机芯HBM 离线方案开发指导手册 v1.0.1

ReleaseNote

版本历史记录

版本：V1.0.1，发布时间：2020/5/11

适配环境：

- 硬件：机芯HBM开发板
- 软件：hb_m v3.0.5

修改记录：

1.二次开发章节中增加串口 log 打印说明

版本：V1.0.0，发布时间：2020/4/30

适配环境：

- 硬件：机芯HBM开发板
- 软件：hb_m v3.0.1

修改记录：

1.第一版发布

机芯HBM 离线方案开发指导手册 v1.0.1

机芯HBM 离线方案开发指导手册 v1.0.1

2、产品说明

1. 硬件说明

机芯HBM芯片采用高性能 32 位 RISC 内核，最高频率 240MHz，支持 DSP 指令，集成 FPU 支持浮点运算，通过神经网络对音频信号进行训练学习，提高语音信号的识别能力。

机芯HBM芯片具有丰富的系统外设，包括 UART, I2C, SPI, PWM, RTC, Timer, ADC, DAC 等，是一款低功耗高性能，主控级别的离线语音识别芯片。

机芯HBM 芯片与通用demo 板

机芯HBM芯片主要特性：

配置	规格
机芯HBM芯片	高速 SRAM
	FLASH 2MB
mic 数量	1
UART 通信接口	1
LED 接口	有

音频接口	I2S, ADC, DAC
控制接口	UART, GPIO, I2C, SPI, PWM, ADC, DAC 等

2. 软件功能

序号	功能	功能描述
1	**单麦拾音**	• 单麦克风方案 • 支持家居场景 5m 远讲
2	**语音唤醒**	• 高性能唤醒引擎 • 低功耗 • 支持带口音的普通话 • 低误唤醒率 (< 1 false in 48 hours)
3	**离线识别**	• 支持本地 100 条控制指令识别
4	**多轮对话**	一次唤醒连续对话，语音操作更加便捷自然。
5	**多种发音人音色**	• 提供标准女声、甜美女声、可爱女声、台湾女声、标准男声、女童声、男童声七种音色可选
6	**UART 主板对接**	提供标准 UART 协议，也支持对接用户自有协议
7	**智能设备平台**	设备平台提供：**唤醒词自定义**，**命令词自定义**，**回复播报语自定义**，**发音人音色选择**等产品自定义配置

机芯HBM 离线方案开发指导手册 v1.0.1

3、获取源码

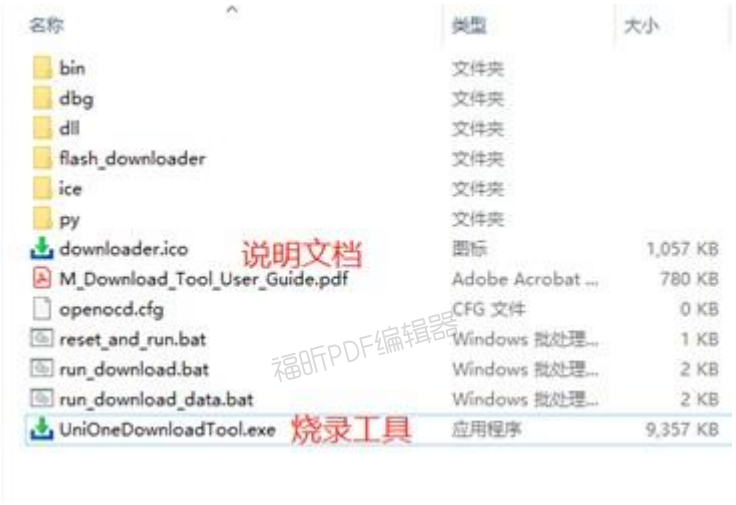
机芯HBM 离线方案开发指导手册 v1.0.1

4、烧录固件

1. 文件介绍

- 产品的软件烧录包下载成功后，解压文件，可以看到如下方文件目录

- 如想直接使用，体验语音功能，请打开“image_demo/Hummingbird-M-Production-Tool”文件夹，双击“UniOneDownloadTool”，运行烧录工具



- 烧录工具配置界面如下图所示：

2. 烧录指南

2.1 烧录前介绍

开始烧录之前，需要准备：

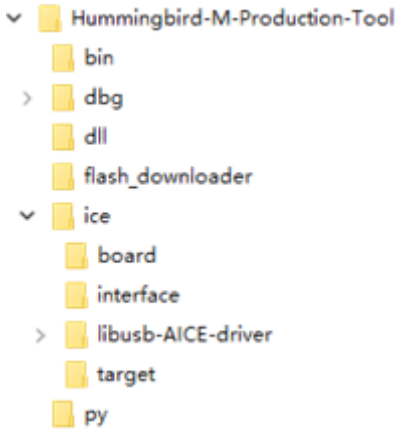
- Micro USB 连接线，用于供电
- 烧录器，用于烧录
- 机芯HBM 开发板
- 固件包

烧录器

蜂鸟开发板示例图

开始烧录之前，需要准备：

打开烧录包，Hummingbird-M-Production-Tool 内，可看到以下目录结构：



和用户相关的目录有以下几个：

- \bin： 用户存放 bin/mva 文件的目录
- \flash_downloader : flash downloader 存放目录，需要提取相应芯片 SDK 中提供的 downloader.exe
- \ice\libusb-AICE-driver： 仿真器驱动

名称	类型	大小
bin	文件夹	
dbg	文件夹	
dll	文件夹	
flash_downloader	文件夹	
ice	文件夹	
py	文件夹	
build.properties	PROPERTIES 文件	10 KB
demo_ap82.png	PNG 文件	16 KB
downloader.ico	图标	1,057 KB
openocd.cfg	CFG 文件	0 KB
readme.pdf	Adobe Acrobat ...	874 KB
reset_and_run.bat	Windows 批处理...	1 KB
run_download.bat	Windows 批处理...	2 KB
run_download_data.bat	Windows 批处理...	2 KB
UniOneDownloadTool.exe	应用程序	9,357 KB

根目录有以下用户相关的文件：

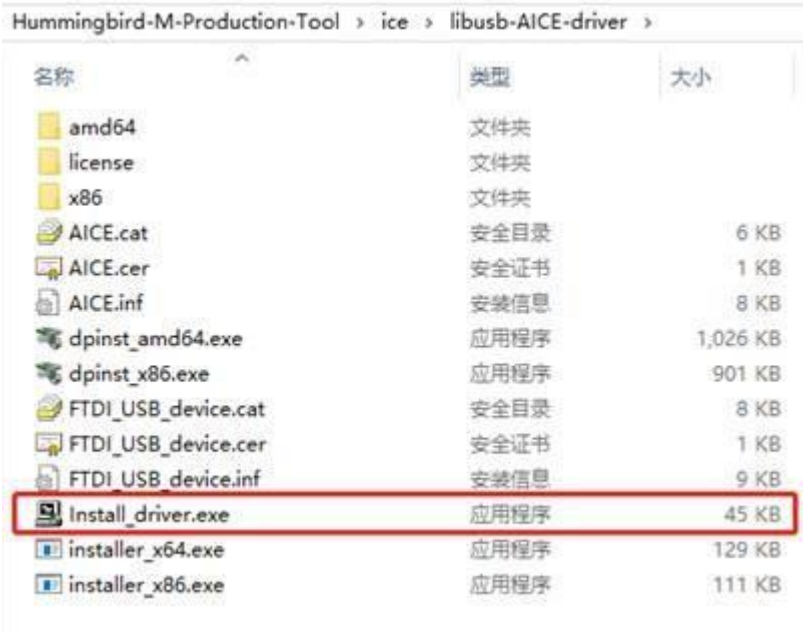
- UniOneDownloadTool_x64.exe： 烧录工具启动 exe
- run_download.bat： 批量执行脚本例程

2.2 安装驱动（如已安装，可跳过）

插入烧录器，如在设备管理器看到未知设备，则需要安装驱动。

1、 准备固件：发布的固件包，如 Hummingbird-M-Production-Tool

2、 打开 Hummingbird-M-Production-Tool\ice\libusb-AICE-driver，双击 Install_driver.exe，点击同意安装



3、 处于设备管理器页面，右击 AICE（未知设备），更新驱动程序

4、 选择浏览计算机查找驱动

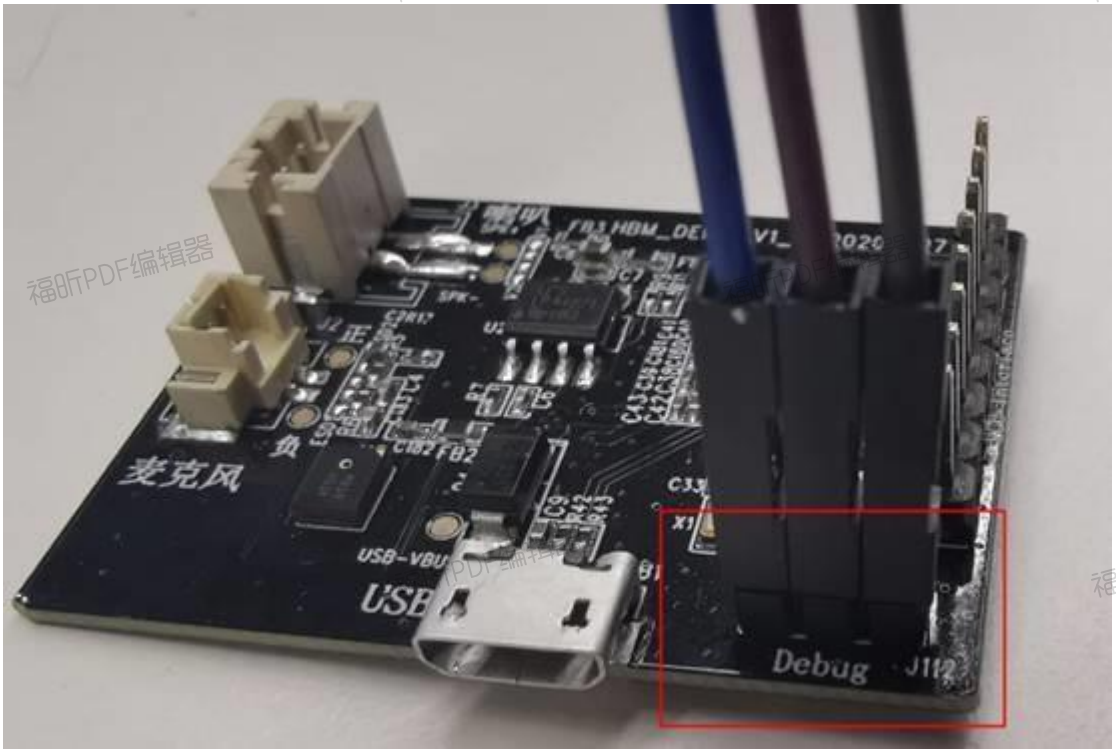
驱动路径： Hummingbird-M-Production-Tool\ice\libusb-AICE-driver

5、 点击确定后，回到更新驱动程序页面，点击下一步，等待驱动安装成功。

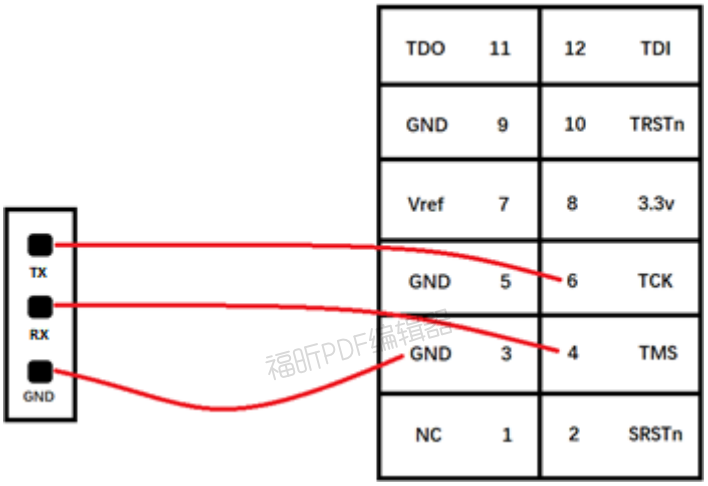
2.3 进入烧录模式

2.3.1 接入烧录器

在开发板上找到“Debug”接口，该接口有三根信号线，按开发板上丝印分别为 GND、RX、TX.



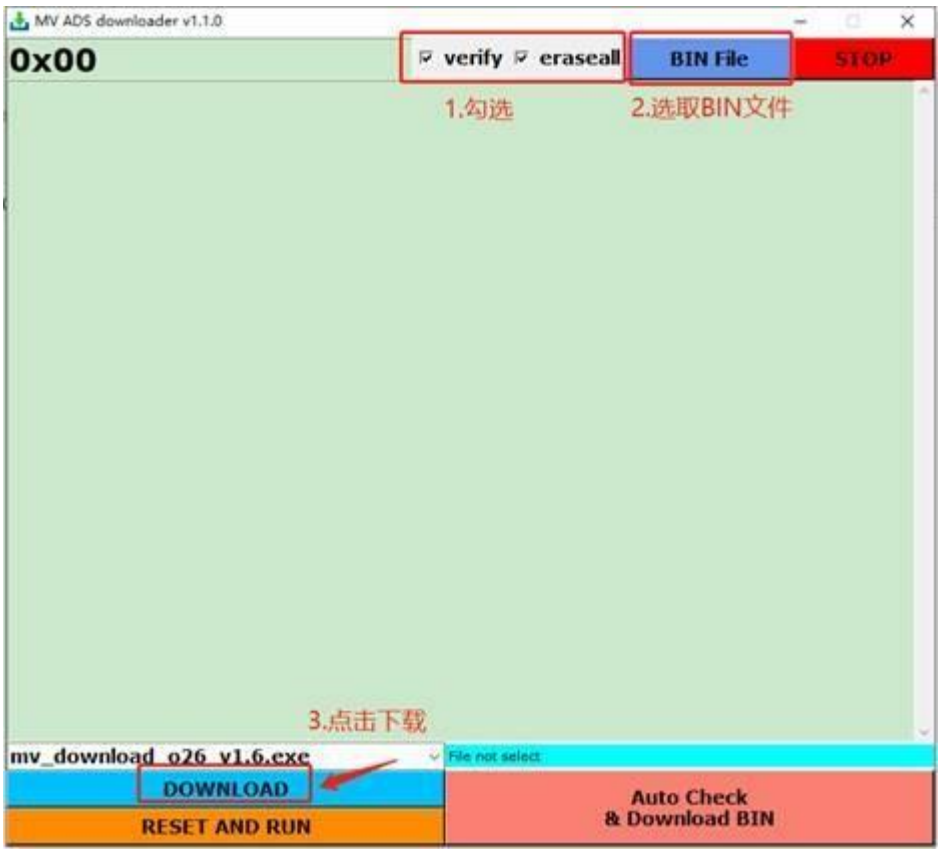
三个接口分别对应烧录器上 GND（3 号脚）、TMS（4 号脚）、TCK(6 号脚），接线示意图如下所示：



2.3.2 开发板供电，开始烧录

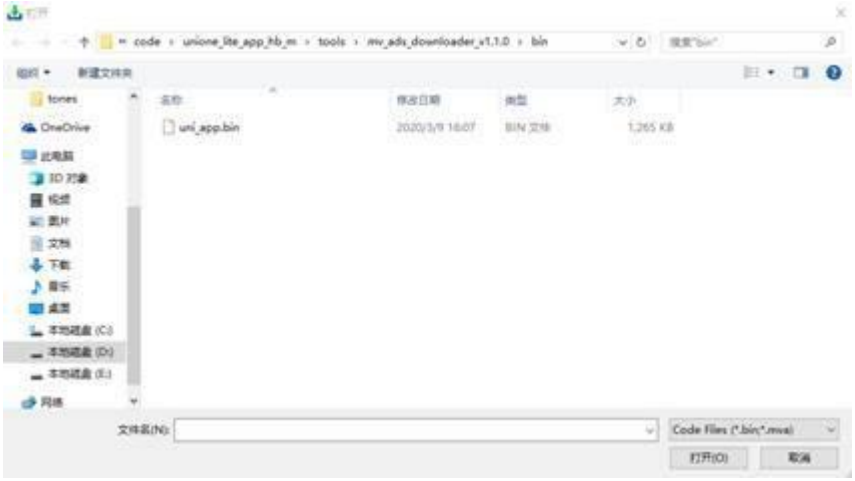
使用 USB 连接线给开发板上电，即接入电源。可直接接电脑，或者任意 5V 电源适配器（如手机充电器）。

点击烧录工具界面“BIN File”按钮，选择待烧录 bin 文件：

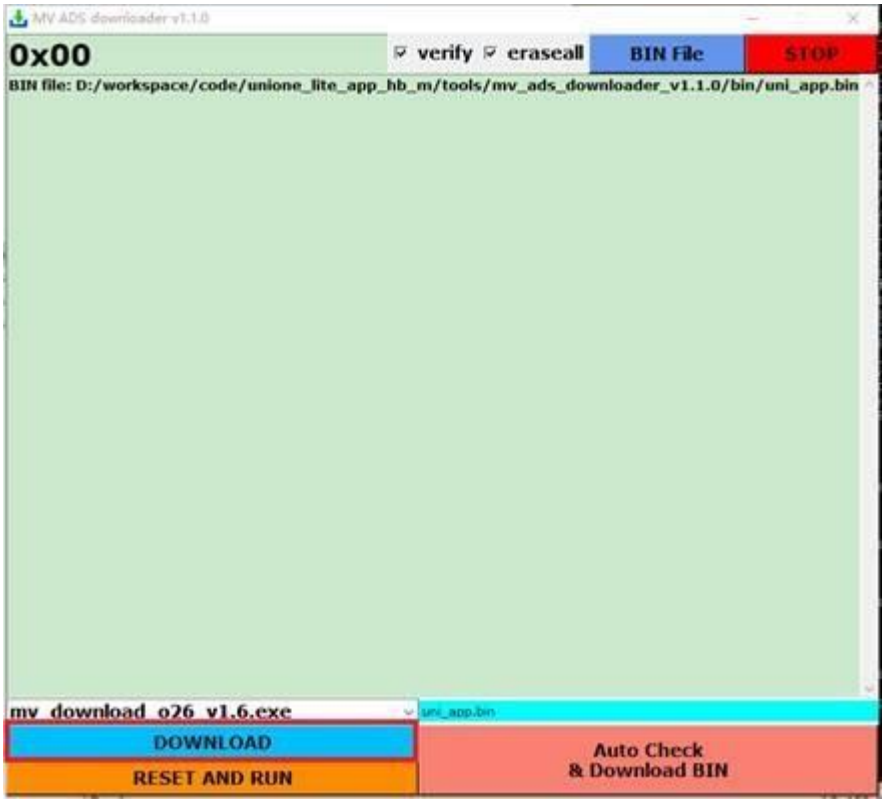


默认选择路径为“image_demo/ Hummingbird-M-Production-Tool /bin”，也可通过文件浏览器选择其他路径：

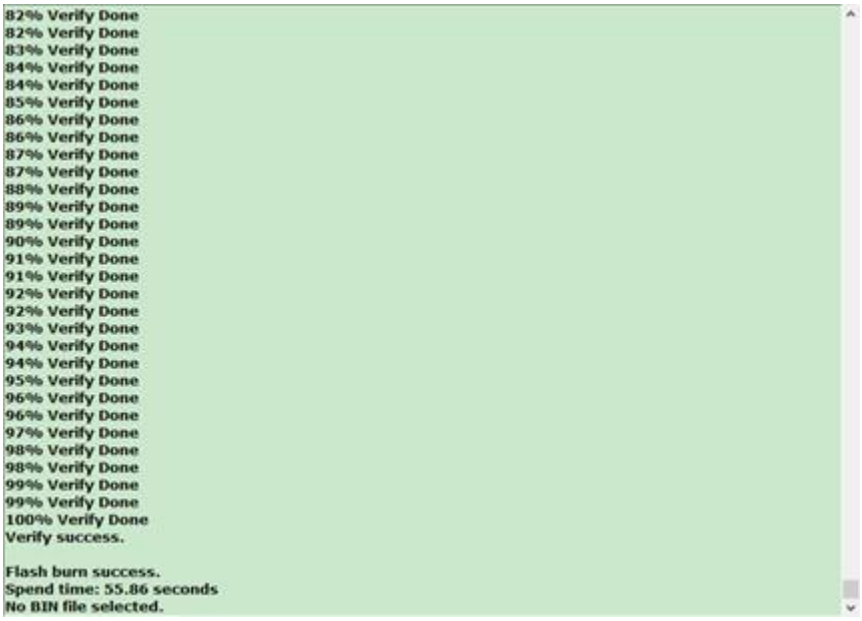
注：目前发布包中同时包含 uni_app_debug.bin 和 uni_app_release.bin 两个版本，其中 release 为正式版本，debug 为带 log 调试版本，与 release 版本相比响应速度较慢，仅用于调试使用。



选择烧录 bin 文件后，点击烧录工具界面“DOWNLOAD”按钮，开始烧录：



烧录成功则烧录工具界面打印烧录进度，直到打印“Verify success”信息：



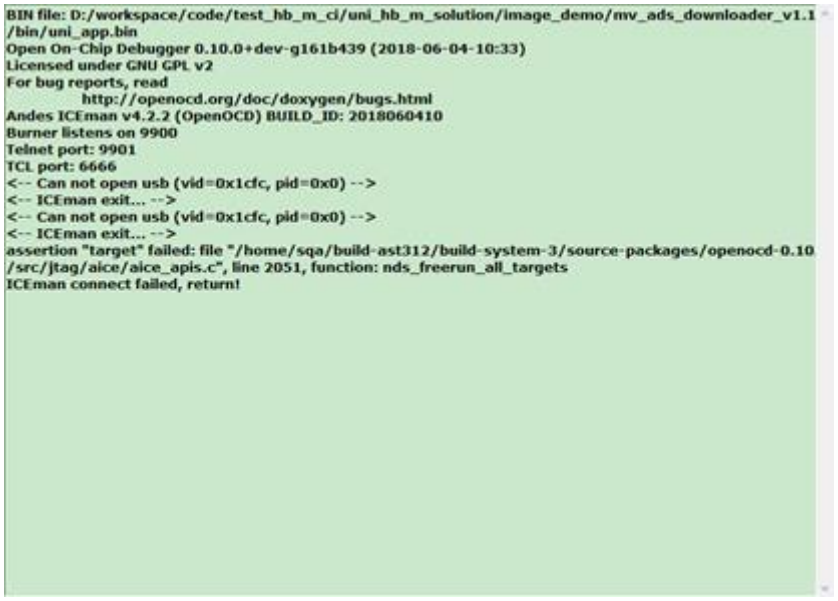
如未显示则表示未识别到 COM 口，请检查：

① 串口线是否接好

② 串口线是否反接

③ 是否安装 UART 对应驱动。

而烧录失败界面将输出错误信息，如下图：



如烧录失败，请检查：

① 开发板是否供电

② 开发板 SW Debug 接口是否接好

③ SW 线是否按照要求反接，确认后请回到步骤 1 重新短接上电，重新进入烧录模式

④ 其他错误，请按照烧录界面提示信息排查问题

机芯HBM 离线方案开发指导手册 v1.0.1

5、搭建开发环境

1 环境准备

（1） 下载编译工具

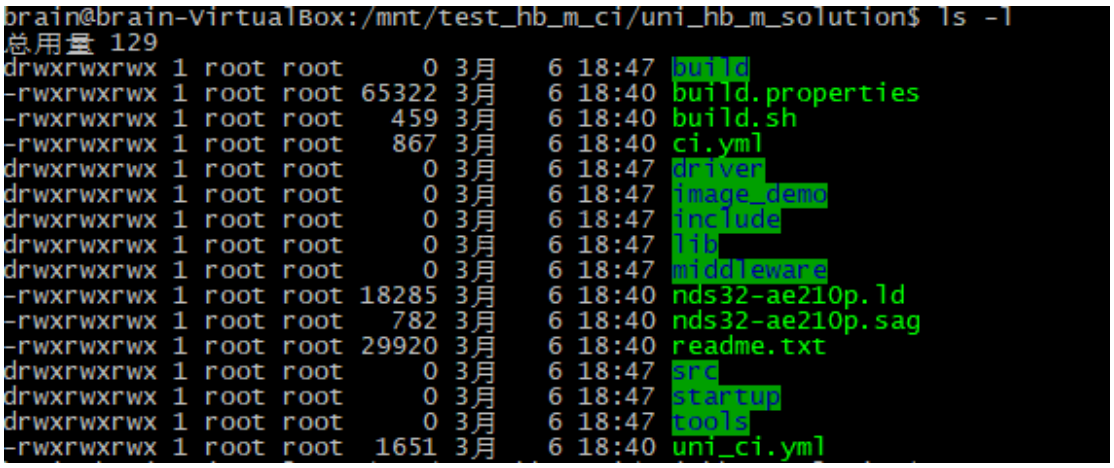
推荐使用 Ubuntu 16.04 / centos7 以上版本作为开发环境。

交叉编译工具下载地址：链接：https://pan.baidu.com/s/1EHa10q7xkoySt5MA8_abew

提取码：cdbx

下载交叉编译工具并解压到开发环境“/opt”路径中，或其他路径，然后修改 makefile 文件中配置选项：

（2） 已下载的版本里除了可直接烧录使用的 image_demo 外，文件夹内包含产品方案源码，如下图所示：



（3） 修改 build/makefile 中交叉工具链的路径为实际解压到的工具路径；

```
################################################################################
# Automatically-generated file. Do not edit!
################################################################################

CODE_ROOT=../

ifndef CROSS_COMPILE
CROSS_COMPILE=/opt/Andestech/BSPv422/toolchains/nds32le-elf-mculib-v3s/bin/nds32le-elf-
endif

ifndef SECONDARY_OUTPUT_PATH
SECONDARY_OUTPUT_PATH=${CODE_ROOT}/output
endif

$(shell mkdir -p $(SECONDARY_OUTPUT_PATH))

-include ../makefile.init

RM := rm -rf

# All of the sources participating in the build are defined here
-include sources.mk
-include subdir.mk
-include driver/driver_api/src/subdir.mk
-include middleware/audio/src/subdir.mk
-include middleware/fatfs/src/subdir.mk
-include middleware/mv_utils/src/subdir.mk
-include middleware/rtos/freertos/src/subdir.mk
-include middleware/rtos/rtos_api/subdir.mk
-include src/app/src/subdir.mk
-include src/app/src/sessions/subdir.mk
-include src/hal/src/subdir.mk
-include src/sdk/audio/audio_player/src/subdir.mk
-include src/sdk/idle_detect/src/subdir.mk
-include src/sdk/player/src/pcm/src/subdir.mk
-include src/sdk/player/src/subdir.mk
-include src/sdk/uart/src/subdir.mk
-include src/sdk/vui/src/subdir.mk
-include src/utils/arpt/src/subdir.mk
-include src/utils/auto_string/src/subdir.mk
-include src/utils/bitmap/src/subdir.mk
-include src/utils/black_board/src/subdir.mk
-include src/utils/cJSON/src/subdir.mk
-include src/utils/config/src/subdir.mk
-include src/utils/crc16/src/subdir.mk
-include src/utils/data_buf/src/subdir.mk
-include src/utils/event/src/subdir.mk
-include src/utils/event_list/src/subdir.mk
-include src/utils/event_route/src/subdir.mk
-include src/utils/float2string/src/subdir.mk
-include src/utils/fsm/src/subdir.mk
-include src/utils/hash/src/subdir.mk
-include src/utils/interruptable_sleep/src/subdir.mk
"build/makefile" 152 lines, 5106 characters
```

（4）安装 python 环境：sudo apt-get install python

（5）安装 lame 音频处理工具：sudo apt-get install lame

（6）安装 32 位兼容库（如果系统未安装过）：sudo apt-get install lib32stdc++6 lib32z1 lib32ncurses5 lib32bz2-1.0（仅适用于 Ubuntu 16.04 版本，其他版本请根据系统说明解决兼容性问题）

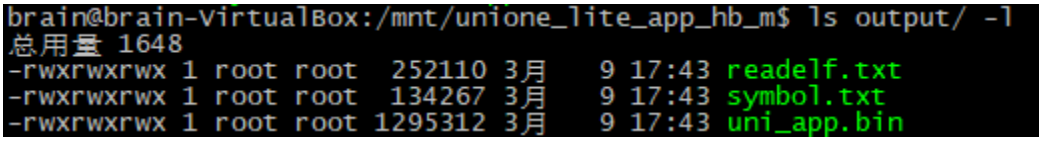
2 编译

(1) 执行编译

执行./build.sh，编译成功会看到如下信息：

```
Finished building target: uni_app.adx

Invoking: NM (symbol listing)
/opt/Andestech/BSPv422/toolchains/nds32le-elf-mculib-v3s/bin/nds32le-elf-nm -n -l -C "uni_app.adx" > ..//output/symbol.txt
Finished building: ..//output/symbol.txt

Invoking: Readelf (ELF info listing)
/opt/Andestech/BSPv422/toolchains/nds32le-elf-mculib-v3s/bin/nds32le-elf-readelf -a "uni_app.adx" > ..//output/readelf.txt
Finished building: ..//output/readelf.txt

Invoking: Objcopy (object content copy)
/opt/Andestech/BSPv422/toolchains/nds32le-elf-mculib-v3s/bin/nds32le-elf-objcopy -S -O binary "uni_app.adx" ..//output/uni_ap
p.bin
Finished building: ..//output/uni_app.bin

Invoking: Size (section size listing)
/opt/Andestech/BSPv422/toolchains/nds32le-elf-mculib-v3s/bin/nds32le-elf-size  "uni_app.adx" | tee ..//output/.PHONY.size
   text (code + rodata)    data     bss     dec     hex filename
1293350 (213774 + 1079576)        1936   34356 1329642  1449ea uni_app.adx
Finished building: ..//output/.PHONY.size

/mnt/unione_lite_app_hb_m
====== build done ==> output/ ======
```

(2) “output”目录下可以看到编译生成的完整固件，使用章节 3 中的下载工具即可进行烧录使用，目前 build.sh 脚本会同时编译生成 debug 和 release 两个版本，如需编译特定版本或者查看 symbol 信息，可以进入 build 目录，运行 make 指令手动编译，该方式生成 bin 文件及 symbol.txt 文件，存放在 output 目录中以供查看：

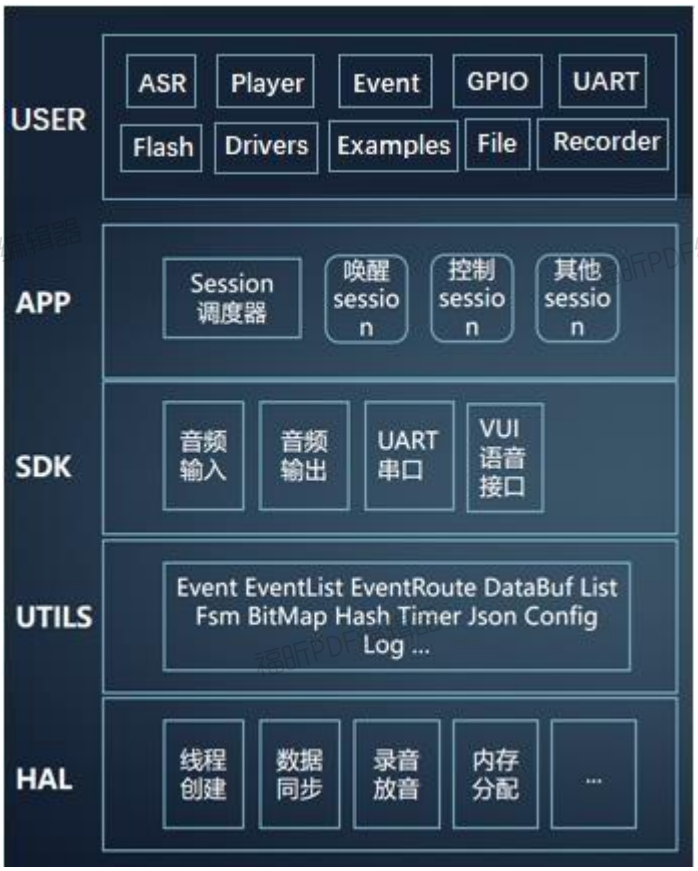

brain@brain-VirtualBox:/mnt/unione_lite_app_hb_m$ ls output/ -l
总用量 1648
-rwxrwxrwx 1 root root 252110 3月   9 17:43 readelf.txt
-rwxrwxrwx 1 root root 134267 3月   9 17:43 symbol.txt
-rwxrwxrwx 1 root root 1295312 3月   9 17:43 uni_app.bin

机芯HBM 离线方案开发指导手册 v1.0.1

6、软件架构

1.软件架构

机芯HBM芯片源码框架采用分层设计，各个层次完成特定的功能封装，最终为二次开发提供 user 层功能接口，基本框架结构如下图所示：



各个层次功能描述如下：

- ✓ **HAL 层**：提供包括线程调度、内存管理、信号量系统级统一接口，以及 Audio、Record、GPIO、Timer、I2C 等外设驱动统一接口；
- ✓ **UTILS 层**：提供常用的工具集代码，包括如：事件队列、Ring Buffer、List、状态机、Hash 运算、Json 解析等，供 SDK 及应用层使用；
- ✓ **SDK 层：**提供核心功能接口，如：Audio Play、MP3 解码、语音识别等；

- ✓ **APP 层：**语音识别基础业务功能实现，其核心由一个事件调度和状态机构成，将语音识别业务抽象成 wakeup、settings、music、watchdog 等不同的 session，每个 session 维护一个状态机，根据 SDK 层的识别结果事件驱动状态机运行，最终实现语音识别、播报业务逻辑；
- ✓ **USER 层:** 提供用户二次开发接口，包括语音识别、音频播放的控制接口，以及 GPIO、UART、Flash、I2C、Timer 等常用的外设驱动接口，还提供了 UCP 通用串口协议。

2.源码目录结构

源码各个文件功能如下所示：

```
├── build----------------------------------------> Makefile 系统
├── build.sh ------------------------------------> 编译脚本
├── ci.yml --------------------------------------> 自动化平台构建脚本，对用户无用
├── include -------------------------------------> 语音识别引擎及其他自动化生成外部头文件，不可修改
├── lib -----------------------------------------> 语音识别引擎及其他底层驱动库
├── middleware----------------------------------> RTOS 系统
├── nds32-ae210p.ld -----------------------------> 链接信息脚本，不可修改
├── nds32-ae210p.sag ----------------------------> 内存段分布配置，不可修改
├── readme.txt ----------------------------------> 发布版本信息
├── src -----------------------------------------> 架构代码文件夹
│   ├── app-------------------------------------> APP 层代码文件夹
│   │   ├── inc
│   │   └── src
│   │       ├── main.c -------------------------> 系统启动主程序，main 函数入口
│   │       ├── sessions -----------------------> sessions 代码文件夹
│   │       │   ├── uni_setting_session.c------ > setting 类事件处理 session
│   │       │   ├── uni_wakeup_session.c--------> wakeup 类事件处理 session
│   │       │   └── uni_watchdog_session.c ----- > watchdog 事件处理 session
│   │       ├── uni_record_save.c --------------> 录音保存功能实现，机芯HBM 暂不支持
│   │       ├── uni_session.c ------------------> 创建释放 session 对象
│   │       ├── uni_session_manage.c -----------> 管理 session 注册
│   │       └── uni_user_meeting.c------------- > APP 层与 USER 层交互接口
│   ├── hal ------------------------------------ > HAL 层实现代码
│   ├── sdk-------------------------------------> SDK 层实现代码
│   │   ├── audio ------------------------------ > Audio 播放器
│   │   ├── idle_detect ------------------------> 设备空闲计时管理
│   │   ├── player------------------------------ > MP3 解码器
│   │   └── vui --------------------------------> 语音识别功能
│   └── utils----------------------------------> UTILS 层实现代码
│       ├── arpt ------------------------------- > ARPT 自动化测试工具
│       ├── auto_string ------------------------> 变长字符串
```

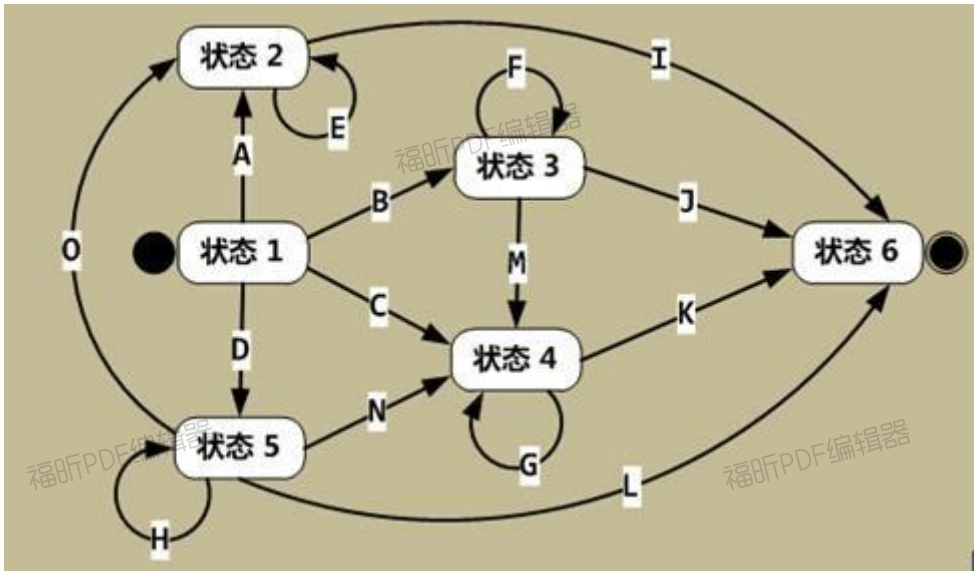
```
│       ├── bitmap ----------------------------> 二值状态变量集合
│       ├── black_board -----------------------> 系统状态管理
│       ├── cJSON -----------------------------> JSON 格式解析
│       ├── config ----------------------------> config.bin 文件内容解析
│       ├── crc16 -----------------------------> CRC16 算法
│       ├── data_buf --------------------------> 一个不用互斥锁管理的Ring Buffer
│       ├── event -----------------------------> 创建事件对象
│       ├── event_list ------------------------> 事件队列
│       ├── event_route -----------------------> 事件分发
│       ├── float2string ----------------------> 浮点转字符串，用于无 float 类型打印能力的printf
│       ├── fsm -------------------------------> 状态机
│       ├── hash ------------------------------> 一个简易 HASH 算法
│       ├── interruptable_sleep ---------------> 非阻塞的 sleep 方式
│       ├── list ------------------------------> 通用链表
│       ├── log -------------------------------> 带等级控制的 LOG 输出接口
│       ├── string ----------------------------> 一套简易的 string 操作接口
│       ├── timer -----------------------------> 基于 RTOS 系统的Timer
│       └── uart ------------------------------> 通用的 UART 接口
├── startup -----------------------------------> 芯片启动代码，不可修改
├── tools -------------------------------------> 自动化构建工具
│   └── scripts -------------------------------> 自动化构建脚本
│       ├── aik_debug.json --------------------> Debug 固件对应的 AIK 配置文件
│       ├── aik_release.json ------------------> Release 固件对应的 AIK 配置文件
│       ├── asrfix.dat ------------------------> 声学模型
│       ├── cmd_reply_data.json ---------------> UDP 平台用户定制命令词和回复语信息
│       ├── config_debug.bin ------------------> Debug 固件对应的应用配置文件
│       ├── config_release.bin ----------------> Release 固件对应的应用配置文件
│       ├── custom_config.json ----------------> UDP 平台用户定制系统配置信息
│       ├── default_tones ---------------------> 默认保底音频文件文件夹
│       ├── grammar.dat -----------------------> 语法模型
│       ├── grammar_jsgf.zip ------------------> 语法模型对应的构建脚本
│       ├── grammar.zip -----------------------> 语法模型文件压缩包
│       ├── input.txt -------------------------> 用户定制回复语列表
│       ├── pcm.bin ---------------------------> MP3 音频 flash 固件，自动生成的中间文件
│       ├── pcm_map.txt -----------------------> MP3 音频文件名及内容列表
│       ├── res_build_tool.py -----------------> 自动化构建脚本
│       ├── thresh.dat ------------------------> 唤醒词阈值推荐表
│       ├── tones -----------------------------> MP3 音频文件夹
│       └── wav_tones -------------------------> WAV 音频文件夹，自动转换到 tones
├── uni_ci.yml --------------------------------> 构建平台脚本，对用户无用
└── user --------------------------------------> USER 层实现代码
    ├── inc
```

```
│   ├── unione.h----------------------------> USER 层使用的底层头文件
│   ├── user_config.h-----------------------> USER 可配置项，包括串口、音量等
└── src
    ├── examples ----------------------------> 包含个别 USER 模块的示例代码
    ├── user_asr.c ------------------------> 语音识别控制接口
    ├── user_event.c ----------------------- > USER 事件分发机制，底层调用 USER 注册的事件回调函数
    ├── user_file.c ------------------------> SD 卡文件系统操作接口，机芯HBM 暂不支持
    ├── user_flash.c-----------------------> Flash 操作接口
    ├── user_gpio.c ------------------------ > GPIO 操作接口
    ├── user_main.c ------------------------> 用户代码入口，参考示例实现user_main()接口以增加业务逻辑
    ├── user_player.c ----------------------> 音频播放控制接口
    ├── user_power.c -----------------------> 功耗操作接口，机芯HBM 暂不支持
    ├── user_pwm.c -------------------------> PWM 操作接口
    ├── user_record.c ----------------------> 录音控制接口，机芯HBM 暂不支持
    ├── user_timer.c -----------------------> Timer 操作接口
    ├── user_uart.c------------------------ > UART 操作接口
    └── user_uni_ucp.c ---------------------> 通用串口协议操作接口
```

3.APP 层 session 状态机

为保证代码业务逻辑的额可扩展性和健壮性，在 APP 层设计中使用了事件分发和状态机的机制，每个session 都维护一个独立的状态机，底层事件分发后，根据各个session 状态机的状态激发对应session 的事件处理，如此一直运行下去。

代码中使用的是有限状态机,也称为 FSM(Finite State Machine)，其在任意时刻都处于有限状态集合中的某一状态。当其获得一个输入事件时，将从当前状态转换到另一个状态，或者仍然保持在当前状态。任何一个 FSM 都可以用状态转换图来描述：



在二次开发时，并不需要关心 APP 层的实现，USER 层已经将各个功能模块控制接口封装，调用 USER 层接口会自动触发状态机运转，从而实现定制功能。

机芯HBM 离线方案开发指导手册 v1.0.1

7、二次开发

1. 运行一个 Demo 示例

为方便用户二次开发，源码中提供了大量的示例 Demo，以供参考，相关源码可在“user\src\examples”路径下找到。

如果想要运行一个已有 Demo，可通过配置“user\inc\user_config.h”内的宏定义来选择 Demo 功能，如运行 ASR 控制相关示例，需修改两个宏定义如下：

```
#define USER_RUN_DEMO        1                                    // 1: 打开 Demo 功能
#define USER_RUN_DEMO_SELECT   USER_DEMO_ASR_CONTROL      // 选择 ASR 控制 Demo
```

然后，运行./build.sh 脚本，编译生成固件，烧录后重启开发板即可测试，默认唤醒词为“你好魔方”。

2. user_main()入口函数

user_main()接口是 USER 层二次开发入口函数，相当于系统中的 main 函数，定制化代码需加入到 user_main()函数中以编译运行，该函数可在“user\src\user_main.c”中找到。

从架构上看，该函数会在系初始化完毕后自动被调用，调用完毕后开始播放开机启动欢迎语，并开始进入循环等待实现 watch dog 喂狗操作，因此，作为一个限制，user_main()函数内代码不能是阻塞的，建议该函数中仅仅做系统初始化和事件注册操作，具体实现，可以参考 Demo 程序：user\src\examples\ hb_smart_ac.c。

3. 第一个定制程序

以下代码，作为一个简单示例，实现了用户定制的语音识别指令的打印：

```
static void _custom_setting_cb(USER_EVENT_TYPE event,
                               user_event_context_t *context) {
```

```
  event_custom_setting_t *setting = NULL;
  if (context) {
    setting = &context->custom_setting;
    printf("Action:      %s\n", setting->cmd);          // 对应 UDP 平台定制基本 action
    printf("ASR word:    %s\n", setting->word_str);     // 识别词
    printf("Reply list: %s\n", setting->reply_files);       // UDP 平台定制回复语编号列表
[1, 2]
    user_player_reply_list_random(setting->reply_files);// 随机播放一条回复语
  }
}

static void _register_event_callback(void)
  { user_event_subscribe_event(USER_CUSTOM_SETTING,
  _custom_setting_cb);
}

int user_main(void)
  { printf("Hello
  World !\n");
  _register_event_callback();   // 注册事件回调
  return 0;
}
```

运行后说“你好魔方”唤醒设备，然后说“打开风扇”打印结果为：

```
                Hello World !
                Action:      ac_power_on
          ASR word:    打开风扇
          Reply list:     [104, 105, 106]
```

然后设备播报回复语可能是一下三条其中一条（根据当前设备状态）：

“已为您打开风扇，当前为自然风”

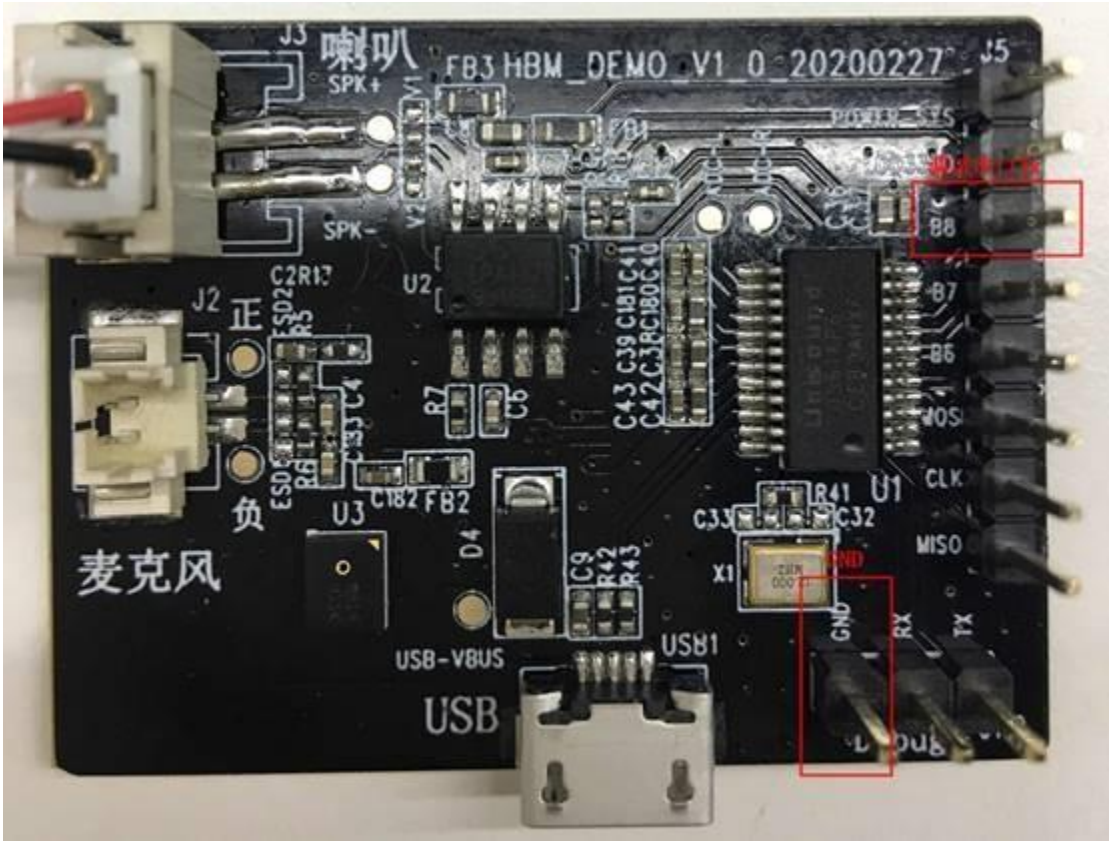
“已为您打开风扇，当前为正常风”

“已为您打开风扇，当前为睡眠风”

4. 串口 LOG 调试

由于芯片本身只有 1 个硬件 UART，默认定义用于外设通信，因此增加 Software UART 功能，设置 B8 引脚（也可修改代码设置为其他引脚）为 log 输出 Tx 端口，在开发板上丝印为“B8”，连接 PC 后使用常见串口工具（如 putty、xshell、secureCRT 等），配置波特率 (57600bps)、数据位 (8)、停止位 (1)、校验 (None)即可。

5.



TX 连接串口调试器 RX，GND 连接 GND(开发板上只有 1 个 GND 接口，为方便烧录和调试，可以将烧录器和串口调试器连接在同一台电脑或 USB Hub 上，以保持共地，则串口调试器只需连接 Tx（B8）引脚即可)，设备上电后，可在串口工具中观察到如下启动信息：

```
*|**||*|\*||**||**/'\    >>Unione Version : v0.0.1--2020030917
*\_//**|*\||**||**\_/    >>Build Data     : 2020030918

[T] 0<main>_system_start_process:137->System start woking !
```

注：在连接 PC 进行调试时，如需掉电重启，请将端口串口调试器与主控板连接后掉电重启，否则可能由于串口调试器漏掉输入使开发板进入低电压模式无法掉电重启。

机芯HBM 离线方案开发指导手册 v1.0.1

模块

这里列出了所有模块:

[详情级别 123]

▼配置宏定义	配置宏定义参考 user/inc/user_config.h
GPIO 相关宏定义	GPIO settings
audio 相关宏定义	Audio player volume settings, default value means (5, 25, 50, 75, 100)
通讯相关宏定义	Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
串口相关宏定义	Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if USE_UNIONE_PROTOCOL is 1, only one UART on HB-M board
语音相关宏定义	Recognize configurations, default multiply dialogue and no AEC
Demo 相关宏定义	使用方法见详细描述
▼音频控制	音频控制接口参考 user/inc/user_player.h
音频控制说明	
音频控制接口	
▼音频控制示例	
hb_player.c	
▼语音控制	语音控制接口参考 user/inc/user_asr.h
语音控制说明	
语音控制接口	
▼音频控制示例	
hb_asr_control.c	
▼标准协议数据通讯	标准协议数据通讯接口参考 user/inc/user_uni_ucp.h
标准协议数据通讯	
标准协议数据通讯接口	

▼标准协议数据通讯示例

hb_uart_ucp.c

▼数据存储 数据存储接口参考 user/inc/user_flash.h

数据存储说明

数据存储接口

▼数据存储示例

hb_flash_example.c

▼串口收发 串口收发接口参考 user/inc/user_uart.h

串口收发说明

串口收发接口

▼串口收发示例

hb_uart.c

▼定时器 定时器接口参考

定时器说明

定时器接口

定时器类型定义

▼定时器示例

hb_sw_timers.c

user_timer_buzzer.c

▼GPIO GPIO 接口参考 user/inc/user_gpio.h

GPIO 说明

GPIO 相关接口

GPIO 类型定义

▼GPIO 示例

hb_gpio_adc.c

hb_gpio_key.c

▼Event 事件 Event 事件接口参考

Event 事件说明

Event 事件接口

Event 事件类型定义

▼PWM PWM 接口参考

PWM 说明

PWM 相关接口

PWM 类型定义

▼PWM 示例

hb_pwm_led.c

▼方案示例 示例代码位于目录 user/src/examples/下

智能电风扇

智能管家

机芯HBM 离线方案开发指导手册 v1.0.1

配置宏定义

配置宏定义参考 user/inc/user_config.h 更多...

模块

GPIO 相关宏定义

GPIO settings

audio 相关宏定义

Audio player volume settings, default value means (5, 25, 50, 75, 100)

通讯相关宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0

串口相关宏定义

Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if **USE_UNIONE_PROTOCOL** is 1, only one UART on HB-M board

语音相关宏定义

Recognize configurations, default multiply dialogue and no AEC

Demo 相关宏定义

使用方法见详细描述

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关宏定义

配置宏定义

GPIO settings 更多...

宏定义

#define **GPIO_OUT_DEF_VAL** 1

0: default is low, 1: default is high, when GPIO set to output mode

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关宏定义

配置宏定义

GPIO settings 更多...

宏定义

#define **GPIO_OUT_DEF_VAL** 1

0: default is low, 1: default is high, when GPIO set to output mode

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

audio 相关宏定义

配置宏定义

Audio player volume settings, default value means (5, 25, 50, 75, 100) 更多...

宏定义

#define **USER_AUDIO_VOL_LEVEL** 5

Level is divided into several 0 - 100

#define **USER_AUDIO_VOL_MIN** 5

The minimum volume

#define **USER_AUDIO_VOL_MAX** 100

The maximum volume

#define **USER_AUDIO_VOL_MID** 50

The middle volume

#define **USER_AUDIO_VOL_DEF** 50

Default volume if reset device

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

audio 相关宏定义

配置宏定义

Audio player volume settings, default value means (5, 25, 50, 75, 100) 更多...

宏定义

#define **USER_AUDIO_VOL_LEVEL** 5

Level is divided into several 0 - 100

#define **USER_AUDIO_VOL_MIN** 5

The minimum volume

#define **USER_AUDIO_VOL_MAX** 100

The maximum volume

#define **USER_AUDIO_VOL_MID** 50

The middle volume

#define **USER_AUDIO_VOL_DEF** 50

Default volume if reset device

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

audio 相关宏定义

配置宏定义

Audio player volume settings, default value means (5, 25, 50, 75, 100) 更多...

宏定义

#define **USER_AUDIO_VOL_LEVEL** 5

Level is divided into several 0 - 100

#define **USER_AUDIO_VOL_MIN** 5

The minimum volume

#define **USER_AUDIO_VOL_MAX** 100

The maximum volume

#define **USER_AUDIO_VOL_MID** 50

The middle volume

#define **USER_AUDIO_VOL_DEF** 50

Default volume if reset device

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

audio 相关宏定义

配置宏定义

Audio player volume settings, default value means (5, 25, 50, 75, 100) 更多...

宏定义

#define **USER_AUDIO_VOL_LEVEL** 5

Level is divided into several 0 - 100

#define **USER_AUDIO_VOL_MIN** 5

The minimum volume

#define **USER_AUDIO_VOL_MAX** 100

The maximum volume

#define **USER_AUDIO_VOL_MID** 50

The middle volume

#define **USER_AUDIO_VOL_DEF** 50

Default volume if reset device

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

audio 相关宏定义

配置宏定义

Audio player volume settings, default value means (5, 25, 50, 75, 100) 更多...

宏定义

#define **USER_AUDIO_VOL_LEVEL** 5

Level is divided into several 0 - 100

#define **USER_AUDIO_VOL_MIN** 5

The minimum volume

#define **USER_AUDIO_VOL_MAX** 100

The maximum volume

#define **USER_AUDIO_VOL_MID** 50

The middle volume

#define **USER_AUDIO_VOL_DEF** 50

Default volume if reset device

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

audio 相关宏定义

配置宏定义

Audio player volume settings, default value means (5, 25, 50, 75, 100) 更多...

宏定义

#define **USER_AUDIO_VOL_LEVEL** 5

Level is divided into several 0 - 100

#define **USER_AUDIO_VOL_MIN** 5

The minimum volume

#define **USER_AUDIO_VOL_MAX** 100

The maximum volume

#define **USER_AUDIO_VOL_MID** 50

The middle volume

#define **USER_AUDIO_VOL_DEF** 50

Default volume if reset device

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define USE_UNIONE_PROTOCOL 0

0: don't use, 1: use

#define OPEN_ASR_EVENT_UCP 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define UNI_UART_DEVICE_NUM 1

Only 1 can be used on HB-M board

#define UNI_UART_PIN_SELECT 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define UNI_UART_BAUD_RATE 115200

4800/9600/19200/38400/57600/115200

#define UNI_UART_PARITY 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define **USE_UNIONE_PROTOCOL** 0

0: don't use, 1: use

#define **OPEN_ASR_EVENT_UCP** 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define **UNI_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **UNI_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **UNI_UART_BAUD_RATE** 115200

4800/9600/19200/38400/57600/115200

#define **UNI_UART_PARITY** 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define **USE_UNIONE_PROTOCOL** 0

0: don't use, 1: use

#define **OPEN_ASR_EVENT_UCP** 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define **UNI_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **UNI_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **UNI_UART_BAUD_RATE** 115200

4800/9600/19200/38400/57600/115200

#define **UNI_UART_PARITY** 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define **USE_UNIONE_PROTOCOL** 0

0: don't use, 1: use

#define **OPEN_ASR_EVENT_UCP** 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define **UNI_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **UNI_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **UNI_UART_BAUD_RATE** 115200

4800/9600/19200/38400/57600/115200

#define **UNI_UART_PARITY** 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define **USE_UNIONE_PROTOCOL** 0

0: don't use, 1: use

#define **OPEN_ASR_EVENT_UCP** 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define **UNI_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **UNI_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **UNI_UART_BAUD_RATE** 115200

4800/9600/19200/38400/57600/115200

#define **UNI_UART_PARITY** 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define **USE_UNIONE_PROTOCOL** 0

0: don't use, 1: use

#define **OPEN_ASR_EVENT_UCP** 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define **UNI_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **UNI_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **UNI_UART_BAUD_RATE** 115200

4800/9600/19200/38400/57600/115200

#define **UNI_UART_PARITY** 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define **USE_UNIONE_PROTOCOL** 0

0: don't use, 1: use

#define **OPEN_ASR_EVENT_UCP** 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define **UNI_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **UNI_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **UNI_UART_BAUD_RATE** 115200

4800/9600/19200/38400/57600/115200

#define **UNI_UART_PARITY** 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

通讯相关宏定义

配置宏定义

Use Unione communication protocol settings, default USE B6/B7 pin, 115200, 8, 1, 0
更多...

宏定义

#define **USE_UNIONE_PROTOCOL** 0

0: don't use, 1: use

#define **OPEN_ASR_EVENT_UCP** 0

0: not out, 1: output ASR result by UCP uart port, payload is action string, cmd is fixed to 10001

#define **UNI_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **UNI_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **UNI_UART_BAUD_RATE** 115200

4800/9600/19200/38400/57600/115200

#define **UNI_UART_PARITY** 0

	0: None, 1: Odd, 2: Even

#define	**UNI_UART_STOP_BIT** 1 1/2

#define	**UNI_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

串口相关宏定义

配置宏定义

Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if USE_UNIONE_PROTOCOL is 1, only one UART on HB-M board 更多...

宏定义

#define USER_UART_DEVICE_NUM 1

Only 1 can be used on HB-M board

#define USER_UART_PIN_SELECT 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define USER_UART_BAUD_RATE 19200

4800/9600/19200/38400/57600/115200

#define USER_UART_PARITY 0

0: None, 1: Odd, 2: Even

#define USER_UART_STOP_BIT 1

1/2

#define **USER_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

串口相关宏定义

配置宏定义

Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if **USE_UNIONE_PROTOCOL** is 1, only one UART on HB-M board 更多...

宏定义

#define **USER_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **USER_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **USER_UART_BAUD_RATE** 19200

4800/9600/19200/38400/57600/115200

#define **USER_UART_PARITY** 0

0: None, 1: Odd, 2: Even

#define **USER_UART_STOP_BIT** 1

1/2

#define **USER_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

串口相关宏定义

配置宏定义

Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if **USE_UNIONE_PROTOCOL** is 1, only one UART on HB-M board 更多...

宏定义

#define **USER_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **USER_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **USER_UART_BAUD_RATE** 19200

4800/9600/19200/38400/57600/115200

#define **USER_UART_PARITY** 0

0: None, 1: Odd, 2: Even

#define **USER_UART_STOP_BIT** 1

1/2

#define **USER_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

串口相关宏定义

配置宏定义

Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if **USE_UNIONE_PROTOCOL** is 1, only one UART on HB-M board 更多...

宏定义

#define **USER_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **USER_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **USER_UART_BAUD_RATE** 19200

4800/9600/19200/38400/57600/115200

#define **USER_UART_PARITY** 0

0: None, 1: Odd, 2: Even

#define **USER_UART_STOP_BIT** 1

1/2

#define **USER_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

串口相关宏定义

配置宏定义

Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if **USE_UNIONE_PROTOCOL** is 1, only one UART on HB-M board 更多...

宏定义

#define **USER_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **USER_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **USER_UART_BAUD_RATE** 19200

4800/9600/19200/38400/57600/115200

#define **USER_UART_PARITY** 0

0: None, 1: Odd, 2: Even

#define **USER_UART_STOP_BIT** 1

1/2

#define **USER_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

串口相关宏定义

配置宏定义

Custom UART settings, default USE B6/B7 pin, 19200, 8, 1, 0 Cannot use it if **USE_UNIONE_PROTOCOL** is 1, only one UART on HB-M board 更多...

宏定义

#define **USER_UART_DEVICE_NUM** 1

Only 1 can be used on HB-M board

#define **USER_UART_PIN_SELECT** 0

0: B7/B6, 1: B0/B1, 2: B2/B3

#define **USER_UART_BAUD_RATE** 19200

4800/9600/19200/38400/57600/115200

#define **USER_UART_PARITY** 0

0: None, 1: Odd, 2: Even

#define **USER_UART_STOP_BIT** 1

1/2

#define **USER_UART_DATA_BIT** 8

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

语音相关宏定义

配置宏定义

Recognize configurations, default multiply dialogue and no AEC 更多...

宏定义

```
#define MULTI_DIALOGUE_MODE 1
```

```
#define WAKEUP_INTERRUPT_MODE 0
```

详细描述

宏定义说明

◆ MULTI_DIALOGUE_MODE

```
#define MULTI_DIALOGUE_MODE 1
```

多轮对话模式

- 1: multiply dialogue mode
- 0: single dialogue mode

◆ WAKEUP_INTERRUPT_MODE

```
#define WAKEUP_INTERRUPT_MODE  0
```

唤醒打断模式

- 1: wakeup interrupt reply
- 0: wakeup cannot interrupt reply

机芯HBM 离线方案开发指导手册 v1.0.1

语音相关宏定义

配置宏定义

Recognize configurations, default multiply dialogue and no AEC 更多...

宏定义

```
#define  MULTI_DIALOGUE_MODE  1
```

```
#define  WAKEUP_INTERRUPT_MODE  0
```

详细描述

宏定义说明

◆ MULTI_DIALOGUE_MODE

```
#define MULTI_DIALOGUE_MODE  1
```

多轮对话模式

- 1: multiply dialogue mode
- 0: single dialogue mode

◆ WAKEUP_INTERRUPT_MODE

```
#define WAKEUP_INTERRUPT_MODE  0
```

唤醒打断模式

- 1: wakeup interrupt reply
- 0: wakeup cannot interrupt reply

机芯HBM 离线方案开发指导手册 v1.0.1

语音相关宏定义

配置宏定义

Recognize configurations, default multiply dialogue and no AEC 更多...

宏定义

```
#define MULTI_DIALOGUE_MODE 1
```

```
#define WAKEUP_INTERRUPT_MODE 0
```

详细描述

宏定义说明

◆ MULTI_DIALOGUE_MODE

```
#define MULTI_DIALOGUE_MODE 1
```

多轮对话模式

- 1: multiply dialogue mode
- 0: single dialogue mode

◆ WAKEUP_INTERRUPT_MODE

```
#define WAKEUP_INTERRUPT_MODE  0
```

唤醒打断模式

- 1: wakeup interrupt reply
- 0: wakeup cannot interrupt reply

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 **USER_RUN_DEMO** 设置为 1
2. 再通过配置 **USER_RUN_DEMO_SELECT** 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 **USER_RUN_DEMO** 设置为 1
2. 再通过配置 **USER_RUN_DEMO_SELECT** 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

Demo 相关宏定义

配置宏定义

使用方法见详细描述 更多...

宏定义

#define **USER_RUN_DEMO** 0

add your code to user_main() to run 更多...

#define **USER_DEMO_SMART_AC** 1

a demo to show smart air fan (have to replace grammar and mp3 to support commands)

#define **USER_DEMO_HOUSEKEEPER** 2

a demo to show IR controller base on HB1688 what is driven by UART

#define **USER_DEMO_KEY_WAKEUP** 3

a demo to show how to wakeup device by digital kay base on GPIO

#define **USER_DEMO_ADC_CTRL** 4

a demo to show how to mute speaker if at night base on photosensitive sensor

#define **USER_DEMO_TIMERS** 5

a demo to show how to set 15 software timers base on 1 hardware timer

#define **USER_DEMO_BUZZER** 6

a demo to show how to drive a buzzer base on timer

#define **USER_DEMO_PWM_LED** 7

a demo to show how to drive a LED base on PWM

#define **USER_DEMO_I2C_TIME** 8

a demo to show how to get real time from DS3231 base on I2C

#define **USER_DEMO_SPI_LCD** 9

a demo to show how to draw chars on LCD waht driven by SSD1306 base on SPI

#define **USER_DEMO_UART** 10

a demo to show how to use uart send and receive data

#define **USER_DEMO_UART_UCP** 11

a demo to show how to use uart send and receive data with uart communication protocol

#define **USER_DEMO_PLAYER** 12

a demo to show how to use player

#define **USER_DEMO_FLASH** 13

a demo to show how to save value into flash

#define **USER_DEMO_ASR_CONTROL** 14

a demo to show how to control asr state

#define USER_RUN_DEMO_SELECT USER_DEMO_SMART_AC

select a demo to run

详细描述

Demo 使用方法：

1. 要使能 Demo 功能，先将 USER_RUN_DEMO 设置为 1
2. 再通过配置 USER_RUN_DEMO_SELECT 定义为需要展示的示例程序宏
3. 重新编译固件，烧录后即可体验 demo 运行效果

宏定义说明

◆ USER_RUN_DEMO

```
#define USER_RUN_DEMO  0
```

1: select a demo to run, 0: no user demo,

机芯HBM 离线方案开发指导手册 v1.0.1

音频控制

音频控制接口参考user/inc/user_player.h 更多...

模块

音频控制说明

音频控制接口

音频控制示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

音频控制说明

音频控制

音频控制说明：

1. 可播放 tools/scripts 路径下的音频，文件名以序号区分
2. 调用播放接口是异步不堵塞的
3. **AUDIO_PLAY_TYPE** 目前只支持AUDIO_PLAY_REPLY

机芯HBM 离线方案开发指导手册 v1.0.1

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

函数

音频控制接口

音频控制

函数

int	**user_player_play** (**AUDIO_PLAY_TYPE** type, const char *file)
	播放文件 更多...
int	**user_player_reply_list_num** (const char *file_list, int num)
	播放列表序号音频 更多...
int	**user_player_reply_list_random** (const char *file_list)
	随机播放列表音频 更多...
int	**user_player_stop** (**AUDIO_PLAY_TYPE** type)
	停止播放 更多...
int	**user_player_set_volume_min** (void)
	设置最小音量
int	**user_player_set_volume_max** (void)
	设置最大音量
int	**user_player_set_volume_mid** (void)

	设置中等音量
int	user_player_set_volume_up (void)
	增加音量
int	user_player_set_volume_down (void)
	减小音量
int	user_player_speaker_mute (void)
	设置喇叭静音
int	user_player_speaker_unmute (void)
	取消设置喇叭静音

详细描述

函数说明

◆ user_player_play()

```
int user_player_play ( AUDIO_PLAY_TYPE type,
                       const char *      file
                     )
```

参数

type 播放类型
file 播放文件名

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_num()

```
int user_player_reply_list_num ( const char * file_list,
                                 int          num
                               )
```

参数

file_list 文件列表 [1, 2, 3]
num 列表序号

返回值

0 操作成功
-1 操作失败

◆ user_player_reply_list_random()

```
int user_player_reply_list_random ( const char * file_list )
```

参数

file_list 文件列表 [1, 2, 3]

返回值

0 操作成功
-1 操作失败

◆ user_player_stop()

```
int user_player_stop ( AUDIO_PLAY_TYPE type )
```

参数

type 播放类型

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

音频控制示例

音频控制

模块

hb_player.c

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_player.c

音频控制 » 音频控制示例

可通过定义 user/inc/user_config.h 中的宏 **USER_RUN_DEMO_SELECT** 为 **USER_DEMO_PLAYER** 运行示例程序

```
/**************************************************************************
        ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
**************************************************************************
        **
*
* Description : hb_player.c
* Author : liuwenzheng@unisound.com
* Date : 2020.04.25
*
**************************************************************************
        **/
#include "user_player.h"
#define TAG "player_demo"
static void _player_test_process(void *args) {
while (1) {
```

```
uni_sleep(10);
user_player_set_volume_min();
LOGT(TAG, "set min vol");
/* 播放9.mp3 文件*/
user_player_play(AUDIO_PLAY_REPLY, "9"); //9.mp3
uni_sleep(10);
user_player_set_volume_mid();
LOGT(TAG, "set mid vol");
/* 播放列表中的101.MP3 文件*/
user_player_reply_list_num("[9,101,102]", 1);
uni_sleep(10);
user_player_set_volume_max();
LOGT(TAG, "set max vol");
/* 随机播放列表中的某个文件*/
user_player_reply_list_random("[9,101,102]");
break;
}
}
static Result _create_player_test_thread(void) {
thread_param param;
uni_pthread_t  pid;
uni_memset(&param, 0, sizeof(param));
param.stack_size = STACK_SMALL_SIZE;
param.priority = OS_PRIORITY_LOW;
uni_strncpy(param.task_name, "player_test", sizeof(param.task_name) - 1);
if (0 != uni_pthread_create(&pid, &param,
_player_test_process, NULL)) {
LOGE(TAG, "create thread failed");
return E_FAILED;
}
uni_pthread_detach(pid);
return E_OK;
}
int hb_player(void) {
_create_player_test_thread();
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

语音控制

语音控制接口参考user/inc/user_asr.h 更多...

模块

- 语音控制说明
- 语音控制接口
- 音频控制示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

语音控制说明

语音控制

语音控制说明:

1. 可控制语音识别工作模式的切换（仅识别唤醒词； 仅识别命令词）
2. 可控制语音识别功能的开关
3. 可控制禁用/启用特定的识别词

机芯HBM 离线方案开发指导手册 v1.0.1

语音控制接口

语音控制

函数

int	**user_asr_goto_sleep** (void) 进入睡眠状态 更多...
int	**user_asr_goto_awakend** (void) 进入唤醒状态 更多...
int	**user_asr_recognize_enable** (void) 使能语音识别 更多...
int	**user_asr_recognize_disable** (void) 关闭语音识别 更多...
int	**user_asr_word_enable** (const char *word) 启用识别词 更多...
int	**user_asr_word_disable** (const char *word) 禁用识别词 更多...

详细描述

函数说明

◆ user_asr_goto_awakend()

int user_asr_goto_awakend (void)

进入后将只会识别命令词

返回值

0 操作成功

◆ user_asr_goto_sleep()

int user_asr_goto_sleep (void)

进入后将只会识别唤醒词

返回值

0 操作成功

◆ user_asr_recognize_disable()

int user_asr_recognize_disable (void)

mic 被关闭，停止拾音

返回值

0 操作成功

◆ user_asr_recognize_enable()

int user_asr_recognize_enable (void)

mic 开始接收数据

返回值

0 操作成功

◆ user_asr_word_disable()

int user_asr_word_disable (const char * word)

禁用识别词

参数

word 要禁用的识别词

返回值

0 操作成功

◆ user_asr_word_enable()

int user_asr_word_enable (const char * word)

启用识别词

参数

word 要启用的识别词

返回值

0 操作成功

机芯HBM 离线方案开发指导手册 v1.0.1

语音控制接口

语音控制

函数

int	**user_asr_goto_sleep** (void)
	进入睡眠状态 更多...
int	**user_asr_goto_awakend** (void)
	进入唤醒状态 更多...
int	**user_asr_recognize_enable** (void)
	使能语音识别 更多...
int	**user_asr_recognize_disable** (void)
	关闭语音识别 更多...
int	**user_asr_word_enable** (const char *word)
	启用识别词 更多...
int	**user_asr_word_disable** (const char *word)
	禁用识别词 更多...

详细描述

函数说明

◆ user_asr_goto_awakend()

int user_asr_goto_awakend (void)

进入后将只会识别命令词

返回值

0 操作成功

◆ user_asr_goto_sleep()

int user_asr_goto_sleep (void)

进入后将只会识别唤醒词

返回值

0 操作成功

◆ user_asr_recognize_disable()

int user_asr_recognize_disable (void)

mic 被关闭，停止拾音

返回值

0 操作成功

◆ user_asr_recognize_enable()

int user_asr_recognize_enable (void)

mic 开始接收数据

返回值

0 操作成功

◆ user_asr_word_disable()

int user_asr_word_disable (const char * word)

禁用识别词

参数

word 要禁用的识别词

返回值

0 操作成功

◆ user_asr_word_enable()

int user_asr_word_enable (const char * word)

启用识别词

参数

word 要启用的识别词

返回值

0 操作成功

机芯HBM 离线方案开发指导手册 v1.0.1

语音控制接口

语音控制

函数

int	**user_asr_goto_sleep** (void)
	进入睡眠状态 更多...
int	**user_asr_goto_awakend** (void)
	进入唤醒状态 更多...
int	**user_asr_recognize_enable** (void)
	使能语音识别 更多...
int	**user_asr_recognize_disable** (void)
	关闭语音识别 更多...
int	**user_asr_word_enable** (const char *word)
	启用识别词 更多...
int	**user_asr_word_disable** (const char *word)
	禁用识别词 更多...

详细描述

函数说明

◆ user_asr_goto_awakend()

int user_asr_goto_awakend (void)

进入后将只会识别命令词

返回值

0 操作成功

◆ user_asr_goto_sleep()

int user_asr_goto_sleep (void)

进入后将只会识别唤醒词

返回值

0 操作成功

◆ user_asr_recognize_disable()

int user_asr_recognize_disable (void)

mic 被关闭，停止拾音

返回值

0 操作成功

◆ user_asr_recognize_enable()

int user_asr_recognize_enable (void)

mic 开始接收数据

返回值

0 操作成功

◆ user_asr_word_disable()

int user_asr_word_disable (const char * word)

禁用识别词

参数

word 要禁用的识别词

返回值

0 操作成功

◆ user_asr_word_enable()

int user_asr_word_enable (const char * word)

启用识别词

参数

word 要启用的识别词

返回值

0 操作成功

机芯HBM 离线方案开发指导手册 v1.0.1

语音控制接口

语音控制

函数

int	**user_asr_goto_sleep** (void)
	进入睡眠状态 更多...
int	**user_asr_goto_awakend** (void)
	进入唤醒状态 更多...
int	**user_asr_recognize_enable** (void)
	使能语音识别 更多...
int	**user_asr_recognize_disable** (void)
	关闭语音识别 更多...
int	**user_asr_word_enable** (const char *word)
	启用识别词 更多...
int	**user_asr_word_disable** (const char *word)
	禁用识别词 更多...

详细描述

函数说明

◆ user_asr_goto_awakend()

int user_asr_goto_awakend (void)

进入后将只会识别命令词

返回值

0 操作成功

◆ user_asr_goto_sleep()

int user_asr_goto_sleep (void)

进入后将只会识别唤醒词

返回值

0 操作成功

◆ user_asr_recognize_disable()

int user_asr_recognize_disable (void)

mic 被关闭，停止拾音

返回值

0 操作成功

◆ user_asr_recognize_enable()

int user_asr_recognize_enable (void)

mic 开始接收数据

返回值

0 操作成功

◆ user_asr_word_disable()

int user_asr_word_disable (const char * word)

禁用识别词

参数

word 要禁用的识别词

返回值

0 操作成功

◆ user_asr_word_enable()

int user_asr_word_enable (const char * word)

启用识别词

参数

word 要启用的识别词

返回值

0 操作成功

机芯HBM 离线方案开发指导手册 v1.0.1

函数

语音控制接口

语音控制

详细描述

函数说明

◆ user_asr_goto_awakend()

int user_asr_goto_awakend (void)

进入后将只会识别命令词

返回值

0 操作成功

◆ user_asr_goto_sleep()

int user_asr_goto_sleep (void)

进入后将只会识别唤醒词

返回值

0 操作成功

◆ user_asr_recognize_disable()

int user_asr_recognize_disable (void)

mic 被关闭，停止拾音

返回值

0 操作成功

◆ user_asr_recognize_enable()

int user_asr_recognize_enable (void)

mic 开始接收数据

返回值

0 操作成功

◆ user_asr_word_disable()

int user_asr_word_disable (const char * word)

禁用识别词

参数

word 要禁用的识别词

返回值

0 操作成功

◆ user_asr_word_enable()

int user_asr_word_enable (const char * word)

启用识别词

参数

word 要启用的识别词

返回值

0 操作成功

机芯HBM 离线方案开发指导手册 v1.0.1

函数

语音控制接口

语音控制

详细描述

函数说明

◆ user_asr_goto_awakend()

int user_asr_goto_awakend (void)

进入后将只会识别命令词

返回值

0 操作成功

◆ user_asr_goto_sleep()

int user_asr_goto_sleep (void)

进入后将只会识别唤醒词

返回值

0 操作成功

◆ user_asr_recognize_disable()

int user_asr_recognize_disable (void)

mic 被关闭，停止拾音

返回值

0 操作成功

◆ user_asr_recognize_enable()

int user_asr_recognize_enable (void)

mic 开始接收数据

返回值

0 操作成功

◆ user_asr_word_disable()

int user_asr_word_disable (const char * word)

禁用识别词

参数

word 要禁用的识别词

返回值

0 操作成功

◆ user_asr_word_enable()

int user_asr_word_enable (const char * word)

启用识别词

参数

word 要启用的识别词

返回值

0 操作成功

机芯HBM 离线方案开发指导手册 v1.0.1

函数

语音控制接口

语音控制

详细描述

函数说明

◆ user_asr_goto_awakend()

int user_asr_goto_awakend (void)

进入后将只会识别命令词

返回值

0 操作成功

◆ user_asr_goto_sleep()

int user_asr_goto_sleep (void)

进入后将只会识别唤醒词

返回值

0 操作成功

◆ user_asr_recognize_disable()

int user_asr_recognize_disable (void)

mic 被关闭，停止拾音

返回值

0 操作成功

◆ user_asr_recognize_enable()

int user_asr_recognize_enable (void)

mic 开始接收数据

返回值

0 操作成功

◆ user_asr_word_disable()

int user_asr_word_disable (const char * word)

禁用识别词

参数

word 要禁用的识别词

返回值

0 操作成功

◆ user_asr_word_enable()

int user_asr_word_enable (const char * word)

启用识别词

参数

word 要启用的识别词

返回值

0 操作成功

机芯HBM 离线方案开发指导手册 v1.0.1

模块

音频控制示例

语音控制

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_asr_control.c

语音控制 » 音频控制示例

可通过定义 user/inc/user_config.h 中的宏 **USER_RUN_DEMO_SELECT** 为 **USER_DEMO_ASR_CONTROL** 运行示例程序

```
/*****************************************************************************
        ***
* Copyright (C) 2020-2020 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
        **
*
* Description : user_flash_example.c
* Author : yuanshifeng@unisound.com
* Date : 2020.04.25
*
*****************************************************************************
        **/
#include "user_asr.h"
#define TAG "asr_control"
static void _hb_asr_control_task(void *args) {
while (1) {
```

```
//进入睡眠模式，此时将只能通过唤醒词唤醒，说命令词无效
user_asr_goto_sleep();
LOGT(TAG, "Now is into sleep mode, you can awake it");
uni_sleep(10);//延迟 10 秒
//进入唤醒模式，此时可以识别命令词
user_asr_goto_awakend();
LOGT(TAG, "Now is into awake mode, you can say cmd word ");
uni_sleep(10);
//仍然在唤醒模式，执行关闭语音控制
user_asr_recognize_disable();
LOGT(TAG, "asr recognize is disabled, nothing will be recongnized");
uni_sleep(10);
//重新打开语音控制
user_asr_recognize_enable();
LOGT(TAG, "asr recognize is enabled");
//延迟 10 秒
uni_sleep(10);
}
}
int hb_asr_control(void) {
//启动一个线程，执行语音控制切换
uni_pthread_t  pid;
thread_param param;
param.stack_size = STACK_SMALL_SIZE;
param.priority = OS_PRIORITY_NORMAL;
strncpy(param.task_name,  "hb-asr", sizeof(param.task_name) - 1);
uni_pthread_create(&pid,  &param, _hb_asr_control_task, NULL);
//禁用一个识别/唤醒词
user_asr_word_disable("退下");
user_asr_word_disable("再见");
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

云知声标准协议数据通讯

云知声标准协议数据通讯接口参考 user/inc/user_uni_ucp.h 更多...

模块

云知声标准协议数据通讯

云知声标准协议数据通讯接口

云知声标准协议数据通讯示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

云知声标准协议数据通讯

云知声标准协议数据通讯

云知声标准协议数据通讯说明:

1. 可通过定义 user/inc/user_config.h 中的宏 **USE_UNIONE_PROTOCOL** 设为 1 来启用此功能

协议说明:

云知声蜂鸟 UART 可靠通信协议接口文档

V3.0

修改记录

版本	修订日期	修订人员	修订纪要
V2.0	2019.12.30	尚金龙	Release 2.0
V2.1	2020.01.06	尚金龙	协议栈自动处理 ack、resend、去除重复帧
V3.0	2020.04.27	尚金龙	支持跨平台移植

文档目录

1. 概述 4
1.1. 目的与受众 4
1.2. 简介 4
2. 接口定义 5
2.1. UART 底层 API. 5
2.1.1. UartInitialize 5
2.1.2. UartFinalize. 6
2.1.3. UartWrite. 6
2.2. UART protocol API. 7
2.2.1. CommProtocolInit 7
2.2.2. CommProtocolFinal 7
2.2.3. CommProtocolPacketAssembleAndSend. 7
2.2.4. CommProtocolReceiveUartData. 8

3. Protocol 帧结构 9
3.1.1. Protocol layout 9
3.1.2. Protocol struct 9
3.1.3. Protocol ACK. 9
3.1.4. Protocol 重要参数指标 10
4. 接口调用流程 10
4.1. 已有自定义协议调用过程 10
4.2. 蜂鸟 UART 协议调用过程 11
5. 移植协议栈到任意平台 13
5.1. 注册跨平台 Hooks 13
5.2. Porting demo 13
6. 附录 14
6.1. 错误码 14

1. 概述

1.1. 目的与受众

本文系芯片UART 应用层可靠通信串口协议，供内部开发人员和获授权客户开发使用，旨在为客户提供简单、通用的 UART 可靠通信协议规范。

1.2. 简介

目标客户：

一、已有自定义 UART 通信协议客户，对此类型客户，开放 UART 读写 API，客户可自行定制基于 UART 的各类传输特性。

二、尚未自定义 UART 通信协议客户，对此类客户，开放一套完整的UART 通信协议，提供 API，实现串口通信。

UART 协议特性：

一、类 TCP 可靠传输模式。二、

类 UDP 不可靠传输模式。三、

自动过滤重复帧。

四、自动重传。

五、自动组装数据帧，自动拆解数据帧。

六、最大帧长检测设置。

七、支持控制命令类型定制，命令参数定制。

八、支持极低内存限制下，自适应内存垃圾回收功能。

九、crc16 校验和。

2. 接口定义

2.1. UART 底层 API

用于uart 底层驱动初始化，数据发送、接收

2.1.1. UartInitialize

```
typedef   enum
   {  UNI_UART
   1,
   UNI_UART2,
   UNI_UART3,
} UartDeviceName;
typedef enum {
   UNI_B_1200,
   UNI_B_2400,
   UNI_B_4800,
   UNI_B_9600,
   UNI_B_14400,
   UNI_B_19200,
   UNI_B_38400,
   UNI_B_57600,
   UNI_B_115200,
} UartSpeed;
typedef enum {
   UNI_PARITY_ODD,
   UNI_PARITY_EVEN,
   UNI_PARITY_NONE,
   UNI_PARITY_MARK,
   UNI_PARITY_SPACE,
} UartParity;
typedef enum {
   UNI_ONE_STOP_BIT,
   UNI_ONE_5_STOP_BIT,
```

```
} UartStop;


/**
 * uart init configure parameter
 */
typedef struct {
    UartDeviceName device; /* device name */
    UartSpeed speed; /* baudrate */
    UartParity parity; /* parity check */
    UartStop stop; /* stop bit */
    int data_bit; /* data bit */
} UartConfig;
typedef void (*RecvUartDataHandler)(char *buf, int len);
/**
 * @brief uart init
 * @param config uart configure parameter
 * @param handler handle uart receive data hook
 * @return 0 means success, -1 means failed
 */
int UartInitialize(UartConfig *config, RecvUartDataHandler handler);
```

2.1.2. UartFinalize

```
/**
 * @brief uart finalize
 * @param void
 * @return void
 */
void UartFinalize(void);
```

2.1.3. UartWrite

```
/**
 * @brief write data by UART, multi-thread unsafe, please write in sync mode
 * @param buf the data buffer to write
 * @param len the data length
 * @return the actual write length by UART
 */
```

```
int UartWrite(char *buf, int len);
```

2.2. UART protocol API

用于串口数据封装、解包

2.2.1. CommProtocolInit

```
typedef struct {
    CommCmd cmd; /* command, such as power_on, power_off */
    CommPayloadLen payload_len; /* parameter length of command */
    char payload[0]; /* parameter of command */
} PACKED CommPacket;
typedef void (*CommRecvPacketHandler)(CommPacket *packet);


/**
 * @brief communication protocol init
 * @param write_handler the write handler, such as UartWrite int uni_uart.h
 * @param recv_handler when uart data disassemble as communication protocol frame, the frame will be
translate to struct CommPacket, then the CommPacket will callback to user
 * @return 0 means success, -1 means failed
 */
int CommProtocolInit(CommWriteHandler write_handler,
                                    CommRecvPacketHandler recv_handler);
```

2.2.2. CommProtocolFinal

```
/**
 * @brief communication protocol finalize
 * @param void
 * @return void
 */
void CommProtocolFinal(void);
```

2.2.3. CommProtocolPacketAssembleAndSend

```
/**
* @brief send one packet(communication protocol frame format)
* @param type customer type, 0 means Unisound
* @param cmd command type, should define as enum (such as power_on、power_off)
* @param payload the payload of cmd, can set as NULL
* @param payload_len the payload length
* @param attribute the attribute for this packet, such as packet need ACK
* @return 0 means success, other means failed
*/
int CommProtocolPacketAssembleAndSend(CommType type,
                                      CommCmd cmd,
                                      char *payload,
                                      CommPayloadLen payload_len,
                                      CommAttribute *attribute);
```

2.2.4. CommProtocolReceiveUartData

```
/**
* @brief receive orignial uart data
* @param buf the uart data buffer pointer
* @param len the uart data length
* @return void
*/
void CommProtocolReceiveUartData(char *buf, int len);
```

3. Protocol 帧结构

3.1.1. Protocol layout

```
/*-------------------------------------------------------------------------*/
/*           layout of uart communication app protocol                     */
/*-------------------------------------------------------------------------*/
/*--6byte-|-1byte-|-1byte-|-2byte-|-2byte-|-2byte-|-2byte-|-N byte-*/
/*"uArTcP"|  seq   |  ctrl  |  cmd  | crc16 |  len  |cs(len)|payload */
/*-------------------------------------------------------------------------*/

/*---------------------------ack frame-----------------------------------*/
/*"uArTcP"|  seq  |  0x0  |  0x0  | crc16 |  0x0  |  0x0  |  NULL  */
/*-------------------------------------------------------------------------*/

/*-------------------------------------*/
/*-------------control-----------------*/
/*| 8 | 7 | 6 | 5 | 4 | 3 |  2  | 1 |*/
/*|RES|RES|RES|RES|RES|NACK|ACKED|ACK|*/
/*-------------------------------------*/
```

3.1.2. Protocol struct

```
typedef struct header {
  unsigned char sync[6];    /**< must be "uArTcP" */
  CommSequence  sequence;   /**< sequence number */
  CommControl   control;    /**< header ctrl */
  unsigned char cmd[2];     /**< command type, such as power on, power off etc */
  unsigned char checksum[2];          /**< checksum of packet, use crc16 */
  unsigned char payload_len[2];       /**< the length of payload */
  unsigned char payload_len_crc16[2];/**< the crc16 of payload_len */
  char          payload[0];           /**< the payload */
} __attribute__ ((packed)) CommProtocolPacket;
```

3.1.3. Protocol ACK

通过 CommProtocolPacketAssembleAndSend 接口参数 attribute 配置是否启用可靠传输，可靠传输时，需接收方协议栈回复 ACK 信息，实现可靠传输。

ACK 为协议栈内部逻辑，当配置为可靠传输时，协议栈自动回复 ACK 信息，上层应用无需关注。

ACK 帧格式详情可参阅 3.1.1 protocol ack frame。

3.1.4. Protocol 重要参数指标

1、最大payload：8182 字节。即超过 8182 字节将被拒绝发送、拒绝拆包解析。
2、自动内存回收阈值：1024 字节。由于接收到的数据长度不确定（即每一个command 帧字节可不同），接收解析器会对 protocol buffer 自动扩容、收缩，以此来减少内存开销，protocol buffer 初始长度 16 字节，扩容策略为当前长度的 2 倍（即 16、32、64、128...）当

解析出一帧有效数据后，会自动回收大于等于 1024 字节的 heap 内存；小于 1024 字节选择常驻内存，以此优化内存分配性能，直到调用 CommProtocolFinal 销毁。

3、ack 超时：目前设定 200ms。

4、自动重传：可靠传输发送方 200ms 未收到 ack 帧，会自动重发 5 次（即最多发 6 次），仍等不到 ack 则返回错误码 E_UNI_COMM_PAYLOAD_ACK_TIMEOUT。

4. 接口调用流程

4.1. 已有自定义协议调用过程

```
#include "uni_uart.h"
#include "uni_log.h"
#define MAIN_TAG "main"
static void recv_uart_data(char *buf, int len)
    { LOGT(MAIN_TAG, "receive uart data");
    /**
     * 接收到 uart 数据，在此进行处理，数据可能是 byte by byte 形式，不能假设发送方发送了
     * 32 字节，本次回调出来就是 32 字节，只能保证多次回调后，最终 32 字节
     */
}
int main() {
    /**
     * step 1. 初始化 UART。
     * 初始化完成后，即可以实现串口数据的读写操作。
     */
    UartConfig config;
    config.device = UNI_UART1;
    config.parity = UNI_PARITY_NONE;
    config.speed = UNI_B_115200;
    config.stop = UNI_ONE_STOP_BIT;
    config.data_bit = 8;
    UartInitialize(&config,_recv_uart_data);
    /**
     * step 2. 写数据
     * 向 UART 写数据，请调用 UartWrite 接口
     */
    char buf[32];
```

```
    UartWrite(buf, sizeof(buf));
    LOGT(MAIN_TAG, "write random data to UART");
    /**
     * step 3. 反初始化 UART。
     */
    UartFinalize();
    return 0;
}
```

4.2. 蜂鸟 UART 协议调用过程

```
#include "uni_uart.h"
#include "uni_communication.h"
#include "uni_log.h"
#define MAIN_TAG "main"
typedef enum {
    POWER_ON = 0,
    POWER_FF,
} CommandType;
static void recv_customer_packet(CommPacket *packet)
    { LOGT(MAIN_TAG, "receive packet, cmd=%d, payload_len=%d",
                packet->cmd, packet->payload_len);
}
int main() {
    /**
     * step 1. 初始化 UART。
     *  初始化完成后，即可以实现串口数据的读写操作。
     */
    UartConfig config;
    config.device = UNI_UART1;
    config.parity = UNI_PARITY_NONE;
    config.speed = UNI_B_115200;
    config.stop = UNI_ONE_STOP_BIT;
    config.data_bit = 8;
    UartInitialize(&config, CommProtocolReceiveUartData);
    /**
     * step 2. 初始化 protocol
```

```
    * 向 protocol 注册 UART 写方法、接收拆包后结构信息回调
    */
   CommProtocolInit(UartWrite, recv_customer_packet);
   /**
    * step 3. 写数据
    *  向 UART 写一帧满足协议格式的数据
    */
   char payload[32];
   for (int i = 0; i < sizeof(payload); i++)
       { payload[i] = (char)i;
   }
   CommAttribute attribute;
   attribute.reliable = true;
   /**
    * payload、attribute 设置为 NULL 则代表无 payload，不需要 ack
    * attribute.reliable = true; 则表示接收方必须发送 ack，否则本次发送失败，
    *  返回 ack 超时错误码
    */
   CommProtocolPacketAssembleAndSend(UNI_CUSTOMER_TYPE,
                                     POWER_ON,
                                     payload,
                                     sizeof(payload),
                                     &attribute);
   /**
    * step 3. 反初始化。
    */
   CommProtocolFinal();
   UartFinalize();
   return 0;
}
```

5. 移植协议栈到任意平台

5.1. 注册跨平台 Hooks

需要注册的跨平台 Hooks 分为三类：

1、动态内存分配函数，必备 APIs，必须注册，否则协议栈无法运行。
2、信号量函数，可选APIs，建议注册，可有效提升协议栈性能。
3、睡眠函数，必备API，必须注册。

```c
/**
 * 协议栈可移植函数钩子注册指针集结构体
 */
typedef struct {
  /* 动态内存分配相关的函数 */
  void* (*malloc_fn)(unsigned long size);            /**< malloc hook */
  void  (*free_fn)(void *ptr);                       /**< free hook */
  void* (*realloc_fn)(void *ptr, unsigned long size); /**< realloc hook */

  /* 信号量相关的函数 */
  void* (*sem_alloc_fn)(void);                       /**< 分配信号量句柄hook */
  void  (*sem_destroy_fn)(void *sem);                /**< 回收信号量句柄hook */
  int   (*sem_init_fn)(void *sem, unsigned int value); /**< 信号量初始化hook */
  int   (*sem_post_fn)(void *sem);                   /**< 信号量释放hook */
  int   (*sem_wait_fn)(void *sem);                   /**< 信号量等待hook */
  int   (*sem_timedwait_fn)(void *sem, unsigned int timeout_msecond); /**< 信号量超时等待hook */

  /* 睡眠函数 */
  int (*msleep_fn)(unsigned int msecond); /**< 睡眠hook */
} CommProtocolHooks;
/** @} uni_communication_uart_struct */

typedef void (*CommRecvPacketHandler)(CommPacket *packet);

/**@defgroup uni_communication_uart_inf
@{ */

/**
 * @brief     协议栈依赖的可移植函数，需要根据系统实际情况，进行注册
 * @param[in] hooks 注册函数指针集结构体
 * @return    void
 */
void CommProtocolRegisterHooks(CommProtocolHooks *hooks);
```

5.2. Porting demo

在开源代码example 目录分别展示了 RT-Thread、Linux、8051 单片机移植实例。
需要注意的点，已在 demo 中详细指出。

6. 附录

6.1. 错误码

结果码	含义	备注
(0)	成功	通用错误码
(-1)	错误	通用错误码
E_UNI_COMM_ALLOC_FAILED (-10001)	申请heap 内存失败	内存不足，申请失败

结果码	含义	备注
E_UNI_COMM_PAYLOAD_TOO_LONG (-10000)	发送的 payload 太长	当前支持最大payload为 8182 字节
E_UNI_COMM_PAYLOAD_ACK_TIMEOUT(-9999)	设置接收数据端需 ack 属性时，接收数据端 ack 超时	目前超时支持[50ms, 2000ms]区间

机芯HBM 离线方案开发指导手册 v1.0.1

云知声标准协议数据通讯接口

云知声标准协议数据通讯

类型定义

typedef void(* **_user_uni_recv**) (uni_u16 cmd, char *payload, int len)

接收数据回调函数定义 更多...

函数

int **user_uni_ucp_init** (**_user_uni_recv** cb_recv)

通讯初始化 更多...

void **user_uni_ucp_final** (void)

通讯逆初始化

int **user_uni_ucp_send** (uni_u16 cmd, char *payload, int len, uni_bool is_block)

通讯数据发送 更多...

详细描述

类型定义说明

◆ _user_uni_recv

typedef void(* _user_uni_recv) (uni_u16 cmd, char *payload, int len)

参数

cmd	命令值
payload	接收字符指针
len	接收字符长度

函数说明

◆ user_uni_ucp_init()

int user_uni_ucp_init (_user_uni_recv cb_recv)

参数

cb_recv	接收数据回调函数

返回值

0	操作成功
-1	操作失败

◆ user_uni_ucp_send()

```
int user_uni_ucp_send ( uni_u16 cmd,
                        char *    payload,
                        int       len,
                        uni_bool  is_block
                      )
```

参数

cmd	命令值
payload	发送字符指针
len	发送字符长度
is_block	是否堵塞

返回值

else 实际发送字符长度
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

云知声标准协议数据通讯接口

云知声标准协议数据通讯

类型定义

typedef void(* **_user_uni_recv**) (uni_u16 cmd, char *payload, int len)

接收数据回调函数定义 更多...

函数

int **user_uni_ucp_init** (**_user_uni_recv** cb_recv)

通讯初始化 更多...

void **user_uni_ucp_final** (void)

通讯逆初始化

int **user_uni_ucp_send** (uni_u16 cmd, char *payload, int len, uni_bool is_block)

通讯数据发送 更多...

详细描述

类型定义说明

◆ _user_uni_recv

typedef void(* _user_uni_recv) (uni_u16 cmd, char *payload, int len)

参数

cmd	命令值
payload	接收字符指针
len	接收字符长度

函数说明

◆ user_uni_ucp_init()

int user_uni_ucp_init (_user_uni_recv cb_recv)

参数

cb_recv	接收数据回调函数

返回值

0	操作成功
-1	操作失败

◆ user_uni_ucp_send()

```
int user_uni_ucp_send ( uni_u16 cmd,
                        char *   payload,
                        int      len,
                        uni_bool is_block
                      )
```

参数

cmd	命令值
payload	发送字符指针
len	发送字符长度
is_block	是否堵塞

返回值

else	实际发送字符长度
-1	操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

云知声标准协议数据通讯接口

云知声标准协议数据通讯

类型定义

typedef void(* **_user_uni_recv**) (uni_u16 cmd, char *payload, int len)

接收数据回调函数定义 更多...

函数

int **user_uni_ucp_init** (**_user_uni_recv** cb_recv)

通讯初始化 更多...

void **user_uni_ucp_final** (void)

通讯逆初始化

int **user_uni_ucp_send** (uni_u16 cmd, char *payload, int len, uni_bool is_block)

通讯数据发送 更多...

详细描述

类型定义说明

◆ _user_uni_recv

typedef void(* _user_uni_recv) (uni_u16 cmd, char *payload, int len)

参数

cmd	命令值
payload	接收字符指针
len	接收字符长度

函数说明

◆ user_uni_ucp_init()

int user_uni_ucp_init (_user_uni_recv cb_recv)

参数

cb_recv	接收数据回调函数

返回值

0	操作成功
-1	操作失败

◆ user_uni_ucp_send()

```
int user_uni_ucp_send ( uni_u16 cmd,
                        char *   payload,
                        int      len,
                        uni_bool is_block
                      )
```

参数

cmd	命令值
payload	发送字符指针
len	发送字符长度
is_block	是否堵塞

返回值

else	实际发送字符长度
-1	操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

云知声标准协议数据通讯接口

云知声标准协议数据通讯

类型定义

typedef void(* **_user_uni_recv**) (uni_u16 cmd, char *payload, int len)

接收数据回调函数定义 更多...

函数

int **user_uni_ucp_init** (**_user_uni_recv** cb_recv)

通讯初始化 更多...

void **user_uni_ucp_final** (void)

通讯逆初始化

int **user_uni_ucp_send** (uni_u16 cmd, char *payload, int len, uni_bool is_block)

通讯数据发送 更多...

详细描述

类型定义说明

◆ _user_uni_recv

typedef void(* _user_uni_recv) (uni_u16 cmd, char *payload, int len)

参数

cmd	命令值
payload	接收字符指针
len	接收字符长度

函数说明

◆ user_uni_ucp_init()

int user_uni_ucp_init (_user_uni_recv cb_recv)

参数

cb_recv	接收数据回调函数

返回值

0	操作成功
-1	操作失败

◆ user_uni_ucp_send()

```
int user_uni_ucp_send ( uni_u16 cmd,
                        char *   payload,
                        int      len,
                        uni_bool is_block
                      )
```

参数

cmd	命令值
payload	发送字符指针
len	发送字符长度
is_block	是否堵塞

返回值

else	实际发送字符长度
-1	操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

云知声标准协议数据通讯接口

云知声标准协议数据通讯

类型定义

typedef void(* **_user_uni_recv**) (uni_u16 cmd, char *payload, int len)

接收数据回调函数定义 更多...

函数

int **user_uni_ucp_init** (**_user_uni_recv** cb_recv)

通讯初始化 更多...

void **user_uni_ucp_final** (void)

通讯逆初始化

int **user_uni_ucp_send** (uni_u16 cmd, char *payload, int len, uni_bool is_block)

通讯数据发送 更多...

详细描述

类型定义说明

◆ _user_uni_recv

typedef void(* _user_uni_recv) (uni_u16 cmd, char *payload, int len)

参数

cmd	命令值
payload	接收字符指针
len	接收字符长度

函数说明

◆ user_uni_ucp_init()

int user_uni_ucp_init (_user_uni_recv cb_recv)

参数

cb_recv	接收数据回调函数

返回值

0	操作成功
-1	操作失败

◆ user_uni_ucp_send()

```
int user_uni_ucp_send ( uni_u16 cmd,
                        char *   payload,
                        int      len,
                        uni_bool is_block
                      )
```

参数

cmd	命令值
payload	发送字符指针
len	发送字符长度
is_block	是否堵塞

返回值

else	实际发送字符长度
-1	操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

模块

云知声标准协议数据通讯示例

云知声标准协议数据通讯

模块

hb_uart_ucp.c

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_uart_ucp.c

云知声标准协议数据通讯 » 云知声标准协议数据通讯示例

可通过定义 user/inc/user_config.h 中的宏 USER_RUN_DEMO_SELECT 为 USER_DEMO_UART_UCP 运行示例程序

```
/*****************************************************************************
 ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
 **
*
* Description : hb_uart_ucp.c
* Author : shangjinlong@unisound.com
* Date : 2020.04.25
*
*****************************************************************************
 **/
#include "user_uni_ucp.h"
#define TAG "hb_ucp"
/* 接收到串口数据后，该接口将被触发回调 */
static void _hb_ucp_recv(uni_u16 cmd, char *payload, int len) {
```

```
LOGT(TAG, "recv ucp packet, cmd=%d, len=%d", cmd, len);
int i;
LOGR(TAG, "payload: ");
for (i = 0; i < len; i++) {
LOGR(TAG, "02x ", payload[i]);
}
LOGR(TAG, "\n");
}
//消息唯一索引码，不可重叠
//规定[1, 10000]消息号预留给客户扩展；[10001, 65535]云知声内部使用
enum {
e_MSG_HEARTBEAT  = 10001,
e_MSG_DEMO,
};
static void _hb_ucp_send_task(void *args) {
char buf[4] = {'p', 'i', 'n', 'g'};
int ret;
while (1) {
LOGT(TAG, "hb uart ucp send heartbeat: ping");
//is_block = 1 指可靠传输，必须接收方 ack 确认（TCP 模式），0 UDP 模式
ret = user_uni_ucp_send(e_MSG_HEARTBEAT, buf, sizeof(buf), 1);
LOGT(TAG, "hb uart send done, ret=%d", ret);
uni_msleep(2000);
}
}
int hb_uart_ucp(void) {
//将宏 USE_UNIONE_PROTOCOL 打开，设置为 1
int ret = user_uni_ucp_init(_hb_ucp_recv);
if (ret != 0) {
LOGE(TAG, "hb uart ucp init failed");
return -1;
}
//启动一个线程，模拟发送串口消息任务
uni_pthread_t  pid;
thread_param param;
param.stack_size = STACK_SMALL_SIZE;
param.priority = OS_PRIORITY_NORMAL;
strncpy(param.task_name, "hb-ucp", sizeof(param.task_name) - 1);
uni_pthread_create(&pid, &param, _hb_ucp_send_task, NULL);
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

数据存储

数据存储接口参考user/inc/user_flash.h 更多...

模块

数据存储说明

数据存储接口

数据存储示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

数据存储说明

数据存储

数据存储说明:

1. 以 key-value 形式保存自定义值到flash 中，常用于保存配置信息
2. 可用于该功能的空间为 64k

机芯HBM 离线方案开发指导手册 v1.0.1

数据存储接口

数据存储

函数

int **user_flash_init** (void)

flash 初始化 更多...

void **user_flash_final** (void)

flash 逆初始化

int **user_flash_set_env_blob** (const char *key, const void *value_buf, int buf_len)

保存键值数据对到 flash 更多...

int **user_flash_get_env_blob** (const char *key, void *value_buf, int buf_len, int *save_len)

读取输入键的值 更多...

详细描述

函数说明

◆ user_flash_get_env_blob()

```
int user_flash_get_env_blob ( const char * key,
                              void *       value_buf,
                              int          buf_len,
                              int *        save_len
                            )
```

Get a blob ENV value by key name.

参数

key	ENV name
value_buf	ENV blob buffer
buf_len	ENV blob buffer length
save_len	return the length of the value saved on the flash, 0: NOT found

返回

the actually get size on successful

◆ user_flash_init()

```
int user_flash_init ( void )
```

返回值

0	操作成功
-1	操作失败

◆ user_flash_set_env_blob()

```
int user_flash_set_env_blob ( const char *  key,
                              const void * value_buf,
                              int          buf_len
                            )
```

参数

key	键名
value_buf	键值

buf_len　键值长度

返回

-成功：返回键值长度 -失败：-1

机芯HBM 离线方案开发指导手册 v1.0.1

数据存储接口

数据存储

函数

int **user_flash_init** (void)

flash 初始化 更多...

void **user_flash_final** (void)

flash 逆初始化

int **user_flash_set_env_blob** (const char *key, const void *value_buf, int buf_len)

保存键值数据对到 flash 更多...

int **user_flash_get_env_blob** (const char *key, void *value_buf, int buf_len, int *save_len)

读取输入键的值 更多...

详细描述

函数说明

◆ user_flash_get_env_blob()

```
int user_flash_get_env_blob ( const char * key,
                              void *       value_buf,
                              int          buf_len,
                              int *        save_len
                            )
```

Get a blob ENV value by key name.

参数

key	ENV name
value_buf	ENV blob buffer
buf_len	ENV blob buffer length
save_len	return the length of the value saved on the flash, 0: NOT found

返回

the actually get size on successful

◆ user_flash_init()

```
int user_flash_init ( void )
```

返回值

0	操作成功
-1	操作失败

◆ user_flash_set_env_blob()

```
int user_flash_set_env_blob ( const char *  key,
                              const void * value_buf,
                              int          buf_len
                            )
```

参数

key	键名
value_buf	键值

buf_len　键值长度

返回

-成功：返回键值长度 -失败：-1

机芯HBM 离线方案开发指导手册 v1.0.1

数据存储接口

数据存储

函数

int **user_flash_init** (void)

flash 初始化 更多...

void **user_flash_final** (void)

flash 逆初始化

int **user_flash_set_env_blob** (const char *key, const void *value_buf, int buf_len)

保存键值数据对到 flash 更多...

int **user_flash_get_env_blob** (const char *key, void *value_buf, int buf_len, int *save_len)

读取输入键的值 更多...

详细描述

函数说明

◆ user_flash_get_env_blob()

```
int user_flash_get_env_blob ( const char * key,
                              void *       value_buf,
                              int          buf_len,
                              int *        save_len
                            )
```

Get a blob ENV value by key name.

参数

key	ENV name
value_buf	ENV blob buffer
buf_len	ENV blob buffer length
save_len	return the length of the value saved on the flash, 0: NOT found

返回

the actually get size on successful

◆ user_flash_init()

```
int user_flash_init ( void )
```

返回值

0	操作成功
-1	操作失败

◆ user_flash_set_env_blob()

```
int user_flash_set_env_blob ( const char *  key,
                              const void * value_buf,
                              int          buf_len
                            )
```

参数

key	键名
value_buf	键值

buf_len　键值长度

返回

-成功：返回键值长度 -失败：-1

机芯HBM 离线方案开发指导手册 v1.0.1

数据存储接口

数据存储

函数

int **user_flash_init** (void)

flash 初始化 更多...

void **user_flash_final** (void)

flash 逆初始化

int **user_flash_set_env_blob** (const char *key, const void *value_buf, int buf_len)

保存键值数据对到 flash 更多...

int **user_flash_get_env_blob** (const char *key, void *value_buf, int buf_len, int *save_len)

读取输入键的值 更多...

详细描述

函数说明

◆ user_flash_get_env_blob()

```
int user_flash_get_env_blob ( const char * key,
                              void *       value_buf,
                              int          buf_len,
                              int *        save_len
                            )
```

Get a blob ENV value by key name.

参数

key	ENV name
value_buf	ENV blob buffer
buf_len	ENV blob buffer length
save_len	return the length of the value saved on the flash, 0: NOT found

返回

the actually get size on successful

◆ user_flash_init()

```
int user_flash_init ( void )
```

返回值

0	操作成功
-1	操作失败

◆ user_flash_set_env_blob()

```
int user_flash_set_env_blob ( const char *  key,
                              const void * value_buf,
                              int          buf_len
                            )
```

参数

key	键名
value_buf	键值

buf_len　键值长度

返回

-成功：返回键值长度 -失败：-1

机芯HBM 离线方案开发指导手册 v1.0.1

数据存储接口

数据存储

函数

int **user_flash_init** (void)

flash 初始化 更多...

void **user_flash_final** (void)

flash 逆初始化

int **user_flash_set_env_blob** (const char *key, const void *value_buf, int buf_len)

保存键值数据对到 flash 更多...

int **user_flash_get_env_blob** (const char *key, void *value_buf, int buf_len, int *save_len)

读取输入键的值 更多...

详细描述

函数说明

◆ user_flash_get_env_blob()

```
int user_flash_get_env_blob ( const char * key,
                              void *       value_buf,
                              int          buf_len,
                              int *        save_len
                            )
```

Get a blob ENV value by key name.

参数

key	ENV name
value_buf	ENV blob buffer
buf_len	ENV blob buffer length
save_len	return the length of the value saved on the flash, 0: NOT found

返回

the actually get size on successful

◆ user_flash_init()

```
int user_flash_init ( void )
```

返回值

0	操作成功
-1	操作失败

◆ user_flash_set_env_blob()

```
int user_flash_set_env_blob ( const char *  key,
                              const void * value_buf,
                              int          buf_len
                            )
```

参数

key	键名
value_buf	键值

buf_len　键值长度

返回

-成功：返回键值长度 -失败：-1

机芯HBM 离线方案开发指导手册 v1.0.1

模块

数据存储示例

数据存储

模块

hb_flash_example.c

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_flash_example.c

数据存储 » 数据存储示例

可通过定义 user/inc/user_config.h 中的宏 USER_RUN_DEMO_SELECT 为 USER_DEMO_FLASH 运行示例程序

```
/*****************************************************************************
 ***
* Copyright (C) 2020-2020 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
 **
*
* Description : user_flash_example.c
* Author : yuanshifeng@unisound.com
* Date : 2020.04.25
*
*****************************************************************************
 **/
#include "user_flash.h"
#define TAG "flash_example"
#define BLOB_KEY_EXAMPLE_INT "EXAMPLE_INT"
#define BLOB_TYPE_U32 uni_u32
```

```
int hb_flash(void) {
BLOB_TYPE_U32  example_int = 0;
int save_len = 0;
if (0 != user_flash_init())
LOGE(TAG, { "user_flash_init
          failed.");
return -1;
}
user_flash_get_env_blob(BLOB_KEY_EXAMPLE_INT, &example_int,
sizeof(BLOB_TYPE_U32), &save_len);
if (save_len != sizeof(BLOB_TYPE_U32)) {
//仅首次开机进入，发现 flash 中没有"EXAMPLE_INT"字段，保存一个 88 进去，待下次开机查
      看是否能直接读取出这个值。
BLOB_TYPE_U32 set_int = 88;
LOGT(TAG, "cannot found %s in flash, save value %d into flash",
      BLOB_KEY_EXAMPLE_INT, set_int);
 user_flash_set_env_blob(BLOB_KEY_EXAMPLE_INT,
      &set_int,sizeof(BLOB_TYPE_U32));
}
if (example_int > 0) {
//打印读取 flash 中"EXAMPLE_INT"字段的值，读出 88 表示成功保存了值到 flash 中
LOGT(TAG, "read %s in flash success, value is %d", BLOB_KEY_EXAMPLE_INT,
      example_int);
}
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发

串口收发接口参考user/inc/user_uart.h 更多...

模块

- 串口收发说明
- 串口收发接口
- 串口收发示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发说明

串口收发

串口收发说明:

1. 使用该功能需设置 user/inc/user_config.h 中的宏**USE_UNIONE_PROTOCOL** 设为 0
2. 串口透传收发

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发接口

串口收发

类型定义

typedef void(* **_user_uart_recv**) (char *buf, int len)

接收数据回调函数定义 更多...

函数

int **user_uart_init** (**_user_uart_recv** cb_recv)

串口初始化 更多...

void **user_uart_final** (void)

串口逆初始化

int **user_uart_send** (char *buf, int len)

串口数据发送 更多...

详细描述

类型定义说明

◆ _user_uart_recv

typedef void(* _user_uart_recv) (char *buf, int len)

参数

buf 接收字符指针
len 接收字符长度

函数说明

◆ user_uart_init()

int user_uart_init (_user_uart_recv cb_recv)

参数

cb_recv 接收数据回调函数

返回值

0 操作成功
-1 操作失败

◆ user_uart_send()

```
int user_uart_send ( char *  buf,
                     int     len
                   )
```

参数

buf 发送字符指针
len 发送字符长度

返回值

else 实际发送字符长度
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发接口

串口收发

类型定义

typedef void(* **_user_uart_recv**) (char *buf, int len)

接收数据回调函数定义 更多...

函数

int **user_uart_init** (**_user_uart_recv** cb_recv)

串口初始化 更多...

void **user_uart_final** (void)

串口逆初始化

int **user_uart_send** (char *buf, int len)

串口数据发送 更多...

详细描述

类型定义说明

◆ _user_uart_recv

typedef void(* _user_uart_recv) (char *buf, int len)

参数

buf 接收字符指针
len 接收字符长度

函数说明

◆ user_uart_init()

int user_uart_init (_user_uart_recv cb_recv)

参数

cb_recv 接收数据回调函数

返回值

0 操作成功
-1 操作失败

◆ user_uart_send()

```
int user_uart_send ( char *  buf,
                     int     len
                   )
```

参数

buf 发送字符指针
len 发送字符长度

返回值

else 实际发送字符长度
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发接口

串口收发

类型定义

typedef void(* **_user_uart_recv**) (char *buf, int len)

接收数据回调函数定义 更多...

函数

int **user_uart_init** (**_user_uart_recv** cb_recv)

串口初始化 更多...

void **user_uart_final** (void)

串口逆初始化

int **user_uart_send** (char *buf, int len)

串口数据发送 更多...

详细描述

类型定义说明

◆ _user_uart_recv

typedef void(* _user_uart_recv) (char *buf, int len)

参数

buf 接收字符指针
len 接收字符长度

函数说明

◆ user_uart_init()

int user_uart_init (_user_uart_recv cb_recv)

参数

cb_recv 接收数据回调函数

返回值

0 操作成功
-1 操作失败

◆ user_uart_send()

```
int user_uart_send ( char *  buf,
                     int     len
                   )
```

参数

buf 发送字符指针
len 发送字符长度

返回值

else 实际发送字符长度
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发接口

串口收发

类型定义

typedef void(* **_user_uart_recv**) (char *buf, int len)

接收数据回调函数定义 更多...

函数

int **user_uart_init** (**_user_uart_recv** cb_recv)

串口初始化 更多...

void **user_uart_final** (void)

串口逆初始化

int **user_uart_send** (char *buf, int len)

串口数据发送 更多...

详细描述

类型定义说明

◆ _user_uart_recv

typedef void(* _user_uart_recv) (char *buf, int len)

参数

buf 接收字符指针
len 接收字符长度

函数说明

◆ user_uart_init()

int user_uart_init (_user_uart_recv cb_recv)

参数

cb_recv 接收数据回调函数

返回值

0 操作成功
-1 操作失败

◆ user_uart_send()

```
int user_uart_send ( char *  buf,
                     int     len
                   )
```

参数

buf 发送字符指针
len 发送字符长度

返回值

else 实际发送字符长度
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发接口

串口收发

类型定义

typedef void(* **_user_uart_recv**) (char *buf, int len)

接收数据回调函数定义 更多...

函数

int **user_uart_init** (**_user_uart_recv** cb_recv)

串口初始化 更多...

void **user_uart_final** (void)

串口逆初始化

int **user_uart_send** (char *buf, int len)

串口数据发送 更多...

详细描述

类型定义说明

◆ _user_uart_recv

typedef void(* _user_uart_recv) (char *buf, int len)

参数

buf 接收字符指针
len 接收字符长度

函数说明

◆ user_uart_init()

int user_uart_init (_user_uart_recv cb_recv)

参数

cb_recv 接收数据回调函数

返回值

0 操作成功
-1 操作失败

◆ user_uart_send()

```
int user_uart_send ( char *  buf,
                     int     len
                   )
```

参数

buf 发送字符指针
len 发送字符长度

返回值

else 实际发送字符长度
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

串口收发示例

串口收发

模块

hb_uart.c

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_uart.c

串口收发 » 串口收发示例

可通过定义 user/inc/user_config.h 中的宏 **USER_RUN_DEMO_SELECT** 为 **USER_DEMO_UART** 运行示例程序

```
/*****************************************************************************
 ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
 **
*
* Description : hb_uart.c
* Author : shangjinlong@unisound.com
* Date : 2020.04.25
*
*****************************************************************************
 **/
#include "user_uart.h"
#define TAG "hb_uart"
/* 接收到串口数据后，该接口将被触发回调 */
static void _hb_uart_recv(char *buf, int len) {
```

```
int i;
for (i = 0; i < len; i++) {
LOGR(TAG, "%02x ", (uint8_t)buf[i]);
}
}
static void _hb_uart_send_task(void *args) {
char buf[6] = {1, 2, 3, 4, 5, 6};
int ret;
/* 每隔 2 秒发送一次 */
while (1) {
LOGT(TAG, "hb uart send[1, 2, 3, 4, 5, 6]");
ret = user_uart_send(buf, sizeof(buf));
LOGT(TAG, "hb uart send done, ret=%d", ret);
uni_msleep(2000);
}
}
int hb_uart(void) {
int ret = user_uart_init(_hb_uart_recv);
if (ret != 0) {
LOGE(TAG, "hb uart init failed");
return -1;
}
//启动一个线程，模拟发送串口消息任务
uni_pthread_t pid;
thread_param param;
param.stack_size = STACK_SMALL_SIZE;
param.priority = OS_PRIORITY_NORMAL;
strncpy(param.task_name, "hb-uart", sizeof(param.task_name) - 1);
uni_pthread_create(&pid, &param, _hb_uart_send_task, NULL);
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

定时器

定时器接口参考 user/inc/user_timer.h 更多...

模块

- 定时器说明
- 定时器接口
- 定时器类型定义
- 定时器示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

定时器说明

定时器

定时器说明:

1. user/src/examples/drivers/user_sw_timer.c 是基于一个硬件timer，配置成多个软定时器的示例，达到突破硬件 timer 个数限制的目的
2. user/src/user_timer.c 为硬件定时器接口

机芯HBM 离线方案开发指导手册 v1.0.1

函数

定时器接口

定时器

函数

int	**user_timer_init** (**eTIMER_IDX** idx, uint32_t usec, bool single_shot, **usr_timer_callback** cb) 定时器初始化 更多...
void	**user_timer_start** (**eTIMER_IDX** idx) 启动定时器 更多...
void	**user_timer_pause** (**eTIMER_IDX** idx) 暂停定时器计数 更多...
void	**user_timer_resume** (**eTIMER_IDX** idx) 重新使能定时器计数 更多...

详细描述

函数说明

◆ user_timer_init()

```
int user_timer_init ( eTIMER_IDX          idx,
                      uint32_t            usec,
                      bool                single_shot,
                      usr_timer_callback  cb
                    )
```

参数

idx	定时器索引
usec	定时设置, 单位：微妙(us)
single_shot	0：连续模式，超时之后，定时器会重新装载计数值．1：单次模式，超时之后，定时器停止工作。
cb	timer 中断处理回调函数，会在中断处理中调用此函数

返回值

0 操作成功
-1 操作失败

◆ user_timer_pause()

```
void user_timer_pause ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_resume()

```
void user_timer_resume ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_start()

```
void user_timer_start ( eTIMER_IDX idx )
```

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器接口

定时器

函数

int	**user_timer_init** (**eTIMER_IDX** idx, uint32_t usec, bool single_shot, **usr_timer_callback** cb) 定时器初始化 更多...
void	**user_timer_start** (**eTIMER_IDX** idx) 启动定时器 更多...
void	**user_timer_pause** (**eTIMER_IDX** idx) 暂停定时器计数 更多...
void	**user_timer_resume** (**eTIMER_IDX** idx) 重新使能定时器计数 更多...

详细描述

函数说明

◆ user_timer_init()

```
int user_timer_init ( eTIMER_IDX        idx,
                      uint32_t          usec,
                      bool              single_shot,
                      usr_timer_callback cb
                    )
```

参数

idx	定时器索引
usec	定时设置, 单位：微妙(us)
single_shot	0：连续模式，超时之后，定时器会重新装载计数值．1：单次模式，超时之后，定时器停止工作。
cb	timer 中断处理回调函数，会在中断处理中调用此函数

返回值

0 操作成功
-1 操作失败

◆ user_timer_pause()

```
void user_timer_pause ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_resume()

```
void user_timer_resume ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_start()

```
void user_timer_start ( eTIMER_IDX idx )
```

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器接口

定时器

函数

int	**user_timer_init** (**eTIMER_IDX** idx, uint32_t usec, bool single_shot, **usr_timer_callback** cb) 定时器初始化 更多...
void	**user_timer_start** (**eTIMER_IDX** idx) 启动定时器 更多...
void	**user_timer_pause** (**eTIMER_IDX** idx) 暂停定时器计数 更多...
void	**user_timer_resume** (**eTIMER_IDX** idx) 重新使能定时器计数 更多...

详细描述

函数说明

◆ user_timer_init()

```
int user_timer_init ( eTIMER_IDX          idx,
                      uint32_t            usec,
                      bool                single_shot,
                      usr_timer_callback  cb
                    )
```

参数

idx	定时器索引
usec	定时设置, 单位：微妙(us)
single_shot	0：连续模式，超时之后，定时器会重新装载计数值．1：单次模式，超时之后，定时器停止工作。
cb	timer 中断处理回调函数，会在中断处理中调用此函数

返回值

0 操作成功
-1 操作失败

◆ user_timer_pause()

```
void user_timer_pause ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_resume()

```
void user_timer_resume ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_start()

```
void user_timer_start ( eTIMER_IDX idx )
```

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器接口

定时器

函数

int	**user_timer_init** (**eTIMER_IDX** idx, uint32_t usec, bool single_shot, **usr_timer_callback** cb) 定时器初始化 更多...
void	**user_timer_start** (**eTIMER_IDX** idx) 启动定时器 更多...
void	**user_timer_pause** (**eTIMER_IDX** idx) 暂停定时器计数 更多...
void	**user_timer_resume** (**eTIMER_IDX** idx) 重新使能定时器计数 更多...

详细描述

函数说明

◆ user_timer_init()

```
int user_timer_init ( eTIMER_IDX          idx,
                      uint32_t            usec,
                      bool                single_shot,
                      usr_timer_callback  cb
                    )
```

参数

idx	定时器索引
usec	定时设置, 单位：微妙(us)
single_shot	0：连续模式，超时之后，定时器会重新装载计数值．1：单次模式，超时之后，定时器停止工作。
cb	timer 中断处理回调函数，会在中断处理中调用此函数

返回值

0 操作成功
-1 操作失败

◆ user_timer_pause()

```
void user_timer_pause ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_resume()

```
void user_timer_resume ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_start()

```
void user_timer_start ( eTIMER_IDX idx )
```

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器接口

定时器

函数

int	**user_timer_init** (**eTIMER_IDX** idx, uint32_t usec, bool single_shot, **usr_timer_callback** cb) 定时器初始化 更多...
void	**user_timer_start** (**eTIMER_IDX** idx) 启动定时器 更多...
void	**user_timer_pause** (**eTIMER_IDX** idx) 暂停定时器计数 更多...
void	**user_timer_resume** (**eTIMER_IDX** idx) 重新使能定时器计数 更多...

详细描述

函数说明

◆ user_timer_init()

```
int user_timer_init ( eTIMER_IDX          idx,
                      uint32_t            usec,
                      bool                single_shot,
                      usr_timer_callback  cb
                    )
```

参数

idx	定时器索引
usec	定时设置, 单位：微妙(us)
single_shot	0：连续模式，超时之后，定时器会重新装载计数值．1：单次模式，超时之后，定时器停止工作。
cb	timer 中断处理回调函数，会在中断处理中调用此函数

返回值

0 操作成功
-1 操作失败

◆ user_timer_pause()

```
void user_timer_pause ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_resume()

```
void user_timer_resume ( eTIMER_IDX idx )
```

参数

idx 定时器索引

◆ user_timer_start()

```
void user_timer_start ( eTIMER_IDX idx )
```

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器类型定义

定时器

类型定义

typedef enum TIMER_IDX	eTIMER_IDX 定时器索引
typedef void(*	usr_timer_callback) (eTIMER_IDX idx) 定时器回调函数 更多...

枚举

enum	TIMER_IDX { **eTIMER2** = 1, **eTIMER5** = 4, **eTIMER6** = 5 } 定时器索引

详细描述

类型定义说明

◆ usr_timer_callback

typedef void(* usr_timer_callback) (eTIMER_IDX idx)

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器类型定义

定时器

类型定义

typedef enum TIMER_IDX	eTIMER_IDX 定时器索引
typedef void(*	usr_timer_callback) (eTIMER_IDX idx) 定时器回调函数 更多...

枚举

enum	TIMER_IDX { **eTIMER2** = 1, **eTIMER5** = 4, **eTIMER6** = 5 } 定时器索引

详细描述

类型定义说明

◆ usr_timer_callback

typedef void(* usr_timer_callback) (eTIMER_IDX idx)

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器类型定义

定时器

类型定义

typedef enum **TIMER_IDX** **eTIMER_IDX**

定时器索引

typedef void(* **usr_timer_callback**) (**eTIMER_IDX** idx)

定时器回调函数 更多...

枚举

enum **TIMER_IDX** { **eTIMER2** = 1, **eTIMER5** = 4, **eTIMER6** = 5 }

定时器索引

详细描述

类型定义说明

◆ usr_timer_callback

typedef void(* usr_timer_callback) (**eTIMER_IDX** idx)

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器类型定义

定时器

类型定义

typedef enum **TIMER_IDX** **eTIMER_IDX**

定时器索引

typedef void(* **usr_timer_callback**) (**eTIMER_IDX** idx)

定时器回调函数 更多...

枚举

enum **TIMER_IDX** { **eTIMER2** = 1, **eTIMER5** = 4, **eTIMER6** = 5 }

定时器索引

详细描述

类型定义说明

◆ usr_timer_callback

typedef void(* usr_timer_callback) (**eTIMER_IDX** idx)

参数

idx 定时器索引

机芯HBM 离线方案开发指导手册 v1.0.1

定时器示例

定时器

模块

- hb_sw_timers.c
- user_timer_buzzer.c

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_sw_timers.c

定时器 » 定时器示例

可通过定义 user/inc/user_config.h 中的宏 **USER_RUN_DEMO_SELECT** 为 **USER_DEMO_TIMERS** 运行示例程序

```
/*****************************************************************************
 ***
* Copyright (C) 2020-2020 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
 **
*
* Description : user_timer_buzzer.c
* Author : liuwenzheng@unisound.com
* Date : 2020.04.21
*
*****************************************************************************
 **/
#include "user_sw_timer.h"
#include "user_gpio.h"
#define TAG "sw_timer_example"
#define USR_TIMER_NUM eTIMER2 // user timer2
```

```
#define  PARTICLE_MS 20 // 20ms
typedef  struct MyTimer_s {
timer_handle_t  handle;
int cnt;
}MyTimer;
static  MyTimer m_timers[4] =
{
{INVALID_TIMER_HANDLE,   0},
{INVALID_TIMER_HANDLE,   0},
{INVALID_TIMER_HANDLE,   0},
{INVALID_TIMER_HANDLE,   0}
};
static  void _sw_timer_cb(timer_handle_t timer) {
int i;
for (i =0; i < 4; i++) {
if (timer == m_timers[i].handle) {
m_timers[i].cnt++;
break;
}
}
}
static void _timer_print_process(void *args) {
while (1) {
int i = 0;
for (i =0; i < 4; i++) {
LOGT(TAG,  "---- Timer %d count = %d ----", i, m_timers[i].cnt);
}
uni_sleep(1);
}
}
static Result _create_timer_print_thread(void) {
thread_param param;
uni_pthread_t  pid;
uni_memset(&param, 0, sizeof(param));
param.stack_size = STACK_SMALL_SIZE;
param.priority = OS_PRIORITY_LOW;
uni_strncpy(param.task_name, "buzzer_set", sizeof(param.task_name) - 1);
if (0 != uni_pthread_create(&pid, &param,
_timer_print_process, NULL)) {
LOGE(TAG, "create thread failed");
return  E_FAILED;
}
uni_pthread_detach(pid);
return E_OK;
```

```
}
int hb_sw_timer(void) {
if (E_OK != _create_timer_print_thread()) {
LOGE(TAG, "_create_timer_print_thread failed");
return -1;
}
user_sw_timer_init(USR_TIMER_NUM, PARTICLE_MS);
m_timers[0].handle = user_sw_timer_add(PARTICLE_MS * 5, false,
        _sw_timer_cb);
m_timers[1].handle = user_sw_timer_add(PARTICLE_MS * 10, false,
        _sw_timer_cb);
m_timers[2].handle = user_sw_timer_add(PARTICLE_MS * 20, false,
        _sw_timer_cb);
m_timers[3].handle = user_sw_timer_add(PARTICLE_MS * 40, true,
        _sw_timer_cb); //count 1 time only
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

user_timer_buzzer.c

定时器 » 定时器示例

可通过定义 user/inc/user_config.h 中的宏 USER_RUN_DEMO_SELECT 为 USER_DEMO_BUZZER 运行示例程序

```
/*****************************************************************************
 ***
* Copyright (C) 2020-2020 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
 **
*
* Description : user_timer_buzzer.c
* Author : liuwenzheng@unisound.com
* Date : 2020.04.21
*
*****************************************************************************
 **/
#include "user_timer_buzzer.h"
#include "user_timer.h"
#include "user_gpio.h"
#define TAG "buzzer"
```

```
#define BUZZER_HZ_MAX 4000
typedef struct {
eTIMER_IDX timer_idx;
GPIO_NUMBER gpio_num;
uni_u32 hz;
uni_u32 time_us;
}buzzer_context_t;
static buzzer_context_t g_buzzer = {0};
static void _buzzer_timer_handle(eTIMER_IDX TimerIdx) {
#if GPIO_OUT_DEF_VAL
volatile static int cur_level = 1;
#else
volatile static int cur_level = 0;
#endif
if (g_buzzer.timer_idx == TimerIdx) {
cur_level ^= 1;
user_gpio_set_value(g_buzzer.gpio_num, cur_level);
}
}
int user_timer_buzzer_init(GPIO_NUMBER num, eTIMER_IDX idx)
{ user_gpio_init();
if (0 != user_gpio_set_mode(num, GPIO_MODE_OUT))
LOGE(TAG, { "user_gpio_set_mode failed.");
}
g_buzzer.timer_idx = idx;
g_buzzer.hz = 0;
g_buzzer.gpio_num  = num;
g_buzzer.time_us = 0;
return 0;
}
void user_timer_buzzer_final(void)
{ user_timer_pause(g_buzzer.timer_idx);
}
int user_timer_buzzer_set_hz(uni_u32 hz) {
if (hz > BUZZER_HZ_MAX) {
LOGE(TAG, "The maximum frequency is 4000Hz.");
return -1;
}
user_timer_pause(g_buzzer.timer_idx);
g_buzzer.hz = hz;
g_buzzer.time_us = 1000 * 1000 / g_buzzer.hz;
user_timer_init(g_buzzer.timer_idx, g_buzzer.time_us, false,
_buzzer_timer_handle);
user_timer_start(g_buzzer.timer_idx);
```

```
    return 0;
}

uint32_t user_timer_buzzer_get_hz(void) {
    return g_buzzer.hz;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO

GPIO 接口参考 user/inc/user_gpio.h 更多...

模块

- GPIO 说明
- GPIO 相关接口
- GPIO 类型定义
- GPIO 示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 说明

GPIO

GPIO 说明:

1. 可配置 gpio 工作模式（pinmux）
2. 当为 gpio 功能时，可配置PD/PU, DIR, VALUE
3. 当为中断功能时，可注册中断回调函数 **gpio_interrupt_cb** ， 设置中断方式 **GPIO_INT_TYPE**

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int **user_gpio_interrupt_enable** (void)

使能 GPIO 中断 更多...

int **user_gpio_interrupt_disable** (void)

关闭 GPIO 中断 更多...

int **user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb)

注册 GPIO 中断 更多...

int **user_gpio_clear_interrupt** (**GPIO_NUMBER** num)

注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int	**user_gpio_interrupt_enable** (void) 使能 GPIO 中断 更多...
int	**user_gpio_interrupt_disable** (void) 关闭 GPIO 中断 更多...
int	**user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb) 注册 GPIO 中断 更多...
int	**user_gpio_clear_interrupt** (**GPIO_NUMBER** num) 注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int **user_gpio_interrupt_enable** (void)

使能 GPIO 中断 更多...

int **user_gpio_interrupt_disable** (void)

关闭 GPIO 中断 更多...

int **user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb)

注册 GPIO 中断 更多...

int **user_gpio_clear_interrupt** (**GPIO_NUMBER** num)

注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int	**user_gpio_interrupt_enable** (void) 使能 GPIO 中断 更多...
int	**user_gpio_interrupt_disable** (void) 关闭 GPIO 中断 更多...
int	**user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb) 注册 GPIO 中断 更多...
int	**user_gpio_clear_interrupt** (**GPIO_NUMBER** num) 注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

_gpio_interrupt_cb cb
)

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

int user_gpio_set_mode (GPIO_NUMBER num,
GPIO_MODE mode
)

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

int user_gpio_set_pull_mode (GPIO_NUMBER num,
GPIO_PULL_MODE mode
)

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int	**user_gpio_interrupt_enable** (void) 使能 GPIO 中断 更多...
int	**user_gpio_interrupt_disable** (void) 关闭 GPIO 中断 更多...
int	**user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb) 注册 GPIO 中断 更多...
int	**user_gpio_clear_interrupt** (**GPIO_NUMBER** num) 注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

_gpio_interrupt_cb cb
)

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

int user_gpio_set_mode (GPIO_NUMBER num,
GPIO_MODE mode
)

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

int user_gpio_set_pull_mode (GPIO_NUMBER num,
GPIO_PULL_MODE mode
)

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int	**user_gpio_interrupt_enable** (void) 使能 GPIO 中断 更多...
int	**user_gpio_interrupt_disable** (void) 关闭 GPIO 中断 更多...
int	**user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb) 注册 GPIO 中断 更多...
int	**user_gpio_clear_interrupt** (**GPIO_NUMBER** num) 注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int **user_gpio_interrupt_enable** (void)

使能 GPIO 中断 更多...

int **user_gpio_interrupt_disable** (void)

关闭 GPIO 中断 更多...

int **user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb)

注册 GPIO 中断 更多...

int **user_gpio_clear_interrupt** (**GPIO_NUMBER** num)

注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int **user_gpio_interrupt_enable** (void)

使能 GPIO 中断 更多...

int **user_gpio_interrupt_disable** (void)

关闭 GPIO 中断 更多...

int **user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb)

注册 GPIO 中断 更多...

int **user_gpio_clear_interrupt** (**GPIO_NUMBER** num)

注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int **user_gpio_interrupt_enable** (void)

使能 GPIO 中断 更多...

int **user_gpio_interrupt_disable** (void)

关闭 GPIO 中断 更多...

int **user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb)

注册 GPIO 中断 更多...

int **user_gpio_clear_interrupt** (**GPIO_NUMBER** num)

注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int	**user_gpio_interrupt_enable** (void) 使能 GPIO 中断 更多...
int	**user_gpio_interrupt_disable** (void) 关闭 GPIO 中断 更多...
int	**user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb) 注册 GPIO 中断 更多...
int	**user_gpio_clear_interrupt** (**GPIO_NUMBER** num) 注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 相关接口

GPIO

函数

int **user_gpio_init** (void)

GPIO 初始化 更多...

void **user_gpio_final** (void)

GPIO 反初始化

int **user_gpio_set_mode** (**GPIO_NUMBER** num, **GPIO_MODE** mode)

设置 GPIO 工作模式 更多...

int **user_gpio_set_pull_mode** (**GPIO_NUMBER** num, **GPIO_PULL_MODE** mode)

设置 GPIO pull 模式 更多...

int **user_gpio_set_value** (**GPIO_NUMBER** num, int val)

设置 GPIO 端口数值 更多...

int **user_gpio_get_value** (**GPIO_NUMBER** num)

获取 GPIO 端口值 更多...

int	**user_gpio_interrupt_enable** (void) 使能 GPIO 中断 更多...
int	**user_gpio_interrupt_disable** (void) 关闭 GPIO 中断 更多...
int	**user_gpio_set_interrupt** (**GPIO_NUMBER** num, **GPIO_INT_TYPE** type, **_gpio_interrupt_cb** cb) 注册 GPIO 中断 更多...
int	**user_gpio_clear_interrupt** (**GPIO_NUMBER** num) 注销 GPIO 中断注册回调函数 更多...

详细描述

函数说明

◆ user_gpio_clear_interrupt()

int user_gpio_clear_interrupt (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

0 操作成功
-1 操作失败

◆ user_gpio_get_value()

int user_gpio_get_value (**GPIO_NUMBER** num)

参数

num GPIO 端口号

返回值

else 读取数值
-1 操作失败

◆ user_gpio_init()

int user_gpio_init (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_disable()

int user_gpio_interrupt_disable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_interrupt_enable()

int user_gpio_interrupt_enable (void)

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_interrupt()

int user_gpio_set_interrupt (GPIO_NUMBER num,
GPIO_INT_TYPE type,

```
                                    gpio_interrupt_cb  cb
                                )
```

参数

num GPIO 端口号
type GPIO 电平跳变模式
cb 中断处理回调函数

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_mode()

```
int user_gpio_set_mode ( GPIO_NUMBER num,
                         GPIO_MODE   mode
                       )
```

参数

num GPIO 端口号
mode GPIO 工作模式

返回值

0 操作成功
-1 操作失败

◆ user_gpio_set_pull_mode()

```
int user_gpio_set_pull_mode ( GPIO_NUMBER      num,
                              GPIO_PULL_MODE mode
                            )
```

参数

num GPIO 端口号
mode GPIO pull 模式

返回值

0 操作成功

-1 操作失败

◆ user_gpio_set_value()

```
int user_gpio_set_value ( GPIO_NUMBER num,
                          int         val
                        )
```

参数

num GPIO 端口号
val 0 or 1

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(*_gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号

is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号

is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号

is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 类型定义

GPIO

类型定义

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

中断回调函数 更多...

枚举

enum GPIO_NUMBER {
GPIO_NUM_A25 = 0, GPIO_NUM_A26, GPIO_NUM_A27, GPIO_NUM_A28,
GPIO_NUM_B0, GPIO_NUM_B1, GPIO_NUM_B2, GPIO_NUM_B3,
GPIO_NUM_B6, GPIO_NUM_B7, GPIO_NUM_B8, GPIO_NUM_MAX
}

GPIO 号 更多...

enum GPIO_MODE {
GPIO_MODE_IN = 0, GPIO_MODE_OUT, GPIO_MODE_ADC, GPIO_MODE_PWM,
GPIO_MODE_UART, GPIO_MODE_I2C, GPIO_MODE_SPI, GPIO_MODE_MAX
}

GPIO 模式

enum GPIO_PULL_MODE { GPIO_PULL_UP, GPIO_PULL_DOWN,

	GPIO_PULL_UP_DOWN }
	GPIO PU/PD 模式选择

enum	GPIO_INT_TYPE { **GPIO_INT_NEG_EDGE**, **GPIO_INT_POS_EDGE**, **GPIO_INT_BOTH_EDGE** }
	GPIO 中断方式选择

详细描述

类型定义说明

◆ _gpio_interrupt_cb

typedef void(* _gpio_interrupt_cb) (GPIO_NUMBER num, uni_bool is_high)

参数

num 中断io 号
is_high 是否高电平

枚举类型说明

◆ GPIO_NUMBER

enum GPIO_NUMBER

枚举值	
GPIO_NUM_A25	可用于 SPI-MISO / ADC / DMIC-DATA
GPIO_NUM_A26	可用于 SPI-CLK / ADC / DMIC-CLK
GPIO_NUM_A27	可用于 SPI-MOSI / PWM / ADC
GPIO_NUM_A28	已占用，PA 使能控制（喇叭功放静音）

GPIO_NUM_B0	已占用，SW-CLK（烧录器接口）
GPIO_NUM_B1	已占用，SW-DATA（烧录器接口）
GPIO_NUM_B2	可用于 UART1-TX / PWM / I2C-SCL
GPIO_NUM_B3	可用于 UART1-RX / PWM / I2C-SDA
GPIO_NUM_B6	UART1-RX（外设串口通信接收脚），不使能 UART 时可用做 GPIO
GPIO_NUM_B7	UART1-TX（外设串口通信发送脚），不使能 UART 是可用做 GPIO
GPIO_NUM_B8	已占用，虚拟Software UART-TX（Log 输出引脚，波特率 115200）

机芯HBM 离线方案开发指导手册 v1.0.1

GPIO 示例

GPIO

模块

- hb_gpio_adc.c
- hb_gpio_key.c

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_gpio_adc.c

可通过定义 user/inc/user_config.h 中的宏 USER_RUN_DEMO_SELECT 为 USER_DEMO_ADC_CTRL 运行示例程序

```
/*****************************************************************************
 ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
 **
*
* Description : hb_gpio_adc.c
* Author : wufangfang@unisound.com
* Date : 2020.04.25
*
*****************************************************************************
 **/
#include "user_gpio.h"
#include "user_ad_gl5528.h"
#include "user_asr.h"
#define TAG "hb_gpio_adc"
```

```
#define USE_AD_GL5528_DRIVER 1 //1: user user_ad_gl5528 driver, 0:  user
       user_gpio directly
#define  KEY_GPIO_NUM GPIO_NUM_A25 // "MISO" on demo board
#define  GL5528_VAL_THRE 1000
static void _adc_read_process(void *args);
static uni_bool g_state = false; // false: disable, true: enable
static Result _create_adc_read_thread(void) {
thread_param param;
uni_pthread_t  pid;
uni_memset(&param, 0, sizeof(param));
param.stack_size = STACK_SMALL_SIZE;
param.priority = OS_PRIORITY_LOW;
uni_strncpy(param.task_name, "gpio_adc", sizeof(param.task_name) - 1);
if (0 != uni_pthread_create(&pid, &param,
_adc_read_process, NULL)) {
LOGE(TAG, "create thread failed");
return E_FAILED;
}
uni_pthread_detach(pid);
return E_OK;
}
#if USE_AD_GL5528_DRIVER
static void _adc_read_process(void *args) {
int val = 0;
while (1) {
uni_sleep(1);
val = user_ad_gl5528_get_val();
if (val >= 0) {
LOGT(TAG, "****** gl5528 adc val = %d *****", val);
if (val < GL5528_VAL_THRE) { // means daytime, enable
if (!g_state) {
user_asr_recognize_enable();
g_state  = true;
}
} else {
if (g_state)
{ user_asr_recognize_disable();
user_asr_goto_sleep();
g_state  = false;
}
}
}
}
}
```

```
int user_adc_ctrl(void)
{ user_asr_recognize_disable();
g_state = false;
if (0 != user_ad_gl5528_init(KEY_GPIO_NUM))
LOGE(TAG, { "user_ad_gl5528_init faild.");
return -1;
}
if (E_OK != _create_adc_read_thread())
LOGE(TAG, { "_create_adc_read_thread
          faild.");
return -1;
}
return 0;
}
#else
static void _adc_read_process(void *args) {
int val = 0;
while (1) {
uni_sleep(1);
val = user_gpio_get_value(KEY_GPIO_NUM);
if (val >= 0) {
LOGT(TAG, "****** gl5528 adc val = %d *****", val);
if (val < GL5528_VAL_THRE) { // means daytime, enable
if (!g_state) {
user_asr_recognize_enable();
g_state = true;
}
} else {
if (g_state) {
user_asr_recognize_disable();
user_asr_goto_sleep();
g_state = false;
}
}
}
}
}
int hb_adc_ctrl(void)
{ user_asr_recognize_disable();
g_state = false;
user_gpio_init();
if (0 != user_gpio_set_mode(KEY_GPIO_NUM, GPIO_MODE_ADC)) {
LOGE(TAG, "user_ad_gl5528_init faild.");
return -1;
}
```

```
if (E_OK != _create_adc_read_thread()) {
LOGE(TAG, "_create_adc_read_thread faild.");
return -1;
}
return 0;
}
#endif
```

机芯HBM 离线方案开发指导手册 v1.0.1

hb_gpio_key.c

可通过定义 user/inc/user_config.h 中的宏 USER_RUN_DEMO_SELECT 为 USER_DEMO_KEY_WAKEUP 运行示例程序

```
/*****************************************************************************
        ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
        **
*
* Description : hb_gpio_key.c
* Author : wufangfang@unisound.com
* Date : 2020.04.25
*
*****************************************************************************
        **/
#include "user_gpio.h"
#include "user_digital_keys.h"
#include "user_asr.h"
#define TAG "hb_gpio_key"
```

```
#define USE_DIGITAL_KEYS_DRIVER 1 //1: user user_digital_keys driver, 0:
       user user_gpio directly
// if 0 will send wakeup event in interrupt fuction,
// it will crush because that's a long process,
// so 0 is a demo show how to use GPIO directly only
// donn't use it on your code
#define KEY_GPIO_NUM GPIO_NUM_A25 // "MISO" on demo board
#if USE_DIGITAL_KEYS_DRIVER
void _key_interrupt_cb(GPIO_NUMBER num)
{ user_asr_goto_awakend();
}
int user_key_wakeup(void)
{ user_digital_keys_init(GPIO_INT_POS_EDGE);
return user_digital_keys_register_key(KEY_GPIO_NUM, _key_interrupt_cb);
}
#else
static void _key_interrupt_cb(GPIO_NUMBER num, uni_bool is_high) {
user_asr_goto_awakend();
}
int hb_key_wakeup(void)
{ user_gpio_init();
if (0 != user_gpio_set_mode(KEY_GPIO_NUM, GPIO_MODE_IN))
LOGE(TAG, { "user_gpio_set_mode failed.");
return -1;
}
user_gpio_set_pull_mode(KEY_GPIO_NUM, GPIO_PULL_UP_DOWN);
if (0 != user_gpio_set_interrupt(KEY_GPIO_NUM, GPIO_INT_POS_EDGE,
_key_interrupt_cb)) {
LOGE(TAG, "user_gpio_set_interrupt failed.");
return -1;
}
user_gpio_interrupt_enable();
}
#endif
```

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件

Event 事件接口参考 user/inc/user_event.h 更多...

模块

- **Event 事件说明**
- **Event 事件接口**
- **Event 事件类型定义**

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件说明

Event 事件

Event 事件说明:

1. 通过订阅指定的 Event 事件，来达到监听某种事件发生的目的。事件类型见 USER_EVENT_TYPE

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件接口

Event 事件

函数

int	**user_event_subscribe_event** (**USER_EVENT_TYPE** event, **_user_event_cb** cb) 订阅一个事件 更多...
int	**user_event_clear_observers** (void) 清空事件订阅队列，不再接受任何事件 更多...
_user_event_cb	**user_event_get_observer** (**USER_EVENT_TYPE** event) 查询事件回调函数 更多...

详细描述

函数说明

◆ user_event_clear_observers()

int user_event_clear_observers (void)

返回值

0 操作成功
-1 操作失败

◆ user_event_get_observer()

_user_event_cb user_event_get_observer (USER_EVENT_TYPE event)

参数

event 待查询事件类型

返回

事件回调函数指针

◆ user_event_subscribe_event()

```
int user_event_subscribe_event ( USER_EVENT_TYPE  event,
                                 _user_event_cb   cb
                               )
```

参数

event 待订阅事件类型
cb 事件回调函数指针

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件接口

Event 事件

函数

int	**user_event_subscribe_event** (**USER_EVENT_TYPE** event, **_user_event_cb** cb) 订阅一个事件 更多...
int	**user_event_clear_observers** (void) 清空事件订阅队列，不再接受任何事件 更多...
_user_event_cb	**user_event_get_observer** (**USER_EVENT_TYPE** event) 查询事件回调函数 更多...

详细描述

函数说明

◆ user_event_clear_observers()

int user_event_clear_observers (void)

返回值

0 操作成功
-1 操作失败

◆ user_event_get_observer()

_user_event_cb user_event_get_observer (USER_EVENT_TYPE event)

参数

event 待查询事件类型

返回

事件回调函数指针

◆ user_event_subscribe_event()

```
int user_event_subscribe_event ( USER_EVENT_TYPE  event,
                                 _user_event_cb   cb
                               )
```

参数

event 待订阅事件类型
cb 事件回调函数指针

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件接口

Event 事件

函数

int	**user_event_subscribe_event** (**USER_EVENT_TYPE** event, **_user_event_cb** cb) 订阅一个事件 更多...
int	**user_event_clear_observers** (void) 清空事件订阅队列，不再接受任何事件 更多...
_user_event_cb	**user_event_get_observer** (**USER_EVENT_TYPE** event) 查询事件回调函数 更多...

详细描述

函数说明

◆ user_event_clear_observers()

int user_event_clear_observers (void)

返回值

0 操作成功
-1 操作失败

◆ user_event_get_observer()

_user_event_cb user_event_get_observer (USER_EVENT_TYPE event)

参数

event 待查询事件类型

返回

事件回调函数指针

◆ user_event_subscribe_event()

```
int user_event_subscribe_event ( USER_EVENT_TYPE  event,
                                 _user_event_cb   cb
                               )
```

参数

event 待订阅事件类型
cb 事件回调函数指针

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件接口

Event 事件

函数

int	**user_event_subscribe_event** (**USER_EVENT_TYPE** event, **_user_event_cb** cb) 订阅一个事件 更多...
int	**user_event_clear_observers** (void) 清空事件订阅队列，不再接受任何事件 更多...
_user_event_cb	**user_event_get_observer** (**USER_EVENT_TYPE** event) 查询事件回调函数 更多...

详细描述

函数说明

◆ user_event_clear_observers()

int user_event_clear_observers (void)

返回值

0 操作成功
-1 操作失败

◆ user_event_get_observer()

_user_event_cb user_event_get_observer (USER_EVENT_TYPE event)

参数

event 待查询事件类型

返回

事件回调函数指针

◆ user_event_subscribe_event()

```
int user_event_subscribe_event ( USER_EVENT_TYPE  event,
                                 _user_event_cb   cb
                               )
```

参数

event 待订阅事件类型
cb 事件回调函数指针

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

event_audio_play_t 结构体 参考

对应USER_AUDIO_PLAY_START 事件

`#include <user_event.h>`

成员变量

AUDIO_PLAY_TYPE	type 音频类型
char *	file_name 播放音频文件名

event_audio_play_t 结构体 参考

对应USER_AUDIO_PLAY_START 事件

```
#include <user_event.h>
```

成员变量

AUDIO_PLAY_TYPE	type 音频类型
char *	file_name 播放音频文件名

event_audio_play_t 结构体 参考

对应USER_AUDIO_PLAY_START 事件

```
#include <user_event.h>
```

成员变量

AUDIO_PLAY_TYPE	type 音频类型
char *	file_name 播放音频文件名

event_audio_end_t 结构体 参考

对应 USER_AUDIO_PLAY_END 事件

`#include <user_event.h>`

成员变量

AUDIO_PLAY_TYPE	type 音频类型
uni_bool	is_auto 是否自动播放完毕，true：自动播放完毕，false：主动停止

event_audio_end_t 结构体 参考

对应USER_AUDIO_PLAY_END 事件

```
#include <user_event.h>
```

成员变量

AUDIO_PLAY_TYPE	type 音频类型
uni_bool	is_auto 是否自动播放完毕，true：自动播放完毕，false：主动停止

机芯HBM 离线方案开发指导手册 v1.0.1

event_audio_end_t 结构体 参考

对应USER_AUDIO_PLAY_END 事件

`#include <user_event.h>`

成员变量

AUDIO_PLAY_TYPE	type 音频类型
uni_bool	is_auto 是否自动播放完毕，true：自动播放完毕，false：主动停止

event_custom_setting_t 结构体 参考

对应USER_CUSTOM_SETTING 事件

`#include <user_event.h>`

成员变量

char * **cmd**

命令词意图，对应UDP 平台上用户定义脚本中的 action

char * **word_str**

识别到的命令词

char * **reply_files**

回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_custom_setting_t 结构体 参考

对应USER_CUSTOM_SETTING 事件

`#include <user_event.h>`

成员变量

char * **cmd**

命令词意图，对应UDP 平台上用户定义脚本中的 action

char * **word_str**

识别到的命令词

char * **reply_files**

回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_custom_setting_t 结构体 参考

对应USER_CUSTOM_SETTING 事件

`#include <user_event.h>`

成员变量

char * **cmd**

命令词意图，对应UDP 平台上用户定义脚本中的 action

char * **word_str**

识别到的命令词

char * **reply_files**

回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_custom_setting_t 结构体 参考

对应USER_CUSTOM_SETTING 事件

`#include <user_event.h>`

成员变量

char * **cmd**

命令词意图，对应UDP 平台上用户定义脚本中的 action

char * **word_str**

识别到的命令词

char * **reply_files**

回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

event_goto_sleeping_t 结构体 参考

Event 事件 » Event 事件类型定义

对应USER_GOTO_SLEEPING 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_sleeping_t 结构体 参考

对应USER_GOTO_SLEEPING 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_sleeping_t 结构体 参考

对应USER_GOTO_SLEEPING 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_sleeping_t 结构体 参考

对应USER_GOTO_SLEEPING 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_sleeping_t 结构体 参考

对应USER_GOTO_SLEEPING 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_awakend_t 结构体 参考

对应USER_GOTO_AWAKENED 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_awakend_t 结构体 参考

对应USER_GOTO_AWAKENED 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_awakend_t 结构体 参考

对应USER_GOTO_AWAKENED 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_awakend_t 结构体 参考

对应 USER_GOTO_AWAKENED 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_goto_awakend_t 结构体 参考

对应USER_GOTO_AWAKENED 事件

`#include <user_event.h>`

成员变量

EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str** 识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

机芯HBM 离线方案开发指导手册 v1.0.1

event_volume_setting_t 结构体 参考

对应USER_VOLUME_SETTING 事件

`#include <user_event.h>`

成员变量

int	**value** 当前音量值
int	**min_val** 可设置最小音量值
int	**max_val** 可设置最大音量值
EVENT_TRIGGER	**trigger** 触发来源
char *	**cmd** 命令词意图，对应UDP 平台上用户定义脚本中的 action
char *	**word_str**

	识别到的命令词
char *	**reply_files** 回复语列表[1, 2, 3]，对应 UDP 平台上用户定义脚本中的回复语列表，在 pcm_map.txt 中可以找到对应关系

user_event_context_t 联合体 参考

事件内容，共用体，根据事件类型具体处理

`#include <user_event.h>`

成员变量

event_audio_play_t	audio_play 对应USER_AUDIO_PLAY_START 事件
event_audio_end_t	audio_end 对应USER_AUDIO_PLAY_END 事件
event_custom_setting_t	custom_setting 对应USER_CUSTOM_SETTING 事件
event_volume_setting_t	voluem_setting 对应USER_VOLUME_SETTING 事件
event_goto_sleeping_t	goto_sleeping 对应USER_GOTO_SLEEPING 事件
event_goto_awakend_t	goto_awakend

对应USER_GOTO_AWAKENED 事件

机芯HBM 离线方案开发指导手册 v1.0.1

user_event_context_t 联合体 参考

事件内容，共用体，根据事件类型具体处理

`#include <user_event.h>`

成员变量

event_audio_play_t	audio_play 对应USER_AUDIO_PLAY_START 事件
event_audio_end_t	audio_end 对应USER_AUDIO_PLAY_END 事件
event_custom_setting_t	custom_setting 对应USER_CUSTOM_SETTING 事件
event_volume_setting_t	voluem_setting 对应USER_VOLUME_SETTING 事件
event_goto_sleeping_t	goto_sleeping 对应USER_GOTO_SLEEPING 事件
event_goto_awakend_t	goto_awakend

对应USER_GOTO_AWAKENED 事件

机芯HBM 离线方案开发指导手册 v1.0.1

user_event_context_t 联合体 参考

事件内容，共用体，根据事件类型具体处理

`#include <user_event.h>`

成员变量

event_audio_play_t	audio_play 对应USER_AUDIO_PLAY_START 事件
event_audio_end_t	audio_end 对应USER_AUDIO_PLAY_END 事件
event_custom_setting_t	custom_setting 对应USER_CUSTOM_SETTING 事件
event_volume_setting_t	voluem_setting 对应USER_VOLUME_SETTING 事件
event_goto_sleeping_t	goto_sleeping 对应USER_GOTO_SLEEPING 事件
event_goto_awakend_t	goto_awakend

对应USER_GOTO_AWAKENED 事件

机芯HBM 离线方案开发指导手册 v1.0.1

user_event_context_t 联合体 参考

事件内容，共用体，根据事件类型具体处理

`#include <user_event.h>`

成员变量

event_audio_play_t	audio_play 对应USER_AUDIO_PLAY_START 事件
event_audio_end_t	audio_end 对应USER_AUDIO_PLAY_END 事件
event_custom_setting_t	custom_setting 对应USER_CUSTOM_SETTING 事件
event_volume_setting_t	voluem_setting 对应USER_VOLUME_SETTING 事件
event_goto_sleeping_t	goto_sleeping 对应USER_GOTO_SLEEPING 事件
event_goto_awakend_t	goto_awakend

对应USER_GOTO_AWAKENED 事件

机芯HBM 离线方案开发指导手册 v1.0.1

user_event_context_t 联合体 参考

事件内容，共用体，根据事件类型具体处理

`#include <user_event.h>`

成员变量

event_audio_play_t	audio_play 对应USER_AUDIO_PLAY_START 事件
event_audio_end_t	audio_end 对应USER_AUDIO_PLAY_END 事件
event_custom_setting_t	custom_setting 对应USER_CUSTOM_SETTING 事件
event_volume_setting_t	voluem_setting 对应USER_VOLUME_SETTING 事件
event_goto_sleeping_t	goto_sleeping 对应USER_GOTO_SLEEPING 事件
event_goto_awakend_t	goto_awakend

对应USER_GOTO_AWAKENED 事件

机芯HBM 离线方案开发指导手册 v1.0.1

user_event_context_t 联合体 参考

事件内容，共用体，根据事件类型具体处理

`#include <user_event.h>`

成员变量

event_audio_play_t	audio_play 对应USER_AUDIO_PLAY_START 事件
event_audio_end_t	audio_end 对应USER_AUDIO_PLAY_END 事件
event_custom_setting_t	custom_setting 对应USER_CUSTOM_SETTING 事件
event_volume_setting_t	voluem_setting 对应USER_VOLUME_SETTING 事件
event_goto_sleeping_t	goto_sleeping 对应 USER_GOTO_SLEEPING 事件
event_goto_awakend_t	goto_awakend

对应USER_GOTO_AWAKENED 事件

机芯HBM 离线方案开发指导手册 v1.0.1

user_event_context_t 联合体 参考

事件内容，共用体，根据事件类型具体处理

`#include <user_event.h>`

成员变量

event_audio_play_t	audio_play 对应USER_AUDIO_PLAY_START 事件
event_audio_end_t	audio_end 对应USER_AUDIO_PLAY_END 事件
event_custom_setting_t	custom_setting 对应USER_CUSTOM_SETTING 事件
event_volume_setting_t	voluem_setting 对应USER_VOLUME_SETTING 事件
event_goto_sleeping_t	goto_sleeping 对应USER_GOTO_SLEEPING 事件
event_goto_awakend_t	goto_awakend

对应USER_GOTO_AWAKENED 事件

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应 USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

Event 事件类型定义

Event 事件

结构 :

struct	**event_audio_play_t** 对应USER_AUDIO_PLAY_START 事件 更多...
struct	**event_audio_end_t** 对应USER_AUDIO_PLAY_END 事件 更多...
struct	**event_custom_setting_t** 对应USER_CUSTOM_SETTING 事件 更多...
struct	**event_goto_sleeping_t** 对应USER_GOTO_SLEEPING 事件 更多...
struct	**event_goto_awakend_t** 对应USER_GOTO_AWAKENED 事件 更多...
struct	**event_volume_setting_t** 对应USER_VOLUME_SETTING 事件 更多...
union	**user_event_context_t**

	事件内容，共用体，根据事件类型具体处理 更多...

类型定义

typedef void(*	**_user_event_cb**) (**USER_EVENT_TYPE** event, **user_event_context_t** *context) 事件回调函数，当用户订阅事件发生时会被调用 更多...

枚举

enum	**USER_EVENT_TYPE** {
	USER_INVALID_EVENT = 0, **USER_AUDIO_PLAY_START**, **USER_AUDIO_PLAY_END**, **USER_CUSTOM_SETTING**, **USER_VOLUME_SETTING**, **USER_GOTO_SLEEPING**, **USER_GOTO_AWAKENED**, **USER_EVENT_MAX**
	}
	用户可订阅的事件类型 更多...

enum	**AUDIO_PLAY_TYPE** { **AUDIO_PLAY_REPLY** = 0, **AUDIO_PLAY_MUSIC**, **AUDIO_PLAY_SKILL** } 音频播放类型枚举（目前只支持 AUDIO_PLAY_REPLY） 更多...

enum	**EVENT_TRIGGER** { **EVENT_TRIGGER_ASR** = 0, **EVENT_TRIGGER_AUTO**, **EVENT_TRIGGER_USER** } 事件触发来源 更多...

详细描述

类型定义说明

◆ _user_event_cb

typedef void(* _user_event_cb) (USER_EVENT_TYPE event, user_event_context_t *context)

参数

event 事件类型

context 事件信息

枚举类型说明

◆ AUDIO_PLAY_TYPE

enum AUDIO_PLAY_TYPE

枚举值	
AUDIO_PLAY_REPLY	回复语播报
AUDIO_PLAY_MUSIC	音乐播放
AUDIO_PLAY_SKILL	技能 TTS 回复播报

◆ EVENT_TRIGGER

enum EVENT_TRIGGER

枚举值	
EVENT_TRIGGER_ASR	语音识别指令触发
EVENT_TRIGGER_AUTO	自动状态切换（比如超时等）
EVENT_TRIGGER_USER	用户接口主动触发

◆ USER_EVENT_TYPE

enum **USER_EVENT_TYPE**

枚举值	
USER_INVALID_EVENT	错误的事件类型
USER_AUDIO_PLAY_START	音频开始播放时发送
USER_AUDIO_PLAY_END	音频播放完毕或被停止时发送
USER_CUSTOM_SETTING	识别到客户自定义的识别词时发送
USER_VOLUME_SETTING	识别到音量调节指令时发送
USER_GOTO_SLEEPING	进入待唤醒状态时发送
USER_GOTO_AWAKENED	进入识别状态（已唤醒）时发送

机芯HBM 离线方案开发指导手册 v1.0.1

PWM

PWM 接口参考 user/inc/user_pwm.h 更多...

模块

- PWM 说明
- PWM 相关接口
- PWM 类型定义
- PWM 示例

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 说明

PWM

PWM 说明:

1. 机芯HBM平台使用 TIMER3,TIMER4 为 PWM timer
2. PWM 频率与系统时钟的分频比，取值范围[1, PWM_MAX_FREQ_DIV_VALUE]，PWM_MAX_FREQ_DIV_VALUE 为（65535 << 15）
3. 默认下降沿计数，中断模式

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
    uni_u32         hz,
    uni_bool        is_high_duty
)
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0	操作成功
-1	操作失败

◆ user_pwm_pause()

int user_pwm_pause (**USER_PWM_NUM** num)

参数

num	PWM 管脚号

返回值

0	操作成功
-1	操作失败

◆ user_pwm_resume()

int user_pwm_resume (**USER_PWM_NUM** num)

参数

num	PWM 管脚号

返回值

0	操作成功
-1	操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
                uni_u32         hz,
                uni_bool        is_high_duty
            )
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0 操作成功
-1 操作失败

◆ user_pwm_pause()

```
int user_pwm_pause ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_resume()

```
int user_pwm_resume ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
                uni_u32         hz,
                uni_bool        is_high_duty
            )
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0 操作成功
-1 操作失败

◆ user_pwm_pause()

```
int user_pwm_pause ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_resume()

```
int user_pwm_resume ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
                uni_u32         hz,
                uni_bool        is_high_duty
            )
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0 操作成功
-1 操作失败

◆ user_pwm_pause()

```
int user_pwm_pause ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_resume()

```
int user_pwm_resume ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
                uni_u32         hz,
                uni_bool        is_high_duty
            )
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0 操作成功
-1 操作失败

◆ user_pwm_pause()

```
int user_pwm_pause ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_resume()

```
int user_pwm_resume ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
                uni_u32         hz,
                uni_bool        is_high_duty
            )
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0 操作成功
-1 操作失败

◆ user_pwm_pause()

```
int user_pwm_pause ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_resume()

```
int user_pwm_resume ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
                uni_u32         hz,
                uni_bool        is_high_duty
            )
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0 操作成功
-1 操作失败

◆ user_pwm_pause()

```
int user_pwm_pause ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_resume()

```
int user_pwm_resume ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 相关接口

PWM

函数

int	**user_pwm_init** (**USER_PWM_NUM** num, uni_u32 hz, uni_bool is_high_duty)
	PWM 初始化 更多...
int	**user_pwm_final** (**USER_PWM_NUM** num)
	PWM 反初始化 更多...
int	**user_pwm_start** (**USER_PWM_NUM** num, uni_u8 duty)
	开使PWM 输出 更多...
int	**user_pwm_stop** (**USER_PWM_NUM** num)
	停止PWM 输出 更多...
int	**user_pwm_pause** (**USER_PWM_NUM** num)
	暂停PWM 输出 更多...
int	**user_pwm_resume** (**USER_PWM_NUM** num)
	恢复PWM 输出 更多...
int	**user_pwm_change_duty** (**USER_PWM_NUM** num, uni_u8 duty)

PWM 占空比切换 更多...

详细描述

函数说明

◆ user_pwm_change_duty()

```
int user_pwm_change_duty ( USER_PWM_NUM num,
                           uni_u8       duty
                         )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_final()

```
int user_pwm_final ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_init()

```
int user_pwm_init ( USER_PWM_NUM num,
```

```
                uni_u32         hz,
                uni_bool        is_high_duty
            )
```

参数

num	PWM 管脚号
hz	频率
is_high_duty	*TRUE* :占空比用高电平持续时间计算; *FALSE* :占空比用低电平持续时间计算

返回值

0 操作成功
-1 操作失败

◆ user_pwm_pause()

```
int user_pwm_pause ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_resume()

```
int user_pwm_resume ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

◆ user_pwm_start()

```
int user_pwm_start ( USER_PWM_NUM  num,
                     uni_u8        duty
                   )
```

参数

num PWM 管脚号
duty 占空比

返回值

0 操作成功
-1 操作失败

◆ user_pwm_stop()

```
int user_pwm_stop ( USER_PWM_NUM num )
```

参数

num PWM 管脚号

返回值

0 操作成功
-1 操作失败

机芯HBM 离线方案开发指导手册 v1.0.1

枚举

PWM 类型定义

PWM

枚举

```
enum    USER_PWM_NUM {
          PWM_NUM_1_A27 = 0, PWM_NUM_1_B0, PWM_NUM_1_B2,
        PWM_NUM_2_A28,
          PWM_NUM_2_B1, PWM_NUM_2_B3, PWM_NUM_MAX
        }
```

详细描述

枚举类型说明

◆ USER_PWM_NUM

enum USER_PWM_NUM

PWM 管脚号

枚举值	
PWM_NUM_1_B2	PWM 1 only 1 pin work a time
PWM_NUM_2_A28	used for PA enable, don't use it on HB-M demo board

PWM_NUM_2_B3	PWM 2 only 1 pin work a time

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 类型定义

PWM

枚举

```
enum    USER_PWM_NUM {
          PWM_NUM_1_A27 = 0, PWM_NUM_1_B0, PWM_NUM_1_B2,
        PWM_NUM_2_A28,
          PWM_NUM_2_B1, PWM_NUM_2_B3, PWM_NUM_MAX
        }
```

详细描述

枚举类型说明

◆ USER_PWM_NUM

enum USER_PWM_NUM

PWM 管脚号

枚举值	
PWM_NUM_1_B2	PWM 1 only 1 pin work a time
PWM_NUM_2_A28	used for PA enable, don't use it on HB-M demo board

PWM_NUM_2_B3	PWM 2 only 1 pin work a time

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 类型定义

PWM

枚举

```
enum    USER_PWM_NUM {
          PWM_NUM_1_A27 = 0, PWM_NUM_1_B0, PWM_NUM_1_B2,
        PWM_NUM_2_A28,
          PWM_NUM_2_B1, PWM_NUM_2_B3, PWM_NUM_MAX
        }
```

详细描述

枚举类型说明

◆ USER_PWM_NUM

enum USER_PWM_NUM

PWM 管脚号

枚举值	
PWM_NUM_1_B2	PWM 1 only 1 pin work a time
PWM_NUM_2_A28	used for PA enable, don't use it on HB-M demo board

PWM_NUM_2_B3	PWM 2 only 1 pin work a time

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 类型定义

PWM

枚举

```
enum    USER_PWM_NUM {
          PWM_NUM_1_A27 = 0, PWM_NUM_1_B0, PWM_NUM_1_B2,
        PWM_NUM_2_A28,
          PWM_NUM_2_B1, PWM_NUM_2_B3, PWM_NUM_MAX
        }
```

详细描述

枚举类型说明

◆ USER_PWM_NUM

enum USER_PWM_NUM

PWM 管脚号

枚举值	
PWM_NUM_1_B2	PWM 1 only 1 pin work a time
PWM_NUM_2_A28	used for PA enable, don't use it on HB-M demo board

PWM_NUM_2_B3	PWM 2 only 1 pin work a time

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 类型定义

PWM

枚举

```
enum    USER_PWM_NUM {
          PWM_NUM_1_A27 = 0, PWM_NUM_1_B0, PWM_NUM_1_B2,
        PWM_NUM_2_A28,
          PWM_NUM_2_B1, PWM_NUM_2_B3, PWM_NUM_MAX
        }
```

详细描述

枚举类型说明

◆ USER_PWM_NUM

enum USER_PWM_NUM

PWM 管脚号

枚举值	
PWM_NUM_1_B2	PWM 1 only 1 pin work a time
PWM_NUM_2_A28	used for PA enable, don't use it on HB-M demo board

PWM_NUM_2_B3	PWM 2 only 1 pin work a time

机芯HBM 离线方案开发指导手册 v1.0.1

PWM 示例

PWM

模块

hb_pwm_led.c

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

hb_pwm_led.c

可通过定义 user/inc/user_config.h 中的宏 USER_RUN_DEMO_SELECT 为 USER_DEMO_PWM_LED 运行示例程序

```
/*****************************************************************************
 ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
 **
*
* Description : hb_pwm_led.c
* Author : wufangfang@unisound.com
* Date : 2020.04.26
*
*****************************************************************************
 **/
#include "user_gpio.h"
#include "user_pwm_led.h"
#define TAG "hb_pwm_led"
#define PWM_LED_GPIO_NUM PWM_NUM_1_A27 // "MOSI" on HB-M demo board
```

```
static void _pwm_led_process(void *args) {
LED_BRIGHT_LEVEL level = BRIGHT_LEVEL_0;
while (1) {
user_pwm_led_set_brightness(PWM_LED_GPIO_NUM, level);
level += 1;
if (level >= BRIGHT_LEVEL_MAX) {
level = BRIGHT_LEVEL_0;
}
//uni_sleep(1);
}
}
static Result _create_pwm_led_thread(void) {
thread_param param;
uni_pthread_t  pid;
uni_memset(&param, 0, sizeof(param));
param.stack_size = STACK_SMALL_SIZE;
param.priority = OS_PRIORITY_LOW;
uni_strncpy(param.task_name, "pwm_led", sizeof(param.task_name) - 1);
if (0 != uni_pthread_create(&pid, &param,
_pwm_led_process, NULL)) {
LOGE(TAG, "create thread failed");
return E_FAILED;
}
uni_pthread_detach(pid);
return E_OK;
}
int hb_pwm_led(void) {
if (0 != user_pwm_led_init(PWM_LED_GPIO_NUM))
LOGE(TAG, { "user_pwm_led_init failed.");
return -1;
}
if (E_OK != _create_pwm_led_thread())
LOGE(TAG, { "_create_pwm_led_thread
          failed.");
user_pwm_led_final(PWM_LED_GPIO_NUM);
return -1;
}
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

模块

方案示例

示例代码位于目录user/src/examples/下 更多...

模块

- **智能电风扇**
- **智能管家**

详细描述

机芯HBM 离线方案开发指导手册 v1.0.1

智能电风扇

方案示例

可通过定义 user/inc/user_config.h 中的宏 **USER_RUN_DEMO_SELECT** 为 **USER_DEMO_SMART_AC** 运行示例程序

```
/*****************************************************************************
        ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
        **
*
* Description : hb_smart_ac.c
* Author : wufangfang@unisound.com
* Date : 2020.04.03
*
*****************************************************************************
        **/
#include "unione.h"
#include "ac_device_simulator.h"
#include "user_asr.h"
#include "user_player.h"
```

```
#define TAG "smart_ac"
static char g_power_off_reply[32] = {0};
static int _ac_power_replay(event_custom_setting_t *setting)
eAcSate { state = eACSTATE_UNKONWN;
eAcWindMode mode = eACMODE_UNKONWN;
if (0 == uni_strcmp(setting->cmd, "ac_power_on")) {
state = AcDeviceSetState(eACSTATE_ON);
} else if (0 == uni_strcmp(setting->cmd, "ac_power_off"))
state = { AcDeviceSetState(eACSTATE_OFF);
} else {
LOGE(TAG, "unknown cmd: %s", setting->cmd);
return -1;
}
LOGT(TAG, "AC power state: %d", state);
if (eACSTATE_OFF == state) {
uni_strncpy(g_power_off_reply,                    setting->reply_files,
       sizeof(g_power_off_reply) - 1);
return user_asr_goto_sleep();
} else if (eACSTATE_ON == state)
mode = { AcDeviceGetWindMode();
switch (mode) {
case eACMODE_NATURE:
return user_player_reply_list_num(setting->reply_files, 0);
case eACMODE_NORMAL:
return user_player_reply_list_num(setting->reply_files, 1);
case eACMODE_SLEEP:
return user_player_reply_list_num(setting->reply_files, 2);
default:
LOGE(TAG, "Unknown mode: %d", mode);
break;
}
} else {
LOGE(TAG, "invalid AC state: %d", state);
}
return -1;
}
static int _ac_off_replay(void) {
#ifdef DEFAULT_PCM_WAKEUP
return user_player_reply_list_num(DEFAULT_PCM_WAKEUP, 0);
#else
return user_player_reply_list_random(HI_AC_OFF);
#endif
}
static int _ac_speed_replay(event_custom_setting_t *setting) {
```

```
eAcSate state = AcDeviceGetState();
eAcWindSpeed speed = eAcWindSpeed_UNKONWN;
if (eACSTATE_ON != state) {
return _ac_off_replay();
}
if (0 == uni_strcmp(setting->cmd, "ac_speed_1")) {
speed = AcDeviceSetWindSpeed(eAcWindSpeed_1);
} else if (0 == uni_strcmp(setting->cmd, "ac_speed_2"))
speed = { AcDeviceSetWindSpeed(eAcWindSpeed_2);
} else if (0 == uni_strcmp(setting->cmd, "ac_speed_3"))
speed = { AcDeviceSetWindSpeed(eAcWindSpeed_3);
} else if (0 == uni_strcmp(setting->cmd, "ac_speed_inc"))
speed = { AcDeviceGetWindSpeed();
if (eAcWindSpeed_3 == speed) {
return user_player_reply_list_num(setting->reply_files, 1);
}
speed = AcDeviceWindSpeedInc();
return user_player_reply_list_num(setting->reply_files, 0);
} else if (0 == uni_strcmp(setting->cmd, "ac_speed_dec")) {
speed = AcDeviceGetWindSpeed();
if (eAcWindSpeed_1 == speed) {
return user_player_reply_list_num(setting->reply_files, 1);
}
speed = AcDeviceWindSpeedDec();
return user_player_reply_list_num(setting->reply_files, 0);
} else {
LOGE(TAG, "unknown cmd: %s", setting->cmd);
return -1;
}
LOGT(TAG, "AC speed state: %d", speed);
return user_player_reply_list_random(setting->reply_files);
}
static int _ac_mode_replay(event_custom_setting_t *setting)
eAcSate { state = AcDeviceGetState();
eAcWindMode mode = eACMODE_UNKONWN;
if (eACSTATE_ON != state) {
return _ac_off_replay();
}
if (0 == uni_strcmp(setting->cmd, "ac_mode_nat")) {
mode = AcDeviceSetWindMode(eACMODE_NATURE);
} else if (0 == uni_strcmp(setting->cmd, "ac_mode_nor"))
mode = { AcDeviceSetWindMode(eACMODE_NORMAL);
} else if (0 == uni_strcmp(setting->cmd, "ac_mode_sleep"))
mode = { AcDeviceSetWindMode(eACMODE_SLEEP);
```

```
} else {
LOGE(TAG, "unknown cmd: %s", setting->cmd);
return -1;
}
LOGT(TAG, "AC mode state: %d", mode);
return user_player_reply_list_random(setting->reply_files);
}
static int _timer_change_replay(eAcTimer timer, const char *file_list)
switch { (timer) {
case eACTIMER_1HR:
return user_player_reply_list_num(file_list, 0);
case eACTIMER_2HR:
return user_player_reply_list_num(file_list, 1);
case eACTIMER_4HR:
return user_player_reply_list_num(file_list, 2);
case eACTIMER_8HR:
return user_player_reply_list_num(file_list, 3);
default:
break;
}
return -1;
}
static int _ac_timing_replay(event_custom_setting_t *setting)
eAcSate { state = AcDeviceGetState();
eAcTimer timer = eACTIMER_UNKONWN;
if (eACSTATE_ON != state) {
return _ac_off_replay();
}
if (0 == uni_strcmp(setting->cmd, "ac_timing_1hr")) {
timer = AcDeviceSetTimer(eACTIMER_1HR);
} else if (0 == uni_strcmp(setting->cmd, "ac_timing_2hr"))
timer = { AcDeviceSetTimer(eACTIMER_2HR);
} else if (0 == uni_strcmp(setting->cmd, "ac_timing_4hr"))
timer = { AcDeviceSetTimer(eACTIMER_4HR);
} else if (0 == uni_strcmp(setting->cmd, "ac_timing_8hr"))
timer = { AcDeviceSetTimer(eACTIMER_8HR);
} else if (0 == uni_strcmp(setting->cmd, "ac_timing_cancel"))
timer = { AcDeviceSetTimer(eACTIMER_NONE);
} else if (0 == uni_strcmp(setting->cmd, "ac_timing_inc"))
timer = { AcDeviceTimerInc();
return _timer_change_replay(timer,  setting->reply_files);
} else if (0 == uni_strcmp(setting->cmd, "ac_timing_dec")) {
timer = AcDeviceTimerDec();
return _timer_change_replay(timer, setting->reply_files);
```

```
} else {
LOGE(TAG, "unknown cmd: %s", setting->cmd);
return -1;
}
LOGT(TAG, "AC timer state: %d", timer);
return user_player_reply_list_random(setting->reply_files);
}
static int _ac_shake_replay(event_custom_setting_t *setting)
eAcSate { state = AcDeviceGetState();
eAcShake shake = eACSHAKE_UNKONWN;
if (eACSTATE_ON != state) {
return _ac_off_replay();
}
if (0 == uni_strcmp(setting->cmd, "ac_shake_on")) {
shake = AcDeviceSetShakeState(eACSHAKE_ON);
} else if (0 == uni_strcmp(setting->cmd, "ac_shake_off"))
shake = { AcDeviceSetShakeState(eACSHAKE_OFF);
} else {
LOGE(TAG, "unknown cmd: %s", setting->cmd);
return -1;
}
LOGT(TAG, "AC shake state: %d", shake);
return user_player_reply_list_random(setting->reply_files);
}
static void _custom_setting_cb(USER_EVENT_TYPE event,
user_event_context_t *context) {
event_custom_setting_t *setting = NULL;
if (context) {
setting = &context->custom_setting;
if (NULL != uni_strstr(setting->cmd, "ac_power")) {
_ac_power_replay(setting);
} else if (NULL != uni_strstr(setting->cmd, "ac_speed")) {
_ac_speed_replay(setting);
} else if (NULL != uni_strstr(setting->cmd, "ac_mode")) {
_ac_mode_replay(setting);
} else if (NULL != uni_strstr(setting->cmd, "ac_timing")) {
_ac_timing_replay(setting);
} else if (NULL != uni_strstr(setting->cmd, "ac_shake")) {
_ac_shake_replay(setting);
} else {
/* don't reply if other setting command */
}
}
}
```

```
static void _goto_awakend_cb(USER_EVENT_TYPE event,
user_event_context_t  *context) {
event_goto_awakend_t  *awakend = NULL;
if (context) {
awakend = &context->goto_awakend;
if (EVENT_TRIGGER_ASR == awakend->trigger) {
LOGT(TAG, "ASR command: %s -> %s -> %s", awakend->cmd, awakend->word_str,
       awakend->reply_files);
} else {
LOGT(TAG,  "External command: %s", awakend->reply_files);
}
if (awakend->reply_files)
{ if(eACSTATE_ON == AcDeviceGetState())
{
user_player_reply_list_num(awakend->reply_files, 1);
} else {
user_player_reply_list_num(awakend->reply_files, 0);
}
}
}
}
static void _goto_sleeping_cb (USER_EVENT_TYPE event,
user_event_context_t *context) {
event_goto_sleeping_t  *sleeping = NULL;
if (context) {
sleeping = &context->goto_sleeping;
if (EVENT_TRIGGER_ASR == sleeping->trigger) {
LOGT(TAG, "ASR command: %s -> %s -> %s", sleeping->cmd, sleeping->word_str,
sleeping->reply_files);
} else {
LOGT(TAG,  "External command: %s", sleeping->reply_files);
}
if (sleeping->reply_files) {
if(EVENT_TRIGGER_USER == sleeping->trigger) {
user_player_reply_list_random(g_power_off_reply);
} else {
user_player_reply_list_random(sleeping->reply_files);
}
}
}
}
static void _register_event_callback(void) {
user_event_subscribe_event(USER_CUSTOM_SETTING, _custom_setting_cb);
user_event_subscribe_event(USER_GOTO_AWAKENED, _goto_awakend_cb);
user_event_subscribe_event(USER_GOTO_SLEEPING, _goto_sleeping_cb);
```

```
}
int hb_smart_ac(void) {
AcDeviceInit();
_register_event_callback();
return 0;
}
```

机芯HBM 离线方案开发指导手册 v1.0.1

智能管家

方案示例

可通过定义 user/inc/user_config.h 中的宏 **USER_RUN_DEMO_SELECT** 为 **USER_DEMO_HOUSEKEEPER** 运行示例程序

```
/*****************************************************************************
        ***
* Copyright (C) 2017-2017 Unisound
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*****************************************************************************
        **
*
* Description : hb_housekeeper.c
* Author : wufangfang@unisound.com
* Date : 2020.04.13
*
*****************************************************************************
        **/
#include "unione.h"
#include "user_asr.h"
#include "user_player.h"
#include "user_uart_hb1688.h"
```

```
#include "user_flash.h"
#define TAG "hb_housekeeoer"
#define KEY_WORD_LEN_MAX 64
#define REPLAY_MATCH_OK "115" // 匹配成功
#define REPLAY_MATCH_FAIL "116" // 匹配失败
#define REPLAY_LEARN_OK "118" // 学习成功
#define REPLAY_LEARN_FAIL "119" // 学习失败
#define REPLAY_CODE_OK "186" // 对码成功
#define REPLAY_CODE_FAIL "187" // 对码失败
#define BLOB_KEY_IS_MATCH "AC_IS_MATCH" // save in flash key, value means
       0: nerver match AC RC, 1: have matched
#define BLOB_TYPE_IS_MATCH uni_u32
#define BLOB_KEY_AC_TYPE "AC_TYPE" // save in flash key, value means number
       get AC RC code from code lib
#define  BLOB_TYPE_AC_TYPE int
typedef  enum {
HB1688_LEARN_NONE  = 0,
HB1688_AC_MATCH,
HB1688_LEARN_ARC,
HB1688_LEARN_URC,
HB1688_LEARN_CODE,
HB1688_LEARN_MAX
}HB1688_LEARN_MODE;
typedef struct {
char key_word[KEY_WORD_LEN_MAX];
uni_u8 code[HB1688_LRC_LEN];
uni_u8 len;
}hb1688_lrc_code_t;
typedef struct
uni_bool  { is_lear
          n;
HB1688_LEARN_MODE  learn_mode;
hb1688_lrc_code_t  lrc_code;
}hb_housekeeper_context_t;
typedef struct {
const char *cmd_str;
uni_u8 ctrl_code;
uni_u8 cmd_num_h;
uni_u8 cmd_num_l;
}mcu_cmd_map_t;
static uni_bool g_speaker_mute = false;
static hb_housekeeper_context_t g_hk_context = {false, HB1688_LEARN_NONE,
       {{0}, {0}, 0}};
static mcu_cmd_map_t g_mcu_cmd_map[] = {
{ "learnMode", 0x03, 0x00, 0x05 },
```

```
{ "learnModeExit", 0x03, 0x00, 0x06 },
{ "userGoOut", 0x01, 0x00, 0x07 },
{ "userGoHome", 0x01, 0x00, 0x08 },
{ "userGoToBed", 0x01, 0x00, 0x09 },
{ "userGetUp", 0x01, 0x00, 0x0A },
{ "rcAcMatch", 0x03, 0x00, 0x0B },
{ "acTurnOn", 0x01, 0x00, 0x0C },
{ "acTurnOff", 0x01, 0x00, 0x0D },
{ "acModeCool", 0x01, 0x00, 0x0E },
{ "acModeHeat", 0x01, 0x00, 0x0F },
{ "acModeBlower", 0x01, 0x00, 0x10 },
{ "acModeMoisture", 0x01, 0x00, 0x11 },
{ "acFanSpeedHigh", 0x01, 0x00, 0x12 },
{ "acFanSpeedMid", 0x01, 0x00, 0x13 },
{ "acFanSpeedLow", 0x01, 0x00, 0x14 },
{ "acFanSpeedAuto", 0x01, 0x00, 0x15 },
{ "acFanSweptOn", 0x01, 0x00, 0x16 },
{ "acFanSweptOff", 0x01, 0x00, 0x17 },
{ "acDegrees16", 0x01, 0x00, 0x18 },
{ "acDegrees17", 0x01, 0x00, 0x19 },
{ "acDegrees18", 0x01, 0x00, 0x1A },
{ "acDegrees19", 0x01, 0x00, 0x1B },
{ "acDegrees20", 0x01, 0x00, 0x1C },
{ "acDegrees21", 0x01, 0x00, 0x1D },
{ "acDegrees22", 0x01, 0x00, 0x1E },
{ "acDegrees23", 0x01, 0x00, 0x1F },
{ "acDegrees24", 0x01, 0x00, 0x20 },
{ "acDegrees25", 0x01, 0x00, 0x21 },
{ "acDegrees26", 0x01, 0x00, 0x22 },
{ "acDegrees27", 0x01, 0x00, 0x23 },
{ "acDegrees28", 0x01, 0x00, 0x24 },
{ "acDegrees29", 0x01, 0x00, 0x25 },
{ "acDegrees30", 0x01, 0x00, 0x26 },
{ "rcLearnKey", 0x03, 0x00, 0x27 },
{ "tvTurnOn", 0x01, 0x00, 0x28 },
{ "tvTurnOff", 0x01, 0x00, 0x29 },
{ "tvChannelPre", 0x01, 0x00, 0x2A },
{ "tvChannelNext", 0x01, 0x00, 0x2B },
{ "tvVolumeUp", 0x01, 0x00, 0x2C },
{ "tvVolumeDown", 0x01, 0x00, 0x2D },
{ "tvSpeakerMute", 0x01, 0x00, 0x2E },
{ "tvSpeakerUnmute", 0x01, 0x00, 0x2F },
{ "tvSettings", 0x01, 0x00, 0x30 },
{ "tvMenu", 0x01, 0x00, 0x31 },
```

```
{ "tvEnter", 0x01, 0x00, 0x32 },
{ "tvReturn", 0x01, 0x00, 0x33 },
{ "tvSwitchLeft", 0x01, 0x00, 0x34 },
{ "tvSwitchRight", 0x01, 0x00, 0x35 },
{ "tvSwitchUp", 0x01, 0x00, 0x36 },
{ "tvSwitchDown", 0x01, 0x00, 0x37 },
{ "tvSwicthSingle", 0x01, 0x00, 0x38 },
{ "stbTurnOn", 0x01, 0x00, 0x39 },
{ "stbTurnOff", 0x01, 0x00, 0x3A },
{ "lampLivingOn", 0x01, 0x00, 0x3B },
{ "lampLivingOff", 0x01, 0x00, 0x3C },
{ "downLightOn", 0x01, 0x00, 0x3D },
{ "downLightOff", 0x01, 0x00, 0x3E },
{ "tapeLightOn", 0x01, 0x00, 0x3F },
{ "tapeLightOff", 0x01, 0x00, 0x40 },
{ "lampDiningOn", 0x01, 0x00, 0x41 },
{ "lampDiningOff", 0x01, 0x00, 0x42 },
{ "lampBedroomOn", 0x01, 0x00, 0x43 },
{ "lampBedroomOff", 0x01, 0x00, 0x44 },
{ "lampBedsideOn", 0x01, 0x00, 0x45 },
{ "lampBedsideOff", 0x01, 0x00, 0x46 },
{ "lampNightOn", 0x01, 0x00, 0x47 },
{ "lampNightOff", 0x01, 0x00, 0x48 },
{ "lampAllOn", 0x01, 0x00, 0x49 },
{ "lampAllOff", 0x01, 0x00, 0x4A },
{ "curtainsOpen", 0x01, 0x00, 0x4B },
{ "curtainsClose", 0x01, 0x00, 0x4C },
{ "curtainsCancel", 0x01, 0x00, 0x4D },
{ "codeLampLiving", 0x01, 0x00, 0x4E },
{ "codeDownLight", 0x01, 0x00, 0x4F },
{ "codeTapeLight", 0x01, 0x00, 0x50 },
{ "codeLampDining", 0x01, 0x00, 0x51 },
{ "codeLampBedroom", 0x01, 0x00, 0x52 },
{ "codeLampBedside", 0x01, 0x00, 0x53 },
{ "codeLampNight", 0x01, 0x00, 0x54 },
{ "codeCurtains", 0x01, 0x00, 0x55 }
};
static int _send_cmd_uart_protocol(const char *cmd)
int i = { 0;
for (i = 0; i < sizeof(g_mcu_cmd_map)/sizeof(mcu_cmd_map_t); i++) {
if (0 == uni_strcmp(cmd, g_mcu_cmd_map[i].cmd_str))
/* TODO: { send to MCU by UART */
LOGT(TAG, "UART protcol: %s : 0x%02x 0x%02x [0x%02x]",
g_mcu_cmd_map[i].cmd_str, g_mcu_cmd_map[i].ctrl_code,
```

```
g_mcu_cmd_map[i].cmd_num_h, g_mcu_cmd_map[i].cmd_num_l);
/* always return 0 if ctrl_code is 0x03 - NACK */
if (0x03 == g_mcu_cmd_map[i].ctrl_code) {
return 0;
}
break;
}
}
return 0;
}
static void _hb1688_lrc_learn_recv(uni_bool is_ok, uni_u8 *buf, uni_u8 len)
      {
BLOB_TYPE_IS_MATCH  is_ac_match = 0;
BLOB_TYPE_AC_TYPE ac_type = 0;
switch (g_hk_context.learn_mode) {
case HB1688_AC_MATCH:
if (is_ok) {
/* TODO: need to match use HB1688 lib, don't support on HB-M */
/* TODO: save match type to flash */
is_ac_match = 1;
ac_type = 65536;
user_flash_set_env_blob(BLOB_KEY_IS_MATCH, &is_ac_match,
sizeof(BLOB_TYPE_IS_MATCH));
user_flash_set_env_blob(BLOB_KEY_AC_TYPE, &is_ac_match,
sizeof(BLOB_TYPE_AC_TYPE));
user_player_play(AUDIO_PLAY_REPLY, REPLAY_MATCH_OK);
} else {
user_player_play(AUDIO_PLAY_REPLY, REPLAY_MATCH_FAIL);
}
break;
case HB1688_LEARN_ARC:
if (is_ok)
{ is_ac_matc= 0;
h
user_flash_set_env_blob(BLOB_KEY_IS_MATCH, &is_ac_match,
sizeof(BLOB_TYPE_IS_MATCH));
user_flash_set_env_blob(g_hk_context.lrc_code.key_word, buf, len);
user_player_play(AUDIO_PLAY_REPLY, REPLAY_LEARN_OK);
} else {
user_player_play(AUDIO_PLAY_REPLY, REPLAY_LEARN_FAIL);
}
break;
case HB1688_LEARN_URC:
if (is_ok) {
user_flash_set_env_blob(g_hk_context.lrc_code.key_word, buf, len);
```

```
user_player_play(AUDIO_PLAY_REPLY, REPLAY_LEARN_OK);
} else {
user_player_play(AUDIO_PLAY_REPLY, REPLAY_LEARN_FAIL);
}
break;
default:
break;
}
}
static int _speaker_mute_process(event_custom_setting_t *setting) {
if (0 == uni_strcmp(setting->cmd, "speakerMute")) {
g_speaker_mute = true;
} else {
g_speaker_mute = false;
user_player_speaker_unmute();
}
user_player_reply_list_random(setting->reply_files);
return 0;
}
static void _goto_ac_match_mode(void)
switch { (g_hk_context.learn_mode) {
case HB1688_LEARN_CODE:
/* TODO: exit code learn mode, don't support on HB-M */
break;
default:
break;
}
g_hk_context.lrc_code.len = 0;
g_hk_context.learn_mode = HB1688_AC_MATCH;
}
static void _goto_arc_learn_mode(void)
switch { (g_hk_context.learn_mode) {
case HB1688_LEARN_CODE:
/* TODO: exit code learn mode, don't support on HB-M */
break;
default:
break;
}
g_hk_context.lrc_code.len = 0;
g_hk_context.learn_mode = HB1688_LEARN_ARC;
}
static void _goto_urc_learn_mode(void)
switch { (g_hk_context.learn_mode) {
case HB1688_LEARN_CODE:
```

```
/* TODO: exit code learn mode, don't support on HB-M */
break;
default:
break;
}
g_hk_context.lrc_code.len = 0;
g_hk_context.learn_mode  = HB1688_LEARN_URC;
}
static void _goto_code_learn_mode(void)
switch { (g_hk_context.learn_mode) {
case HB1688_AC_MATCH:
user_uart_hb1688_learn_exit();
break;
case HB1688_LEARN_ARC:
user_uart_hb1688_learn_exit();
break;
case HB1688_LEARN_URC:
user_uart_hb1688_learn_exit();
break;
default:
break;
}
g_hk_context.lrc_code.len = 0;
g_hk_context.learn_mode = HB1688_LEARN_CODE;
}
static void _enter_learn_mode(void) {
LOGT(TAG, "enter learn_mode: %d", g_hk_context.learn_mode);
switch (g_hk_context.learn_mode) {
case HB1688_AC_MATCH:
user_uart_hb1688_arc_learn(_hb1688_lrc_learn_recv);;
break;
case HB1688_LEARN_ARC:
user_uart_hb1688_arc_learn(_hb1688_lrc_learn_recv);;
break;
case HB1688_LEARN_URC:
user_uart_hb1688_urc_learn(_hb1688_lrc_learn_recv);;
break;
case HB1688_LEARN_CODE:
/* TODO: enter code learn mode, don't support on HB-M */
break;
default:
break;
}
}
```

```
static void _exit_learn_mode(void) {
if (HB1688_LEARN_NONE != g_hk_context.learn_mode) {
user_uart_hb1688_learn_exit();
/* TODO: exit code learn mode, don't support on HB-M */
g_hk_context.lrc_code.len = 0;
g_hk_context.learn_mode  = HB1688_LEARN_NONE;
}
}
static int _learn_mode_process(event_custom_setting_t *setting) {
if (0 == uni_strcmp(setting->cmd, "learnModeExit")) {
_exit_learn_mode();
g_hk_context.is_learn  = false;
} else {
g_hk_context.is_learn = true;
g_hk_context.learn_mode  = HB1688_LEARN_NONE;
}
if (0 == _send_cmd_uart_protocol(setting->cmd)) {
user_player_reply_list_random(setting->reply_files);
}
return 0;
}
static int _work_mode_process(event_custom_setting_t *setting) {
/* TODO: send UART protocol at here */
if (0 == _send_cmd_uart_protocol(setting->cmd)) {
user_player_reply_list_random(setting->reply_files);
}
return 0;
}
static int _code_learn_mode_process(event_custom_setting_t *setting) {
_goto_code_learn_mode();
user_player_reply_list_num(setting->reply_files, 0);
return 0;
}
static int _lrc_learn_mode_process(event_custom_setting_t *setting) {
/* default to URC learn mode, will switch to ARC mode if user say AC command
       */
_goto_urc_learn_mode();
if (0 == _send_cmd_uart_protocol(setting->cmd)) {
user_player_reply_list_num(setting->reply_files, 0);
}
return 0;
}
static int _ac_match_process(event_custom_setting_t *setting) {
_goto_ac_match_mode();
```

```
_enter_learn_mode();
if (0 == _send_cmd_uart_protocol(setting->cmd)) {
user_player_reply_list_num(setting->reply_files, 0);
}
return 0;
}
static int _ac_key_learn_process(event_custom_setting_t *setting) {
/* default to URC learn mode, will switch to ARC mode if user say AC command
       */
if (HB1688_LEARN_ARC != g_hk_context.learn_mode
&& HB1688_LEARN_URC != g_hk_context.learn_mode) {
LOGE(TAG, "not in HB1688_LEARN_ARC or HB1688_LEARN_URC mode.");
return -1;
}
uni_strncpy(g_hk_context.lrc_code.key_word, setting->cmd,
sizeof(g_hk_context.lrc_code.key_word) - 1);
LOGT(TAG, "key_word: %s", g_hk_context.lrc_code.key_word);
_goto_arc_learn_mode();
_enter_learn_mode();
/* don't send protocol to MCU on learn mode */
user_player_reply_list_num(setting->reply_files, 0);
return 0;
}
static int _ac_key_use_process(event_custom_setting_t *setting) {
BLOB_TYPE_IS_MATCH is_ac_match = 0;
BLOB_TYPE_AC_TYPE ac_type = 0;
int save_len = 0;
/* send protocol in any case */
if (0 == _send_cmd_uart_protocol(setting->cmd)) {
user_player_reply_list_num(setting->reply_files, 0);
}
/* is AC match or not */
user_flash_get_env_blob(BLOB_KEY_IS_MATCH, &is_ac_match,
sizeof(BLOB_TYPE_IS_MATCH), &save_len);
if (save_len != sizeof(BLOB_TYPE_IS_MATCH)) {
LOGE(TAG, "cannot found %s in flash", BLOB_KEY_IS_MATCH);
return -1;
}
if (is_ac_match) {
user_flash_get_env_blob(BLOB_KEY_AC_TYPE, &ac_type,
sizeof(BLOB_TYPE_AC_TYPE), &save_len);
if (save_len != sizeof(BLOB_TYPE_AC_TYPE)) {
LOGE(TAG, "cannot found %s in flash", BLOB_KEY_AC_TYPE);
return -1;
```

```
}
/* TODO: get ARC code from code lib and send to UART */
} else {
/* find LRC code from flash by cmd */
user_flash_get_env_blob(setting->cmd, g_hk_context.lrc_code.code,
sizeof(g_hk_context.lrc_code.code),
&save_len);
if (save_len <= 0) {
LOGE(TAG, "cannot found %s in flash", setting->cmd);
return -1;
}
if (save_len > sizeof(g_hk_context.lrc_code.code)) {
save_len = sizeof(g_hk_context.lrc_code.code);
}
g_hk_context.lrc_code.len = (uni_u8)save_len;
user_uart_hb1688_arc_send(g_hk_context.lrc_code.code,
g_hk_context.lrc_code.len);
}
return 0;
}
static int _ac_command_process(event_custom_setting_t *setting) {
LOGT(TAG, "is_learn = %d", g_hk_context.is_learn);
if (g_hk_context.is_learn) {
return _ac_key_learn_process(setting);
} else {
return _ac_key_use_process(setting);
}
}
static int _tv_key_learn_process(event_custom_setting_t *setting) {
/* default to URC learn mode, will switch to ARC mode if user say AC command
      */
if (HB1688_LEARN_ARC != g_hk_context.learn_mode
&& HB1688_LEARN_URC != g_hk_context.learn_mode) {
LOGE(TAG, "not in HB1688_LEARN_ARC or HB1688_LEARN_URC mode.");
return -1;
}
uni_strncpy(g_hk_context.lrc_code.key_word, setting->cmd,
sizeof(g_hk_context.lrc_code.key_word) - 1);
_goto_urc_learn_mode();
_enter_learn_mode();
/* don't send protocol to MCU on learn mode */
user_player_reply_list_num(setting->reply_files, 0);
return 0;
}
```

```
static int _tv_key_use_process(event_custom_setting_t *setting) {
int save_len = 0;
/* send protocol in any case */
if (0 == _send_cmd_uart_protocol(setting->cmd)) {
user_player_reply_list_num(setting->reply_files, 0);
}
/* find LRC code from flash by cmd */
user_flash_get_env_blob(setting->cmd, g_hk_context.lrc_code.code,
sizeof(g_hk_context.lrc_code.code),
&save_len);
if (g_hk_context.lrc_code.len <= 0) {
LOGE(TAG, "cannot found %s in flash", setting->cmd);
return -1;
}
if (save_len > sizeof(g_hk_context.lrc_code.code)) {
save_len = sizeof(g_hk_context.lrc_code.code);
}
g_hk_context.lrc_code.len = (uni_u8)save_len;
user_uart_hb1688_urc_send(g_hk_context.lrc_code.code,
g_hk_context.lrc_code.len);
return 0;
}
static int _tv_command_process(event_custom_setting_t *setting) {
if (g_hk_context.is_learn) {
return _tv_key_learn_process(setting);
} else {
return _tv_key_use_process(setting);
}
}
static int _stb_command_process(event_custom_setting_t *setting) {
if (g_hk_context.is_learn) {
return _tv_key_learn_process(setting);
} else {
return _tv_key_use_process(setting);
}
}
static void _custom_setting_cb(USER_EVENT_TYPE event,
user_event_context_t *context) {
event_custom_setting_t *setting = NULL;
if (context) {
setting = &context->custom_setting;
LOGT(TAG, "user command: %s", setting->cmd);
if (0 == uni_strncmp(setting->cmd, "speaker", uni_strlen("speaker"))) {
_speaker_mute_process(setting);
```

```
}    else    if    (0    ==    uni_strncmp(setting->cmd,    "learnMode",
       uni_strlen("learnMode"))) {
_learn_mode_process(setting);
} else if (0 == uni_strncmp(setting->cmd, "user", uni_strlen("user"))) {
_work_mode_process(setting);
} else if (0 == uni_strncmp(setting->cmd, "ac", uni_strlen("ac"))) {
_ac_command_process(setting);
} else if (0 == uni_strncmp(setting->cmd, "tv", uni_strlen("tv"))) {
_tv_command_process(setting);
} else if (0 == uni_strncmp(setting->cmd, "stb", uni_strlen("stb"))) {
_stb_command_process(setting);
} else if (0 == uni_strncmp(setting->cmd, "code", uni_strlen("code"))) {
_code_learn_mode_process(setting);
} else if (0 == uni_strcmp(setting->cmd, "rcLearnKey")) {
_lrc_learn_mode_process(setting);
} else if (0 == uni_strcmp(setting->cmd, "rcAcMatch")) {
_ac_match_process(setting);
} else {
if (_send_cmd_uart_protocol(setting->cmd))
{ user_player_reply_list_random(setting->reply_files);
}
}
}
}
static void _audio_play_end_cb(USER_EVENT_TYPE event,
user_event_context_t *context) {
LOGT(TAG, "g_speaker_mute = %d", g_speaker_mute);
if (g_speaker_mute) {
g_speaker_mute = false;
user_player_speaker_mute();
}
}
static void _goto_sleeping_cb (USER_EVENT_TYPE event,
user_event_context_t *context) {
event_goto_sleeping_t  *sleeping = NULL;
if (context) {
sleeping = &context->goto_sleeping;
user_player_reply_list_random(sleeping->reply_files);
_exit_learn_mode();
g_hk_context.is_learn  = false;
}
}
static void _register_event_callback(void) {
user_event_subscribe_event(USER_CUSTOM_SETTING, _custom_setting_cb);
```

```
user_event_subscribe_event(USER_AUDIO_PLAY_END, _audio_play_end_cb);
user_event_subscribe_event(USER_GOTO_SLEEPING, _goto_sleeping_cb);
}
static void _hb1688_version_recv(uni_bool is_ok, uni_u16 version)
uni_u8 { ver_master = version >> 8;
uni_u8 ver_major = version & 0x0f;
if (is_ok) {
LOGT(TAG, "HB1688 version: %d.%d", ver_master, ver_major);
} else {
LOGE(TAG, "Get HB1688 version failed.");
}
}
int hb_housekeeper(void) {
BLOB_TYPE_IS_MATCH is_ac_match = 0;
if (0 != user_uart_hb1688_init())
LOGE(TAG, { "user_uart_hb1688_init
          failed.");
return -1;
}
if (0 != user_flash_init())
LOGE(TAG, { "user_flash_init
          failed.");
user_uart_hb1688_final();
return -1;
}
user_flash_set_env_blob(BLOB_KEY_IS_MATCH, &is_ac_match,
sizeof(BLOB_TYPE_IS_MATCH));
user_uart_hb1688_get_version(_hb1688_version_recv);
_register_event_callback();
return 0;
}
```