HID_Remote Example Application Note

Revision History

Date	Version	Description
July 14, 2021	0.50	Initial version created.
September 30, 2021	0.51	Updated Figure 1 - Remote Example Hierarchy, Table 3 - IR and PV Build Options, 3 State Machines, 8 Voice Search, 9 LED
December 22, 2021	0.52	Updated Table 3 - IR and PV Build Options, 6.2.2 Connection parameters and negotiation, Table 28 - ATVV Configuration Options, Table 29 - ATVV APIs, Table 31 - Rc_atvv APIs and Callers, 8.2.1.2.2 Voice search (o.4e), added 8.2.1.2.3 Initialization (1.0), 8.2.1.2.4 Voice search (1.0)

CONFIDENTIAL

©2021, Atmosic Technologies Inc. All rights reserved. Atmosic logo is a registered trademark of Atmosic Technologies Inc. All other trademarks are the properties of their respective holders. This preliminary document is subject to change without notice.

Table of Contents

CONFIDENTIAL

CONFIDENTIAL

List of Figures

List of Tables

CONFIDENTIAL

CONFIDENTIAL

1 Overview

This application note describes the settings, functionality, and code flow of the (Human Interface Device) HID_remote example code running on an Atmosic ATM2/ATM3. The Atmosic HID_remote example is developed as a Bluetooth LE HOGP (HID over GATT profile) remote controller with voice recognition functionalities, which include Google Voice 0.4e/1.0 specification and voice over HID with ADPCM and other encoder, such as mSBC (Users should have their own license for mSBC)

1.1 Quick Start

- Install Atmosic SDK 4.1.0 or latest version
- Refer to section 5.1 Pin Setup
- Go to the platform/atmx[1]/ATMx[2]2xx-xx[3]x/example/HID_remote folder of the Atmosic Software Development Kit (SDK) and type “make clean” then “**make run_all <LAYOUT=ATMx221 or ATMx202>**” to program Flash (see section 5.2 Configure Flash Layout). Press and hold the <OK>+<-> for 5 seconds (see section 7 User Input Management), the LED will blink (see 9 LED) and start sending pairing advertisements with “Atmosic remote 100” as the device name.

```
@00019ef0 UPGD init debug: 0x800000f
@00019f68 [   atm_gap][W]: Unhandled GAPM msg 0xd1c
@0001a027 [    rc_gap][V]: GAP_S_IDLE (GAP_OP_INITED)
@0001a115 [    rc_mmi][V]: MMI_S_IDLE (MMI_OP_INIT_DONE)
@0001a1fe lock slots of (hibernation, retention, sleep): 0, 0, 0
@0001a2c6 [    rc_gap][N]: FW: 0.2.0.0(May  6 2021 16:37:29)
@0001a404 [   atm_adv][D]: Adv0: ON (0)
@0001a4a2 [    rc_gap][V]: GAP_S_ADV1ING (GAP_OP_ADV1ING)
@0001a58f [    rc_mmi][V]: MMI_S_PAIRING (MMI_OP_PAIRING)
@0001a67b lock slots of (hibernation, retention, sleep): 0x4, 0, 0
```

- In the Android TV, select “Settings->Connected Devices->Connect remote” and pair the “Atmosic remote 100”.
- After pairing successfully, the TV could be controlled by pressing the key matrix:

[1] atm2 or atm3
[2] ATM22xx or ATM32xx
[3] x0x or x1x

```
@000c1145 [   atm_gap][D]: + Peer (0) 6C:21:A2:DB:76:77
@000c1216 [   atm_gap][D]: + Connection interval +    40 (unit:1.25ms)
@000c130a [   atm_gap][D]: + Slave latency       +     0
@000c13d2 [   atm_gap][D]: + Supervision timeout +   200 (unit:  10ms)
@000c14ca [    rc_gap][V]: GAP_S_CONNECTED (GAP_OP_CONNECTED)
@000c15c4 [    rc_mmi][V]: MMI_S_PAIRING (MMI_OP_CONNECTED)
@000c16b5 lock slots of (hibernation, retention, sleep): 0x5, 0, 0
@000c1786 [   atm_gap][D]: ConnInd idx: 0 role: S
@000c1845 [   atm_adv][D]: Adv0: OFF (0)
@000c18e4 [    rc_gap][V]: GAP_S_CONNECTED (GAP_OP_ADV_STOP)
@000c19f5 ble_get_handler - Unhandled msgid 0xe01 from 0xff
@000c1af2 [  app_bass][V]: app_bass_send_lvl_cb: result 3.328V, Capacity 100.0% 100%
@000c1c48 otaps: idx:0 ENABLE: 0
@000c1cba ble_hid :BLE_HOGPD_ENABLED
@000c1d4e ble_atvv :ATVVS_ENALBED
@000c248f [    rc_gap][V]: rc_pair_req_ind
@000c2541 [    rc_gap][N]: Repairing (host delete bond)
@000ceb81 [    rc_mmi][V]: MMI_S_CONNECTED (MMI_OP_PAIR_SUCCESS)
@000cec82 lock slots of (hibernation, retention, sleep): 0x5, 0, 0
@000ced54 lock slots of (hibernation, retention, sleep): 0x4, 0, 0
@000cee30 ble_get_handler - Unhandled msgid 0xe15 from 0xff
@000ceee8 ble_get_handler - Unhandled msgid 0xe15 from 0xff
@000d5136 [    rc_mmi][V]: MMI_S_ATVV_ONLY (MMI_OP_ATVV_READY)
@000d5231 lock slots of (hibernation, retention, sleep): 0x4, 0, 0
@000d52ff ble_atvv :ATVVS_READY
@000d988d [   atm_gap][D]: + Peer (0) 6C:21:A2:DB:76:77
@000d9961 [   atm_gap][D]: + Connection interval +     8 (unit:1.25ms)
@000d9a53 [   atm_gap][D]: + Slave latency       +    99
@000d9b1d [   atm_gap][D]: + Supervision timeout +   300 (unit:  10ms)
@000fa04e ble_hid :BLE_HOGPD_READY
@000fa0dc [    rc_mmi][V]: MMI_S_HID_ATVV (MMI_OP_HID_READY)
```

- Search by pressing the MIC button, and speak into it to perform voice search.

```
@00718de9 [   atm_gap][V]: atm_gapc_cmp_evt_lower_sniff_latency_locally: status (0)
@0071a256 [    rc_mmi][V]: MMI_S_ATVVING (MMI_OP_OPEN_MIC)
@0071a345 lock slots of (hibernation, retention, sleep): 0x4, 0, 0
@0071a465 [   atm_gap][V]: atm_gapc_cmp_evt_lower_sniff_latency_locally: status (0)
@0071a589 ble_atvv :ATVVS_STREAMING
@0071a620 is8K = 1
@0071a660 PDM: start(85)
@0074010a [    rc_mmi][V]: MMI_S_HID_ATVV (MMI_OP_CLOSE_MIC)
@00740217 lock slots of (hibernation, retention, sleep): 0x4, 0, 0x1
@00740305 PDM: stop
@00740344 PDM ints 2306, data 2306, nobuf 0, overflow 0 0
@0074040a average sw time 631 us / 2 ms, 31%
@0074049c bytes: 151116, start 7448310, end 7602991, total 5 s
@0074058a 16bit with 16006 Hz
@00740619 ble_atvv :ATVVS_READY
```

1.2 Hierarchy and Files

The example was designed to allow users to easily adapt to their final products. Every module is designed to be a single function and interacts with a single module of the Atmosic framework. Figure 1 shows the hierarchy of modules in the remote example.

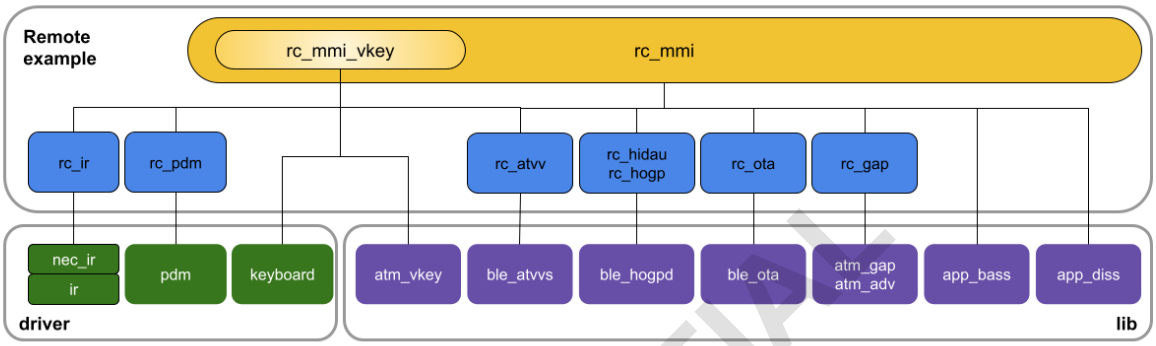

Figure 1 - Remote Example Hierarchy

Table 1 provides description of the modules' functionality.

Directory	File Name	Description
src/	rc_mmi.c rc_mmi.h	Interacting with all other modules to maintain the flow according to MMI states.
	rc_mmi_vkey.c rc_mmi_vkey.h rc_mmi_vkey_default_x202.h rc_mmi_vkey_default_x221.h	Handle virtual key behavior in different MMI states. Key code definitions of different products.
	app_config.h	Bluetooth parameters.
src/bt/	rc_gap.c rc_gap.h	Agent of GAP. It provides related operations for rc_mmi and maintains Bluetooth states and informs rc_mmi while changed.
	rc_atvv.c rc_atvv.h	Agent of ATVV. It provides voice search functionalities for rc_mmi and rc_mmi_vkey.
	rc_hogp.c rc_hogp.h	Agent of HOGP. It provides HID (Human Interface Device) functionality to rc_mmi and rc_mmi_vkey to send reports to TV.
	rc_hidau.c rc_hidau.h	Interact with rc_pdm and rc_hogp to handle Voice over HID.

CONFIDENTIAL

	rc_ota.c rc_ota.h	Specific Over-the-Air (OTA) parameters to ble_ota.
src/ non_bt/	rc_pdm.c rc_pdm.h	Provide an interface for rc_mmi and rc_atvv to control Pulse Density Modulation (PDM) start, stop, pause and resume. It also acts as a collaborator of audio flow. It passes raw Pulse Code Modulation (PCM) to Adaptive differential pulse-code modulation (ADPCM) encoder then sends it to rc_atvv while streaming is on.
	rc_ir.c rc_ir.h	Provide an interface for rc_mmi_vkey to send Infrared (IR) codes.

Table 1 - Modules' Functionality

CONFIDENTIAL

2 Building Application

In this application, two series of chips, variable features and configurations are supported. Parameter configurations are defined in `app_config.h`.

2.1 Devices

Two series of Atmosic devices are supported by this application. The devices are selected through a LAYOUT variable from the command line. Table 2 describes build options of the devices.

Device	Description	Build Option
ATMx221	• 6x6 mm package • External Flash	LAYOUT=ATMx221
ATMx202	• 5x5 mm package • Internal Flash	LAYOUT=ATMx202

Table 2 - Device Build Options

If LAYOUT variable is not added in command line, ATMx221 option will be applied.

2.2 Features

In this application, additional features of IR and Photovoltaics (PV) cell energy harvesting are provided. Table 3 describes the build options:

Feature		Description	Build Option
IR	Disconnected Only	NEC IR transmission only on advertisement.	CFG_RC_IR=1
	All time	NEC IR transmission at any time.	CFG_RC_IR=1 CFG_RC_IR_ON_HID=1
PV		Power harvesting from PV Cell. This feature is only allowed with ATM3xxx.	PV_HARV_EN=1
Voice	ATVV 0.4e	Voice search complies with Google Voice over Bluetooth LE 0.4e.	*(default)
	ATVV 1.0	Voice search complies with Google Voice over Bluetooth LE 1.0. Three assistant models are introduced in this edition. Users can specify which models are supported by the remote application. 0x0: Legacy (On-request, ***default**) 0x1: Legacy and PTT (press-to-talk) 0x3: Legacy, PTT and HTT (hold-to-talk) Others: invalid	CFG_ATVV_VER_100=1 CFG_ATVV_ASST_MODEL =< 0 or 1 or 3 >
	ADPCM over HID	Voice encoded with ADPCM encoder is sent through HID.	CFG_VOICE=HID_ADPCM
	mSBC over HID	Voice encoded with mSBC encoder is sent through HID.	CFG_VOICE=HID_MSBC

Table 3 - IR and PV Build Options

2.3 Build Combinations

By assigning build options of features (IR, PV and Voice) and devices (ATMx221 and ATMx202), the user can create his own combination.

For example, an ATM3202 device with PV harvesting and Google Voice over Bluetooth LE 1.0 and without IR function, the build command would be "**make LAYOUT=ATMx202 PV_HARV_EN=1 CFG_ATVV_VER_100=1**".

CONFIDENTIAL

3 State Machines

State machines are the core of the application. Users can easily change and optimize design by understanding the state machines. Refer to 3.1 MMI State Descriptions and 3.2 GAP State Descriptions sections. State machines are created by defining state transition table mechanisms provided by the atm_asm module.

3.1 MMI State Descriptions

In this application, 12 Man Machine Interface (MMI) states and 17 operations are defined in rc_mmi. Table 4 describes those states.

MMI State Name	Description
MMI_S_BOOTED	The initial state. Device booted or woke up from hibernation.
MMI_S_INITING	Under initialization.
MMI_S_IDLE	The end state (Disconnected then enter hibernate) or intermediate state (Initialized then pairing or reconnecting).
MMI_S_PAIRING	Bonding with new TV.
MMI_S_RECONNING	Reconnecting the bonded TV.
MMI_S_CONNECTED	Bluetooth connected but profiles are not ready.
MMI_S_DISCONNING	Disconnection is ongoing due to timer timeout events.
MMI_S_RF_TEST	Under RF testing
States exist only in Google Voice over Bluetooth LE	
MMI_S_HID_ONLY	HOGP is ready but ATVV is not.
MMI_S_ATVV_ONLY	ATVV is ready but HOGP is not.
MMI_S_HID_ATVV	Both ATVV and HOGP are ready.
MMI_S_ATVVING	Voice streaming is ongoing.
States exist only in Voice over HID	
MMI_S_HID_READY	HOGP is ready
MMI_S_HID_STREAMING	Voice streaming is ongoing.

Table 4 - rc_mmi States

CONFIDENTIAL

Operations are used to control the state machine transition. Each operation is triggered by rc_xxx modules. Table 5 describes the operations and the triggering module.

Operations	Description	Triggered by
MMI_OP_INITING	Initialization is starting.	rc_mmi
MMI_OP_INIT_DONE	Initialization is finished.	rc_gap
MMI_OP_RECONNING	Device is starting advertising for reconnection.	rc_gap
MMI_OP_PAIRING	Device is starting advertising for pairing.	rc_gap
MMI_OP_PAIR_SUCCESS	Device is paired successfully.	rc_gap
MMI_OP_CONNECTED	Device is connected.	rc_gap
MMI_OP_HID_READY	HID is ready.	rc_hogp
MMI_OP_HID_UNREADY	HID becomes unready.	rc_hogp
MMI_OP_OPEN_MIC	ATVV streaming started.	rc_atvv/rc_hidau
MMI_OP_CLOSE_MIC	ATVV streaming stopped.	rc_atvv/rc_hidau
MMI_OP_DISCONNING	MMI wants to disconnect the link.	rc_mmi
MMI_OP_ADV_STOPPED	Advertisement is stopped normally.	rc_gap
MMI_OP_RECONN_FAIL	Reconnection failed.	rc_gap
MMI_OP_RECONN_TOUT	Reconnection timeout	rc_gap
MMI_OP_PAIR_FAIL	Pairing failed.	rc_gap
MMI_OP_PAIR_TOUT	Pairing timeout	rc_gap
MMI_OP_DISCONNED	Link is disconnected	rc_gap
States exist only in Google Voice over Bluetooth LE		
MMI_OP_ATVV_READY	ATVV is ready.	rc_atvv
MMI_OP_ATVV_UNREADY	ATVV becomes unready.	rc_atvv

Table 5 - rc_mmi Operations

CONFIDENTIAL

By triggering those operations, states will transition. Figure 2 and Figure 3 show the ordinary transition paths of rc_mmi states. Figure 4 and Figure 5 show the transition paths of rc_mmi states in disconnection or error cases.

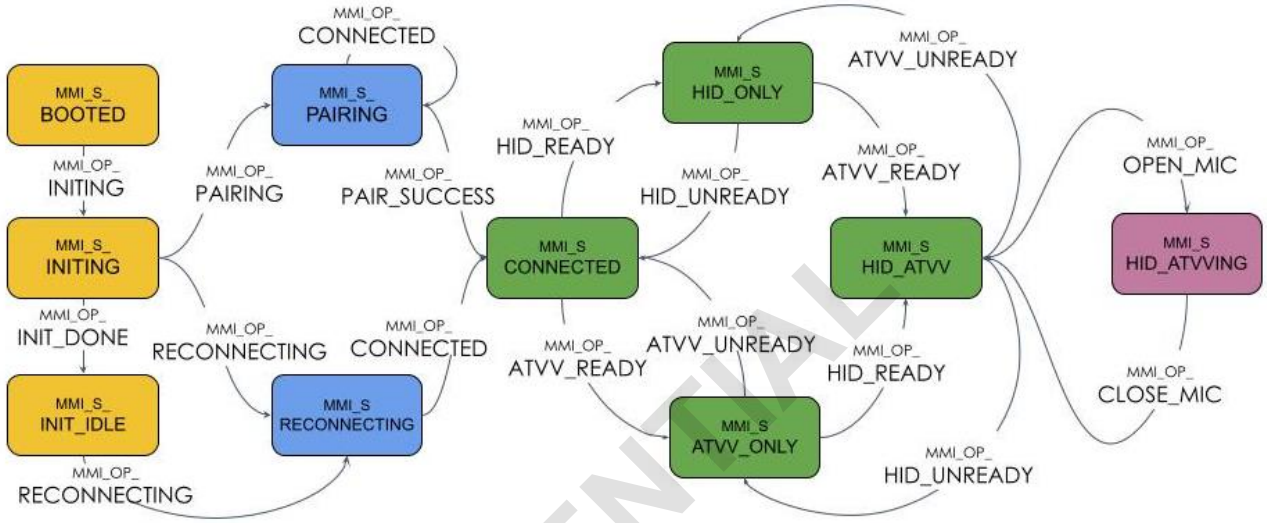

Figure 2 - rc_mmi State Transitions in ATVV Device (Ordinary Cases)

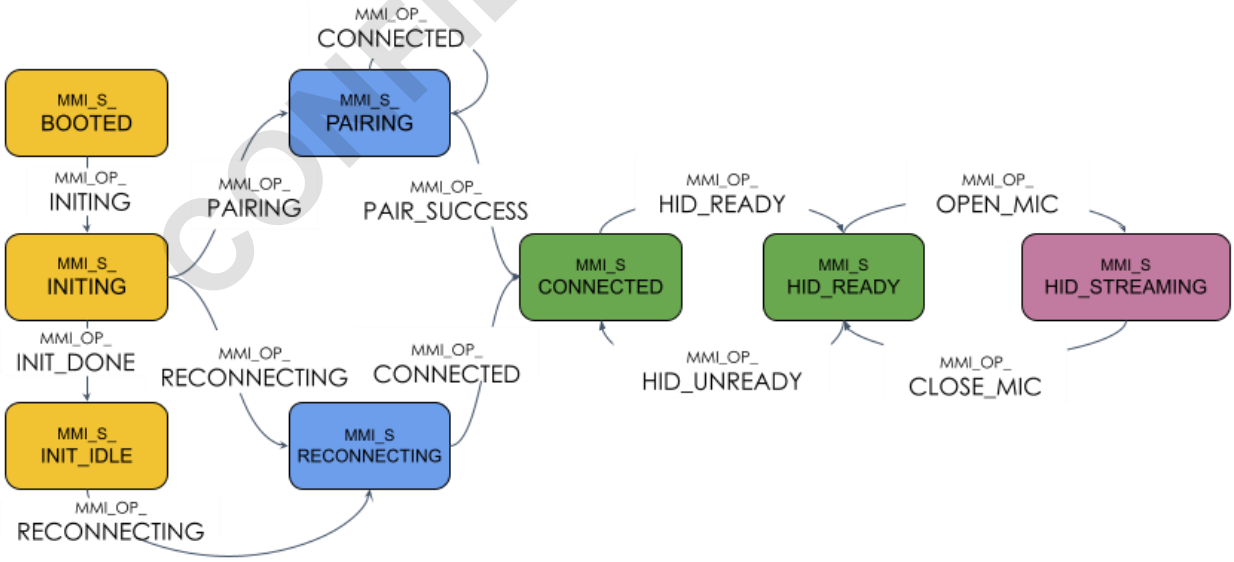

Figure 3 - rc_mmi State Transitions in VoHID Device (Ordinary Cases)

CONFIDENTIAL

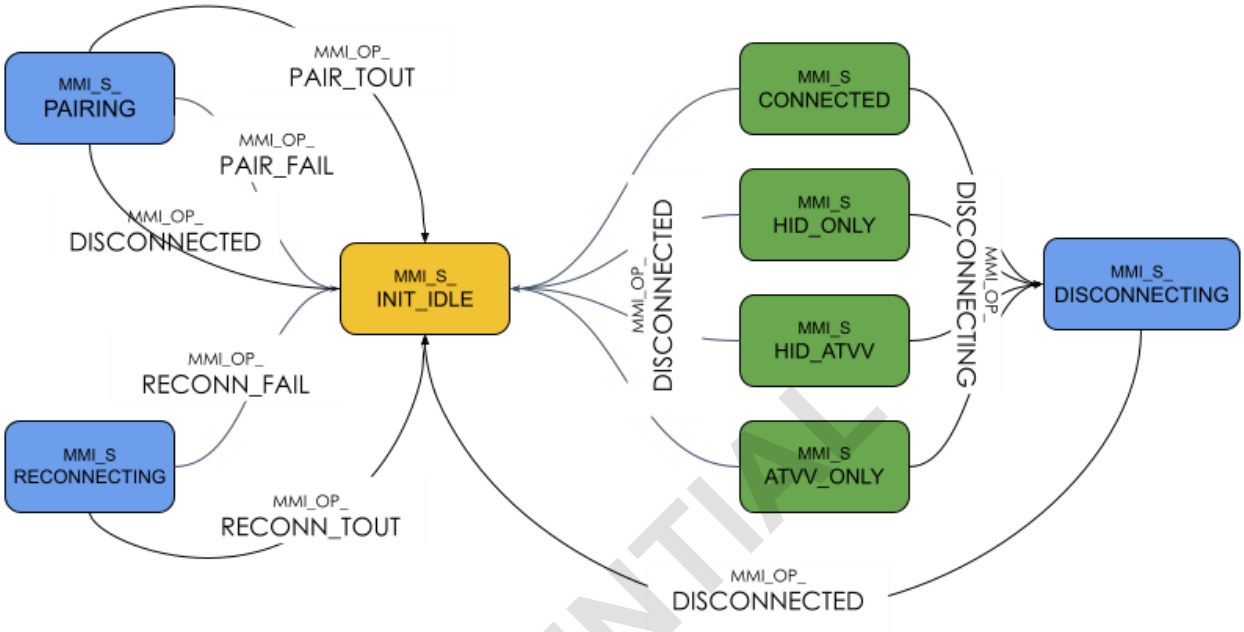

Figure 4 - rc_mmi State Transitions in ATVV device (Disconnection or Error Cases)

CONFIDENTIAL

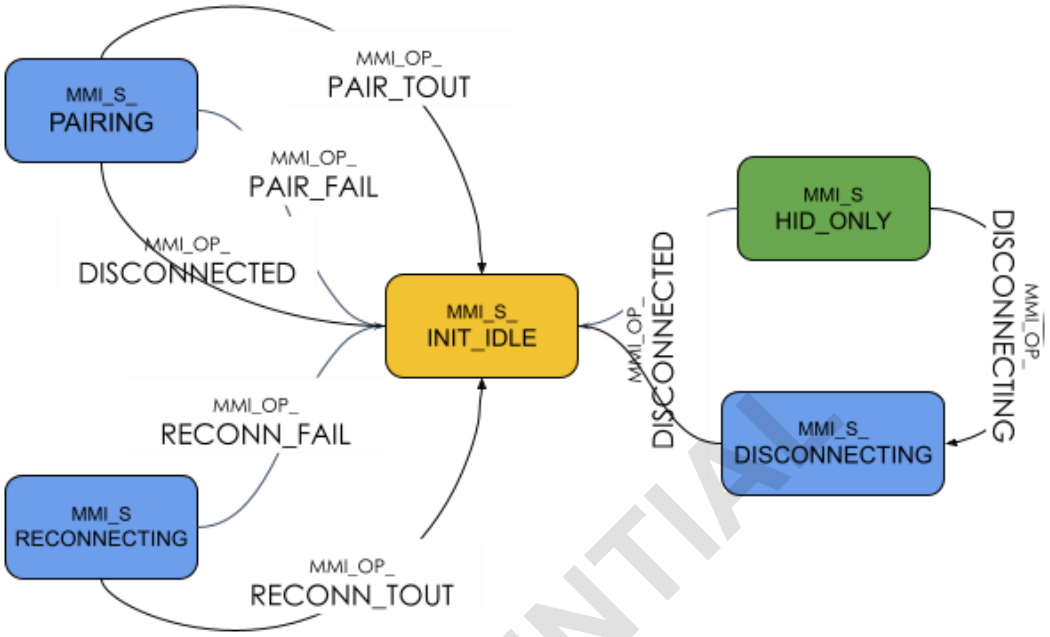

Figure 5 - rc_mmi State Transitions in VoHID Device (Disconnection or Error Cases)

Figure 6 shows rc_mmi state transition table mmi_s_tbl[]. Each entry of this table consists of two states (A and B), an operation (O) and a function (F), that means state A will change to state B when receiving operation O and function F will be executed. For example, the MMI_S_PAIRING state receiving `MMI_OP_PAIR_SUCCESS` operation will move to `MMI_S_CONNECTED` and call `mmi_s_pairing_stopped()` to stop LED blinking. Please refer to the source code for more information.

```
static state_entry const mmi_s_tbl[] = {
   {MMI_S_BOOTED, MMI_OP_INITING, MMI_S_INITING, mmi_s_booted_op_initing},
   {MMI_S_IDLE, MMI_OP_RECONNING, MMI_S_RECONNING, NULL},
   {MMI_S_IDLE, MMI_OP_ATVV_UNREADY, MMI_S_IDLE, NULL},
   {MMI_S_IDLE, MMI_OP_PAIRING, MMI_S_PAIRING, mmi_s_paring_started},
   {MMI_S_IDLE, MMI_OP_DISCONNING, MMI_S_IDLE,  NULL},
   {MMI_S_INITING, MMI_OP_INIT_DONE, MMI_S_IDLE, mmi_s_initing_op_done,},
   {MMI_S_INITING, MMI_OP_RECONNING, MMI_S_RECONNING, NULL},
   {MMI_S_INITING, MMI_OP_PAIRING, MMI_S_PAIRING, mmi_s_paring_started},
   {MMI_S_RECONNING, MMI_OP_CONNECTED, MMI_S_CONNECTED, mmi_s_connected},
   {MMI_S_RECONNING, MMI_OP_RECONN_TOUT, MMI_S_IDLE, NULL},
   {MMI_S_RECONNING, MMI_OP_ADV_STOPPED, MMI_S_IDLE, NULL},
   {MMI_S_PAIRING, MMI_OP_CONNECTED, MMI_S_PAIRING, mmi_s_connected},
   {MMI_S_PAIRING, MMI_OP_DISCONNED, MMI_S_IDLE, mmi_s_paring_stopped},
   {MMI_S_PAIRING, MMI_OP_PAIR_FAIL, MMI_S_IDLE, mmi_s_paring_stopped},
   {MMI_S_PAIRING, MMI_OP_PAIR_TOUT, MMI_S_IDLE, mmi_s_paring_stopped},
   {MMI_S_PAIRING, MMI_OP_PAIR_SUCCESS, MMI_S_CONNECTED, mmi_s_paring_stopped},
   {MMI_S_PAIRING, MMI_OP_ADV_STOPPED, MMI_S_IDLE, mmi_s_paring_stopped},
   {MMI_S_CONNECTED, MMI_OP_ATVV_READY, MMI_S_ATVV_ONLY, NULL},
```

CONFIDENTIAL

```
    {MMI_S_CONNECTED, MMI_OP_HID_READY, MMI_S_HID_ONLY, mmi_s_hid_ready},
    {MMI_S_CONNECTED, MMI_OP_DISCONNED, MMI_S_IDLE, NULL},
    {MMI_S_CONNECTED, MMI_OP_ATVV_UNREADY, MMI_S_CONNECTED, NULL},
    {MMI_S_CONNECTED, MMI_OP_PAIR_SUCCESS, MMI_S_CONNECTED, NULL},
    {MMI_S_CONNECTED, MMI_OP_PAIR_FAIL, MMI_S_CONNECTED, NULL},
    {MMI_S_HID_ONLY, MMI_OP_ATVV_READY, MMI_S_HID_ATVV, mmi_s_hid_ready},
    {MMI_S_HID_ONLY, MMI_OP_ATVV_UNREADY, MMI_S_HID_ONLY, NULL},
    {MMI_S_ATVV_ONLY, MMI_OP_ATVV_UNREADY, MMI_S_CONNECTED, NULL},
    {MMI_S_ATVV_ONLY, MMI_OP_HID_READY, MMI_S_HID_ATVV, mmi_s_hid_ready},
    {MMI_S_HID_ATVV, MMI_OP_ATVV_UNREADY, MMI_S_HID_ONLY, NULL},
    {MMI_S_HID_ATVV, MMI_OP_OPEN_MIC, MMI_S_ATVVING, mmi_s_open_mic},
    {MMI_S_HID_ATVV, MMI_OP_DISCONNED, MMI_S_IDLE, mmi_s_disconnected},
    {MMI_S_ATVV_ONLY, MMI_OP_DISCONNING, MMI_S_DISCONNING, mmi_s_disconnecting},
    {MMI_S_HID_ATVV, MMI_OP_DISCONNING, MMI_S_DISCONNING, mmi_s_disconnecting},
    {MMI_S_HID_ONLY, MMI_OP_DISCONNING, MMI_S_DISCONNING,  mmi_s_disconnecting},
    {MMI_S_HID_ONLY, MMI_OP_DISCONNED, MMI_S_IDLE,  mmi_s_disconnected},
    {MMI_S_ATVVING, MMI_OP_CLOSE_MIC, MMI_S_HID_ATVV, mmi_s_close_mic},
    {MMI_S_DISCONNING, MMI_OP_DISCONNED, MMI_S_IDLE, NULL},
    {MMI_S_RF_TEST, MMI_OP_DISCONNED, MMI_S_RF_TEST, NULL},
};
```

Figure 6 - rc_mmi Transition Array

3.2 GAP State Descriptions

As rc_mmi, the state machine of rc_gap is created in the application layer to simplify the design. There are 7 states and 9 operations defined in rc_gap state machines. Table 6 describes those states.

States	Description
GAP_S_INIT	Initial state.
GAP_S_IDLE	No ongoing Bluetooth activity. Enter hibernate if no further request from rc_mmi.
GAP_S_ADV0ING	Reconnecting advertisement is ongoing.
GAP_S_ADV1ING	Pairing advertisement is ongoing.
GAP_S_ADV_STOPPING	Advertisement is stopping.
GAP_S_CONNECTED	Bluetooth link exists.
GAP_S_RFTEST	RF test mode is ongoing.

Table 6 - rc_gap States

Each operation is triggered by atm_gap modules. Table 7 describes the operations and the triggering function.

CONFIDENTIAL

Operations	Description	Trigger by
GAP_OP_INITING	Bluetooth is initialing.	rc_gap_init()
GAP_OP_INITED	Bluetooth initialized.	rc_gap_init_cfm()
GAP_OP_ADV0ING	ADV 0 is started	rc_gap_adv_start_cfm()
GAP_OP_ADV1ING	ADV 1 is started	rc_gap_adv_start_cfm()
GAP_OP_ADV_STOP	ADV stopped due to connection.	rc_gap_adv_stop_ind()
GAP_OP_ADV_STOP_TOUT	ADV stopped due to timeout	rc_gap_adv_stop_ind()
GAP_OP_ADV_STOPPING	ADV is stopping by application.	rc_gap_discoverable(false)
GAP_OP_CONNECTED	Bluetooth is connected.	rc_conn_ind()
GAP_OP_DISCONNED	Bluetooth is disconnected	rc_disc_ind()

Table 7 - rc_gap Operations

Figure 7 shows the transition paths of rc_gap states.

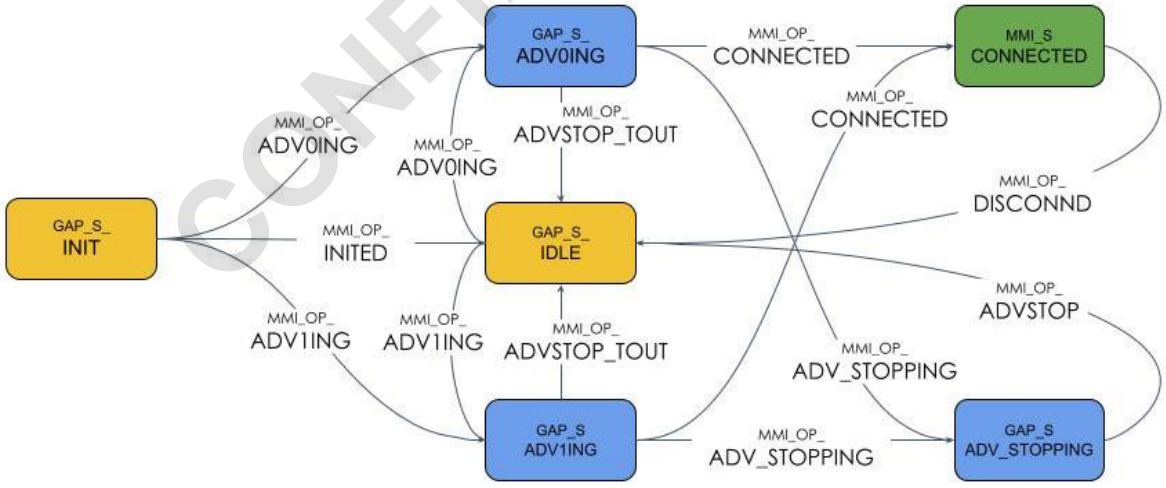

Figure 7 - rc_gap State Transition

4 Power Management

In this example code, the atm_pm module's lock scheme is used to manage whether or not to prevent entering a power state. Three kinds of locks from high to low power are

CONFIDENTIAL

PM_LOCK_SLEEP, PM_LOCK_RETENTION and PM_LOCK_HIBERNATION which are associated with sleep, retention and hibernation respectively. Once all the locks in one level are unlocked and no higher level lock is locked, the system will enter that power state. Table 8 lists all the locks provided in this example and descriptions:

PM_LOCK_SLEEP		
Name	**Modules**	**Description**
ksm_lock_sleep	driver/keyboard	Mostly stay unlocked. Only be locked for key interrupt processing.
pdm_lock_sleep	driver/PDM	Locked/unlocked depends on when PDM recording is started/stopped.
PM_LOCK_HIBERNATION		
Name	**Modules**	**Description**
vk_lock_hiber	driver/atm_vkey	Locked when virtual keys are not all released.
led_lock_hiber	driver/led_blink	Locked when the LED is blinking.
rc_gap_lock_hib	rc_gap	Locked when rc_gap state is not GAP_S_IDLE.

Table 8 - Power Management Locks and Description

CONFIDENTIAL

5 Hardware Setup

The makefile in this application includes two kinds of predefined settings for IR, PDM, keypad and Flash layout, which is decided by value of the LAYOUT variable. The value of LAYOUT would be ATMx221 or ATMx202. If not set, it will be ATMx221 by default.

5.1 Pin Setup

Table 9 lists the pin settings in makefile according to LAYOUT value of ATMx221 and ATMx202:

ATMx221			
Module	**Setting in Makefile**	**Pin**	**Description**
keyboard	NUM_ROW=4 NUM_COL=4	row: 0 1 2 3	Use default 4 by 4 key matrix settings.

<table>
<tr><td></td><td></td><td><table><tr><td>pin</td><td>P23</td><td>P22</td><td>P21</td><td>P20</td></tr><tr><td>ksi</td><td>0</td><td>1</td><td>2</td><td>3</td></tr></table><table><tr><td>col</td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>pin</td><td>P31</td><td>P30</td><td>P29</td><td>P28</td></tr><tr><td>kso</td><td>0</td><td>1</td><td>2</td><td>3</td></tr></table></td><td>Please refer to keyboard_param.h for more details.</td></tr>
<tr><td>led_blink</td><td>CFG_LED_GPIO=28</td><td><table><tr><td>led io</td><td>P32 (GPIO 28)</td></tr></table></td><td>LED indicator. Please refer to led_blink.c for more details.</td></tr>
<tr><td>pdm</td><td>*(default)</td><td><table><tr><td>pdm clock</td><td>P24</td></tr><tr><td>Pdm data</td><td>P25</td></tr></table></td><td>Please refer to pdm.c for more details.</td></tr>
<tr><td>rc_pdm</td><td>*(default)</td><td><table><tr><td>pdm io</td><td>P26 (GPIO 22)</td></tr></table></td><td>PDM device power control. Please refer to rc_pdm.c for more detail.</td></tr>
<tr><td>ir</td><td>IR_IO=13</td><td><table><tr><td>ir io</td><td>P13 (GPIO13)</td></tr></table></td><td>Please refer to ir.c for more details.</td></tr>
<tr><th colspan="4">ATMx202</th></tr>
<tr><th>Module</th><th>Setting in Makefile</th><th>Pin</th><th>Description</th></tr>
<tr><td>keyboard</td><td>ROW0=23
ROW0_KSI=0
ROW1=22
ROW1_KSI=1
ROW2=20
ROW2_KSI=3
ROW3=13
ROW3_KSI=6
COL0=30
COL0_KSO=1
COL1=11
COL1_KSO=8
COL2=10</td><td><table><tr><td>row</td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>pin</td><td>P23</td><td>P22</td><td>P21</td><td>P13</td></tr><tr><td>ksi</td><td>0</td><td>1</td><td>2</td><td>6</td></tr></table><table><tr><td>col</td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>pin</td><td>P30</td><td>P11</td><td>P10</td><td>P9</td></tr><tr><td>kso</td><td>1</td><td>8</td><td>9</td><td>10</td></tr></table></td><td>Customized key matrix assignment for 5x5 limited pin count. Please refer to keyboard_param.h for more details.</td></tr>
</table>

CONFIDENTIAL

	COL2_KSO=9 COL3=9 COL3_KSO=10		
led_blink	CFG_LED_GPIO=29	led io: P33 (GPIO 29)	LED indicator. Please refer to led_blink.c for more details. Note: this is shared with the debug log (UART1 TX).
pdm	*(default)	pdm clock: P24 pdm data: P25	Please refer to pdm.c for more details.
rc_pdm	PDM_POWER_SWITCH=28	pdm power switch: P32 (GPIO 28)	PDM device power control. Please refer to rc_pdm.c for more detail.
ir	IR_IO=20	ir_io: P24 (GPIO 20)	Please refer to ir.c for more detail. Note: this pin is shared with pdm_clock

Table 9 - Make File Pin Setting Based on LAYOUT Value

5.2 Configure Flash Layout

OTA feature is enabled by default. The Flash essentially utilizes half of its size. The makefile uses 256 KB (FLASH_SIZE=0x40000) and 512 KB (FLASH_SIZE=0x80000) for ATMx221 and ATMx202 respectively by default. The NVDS_SIZE default value is 0x8000.
Figure 8 and Figure 9 illustrate the default Flash layout of ATMx221 and ATMx201.

CONFIDENTIAL

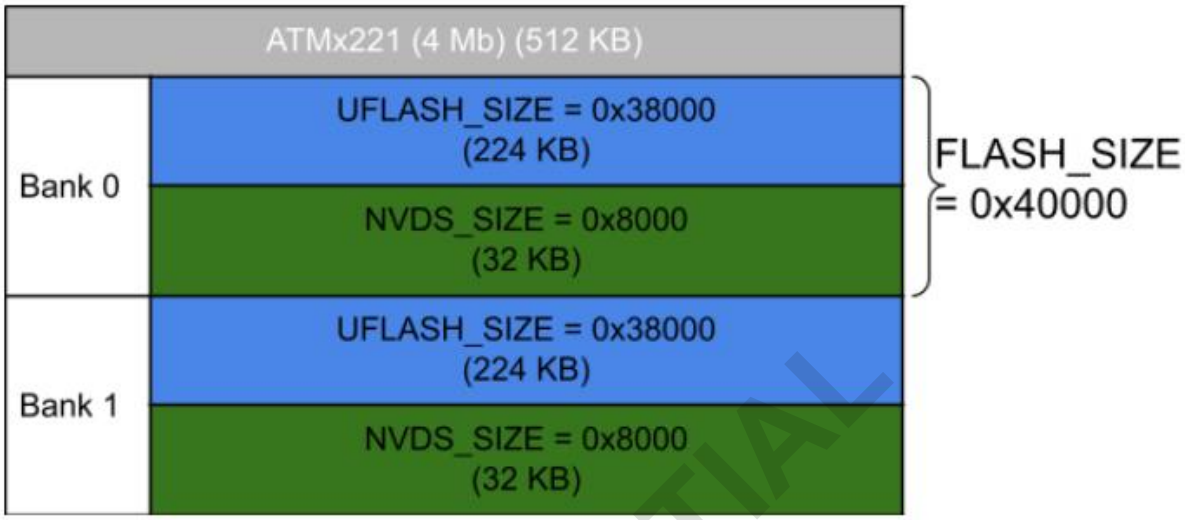

Figure 8 - Default Flash Layout of ATMx221

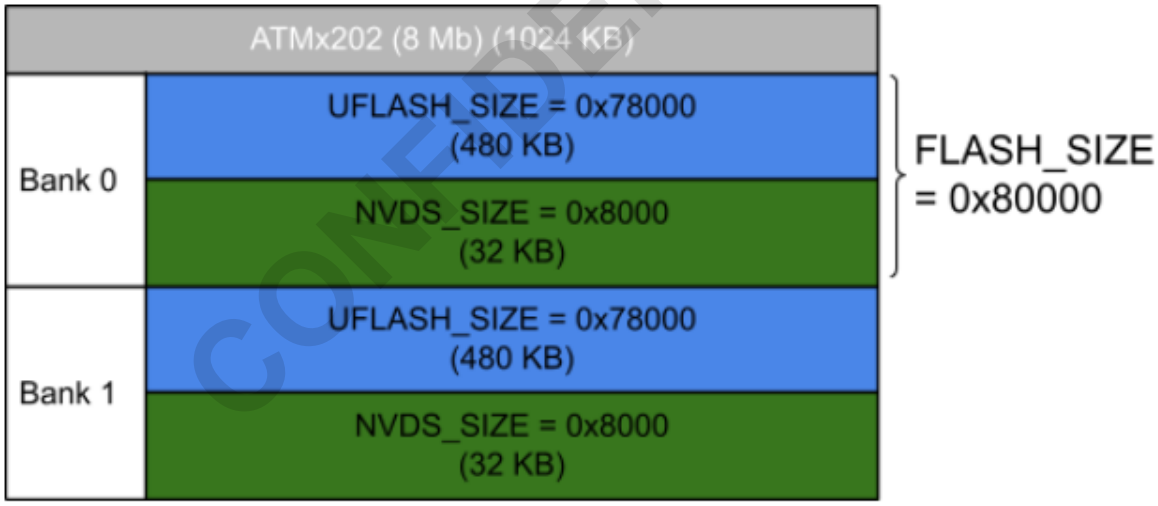

Figure 9 - Default Flash Layout of ATMx202

6 Bluetooth Parameters

All the related Bluetooth parameters are defined in src/app_config.h.

CONFIDENTIAL

6.1 Timeout Parameters

6.1.1 Timeout after HID ready

A timer with this timeout value (RC_CONN_READY_TOUT_CS) will be set when the link is connected and HID is ready. When this timeout occurs, the system will disconnect the link and enter hibernate. The timer will be reset on any of the user input. If this value is zero, the link will always be maintained. Default value is 0.

```
#ifndef RC_CONN_READY_TOUT_CS
#define RC_CONN_READY_TOUT_CS 0
#endi
```

6.1.2 Timeout after connected

A timer with this timeout value (RC_CONN_NOT_READY_TOUT_CS) will be set when the link is connected but HID is not ready. When this timeout occurs, the system will disconnect the link and enter hibernate. This timer will be clear after HID is ready. Default value is 1000 (10 seconds).

```
#ifndef RC_CONN_NOT_READY_TOUT_CS
#define RC_CONN_NOT_READY_TOUT_CS 1000
#endi
```

6.1.3 Timeout to poll battery capacity

There are two values associated with battery capacity polling timer. When battery capacity is low, the timer will use RC_LOW_BATT_POLL_TIME_CS as its timeout value. Otherwise it will use RC_HIGH_BATT_POLL_TIME_CS. Default values are 10000 (100 seconds) and 60000 (600 seconds) respectively.

```
#define RC_LOW_BATT_POLL_TIME_CS 10000
#define RC_HIGH_BATT_POLL_TIME_CS 6000
```

6.2 GAP Parameters

6.2.1 Advertisement

This example code uses two advertisement sets, (#define CFG_GAP_ADV_MAX_INST 2). Please refer to CFG_ADV0_* in app_config.h for reconnecting advertisements and CFG_ADV1_* for pairing advertisements.

6.2.2 Connection parameters and negotiation

Four parameters are related to the connection parameters: CFG_GAP_CONN_INT_MIN, CFG_GAP_CONN_INT_MAX, CFG_GAP_CONN_TIMEOUT, and CFG_GAP_SLAVE_LATENCY

```
// Minimal connection interval
#define CFG_GAP_CONN_INT_MIN 8
```

CONFIDENTIAL

```
// Maximal connection interval
#define CFG_GAP_CONN_INT_MAX 8
// Slave latency
#define CFG_GAP_SLAVE_LATENCY 99
// Connection timeout
#define CFG_GAP_CONN_TIMEOUT 300
```

These four values would reflect the value of peripheral preferred parameters characteristic in GAP service. Essentially, central will update connection parameters by referring to this characteristic. Except for the updating from central, peripheral could request itself.Depending on the CFG_SLAVE_PARAM_NEGO compile option, the device will perform connection parameter update negotiation after connecting with central. In `rc_gap.c - rc_gap_connect_param_nego()`, `param_nego` is the parameter for connection parameter negotiation. Users can modify the parameter depending on the application.

```
void rc_gap_nego_parameter(void)
{
#ifdef CFG_RC_SLAVE_PARAM_NEGO
   if (cur_lidx != GAP_INVALID_CONIDX) {
       static struct gapc_conn_param const conn_param = {
           CFG_GAP_CONN_INT_MIN,
           CFG_GAP_CONN_INT_MAX,
           CFG_GAP_SLAVE_LATENCY,
           CFG_GAP_CONN_TIMEOUT,
       };
       static atm_gap_param_nego_t const param_nego = {
           .param_nego_cfm = rc_gap_param_nego_cfm,
           .force_retry = false,
           .retry_times = RC_PARAM_NEGO_TIMES,
           .check_result = RC_PARAM_NEGO_TOUT_CS,
           .target = &conn_param,
       };
       atm_gap_connect_param_nego(cur_lidx, &param_nego);
   }
#endif
}
```

CONFIDENTIAL

6.3 Device Information Service

The strings of device information service such as manufacture name, model name, firmware revision, software revision, etc,.. are defined with `APP_DIS_*`. They can be modified by users if needed.

```
#define APP_DIS_MANUFACTURER_NAME "Atmosic Tech."
#define APP_DIS_MODEL_NB_STR "ATV-RC-01"
#define APP_DIS_SERIAL_NB_STR "1.0.0.0"
#define RC_VERSION "0.2.0.0"
#define APP_DIS_FIRM_REV_STR RC_VERSION
#define APP_DIS_SW_REV_STR RC_VERSION
#define APP_DIS_HARD_REV_STR "1.0.0"
#define APP_DIS_SYSTEM_ID "\x12\x34\x56\xFF\xFE\x9A\xBC\xDE"
#define APP_DIS_IEEE "\xFF\xEE\xDD\xCC\xBB\xAA"
#define APP_DIS_PNP_ID "\x01\x45\x75\x21\x00\x10\x01"
```

CONFIDENTIAL

7 User Input Management

The remote example uses three modules to complete the user input via key scan matrix: keyboard, atm_vkey and rc_mmi_vkey. Keyboard is a key scan matrix hardware driver. Figure 8 shows the hierarchy.

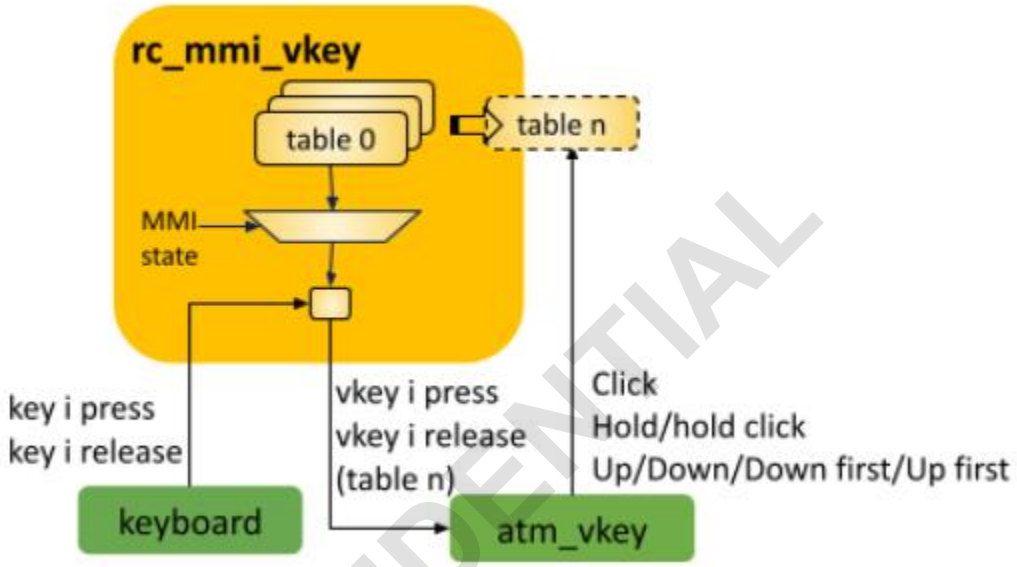

Figure 10 - Key Scan Matrix Software Hierarchy

7.1 Keyboard

The keyboard is a key scan matrix hardware driver. There are two ways to configure the pins for key matrix in keyboard driver:

1. Use default order: define NUM_ROWS and NUM_COLS in makefile
2. Assign customized order: define ROWx/ROWx_KSI and COLy/COLy_KSO in makefile

Please refer to the pin_setting in keyboard driver of Atmosic SDK API reference for more information. After the upper layer registers callback, it will generate keyboard events with a key index when keys are pressed or released. Table 10 shows the keyboard events.

Keyboard events	Description
KSM_RELEASE	Key index n was pressed. n=r*4+c where r is row and c is column.
KSM_PRESS	Key index n was released. n=r*4+c where r is row and c is column.
KSM_ERR_HW_OVF	Keyboard driver overflow error.

Table 10 - Keyboard Events

CONFIDENTIAL

7.2 Atm_key

atm_vkey is a framework module which handles key behavior such as click, double click, hold...etc, and generates virtual key events to the upper layer which is rc_mmi_vkey in this example.

7.3 Rc_mmi_vkey

rc_mmi_vkey is the application module which is designed to listen to keyboard events from the keyboard driver and pass its corresponding virtual key number to atm_vkey. Once atm_vkey detects virtual key behaviors, the registered callback function will be called to run some Bluetooth or IR operations.

7.3.1 Key index, virtual key number and key codes

In this example, the virtual key number is equal to the key index of the keyboard event. Each virtual key number is associated with bluetooth or IR keycode and varies due to the board design. That default mapping is defined in `rc_mmi_vkey_default_x202.h` and `rc_mmi_vkey_default_x221.h`. Table 11 lists the mapping of keycodes.

Key Index	rc_mmi_vkey_default_x221.h			rc_mmi_vkey_default_x202.h		
	Virtual	Bluetooth (page, key)	IR (addr, cmd)	Virtual	Bluetooth (page, key)	IR (addr, cmd)
0	VK_POWER	(0xC, 0x30)	(0x80, 0x46)	VK_POWER	(0xC, 0x30)	(0x80, 0x46)
1	VK_MIC	(0xC, 0x221)	N/A	VK_TVPOWER	N/A	N/A
2	VK_UP	(0xC, 0x42)	(0x80, 0x52)	VK_UP	(0xC, 0x42)	(0x80, 0x52)
3	VK_RIGHT	(0xC, 0x45)	(0x80, 0x1A)	VK_RIGHT	(0xC, 0x45)	(0x80, 0x1A)
4	VK_LEFT	(0xC, 0x44)	(0x80, 0x06)	VK_LEFT	(0xC, 0x44)	(0x80, 0x06)
5	VK_BACK	(0xC, 0x224)	(0x80, 0x1B)	VK_BACK	(0xC, 0x224)	(0x80, 0x1B)
6	VK_HOME	(0xC, 0x223)	(0x80, 0x17)	VK_MIC	(0xC, 0x221)	N/A
7	VK_MENU	(0x7, 0x76)	(0x80, 0x07)	VK_MENU	(0x7, 0x76)	(0x80, 0x07)
8	VK_OK	(0xC, 0x41)	(0x80, 0x0F)	VK_OK	(0xC, 0x41)	(0x80, 0x0F)
9	VK_RICE	N/A	N/A	VK_VOLUP	(0xC, 0xE9)	(0x80, 0x16)
10	VK_PLAY	(0xC, 0xCD)	N/A	VK_HOME	(0xC, 0x223)	(0x80, 0x17)
11	VK_VOLUP	(0xC, 0xE9)	(0x80, 0x16)	VK_BW	(0xC, 0xB6)	N/A

CONFIDENTIAL

12	VK_DOWN	(0xC, 0x43)	(0x80, 0x13)	VK_DOWN	(0xC, 0x43)	(0x80, 0x13)
13	VK_BW	(0xC, 0xB6)	N/A	VK_VOLDN	(0xC, 0xEA)	(0x80, 0x15)
14	VK_FW	(0xC, 0xB5)	N/A	VK_RICE	N/A	N/A
15	VK_VOLDN	(0xC, 0xEA)	(0x80, 0x15)	VK_FW	(0xC, 0xB5)	N/A

Table 11 - Keycodes Mapping

7.4 Virtual Key Tables

atm_vkey handles the key behavior and generates events according to the virtual key table registered by rc_mmi_vkey. In initialization, rc_mmi_vkey registers 3 virtual key tables into atm_vkey.

For rc_mmi_vkey, these virtual key tables work according to the different MMI states.
In different MMI states, the rc_mmi_vkey will pass vkey into atm_vkey with the corresponding virtual key table. Table 12 lists 3 tables in different MMI states. Table 13, Table 14 and Table 15 list registered keys in each virtual key table.

Virtual Key Table	MMI States
rc_key_event_hid_not_ready	MMI_S_BOOTED MMI_S_INITING MMI_S_IDLE MMI_S_PAIRING MMI_S_RECONNING MMI_S_CONNECTED MMI_S_ATVV_ONLY
rc_key_event_hid_ready	MMI_S_HID_ONLY MMI_S_HID_ATVV MMI_S_ATVVING MMI_S_HID_READY MMI_S_HID_STREAMING
rc_key_event_rf_test	MMI_S_RF_TEST

Table 12 - Virtual Key Tables

rc_key_event_hid_not_ready

CONFIDENTIAL

Key Event	Action
Down (any key)	Save the key index.
Hold (forward+backward) then click (menu)	Reset device
Hold (forward+backward) then click (asterisk)	Enter RF test mode

Table 13 - rc_key_event_hid_not_ready Virtual Key Table

rc_key_event_hid_ready	
Key Event	**Action**
Hold (OK + Vol-) for 2 sec.	Delete bonding and enter pairing.
First Down (any key)	Send Bluetooth and Infrared (If enabled) key
Non-first Down (any key)	Send Bluetooth key up.
Last Up (any key)	Send Bluetooth key up.
Hold (forward+backward) then click (menu)	Reset device
Hold (forward+backward) then click (asterisk)	Enter RF test mode.
Hold (forward+backward) then click (down)	Disconnect link
Hold (forward+backward) then click (OK)	Key sent auto test
Hold (asterisk) then click (vol+)	Audio gain add 1 (value lost after hibernate)
Hold (asterisk) then click (vol-)	Audio gain minus 1 (value lost after hibernate)
Hold (asterisk) then click (mic)	ATVV streaming stress test.

CONFIDENTIAL

Table 14 - rc_key_event_hid_ready Virtual Key Table

rc_key_event_rf_test	
Key behavior	**Action**
Last up (vol+)	Increase TX power
Last up (vol-)	Decrease TX power
Last up (up)	Increase current channel number
Last up (down)	Decrease current channel number

Table 15 - rc_key_event_rf_test Virtual Key Table

7.5 Bluetooth Key

The HID_remote uses HID service to send key codes to peer device. In this application, rc_hogp is designed to interact with ble_hogpd and implements APIs for virtual key tables in rc_mmi_vkey to send the key code . Figure 11 shows the hierarchy:

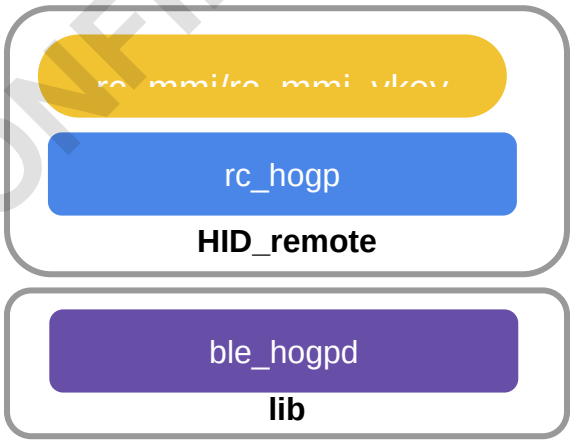

Figure 11 -HID API Hierarchy

CONFIDENTIAL

7.5.1 ble_hogpd

Ble_hogpd implements APIs and parameters for the application to easily configure and use the HID service functionalities. There are four states with the type of ble_hogpd_state_t defined in this module. It reflects current HID service status and can be obtained from ble_hopgd_get_peer_info() API. Table 16 describes these states:

ble_hogpd_state_t	Description
BLE_HOGPD_DISABLED	ble_hogpd is registered. But the framework is not initialized by atm_gap_start.
BLE_HOGPD_IDLE	HID service is added after the framework initialization.
BLE_HOGPD_ENABLED	HID service is enabled after Bluetooth link connection.
BLE_HOGPD_READY	HID service is ready to use due to client characteristic configuration descriptor(s)(CCCD) was(were) enabled.

Table 16 - ble_hogpd_state_t States

The configuration of HID service is accomplished after registering ble_hogpd module by calling atm_gap_prf_reg(BLE_HOGPD_MODULE_NAME, P), where P is the parameter with type of ble_hogpd_param_t provided by the application, including configure settings, support reports, report maps and callback functions. Table 17 describes the APIs and callback functions:

API/callbacks	Description
ble_hogpd_send_report	Send a report to the HID host.
ble_hogpd_send_report_read_cfm	Send a confirmation of HID host when received a read request from HID host.
ble_hogpd_send_report_write_cfm	Send a confirmation of HID host when received a write request from host
ble_hogpd_report_claim	Claim a report buffer for filling data. This report would be sent later through ble_hogp_report_send API.
ble_hogpd_report_send	Send a report to the HID host. This API is used to send the report buffer claimed from ble_hogp_report_claim.
ble_hogpd_get_peer_info	Get current state and CCCD mask status.
ble_hogpd_param_t:: state_ind	Callback function which is called when state is changed.
ble_hogpd_param_t:: report_read_req	Callback function which is called when received a read request from HID host. The Application should respond by

CONFIDENTIAL

	ble_hogpd_send_report_read_cfm API.
ble_hogpd_param_t:: report_write_req	Callback function which is called when received a write request from HID host. The Application should respond by ble_hogpd_send_report_write_cfm API.

Table 17 - ble_hogpd APIs and Callback Functions

7.5.2 rc_hogp

rc_hogp is part of the HID_remote application. It implements simple APIs for other application modules in order to utilize functions of ble_hogpd. It also notifies rc_mmi when the callback function was called from ble_hogpd. Table 18 shows the APIs and its caller:

API	Description	Caller
rc_hogp_param	Get parameter for ble_hogpd registrations. There are two input reports supported in this example. One is usage page 0x0C and another is usage page 0x07, please refer to rc_hogp source file for more information.	rc_gap
rc_hogp_send_single_key	Send Bluetooth key code.	rc_mmi_vkey
rc_hogp_state	Get current ble_hogpd state.	rc_pdm
rc_hogp_get_audio_buf	Get an audio buffer.	rc_hidau

Table 18 - rc_hogp APIs and Caller

CONFIDENTIAL

7.6 IR Key

This application implements NEC IR format to demonstrate IR functionality. The rc_ir application module is designed for rc_mmi_vkey to send the IR key code triggered by virtual key table callback functions. The rc_ir uses the two drivers: nec_ir and ir. Nec_ir defines NEC IR format and uses ir to generate corresponding signals. In these two layer design, users could redefine IR format by replacing nec_ir with another module in other format. See Figure 12.

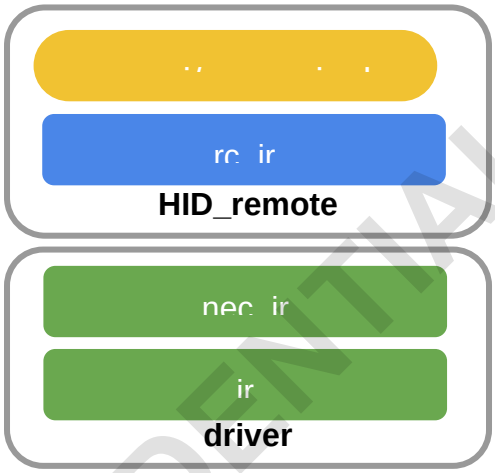

Figure 12 - IR Key Hierarchy

7.6.1 IR

The ir module utilizes timer and GPIO to provide the IR transport layer with configurable IR carrier frequency and duty cycle The parameters were added by calling ir_init in its first arguments. After initialization, the upper layer could register the signal sequence by calling ir_add_period to add in many times with appropare time arguments. After waveform registration, the upper layer could use send_ir_sequence to generate the signal sequence. After the signal is generated, the callback function, which is registered in second arguments of ir_init, will be called to notify the upper layer. Table 19 list the APIs which ir offers:

API	Description		
ir_init	Initialize IR with dedicated carrier frequency and carrier clock duty cycle. Arguments:		
	ir_prot	IR setting. Only the freq and duty_cycle is necessary.	
		freq	Carrier frequency in kHz.
		duty_cycle	Percentage number of carrier duty cycles.

CONFIDENTIAL

	Callback	Callback function after the whole IR sequence is completed.
ir_add_period	Add a period into sequence.	
	carrier_on	If the value is true, a period of carrier frequency will be added in the sequence.
	period_us	The period of time in μs.
reset_ir_sequence	Clear the sequence registered by calling ir_add_period previously.	
send_ir_sequence	Start to send the sequence registered by calling ir_add_period. After the whole sequence is sent out, callback function will be called.	

Table 19 - IR APIs

7.6.2 Nec_ir

The nec_ir bases on the ir module to confirm NEC IR format. It implements 3 essential APIs for the upper layer to use. Table 20 shows the APIs:

API	Description	
nec_ir_init	Initialize NEC IR format and register callback. Arguments:	
	cb_end	Callback function after the sequence is completed.
nec_ir_send	Send a NEC IR code. Arguments:	
	address	Address part of code.
	cmd	Command part of code.
	repeat	If the value is true, repeat code will send continuously after the command is sent out. The repeat code is sent until nec_ir_repeat_end is called. Note: the callback is called after the command is sent without repeat code or after repeat code is stopped.
nec_ir_repeat_end	Stop the current ongoing repeat code.	

Table 20 - Nec_ir APIs

CONFIDENTIAL

Variation of NEC IR is also supported in nec_ir by using compile options. Please refer to source file for more information.

7.6.3 Rc_ir

Rc_ir is designed for two purposes. First is to accept multiple IR keys and send them sequentially. Secondly is to support delay sent in each API call. Table 21 list its APIs:

API	Description
Rc_ir_init	Initialize nec_ir and internal queue.
Rc_ir_send	Send IR key code. If an ongoing IR key exists, save it into the queue and send after previous key codes are sent out.
Rc_ir_repeat_end	Stop the repeat code since rc_ir always sends out the repeat code after the key code was sent. This API should be called as many times as rc_ir_send.

Table 21 - rc_ir APIs

CONFIDENTIAL

8 Voice Search

This example provides voice search functionality through ATVV or HID transmission. The ATVV transmission uses ADPCM voice encoder. In HID transmission, different encoders can be used. In this example, mSBC and ADPCM encoders are used.

- Audio input
  Collect audio data from the microphone and compress it to conformed format, which are the PDM driver, encoders and the corresponding application module called `rc_pdm`. Two encoders are supported in this example.
  - ADPCM encoder
    Interactive Multimedia Association (IMA) ADPCM encoder implemented in `adpcm_enc`.
  - mSBC encoder
    Iimplemented in `sbc_enc_wrapper` which is a wrapper APIs used to fill customer's own mSBC encoder.
- Audio transmission
  Transmit encoded audio data to the TV device, which are Bluetooth API modules and its corresponding application modules. Two Bluetooth transmissions are supported in this example.
  - Voice over HID
    - Bluetooth API: ble_hogpd
    - Application: rc_hidau
  - Voice over ATVV
    - Bluetooth API: ble_atvvs
    - Application: rc_atvv

Figure 13 shows the hierarchy and data flows in ATVV, ADPCM over HID and mSBC over HID .

CONFIDENTIAL

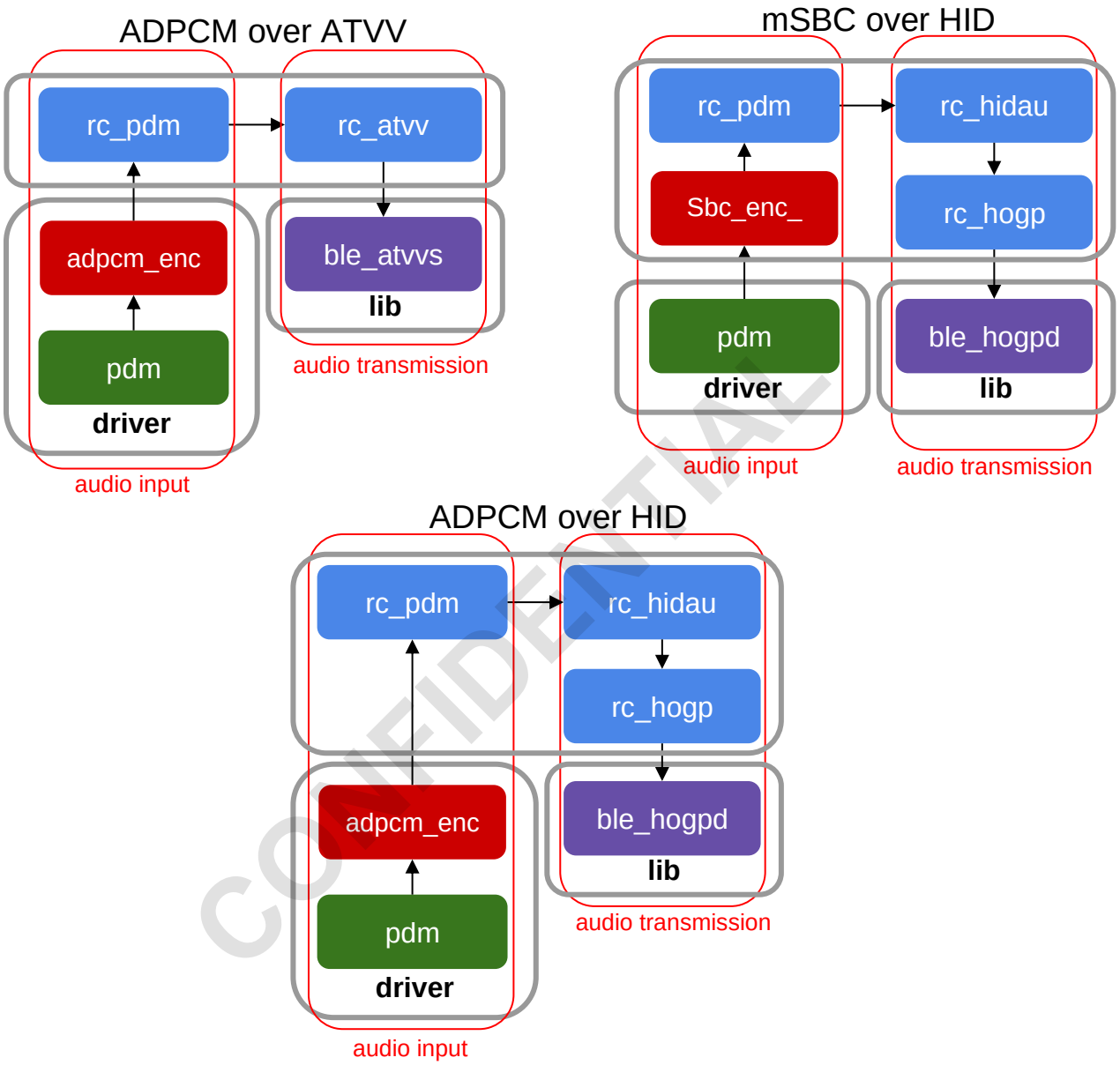

Figure 13 - Voice Search Function Hierarchy and Data Flows

8.1 Audio Input

This example uses the PDM microphone to collect 16 kHz by 16 bits PCM data and encode it before sending via Bluetooth. The pdm driver provides simple APIs and configuration to control ATM2/ATM3 PDM hardware.. The application layer module rc_pdm is designed to receive data from pdm driver and compress it by calling encoders and handing it over to the audio transmission part.

CONFIDENTIAL

8.1.1 PDM

The ATM2/ATM3 provides PDM microphone using two pins and supports three clock speeds (500 KHz, 1 MHz, and 2 MHz). By adding -DPDM_CLOCK=<value>, -DPDM_CLK_IO=<value> and -DPDM_DATA_IO=<value> options in the makefile CFLAGS variable, the pdm driver would change to the corresponding configurations. Table 22 describes the options:

Options	Description
PDM_CLK_IO	Allowed values: ● 24: Use P24 as the PDM clock pin. ● 9: Use P9 as the PDM clock pin.
	Pdm clock pin. If not set, default value is 24.
PDM_DATA_IO	Allowed values: ● 25: Use P25 as the PDM data pin. ● 10: Use P10 as the PDM data pin.
	Pdm data pin. If not set, default is P25.
PDM_CLOCK	Allowed values: ● 0: Use 500 kHz as the frequency of the PDM clock. ● 1: Use 1 MHz as the frequency of the PDM clock. ● 2: Use 2 MHz as the frequency of the PDM clock. ● 3: Use 2 MHz as the frequency of the PDM clock.
	Pdm clock rate. If not set, default is 2Mhz

Table 22 - PDM Microphone Configuration Options

The pdm driver provides 5 APIs: pdm_start, pdm_stop, pdm_pause, pdm_resume, and pdm_is_paused. After calling the pdm_start, the pcm data will pass to the callback function which is provided as an argument of the pdm_start API. See Table 23.

API	Description
pdm_start	Start collecting audio data from the PDM microphone with provided gain value. After started, 32 samples will pass to callback function every 2 ms Arguments: ● gain: gain value from 0 to 120 ● feed_cb: callback function that received data ● ovf_cb: callback function called when overflow happens.
pdm_stop	Stop collecting audio data.

CONFIDENTIAL

pdm_pause	Pause the started collecting operation.
pdm_resume	Resume the paused operation.
pdm_is_paused	Check if the operation is paused

Table 23 - PDM Driver APIs

8.1.2 Encoders

8.1.2.1 Adpcm_enc

Adpcm_enc is a standard IMA ADPCM ¼ compression ratio encoder. There are three APIs in this design: adpcm_reset, adpcm_get_index and adpcm_encode_sample. After adpcm_reset calling at the beginning of the procedure, the application would start to call adpcm_encode_sample by providing a 16 bits audio sample to produce nibble encoded data. See Table 24.

API	Description
adpcm_reset	Reset encoder.
adpcm_get_index	Get the current ADPCM index for frame format usage.
adpcm_encode_sample	Encode 16 bits PCM data to 4 bits.

Table 24 - ADPCM Encoder APIs

CONFIDENTIAL

8.1.2.2 Sbc_enc_wrapper

The Sbc_enc_wrapper wrapper module provides two empty functions for customers to fill their mSBC encoder or any other voice compression algorithm into its functions. In addition, there are two macro definitions that need to be modified. Table 25 shows the descriptions of APIs and Macros.

API / Macros	Description
MSBC_AU_BUF_SIZE	The number of bytes that rc_pdm will receive to the PCM audio buffer for every sbc_enc calling.
MSBC_ENC_SIZE	The number of bytes that will be generated on every sbc_enc calling.
sbc_init	Encoder initialization function. It will be called once

		before every voice search starts.
param[out]	void **context	Provide the private opaque context object to the caller.
sbc_enc		Audio buffer encoding. This will be called when the audio buffer is reached MSBC_AU_BUF_SIZE and.
param[in]	void *context	The private opaque context was from sbc_init.
param[in]	uint8_t *src	Buffer will be encoded. It contains pcm samples from PDM with MSBC_AU_BUF_SIZE bytes.
param[out]	uint8_t *dst	Buffer which is used to store encoded data. Its size is MSBC_EN_SIZE bytes.

Table 25 - Sbc_enc_wrapper Module APIs and Macros

8.1.3 Rc_pdm

Rc_pdm is part of the HID_remote application. Rc_pdm receives PCM samples from PDM driver and uses encoders to compress it and sends them to audio transmission modules, which are rc_atvv and rc_hidau. The features are shown on Table 26.

Feature	Description
Gain adjustment	PDM_DEFAULT_GAIN definition to specify default gain. If it is not defined, default is 85.
	Use rc_pdm_gain_adjust API to adjust.
PDM microphone power switch control	PDM_IO definition (-DPDM_IO = xx) to specify GPIO for power switch control. Default is 22 if not defined.
Audio data process	Encode PCM data and provide it to the application Bluetooth module. • ATVV: adpcm_encode_sample()->rc_atvv_fill_pcm(). • ADPCM over HID: adpcm_encode_sample()->rc_hidau_fill_pcm(). • mSBC over HID: sbc_enc()->rc_hidau_frame_ptr_move().
	Support 8 K PCM mode.

CONFIDENTIAL

	Stop Bluetooth transmit when PCM overflow ● ATVV: rc_atvv_stop_search(). ● Over HID: rc_hidau_stop_search().
	PCM data profiling ● PCM frequency and quantity. ● CPU utilization of encoder.

Table 26 - rc_pdm Features.

Table 27 describes the options:

Options	Description
PDM_DEFAULT_GAIN	Allowed values: 0 to 120.
	Pdm gain. If not set, default value is 85.
PDM_IO	Allowed values: Any GPIO number.
	Pdm power switch GPIO pin. If not set, default is 22.

Table 27 - rc_dpm Options

8.2 Audio Transmission

8.2.1 Voice over ATVV

This section describes the ATVV related modules and sequence charts.

8.2.1.1 ATVV modules

8.2.1.1.1 ble_atvv

ble_atvv implements essential profile APIs and configurations of the ATVV profile for upper layers to design their own voice search flows. By adding -DATVV_SUPPORTED_CODEC=<value>, -DATVV_AUDIO_TIMEOUT_MS=<value> and -ATVV_AUDIO_TIMEOUT_MS=<value> options in the makefile CFLAGS variable, the ble_atvv would change to the corresponding configurations. Table 28 describes the options:

Option	Description
ATVV_SUPPORTED_CODEC	Allowed values: ● CODEC_ADPCM_8K_16BIT: Support 8 kHz ADPCM

CONFIDENTIAL

	only. • CODEC_ADPCM_8K_16K_16BIT: Support 8 kHz/16 kHz ADPCM.
	Supported codec. If not set, the default value is CODEC_ADPCM_8K_16K_16BIT.
ATVV_AUDIO_TIMEOUT_MS	Allowed values are from 0 to 65535 Recommended values are bigger than 5000.
	Audio timeout value in milliseconds. If not set, the default value would be 10000 (10 seconds).
ATVV_MAX_AUDIO_PACKET	Allowed values are from 1 to 255.
	Maximal number of unsent audio packets. If not set, the default is 48.

Table 28 - ATVV Configuration Options

Like other profile APIs in SDK, `ble_atvv` is registered by calling `atm_gap_prf_reg()` API with `BLE_ATVVS_MODULE_NAME` macro as its first argument and profile parameters as its second argument. In this example, `atm_gap_prf_reg()` was called in `rc_gap_init()` and the profile argument is derived from the return value of `rc_atvv_param()` in `rc_atvv`. There are 3 profile parameters defined as type of `ble_atvvs_param_t`. All parameters are callback functions. There are 10 APIs defined in `ble_atvv`. Table 29 describes the 13 APIs and Table 30 describs 3 callback functions:

API	Description
ble_atvvs_start_search	Send START_SEARCH to ATV.
ble_atvvs_dpad_select	Send DPAD_SELECT to ATV.
ble_atvvs_audio_start	Send AUDIO_START to ATV.
ble_atvvs_audio_stop	Send AUDIO_STOP to ATV.
ble_atvvs_mic_open_error	Send MIC_OPEN_ERROR to ATV.
ble_atvvs_claim_audio_buf	Get ATVV audio buffer which is used to carry ATVV audio frame.
ble_atvvs_send_audio_buf	Send ATVV audio buffer to ATV.
ble_atvvs_free_audio_buf	Free ATVV audio buffer.
ble_atvvs_state	Get current ATVV state.

CONFIDENTIAL

ble_atvvs_asst_model	Get negotiated assistant model with current ATV
ble_atvvs_reg_replish_callback	Register callback function for indication of buffer replenish. If the callback function was registered, it will be called when every buffer is sent out.

Table 29 - ATVV APIs

Callback Functions	Description
cb_atvv_ready	Called when ATVV is ready or not ready. When all the client characteristic configuration descriptions are enabled, ATVV will become ready.
cb_atvv_mic_open_ind	Called when received ATV_MIC_OPEN from ATV.
cb_atv_mic_close_ind	Called when received ATV_MIC_CLOSE from ATV.

Table 30 - ATVV Callback Functions

Note: The `AUDIO_STOP`, `AUDIO_START`, `DPAD_SELECT`, `START_SEARCH` and `MIC_OPEN_ERROR` are defined in the ATVV specification.

8.2.1.1.2 rc_atvv

`rc_atvv` is part of the HID_remote application. It implements simple APIs to `rc_gap`, `rc_mmi`, `rc_mmi_vkey` and `rc_pdm` for ATVV control flow and data transfer. It also notifies `rc_mmi` when the callback function was called from `ble_atvv`. Table 31 shows the APIs and its caller:

API	Description	Caller
rc_atvv_param	Get parameter for ble_atvv registrations	rc_gap
rc_atvv_start_search	Start audio search.	rc_mmi_vkey
rc_atvv_stop_search	Stop audio search.	rc_pdm
rc_atvv_dpad_select	Send DPAD_SELECT to TV	rc_mmi_vkey
rc_atvv_fill_pcm	Fill encoded audio sample byte to the current audio frame buffer which is allocated from ble_atvvs_claim_audio_buf. And the current audio frame buffer will be sent out by calling ble_atvvs_send_audio_buf when the frame buffer is full.	rc_pdm
rc_atvv_is_8k	Return whether the current microphone is open in 8K mode.	rc_mmi

CONFIDENTIAL

rc_atvv_is_legacy_model	Return whether legacy model is used	rc_mmi_vkey
rc_atvv_is_htt_model	Return whether HTT model is used	rc_mmi_vkey
rc_atvv_init	Initialize internal events and register the replenish callback to ble_atvv.	rc_mmi

Table 31 - Rc_atvv APIs and Callers

8.2.1.2 ATVV Sequence Chart

The overall procedures are accomplished with cooperation of `rc_mmi`, `rc_mmi_vkey`, `rc_atvv`, `rc_pdm` and `ble_atvv`. This section lists the mapping between sequences and implementation.

8.2.1.2.1 Initialization (0.4e)

After the Remote device is paired and connected, Android TV Device will send it a command to get capabilities of the Remote. The Remote should then respond back with its capabilities. Figure 14 shows the sequence:

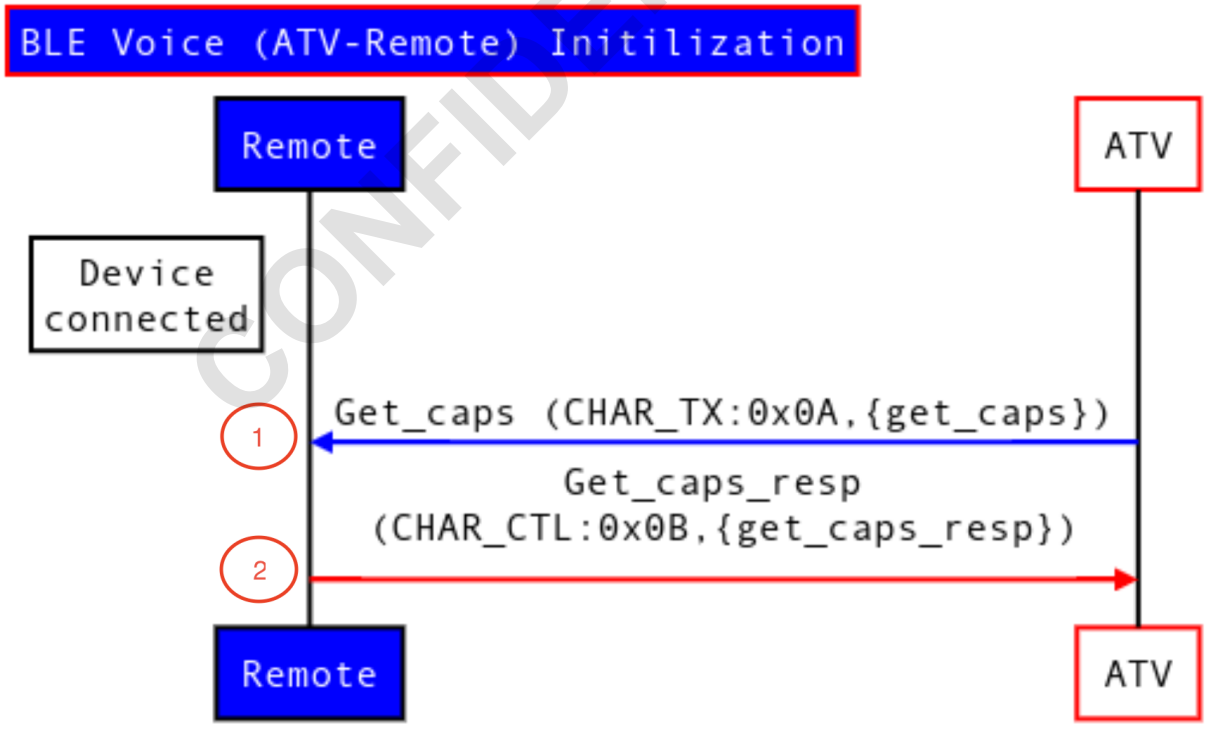

Figure 14 - ATVV Initialization Sequence

Table 32 list the locations of mapping points (red circles in Figure 14):

CONFIDENTIAL

Mapping Point	Module	Location
①	ble_atvv	atvvs_atvv_tx_recv_ind_handler()::case TX_GET_CAPS
②	ble_atvv	atvvs_atvv_tx_recv_ind_handler() →
	ble_atvv	ble_atvvs_send_ctl()
	The capability is responded based on the ATVV_SUPPORTED_CODEC.	

Table 32 - ATVV Initialization Sequence Mapping Points

8.2.1.2.2 Voice search (0.4e)

The user will initiate the voice search by pressing the microphone button. The Remote will send a key event to the Android TV Device. It will also send the `START_SEARCH` to Android TV. When the Android TV Device is ready to start receiving data from the Remote it will send `ATV_MIC_OPEN` to Remote. If the `ATV_MIC_OPEN` is not accepted by Remote, the Remote will send `MIC_OEPN_ERROR` to the Android TV Device. If the `ATV_MIC_OPEN` is accepted by Remote, the Remote will send `AUDIO_START` then periodically send audio frames (`AUDIO_DATA`) and `AUDIO_SYNC` to the Android TV Device. When the Android TV Device has detected the user has stopped speaking it will send `ATV_MIC_CLOSE` to Remote and Remote will send `AUDIO_START`

CONFIDENTIAL

back to the Android TV Device. Figure 15 shows the sequence:

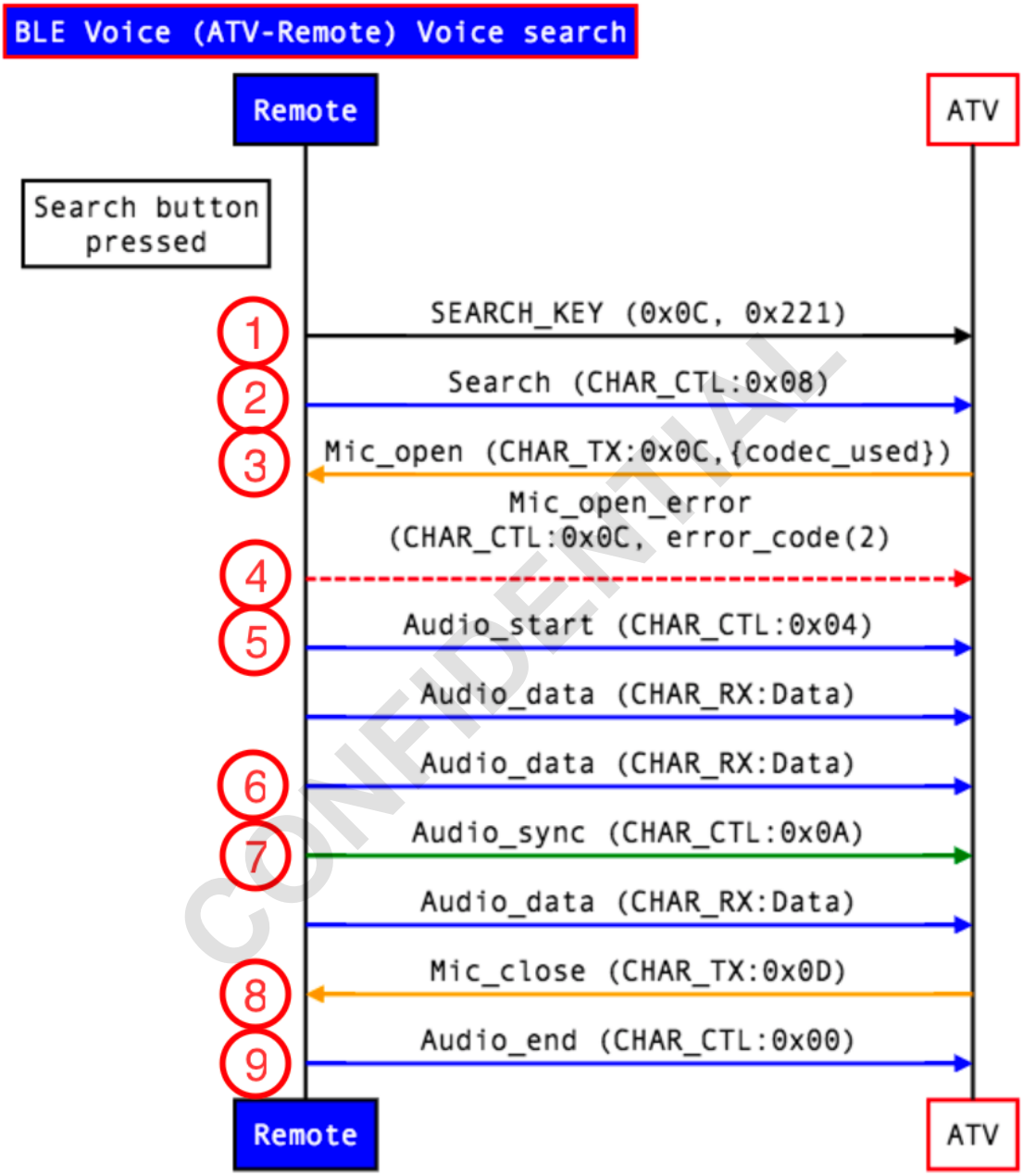

Figure 15 - ATVV Voice Search Sequence

Table 33 list the locations of red circles from Figure 15:

Mapping Point	Module	Location
①	rc_mmi_vkey	rc_send_rpt() →
	rc_hogp	rc_hogp_send_single_key(BT_MIC)
②	rc_mmi_vkey	rc_send_rpt() →
	rc_atvv	rc_atvv_start_search() →
	ble_atvv	ble_atvvs_start_search()
③	ble_atvv	Atvvs_atvv_tx_recv_ind_handler()::case TX_MIC_OPEN →
	rc_atvv	rc_atvv_mic_open_ind() →
	rc_mmi	rc_mmi_transition(MMI_OP_OPEN_MIC) →
	rc_mmi	mmi_s_open_mic() →
	rc_pdm	rc_pdm_device_pwr_on()
	Delay for RC_MMI_OPEN_MIC_DELAY_CS centiseconds	
	rc_mmi	rc_mmi_mic_ctl_timer_msg_ind() →
	rc_pdm	rc_pdm_start(RC_PDM_NORMAL, rc_atvv_is8k())
④	rc_atvv	rc_atvv_mic_open_ind() →
	ble_atvv	ble_atvvs_mic_open_error()
⑤	rc_atvv	rc_atvv_mic_open_ind() →
	ble_atvv	ble_atvvs_audio_start()
⑥	rc_pdm	rc_pdm_event() →

CONFIDENTIAL

	rc_atvv	Rc_atvv_fill_pcm() →
	ble_atvv	ble_atvvs_send_audio_buf()
⑦	ble_atvv	ble_atvvs_send_audio_buf() →
	ble_atvv	atvv_ctl_sync()
⑧	ble_atvv	Atvvs_atvv_tx_recv_ind_handler()::case TX_MIC_CLOSE →
	rc_atvv	rc_atvv_mic_close_ind() →
	rc_pdm	rc_pdm_stop()
⑨	ble_atvvs	Atvvs_atvv_tx_recv_ind_handler()::casse TX_MIC_CLOSE →
	ble_atvvs	atvv_ctl_audio_stop()

Table 33 - Voice Search Sequence Mapping Points

CONFIDENTIAL

8.2.1.2.3 Initialization (1.0)

In ATVV version 1.0, the Android TV device and the Remote will negotiate the supported assistant interaction models with the get capability command after paired and connected. There are currently three models introduced: On-request, Press-to-talk (PTT) and Hold-to-talk (HTT). The On-request model is also called the legacy model and must be supported by both sides. If the Android TV device and the Remote both support either PTT or HTT model, the Remote can then arbitrarily decide which model should be used. In current implementation, the Remote will prefer the HTT over the PTT model.

8.2.1.2.4 Voice search (1.0)

In the legacy model, the Remote will notify about every assistant button pressed event by sending a START_SEARCH message and another KEYCODE_ASSIST HID key event to the Android TV. The Android TV device can then decide whether the audio data is requested or not by sending a MIC_OPEN command. The Remote device should keep the microphone open until a MIC_CLOSE command is received or the timeout event occurs before a MIC_EXTEND message is received.

In the PTT and HTT interaction model, the KEYCODE_ASSIST won't be sent to the Android TV

over the HID interface after the assistant button is pressed. The Remote device will directly start to transfer the audio data without waiting for the request message from the Android TV. The audio stream will last until the timeout event occurs. However The TV side can still proactively stop the audio data by sending the MIC_CLOSE command once it gets enough information or detects the user is not speaking. The audio stream will also stop immediately if the assistant button is released in the HTT model.

8.2.2 Voice over HID (VoHID)

This section describes the modules related to voice over HID and sequence flow chart.

8.2.2.1 VoHID Modules

8.2.2.1.1 ble_hogpd

Ble_hogpd implements essential profile APIs and configurations of the HID profile for upper layers to interact with the HID host. Please refer to 7.5.1 ble_hogpd for more details. The VoHID uses ble_hogpd_report_claim and ble_hogpd_report_send to accomplish audio transmission. Please refer to the next sections for more details.

8.2.2.1.2 rc_hogpd

rc_hogp is part of the HID_remote application. It implements simple APIs for other application modules in order to utilize functions of ble_hogpd. Please refer to 7.5.2 rc_hogp for more details.

8.2.2.1.3 rc_hidau

rc_hidau is part of the HID_remote application. It implements simple APIs to `rc_mmi`, `rc_mmi_vkey` and `rc_pdm` for VoHID control flow and data transfer. Table 34 shows the APIs and its caller:

API	Description	Caller
rc_hidau_start_search	Start audio search.	rc_mmi_vkey
rc_hidau_stop_search	Stop audio search.	rc_mmi_vkey
rc_hidau_frame_get	Get the current audio frame buffer and its residue. The audio frame buffer is allocated by rc_hogpd_get_audio_buf.	rc_pdm
rc_hidau_frame_ptr_move	Move the current audio frame buffer pointer. It's called when an amount of audio samples have been filled into the frame buffer. When the pointer is moved to the end, it will be sent out by calling ble_hogpd_report_send.	rc_pdm
rc_hidau_fill_pcm	Fill encoded audio sample byte to the current	rc_pdm

CONFIDENTIAL

	audio frame buffer which is allocated from rc_hogpd_get_audio_buf. And the current audio frame buffer will be sent out by calling ble_hogpd_report_send when the frame buffer is full.	
rc_hidau_init	Initialize internal events and timers.	rc_mmi

Table 34 - rc_hidau APIs and Callers

The HID_AU_REPORT_SIZE value is defined for byte number in an audio frame buffer. The default values of ADPCM and mSBC are 128 and (MSBC_ENC_SIZE *2) respectively.

8.2.2.2 VoHID Sequence Chart

The overall procedures are accomplished with cooperation of `rc_mmi`, `rc_mmi_vkey`, `rc_hidau`, `rc_hogp`, `rc_pdm` and `ble_hogpd`.

8.2.2.2.1 Initialization

After the Remote device is paired and connected, TV Device will make HID ready to use by enabling the Client Characteristic Configuration Descriptor of audio input report. After HID is ready, the voice search function is enabled.

8.2.2.2.2 Voice search

The user will initiate the voice search by clicking the microphone button. The Remote will start sending audio reports to TV. And the user clicking the microphone button again will stop sending the audio reports. Figure 16 shows the sequence.

CONFIDENTIAL

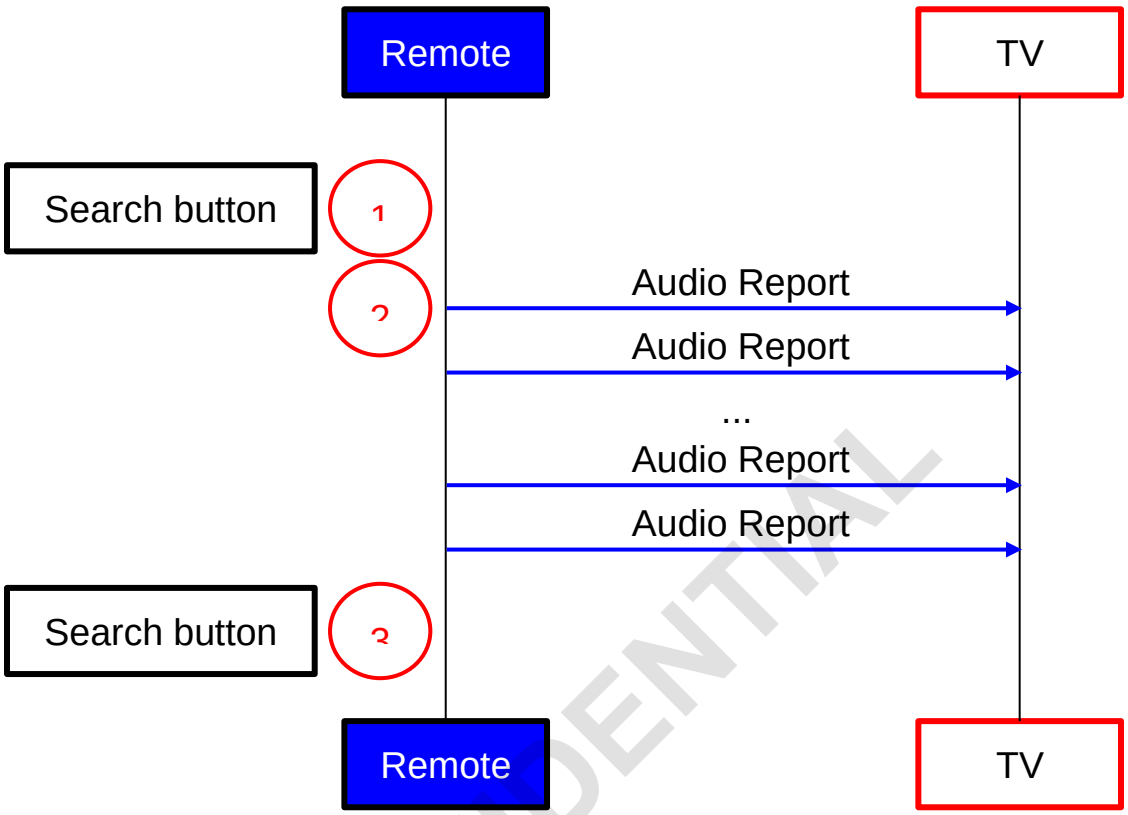

Figure 16 - VoHID Voice Search Sequence

Table 35 list the locations of red circles in Figure 16:

Mapping Point	Location	Module
①	rc_send_rpt(...)	rc_mmi_vkey
	➤ rc_hidau_start_search()	rc_hidau
	HID_AUDIO_OPEN_MIC_DELAY_MS milliseconds	
	rc_mmi_transition(MMI_OP_OPEN_MIC)	rc_mmi
	➤ mmi_s_open_mic()	rc_mmi
	➤ ➤ rc_pdm_device_pwr_on()	rc_pdm
	➤ ➤ rc_mmi_open_mic()	rc_mmi

CONFIDENTIAL

	RC_MMI_OPEN_MIC_DELAY_CS centiseconds	
	rc_mmi_mic_ctl_timer_msg_ind(...) →	rc_mmi
	➤ rc_pdm_start(RC_PDM_NORMAL, false)	rc_pdm
②	**ADPCM over HID**	
	rc_pdm_event(...)	rc_pdm
	➤ adpcm_encode_sample(...)	adpcm_enc
	➤ rc_hidau_fill_pcm(...)	rc_hidau
	➤ ➤ ble_hogpd_report_send(...)	ble_hogpd
	mSBC over HID	
	rc_pdm_event(...)	rc_pdm
	➤ rc_hidau_frame_get(...)	rc_hidau
	➤ sbc_enc(...)	sbc_enc_wrapper
	➤ rc_hidau_frame_ptr_move(...)	rc_hidau
	➤ ➤ ble_hogpd_report_send(...)	ble_hogpd

Table 35 - Voice Search Sequence Mapping Points

CONFIDENTIAL

9 LED

LED control is implemented by using led_blinking module. GPIO and a timer are used to provide a set of Application Programming Interfaces (APIs) for the rc_mmi. It was called in the transition function of the rc_mmi state machine. Table 36 shows the LED definition in each mmi state.

Device State	LED Behavior
RECONNECT	LED blinking with 100 ms interval
PAIRING	LED blinking with 250 ms interval
CONNECTED	When key pressed, LED on for 100 ms then off
VOICE SEARCH	When key pressed, LED on for 100 ms then off
HIBERNATE	LED off

Table 36 - LED Behavior

CONFIDENTIAL

CONFIDENTIAL

ATMOSIC TECHNOLOGIES - DISCLAIMER

This product document is intended to be a general informational aid and not a substitute for any literature or labeling accompanying your purchase of the Atmosic product. Atmosic reserves the right to amend its product literature at any time without notice and for any reason, including to improve product design or function. While Atmosic strives to make its documents accurate and current, Atmosic makes no warranty or representation that the information contained in this document is completely accurate, and Atmosic hereby disclaims (i) any and all liability for any errors or inaccuracies contained in any document or in any other product literature and any damages or lost profits resulting therefrom; (ii) any and all liability and responsibility for any action you take or fail to take based on the information contained in this document; and (iii) any and all implied warranties which may attach to this document, including warranties of fitness for particular purpose, non-infringement and merchantability. Consequently, you assume all risk in your use of this document, the Atmosic product, and in any action you take or fail to take based upon the information in this document. Any statements in this document in regard to the suitability of an Atmosic product for certain types of applications are based on Atmosic's general knowledge of typical requirements in generic applications and are not binding statements about the suitability of Atmosic products for any particular application. It is your responsibility as the customer to validate that a particular Atmosic product is suitable for use in a particular application. All content in this document is proprietary, copyrighted, and owned or licensed by Atmosic, and any unauthorized use of content or trademarks contained herein is strictly prohibited.

Copyright ©2021 by Atmosic Technologies. All rights reserved.

www.atmosic.com