RTL8762C Keyboard Application Design Spec

V1.0

2018/5/9

Revision History

Date	Version	Comments	Author	Reviewer
2018/5/9	Draft		Parker_xue	
2018/9/12	Release	Add key input/output report Formatting and correction		Astor

Contents

Figure List

Table List

1 Overview

1.1 Device List

1. Bee2 Evaluation Board
2. 4*4 Matrix Keyboard

1.2 System Requirement

Tools that need to be downloaded and installed on PC:

1. Keil MDK-ARM
2. SEGGER's J-Link tools
3. RTL8762C SDK
4. RTL8762C Flash programming algorithm

1.3 Terminology Definition

1. DLPS: Deep Low Power State
2. SUT: System Under Test, Android or IOS mobile devices with Keyboard application
3. HID: Human Interface Device protocol

2 Software Structure

Keyboard application mainly interacts with IO Driver and BT to accomplish specific function. IO module includes Keyscan module and GPIO key module. Single GPIO module is used to pair key. The software structure is shown below:

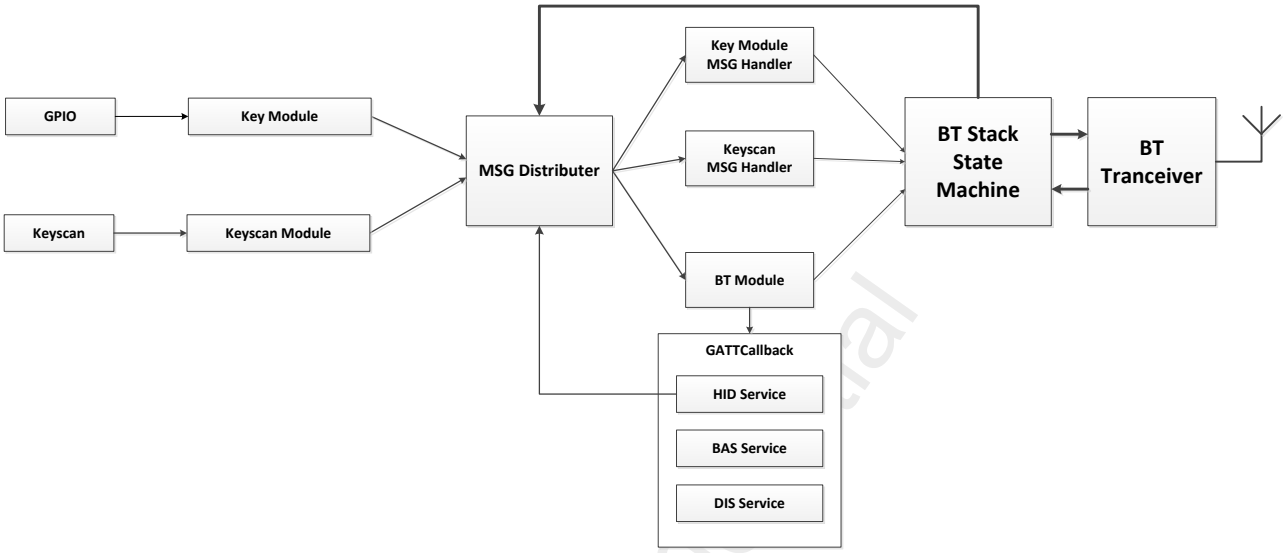

Figure 2.1 Software Structure

After booting and initializing system, APP task starts to run and wait for messages from GPIO key, keyscan and upper stack. To simplify handling of various types of message, App Task will create an Event Queue to store type of external message instead of receiving message directly. Then APP Task acquires information from corresponding queue based on message type stored in Event Queue and performs further operation.

```
1.  while (true)
2.  {
3.      if (os_msg_recv(evt_queue_handle, &event, 0xFFFFFFFF) == true)
4.      {
5.          if (event == EVENT_IO_TO_APP)
6.          {
7.              T_IO_MSG io_msg;
8.              if (os_msg_recv(io_queue_handle, &io_msg, 0) == true)
9.              {
10.                 app_handle_io_msg(io_msg);
11.             }
12.         }
13.         else
14.         {
```

```
15.             gap_handle_msg(event);
16.         }
17.     }
18. }
```

2.1 Key Module

Pairing key of Keyboard application is accomplished by applying single GPIO. If not paired, long press pair key to send pairing advertising. If paired, long press pair key to clear pairing information and resend pairing advertising.

GPIO connected to key needs to be set to level trigger mode and accomplish key debounce with hardware timer. When configuring GPIO interrupt, first set to low level trigger. When key is pressed to raise GPIO interrupt, system reads key status in interrupt handler function, set press and release debounce time and then start timer.

In timer interrupt handler function, system reread key status. If current status is different from the previous status, stop current operation which contains key bouncing. Otherwise, generate key press message and reverse GPIO interrupt trigger level to prepare for next press/release operation. Finally, enable GPIO interrupt to send key pressed message to APP task

8762C GPIO supports hardware debounce with edge trigger to minimize code size by omitting timer. But this function cannot be used in DLPS mode.

Long press can be implemented with timer. After key press event happens, APP Task receives key press message and start timer. If task does not receive key release event before timer is timeout, task will handle key long press event in timeout callback function. Otherwise, if key is released before timer is timeout, timer will be turned off.

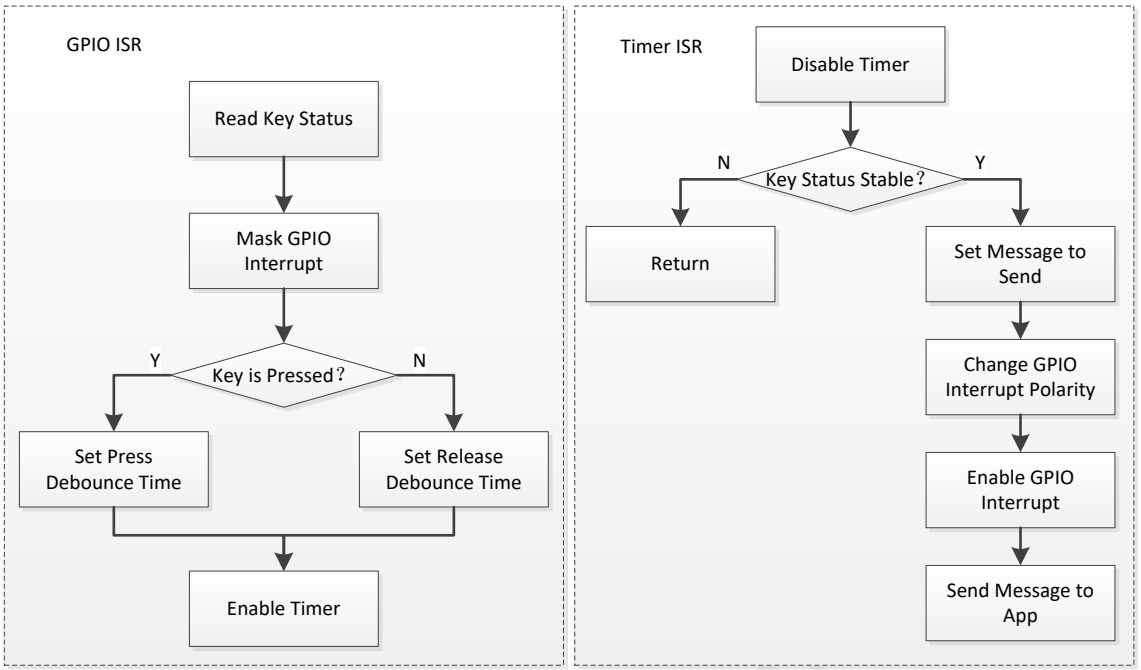

Figure 2.2 Key press handler process

2.2 Keyscan Module

Keyboard application uses Keyscan module to scan matrix keyboard. When key press event is scanned, message is sent to APP through keyscan interrupt.

```
1.  void keyscan_interrupt_handler(void)
2.  {
3.      APP_PRINT_INFO0("[keyscan_interrupt_handler] interrupt handler");
4.
5.      T_IO_MSG bee_io_msg;
6.
7.      if (KeyScan_GetFlagState(KEYSCAN, KEYSCAN_INT_FLAG_SCAN_END) == SET)
8.      {
9.          keyscan_global_data.is_allowed_to_enter_dlps = true;
10.         KeyScan_INTMask(KEYSCAN, KEYSCAN_INT_SCAN_END, ENABLE);  /* Mask keyscan interrupt */
11.
12.         keyscan_global_data.cur_fifo_data.len = KeyScan_GetFifoDataNum(KEYSCAN);
13.         if (keyscan_global_data.cur_fifo_data.len != 0)
14.         {
15.             /* read keyscan fifo data */
16.             KeyScan_Read(KEYSCAN, (uint16_t *) & (keyscan_global_data.cur_fifo_data.key[0]),
17.                          keyscan_global_data.cur_fifo_data.len);
18.             keyscan_global_data.is_key_pressed = true;
19.             keyscan_global_data.is_all_key_released = false;
```

```
20.
21.             /* start sw timer to check press status */
22.             if (!os_timer_restart(&keyscan_timer, KEYSCAN_SW_INTERVAL))
23.             {
24.                 APP_PRINT_ERROR0("[keyscan_interrupt_handler] restart xTimersKeyScan
    failed!");
25.                 /* set flag to default status and reinit keyscan module with debounce enabled
    */
26.                 keyscan_init_data();
27.                 keyscan_init_driver(KeyScan_Debounce_Enable);
28.                 return;
29.             }
30.
31.             if (false == keyscan_global_data.is_allowed_to_repeat_report)
32.             {
33.                 if (!memcmp(&keyscan_global_data.cur_fifo_data,
    &keyscan_global_data.pre_fifo_data,
34.                             sizeof(T_KEYSCAN_FIFIO_DATA)))
35.                 {
36.                     /* some keyscan FIFO data, just return */
37.                     return;
38.                 }
39.                 else
40.                 {
41.                     /* updata previous keyscan FIFO data */
42.                     memcpy(&keyscan_global_data.pre_fifo_data,
    &keyscan_global_data.cur_fifo_data,
43.                            sizeof(T_KEYSCAN_FIFIO_DATA));
44.                 }
45.             }
46.
47.             bee_io_msg.type = IO_MSG_TYPE_KEYSCAN;
48.             bee_io_msg.subtype = IO_MSG_KEYSCAN_RX_PKT;
49.             bee_io_msg.u.buf   = (void *)(&keyscan_global_data.pre_fifo_data);
50.             if (false == app_send_msg_to_apptask(&bee_io_msg))
51.             {
52.                 APP_PRINT_ERROR0("[keyscan_interrupt_handler] send IO_MSG_KEYSCAN_RX_PKT
    message failed!");
53.                 /* set flag to default status and reinit keyscan module with debounce enabled
    */
54.                 keyscan_init_data();
55.                 os_timer_stop(&keyscan_timer);
56.                 keyscan_init_driver(KeyScan_Debounce_Enable);
57.                 return;
```

```
            }
        }
        else
        {
            if (false == keyscan_global_data.is_all_key_released)
            {
                /* keyscan release event detected */
                APP_PRINT_INFO0("[keyscan_interrupt_handler] keyscan release event detected");
                T_IO_MSG bee_io_msg;
                bee_io_msg.type = IO_MSG_TYPE_KEYSCAN;
                bee_io_msg.subtype = IO_MSG_KEYSCAN_ALLKEYRELEASE;

                if (false == app_send_msg_to_apptask(&bee_io_msg))
                {
                    APP_PRINT_ERROR0("[keyscan_interrupt_handler] Send IO_MSG_TYPE_KEYSCAN message failed!");
                }

                keyscan_init_data();
                keyscan_init_driver(KeyScan_Debounce_Enable);
            }
            else
            {
                /*if system active, keyscan no debounce can arrive here*/
                APP_PRINT_INFO0("[keyscan_interrupt_handler] if system active, keyscan no debounce can arrive here");
                keyscan_init_data();
                keyscan_init_driver(KeyScan_Debounce_Enable);
                return;
            }
        }
    }
    else
    {
        /* if not KEYSCAN_INT_FLAG_SCAN_END interrupt */
        APP_PRINT_INFO0("[keyscan_interrupt_handler] not KEYSCAN_INT_FLAG_SCAN_END interrupt");
        keyscan_init_data();
        keyscan_init_driver(KeyScan_Debounce_Enable);
        return;
    }
}
```

97.

2.3 APP Task Message Handler

GPIO key module and Keyscan module send message to APP task, and APP task invokes different functions to handle message according to message type. At last, send data in the format defined in HID Report map (please refer to chapter 3.2) to SUT

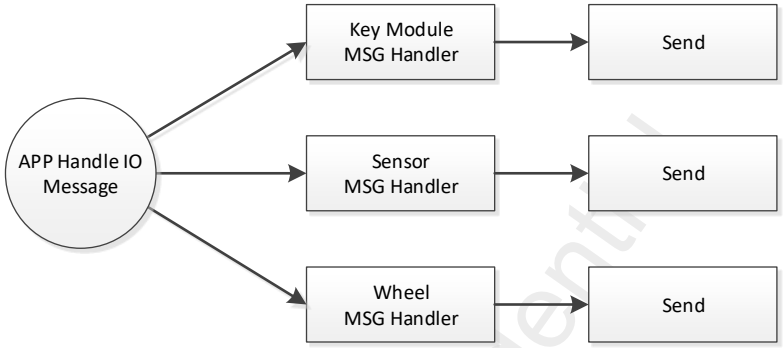

Figure 2.3 APP Task Handler for IO Messages

2.4 Pair and Reconnect Process

Device is not in pair mode when first used. User can long press pair key to trigger pairing mode, then system will send undirected advertising. After host device scan and find the Keyboard device, it can start pairing operation. Device will be added to resolving list and white list after paired. The resolving list is used to resolve Resolvable Private Address.

When reconnecting, APP will send different reconnection advertising type based on different device address type. If address type is public address or static random address, APP will send directed advertising. If address type is resolvable private address, APP will send undirected advertising and use white list to filter. After configuring resolving list and enable resolving, APP can filter device for resolvable private address, for RTL8762C supports LL Privacy.

Besides, when paired device resets, APP will check if pair information exists. If pair information exists, add corresponding device to resolving list and white list. Otherwise, device is not paired.

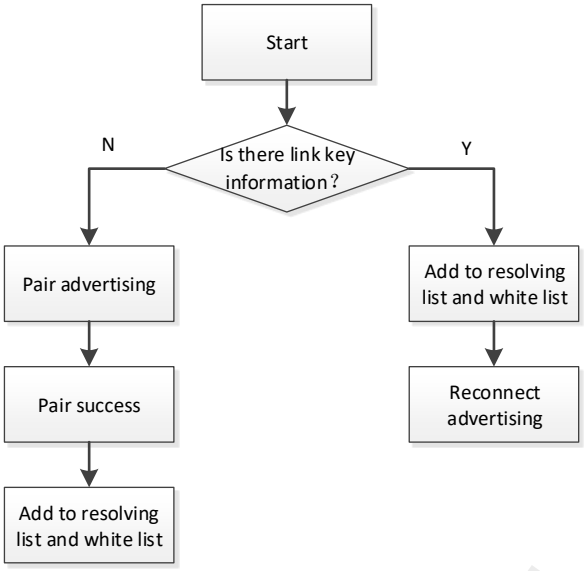

Figure 2.4 Pair and Reconnect Process

3 Bluetooth Related Operations

3.1 Service and Characteristic

Keyboard application contains the following services:

1. HID Service: human interface device protocol
2. Battery Service: report battery level of device, remind user to change battery, and prohibit OTA when battery level is low.
3. Device Information Service: display device information.

Service name and corresponding UUID are shown below

Table 3.1 Service name and corresponding UUID

Service Name	Service UUID
HID Service	0x1812
Battery Service	0x180F
Device Information Service	0x180A

3.1.1 HID Service

Table 3.2 HID Service Characteristic List

Characteristic Name	Requirement	Characteristic UUID	Properties	Description
Protocol Mode	M	0x2A4E	Read/WriteWithoutResponse	See Protocol Mode
Report	O			
Report:Input	M	0x2A4D	Read/Write/Notify	See Report Characteristict
Report:Output	M	0x2A4D	Read/Write/ WriteWithoutResponse	See Report Characteristic
Report:Feature	M	0x2A4D	Read/Write	See Report Characteristic
Report Map	M	0x2A4B	Read	See Report Map
Boot Key Input Report	M	0x2A22	Read/Write/Notify	See Boot Key Input Report
Boot Key Output Report	M	0x2A32	Read/Write/ WriteWithoutResponse	See Boot Key Output Report

HID Information	M	0x2A4A	Read	See HID Information
HID Control Point	M	0x2A4C	WriteWithoutResponse	See HID Control Point
Report: Input(For multimedia key)	M	0x2A4D	Read/Write/Notify	See Report Characteristic

3.1.2 Protocol Mode Characteristic

Protocol Mode Characteristic indicates protocol mode for current HID service, and set protocol mode for expected HID service

Table 3.3 Protocol Mode Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Protocol Mode	Mandatory	uint8	N/A	N/A	Enumerations Key: Value 0: Boot Protocol Mode 1: Report Protocol Mode 2-255: Reserved for future use

3.1.3 Report Characteristic

Report Characteristic includes data in input report, output report and feature report transmitted between HID device and host device

Table 3.4 Report Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Report	Mandatory	uint8	N/A	N/A	This field may be repeated

3.1.4 Report Map Characteristic

Report Map Characteristic is used to define data format in input report, output report and feature report transmitted between HID device and host device.

Table 3.5 Report Map Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Report Map	Mandatory	uint8	N/A	N/A	This field may be repeated

3.1.5 Boot Key Input Report Characteristic

Boot Key Input Report Characteristic is used to boot the HID host device operated in protocol mode and input report with fixed format and length transmitted between HID host device and keyboard.

Table 3.6 Boot Mouse Input Report Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Boot Mouse Input Report	Mandatory	uint8	N/A	N/A	This field may be repeated

3.1.6 Boot Key Output Report Characteristic

Boot Key Input Report Characteristic is used to boot the HID host device operated in protocol mode and output report with fixed format and length transmitted between HID host device and keyboard.

Table 3.7 Boot Mouse Input Report Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Boot Key Output Report	Mandatory	uint8	N/A	N/A	This field may be repeated

3.1.7 HID Information Characteristic

HID Information Characteristic contains HID attributes. The characteristic value is static and can be permanently saved for bonding HID device and host device

Table 3.8 HID Information Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
bcdHID	Mandatory	uint16	N/A	N/A	None
bCountryCode	Mandatory	8bit	N/A	N/A	None
Flags	Mandatory	8bit	N/A	N/A	See Bit Field

Table 3.9 Bit Field

Bit	Size	Name	Definition	
			Key	Value
0	1	Remote Wake	0	The device is not designed to be capable of

				providing wake-up signal to a HID host
			1	The device is designed to be capable of providing wake-up signal to a HID host
1	1	Normally Connectable	0	The device is not normally connectable
			1	The device is normally connectable
2	6	Reserved for future use		

3.1.8 HID Control Point Characteristic

HID Control Point Characteristic is a control point attribute which defines the following HID commands:

1. Suspend
2. Exit Suspend

Table 3.10 HID Control Point Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
HID Control Point Command	Mandatory	uint8	N/A	N/A	Enumerations Key: Value 0: Suspend 1: Exit Suspend 2-255: Reserved for future use

3.1.9 Battery Service

Battery Service includes a battery level characteristic, shown in Table 3.10

Table 3.11 Battery Service Characteristic List

Characteristic Name	Requirement	Characteristic UUID	Properties	Description
Battery Level	M	0x2A19	Read/Notify	See Battery Level

3.1.10 Battery Service Characteristic

Battery level represents current power level which ranges from 0 to 100 percent. Its data format is unsigned 8-bit integer, as shown in Table 3.11

Table 3.12 Battery Level Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information

Battery Level	Mandatory	uint8	0	100	Enumerations Key: Value 101-255: Reserved

3.1.11 Device Information Service

Device Information Service includes 9 characteristics, shown in Table 3.12

Table 3.13 Device Information Service Characteristic List

Characteristic Name	Requirement	Characteristic UUID	Properties
Manufacturer Name String	O	0x2A29	Read
Model Number String	O	0x2A24	Read
Serial Number String	O	0x2A25	Read
Hardware Revision String	O	0x2A27	Read
Firmware Revision String	O	0x2A26	Read
Software Revision String	O	0x2A28	Read
System ID	O	0x2A23	Read
Regulatory Certification Data List	O	0x2A2A	Read
PnP ID	O	0x2A50	Read

3.1.12 Device Information Service Characteristic

Device Information Service contains characteristics of displaying basic information on device name and firmware version, shown in Table 3.13

Table 3.14 Device Information Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Manufacturer Name	Mandatory	utf8s	N/A	N/A	None
Model Number	Mandatory	utf8s	N/A	N/A	None
Serial Number	Mandatory	utf8s	N/A	N/A	None
Hardware Revision	Mandatory	utf8s	N/A	N/A	None
Firmware Revision	Mandatory	utf8s	N/A	N/A	None
Software Revision	Mandatory	utf8s	N/A	N/A	None

1) System ID Characteristic

System ID consists of two fields, 40-bit vendor-defined ID and 24-bit OUI (Organization Unique Identity), as

shown in Table 3.14

Table 3.15 System ID Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Manufacturer Identifier	Mandatory	uint40	0	1099511627775	None
Organization Unique Identifier	Mandatory	uint24	0	16777215	None

2) IEEE 11073-20601 Regulatory Certification Data List Characteristic

IEEE 11073-20601 Regulatory Certification Data List lists different certifications that device need to comply with.

Table 3.16 IEEE 11073-20601 Regulatory Certification Data List Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Data	Mandatory	reg-cert-data-list[1]	N/A	N/A	None

3) PnP ID Characteristic

PnP ID is a group of ID for identifying unique device, including Vendor ID source, Vendor ID and Product Version. These values are used to identify specific type/mode/version of device, as shown in Table 3.16.

Table 3.17 PnP ID Characteristic Value Format

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Vendor ID Source	Mandatory	uint8	1	2	See Enumerations
Vendor ID	Mandatory	uint16	N/A	N/A	None
Product ID	Mandatory	uint16	N/A	N/A	None
Product Version	Mandatory	uint16	N/A	N/A	None

Table 3.18 Enumerations

Key	1	2	3-255	0
Value	Bluetooth SIG assigned Company Identifier value from the Assigned Numbers document	USB Implementer's Forum assigned Vendor ID value	Reserved for future use	Reserved for future use

3.2 Report Map and Data Sending Format

Report Map is used to define data format for transporting input report, output report and feature report data between HID device and host device.

Keyboard includes modifier keys and common keys. The press and release status for each modifier key is represented by 1 bit. Common keys are pressed to send its corresponding usage ID and at most 6 key values can be sent at the same time.

Keyboard data format:

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Modifier Keys	Reserved	Keycode1	Keycode2	Keycode3	Keycode4	Keycode5	Keycode6

Multimedia keys have different report ID, and each has its own data format. HID Keyboard profile in SDK lists 24 Multimedia keys in enum format. Sending data length is 3 bytes, and each bit corresponds to a multimedia key.

Multimedia key data format:

Byte	Bit							
	7	6	5	4	3	2	1	0
1	Scan Next Track	Scan Previous Track	Stop	Play/Pause	Mute	Bass Boost	Loudness	Volume Increment
2	Volume Decrement	Bass Increment	Bass Decrement	Treble Increment	Treble Decrement	AL Consumer Control Configuration	AL Email Reader	AL Calculator
3	AL Local Machine Browser	AC Search	AC Home	AC Back	AC Forward	AC Stop	AC Refresh	AC Bookmarks

Take data 00000001 00000000 00100000 as example, it represents Volume increment and Home keys pressed (refer to bit sequence of report map)

```
const uint8_t hids_report_descriptor[] =
{
    0x05, 0x01,     /* USAGE_PAGE       (Generic Desktop) */
    0x09, 0x06,     /* USAGE            (Keyboard) */
    0xa1, 0x01,     /* COLLECTION       (Application) */
    0x85, HOGP_KB_REPORT_ID,    /* REPORT_ID        (3) */
    0x05, 0x07,     /* USAGE_PAGE       (Keyboard) */
```

```
0x19, 0xe0,     /* USAGE_MINIMUM    (Keyboard Left Control) */
0x29, 0xe7,     /* USAGE_MAXIMUM    (Keyboard Right GUI) */
0x15, 0x00,     /* LOGICAL_MINIMUM  (0) */
0x25, 0x01,     /* LOGICAL_MAXIMUM  (1) */
0x75, 0x01,     /* REPORT_SIZE      (1) */
0x95, 0x08,     /* REPORT_COUNT     (8) */
0x81, 0x02,     /* INPUT           (Data,Var,Abs) */
0x95, 0x01,     /* REPORT_COUNT     (1) */
0x75, 0x08,     /* REPORT_SIZE      (8) */
0x81, 0x01,     /* INPUT           (Cnst,Var,Abs) */
0x95, 0x05,     /* REPORT_COUNT     (5) */
0x75, 0x01,     /* REPORT_SIZE      (1) */
0x05, 0x08,     /* USAGE_PAGE       (LEDs) */
0x19, 0x01,     /* USAGE_MINIMUM    (Num Lock) */
0x29, 0x05,     /* USAGE_MAXIMUM    (Kana) */
0x91, 0x02,     /* OUTPUT          (Data,Var,Abs) */
0x95, 0x01,     /* REPORT_COUNT     (1) */
0x75, 0x03,     /* REPORT_SIZE      (3) */
0x91, 0x01,     /* OUTPUT          (Cnst,Var,Abs) */
0x95, 0x06,     /* REPORT_COUNT     (6) */
0x75, 0x08,     /* REPORT_SIZE      (8) */
0x15, 0x00,     /* LOGICAL_MINIMUM  (0) */
0x25, 0xa4,     /* LOGICAL_MAXIMUM  (164) */    /* Can be 255 */
0x05, 0x07,     /* USAGE_PAGE       (Keyboard) */
0x19, 0x00,     /* USAGE_MINIMUM    (Reserved-no event indicated) */
0x29, 0xa4,     /* USAGE_MAXIMUM    (Keyboard Application) */      /* Can be 255 */
0x81, 0x00,     /* INPUT           (Data,Ary,Abs) */
0xc0,           /* END_COLLECTION */
#ifdef MULTIMEDIA_KEYBOARD
0x05, 0x0c,     /* USAGE_PAGE       (Consumer) */
0x09, 0x01,     /* USAGE           (Consumer Control) */
0xa1, 0x01,     /* COLLECTION       (Application) */
0x85, 0x04,     /* REPORT_ID        (4) */
0x15, 0x00,     /* LOGICAL_MINIMUM  (0) */
0x25, 0x01,     /* LOGICAL_MAXIMUM  (1) */
0x75, 0x01,     /* REPORT_SIZE      (1) */
0x95, 0x18,     /* REPORT_COUNT     (24) */
0x09, 0xb5,     /* USAGE           (Scan Next Track) */
0x09, 0xb6,     /* USAGE           (Scan Previous Track) */
0x09, 0xb7,     /* USAGE           (Stop) */
0x09, 0xcd,     /* USAGE           (Play/Pause) */
0x09, 0xe2,     /* USAGE           (Mute) */
0x09, 0xe5,     /* USAGE           (Bass Boost) */
0x09, 0xe7,     /* USAGE           (Loudness) */
```

```
    0x09, 0xe9,     /* USAGE           (Volume Increment) */
    0x09, 0xea,     /* USAGE           (Volume Decrement) */
    0x0a, 0x52, 0x01,     /* USAGE      (Bass Increment) */
    0x0a, 0x53, 0x01,     /* USAGE      (Bass Decrement) */
    0x0a, 0x54, 0x01,     /* USAGE      (Treble Increment) */
    0x0a, 0x55, 0x01,     /* USAGE      (Treble Decrement) */
    0x0a, 0x83, 0x01,     /* USAGE      (AL Consumer Control Configuration) */
    0x0a, 0x8a, 0x01,     /* USAGE      (AL Email Reader) */
    0x0a, 0x92, 0x01,     /* USAGE      (AL Calculator) */
    0x0a, 0x94, 0x01,     /* USAGE      (AL Local Machine Browser) */
    0x0a, 0x21, 0x02,     /* USAGE      (AC Search) */
    0x0a, 0x23, 0x02,     /* USAGE      (AC Home) */
    0x0a, 0x24, 0x02,     /* USAGE      (AC Back) */
    0x0a, 0x25, 0x02,     /* USAGE      (AC Forward) */
    0x0a, 0x26, 0x02,     /* USAGE      (AC Stop) */
    0x0a, 0x27, 0x02,     /* USAGE      (AC Refresh) */
    0x0a, 0x2a, 0x02,     /* USAGE      (AC Bookmarks) */
    0x81, 0x02,     /* INPUT           (Data,Var,Abs) */
    0xc0           /* END_COLLECTION */
#endif
};

```

3.3 Scan

SUT needs to perform search operation before connected with Keyboard, then obtain module name and service UUID from advertising data and scan response data. Advertising data and scan response data format are defined below:

```
 /** @brief  GAP - scan response data (max size = 31 bytes) */
static const uint8_t scan_rsp_data[] =
{
    0x03,                           /* length */
    GAP_ADTYPE_APPEARANCE,           /* type="Appearance" */
    LO_WORD(GAP_GATT_APPEARANCE_KEYBOARD),
    HI_WORD(GAP_GATT_APPEARANCE_KEYBOARD),
};

/** @brief  GAP - Advertisement data (max size = 31 bytes, best kept short to conserve power) */
static const uint8_t adv_data[] =
{
```

```
    /* Flags */
    0x02,            /* length */
    GAP_ADTYPE_FLAGS, /* type="Flags" */
    GAP_ADTYPE_FLAGS_LIMITED | GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,
    /* Service */
    0x03,            /* length */
    GAP_ADTYPE_16BIT_COMPLETE,
    0x12,
    0x18,
    /* Local name */
    0x0D,            /* length */
    GAP_ADTYPE_LOCAL_NAME_COMPLETE,
    'B', 'L', 'E', '_', 'K', 'E', 'Y', 'B', 'O', 'A', 'R', 'D'
};
```

4 Reference

[1] IEEE Std 11073-20601 ™- 2008 Health Informatics - Personal Health Device Communication - Application Profile - Optimized Exchange Protocol - version 1.0 or later.