RTL8762E 语音遥控器设计说明书

V1.1 by Realtek

2022/04/15

修订历史

日期	版本	修改
2021/07/01	V1.0	First version of RTL8762E RCU design specification
2021/04/15	V1.1	Add software EQ

目录

图表

1 概述

本文主要介绍 RTL8762E 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 RTL8762E 语音遥控器为 Bee3 RCU。

2 功能特性

- 支持 BLE 5.1
- 支持自动配对
- 支持蓝牙开关机
- 支持按键扫描功能，最多可扩展到 240 个按键（12x20）
- 支持语音输入与传输
- 支持 IR 发送
- 支持 IR 学习
- 支持 LED 指示
- 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：1.8~3.3V (2 节干电池供电)

3 GPIO 配置

Bee3 RCU SDK 默认支持三种 RCU 硬件配置，在 board.h 中，通过宏定义 RCU_HD_PLATFORM_SEL 进行配置：

```
#define H_DEMO_RCU              0  /* H Demo RCU */
#define R_DEMO_RCU              1  /* R Demo RCU */
#define G_MIN_DEMO_RCU          2  /* Google Minimal Demo RCU */

#define RCU_HD_PLATFORM_SEL     H_DEMO_RCU  /* RCU Hardware Platform selection */
```

- H_DEMO_RCU：适配 RTL8762E EVB (Evaluation Board)硬件设计
- R_DEMO_RCU：适配 Realtek 公版 Bee3 RCU PCBA 硬件设计
- G_MIN_DEMO_RCU：适配 Google RCU 硬件设计

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

- H_DEMO_RCU

```
#define KEYPAD_ROW_SIZE      4
#define KEYPAD_COLUMN_SIZE   4

#define ROW0            P3_0
#define ROW1            P3_1
#define ROW2            P1_0
#define ROW3            P1_1

#define COLUMN0         P4_0
#define COLUMN1         P4_1
#define COLUMN2         P4_2
#define COLUMN3         P4_3
```

- R_DEMO_RCU

```
#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
```

- G_MIN_DEMO_RCU

```
#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
```


3.3 Voice MIC GPIO

- H_DEMO_RCU，通过外接 MIC 子板，可支持 AMIC 和 DMIC；

```
#define AMIC_MIC_N_PIN          P2_6  /* MIC_N is fixed to P2_6 */
#define AMIC_MIC_P_PIN          P2_7  /* MIC_P is fixed to P2_7 */
#define AMIC_MIC_BIAS_PIN       H_0   /* MICBIAS is fixed to H_0 */

#define DMIC_CLK_PIN      P2_6  /* dual DMICs share clock PIN, support PINMUX */
```

```
#define DMIC_DATA_PIN     P2_7  /* dual DMICs share data PIN, support PINMUX */
```

- R_DEMO_RCU，PCBA 上件 AMIC

```
#define AMIC_MIC_N_PIN          P2_6  /* MIC_N is fixed to P2_6 */
#define AMIC_MIC_P_PIN          P2_7  /* MIC_P is fixed to P2_7 */
#define AMIC_MIC_BIAS_PIN       H_0   /* MICBIAS is fixed to H_0 */
```

- G_MIN_DEMO_RCU，通过外接 MIC 子板，可支持 AMIC 和 DMIC；

```
#define AMIC_MIC_N_PIN          P2_6  /* MIC_N is fixed to P2_6 */
#define AMIC_MIC_P_PIN          P2_7  /* MIC_P is fixed to P2_7 */
#define AMIC_MIC_BIAS_PIN       H_0   /* MICBIAS is fixed to H_0 */

#define DMIC_CLK_PIN      P2_0  /* dual DMICs share clock PIN, support PINMUX */
#define DMIC_DATA_PIN     P2_1  /* dual DMICs share data PIN, support PINMUX */
```

3.4 IR GPIO

- H_DEMO_RCU

```
#define IR_SEND_PIN         P2_4
#define IR_LEARN_PIN        P2_5
```

- R_DEMO_RCU

```
#define IR_SEND_PIN         P2_3
#define IR_LEARN_PIN        P2_2
```

- G_MIN_DEMO_RCU

```
#define IR_SEND_PIN         P2_3
#define IR_LEARN_PIN        P2_2
```

3.5 LED GPIO

- H_DEMO_RCU

```
#define  LED_PIN   P0_2
```

- R_DEMO_RCU

```
#define  LED_PIN   P2_4
```

- G_MIN_DEMO_RCU

```
#define  LED_PIN   P2_4
```

3.6 MP Test GPIO

- H_DEMO_RCU

```
#define MP_TEST_TRIG_PIN_1      P5_1
#define MP_TEST_TRIG_PIN_2      P5_2

#define MP_TEST_UART_TX_PIN      P3_0
#define MP_TEST_UART_RX_PIN      P3_1
```

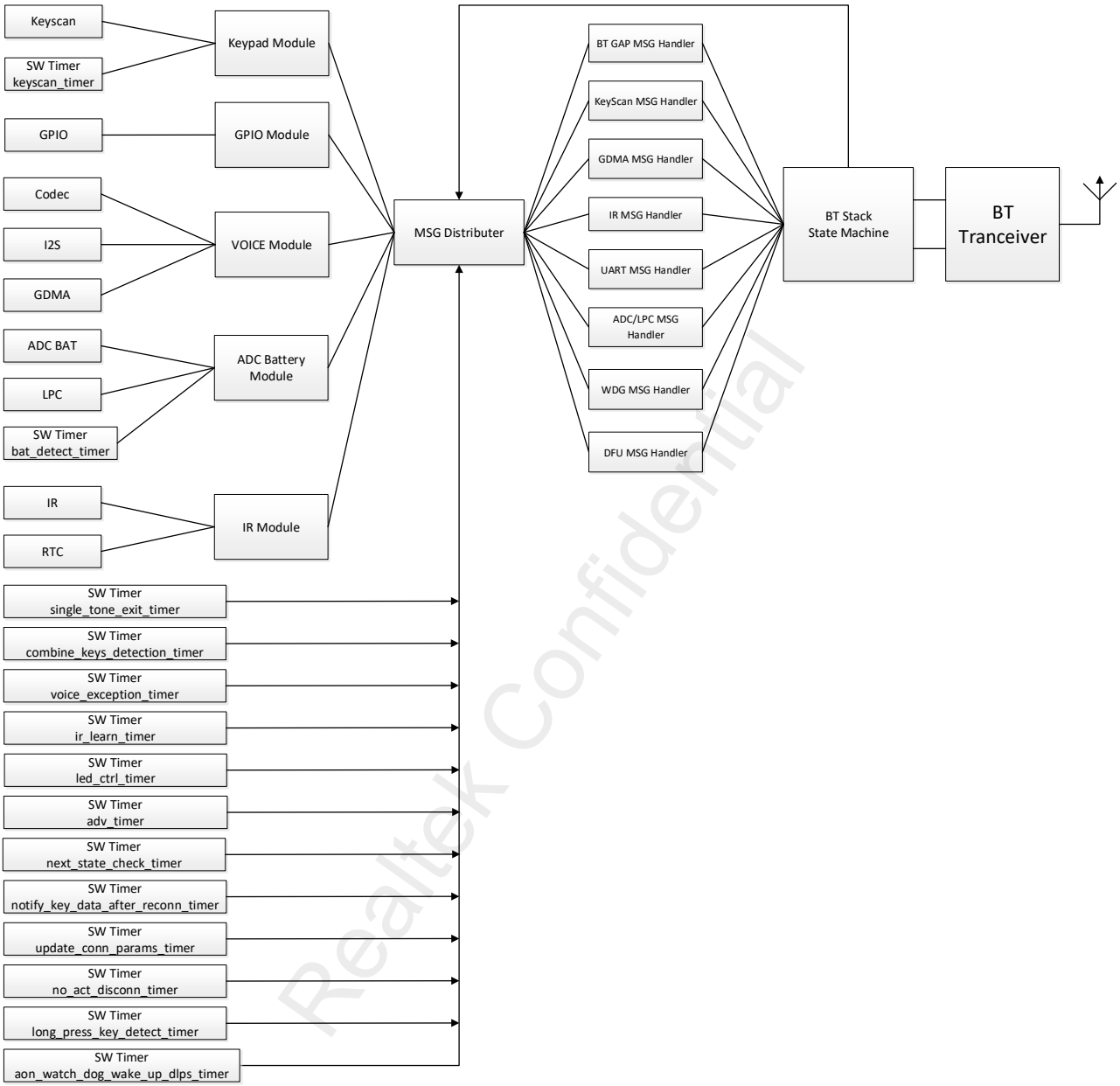
- R_DEMO_RCU

```
#define MP_TEST_TRIG_PIN_1      P3_2
#define MP_TEST_TRIG_PIN_2      P3_3

#define MP_TEST_UART_TX_PIN      P3_0
#define MP_TEST_UART_RX_PIN      P3_1
```

- G_MIN_DEMO_RCU

```
#define MP_TEST_TRIG_PIN_1      P3_2
#define MP_TEST_TRIG_PIN_2      P3_3

#define MP_TEST_UART_TX_PIN      P3_0
#define MP_TEST_UART_RX_PIN      P3_1
```

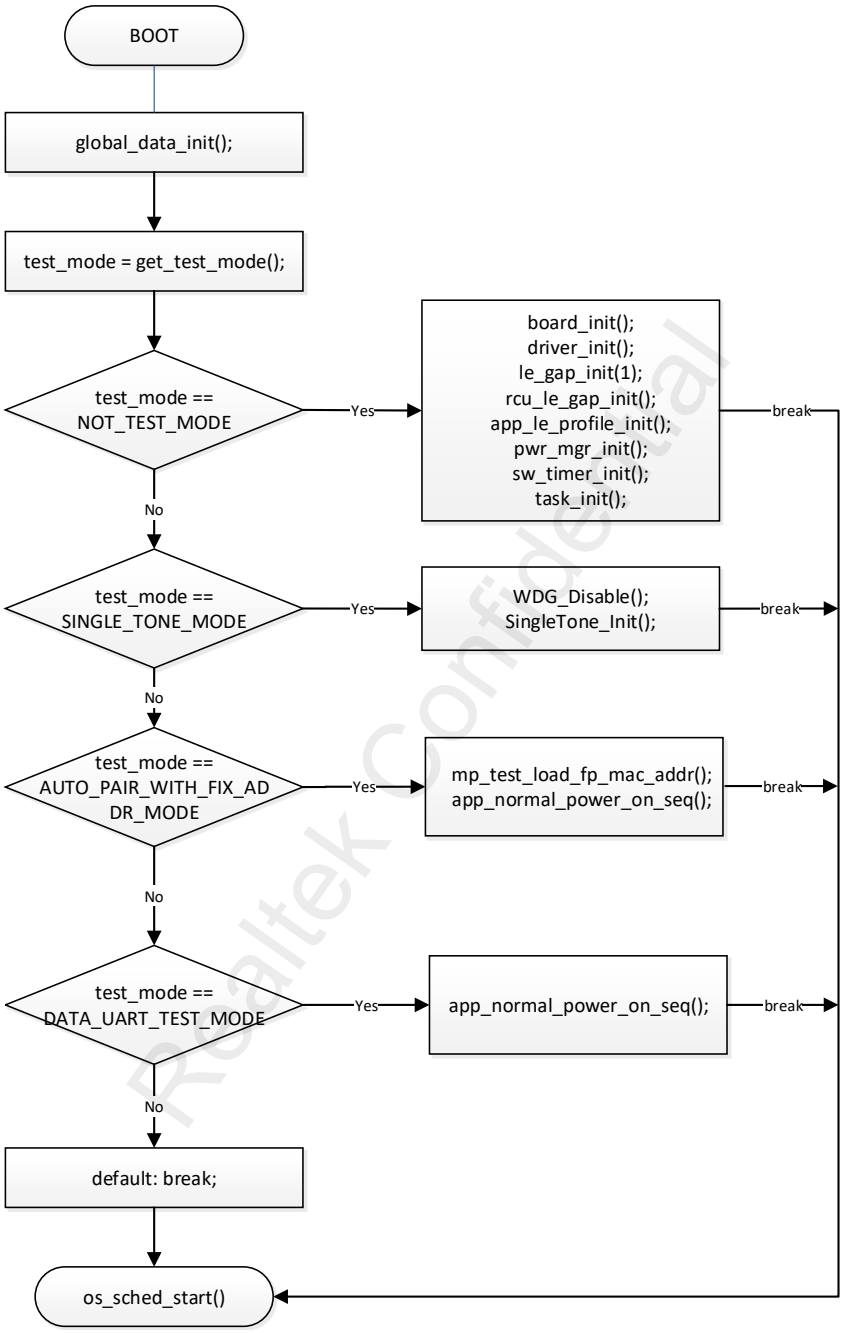
4 系统框图



图表 1 系统框图

5 初始化过程

5.1 系统启动流程

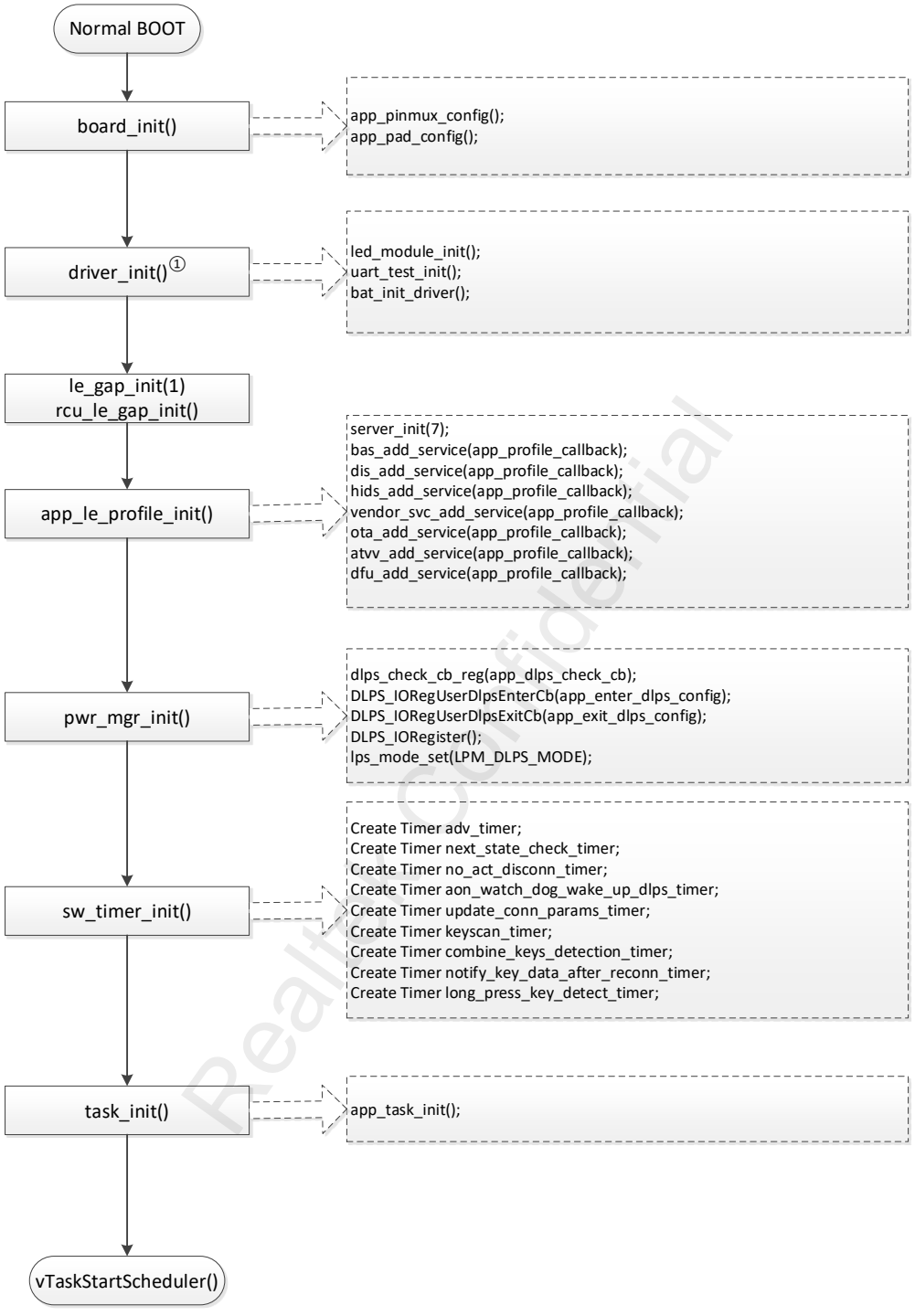


图表 2 系统启动流程图

系统模式

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

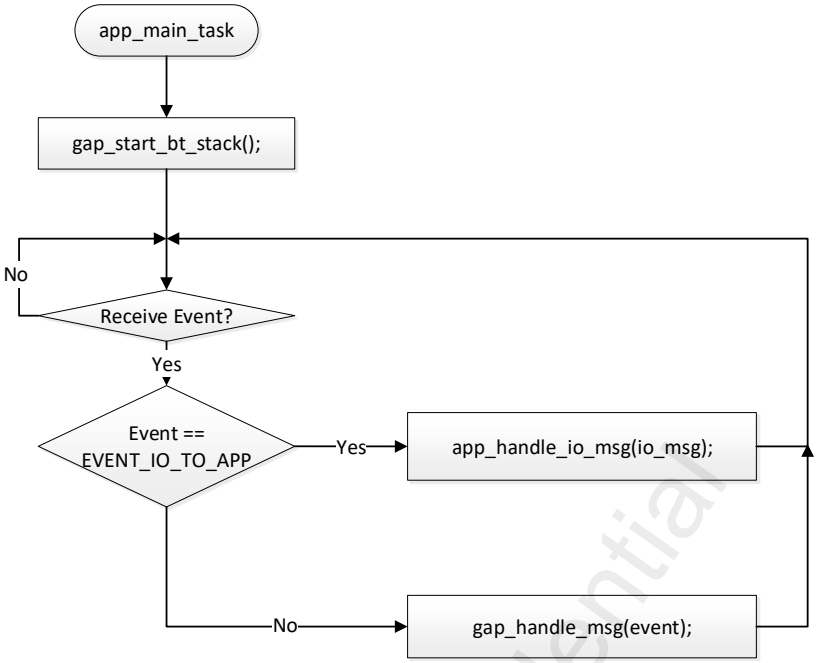
5.2 正常启动流程



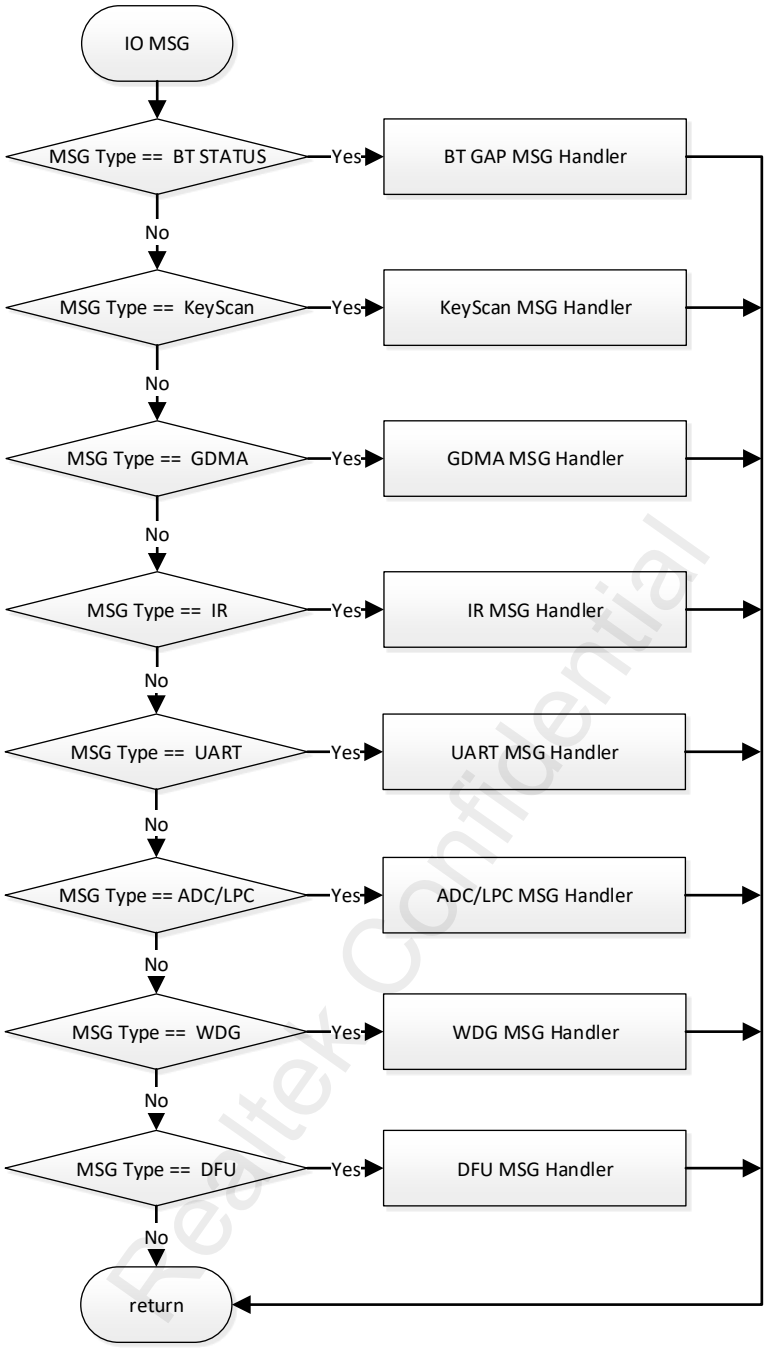
图表 3 正常启动流程图

注①：driver_init()函数中未包含 keyscan_init_driver()函数，而是同 keyscan_nvic_config()一起放在 app_nvic_config()中，防止上电到 nvic enable 期间可能产生的 keyscan 数据影响按键模块的正常使用。

6 APP TASK



图表 4 App task flow



图表 5 APP IO Message Handler

7 遥控器状态

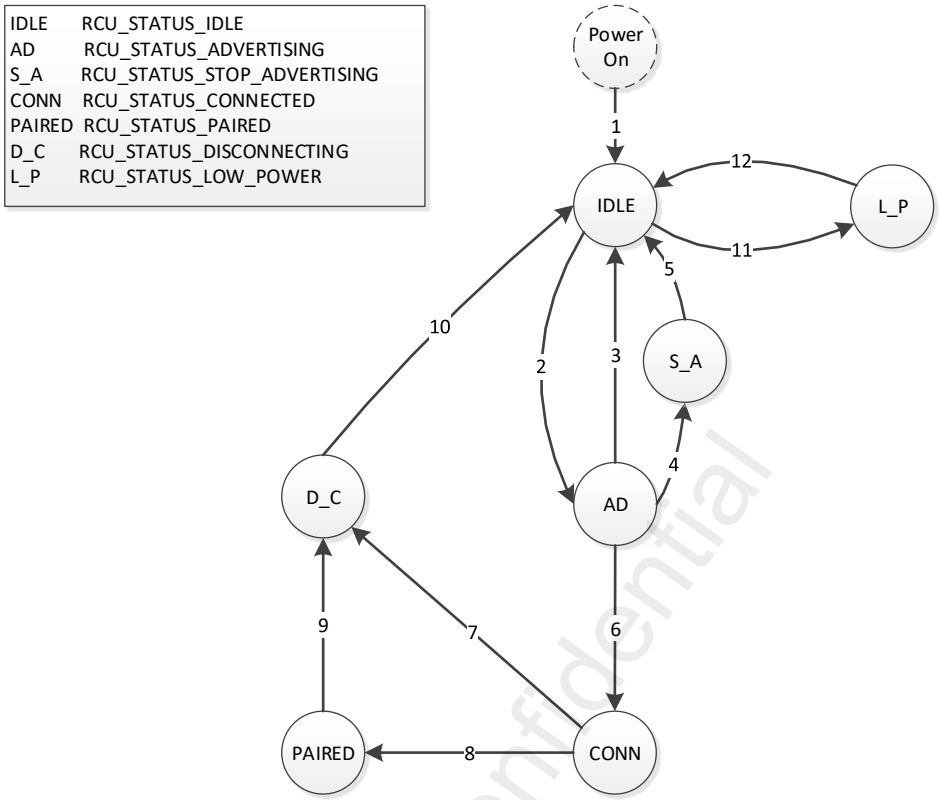
7.1 遥控器状态表

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 14 个软件定时器，用途如下表所示。

No.	SW Timer	说明
1	adv_timer	用于超时停止广播
2	next_state_check_timer	用于处理配对异常超时问题
3	no_act_disconn_timer	用于长时间无操作，遥控器主动断开 BLE 连接
4	update_conn_params_timer	用于在超时后进行连接参数更新操作
5	aon_watch_dog_wake_up_dlps_timer	用于 AON Watch Dog 定时喂狗
6	keyscan_timer	用于检测 KeyScan 按键状态
7	combine_keys_detection_timer	用于检测组合按键状态
8	notify_key_data_after_reconn_timer	用于回连后延时补发按键键值
9	voice_exception_timer	用于 ATV 语音流程异常停止
10	long_press_key_detect_timer	用于检测长按键保护的触发
11	ir_learn_timer	用于红外学习超时控制
12	led_ctrl_timer	用于 LED 控制
13	bat_detect_timer	用于定时检测电池电量
14	single_tone_exit_timer	用于超时退出 SingleTone 测试模式

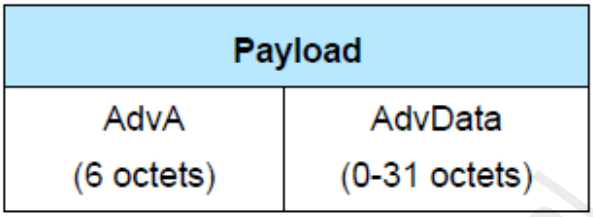
图表 11 SW timer 使用介绍

9 BLE 广播

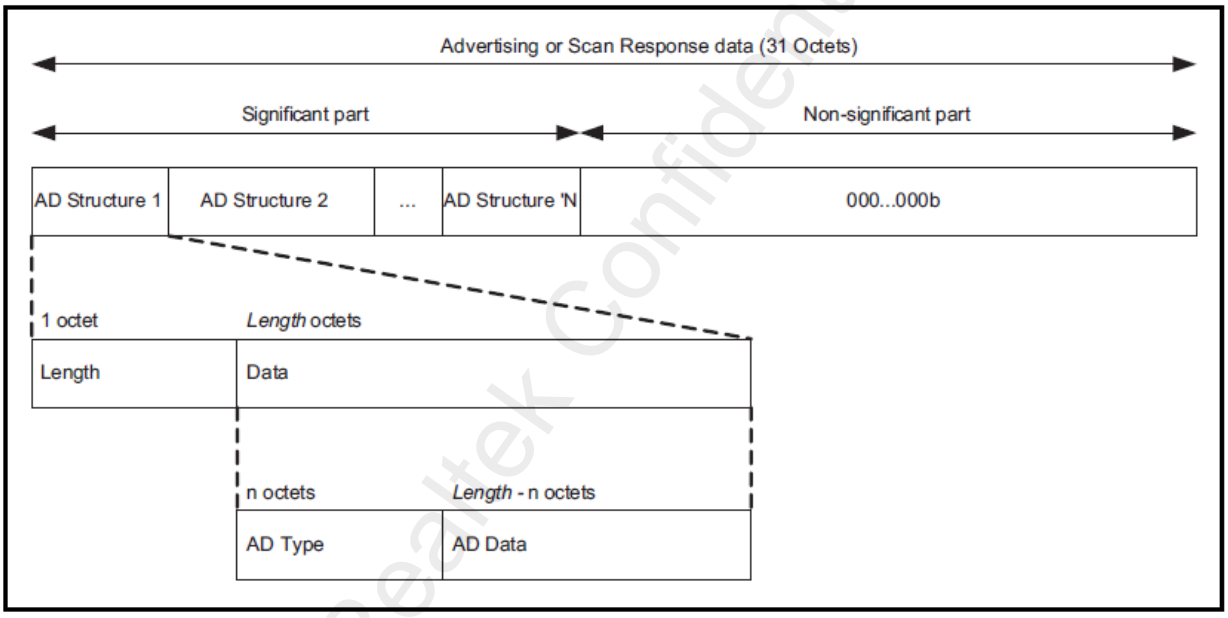
9.1 广播包格式

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 格式如图表 13 所示。

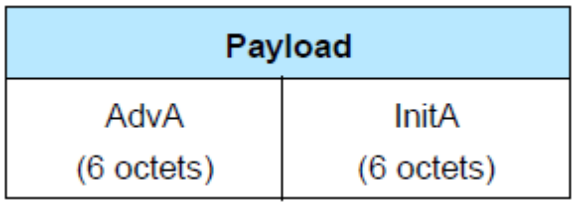


图表 12 ADV_IND PDU Payload



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。



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;
```

遥控器有五种有效广播类型：ADV_UNDIRECT_PAIRING，ADV_DIRECT_HDC，ADV_UNDIRECT_RECONNECT，ADV_UNDIRECT_PROMPT 和 ADV_UNDIRECT_POWER。分别对应配对模式，Undirected 回连模式，Directed 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），广播超时时间通过宏 ADV_UNDIRECT_PAIRING_TIMEOUT 进行设置，默认为 30s。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_REMOVE_LINK_KEY_BEFORE_PAIRING 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_REMOVE_LINK_KEY_BEFORE_PAIRING 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_REMOVE_LINK_KEY_BEFORE_PAIRING 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 (byte11~23)	Vendor Information Field (byte24~29)
0x02,0x01,0x05	0x03,0x03, 0x12,0x18	0x03,0x19,0x80,0x01	0x0b,0x09, 'B', 'e', 'e', '3', ' ', ' R', ' ', 'R', 'C', 'U	0x05,0xff,0xXX,0xXX(VID),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。

根据 board.h 中宏定义 FEATURE_SUPPORT_PRIVACY，遥控器可以选择是否支持解析对端 Privacy Address 的功能，默认打开 FEATURE_SUPPORT_PRIVACY：

1. FEATURE_SUPPORT_PRIVACY 关闭，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。
2. FEATURE_SUPPORT_PRIVACY 打开，遥控器使用 Undirect Advertising + White List 的方式进行回连，Undirect Advertising 广播包内容和配对广播包内容相同，超时时间通过宏定义 ADV_UNDIRECT_RECONNECT_TIMEOUT 进行设置，默认为 3s。

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，任意按键均

会触发遥控器进入回连模式。

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），广播超时时间通过宏定义 ADV_UNDIRECT_PROMPT_TIMEOUT 进行设置，默认为 5 秒。

程序中，通过宏 FEATURE_SUPPORT_UNDIRECT_PROMPT_ADV 来控制是否支持发送提示广播包。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，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 (byte11~23)	Vendor Information Field (byte24~29)
0x02,0x01,0x04	0x03,0x03, 0x12,0x18	0x03,0x19,0x80,0x01	0x0b,0x09, 'B', 'e', 'e', '3', ' ', ' R', ' ', 'R', 'C', 'U	0x05,0xff,0xXX,0xXX(VID),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），广播超时时间通过宏定义 ADV_UNDIRECT_POWER_TIMEOUT 进行设置，默认为 12 秒。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。

在如下情况会进入开机模式： 在蓝牙断线状态，按开机键，会进入开机模式发送开机广播包。

开机广播包格式：

Flag field（byte0~2）	Service Field（byte3~6）	Appearance Field – Device type (byte7~10)	Vendor Information Field (byte23~28)
0x02,0x01,0x04	0x03,0x03, 0x12,0x18	0x03,0x19,0x80,0x01	0x0D,0xFF,0x5D,0x00,0x03,0x00,0x01,0xXX(index), 0xXX,0xXX, 0xXX, 0xXX, 0xXX,0xXX,(TV BD Address)

图表 17 开机广播包格式

9.3 广播参数设置

9.3.1 设备名称

在 Bee3 RCU SDK 中，可以通过宏 C_DEVICE_NAME 和 C_DEVICE_NAME_LEN 进行 BLE 设备名称的定义。

```
#if (RCU_HD_PLATFORM_SEL == R_DEMO_RCU)
#define C_DEVICE_NAME           'B','e','e','3',' ','R',' ','R','C','U'
#define C_DEVICE_NAME_LEN       (10+1)  /* sizeof(C_DEVICE_NAME) + 1 */

#elif (RCU_HD_PLATFORM_SEL == H_DEMO_RCU)
#define C_DEVICE_NAME           'B','e','e','3',' ','H',' ','R','C','U'
#define C_DEVICE_NAME_LEN       (10+1)  /* sizeof(C_DEVICE_NAME) + 1 */

#elif (RCU_HD_PLATFORM_SEL == G_MIN_DEMO_RCU)
#define C_DEVICE_NAME           'R','e','m','o','t','e','G','1','0'
#define C_DEVICE_NAME_LEN       (9+1)  /* sizeof(C_DEVICE_NAME) + 1 */

#endif
```

9.3.2 VID 和 PID

在 Bee3 RCU SDK 中，可以通过宏 VID 和 PID 进行 VID、PID 的定义。

```
#define VID  0x005D
#define PID  0x0001
```

10 连接和配对

10.1 连接配对流程

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 Directed 回连广播无法建立连接，如果发送 Undirected 回连广播或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，遥控器发送配对广播可建立连接，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。

10.2 连接参数更新

为了降低连接功耗，在连接绑定之后，遥控器会主动发送更新连接参数请求，请求主机端使用指定的连接参数。根据目前是走配对还是回连流程，RCU 主动更新连接的策略会有所不同：

1. 配对流程：在配对成功之后，开启定时器，在 UPDATE_CONN_PARAMS_TIMEOUT 之后进行参数更新请求，以保证 GATT 服务发现及 CCCD 使能流程可以快速完成；
2. 回连流程：
    a. 语音按键触发回连流程：回连之后如果语音在进行，则不允许立即更新参数请求，开启定时器，在 UPDATE_CONN_PARAMS_TIMEOUT 之后进行判断，若语音仍在进行，则进行 timer restart 动作，否则发送参数更新请求；
    b. 普通按键触发回连流程：回连成功之后，立即发送参数更新请求；

在 swtimer.h 中，可以通过 UPDATE_CONN_PARAMS_TIMEOUT 配置参数更新 timeout 时间，默认为 5 秒。

```
#define UPDATE_CONN_PARAMS_TIMEOUT 5000  /* 5s */
```

在 BEE3 RCU SDK 中，通过 rcu_application.h 的宏定义进行连接参数的配置。设默认应用方案使用的连接参数为：Connection Interval 15ms，Slave Latency 49，Supervision Timeout Period 5s。

```
#define RCU_CONNECT_INTERVAL          0x0c  /*0x0c * 1.25ms = 15ms*/
```

```
2. #define RCU_CONNECT_LATENCY          49
3. #define RCU_SUPERVISION_TIMEOUT      5000  /* 5s */
```

10.3 Privacy Feature 支持

在 board.h 中设置宏 FEATURE_SUPPORT_PRIVACY 进行 Privacy Feature 的打开和关闭，默认开启该功能。打开 FEATURE_SUPPORT_PRIVACY 后，支持对端设备为 Resolvable Random Address 的解析和回连操作：

1. 打开 FEATURE_SUPPORT_PRIVACY，RCU 使用 Undirect Adv + White List 进行回连，回连超时为 1 次 3 秒，预期可以回连开启 Privacy 功能的对端设备；
2. 关闭 FEATURE_SUPPORT_PRIVACY，RCU 使用 High Duty Cycle Direct Adv 进行回连，回连超时为 3 次，每次 1.28s，预期只可回连关闭 Privacy 功能的对端设备，不可回连开启 Privacy 功能的对端设备；

```
• #define FEATURE_SUPPORT_PRIVACY      1  /* set 1 to enable privacy feature */
```

10.4 无操作超时断线功能

在 board.h 中设置宏 FEATURE_SUPPORT_NO_ACTION_DISCONN 进行无操作超时断线功能关闭或打开。默认关闭该功能。当宏打开后，RCU 如果处于配对状态，在超时时间 NO_ACTION_DISCON_TIMEOUT 之内都没有进行按键或 OTA 操作，RCU 会主动进行断线，以节省功耗。

在 swtimer.h 中，通过宏 NO_ACTION_DISCON_TIMEOUT 可以设置超时时间 NO_ACTION_DISCON_TIMEOUT，默认为 5 分钟。

```
1. #define FEATURE_SUPPORT_NO_ACTION_DISCONN    0  /* set 1 to enable NO_ACTION_
   DISCONN after timeout */
2. #define NO_ACTION_DISCON_TIMEOUT             300000  /* 300s */
```

10.5 配对前清除配对信息功能

在 board. h 中设置宏 FEATURE_SUPPORT_REMOVE_LINK_KEY_BEFORE_PAIRING 进行配对前清除配对信息功能的关闭或打开，默认打开该功能。当 FEATURE_SUPPORT_REMOVE_LINK_KEY_BEFORE_PAIRING 打开后，配对前会清除之前的配对信息，否则保留配对信息直到配对信息更新。

```
1. #define FEATURE_SUPPORT_REMOVE_LINK_KEY_BEFORE_PAIRING  1 /* set 1 to enabl
   e remove link key before paring */
```

10.6 备份和恢复配对信息功能

在 board.h 中设置宏 FEATURE_SUPPORT_RECOVER_PAIR_INFO 进行配对信息备份和恢复功能的关闭或打开，默认关闭该功能，且只有配置宏 FEATURE_SUPPORT_RECOVER_PAIR_INFO 为 1 时，该功能才生效。该功能旨在保证已配对的遥控器中时刻保存有一组配对信息。当 FEATURE_SUPPORT_RECOVER_PAIR_INFO 打开后，在清除配对信息前，先备份当前的配对信息，如果后续没有和其他设备成功配对并进入 Idle 状态时，会将备份的配对信息恢复，并清除备份信息。

```
1. #define FEATURE_SUPPORT_REMOVE_LINK_KEY_BEFORE_PAIRING  1 /* set 1 to enabl
   e remove link key before paring */
```

10.7 一键配对功能

在 board.h 中设置宏 FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV 进行一键配对功能关闭或打开，默认关闭该功能。当 FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV 打开后，在 RCU 没有配对信息时，可以通过按任意键触发配对广播。

```
1. #define FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV      0  /* set 1 to enable an
   y key pressed event to trigger paring adv */
```

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 Voice Service	Realtek Vendor 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。

12.1.1 Keyscan 工作方式

Keyscan 模块 Manual mode 提供两种触发方式： 按键触发方式和寄存器触发方式。

1. 当 Keyscan 配置为按键触发时，Keyscan 模块检测到有按键被按下后，会自动触发 Scan 操作；
2. 当 Keyscan 配置为寄存器触发时，通过软件设置触发 Scan 操作，按键不会自动触发。

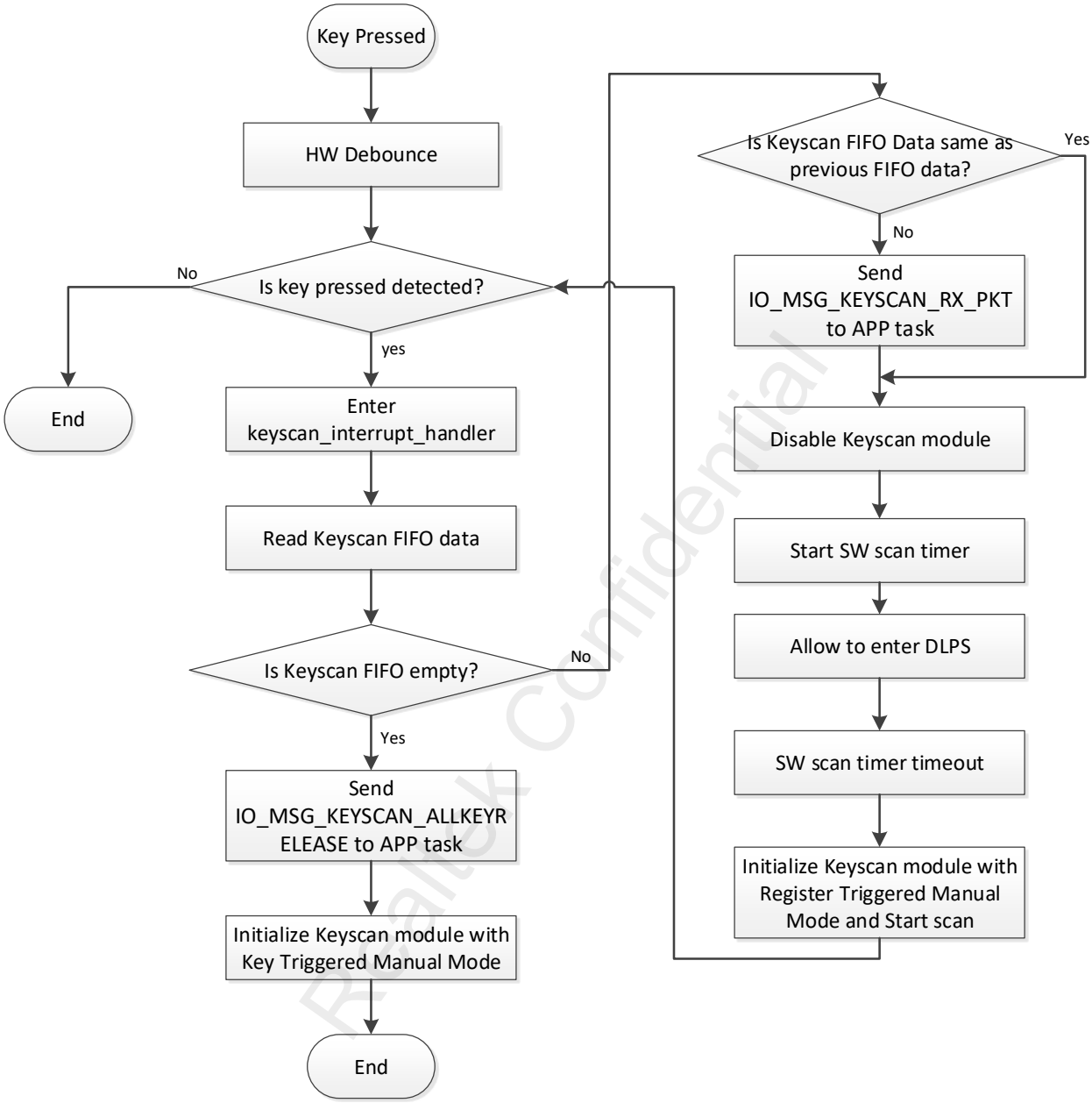
12.1.2 HW Debounce 机制

针对系统处于 Active Mode 还是 Sleep Mode，Keyscan 模块采用两种 Debounce 机制：Keyscan HW Debounce 和 PAD HW Debounce。

1. 当系统处于 Active Mode 时，Keyscan 处于工作模式，使用 Keyscan 模块的硬件 debounce 机制；
2. 当系统处于 Sleep Mode 时，Keyscan 模块不工作，使用 PAD 模块的 debounce 机制，在 debounce 阶段，系统还可以处于低功耗模式；

12.1.3 Keyscan 检测流程

Keyscan 检测流程如下图所示：



图表 19 Keyscan 流程

上电后，初始化 Keyscan 模块为 Manual Mode，选择按键触发方式，等待按键事件发生。

当有按键被按下后，在 HW debounce 之后会进行一次按键矩阵扫描。扫描结束后，会产生 scan interrupt，在中断处理函数中读取 Keyscan FIFO data。

如果 Keyscan FIFO 为空，表示按键已经被释放，则发送 IO_MSG_KEYSCAN_ALLKEYRELEASE 消息给 APP task 处理。同时，重新初始化 Keyscan 模块为 Manual Mode，选择按键触发方式，等待下一次按键事件。

如果 Keyscan FIFO 有数据，且和上次保存 Keyscan FIFO 数据不同，则发送 IO_MSG_KEYSCAN_RX_PKT 消息给 APP task 处理，若 FIFO 数据相同，则跳过消息发送操作。之后关闭 Keyscan 模块，开启软件定时器后，进入 DLPS。软件定时器超时唤醒系统，重新初始化 Keyscan 模块为 Manual Mode，选择寄存器触发方式，触发 Keyscan 进行一次 scan，继续检测按键的状态。

12.2 Keyscan 配置

遥控器应用程序中，行列的 PIN 脚在 board.h 文件中定义，可根据实际项目需求进行修改：

```
/* keypad row and column */
#if (RCU_HD_PLATFORM_SEL == R_DEMO_RCU)
#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

#elif (RCU_HD_PLATFORM_SEL == H_DEMO_RCU)
#define KEYPAD_ROW_SIZE      4
#define KEYPAD_COLUMN_SIZE   4

#define ROW0          P3_0
#define ROW1          P3_1
#define ROW2          P1_0
#define ROW3          P1_1

#define COLUMN0           P4_0
#define COLUMN1           P4_1
#define COLUMN2           P4_2
#define COLUMN3           P4_3

#elif (RCU_HD_PLATFORM_SEL == G_MIN_DEMO_RCU)
#define KEYPAD_ROW_SIZE      5
#define KEYPAD_COLUMN_SIZE   4
```

```
34.
35. #define ROW0            P4_3
36. #define ROW1            P4_2
37. #define ROW2            P4_1
38. #define ROW3            P4_0
39. #define ROW4            P0_6
40.
41. #define COLUMN0          P3_0
42. #define COLUMN1          P3_1
43. #define COLUMN2          P3_2
44. #define COLUMN3          P3_3
45.
46. #endif
```

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

```
1.  /* Key Mapping Table Definiton */
2.  #if (RCU_HD_PLATFORM_SEL == R_DEMO_RCU)
3.  static const T_KEY_INDEX_DEF KEY_MAPPING_TABLE[KEYPAD_ROW_SIZE][KEYPAD_COLUMN_SI
    ZE] =
4.  {
5.     {VK_TV_POWER,   VK_EXIT,          VK_ENTER,     VK_MOUSE_EN},
6.     {VK_POWER,      VK_VOLUME_UP,     VK_DOWN,      VK_RIGHT},
7.     {VK_TV_SIGNAL,  VK_VOLUME_DOWN,   VK_VOICE,     VK_MENU},
8.     {VK_UP,         VK_PAGE_DOWN,     VK_HOME,      VK_PAGE_UP},
9.     {VK_LEFT,       VK_NONE,          VK_NONE,      VK_NONE},
10. };
11. #elif (RCU_HD_PLATFORM_SEL == H_DEMO_RCU)
12. static const T_KEY_INDEX_DEF KEY_MAPPING_TABLE[KEYPAD_ROW_SIZE][KEYPAD_COLUMN_SI
    ZE] =
13. {
14.    {VK_POWER,            VK_VOICE,             VK_HOME,       VK_MENU},
15.    {VK_VOLUME_UP,        VK_VOLUME_DOWN,       VK_ENTER,      VK_EXIT},
16.    {VK_LEFT,             VK_RIGHT,             VK_UP,         VK_DOWN},
17.    {MM_VolumeIncrement,  MM_VolumeDecrement,   VK_TV_POWER,   VK_TV_SIGNAL},
18. };
19. #elif (RCU_HD_PLATFORM_SEL == G_MIN_DEMO_RCU)
20. static const T_KEY_INDEX_DEF KEY_MAPPING_TABLE[KEYPAD_ROW_SIZE][KEYPAD_COLUMN_SI
    ZE] =
21. {
22.     {VK_POWER,          MM_AC_Back,         MM_DPadCenter,  MM_Dashboard},
23.     {MM_Mute,           VK_VOLUME_UP,       MM_DPadDown,    MM_DPadRight},
24.     {MM_AC_Bookmarks,   VK_VOLUME_DOWN,     MM_AC_Search,   MM_Guide},
```

```
25.     {MM_DPadUp,          VK_YOUTUBE,          MM_AC_Home,      MM_Live},
26.     {MM_DPadLeft,        VK_NETFLIX,          VK_APP04,        VK_NONE},
27. };
28. #endif
```

12.3 按键类型

根据不同按键功能，BEE3 RCU SDK 中将按键分为 4 种不同的类型：

```
1.  /* define the key types */
2.  typedef enum
3.  {
4.     KEY_TYPE_NONE         = 0x00,  /* none key type */
5.     KEY_TYPE_BLE_ONLY     = 0x01,  /* only BLE key type */
6.     KEY_TYPE_IR_ONLY      = 0x02,  /* only IR key type */
7.     KEY_TYPE_BLE_OR_IR    = 0x03,  /* BLE or IR key type */
8.  } T_KEY_TYPE_DEF;
```

1. KEY_TYPE_NONE：按键按下不做处理，不发送蓝牙或红外键值；
2. KEY_TYPE_BLE_ONLY：按键只发送蓝牙键值；
3. KEY_TYPE_IR_ONLY：按键只发送红外键值；
4. KEY_TYPE_BLE_OR_IR：根据蓝牙状态，连接状态发送蓝牙键值，非连接状态发送红外键值；

12.4 按键键值定义

在 Bee3 RCU SDK 中，通过 KEY_CODE_TABLE 定义各个按键的类型，红外键值，蓝牙键值等参数，实际可根据具体项目需求进行相应调整。

```
1.  /* BLE HID code table definition */
2.  const T_KEY_CODE_DEF KEY_CODE_TABLE[KEY_CODE_TABLE_SIZE] =
3.  {
4.     /* key_type,          ir_key_code,    hid_usage_page,        hid_usage_id */
5.     {KEY_TYPE_NONE,         0x00,   HID_UNDEFINED_PAGE, 0x00},     /* VK_NONE */
6.     {KEY_TYPE_BLE_OR_IR,    0x18,   HID_KEYBOARD_PAGE,  0x66},     /* VK_POWER */
7.     {KEY_TYPE_BLE_OR_IR,    0x56,   HID_KEYBOARD_PAGE,  0x76},     /* VK_MENU */
8.     {KEY_TYPE_BLE_OR_IR,    0x14,   HID_KEYBOARD_PAGE,  0x4A},     /* VK_HOME */
9.     {KEY_TYPE_BLE_ONLY,     0x3E,   HID_KEYBOARD_PAGE,  0x3E},     /* VK_VOICE */
10.    ......
11.    {KEY_TYPE_IR_ONLY,      0x01,   HID_KEYBOARD_PAGE,  0x00},   /* VK_TV_POWER */
12.    {KEY_TYPE_IR_ONLY,      0x5A,   HID_KEYBOARD_PAGE,  0x00},  /* VK_TV_SIGNAL */
13. #if FEATURE_SUPPORT_MULTIMEDIA_KEYBOARD
```

```
14.    {KEY_TYPE_BLE_OR_IR,  0x00,  HID_CONSUMER_PAGE,  0xb5},  /* MM_ScanNext */
15.    {KEY_TYPE_BLE_OR_IR,  0x00,  HID_CONSUMER_PAGE,  0x0152}, /* MM_BassIncrement */
16.    {KEY_TYPE_BLE_OR_IR,  0x00,  HID_CONSUMER_PAGE,  0x0153}, /* MM_BassDecrement */
17.    {KEY_TYPE_BLE_OR_IR,  0x00,  HID_CONSUMER_PAGE,  0x0154}, /* MM_TrebleIncrement */
18.    {KEY_TYPE_BLE_OR_IR,  0x00,  HID_CONSUMER_PAGE,  0x0155}, /* MM_TrebleDecrement */
19.    {KEY_TYPE_BLE_OR_IR,  0x00,  HID_CONSUMER_PAGE,  0x0221},   /* MM_AC_Search */
20.    ......
21. #endif
22. };
```

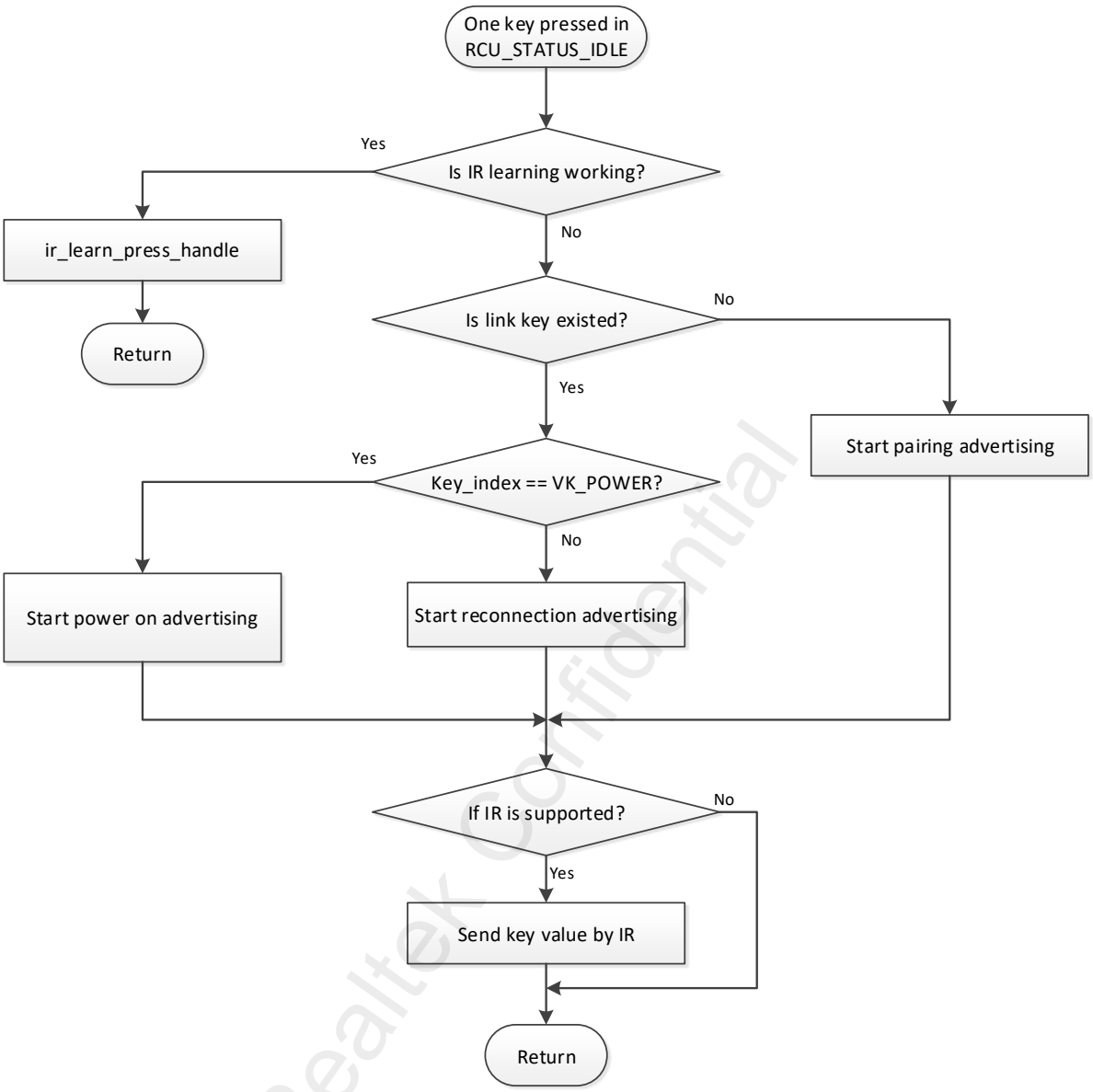
12.5 按键行为规范

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。

12.5.1 单个按键行为

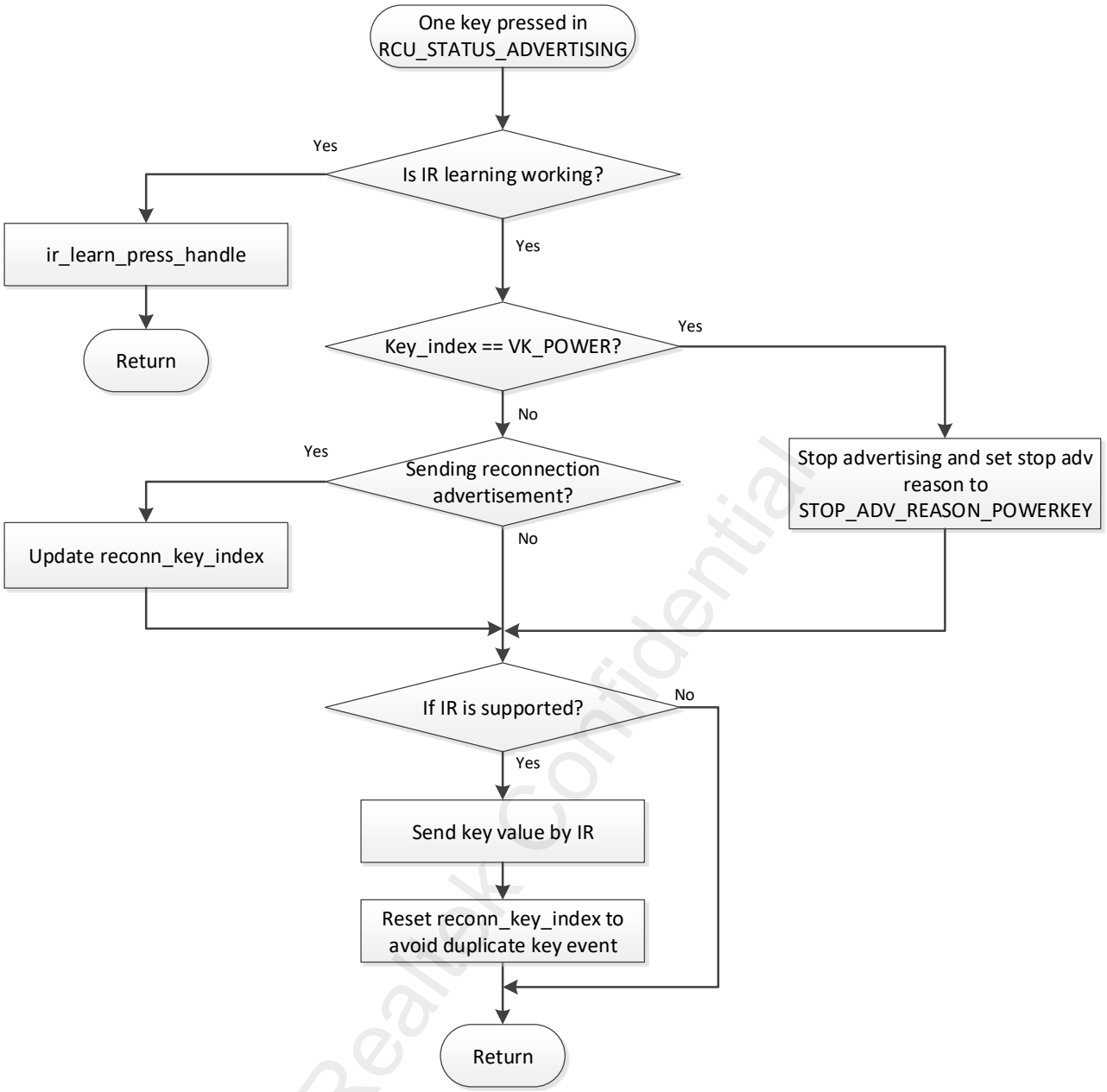
单个按键被按下之后，遥控器应用程序收到 Keyscan 模块发送的 IO_MSG_KEYSCAN_RX_PKT，根据当前 APP 的状态，分别进行如下处理：

- RCU_STATUS_IDLE 时，



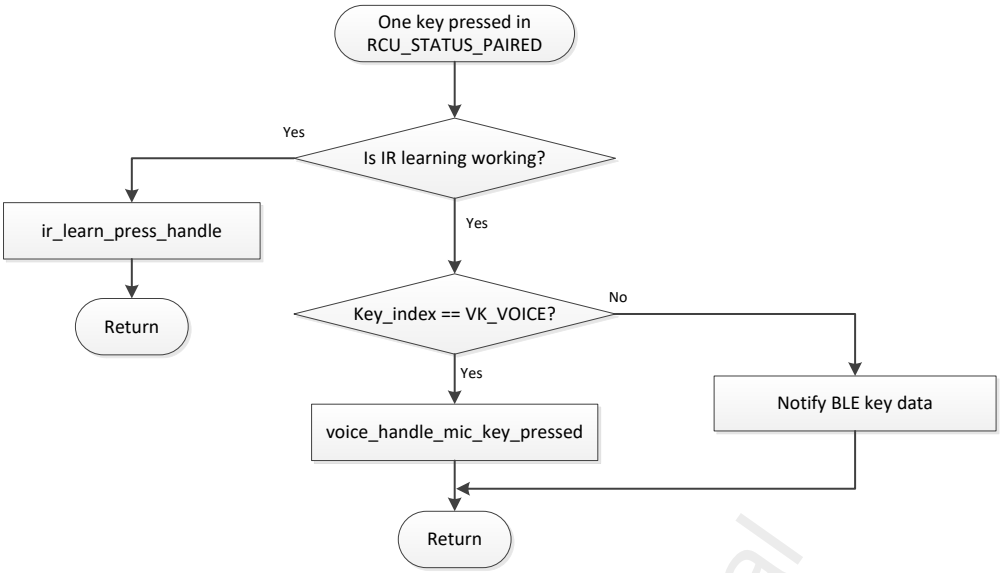
图表 20 RCU_STATUS_IDLE 单个按键处理流程

- RCU_STATUS_ADVERTISING 时，



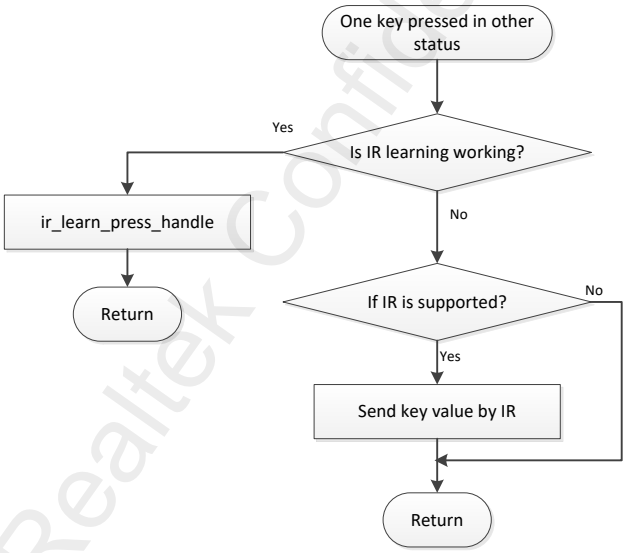
图表 21 RCU_STATUS_ADVERTISING 单个按键处理流程

- 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

12.5.2 多个按键行为

当有多个按键同时被按下时，Bee3 RCU 根据 board.h 中宏 FEATURE_SUPPORT_REPORT_MULTIPLE_HID_KEY_CODES 的设置，进行不同逻辑处理：

1. 如果 FEATURE_SUPPORT_REPORT_MULTIPLE_HID_KEY_CODES 设置为关闭，则多按键被按下后，停止之前的按键操作，同按键抬起处理一样；
2. 如果 FEATURE_SUPPORT_REPORT_MULTIPLE_HID_KEY_CODES 设置为打开，且 RCU 处于 RCU_STATUS_PAIRED 状态，发送包含多按键键值的 Notification；

12.6 蓝牙回连补发键值功能

RCU APP 针对 RCU_STATUS_IDLE 和 RCU_STATUS_ADVERTISING 状态，有 KEY_TYPE_BLE_ONLY 或 KEY_TYPE_BLE_OR_IR 类型按键按下，且没有通过 IR 发送红外键值的情况，在回连之后需要补发蓝牙键值：

1. 如果在回连之前，按键短按（按下、抬起），需要补发按键按下和抬起的蓝牙键值；
2. 如果在回连时，按键仍然被按下（长按键），只补发按键按下键值，待后续抬起后，再发送按键抬起键值；

12.7 特殊组合按键行为

当按键被按下时，RCU APP 会检测是否为定义的特殊组合按键，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）
11	0x8000	VK_LEFT + VK_HOME + VK_MENU	恢复出厂设置

图表 24 特殊组合键定义

12.8 长按键保护

在 board.h 中设置宏 FEATURE_SUPPORT_KEY_LONG_PRESS_PROTECT 进行长按键保护功能的关闭或打开，默认关闭该功能。当打开该功能时，会检测按键按下时长，当一个或多个按键被按下超过一定时间未释放时，遥控器会模拟执行按键释放的操作，同时所有按键均无效。只有当所有按键被释放后，遥控器按键才能继续正常使用。

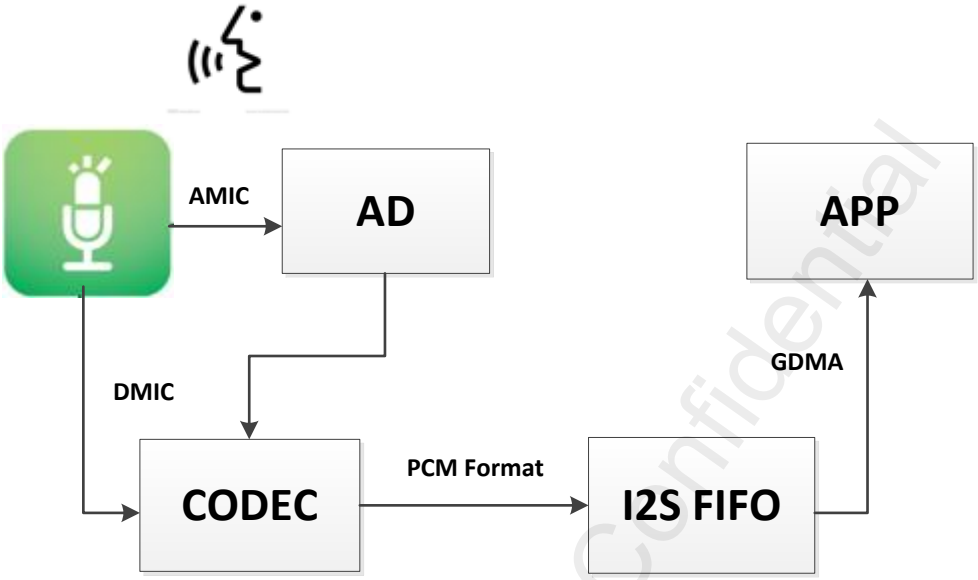
长按键保护的检测时间通过宏 LONG_PRESS_KEY_DETECT_TIMEOUT 来进行修改。

```
1. #define FEATURE_SUPPORT_KEY_LONG_PRESS_PROTECT        0 /*set 1 to stop scan
    when press one key too long*/
2. #define LONG_PRESS_KEY_DETECT_TIMEOUT     30000 /* 30 sec */
```

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

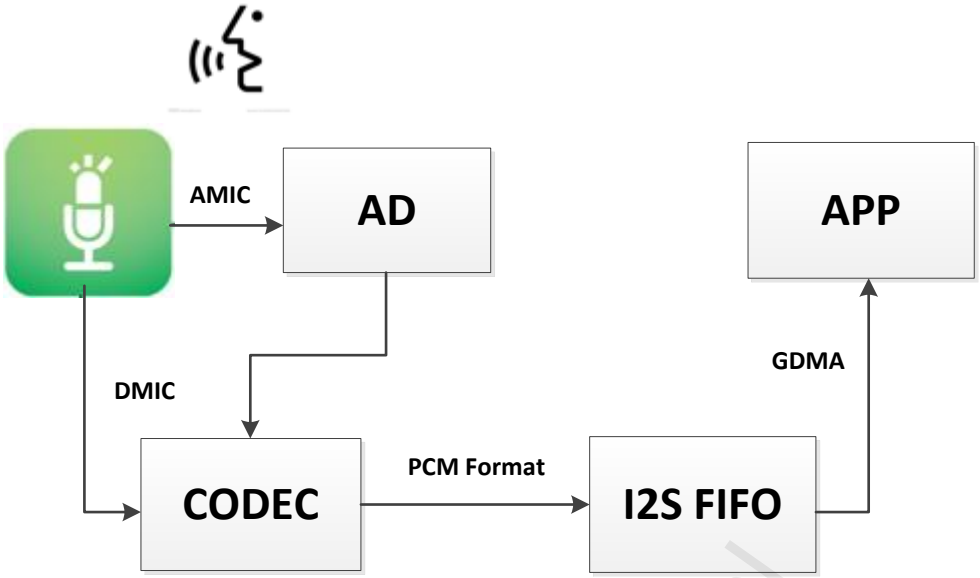


图表 25 语音功能整体框架图。

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，然后再把 PCM 格式的 data 放到 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 FIFO 中，之后 GDMA 再把这些 PCM 格式的 data 通过 I2S 接口搬移到 RAM 中提供给 App 处理。

Bee3 Codec 支持的采样频率为 8 KHz 和 16KHz，在 SDK board.h 中设置宏 CODEC_SAMPLE_RATE_SEL 进行选择。

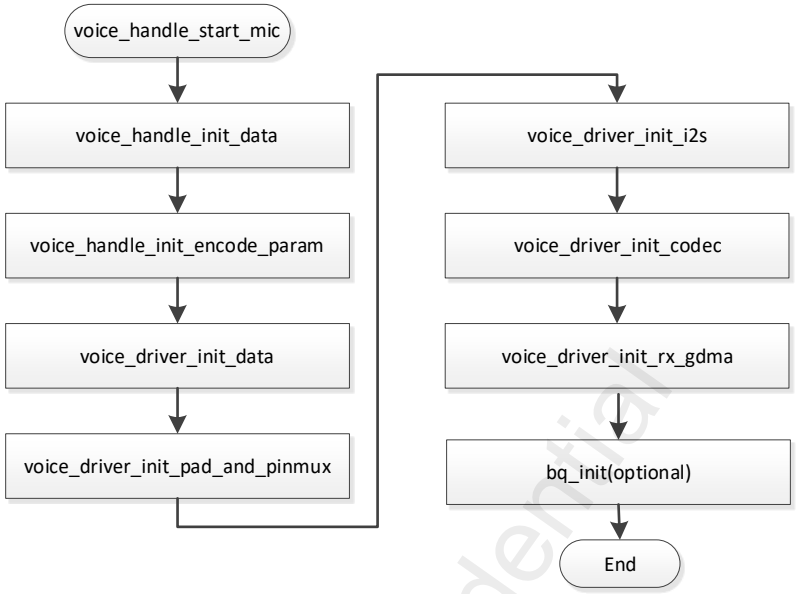
Bee3 支持 5-band 软件 EQ 对录取的语音数据做一些特殊处理。



图表 25 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，必须先初始化硬件模块，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 26 语音模块初始化流程

13.2.1 语音 PAD 初始化

Bee3 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 P2_6、P2_7，可以使用 PINMUX 来切换其他 GPIO。具体初始化代码如下：

```
1.  void voice_driver_init_pad_and_pinmux(void)
2.  {
3.  #if SUPPORT_DUAL_MIC_FEATURE
4.      /* dual MIC configuration */
5.  #if ((VOICE_MIC0_TYPE == AMIC_TYPE) || (VOICE_MIC1_TYPE == AMIC_TYPE))
6.      Pad_Config(AMIC_MIC_N_PIN, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT
    _DISABLE,
7.                 PAD_OUT_HIGH);
8.      Pad_Config(AMIC_MIC_P_PIN, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_
    DISABLE, PAD_OUT_LOW);
9.  #endif
10. #if ((VOICE_MIC0_TYPE == DMIC_TYPE) || (VOICE_MIC1_TYPE == DMIC_TYPE))
11.     Pad_Config(DMIC_CLK_PIN, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT
    _DISABLE,
12.                PAD_OUT_LOW);
```

```
    Pad_Config(DMIC_DATA_PIN, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_DISABLE,
            PAD_OUT_LOW);
    Pinmux_Deinit(DMIC_CLK_PIN);
    Pinmux_Deinit(DMIC_DATA_PIN);
    Pinmux_Config(DMIC_CLK_PIN, DMIC1_CLK);
    Pinmux_Config(DMIC_DATA_PIN, DMIC1_DAT);
#endif
#if SUPPORT_MIC_BIAS_OUTPUT
    Pad_Config(H_0, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE, PAD_OUT_HIGH);
#endif

#else
    /* single MIC configuration */
#if (VOICE_MIC0_TYPE == AMIC_TYPE)
    Pad_Config(AMIC_MIC_N_PIN, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_DISABLE,
            PAD_OUT_HIGH);
    Pad_Config(AMIC_MIC_P_PIN, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_DISABLE, PAD_OUT_LOW);
#elif (VOICE_MIC0_TYPE == DMIC_TYPE)
    Pad_Config(DMIC_CLK_PIN, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_DISABLE,
            PAD_OUT_LOW);
    Pad_Config(DMIC_DATA_PIN, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_DISABLE,
            PAD_OUT_LOW);
    Pinmux_Deinit(DMIC_CLK_PIN);
    Pinmux_Deinit(DMIC_DATA_PIN);
    Pinmux_Config(DMIC_CLK_PIN, DMIC1_CLK);
    Pinmux_Config(DMIC_DATA_PIN, DMIC1_DAT);
#endif
#if SUPPORT_MIC_BIAS_OUTPUT
    Pad_Config(H_0, PAD_SW_MODE, PAD_NOT_PWRON, PAD_PULL_NONE, PAD_OUT_DISABLE, PAD_OUT_HIGH);
#endif

#endif
}
```

13.2.2 语音 I2S 初始化

Bee3 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.  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.      I2S_InitStruct.I2S_DataFormat       = I2S_Mode;
9.      I2S_InitStruct.I2S_DeviceMode       = I2S_DeviceMode_Master;
10.     I2S_InitStruct.I2S_RxWaterlevel     = 0x4;
11.
12.     /* config I2S clock speed */
13.     /* BCLK = 40MHz*(I2S_BClockNi/I2S_BClockMi) = Sample Rate * 64BCLK/sample */
14.     if (voice_driver_codec_params.codec_sample_rate == SAMPLE_RATE_8KHz)
15.     {
16.         I2S_InitStruct.I2S_BClockMi       = 0x186A;
17.         I2S_InitStruct.I2S_BClockNi       = 0x50;
18.     }
19.     else if (voice_driver_codec_params.codec_sample_rate == SAMPLE_RATE_16KHz)
20.     {
21.         I2S_InitStruct.I2S_BClockMi       = 0x186A;
22.         I2S_InitStruct.I2S_BClockNi       = 0xA0;
23.     }
24.     else
25.     {
26.         APP_PRINT_WARN1("[voice_driver_init_i2s] invalid codec sample rate: %d",
27.                         voice_driver_codec_params.codec_sample_rate);
28.         I2S_InitStruct.I2S_BClockMi       = 0x186A;
29.         I2S_InitStruct.I2S_BClockNi       = 0xA0;
30.     }
31.
32.     /* config I2S channel type */
33.     I2S_InitStruct.I2S_ChannelType      = I2S_Channel_Mono;
34.
35.     I2S_Init(I2S0, &I2S_InitStruct);
36.     I2S_Cmd(I2S0, I2S_MODE_RX, ENABLE);
37. }
```

13.2.3 CODEC 初始化

Bee3 RCU Codec 配置可分为如下几部分：

1. Codec 基本参数配置

   包括 CODEC_SampleRate，CODEC_DmicClock，CODEC_I2SFormat，CODEC_I2SDataWidth，CODEC_I2SChSequence，CODEC_MicBIAS，CODEC_MicBstGain，CODEC_MicBstMode：

- CODEC_SampleRate：设置采样率，支持 8 KHz /16KHz；
- CODEC_DmicClock：DMIC 时钟，支持 312500Hz/ 625KHz/ 1250KHz/ 2500KHz/ 5MHz；
- CODEC_I2SFormat: I2S Format，支持 I2S/ LeftJustified/ PCM_A/ PCM_B；
- CODEC_I2SDataWidth：I2S 数据宽度，支持 8Bits/ 16Bits/24Bits；
- CODEC_I2SChSequence：I2S Channel 顺序，支持 CODEC_I2S_CH_L_R/ CODEC_I2S_CH_R_L/ CODEC_I2S_CH_L_L/ CODEC_I2S_CH_R_R；
- CODEC_MicBIAS：MIC BIAS 输出电压，支持 1.507/1.62/1.705/1.8/1.906/2.025/2.16/2.314V；
- CODEC_MicBstGain：AMIC BST Gain 值，支持 0/20/30/40dB；
- CODEC_MicBstMode：AMIC BST Mode，支持 MICBST_Mode_Single/ MICBST_Mode_Differential;

2. MIC 通道 0 参数配置

   包括 CODEC_Ch0Mute，CODEC_Ch0MicType，CODEC_Ch0DmicDataLatch，CODEC_Ch0AdGain，CODEC_Ch0BoostGain，CODEC_Ch0ZeroDetTimeout：

   - CODEC_Ch0Mute：Channel 0 数据是否 Mute，支持 MUTE 和 UNMUTE;
   - CODEC_Ch0MicType：MIC 类型选择，支持 AMIC 和 DMIC；
   - CODEC_Ch0DmicDataLatch：DMIC Data Latch 类型，支持 Rising_Latch 和 Falling_Latch；
   - CODEC_Ch0AdGain：Channel 0 digit volume 增益，支持设置范围-17.625dB 到 30dB，精度 0.375dB；
   - CODEC_Ch0BoostGain：Channel 0 digit Boost 增益，支持 0/12/24/36dB；

初始化代码如下：

```
1.  void voice_driver_init_codec(void)
2.  {
3.      /* switch power mode */
4.      SystemCall(3, 1);
5.
6.      CODEC_AnalogCircuitInit();
7.
8.      RCC_PeriphClockCmd(APBPeriph_CODEC, APBPeriph_CODEC_CLOCK, ENABLE);
9.
10.     CODEC_InitTypeDef CODEC_InitStruct;
11.     CODEC_StructInit(&CODEC_InitStruct);
12.
13.     /* Basic parameters section */
14.     CODEC_InitStruct.CODEC_SampleRate = voice_driver_codec_params.codec_sample_rate;
15.     CODEC_InitStruct.CODEC_DmicClock = voice_driver_codec_params.dmic_clock;
16.     CODEC_InitStruct.CODEC_I2SFormat = voice_driver_codec_params.codec_i2s_format;
```

```
    CODEC_InitStruct.CODEC_I2SDataWidth = voice_driver_codec_params.codec_i2s_data_width;
    CODEC_InitStruct.CODEC_I2SChSequence = voice_driver_codec_params.codec_i2s_ch_sequence;
    CODEC_InitStruct.CODEC_MicBIAS = voice_driver_codec_params.mic_bias_voltage;
    CODEC_InitStruct.CODEC_MicBstGain = voice_driver_codec_params.amic_bst_gain;
    CODEC_InitStruct.CODEC_MicBstMode = voice_driver_codec_params.amic_bst_mode;

    /* MIC channel 0 initialization parameters section */
    CODEC_InitStruct.CODEC_Ch0Mute = voice_driver_codec_params.codec_ch0_mute;
    CODEC_InitStruct.CODEC_Ch0MicType = voice_driver_codec_params.codec_ch0_mic_type;
    CODEC_InitStruct.CODEC_Ch0DmicDataLatch = voice_driver_codec_params.codec_ch0_dmic_data_latch;
    CODEC_InitStruct.CODEC_Ch0AdGain = voice_driver_codec_params.codec_ch0_ad_gain;
    CODEC_InitStruct.CODEC_Ch0BoostGain = voice_driver_codec_params.codec_ch0_boost_gain;
    CODEC_InitStruct.CODEC_Ch0ZeroDetTimeout = voice_driver_codec_params.codec_ch0_zero_det_timeout;

    CODEC_Init(CODEC, &CODEC_InitStruct);
}
```

取消初始化代码如下：

```
void voice_driver_deinit_codec(void)
{
    CODEC_DeInit(CODEC);

    /* restore power mode */
    SystemCall(3, 0);
}
```

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。Bee3 RCU 通过 GDMA Multi-Block 功能实现 Ping-Pong Buffer。

具体初始化代码如下:

```
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          = VOICE_GDMA_Channel_NUM;
    GDMA_InitStruct.GDMA_DIR                 = GDMA_DIR_PeripheralToMemory;
    GDMA_InitStruct.GDMA_BufferSize          = VOICE_GDMA_FRAME_SIZE / 4;
    GDMA_InitStruct.GDMA_DestinationInc      = DMA_DestinationInc_Inc;
    GDMA_InitStruct.GDMA_SourceInc           = DMA_SourceInc_Fix;
    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_SourceHandshake     = GDMA_Handshake_I2S0_RX;
    GDMA_InitStruct.GDMA_DestinationAddr     = (uint32_t)voice_driver_global_data.gdma_buffer.b
uf0;
    GDMA_InitStruct.GDMA_DestHandshake       = GDMA_Handshake_SPIC0_RX;
    GDMA_InitStruct.GDMA_Multi_Block_En      = 1;
    GDMA_InitStruct.GDMA_Multi_Block_Struct  = (uint32_t)voice_driver_gdma_link_list;
    GDMA_InitStruct.GDMA_Multi_Block_Mode    = LLI_TRANSFER;

    /* Initialize GDMA Link List Struct */
    for (uint8_t i = 0; i < 2; i++)
    {
        if (i == 0)
        {
            voice_driver_gdma_link_list[i].DAR = (uint32_t)voice_driver_global_data.gdma_buffer.buf
0;
            voice_driver_gdma_link_list[i].LLP = (uint32_t)&voice_driver_gdma_link_list[i +
                                                                1];  /* link to buffer 1 */
        }
        else
        {
            voice_driver_gdma_link_list[i].DAR = (uint32_t)voice_driver_global_data.gdma_buffer.buf
1;
            voice_driver_gdma_link_list[i].LLP = (uint32_t)&voice_driver_gdma_link_list[i -
                                                                1];  /* link back to buffer 0 */
        }
        voice_driver_gdma_link_list[i].SAR = GDMA_InitStruct.GDMA_SourceAddr;

        /* Configure low 32 bit of CTL register */
        voice_driver_gdma_link_list[i].CTL_LOW = BIT(0)
                                    | (GDMA_InitStruct.GDMA_DestinationDataSize << 1)
                                    | (GDMA_InitStruct.GDMA_SourceDataSize << 4)
                                    | (GDMA_InitStruct.GDMA_DestinationInc << 7)
                                    | (GDMA_InitStruct.GDMA_SourceInc << 9)
```

```
                                    | (GDMA_InitStruct.GDMA_DestinationMsize << 11)
                                    | (GDMA_InitStruct.GDMA_SourceMsize << 14)
                                    | (GDMA_InitStruct.GDMA_DIR << 20)
                                    | (GDMA_InitStruct.GDMA_Multi_Block_Mode & LLP_SELECTED_BIT);
        /* Configure high 32 bit of CTL register */
        voice_driver_gdma_link_list[i].CTL_HIGH = GDMA_InitStruct.GDMA_BufferSize;
    }

    GDMA_Init(VOICE_GDMA_Channel, &GDMA_InitStruct);

    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = VOICE_GDMA_Channel_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPriority  = 3;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    GDMA_INTConfig(VOICE_GDMA_Channel_NUM, GDMA_INT_Block, ENABLE);
    GDMA_Cmd(VOICE_GDMA_Channel_NUM, ENABLE);
}
```

13.2.5 SW EQ 初始化

在 Bee3 SDK board.h 中设置宏 SUPPORT_SW_EQ 进行软件 EQ 功能关闭或打开。

支持 5-band EQ 对录取的语音数据做一些特殊处理，相关参数可以配合 RTL8762E EQ Tool 进行调整和设置，可将 EQ Tool 生成参数填入 SW_EQ_PARAMS_TABLE[SW_EQ_MAX_STAGE][6]作为默认参数。

具体初始化代码如下：

```
void bq_init(void)
{
    uint8_t i;
    for (i = 0; i < SW_EQ_MAX_STAGE && SW_EQ_PARAMS_TABLE[i][0] == EQ_CH_Cmd_ENABLE; i++)
    {
        sw_eq_stage = i + 1;
    }
    APP_PRINT_INFO1("[sw equalizer parameters init]sw_eq_stage=%d", sw_eq_stage);
    for (i = 0; i < sw_eq_stage; i++)
    {
        bq[i].b0 = SW_EQ_PARAMS_TABLE[i][1] ;
        bq[i].b1 = (SW_EQ_PARAMS_TABLE[i][2] * SW_EQ_PARAMS_TABLE[i][1] + EQ_ROUND)
     >> EQ_FACTOR;
```

```
13.         bq[i].b2 = (SW_EQ_PARAMS_TABLE[i][3] * SW_EQ_PARAMS_TABLE[i][1] + EQ_ROUND)
    >> EQ_FACTOR;
14.         bq[i].a1 = SW_EQ_PARAMS_TABLE[i][4] ;
15.         bq[i].a2 = SW_EQ_PARAMS_TABLE[i][5] ;
16.         bq[i].prev_input_1 = 0;
17.         bq[i].prev_input_2 = 0;
18.         bq[i].prev_output_1 = 0;
19.         bq[i].prev_output_2 = 0;
20.     }
21. }
```

13.3 语音 Buffer 介绍

Bee3 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。

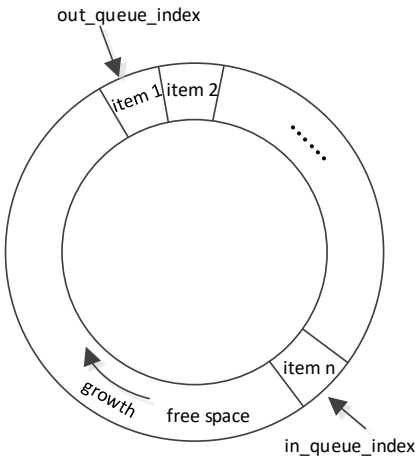
13.3.1 GDMA Ping-Pong Buffer

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 block 传输完成之后，进行交替使用。

13.3.2 Voice Data Loop Buffer

Bee3 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 6000Bytes 的内存块，可以通过宏 VOICE_QUEUE_MAX_BUFFER_SIZE 进行调整。入队列指针即图表 27 语音 Loop Buffer 示意图中的 in_queue_index，出队列指针为图表 27 语音 Loop Buffer 示意图中的 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 27 语音 Loop Buffer 示意图

13.4 语音编码算法

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。

Bee3 RCU 支持多种软件压缩编码算法，包括 SW mSCB encoder, SW SBC encoder 和 SW IMA ADPCM encoder。Bee3 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。

```
/* 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 */
```

13.5 语音交互流程

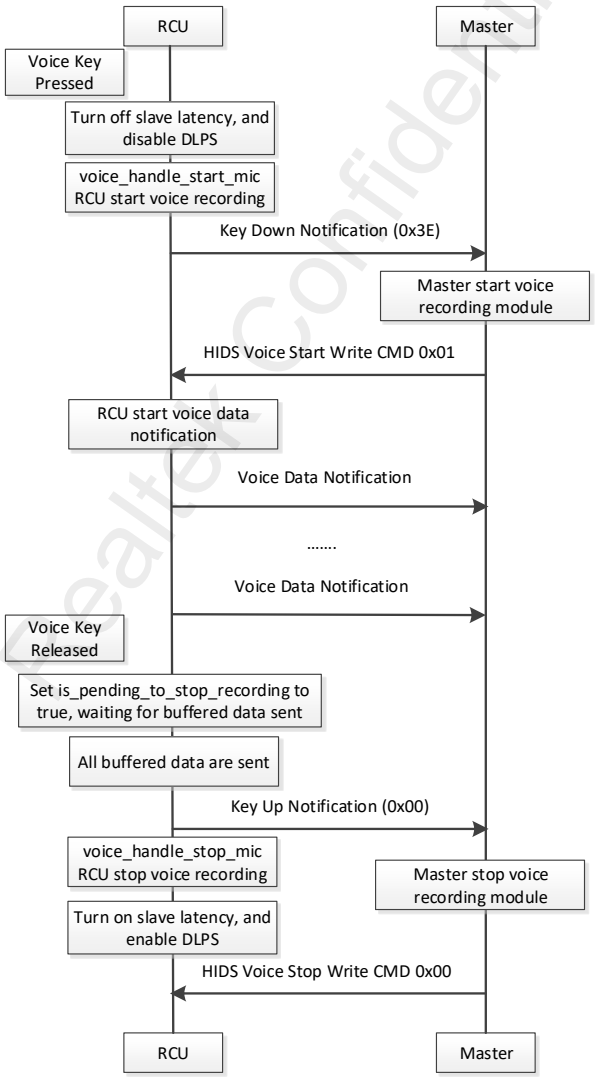
Bee3 RCU 支持如下几种语音交互流程：

- IFLYTEK_VOICE_FLOW
- HIDS_GOOGLE_VOICE_FLOW
- ATV_GOOGLE_VOICE_FLOW
- RTK_GATT_VOICE_FLOW

Bee3 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：



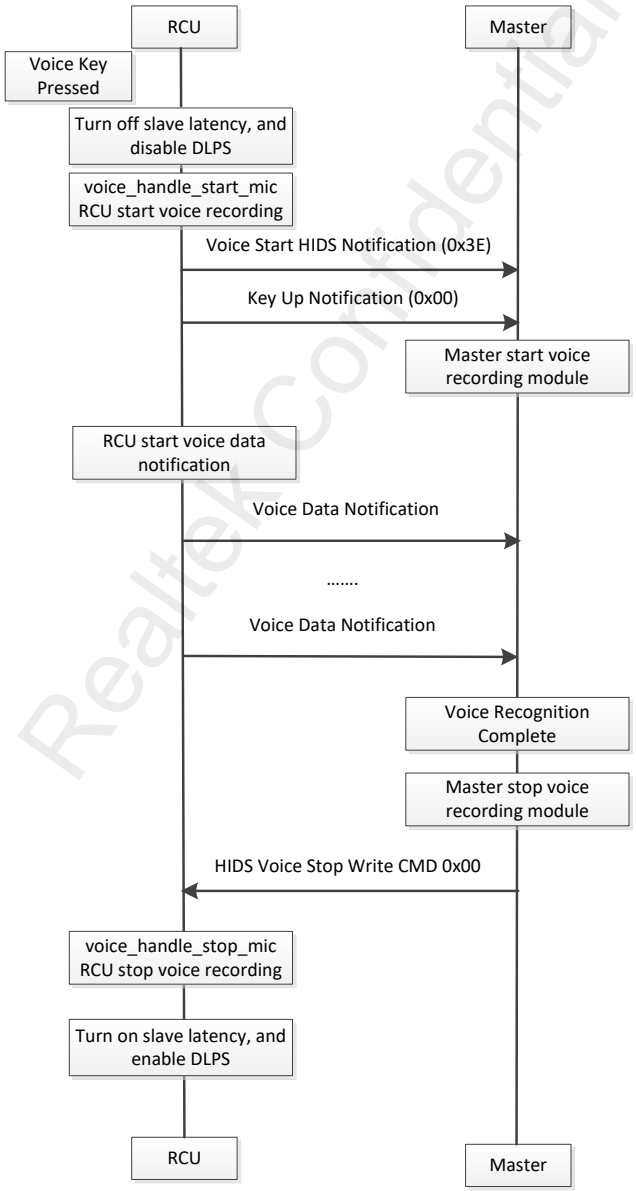
图表 28 IFLYTEK VOICE FLOW

13.5.2 HIDS Google VOICE FLOW

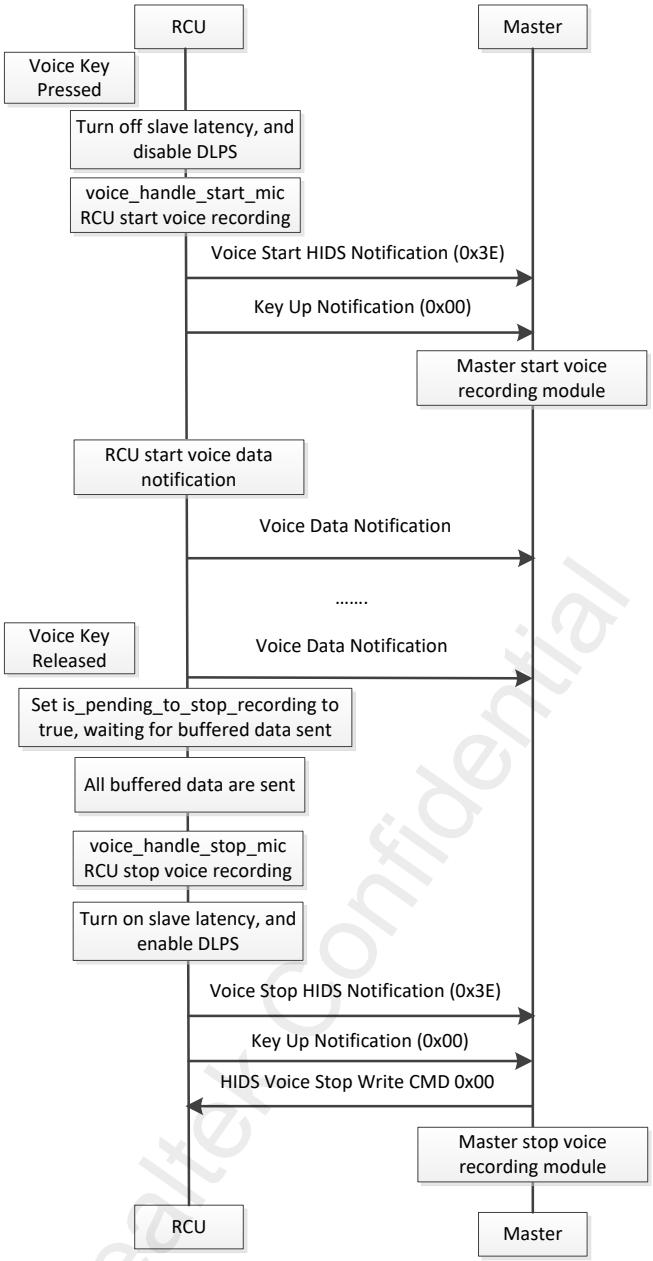
HIDS Google Voice 交互流程和 IFLYTEK 语音交互流程类似，都是基于 HID Service，区别在于：

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 29 HIDS Google VOICE FLOW – 1



图表 30 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。

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 v0.4 和 Google Voice over BLE spec 1.0 文档。Bee3 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。当使用 ATV v0.4 时，一次 Voice Data 的 Notification size 是 134bytes；当使用 ATV v1.0 时，一次 Voice Data 的 Notification size 是 128bytes。

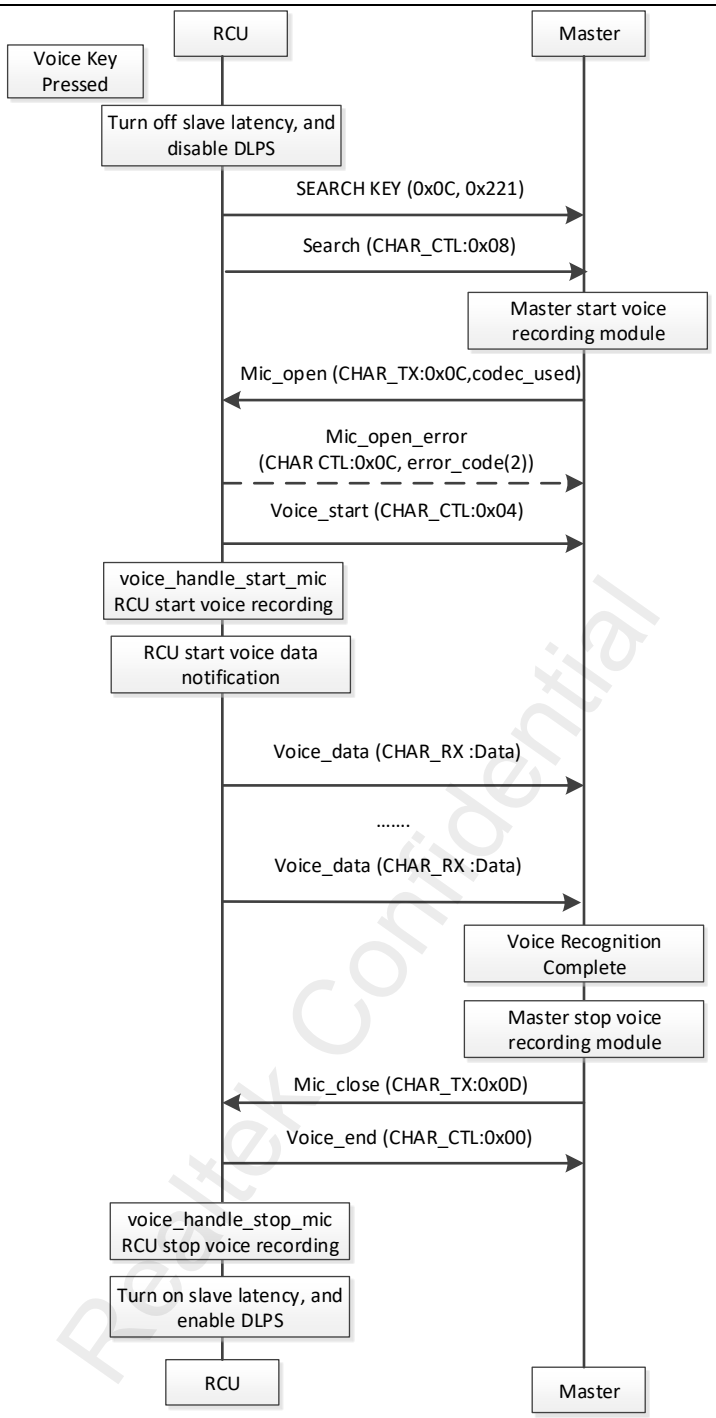
Bee3 RCU 可以通过 board.h 中的 ATV_VOICE_VERSION 宏定义来选择使用的 ATV 语音版本。默认使用的是 ATV v1.0 版本。

```
/* ATV Google voice version type */
#define ATV_VERSION_0_4            1  /* v0.4 */
#define ATV_VERSION_1_0            2  /* v1.0 */
#define ATV_VOICE_VERSION          ATV_VERSION_1_0  /* default version is v1.0 */
```

ATV v1.0 中包含三种 assistant interaction type，可以通过 board.h 中的 ATV_VOICE_INTERACTION_MODEL 宏定义来选择使用的 assistant interaction type。ATV v0.4 中仅支持 on request type。具体 assistant interaction type 内容可以参见 Google Voice over BLE spec 1.0 文档。

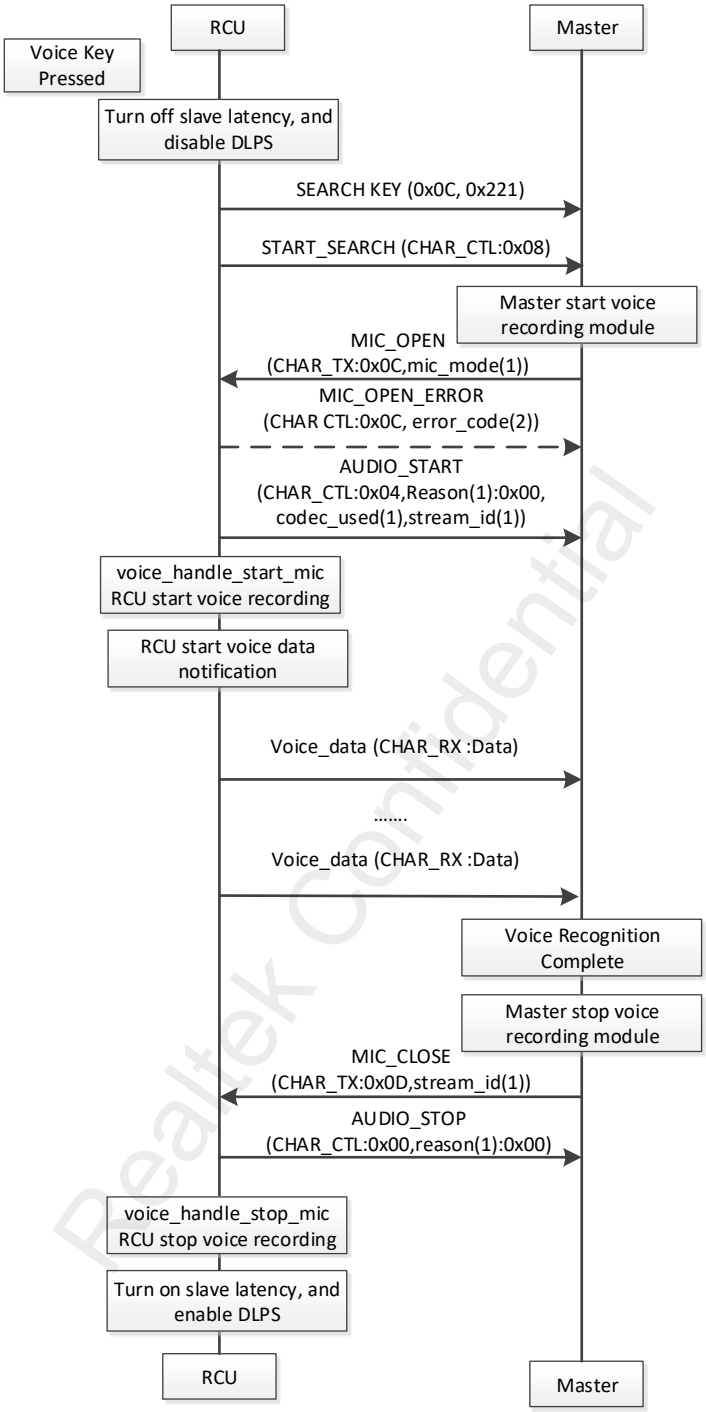
```
/* ATV Google voice assistant interaction type */
#define ATV_INTERACTION_MODEL_ON_REQUEST          1  /* on request */
#define ATV_INTERACTION_MODEL_PRESS_TO_TALK       2  /* PTT */
#define ATV_INTERACTION_MODEL_HOLD_TO_TALK        3  /* HTT */

#define ATV_VOICE_INTERACTION_MODEL ATV_INTERACTION_MODEL_ON_REQUEST
/* default assistant interaction is on request */
```

当使用 ATV v0.4 时，遥控器和主机端之间的语音交互流程如下图 31 所示：

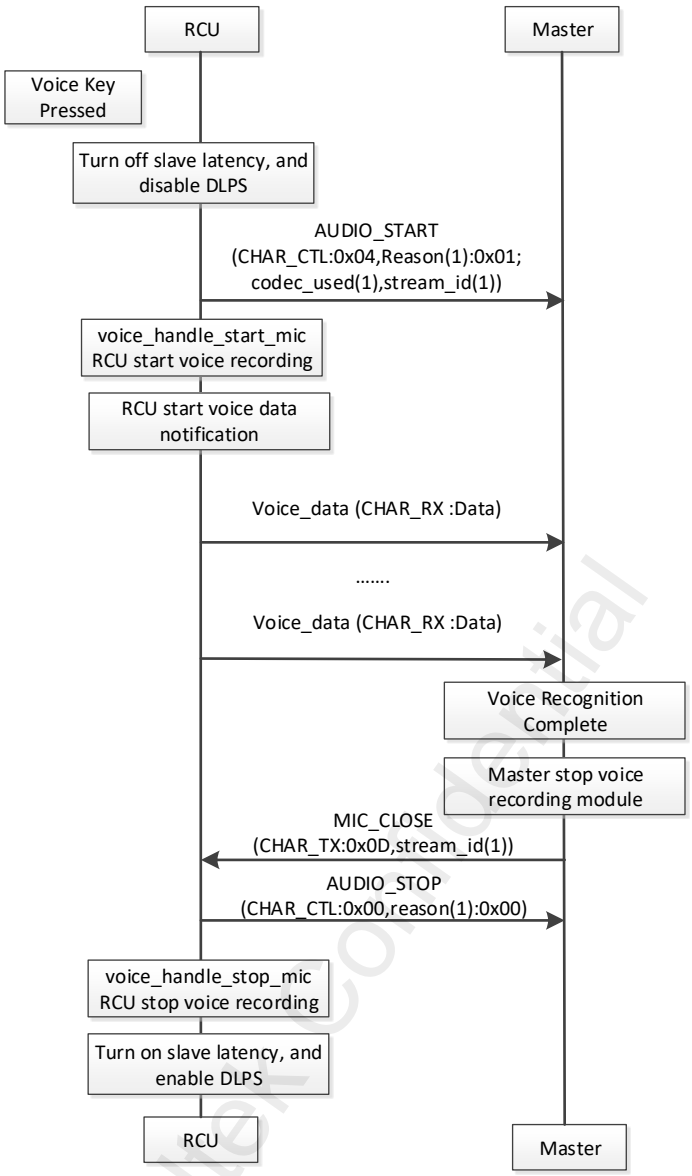


图表 31 ATV v0.4 Google VOICE FLOW

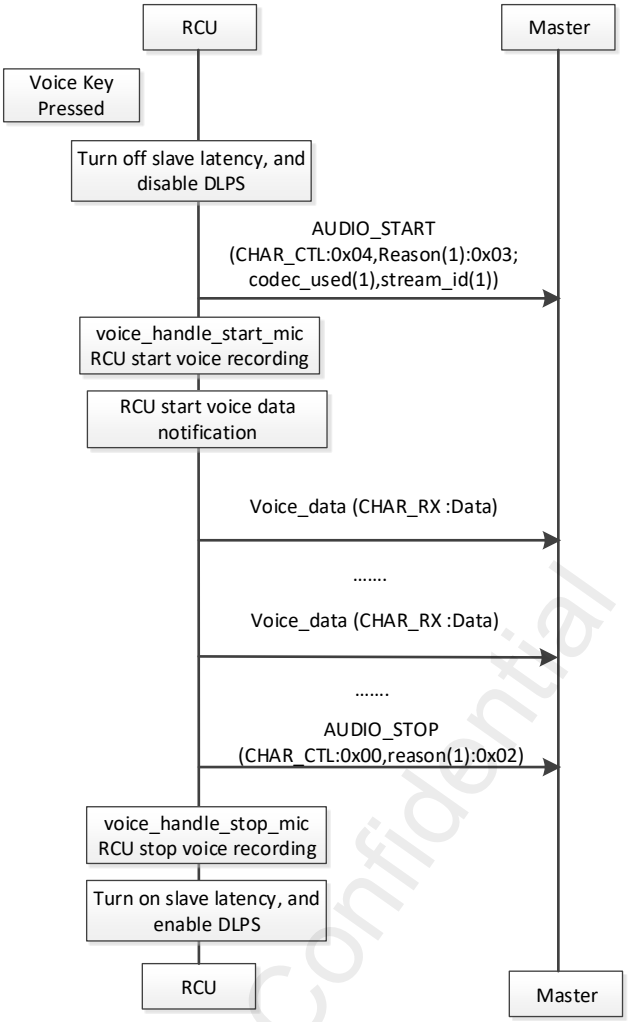
ATV v1.0 中包含三种 assistant interaction type，不同方式遥控器和主机端之间的语音交互流程如下所示：



图表 32 ATV v1.0 Google VOICE FLOW-ON_REQUEST_TYPE



图表 33 ATV v1.0 Google VOICE FLOW-PTT_TYPE



图表 34 ATV v1.0 Google VOICE FLOW-HTT_TYPE

13.5.4 RTK GATT VOICE FLOW

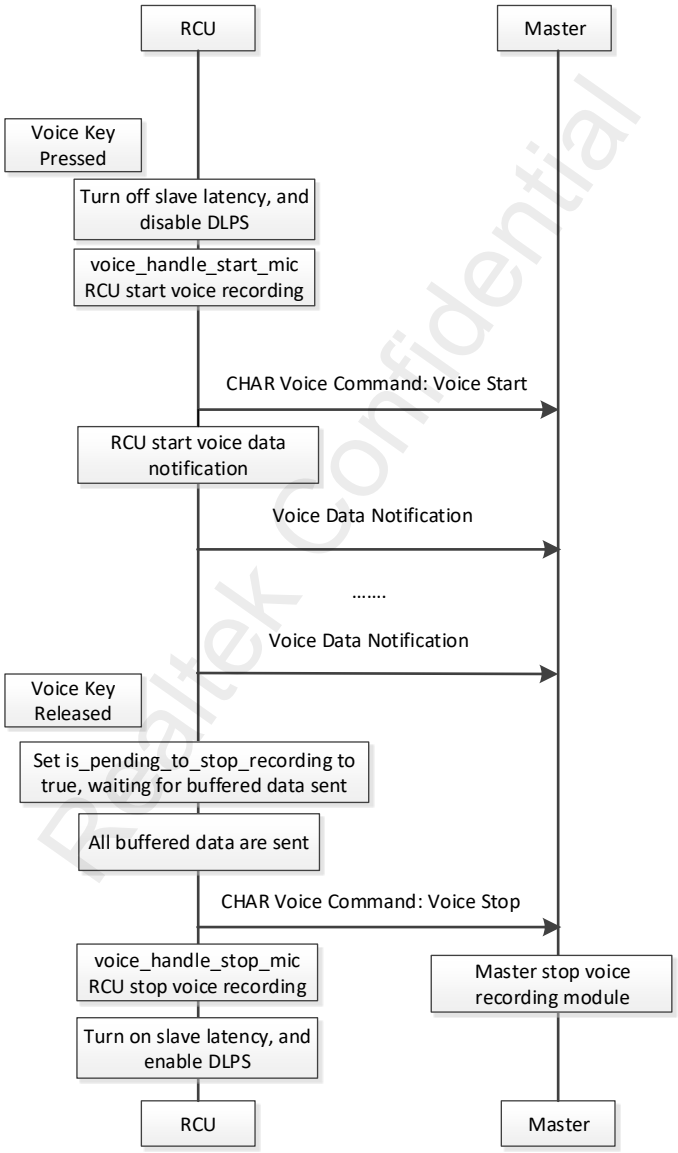
RTK GATT 语音交互流程是基于 Realtek GATT Vendor Service，可以和安装 Realtek Voice APK 的 Android 设备进行语音传输。

Realtek GATT Vendor Service UUID 定义：{0x12, 0xA2, 0x4D, 0x2E, 0xFE, 0x14, 0x48, 0x8e, 0x93, 0xD2, 0x17, 0x3C, ***0xFD, 0x03***, 0x00, 0x00}；

通过 Voice Command Characteristic 发送语音控制命令 Notification，Characteristic UUID 为 0x3A40;

通过 Voice Data Characteristic 发送语音数据 Notification，Characteristic UUID 为 0x3A41；

遥控器和主机端之间的语音交互流程如下：



图表 35 RTK GATT VOICE FLOW

14 IR 红外

遥控器支持红外功能，包括红外发送和红外学习。Bee3 IR 载波频率支持 10KHz – 2MHz,可以支持几乎所有的红外协议。

14.1 红外发送

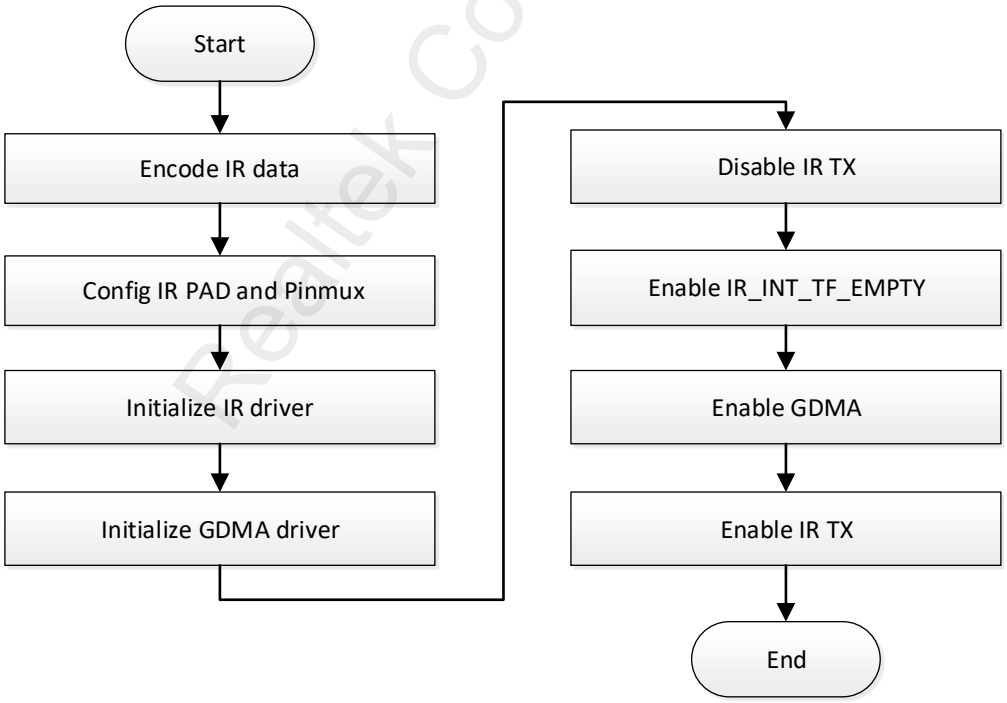
在 Bee3 SDK board.h 中设置宏 SUPPORT_IR_TX_FEATURE 进行红外发送功能关闭或打开。

在默认状态，红外按键发送 KEY_CODE_TABLE 中定义的红外键值。对于红外学习成功之后的红外按键，按键发送学习到的红外波形。红外学习成功后，通过回复出厂设置组合键后，红外按键恢复发送默认的红外键值。

Bee3 RCU SDK 中，默认支持 NEC 红外协议，程序设计上也留出相关接口，方便支持其他红外协议的扩展。

14.1.1 红外发送初始化流程

Bee3 RCU 红外发送初始化流程如下图所示：



图表 36 红外发送初始化流程

14.1.2 红外发送 GDMA 功能

Bee3 RCU 支持 IR 发送 +GDMA 功能，可以直接通过 GDMA 将红外波形数据从内存搬运到红外发送模块，方便软件逻辑的处理。

具体初始化代码如下:

```
void ir_driver_tx_gdma_init(uint32_t source_addr, uint32_t buffer_size)
{
    RCC_PeriphClockCmd(APBPeriph_GDMA, APBPeriph_GDMA_CLOCK, ENABLE);
    GDMA_InitTypeDef GDMA_InitStruct;

    /*--------------GDMA init-----------------------------*/
    GDMA_StructInit(&GDMA_InitStruct);
    GDMA_InitStruct.GDMA_ChannelNum          = IR_TX_GDMA_Channel_num;
    GDMA_InitStruct.GDMA_DIR                 = GDMA_DIR_MemoryToPeripheral;
    GDMA_InitStruct.GDMA_SourceInc           = DMA_SourceInc_Inc;
    GDMA_InitStruct.GDMA_DestinationInc      = DMA_DestinationInc_Fix;
    GDMA_InitStruct.GDMA_SourceDataSize      = GDMA_DataSize_Word;
    GDMA_InitStruct.GDMA_DestinationDataSize = GDMA_DataSize_Word;
    GDMA_InitStruct.GDMA_SourceMsize         = GDMA_Msize_4;
    GDMA_InitStruct.GDMA_DestinationMsize    = GDMA_Msize_4;
    GDMA_InitStruct.GDMA_DestinationAddr     = (uint32_t)(&IR->TX_FIFO);
    GDMA_InitStruct.GDMA_DestHandshake       = GDMA_Handshake_IR_TX;
    GDMA_InitStruct.GDMA_BufferSize          = buffer_size;
    GDMA_InitStruct.GDMA_SourceAddr          = source_addr;
    GDMA_Init(IR_TX_GDMA_CHANNEL, &GDMA_InitStruct);
}
```

14.1.3 红外 Repeat Code

Bee3 RCU SDK 实现红外发送 Repeat Code 功能，Repeat Code 由 RTC 进行计时。RTC 模块的初始化代码如下：

```
void rtc_driver_init (void)
{
    APP_PRINT_INFO0("[driver_rtc_init] init RTC");
    rtc_ovf_cnt = 0;
    RTC_DeInit();
    RTC_SetPrescaler(RTC_PRESCALER_VALUE);
```

```
7.
8.      /* Config RTC interrupt */
9.      NVIC_InitTypeDef NVIC_InitStruct;
10.     NVIC_InitStruct.NVIC_IRQChannel = RTC_IRQn;
11.     NVIC_InitStruct.NVIC_IRQChannelPriority = 3;
12.     NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
13.     NVIC_Init(&NVIC_InitStruct);
14.
15.     RTC_SystemWakeupConfig(ENABLE);
16.      RTC_NvCmd(ENABLE);
17.     /* Start RTC */
18.     RTC_ResetCounter();
19.     RTC_Cmd(ENABLE);
20. }
```

14.2 红外学习

在 Bee3 RCU SDK board.h 中设置宏 SUPPORT_IR_LEARN_FEATURE 进行红外学习功能关闭或打开。

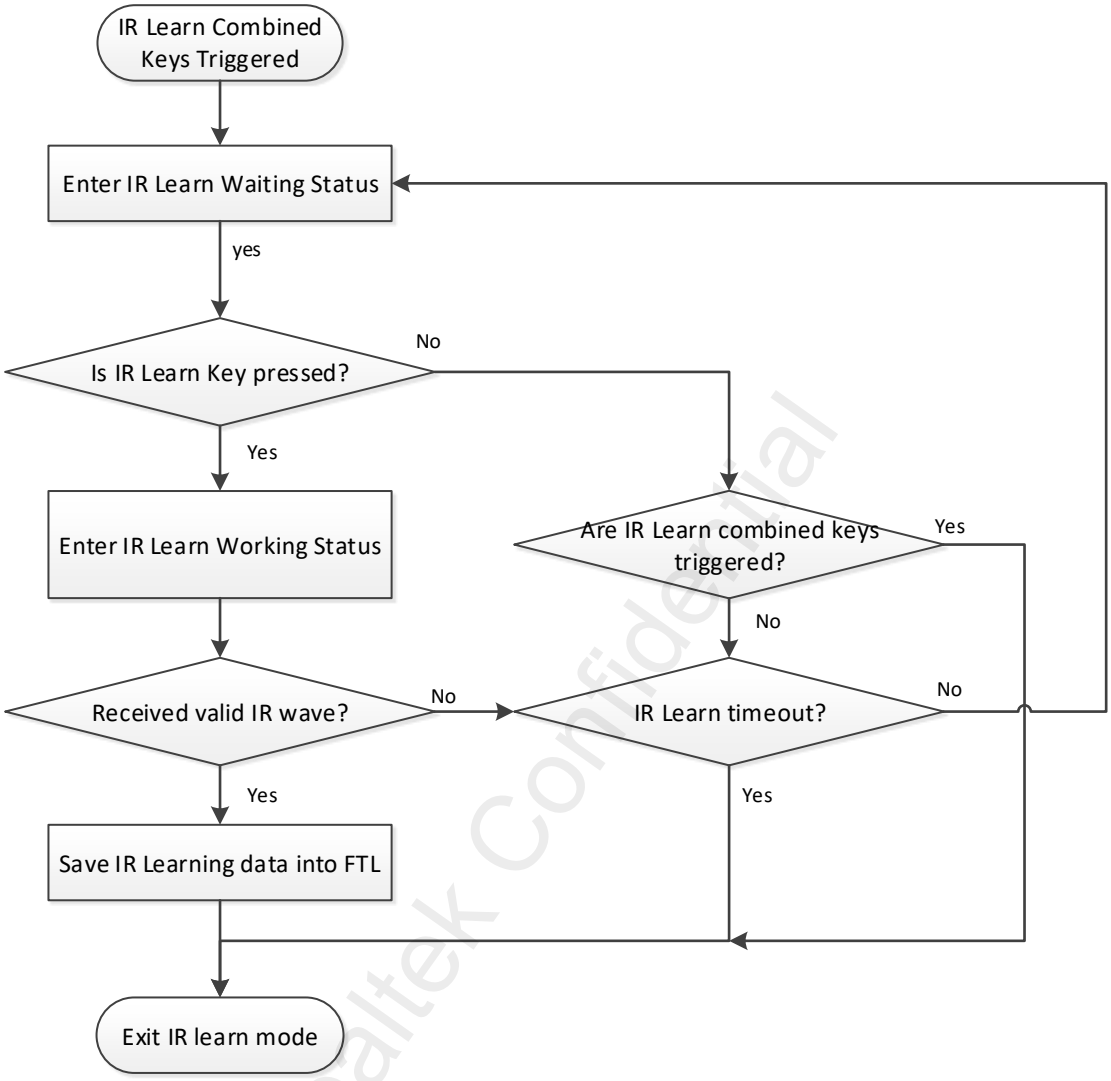
Bee3 RCU SDK 红外学习实例是针对波长数据在 70 个数据以下的红外协议，可以根据宏 IR_LEARN_WAVE_MAX_SIZE 来进行调节。红外数据存储在 FTL 中，红外波形的数据的存储基地址可以使用宏 IR_WAVE_DATA_BASE_ADDR 来进行调节，但是注意不要超出 FTL 的内存大小。红外发送的时候会从存储的地址读取指定的红外数据，直接进行发送。

Bee3 RCU SDK 过设置 ir_learn_table 来限制可以红外学习的按键，只有在定义中的按键可以学习波形，其他按键均不允许。默认支持三个红外学习按键：VK_TV_POWER，VK_TV_SIGNAL，VK_POWER。可根据实际项目，进行修改。

```
1.  #define  IR_LEARN_MAX_KEY_NUM      0x03  /* max key number can be storaged */
2.
3.  static T_IR_LEARN_KEY_INFO ir_learn_table[IR_LEARN_MAX_KEY_NUM] =
4.  {
5.     {0, VK_TV_POWER},
6.     {1, VK_TV_SIGNAL},
7.     {2, VK_POWER},
8.  };
```

14.3 红外学习操作流程

Bee3 RCU SDK 提供一种红外学习操作流程，如下图所示：



图表 37 红外学习流程图

1. 通过红外学习组合键 VK_ENTER + VK_RIGHT （2 秒），进入红外学习等待模式；
2. 在红外学习等待模式，按下红外学习按键并保持，进入红外学习工作模式；
3. 在红外工作模式下，如果 RCU 收到有效红外信号，则红外学习成功，存储红外波形数据到 FTL；
4. 红外学习成功之后，返回红外学习等待模式；
5. 如果在红外学习工作模式下，松开红外学习按键，则本次红外学习按键失败，返回红外学习等待模式；
6. 在红外学习模式下，通过红外学习组合键 VK_ENTER + VK_RIGHT （2 秒）或长时间无按键操作超时（IR_LEARN_TIMEOUT 20 秒），可以退出红外学习模式；

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《RTL8762E OTA User Manual》。

15.1 Normal OTA 方式

在 RCU SDK board.h 中，通过设置宏 SUPPORT_NORMAL_OTA 进行 Normal OTA 功能的关闭或打开。

Normal OTA 方式是通过 OTA Service 来实现的。RCU APP 需要重启进入 OTA 模式，OTA 模式下不会运行 APP task。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

在 RCU SDK board.h 中，通过设置宏 SUPPORT_SILENT_OTA 进行 Silent OTA 功能的关闭或打开。

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：

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

16 电量检测及低电量保护

RCU SDK 支持 ADC 电量检测和低电量保护功能。在 RCU SDK board.h 中设置宏 SUPPORT_BAT_DETECT_FEATURE 进行电量检测功能的关闭或打开。

16.1 电量检测方案

在 RCU SDK 中，有如下几种情况会触发电量检测操作：

16.1.1 开机电量检测

RCU SDK 会在开机完成之后调用 API bat_update_battery_info 进行电量信息更新。

16.1.2 蓝牙 Battery Service 读取电量

在蓝牙连接之后，对端设备通过电池服务读取的电池信息，会调用 API bat_update_battery_info 进行电量信息更新，并上报电量百分比。

16.1.3 按键触发电量检测

在 RCU SDK board.h 中设置宏 SUPPORT_BAT_KEY_PRESS_DETECT_FEATURE 进行按键触发功能的关闭或打开，默认打开。

当 RCU 的按键次数到达 BAT_DETECT_TRIGGER_CNT 时，主动触发电量检测。BAT_DETECT_TRIGGER_CNT 可进行修改，默认为 10，即每 10 次按键触发一次电量检测。

16.1.4 定时触发电量检测

在 RCU SDK board.h 中设置宏 SUPPORT_BAT_PERIODIC_DETECT_FEATURE 进行定时检测电量功能的关闭或打开，默认关闭。

当 SUPPORT_BAT_PERIODIC_DETECT_FEATURE 功能打开之后，RCU 每隔 BAT_PERIODIC_DETECT_TIMEOUT 时间，进行一次电池电量检查。BAT_PERIODIC_DETECT_TIMEOUT 可进行修改，默认为 10 分钟

16.1.5 LPC 触发电量检测

在 RCU SDK board.h 中设置宏 SUPPORT_BAT_LPC_FEATURE 进行硬件 LPC 功能的关闭或打开。

上电时如果 RCU 处于低电量模式则不进行 LPC 初始化，如果 RCU 处于正常模式则进行 LPC 初始化。当 RCU 电量低于设置的阈值 BAT_LPC_COMP_VALUE 时，会立即触发 LPC 中断执行 RCU 重启操作。

```
void bat_nvic_config(void)
{
#if SUPPORT_BAT_LPC_FEATURE
    if (BAT_MODE_NORMAL == bat_get_current_mode())
    {
        APP_PRINT_INFO0("[bat_nvic_config] Init LPC");
        bat_driver_lpc_init();
    }
    else
    {
        APP_PRINT_INFO0("[bat_nvic_config] In low power mode. Don't init LPC");
    }
#endif
}
```

LPC 初始化流程如下：

```
void bat_driver_lpc_init(void)
{
    LPC_InitTypeDef LPC_InitStruct;
    LPC_StructInit(&LPC_InitStruct);
    LPC_InitStruct.LPC_Channel   = LPC_CHANNEL_VBAT ;
    LPC_InitStruct.LPC_Edge      = LPC_Vin_Below_Vth ;
    LPC_InitStruct.LPC_Threshold = BAT_LPC_COMP_VALUE;
    LPC_Init(&LPC_InitStruct);
    LPC_Cmd(ENABLE);
    RamVectorTableUpdate(LPCOMP_VECTORn, bat_lpc_handler);

    LPC_ResetCounter();
    LPC_SetCompValue(1);
    LPC_CounterCmd(ENABLE);
    LPC_ClearINTPendingBit(LPC_INT_LPCOMP_CNT);
    LPC_INTConfig(LPC_INT_LPCOMP_CNT, ENABLE);

    LPC_INTCmd(ENABLE);

```

```
20.     /* Config LPC interrupt */
21.     NVIC_InitTypeDef NVIC_InitStruct;
22.     NVIC_InitStruct.NVIC_IRQChannel = LPCOMP_IRQn;
23.     NVIC_InitStruct.NVIC_IRQChannelPriority = 3;
24.     NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
25.     NVIC_Init(&NVIC_InitStruct);
26. }
```

16.2 低电量模式

当电量检测小于阈值 BAT_ENTER_LOW_POWER_THRESHOLD 后，RCU 进入低功耗模式。

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。

处于低电量模式下，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，不响应按键操作；

若此时系统电池的电压高于指定的电池电压检测阈值 BAT_ENTER_NORMAL_MODE_THRESHOLD，就会退出低电量模式，并执行相应的按键操作。

16.3 电量检测阈值

Bee3 RCU SDK 电量检测部分有定义如下几个阈值：

BAT_ENTER_LOW_POWER_THRESHOLD：从正常工作模式进入低电量模式的阈值，默认为 2.0V；

BAT_ENTER_NORMAL_MODE_THRESHOLD：从低电量模式恢复正常工作模式的阈值，默认为 2.2V；

BAT_ENTER_OTA_MODE_THRESHOLD：允许进入 OTA 模式的最低阈值，默认为 2.5V；

BAT_LPC_COMP_VALUE：LPC 触发阈值，默认为 1.84V；

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，并且支持按优先级多事件闪烁。

17.1 LED 的配置

整个 LED 模块的控制通过宏 SUPPORT_LED_INDICATION_FEATURE 来进行控制。LED 的数量通过 LED_NUM_MAX 来进行控制，目前默认使用 1 个 LED，可以进行修改扩展。

```
/*************************************************************
*           LED Module Config
*************************************************************/
#define SUPPORT_LED_INDICATION_FEATURE       1

#if SUPPORT_LED_INDICATION_FEATURE

#if (RCU_HD_PLATFORM_SEL == R_DEMO_RCU)
#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)

#elif (RCU_HD_PLATFORM_SEL == H_DEMO_RCU)
#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) | P0_2)

#elif (RCU_HD_PLATFORM_SEL == G_MIN_DEMO_RCU)
#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)

#endif

/* 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
```

```
33.
34. #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。

```
1.  #define LED_ON(index)                    led_blink_start(index, LED_TYPE_ON, 0)
2.  #define LED_OFF(index)                   led_blink_exit(index, LED_TYPE_ON)
3.  #define LED_BLINK(index, type, n)        led_blink_start(index, type, n)
4.  #define LED_BLINK_EXIT(index, type)      led_blink_exit(index, type)
```

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。

17.3 LED 场景

17.3.1 LED 闪烁时间定义

目前 Bee3 SDK 中使用三种不同闪烁时间定义：SHORT_BLINK，NORMAL_BLINK，LONG_BLINK。

Parameter	On Duration	Off Duration (when blinking more than 1 time)	Units
SHORT_BLINK	100	100	milliseconds
LONG_BLINK	250	750	milliseconds
NORMAL_BLINK	250	250	milliseconds

图表 38 LED 闪烁时间定义

17.3.2 LED 指示场景

基于上述三种闪烁类型，RCU SDK 中目前已经添加可以使用的 LED 指示场景如下：

1. 组合键恢复出厂模式指示：NORMAL_BLINK 3 次；

2. 进入低电量模式指示：SHORT_BLINK 5 次；
3. 低电量模式按键指示：SHORT_BLINK 1 次；
4. 红外学习成功指示：SHORT_BLINK 5 次；
5. 红外学习等待模式指示：LONG_BLINK；
6. 红外学习工作模式指示：LED 常亮；
7. 配对广播模式指示：NORMAL_BLINK；
8. 配对成功模式指示：SHORT_BLINK 5 次；
9. 正常模式按键指示：按键按下 LED 常亮，按键释放 LED 熄灭；

若需要添加新的 LED 指示类型需要按如下方法进行：

```
/**
 *  @brief user guide for led driver
 *  The driver support multi prio, multi event, and multi led.
 *  If you want to control a led with a event as you designed, there is two way to
 *  achieve this, modify the exist event and redefine a new event.
 *
 *  The led driver is drived by software timer, the timer peroid is 50ms by default.
 *
 *  #define  LED_PERIOD   50 //you an change the value according your requirement
 *
 *  For the whole led driver system, there is 5 place you might modify.
 *
 *  1. Define led num and led pin, which is in file board.h;
 *     For example, define 2 leds as followed:
 *     #define  LED_NUM_MAX   0x02
 *     #define  LED_INDEX(n)   (n<<8)
 *
 *     //uint16_t, first byte led index, last byte led pin
 *     #define  LED_1         (LED_INDEX(0) | P2_2)
 *     #define  LED_2         (LED_INDEX(1) | P0_2)
 *
 *  2. Define led type index, which is in file led_driver.h, these types must be
 *     defined in sequence.
 *     For example,
 *     #define  LED_TYPE_BLINK_OTA_UPDATE      (8)
 *
 *  3. Define led loop bits num, in file led_driver.h
 *     //max value 32, min value 1, it indicate that there is how many bits to loop
 *     //from LSB
 *     #define LED_LOOP_BITS_OTA_UPDATE        (15)
 *
```

```
32. *  4. Define led event bit map, in file led_driver.h
33. *     // it must be cooperated with led loop bits num
34. *     #define  LED_BIT_MAP_OTA_UPDATE       (0x4210)//on 50ms, off 200ms, period 50ms
35. *
36. *  5. Update led event table, led_event_arr[LED_TYPE_MAX], in file led_driver.c
37. *     Before you use the event you define ,you need to add led type, led loop bit num,
38. *     and led event bit map into event table.
39. *     const T_LED_EVENT_STG led_event_arr[LED_TYPE_MAX] =
40. *     {
41. *             ... ...
42. *      {LED_TYPE_BLINK_OTA_UPDATE, LED_LOOP_BITS_OTA_UPDATE,  LED_BIT_MAP_OTA_UPD
    ATE},
43. *     };
44. *
45. *  There are three interfaces for Led driver, as follow.
46. *
47. *  void led_module_init(void); // called when system boot;
48. *  T_LED_RET_CAUSE led_blink_start(uint16_t index, LED_TYPE type, uint8_t cnt);
49. *  T_LED_RET_CAUSE led_blink_exit(uint16_t index, LED_TYPE type);
50. */
```

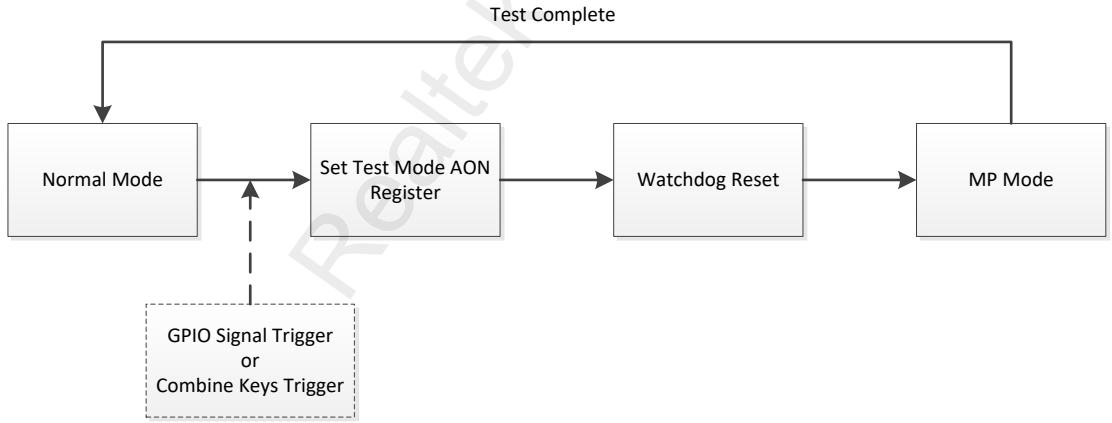
18 量产测试

Bee3 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《RTL8762E RCU MP Test Mode Design Spec》 。Bee3 RCU SDK 中，在 board.h 中有相关的宏定义来支持量产测试模式。

```
/*************************************************************
*                MP Test Config
*************************************************************/

#if FEATURE_SUPPORT_MP_TEST_MODE

#define MP_TEST_FTL_PARAMS_TEST_MODE_FLG_OFFSET (MP_TEST_FTL_PARAMS_BASE_ADDR)
#define MP_TEST_FTL_PARAMS_TEST_MODE_FLG_LEN    4

#define MP_TEST_FTL_PARAMS_LOCAL_BD_ADDR_OFFSET (MP_TEST_FTL_PARAMS_TEST_MODE_FLG_OFFSET + MP_TEST_FTL_PARAMS_TEST_MODE_FLG_LEN)
#define MP_TEST_FTL_PARAMS_LOCAL_BD_ADDR_LEN    8

#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)

#if (MP_TEST_MODE_TRIG_SEL & MP_TEST_MODE_TRIG_BY_GPIO)
```

```
26.
27. #if (RCU_HD_PLATFORM_SEL == R_DEMO_RCU)
28. #define MP_TEST_TRIG_PIN_1       P3_2
29. #define MP_TEST_TRIG_PIN_2       P3_3
30.
31. #elif (RCU_HD_PLATFORM_SEL == H_DEMO_RCU)
32. #define MP_TEST_TRIG_PIN_1       P5_1
33. #define MP_TEST_TRIG_PIN_2       P5_2
34.
35. #elif (RCU_HD_PLATFORM_SEL == G_MIN_DEMO_RCU)
36. #define MP_TEST_TRIG_PIN_1       P3_2
37. #define MP_TEST_TRIG_PIN_2       P3_3
38.
39. #endif
40. #endif
41.
42. #if MP_TEST_MODE_SUPPORT_DATA_UART_TEST
43. #define MP_TEST_UART_TX_PIN      P3_0
44. #define MP_TEST_UART_RX_PIN      P3_1
45. #endif
46.
47. #endif
```

Bee3 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 39 切换量产测试模式流程

Bee3 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《RTL8762E RCU MP Test Sample Flow》 。

19 参考文献

[1] RTL8762E OTA User Manual

[2] RTL8762E RCU MP Test Mode Design Spec

[3] RTL8762E RCU MP Test Sample Flow