Rockchip Linux Partybox 应用开发指南

文档标识：RK-KF-YF-574

发布版本：V1.0.0

日期：2024-08-30

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

版权所有© 2024 瑞芯微电子股份有限公司

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文主要描述RK3308 Linux Partybox App的基本框图和基础的使用方法，旨在帮忙开发者快速了解并开发相应的App。

产品版本

芯片名称	内核版本
RK3308B/RK3308H	5.10

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	ZDM	2024-08-30	初始版本

目录

1. 应用简介

此应用适合音箱类的产品开发(例如Partybox)，也适合基于音乐的频谱分析产品开发，例如车位氛围灯等。

rkpartybox应用是基于消息队列和事件驱动，各任务是解耦的，主要框架上没有延时(delay)写法；

主任务作为app应用主要模块，其他LCD， LED显示， Rockit， 蓝牙设备，存储模块，按键， USB等可插拔设备都抽象为输入/输出(显示)设备。 (见第2章节图示)

提供的主要功能：

- 支持音源播放，例如BT， USB， UAC， AUX音源[外部&板载]。
- 支持MIC， Guitar， 和场景mic。
- 支持人声/背景声/吉他声AI消除， AI男女声识别。
- 支持室内外，左右声道，横竖放等AI自动识别。
- 支持防啸叫，混响，音效增强等算法。
- 支持LCD UI显示和调整，频谱显示，灯效显示等。
- 按键支持ADC Key， SAR旋钮(后面也称作SARA)，编码器和GPIO。
- 支持背景音源(BT，USB，UAC，AUX等)自动/手动切换。
- 支持多项目硬件配置，多厂商蓝牙串口协议。

2. 应用框图

从应用整体层次来看，主要分为App Tasks， Modules， Vendor， Interface Level和Hardware Level构成，如下图：

2.1 App Tasks

这里包含整个应用需要的各个任务, 这里说明各任务功能和任务之间控制流数据流转。

如上APP框图，整个系统是解耦的，各部分主要任务(线程)有单独的Task来处理。 整体应用是基于消息队列，主线程(后都称为Main Task)接收各子线程消息，并做处理，然后基于实际情况，分发消息到子线程中。

例如： 用户按下"音量+"后， KeyScan Task获取到按键信息，并通过Event Queue事件消息，发送按键通知到Main Task. Main Task收到按键消息后，发送指令到LED Task， 显示音量变化灯效；并发送指令到LVGL Task， LCD图形界面需要更新音量值；然后，还需要发送指令给Rockit Task， 改变实际输出音量；最后，假设有保存音量需求，主任务执行数据保存接口，存储数据到Storage模块上。

由于代码是基于消息队列的，所以各线程风格基本一致，主体结构都是基于Posix接口收发消息，再基于相应的消息事件处理，具体见第三章节<<源码说明>>。

由于程序主体框架是基于消息队列的，所以这里Main Task需要考虑实时性。 如果Main Task中使用delay函数，会导致阻塞其他线程的消息处理。 因此，不建议在Main Task中使用delay函数。 其他各个主要子线程也应尽量少直接使用delay函数。 具体的解决办法参考下`maintask_timer_fd_process()`处理。 以后可以进一步更新为基于消息队列的Timer机制。

2.1.1 Main Task

文件： `main.c`，主要函数： `maintask_read_event()`

此任务是应用主线程。 主要用来接收子线程事件，并处理和分发指令到其他线程中。

这里， Main Task相当于一个事件中转处理中心，处理的实际上是控制流数据。 而音频播放和采集的数据流，实际上目前是通过Rockit Task直接处理的。 (控制流和数据流分开，并且数据流不会因为消息中转而花费时间)

2.1.2 Rockit Task

文件： `pbox_rockit.c`，主要函数： `pbox_rockit_server()`

此任务是音频处理主要线程。 这个线程通过调用Rockit接口，完成大部分的音频数据流处理。

注意: 音频数据流大部分是Rockit直接处理的, 也就是除控制流数据外, 音频设备和驱动等数据流会直接和Rockit模块交互。

2.1.3 BT Task

文件： `pbox_soc_bt.c`，主要函数： `btsoc_sink_server()`
或者文件： `rk_btsink.c`，主要代码： `btsink_server()`

这里可能是MCU-BT(`pbox_soc_bt.c`)，也可能是HCI-BT(`rk_btsink.c`)。MCU-BT通常由MCU蓝牙芯片完成Host蓝牙协议栈数据处理，一般通过UART/I2C和RK3308通信，音频数据通过I2S/PCM接口传送到RK3308。（此处RK3308代指RK3308系列芯片，包括RK3308B，RK3308H等型号，以实际应用为准，后同）

而HCI-BT， Host蓝牙协议栈是跑在RK3308主控上，并且主控直接完成音频数据解析和播放等处理的。

目前我们开发板是基于HCI-BT和Bluez蓝牙协议栈的，同时, 我们也准备好了配套MCUBT的软件方案。HCI-BT的数据收发处理在`btsink_server()`线程， MCU-BT在`btsoc_sink_server()`，开发者根据实际硬件情况，选择对应的蓝牙工作任务(Task)。

2.1.4 HotPlug Task

源文件： `pbox_hotplug.c`，主要函数： `pbox_hotplug_dev_server()`

最初，这个任务是用于检测热插拔事件，处理USB、UAC和麦克风的插拔事件，以及控制流消息。现在，还加入了重力感应处理。如果客户需要添加电池检测，也可以放这里处理。

所以，这里的HotPlug可以认为是广义的热插拔设备，即理解为基于IO文件接口的外设(普通文件也可以支持，具体方法见下文)。

2.1.5 KeyScan Task

源文件： `pbox_keyscan.c`，主要函数： `pbox_KeyEvent_server()`

这个任务用于扫描普通ADC Key， Sar ADC key(Knob)，编码器， IO按键等。 按键事件扫描到后，对应消息会发送主线程处理函数接口处理(可参考应用代码按键映射章节)。

2.1.6 LED Task

源文件： `pbox_light_effect.c`，主要函数： `pbox_light_effect_server()`

这个任务主要是用来显示各种灯效，包括播放，暂停，音量调大小和频谱等灯效。

2.1.7 LVGL Task

源文件： `pbox_lvgl.c`，主要函数： `pbox_touchLCD_server()`

这个任务主要用来做LCD UI显示，包括频谱显示等。

2.2 Vendor

目前主要是各个Vendor私有的MCU蓝牙协议，和对应的协议解析，封装。 可进一步参考6.2章节。
注意： App Tasks中的各种参数计量单位可能与客户协议不一致，建议在此模块中进行统一的单位转换。

例如，客户协议音量是[0-31]， 而我们Main task和Rockit Task主要是以db来做单位计量。所以，音量需要在此模块做转换。

```
void rkdemo_btsoc_notify_master_volume(uint32_t opcode, uint8_t *buff,
int32_t len) {
    float volume;
    ...
    //covert to rockchip standard volume db(max: 0db)
    volume = hw_main_gain(buff[0])/10;
    rkdemoNotifyFuncs->notify_master_volume(opcode, volume);
}
```

2.3 Modules

此模块包含了应用需要的各个组件，如UAC，UART，存储，Audio(Alsa)，OS和Log等各个模块，并且API接口设计都设计为可重入的。

OS组件是我们封装过的系统调用。 里面包含List，File，GPIO接口，线程管理，Time，信号量和锁等相关API。推荐开发者使用这些API接口。

例如线程管理中，通过封装好的API， 可以轻松指定线程名称，优先级，堆栈大小，保证线程正确创建，并友好的销毁函数。

以上这些模块，甚至整个partybox应用，系统调用部分，都是基于Posix接口， 并且未使用复杂的Posix接口。因此这些代码可以方便的移植到其他操作系统上， 只要目标操作系统支持IO文件接口。

2.4 Interface Level

主要目的是隔离[App Tasks，Vendor]和Hardware层；转换上层和下层的代码定义；或者封装和硬件相关的应用代码。

Hardware层代码如果需要引用上层代码定义(不推荐)，也可以通过这个Interdface层，获取相应参数。

2.5 Hardware Level

硬件层， 主要包含硬件相关的项目配置，例如背景音源(BT，USB，UAC，AUX， SPDIF...)支持情况，对应优先级， MIC和Guitar个数，相应声卡参数等配置。

由于这是硬件配置层，因此这些配置仅供上层读取，不建议参数写入。

3. 源码说明

```
├── app                    //App主要代码，见2.1章节。
│   ├── main.c
│   ├── music_ui           //LVGL GUI接口库封装，和应用相关绘图。
│   ├── pbox_xxx.c         //App Tasks主要相关代码
└── vendor                 //Vendor， 见2.2章节。
│   └── rkdemo             //蓝牙通信协议，见2.2章节。开发者可创建同级私有协议目录
├── modules                 //modules， 见2.3章节。
│   ├── audio              //音频(Alsa)子模块
│   ├── os                 //OS子模块
│   ├── storage            //存储子模块
│   ├── uac                //uac子模块
│   └── userial            //串口子模块
├── utils
│   └── log                //log子模块
├── interface              //接口层，见2.4章节。
├── hal                    //Hardware level相关代码，见2.5章节
│   ├── include            //向上层提供的接口
│   ├── rockchip_coreboard //核心板demo.
│   └── rockchip_evb       //evb板demo， 开发者可创建同级私有硬件目录
├── include                //这里主要是lib库需要的头文件
├── lib64                  //lib库存放目录。
├── bins                   //partybox应用相关的bin， conf， pcm， json， sh脚本等。
├── CMakeLists.txt         //CMake编译脚本
├── doc                    //文档
├── test
│   ├── inputkey.c         //按键测试小工具
│   └── testlib            //生成静态库例子
```

4. 编译环境

4.1 SDK整体环境

整体与Buildroot的编译指令已经在QuickStart文档中详细说明。为了更好地说明rkpartybox应用与Buildroot编译环境的关系，特在此进行简要说明。

```
docs/cn/RK3308/Rockchip_RK3308_Partybox_Quick_Start_Linux_CN.pdf
```

首先在SDK目录执行`./build.sh lunch`，然后选择相应的项目：

```
rk3308_sdk$ ./build.sh lunch

############### Rockchip Linux SDK ###############

Manifest: rk3308_partybox_alpha_v0.0.1_20240422.xml
Version: linux-5.10-gen-rkr7" dest-branch="refs/tags/linux-5.10-gen-rkr7

...
Pick a defconfig:

1. rockchip_rk3308_evb_audio_v10_64bit_defconfig
2. rockchip_rk3308b_64bit_rkpartybox_coreboard_defconfig
3. rockchip_rk3308b_64bit_rkpartybox_defconfig
4. rockchip_rk3308bs_64bit_defconfig
Which would you like? [1]:
```

目前，Partybox 有两个示例项目可供选择：

- 带屏幕并支持HCI蓝牙：`rockchip_rk3308b_64bit_rkpartybox_defconfig`, 选择`3`。
- 不带屏幕并使用外部MCU蓝牙：`rockchip_rk3308b_64bit_rkpartybox_coreboard_defconfig`, 选择`2`。

选择后，整个SDK编译环境就设置好了(直接执行`./build.sh all-release`可以编出全部固件)。

4.2 buildroot环境

上述选项确定后，可以通过以下命令查看partybox项目的Buildroot配置（这里以`3. rockchip_rk3308b_64bit_rkpartybox_defconfig`为例：

```
rk3308_sdk$ vi device/rockchip/rk3308/rockchip_rk3308b_64bit_rkpartybox_defconfig
  1 RK_BUILDROOT_BASE_CFG="rk3308_b_rkpartybox" ...
```

从以上第一行代码可以看到， buildroot的默认配置是`rk3308_b_rkpartybox`。

Buildroot配置文件路径为：
`buildroot/configs/rockchip_rk3308_b_rkpartybox_defconfig`

执行source指令配置好buildroot环境：

```
rk3308_sdk$ cd buildroot/
rk3308_sdk/buildroot$ source envsetup.sh
...
Pick a board:
...
14. rockchip_rk3308_b_rkpartybox
15. rockchip_rk3308_b_rkpartybox_nodisp
```

也可以直接执行`source envsetup.sh rockchip_rk3308_b_rkpartybox`。

这样buildroot环境也配置好了。

4.3 应用编译

```
cd buildroot
make rkpartybox-dirclean && make rkpartybox-rebuild
```

编译好的bin在`buildroot/output/rockchip_rk3308_b_rkpartybox/target/usr/bin/rkpartybox`

调试可以直接推送到`/data`目录，并重启生效。

```
adb push
sdkto/buildroot/output/rockchip_rk3308_b_rkpartybox/target/usr/bin/rkpartybox
/data/
adb shell chmod 777 /data/rkpartybox
```

4.4 Buildroot其他相关

参考以下文档：

`docs/cn/RK3308/Rockchip_RK3308_Partybox_Quick_Start_Linux_CN.pdf`
`docs/cn/Linux/System/Rockchip_Developer_Guide_Buildroot_CN.pdf`

5. 功能配置

rkpartybox应用的功能配置分为两部分，一部分位于Buildroot中，另一部分位于应用的硬件层代码中。

将配置放在Buildroot中的原因在于，这部分代码通常需要预编译，且代码量较大。为了优化代码空间，选择通过Buildroot进行配置。

而硬件层的配置则主要与主板硬件相关，通常不会显著增加编译后的代码空间占用。

5.1 buildroot功能配置

如前所述，预编译的配置更倾向于通过Buildroot进行设置。以 rk3308_b_rkpartybox 为例，以下是整个配置的说明：

- 项目配置：
  `buildroot/configs/rockchip_rk3308_b_rkpartybox_defconfig`
- rkpartybox应用配置：
  `buildroot/package/rockchip/rkpartybox/rkpartybox.mk`
- mk文件：
  `package/rockchip/rkpartybox/rkpartybox.mk`
- makefile/Cmake：
  mk编译脚本会继续调用`app/rkpartybox/CMakeLists.txt`文件，从而最终影响代码rkpartybox应用代码的宏定义。

buildroot目录，执行`make menuconfig`，可配置rkpartybox功能，路径：

```
Prompt: RKPARTYBOX demo                                          |
  |   Location:                                                    |
  |     -> Target packages                                          |
  |       -> Hardware Platforms                                     |
  |         -> Rockchip Platform (BR2_PACKAGE_ROCKCHIP [=y])
```

功能选项

```
        [*] RKPARTYBOX demo
        [ ] RKPARTYBOX core board only
        [*] RKPARTYBOX LCD display
        [*] RKPARTYBOX led effect
```

`RKPARTYBOX demo`指partybox demo应用，选中这个后，才有后面其他选项。
`RKPARTYBOX core board only`表示核心板，一般选择核心板，也会选择rk3308+btmcu方案。
`RKPARTYBOX LCD display` 表示是否有屏幕显示。
`RKPARTYBOX led effect` 表示是否有LED灯效显示。

选择好功能后，最后执行`make update-defconfig`保存buildroot配置。

```
rk3308_sdk/buildroot$ make update-defconfig
...
Updating defconfig: configs/rockchip_rk3308_b_rkpartybox_defconfig
...
Done updating configs/rockchip_rk3308_b_rkpartybox_defconfig.
```

以上配置最终会保存为项目的实际配置文件。

调试时，如果不想修改buildroot配置，可以直接改代码，方法见宏定义功能调试技巧。

以下介绍是buildroot配置方法：

5.1.1 屏幕配置

以上buildroot menuconfig中，选择RKPARTYBOX LCD display，保存退出，更新buildroot配置后，在buildroot目录执行`clear && make rkpartybox-dirclean && make rkpartybox-rebuild`重新编译rkpartybox即可。

5.1.2 LED灯效

同理，选择`RKPARTYBOX led effect`后，执行相应步骤即可。

5.2 Hardware配置

代码量较小且与硬件相关的配置都集中在此处。对应的配置位于应用源代码的 hal/ 目录中。要修改这些配置时，可通过增加或修改对应板级文件 board_audio_hw.h 完成。
原来两个板子，如下：

```
rk3308_sdk/app/rkpartybox/hal$ tree -L 2
.
├── hal_hw.c
├── hal_input.c
├── include
│   └── hal_partybox.h
├── rockchip_coreboard    //核心板
│   └── board_audio_hw.h
└── rockchip_evb          //evb板
    └── board_audio_hw.h
```

新增一个板级配置：

```
rk3308_sdk/app/rkpartybox/hal$ tree -L 2
...
├── rockchip_evb
│   └── board_audio_hw.h
└── vendor_xxx
    └── board_audio_hw.h
```

`vendor_xxx`目录即新增的板级配置，后面此节的配置默认都在board_audio_hw.h.

Cmake编译脚本需要同步修改：

```
diff --git a/CMakeLists.txt b/CMakeLists.txt
index 72fe0e9..5a132c1 100644
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -28,7 +28,7 @@ include_directories(${DBUS_INCLUDE_DIRS})
 if(RK3308_PBOX_CORE_BOARD)
 include_directories(${PROJECT_SOURCE_DIR}/hal/rockchip_coreboard/)
 else()
-include_directories(${PROJECT_SOURCE_DIR}/hal/rockchip_evb/)
+include_directories(${PROJECT_SOURCE_DIR}/hal/vendor_xxx/)
 endif()
```

5.2.1 输入源配置

目前SDK支持的输入源：

```
typedef enum {
    SRC_CHIP_USB,
    SRC_CHIP_BT,
    SRC_CHIP_UAC,
    SRC_EXT_BT,
    SRC_EXT_USB,
    SRC_EXT_AUX,
    SRC_NUM
} input_source_t;

#define MASK_SRC_CHIP_USB   (1 << SRC_CHIP_USB)
#define MASK_SRC_CHIP_BT    (1 << SRC_CHIP_BT)
#define MASK_SRC_CHIP_UAC   (1 << SRC_CHIP_UAC)
#define MASK_SRC_EXT_BT     (1 << SRC_EXT_BT)
#define MASK_SRC_EXT_USB    (1 << SRC_EXT_USB)
#define MASK_SRC_EXT_AUX    (1 << SRC_EXT_AUX)
```

配置支持的输入源：

```
    #define HW_SUPPORT_SRCS
(MASK_SRC_CHIP_USB|MASK_SRC_CHIP_BT|MASK_SRC_CHIP_UAC)
```

输入源优先级配置：

```
    #define FAVOR_SRC_ORDER {SRC_CHIP_BT, SRC_CHIP_USB, SRC_CHIP_UAC,
SRC_EXT_BT, SRC_EXT_USB, SRC_EXT_AUX}
```

以上优先级配置：`SRC_CHIP_BT > SRC_CHIP_USB > SRC_CHIP_UAC > SRC_EXT_BT > SRC_EXT_USB > SRC_EXT_AUX`

因为配置只支持`SRC_CHIP_USB， SRC_CHIP_BT， SRC_CHIP_UAC`， 所以实际优先级是：`SRC_CHIP_BT > SRC_CHIP_USB > SRC_CHIP_UAC`

5.2.2 话筒/场景MIC和spk配置

实际mic和spk在linux系统中都会映射成声卡，所以需要配置实际硬件的声卡名称，如下：

```
#define AUDIO_CARD_SPK_CODEC    "hw:0,0"
#define AUDIO_CARD_CHIP_GUITAR  NULL
#define AUDIO_CARD_CHIP_KALAOK  "mic"
#define AUDIO_CARD_CHIP_SCENE   "scene"
#define AUDIO_CARD_RKCHIP_BT    "hw:7,1,0"
#define AUDIO_CARD_RKCHIP_UAC   "hw:3,0"
```

以上， NULL代表实际无此声卡，或者配置项目不支持此输入源。

以"hw:"开头的是Linux系统生成的声卡。 而其他例如"mic"和"scence"声卡是alsalib和rockit拆分出来的声卡。

rockit相关的声卡配置：

```
#define SPK_CODEC_CHANNEL       2

#define KALAOK_REC_CHANNEL      4
#define KALAOK_POOR_COUNT       (AUDIO_CARD_RKCHIP_UAC?1:0)
#define KALAOK_REF_LAYOUT       0x03
#define KALAOK_REC_LAYOUT       0x04
#define KALAOK_REF_CHN_LAYOUT   0x0f
#define KALAOK_REF_HARD_MODE    ECHO_REF_MODE_HARD_COMBO
#define KALAOK_REC_SAMPLE_RATE  48000

#define SCENE_REC_CHANNEL       4
#define SCENE_REF_LAYOUT        0x03
#define SCENE_REC_LAYOUT        0x0c
#define SCENE_REF_HARD_MODE     ECHO_REF_MODE_SOFT
#define SCENE_REC_SAMPLE_RATE   48000
```

以上，`XXX_CHANNEL`代表声道数量；`XXX_SAMPLE_RATE`代表采样率； `XXX_REF_HARD_MODE`是mic/guitar/场景mic的回采设置，主要用来回声消除，或者混响增强的。
其他参数含义参考Rockit音频调试指南。

5.2.3 sar key配置

在evb对应的Rolling Board中，旋钮用于调节mic volume, reverb等参数，这些功能通过SAR ADC实现。

以下代码中，`USE_SARA_ADC_KEY`用于指示是否启用SARA_ADC key功能；`MIN_SARA_ADC`表示旋钮调到最小位置时的ADC key值；`MAX_SARA_ADC`则表示旋钮调到最大位置时的ADC key值。

```
#defne MAX_SARA_ADC 1023
#define MIN_SARA_ADC 0
#define USE_SARA_ADC_KEY 0
```

6. 调试说明

6.1 rkpartybox bin和库调试

由于大部分客户使用的小容量Flash存储（≤128MB），因此rootfs文件系统默认采用squashfs格式。配置如下：

```
rk3310_sdk$ vi
device/rockchip/rk3308/rockchip_rk3308b_64bit_rkpartybox_defconfig
  1 RK_BUILDROOT_BASE_CFG="rk3308_b_rkpartybox"
  2 RK_ROOTFS_TYPE="squashfs"
  ...
  8 RK_KERNEL_DTS_NAME="rk3308-partybox-ext-rolling-v10"
  ...
```

由于采用了squashfs文件系统，客户在调试时，除了/tmp、/oem和/data目录外，其他目录都是只读的。针对rkpartybox应用，我们已配置了`bin`和`lib`s环境变量，具体如下：

```
rk3308_sdk/app/rkpartybox$ vi bins/partybox_app_evb.sh
  1 #! /bin/sh
  2
  3 export LD_LIBRARY_PATH=/data/:$LD_LIBRARY_PATH
  4 export PATH=/data:$PATH
  ...
```

所以rkpartybox bin和库可以推送到/data/目录， bin推送后，需要另外加上可执行权限：

```
chmod +x /data/rkpartybox
或者
chmod 775 /data/rkpartybox
```

然后重启应用即可(`killall rkpartybox && /oem/partybox_app.sh`)。

有时我们可能需要替换`/oem/`目录下的文件，而当前`/oem/`目录中的文件都是一些软链接。因此，在调试时，可以将这些软链接替换为实际的文件。

```
...
lrwxrwxrwx 1 root root    27 Jan  1 00:00 eq_drc_player.bin ->
/etc/pbox/eq_drc_player.bin
lrwxrwxrwx 1 root root    29 Jan  1 00:00 eq_drc_recorder.bin ->
/etc/pbox/eq_drc_recorder.bin
lrwxrwxrwx 1 root root    24 Jan  1 00:00 partybox_app.sh ->
/usr/bin/partybox_app.sh
lrwxrwxrwx 1 root root    25 Jan  1 00:00 partybox_play.sh ->
/usr/bin/partybox_play.sh
...
```

例如，我们如果想修改`/oem/partybox_app.sh`， 则可以：

```
adb pull /oem/partybox_app.sh
start partybox_app.sh
#本地电脑修改文件内容。
adb push partybox_app.sh /oem/
chmod 777 /oem/partybox_app.sh
```

或者在设备上：

```
rm /oem/partybox_app.sh #删除软连接文件
cp /usr/bin/partybox_app.sh /oem/
vi /oem/partybox_app.sh
```

6.2 MCU蓝牙串口协议

考虑到不同客户使用的MCU蓝牙串口协议各异，我们实现了一套支持多厂商蓝牙串口协议的代码。代码在vendor目录，供开发参考。具体协议见MCUBT蓝牙协议。

```
app/rkpartybox/vendor/
├── include
│   └── bt_vendor_protol.h
└── rkdemo
    └── bt_vendor_protol_rkdemo.c
```

例程MCU蓝牙协议放在`bt_vendor_protol_rkdemo.c`文件。
开发者在开发自己私有的协议时，可以在vendor下(rkdemo同级目录)，新建一个目录，并实现相应协议。
编译脚本需要修改：

```
aux_source_directory(./vendor/rkdemo SRCS) #rkdemo改成自己的目录名称。
```

以下是运行机制的说明：
在pbox_soc_bt.c文件中，btsoc_sink_server函数会注册相关接口：

```
static void *btsoc_sink_server(void *arg) {
    ...
    vendor_data_recv_handler_t  uart_vendor_data_recv_hander =
vendor_get_data_recv_func();
    ...
    btsoc_register_vendor_notify_func(pbox_socbt_get_notify_funcs());
```

以下分别说明各部分内容：

6.2.1 MCU蓝牙协议实现

`vendor_get_data_recv_func()`：获取蓝牙协议处理函数的入口。示例如下：

```
vendor_data_recv_handler_t vendor_get_data_recv_func(void) {
    return rkdemo_uart_data_recv_handler;
}
```

`rkdemo_uart_data_recv_handler()`：实现当前蓝牙串口协议的数据接收和处理。如下：

```
void rkdemo_uart_data_recv_handler(int fd, struct uart_data_recv_class*
pMachine) {
    unsigned char buf[BUF_SIZE];
    ...
    switch (pMachine->current_state) {
        case READ_INIT:     //协议解析状态机入口
            ...
            pMachine->current_state = READ_HEADER;
            break;
        case READ_HEADER:   //解析协议头字节
            ...
                if (buf[pMachine->index] == 0xCC) { //已获取到协议数据头"0xCC"
                    ...
                    pMachine->current_state = READ_LENGTH;
                }
            ...
            break;
        case READ_LENGTH:   //获取数据长度
            ...
            break;
        case READ_DATA:     //获取完整数据
                ...
                if (is_check_sum_ok(buf, pMachine->index)) { //checksum检查
                    process_data(buf, pMachine->index);      //检查到完整数据
帧，进一步处理协议数据*/
                    pMachine->current_state = READ_INIT;    //状态机reset， 下
一步重新解析
                }
                ...
            break;
        ...
    }
}
```

6.2.2 协议数据上报

以下代码用于注册上报接口。需要注意的是，`pbox_socbt_get_notify_funcs()`函数具有通用性，可以适用于不同的协议实现。

```
    btsoc_register_vendor_notify_func(pbox_socbt_get_notify_funcs());
```

以上函数是在协议文件(`bt_vendor_protol_rkdemo.c`)中实现的。

```
int btsoc_register_vendor_notify_func(const NotifyFuncs_t* notify_funcs) {
    rkdemoNotifyFuncs = notify_funcs;
    return 0;
}
```

具体的上报接口如下：

```
const static NotifyFuncs_t notify_funcs = {
    .notify_dsp_version = soc2pbox_notify_dsp_version,
    .notify_master_volume = soc2pbox_notify_master_volume,
    .notify_placement = soc2pbox_notify_placement,
    ...
};
const NotifyFuncs_t* pbox_socbt_get_notify_funcs(void) {
    return &notify_funcs;
}
```

以音量为例，实际上报调用过程：

```
void rkdemo_btsoc_notify_master_volume(uint32_t opcode, uint8_t *buff,
int32_t len) {
    float volume;
    ...
    rkdemoNotifyFuncs->notify_master_volume(opcode, volume);
}
```

6.3 按键功能修改

6.3.1 kernel部分

按键功能的基础定义通常在DTS文件中。以下是一个示例说明：

- gpio按键：

```
  gpio-keys {
    compatible = "gpio-keys";
    ...
    play {
      gpios = <&gpio2 RK_PB6 GPIO_ACTIVE_LOW>;
      linux,code = <KEY_PLAY>;
      label = "GPIO Play Pause";
      debounce-interval = <100>;
    };
    light {
      gpios = <&gpio1 RK_PC7 GPIO_ACTIVE_LOW>;
      linux,code = <KEY_LIGHTS_TOGGLE>;
      label = "GPIO Light Mode";
      debounce-interval = <100>;
    };
    ...
  };
```

- GPIO编码器：

```
rotary {
  compatible = "rotary-encoder";
  pinctrl-names = "default";
  pinctrl-0 = <&rotary_gpio>;
  gpios = <&gpio2 RK_PB3 GPIO_ACTIVE_LOW>,
    <&gpio2 RK_PB4 GPIO_ACTIVE_LOW>;
  linux,axis = <0>; /* REL_X */
  rotary-encoder,relative-axis;
  status = "okay";
};
```

- ADC key

```
adc-keys {
  compatible = "adc-keys";
  io-channels = <&saradc 1>;
  io-channel-names = "buttons";
  poll-interval = <100>;
  keyup-threshold-microvolt = <1800000>;
  ...
  menu-key {
    linux,code = <KEY_PLAY>;
    label = "play";
    press-threshold-microvolt = <624000>;
  };

  vol-down-key {
    linux,code = <KEY_VOLUMEDOWN>;
    label = "volume down";
    press-threshold-microvolt = <300000>;
  };

  vol-up-key {
    linux,code = <KEY_VOLUMEUP>;
    label = "volume up";
    press-threshold-microvolt = <18000>;
  };
};
```

6.3.2 应用代码按键映射

- 内核到rkpartybox的映射

应用中，有一个内核到rkpartybox空间的按键映射：

```
static const key_pair_t KEY_TABLE[] = {
    /*kernel    user*/
    {373,       HKEY_MODE}, //KEY_MODE
    {207,       HKEY_PLAY}, //KEY_PLAY
    {209,       HKEY_GPIO_BOOST},
    {115,       HKEY_VOLUP}, //KEY_VOLUMEUP
    {114,       HKEY_VOLDOWN}, //HKEY_VOLDOWN
    {248,       HKEY_MIC1MUTE}, //KEY_MICMUTE
    {0x21e,     HKEY_GPIO_LIGHTS},
};
```

- 按键执行功能定义

目前，按键功能支持单击、双击、长按，以及通过ADC SAR旋钮和GPIO编码器的输入。具体的按键功能定义在最后一列。

```
const struct dot_key support_keys [] =
{
    /*key           keyb    press_type vaild comb    func                */
    /*短按*/
    {HKEY_PLAY,     0,      K_SHORT,       1, 0, pbox_app_key_set_playpause},
    {HKEY_VOLUP,    0,      K_SHORT,       1, 0,
pbox_app_key_set_volume_up},/*VOL_UP*/
    {HKEY_VOLDOWN,  0,      K_SHORT,       1, 0,
pbox_app_key_set_volume_down},/*VOL_DOWN*/
    {HKEY_MODE,     0,      K_SHORT,       1, 0,
pbox_app_key_switch_input_source},
    ...
    /*长按> 3s */
    {HKEY_PLAY,     0,      K_LONG,        1, 0, enter_long_playpause_mode},
    {HKEY_VOLDOWN,  0,      K_LONG,        1, 0,
long_volume_step_down},/*VOL_DOWN*/
    ...
    /*长按> 10s */
    {HKEY_MODE,     0,      K_VLONG,       1, 0, enter_recovery_mode},/*10s长
按进recovery*/
    /*双击*/
    {HKEY_PLAY,     0,      K_DQC,         1, 0, pbox_key_music_album_next},
    ...
    /*knob*/
    {HKEY_MIC1BASS, 0,      K_KNOB,        1, 0,
pbox_app_knob_set_mic1_bass},
    {HKEY_MIC1TREB, 0,      K_KNOB,        1, 0,
pbox_app_knob_set_mic1_treble},
    {HKEY_MIC2REVB, 0,      K_KNOB,        1, 0,
pbox_app_knob_set_mic2_reverb},
    {HKEY_MIC2_VOL, 0,      K_KNOB,        1, 0,
pbox_app_knob_set_mic2_volume},
    ...
};
```

6.4 音频调试

参考Rockit音频调试指南。

6.5 gdb调试

rkpartybox程序运行前，需要加载一些环境变量。 具体应用是从`/oem/partybox_app.sh`开始运行的。对应源代码： `bin/partybox_app_xxx.sh`(编译时，会根据实际项目，重命名为`partybox_app.sh`)

```
...
ulimit -c unlimited    #coredump环境变量
echo "/tmp/core-%p-%e" > /proc/sys/kernel/core_pattern    #如果coredump， 会产生coredump文档
rkpartybox
```

- 如果是单步gdb调试或者查堆栈，执行`gdb attach rkpartybox`，如果rkpartybox bin正在调试并放在/data/目录下，则`gdb attach /data/rkpartybox`
- 如果系统crash了，如上代码注释， partybox_app.sh脚本会在/tmp目录产生coredump文档，此时可以执行`gdb rkpartybox /tmp/core-xxx`，查看对应的coredump 信息。如下例子，core-xxx对应的实际文件名为`core-579-mic`。因此，最终指令是：`gdb rkpartybox /tmp/core-579-mic`。

```
root@rk3308b-buildroot:/# ls /tmp -al
...
-rw-------  1 root root 354500608 Jan  1 00:24 core-579-mic
...
```

同理，如果bin文件正调试并在/data/目录，则执行`gdb /data/rkpartybox /tmp/core-xxx`

gdb指令执行后，再执行`bt`指令查看函数栈调用线程信息，执行`thread appply all bt`查看所有堆栈信息。

如果crash信息是在librockit.so中，则可以参考Rockit音频调试指南进一步分析。如果需要更多符号定位信息时，则推送SDK `app/rkpartybox/lib64`目录中的`librockit.so.debug`文件到RK3308机器`/data/`目录并重启。(librockit.so.debug文件较大，请确保`/data/`目录剩余空间大于此文件大小)：

```
adb push path2sdk/app/rkpartybox/lib64/librockit.so.debug /data/librockit.so
adb reboot
```

如果在分析这些信息后仍无法解决问题，可以导出相应log/堆栈信息，通过Redmine或其他技术支持渠道进行反馈。

6.6 宏定义功能调试技巧

调试时，如果功能配置不想在buildroot中配置，可以在pbox_model.h中配置。

例如，以下配置关闭LCD：

```
#ifndef ENABLE_LCD_DISPLAY
#define ENABLE_LCD_DISPLAY 0
#endif
```

可以修改代码绕开buildroot：

```
#undef ENABLE_LCD_DISPLAY
#ifndef ENABLE_LCD_DISPLAY
#define ENABLE_LCD_DISPLAY 1
#endif
```

以上使用`#undef`语句关闭Buildroot等定义的LCD配置，然后根据调试目的，打开或者关闭`ENABLE_LCD_DISPLAY`。

注意：调试时可以这样配置功能选项，而正式更新代码时，建议回到Buildroot中配置。

7. 相关文档

此节介绍partybox其他相关的开发文档。涵盖Rockit音频调试文档，蓝牙协议文档，Rkstudio调试文档，防啸叫调试文档等。

7.1 Rockit音频调试指南

主要介绍 Partybox Audio相关流程，以及调试方法。相应音频库对应`librockit.so`。

`partybox app`应用也是基于此开发的，相应部分见Rockit Task

文档： `app/rkpartybox/doc/Rockchip_Developer_Guide_Partybox_Audio_CN.pdf`

7.2 MCUBT蓝牙协议

主要介绍 Partybox MCUBT的串口示例协议。通过UART/I2C接口，该协议用于RK3308芯片和MCUBT芯片之间的通信。

客户可以适配自己的蓝牙协议，适配方法见MCU蓝牙串口协议章节。

注意：如果是HCI蓝牙，是不需要这个协议的。因为这种情况下，Host蓝牙协议栈都在RK3308中运行。

文档: `app/rkpartybox/doc/Rockchip_partybox_mcubt_profile.xlsx`

7.3 RkStudio

RkStudio是图形化的音频效果调试工具。 客户可以基于此工具，调试需要的音效效果。

工具和相应文档放在ftp服务器，如下：

```
ftp://www.rockchip.com.cn
用户名：rkwifi
密码：Cng9280H8t
目录：/15-RK3308/RkStudioTool
```

7.4 防啸叫调试

主要介绍防啸叫调试。

文档： `app/rkpartybox/doc/Rockchip_Developer_Guide_Howling_Suppresion_Tuning.pdf`