EN|BT_BLE_EN) //Dual mode configuration


//GATT role configuration defined by GAP
#if (BLE_BREDR_MODE&BT_BLE_EN)
    //Optional configuration: 0--server, 1--client
    #define BLE_GAP_ROLE                0
#endif
```

BRDER and BLE can be turned on separately, which is equivalent to single-mode function.

They can also be opened at the same time, which is equivalent to a dual-mode function.

16.4 BLE **Server** Role

The corresponding file le_server_module.c is responsible for module initialization, processing protocol stack events, and command data control sending, etc.

The server role is implemented as a slave in the SDK and is searched and connected by the host (such as a mobile app).

1. Protocol stack initialization

Interface: void ble_stack_config_init(void)

Function: Define protocol stack configuration initialization, profile data registration, event registration processing, read and write registration processing.

2. Profile generation configuration

In the le_server_module.h file, profile_data[] is the profile data used by the server; this structure is created by

The profile tool make_gatt_services generates the profile based on the configuration cfg file. For detailed usage, please refer to the tool documentation

"make_gatt_services Tool Description".

The following figure is a partial screenshot of the CFG configuration and generated profile_data corresponding to the SDK.


```
PRIMARY_SERVICE, 1800
CHARACTERISTIC,  2a00, READ | WRITE | DYNAMIC,

PRIMARY_SERVICE, ae00
CHARACTERISTIC,  ae01, WRITE_WITHOUT_RESPONSE | DYNAMIC,
CHARACTERISTIC,  ae02, NOTIFY,

CHARACTERISTIC,  ae03, WRITE_WITHOUT_RESPONSE | DYNAMIC,
CHARACTERISTIC,  ae04, NOTIFY,

CHARACTERISTIC,  ae05, INDICATE,

CHARACTERISTIC,  ae10, READ | WRITE | DYNAMIC,

PRIMARY_SERVICE, 1812
```

```
const uint8_t profile_data[] = {
    ////////////////////////////////////////////////////
    //
    // 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,

    ////////////////////////////////////////////////////
    //
    // 0x0004 PRIMARY_SERVICE  ae00
    //
    ////////////////////////////////////////////////////
    0x0a, 0x00, 0x02, 0x00, 0x04, 0x00, 0x00, 0x28, 0x00, 0xae,

    /* CHARACTERISTIC,  ae01, WRITE_WITHOUT_RESPONSE | DYNAMIC, */
    // 0x0005 CHARACTERISTIC ae01 WRITE_WITHOUT_RESPONSE | DYNAMIC
    0x0d, 0x00, 0x02, 0x00, 0x05, 0x00, 0x03, 0x28, 0x04, 0x06, 0x00, 0x01, 0xae,
    // 0x0006 VALUE ae01 WRITE_WITHOUT_RESPONSE | DYNAMIC
    0x08, 0x00, 0x04, 0x01, 0x06, 0x00, 0x01, 0xae,

    /* CHARACTERISTIC,  ae02, NOTIFY, */
    // 0x0007 CHARACTERISTIC ae02 NOTIFY
    0x0d, 0x00, 0x02, 0x00, 0x07, 0x00, 0x03, 0x28, 0x10, 0x08, 0x00, 0x02, 0xae,
    // 0x0008 VALUE ae02 NOTIFY
    0x08, 0x00, 0x10, 0x00, 0x08, 0x00, 0x02, 0xae,
    // 0x0009 CLIENT_CHARACTERISTIC_CONFIGURATION
    0x0a, 0x00, 0x0a, 0x01, 0x09, 0x00, 0x02, 0x29, 0x00, 0x00,

    /* CHARACTERISTIC,  ae03, WRITE_WITHOUT_RESPONSE | DYNAMIC, */
    // 0x000a CHARACTERISTIC ae03 WRITE_WITHOUT_RESPONSE | DYNAMIC
    0x0d, 0x00, 0x02, 0x00, 0x0a, 0x00, 0x03, 0x28, 0x04, 0x0b, 0x00, 0x03, 0xae
```

3. Broadcast configuration initialization

Interface: void advertisements_setup_init()

Function: Define the advertising cycle, advertising packet content and scan response packet content.

The broadcast cycle parameter is in units of 0.625ms, as shown below:

```
#define ADV_INTERVAL_MIN          160
#define ADV_INTERVAL_MAX          160
```

The composition of the broadcast packet, the user can modify and add types, as shown below:

```
static int make_set_adv_data(void)
{
    u8 offset = 0;
    u8 *buf = adv_data;

    offset += make_eir_packet_val(&buf[offset], offset, HCI_EIR_DATATYPE_FLAGS, 6, 1);
    offset += make_eir_packet_val(&buf[offset], offset, HCI_EIR_DATATYPE_COMPLETE_16BIT_SERVICE_UUIDS, 0x1812, 2);

    if (offset > sizeof(adv_data)) {
        puts("***adv_data overflow!!!!!!\n");
        return -1;
    }
    /* printf_buf(adv_data,offset); */
    gap_advertisements_set_data(offset, (uint8_t *)adv_data);
    return 0;
}
```

The scan response package consists of two types, which can be modified and added by the user, as shown below:

```
static int make_set_rsp_data(void)
{
    u8 offset = 0;
    u8 *buf = scan_rsp_data;

    u8  tag_len = sizeof(user_tag_string);
    offset += make_eir_packet_data(&buf[offset], offset, HCI_EIR_DATATYPE_MANUFACTURER_SPECIFIC_DATA, (void *)user_tag_string, tag_len);

    u8 name_len = strlen(gap_device_name);
    u8 vaild_len = ADV_RSP_PACKET_MAX - (offset + 2);
    if (name_len > vaild_len) {
        name_len = vaild_len;
    }
    offset += make_eir_packet_data(&buf[offset], offset, HCI_EIR_DATATYPE_COMPLETE_LOCAL_NAME, (void *)gap_device_name, name_len);

    if (offset > sizeof(scan_rsp_data)) {
        puts("***rsp_data overflow!!!!!!\n");
        return -1;
    }
    /* printf_buf(scan_rsp_data,offset); */
    gap_scan_response_set_data(offset, (uint8_t *)scan_rsp_data);
    return 0;
}
```

4. Protocol stack event processing

Interface: void cbk_packet_handler(...)

Function: Processing event message processing of protocol stack callback, such as disconnection, sending completion, etc.

The protocol stack initialization completes event processing, as shown below:

```
case BTSTACK_EVENT_STATE:
    if (btstack_event_state_get_state(packet) != HCI_STATE_WORKING) {
        break;
    }
    log_info("init complete\n");
    set_ble_work_state(BLE_ST_IDLE);
```

Connection success event processing to confirm that the device has been connected, as shown below:

```
case HCI_SUBEVENT_LE_CONNECTION_COMPLETE: {
    set_adv_enable(0, 0);
    con_handle = little_endian_read_16(packet, 4);
    log_info("HCI_SUBEVENT_LE_CONNECTION_COMPLETE : %0x\n", con_handle);
```

Disconnection event processing to confirm that the device has been disconnected, as shown below:

```
case HCI_EVENT_DISCONNECTION_COMPLETE:
    log_info("---ble disconnect!\n");
    ble_test_flag = 0;
    con_handle = 0;
```

5. Read and **write** callback processing

When the read and write definition attributes in the profile's cfg configuration file have the DYNAMIC keyword, there will be a callback process.

Interface: uint16_t att_read_callback(...)

Function: Return the corresponding read content according to the read operation handle.

When buffer is NULL, returns the total length of the read attribute.

When buffer is not NULL, the content obtained by read is returned.

```
static uint16_t att_read_callback(hci_con_handle_t connection_handle, uint16_t att_handle, uint16_t offset,
```

Interface: int att_write_callback(...)

Function: According to the write operation handle, receive the write content, including enabling the switch notify&indicate function.

```
static int att_write_callback(hci_con_handle_t connection_handle, uint16_t att_handle, uint16_t transaction_mode,
```

6. Connection parameter update

The slave can request to change the connection parameters, but the master must agree to the change before it can be successful.

The main three steps are as follows:

(1) The slave sends a parameter request, as shown below:

```
static void send_request_connect_parameter(u8 table_index)
{
    struct conn_update_param_t *param = (void *)&connection_param_table[table_index];

    log_info("update_request:-%d-%d-%d-%d-\n", param->interval_min, param->interval_max, param->latency, param->timeout);
    if (con_handle) {
        gap_request_connection_parameter_update(con_handle, param->interval_min, param->interval_max, param->latency, param->timeout);
    }
}
```

```
static const struct conn_update_param_t connection_param_table[] = {
    {16, 24, 0, 600},//11
    {12, 28, 0, 600},//3.7
    {8,  20, 0,  600},
```

The connection parameter interval is in units of 1.25ms, and the timeout is in units of 10ms. Users should modify it carefully.

Modifications are defined according to the Bluetooth protocol specification.

(2) The host responds to the request result and the event is called back, as shown below:

```
case L2CAP_EVENT_CONNECTION_PARAMETER_UPDATE_RESPONSE:
    tmp = little_endian_read_16(packet, 4);
    log_info("-update_rsp: %02x\n", tmp);
    if (tmp) {
        connection_update_cnt++;
        log_info("remoter reject!!!\n");
        check_connetion_updata_deal();
    } else {
        connection_update_cnt = CONN_PARAM_TABLE_CNT;
    }
    break;
```

(3) The connection parameters are changed successfully and the event is called back, as shown below:

```
case HCI_SUBEVENT_LE_CONNECTION_UPDATE_COMPLETE:
    connection_update_complete_success(packet);
    break;
```

7. Provide interface calls to external parties

The interface includes broadcast enable, disconnect, get send buffer space, send data, etc. Please check the specific implementation of the interface for details.

```
struct ble_server_operation_t {
    int(*adv_enable)(void *priv, u32 enable);
    int(*disconnect)(void *priv);
    int(*get_buffer_vaild)(void *priv);
    int(*send_data)(void *priv, void *buf, u16 len);
    int(*regist_wakeup_send)(void *priv, void *cbk);
    int(*regist_recieve_cbk)(void *priv, void *cbk);
    int(*regist_state_cbk)(void *priv, void *cbk);
};
void ble_get_server_operation_table(struct ble_server_operation_t **interface_pt);
```

```
static const struct ble_server_operation_t mi_ble_operation = {
    .adv_enable = set_adv_enable,
    .disconnect = ble_disconnect,
    .get_buffer_vaild = get_buffer_vaild_len,
    .send_data = (void *)app_send_user_data_do,
    .regist_wakeup_send = regiest_wakeup_send,
    .regist_recieve_cbk = regiest_recieve_cbk,
    .regist_state_cbk = regiest_state_cbk,
};
```

Note: The data sending interface uses the server's notify&indicate notification function to implement, which requires the host client to enable

Only when notification is turned on can data be sent out.

16.5 BLE CLIENT Role

The corresponding file le_client_module.c is responsible for module initialization, processing protocol stack events, and command data control sending, etc.;

The client role is implemented in the SDK as a host master, which actively initiates searches and connections to other BLE devices.

The profile list data.

The SDK example uses the profile in Jie Li's server as the search target, and users can also search and filter according to their needs.

profileÿ

1. Protocol stack configuration initialization

Interface: void ble_stack_config_init(void)

Function: Define protocol stack configuration initialization, registration event processing and process processing.

2. Protocol stack event processing

Interface: void cbk_packet_handler(...)

Function: Processing event message processing of protocol stack callback, such as disconnection, sending completion, etc.

The protocol stack initialization completion event is as shown below:

```
case BTSTACK_EVENT_STATE:
    if (btstack_event_state_get_state(packet) != HCI_STATE_WORKING) {
        break;
    }
    log_info("init complete\n");
    set_ble_work_state(BLE_ST_IDLE);
    client_scan_set_param(1, SET_SCAN_INTERVAL, SET_SCAN_WINDOW);
    ACT_SET_FLAG(CLI_SCAN_ENABLE);
    break;
```

Connection success event processing to confirm that the device has been connected, as shown below:

```
case HCI_SUBEVENT_LE_CONNECTION_COMPLETE: {
    if (tc_state != TC_W4_CONNECT) {
```

Disconnection event processing to confirm that the device has been disconnected, as shown below:

```
case HCI_EVENT_DISCONNECTION_COMPLETE:
    tc_state = TC_IDLE;
    gc_handle = 0;
    set_ble_work_state(BLE_ST_IDLE);
    client_puts("client disconn\n");
    ACT_SET_FLAG(CLI_SCAN_ENABLE);
```

Scan and return broadcast event processing, as shown below:

```
case GAP_EVENT_ADVERTISING_REPORT:
    /* client_puts("GAP_LE_ADVERTISING_REPORT\n"); */
    /* putchar('V'); */
    if (tc_state != TC_W4_SCAN_RESULT) {
        client_printf("SCAN ACTIVE err,%d\n", tc_state);
        break;
    }
    client_report_adv_data((void *)&packet[2], packet[1]);
    break;
```

3. Create connection control

After enabling scanning, you can broadcast information based on the searched devices and initiate a connection.

Device information return interface void client_report_adv_data(adv_report_t *report_pt, u16 len) .

When creating a connection, you can initiate a connection to the corresponding device based on the name, address, or ID. Users can modify the conditions for creating a connection.

The following are the optional conditions:

```
enum {
    CLI_CREAT_BY_ADDRESS = 0,
    CLI_CREAT_BY_NAME,
    CLI_CREAT_BY_TAG,
    CLI_CREAT_BY_LAST_SCAN,
};
```

```
static const u8 create_conn_mode = BIT(CLI_CREAT_BY_LAST_SCAN);// BIT(CLI_CREAT
static const u8 create_conn_remoter[6] = {0x11, 0x22, 0x33, 0x88, 0x88, 0x88};
static const u8 create_remoter_name[32] = "AC692x_testAa18";
static u8 adv_last_remoter_name[32];
static u8 conn_last_remoter_flag = 0;
static u8 create_conn_flag = 0;
```

After finding the corresponding device, you can initiate the action of creating a connection, as shown below:

```
if (find_remoter) {
    if (create_conn_flag) {
        puts("\n*********creat_connection**********\n");
        printf("***remote type %d,addr:", report_pt->address_type);
        put_buf(report_pt->address, 6);
        puts("\n");
        create_conn_flag = 0;
        client_creat_connection(report_pt->address, report_pt->address_type);
        ACT_SET_FLAG(CLI_SCAN_DISABLE);
    }
}
```

4. Find the corresponding **UUID** and **HANDLE**

The client searches for the target's UUID, records the corresponding HANDLE, and then uses the HANDLE to call the server.

If there is notify&indicate, the notification will be automatically enabled.

Function.

The search service operates in 4 steps, as described below.

(1) Set the UUID of the search target, as shown below:

```
static const target_service_t target_services = {
    .service_uuid16 = 0xae00,
    .read_uuid16 = 0xae10,
    .write_uuid16 = 0xae01,
    .notify_uuid16 = 0xae02,
    .indicate_uuid16 = 0,
};
```

(2) Start the service search operation, as shown below:

```
static u16  client_search_profile_start(u8 search_pfl_type, u32 search_data)
{
    u16 res = 0;
```

(3) Event processing during the profile search process, as shown in the following figure:

```
static void s_client_gatt_event(uint8_t packet_type, uint16_t channel, uint8_t *packet, uint16_t size)
{
    u32 tmp32, ret_type;
    /* printf("client cbk gatt event %x \n",tc_state);   */
    switch (tc_state) {
```

(4) Search for profile and complete the operation, as shown below:

```
static void client_search_profile_complete(void)
{
    u16 mtu;
```

5. Scan parameter and connection parameter configuration

The scanning parameters are set in units of 0.625ms as shown below:

```
#define SET_SCAN_WINDOW     16
#define SET_SCAN_INTERVAL   48
```

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

```
#define SET_CONN_INTERVAL_MIN   0x10
#define SET_CONN_INTERVAL_MAX   0x20
#define SET_CONN_TIMEOUT        0x180
```

Please modify the above two sets of parameters carefully and must define and modify them according to the Bluetooth protocol specifications.

6. Provide interface calls to external parties

The interface includes search enable, disconnect, get send buffer space, send data, etc. Please check the specific implementation of the interface for details.

```
struct ble_client_operation_t {
    int(*scan_enable)(void *priv, u32 enable);
    int(*disconnect)(void *priv);
    int(*get_buffer_vaild)(void *priv);
    int(*write_data)(void *priv, void *buf, u16 len);
    int(*read_do)(void *priv);
    int(*regist_wakeup_send)(void *priv, void *cbk);
    int(*regist_recieve_cbk)(void *priv, void *cbk);
    int(*regist_state_cbk)(void *priv, void *cbk);
};
void ble_get_client_operation_table(struct ble_client_operation_t **interface_pt);
```

```
static const struct ble_client_operation_t client_operation = {
    .scan_enable = app_client_scan_enable,
    .disconnect = app_client_disconn,
    .get_buffer_vaild = app_client_get_buffer_vaild_len,
    .write_data = (void *)app_client_write_without_respond_send,
    .read_do = (void *)app_client_read_send,
    .regist_wakeup_send = app_client_regiest_wakeup_send,
    .regist_recieve_cbk = app_client_regiest_recieve_cbk,
    .regist_state_cbk = app_client_regiest_state_cbk,
};


void ble_get_client_operation_table(struct ble_client_operation_t **interface_pt)
{
    *interface_pt = (void *)&client_operation;
}
```

Note: The data sending interface is implemented using the client's write function.

16.6 BLE ATT sending mechanism

The corresponding file att_send.c is compatible with the data sending calls of the server and client roles, and is responsible for the ATT command read,

Write, notify and indicate commands are sent and controlled. The working principle is to use cbuff to cache ATT sending commands.

According to the sending mechanism of the protocol stack, data is filled in and sent in real time.

1. Configure the cache size of **cbuff .** The size can be modified according to needs, as shown in the following figure:

```
#define USER_BUF_LEN          (512*4)      //发送buf大小，可根据需求修改
cbuffer_t user_send_cbuf;
static u8 user_data_buf[USER_BUF_LEN] __attribute__((aligned(4)));
```

2. Get the remaining space size **of cbuff ,** as shown below:

```
u32 user_data_cbuf_is_write_able(u32 len)
{
    u32 buf_space, w_len;
    u16 head_size = sizeof(user_send_head_t);
    u16 pack_size = sizeof(user_send_head_t) + att_payload_size;

    if (!att_conn_handle) {
        return 0;
    }
```

3. Get whether **cbuff** is empty, as shown below:

```
u32 user_data_cbuf_is_null(void)
{
    return !cbuf_get_data_size(&user_send_cbuf);
}
```

4. User data sending interface, as shown below:

```
u32 user_data_att_send(u16 handle, u8 *data, u16 len, u8 send_type)
{
    u16 wlen;
    u32 ret_val;
    user_send_head_t send_head;
    u16 packet_cnt = 0;
```

Appendix A: Functional Differences between **AC692x ,** AC690x, and AC460x

AC692x与AC690x、AC460x功能差异				
序号	功能	AC692x	AC690x	AC460x
1	内置FM模式下蓝牙跑后台	不支持	支持	支持
2	K歌宝：混响模式下MIC和AUX是否支持同时录音	不支持	支持	支持
3	MIC和AUX是否支持分开独立调音量	不支持	支持	支持
4	内置充电	不支持	支持	不支持
5	对耳，对箱	支持	支持	不支持
6	DC-DC	不支持	支持	不支持
7	NFC	不支持	支持	支持
8	在线调EQ	支持	支持	不支持
9	普通IO口内部上下拉	上拉10K，下拉10K	上拉10K，下拉60K	上拉10K，下拉60K
10	SPDIF（数字音频输出）	支持	不支持	不支持

Appendix **B JL** Bluetooth Call Debugging Instructions

Documentation

This document is used to explain how to debug Bluetooth calls and is applicable to most scenarios.

Debugging instructions

1. **dac_analog_gain: Debug the volume of the near-end prototype speaker (** Analog Gain ÿ

2. **mic_analog_gain: Debug the volume heard by the remote phone (** Analog Gain ÿ

3. NDT_max_gain: The maximum digital gain of the sound volume heard by the remote phone (Magnification *=NDT_MAX_gain/64ÿ*

NDT_min_gain: The minimum digital gain of the sound volume heard by the remote phone (Magnification *=NDT_min_gain/64ÿ*

NDT_Fade_Speed: fade in and fade out speed

NearEnd_Begin_Threshold: Amplification threshold

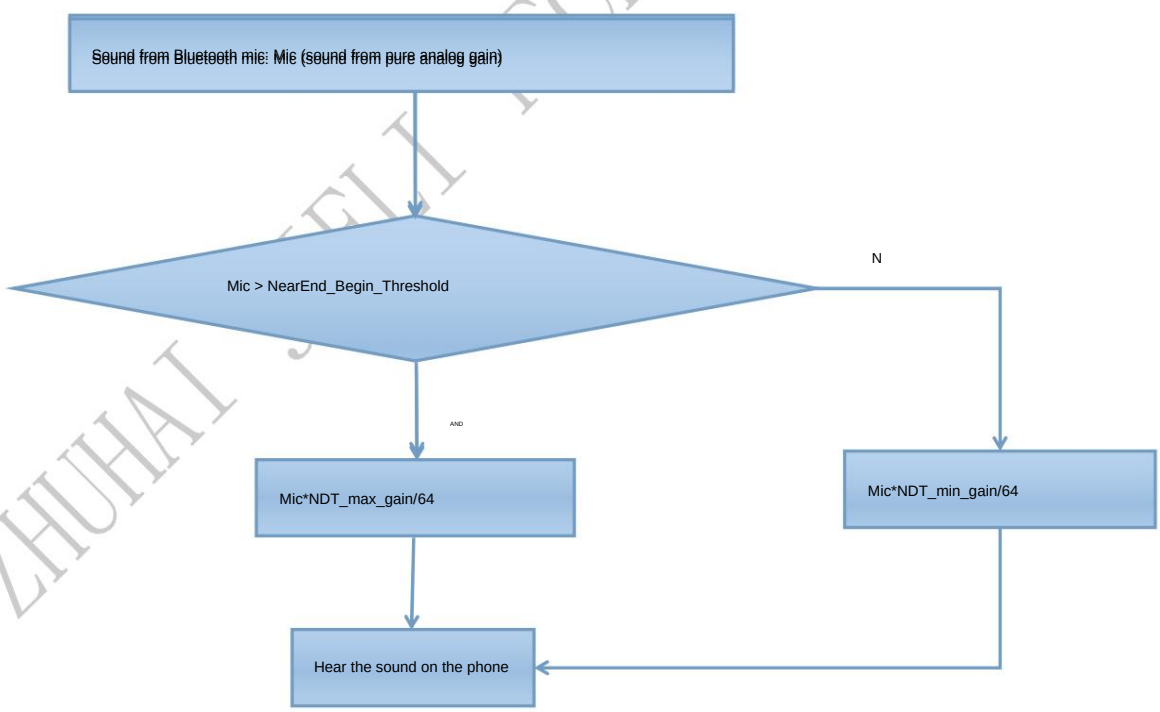
The above four parameters are digital parameters and are used together to adjust the sound effect heard by the remote phone.

Directly using the analog gain of the mic to control the volume is because the analog gain of the mic is too large, 1 means the echo is too large and difficult to handle;

2. The surrounding environmental sounds are easily collected, affecting the listening experience.

Note: The gain of the mic cannot be too small, otherwise the clarity will be insufficient.

The operation process is as follows:



If the NearEnd_Begin_Thrashold threshold is too large, it is easy to zoom in. If it is too small, the far-end phone can easily hear the near-end ambient sound.

The program increases or decreases in units of 10 to 20, and the specific amount depends on the analog gain of the mic.

NDT_Fade_Speed indicates the fade speed between NDT_Max_gain and NDT_Min_gain

4ÿaec_ctl

AEC_ADVANCE: Enables all operation modules, suitable for solutions with large echo, such as speakers

AEC_REDUCE: Turn off some computing modules (save power consumption and improve duplex effect), suitable for solutions with small echo, such as headphones

5. suppress_coeff1: echo suppression coarse adjustment (suppression level), the effect is more obvious

The larger the value, the greater the echo suppression, and the corresponding call is more likely to be intermittent, because the voice of the person speaking may also be suppressed.

The smaller the value, the smaller the echo suppression, and the corresponding call quality will be improved and become more uniform. If it is too small, there may be echo.

The debugging process is performed in 1-bit units. The recommended debugging range is (0~5)

6. suppress_coeff2: echo suppression fine-tuning (clearing threshold), mainly used to optimize call quality

The effect of this value is the same as suppress_coeff1, and the value is proportional to the echo suppression strength.

If it is a headset solution, this parameter should be set to 0 to achieve duplex effect.

The debugging process is incremented and decremented in units of 100, and the recommended range is (500~1500)

Note 1: Among the above parameters, only parameter 1 (dac_analog_gain) is used to adjust the volume of the near-end prototype speaker. The rest are Used to adjust the effect heard by the remote phone.

Note 2: The volume heard by the remote phone can be adjusted by analog gain and digital gain. The difference is that analog gain can be increased Clarity and resolution, too much will cause echo or easy to hear the near-end environment. Digital gain will not cause echo, but if the analog The gain is too small, and the clarity will be a little worse if it is amplified only by digital gain.