RTL8762C 语音遥控器设计说明书

V1.2 by Realtek

2020/01/15

修订历史

日期	版本	修改
2018/06/15	V1.0	First Bee2 RCU spec for beta Bee2 RCU SDK
2018/09/12	V1.1	Bee2 RCU design spec for RCU SDK V1.0.2
2020/01/15	V1.2	Bee2 RCU design spec for RCU SDK V1.2.2

目录

图表

1 概述

本文主要介绍 RTL8762C 语音遥控器方案的软件相关技术参数和行为规范，包括广播包定义，待机与唤醒，配对过程，按键操作等行为规范和语音功能，用以指导遥控器的开发并追溯软件测试中遇到的问题。下文简称 RTL8762C 语音遥控器为 Bee2 RCU。软件支持两种模式：R_DEMO_RCU 和 H_DEMO_RCU，本文以 R_DEMO_RCU 为例进行详细说明。

2 功能特性

- *支持 BLE 4.2 (RTL8762C/RTL8752C)*
- *支持 BLE 5.0 (RTL8762C)*
- *支持自动配对*
- *支持蓝牙开关机*
- *支持语音输入与传输*
- *支持 20 个按键（5x4），具有键盘扫描模块，最多可扩展到 208 个按键（8x26）*
- *支持 LED 指示*
- *支持 IR 发送*
- *支持 IR 学习*
- *支持电量检测*
- *支持低电量保护*
- *支持 OTA 升级*
- *支持量产测试模式*
- *电源：1.8~3V (2 节干电池供电)*

3 GPIO 配置

3.1 System GPIO

P0_3 --- LOG
P1_0 --- SWDIO
P1_1 --- SWDCLK
P3_0 --- HCI UART TX
P3_1 --- HCI UART RX

3.2 Keyscan GPIO

P4_3 --- Keyscan ROW 0
P4_2 --- Keyscan ROW 1
P4_1 --- Keyscan ROW 2
P4_0 --- Keyscan ROW 3
P0_6 --- Keyscan ROW 4

P3_0 --- Keyscan COLUMN 0
P3_1 --- Keyscan COLUMN 1
P3_2 --- Keyscan COLUMN 2
P3_3 --- Keyscan COLUMN 3

3.3 Voice MIC GPIO

Digital MIC
P1_6 --- DMIC CLK
P1_7 --- DMIC DAT

Analog MIC
P2_6 --- AMIC_N
P2_7 --- AMIC_P

3.4 IR GPIO

P2_3 --- IR Transmit Pin
P2_2 --- IR Learn Pin

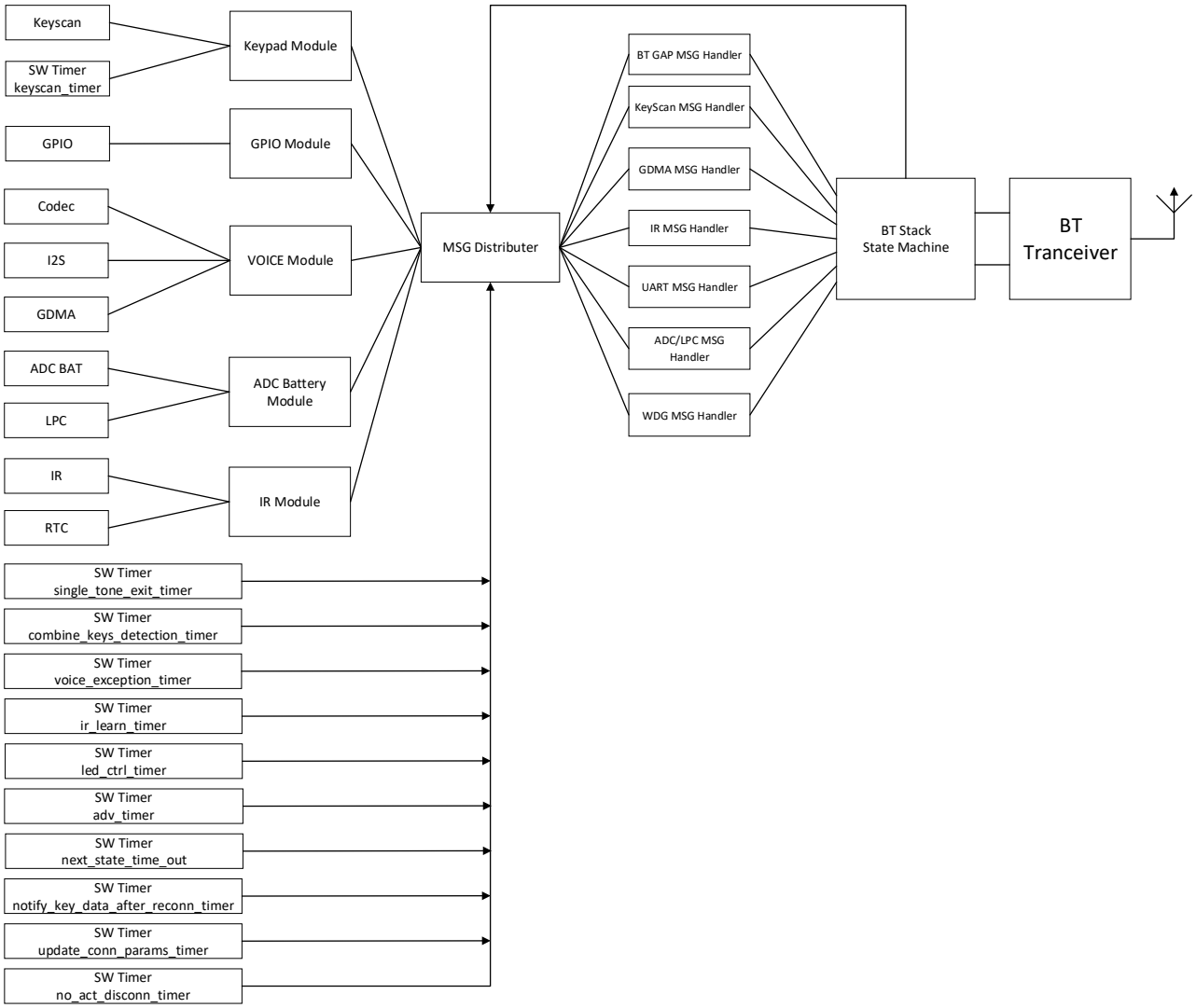
3.5 LED GPIO

P2_4 --- LED_1

3.6 MP Test GPIO

P3_2 --- MP Test Trigger Pin 1
P3_3 --- MP Test Trigger Pin 2
P3_0 --- MP Test UART TX
P3_1 --- MP Test UART RX

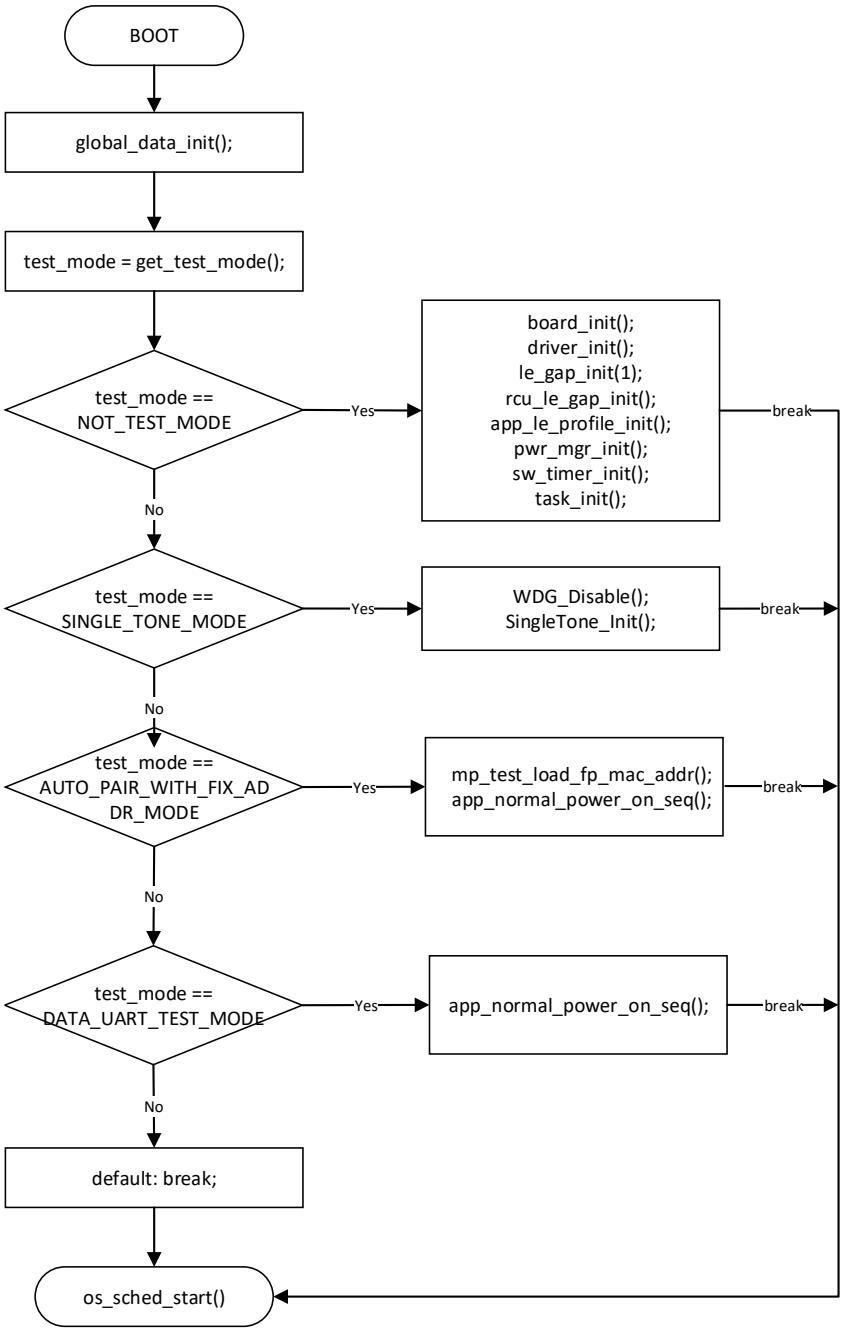
4 系统框图



图表 1 系统框图

5 初始化过程

5.1 系统启动流程

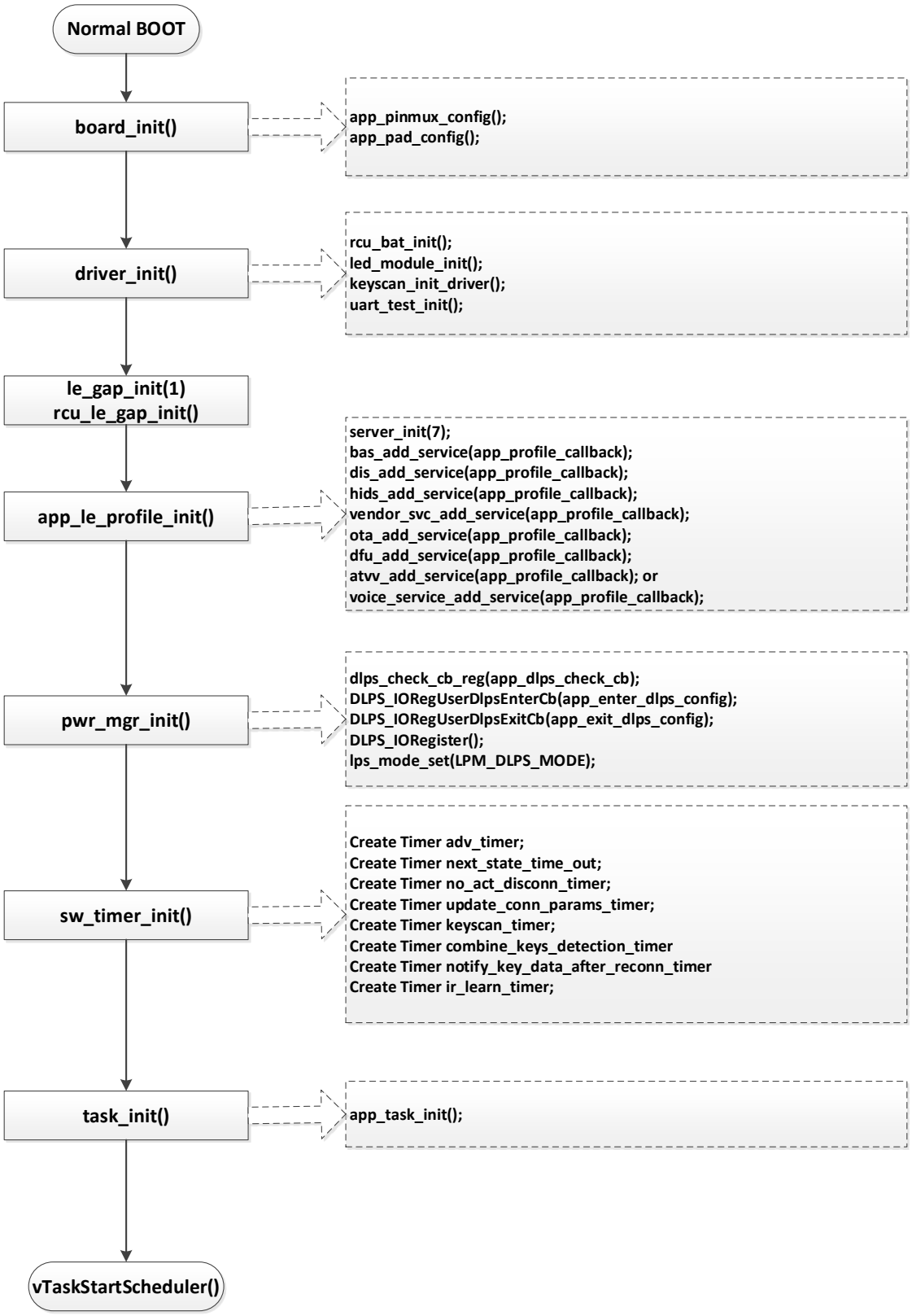


图表 2 系统启动流程图

系统模式

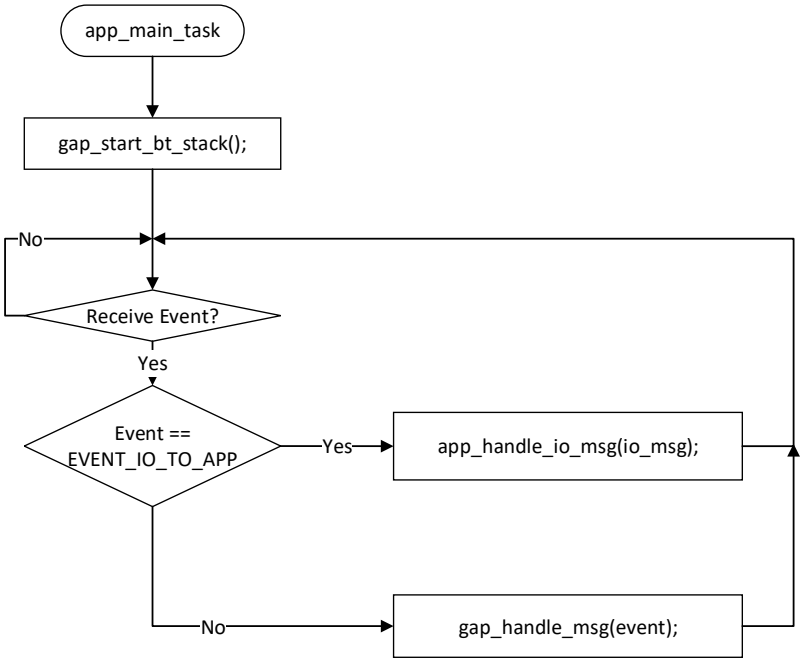
- Normal Mode
- Single Tone Mode
- Auto Pair with Fix Address Mode
- Data UART Test Mode

5.2 正常启动流程

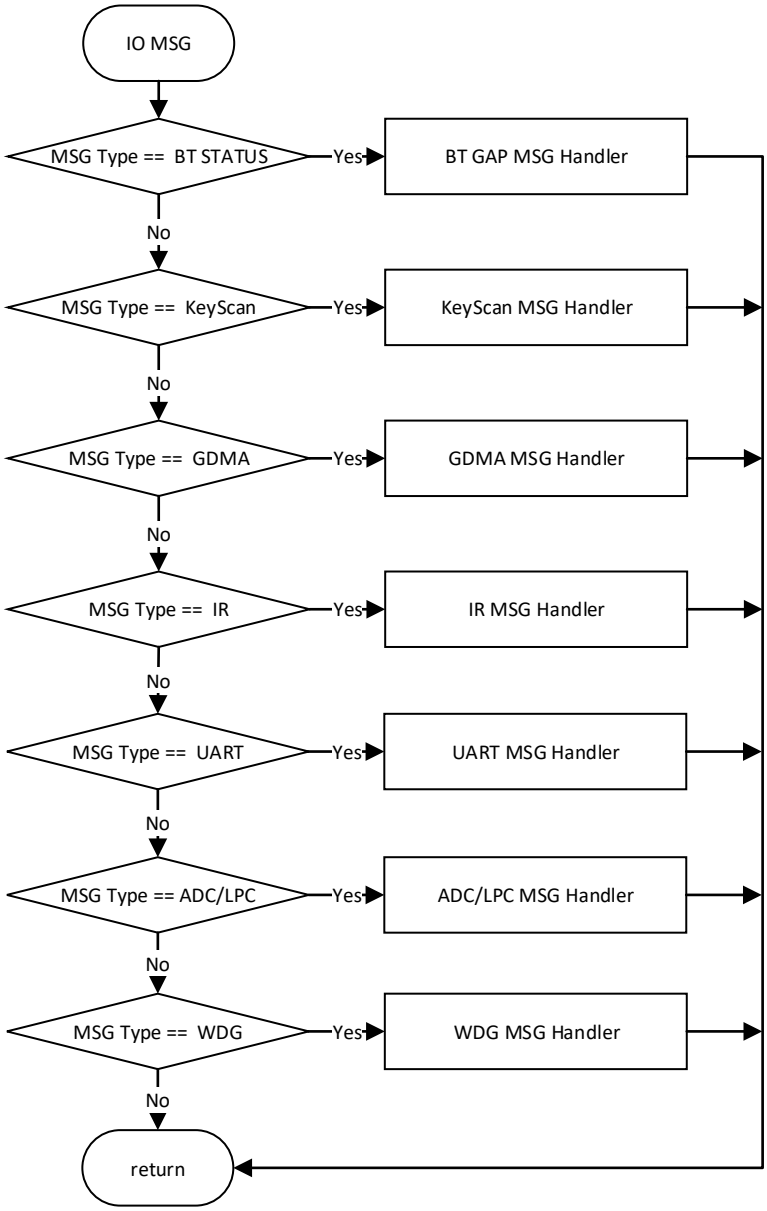


图表 3 正常启动流程图

6 APP TASK



图表 4 App task flow



图表 5 APP IO Message Handler

7 遥控器状态

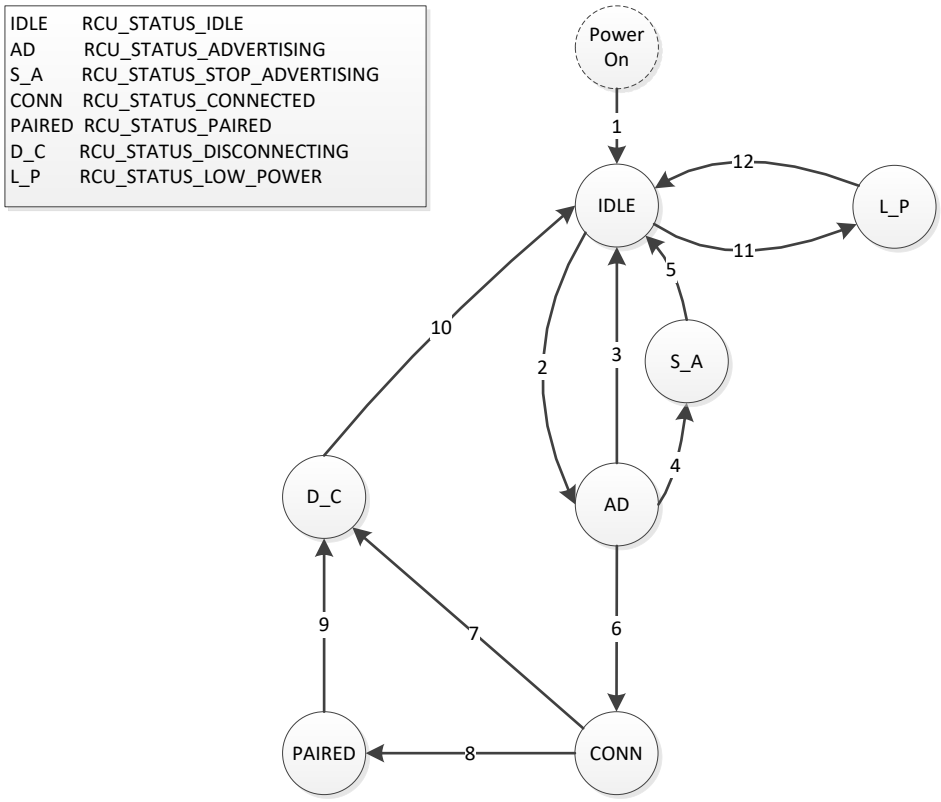
7.1 遥控器状态表

遥控器在运行过程中,在如下几种状态中进行切换。

编号	状态	说明
1	RCU_STATUS_IDLE	IDLE state
2	RCU_STATUS_ADVERTISING	Advertising state
3	RCU_STATUS_STOP_ADVERTISING	Stop advertising command has issued to stack, but GAP state is advertising (temporary state)
4	RCU_STATUS_CONNECTED	Connection has been established, but pairing is not started yet
5	RCU_STATUS_PAIRED	Pairing successfully state
6	RCU_STATUS_DISCONNECTING	Disconnecting command has issued to stack, but GAP state is connected (temporary state)
7	RCU_STATUS_LOW_POWER	Low power mode state

图表 6 遥控器状态

7.2 遥控器状态转换关系图



图表 7 遥控器状态转换图

7.3 遥控器状态转换条件

1	Power On after GAP ready
2	When APP call le_adv_start in idle status
3	High duty cycle direct advertising time out, no connect request received
4	When APP call le_adv_stop in advertising status
5	When BT stack send GAP state change callback message from advertising to idle status
6	When connection established
7	When connection terminated in connected status
8	When pairing successfully in connected status
9	When connection terminated in paired status
10	When BT stack send GAP state change callback message from connection to idle status
11	When low power voltage detected in idle status
12	When normal power voltage detected in low power status

图表 8 遥控器状态转换条件

7.4 遥控器进入 IDLE 状态说明

除第一次上电，系统初始化完成后，进入 IDLE 状态外，其他进入 IDLE 状态主要分为以下两大类:

A) 广播状态结束，没有建立连接；

B) 连接状态下，连接断开。

对 A 种状态，程序把这类归类为停止广播原因，枚举如下：

```
typedef enum
{
    STOP_ADV_REASON_IDLE = 0,
    STOP_ADV_REASON_PAIRING,
    STOP_ADV_REASON_TIMEOUT,
    STOP_ADV_REASON_POWERKEY,
    STOP_ADV_REASON_LOWPOWER,
    STOP_ADV_REASON_UART_CMD,
} T_STOP_ADV_REASON;
```

No.	STOP_ADV_REASON	说明
1	STOP_ADV_REASON_IDLE	收到 ADV_DIRECT_HDC 广播停止的 stack callback message
2	STOP_ADV_REASON_PAIRING	要进行配对广播的发送，停止当前的广播
3	STOP_ADV_REASON_TIMEOUT	APP 广播超时后调用 le_adv_stop 停止广播
4	STOP_ADV_REASON_POWERKEY	要进行开机广播的发送，停止当前的广播
5	STOP_ADV_REASON_LOWPOWER	停止广播，进入 Low-Power 模式
6	STOP_ADV_REASON_UART_CMD	收到 UART 命令停止广播

图表 9 STOP ADV REASON

对 B 种状态，程序把这类归类为断线原因，枚举如下：

```
typedef enum
{
    DISCONN_REASON_IDLE = 0,
    DISCONN_REASON_PAIRING,
    DISCONN_REASON_TIMEOUT,
    DISCONN_REASON_OTA,
    DISCONN_REASON_PAIR_FAILED,
    DISCONN_REASON_LOW_POWER,
    DISCONN_REASON_UART_CMD,
    DISCONN_REASON_SILENT_OTA,
} T_DISCONN_REASON;
```

No.	DISCONN_REASON	说明
1	DISCONN_REASON_IDLE	默认初始化值
2	DISCONN_REASON_PAIRING	要进行配对广播的发送，断开 BLE 连接
3	DISCONN_REASON_TIMEOUT	当程序中开启 FEATURE_SUPPORT_NO_ACTION_DISCONN 后，在设定 timeout 时间内没有按键等操作

4	DISCONN_REASON_OTA	要进行 OTA 升级重启，断开 BLE 连接
5	DISCONN_REASON_PAIR_FAILED	配对失败，断开当前的连接
6	DISCONN_REASON_LOW_POWER	要进入 Low Power Status，断开 BLE 连接
7	DISCONN_REASON_UART_CMD	收到 UART 断线命令，断开 BLE 连接
8	DISCONN_REASON_SILENT_OTA	要进行静默升级重启，断开 BLE 连接

图表 10 DISCONNECT REASON

8 SW Timer 使用场景和作用

目前遥控器应用程序定义了 11 个软件定时器，用途如下表所示。

No.	SW Timer	说明
1	adv_timer	定时器用于超时停止广播
2	next_state_time_out	定时器用于处理配对异常超时问题
3	no_act_disconn_timer	定时器用于长时间无操作，遥控器主动断开 BLE 连接
4	update_conn_params_timer	定时器用于在超时后进行连接参数更新操作
5	keyscan_timer	定时器用于检测 KeyScan 按键状态
6	combine_keys_detection_timer	定时器用于检测组合按键状态
7	notify_key_data_after_reconn_timer	定时器用于回连后延时补发按键键值
8	ir_learn_timer	定时器用于红外学习超时控制
9	led_ctrl_timer	定时器用于 LED 控制
10	single_tone_exit_timer	定时器用于超时退出 SingleTone 测试模式
11	voice_exception_timer	定时器用于限制单次 ATV 语音最长工作时间

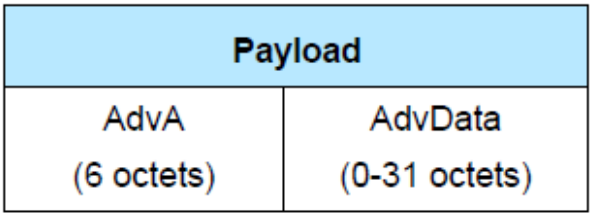
图表 11 SW timer 使用介绍

9 广播包

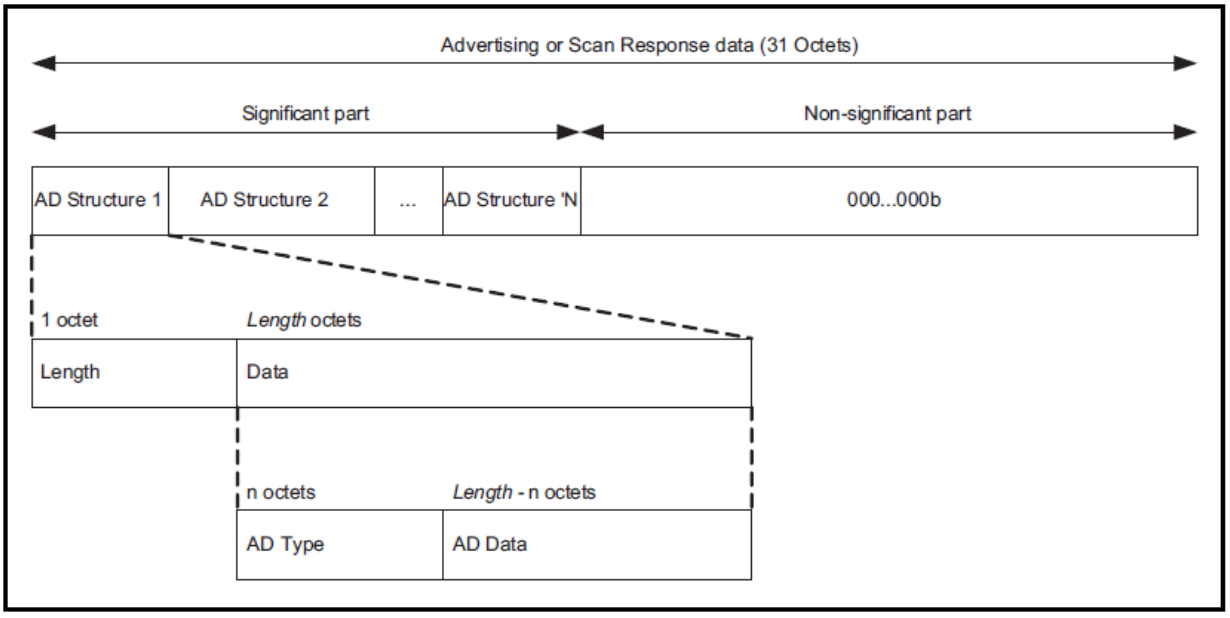
9.1 广播包格式

遥控器使用两种格式的广播包：

1. Undirected advertising event。其中，AdvA 字段是发 advertising 封包设备的地址，AdvData 格式如图 13 所示。



图表 12 ADV_IND PDU Payload



图表 13 Advertising and Scan Response data format

2. Directed advertising event。其中 AdvA 字段是发 advertising 封包设备的地址，InitA 是回连设备的地址。

Payload	
AdvA (6 octets)	InitA (6 octets)

图表 14 ADV_DIRECT_IND PDU Payload

9.2 广播包类型

遥控器应用程序通过调用 rcu_start_adv 进行发送广播，并设定广播类型，遥控器广播类型的枚举如下：

```
1. typedef enum
2. {
3.     ADV_IDLE = 0,
4.     ADV_DIRECT_HDC,
5.     ADV_UNDIRECT_RECONNECT,
6.     ADV_UNDIRECT_PAIRING,
7.     ADV_UNDIRECT_PROMPT,
8.     ADV_UNDIRECT_POWER,
9. } T_ADV_TYPE;
```

遥控器的广播包有四种类型：ADV_UNDIRECT_PAIRING，ADV_DIRECT_HDC/ADV_UNDIRECT_RECONNECT，ADV_UNDIRECT_PROMPT 和 ADV_UNDIRECT_POWER。分别对应配对模式，回连模式，提示配对模式和开机模式。

9.2.1 配对广播包

遥控器在配对模式时发送配对广播包，用于触发主机端发起自动配对过程。配对广播包的格式为 Undirected Advertising Packet，Advertising Interval 范围建议设为 0x20 - 0x30（即 20ms - 30ms），持续时间为 20s。在以下 2 种情况下会进入配对模式：

1. 按配对组合键（音量+键和确认键）并持续 2s 后，遥控器上存储的 Link Key 会被清除，遥控器进入配对模式。
2. 如果遥控器上没有配对信息，并且程序中宏 FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV 被置 1，则任意按键事件均会触发遥控器进入配对模式。

   如果 FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV 是 0，则只有配对组合键可以进入配对模式，其他任意键无效。默认 FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV 是 0。

配对广播包格式如下：

Flag field（byte0~2）	Service Field（byte3~6）	Appearance Field – Device type (byte7~10)	Local Name Field (byte11~20)	Vendor Information Field (byte21~26)
0x02,0x01,0x05	0x03,0x03, 0x12,0x18	0x03,0x19,0x80,0x01	0x09,0x09, 'B', 'e', 'e', '2', ' ', 'R', 'C', 'U'	0x05,0xff,0x5d,0x00,0x04, 0x00

图表 15 配对广播包格式

9.2.2 回连广播包

遥控器发送回连广播包，用于在遥控器保存有 Link Key 时迅速和对端设备建立连接。

回连广播包的格式是 Direct Advertising High Duty Packet，每次连续发送 1.28s，超时后 Stack 会自动停止广播发送，并送 message 给 APP。默认 APP 会进行 3 次回连尝试。在以下 2 种情况下遥控器进入回连模式：

1. 遥控器上电，如果遥控器成功配对过并且保存有 Link Key，在上电初始化完成后遥控器进入回连模式。
2. 遥控器和对端连接上后，由于对端关机或遥控器超出控制距离等情况被动断开连接，或者遥控器进入低功耗模式时主动断开连接后，任意按键均会触发遥控器进入回连模式。

默认遥控器应用程序采用 Direct Advertising High Duty Packet 的方式进行回连，该模式的回连效率最高，回连后补发按键键值的延时最短。但该模式的运行功耗较高，如果方案中需要进行较长时间的回连尝试，可以使用另外两种回连方案：Low Duty Cycle 的 Direct Advertising 和设置 white list 的 Undirect Advertising。

设置 white list 的 Undirect Advertising 的广播包格式为 Undirected Advertising Packet，Advertising Interval 建议设为 0x20（即 20ms），持续时间为 3s。当宏 FEATURE_SUPPORT_PRIVACY 置 1 时，回连模式发送 Undirected

Advertising Packet，当宏 FEATURE_SUPPORT_PRIVACY 置 0 时，回连模式发送 Direct Advertising High Duty Packet。默认情况下 FEATURE_SUPPORT_PRIVACY 为 0，表示不支持 PRIVACY 功能。

9.2.3 提示广播包

遥控器发送不可连接的提示广播包，用于触发主机端弹出连接提示框，提示客户按照要求进行配对操作。提示广播包是 Undirected Advertising Packet，Advertising Interval 范围建议为 0x20 - 0x30（即 20ms - 30ms），持续时间为 5s。

程序中，通过宏 SUPPORT_ADV_UNDIRECT_NONCON 来控制是否支持发送不可见广播包。默认情况下，FEATURE_SUPPORT_UNDIRECT_PROMPT_ADV 为 0，表示不支持发送提示广播包。如果 FEATURE_SUPPORT_UNDIRECT_PROMPT_AD 被置 1，在如下情况会进入提示配对模式：

1. 在有 link key 情况下，按任意按键（除开机键）进入回连模式，连续三次回连广播包都没有建立连接后，会进入提示配对模式发送不可发现广播包。

提示广播包格式：（和配对广播包的区别就是 Limited Discoverable Mode bit 为 0 且不可连接）

Flag field（byte0~2）	Service Field（byte3~6）	Appearance Field – Device type (byte7~10)	Local Name Field (byte11~20)	Vendor Information Field (byte21~26)
0x02,0x01,0x04	0x03,0x03, 0x12,0x18	0x03,0x19,0x80,0x01	0x09,0x09, 'B', 'e', 'e', '2', ' ', 'R', 'C', 'U'	0x05,0xff,0x5d,0x00,0x04, 0x00

图表 16 提示广播包格式

9.2.4 开机广播包

遥控器发送开机广播包，用于唤醒处于休眠模式的主机端。开机广播包是 Undirected Advertising Packet，Advertising Interval 范围为 0x20 - 0x30（即 20ms - 30ms），持续时间为 12s。开机广播包含对端 BD Address，因此遥控器只能唤醒配对过的对端设备。

在以下 2 种情况下会进入开机模式：

1. 在有 link key 的情况下，按下 power 按键，进入开机模式；
2. 在回连模式或者提示配对模式中，按下 power 按键，进入开机模式。

开机广播包格式：

Flag field（byte0~2）	Service Field（byte3~6）	Appearance Field – Device type (byte7~10)	Vendor Information Field (byte11~24)
0x02,0x01,0x04	0x03,0x03, 0x12,0x18	0x03,0x19,0x80,0x01	0x0D,0xFF,0x5D,0x00,0x03,0x00,0x01,0x01(index), 0xXX,0xXX, 0xXX, 0xXX, 0xXX,0xXX,(TV BD Address)

图表 17 开机广播包格式

10 连接和配对

遥控器和主机端的连接配对流程根据配对信息状态不同，分为如下几种情况：

1. 遥控器和主机端已配对过并且都有保存配对信息时，遥控器任意按键可迅速回连，不需要重新配对。
2. 遥控器和主机端都没有保存配对信息时，遥控器按配对组合按键触发自动配对过程，建立连接，并进行配对。
3. 遥控器保存配对信息，主机端清除配对信息时，遥控器发送回连广播无法建立连接，需要按组合键触发自动配对过程，建立连接，进行配对。
4. 遥控器清除了配对信息，主机端保存之前的配对信息时，遥控器发送回连广播可建立连接，主机端走回连流程，遥控器端会出现 Key Missing 并配对失败。为处理上述现象，遥控器会断开连接，并再次发送配对广播和主机端建立连接，并重新配对。

配对成功后，主机端会读取遥控器的 GATT 服务，并使能相关的 CCCD 信息。为了遥控器使用功耗更低，在配对之后，遥控器会主动发送更新连接参数请求，使主机端使用设定的连接参数。默认应用方案使用的连接参数为：Connection Interval 15ms，Slave Latency 14，Supervision Timeout Period 2s。

11 BLE Service

遥控器支持的 BLE Service 如下表所示：

No.	Service	说明
1	GAP	通用访问协议
2	DIS	设备信息服务
3	HIDS	HID 服务
4	BAS	电量查询服务
5	OTA Service	Realtek OTA 升级服务
6	DFU Service	Realtek 静默升级服务
7	Vendor Service	Realtek RCU Test Tool 相关服务
8	ATV Voice Service	Google Vendor ATV Voice Service
9	RTK GATT Voice Service	RTK GATT Voice Service

图表 18 BLE Service

12 按键

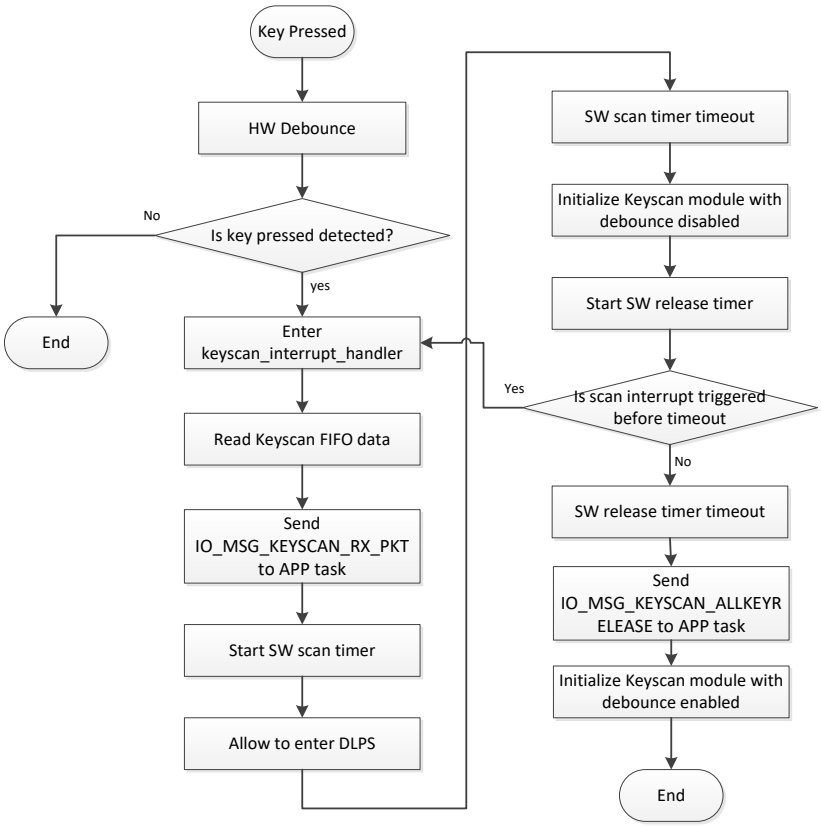
12.1 Keyscan 方案

遥控器应用程序使用硬件 Keyscan + SW Timer + HW debounce 的方案，来达到 Keyscan 模块最低功耗的目的。Keyscan 的设置分为 Platform always Active mode 和 Platform Sleep at scan interval 两种状态。两者的区别是 debounce 的使用方案：对于 Platform always Active mode，debounce 正常使用 Keyscan 模块的 debounce；而在 Platform Sleep at scan interval 时，debounce 使用 PAD 的 debounce，进行缩短产生 Keyscan Interrupt 时间。

当有按键被按下后，HW 做 debounce 和一次 Scan。Scan 结束后，软件会产生 scan interrupt，在中断处理函数中读取 Keyscan FIFO data，并发送 IO_MSG_KEYSCAN_RX_PKT 消息给 APP task 处理。同时开启一个 scan interval 的 SW scan timer 后，进入 DLPS。SW scan timer 到时后唤醒系统，重新初始化 Keyscan 模块，此时 debounc 应 disable，并开启一个 10ms 的 SW release timer，进行按键状态检测。

若按键仍然处于被按下状态，则会触发 Keyscan 进行一次 scan，scan 结束后收到 scan interrupt，中断处理函数中按之前的流程操作，重置 SW scan timer，继续检测按键的状态。

若所有按键都被释放，则不会触发 scan interrupt，在 SW release timer 到时后，发送 IO_MSG_KEYSCAN_ALLKEYRELEASE 给 APP task 处理。并重新初始化 Keyscan 模块，此时 debounce 应 enable。



图表 19 Keyscan 流程

12.2 Keyscan 配置

遥控器应用程序中，默认配置为 5 行 4 列的按键矩阵，行列 GPIO 在 board.h 文件中定义，可根据实际项目需求进行修改：

```
/*************************************************************
*                  RCU Keyscan Config
*************************************************************/
/* keypad row and column */
#define KEYPAD_ROW_SIZE       5
#define KEYPAD_COLUMN_SIZE    4

#define ROW0                  P4_3
#define ROW1                  P4_2
#define ROW2                  P4_1
#define ROW3                  P4_0
#define ROW4                  P0_6

#define COLUMN0               P3_0
#define COLUMN1               P3_1
#define COLUMN2               P3_2
#define COLUMN3               P3_3
```

Keyscan 的行列按键定义在 key_handle.c 中，可根据实际项目需求进行修改：

```
/* Key Mapping Table Definiton */
static const T_KEY_INDEX_DEF KEY_MAPPING_TABLE[KEYPAD_ROW_SIZE][KEYPAD_COLUMN_SIZE] =
{
    {VK_TV_POWER,      VK_EXIT,         VK_ENTER,     VK_MOUSE_EN},
    {VK_POWER,         VK_VOLUME_UP,    VK_DOWN,      VK_RIGHT},
    {VK_TV_SIGNAL,     VK_VOLUME_DOWN,  VK_VOICE,     VK_MENU},
    {VK_UP,            VK_PAGE_DOWN,    VK_HOME,      VK_PAGE_UP},
    {VK_LEFT,          VK_NC,           VK_NC,        VK_NC},
};
```

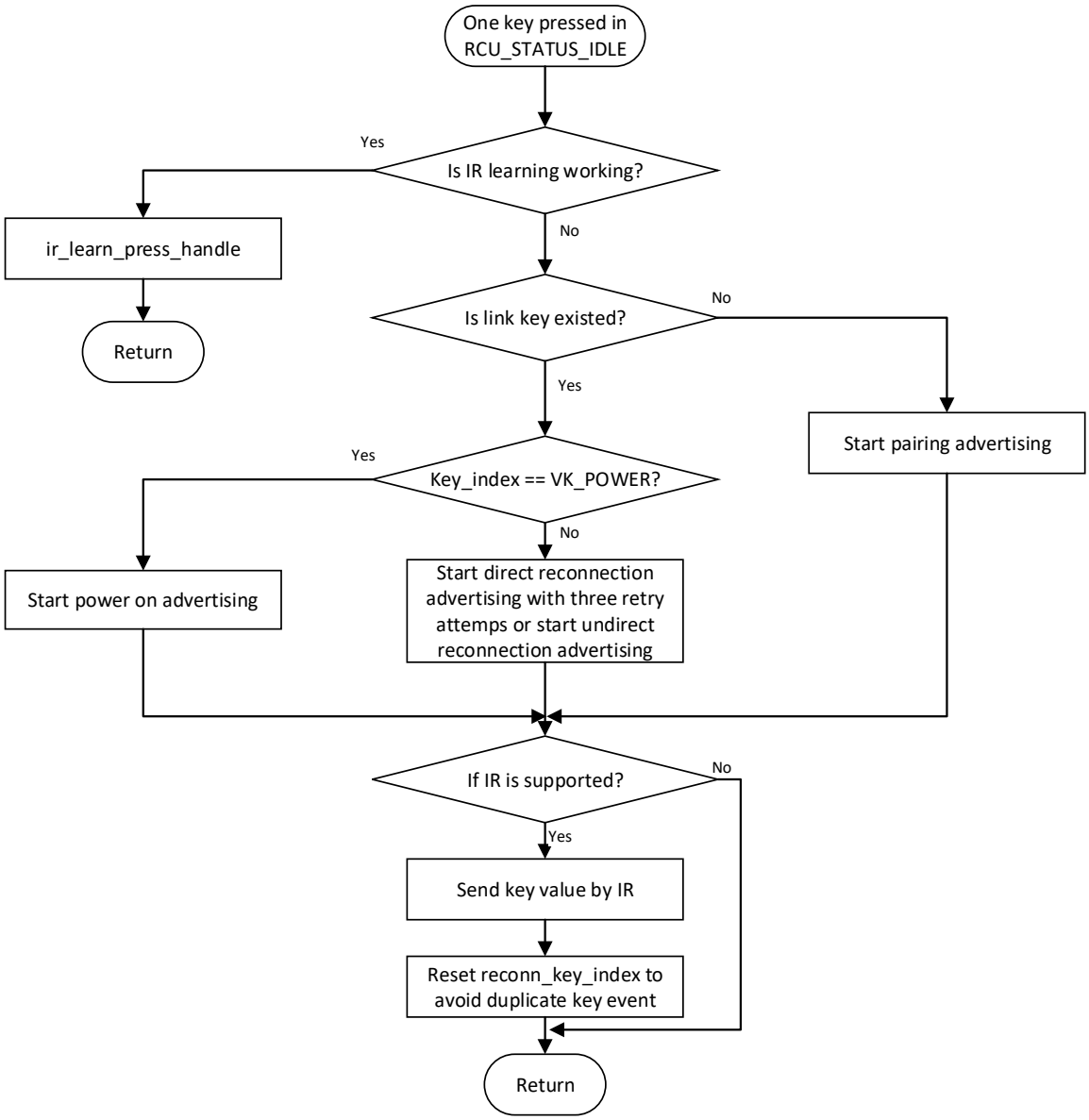
12.3 按键行为规范

本节主要介绍按键的行为规范，提供一种按键处理流程方案，实际可根据具体项目需求进行相应调整。该方案主要依据检测到的按键个数及当前遥控器状态，来进行不同的按键行为。

12.3.1 单个按键行为

在单个按键被按下之后，遥控器应用程序收到 Keyscan 模块发送的 IO_MSG_KEYSCAN_RX_PKT，根据当前 APP 的状态，分别进行如下处理：

- RCU_STATUS_IDLE 时，

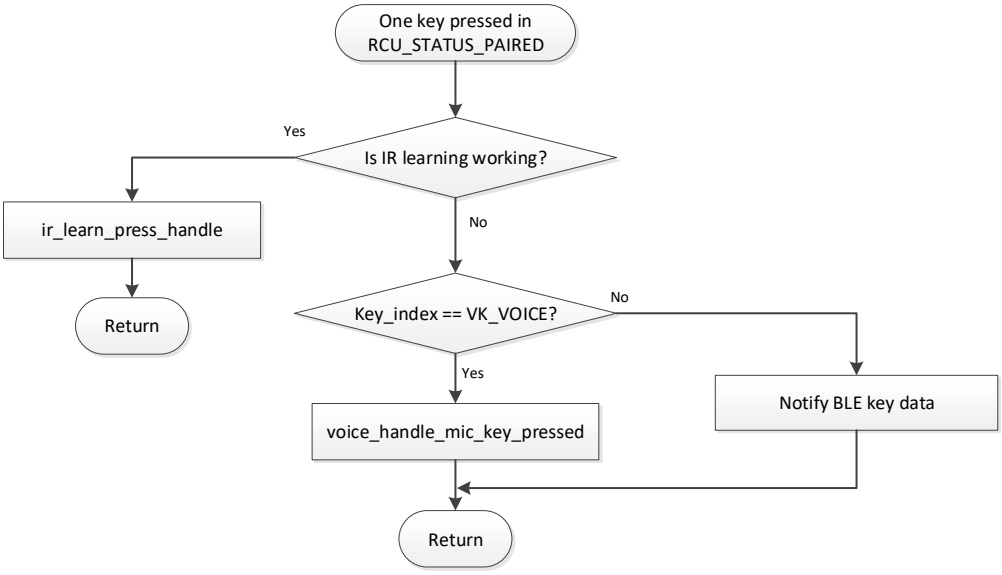


图表 20 RCU_STATUS_IDLE 单个按键处理流程

- RCU_STATUS_ADVERTISING 时，

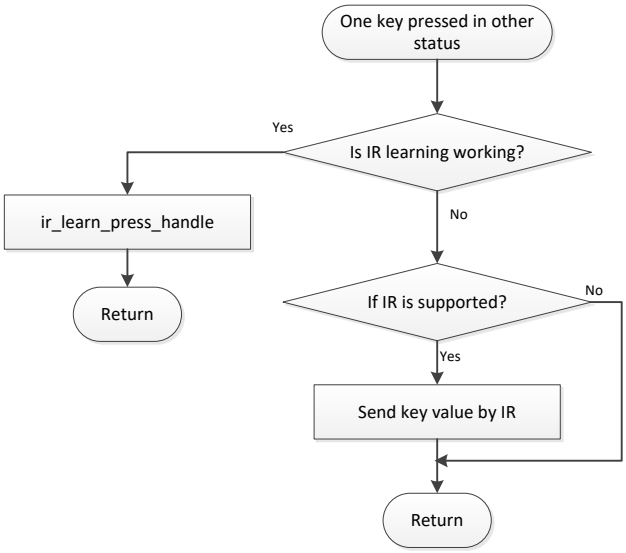
图表 21 RCU_STATUS_ADVERTISING 单个按键处理流程

- RCU_STATUS_PAIRED 时，



图表 22 RCU_STATUS_PAIRED 单个按键处理流程

- Other status 时，



图表 23 其他状态单个按键处理流程

12.3.2 两个按键行为

当两个按键同时被按下时，RCU APP 会检测是否为定义的组合按键：

如果不是组合按键，则 RCU APP 会发送 Key Up 的 BLE Notification 或停止之前的红外发送；

如果是组合按键，则会开启 SW Timer，来确认组合按键是否满足按下时间要求；

RCU APP 方案使用一个 32bit 的全局变量来区分不同的组合按键，每种组合键对应一个 bit，目前有如下几组组合按键，实际可根据具体项目需求进行相应调整。

No.	Bit Mask	组合键	说明
1	0x0001	VK_ENTER + VK_VOLUME_UP	进入配对模式
2	0x0002	VK_ENTER + VK_LEFT	进入红外学习模式（IR_LEARN_MODE 有效）
3	0x0004	VK_POWER + VK_VOLUME_UP	进入 HCI UART 测试模式 （MP_TEST_MODE_SUPPORT_HCI_UART_TEST 有效）
4	0x0008	VK_POWER + VK_VOLUME_DOWN	进入 Data UART 测试模式 （MP_TEST_MODE_SUPPORT_DATA_UART_TEST 有效）
5	0x0010	VK_POWER + VK_EXIT	进入 SingleTone 测试模式 （MP_TEST_MODE_SUPPORT_SINGLE_TONE_TEST 有效）
6	0x0020	VK_POWER + VK_UP	进入快速配对模式 1 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST 有效）
7	0x0040	VK_POWER + VK_DOWN	进入快速配对模式 2 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST 有效）
8	0x0080	VK_POWER + VK_LEFT	进入快速配对模式 3 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST 有效）
9	0x0100	VK_POWER + VK_RIGHT	进入快速配对模式 4

			（MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST 有效）
10	0x0200	VK_POWER + VK_ENTER	进入快速配对模式 5 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST 有效）

图表 24 组合键定义

12.3.3 三个及以上按键行为

当三个及以上按键被按下时，RCU APP 会发送 Key Up 的 BLE Notification 或停止之前的红外发送。

12.4 按键 HID 码值表格

RCU APP 支持两种 HID Usage Page 的键值：Keyboard/Keypad Page(0x07)和 Consumer Page(0x0C)。

对于 Keyboard/Keypad Page 的按键，其键值直接定义在 key_handle.c 的 BLE_KEY_CODE_TABLE 中。在发 key notification 时，直接发送 HID Key Value。

对于 Consumer Page 的按键，其键值定义在 hids_rmc.c 的 hids_report_descriptor 中，而 key_handle.c 的 BLE_KEY_CODE_TABLE 中是其对应的 Key Index。在发 key notification 时发送 Key Index，主机端根据 Key Index 及 hids_report_descriptor 来获得对应的 Key Value。

Key No.	HID Usage Page	HID Value
VK_POWER	Keyboard/Keypad Page(0x07)	0x66
VK_PAGE_UP	Keyboard/Keypad Page(0x07)	0x4B
VK_PAGE_DOWN	Keyboard/Keypad Page(0x07)	0x4E
VK_MENU	Keyboard/Keypad Page(0x07)	0x76
VK_HOME	Keyboard/Keypad Page(0x07)	0x4A
VK_VOICE	Keyboard/Keypad Page(0x07)	0x3E
VK_ENTER	Keyboard/Keypad Page(0x07)	0x28
VK_EXIT	Keyboard/Keypad Page(0x07)	0x29
VK_LEFT	Keyboard/Keypad Page(0x07)	0x50
VK_RIGHT	Keyboard/Keypad Page(0x07)	0x4F
VK_UP	Keyboard/Keypad Page(0x07)	0x52
VK_DOWN	Keyboard/Keypad Page(0x07)	0x51
VK_VOLUME_MUTE	Keyboard/Keypad Page(0x07)	0x7F
VK_VOLUME_UP	Keyboard/Keypad Page(0x07)	0x80
VK_VOLUME_DOWN	Keyboard/Keypad Page(0x07)	0x81
VK_VOICE_STOP	Keyboard/Keypad Page(0x07)	0x3F
MM_ScanNext	Consumer Page(0x0C)	0xB5
MM_ScanPrevious	Consumer Page(0x0C)	0xB6
MM_Stop	Consumer Page(0x0C)	0xB7
MM_Play_Pause	Consumer Page(0x0C)	0xCD

MM_Mute	Consumer Page(0x0C)	0xE2
MM_BassBoost	Consumer Page(0x0C)	0xE5
MM_Loudness	Consumer Page(0x0C)	0xE7
MM_VolumeIncrement	Consumer Page(0x0C)	0xE9
MM_VolumeDecrement	Consumer Page(0x0C)	0xEA
MM_BassIncrement	Consumer Page(0x0C)	0x0152
MM_BassDecrement	Consumer Page(0x0C)	0x0153
MM_TrebleIncrement	Consumer Page(0x0C)	0x0154
MM_TrebleDecrement	Consumer Page(0x0C)	0x0155
MM_AL_ConsumerControl	Consumer Page(0x0C)	0x0183
MM_AL_EmailReader	Consumer Page(0x0C)	0x018A
MM_AL_Calculator	Consumer Page(0x0C)	0x0192
MM_AL_LocalMachineBrowser	Consumer Page(0x0C)	0x0194
MM_AC_Search	Consumer Page(0x0C)	0x0221
MM_AC_Home	Consumer Page(0x0C)	0x0223
MM_AC_Back	Consumer Page(0x0C)	0x0224
MM_AC_Forward	Consumer Page(0x0C)	0x0225
MM_AC_Stop	Consumer Page(0x0C)	0x0226
MM_AC_Refresh	Consumer Page(0x0C)	0x0227
MM_AC_Bookmarks	Consumer Page(0x0C)	0x022A

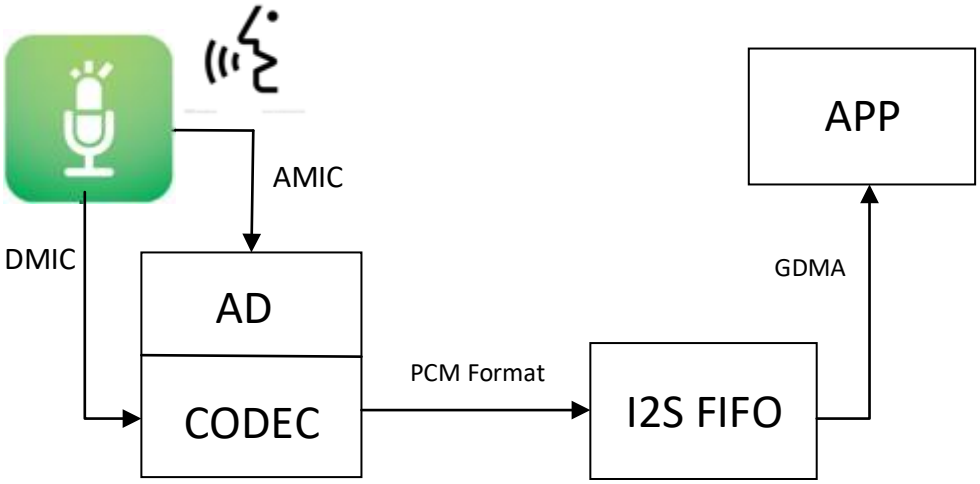
图表 25 按键编码表

13 语音

13.1 语音功能介绍

遥控器有语音功能，通过数字麦克风(Digital Microphone/DMIC)或者模拟麦克风(Analog Microphone/AMIC)可以录入外界声音信息，再进行语音编码并通过蓝牙传送到盒子端，配合语音识别软件，实现丰富多彩的应用。功能的整体框架可以参考图表 26 语音功能整体框架图。

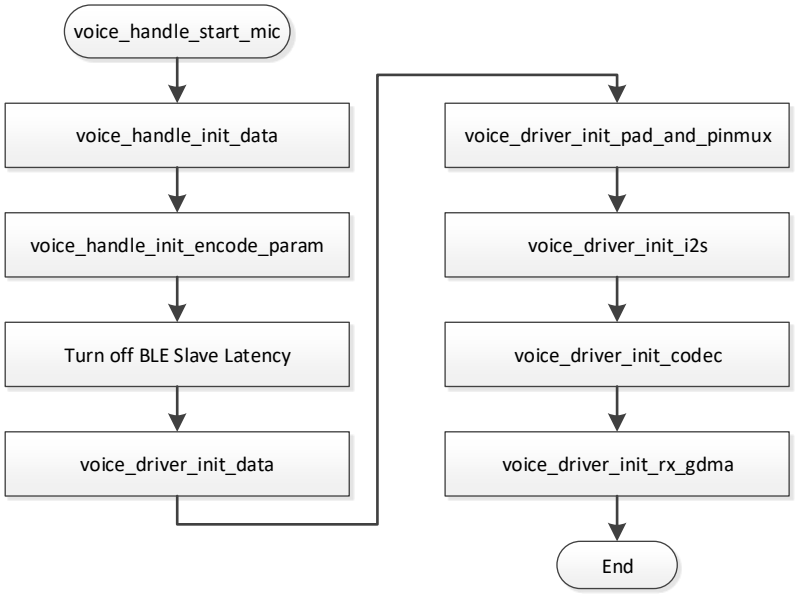
当使用 DMIC 时，通过 PIN 脚输入的是外界声音经过 AD 转换过的数字信号，遥控器直接把这些 data 搬到 CODEC，由 CODEC 编码成 PCM 格式，再放到 I2S FIFO 中，之后 GDMA 再把这些 data 搬移到 RAM 中提供给 App 处理。当使用 AMIC 时，通过 PIN 脚输入的是模拟电压信号，遥控器会先进行 AD 转换，然后将转换完 data 通过 CODEC 编码成 PCM 格式，再放到 I2S FIFO 中，之后 GDMA 再把这些 PCM 格式的 data 搬移到 RAM 中提供给 App 处理。



图表 26 语音功能整体框架图

13.2 语音模块初始化

要使用语音功能，必须先初始化模块，包括语音模块数据初始化和 Pad/Pinmux/I2S/CODEC/GDMA 等 IO 初始化。程序中，语音模块的初始化调用函数 voice_handle_start_mic。



图表 27 语音模块初始化流程

13.2.1 语音 PAD 初始化

Bee2 RCU 如果使用 AMIC，需要使用 MIC_N、MIC_P 和 MICBIAS 三个 PAD，分别对应 P2_6、P2_7 和 H0。如果使用 DMIC，需要使用 DMIC_CLK 和 DMIC_DAT 两个 PAD，默认使用 P1_6、P1_7，可以使用 PINPUX 来切换其他 GPIO。具体初始化代码如下：

```
1.  static void voice_driver_init_pad_and_pinmux(void)
2.  {
3.  #if (VOICE_MIC_TYPE == AMIC_TYPE)
4.      Pad_Config(H_0, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE, PAD
    _OUT_HIGH);
5.      Pad_Config(AMIC_PIN1, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_DISAB
    LE, PAD_OUT_HIGH);
6.      Pad_Config(AMIC_PIN2, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_DISAB
    LE, PAD_OUT_LOW);
7.  #elif (VOICE_MIC_TYPE == DMIC_TYPE)
8.      Pad_Config(DMIC_MSBC_CLK, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OU
    T_DISABLE,  PAD_OUT_LOW);
9.      Pad_Config(DMIC_MSBC_DAT, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_O
    UT_DISABLE,  PAD_OUT_LOW);
10.     Pinmux_Deinit(DMIC_MSBC_CLK);
11.     Pinmux_Deinit(DMIC_MSBC_DAT);
12.     Pinmux_Config(DMIC_MSBC_CLK, DMIC1_CLK);
13.     Pinmux_Config(DMIC_MSBC_DAT, DMIC1_DAT);
14. #endif
15. }
```

13.2.2 语音 I2S 初始化

Bee2 RCU 使用 I2S 来将 CODEC 的 PCM data 传输到 APP。I2S 配置时，需要注意配置的 BCLK 要和 CODEC 的采样率保持一致。具体初始化代码如下：

```
1.  static void voice_driver_init_i2s(void)
2.  {
3.      RCC_PeriphClockCmd(APBPeriph_I2S0, APBPeriph_I2S0_CLOCK, ENABLE);
4.
5.      I2S_InitTypeDef I2S_InitStruct;
6.      I2S_StructInit(&I2S_InitStruct);
7.      I2S_InitStruct.I2S_ClockSource      = I2S_CLK_40M;
8.  #if (CODEC_SAMPLE_RATE_SEL == CODEC_SAMPLE_RATE_8K)
9.      I2S_InitStruct.I2S_BClockMi         = 0x271;/* <!BCLK = 8K */
10.     I2S_InitStruct.I2S_BClockNi         = 0x08;
11. #elif (CODEC_SAMPLE_RATE_SEL == CODEC_SAMPLE_RATE_16K)
12.     I2S_InitStruct.I2S_BClockMi         = 0x271;/* <!BCLK = 16K */
13.     I2S_InitStruct.I2S_BClockNi         = 0x10;
14. #endif
15.     I2S_InitStruct.I2S_DeviceMode       = I2S_DeviceMode_Master;
16.     I2S_InitStruct.I2S_ChannelType      = I2S_Channel_Mono;
17.     I2S_InitStruct.I2S_DataFormat       = I2S_Mode;
18.     I2S_InitStruct.I2S_RxWaterlevel      = 0x4;
19.     I2S_InitStruct.I2S_TxWaterlevel      = 0x8;
20.     I2S_Init(I2S0, &I2S_InitStruct);
21.
22.     I2S_Cmd(I2S0, I2S_MODE_TX | I2S_MODE_RX, ENABLE);
23. }
```

13.2.3 语音 CODEC 初始化

Bee2 RCU Codec 可设置 8KHz 和 16KHz 两种采样率。

对于 AMIC 来说，初始化时需要配置 MicBIAS、MICBstMode、MICBstGain 和 AdGain。

- MICBIAS 为 MIC 的偏置电压，需要根据 MIC 的参数来设置；
- MICBstMode 为语音输入方式选择，分为 MICBST_Mode_Single 和 MICBST_Mode_Differential，需根据硬件电路的连接方式来选择，建议采用默认的 MICBST_Mode_Differential；
- MICBstGain 可以调整 Codec 的 Analog Gain，默认为 MICBST_Gain_20dB。AdGain 可以调整 Codec 的 Digital Gain，默认为 0x2F。该值需要根据要求的拾音距离、遥控器外壳综合进行调整。调整原则为：在可以调整 MICBstGain 达到需求的情况下，尽量保持 AdGain 为默认值，这样可以使放大后的信号的信噪较好。

对于 DMIC 来说，初始化时需要配置 DmicClock、DmicDataLatch 和 BoostGain。

- DmicClock 设置 DMIC 的时钟，默认为 DMIC_Clock_2MHz；
- DmicDataLatch 设置 DMIC 的 Data Latch Type，默认为 DMIC_Rising_Latch；
- BoostGain 设置 Codec 的 Digital Gain，默认为 Boost_Gain_0dB；

具体初始化代码如下：

```
1.  static void voice_driver_init_codec(void)
2.  {
3.      RCC_PeriphClockCmd(APBPeriph_CODEC, APBPeriph_CODEC_CLOCK, ENABLE);
4.
5.      CODEC_InitTypeDef CODEC_InitStruct;
6.      CODEC_StructInit(&CODEC_InitStruct);
7.
8.      /* set codec common parameters */
9.      CODEC_InitStruct.CODEC_SampleRate = voice_driver_codec_params.codec_sample_rate;
10.     CODEC_InitStruct.CODEC_MicType = voice_driver_codec_params.mic_type;
11.
12.     if (voice_driver_codec_params.mic_type == CODEC_AMIC)
13.     {
14.         CODEC_InitStruct.CODEC_MicBIAS = voice_driver_codec_params.amic_bias_voltage;
15.         CODEC_InitStruct.CODEC_MICBstMode = voice_driver_codec_params.amic_input_mode;
16.         CODEC_InitStruct.CODEC_MICBstGain = voice_driver_codec_params.amic_analog_gain;
17.         CODEC_InitStruct.CODEC_AdGain = voice_driver_codec_params.amic_digital_gain;
18.     }
19.     else if (voice_driver_codec_params.mic_type == CODEC_AMIC)
20.     {
21.         CODEC_InitStruct.CODEC_DmicClock = voice_driver_codec_params.dmic_clock;
22.         CODEC_InitStruct.CODEC_DmicDataLatch = voice_driver_codec_params.dmic_data_latc
    h;
23.         CODEC_InitStruct.CODEC_BoostGain = voice_driver_codec_params.dmic_boost_gain;
24.     }
25.
26.     CODEC_InitStruct.CODEC_I2SFormat          = CODEC_I2S_DataFormat_I2S;
27.     CODEC_InitStruct.CODEC_I2SDataWidth       = CODEC_I2S_DataWidth_16Bits;
28.     CODEC_Init(CODEC, &CODEC_InitStruct);
29.
30. #if SUPPORT_CODEC_EQ_CONFIG_FEATURE
31.     /* init codec EQ parameters */
32.     for (uint8_t eq_index = 0; eq_index < CODEC_EQ_MAX_NUM; eq_index++)
33.     {
34.         CODEC_EQInitTypeDef codec_eq_params;
35.         CODEC_EQStructInit(&codec_eq_params);
36.         if (voice_driver_codec_params.codec_eq_params[eq_index].CODEC_EQChCmd == EQ_
    CH_Cmd_ENABLE)
37.         {
```

```
            codec_eq_params.CODEC_EQCoefH0 = voice_driver_codec_params.codec_eq_para
ms[eq_index].CODEC_EQCoefH0;
            codec_eq_params.CODEC_EQCoefB1 = voice_driver_codec_params.codec_eq_param
s[eq_index].CODEC_EQCoefB1;
            codec_eq_params.CODEC_EQCoefB2 = voice_driver_codec_params.codec_eq_param
s[eq_index].CODEC_EQCoefB2;
            codec_eq_params.CODEC_EQCoefA1 = voice_driver_codec_params.codec_eq_param
s[eq_index].CODEC_EQCoefA1;
            codec_eq_params.CODEC_EQCoefA2 = voice_driver_codec_params.codec_eq_param
s[eq_index].CODEC_EQCoefA2;
        }
        CODEC_EQInit(CODEC_EQ_TABLE[eq_index], &codec_eq_params);
    }
#endif
}
```

13.2.4 语音 GDMA 初始化

语音 GDMA 主要将 I2S FIFO 中的 data 传输到 RCU APP。具体初始化代码如下:

```
static void voice_driver_init_rx_gdma(void)
{
    /* Enable GDMA clock */
    RCC_PeriphClockCmd(APBPeriph_GDMA, APBPeriph_GDMA_CLOCK, ENABLE);

    /* Initialize GDMA peripheral */
    GDMA_InitTypeDef GDMA_InitStruct;
    GDMA_StructInit(&GDMA_InitStruct);
    GDMA_InitStruct.GDMA_ChannelNum          = GDMA_Channel_RX_NUM;
    GDMA_InitStruct.GDMA_DIR                 = GDMA_DIR_PeripheralToMemory;
    GDMA_InitStruct.GDMA_BufferSize          = VOICE_GDMA_FRAME_SIZE / 4;
    GDMA_InitStruct.GDMA_SourceInc           = DMA_SourceInc_Fix;
    GDMA_InitStruct.GDMA_DestinationInc      = DMA_DestinationInc_Inc;
    GDMA_InitStruct.GDMA_SourceDataSize      = GDMA_DataSize_Word;
    GDMA_InitStruct.GDMA_DestinationDataSize = GDMA_DataSize_Byte;
    GDMA_InitStruct.GDMA_SourceMsize         = GDMA_Msize_4;
    GDMA_InitStruct.GDMA_DestinationMsize    = GDMA_Msize_16;
    GDMA_InitStruct.GDMA_SourceAddr          = (uint32_t)(&(I2S0->RX_DR));
    GDMA_InitStruct.GDMA_DestinationAddr     = (uint32_t)voice_driver_global_data.gdma_buffer.b
uf0;
    GDMA_InitStruct.GDMA_SourceHandshake     = GDMA_Handshake_SPORT0_RX;
    GDMA_InitStruct.GDMA_DestHandshake       = GDMA_Handshake_SPIC_RX;
```

```
22.     GDMA_Init(GDMA_Channel_RX, &GDMA_InitStruct);
23.     GDMA_INTConfig(GDMA_Channel_RX_NUM, GDMA_INT_Transfer, ENABLE);
24.
25.     NVIC_InitTypeDef NVIC_InitStruct;
26.     NVIC_InitStruct.NVIC_IRQChannel = GDMA0_Channel_RX_IRQn;
27.     NVIC_InitStruct.NVIC_IRQChannelPriority = 3;
28.     NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
29.     NVIC_Init(&NVIC_InitStruct);
30.
31.     GDMA_Cmd(GDMA_Channel_RX_NUM, ENABLE);
32. }
```

13.3 语音 Buffer 介绍

Bee2 RCU 的语音模块使用到了两种 buffer：GDMA 的 ping pong buffer 和存放 Voice data 的 loop buffer。

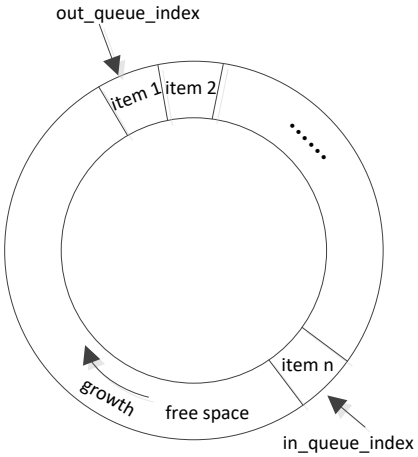
13.3.1 GDMA Ping-Pong Buffer

GDMA ping-pong buffer 的设计目的是为了使 GDMA 从 I2S FIFO 搬运 voice data 到 APP task。Ping-Pong buffer 为两块 GDMA buffer size 的缓存，其中一块作为 GDMA 搬运的目的缓存，另一块是 GDMA 中断向 APP task 发送 message 的数据缓存。每次 GDMA 中断后，进行交替使用。

13.3.2 Voice Data Loop Buffer

Bee2 RCU 针对语音延时及瞬时 RF 干扰问题，设计了一个语音数据的 Loop Buffer，用于存储压缩之后的 Voice 数据。

语音缓存队列是一块内存地址连续的缓存区，当缓存区内的数据存满时，继续存入数据就覆盖掉旧的数据。默认情况下，语音缓存队列是一块 7200Bytes 的内存块，可以存储 900ms 语音数据。入队列指针即**错误!未找到引用源。**为中的 in_queue_index，出队列指针为**错误!未找到引用源。**中的 out_queue_index。当数据写入 buffer 时，in_queue_index 往前移动；当从 buffer 中取数据时，out_queue_index 往前移动。



图表 28 语音 Loop Buffer 示意图

13.4 语音编码算法

Bee2 RCU 语音数据的采用率为 16Ksample/s，采样精度是 16bit。为了更有效的利用低功耗蓝牙的带宽传送语音数据，语音数据需要进行压缩编码。

Bee2 RCU 支持多种软件压缩编码算法，包括 SW mSCB encoder, SW SBC encoder, SW IMA/DVI ADPCM encoder 和 SW 16:3 ADPCM encoder。Bee2 RCU 可以通过 board.h 中的 VOICE_ENC_TYPE 宏定义来选择使用的软件编码算法，默认为 SW mSCB encoder。

```
/* voice encode type */
#define SW_MSBC_ENC      1  /* software msbc encode */
#define SW_SBC_ENC       2  /* software sbc encode */
#define SW_IMA_ADPCM_ENC 3  /* software IMA/DVI adpcm encode */
#define SW_OPT_ADPCM_ENC 4  /* software optimized adpcm encode */

#define VOICE_ENC_TYPE     SW_MSBC_ENC
```

13.5 语音交互流程

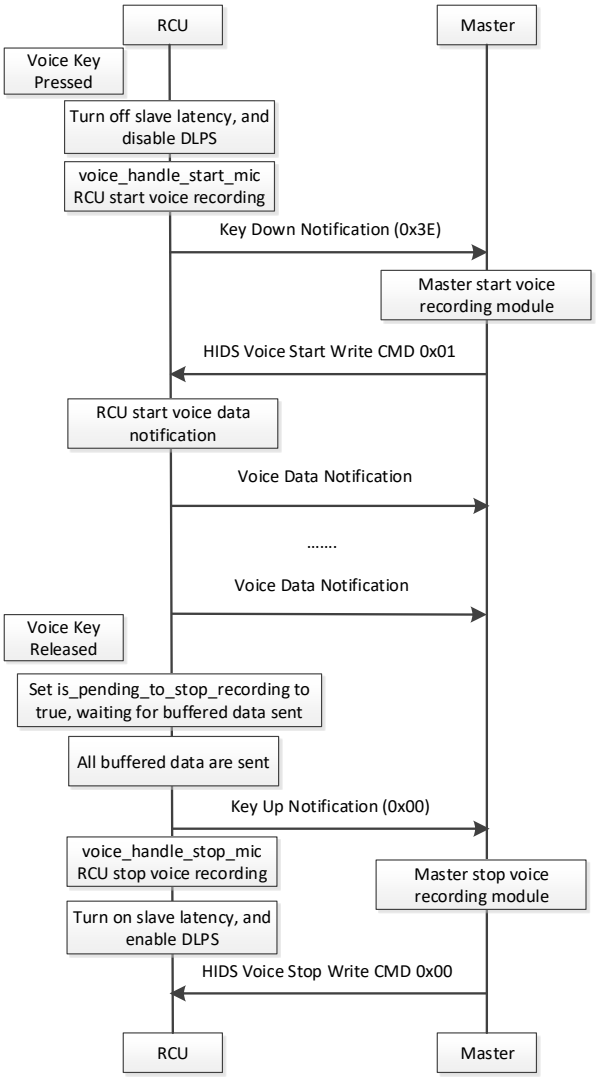
Bee2 RCU 支持三种语音交互流程：

- IFLYTEK_VOICE_FLOW
- HIDS_GOOGLE_VOICE_FLOW
- ATV_GOOGLE_VOICE_FLOW
- RTK_GATT_VOICE_FLOW

Bee2 RCU 可以通过 board.h 中的 VOICE_FLOW_SEL 宏定义来选择使用的语音交互流程。

13.5.1 IFLYTEK VOICE FLOW

IFLYTEK 语音交互流程是基于 HID Service，遥控器和主机端之间的语音交互流程如下：



图表 29 IFLYTEK VOICE FLOW

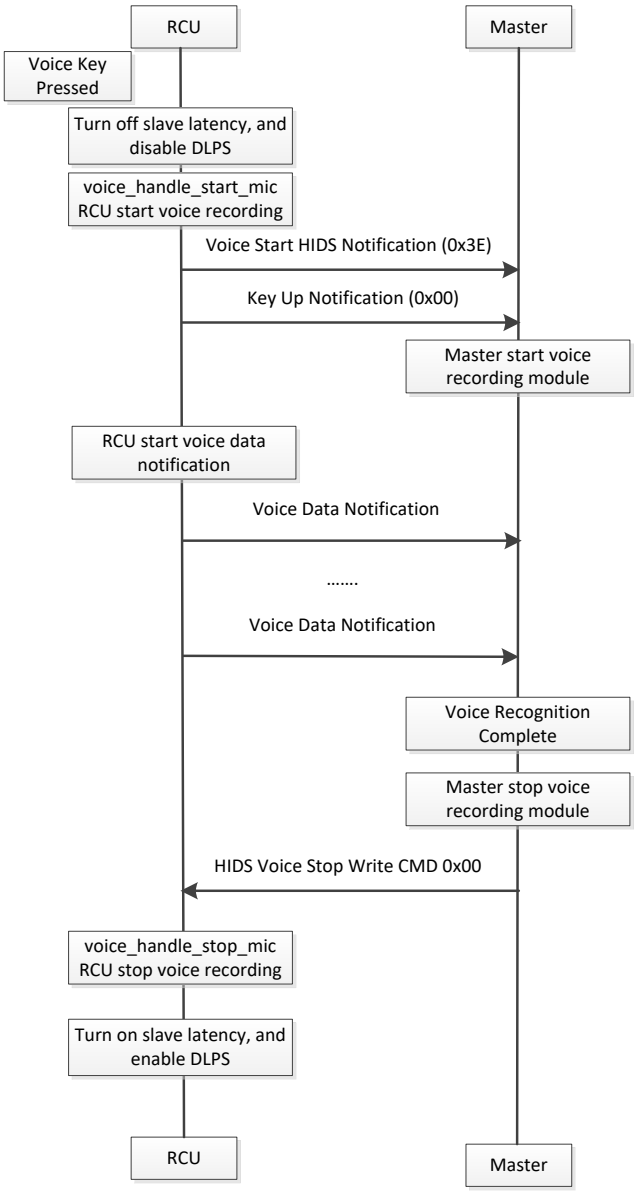
13.5.2 HIDS Google VOICE FLOW

HIDS Google Voice 交互流程和 IFLYTEK 语音交互流程类似，都是基于 HID Service，区别在于：

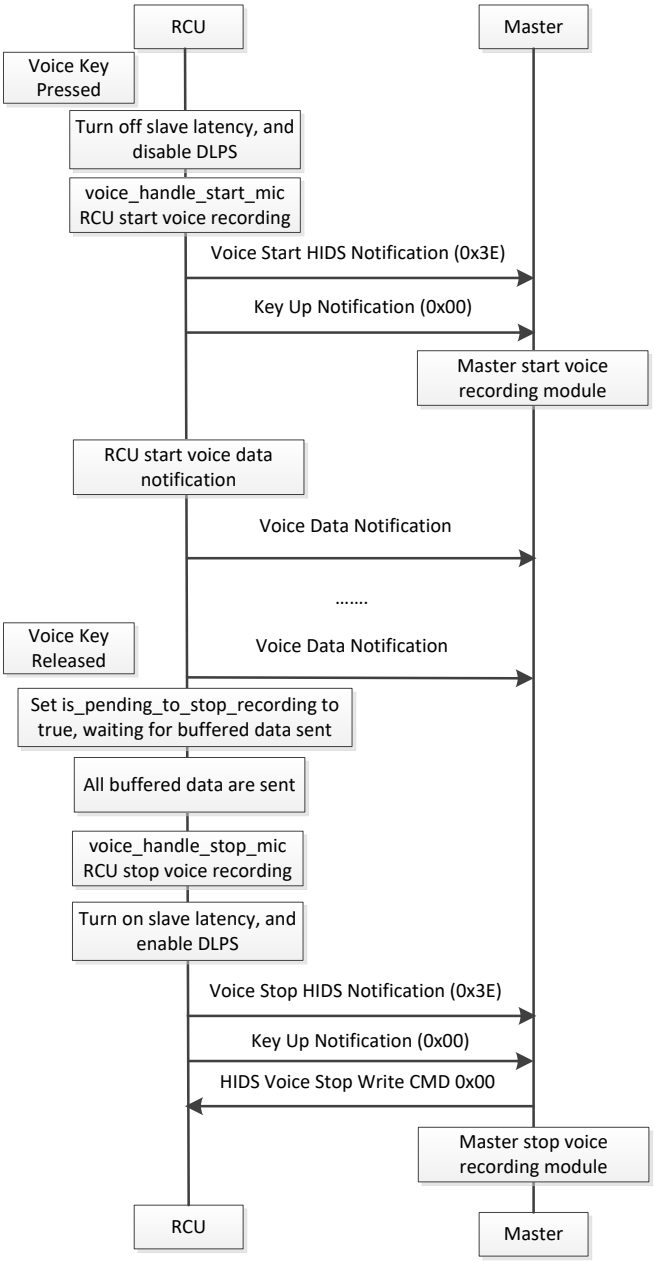
1. Voice Data Notification 开始发送，不需要收到主机端 voice start write cmd 之后进行，默认在 Voice Start Notification 之后，直接开始 Voice Data Notification；
2. 语音结束方式有两种：
   a) 语音数据传输之后，主机端的语音识别 server 会自动判别当前语音流程是否完成，若完成则发送 voice stop write cmd 停止 RCU 语音录音；
   b) 如果在收到主机端 voice stop write cmd 之前，用户松开了语音按键，则等待语音缓存数据都

发送完之后，主动停止语音录音，并发送 Voice Stop Notification，通知主机端结束语音。

遥控器和主机端之间的语音交互流程如下：



图表 30 HIDS Google VOICE FLOW – 1



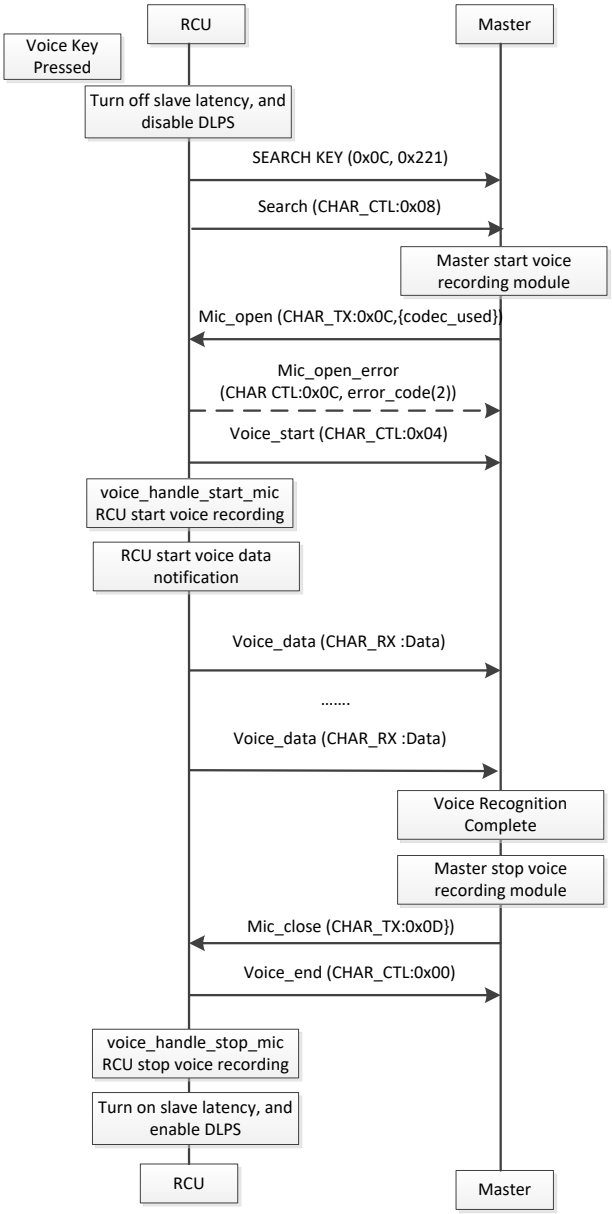
图表 31 HIDS Google VOICE FLOW – 2

13.5.3 ATV Google VOICE FLOW

从 Android 版本 8.0 开始，Google 有定义一套标准的 Voice Solution 方案，该方案基于 Google 标准的 ATV Voice Service 进行语音传输。目前最新的 spec 为 draft V0.5 版本。

遥控器使用 ATV Voice Service 来进行语音数据交互，具体规范请参见 Google ATV Voice 文档。Bee2 RCU 的 ATV 语音流程基本遵循上述 spec，在此基础上进行了语音传输效率的优化：在连接配对之后，RCU 会主动发起 MTU Exchange 的 request，之后使用较大的 MTU size 进行语音交互，一次 Voice Data 的 Notification size 是 134bytes。

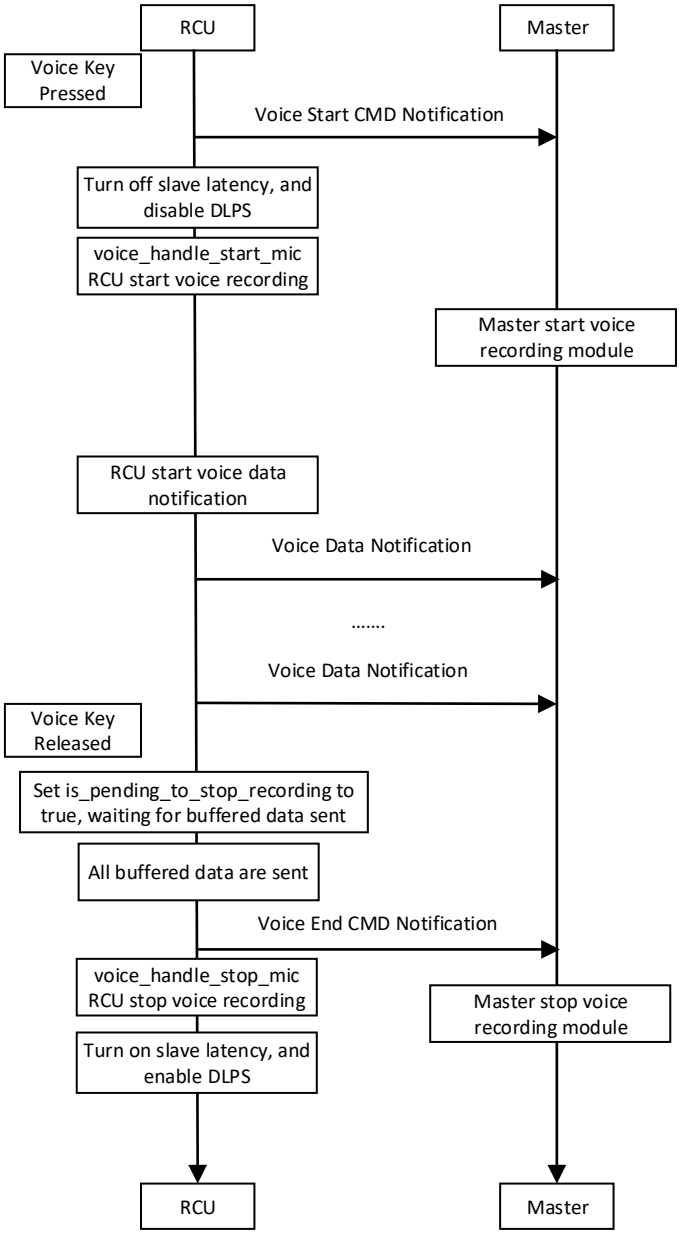
遥控器和主机端之间的语音交互流程如下：



图表 32 ATV Google VOICE FLOW

13.5.4 RTK GATT VOICE FLOW

RTK 语音交互流程是基于 Voice Service，Voice Service 相关内容请参见《AI 耳机语音传输协议(v0.3)》。遥控器和主机端之间的语音交互流程如下：



图表 33 RTK GATT VOICE FLOW

14 IR 红外

遥控器支持红外功能，包括红外学习和红外发送。红外学习支持 6 个按键的学习并且支持存储功能，红外发送可以发送学习到的红外波形，也可以发送用户自定义的红外数据和地址。发送时如果存在学习的波形，则学习的波形优先发送，若本地数据无效或者没有进行过红外学习就会发送本地预先设定的红外数据。

14.1 红外模块的配置

如若使用红外模块需要在 board.h 中打开定义红外模块的控制宏 IR_FUN,红外学习的功能由宏 IR_LEARN_MODE 来进行控制，前提是红外模块的宏 IR_FUN 已经打开。

```
/*******************************************************
*            IR Module Config
*******************************************************/
#define IR_FUN        0

#if IR_FUN

/*IR send config*/
#define IR_SEND_PIN           P2_3

#define IR_SEND_WAVE_MAX_SIZE   70

/*IR repeat code period config */
#define IR_REPEAT_CODE_PERIOD 108  /* 108 ms, unit ms */

/*IR learn module config*/
#define IR_LEARN_MODE    0

#if IR_LEARN_MODE

#define IR_LEARN_PIN          P2_2
#endif

/* IR_LEARN_TRIG_MODE need to be configured according to hardware */
#define IR_LEARN_TRIG_RISING_EDGE 0
#define IR_LEARN_TRIG_FALL_EDGE   1
#define IR_LEARN_TRIG_MODE          IR_LEARN_TRIG_FALL_EDGE

```

```
/*max key number can be storged*/
#define  IR_LEARN_MAX_KEY_NUM        0x06
#define  IR_WAVE_CONFIG_BASE_ADDR    IR_MODULE_FTL_PARAMS_BASE_ADDR
#define  IR_WAVE_DATA_BASE_ADDR      (IR_WAVE_CONFIG_BASE_ADDR + sizeof(uint32_t)/*config CRC value*/ + \
                                     sizeof(IR_LearnStorgeInfo)*IR_LEARN_MAX_KEY_NUM)
#define  IR_LEARN_WAVE_MAX_SIZE      IR_SEND_WAVE_MAX_SIZE

#endif
#endif
```

目前，公版的 RCU 红外学习实例是针对波长数据在 70 个数据以下的红外协议，可以根据宏 IR_LEARN_WAVE_MAX_SIZE 来进行调节。红外数据存储在 FTL 中，红外波形的数据的存储基地址可以使用宏 IR_WAVE_CONFIG_BASE_ADDR 和 IR_WAVE_DATA_BASE_ADDR 来进行调节，但是注意不要超出 FTL 的内存大小。数据的存储格式是红外学习数据的配置信息和 CRC 校验信息，从地址 IR_WAVE_DATA_BASE_ADDR 开始才是红外数据存储。红外发送的时候会从存储的地址读取指定的红外数据，直接进行发送。

14.2 红外模块的操作

红外学习模式的进入是通过组合按键触发的，按住按键 ENTER 和 LEFT 2s 不放会进入红外学习模式，此时指示灯会不断的闪烁，指示红外学习状态，此时超时定时器也开始计时 20s，若 20s 超时后会退出红外学习状态，指示灯会退出闪烁状态。在红外学习模中进行学习红外波形，可以按下指定的按键然后输入红外波形就可以了。按键按下时，指示灯保持常亮，红外学习成功指示灯会快速闪烁几下然后退出快闪。

若想在红外模式中快速退出红外学习模式，可以直接在模式下长按组合按键退出，LED 的指示是从闪烁退出到其他状态。

15 空中升级（OTA）

遥控器支持了私有 OTA Service 和 DFU Service 用于空中升级固件。RCU SDK 支持两种升级方式，一种是传统的升级方式，normal OTA；另一种是静默升级方式，silent OTA。

Normal OTA 的是需要遥控器进入一种 OTA 模式的升级方式，此时遥控器处于升级模式无法使用，只能在升级成功之后使用新的遥控器 image 功能。

Silent OTA 是遥控器在正常模式下进行升级的，遥控器在升级时可以正常使用。升级成功之后，遥控器重启后，执行新升级的软件版本。

OTA 的详细原理请参考文档《RTL8762C OTA User Manual》。

15.1 Normal OTA 方式

Normal OTA 方式是通过 OTAservice 来实现的。OTA 模式是独立运行在 ROM 区的一段代码，所以遥控器进入 OTA 模式后，对端设备需要重新搜索到 OTA 模式下的遥控器并建立连接，然后根据 OTA Service 的要求实现 OTA 升级的动作。OTA 升级完成后遥控器会自动重启，并执行更新后的固件程序。

Ota service UUID 如下:

```
const uint8_t GATT_UUID_OTA_SERVICE[16] = { 0x12, 0xA2, 0x4D, 0x2E, 0xFE, 0x14, 0x48, 0x8e, 0x93, 0xD2, 0x17, 0x3C, 0xFF, 0xD0, 0x00, 0x00};
```

OTA Service 下有如下 Characteristics。

```
#define GATT_UUID_CHAR_OTA                  0xFFD1
#define GATT_UUID_CHAR_MAC                  0xFFD2
#define GATT_UUID_CHAR_PATCH                0xFFD3
#define GATT_UUID_CHAR_APP_VERSION          0xFFD4
#define GATT_UUID_CHAR_DEVICE_INFO          0xFFF1
#define GATT_UUID_CHAR_IMAGE_VERSION        0xFFE0
```

0xFFD1 为 OTA 命令接收使用，属性为 Write no response，对端通过 OTA Service 向 0xFFD1 写入 0x1，遥控器收到这个值后会切换到 OTA 模式。

0xFFD2 作用是对端读取遥控器的 MAC 地址（BD Address）。

0xFFD3 作用是对端读取遥控器的 Patch 版本号。

0xFFD4 作用是对端读取遥控器的 APP 版本号。

0xFFF1 作用是对端读取遥控器设备的信息，比如是否支持加密或者是否 buffer check 等；

0xFFE0 作用是对端读取遥控器的地址和版本号；

15.2 Silent OTA 方式

Silent OTA 是在遥控器正常使用的情况下进行升级的一种方式，同时需要 OTA service 和 DFU service 的支持。升级前，对端通过 OTA service 读取遥控器设备的信息，升级时对端设备通过 DFU service 来进行升级流程的控制和数据的传输。

DFU service UUID 如下：

```
const uint8_t  SILENCE_GATT_UUID128_DFU_SERVICE[16] = {0x12, 0xA2, 0x4D, 0x2E, 0xFE, 0x14, 0x48, 0x8e, 0x93, 0xD2, 0x17, 0x3C, 0x87, 0x62, 0x00, 0x00};
```

Dfu service 定义了两个 characteristics，Data Characteristic 和 Control Point Characteristic 。Data Characteristic 接受 image 的数据通道，属性 write no response；Control Point Characteristic 接收控制指令的通道，属性 write/notification。

两个属性的 UUID 如下：

```
1. #define GATT_UUID128_DFU_PACKET 0x12, 0xA2, 0x4D, 0x2E, 0xFE, 0x14, 0x48, 0x8e, 0x93, 0xD2, 0x17, 0x3C, 0x87, 0x63, 0x00, 0x00
2. #define GATT_UUID128_DFU_CONTROL_POINT 0x12, 0xA2, 0x4D, 0x2E, 0xFE, 0x14, 0x48, 0x8e, 0x93, 0xD2, 0x17, 0x3C, 0x87, 0x64, 0x00, 0x00
```

16 电量检测及低电保护

RCU SDK 支持 ADC 电量检测和 LPC 低电量保护功能。电量检测是在蓝牙连接之后，对端设备通过电池服务读取的电池信息，可以是电池电量的绝对电压值或者电压百分比。

16.1 电量检测的配置

```
/*******************************************************
*          Battery Module Config
*******************************************************/
#define BAT_EN    1

#if BAT_EN

#define BAT_LPC_EN    0 /* 1 enable lpc, 0 disable */
#if BAT_LPC_EN
#define BAT_LPC_CONFIG            LPC_2000_mV /*lpc detect bat threshold value*/
#endif

#define BAT_LOW_POWER_THRESHOLD    2000 /*2.0v threshold for rcu system*/
#define BAT_ADC_DETECT_THRESHOLD   (BAT_LOW_POWER_THRESHOLD + 200)
#define BAT_LOW_POWER_INDICATE     1 /*1 led indicate enable, 0 disable */

#endif
```

整个电量模块的控制通过宏 BAT_EN 来实现的。通过配置 BAT_LOW_POWER_THRESHOLD 和 BAT_ADC_DETECT_THRESHOLD 来配置低电量模式的阈值和电池检测模式的阈值。BAT_LOW_POWER_INDICATE 用来指示 RCU 处于低电量时是否需要 LED 灯来进行指示。BAT_LPC_EN 用来选择是否使用 LPC 硬件模块。BAT_LPC_CONFIG 用来设置 LPC 比较器的电压阈值。

16.2 电量模块的工作流程

低电量模式可以通过 LPC 硬件模块触发，也可以通过按键方式触发，默认使用按键方式触发。低电量的默认阈值是 2.0V，如果低电量模式由 LPC 硬件模块触发，当电量高于这个阈值时，RCU 处于正常工作状态，当电压低于这个阈值时，LPC 中断会被触发，执行一次低电检测操作；如果低电量模式由按键方式触发，VBAT_DETECT_KEY_CNT 次按键会触发一次低电检测操作。低电检测操作中，ADC 会进行精确电压采集，采集 4 次去掉最大和最小值，然后取剩余的平均值。若平均值低于设置的阈值就需要系统进入低电量模式。

进入低电量模式之前，若系统处于蓝牙连接或者广播状态，需要将系统断线或者停止广播回到 Idle 状态，然后 RCU 进入低电量模式。此时操作按键会检测系统的电压，若此时电压低于指定的电池检测阈值，按键不响应按键不会响应；若此时系统电池的电压高于指定的电池电压检测阈值，就会退出低电量模式，然后执行刚才的按键动作。

17 LED 指示灯

RCU SDK 支持多功能 LED 模块，该模块支持多个 LED 灯同时闪烁，并且支持单个 LED 灯多个事件按优先级闪烁。

17.1 LED 的配置

整个 LED 模块的控制通过宏 LED_EN 来进行控制。LED 的数量通过 LED_NUM_MAX 来进行控制，目前默认支持一个。LED_ON_LEVEL_TRIG 用来选择 LED 亮时对应的驱动电平状态。

```
/*************************************************************
*          LED Module Config
*************************************************************/

#define LED_EN    1

#if LED_EN

#define  LED_NUM_MAX   0x01
#define  LED_INDEX(n)   (n<<8)
/*uint16_t, first byte led index, last byte led pin*/
#define  LED_1         (LED_INDEX(0) | P2_4)

/* voltage level to trigger LED On action */
#define LED_ON_LEVEL_HIGH 0
#define LED_ON_LEVEL_LOW  1

#define LED_ON_LEVEL_TRIG LED_ON_LEVEL_HIGH

#endif
```

17.2 LED 使用接口

LED 的使用接口有四个，定义在 led_driver.h 中，分别是 LED_ON，LED_OFF, LED_BLINK 和 LED_BLINK_EXIT。

```
#if LED_EN
#define LED_ON(index)              led_blink_start(index, LED_TYPE_ON, 0)
#define LED_OFF(index)             led_blink_exit(index, LED_TYPE_ON)
#define LED_BLINK(index, type, n)      led_blink_start(index, type, n)
```

```
#define LED_BLINK_EXIT(index, type)    led_blink_exit(index, type)
#else
#define LED_ON(index)
#define LED_OFF(index)
#define LED_BLINK(index, type, n)
#define LED_BLINK_EXIT(index, type)
#endif
```

LED_ON 和 LED_OFF 主要是指示按键的状态亮和灭。

LED_BLINK 和 LED_BLINK_EXIT 这两个宏主要是控制 LED 的闪烁和退出。LED_BLINK 有三个参数，第一个是 LED 的索引，定义在 board.h 中；第二个参数是 LED 闪烁的类型，定义在 led_driver.h 中；第三个是 LED 闪烁的次数，若填写 0 则闪烁不停止。若填写指定的数值，则闪烁次数根据填写的数据而定，闪烁次数最大 255。

目前默认添加的闪烁类型有 7 种，分布在四个模块：红外，低电量，蓝牙和按键。若需要添加新的 LED 类型需要添加三个地方，一个是 LED type，一个是 LED 闪烁的 bit map，最后一个是在 led_driver.c 中添加执行分支。

18 量产测试

Bee2 RCU 支持量产测试模式，包括 HCI UART 测试模式，Data UART 测试模式，Single Tone 测试模式，及快速配对测试模式。相关量产测试模式的详细介绍请参见《RTL8752C/RTL8762C RCU MP Test Mode Design Spec》 。Bee2 RCU SDK 中，在 board.h 中有相关的宏定义来支持量产测试模式。

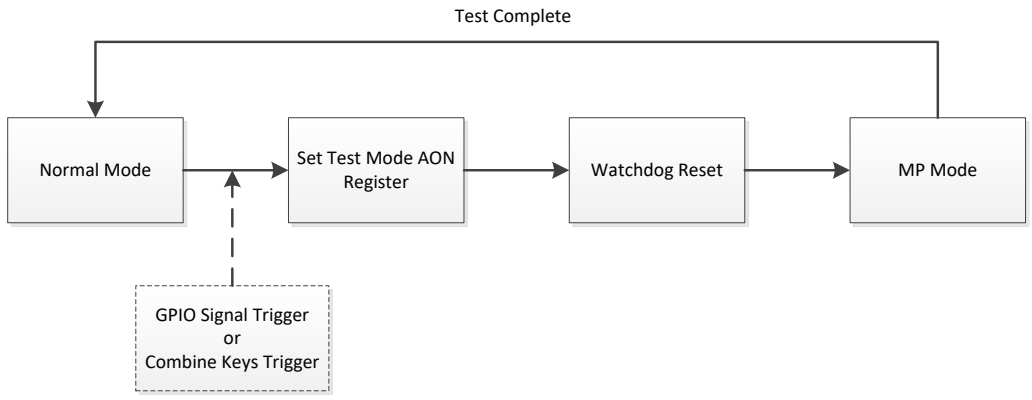
```
#define FEATURE_SUPPORT_MP_TEST_MODE 1  /* set 1 to enable MP test */

#define MP_TEST_MODE_SUPPORT_HCI_UART_TEST      1  /* set 1 to support HCI Uart Test Mode */
#define MP_TEST_MODE_SUPPORT_DATA_UART_TEST     1  /* set 1 to support Data Uart Test Mode */
#define MP_TEST_MODE_SUPPORT_SINGLE_TONE_TEST   1  /* set 1 to support SingleTone Test Mode */
#define MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST     1  /* set 1 to support Fast Pair Test */
#define MP_TEST_MODE_SUPPORT_AUTO_K_RF          0  /* set 1 to support Auto K RF */
#define MP_TEST_MODE_SUPPORT_DATA_UART_DOWNLOAD 0  /* set 1 to support Data UART download */

#define MP_TEST_MODE_TRIG_BY_GPIO           0x0001  /* GPIO signal while power on to trigger MP test mode */
#define MP_TEST_MODE_TRIG_BY_COMBINE_KEYS   0x0002  /* Combine keys to trigger MP test mode */

#define MP_TEST_MODE_TRIG_SEL               (MP_TEST_MODE_TRIG_BY_GPIO | MP_TEST_MODE_TRIG_BY_COMBINE_KEYS)
```

Bee2 通过 Test Mode AON 寄存器和看门狗重启的方式，从正常模式切换成量产测试模式，切换量产测试模式的流程如下：



图表 34 切换量产测试模式流程

Bee2 RCU 提供参考的量产测试流程，包括：遥控器烧录、PCBA 级测试和整机功能测试。量产测试流程的详细介绍请参见《RTL8752C/RTL8762C RCU MP Test Sample Flow》 。

19 参考文献

[1] RTL8762C OTA User Manual

[2] RTL8762C RCU MP Test Mode Design Spec

[3] RTL8762C RCU MP Test Sample Flow

[4] AI 耳机语音传输协议(v0.3)