RTL8762E RCU Design Spec

V1.1 by Realtek

2022/04/15

Revision History

Date	Version	Modification
2021/07/01	V1.0	First version of RTL8762E RCU design specification
2021/04/15	V1.1	Add software EQ

Contents

Figures

1 Overview

This document mainly introduces the software related technical parameters and behavior specifications of RTL8762E voice RCU, including advertising package definition, standby and wake-up mechanism, pairing process, voice functions, key operation and other behavior specifications to guide the development of RCU and trace the problems encountered in software testing. RTL8762E voice RCU is referred to as Bee3 RCU in the following.

2 Function & Feature

- *Support BLE 5.1*
- *Support Automatic Pairing*
- *Support Power ON/OFF via Bluetooth*
- *Hardware Keyscan, Support extension up to 240 keys (12x20)*
- *Support Voice Input and Transmission*
- *Support IR Transmission*
- *Support IR Learning*
- *Support LED Indication*
- *Support Battery Voltage Detection*
- *Support Low-power protection mode*
- *Support OTA*
- *Support MP Mode*
- *Power Supply: 1.8~3V*

3 GPIO configuration

Bee3 RCU SDK supports three kinds of RCU hardware configurations by default. It can be configured by macro RCU_ HD_ PLATFORM_ SEL in board.h.

```
#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: adapt RTL8762E EVB (Evaluation Board) hardware design

R_DEMO_RCU: adapt Realtek Bee3 RCU demo PCBA hardware design

G_MIN_DEMO_RCU: adapt Google RCU hardware design

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: Through the external mic board, it can support AMIC and 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: AMIC in PCBA;

```
#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: Through the external mic board, it can support AMIC and 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 System block diagram

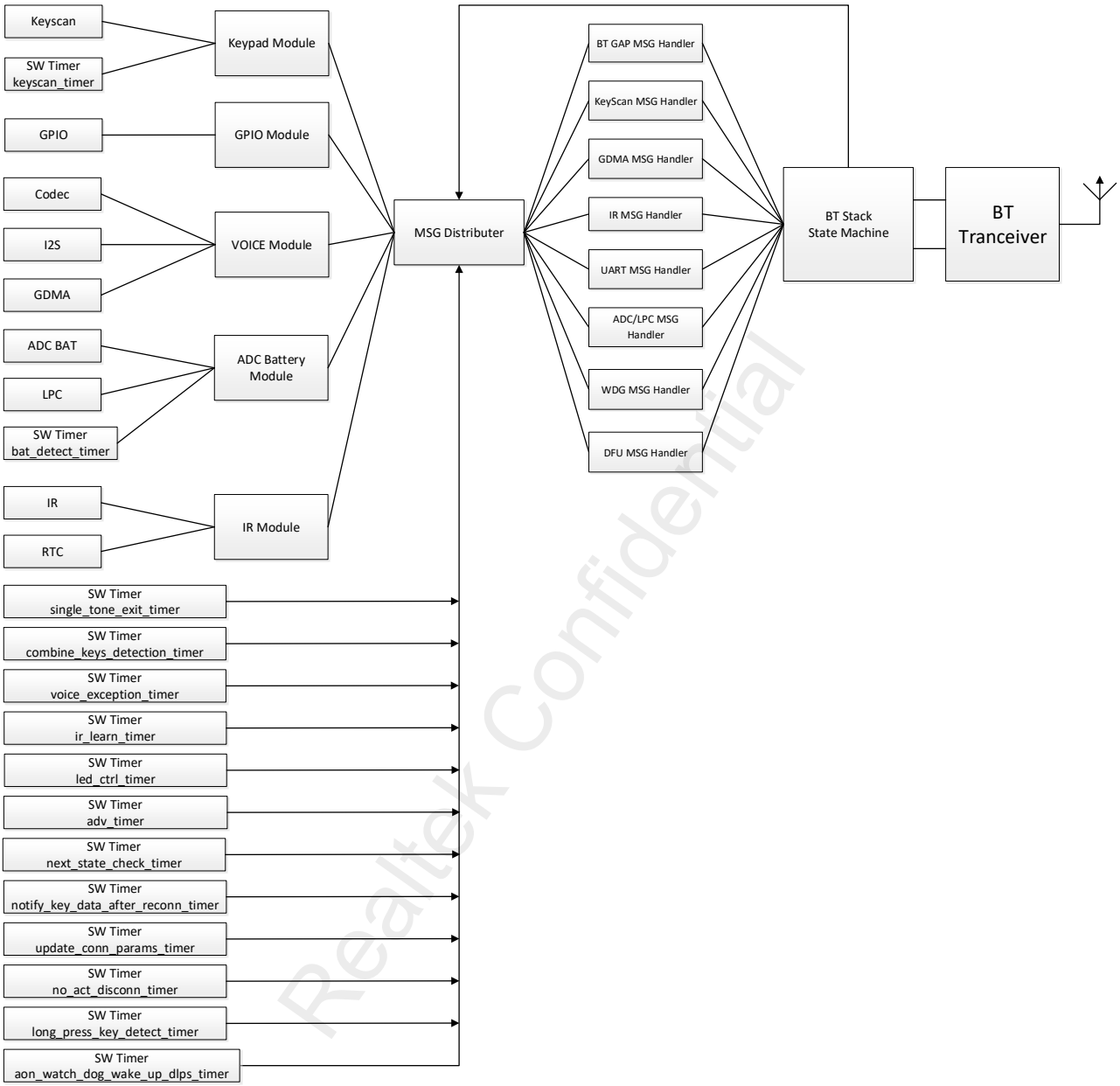

Figure 1 System Block Diagram

5 Initialization process

5.1 System startup flow

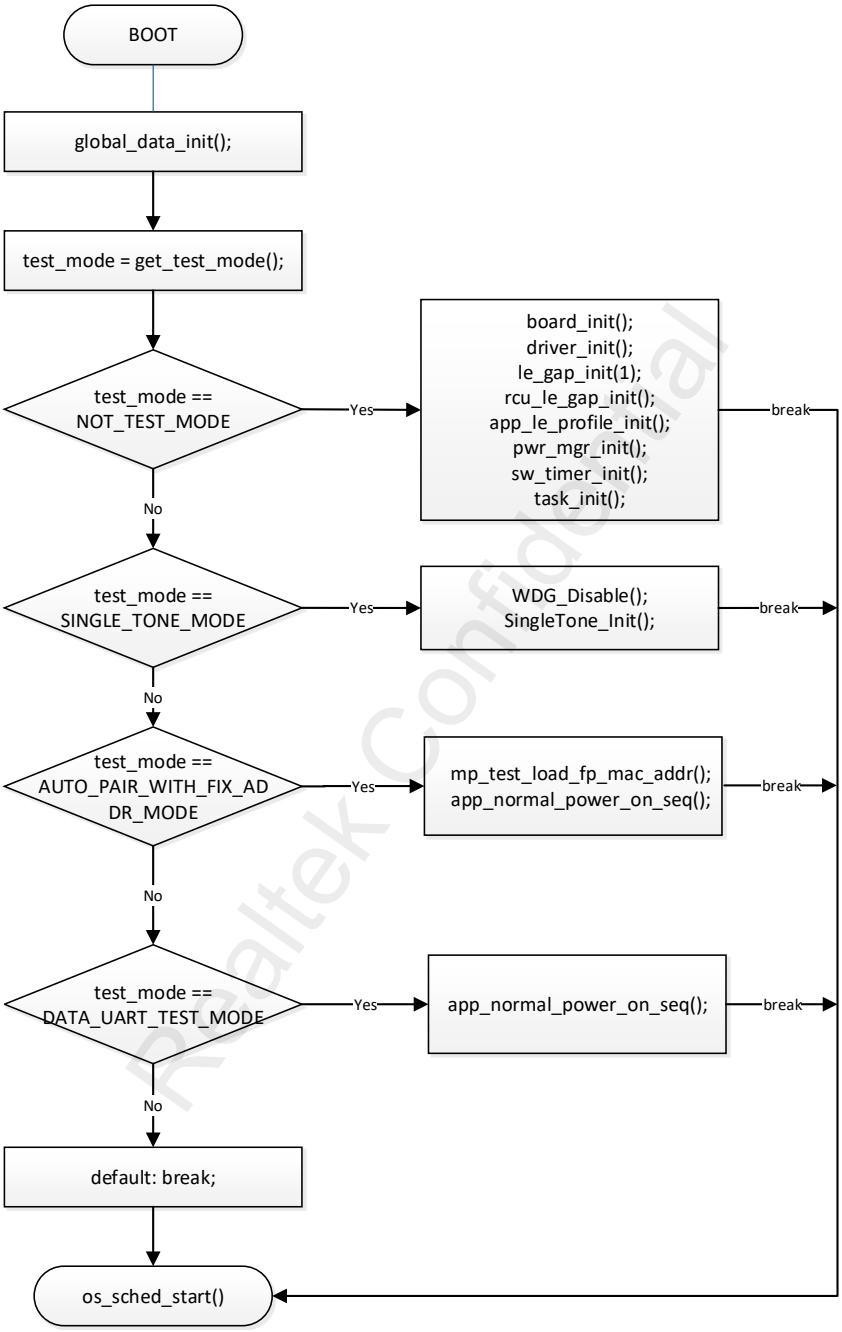

Figure 2 System Startup Flow

System Mode

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

5.2 Normal startup flow

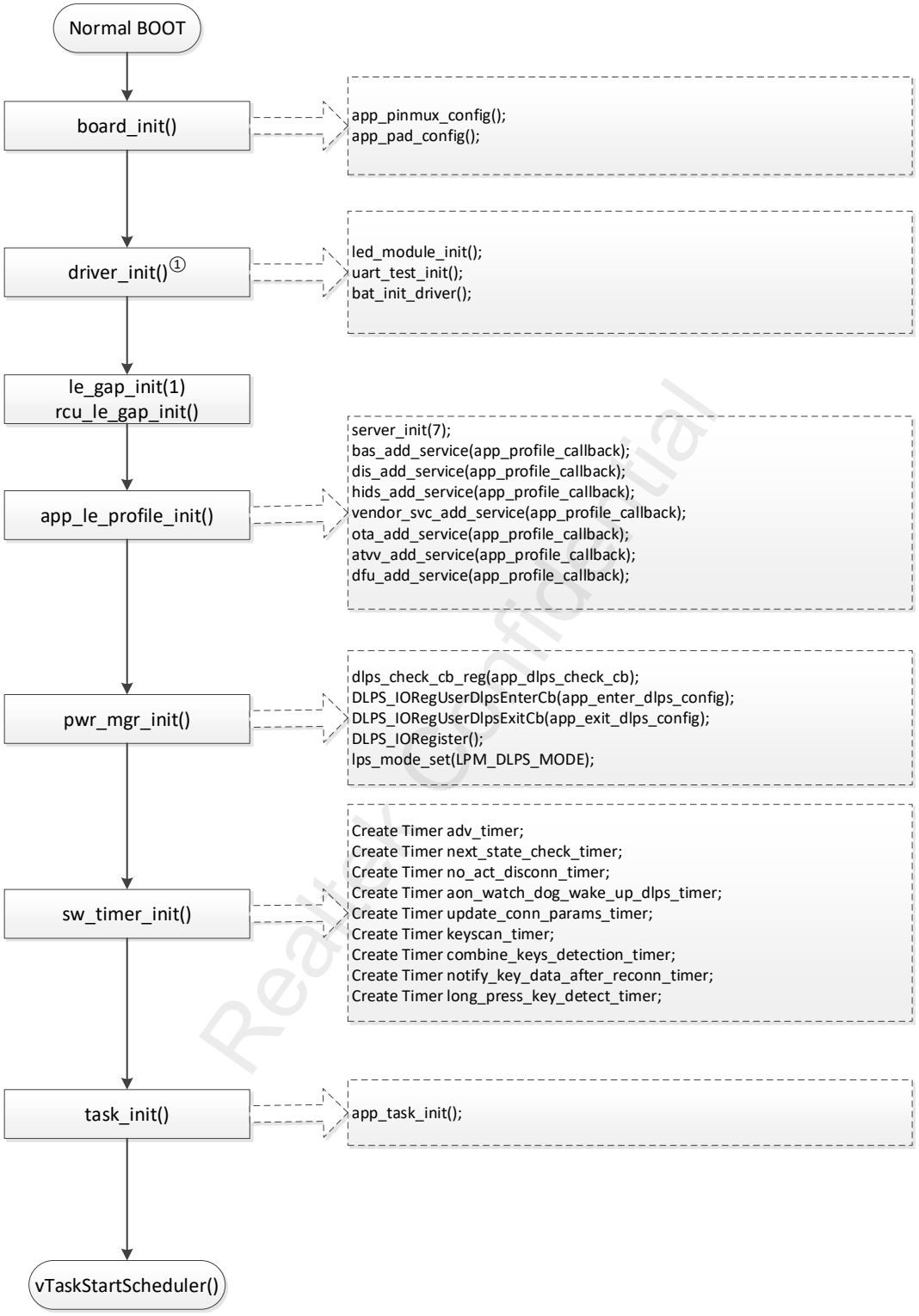

Figure 3 Normal startup flow

①: driver_init() does not contain keyscan_init_driver() function, but it is put in function app_nvic_config() together with keyscan_nvic_config(), which is to prevent the keyscan data generated during power on to NVIC enable time affecting the normal use of key module.

6 APP task

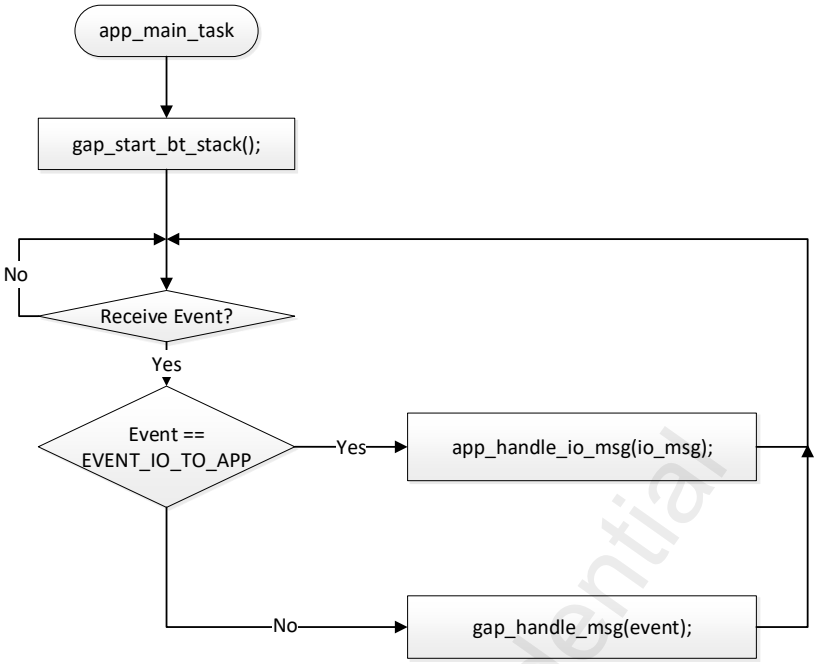

Figure 4 App task flow

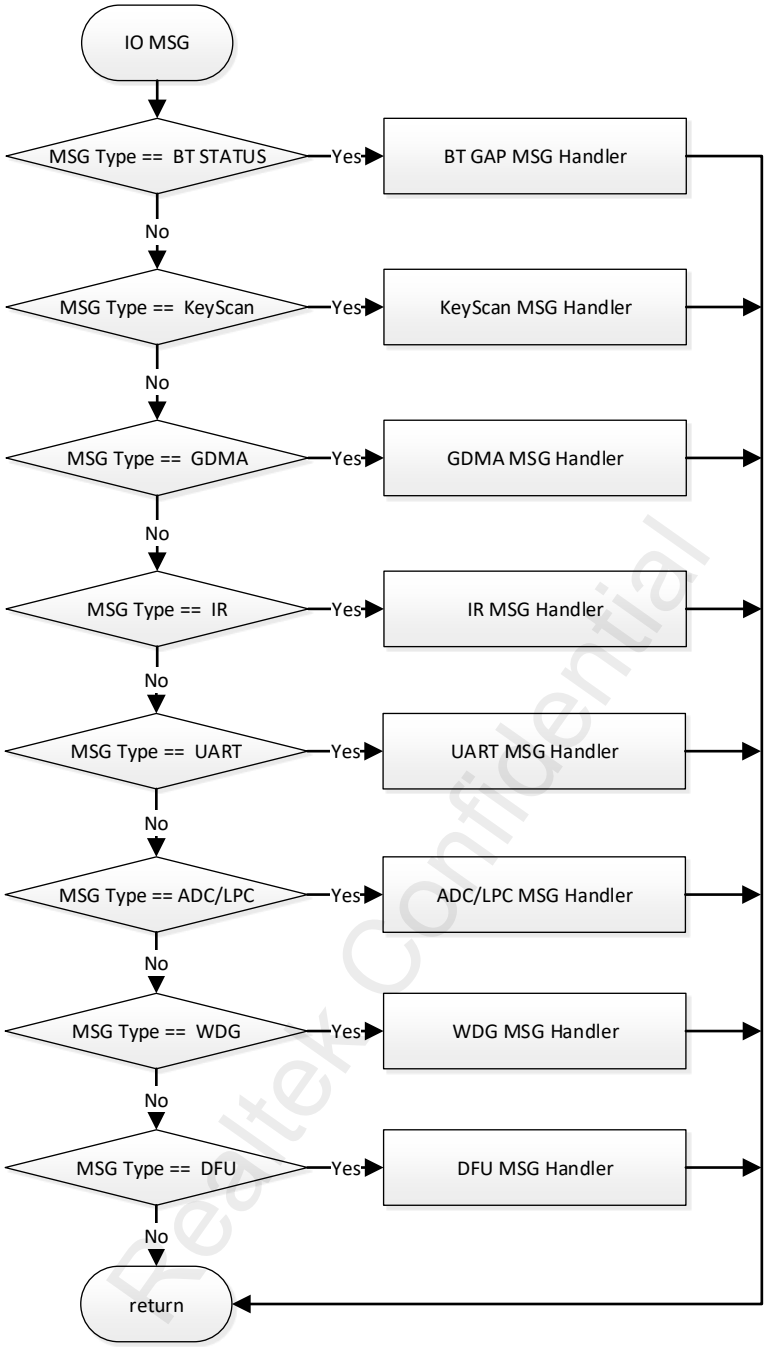

Figure 5 APP IO Message Handler

7 RCU status

7.1 RCU status

During the operation of the RCU, it switches between the following status.

Number	Status	Description
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

Figure 6 RCU Status

7.2 RCU state transformation diagram

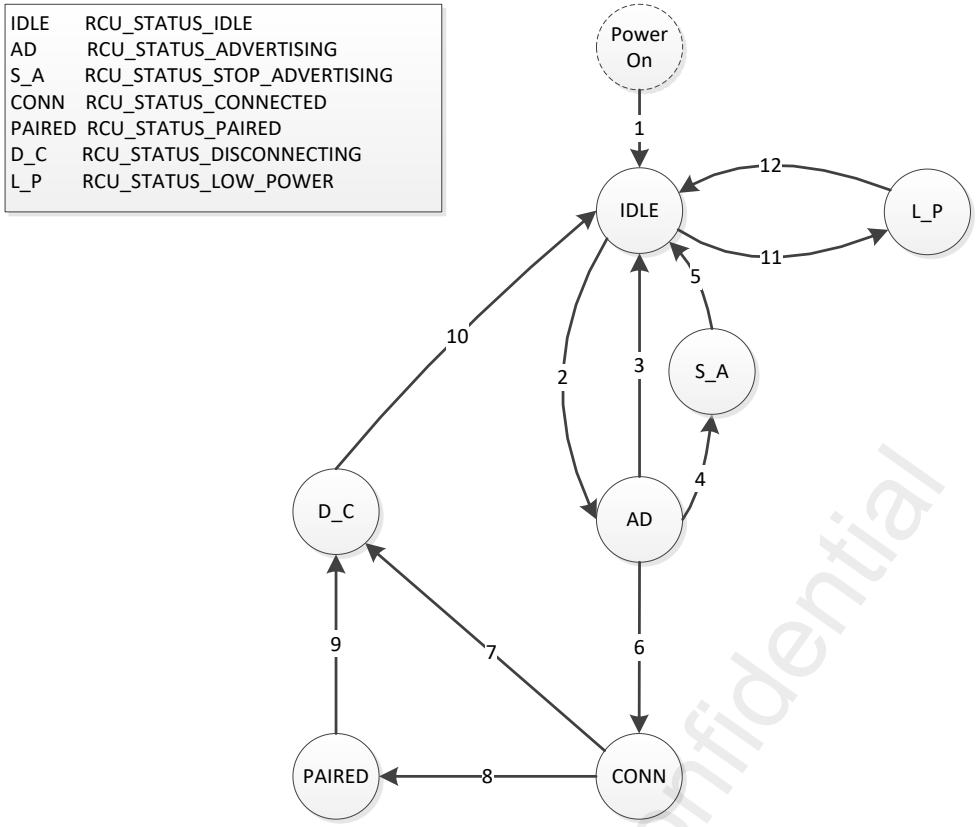

Figure 7 RCU state transformation diagram

7.3 RCU state transformation conditions

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

Figure 8 RCU state transformation conditions

7.4 RCU idle status description

Besides the case of system initialized upon first powered up, there are 2 scenarios in which system will switch to idle status.

A) No connection is established after an advertising event ends.

B) Connection is terminated under connection status.

For scenario A, the program issues a reason list for advertising termination, as enumerated below:

```
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	Description
1	STOP_ADV_REASON_IDLE	When receiving stack callback message for ADV_DIRECT_HDC timeout
2	STOP_ADV_REASON_PAIRING	Stop current advertisement to start pairing advertisement
3	STOP_ADV_REASON_TIMEOUT	When APP calls le_adv_stop after software advertising timer timeout
4	STOP_ADV_REASON_POWERKEY	Stop current advertisement to start power advertisement
5	STOP_ADV_REASON_LOWPOWER	Stop current advertisement to enter Low-Power mode
6	STOP_ADV_REASON_UART_CMD	When receiving UART stop advertisement command

Figure 9 Stop ADV reason

For scenario B, the program issues a reason list for disconnection, as enumerated below:

```
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	Description
1	DISCONN_REASON_IDLE	Default value
2	DISCONN_REASON_PAIRING	Terminate connection to start pairing advertisement
3	DISCONN_REASON_TIMEOUT	Terminate connection, if no action detected for a certain period (FEATURE_SUPPORT_NO_ACTION_DISCONN is enabled)
4	DISCONN_REASON_OTA	Terminate connection to start OTA process
5	DISCONN_REASON_PAIR_FAILED	Terminate connection for pairing failed
6	DISCONN_REASON_LOW_POWER	Terminate connection to enter low power mode
7	DISCONN_REASON_UART_CMD	Terminate connection to receive UART command
8	DISCONN_REASON_SILENT_OTA	Need to restart the silent upgrade, disconnect the BLE connection

Figure 10 Disconnect reason

8 SW timer application case and purpose

At present, the RCU application program defines 13 software timers for the following purposes.

No.	SW Timer	Description
1	adv_timer	Timer is used to stop advertising after timeout
2	next_state_check_timer	Timer is used to handle pairing exception timeout
3	no_act_disconn_timer	Timer is used to terminate connection after timeout if there is no action under connection
4	update_conn_params_timer	Timer is used to update desired connection parameters after timeout
5	aon_watch_dog_wake_up_dlps_timer	Timer is used for AON watch dog
6	keyscan_timer	Timer is used to detect keyscan status
7	combine_keys_detection_timer	Timer is used to detect combined keys after timeout
8	notify_key_data_after_reconn_timer	Timer is used to notify key data after timeout
9	voice_exception_timer	Timer is used to limit the maximum working time of a single ATV voice
10	long_press_key_detect_timer	Timer is used to detect the trigger of long press protection
11	ir_learn_timer	Timer is used for IR learning timeout
12	led_ctrl_timer	Timer is used to control LED timing
13	bat_detect_timer	Timer is used to check battery status
14	single_tone_exit_timer	Timer is used to exit Single Tone test mode after timeout

Figure 11 SW timer introduction

9 BLE advertising

9.1 Advertising packet format

RCU uses advertising packets in two formats:

1. Undirected advertising event: AdvA field is an address of the device sending advertising packets, and AdvData is in the format as is shown in Figure 13

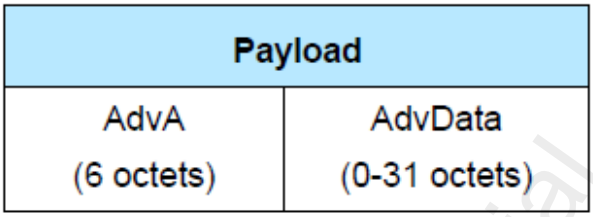

Figure 12 ADV_IND PDU payload

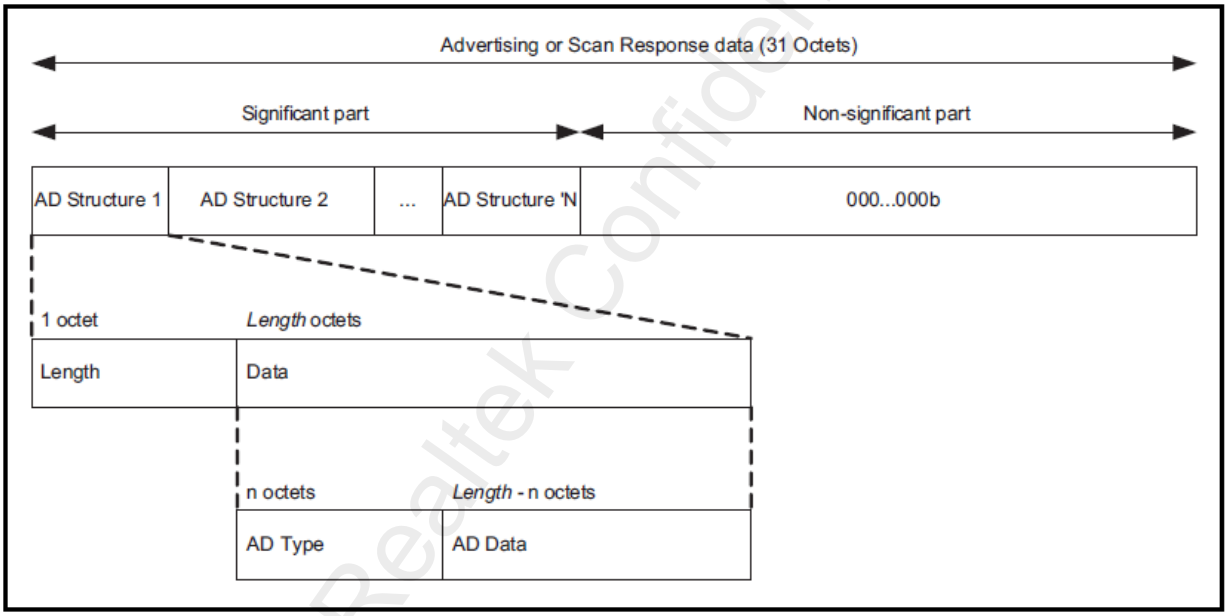

Figure 13 Advertising and Scan Response data format

2. Directed advertising event: AdvA field is an address of the device sending advertising packets, while InitA is an address of the remote device.

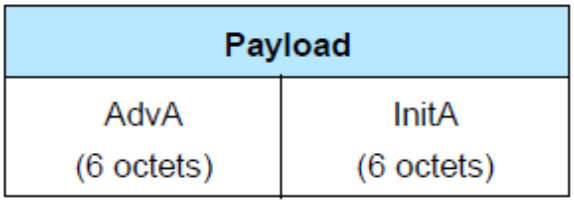

Figure 14 ADV_DIRECT_IND PDU payload

9.2 Advertising packet type

Bee3 RCU shall call rcu_start_adv to set advertisement type and start advertising, and the advertising type is enumerated below:

```
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;
```

There are five valid advertising types of RCU: ADV_UNDIRECT_PAIRING, ADV_DIRECT_HDC, ADV_UNDIRECT_RECONNECT, ADV_UNDIRECT_PROMPT and ADV_UNDIRECT_POWER, corresponding to pairing mode, directed reconnecting mode, undirected reconnecting mode, prompt pairing mode and power on mode.

9.2.1 Pairing advertising packet

RCU sends pairing advertising packets in pairing mode to activate automatic pairing flow initiated by master device. It is recommended to set the range of Advertising Interval to 0x20 - 0x30 (20ms - 30ms), and the advertising timeout can be configured by macro ADV_UNDIRECT_PAIRING_TIMEOUT which is set to 30s by default. Pairing mode will be activated in one of the following 2 cases:

1. Press and hold pairing combined keys (<<Volume+>> and <<OK>>) for 2s, Link Key stored in RCU will be cleared and the RCU will enter pairing mode.
2. If RCU is not paired before and macro FEATURE_SUPPORT_REMOVE_LINK_KEY_BEFORE_PAIRING is set to 1, any key event will activate RCU to enter pairing mode.

If FEATURE_SUPPORT_REMOVE_LINK_KEY_BEFORE_PAIRING is set to 0, only pairing combined keys will trigger paring, other keys are invalid. In RCU SDK, default value of this macro is 0.

Pairing advertising packet is in the following format:

Flag field （byte0~2）	Service Field （byte3~6）	Appearance Field – Device type (byte7~10)	Local Name Field (byte11~22)	Vendor Information Field (byte23~28)
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

Figure 15 Pairing advertising packet

9.2.2 Reconnection advertising packet

RCU sends reconnection advertising packets and connection can be established quickly if link key has been stored.

RCU can choose whether to support the function of resolving peer device privacy address according to the macro FEATURE_SUPPORT_PRIVACY in board.h, and FEATURE_ SUPPORT_ Privacy is turned on by default.

1. When FEATURE_SUPPORT_PRIVACY is 0, the format of Reconnection Advertising packet is Direct Advertising High Duty Packet, which is sent continuously for 1.28s each time. After timeout, stack will automatically stop advertising and send message to application layer. By default, application will make 3 reconnecting attempts.

2. When FEATURE_SUPPORT_PRIVACY is 1, The RCU uses the method of Undirect Advertising + White List to reconnect. The content of the undirect advertising package is the same as pairing advertising package. The timeout is defined by ADV_UNDIRECT_RECONNECT_TIMEOUT, and the default value is 3s.

RCU will enter reconnection mode in one of the following two cases.

1. RCU enters reconnection mode when it is powered on, if it has ever been paired successfully and Link Key is stored in it.
2. After RCU connecting to the peer device, any key event will activate it to enter reconnection mode when the connection is terminated in case that the peer device is shutdown, or the RCU is out of range.

9.2.3 Prompt advertising packet

Bee3 RCU transmits non-connectable advertising packets to activate the peer device to pop up a connection prompt, prompting user to perform pairing operations by the following the instructions. Prompt advertising packet is Undirected Advertising Packet, and the Advertising Interval can be set to 0x20-0x30 (20ms-30ms). The advertising timeout is set by macro definition ADV_UNDIRECT_PROMPT_TIMEOUT, which is 5 seconds by default.

In SDK, prompt advertising can be enabled or disabled by FEATURE_SUPPORT_UNDIRECT_PROMPT_ADV. In default, FEATURE_SUPPORT_UNDIRECT_PROMPT_ADV is set to 0, which means RCU shall not send prompt advertising packets. If FEATURE_SUPPORT_UNDIRECT_PROMPT_ADV is set to 1, prompting advertising mode will be activated in the following cases:

1. In case of available link key, press any key, except for Power key, to enter reconnection mode. If connection cannot be established with reconnection advertising packets, prompting pairing mode is activated to send prompt advertising packet.

Prompt advertising packet format: (the difference with pairing advertising packet is that the Limited Discoverable Mode bit is 0 and non-connectable)

Flag field （byte0~2）	Service Field （byte3~6）	Appearance Field – Device type (byte7~10)	Local Name Field (byte11~22)	Vendor Information Field (byte23~28)
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

Figure 16 Prompt advertising packet format

9.2.4 Power-on advertising packet

RCU sends power-on advertising packet to wake up peer device in sleep mode, power-on advertising packet is Undirected Advertising Packet, Advertising interval range is 0x20 - 0x30(20ms - 30ms), the advertising timeout is defined through macro ADV_UNDIRECT_POWER_TIMEOUT which is set to 12 seconds by default.

In the following conditions, it will enter the power on mode: Press power key in the BLE disconnection state.

Power-On advertising packet format:

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,0x00,0x03,0x00,0x01,0xXX(index), 0xXX,0xXX, 0xXX, 0xXX, 0xXX,0xXX,(TV BD Address)

Figure 17 Power-On advertising packet format

9.3 Advertising parameter configuration

9.3.1 Device name

In Bee3 RCU SDK, the BLE device name can be defined through micro C_DEVICE_NAME and C_DEVICE_NAME_LEN.

```
#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 and PID

In Bee3 RCU SDK, the VID and PID can be defined through micro VID and PID.

```
1. #define VID  0x005D
2. #define PID  0x0001
```

10 Connection and Pairing

10.1 Pairing procedure

According to the different status of pairing information, the connection and pairing process of RCU and host terminal can be divided into the following situations:

1. When RCU and the peer device have been paired and the pairing information has been saved, any key on RCU can reconnect quickly without pairing again.
2. When RCU and the peer device do not save the pairing information, the pairing combination key should be pressed to trigger the automatic pairing process, then establish the connection and process pairing procedure.
3. When RCU saves the pairing information but the peer device clears the pairing information, RCU cannot establish a connection by sending reconnection advertising packets. If RCU sends the undirected reconnection advertising or presses the combination key to trigger the automatic pairing process, establish the connection and pair.
4. While RCU clears the pairing information but the peer device saves the previous pairing information, RCU can establish a connection by sending reconnection advertising. The peer device goes back to the connection process, however the key missing will be occurred and pairing failure will appear on RCU side. In order to deal with the above failure scenario, RCU will disconnect and send the pairing advertising again to establish the connection with the peer device, and then pair again.

10.2 Connection parameter update

In order to reduce the power consumption of the connection, after the connection is bounded, RCU will send a request to update the connection parameters, requesting the peer device to use the specified connection parameters. According to the current process of pairing or reconnecting, the timing of RCU to actively update the connection will be different:

1. Pairing process: after the pairing is successful, start software timer and make parameter update request after UPDATE_CONN_PARAMS_TIMEOUT to ensure that GATT service discovery and CCCD enabling process can be completed quickly;
2. Connection process:
    a. The voice button triggers the reconnection process: if the voice is working after reconnection, it is not allowed to immediately update the parameter request, start software timer, and judge whether the voice is still working after UPDATE_CONN_PARAMS_TIMEOUT. if so, restart timer. Otherwise, send the parameter update request;

b. Common keys trigger reconnection process: after the reconnection is completed, send the parameter update request immediately;

Parameter update timeout time can be configured through UPDATE_CONN_PARAMS_TIMEOUT in swtimer.h, and the default is 5 seconds.

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

In BEE3 RCU SDK, connection parameters can be configured by macro in rcu_application.h. The connection parameters in default application scheme is: Connection Interval 15ms, Slave Latency 49, Supervision Timeout Period 5s

```
1. #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 support

Turn on or turn off Privacy Feature by FEATURE_SUPPORT_PRIVACY in board.h, and the function is turned on by default. After turn on the macro FEATURE_SUPPORT_PRIVACY, RCU supports the resolving and reconnecting operation of peer device Resolvable Random Address.

1. When FEATURE_SUPPORT_PRIVACY is 1, RCU uses Undirected Adv and White List to reconnect, the timeout should be one time, and last 3 seconds. It is expected to reconnect peer device who have turned on Privacy function;
2. When FEATURE_SUPPORT_PRIVACY is 0, RCU use High Duty Cycle Directed Adv to reconnect, the timeout should be 3 times, and last 1.28 seconds each time. It is expected to reconnect peer device only who have turned off Privacy function;

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

10.4 No-operation disconnection timeout

No-operation disconnection timeout can be controlled by FEATURE_SUPPORT_NO_ACTION_DISCONN in board.h, and it is turned off by default. When the macro is turned on, if RCU is in pairing state and there is no key or OTA operation within the NO_ACTION_DISCON_TIMEOUT period, RCU will take the initiative to disconnect to save power consumption.

NO_ACTION_DISCON_ TIMEOUT can be configured by NO_ACTION_DISCON_TIMEOUT in swtimer.h and it is 5 minutes by default.

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

10.5 Clear pairing information before pairing

The macro FEATURE_SUPPORT_REMOVE_LINK_KEY_BEFORE_PAIRING can control the clear pairing information before pairing function, and it is turned on by default. When this macro is turned on, the previous pairing information will be cleared before pairing, otherwise the pairing information will be kept until the pairing information is updated.

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

10.6 Backup and restore pairing information

The macro FEATURE_SUPPORT_RECOVER_PAIR_INFO can be set to control the backup and restore pairing information function, and it is turned off by default. This function will only take effect when the configuration macro FEATURE_SUPPORT_RECOVER_PAIR_INFO is set to 1. This function is designed to ensure that a set of pairing information is always stored in the paired RCU. When then function is turned on, the current pairing information is backed up before clearing the pairing information. After that if the RCU is not successfully paired with other devices and enters the idle state, the backup pairing information will be restored then it will be cleared.

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

10.7 One key pairing

One key pairing function can be controlled by macro FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV, and it is turned off by default. When FEATURE_SUPPORT_ANY_KEY_TRIG_PAIRING_ADV is turned on, if RCU does not store pairing information, any key can trigger pairing advertising.

```
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

Bee3 RCU supports the following BLE Services:

No.	Service	Description
1	GAP	Generic Access Profile
2	DIS	Device Information Service
3	HIDS	HID Service
4	BAS	Battery Access Service
5	OTA Service	Realtek OTA Service
6	DFU Service	Realtek Silent OTA Service
7	Vendor Service	Realtek RCU Test Tool Service
8	ATV Voice Service	Google Vendor ATV Voice Service
9	RTK Voice Service	RTK GATT Voice Service

Figure 18 BLE Service

12 Key

12.1 Keyscan scheme

To achieve the minimum power consumption of the Keyscan module, RCU uses the hardware Keyscan + SW Timer + HW debounce solution.

12.1.1 Keyscan working mode

Keyscan module provides two working mode: Key triggered manual mode and Register triggered manual mode.

1. When Key triggered manual mode is working, Keyscan module will automatically trigger scan operation when Key pressed event detected.
2. When Register triggered manual mode is working, scan operation will be triggered through software setting, instead of triggered automatically by key pressed event.

12.1.2 HW Debounce

Keyscan mode have two debounce mechanism to dealing with two different modes Active Mode and Sleep Mode: Keyscan HW Debounce and PAD HW Debounce.

1. System in Active mode: Keyscan mode is in working mode, Keyscan HW Debounce is used;
2. System in Sleep mode: Keyscan mode is not in working mode, PAD HW Debounce is used. In the debounce stage, the system can also stay in low power consumption mode;

12.1.3 Keyscan detect flow

Keyscan detect flow is shown in the figure below:

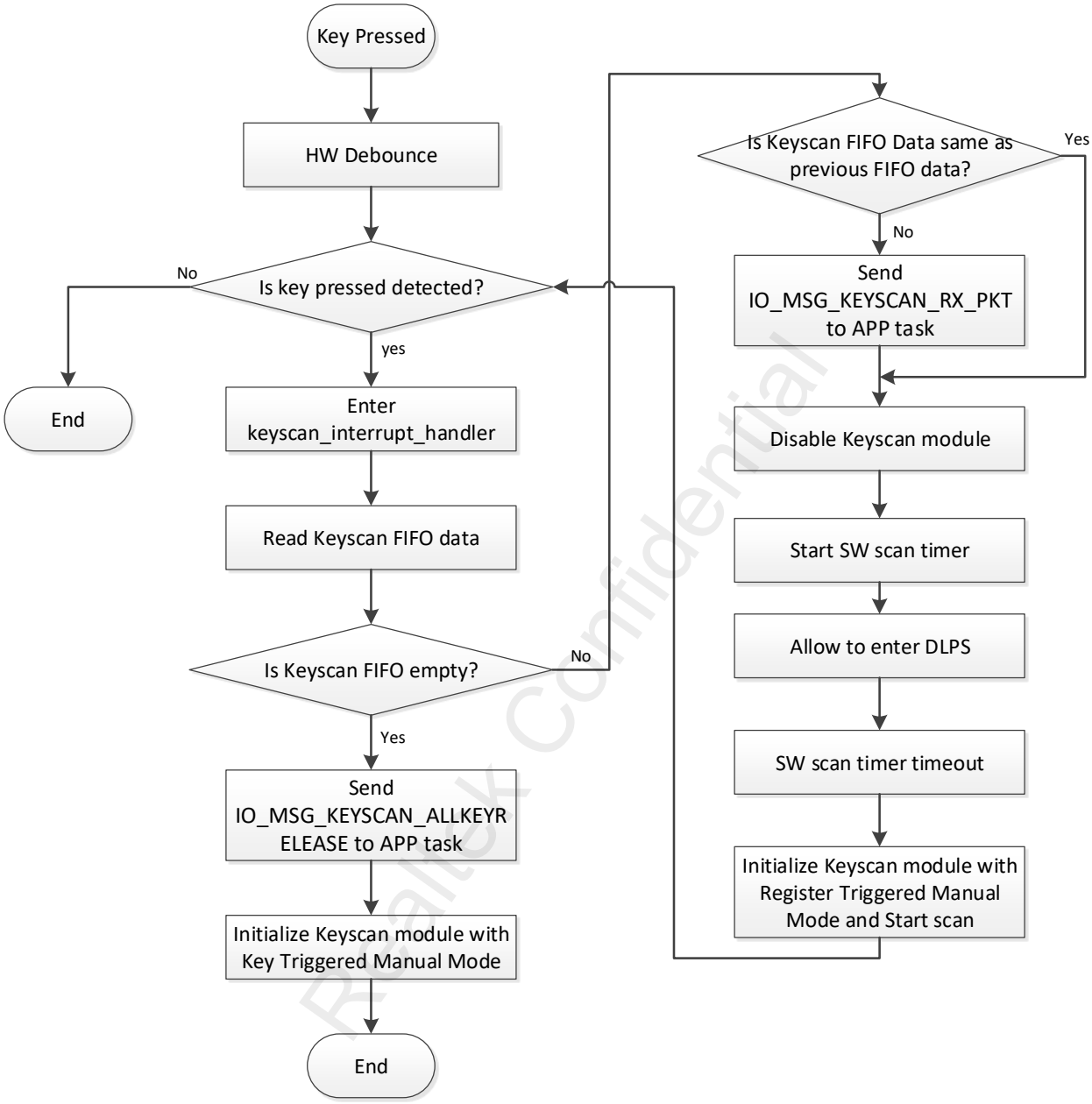

Figure 19 Keyscan flow

After power on, initialize KeyScan module to Key Triggered Manual Mode and wait for the keystroke event to occur.

When any key is pressed, a key matrix scan will be performed after HW debounce. After scanning, scan interrupt will be generated to read Keyscan FIFO data in the interrupt processing function.

If the Keyscan FIFO is empty, it means that the key has been released, then IO_MSG_KEYSCAN_ ALLKEYRELEASE message will be sent to app task for processing. At the same time, reinitialize the Keyscan

module to Key Triggered Manual Mode and wait for the next key event.

If the Keyscan FIFO has data and is different from the last Keyscan FIFO data saved, IO_MSG_KEYSCAN _RX_PKT message will be sent to app task for processing. If FIFO data is the same, message sending is skipped. Then disable the Keyscan module, and start the software timer, which allows system to enter DLPS. After that, Software timer timeout wakes up the system, reinitializes the Keyscan module to Register Triggered Manual Mode, triggers Keyscan module to scan once, and continues to detect the status of the key.

12.2 Keyscan configuration

In RCU application, the pin of the row and column is defined in board.h and can be modified according to the actual project requirements:

```
/* 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

#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

#endif
```

The key definition of Keyscan is in key_handle.c, it can be modified according to the actual project requirements:

```
/* Key Mapping Table Definiton */
#if (RCU_HD_PLATFORM_SEL == R_DEMO_RCU)
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_NONE,             VK_NONE,      VK_NONE},
};
#elif (RCU_HD_PLATFORM_SEL == H_DEMO_RCU)
static const T_KEY_INDEX_DEF KEY_MAPPING_TABLE[KEYPAD_ROW_SIZE][KEYPAD_COLUMN_SIZE] =
{
   {VK_POWER,            VK_VOICE,              VK_HOME,       VK_MENU},
   {VK_VOLUME_UP,        VK_VOLUME_DOWN,        VK_ENTER,      VK_EXIT},
   {VK_LEFT,                 VK_RIGHT,              VK_UP,           VK_DOWN},
   {MM_VolumeIncrement,  MM_VolumeDecrement,  VK_TV_POWER,  VK_TV_SIGNAL},
};
#elif (RCU_HD_PLATFORM_SEL == G_MIN_DEMO_RCU)
static const T_KEY_INDEX_DEF KEY_MAPPING_TABLE[KEYPAD_ROW_SIZE][KEYPAD_COLUMN_SIZE] =
{
    {VK_POWER,           MM_AC_Back,          MM_DPadCenter,  MM_Dashboard},
    {MM_Mute,            VK_VOLUME_UP,        MM_DPadDown,    MM_DPadRight},
```

```
    {MM_AC_Bookmarks,    VK_VOLUME_DOWN,     MM_AC_Search,   MM_Guide},
    {MM_DPadUp,          VK_YOUTUBE,         MM_AC_Home,     MM_Live},
    {MM_DPadLeft,        VK_NETFLIX,         VK_APP04,       VK_NONE},
};
#endif
```

12.3 Key type

According to different key functions, Bee3 RCU SDK divides keys into four different types:

```
/* define the key types */
typedef enum
{
    KEY_TYPE_NONE        = 0x00, /* none key type */
    KEY_TYPE_BLE_ONLY    = 0x01, /* only BLE key type */
    KEY_TYPE_IR_ONLY     = 0x02, /* only IR key type */
    KEY_TYPE_BLE_OR_IR   = 0x03, /* BLE or IR key type */
} T_KEY_TYPE_DEF;
```

1. KEY_TYPE_NONE: Do nothing when key pressed, BLE or IR key value will not be sent;
2. KEY_TYPE_BLE_ONLY: Only send BLE HID value;
3. KEY_TYPE_IR_ONLY: Only send IR value;
4. KEY_TYPE_BLE_OR_IR: Send the key value according to the BLE status, the BLE key value will be sent in connected status, and the IR key value will be sent in disconnected status;

12.4 Key value definition

In Bee3 RCU SDK, the key type, IR key value, BLE HID key value and other parameters are defined through KEY_CODE_TABLE, which can be adjusted according to specific project requirements.

```
/* BLE HID code table definition */
const T_KEY_CODE_DEF KEY_CODE_TABLE[KEY_CODE_TABLE_SIZE] =
{
    /* key_type,          ir_key_code,   hid_usage_page,       hid_usage_id */
    {KEY_TYPE_NONE,       0x00,          HID_UNDEFINED_PAGE,   0x00},   /* VK_NONE */
    {KEY_TYPE_BLE_OR_IR,  0x18,          HID_KEYBOARD_PAGE,    0x66},   /* VK_POWER */
    {KEY_TYPE_BLE_OR_IR,  0x56,          HID_KEYBOARD_PAGE,    0x76},   /* VK_MENU */
    {KEY_TYPE_BLE_OR_IR,  0x14,          HID_KEYBOARD_PAGE,    0x4A},   /* VK_HOME */
    {KEY_TYPE_BLE_ONLY,   0x3E,          HID_KEYBOARD_PAGE,    0x3E},   /* VK_VOICE */
    ......
    {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 Key behavior rules

This section mainly introduces key behavior rules and provides a scheme for key processing, which can be adjusted according to specific project requirements. There are different key behaviors based on the number of keys detected and the current state of the RCU.

12.5.1 Single key behavior

After a single key is pressed, RCU application software receives IO_MSG_KEYSCAN_RX_PKT sent by the Keyscan module. According to the current state of the APP, the following processing is carried out.

- RCU_STATUS_IDLE

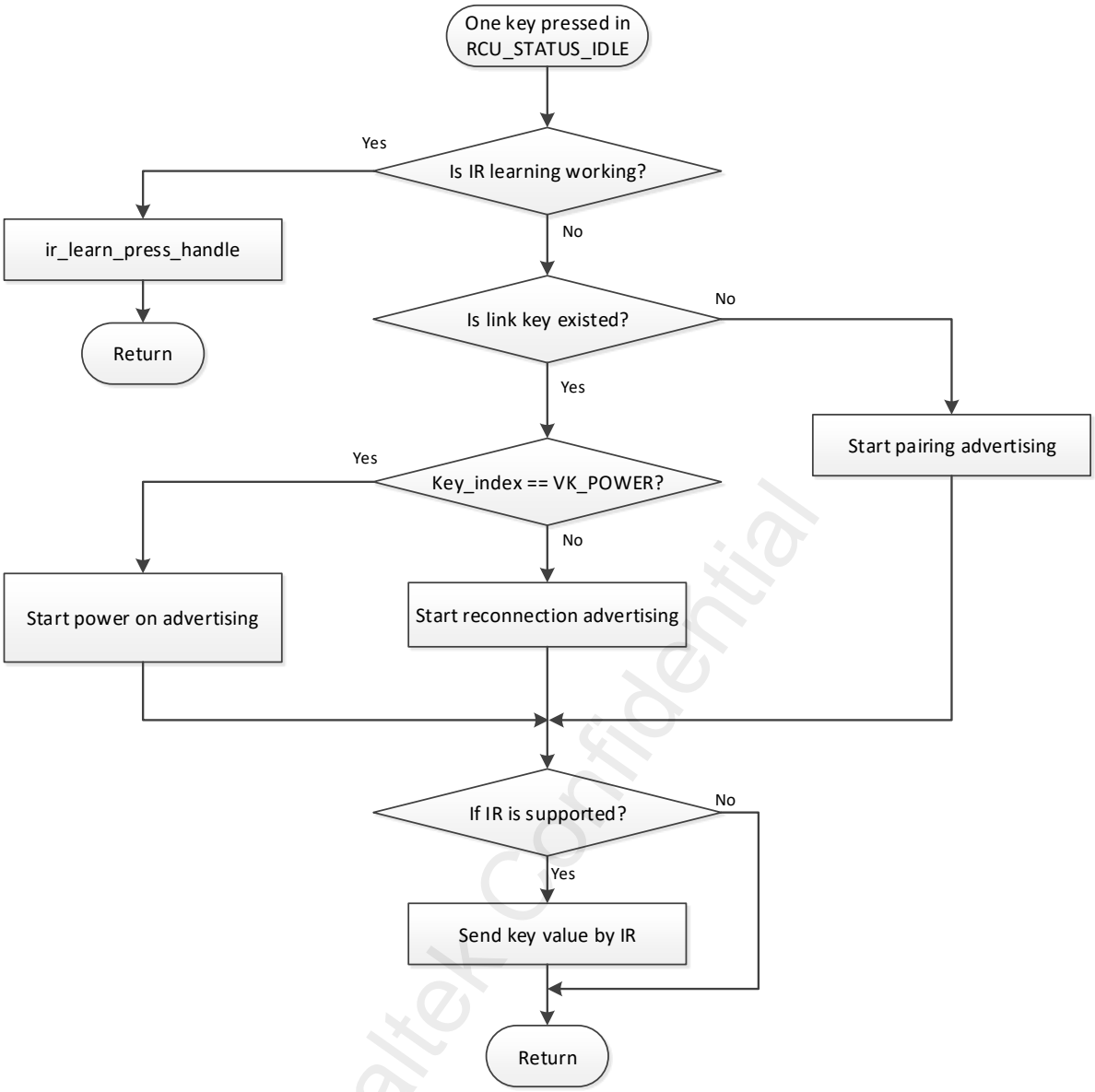

Figure 20 RCU_STATUS_IDLE Single Key Processing Flow

- RCU_STATUS_ADVERTISING

Figure 21 RCU_STATUS_ADVERTISING Single Key Processing Flow

- RCU_STATUS_PAIRED

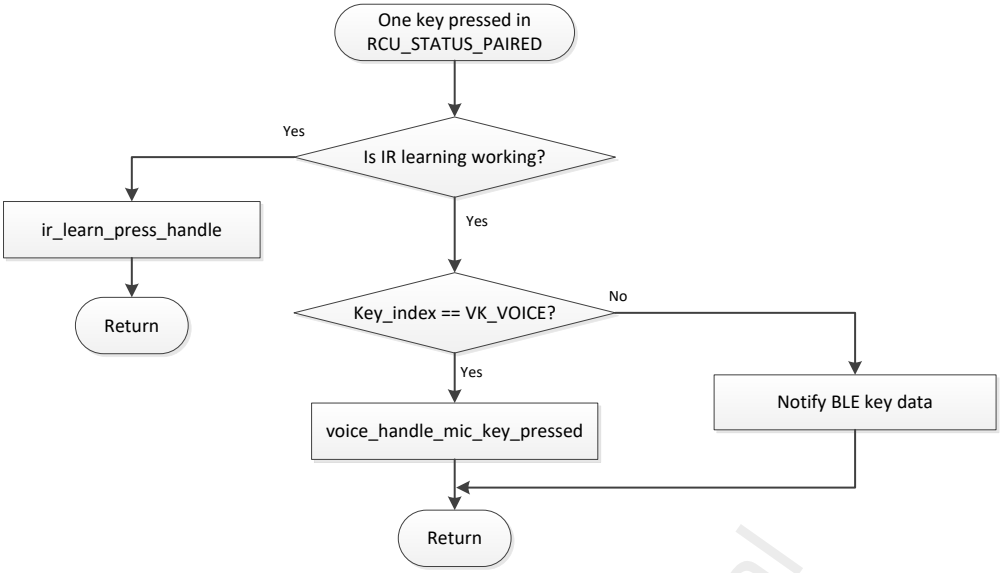

Figure 22 RCU_STATUS_PAIRED Single Key Processing Flow

- Other status

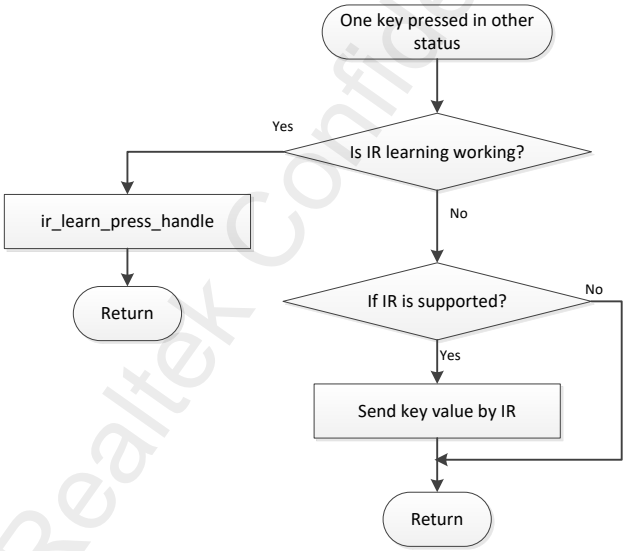

Figure 23 Other States Single Key Processing Flow

12.5.2 Multiple key behavior

When multiple keys are pressed at the same time, Bee3 RCU performs different logic processing according to the macro FEATURE_SUPPORT_REPORT _MULTIPLE_HID_KEY_CODES setting in board.h:

1. If FEATURE_SUPPORT_REPORT_MULTIPLE_HID_KEY_CODES is set to 0, After multiple keys are pressed, the previous key operation will be stopped, the same as the key release process;
2. If FEATURE_SUPPORT_REPORT_MULTIPLE_HID_KEY_CODES is set to 1 and RCU is in RCU_STATUS_PAIRED status, notification with multiple keys value will be sent;

12.6 Bluetooth reconnect and resend key value

RCU needs to resend BLE HID value after reconnecting, when KEY_TYPE_BLE_ONLY or KEY_TYPE_BLE_OR_IR type key is pressed and IR value is not sent, to handle with RCU_STATUS_IDLE and RCU_STATUS_ADVERTISING state:

1. If the key is pressed and released before reconnecting, BLE HID key value ‘press’ and ‘release’ should be resent.
2. If the key is still pressed (long press) after reconnecting, only BLE HID key value ‘press’ should be resent. BLE HID key value ‘release’ will be sent when key is released.

12.7 Special combination key behavior

When the key is pressed, RCU app will detect whether it is a special combination key defined. If it is a combination key, it will turn on SW timer to confirm whether the combination key meets the pressing time requirements.

RCU app scheme uses a 32bit global variable to distinguish different combination keys. Each combination key corresponds to a bit. At present, there are several groups of combination keys as follows, which can be adjusted according to specific project requirements.

No.	Bit Mask	combination keys	introduction
1	0x0001	VK_ENTER + VK_VOLUME_UP	Enter pairing mode
2	0x0002	VK_ENTER + VK_LEFT	Enter ir learning mode（IR_LEARN_MODE valid）
3	0x0004	VK_POWER + VK_VOLUME_UP	Enter HCI UART test mode （MP_TEST_MODE_SUPPORT_HCI_UART_TEST valid）
4	0x0008	VK_POWER + VK_VOLUME_DOWN	Enter Data UART test mode （MP_TEST_MODE_SUPPORT_DATA_UART_TEST valid）
5	0x0010	VK_POWER + VK_EXIT	Enter SingleTone test mode （MP_TEST_MODE_SUPPORT_SINGLE_TONE_TEST valid）
6	0x0020	VK_POWER + VK_UP	Enter fast pair mode 1 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST valid）
7	0x0040	VK_POWER + VK_DOWN	Enter fast pair mode 2 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST valid）
8	0x0080	VK_POWER + VK_LEFT	Enter fast pair mode 3 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST valid）
9	0x0100	VK_POWER + VK_RIGHT	Enter fast pair mode 4 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST valid）
10	0x0200	VK_POWER + VK_ENTER	Enter fast pair mode 5 （MP_TEST_MODE_SUPPORT_FAST_PAIR_TEST valid）
11	0x8000	VK_LEFT + VK_HOME + VK_MENU	Factory reset

Figure 24 Special combination key definition

12.8 Long press protection

The macro FEATURE_SUPPORT_KEY_LONG_PRESS_PROTECT can be set to control the long press protection function, and it is turned off by default. When the function is turned on, the press time will be detected. When one or more keys are pressed for more than a certain period of time, the remote control will simulate the key release operation, and all the keys will be invalid. Only when all keys are released the RCU keys can continue to work normally.

The detection time of long press protection can be modified by macro LONG_PRESS_KEY_DETECT_TIMEOUT.

```
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 Voice

13.1 Voice module introduction

RCU supports voice function, which recording voice information by Digital Microphone (DMIC) or Analog Microphone (AMIC). Then voice data is encoded and sent to the peer device by BLE. After that, with the help of speech recognition software, we can achieve a variety of applications. The voice module block diagram is shown in figure 25.

When using DMIC, RCU directly move the digital voice data to CODEC. Then COEDC encodes raw data into PCM format and put it into I2S FIFO. After that, GDMA automatically moves the encoded data into RAM which can be used by App.

When using AMIC, RCU uses AD converter to process input analogy voltage signal firstly, then CODEC encodes converted data into PCM format and put it into I2S FIFO. After that, as same as DMIC, GDMA automatically moves the encoded data into RAM, which can be used by App.

The sampling frequency supported by Bee3 Codec is 8KHz and 16KHz, which can be chosen by macro CODEC_SAMPLE_RATE_SEL.

Bee3 supports 5-band software EQ to do some special processing for the voice data.

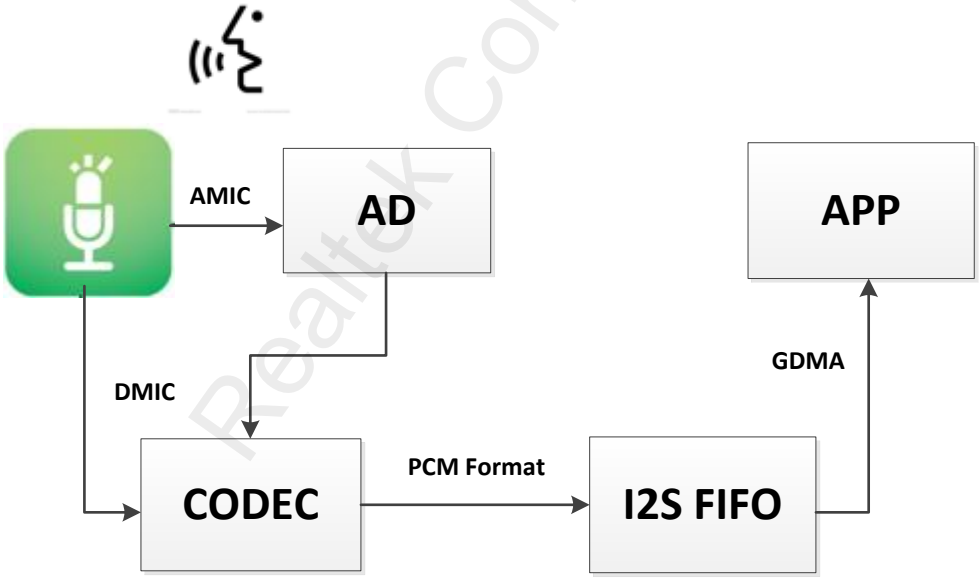

Figure 25 voice function diagram

13.2 Voice module initialization

Voice module initialization includes, voice module data initialization and I/O initialization such as Pad/ Pinmux/ I2S/ CODEC/ GDMA. In program, the initialization of voice module calls voice_handle_start_mic function.

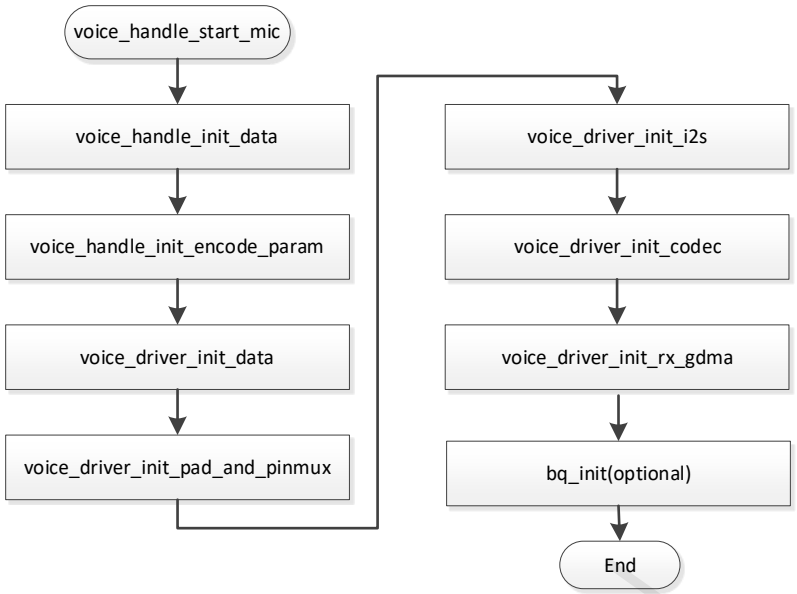

Figure 26 Voice module initialization flow

13.2.1 Voice PAD initialization

If Bee3 RCU uses AMIC, the three PAD MIC_N, MIC_P and MICBIAS which corresponding to P2_6, p2_7 and H0 need to be used. If DMIC is used, the two PAD DMIC_CLK and DMIC_DAT which use P2_6 and P2_7 by default need to be used. PINPUX can be used to switch to other GPIO.

The specific initialization code is as below:

```
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);
13.     Pad_Config(DMIC_DATA_PIN, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OU
    T_DISABLE,
14.                PAD_OUT_LOW);
15.     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 Voice I2S initialization

Bee3 RCU uses I2S to transfer PCM data of CODEC to APP. In I2S configuration, it is necessary to note that the BCLK configuration should be consistent with the sampling rate of CODEC. The specific initialization code is as below:

```
void voice_driver_init_i2s(void)
{
    RCC_PeriphClockCmd(APBPeriph_I2S0, APBPeriph_I2S0_CLOCK, ENABLE);

    I2S_InitTypeDef I2S_InitStruct;
    I2S_StructInit(&I2S_InitStruct);
    I2S_InitStruct.I2S_ClockSource      = I2S_CLK_40M;
    I2S_InitStruct.I2S_DataFormat       = I2S_Mode;
    I2S_InitStruct.I2S_DeviceMode       = I2S_DeviceMode_Master;
    I2S_InitStruct.I2S_RxWaterlevel     = 0x4;

    /* config I2S clock speed */
    /* BCLK = 40MHz*(I2S_BClockNi/I2S_BClockMi) = Sample Rate * 64BCLK/sample */
    if (voice_driver_codec_params.codec_sample_rate == SAMPLE_RATE_8KHz)
    {
        I2S_InitStruct.I2S_BClockMi         = 0x186A;
        I2S_InitStruct.I2S_BClockNi         = 0x50;
    }
    else if (voice_driver_codec_params.codec_sample_rate == SAMPLE_RATE_16KHz)
    {
        I2S_InitStruct.I2S_BClockMi         = 0x186A;
        I2S_InitStruct.I2S_BClockNi         = 0xA0;
    }
    else
    {
        APP_PRINT_WARN1("[voice_driver_init_i2s] invalid codec sample rate: %d",
                        voice_driver_codec_params.codec_sample_rate);
        I2S_InitStruct.I2S_BClockMi         = 0x186A;
        I2S_InitStruct.I2S_BClockNi         = 0xA0;
    }

    /* config I2S channel type */
    I2S_InitStruct.I2S_ChannelType      = I2S_Channel_Mono;

    I2S_Init(I2S0, &I2S_InitStruct);
    I2S_Cmd(I2S0, I2S_MODE_RX, ENABLE);
}
```

13.2.3 Voice CODEC Initialization

Bee3 RCU Codec configuration can be classified as below:

1. Codec basic parameters configuration

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

- CODEC_SampleRate: configure sampling rates, support 8/16KHz;
- CODEC_DmicClock: DMIC clock, support 312500Hz/ 625KHz/ 1250KHz/ 2500KHz/ 5MHz;
- CODEC_I2SFormat: I2S Format, support I2S/ LeftJustified/ PCM_A/ PCM_B;
- CODEC_I2SDataWidth: I2S data width, support 8Bits/ 16Bits/24Bits;
- CODEC_I2SChSequence: I2S Channel order, support CODEC_I2S_CH_L_R/ CODEC_I2S_CH_R_L/ CODEC_I2S_CH_L_L/ CODEC_I2S_CH_R_R;
- CODEC_MicBIAS: MIC BIAS output voltage, support 1.507/1.62/1.705/1.8/1.906/2.025/2.16/2.314V；
- CODEC_MicBstGain: AMIC BST Gain value, support 0/20/30/40dB;
- CODEC_MicBstMode: AMIC BST Mode, support MICBST_Mode_Single/ MICBST_Mode_Differential;

2. MIC channel 0 parameter configuration

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

   - CODEC_Ch0Mute: whether channel0 data Mute, support MUTE and UNMUTE;
   - CODEC_Ch0MicType: MIC type chosen, support AMIC and DMIC;
   - CODEC_Ch0DmicDataLatch: DMIC Data Latch type, support Rising_Latch and Falling_Latch;
   - CODEC_Ch0AdGain: Channel 0 digit volume gain , support range -17.625dB to30dB, accuracy 0.375dB；
   - CODEC_Ch0BoostGain: Channel 0 digit Boost gain, support 0/12/24/36dB;

   The specific initialization code is as below:

```
void voice_driver_init_codec(void)
{
    /* switch power mode */
    SystemCall(3, 1);

    CODEC_AnalogCircuitInit();

    RCC_PeriphClockCmd(APBPeriph_CODEC, APBPeriph_CODEC_CLOCK, ENABLE);

    CODEC_InitTypeDef CODEC_InitStruct;
```

```
    CODEC_StructInit(&CODEC_InitStruct);

    /* Basic parameters section */
    CODEC_InitStruct.CODEC_SampleRate = voice_driver_codec_params.codec_sample_rate;
    CODEC_InitStruct.CODEC_DmicClock = voice_driver_codec_params.dmic_clock;
    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_sequenc
e;
    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_da
ta_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_d
et_timeout;

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

The uninitialized code is as below:

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

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

13.2.4 Voice GDMA initialization

Voice GDMA mainly transmits data from I2S FIFO to RCU APP. Bee3 RCU implements the ping-pong Buffer through GDMA multi-block function.

The specific initialization code is as below:

```
1.  void voice_driver_init_rx_gdma(void)
2.  {
3.      /* Enable GDMA clock */
4.      RCC_PeriphClockCmd(APBPeriph_GDMA, APBPeriph_GDMA_CLOCK, ENABLE);
5.
6.      /* Initialize GDMA peripheral */
7.      GDMA_InitTypeDef GDMA_InitStruct;
8.      GDMA_StructInit(&GDMA_InitStruct);
9.      GDMA_InitStruct.GDMA_ChannelNum          = VOICE_GDMA_Channel_NUM;
10.     GDMA_InitStruct.GDMA_DIR                 = GDMA_DIR_PeripheralToMemory;
11.     GDMA_InitStruct.GDMA_BufferSize          = VOICE_GDMA_FRAME_SIZE / 4;
12.     GDMA_InitStruct.GDMA_DestinationInc      = DMA_DestinationInc_Inc;
13.     GDMA_InitStruct.GDMA_SourceInc           = DMA_SourceInc_Fix;
14.     GDMA_InitStruct.GDMA_SourceDataSize      = GDMA_DataSize_Word;
15.     GDMA_InitStruct.GDMA_DestinationDataSize = GDMA_DataSize_Byte;
16.     GDMA_InitStruct.GDMA_SourceMsize         = GDMA_Msize_4;
17.     GDMA_InitStruct.GDMA_DestinationMsize    = GDMA_Msize_16;
18.     GDMA_InitStruct.GDMA_SourceAddr          = (uint32_t)(&(I2S0->RX_DR));
19.     GDMA_InitStruct.GDMA_SourceHandshake     = GDMA_Handshake_I2S0_RX;
20.     GDMA_InitStruct.GDMA_DestinationAddr     = (uint32_t)voice_driver_global_data.gdma_buffer.b
    uf0;
21.     GDMA_InitStruct.GDMA_DestHandshake       = GDMA_Handshake_SPIC0_RX;
22.     GDMA_InitStruct.GDMA_Multi_Block_En      = 1;
23.     GDMA_InitStruct.GDMA_Multi_Block_Struct  = (uint32_t)voice_driver_gdma_link_list;
24.     GDMA_InitStruct.GDMA_Multi_Block_Mode    = LLI_TRANSFER;
25.
26.     /* Initialize GDMA Link List Struct */
27.     for (uint8_t i = 0; i < 2; i++)
28.     {
29.         if (i == 0)
30.         {
31.             voice_driver_gdma_link_list[i].DAR = (uint32_t)voice_driver_global_data.gdma_buffer.buf
    0;
32.             voice_driver_gdma_link_list[i].LLP = (uint32_t)&voice_driver_gdma_link_list[i +
33.                                                                 1];  /* link to buffer 1 */
34.         }
35.         else
36.         {
37.             voice_driver_gdma_link_list[i].DAR = (uint32_t)voice_driver_global_data.gdma_buffer.buf
    1;
38.             voice_driver_gdma_link_list[i].LLP = (uint32_t)&voice_driver_gdma_link_list[i -
39.                                                                 1];  /* link back to buffer 0 */
40.         }
41.         voice_driver_gdma_link_list[i].SAR = GDMA_InitStruct.GDMA_SourceAddr;
```

```
42.
43.       /* Configure low 32 bit of CTL register */
44.       voice_driver_gdma_link_list[i].CTL_LOW = BIT(0)
45.                                   | (GDMA_InitStruct.GDMA_DestinationDataSize << 1)
46.                                   | (GDMA_InitStruct.GDMA_SourceDataSize << 4)
47.                                   | (GDMA_InitStruct.GDMA_DestinationInc << 7)
48.                                   | (GDMA_InitStruct.GDMA_SourceInc << 9)
49.                                   | (GDMA_InitStruct.GDMA_DestinationMsize << 11)
50.                                   | (GDMA_InitStruct.GDMA_SourceMsize << 14)
51.                                   | (GDMA_InitStruct.GDMA_DIR << 20)
52.                                   | (GDMA_InitStruct.GDMA_Multi_Block_Mode & LLP_SELECTED_BI
    T);
53.       /* Configure high 32 bit of CTL register */
54.       voice_driver_gdma_link_list[i].CTL_HIGH = GDMA_InitStruct.GDMA_BufferSize;
55.    }
56.
57.    GDMA_Init(VOICE_GDMA_Channel, &GDMA_InitStruct);
58.
59.    NVIC_InitTypeDef NVIC_InitStruct;
60.    NVIC_InitStruct.NVIC_IRQChannel = VOICE_GDMA_Channel_IRQn;
61.    NVIC_InitStruct.NVIC_IRQChannelPriority  = 3;
62.    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
63.    NVIC_Init(&NVIC_InitStruct);
64.
65.    GDMA_INTConfig(VOICE_GDMA_Channel_NUM, GDMA_INT_Block, ENABLE);
66.    GDMA_Cmd(VOICE_GDMA_Channel_NUM, ENABLE);
67. }
```

13.2.5 SW EQ initialization

Set the macro SUPPORT_SW_EQ in board.h of Bee3 SDK to turn off or turn on the software EQ function.
Bee3 supports 5-band EQ to do some special processing for the speech data. RTL8762E EQ Tool can help to adjust the related parameter.
The EQ Tool generation parameter can be filled with SW_EQ_PARAMS_TABLE[SW_EQ_MAX_STAGE][6] as the default parameter.
The specific initialization code is as below.

```
1.  void bq_init(void)
2.  {
3.      uint8_t i;
4.      for (i = 0; i < SW_EQ_MAX_STAGE && SW_EQ_PARAMS_TABLE[i][0] == EQ_CH_Cmd_ENA
    BLE; i++)
5.      {
```

```
        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;
        bq[i].b2 = (SW_EQ_PARAMS_TABLE[i][3] * SW_EQ_PARAMS_TABLE[i][1] + EQ_ROUND)
>> EQ_FACTOR;
        bq[i].a1 = SW_EQ_PARAMS_TABLE[i][4] ;
        bq[i].a2 = SW_EQ_PARAMS_TABLE[i][5] ;
        bq[i].prev_input_1 = 0;
        bq[i].prev_input_2 = 0;
        bq[i].prev_output_1 = 0;
        bq[i].prev_output_2 = 0;
    }
}
```

13.3 Voice Buffer Introduction

The Bee3 RCU voice module uses two kinds of buffer: GDMA ping-pong buffer and voice data loop buffer.

13.3.1 GDMA Ping-Pong Buffer

The GDMA ping-pong buffer is designed to let GDMA carry voice data from the I2S FIFO to the APP task. Ping-Pong buffer is a buffer of two GDMA buffer sizes, one of which is used as a GDMA transport destination cache, and the other is a data cache that GDMA interrupt to send message to APP task. Furthermore, the two buffer will be switched after GDMA block transferred.

13.3.2 Voice Data Loop Buffer

Bee3 RCU designs a Loop Buffer of voice data for voice delay and instantaneous RF interference to store the compressed voice data.

The voice buffer queue is a buffer area with a continuous memory address. When the buffer area is full, the data continues to be stored over the old data. The voice buffer queue is a 6000 Bytes memory block by default. The enqueue pointer is in_queue_index in figure 27 and the dequeue pointer is out_queue_index in figure 27. As for the following diagram, in_queue_index moves forward if enqueue, and out_queue_index moves forward if dequeue.

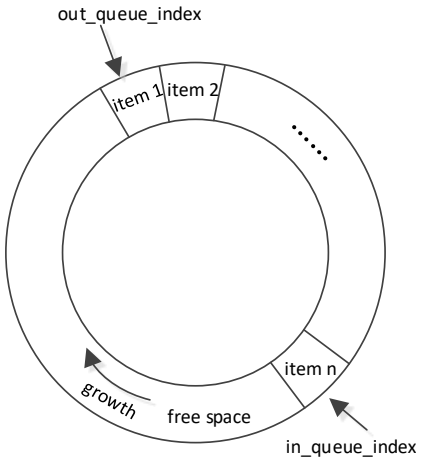

Figure 27 Voice Buffer Queue Diagram

13.4 Voice encoding algorithm

The program uses voice compression encoding to improve BLE transmission efficiency.

Bee3 RCU provides several methods of compression encoding: SW mSCB encoder, SW SBC encoder and SW IMA ADPCM encoder. Bee3 RCU can use the VOICE_ENC_TYPE macro defined in board.h to select the software encoding algorithm.

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

13.5 Voice Interaction Flow

Bee3 RCU supports four voice interaction flow:

- IFLYTEK_VOICE_FLOW
- HIDS_GOOGLE_VOICE_FLOW
- ATV_GOOGLE_VOICE_FLOW
- RTK_GATT_VOICE_FLOW

Bee3 RCU can use the VOICE_FLOW_SEL macro defined in board.h to select the voice interaction flow.

13.5.1 IFLYTEK voice flow

IFLYTEK voice interaction flow is based on HID Service, and the voice interaction flow between RCU and the peer device is as below.

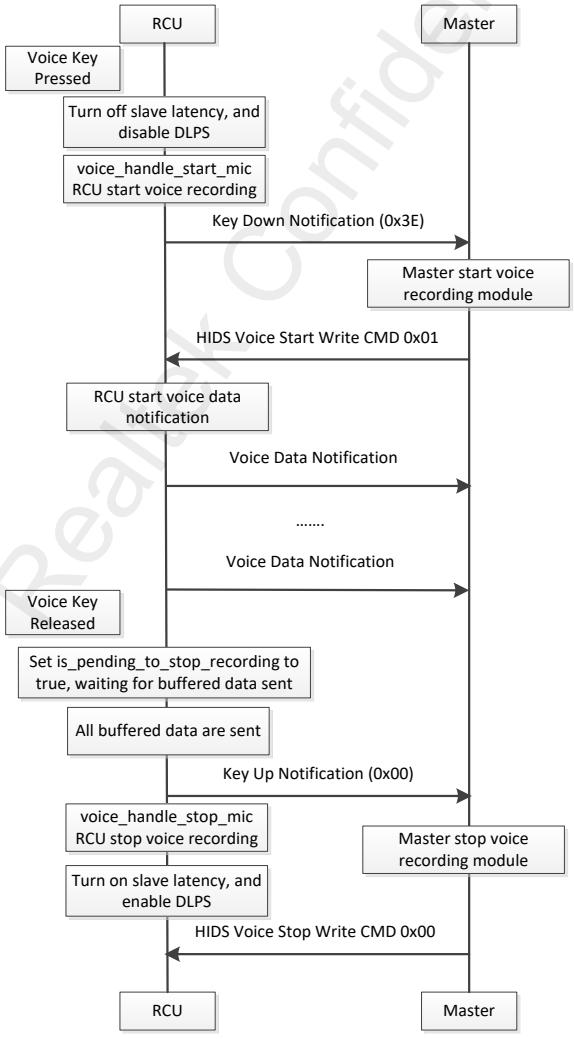

Figure 28 IFLYTEK voice flow

13.5.2 HIDS Google voice flow

The HIDS Google Voice interaction flow is similar to the IFLYTEK speech interaction flow. Both of them are based on HID Service. The difference is as below.

1. Voice Data Notification does not need to send after the host has received the ***voice start write*** cmd. Start to send Voice Data Notification directly after Voice Start Notification by default.
2. Two ways to end the voice.
    1) After the voice data transmission, the peer device voice recognition server will automatically determine whether the current voice flow is completed. If it is completed, then ***voice stop write*** command will be used to stop the RCU voice recording.
    2) If the user releases the voice key before receiving the voice stop write command from the peer device, stop the voice recording actively and send a Voice Stop Notification to notify the peer device to end the voice after waiting for the voice buffer data to be sent.

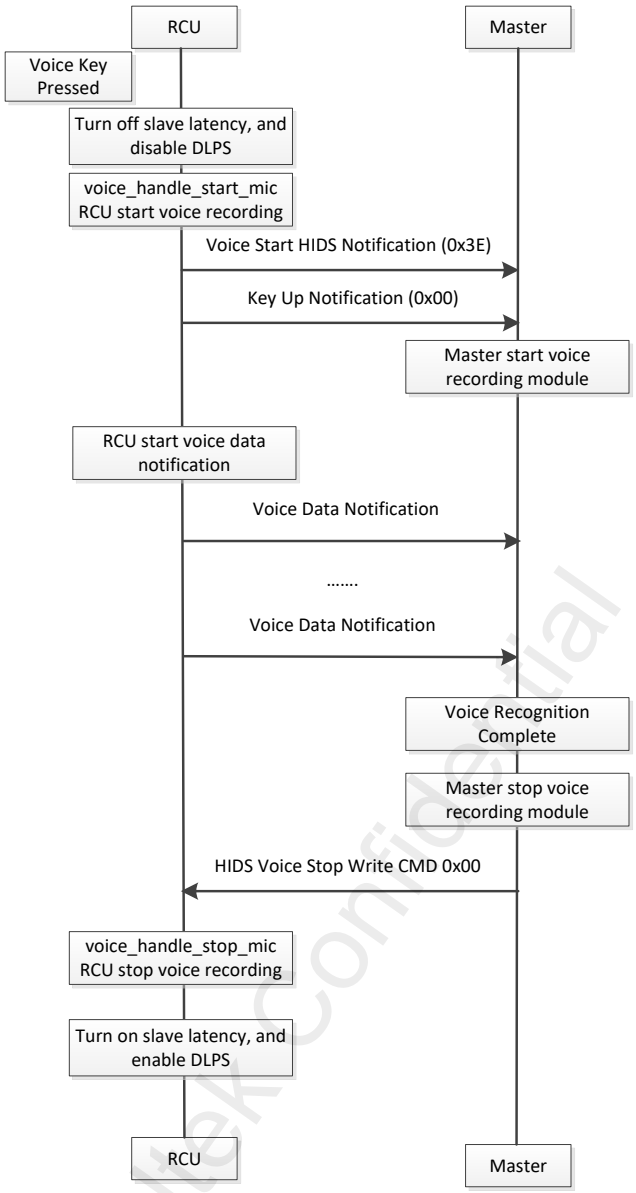

Figure 29 HIDS Google voice flow – 1

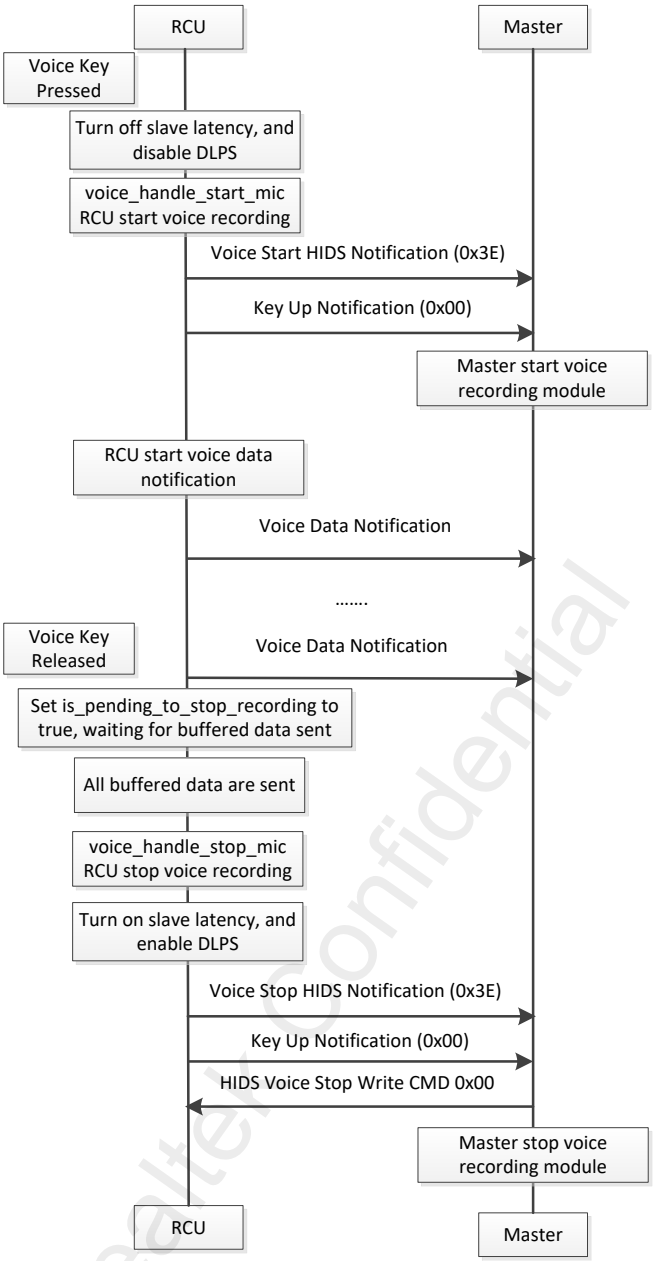

Figure 30 HIDS Google voice flow – 2

13.5.3 ATV Google VOICE FLOW

Starting from Android version 8.0, Google has defined a set of standard Voice Solution, which is based on Google's standard ATV Voice Service for voice transmission.

RCU uses the ATV Voice Service for voice data interaction. Refer to Google ATV Voice v0.4 documentation and Google Voice over BLE spec 1.0 documentation for details. The ATV Voice flow of Bee3 RCU basically follows the above spec. On this basis, the efficiency of speech transmission is optimized. RCU will actively initiate the request of the MTU Exchange after connecting and pairing. Then use the larger MTU size for voice interaction. When using ATV v0.4, the Notification size of the Voice Data is 134 bytes. When using ATV v1.0, the Notification size of the Voice Data is 128 bytes.

Bee3 RCU can use the ATV_VOICE_VERSION macro defined in board.h to select the voice version of ATV. ATV v1.0 is used by default.

```
/* 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 */
```

There are three Assistant interaction types included in ATV v1.0, which can be used selectively through the ATV_VOICE_INTERACTION_MODEL macro definition in board.h. ATV v0.4 only supports On Request Type. Details on Assistant interaction type can be found in the Google Voice over BLE Spec 1.0 documentation.

```
/* 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 */
```

When ATV v0.4 is used, the voice interaction flow between the RCU and the peer device as below.

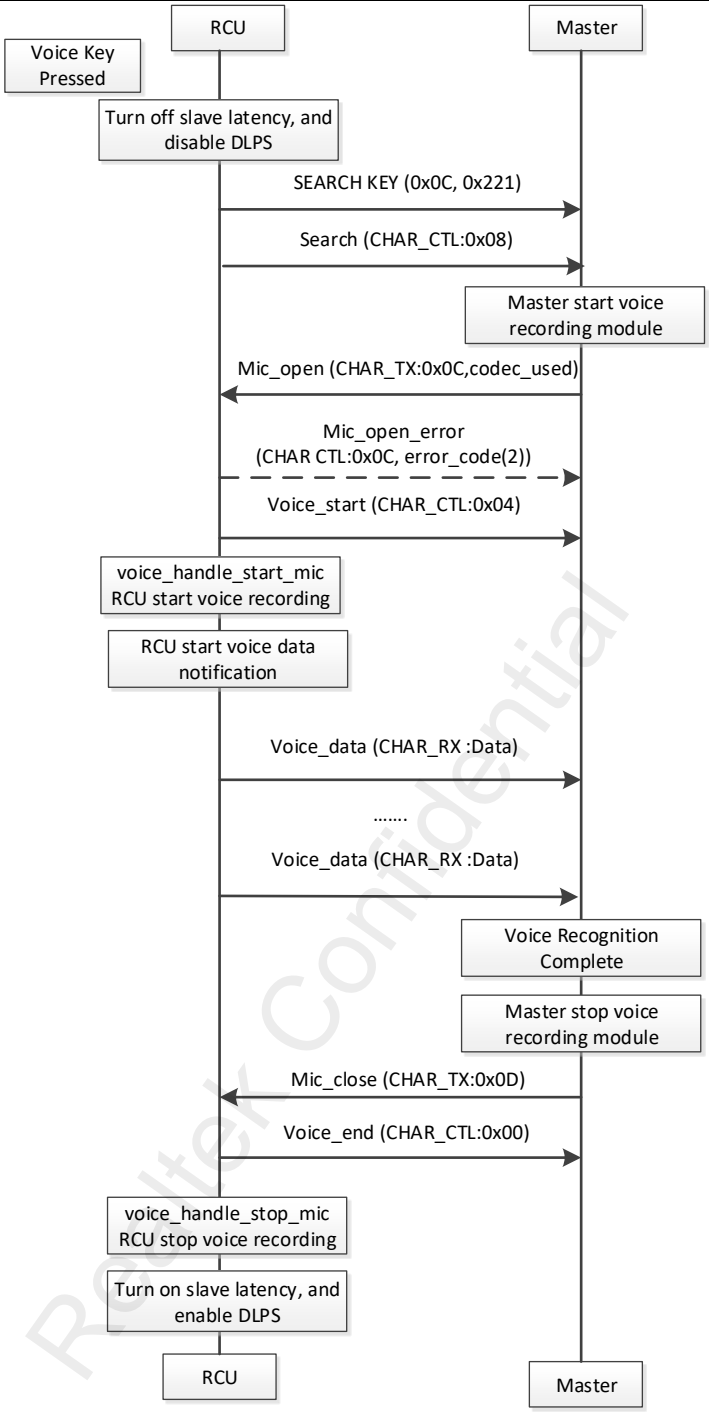

Figure 31 ATV v0.4 Google voice flow

ATV v1.0 contains three assistant interaction types. The voice interaction flow between the RCU and the peer device in different ways as below:

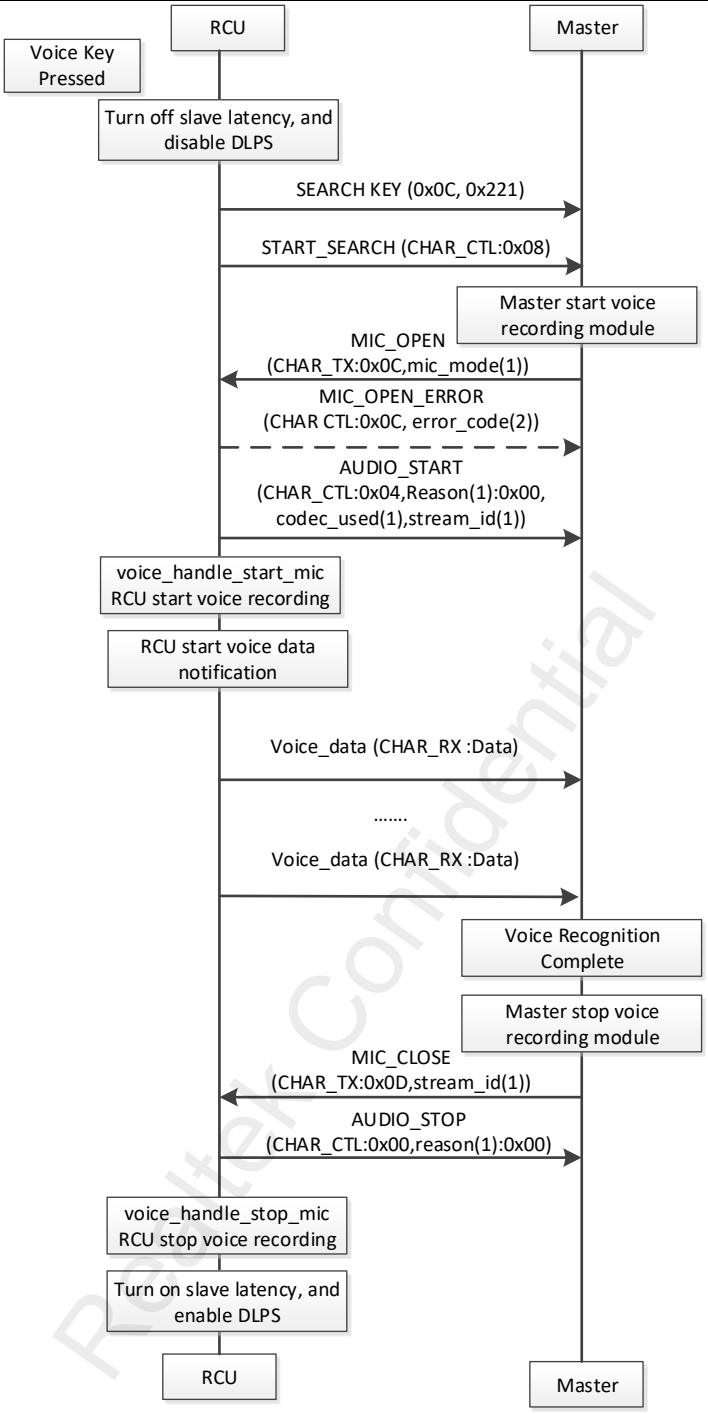

Figure 32 ATV v1.0 Google VOICE FLOW-ON_REQUEST_TYPE

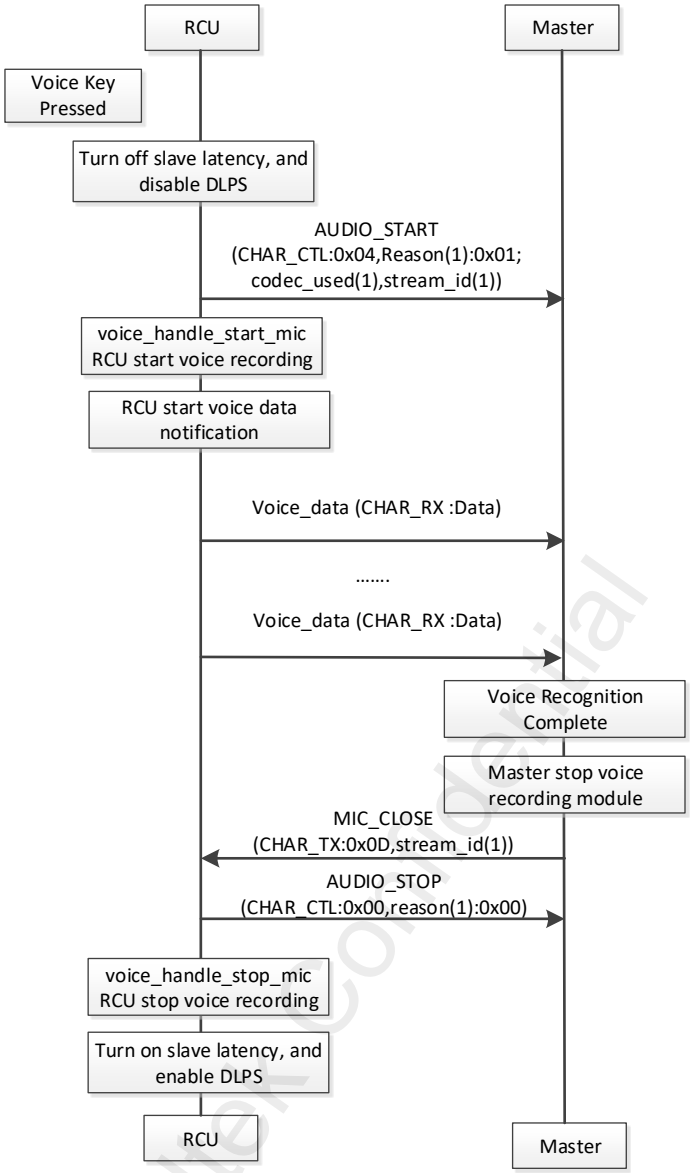

Figure 33 ATV v1.0 Google VOICE FLOW-PTT_TYPE

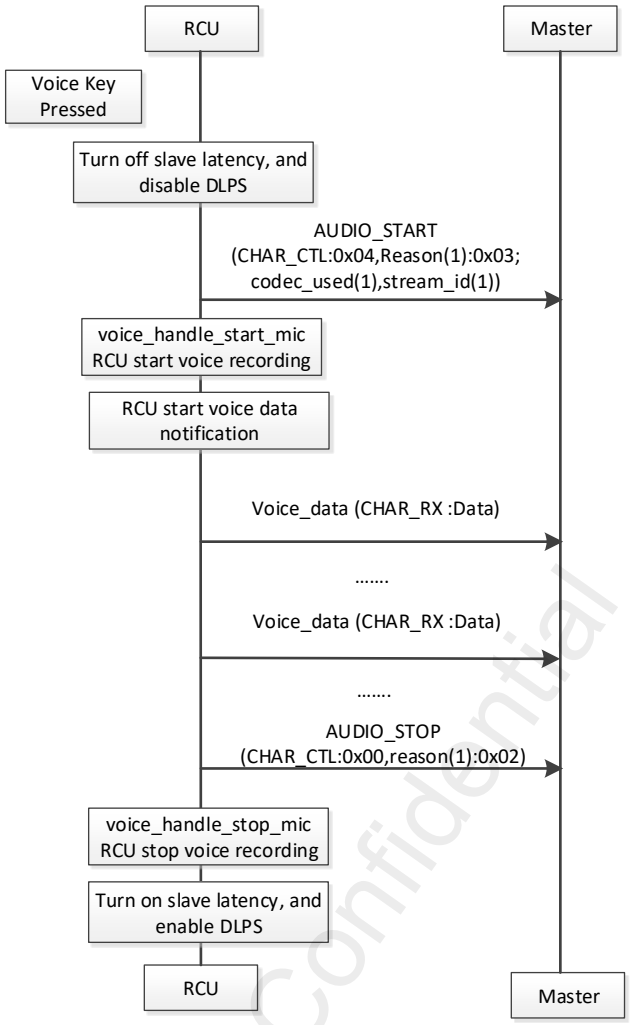

Figure 34 ATV v1.0 Google VOICE FLOW-HTT_TYPE

13.5.4 RTK GATT voice flow

RTK GATT VOICE FLOW is based on Realtek GATT vendor service, it can transmit voice data with Android device who install Realtek Voice APK.

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

Send the voice control command notification through voice command characteristic, and the characteristic UUID is 0x3A40;

Send voice data notification through voice data characteristic, and the characteristic UUID is 0x3A41;

The voice interaction process between RCU and the peer device is as follows:

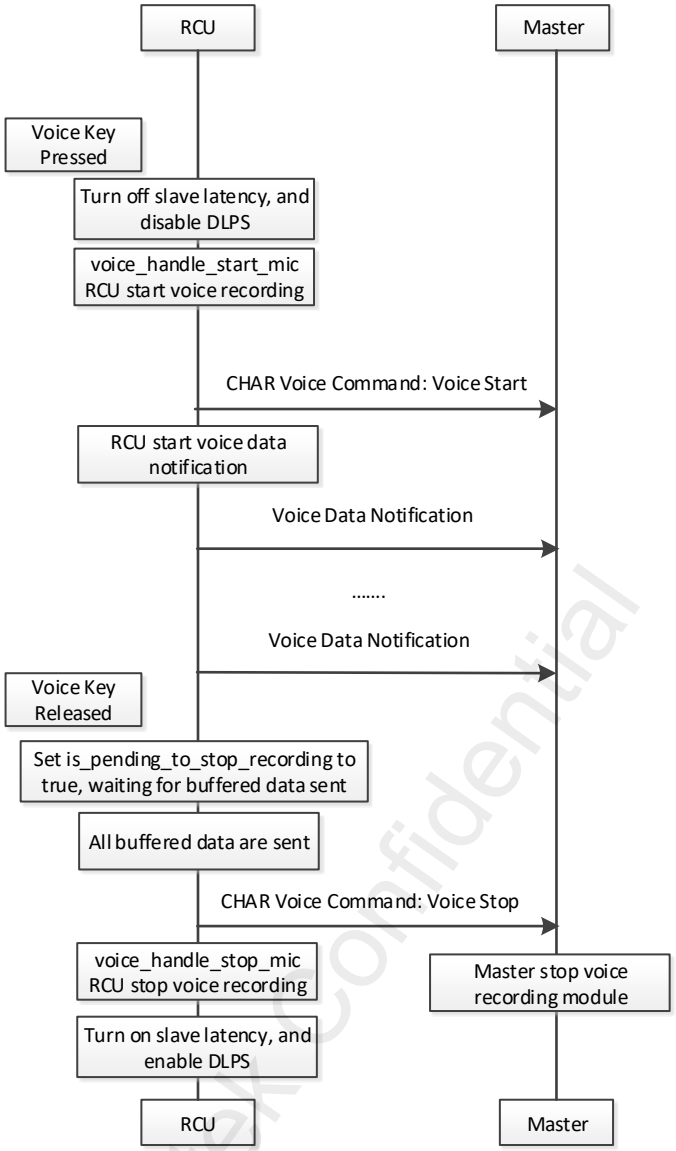

Figure 35 RTK GATT voice flow

14 IR

Bee3 RCU supports IR function, including IR transmission and IR learning. Bee3 IR carrier frequency supports 10kHz – 2MHz, which can support almost all IR protocols.

14.1 IR Transmission

Set the macro SUPPORT_IR_TX_FEATURE in board.h of Bee3 SDK to turn off or turn on the IR transmission function.

In the default state, IR key transmits value defined in KEY_CODE_TABLE. For the IR key after the IR learning is successful, learned IR waveform will be transmitted. IR key can restore to transmit default value after factory reset.

In Bee3 RCU SDK, NEC protocol is supported by default, relevant interfaces are also reserved in the program design to facilitate the extension of other IR protocols.

14.1.1 IR transmission initialization process

Bee3 RCU diagram of IR transmission initialization process is as flows

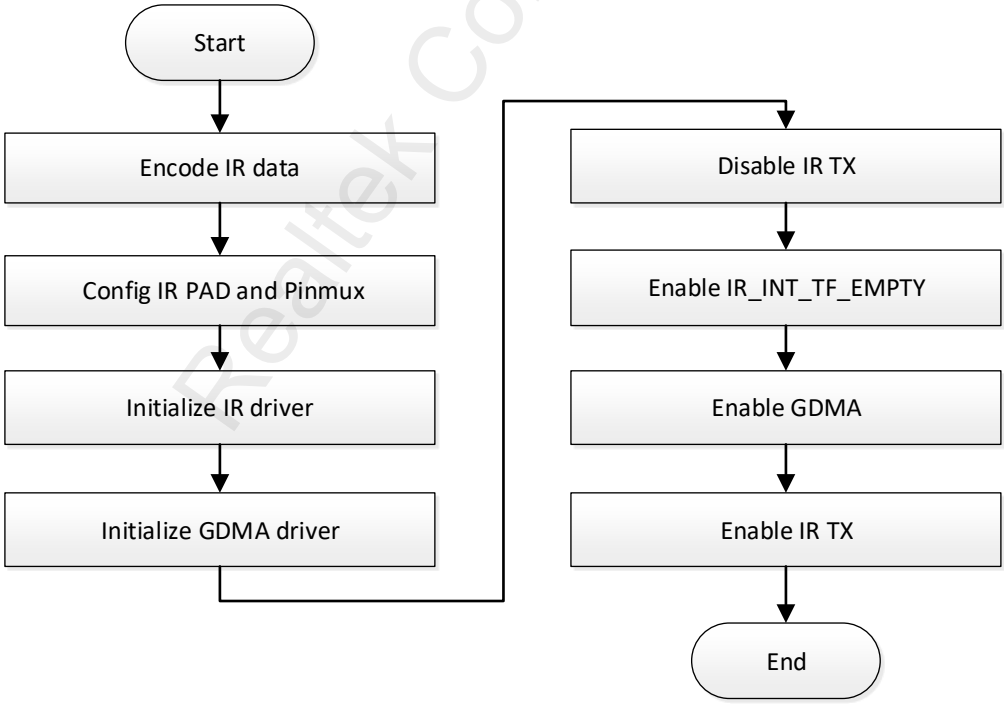

Figure 36 IR transmission initialization process

14.1.2 IR transmission GDMA function

Bee3 RCU support IR transmission with GDMA, GDMA can be used to carry IR DATA from memory to IR transmission module to facilitate the processing of software logic.

Specific initialization code is as follows:

```
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 IR repeat code

Bee3 RCU SDK realizes IR Repeat Code transmission function. Repeat Code uses RTC timer to transmit IR periodly. The initialization code of RTC module is as follows:

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

    /* Config RTC interrupt */
    NVIC_InitTypeDef NVIC_InitStruct;
```

```
    NVIC_InitStruct.NVIC_IRQChannel = RTC_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPriority = 3;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    RTC_SystemWakeupConfig(ENABLE);
     RTC_NvCmd(ENABLE);
    /* Start RTC */
    RTC_ResetCounter();
    RTC_Cmd(ENABLE);
}
```

14.2 IR learning

In board.h of Bee3 RCU SDK, macro SUPPORT_IR_LEARN_FEATURE can control the enabling of IR learning function.

Bee3 RCU SDK IR learning example is aimed at IR protocol with wavelength data below 70 bytes, and it can be adjusted by macro IR_LEARN_WAVE_MAX_SIZE. IR data is stored in FTL, the basic address of storage for IR waveform data can be adjusted by IR_WAVE_DATA_BASE_ADDR. However, that address cannot exceed the range of FTL district. When transmitting IR data, the specified IR data will be read from the stored address and transmit directly.

Bee3 RCU SDK limits the number of IR learning key by ***ir_learn_table***, only the keys in that table have the function of IR learning. Only 3 keys support IR learning function by default: ***VK_TV_POWER***, ***VK_TV_SIGNAL***, ***VK_POWER***. It can be modified according to the actual project.

```
#define  IR_LEARN_MAX_KEY_NUM      0x03  /* max key number can be storaged */

static T_IR_LEARN_KEY_INFO ir_learn_table[IR_LEARN_MAX_KEY_NUM] =
{
    {0, VK_TV_POWER},
    {1, VK_TV_SIGNAL},
    {2, VK_POWER},
};
```

14.3 IR learning operation flow

Bee3 RCU SDK provide a IR learning operation flow which is shown below:

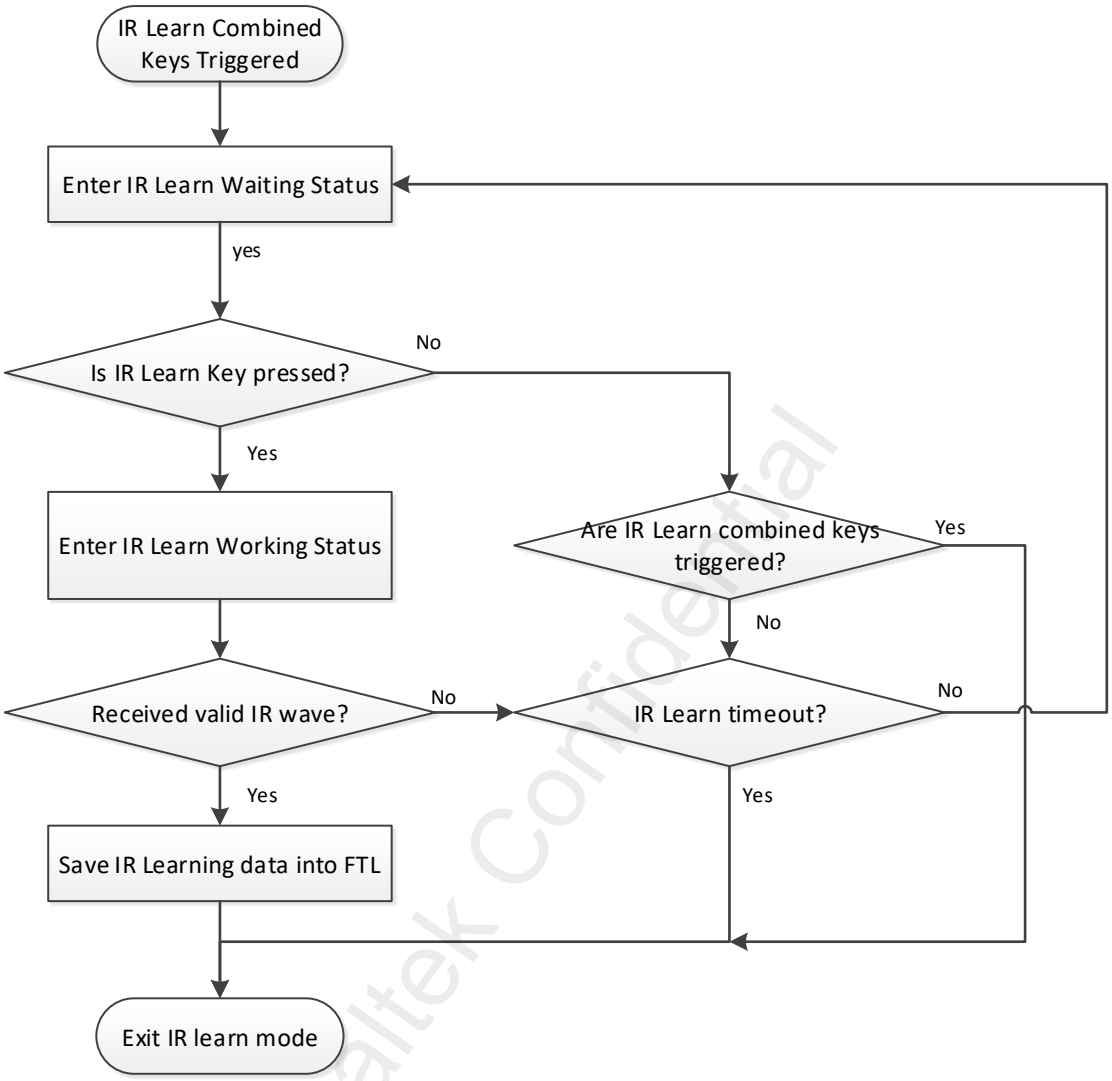

Figure 37 IR learning operation flow

1. Press combination keys VK_ENTER + VK_RIGHT (2s) to enter IR learning waiting mode;
2. Press and keep IR learning key to enter IR learning working mode;
3. In working mode, if RCU receive valid IR signal, the IR learning should be successful and IR waveform will be stored into FTL;
4. Return to waiting mode after IR learning finish;
5. If the IR learning button is released in the IR learning working mode, the IR learning fails this time and returns to the IR learning waiting mode;
6. Press combination keys VK_ENTER + VK_RIGHT(2s) or have no operation for a long time (***IR_LEARN_TIMEOUT*** 20s) to exit IR learning mode, when it is in IR learning mode;

15 OTA

RCU device supports Realtek vendor OTA protocol through OTA service and DFU service. The OTA on RCU device has two types, normal OTA and silent OTA.

Normal OTA needs RCU reboot to enter OTA mode, in which RCU focus on OTA flow and RCU keeps in unused state. After OTA updated successfully, new RCU image can work.

Silent OTA can be executed in RCU normal state. When OTA updated successfully, RCU device will reboot and run new image.

For OTA's detailed principles, please refer to the document of "***RTL8762E OTA User Manual***"

15.1 Normal OTA

In board.h in RCU SDK, macro SUPPORT_NORMAL_OTA can control the enabling of Normal OTA function.

Normal OTA is realized through OTA service. RCU app needs to reboot to enter OTA mode without normal application task. So the peer device needs to search for RCU device in OTA mode and establish connection, then executes OTA flow according to OTA service information. After new image updated successfully, RCU device reboots and executes new image code.

OTA service UUID is as follows.

```
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 includes the following 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 is used for OTA command, Write no response. Peer device writes 0x01 data into 0xFFD1, and then RCU device reboots and enters normal OTA mode.

0xFFD2 is used to read MAC address (BD Address) by peer device.

0xFFD3 is used to read Patch version by peer device.

0xFFD4 is used to read application version by peer device.

0xFFF1 is used to read device information by peer device, such as AES encryption and buffer check.

0xFFE0 is used to read device RCU image address and versions by peer device.

15.2 Silent OTA

In board.h in RCU SDK, macro SUPPORT_SILENT_OTA can control the enabling of Silent OTA function.

Silent OTA is a way to update when the remote controller is in normal state, and it requires to be supported in both OTA service and DFU service. Before updating, peer devices read RCU device information through OTA service; when updating, peer devices can control updating process and data transmission through DFU service. DFU service UUID is as follows:

```
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 has two characteristics, data characteristic and Control point characteristic. Data Characteristic, responsible for data transmission, is writable attributes. Control Point characteristic, responsible for OTA time series control, is writable and notification attributes.

The UUID of the two attributes is as follows.

```
#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 Battery Detection

RCU SDK support ADC battery voltage detect and low power protect function. Set macro SUPPORT_BAT_DETECT_ FEATURE to turn on/off battery voltage detect function.

16.1 Battery detection scheme

In RCU SDK, the following conditions will trigger the power detection operation.

16.1.1 power on battery detection

RCU SDK will call API ***bat_update_battery_info*** to update the battery information after the boot is completed.

16.1.2 BLE Battery Service read battery level

After connection completed, peer device read the battery information through the battery service. Then RCU updates battery information by calling API ***bat_update_battery_info*** and report percentage of battery level.

16.1.3 Key triggered battery detection

In board.h in RCU SDK, SUPPORT_BAT_KEY_PRESS_DETECT_FEATURE can be set to turn on /off Key triggered battery level detect function, and it is turned on by default.

When the number of keys operation reaches BAT_DETECT_TRIGGER_CNT, the battery detection will be triggered. BAT_DETECT_TRIGGER_CNT can be modified. The default value is 10, that is to say, every 10 key operation trigger a power detection.

16.1.4 Timer triggered battery level detection

In board.h in RCU SDK, SUPPORT_BAT_PERIODIC_DETECT_FEATURE can be set to turn on /off Timer triggered battery level detection function, and it is turned off by default.

When this function is turned on, RCU will check the battery level every BAT_PERIODIC_DETECT_TIMEOUT. BAT_PERIODIC_DETECT_TIMEOUT can be modified, default is 10 minutes

16.1.5 LPC triggered battery level detection

In board.h in RCU SDK, SUPPORT_BAT_LPC_FEATURE can be set to turn on /off LPC triggered battery level detection function.

When powered on, LPC is not initialized if RCU is in low power mode, and LPC is initialized if RCU is in normal mode.

When RCU battery level is lower than BAT_LPC_COMP_VALUE, LPC interrupt will be triggered immediately to restart RCU.

```
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
}
```

The LPC initialization process is as follows.

```
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);

    /* Config LPC interrupt */
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = LPCOMP_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPriority = 3;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
}
```

16.2 Low power mode

When the battery voltage is less than threshold BAT_ENTER_LOW_POWER_THRESHOLD, RCU enters low power mode.

Before entering the low power mode, if the system is in the BLE connection or advertising state, it is necessary to disconnect the system or stop advertising to return to the idle state, and then RCU enters the low power mode.

In low power mode, key pressing will trigger battery detection: If the voltage at this time is lower than the specified battery detection threshold, app will not respond to key operation; If the system battery voltage is higher than the specified battery voltage detection threshold BAT_ENTER_NORMAL_MODE_THRESHOLD at this time, app will exit the low power mode and perform the corresponding key operation.

16.3 Battery level detection threshold

The following thresholds are defined in the power detection part of Bee3 RCU SDK

BAT_ENTER_LOW_POWER_THRESHOLD: Threshold from normal mode to low power mode(2.0V);

BAT_ENTER_NORMAL_MODE_THRESHOLD: Threshold from low power mode to normal mode(2.2V);

BAT_ENTER_OTA_MODE_THRESHOLD: Minimum threshold allowed to enter OTA mode(2.5V);

BAT_LPC_COMP_VALUE: Threshold LPC interrupt triggered(1.84V);

17 LED

The SDK supports a multi-function LED module. The module supports simultaneous flashing of multiple LEDs, and supports multiple events of a single LED flashing by priority.

17.1 LED configuration

LED module is controlled by the macro SUPPORT_LED_INDICATION_FEATURE. The number of LED is controlled by LED_NUM_MAX. RCU SDK currently supports one by default, and the number can be extended.

```
/*******************************************************
*           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 interface

There are 4 LED interface APIs, which is defined in led_driver.h, including LED_ON, LED_OFF, LED_BLINK and 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 and LED_OFF are mainly used to indicate whether the state of the key is on or off.

LED_BLINK and LED_BLINK_EXIT are mainly used to control the flashing and exiting of LED. LED_BLINK has three parameters, the first is LED index, which is defined in board.h, the second is LED flashing type, which is defined in led_driver.h, the third on is the times of LED flashes. If 0 is filled in, flashing will be continuous. If the specified value is filled in, the number of flashes depends on the data filled in, and the maximum number of flashes is 255.

17.3 LED scenarios

17.3.1 LED flash time definition

There are three different flash time definition in Bee3 SDK: ***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

Figure 38 LED flash time definition

17.3.2 LED indication scenarios

Based on the above three flash time definition, the LED indication scenarios added in the RCU SDK are as follows:

1. Combination key restores factory mode indication: ***NORMAL_BLINK*** 3 times;
2. Enter low power mode indication: ***SHORT_BLINK*** 5 times;
3. Low power mode key indication: ***SHORT_BLINK*** once;
4. IR learning success indicator: ***SHORT_BLINK*** 5 times;
5. IR learning waiting mode indication: ***LONG_BLINK***;
6. IR learning working mode indication: LED is always on;
7. Pairing advertising mode indication: ***NORMAL_BLINK***;
8. Pairing success mode indication: ***SHORT_BLINK*** 5 times;
9. Normal mode key indication: press the key to keep the LED on, and key released the led off;

If a new LED indicator type needs to be added, the following should be done:

```
/**
 *  @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)
 *
```

```
27. *   3. Define led loop bits num, in file led_driver.h
28. *      //max value 32, min value 1, it indicate that there is how many bits to loop
29. *      //from LSB
30. *      #define LED_LOOP_BITS_OTA_UPDATE          (15)
31. *
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 MP mode

Bee3 RCU support MP mode, including HCI UART test mode, Data UART test mode, Single Tone test mode and fast-pair mode. Please refer to "***RTL8762E RCU MP Test Mode Design Spec***" for details of relevant MP test modes. In Bee3 RCU SDK, there are related macro definitions in board.h to support MP test mode.

```
/*******************************************************
*              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)

```

```
#if (RCU_HD_PLATFORM_SEL == R_DEMO_RCU)
#define MP_TEST_TRIG_PIN_1       P3_2
#define MP_TEST_TRIG_PIN_2       P3_3

#elif (RCU_HD_PLATFORM_SEL == H_DEMO_RCU)
#define MP_TEST_TRIG_PIN_1       P5_1
#define MP_TEST_TRIG_PIN_2       P5_2

#elif (RCU_HD_PLATFORM_SEL == G_MIN_DEMO_RCU)
#define MP_TEST_TRIG_PIN_1       P3_2
#define MP_TEST_TRIG_PIN_2       P3_3

#endif
#endif

#if MP_TEST_MODE_SUPPORT_DATA_UART_TEST
#define MP_TEST_UART_TX_PIN      P3_0
#define MP_TEST_UART_RX_PIN      P3_1
#endif

#endif
```

Bee3 switches from normal mode to test mode through test mode AON register and watchdog reboot. The process of switching MP test mode is as follows:

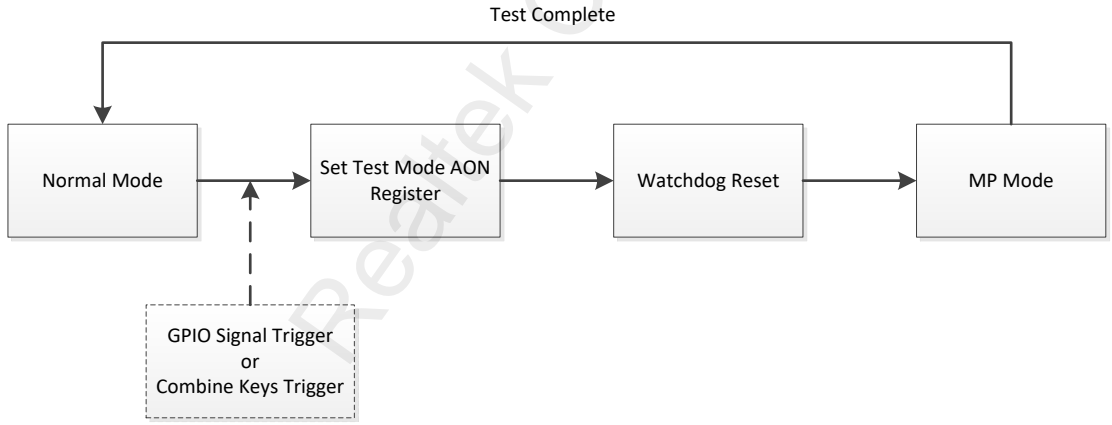

Figure 39 switching MP test mode

Bee3 RCU SDK provides sample of MP flow, including: Image Download, PCBA Level Test and Product Function Test. Please refer to document "***RTL8762E RCU MP Test Sample Flow***" for more information.

19 Reference

[1] RTL8762E OTA User Manual

[2] RTL8762E RCU MP Test Mode Design Spec

[3] RTL8762E RCU MP Test Sample Flow