MVs 18 SDK 使用手册

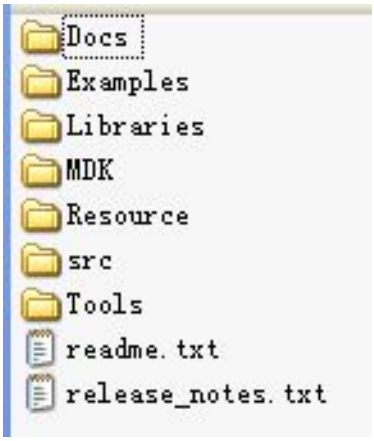
目录

1 概述

MVs18 SDK 是用于山景集成电路 O18 芯片的软件开发包。此开发包为客户提供了文档、示例、工具、Lib 库文件以及 SDK 源文件。此版 SDK 具有快速开发、灵活、易扩展等特点。既提供了带有 OS 的整体解决方案，又提供了脱离 OS 运行的 driver 库、audio 库以及实例工程。

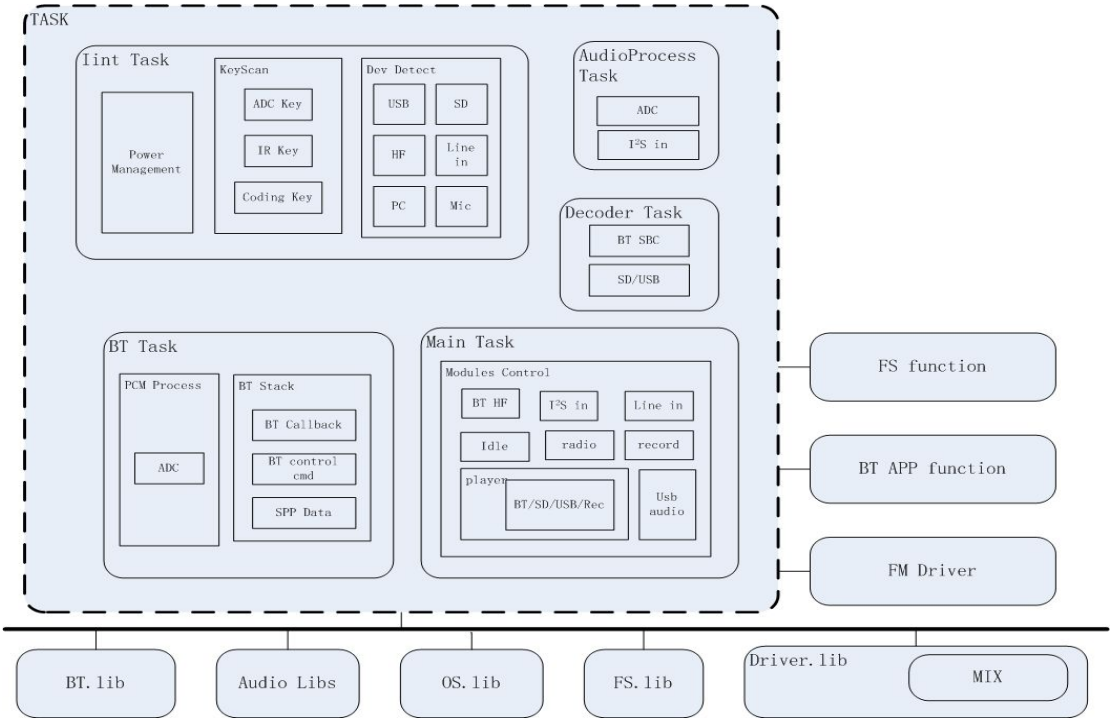
1.1 SDK 开发包说明

SDK 开发包包含了以下目录以及文件



[Docs] --开发文档
[Examples]--基于各硬件模块和库功能的示例代码
[Libraries]-- 库文件及库的头文件
[MDK]--音频应用方案 IDE 工程
[MDK\Flash]--MDK 的 Flash 驱动文件
Resource]--资源文件夹，含语音播报 MVA 升级包
[src]--音频应用方案的源文件
[Tools]--工具包，提供一些实用工具
release_history.txt --- 发布包历史更新记录

2 SDK 软件架构



SDK 是由 5 个主要 TASK、外部函数以及底层提供的库组成。5 个 Task 分别为：

任务名称	优先级	功能	备注
Init Task	3	1. 系统初始化； 2. 创建任务：创建 BtTask/DecoderTask 等 Task； 3. 电压检测 4. 升级检测 5. 按键检测；(ADC/IR/...) 6. 设备状态检测；(U/SD/..)	此 Task 是由系统启动自动创建的主 Task
Bluetooth Task	2	1. 运行蓝牙协议栈 2. 处理蓝牙 HF 通话时的语音数据	Bt task 优先级设置为 2，如果遇到蓝牙协议栈运行不流畅可以将优先级适当调为 3
Decoder Task	3	负责本地音频解码，蓝牙音	本地播歌时，该

		频解码以及语音提示音的解码	Task 负责解码工作，与 MainTask 中的 PlayControl 模块组合形成完整的播放模块；
Main Task	2	1. 处理设备插拔消息，负责模式切换； 2. 进入某一模式，并在此模式中处理各种消息。	
AudioProcess Task	3	负责 MIC、LINEIN 等 ADC 采样转换，以及 I^2S IN 数据等输入方面的处理工作	

3 OS

MVs 18 SDK 采用了 FreeRTOS 作为自己的操作系统。FreeRTOS 是一款开源免费的微内核嵌入式 OS。目前此版 SDK 主要用到了以下几个功能：

任务管理

内存管理

消息管理

3.1 任务管理

3.1.1 任务(TASK)概念

任务就是一段可以执行的代码，拥有自己的栈资源，有自己的生命周期，可以被创建、删除。一般任务都是由 while(1)来循环执行的。

3.1.2 创建任务

任务要有自己的入口函数，犹如主程序的 main 函数，任务创建后，系统会去从入口函数执行里面的代码。任务还要有优先级，当 OS 进行调度时，会根据优先级的高低来优先调度高优先级的任务。同时，随着任务的创建，系统会分配给任务相应的栈资源空间。

OSTaskCreate(TaskEntry,TaskName,TaskStackSize,TaskPara,TaskPri,TaskHandler)
参数说明：

TaskEntry – 入口函数
TaskName – 任务名称
TaskStackSize – 任务栈大小
TaskPara – 任务参数
TaskPri – 任务优先级，数值越高优先级越高
TaskHandler – 任务创建后返回的句柄

例子：

OSTaskCreate(BTEntrance, "BT", 1920, NULL, 2, &BtTaskHandle)

3.1.3 任务出让 CPU 机制

FreeRTOS 支持多任务并发执行，所以各个任务之间可以独自运行。在某一个时间片内，只能有一个任务在执行，任务的切换是由 OS 来管理的。目前，OS 给每个任务分配的时间片为 10ms。但是根据任务的优先级，OS 会优先分配给高优先级的任务，所以在高优先级的任务中，一定要有主动出让 CPU 机制，才能让低优先级的任务得到 CPU 执行程序。

SDK 中任务的出让机制有两种：

OSRescheduleTimeout(TimeOut);

OSQueueMsgRecv(MsgBody,MsgLen,WaitTime);

两种出让 CPU 机制的区别：

OSRescheduleTimeout -指定时间内不会唤醒此 Task，只有到了时间，Task 会被唤醒执行

OSQueueMsgRecv -在指定时间内，如果有消息接收到，会自动唤醒 Task。第二种机制的改进写法是：uint16_t MsgRecv(uint16_t WaitTime)

3.2 内存管理

O18B 有 128KB 可用 RAM 空间，分为 X-MEM(64KB), V-MEM(32KB), P-MEM(32KB)。OS 负责管理的是 64KB X-MEM。（有关 MEM 详细分配情况请参照 Memory Map 章）

3.2.1 申请内存

程序可以通过调用 APP_MMM_MALLOC 向 OS 申请内存。

APP_MMM_MALLOC(MemSize, MemType)

返回值：成功为内存指针地址，否则为 NULL

MemType 指的是类型，主要用于优化内存配置，主要有以下几个参数：

MMM_ALLOC_NORMAL
MMM_ALLOC_FAST
MMM_ALLOC_TASK
MMM_ALLOC_BOOTUP
MMM_ALLOC_MERGE

几种参数的区别：

MMM_ALLOC_NORMAL

MMM_ALLOC_FAST

normal 和 fast 类似，都是分配 low memory，适用于经常申请和释放的内存代码

MMM_ALLOC_TASK

MMM_ALLOC_BOOTUP

task 和 bootup 都是分配 high memory，用于申请之后不再释放的内存代码

MMM_ALLOC_MERGE

在申请大内存时，与上面几种类型配合使用，OS 内部会将内存重新整理，将空余出合适的内存给用户使用

以上的使用都是内存申请的高级应用，一般建议用户使用 NORMAL 类型

3.2.2 释放内存

使用 APP_MMM_FREE 释放掉内存

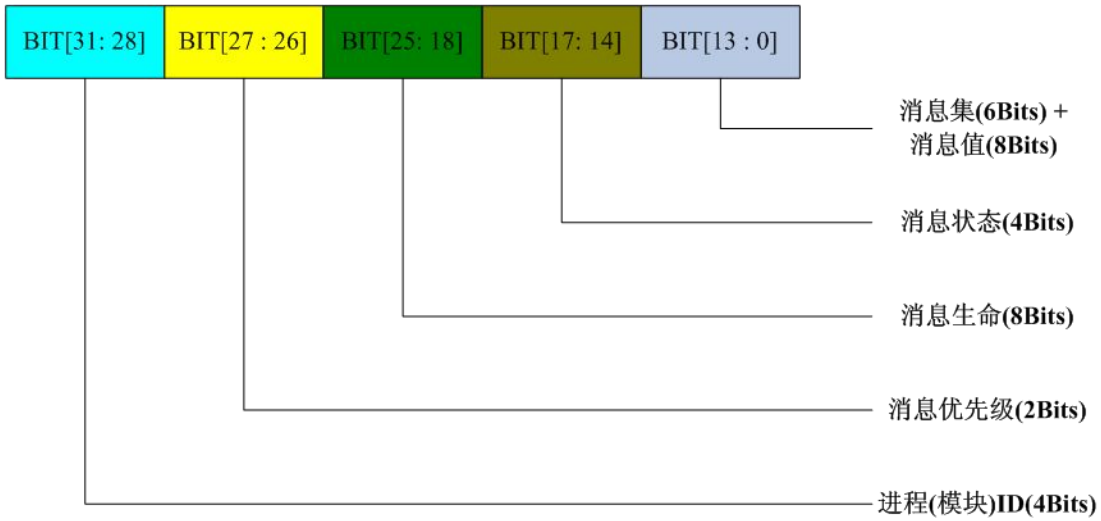
```
APP_MMM_FREE(MemPtr);
```

3.3 消息管理

消息机制是模块之间进行异步通信的一种机制，是多 Task 系统中常用的一种通信方式，消息机制的基本流程是一个模块先发送消息到消息队列，然后另一个模块(可以是发送者本身)轮询消息队列，按优先级逐个获取和自己相关的消息。

3.3.1 消息定义

一般用户只需要关心低位 14Bits 的消息集+消息值，高位都由 OS 内部维护。



3.3.2 消息集

简单称一组消息的集合。消息集是消息 ID 高 6Bits 相同的一类消息集合，每个消息集最多包含 255 个消息。一般是按照模块或者功能相关的一组消息定义为一个集合。这样做的目的是，在任何一个 Task 中，只要将消息集添加进 Task 的消息队列就可以接收这个消息集中的任何一个消息。目前 SDK APP 层的消息集有 MSG_MAIN_CLASS、MSG_DEV_CLASS、MSG_DECODER_INT_CLASS、MSG_USB_DEVICE_CLASS、MSG_BT_CLASS 这五个消息集。（参照 app_msg.h）

3.3.3 消息注册函数

只有 Task 注册了消息，OS 才能将消息发送给此 Task。消息注册是以消息集为单位的，对应的函数：

void MsgAddSet(uint32_t Msg);

参数 Msg 指的是消息集

例如 MsgAddSet(MSG_MAIN_CLASS);

3.3.4 消息注销函数

当注销掉消息后，此 Task 将不再会接收到注销掉的消息，消息注销也是以消息及为单位的。对应的函数为：

void MsgDelSet(uint32_t Msg);

参数 Msg 指的是消息集

3.3.5 消息发生函数

消息发生函数定义为：

OSQueueMsgSend(Msg,MsgBody,MsgLen,Prio);

参数说明：Msg，消息号；MsgBody，消息体；MsgLen，消息体长度；Prio，优先级，优先级定义为以下三种

```
#define     MSGPRIO_LEVEL_HI    2
#define     MSGPRIO_LEVEL_MD    1
#define     MSGPRIO_LEVEL_LO    0
```

高优先级的消息会在接收任务的消息队列中首先接收到，一般设备插拔的消息都是设为高优先级。

另外，在只关注消息号的情况下，SDK 中做了简化，void MsgSend(uint16_t Msg)，此处参数 Msg 为消息号，优先级设置为高

3.3.6 消息接收函数

消息接收函数定义为：

OSQueueMsgRecv(MsgBody,MsgLen,WaitTime);

返回值：为消息号

参数说明：MsgBody，消息体；MsgLen，消息体长度；WaitTime，等待接收消息的超时时间

注：此接收消息函数会主动出让 CPU，在接收到消息时会立即返回。

在只关注消息号的情况下，SDK 中做了简化：

uint16_t MsgRecv(uint16_t WaitTime);

4 中断

中断是由硬件产生的一种信号。当一种中断使能，并且中断产生的时候，系统会调用相应的中断函数进行处理。

在 SDK 中只需要关心可以使用的中断号，以及对应的中断处理函数即可。中断处理函数与中断号对应关系可以在中断向量表中得到。中断向量表定义在 startup.c 中

中断号定义	中断处理函数
GPIO_IRQn = 0	GpioInterrupt,
RTC_IRQn = 1	RtcInterrupt,
IR_IRQn = 2	IrInterrupt,
FUART_IRQn = 3	FuartInterrupt,
BUART_IRQn = 4	BuartInterrupt,
PWC_IRQn = 5	PwcInterrupt,
TMR0_IRQn = 6	Timer0Interrupt,
USB_IRQn = 7	UsbInterrupt,
DMACH0_IRQn = 8	DmaCh0Interrupt,
DMACH1_IRQn = 9	DmaCh1Interrupt,
DECODER_IRQn = 10	audio_decoder_interrupt_handler,
SPIS_IRQn = 11	SpisInterrupt,
SD_IRQn = 12	SdInterrupt,
SPIM_IRQn = 13	SpimInterrupt,
TMR1_IRQn = 14	Timer1Interrupt,
WDG_IRQn = 15	WatchDogInterrupt,

中断处理函数举例

```
// Coding key signal-A interrupt.
__attribute__((section(".driver.isr"))) void GpioInterrupt(void)
{
    if(GpioIntFlagGet(CODING_KEY_A_PORT_INT) == CODING_KEY_A_BIT)
    {
        GpioIntClr(CODING_KEY_A_PORT_INT, CODING_KEY_A_BIT);
        if((GpioGetReg(CODING_KEY_A_PORT_IN) & CODING_KEY_A_BIT) || (Cl
        {
            return;
        }

        if(GpioGetReg(CODING_KEY_B_PORT_IN) & CODING_KEY_B_BIT)
        {
            //clockwise rotation
            ClockWiseCnt++;
        }
        else
        {
            //counterclockwise rotation
            CounterClockWiseCnt++;
        }
    }
}
```

5 SDK 中的 Task 功能以及流程

5.1 Init Task

主要用途：

喂狗 WdgFeed();

电压检测 PowerMonitor();

升级检测 BootUpgradeChk();

键盘扫描 KeyScan();

设备检测 DeviceDetect(); 用于切换模式。

主要流程图：

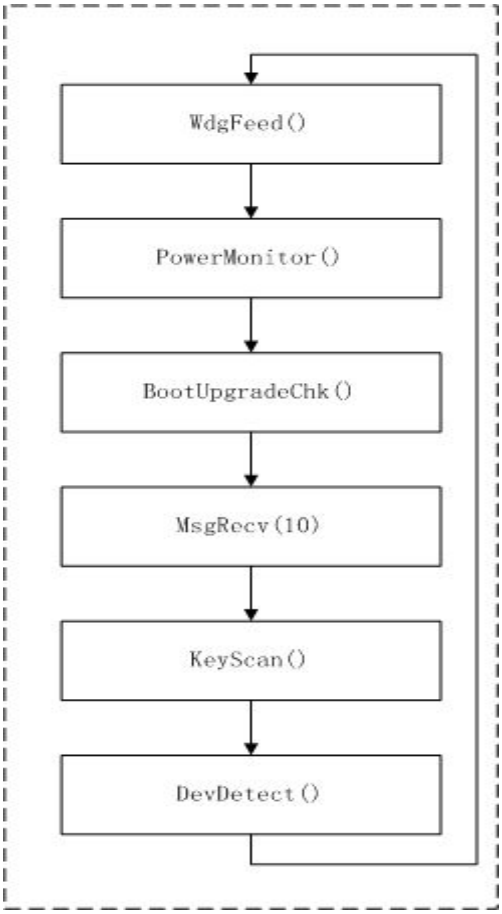


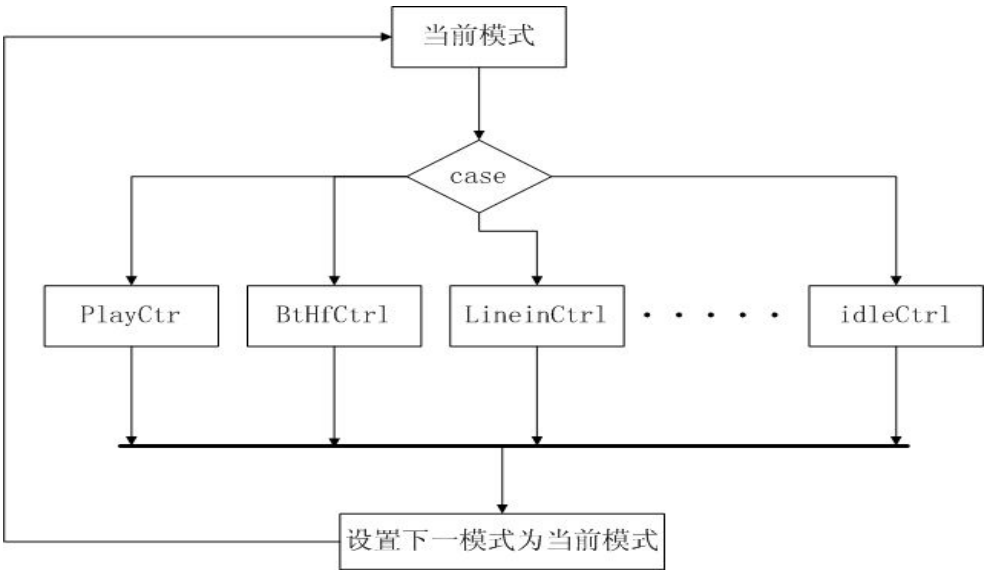
图 InitTask 流程图

5.2 Main Task

主要用于进入某一模式，并在此模式中执行该模式的操作。

在 main 函数中创建，如下：

```
OSTaskCreate(GuiTaskEntrance, "MainTask", 1280, NULL, 2, &MainTaskHandle);
```



5.3 Decoder Task

主要用于蓝牙 A2DP/USB/SD 卡音频数据以及提示音的 Decoder

Main 函数中创建：

OSTaskCreate(DecoderTaskEntrance, "Decoder", 2048, NULL, 3, NULL);

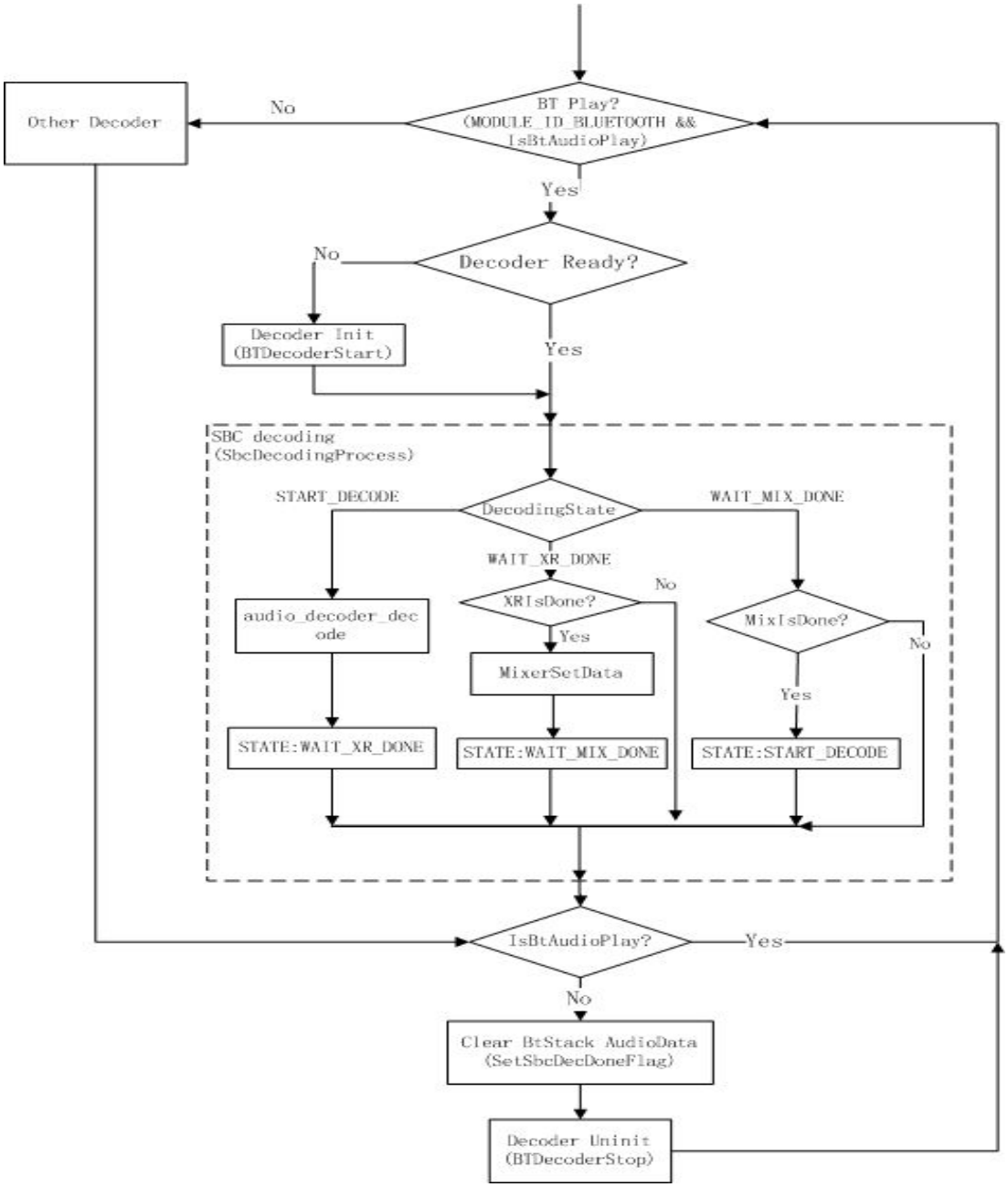


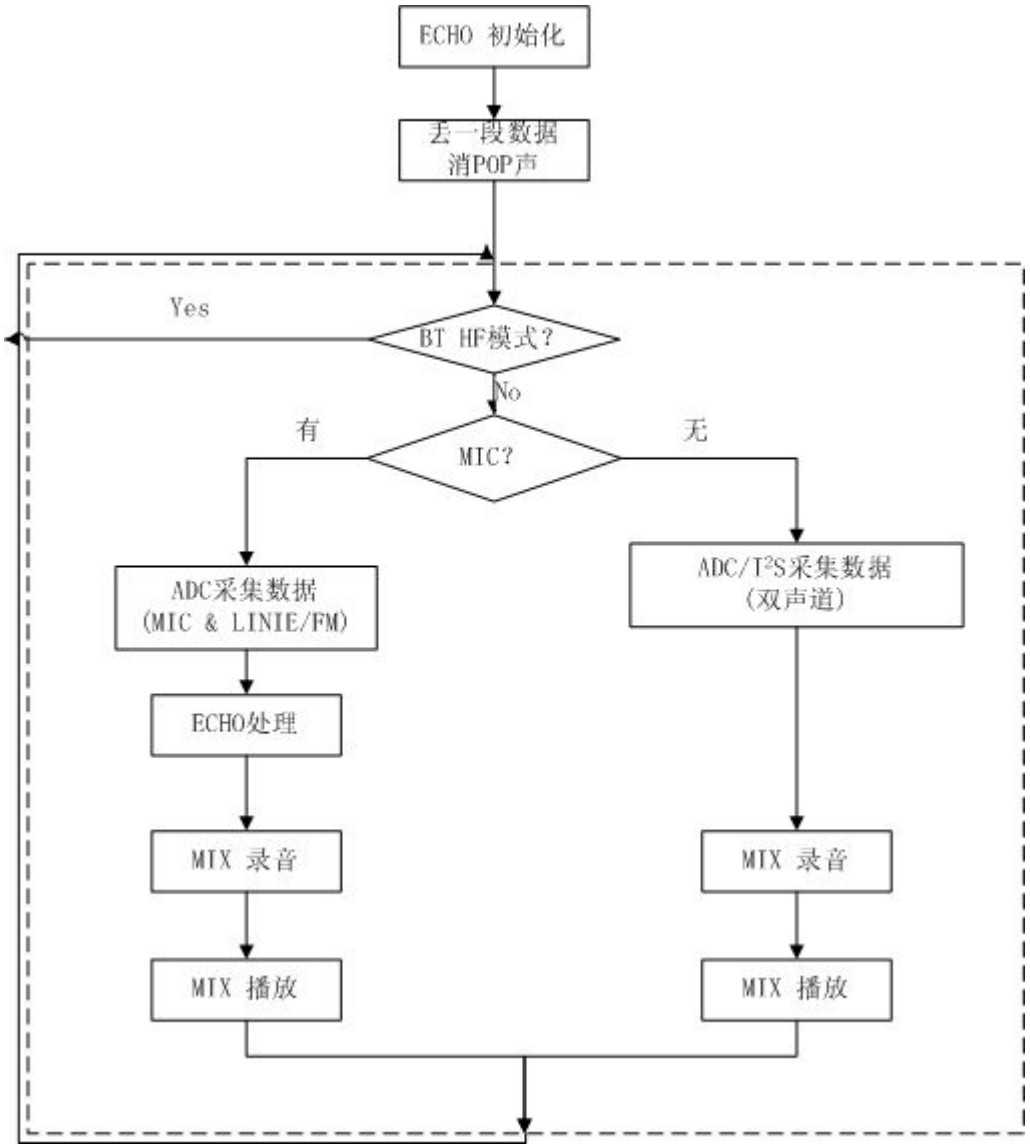
图 Decoder Task（蓝牙部分）流程图

5.4 AudioProcess Task

主要用于 MIC/I2S/LineIn/Recoder/Radio 音频数据采集与处理的 task

main 函数中创建：

OSTaskCreate(AudioProcessTaskEntrance, "AudioProcessTask", 1024, NULL, 3, NULL);



5.5 Bluetooth Task

主要运行蓝牙协议栈、以及通话过程中的数据(SCO)处理

创建：

OSTaskCreate(BTEntrance, "BT", 1920, NULL, 2, &BtTaskHandle);

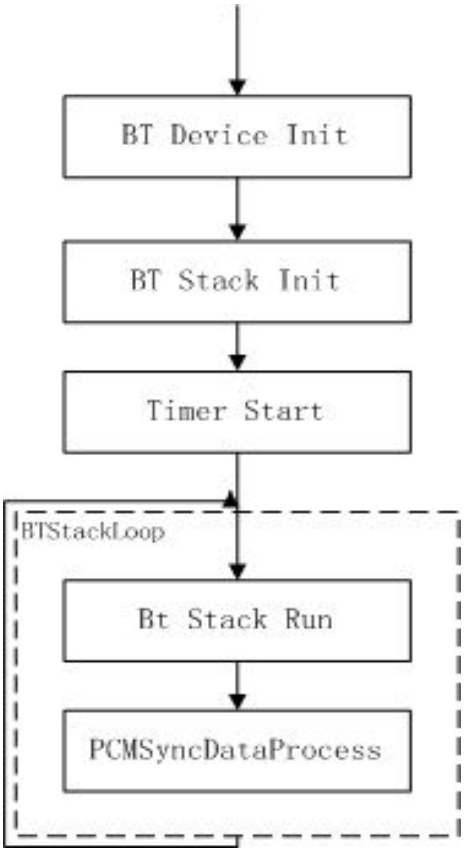


图 Bluetooth Task 流程图

6 模式以及流程控制

目前 SDK 有 13 种主要模式，9 个 control 模块。有些模式是有单独的 control 模块进行控制，有些模式是合并在一种 control 模块中，例如蓝牙的播歌模式与 SD 卡播放模式、USB 播放模式、录音回放模式都是放在了 play control 中。还有 USB 声卡模式、USB 读卡器模式、USB 声卡/读卡器模式放在了 usb audio control。

模式与控制之间的关系如下图所示：

模式	控制
MODULE_ID_PLAYER_SD, MODULE_ID_PLAYER_USB, MODULE_ID_BLUETOOTH, MODULE_ID_REC_BACK_PLAY	Play control
MODULE_ID_RECORDER,	Record Control
MODULE_ID_RADIO,	Radio Control
MODULE_ID_LINEIN	Line In Control
MODULE_ID_I2SIN,	I2S In Control
MODULE_ID_USB_AUDIO, MODULE_ID_USB_READER, MODULE_ID_USB_AUDIO_READER,	USB Audio Control

MODULE_ID_RTC	RTC Control
MODULE_ID_BT_HF	Bluetooth HF Control
NONE MODULE	Idle Control

6.1 模式切换流程

当设备进行插入/拔出动作时，会产生插入/拔出的消息，此消息发送给 Main Task，由 Main Task 进行模式的切换动作。进入蓝牙 HF 模式是通过蓝牙的事件通知，得到蓝牙来电、去电或者挂断电话的通知时，蓝牙 Task 就发送类似插入/拔出的消息给 Main Task，Main Task 再进行模式切换，进入/退出蓝牙 HF 模式。

具体流程如下图所示：

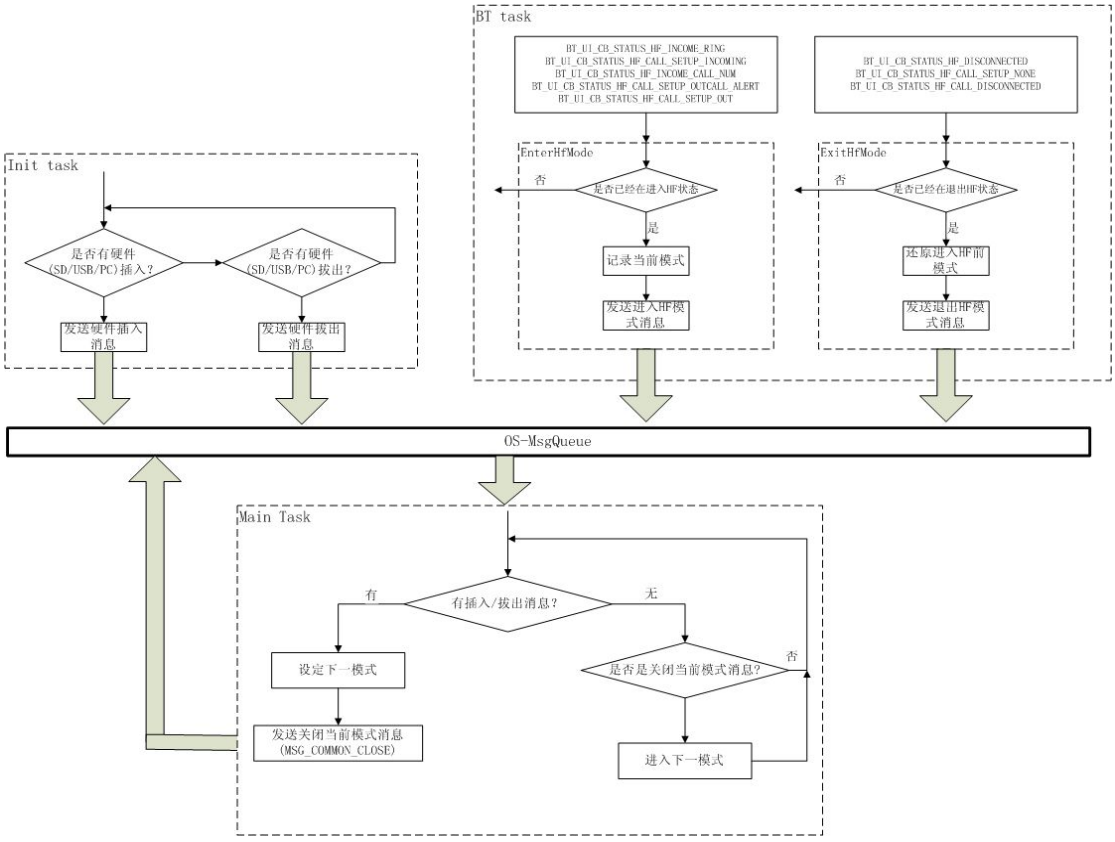


图 模式切换流程图

6.2 流程控制

9 种控制模块，其中 PlayControl 较为复杂，下面为 Playcontrol 示意图

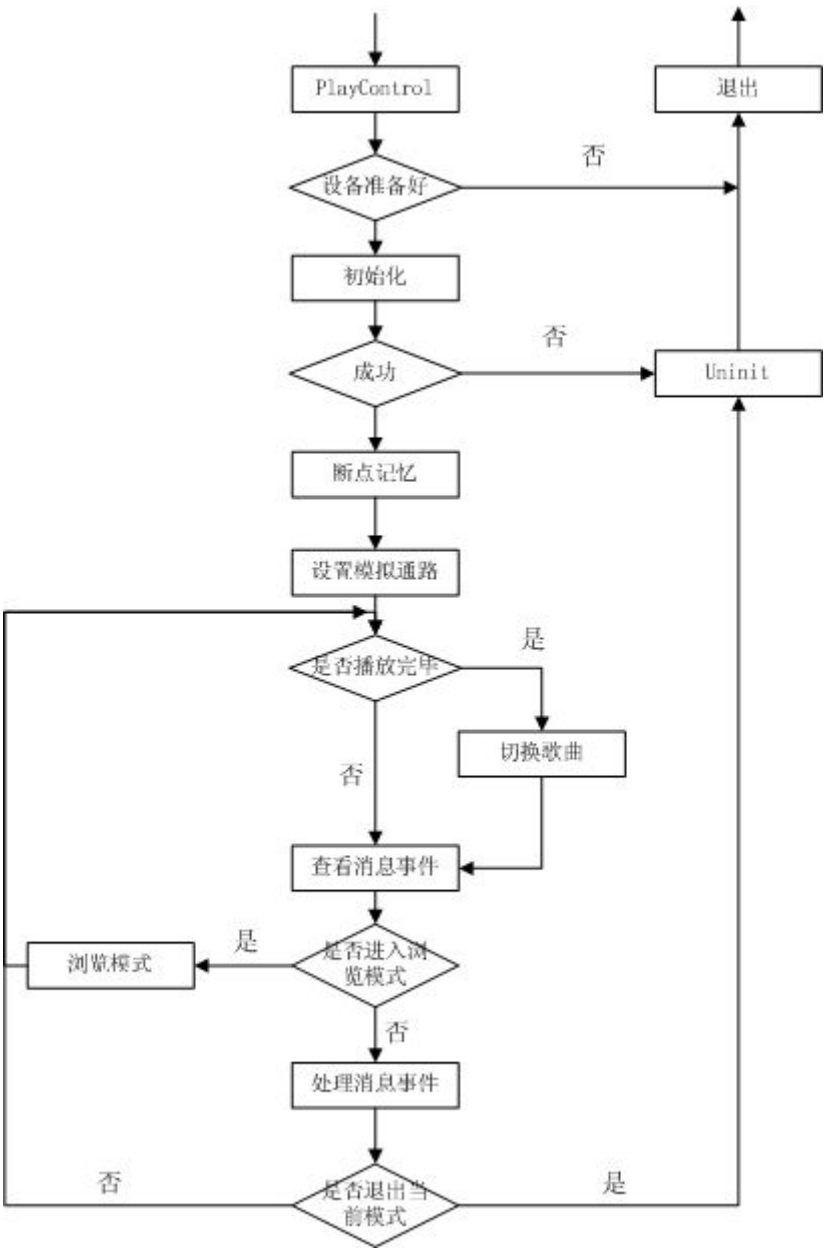


图 PlayControl 流程示意图

注：在播放过程中，包含了进入浏览模式的情况。浏览模式就是可以浏览 U 盘/SD 卡上的目录内容以及文件。在目前的 SDK 版本中，进入浏览模式必须先进入播放模式才可以。

7 按键（KEY）

7.1 Key 概述

目前支持 ADC Key、IR Key、Coding Key。按键的扫描是在 init task 中，会按照 ADC、IR、Coding 的顺序扫描三种按键。例如当 ADC 按键得到取值后，其他两种按键就在本次循环中不再扫描。同样，当 IR 键值有效时，Coding Key 将不做扫描。

Key 扫描流程如下图

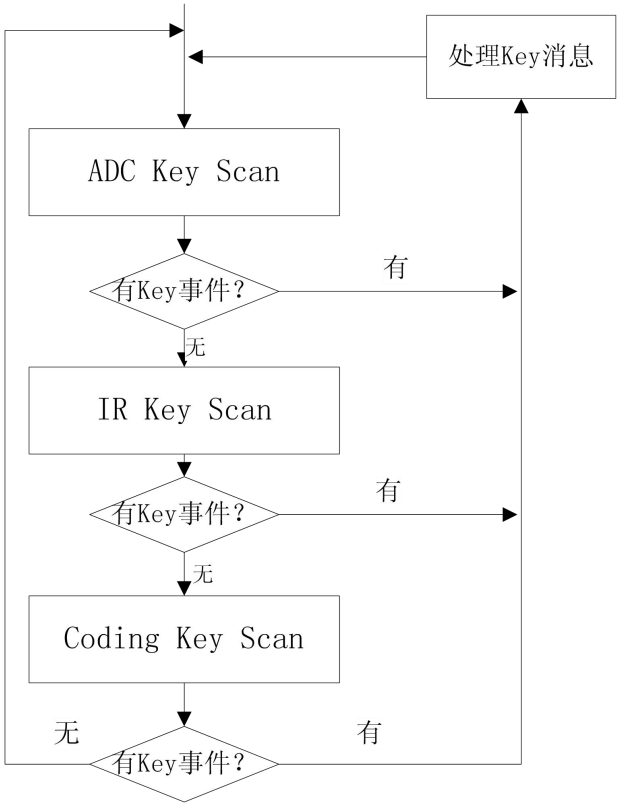


图 KeyScan 流程

7.2 Key 与消息

7.2.1 ADC Key 与消息

ADC Key 与消息的对应关系如下表

消息 / 动作 / 按键	短按抬起	长按开始	长按保持	长按抬起
SW 1	MSG_PLAY_PAUSE	MSG_STOP	MSG_NONE	MSG_NONE
SW 2	MSG_PRE	MSG_FB_START	MSG_FB_START	MSG_FF_FB_END
SW 3	MSG_NEXT	MSG_FF_START	MSG_FF_START	MSG_FF_FB_END
SW 4	MSG_VOL_UP	MSG_VOL_UP	MSG_VOL_UP	MSG_NONE
SW 5	MSG_VOL_DW	MSG_VOL_DW	MSG_VOL_DW	MSG_NONE
SW 6	MSG_EQ	MSG_3D	MSG_NONE	MSG_NONE
SW 7	MSG_MUTE	MSG_NONE	MSG_NONE	MSG_NONE
SW 8	MSG_REC	MSG_REC_PLAYBACK	MSG_NONE	MSG_NONE
SW 9	MSG_MODE	MSG_POWER	MSG_NONE	MSG_NONE
NONE	MSG_NONE	MSG_NONE	MSG_NONE	MSG_NONE
SW 10	MSG_MENU	MSG_BROWSE	MSG_NONE	MSG_NONE

SW 11	MSG_FOLDER_PRE	MSG_FOLDER_MODE	MSG_NONE	MSG_NONE
SW 12	MSG_FOLDER_NEXT	MSG_FOLDER_MODE	MSG_NONE	MSG_NONE
SW 13	MSG_REPEAT	MSG_REPEAT_AB	MSG_NONE	MSG_NONE
SW 14	MSG_REMIND	MSG_LANG	MSG_NONE	MSG_NONE
SW 15	MSG_NONE	MSG_NONE	MSG_NONE	MSG_NONE
SW 16	MSG_NONE	MSG_NONE	MSG_NONE	MSG_NONE
SW 17	MSG_RTC_SET_TIME	MSG_NONE	MSG_NONE	MSG_NONE
SW 18	MSG_RTC_SET_ALARM	MSG_NONE	MSG_NONE	MSG_NONE
NONE	MSG_NONE	MSG_NONE	MSG_NONE	MSG_NONE

7.2.2 IR Key 与消息

（略，可参照 IR_Key.c 中定义）

7.2.3 Coding Key 与消息

Coding Key 产生的消息只有两种：

MSG_VOL_UP

MSG_VOL_DW

7.3 Key 与 Beep 声

为了简化代码，将 Beep 声放在 KeyScan 中，当有 Key 的事件产生时，就产生 Beep 声

Beep 声产生函数：

bool BeepStart(uint16_t BeepFreq, uint16_t VolGain, uint16_t SampleRate, uint16_t OnTime, uint16_t FallTime);

参数介绍：

BeepFreq：Beep 的频率

VolGain：Beep 的音量

SampleRate：Beep 采样率，要跟 DAC 一致

OnTime：Beep 持续时间 ms

FallTime：Fade out 时间 ms

例子：

BeepStart(3200, 2048, SampleRateTmp, 25, 10);

7.4 Key 的复用（音效设置）

当进行 MIC 或音效设置时，音量+-键将变为，音效设置+-键。例如：当进行 ECHO DEPTH 音效设置时，音量+ 即为回声深度+，音量-为回声深度-

当 5 秒不按按键操作或者按 MENU 键适当次数退出音效设置时，音量+-键将回到原始功能，即音量的+-。

Key 的复用是使用宏 FUNC_AUDIO_MENU_EN 来定义的。

8 蓝牙(Bluetooth)

8.1 概述

目前 SDK 支持的蓝牙 Profile 有 A2DP、AVRCP(Advance AVRCP)、HFP 和 SPP。支持的蓝牙 RF 芯片为：RDA5875,CSRBC6,BCM20702,MTK662X。

8.2 初始化

8.2.1 设备初始化

设备初始化函数为：

bool BTDeviceInit(uint8_t BTDeviceType, int8_t* BdAdd/*6Bytes,LE*/);

参数说明：

BTDeviceType 即芯片的类型

BdAdd 为蓝牙的地址

8.2.2 协议栈初始化

协议栈初始化函数为：

uint32_t BTStackRunInit(uint8_t BTDeviceType, uint8_t *UserDefinedDevLocalName, uint8_t BtFeatureID);

参数说明：

BTDeviceType 要与设备初始化的类型一致；

UserDefinedDevLocalName 为蓝牙的可见名字。

BtFeatureID 为协议栈支持的 Profile,目前支持 A2DP（AVRCP）、Advance AVRCP、HFP 和 SPP。支持 Profile 组合为下定义：

```
//supported all features:A2DP+HF+SPP
#define    BT_FEATURE_SUPPORTED_ALL                    0x00    /*default*/

//supported all featrues except the SPP
#define    BT_FEATURE_SUPPORTED_NORMAL                 0x01

//supported A2DP ONLY
#define    BT_FEATURE_SUPPORTED_HIFI_ONLY              0x02

//supported A2DP + SPP
#define    BT_FEATURE_SUPPORTED_HIFI_SPP               0x04
```

8.3 蓝牙回连机制

蓝牙回连指的是 SDK 连接最近一次连接过的蓝牙（手机）设备。如果没有连接过或者 Flash 被擦除，则无法回连。

回连机制相关函数：

void StartBtReConnection(uint8_t reconnection_count);

此函数为，启动回连机制，参数 reconnection_count 为尝试连接次数

void StopBtReConnection(void);

此函数为，停止回连机制，不管是否已经达到尝试次数，都将停止。

bool IsBtReConnectionFinish(void);

此函数为判断是否回连机制结束或者代表是否正在回连中。

重连间隔时间(RECONNECTION_PERIOD_TIME)

在回连尝试次数大于一次时，就存在两次回连间隔时间问题。间隔时间是为了让其他设备在此间隔时间内与 SDK 的蓝牙进行配对和连接。避免了在整个回连机制中，其他设备无法与 SDK 进行连接的问题。

8.4 Bluetooth Audio Play(A2DP & AVRCP)

A2DP 指的是蓝牙的音乐，AVRCP 指的是蓝牙的播放控制。目前 SDK 同时支持这两种 Profile，连接 A2DP 时，AVRCP 也会自动连接上。

8.4.1 A2DP

A2DP，SDK 中 A2DP 有 4 中状态

```
enum
{
    BT_A2DP_STATE_NONE = 0,
    BT_A2DP_STATE_CONNECTING,
    BT_A2DP_STATE_CONNECTED,
    BT_A2DP_STATE_STREAMING
};
```

其中 BT_A2DP_STATE_STREAMING 状态代表有蓝牙音频数据在传输

8.4.2 AVRCP

AVRCP 是用来传输蓝牙的控制命令，用来控制音乐的播放;

ACRCP 控制命令见 bt_control_api.h 定义

8.4.3 Advance AVRCP

即高级 AVRCP 功能，支持 Advance AVRCP 的代表是 APPLE 产品。Advance AVRCP 可以支持显示更多音乐播放的内容信息和更多控制。例如可以显示音乐的播放时间，可以设置重复或者随机等，其中 Advance AVRCP 占用 Memory 1.5K

8.5 Bluetooth HFP

Hands Free 是蓝牙比较重要的一个 Profile，是指在打电话时蓝牙的一些行为。

8.5.1 HFP 事件通知

HFP 事件通知是由底层蓝牙协议栈收到蓝牙设备发送来的的事件，然后上报给上层应用，来表示蓝牙设备的动作。这些事件定义在 bt_app_ui_callback.h 和 bt_play_internal_api.h 中。

HFP 的事件通知与状态机如下图所示

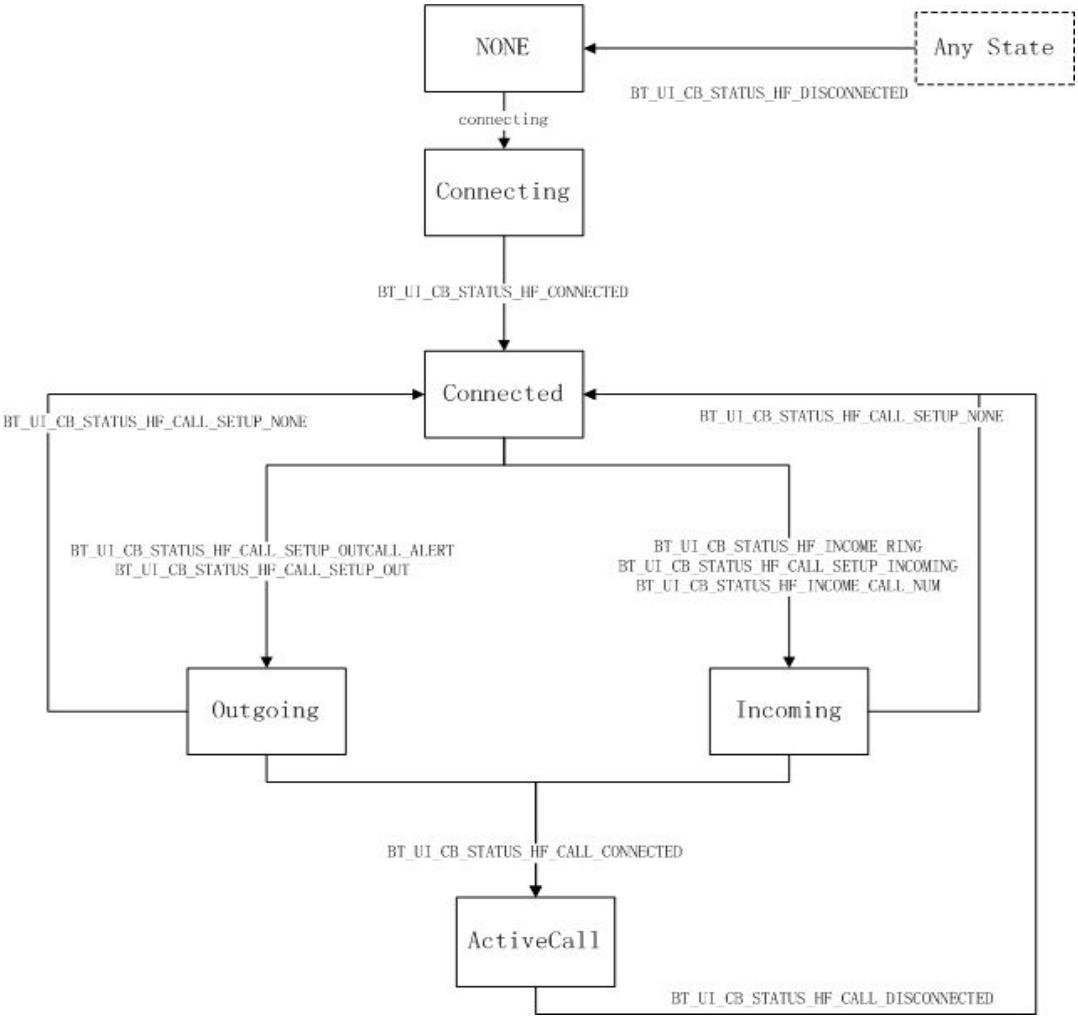


图 蓝牙事件通知与状态机

8.5.2 HFP 控制命令

HFP 的控制命令详见 bt_control_api.h 定义

8.6 Bluetooth SPP

8.6.1 BT SPP 功能

手机可以通过 SPP 发送数据到 SDK 上，然后 SDK 软件端解析数据，执行相应的命令来控制本地的行为。例如可以读取 SD 卡/U 盘上的内容，播放 SD 卡/U 盘的歌曲，同时可以进行上一首、下一首等操作。目前 SPP 只支持 Android 手机

8.6.2 占用 RAM 空间

其中 SPP 占用 Memory 4.3K 空间

9 通路与 MIX

9.1 通路概念

通路指的就是数据从输入到处理到输出的路径。

函数：

bool PhubPathSel(PHUB_PATH_SEL PhubPath)

参数说明：PhubPath，见 audio_path.h 定义。

函数参数非常繁琐，所以做了简化。

9.2 输入

目前 SDK 支持的模拟输入源为 MIC、LINEIN、FM、I2SIN、MIC+LINEIN、MIC+FM 这六种，使用函数 void AudioAnaSetChannel(uint8_t Channel); 来设置模拟输入源，对应的参数值为：

```
//模拟输入源定义
#define AUDIO_CH_NONE     0     //无输入
#define AUDIO_CH_MIC      1     //单声道 MIC
#define AUDIO_CH_LINEIN   2     //双声道 LINEIN
#define AUDIO_CH_FM       3     //双声道 FM
#define AUDIO_CH_MIC_LINEIN   4     //MIC +  单声道 LINEIN
#define AUDIO_CH_MIC_FM 5     //MIC +  单声道 FM
#define AUDIO_CH_I2SIN          6          //双声道 I2Sin
```

9.3 输出

O18 输出通路，默认为 DAC 输出。可以支持 DAC、I²S、ClassD、I²S+DAC 4 种输出。 如果使用的是 DAC、I²S、I²S+DAC 这三种输出，时钟使用的是 USB MODE 即 12M 的时钟。此时函数为：

void AudioI2sOutSetChannel(uint32_t SampleRate, bool isAndDacOut);

参数说明：

SampleRate – 采样率

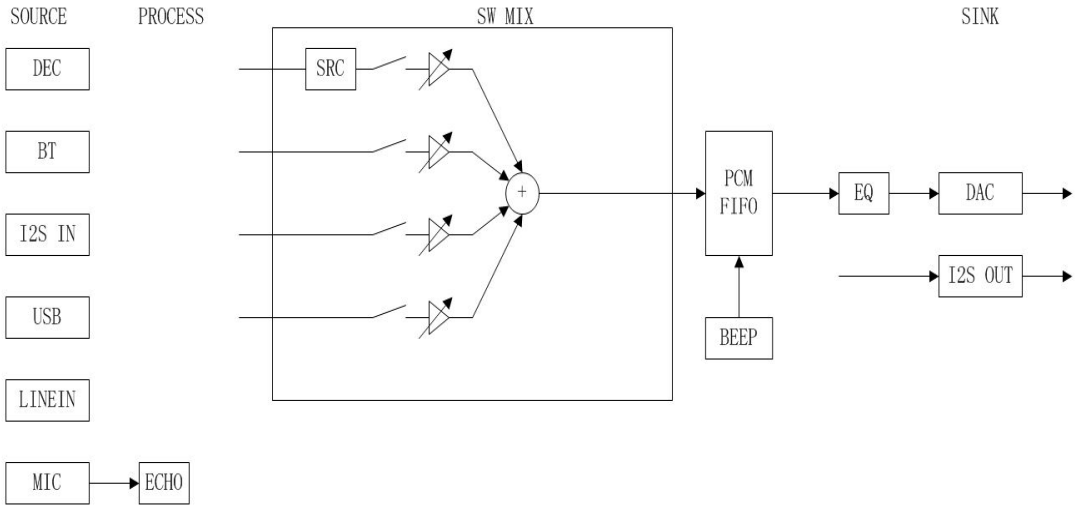
isAndDacOut – 是否同时输出到 DAC

如果要使用外接 ClassD 功放，时钟需要使用 Normal Mode 即 12.288M 的时钟。此时函数为：

void AudioStaOutSetChannel(uint32_t SampleRate);

9.4 Mixer 模块

Mixer 模块是用来进行数字音源的混合处理，借用了硬件 mix 的思想，所以称为 Mixer 模块。Mixer 模块的示意图如下：



Mixer 模块支持 4 路数字音源的输入，除了第一路支持 9 种采样率之外，其余 3 路只能使用 44.1KHz 的采用率。另外，第一路也支持采样率转换（SRC），可以将多种采样率音源转换为 44.1KHz。

Mixer 模块的主要函数为：

void MixerConfigFormat(uint8_t SourceID, uint16_t SampleRate, uint8_t PcmFormat)

函数说明

配置某一路输入源

参数说明

SourceID – 输入源(0 – 3)

SampRate – 采样率，只有第一路可以设置 9 种采样率，其他必须为 44.1K

PcmFormat – 单声道、双声道模式

void MixerSetData(uint8_t SourceID, void* PcmBuf, uint16_t SampleCnt);

函数说明

设置某一路输入源 PCM 数据首地址，并且同时开始做 Mix 处理

参数说明

SourceID – 输入源 (0 - 3)

PcmBuf - PCM 缓冲区首地址

SampleCnt – 采样点数

bool MixerIsDone(uint8_t SourceID);

函数说明

检测某一路输入源是否完成 Mix 处理，在函数内部会同时进行 Mix 处理

参数说明

SourceID – 输入源 (0 – 3)

10 语音提示(SoundRemind)

10.1 语音提示函数

void SoundRemind(uint16_t SoundId);

注意：

1) 语音提示流程需要在 main task 中运行，否则会有重入问题；
2) 语音提示函数是一个阻塞 Task 的函数，不执行完成，不会返回。

10.2 语音提示流程图

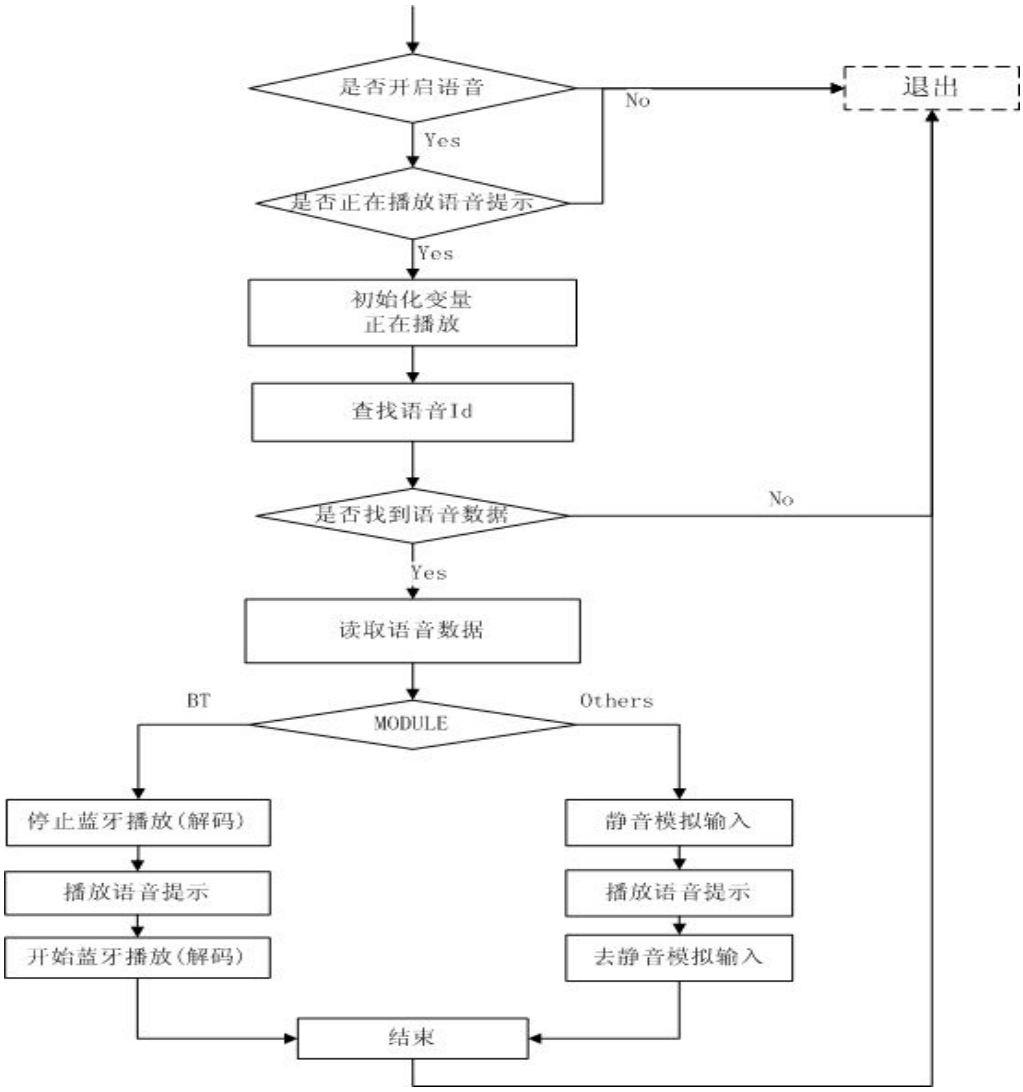


图 语音提示流程图

10.3 添加语音提示功能

使用工具 MVAssistant

11 断点记忆(Breakpoint)

11.1 功能

系统信息

1.当前模式

2.卡信息（SD or USB）
3 系统默认音量
4 提示音开关以及提示语言

播放信息

1. 播放音量
2. PlayMode
3. EQ 开关
4. 歌词显示开关（LRC）
5. 播放歌曲位置

FM 信息

1. 电台列表
2. FM 音量
3. 当前 FM 频段
4. 步长
5. 当前电台索引
6. 已保存电台个数

RTC 信息

11.2 记录位置

断点记忆信息记录在 Backup Register 中。当掉电后，信息将无法保持。如果要将信息记忆在 Flash 中，可以将宏 FUNC_NVM_TO_FLASH_EN 打开。

12 电源管理(Power Management)

12.1 概述

电源管理包含了以下三个功能：

1) 低电压的检测；
2) powerkey 检测；
3) 低功耗模式控制。

12.2 低电压的检测

低电压检测用于监测系统电池电量，当 LDOIN 上电压小于等于 3.3V 时，系统将会 Power Down。低电压检测分为两种情况：

1) 开机检测到低电压，那么系统将直接关机；
2) 在使用过程中(如播歌)检测到低电压，那么将通过发送消息，在当前模式下保存信息后切换到关机模式下进行关机

实现函数：

PowerMonitorInit()用于初始化和开机电压检测。在 main()中调用

PowerMonitor()用于系统运行过程中周期性低电压检测。在 main()中 task 循环中调用

12.3 powerkey 检测

powerkey 可以配置成三种模式

硬开关模式 (POWERKEY_MODE_SLIDE_SWITCH)

软开关模式 (POWERKEY_MODE_PUSH_BUTTON)

bypass 模式 (POWERKEY_MODE_BYPASS)

配置的初始化函数：

void SysPowerKeyInit(uint8_t PowerKeyMode, uint16_t SwitchOnTime)

powerkey 关机检测

powerkey 关机检测函数为 SystemPowerOffDetect()，在 timer1 中断里调用

在硬开关模式下，检测到 powerkey 后直接进行关机

在软开发模式下，检测到 powerkey 后将通过发送消息，在当前模式下保存信息后切换到关机模式下进行关机。

12.4 低功耗模式控制

Powerdown 模式控制，即关闭所有的数字电部分，功耗降到最低。

函数为：SystemPowerOffControl()

当检测到 powerkey 关机(软开关)或系统运行过程中检测到低电压时，将通过发送消息切换到该模式下处理关机流程。在模式下，只有 RTC 和 powerkey 才能唤醒

Standby 模式(deep sleep)控制，关闭大部分数字电，保留 GPIO 口的检测，功耗略高于 PowerDown 模式

函数为：SystemStandbyControl()

长按 ADC key POWER 键进入该模式，处理 standby 流程。该模式下 RTC、powerkey，以及部分 GPIO(如红外，详见 wakeup.h)可以唤醒系统

13 Memory Map

13.1 RAM 空间分配

根据硬件部分的设计，O18 将 128KB RAM 空间分配为以下三种

X-MEM

0x20000000 - 0x2000FFFF

V-MEM

0x20010000 – 0x20017FF

P-MEM

0x20018000 - 0x2001FFF

如图：

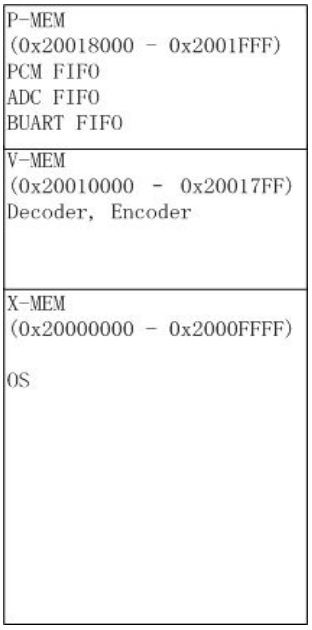


图 RAM 空间划分

V-MEM 主要用于 Decoder 与 Encoder，P-MEM 主要用于 PCM FIFO，ADC FIFO， BUART FIFO。这里需要注意的是，PCM FIFO，ADC FIFO， BUART FIFO 这三个 FIFO 必须要定义在 P-MEM 中。这是受硬件设计的约束导致的。

V-MEM 与 P-MEM 详细分配如下：

Decoder Mem Map

start: VMEM_ADDR , size: 28K

Encoder Mem Map

start: VMEM_ADDR+1K, size:17K

Recoder Buf

start: VMEM_ADDR+18K, size:19K (占用 P-MEM)

PCM FIFO

start:PMEM_ADDR+5K, size: 16K

ADC FIFO

start:PMEM_ADDR+21K, size: 4K

BUART FIFO

start: PMEM_ADDR+25K, size:7K

13.2 X-MEM 使用情况

X-MEM 是由 OS 来分配的，总大小为 64K。这里简单介绍几种占用 X-MEM 较大的模块与应用。用户可以根据需要来增加或者减少各功能模块。

功能	占用 MEM 大小(KB)	备注
Bluetooth SPP	4.3	
Bluetooth Adv AVRCP	1.5	
MIC ECHO	7	不需要 Mic 情况下，可去掉
SRA	4	软件微调

注：目前 SDK 默认功能有 HFP, A2DP, AVRCP, SRA, Advanced AVRCP, MIC ECHO，还剩余 RAM 空间为 2.7KB

Version	History	Description	Author
1.0	2014-8-28	Init Version	Halley